

Paso a paso con la Máquina Virtual de Java (JVM)



Ellen Pimentel

7 días atrás

Paso a paso con la JVM

El ecosistema de la Java es una sopa de letras llena de siglas, y la JVM (Java Virtual Machine) es una de ellas. Además de ser la base de la plataforma Java, también fue una de las razones para la popularización del lenguaje.

Y, ¿cómo funciona exactamente? Vamos a averiguar paso a paso.

Tip: Para realizar la actividad no basta con tener instalado **Java**, es necesario tener configuradas las variables de entorno como el *JAVA_HOME*, el *PATH*, y el *CLASS_PATH*.

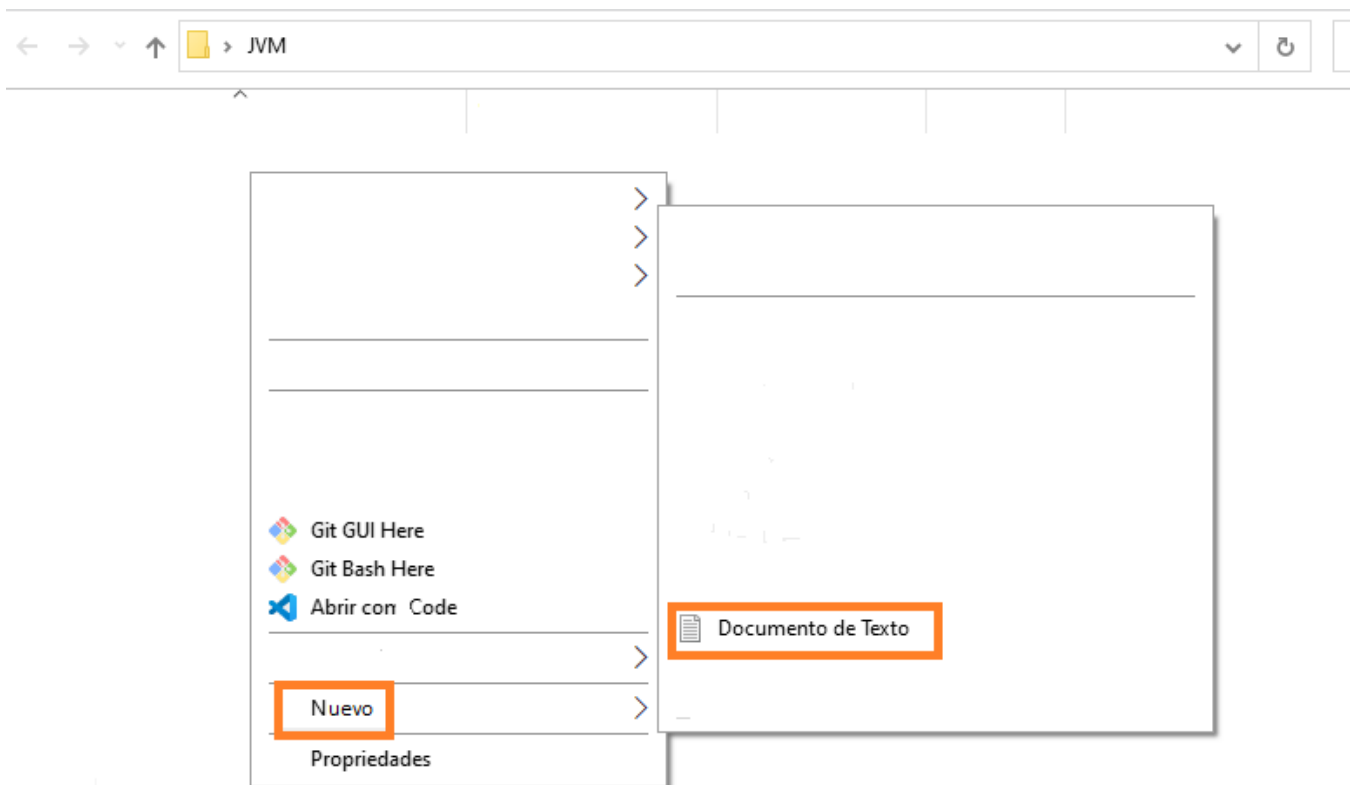
¡Hora de practicar!

