GUÍA DE EXTRAORDINARIO

Cibernética y Computación II

Elaboraron: Claudia del Socorro Rodríguez Saldivar. (Coordinadora de la guía)
Héctor Gabriel Rivera Vargas.
Ramón Rodríguez Jiménez.
Salvador Gómez Moya.

Contenido

Unidad 1. Lenguaje de programación orientada a objetos con Java

Conceptos básicos de la programación orientada a objetos (POO)

El paradigma de la programación orientada a objetos consiste en la representación de la realidad. En éste se manejan algunos conceptos básicos como son: clases, objetos, atributos, métodos y se caracteriza por emplear la abstracción de datos, herencia, encapsulamiento y polimorfismo.

Actividad 1.1. Realiza la siguiente tabla con las características principales de la POO

CONCEPTO	DEFINICIÓN
Abstracción	
Encapsulamiento	
Herencia	
Polimorfismo	

Organización general de un programa en Java

Para escribir un programa en Java lo primero que tienes que hacer es crear una clase, utilizando un entorno de programación IDE tal como: NetBeans, Eclipse, BlueJ etc.

A continuación, se muestra un ejemplo de una clase con la estructura básica.

```
/* Éste es un programa simple de Java.
* Para revisar la estructura básica del mismo.        (1)
*/
/**
 * Este método se encarga de iniciar la ejecución del programar        (1)
 * Éste es el método principal del proyecto
*/
import java.util.Scanner; (2)
 class Ejemplo {
// Un programa de Java empieza con una llamada a main(). (1)

int numero;          (3)
string nombre;
public static void main(String args[]) {
System.out.println("Primer programa de Java.");
}
} (7)
```

Comentarios (1)

Son una herramienta que sirve para apoyar la documentación de los programas que desarrollamos y así facilitar su posterior comprensión por parte de alguna otra persona que comprenda algo de Java.

Java soporta tres estilos de comentarios:

- **Comentarios en una sola línea.** Pueden ser colocados en cualquier parte de nuestro código en Java y comienzan por una doble diagonal **//**

- **Comentarios de múltiples líneas.** Permiten comentar varias líneas del código. Estos comentarios inician con /* y deben terminar con */, deben tener un comienzo y un final. Todo lo que se encuentre entre estos dos símbolos de comentario es ignorado por el compilador.

- **Comentarios de documentación (Javadoc).** Son de especial utilidad al momento de documentar no sólo el código fuente, sino el proyecto como tal. Por medio de herramientas externas, podremos generar de forma automática la documentación de un proyecto Java, a partir de estos comentarios de documentación o Javadocs.

Bibliotecas (2)

Una biblioteca en Java se puede entender como un conjunto de clases, que poseen una serie de métodos y atributos. Estas bibliotecas facilitan muchas operaciones. De una forma más completa, las librerías en Java nos permiten reutilizar código, es decir que podemos hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar nosotros mismos esas funcionalidades.

Identificadores (3)

Son el nombre que le asignamos a nuestras variables, métodos, clases, constantes, paquetes, objetos y demás elementos que conforman un programa Java.

Para poder definir un identificador válido, debemos de seguir las siguientes reglas:

Debe comenzar con una letra, un símbolo de subrayado (_) o un símbolo de dólar($). Los siguientes caracteres pueden ser letras, dígitos, símbolos de subrayado o símbolos de dólar.

Las mayúsculas y minúsculas se consideran diferentes.

Son válidos los siguientes identificadores.

edadMaxima, edadmaxima, monto_total, sueldo_bruto, suelbru, $ganancia, nota1, nota2, importeCompra

Identificadores No válidos.

1cuenta (no puede comenzar con número)
monto total (no puede haber espacios en blanco)
premio# (contiene carácter invalido #)
final (no puede ser una palabra reservada del lenguaje)
continue (no puede ser una palabra reservada del lenguaje)
Note que los identificadores edadMaxima y edadmaxima no son iguales dado que M mayúscula no es lo mismo que n minúscula.

Actividad 1.2. Indica cuáles de los siguientes identificadores son válidos en Java. Si el identificador no es válido explica por qué no lo es.

1) registro1	2) 1registro	3) archivo_3	4) while
5) $impuesto	6) año	7) primer apellido	8) primer_apellido
9) primer-apellido	10) primerApellido	11) Tom's	12) C3PO
13) 123#	14) PesoMáximo	15) %descuento	16) Weight
17) $$precioMínimo	18) _$Único	19) tamaño_màximo	20) peso.maximo
21) Precio___	22) matrícula?	23) cuántoVale	24) high
25) barça	26) piragüista	27) B_011	28) X012AB
29) 70libro	30) nombre&apellido	31) 0X1A	32) else

Palabras reservadas

Las palabras reservadas, como su nombre lo indica, son palabras las cuales el lenguaje de programación ya ha reservado para realizar ciertas tareas. En Java actualmente existen 50 palabras reservadas.

Algo importante a tener en cuenta es que las palabras reservadas no pueden ser utilizadas como nombres de variables, clases o métodos.

• abstract	• continue	• for	• new	• switch
• assert	• default	• goto	• package	• synchronized
• boolean	• do	• if	• private	• this
• break	• double	• implements	• protected	• throw
• 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

Sentencias

Las sentencias son, a grandes rasgos, equivalentes a las oraciones del lenguaje natural. Una sentencia es una orden que se le da al programa para realizar una tarea específica, esta puede ser: mostrar un mensaje en la pantalla, declarar una variable (para reservar espacio en memoria), inicializarla, llamar a una función, etc. Las sentencias acaban con punto y coma **(;)**. Este carácter separa una sentencia de la siguiente. Normalmente, las sentencias se ponen unas debajo de otras, aunque sentencias cortas pueden colocarse en una misma línea. He aquí algunos ejemplos de sentencias

```
int i=1;
import java.awt.*;
System.out.println("El primer programa");
rect.mover(10, 20);
```

Tipos de datos primitivos

Se cuenta con 8 tipos de datos primitivos, cuatro son para valores enteros, 2 son para valores con punto decimal, uno es para almacenar un carácter, y uno más para datos booleanos, que solo puede tener valor de verdadero o falso.

Tipo primitivo	Tamaño	Valor Mínimo	Valor Máximo	Clase

byte	8 bits	-128	127	Byte
short	16 bits	-32,768	32,767	Short
int	32 bits	-2,147,483,648	2,147,483,647	Int
long	64 bits	-9,223,372,036,854,775,808	9,223,372,036,854,775,807	Long
float	32 bits	-3.4028235E + 38 to -1.4E – 45	1.4E–45 to 3.4028235E + 38	Float
double	64 bits	-1.7976931348623157E + 308 to -4.9E–324	4.9E - 324 to 1.7976931348623157E + 308	Doble
char	16 bits	Unicode 0	Unicode 2	Caracter
boolean	-	false (no es valor mínimo)	true (no es valor máximo)	Boolean

Bloque de código (7)

Es un grupo de sentencias que se comportan como una unidad. Un bloque de código está limitado por las llaves de apertura **{** y cierre **}.** Como ejemplos de bloques de código tenemos la definición de una clase, la definición de una función miembro, una sentencia iterativa **for**, los bloques **try ... catch**, para el tratamiento de las excepciones, etc.

Operadores

Operadores Aritméticos

Se utilizan para realizar operaciones aritméticas simples en tipos de datos primitivos.

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)	`0.050 / 0.2` `7 / 2`	`0.25` `3`
%	Resto de la división entera	`20 % 7`	`6`

Operadores Aritméticos Incrementales

Los operadores aritméticos incrementales son operadores unarios (un único operando).

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo

++	Incremento
	`i++` primero se utiliza la variable y luego se incrementa su valor
	`++i` primero se incrementa el valor de la variable y luego se utiliza
		`4++`
		`a=5;`
		`b=a++;`
		`a=5;`
		`b=++a;`
			`5`
			`a vale 6 y b vale 5`
			`a vale 6 y b vale 6`

--	decremento	`4--`	`3`

Operadores Aritméticos Combinados

Combinan un operador aritmético con el operador asignación. Como en el caso de los operadores aritméticos pueden tener operandos numéricos enteros o reales y el tipo específico de resultado numérico dependerá del tipo de éstos. En la siguiente tabla se resumen los diferentes operadores de esta categoría.

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

Operadores De Relación

Realizan comparaciones entre datos compatibles de tipos primitivos (numéricos, carácter y booleanos) teniendo siempre un resultado booleano. Los operandos booleanos sólo pueden usar los operadores de igualdad y desigualdad.

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
==	igual que	`7 == 38`	`false`
!=	distinto que	`'a' != 'k'`	`true`
<	menor que	`'G' < 'B'`	`false`
>	mayor que	`'b' > 'a'`	`true`
<=	menor o igual que	`7.5 <= 7.38`	`false`
>=	mayor o igual que	`38 >= 7`	`true`

Operadores Lógicos y/o Booleanos

Se usa ampliamente para probar varias condiciones para tomar una decisión.

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

Expresiones

Las expresiones son una combinación de operadores y operandos. La precedencia y asociatividad determina de forma clara la evaluación. El orden aplicado es:

- Paréntesis: ( )
- Unarios: + - !
- Multiplicativos: * / %
- Aditivos: + -
- Relacionales: < > <= >=
- Igualdad: == !=
- Y lógico: & &&
- O lógico: | ||

Clases

Definición

En la programación orientada a objetos, el modelado de la realidad se realiza a partir de clases. Las clases son un tipo definido por el usuario en el cual se especifican los atributos y métodos de los objetos que se crearan a partir de la misma. Los atributos definen el estado de un determinado objeto y los métodos son operaciones que definen su comportamiento. Los atributos y los métodos forman parte general de una clase.

Ejemplo. Un Carro y este tiene

Atributos (Características)	Métodos (Acciones)
➢ color	➢ encender
➢ marca	➢ acelerar
➢ km	➢ frenar

Un objeto puede ser tanto tangible como no tangible ejemplo como una fracción:

Atributos (Características)	Métodos (Acciones)
➢ numerador	➢ simplificarse
➢ denominador	➢ sumarse con otra fracción
	➢ restarse con otra fracción

$$\frac{3}{2}$$

Se considera como un molde del que luego se pueden crear múltiples objetos, con similares características. Es una plantilla (molde), que define atributos (variables) y métodos (funciones) Define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

El modelado que se hace de la información y del comportamiento de los miembros de la clase se hace dentro del dominio del problema, es decir, sólo se incluyen las partes que son relevantes para poder resolver el problema planteado. Esto se conoce como **principio de abstracción**.

Ejemplo para el modelado de una Computadora y de un Celular

Computadora ▲	**Identificador de la clase**	**Celular** ▲	**Identificador de la clase**
número de serie marca procesador sistema operativo memoria aula	**Atributos**	marca modelo número de serie cámara memoria	**Atributos**
encender la computadora ejecutar programa almacenar archivos cerrar programa apagar la computadora	**Métodos**	Llamar colgar utilizar calculadora	**Métodos**

Sigue esta nomenclatura y podrás identificar rápidamente a qué se refiere cada identificador, aún sin conocer completamente la estructura de la clase.

- Clases: van capitalizadas e inician con mayúsculas. Se recomienda redactarlo con un sustantivo en singular.
- Atributos: van capitalizados e inician con minúscula.
- Métodos: van capitalizados e inician con minúscula y se incluyen paréntesis al finalizar "`()`". Se recomienda redactarlos con un verbo en infinitivo.
- Los métodos y atributos no incluyen el nombre de la clase, por ejemplo, si la clase es "`Celular`" no se agrega un atributo "`memoriaDelCelular`" ni un método

"`encenderCelular`", ya que es evidente que son elementos de "`Celular`"; en su lugar hay que usar simplemente "`memoria`" y "`llamar()`".

Actividad 1.3.

- Define una clase para un estudiante inscrito en una escuela con los siguientes atributos: nombre, edad, escolaridad, dirección y sus métodos asociados dar de alta, modificar y dar de baja.

- Define una clase para modelar un perro con siguientes atributos: raza, edad, y peso y sus métodos asociados ladrar, comer y dormir.

¿Qué atributos y métodos podrías identificar para modelar el área de un rectángulo?

- Nombre de la clase:
- Atributos de la clase
- Métodos asociados a los miembros de la clase:

Declaración

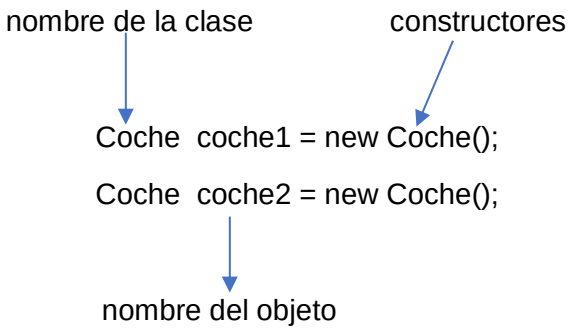
Creación de Clase y objetos

La forma básica para crear las Clases es de la siguiente manera.

```
public class Coche {
.
.
.
}
```

public: modificador de acceso

Coche: nombre de la clase



Si lo creas usando algún IDE como Eclipse, Netbeans o Bluej quedaría de la siguiente manera:

```
*
 * Creación de clases y objetos
 */
public class Coche
{
    //Atributos
    String marca;
    String color;
    int km;

    //Metodo

    public static void main (String[]args){
        Coche coche1 = new Coche();
        coche1.color ="oro";
        coche1.marca ="Audi";
```

```
El color del coche es: oro
La marca del coche es: Audi
El kilometraje del coche es: 0

El color del coche es: rojo
La marca del coche es: Mazda
El kilometraje del coche es: 1700
```

Actividad 1.4. Utiliza cualquier IDE y crea una clase “Animal” con los siguientes atributos raza, nombre y edad, adicionalmente tenemos dos constructores que reciben un nombre y se lo asigna al animal, la raza y la edad.

Atributos

Definición

Los atributos, también llamados datos o variables miembro, son porciones de información que un objeto posee o conoce de sí mismo. Una clase puede tener cualquier número de atributos o no tener ninguno.

Declaración

Para declarar un atributo se hace de la misma manera que al declarar una variable cualquiera, exceptuando por que se tiene que especificar el modificador de acceso. Se declaran con un identificador y el tipo de dato correspondiente. Además, los atributos tienen asociado un modificador que define su visibilidad según se muestra en la siguiente tabla.

Modificador	Visibilidad
`public`	Pública (+)
`protectec`	Protegida / en la herencia(#)
`private`	Privada(-)
`package`	De paquete (~)

Para encapsular por completo el estado de un objeto, todos sus atributos se declaran variables de instancias privadas (usando el modificador de acceso private).

Ejemplo de declaración de atributos en la clase Fecha

```
public class Fecha {
   // Declaracion de atributos o variables miembro
   private int dia;
   private int mes;
   private int ano;
   // Declaracion de metodos . . .
}
```

Métodos

Definición.

Son el conjunto de instrucciones que se definen dentro del cuerpo de una clase, los cuales tienen por objetivo realizar una determinada tarea, y a los que podemos invocar mediante un nombre.

Cuando se llama a un método, la ejecución del programa pasa al método realizando la tarea que se especifica en el cuerpo, al encontrar la palabra **`return`** éste acaba, y la ejecución del programa continúa a partir del punto donde se produjo la llamada.

Las ventajas de emplear métodos son:

- Se construyen programas modulares.
- El código se reutiliza.

En el lenguaje de programación Java existen 2 tipos de métodos: de instancia y de clase. Un método de instancia está siempre asociado a un objeto, esto significa que, para poder emplearlo primero se crea un objeto; por otro lado, un método de clase es aquel que puede ser invocado sin existir una instancia, en la definición de un método de clase se agrega la palabra reservada `static` antes del tipo.

En toda aplicación escrita en el lenguaje de programación Java, en alguna de sus clases, debe de existir el método **`main`**`();` `main` es un método de clase (`static`), en virtud de que no es necesario crear un objeto para invocar, el método **main()** es el punto de inicio de ejecución del programa y el de salida.

Declaración.

La estructura general de un método es:

[modifVisibilidad] [modifFunción] tipo nombreMétodo ([listaParámetros])

[throws listaExcepciones]

[modifVisibilidad]: determinan el tipo de acceso al método.

t**ipo:** tipo dedato devuelto: indica el tipo de valor que devuelve el método, es imprescindible qué en la declaración del método, se indique el tipo de dato que ha de devolver. El dato se devuelve mediante la instrucción **return**, si el método es declarado void (vacio), el método no devuelve ningún valor.

nombremétodo: para definir el nombre del método, hay que seguir las mismas reglas que se especificaron para la creación de identificadores.

