Mi primera aplicación en Java

Primeros Pasos

Empezando con Java Me siento ante el ordenador y pienso que voy a escribir mi primera aplicación Java. ¿Cómo? ¿Qué necesito? ¿Por dónde empiezo?…… (las dudas me asaltan). No te preocupes, para eso estamos nosotros.

Antes de crear mi primer programa en Java deberé de asegurarme que tengo en el equipo el siguiente software:

- **Un editor de textos** (Por ejemplo, el bloc de notas de Windows, Wim, AM-Notebook, Win32Pad, EditPad Lite, NotePad2,…)
- **El compilador de Java**

Supongo que el primero, por descontado, lo tendréis a mano. O, al menos, algo similar. Para los usuarios avanzados en el tema les dejare utilizar el **Atom**, **Sublime** y similares.

El compilador de Java será el que nos permita transformar nuestro código fuente en programas ejecutables. O…. bueno, podríamos decir que en algo similar a programas ejecutables. Ya veremos en que.

Siguiendo los pasos que se explican en el artículo Cómo instalar Java podremos tener nuestro entorno preparado para poder desarrollar nuestra primera aplicación Java

Hola Mundo en Java

Ahora que tenemos todo el entorno de desarrollo instalado nos lanzamos a desarrollar, ni más, ni menos, que nuestra primera aplicación Java.

Lógicamente, nuestra primera aplicación no podría ser otra que “Hola Mundo”. Por si algún despistado todavía no se ha enterado de que va esta aplicación, decirle, simplemente, que es mostrar por pantalla la frase “Hola Mundo”. Complejo, ¿verdad?.

El código de nuestra primera aplicación Java es el siguiente:

```
public class MiPrimeraAplicacion {
  public static void main (String[] args) {
    System.out.println("Hola Mundo");
  }
}
```

El fichero lo guardaremos como **MiPrimeraAplicacion.java**. Este será nuestro fichero con el código fuente.

> Deberemos de tener cuidado en cómo escribimos el nombre del fichero ya que Java es un **lenguaje sensible a mayúsculas**, es decir, que no es lo mismo poner miprimeraaplicacion o MiprimeraAplicacion o MIPRIMERAAPLICACION o … El nombre del fichero deberá de coincidir con el nombre de la clase principal.

```
public class MiPrimeraAplicacion <--> MiPrimeraAplicacion.java
```

La verdad es que a estas alturas de la película no nos vamos a centrar en que significa cada una de las líneas de código.

Si bien, no es que haya que ser muy listo, para, al menos, darnos cuenta de que con la sentencia System.out.println se pueden volcar contenidos a la pantalla del ordenador.

Uso del compilador javac

El compilador de Java se llama **javac** (la c es de compilador, claro). Este no deja de ser un programa ejecutable como otro cualquiera.

En el caso de estar en un sistema operativo Windows, el compilador suele estar instalado (si hemos seguido la instalación por defecto) en:

```
C:\\Program Files\\Java\\jdk1.8.0_51.jdk\\bin
```

Si estamos trabajando con un MacOS podemos ejecutar el comando.

```
/usr/libexec/java_home
```

El cual nos indicará en qué directorio se encuentra instalado Java.

Lo que ya podemos aventurarnos a ejecutar el compilador. Para ello ejecutaremos el programa:

```
javac
```

Al ejecutar el compilador veremos por pantalla algo así:

```
Usage: javac <options> <source files>
where possible options include:
  -g                         Generate all debugging info
  -g:none                    Generate no debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -nowarn                    Generate no warnings
  -verbose                   Output messages about what the compiler is doing
  -deprecation               Output source locations where deprecated APIs are used
  -classpath <path>          Specify where to find user class files and annotation processors
  -cp <path>                 Specify where to find user class files and annotation processors
  -sourcepath <path>         Specify where to find input source files
  -bootclasspath <path>      Override location of bootstrap class files
  -extdirs <dirs>            Override location of installed extensions
  -endorseddirs <dirs>       Override location of endorsed standards path
  -proc:{none,only}          Control whether annotation processing and/or compilation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypasses default discovery process
  -processorpath <path>      Specify where to find annotation processors
  -parameters                Generate metadata for reflection on method parameters
  -d <directory>             Specify where to place generated class files
  -s <directory>             Specify where to place generated source files
  -h <directory>             Specify where to place generated native header files
  -implicit:{none,class}     Specify whether or not to generate class files for implicitly referenced files
  -encoding <encoding>       Specify character encoding used by source files
  -source <release>          Provide source compatibility with specified release
  -target <release>          Generate class files for specific VM version
  -profile <profile>         Check that API used is available in the specified profile
  -version                   Version information
  -help                      Print a synopsis of standard options
  -Akey[=value]              Options to pass to annotation processors
  -X                         Print a synopsis of nonstandard options
  -J<flag>                   Pass <flag> directly to the runtime system
  -Werror                    Terminate compilation if warnings occur
  @<filename>                Read options and filenames from file
```

Uff… vaya cantidad de opciones… No te preocupes por ellas, ya que para compilar mi aplicación deberé de poner por consola lo siguiente…

```
javac MiPrimeraAplicacion.java
```

Esta ejecución supone que tenemos el código fuente en el mismo directorio que el compilador, si bien, eso no será lo más corriente.