Hay varios IDEs (Eclipse, IntelliJ IDEA, NetBeans) y Editores de Código (VSCode) que podemos usar para crear un programa Java. Incluso es posible a través de un simple bloc de notas y será él que usaremos para la demostración.

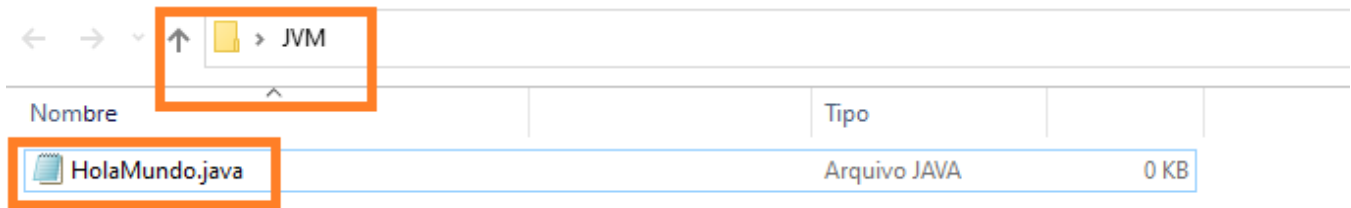
Vamos a crear una carpeta JVM en nuestra computadora:



Abre la carpeta y haz clic derecho dentro de ella, selecciona la opción **Nuevo** y luego **Documento de Texto**:

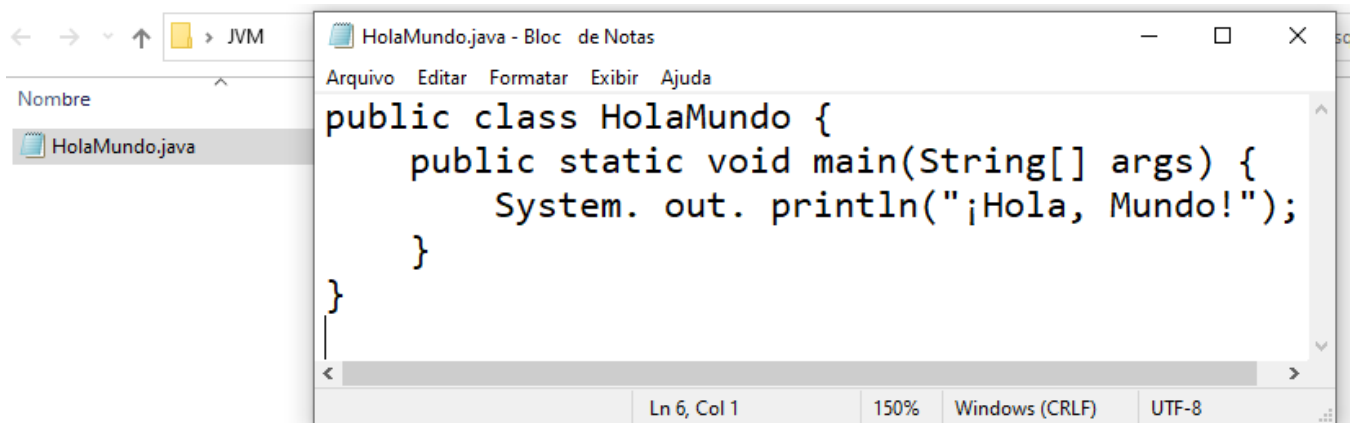


Nombra el archivo como HolaMundo, es importante cambiar también la extensión `.txt` a `.java`. De esta forma, el nombre del archivo será **HolaMundo.java**. Esta es la llamada fase de edición.



