FUNDAMENTOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

Dentro de las distintas formas de hacer las cosas en la programación, distinguimos dos paradigmas fundamentales:

Programación estructurada: se crean funciones o procedimientos que defines las acciones a realizar, y que posteriormente forman los programas

Programación orientada a Objetos: considera los programas en términos de objetos y todo gira alrededor de ellos.

La **programación orientada a objetos** es un **sistema o conjunto de reglas** que nos ayudan a **descomponer la aplicación en objetos**. A menudo se trata de **representar las entidades y objetos** que nos encontramos en el mundo real mediante componentes de una aplicación. Es decir, debemos establecer una **correspondencia directa entre el espacio del problema y el espacio de la solución.**

A la hora de escribir un programa, nos fijaremos en los objetos involucrados, sus características comunes y las acciones que podemos realizar. Una vez localizados estos objetos (problema), se trasladarán al programa informático (solución).

CARACTERÍSTICAS:

- **Abstracción**: Es el proceso por el cual definimos las características más importantes de un objeto de forma general. La herramienta mas importante para soportar la abstracción es la **clase**. Una clase es un tipo de dato que agrupa las características comunes de un conjunto de objetos.
- **Modularidad**: Una vez representado el escenario, 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 una clase que define un objeto, sin que esto afecte al resto de clases de la aplicación.
- **Encapsulación**: También llamada "Ocultamiento de la información". La encapsulación es el mecanismo básico para ocultar información de las partes internas de un objeto a los demás objetos de la aplicación.
- **Jerarquía**: Mediante esta propiedad podemos definir relaciones entre jerarquías entre clases y objetos. Existen la **jerarquía de generalización** o especialización o la **jerarquía de agregación**.

-**La jerarquía de generalización**: es la que se conoce como **HERENCIA,** permite crear una clase nueva en términos de una ya existente (h. **simple**) o de varias clases ya existentes (**múltiple**).

-La jerarquía de agregación: permite agrupar objetos relacionados entre si dentro de una clase.

- **Polimorfismo:** Esta propiedad indica la capacidad de que varias clases creadas a partir de una antecesora realicen una misma acción de forma diferente.

MODIFICADORES DE ACCESO: PUBLIC, PRIVATE, PROTECTED.

Una subclase no tiene acceso a los campos de una superclase de acuerdo con el principio de **ocultación de la información.** Sin embargo, esto podría considerarse como demasiado restrictivo.

Decimos que podría considerarse demasiado restrictivo porque limita el acceso a una subclase como si se tratara de una clase cualquiera, cuando en realidad la relación de una superclase con una subclase es más estrecha que con una clase externa. Por ello en diferentes lenguajes, Java entre ellos, se usa un nivel de acceso intermedio que no es ni **public** ni **private**, sino algo intermedio que se denomina como "**acceso protegido**", expresado con la palabra clave **protected**, que significa que las subclases sí pueden tener acceso al campo o método.

El modificador de acceso **protected** puede aplicarse a todos los miembros de una clase, es decir, tanto a campos como a métodos o constructores. En el caso de métodos o constructores protegidos, estos serán visibles/utilizables por las subclases y otras clases de este package. El acceso protegido suele aplicarse a métodos o constructores, pero preferiblemente no a campos, para evitar debilitar el encapsulamiento. En ocasiones puntuales sí resulta de interés declarar campos con acceso protegido.

Java admite una variante más en cuanto a modificadores de acceso: la omisión del mismo (no declarar ninguno de los modificadores public, private o protected). En la siguiente tabla puedes comparar los efectos de usar uno u otro tipo de declaración en cuanto a visibilidad de los campos o métodos:

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
public	Sí	Sí	Sí	Sí
protected	Sí	Sí	Sí	No
No especificado	Sí	Sí	No	No
private	Sí	No	No	No

CONCEPTO DEL CONSTRUCTOR

Un **constructor** es un método especial con el **mismo nombre de la clase** y que no **devuelve ningún valor** de forma explícita tras su ejecución, aunque implícitamente se **devuelve el objeto que se está creando a través del operador new**

Características:

- Es invocado automáticamente en la creación de un objeto, solo esa vez
- Los nombres de los constructores no empiezan con minúscula.
- Puede haber más de un constructor para una clase, que se llamaran todos igual pero que se diferenciaran por su lista de argumentos (**sobrecarga**)
- El constructor puede tener parámetros para indicar con que valores iniciar o configurar los atributos del objeto que se va a crear
- Toda clase debe de tener un constructor.

SUPER

El método **super()** es un uso especial de la palabra clave super donde se llama un constructor principal. En general, la palabra clave super se puede utilizar para llamar a métodos anulados, acceder a propiedades ocultas o invocar al constructor de una superclase.

Si usamos **super**. Desde una subclase, podemos invocar el método de la clase a la que pertenece