**CLASE 1**

**CARACTERISTICAS DE JAVA**

JAVA es una tecnología pensada para desarrollo de aplicaciones de gran envergadura, altamente escalables, de gran integración con otras tecnologías y sumamente robustas

**FRONT (usuario) => CAPA INTERMEDIA o MIDDLEWARE => BACKEND (otras apps y BBDD).**

**Lenguaje orientado a objetos**

Respeta el paradigma de orientación a objetos, permitiendo utilizar los fundamentos del mismo:

● Herencia

● Polimorfismo

● Encapsulamiento

Iremos viendo estos conceptos en el transcurso de la clase.

**Sintaxis basada en C/C++**

Aporta gran simplicidad ya que es una de las formas de escribir código más reconocidas y difundidas, y permite incorporar rápidamente a los programadores que conocen este lenguaje.

**Es Multiplataforma**

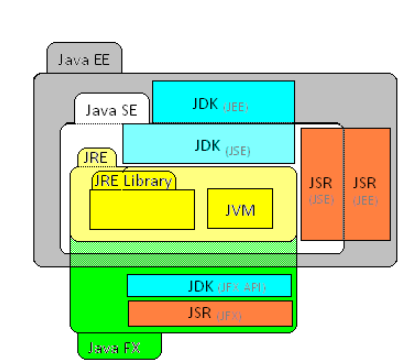
Significa que su **código** es **portable**, es decir se puede transportar por **distintas** **plataformas**. De esta manera es posible codificar una única vez una aplicación, y luego ejecutarla sobre cualquier plataforma y/o sistema operativo.

El **JRE (Java Runtime environment**) es lo **mínimo** que debe contener un **sistema** para poder **ejecutar** una **aplicación** **JAVA** sobre el mismo. Esto incluye una **JVM** (Maquina virtual de java) y un conjunto de **bibliotecas** para proporcionar los servicios que pueda necesitar la aplicación.

La **JVM** es la encargada de **traducir** (interpretar o compilar al vuelo) el **bytecode** **en** **instrucciones** **nativas** de la plataforma destino. Y el **bytecode** es el codigo **resultante** de la **compilación** del **codigo** **fuente (.java)**.

.JAVA => .class (bytecode) JVM=> EJECUTABLE (SO DESTINO).

**Plataforma de Java**



**Manejo automatico de Memoria**

No hay que preocuparse por liberar memoria manualmente ya que un proceso propio de la tecnología se encarga de monitorear, y por consiguiente eliminar el espacio ocupado que no está siendo utilizado. El proceso encargado de realizar este trabajo se denomina **Garbage** **Collector**.

**JAVA esta en Evolucion Permanente**

**JAVA esta organizado en áreas de trabajo. Comentamos brevemente JEE, JSE y JME. Y que se puede construir con cada una**

**JME (Java Micro Edition):**

Aplicaciones móviles, tales como GPS, Handhelds (por ejemplo la conocida Palm), celulares y otros dispositivos móviles programables. JME significa Java Micro Edition.

**JSE (Java Standard Edition):**

Esta área tiene como objetivo el desarrollo de aplicaciones de escritorio, similares a las aplicaciones tipo ventanas creadas con Visual Basic o Delphi. Incluye la funcionalidad básica del lenguaje como manejo de clases, colecciones, entrada/salida, acceso a base de datos, manejo de sockets, hilos de ejecución.

**JEE (Java Enterprise Edition):**

Esta área tiene como objetivo el desarrollo de aplicaciones empresariales, de gran envergadura. Contempla ambientes web, como los ambientes manejados por servidores de aplicación. Las tecnologías principales incluidas en esta área son Servlets, JSP y EJB, entre otras

**que significa JDK?**

**que incluye el compilador y la JRE?.**

**Diferencias entre JDK, JRE y JVM**

**JDK:**

Es un [software](https://es.wikipedia.org/wiki/Software) que provee [**herramientas de desarrollo**](https://es.wikipedia.org/wiki/Kit_de_desarrollo_de_software) para la creación de [programas](https://es.wikipedia.org/wiki/Programa_inform%C3%A1tico) en [Java](https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)). Puede instalarse en una [computadora](https://es.wikipedia.org/wiki/Computadora) local o en una unidad de red.

