



# Lenguajes Compilados vs Interpretados

# Lenguajes compilados vs Interpretados

Objetivos del tema:

- Conocer conceptos básicos de los lenguajes compilados
- Conocer conceptos básicos de los lenguajes interpretados
- Qué tipo de lenguaje es Java
- Entender la integración de Scala con Java



# 1 Lenguajes compilados



# Lenguajes compilados

Son aquellos cuyo código fuente es:

- escrito en un lenguaje de alto nivel por un programador.
- traducido por un compilador a un archivo ejecutable entendible para la máquina en determinada plataforma.
- con el archivo ejecutable se puede ejecutar el programa cuantas veces sea necesario sin tener que repetir el proceso por lo que el tiempo de espera entre ejecución y ejecución es ínfimo.
- Algunos lenguajes de puramente compilados serían: C, C++, Pascal, Haskell

Además, los lenguajes compilados tienen un tipado estático, esto quiere decir que cada variable tiene que ser definida con un tipo de datos asociado a ella y este tipo no puede cambiar.

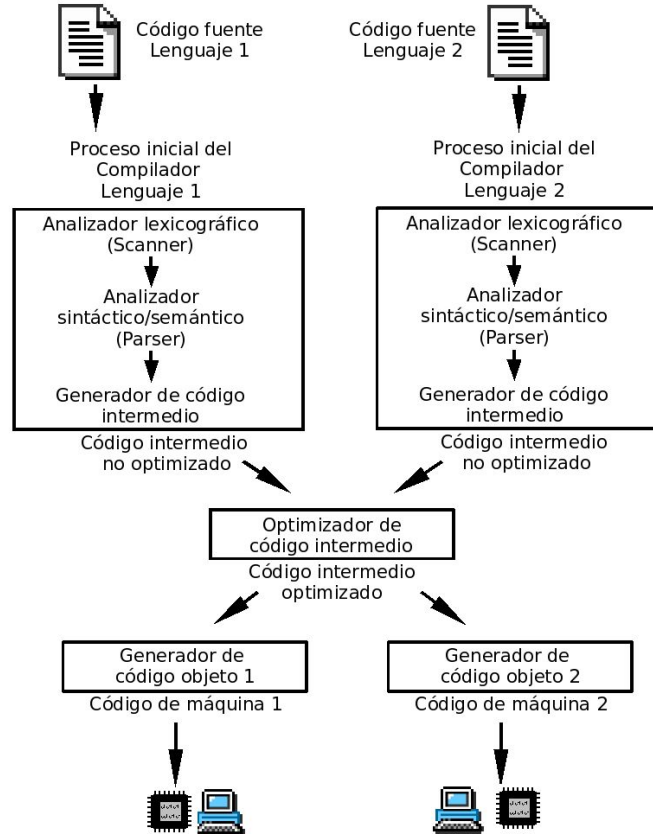


# Lenguajes compilados

Todo lenguaje compilado debe pasar por una compilación donde:

- el compilador lee el fichero una o varias veces.
- existen compiladores de una pasada o de varias pasadas, esto quiere decir que algunos compiladores solo necesitan leer el código fuente una vez, otros compiladores necesitan leer el código fuente dos o más veces.
- los compiladores que leen más de una vez código fuente se debe a que separan los procesos, por ejemplo:
  - la primera vez que leen el código fuente verifican la sintaxis del código
  - en la segunda vez que revisan código comprueban la lógica
- además de verificar la sintaxis, la lógica, se pueden hacer optimizaciones en el código antes de generar el código objeto (archivo ejecutable).

# Lenguajes compilados



## 2 Lenguajes interpretados

# Lenguajes interpretados

Son aquellos en los cuales sus instrucciones o más bien el código fuente es:

- escrito por el programador en un lenguaje de alto nivel, es traducido por el intérprete a un lenguaje entendible para la máquina paso a paso, instrucción por instrucción. El proceso se repite cada vez que se ejecuta el programa.
- Haciendo una comparación, se podría decir que un lenguaje interpretado funciona como un traductor de idioma, véase google translator, donde por un lado se escribe en un idioma de entrada (código fuente en un lenguaje de programación de alto nivel) y devuelve su traducción a otro idioma de salida (código fuente traducido a lenguaje máquina) el cual se ejecuta.

