

# 1.E. Programando con Java.

Sitio: Aula Virtual  
Curso: Programación\_DAM  
Libro: 1.E. Programando con Java.  
Imprimido por: LUIS PUJOL  
Día: martes, 5 de noviembre de 2019, 09:34

# Tabla de contenidos

- 1 Programas en Java.
  - 1.1 Estructura de un programa.
  - 1.2 El entorno básico de desarrollo Java.
  - 1.3 La API de Java.
  - 1.4 Afinando la configuración.
  - 1.5 Codificación, compilación y ejecución de aplicaciones.
  - 1.6 Tipos de aplicaciones en Java.

# 1 Programas en Java.

Hasta ahora, hemos descrito el lenguaje de programación Java, hemos hecho un recorrido por su historia y nos hemos instruido sobre su filosofía de trabajo, pero te preguntarás ¿Cuándo empezamos a desarrollar programas? ¿Qué elementos forman parte de un programa en Java? ¿Qué se necesita para programar en este lenguaje? ¿Podemos crear programas de diferente tipo?



No te impacientes, cada vez estamos más cerca de comenzar la experiencia con el lenguaje de programación Java. Iniciaremos nuestro camino conociendo cuales son los elementos básicos de un programa Java, la forma en que debemos escribir el código y los tipos de aplicaciones que pueden crearse en este lenguaje.

# 1.1 Estructura de un programa.

En el gráfico al que puedes acceder a continuación, se presenta la estructura general de un programa realizado en un lenguaje orientado a objetos como es Java.

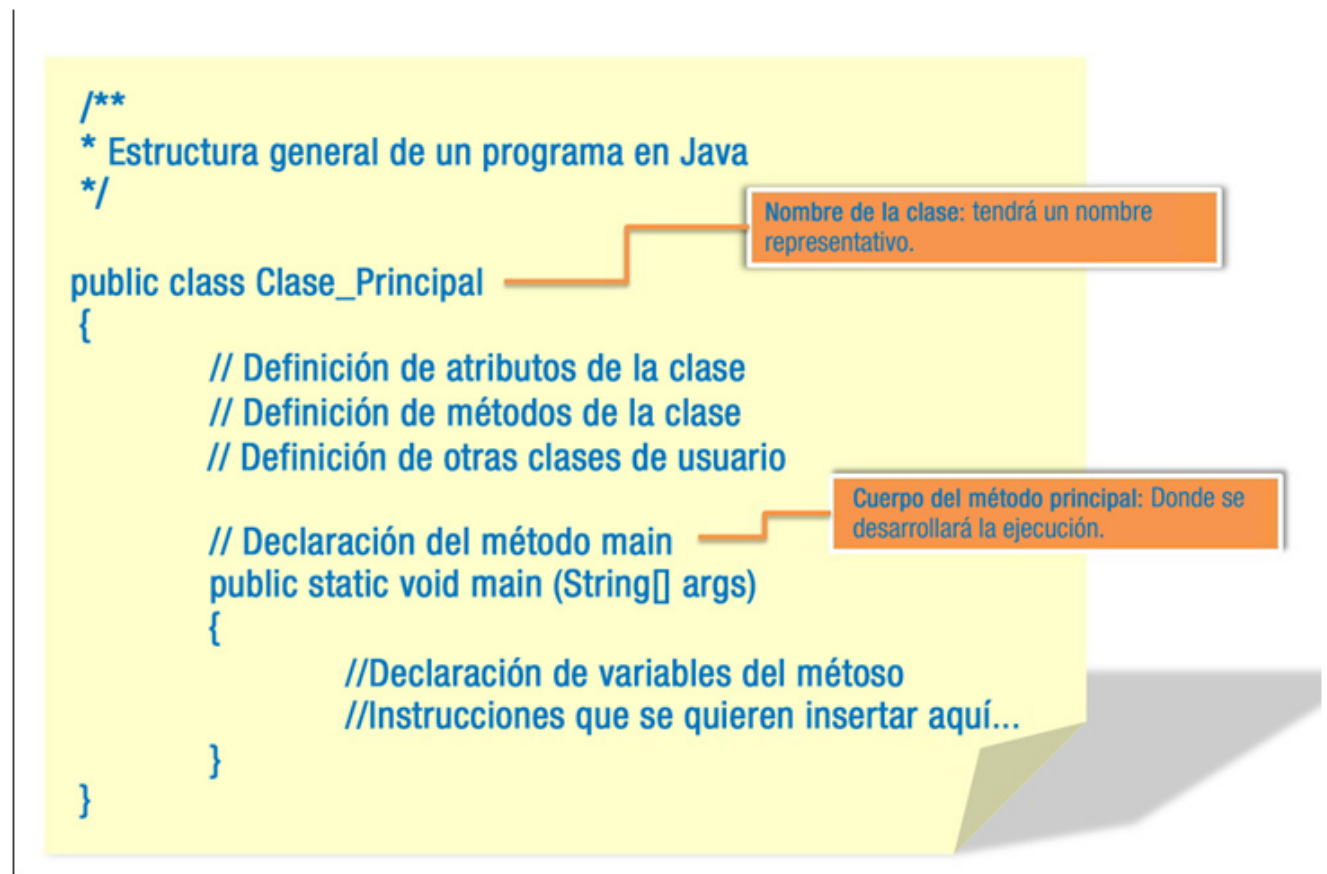


Imagen extraída de curso Programación del MECD.

Vamos a analizar cada uno de los elementos que aparecen en dicho gráfico:

`public class Clase_Principal:` Todos los programas han de incluir una clase como esta. Es una clase general en la que se incluyen todos los demás elementos del programa. Entre otras cosas, contiene el método o función `main()` que representa al programa principal, desde el que se llevará a cabo la ejecución del programa. Esta clase puede contener a su vez otras clases del usuario, pero sólo una puede ser `public`. El nombre del fichero `.Java` que contiene el código fuente de nuestro programa, coincidirá con el nombre de la clase que estamos describiendo en estas líneas.

## Recomendación

Ten en cuenta que **Java distingue entre mayúsculas y minúsculas**. Si le das a la clase principal el nombre `PrimerPrograma`, el archivo `.Java` tendrá como identificador **`PrimerPrograma.Java`**, que es totalmente diferente a `primerprograma.Java`. Además, para Java los elementos `PrimerPrograma` y `primerprograma` serían considerados dos clases diferentes dentro del código fuente.

· `public static void main (String[] args):` Es el método que representa al programa principal, en él se podrán incluir las instrucciones que estimemos oportunas para la ejecución del programa. Desde él se podrá hacer uso del resto de clases creadas. Todos los programas Java tienen un método `main`.

- **Comentarios:** Los comentarios se suelen incluir en el código fuente para realizar aclaraciones, anotaciones o cualquier otra indicación que el programador estime oportuna. Estos comentarios pueden introducirse de dos formas, **con** `//` y **con** `/* */`. Con la primera forma estaríamos estableciendo una única línea completa de comentario y, con la segunda, con `/*` comenzaríamos el comentario y éste no terminaría hasta que no insertáramos `*/`.
- **Bloques de código:** son conjuntos de instrucciones que se marcan mediante la apertura y cierre de llaves `{ }`. El código así marcado es considerado interno al bloque.
- **Punto y coma:** aunque en el ejemplo no hemos incluido ninguna línea de código que termine con punto y coma, hay que hacer hincapié en que cada línea de código ha de terminar con punto y coma (`;`). En caso de no hacerlo, tendremos errores sintácticos.

## Autoevaluación

`public static void main (String[] args)` es la clase general del programa.

- ☐ Verdadero
- ☐ Falso

## 1.2 El entorno básico de desarrollo Java.

Ya conoces cómo es la estructura de un programa en Java, pero, ¿qué necesitamos para llevarlo a la práctica? La herramienta básica para empezar a desarrollar aplicaciones en Java es el **JDK (Java Development Kit o Kit de Desarrollo Java)**, que incluye un compilador y un intérprete para línea de comandos. Estos dos programas son los empleados en la precompilación e interpretación del código.



Imagen extraída de curso Programación del MECD.

