



Ministerio de Producción  
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



# Introducción a la Plataforma Java

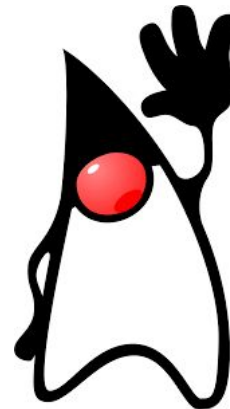
# ¿A qué se denomina Plataforma Java?

A un conjunto de herramientas de software (programas y bibliotecas) que permiten desarrollar aplicaciones multiplataforma, desarrollado originalmente por Sun, hoy en día propiedad de Oracle.

Una de las formas más usadas para crear aplicaciones para la plataforma Java es utilizando el **lenguaje de programación Java** (aunque no es la única forma).

# Historia

- Java se inicia en 1990 con la creación del lenguaje y su compilador. Java debe su nombre al café java.
- El lenguaje fue creado con una sintaxis similar a C/C++, con el objetivo de captar desarrolladores de dichos lenguajes (en esa época eran muchos).
- Fue diseñado originalmente para ser usado en televisión y más tarde en navegadores (Java Applets), pero no tuvo éxito.



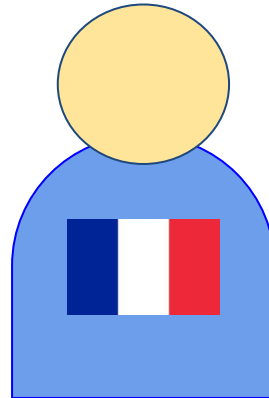
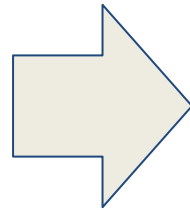
# Java: Presente

- En la actualidad, Java se utiliza ampliamente en desarrollo de aplicaciones del lado del servidor (servicios cloud, servidores web, e-commerce, etc.) y desarrollo móvil (Android), entre otros (big data, aplicaciones científicas, etc.).

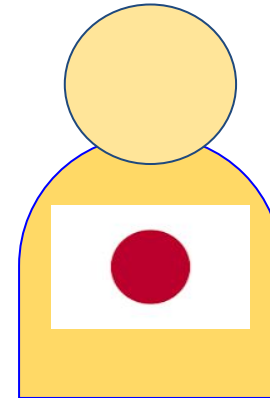


Texto 1  
en un  
idioma X

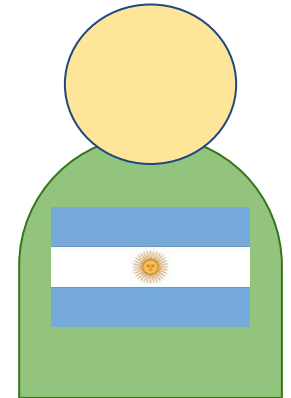
Traducción a cada  
lenguaje  
existente



Texto 1 en  
Francés



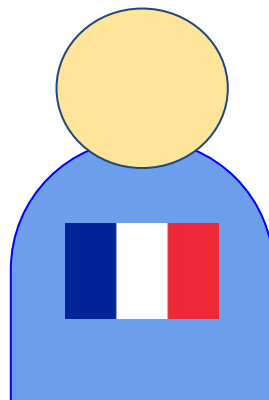
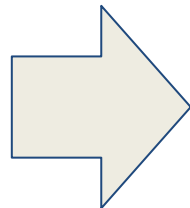
Texto 1 en  
Japonés



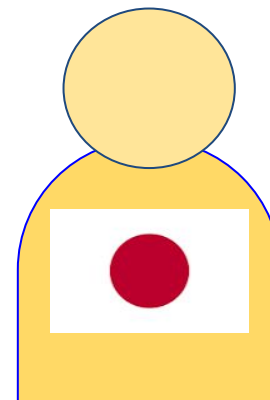
Texto 1 en  
Español

Texto 2  
en un  
idioma Y

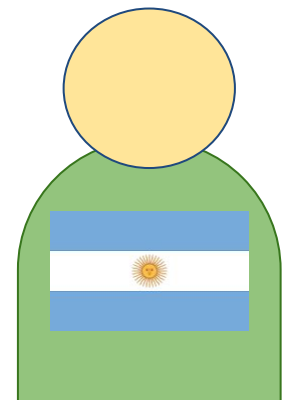
Traducción a cada  
lenguaje  
existente