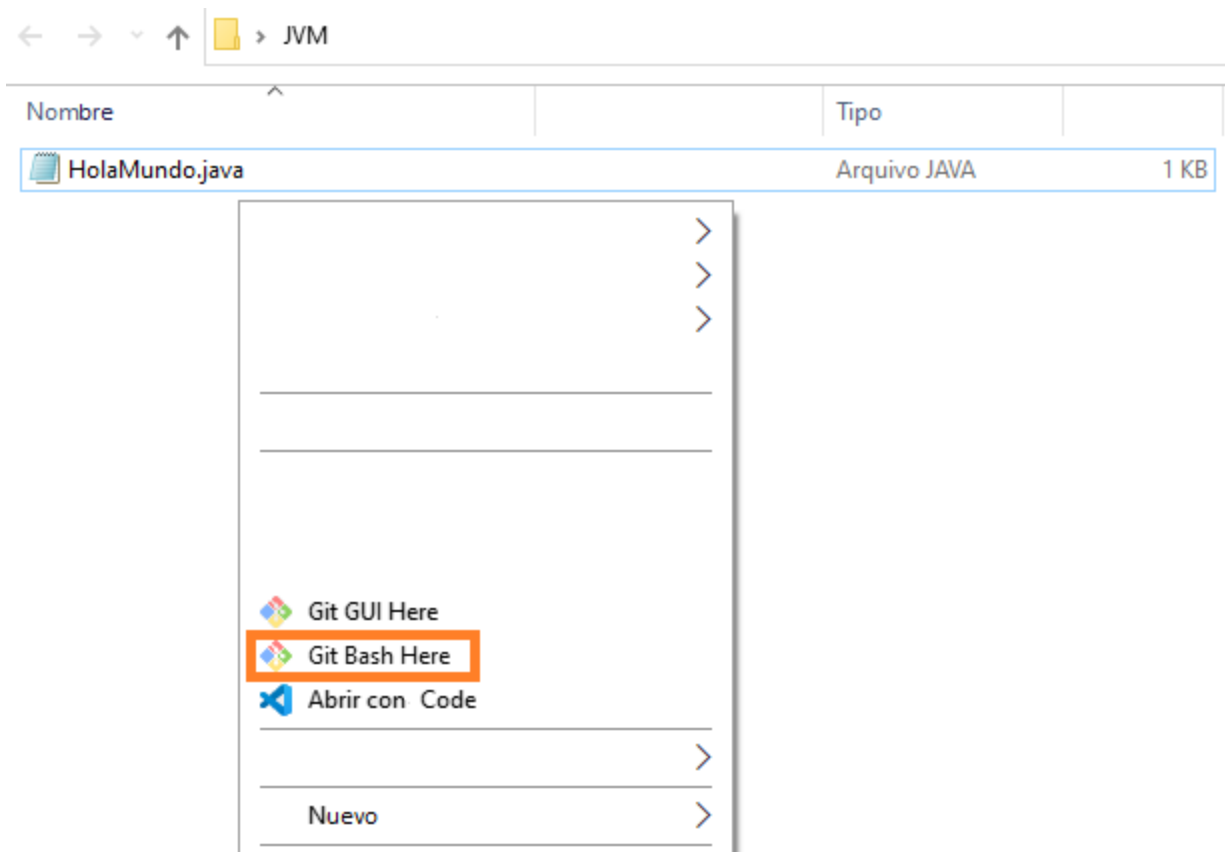
Haz doble clic en el archivo para abrirlo. Ahora, crearemos el clásico "¡Hola mundo!" con la clase y el método principal. Es importante que el nombre del archivo sea el mismo que el de la clase.

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



Guarda el código con **Ctrl + S**, o haciendo clic en **Archivo, Guardar**.

A partir de ahora trabajaremos con el terminal **Git Bash**. Haz clic derecho en un espacio en blanco dentro de la carpeta y selecciona **Git Bash Here**.



Con el terminal abierta, usaremos el comando **ls** para demostrar que tenemos el archivo que acabamos de crear en la carpeta. Después del comando, usa el botón *Enter*.