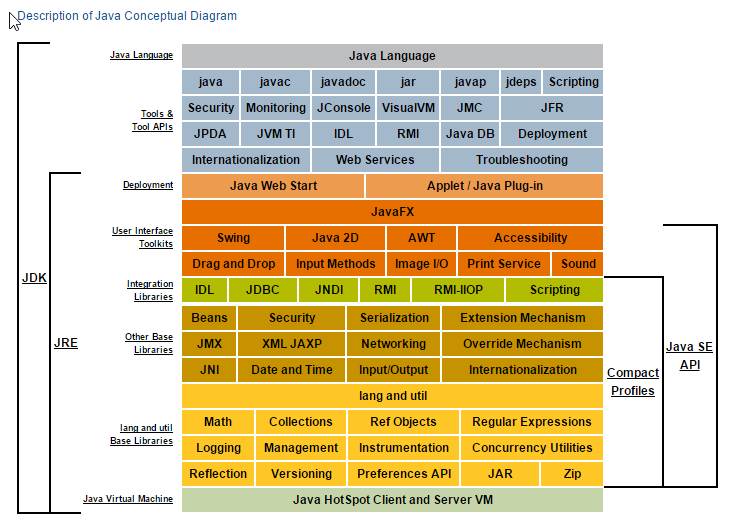
En el JDK se **incluyen** los siguientes programas:

***Javac.exe*: compilador de java**. **Que transforma los .java (codigo fuente) en .class (codigo compilado).**

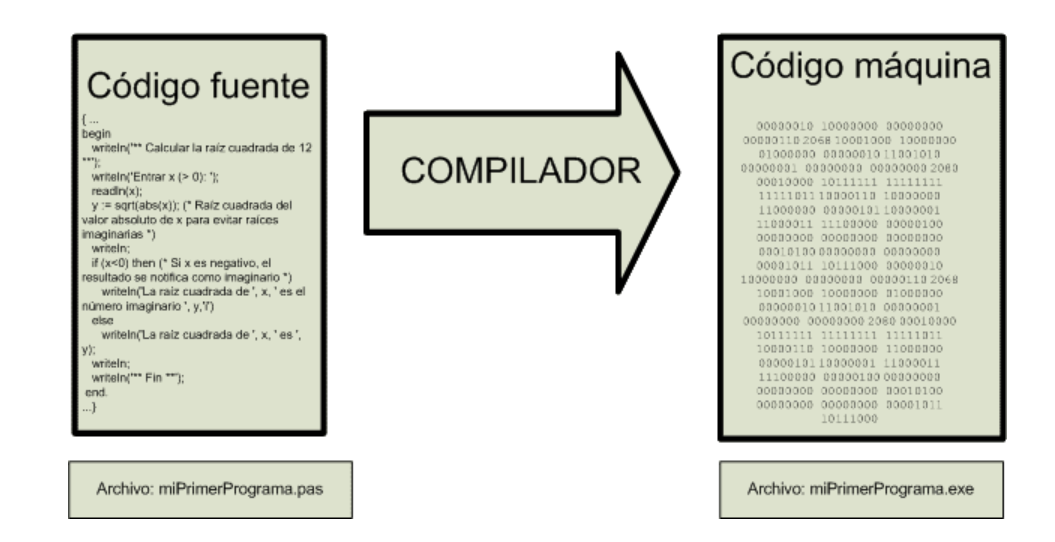
***Java.exe*: Interprete de java. Se encuentra en la JVM y es el encargado de interpretar un codigo .class (bytecode) que surge de la compilación. En un codigo de lenguaje de maquina. .JAVA > .CLASS > binario**