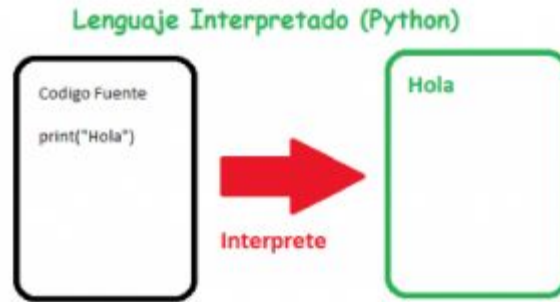
Los lenguajes interpretados permiten el tipado dinámico de datos, es decir, no es necesario inicializar una variable con determinado tipo de dato sino que esta puede cambiar su tipo en condicion al dato que almacena entre otras características más.

Algunos lenguajes interpretados: Python, Javascript, Perl, MATLAB





# Lenguajes interpretados



### 3 Comparativa compilado e interpretado

# Comparativa compilado e interpretado

Diferencias entre compilados e interpretados:

- interpretados:

- tipado dinámico
- desarrollo más rápido: escribir-ejecutar en lugar de escribir-compile-ejecutar.
- hace falta tener instalado el intérprete en la plataforma donde vaya a ejecutarse

- compilados:

- tipado estático
- ejecución más rápida mejor performance: se debe a que el código ya está en código objeto e incluye algunas optimizaciones hechas por el compilador.
- no es multiplataforma: **excepto Java**



## 4 Java; es compilado e interpretado

# Java; es compilado e interpretado

Java es un lenguaje particular porque es compilado, pero es compilado a un lenguaje intermedio llamado bytecode, que después es interpretado.

Los código fuentes de Java se compilan con la JDK (Java Development Kit).

El código intermedio generado por el proceso de compilación de Java (ByteCode) con la JDK es interpretado por la JRE (Java Runtime Environment).

Los creadores de Java querían crear un lenguaje compilado, pero que se pudiera ejecutar en cualquier sistema operativo y procesador sin necesidad de crear varios ejecutables.

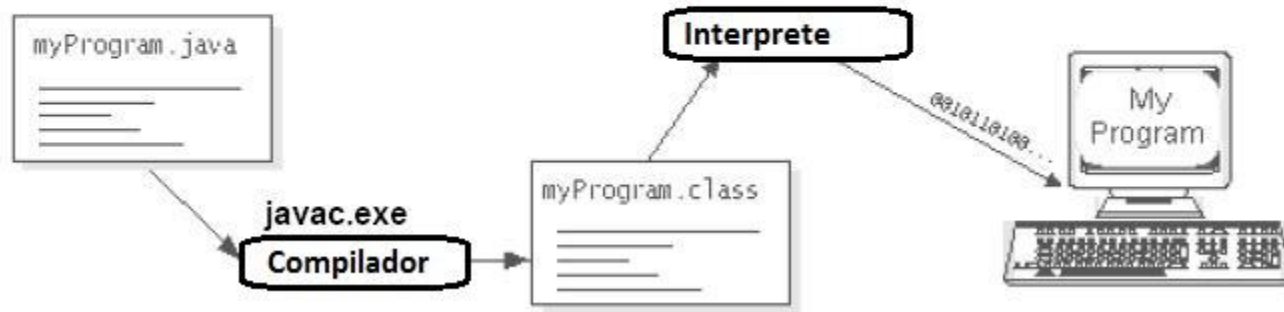
Este bytecode es multiplataforma debido a que corre sobre una máquina virtual: Java Virtual Machine.

Es capaz de correr sobre cualquier sistema operativo que tenga disponible una JVM.



