Fundamentos – javadesde0

¿Qué aprenderás en Fundamentos de Java?

En este curso de fundamentos de Java usted aprenderá lo siguiente:

* Conocer la historia y la filosofía de Java
* Comprender la contribución de Java a Internet
* Comprender la importancia de bytecode
* Conozca las palabras de moda de Java
* Comprender los principios fundamentales de la programación orientada a objetos
* Crear, compilar y ejecutar un programa Java Simple

Historia Breve Historia de Java: Características y Aplicaciones

¿Por qué se creó Java?

En 1991, el equipo “**Green Team**” dirigido por James Gosling de Sun Microsystems creó un lenguaje de programación para dispositivos digitales de consumo. El lenguaje se llamaba **Oak**, entonces ¿**por qué Oak**? Porque había un roble (‘oak’ en inglés) afuera de la oficina de Gosling.

El “Green Team” demostró el uso del lenguaje con una televisión interactiva. Sin embargo, era demasiado avanzado para la televisión digital por cable en ese momento, y era más adecuada una tecnología que comenzaba a despegar, Internet.

Más tarde, el lenguaje pasó a llamarse “**Green**” y finalmente se le cambió el nombre a “**Java**” del café precisamente llamado ‘Java’; de ahí el logo de la taza de café.

Como C/C ++ era popular en aquel entonces, James Gosling diseñó el lenguaje con la sintaxis de estilo C/C++ y la filosofía “**escribe una vez, ejecuta en cualquier lado**”. Después de años, Sun Microsystems lanzó la primera implementación pública de Java en 1995. Se anunció que el navegador de Internet **Netscape Navigator** incorporaría la tecnología Java.

En 2010, Sun Microsystems fue completamente adquirida por Oracle Corporation junto con Java.

Historial de versiones de Java

* Junio ​​de 1991 – Se inició el proyecto de lenguaje Java
* JDK 1.0 – enero de 1996
* JDK 1.1 – febrero de 1997
* J2SE 1.2 – diciembre de 1998
* J2SE 1.3 – mayo de 2000
* J2SE 1.4 – febrero de 2002
* J2SE 5.0 – septiembre de 2004
* Java SE 6 – diciembre de 2006
* Java SE 7 – julio de 2011
* Java SE 8 – 18 de marzo de 2014
* Java SE 9 – julio de 2017

Características del lenguaje de programación

* *Java es independiente de la plataforma*
* *Un lenguaje orientado a objetos*
* *Java es rápido*
* *Java es seguro*
* *Amplio conjunto de Bibliotecas*
  + ***Java.lang*** *– para funciones avanzadas de cadenas, matrices, etc.*
  + ***Java.util*** *– para estructuras de datos, expresiones regulares, funciones de fecha y hora, etc.*
  + ***Java.io*** *– para archivos E/S, manejo de excepciones, etc.*

Aplicaciones de Java

* ***Aplicaciones de Android:*** *el lenguaje de programación Java que usa Android SDK (Kit de desarrollo de software) generalmente se usa para desarrollar aplicaciones Android.*
* ***Aplicaciones web:*** *Java se usa para crear aplicaciones web a través de Servlets, Struts o JSPs.*
* ***Desarrollo software:*** *Software como Eclipse, Open Office, Vuze, MATLAB, etc. Usan Java.*
* ***Procesamiento de Big Data:*** *Puede utilizar un marco de software popular como Hadoop (que a su vez está escrito en Java) para procesar Big Data. Para usar Hadoop, debe comprender la programación de Java.*
* ***Sistemas de negociación:*** *Puede crear aplicaciones de negociación que tengan baja latencia utilizando Oracle Extreme Java Trading Platform.*
* ***Dispositivos incorporados:*** *Las tecnologías Java Embedded de Oracle proporcionan plataformas y tiempo de ejecución para miles de dispositivos integrados como: televisores, tarjetas SIM, reproductores de discos Blu-ray, etc.*

Requisitos para Java Instalar Java y Configurar Variables de Entorno: Linux y Windows

JDK, JRE, JVM, (IDE)