Como veremos, existen diferentes entornos para la creación de programas en Java que incluyen multitud de herramientas, pero por ahora nos centraremos en el entorno más básico, extendido y gratuito, el Java Development Kit (JDK). Según se indica en la propia página web de Oracle, JDK es un entorno de desarrollo para construir aplicaciones, applets y componentes utilizando el lenguaje de programación Java. Incluye herramientas útiles para el desarrollo y prueba de programas escritos en Java y ejecutados en la Plataforma Java.

Así mismo, junto a JDK se incluye una implementación del entorno de ejecución Java, el **JRE (Java Runtime Environment)** para ser utilizado por el JDK. El JRE incluye la Máquina Virtual de Java (MVJ ó JVM – Java Virtual Machine), bibliotecas de clases y otros ficheros que soportan la ejecución de programas escritos en el lenguaje de programación Java.

### Debes conocer


Para poder utilizar JDK y JRE es necesario realizar la descarga e instalación de éstos. Para ello podemos ir a la página <https://www.oracle.com/technetwork/java/javase/downloads/index.html>, cuya apariencia es (las imágenes son de una versión previa, la versión disponible en octubre de 2019 es 13.0.1):

The screenshot shows the Oracle Java SE Downloads page. The browser's address bar displays 'a/javase/downloads/index.htm'. The Oracle logo is visible in the top left. The page title is 'Java SE Downloads'. The main content area features a large 'DOWNLOAD' button for the 'Java Platform (JDK) 10'. Below this, the 'Java Platform, Standard Edition' section highlights 'Java SE 10.0.2' as the latest feature release. A list of links for installation instructions, release notes, and license information is provided. On the right, a sidebar lists 'Java SDKs and Tools' and 'Java Resources'. The bottom of the page includes a section titled 'Which Java package do I need?' with three options: Software Developers (JDK), Administrators (Server JRE), and End user (JRE).

Java SE  
Java EE  
Java ME  
Java SE Subscription  
Java Embedded  
Java Card  
Java TV  
Community  
Java Magazine

Overview Downloads Documentation Community Technologies Training

**Java SE Downloads**

  
DOWNLOAD +  
Java Platform (JDK) 10

**Java Platform, Standard Edition**

**Java SE 10.0.2**  
Java SE 10.0.2 is the latest feature release for the Java SE Platform  
[Learn more](#)

- Installation Instructions
- Release Notes
- Oracle License
- Java SE Licensing Information User Manual
  - Includes Third Party Licenses
- Certified System Configurations
- Readme

**JDK**  
DOWNLOAD +

**Server JRE**  
DOWNLOAD +

**JRE**  
DOWNLOAD +

**Which Java package do I need?**

- Software Developers: JDK** (Java SE Development Kit). For Java Developers. Includes a complete JRE plus tools for developing, debugging, and monitoring Java applications.
- Administrators running applications on a server: Server JRE** (Server Java Runtime Environment) For deploying Java applications on servers. Includes tools for JVM monitoring and tools commonly required for server applications, but does not include browser integration (the Java plug-in), auto-update, nor an installer. [Learn more](#)
- End user running Java on a desktop: JRE**: (Java Runtime Environment). Covers most end-users needs. Contains everything required to run Java applications on your system.

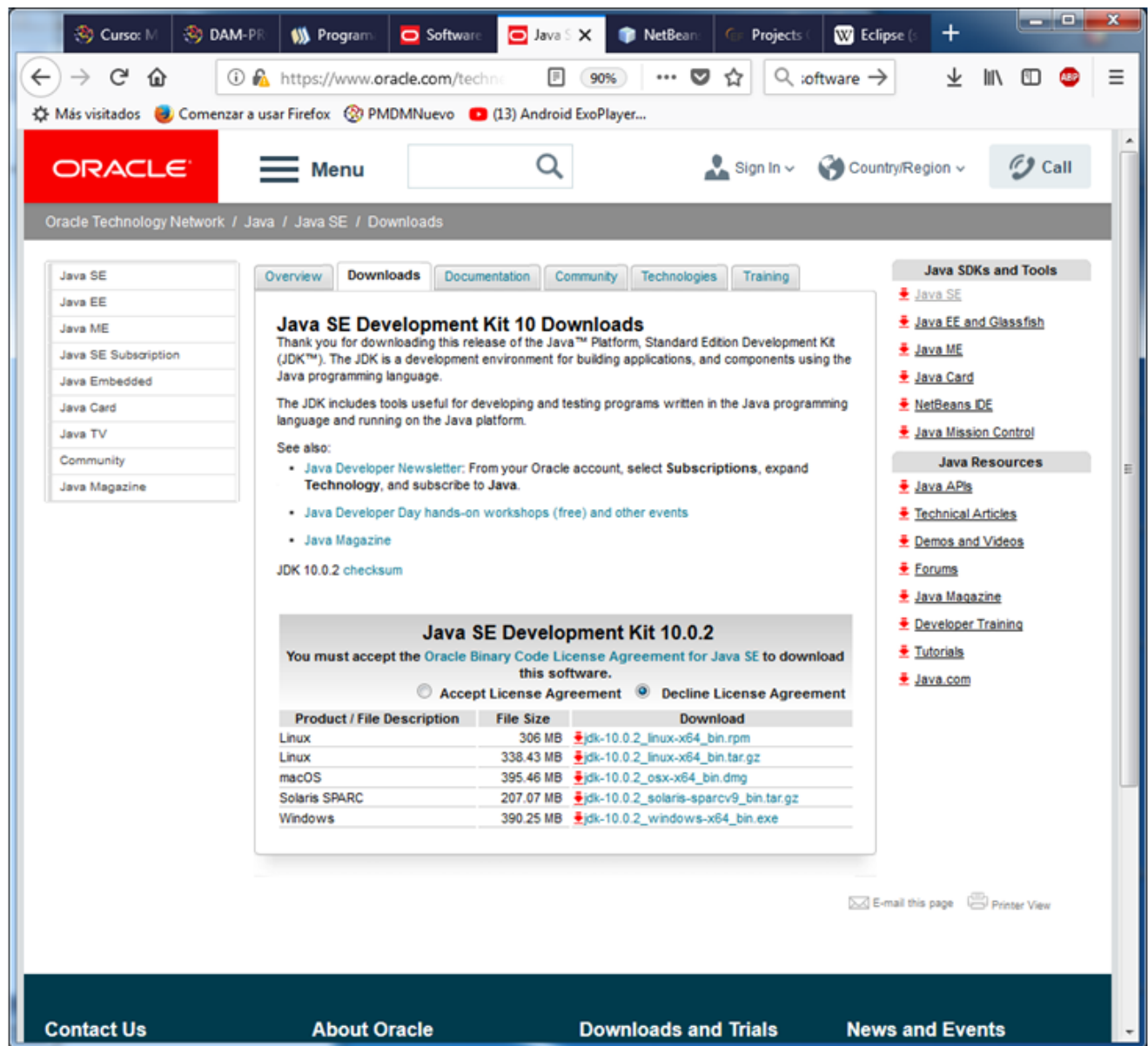
**Java SDKs and Tools**

- [Java SE](#)
- [Java EE and Glassfish](#)
- [Java ME](#)
- [Java Card](#)
- [NetBeans IDE](#)
- [Java Mission Control](#)

**Java Resources**

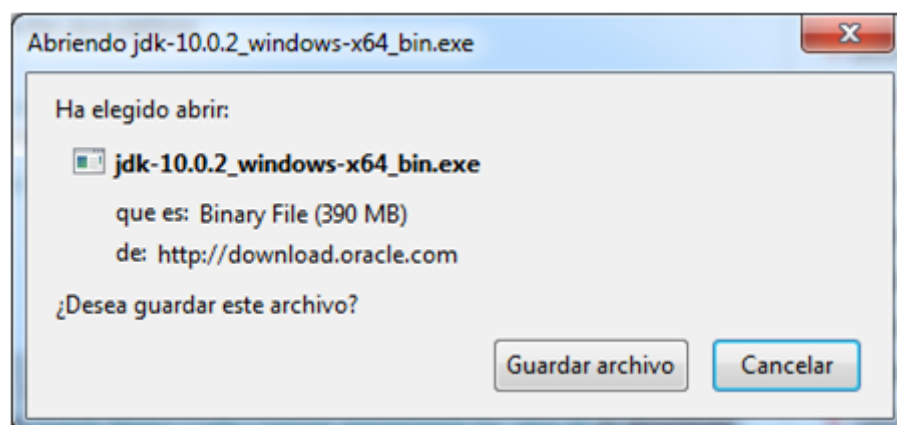
- [Java APIs](#)
- [Technical Articles](#)
- [Demos and Videos](#)
- [Forums](#)
- [Java Magazine](#)
- [Developer Training](#)
- [Tutorials](#)
- [Java.com](#)

