# PAQUETE DIDÁCTICO PARA LA ASIGNATURA DE CIBERNÉTICA Y COMPUTACIÓN II.

**UNIDAD 1**

Lenguaje de programación orientada a objetos con Java.

**UNIDAD 2**

Estructuras de control de secuencia en Java.

**UNIDAD 3**

Polimorfismo, constructores, colaboración y herencia de clases.

**UNIDAD 4**

Interfaz gráfica de usuario.

## AUTORES


Juan Gutiérrez Quiroz

*Coordinador*

María del Socorro Ávila Nicolás • Guillermina Luna Santiago

Gabriela López Vargas • José Luis Hermoso Sandoval

Alejandro Vela Bustamante.

# ÍNDICE

# UNIDAD 1

## Lenguaje de programación orientada a objetos con


- Conceptos Basicos
- Organización general programa
- Clases
- Metodos
- Objetos
- La Clase Scanner
- Errores sintácticos y lógicos
- Implementa una aplicación

# Unidad 1. Lenguaje de programación orientada a objetos con Java

## Propósito:

Al finalizar la unidad el alumno:

Conocerá las características del lenguaje de programación orientado a objetos Java y su entorno de desarrollo, definiendo clases, atributos y métodos para la implementación de objetos en programas.

## Aprendizajes:

El alumno:

- Conoce los conceptos básicos de la programación orientada a objetos.
- Conoce la organización general de un programa en Java como lenguaje orientado a objetos.
- Describe los conceptos de Clase y atributo del lenguaje Java.
- Implementa programas utilizando Clases y atributos.
- Describe los conceptos de métodos del lenguaje java.
- Conoce cómo instanciar objetos a partir de una Clase.
- Implementa programas utilizando métodos.
- Empleará la Clase Scanner para la entrada de datos en la creación de un programa
- Reafirma los conceptos adquiridos en la unidad.

## Introducción

La Programación Orientada a Objetos (POO) utiliza una serie de conceptos y técnicas de programación para representar acciones o cosas de la vida real basada en objetos, cuando relacionamos diferentes conceptos tales como clases, objetos, métodos, propiedades, estados, herencia, encapsulación entre otros, obtendremos un programa que será el conjunto de estos conceptos relacionados entre sí.

En la POO las clases son un elemento fundamental, de ellas se derivan los objetos y con ello nos permiten utilizar sus características como abstracción, encapsulamiento, herencia y polimorfismo.

En Java existen tres tipos de módulos: métodos, clases y paquetes; que están disponibles en la biblioteca de clases de Java con una vasta colección para realizar cálculos matemáticos, manipulación de cadenas, manipulación de caracteres, operaciones de entrada/salida y muchas otras operaciones útiles.

Al desarrollar una aplicación en un lenguaje de POO se crea un modelo que se asemeja a una situación de la vida real para resolver un problema. Para ello utilizamos el término objeto el cual presenta determinadas características y realiza ciertas acciones.

Por ejemplo, podemos pensar en un automóvil como un objeto que tiene las siguientes características: marca, modelo, color, número de placa, número de puertas, etc. Además de que puede realizar ciertas acciones como encenderse, apagarse, acelerar, frenar, girar, etc.

# Lenguaje de Programación Orientado a Objetos

**Propósito:**

Comprender los conceptos básicos de la Programación Orientada a Objetos.

## 1.1 Conceptos básicos de la Programación Orientada a Objetos.

La POO es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento. Su uso se popularizó a principios de la década de 1990.

Indica más una forma de diseño y una metodología de desarrollo de software que un lenguaje de programación, ya que en realidad se puede aplicar el Diseño Orientado a Objetos (OOD, por sus siglas en ingles), a cualquier tipo de lenguaje de programación.

Se utilizan variables básicas asociadas a los tipos primitivos de datos, lo que permite que objetos y variables se relacionen entre sí.

En este tipo de programación utilizamos varios conceptos que se explican a continuación.

**Un Objeto** es una unidad dentro de un programa de computadora, es una abstracción utilizada en programación que permite separar los diferentes componentes de un programa, simplificando así su elaboración, depuración y posteriores mejoras. Todo objeto tiene los siguientes elementos:

- **Identidad**.
  Es una propiedad de los objetos que los distinguen por su existencia propia y no por propiedades descriptivas que puedan tener, esto permite que se pueda diferenciar de los demás.

- **Atributos o características.**
  Son propiedades individuales que diferencian a un objeto de otro y determinan sus características. Los atributos se guardan en variables y cada objeto particular puede tener valores distintos para estas variables. Por ejemplo, nombre, edad o peso son atributos del objeto Persona.

- **Comportamiento.**
  Es la forma en la que un objeto puede realizar diferentes acciones. Los comportamientos se representan por medio de métodos, esto es, un conjunto de instrucciones que realizan una determinada tarea.

Los métodos (comportamiento) y atributos (estado) están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta.

**Ejemplo del comportamiento de un objeto:**

*Objeto: Cuenta Bancaria*

*Atributos:*

Para obtener los atributos del objeto, debemos tomar en cuentas las características individuales de la cuenta. Esto es, una cuenta bancaria se puede diferenciar por el tipo de cuenta, el nombre del titular y el saldo.

*Métodos.*

¿Qué se puede hacer con una cuenta bancaria? entre las acciones están Depositar y Extraer.

Entonces se tiene:

*Objeto:* Cuenta bancaria
*Atributos:* tipo, titular, saldo.
*Métodos:* Depositar, Extraer.

- **Abstracción.**

  Consiste en captar las características esenciales de un objeto, donde se capturan sus comportamientos. Es un método por el cual consideraremos por separado las cualidades y funciones que desempeña un objeto, estos son representados en clases por medio de atributos y métodos de dicha clase.

  El proceso de abstracción permite seleccionar las características relevantes dentro de un conjunto e identificar comportamientos comunes para definir nuevos tipos de entidades en el mundo real. La abstracción es clave en el proceso de análisis y diseño orientado a objetos, ya que mediante ella podemos llegar a armar un conjunto de clases que permitan modelar la realidad o el problema que se quiere atacar.

  **Ejemplos de abstracción:**

  *Ejemplo 1:* ¿Qué características podemos abstraer de los automóviles? o ¿Qué características semejantes tienen todos los automóviles?

  **Características:** Marca, Modelo, Número de chasis, llantas, Puertas, Ventanas...
  **Comportamiento:** Acelerar, Frenar, Retroceder...

*Ejemplo 2:* En una estética canina, necesitan llevar el control de los perros que atienden. En este caso:

**Las características** esenciales de la clase perro son: nombre, color, raza, altura, etc., y sus **comportamientos:** ladrar, dormir, comer, etc.

La abstracción es un elemento clave y esencial para diseñar un buen programa.

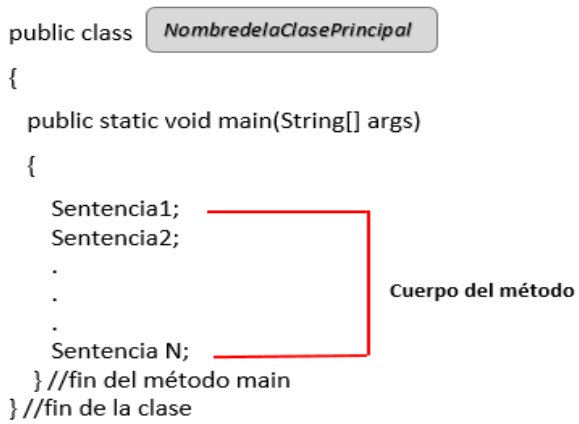
- **Encapsulamiento.**
  Consiste en unir en una Clase las características y comportamientos, esto es, las variables y métodos. También es la conjunción de todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema.

## 1.2 Organización general de un programa en Java

La estructura de un programa en Java va a contener varios elementos: comentarios, bibliotecas, palabras reservadas, sentencias, datos primitivos, bloques de código, entre otros.

A continuación, se explican los componentes que están presentes en la programación con Java.

La estructura de un programa simple en Java es la siguiente:




Donde *NombredelaclasePrincipal* es el nombre del programa o clase principal que contiene el código fuente que se almacena en la computadora con la extensión .java.

Entonces podemos decir que una clase es una la plantilla a partir de la cual se crean los objetos. También podemos definir a una clase en java como un conjunto de instrucciones que especifican la secuencia ordenada de acciones a realizar, para resolver un problema.

Todas las aplicaciones de Java tienen un método main que contiene un conjunto de instrucciones.

El método main es el punto de entrada de cualquier programa de Java. Su sintaxis es

public static void main(String[ ] args)

donde: las palabras **public y static** son atributos del método public esto significa que es un método público y que puede ser llamado por cualquier otro, static significa que es un método estático o de clase.

La palabra **void** indica que el método main no retorna ningún valor.

La forma (String[ ] args) es la definición de los argumentos que recibe el método main. En este caso los corchetes [ ] indican que el argumento es un arreglo y la palabra String es el tipo de los elementos del arreglo.

Los bloques de sentencias se indican entre llaves { }.

A continuación, se muestra un ejemplo de un programa muy sencillo en Java.

```
public class Hola
{
    public static void main(String[ ] args)
   {
        System.out.print("Hola, “);
        System.out.println(" Me llamo Juan ");
        System.out.println("Estoy programando en Java");
   }
}
```

En el ejemplo anterior, *Hola* es el nombre de la clase principal y del archivo que contiene el código fuente. Se puede distinguir la declaración del método main o principal que, a su vez, contiene un conjunto de sentencias que se indican entre llaves { }. Este se reduce a tres sentencias, que son llamadas a dos métodos predefinidos en Java (print y println) que permiten visualizar texto en la pantalla que es el dispositivo de salida de datos por defecto, esto se logra en conjunción con la sentencia: System.out.print o System.out.println, cuya función es mostrar datos en pantalla.

Por el momento y hasta que se explique con detalle el concepto de clase, los ejemplos de programa que se utilizarán constaran de una sola clase en la que se declara el método main. Este método es el punto de arranque de la ejecución de todo programa en Java.

- **Comentarios.**

  Un comentario en programación es una nota para el programador, es la línea o conjunto de líneas de texto en el código fuente que el compilador ignora.

  Cuando el compilador del lenguaje encuentra un comentario, esa línea, o varias de ellas, no las traduce, y continúa buscando en la instrucción siguiente. Los comentarios se utilizan para explicar y documentar el código fuente del programa.

  En Java hay tres tipos de comentarios:

  **a) Comentarios de una sola línea**

Las características de los comentarios de una línea son las siguientes:

- Comienzan con doble barra ( // )
- Pueden escribirse al principio de la línea o a continuación de una instrucción.
- No tienen carácter de terminación.

**Ejemplo**:

```
// Programa que calcula el área de un triángulo
```

**b) Comentario tradicional**

Las características de los comentarios tradicionales en Java son:

- Empieza con los caracteres /* y acaba con */.
- Pueden ocupar más de una línea y aparecer en cualquier lugar donde pueda aparecer un espacio en blanco.
- No pueden anidarse.

*Ejemplos:*

```
/* Programa para calcular el área de un círculo
   El radio debe ser mayor de cero */

/* Verifica que los valores sean mayores de cero */
```

*Ejemplo de comentario no válido:*

```
/* linea comentario 1
        /* linea comentario 2
          linea comemtario 3
        */
        linea comentario 4
        linea comentario 5
*/
```

Los comentarios no se pueden anidar.

**c) Comentarios de documentación JAVADOC**

Son comentarios especiales para generar documentación del programa.
Comienza con /** y termina con */
Ejemplo:

```
/**
 *
 * @author Programa realizado por Juan
 * @version 1.0
 *
 */
```

- **Uso de bibliotecas.**

  Una biblioteca es un conjunto de subprogramas utilizados para desarrollar software. Las bibliotecas contienen código y datos, que sirven a otros programas independientes, esto es, se convierten en parte del código de estos. Algunos programas ejecutables pueden ser a la vez programas independientes y bibliotecas, pero la mayoría de estas no son ejecutables. Los programas ejecutables y las bibliotecas, también conocidas como librerías, hacen referencias o llamados entre sí a través de un proceso conocido como enlace, que por lo general es realizado por un software denominado enlazador.

  Algunos ejemplos son:

  - *java.lang:* Se compone de declaraciones de objetos, clases, threads, excepciones, wrappers de los tipos de datos primitivos y otras clases fundamentales.
  - *java.io:* Librería estándar de entrada y salida.
  - *java.net:* Contiene clases que permiten a un programa comunicarse a través de redes (Internet o intranet), estás librerías apoyan interfaces con telnet y URLs.
  - *java.awt:* Abstract Windowing Toolkit proporciona una capa abstracta que permite llevar una aplicación en Java de un sistema de ventanas a otro. Contiene muchas clases e interfaces necesarias para trabajar con la interfaz de usuario gráfica clásica.
  - *java.util:* Contiene clases e interfaces de utilidades que sirven para operaciones con la fecha y la hora, generación de números aleatorios.

  Por ejemplo, la clase Scanner, sirve para leer datos desde el teclado y se encuentra en el paquete java.util por lo tanto se debe incluir al inicio del programa la instrucción:

  *import java.util.Scanner;*

- **Identificadores.**

  Son un conjunto de caracteres alfanuméricos de cualquier longitud que sirve para nombrar a las variables a utilizar en un programa. Los identificadores o nombres de variables pueden ser combinaciones de letras y números y seguirán las reglas que dicte el lenguaje de programación que se esté utilizando.

  Características de un identificador en Java:

  - Los caracteres permitidos para los identificadores son todos los caracteres alfanuméricos ([AZ], [az], [0-9]), "$" (signo de dólar) y '_' (guion bajo).
  - Pueden comenzar con una letra, el carácter subrayado (_) o el carácter dólar ($).
  - Puede incluir, pero no comenzar por un número.
  - No puede incluir el carácter espacio en blanco.
  - No pueden incluir el guion medio (-).

Ejemplos de identificadores válidos:

| Edad | nombre | _Precio | Año | año_nacimiento |
|---|---|---|---|---|
| AÑO0 | $cantidad | _$cantidad | cantidad_10_1 | PrecioVenta |
| num4 | bl4nc0 | miércoles | PrIvAdo | Máximo |

***Tabla 1.2.1 Identificadores***

Ejemplos de identificadores NO válidos:

- 4num: Identificador no válido porque comienza por un dígito.
- z#: No válido porque contiene el carácter especial #.
- "Edad": No válido porque no puede contener comillas.
- Tom's: No válido porque contiene el carácter (').
- año-nacimiento: no válido porque contiene el carácter -.
- public: no válido porque es una palabra reservada del lenguaje.
- __precio:final: no válido porque contiene el carácter :.

- **Palabras reservadas.**
  Son identificadores predefinidos que tienen un significado especial, generalmente corresponden a nombres de instrucciones del lenguaje de programación y no se pueden utilizar como variables en un programa, por ejemplo, el identificador *If*, en la mayoría de los lenguajes de programación es una **palabra reservada** que corresponde a una instrucción condicional y por lo tanto no puede ser utilizada como nombre de variable.

  A continuación, se muestran las palabras reservadas de Java.

| **PALABRAS RESERVADAS** | | | | |
|---|---|---|---|---|
| **abstract** | **continue** | **for** | **new** | **switch** |
| **assert** | **default** | **goto** | **package** | **synchronized** |
| **boolean** | **do** | **if** | **private** | **this** |
| **break** | **double** | **implements** | **protected** | **throw** |

| PALABRAS RESERVADAS | | | | |
|---|---|---|---|---|
| **byte** | **else** | **import** | **public** | **Throws** |
| **case** | **enum** | **instanceof** | **return** | **Transient** |
| **catch** | **extends** | **int** | **short** | **Try** |
| **char** | **final** | **interface** | **static** | **Void** |
| **class** | **finally** | **long** | **strictfp** | **Volatile** |
| **const** | **float** | **native** | **super** | **while** |

***Tabla 1.2.2 Palabras reservadas***

- **Sentencias.**
  Son los elementos básicos en los que se divide el código de un programa realizado en un lenguaje de programación. Es una orden que se le da al programa para realizar una tarea específica, por ejemplo: mostrar un mensaje en la pantalla, declarar una variable, inicializarla, llamar a una función, etcétera. Terminan con *punto y coma* (;) esto sirve para separar una sentencia de la siguiente. Normalmente se ponen unas debajo de otras, aunque también podemos colocar sentencias cortas en una misma línea.

  **Ejemplos:**

  *Sentencias de Expresión:*

```
valorA = 8933.234; // asignación
valorA++;                  // incremento
System.out.println(valorA);  // llamada a un método
Integer objetoInt = new Integer(4); // creación de objetos
```

  *Las sentencias de declaración de variables:*

```
int  bValue;
double  aValue = 8933.234;
String varCad;
```

- **Tipos de datos primitivos.**

  Los tipos de datos primitivos son conjuntos de valores simples procesados por operaciones definidas en el lenguaje de programación. Los tipos de datos más comunes en los lenguajes de programación son los números enteros, los números reales, los caracteres, los valores lógicos y los apuntadores.

| Tipo | Tamaño en bits | Valores |
|---|---|---|
| boolean | | true o false |
| char | 16 | 0 a 65535 |
| byte | 8 | -128 a +127 |
| short | 16 | -32,768 a + 32,767 |
| int | 32 | -2,147,483,648 a +2,147,483,647 |
| long | 64 | -9,223,3723036,854,775,808, a +9,223,372,036,854,775,807 |
| float | 32 | Rango negativo<br>-3.4028234663852886E+38 a -1.40129846432481707e-45<br>Rango positivo<br>1.40129846432481707e-45 a<br>3.4028234663852886E+38 a |
| double | 64 | Rango negativo<br>-1.7976931348623157E+308 a -4.94065645841246544e-324<br>Rango positivo<br>4.94065645841246544e-324 a 1.7976931348623157E+308 |

***Tabla 1.2.3**. **Tipos primitivos de Java.***

- **Bloque de código.**

  Es un conjunto de instrucciones con una o más declaraciones y sentencias en donde el inicio es indicado por una llave "{" y el final del bloque por una llave de cierre "}". Los bloques de código son el principal mecanismo de encapsulamiento y se forman con un grupo de sentencias y también por otros bloques de código.

  Ejemplos de bloques de código: la definición de una clase, la definición de un método, etcétera.

  Un ejemplo de un bloque sencillo de código es:

```
{
    saludo=" Hola mundo";
   System.out.println(saludo);
}
```

- **Operadores.**

Llevan a cabo operaciones sobre datos u operandos de tipo primitivo obteniendo como resultado un valor determinado también de un tipo primitivo. El tipo de valor devuelto tras la evaluación depende del operador y del tipo de los operandos. Por ejemplo, los operadores aritméticos trabajan con operandos numéricos, llevan a cabo operaciones aritméticas básicas y devuelven el valor numérico correspondiente. Los operadores se pueden clasificar en distintos grupos según se muestra en los siguientes apartados.

**OPERADOR ASIGNACIÓN**

El operador asignación =, asigna el valor del término de la derecha al operando de la izquierda. El operando generalmente es el identificador de una variable. El término de la derecha es, en general, una expresión de un tipo de dato compatible; en particular puede ser una constante u otra variable. Este operador no se evalúa, el operando contiene el valor asignado.

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| = | Operador asignación | n = 4 | n vale 4 |

***Tabla 1.2.4***. ***Operador asignación***

**OPERADORES ARITMÉTICOS**

Java tiene varios operadores aritméticos para los datos numéricos enteros y reales. En la Tabla 1.2.5, se resumen los diferentes operadores de esta categoría.

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| - | operador unario de cambio de signo | - 4 | - 4 |
| + | Suma | 2.5 + 7.1 | 9.6 |
| - | Resta | 235.6 - 103.5 | 132.1 |
| * | Producto | 1.2 * 1.1 | 1.32 |
| / | División (tanto entera como real) | 0.05 / 0.2<br>7 / 2 | 0.25<br>3 |
| % | Resto de la división entera | 20 % 7 | 6 |

***Tabla 1.2.5***. ***Operadores aritméticos básicos.***

El resultado exacto de las expresiones donde utilizamos operadores aritméticos depende de los tipos de operando involucrados. Es conveniente considerar los siguientes casos:

- El resultado es de tipo *long* si, al menos, uno de los operandos es de tipo *long* y ninguno es *rea*l (*float o double*).
- El resultado es de tipo *int* si ninguno de los operandos es de tipo long y tampoco es real (float o double).
- El resultado es de tipo *double* si, al menos, uno de los operandos es de tipo *double*.
- El resultado es de tipo *float* si, al menos, uno de los operandos es de tipo *float* y ninguno es *double*.
- En la aritmética entera no se producen nunca desbordamientos (overflow) aunque el resultado sobrepase el intervalo de representación (int o long).
- La división entera se trunca hacia 0. La división o el resto de dividir por cero es una operación válida que genera una excepción ArithmeticException que puede dar lugar a un error de ejecución y la consiguiente interrupción de la ejecución del programa.
- La aritmética real (en coma flotante) puede desbordar al infinito (demasiado grande, overflow) o hacia cero (demasiado pequeño, underflow).
- El resultado de una expresión inválida, por ejemplo, dividir infinito por infinito, no genera una excepción ni un error de ejecución: es un valor NaN (Not a Number).

**OPERADORES ARITMÉTICOS INCREMENTALES**

Requieren de un solo operador. El operando puede ser numérico o de tipo *char* y el resultado es del mismo tipo. Pueden emplearse de dos formas dependiendo de su posición con respecto al operando.

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| ++ | Incremento. | 4++ | 5 |
| | i++ primero se utiliza la variable y luego se incrementa su valor. | a=5;<br>b=a++; | a vale 6 y<br>b vale 5 |
| | ++i primero se incrementa el valor de la variable y luego se utiliza. | a=5;<br>b=++a; | a vale 6 y<br>b vale 6 |

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| -- | Decremento. | 4-- | 3 |

***Tabla 1.2.6. Operadores aritméticos incrementales.***

**OPERADORES ARITMÉTICOS COMBINADOS**

Combinan un operador aritmético con el operador asignación. Pueden tener operandos numéricos enteros o reales y el tipo específico del resultado numérico dependerá del tipo de éstos. Esto se puede visualizar a continuación:

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| += | Suma combinada | a+=b | a=a+b |
| -= | Resta combinada | a-=b | a=a-b |
| *= | Producto combinado | a*=b | a=a*b |
| /= | División combinada | a/=b | a=a/b |
| %= | Resto combinado | a%=b | a=a%b |

***Tabla 1.2.7. Operadores aritméticos combinados.***

**OPERADORES DE RELACIÓN**

Se utilizan para realizar comparaciones entre datos compatibles de tipos primitivos (numéricos, carácter y booleanos) teniendo siempre un resultado booleano (*true* o *false*). Los operandos booleanos sólo pueden emplear operadores de igualdad y desigualdad.

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| == | igual que | 7==8 | false |
| != | distinto que | ‘a’ != ‘k’ | true |
| < | menor que | ‘G’ < ‘B’ | false |

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| > | mayor que | 'b' > 'a' | true |
| <= | menor o igual que | 7.5 <= 7.38 | false |
| >= | mayor o igual que | 38 >= 7 | true |

***Tabla 1.2.8. Operadores de relación.***

**OPERADORES LÓGICOS O BOOLEANOS**

Se utilizan para realizar operaciones sobre datos booleanos y tienen como resultado un valor booleano. A continuación, se muestran los operadores de este tipo.

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| ! | Negación - NOT (unario) | !false<br>!(5==5) | true<br>false |
| \| | Suma lógica - OR (binario) | true \| false<br>(5==5) \| (5<4) | true<br>true |
| ^ | Suma lógica exclusiva - XOR | true ^ false<br>(5==5) ^ (5<4) | true<br>true |
| & | Producto lógico - AND (binario) | true & false<br>(5==5) & (5<4) | false<br>false |
| \|\| | Suma lógica con cortocircuito: si el primer operador es true entonces el segundo se salta y el resultado es true | true \|\| false<br>(5==5) \|\| (5<4) | true<br>true |
| && | Producto lógico con cortocircuito: si el primer operando es false entonces el segundo se salta y el resultado es false | false && true<br>(5==5) &&(5<4) | false<br>false |

***Tabla 1.2.9. Operadores booleanos.***

**EL OPERADOR CONDICIONAL**

Este regresa valores en función de una expresión lógica.

Sintaxis:

*expresionLogica ? expresion_1 : expresion_2*

Si el resultado de evaluar la expresión lógica es verdadero, devuelve el valor de la primera expresión, y en caso contrario, devuelve el valor de la segunda expresión.

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| ?: | Operador condicional | a = 4;<br>b = a == 4 ? a+5 : 6-a;<br>b = a > 4 ? a*7 : a+8; | <br>b vale 9<br>b vale 12 |

***Tabla 1.2.10. Operador condicional***

**OPERADOR CONCATENACIÓN DE CADENAS**

Devuelve una cadena como resultado de concatenar dos cadenas que actúan como operandos. Si sólo uno de los operandos es de tipo cadena, el otro operando se convierte implícitamente en tipo cadena.

| Operador | Descripción | Ejemplo de expresión | Resultado del ejemplo |
|---|---|---|---|
| + | Operador concatenación | “Hola” + “Juan” | “HolaJuan” |

***Tabla 1.2.11. Operadores de concatenación***

**SEPARADORES**

Existen algunos caracteres que tienen un significado especial en el lenguaje Java. En la siguiente tabla se resumen los diferentes separadores que pueden encontrarse en el código fuente de un programa.

| Separador | Descripción |
|---|---|
| ( ) | Permiten modificar la prioridad de una expresión, contener expresiones para el control de flujo y realizar conversiones de tipo. Por otro lado, pueden contener la lista de parámetros o argumentos, tanto en la definición de un método como en la llamada al mismo. |
| { } | Permiten definir bloques de código y ámbitos y contener los valores iniciales de las variables array. |
| [ ] | Permiten declarar variables de tipo array (vectores o matrices) y |

| Separador | Descripción |
|---|---|
| | referenciar sus elementos. |
| ; | Permite separar sentencias. |
| , | Permite separar identificadores consecutivos en la declaración de variables y en las listas de parámetros. También se emplea para encadenar sentencias dentro de un bucle for. |
| . | Permite separar el nombre de un atributo o método de su instancia de referencia. También separa el identificador de un paquete de los subpaquetes y clases. |

***Tabla 1.2.12. Separadores.***

**PRIORIDAD ENTRE OPERADORES**

Si dos operadores se encuentran en la misma expresión, el orden en el que se evalúan puede determinar el valor de la expresión. En la siguiente tabla se muestra el orden o prioridad en el que se ejecutan los operadores que se encuentren en la misma sentencia. Los operadores de la misma prioridad se evalúan de izquierda a derecha dentro de la expresión.

| Prioridad | Operador | Tipo de operador | Operación |
|---|---|---|---|
| 1 | ++<br>--<br>+, -<br>! | Aritmético<br>Aritmético<br>Aritmético<br>Booleano | Incremento previo o posterior (unario)<br>Decremento previo o posterior (unario)<br>Suma unaria, resta unaria<br>Negación (unaria) |
| 2 | (tipo) | Cualquiera | |
| 3 | *, /, % | Aritmético | Multiplicación, división, resto |
| 4 | +, -<br>+ | Aritmético<br>Cadena | Suma, resta<br>Concatenación de cadenas |
| 5 | <, <=<br>>, >=<br>instanceof | Aritmético<br>Aritmético<br>Objeto, tipo | Menor que, menor o igual que<br>Mayor que, mayor o igual que<br>Comparación de tipos |
| 6 | == | Primitivo | Igual (valores idénticos) |

| Prioridad | Operador | Tipo de operador | Operación |
|---|---|---|---|
| | != | Primitivo | Desigual (valores diferentes) |
| | == | Objeto | Igual (referencia al mismo objeto) |
| | != | Objeto | Desigual (referencia a distintos objetos) |
| 7 | & | Booleano | Producto booleano |
| 8 | ^ | Booleano | Suma exclusiva booleana |
| 9 | \| | Booleano | Suma booleana |
| 10 | && | Booleano | AND condicional |
| 11 | \|\| | Booleano | OR condicional |
| 12 | ? : | Booleano, cualquiera, | Operador condicional (ternario) |
| 13 | =<br>*=, /=, %=<br>+=, -=<br>&=, ^=, \|= | Variable, cualquiera | Asignación, asignación con operación |

***Tabla 1.2.13. Prioridad de los operadores.***

- **Expresiones.**
  Son una combinación de variables, operadores y llamadas de métodos construidas de acuerdo con la sintaxis del lenguaje y devuelven un valor.

  Se utilizan para calcular y asignar valores a las variables y para controlar el flujo de un programa Java. El trabajo de una expresión se divide en dos partes: realizar los cálculos indicados por los elementos de la expresión y devolver algún valor. El tipo de dato del valor regresado por una expresión depende de los elementos usados en la expresión.

  Una expresión es todo aquello que se puede poner a la derecha del operador asignación =.

  Por ejemplo:

| | |
|---|---|
| Asignación | a = b |
| Suma y asignación | a += b (a=a + b) |

| | |
|---|---|
| Resta y asignación | a -= b (a=a - b) |
| Multiplicación y asignación | a *= b (a=a * b) |
| División y asignación | a /= b (a=a / b) |
| Módulo y asignación | a %= b (a=a % b) |

***Tabla 1.2.14. Expresiones.***

Actividad 1.1

**Selecciona la opción correcta de la descripción que corresponde al concepto:**

| | **Concepto** | | | **Descripción** |
|---|---|---|---|---|
| 1 | Abstracción | ( ) | a | Conjunto de instrucciones que especifican la secuencia ordenada de acciones a realizar, para resolver un problema. |
| 2 | Comportamiento | ( ) | b | Es una unidad dentro de un programa de computadora, es una abstracción utilizada en programación para separar los diferentes componentes de un programa. |
| 3 | Atributos | ( ) | c | Serie de conceptos y técnicas de programación para representar acciones o cosas de la vida real basada en objetos. |
| 4 | Clase | ( ) | d | Es una propiedad de los objetos que los distinguen por su existencia propia. |
| 5 | Características | ( ) | e | Son las variables de un objeto. Por ejemplo, nombre, edad o peso. |
| 6 | Encapsulamiento | ( ) | f | Nombre, color, raza, altura, etc, son_____ de un objeto. |
| 7 | Método main | ( ) | g | Es la forma en la que un objeto puede realizar diferentes acciones. |

| | Concepto | | | Descripción |
|---|---|---|---|---|
| 8 | Identidad | ( ) | h | Es parte de todas las aplicaciones Java, a su vez, está formado por un conjunto de instrucciones. |
| 9 | Objeto | ( ) | i | Es un método por el cual consideraremos por separado las cualidades y funciones que desempeña un objeto. |
| 10 | Programación Orientada a Objetos | ( ) | j | Es la conjunción de todos los elementos que pueden considerarse pertenecientes a una misma entidad. |

Actividad 1.2

***CUESTIONARIO***

1. Plantilla a partir de la cual se crean los objetos:

a) Clase b) Método c) Bloque de código d) Librería

2. Palabra que indica que el método main no retorna ningún valor:

a) public b) static c) void d) String[ ] args

3. Palabra que indica que el método puede ser llamado por cualquier otro:

a) public b) static c) void d) String[ ] args

4. Es la definición de los argumentos que recibe el método main:

a) public b) static c) void d) String[ ] args

5. En programación es una nota para el programador:

a) { } b) Comentario c) Bloque de código d) Sentencia

6. Conjunto de subprogramas utilizados para desarrollar programas:

a) Bloque de código b) Bibliotecas c) Sentencias d) Operadores

7. Conjunto de caracteres alfanuméricos que sirve para nombrar a las variables a utilizar en un programa:

a) Identificadores b) Sentencias c) Operadores d) Librerías

8. Son identificadores predefinidos que tienen un significado especial, generalmente corresponden a nombres de instrucciones:

a) Sentencias b) Librerías c) Bloque de código d) Palabras Reservadas

9. Es un identificador inválido:

a) "nombre" b) Nombre c) $nombre d) nom_bre

10. Es un identificador válido:

a) "variable" b) vari-able c) variable# d) vari_able

11. Son los elementos básicos en los que se divide el código de un programa realizado en un lenguaje de programación:

a) Sentencias b) Librerías c) Bloque de código d) Palabras Reservadas

12. Son conjuntos de valores simples procesados por operaciones definidas en el lenguaje de programación:

a) Identificadores b) Tipos primitivos c) Operadores d) Expresiones

13. Es un conjunto de instrucciones con una o más declaraciones y sentencias:

a) Sentencias b) Librerías c) Bloque de código d) Expresiones

14. Lleva a cabo operaciones sobre datos u operandos de tipo primitivo:

a) Operador b) Expresión c) Bloque de código d) Separador

15. Es una combinación de variables, operadores y de acuerdo con la sintaxis del lenguaje devuelve un valor:

a) Operador b) Expresión c) Bloque de código d) Separador

## 1.3 Clases

## Propósito.

Comprende el concepto de Clases y atributo.

- ## Definición

Una clase representa un conjunto de objetos que comparten una estructura y comportamientos comunes, esto es, representa la generalización de un objeto; siendo una plantilla que define las características (propiedades - atributos) y los comportamientos (métodos) que son comunes para todos los objetos.

**Ejemplo:**

Observa la siguiente imagen.

Clase - Persona

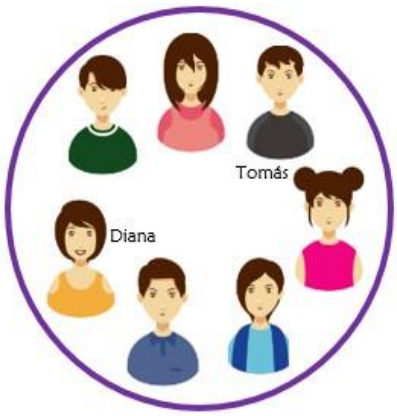



***Figura 1.3.1. Ejemplo de clase***

Tomás es una persona que tiene un nombre, edad y altura. Diana también, tiene un nombre, edad y altura, al igual que los otros miembros del conjunto. Ellos cuentan con esas mismas cualidades, por lo que se aprecia que pertenecen a un mismo conjunto, el cual denominamos “persona”. Entonces la **clase es Persona**.

- ## Declaración

Para declarar una clase se utiliza la palabra reservada **class**, seguida del nombre de la clase este debe ser un identificador válido en java.

La sintaxis **mínima** de una clase es:

class *NombreClase***{**

**bloque de Código**

**}**

Los nombres de las clases empiezan con una letra mayúscula, si el nombre se compone por varias palabras se escribe de manera seguida, sin espacios, y se recomienda colocar la primera letra mayúscula de cada palabra.

**Ejemplo:**

Definiendo la clase Persona, nos queda:

class ***Persona {***

**bloque de Código**

**}**

**Otros ejemplos de clases son:**

```
class Clientes{
    bloque de Código
}
```

```
class Vehiculos{
    bloque de Código
}
```

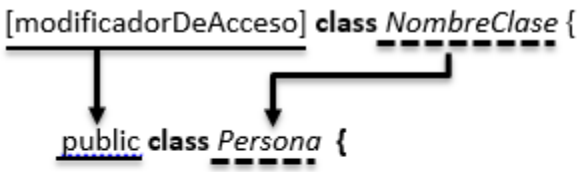
```
class Animales{
    bloque de Código
}
```

Asimismo, a la sintaxis se le puede agregar un modificador de acceso (alcance de acceso a la clase en el programa), quedando:

```
[modificadorDeAcceso] class NombreClase {

    // atributos de la clase
       [modificadorDeAcceso]  tipo nombreAtributo;


    // métodos de la clase

}
```

Nota. Lo que aparece entre corchetes es *opcional*

**Ejemplo:**




**bloque de Código**

**}**

Siendo

| | |
|---|---|
| [modificadorDeAcceso] | *public* |
| *NombreClase* | *Persona* |

- **Atributos**
  - ✓ **Definición**

Los atributos constituyen la estructura interna de los objetos de una clase. Son las características o propiedades de un objeto, las cuales están representadas por variables. Además, es posible asignar un valor de inicio a cada atributo de una clase de ser necesario.

**Ejemplo:**

Siguiendo con el ejemplo de **clase Persona**, se considera que todas las personas pueden tener un ***nombre, edad y altura,*** estas cualidades que son relevantes para la clase son los atributos.

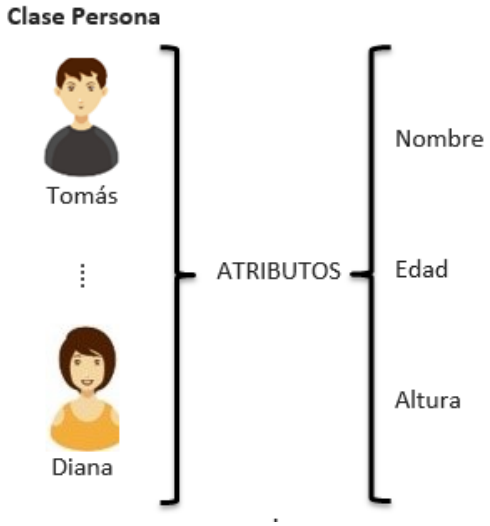



***Figura 1.3.2. Ejemplo de atributos***

  - ✓ **Declaración**

Para declarar un atributo en Java se utiliza la siguiente sintaxis:

*tipodeDato nombreAtributo;*

Donde:

*tipodeDato* corresponde al tipo de variable que se espera.

*nombreAtributo* es el nombre que daremos a la variable, siendo un nombre válido. Los nombres de las variables empiezan con una letra minúscula (los nombres de las clases empiezan con una letra mayúscula).

**Ejemplos:**

*String nombre;*

*int edad;*

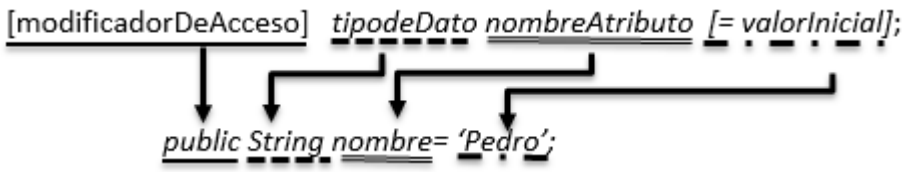
*float altura;*

La sintaxis general para declarar un atributo es:

[modificadorDeAcceso] *tipodeDato nombreAtributo [= valorInicial]*;

Nota: Lo que aparece entre corchetes es *opcional*.

**Ejemplo:**




Otros ejemplos, considerando que lo que aparece en corchetes es opcional, se puede omitir el modificador de acceso, como se muestra:

*int edad=17;*

*float altura=1.64f;*

### ✓ Niveles de visibilidad.

Al declarar una clase se pueden establecer distintos niveles de visibilidad también conocidos como modificadores de acceso, para los atributos y operaciones en función desde dónde se quiere tener acceso a ellos.

En la siguiente tabla 1.3.1 se describe el nivel de acceso (Nivel), el alcance (Significado), la sintaxis y los elementos a los que se les puede aplicar.

| Nivel | Significado | Sintaxis en Java | Aplicable a: |
|---|---|---|---|
| Pública | Se puede tener acceso al miembro de la clase desde cualquier parte del programa. | public | Clases<br>Atributos<br>Métodos |
| Protegida | Sólo se puede tener acceso al miembro de la clase desde la propia clase o de una clase que herede de ella. | protected | Atributos<br>Métodos |
| Privada | Sólo se puede tener acceso al miembro de la clase desde la propia clase. | private | Atributos<br>Métodos |

| Nivel | Significado | Sintaxis en Java | Aplicable a: |
|---|---|---|---|
| <Sin modificador> | Cuando no se especifica un modificador puede ser usado por las clases dentro de su misma ubicación (carpeta - paquete) donde se define la clase. | | Clases<br>Atributos<br>Métodos |

***Tabla 1.3.1. Niveles de visibilidad.***

Asimismo, se hace uso del modificador ***static*** para definir a los atributos de una clase y estos sólo se pueden ocupar por los métodos de la misma clase.

Un Atributo static:

- No es específico de cada objeto. Solo hay una copia del mismo y su valor es compartido por todos los objetos de la clase.
- Existe y puede utilizarse, aunque no existan objetos de la clase.

Para acceder a un atributo de clase se escribe:

***NombreClase.atributo***

- **Implementación**

Para implementar un programa, se debe identificar la clase y sus atributos.

Por el momento, seguiremos trabajando con nuestro ejemplo de clase Persona. Construir un programa donde declare la clase Persona con sus atributos: nombre, apellido paterno, apellido materno, edad, altura y género (F- femenino, M - masculino)**.** Obteniendo como salida en pantalla la información de la persona.

1. Se define la clase Persona

```
public class Persona
{

}
```

2. Se determinan los atributos de la clase

```
    String nombre= "Pedro";
    String apPat= “Hernández”;
    String apMat= “López”;
    int edad=17;
    float altura=1.64f;
    char genero=”M”;
```

3. Se crea el método main

```
public static void main(String[ ] args){

}
```

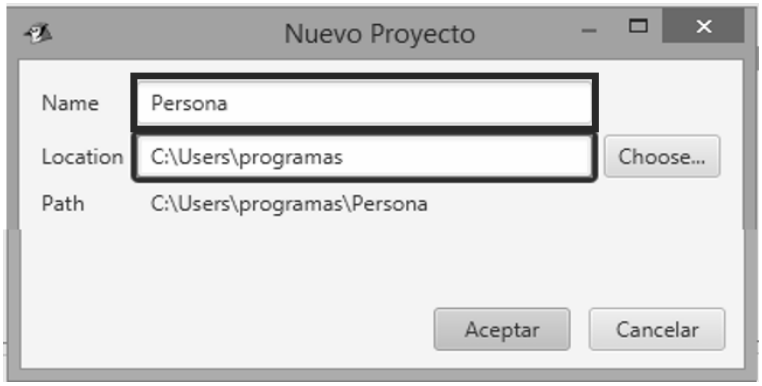
4. Se utiliza la sentencia System.out para ver datos en pantalla

```
System.out.println("Información de la persona");
System.out.println("Nombre completo: " + nombre + " "+ apPat + " " + apMat);
System.out.println("Edad: " + edad + "años");
System.out.println("Altura: " + altura);
System.out.println("Género: " + genero);
```

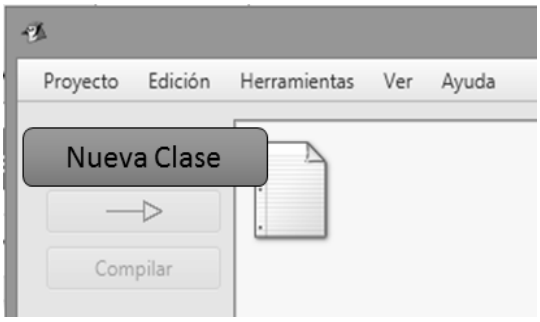
5. Se edita en BlueJ

```
// Se declara la clase Persona
public class Persona
{
/* Declaración de los atributos de la clase.  Se agrega el método de acceso static para poderlos ocupar en el método principal */
    static String nombre= "Pedro";
    static String apPat= "Hernández";
    static String apMat= "López";
    static int edad=17;
    static float altura=1.64f;
    static char genero='M';
 // Método principal
  public static void main(String[ ] args){
    System.out.println("Información de la persona");
    System.out.println("Nombre completo : " + nombre + " "+ apPat + " " + apMat);
    System.out.println("Edad: " + edad + "años");
    System.out.println("Altura : " + altura);
    System.out.println("Género: " + genero);
  }
}
```

Se abre el editor de BlueJ. Y seleccionamos en el menú Proyecto y Nuevo Proyecto y aparecerá la siguiente pantalla, en donde ponemos el Nombre del proyecto y la localización. Presionar aceptar.



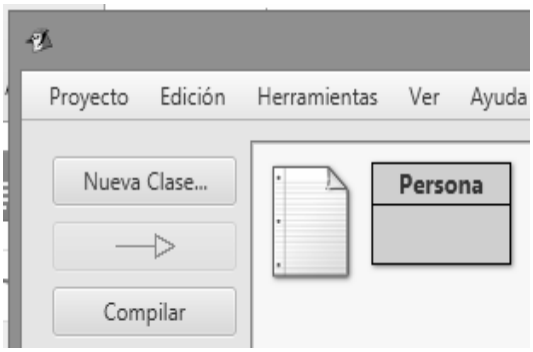

Presionar aceptar aparece la siguiente ventana



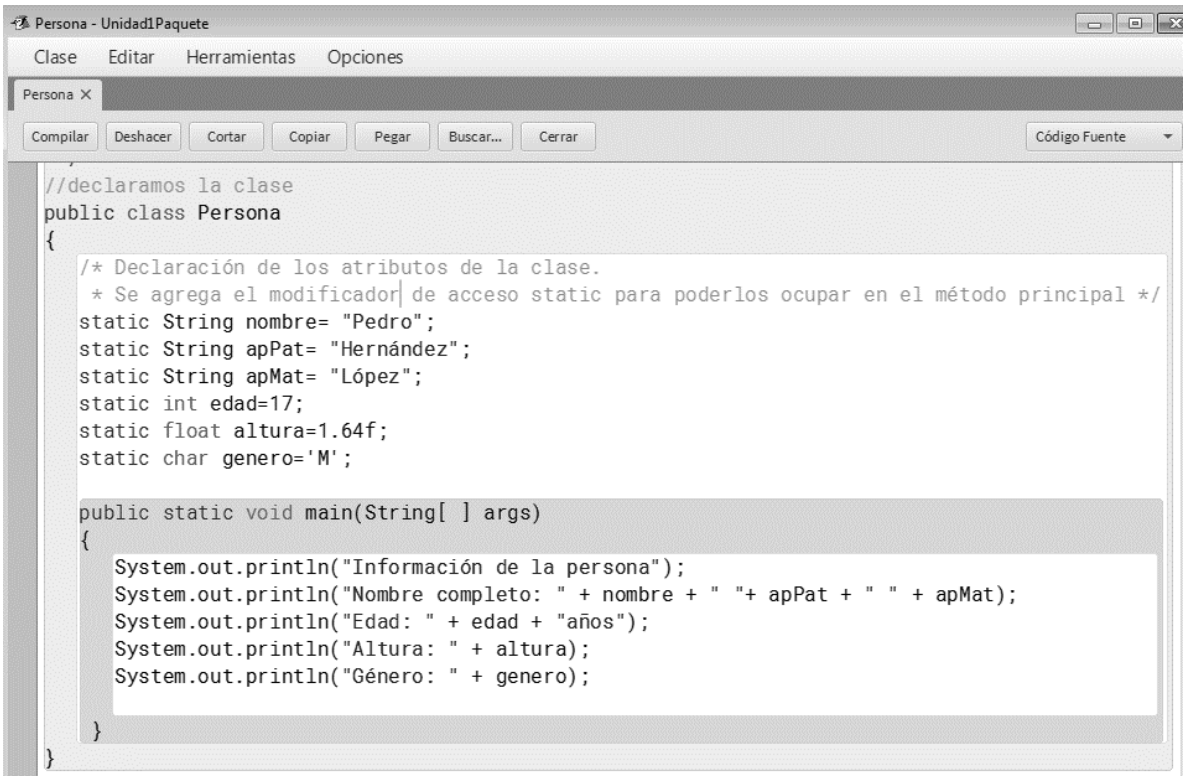

Dar clic a Nueva clase, en la siguiente ventana escribe en Nombre de Clase "Persona".


A continuación, se muestra la pantalla de vista gráfica de clase.

A continuación, se muestra la pantalla de vista gráfica de clase.



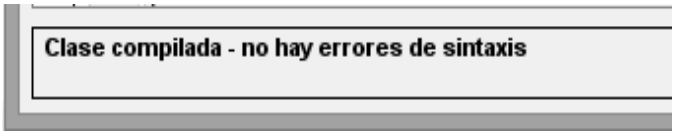

Da clic en Persona y a continuación se presenta la siguiente ventana en donde se escribe el siguiente código.



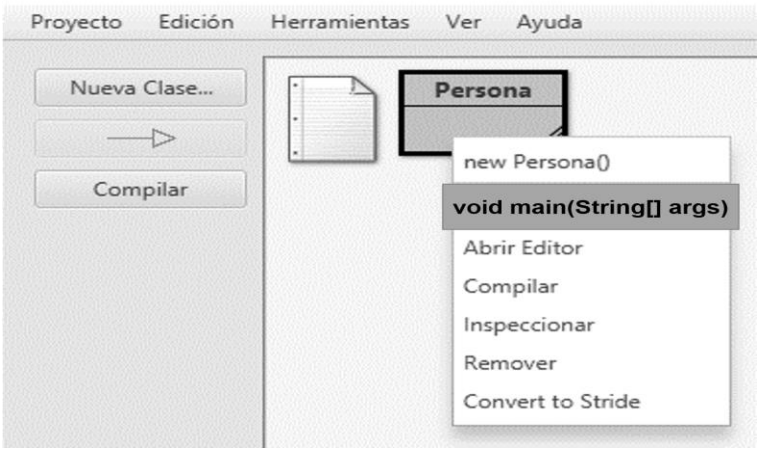

```
//declaramos la clase
public class Persona
{
   /* Declaración de los atributos de la clase.
    * Se agrega el modificador de acceso static para poderlos ocupar en el método principal */
   static String nombre= "Pedro";
   static String apPat= "Hernández";
   static String apMat= "López";
   static int edad=17;
   static float altura=1.64f;
   static char genero='M';

   public static void main(String[ ] args)
   {
      System.out.println("Información de la persona");
      System.out.println("Nombre completo: " + nombre + " "+ apPat + " " + apMat);
      System.out.println("Edad: " + edad + "años");
      System.out.println("Altura: " + altura);
      System.out.println("Género: " + genero);

    }
}
```

6. Al finalizar la captura del código selecciona compilar.

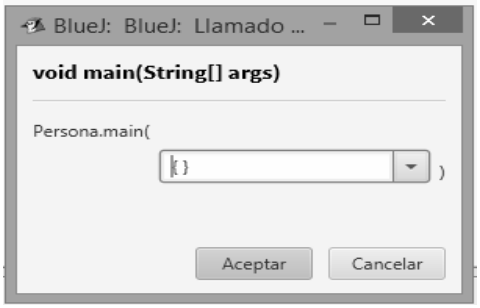
Al terminar la compilación aparecerá el siguiente mensaje.



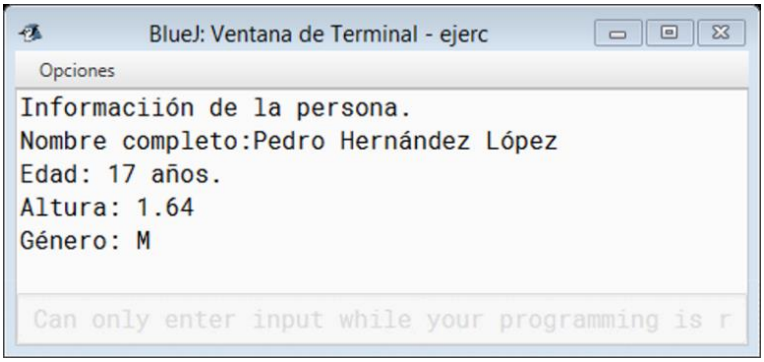

7. Para ejecutar el programa da clic derecho sobre la clase Persona y aparecerá la siguiente pantalla. Selecciona void main.




Aparecerá la siguiente ventana. Da clic en aceptar.




8. A continuación se muestra el resultado.



Actividad 1.3

1. Define una clase libro con los siguientes atributos: nombre, autor, editorial y número de páginas.

2. Define la clase Vehículo con los atributos marca, modelo, color y año.

Actividad 1.4

**Cuestionario**

1. Es una plantilla que define las características y los comportamientos que son comunes para todos los objetos.

   a) Clase b) Atributo c) Objeto d) Método

2. Son las características o propiedades de un objeto.

   a) Clase b) Atributo c) Objeto d) Método

3. Son ejemplos de una clase.

   a) Persona, Animales, Autos b) nombre, apellido, edad c) Persona, nombre, Animal d) nombre, apellido, Persona

4. Son ejemplos de atributos.

   a) Persona, Animales, Autos b) nombre, apellido, edad c) Persona, nombre, Animal d) nombre, apellido, Persona

5. ¿Cuál de los siguientes códigos en Java es el correcto?

a)
```
public class Cliente
{
    int ide;
    String Empresa;
    String Dirección;
    int CP;
 }
```

b)
```
public class Cliente
   int ide;
   String Empresa;
   String Dirección;
   int CP;
```

c)
```
class Cliente;
{
   int ide,
   int CP;
   String Empresa.
   String Dirección;
}
```

d)
```
public class cliente
{
   int ide,
   String Empresa,
   String Dirección,
   int CP;
 }
```

## 1.4 Métodos

**Propósito**:

Describir los conceptos de métodos del lenguaje Java para aprender a declararlos y llamarlos en la implementación de aplicaciones y así utilizar sus ventajas.

- ## Definición

Los métodos son conocidos como comportamientos o funciones que permiten dividir un programa en módulos, por medio de la separación de tareas o acciones de la clase. Un comportamiento es algo que un objeto puede realizar y que se define en la clase, para que el objeto lo pueda realizar; por ejemplo: si se tiene la clase ***Lavadora*** podemos decir que sus atributos pueden ser marca, modelo, número de serie, capacidad; y sus comportamientos pueden ser: remojar, lavar, enjuagar, activar, apagar, centrifugar.


*Figura 1.4.1. Funciones de una lavadora.*

En cuanto al lenguaje de programación Java, éste soporta dos tipos de métodos:

- **Métodos de clase**: son invocados en una clase y realizan una tarea que no depende del contenido de ningún objeto y se conocen como *static*. Un ejemplo es el método *main,* el cual es el punto de inicio de toda aplicación en Java y debe definirse como se muestra a continuación ya que, de no declararse, no se ejecutará la aplicación.

```
public static void main(String[ ] args)
{
        //bloque de código
}
```

Otro ejemplo, es la clase ***Math*** que tiene varios métodos para diferentes tareas como calcular el valor absoluto, redondear un entero, calcular el coseno trigonométrico, entre otros. (Tabla 1.4.1).

| Método | Descripción | Ejemplo |
|---|---|---|
| abs( x ) | valor absoluto de x | abs( -23.7 ) es 23.7 |
| max( x, y ) | el valor más grande x y y | max( 2.3, 12.7 ) es 12.7 |
| min( x, y ) | el valor más pequeño de x y y | min( -2.3, -12.7 ) es -12.7 |
| sqrt( x ) | raíz cuadrada de x | sqrt( 900.0 ) es 30.0 |

***Tabla 1.4.1. Métodos de la clase Math***

- **Métodos de instancia**: son invocados en objetos, una vez que se tiene un objeto que soporta métodos, se pueden usar los métodos de esos objetos.

## • Declaración

Un método es un bloque de código al que se puede transferir el control. La sintaxis para declarar un método es la siguiente y a continuación se describe cada parte:


***1) Modificador de acceso.***
Los valores posibles del modificador de acceso son los siguientes: public (público), private (privado) y protected (protegido), los mismos que para la clase.

En el siguiente ejemplo, **public** es el modificador de acceso:

**public** int suma(int a, int b)

Mientras que para este ejemplo, el modificador de acceso es **private**:

**private** float promedio (int x, int y, int z)

***2) Valor de regreso.***
En un método se utiliza la sentencia *return* para devolver un valor y en la declaración se indica **el tipo de dato** que se regresa, por ejemplo: int, float, double, etc. Si el método no devuelve nada se coloca la palabra reservada ***void*** que se puede traducir como vacío. En el siguiente ejemplo, se regresa un valor de tipo **int**.

public **int** suma(int a, int b)

***3) Nombre***

Debe comenzar en minúscula; en el caso de tener un nombre compuesto por más de una palabra, entonces, cada inicial debe comenzar en mayúscula, excepto la primera. Observa los siguientes ejemplos:

public int **suma**(int a, int b)
public float **promedioCalificaciones**(int x, int y, int z)
private double **hipotenusaTriangulo** (int cateto_a, int cateto_b)

***4) Parámetros***

Cuando se declara un método, se puede especificar una lista de parámetros separados por comas e indicando el tipo de dato e identificador y todo entre paréntesis. Los parámetros o datos de entrada estarán accesibles en el cuerpo del método, usando los nombres que se les dio en la declaración. Por ejemplo: el método suma va a solicitar dos valores enteros **int a** , **int b**.

public int suma(**int a, int b**)

En seguida te mostramos algunos ejemplos de métodos.

**Ejemplo 1. Mostrar datos en pantalla.**

El siguiente método lo llamaremos ***mostrarDatos,*** el cual recibe como parámetros los valores de color, marca, modelo y año; y posteriormente los imprime en pantalla.

```
public void mostrarDatos(String color, String marca, String modelo, int anio)
{
        System.out.println("Datos Automóvil");
        System.out.println("Color: "+color);
        System.out.println("Marca:"+marca);
        System.out.println("Modelo:"+modelo);
        System.out.println("Año:"+anio);
}
```

**Ejemplo 2. Área de un cuadrado.**

El siguiente método calcula el área de un cuadrado sin solicitar datos de entrada, es decir no recibe parámetros, pero regresa el resultado del área:

```
public double calcularArea( )
{
        double lado=5;
        double area=lado*lado;
        return area;
}
```

**Ejemplo 3. Conversión de temperatura.**
En este ejemplo el método recibe la temperatura en grados Fahrenheit y regresa el valor equivalente en grados Celsius.

```
public double gradosCelsius(double  grados_Fahrenheit)
{
        double celsius = (grados_Fahrenheit - 32)*(5/9);
        return celsius;
}
```

- ## Formas de llamar a un método

Una vez declarado un método existen tres formas de llamarlo: en la clase que se declaró o desde otra clase:

**1. Mediante su nombre dentro de la misma clase.**
La siguiente clase llamada Automovil muestra los datos de un automóvil declarando primero el método **mostrarDatos** y después mandándolo a llamar en el método principal.

```
//se declara la clase
public class Automovil
{
   //se declara el método de clase para mostrar los datos
   public static void mostrarDatos(String color,String marca, String modelo, int anio)
   {
      //se imprimen los datos en pantalla
      System.out.println("Datos Automóvil");
      System.out.println("Color: "+color);
      System.out.println("Marca:"+marca);
      System.out.println("Modelo:"+modelo);
      System.out.println("Año:"+anio);
   }
   //se declara el método principal
   public static void main(String[ ] args)
   {
      //se llama al método mostrarDatos y se pasan los parámetros
      mostrarDatos("negro","Vokswagen","Beetle",2001);
   }
}
```

**2. Utilizando una variable que hace referencia a un objeto.**
Para ello se utiliza una variable que hace referencia a un objeto seguida de un punto (.) y el nombre del método que se utilizará. Por ejemplo:

*triangulo1.calcularArea(double b, double h)*

Donde ***triángulo1*** es un objeto y ***calcularArea(double b, double h)*** es el método. Esta forma de llamar a un método se verá con más detenimiento en el siguiente tema.

**3. Mediante métodos static (estáticos).**

Una clase o método declarado static puede ser accedido sin la necesidad de utilizar un objeto como en el caso anterior. Para ello, utilizamos el nombre de la clase y un punto (.) para llamar a un método static de una clase.

La siguiente clase llamada Raiz calcula la raíz cuadrada de un valor invocando el método de clase sqrt de la clase Math.

```
//importamos la biblioteca de la clase Math
import java.lang.Math.*;
//se declara la clase Raiz
public class Raiz
{
   //se declara el método principal
   public static void main(String[ ] args)
   {
      //se declaran dos variables
      double a=900;
      double raiz;
      //se llama al método sqrt de la clase Math
      raiz=Math.sqrt(a);
      //Se imprime en pantalla el valor de la variable y su raíz cuadrada
      System.out.println("Raiz cuadrada de "+a+" = "+raiz);
   }
}
```

- ## Ventajas de usar métodos

Ahora bien, dentro de las ventajas que obtenemos al trabajar en módulos mediante el uso de métodos tenemos:

1. Una mayor facilidad para administrar los programas.
2. Reutilizar el código para evitar volver a escribirlo.
3. Evitar repetir la escritura del código del programa.

Veamos la conveniencia entre utilizar o no un método, pensemos en realizar un programa que lleve a cabo la suma de dos números enteros y que muestre el resultado en pantalla, hagamos en primer lugar el programa sin utilizar un método, con ello tendríamos algo como esto:

```
public class Ejemplo {
        public static void main(String[ ] args) {
            int x = 10;
            int y = 20;
```

```
        int z = x + y;
        System.out.println("La suma es: "+z);
    }
}
```

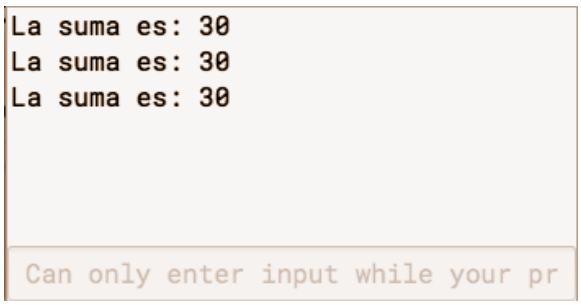
Ahora utilicemos un método para llevar a cabo el mismo proceso, vamos a llamarlo sumar y tendrá que realizar la suma de dos números y presentar el resultado, el código es el siguiente:

```
//Se crea la clase EjemploSuma
public class EjemploSuma {
//Se crea el método sumar
public static void sumar ( ) {
        int x = 10;
        int y = 20;
        int z = x + y;
        System.out.println("La suma es: "+z);
        }
//Se ejecuta el método principal
public static void main(String[ ] args) {
        //Se llama al método sumar
        sumar( );
        }
}
```

Seguramente aquí te preguntarás: ¿y qué ventajas me ofrece si parece lo mismo? La respuesta es que el método (sumar) puede mandarse a llamar la cantidad de veces que uno requiera sin tener que volver a escribir las instrucciones que lo componen. A continuación, se muestra el programa llamando tres veces al método.

```
//Se crea la clase EjemploSuma
public class EjemploSuma {
//Se crea el método sumar
public static void sumar ( ) {
        int x = 10;
        int y = 20;
        int z = x + y;
        System.out.println("La suma es: "+z);
        }
//Se ejecuta el método principal
public static void main(String[ ] args) {
        //Se llama al método sumar 3 veces
        sumar( );
        sumar( );
        sumar( );
        }
}
```

Lo cual nos da como resultado la siguiente salida:




Hagamos una pequeña modificación para que en el ejemplo anterior tengamos la posibilidad de mandar información al método y de esta forma sea más clara la ventaja de su utilización.

Al mandar a llamar al método **sumar,** vamos a agregar como argumento el valor de los números que queremos sumar, es decir modificamos la línea de código:

***public static void sumar ( ) {***

por la siguiente línea:

***public static void sumar (int x, int y) {***

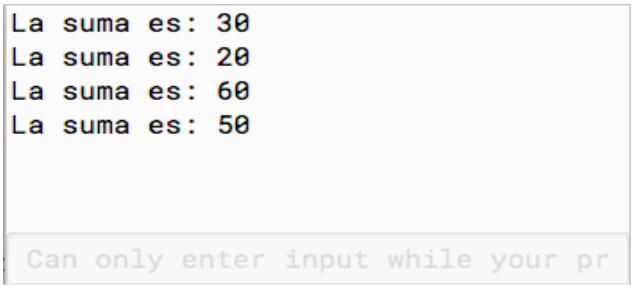
Observa con detenimiento los cambios hechos:

```
//Se crea la clase EjemploSuma
public class EjemploSuma {
//Se crea el método sumar que recibe como parámetros los números a sumar.
public static void sumar (int x, int y) {
        int z = x + y;
        System.out.println("La suma es: "+z);
        }
//Se ejecuta el método principal
public static void main(String[ ] args) {
        //Se llama al método enviando los números a sumar
        sumar (10,20);
        }
}
```

Ahora vamos a utilizar este método que ya está implementado para hacer cuatro sumas diferentes, tal como puedes observar no es necesario volver a escribir el método y las instrucciones que lo componen, basta con llamarlo la cantidad de veces que se requieran enviando como argumento los números a sumar, observa el código y la correspondiente salida:

```
//Se crea la clase EjemploSuma
public class EjemploSuma {
```

```
//Se crea el método sumar que recibe como parámetros los números a sumar.
public static void sumar (int x, int y) {
        int z = x + y;
        System.out.println("La suma es: "+z);
        }
//Se ejecuta el método principal
public static void main(String[ ] args) {
        //Se llama al método enviando los números a sumar
        sumar (10,20);
        sumar (5,15);
        sumar (20,40);
        sumar (35,15);
        }
}
```




- **Métodos Getter y Setter**

Son dos métodos convencionales para obtener (get) y actualizar (set) información. En la POO es importante que los atributos que pertenecen a una clase sean de tipo privado, es decir, otras clases no tienen permiso de ver y modificar directamente los valores; sin embargo, es necesario que se pueda acceder a ellos de una forma indirecta y para ello se utilizan estos métodos.

La declaración del método Getter es la siguiente:

```
public   tipo_dato_atributo   getAtributo( )
{
        return atributo;
}
```

Las características de este método son:

- Es público, porque queremos mostrar el valor desde fuera de la clase.
- Devuelve un valor, por lo que se emplea return.
- El nombre del método empieza con *get* y el nombre del atributo con mayúscula.
- No tiene parámetros.
- Sirve para obtener o recuperar el valor de un atributo.

La forma de declarar un método Setter es:

```
public void setAtributo (tipo_dato_atributo variable)
{
        this.atributo = variable;
}
```

Las siguientes características de este método son:

- Es público, porque queremos modificar el valor desde fuera de la clase.
- No devuelve un valor, por lo que se emplea *void*.
- El nombre del método empieza con *set* y el nombre del atributo con mayúscula.
- Tiene parámetros, se indica el tipo de dato y nombre.
- Utiliza la palabra reservada *this* para guardar el valor de la variable en el atributo de la clase.
- Sirve para actualizar o modificar el valor de un atributo.

Analicemos un ejemplo de cómo utilizar los métodos Getter y Setter en una clase. Pensemos en una clase llamada ***Examenes*** que contiene como atributos la calificación de un examen, y como métodos se tienen el Getter (getCalificacion) con la cual se obtiene el valor de esta y el Setter (setCalificacion) para actualizar el valor correspondiente. El código de estos métodos es el siguiente:

```
//se declara la clase Examenes
public class Examenes
{
   //se declaran los atributos
   private int calificacion;
   //se declara el método get del atributo
   public int getCalificacion( )
   {
      //se regresa el valor del atributo
      return calificacion;
   }
   //se declara el método set del atributo
   public void setCalificacion(int numero)
   {
      //se modifica el valor del atributo
      this.calificacion=numero;
   }
}
```

Actividad 1.5

## Cuestionario

1. Selecciona algunos métodos que puede tener la clase Automovil:

| a) color, marca, modelo | b) auto1, auto2, auto3 | c) frenar, arrancar, acelerar | d) placas, año, puertas |
|---|---|---|---|

2. Una ventaja de emplear métodos es:
   a) Reutilizar código. b) Borrar código. c) Reescribir código. d) Corregir código.

3. Método que está correctamente declarado:
   a) public mover( )
   b) public void mover( )
   c) mover( )
   d) public void mover

4. Cómo se llama a un método de clase:
   a) sqrt(a) b) Sqrt(a) c) raiz1.sqrt(a) d) Math.sqrt(a)

5. Una característica del método Getter:
   a) Emplea void. b) Tiene parámetros. c) Emplea return. d) Emplea this.

6. Sintaxis para instanciar un objeto:
   a) objeto=new NombreClase( );
   b) NombreClase objeto=new NombreClase( );
   c) NombreClase new objeto;
   d) new objeto NombreClase( );

7. Elemento que no debe estar presente en la declaración del método Setter:
   a) return b) setColor c) this d) void

8. Método para actualizar desde fuera de la clase el valor de un atributo:
   a) main b) Getter c) Setter d) constructor

9. Definición de objeto:
   a) Comportamiento de una clase.
   b) Característica de una clase.
   c) Instancia o ejemplo de una clase.
   d) Plantilla que define características y comportamientos.

10. Forma de mandar a llamar a un método de instancia:
    a) sumar(5,6) b) s1.sumar(5,6) c) Math.sumar(5,6) d) Sumar(5,6)

## 1.5 Objetos

**Propósito:**

Conocer qué es un objeto, cómo se declara y aprender a instanciar objetos y llamar los métodos de una clase.

- **Definición**

Es un ejemplar o instancia (del término inglés *instance = ejemplo*) de datos específicos y rutinas que pueden operarse con esos datos de una clase dada. Un objeto tiene los atributos de la clase a la que pertenece y lleva a cabo acciones con los atributos llamados métodos. En la siguiente figura se muestra la clase ***Persona*** y tres objetos de esta clase.

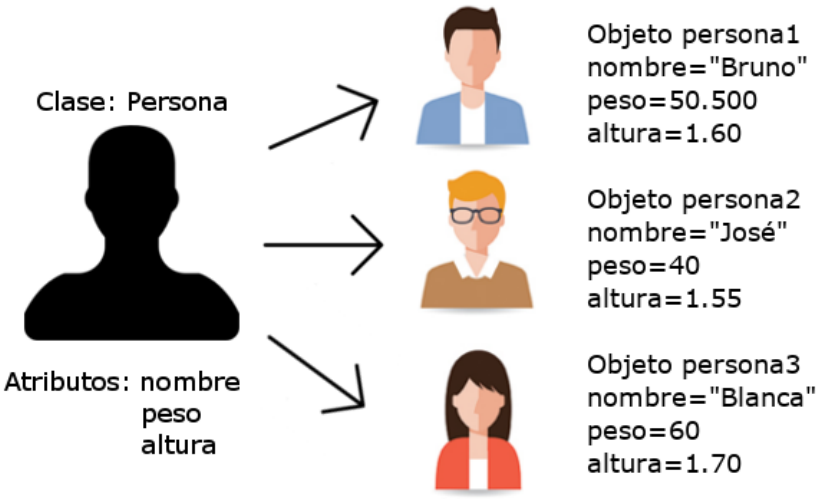



***Figura 1.5.1. Ejemplos de objetos de la clase Persona.***

En un segundo ejemplo pensemos en la clase ***PoligonosRegulares*** en donde los atributos serán lados y color.

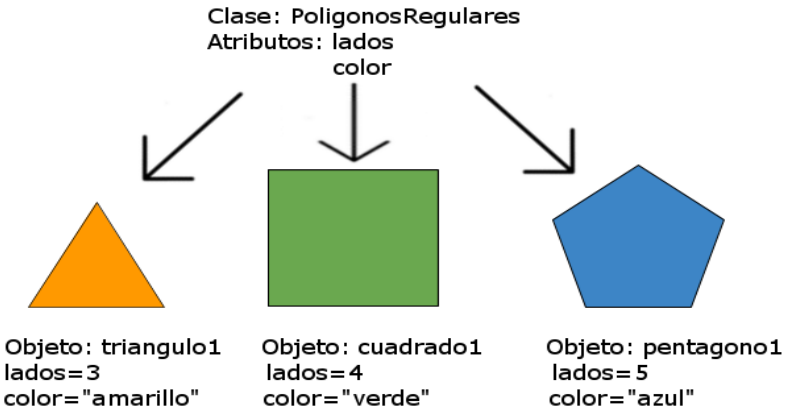



***Figura 1.5.2. Ejemplo de objetos de la clase PoligonosRegulares.***

- **Declaración**

Cuando creamos una clase en Java, ésta se convierte en un nuevo tipo de dato que puede usarse para crear objetos. La siguiente sintaxis se utiliza para instanciar un objeto como regularmente se le conoce:

*NombreClase objeto = new  NombreClase( );*

Donde:

***NombreClase***: se refiere al nombre de la clase que va a instanciar el objeto.
***objeto***: es el identificador o nombre que vamos a asignar al objeto.
***new***: es una palabra reservada de Java que sirve para crear objetos.

