Clases desarrollo avanzado

PROGRAMACIÓN

Contenidos:

1) Características de la Programación Orientada a Objetos.
2) Los objetos.
3) Encapsulación y visibilidad.
4) Herencia
5) JOptionPane
6) La referencia a null.

Características de la Programación Orientada a Objetos

Las características más importantes del paradigma de la programación orientada a objetos son:

- **Abstracción**: Es el proceso por el cual definimos las características más importantes de un objeto, sin preocuparnos de cómo se escribirán en el código del programa, simplemente lo definimos de forma general. En la Programación Orientada a Objetos la herramienta más importante para soportar la abstracción es la clase. Básicamente, una clase es un tipo de dato que agrupa características comunes de un conjunto de objetos. Poder ver los objetos del mundo real que deseamos trasladar a nuestros programas, en términos abstractos, resulta de gran utilidad para un buen diseño del software, ya que nos ayuda a comprender mejor el problema y a tener una visión global del conjunto. Por ejemplo, si pensamos en una clase Vehículo que agrupa las características comunes de todos ellos, a partir de dicha clase podríamos crear objetos como Coche y Camión. Entonces se dice que Vehículo es una abstracción de Coche y de Camión.
- **Modularidad**: Una vez que hemos representado el escenario del problema en nuestra aplicación, tenemos como resultado un conjunto de objetos software a utilizar. Este conjunto de objetos se crean a partir de una o varias clases. Cada clase se encuentra en un archivo diferente, por lo que la modularidad nos permite modificar las características de la clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.

Características de la Programación Orientada a Objetos

- **Encapsulación**: También llamada "ocultamiento de la información". La encapsulación o encapsulamiento es el mecanismo básico para ocultar la información de las partes internas de un objeto a los demás objetos de la aplicación. Con la encapsulación un objeto puede ocultar la información que contiene al mundo exterior, o bien restringir el acceso a la misma para evitar ser manipulado de forma inadecuada. Por ejemplo, pensemos en un programa con dos objetos, un objeto Persona y otro Coche. Persona se comunica con el objeto Coche para llegar a su destino, utilizando para ello las acciones que Coche tenga definidas como por ejemplo conducir. Es decir, Persona utiliza Coche pero no sabe cómo funciona internamente, sólo sabe utilizar sus métodos o acciones

- **Jerarquía**: Mediante esta propiedad podemos definir relaciones de jerarquías, entre clases y objetos. Las dos jerarquías más importantes son la jerarquía "es un", llamada generalización o especialización, y la jerarquía "es parte de", llamada agregación. Conviene detallar algunos aspectos:
  - ✓ La generalización o especialización, también conocida como herencia, permite crear una clase nueva en términos de una clase ya existente (herencia simple) o de varias clases ya existentes (herencia múltiple). Por ejemplo, podemos crear la clase CochedeCarreras a partir de la clase Coche, y así sólo tendremos que definir las nuevas características que tenga.
  - ✓ La agregación, también conocida como inclusión, permite agrupar objetos relacionados entre sí dentro de una clase. Así, un Coche está formado por Motor, Ruedas, Frenos y Ventanas. Se dice que Coche es una agregación y Motor, Ruedas, Frenos y Ventanas son agregados de Coche.

Los objetos

Un objeto es un conjunto de datos con las operaciones definidas para ellos. Los objetos tienen un estado y un comportamiento.

Los objetos tienen unas características fundamentales que los distinguen:

- **Identidad:** Es la característica que permite diferenciar un objeto de otro. De esta manera, aunque dos objetos sean exactamente iguales en sus atributos, son distintos entre sí. Puede ser una dirección de memoria, el nombre del objeto o cualquier otro elemento que utilice el lenguaje para distinguirlos. Por ejemplo, dos vehículos que hayan salido de la misma cadena de fabricación y sean iguales aparentemente, son distintos porque tienen un código que los identifica.
- **Estado:** El estado de un objeto viene determinado por parámetros o atributos que lo describen y los valores de éstos. Por ejemplo, si tenemos un objeto Coche, el estado estaría definido por atributos como Marca, Modelo, Color, Cilindrada, etc.
- **Comportamiento:** Son las acciones que se pueden realizar sobre el objeto. En otras palabras, son los métodos o procedimientos que realiza el objeto. Siguiendo con el ejemplo del objeto Coche, el el comportamiento serían acciones como: arrancar(), parar(), acelerar(), frenar(), etc.