Configurando en Path para Java

Para poder ejecutar el compilador en cualquier directorio de máquinas Windows deberemos de insertar el directorio donde se ubica el compilador en la **variable de entorno PATH**.

Para ello, escribiremos lo siguiente….

```
SET PATH = %PATH%;C:\\Program Files\\Java\\jdk1.8.0_51.jdk\\bin
```

Ahora podremos ejecutar el compilador desde cualquier sitio. Así, debería de funcionarnos lo siguiente…

```
C:\\WORK\\Ejemplos1\\javac MiPrimeraAplicacion.java
```

Compilando mi primera aplicación Java

Si hemos compilado de forma correcta nuestro programa, simplemente la respuesta por pantalla será la siguiente:

```
victor$ javac MiPrimeraAplicacion.java
victor$
```

Vamos que si no nos dice nada de nada es que lo hemos hecho muy bien. En el caso de que hubiéramos metido la pata saldrían cosas como las siguientes…

```
LineaDeCodigo:mi-primera-aplicacion-java victor$ javac MiPrimeraAplicacion.java
MiPrimeraAplicacion.java:1: error: class miprimeraaplicacion is public, should be declared in a file named miprimeraaplicacion.java
public class miprimeraaplicacion {
       ^
1 error
LineaDeCodigo:mi-primera-aplicacion-java victor$
```

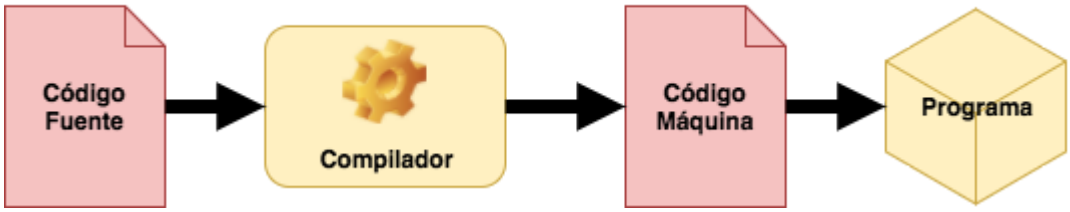
Esto es que el nombre de la clase y del fichero no existe. Múltiples errores se nos pueden producir.

Ejecutando mi aplicación

Una vez que hemos ejecutado correctamente la compilación, sorpresa, no obtenemos un fichero ejecutable, es decir, un .exe o similar. Si no que obtenemos un fichero **.class**. En este caso obtendremos un fichero **MiPrimeraAplicacion.class**.

Java es un lenguaje multiplataforma que está construido bajo el principio de *"write once, run anywhere"*. Esto quiere decir que, una vez creado el fichero fuente y compilado, el resultado (llamémoslo, de momento, nuestro pseudo-fichero ejecutable) lo podemos ejecutar en cualquier otro ordenador.

Revisemos algún concepto sobre compiladores. En un proceso de compilación normal seguimos los siguientes pasos:



Esto nos viene a decir que si yo compilo un programa, por ejemplo, en C, en mi máquina Windows 8.1 sobre una plataforma Intel. Solo va a funcionar en maquinas con esa configuración.

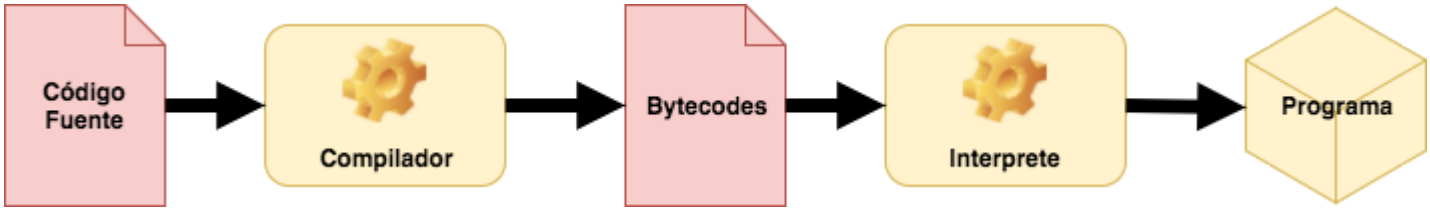
Si yo llevo mi programa a una máquina con UNIX en una plataforma Solaris no me va a funcionar. ¿Qué hace Java para que eso pueda hacerse?.

Java, más concreto, en los lenguajes interpretados, el compilador genera un código intermedio (más o menos legible).

En el caso de Java, el código intermedio se llama **bytecodes**. Este código no es dependiente ni del sistema operativo ni de la máquina en el cual lo ejecutamos.