Tal y como vemos, no sugiere la instalación de la última versión disponible de JDK (octubre 2019 - versión 13.0.1). Éste incluye el JRE. Si vamos a dicho enlace, nos muestra las versiones para las diferentes plataformas:



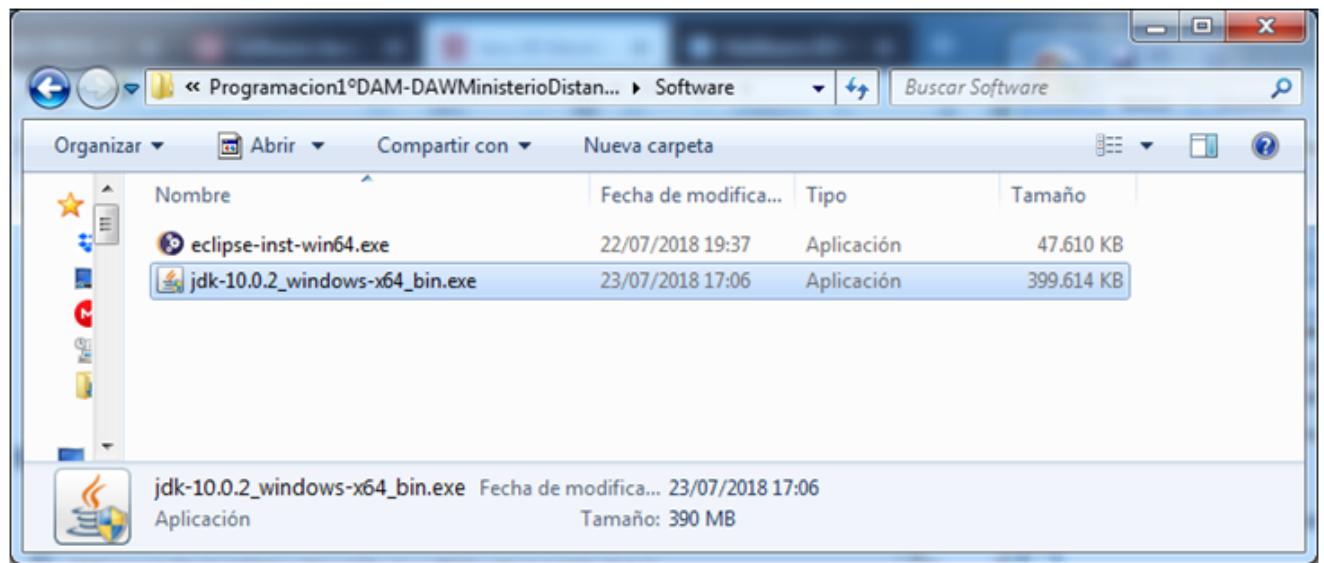
Procederemos a descargar la versión que más nos interese y lanzaremos la instalación guiada.

Concretamente describiremos la instalación en Windows. Para ello aceptamos los términos de la instalación y clicamos sobre el enlace. Comenzará la descarga:



Tras la descarga del fichero tendremos el ejecutable para la instalación de la versión descargada:

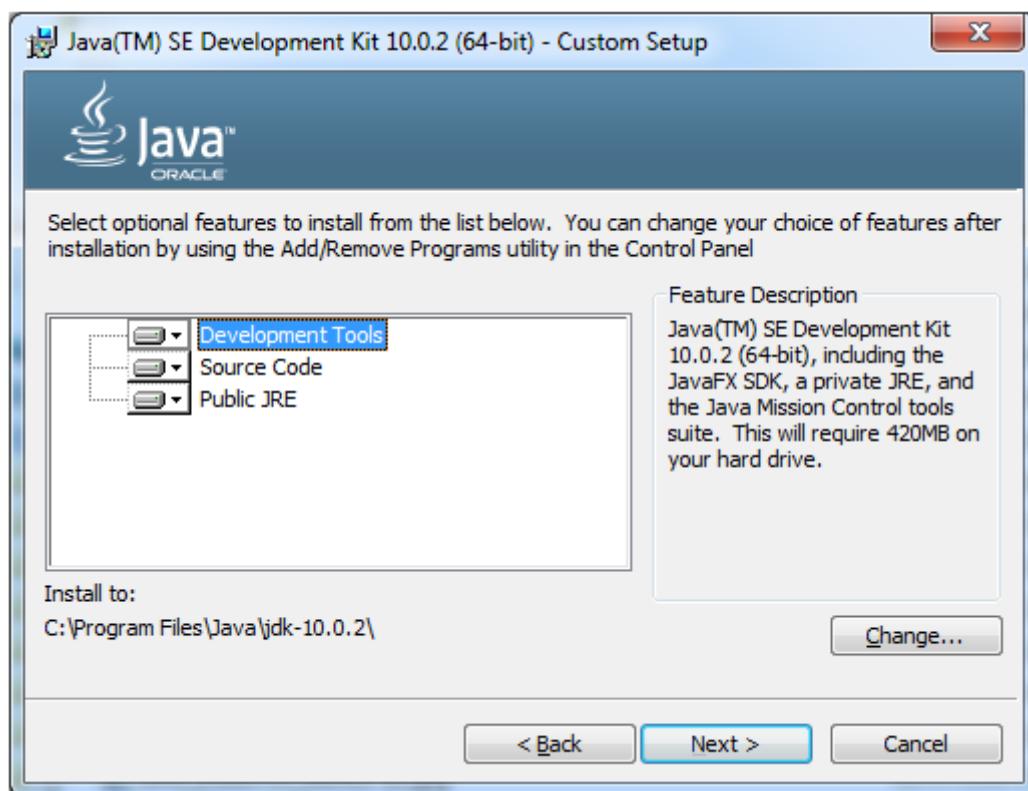




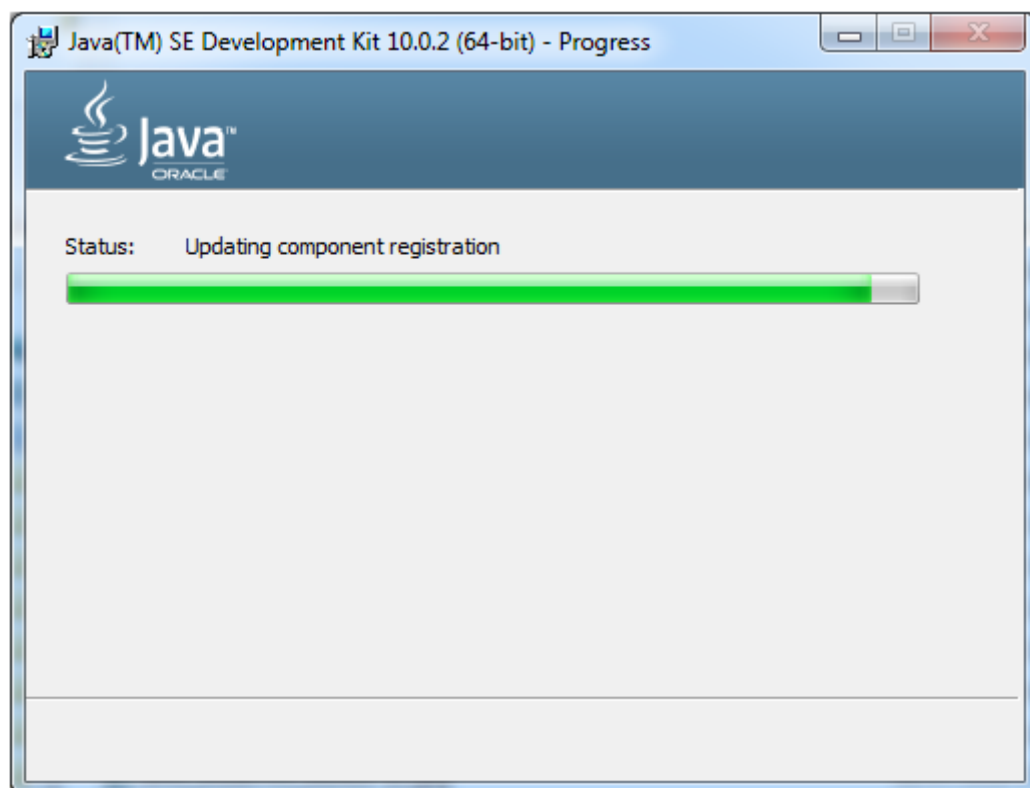
Comenzamos con la instalación:



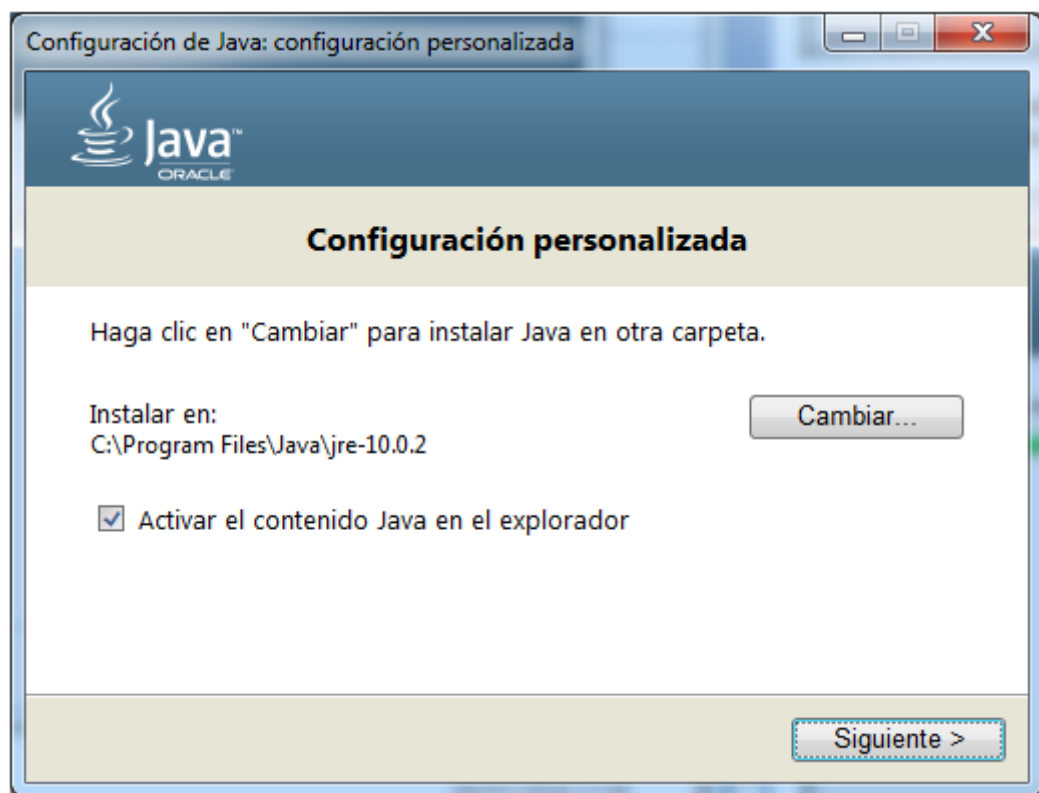
Damos a siguiente:



Damos a siguiente, pudiendo modificar la ruta de la instalación del JDK previamente, si queremos, y ya se inicia el proceso de instalación:



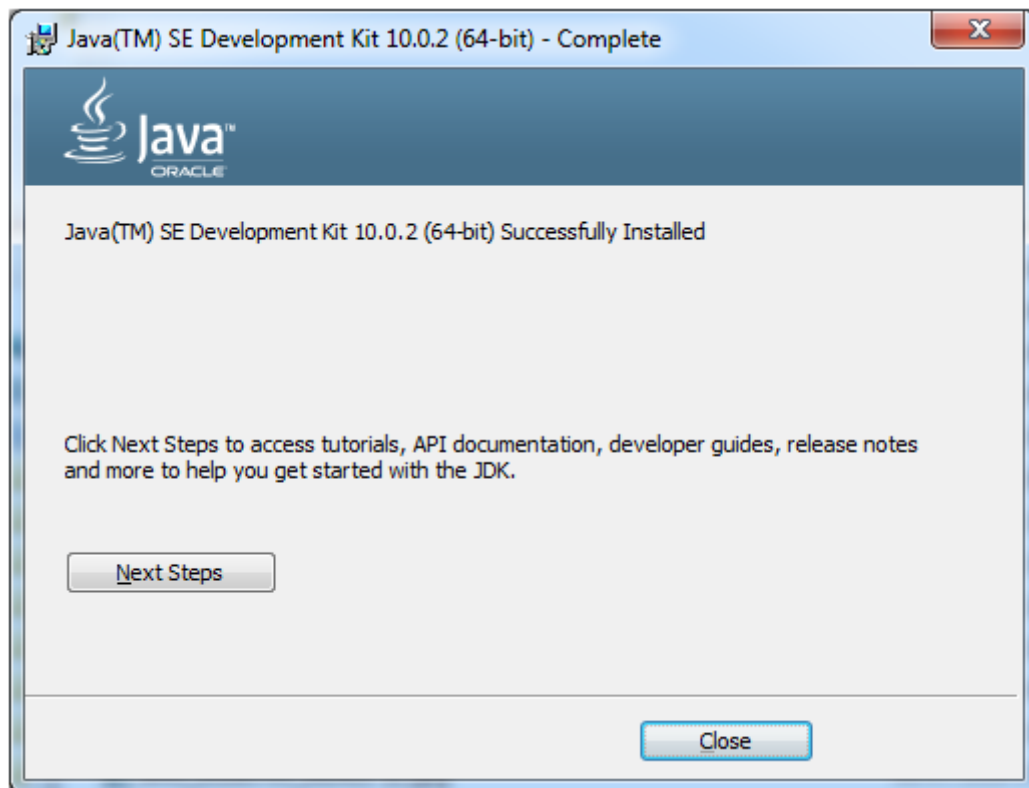
aparece una nueva ventana para personalizar la configuración:



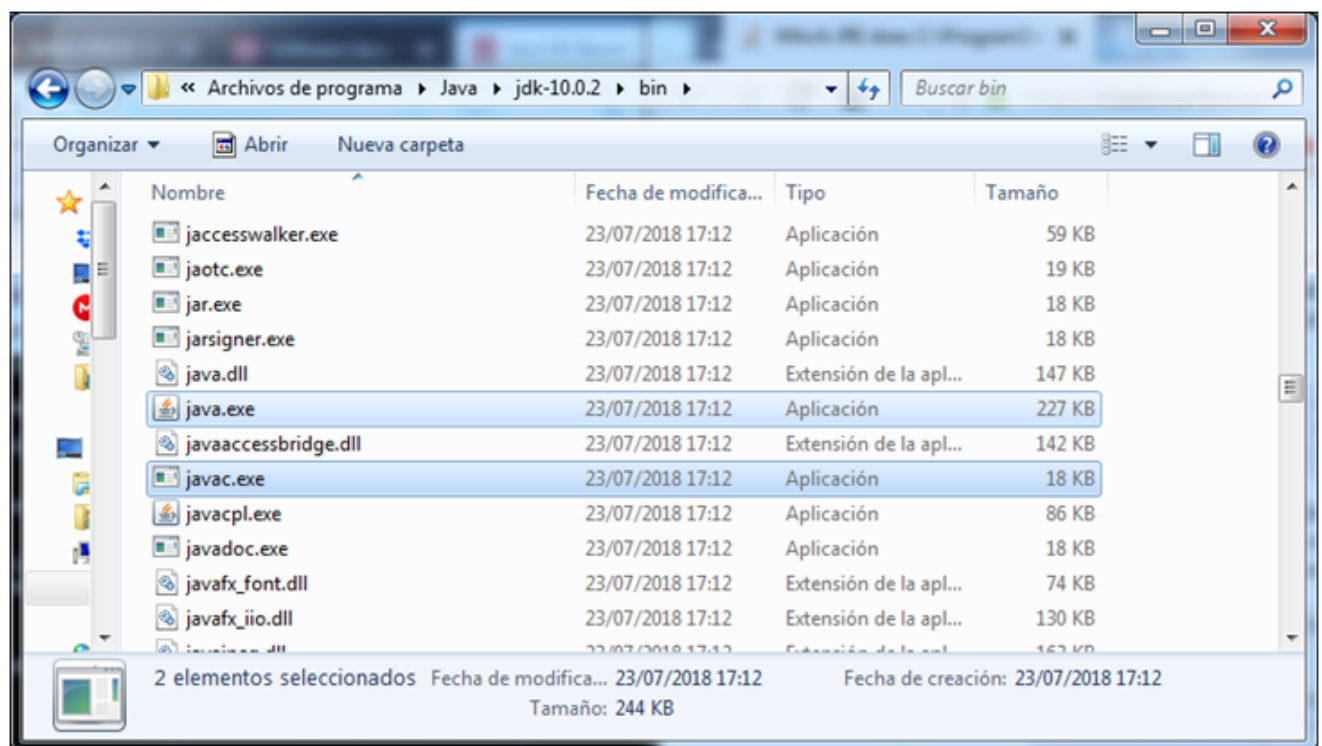
No cambiando nada dando simplemente a Siguiente:



Finalmente aparece la ventana siguiente, en la cual finalizamos la instalación con Close.



Tras la instalación, los ficheros propios de ésta, se encontrarán en la ruta: "C:\Program Files\Java\jdk-13.0.1" (para la versión 13.0.1), y concretamente los ficheros ejecutables con los que interactuaremos se encuentran en la carpeta "bin". Serán, para ser más precisos, **javac.exe** y **java.exe**:



Para poder desarrollar nuestros primeros programas en Java sólo necesitaremos un editor de texto plano y los elementos que acabamos de instalar a través de Java SE. Pero para ello debemos tener la variable de entorno PATH debidamente configurada. Esto se trata en el apartado "Afinando la configuración". Gracias a ello, podremos ejecutar desde cualquier ruta, en la consola, las aplicaciones necesarias para compilar y generar el fichero .class y correr este ejecutable generado.

## Autoevaluación

**Podemos desarrollar programas escritos en Java mediante un editor de textos y a través del JRE podremos ejecutarlos.**

- ☐ Verdadero
- ☐ Falso

## 1.3 La API de Java.

Junto con el kit de desarrollo que hemos descargado e instalado anteriormente, vienen incluidas gratuitamente todas las bibliotecas de la API (Application Programming Interface – Interfaz de programación de aplicaciones) de Java, es lo que se conoce como Bibliotecas de Clases Java. Este conjunto de bibliotecas proporciona al programador paquetes de clases útiles para la realización de múltiples tareas dentro de un programa. Está organizada en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente.



Imagen extraída de curso Programación del MECD.

En décadas pasadas una biblioteca era un conjunto de programas que contenían cientos de rutinas (una rutina es un procedimiento o función bien verificados, en determinado lenguaje de programación). Las rutinas de biblioteca manejaban las tareas que todos o casi todos los programas necesitaban. El programador podía recurrir a esta biblioteca para desarrollar programas con rapidez.

Una biblioteca de clases es un conjunto de clases de programación orientada a objetos. Esas clases contienen métodos que son útiles para los programadores. En el caso de Java cuando descargamos el JDK obtenemos la biblioteca de clases API. Utilizar las clases y métodos de las APIs de Java reduce el tiempo de desarrollo de los programas. También, existen diversas bibliotecas de clases desarrolladas por terceros que contienen componentes reutilizables de software, y están disponibles a través de la Web.

### Para saber más

Si quieres acceder a la información oficial sobre la API de Java, te proponemos el siguiente enlace (está en Inglés).

Información oficial sobre la API de Java

### Autoevaluación

#### Indica qué no es la API de Java:

- ☐ Un entorno integrado de desarrollo.
- ☐ Un conjunto de bibliotecas de clases.
- ☐ Una parte del JDK, incluido en el Java SE.

## 1.4 Afinando la configuración.

Para que podamos compilar y ejecutar ficheros Java es necesario que realicemos unos pequeños ajustes en la configuración del sistema. Vamos a indicarle dónde encontrar los ficheros necesarios para realizar las labores de compilación y ejecución, en este caso `Javac.exe` y `Java.exe`, así como las librerías contenidas en la API de Java y las clases del usuario.

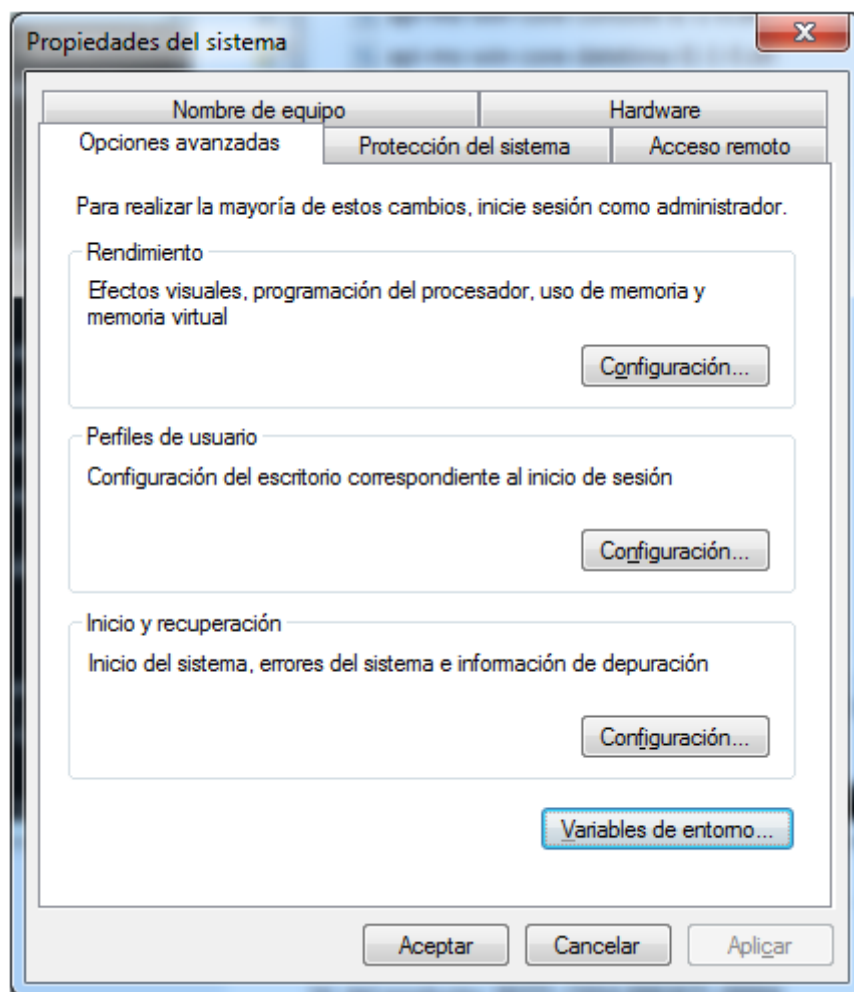


Imagen extraída de curso Programación del MECD.