Los objetos: Constructores

Los constructores son métodos especiales con el mismo nombre de la clase y que no devuelven ningún valor tras su ejecución.
Cuando creamos un objeto debemos instanciarlo utilizando el constructor de la clase., para ello:

```
public class Ejemplo {..}
//En otra clase, dentro de un método main:
Ejemplo obj = new Ejemplo();//Constructor por defecto.
```

Los constructores tienen las siguientes características:

- ✓Se invocan automáticamente solo una vez en la creación de un objeto.
- ✓Los constructores no empiezan con minúscula, como el resto de los métodos, ya que se llaman igual que la clase y las clases empiezan con letra mayúscula.
- ✓Pueden haber varios constructores para una clase: por defecto, por parámetro y por copia.
- ✓Puede tener parámetros para definir qué valores dar a los atributos del objeto.
- ✓El constructor por defecto es aquel que no tiene argumentos o parámetros.
- ✓Toda clase tiene que tener un constructor. En caso contrario, el compilador crea un constructor por defecto vacío, que inicializa los atributos a sus valores por defecto, según del tipo que sean: 0 para los tipos numéricos, false para los boolean y **null** para los *tipo carácter y las referencias*.
- ✓En herencia, el constructor llama la constructor sin argumentos llama al de la superclase; si la superclase no tiene constructor sin argumentos se produce un error de compilación.

Los objetos: Constructores

```
public class Ejemplo
 {
        int a1, a2;
        Ejemplo() //Constructor por defecto.
        {
                this.a1 =-1;
                this.a2 =-1;
        }
        Ejemplo(int a1, int a2) //Constructor por referencia o por valor.
        {
                this.a1 =a1;
                 this.a2 =a2;
        }
        Ejemplo(Ejemplo o1) //Constructor por copia
        {
                this.a1 =o1.a1;
                 this.a2 = o1.a2;
        }
}
```

Los objetos: Interacción entre objetos

Dentro de un programa los objetos se comunican llamando a sus métodos. Los métodos están dentro de los objetos y describen el comportamiento de un objeto cuando recibe una llamada. En otras palabras, cuando un objeto, objeto1, quiere actuar sobre otro, objeto2, tiene que ejecutar uno de sus métodos. Entonces se dice que el objeto2 recibe un mensaje del objeto1.

Un **mensaje** es la acción que realiza un objeto. Un método es la función o procedimiento al que se llama para actuar sobre un objeto. Los distintos mensajes que puede recibir un objeto o a los que puede responder reciben el nombre de **protocolo** de ese objeto.

El proceso de interacción entre objetos se suele resumir diciendo que se ha "enviado un mensaje" (hecho una petición) a un objeto, y el objeto determina "qué hacer con el mensaje" (ejecuta el código del método). Cuando se ejecuta un programa se producen las siguientes acciones:

- Creación de los objetos a medida que se necesitan.
- Comunicación entre los objetos mediante el envío de mensajes unos a otros, o el usuario a los objetos.
- Eliminación de los objetos cuando no son necesarios para dejar espacio libre en la memoria del computador.

Relaciones entre clases. Herencia

Las clases, igual que los objetos, no existen de modo aislado. La Orientación a Objetos (POO) intenta modelar aplicaciones del mundo real tan fielmente como sea posible y por lo tanto debe reflejar estas relaciones entre clases y objetos.

De todas las relaciones posibles entre las distintas clases y objetos, hay que destacar por su importancia en O.O. la relación de herencia. La relación de herencia es una relación entre clases que comparten su estructura y el comportamiento.

- Se denomina **herencia simple** a la relación en que una clase comparte la estructura y comportamiento de una sola clase.
- Se denomina **herencia múltiple** a la relación en que una clase comparte la estructura y comportamiento de varias clases.

Para que un lenguaje de programación pueda ser considerado orientado a objetos, debe implementar el mecanismo de herencia.

La clase superior de la jerarquía, en cada relación, se denomina **superclase**, clase base ó clase padre y, la clase que hereda de la superclase, se denomina **subclase**, clase derivada ó clase hija.

Relaciones entre clases. Herencia

La herencia es:
HERENCIA = TRANSMISIÓN + REDEFINICIÓN + ADICIÓN
Las implicaciones de la herencia sobre los objetos de las clases involucradas son las siguientes:

- Sobre los objetos de la superclase A: ninguna
- Sobre los objetos de la subclase B:
  - ✓ Contienen todos los atributos que contienen los objetos de la superclase.
  - ✓ Contiene los atributos añadidos de la subclase.
  - ✓ Responden a los mensajes que corresponden con métodos transmitidos a la subclase; es decir, como dicta el método de la superclase.
  - ✓ Responden a los mensajes que corresponden con métodos añadidos a la subclase; es decir, como dicta el método de la subclase.
  - ✓ Responden a los mensajes que corresponden con métodos redefinidos en la subclase; es decir, como dicta el método de la subclase anulando el método de la superclase.