[lista de parámetros o argumentos]: indica los datos de entrada que recibe el método para trabajar con ellos, cada argumento o dato debe especificar previamente su tipo de dato y se escriben separados por comas; un método puede recibir tantos argumentos como sean necesarios y en el caso de que no reciba datos, los paréntesis son obligatorios, aunque se encuentren vacíos**.**

Ejemplo completo:
Imaginemos que tenemos estacionado en el garaje un Ford Focus color azul que corre hasta 260 km/h. Si pasamos ese *objeto* del **mundo real** al **mundo del software**, tendremos un *objeto* Automóvil con sus características predeterminadas:

Marca = Ford Modelo = Focus
Color = Azul Velocidad Máxima = 260 km/h

Por lo tanto, dentro de una clase tendremos como variables de clase las características descritas anteriormente y como métodos tendremos en este caso concreto (frenar, acelerar, retroceder, llenar combustible, etc…).

```
class Automovil {
 // VARIABLES DE CLASE
   private String marca;
   private String modelo;
   private String color;
   private String velocidadMaxima;

   // CONSTRUCTOR QUE INICIALIZA LAS VARIABLES DE CLASE
   public Automovil(String marca, String modelo, String color, String velocidadMaxima) {
      this.marca = marca;
      this.modelo = modelo;
      this.color = color;
      this.velocidadMaxima = velocidadMaxima;
   }
   // METODOS GETTER Y SETTER PARA PODER RECUPERAR O CAMBIAR
   // LOS DATOS DE LAS VARIABLES DE CLASE
```

```
public String getMarca() {
  return marca;
}
public void setMarca(String marca) {
  this.marca = marca;
}
public String getModelo() {
  return modelo;
}
public void setModelo(String modelo) {
  this.modelo = modelo;
}
public String getColor() {
  return color;
}
```

Métodos (getter y setter).

Java es un lenguaje de programación orientado a objetos, su esencia es la “clase”, dentro de la cual se definen las variables (atributos) y los métodos, por lo general las variables (atributos) se definen como privados, esta práctica es conocida como “encapsulación” lo que significa proteger los datos de modificaciones sin permiso, cuando las variables (atributos) se encuentran ocultos solo es posible acceder a ellos a través de unos métodos denominados “get” y “set”, también conocidos como “getters” y “setters”, los cuales son métodos especiales, que se declaran como públicos y tienen por propósito interactuar sobre las variables declaradas como privadas.

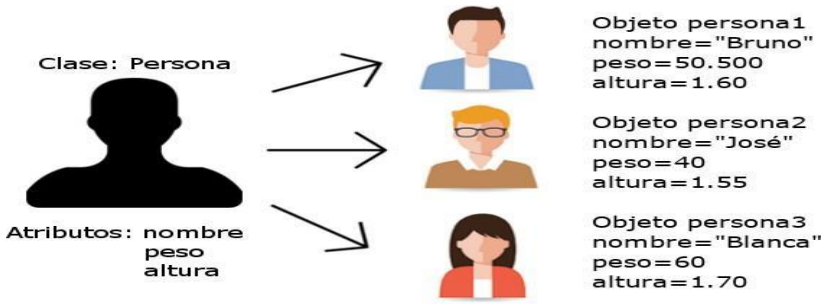
• Getter. Get significa obtener, se usa para recuperar o acceder a un valor ya asignado a un atributo y utilizarlo para cierto método, el método get regresa un valor por lo cual el nombre del método debe ser precedido por el tipo de valor a retornar.

• Setter. Set significa establecer, sirve para asignar un valor a un atributo, el set no regresa dato, así que su valor de retorno debe ser void y siempre debe de tener un parámetro de entrada.

Objetos.

Definición.

Es un componente autónomo creado a partir de una clase, es una estructura de datos existente en la memoria de la computadora que tiene atributos o datos almacenados por el objeto y operaciones disponibles (métodos).



Declaración (instanciación).

Una clase es el plano que describe cómo es un objeto de la clase, a partir de la clase podemos fabricar objetos, a ese objeto construido se le denomina instancia, y al proceso de construir un objeto en la memoria de la computadora se le llama instanciación.

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.

Creación de un objeto.

Para crear un objeto en Java se deben de comprender claramente dos conceptos importantes, el primero es el nombre de la clase para la cual vamos a crear el objeto y el segundo el método constructor que dicha clase posee, es decir, si el método constructor recibe o no recibe parámetros, es importante comentar que el método constructor sirve para inicializar los atributos en el momento de la instanciación, en la unidad 3 de esta guía se maneja a detalle este aprendizaje.

Para crear objetos en el lenguaje de programación Java, empleamos el comando **new**, con este comando le indicamos a Java que se creará un nuevo objeto de una clase determinada, en caso de qué él método constructor lo solicite le enviamos los parámetros necesarios, el formato para crear un objeto es el siguiente:

NombreClase objeto = new NombreClase( );

Implementación

Implementar un programa que, donde se cree una clase celular, con sus atributos: color, modelo, marca, con dos constructores y los métodos llamar, cortar llamada, informar características y los métodos getters y setters que permiten acceder a los atributos de la clase.

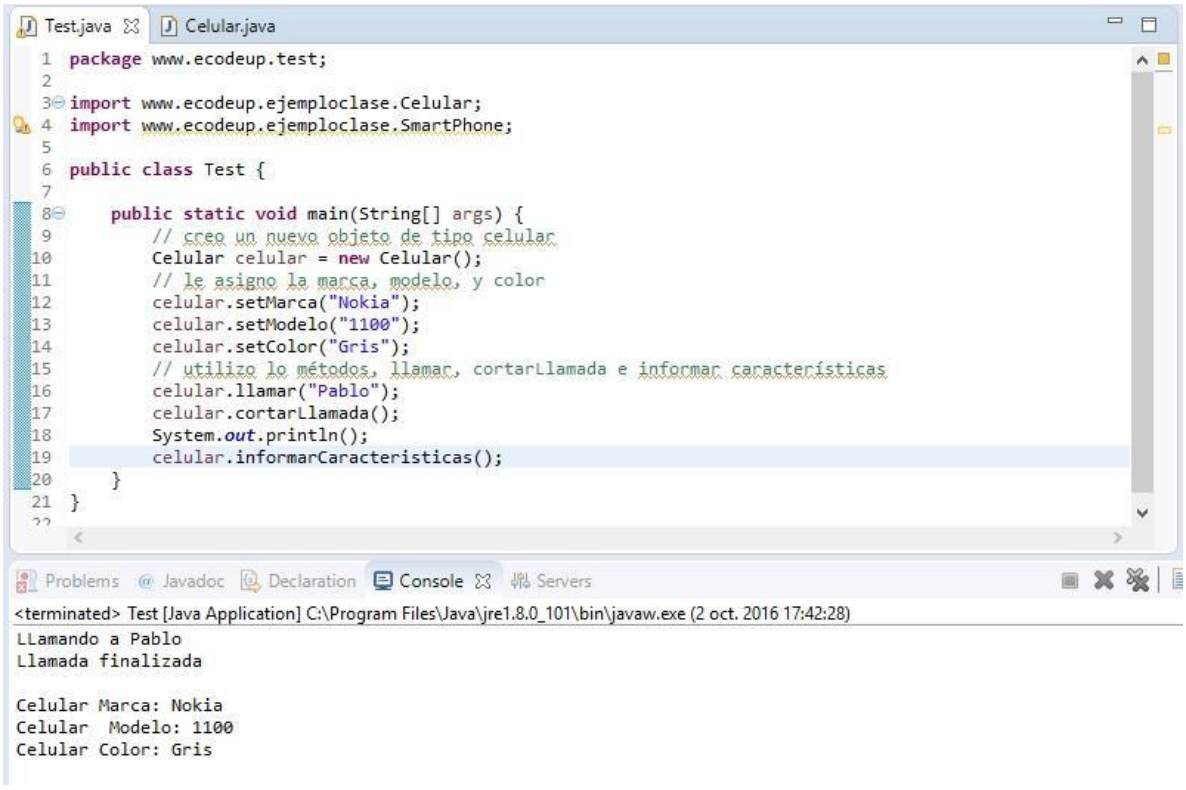
```
public class Celular {
        //atributos de la clase
  private String marca;
  private String modelo;
  private String color;
  // constructor con parámetros
```

```
  public Celular(String marca, String modelo, String color) {
              this.marca = marca;
              this.modelo = modelo;
              this.color = color;
  }
  //constructor vacio
  public Celular(){
  }
  // método hacer llamada
  public void llamar(String nombre){
              System.out.println("LLamando a "+nombre);
  }
  //método finalizar llamada
  public void cortarLlamada(){
              System.out.println("Llamada finalizada");
  }
  //método para informar de la caracteristicas del celular
  public void informarCaracteristicas(){
  System.out.println(String.format("Celular Marca: %s", marca));
  System.out.println(String.format("Celular  Modelo: %s",modelo));
  System.out.println(String.format("Celular Color: %s",
color));
  }
  //getters y Setters
  public String getMarca() {
              return marca;
  }
  public void setMarca(String marca) {
              this.marca = marca;
  }
  public String getModelo() {
              return modelo;
  }
  public void setModelo(String modelo) {
              this.modelo = modelo;
  }
  public String getColor() {
              return color;
  }
  public void setColor(String color) {
              this.color = color;
  }
}
```

¿Cómo crear un objeto de tipo Celular?

```
public class Test {
public static void main(String[] args) {
//creo un nuevo objeto de tipo celular, con el constructor vacío
     Celular celular = new Celular();
// le asigno la marca, modelo, y color
     celular.setMarca("Nokia");
     celular.setModelo("1100");
```

```
      celular.setColor("Gris");
//utilizo lo métodos, llamar, cortarLlamada e informar
características
       celular.llamar("Pablo");
       celular.cortarLlamada();
       System.out.println();
       celular.informarCaracteristicas();
       }
}
```



La clase Scanner.

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 del computador.

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.

Ejemplo:

int edad;

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

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**( );

Métodos para ingresar datos por teclado.

Errores sintácticos y lógicos.

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( );

Actividad 1.5. Relaciona los siguientes conceptos.

	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 basadas 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.
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.

ACTIVIDADES DE PROGRAMACION.

1.1Cree una clase llamada “ControlEscolar” que contenga los siguientes atributos privados.

Nombre atributo	Tipo de dato
Nombre	Cadena
Domicilio	Cadena
Semestre	número entero
numeroMaterias	número entero
Promedio	número decimal

1.2. Cree una clase llamada “Trabajador” que contenga los siguientes atributos privados.

Nombre atributo	Tipo de dato
numeroEmpleado	número entero
Nombre	Cadena
sueldoDiario	numero decimal
diasTrabajados	número entero
importehExtras	numero entero

- Construye los métodos públicos `get` y `sets correspondientes` a cada tipo de dato.

- Construye el método sueldoSemanal, que deberá ser “publico”, devolverá un dato numérico de tipo decimal, y su propósito es calcular el sueldo semanal de un empleado de conformidad a lo siguiente: importeSemanal=(sueldoDiario * diasTrabajados)+importehextras

1.3. Calcular el área y el perímetro de las siguientes figuras geométricas

- ✓ Cuadrado
- ✓ Rectángulo
- ✓ Triángulo
- ✓ Circulo

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.

Estructuras condicionales: simple y múltiple

Las estructuras condicionales son sentencias que dirigen la ejecución de un programa dependiendo del resultado de una expresión lógica.

Estructura if

Sintaxis de la estructura if:

La sintaxis de la sentencia es:

```
if (condición)
  {
    [sentencias]
  }
 else if (condición)
  {
     [sentencias]
  }
 else if (condición)
  {
     [sentencias]
  }
 else
  {
    [sentencias]
  }
```

Considera que:

- ✓ los bloques *else if* y *else* son optativos.

Esta sentencia ejecutará el conjunto de sentencias que se encuentran dentro de las llaves cuando la expresión lógica sea verdadera. Si es falsa ese conjunto de sentencias jamás se ejecutará y se procederá a evaluar la expresión lógica de cada bloque else if. De no evaluarse ninguna expresión lógica como verdadera, se ejecutarán las sentencias del bloque else.

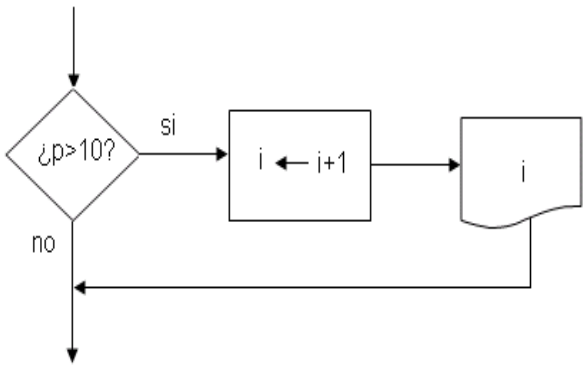
Ejemplo 2.1: Si x es igual a 0 entonces se mostrará el mensaje "El número es par", sino se mostrará el mensaje "El número es impar".

```
if (x==0)  {
        System.out.println("El número es par");
         }
 else  {
       System.out.println("El número es impar");

        }
```

Ejemplo 2.2: Si p es mayor a 10 entonces se incrementa la variable i en uno y se muestra el valor de i.

```
if (p>10)
  {
         i = i++;
         System.out.println (i);
  }
```

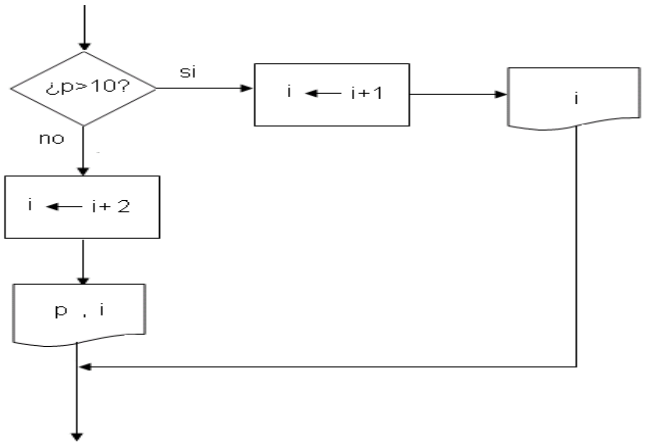
El bloque de diagrama de flujo que representa este ejemplo es:



Ejemplo 2.3: Si p es mayor a 10 entonces se incrementa el valor de i en uno y se muestra el valor de i, sino se incrementa el valor de i en dos y se muestra el valor de p y el de i.

```
if (p>10)
  {
     i =i++;
    System.out.println (i);
   }
else
  {
    i=i+2;
   System.out.println (p,i);
   }
```

El bloque de diagrama de flujo que representa este ejemplo es:



Estructura switch

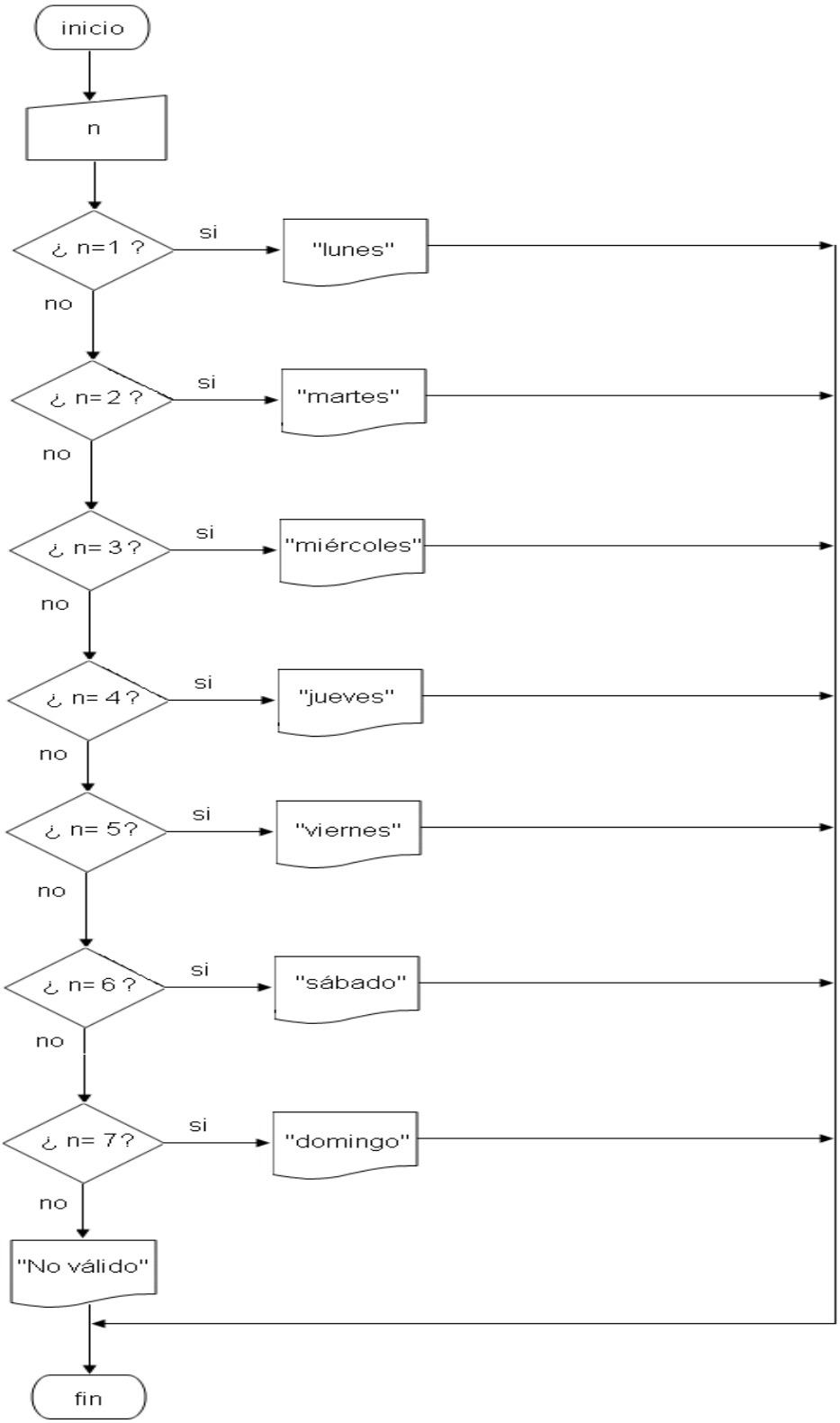
Esta estructura se utiliza cuando se requiere comparar una variable de **tipo entero** contra una lista de posibles valores.