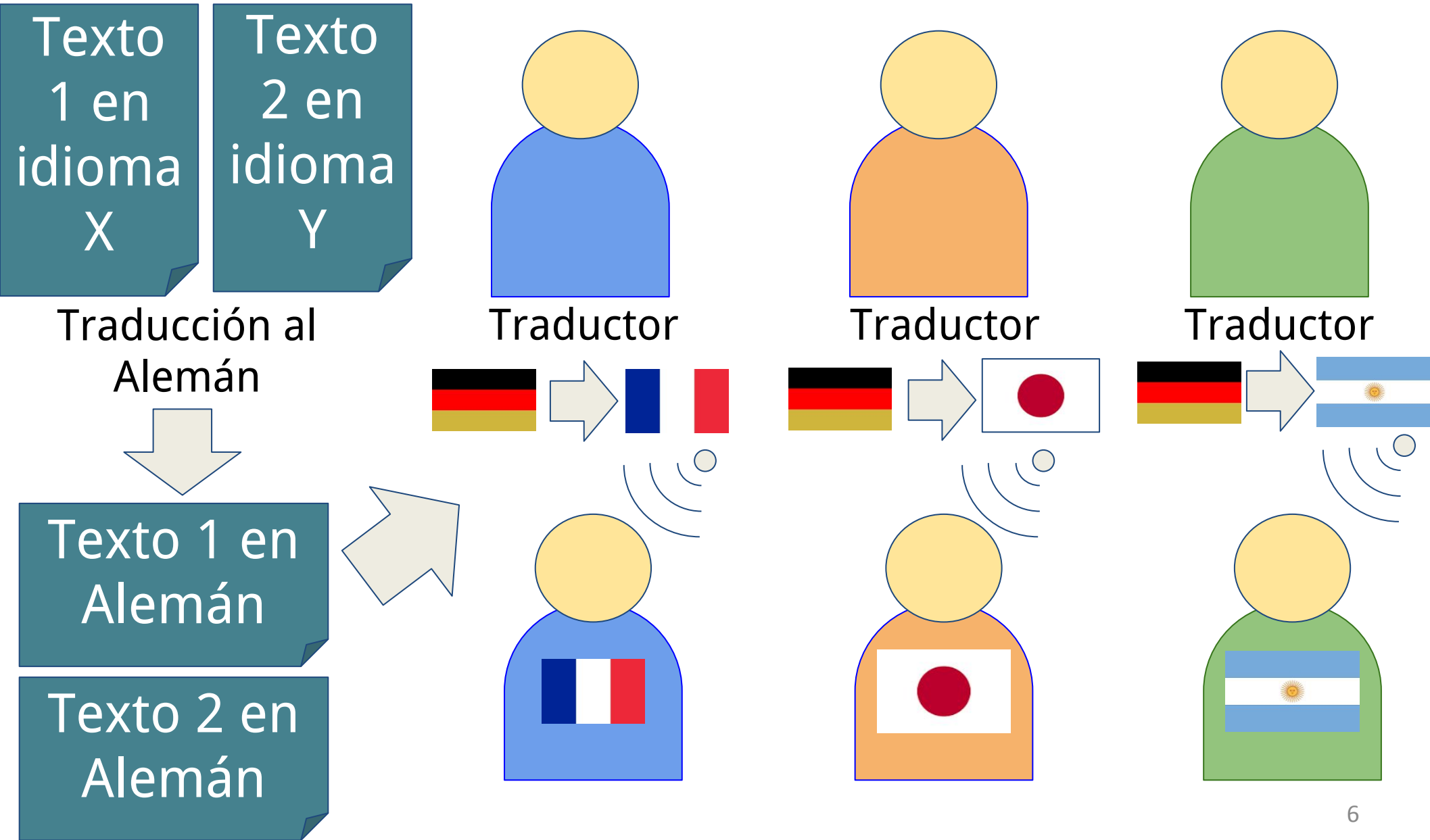
Texto 2 en  
Francés



Texto 2 en  
Japonés

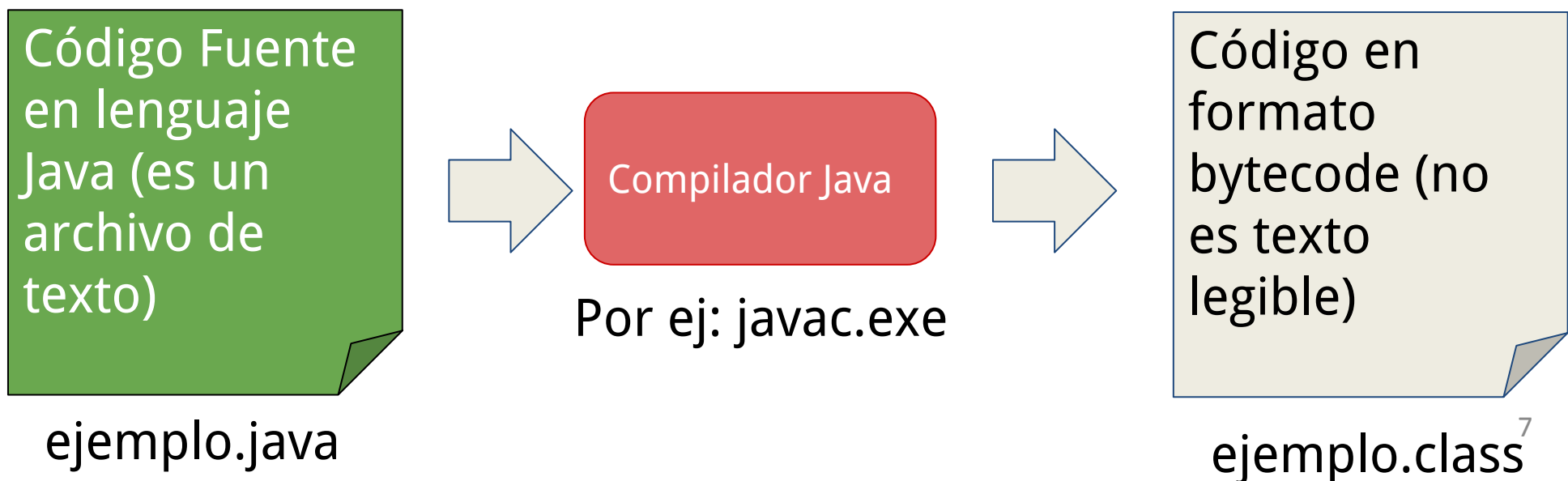


Texto 2 en  
Español



# Multiplataforma: Compilación a bytecode

- La premisa con la cual se diseñó Java es la de poder ejecutar aplicaciones escritas para dicha plataforma en cualquier dispositivo (PC, teléfonos, tablets, sensores, televisores, etc.). Una característica que se denomina **Write Once-Run Anywhere**.
- El código se compila (usando un compilador) en una “representación intermedia”, denominada bytecode de Java.