Sintaxis de la herencia en Java.

```
modificador NombreSubClase extends NombreSuperClase{
                    …
}
```

Relaciones entre clases. Herencia

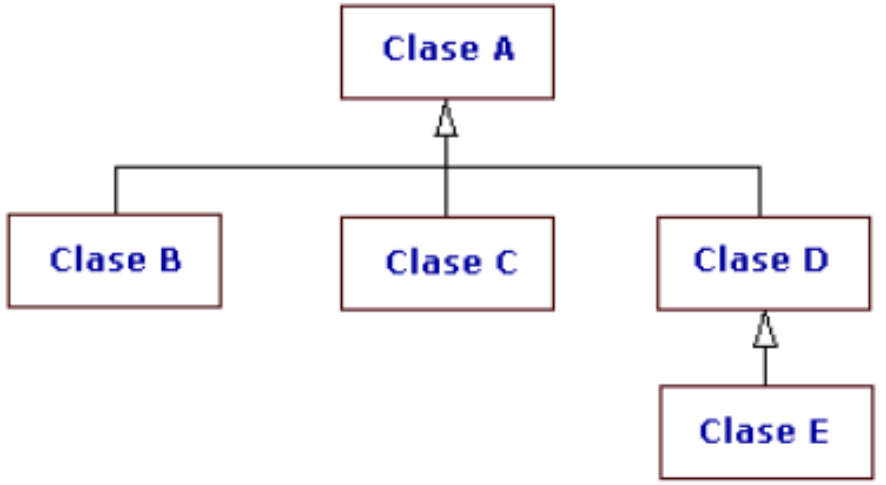
Existen dos tipos de herencia:

- Herencia **simple**: Permite definir nuevas clases a partir de una inicial.
- Herencia **múltiple**: Permite definir nuevas clases a partir de dos o más clases iniciales.

Java solo permite la primera, mientras que la segunda es “implementada” mediante el uso de interfaces.

La herencia conduce a una estructura jerárquica de clases o estructura de **árbol**, lo cual significa que en la POO todas las relaciones entre clases deben ajustarse a dicha estructura.

¿Cuáles son superclases? ¿Cuáles son subclases?



Referencia a null

- A la hora de trabajar con **objetos**, necesitamos tener una **referencia de memoria** asignada.
- Para ello tenemos que utilizar el operador **new** antes de trabajar con el objeto.
- Esto es debido a que podemos declarar el objeto e instanciarlo posteriormente.
- En aquellos casos que no hemos referenciado el objeto, su referencia vale **null** (sin dirección de memoria). En Java, siempre que vayamos a trabajar con un objeto, tenemos que asignarle una referencia válida.

Ejemplo:

```
public static void main( String[] args)
{
  Punto p;//también puedes asignarle el valor null: p = null;
  if(p == null) //No se puede utilizar el objeto, ya que no existe como tal.
 {
      System.out.println("Referencia nula a punto");
 }
}
```

JOptionPane

Esta clase nos permite crear ventanas que permiten una comunicación simple entre el usuario y el sistema.

Mediante estas ventanas, podemos ingresar y mostrar información, para ello, tenemos que importar la librería:

```
import javax.swing.JOptionPane; // import javax.swing.*; también nos vale.
```

Mediante esta clase, podemos crear diferentes tipos de ventanas de diálogo:

- **showConfirmDialog**
- **showInputDialog**
- **showMessageDialog**
- **showOptionDialog**

JOptionPane

Esta clase nos permite crear ventanas que permiten una comunicación simple entre el usuario y el sistema.

Mediante estas ventanas, podemos ingresar y mostrar información, para ello, tenemos que importar la librería:

```
import javax.swing.JOptionPane; // import javax.swing.*; también nos vale.
```

Mediante esta clase, podemos crear diferentes tipos de ventanas de diálogo:

- **showConfirmDialog**
- **showInputDialog**
- **showMessageDialog**
- **showOptionDialog**

JOptionPane

Esta clase nos permite crear ventanas que permiten una comunicación simple entre el usuario y el sistema.

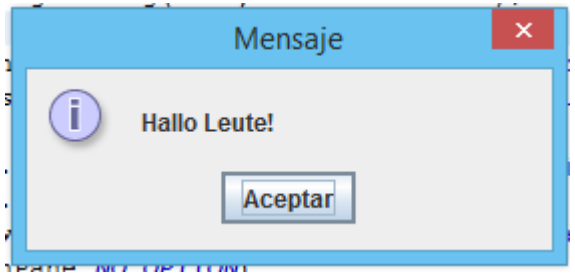
Mediante estas ventanas, podemos ingresar y mostrar información, para ello, tenemos que importar la librería:

```
import javax.swing.JOptionPane; // import javax.swing.*; también nos vale.
```

Mediante esta clase, podemos crear diferentes tipos de ventanas de diálogo:

- **showConfirmDialog**

```
JOptionPane.showMessageDialog(null, "Hallo Leute!");
```



JOptionPane

Esta clase nos permite crear ventanas que permiten una comunicación simple entre el usuario y el sistema.