Para instanciar los objetos de la figura 1.5.2 emplearíamos las siguientes sentencias:

```
PoligonosRegulares triangulo1=new PoligonosRegulares( );
PoligonosRegulares cuadrado1=new PoligonosRegulares( );
PoligonosRegulares pentagono1=new PoligonosRegulares( );
```

Al declarar un objeto podemos llamar los métodos de la clase con la siguiente sintaxis: nombre del objeto punto nombre del método y los parámetros entre paréntesis.

*nombre_del_objeto.nombre_del_metodo(parámetros)*

- **Implementación**

Implementar un programa que declare la clase ***Persona*** (Fig. 1.5.1) con los atributos nombre, peso y altura, y que muestre los datos en pantalla de los objetos de la clase.

Primero declaramos la clase con sus atributos:

```
public class Persona
{
   private String nombre;
   private double peso;
   private double altura;
```

En seguida declaramos los métodos Setter de los atributos para modificar el valor de los objetos:

```
   public void setNombre(String nombre)
   {
      this.nombre=nombre;
   }

   public void setPeso(double peso)
   {
      this.peso=peso;
   }

   public void setAltura(double altura)
   {
      this.altura=altura;
   }
```

Después declaramos un método para mostrar los atributos del objeto:

```
public void mostrarDatos( )
{
      System.out.println("Nombre:"+nombre);
      System.out.println("Peso:"+peso);
      System.out.println("Altura:"+altura);
}
```

Finalmente, en el método principal *main* instanciamos un objeto llamado persona1, modificamos los valores de los atributos llamando los métodos Setter y mostramos en pantalla los valores del objeto.

```
public static void main(String[ ] args)
{
     Persona persona1=new Persona( );
     System.out.print("\u000C");
      persona1.setNombre("Bruno");
      persona1.setPeso(50.500);
      persona1.setAltura(1.60);
      System.out.println("Objeto 1");
      persona1.mostrarDatos( );
```

Hacemos lo mismo con los otros dos objetos de la figura: instanciamos, modificamos variables de instancia y mostramos los atributos del objeto.

```
      Persona persona2=new Persona( );
      persona2.setNombre("José");
      persona2.setPeso(50.500);
      persona2.setAltura(1.55);
       System.out.println("Objeto 2");
      persona2.mostrarDatos( );
      Persona persona3=new Persona( );
      persona3.setNombre("Blanca");
      persona3.setPeso(60);
      persona3.setAltura(1.70);
       System.out.println("Objeto 3");
      persona3.mostrarDatos( );
}
```

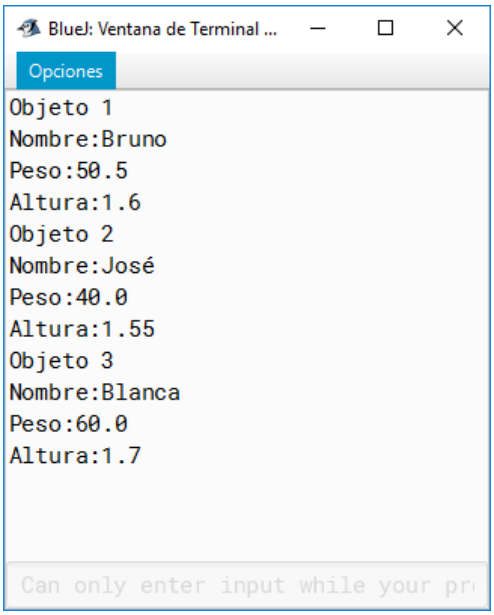
Escribiendo el programa completo queda como sigue:

```
//Se declara la clase Persona
public class Persona
{
   //Se declaran los atributos de la clase
   private String nombre;
   private double peso;
   private double altura;
```

```
//Se declaran los métodos Setter de los atributos
public void setNombre(String nombre)
{
   this.nombre=nombre;
}
public void setPeso(double peso)
{
   this.peso=peso;
}
public void setAltura(double altura)
{
   this.altura=altura;
}
//Se declara el método para mostrar los atributos
public void mostrarDatos( )
{
   System.out.println("Nombre:"+nombre);
   System.out.println("Peso:"+peso);
   System.out.println("Altura:"+altura);
}
//Se declara el método principal
public static void main(String[ ] args)
{
   //Se limpia la pantalla
   System.out.print("\u000C");
   //Se instancia el objeto 1
   Persona persona1=new Persona( );
   //Se modifican los valores de los atributos con los métodos Setter
   persona1.setNombre("Bruno");
   persona1.setPeso(50.500);
   persona1.setAltura(1.60);
   //Se imprime los valores del objeto 1
   System.out.println("Objeto 1");
   persona1.mostrarDatos( );
   //Se instancia el objeto 1
   Persona persona2=new Persona( );
   //Se modifican los valores de los atributos con los métodos Setter
   persona2.setNombre("José");
   persona2.setPeso(50.500);
   persona2.setAltura(1.55);
   //Se imprime los valores del objeto 2
   System.out.println("Objeto 2");
   persona2.mostrarDatos( );
   //Se instancia el objeto 1
   Persona persona3=new Persona( );
   //Se modifican los valores de los atributos con los métodos Setter
   persona3.setNombre("Blanca");
   persona3.setPeso(60);
```

```
        persona3.setAltura(1.70);
        //Se imprime los valores del objeto 3
        System.out.println("Objeto 3");
        persona3.mostrarDatos( );
    }
}
```

La salida del programa muestra los valores de los objetos declarados.




En el siguiente ejemplo vamos a declarar objetos de la clase ***Examenes*** y utilizar los métodos Setter y Getter para calcular en el método principal el promedio de tres calificaciones.

Primero declaramos la clase y sus atributos:

```
public class Examenes
{
        private int calificacion;
```

En seguida se declara los métodos Getter y Setter:

```
public int getCalificacion( )
{
        return calificacion;
}

public setCalificacion(int numero)
{
        this.calificacion=numero;
}
```

Después en el método principal se declaran tres objetos:

```
public static void main(String[ ] args)
{
        Examenes cal1=new Examenes( );
        Examenes cal2=new Examenes( );
        Examenes cal3=new Examenes( );
```

En seguida se utiliza el método Setter para modificar los valores del atributo de cada objeto:

```
        cal1.setCalificacion(6);
        cal2.setCalificacion(8);
        cal3.setCalificacion(10);
```

Se declara una variable para calcular el promedio con el método Getter que obtiene el atributo calificación de cada objeto:

```
        double promedio;
        promedio=(cal1.getCalificacion( )+cal2.getCalificacion( )+cal3.getCalificacion( ))/3;
```

Por último, se imprime en pantalla las calificaciones y su promedio.

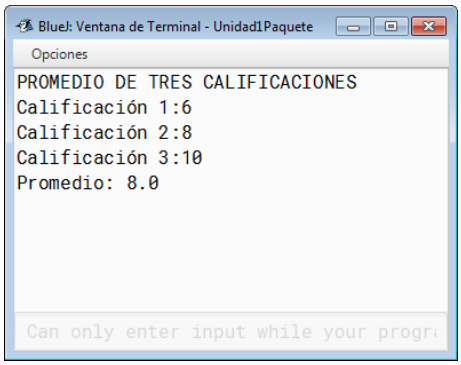
```
        System.out.println("PROMEDIO DE TRES CALIFICACIONES");
        System.out.println("Calificación 1:"+cal1.getCalificacion( ));
        System.out.println("Calificación 2:"+cal2.getCalificacion( ));
        System.out.println("Calificación 3:"+cal3.getCalificacion( ));
        System.out.println("Promedio: "+promedio);
}
```

Escribiendo el programa completo queda como sigue:

```
//se declara la clase Examenes
public class Examenes
{
   //se declaran los atributos
   private int calificacion;
   //se declara el método Getter del atributo
   public int getCalificacion( )
   {
      //se regresa el valor del atributo
      return calificacion;
   }
   //se declara el método Setter del atributo
   public void setCalificacion(int numero)
   {
```

```
    //se modifica el valor del atributo
    this.calificacion=numero;
  }
  //se declara el método main
  public static void main(String[ ] args)
  {
    //se declaran tres objetos de la clase
    Examenes cal1=new Examenes( );
    Examenes cal2=new Examenes( );
    Examenes cal3=new Examenes( );
    //se actualizan los valores de los objetos con el método Setter
    cal1.setCalificacion(6);
    cal2.setCalificacion(8);
    cal3.setCalificacion(10);
    //se declara la variable promedio
    double promedio;
    //se calcula el promedio de los valores de los objetos
    promedio=(cal1.getCalificacion( )+cal2.getCalificacion( )+cal3.getCalificacion( ))/3;
    //mostramos en pantalla los valores de los objetos y el promedio
    System.out.println("PROMEDIO DE TRES CALIFICACIONES");
    System.out.println("Calificación 1:"+cal1.getCalificacion( ));
    System.out.println("Calificación 2:"+cal2.getCalificacion( ));
    System.out.println("Calificación 3:"+cal3.getCalificacion( ));
    System.out.println("Promedio: "+promedio);
  }
}
```

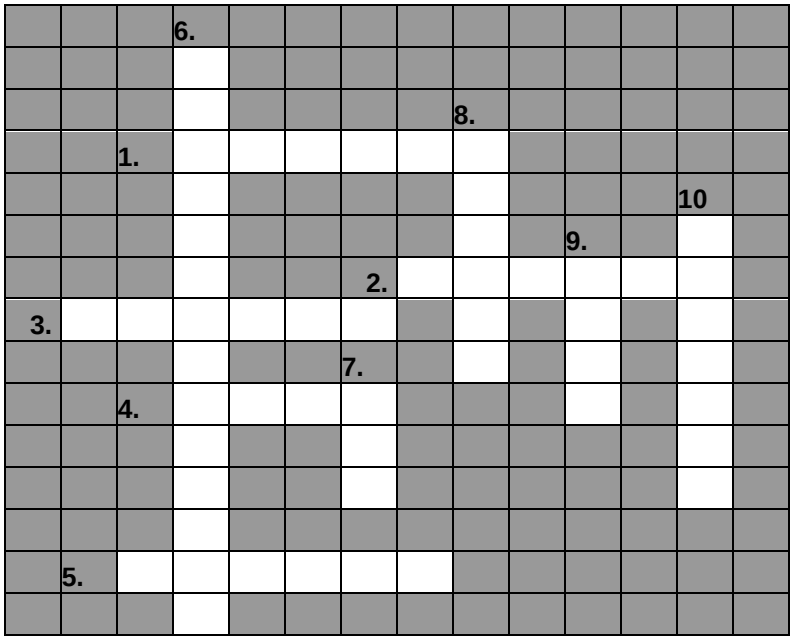
La salida del programa muestra las calificaciones y su promedio:



Actividad 1.6

**Crucigrama**

| HORIZONTALES | VERTICALES |
|---|---|
| 1. Es una acción que un objeto puede realizar y se define en la clase. | 6. Otra forma como se le conoce a un método. |
| 2. Método para obtener desde fuera el atributo de una clase. | 7. Palabra reservada con la que se declara un objeto. |
| 3. Palabra reservada para regresar un valor en un método. | 8. Es un ejemplar o instancia de una clase dada. |
| 4. Método que si no se declara no se ejecuta la aplicación. | 9. Palabra reservada con la que se hace referencia a una variable. |
| 5. Con esta palabra se declara un método de clase. | 10. Modificador de acceso que permite usar el método internamente en la clase. |



## 1.7 La clase Scanner.

**Propósito:**

Emplear la Clase Scanner para la entrada de datos en la creación de un programa.

En programación casi siempre se tienen que procesar datos, cuando los datos los tiene que proporcionar el usuario por medio del teclado, se debe tener un mecanismo mediante el cual esto sea posible. Este mecanismo es proporcionado por la clase Scanner.

Java tiene una clase que nos permite leer datos desde un archivo de texto y también desde el teclado, esta es la clase **Scanner** y está dentro del paquete **java.util**.
Los objetos de tipo Scanner pueden llamar a los métodos que permiten leer datos del tipo que sea necesario.

Los pasos para hacer uso de la clase Scanner son los que se indican a continuación:

- **Importar la Clase java.util.Scanner.**

  La clase Scanner pertenece a una librería del sistema y para utilizarla debemos importarla colocando la siguiente línea en la sección de importación de clases externas justo antes de iniciar el código de la clase.

  **import java.util.Scanner;**

- **Definición del objeto de la Clase Scanner (instanciación).**

  Para poder utilizar los métodos de la clase ***Scanner*** e ingresar datos debe instanciarse en el objeto que deseamos utilizar para tener acceso a los métodos de la clase, en este caso se crea una instancia en el objeto llamado **teclado**.

  Scanner **teclado** = new Scanner(System.in);

  En donde:

| | |
|---|---|
| Scanner teclado | Se define el objeto de tipo Scanner en la variable teclado |
| =new Scanner | Se crea la instancia y se asigna a la variable |
| (System.in) | Especifica el origen de los datos |

  Cuando se utiliza el canal definido por default o por omisión en el método System.in, este se refiere al canal de entrada de datos desde el teclado físico de la computadora.

De esta forma ya podemos utilizar los métodos de lectura desde el objeto instanciado **teclado**.

Cabe aclarar que **teclado** es el nombre asignado al objeto, pudo haber sido cualquier otro, por ejemplo: leer, lectura, teclas, entrada, canal, ingresarDato, etc. respetando las reglas para los nombres de los identificadores.

- ## Método System.in

Es el método mediante el cual se ingresa información, representa el canal de entrada estándar que por default corresponde al teclado de la computadora.

A continuación mostramos un ejemplo de su uso:

**Ejemplo:**

```
int edad;
Scanner teclado = new Scanner(System.in);
System.out.println("¿Qué edad tienes?: ");
edad = teclado.nextInt( );
```

Veamos paso a paso que es lo que significa cada línea:

*int edad;*

Estamos declarando una variable de tipo **int** llamada **edad** la cual va a almacenar un número entero.

*Scanner teclado = new Scanner(System.in);*

Estamos instanciando un objeto de tipo Scanner el cual va a almacenar la información que el usuario ingrese.

*System.out.println("¿Qué edad tienes?: ");*

Estamos usando el método **println** para preguntar al usuario por su edad.

*edad = teclado.nextInt( );*

Estamos usando el objeto **teclado** para obtener la información del usuario, luego empleamos el método **nextInt( )** porque vamos a leer un número entero, y por último estamos pasando el valor **int** a la variable edad.

- **Introducción de datos desde el teclado**

  Para ingresar datos utilizamos la instancia de la clase **Scanner** que hemos llamado **teclado** con alguno de los métodos para la conversión del dato leído al tipo de dato que sea requerido.

| Tipo de dato | Definición de variable | Método para ingresar dato |
|---|---|---|
| cadena | String cadena; | cadena = teclado.**next**( ); |
| línea | String linea; | linea = teclado.**nextLine**( ); |
| booleano | boolean valorLogico; | valorLogico = teclado.**nextBoolean( )**; |
| entero | int numeroEntero; | numeroEntero = teclado.**nextInt( )**; |
| entero largo | long numeroEntero; | numeroEntero = teclado.**nextLong**( ); |
| flotante | float numeroFlotante; | numeroFlotante = teclado.**nextFloat**( ); |
| doble | float numeroDoble; | numeroDoble = teclado.**nextDouble**( ); |

***Tabla 1.6.1. Métodos para ingresar datos por teclado.***

## 1.8 Errores sintácticos y lógicos.

Los errores de sintaxis o sintácticos son aquellos que se presentan cuando dentro del código no se respetan las reglas de sintaxis y aparecen al escribir un programa. Típicamente se detectan durante la etapa de compilación y son relativamente fáciles de corregir. Si no se corrigen no es posible generar el código objeto o "código ejecutable" y por lo tanto no se puede ejecutar o "correr" el programa. En este punto cabe mencionar que Java es un lenguaje en el que el código ejecutable se representa por medio de los llamados "*bytecodes*", este código es ejecutado por la máquina virtual de Java en cualquier dispositivo, lo cual lo hace ser multiplataforma.

Un ejemplo se presenta si en el momento de escribir el código definimos una variable de cierto tipo y pretendemos ingresar datos de diferente tipo. Ejemplo:

*int numeroEntero;*

*numeroEntero = teclado.nextLine( );*

Se produce el error de sintaxis, la forma correcta debe ser:

*numeroEntero = teclado.nextInt( );*

Los errores lógicos o de lógica son aquellos errores que se producen al momento de ejecutar un programa. Como se mencionó antes los errores de sintaxis en este punto ya han sido resueltos pero no por eso el programa se ejecutará sin fallas. Todos los programas pueden

presentar casos de excepción cuando se están ejecutando, es por eso que los procesos de validación de datos son tan importantes. Un caso típico que ilustra la aparición de un error de lógica es la división por cero. Si dentro del programa se presenta este caso, el programa terminará abruptamente ya que la división por cero no está definida. Es responsabilidad del programador prever este y otro casos en los que se puedan dar situaciones como esta.

Cuando utilizamos la clase Scanner puede ocurrir un error en el momento de ingresar los datos. Si esperamos un valor de cierto tipo y se ingresa un valor del tipo distinto se produce un error de lógica.

**Ejemplo:**

Al estar ejecutando el programa en la línea:

```
numeroEntero = teclado.nextInt( );
```

Se espera que ingrese un valor de tipo numérico entero y si se ingresar un valor numérico con parte decimal se produce un error lógico en tiempo de ejecución y el programa termina abruptamente.

A continuación se presentan ejemplos de los casos más comunes donde se presentan errores.

1. Olvidar el punto y coma al final de las líneas, aunque el punto y coma no se coloca en al final de todas las líneas.

```
if (a==b);  //error, no va el punto y coma en esta línea
    a=1;
```

2. Muchas veces olvidamos o confundimos el operador de asignación = con el operador de comparación ==.

```
if (a=b)  //error
```

debe ser:

```
if (a==b)
```

El punto y coma en el if no se pone, para evitar esto se deben ponen llaves { }

```
if (a==b){
    a=1;
}
```

3.- Es un error colocar los tipos de datos al utilizar un método.

```
z=objeto.metodo(int x, String y); //error
```

Cuando llamamos un método no se ponen los tipos de datos.

```
z=objeto.metodo(x, y); //correcto
```

4. Utilizar el comparador == para comparar 2 variables de tipo String es un error. Se debe usar el método equals.

```
String a="Hola", b="Hola";
if(a==b) //error
{
        System.out.println("Las cadenas son iguales")
}
```

Lo correcto es: *if (a.equals(b)*

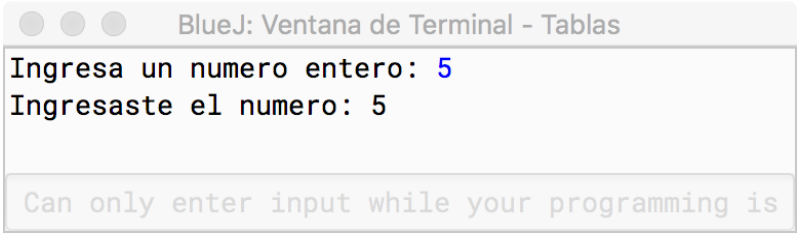
5. Cuando definimos un método es muy común no incluir el valor de retorno cuando el método no es void.
Existen muchos errores más, pero estos son algunos de los más comunes.

## 1.9 Ejecución del programa

Ejemplo de como ingresar un dato numérico desde el teclado y mostrarlo en pantalla.

```
//Se importa la biblioteca Scanner
import java.util.Scanner;
//Se declara la clase IngresaDato
public class IngresaDato {
        //Se declara el método principal
        public static void main(String[] args) {
        //Se instancia un objeto de la clase Scanner
        Scanner teclado=new Scanner(System.in);
        //Se declara una variable entera
        int numeroEntero;
        System.out.println("Ingresa un número entero: ");
        //Se ingresa el número desde el teclado
        numeroEntero = teclado.nextInt();
        //Se muestra en pantalla en número anterior
        System.out.println("Ingresaste el número: " +numeroEntero);
        }
}
```

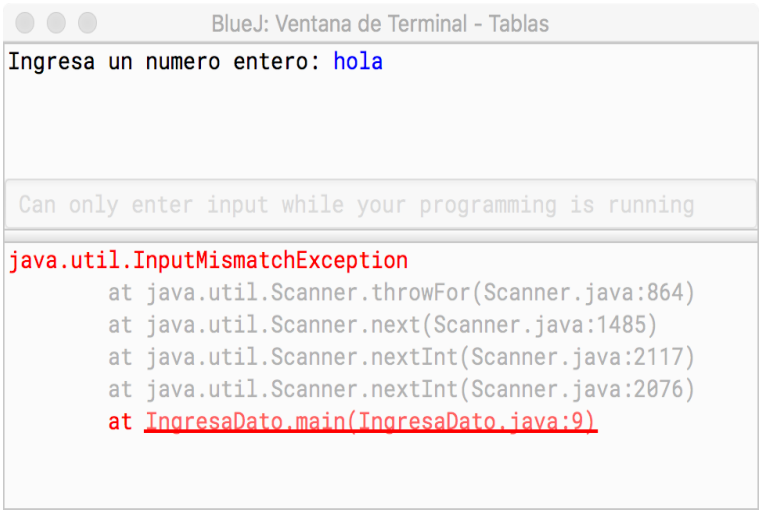
**Ejecución:**




Algunos ejemplos de errores de lógica en tiempo de ejecución son los siguientes:

1. Se espera un número entero y se ingresa un número con punto decimal (**7.5**)


2. Se espera un número entero y se ingresa la palabra **hola**




En ambos casos se produce un error que se muestra en el mensaje de la parte baja de la imagen anterior la cual corresponde a la ventana de terminal donde se muestra la ejecución del programa.

Actividad 1.7

**Cuestionario.**

1. La clase Scanner se usa cuando queremos:
   a) Imprimir datos b) Leer datos c) Borrar datos d) Limpiar datos

2. Uno de los métodos de la clase Scanner para ingresar números enteros es:
   a) next b) nextLine c) nextInt d) nextFloat

3. La forma de importar la clase Scanner es:
   a) import java.util.Scanner; b) import Scanner.util.java;
   c) import util.java.Scanner**;** d) import util.Scanner;

4. La forma de instanciar la clase Scanner es:
   a) Scanner teclado = new Scanner(System.in); b) Scanner teclado;
   c) new Scanner(System.in); d) new Scanner(System.in);

5. Los errores de sintaxis se detectan:
   a) Cuando se ejecuta el programa b) Cuando se compila el programa c) Al ingresar los datos d) Al escribir el código en block de notas

## 1.10 Implementación

**Propósito:**

Construir programas de computadora para reafirmar los conceptos adquiridos en la unidad**.**

- **Calcular el ángulo y perímetro de un polígono regular**

Realiza un programa que solicite el número de lados de un polígono regular así como la longitud del lado, y calcule el valor del ángulo con la expresión ángulo=360/(número de lados); y el perímetro con la fórmula perímetro=longitud*(número de lados).

**Solución:**

Debemos importar la clase Scanner para poder solicitar los datos. En seguida se crea la clase, se indican los atributos y se declara el constructor sin parámetros.

```
import java.util.Scanner;
public class Poligonos
{
   int numeroDeLados;
   float longitudDelLado;
```

Después es momento de declarar los métodos Setter y Getter de los atributos.

```
public void setNumeroDeLados(int lados)
{
   this.numeroDeLados=lados;
}
public void setLongitudDelLado(float longitud)
{
   this.longitudDelLado=longitud;
}
public int getNumeroDeLados( )
{
   return numeroDeLados;
}
public float getLongitudDelLado( )
{
   return longitudDelLado;
}
```

Ahora se crean los métodos que van a calcular el ángulo y perímetro con las expresiones dadas.

```
public float calcularAngulo(int lados)
{
   float angulo;
   angulo=360/lados;
   return angulo;
}

public float calcularPerimetro(int lados, float longitud)
{
   float perimetro;
   perimetro=lados*longitud;
   return perimetro;
}
```

En el método principal se crea un objeto de la clase Scanner, otro de la clase Polígonos.

```
public static void main(String[ ] args)
  {
     Scanner teclado=new Scanner(System.in);
     Poligonos poligono=new Poligonos( );
```

Se solicitan los datos y se ocupan los métodos Setter para modificar los atributos.

```
System.out.println("Dame el número de lados:");
poligono.setNumeroDeLados(teclado.nextInt( ));
System.out.println("Dame la longitud del lado");
poligono.setLongitudDelLado(teclado.nextFloat( ));
```

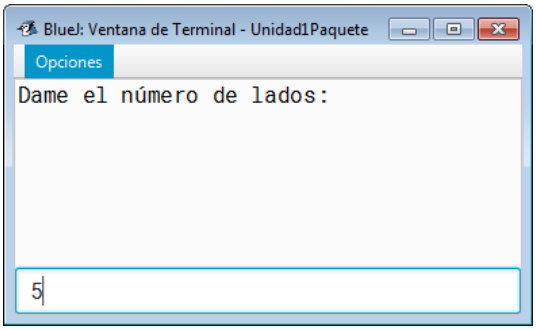
Mostramos en pantalla los datos ingresados, utilizando los métodos Getter.

```
System.out.println("POLIGONOS");
System.out.println("Número de lados="+poligono.getNumeroDeLados( ));
System.out.println("Longitud del lado="+poligono.getLongitudDelLado( ));
```

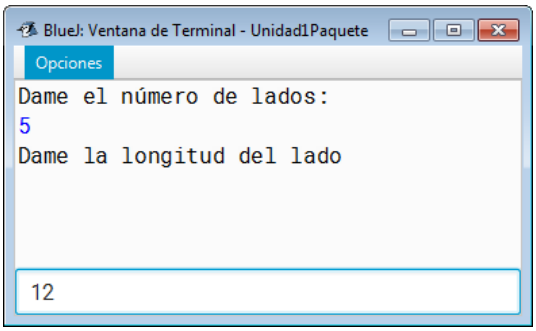
Llamando los método de instancia calcularAngulo y calcularPerímetro.

```
    System.out.println("Ángulo =
"+poligono.calcularAngulo(poligono.getNumeroDeLados( ))+" grados.");
System.out.println("Perímetro="+poligono.calcularPerimetro(poligono.getNumeroDeLados
( ),poligono.getLongitudDelLado( )));
  }
}
```

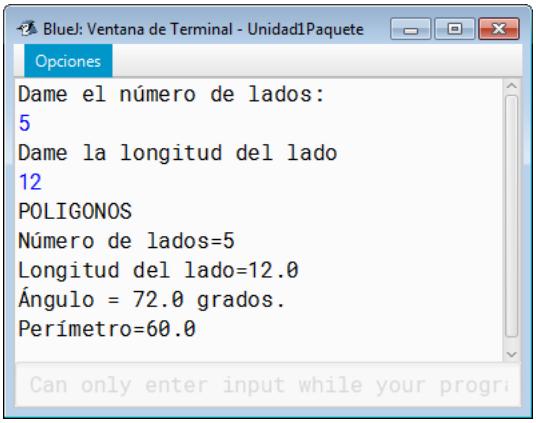
Al compilar y ejecutar el programa primero solicita el número de lados:




En seguida pide la longitud de cada lado:



Finalmente muestra los datos del polígono y el cálculo del ángulo y perímetro:




- **Hipotenusa de un triángulo rectángulo**

Realiza un programa que solicite el cateto opuesto y el cateto adyacente de un triángulo rectángulo y que calcule la hipotenusa correspondiente. Recuerda que la hipotenusa de un triángulo rectángulo es la raíz cuadrada de la suma del cuadrado de los catetos.

**Solución:**

En primer lugar, se importa la clase Scanner para poder leer datos desde el teclado, posteriormente se crea la clase TrianguloRectangulo que tiene de atributos la hipotenusa la cual es de tipo double de la siguiente forma:

```
import java.util.Scanner;
public class TrianguloRectangulo {
private double hipotenusa;
```

Se definen los métodos Setter y Getter de la clase

```
  //Se declara el método Setter de la clase
  public void setHipotenusa(double h) {
      this.hipotenusa = h;
  }
   //Se declara el método Getter de la clase
  public double getHipotenusa( ) {
      return hipotenusa;
   }
```

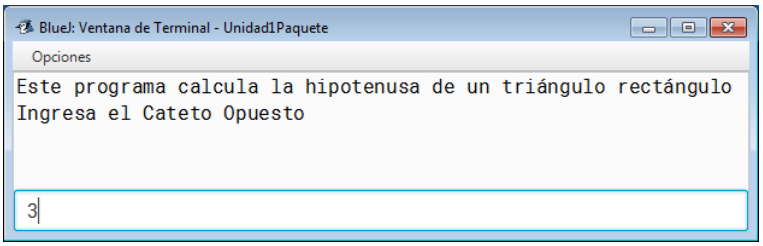
Se crea el método que calcula la hipotenusa y regresa el valor de esta.

```
  //Se envian como argumentos los valores de los angulos
public double calculaHipotenusa (double x , double y)  {
double hipo = Math.sqrt((x*x)+(y*y));
return hipo;
  }
```

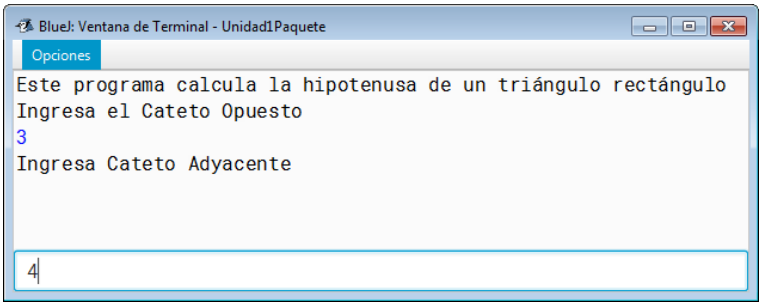
Se inicia el método principal

```
//Se crea la clase principal
  public static void main (String[ ] args) {
        //Se crea un objeto de la clase Scanner
        Scanner teclado = new Scanner (System.in);
        //Se crea un objeto de la clase TrianguloRectangulo
TrianguloRectangulo triangulo = new TrianguloRectangulo( );
        //Se crean dos variables de tipo double para ingresar los valores
        //del Cateto Opuesto y del Cateto Adyacente
        double x, y;
        System.out.println("Este programa calcula la hipotenusa de un triángulo rectángulo");
        System.out.println("Ingresa el Cateto Opuesto");
        x = teclado.nextDouble( );
        System.out.println("Ingresa Cateto Adyacente ");
        y = teclado.nextDouble( );
        //Se llama al método Setter y se le envia como argumento
        //El valor que regresa el método calculaHipotenusa
        triangulo.setHipotenusa (triangulo.calculaHipotenusa (x,y));
        //Se imprime en pantalla el valor de la hipotenusa mediante el método Getter
System.out.println("La Hipotenusa es: " + triangulo.getHipotenusa( ));
  }
}
```

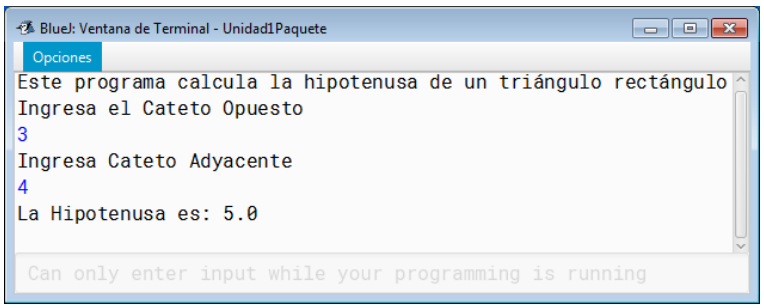
Si ejecutamos el programa primero solicita ingresar el valor del cateto opuesto:




En seguida se ingresa el valor del cateto adyacente:



Y finalmente nos muestra el valor calculado de la hipotenusa.



# Solución de las actividades

## Unidad 1

### Actividad 1.1

| | | | | |
|---|---|---|---|---|
| 1. i | 2. g | 3. e | 4. a | 5. f |
| 6. j | 7. h | 8. d | 9. b | 10. c |

### Actividad 1.2

| | | | | |
|---|---|---|---|---|
| 1. a | 2. b | 3. a | 4. d | 5. b |
| 6. b | 7. a | 8. d | 9. a | 10. c |
| 11. a | 12. b | 13. c | 14. a | 15. b |

### Actividad 1.3

1.

```
public class Libro
{
    String nombre;
    String autor;
    String editorial;
    int numPag;
 }
```

2.

```
public class Vehiculo
{
    String marca;
    String modelo;
    String color;
    int año;
 }
```

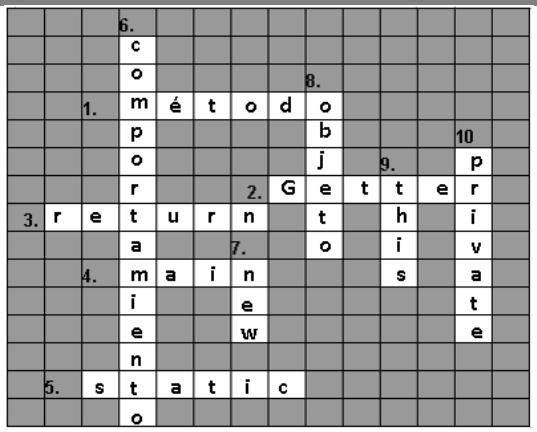
### Actividad 1.4

| | | | | |
|---|---|---|---|---|
| 1. a | 2. b | 3. a | 4. d | 5. a |

### Actividad 1.5

| | | | | |
|---|---|---|---|---|
| 1. c | 2. a | 3. b | 4. d | 5. c |
| 6. b | 7. a | 8. c | 9. c | 10. b |

### Actividad 1.6




### Actividad 1.7

| | | | | |
|---|---|---|---|---|
| 1. b | 2. c | 3. a | 4. a | 5. b |

## Referencias

## Unidad 1


Biblioteca informática - LÓGICA DE PROGRAMACIÓN. [en línea]. Recuperado el 24 de octubre de 2018 de https://sites.google.com/site/israelcortess/documentos/biblioteca-informatica-3

Ceballos Sierra, F. (2000). *Java 2*. RA-MA Editorial.

Deitel, P. (2008) *Cómo programar en Java. Pearson Educación. México.*

*EcuRed. [en línea]. Recuperado el 24 de octubre de 2018 en https://www.ecured.cu/EcuRed:Enciclopedia_cubana*

Profesor Java(s.f.). *Error de sintaxis.* [en línea]. Recuperado 17 noviembre, 2018, de <http://profejavaoramas.blogspot.com/2010/04/error-de-sintaxis.html>

Facultad de Ingeniería, UNAM. *Programación Orientada a objetos con Java* [en línea]. Recuperado el 8 de octubre de 2018 en http://profesores.fib.unam.mx/carlos/java/indice.html

Fundamentos de programación / Tipos de datos primitivos - Wikiversidad. [en línea]. Recuperado el 27 de octubre de 2018 de https://es.wikiversity.org/wiki/Fundamentos_de_programaci%C3%B3n/Tipos_de_datos_primitivos

Garro, A. [en línea]. Operadores | Java. Recuperado el 24 de octubre de 2018 de https://www.arkaitzgarro.com/java/capitulo-4.html

Henao, C. [en línea]. Conceptos Básicos de Programación Orientada a Objetos Recuperado el 24 de octubre de 2018 de http://codejavu.blogspot.com/2013/05/conceptos-de-programacion-orientada.html

Hernández, E. G. (s.f.). *Programación Java.* [en línea]. Recuperado el 11 de septiembre de 2018, de Tutorial Java. Aprende a programar con Java desde cero: http://puntocomnoesunlenguaje.blogspot.com/2012/07/clases-y-objetos-en-java.html

Holzner, S. (2000). *La Biblia de Java 2*. Anaya multimedia. Madrid.

Librerías De Java. [en línea]. Recuperado el 27 de octubre de 2018 de https://es.slideshare.net/aoteroe/libreras-de-java

Lima Díaz, F. (2010). *Manual avanzado de Java 6*. Madrid: Anaya Multimedia.

Munguia, E.(2019). *Los 10 errores más frecuentes en programación.* [en línea]. Recuperado 15 octubre, 2018, de <http://www.enrique7mc.com/2016/05/los-10-errores-mas-frecuentes-en-programacion/>

Programacion.net [en línea]. Recuperado el 8 de octubre de 2018 en https://programacion.net/articulo/programacion_orientada_a_objetos_279

Programación Orientada a Objetos/Características de la POO – Wikilibros [en línea]. Recuperado el 24 de octubre de 2018 en https://es.wikibooks.org/wiki/Programaci%C3%B3n_Orientada_a_Objetos/Caracter%C3%ADsticas_de_la_POO

Reservadas, J. [en línea]. Java: Identificadores y palabras reservadas. Recuperado el 24 de octubre de 2018 de http://puntocomnoesunlenguaje.blogspot.com/2012/04/identificadores-y-palabras-reservadas.html

Riquelme, J. (s.f.). Introducción a la programación. [en línea]. Recuperado 15 octubre, 2018, de <http://www.lsi.us.es/docencia/get.php?id=5960>

Sznajdleder, P. and Fernández, D. (2000). *Java a fondo*. Alfaomega Grupo Editor.

Vecteezy. (s.f.). [en línea]. Recuperado el 11 de septiembre de 2018, de Personas Vektor [Imágenes]: <https://de.vecteezy.com/vektorkunst/133402-personas-vektor>

# UNIDAD 2

## Estructuras de control de secuencia en


- Estructuras condicionales
- Estructura condicional multiple
- Estructura repetitiva for
- Estructura repetitiva while
- Estructura repetitiva do-while
- Arreglos unidimensionales
- Aplicación de arreglos unidimensionales
- Arreglos bidimensionales
- Aplicación de los arreglos bidimensionales
- Diseño de una aplicación

# Unidad 2. Estructuras de control de secuencia en Java

## Propósito:

Al finalizar la unidad el alumno:

Utilizará las estructuras de control de secuencia para la resolución de problemas a través del lenguaje de programación orientado a objetos con Java.

## Aprendizajes:

El alumno:

- Desarrolla programas que involucren las estructuras condicionales simples, compuestas y anidadas en los métodos de una Clase.
- Desarrolla programas que involucren la estructura condicional múltiple en los métodos de una Clase.
- Desarrolla programas para resolver problemas que involucren la estructura repetitiva for en los métodos de una Clase.
- Desarrolla programas que involucren la estructura repetitiva while en los métodos de una Clase.
- Desarrolla programas que involucren la estructura repetitiva do–while en los métodos de una Clase.
- Resuelve problemas que involucren el uso de los arreglos unidimensionales en los métodos de una Clase.
- Desarrolla programas que involucren el uso de los arreglos unidimensionales en los métodos de una Clase.
- Realiza programas que involucren el uso de los arreglos bidimensionales.
- Desarrolla programas que involucren el uso de los arreglos bidimensionales en los métodos de una Clase.
- Desarrolla un proyecto que utilice las sentencias vistas hasta el momento, incluyendo los arreglos.

## Introducción

En esta unidad utilizaremos las estructuras de control de secuencia para desarrollar programas a través del lenguaje de programación orientado a objetos Java.

Las estructuras de control de secuencia nos permiten cambiar el flujo de un programa, estas pueden ser condicionales o de repetición.

Las estructuras condicionales evalúan una expresión lógica, obteniendo un resultado verdadero o falso, de ello dependerá las acciones a seguir. Una expresión lógica se forma por medio de la comparación de valores haciendo uso de operadores relacionales y si es necesario, operadores lógicos.

La estructura condicional de selección simple es la sentencia *if*, mientras que para la selección múltiple se ocupa la sentencia *switch*.

Las estructuras de repetición, también conocidos como ciclos repetitivos o bucles, nos permiten repetir una serie de instrucciones. El ciclo *for* se utiliza cuando sabemos cuántas veces queremos repetir un ciclo, sin embargo, no siempre sabemos esto de antemano, por tanto, Java nos permite tener otro tipo de estructuras repetitivas como las sentencias *while* y d*o while* las cuales se realizan mientras una condición se cumple, una vez que esta deja de cumplirse el ciclo se termina.

Un arreglo es un tipo de dato estructurado, esto es, un conjunto de elementos del mismo tipo de datos que se almacenan bajo un mismo nombre y se diferencian por la posición que tiene cada uno dentro del mismo. Para declarar un arreglo se tiene que indicar su tipo, un nombre único y la cantidad de elementos que va a contener. Para ubicar a los elementos se utiliza un índice.

La dimensión de un arreglo se representa por el número de índices utilizados para referirse a un elemento particular en el arreglo, cuando solo se utiliza un índice, tenemos un arreglo unidimensional, conocido como vector.

Un arreglo bidimensional, representa conjuntos de datos ordenados en filas y columnas, También es conocido como una matriz**,** esto es**,** es un conjunto ordenado en una estructura de filas y columnas.

## 2.1 Estructuras condicionales

**Propósito**

El alumno describirá el funcionamiento de las estructuras condicionales simples.

- ## Sentencia condicional *if*

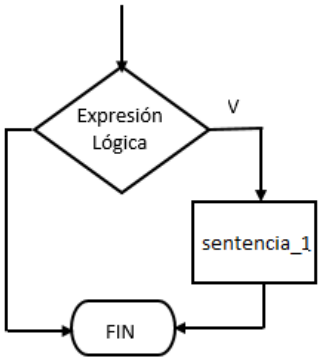
La sentencia *if* controla el flujo de un programa dependiendo de la evaluación de una condición dada, esto es, si la condición resulta verdadera se ejecutan ciertas instrucciones, si resulta falsa se ejecutan otras instrucciones.

Una condición es evaluada haciendo uso de los operadores relacionales (<, >, <=, >=, ==, =!) y operadores lógicos (&&, ||, !), revisados en la Unidad 1 *Tabla 1.2.8. Operadores de relación y Tabla 1.2.9. Operadores booleanos.*

- ## If Simple

La sentencia condicional *if* simple se utiliza cuando se va a llevar a cabo una acción después de evaluar la condición (expresión lógica) y esta resulta verdadera.
Su representación en diagrama de flujo y su sintaxis son:

Diagrama de flujo




Sintaxis

*if* (expresión Lógica)
sentencia_1;

***Figura 2.1.1. Diagrama de if***

**Ejemplo:**

Calcular el promedio de un alumno, solicitando por el teclado su nombre y tres calificaciones, para indicar si es un alumno Aprobado. Un alumno es aprobado si su promedio es mayor o igual a 6.

**Solución.**

Se retomará el ejemplo de la clase Exámenes visto en la Unidad 1, del tema 1.5.3 Métodos Implementación el cual ya obtiene el promedio de tres calificaciones. Agregando solamente la condición para que nos indique si es alumno Aprobado.

Por lo anterior, sólo se agrega la comparación del promedio y el mensaje como se muestra, el seudocódigo correspondiente a este fragmento es:

**Seudocódigo**

```
Si(promedio>=6)

  Escribir (n.getNombre( ) +" Es un alumn@ Aprobad@");
```

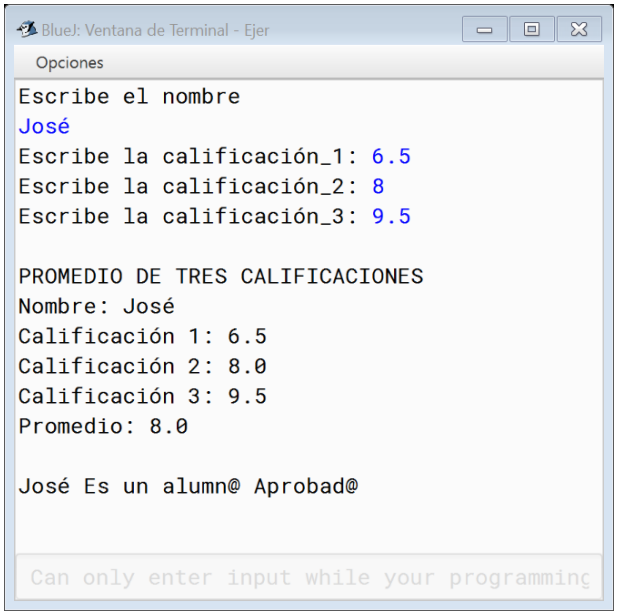
**Código – Desarrollo del programa**

```
import java.util.Scanner;

//se declara la clase Examenes
public class Examenes
{
   //se declaran los atributos
   private double calificacion;
   private String nombre;

   //se declara el método Getter del atributo
   public double getCalificacion( )
   {
      //se regresa el valor del atributo
      return calificacion;
```

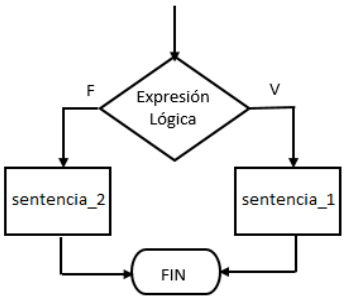
```
 }

 //se declara el método Setter del atributo
 public void setCalificacion(double numero)
 {
   //se modifica el valor del atributo
   this.calificacion=numero;
 }

 public String getNombre( )
 {
   return nombre;
 }

 //se declara el método Setter del atributo
 public void setNombre(String nombre)
 {
   //se modifica el valor del atributo
   this.nombre = nombre;
 }

//se declara el método main
 public static void main(String[ ] args)
 {
   //se declaran tres objetos de la clase
   Examenes n=new Examenes( );
   Examenes cal1=new Examenes( );
   Examenes cal2=new Examenes( );
   Examenes cal3=new Examenes( );

   Scanner teclado=new Scanner(System.in);

   //se actualizan los valores de los objetos con el método Setter
   System.out.println("Escribe el nombre");
   n.setNombre(teclado.nextLine( ));
   System.out.print("Escribe la calificación_1: ");
   cal1.setCalificacion(teclado.nextDouble( ));
   System.out.print("Escribe la calificación_2: ");
   cal2.setCalificacion(teclado.nextDouble( ));
   System.out.print("Escribe la calificación_3: ");
   cal3.setCalificacion(teclado.nextDouble( ));

   //se declara la variable promedio
   double promedio;
   //se calcula el promedio de los valores de los objetos
   promedio=(cal1.getCalificacion( )+cal2.getCalificacion( )+cal3.getCalificacion( ))/3;

   //mostramos en pantalla los valores de los objetos y el promedio
```

```
        System.out.println("\nPROMEDIO DE TRES CALIFICACIONES");
        System.out.println("Nombre: "+n.getNombre( ));
        System.out.println("Calificación 1: "+cal1.getCalificacion( ));
        System.out.println("Calificación 2: "+cal2.getCalificacion( ));
        System.out.println("Calificación 3: "+cal3.getCalificacion( ));
        System.out.println("Promedio: "+promedio+ "\n");

        //Utilizamos el if para definir si el alumno aprobó
        if (promedio>=6)
          System.out.println(n.getNombre( ) +" Es un alumn@ Aprobad@");    }
}
```

**Ejecución**




Asimismo, si es necesario se puede agregar la cláusula ***else*** la cual nos va a permitir ejecutar otra acción si la condición es falsa. El diagrama de flujo y sintaxis son:

| Diagrama de flujo | Sintaxis |
|---|---|
|   ***Figura 2.1.2. Diagrama de Estructuras condicionales*** | *if* (expresionLogica)<br>sentencia_1;<br>*else*<br>sentencia_2; |

**Ejemplo:**

Calcular el promedio de un alumno, solicitando por el teclado su nombre y tres calificaciones, para indicarle si es un alumno Aprobado o No Aprobado. Un alumno es aprobado si su promedio es mayor o igual a 6.

**Solución**

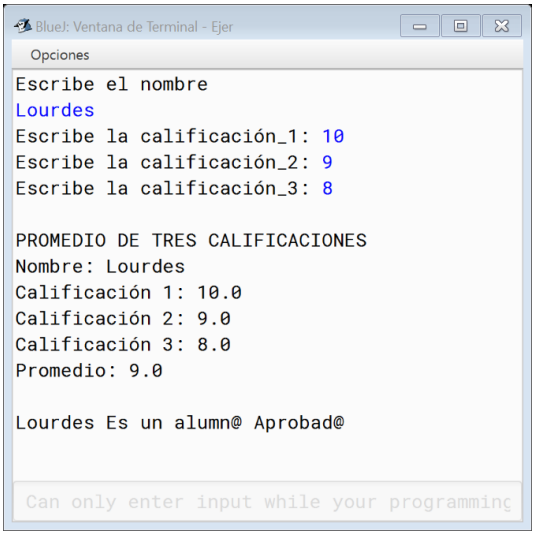
Se adaptará el programa del ejemplo anterior ya que solo le agregaremos un mensaje cuando no es aprobado. Su representación del segmento en seudocódigo es:

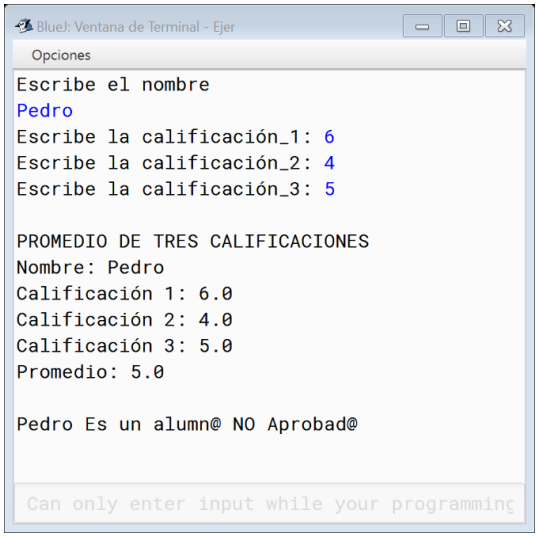
Seudocódigo

```
Si (promedio>=6)
        Escribir(n.getNombre( ) +" Es un alumn@ Aprobad@");
sino
        Escribir(n.getNombre( ) +" Es un alumn@ NO Aprobad@");
```

**Ejecución**

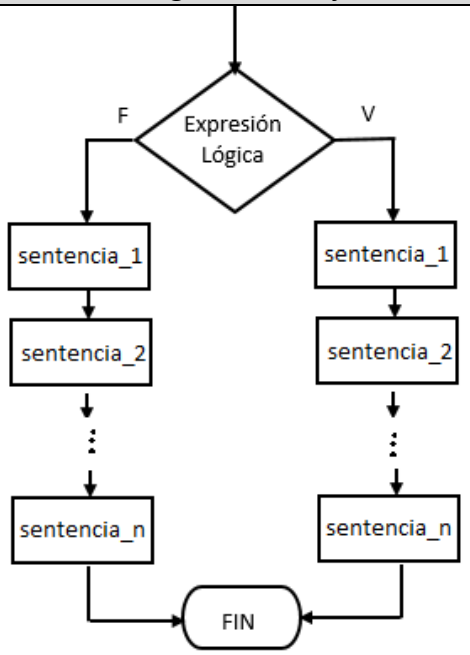
| Cuando la condición es verdadera | Cuando la condición es falsa |
|---|---|




- ***If* Compuesta**

La sentencia if es compuesta cuando al evaluar la condición se tienen que llevar a cabo una o más sentencias, según se cumpla si el valor de la sentencia es verdadero o falso. Esto es, se tiene que realizar un bloque de sentencias.

| Diagrama de flujo | Sintaxis |
|---|---|
|   ***Figura 2.1.3. Diagrama de Flujo de if compuesta*** | *if* (expresionLogica) **{**<br>sentencia_1;<br>sentencia_2;<br>⋮<br>Sentencia_n;<br>**}**<br>*else* **{**<br>sentencia_1;<br>sentencia_2;<br>⋮<br>Sentencia_n;<br>**}** |

Como se observa cada bloque está indicado por la apertura y cierre de llaves { }, esto es, cada bloque se diferencia porque se encuentra entre llaves.

**Ejemplo:**

Calcular el promedio de un alumno, solicitando por el teclado su nombre y tres calificaciones, para indicarle si es un alumno Aprobado o No Aprobado. Para cada caso indica un mensaje motivacional. Un alumno es aprobado si su promedio es mayor o igual a 6. En caso contrario estará reprobado.

**Solución.**

Para este caso seguiremos adaptando el programa anterior, sólo agregando la frase motivacional para cada caso. Por lo cual el segmento de seudocódigo es:

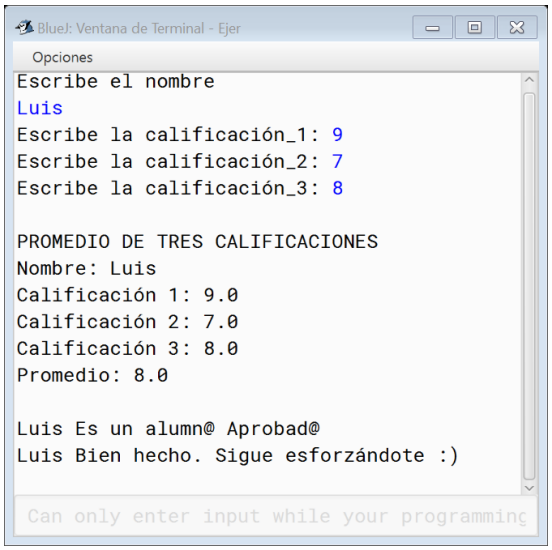
Seudocódigo

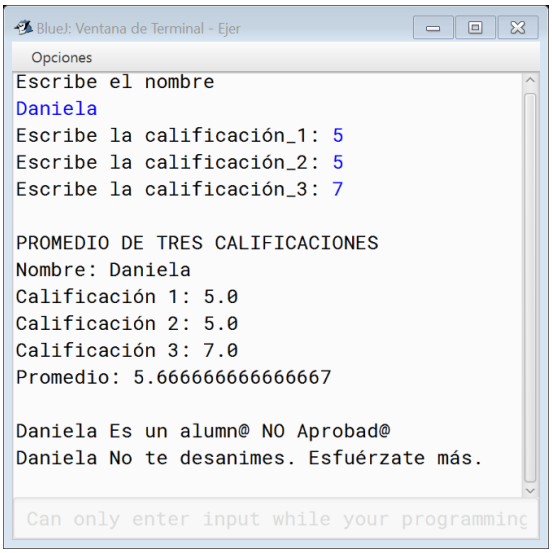
```
Si (promedio>=6) {
        Escribir(n.getNombre( ) +" Es un alumn@ Aprobad@");
        Escribir("Bien hecho. Sigue esforzándote :)");
}
sino {
        Escribir(n.getNombre( ) +" Es un alumn@ NO Aprobad@");
```

```
        Escribir(" No te desanimes. Esfuérzate más.");
    }
```

**Ejecución**

| Cuando la condición es verdadera | Cuando la condición es falsa |
|---|---|




- ***If* anidadas**

En cualquiera de los bloques de sentencias de la *if* compuesta se puede contar con otra sentencia *if* sea simple o compuesta, a esto se les llama *if* anidadas. El esquema de diagrama de flujo puede ser:

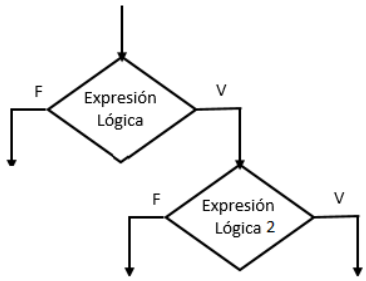



***Figura 2.1.4. Diagrama de Flujo de if anidadas***

**Ejemplo:**

Calcular el promedio de un alumno, solicitando por el teclado su nombre y tres calificaciones, para indicarle si es un alumno Aprobado o No Aprobado. Para cada caso indica un mensaje motivacional. Además, se le debe indicar si tiene derecho a una beca. Un alumno es aprobado si su promedio es mayor o igual a 6 y tiene derecho a beca si su promedio es mayor o igual a 9. Si el promedio es menor de 6 el alumno estará reprobado.

**Solución.**

Seguiremos adaptando el programa anterior, sólo agregando la condición para saber si tiene derecho a una beca. Por lo cual el segmento de seudocódigo es:

**Seudocódigo**

```
Si (promedio>=6){
        Escribir(n.getNombre( ) +" Es un alumn@ Aprobad@");
        Escribir("Bien hecho. Sigue esforzándote :)");
        //Sentencia if simple anidada
        Si (promedio>=9)
        Escribir("\n"+ n.getNombre( ) +" Tienes derecho a una beca");
}
 sino{
        Escribir(n.getNombre( ) +" Es un alumn@ NO Aprobad@");
        Escribir(" No te desanimes. Esfuérzate más.");
 }
```

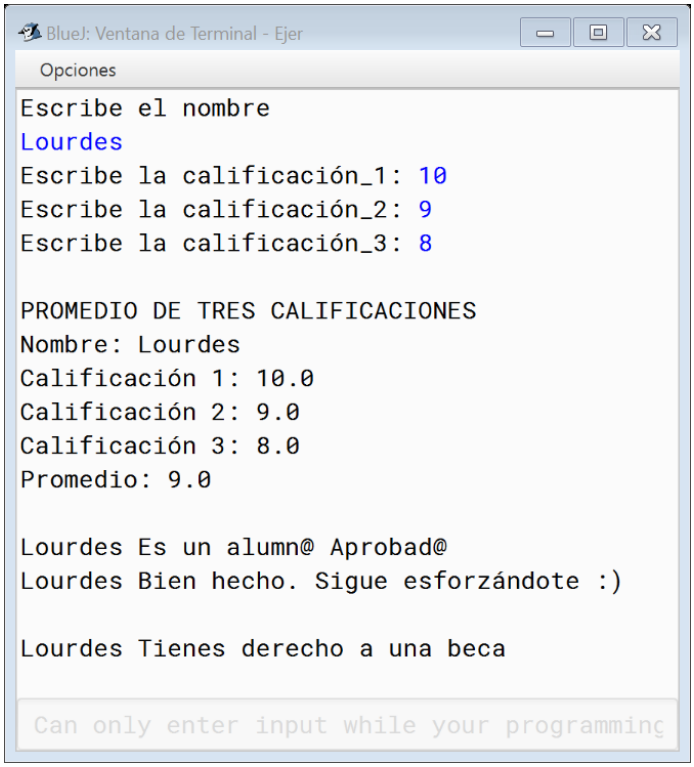
**Código – Desarrollo del Programa**

```
import java.util.Scanner;
//se declara la clase Examenes
public class Examenes
{
   //se declaran los atributos
   private double calificacion;
   private String nombre;

 //se declara el método Getter del atributo
   public double getCalificacion( )
   {
      //se regresa el valor del atributo
      return calificacion;
   }

 //se declara el método Setter del atributo
   public void setCalificacion(double numero)
   {
      //se modifica el valor del atributo
      this.calificacion=numero;
   }

   public String getNombre( )
   {
      return nombre;
   }

   //se declara el método Setter del atributo
```

```
  public void setNombre(String nombre)
  {
     //se modifica el valor del atributo
     this.nombre = nombre;
  }

//se declara el método main
  public static void main(String[ ] args)
  {
     //se declaran tres objetos de la clase
     Examenes n=new Examenes( );
     Examenes cal1=new Examenes( );
     Examenes cal2=new Examenes( );
     Examenes cal3=new Examenes( );

     Scanner teclado=new Scanner(System.in);

     //se actualizan los valores de los objetos con el método Setter
     System.out.println("Escribe el nombre");
     n.setNombre(teclado.nextLine( ));
     System.out.print("Escribe la calificaión_1: ");
     cal1.setCalificacion(teclado.nextDouble( ));
     System.out.print("Escribe la calificaión_2: ");
     cal2.setCalificacion(teclado.nextDouble( ));
     System.out.print("Escribe la calificaión_3: ");
     cal3.setCalificacion(teclado.nextDouble( ));

     //se declara la variable promedio
     double promedio;

      //se calcula el promedio de los valores de los objetos
     promedio=(cal1.getCalificacion( )+cal2.getCalificacion( )+cal3.getCalificacion( ))/3;

     //mostramos en pantalla los valores de los objetos y el promedio
     System.out.println("\nPROMEDIO DE TRES CALIFICACIONES");
     System.out.println("Nombre: "+n.getNombre( ));
     System.out.println("Calificación 1: "+cal1.getCalificacion( ));
     System.out.println("Calificación 2: "+cal2.getCalificacion( ));
     System.out.println("Calificación 3: "+cal3.getCalificacion( ));
     System.out.println("Promedio: "+promedio+ "\n");

     if (promedio>=6){
        System.out.println(n.getNombre( ) +" Es un alumn@ Aprobad@");
        System.out.println("Bien hecho. Sigue esforzándote :)");
        if (promedio>=9)
        System.out.println("\n"+ n.getNombre( ) +" Tienes derecho a una beca");
     }
```

```
    else{
      System.out.println(n.getNombre( ) +" Es un alumn@ NO Aprobad@");
      System.out.println(" No te desanimes. Esfuérzate más.");
    }
  }
}
```

**Ejecución**

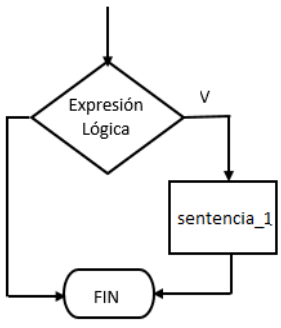



Actividad 2.1

1. Esta sentencia se utiliza cuando al evaluar la condición y su valor sea verdadero o falso se tienen que llevar a cabo una o más sentencias.

a) if simples b) if compuestas c) switch d) for

2. El siguiente diagrama de flujo representa la estructura de control.




a) if simples b) if compuestas c) switch d) for

3. En una sentencia esta cláusula nos va a permitir ejecutar otra acción si la condición es falsa.

a) case b) else c) break d) if

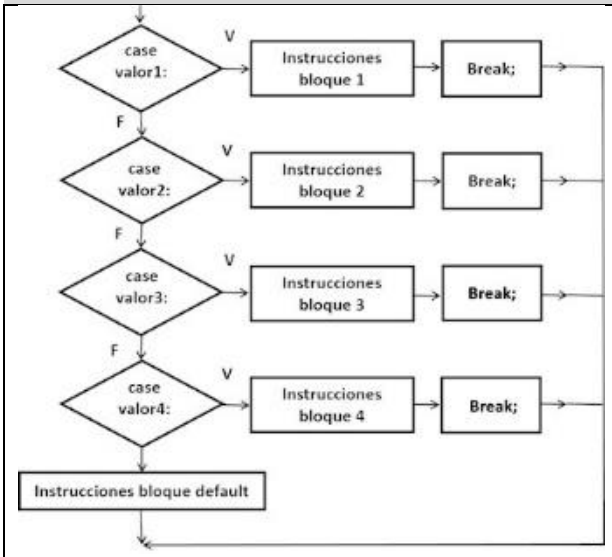
## 2.2 Estructura condicional múltiple

**Propósito**

El alumno comprende el funcionamiento de las estructuras condicionales múltiples.

- ***Switch***

La sentencia *switch* es una estructura condicional múltiple que permite seleccionar una entre varias alternativas. La sentencia switch evalúa una expresión, el resultado lo compara con cada valor consecutivamente, hasta encontrar uno que coincida. Cuando coincide un valor, se ejecutan las instrucciones de esa cláusula.

| Diagrama de flujo | Sintaxis |
|---|---|
|  ***Figura 2.1. Diagrama de estructura condicional múltiple*** | switch (<expresión>) {<br>case <valor 1>:<br><sentencias separadas por punto y coma>;<br>break;<br>case <valor 2>:<br><sentencias separadas por punto y coma>;<br>break;<br>…<br>default:<br><sentencias separadas por punto y coma>;<br>} |

En donde:

- El tipo de variable de la <expresión> debe ser de tipo entero (int) o caracter (char).
- El tipo de variable de la <expresión> y el <valor> deben coincidir.
- La cláusula *default* es opcional. Se utiliza en caso de que ninguno de los valores coincida con el proporcionado o el resultado de la expresión.
- Puede haber tantas cláusulas *case* como se requiera.
- El <valor> no puede ser una expresión, sólo puede ser un literal.

- Los valores duplicados de los *case* no están permitidos.
- La declaración *break* se usa dentro del *switch* para finalizar una secuencia de instrucción, esto es, para el *case* de un <valor>. Si se omite, la ejecución continuará en el siguiente *case*.

**Ejemplo:**

Realizar un programa que solicite dos números, introducidos desde el teclado, y nos permita elegir una de las cuatro operaciones básicas (suma, resta, multiplicación y división) mostrando un menú. Si se elige una opción diferente, nos indicará que la opción no es válida.

**Código – Desarrollo del programa**

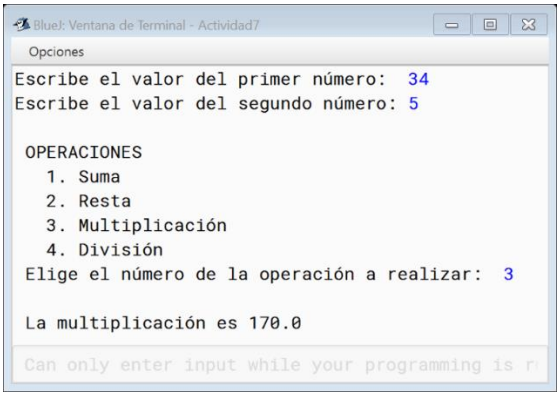
```
import java.util.Scanner;

// Se declara la clase Operaciones
public class Operaciones {

   //Se declaran los atributos
   private double numero1 = 0.0;
   private double numero2 = 0.0;
   private double resultado = 0.0;
   private int opcion = 0;

   //Se declaran los métodos Getter y Setter para cada atributo

   public double getNumero1( ){
         return numero1;
   }

   public void setNumero1(double numero1){
         this.numero1 = numero1;
   }

   public double getNumero2( ){
         return numero2;
   }

   public void setNumero2(double numero2){
         this.numero2 = numero2;
   }

   public int getOpcion( ){
         return opcion;
   }

   public void setOpcion(int opcion){
         this.opcion = opcion;
   }
```

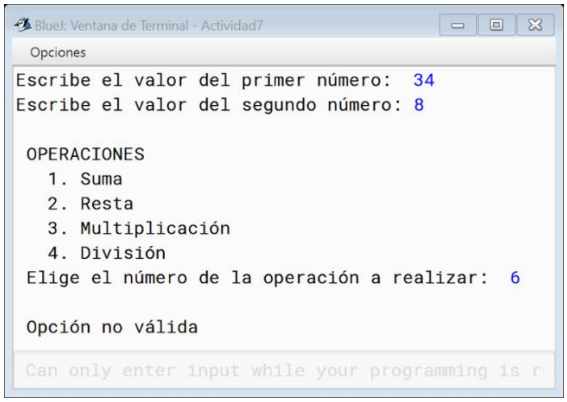
```
   //Se realizan los métodos para cada operación
public double Suma( ){
      resultado = numero1 + numero2;
      return resultado;
}

public double Resta( ){
      resultado = numero1 - numero2;
      return resultado;
}

public double Multiplicacion( ){
      resultado = numero1 * numero2;
      return resultado;
}

public double Division( ){
      if (numero2 != 0)
         resultado = numero1 / numero2;
      else
         System.out.println("No es posible realizar la división");
      return resultado;
}
// Se declara el método Opciones
public void Opcion( ){
      System.out.println("\n OPERACIONES");
      System.out.println("   1. Suma");
      System.out.println("   2. Resta");
      System.out.println("   3. Multiplicación");
      System.out.println("   4. División");
      System.out.print(" Elige el número de la operación a realizar:  ");
}
   public static void main(String[ ] args) {
   Operaciones oper = new Operaciones( );
   Scanner teclado=new Scanner(System.in);

   //se actualizan los valores de los atributos con el método Setter
   System.out.print("Escribe el valor del primer número:  ");
   oper.setNumero1(teclado.nextDouble( ));
   System.out.print("Escribe el valor del segundo número: ");
   oper.setNumero2(teclado.nextDouble( ));
   oper.Opcion( );  //muestra las opciones
   oper.setOpcion(teclado.nextInt( )); //se ingresa alguna de las 4 opciones
   int opc= oper.getOpcion( );

   //Se realiza la operación seleccionada en opc
   switch (opc) {
    case 1: //opción 1 es la suma
       System.out.println("\n La suma es "+ oper.Suma( ));
       break;
```

```
        case 2: //opción 2 es la resta
          System.out.println("\n La resta es "+ oper.Resta( ));
          break;
        case 3: //opción 3 es la multiplicacion
          System.out.println("\n La multiplicación es "+ oper.Multiplicacion( ));
          break;
        case 4: //opción 4 es la división
          System.out.println("\n La división es "+ oper.Division( ));
          break;
        default: //ninguna de las 4 opciones
           System.out.println("\n Opción no válida");
      }

   }
}
```

**Ejecución**

| Cuando la opción es válida | Cuando la opción No es válida |
|---|---|




## Actividad 2.2

1. Es una estructura condicional múltiple que permite seleccionar una entre varias alternativas:

a) if simples b) if compuestas c) switch d) for

2. En una sentencia *switch* el tipo de variable de la <expresión> debe ser de tipo:

a) int b) String c) boolean d) float

3. Esta cláusula es opcional para la sentencia switch.

a) case b) default c) break d) expresión

4. En la sentencia switch, esta declaración se usa para finalizar una secuencia de instrucción.

a) case b) default c) break d) expresión

## 2.3 Estructura repetitiva for

**Propósito**

El alumno describirá el funcionamiento de la estructura repetitiva for.

- ***for***

La sentencia *for* es una estructura de control cíclica, nos permite ejecutar una o varias líneas de código de forma iterativa (repetitiva), teniendo cierto control y conocimiento sobre las iteraciones. Para utilizarlo, es necesario tener un valor inicial y un valor final, y opcionalmente podemos hacer uso del tamaño del "paso" (incremento o decremento) entre cada "giro" (repetición).

Diagrama de flujo

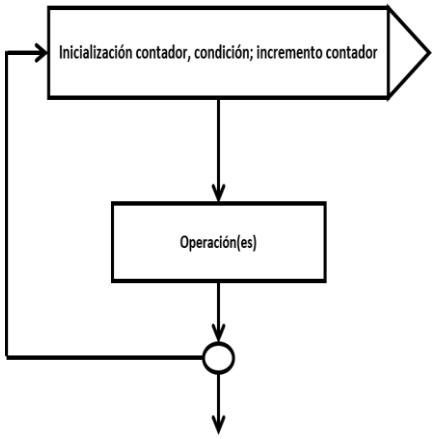



***Figura 2.3.1. Diagrama Estructura repetitiva for***

Sintaxis

*For (Valor inicial; condición*; iteración por ciclo*)*
*{*
*Instrucciones que vamos a ejecutar mientras se cumpla la condición;*
*}*

```
for (int i = valor inicial; i <= valor final; i = i + paso)
{
    ....
    ....
  Bloque de Instrucciones....
    ....
    ....
}
```

En donde:

- Valor inicial especifica el valor previo a su inicio.
- La condición debe estar directamente relacionada con el valor inicial.

La iteración por ciclo indica cómo serán manipulados los valores iniciales en cada repetición del ciclo, esto es, el tamaño del paso. En el tamaño del paso se pueden hacer uso de los operadores aritméticos unitarios (i++ incremento o i-- decremento) vistos en la Unidad 1 tema Operadores *Tabla 1.2.6. Operadores aritméticos incrementales.*

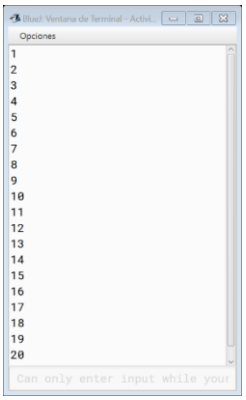
Es importante contemplar el valor de inicio, la condición de término y el tamaño del paso o incremento, ya que si no llegan a coincidir se vuelve un ciclo indeterminado el cual no se ejecutará o se puede volver un ciclo sin término.