# ¿Por qué bytecode?

- Java es un **lenguaje de programación**, y como tal, está diseñado para ser leído por humanos: desde el punto de vista de una computadora, es verborrágico (tiene elementos sintácticos que ayudan al humano, pero que una computadora no necesita).
- Bytecode es el **conjunto de operaciones** que “entiende” la plataforma Java (específicamente, la máquina virtual).
- Originalmente solo Java era compilado a bytecode, pero debido a que la especificación de bytecode está disponible, **se extendió el uso a otros lenguajes**.
- ¿Solo se puede compilar Java a bytecode? No, es posible compilar a código nativo (por ej, generar un .exe para AMD de 64 bits). El lenguaje está separado de cómo el programa se ejecuta.



# ¿Cómo se ve el lenguaje Java?

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

# ¿Cómo se ve el bytecode?

```
Ëp°%NUL NUL NUL 4 NUL GS
NUL ACK NUL SI      NUL DLE NUL DC1 BS NUL DC2
NUL DC3 NUL DC4 BEL NUL NAK BEL NUL SYN SOH NUL ACK<init> SOH NUL ETX () V SOH NUL EOT Code SOH
NUL SILineNumberTable SOH NUL EOT main SOH NUL SYN ([Ljava/lang/String;) V SOH NUL
SourceFile SOH NUL DC1 MiAplicacion.java FF NUL BEL NUL BS BEL NUL ETB FF NUL CAN NUL EM SOH
NUL ENOHola! BEL NUL SUB FF NUL ESC NUL FS SOH NUL FF MiAplicacion SOH NUL DLE java/lang/Obj
ect SOH NUL DLE java/lang/System SOH NUL ETX out SOH NUL NAK Ljava/io/PrintStream; SOH NUL
DC3 java/io/PrintStream SOH NUL BEL println SOH NUL NAK (Ljava/lang/String;) V NUL ! NUL
ENONUL ACK NUL NUL NUL NUL NUL STX NUL SOH NUL BEL NUL BS NUL SOH NUL
NUL NUL NUL GS NUL SOH NUL SOH NUL NUL NUL ENO* - NUL SOH + NUL NUL NUL SOH NUL
NUL NUL NUL ACK NUL SOH NUL NUL NUL SOH NUL      NUL VT NUL FF NUL SOH NUL
NUL NUL NUL % NUL STX NUL SOH NUL NUL NUL      - NUL STX DC2 ETX 1 NUL EOT + NUL NUL NUL SOH NUL
NUL NUL NUL
NUL STX NUL NUL NUL ETX NUL BS NUL EOT NUL SOH NUL
NUL NUL NUL STX NUL SO
```

# Se puede volver a transformar para la lectura humana

Compiled from "MiAplicacion.java"

```
public class MiAplicacion {  
    public MiAplicacion();
```

Code:

```
    0: aload_0  
    1: invokespecial #1 // Method java/lang/Object."<init>":()V  
    4: return
```

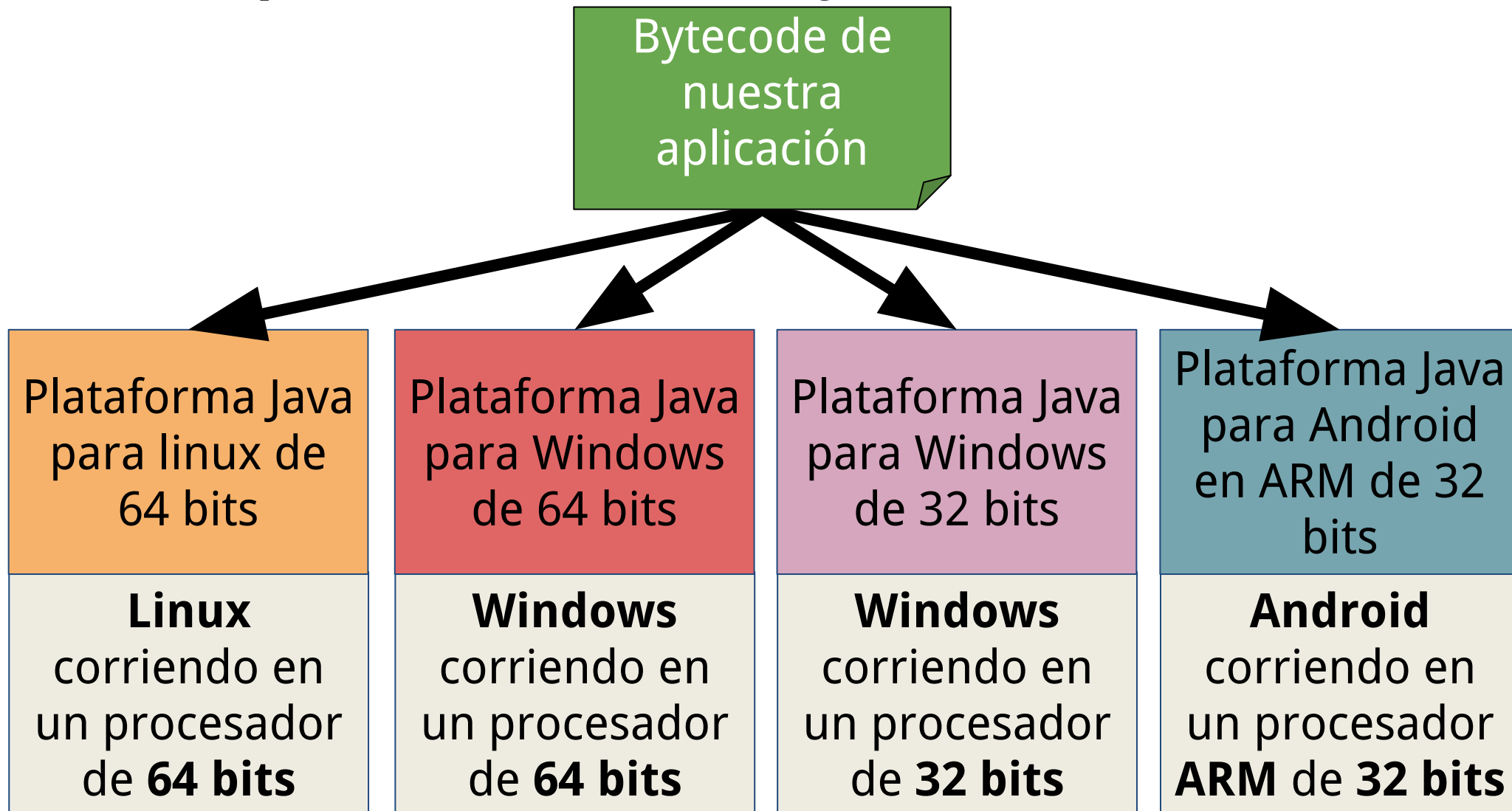
```
public static void main(java.lang.String[]);
```

