**MODIFICADOR FINAL**

***En una clase***

* Una clase con modificador final no se puede heredar.
* String, LocalDate, LocalDateTime son final, por lo tanto no se pueden heredar

***En un atributo o variable local***

* Un atributo o variable local con modificador **final**, no puede ser modificada

**En un método**

* Un método final no puede ser sobreescrito

**EXCEPCIONES**

* ***Throwable*** es el padre de ***Exception*** y ***Error***
* Se puede heredar de Exception y de Error
* En un bloque ***try/catch***, el parámetro del catch es ***Throwable***, por tal motivo se puede hacer catch(Throwable ex) catch(Exception ex) catch(Error ex)
* La diferencia entre Exception y Error es que de un error no se puede recuperar

Ejemplo de Error:

ArrayList<String> cadenas=new ArrayList<>();

while(true){

cadenas.add(“msg”);

}

Produce un ***OutOfMemoryError***

* RuntimeException y todas sus hijas son Unchecked
* Exception y todas sus hijas excepto RuntimeException son Checked
* Las excepciones Checked deben ser capturadas o relanzadas
* Todas las excepciones pueden ser capturadas o relanzadas
* Los errores no son recuperables
* Las excepciones son recuperables
* Se puede lanzar una excepción con la palabra reservada throw
* En la firma del método se especifica que el método puede lanzar excepciones con la palabra reservada throws
* Las excepciones pueden ser tratadas en los métodos donde se originan o se pueden propagar. Se pueden crear nuevas Excepciones heredando de Exception y nuevos errores heredando de Error
* El bloque try/catch permite separar el manejo de la excepción del flujo normal del programa

**HERENCIA**

* Si class Perro extends Animal, implica que *Perro* **ES UN** *Animal*
* Si class Gato implements Domable, implica que **Gato ES UN Domable**

Ejemplo: Como Exception hereda de Throwable

Exception **ES UN** Throwable

* Object es la clase padre de todas las clases java
* La herencia permite compartir el comportamiento de las clases padres con las clases hijas
* La herencia permite reusar el código en las clases hijas

**ESTRUCTURA DE LA CLASE**

* Las clases se organizan en paquetes.
* Por defecto una clase solo puede ver las clases del mismo paquete, si se requiere ver clases de otro paquete se debe utilizar la sentencia import.
* Solamente la clase que tiene el método main es ejecutable.
* Si una clase no tiene constructor, el lenguaje le agrega un constructor por defecto que no recibe parámetros.
* Si se define un constructor en una clase, se pierde el constructor por defecto.
* Una clase puede tener constructores privados, estos se utilizan por ejemplo para aplicar el patrón de diseño Singleton, ya que impiden que se pueda instanciar un objeto desde fuera de la clase.
* Un método puede tener el mismo nombre de un atributo.

**SOBRECARGA Y SOBREESCRITURA**

* Los métodos sobrecargados tienen el mismo nombre, pero diferentes parámetros.
* Los métodos estáticos también se pueden sobrecargar.
* Los métodos sobreescritos tienen el mismo nombre y los mismos parámetros que la clase padre.

**VARIABLES LOCALES vs ATRIBUTOS**

* Los atributos tienen valores iniciales por defecto según el tipo de dato, int:0 double:0.0 y los atributos de tipo clase, tienen valor por defecto null.
* Las variables locales no tienen valor inicial, por lo tanto deben ser inicializadas antes de usarse.
* Las variables locales no pueden tener modificadores de acceso (Calambre al ojo)
* Si dentro de un método hay una variable local que se llame igual que un atributo, tiene preferencia la variable local.

**SWITCH**

* El parámetro que recibe el switch puede ser de tipo int o de menor tamaño y sus respectivos Wrappers, también puede ser de tipo String o Enum. Es válido que reciba un char, ya que el char tiene una representación de int (usa el código ASCII)

Ejemplo:

char c1=’a’;

char c2=64; //se guarda @

int x=’a’;//se guarda el ASCII de a

* En cada switch existen uno o varios cases.
* Una vez que ingresa al switch, se analiza cada uno de los cases, en caso de que no coincida ninguno se ejecuta la sentencia default (es opcional que exista un default dentro del switch).
* Si no se coloca la instrucción break luego de un case o un default, el código compila correctamente, sin embargo al ejecutar va a continuar la ejecución de los siguientes cases hasta encontrar un break.
* El switch se debe evaluar a un valor único, es decir no se pueden repetir los valores de los cases en un mismo switch.
* El tipo de dato del case debe tener concordancia con el parámetro que recibe el switch
* Los valores del case son valores fijos.

**MAIN**

* El main es el primer método que se ejecuta.
* El main debe cumplir las siguientes características:
  + Debe ser public
  + Debe ser static
  + Retorna void
  + Debe recibir un arreglo de String
* Si no cumple estas condiciones, compila sin problemas, pero será tratado como un método normal, es decir no se ejecuta cuando se ejecuta la clase

**ENCAPSULAMIENTO, HERENCIA Y POLIMORFISMO**

* ***ENCAPSULAMIENTO:*** Consiste en ocultar los atributos de una clase para que no puedan ser accedidos directamente desde otras clases. Los atributos encapsulados solo pueden accedidos a través de los métodos de la clase, generalmente los getters y setters. Los métodos también pueden ser encapsulados, de manera que no puedan ser accedidos desde otra clase.Este ocultamiento de miembros se lo hace utilizando los modificadores de acceso.
* ***POLIMORFISMO:*** Permite que un mismo método tenga diferente comportamiento, dependiendo del objeto al que acceda la referencia en ejecución. El polimorfismo permite generar código dinámico y más flexible.

**Ejemplo:**

Animal a=new Cat();

a.comer();//ejecuta el comer de Cat

a = new Dog();

a.comer();//ejecuta de comer de Dog

la misma llamada a.comer(); ejecuta comportamiento diferente dependiendo de a que objeto lleva la referencia en ejecución. De manera que el comportamiento de comer() es DINAMICO

* ***HERENCIA:*** La clase hija hereda los atributos y métodos del padre. Los miembros private no se heredan.

**BYTECODE**

* El comando javac compila el fuente y genera un archivo con el bytecode en un archivo con extensión .class.
* El bytecode puede correr en cualquier plataforma que tenga instalada un J**RE (Java Runtime Environment).** Write once run anywhere, hace referencia a que se escribe el código fuente una sola vez en cualquier plataforma y una vez compilado se puede ejecutar en cualquier plataforma que tenga un **JRE**.

**FOR STANDARD Y ENHANCED FOR**

* El bucle enhanced for, permite iterar la colección completa en el orden en que están ingresados los datos.
* El bucle for standard, permite iterar la colección de acuerdo a la manipulación de la posición, por ejemplo se podría iterar en orden inverso.