Unidad 3. Programación Orientada a Objetos

La **Programación Orientada a Objetos** se basa en un conjunto de objetos interactuando entre sí.

Una **clase** es un modelo abstracto de un tipo de objeto, el cual define sus métodos y atributos.
Un **objeto** es una instancia de una clase, es decir, la implementación de un modelo abstracto con valores.

Encapsulamiento

El **encapsulamiento** se encarga de que los datos/funcionamiento de un objeto sean invisibles al resto, permitiendo una seguridad mayor ya que el acceso a estos datos depende directamente de cada objeto.

Herencia

La **herencia** consiste en la adopción de todas las características de una clase por parte de otra, definida como heredera o descendiente de la primera. Son necesarias ya que sirve como modelado de la realidad (existen relaciones similares entre entidades del mundo real), evitan redundancias (evitamos repetir código) y facilita la reutilización (pueden adaptarse a otras características).

Ej: Si tenemos una clase A, y una clase B que hereda de A:

- A es un ascendiente o superclase de B. A es la clase padre de B.
- B es un descendiente o subclase de A. B es la clase hija de A.

Todas las clases heredan automáticamente de una superclase universal, la clase Object.

Dentro de los diferentes niveles de protección en relación a las herencias nos encontramos con que:

- Miembros públicos (public). Son accesibles desde los descendientes.
- Miembros privados (private). No son accesibles desde los descendientes.
- Miembros con acceso a nivel paquete. Son accesibles desde los descendientes si están en el mismo paquete que el ascendiente.
- Miembros protegidos. Es accesible únicamente desde los descendientes.

Polimorfismo

El **polimorfismo** consiste en la posibilidad de que una referencia de una clase padre puede apuntar a objetos de sus clases hijas.

Clases en Java

Una **clase** es una agrupación de variables y métodos que operan sobre esos datos (atributos y métodos).

La **programación orientada a objetos** se basa en la programación de clases, siendo un **programa** un conjunto de clases.

Constructores

Sirven para la construcción (instanciación) de objetos (instancias) a partir de esa clase dándole valores.

Convenciones en Java

- Los **nombres de las clases** comienzan con **mayúsculas.**
- Los **nombres de los atributos** comienzan con **minúsculas.**
- Los **nombres de los métodos** comienzan con **minúsculas.**

Objetos

Los **objetos** son variables de tipo complejo frente a las de tipo primitivo. El **tipo de un objeto** es la clase de la que se ha instanciado.

Ej: Cuenta miCuenta; *miCuenta es de tipo Cuenta*

Su valor sin inicializar es **null.**

Unidad 4. Herencia

La **herencia** se basa en la existencia de relaciones de generalización/especialización entre clases, dándole así diferentes jerarquías, donde una clase hereda los atributos y métodos de otra superior.

- Las clases por encima en la jerarquía a una clase dada, es una **superclase.**
- Las clases por debajo en la jerarquía a una clase dada, es una **subclase.**

Una clase puede ser superclase y subclase al mismo tiempo. Existe la herencia simple y múltiple (esta última no es soportada en Java).

En Java todas las clases heredan de otra clase. Si especificamos cual, heredará de esta. Si no lo especificamos, el compilador hace que la clase herede de la clase Object (raíz de la jerarquía). Esto significa que nuestra clase hereda atributos de la clase Object, como por ejemplo:

- **public boolean equals(Object o);** Compara dos objetos y dice si son iguales.
- **public String toString();** Devuelve la representación visual de un objeto.
- **public Class getClass();** Devuelve la clase de la cual es instancia el objeto.
- **public int hashCode();** Devuelve un identificador unívoco después de aplicarle algoritmo hash.
- **public Object clone ();** Devuelve una copia del objeto.
- **public void finalize();** Un método llamado por el Garbage Collector.

Sobrecarga de métodos

La **sobrecarga de métodos** consiste en tener varias implementaciones del mismo método con parámetros distintos:

- Su nombre ha de ser el mismo.
- El tipo de retorno puede ser distinto.
- Los parámetros tienen que ser distintos.
- El modificador de acceso puede ser distinto.

Sobreescritura de métodos

La **sobreescritura de métodos** consiste en que una subclase implemente un método heredado.

- Su nombre ha de ser el mismo.
- El tipo de retorno puede ser distinto.
- Los parámetros han de ser los mismos.
- El modificador de acceso no puede ser más restrictivo.

En Java 5.0 se permite la sobreescritura de métodos cambiando su retorno siempre y cuando el método que se esté sobreescribiendo sea de una clase padre.

Al ejecutar un método se busca su implementación de abajo hacia arriba según la jerarquía de clases.

Super

La keyword **super** se usa para acceder a un atributo/método del padre desde un objeto.

This

La keyword **this** se usa para acceder a un atributo/método del propio objeto.

Modificadores de acceso

Ordenados de **menor a mayor** restricción nos encontramos:

- public
- protected
- private

Acceso a…	public	protected	package	private
Clases del mismo paquete	Si	Si	Si	No
Subclases de mismo paquete	Si	Si	Si	No
Clases de otros paquetes	Si	No	No	No
Subclases de otros paquetes	Si	Si	No	No