# Java; es compilado e interpretado

## PROCESO DE COMPILACION Y EJECUCION EN JAVA



# 5 Scala sobre Java

# Scala sobre Java

Como ya sabemos Scala es un lenguaje de programación de propósito general que soporta la programación funcional y a un sistema de tipado estático fuerte.

El código fuente de Scala es compilado en bytecode de Java, para que el código ejecutable resultante se ejecute en una máquina virtual de Java.

Para hacer una comparativa y ver las similitudes entre ambos lenguajes, veamos el típico programa, HolaMundo en ambos lenguajes:

```
// scala
object HolaMundo {
  def main(args: Array[String]) {
    println("¡Hola, mundo!")
  }
}
```

```
// java
class HolaMundo {
  public static void main(String[] args) {
    System.out.println("¡Hola, mundo!");
  }
}
```



# Scala sobre Java

La estructura del programa es similar en ambos lenguajes, Scala y Java:

- Para definir un programa en Scala se declara un Objeto y en Java se declara una Clase.
- El objeto declarado en Scala es lo que se conoce como objeto singleton, clase con una sola instancia.
- en Java se usa ';' para indicar el final de una sentencia, pero Scala detecta el final de sentencia con el salto de línea.
- en ambos se tiene un método main que toma como argumentos un array de objetos String pasados como argumentos de línea de comandos. Pero la diferencia entre la definición de Scala y Java reside en que en Java se declara este método como static, pero en Scala no, esto sucede porque en Scala no existen miembros estáticos. En vez de definir miembros estáticos como en Java, se declaran los miembros en un objeto singleton.
- en el cuerpo del método se observa una llamada al método predefinido println al que se le pasa un String con un saludo. Pero Scala importa por defecto librerías y no hace falta referenciarlas como en Java, por ejemplo la librería System.out:
  - En Java: `System.out.println("¡Hola, mundo!");`
  - En Scala: `println("¡Hola, mundo!")`
- el método main no devuelve ningún valor.



# Scala sobre Java

Veamos un ejemplo de compilación

- Para compilar el ejemplo utilizaremos el compilador de Scala, `scalac`. `scalac` funciona como la mayoría de los compiladores; toma un archivo fuente como argumento, algunas opciones y produce uno o varios archivos objeto. Los archivos objeto que produce son archivos `.class` como los que produce Java y su compilador `javac`.
- Si se ha creado un fichero de ejemplo `HolaMundo.scala` con el contenido del ejemplo anterior, podemos compilarlo ejecutando desde una consola de comando:

```
scalac HolaMundo.scala
```

- Esto generará algunos archivos `class` en el directorio actual. Uno de ellos se llamará `HolaMundo.class` y contiene una clase que puede ser directamente ejecutada utilizando el comando `scala`.

```
cflores@cflores:~/scala$ scalac HolaMundo.scala
warning: 1 deprecation (since 2.13.0); re-run with -deprecation for details
1 warning
cflores@cflores:~/scala$ ls
HolaMundo.class  'HolaMundo$.class'  HolaMundo.scala
```



# Scala sobre Java

Una vez compilado, un programa Scala puede ser ejecutado utilizando el comando `scala`. Su uso es muy similar al comando `java` utilizado para ejecutar programas Java, y acepta las mismas opciones. El ejemplo puede ser ejecutado utilizando el siguiente comando:

```
scala -classpath . HolaMundo
```

Teniendo como resultado, el saludo que definimos en el programa:

```
cflores@cflores:~/scala$ scala -classpath . HolaMundo
¡Hola, mundo!
cflores@cflores:~/scala$
```

## 6 Interacción de Scala con Java

# Interacción de Scala con Java

Una de las fortalezas de Scala es que hace muy fácil interactuar con código Java.

Todas las clases del paquete `java.lang` son importadas por defecto, mientras otras necesitan ser importadas explícitamente.

Si vemos el siguiente ejemplo, donde queremos obtener y formatear la fecha actual según las convenciones por un país específico, España, por ejemplo.

Las librerías de Java definen clases como `Date` y `DateFormat`, las cuales se pueden importar directamente desde Scala, con esto se evita tener que implementar clases equivalentes en Scala.