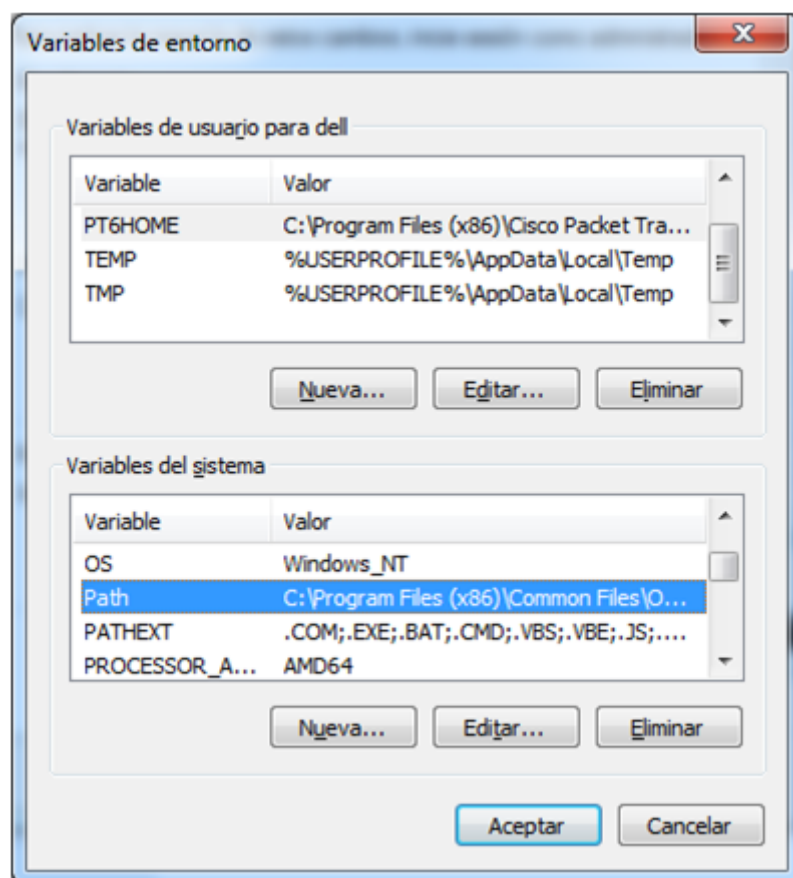
**La variable PATH:** Como aún no disponemos de un IDE (Integrated Development Environment - Entorno Integrado de Desarrollo) la única forma de ejecutar programas es a través de línea de comandos. Pero sólo podremos ejecutar programas directamente si la ruta hacia ellos está indicada en la variable PATH del ordenador. Es necesario que incluyamos la ruta hacia estos programas en nuestra variable PATH. Esta ruta será el lugar donde se instaló el JDK hasta su directorio `bin`.

En Windows:

Vamos a las propiedades del sistema:

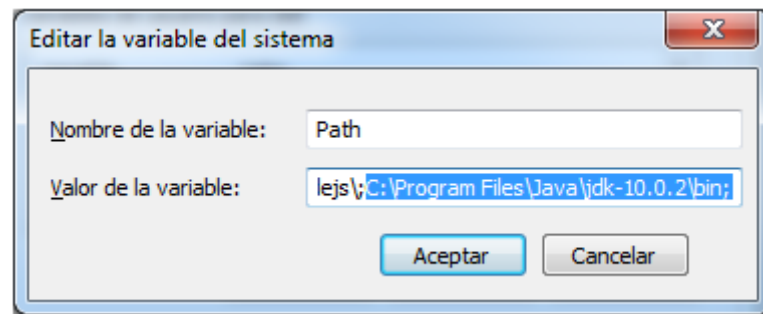


Vamos a las Variables de entorno...:



En la variable Path, damos a Editar... e incorporamos la ruta a la carpeta bin del JDK, la cual contiene los ficheros "javac.exe" y "java.exe":





Damos a Aceptar, y a continuación abrimos el interprete de comandos y en él, debemos probar tanto el comando "javac.exe" como el comando "java.exe":

javac.exe funciona:

```
C:\Windows\system32\cmd.exe
C:\Users\dell>javac
Usage: javac <options> <source files>
where possible options include:
  @<filename>                Read options and filenames from file
  -Akey[=value]              Options to pass to annotation processors
  --add-modules <module>[,<module>]*
                             Root modules to resolve in addition to the initial modules, or all modules
                             on the module path if <module> is ALL-MODULE-PATH.
  --boot-class-path <path>, -bootclasspath <path>
                             Override location of bootstrap class files
  --class-path <path>, -classpath <path>, -cp <path>
                             Specify where to find user class files and annotation processors
  -d <directory>             Specify where to place generated class files
  -deprecation
                             Output source locations where deprecated APIs are used
  -encoding <encoding>       Specify character encoding used by source files
  -endorseddirs <dirs>       Override location of endorsed standards path
  -extdirs <dirs>            Override location of installed extensions
  -g                          Generate all debugging info
  -g:{lines,vars,source}     Generate only some debugging info
  -g:none                    Generate no debugging info
  -h <directory>             Specify where to place generated native header files
  --help, -help              Print this help message
  --help-extra, -X           Print help on extra options
  -implicit:{none,class}     Specify whether or not to generate class files for implicitly referenced
files
  -J<flag>                   Pass <flag> directly to the runtime system
  --limit-modules <module>[,<module>]*
                             Limit the universe of observable modules
  --module <module-name>, -m <module-name>
                             Compile only the specified module, check timestamps
  --module-path <path>, -p <path>
                             Specify where to find application modules
  --module-source-path <module-source-path>
                             Specify where to find input source files for multiple modules
  --module-version <version>
                             Specify version of modules that are being compiled
  -nowarn                    Generate no warnings
  -parameters
                             Generate metadata for reflection on method parameters
  -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
  --processor-module-path <path>
                             Specify a module path where to find annotation processors
  --processor-path <path>, -processorpath <path>
                             Specify where to find annotation processors
  -profile <profile>          Check that API used is available in the specified profile
  --release <release>         Compile for a specific VM version. Supported targets: 10, 6, 7, 8, 9
  -s <directory>             Specify where to place generated source files
  -source <release>           Provide source compatibility with specified release
  --source-path <path>, -sourcepath <path>
                             Specify where to find input source files
  --system <jdk>!none         Override location of system modules
  -target <release>           Generate class files for specific VM version
  --upgrade-module-path <path>
                             Override location of upgradeable modules
  -verbose                   Output messages about what the compiler is doing
  --version, -version         Version information
  -Werror                     Terminate compilation if warnings occur
```

y java.exe también funciona:

```
C:\Windows\system32\cmd.exe
C:\Users\dell>java
Sintaxis: java [opciones] <clase principal> [argumentos...]
    (para ejecutar una clase)
o java [opciones] -jar <archivo jar> [argumentos...]
    (para ejecutar un archivo jar)
o java [opciones] -m <módulo>[/<clase principal>] [argumentos...]
    java [opciones] --module <módulo>[/<clase principal>] [argumentos...]
    (para ejecutar la clase principal en un módulo)

Argumentos que siguen la clase principal, -jar <archivo jar>, -m o --module
<módulo>/<clase principal> se transfieren como argumentos a una clase principal
.

donde las opciones incluyen:

-cp <ruta de búsqueda de clase de directorios y archivos zip/jar>
-classpath <ruta de búsqueda de clase de directorios y archivos zip/jar>
--class-path <ruta de búsqueda de clase de directorios y archivos zip/jar>
    Una lista separada por el carácter ;, archivos JAR
    y archivos ZIP para buscar archivos de clases.
-p <ruta módulo>
--module-path <ruta módulo>...
    Una lista de directorios separada por el carácter ;, cada dire
    ctorio
    es un directorio de módulos.
--upgrade-module-path <ruta módulo>...
    Una lista de directorios separada por el carácter ;, cada dire
    ctorio
    es un directorio de módulos que sustituye a
    los módulos actualizables en la imagen de tiempo de ejecución
--add-modules <nombre módulo>[,<nombre módulo>...]
    módulos de raíz que resolver, además del módulo inicial.
    <nombre módulo> también puede ser ALL-DEFAULT, ALL-SYSTEM,
    ALL-MODULE-PATH.
--list-modules
    mostrar módulos observables y salir
-d <nombre de módulo>
--describe-module <nombre módulo>
    describir un módulo y salir
--dry-run
    crear VM y cargar la clase principal pero sin ejecutar el méto
    do principal.
    La opción --dry-run puede ser útil para validar
    las opciones de línea de comandos, como la configuración del s
    istema de módulos.
--validate-modules
    validar todos los módulos y salir
    La opción --validate-modules puede ser útil para encontrar
    conflictos y otros errores con módulos en la ruta de módulos.
-D<nombre>=<valor>
    definir una propiedad de sistema
-verbose:[class|module|gc|jni]
    activar la salida en modo verbose
-version
    imprimir versión de producto en el flujo de errores y salir
--version
    imprimir versión de producto en el flujo de salida y salir
-showversion
    imprimir versión de producto en el flujo de errores y continua
    r
--show-version
    -showversion imprimir versión de producto en el flujo de sali
    da y continuar
--show-module-resolution
    mostrar la salida de resolución de módulo durante el inicio
-? -h -help
    imprimir este mensaje de ayuda en el flujo de errores
--help
    imprimir este mensaje de ayuda en el flujo de salida
-X
    imprimir ayuda de opciones adicionales en el flujo de errores
--help-extra
    imprimir ayuda de opciones adicionales en el flujo de salida
-ea[:<nombre paquete>...]:<nombre clase>]
-enableassertions[:<nombre paquete>...]:<nombre clase>]
    activar afirmaciones con una granularidad especificada
```