***Javadoc.exe*: genera la documentación** de las clases java de un programa.

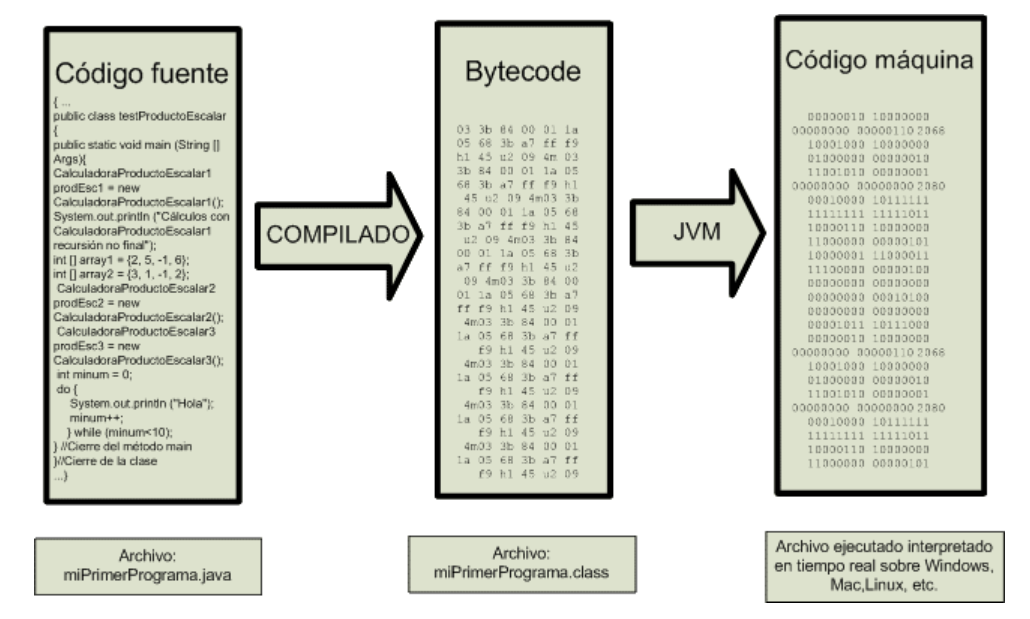
*“Compilar” significa traducir el código escrito en “Lenguaje entendible por humanos” (por ejemplo Java, C, Pascal, Fortran), a un código en “Lenguaje Máquina”, que entienden las máquinas, pero no entendible por nosotros.*



Proceso de compilación estándar (que puede ser de un lenguaje previo a Java)



La novedad es que con Java se incluye el nivel de **bytecode.**

****

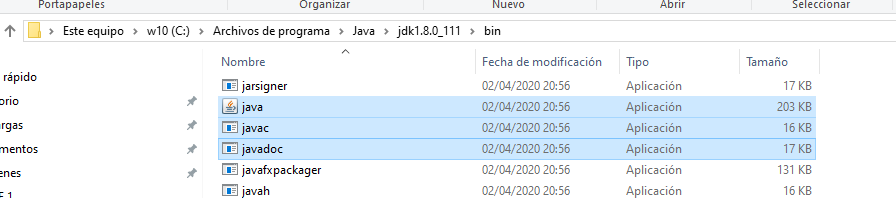
Java se hizo independiente de la plataforma añadiendo un paso intermedio: **los programas Java no se ejecutan en nuestra máquina real** (en nuestro ordenador o servidor) **sino que Java simula una “máquina virtual”** con su propio hardware y sistema operativo. En resumen, el proceso se amplía en un paso: del **código fuente**, se pasa a un **código intermedio** denominado habitualmente “**bytecode**” **entendible** **por** la **máquina virtual Java**. Y es esta máquina virtual simulada, denominada Java Virtual Machine o **JVM**, la encargada de **interpretar** el **bytecode** dando lugar a la **ejecución** del programa.