Code:

```
    0: getstatic      #2 // Field java/lang/System.out:Ljava/io/PrintStream;  
    3: ldc            #3 // String Hola!  
    5: invokevirtual #4 // Method java/io/PrintStream.println:(Ljava/lang/String;)V  
    8: return
```

```
}
```

# Multiplataforma: Ejecución



# Java Standard Edition (Java SE)

- Es una de las versiones más difundidas de la plataforma Java.
- Incluye todo lo necesario para ejecutar programas escritos en Java en una PC o un servidor.
- Viene en 2 formatos para instalar en un dispositivo:
  - **JRE (Java Runtime Environment):** solo para ejecutar aplicaciones Java (lo debería tener instalado el usuario final de nuestra aplicación).
  - **JDK (Java Development Kit):** contiene el JRE y, además, software que usan los desarrolladores para monitorear y encontrar errores en los programas Java (lo deberíamos tener instalado nosotros).

# Máquina Virtual de Java

- La Java Virtual Machine (JVM) o Máquina Virtual de Java **es la parte central del JRE.**
- Por ejemplo, si descargamos Java SE para Windows, la máquina virtual se encuentra en el ejecutable “java.exe”, al cual **se le pasa como argumento nuestro programa en formato bytecode.**
- En términos sencillos, el **sistema subyacente ejecuta la máquina virtual y la máquina virtual ejecuta nuestro programa.** En Windows podríamos hacer lo siguiente (la primer línea compila a bytecode nuestro código y la segunda ejecuta el bytecode en la JVM):

```
C:\> javac.exe MiProgramaEnJava.java
```

```
C:\> java.exe MiProgramaEnJava
```

# Máquina Virtual de Java

- Al igual que el ejemplo de la traducción, **una vez que generamos el bytecode, nuestro programa no requiere modificación alguna**. Solo requiere que el usuario tenga instalado un JRE en su sistema.
- La **máquina virtual de java es genérica** (aunque su nombre indique lo contrario). Es decir, **soporta cualquier lenguaje que se pueda compilar en bytecode** (por ej: Scala, Groovy, Clojure).

# Manejo de Memoria en Java

- En general, cuando un programa pide un espacio de memoria RAM (en bytes) para almacenar datos, por ejemplo, una cadena de caracteres que diga “Hola!”. El sistema (Windows, Linux, Mac) guarda ese espacio de memoria hasta que el programa finaliza.
- Debido a que la memoria **RAM es limitada**, en muchos lenguajes el desarrollador es el encargado de **decidir cuando no utiliza más un espacio de memoria**.
- Esta última es una tarea que genera muchísimos errores en los programas, sobre todo porque el desarrollador muchas veces olvida liberar memoria.





# Manejo de Memoria en Java

- En Java, el desarrollador no se preocupa por liberar memoria, ya que la máquina virtual detecta cuando un objeto deja de ser utilizado, y automáticamente marca el espacio como libre.
- A este mecanismo se le llama **recolección de basura** o **garbage collection** y, al programa que hace la recolección se lo llama **recolector de basura** o **garbage collector (GC)**.
- El GC facilita mucho el desarrollo y reduce errores de uso de memoria



# Java es un estándar *de facto*

- No está manejado por un organismo de estandarización (ej. IEEE).
- Existe un comité (Java Community Process executive committee) que analiza e introduce mejoras al estándar.
- Dependiendo la versión, se incluyen ciertas mejoras en el lenguaje Java, en la máquina virtual y en las APIs soportadas (más sobre esto en la próxima filmación)
- Actualmente, la última versión disponible es Java SE 8.

# Application Programming Interface

- Es un término genérico para referirse a **cómo se comunican diferentes componentes de software y cómo los puede utilizar el desarrollador.**
- **En Java SE, hay muchas APIs diferentes,** organizadas en “paquetes”. Por ejemplo, la API de manejo de archivos define cómo se abre o crea un archivo, cómo se borra, cómo se mueve de carpeta, etc.
- La idea es que una vez que los programadores empiezan a usar una API, **ésta no cambie entre versiones.** De otra manera, es más difícil usar nuestro programa en versiones de Java más nuevas.

# Paquetes de la edición estándar

Las diferentes APIs que componen la edición estándar de Java están agrupadas en paquetes, según su función. Algunos ejemplos:

- **java.lang**: Todo lo necesario para ejecutar Java y soportar el lenguaje.
- **java.io**: Manejo de entrada/salida general: archivos y streams.
- **java.math**: Todo lo necesario para cálculo matemático.
- **java.util**: Contiene muchas utilidades (organizadas en paquetes también). Por ejemplo, contiene toda la API de Collections (colecciones de Java), la cual abarca listas de elementos, tablas de hash, árboles, etc.

# ¿Qué necesito para desarrollar en Java?

- Un JDK.
  - Se descarga de la página de Oracle.
  - En Google se puede buscar: Java SE.
  - No confundir con el JRE, que viene con menos herramientas de desarrollo.

