TEMA 4. ESTUDIO DE LOS FUNDAMENTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

INDICE

PROGRAMACIÓN ORIENTADA A OBJETOS. INTRODUCCIÓN

La Programación Orientada a Objetos (POO u OOP en inglés) es un paradigma de programación (método de programación) que hace más hincapié en la forma en que se relacionan los elementos que forman parte de la solución del problema a resolver que en los pasos necesarios para solucionar el problema.

A la hora de resolver un problema usando Programación Orientada a Objetos debemos determinar qué elementos (objetos) necesitamos.

La Programación Orientada a Objetos utiliza técnicas como la herencia, abstracción, polimorfismo y encapsulamiento para facilitar la resolución de problemas.

La Programación Orientada a Objetos presenta las siguientes ventajas

Reutilización de código. Una vez desarrollada una clase y comprobado su correcto funcionamiento la podemos reutilizar a la hora de crear otras clases. Uno de los mecanismos de reutilización de código es la herencia.

Modularidad. Podemos añadir o modificar algunos de los elementos de una clase sin afectar al resto de elementos. Esto simplifica el mantenimiento de los programas.

Encapsulación. Al definir una clase podemos especificar a qué partes y de qué modo se puede acceder a la misma. Esto minimiza muchos los problemas a la hora de trabajar con la clase. El usuario de la clase sólo tiene que conocer la parte pública de la clase (interfaz) para trabajar con ella.

CLASES EN PROGRAMACIÓN ORIENTADA A OBJETOS

Los objetos se definen mediante **clases** que contienen todo lo necesario para representar y manipular cualquier objeto de esa clase. Es muy importante realizar un estudio detallado (**abstracción**) de todas las posibles características de todos los objetos de la clase antes de crear la clase para evitar tener que modificar la clase una vez que sea definida.

Una de las ventajas de la Programación Orientada a Objetos es que una vez que se define una clase ésta se puede reutilizar completamente para la solución de futuros problemas.

BIBLIOTECAS DE CLASES. PAQUETES

Una de las principales ventajas de la Programación Orientada a Objetos es que no se comienza a trabajar desde cero ya que todos los lenguajes de Programación Orientada a Objetos tienen una **biblioteca de clases predefinidas** que facilita mucho la resolución de problemas comunes. Por ejemplo, Java consta de la clase **System** que contiene la subclase **out** que nos permite mostrar un mensaje por pantalla de manera sencilla.

Las clases no son independientes sino que se encuentran dentro de una **jerarquía de clases**.

Las clases que tienen una función similar se agrupan dentro de paquetes (packages). Por ejemplo, las clases que se encargan de escribir algo en el dispositivo de salida estándar (pantalla) se encuentran en el paquete **System.out**.

A la hora de crear una nueva clase se cargan algunos paquetes de clases predefinidos (por ejemplo System.out) lo que nos permite usar algunas clases sin realizar ninguna operación adicional (por ejemplo podemos escribir un mansaje por pantalla usando **System.out.println("Hola.");**).

Si queremos usar otras clases predefinidas debemos importarlas dentro de nuestra clase usando la sentencia **import** antes de la definición de la clase. Por ejemplo, si queremos usar la clase Scanner que se encuentra en java.util antes de la clase debemos escribir

```
import java.util.Scanner;
```

ESTRUCTURA Y MIEMBROS DE UNA CLASE

Una clase consta de una parte de **datos miembro** o **atributos** que permiten guardar las propiedades de cada objeto de la clase y de otra de **funciones miembro** o **métodos** que permite realizar operaciones sobre los datos de los objetos de la clase.

Es muy importante darse cuenta de que lo que diferencia un objeto de una clase de otro son sus datos miembro ya que todos los objetos de la clase comparten el código de las funciones de la clase.

El **estado de un objeto** es la situación de un objeto en un momento dado de la ejecución del programa. Está definido por el valor de sus datos miembro.

Para crear una clase de nombre Complejo en Java escribimos

```
class Complejo{
}
```

OBJETOS EN PROGRAMACIÓN ORIENTADA A OBJETOS

Un objeto es una instancia de una clase o lo que es lo mismo un objeto es la representación real de una clase. Las clases son representaciones teóricas mientras que los objetos son representaciones reales que consumen recursos (por ejemplo memoria).

Para usar un objeto de una clase debemos crear una clase distinta a la clase del objeto. Por ejemplo, para usar un objeto de la clase Complejo creamos la clase **ComplejoMain** dentro de la cual hay una función **main** en la que escribiremos el código necesario para crear un objeto de la clase Complejo y de nombre c

```
Complejo c;
```

PROPIEDADES O ATRIBUTOS GENERALES

Los atributos de un objeto son los atributos que vienen definidos en la clase. Cambian de un objeto a otro y son lo que define el estado del objeto.

Por ejemplo, si la clase **Complejo** tiene un atributo de tipo **double** y de nombre **real** que guarda el valor de la parte real de un número complejo y otro atributo de tipo **double** y de nombre **imaginaria** que guarda el valor de la parte imaginaria escribimos

```
class Complejo{
 double real;
 double imaginaria;
}
```

EL OBJETO THIS.

Java mantiene siempre una referencia al objeto que se está utilizando en cada momento. Podemos saber cuál es el objeto que se está utilizando mediante el identificador **this**.