En un segundo paso, un interprete, ejecutará dichos **bytecodes** en la plataforma que queramos. Es decir, que el interprete ya es especifico del sistema operativo y de la plataforma de ejecución.

El esquema quedaría de la siguiente forma…



Centrándonos, nuevamente, en nuestra aplicación, encontraremos un fichero **MiPrimeraAplicacion.class** que será el fichero con los **bytecodes**.

El programa que va a ejecutar dichos **bytecodes** es java. Este programa está en el mismo directorio en el que estaba el compilador.

Volvamos a arriesgarnos y ejecutemos el compilador. Recordad que al tener el directorio en la variable de entorno PATH podremos estar en cualquier directorio.

```
java
```

Este, tiene más opciones que el compilador…

```
Sintaxis: java [-options] class [args...]
          (para ejecutar una clase)
   o  java [-options] -jar jarfile [args...]
          (para ejecutar un archivo jar)
donde las opciones incluyen:
    -d32          usar un modelo de datos de 32 bits, si está disponible
    -d64          usar un modelo de datos de 64 bits, si está disponible
    -server       para seleccionar la VM "server"
                  La VM por defecto es server,
                  porque la ejecución se está llevando a cabo en una máquina de clase de servidor.


    -cp <ruta de acceso de búsqueda de clases de los directorios y los archivos zip/jar>
    -classpath <ruta de acceso de búsqueda de clases de los directorios y los archivos zip/jar>
                  Lista separada por : de directorios, archivos JAR
                  y archivos ZIP para buscar archivos de clase.
    -D<nombre>=<valor>
                  definir una propiedad del sistema
    -verbose:[class|gc|jni]
                  activar la salida verbose
    -version      imprimir la versión del producto y salir
    -version:<valor>
                  es necesario que se ejecute la versión especificada
    -showversion  imprimir la versión del producto y continuar
    -jre-restrict-search | -no-jre-restrict-search
                  incluir/excluir JRE privados de usuario en la búsqueda de versión
    -? -help      imprimir este mensaje de ayuda
    -X            imprimir la ayuda sobre las opciones que no sean estándar
    -ea[:<nombre_paquete>...|:<nombre_clase>]
    -enableassertions[:<nombre_paquete>...|:<nombre_clase>]
                  activar afirmaciones con la granularidad especificada
    -da[:<nombre_paquete>...|:<nombre_clase>]
    -disableassertions[:<nombre_paquete>...|:<nombre_clase>]
                  desactivar afirmaciones con la granularidad especificada
    -esa | -enablesystemassertions
                  activar afirmaciones del sistema
    -dsa | -disablesystemassertions
                  desactivar afirmaciones del sistema
    -agentlib:<nombre_bib>[=<opciones>]
                  cargar la biblioteca de agente nativa <nombre_bib>, como -agentlib:hprof
                  véase también -agentlib:jdwp=help y -agentlib:hprof=help
    -agentpath:<nombre_ruta_acceso>[=<opciones>]
                  cargar biblioteca de agente nativa con el nombre de la ruta de acceso completa
    -javaagent:<ruta_acceso_jar>[=<opciones>]
                  cargar agente de lenguaje de programación Java, véase java.lang.instrument
    -splash:<ruta_acceso_imagen>
                  mostrar una pantalla de presentación con la imagen especificada
Consulte http://www.oracle.com/technetwork/java/javase/documentation/index.html para obtener más información.
```

Para ejecutar nuestra aplicación escribiremos

```
java MiPrimeraAplicacion
```

Ahhhhhhhhhhhhhhh…… ya me lo he cargado … **Exception in thread “main” java.lang.NoClassDefFoundError** ¡y yo con estos pelos!

Es normal que la primera vez que ejecutemos nos pueda suceder esto. A si que no nos preocupemos.

Esto sucede debido a que el interprete java busca los ficheros .class en los directorios que define la variable de entorno **CLASSPATH**.

Es por ello que si queremos ejecutar una clase que esta en el directorio actual deberemos de tener, al menos, dicho directorio en la variable de entorno.

Cuando escribamos aplicaciones más grandes utilizaremos clases creadas por Java, a si que deberemos de tener en el CLASSPATH la ruta de dichas clases. Para solucionar todo este embrollo podemos escribir lo siguiente.

```
set CLASSPATH=.
```

Notar que el punto hace referencia al directorio actual.

Si tu eres una de esas personas que no puede dejar nada fuera de control, te recomiendo que te leas el documento JDK Installation for Microsoft Windows . Todo lo que siempre quisiste saber sobre la variable **CLASSPATH** y nunca te atreviste a preguntar. :-)

Ahora, ya si que podremos ejecutar nuestra aplicación. Al fin, el resultado esperado…

```
LineaDeCodigo:mi-primera-aplicacion-java victor$ java MiPrimeraAplicacion
Hola Mundo
```