Términos que utilizamos:

* **Instanciación**, consiste en crear un objeto inicializándolo.
* **Encapsulación**, sirve para no acceder directamente a los datos y que estos permanezcan ocultos.
* **Paquete**, es una agrupación de clases e interfaces que facilitan la gestión de acceso a las clases. Permiten proteger a las clases de otras clases.
* **Interface**, es una clase donde solo escribimos las cabeceras de los métodos ( no se desarrollan ).
* **Herencia** sirve para reutilizar código. Para que una clase herede características y comportamientos de otra clase.
* **Polimorfismo**. Existen dos tipos: **estático** y **dinámico**. El estático es cuando tenemos varios métodos con el mismo nombre y argumentos distintos. Y el dinámico no trabaja con ningún tipo de objeto en concreto.
* **Sobrecarga,** cuando varios métodos tienen el mismo nombre pero están definidos en clases distintas. Podemos decir que es un polimorfismo estático.

**Declaración de una clase**

Los elementos que puede tener son:

**Modo\_de\_acceso [abstract| final] class nombre extends Clasepadre Implements Interfaz1,Interfaz2,…**

* Modo de acceso, indica quien puede utilizar la clase. Pueden ser:
  + **public**, puede ser accedido desde fuera de la clase y por cualquier otra clase.
  + **prívate**, solo se puede acceder desde la propia clase donde se define.
  + **package**, se accede para todas las clases del mismo paquete. Es el valor por defecto que toma, no es necesario escribirla.
  + **protected**, se accede desde las clases del mismo paquete o clase heredadas.

El modo de acceso suele ser: **public** o **package**.

* **abstract**, no podemos crear objetos de la clase, por lo que no podemos utilizar **new**. Se utiliza para crear un nuevo tipo de objeto y heredar a partir de ella, está obligado a implementar los métodos abstractos que tenga la clase. Podemos tener métodos abstractos y no abstractos, pero los no abstractos no está obligado a heredarlos. Si una clase tiene métodos abstractos la clase debe ser abstracta.
* **final**, no tiene clases hijas, no hereda de esta clase.
* **extends**, indica que hereda los métodos y atributos de la clase indicada.
* **implements**, es una especie de herencia y se pueden implementar una o varias interfaces. Sirve para solucionar el problema de la herencia múltiple de la que no dispone Java.

En las clases se oculta la implementación. Para acceder a los datos internos empleamos los métodos get y set. Estas palabras las anteponemos al nombre del método, se escribe la primera letra del atributo en mayúscula ( getRadio). Para visualizar get y para modificar set.

**Atributos de una clase**

Los modificadores de acceso son:

* **public**, cualquier otra clase puede acceder desde cualquier sitio.
* **prívate**, el acceso solo es permitido desde la propia clase.
* **protected**, solo se puede al atributo desde clases que se encuentren en el mismo paquete o desde clases heredadas.
* **package**, ( valor por defecto y no se escribe ) se accede desde clases del mismo paquete.

Otros que podemos poner añadidos a los anteriores son:

* **static**, todos los objetos comparten la misma variable.
* **final**, su valor no puede ser cambiado.

Los atributos suelen ser prívate.

**Modificadores de los métodos**

**Modo\_de\_acceso [static | abstract| final] tipo\_a\_devolver nombre(argumentos ) throws excepción1, excepción2,..**

Son los mismos que hemos visto para las clases y atributos: **public**, **prívate**, **protected** y **package**.

También podemos añadir con los otros:

* **static**, no son métodos de ningún objeto, se pueden utilizar desde la clase.
* **abstract**, indica que el método no tendrá implementación y obliga a que la clase sea abstracta (no se pueden crear objetos de esta clase). Puede servir como padre de una jerarquía de clases.
* **final**, protege los métodos que no queremos que sean modificados. Si es heredado no podrá ser sobrecargado.

Un método static puede ser utilizado sin tener declarado ningún objeto. Solo podemos acceder a atributos de la clase declarados como estáticos. Como el uso de la librería Math.

Los métodos static no tienen referencias this, no pueden acceder a miembros que no sean static.

Un método no static puede acceder a métodos static y no static.

Un objeto declarado como final, impide que tengamos otro objeto con la misma referencia.