La sintaxis de la estructura switch es:

```
switch(variable)
{
   case valor1:
      [sentencias]
      break;
   case valor2:
      [sentencias]
      break;
   ….
   case valorN:
      [sentencias]
      break;
   [default:
      sentencias]
}
```

Ejemplo 2.4. En la siguiente estructura **switch**, se compara la variable n que es de tipo entero y la cual representa el número de día de la semana:

El bloque de diagrama de flujo que representa este ejemplo es:



Ejemplo 2.5. Programa que obtiene soluciones de ecuaciones de segundo grado

Entrada	Proceso	Salida
a b c	Obtener discriminante Toma de decisión Si discriminante<0 mostrar “Soluciones complejas” en caso contrario obtener x1 y x2	Mostrar el mensaje “Soluciones complejas” o mostrar los valores de x1 y x2

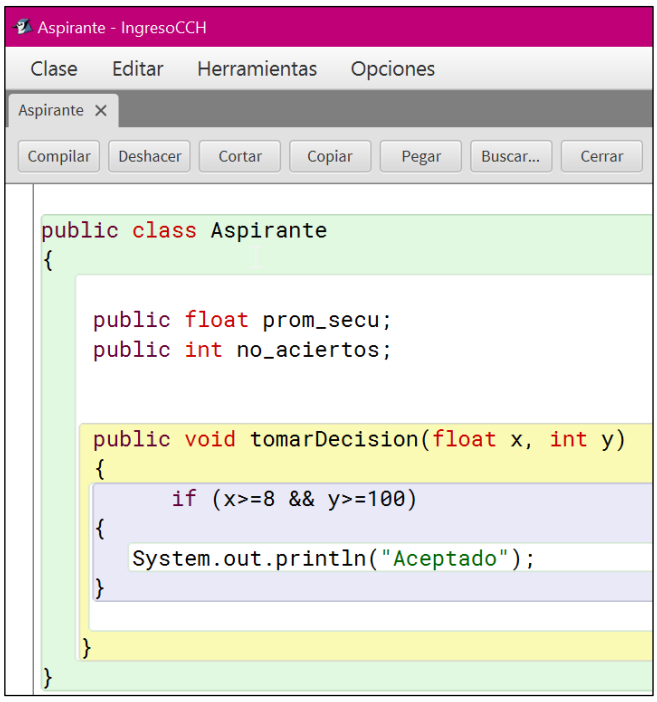
```
public class Ecuacion2
{

    public double a;
    public double b;
    public double c;
    public double x1;
    public double x2;

    public double obtenerDiscriminante(double x,double y, double z)
    {
        double d=Math.pow(y,2)-4*x*z;
        return d;
    }

    public double obtenerX1(double x, double y, double z, double d)
    {
        double r1=(-y+Math.sqrt(d))/(2*x);
        return r1;
    }

    public double obtenerX2(double x, double y, double z, double d)
    {
        double r2=(-y-Math.sqrt(d))/(2*x);
        return r2;
    }
}
```

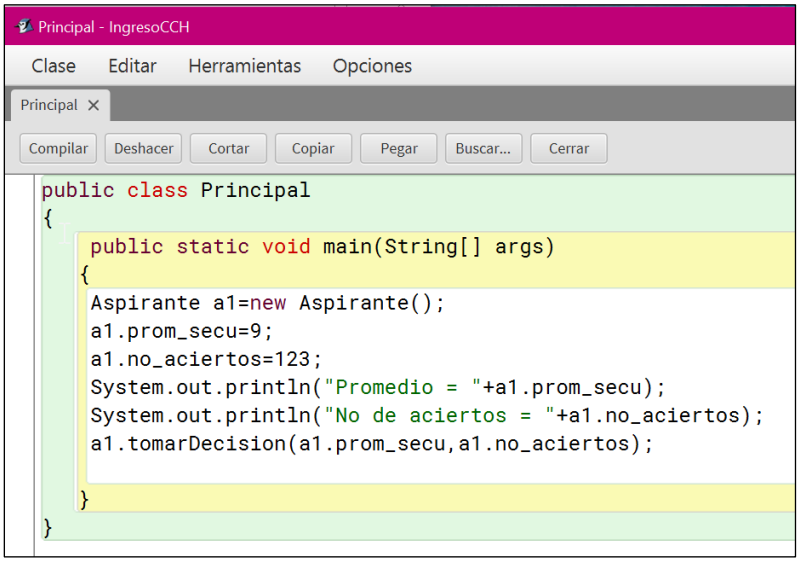
```java
import java.util.Scanner;
public class Principal
{
    public static void main(String[] args)
    {
    Ecuacion2 ecu=new Ecuacion2();
    System.out.print("a = ");
    Scanner sc1=new Scanner(System.in);
    ecu.a=sc1.nextDouble();
    System.out.print("b = ");
    Scanner sc2=new Scanner(System.in);
    ecu.b=sc2.nextDouble();
    System.out.print("c = ");
    Scanner sc3=new Scanner(System.in);
    ecu.c=sc3.nextDouble();
    double discriminante=ecu.obtenerDiscriminante(ecu.a,ecu.b,ecu.c);
    System.out.println("El discriminante es "+discriminante);
    if (discriminante<0)
    {
       System.out.println("Soluciones Complejas");
    }
    else
    {
      System.out.println("X1 = "+ecu.obtenerX1(ecu.a,ecu.b,ecu.c,discriminante));
      System.out.println("X2 = "+ecu.obtenerX2(ecu.a,ecu.b,ecu.c,discriminante));

    }
    }

}
```

Ejemplo 2.6: Un aspirante para ingresar al CCH requiere por lo menos de 100 aciertos en su examen de admisión, pero siempre y cuando su promedio de la secundaria sea mínimo de 8. Si reúne las características mandar mensaje “Aceptado”

Entrada	Proceso	Salida
prom_secu no_aciertos	Toma de decisión Si no_aciertos es por lo menos de 100 aciertos y prom_secu es mínimo de 8 escribe “Aceptado”	Si reúne las condiciones Mensaje de “Aceptado”

Implementación del programa en Java





Actividad 2.1 En cada una de las siguientes situaciones selecciona la estructura condicional que consideres más conveniente o adecuada para representarla en Java. Usa variables de acuerdo con el contexto. De ser necesario utiliza los bloques *else if*, *else* y *default*. Nota: No se solicita que realices el programa.

1. En el sistema de pago de un restaurante, si la variable cuenta es mayor de 400, se realiza un descuento del 10%.
2. En el registro de la base de datos correspondiente a cada empleado de una empresa, si la variable anios es mayor a 5, la empresa aumenta a la variable salario, el 2% por el valor de la variable anios.

3. “Estoy muy preocupado y debo estudiar mucho, ya que, si obtengo un promedio inferior a 7 en Cibernética, me descontarán $100 a la cantidad de dinero que me dan por semana”
4. “Me motiva pensar que si obtengo un promedio mínimo de 8 me darán $100 más por semana, pero me estresa pensar que si no lo obtengo me darán $50 menos.
5. En el trabajo de Karla, si el número de ventas por semana es por lo menos de 20 le aumentarán el sueldo 20%, sino solamente le dirán que le “eche más ganas”.
6. Si la edad de los empleados de la empresa Bachoco es mínimo 20 y máximo 25 años la empresa les da un 10% adicional a su salario.
7. Si obtengo un par al lanzar un dado perderé $20 de cierta cantidad, sino ganaré $50.
8. Si el valor de la variable d es 0 mostrar el mensaje “raíces iguales”, si el valor de la variable d es negativo mostrar el mensaje “raíces imaginarias, en cualquier otro caso mostrar el mensaje “raíces reales”.
9. Si tu estatura es mayor a 1.50 metros y no pesas más de 55 kilos podrás ingresar como edecán y tu sueldo base será de $9000 mensuales, sino podrán ingresar como vendedora y tu sueldo base será de $8000.
10. Si en tu examen obtuviste una calificación superior a 8 o el número de errores no excede a 3 tendrán un punto adicional en el examen.
11. Si tu calificación es menor a 6 mostrar el mensaje “Reprobado”, si tu calificación es mayor o igual a 6 pero menor a 7.5 mostrar el mensaje “Suficiente”, si tu calificación es mayor o igual a 7.5 pero menor a 9.0 mostrar el mensaje “Bien” y si tu calificación es mayor o igual a 9.0 y menor o igual a 10 mostrar el mensaje “Muy bien”. Si tu calificación no se encuentra en ninguno de los rangos anteriores mostrar el mensaje “Inválida”.
12. Si el valor de la variable num_sem es 1 se mostrará el mensaje “primero”, si es valor de la variable num_sem es 2 se mostrará el mensaje “segundo”, así hasta si el valor de la variable num_sem es 6 se mostrará el mensaje “sexto”. En caso de que la variable num_ sem no tome alguno de los anteriores valores, mostrar el mensaje “inválido”.
13. Considerando el valor de las variables a, b y c, mostrar el mensaje “tercia” si las tres variables tienen el mismo valor, mostrar el mensaje “par” si sólo un par de variables tienen el mismo valor y mostrar el mensaje “nada” si los valores de las tres variables son diferentes.

14. Mostrar con letra el valor de un número entero entre 20 y 50. Si el número no está dentro del rango, mandar el mensaje “Número no permitido”

Actividad 2.2 Desarrolla usando el compilador de Java, los siguientes programas:

a) Programa para mostrar el mensaje de “REPROBADO” si el promedio de 3 calificaciones es menor a 6 o si alguna de las calificaciones es menor a 6, en caso contrario mostrar el mensaje “APROBADO”.

b) Programa que muestre y ordene de mayor a menor 4 números cualesquiera.

Estructuras de control de repetición

¿Qué es un contador y un acumulador?

Contador
Variable que inicia con un valor numérico, generalmente 0 o 1, y que va aumentando o disminuyendo a condición del programa. Por ejemplo: i++, significa que la variable «i» aumentará su valor de 1 en 1.

Acumulador
También es una variable, como su nombre indica, va acumulando valores diferentes.
Por ejemplo:

$$SumadeCalificaciones = SumadeCalificaciones + Calificación$$

La diferencia entre un contador y un acumulador es su incremento, en el contador el incremento es una constante y en el acumulador es una variable.

Estructura repetitiva for

Repite un conjunto de sentencias mientras una condición se cumpla. Por lo regular se usa cuando está determinado el número de veces que se ejecutarán el conjunto de sentencias.

```
for (inicio; condición de paro; cambio) {
    [sentencias]
}
```

Ejemplo 2.7

```
for (int i=0;i<10; i++)  {
                        System.out.println(
i);
                }
```

Ejemplo 2.8

Código para mostrar en la pantalla los números enteros entre el 1 y 100 utilizando el ciclo FOR.

```
public class ejemplo
{
    public static void main (String[]args){
            for (int cont=1; cont<=100; cont++){
                System.out.println(cont);
             }
    }
}
```

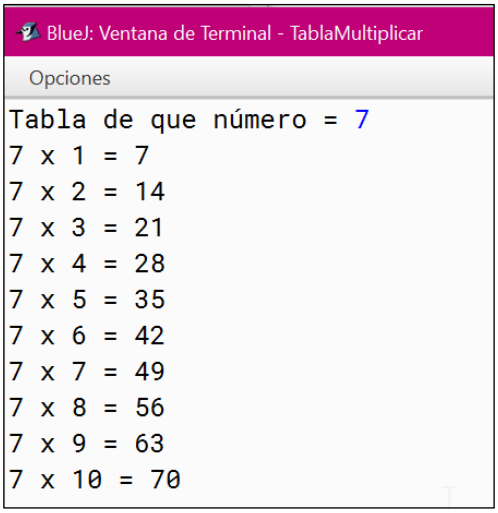
Ejemplo 2.9 Desplegar una tabla de multiplicar, del número pedido por el usuario.

Entrada	Proceso	Salida
número	Obtener y mostrar el multiplicar: $n * i = r$ donde i empieza en 1 y termina en 10	Mostrar cada uno de los renglones que conforman la tabla

```
public class Tabla
{

    public int n;
    public void mostrarTabla(int x)
    {
        for (int i=1;i<=10; i++)  {
            System.out.println(x+" x "+i+" = "+x*i);
        }

    }
}
```

```
import java.util.Scanner;
public class Principal
{
     public static void main(String args[])
    {
        Tabla t=new Tabla();
        System.out.print("Tabla de que número = ");
        Scanner sc1=new Scanner(System.in);
        t.n=sc1.nextInt();
        t.mostrarTabla(t.n);
    }
}
```

Impresión de la salida de la ejecución del programa



Actividad 2.3 Elabora un programa en Java para cada una de las siguientes situaciones:

a) Mostrar los primeros 200 pares del mayor al menor.

b) Mostrar los divisores de un número.

c) Elevar un valor *a* de tipo real proporcionado por el usuario a un número *b* de tipo entero también proporcionado por el usuario

d) Menciona si un número introducido por el usuario es primo o no.

Estructura repetitiva while

Repite un conjunto de sentencias. Se utiliza cuando el conjunto de sentencias se ejecutará in indeterminado número de veces.

```
while (condición) {
    [sentencias]
}
```

Ejemplo 2.10 Mientras la variable i sea menor a 5 se mostrará el valor de i y se incrementará su valor en 1

```
while (i<5)  {
  System.out.println( i);
  i++;
}
```

Estructura repetitiva do while

La diferencia entre ciclo while y el ciclo do while consiste en que el conjunto de sentencias por lo menos se ejecutará una vez.

```
do {
    [sentencias]
} while (condición);
```

Ejemplo 2.11 Se mostrará el valor de i y se incrementará su valor en 1 mientras la variable i sea menor a 5

```
do {
  System.out.println( i);
  i++;
} while (i<5);
```

Actividad 2.4 Elabora un programa en Java para cada una de las siguientes situaciones:

a) Un empleado dejará de capturar nombres hasta que haya capturado la frase “salida”
b) Se dará la oportunidad de introducir una clave hasta que sea la correcta. La clave correcta es “7cchn7w”
c) Se solicita la clave personal al usuario y se le da tres oportunidades para hacerlo correctamente. La clave personal es 4334.
d) Usando la estructura while para mostrar los números enteros entre 10 y 200.
e) Se selecciona de manera aleatoria una letra minúscula, el usuario no sabrá cuál se seleccionó y tratará de adivinarla hasta que acierte la letra minúscula seleccionada de manera aleatoria. También el programa contará el número de intentos que el usuario realizó.
f) Obtener el Mínimo Común Múltiplo de dos números enteros proporcionados por el usuario. Sugerencia: Utiliza un contador que inicie en a y su incremento sea de a en a, donde a y b son los dos números introducidos por el usuario. Dejarás de incrementar el valor de ese contador cuando el residuo de la división de b entre ese contador sea 0. El MCM será el valor final del contador.
g) Programa que proporcione de manera aleatoria un número entero del 1000 al 9999 sin mostrarlo al usuario, él tratará de adivinarlo y el programa mostrará mensajes que le dirán si el número que proporcionó es mayor o menor al que debe de adivinar; esto lo hará hasta que haya insertado el número de la computadora (aleatorio). Finalmente, el programa deberá de mencionar el número de valores que introdujo para adivinar el número.