* ***JDK (Java Development Kit):*** *El JDK está destinado a desarrolladores de software e incluye herramientas de desarrollo como el compilador Java, Javadoc, Jar y un depurador.*
* ***JRE (Java Runtime Enviroment):*** *El JRE contiene las partes de las bibliotecas Java necesarias para ejecutar programas JAVA y está destinado a usuarios finales. JRE se puede ver como un subconjunto de JDK.*
* ***JVM (Java Virtual Machine):*** *El JVM es una máquina abstracta. Es una especificación que proporciona un entorno de tiempo de ejecución en el que se puede ejecutar “bytecode” de Java. Las JVM están disponibles para muchas plataformas de hardware y software.*
* ***IDE:*** *Entorno de desarrollo integrado (Integrated Development Enviroment - IDE), es una aplicación que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo del software.*

Mejores IDE para Java IDE para desarrolladores Java

*(Eclipse IDE, NetBeans, IntelliJ IDEA …)*

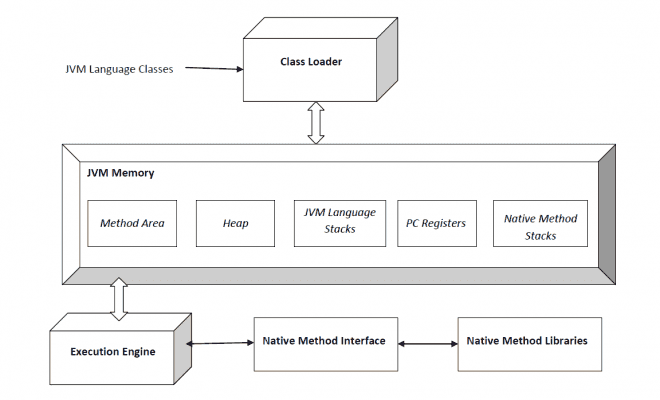
JMV en Java ¿Qué es y cómo funciona la Máquina Virtual Java?

La JVM (Java Virtual Machine) actúa como un motor de tiempo de ejecución para ejecutar aplicaciones Java. JVM es la que realmente llama al método principal presente en un código Java. JVM es una parte del JRE (Java Run Enviroment).

Las aplicaciones Java se llaman WORA (Write Once Run Everywhere). Esto significa que un programador puede desarrollar código Java en un sistema y puede esperar que se ejecute en cualquier otro sistema habilitado para Java sin ningún ajuste.

Así funciona la JVM

Cuando compilamos un archivo **.java**, el compilador Java genera un archivo **.class** (contiene byte-code) con el mismo nombre del archivo. Este archivo .class entra en varios pasos cuando lo ejecutamos. Estos pasos juntos describen toda la arquitectura JVM.



Subsistema Java Class Loader

Responsable de tres actividades:

* Carga
* Enlace
* Inicialización

Carga

El cargador de clases lee el archivo .class, **genera los datos binarios correspondientes y los guarda en el área de métodos** (“method area” –vea la imagen anterior). Para cada archivo .class, JVM almacena la siguiente información en el área de método.

1. Nombre completamente calificado de la clase cargada y su clase primaria inmediata.
2. Si el archivo .class está relacionado con Class o Interface o Enum
3. Información sobre modificadores, variables, métodos, etc.

Después de cargar el archivo .class, JVM crea un objeto de tipo Class para representar este archivo en la memoria heap. Tenga en cuenta que este objeto es de tipo Class predefinido en el paquete java. lang. Este objeto Class puede ser utilizado por el programador para obtener información de nivel de clase como nombre de clase, nombre principal, métodos e información de variable, etc. Para obtener esta referencia de objeto, podemos usar el método getClass () de la clase Object.

#### Vinculación o Enlace

Realiza la verificación, la preparación y (opcionalmente) la resolución.

* ***Verificación:*** *asegura la exactitud del archivo .class, es decir, comprueba si este archivo está formateado correctamente y generado por un compilador válido o no. Si la verificación falla, obtenemos la excepción de tiempo de ejecución****java. lang. VerifyError****.*
* ***Preparación:*** *JVM asigna memoria para las variables de clase e inicializa la memoria a los valores predeterminados.*
* ***Resolución:*** *es el proceso de reemplazar referencias simbólicas del tipo con referencias directas. Se realiza buscando en el área del método (method area) para localizar la entidad a la que se hace referencia.*