MINGW64:/c/Users/Ellen/Desktop/JVM

```
E11en@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ ls
HolaMundo.java
E11en@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$
```

Para ver su contenido, escribe **cat HolaMundo.java**.

```
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ ls
HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ cat HolaMundo.java
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ |
```

El proceso de compilación es la transformación del código fuente en código ejecutable, en este caso se transforma en bytecode, un archivo intermedio. Para ello utilizaremos el compilador *javac* (*java compiler*). El comando también debe ir acompañado del nombre del archivo:

javac HolaMundo.java

```
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ ls
HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ cat HolaMundo.java
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javac HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$
```

Si no hay errores, significa que el código ha sido compilado. Al usar el comando **ls** nuevamente, ahora tendremos dos archivos, el código fuente y el bytecode.

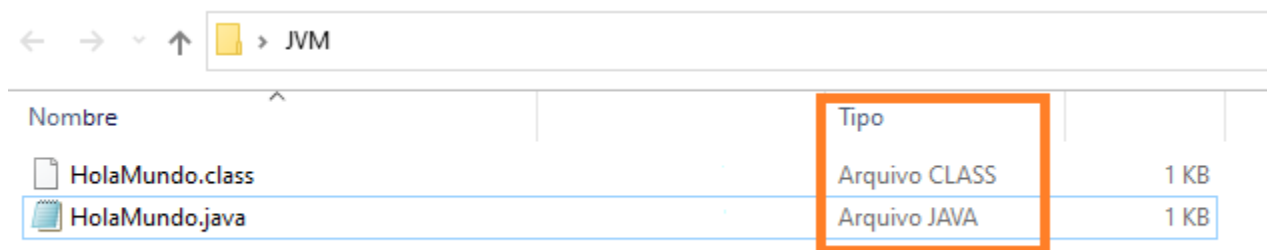
```
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ ls
HolaMundo.java



Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ cat HolaMundo.java
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javac HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ ls
HolaMundo.class  HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$
```



Nombre			Tipo	
	HolaMundo.class		Archivo CLASS	1 KB
	HolaMundo.java		Archivo JAVA	1 KB

Para ejecutar el programa, simplemente escribe java junto con el nombre de la clase:

java HolaMundo

El comando java invocará la JVM, que interpretará los bytecodes generados a través del archivo **HolaMundo.class** para la computadora.

```
$ ls
HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ cat HolaMundo.java
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javac HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ ls
HolaMundo.class  HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ java HolaMundo
¡Hola, Mundo!

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ |
```

Un poco más allá...

Vale, hicimos el proceso de edición, compilación e interpretación. Ahora veamos parte del detrás de escena de este proceso.

Limpiemos nuestro terminal con el comando **clear**.

MINGW64:/c/Users/Ellen/Desktop/JVM

```
$ ls
HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ cat HolaMundo.java
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("¡Hola, Mundo!");
    }
}

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javac HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ ls
HolaMundo.class  HolaMundo.java

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ java HolaMundo
¡Hola, Mundo!

Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ clear
```

Para visualizar el contenido del archivo `.class` es necesario activar el editor `vim` junto con el nombre del archivo con esta extensión:

vim HolaMundo.class

MINGW64:/c/Users/Ellen/Desktop/JVM

```
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ vim HolaMundo.class
```

El contenido presentado está repleto de símbolos, mezclados con partes del código y

una lectura poco comprensible. Este es el bytecode.

```

MINGW64: c:/Users/Ellen/Desktop/JVM
javac HolaMundo.java
HolaMundo.class [unix]
"HolaMundo.class" [noel][converted][unix] 6L, 441B

```

En *vim* hay una forma para ver los archivos *.class* en formato hexadecimal, accedamos a él a través del siguiente comando:

:%!xxd

[illegible]

Tras pulsar *Enter* tenemos acceso a la visualización de los bytecodes en formato

hexadecimal.