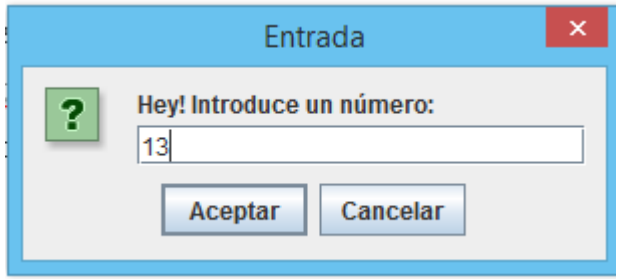
Mediante estas ventanas, podemos ingresar y mostrar información, para ello, tenemos que importar la librería:

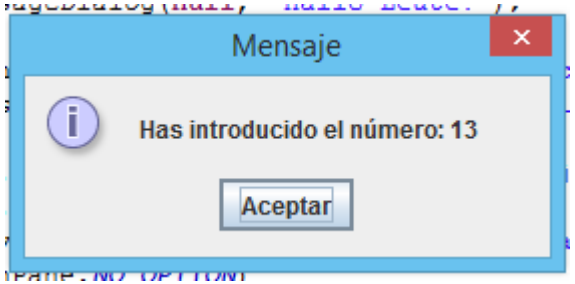
import javax.swing.JOptionPane; // import javax.swing.*; también nos vale.

Mediante esta clase, podemos crear diferentes tipos de ventanas de diálogo:

- **showConfirmDialog**
- **showInputDialog**

```
String aux = JOptionPane.showInputDialog("Hey! Introduce un número: ");
JOptionPane.showMessageDialog(null, "Has introducido el número: "+aux);
```





JOptionPane

Esta clase nos permite crear ventanas que permiten una comunicación simple entre el usuario y el sistema.

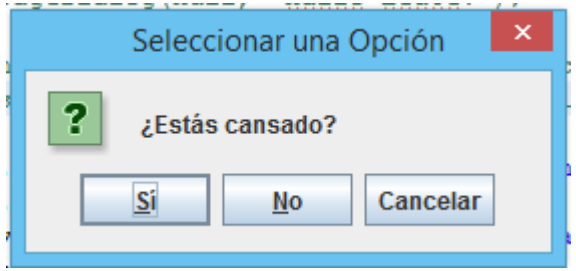
Mediante estas ventanas, podemos ingresar y mostrar información, para ello, tenemos que importar la librería:

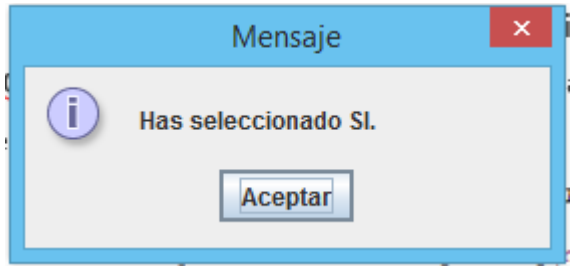
import javax.swing.JOptionPane; // import javax.swing.*; también nos vale.

Mediante esta clase, podemos crear diferentes tipos de ventanas de diálogo:

- **showConfirmDialog**
- **showInputDialog**
- **showMessageDialog**

```
int i = JOptionPane.showConfirmDialog(null, "¿Estás cansado?");
if(i == JOptionPane.YES_OPTION)
    JOptionPane.showMessageDialog(null, "Has seleccionado SI.");
else if(i == JOptionPane.NO_OPTION)
    JOptionPane.showMessageDialog(null, "Has seleccionado NO.");
```





JOptionPane

Esta clase nos permite crear ventanas que permiten una comunicación simple entre el usuario y el sistema.

Mediante estas ventanas, podemos ingresar y mostrar información, para ello, tenemos que importar la librería:

import javax.swing.JOptionPane; // import javax.swing.*; también nos vale.

Mediante esta clase, podemos crear diferentes tipos de ventanas de diálogo:

- **showConfirmDialog**
- **showInputDialog**
- **showMessageDialog**
- **showOptionDialog**

```
int opcion = JOptionPane.showOptionDialog(
                        null                                    //componente
                      , "¿Cómo se encuentra el semáforo?"       // Mensaje
                      , "Prueba ocular"                         // Titulo en la barra del cuadro
                      , JOptionPane.DEFAULT_OPTION              // Tipo de opciones
                      , JOptionPane.INFORMATION_MESSAGE         // Tipo de mensaje (icono)
                      , null                                    // Icono (ninguno)
                      , semáforo                                // Opciones personalizadas
                      , null                                    // Opcion por defecto
                    );

JOptionPane.showMessageDialog(null, "El semáforo se encuentra en: "+semáforo[opcion]);
```