**Ejemplo:**

Realizar un programa que muestre una cuenta sucesiva del 1 al 20

| Programa | Ejecución |
|---|---|

```
public class CicloForIncremento
{
   public static void main(String args[ ])
   {
      for(int i=1; i<=20; i++)
      {
         System.out.println(i);
      }
   }
}
```




Nota: El tamaño del paso puede escribirse con diferente notación como:

i++
i+=1
i=i+1

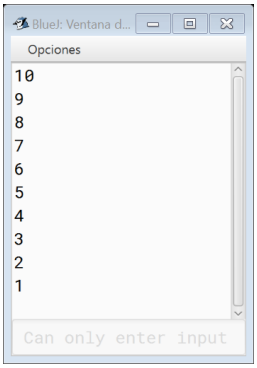
Distinguiendo que el incremento es de uno en uno. Si se quiere que el incremento del paso sea diferente a uno se utilizará la segunda o tercera notación indicando el número del tamaño del paso.

**Ejemplo:**

Ahora realiza un programa que muestre una cuenta regresiva del 10 al 1.

| **Programa** | **Ejecución** |
|---|---|

```
public class CicloForDecremento
{
   public static void main(String args[ ])
   {
      for(int i=10;i>=1;i--)
      {
         System.out.println(i);
      }
   }
}
```




*Nota*. Aplican las mismas notaciones que en incremento sólo cambiando el operador + por -

**Implementación**

Calcular el promedio de un alumno, solicitando por el teclado el nombre del alumno y el número de calificaciones para obtener su promedio.

**Solución**

Utilizaremos un ciclo ***for*** para solicitar las calificaciones indicadas y con ello se vayan acumulando.

Esto es:

**Seudocódigo**

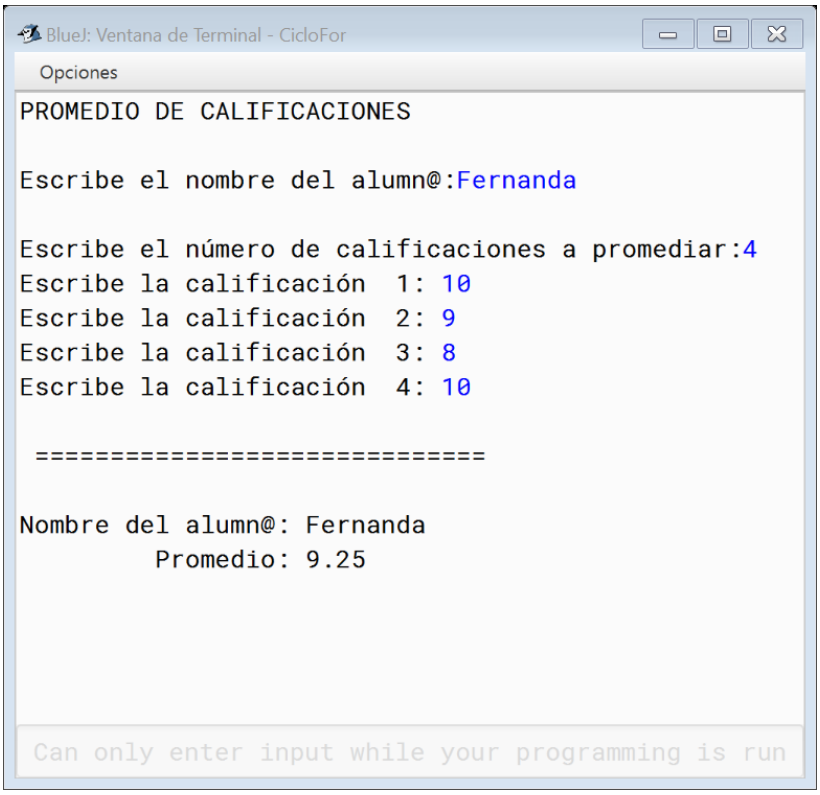
```
Para (int i=1; i<=numCal; i++){
        Escribir("Escribe la calificación  " + i + ": ");
        calif=teclado.nextDouble( ); // asignando el valor de la calificación al atributo cal
        suma+=calif;            // acumulando las calificaciones (calif) en la variable suma
}
```

**Código**

```
import java.util.Scanner;

//se declara la clase Examenes
public class Examenes
{
   //se declaran los atributos
   private double calif, suma, prom;
   private String nombre;
   private int numCal;

   //se declara el método Getter del atributo
   public String getNombre( )
   {
      return nombre;
   }

   //se declara el método Setter del atributo nombre
   public void setNombre(String nombre)
   {
      //se modifica el valor del atributo
      this.nombre = nombre;
   }

   public int getNumCal( )
   {
      return numCal;
   }

   //se declara el método Setter del atributo número de calificaciones
   public void setNumCal(int numCal)
   {
      //se modifica el valor del atributo
      this.numCal = numCal;
   }
```

```
  //Se declara el método para solicitar las calificaciones y sumarlas
  public void sumaCalificaciones( )
  {
     Scanner teclado=new Scanner(System.in);
     for (int i=1; i<=numCal; i++){
         System.out.print("Escribe la calificación  " + i + ": ");
         calif=teclado.nextDouble( ); // asignando el valor de la calificación
         suma+=calif;              // acumulando las calificaciones
     }
  }

  // Se hace el método para obtener el promedio
  public double promedio( )
  {
     prom=suma/numCal;
     return(prom);
  }

  //se declara el método main
  public static void main(String[ ] args)
  {
     //se declaran el objeto de la clase
     Examenes exa=new Examenes( );

     Scanner teclado=new Scanner(System.in);

     //Se solicitan datos
     System.out.println("PROMEDIO DE CALIFICACIONES");
     System.out.print("\nEscribe el nombre del alumn@: ");
     exa.setNombre(teclado.nextLine( ));
     System.out.print("\nEscribe el número de calificaciones a promediar: ");
     exa.setNumCal(teclado.nextInt( ));
     exa.sumaCalificaciones( );
     // Se muestran resultados
     System.out.println("\n ============================\n");
     System.out.println("Nombre del alumn@: "+exa.getNombre( ));
     System.out.println("        Promedio: "+exa.promedio( ));
  }
}
```

**Ejecución.**




Actividad 2.3

1. Es una estructura de control para la que es necesario tener un valor inicial y un valor final.

a) if simples b) if compuestas c) switch d) for

2. Este operador aritmético se utiliza para incrementar en uno.

a) i++ b) i-- c) i- d) i+

3. La siguiente sentencia realiza una cuenta regresiva de 10 al 1.

a) for(int i=10;i>=1;i--) b) for(int i=10;i>=1;++) c) for(float i>10;i=1;i--) d) for(int i=10;i>=1)

## 2.4 Estructura repetitiva: while.

**Propósito**

El alumno desarrollará programas que involucren la estructura repetitiva while.

- **Ciclo while**

Para iniciar con el ciclo while primero vamos a visualizar la forma en la cual se implementa mediante la ayuda del diagrama de flujo, recuerda que este tipo de diagrama es la representación gráfica de un algoritmo.

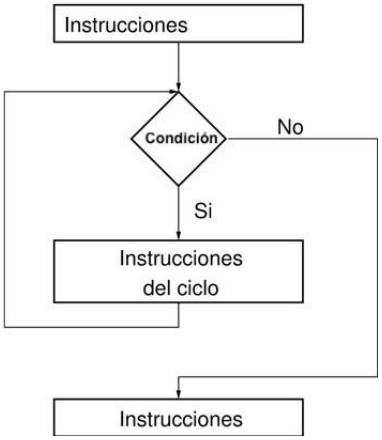



***Figura. 2.4.1. Diagrama de flujo del ciclo while***.

De acuerdo con el diagrama de flujo anterior, el funcionamiento del ciclo *while* se desarrolla de la siguiente manera: primeramente, antes de entrar al ciclo, se verifica la condición de este. En caso de que sea verdadera, entonces se ingresa al ciclo y se llevan a cabo las instrucciones correspondientes, repitiéndose hasta que la condición deje de ser verdadera, en ese momento el ciclo se termina y se continúa con la ejecución de las demás instrucciones que conforman al programa.

Es muy importante mencionar que, si la condición inicialmente es falsa, no se entra al ciclo, por lo cual este no se ejecuta ni una sola vez y se continúa con las siguientes instrucciones del programa.

También es pertinente mencionar que, si la condición del ciclo siempre permanece como verdadera, entonces este se ejecutará indefinidamente sin poder terminarse, lo cual se conoce como un ciclo infinito y por tanto el programa no podrá finalizar.

**Sintaxis**

Para crear un ciclo *while* en Java utilizamos la siguiente estructura:

```
Inicializar variables
while (condicion) {
        Instrucciones;
}
```

### Ejemplo 1

Para ejemplificar lo anterior comencemos con un programa sencillo, utilicemos el ciclo *while* para escribir una frase un determinado número de veces. Para ello utilizaremos el siguiente código:

```
import java.util.Scanner;

//Creamos la clase

public class EjemploWhile
{
  //Creamos el método main
  public static void main(String[ ] args) {
      Scanner teclado=new Scanner(System.in);
      int n,x;
      //Preguntamos cuantas veces se quiere escribir la frase
      System.out.print("¿Cuántas veces quieres escribir la frase 'Hola Mundo'");
      //El número de veces se almacena en la variable n
      n=teclado.nextInt( );
      //Se inicializa la variable x que sirve como contador
      x=1;
      //Se inicia el ciclo
      while (x<=n) {  //mientras x sea menor o igual a n
        System.out.println(x+".- Hola Mundo");
        x = x + 1;
      }
   }
}
```

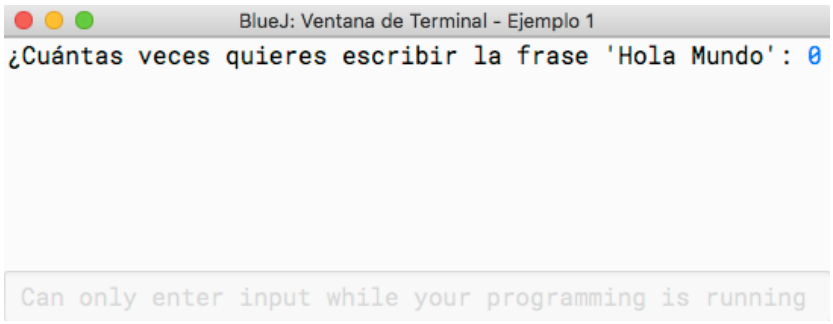
- La inicialización de la variable x = 1 y la estructura del ciclo están resaltadas en negritas.
- La variable n la utilizamos para almacenar el número de veces que el usuario desea repetir el mensaje en la pantalla.
- La variable x es un contador que se encuentra dentro del ciclo el cual se incrementa mediante x=x+1, cada vez que escribe en la pantalla la frase “Hola Mundo”.

Hagamos una prueba de escritorio para los siguientes valores:

1) El usuario elige que no se escriba la frase en pantalla, por lo cual ingresa el valor de 0.

| n | x | Condición | Valor | Salida |
|---|---|---|---|---|
| 0 | 1 | x<=n | 1<=0<br>**Falso** | Dado la condición no es verdadera, el ciclo no se ejecuta y no se despliega ningún mensaje |

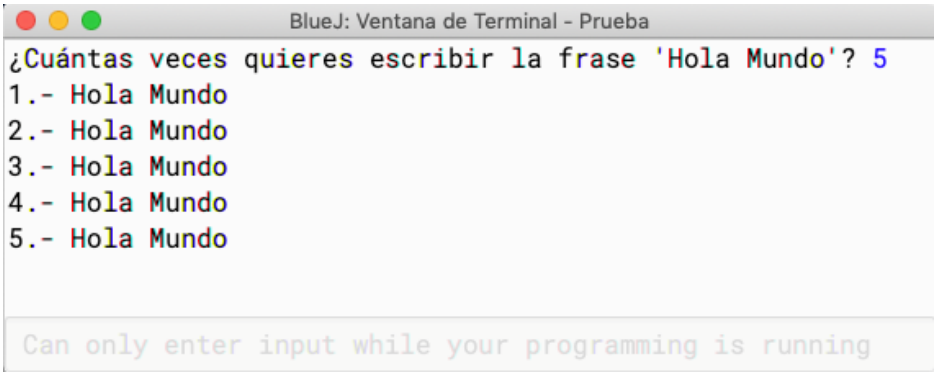
Si ejecutamos el programa, la salida será la siguiente:




2) El usuario quiere que el mensaje se muestre 5 veces, por lo cual ingresa el valor este valor para posteriormente almacenarse en la variable n.

| n | x | Condición | Valor | Salida |
|---|---|---|---|---|
| 5 | 1 | x<=n | 1<=5 **Verdadero** | Dado que la condición es verdadera, se ingresa al cuerpo del ciclo y se despliega el mensaje, posteriormente se incrementa en uno la variable x mediante la instrucción: x=x+1; es decir x= 1+1 = 2 |
| 5 | 2 | x<=n | 2<=5 **Verdadero** | Se despliega el mensaje, posteriormente se incrementa en uno la variable x, por tanto: x=3 |
| 5 | 3 | x<=n | 3<=5 **Verdadero** | Se despliega el mensaje, posteriormente se incrementa en uno la variable x, por tanto: x=4 |
| 5 | 4 | x<=n | 4<=5 **Verdadero** | Se despliega el mensaje, posteriormente se incrementa en uno la variable x, por tanto: x=5 |
| 5 | 5 | x<=n | 5<=5 **Verdadero** | Se despliega el mensaje, posteriormente se incrementa en uno la variable x, por tanto: x=6 |
| 5 | 6 | x<=n | 6<=5 **Falso** | Al no ser verdadera la condición, el ciclo se termina. |

La salida del programa es:



## Ejemplo 2

Retomando el programa que calcula el promedio de tres calificaciones visto anteriormente, se implementará ahora con un ciclo ***while***, de tal forma que después de mostrar los resultados, el programa deberá preguntar si desea repetir el cálculo del promedio, para ello se debe presionar la tecla s, mientras que, si se desea salir del programa se deberá presionar cualquier otra tecla.

Se inicia importando la clase Scanner y declarando la clase Examenes y sus atributos:

```
import java.util.Scanner;
//Se declara la clase Examenes
public class Examenes {
  //se declaran los atributos
  private double calif1, calif2, calif3;
  private String nombre;
```

Posteriormente se declaran los setters y getters de los atributos de la clase:

```
//Se declaran los métodos Getter de los atributos de la clase
  public double getCalif1( ) {
    return calif1;
  }
  public double getCalif2( ) {
    return calif2;
  }
  public double getCalif3( ) {
    return calif3;
  }
  public String getNombre( ) {
    return nombre;
  }

//se declaran los métodos Setter de los atributos de la clase
  public void setCalif1 (double calif1) {
    this.calif1=calif1;
  }
   public void setCalif2 (double calif2) {
    this.calif2=calif2;
  }
  public void setCalif3 (double calif3) {
    this.calif3=calif3;
  }
  public void setNombre (String nombre) {
    this.nombre = nombre;
  }
```

Se declara el método principal, un objeto de la clase Scanner, un objeto de la clase exámenes asi como las variables que se van a utilizar en el método principal:

```
//Se declara el metodo principal
  public static void main(String[ ] args) {
    //Se declara el objeto teclado de la clase Scanner
    Scanner teclado = new Scanner(System.in);
    //Se declara el objeto datos de la clase Scanner
    Examenes datos = new Examenes( );
    //Se declaran las variables del metodo principal
    double calif, promedio;
    String nombre;
    //Se declara la variable salir y tmp para terminar con el ciclo while
    String salir, tmp;
```

Se declara la variable que controla la ejecución del ciclo while y una variable temporal para limpiar el cache, adicionalmente se inicializa la variable para que la condición sea verdadera y se pueda ejecutar el ciclo while.

```
    //Se declara la variable salir y tmp para terminar con el ciclo while
    String salir, tmp;
    /*Se inicializa la variable salir igual a "s" para que la condición
    *del ciclo while sea verdadera y se pueda ingresar a él
    */
    salir = "s";
```

Se inicia con el ciclo while:

```
while (salir.equals("s")){
```

Se le pregunta al usuario por tres calificaciones las cuales se almacen en los atributos de la clase:

```
      //Se preguntan por las calificaciones y se almacenan en los atributos de la clase
      System.out.print('\u000C');
      System.out.println("Este programa calcula el promedio de tres calificaciones");
      System.out.print("Escribe el nombre del alumno: ");
      nombre=teclado.nextLine( );
      datos.setNombre(nombre);
      System.out.print("Ingresa la primer calificación: ");
      calif=teclado.nextDouble( );
      datos.setCalif1(calif);
      System.out.print("Ingresa la segunda calificación: ");
      calif=teclado.nextDouble( );
      datos.setCalif2(calif);
      System.out.print("Ingresa la tercer calificación: ");
      calif=teclado.nextDouble( );
      datos.setCalif3(calif);
```

Se calcula el promedio y se muestran resultados, sí el promedio es mayor o igual a 6 se menciona que la calificación es aprobatoria, si es mayor o igual a 9 se indica que tiene derecho a una beca y en caso contrario se indica que la calificación es no aprobatoria :

```
      //Se calcula el promedio
      promedio=(datos.getCalif1( )+ datos.getCalif2( )+ datos.getCalif3( ))/3;
      //Se muestran los resultados en la pantalla
      System.out.println("\nRESULTADO DEL PROMEDIO DE TRES CALIFICACIONES");
      System.out.println("Nombre: "+datos.getNombre( ));
      System.out.println("Calificación 1: "+datos.getCalif1( ));
      System.out.println("Calificación 2: "+datos.getCalif2( ));
      System.out.println("Calificación 3: "+datos.getCalif3( ));
      System.out.println("Promedio: "+promedio+ "\n");
      //Si el promedio es mayor o igual a 6 se le indica que es aprobado
      if (promedio>=6){
        System.out.println(datos.getNombre( ) +" Eres un alumn@ Aprobad@");
        System.out.println("Bien hecho, ¡Sigue esforzándote!");
      //Si además el promedio es mayor o igual a 9 se le dice que tiene beca
        if (promedio>=9)
          System.out.println(datos.getNombre( ) +", además ¡tienes derecho a una beca!");
      }
      //En caso contrario se le indica que su calificación no es aprobatoria
      else{
        System.out.println(datos.getNombre( ) +" Eres un alumn@ NO Aprobad@");
        System.out.println("No te desanimes, ¡Esfuérzate más!");
      }
```

Finalmente, se le pregunta al usuario si desea terminar o continuar con el ciclo:

```
      //Se le indica la condición para salir o permanecer en el ciclo
      System.out.println("Presione la tecla s para repetir el programa o cualquier otra para salir");
      //Se limpia el buffer de entrada (teclado)
      tmp=teclado.nextLine( );
      salir=teclado.nextLine( );
      }
  }
}
```

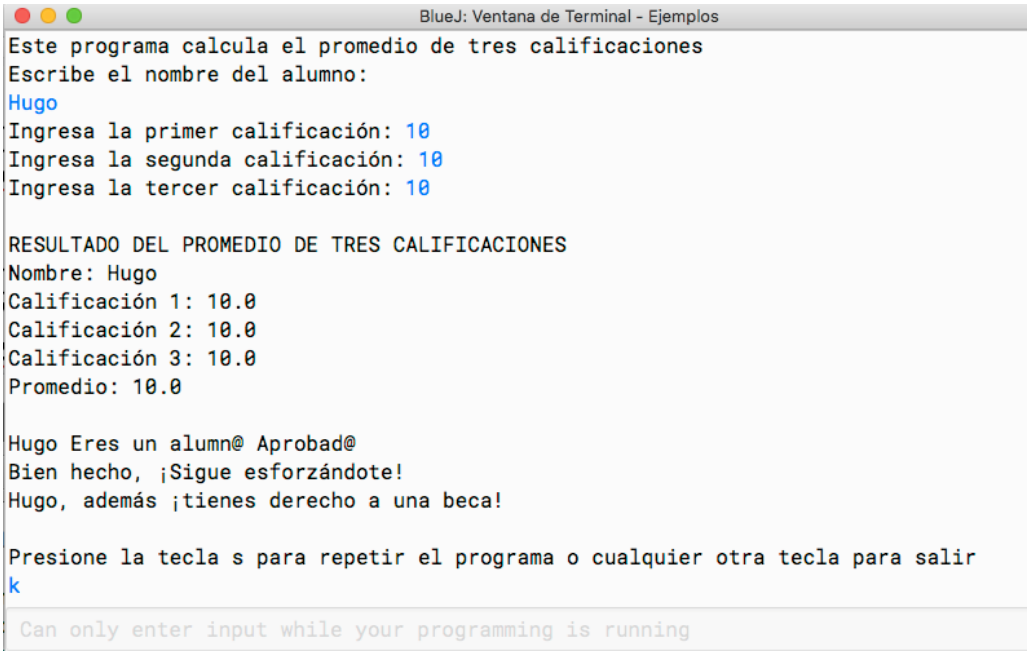
El código completo es el siguiente:

```
import java.util.Scanner;
//Se declara la clase Examenes
public class Examenes {
  //se declaran los atributos
  private double calif1, calif2, calif3;
  private String nombre;
  //Se declaran los métodos Getter de los atributos de la clase
  public double getCalif1( ) {
    return calif1;
  }
  public double getCalif2( ) {
    return calif2;
```

```
}
public double getCalif3( ) {
  return calif3;
}
public String getNombre( ) {
  return nombre;
}
//se declaran los métodos Setter de los atributos de la clase
public void setCalif1 (double calif1) {
  this.calif1=calif1;
}
public void setCalif2 (double calif2) {
  this.calif2=calif2;
}
public void setCalif3 (double calif3) {
  this.calif3=calif3;
}
public void setNombre (String nombre) {
  this.nombre = nombre;
}
//Se declara el metodo principal
public static void main(String[ ] args) {
  //Se declara el objeto teclado de la clase Scanner
  Scanner teclado = new Scanner(System.in);
  //Se declara el objeto datos de la clase Scanner
  Examenes datos = new Examenes();
  //Se declaran las variables del metodo principal
  double calif, promedio;
  String nombre;
  //Se declara la variable salir y tmp para terminar con el ciclo while
  String salir, tmp;
  /*Se inicializa la variable salir igual a "s" para que la condición
  *del ciclo while sea verdadera y se pueda ingresar a él
  */
  salir = "s";
  while (salir.equals("s")){
    //Se preguntan por las calificaciones y se almacenan en los atributos de la clase
    System.out.print('\u000C');
    System.out.println("Este programa calcula el promedio de tres calificaciones");
    System.out.print("Escribe el nombre del alumno: ");
    nombre=teclado.nextLine( );
    datos.setNombre(nombre);
    System.out.print("Ingresa la primer calificación: ");
    calif=teclado.nextDouble();
    datos.setCalif1(calif);
    System.out.print("Ingresa la segunda calificación: ");
    calif =teclado.nextDouble();
    datos.setCalif2(calif);
```

```
      System.out.print("Ingresa la tercer calificación: ");
      calif =teclado.nextDouble();
      datos.setCalif3(calif);
      //Se calcula el promedio
      promedio=(datos.getCalif1( )+ datos.getCalif2( )+ datos.getCalif3( ))/3;
      //Se muestran los resultados en la pantalla
      System.out.println("\nRESULTADO DEL PROMEDIO DE TRES CALIFICACIONES");
      System.out.println("Nombre: "+datos.getNombre( ));
      System.out.println("Calificación 1: "+datos.getCalif1( ));
      System.out.println("Calificación 2: "+datos.getCalif2( ));
      System.out.println("Calificación 3: "+datos.getCalif3( ));
      System.out.println("Promedio: "+promedio+ "\n");
      //Si el promedio es mayor o igual a 6 se le indica que es aprobado
      if (promedio>=6){
        System.out.println(datos.getNombre( ) +" Eres un alumn@ Aprobad@");
        System.out.println("Bien hecho, ¡Sigue esforzándote!");
      //Si además el promedio es mayor o igual a 9 se le dice que tiene beca
        if (promedio>=9)
          System.out.println(datos.getNombre( ) +", además ¡tienes derecho a una beca!");
      }
      //En caso contrario se le indica que su calificación no es aprobatoria
      else{
        System.out.println(datos.getNombre( ) +" Eres un alumn@ NO Aprobad@");
        System.out.println("No te desanimes, ¡Esfuérzate más!");
      }
      //Salto de línea
      System.out.println("");
      //Se le indica la condición para salir o permanecer en el ciclo
      System.out.println("Presione la tecla s para repetir el programa o cualquier otra para salir");
      //Se limpia el buffer de entrada (teclado)
      tmp=teclado.nextLine( );
      salir=teclado.nextLine( );
      }
  }
}
```

Salida del programa:




## Actividad 2.4

1) El ciclo while se ejecuta al evaluar una condición, aunque esta sea falsa desde un inicio.
   a) Verdadero b) Falso

2) El ciclo while se ejecutará indefinidamente hasta que la condición sea verdadera.
   a) Verdadero b) Falso

3) El ciclo while requiere tener una condición inicial
   a) Verdadero b) Falso

4) Si la condición siempre permanece verdadera se produce lo que se conoce como ciclo infinito.
   a) Verdadero b) Falso

5) En el cuerpo del ciclo while debe existir una variable que cambie de valor para que la condición del ciclo pueda pasar de verdadera a falsa.
   a) Verdadero b) Falso

## 2.5 Estructura repetitiva: do while.

**Propósito**

El alumno desarrollará programas que involucren la estructura repetitiva do–while.

### 2.5.1. Ciclo do while

En este tipo de ciclo, a diferencia del anterior, las instrucciones se ejecutan en primer lugar y posteriormente se evalúa la condición para continuar o terminar con el ciclo, si esta condición es verdadera el ciclo se sigue ejecutando, mientras que, si la condición del ciclo es falsa, entonces este se termina.

Veamos la representación de este ciclo mediante el diagrama de flujo, recuerda que este es la representación gráfica de un algoritmo.

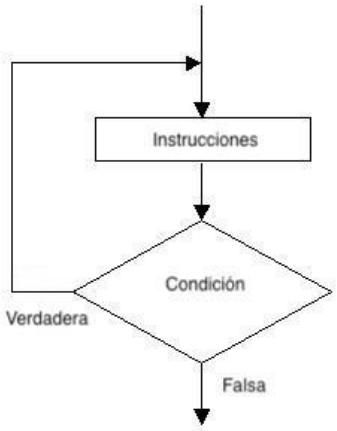



***Figura 2.5.1. Diagrama de flujo del ciclo do while***.

Observa que las instrucciones siempre se van a ejecutar al menos una vez y posteriormente se evalúa la condición, si esta es falsa entonces el ciclo termina y se continua con la ejecución del programa, en caso contrario las instrucciones se vuelven a ejecutar y el ciclo continúa de forma indefinida hasta que la condición deje de ser verdadera.

Al igual que en el ciclo anterior *while*, es importante tener en cuenta que la condición debe ser en algún momento falsa, ya que, de no cumplirse con esto, entonces se entraría en un ciclo infinito y por tanto el programa no podrá finalizar.

### Sintaxis

Para crear un ciclo *do while* en Java utilizamos la siguiente estructura:

```
Inicializar variables
do {
      instrucciones
    } while (condición);
```

**Ejemplo 1**

Hagamos un primer ejemplo, en este vamos a imprimir un número aun cuando la condición del ciclo sea falsa de primera entrada, con ello podremos observar que las instrucciones del ciclo se pueden ejecutar al menos una vez, aunque esta condición no sea verdadera.

En el ejemplo el valor inicial de la variable x es igual a 11 mientras que la condición para terminar con el ciclo es que x sea igual o menor a 10, como puede observar la condición es falsa, pero a pesar de ello las instrucciones (imprimir el número en pantalla) se ejecuta una vez.

```
//Creamos la clase
class Ejemplo_1DoWhile
 {
   //Creamos el método principal
    public static void main(String args                              )
    {
      //Se inicializa la variable x
      int x = 11;
      //Inicia el ciclo
      do
      {
        //El valor de la variable se imprime al menos una vez aun cuando la condición es falsa
          System.out.println("Valor de x :" + x);
          x++;
      }
      //Condición para terminar con el ciclo
      while (x < 10);
     }
  }
```

La salida del programa es la siguiente:


**Ejemplo 2**

Vamos a realizar un programa el cual va a calcular la suma y el promedio de todos los valores que se hayan ingresado desde el teclado, para terminar de ingresar valores se debe escribir el número 0.

Se importa la clase Scanner, se declara la clase principal y sus atributos:

```
import java.util.Scanner;
//Se crea la clase principal
public class EjemploDoWhile {
private double suma=0.0;
private double promedio=0.0;
```

Se crean los getters de los atributos:

```
//Se crean los getters de los atributos
public double getSuma( ) {
  return suma;
}
public double getPromedio( ) {
  return promedio;
}
```

Se crean los setters de los atributos:

```
public void setSuma(double suma) {
  this.suma=suma;
}
public void setPromedio(double promedio) {
  this.promedio = promedio;
}
```

Se crea el método principal asi como las instancias de las clases utilizadas y las varibles del método principal:

```
  //Se crea el método principal
  public static void main(String[ ] args) {
  //Se crea un objeto de la clase Scanner y otro de la clase principal
  Scanner teclado=new Scanner(System.in);
  EjemploDoWhile elemento = new EjemploDoWhile( );
  //Se crean e inicializan variables del método principal
  double valor, suma, cantidad;
  cantidad = 0.0;
  suma = 0.0;
```

Se muestra en pantalla la utilidad del programa y las instrucciones:

```
//Se muestra en pantalla la finalidad del programa
  System.out.print('\u000C');
  System.out.println("Este programa suma los números que se ingresan consecutivamente,");
  System.out.println("para terminar de ingresar números se debe ingresar el numero 0."+"\n");
  System.out.println("Al final se muestra la suma y el promedio de todos los números que se
                      ingresarón."+"\n");
```

Se inicia el ciclio do – while y posteriormente se comienzan a ingresar los valores:

```
//Inicia el ciclo Do - While
```

```
do {
  System.out.println("Ingrese un valor para sumarlo y posteriormente encontrar el promedio:");
  valor=teclado.nextDouble( );
```

Si el valor es diferente de 0 este se va almacenando en el atributo suma, el ciclo continua mientras el valor que se ingrese sea diferente de 0:

```
/*Si el valor es diferente de 0, suma es igual a suma más el valor que se ingresó.
    cantidad se incrementa en uno.*/
    if (valor != 0.0) {
      suma = elemento.getSuma( ) + valor;
      elemento.setSuma(suma);
      cantidad++;
     }
   }
   //El ciclo continua mientras el valor sea diferente de 0.
   while (valor != 0.0);
```

Si se ingresaron valores diferentes de 0, se muestra la suma y el promedio:

```
if (cantidad != 0.0) {
    elemento.setPromedio(elemento.getSuma( )/cantidad);
    System.out.println("La suma de los valores ingresados es: "+elemento.getSuma( ));
    System.out.println("El promedio de los valores ingresados es: "+elemento.getPromedio( ));
    System.out.println();
  }
```

Si el primer valor que se ingreso fue cero se muestra el mensaje de error y se termina con el programa:

```
  //Si no se ingreso ningun elemento
  else {
    System.out.println("No se ingresaron valores.");
  }
 }
}
```

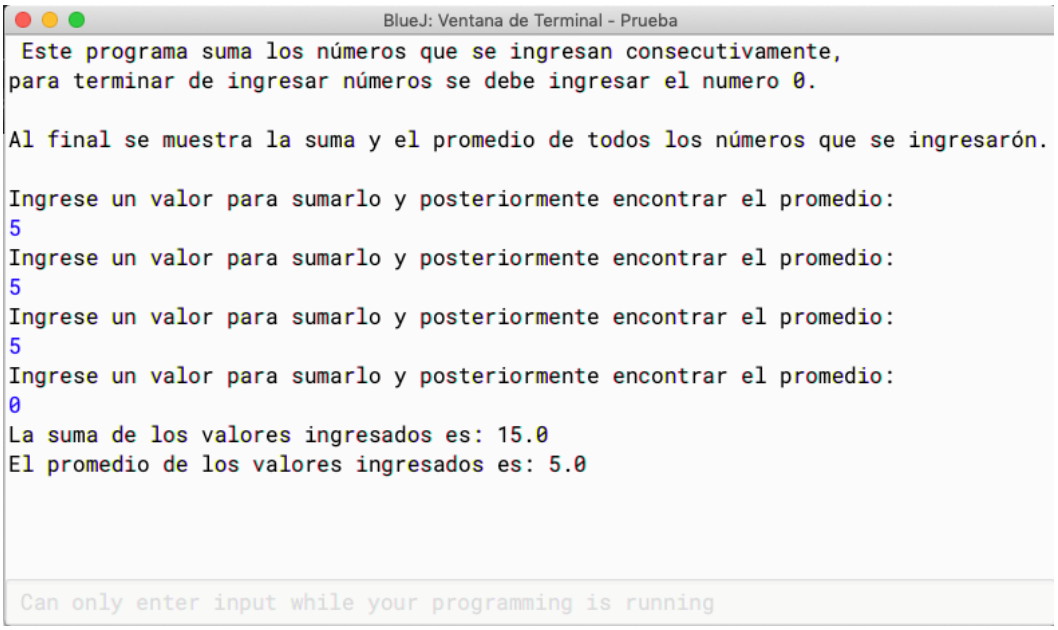
El código completo del programa es el siguiente:

```
import java.util.Scanner;
//Se crea la clase principal
public class EjemploDoWhile {
private double suma=0.0;
private double promedio=0.0;
//Se crean los getters de los atributos
public double getSuma( ) {
  return suma;
}
```

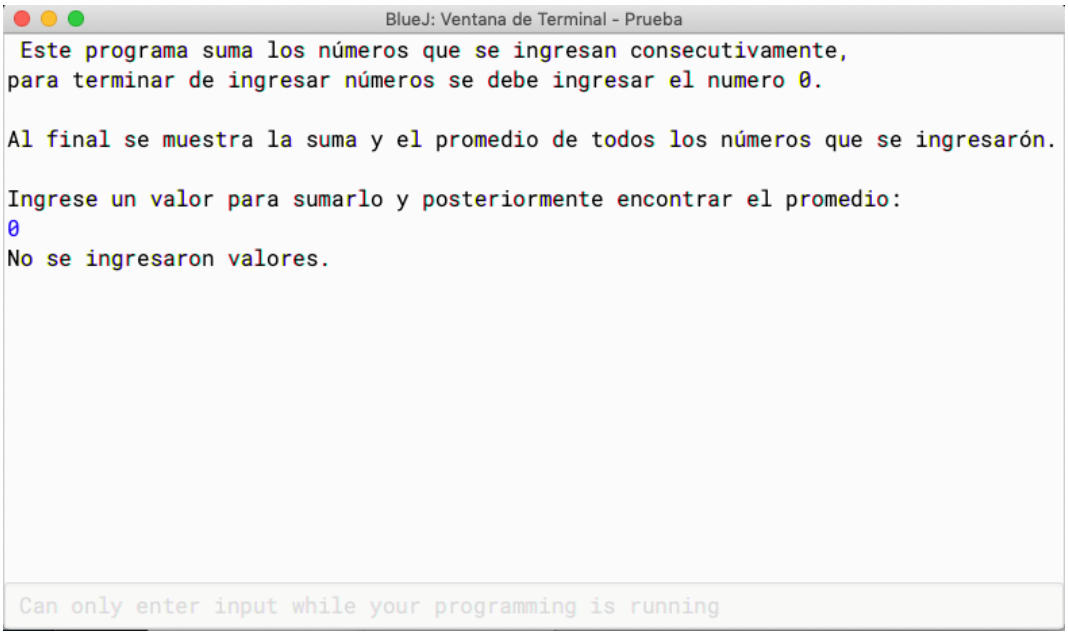
```
public double getPromedio( ) {
  return promedio;
}
//Se crean los setters de los atributos
public void setSuma(double suma) {
  this.suma=suma;
}
public void setPromedio(double promedio) {
  this.promedio = promedio;
}
  //Se crea el método principal
  public static void main(String[ ] args) {
  //Se crea un objeto de la clase Scanner y otro de la clase principal
  Scanner teclado=new Scanner(System.in);
  EjemploDoWhile elemento = new EjemploDoWhile( );
  //Se crean e inicializan variables del método principal
  double valor, suma, cantidad;
  cantidad = 0.0;
  suma = 0.0;
  //Se muestra en pantalla la finalidad del programa
  System.out.print('\u000C');
  System.out.println("Este programa suma los números que se ingresan consecutivamente,");
  System.out.println("para terminar de ingresar números se debe ingresar el numero 0."+"\n");
  System.out.println("Al final se muestra la suma y el promedio de todos los números que se
                      ingresarón."+"\n");

 //Inicia el ciclo Do - While
  do {
    System.out.println("Ingrese un valor para sumarlo y posteriormente encontrar el promedio:");
    valor=teclado.nextDouble( );
    /*Si el valor es diferente de 0, suma es igual a suma más el valor que se ingresó.
    Cantidad se incrementa en uno.*/
    if (valor != 0.0) {
      suma = elemento.getSuma( ) + valor;
      elemento.setSuma(suma);
      cantidad++;
    }
   }
   //El ciclo continua mientras el valor sea diferente de 0.
   while (valor != 0.0);

 //Si la cantidad de numeros ingresados fueron diferentes a cero
  if (cantidad != 0.0) {
    elemento.setPromedio(elemento.getSuma( ) / cantidad);
    System.out.println("La suma de los valores ingresados es: " + elemento.getSuma( ));
    System.out.println("El promedio de los valores ingresados es: " + elemento.getPromedio( ));
    System.out.println( );
  }
```

```
  //Si no se ingreso ningun elemento
  else {
    System.out.println("No se ingresaron valores.");
  }
 }
}
```

La salida del programa cuando se ingresan 5, 5, 5, es la siguiente:




Salida del programa cuando no se ingresa ningún valor:



## Actividad 2.5

1. Se requiere realizar un programa que nos diga cuando un número es mayor o igual a 20 o en caso contrario cuando es menor a 20. El programa se repite indefinidamente hasta que se ingresa el número 0. A continuación encontraras el programa en bloques desordenados es necesario que los ordenes de forma correcta.

| | |
|---|---|
| public static void main(String[ ] ar) {<br>Scanner teclado=new Scanner(System.in);<br>int valor;<br>valor=0; | 1 |
| if (valor==0){<br>System.out.println("Se termina el programa");<br>} | 2 |
| while (valor!=0);<br>}<br>} | 3 |
| else {<br><br>if (valor > 20) {<br>System.out.println("El número "+ valor +" es mayor o igual que 20");<br>}<br>else{<br>System.out.println("El número "+ valor +" es menor que 20");<br>}<br>}<br>} | 4 |
| do {<br>System.out.println("Ingrese un número para saber si es mayor que 20");<br>System.out.println("Si desea terminar el programa ingrese el número 0");<br>valor=teclado.nextInt( ); | 5 |
| import java.util.Scanner;<br><br>public class Ejemplo_3 { | 6 |

Escribe la secuencia correcta: _____

## 2.6 Arreglos unidimensionales

**Propósito:**

El alumno comprenderá qué son los arreglos unidimensionales, cómo se declaran, los tipos que existen, y su aplicación en la resolución de problemas desarrollando programas que involucren los métodos de una Clase.

- **Definición**

Un arreglo es un grupo de elementos que contienen valores del mismo tipo. Para hacer referencia a un elemento en particular debemos especificar el nombre del arreglo y el número de la posición del elemento, el cual se conoce como índice y debe ser un número entero positivo. En la figura 2.6.1 se muestra un arreglo de diez elementos en el que se indica el primero con índice cero, el mayor índice es 9, y su tamaño es 10.

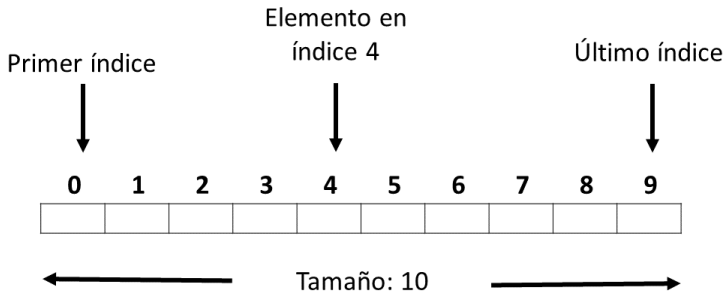



***Figura 2.6.1. Arreglo unidimensional.***

Un arreglo es unidimensional, también llamado vector, porque es una lista de elementos que emplea un solo índice.

- **Declaración**

La sintaxis para crear un arreglo es la siguiente:

*tipo_de_dato[ ] nombre_del_Arreglo;*

Donde *tipo_de_dato* puede ser cualquier tipo de dato, después se ponen corchetes, a continuación, el identificador con el nombre del arreglo, para más adelante, hacer referencia a él durante el desarrollo del programa.

Además, se puede declarar el arreglo con los corchetes después del nombre:

*tipo_de_dato nombre_del_Arreglo[ ];*

En el siguiente ejemplo se declara un arreglo de enteros:

```
public class Arreglos
{
   public static void main(String[ ] args)
   {
      int[ ] numeros; //arreglo de enteros llamado números.
   }
}
```

También podemos declarar un arreglo como un objeto con la palabra reservada **new**, especificando el tipo de dato de cada elemento y el número de elementos que se requieren para el arreglo. La sintaxis es la siguiente:

*tipo_de _dato nombre_del_arreglo[ ] = new tipo_de_dato[número_de_elementos];*

La siguiente sentencia crea un arreglo que contiene 10 elementos de tipo entero (*int*) y de nombre calificaciones:

*int calificaciones[ ] = new int[10];*

En la declaración, los corchetes que van después del nombre indican que el identificador *calificaciones* es una variable que hará referencia a un arreglo de valores enteros. Al crear un arreglo, cada uno de sus elementos recibe un valor predeterminado el cual es cero para los elementos numéricos.

- **Tipos**

Como ya se mencionó, un arreglo se declara indicando el tipo de dato y éste puede ser cualquiera de los admitidos por Java, estos se explicaron en la primera unidad de este material, de esta manera vamos a tener diferentes tipos de arreglos como se muestra en la tabla 2.6.1.

Los valores por defecto de cada tipo de arreglo, son los siguientes:

a) Para números el valor cero “0”.
b) Para cadenas y letras el valor vacío.
c) Para booleanos el valor false.

- **Uso**

Un arreglo es la forma más simple de agrupar componentes de un mismo tipo y asociarles un orden a través de un índice, por lo que son utilizados para manejar conjuntos de elementos con alguna característica en común.

Los arreglos representan, en programación, a los conceptos matemáticos de matrices y vectores. Una gran motivación para usar arreglos es que hay mucha teoría detrás de ellos, que puede ser utilizada en el diseño de algoritmos para resolver problemas verdaderamente interesantes.

Las acciones para manipular y usar un arreglo son la inicialización, esto es guardar valores en el arreglo, y el acceso o lectura de los elementos. En seguida, se explica cómo se emplea cada uno.

**Inicialización de un arreglo**

Para poder utilizar un arreglo, después de declararlo, es necesario enlistar sus elementos, existen dos maneras de poder hacerlo:

1. Con una lista inicializadora encerrada entre llaves, donde la longitud del arreglo se determina en base al número de elementos enlistados. Por ejemplo, la siguiente sentencia declara un arreglo de tipo entero y tamaño 5:

*int n[ ] = { 10, 20, 30, 40, 50 };*

En la siguiente tabla podemos ver más ejemplos de inicialización según el tipo de arreglo.

| Tipo de arreglo | Ejemplo de declaración e inicialización |
|---|---|
| byte[ ] | byte[ ] edad = {10,11,12,13,14,15}; |
| short[ ] | short[ ] anio = {1980, 1990, 2001}; |
| int[ ] | int[ ] precio = {11589, 14987, 16324, 20549}; |
| long[ ] | long[ ] valor = {1597, 2547, 3654, 4589}; |
| float[ ] | float[ ] peso = {50.0, 55.500, 61.0, 62.300}; |
| double[ ] | double[ ] estatura = {0.90, 1.10, 1.30, 1.50}; |
| boolean[ ] | boolean[ ] estado = {true, false}; |
| char[ ] | char[ ] sexo = {‘F’, ‘M’}; |
| String[ ] | String[ ] nombre = {“Hugo”, “Paco”, “Luis”}; |

***Tabla 2.6.1. Tipos de arreglos e inicialización.***

2. Declarando cada elemento del arreglo empleando el índice entre los corchetes. Por ejemplo, para declarar el arreglo anterior *n* se emplearían las siguientes sentencias:

```
int n[ ] = new int[5];
int n[0]=10;
int n[1]=20;
int n[2]=30;
int n[3]=40;
int n[4]=50;
```

- **Acceso a los elementos de un arreglo**

Vamos a acceder a un arreglo cuando leamos los valores guardados y esto se puede hacer de dos maneras:

1. Accediendo a cada elemento de forma separada indicando el valor del índice entre corchetes como se muestra a continuación:

```
int n[ ] = { 10, 20, 30, 40, 50 };

System.out.println(n[3]);
```

La sentencia anterior imprime el elemento del arreglo ubicado en la posición 3, esto es el valor 40.

2. Mostrando todos los elementos del arreglo de forma dinámica con la sentencia de control *for,* utilizándola como ya se explicó en temas anteriores:

```
int n[ ] = { 10, 20, 30, 40, 50 };

for(int i = 0; i < 5; i++)
{
      System.out.println(n[i]);
}
```

Podemos emplear el atributo de tamaño del objeto arreglo (n.*length*) en la sentencia *for* y así fijar el valor final del ciclo.

```
int n[ ] = { 10, 20, 30, 40, 50 };

//se utiliza el atributo length del objeto arreglo
for(int i = 0; i < n.length; i++)
{
  //se imprime el valor del arreglo n en la posición i
  System.out.println(n[i]);
}
```

## 2.7 Aplicación de arreglos unidimensionales

Vamos a utilizar arreglos unidimensionales para resolver el siguiente problema:

Desarrollar un programa que lea n nombres de alumnos, 3 calificaciones de cada uno de ellos, que calcule el promedio y finalmente imprima una tabla con los resultados, en donde se indique si el alumno aprobó o reprobó, el promedio para aprobar debe ser mayor o igual a 7.0.

**Seudocódigo.**

Inicio
Declarar Variables;

```
Declarar arreglos;
Leer variable de número de alumnos;
Leer variable de calificaciones;
promedio[i] = ( calif1[i] + calif2[i] + calif3[i] )/3;
Si(promedio[i] > 7)
        estatus[i] = “Aprobado”
Sino
        estatus[i] = “Reprobado”
Escribir “ nombre,  calificacion1,  calificacion2,  calificacion3,  promedio,  estatus ”
Fin
```

- ## Acceso

Acceder a un arreglo significa obtener la información almacenada en sus elementos y podemos acceder a un elemento por medio de un índice, por ejemplo, queremos acceder a la información contenida en el primer elemento tendremos que hacerlo de esta manera:

*int dato = arreglo[0];*

En la variable entera *dato* se almacenará el valor guardado en arreglo[0].

El número cero describe la posición del elemento dentro del arreglo.

- ## Recorrido

Recorrer un arreglo significa acceder a cada uno de los elementos, para esto se utiliza el nombre del arreglo y un índice que indica la posición que ocupa el elemento dentro del mismo.

Como ya hemos mencionado el primer elemento ocupa la posición 0, el segundo la posición 1, etc. En un arreglo de N elementos el último ocupará la posición N-1.

En los arreglos, a menudo necesitamos recorrer todos sus índices, ya sea para buscar un valor en concreto o para mostrar todos los valores que almacena. Esta acción por lo general se realiza mediante algún ciclo repetitivo, el más adecuado es el ciclo *for*.

En este ejemplo tomaremos los arreglos que hemos utilizado: nombre, calif1, calif2, calif3, promedio, estatus, los vamos a recorrer para mostrar todos y cada uno de los valores que contienen.

- ## Implementación

Ahora vamos a aplicar las acciones de acceso y recorrido además del llenado de arreglos.

Para ello vamos a desarrollar un programa que lea un número N de alumnos y 3 calificaciones de cada uno de ellos, después el programa debe calcular el promedio y asignar un estatus de “Aprobado” si el promedio es mayor o igual a 7 o “Reprobado” si es menor de 7. Finalmente se debe imprimir una tabla con los resultados.

A continuación, vamos a explicar los métodos que componen a la clase principal y la acción que realizan, finalmente se mostrara el programa completo.

*Almacenamiento de datos en un arreglo.*

El método que se muestra a continuación realizará la acción de leer ***n*** nombres de alumnos y los almacenará en un arreglo de tamaño ***n***, el valor de la variable ***n*** pasará a este método de almacenamiento como un parámetro desde el método principal. El código correspondiente es el siguiente:

```
//Método para leer los nombres y almacenarlos en un arreglo.
   public String[ ] pedirNombres(int n)
   {
      Scanner teclado=new Scanner(System.in);
      // Se declara el arreglo
      String nombre[ ]=new String[n];
      // Se almacenan los nombres de los alumnos en el arreglo nombre
      for(int i=0; i<n; i++)
      {
         System.out.print("Ingresa el nombre " + (i+1) + ": " );
         nombre[i] = teclado.nextLine( );
      }
      // Regresa el arreglo de nombres
      return(nombre);
    }
```

Después de haber almacenado los nombres de los alumnos en el arreglo nombre, **accederemos** a este arreglo, esto es **recorremos** uno por uno los elementos del arreglo y traemos el nombre del alumno y capturamos una calificación a la vez, por lo que utilizaremos el siguiente método para almacenar, una por una, las calificaciones de los alumnos. Los datos o parámetros de entrada son el arreglo ***nombre*** y el valor ***n*** que es el tamaño del arreglo, regresa un arreglo con las calificaciones.

```
/* Método para leer una calificación de cada uno de los alumnos y las almacena
     en un arreglo.
     Se utilizará para leer una calificación de cada uno de los alumnos.
*/
// Parámetros de entrada el arreglo nombre y el valor entero n que es el tamaño del arreglo
   public double[ ] pedirCalif(String[ ] nombre,int n)
   {
      Scanner teclado=new Scanner(System.in);
      double [ ] calif=new double[n];
       // Recorre el arreglo nombres y se almacena una calificación en el arreglo calif
      for(int i=0;i<n;i++)
      {
         System.out.print("==> "+nombre[i]+":");
         calif[i]=teclado.nextDouble( );
      }
```

```
    //Regresa el arreglo calif
    return(calif);
  }
```

El siguiente método realizará la impresión de los datos y los resultados esperados, esto es, imprimirá los nombres de los alumnos y las tres calificaciones de cada uno, así como el promedio y el estatus, este método será invocado desde el método principal, ahí explicaremos como se obtienen los datos.

```
// Método para imprimir datos, promedio y estatus
/*Parámetros de entrada, el arreglo nombre y los arreglos de calificaciones calif1, calif2 y calif3, y
  El tamaño del arreglo n */
  public void imprimeDatos(String[ ] nombre,double calif1[ ], double calif2[ ],double calif3[ ], int n)
  {
     double promedio[ ] = new double[n];
     String estatus[ ] = new String[n];
     /*Accesa a los elementos de los arreglos nombre y calif1, calif2, calif3,
       calcula el promedio y define el estatus */
     for(int i=0;i<n;i++)
     {
        promedio[i] = ( calif1[i] + calif2[i] + calif3[i] )/3;
        if(promedio[i] >=7 )
        {
           estatus[i] = "Aprobado";
        }else{
           estatus[i] = "Reprobado";
        }
        String cadena=nombre[i];
        /*Calcula el número de espacio con los que complementará el arreglo nombre
        para escribir la tabla en forma ordenada */
        int largo=cadena.length( );
        int espacios=15-largo;
        // Imprime la tabla
        System.out.print((i+1) + ": " + nombre[i]);
        for(int k=0;k<=espacios;k++)
        {
           System.out.print(" ");
        }
        System.out.print(calif1[i] + "  "
        + calif2[i] + "  " + calif3[i] + "\t" + String.format("%.2f", promedio[i]) + "\t\t" +
estatus[i]+"\n");
     }
  }
```

A continuación, se muestra el método principal del programa, aquí es en donde se llaman o invocan a los métodos anteriores, recordando la estructura de un programa, primero se codifican los métodos y después el método principal, a continuación, se describe el código correspondiente:

```
  public static void main(String[ ] args)
   {
      Scanner teclado=new Scanner(System.in);
      //instanciamos el objeto p de la clase promedios
      Promedios p=new Promedios( );
      /* Limpiamos la pantalla */
      System.out.print('\u000C');
      System.out.print("¿Cuantos alumnos calificarás? ");
      int n=teclado.nextInt( );
      String nombres[ ]=new String[n];
      //Se leen los nombres y se almacenan en el arreglo nombre
      nombres=p.pedirNombres(n);
      //Se lee la calificación 1  y se almacena en el arreglo calif1
      System.out.println("Primera calificación:");
      double calif1[ ]=new double[n];
      // Invocamos al metodo perdirCalif y guardamos las calificaciones en el arreglos calif1
      calif1=p.pedirCalif(nombres, n);
      //Se lee la calificación 2  y se almacena en el arreglo calif2
      System.out.println("Segunda calificación:");
      double calif2[ ]=new double[n];
      // Invocamos al metodo perdirCalif y guardamos las calificaciones en el arreglos calif2
      calif2=p.pedirCalif(nombres, n);
      //Se lee la calificación 3  y se almacena en el arreglo calif3
      System.out.println("Tercera calificación:");
      double calif3[ ]=new double[n];
      // Invocamos al metodo perdirCalif y guardamos las calificaciones en el arreglos calif3
      calif3=p.pedirCalif(nombres, n);
      //Se imprime el encabezado de la tabla
      System.out.println("\n\n\t\t\t TABLA DE CALIFICACIONES");
System.out.println("_____________________________________________________________________");
      System.out.println("Nombre" + "\t\t  Calificaciones\t" + "Promedio\t" + " Estatus");
      System.out.println("_____________________________________________________________________");
      //Impresión de la tabla
      p.imprimeDatos(nombres,calif1,calif2,calif3,n);
   }
}
```

A continuación, se muestra el código completo del programa y los resultados después de ejecutarlo.

```
/*Programa que lee N nombres de alumnos y 3 calificaciones de cada uno
  Regresa el promedio de cada uno y el estatus de aprobado o reprobado
  Si el promedio es mayor o igual a 7
*/

import java.util.Scanner;
```

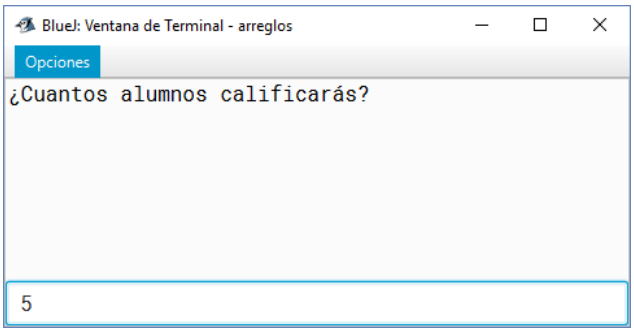
```
public class Promedios
{
   //Método para leer los nombres y almacenarlos en un arreglo.
   public String[ ] pedirNombres(int n)
   {
      Scanner teclado=new Scanner(System.in);
      String nombre[ ]=new String[n];
      for(int i=0;i<n;i++)
      {
         System.out.print("Ingresa el nombre " + (i+1) + ": " );
         nombre[i] = teclado.nextLine( );
      }
      return(nombre);
   }
   /* Método para leer una calificación de cada uno de los alumnos y las almacena
     en un arreglo.
     Se utilizará para leer las tres calificaciones.
       */
   public double[ ] pedirCalif(String[ ] nombre,int n)
   {
      Scanner teclado=new Scanner(System.in);
      double [ ] calif=new double[n];
      for(int i=0;i<n;i++)
      {
         System.out.print("==> "+nombre[i]+":");
         calif[i]=teclado.nextDouble( );
      }
      return(calif);
   }
   // Método para imprimir datos, promedio y estatus
   public void imprimeDatos(String[ ] nombre,double calif1[ ], double calif2[ ],double calif3[ ],
int n)
   {
      double promedio[ ] = new double[n];
      String estatus[ ] = new String[n];
      for(int i=0;i<n;i++)
      {
         promedio[i] = ( calif1[i] + calif2[i] + calif3[i] )/3;
         if(promedio[i] >= 7 )
         {
            estatus[i] = "Aprobado";
         }else{
            estatus[i] = "Reprobado";
         }
         String cadena=nombre[i];
         int largo=cadena.length( );
         int espacios=15-largo;
         System.out.print((i+1) + ": " + nombre[i]);
```

```
        for(int k=0;k<=espacios;k++)
        {
           System.out.print(" ");
        }
        System.out.print(calif1[i] + "  "
        + calif2[i] + "  " + calif3[i] + "\t" + String.format("%.2f", promedio[i]) + "\t\t" +
estatus[i]+"\n");
     }
  }
//Método principal
   public static void main(String[ ] args)
   {
      Scanner teclado=new Scanner(System.in);
      //instanciamos el objeto p
      Promedios p=new Promedios( );
      /* Limpiamos la pantalla */
      System.out.print('\u000C');
      System.out.print("¿Cuantos alumnos calificarás? ");
      int n=teclado.nextInt( );
      String nombres[ ]=new String[n];
      //Se leen los nombres y se almacenan en el arreglo nombre
      nombres=p.pedirNombres(n);
      //Se lee la calificación 1  y se almacena en el arreglo calif1
      System.out.print('\u000C');
      System.out.println("Primera calificación:");
      double calif1[ ]=new double[n];
      calif1=p.pedirCalif(nombres, n);
      //Se lee la calificación 2  y se almacena en el arreglo calif2
      System.out.print('\u000C');
      System.out.println("Segunda calificación:");
      double calif2[ ]=new double[n];
      calif2=p.pedirCalif(nombres, n);
      System.out.print('\u000C');
      //Se lee la calificación 3  y se almacena en el arreglo calif3
      System.out.println("Tercera calificación:");
      double calif3[ ]=new double[n];
      calif3=p.pedirCalif(nombres, n);
      /* Limpiamos la pantalla */
      System.out.print('\u000C');
      //Se imprime el encabezado de la tabla
      System.out.println("\n\n\t\t\t ** TABLA DE CALIFICACIONES **");
System.out.println("_____
_____");
      System.out.println("Nombre" + "\t\t  Calificaciones\t" + "Promedio\t" + " Estatus");

System.out.println("_____
_____");
      //Impresión de la tabla
```

```
        p.imprimeDatos(nombres,calif1,calif2,calif3,n);
    }
}
```

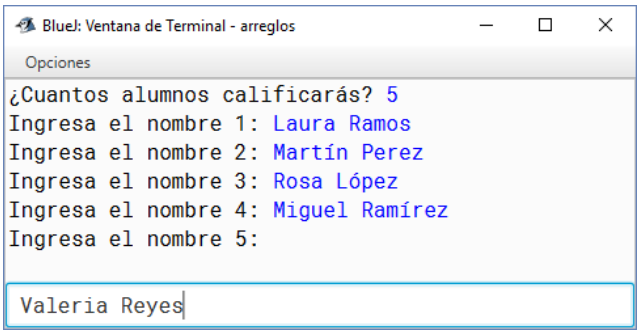
Después de compilar y ejecutar el programa obtendremos las siguientes pantallas:

Se pregunta el número de alumnos a calificar:



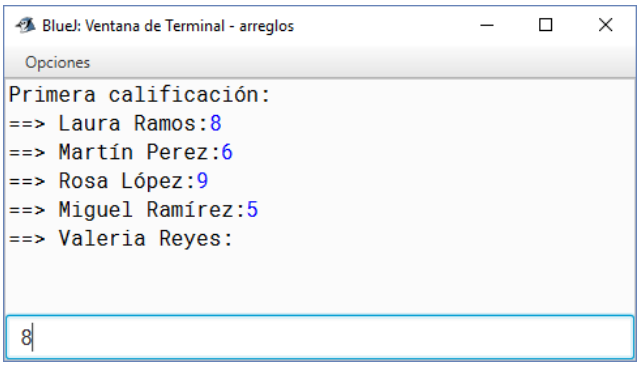

Se calificarán 5 alumnos.

Se ingresan los nombres de los alumnos:

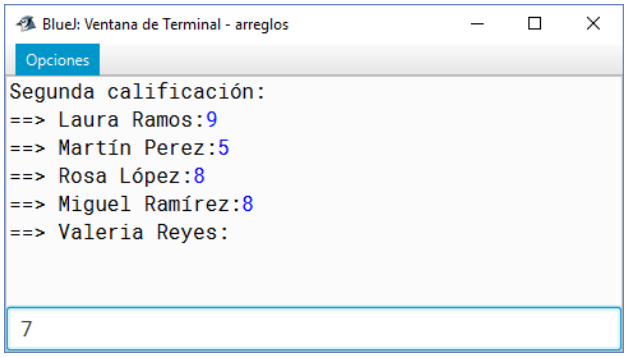



Se ingresan las calificaciones por cada alumno.

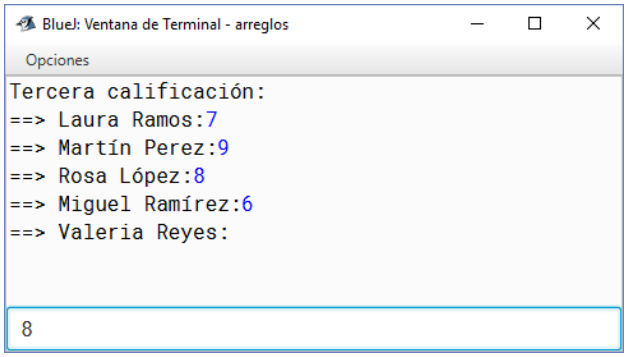
Primera Calificación:



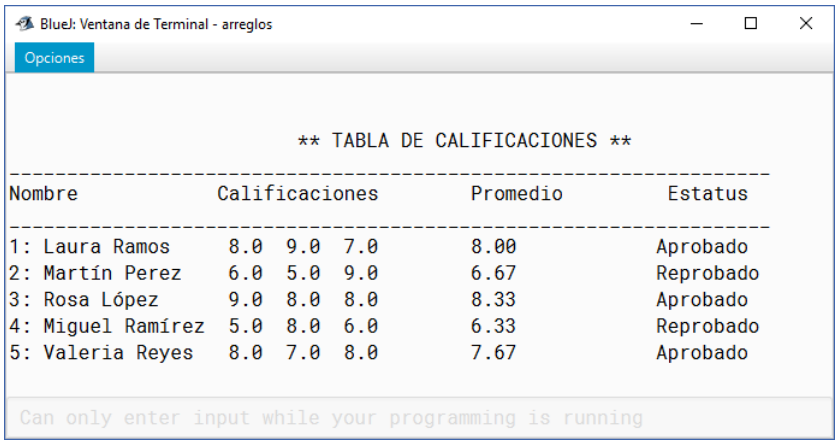
Segunda calificación:




Tercera Calificación:




A continuación, obtenemos la tabla de resultados.



## Actividad 2.6

1. Un arreglo es:
   a) Un conjunto de elementos que contienen valores del mismo tipo.
   b) Un tipo de datos.
   c) Una estructura en la que puedo almacenar datos de una entidad (personas o cosas).
   d) Un conjunto de caracteres alfanuméricos que permite identificar variables.

2. Un arreglo es _____, porque es una lista de elementos que emplea un solo índice.
   a) Bidimensional b) Unidimensional c) Multidimensional d) De acceso directo

3. En un arreglo la posición de los elementos se da a través de:
   a) El tipo de datos b) Un ciclo repetitivo c) Un índice d) Identificador

4. Es la forma correcta de declarar un arreglo de cinco enteros:
   a) int n[ ] = new int[ ];
   b) int n[5] = new int[ ];
   c) int n = new int[5];
   d) int n[ ] = new int[5];

5. ¿Cuál es el resultado de ejecutar las siguientes instrucciones?

```
for(int i = 0; i < 5; i++)
{
        System.out.println(n[i]);
}
```

   a) Inicializa el arreglo n[i]
   b) Imprime el contenido de los elementos n[0], n[1], n[2], n[3], n[4]
   c) Imprime el contenido de los elementos n[1], n[2], n[3], n[4], n[5]
   d) Asigna el valor de i a los elementos del arreglo n[i]

6. ¿Cuál es la acción resultante después de ejecutar el código del siguiente método?

```
public String[ ] pedirNombres(int n)
{
  Scanner teclado=new Scanner(System.in);
  String nombre[ ]=new String[n];
  for(int i=0;i<n;i++)
  {
    System.out.println("Ingresa el nombre " + (i+1) + ": " );
    nombre[i] = teclado.nextLine( );
  }
  return(nombre);
}
```

a) Leer *n* nombres de alumnos y los almacena en un arreglo entero de tamaño *n*.
b) Leer nombres de *n* alumnos y los almacena en un arreglo String de tamaño *n*.
c) Leer nombres de n alumnos y los almacena en un arreglo String de tamaño n y regresa el valor del arreglo.
d) Leer nombres de n alumnos y los almacena en un arreglo String de tamaño *n* e imprime el arreglo.

7. Al obtener la información almacenada de algunos elementos de un arreglo, se realiza una acción de:

a) Recorrido b) Inicialización c) Almacenamiento d) Acceso

8. Al acceder a cada uno de los elementos del arreglo se realiza una acción de:

a) Recorrido b) Inicialización c) Almacenamiento d) Acceso

9. ¿Cuál es la acción que realiza el siguiente código?

```
for(int i=0;i<n;i++)
 {
   System.out.print("==> "+nombre[i]+":");
   calif[i]=teclado.nextDouble( );
 }
return(calif);
```

a) Accesa al arreglo nombre y almacena un valor en el arreglo calif
b) Recorre el arreglo calif e imprime los nombres
c) Recorre el arreglo nombre y almacena valores en el arreglo calif
d) Recorre el arreglo nombre y almacena un valor en el arreglo calif

10. Es la sentencia que determina el tamaño del arreglo nombre:

a) t=nombre.String( ) b) length =t.nombre c) t=nombre.length( ) d) t = nombre[n];

## 2.8 Arreglos bidimensionales

**Propósito:**

El alumno comprenderá qué son los arreglos bidimensionales, cómo se declaran, los tipos que existen y el uso que tienen para resolver problemas desarrollando programas que los involucren en los métodos de una Clase.

- **Definición**

Los arreglos bidimensionales, representan tablas de valores las cuales agrupan datos ordenados en filas y columnas. Un arreglo bidimensional es conocido como una **matriz,** esto es**,** un conjunto ordenado en una estructura de filas y columnas.

Para identificar un elemento específico del arreglo debemos especificar dos índices, el primero identifica la fila y el segundo la columna. La figura 2.7.1 ilustra un arreglo bidimensional, al que llamaremos ***a***, que contiene tres filas y cuatro columnas (es decir, un arreglo de tres por cuatro). En general, un arreglo con m filas y n columnas se le llama arreglo de m por n.

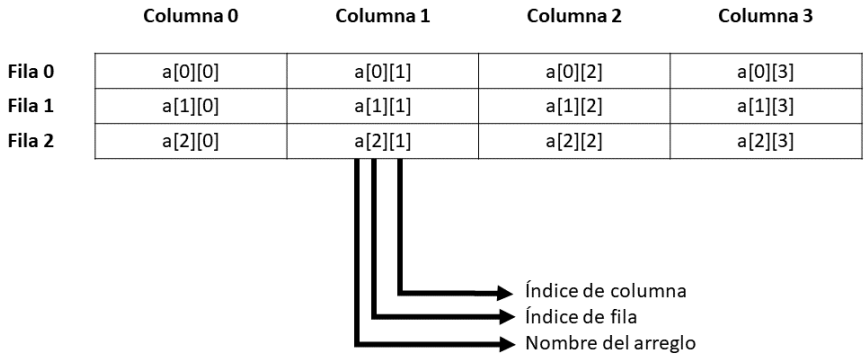



***Figura 2.7.1 Arreglo bidimensional de 3 por 4.***

- **Declaración**

La sintaxis para declarar un arreglo bidimensional es semejante a un arreglo unidimensional, solamente debemos agregar un segundo corchete, así su estructura es la siguiente:

*tipo_de_dato[ ][ ] nombre_del_arreglo;*

Y también se puede declarar un objeto:

*tipo_de_dato[ ][ ] nombre_del_arreglo = new tipo_de_dato [filas][columnas];*

Para declarar el arreglo entero del ejemplo de la figura 2.7.1 las sentencias quedarían como sigue:

*int [ ][ ] matrizDeEnteros;*

*int matrizDeEnteros[ ][ ]= new int [3][4];*

**Arreglos irregulares.**

Se pueden crear arreglos en los que el número de columnas de cada fila es variable, estos se conocen como *arreglos irregulares*. Para declararlos primero se indica el número de filas. Por ejemplo:

*int b[ ][ ] = new int[3][ ]; //crea una matriz b de 3 filas*

A continuación, a cada fila se le puede asignar un número distinto de columnas:

*b[0] = new int[5]; //crea 5 columnas para la fila 0*
*b[1] = new int[3]; //crea 3 columnas para la fila 1*
*b[2] = new int[4]; //crea 4 columnas para la fila 2*

Estas sentencias crean un arreglo irregular con tres filas. La fila 0 tiene 5 columnas, la fila 1 tienen 3 columnas y la fila 2 tiene 4 columnas como se muestra en la siguiente figura:

| | Columna 0 | Columna 1 | Columna 2 | Columna 3 | Columna 4 |
|---|---|---|---|---|---|
| Fila 0 | b[0][0] | b[0][1] | b[0][2] | b[0][3] | b[0][4] |
| Fila 1 | b[1][0] | b[1][1] | b[1][2] | | |
| Fila 2 | b[2][0] | b[2][1] | b[2][2] | b[2][3] | |

***Figura 2.7.2 Arreglo irregular***.

Y también podemos declararlo una lista de elementos, como se muestra en la siguiente sentencia:

*int[ ][ ] b = {{0,1,2,3,4},{0,1,2},{0,1,2,3}};*

- **Uso**

Los arreglos bidimensionales son conocidos como matrices y corresponden a una estructura de datos que pueden almacenar muchos más datos que los arreglos unidimensionales, ya que se componen de *m* filas y *n* columnas.

Podemos usar los arreglos bidimensionales para almacenar los datos de cualquier tabla de valores y hacer operaciones con ellos. Por ejemplo, la tabla de calificaciones de un profesor como se muestra a continuación:

| | Prueba 1 | Prueba 2 | Prueba 3 | Promedio |
|---|---|---|---|---|
| Estudiante 1 | 8.7 | 9.6 | 7.0 | 8.4 |
| Estudiante 2 | 6.8 | 8.7 | 9.0 | 8.2 |
| Estudiante 3 | 9.4 | 10.0 | 9.0 | 9.5 |
| Estudiante 4 | 10.0 | 8.1 | 8.2 | 8.8 |
| Estudiante 5 | 8.3 | 6.5 | 8.5 | 7.8 |
| Estudiante 6 | 7.8 | 8.7 | 6.5 | 7.7 |
| Estudiante 7 | 8.5 | 7.5 | 8.3 | 8.1 |
| Estudiante 8 | 9.1 | 9.4 | 10.0 | 9.5 |

***Tabla 2.7.3 Arreglo bidimensional utilizado para almacenar calificaciones.***

La estructura del arreglo necesaria para almacenar las calificaciones de la tabla sería de ocho filas y cuatro columnas y de tipo double. Los datos del arreglo se utilizarán de la siguiente forma: el promedio de las calificaciones se obtendrá sumando los elementos de la fila correspondientes a *Prueba1, Prueba2 y Prueba3,* la suma obtenida se dividirá entre tres y este resultado será el promedio.