Arreglos

Son conjuntos de variables en los que todos son del mismo tipo. Cuando se crea el arreglo se determina el número de variables que lo conforman y este no podrá ser modificado. Cada variable se llama elemento y se podrá acceder a cada uno por medio de un índice, el cual comenzará en cero, por lo que el último será igual al tamaño del arreglo menos uno.

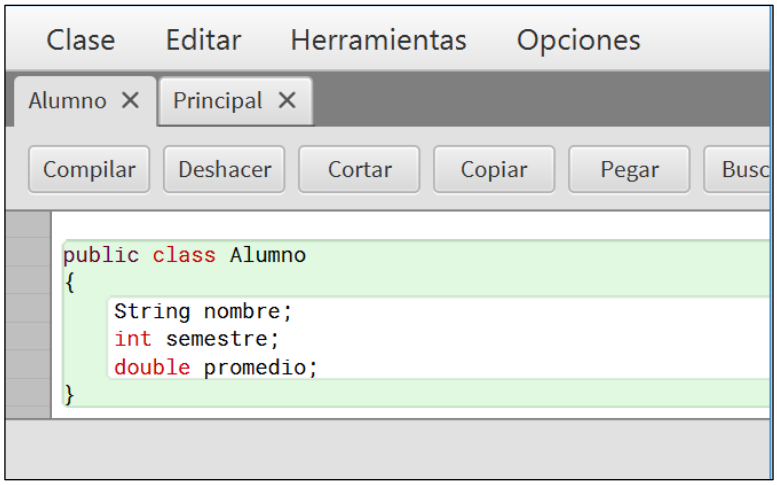
Arreglos unidimensionales

La sintaxis de su creación es la siguiente:

TipoDeDato identificador[] = new TipoDeDato[longitud del arreglo];

```
int arreglo [] = new int [10]

/*se declara el objeto de tipo arreglo de enteros llamado arreglo y se aparta en memoria el
  espacio para almacenar 10 enteros y se asigna al objeto arreglo */
```

Ejemplo 2.12 En la siguiente imagen observa que se crea una *clase Alumno* y en la imagen subsecuente observa se declara el objeto de tipo arreglo de elementos *Alumno* llamado grupo y se aparta en memoria el espacio para almacenar 4 elementos de tipo Alumno y se asigna al objeto grupo



```
import java.util.Scanner;
public class Principal
{

   public static void main(String args[])
   {
    Alumno grupo[]=new Alumno[4];
    Scanner sc1=new Scanner (System.in);
    for (int i=0;i<=6;i++)
    {
      System.out.print("Dame el nombre del alumno "+(i+1)+" : ");
      String nombre2=sc1.nextLine();

      System.out.print("Semestre del alumno "+i+1+" : ");
      grupo[i].semestre=sc1.nextInt();

      System.out.print("Promedio del alumno "+i+1+" : ");
      grupo[i].promedio=sc1.nextDouble();

    }
   }
}
```

Ejemplo 2.13 Programa que ordena un conjunto de números enteros

```java
import java.util.Scanner;
public class Numeros {
    int[] numeros=new int[6];

    public void pedirNumeros(){
        for (int i=0;i<numeros.length;i++){
            System.out.println("número "+i);
            Scanner sc=new Scanner(System.in);
            numeros[i]=sc.nextInt();
        }
    }

    public void ordenarNumeros(){
        int aux;int i; int j;
        for (i=0;i<numeros.length-1;i++)
        {
            for (j=i+1;j<numeros.length;j++)
            {
                if (numeros[i]>=numeros[j])
                {
                    aux=numeros[i];
                    numeros[i]=numeros[j];
                    numeros[j]=aux;

                }
            }
        }
    }

    public void mostrarNumeros(){
        int i;
        for (i=0;i<numeros.length;i++)
        {
            System.out.println(numeros[i]);
        }
        }
    }
```

```java
public class Principal {

    public static void main(String[] args) {
        Numeros numeros= new Numeros();
        numeros.pedirNumeros();
        numeros.ordenarNumeros();
        numeros.mostrarNumeros();
        }
}
```

Actividad 2.4 Elabora un programa en Java para cada una de las siguientes situaciones:

a) Solicitar un número entero, obtener y mostrar su equivalente en sistema de numeración binario.
b) Obtener la mediana de un conjunto de valores. Sugerencia: Debes de ordenar los valores primero.
c) Obtener la moda de un conjunto de valores.
d) Solicitar al usuario n cantidades de ventas de un restaurante, mostrar las mayores a $400 y mencionar cuántas son.

Arreglos bidimensionales

Los arreglos bidimensionales son conocidos como matrices y las dimensiones deben ser números enteros al igual que los arreglos unidimensionales. Se crean usando doble corchetes [] [], ya que representan el tamaño de filas por columnas que puede tener

```
<tipoDato>  identificador[ ] [ ];
```

Ejemplo 2.14: int ArregloDeEnteros[ ] [ ];

Considera que:

- Un arreglo bidimensional necesita dos índices para acceder a sus elementos.
- Los valores iniciales se escriben entre llaves separados por comas.
- Los valores que se le asignen a cada fila aparecerán a su vez entre llaves separados por comas.
- El número de valores determina el tamaño de la matriz,

Ejemplo 2.15:

```
int [][] valores = {{4,3},{1,4}, {0,7}, {5,3}};
```

Se creó un arreglo bidimensional llamado valores de tipo int, de 4 filas y 2 columnas. Se le asignaron valores iniciales.

Para recorrer una matriz se requieren dos estructuras for.

Ejemplo 2.16 Uso de un arreglo bidimensional

```
public class ArregloBi{
    public static void main(String[] args) {
        char arreglo[][] = new int[4][4];
        for (int i = 0; i < arreglo.length; i++) {
            for (int j = 0; j < arreglo[0].length; j++) {
                arreglo[i][j] = 0;
                if (i == j) {
                    arreglo[i][j] = 1;
                }
            }
        }
        for (int i = 0; i < arreglo.length; i++) {
                    System.out.println();
                    for (int j = 0; j < arreglo[0].length; j++)
{
                                System.out.print(arreglo[i][j] + " ");
            }
        }
    }
  }
```

Actividad 2.5 Escribe un comentario a cada una de las líneas del código anterior

Actividad 2.6 ¿Cuál es la salida si se ejecuta el código Java anterior?

Unidad 3. Polimorfismo, constructores, colaboración y herencia de clases.

Propósito:

Al finalizar la unidad el alumno:

Implementará programas en Java utilizando polimorfismos, constructores, colaboración y herencia de Clases para aprovechar las bondades de la programación orientada a objetos.

Concepto de Polimorfismo.

El polimorfismo es una técnica asociada a la programación orientada a objetos que permite la posibilidad de que una referencia a un objeto de una clase pueda utilizarse también en las clases derivadas de éstas, es decir, se refiere a la capacidad de clases derivadas de utilizar un método de modo diferente.

También podemos decir que se refiere a la capacidad de los objetos de comportarse de acuerdo con la funcionalidad requerida. Es decir, establece diferentes comportamientos para los métodos del objeto, de manera que implementa diversas funcionalidades en relación con parámetros recibidos a través de estos. Por ejemplo, se puede realizar una suma de enteros que devuelvan enteros y reciba parámetros enteros, o una suma de valores flotantes donde se reciban parámetros flotantes y devuelva un valor flotante. Ambos métodos pueden llamarse igual, pero se diferencian por los parámetros que reciben. Por último, podemos concluir diciendo que el polimorfismo es una característica de la programación orientada a objetos en donde un método se comunica con clases diferentes, y dependiendo de la clase con la que se tenga comunicación, su comportamiento será completamente diferente.

Existen dos tipos de polimorfismos: dinámico y estático.

El polimorfismo dinámico, llamado también polimórfico, es el que no incluye especificación alguna sobre el tipo de datos en el que opera; por ende, se puede utilizar en cualquier dato que sea compatible. Al polimorfismo dinámico se le conoce también como programación genérica.

El polimorfismo estático, también denominado ad hoc, es aquel donde los tipos de datos deben de ser explícitos y declarados en forma individual antes de ser utilizados.

Ejemplo 3.1

Veamos la siguiente clase que implementa cinco métodos, tres con un mismo nombre y dos con otro nombre pero que estos reciben parámetros de distinto tipo y devuelven resultados también de diferentes tipos.

Una de las clases es para el manejo de cadenas, otra para enteros y otro para reales, siendo las operaciones de suma y resta.

```
public class EjemploPolimorfismo {
        String sumar(String a, String b){
        return a+b;
        }
        int sumar(int a, int b){
        return a+b;
        }
        double sumar(double a, double b){
 return a+b;
         }
        int restar(int a, int b){
        return a-b;
        }
        double restar(double a, double b){     return a-b;
         }
}
```

Figura 3.1 Clase de un ejemplo de Polimorfismo.

Dentro de esta clase Ejemplo.Polimorfismo, creamos una clase principal.

```
public static void main(String [ ] args) {
        EjemploPolimorfismo aux=new  EjemploPolimorfismo();
        System.out.println("Suma de cadenas:  " + aux.sumar(
"junta ","palabras"));
        System.out.println("suma enteros: " + aux.sumar(10,4));
        System.out.println("suma reales: " + aux.sumar(10.3,8.6));
        System.out.println("resta enteros: " + aux.restar(4,9));
        System.out.println("suma reales: " + aux.restar(16.4,7.2));
}
```

Figura 3.2 Clase Principal del ejemplo de un Polimorfismo.

El programa completo quedaría como sigue:

```
1
2   package ejemplopolimorfismo;
3
4
5   public class EjemploPolimorfismo {
6           String sumar(String a, String b){
7           return a+b;
8           }
9           int sumar(int a, int b){
10          return a+b;
11          }
12          double sumar(double a, double b){
13          return a+b;
14          }
15          int restar(int a,int b){
16          return a-b;
17          }
18           double restar(double a,double b){
19          return a-b;
20          }
21
22      public static void main(String[] args) {
23        EjemploPolimorfismo aux=new EjemploPolimorfismo();
24        System.out.println("suma de cadenas: "+aux.sumar("junta "," palabras"));
25        System.out.println("suma enteros:  "+aux.sumar(10,4));
26        System.out.println("suma reales:  "+aux.sumar(10.3,8.6));
27        System.out.println("resta enteros:  "+aux.restar(4,9));
28        System.out.println("resta reales:  "+aux.restar(16.4,7.2));
29      }
30
31  }
```

Figura 3.3 Programa de un ejemplo de Polimorfismo.

Al correr el programa se obtiene:

```
Output - EjemploPolimorfismo (run)
run:
suma de cadenas: junta  palabras
suma enteros:  14
suma reales:  18.9
resta enteros:  -5
resta reales:  9.2
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 3.4 Salida del Programa EjemploPolimorfismo.

Concepto de constructor

Los constructores en Java son métodos que se encargan de dar un estado inicial a nuestro objeto. Estos métodos tienen la característica de que se llaman igual que la propia clase, por lo cual debemos tener un método dentro de la clase que se llame igual que ella.

Cuando se crea un objeto, debemos construir un constructor, sin embargo, si no lo hacemos, Java construye un constructor por defecto.

Podemos afirmar que todas las clases tienen constructores, ya sea que definamos uno o no, porque Java proporciona automáticamente un constructor determinado. Una vez que definimos un constructor, el constructor por defecto de Java ya no se usa.

Un constructor inicializa un objeto cuando se crea. Tiene el mismo nombre que su clase y es similar a un método.

Vamos a construir un programa en donde se muestre un Constructor.

Ejemplo 3.2

```
Source  History
1
2   package ejemploconstructor;
3
4   class MiConstructor {
5   int a;
6   MiConstructor (){
7   a=20;
8
9   }
10
11  }
12
13  public class EjemploConstructor {
14
15      public static void main(String[] args) {
16
17          MiConstructor x=new MiConstructor();
18          MiConstructor y=new MiConstructor();
19           System.out.println(x.a + " Salida del valor inicial "+y.a);
20
21      }
22
23  }
24
```

Figura 3.5 Ejemplo de un Constructor.

Al correr el programa se tiene:

```
Output - EjemploConstructor (run)
run:
20 Salida del valor inicial 20
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 3.6 Salida del programa EjemploConstructor.

Implementación de constructores con polimorfismo

Realizaremos un programa en Java en donde se muestre la manera en que se utilizan las características del Polimorfismo y constructores.

Ejemplo 3.3

Elaborar un programa en Java que calcule e imprima la permutación ${}_nP_r$, dando a “n” y “r” como dato.

La permutación solicitada se define como:

$$ {}_nP_r = \frac{n!}{(n-r)!} $$

Donde n! nos indica el factorial de un número n entero no negativo definido como:

$$ n! = 1x2x3x \ldots x\ (n-1)xn $$

Por ejemplo:

$$ 3! = 1x2x3 = 6 $$

$$ 5! = 1x2x3x4x5 = 120 $$

Por definición

$$ 0! = 1! = 1 $$

Un ejemplo para la Permutación sería:

$$ P_5 = \frac{8!}{(8-5)!} = \frac{1x2x3 \ldots x8}{1x2x3} = \frac{40320}{6} = 6720 $$

Primero elaboramos el programa que llamamos **Factorial** que será usado en nuestro ejemplo.

Recordamos que el factorial de un número entero positivo “n”, denotado por n! se define como sigue:

$$ n! = 1 * 2 * 3 * \ldots * (n-1) * n $$

Por ejemplo:

El programa en Java para calcular el factorial de un número n puede ser:

```java
package factorial;
import java.util.Scanner;

public class Factorial {


    public static void main(String[] args) {
        Scanner entrada=new Scanner(System.in);
        int i,n;
        double fact;
        System.out.println("dame un número entero no negativo");
        n=entrada.nextInt();
        fact=1;
        for(i=1;i<=n;i++)
        {
            fact=fact*i;

        }
     System.out.println("El factorial "+n+"!="+fact);
    }
}

```

Figura 3.7 Programa Factorial.

Al correr el programa se tiene:

```
Output - Factorial (run)
dame un número entero no negativo
5
El factorial 5!=120.0
BUILD SUCCESSFUL (total time: 3 seconds)
```

Figura 3.8 Salida del programa Factorial.

Observación: La variable **fact** se declara **double** para que se puedan realizar cálculos más grandes del factorial**,** ya que, para enteros, después del número 10 los resultados del factorial ya no son confiables.

Utilizando el programa **Factorial** elaboramos nuestro programa que llamaremos: PermutaciónPolimorfismo, quedando el código de la siguiente manera:

```java
package permutaciónpolimorfismo;

import java.util.Scanner;


public class PermutaciónPolimorfismo {

int factorialn(int n){
    int fact=1;
    for(int i=1;i<=n;i++)
    {
        fact=fact*i;
    }
    return fact;
}
    public static void main(String[] args) {
         PermutaciónPolimorfismo aux=new PermutaciónPolimorfismo();
        Scanner sc=new Scanner(System.in);
        System.out.println("Dame el valor de n");
        int n=sc.nextInt();
         System.out.println("Dame el valor de r");
        int r=sc.nextInt();
        System.out.println("el factorial de n es ="+aux.factorialn(n));
        int v=n-r;
        System.out.println("el factorial de n-r es ="+aux.factorialn(v));

        System.out.println("La permutación nPr es ="+aux.factorialn(n)/aux.factorialn(n-r));
    }

}
```

Figura 3.9 Programa PermutaciónPolimorfismo.