Inicialización

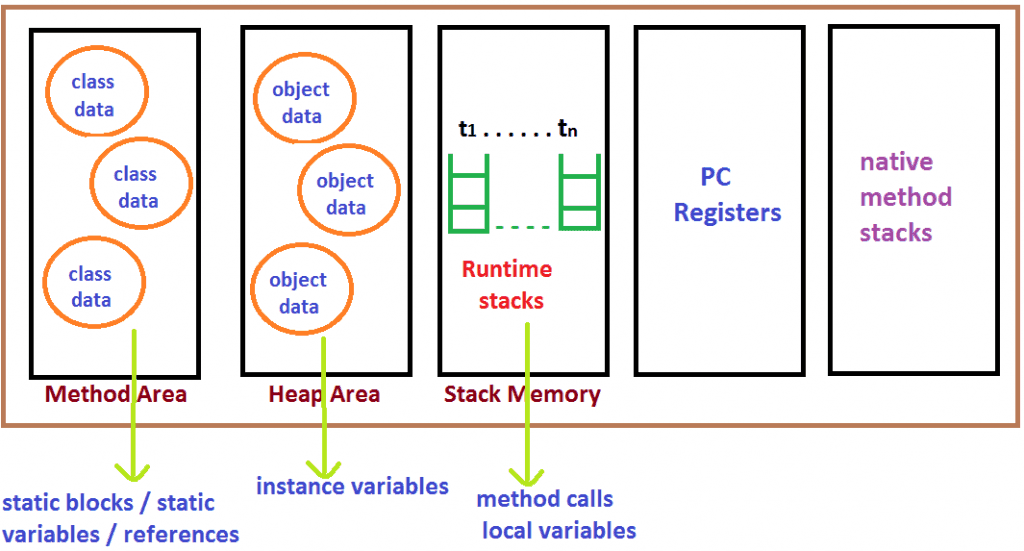
En esta fase, todas las variables estáticas se asignan con sus valores definidos en el código y en el bloque estático (si corresponde). Esto se ejecuta de arriba a abajo en una clase y de padres a hijos en la jerarquía de clases.

En general, hay tres cargadores de clase (class loaders):

* **Cargador de clases Bootstrap** **(Bootstrap class loader):** *cada implementación de JVM debe tener un cargador de clases de arranque, capaz de cargar clases confiables. Carga las clases API centrales de java presentes en el directorio JAVA\_HOME/jre/lib. Esta ruta se conoce popularmente como ruta de arranque. Se implementa en lenguajes nativos como C, C++.*
* **Cargador de clases de extensiones (Extension class loader):** *es un elemento secundario del Bootstrap class loader. Carga las clases presentes en los directorios de extensiones JAVA\_HOME/jre/lib/ext (Ruta de extensión) o cualquier otro directorio especificado por la propiedad del sistema java.ext. dirs. Está implementado en Java por la clase sun. misc. Launcher$ExtClassLoader.*
* **Cargador de clases de sistema/aplicación (system/Application class loader):** *es secundario del cargador de clases de extensión. Es responsable de cargar las clases desde la ruta de la clase de aplicación. Internamente utiliza la variable de entorno asignada a java.class. path. También se implementa en Java mediante la clase sun. misc. Launcher$AppClassLoader.*

### **Memoria JVM**

* **Área de método (Method area):** *en el área de método, se almacena toda la información del nivel de clase, como el nombre de clase, el nombre inmediato de la clase principal, la información de métodos y variables, etc., incluidas las variables estáticas. Solo hay un área de método por JVM, y es un recurso* compartido.
* **Área Heap (Heap area):** *la información de todos los objetos se almacena en el área heap. También hay un área heap por JVM. También es un recurso compartido.*
* **Área de pila (Stack area):** *para cada subproceso, JVM crea una pila en tiempo de ejecución que se almacena aquí. Cada bloque de esta pila se llama registro de activación/marco de pila que almacena los métodos de llamadas. Todas las variables locales de ese método se almacenan en su marco correspondiente. Una vez que finaliza un hilo, JVM destruirá la pila en tiempo de ejecución. No es un recurso compartido.*
* **Registros de PC (PC Registers):** *Almacena la dirección de la instrucción de ejecución actual de un hilo. Obviamente, cada hilo tiene registros de PC separados.*
* **Pilas de métodos nativos (Native method stacks):** *para cada hilo, se crea una pila nativa separada. Almacena información del método nativo.*