# Interacción de Scala con Java

Veamos un ejemplo de cómo se integra Scala con Java:

En el ejemplo vemos que:

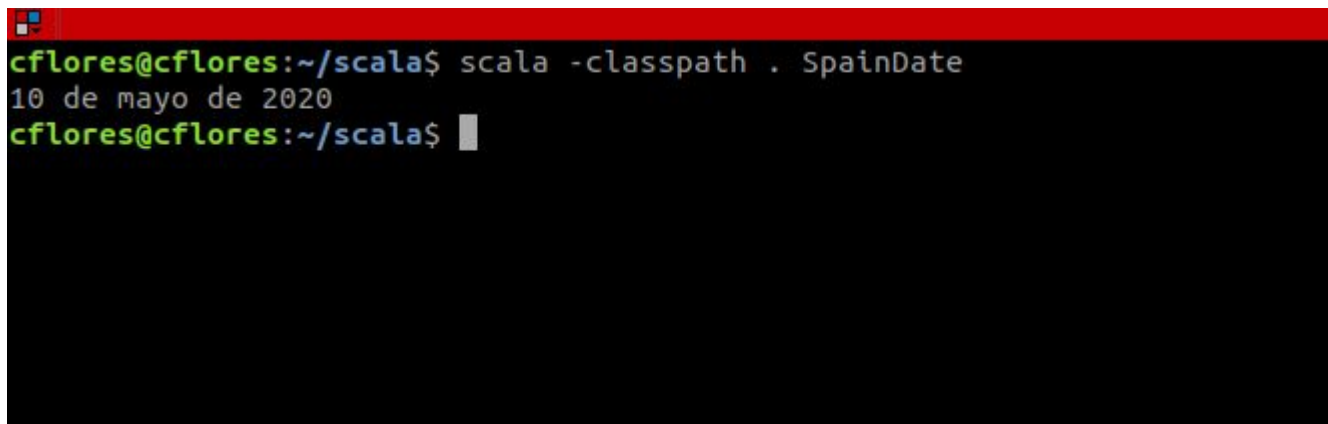
- Las declaraciones de importación de Scala son muy parecidas a las de Java, pero añaden funcionalidad extra:
  - se pueden importar múltiples clases de un mismo paquete al encerrarlas entre llaves: `import java.util.{Date, Locale}`
  - también se puede importar todas las clases contenidas en un paquete o todos los miembros de una clase usando el guión bajo (`_`): `import java.text.DateFormat._`
- Dentro del método `main` primero creamos una instancia de la clase `Date` la cual por defecto contiene la fecha actual.
- A continuación definimos un formateador de fechas utilizando el método estático `getDateInstance` que importamos previamente.
- Finalmente, imprimimos la fecha actual formateada de acuerdo a la instancia de `DateFormat` que fue "localizada".
- Esta última línea muestra una propiedad interesante de la sintaxis de Scala. Los métodos que toman un solo argumento pueden ser usados con una sintaxis de infijo.
  - Es decir, la expresión: `df format ahora` es sólo otra manera más corta de escribir: `df.format(ahora)`

```
import java.util.{Date, Locale}
import java.text.DateFormat._

object SpainDate {
  def main(args: Array[String]) {
    val ahora = new Date
    val spainLocale = new
Locale("es", "ES")
    val df = getDateInstance(LONG,
spainLocale)
    println(df format ahora)
  }
}
```

# Interacción de Scala con Java

Si compilamos y ejecutamos el programa del ejemplo anterior, debemos obtener una salida como la siguiente:

A screenshot of a terminal window with a black background and a red title bar. The prompt is 'cflores@cflores:~/scala\$'. The command 'scala -classpath . SpainDate' has been executed, resulting in the output '10 de mayo de 2020'. The prompt is now 'cflores@cflores:~/scala\$' with a cursor.

```
cflores@cflores:~/scala$ scala -classpath . SpainDate
10 de mayo de 2020
cflores@cflores:~/scala$
```