Aquí tienes unos enlaces donde se describe esto con más detalle:

- Variables de entorno para Java en Windows

## CONFIGURAR VARIABLES DE ENTORNO JAVA Windows 7 8 8.1 10



- Variables de entorno para Java en Ubuntu (versión antigua)

<http://chicomonte.blogspot.com/2010/04/instalar-java-jdk-jre-en-ubuntu.html>

**La variable** `CLASSPATH`: esta variable de entorno establece dónde buscar las clases o bibliotecas de la API de Java, así como las clases creadas por el usuario. Es decir, los ficheros `.class` que se obtienen una vez compilado el código fuente de un programa escrito en Java. Es posible que en dicha ruta existan directorios y ficheros comprimidos en los formatos `zip` o `jar` que pueden ser utilizados directamente por el JDK, conteniendo en su interior archivos con extensión `class`.

(Por ejemplo: `C:\Program Files\Java\jdk1.6.0_25\bin`)

Si no existe la variable **CLASSPATH** debes crearla, para modificar su contenido sigue el mismo método que hemos empleado para la modificación del valor de la variable `PATH`, anteriormente descrito. Ten en cuenta que la ruta que debes incluir será el lugar donde se instaló el JDK hasta su directorio `lib`.

(Por ejemplo: `C:\Program Files\Java\jdk1.6.0_25\lib`)

### Autoevaluación

¿Qué variable de sistema o de entorno debemos configurar correctamente para que podamos compilar directamente desde la línea de comandos nuestros programas escritos en lenguaje Java?

- ☐ `CLASSPATH`.
- ☐ `PATH`.
- ☐ `Javac.exe`.

## 1.5 Codificación, compilación y ejecución de aplicaciones.

Una vez que la configuración del entorno Java está completada y tenemos el código fuente de nuestro programa escrito en un archivo con extensión `.java`, la compilación de aplicaciones se realiza mediante el programa **javac** incluido en el software de desarrollo de Java.

Para llevar a cabo la compilación desde la línea de comandos, escribiremos:

```
javac archivo.java
```

Donde `javac` es el compilador de Java y `archivo.java` es nuestro código fuente.



Imagen extraída de curso Programación del MECD.

El resultado de la compilación será un archivo con el mismo nombre que el archivo Java pero con la **extensión** `class`. Esto ya es el archivo con el código en forma de bytecode. Es decir con el código precompilado. Si en el código fuente de nuestro programa figuraran más de una clase, veremos como al realizar la compilación se generarán tantos archivos con extensión `.class` como clases tengamos. Además, si estas clases tenían método `main` podremos ejecutar dichos archivos por separado para ver el funcionamiento de dichas clases.

Para que el programa pueda ser ejecutado, siempre y cuando esté incluido en su interior el método `main`, podremos utilizar el interprete incluido en el kit de desarrollo.

La ejecución de nuestro programa desde la línea de comandos podremos hacerla escribiendo:

```
java archivo
```

Donde `Java` es el intérprete y `archivo` es el archivo con el código precompilado con extensión `".class"`.

### Ejercicio resuelto

Vamos a llevar a la práctica todo lo que hemos estado detallando a través de la creación, compilación y ejecución de un programa sencillo escrito en Java.

Observa el código que se muestra más abajo, seguro que podrás entender parte de él. Cópialo en un editor de texto, respetando las mayúsculas y las minúsculas. Puedes guardar el archivo con extensión `.java` en la ubicación que prefieras. Recuerda que el nombre de la clase principal (en el código de ejemplo `MiModulo`) debe ser exactamente igual al del archivo con extensión `.java`, si tienes esto en cuenta la aplicación podrá ser compilada correctamente y ejecutada.

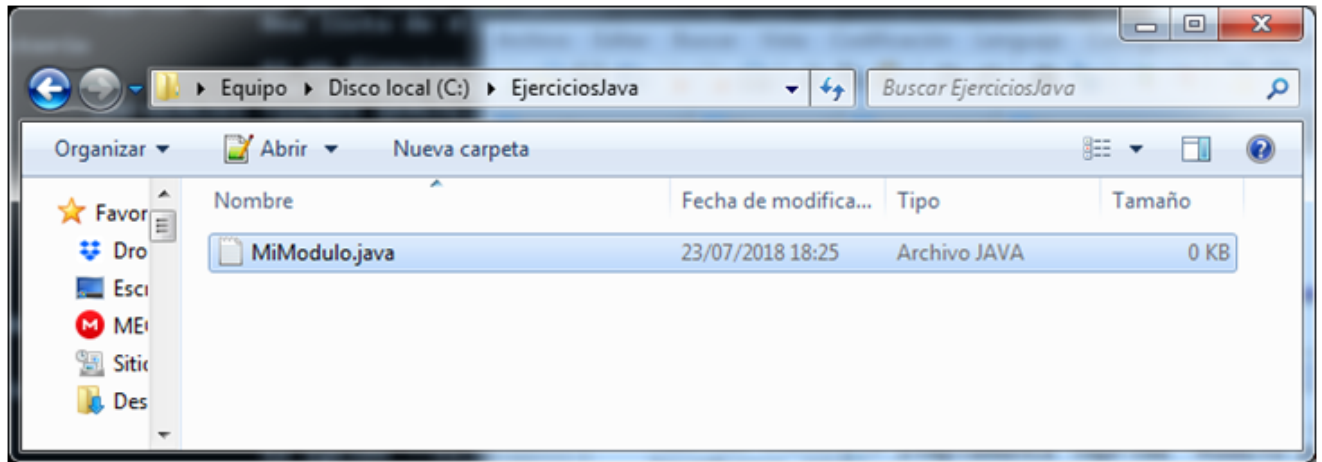
```
/**
 * La clase MiModulo implementa una aplicación que
 * simplemente imprime "Módulo profesional - Programación" en pantalla.
 */
class MiModulo {
```

```

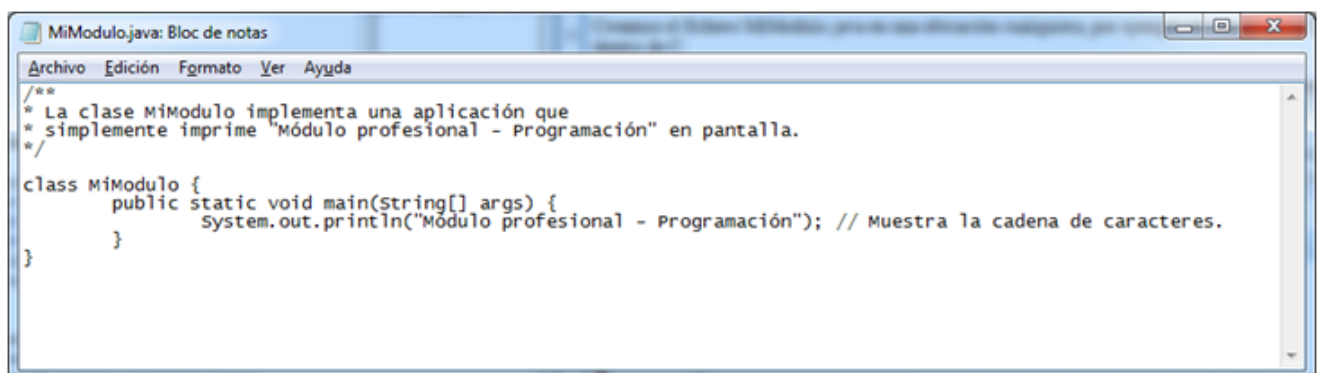
public static void main(String[] args) {
System.out.println("Módulo profesional - Programación"); // Muestra la cadena de caracteres.
}
}