Así cualquier tabla puede ser almacenada en un arreglo bidimensional, solo necesitamos declararlo, inicializarlo y recorrerlo para leer sus valores como se explica a continuación.

- **Inicialización de un arreglo bidimensional.**

Para asignar los valores a una matriz podemos inicializar el arreglo en una línea como se muestra a continuación:

*int[ ][ ] matrizDeEnteros = {{1,3,5,7},{5,4,1,6},{7,9,6,1}};*

Debemos notar que los elementos de toda la matriz se expresan en una sola línea, de la siguiente forma: primero se abre la llave que considera a todos los elementos de la matriz, después se expresan entre llaves los elementos de cada una de las filas, separados por comas, en nuestro ejemplo tenemos un arreglo de tres filas por cuatro columnas.

Para llenar la matriz de 3X4 del mismo ejemplo, pero elemento por elemento empleamos las siguientes sentencias:

Para la fila 0 y de las columnas 0 a la 3:

*matrizDeEnteros [0][0] = 1;*
*matrizDeEnteros [0][1] = 3;*
*matrizDeEnteros [0][2] = 5;*
*matrizDeEnteros [0][3] = 7;*

Para la fila 1 y de las columnas 0 a la 3:

*matrizDeEnteros [1][0] = 5;*
*matrizDeEnteros [1][1] = 4;*
*matrizDeEnteros [1][2] = 1;*
*matrizDeEnteros [1][3] = 6;*

Para la fila 2 y de las columnas 0 a la 3:

*matrizDeEnteros [2][0] = 7;*
*matrizDeEnteros [2][1] = 9;*
*matrizDeEnteros [2][2] = 6;*
*matrizDeEnteros [2][3] = 1;*

La matriz que se inicializó en ambos casos es la que se muestra a continuación:

| | Columna[0] | Columna[1] | Columna[2] | Columna[3] |
|---|---|---|---|---|
| Fila[0] | 1 | 3 | 5 | 7 |
| Fila[1] | 5 | 4 | 1 | 6 |
| Fila[2] | 7 | 9 | 6 | 1 |

***Tabla 2.7.4. Matriz de números enteros***

Podemos guardar los valores del arreglo utilizando la estructura de control *for* en dos ocasiones, una para recorrer las filas y otra las columnas, y solicitando los valores por teclado:

```
//Itera a través de las filas del arreglo.
for ( int m = 0; m < matrizDeEnteros.length; m++ )
{
  //Recorre las columnas de la fila actual.
  for ( int n = 0; n < matrizDeEnteros[ m ].length; n++ )
  {
    //Se imprime el mensaje y la posición del arreglo para ingresar el valor.
    System.out.print("Dame el valor ["+m+"]["+n+"]: ");
    //Se lee el valor por teclado y se guarda en el arreglo.
   matrizDeEnteros[m][n]=teclado.nextInt( );
  }
  System.out.println( ); // Inicia nueva línea de salida.
}
```

- **Acceso de un arreglo bidimensional.**

Para leer los valores de la matriz debemos hacer uso de la estructura de control *for* en dos ocasiones, una para recorrer las filas y otra las columnas e imprimir el valor del arreglo:

```
//Itera a través de las filas del arreglo.
for ( int m = 0; m < matrizDeEnteros.length; m++ )
{
  //Itera a través de las columnas de la fila actual.
  for ( int n = 0; n < matrizDeEnteros[ m ].length; n++ )
  {
    //Se imprime el valor de la matriz y se emplea la tabulación \t para dejar espacio.
    System.out.print(matrizDeEnteros[m][n]+"\t");
  }
  System.out.println( ); // inicia nueva línea de salida
}
```

## 2.9 Aplicación de los arreglos bidimensionales

Vamos a emplear el uso de arreglos bidimensionales para resolver el siguiente problema:

Desarrollar un programa para una cadena de tiendas de artículos deportivos que pida y almacene la cantidad de artículos vendidos por tienda y muestre la tabla con los totales. La cadena cuenta con cinco tiendas (Oriente, Vallejo, Naucalpan, Sur y Azcapo) y tres artículos deportivos (futbol, basquetball y volleyball) y la tabla final obtenida debe ser la siguiente:

| | Futbol | Basquetball | Volleyball | Total |
|---|---|---|---|---|
| 1. Oriente | | | | Total Oriente |
| 2. Vallejo | | | | Total Vallejo |
| 3. Naucalpan | | | | Total Naucalpan |
| 4. Sur | | | | Total Sur |
| 5. Azcapo | | | | Total Azcapo |
| Total | Total Futbol | Total Basquetball | Total Volleyball | Suma total |

***Tabla 2.9.1. de artículos deportivos.***

**Solución:**

Se declara una clase Tienda con los atributos enteros tiendas, artículos, y un arreglo bidimensional también entero del número de tiendas (5) como filas y los artículos (3) como columnas.

```
import java.util.Scanner;
public class Tienda
{
   int tiendas=5;
   int articulos=3;
   int[ ][ ] venta=new int[tiendas][articulos];
```

Declaramos el método Getter para obtener el valor de regreso del número de tiendas y artículos.

```
   public int getTiendas( )
   {
      return(tiendas);
   }

   public int getArticulos( )
   {
      return(articulos);
   }
```

- **Con estructuras repetitivas**

Para el arreglo *venta* debemos declarar los métodos Setter y Getter, para modificar una vez solicitados los datos y obtener el arreglo de regreso para mostrarlo en pantalla. Primero declaramos el método Setter que solicita un arreglo como parámetro y no retorna nada. A continuación, explicamos este método.

Con la estructura repetitiva *for* anidado vamos a acceder al arreglo *venta* para cambiar los valores. El primer *for* se debe ejecutar desde cero hasta el total de las filas, que está determinado por el tamaño del arreglo venta (venta.length), el segundo *for* debe ejecutarse desde cero hasta el total de columnas de cada fila determinado por el índice de la fila y el tamaño del arreglo venta (venta[i].length) quedando de la siguiente manera:

```
//Inicia el método set para el atributo venta que solicita como parámetro un arreglo
   public void setVenta(int[ ][ ] venta)
   {
      //Empleo de estructura repetitiva for anidado para modificar el atributo venta
      for(int i=0;i<venta.length;i++)
      {
         for(int j=0;j<venta[i].length;j++)
         {
            this.venta[i][j]=venta[i][j];
         }
      }
   }
```

El método Getter solo regresa un arreglo bidimensional, como se muestra a continuación:

```
   public int[ ][ ] getVenta( )
   {
      return(venta);
   }
```

- **Acceso**

Vamos a desarrollar un método para leer y guardar los datos en un arreglo de la siguiente forma, las filas estarán dadas por el número de tiendas y las columnas por los artículos.

```
//Inicia el método para leer y guardar los datos
/*Se solicitan los parámetros de número de tiendas y número de artículos y devuelve el arreglo
ingresado*/
   public int[ ][ ] pedirDatos(int tiendas, int articulos)
   {
      Scanner teclado=new Scanner(System.in);
      /*Inicializamos un arreglo con los nombres de las tiendas, otro con los artículos y otro
      para la venta.*/
      String[ ] t={"Oriente","Vallejo","Sur","Naucalpan","Azcapo"};
      String[ ] a={"Futbol","Basquetball","Volleyball"};
      int venta[ ][ ]=new int[tiendas][articulos];

      /*Utilizamos el for anidado para leer los valores de la tienda con el índice [i], del
      artículo  con el índice [j] y guardando el valor en el arreglo de la venta[i][j]*/
      for(int i=0;i<tiendas;i++)
      {
         for(int j=0;j<articulos;j++)
```

```
        {
         System.out.print("Ingresa la venta en la tienda "+t[i]+" de artículos de "+a[j]+":  ");
           venta[i][j]=teclado.nextInt( );
        }
        System.out.println( );
     }
     return(venta);
}
//Finaliza el método para leer y guardar los datos.
```

- **Recorrido**

Para obtener la suma de las ventas por tienda y por artículo, esto es por fila y columna, vamos a calcular los totales al momento de imprimir los datos. Para ello vamos a declarar un método que imprima la tabla solicitando un arreglo como parámetro inicial y declarando los arreglos de tc[ ] para guardar la suma por columna que corresponde a los artículos y tf[] para calcular la suma total por fila que corresponde a los totales por tienda.

```
//Inicio del método para imprimir tabla y calcular los totales de filas y columnas
/*Se solicita como parámetro el arreglo venta y no regresa ningún valor.*/
   public void imprimirTabla(int[ ][ ] venta)
   {
      String[ ] tiendas={"Oriente","Vallejo","Sur","Naucalpan","Azcapo"};
       //Se inicializa el arreglo tc a cero para hacer la suma por columna
       int tc[ ]={0,0,0};
      for(int i=0;i<venta.length;i++)
      {
         String cadena=tiendas[i];
         /*Calcula el número de espacio con los que complementará el arreglo tienda
         para escribir la tabla en forma ordenada */
         int largo=cadena.length( );
         int espacios=15-largo;
         System.out.print((i+1) + ". " + tiendas[i]);
         for(int k=0;k<=espacios;k++)
         {
            System.out.print(" ");
         }
         //Se inicializa el arreglo a cero para hacer la suma por fila
         int tf[ ]={0,0,0,0,0};
         for(int j=0;j<venta[i].length;j++)
         {
        /*En cada iteración del for anidado se suman las cantidades por columna y fila y se
        imprime el valor del arreglo.*/
            tc[j]=tc[j]+venta[i][j];
            System.out.print(venta[i][j]+"\t\t");
            tf[i]=tf[i]+venta[i][j];
         }
```

```
        //Se imprime la suma de la fila
        System.out.println(tf[i]);
      }
      System.out.println("-------------------------------------------------------------------");
      String c="Total";
      int largo=c.length( );
      int espacios=18-largo;
      System.out.print(c);
      for(int k=0;k<=espacios;k++)
      {
        System.out.print(" ");
      }
      //Se calcula la suma de todas las columnas
      int total=tc[0]+tc[1]+tc[2];
      //Se imprimen los valores totales por columna
      System.out.print(tc[0]+"\t\t"+tc[1]+"\t\t"+tc[2]+"\t\t"+total);
    }
}
//Finaliza el método para imprimir tabla y calcular los totales de filas y columnas
```

*Método principal*

A continuación, se muestra el método principal del programa, aquí es en donde se llaman los métodos Setter, Getter, pedirDatos e imprimirTabla después de instanciar un objeto de la clase Tienda al que llamaremos t; en seguida se describe el código correspondiente:

```
  public static void main(String[ ] args)
  {
    //Se instancia un objeto t de la clase Tienda
    Tienda t=new Tienda( );
    //Se limpia la pantalla
    System.out.print('\u000C');
    //Se coloca el encabezado del programa
    System.out.println("CADENA DE TIENDAS DEPORTIVAS");
    System.out.println("-------------------------------------------------------------------");
    int[ ][ ] venta=new int[t.getTiendas( )][t.getArticulos( )];
    /*Se llama al método pedirDatos con los valores que regresa getTiendas y getArticulos
    y se guarda en el arreglo venta*/
    venta=t.pedirDatos(t.getTiendas( ),t.getArticulos( ));
    //Se modifica el atributo venta con el método setVenta
    t.setVenta(venta);
    //Se limpia la pantalla
    System.out.print('\u000C');
    //Se coloca el encabezado de la tabla
    System.out.println("  ***  ***  ***  ***  V E N T A S  ***  ***  ***  ***\n");
    System.out.println("\t\tFutbol\t  Basquetball\t  Volleyball\t\tTotal");
    System.out.println("-------------------------------------------------------------------");
```

```
    //Invocamos al método imprimirTabla con el método getVenta que retorna un arreglo
    t.imprimirTabla(t.getVenta( ));
  }
}
```

## 2.10 Implementación

El código final del programa queda de la siguiente forma:

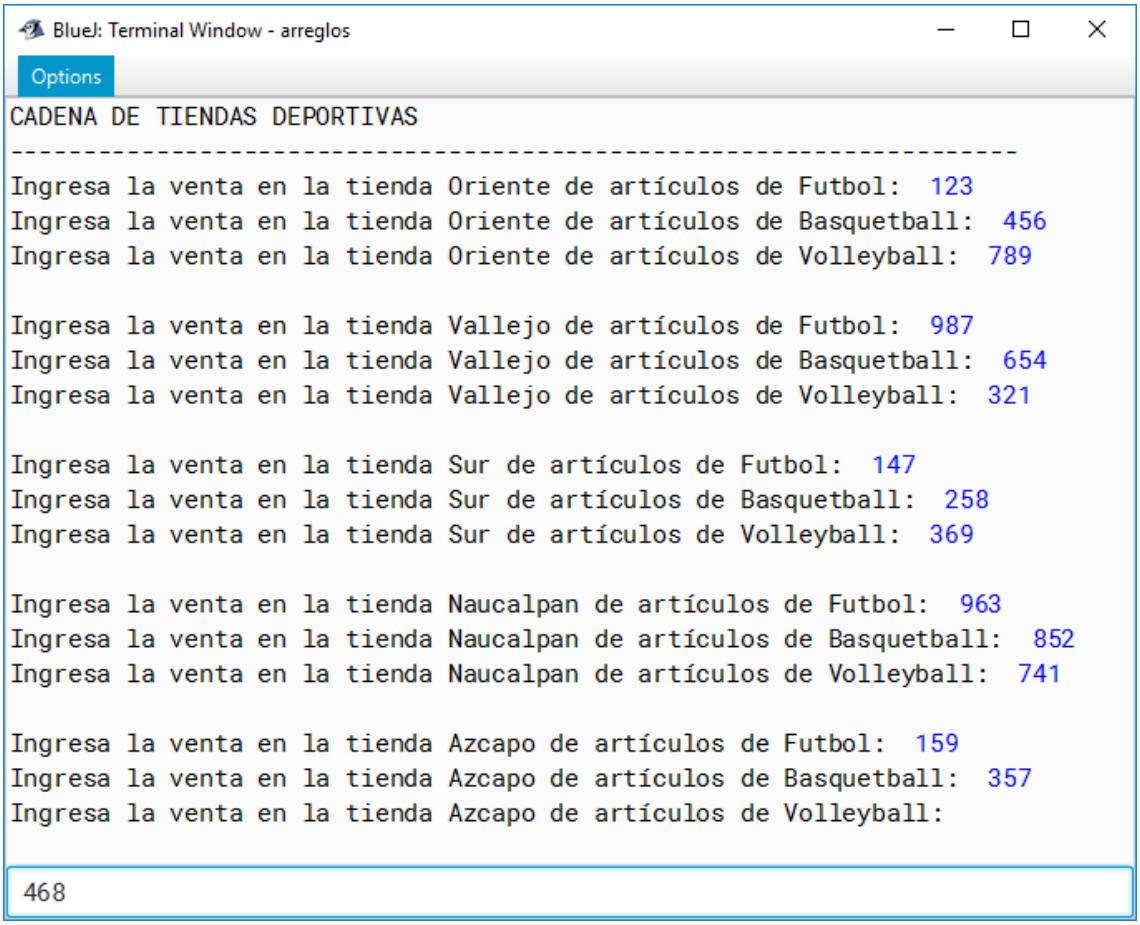
```
import java.util.Scanner;
public class Tienda
{
   int tiendas=5;
   int articulos=3;
   int[ ][ ] venta=new int[tiendas][articulos];
   public int getTiendas( )
   {
      return(tiendas);
   }
   public int getArticulos( )
   {
      return(articulos);
   }
   public void setVenta(int[ ][ ] venta)
   {
      for(int i=0;i<venta.length;i++)
      {
         for(int j=0;j<venta[i].length;j++)
         {
            this.venta[i][j]=venta[i][j];
         }
      }
   }
   public int[ ][ ] getVenta( )
   {
      return(venta);
   }
   public int[ ][ ] pedirDatos(int tiendas, int articulos)
   {
      Scanner teclado=new Scanner(System.in);
      String[ ] t={"Oriente","Vallejo","Sur","Naucalpan","Azcapo"};
      String[ ] a={"Futbol","Basquetball","Volleyball"};
      int venta[ ][ ]=new int[tiendas][articulos];
      for(int i=0;i<tiendas;i++)
      {
         for(int j=0;j<articulos;j++)
         {
```

```
            System.out.print("Ingresa la venta en la tienda "+t[i]+" de artículos de "+a[j]+":  ");
            venta[i][j]=teclado.nextInt( );
         }
         System.out.println( );
      }
      return(venta);
   }
   public void imprimirTabla(int[ ][ ] venta)
   {
      String[ ] tiendas={"Oriente","Vallejo","Sur","Naucalpan","Azcapo"};
      int[ ] tc={0,0,0};
      for(int i=0;i<venta.length;i++)
      {
         String cadena=tiendas[i];
         int largo=cadena.length( );
         int espacios=15-largo;
         System.out.print((i+1) + ". " + tiendas[i]);
         for(int k=0;k<=espacios;k++)
         {
            System.out.print(" ");
         }
         int tf=0;
         for(int j=0;j<venta[i].length;j++)
         {
            tc[j]=tc[j]+venta[i][j];
            System.out.print(venta[i][j]+"\t\t");
            tf=tf+venta[i][j];
         }
         System.out.println(tf);
      }
      System.out.println("---------------------------------------------------------------------");
      String c="Total";
      int largo=c.length( );
      int espacios=18-largo;
      System.out.print(c);
      for(int k=0;k<=espacios;k++)
      {
         System.out.print(" ");
      }
      int total=tc[0]+tc[1]+tc[2];
      System.out.print(tc[0]+"\t\t"+tc[1]+"\t\t"+tc[2]+"\t\t"+total);
   }
   public static void main(String[ ] args)
   {
      Tienda t=new Tienda( );
      System.out.print('\u000C');
      System.out.println("CADENA DE TIENDAS DEPORTIVAS");
      System.out.println("---------------------------------------------------------------------");
```

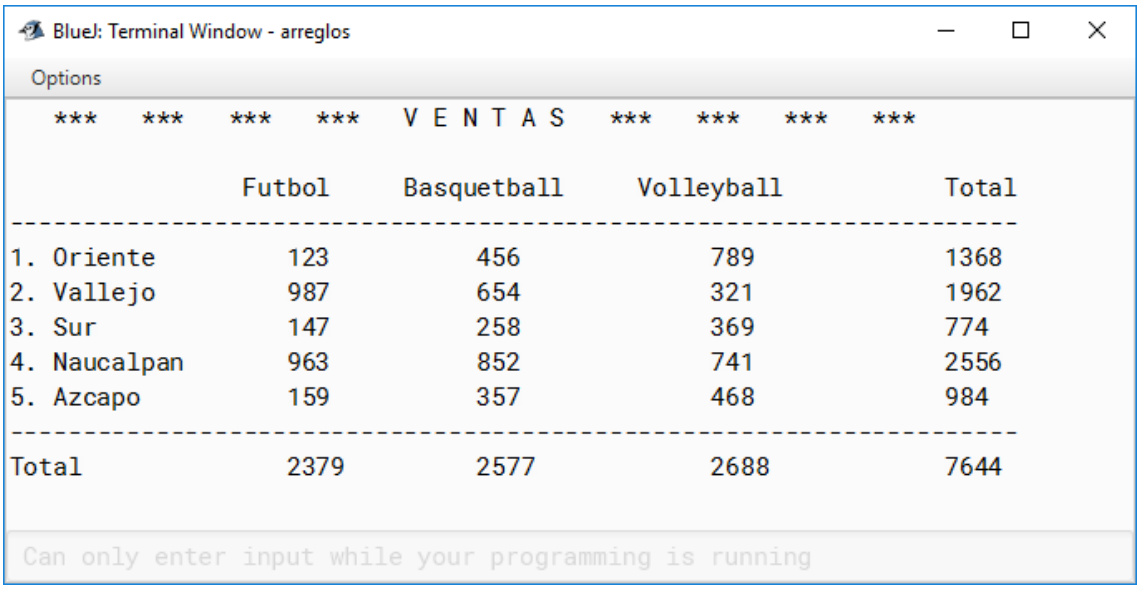
```
        int[ ][ ] venta=new int[t.getTiendas( )][t.getArticulos( )];
        venta=t.pedirDatos(t.getTiendas( ),t.getArticulos( ));
        t.setVenta(venta);
        System.out.print('\u000C');
        System.out.println("  ***  ***  ***  ***  V E N T A S  ***  ***  ***  ***\n");
        System.out.println("\t\tFutbol\t  Basquetball\t  Volleyball\t\tTotal");
        System.out.println("-------------------------------------------------------------------------");
        t.imprimirTabla(t.getVenta( ));
    }
}
```

Al compilar y ejecutar el programa tenemos los siguientes resultados:

Se leen las ventas por tienda y artículo.




Se muestra la tabla de ventas y los totales por tienda y artículo.




## Actividad 2.7

1. Una matriz necesita _____ para acceder a sus elementos.

a) Un índice b) Dos índices c) Una coordenada d) Un vector

2. Para acceder a los elementos de un arreglo bidimensional el orden de los índices es:

a) [columna,fila] b) [fila] [columna] c) [columna] [fila] d) [fila, columna]

3. Al arreglo bidimensional se le conoce también como:

a) Lista b) Vector c) Matriz d) Contador

4. Selecciona la sentencia que crea un arreglo bidimensional de tres filas con cinco columnas:

a) int [3][5] valor = new int[ ][ ];

b) int [ ][ ] valor = new int[5][3];

c) int [ ][ ] valor = new int[3][5];

d) array [ ][ ] valor = new int[3][5];

5. Selecciona la sentencia que inicializa un arreglo de 3 X 4

a) int[ ][ ] matrizDeEnteros = {{1,3,5},{5,4,1},{7,9,6},{7,6,1}};

b) int[ ][ ] matrizDeEnteros = {1,3,5,7,5,4,1,6,7,9,6,1};

c) int[ ][ ] matrizDeEnteros = {{1,3,5,7}{5,4,1,6}{7,9,6,1}};

d) int[ ][ ] matrizDeEnteros = {{1,3,5,7},{5,4,1,6},{7,9,6,1}};

6. Que elemento de la siguiente matriz ocupa la posición [1][2]

| | | |
|---|---|---|
| Carolina | Daniel | Mónica |
| Paola | Fernando | Carmen |
| Ximena | Román | Julia |
| Ana | Israel | Rebeca |

a) Daniel b) Fernando c) Carmen d) Román

7. En un arreglo irregular se cumple lo siguiente:

a) El número de columnas de cada fila es variable.

b) El número de filas de cada columna es igual.

c) El número de columnas de cada fila no es variable.

d) El número de columnas de cada fila es igual.

8. Qué acción realiza el siguiente código:

```
for(int i=0;i<costo.length;i++)
{
        for(int j=0;j<costo[i].length;j++)
        {
                System.out.print([i]+”.”+[j]+”: “);
            costo[i][j]=teclado.nextInt( );
       }
}
```

a) Muestra cada elemento del arreglo.

b) Guarda un valor en cada elemento del arreglo.

c) Borra a cero cada elemento del arreglo.

d) Modifica los valores del arreglo costo por otro arreglo.

9. La siguiente sentencia NO cumple la definición de arreglo en su declaración:

a) int[ ][ ] calificaciones = new int[20][3];

b) double[ ][ ] promedios = new String[20][2];

c) String nombre[ ][ ] = new String[20][3];

d) char sexo[ ][ ] = new char[20][1];

10. La siguiente sentencia inicializa un arreglo irregular que al imprimirlo forma un triángulo de * con altura 5 y base 5:

a) char[ ][ ] a={{'*'},{'*','*'},{'*','*','*'},{'*','*','*','*'}};

b) char[ ][ ] a={{'*','*','*','*','*'},{'*','*','*'},{'*','*'},{'*'}};

c) char[ ][ ] a={{'*'},{'*','*'},{'*','*','*'},{'*','*','*','*'},{'*','*','*','*','*'}};

d) char[ ][ ] a={{'*','*','*','*','*'},{'*','*','*','*'},{'*','*','*'},{'*','*'},{'*'}};

## 2.11 Implementación.

**Propósito.**

Desarrolla un proyecto que utilice las sentencias vistas hasta el momento, incluyendo los arreglos.

**Problema para resolver.**

Realizar un programa para ingresar los nombres y las calificaciones de un grupo de alumnos. El programa debe mostrar opciones para ingresar nombres y calificaciones, calcular los promedios y mostrar listas de alumnos, calificaciones y promedios.

**Solución.**

Para resolver este problema se propone utilizar dos arreglos unidimensionales, para los nombres y los promedios. Un arreglo bidimensional para las calificaciones.

El método principal debe realizar las siguientes acciones:

| Mostrar lista de opciones | Iniciar arreglos | Definir arreglos |
|---|---|---|

La lista de opciones determina los siguientes métodos a realizar:

| Lista de alumnos | Ingresar calificaciones | Ingresar nombres |
|---|---|---|

| calcular promedios | mostrar nombres y calificaciones |
|---|---|

| mostrar nombres, calificaciones y promedios |
|---|

**Desarrollo del programa**

Definimos la clase Alumno con sus atributos y sus métodos setter y getter.

```
public class Alumno {

   private String nombre;
   private double promedio;
   private boolean aprobado;

   public String getNombre( ) {
```

```
    return nombre;
  }

  public void setNombre(String nombre) {
    this.nombre = nombre;
  }

  public double getPromedio( ) {
    return promedio;
  }

  public void setPromedio(double promedio) {
    this.promedio = promedio;
  }

  public boolean getAprobado( ) {
    return aprobado;
  }

  public void setAprobado(boolean aprobado) {
    this.aprobado = aprobado;
  }
}
```

Definimos la clase Promedios con sus atributos y métodos. Esta clase proporciona la funcionalidad para poder trabajar con los arreglos, *alumnos* es un arreglo unidimensional para los datos de cada alumno y *califs* es un arreglo bidimensional para sus calificaciones. El concepto de método contructor será explicado en la siguiente unidad. Aquí lo usaremos ya que en este ejemplo es necesario para inicar los arreglos.

```
import java.util.Scanner;

public class Promedios {

  Scanner leer = new Scanner(System.in);

  //Se definen los arreglos como atributos de la clase Promedios
  private Alumno[ ] alumnos; //arreglo de alumnos
  private double[ ][ ] califs; //arreglo de calificaciones

  //Método constructor para la clase Promedios
  //Recibe el número de alumnos y el número de calificaciones
  public Promedios(int numAlumnos, int numCalifs) {
    //Dimensionamos los arreglos con los datos ingresados
    alumnos = new Alumno[numAlumnos];
    califs = new double[numAlumnos][numCalifs];
    //Inicializa arreglo de alumnos
```

```
    iniciaArregloAlumnos( );
    //Inicializa arreglos de calificaciones
    iniciaArregloCalificaciones( );
}

//Método privado para iniciar arreglo de alumnos
//Este método es necesario para llenar el arreglo de alumnos
private void iniciaArregloAlumnos( ) {
    for (int i = 0; i < alumnos.length; i++) {
        //Se crea una instancia de alumno
        Alumno alum = new Alumno( );
        //Se asigna al elemento correspondiente del arreglo
        alumnos[i] = alum;
    }
}

//Método privado para iniciar o limpiar el arreglo de calificaciones
private void iniciaArregloCalificaciones( ) {
    //numero de renglones representa el número de alumnos
    for (int i = 0; i < califs.length; i++) {
        //numero de columnas representa el número de calificaciones
        for (int j = 0; j < califs[0].length; j++) {
            //asignamos un valor inicial a las calificaciones
            califs[i][j] = 0.0;
        }
    }
}

//Método para ingresar los nombres de alumnos
public void ingresarNombresAlumnos( ) {
    String nombre, tmp;
    System.out.println("Ingresar nombres de alumnos");
    for (int i = 0; i < alumnos.length; i++) {
        System.out.print("Ingresar nombre del alumno " + (i + 1) + ": ");
        //ingresamos el nombre del alumno
        nombre = leer.next( );
        //asignamos nombre al elemento i del arreglo
        alumnos[i].setNombre(nombre);
        //limpiamos el buffer de entrada
        tmp = leer.nextLine( );
    }
}

//Método para ingresar las calificaciones de alumnos
public void ingresarCalificaciones( ) {
    System.out.println("Ingresar calificaciones");
    //Ingresar calificaciones de los Alumnos
    for (int i = 0; i < alumnos.length; i++) {
```

```
        System.out.println("Ingresa calificaciones de " + alumnos[i].getNombre());
        //Ingresa las calificaciones del alumno
        for (int j = 0; j < califs[0].length; j++) {
            System.out.print("Calificacion " + (j + 1) + ": ");
            califs[i][j] = leer.nextDouble( );
        }
    }
}

//Método para mostrar la lista de alumnos
public void mostrarListaAlumnos( ) {
    System.out.println("Lista de alumnos");
    for (int i = 0; i < alumnos.length; i++) {
        System.out.println(alumnos[i].getNombre());
    }
}

//Método para mostrar la lista de alumnos y calificaciones
public void mostrarCalificaciones( ) {
    System.out.println("Lista de calificaciones");
    for (int i = 0; i < alumnos.length; i++) {
        System.out.print(alumnos[i].getNombre() + " - ");
        //numero de columnas representa el numero de calificacioens
        for (int j = 0; j < califs[0].length; j++) {
            // Mostramos la calificacion correspondiente
            System.out.print(califs[i][j] + " ");
        }
        System.out.println( ); //salto de linea
    }
}

//Método para calcular promedios
public void calculaPromedios( ) {
    double suma, promedio;
    int numCalifs = califs[0].length;
    for (int i = 0; i < alumnos.length; i++) {
        //limpia suma
        suma = 0.0;
        for (int j = 0; j < numCalifs; j++) {
            //acumula las calificaciones del alumno
            suma += califs[i][j];
        }
        //calcula el promedio del alumno ubicado en la posición i
        promedio = suma / numCalifs;
        //registra el promedio del alumno correspondiente
        alumnos[i].setPromedio(promedio);
        if (promedio >= 6) {
            alumnos[i].setAprobado(true);
```

```
        } else {
            alumnos[i].setAprobado(false);
        }
    }
}

//Método para mostrar la lista de nombres con promedio y si esta aprobado
public void mostrarPromedios( ) {
    for (int i = 0; i < alumnos.length; i++) {
        System.out.print(alumnos[i].getNombre( ) + " - " + alumnos[i].getPromedio( ) + " - ");
        if (alumnos[i].getAprobado( )) {
            System.out.println("Aprobado");
        } else {
            System.out.println("Reprobado");
        }
    }
}

//Método para mostrar la lista de nombres con calificaciones, promedio y aprobado
public void mostrarCalificacionesYPromedios( ) {
    System.out.println("Lista de calificaciones y promedios");
    for (int i = 0; i < alumnos.length; i++) {
        System.out.print(alumnos[i].getNombre( ) + " - ");
        for (int j = 0; j < califs[0].length; j++) {
            System.out.print(califs[i][j] + " ");
        }
        System.out.print(" - " + alumnos[i].getPromedio( ) + " - ");
        if (alumnos[i].getAprobado( )) {
            System.out.println("Aprobado");
        } else {
            System.out.println("Reprobado");
        }
    }
}

 //Método para mostrar las opciones y elegir alguna
public int seleccionaOpcion( ) {
    int opcion;
    System.out.println("Lista de opciones a elegir");
    System.out.println("1) Ingresar nombres de Alumnos");
    System.out.println("2) Ingresar calificaciones");
    System.out.println("3) Mostrar lista de alumnos");
    System.out.println("4) Mostrar calificaciones");
    System.out.println("5) Mostrar promedios");
    System.out.println("6) Mostrar calificaciones y promedios");
    System.out.println("7) Fin");
    System.out.print("Selecciona una opcion: ");
    opcion = leer.nextInt( );
```

```
        return opcion;
    }
}
```

Definimos la clase Principal donde inicia la ejecución del programa. También se puede integrar a la clase Promedios para que sea el método principal como en los ejemplos anteriores. En este ejemplo lo definimos por separado.

```
import java.util.Scanner;

public class Principal {

    public static void main(String[ ] args) {

        Scanner leer = new Scanner(System.in);

        System.out.println("Inicio del programa");
        //Definimos variables para el número de alumnos y calificaciones
        int numAlumnos;
        int numCalifs;
        int opcion; //variable para elegir de la lista o menu de opciones
        //Ingresamos número de alumnos
        System.out.print("Ingresa el número de alumnos: ");
        numAlumnos = leer.nextInt( );
        //Ingresamos número de calificaciones
        System.out.print("Ingresa el número de calificaciones: ");
        numCalifs = leer.nextInt( );
        //Se crea una instancia de la clase Promedios para poder acceder a sus métodos
        Promedios prom = new Promedios(numAlumnos, numCalifs);
        boolean fin = false;
        do {
            opcion = prom.seleccionaOpcion( );
            switch (opcion) {
                case 1:
                    prom.ingresarNombresAlumnos( );
                    break;
                case 2:
                    prom.ingresarCalificaciones( );
                    break;
                case 3:
                    prom.mostrarListaAlumnos( );
                    break;
                case 4:
                    prom.mostrarCalificaciones( );
                    break;
                case 5:
                    prom.calculaPromedios( );
```

```
            prom.mostrarPromedios( );
            break;
         case 6:
            prom.mostrarCalificacionesYPromedios( );
            break;
         case 7:
            fin = true;
            break;
         default:
            System.out.println("Opción no valida");
      }
    } while (!fin);
    System.out.println("Adios !!!!");
  }
}
```

Ejecución del programa:

```
Inicio del programa
Ingresa el número de alumnos: 3
Ingresa el número de calificaciones: 3
Lista de opciones a elegir
1) Ingresar nombres de Alumnos
2) Ingresar calificaciones
3) Mostrar lista de alumnos
4) Mostrar calificaciones
5) Mostrar promedios
6) Mostrar calificaciones y promedios
7) Fin
Selecciona una opcion: 1
Ingresar nombres de alumnos
Ingresar nombre del alumno 1: Alumno1
Ingresar nombre del alumno 2: Alumno2
Ingresar nombre del alumno 3: Alumno3
```

```
Lista de opciones a elegir
1) Ingresar nombres de Alumnos
2) Ingresar calificaciones
3) Mostrar lista de alumnos
4) Mostrar calificaciones
5) Mostrar promedios
6) Mostrar calificaciones y promedios
7) Fin
Selecciona una opcion: 2
Ingresar calificaciones
Ingresa calificaciones de Alumno1
Calificacion 1: 10
Calificacion 2: 9
Calificacion 3: 8
Ingresa calificaciones de Alumno2
Calificacion 1: 6
Calificacion 2: 7
Calificacion 3: 8
Ingresa calificaciones de Alumno3
Calificacion 1: 4
Calificacion 2: 5
Calificacion 3: 6
```

```
Lista de opciones a elegir
1) Ingresar nombres de Alumnos
2) Ingresar calificaciones
3) Mostrar lista de alumnos
4) Mostrar calificaciones
5) Mostrar promedios
6) Mostrar calificaciones y promedios
7) Fin
Selecciona una opcion: 3
Lista de alumnos
Alumno1
Alumno2
Alumno3
```

```
Lista de opciones a elegir
1) Ingresar nombres de Alumnos
2) Ingresar calificaciones
3) Mostrar lista de alumnos
4) Mostrar calificaciones
5) Mostrar promedios
6) Mostrar calificaciones y promedios
7) Fin
Selecciona una opcion: 4
Lista de calificaciones
Alumno1 - 10.0 9.0 8.0
Alumno2 - 6.0 7.0 8.0
Alumno3 - 4.0 5.0 6.0
```

```
Lista de opciones a elegir
1) Ingresar nombres de Alumnos
2) Ingresar calificaciones
3) Mostrar lista de alumnos
4) Mostrar calificaciones
5) Mostrar promedios
6) Mostrar calificaciones y promedios
7) Fin
Selecciona una opcion: 5
Alumno1 - 9.0 - Aprobado
Alumno2 - 7.0 - Aprobado
Alumno3 - 5.0 - Reprobado
```

```
Lista de opciones a elegir
1) Ingresar nombres de Alumnos
2) Ingresar calificaciones
3) Mostrar lista de alumnos
4) Mostrar calificaciones
5) Mostrar promedios
6) Mostrar calificaciones y promedios
7) Fin
Selecciona una opcion: 6
Lista de calificaciones y promedios
Alumno1 - 10.0 9.0 8.0  - 9.0 - Aprobado
Alumno2 - 6.0 7.0 8.0  - 7.0 - Aprobado
Alumno3 - 4.0 5.0 6.0  - 5.0 - Reprobado
```

```
Lista de opciones a elegir
1) Ingresar nombres de Alumnos
2) Ingresar calificaciones
3) Mostrar lista de alumnos
4) Mostrar calificaciones
5) Mostrar promedios
6) Mostrar calificaciones y promedios
7) Fin
Selecciona una opcion: 7
Adios !!!!
```

## Solución de las actividades

## Unidad 2

### Actividad 2.1

| | | |
|---|---|---|
| 1. b | 2. a | 3. b |

### Actividad 2.2

| | | | |
|---|---|---|---|
| 1. c | 2. a | 3. b | 4. c |

### Actividad 2.3

| | | | |
|---|---|---|---|
| 1. d | 2. a | 3. a | 4. a |

### Actividad 2.4

| | | | | |
|---|---|---|---|---|
| 1. b | 2. b | 3. a | 4. a | 5. a |

### Actividad 2.5

**6, 1, 5, 2, 4, 3**

### Actividad 2.6

| | | | | |
|---|---|---|---|---|
| 1. a | 2. b | 3. c | 4. d | 5. b |
| 6. c | 7. d | 8. a | 9. c | 10. c |

### Actividad 2.7

| | | | | |
|---|---|---|---|---|
| 1. b | 2. b | 3. c | 4. c | 5. d |
| 6. c | 7. a | 8. b | 9. b | 10. c |

## Referencias

## Unidad 2

Ceballos Sierra, F. (2000). *Java 2*. RA-MA Editorial.

Cuenca, W. (2019). *Arrays (arreglos) multidimensionales en Java. Declaración y uso. Ejemplos y ejercicios resueltos*. (CU00905C). [online] Aprenderaprogramar.com. Disponible en: https://www.aprenderaprogramar.com/index.php?option=com_content&view=article&id=233:arrays-arreglos-multidimensionales-en-java-declaracion-y-uso-ejemplos-y-ejercicios-resueltos-cu00905c&catid=58&Itemid=180 [Recuperado el 10 diciembre 2018].

Dean, J. (2009). *Introducción a la programación con Java*. México: Mc. Graw-Hill.

García Hernández, E. (2019). *Programación Java*. [online] Puntocomnoesunlenguaje.blogspot.com.es. Disponible En: http://puntocomnoesunlenguaje.blogspot.com.es [Recuperado el 10 diciembre 2018].

Lima Díaz, F. (2010). *Manual avanzado de Java 6*. Madrid: Anaya Multimedia.

Walton, A. (2019). *Javadesdecero.es*. [online] Disponible en: https://javadesdecero.es/arrays/unidimensionales-multidimensionales/#2_Arrays_unidimensionales [Recuperado el 10 diciembre 2018].

Suarez, M. (n.d.). *Diagrama de Flujo*. [imagen] Disponible en: <https://www.researchgate.net/profile/Miguel_Suarez_Ledo/publication/313664762/figure/fig3/AS:461396619075585@1487016934061/Figura-32-Diagrama-de-Flujo-mostrando-un-ciclo-while.png> [Recuperado el 16 Diciembre 2018].

Sznajdleder, P. and Fernández, D. (2000). *Java a fondo*. Alfaomega Grupo Editor.

# UNIDAD 3

## Polimorfismo, constructores, colaboración y herencia de clases.


- Concepto de polimorfismo
- Concepto de constructor
- Implementación de constructores y polimorfismo
- Interacción y comunicación entre Clases
- Implementación de interacción y comunicación entre clases
- Herencia
- Implementación de la herencia de clase

# UNIDAD 3. Polimorfismo, constructores, colaboración y herencia de clases.

**Propósito:**

Al finalizar la unidad el alumno:
Implementará programas en Java utilizando polimorfismo, constructores, colaboración y herencia de Clases para aprovechar las bondades de la programación orientada a objetos.

**Aprendizajes:**

El alumno:

- Conoce el concepto de polimorfismo y constructor.
- Desarrolla programas que involucren polimorfismo y constructores.
- Comprende la colaboración de Clases para la resolución de problemas.
- Desarrolla programas que involucren la colaboración de Clases.
- Comprende el concepto de herencia en la resolución de un problema.
- Desarrolla programas que involucren la herencia de Clases.

## Introducción

En la programación orientada a objetos, una de las características principales que tiene como ventaja con respecto a la programación estructurada, es la reutilización del código. Para ello se utilizan diversos conceptos tales como el polimorfismo o la herencia los cuales vamos a desarrollar a continuación.

## 3.1 Concepto de polimorfismo.

**Propósito**

El alumno conocerá el concepto de polimorfismo.

El polimorfismo se refiere a la propiedad por la cual es posible enviar mensajes sintácticamente iguales a objetos de tipos distintos. El único requisito que deben cumplir los objetos que se utilizan de manera polimórfica es saber responder al mensaje que se les envía.

El polimorfismo consiste en utilizar una misma expresión para invocar a diferentes versiones de un mismo método, dicho de otra forma, es hacer que un mismo método haga diferentes cosas dependiendo de la clase donde se ejecute.

Por ejemplo, para llamar al método area( ) de Cuadrado podemos hacer:

*a=new Cuadrado(..);*

*a.area( );*

y para llamar al método area( ) de Rectangulo haríamos:

*a=new Rectangulo(..);*

*a.area( );*

Estamos utilizando la sentencia *a.area( );* para llamar a dos versiones diferentes de un mismo método, la ventaja que obtenemos con esto es que se reutiliza código, pues las mismas instrucciones se pueden utilizar con varios objetos.

## 3.2 Concepto de constructor.

**Propósito**

El alumno conocerá el concepto de constructores.

En la programación orientada a objetos, un constructor es un método que se ejecuta cada vez que se crea un objeto nuevo, su finalidad consiste en reservar la memoria necesaria además de inicializar las variables que forman parte de los atributos de la clase con valores válidos.

Existen tres tipos de constructores en Java:

a. Constructor por default. En este caso, si no se especifica ningún constructor, Java genera automáticamente uno por defecto.
b. Constructor sin parámetros. Este tipo de constructor se define al escribirse en el código del programa con el mismo nombre de la clase y los atributos se pueden inicializar.
c. Constructor con parámetros. Es aquel en el cual se envían determinados valores para asignarlos a los argumentos de la clase mediante el uso de parámetros.

Adicionalmente, es importante tener en cuenta lo siguiente con respecto a los constructores:

- No regresan ningún tipo de valor ni siquiera “void”.
- El nombre del constructor debe ser el mismo que el de la clase.
- Los constructores no se pueden heredar.

- Una clase puede tener varios constructores cuya diferencia entre ellos son los argumentos y su tipo.
- Los constructores utilizan el modificador de acceso public y solo en casos especiales pueden contener otro tipo de modificador.

A continuación, se muestra un ejemplo:

```
public class LeeNombre
{
        //declaración de atributos
         private String nombre;
       //constructor sin parámetros
       public LeeNombre ( );
         {
                nombre=”sin nombre”;
         }
      //Constructor con parámetros
       public LeeNombre (String nombre)
       {
                this.nombre=nombre;
       }
}
```

## 3.3 Implementación de constructores y polimorfismo.

**Propósito**

El alumno realizará programas orientado a objetos que tanto contengan constructores, así como polimorfismo.

- **Implementación de constructores**

**Ejemplo 1. Constructor por Defecto.**

A continuación, vamos a explicar el uso de constructores, utilizando para ello la clase Persona, la cual tendrá como atributos nombre y edad.

También vamos a emplear cuatro métodos, dos de ellos serán *Getter*, que como vimos en la primera unidad, se utilizan para leer la información de los atributos y otros dos métodos *Setter* que se utilizan para asignar valores a los atributos de la clase.

Definamos un primer conjunto de instrucciones que contiene el nombre de la clase y los atributos nombre y edad:

```
public class Persona {
private String nombre;
private int edad;
```

En las siguientes instrucciones vamos a definir los métodos *Getter* los cuales vamos a utilizar para leer la información que esta almacenada en los atributos de la clase, es decir *nombre y edad*.

```
//Método Getter para leer nombre
   public String getNombre( )   {
     return nombre;
   }

  //Método Getter para leer edad
  public int getEdad( )   {
    return edad;
  }
```

Posteriormente vamos a definir los métodos Setter, los cuales nos van a servir para asignarle valores a los atributos.

```
//Método Setter para asignar un valor a nombre
public void setNombre(String n)   {
    nombre = n;
  }

//Método Setter para asignar un valor a edad
public void setEdad(int e)    {
edad = e;
}
```

Finalmente tendremos el método principal, en el cual vamos a crear el objeto *individuo* que pertenece a la clase *Persona*, en donde se van a mostrar los valores que tienen los atributos, es decir los valores de las variables nombre y edad.

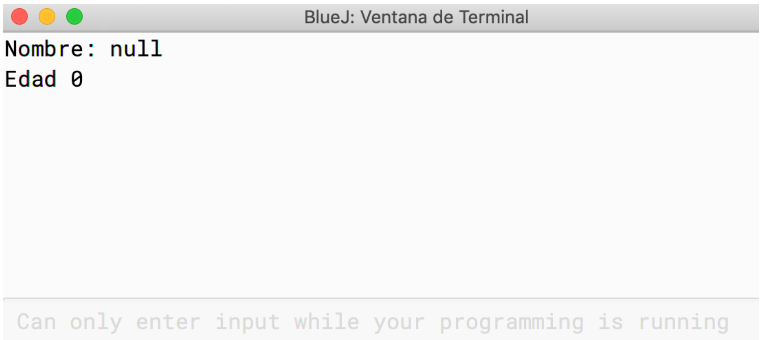
Aquí es muy importante observar que nosotros *no les hemos asignado ningún valor a los atributos* y, por tanto, lo que se muestre en pantalla serán los valores que Java da por default a las variables *nombre y edad* cuando se crea el objeto *individuo*, por esta razón se llama constructor por defecto.

```
//Se inicia el método principal
      public static void main(String[ ] args)  {
        //Se crea el objeto individuo
        Persona individuo = new Persona( );
        //Se muestran en pantalla los valores de las variables nombre y edad
        System.out.println("Nombre: " + individuo.getNombre( ));
        System.out.println("Edad " + individuo.getEdad( ));
     }
}
```

El programa completo es el siguiente:

```
public class Persona
{
  private String nombre;
  private int edad;

//Método Getter para leer nombre
   public String getNombre( )   {
     return nombre;
   }
  //Método Getter para leer edad
  public int getEdad( )   {
    return edad;
  }

//Método Setter para asignar un valor a nombre
public void setNombre(String n)   {
    nombre = n;
  }
//Método Setter para asignar un valor a edad
public void setEdad(int e)    {
edad = e;
}
//Se inicia el método principal
      public static void main(String[ ] args)  {
        //Se crea el objeto individuo
        Persona individuo = new Persona( );
        //Se muestran en pantalla los valores de las variables nombre y edad
        System.out.println("Nombre: " + individuo.getNombre( ));
        System.out.println("Edad " + individuo.getEdad( ));
     }
}
```

La salida al ejecutar el programa es la siguiente:



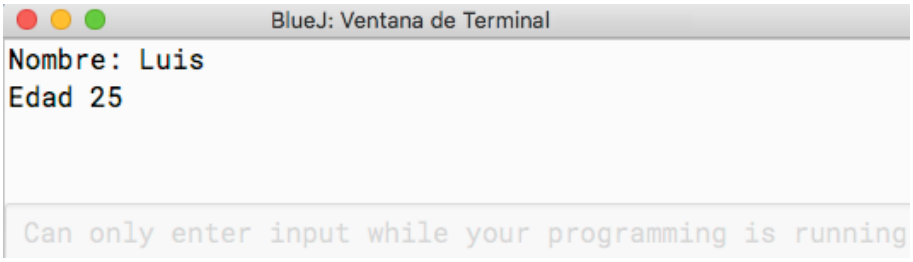
Como podemos observar del código anterior, nosotros **no** definimos ningún constructor por lo cual Java creo uno por default, de esta forma le dio el valor de **null** a nombre y **0** a edad al ejecutarse el programa.

**Ejemplo 2. Constructor sin parámetros.**

A continuación, haremos una modificación al programa anterior, vamos a agregar un constructor sin parámetros. Para ello incluimos el siguiente bloque de código dentro de la clase Persona antes de los métodos Getter y Setter:

```
//Constructor sin parámetros
public Persona( )   {
  nombre = "Luis";
  edad = 25;
  }
```

Ejecutamos nuevamente el programa para ver las diferencias que existen una vez que incorporamos el constructor sin parámetros, de tal forma que ahora la salida es la siguiente:




Observamos que la salida del programa *muestra los mismos valores que fueron definidos en el constructor sin parámetros*.

**Ejemplo 3. Constructor con parámetros.**

En este tipo de constructor, a través de parámetros se envían los valores que deseamos asignar a los atributos de la clase, si continuamos con el ejemplo anterior, entonces modificamos el constructor para que reciba parámetros, por tanto, debemos agregar el siguiente código:

```
//Constructor con parametros
  public Persona(String nombre, int edad)   {
      this.nombre = nombre;
      this.edad = edad;
  }
```

Adicionalmente debemos de modificar los métodos Setter ya que vamos a enviar parámetros, por lo tanto, agregamos el termino this para para hacer una diferencia entre el atributo del objeto y el parámetro que se envía mediante el constructor.

```
//Modificación del Método Setter para asignar un valor a nombre
 public void setNombre (String nombre)   {
    this.nombre = nombre;
  }

//Modificación del Método Setter para asignar un valor a nombre
public void setEdad (int edad)    {
  this.edad = edad;
  }
```

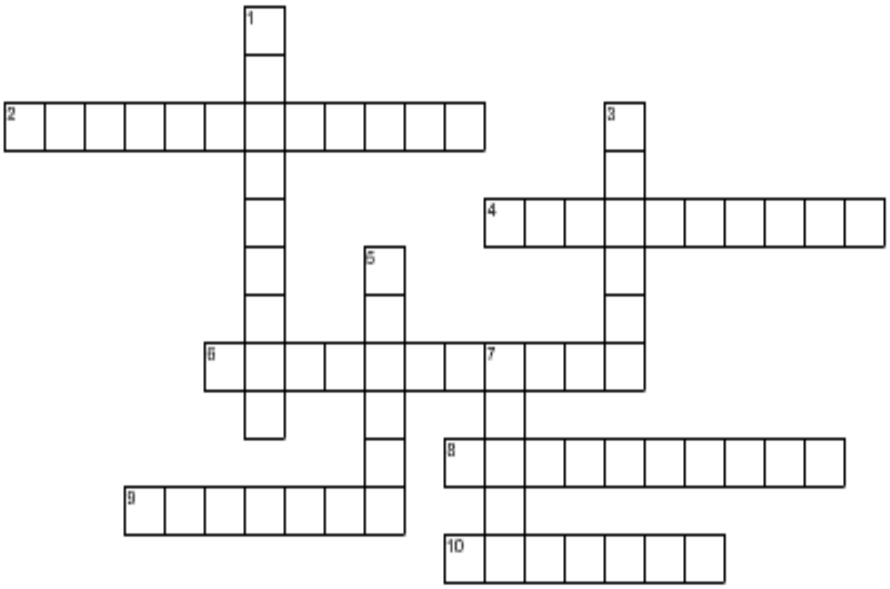
Finalmente, en el método principal al momento de que se crea el objeto *individuo*, se envían como argumentos los parámetros *Juan* y *40* para que sean asignados a los atributos de la clase, es decir: nombre y edad.

```
//Se inicia el método principal
      public static void main(String[ ] args)  {
        //Se crea el objeto individuo y se envian los parametros Juan y 40
        Persona individuo = new Persona("Juan", 40);
       //Se muestran los valores de las variables nombre y edad
       System.out.println("Nombre: " + individuo.getNombre( ));
       System.out.println("Edad " + individuo.getEdad( ));
     }
```

La salida del programa será la siguiente:

Actividad 3.1.

Instrucciones:
Lee detenidamente las pistas y resuelve el crucigrama




| VERTICAL | HORIZONTAL |
|---|---|
| 1. Son las variables de los constructores. | 2. Característica de la POO por la cual es posible definir varios métodos o comportamientos de un objeto bajo un mismo nombre. |
| 3. Son los métodos que se utilizan para asignar valores a los atributos. | 4. El polimorfismo permite _____ código. |
| 5. Son los métodos que se utilizan para leer la información. | 6. Es un método que se ejecuta cada vez que se crea un objeto nuevo. |
| 7. El nombre del constructor es el mismo que el de la _____. | 8. Constructor con _____, Es aquel en el cual se envían determinados valores. |
| | 9. Los constructores no se pueden _____. |
| | 10. Cuando no se declara un constructor, java genera uno por_____. |

- **Implementación de polimorfismo.**

El polimorfismo se representa con un nuevo concepto que es la sobrecarga de métodos. La sobrecarga de métodos consiste en definir dos o más métodos dentro de la misma clase, que compartan nombre y que las declaraciones de sus parámetros sean diferentes.

Cuando Java encuentra una llamada a un método sobrecargado, ejecuta la versión del que tiene parámetros en número y tipo que coinciden con los argumentos utilizados en la llamada.

**Ejemplo:**

Se define la clase llamada **Sobrecarga** con cuatro métodos sobrecargados llamados *prueba*, diferenciándose entre ellos por el número y tipo de parámetros, en el método principal (main) se llama a cada método y se muestra en pantalla la salida de cada método. En seguida se describe el programa y su ejecución:

Creamos la clase llamada Sobrecarga:

```
public class Sobrecarga
{
```

Creamos el primer método llamado prueba que no regresa nada, no tiene parámetros e imprime un mensaje de salida:

```
   public void prueba( )
   {
      System.out.println("Método sin argumentos.");
   }
```

En seguida escribimos un segundo método que no regresa nada y se llama igual que el anterior, pero esta vez solicita un parámetro tipo entero. Al final imprime un mensaje y el valor del argumento.

```
   public void prueba(int  x)
   {
      System.out.println("Método con 1 argumento.");
      System.out.println("x="+x);
   }
```

Declaramos un tercer método que no regresa nada, se llama igual que los anteriores, en esta ocasión pide dos parámetros de tipo entero y muestra un mensaje con los valores de los argumentos.

```
   public void prueba(int  x, int y)
   {
      System.out.println("Método con 2 argumentos.");
      System.out.println("x="+x+"y="+y);
   }
```

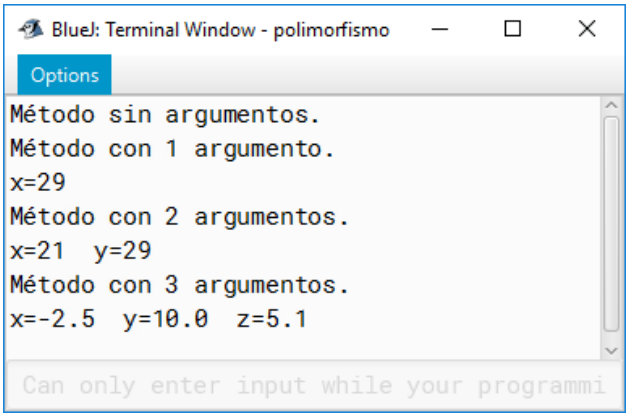
Escribimos un cuarto método que no regresa nada, se llama igual que los anteriores, solicita tres parámetros ahora de tipo double y muestra un mensaje con los valores de los tres argumentos.

```
public void prueba(double  x, double y, double z)
{
   System.out.println("Método con 3 argumentos.");
   System.out.println("x="+x+"  y= "+y+"  z= "+z);
}
```

Tenemos cuatro métodos sobrecargados, ahora vamos a escribir el método principal en el cual instanciaremos un objeto de la clase y mandaremos a llamar cada método con sus diferentes parámetros.

```
public static void main(String[ ] args)
   {
      Sobrecarga objeto=new Sobrecarga( );
      objeto.prueba( );
      objeto.prueba(29);
      objeto.prueba(21,29);
      objeto.prueba(-2.5,10.0,5.1);
   }
}
```

El resultado de ejecutar el programa muestra diferentes versiones del método prueba( ) y cada versión opera sobre diferentes tipos de datos.




- **Implementación de constructores con polimorfismo**

Además de la sobrecarga de métodos normales, se pueden sobrecargar los constructores; estos últimos normalmente se sobrecargan en las clases creadas, aunque no siempre porque una clase puede definirse sin constructor.

**Ejemplo:**

Se define la clase Persona con dos constructores sobrecargados: uno sin argumentos, y otro con cuatro argumentos. La clase Persona tiene los atributos: nombre, apellido paterno, apellido materno y edad.

Primero declaramos la clase Persona con sus atributos:

```
public class Persona
{
   //Declaramos los atributos
   private String nombre;
   private String apPat;
   private String apMat;
   private int edad;
```

Ahora vamos a declarar los dos constructores, el que no pide parámetros y el que sí pide parámetros.

En el constructor que no pide parámetros vamos a inicializar los atributos:

```
   public Persona( )
   {
      nombre= "Pedro";
      apPat= "Hernández";
      apMat= "López";
      edad=17;
   }
```

En el constructor que sí pide parámetros vamos a asignar los valores de los parámetros a las variables de instancia.

```
   public Persona(String nombre, String aP, String aM, int edad)
   {
      this.nombre=nombre;
      this.apPat=aP;
      this.apMat=aM;
      this.edad=edad;
   }
```

Vamos a escribir un método para imprimir los valores que va a tomar el objeto:

```
   public String imprimirObjeto( )
   {
      return "Persona{ "+nombre+", "+apPat+", "+apMat+", "+edad+"}";
   }
```

En el método principal se instancian dos objetos, uno con el constructor que no pide parámetros pero tiene valores de inicio y el segundo con el constructor que pide

parámetros. Después los vamos a imprimir llamando al método imprimirObjeto( ) con cada objeto creado.

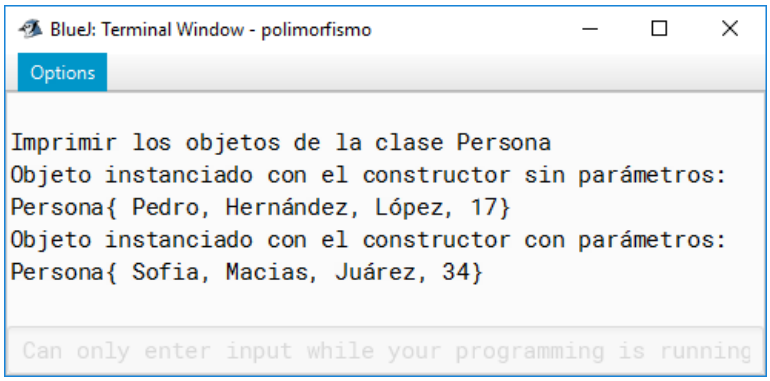
```
public static void main(String[ ] args){
      Persona persona1=new Persona( );
      Persona persona2=new Persona("Sofia","Macias","Juárez",34);
      //Limpiamos la pantalla
      System.out.println("\u000C");
      System.out.println("Imprimir los objetos de la clase Persona");
      System.out.println(persona1.imprimirObjeto( ));
      System.out.println(persona2.imprimirObjeto( ));
   }
}
```

A continuación, se muestra el programa completo de la clase Persona.

```
public class Persona
{
  //Inicializamos los atributos
  private String nombre;
  private String apPat;
  private String apMat;
  private int edad;
  /*Método constructor de la clase Persona que no pide parámetros*/
  public Persona( )
  {
    //Inicializamos los atributos
    nombre= "Pedro";
    apPat= "Hernández";
    apMat= "López";
    edad=17;
  }
  /*Método constructor de la clase Persona que pide parámetros*/
  public Persona(String nombre, String aP, String aM, int edad)
  {
    //Hacemos referencia de variables
    this.nombre=nombre;
    this.apPat=aP;
    this.apMat=aM;
    this.edad=edad;
  }
  /*Método para imprimir un objeto de la clase persona, esto es sus atributos*/
  public String imprimirObjeto( )
  {
    return "Persona{ "+nombre+", "+apPat+", "+apMat+", "+edad+"}";
  }
  /*Método principal*/
  public static void main(String[ ] args){
```

```
        //Instanciamos un objeto de la clase con el método constructor que no pide parámetros
        Persona persona1=new Persona( );
        //Instanciamos un objeto de la clase con el método constructor que pide parámetros
        Persona persona2=new Persona("Sofia","Macias","Juárez",34);
        //Limpiamos la pantalla
        System.out.println("\u000C");
        System.out.println("Imprimir los objetos de la clase Persona");
        //imprimimos los objetos
        System.out.println("Objeto instanciado con el constructor sin parámetros:");
        System.out.println(persona1.imprimirObjeto( ));
        System.out.println("Objeto instanciado con el constructor con parámetros:");
        System.out.println(persona2.imprimirObjeto( ));
    }
}
```

Si ejecutamos el código, la salida del programa es lo siguiente:




Notemos el polimorfismo en las siguientes líneas del programa:

**Persona persona1=new Persona( );**

**Persona persona2=new Persona("Sofia","Macias","Juárez",34);**

Los métodos constructores tienen el mismo nombre y lo que define a qué constructor se ingresa, son los argumentos que se envían como parámetros.

Aunque en el primer constructor no se piden parámetros, muestra valores por la inicialización que se hizo, y en el segundo constructor, los valores se colocan como parámetros. De esta manera podemos cambiar los valores de un objeto sin los métodos Setter y mostrar los valores con un método que imprima los atributos y no con los métodos Getter.

### Actividad 3.2.

**Problema 1.** Construir una clase Calculadora con dos métodos suma sobrecargados, uno que pida valores enteros y el otro método valores reales; ambos deben realizar la operación de suma y regresar el resultado. En el método principal se deberá instanciar un objeto y

mandar a llamar los dos métodos indicando los parámetros y mostrar mensajes en pantalla con los valores.

**Problema 2**. Construir una clase Mascota con los atributos raza, color y nombre; sin declarar los métodos Getter y Setter, en el método principal mostrar en pantalla los valores de inicio. El programa debe pedir por teclado los valores de los atributos de una mascota y mostrar los valores finales empleando un método de instancia. Utilizar los métodos constructores para resolver este problema.

## 3.4 Interacción y comunicación entre Clases.

**Propósito:**

El alumno:

Comprenderá la colaboración de clases para la resolución de problemas.
Desarrolla programas que involucren la colaboración de Clases.

Cuando dos clases se comunican entre sí para realizar una tarea, se dice que colaboran. Una colaboración consiste en que un objeto de una clase envía un mensaje a un objeto de otra clase mediante llamadas a métodos.

Hasta el momento hemos trabajado el método principal dentro de la misma clase que definíamos, pero ahora vamos a trabajar con una clase Principal, esto es, una clase que se encarga de ejecutar otras clases y así observar la interacción entre ellas.

Una clase Principal va a contener el método principal el cual es el punto de arranque del programa pues con este método llama a las clases controladas para crear objetos y manipularlos. La clase Principal está en un archivo separado de las clases que controla.

En la figura 3.4.1 se presenta como ejemplo la clase Principal que controla a la clase Profesor o clase controlada porque va a atender la creación y manipulación de objetos.

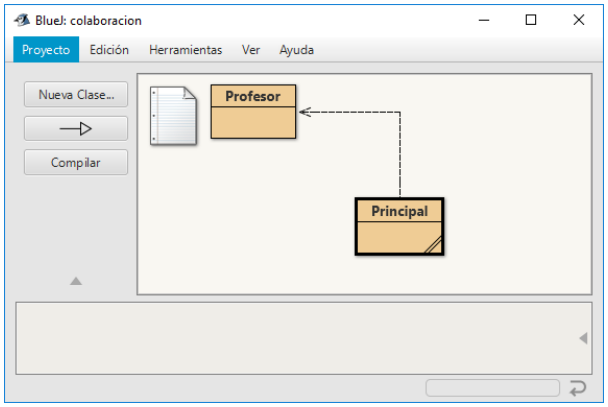



***Figura 3.4.1. Colaboración entre clases en BlueJ.***

Ambas clases, la principal y la controlada, contienen sentencias que expresan su colaboración; sin embargo, de esta manera no podemos conocer los métodos o atributos que contienen.

La programación orientada a objetos (POO) permite un diseño orientado a objetos (DOO) a partir de las clases en las que se definen los atributos y métodos. Para analizar las clases que intervienen en un problema se debe seguir un proceso detallado y después diseñar la colaboración de las clases implicadas.

Existe un lenguaje gráfico, conocido como Lenguaje Unificado de Modelado (UML por sus siglas en inglés: Unified Modeling Language), para comunicar los resultados de interacción y comunicación en el diseño orientado a objeto. UML es un esquema de representación gráfica para modelar programas orientados a objetos, que emplea diagramas con un conjunto de notaciones gráficas estándar. En UML, cada clase se modela en un diagrama de clases en forma de un rectángulo con tres partes: nombre de la clase hasta arriba, atributos en la parte de en medio y operaciones o métodos al final. En la figura 3.4.2 puede verse la colaboración entre clase mediante el diagrama de clases de Profesor y Principal, con la diferencia que éste nos muestra los atributos y métodos de las clases, dándonos más información de la colaboración que existe entre ellas.

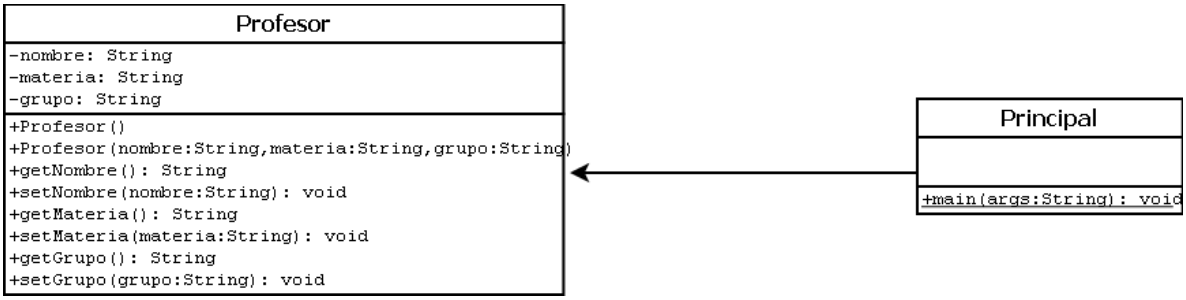



***Figura 3.4.2 Diagrama de clases de la clase Profesor y la clase Principal***

Con referencia a la figura anterior, un diagrama de clase tiene las siguientes características:

- El tipo de variables aparece a la derecha de la variable.

Por ejemplo: **nombre: String** el nombre de la variable es *nombre* y el tipo es *String.*

- Se antepone el signo "-" para acceso de tipo private.

Por ejemplo: **-materia: String**, es una variable de tipo private por que tiene el signo "-"

- Se antepone el signo "+" para acceso tipo public.

Por ejemplo: **+Profesor( )**, es el constructor que no pide parámetros y es de tipo public

- Se emplea el signo “=<valor>” para agregar la inicialización a cada declaración de variable.

En el ejemplo no aparece la inicialización de variables, pero se mostraría como sigue:

**-nombre: String=”María”**

- Se subraya el método main porque es declarado static.

**main(args: String[ ]): void**

- Se incluye un sufijo “<tipo>” en cada método para especificar el tipo de valor que el método devuelve

Por ejemplo: **+setNombre(nombre: String): void**, es el método set que pide un parámetro de tipo String y devuelve un dato de tipo void.

El diseño del diagrama de clases es el primer paso para manejar de manera esquemática la colaboración entre clases, para después implementar el código. Con el diagrama de clase nos damos cuenta de que la clase Profesor va a tener tres atributos de tipo String llamados nombre, materia y grupo, los constructores con y sin parámetros, y los métodos Getter y Setter para obtener y modificar los atributos respectivamente; y la clase Principal solo contendrá el método principal.

En la implementación de la clase Profesor vamos a declarar los atributos indicados en el diagrama con “-“ como private:

```
public class Profesor
{
   //Declaración de atributos.
   private String nombre;
   private String materia;
   private String grupo;
```

Se declara el constructor que no pide parámetros pero que tiene inicialización de ellos:

```
public Profesor( ){
      nombre="Gabriela López Vargas";
      materia="Taller de Cómputo";
      grupo="192B";
   }
```

Se declara el constructor que pide parámetros:

```
public Profesor(String nombre,String materia,String grupo){
      this.nombre=nombre;
      this.materia=materia;
      this.grupo=grupo;
   }
```

Se declaran los métodos Getter de cada atributo para obtener los valores que tomen; y también los métodos Setter para cambiar los valores:

```
public String getNombre( )
   {
      return nombre;
   }

   public void setNombre(String nombre)
   {
      this.nombre = nombre;
   }

   public String getMateria( )
   {
      return materia;
   }

   public void setMateria(String materia)
   {
      this.materia = materia;
   }

   public String getGrupo( )
   {
      return grupo;
   }

   public void setGrupo(String grupo)
   {
      this.grupo = grupo;
   }
}
```

El código completo de la clase Profesor se muestra a continuación:

```
public class Profesor
{
   //Declaración de atributos.
   private String nombre;
   private String materia;
   private String grupo;
   //Métodos constructor sin parámetros
   public Profesor( ){
      nombre="Gabriela López Vargas";
      materia="Taller de Cómputo";
      grupo="192B";
   }
```

```
  //Método constructor con parámetros
  public Profesor(String nombre,String materia,String grupo){
     this.nombre=nombre;
     this.materia=materia;
     this.grupo=grupo;
  }
  //Método Getter para leer nombre
  public String getNombre( )
  {
     return nombre;
  }
  //Método Setter para asignar un valor a nombre
  public void setNombre(String nombre)
  {
     this.nombre = nombre;
  }
  //Método Getter para leer materia
  public String getMateria( )
  {
     return materia;
  }
  //Método Setter para asignar un valor a materia
  public void setMateria(String materia)
  {
     this.materia = materia;
  }
  //Método Getter para leer grupo
  public String getGrupo( )
  {
     return grupo;
  }
  //Método Setter para asignar un valor a grupo
  public void setGrupo(String grupo)
  {
     this.grupo = grupo;
  }
}
```

En la clase Principal de este ejemplo vamos a pedir por teclado los datos del profesor y mostrarlos en pantalla llamando a los métodos de la clase Profesor. A continuación, se explica el código:

Antes de declarar la clase debemos importar la biblioteca java.util.Scanner porque vamos a leer los datos por teclado.

```
impor java.util.Scanner;
public class Principal
{
```

Declaramos el método principal, un objeto de la clase Scanner y un objeto de la clase Profesor:

```
public static void main(String[ ] args)
{
   Scanner teclado=new Scanner(System.in);
   Profesor profe=new Profesor( );
```

En seguida vamos a pedir los datos con el objeto teclado y asignarlos a los atributos con los métodos Setter del objeto profe:

```
   System.out.println("\u000C");
   System.out.println("DATOS DEL PROFESOR");
   System.out.print("Nombre: ");
   profe.setNombre(teclado.nextLine( ));
   System.out.print("Materia: ");
   profe.setMateria(teclado.nextLine( ));
   System.out.print("Grupo: ");
   profe.setGrupo(teclado.nextLine( ));
```

Finalmente obtenemos los valores de los atributos del objeto profe con los métodos Getter y los guardamos en variables para mostrar los datos del profesor en pantalla.

