Curso de java

## javac

Es el compilador que nos provee el JDK, convierte el codigo de java en bytecode, que luego es utilizado por la JVM para correr el programa. La JVM convierte el codigo bytecode en codigo maquina, que puede ser ejecutado por el procesador

## jshell

Consola de comandos de java que permite ejecutar codigo sin crear clases o un metodo principal, sirve probar fragmentos de codigo u aprender.

## Como corre java?

Compilador: se encarga de convertir el human-readable code en bytecode.

JVM: Java Virtual Machine, transforma el btecodigo en machine code y lo ejecuta.

JRE: Provee a la JVM de las librerias y configuraciones necesarias para funcionar.

java -> **COMPILADOR** -> bytecode -> **JVM** -> machine code -> **RUN**

## JRE vs JVM

JRE es un paquete que incluye a la JVM, pero ademas tiene archivos de configuracion y las librerias necesarias para que la JVM pueda correr el codigo.

## JDK

Paquete que contiene un JRE, por lo tanto una JVM, y tambien un compilador de java (javac).

## Compilar y ejecutar un .java

javac ${nombreArchivo}.java

java ${nombreArchivo}

## Variables primitivas y sus wrappers

|  |  |  |
| --- | --- | --- |
| Tipo primitivo (tamaño) | Rango | Clase Wrapper |
| byte (1) | -128 a 127 | Byte |
| short (2) | -32,768 a 32,767 | Short |
| int (4) | ±2 mil millones aprox | Integer |
| long (8) | ±9 quintillones aprox | Long |
| float (4) | Precisión simple (32 bits IEEE 754) | Float |
| double (8) | Precisión doble (64 bits IEEE 754) | Double |
| char (2) | Representa un carácter Unicode | Character |
| boolean (1) | Técnicamente usa 1 bit, pero en objetos puede ocupar 1 byte o más | Boolean |

En java, todo es un objeto, excepto los tipos primitivos. Estos son mas eficientes de utilizar, pero no tienen las funcionalidades que nos otorgan sus wrappers. Por ejemplo, int es mas rapido, porque no es necesario alojar un objeto Int en el heap. Pero su wrapper, Integer, es mas lenta de utilizar, pero nos provee de metodos como el toString() y muchos mas.

## Method overloading

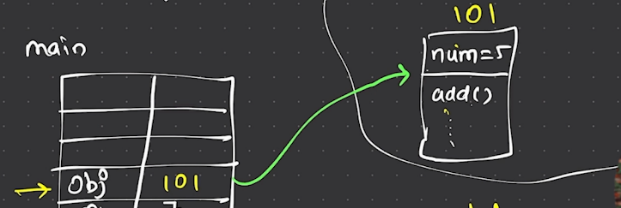
Una clase puede tener metodos con los mismos nombres siempre y cuando tengan distintos parametros.

## Method overriding

Es cuando reemplazamos el comportamiento de un metodo existente al heredar una clase, o implementamos el metodo de una interfaz.

## Stack

Es una pila con disciplina LIFO, en formato KEY:VALUE . Cada key es el nombre de una variable, y el value es su valor. Cada metodo de una clase, va a tener su propio stack.

Al instanciar un objeto y asignarlo a una variable “obj” , el objeto, junto con sus variables de instancia y metodos, seran guardados en el heap, mientras que el value de “obj” será la direccion en memoria de este objeto en el heap. Entonces las variables objeto, son en realidad, referencias al objeto real.

## Heap

Es un espacio en memoria que puede variar en tamaño, ahi se almacenan los objetos instanciados. Las variables de referencia, contienen la direccion del objeto instanciado en el heap.

## Jagged array

An array that contains other arrays that vary in size.

## 3D array

An array that contains an array of arrays. int[][][]

## String y StringBuffer

Al instanciar un objeto de tipo String, una variable en el stack va a hacer referencia a la direccion del objeto en el heap. La cadena de caracteres del string va a estar en el heap, por ejemplo “hola”. Si yo creo una

## Metodos y variables estaticos

Lo estatico pertenece a la clase y no al objeto. Se pueden acceder a valores y metodos directamente de una clase, sin la necesidad de instanciarla. No se pueden utilizar variables de instancia en un metodo estatico.

* clases: no pueden ser estaticas menos que sean una clase dentro de otra clase (clase interna).
* variables y metodos: pueden ser estaticos, osea que se puede acceder a ellos sin la necesidad de instanciar la clase.

## Static block

El bloque estatico ejecuta codigo estatico de una clase, justo antes de ser instanciada, y solo UNA vez en todo el programa. Sirve para inicializar recursos, asignar valores a las variables estaticas, etc. Si se crea un segundo objeto de esta clase, el codigo no correrá. Si queremos ejecutar el bloque estatico sin instanciar la clase, usamos el Class.forName()

## Palabra reservada: Final (como constant en C)

Una variable, un metodo o una clase pueden ser constantes con FINAL:

* variable final: no puede ser modificada
* metodo final: no puede ser reemplazado (override) pero si puede ser sobrecargado (overloaded)
* clase final: no puede ser heredada

## Clases y Herencia

No lo voy a resumir porque ya esta muy visto, pero voy a remarcar conceptos importantes.