En primer lugar tenemos el código **café babe**, el *número mágico de java*, que identifica que se trata de un archivo de tipo java. Al lado derecho en la parte superior, muestra el método especial *init*. Cada doble hexadecimal que tiene un *byte* significa un comando.

```
MINGW64:/c:/Users/Ellen/Desktop/JVM
00000000: cafe babe 0000 003d 001d 0a00 0200 0307 .....=.....
00000010: 0004 0c00 0500 0601 0010 6a61 7661 2f6c .....java/l
00000020: 616e 672f 4f62 6a65 6374 0100 063c 696e ang/Object...<in
00000030: 6974 3e01 0003 2829 5609 0008 0009 0700 it>...()v.....
00000040: 0a0c 000b 000c 0100 106a 6176 612f 6c61 .....java/la
00000050: 6e67 2f53 7973 7465 6d01 0003 6f75 7401 ng/System...out.
00000060: 0015 4c6a 6176 612f 696f 2f50 7269 6e74 ..Ljava/io/Print
00000070: 5374 7265 616d 3b08 000e 0100 10c3 82c2 Stream;.....
00000080: a148 6f6c 612c 204d 756e 646f 210a 0010 .Hola, Mundo!...
00000090: 0011 0700 120c 0013 0014 0100 136a 6176 .....jav
000000a0: 612f 696f 2f50 7269 6e74 5374 7265 616d a/io/PrintStream
000000b0: 0100 0770 7269 6e74 6c6e 0100 1528 4c6a ...println...(Lj
000000c0: 6176 612f 6c61 6e67 2f53 7472 696e 673b ava/lang/String;
000000d0: 2956 0700 1601 0009 486f 6c61 4d75 6e64 )V.....HolaMund
000000e0: 6f01 0004 436f 6465 0100 0f4c 696e 654e o...Code...LineN
000000f0: 756d 6265 7254 6162 6c65 0100 046d 6169 umberTable...mai
00000100: 6e01 0016 285b 4c6a 6176 612f 6c61 6e67 n...([Ljava/lang
00000110: 2f53 7472 696e 673b 2956 0100 0a53 6f75 /String;)V...Sou
00000120: 7263 6546 696c 6501 000e 486f 6c61 4d75 rceFile...HolaMu
00000130: 6e64 6f2e 6a61 7661 0021 0015 0002 0000 ndo.java.!.....
00000140: 0000 0002 0001 0005 0006 0001 0017 0000 .....
00000150: 001d 0001 0001 0000 0005 2ab7 0001 b100 .....*.....
HolaMundo.class[+] [unix]
6 lines filtered
```

Esta lectura tampoco es sencilla, pero existe un formato intermedio que facilita la comprensión del proceso interno de la JVM.

Para salir de esta visualización, usaremos el comando **:q!**

```

00000000: cafe babe 0000 003d 001d 0a00 0200 0307 .....=.....
00000010: 0004 0c00 0500 0601 0010 6a61 7661 2f6c .....java/l
00000020: 616e 672f 4f62 6a65 6374 0100 063c 696e ang/Object...<in
00000030: 6974 3e01 0003 2829 5609 0008 0009 0700 it>...()V.....
00000040: 0a0c 000b 000c 0100 106a 6176 612f 6c61 .....java/la
00000050: 6e67 2f53 7973 7465 6d01 0003 6f75 7401 ng/System...out.
00000060: 0015 4c6a 6176 612f 696f 2f50 7269 6e74 ..Ljava/io/Print
00000070: 5374 7265 616d 3b08 000e 0100 10c3 82c2 Stream;.....
00000080: a148 6f6c 612c 204d 756e 646f 210a 0010 .Hola, Mundo!...
00000090: 0011 0700 120c 0013 0014 0100 136a 6176 .....jav
000000a0: 612f 696f 2f50 7269 6e74 5374 7265 616d a/io/PrintStream
000000b0: 0100 0770 7269 6e74 6c6e 0100 1528 4c6a ...println...(Lj
000000c0: 6176 612f 6c61 6e67 2f53 7472 696e 673b ava/lang/String;
000000d0: 2956 0700 1601 0009 486f 6c61 4d75 6e64 )V.....HolaMund
000000e0: 6f01 0004 436f 6465 0100 0f4c 696e 654e o...Code...LineN
000000f0: 756d 6265 7254 6162 6c65 0100 046d 6169 umberTable...mai
00000100: 6e01 0016 285b 4c6a 6176 612f 6c61 6e67 n...([Ljava/lang
00000110: 2f53 7472 696e 673b 2956 0100 0a53 6f75 /String;)V...Sou
00000120: 7263 6546 696c 6501 000e 486f 6c61 4d75 rceFile...HolaMu
00000130: 6e64 6f2e 6a61 7661 0021 0015 0002 0000 ndo.java.!.....
00000140: 0000 0002 0001 0005 0006 0001 0017 0000 .....
00000150: 001d 0001 0001 0000 0005 2ab7 0001 b100 .....*.....