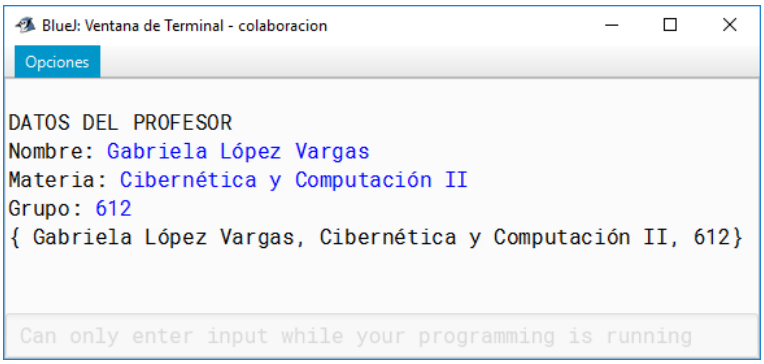
```
   String nombre=profe.getNombre( );
   String materia=profe.getMateria( );
   String grupo=profe.getGrupo( );
   System.out.print("{  "+nombre+", "+materia+", "+grupo+" }");
```

A continuación, se muestra el programa completo de la clase Principal:

```
import java.util.Scanner;
public class Principal
{
 //Método principal
  public static void main(String[ ] args){
    Scanner teclado=new Scanner(System.in);
    //Instanciamos un objeto de la clase Profesor
    Profesor profe=new Profesor( );
    //Limpiamos la pantalla
    System.out.println("\u000C");
    //Pedimos los datos del profesor
    System.out.println("DATOS DEL PROFESOR");
    System.out.print("Nombre: ");
    profe.setNombre(teclado.nextLine( ));
    System.out.print("Materia: ");
    profe.setMateria(teclado.nextLine( ));
    System.out.print("Grupo: ");
    profe.setGrupo(teclado.nextLine( ));
    //Obtenemos los datos del profesor
```

```
    String nombre=profe.getNombre( );
    String materia=profe.getMateria( );
    String grupo=profe.getGrupo( );
    //Mostramos los datos en pantalla
    System.out.print("{ "+nombre+", "+materia+", "+grupo+" }");
 }
}
```

Una vez implementado el código de ambas clases, la clase que vamos a ejecutar es la Principal porque contiene el método principal. La ejecución del programa solicita los datos del profesor y después muestra los datos en una línea y entre llaves.




## Actividad 3.3

Relaciona las siguientes columnas anotando la letra correcta dentro del paréntesis:

| | | |
|---|---|---|
| 1. Esquema en donde se especifica el nombre, los atributos y métodos de una clase. | ( ) | a) DOO |
| 2. Se encarga de controlar otras clases y cuando se ejecuta interactúa con ellas. | ( ) | b) UML |
| 3. Es una característica de la clase controlada por lo que no se ejecuta. | ( ) | c) Clase controlada |
| 4. Es un ejemplo de inicialización de variables en un diagrama de clase. | ( ) | d) Método principal |
| 5. Es la acción en que dos o más clases se comunican para realizar una tarea. | ( ) | e) +sumar(int:a, int:b):int |
| 6. En ella se crean los métodos y atiende las llamadas a ellos. | ( ) | f) Clase principal |
| 7. Lenguaje gráfico que sirve para comunicar los resultados de interacción en el DOO. | ( ) | g) Diagrama de clase |

8. Consiste en modelar programas orientados a objetos mostrando las clases con esquemas. ( )
9. Es el punto de arranque del programa y si no lo contiene una clase, esta no ejecuta nada. ( )
10. Es un ejemplo en el que se especifica el tipo de valor que devuelve un método. ( )

h) Sin método principal
i) Colaboración
j) -edad:int=40

**Problema**

Un banco desea realizar una aplicación para las cuentas bancarias de sus clientes. La aplicación debe solicitar al cliente su número de cuenta, nombre, capital invertido y tasa de interés; y calcular la ganancia y el saldo con las siguientes fórmulas:

ganancia = capital invertido * tasa de interés
saldo= capital invertido + ganancia.
Al finalizar la aplicación mostrará al cliente sus datos y la ganancia y saldo obtenidos.

**Solución**

Observemos que en este problema existe la colaboración de clases, a una la llamaremos CuentaBancaria que va a tener los datos del cliente y la otra clase Principal que va a pedir los datos y mostrarlos en pantalla.

A continuación, se muestra el diagrama de clase:

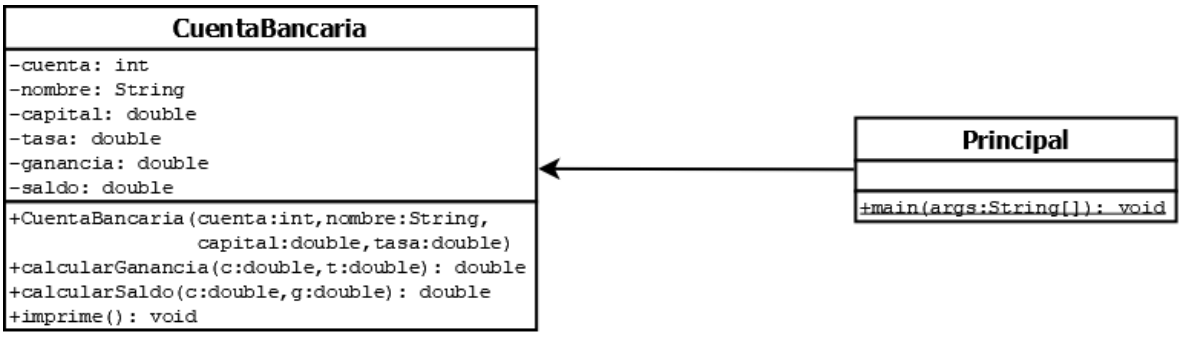



***Figura 3.4.3 Diagrama de clases de la clase CuentaBancaria y la clase Principal***

Ahora vamos a escribir el código de la clase CuentaBancaria. Primero declaramos la clase y los atributos de ésta:

```
public class CuentaBancaria
{
   private int cuenta;
   private String nombre;
   private double capital;
   private double tasa;
```

```
   private double ganancia;
   private double saldo;
```

En seguida vamos a declarar el constructor que pide parámetros, con los atributos que va a ingresar el cliente: cuenta, nombre, capital y tasa:

```
   public  CuentaBancaria(int cuenta, String nombre, double capital, double tasa)
   {
      this.cuenta=cuenta;
      this.nombre=nombre;
      this.capital=capital;
      this.tasa=tasa;
   }
```

Vamos ahora a escribir los métodos para calcular la ganancia y el saldo:

```
public double calculaGanancia(double c,double t)
   {
      ganancia=c*t;
      return ganancia;
   }

   public double calculaSaldo(double c, double g)
   {
      saldo=c+g;
      return saldo;
   }
```

Y finalmente el método que imprime los datos completos del cliente:

```
   public void imprime( )
   {
      System.out.println("Número de cuenta: "+cuenta);
      System.out.println("Nombre: "+nombre);
      System.out.println("Capital invertido: "+capital);
      System.out.println("Tasa de interés anual: "+tasa);
      System.out.println("Ganancia: "+ganancia);
      System.out.println("Saldo final: "+saldo);
   }
}
```

El código completo de la clase CuentaBancaria se muestra a continuación:

```
public class CuentaBancaria
{
   private int cuenta;
   private String nombre;
   private double capital;
   private double tasa;
   private double ganancia;
```

```
  private double saldo;
  //Constructor que pide parámetros
  public  CuentaBancaria(int cuenta, String nombre, double capital, double tasa)
  {
     this.cuenta=cuenta;
     this.nombre=nombre;
     this.capital=capital;
     this.tasa=tasa;
  }
  //Método que calcula la ganancia
  public double calculaGanancia(double c,double t)
  {
     ganancia=c*t;
     return ganancia;
  }
  //Método que calcula el saldo
  public double calculaSaldo(double c, double g)
  {
     saldo=c+g;
     return saldo;
  }
  //Método que imprime los atributos
  public void imprime( )
  {
     System.out.println("Número de cuenta: "+cuenta);
     System.out.println("Nombre: "+nombre);
     System.out.println("Capital invertido: "+capital);
     System.out.println("Tasa de interés anual: "+tasa);
     System.out.println("Ganancia: "+ganancia);
     System.out.println("Saldo final: "+saldo);
  }
}
```

Ahora vamos a escribir el código de la clase Principal. Primero importamos la biblioteca con java.util.Scanner porque vamos a pedir datos por teclado y después declaramos la clase.

```
import java.util.Scanner;
public class Principal
{
```

En seguida declaramos el método principal, un objeto de la clase Scanner y limpiamos la pantalla.

```
public static void main(String[ ] args) {
      Scanner teclado=new Scanner(System.in);
      System.out.print("\u000C");
```

Ahora vamos a pedir los datos del cliente y guardarlos en variables:

```
System.out.print("Escribe el número de cuenta:");
    int cuenta=teclado.nextInt( );
    System.out.print("Escribe el nombre:");
    String nombre=teclado.nextLine( );
    System.out.print("Escribe el capital invertido:");
    double capital=teclado.nextDouble( );
    System.out.print("Escribe la tasa de interés:");
    double tasa=teclado.nextDouble( );
```

Vamos a crear un objeto de la clase CuentaBancaria con el constructor que pide parámetros y utilizaremos las variables anteriores.

```
CuentaBancaria cliente=new CuentaBancaria(cuenta,nombre,capital,tasa);
```

Finalmente llamamos a los métodos de la clase CuentaBancaria para calcular la ganancia, el saldo e imprimir los datos del cliente.

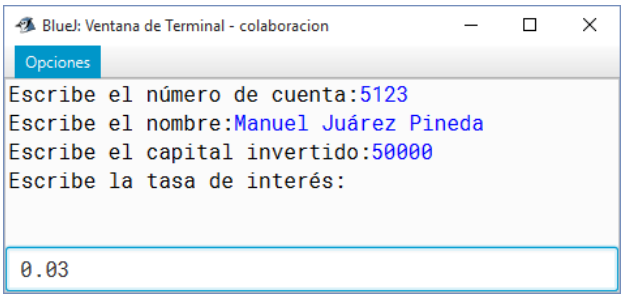
```
double ganancia=cliente.calculaGanancia(capital,tasa);
    double saldo=cliente.calculaSaldo(capital, ganancia);
    cliente.imprime( );
```

En seguida se muestra el código completo de la clase Principal.

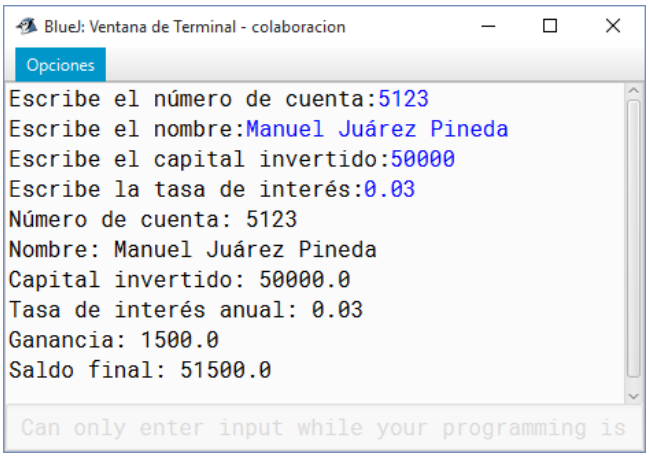
```
import java.util.Scanner;
public class Principal
{
   public static void main(String[ ] args) {
      Scanner teclado=new Scanner(System.in);
      //Limpiamos pantalla
      System.out.print("\u000C");
      //Pedimos los datos y los guardamos en variables
      System.out.print("Escribe el número de cuenta:");
      int cuenta=teclado.nextInt( );
      System.out.print("Escribe el nombre:");
      String tmp=teclado.nextLine( ); //sirve para limpiar el buffer
      String nombre=teclado.nextLine( );
      System.out.print("Escribe el capital invertido:");
      double capital=teclado.nextDouble( );
      System.out.print("Escribe la tasa de interés:");
      double tasa=teclado.nextDouble( );
      //Creamos un objeto de la clase CuentaBancaria con el constructor
      CuentaBancaria cliente=new CuentaBancaria(cuenta,nombre,capital,tasa);
      //Calculamos la ganancia y el saldo con los métodos de CuentaBancaria
      double ganancia=cliente.calculaGanancia(capital,tasa);
      double saldo=cliente.calculaSaldo(capital, ganancia);
```

```
  //Imprimimos los datos del cliente con el método imprime( )
      cliente.imprime( );
   }
}
```

Al ejecutar la clase Principal tenemos la siguiente salida del programa. Primero pide los datos de la cuenta.




Y en seguida muestra los datos ingresados y la ganancia y saldo final calculados.




## 3.5 Herencia

**Propósito:**

El alumno: comprenderá los beneficios del uso de la herencia para el desarrollo de programas y su implementación.

La Programación Orientada a Objetos permite generalizar características y comportamientos de clases para su asociación. A esta generalización se le conoce como Herencia, esto es, ampliar o extender la funcionalidad de una clase, derivando de ella nuevas clases.

- **Herencia**

La Herencia es una característica de la Programación Orientada a Objetos, la cual hace posible que se pueda definir una clase a partir de otra ya existente, logrando reutilizar código y jerarquización de clases.

Por ejemplo, tenemos la clase Persona

```
public class Persona
{



}
```

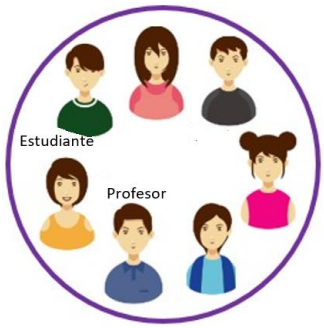



***Figura 3.5.1. Ejemplo de clase Persona***

Y como se puede ver en la figura 3.6.1 dentro de esa clase existen personas con diferentes roles, en este caso Profesor y Estudiante. Por ello se pueden crear las clases Profesor y Estudiante derivadas de la clase Persona.

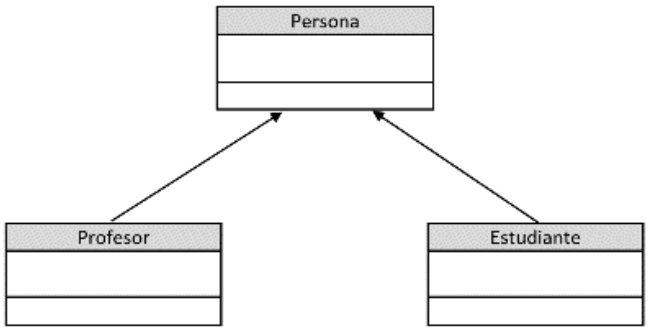



***Figura 3.5.2 Esquema de clases***

Esto quiere decir que:

El profesor **es una** Persona
El estudiante **es una** Persona

También, la herencia es el proceso mediante el cual una clase adquiere las propiedades (atributos) y comportamiento (métodos) de otra, haciendo también posible añadir nuevos elementos (atributos o métodos) o redefinir los elementos existentes.

Ejemplo:

Partiendo del ejemplo anterior, revisaremos los atributos que puede tener cada clase

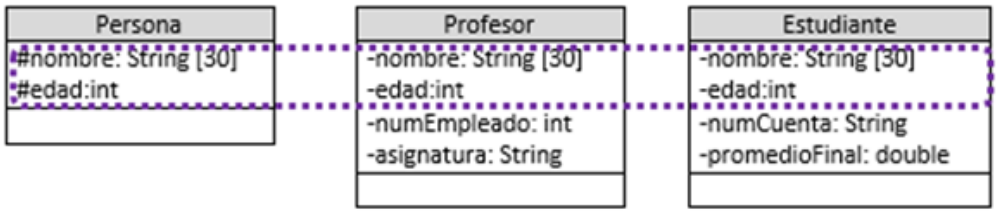



***Figura 3.5.3 Clases con sus atributos***

Como se observa en la figura anterior una Persona tiene como atributos un nombre y una edad, mientras que el Profesor y el Estudiante también cuentan con esos atributos, entre otros específicos.

Por lo anterior, es suficiente que cuente con los atributos nombre y edad la clase Persona, ya que las clases derivadas van a adquirir los atributos de la clase Persona, como se muestra a continuación:

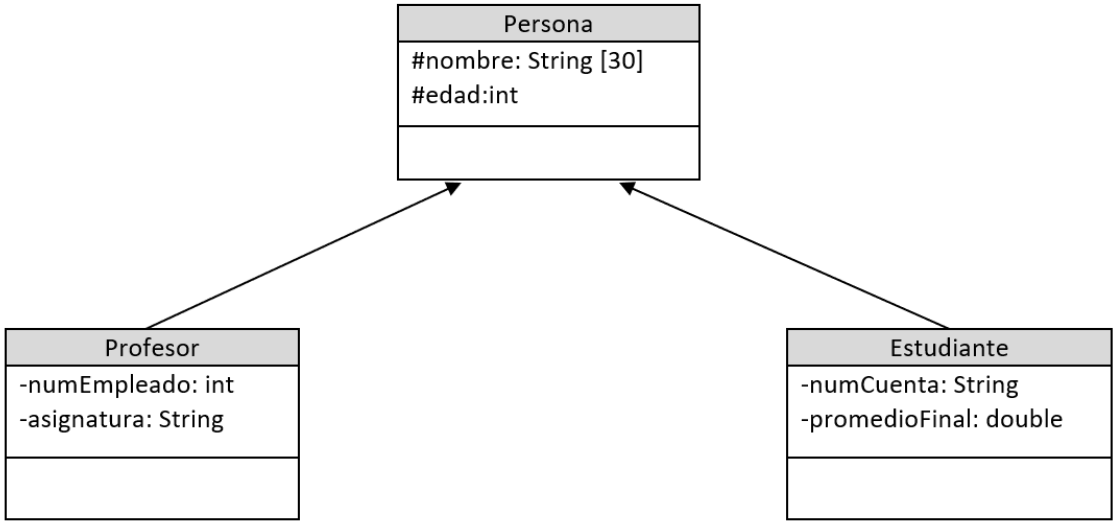



***Figura 3.5.4 Clases derivadas y atributos***

- **Superclase y Subclases**

El concepto de herencia conduce a una estructura jerárquica de clases o estructura de árbol, lo cual significa que en la Programación Orientada a Objetos todas las relaciones entre clases deben ajustarse a dicha estructura. En esta estructura jerárquica, se identifica la superclase y las subclases.

**Ejemplo:**

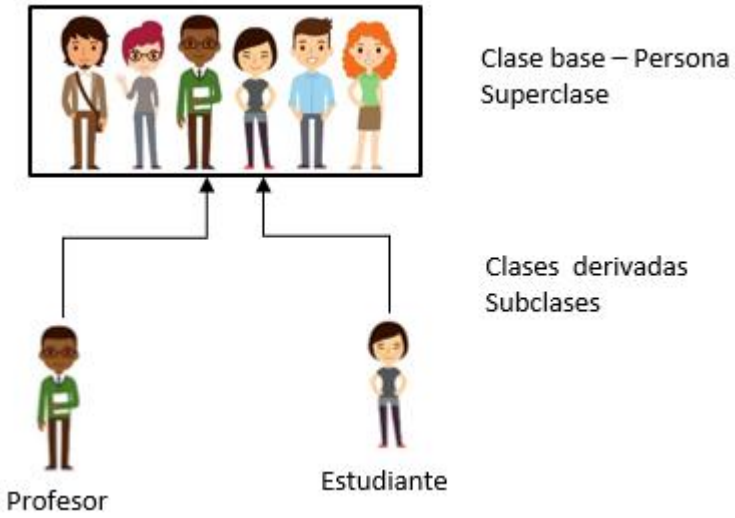



***Figura 3.5.5 Jerarquía de Clases***

En la figura anterior se puede apreciar la jerarquía entre clases.

Una **superclase** tiene otras denominaciones como clase padre o clase base, está puede tener cualquier número de subclases. Cuentan con sus propios métodos y atributos que pueden servir de base para la creación de otras clases (subclase). Por ejemplo, la clase Persona servirá de base para crear a las subclases Profesor y Estudiante.

Una **subclase**, también es conocida como clase hija o clase derivada, esta tiene solo una superclase. Las subclases tienen acceso a todos los atributos y métodos *public* y *protected* de la surperclase, no a los *private*. Asimismo, puede contar con atributos o métodos específicos o redefinir elementos existes.

El siguiente esquema muestra la jerarquía entre clases, en donde se puede visualizar la superclase y subclases; además de visualizar la relación entre ellas.

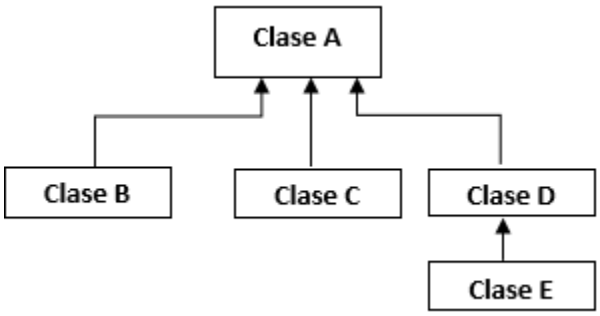



***Figura 3.5.6 Esquema de jerarquía de Clases***

- A es la superclase de B, C y D.
- D es la superclase de E.
- B, C y D son subclases de A.
- E es una subclase de D.

En este esquema de clases se puede apreciar que la herencia es transitiva, es decir, una clase puede heredar características de superclases que se encuentran muchos niveles más arriba en la jerarquía de herencia. En este caso, la Clase E hereda lo de la Clase D y lo de la Clase A.

- ## Ventajas

Entre las ventajas de la herencia se tiene:

Reutilización del código: La herencia evita escribir el mismo código varias veces.

Mantenimiento de aplicaciones existentes: Utilizando la herencia, si tenemos una clase con una determinada funcionalidad y tenemos la necesidad de ampliar dicha funcionalidad, no necesitamos modificar la clase existente (la cual se puede seguir utilizando para el tipo de programa para la que fue diseñada) sino que podemos crear una clase que herede a la primera, adquiriendo toda su funcionalidad y añadiendo la suya propia.

- ## Sintaxis

La herencia se expresa mediante la palabra ***extends***. Por ejemplo, para declarar la clase B que hereda de una clase A, se muestra el siguiente código:

```
public class B extends A
{
….
}
```

Como se puede ver en la sintaxis anterior a las clases derivadas o subclases se les agrega la palabra reservada *extends* y se indica el nombre de la superclase.

**Ejemplo.**
Si tenemos a la superclase Persona y se crean las subclases Profesor y Estudiante con sus respectivos atributos como se puede visualizar en la figura 3.6.4, tenemos:

```
public class Persona
{
   protected String nombre;
   protected int edad;
 }
```

```
public class Profesor extends Persona
{
   private int numEmpleado;
   private String asignatura;
}
```

```
public class Estudiante extends Persona
{
   private String  numCuenta;
   private double promedioFinal;
}
```

- **Palabras reservadas super y this**

Como se ha mencionado la herencia también permite que las subclases adquieran los métodos de la superclase. Veamos los posibles métodos, para cada clase.

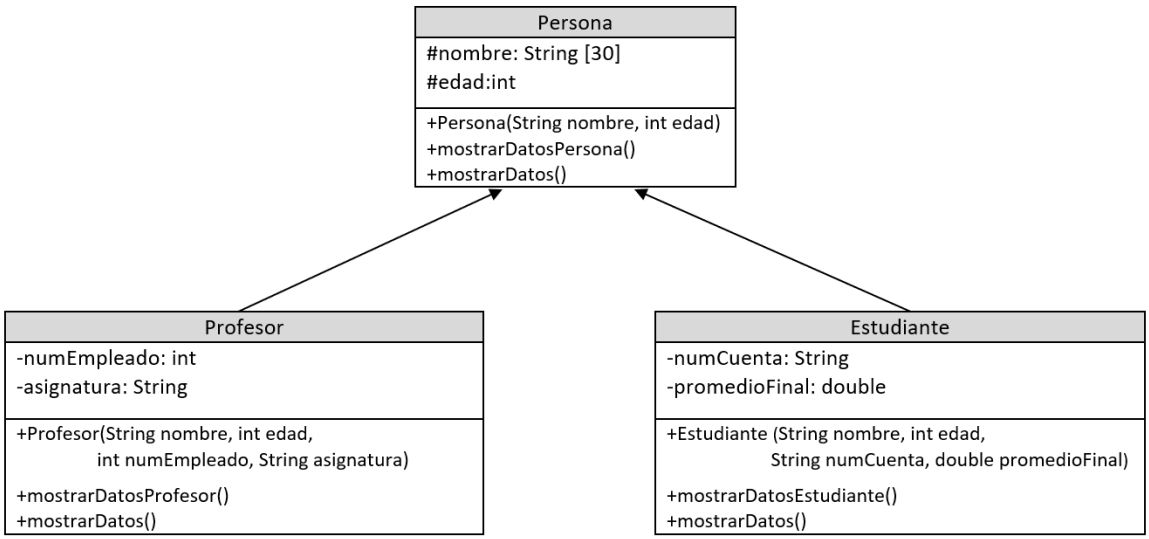



***Figura 3.5.7 Esquema de Clases y métodos***

La palabra reservada super se utiliza para hacer referencia al constructor de la clase padre. Se debe tener en cuenta que los constructores no se heredan.

Por ejemplo:

Para crear el constructor de la clase Profesor, vamos a hacer referencia a los atributos nombre y edad, los cuales estamos heredando de la clase Persona, para ello, hacemos uso de la palabra reservada **super**.

```
public Profesor(String nombre, int edad)
{
      super(nombre, edad);
 }
```

La palabra **this** se utiliza para hacer referencia a los constructores de la misma clase. Se pueden invocar tanto atributos como métodos.

**Ejemplo:**
Para completar el constructor de la clase Profesor, colocamos los atributos específicos de esa clase, para ello hacemos uso de la palabra reservada this.

```
public Profesor(String nombre, int edad, int numEmpleado, String asignatura)
{
      super(nombre, edad);
      this.numEmpleado = numEmpleado;
      this.asignatura = asignatura;
 }
```

Como se puede observar en la figura 3.5.7 cada una de las clases tiene el método *mostrarDatos* los cuales imprimen los atributos específicos de cada clase.

**Ejemplos:**

Para la clase Persona tenemos:

```
public void mostrarDatosPersona( )
   {
      System.out.println("Nombre: " + nombre  + "\nEdad: " + edad);
   }
```

Para la clase Estudiante, se tiene:

```
public void mostrarDatosEstudiante( )
   {
```

```
        System.out.println("Numero de Cuenta: " + numCuenta
            +"\nPromedio Final: " + promedioFinal);
    }
```

Asimismo, en la figura 3.5.7, se observa que la superclase y las subclases tienen un método con el mismo nombre *mostrarDatos*. Esto es posible porque la herencia nos permite redefinir elementos ya existentes, es decir, a pesar de que las subclases hereden el método *mostrarDatos*, cada subclase está redefiniendo este método dentro de sí misma. En resumen, las subclases heredan la funcionalidad de la superclase y agregan su propia funcionalidad.

**Ejemplos.**

La clase Persona cuenta con dos atributos los cuales se muestran en este método.

```
public void mostrarDatos( )
    {
        System.out.println("Nombre: " + nombre  + "\nEdad: " + edad);
    }
```

La clase Estudiante cuenta con más atributos, por ello, hace una redefinición del método, esto es, cambia la funcionalidad mostrando todos los atributos los heredados y los propios.

```
public void mostrarDatos( )
    {
        System.out.println("Estudiante: " + nombre
            + "\nEdad: " + edad
            + "\nNumero de Cuenta: " + numCuenta
            +"\nPromedio Final: " + promedioFinal);
    }
```

Al contar con la implementación de todas las clases (superclases y subclases), necesitamos de una clase que las invoque, por eso construiremos una clase llamada Principal, la cual contendrá el método *main*.

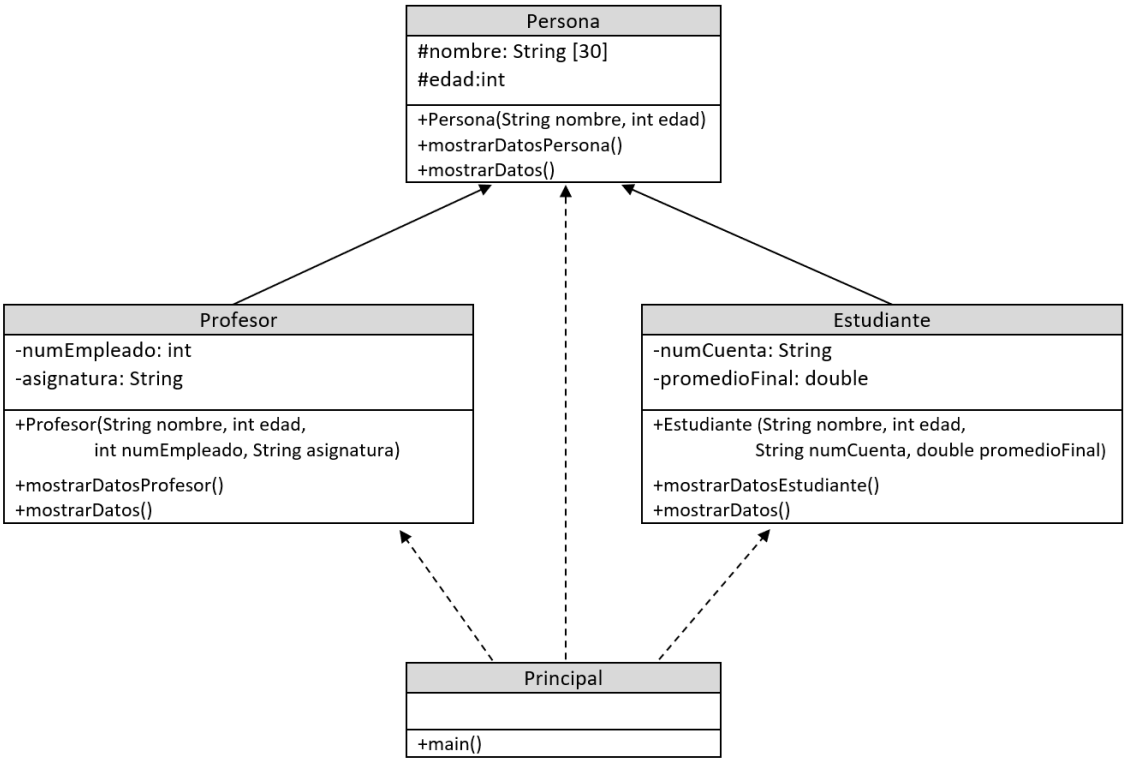



***Figura 3.5.8 Esquema de Clases con la clase Principal.***

En la clase Principal se hará el llamado de las diferentes clases y sus métodos. A continuación, se muestra el código:

```
public class Principal
{

   public static void main(String[ ] args)
   {
      //Invocando a la clase Persona
      System.out.println("\n Uso de la clase Persona");
      //Creando el objeto per1 y dando valor a sus parámetros nombre y edad
      Persona per1 = new Persona("Jorge",17);
       //Invocando a los métodos de la clase Persona
      per1.mostrarDatosPersona( );
      per1.mostrarDatos( );

      //Invocando a la clase Profesor
      System.out.println("\nInstanciando Profesor");
      //Creando el objeto profe1 y dando valor a sus parámetros
       Profesor profe1=new Profesor("Socorro", 25, 80200, "Cibernetica y
Computación");
     //Invocando al método de la clase Persona
      profe1.mostrarDatosPersona( );
```

```
        //Invocando al método de la clase Profesor
         profe1.mostrarDatosProfesor( );
        //Invocando al método de la clase Profesor
         profe1.mostrarDatos( );

       //Invocando a la clase Estudiante
        System.out.println("\nInstanciando Estudiante");
        //Creando el objeto est1 y dando valor a sus parámetros
         Estudiante est1= new Estudiante("María", 17, "317123456", 8.5);
        //Invocando al método de la clase Persona
          est1.mostrarDatosPersona( );
         //Invocando al método de la clase Estudiante
          est1.mostrarDatosEstudiante( );
        //Invocando al método de la clase Estudiante
          est1.mostrarDatos( );
     }

  }
```

***Ejemplo:***

Para ejemplificar las definiciones anteriores, se realizará el programa completo del diagrama de clases de la figura 3.6.8. Para ello, se hará uso del código que se utilizó como ejemplo para cada definición.

Superclase Persona

```
public class Persona
{
  //Declarando los atributos específicos de la clase
   protected String nombre;
   protected int edad;

   //Constructor
   public Persona(String nombre,int edad)
   {
      this.nombre = nombre;
      this.edad = edad;
   }

   public void mostrarDatosPersona( )
   {
         System.out.println("\n *** Método mostrarDatosPersona de la clase Persona ***");
         System.out.println("\t Nombre: " + nombre + "\n\t Edad: " + edad);
   }
   public void mostrarDatos( )
   {
         System.out.println("*** Método mostrarDatos de la clase Persona ***");
         System.out.println("\t Nombre: " + nombre + "\n\t Edad: " + edad);
   }
```

```
}
```

Subclase Profesor

```
public class Profesor extends Persona
{
  //Declarando los atributos específicos de la clase
  private int numEmpleado;
   private String asignatura;

   //Constructor con parámetros
   public Profesor(String nombre, int edad, int numEmpleado, String asignatura)
   {
      //Haciendo referencia de los atributos de la superclase
      super(nombre, edad);
     //Haciendo referencia de los atributos de la subclase Profesor
      this.numEmpleado = numEmpleado;
      this.asignatura = asignatura;
   }

//Definiendo el método mostrarDatosProfesor con atributos específicos
   public void mostrarDatosProfesor( )
   {
         System.out.println("\n *** Método mostrarDatosProfesor de la clase Profesor,
mostrando sólo sus atributos específicos ***");
         System.out.println("\t Numero de Empleado: " + numEmpleado
        + "\n\t Asignatura: " + asignatura);
   }

   //Redefine el metodo mostrarDatos
   public void mostrarDatos( )
   {
        System.out.println("\n *** Redefiniendo el método mostrarDatos con todos los atributos
de la clase Profesor ***");
        System.out.println("\t Profesor: " + nombre
        + "\n\t Edad: " + edad
        + "\n\t Numero de Empleado: " + numEmpleado
        + "\n\t Asignatura: " + asignatura);
   }
}
```

Subclase Estudiante

```
public class Estudiante extends Persona
{
  //Declarando los atributos específicos de la clase
   private String numCuenta;
```

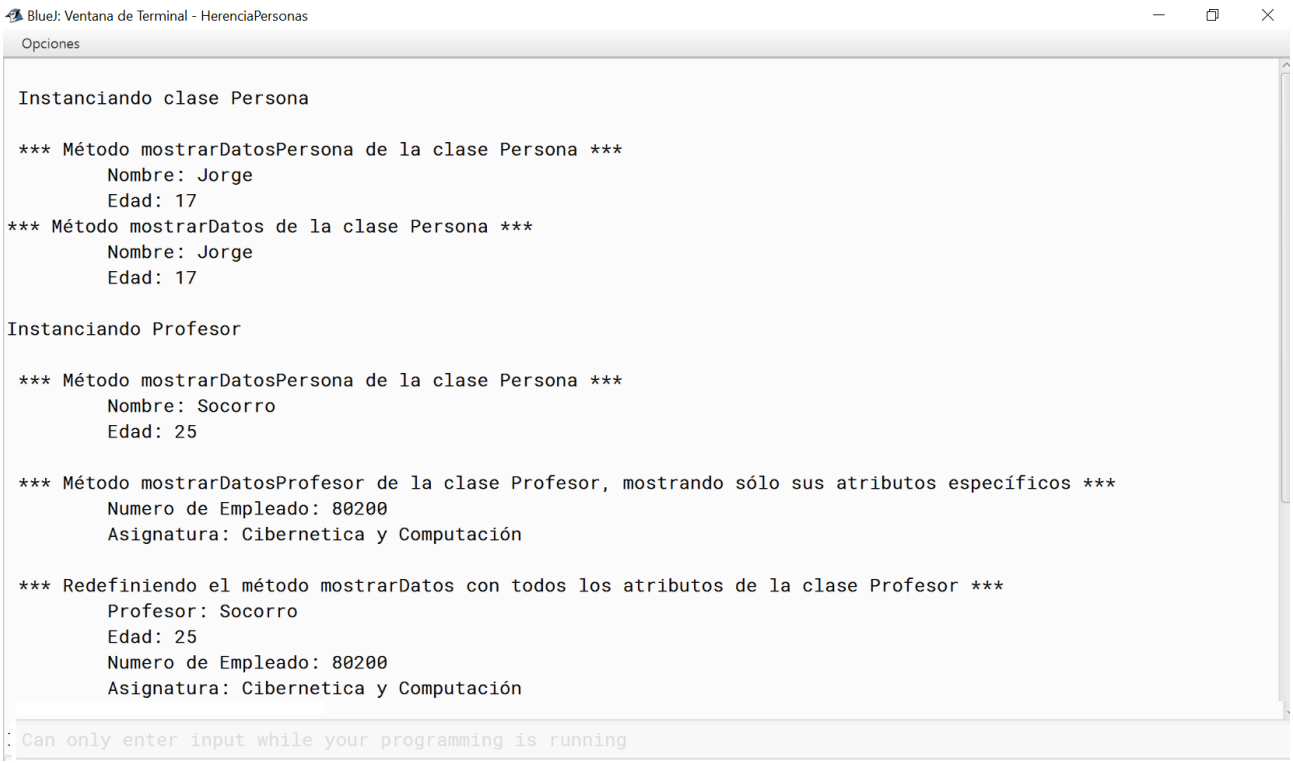
```
   private double promedioFinal;

   //Constructor con parámetros
   public Estudiante(String nombre, int edad, String numCuenta, double promedioFinal)
   {
      //Haciendo referencia de los atributos de la superclase
      super(nombre, edad);
     //Haciendo referencia de los atributos de la subclase Profesor
      this.numCuenta = numCuenta;
      this.promedioFinal = promedioFinal;
   }

//Definiendo el método mostrarDatosEstudiante con atributos específicos
   public void mostrarDatosEstudiante( )
   {
         System.out.println("\n *** Método mostrarDatosProfesor de la clase Estudiante,
mostrando sólo sus atributos específicos ***");
          System.out.println("\t Numero de Cuenta: " + numCuenta
          +"\n\t Promedio Final: " + promedioFinal);
   }

   //Redefine el metodo mostrarDatos
   public void mostrarDatos( )
   {
        System.out.println("\n *** Redefiniendo el método mostrarDatos con todos los atributos
de la clase Estudiante ***");
        System.out.println("\t Estudiante: " + nombre
        + "\n\t Edad: " + edad
        + "\n\t Número de Cuenta: " + numCuenta
        +"\n\t Promedio Final: " + promedioFinal);
   }
}
```

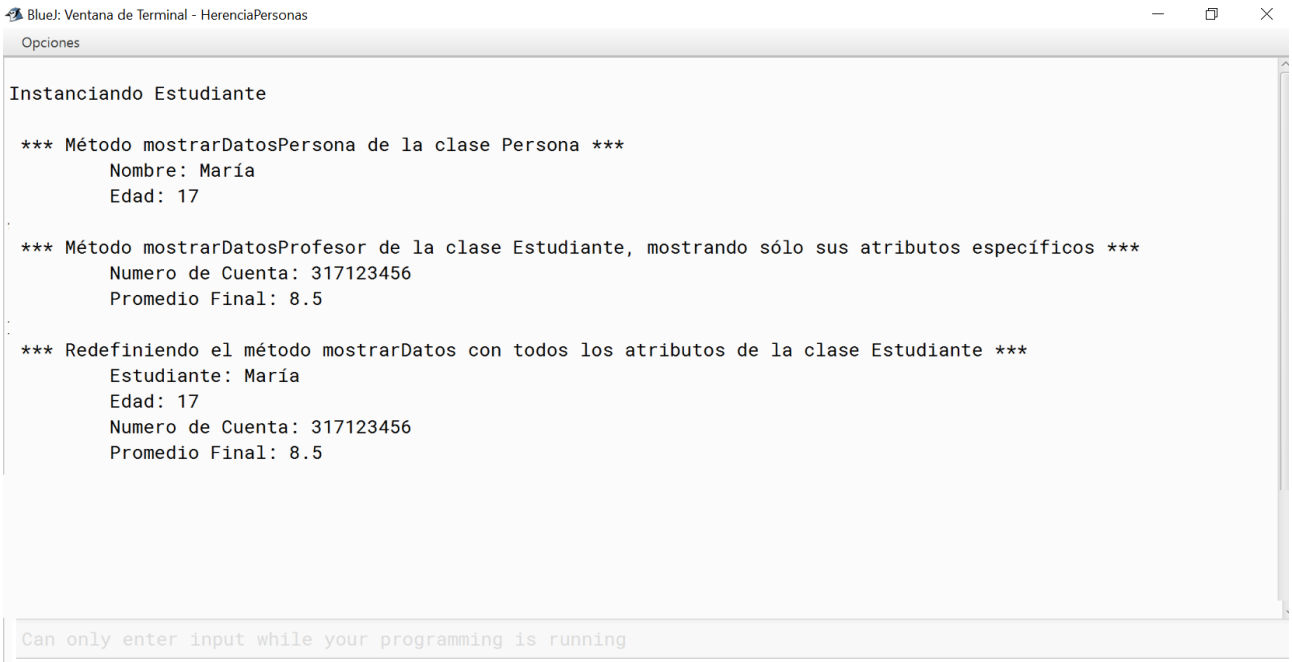
Clase Principal

```
public class Principal
{
   public static void main(String[ ] args)
   {
      //Invocando a la clase Persona
      System.out.println("\n Instanciando clase Persona");
      //Creando el objeto per1 y dando valor a sus parámetros nombre y edad
      Persona per1 = new Persona("Jorge",17);
       //Invocando a los métodos de la clase Persona
      per1.mostrarDatosPersona( );
      per1.mostrarDatos( );
```

```
      //Invocando a la clase Profesor
      System.out.println("\nInstanciando Profesor");
      //Creando el objeto profe1 y dando valor a sus parámetros
      Profesor profe1=new Profesor("Socorro", 25, 80200, "Cibernetica y Computación");
      //Invocando al método de la clase Persona
      profe1.mostrarDatosPersona( );
      //Invocando al método mostrarDatosProfesor de la clase Profesor
      profe1.mostrarDatosProfesor( );
      //Invocando al método mostrarDatos de la clase Profesor
      profe1.mostrarDatos( );

      //Invocando a la clase Estudiante
      System.out.println("\nInstanciando Estudiante");
     //Creando el objeto est1 y dando valor a sus parámetros
      Estudiante est1= new Estudiante("María", 17, "317123456", 8.5);
     //Invocando al método de la clase Persona
      est1.mostrarDatosPersona( );
      //Invocando al método mostrarDatosEstudiante de la clase Estudiante
      est1.mostrarDatosEstudiante( );
     //Invocando al método mostrarDatos de la clase Estudiante
      est1.mostrarDatos( );
   }
}
```

## Ejecución






## Actividad 3.4

### Cuestionario

1. Permite la definición de una clase a partir de otra ya existente.
   a) Polimorfismo b) Subclase c) Superclase d) Herencia

2. Utiliza el código y jerarquización de clases.
   a) Polimorfismo b) Subclase c) Superclase d) Herencia

3. Sus atributos y métodos pueden servir de base para la creación de otras clases.
   a) Superclase b) Subclase c) Clase d) Atributo

4. Tienen acceso a todos los atributos y métodos public y protected, pero no a los private.
   a) Superclase b) Subclase c) Clase d) Atributo

5. Es la palabra reservada que permite definir una subclase.
   a) this b) extends c) super d) private

6. Utiliza la palabra reservada extends.
   a) Polimorfismo. b) Subclase c) Superclase d) Método

7. Esta palabra se utiliza para hacer referencia a los atributos de la misma clase.
   a) this b) extends c) super d) private

8. Se utiliza para hacer referencia a los atributos de la clase padre.
   a) this b) extends c) super d) private

9. El nombre de este método existe en la superclase y en la subclase a esto se le llama.
   a) sobrecarga b) extends c) herencia d) redefinición

10. Es una propiedad de la herencia mediante la cual una superclase hereda sus atributos y métodos a las subclases y a las subclases que se deriven de estas.
    a) sobrecarga b) extends c) transitividad d) redefinición

### Actividad 3.5

Relaciona las siguientes columnas.

Los atributos y métodos que tienen este modificador de aceso:

| | | |
|---|---|---|
| 1. No se heredan | ( ) | a) public |
| 2. Se heredan | ( ) | b) private |
| 3. Todas las clases tienen acceso | ( ) | c) protected |

### Actividad 3.6

Relaciona las siguientes columnas.

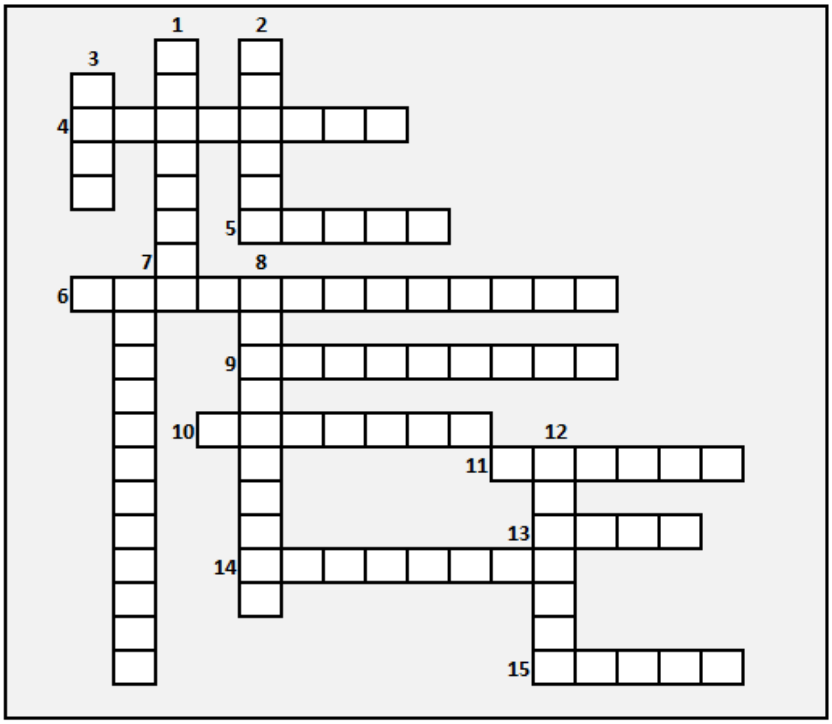
Las siguientes palabras reservadas sirven para:

| | | |
|---|---|---|
| 1. Referenciar atributos de la clase padre | ( ) | a) this |
| 2. Heredar los atributos y métodos de la superclase | ( ) | b) extends |
| 3. Referenciar atributos de la misma clase | ( ) | c) super |

### Actividad 3.7

**Crucigrama**

Instrucciones. Verifica la definición de cada número en vertical u horizontal para colocar el concepto dentro de las celdas correspondientes.




Palabras clave:

Herencia, clase, método, atributo, superclase, subclase, this, extends, super, redefinición, transitividad, protected, private, public, main.

| VERTICALES | | HORIZONTALES | |
|---|---|---|---|
| 1. | Permite la definición de una clase a partir de otra ya existente. | 4. | Es una característica de un objeto. |
| 2. | Este modificador de acceso permite modificar los atributos de una clase desde cualquier clase. | 5. | Es la plantilla que sirve para instanciar objetos. |
| 3. | Es el método desde donde se comienza a ejecutar una clase. | 6. | Es una propiedad de la herencia mediante la cual una superclase hereda sus atributos y métodos a las subclases y a las subclases que se deriven de estas. |
| 7. | El nombre de este método existe en la superclase y en la subclase a esto se le llama. | 9. | Este modificador de acceso permite heredar los atributos de una clase. |
| 8. | Sus atributos y métodos pueden servir de base para la creación de otras clases. | 10. | Este modificador de acceso no deja modificar los atributos de una clase. |
| 12. | Es la palabra reservada que permite definir una subclase. | 11. | Proporciona funcionalidad a las clases. |

| VERTICALES | HORIZONTALES |
|---|---|
| | 13. Esta palabra se utiliza para hacer referencia a los atributos de la misma clase. |
| | 14. Tienen accesos a todos los atributos y métodos public y protected pero no a los private. |
| | 15. Se utiliza para hacer referencia a los atributos de la clase padre. |

## 3.6 Implementación de la herencia de clase

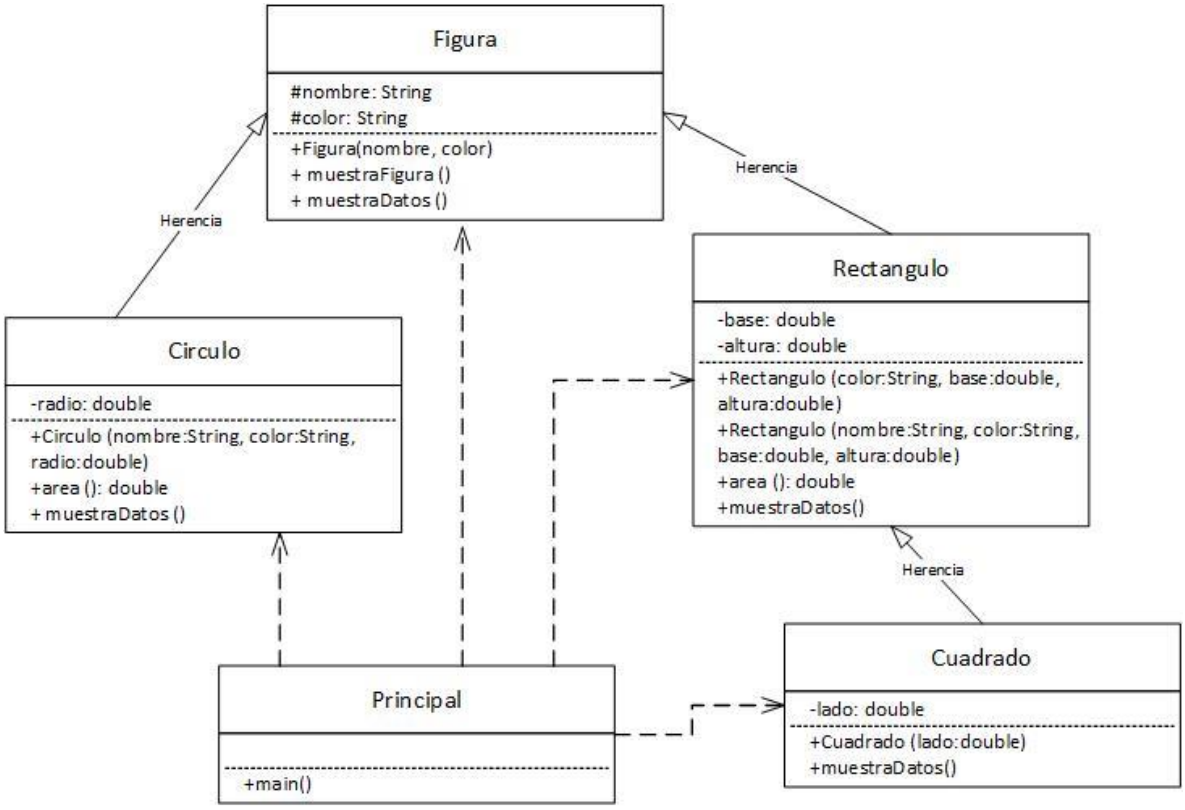
**Propósito:**

Desarrolla programas que involucren la herencia de Clases.

Se plantea la situación de realizar varias clases donde se tengan descritas varias figuras geométricas en las cuales se definan propiedades y métodos para realizar ciertas acciones de las figuras geométricas como el cálculo del área y mostrar sus datos.

Comenzamos definiendo la clase figura que tiene solamente dos atributos, uno para el nombre y otro para el color. Definiremos dos métodos para mostrar los datos de la figura y uno de ellos será redefinido en las subclases que los heredan.

Luego vamos a definir otras tres clases, una por cada figura geométrica que vamos a trabajar, las cuales serán: círculo, rectángulo y cuadrado. En cada una se heredarán los atributos y el método de la clase figura y para círculo y rectángulo se definirá un método propio para el cálculo del área, en el caso del cuadrado el cálculo del área se heredará del rectángulo. Finalmente, cada figura redefinirá el método para mostrar sus propios datos.



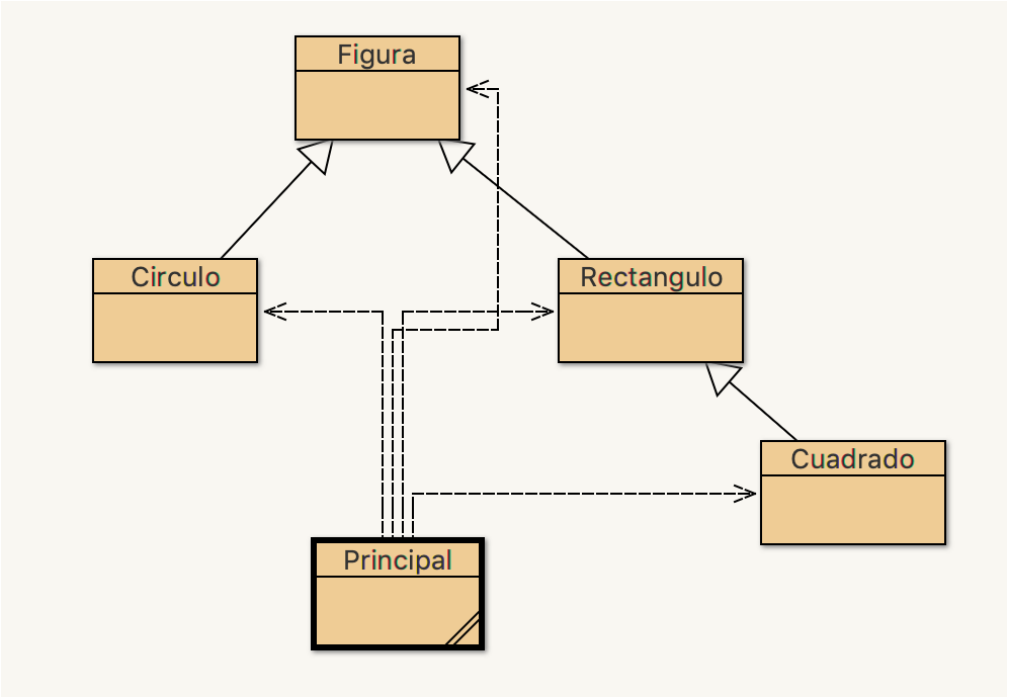




***Fig. 3.6.1. Diagrama de clases.***

Definimos la clase *Figura* con dos atributos de tipo String, nombre y color. El modificador de acceso se pone como *protected* para poder heredarlos a las subclases y poder modificarlos desde estas.

```
public class Figura
{
   protected String nombre;
   protected String color;

   //Constructor de la clase Figura
   public Figura(String nombre, String color)
   {
      this.nombre = nombre;
      this.color = color;
   }
   //Método para mostrar los datos de Figura
   public void muestraFigura( )
   {
      System.out.println("La figura es un(a) "+nombre+" de color "+color);
   }
   //Método para mostrar datos, este método será redefinido en las clases hijas
  public void muestraDatos( )
  {
      System.out.println("La figura es un(a) "+nombre+" de color "+color);
  }
}
```

Definimos la clase *Circulo* que extiende de la clase *Figura* con un atributo de tipo *double*, radio. Todos los círculos tienen radio. Tiene el método *area* que devuelve el área del círculo. Se redefine el método *muestraDatos* para los datos del círculo.

```
public class Circulo extends Figura
{
   private double radio;
   //Constructor de la clase Circulo
   public Circulo(String nombre,String color,double radio)
   {
      super(nombre,color);
      this.radio=radio;
   }
   //Método para calcular el érea del círculo
   public double area( )
   {
      return (Math.PI*Math.pow(radio, 2));
   }
   //Redefine el método muestraDatos para Circulo
  public void muestraDatos( )
  {
      System.out.println("Este " + nombre+ " de color "+color
```

```
            + " tiene un radio de " + radio);
    }
}
```

Definimos la clase *Rectangulo* que extiende de la clase *Figura* con dos atributos de tipo *double*, base y altura. Todos los rectángulos tienen base y altura. Se definen dos constructores para *Rectangulo* el primero no recibe el nombre porque se entiende que se trata de un "Rectángulo", en el otro se puede especificar algo con figura rectangular como un terreno o una mesa, etcétera. También tiene un método *area* propio del rectángulo y redefine el método *muestraDatos*.

```
public class Rectangulo extends Figura
{
   private double base;
   private double altura;

   //Constructor de Rectangulo
   public Rectangulo(String color, double base, double altura)
   {
      super("Rectangulo",color);
      this.base=base;
      this.altura=altura;
   }
   //Sobrecarga el constructor de Rectangulo, recibiendo además el nombre
   public Rectangulo(String nombre,String color,double base, double altura)
   {
      super(nombre,color);
      this.base=base;
      this.altura=altura;
   }
   //Método para calcular el área del rectángulo
   public double area( )
   {
      return (base*altura);
   }
   //Método para mostrar lo datos del rectángulo
  public void muestraDatos( )
  {
      System.out.println("Este " + nombre + " de color "+color
            +" tiene una base de " + base + " y una altura de " + altura);
  }
}
```

Definimos la clase *Cuadrado* que extiende de *Rectangulo* como un caso particular con un atributo de tipo double, lado. Todos los cuadrados tienen lado. El constructor de *Cuadrado* solamente recibe la medida del lado y lo envía hacia la clase padre que es el *Rectangulo*

donde base y altura son iguales ya que el cuadrado es un caso particular del rectángulo. El área se calcula en la superclase *Rectangulo*. Y se redefine el método *muestraDatos* para adaptarlo al cuadrado.

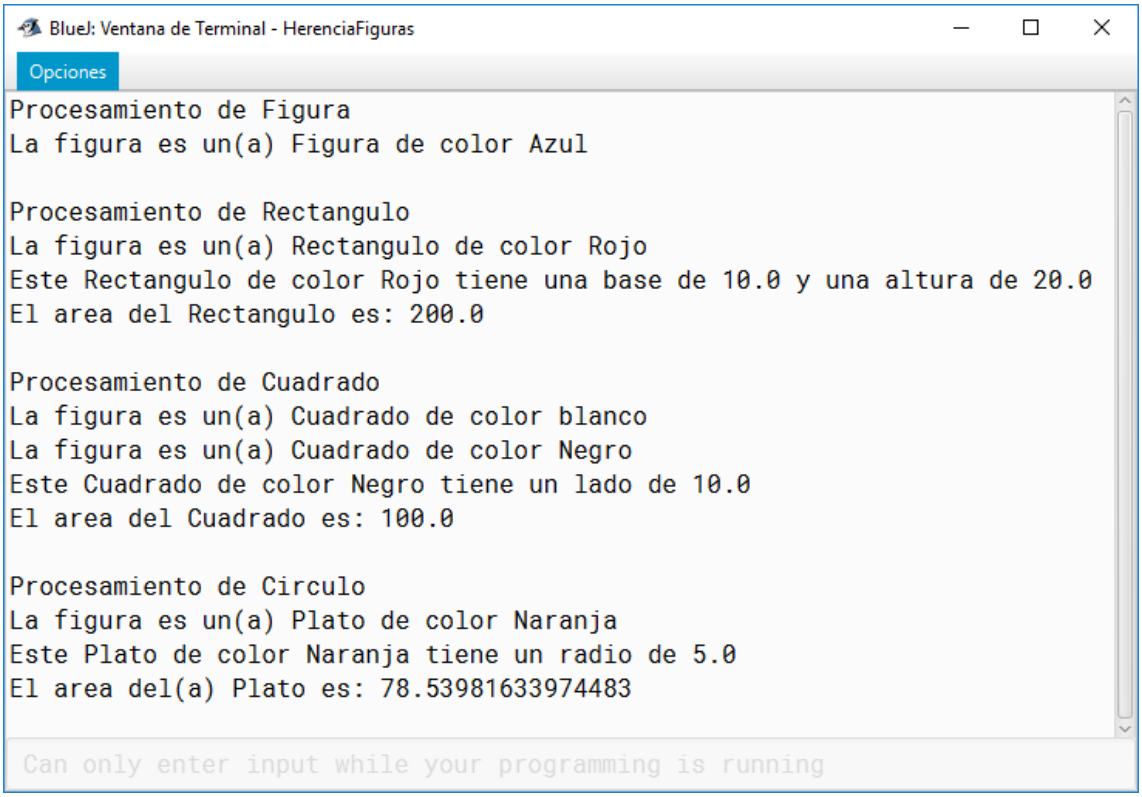
```
public class Cuadrado extends Rectangulo
{
   private double lado;

   //Constructor de la clase Cuadrado
   public Cuadrado(double lado)
   {
      super("Cuadrado","blanco",lado,lado);
      this.lado=lado;
   }
   //Método que muestra los datos del cuadrado
   public void muestraDatos( )
   {
      System.out.println("Este "+nombre+ " de color "+color
            +" tiene un lado de "+lado);
   }
}
```

La clase Principal que utiliza las clases *Figura, Rectangulo, Cuadrado* y *Circulo*, utilizando los métodos de cada uno.

```
public class Principal
{
   public static void main(String[ ] args)
   {
      System.out.println("Procesamiento de Figura ");
      Figura fig=new Figura("Figura","Azul");
      fig.muestraFigura( );
      //Rectangulo
      System.out.println("\nProcesamiento de Rectangulo");
      Rectangulo rect=new Rectangulo("Rojo", 10, 20);
      rect.muestraFigura( );
      rect.muestraDatos( );
      System.out.println("El area del "+rect.nombre+" es: "+rect.area( ));
      //Cuadrado
      System.out.println("\nProcesamiento de Cuadrado");
      Cuadrado cuad=new Cuadrado(10);
      cuad.muestraFigura( );
      //cambiamos el color
      cuad.color="Negro";
      cuad.muestraFigura( );
      cuad.muestraDatos( );
      System.out.println("El area del "+cuad.nombre+" es: "+cuad.area( ));
      //Circulo
```

```
        System.out.println("\nProcesamiento de Circulo");
        Circulo circ=new Circulo("Plato","Naranja",5);
        circ.muestraFigura( );
        circ.muestraDatos( );
        System.out.println("El area del(a) "+circ.nombre+" es: "+circ.area( ));
    }
}
```

**Ejecución**




Se deja como retos para los alumnos con ayuda del profesor:

1- Realizar un método para calcular el perímetro de cada una de las figuras y realizar la modificación correspondiente en la clase *Principal* para mostrarlo después del área.

2- Nótese que en la clase *Figura* los atributos son *protected* y en las clases hijas los atributos son *private* por lo que solamente se asignan en los constructores y no pueden modificarse en los objetos instanciados. Hacer las modificaciones necesarias para poder modificarlos cuando ya han sido instanciados, así como podemos modificar el nombre o el color, modificar el radio, la base, la altura o el lado en cada caso respectivamente.

3- Reflexionar con el profesor que es mejor, si cambiar los modificadores de acceso a los atributos o crear métodos *setAtributo* para cada uno.

**Respuestas.**

Respuesta al reto número 1

Para la clase *Rectangulo:*

```
//Método para calcular el perímetro del rectángulo
public double perimetro( )
{
   return (2*base+2*altura);
}
```

Para la clase *Cuadrado* no es necesario porque se hereda de *Rectangulo*.

Para la clase *Circulo:*

```
//Método para calcular el perímetro del círculo
public double perimetro( )
{
   return (Math.PI*2*radio);
}
```

Respuesta al reto número 2

La solución es crear los métodos *Setter* para poder modificar los atributos. Otra opción es cambiar el modificador de acceso a *public*.

Respuesta al reto número 3

Si creamos los métodos *Setter* estaremos respetando el encapsulamiento, el cual es uno de los propósitos de la programación orientada a objetos. El cambiar el modificador de acceso soluciona el problema, pero estaremos en contra del encapsulamiento.

# Solución de las actividades

## Unidad 3

### Actividad 3.1


### Actividad 3.3

| | | | | |
|---|---|---|---|---|
| 1. g | 2. f | 3. h | 4. j | 5. i |
| 6. c | 7. b | 8. a | 9. d | 10. e |

### Actividad 3.4

| | | | | |
|---|---|---|---|---|
| 1. d | 2. d | 3. a | 4. b | 5. b |
| 6. b | 7. a | 8. c | 9. d | 10. c |

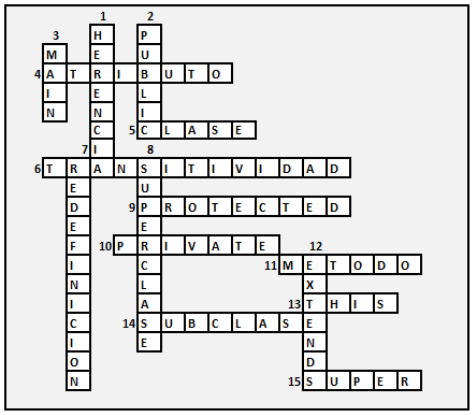
### Actividad 3.5

| | | |
|---|---|---|
| 1. b | 2. c | 3. a |

### Actividad 3.6

| | | |
|---|---|---|
| 1. c | 2. b | 3. a |

### Actividad 3.7



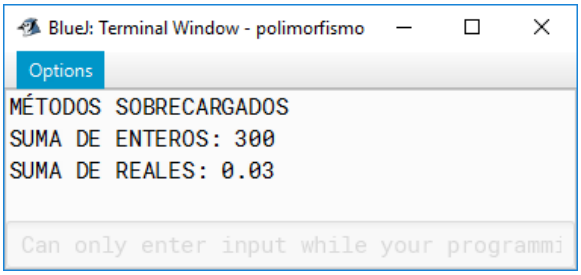

### Actividad 3.2

**Problema 1.**

```
public class Calculadora
{
   /*Método suma de enteros, pide dos parámetros de tipo entero y regresa un valor de tipo entero.
   */
   public int suma(int a, int b)
   {
      int suma=a+b;
```

```
    return suma;
  }
  /*Método suma de decimales, pide dos parámetros de tipo double y regresa un valor de tipo double.
  */
  public double suma(double a, double b)
  {
    double suma=a+b;
    return suma;
  }
  //Método principal
  public static void main(String[ ] args)
  {
    //Instanciamos un objeto de la clase con el constructor por default
    Calculadora calc=new Calculadora( );
    System.out.print("\u000C");
    System.out.println("MÉTODOS SOBRECARGADOS");
    //Llamamos al método suma de enteros y lo imprimimos
    System.out.println("SUMA DE ENTEROS: "+calc.suma(100,200));
    //Llamamos al método suma de reales y lo imprimimos
    System.out.println("SUMA DE REALES: "+calc.suma(0.01,0.02));
  }
}
```

**Ejecución:**

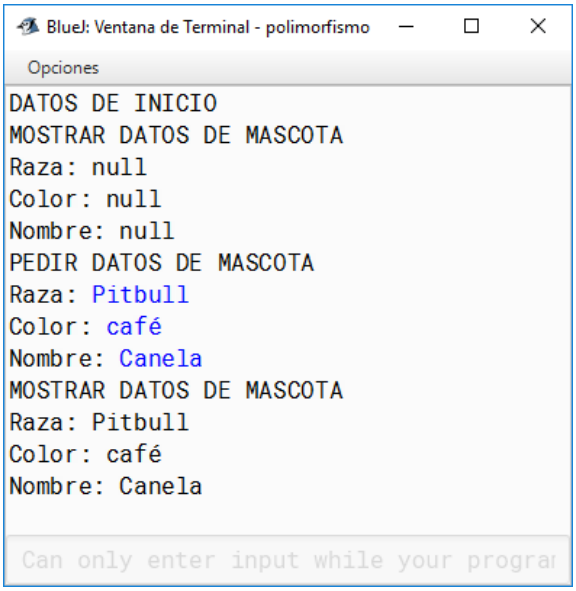



## Problema 2

```
//Importamos la biblioteca java.util para pedir datos por teclado
import java.util.Scanner;
//Declaramos la clase Mascota
public class Mascota
{
  //Declaramos los atributos de la clase
  private String raza;
  private String color;
  private String nombre;
  //Declaramos el constructor que no pide parámetros
  public Mascota( )
  {
    //no inicializamos valores
```

```
  }
  //Declaramos el constructor que pide parámetros
  public Mascota(String raza, String color, String nombre)
  {
    //Los argumentos los guardamos en las variables e instancia
    this.raza=raza;
    this.color=color;
    this.nombre=nombre;
  }
  //Declaramos un método para mostrar los datos del objeto
  public void mostrarDatos( )
  {
    System.out.println("MOSTRAR DATOS DE MASCOTA");
    System.out.println("Raza: "+raza);
    System.out.println("Color: "+color);
    System.out.println("Nombre: "+nombre);
  }
  //Método principal
  public static void main(String[ ] args)
  {
    //Instanciamos un objeto de la clase Scanner
    Scanner teclado=new Scanner(System.in);
    System.out.print("\u000C");
    System.out.println("DATOS DE INICIO");
    //Instanciamos un objeto sin valores de la clase Mascota
    Mascota masc1=new Mascota( );
    //Mostramos los valores de inicio con el método de instancia
    masc1.mostrarDatos( );
    System.out.println("PEDIR DATOS DE MASCOTA");
    //Pedimos los datos y los guardamos en variables
    System.out.print("Raza: ");
    String raza=teclado.nextLine( );
    System.out.print("Color: ");
    String color=teclado.nextLine( );
    System.out.print("Nombre: ");
    String nombre=teclado.nextLine( );
    //Instanciamos un segundo objeto con las variables guardadas
    Mascota masc2=new Mascota(raza,color,nombre);
    //Mostramos los valores de los atributos del objeto instanciado
    masc2.mostrarDatos( );
  }
}
```

El resultado de ejecutar el programa es el siguiente:




## Referencias

**Unidad 3**

Dean, J. (2009). Introducción *a la programación con Java*. Mc. Graw-Hill. México.

Aponte, I. (2009). *Constructores en Java*. [en línea] Disponible en: http://programandoenjava.over-blog.es/article-32829724.html [Consultado el 6 de febrero de 2019].

Ricardo, M. (2014*). Herencia en Java, con ejemplos*. [en línea] Disponible en: https://jarroba.com/herencia-en-la-programacion-orientada-a-objetos-ejemplo-en-java/ [Consultado el 1 de marzo de 2019].

Zárate, U. (s.f.). *Programación con Java.* [en línea] Disponible en: http://profesores.fi-b.unam.mx/carlos/java/ [Consultado el 1 de marzo de 2019].

# UNIDAD 4

## Interfaz gráfica de usuario


- Concepto de Interfaz Gráfica de usuario (GUI)
- La clase swing
- Componentes javax.swing
- La Clase AWT como antecedente de la clase Swing
- Relación de la clase Swing con la librería AWT
- Ambiente de desarrollo: NetBeans
- Clase Swing
- Clase Graphics

# Unidad 4 Interfaz gráfica de usuario

**Propósito:**

Al finalizar la unidad el alumno: Desarrollará programas en Java utilizando interfaces gráficas de usuario para aplicar y ampliar sus conocimientos de la programación orientada a objetos.

**Aprendizajes:**

El alumno:

- Conoce las características de la clase Swing.
- Elabora programas con interfaz gráfica de usuario aplicando las Clases: JFrame, JLabel y JButton.
- Propone un proyecto que utilice las clases JFrame, JLabel y JButton.
- Elabora programas con interfaz gráfica de usuario aplicando las Clases: JTextField, JTextArea y JComboBox
- Propone un proyecto que utilice las Clases: JTextField, JTextArea y JComboBox.
- Elabora programas con interface gráfica de usuario aplicando las Clases: JCheckBox, JRadioButton
- Elabora programas con interfaz gráfica de usuario aplicando las Clases: JMenuBar, JMenu, JMenuItem.
- Elabora programas con interface gráfica de usuario aplicando las Clases: JCheckBox, JRadioButton.
- Elabora programas con interfaz gráfica de usuario aplicando las Clases: setColor, drawLine, drawRect, drawRoundRect, drawOval, drawPolygon.
- Elabora programas con interfaz gráfica de usuario aplicando las Clases: fillRect, fillRoundRect, fillOval, fillPolygon.
- Desarrolla un proyecto que integre las Clases estudiadas en esta unidad.