Motor de ejecución (Execution Engine)

El motor de ejecución ejecuta .class (bytecode). Lee el código de bytes línea por línea, usa datos e información presente en varias áreas de memoria y ejecuta instrucciones. Se puede clasificar en tres partes:

* **Intérprete:** *interpreta el bytecode línea por línea y luego lo ejecuta. La desventaja aquí es que cuando se llama un método varias veces, cada vez se requiere interpretación.*
* **Compilador Just-In-Time (JIT):** *se usa para aumentar la eficiencia del intérprete. Compila todo el bytecode y lo cambia a código nativo para que cada vez que el intérprete vea llamadas a métodos repetidos, JIT proporcione código nativo directo para esa parte, de modo que la reinterpretación no es necesario, por lo tanto, se mejora la eficiencia.*
* **Recolector de basura:** *destruye objetos no referenciados.*

Resumen Funcionamiento JVM

1. Compilar

0. Ejecutar código Java

2. Se genera un archivo “.class” (byte-code) con el mismo nombre del archivo

3. ***Carga (Primer Paso)***

4. Class Loader (Cargador de clases) lee el archivo “.class”, genera los datos correspondientes y los guarda en el área de métodos (method area).

5. Después de cargar el archivo “.class” crea un objeto de tipo class para representar este archivo en la memoria (heap).

6. ***Vinculación o enlace (Segundo Paso)***, realiza la verificación, preparación y (opcionalmente) la resolución.

* **Verificación:** el código esta formateado y generado por un compilador válido.
* **Preparación:** asigna memoria a las variables e inicializa la memoria a los valores predeterminados.
* **Resolución:** remplazar referencias *simbólicas del tipo con referencias directas. Se realiza buscando en el área del método (method area).*

7. **Inicialización (Tercer Paso)**, todas las variables estáticas se asignan con sus valores definidos en código y se ejecuta de arriba abajo y de padres a hijos en la jerarquía de clases.

JRE en Java ¿Qué es JRE, para qué sirve y cómo funciona?

Componentes del JRE

Tecnologías de implementación

Java Plug-in

Java Web Start

Deployment

Herramientas de interfaz de usuario

Java 2D

Swing

Abstract Windows Toolkit (AWT)

Print Service

Image I/O

Accessibility

Input Methods

drag and drop (DnD)

Sound

Bibliotecas de integración

Java Database Connectivity (JDBC)

Interface Definition Language (IDL)

Remote Method Invocation (RMI)

Java Naming and Directory Interface (JNDI)

Scripting

Remote Method Invocation Over Internet Inter-Orb Protocol (RMI-IIOP)

Otras bibliotecas bases

Extension Mechanism

Input/output (I/O)

Internal Support

Java Native Interface (JNI)

Java Management Extension (JMX)

Beans

Segurity

Override Mechanism

Networking

Math

Java for XML Processing (XML JAXP)

Serialization

Bibliotecas de base Lang y util

Collections

Reflection

Instrument

Versioning zip

Management

Lang y util

Loggins

Java Archive (JAR)

Concurrency Utilities

Regular Expressions

Ref Objects

Preferences API

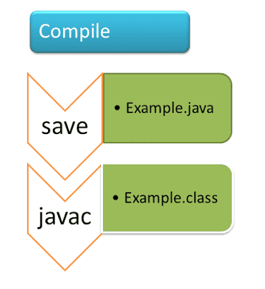
Java Virtual Machine (JVM)

Server Virtual Machines

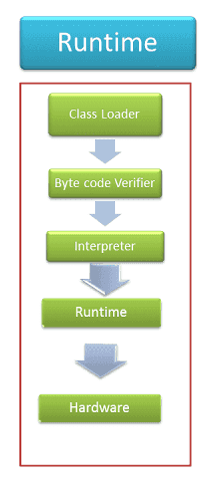
Java HotSpot Client

Cómo funciona JRE

Para comprender cómo funciona el JRE, consideremos un archivo fuente Java guardado como **Example.java**. El archivo se compila en un conjunto Byte Code que se almacena en un archivo “.class “. Aquí estará “Example.class“.