La salida del programa para n=8 y r=3 es:

```
Output - PermutaciónPolimorfismo (run)
run:
Dame el valor de n
8
Dame el valor de r
3
el factorial de n es =40320
el factorial de n-r es =120
La permutación nPr es =336
BUILD SUCCESSFUL (total time: 5 seconds)
```

Figura 3.10 Salida del programa PermutaciónPolimorfismo.

Interacción y comunicación entre clases

Para mostrar un ejemplo de un programa en Java en donde se observe la interacción y comunicación de las clases, utilizaremos el programa **Factorial** de una manera un poco diferente a los anteriores.

Ejemplo 3.4

Elaborar un programa en Java que calcule e imprima el valor de A dada por la expresión:

$$A = \frac{B^3 - C!}{C - \frac{2B! + 3}{\sqrt{D}}}$$

Dando los valores de B, C y D como datos, utilizando el programa de factorial como una subclase de la clase principal.

El programa realizado puede ser:

```
Source  History
1
2   package evaluarexpresión;
3
4   import java.util.Scanner;
5
6   public class EvaluarExpresión {
7
8       public static void main(String[] args) {
9           Scanner entrada= new Scanner(System.in);
10          int B,C,D;
11          double A;
12          System.out.println("dame B");
13          B=entrada.nextInt();
14          System.out.println("dame C");
15          C=entrada.nextInt();
16          System.out.println("dame D");
17          D=entrada.nextInt();
18          A=(double)(Math.pow(B,3)-factorial(C))/(C-(2*factorial(B)+3)/Math.sqrt(D));
19          System.out.println("El valor de A = "+ A);
20      }
21      public static int factorial(int n)
22      {
23          int fact=1;
24          int i;
25          for(i=1;i<=n;i++)
26          {
27              fact=fact*i;
28          }
29          return fact;
30      }
31  }
```

Figura 3.11 Programa EvaluarExpresión.

Probando con B=2, C=3 y D=9, tenemos la siguiente salida:

```
Output - EvaluarExpresión (run)
dame B
2
dame C
3
dame D
9
El valor de A = 3.000000000000001
BUILD SUCCESSFUL (total time: 7 seconds)
```

Figura 3.12 Salida del programa EvaluarExpresión.

Observación: En este programa se evitó el uso de la variable auxiliar **aux.**

Herencia

La herencia en la programación orientada a objetos es la facilidad mediante la cual una clase hereda las propiedades y métodos públicos de otra. La herencia permite compartir métodos y datos entre clases, subclases y objetos.

Por medio de la herencia podemos definir una clase a partir de otra ya existente. Podemos llamar a la existente super clase y a la segunda subclase.

Con la herencia podemos reutilizar código de nuestro programa. Las subclases pueden utilizar el código de la super clase sin tener que volver a definirlos en cada una de ellas.

La herencia en Java tiene la siguiente sintaxis:

```
public class B extends A {
⋮
}
```

Lo cual nos dice que una clase B hereda de una clase A

Ejemplo 3.5

Analizaremos la Herencia en Java construyendo dos clases: Una llamada **Trabajador** y otra llamada **Docente**.

La clase **Trabajador** tiene los atributos **Ct** (clave de trabajador) y **nombre**.

```
// Super Clase Trabajador
public class Trabajador {
        private String ct;
        private String nombre;
public String getCt () {
                return Ct;
        }
public void setCt(String Ct) {
                this.Ct=Ct;
        }
Public String getNombre() {
                Return nombre;
        }
Public void setNombre(String nombre) {
                This.nombre=nombre;
        }
```

Figura 3.13 Código de la Super Clase Trabajador.

```
// La subclase Docente
public class Docente extends Trabajador{
        private String materia;
public String getMateria () {
return materia;
}
Public void setMateria(String materia) {
This.materia=materia;
}
```

Figura 3.14 Código de la subclase Docente.

La super clase **Trabajador** tiene los atributos: **Ct** y **nombre** que hereda a la clase **Docente**, la cual tiene a su vez el atributo **materia** como propio.

Implementación de la herencia de Clases

Describimos la implementación de la Herencia en Java con el código del siguiente ejemplo:

```
1
2   package ejemploherenciaclase;
3
4   import java.util.Scanner;
5
6   public class EjemploHerenciaClase {
7    public abstract class Operaciones {
8        int v1,v2,resultado;
9       Scanner entrada=new Scanner(System.in);
10
11      public void SolicitarDatos() {
12      System.out.println("Dame el primer valor");
13      v1=entrada.nextInt();
14      System.out.println("Dame el segundo valor");
15      v2=entrada.nextInt();
16      }
17  public abstract void OperacionesYa();
18  public void MostrarResultado (){
19  System.out.println("resultado = " + resultado);
20  }
21  }
22   public static class Suma extends Operaciones{
23  @Override
24
```

Figura 3.15 Primera parte del programa EjemploHerenciaClase.

```
Source  History
22  public static class Suma extends Operaciones{
23  @Override
24
25  public void OperacionesYa () {
26  resultado=v1+v2;
27  }
28  }
29  public static class Resta extends Operaciones{
30  @Override
31  public void OperacionesYa () {
32  resultado=v1-v2;
33  }
34  }
35      public static void main(String[] args) {
36         Operaciones realizarsuma=new Suma();
37          System.out.println("Operación suma");
38          realizarsuma.SolicitarDatos();
39          realizarsuma.OperacionesYa();
40          realizarsuma.MostrarResultado();
41          System.out.println("Operación resta");
42          Operaciones realizaresta=new Resta();
43          realizaresta.SolicitarDatos();
44          realizaresta.OperacionesYa();
45          realizaresta.MostrarResultado();
46      }
47  }
```

Figura 3.16 Segunda parte del programa EjemploHerenciaClase.

La ejecución del programa para dos valores nos muestra la salida:

```
Output - EjemploHerenciaClase (run)
run:
Operación suma
Dame el primer valor
15
Dame el segundo valor
13
resultado = 28
Operación resta
Dame el primer valor
15
Dame el segundo valor
13
resultado = 2
BUILD SUCCESSFUL (total time: 17 seconds)
```

Figura 3.17 Salida del programa EjemploHerenciaClase.

Actividades 3.1

1. Elaborar un programa en Java que calcule e imprima la combinación $(n\ r\ )$, dando a “n” y “r” como dato.

   La combinación solicitada se define como sigue:

$$(n\ r\ ) = \frac{n!}{r!\,(n-r)!}$$

Un ejemplo para la Combinación sería:

$$(8\ 5\ ) = \frac{8!}{5!\,(8-5)!} = 56$$

2. Tomando en cuenta que la función exponencial e^x se puede expresar como la serie infinita:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Y además que:

$$e^x = \frac{x^0}{0!} + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$$

Calcule el valor aproximado de:

a) $e = e^1$
b) e^2
c) $e^{1.5}$

Utilizando los cinco primeros sumandos de la serie infinita. (recuerde que e=2.71828… y compare sus resultados)

3. Elaborar un programa en Java que calcule e imprima el valor de A, dada por la expresión:

   a)

$$A = \frac{2C + B!}{C^2 - \frac{\sqrt{D}}{4}}$$

   b)

$$A = \frac{5B - 4C!}{8 + \sqrt{\frac{D^3}{8}}}$$

Dando los valores de B, C y D como datos, utilizando el programa **Factorial** como una subclase de la clase principal.

4. Elaborar un programa en Java que calcule e imprima el área de un polígono regular de n lados, dando “n”, el apotema y la longitud “L” de un lado como datos, utilizando un programa “Perímetro” como una subclase de la clase principal que calcula el área.

5. Construir un programa en Java que solicite 10 números enteros positivos,

   a) Que los imprima de la manera en que se dieron.
   b) Que los ordene del menor al mayor y los imprima.
   c) Que los ordene del mayor al menor y que los imprima, indicando si cada cifra es un número primo o no.

   Observación: Para el inciso c) utilice un programa “NúmeroPrimo” como una subclase de la clase principal que “OrdenaCifras”.

6. Realizar un programa en Java que solicite un número entero positivo de 4 cifras, que separe las cifras y que las imprima indicando si cada cifra es un número primo o no.
   Observación: Utilice un programa “número primo” como una subclase de la clase principal que “SeparaCifras”.

7. Elaborar un programa en Java que calcule e imprima la tabla de los factoriales del 1 a 10, utilizando la subclase “factorial”.

1!	1
2!	2
3!	6
4!	24
⋮	⋮
8!	40320
9!	362880
10!	3628800

8. Construir un programa en Java que calcule el volumen de un cilindro, dando el radio de la base y la altura del cilindro como datos, utilizando una subclase que calcule el área de la base del cilindro.

9. Elaborar un programa en Java que calcule el volumen de un cono, dando el radio de la base y la altura del cono como datos, utilizando una subclase que calcule el área de la base del cono.

10. Realizar un programa en Java que convierta números del sistema decimal al sistema binario, dando el número decimal como dato.

Unidad 4: La interfaz gráfica de usuario

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.

La interfaz gráfica de usuario

Graphical user interface, GUI

Hasta ahora hemos visto programas que, al compilarlos, se muestran en consola y reciben datos por el teclado, tal como se hacía en los primeros tiempos de la computación. Una aplicación de este tipo se muestra a continuación.

Main.java

```
1  import java.util.Scanner;
2  public class Main
3  {
4      public static void main(String[] args) {
5      Scanner entrada = new Scanner(System.in);
6
7      String nombre;
8      int sueldo;
9      int empleado;
10     int añodenacimiento;
11     int piso;
12 System.out.println("Ingresa tu nombre");
13 nombre =entrada.nextLine();
14
```

Input

```
Ingresa tu nombre
Salvador
```

Figura 4.1 Programa que pide datos por la consola.

La interfaz gráfica de usuario (o GUI, del inglés graphical user interface) es una manera más amigable de interactuar y de mostrar datos, pues contamos con componentes como botones, ventanas, menús, campos de texto, entre otros, tal como se muestra en la

siguiente imagen. En la presente unidad veremos los elementos más importantes para crear una GUI.

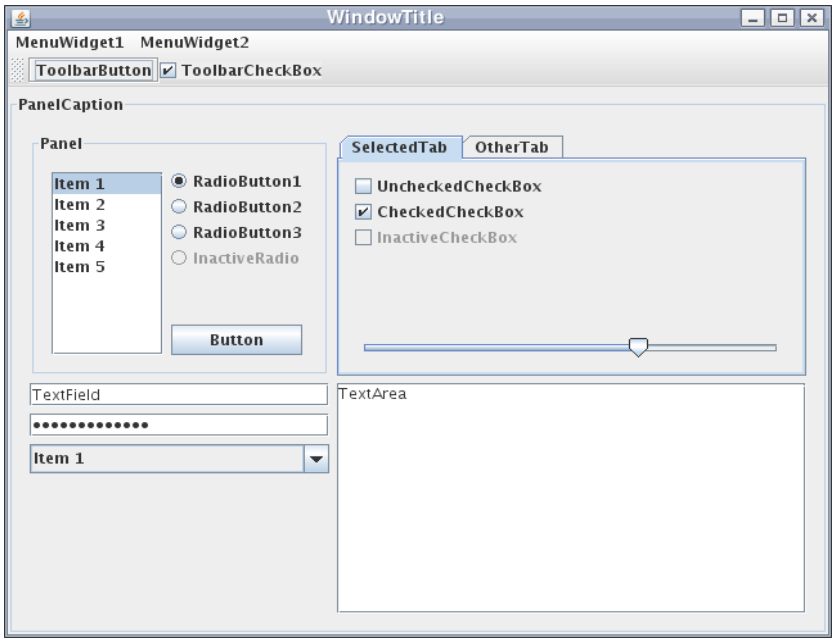


Figura 4.2 Ejemplo de una GUI. Imagen de Wikipedia Commons.

Awt como antecedente de la clase Swing

En sus primeras versiones, Java usaba el paquete *Abstracción Window Toolkit* o AWT, para crear una interfaz de usuario. Posteriormente, en la versión Java SE 1.2, se introdujo la biblioteca Swing, que es la que se utiliza actualmente y que ha reemplazado por completo al AWT. Sin embargo, hay que recordar que AWT es la base de Swing, sobre todo por que las clases de la primera siguen existiendo, y es un aspecto a tomar en cuenta, pues las sentencias de los dos paquetes son muy parecidas y pueden coexistir en un mismo programa; así pues, se puede tener por error una sentencia en AWT y las demás en Swing, y el programa se compila, pero su comportamiento no será el adecuado.

Debido a que en la actualidad la gran mayoría de los programas con GUI se realizan con la clase Swing, es la que se recomienda y la única que se abordará en el presente curso. Para llamar a la totalidad de las clases del paquete Swing se usa la siguiente sentencia al inicio del programa.

```
import javax.swing.*;
```

Marco, etiqueta y botón

Marco o clase JFrame

El primer objeto que aprenderemos a crear es el marco, pues será el contenedor de los botones, etiquetas y otros elementos que le darán interacción y visibilidad a nuestra aplicación. Para esto debemos heredar de la clase JFrame. Si tenemos una clase que se llama EjemploMarco, y queremos hacer un marco, debemos escribir la siguiente línea de código.

```
public class ClaseMarco extends JFrame
```

Por defecto, un marco no se ve, para hacerlo visible, hay que agregar la siguiente sentencia.

```
objetoMarco.setVisible(true);
```

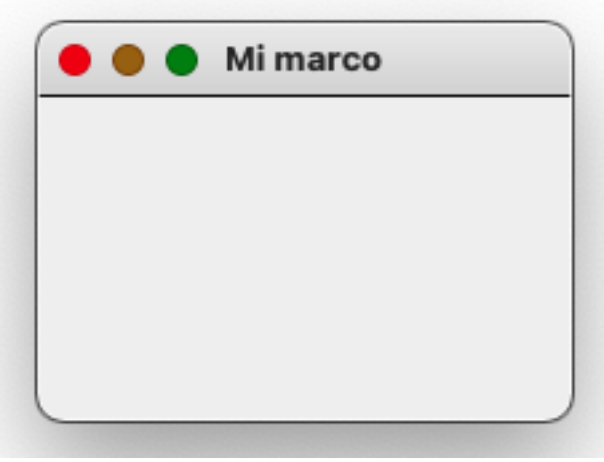


Figura 4.3. Ejemplo de un marco o *frame*.

Para personalizar nuestro contenedor podemos usar algunos de los siguientes métodos.

Método	Función
setTitle()	Coloca un título en el marco
setIconImage()	Coloca un icono en el marco
setDefaultCloseOperation()	Indica la operación a realizarse cuando se cierra el marco.

setVisible()	Hace visible al marco.
setSize()	Especifica una dimensión al marco.
setLocation()	Especifica una posición al marco.
setBounds()	Especifica tamaño y posición del marco.
add()	Para añadir componentes al marco.
setLayout()	Establece la disposición de los componentes dentro del marco.

Observa el siguiente video, en el que comenzaremos a elaborar una primera interfaz gráfica de usuario; utilizaremos la clase JFrame para crear un marco en el que colocaremos todos los elementos que veremos más adelante.

https://bit.ly/3wbPV9e

Video 1. Mi primera interfaz gráfica de usuario (Parte I). Autor: Salvador Gómez Moya

Etiquetas

Una etiqueta es un componente que muestra texto, es inerte porque no tiene eventos asociados. Las etiquetas pertenecen a la clase JLabel; así pues, su constructor es JLabel(), y si dentro de los paréntesis se coloca un texto, éste aparecerá en la etiqueta. En la siguiente imagen se muestra el resultado de usar JLabel("Hola mundo").

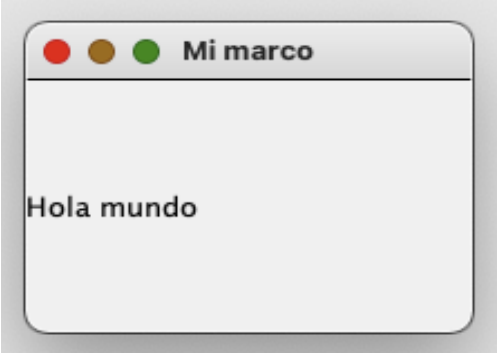


Figura 4.4. La etiqueta “Hola mundo” dentro de un marco.

Algunos métodos de las etiquetas son los siguientes.