[Overview](#) [Downloads](#) [Documentation](#) [Community](#) [Technologies](#) [Training](#)

## Java SE Development Kit 8 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, applets, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- [Java Developer Newsletter](#): From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- [Java Developer Day](#) hands-on workshops (free) and other events
- [Java Magazine](#)

JDK 8u121 checksum

### Java SE Development Kit 8u121

You must accept the [Oracle Binary Code License Agreement for Java SE](#) to download this software.

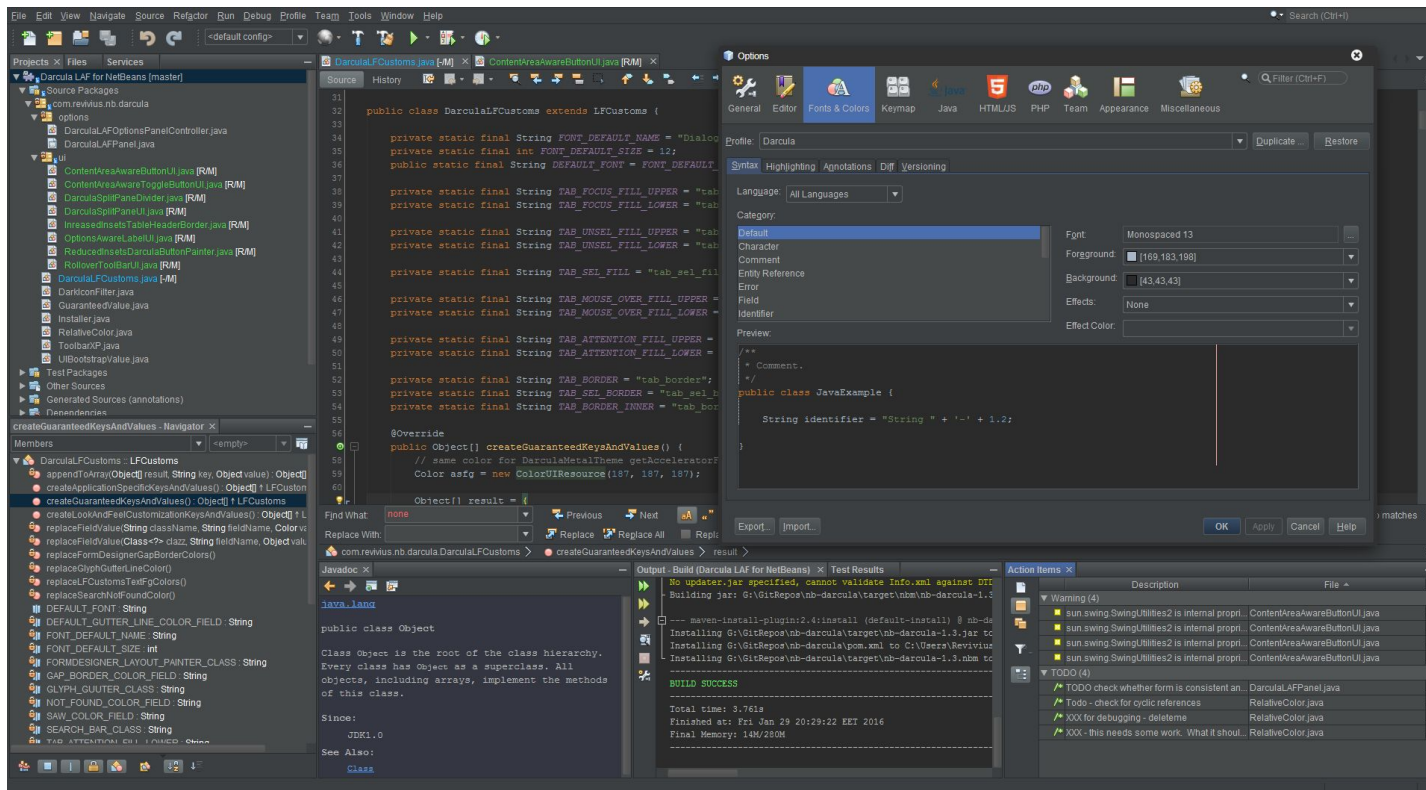
☐ Accept License Agreement ☒ Decline License Agreement

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.86 MB	<a href="#">jdk-8u121-linux-arm32-vfp-hflt.tar.gz</a>
Linux ARM 64 Hard Float ABI	74.83 MB	<a href="#">jdk-8u121-linux-arm64-vfp-hflt.tar.gz</a>
Linux x86	162.41 MB	<a href="#">jdk-8u121-linux-i586.rpm</a>
Linux x86	177.13 MB	<a href="#">jdk-8u121-linux-i586.tar.gz</a>
Linux x64	159.96 MB	<a href="#">jdk-8u121-linux-x64.rpm</a>
Linux x64	174.76 MB	<a href="#">jdk-8u121-linux-x64.tar.gz</a>
Mac OS X	223.21 MB	<a href="#">jdk-8u121-macosx-x64.dmg</a>
Solaris SPARC 64-bit	139.64 MB	<a href="#">jdk-8u121-solaris-sparcv9.tar.Z</a>
Solaris SPARC 64-bit	99.07 MB	<a href="#">jdk-8u121-solaris-sparcv9.tar.gz</a>
Solaris x64	140.42 MB	<a href="#">jdk-8u121-solaris-x64.tar.Z</a>
Solaris x64	96.9 MB	<a href="#">jdk-8u121-solaris-x64.tar.gz</a>
Windows x86	189.36 MB	<a href="#">jdk-8u121-windows-i586.exe</a>
Windows x64	195.51 MB	<a href="#">jdk-8u121-windows-x64.exe</a>



# ¿Qué necesito para desarrollar en Java?

- Algún software donde escribir código Java:
  - Puede ser un editor de texto, pero es preferible un entorno de desarrollo integrado (IDE).
  - Ejemplo: Eclipse, JDeveloper, IntelliJ IDEA, NetBeans.