*“El siguiente diagrama muestra lo que se hace en tiempo de compilación”*



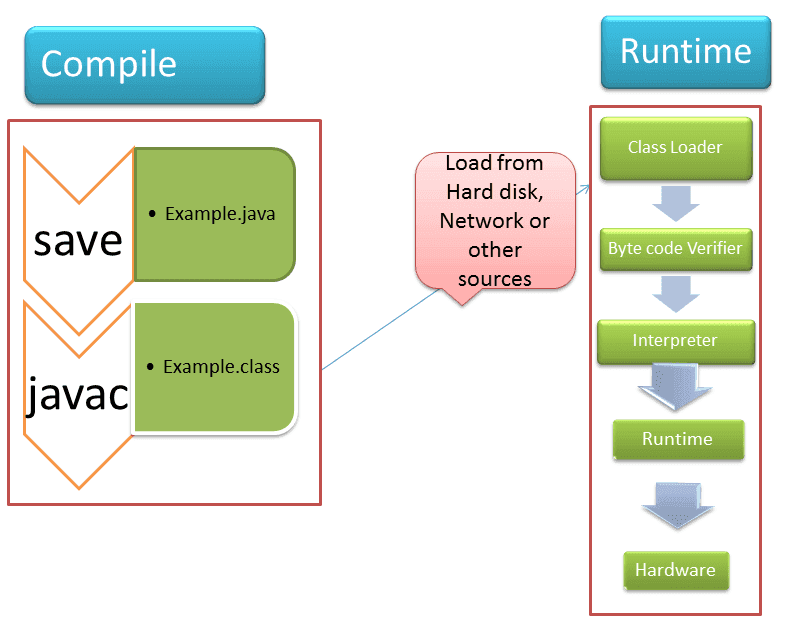
Las siguientes acciones ocurren en tiempo de ejecución.

* **Class Loader:** *El Class Loader carga todas las clases necesarias para la ejecución de un programa. Proporciona seguridad al separar los espacios de nombres del sistema de archivos local, de los importados a través de la red. Estos archivos se cargan desde un disco duro, una red o desde otras fuentes.*
* **Byte Code Verifier:** *La JVM coloca el código en Byte Code Verifier que verifica el formato y busca un código ilegal. El código ilegal, por ejemplo, es un código que infringe los derechos de acceso en los objetos o viola la implementación de los punteros.*
* **Intrepreter:** *En tiempo de ejecución, el intérprete carga, verifica y ejecuta el Byte Code. El intérprete tiene las siguientes dos funciones:*

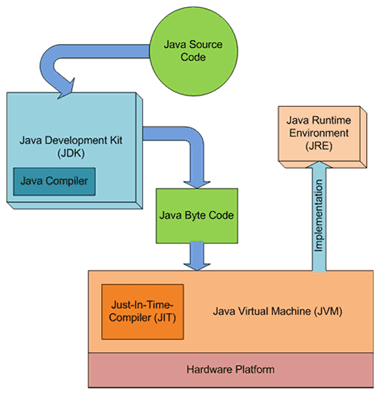
1. Ejecute el ByteCode

2. Hace llamadas apropiadas al hardware subyacente

*“Ambas operaciones se pueden mostrar cómo”*

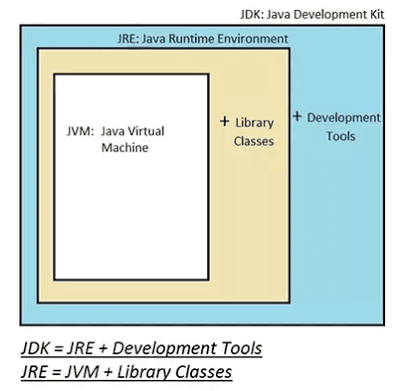


*“Para comprender las interacciones entre JDK y JRE, considere el siguiente diagrama”*



JDK, JRE y JVM Diferencias entre JDK, JRE y JVM (Indispensables en Java)

*“Para entender la diferencia entre estos tres, consideremos el siguiente diagrama”*



**JDK – Java Development Kit (en resumen, JDK):** *es un kit que proporciona el entorno para desarrollar y ejecutar el programa Java. JDK es un kit (o paquete) que incluye dos cosas:*

1. Herramientas de desarrollo (para proporcionar un entorno para desarrollar sus programas de Java)
2. JRE (para ejecutar su programa Java).