* java no soporta herencia multiple. Supongamos las clases independientes A,B,C. etonces C puede heredar de A o de B, pero no de ambas a la vez.
* El metodo super() es llamado implicitamente aunque no lo hayamos escrito.
* Todas las clases en java extienden la clase Object

## equals y hashCode()

* **equals**(), por defecto, calcula si dos referencias apuntan al mismo objeto en memoria, es decir, si dos variables, o dos objetos representan el mismo. Este metodo puede ser reemplazado, para comparar si dos objetos distintos, tienen las mismas variables internas, como el mismo nombre.
* **hashCode**() calcula el hash de un objeto en funcion de sus variables.

## Upcasting y Downcasting en clases

supongamos que tengo dos clases A y B que extiende A.

* upcasting: creo B y lo casteo en una referencia a una clase A. Esto se puede hacer implicitamente y es seguro, ya que B es un tipo de A.
* downcasting: creo una clase de A y lo casteo en una referencia a una clase B. Esto se debe castear de forma explicita y es inseguro, ya que A no conoce a B.

## Clases abstractas

Son clases que no se pueden instanciar por si solas, pero si se pueden extender. Pueden contener metodos comunes para sobreescribir opcionalmente, o metodos abstractos que solo estan declarados, y obligatoriamente deben ser implementados. Por ejemplo, la clase Animal, debe ser abstracta, ya que solo es posible instanciar por ejemplo, un Perro o un Gato, pero un animal, es un concepto abstracto.

El metodo abstracto se declara asi: **public abstract void method();**

## Clases internas

Una clase puede contener dentro otras clases, ya sean abstractas o concretas. Su forma de instanciacion varía dependiendo de eso:

* A a = new A();
* A.B b = new A.B(); // For static inner classes
* A.C c = a.new C(); // For non static inner classes

## Interfaces

Son estructuras, en donde se declaran metodos (semejante a una clase abstracta) solo que no son clases, y por defecto, todos los metodos declarados son publicos y abstractos. Sus variables son FINAL y STATIC por defecto.

## Extensiones recomendadas para VSCODE

Extension Pack for Java

## Configuracion recomendada para VSCODE

### Formateo automatico

in vscode:

* ctrl + shift + x
* install "Extension Pack for Java"
* Open vscode settings (Ctrl + ,).
* Search for: Format On Save
* Check option: Editor: Format On Save

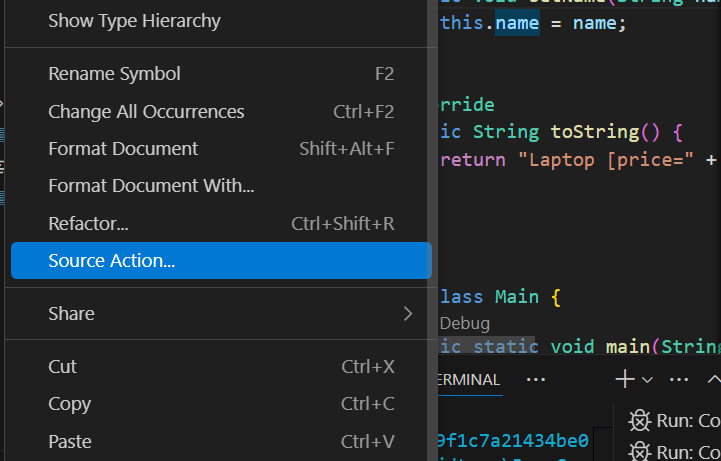
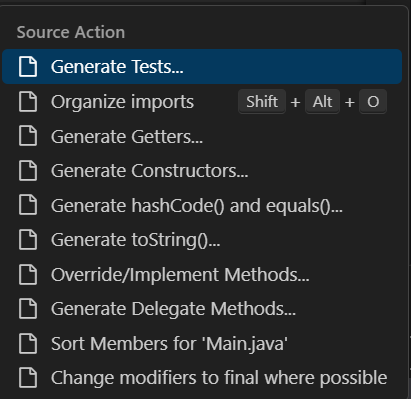
or mannualy:

download: https://github.com/google/google-java-format/releases

java -jar google-java-format-1.27.0-all-deps.jar --replace YourClass.java

## Source Action (vscode)

Nos deja realizar acciones repetitivas de manera automatica.



## Trucos / Preguntas dificiles

### 1 Caracteres

char c = ‘a’;

char result = (char) c + 1;

result vale ‘b’ porque ‘a’ es 98 en ascii, por lo que al sumarle 1 y castearlo como char, es la letra ‘b’.

### 2 Condicionales con post increment

What will be the value of variables ‘x’ and ‘y’ from the given code?

int x = 5;

int y = 10;

int z = (x++ > 5 && y-- < 10) ? x-- : y;

La respuesta es (6,10) porque al evaluar 5 > 5 es false entonces no se ejecuta el resto del condicional, luego a X se le asigna el valor de 6 por ser un post increment, y el valor de Y queda intacto.

### 2 Condicionales que modifican variables

int i, j;

i = 100;

j = 300;

while(++i < --j);

Por mas que sea un condicional, las variables SI son modificadas, por el hecho de estar usando “++” y “—“ en cambio de j + 1, en donde no se modificaria la variable.