El identificador this es muy importante para determinar cuál de los posibles objetos de la clase es el que se está utilizando en un determinado momento.

Para acceder a las propiedades del objeto actual escribimos **this.propiedad**.

Por ejemplo, para acceder al atributo real de la clase complejo desde un método podemos poner

```
this.real;
```

MODIFICADORES DE ACCESO

A la hora de definir un elemento de una clase podemos especificar modificadores de acceso. Estos modificadores de acceso indican que elementos tienen acceso al elemento en cuestión.

Existen varios modificadores de acceso:

public. Un elemento definido como public puede ser accedido por cualquier elemento pertenezca o no o la misma clase. Por ejemplo, si queremos que la clase Complejo pueda ser accedida desde otra clase a la hora de definirla escribimos

```
public class Complejo{...}
```

private. Un elemento definido como private sólo puede ser accedido por elementos que pertenezcan a la misma clase. Se suelen definir como private los atributos de la clase. Por ejemplo, si en la clase Complejo queremos que los atributos sólo puedan ser accedidos por miembros de la misma clase escribimos

```
public class Complejo{
 private double real;
 private double imaginaria;
}
```

protected. Un elemento definido como protected sólo puede ser accedido por elementos que pertenezcan a la misma clase o a alguna de las clases derivadas de ella. Lo veremos más adelante cuando estudiemos la herencia de clases.

default. Es el modificador por defecto si no se indica nada. Se le suele llamar **default** o **package-private**. Un elemento definido como default sólo puede ser accedido por elementos que pertenezcan al mismo paquete (package).

CONSTRUCTOR DE UNA CLASE.

Un constructor de una clase es un método especial que se ejecuta cuando se crea un nuevo objeto de la clase. Se usa para inicializar de la manera que se quiera el valor de los atributos del objeto.

Su nombre coincide con el nombre de la clase. Por ejemplo, si la clase se llama Complejo el nombre del constructor será también Complejo.

TIPOS DE CONSTRUCTORES.

Existen varios tipos de métodos constructores aunque todos ellos tienen el mismo nombre (que coincide con el de la clase).

Constructor por defecto. Es el que se ejecuta cuando la clase no tiene ningún constructor. Si la clase no tiene ningún constructor el compilador asigna a cada uno de los atributos el valor inicial que corresponda a su tipo de dato (por ejemplo 0.0 a los datos de tipo double). Por ejemplo, si queremos que al crear un nuevo objeto de la clase Complejo se ejecute el constructor por defecto escribimos

```
Complejo c1;
```

Constructor copia. Es un tipo especial de constructor que permite crear un nuevo objeto de una clase tomando como base los datos de otro objeto de la misma clase.

El código del constructor copia de la clase Complejo es

```
Complejo (Complejo c){
 real = c.real;
 imaginaria = c. imaginaria;
}
```

Por ejemplo, si queremos que al crear un nuevo objeto de la clase Complejo se ejecute el constructor copia escribimos

```
Complejo c2 = new Complejo(c1);
```

Constructores personalizados. Podemos crear tantos constructores como queramos. La única condición es que el tipo de sus parámetros tiene que ser distinto.

El código del constructor personalizado que da un valor a la parte real y otro a la parte imaginaria al crear un nuevo objeto de la clase Complejo es

```
Complejo (double r, double i){
 real = r;
 imaginaria = i;
}
```

Por ejemplo, si queremos que al crear un nuevo objeto de la clase Complejo se ejecute un constructor personalizado que pone un valor inicial tanto a la parte real como a la parte imaginaria escribimos

```
Complejo c3 = new Complejo(2.0,3.0);
```

DESTRUCTOR DE UNA CLASE.

Un destructor de una clase es un método especial que se ejecuta cuando se destruye objeto de la clase.

La función principal de un destructor es la de liberar los recursos asignados a un objeto antes de que éste se destruya. Esto puede ser necesario si al trabajar con el objeto se han utilizado métodos de asignación dinámica de memoria.

En Java no existen los destructores como tal ya que de la liberación de recursos se encarga el **recolector de basura** (garbage collector).

Si en algún momento de la ejecución del programa consideramos que es necesario invocar al recolector de basura escribimos

```
System.gc(); // o Runtime.getRuntime().gc();
```

Aunque en Java no exista un método destructor como tal, si cuando se destruye un objeto necesitamos realizar alguna operación especial podemos utilizar el método especial **finalize**. Por ejemplo, si queremos que cuando se destruya un objeto de la clase Complejo aparezca el mensaje "Complejo destruido" escribimos dentro de la clase Complejo

```
public void finalize(){
 System.out.print("Complejo destruido");
}
```

Si ejecutamos ahora la clase ComplejoMain aparecerá el mensaje "Complejo destruido" cada vez que se destruya un mensaje de la clase Complejo.

Dado que Java ya incorpora un sistema de recuperación de recursos se aconseja no usar el método finalize salvo en los casos en los que sea estrictamente necesario.

EJERCICIOS DE CLASES

1. Realiza la clase Java Complejo de los ejemplos anteriores. Los objetos de la clase muestran el valor de sus atributos por pantalla usando un método de nombre **escribir** con el formato ***real " + " imaginaria + "i"***

2. Realiza la clase Java ComplejoMain para probar la clase Complejo.