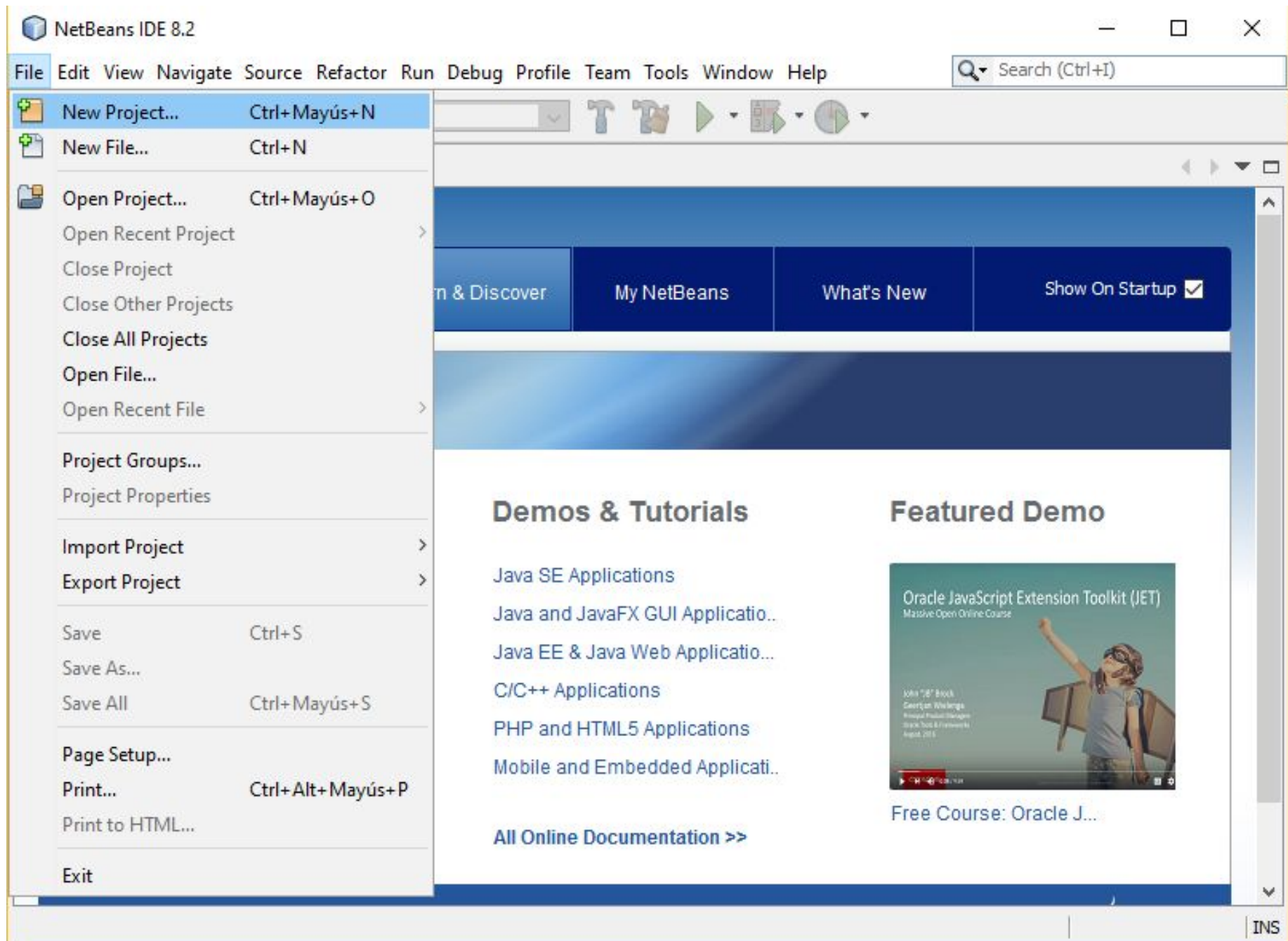
# IDE

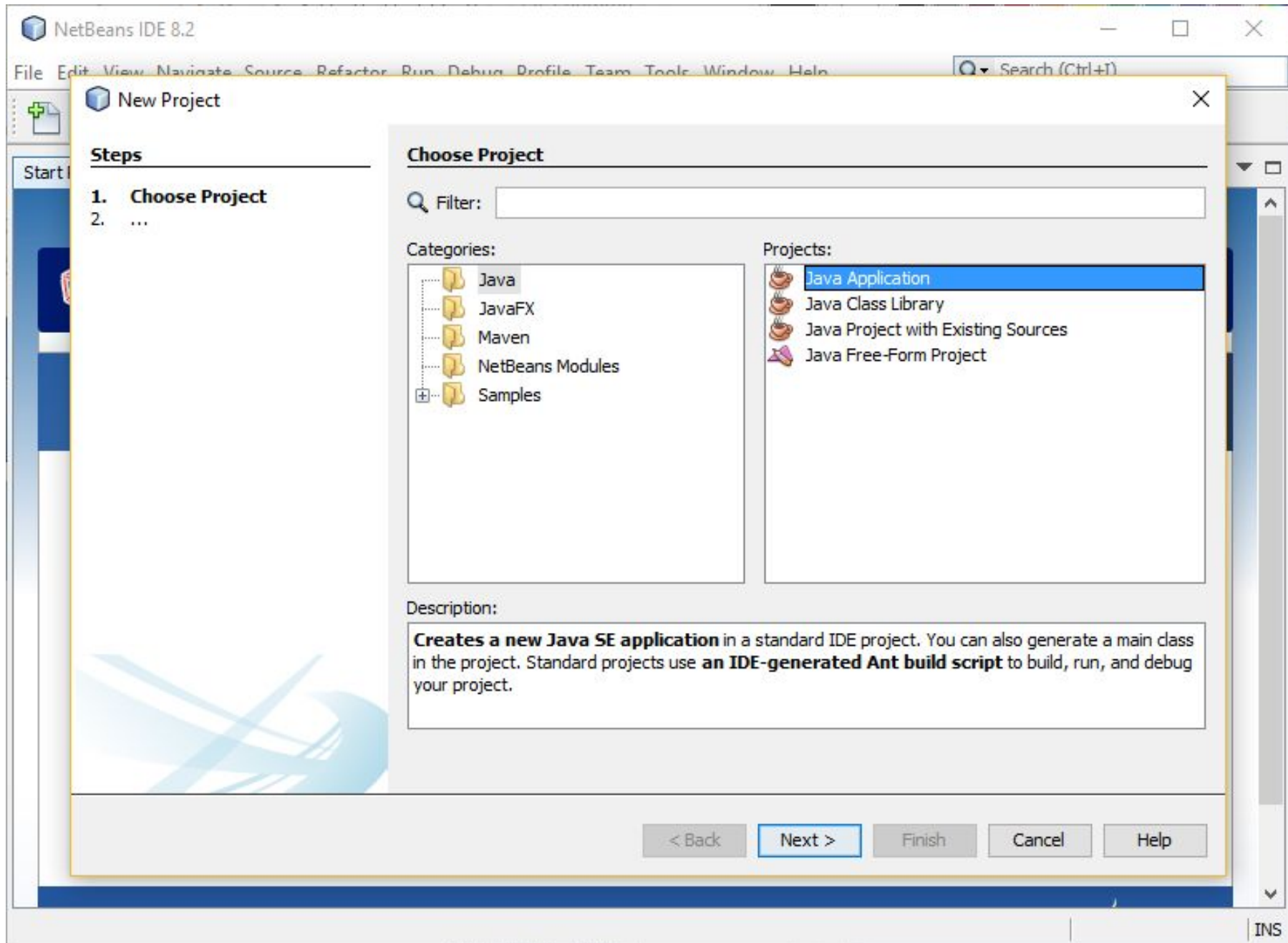
- Un software que permite editar, ejecutar y monitorear un programa en un lenguaje dado.
- El IDE para Java que vamos a utilizar en este curso es **NetBeans**.
- Algunas características:
  - **Marcado de sintaxis:** Permite distinguir distintas partes del código de acuerdo a su significado. Por ejemplo, la palabra **class** en Java, es una palabra reservada (tiene un uso específico) y se muestra en otro color.
  - **Marcado de errores:** Se pueden ver los errores sintácticos en el código en la posición del texto donde ocurren.
  - **Ejecución de la máquina virtual desde la interfaz:** se puede correr nuestra aplicación sin salir del entorno.
  - **Debugging:** Podemos hacer un seguimiento de lo que hace la aplicación y controlar paso a paso las operaciones para encontrar errores.