```

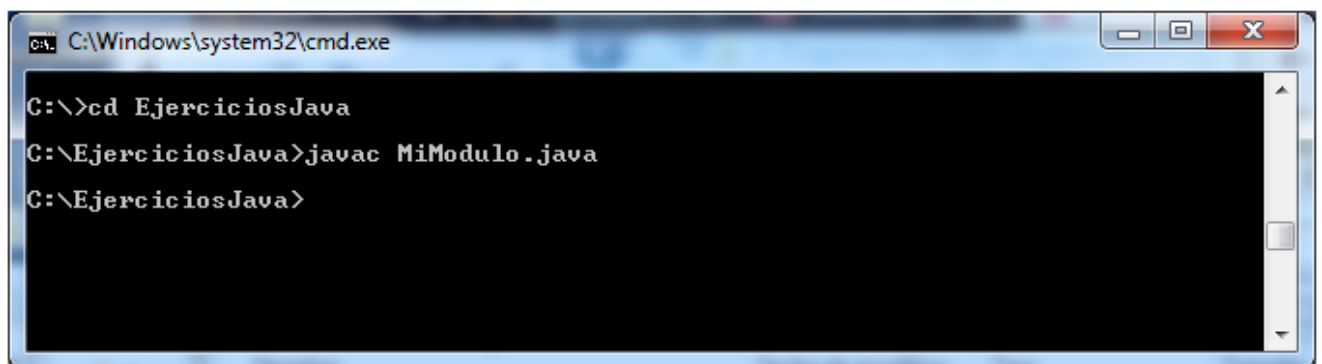
Creamos el fichero MiModulo.java en una ubicación cualquiera, por ejemplo en una carpeta "EjerciciosJava" dentro de C:



Podemos usar un editor de texto plano como el bloc de notas de Windows:

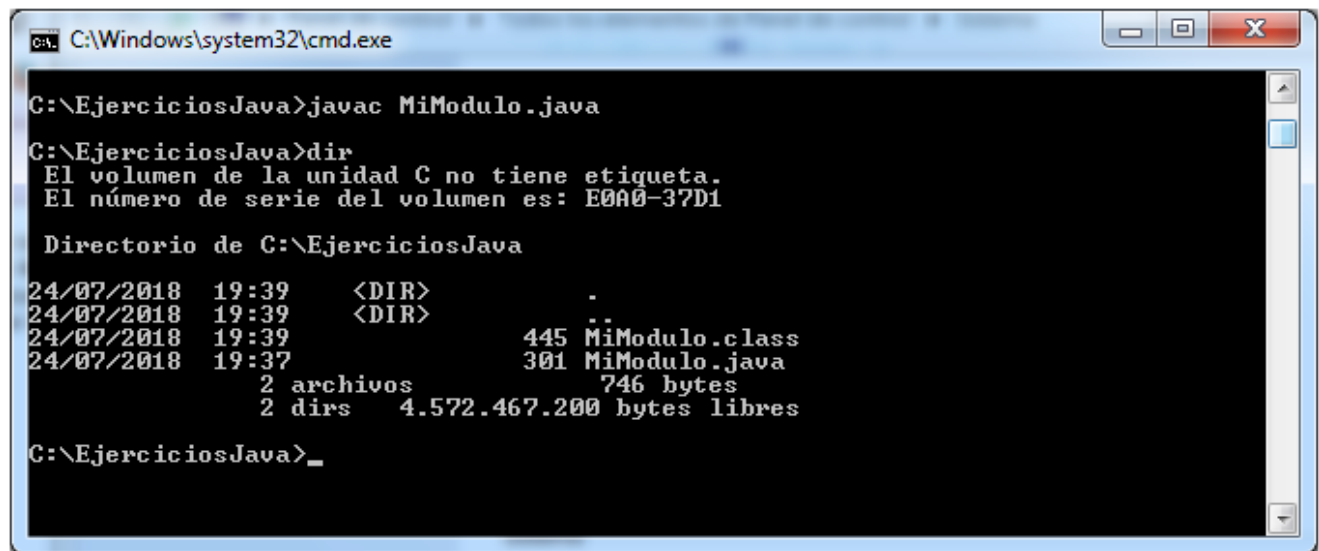


Accede a la línea de comandos y teclea, en la carpeta donde has guardado el archivo Java, el comando **para compilarlo**: `Javac MiModulo.java`



Aparentemente no hace nada, ya que no devuelve ningún mensaje. Precisamente esto es síntoma de que todo ha ido bien.

El compilador habrá generado entonces un fichero de código de bytes: `MiModulo.class`. Si visualizas ahora el contenido de la carpeta verás que en ella está el archivo `.java` y uno o varios (depende de las clases que contenga el archivo con el código fuente) archivos `.class`.



```
C:\Windows\system32\cmd.exe

C:\EjerciciosJava>javac MiModulo.java

C:\EjerciciosJava>dir
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: E0A0-37D1

Directorio de C:\EjerciciosJava
24/07/2018  19:39    <DIR>          .
24/07/2018  19:39    <DIR>          ..
24/07/2018  19:39                445 MiModulo.class
24/07/2018  19:37                301 MiModulo.java
                2 archivos              746 bytes
                2 dirs    4.572.467.200 bytes libres

C:\EjerciciosJava>_
```

Finalmente, **para realizar la ejecución** del programa debes utilizar la siguiente sentencia:

```
java MiModulo
```

Si todo ha ido bien, verás escrito en pantalla: "Módulo profesional - Programación".

IMPORTANTE: para que encuentre la clase el ejecutable "java", es imprescindible configurar bien la variable CLASSPATH de modo que esté la ruta de la clase y librerías necesarias para la ejecución. Si no está incluida es necesario usar el parámetro -cp para indicar la ruta de la clase. Normalmente si se ejecuta en el directorio en el que está la clase no suele ser necesario, pero hay versiones de java que necesitan tenerlo indicado en la variable o con el parámetro. Por ejemplo, si no hemos incluido en la variable CLASSPATH la ruta de nuestra clase, aunque estemos en el mismo directorio, con algunas versiones es preciso indicarlo con "java -cp . MiModulo".

## 1.6 Tipos de aplicaciones en Java.

La versatilidad del lenguaje de programación Java permite al programador crear distintos tipos de aplicaciones. A continuación, describiremos las características más relevantes de cada uno de ellos:

- **Aplicaciones de consola:**

- Son programas independientes al igual que los creados con los lenguajes tradicionales.
- Se componen como mínimo de un archivo .class que debe contar necesariamente con el método main.
- No necesitan un navegador web y se ejecutan cuando invocamos el comando Java para iniciar la Máquina Virtual de Java (JVM). De no encontrarse el método main la aplicación no podrá ejecutarse.
- Las aplicaciones de consola leen y escriben hacia y desde la entrada y salida estándar, sin ninguna interfaz gráfica de usuario.

- **Aplicaciones gráficas:**

- Aquellas que utilizan las clases con capacidades gráficas, como Swing que es la biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE.
- Incluyen las instrucciones import, que indican al compilador de Java que las clases del paquete javax.swing se incluyan en la compilación.

- **Applets:**

- Son programas incrustados en otras aplicaciones, normalmente una página web que se muestra en un navegador. Cuando el navegador carga una web que contiene un applet, éste se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.
- Se pueden descargar de Internet y se observan en un navegador. Los applets se descargan junto con una página HTML desde un servidor web y se ejecutan en la máquina cliente.
- No tienen acceso a partes sensibles (por ejemplo: no pueden escribir archivos), a menos que uno mismo le dé los permisos necesarios en el sistema.
- No tienen un método principal.
- Son multiplataforma y pueden ejecutarse en cualquier navegador que soporte Java.

- **Servlets:**

- Son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.
- Los servlets, al contrario de los applets, son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones Web que interactúen con los clientes.

- **Midlets:**

- Son aplicaciones creadas en Java para su ejecución en sistemas de propósito simple o dispositivos móviles. Los juegos Java creados para teléfonos móviles son midlets.
- Son programas creados para dispositivos embebidos (se dedican a una sola actividad), más específicamente para la máquina virtual Java MicroEdition (Java ME).
- Generalmente son juegos y aplicaciones que se ejecutan en teléfonos móviles.





### **Autoevaluación**

**Un Applet es totalmente seguro ya que no puede acceder, en ningún caso, a zonas sensibles del sistema. Es decir, no podría borrar o modificar nuestros archivos.**

- ☐ Verdadero
- ☐ Falso