**JRE –**[**Java Runtime Enviroment**](https://javadesdecero.es/fundamentos/jre-java-runtime-environment/)**(JRE):** *es un paquete de instalación que proporciona un entorno para ejecutar (no desarrollar) el programa Java (o la aplicación) en su máquina. JRE solo lo utilizan aquellos que solo desean ejecutar los Programas Java, es decir, los usuarios finales de su sistema.*

**JVM –**[**Java Virtual Machine**](https://javadesdecero.es/fundamentos/como-funciona-maquina-virtual/)**(JVM):** *es una parte muy importante de JDK y JRE porque está contenida o incorporada en ambos. Cualquier programa Java que ejecute utilizando JRE o JDK entra en la JVM y la JVM es responsable de ejecutar el programa Java línea por línea, por lo que también se lo conoce como intérprete.*

Crear, compilar y ejecutar un programa simple Hola Mundo (Hello World) en Java

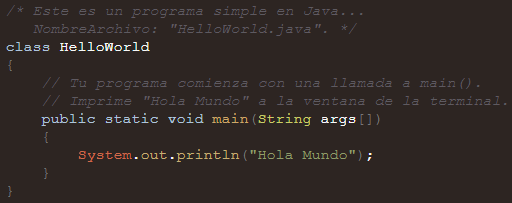
Código de Hola Mundo en Java

El siguiente programa es el más simple de impresión en Java, el famoso “Hola Mundo” o “Hello World”. Tratemos de comprender cada paso del código paso a paso.

Explicación del Código “HolaMundo”

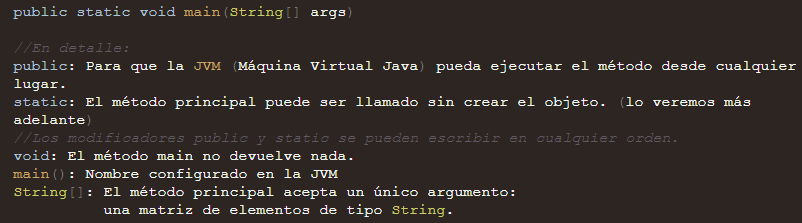
El programa “Hola Mundo” consta de tres componentes principales: la definición de la clase **HelloWorld**, el método **main** y los **comentarios** del código fuente. La siguiente explicación le proporcionará una comprensión básica del código.

* **Definición de clase:** esta línea usa la palabra clave “**class**” para declarar que se está definiendo una nueva clase.
* **Método Principal:** en el lenguaje de programación Java, cada aplicación debe contener un método “main” cuya firma sea (***public static void main (String [] args)***)



*“El método* ***main*** *es el punto de entrada para su aplicación y posteriormente invocará todos los otros métodos requeridos por su programa”*

La siguiente línea de código que se muestra es (System.out.println(“Hello World”). Esto ocurre dentro del **main ()**)

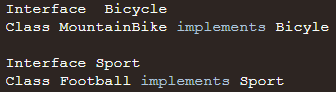


Esta línea muestra la cadena “Hello World”. La salida se logra realmente con el método integrado “**println ()**”. Bien, “System” es una clase predefinida que proporciona acceso al sistema y “**out**” es la variable del tipo flujo de salida que está conectado a la consola.

Nomenclatura Java Convenciones de Nomenclatura en Java

Clases e interfaces en Java

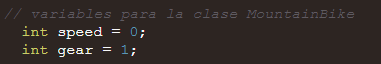
*Clases* 🡪 sustantivos (la primero letra de cada palabra en mayúscula).



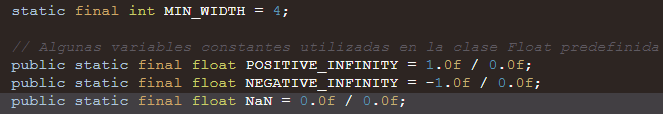
*Métodos* 🡪 verbos (en mayúscula y minúscula), con la primera letra de cada palabra interna (a partir de la segunda) en mayúscula.



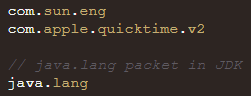
*Variables/Atributos* 🡪 cortos pero significativos, en minúscula.



*Variables Constantes* 🡪 todo en mayúscula con palabras separadas por guiones bajos (“\_”).



*Paquetes* 🡪 siempre se escribe en minúscula (palabras separas por puntos), terminar con nombres de dominio de nivel superior (con, edu, gov, mil, net, org, etc.)