## Introducción

La Interfaz grafica de usuario, también conocida como GUI (Graphical User Interface), permite al usuario comunicarse de forma sencilla y amigable con la computadora para ello utiliza conjuntos de formas gráficas. En esta unidad se elaborarán programas sencillos en Java utilizando interfaces gráficas de usuario y los conceptos abordados en las unidades anteriores.

El objetivo principal de una interfaz gráfica es simplificar y hacer mucho más cómoda la interacción entre una persona y un dispositivo. Utilizaremos objetos como Jpanel, Jframe, JButton, jlabel, JTextField, entre otros para realizar interfaces gráficas e instrumentar propiedades de la programación orientada a objetos.

## 4.1 Concepto de Interfaz Gráfica de usuario (GUI)

La Interfaz Gráfica de Usuario, también conocida como GUI, (*Graphic User Interface* por sus siglas en inglés), se compone de un conjunto de formas gráficas y métodos que permiten a los usuarios la interacción con un sistema, para lo cual se emplean un conjunto de formas gráficas e imágenes, estas se componen de elementos como botones, íconos, ventanas, fuentes, entre otros, estos representan funciones, acciones e información correspondiente al contexto de ese sistema.

Una interfaz gráfica debe cumplir con algunas de las características que se mencionan a continuación:

- ✓ Facilitar la comprensión, el uso y aprendizaje del objeto para el cual fue diseñado.
- ✓ El usuario puede identificar fácilmente el objeto de interés.
- ✓ Diseño ergonómico mediante el establecimiento de menús, barras de acciones e iconos de fácil acceso.
- ✓ Las interacciones del usuario con el sistema serán con base a acciones físicas sobre elementos de código visual o auditivo (iconos, botones, imágenes, mensajes de texto o sonoros, barras de desplazamiento y navegación) y en selecciones de tipo menú con sintaxis y órdenes.
- ✓ Las operaciones serán rápidas, incrementales y reversibles, con efectos inmediatos.
- ✓ Se deben incluir herramientas de ayuda y consulta.
- ✓ Tratamiento del error, por parte del programador, bien cuidado y adecuado al nivel de usuario.

La interfaz de navegación no debe limitarse a la parte visible de la información, al contrario, debe ser capaz de ofrecer al usuario el acceso a la parte del documento que le interesa y en la forma que desea. Aun tratándose de un entorno navegacional complejo, éste se debe presentar al usuario de una forma sumamente sencilla y que sea lo más normalizada posible.

## 4.2 La clase Swing

Es un conjunto completo de componentes gráficos, escrito en Java. Que ofrece un gran número de herramientas al programador.
Ofrece numerosas ventajas, por ejemplo, la navegación con el teclado es automática, cualquier aplicación Swing se puede utilizar sin ratón, sin tener que escribir ni una línea de código adicional.

## 4.3 Componentes javax.swing

Los componentes gráficos son los que permiten brindar una interacción con el usuario del sistema, cada componente corresponde a una clase en Java, actualmente para desarrollar interfaces gráficas se utilizan los componentes de la clase Swing.

Dentro de los componentes más importantes de la clase Swing tenemos los siguientes:

**a) Contenedores**

Estos elementos permiten agrupar y contener a los elementos gráficos, entre los más importantes tenemos:

| Componente | Función |
|---|---|
| **JFrame** | Es el contenedor principal, es decir la ventana de aplicación. |
| **JDialog** | Es una ventana secundaria en donde se muestra una iteración con el usuario. |
| **JPanel** | Permite crear paneles independientes dentro de la ventana de aplicación para almacenar componentes distintos. |
| **JScrollPane** | Con esta opción, una barra de desplazamiento puede ser incluida en un contenedor. |
| **JSplitPane** | Nos da la opción de que un contenedor pueda ser dividido en dos secciones. |
| **JTabbedPane** | Sirve para la creación de pestañas y que en cada pestaña se tenga asociado un contenedor diferente. |
| **JDesktopPane** | Se utiliza para crear ventanas secundarias dentro de la ventana principal. |
| **JToolBar** | Permite que se pueda utilizar una barra de herramientas. |

***Tabla 4.3.1. Contenedores de la clase Swing.***

**b) Componentes atómicos**

Son aquellos elementos que no pueden almacenar a otros elementos gráficos, dentro de los más importantes tenemos los siguientes:

| Componente | Función |
|---|---|
| **JLabel** | Permite la creación de etiquetas que pueden contener tanto texto e imágenes. |
| **JButton** | Sirve para la inclusión de botones simples en una ventana. |
| **JCheckBox** | Se utilizan como casillas de verificación para una selección múltiple. |
| **JRadioButton** | Es similar a la opción anterior sin embargo solo es posible seleccionar un único valor. |

| Componente | Función |
|---|---|
| **JToggleButton** | Es un tipo de botón que se queda presionado hasta que se ejecuta otro evento. |
| **JComboBox** | Muestra una lista de elementos como un combo de selección. |
| **JScrollBar** | Nos permite agregar una barra de desplazamiento en áreas de texto o en paneles. |
| **JSeparator** | Es una barra simple para separar opciones. |

***Tabla 4.3.2. Componentes atómicos de la clase Swing***

**c) Componentes de texto.**

Son aquellos que nos permiten presentar texto tanto para la entrada como para la salida de información:

| Componente | Función |
|---|---|
| **JTextField** | Permite la introducción de texto simple. |
| **JFormattedTextField** | Se utiliza para introducir texto con formato. |
| **JPasswordField** | Lleva a cabo la introducción de texto sin mostrar los caracteres que se van ingresando. |
| **JTextArea** | Vincula un área de texto en la ventana donde se ingresa o se muestra información. |
| **JEditorPane** | Permite crear en la ventana un área para editar texto. |
| **JTextPane** | Es similar al anterior, pero con más opciones como iconos, color, etcétera. |

***Tabla 4.3.3. Componentes de texto de la clase Swing.***

**d) Componentes de menú.**

Permiten la creación y vinculación de opciones de menú en una ventana de aplicación, entre los principales tenemos:

| Componente | Función |
|---|---|
| **JMenuBar** | Se utiliza para incluir una barra de menú en la ventana. |
| **JMenu** | Permite que se vinculen botones o enlaces los cuales se despliegan en el menú principal. |

| Componente | Función |
|---|---|
| **JMenuItem** | Elemento que se encuentra en la barra de menú. |
| **JCheckBoxMenuItem** | En un elemento similar a *JCheckBox* pero que se incrusta en el menú. |
| **JRadioButtonMenuItem** | Similar al elemento anterior pero que solo permite seleccionar un único elemento dentro del menú. |
| **JPopupMenu** | Permite la generación de menús emergentes. |

***Tabla 4.3.4. Componentes de menú de la clase Swing.***

## 4.4 La Clase AWT como antecedente de la clase Swing

En los años 90's cuando se buscaba mejorar la portabilidad entre aplicaciones desarrolladas para diferentes sistemas operativos surge Java con el concepto de programar una vez y ejecutarse en cualquier plataforma.

Inicialmente los programas se ejecutaban en modo consola, es decir solo texto, pero posteriormente al desarrollarse sistemas operativos que funcionan bajo el concepto gráfico de ventana es necesario replantear el reto, ya que una aplicación que fue desarrollada inicialmente para Windows tendría que volver a reescribirse si esta se quiere ejecutar en MacOS o en GNU/Linux.

En Java este problema se aborda mediante el uso de la clase AWT (Abstract Window Toolkit) la cual se encarga de gestionar la interfaz gráfica con independencia del sistema operativo en donde se ejecute el programa.

La clase AWT además de proporcionar un conjunto en común de herramientas para la interfaz gráfica de usuario, también proporciona elementos para manipular imágenes y generar gráficos. Los sistemas operativos modernos se basan en la idea de "componentes" los cuales son elementos reutilizables que se implementan en la interfaz.

Windows utiliza elementos como ventanas, botones, cuadros de texto, barras de desplazamientos, casillas de verificación entre otros. La clase AWT contiene un repertorio de componentes básicos de interfaz de usuario, similares a los de Windows, además de la posibilidad de crear componentes propios, es decir la combinación de los componentes básicos.

## 4.5 Relación de la clase Swing con la librería AWT: java.awt.* y java.awt.event.*

Existen dos conjuntos de componentes GUI en Java: AWT y Swing. En la siguiente tabla se muestran algunas de sus características.

| Características | AWT | Swing |
|---|---|---|
| Paquete | java.awt | javax.swing |
| Componentes | Pesados: porque están enlazados a la plataforma local. | Ligeros, porque no están enlazados a la plataforma local. |
| Nombre de los componentes | Button | JButton |
| Apariencia visual | La misma apariencia de las demás aplicaciones de la plataforma. | Apariencia uniforme para una aplicación a través de todas las plataformas. |
| Portabilidad | Presenta problemas porque es dependiente de la plataforma. | Es flexible, porque es independiente de la plataforma. |

***Tabla 4.5.1. Características de la librería AWT y la clase Swing.***

Antes de introducir los componentes Swing en Java SE 1.2, las GUIs de Java se creaban a partir de componentes del AWT en el paquete java.awt. La clase Swing implementa un conjunto de componentes construidos sobre AWT y además proporciona mejoras en la visualización de los diferentes elementos. En las siguientes tablas se muestran algunas clases de AWT y Swing.

| Clases del paquete java.awt | Descripción |
|---|---|
| java.awt.Component | Un componente es un objeto que tiene una representación gráfica que se puede mostrar en la pantalla y que puede interactuar con el usuario. |
| java.awt.Container | Un contenedor es un componente que puede contener otros componentes de AWT. |
| java.awt.Window | Un objeto de Window es una ventana de nivel superior sin bordes y sin barra de menú. |
| java.awt.Frame | Un marco es una ventana de nivel superior con un título y un borde. |

***Tabla 4.5.2. Clases de AWT.***

| Clases de javax.swing | Descripción |
|---|---|
| javax.swing.JComponent | JComponent es la clase base para todos los componentes Swing. |
| javax.swing.JPanel | Es un contenedor genérico ligero. |
| javax.swing.JFrame | Una versión extendida de java.awt.Frame que agrega soporte para la arquitectura de componentes. |
| javax.swing.JPopupMenu | Una implementación de un menú emergente: una pequeña ventana que aparece y muestra una serie de opciones. |
| javax.swing.JToolBar | Proporciona un componente que es útil para mostrar controles de uso común o barras de herramientas. |

***Tabla 4.5.3. Clases de Swing.***

Para crear una GUI es necesario al menos un objeto contenedor y varios componentes. Importando los paquetes javax.swing.* y java.awt.* incluimos todas las clases preconstruidas de Java que necesitamos.

Los componentes de GUI pueden generar una variedad de eventos en respuesta a las interacciones del usuario. Un evento es un mensaje que indica al programa que algo ha ocurrido. En la siguiente tabla se muestran algunos eventos.

| Acción del usuario | Qué ocurre |
|---|---|
| Oprimir la tecla Enter mientras el cursor está dentro de una ventana de diálogo. | El objeto de la ventana de diálogo activa un evento e indica al programa que la tecla Enter fue oprimida dentro de la ventana de diálogo. |
| Hacer clic en el botón de un menú. | El objeto en el menú activa un evento, e indica al programa que se seleccionó ese elemento del menú. |
| Cerrar una ventana (hacer clic en el botón “X” en el ángulo superior derecho). | El objeto en la ventana activa un evento, e indica al programa que hizo clic en el botón para cerrar la ventana. |

***Tabla 4.5.4. Ejemplos de eventos.***

El paquete java.awt.event.* contiene todos los eventos que se pueden programar en Java y que se van a explicar en su momento.

Actividad 4. 1

Instrucciones: Relaciona ambas columnas con las respuestas correspondientes:

| | |
|---|---|
| 1) GUI | (  ) Características de una Interfaz Gráfica de Usuario |
| 2)<br>- El usuario puede identificar fácilmente un objeto de interés.<br>- Se deben incluir herramientas de ayuda y consulta. | (  ) Atómicos |
| 3) La clase Swing | (  ) Se conoce como Interfaz Gráfica de Usuario por sus siglas en inglés. |
| 4) En la clase Swing los contenedores: | (  ) Es un conjunto completo de Componentes gráficos |
| 5) Son los componentes que no pueden almacenar a otros elementos tales como JLabel, JButton etcétera. | (  ) Se utiliza para tener múltiples casillas de verificación, pero solo es posible seleccionar un único valor. |
| 6) JRadioButton | (  ) La apariencia visual es uniforme ya que no depende del sistema operativo por lo cual los componentes son ligeros. |
| 7) Permite crear en la ventana un área para editar texto. | (  ) JEditorPane |
| 8) Fue la primera clase en Java que permitió gestionar una interfaz gráfica independientemente del sistema operativo. | (  ) Son elementos que permiten la agrupación de elementos gráficos tales como Jframe, JPanel, JDialog entre otros. |
| 9) En AWT | (  ) AWT (Abstract Window Toolkit) |
| 10) En Swing | (  ) La apariencia visual depende del sistema operativo por lo cual los componentes son pesados. |

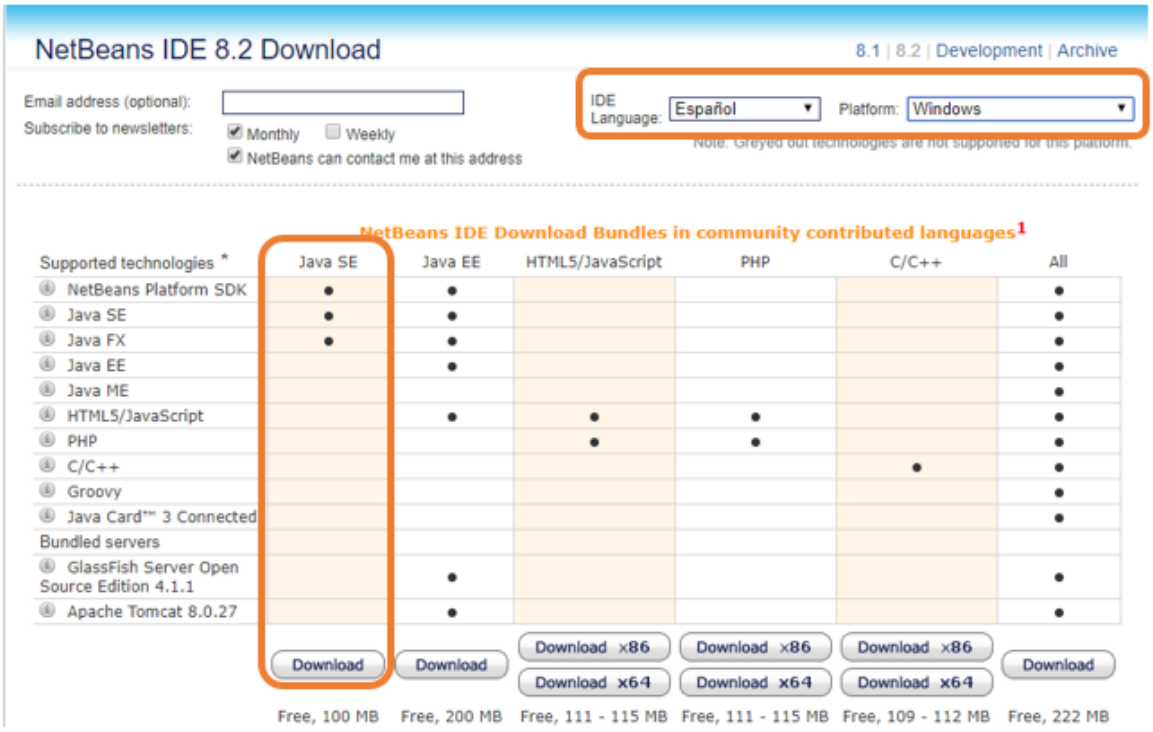
## 4.6 Ambiente de desarrollo: NetBeans

NetBeans IDE es un entorno de desarrollo, esto es, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas escritos en Java, aunque también puede servir para otros lenguajes de programación. Existe también un número importante de módulos para extender el NetBeans IDE. Además, es un producto libre y gratuito sin restricciones de uso.

NetBeans IDE simplifica el desarrollo de aplicaciones para Java Swing, la cual es una biblioteca gráfica para Java, que incluye widgets (pequeñas aplicaciones), para interfaz gráfica de usuario tales como caja de texto, botones, listas desplegables, entre otros.

Los pasos para instalar el software son los siguientes:

1.Entrar al sitio https://netbeans.org/downloads/8.2/ y elegir el idioma, la plataforma y la opción Java SE.



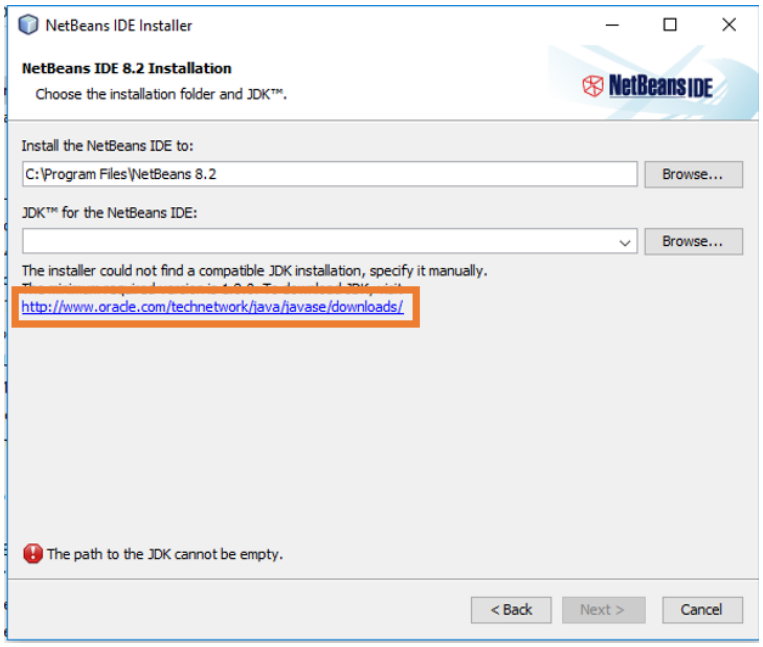

2. Se comenzará a descargar el archivo.

3. Terminada la descarga, abrir el archivo y dar clic en el botón Sí del cuadro de diálogo.

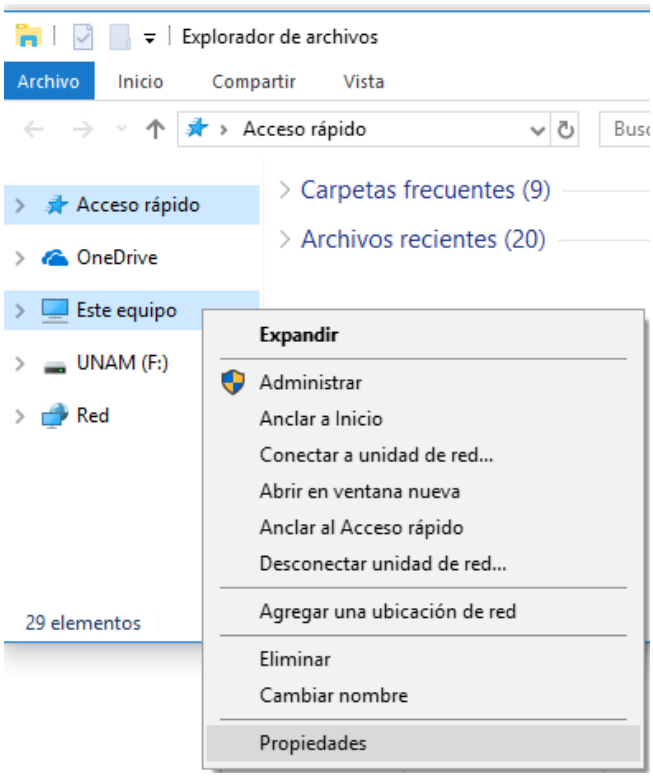
4. Dar clic en el botón Next >.

5. Aceptar los términos de la licencia y dar clic en Next >.

6. Entrar a la página http://www.oracle.com/technetwork/java/javase/downloads/ que indica el siguiente cuadro de diálogo para descargar el JDK (Java Development Kit, es el software que provee herramientas de desarrollo para la creación de programas en Java).




7. Para conocer la versión del sistema operativo, abrimos una ventana de Explorador de archivos y damos clic con el botón derecho del ratón sobre Este equipo, elegir la opción Propiedades.



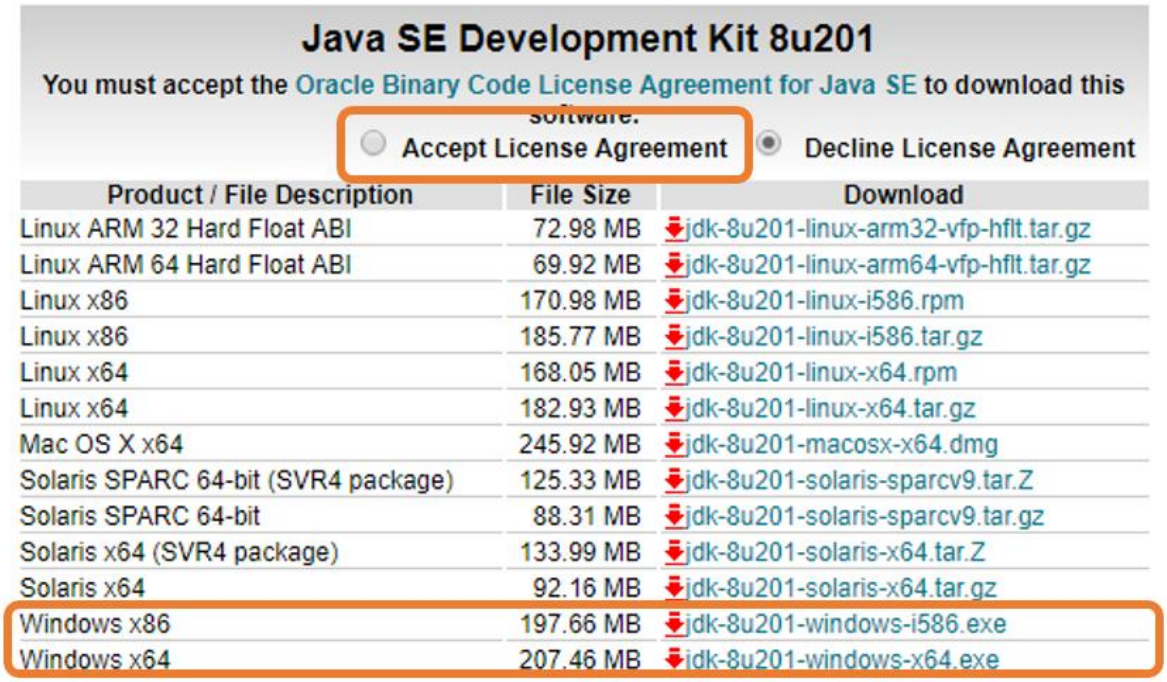

Se abrirá una ventana en donde indica el tipo de procesador.

Sistema

| | |
|---|---|
| Procesador: | Intel(R) Core(TM) i5-4570 CPU @ 3.20GHz 3.20 GHz |
| Memoria instalada (RAM): | 8.00 GB (7.89 GB utilizable) |
| Tipo de sistema: | Sistema operativo de 64 bits, procesador x64 |
| Lápiz y entrada táctil: | La entrada táctil o manuscrita no está disponible para esta pantalla |

8. Dentro de la página para descargar el JDK elegir la versión Java SE 8u201 tanto para un sistema de 32 bits como para 64 bits.

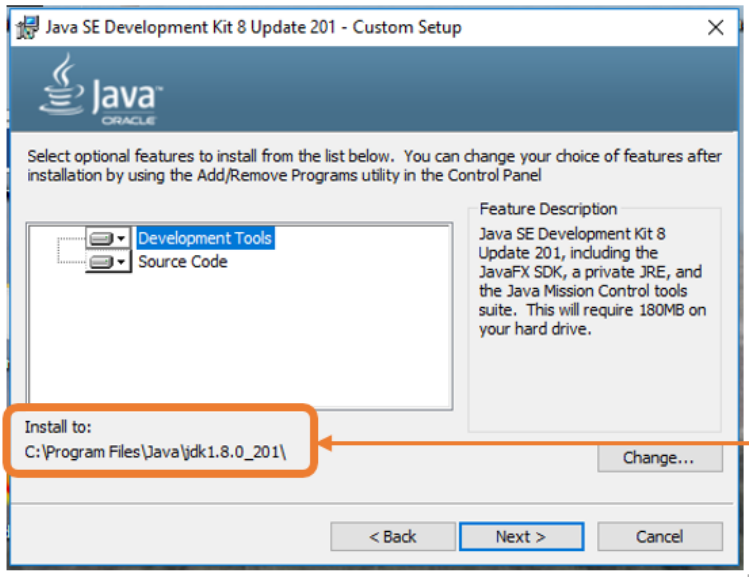

9. En la página que abre seleccionar Accept License Agreement y elige el archivo para el tipo de sistema que tiene la computadora: en caso de un sistema de 32 bits elige el archivo Windows x86 y para 64 bits selecciona Windows x64.




| Product / File Description | File Size | Download |
|---|---|---|
| Linux ARM 32 Hard Float ABI | 72.98 MB | jdk-8u201-linux-arm32-vfp-hflt.tar.gz |
| Linux ARM 64 Hard Float ABI | 69.92 MB | jdk-8u201-linux-arm64-vfp-hflt.tar.gz |
| Linux x86 | 170.98 MB | jdk-8u201-linux-i586.rpm |
| Linux x86 | 185.77 MB | jdk-8u201-linux-i586.tar.gz |
| Linux x64 | 168.05 MB | jdk-8u201-linux-x64.rpm |
| Linux x64 | 182.93 MB | jdk-8u201-linux-x64.tar.gz |
| Mac OS X x64 | 245.92 MB | jdk-8u201-macosx-x64.dmg |
| Solaris SPARC 64-bit (SVR4 package) | 125.33 MB | jdk-8u201-solaris-sparcv9.tar.Z |
| Solaris SPARC 64-bit | 88.31 MB | jdk-8u201-solaris-sparcv9.tar.gz |
| Solaris x64 (SVR4 package) | 133.99 MB | jdk-8u201-solaris-x64.tar.Z |
| Solaris x64 | 92.16 MB | jdk-8u201-solaris-x64.tar.gz |
| Windows x86 | 197.66 MB | jdk-8u201-windows-i586.exe |
| Windows x64 | 207.46 MB | jdk-8u201-windows-x64.exe |

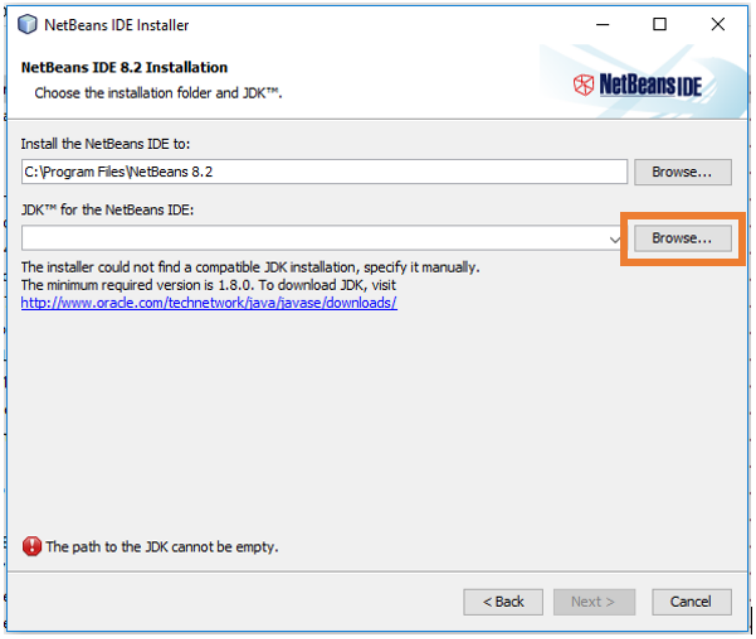
10. Después de que se descargue el archivo abrimos y damos clic en el botón sí del cuadro de diálogo.

11. Abrirá el programa instalador y para continuar das clic en Next >. En seguida observamos la carpeta donde guardará el programa y dar clic en Next >.

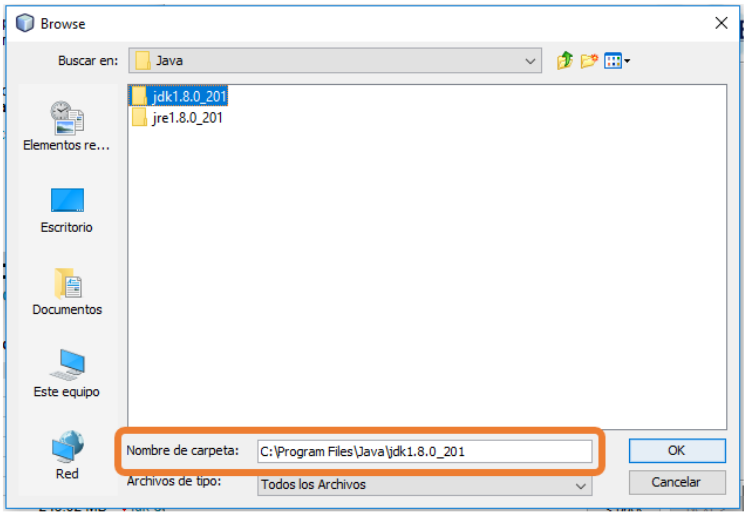



12. El software se instalará y cuando termine hacemos clic en Close.

13. En la instalación de NetBeans da clic en el botón Browse…




Buscar la carpeta en donde se instaló el JDK y da clic en el botón OK.




14. Da clic en Next > para continuar y clic en Install para instalar.

15. El programa de NetBeans tardará unos minutos en instalarse y cuando termine oprime el botón Finish.

## 4.7 Clase Swing

Ya hemos descrito lo que es la clase Swing y los programas que hemos realizado se han ejecutado en modo consola, ahora vamos a utilizar la interfaz gráfica y a explicar sus componentes.

- **JFrame**

Hoy en día, casi todo el software está basado en ventanas que contienen una barra de título, límites, un botón para minimizar, un botón para cerrar la ventana, la función de modificar el tamaño de la ventana, etcétera. Estas características pueden implementarse con la clase JFrame.

Para crear un JFrame con NetBeans se deben seguir los siguientes pasos:

1. Abrir el programa NetBeans y crear un nuevo proyecto desde el menú Archivo y la opción Proyecto Nuevo.

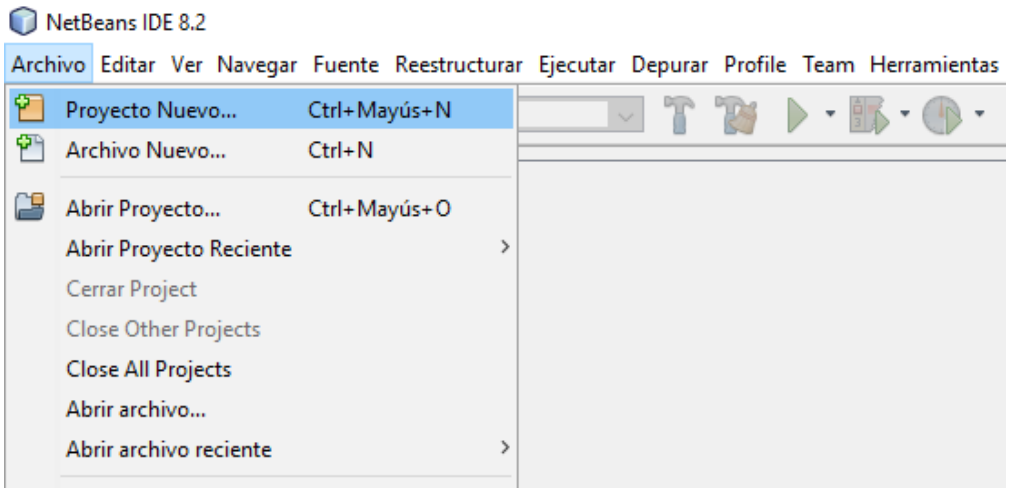



***Figura 4.7.1. Crear un proyecto nuevo.***

2. En la ventana que se muestra debes elegir Categorías Java y Proyectos Java Application. Oprime Siguiente.

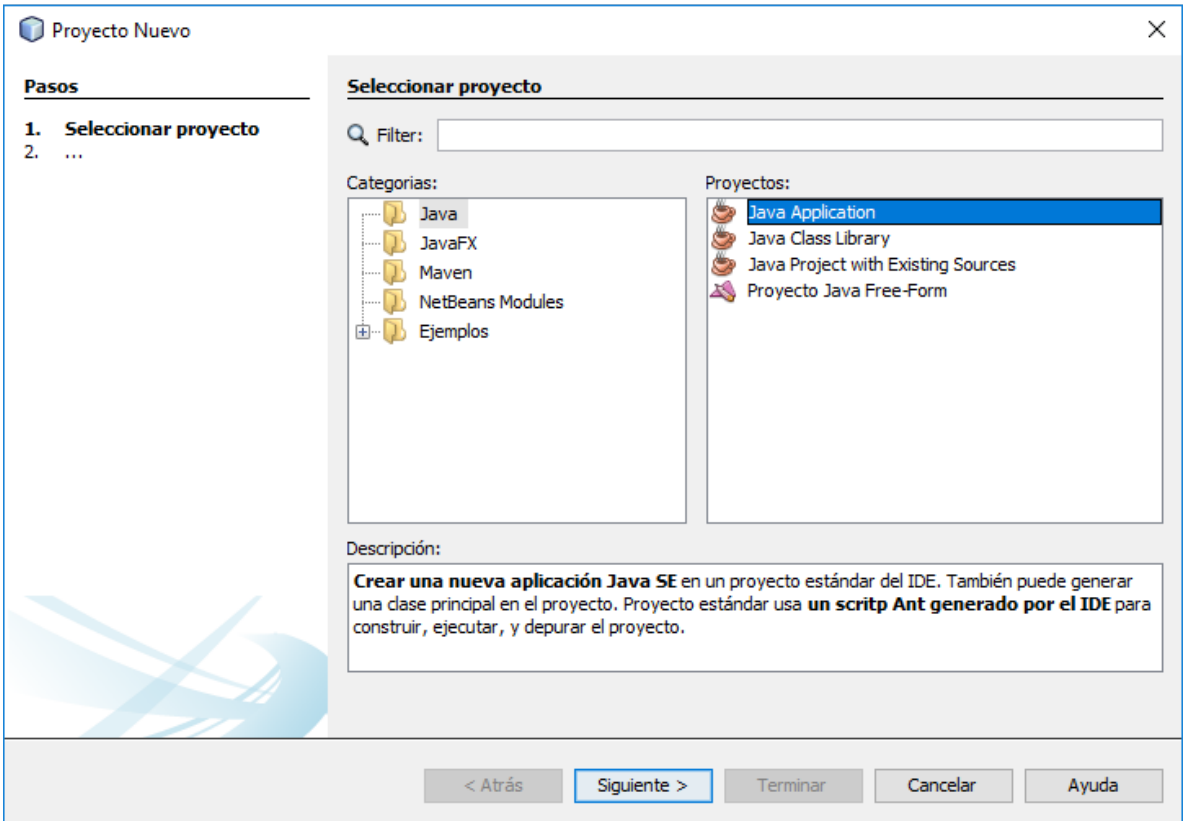



***Figura 4.7.2. Elegir una aplicación Java.***

3. En seguida debes escribir un nombre al proyecto y elegir la ubicación donde se va a guardar y deseleccionar la opción Crear clase principal.

4. Después de esto, aparecerá el programa abierto y del lado izquierdo el proyecto como lo nombraste. Despliega las carpetas que lo conforman dando clic en el símbolo [+]. Da clic derecho en la carpeta Paquete de fuentes, selecciona Nuevo y la opción Java Package y escribe un nombre al paquete.

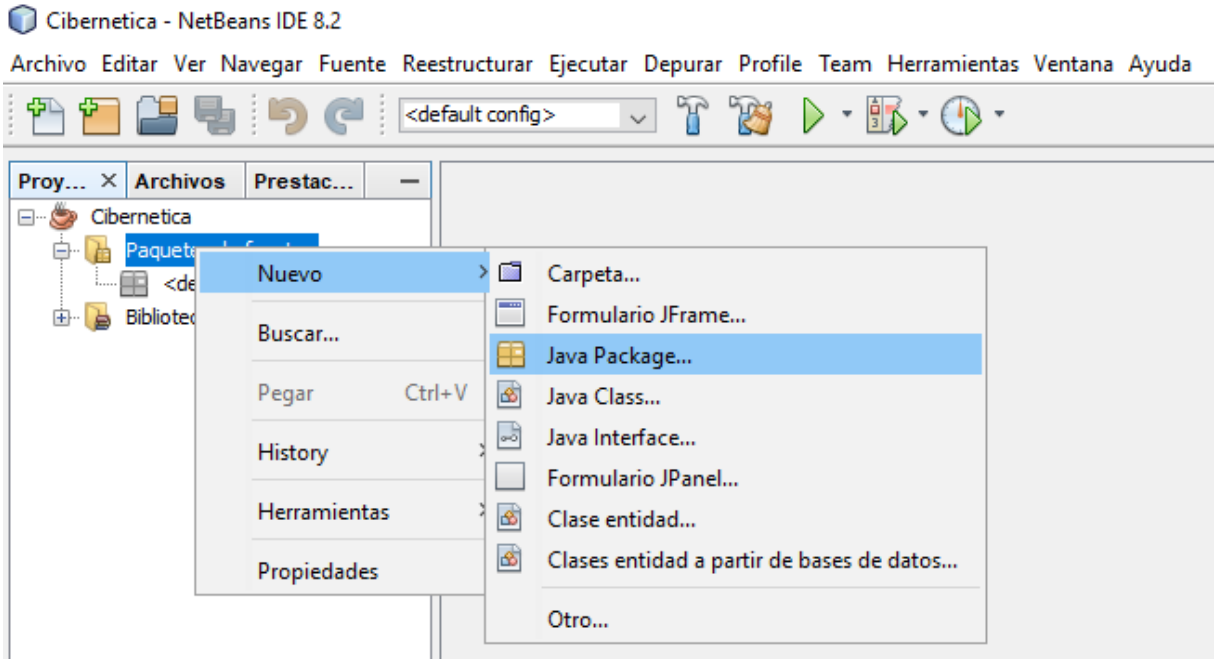



***Figura 4.7.3. Crear un nuevo paquete.***

5. Ahora da clic con el botón derecho del ratón sobre el paquete creado, elige Nuevo y la opción Formulario JFrame y escribe un nombre a la clase que se va a crear.

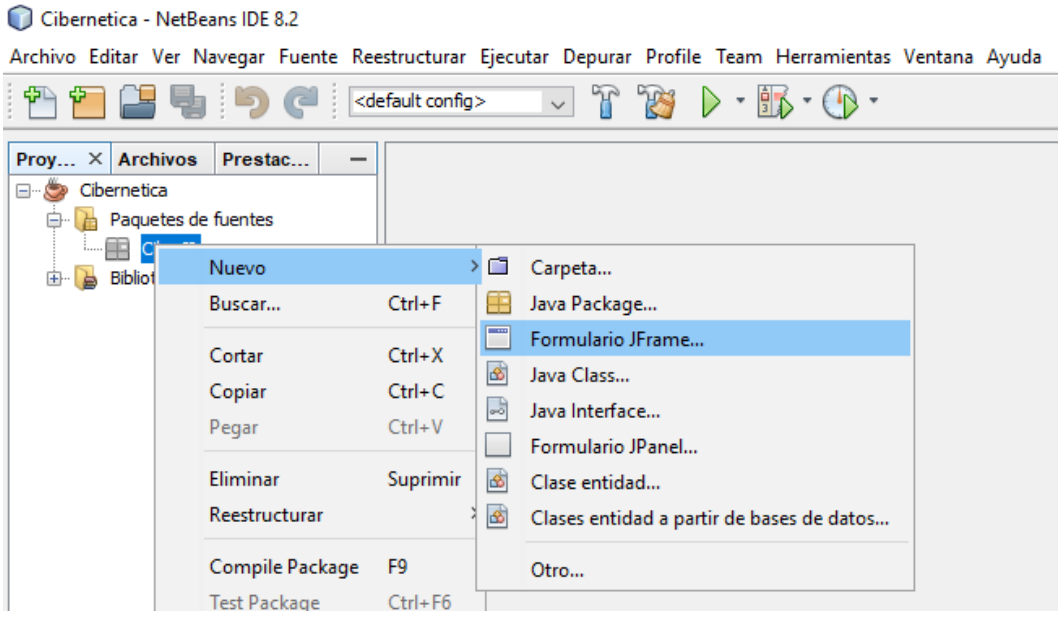



***Figura 4.7.4. Crear un JFrame.***

De esta manera habrás creado un JFrame. A continuación, se muestran y explican algunos elementos del entorno de trabajo de NetBeans.

**Barra de título:** presenta el nombre del proyecto y el nombre del programa.
**Barra de menús:** contiene los diferentes menús a los que se puede tener acceso.
**Barra de herramientas:** está área tiene botones para crear un nuevo archivo, crear un nuevo proyecto, abrir proyecto, guardar todo, deshacer, rehacer y ejecutar proyecto, entre otras.

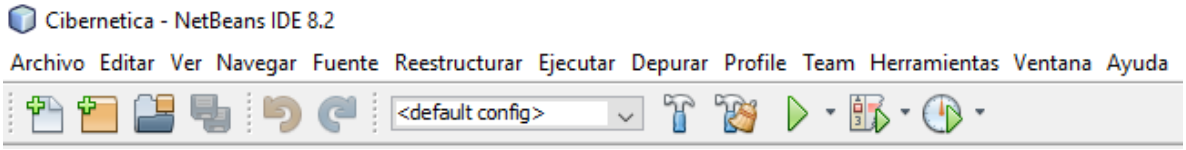



*Figura 4.7.5. Barra de título, menús y herramientas.*

### Menú Archivo

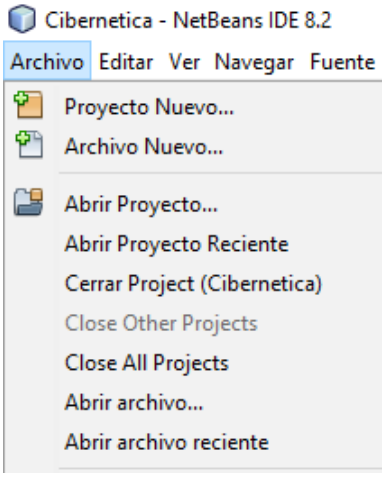



*Figura 4.7.6. Menú archivo*

**Proyecto Nuevo.** Al seleccionar este comando se crea un nuevo proyecto al cual se le asigna un nombre.

**Archivo Nuevo.** En este comando se puede crear un archivo de diferente tipo como: clase, interface, clase main, etcétera.

**Abrir Proyecto.** Sirve para seleccionar la carpeta donde se encuentra el proyecto que se quiere abrir.

**Abrir Proyecto Reciente.** Muestra una lista de los proyectos usados recientemente.

**Cerrar Proyecto.** Cierra el proyecto en uso.

### Menú Edición

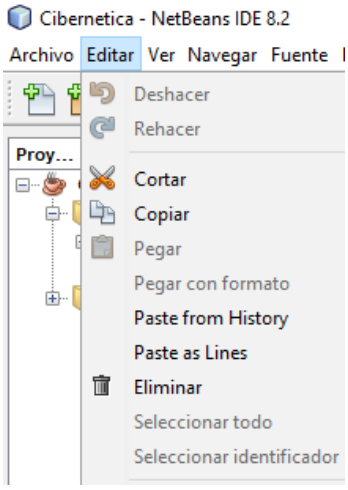



*Figura 4.7.7. Menú edición.*

**Deshacer.** Permite borrar la última acción hecha.

**Rehacer.** Permite regresar la última acción borrada.

**Cortar.** Permite cortar código o archivos.

**Copiar.** Copia al portapapeles el código o archivo seleccionado.

**Pegar.** Pega lo cortado o copiado al portapapeles.

**Eliminar.** Borra por completo lo seleccionado.

### Menú Ver

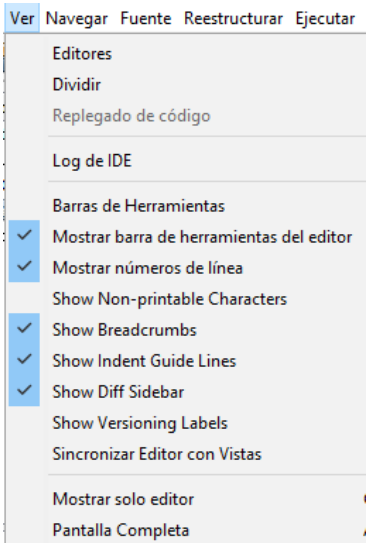



*Figura 4.7.8. Menú ver.*

**Editores.** Sirve para seleccionar entre la vista código (Source) y diseño (Design).

**Barra de Herramientas.** Se utiliza para indicar las herramientas que deseamos tener en la barra visible de herramientas.

**Mostrar barra de herramientas del editor.** Permite elegir ver o no ver la barra en el área de trabajo.

**Mostrar números de línea.** Para quitar o mostrar los números en la línea de código.

Cabe aclarar que únicamente se mencionaron los comandos más usados de algunos menús.

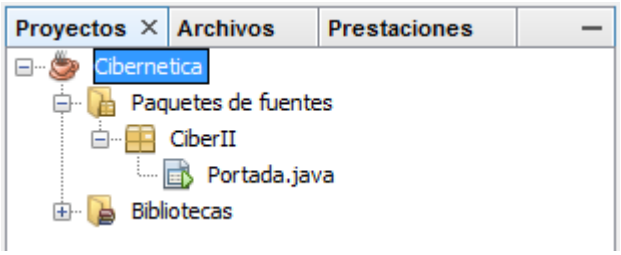



*Figura 4.7.9. Lista de proyectos.*

**Lista de proyectos:** muestra los paquetes y clases que contiene el o los proyectos.

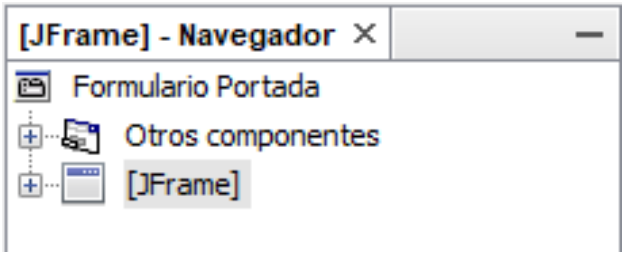



*Figura 4.7.10. Navegador de componentes.*

**Navegador de componentes:** enlista todos los componentes que contiene el JFrame.

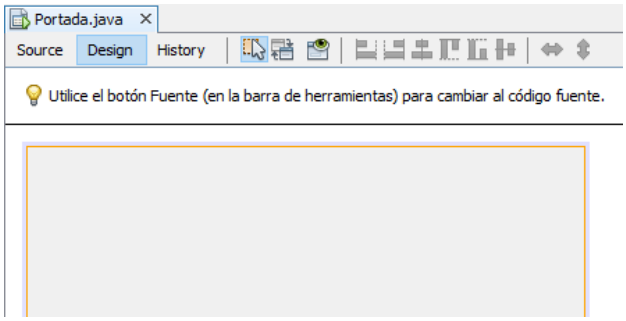



*Figura 4.7.11. Ventana del diseñador visual.*

**Ventana del diseñador visual:** está formada por la pestaña de archivos abiertos, los botones para intercambiar entre la vista código (Source) y vista diseño (Design) así como modo selección, modo colección, vista previa, tipos de alineaciones y el área del JFrame.

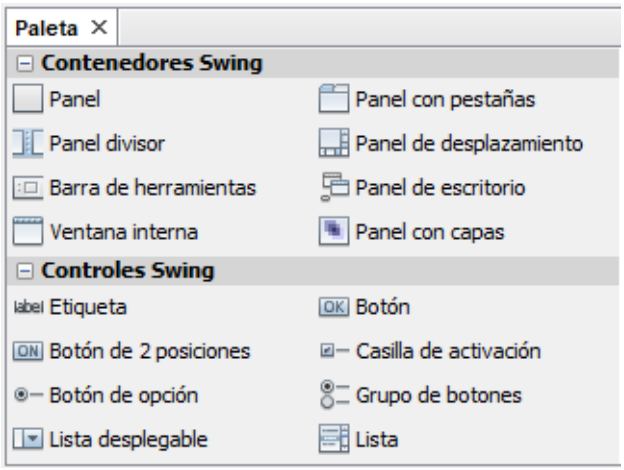



*Figura 4.7.12. Paleta de componentes.*

**Paleta de componentes:** contiene todos los elementos que se pueden agregar al JFrame, como: Panel, Etiqueta, Botón, Casilla de activación, Botón de opción, Grupo de botones, Lista desplegable, Barra de Menú, Menú, Elemento de Menú, etcétera.
Para ver más elementos damos clic en el botón [-] de cada apartado o en [+] para desplegar la lista.

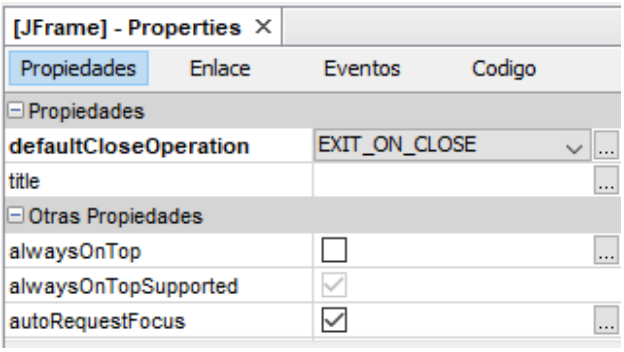



*Figura 4.7.13. Propiedades de los elementos.*

**Propiedades de los elementos:** dependiendo del componente seleccionado muestra las características que pueden ser modificadas de ese elemento.
En la pestaña de Código se puede cambiar el nombre la variable del componente.

**Panel de salida:** después de ejecutar el proyecto, este panel muestra los mensajes de error o de éxito de la compilación del programa.

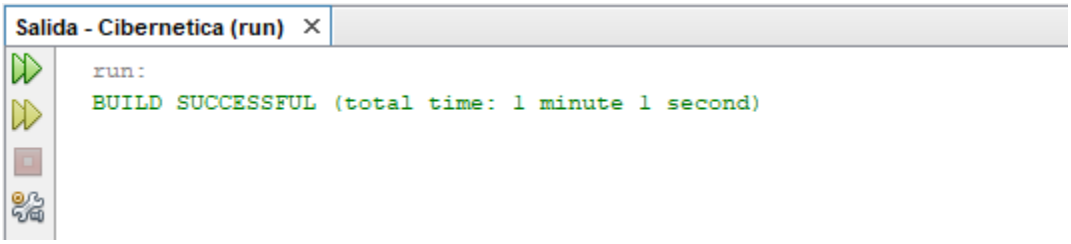



***Figura 4.7.14. Panel de salida.***

Ahora vamos a modificar algunas propiedades importantes del contenedor JFrame.

1. En la propiedad default Close Operation debemos seleccionar la opción EXIT_ON_CLOSE para que la ventana se cierre y se termine el programa.

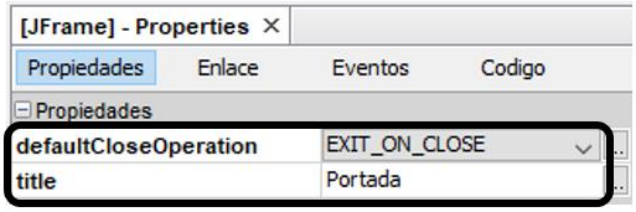



***Figura 4.7.15. Propiedades de cerrar y título de JFrame.***

2. En la opción title se escribe el título que va a tener la ventana actual.

3. Buscamos la opción preferredSize para cambiar el tamaño de la ventana indicando la anchura y la altura.

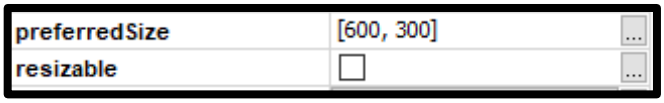



***Figura 4.7.16. Propiedades de tamaño y redimensión de JFrame.***

4. Desactivamos la casilla de la propiedad resizable para que el usuario no pueda modificar las dimensiones de la ventana.

5. Para cambiar el color de fondo del JFrame vamos a entrar a la pestaña Source para escribir la siguiente sentencia:

*this.getContentPane( ).setBackground(Color.green);*

Con esta sentencia hacemos referencia al método para obtener el contenedor y modificar su atributo de fondo con un atributo de la clase Color que puede tomar los valores black, blue, cyan, dark_gray, gray, green, ligthGray, magenta, orange, pink, red, white y yellow. Para hacer uso de la clase Color debemos importar la librería java.awt.Color. En la siguiente imagen se muestra cómo debe quedar el código.

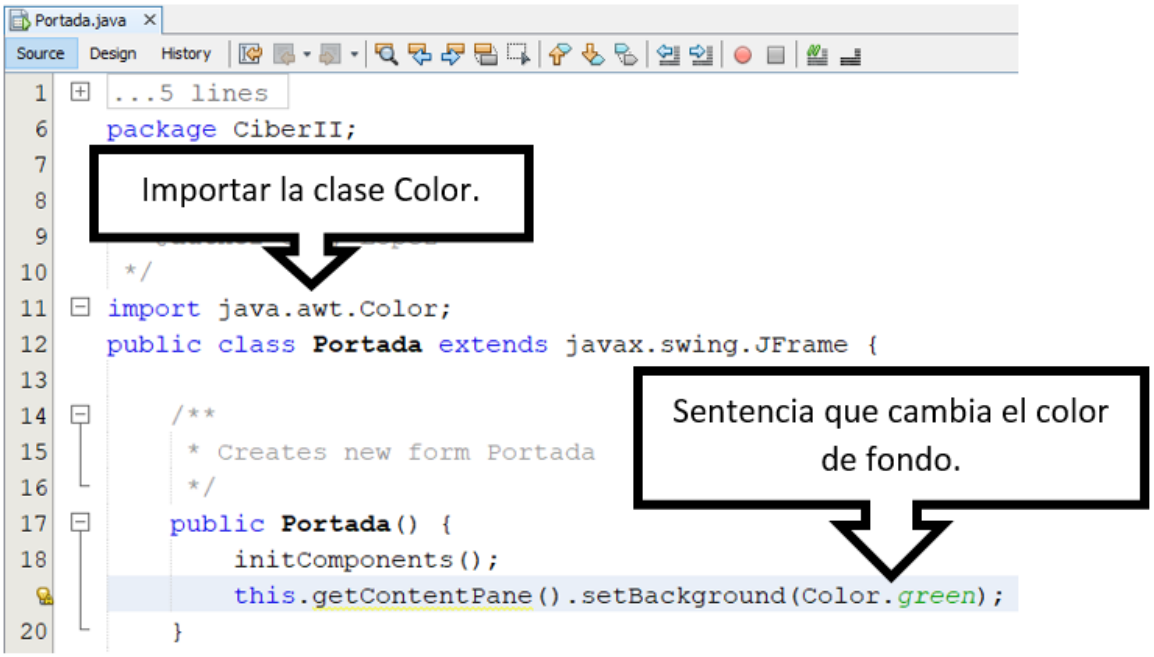



***Figura 4.7.17. Sentencias para cambiar el color de fondo del JFrame.***

Después ejecutamos el programa pulsando la tecla F6, o dando clic en el botón del triángulo verde en la barra de herramientas, o dando clic con el botón derecho del ratón sobre el archivo y seleccionando Ejecutar Archivo como se muestra en la siguiente figura.

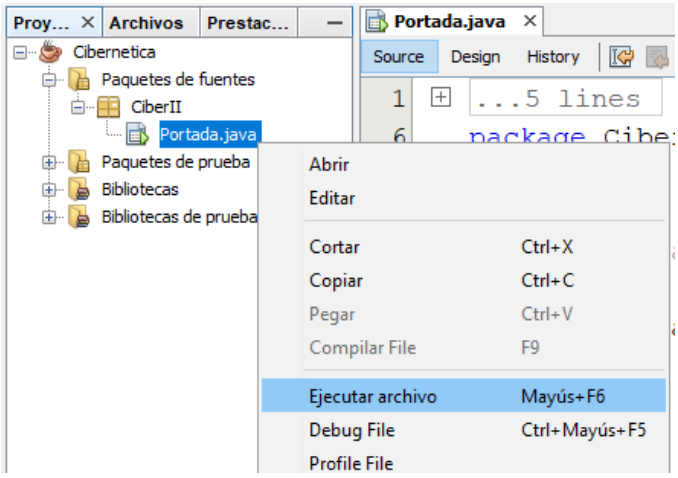



***Figura 4.7.18. Ejecutar un JFrame.***

El resultado del JFrame modificado es una ventana con título Portada, de tamaño 600 X 300, que no puede modificarse sus dimensiones y color de fondo verde, el cual se muestra a continuación.

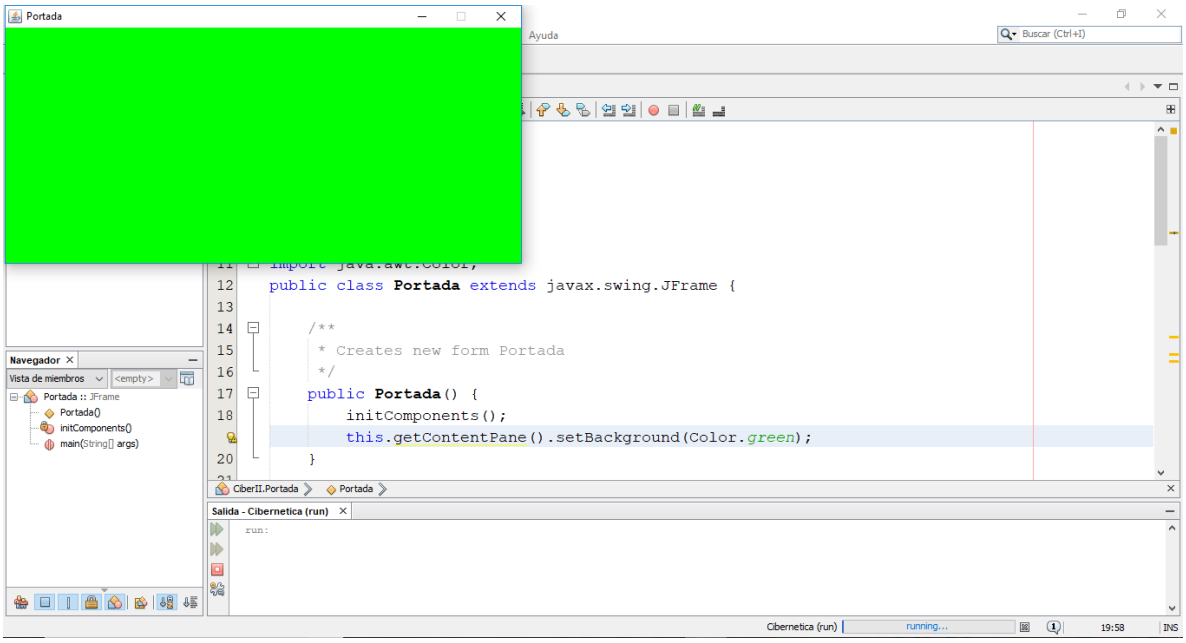



***Figura 4.7.19. Ejecución de la clase Portada.***

- **JLabel - Etiqueta**


El componente JLabel muestra una sola línea de texto que se ha especificado. Se considera un componente de sólo lectura porque el usuario puede leerlo, pero no puede interactuar con él. Normalmente, una etiqueta muestra una sola línea de texto, no varias.

Para insertar una etiqueta y modificar sus propiedades hacemos lo siguiente:

1. Seleccionamos el elemento de los controles Swing y lo colocamos en el lugar que deseemos.

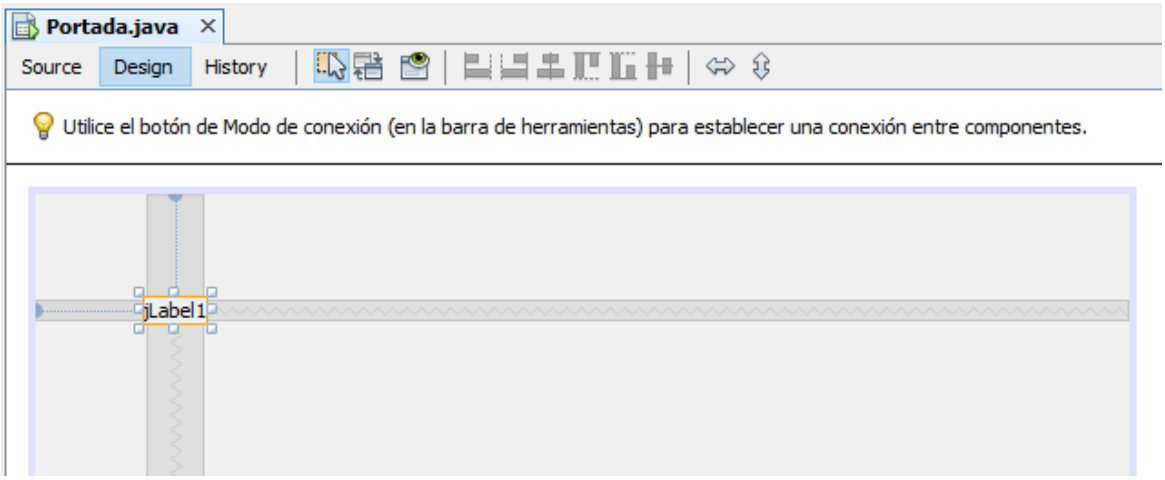



*Figura 4.7.20. Insertar una etiqueta.*

2. Dando clic con el botón derecho del ratón sobre la etiqueta insertada y seleccionando Propiedades podemos ver y editar sus características, o también en la sección de Propiedades de la pantalla de trabajo. Algunas características que podemos cambiar de este componente son:

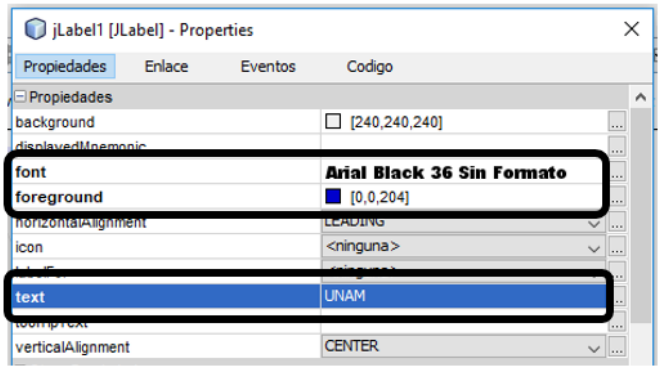



*Figura 4.7.21. Propiedades de letra, color y texto de JLabel.*

**Font**: dando clic sobre el botón que tiene los tres puntos […] podemos elegir el tipo, estilo y tamaño de la letra que se va a mostrar.

**Foreground**: podemos cambiar el color de la letra.

**Text**: para cambiar el texto que aparece en la etiqueta.

Si agregamos dos etiquetas más y editamos sus propiedades tenemos como resultado la siguiente imagen.


*Figura 4.7.22. Ejemplo de etiquetas.*

- **JButton – Botón**


Un botón es un componente en el que el usuario hace clic para desencadenar cierta acción. Un botón de comando genera un evento ActionEvent cuando el usuario hace clic en él.

Para insertar un botón en el JFrame seguimos los siguientes pasos:

1. Seleccionamos el elemento de los controles Swing y lo colocamos en el lugar que deseemos.


***Figura 4.7.23. Insertar un botón.***

2. Cambiamos sus propiedades de texto, letra y color de la letra desde las propiedades (igual que una etiqueta), y el nombre de la variable desde la pestaña código, como se muestra en la figura.

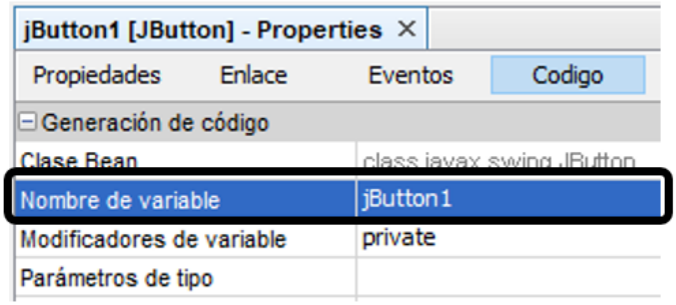



***Figura 4.7.24. Nombre de variable de un botón.***

3. Dando doble clic sobre el botón insertado se abre la pantalla de código mostrándonos el método para programar la acción que va a realizar el botón. Por ejemplo, para salir del programa agregamos la siguiente línea.

*System.exit(0);*

El código que tenemos es el siguiente:

```
97
98      private void salirActionPerformed(java.awt.event.ActionEvent evt) {
99          // TODO add your handling code here:
            System.exit(0);
101     }
```

***Figura 4.7.25. Método para cerrar la aplicación.***

Después de ejecutar el proyecto y dar clic sobre el botón SALIR, la aplicación se cierra.


***Figura 4.7.26. Ejecución de la clase Portada.***

Actividad 4.2

Lee cuidadosamente los reactivos y selecciona la opción correcta.

1. Barra de título, límites, botón para minimizar, botón para cerrar la ventana, función para modificar el tamaño de ventana. Son características que pueden implementarse con la clase:

a) JFrame b) JScrollPane c) JPanel d) JDialog

2. NetBeans IDE es:
   a) Es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas escritos solo en Java.
   b) Es una herramienta para que los programadores puedan compilar, depurar y ejecutar programas escritos solo en Java.
   c) Es una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas escritos en Java y en otros lenguajes.
   d) Es una herramienta para que los programadores puedan compilar programas escritos en Java y en otros lenguajes.

3. Botones para crear un nuevo archivo, crear un nuevo proyecto, abrir proyecto, guardar todo, deshacer, rehacer y ejecutar proyecto, entre otras.

a) Barra título b) Barra de menús c) Barra de Herramientas d) Menú archivo

4. Al seleccionar este comando, en NetBeans, se crea un nuevo proyecto al cual se le asigna un nombre.

a) Archivo Nuevo b) Proyecto Nuevo c) Barra de Herramientas d) Menú Archivo

5. Contiene todos los elementos que se pueden agregar al JFrame, como: Panel, Etiqueta, Botón, Casilla de activación, Botón de opción, Grupo de botones, Lista desplegable, Barra de Menú, Menú, Elemento de Menú, etcétera

a) Barra de herramientas
b) Ventana del diseñador (Design)
c) Editores
d) Paleta de componentes

6. Es la propiedad de Jframe que nos permite modificar los parámetros correspondientes al tamaño de la ventana.

a) resizable b) preferredSize c) foreground d) background

7. Es el componente que despliega una línea de texto, el usuario puede leerla, pero no puede modificarla.

a) JButton b) JTextArea c) JLabel d) JTextField

8. Permite generar un evento cuando el usuario hace clic en él.

a) JButton b) JTextArea c) JLabel d) JTextField

9. Es la propiedad de JLabel que nos permite modificar los parámetros correspondientes al color de la letra.

a) Icon b) Labelfor c) foreground d) background

10. Permite interactuar al programa con el usuario cuando hace clic en él.

a) JTextArea b) JLabel c) JButton d) JTextField

## Actividad 4.3

Elabora un JFrame que se llame Bienvenida y tenga las siguientes características:

- Título: Bienvenida.
- Tamaño: 500 X 150.
- No redimensionable.
- Color de fondo: blanco.
- Mensajes: **Bienvenid@ a Swing**, letra Monotype Cursiva, número 36, negrita y color azul. **Elige el color de fondo**, letra Comic Sans Ms, número 18, negrita, cursiva y color lila.

- 4 botones con las palabras Azul, Amarillo, Rojo, Verde. Que el color del botón tenga color lila, la letra Courier New, número 11 y color morado.
- Cada botón debe tener el nombre de la variable según el color y al oprimirlo debe cambiar el color de fondo del JFrame a azul, amarillo, rojo o verde.

- **Objeto JTextField – Campo de Texto**


Este objeto permite dibujar en el formulario un campo de texto, se utiliza para la introducción de un dato. Así como el control JLabel remplaza la salida estándar System.out.print, el control JTextField cumple la función de la clase Scanner para la entrada de datos.

Algunos de sus métodos más usados son:

| Método | Descripción |
|---|---|
| Strings = texto.getText( ); | Obtiene el texto del campo. |
| texto.setText("nuevo texto"); | Se asigna un nuevo texto. |
| requestFocus( ); | Permite asignar el cursor al objeto de control. |

Sus propiedades más usadas son:

| Propiedad | Descripción |
|---|---|
| Text | Contiene el valor o dato introducido en el cuadro de texto. |
| Font | Permite establecer el tipo de letra del texto en la caja. |
| Border | Para establecer el tipo de borde del cuadro de texto. |
| Enabled | Para habilitar o inhabilitar el uso del objeto de control. |

Es importante considerar que los datos que recibe este objeto son tomados como tipo texto (String) es decir, para que los datos introducidos sean tomados como números, se requiere hacer uso de las siguientes funciones según el tipo de dato numérico.

```
int num = Int.parseInt(jTextField1.getText( ));
float num = Float.parseFloat(jTextField1.getText( ));
double num = Double.parseDouble(jTextField1.getText( ));
```

En donde:

| | |
|---|---|
| num | Representa la variable que va a tomar el valor numérico. |
| jTextField1 | Representa el campo de texto en donde el usuario escribe el valor. |
| getText( ) | Método para acceder al valor escrito en el campo. |

| | |
|---|---|
| parseInt, parseFloat parseDouble | Métodos para convertir los valores del tipo String a los tipos numéricos correspondientes que generalmente son utilizados para realizar cálculos o para algún otro propósito. |

Asimismo, al obtener los resultados deseados y se tengan que mostrar se tiene que realizar el proceso inverso, convertir los cálculos numéricos a tipo texto para desplegar su valor en un jLabel, para ello se ocupa la siguiente función:

```
jLabelN.setText(Double.toString(resultado));
```

En donde:
9-

| | |
|---|---|
| jLabelN | Representa el nombre de la etiqueta en donde se va a mostrar. |
| setText | Función que asigna un valor del tipo String |
| Double | Clase para usar los métodos del tipo de dato double |
| resultado | Variable que contiene el resultado del cálculo a mostrar |

**Ejemplo:**
Elaborar un proyecto que permita calcular el perímetro y área de un cuadrado. En donde se solicite la medida del lado; para poder introducir con punto decimal es necesario utilizar el tipo de dato **double** y para mostrar el resultado se debe usar la función Double.parseDouble.

Para lo anterior, se crea un nuevo Proyecto en NetBeans y se coloca el nombre Grafico y en el paquete generado se coloca un nuevo formulario JFrame.

En el formulario se colocarán los objetos, en este caso, se requieren etiquetas para colocar título, un letrero para solicitar la medida del lado, otras para los letreros de los resultados y los resultados en sí. Además, un campo de texto TextField en donde se introducirá el dato y un botón que realice la acción Calcular.

Se colocarán:

- ✓ Seis etiquetas
- ✓ Un campo de texto
- ✓ Un botón.

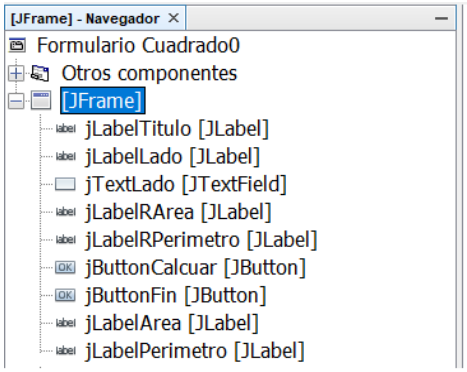
Como se muestra.