Método	Función
getText()	Devuelve el texto que contiene la etiqueta.
setText()	Se establece un texto para la etiqueta.
setBounds()	Se establece una posición y tamaño a la etiqueta

Botones

Java cuenta con muchos tipos de botones, sin embargo, el más común y en el cual nos centraremos en este curso, es el que se crea con la clase JButton. En la siguiente imagen se muestra un botón de esta clase.

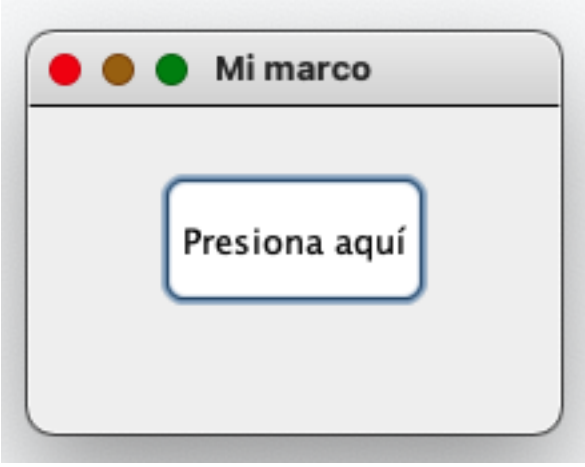


Figura 4.5. Botón de la clase JButton con la leyenda “Presiona aquí”.

Algunos métodos de los botones son los siguientes.

Método	Función
getText()	Devuelve el texto que contiene el botón.
setText()	Se establece un texto para el botón.
setBounds()	Se establece una posición y tamaño al botón.

isSelected()	Devuelve un true si el botón presionó.
setSelected()	Se selecciona el botón.

A diferencia de la etiqueta, un botón puede tener asociados eventos que se ejecutan cuando se presiona. Esto se realiza por medio del siguiente método. Por ejemplo, si se tiene un botón llamado btn, su *listener* u *oyente*, quedaría:

```
btn.addActionListener(this));
```

El método addActionListener(), del inglés *escucha*, se llama así porque queda a la espera o presta atención a que el usuario presione el botón para desencadenar algún evento. Para usar el listener del botón, es necesario implementar la clase ActionListener, el cual requiere que se sobrecargue el método actionPerformed(), el cual a su vez contendrá todas la instrucciones que se ejecutarán cuando se da click al botón.

El siguiente video, es la segunda parte de la primera interfaz gráfica de usuario que comenzamos a elaborar anteriormente; ahora le agregaremos los elementos vistos hasta este momento: la etiqueta y el botón.

https://bit.ly/3px8Ebi

Video 2. Mi primer interfaz gráfica de usuario (Parte II). Autor: Salvador Gómez Moya

Campo de texto

El campo de texto es un espacio donde el usuario introduce una línea de texto. Este componente corresponde a la clase JTextField. Es útil, por ejemplo, para pedir un dato o una cantidad al usuario. A continuación se muestra la imagen de un campo de texto.

Figura 4.6. Campo de texto creado con JTextField.

Algunos métodos de los campos de texto son los siguientes.

Método	**Función**
getText()	Devuelve el texto que contiene el campo.
setText()	Se establece un texto para el campo.
setBounds()	Se establece una posición y tamaño al campo.
setEditable()	Con un argumento booleano podemos especificar si el campo se puede editar o no.
setFont()	Se elige la fuente.
setHorizontalAlignment()	Para alinear el texto.

En el siguiente video, terminamos de diseñar toda nuestra primera interfaz gráfica de usuario, para posteriormente empezar a darle funcionalidad.

https://bit.ly/3AuwcUJ

Video 3. Mi primer interfaz gráfica de usuario (Parte III). Autor: Salvador Gómez Moya

Ejemplo 4.1

A continuación, mostramos el código de la primera interfaz de usuario que hicimos en los videos anteriores.

```
import javax.swing.*;
public class Marco extends JFrame{
	JLabel et;
	JButton btn;
	JTextField entrada;
	JTextArea area;
	Marco(){
	//Propiedades del marco
	setLayout(null);
	setTitle("Mi marco");
	setBounds(300,300,300,300);
	//Etiqueta
	et = new JLabel("Hola mundo");
	et.setBounds(20,20,100,30);
	add(et);
	//Botón
	btn = new JButton("Presione aquí");
	btn.setBounds(20,70,100,40);
	add(btn);
	//Campo de texto
	entrada = new JTextField();
	entrada.setBounds(20,120,100,40);
	add(entrada);
	//Area de texto
	area = new JTextArea("Escribe aquí");
	area.setBounds(20,170,120,70);
	add(area);
	}

	public static void main(String[] args) {
	Marco miMarco = new Marco();
	miMarco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	miMarco.setVisible(true);
```

```
        }
}
```

Aplicaciones prácticas de una GUI

Ya que sabes cómo se construye una interfaz gráfica de usuario, ahora veremos algunas aplicaciones prácticas. Como ejemplo, en el siguiente video te presentamos cómo hacer una GUI que convierte los grados centígrados que ingresa el usuario a grados Fahrenheit. En esta primera parte (de dos), colocaremos todos los componentes de la aplicación, para posteriormente darle funcionalidad.

https://bit.ly/3K3whSj

Video 4. Aplicación que convierte de grados Celsius a grados Fahrenheit (Parte I).
Autor: Salvador Gómez Moya

En esta segunda parte del video, veremos cómo darle interacción a nuestra aplicación que convierte de grados Celsius a grados Fahrenheit, es decir, veremos cómo desencadenar eventos a partir de darle click a un botón.

https://bit.ly/3wbjLdQ

Video 5. Aplicación que convierte de grados Celsius a grados Fahrenheit (Parte II).

Autor: Salvador Gómez Moya

Ejemplo 4.2

A continuación, se muestra el código de la aplicación que convierte de grados Celsius a grados Fahrenheit.

```
import java.awt.Color;
import javax.swing.*;
import java.awt.event.*;


public class Convertidor extends JFrame implements ActionListener{
      JLabel et, resul;
      JTextField entrada;
      JButton btn;
      Float celcius,fahre;
      Convertidor(){
      setLayout(null);
      setTitle("Convertidor de Celcius a Fahrenheit");
      setBounds(300,300,400,200);
      //Etiqueta de instrucciones
      et = new JLabel("Ingresa los grados C para convertir a grados
F");
      et.setBounds(20,15,350,30);
      add(et);
      //Entrada de datos
      entrada = new JTextField();
      entrada.setBounds(50,60,150,30);
```

```
        add(entrada);
        //Resultado
        resul = new JLabel();
          resul.setBounds(220,60,100,30);
        resul.setOpaque(true);
        resul.setBackground(Color.WHITE);
        add(resul);
        //Botón
        btn = new JButton("Convertir");
        btn.setBounds(50,110,100,30);
        btn.addActionListener(this);
        add(btn);

}

        public static void main(String[] args) {
        Convertidor marco = new Convertidor();
        marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        marco.setVisible(true);
        }

        @Override
        public void actionPerformed(ActionEvent e) {
        celcius= Float.parseFloat(entrada.getText());
        fahre=(celcius*9/5)+32;
        resul.setText(fahre+" F");
        }

}
```

Actividad 4.1

Elabora una GUI que pida una cantidad en dólares y devuelva su conversión a pesos. Con otro campo de texto, otro botón y otra etiqueta, que pida una cantidad en pesos y lo convierta a dólares. ¿Se podría hacer con un solo botón?

Mientras navegamos por internet, es común encontrarnos con formularios, como los que llenamos para registrarnos en alguna página; el siguiente video, es la primera parte de una aplicación en la que te mostraremos cómo hacer un formulario de este tipo. Comenzaremos por elementos que ya conoces.

https://bit.ly/3c4C25S

Video 6. Cómo hacer un formulario con una interfaz gráfica de usuario con Java (Parte I).
Autor: Salvador Gómez Moya

Área de texto

El área de texto, al igual que el campo de texto, es un componente para ingresar cadenas de caracteres, la diferencia es que en este caso no son una, sino varias líneas de texto. Este componente corresponde a la clase JTextArea, y es útil, por ejemplo, cuando se pide al usuario ingresar su dirección. En la siguiente imagen se muestra un área de texto.

Figura 4.7. Área de texto creada con JTextField.

Un componente JTextArea tiene los mismos métodos que JTextField mencionados anteriormente; algunos más se muestran a continuación.

Método	Función
setColumns()	Se establece el ancho de cada línea.
append()	Añade un texto al final del documento.
getLineCount()	Determina el número de líneas contenidas en el área de texto.

Lista desplegable

Una lista desplegable se muestra como un botón, pero al hacer clic aparecen varias opciones a elegir. Este elemento corresponde a la clase JComboBox; se usan cuando se pide al usuario que escoja solo una de varias opciones predefinidas, por ejemplo, el semestre en el que está inscrito. Y es que si se usa un campo de texto, el usuario podría escribir, por ejemplo: 6, o sexto, 6to, sexto semestre… Las opciones son infinitas si tomamos en cuenta que la respuesta puede estar mal escrita. En estos casos se recomienda usar una lista desplegable. En la siguiente imagen se muestra una lista desplegable.

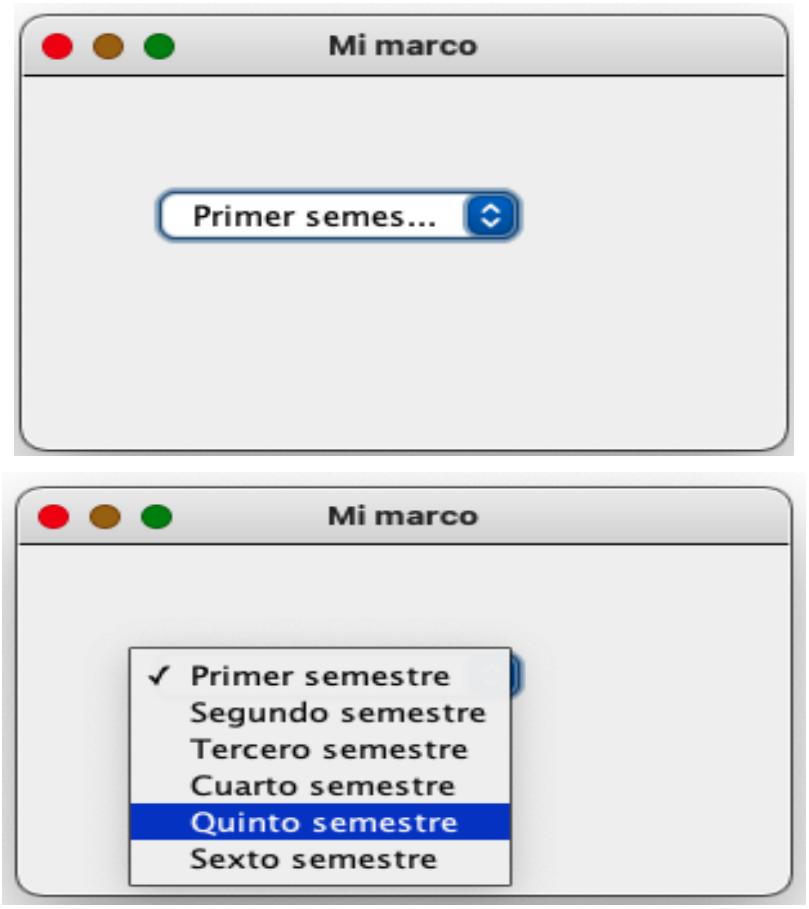


Figura 4.8. Lista desplegable creada con JComboBox.

A continuación se muestran algunos de los métodos del componente JComboBox.

Método	Función
setBounds()	Se establece una posición y tamaño de la lista.
addItem()	Agrega un elemento a la lista.
setMaximumRowCount()	Pone un máximo de elementos a mostrar, para ver el resto muestra un scroll.

En el siguiente video, agregaremos una lista desplegable y dos campos de texto al formulario que empezamos anteriormente.

https://bit.ly/3SXzI15

Video 7. Cómo hacer un formulario con una interfaz gráfica de usuario con Java (Parte II).
Autor: Salvador Gómez Moya

Al igual que el botón, una lista desplegable puede tener asociados eventos que se ejecutan cuando se selecciona una de las opciones. Por ejemplo, si se tiene una lista desplegable llamada lista, su *listener* u *oyente*, quedaría:

lista.addItemListener(this);

El método addItemListener(), queda a la espera o presta atención a que el usuario presione y elija una opción de la lista para desencadenar algún evento. Para usar el listener, es necesario implementar la clase ItemListener, la cual requiere que se sobrecargue el método itemStateChanged(); este método contendrá todas las instrucciones que se ejecutarán cuando se elige una opción de la lista.

En el siguiente video, veremos cómo agregarle funcionalidad a la lista desplegable, esto es, usando su listener.

https://bit.ly/3QyVFSA

Video 8. Cómo hacer un formulario con una interfaz gráfica de usuario con Java (Parte III).
Autor: Salvador Gómez Moya

Ha llegado el momento de finalizar la aplicación del formulario; en el siguiente video, le daremos funcionalidad al botón para que se procese la información de los campos de texto, el área de texto y la lista desplegable, y se nos presente un resultado.

https://bit.ly/3K4Dbqp

Video 9. Cómo hacer un formulario con una interfaz gráfica de usuario con Java (Parte III).
Autor: Salvador Gómez Moya

Ejemplo 4.3

A continuación, se muestra el código del formulario realizado anteriormente.

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
```

```
import javax.swing.*;

public class Beca extends JFrame implements ActionListener,
ItemListener{

      JLabel et,nombre_et, promedio_et, direccion_et, semestre_et;

      JTextField nombre, promedio;

      JTextArea direccion, mensaje;

      JComboBox semestre_lista;

      JButton enviar;

      String semestre_elegido;

      Float prom;


      Beca(){

      setLayout(null);

      setTitle("Formulario para alumnos");

      setBounds(300,300,400,600);

      //Indicación

      et = new JLabel("Completa la siguiente información");

      et.setBounds(20,15,350,30);

      add(et);

      //Nombre

      nombre_et = new JLabel("Nombre completo:");

      nombre_et.setBounds(20,50,200,30);

      add(nombre_et);

      nombre = new JTextField();

      nombre.setBounds(150,50,200,30);

      add(nombre);

      //Promedio

      promedio_et = new JLabel("Promedio general:");

      promedio_et.setBounds(20,90,200,30);

      add(promedio_et);

      promedio = new JTextField();

      promedio.setBounds(150,90,50,30);

      add(promedio);
```

```
        //Dirección
        direccion_et = new JLabel("Escribe tu dirección:");
        direccion_et.setBounds(20,140,200,30);
        add(direccion_et);
        direccion = new JTextArea();
        direccion.setBounds(20,180,200,50);
        add(direccion);
        //Semestre
        semestre_et = new JLabel("Selecciona el semestre:");
        semestre_et.setBounds(20,250,200,30);
        add(semestre_et);
        semestre_lista = new JComboBox();
        semestre_lista.setBounds(20,290,200,30);
        semestre_lista.addItem("");
        semestre_lista.addItem("Primer semestre");
        semestre_lista.addItem("Segundo semestre");
        semestre_lista.addItem("Tercer semestre");
        semestre_lista.addItemListener(this);
        add(semestre_lista);
        //Botón
        enviar = new JButton("Enviar información");
        enviar.setBounds(120,350,150,40);
        enviar.addActionListener(this);
        add(enviar);
        //Mensaje final
        mensaje = new JTextArea();
        mensaje.setBounds(50,430,280,100);
        add(mensaje);

        }

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

```
	Beca marco = new Beca();
	marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	marco.setVisible(true);
	}

	@Override
	public void actionPerformed(ActionEvent e) {
	prom = Float.parseFloat(promedio.getText());
	if(prom>8 && semestre_elegido.equals("Tercer semestre")){
		mensaje.setText("El alumno "+nombre.getText()+" obtuvo la
\nbeca,la cual se enviará a la dirección:\n"+direccion.getText());
	}else{
		mensaje.setText("El alumno"+nombre.getText()+"\nno obtuvo
la beca");
	}
	}

	@Override
	public void itemStateChanged(ItemEvent e) {
	semestre_elegido = (String)semestre_lista.getSelectedItem();
	}

}
```

Casillas de verificación

Una casilla de verificación es un elemento que puede ser seleccionado o deseleccionado. En general se suelen colocar varias casillas y el usuario puede seleccionar una, varias o todas las opciones, como se muestra en la siguiente imagen. Se usa la clase JCheckBox para crear objetos de este tipo.