Esto permite que Java pueda ejecutarse en una máquina con el Sistema Operativo Unix, Windows, Linux o cualquier otro, porque **en realidad no va a ejecutarse en ninguno de los sistemas operativos, sino en su propia máquina virtual** que se instala cuando se instala Java. El precio a pagar o desventaja de este esquema es que todo ordenador que quiera correr una aplicación Java ha de tener instalado Java con su máquina virtual.

**SCANNER**



La ruta donde se encuentran estos archivos es:



**SINTAXIS DE JAVA – Parte 1**

**Definir variables. Inicializar antes de usarlas.**

Las variables => no pueden comenzar con un numero ni utilizar caracteres %, @.

Si pueden comenzar con \_ o $ anque este ultimo no es

Recomendable su uso.

*Ej: Declaracion de variables.*

int var;

Inicializacion de variables

var = 0;

*Declaracion con Inicializacion*

String nombre = “pepito”;

char carácter = ‘1’;

System.out.println(nombre);

**Tipos de Datos.**

**boolean**

boolean var1 = true;

if(var1){

System.out.println("Verdadera");

}

**char**

1 carácter en memoria, 2 bytes.

char var2 = 'j';

System.out.println(var2+"uan");

**String**

Si bien es una clase y no un tipo primitivo lo mencionamos porque se utiliza para definir el tipo de una variable

String variable = "Hola Mundo";

System.out.print("Hola Mundo");

**short (numerico entero)**

2 bytes en memoria. [-32768, 32767]

short numero = 32000;

short total = (short) (numero + 30);

System.out.println(total);

**int (numerico entero)**

entre -2.147.483.648 y 2.147.483.647. Ocupa 4 bytes de memoria.

int variable = 1;

int resto = 0;

resto = variable % 5;

System.out.println("Resto: "+resto);

**long (numerico entero)**

-9.223.372.036.854.775.808 y 9.223.372.036.854.775.807. Ocupa 8 bytes de memoria.

long numLargo = -9222345222345222345l;

long resultado = numLargo / 444444;

System.out.println("Resultado: "+resultado);

**float (numerico, flotante)**

32 bits

Ocupa 4 bytes de memoria, y maneja entre 6 y 7 cifras decimales.

float ventas = 20000f;

float iva = ventas \* 0.21f;

System.out.println("Iva: "+ iva);

**double (numerico, punto flotante)**

64 bits

8 bytes y maneja 15 cifras

double peso = 88.77;

System.out.println("Mi peso es: "+peso);

**Diferencia entre double y float**

Basicamente es la precisión. Double es mas preciso que el float en una misma cantidad de decimales.

**Estructura IF-ELSE**

**IF**

Me permite **saltear** una sentencia en caso de que la condición sea FALSA.

Sintaxis:

[code]

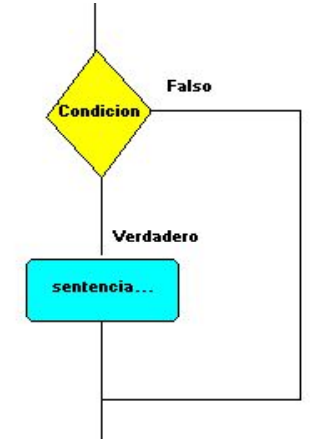
if (expresionBooleana)

{

sentencias1;

}

[/code]



int temperatura = 26;

if(temperatura > 25){

System.out.println("Vamos a la playa!!!");

} else{

System.out.println("Esperando al buen tiempo...");

}

**Operadores >, <, ==, ¡= ejemplos con IF**

int a = 20, b = 10;

String x = "Thank", y = "Thank";

int ar[] = { 1, 2, 3 };

int br[] = { 1, 2, 3 };

boolean condicion = true;

//varios operadores condicionales

System.out.println("a == b :" + (a == b));

System.out.println("a < b :" + (a < b));

System.out.println("a <= b :" + (a <= b));

System.out.println("a > b :" + (a > b));

System.out.println("a >= b :" + (a >= b));

System.out.println("a != b :" + (a != b));

// Los Arrays no se pueden comparar con

// operadores relacionales porque los objetos

// almacenan referencias, mas no el valor

System.out.println("x == y : " + (ar == br));