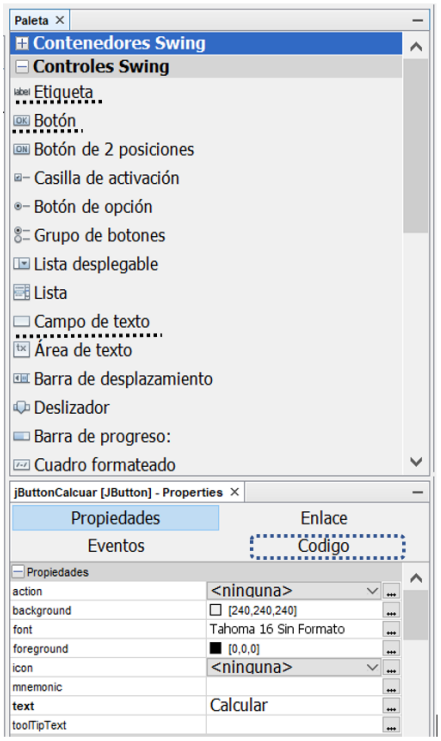
A continuación, se asignarán los textos correspondientes a cada objeto, así como su nombre. En la siguiente tabla se muestran sus valores.

| Objeto | Propiedad - Text | Código – Nombre de la variable |
|---|---|---|
| jLabel1 | Perímetro y Área de un Cuadrado | jLabelTitulo |
| jLabel2 | Lado | jLabelLado |
| jLabel3 | Perímetro | jLabelPerimetro |
| jLabel4 | Vacío (En blanco) | jLabelRPerimetro |
| jLabel5 | Área | jLabelArea |
| jLabel6 | Vacío (En blanco) | jLabelRArea |
| jTextField1 | Vacío (En blanco) | jTextLado |
| jButton1 | Calcular | jButtonCalcular |

Recuerda que estos objetos los encuentras en la Paleta de controles Swing al igual que sus propiedades y código en las pestañas inferiores.

Asimismo, podrás visualizar todos los elementos del proyecto en la ventana Navegador son su nombre asignado.




Cabe mencionar que se puede editar el texto de cualquier objeto al dar doble clic en él, o dando clic con el botón derecho del mouse y en el menú seleccionar Editar Texto.

Se continuará con la programación del botón, para ello, se da doble clic en el botón Calcular y se abre la pestaña de **source** la cual contiene todo el código predefinido, nos dirigimos en donde encontremos el método ActionPerformed del botón correspondiente, esto significa que el código que sea colocado allí, será ejecutado cuando se de clic en el botón correspondiente.

```
private void jButtonCalcularActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

En el cual se introducirá el siguiente código:
Se declaran las tres variables numéricas necesarias

```
double lado, perimetro, area;
```

En seguida se hace la conversión del dato introducido por el usuario en el objeto TextField, esto es, de String a double (numérico), asignando el valor introducido a la variable *lado.*

```
lado = Double.parseDouble(jTextLado.getText( ));
```

Se sigue con el cálculo del perímetro

```
perimetro=lado*4;
```

Después, el valor numérico del perímetro se convierte a texto y se asigna a jLabelRPerimetro

```
jLabelRPerimetro.setText(Double.toString(perimetro));
```

Ahora, se calcula el área y el resultado se asigna al objeto jLabelRArea

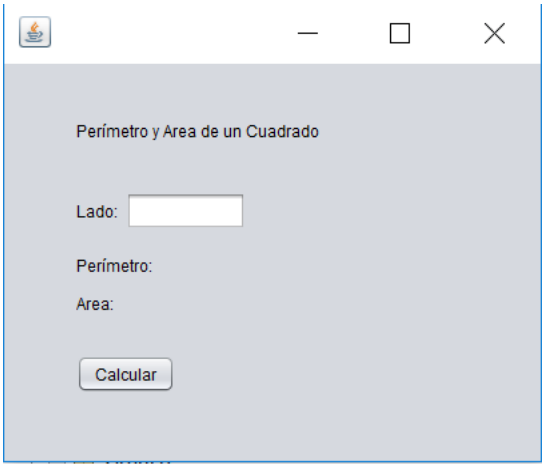
```
area=lado*lado;
jLabelRArea.setText(Double.toString(area));
```

Resultando

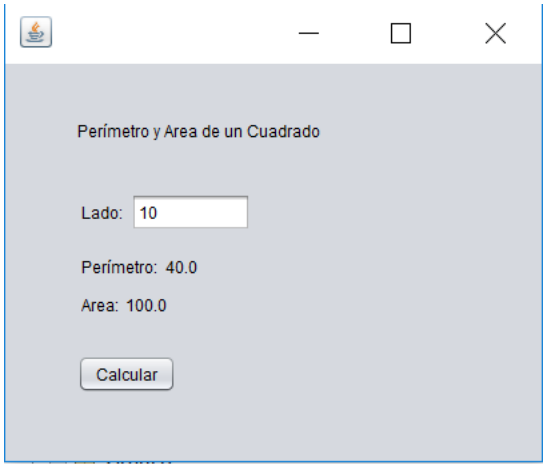
```
private void jButtonCalcuarActionPerformed(java.awt.event.ActionEvent evt) {
    double lado,perimetro,area;
    lado = Double.parseDouble(jTextLado.getText());
    perimetro=lado*4;
    jLabelRPerimetro.setText(Double.toString(perimetro));
    area=lado*lado;
    jLabelRArea.setText(Double.toString(area));
}
```

Al terminar de editar el código, ejecutamos el programa.

**Ejecución del programa**




**Datos y Resultados**




- **JTextArea – Área de texto**


Este control nos permite ingresar o mostrar varias líneas de texto, su limitación es que sólo permite mostrar texto sencillo en el que sólo utilice un tipo de letra.

Este objeto se encuentra en la paleta de Controles Swing y la propiedad para ingresar o modificar el texto es **text**.

Por ejemplo, se retomará el proyecto anterior, para que en un objeto Área de Texto se muestren las características del cuadrado.

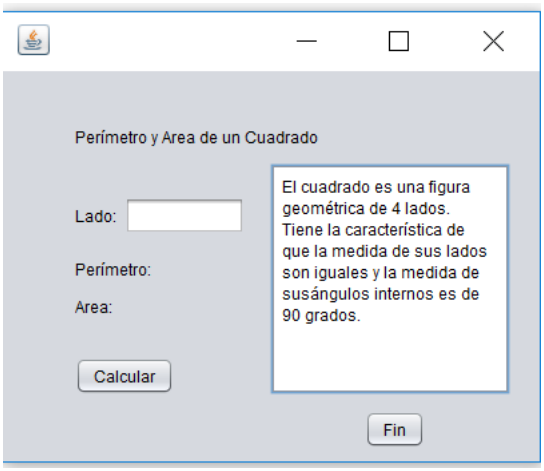
El cuadrado es una figura geométrica de 4 lados. Tiene la característica de que la medida de sus lados son iguales y la medida de sus ángulos internos es de 90 grados.

Para ello, se abre el proyecto anterior en Netbeans, y en el formulario se agrega un objeto Área de texto, se renombra este objeto dirigiéndonos a Codigo – Nombre de Variable y colocar jTextCuadrado.

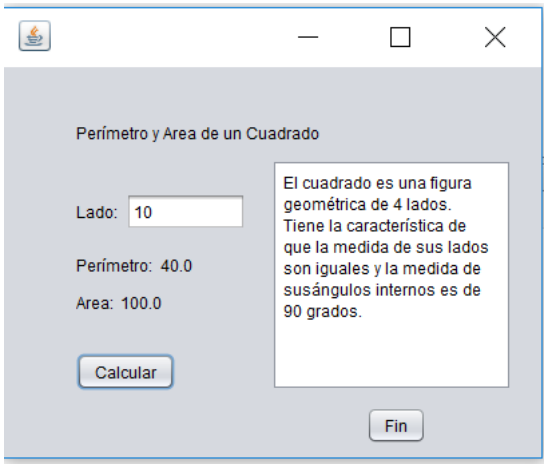
Ahora colocaremos el texto en el objeto jTextArea, se da doble clic al objeto para ingresar el texto indicado.

Para finalizar se ejecuta el programa

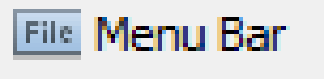
**Ejecución del programa**




**Datos y Resultados**




- **JMenuBar – Barra de menú**

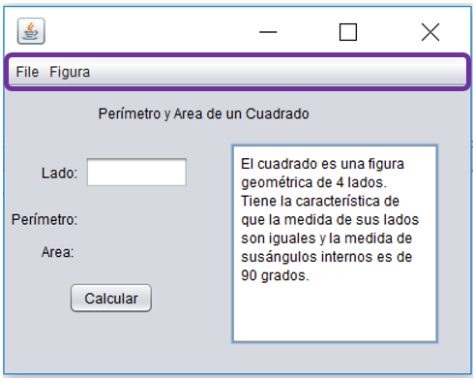



Es la barra de menú principal. Una barra horizontal alargada en la que se colocarán las distintas opciones. A cada una de estas opciones se puede agregar un **JMenu** o un **JMenuItem**.

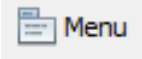
**Ejemplo:**
Retomando el proyecto anterior se le va a agregar un JMenuBar, que tenga como opciones File y Figura.

Se abre el proyecto anterior y en el formulario sólo se agrega un JMenuBar que se encuentra en la paleta de Menús Swing y en automático aparece la barra de menú con las dos opciones solicitadas, si fuera necesario cambiar el nombre basta con dar doble clic a la opción correspondiente y modificar el texto.

Si ejecutamos el programa observaremos la barra de menú.




- **JMenuItem - Elemento de Menú**



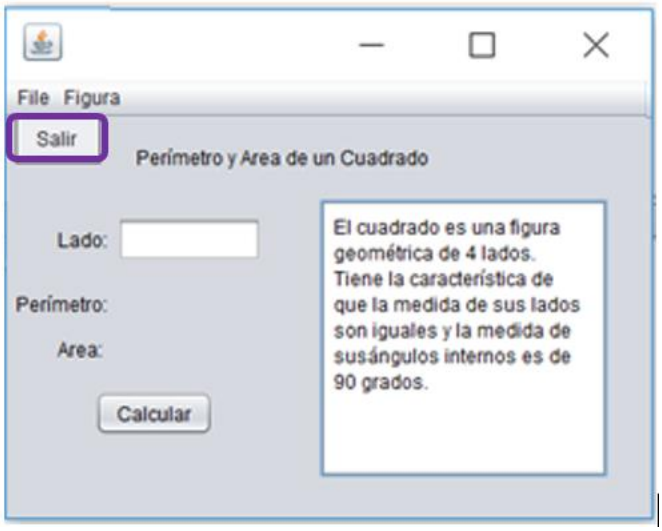

Este objeto sirve para agregar acciones a la opción correspondiente de un JMenuBar. Cuando añadimos uno de estos, tendremos algo que al pulsar despliega un nuevo menú.

Por ejemplo, adecuando el proyecto anterior en la opción File de la barra de menú, que se agregue un elemento de menú con la opción Salir.

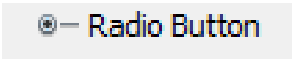
En el JFrame correspondiente, se agrega un elemento de menú a la opción File, se toma de la paleta de Menús Swing el Menú y se coloca encima de File para que correspondan, se da doble clic para editar el texto de JMenuItem por Salir, se le asigna a éste el código pertinente para la salida, dando doble clic para ingresar al código.

```
System.exit(0);
```

```
private void jMenuSalirActionPerformed(java.awt.event.ActionEvent evt(){
        System.exit(0);
}
```




- **JRadioButton – Botón de opción**



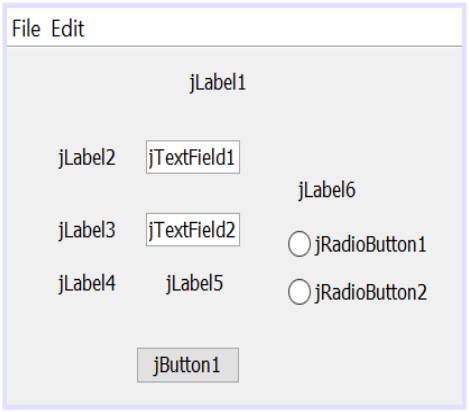

Es un control de botones de opción o botones de radio, se usan cuando se quiere que el usuario pueda elegir una opción entre varias. Para que esto suceda se deben asociar los RadioButton de las opciones posibles, esto se logra con colocarlos dentro de un objeto jPanel - Panel y asociarlos a un grupo de botones. En caso contrario, será posible activar varios botones de opción a la vez.

**Ejemplo:**
Elaborar un programa que permita seleccionar entre el cálculo del perímetro o área de un cuadrado solicitando la medida de uno de sus lados, haciendo uso de RadioButton.

Por lo anterior, se crea un se crea un nuevo Proyecto en NetBeans y en el paquete generado se coloca un nuevo formulario JFrame.

Ahora, en ese formulario se colocarán los objetos necesarios, en este caso, colocaremos los siguientes objetos:

- ✓ Seis etiquetas (jLabel)
- ✓ Dos campos de texto (jTextField)
- ✓ Un botón (jButton)
- ✓ Un panel (jPanel)
- ✓ Dos RadioButton (jRadioButton)
- ✓ Un grupo de botones (jButtonGroup)
- ✓ MenuBar



Como se muestra en la siguiente ventana:

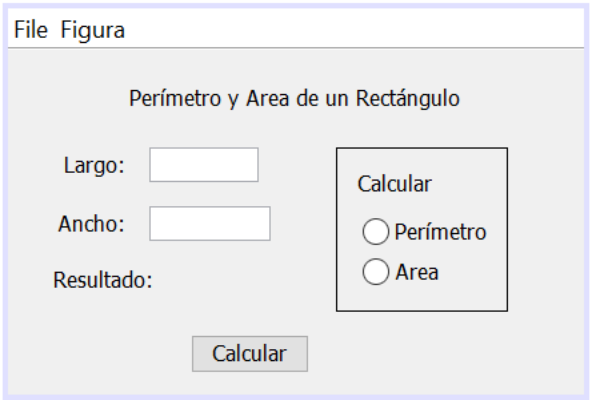
Observa:

- ✓ El jLabel6, jRadioButton1 y jRadioButton2 están incluidos en un Panel, esto es, primero se coloca un Panel y después se agregan dentro de él los objetos requeridos.
- ✓ El grupo de botones no se visualiza, pero se debe contener.

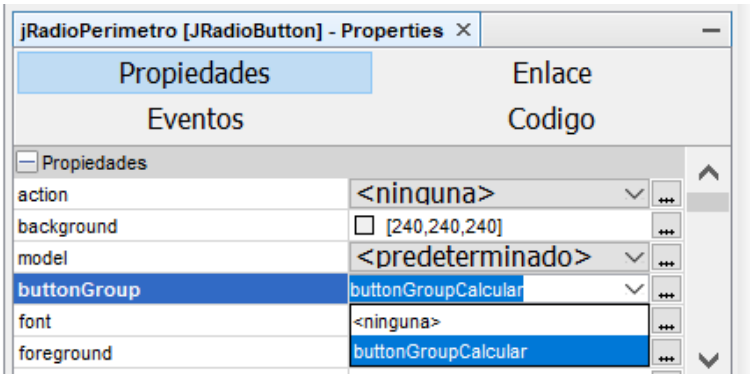
En seguida, se asignarán los textos correspondientes a cada objeto, así como su nombre. A continuación, se muestran sus valores respectivos. En la siguiente tabla se muestran sus valores.

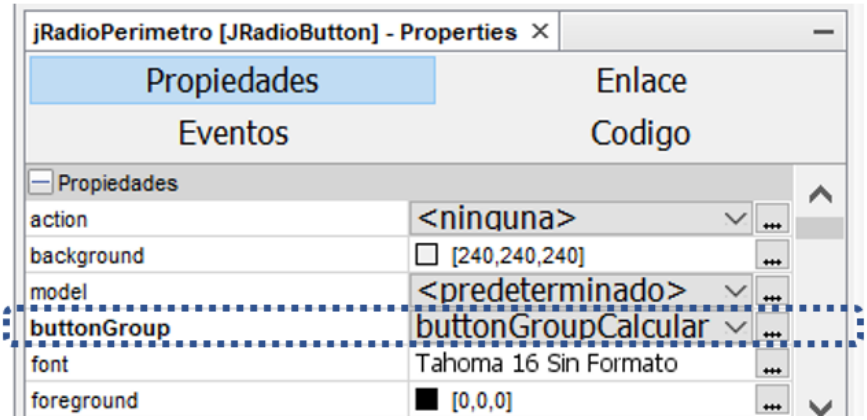
| Objeto | Propiedad - Text | Código – Nombre de la variable |
|---|---|---|
| jLabel1 | Perímetro y Área de un Cuadrado | jLabelTitulo |
| jLabel2 | Largo | jLabelLado |
| jLabel3 | Ancho | jLabelAncho |
| jLabel4 | Resultado | jLabelResultado |
| jLabel5 | Vacío (En blanco) | jLabelRResultado |
| jLabel6 | Calcular | jLabelCalcular (Título del Panel) |
| jTextField1 | Vacío (En blanco) | jTextLargo |
| jTextField2 | Vacío (En blanco) | jTextAncho |
| jRadioButton1 | Perímetro | jRadioPerimetro |
| jRadioButton2 | Área | jRadioArea |
| jButton1 | Calcular | jButtonCalcular |
| jPanel | | jPanelCalcular |
| ButtonGroup | | buttonGroupCalcular |

El JFrame resultante es:




Para terminar, se van a asociar los RadioButton, se selecciona uno, nos dirigimos a la ventana de Propiedades y nos dirigimos a la opción buttonGroup y del lado derecho (valor de la propiedad) se da clic para elegir el correspondiente, en este caso va a ser el buttonGroupCalcular.




**Antes de programar se deben tomar en cuenta las siguientes consideraciones:**

1. Para distinguir que RadioButton es seleccionado se utiliza la sentencia

```
if (jRadioButton1.isSelected( )){
```

   Lo que quiere decir si esta seleccionado el RadioButton1 pasa lo que sigue después de la llave.

Teniendo en cuenta lo anterior se da doble clic al botón Calcular y agregar el siguiente código.

```
private void jButtonCalcularActionPerformed(java.awt.event.ActionEvent evt) {
    double largo,ancho,area,perimetro;

    largo = Double.parseDouble(jTextLargo.getText( ));
    ancho = Double.parseDouble(jTextAncho.getText( ));

    if (jRadioPerimetro.isSelected( )){
       perimetro=largo*2+ancho*2;
       jLabelRResultado.setText(Double.toString(perimetro));
    }

    if (jRadioArea.isSelected( )){
```

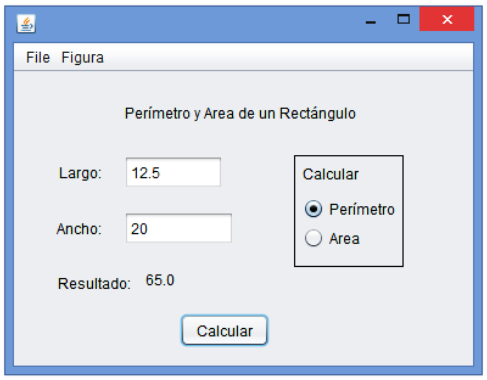
```
        area=largo*ancho;
        jLabelRResultado.setText(Double.toString(area));
    }
}
```

Resultado del código

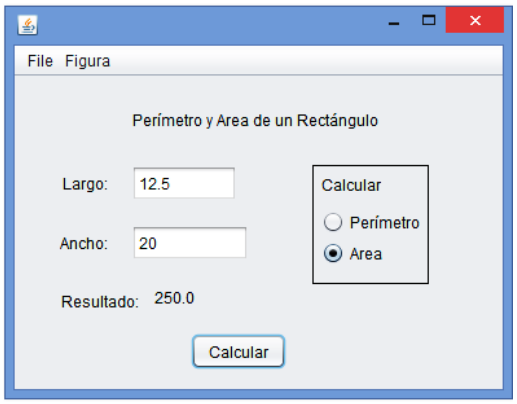
```
private void jButtonCalcuarActionPerformed(java.awt.event.ActionEvent evt) {
    double lado,area,perimetro;

    lado = Double.parseDouble(jTextLado.getText());
    if (jRadioPerimetro.isSelected()){
        perimetro=lado*4;
        jLabelRResultado.setText(Double.toString(perimetro));
    }

    if (jRadioArea.isSelected()){
        area=lado*lado;
        jLabelRResultado.setText(Double.toString(area));
    }

}
```

Ejecución del programa

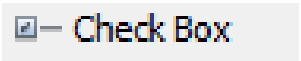
Aquí se eligió calcular el perímetro de un rectángulo, de largo 12.5 y ancho 20, mostrando el resultado.




Ejecución calculando el área




- **JCheckBox – Casilla de verificación**

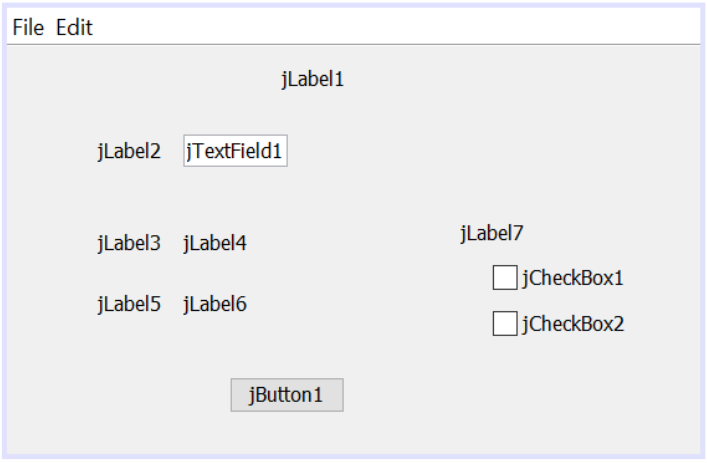



Los cuadros de verificación son útiles para seleccionar uno o más elementos de una lista. Por ejemplo, se va a calcular el perímetro y área de un círculo, con el fin de poder seleccionar una de las opciones a ambas para que se muestre el cálculo.

Para lo anterior, se abre el proyecto Grafico y se genera un nuevo Formulario (JFrame), en el cual se colocarán los siguientes objetos:

- ✓ Siete etiquetas (jLabel)
- ✓ Un campo de texto (jTextField)
- ✓ Un botón (jButton)
- ✓ Un panel (jPanel)
- ✓ Dos casillas de verificación (jCheckBox)
- ✓ MenuBar

Distribuidos como se muestra

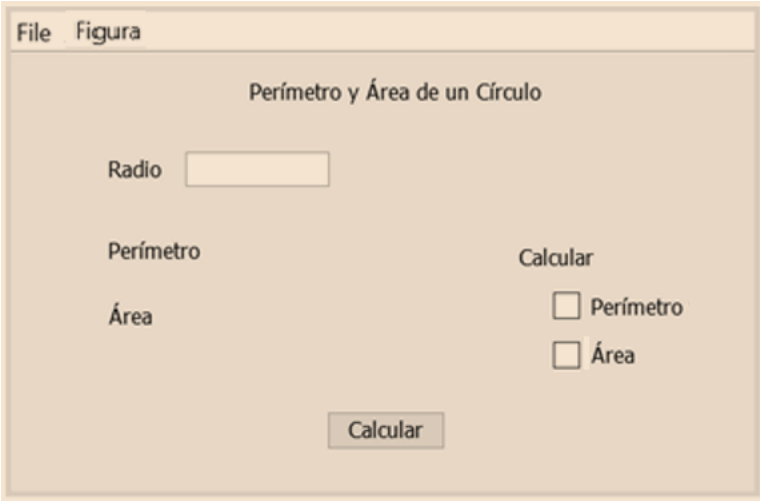



En seguida, se asignarán los textos correspondientes a cada objeto, así como su nombre. A continuación, se muestran sus valores respectivos. En la siguiente tabla se presentan sus valores.

| Objeto | Propiedad - Text | Código – Nombre de la variable |
|---|---|---|
| jLabel1 | Perímetro y Área de un Cuadrado | jLabelTitulo |
| jLabel2 | Radio | jLabelRadio |
| jLabel3 | Perímetro | jLabelPerimetro |
| jLabel4 | Vacío (En blanco) | jLabelRPerimetro |
| jLabel5 | Área | jLabelArea |
| jLabel6 | Vacío (En blanco) | jLabelRArea |
| jLabel7 | Calcular | jLabelCalcular (Título del Panel) |
| jTextField1 | Vacío (En blanco) | jTextLado |
| jCheckBox1 | Perímetro | jCheckBoxPerimetro |
| jCheckBox2 | Área | jCheckBoxArea |
| jButton1 | Calcular | jButtonCalcular |
| jPanel | | jPanelCalcular |

**Nota: Los objetos jLabel7, jCheckBox1 y 2 van dentro del jPanel.**

**Para que resulte:**




Antes de programar se deben tomar en cuenta las siguientes consideraciones:

1. Para obtener el valor de Pi ($\pi$) nos auxiliamos de la biblioteca Math, haciendo uso de la función

**Math.PI**

2. Para distinguir si un CheckBox es seleccionado se utiliza la sentencia

**if (jCheckPerimetro.isSelected( )){**

Lo que quiere decir si esta seleccionado el jCheckPerimetro pasa…

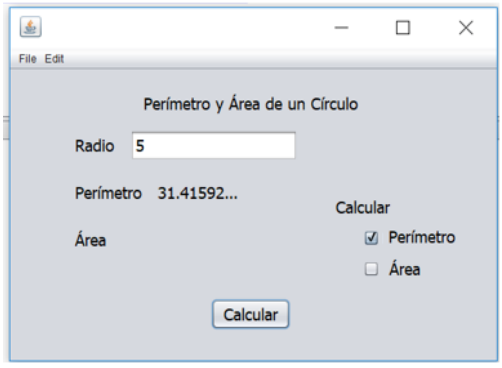
Teniendo en cuenta lo anterior se da doble clic al botón Calcular y agregar el siguiente código.

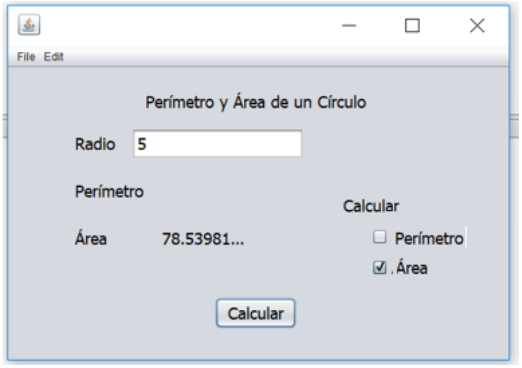
A continuación, se debe programar el botón Calcular.

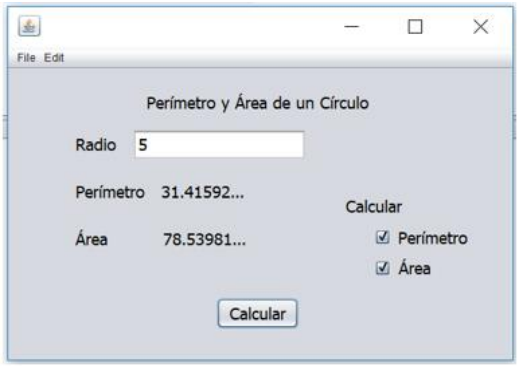
```
private void jButtonCalcularActionPerformed(java.awt.event.ActionEvent evt) {
    double radio,area,perimetro;

     radio = Double.parseDouble(jTextRadio.getText( ));

     if (jCheckPerimetro.isSelected( )){
        perimetro=Math.PI*2*radio;
        jLabelRPerimetro.setText(Double.toString(perimetro));
```

```
    } else {
       jLabelRPerimetro.setText("");
    }

    if (jCheckArea.isSelected( )){
       area=Math.PI*radio*radio;
       jLabelRArea.setText(Double.toString(area));
    } else {
       jLabelRArea.setText("");
}
```

## Ejecución

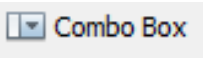



- 

- **JComboBox – Lista desplegable**




Este control nos permite seleccionar un valor entre una serie de opciones, es muy útil cuando se sabe cuáles son los datos que se pueden ingresar, esto es de ayuda para evitar errores.

De una lista desplegable podemos obtener dos tipos de valores ellos son:

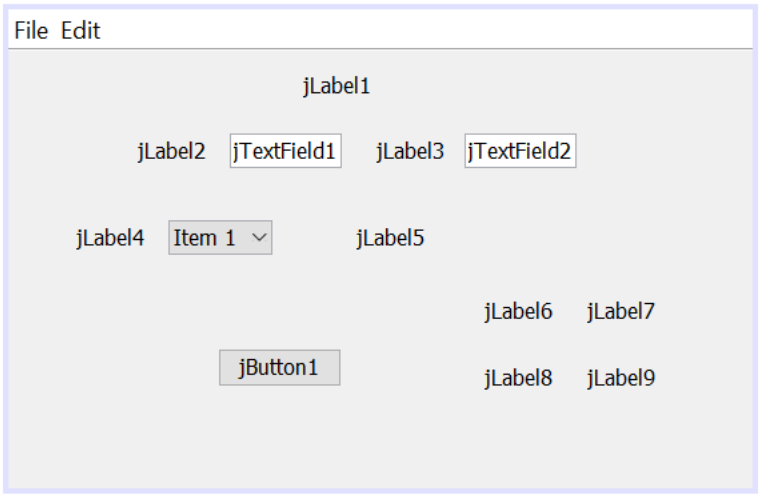
- Índice o index (número de elemento).
- El ítem Seleccionado (nombre de elemento).

Por ejemplo, realizar un programa para calcular el perímetro y área de polígonos regulares con el control jComboBox desplegar el número de lados del polígono del pentágono hasta el decágono (5, 6, 7, 8, 9 y 10 lados).

Para lo anterior se abre el Proyecto Grafico y se genera un formulario nuevo JFrame para colocar los siguientes objetos:

- ✓ Nueve etiquetas (jLabel)
- ✓ Dos campos de texto (jTextField)
- ✓ Un botón (jButton)
- ✓ Un panel (jPanel)
- ✓ Lista desplegable (jComboBox)
- ✓ BarMenu

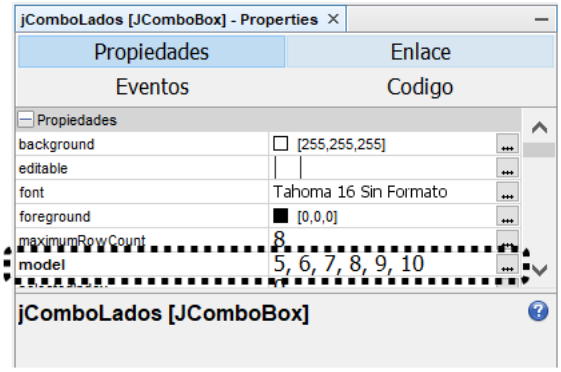
Como se muestra en la ventana que sigue:



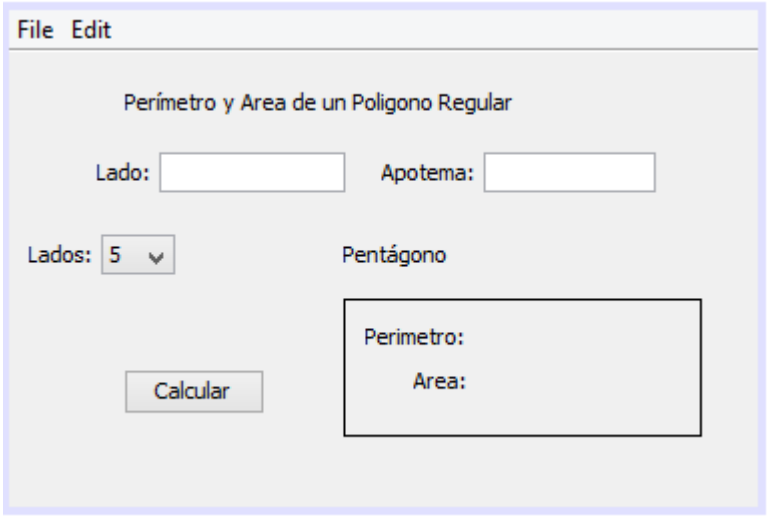

En seguida, se asignarán los textos correspondientes a cada objeto, así como su nombre. A continuación, se muestran sus valores respectivos.

| Objeto | Propiedad - Text | Código – Nombre de la variable |
|---|---|---|
| jLabel1 | Perímetro y Área de un Polígono Regular | jLabelTitulo |
| jLabel2 | Lado: | jLabelLado |
| jLabel3 | Apotema: | jLabelApotema |
| jLabel4 | Lados: | jLabelLados |
| jLabel5 | Pentágono | jLabelNombre |
| jLabel6 | Perímetro | jLabelPerimetro |
| jLabel7 | Vacío (En blanco) | jLabelRPerimetro |
| jLabel8 | Área | jLabelArea |
| jLabel9 | Vacío (En blanco) | jLabelRArea |
| jTextField1 | Vacío (En blanco) | jTextLado |
| jTextField2 | Vacío (En blanco) | jTextApotema |
| jButton1 | Calcular | jButtonCalcular |
| jComboBox1 | | jComboLados |
| jPanel | | |

Para agregar los valores de la lista desplegable, se selecciona el ComboBox correspondiente, nos dirigimos a la propiedad model y se colocan los posibles valores separados por una coma, en este caso deben ser: 5, 6, 7, 8, 9, y 10. También, se pueden agregar dando doble clic a los tres puntos que se encuentran en la propiedad model del lado derecho y en la ventana que aparece se escriben los posibles valores en lista




Por lo anterior, la lista desplegable ya cuenta con sus posibles valores y nuestro formulario queda finalmente como se muestra:




Para empezar a programar se deben tomar en cuenta las siguientes consideraciones:

1. Para recuperar el valor seleccionado de la lista desplegable (jComboBox) se utiliza la función:

```
jComboLados.getSelectedItem( )
```

Enseguida escribimos el siguiente código en el botón calcular.

```
private void jButtonCalcularActionPerformed(java.awt.event.ActionEvent evt) {
      double lado,apotema,area,perimetro;
      int numLados;

      lado = Double.parseDouble(jTextLado.getText( ));
      apotema = Double.parseDouble(jTextApotema.getText( ));

      numLados = Integer.parseInt(jComboLados.getSelectedItem( ).toString( ));
      perimetro=lado*numLados;
      jLabelRPerimetro.setText(Double.toString(perimetro));
      area=perimetro*apotema/2;
      jLabelRArea.setText(Double.toString(area));
   }
```

Ahora, escribimos el siguiente código en el menu File-Salir. Con este código se cierra la ventana y termina el programa.

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt)
  {
      System.exit(0);
```

```
}
```

Para escribir el nombre correspondiente del polígono según el número de sus lados; escribimos el código para el comboBox “jComboLados” que contiene los nombres de los polígonos.

- Se define el arreglo "polígonos” el cual contiene el nombre del polígono de acuerdo al número de lados. El índice en el arreglo corresponde al número de lados y a partir de allí se obtiene el nombre del polígono. Los polígonos con 0 a 4 lados no aparecen en el combo por eso en el arreglo no hay ninguna descripción.

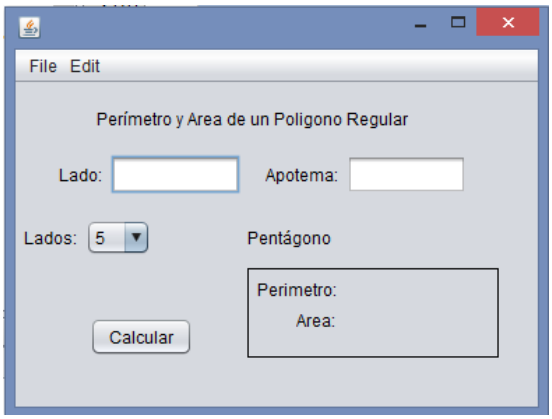
En este caso, se define el arreglo de la siguiente manera:

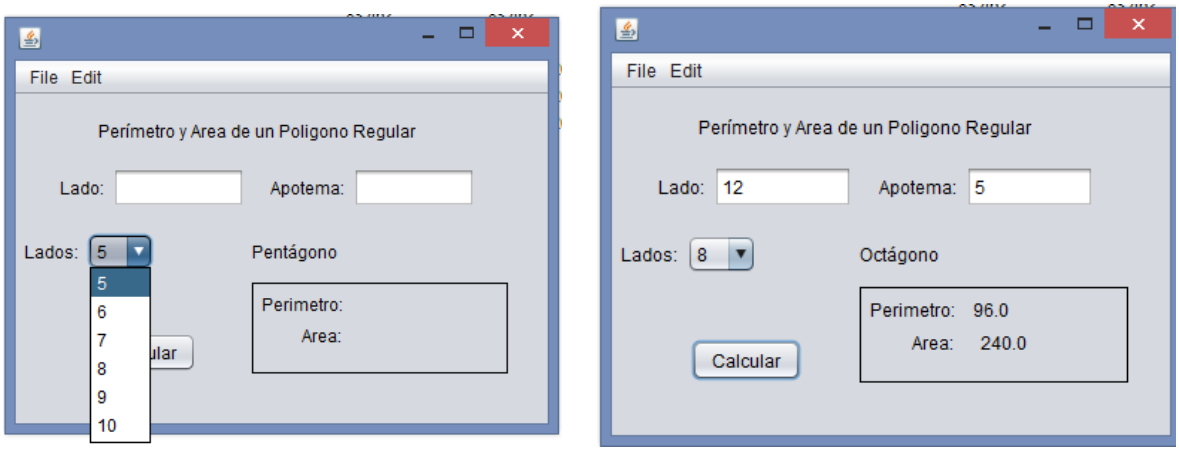
```
String[ ] poligonos = {"","","","","","Pentágono","Hexágono","Heptágono","Octágono","Eneágono","Decágono"};
```

Ahora, se debe dar doble clic al comboBox “jComboLados” para incluir el siguiente código:

```
private void jComboLadosActionPerformed(java.awt.event.ActionEvent evt)
  {
      int numLados;
  //Se define el arreglo para los nombres de los polígonos de acuerdo al número de lados
  String[ ] poligonos =
{"","","","","","","Pentágono","Hexágono","Heptágono","Octágono","Eneágono","Decágono"};
      numLados = Integer.parseInt(jComboLados.getSelectedItem( ).toString( ));
      jLabelNombre.setText(poligonos[numLados]);
      jLabelRPerimetro.setText(" "); //Limpia el campo
      jLabelRArea.setText(" "); //Limpia el campo
   }
```

**Ejecución**






- **JMenú**

Un menú es un objeto que se le añade al jMenuBar, y sirve para almacenar items comunes. Esto es, se crean los elementos comunes para después agregarlos como ItemsMenu al Menu.

Para ser más específicos, vamos a crear un jMenu que se llame Triángulo y como elementos (jItemMenu) se contemplen Equilátero, Isósceles y Escaleno.

Para obtener el área y perímetro de cada uno de los triángulos se hará uso de las siguientes fórmulas:

| Triángulo | Perímetro | Área |
|---|---|---|
| Equilátero | perimetro=3*lado; | $Área = \frac{\sqrt{3}}{4} \cdot a^2$ |
| Isósceles | perimetro=base+2*(Math.sqrt(Math.pow(base/2, 2)+Math.pow(altura,2))); | area=(base*altura)/2; |
| Escaleno | perimetro=ladoA+ladoB+ladoC; | $Área = \sqrt{s(s-a)(s-b)}$ con $a$, $b$, $c$ los tres lados semiperímetro $s = \frac{a+}{}$ //formula de Herón s=(perimetro/2); //semiperimetro area=Math.sqrt(s*(s-ladoA)*(s-ladoB)*(s-ladoC)); |

Para lo anterior, vamos a crear el programa TrianguloEquilatero, abrimos el proyecto Grafico, para obtener el perímetro y área de un triángulo equilátero.

En el formulario, se colocan los siguientes objetos:

- ✓ Seis etiquetas (jLabel)
- ✓ Un campo de texto (jTextField)
- ✓ Un botón (jButton)
- ✓ Un área de texto (jTextAreal)
- ✓ Barra de Menu
- ✓ Menu
- ✓ Dos ItemMenu (Sobre el Menú)


En seguida, se asignarán los textos correspondientes a cada objeto, así como su nombre. A continuación, se muestran sus valores respectivos.

| Objeto | Propiedad - Text | Código – Nombre de la variable |
|---|---|---|
| jLabel1 | Perímetro y Área de un Triángulo Equilátero | jLabelTitulo |
| jLabel2 | Lado: | jLabelLado |
| jLabel3 | Perímetro | jLabelPerimetro |
| jLabel4 | Vacío (En blanco) | jLabelRPerimetro |
| jLabel5 | Área | jLabelArea |
| jLabel6 | Vacío (En blanco) | jLabelRArea |
| jTextField1 | Vacío (En blanco) | jTextLado |
| jButton1 | Calcular | jButtonCalcular |

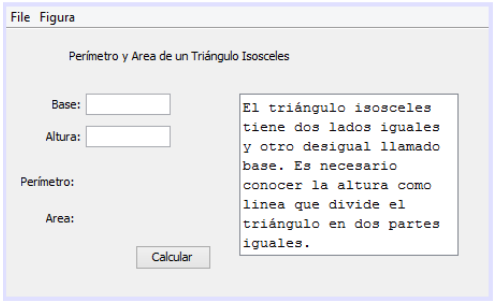
| Objeto | Propiedad - Text | Código – Nombre de la variable |
|---|---|---|
| jTextArea1 | El triángulo equilátero sus tres lados miden lo mismo y sus ángulos internos también son iguales y miden 60 grados ya que la suma de los ángulos interiores de un triángulo es 180 grados. | jTextTriamguloEq |

Ahora pasaremos al código del botón Calcular

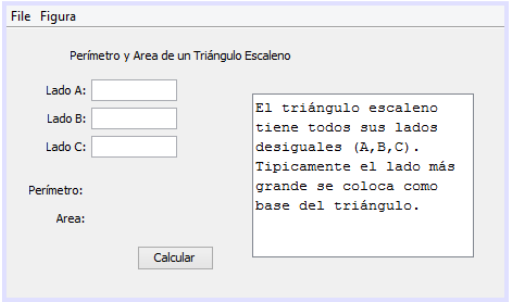
```
private void jButtonCalcularActionPerformed(java.awt.event.ActionEvent evt)
  {
     double lado,area,perimetro;
     lado = Double.parseDouble(jTextLado.getText( ));
     perimetro=3*lado;
     jLabelRPerimetro.setText(Double.toString(perimetro));
     area=(Math.sqrt(3.0)/4)*Math.pow(lado, 2);
     jLabelRArea.setText(Double.toString(area));
   }
```

Para integrar los otros dos tipos de triángulo se realiza lo correspondiente de la misma forma como se generó el TrianguloEquilatero. Esperando la ventana de cada tipo de triángulo como sigue:

Triángulo Isósceles




Triángulo Escaleno




Después de contar con los tres programas en cada uno de los Items del Menú y se va a instanciar el tipo de triángulo correspondiente al menú y se va a hacer visible. Por ejemplo, nos encontramos en el programa de TrianguloEquilatero y seleccionamos del menú Triángulo a Isósceles, lo que correspondería:

Instanciando al tipo de triángulo

```
TrianguloIsosceles isosceles = new TrianguloIsosceles( );
```

Y para hacer visible el programa correspondiente se ocupa la siguiente sentencia:

```
isosceles.setVisible(true);
```

Para colocar lo anterior en el código se da doble clic en el menú Triangulo en la opción Isósceles y queda:

```
private void jMenuTrianguloIsoscelesActionPerformed(java.awt.event.ActionEvent evt)
  {
     TrianguloIsosceles isosceles = new TrianguloIsosceles( );
     isosceles.setVisible(true);
     this.setVisible(false);
   }
```

Código para el Triángulo Isósceles

- Botón Calcular

```
private void jButtonCalcularActionPerformed(java.awt.event.ActionEvent evt)
 {
     double ladoA,ladoB,ladoC,altura,area,perimetro,s;
     //Se ingresan los datos
     ladoA = Double.parseDouble(jTextLadoA.getText( ));
     ladoB = Double.parseDouble(jTextLadoB.getText( ));
     ladoC = Double.parseDouble(jTextLadoC.getText( ));
     //Se calcula el perímetro y el área
     perimetro=ladoA+ladoB+ladoC;
     jLabelRPerimetro.setText(Double.toString(perimetro));
     //formula de Herón
     s=(perimetro/2); //semiperimetro
     area=Math.sqrt(s*(s-ladoA)*(s-ladoB)*(s-ladoC));
     jLabelRArea.setText(Double.toString(area));
  }
```

- Menú MenuTrianguloEscaleno

```
private void jMenuTrianguloEscalenoActionPerformed(java.awt.event.ActionEvent evt)
  {
     TrianguloEscaleno escaleno = new TrianguloEscaleno( );
     escaleno.setVisible(true);
     this.setVisible(false);
   }
```

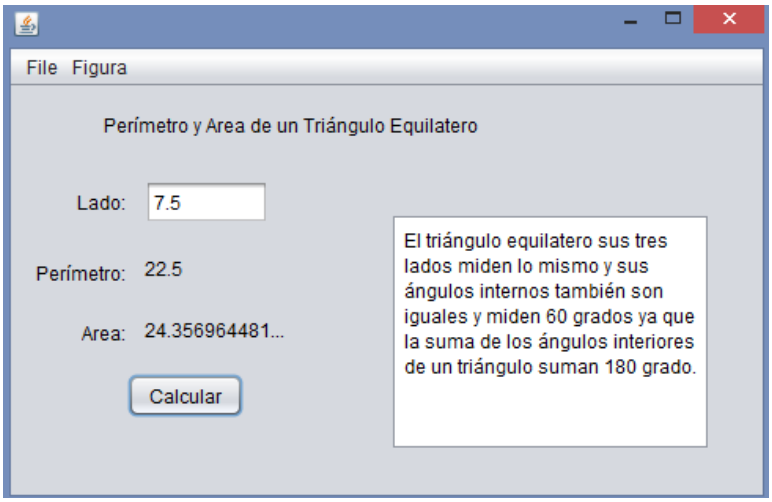
**Código para el Triángulo Escaleno**

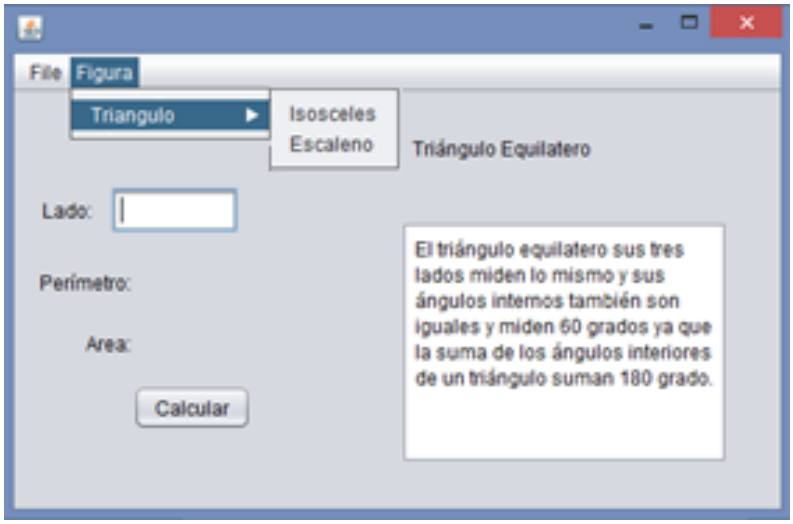
- Botón Calcular

```
private void jButtonCalcularActionPerformed(java.awt.event.ActionEvent evt)
  {
     double base,altura,area,perimetro;
     //Se ingresan los datos
     base = Double.parseDouble(jTextBase.getText( ));
     altura = Double.parseDouble(jTextAltura.getText( ));
     //Se calcula el perímetro y el área
     perimetro=base+2*(Math.sqrt(Math.pow(base/2,2)+Math.pow(altura,2)));
     jLabelRPerimetro.setText(Double.toString(perimetro));
     area=(base*altura)/2;
     jLabelRArea.setText(Double.toString(area));
   }
```

Con lo anterior ya se pueden visualizar los diferentes tipos de triángulos para calcular su perímetro y área.

A continuación, se muestra la ejecución del programa:




- **Integración de un proyecto**

En esta sección se realizará la integración de los programas anteriores para poder navegar a través de la barra de Menú y poder escoger la figura que se quiere. Para ello, nos apoyaremos de la barra de menú de cada programa, en la opción Figura.

El siguiente método corresponde al código que se ejecutará al seleccionar el menú de la figura Cuadrado. Los formularios o jFrames son objetos que pueden instanciarse y de esa forma modificar sus atributos dinámicamente.

Primero se crea una instancia del formulario o jFrame Cuadrado y luego se muestra activándose con el atributo setVisible(true) de la misma forma el formulario actual referenciado como **this** se cierra mediante setVisible(false).

```
private void jMenuCuadradoActionPerformed(java.awt.event.ActionEvent evt) {
```

```
    Cuadrado cuadrado = new Cuadrado( );
    cuadrado.setVisible(true);
    this.setVisible(false);
}
```

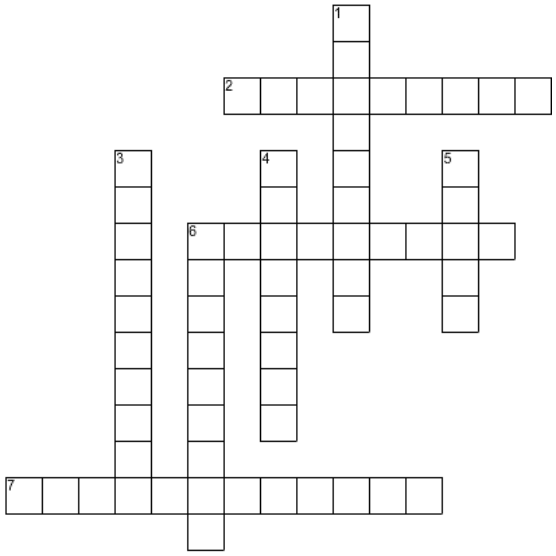
El siguiente método sirve para terminar el proceso actual cerrando todas las ventanas o formularios activos. Se puede poner dentro de un botón de **Fin**, solo que en este caso se colocó en un elemento del menú **File** con el nombre **Salir**.

```
private void jMenuSalirActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}
```

## Actividad 4.4

### Crucigrama

Resuelve el siguiente crucigrama.
Instrucciones. Verifica la definición de cada número en vertical u horizontal para colocar el concepto dentro de las celdas correspondientes.




| Verticales | Horizontales |
|---|---|
| 1. Es un elemento del menú y al ser pulsado genera un evento, o sea, abre una ventana y pide datos. | 2. Se utiliza para seleccionar uno o más elementos de una lista. |

| Verticales | Horizontales |
|---|---|
| 3. Se utiliza para la introducción de un dato | 6. Permite ingresar o mostrar varias líneas de texto. |
| 4. Es la barra de menú principal | 7. Este control se usa cuando se quiere que el usuario pueda elegir una opción de entre varias |
| 5. Este objeto sirve para agregar acciones a la opción correspondiente de un menú. | |
| 6. Este control nos permite seleccionar una serie de opciones | |

Actividad 4.5

Relaciona la columna con el objeto que le corresponda

| | | |
|---|---|---|
| 1.  | ( ) | **Campo de Texto** |
| 2.  | ( ) | **Menú** |
| 3.  | ( ) | **Barra de menú** |
| 4.  | ( ) | **Área de texto** |
| 5.  | ( ) | **Botón de opción** |
| 6.  | ( ) | **CheckBox** |
| 7.  | ( ) | **Menú Item** |
| 8.  | ( ) | **Combo Box** |

## 4.8 Clase Graphics

**Propósitos**

El alumno:

Elabora programas con interfaz gráfica de usuario aplicando las Clases: setColor, drawLine, drawRect, drawRoundrect, drawOval, drawPolygon.

Desarrolla programas haciendo usos de los métodos: fillRect, fillRoundRect, fillOval, fillPolygon

Esta clase se encuentra en el paquete java.awt y proporciona varios métodos para dibujar texto y figuras como elipses, cuadrados, líneas, entre otros, en la pantalla.

Adicionalmente proporciona el entorno de trabajo para cualquier operación gráfica que se realice dentro del Kit de Herramientas de Ventana Abstracta, AWT (Abstract Window Toolkit, por sus siglas en ingles).

Para poder dibujar en Java se necesita un contexto gráfico válido, representado por una instancia de la clase Graphics. Pero esto no se puede hacer directamente; así que debemos crear un componente y pasarlo al programa como un argumento al método paint( ). El único argumento del método paint( ) es un objeto de esta clase.

Algunos métodos que tiene la clase Graphics se muestran en la siguiente tabla:

| Método | Descripción |
|---|---|
| setColor(Color c) | Establece un color actual. |
| drawLine(int x1, int y1, int x2, int y2) | Dibuja una línea entre dos puntos. |
| drawRect(int x, int y, int anchura, int altura) | Dibuja las líneas de un rectángulo. |
| drawOval(int x, int y, int anchura, int altura) | Dibuja el contorno de un óvalo. |

***Tabla 4.8.1. Métodos de la clase Graphics.***

Para dibujar en Java se debe comprender su sistema de coordenadas, esto es un esquema para identificar cada uno de los puntos en la pantalla.

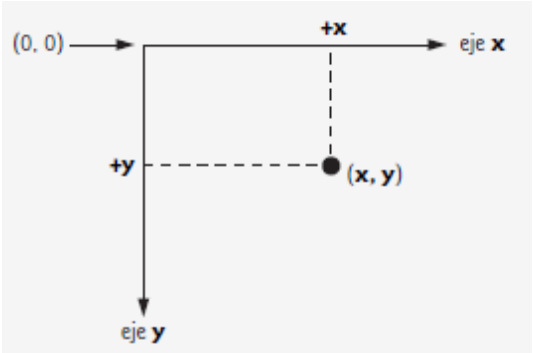



***Figura 4.8.1 Sistema de coordenadas gráficas en Java.***

Como muestra la figura 4.8.1, la esquina superior izquierda de un componente de la interfaz gráfica, por ejemplo, un JFrame tiene las coordenadas (0,0). Un par de coordenadas está compuesto por una coordenada **x** (la coordenada horizontal) y una coordenada **y** (la coordenada vertical). La coordenada horizontal se desplaza de izquierda a derecha y la

coordenada vertical se desplaza de arriba hacia abajo. Las coordenadas se miden siempre en pixeles, esto es la unidad más pequeña y diminuta de una imagen digital.

Para dibujar, utilizando el IDE NetBeans, vamos a realizar los siguientes pasos:

1. Crear un nuevo JFrame, definir las dimensiones de ancho y alto de 200 X 200, y deshabilitar la opción de redimensionar en la opción de propiedades como se muestra en la figura:

| | |
|---|---|
| **preferredSize** | [200, 200] |
| **resizable** | ☐ |

***Figura 4.8.2 Propiedades del JFrame***

Después de esto tenemos un plano de coordenadas, de 0 a 200 en el eje horizontal y de 0 a 200 en el eje vertical.

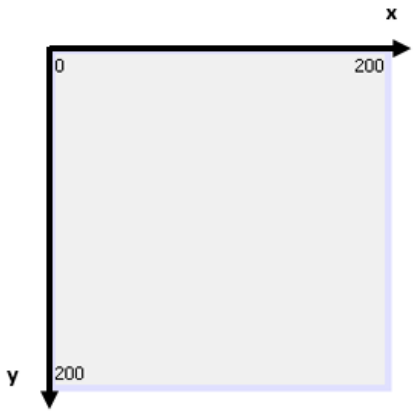



***Figura 4.8.3. Coordenadas de un JFrame de 200 x 200***

2. En la pestaña de código (source) debemos importar las librerías java.awt.Color y java.awt.Graphics. Nótese como está escrito después del paquete y antes de la clase.

```
6   package CiberII;
7
9   import java.awt.Color;
    import java.awt.Graphics;
10
11  /**
12   *
13   * @author Gaby López
14   */
15  public class Grafico extends javax.swing.JFrame {
```

***Figura 4.8.4. Librerías java.awt.Color y java.awt.Graphics.***

3. Se sobreescribe el método paint que es heredado de la clase JFrame. Este método se declara después del método constructor de la clase, como se muestra a continuación.

```
15  public class Grafico extends javax.swing.JFrame {
16
17      /**
18       * Creates new form Grafico1
19       */
20
21      public Grafico() {
22          initComponents();
23      }
24
25      public void paint (Graphics g)
26      {
27          super.paint(g);
28
29      }
```

***Figura 4.8.5. Método paint.***

El método paint se ejecuta cada vez que el JFrame debe ser redibujado y recibe como parámetro un objeto de la clase Graphics. Este método nos permite acceder al fondo del JFrame para poder dibujar líneas, rectángulos, etcétera.

Lo primero que hacemos es llamar al método paint de la clase padre y esto se hace con la palabra super y el parámetro **g**.

Después de estos tres pasos, podemos llamar los métodos necesarios para hacer dibujos tal como se explica a continuación.

- **setColor**

Con este método se establece el color actual del contexto gráfico en el color especificado como parámetro (Color c), el modo de escribirlo se muestra a continuación:

Sintaxis:

```
                    setColor   (Color c)
          _________/\_____   ____/\________
Nombre del método   Parámetro de color
```

Los valores que puede tomar el parámetro de color se ven representados en la siguiente tabla de argumentos de color.

| Argumento de color | Color |
|---|---|
| Color.black | Negro. |
| Color.blue | Azul. |
| Color.cyan | Cian. |
| Color.darkGray | Gris oscuro. |
| Color.gray | Gris. |

| | |
|---|---|
| Color.green | Verde. |
| Color.lightGray | Gris claro. |
| Color.magenta | Magenta. |
| Color.orange | Naranja. |
| Color.pink | Rosa. |
| Color.red | Rojo. |
| Color.white | Blanco. |
| Color.yellow | Amarillo. |

***Tabla 4.8.2. Argumentos de color.***

Debemos cambiar el parámetro por el argumento, por ejemplo: (Color.blue). Para indicar el color de cada figura que dibujemos debemos escribir el método setColor antes de la figura, de lo contrario el color definido inicialmente será el que se emplee en todos los dibujos.

SetColor es un método de instancia por lo tanto se llama con el objeto (g) de la clase Graphics como se muestra a continuación:

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.blue); //elegimos el color azul
}
```

Si ejecutamos la aplicación solamente va a mostrar el JFrame vacío ya que no hemos dibujado ninguna figura, solo hemos definido el color con el que se va a dibujar.

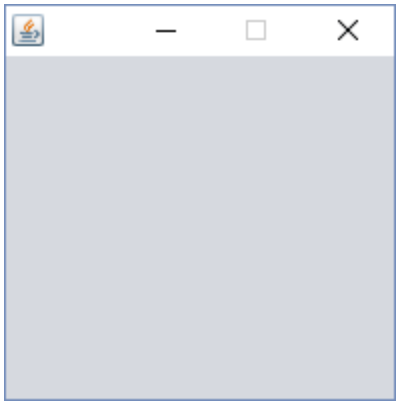


***Figura 4.8.6. Ejemplo de JFrame.***

- **drawLine**

Esta sentencia dibuja una línea, usando el color actual, entre los puntos (x1, y1) y (x2, y2) en el sistema de coordenadas de este contexto gráfico. Los parámetros que tiene el método son:

Sintaxis:

drawLine (int x1, int y1, int x2, int y2);

Punto inicial — Punto final

Los primeros dos argumentos son las coordenadas (x,y) para un punto inicial de la línea, y los últimos dos argumentos son las coordenadas para el otro punto final de la línea.

**Ejemplo:**

En el JFrame anterior donde indicamos el color azul, dibujamos una línea del punto (0,0) al (200,200) como sigue:

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.blue); //elegimos el color azul
        g.drawLine(0,0,200,200); //dibujamos una línea
}
```

El resultado es una línea de color azul con inicio en las coordenadas (0,0) y final en (200,200). La coordenada (0,0) esta exactamente en la esquina superior izquierda de la ventana, por ese motivo no se muestra la línea desde el inicio.

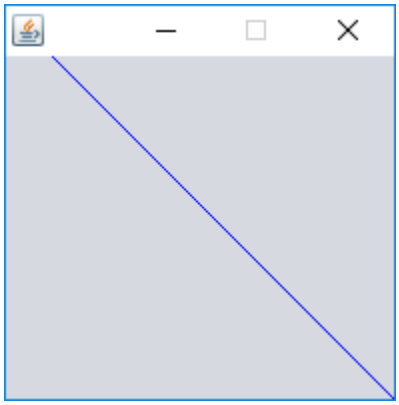
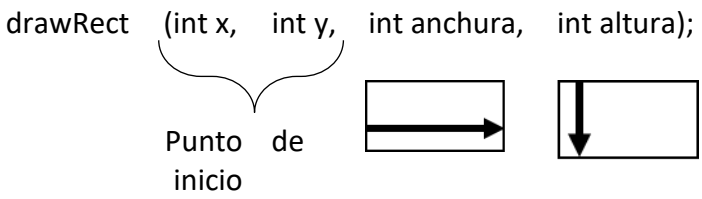


***Figura 4.8.7. Dibujo de una línea.***

- **drawRect**

Dibuja el contorno del rectángulo especificando una coordenada (x,y) de inicio, la dimensión de anchura y la dimensión de altura como lo indica la siguiente sintaxis:




**Ejemplo:**

Dibujar un rectángulo color magenta desde la coordenada de inicio (50,50), con ancho 30 y alto 100. El método paint quedaría de la siguiente forma:

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.magenta); //elegimos el color magenta
        g.drawRect(50,50,30,100); //dibujamos un rectángulo
}
```

La ejecución de la aplicación es un rectángulo de contorno color magenta con dimensiones de 30 X 100.

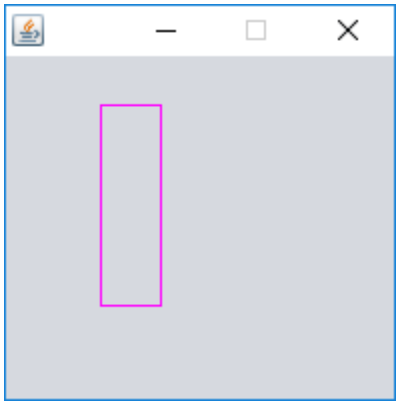


***Figura 4.8.8. Dibujo de un rectángulo.***

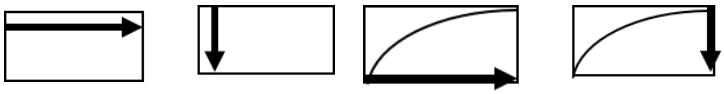
- **drawRoundRect**

Este método dibuja un rectángulo de esquinas redondeadas con el color actual del contexto gráfico. Además de especificar la coordenada de inicio, el ancho y alto, se debe indicar el largo y ancho de los ángulos de las esquinas. Estos dos argumentos determinan que tan lejos de los límites del rectángulo debe comenzar el arco en las esquinas. El orden como deben escribirse los parámetros se muestra a continuación:

Sintaxis:

drawRoundRect (int x, int y, int anchura, int altura, int arcAncho, int arcAlto);

Punto de inicio



El valor del arcAncho es la distancia de una esquina de rectángulo hacia el eje horizontal y el valor del arcAlto es la distancia de la esquina sobre el eje vertical. Para un rectángulo de ancho 60 y alto 30 como el que muestra la figura 4.8.9, podemos ver dos tipos de esquinas redondeadas.

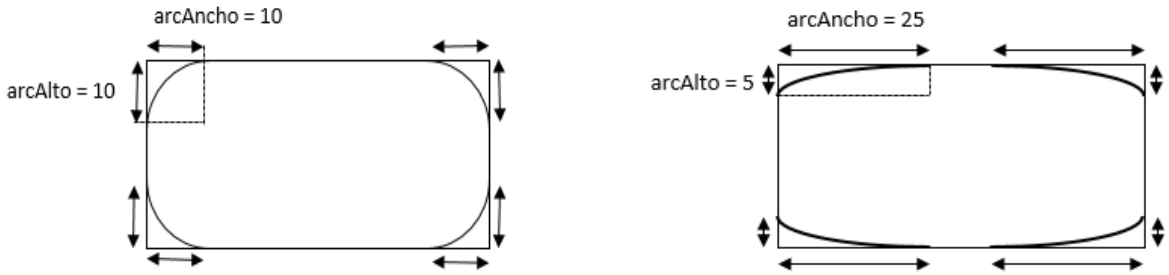



***Figura 4.8.9. Ancho y alto del arco de un rectángulo con esquinas redondeadas.***

Valores mayores hacen más redondeado el rectángulo; valores iguales al ancho y alto del rectángulo producirán un círculo.

**Ejemplo:**

Vamos a dibujar un rectángulo color gris oscuro desde la coordenada (50,50), con dimensiones de 100 de ancho, 30 de alto y con arco de ancho y alto igual a 10.

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.darkGray); //elegimos el color gris oscuro
        g.drawRoundRect(50,50,30,10,10); //dibujamos un rectángulo con esquinas redondeadas
}
```

La ejecución del programa es un rectángulo con las esquinas redondeadas levemente.

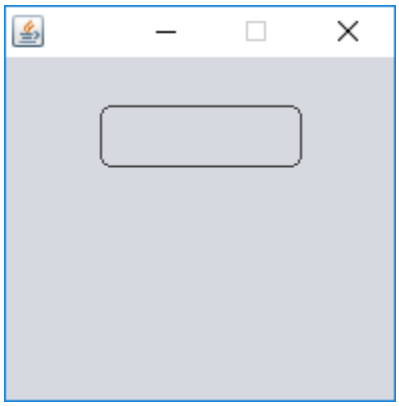


***Figura 4.8.10. Dibujo de un rectángulo redondeado.***

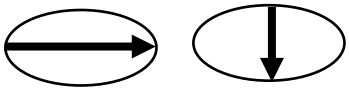
- **drawOval**

Con este método dibujamos el contorno de un óvalo. Es muy semejante a dibujar un rectángulo, ya que este lo contiene, solo que la figura final será una elipse. Su sintaxis es la siguiente:

Sintaxis:

drawOval (int x, int y, int anchura, int altura);

Punto de inicio



**Ejemplo 1:**

Vamos a dibujar un círculo color negro con inicio en las coordenadas (50,50) y diámetro 100, entonces la anchura y la altura serán igual a 100. El siguiente código muestra la forma de hacerlo.

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.black); //elegimos el color negro
        g.drawLOval(50,50,100,100); //dibujamos una circunferencia
}
```

Al ejecutar la aplicación obtenemos el siguiente resultado, una circunferencia de 100 pixeles de diámetro:

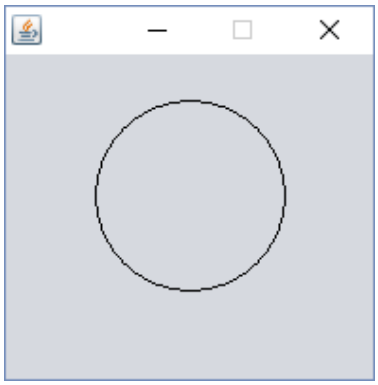


***Figura 4.8.11. Dibujo de una circunferencia.***

Si modificamos la altura de 100 a 20 del método anterior obtendremos un óvalo.

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.black); //elegimos el color negro
        g.drawOval(50,50,100,20); //dibujamos un óvalo
}
```

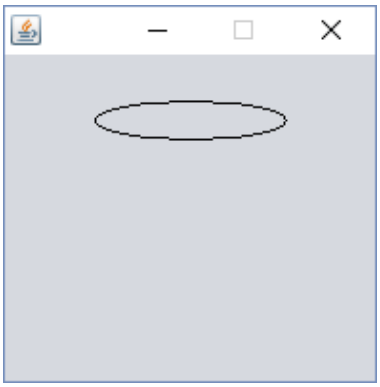


***Figura 4.8.12. Dibujo de un óvalo.***

- **drawPolygon**

Dibuja un polígono cerrado definido por arreglos de coordenadas (x,y) y la cantidad de puntos que forman la figura. El polígono se dibuja como un conjunto de líneas rectas que van del primer punto al segundo, del segundo al tercero y así sucesivamente. A continuación, se muestra el orden y descripción de los parámetros:

Sintaxis:

| drawPolygon | (int[ ] x, | int[ ] y, | int puntos); |
|---|---|---|---|
| | Arreglo de los puntos en x | Arreglo de los puntos en y | Número de puntos |

**Ejemplo:**

Dibujar un triángulo rectángulo con los siguientes datos que muestra la figura.

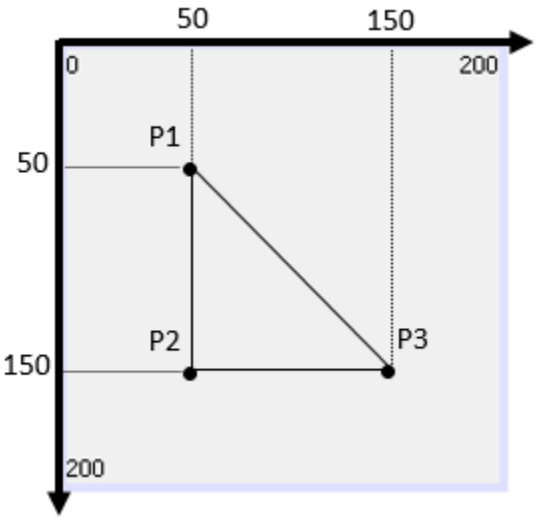



***Figura 4.8.13. Coordenadas de un triángulo rectángulo.***

Las coordenadas que intervienen en la figura son:

| Puntos | x[ ] | y[ ] |
|---|---|---|
| P1(x0,y0) | 50 | 50 |
| P2(x1,y1) | 50 | 150 |
| P3(x2,y2) | 150 | 150 |

***Tabla 4.8.3. Coordenadas de un triángulo rectángulo.***

Definimos los arreglos de coordenadas con los valores de la tabla anterior; un arreglo de tipo entero para los valores de la columna x[ ]={x0,x1,x2} y otro arreglo también de tipo entero para los de la columna y[ ]={y0,y1,y2}. Las sentencias quedarían así:

*int[ ] x={50,50,150};*

*int[ ] y={50,150,150};*

Utilizamos las variables de los arreglos en el método para dibujar un polígono e indicamos los 3 puntos.