Figura 4.9. Casillas de verificación creadas con JCheckBox.

RadioButton

El RadioButton es un elemento que al igual que la casilla de verificación, puedes seleccionar o deseleccionar; la diferencia radica en que si se tienen varios elementos de este tipo, solo uno puede estar seleccionado. Para lograr este funcionamiento en conjunto, además de crear objetos de la clase RadioButton, se crea un objeto de la clase ButtonGroup. En la siguiente imagen se muestra un grupo de RadioButtons.

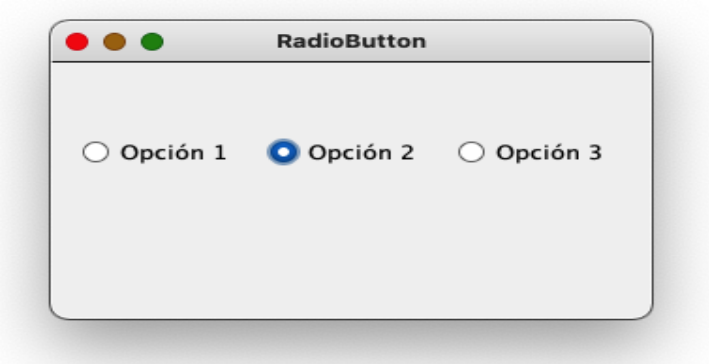


Figura 4.10. Tres RadioButtons; solo uno puede estar seleccionado.

En el siguiente video, retomaremos el formulario que pedía los datos de un alumno y le otorgaba una beca dependiendo de su promedio y semestre, pero ahora le agregaremos algunos RadioButtons y algunas casillas de verificación.

https://bit.ly/3AvfCEa

Video 10. Cómo agregar un conjunto de RadioButtons a la GUI.

Autor: Salvador Gómez Moya

Ya que vimos cómo declarar y mostrar un conjunto de RadioButtons y algunas casillas de verificación, a continuación te mostramos algunos de sus métodos; comenzamos por los de la clase JCheckBox.

Método	Función
add()	Para agregar la casilla al marco.
setBounds()	Se establece una posición y tamaño a la casilla.
isSelected()	Para conocer el estado de la casilla.

La casilla de verificación tiene un funcionamiento similar al del botón, de hecho los dos componentes heredan de la misma clase AbstractButton. Su listener, si hubiera un objeto de JCheckBox que se llamara *casilla*, sería el siguiente:

```
casilla.addChangeListener(this);
```

Este listener llama al método stateChanged(), en el cual se deben colocar las instrucciones que se ejecutarán cuando se selecciona la casilla.

A continuación, se muestran algunos de los métodos de la clase JRadioButton.

Método	Función
add()	Para agregar el RadioButton al marco.
setBounds()	Se establece una posición y tamaño al RadioButton.
isSelected()	Para conocer el estado del RadioButton.

El RadioButton, como la casilla de verificación, tiene un funcionamiento similar al de un botón. Tiene el mismo listener que la casilla de verificación, es decir:

```
addChangeListener(this);
```

y por lo tanto, se llama al mismo método stateChanged().

En el siguiente video te mostramos cómo agregar los listener a los RadioButtons y a las casillas de verificación.

https://bit.ly/3ppiCM7

Video 11.Cómo agregar los listener a los RadioButtons y a las casillas de verificación.

Autor: Salvador Gómez Moya

Ya que agregamos los listener, en el siguiente vídeo veremos cómo agregar funcionalidad al cambio de estado de los RadioButtons, es decir, cuando le damos click a uno de estos elementos.

https://bit.ly/3bX8UO1

Video 12. Cómo agregar funcionalidad al cambio de estado de los RadioButtons.

Autor: Salvador Gómez Moya

Para finalizar el formulario, solo nos falta agregar funcionalidad al cambio de estado de las casillas de verificación, es decir cuando las seleccionamos o deseleccionamos.

https://bit.ly/3CeB8OX

Video 13. Cómo agregar funcionalidad al cambio de estado en las casillas de verificación.

Autor: Salvador Gómez Moya

Ejemplo 4.4

El código del formulario ya terminado se muestra a continuación.

```
import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.awt.event.ItemEvent;

import java.awt.event.ItemListener;

import javax.swing.*;

import javax.swing.event.ChangeEvent;

import javax.swing.event.ChangeListener;

public class Beca extends JFrame implements ActionListener, ItemListener,
ChangeListener{

      JLabel et,nombre_et, promedio_et, direccion_et, semestre_et, regular_et,
equipos_et;

      JTextField nombre, promedio;
```

```
JTextArea direccion, mensaje;
JComboBox semestre_lista;
JButton enviar;
String semestre_elegido, equipos;
Float prom;
JRadioButton regular, no_regular;
ButtonGroup bg;
JCheckBox laptop, tablet, calculadora;
Boolean status;

Beca(){
setLayout(null);
setTitle("Formulario para alumnos");
setBounds(300,300,600,600);
//Indicación
et = new JLabel("Completa la siguiente información");
et.setBounds(20,15,350,30);
add(et);
//Nombre
nombre_et = new JLabel("Nombre completo:");
nombre_et.setBounds(20,50,200,30);
add(nombre_et);
nombre = new JTextField();
nombre.setBounds(150,50,200,30);
add(nombre);
//Promedio
promedio_et = new JLabel("Promedio general:");
promedio_et.setBounds(20,90,200,30);
add(promedio_et);
promedio = new JTextField();
promedio.setBounds(150,90,50,30);
add(promedio);
//Dirección
```

```
direccion_et = new JLabel("Escribe tu dirección:");
direccion_et.setBounds(20,140,200,30);
add(direccion_et);
direccion = new JTextArea();
direccion.setBounds(20,180,200,50);
add(direccion);
//Semestre
semestre_et = new JLabel("Selecciona el semestre:");
semestre_et.setBounds(20,250,200,30);
add(semestre_et);
 semestre_lista = new JComboBox();
semestre_lista.setBounds(20,290,200,30);
semestre_lista.addItem("");
semestre_lista.addItem("Primer semestre");
semestre_lista.addItem("Segundo semestre");
semestre_lista.addItem("Tercer semestre");
semestre_lista.addItemListener(this);
add(semestre_lista);
//Botón
enviar = new JButton("Enviar información");
enviar.setBounds(230,350,150,40);
enviar.addActionListener(this);
add(enviar);
//Mensaje final
mensaje = new JTextArea();
mensaje.setBounds(110,430,380,100);
add(mensaje);
//RadioButtons
regular_et = new JLabel("¿Eres alumno regular?");
regular_et.setBounds(260,140,200,20);
add(regular_et);
regular = new JRadioButton("Sí");
regular.setBounds(280,170,50,20);
```

```
        regular.addChangeListener(this);
        add(regular);
        no_regular = new JRadioButton("No");
        no_regular.setBounds(330,170,50,20);
        no_regular.addChangeListener(this);
        add(no_regular);
        bg = new ButtonGroup();
        bg.add(regular);
        bg.add(no_regular);
        //Casillas de verificación
        equipos_et = new JLabel("¿Cuáles de los siguientes equipos NO cuentas?");
        equipos_et.setBounds(260,210,330,30);
        add(equipos_et);
        laptop = new JCheckBox("Laptop");
        laptop.setBounds(260,240,80,20);
        laptop.addChangeListener(this);
        add(laptop);
        tablet = new JCheckBox("Tablet");
        tablet.setBounds(340,240,80,20);
        tablet.addChangeListener(this);
        add(tablet);
        calculadora = new JCheckBox("Calculadora");
        calculadora.setBounds(420,240,110,20);
        calculadora.addChangeListener(this);
        add(calculadora);




        }


        public static void main(String[] args) {
        Beca marco = new Beca();
        marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        marco.setVisible(true);
        }

        @Override
        public void actionPerformed(ActionEvent e) {
        prom = Float.parseFloat(promedio.getText());
        if(prom>8 && semestre_elegido.equals("Tercer semestre")&&status==true){
              mensaje.setText("El alumno "+nombre.getText()+" obtuvo la
\nbeca,la cual se enviará a la dirección:\n"+direccion.getText()+"\n"+equipos);
        }else{
              mensaje.setText("El alumno"+nombre.getText()+"\nno obtuvo la
beca");
        }
        }

        @Override
        public void itemStateChanged(ItemEvent e) {
        semestre_elegido = (String)semestre_lista.getSelectedItem();
        }

        @Override
        public void stateChanged(ChangeEvent e) {
        if(regular.isSelected()) status = true;
         else status = false;

        equipos = "También recibirá los sig. equipos: ";
        if(laptop.isSelected()) equipos+="laptop/";
        if(tablet.isSelected()) equipos+="tablet/";
        if(calculadora.isSelected()) equipos+="calculadora";

        }

}
```

Actividad 4.2

Ahora que ya sabes cómo funcionan la mayoría de los elementos básicos de una interfaz gráfica de usuario, **elabora una aplicación que pida ciertos datos** (que incluya un área de texto, una lista desplegable, unas casillas de verificación o unos radiobuttons), además de una contraseña; si la contraseña es correcta, aparecerá un mensaje de bienvenida con los otros datos que se solicitaron.

Menú de opciones

Un menú como el que se muestra en la imagen, proporciona una manera de organizar y ofrecer varias opciones al usuario, buscando ahorrar espacio. Se utilizan tres clases para crear un menú de este tipo. Con JMenuBar se crea un objeto que es una especie de contenedor, al cual se le agregan objetos del tipo JMenu, que corresponden a las opciones principales, en el caso de la imagen de abajo, serían Sección 1 y Sección 2. Con JMenuItem se agregan las alternativas pertenecientes a la opciones principales, en el caso de la imagen, serían Subsección 1-1, 1-2, 1-3 y 1-4.

Figura 4.11. Menú de opciones creado con JMenuBar, JMenu y JMenuItem.

En el siguiente video comenzaremos un proyecto en el que veremos cómo crear una ventana y cómo agregarle las opciones principales de un menú.

https://bit.ly/3JZBieR

Video 14. Cómo agregar un menú principal a una ventana.
Autor: Salvador Gómez Moya

A continuación, se muestran algunos de los métodos de las clases JMenuBar, JMenu y JMenuItem.

Método	Función
add()	Para agregar opciones al menú.
addSeparator()	Se agrega una línea para separar las opciones.
setMenuBar()	Pone el menú para que aparezca en el marco.

En el siguiente video veremos cómo crear los submenús de cada una de las opciones principales; también usaremos algunos de los métodos mostrados en la tabla anterior.

https://bit.ly/3psxDwC

Video 15. Cómo agregar los submenús a las opciones principales.
Autor: Salvador Gómez Moya

Las opciones de un menú son como botones; en el video anterior ya agregamos todas las opciones y les agregamos su listener. En el siguiente video veremos cómo desencadenar eventos cada vez que le damos click a una de estas opciones.

https://bit.ly/3A39Fgb

Video 16. Cómo desencadenar eventos cada vez que le damos click a un menú de opciones. Autor: Salvador Gómez Moya

Ejemplo 4.5

A continuación, se muestra el código del menú de opciones realizado anteriormente.

```
package menu;

import java.awt.Color;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.*;


public class Menu extends JFrame implements ActionListener{

	JLabel et;

	JMenuBar menuBar;

	JMenu menu;

	JMenuItem fondo_azul, fondo_rojo, fuente_blanca, fuente_negra, arial_18,
times_36;


	Menu(){

	setLayout(null);

	setTitle("Menú de opciones");

	setBounds(300,300,400,400);

	//Etiqueta

	et = new JLabel("Hola mundo");

	et.setBounds(150,150,170,50);

	et.setOpaque(true);

	add(et);

	//Menú

	menuBar = new JMenuBar();


	menu = new JMenu("Colores");

	menuBar.add(menu);

	fondo_azul = new JMenuItem("Fondo azul");

	fondo_azul.addActionListener(this);

	menu.add(fondo_azul);

	fondo_rojo = new JMenuItem("Fondo rojo");

	fondo_rojo.addActionListener(this);
```

```
        menu.add(fondo_rojo);
        menu.addSeparator();

        fuente_blanca = new JMenuItem("Fuente blanca");
        fuente_blanca.addActionListener(this);
        menu.add(fuente_blanca);
        fuente_negra = new JMenuItem("Fuente negra");
        fuente_negra.addActionListener(this);
        menu.add(fuente_negra);



        menu = new JMenu("Fuente");
        menuBar.add(menu);
        arial_18 = new JMenuItem("Arial 18");
        arial_18.addActionListener(this);
        menu.add(arial_18);
        times_36 = new JMenuItem("Times 36");
        times_36.addActionListener(this);
        menu.add(times_36);

        setJMenuBar(menuBar);

        }



        public static void main(String[] args) {

        Menu marco = new Menu();
        marco.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        marco.setVisible(true);

        }
```

```
        @Override
        public void actionPerformed(ActionEvent e) {
        Font arial = new Font("Arial",Font.ITALIC,18);
        Font times = new Font("Times",Font.BOLD,36);



        if(e.getSource()==fondo_azul) et.setBackground(Color.BLUE);
        if(e.getSource()==fondo_rojo) et.setBackground(Color.red);
        if(e.getSource()==fuente_blanca) et.setForeground(Color.WHITE);
        if(e.getSource()==fuente_negra) et.setForeground(Color.BLACK);


        if(e.getSource()==arial_18) et.setFont(arial);
        if(e.getSource()==times_36) et.setFont(times);


        }

}
```

Ya que revisamos todos los elementos que componen una interfaz gráfica de usuario, relaciona algunos de los métodos que hemos visto con su funcionamiento en la siguiente actividad.

Actividad 4.3

Coloca los siguientes métodos donde corresponda.

add()

setFont()

setTitle()

append()

isSelected()

setBounds()

addSeparator()

addItem()

setText()

getText()

Muestra un título en la barra superior de la ventana.	
Le otorga una posición y una dimensión al objeto	
Para añadir componentes al marco.	
Se establece un texto para la etiqueta.	
Devuelve el texto que contiene la etiqueta.	
Devuelve un true si el botón presionó.	
Se elige la fuente.	
Añade un texto al final del documento.	
Agrega elemento a la lista.	
Se agrega una línea para separar las opciones.	

La clase Graphics

Con la clase Graphics podemos dibujar figuras regulares como líneas, cuadrados, círculos, entre otras. Esta clase se encuentra dentro del paquete awt, por lo tanto debemos declarar la siguiente instrucción al inicio del programa.

```
import java.awt.Graphics;
```

Una vez que se llama a la clase Graphics, el siguiente paso necesario es sobrecargar uno de sus métodos, el método paint(), como se muestra a continuación.

```
@Override
public void paint(Graphics g){

}
```

Como se observa, dentro del paréntesis del método, se crea un objeto de la clase Graphics; este objeto contiene los métodos para dibujar y darle color a las figuras.

En el siguiente video te mostramos como crear un lienzo con la clase Graphics, con el cual podremos hacer nuestros primeros dibujos.

https://bit.ly/3QUYs8b

Video 17. Creando un lienzo con la clase Graphics. Autor: Salvador Gómez Moya

La línea

Para dibujar una línea, usamos el objeto *g* antes mencionado y llamamos a su método drawLine(), el cual recibe cuatro parámetros, el primero es la coordenada en *x* de donde inicia la línea, el segundo es lo mismo pero en *y*; el tercer parámetro es la coordenada en *x* de donde termina la línea, la cuarta es lo mismo pero en *y*.

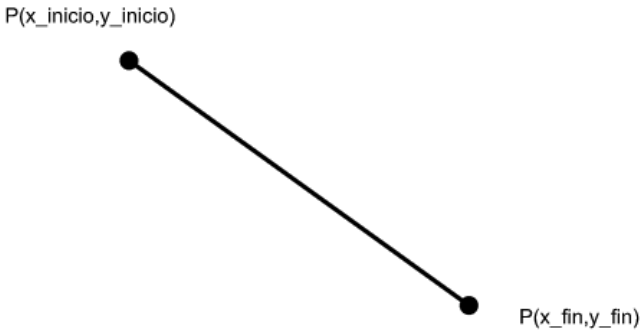


Figura 4.12. La línea necesita cuatro parámetros para dibujarse con g.drawLine(x_inicio,y_inicio,x_fin_y_fin).