Estructura SWITCH con ejemplo

var1 = “árbol”;

switch (var1)

{

case “árbol”:

sentencias1;

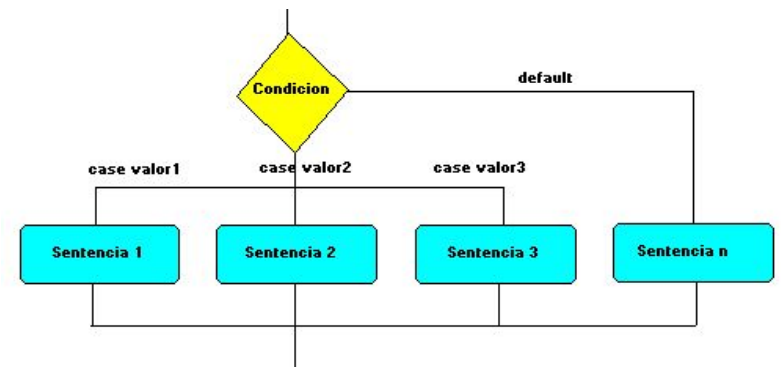
break;

case “flor”: sentencias2; break;

case “raiz”: sentencias3; break;

default: sentencias4;

}

****

**Ejercicio a:**

1. Crear una variable entera llamada dia la cual reciba el dia de hoy en numero.
2. Esto se puede hacer invocando al método getInstance() de la clase Calendar y finalmente hacer un get() cuyo parámetro sea Calendar.DAY\_OF\_WEEK.
3. Crear un Switch cuya condición sea la variable en cuestión y cuyos case representen el nro del dia.
4. Sacar en cada case el nombre de cada dia por consola.
5. Por default poner “No trajo dia”.

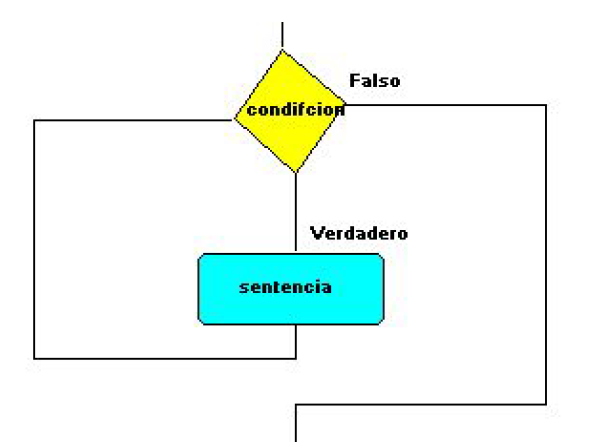
**Estructura WHILE con ejemplo**

**while** ( condición ){

sentencia ;

}

**Diagrama:**



int n = 0;

while ( n > 0 )

System.out.println("Esto nunca lo verás");

**Bucle infinito y vemos como salimos**

int a=0;

while (a<=10) {

System.out.println(a);

a--;

Para cancelar el proceso una de los opciones es

Window => IDE Tool => Process , botón derecho del mouse sobre el proceso “cancel”.

**Uso del Break dentro del While con ejemplo**

boolean prueba = true;

int cont =0;

while ( prueba ) {

System.out.println("Esto lo verás muchas veces si no estuviese el break");

if(cont==2){

break;

}

cont++;

}

**Uso del Continue dentro del while con ejemplo (b)**

1. Definir una variable de tipo short llamada var1.
2. Plantear una estructura while cuya condición evalue si la variable es menor o igual a 10.
3. Dentro del while plantear una estructura if (cuando sea menor a 5)donde incrementemos la variable var1 en 1 unidad.

En este if poner la palabra reservada continue.

1. Fuera del if sacar por consola el contenido de la variable (dentro del while).
2. Incrementar la variable en 1 unidad.
3. Cerrar bucle while.