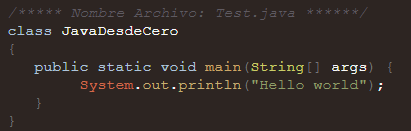


Mitos en Java Mito sobre el Nombre de archivo y Nombre de la clase en Java

*“El nombre del archivo Java y el nombre de la clase deben ser iguales”*

El mito anterior puede ser desmentido de tal forma que podemos compilar una que tiene un nombre diferente al archivo Java. ¿Cómo?

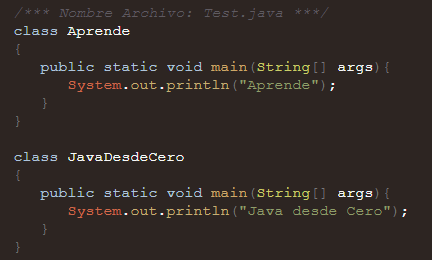
* **Paso 1:** Se crea una clase llamada “Test.class” (byte code) sin ningún mensaje de error ya que **la clase no es pública**.
* **Paso 2:** Ahora la salida será “Hello Word”



*“El mito sobre que el nombre de archivo y el nombre de la clase deben ser el mismo es* ***solo así cuando la clase se declara en public****”*

Ejemplo Práctico: Nombre Archivo ≠ Nombre Clase

Es posible tener muchas clases en un archivo Java. Para fines de depuración este enfoque puede ser utilizado. Cada clase se puede ejecutar por separado para probar sus funcionalidades (**no se debe utilizar el concepto de herencia**).

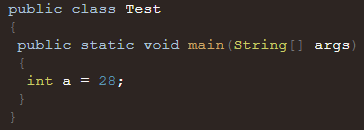


Cuando el archivo anterior se compila (Test.java) creará dos archivos “.class” (Aprende.class y JavadesdeCero.class).

* Cada clase tiene un “main” se pueden probar individualmente.
* Cuando se ejecuta la **clase aprende**, la salida es “Aprende”
* Cuando se ejecuta la **clase java desde cero**, la salida es “Java desde Cero”

Palabras Reservadas Identificadores y Palabras Reservadas en Java (Ejemplos)

En los lenguajes de programación, los identificadores (se utilizan con fines de identificación). En Java un **identificador** puede ser un “*nombre de clase*”, “*un nombre de método*” o “*un nombre de variable*”.

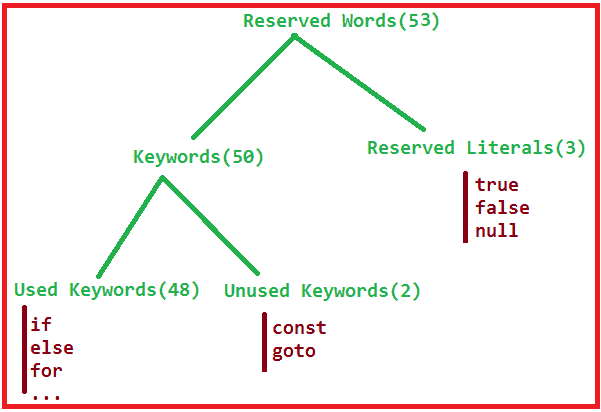


* **Test:** nombre de clase.
* **Main:** nombre del método.
* **String:** nombre de clase predefinido.
* **args:** nombre de la variable.
* **a:** nombre de la variable.

Palabras reservadas en Java

Cualquier lenguaje de programación reserva algunas palabras para representar funcionalidades definidas por ese lenguaje. Estas palabras se llaman **palabras reservadas**.

* *Palabras clave/keywords*
* *Palabras Literales* (true, false, null)



***Todas las Palabras reservadas en Java***

default

continue

const

class

char

catch

case

byte

break

boolean

assert

abstract

false

extends

enum

else

double

do

if

goto

for

float

finally

final

new

native

long

instanceof

import

implements

return

public

protected

private

package

null

switch

super

String

strictfp

static

short

this

synchronized

true

transients

throws

throw

requires

module

open

while

volatile

void

to

uses

while

provides

opens

exports

transitives

*“****Las palabras clave no se pueden usar como nombres para una variable, clase o método****”*

Java Shell (JShell) Introducción a JShell