*g.drawPolygon(x, y, 3);*

El código final con su ejecución se muestra a continuación:

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.black); //elegimos el color negro
```

```
        int[ ] x={50,50,150}; //declara las coordenadas en x
        int[ ] y={50,150,150}; //declara las coordenadas en y
        g.drawPolygon(x,y,3); //dibujamos un triángulo
}
```

El resultado es un triángulo rectángulo con líneas negras como se muestra en la figura 4.8.14.

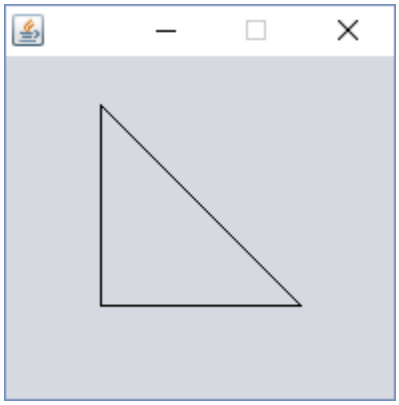


***Figura 4.8.14. Dibujo de un triángulo.***

**Ejemplo 2:**

Ahora vamos a dibujar un polígono semejante a una escalera según lo muestra la siguiente imagen.

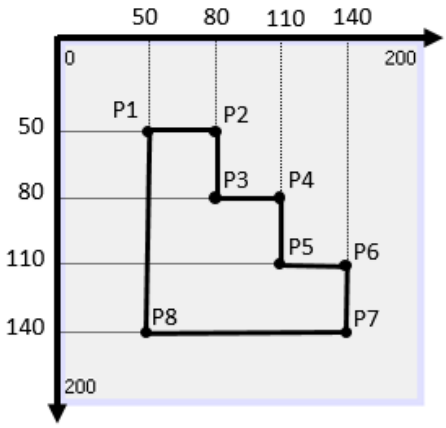



***Figura 4.8.15. Coordenadas de un polígono.***

Notemos que el polígono está formado por 8 puntos y las coordenadas que intervienen en la figura son las siguientes:

| Puntos | x[ ] | y[ ] |
|---|---|---|
| P1(x0,y0) | 50 | 50 |
| P2(x1,y1) | 80 | 50 |
| P3(x2,y2) | 80 | 80 |
| P4(x3,y3) | 110 | 80 |

| | | |
|---|---|---|
| P5(x4,y4) | 110 | 110 |
| P6(x5,y5) | 140 | 110 |
| P7(x6,y6) | 140 | 140 |
| P8(x7,y7) | 50 | 140 |

***Tabla 4.8.4. Coordenadas de un polígono.***

Definimos los arreglos de coordenadas (x,y) con los valores de la tabla anterior: x={x0,x1,x2,x3,x4,x5,x6,x7} y={y0,y1,y2,y3,y4,y5,y6,y7}

```
int[ ] x={50,80,80,110,110,140,140,50};

int[ ] y={50,50,80,80,110,110,140,140};
```

Utilizamos los arreglos como parámetros del método drawPolygon para dibujar un polígono e indicamos los 8 puntos que tiene.

```
g.drawPolygon(x, y, 8);
```

El código final del método paint y su ejecución se muestran a continuación:

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.blue); //elegimos el color azul
        int[ ] x={50,80,80,110,110,140,140,50}; //declara las coordenadas en x
        int[ ] y={50,50,80,80,110,110,140,140}; //declara las coordenadas en y
        g.drawPolygon(x,y,8); //dibuja el polígono de 8 puntos
}
```

Al ejecutar el programa se muestra un polígono con líneas azules y de 8 lados similar a una escalera.

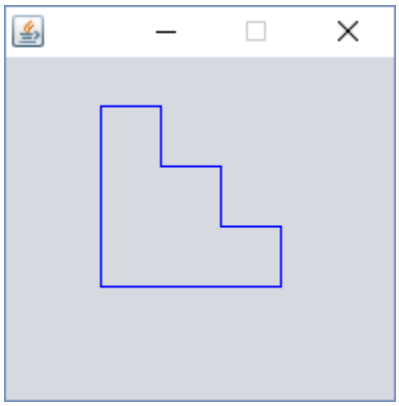


*Figura 4.8.16. Dibujo de una escalera.*

De esta manera podemos dibujar cualquier polígono regular o irregular, basta indicar las ***n*** coordenadas (x,y) y los ***n*** lados.

También podemos hacer el dibujo compuesto, por ejemplo, de un carro observando las figuras geométricas que lo forman, sus coordenadas y sus dimensiones. La siguiente imagen muestra una retícula de 500 X 400 y cada escala es de 50.

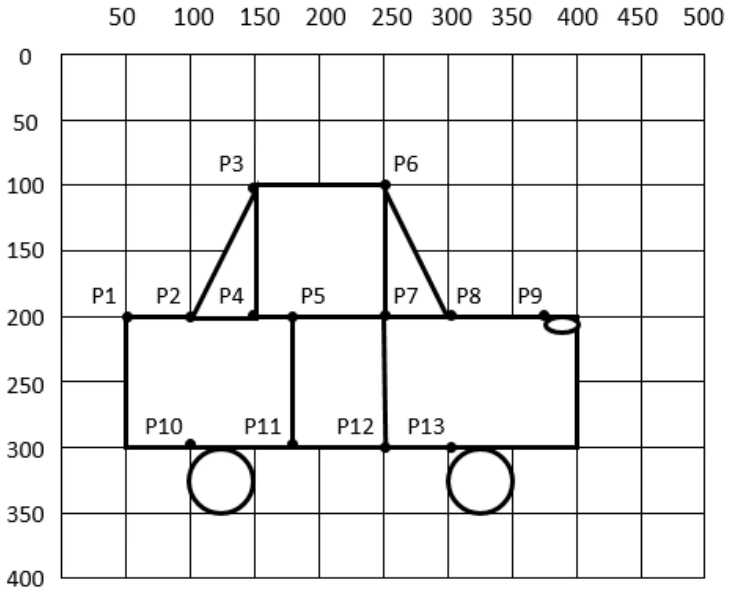



*Figura 4.8.17. Ejemplo de una figura compuesta.*

Observando la figura y la cuadrícula podemos concluir que la imagen está formada por las figuras que se muestran en la siguiente tabla:

| Figura | Punto | x | y | Anchura | Altura |
|---|---|---|---|---|---|
| Rectángulo | P1 | 50 | 200 | 350 | 100 |
| Triángulo 1 | P2<br>P3<br>P4 | 100<br>150<br>150 | 200<br>100<br>200 | | |
| Cuadrado | P3 | 150 | 100 | 100 | 100 |
| Triángulo 2 | P6<br>P7<br>P8 | 250<br>250<br>300 | 100<br>200<br>200 | | |
| Óvalo | P9 | 375 | 200 | 25 | 15 |
| Línea 1 | P5<br>P11 | 175<br>175 | 200<br>300 | | |
| Línea 2 | P7<br>P12 | 250<br>250 | 200<br>300 | | |

| Figura | Punto | x | y | Anchura | Altura |
|---|---|---|---|---|---|
| Círculo 1 | P10 | 100 | 300 | 50 | 50 |
| Círculo 2 | P13 | 300 | 300 | 50 | 50 |

***Tabla 4.8.5. Figuras que forman un carro.***

Vamos a programar una aplicación para mostrar la figura con la clase Graphics. Primero abrimos un JFrame que llamaremos Carro y que tenga las dimensiones de 500 X 400, las mismas que la cuadrícula, y que no sea redimensionable.

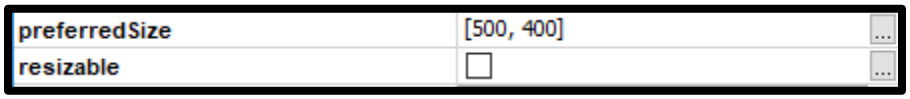



***Figura 4.8.18. Propiedades de tamaño y redimensión del JFrame Carro.***

En seguida importamos las librerías java.awt.Color y java.awt.Graphics después del package y antes de la declaración de la clase.

```
6   package CiberII;
7   import java.awt.Color;
8   import java.awt.Graphics;
9   /**
10   *
11   * @author Gaby López
12   */
13  public class Carro extends javax.swing.JFrame {
```

***Figura 4.8.19. Librerías java.awt.Color y java.awt.Graphics.***

Ahora vamos a escribir el método paint, después del método constructor.

```
13  public class Carro extends javax.swing.JFrame {
14
15      /**
16       * Creates new form Carro
17       */
18      public Carro() {
19          initComponents();
20      }
21
        public void paint (Graphics g)
23      {
24          super.paint(g);
25
26      }
```

***Figura 4.8.20. Método paint para el JFrame Carro.***

El siguiente paso es elegir el color con el que queremos dibujar las figuras.

```
g.setColor(Color.blue); //elegimos el color azul
```

Ahora vamos a insertar la línea que dibuje el rectángulo, colocando las coordenadas de inicio P1(50,200), el ancho de 350 y el alto de 100.

```
g.drawRect(50,200,350,100); //dibuja un rectángulo
```

Para dibujar el triángulo de la izquierda declaramos el arreglo de las coordenadas P2(100,200), P3(150,100) y P4(150,200).

```
int[ ] x1={100,150,150}; //coordenadas x del triángulo
int[ ] y1={200,100,200}; //coordenadas y del triángulo
```

Empleamos las variables x1, y1 y número de lados 3 para dibujar el polígono.

```
g.drawPolygon(x1,y1,3);  //dibuja el triángulo 1
```

Dibujamos el cuadrado indicando en los parámetros las coordenadas P3(150,100) y las dimensiones de 100 x100.

```
g.drawRect(150,100,100,100); //dibuja un cuadrado
```

Para dibujar el triángulo derecho declaramos otros arreglos para las coordenadas P6(250,100), P7(250,200) y P8(300,200).

```
int[ ] x2={250,250,300}; //coordenadas x del triángulo 2
int[ ] y2={100,200,200}; //coordenadas y del triángulo 2
```

Utilizamos estos arreglos para dibujar otro polígono de tres lados.

```
g.drawPolygon(x2,y2,3); //dibuja el triángulo 2
```

Dibujamos el faro del carro con un óvalo de 25 X 15 y las coordenadas P9(375,200).

```
g.drawOval(375, 200, 25, 15); //dibuja un óvalo
```

Marcamos la puerta con dos líneas con sus respectivos puntos P5(175,200), P11(175,300) y P7(250,200), P12(250,300).

```
g.drawLine(175,200,175,300); //dibuja la línea 1
g.drawLine(250,200,250,300); //dibuja la línea 2
```

Finalmente, las llantas son dos círculos de diámetro 50 y coordenadas P10(100,300) y P13(300,300).

```
g.drawOval(100,300,50,50); //dibuja el círculo 1
```

```
g.drawOval(300,300,50,50); //dibuja el círculo 2
```

El código final del método paint y la ejecución se muestran a continuación:

```
public void paint(Graphics g)
{
        super.paint(g);
        g.setColor(Color.blue); //elegimos el color azul
        g.drawRect(50,200,350,100); //dibuja un rectángulo
        int[ ] x1={100,150,150}; //coordenadas x del triángulo 1
        int[ ] y1={200,100,200}; //coordenadas y del triángulo 1
        g.drawPolygon(x1,x2,3); //dibuja el triángulo 1
        g.drawRect(150,100,100,100); //dibuja un cuadrado
        int[ ] x2={250,250,300}; //coordenadas x del triángulo 2
        int[ ] y2={100,200,200}; //coordenadas y del triángulo 2
        g.drawPolygon(x2,y2,3); //dibuja el triángulo 2
        g.drawOval(375,200,25,15); //dibuja un óvalo
        g.drawLine(175,200,175,300); //dibuja la línea 1
        g.drawLine(250,200,250,300); //dibuja la línea 2
        g.drawOval(100,300,50,50); //dibuja la circunferencia 1
        g.drawOval(300,300,50,50); //dibuja la circunferencia 2
}
```

El resultado de ejecutar el programa nos muestra un carro con líneas en color azul.

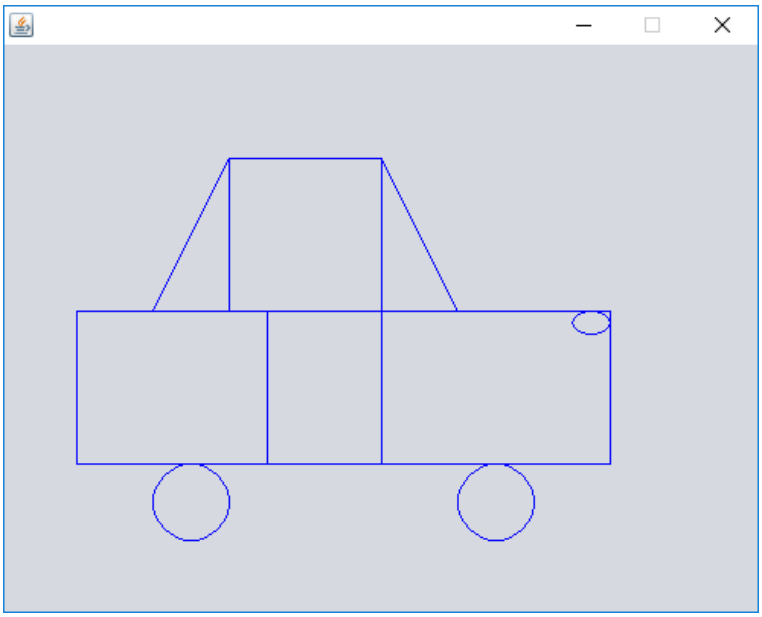


***Figura 4.8.21. Dibujo de un carro.***

Actividad 4.6

**Cuestionario**

1) Es quien proporciona los métodos para dibujar texto y figuras en la pantalla.

a) Graphics b) java.awt c) paint( ) d) El JFrame

2) Este método nos permite acceder al fondo del JFrame para poder dibujar

a) paint( ) b) Graphics c) java.Swing d) Set Color

3) En Java, ¿a qué nos estamos refiriendo cuando hablamos de 'Swing'?

a) Una función utilizada para intercambiar valores.
b) Un framework para dibujar.
c) Una librería para construir interfaces gráficas.
d) Un método para acceder a las funciones gráficas.

4) Sentencia correcta para dibujar una línea recta desde el punto (10,15) hasta el punto (45,25)

a) g.drawLine(int 10, int 15, int 45, int 25);
b) g.drawRect(int 10, int 15, int 45, int 25);
c) g.drawLine(10, 15, 45, 25);
d) g.drawRect(10, 15, 45, 25);

5) ¿Cuál es el resultado obtenido después de ejecutar la siguiente sentencia?

```
g.drawOval(50, 50, 40, 40);
```

a) Dibuja una elipse.
b) Dibuja un rectángulo.
c) Dibuja un rectángulo con esquinas ovaladas.
d) Dibuja una circunferencia.

6) La siguiente sentencia g.drawRect(25, 50, 50, 100);

a) Dibuja una línea recta de (25,50) a (50,100)
b) Dibuja un rectángulo que inicia en (25,50) y termina (50,100)
c) Dibuja un rectángulo de altura = 50 y ancho= 100
d) Dibuja un rectángulo de ancho = 50 y altura= 100

7) Sentencia que elige el color gris para dibujar:

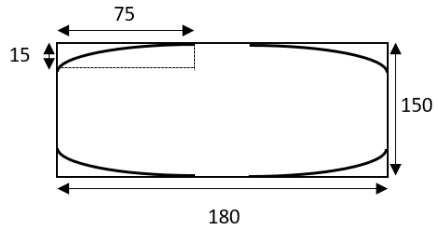
a) g.setColor(gris);
b) g.setColor(gray);
c) g.setColor(Color.gris);
d) g.setColor(Color.gray);

8) Sentencia para dibujar una estrella utilizando los siguientes vectores:

```
int[ ] x={200,250,350,300,350,250,200,150,50,100,50,150};

int[ ] y={50,125,125,200,275,275,350,275,275,200,125,125};
```

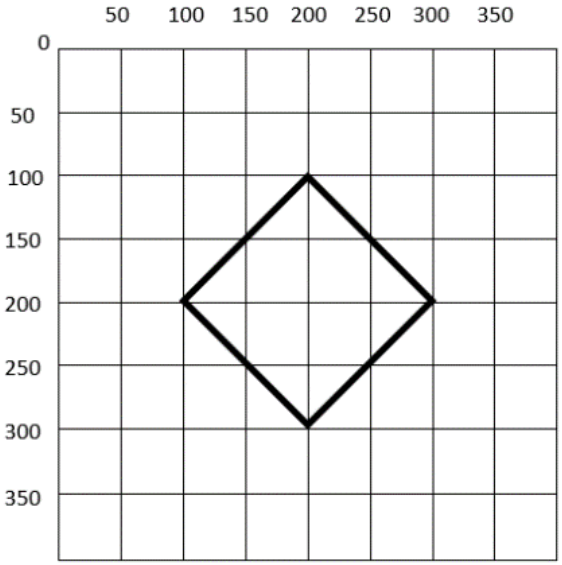
a) g.drawPolygon(x,y,12);
b) g.drawLine(x,y,12,12);
c) g.drawRect(x,y,12,12);
d) g.drawOval(x,y,12,12);

9) Parámetros correctos del método drawRoundRect para dibujar la siguiente figura:




a) g.drawRoundRect(75,15,180,150);
b) g.drawRoundRect(180,150,75,15);
c) g.drawRoundRect(180,75,150,15);
d) g.drawRoundRect(150,180,15,75);

10) Sentencias para dibujar la siguiente figura:




a) int x=100;
   int y=100;
   g.drawRect(x,y,200,200);

b) int[ ] x={200,100,200,300};
   int[ ] y={100,200,300,200};
   g.drawLine(x[0],y[0],x[3],y[3]);

c) int[ ] x={200,100,200,300};
   int[ ] y={100,200,300,200};
   g.drawPolygon(x,y,4);

d) int x=100;
   int y=100;
   g.drawRoundRect(x,y,x,y,x,y);

Los siguientes métodos sirven para rellenar (iluminar) el objeto trazado, para ello se debe colocar antes el método del color que se desea se ilumine el objeto. Cabe mencionar que se ocupa la misma sintaxis que cuando sólo se traza, sólo cambia la palabra *draw* por *fill.*

- **fillRect**

Dibuja un rectángulo relleno especificando una coordenada (x,y) de inicio, y la dimensión de anchura y altura.

Sintaxis:

fillRect (int x, Int y, int anchura, int altura);

**Ejemplo:**
Se crea un nuevo proyecto para ver los trazos rellenos que se van generando con el uso de diferentes métodos. Se genera un nuevo jFrame de tamaño 420 x 380, en donde se colocarán 3 etiquetas y un botón. A continuación, se muestra el valor de la propiedad text de cada elemento:

| Elemento | Propiedad Text |
|---|---|
| jLabel1 | Clase Graphics |
| jLabel2 | Métodos: fillRect, fillRoundRect, fillOval, fillPolygon |
| jButton1 | Salir |

El código del botón salir es:

```
System.exit(0);
```

Ahora, se crea un nuevo método:

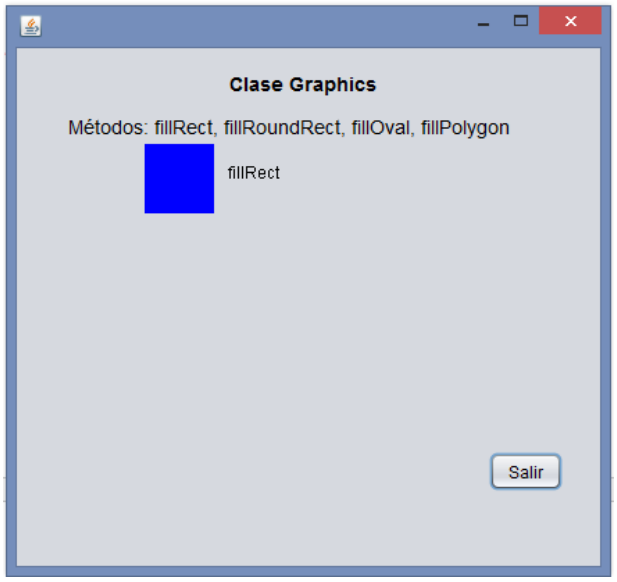
```
public void paint(Graphics g){
    super.paint(g);
```

Nota: No olvides agregar las bibliotecas correspondientes.

Ahora, agregamos el código para visualizar un rectángulo relleno de color azul

```
    //Se asigna color de relleno
    g.setColor(Color.blue);
   // Se indica el método y sus parámetros, punto de inicio del rectángulo y el valor de ancho y alto
    g.fillRect(100, 100, 50, 50);
    //Cambiamos el color a negro
    g.setColor(Color.black);
   //Con la siguiente instrucción se puede mostrar un texto, indicando el valor de la cadena entre “” y en seguida el punto de inicio
    g.drawString("fillRect", 160, 125);
```

Ahora podemos ejecutar el archivo, teniendo como resultado:




- **fillRoundRect**

Este método crea el relleno de un rectángulo de esquinas redondeadas con el color actual del contexto gráfico, sino se especifica un color antes. Su sintaxis solicita seis parámetros: punto de inicio (x, y), ancho y largo (anchura, altura) y los valores de ancho y largo de las esquinas (arcAncho, arcAltura).

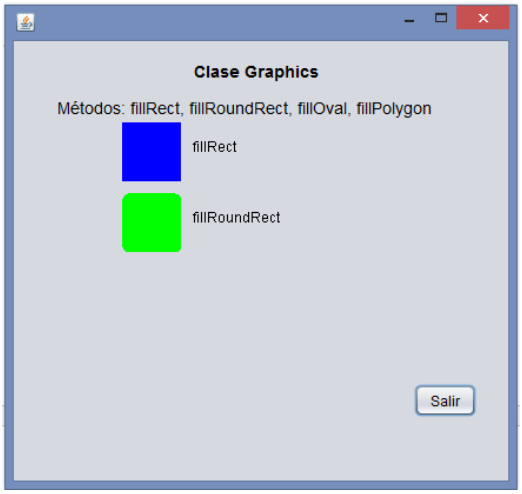
fillRoundRect ( int x, int y, int anchura, int altura, int arcAncho, int arcAlto);

**Ejemplo:**

En el proyecto anterior, vamos agregar este ejemplo, por lo que sólo colocamos las siguientes líneas de código.

```
// Se especifica el color antes
g.setColor(Color.green);
// Se utiliza el método para mostrar un rectángulo con esquinas redondeadas
g.fillRoundRect(100, 160, 50, 50, 10, 10);
// Se especifica el color negro
g.setColor(Color.black);
// Se mostrará el nombre del método en color negro
g.drawString("fillRoundRect", 160, 185);
```

Si ejecutamos el programa se tiene:




- **fillOval**

Este método permite crear un óvalo iluminado (relleno). Los parámetros que utiliza para realizarlo son cuatro, punto de inicio(x,y), ancho y alto; su sintaxis es:

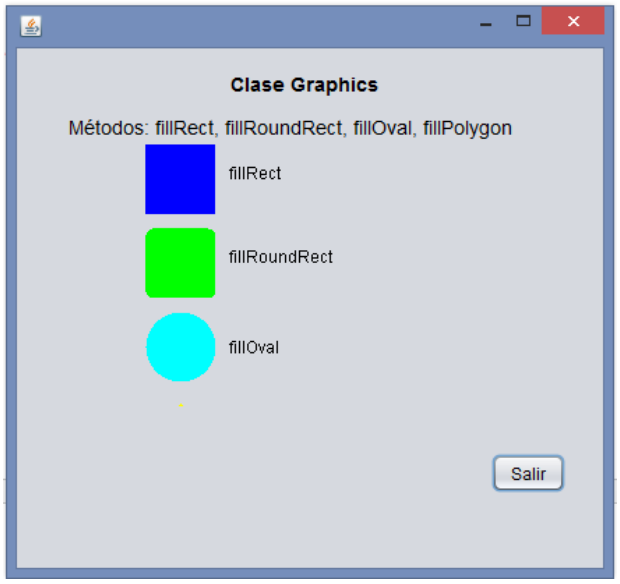
fillOval (int x, int y, int anchura, int altura);

Observa, que la sintaxis anterior sirve para iluminar un óvalo (elipse), si se quiere iluminar un círculo el ancho y alto deben ser iguales.

**Ejemplo:**
Ahora, agregamos en el mismo proyecto las líneas:

```
g.setColor(Color.cyan);
g.fillOval(100, 220, 50, 50);
g.setColor(Color.black);
g.drawString("fillOval", 160, 250);
```

y visualizamos en su ejecución:




- **fillPolygon**

Ilumina un polígono cerrado definido por arreglos de coordenadas (x,y) y la cantidad de puntos que forman la figura. El polígono ilumina el conjunto de líneas rectas que van del primer punto al segundo, del segundo al tercero y así sucesivamente. A continuación, se muestra el orden y descripción de los parámetros:

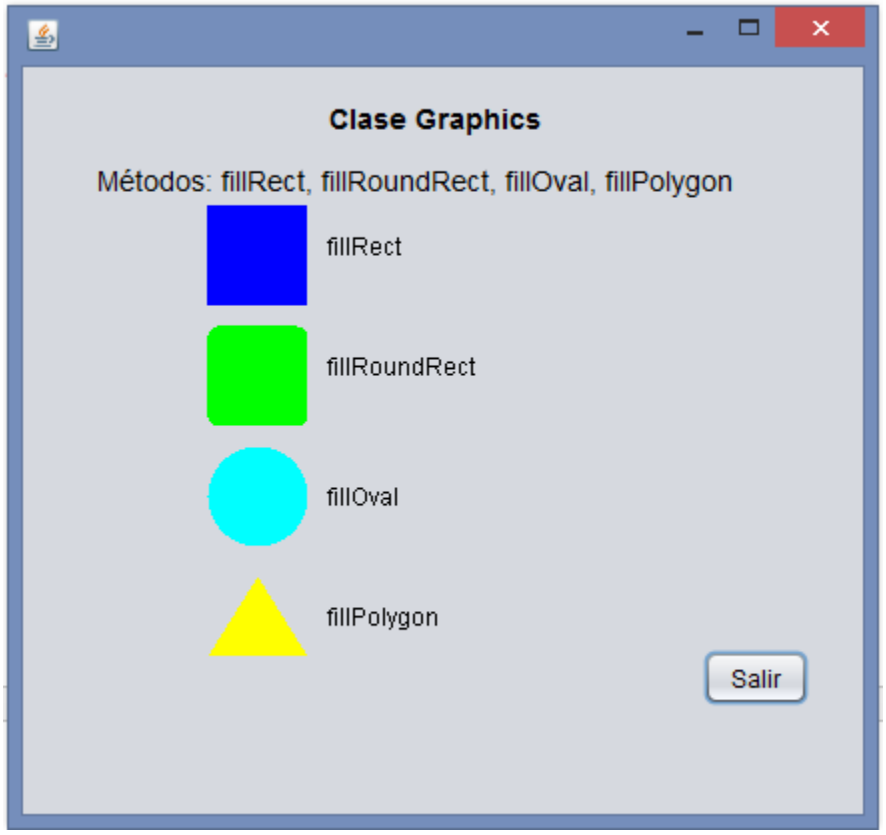
Sintaxis:

fillPolygon ( int[ ] x, int[ ] y, int puntos);

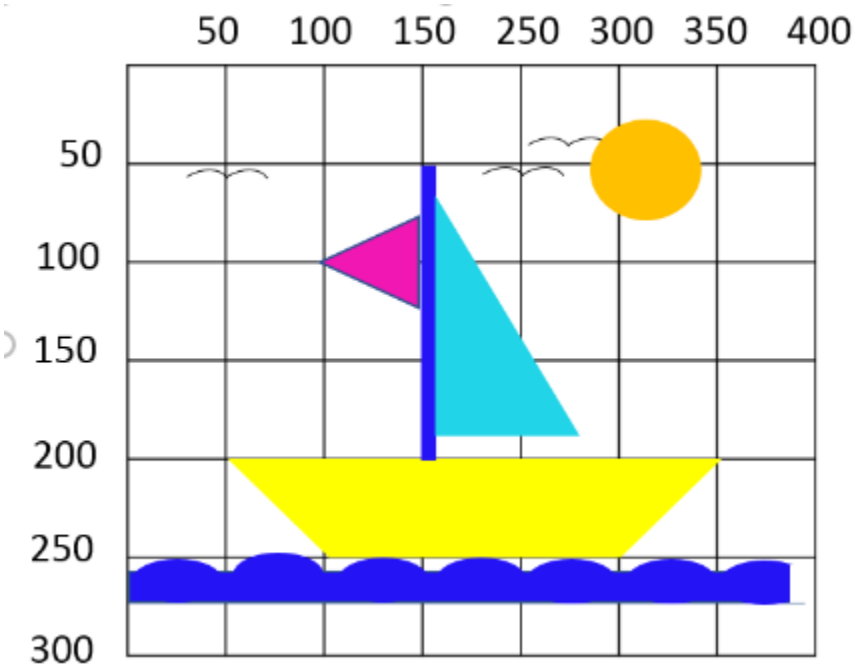
**Ejemplo:**
En el proyecto anterior, también se visualizará un triángulo relleno de color amarillo. Para ello, agregamos las líneas:

```
//se especifica el color del triángulo
 g.setColor(Color.yellow);
// Se especifica las coordenadas X de los tres puntos
 int[ ] vectorX = {125,150,100};
// Se especifican las coordenadas Y de los tres puntos
 int[ ] vectorY = {285,325,325};
 // Se utiliza el método indicando los vectores X y Y y la cantidad de puntos
 g.fillPolygon(vectorX, vectorY, 3);
 //Se especifica el color negro para el siguiente elemento
 g.setColor(Color.black);
 // Se mostrará el nombre del método
 g.drawString("fillPolygon", 160, 310);
```

y su ejecución es:




Ahora vamos a realizar un paisaje utilizando los métodos anteriores, para ello, nos vamos a basar en la siguiente figura:




Para dibujar las gaviotas vamos a trazar un arco desde 0 a 180 grados, para ello utilizaremos el método drawArc( ).

- **drawArc**

Este método tiene seis parámetros, los primeros cuatro corresponden al rectángulo en donde se quiere trazar el arco, esto es, el punto de inicio (x, y), ancho y alto (anchura y altura) y los dos últimos parámetros corresponden al ángulo de inicio y el ángulo de término para el arco que pueden ir desde 0 hasta 180 grados (angIni, angFin). Su sintaxis es:

drawArc(int x, int y,  int anchura, int altura,  int angIni, int angFin);

Cada gaviota esta dibujada por 2 arcos, el rectángulo que las contiene es de 20 x 10 puntos y sólo cambia la posición en X con una diferencia de 20 puntos.

Primero debes identificar los métodos a utilizar

| Parte del paisaje | Método y descripción |
|---|---|
| Base de barquito | fillPolygon – Polígono de cuatro puntos |
| Mastil | fillRect – Un Rectángulo |
| Banderín | fillPolygon  – Polígono de tres puntos |
| Veleta | fillPolygon  – Polígono de tres puntos |

| | |
|---|---|
| Olas | fillOval – Como se repiten se utiliza un ciclo |
| Sol | fillOval – Con ancho y alto iguales |
| Gaviotas | drawArc – dibuja un arco |

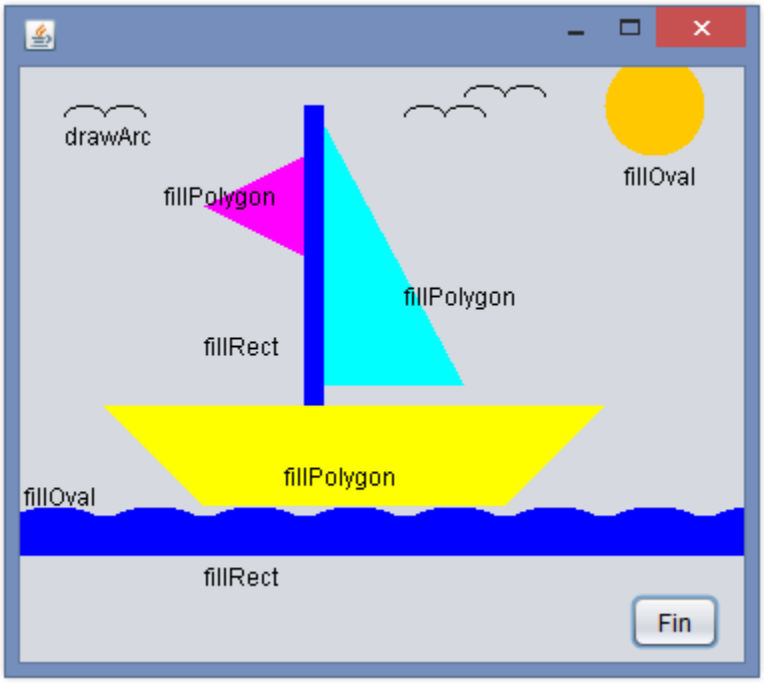
## Código

```
public void paint(Graphics g){
    super.paint(g);

    //Base barquito, es un poligono de cuatro lados
    //Están delimitados por 4 puntos
    //Se define el color amarillo para la base del barquito
    g.setColor(Color.yellow);
    //Definimos coordenadas de los 4 puntos en el eje X
    int[ ] vectorX = {50,300,250,100};
    //Definimos coordenadas de los 4 puntos en el eje Y
    int[ ] vectorY = {200,200,250,250};
    //Traza el poligono con las coordenadas definidas y el número de puntos
    g.fillPolygon(vectorX, vectorY, 4);
    //Se define el color negro para el texto
    g.setColor(Color.black);
    //Se escribe el texto “fillPolygon” en las coordenadas especificadas
    g.drawString("fillPolygon", 140, 240);

    //Mastil
    //Se define el color azul para el mastil
    g.setColor(Color.blue);
    //Se traza un rectángulo en las coordenadas X=150, Y=50, de ancho 10 y alto 150 puntos.
    g.fillRect(150, 50, 10, 150);
    //Se define el color negro para el texto
    g.setColor(Color.black);
    //Se escribe el texto “fillRect” en las coordenadas especificadas
    g.drawString("fillRect", 100, 175);

    //Banderin
    int[ ] vectorX1 = {150,150,100};
    int[ ] vectorY1 = {75,125,100};
    g.setColor(Color.magenta);
    g.fillPolygon(vectorX1, vectorY1, 3);
    //Se define el color negro para el texto
    g.setColor(Color.black);
    //Se escribe el texto “fillPolygon” en las coordenadas especificadas
    g.drawString("fillPolygon", 80, 100);
```

```
//Veleta
int[ ] vectorX2 = {160,160,230};
int[ ] vectorY2 = {60,190,190};
g.setColor(Color.cyan);
g.fillPolygon(vectorX2, vectorY2, 3);
//Se define el color negro para el texto
g.setColor(Color.black);
//Se escribe el texto “fillPolygon” en las coordenadas especificadas
g.drawString("fillPolygon", 200, 150);

//Olas
//Se define el color azul para el agua
g.setColor(Color.blue);
/*Se van a trazar  8 ovalos para las olas utilizando el ciclo for
  Cada ovalo está dentro de un rectángulo con los siguentes parámetros:
  Las coordenadas en X con una separación de 50 puntos iniciando en 0
  El valor para Y no cambia, es 250 puntos
  El ancho 50 y alto 20 puntos*/
for (int i=0; i<8; i++){
    g.fillOval(50*i, 250, 50, 20);
}
//Se define el color negro para el texto
g.setColor(Color.black);
//Se escribe el texto “fillOval” en las coordenadas especificadas
g.drawString("fillOval", 10, 250);

//Agua
//Es otro rectángulo con las siguentes parámetros:
//Se define el color azul para el agua
g.setColor(Color.blue);
//Es una franja del ancho de la ventana: this.getWidth( ),  y de fondo 20 puntos
//Inicia en X=0, Y=255
g.fillRect(0, 255, this.getWidth( ), 20);
//Se define el color negro para el texto
g.setColor(Color.black);
//Se escribe el texto “fillRect” en las coordenadas especificadas
g.drawString("fillRect", 100, 290);

//Sol
//Se define el color naranja para el sol
g.setColor(Color.orange);
//Se traza un círculo dentro de un cuadrado de 50 x 50
//en las coordenadas X=300, Y=25
g.fillOval(300, 25, 50, 50);
```

```
    //Se define el color negro para el texto
    g.setColor(Color.black);
    //Se escribe el texto “fillOval” en las coordenadas especificadas
    g.drawString("fillOval", 310, 90);

    //Gaviotas
    //Se define el color negro para las gaviotas
    g.setColor(Color.black);
    /*Para las gaviotas vamos a trazar un arco desde 0 a 180 grados que son los
      últimos valores del método drawArc( ), los primeros 4 parámetros son los mismos
      para el trazo de un rectángulo, el origen: X,Y y las dimensiones: ancho y alto
      cada gaviota esta dibujada por 2 arcos, el rectángulo que las contiene es de
      20 x 10 puntos y solo cambia la posición en X con una diferencia de 20 puntos.
      drawArc(int x, int y,   int anchura, int altura,  int gradoIni, int gradoFin); */
    //g.drawArc
    //primera gaviota
    g.drawArc(30, 50, 20, 10, 0, 180);
    g.drawArc(50, 50, 20, 10, 0, 180);
    //segunda gaviota
    g.drawArc(230, 40, 20, 10, 0, 180);
    g.drawArc(250, 40, 20, 10, 0, 180);
    //tercera gaviota
    g.drawArc(200, 50, 20, 10, 0, 180);
    g.drawArc(220, 50, 20, 10, 0, 180);
    //Se define el color negro para el texto
    g.setColor(Color.black);
    //Se escribe el texto “drawArc” en las coordenadas especificadas
    g.drawString("drawArc", 30, 70);
  }
}
```

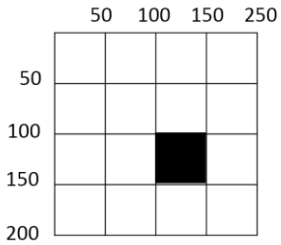
**Ejecución:**




Actividad 4.7

**Cuestionario.**

1. Para dibujar el siguiente rectángulo utilizas el código.



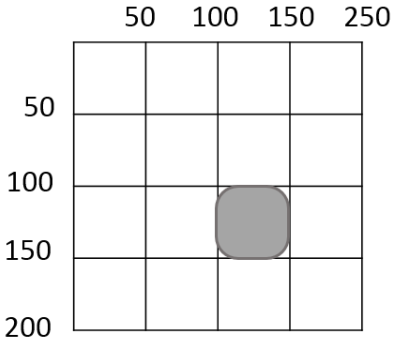

a) g.setColor(Color. black);
g.fillRect(100, 100, 50, 50);

b) g.setColor(Color. black);
g.fillRoundRect(100, 160, 50, 50, 10, 10);

c) g.setColor(Color. black);
g.drawRect(50,50,30,100);

d) g.setColor(Color. black);
g.drawRoundRect(50,50,30,10,10);

2. Para dibujar el siguiente rectángulo utilizas el código.




a) g.setColor(Color.gray);
g.fillRect(100, 100, 50, 50);

b) g.setColor(Color.gray);
g.fillRoundRect(100, 160, 50, 50, 10, 10);

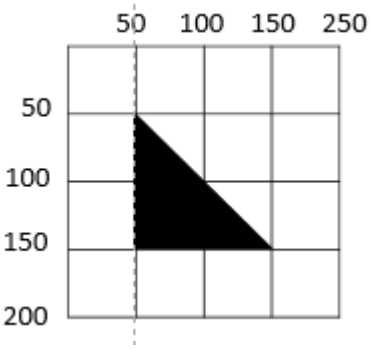
c) g.setColor(Color.gray);
g.drawRect(50,50,30,100);

d) g.setColor(Color.gray);
g.drawRoundRect(50,50,30,10,10);

3. Sí dibujamos un óvalo con el siguiente código g.fillOval(100, 220, 50, 45), ¿cuál es la anchura del óvalo?

a) 100 b) 220 c) 50 d) 45

4. Elige el código para realizar la siguiente figura.



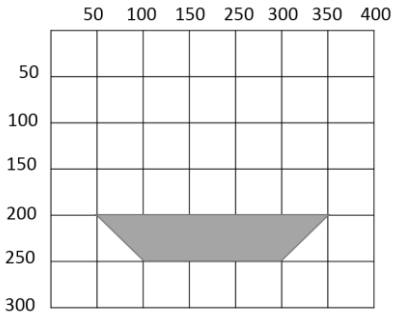

a) g.setColor(Color. black);
int[ ] x={50,50,150};
int[ ] y={50,150,150};
g.fillPolygon(x,y,3);

b) g.setColor(Color. black);
int[ ] x={50,150,150};
int[ ] y={50,150,150};
g.fillPolygon(x,y,4);

c) g.setColor(Color.black);
int[ ] x={50,50,150};
int[ ] y={50,150,150};
g.drawPolygon(x,y,3);

d) g.setColor(Color.black);
int[ ] x={50,50,150};
int[ ] y={50,50,150};
g.fillPolygon(x,y,3);

5. ¿Cuál es el código para realizar la siguiente figura?



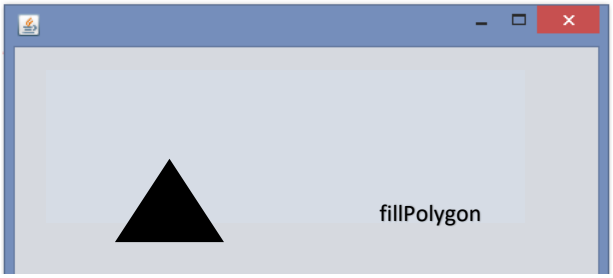

a)
```
g.setColor(Color. gray);
int[ ] vectorX = {50,350,300,100};
int[ ] vectorY = {200,200,250,250};
g.fillPolygon(vectorX, vectorY, 4);
```

b)
```
g.setColor(Color. gray);
int[ ] vectorX = {200,300,250,100};
int[ ] vectorY = {50,200,250,250};
g.fillPolygon(vectorX, vectorY, 4);
```

c)
```
g.setColor(Color. gray);
int[ ] vectorX = {50,200,250,100};
int[ ] vectorY = {200,300,250,250};
g.fillPolygon(vectorX, vectorY, 4);
```

d)
```
g.setColor(Color. gray);
int[ ] vectorX = {50,300,250,100};
int[ ] vectorY = {200,200,250,250};
g.fillPolygon(vectorX, vectorY, 4);
```

6. Ordena el programa para visualizar el siguiente triangulo.




I. g.setColor(Color.black);

III. int[ ] vectorX = {125,150,100};

IV. int[ ] vectorY = {285,325,325};

V. g.fillPolygon(vectorX, vectorY, 3);

II. g.drawString("fillPolygon", 160, 310);

a) I-II-III- V- IV  b) IV-III-V- II- I  c) I- IV-III-V-II  d) I-III-IV-V-II

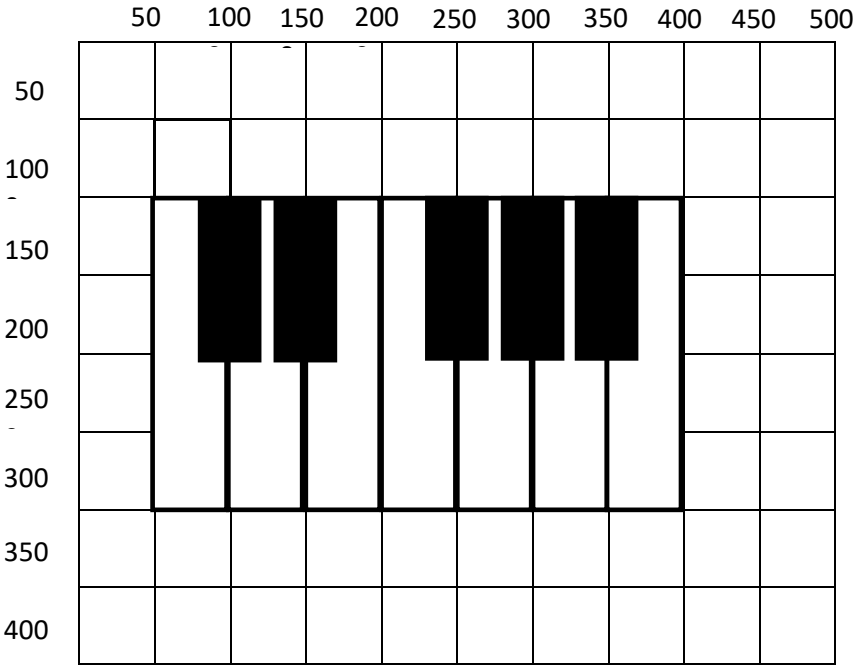
## 4.9 Ejercicio guiado.

**Propósito**:

Que el alumno pueda utilizar clase Graphics para dibujar el teclado de un piano.

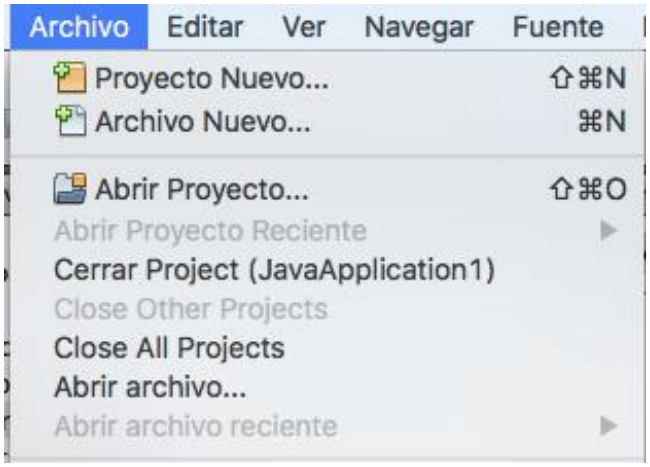
**DESARROLLO:**

Vamos a dibujar el teclado de un piano dentro de un formulario, este quedará como se muestra en la siguiente figura:

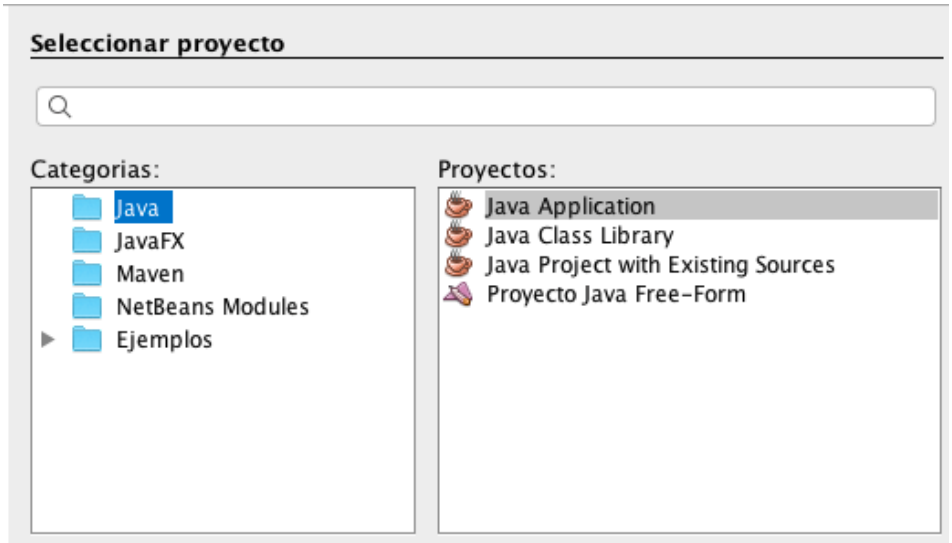



Para ello vamos a desarrollar los siguientes pasos:

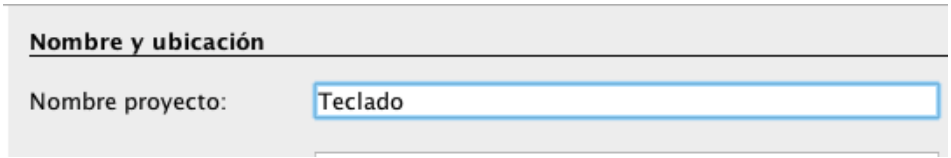
1) Crea un nuevo formulario en Netbeans mediante: Archivo -> Nuevo Proyecto:



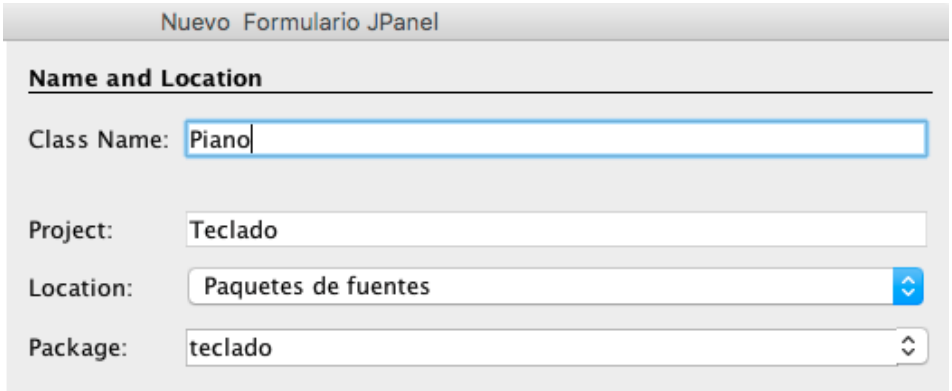
2) El tipo de Proyecto será Java Application.



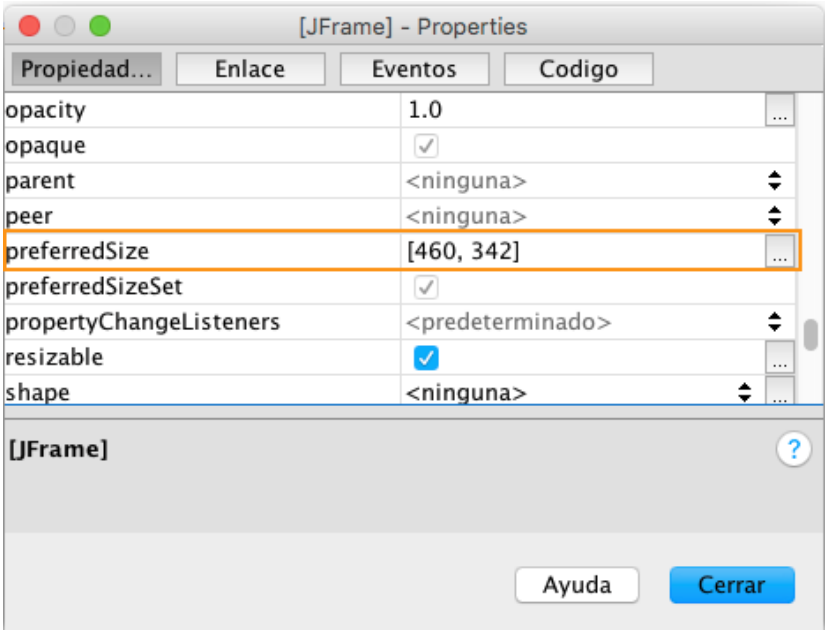

3) Al Proyecto le llamaremos Teclado




4) Una vez creado el Proyecto, vamos a crear un Formulario Jframe al que llamaremos Piano.




5) En las propiedades del Jframe vamos a asignarle un tamaño de 460 x 342 pixeles




6) Debemos abrir el código del Jframe para comenzar a dibujar nuestro teclado, por tanto, primero debemos importar las siguientes clases al principio del código:

```
import java.awt.Color;
import java.awt.Graphics;
```

7) Posteriormente iremos donde termina el constructor del Jframe, es decir donde se indica en la siguiente imagen:

8) Aquí utilizaremos la clase Graphics y crearemos un objeto llamado **teclas** mediante el siguiente código:

```
public void paint (Graphics teclas){
    super.paint(teclas);
}
```

9) Para dibujar la primer tecla haremos uso de los métodos: **setColor( ), fillRect( ), y drawRect( )**.

**SetColor ( )** nos sirve para asignar un color y las siguientes figuras que se dibujen tendrán ese mismo color. Dado que primero vamos a dibujar un rectángulo cuyo relleno será de color blanco para representar una tecla, entonces escribimos lo siguiente:

```
teclas.setColor(Color.white);
```

Posteriormente usamos el método **fillRect( )**; que requiere de cuatro parámetros: la coordenada inicial en x, la coordenada inicial en y, el ancho y la altura en pixeles. Para este ejemplo nuestra coordenada inicial será (50,100), el ancho de 50 pixeles y el largo de 200. Por tanto, nuestro código será el siguiente:

```
teclas.fillRect(50, 100, 50, 200);
```

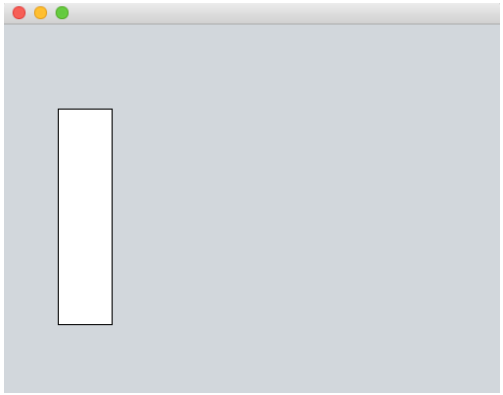
Para dibujar el contorno de la teclas requerimos utilizar el color negro y usar el método **drawRect( )**, el cual es similar a **fillRect( )** pero solo dibuja el contorno, por tanto, nuestro código quedaría como:

```
teclas.setColor(Color.black);
teclas.drawRect(50, 100, 50, 200);
```

El código completo seria:

```
public void paint (Graphics teclas){
    //Se crea un objeto de la clase Graphics llamado teclas
    super.paint(teclas);
    teclas.setColor(Color.white);
    teclas.fillRect(50, 100, 50, 200);
    teclas.setColor(Color.black);
    teclas.drawRect(50, 100, 50, 200);
}
```

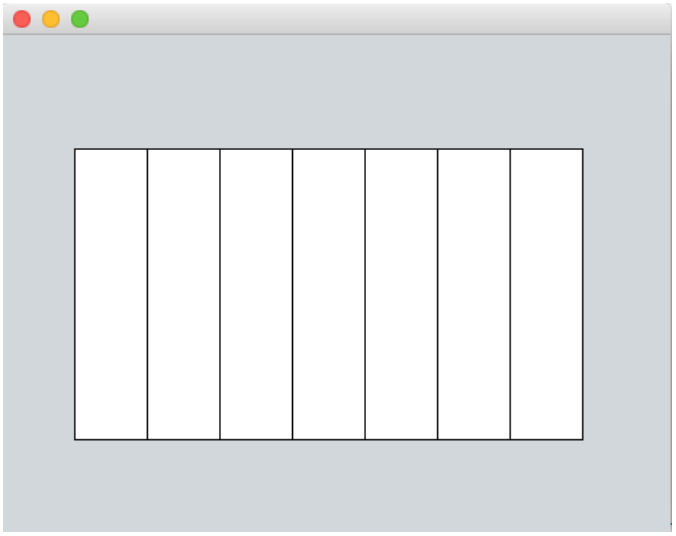
Lo cual no daría lo siguiente imagen una vez que ejecutemos el programa:



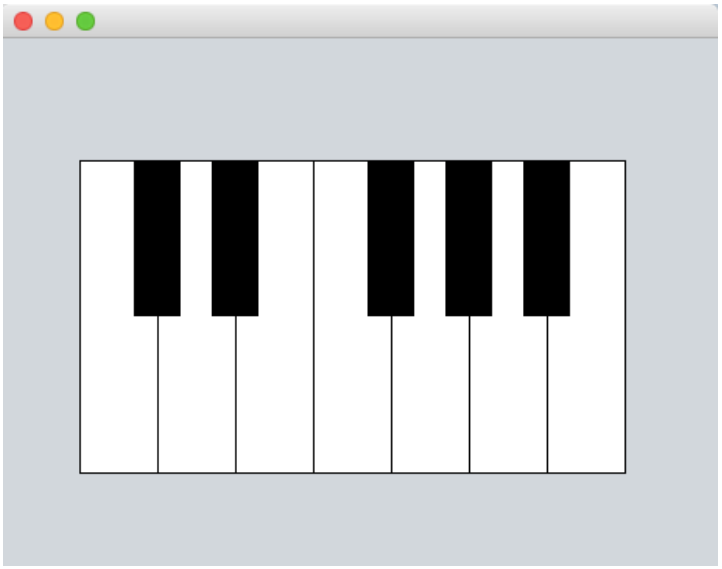
10) Podemos utilizar un ciclo For para dibujar las 7 teclas de nuestro Piano en una sola ejecución, observa el siguiente código:

```
    public void paint (Graphics teclas){
    //Se crea un objeto de la clase Graphics llamado teclas
     super.paint(teclas);
    //La coordenada inicial del rectángulo será 50
    int x1= 50;
    //Para un valor inicial de 1 hasta 7, con incrementos de uno en uno se hará lo siguiente:
    for (int i=1; i<=7;i++ ) {
   //Dibuja el rectángulo de color blanco con contorno de color negro
    teclas.setColor(Color.white);
    teclas.fillRect(x1, 100, 50, 200);
    teclas.setColor(Color.black);
    teclas.drawRect(x1, 100, 50, 200);
    //Incrementa la coordenada X1 en 50, de tal forma que el siguiente rectángulo inicie
   //50 pixeles a la derecha del primero.
    x1=x1+50;
    }
}
```

La salida que corresponde:



Como actividad, se te propone que coloques los rectángulos negros que forman parte de las teclas “sostenido” de cada una de las notas musicales con las que cuenta el piano, una vez que lo hagas, al ejecutar tu programa debes tener a una imagen como la siguiente:



**RESPUESTA**

**CÓDIGO COMPLETO:**

```
public void paint (Graphics teclas ){
    //Se crea un objeto de la clase Graphics llamado teclas
     super.paint(teclas);
    //La coordenada inicial del rectángulo será 50
    int x1= 50;
    //Para un valor inicial de 1 hasta 7, con incrementos de uno en uno se hará lo siguiente:
    for (int i=1; i<=7;i++ ) {
   //Dibuja el rectangulo de color blanco con contorno de color negro
```

```
  teclas.setColor(Color.white);
  teclas.fillRect(x1, 100, 50, 200);
  teclas.setColor(Color.black);
  teclas.drawRect(x1, 100, 50, 200);
  //Incrementa la coordenada X1 en 50, de tal forma que el siguiente rectangulo inicie
 //50 pixeles a la derecha del primero.
  x1=x1+50;
  }

  //Dibuja la primer tecla iniciando en x=85, y=100, ancho de 30 y altura de 100 pixeles
  int x2= 85;
  teclas.setColor(Color.black);
  teclas.fillRect(x2, 100, 30, 100);

   //Dibuja la segunda tecla en x=x2+50, y=100, ancho de 30 y altura de 100 pixeles
  x2= x2+50;
  teclas.setColor(Color.black);
  teclas.fillRect(x2, 100, 30, 100);

   //Dibuja la tercera tecla en x=x2+100, y=100, ancho de 30 y altura de 100 pixeles
  //Se aumentan 100 ya que debemos saltarnos una posición
   x2= x2+100;
  teclas.setColor(Color.black);
  teclas.fillRect(x2, 100, 30, 100);

   //Dibuja la cuarta tecla en x=x2+50, y=100, ancho de 30 y altura de 100 pixeles
  x2= x2+50;
  teclas.setColor(Color.black);
  teclas.fillRect(x2, 100, 30, 100);

   //Dibuja la última tecla en x=x2+50, y=100, ancho de 30 y altura de 100 pixeles
  x2= x2+50;
  teclas.setColor(Color.black);
  teclas.fillRect(x2, 100, 30, 100);
}
```

## Solución de las actividades

## Unidad 4

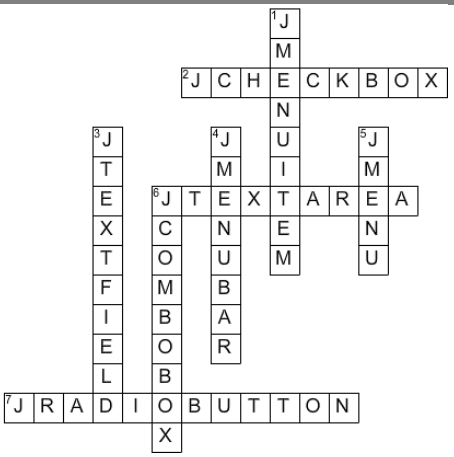
### Actividad 4.1

2,5,1,3,6,10,7,4,8,9

### Actividad 4.2

| | | | | |
|---|---|---|---|---|
| 1. a | 2. c | 3. c | 4. b | 5. d |
| 6. b | 7. c | 8. a | 9. c | 10. c |

### Actividad 4.4




### Actividad 4.5

1,4,3,2,5,8,7,6

### Actividad 4.6

| | | | | |
|---|---|---|---|---|
| 1. a | 2. c | 3. c | 4. b | 5. d |
| 6. b | 7. c | 8. a | 9. c | 10. c |

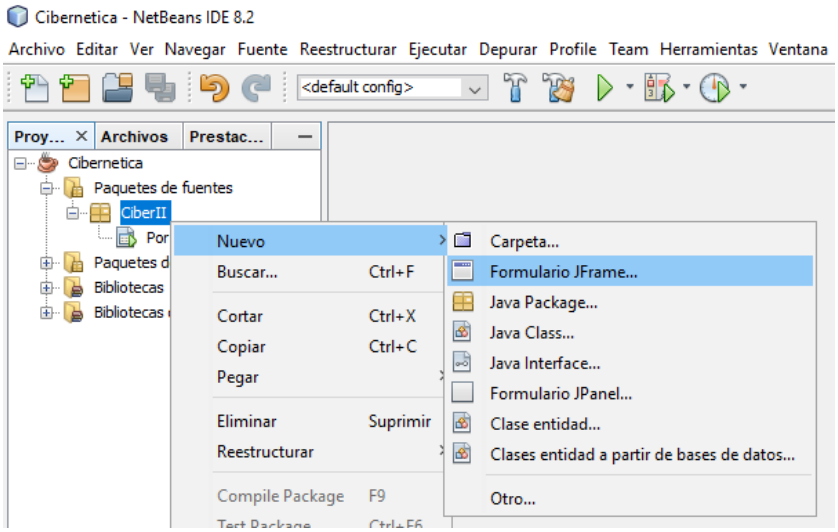
### Actividad 4.7

| | | | | |
|---|---|---|---|---|
| 1. a | 2. b | 3. c | 4. a | 5. d |
| 6. d | | | | |

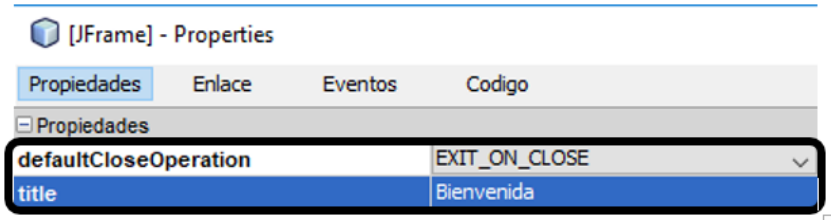
### Actividad 4.3

**Solución.**

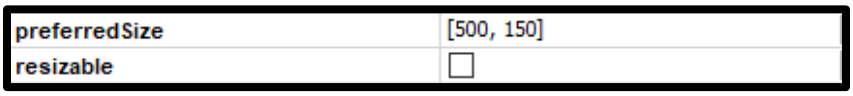
1. Elaborar un nuevo JFrame en el paquete CiberII del proyecto Cibernética y lo nombramos Bienvenida como se muestra en seguida:




2. Abre las propiedades de la ventana y en la opción title cambias el título como se muestra en la imagen.



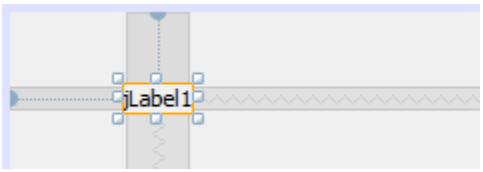

3. Busca la propiedad de preferredSize para cambiar el tamaño y quita la selección de resizable para que no sea redimensionable.

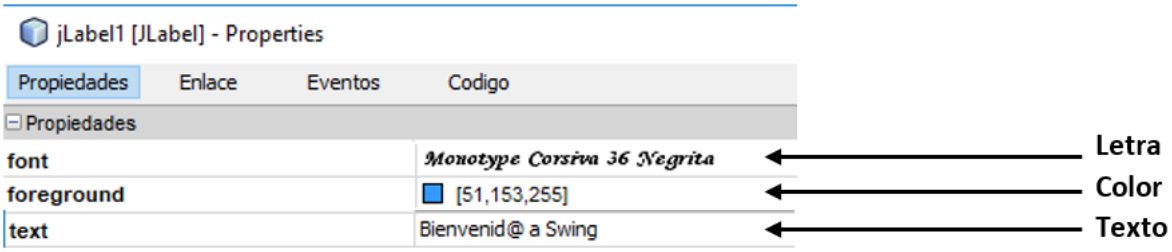



4. Selecciona la pestaña de source y agrega las sentencias indicadas en los recuadros de la siguiente imagen para cambiar el color de fondo:

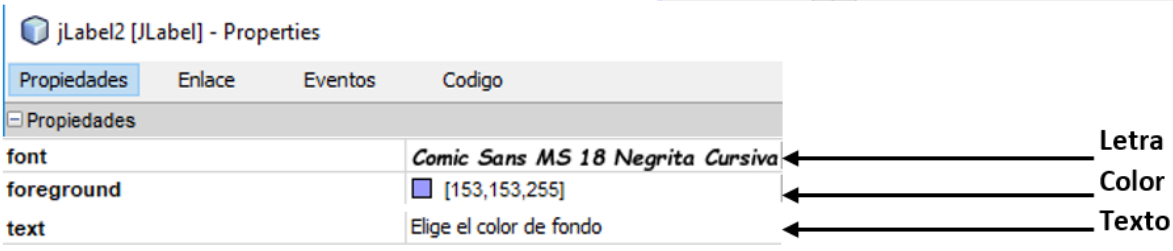
```
import java.awt.Color;

/**
 *
 * @author Gaby López
 */
public class Bienvenida extends javax.swing.JFrame {

    /**
     * Creates new form Bienvenida
     */
    public Bienvenida() {
        initComponents();
        this.getContentPane().setBackground(Color.white);
    }
```

5. Inserta una etiqueta y modifica las propiedades de letra, su color y texto que va a mostrar.

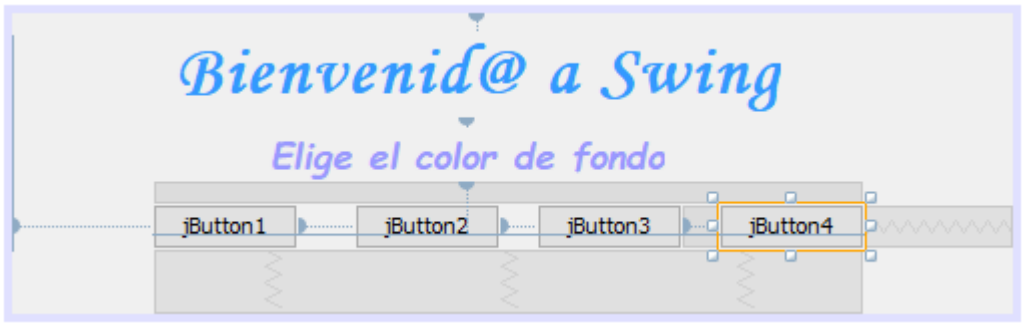


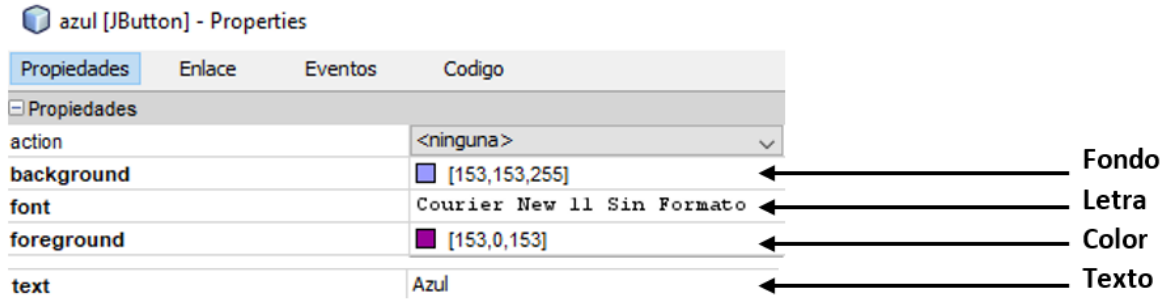

6. Inserta una segunda etiqueta y modifica las propiedades indicadas a continuación:

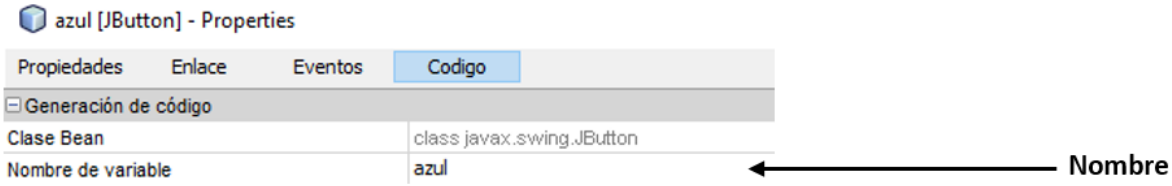
7. Inserta los cuatro botones.




8. Cambia las propiedades de los botones: color de fondo, letra, color de letra y el mensaje que van a mostrar.




9. En la pestaña de Código cambia el nombre de la variable con el nombre del color escrito.




10. Para programar la acción de cada botón, da doble clic sobre él y escribe el siguiente código según el color.

```
//código del botón para cambiar el color del fondo a azul
Private void azulActionPerformed(java.awt.event.ActionEvent evt){
        this.getContentPane().setBackground(Color.blue);
}
//código del botón para cambiar el color del fondo a amarillo
Private void amarilloActionPerformed(java.awt.event.ActionEvent evt){
        this.getContentPane().setBackground(Color.yellow);
}
//código del botón para cambiar el color del fondo a rojo
Private void rojoActionPerformed(java.awt.event.ActionEvent evt){
```

```
        this.getContentPane().setBackground(Color.red);
}
//código del botón para cambiar el color del fondo a verde
Private void verdeActionPerformed(java.awt.event.ActionEvent evt){
        this.getContentPane().setBackground(Color.green);
}
```

11. Después de modificar las propiedades y el código de acción de cada botón, ejecuta el programa pulsando F6 y Mayus. Se mostrará una ventana con fondo blanco y el siguiente diseño.


Al pulsar el botón que dice Azul el fondo se cambia a color azul.


Al oprimir el botón con el mensaje Amarillo el fondo cambia a ese color.


Presionando el botón Rojo cambiamos el fondo a color rojo.


La última opción es pulsar el botón Verde y cambiar el fondo a verde.


## Referencias

### Unidad 4

Ceballos, F. (2015). *Java.* Interfaces *gráficas y aplicaciones para internet*. México. RA-MA Editorial.

Codejavu.blogspot.com. (2013). *Componentes Java Swing*. [en línea] Disponible en: http://codejavu.*blogspot*.com/2013/09/componentes-java-swing.html [Consultado el 4 abril de 2019].

Dean, J. (2009). *Introducción a la* programación *con Java*. México. Mc. Graw-Hill.

Deitel, P. (2008). *Cómo* programar *en Java*. México. Pearson Educación.

Gigena, M. (2012). *La clase Graphics.* [en línea] Disponible en http://labojava.blogspot.com/2012/05/la-clase-graphics.html [Consultado el 20 de mayo de 2019].

Hipertexto.info. (2019). *La interfaz gráfica*. [en línea] Disponible en: http://www.hipertexto.info/documentos/interfaz.htm [Consultado el 9 de abril de 2019].

Java México User Group. (2019). *Código JAVA: Cambiar el color de fondo de un JFrame y JPanel*. [en línea] Disponible en https://www.javamexico.org/blogs/edbast/codigo_java_cambiar_el_color_de_fondo_de_un_jframe_y_jpanel [Consultado el 3 de abril de 2019]

NetBeans. (2019). Disponible en: https://es.wikipedia.org/wiki/NetBeans [Consultado el 2 de abril de 2019].

Moisset, D. (2019). *Java Ya*. [en línea] Disponible en http://www.tutorialesprogramacionya.com/javaya/detalleconcepto.php?codigo=130&punto=&inicio= [Consultado el 20 de mayo de 2019].

Oracle America, (2018). *Class Graphics*. [en línea] Disponible en https://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html [Consultado el 20 de mayo de 2019].

Tecnolog a, D. y usuario, I. (2019). *Definición de GUI (Interfaz Gráfica de Usuario)*. [en línea] Alegsa.com.ar. Disponible en: http://www.alegsa.com.ar/Dic/gui.php [Consultado el 9 de abril de 2019].