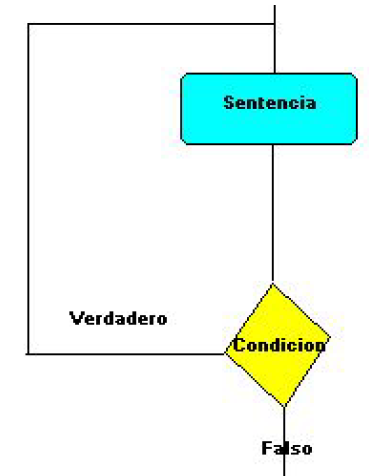
**DO-WHILE**

do

{

Sentencias;

} while (expresionBooleana);

****

do {

System.out.println("Lo veras una vez");

}while ( false );

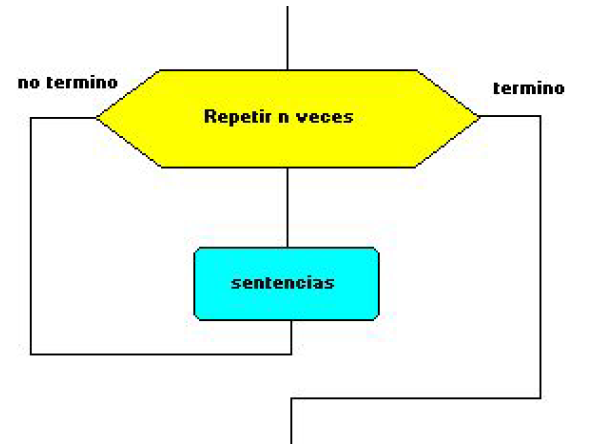
**Estructura FOR con ejemplo**

**Sintaxis**

for(inicialización; expresión booleana; post acción){

sentencias;

}



queremos mostrar los números pares (múltiplos de dos :P) entre el 500 y el 1000.

for(int i=500;i<=1000;i+= 2 ){

System.out.print(i + ", ");

}

**FOR Anidados (Horas: Minutos: Segundos) que recorre un reloj en 1 dia**

for (int hora = 0; hora < 24; hora++) {

for (int minuto = 0; minuto < 60; minuto++) {

for (int segundo = 0; segundo < 60; segundo++) {

System.out.println(hora + " : " + minuto + " : "+segundo);

}

}

**Vectores y cuando se utilizan**

En Java, un Array o (en español arreglo) es un grupo de variables llamadas elementos o componentes que contienen variables las cuales son de un **mismo tipo.**

Cuando uno **declara** un array, esta declarando una determinada cantidad de elementos del mismo tipo que se pueden **acceder**

**Array (Arreglo) : Vector – Matriz (bi-dimensional) – Matriz (n-dimensional)**

**Ej: declaración y asignación (tradicional)**

int [] losValores = new int[10];

losValores[0]= 1;

losValores[1]= 34;

losValores[2]= 191;

losValores[3]= 9878;

losValores[4]= 222;

losValores[5]= 58;

losValores[6]= -389;

losValores[7]= 0;

losValores[8]= -3902;

losValores[9]= -1;

System.out.println("El valor de la primera posición es: "+ losValores[0]);

System.out.println("El valor de la octava posición es: "+ losValores[8]);

**Ej: declaración y asignación (abreviada)**

float [] losValores = {1.5f, 0.24f, -1.1f, -8f, -99999f };

System.out.println("El valor de la primera posición es: "+ losValores[0]);

//System.out.println("El valor de la octava posición es: "+ losValores[8]); // existe?? Da un arrayIndexOutFoBoundsException

System.out.println("El valor de la quinta posición es: "+ losValores[4]);

**Recorrer un vector (c)**

1. Definir un vector llamado losCaracteres que contenga los valores a, b y 1. Tipo char.
2. Con un for recorrer el vector utilizando la propiedad length y mostrar el contenido del vector en cada posición.