**Destrucción de objetos**

Destrucción de objetos y liberación de memoria. Se destruyen automáticamente por medio del **Garbarger Collector** o recolector de basura. Podemos forzarlo con **System.gc( )** lo que hace es ejecutar el método **protected void finalize( )**. No tiene valor de retorno ni argumentos, y no puede ser **static**.

Generalmente se utilizan para liberar memoria, cerrar ficheros, conexiones, etc… Podemos forzar la ejecución del recolector de basura con el método:

**System.runFinalization( );**

**System.gc( );**

**HERENCIA**

La herencia sirve para reutilizar código. Es un mecanismo de definir objetos basados en otros existentes Cuando una clase deriva de una clase padre, está hereda todos los miembros y métodos.

La clase hija recibe el nombre de **subclase** o **clase extendida** que hereda los miembros de la clase existente y que puede añadir otros miembros propios ( atributos y métodos ). Una subclase puede ser una superclase de otra clase.

Hereda:

* La subclase hereda los miembros declarados como **public** o **protected**.

No hereda:

* No hereda los constructores.
* La subclase no hereda un miembro con el mismo nombre de la subclase.
* Las subclases no heredan los miembros (atributos /métodos) privados de la superclase.

Los miembros declarados como package son heredados si pertenecen al mismo paquete.

Empleamos la palabra reservadas **this**, que hace referencia a la propia clase. Puede ir asociada a atributos o a métodos.

La clase padre es una **superclase**. No puede utilizar los atributos o métodos de la clase hija.

**Super** se utiliza para referenciar a la superclase y poder llamar a métodos de la misma, aunque estén redefinidos ( tienen el mismo nombre que en la clase hija ). Está disponible para cualquier método no estático de una subclase. Podemos acceder a miembros del tipo protected con la palabra super. Con la palabra super dentro del constructor de una subclase, llamamos al constructor de la superclase, debe ser la primera dentro del constructor.

No existe la herencia múltiple, solo se puede heredar de una clase. Para que un objeto herede de varios se emplea los interfaces.

Podemos tener clases anidadas. Dentro de una clase podemos escribir otra clase. La clase anidada es una clase que es miembro de otra clase, tendrá acceso a todos sus métodos y atributos; incluso los privados.

Tendríamos que instanciar primero la clase externa y luego la clase interna. Ej:

**externa.interna objeto = objetoexterno.new interna( );**

Ejemplo: clase TRABAJADOR

subclases PROFESOR ADMINISTRATIVO LIMPIEZA

Para indicar que una clase hereda de otra ponemos la clausula **extends** a continuación del nombre de la clase.

**class profesor extends trabajador{…………….}**

Toda clase tiene una clase padre que es Object (java.lang.Object). Algunos de los métodos que se heredan son: clone( ), equals( ), toString( ) y finalize( ), devuelve el nombre de la clase, seguido del símbolo @ y a continuación en hexadecimal el código hash del objeto ( su dirección de memoria ).

La clase profesor puede utilizar los métodos de la clase trabajador sino han sido declarados como prívate. Los métodos de la subclase no pueden ser utilizados por la clase principal o padre.

**Interfaces**

Un interfaz y una clase abstracta son muy parecidas. Te recuerdo que es un grupo de métodos con sus cuerpos vacios.

Java soporta **interfaces** que se utilizan para definir un protocolo de comportamiento que se puede implementar por cualquier clase. Es una clase abstracta

Tan solo contiene la declaración de:

1. Métodos, todos los métodos son del tipo **public abstract** por defecto.
2. Atributos, son constantes y por defecto son del tipo **public static final**.

Con la interfaces podemos obtener el efecto de herencia múltiple.

**modificador interface nombre\_interfaz**

**{**

**declaración de constantes.**

**declaración de métodos.**

**}**

No podemos crear objetos de una clase interface

Una clase que hereda de una Interfaz o de varias es una implementación.

**modificador class nombrClase implements nombre\_interfaz, nombre\_interfaz2,…**

Para compilar correctamente una clase que implementa una interfaz, ésta debe contener los métodos declarados en dicha interfaz.

Debemos tener en cuenta:

* Los atributos que contiene la interfaz debes asignar algún valor.
* Los atributos y métodos no pueden tener modificadores: **prívate** o **protected**.
* Una interfaz no puede tener **constructor**.
* No necesitas poner **abstract** al método lo es por defecto.
* Dentro de la clase que implementa un interfaz, todos los métodos deber ser **public**.
* Se puede declarar un objeto de una interfaz pero no puede ser instanciado.