Para darle color a esta línea, así como a todas las figuras que veremos en la presente lección, usaremos el método setColor(), el cual recibe como parámetro un color de la clase Color. Esta instrucción se escribe antes de la figura que se quiere colorear. A continuación, un ejemplo de cómo se le daría color azul a cierta figura.

```
g.setColor(Color.BLUE).
```

El rectángulo

Para dibujar un rectángulo, usamos el método drawRect(), el cual recibe cuatro parámetros, el primero es la ubicación en x de la esquina superior izquierda; el segundo parámetro es lo mismo pero en su coordenada y; el tercer parámetro es su ancho, y el cuarto su alto, en pixeles.

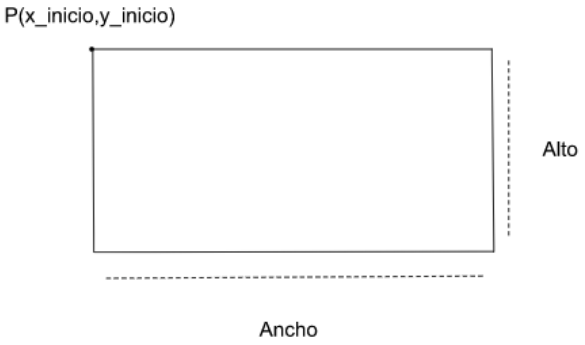


Figura 4.13. El rectángulo necesita 4 parámetros, a saber, g.drawRect(x_inicio, y_inicio,Ancho, Alto).

Si quisieras dibujar un cuadrado, se puede hacer con drawRect(), solo tienes que darle las mismas dimensiones al alto y ancho.

Con el método drawRect() solo se dibuja el contorno del rectángulo; para crear esta figura con un relleno, se usa la siguiente instrucción, que como se observa, contiene los mismos cuatro parámetros de cuando solo se dibuja el contorno.

g.fillRect(x_inicio, y_inicio,Ancho, Alto).

Figura 4.14. Con fillRect() se obtiene el mismo rectángulo, pero con relleno de un color.

También podemos dibujar un rectángulo con las esquinas redondeadas; esto se hace con la instrucción drawRoundRect(), que recibe seis parámetros, los primeros 4 son los mismos que los rectángulos que ya mencionamos anteriormente, los últimos dos corresponden al tamaño en pixeles del redondeo en *x* y *y*; la siguiente imagen muestra esto más claramente.

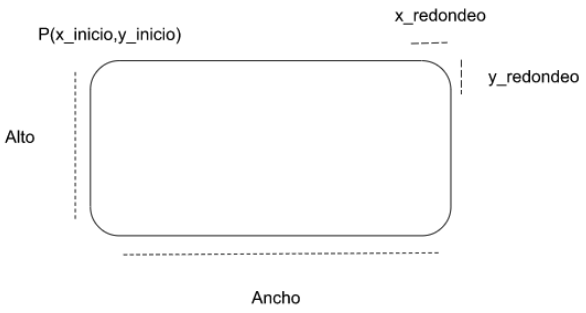


Figura 4.15. Rectángulo con la esquina redondeada hecho con la instrucción g.drawRoundRect(x_inicio, y_inicio, Ancho, Alto, x_redondeo, y_redondeo).

Y si queremos dibujar un rectángulo con las esquinas redondeadas y con relleno, lo hacemos con la siguiente instrucción, que como se observa, contiene los mismos parámetros que la instrucción pasada.

g.fillRoundRect(x_inicio, y_inicio, Ancho, Alto, x_redondeo, y_redondeo).

El óvalo o círculo

Para dibujar un óvalo, usamos el método drawOval(), el cual recibe cuatro parámetros, los dos primeros corresponden a las coordenadas del centro del óvalo en *x* y *y*, respectivamente; el tercer parámetro es el ancho, y el cuarto el alto en píxeles. Si quisiéramos dibujar un círculo, simplemente daríamos la misma cantidad en estos últimos parámetros.

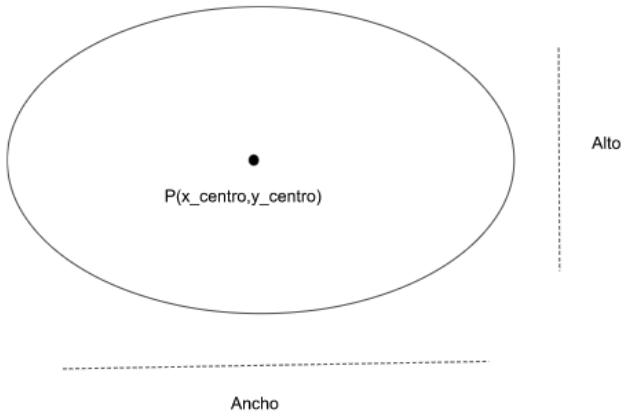


Figura 4.16. Con g.drawOval(x_centro, y_centro, Ancho, Alto) se dibuja un óvalo como el de la imagen.

Para dibujar un óvalo o círculo con relleno, se hace con la siguiente instrucción, observa que solo cambia el nombre del método, pero los parámetros son los mismos.

g.fillOval(x_centro, y_centro, Ancho, Alto).

Polígonos

Hasta ahora hemos visto figuras conocidas, pero ¿qué pasa si queremos dibujar por ejemplo un hexágono o una estrella? Con el método drawPolygon() lo podemos hacer, el

cual recibe parámetros un tanto diferentes a lo visto hasta ahora, y es que si bien recibe tres parámetros, el primero y el segundo son vectores que contienen las coordenadas de los vértices de la figura; el tercer parámetro es para indicar el número de lados. Por su puesto, el número de vértices debe coincidir con el número de lados, si no el programa arrojará un error al momento de compilar.

Así pues, debemos declarar un vector con todas las coordenadas en *x*, y otro vector con todas las coordenadas en *y*. Si se tratara de dibujar un triángulo, el último parámetro recibiría un 3. La siguiente imagen muestra este método para un triángulo.

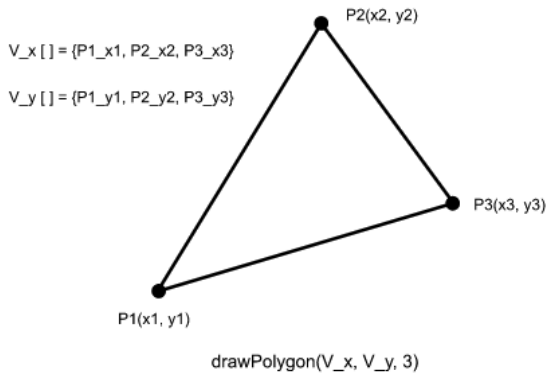


Figura 4.17. Triángulo dibujado con la instrucción drawPolygon(V_x, V_y, 3).

Para dibujar un polígono con relleno, al igual que con los casos anteriores, se usa un método diferente, pero los parámetros son los mismos a los de la figura con solo contornos; esta instrucción se muestra a continuación.

g.fillPolygon(V_x, V_y, n).

Donde n es el número de lados, en el caso de un triángulo como el anterior, sería 3.

Ya que vimos las clases y métodos para crear figuras en Java, te invitamos a ver el siguiente video donde te explicamos paso a paso cómo dibujar todas estas formas geométricas.

https://bit.ly/3Aureap

Video 18. Dibujando figuras regulares. Autor: Salvador Gómez Moya

Para finalizar la presente lección y la unidad, veremos cómo dibujar las mismas figuras pero con relleno de algún color.

https://bit.ly/3pryw8B

Video 19. Dibujando figuras regulares con relleno de algún color. Autor: Salvador Gómez Moya

Ejemplo 4.6

A continuación, se muestra el código de los dibujos realizados anteriormente.

```
import java.awt.Color;
import javax.swing.JFrame;
import java.awt.Graphics;

public class Dibujo extends JFrame{

	Dibujo(){
	setLayout(null);
	setBounds(300,300,800,400);

	}

	public static void main(String[] args) {
	Dibujo lienzo = new Dibujo();
	lienzo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	lienzo.setVisible(true);

	}

	@Override
	public void paint(Graphics g){
```

```
        super.paint(g);
        g.setColor(Color.red);
        g.drawLine(200, 100, 250, 200);
        g.drawOval(50, 100, 100, 100);
        g.drawRect(300, 100, 100, 100);
        g.setColor(Color.BLACK);
        g.drawRoundRect(450, 100, 100, 50, 20, 20);

        int v_x [] = {650,600,650,700};
        int v_y [] = {100,150,200,100};
        g.drawPolygon(v_x,v_y,4);

        g.setColor(Color.BLUE);
        g.fillRect(50, 250, 100, 50);
        g.fillOval(200, 250, 150, 100);
        g.fillRoundRect(400, 250, 150, 100,5, 5);

        int vx_t [] = {600,650,700};
        int vy_t [] = {350,250,350};
        g.fillPolygon(vx_t, vy_t, 3);
        }

}
```

A continuación te mostramos algunas actividades para que demuestres todo lo aprendido en la presente lección.

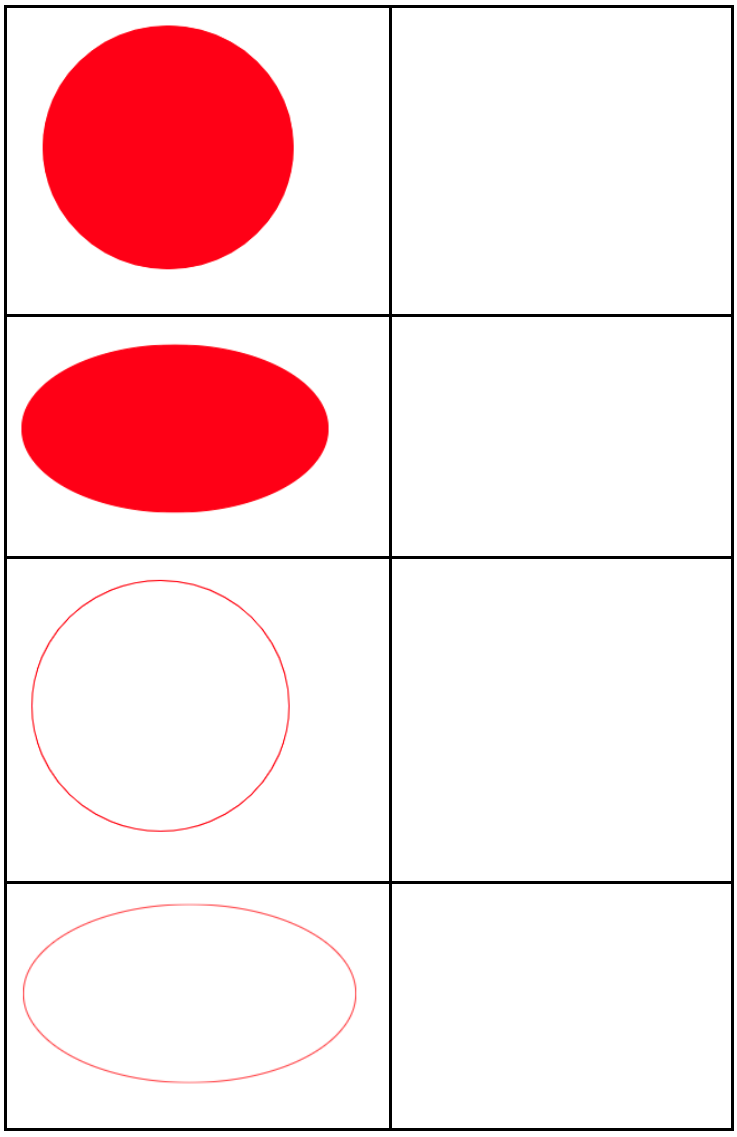
Actividad 4.4

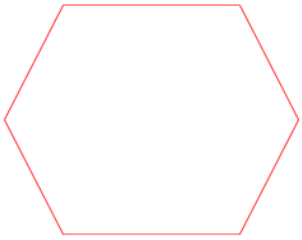
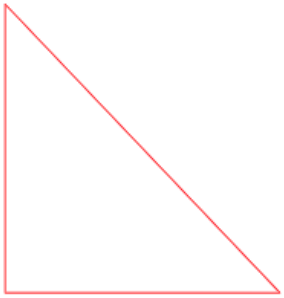
Para esta primera actividad, elige el método para dibujar la figura correspondiente.

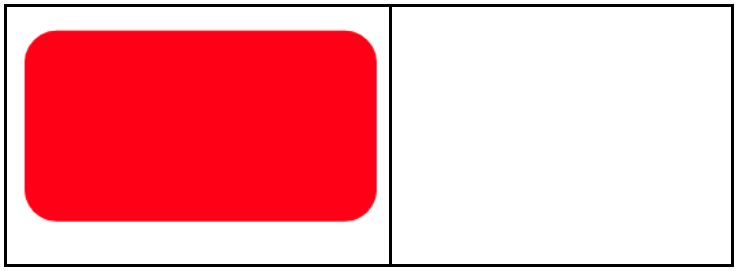
fillRect(100, 100, 150, 100);

fillOval(100, 100, 100, 100);

```
fillRoundRect(100, 100, 150, 100,5, 5)
drawOval(100, 100, 100, 100);
fillRect(100, 100, 100, 100);
drawOval(100, 100, 150, 100);
drawPolygon(v_x,v_y,6);
drawPolygon(v_x,v_y,3);
fillOval(100, 100, 150, 100);
drawRect(100, 100, 100, 100);
```

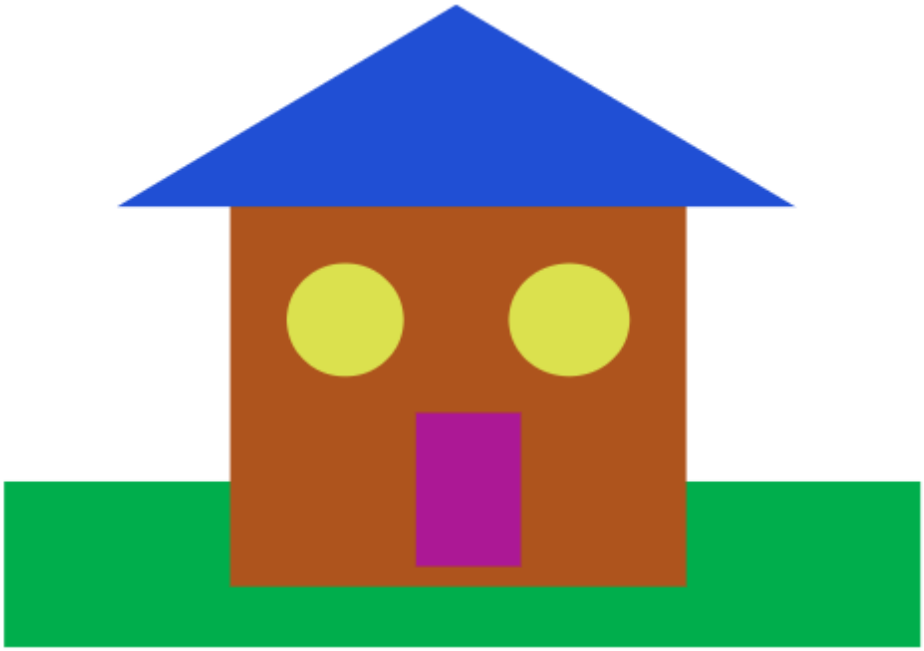




Actividad 4.5

Ahora, con todas las instrucciones y funciones vistas hasta ahora, **realiza el siguiente dibujo en Java**.



Bibliografía

Cairó, O. (2003). Metodología de la programación. Algoritmos, diagramas de flujo y programas. México: Alfaomega.

Ceballos, F. J. (2006). Java: curso de programación. México: Alfaomega.

Deitel, P. (2012). Cómo programar en Java. México: Pearson Educación.

Eckel, B. (2007). Piensa en Java. Madrid: Pearson–Prentice Hall.

Joyanes, L. (2003). Fundamentos de la programación. Algoritmos, estructura de datos y objetos, México: Mc. Graw–Hill. Moisset, D. (2016).

Curso de programación Java [en línea]. Recuperado 27 de julio de 2022 en. http://codigofacilito.com/cursos/JAVA

Pinales Delgado, F. & Velázquez Amador C. (2014). ALGORITMOS RESUELTOS CON Diagramas de flujo y pseudocódigo. México: Universidad Autónoma de Aguascalientes.

Rodríguez, A. Curso aprender programación Java desde cero [en línea]. Recuperado 29 de agosto 2022 en. http://www.aprenderaprogramar/