**Creamos un proyecto Maven desde cero (d)**

1. Definir un vector int llamado vecNumeros que contenga los siguientes valores:

11, -22, 33, -44, 55, -66, 77, -88, 99

1. Definir una variable llamada totalPositivos.
2. Idem totalNegativos.
3. Crear un for para recorrer el vector utilizando la propiedad length.
4. Dentro del for crear una estructura IF-ELSE a través de la cual se sumen los positivos por un lado y los negativos por el otro.
5. Fuera del for imprimir el total de Positivos y el total de negativos.

Por si sale el error 501 de HTTPS dejo un tip.

<https://stackoverflow.com/questions/59763531/maven-dependencies-are-failing-with-a-501-error>

**Para que sirve y como se utiliza el vector args[] del main. Como setear los argumentos desde el Netbeans. Con ejemplo.**

El método main es un método estatico estándar de la JVM (no es palabra reservada) por el cual es punto de inicio para ejecutar el codigo en una clase determinada. Por eso es public (accesible desde cualquier lado) y static (no necesita una instancia para ejecutarse). Y posee un tipo de retorno **void** por que no tiene sentido que devuelva algo, ya que no hay codigo anterior existente al main. Por lo tanto no será necesario devolver algún valor.

Ingreso el siguiente codigo

if (args.length > 1) { //si hay más de 1 parámetro

System.out.println("Hay demasiados parámetros. ");

} else if (args.length == 0) { //si no hay parámetros

System.out.println("Falta el nombre de la persona");

} else {

System.out.println("Buenos Días " + args[0]);

}

Luego botón derecho sobre el proyecto + Properties + Run + ingresar un argumento para el main. En este caso le ingresamos “Juan”.

En esta entrada veremos para qué sirve el parámetro args del método main.

El parámetro args es un array de Strings que debe aparecer obligatoriamente como argumento del método main en un programa Java.

public static void main(String[] args){

     …

}

Aunque se le suele dar el nombre args, no es obligatorio que este parámetro se llame así, podemos darle el nombre que queramos. Por ejemplo sería valido un método main escrito así:

public static void main(String[] argumentos){

     …

}

Para qué sirve el array String [] args de main?

Sirve para recibir argumentos en formato String. Si recibe otro tipo de formato entonces es necesario realizar la conversión correspondiente luego.

Pero como en realidad el main es el inicio del codigo mas que nada cumple esa función.

**Maven**

[Maven](http://maven.apache.org/) es una herramienta open-source, que se creó en 2001 con el objetivo de **simplificar** los **procesos** de **build** (compilar y generar ejecutables a partir del código fuente). Nos **permite compilar cualquier tipo de proyecto**, librándonos de todas las dificultades que hay por detrás (ej: configurar el proceso de build, **analizar** que **partes** del **codigo** se deben **compilar**, que **librerias** utiliza el codigo, donde incluirlas, que **dependencias** de compilación hay en un proyecto). Antes de Maven, esto hacia tardar mucho tiempo el proceso de compilación.

Maven es **mucho más** que una **herramienta** que hace **builds** del **código**.  
Podríamos decir, que Maven es una herramienta capaz de **gestionar un proyecto software completo**, desde la etapa en la que se comprueba que el código es correcto, hasta que se despliega la aplicación, pasando por la ejecución de pruebas y generación de informes y documentación.

En Maven se definen tres ciclos de build del software con una serie de etapas diferenciadas. Por ejemplo el ciclo por defecto tiene las etapas de:

– **Validación** (validate): Validar que el proyecto es correcto.  
– **Compilación** (compile).  
– **Test** (test): Probar el código fuente usando un framework de pruebas unitarias.  
– **Empaquetar** (package): Empaquetar el código compilado y transformarlo en algún formato tipo .jar o .war.  
– **Pruebas de integración** (integration-test): Procesar y desplegar el código en algún entorno donde se puedan ejecutar las pruebas de integración.  
– **Verificar** que el código empaquetado es válido y cumple los criterios de calidad (verify).  
– **Instalar** el código empaquetado en el repositorio local de Maven, para usarlo como dependencia de otros proyectos (install).  
– **Desplegar** el código a un entorno (deploy).