```

```
HolaMundo.class[+] [unix]
```

```
:q! ↩
```

En el JDK existe una herramienta de utilidad para acceder a este código intermedio, y debe contener el nombre de la clase:

javap -c HolaMundo

```
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ vim HolaMundo.class
```

```
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javap -c HolaMundo
```

Ahora tenemos el código con las instrucciones de la JVM. [Oracle](#) tiene en su documentación la explicación de los comandos y códigos de operación.

```
MINGW64:/c/Users/Ellen/Desktop/JVM
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javap -c HolaMundo
Compiled from "HolaMundo.java"
public class HolaMundo {
    public HolaMundo();
        Code:
            0: aload_0
            1: invokespecial #1                  // Method java/lang/Object."<init>":
()V
            4: return

    public static void main(java.lang.String[]);
        Code:
            0: getstatic     #7                  // Field java/lang/System.out:Ljava/
io/PrintStream;
            3: ldc           #13                 // String ¡Hola, Mundo!
            5: invokevirtual #15                 // Method java/io/PrintStream.printl
n:(Ljava/lang/String;)V
            8: return
}
```

En la primera parte tenemos la indicación del archivo fuente que generó la clase (*compiled from "HolaMundo.java"*), la carga de la referencia de la variable local (*aload_0*), el método especial *init* con el constructor por defecto (*invokespecial*) que regresa vacío.

```
MINGW64:/c/Users/Ellen/Desktop/JVM
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javap -c HolaMundo
Compiled from "HolaMundo.java"
public class HolaMundo {
    public HolaMundo();
        Code:
            0: aload_0
            1: invokespecial #1                  // Method java/lang/Object."<init>":
()V
            4: return

    public static void main(java.lang.String[]);
        Code:
            0: getstatic     #7                  // Field java/lang/System.out:Ljava/
io/PrintStream;
            3: ldc           #13                 // String ¡Hola, Mundo!
            5: invokevirtual #15                 // Method java/io/PrintStream.printl
n:(Ljava/lang/String;)V
            8: return
}
```

En la segunda parte, presenta el método principal con el campo estático de la clase (*getstatic*), la inserción del elemento en el grupo de constantes (*ldc*), una instancia del método y su tipo (*invokevirtual*). El tipo dice que este método toma una matriz *String* primitiva como parámetro y devuelve *Void*.

```
MINGW64:/c:/Users/Ellen/Desktop/JVM
Ellen@LAPTOP-M3O9FEBA MINGW64 ~/Desktop/JVM
$ javap -c HolaMundo
Compiled from "HolaMundo.java"
public class HolaMundo {
    public HolaMundo();
        Code:
            0: aload_0
            1: invokespecial #1          // Method java/lang/Object."<init>":
()v
            4: return

    public static void main(java.lang.String[]);
        Code:
            0: getstatic      #7          // Field java/lang/System.out:Ljava/
io/PrintStream;
            3: ldc            #13         // String ¡Hola, Mundo!
            5: invokevirtual #15         // Method java/io/PrintStream.printl
n:(Ljava/lang/String;)V
            8: return
}
```

Por supuesto, no es necesario que conozcas todos estos comandos, pero es una curiosidad que te ayudará a diario y comprenderás mejor el funcionamiento de la JVM.

Conclusión

Esta es una breve introducción de cómo funciona la JVM, seguramente hay mucho más por explorar. Más que aprender un lenguaje es importante entender cómo este funciona y cómo podemos aplicar estos conocimientos para mejorar el desempeño de nuestros programas.