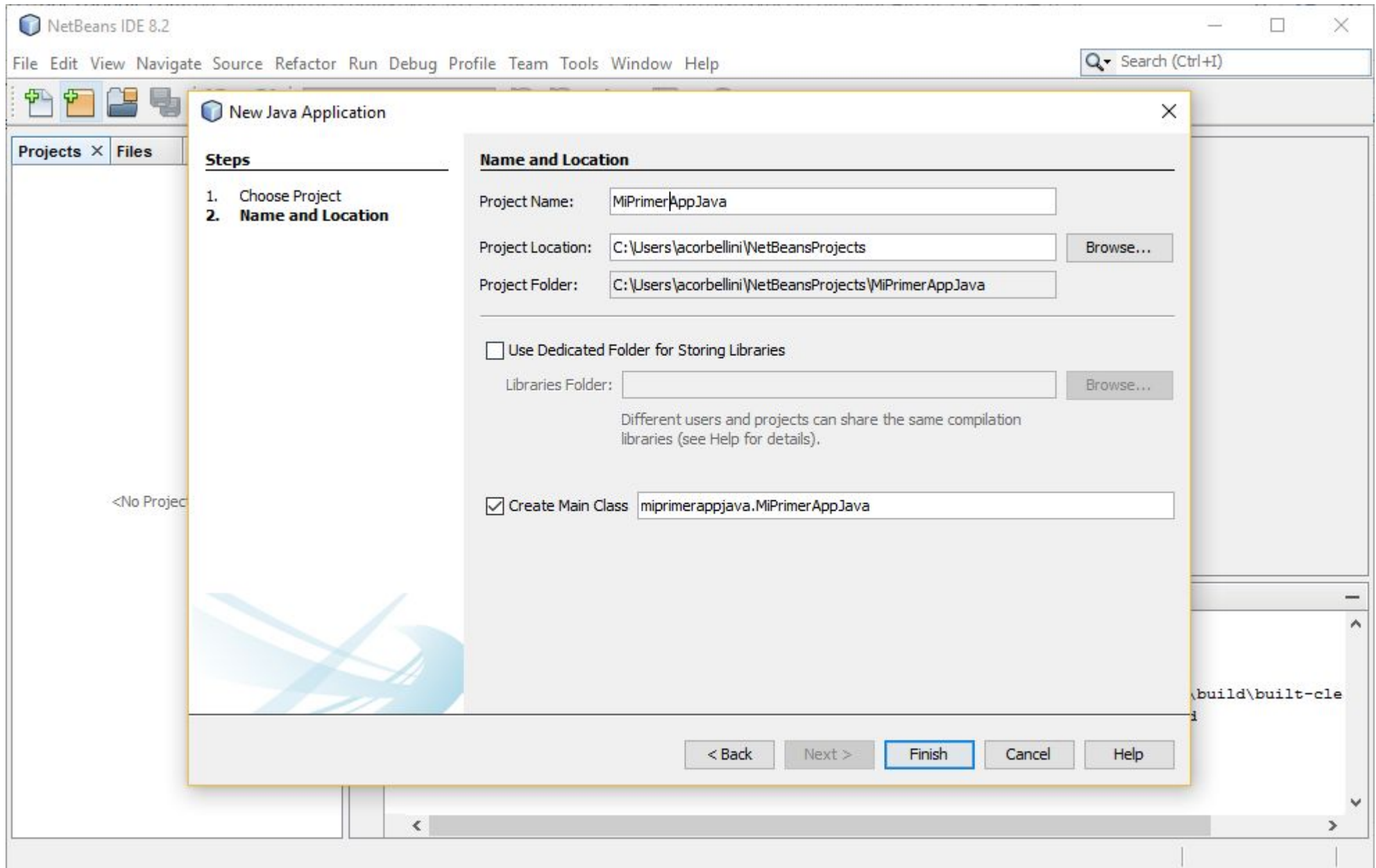
# Mi primer programa en Java

- Vamos a implementar un hola mundo en Java, usando Netbeans.
- Los pasos para crear una aplicación Java suele ser similar en todos los IDEs:
  - Crear un proyecto (por ejemplo, mi aplicación del clima, o mi aplicación de pizzería).
  - Crear el primer archivo de nuestra aplicación.
  - Codear en Java.
  - Verificar que la aplicación no tenga errores de compilación.
  - Ejecutar y probar que funciona correctamente.









# Lo primero es el orden

- El lenguaje Java permite organizar diferentes porciones de código en carpetas. Por ejemplo, podemos tener un archivo con el código que calcula el precio de un pedido por un lado, y por otro el código que permite administrar las facturas emitidas.
- NetBeans automáticamente nos sugiere “**miprimerappjava**” como el primer paquete de nuestro proyecto y, además, nos crea el primer archivo, llamado **MiPrimerAppJava.java**, el cual será **el punto de entrada a la aplicación** (lo primero que se ejecuta).

☒ Create Main Class `miprimerappjava.MiPrimerAppJava`

MiPrimerAppJava - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects Files Services

MiPrimerAppJava

- Source Packages
  - miprimerappjava
    - MiPrimerAppJava.java
- Libraries

Navigator

Members

MiPrimerAppJava

- main(String[] args)

Source History

```
1  /*
2  * To change this license header, choose License Headers in Project Properties
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package miprimerappjava;
7
8  /**
9   *
10   * @author acorbellini
11   */
12  public class MiPrimerAppJava {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19      }
20
21  }
22
```

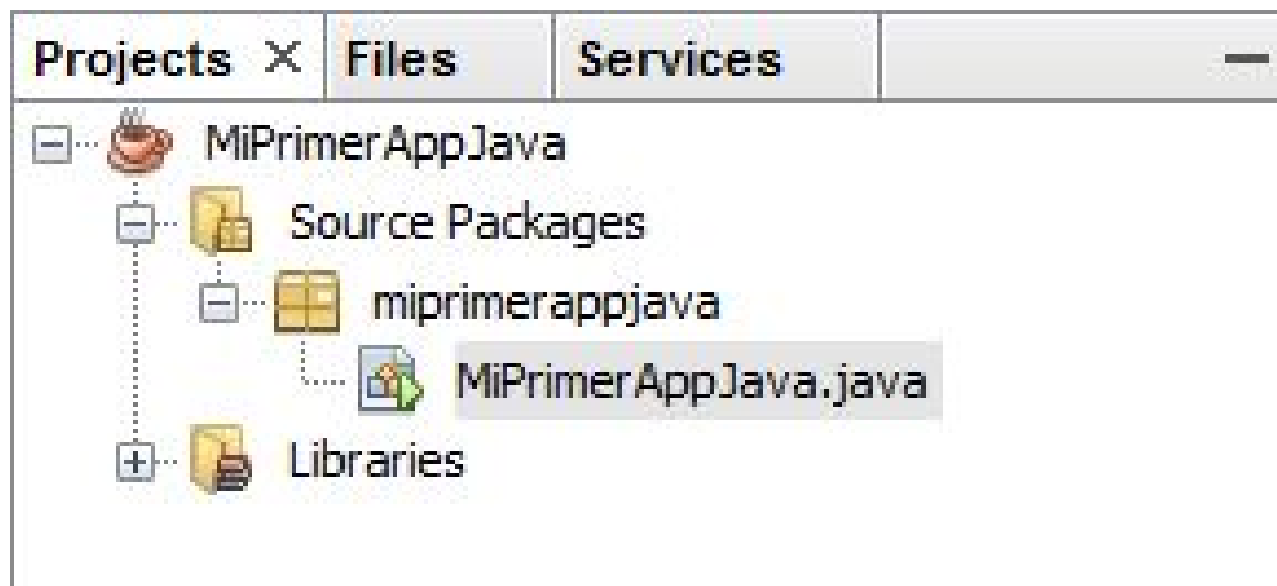
Output

1:1 INS




# Paquetes

- En panel de Projects se puede ver el nombre de nuestro proyecto **MiPrimerAppJava** (**puede tener cualquier nombre, incluso con espacios**), dentro del proyecto hay una carpeta “Source Packages” que contendrá todo el código fuente. Finalmente vemos el paquete **miprimerappjava** y dentro nuestro primer archivo **MiPrimerAppJava.java**.



# MiPrimerAppJava.java

- En este caso, NetBeans la creó automáticamente. Podemos crearla a mano, usando el menú File -> New File... ó el ícono 
- Allí, elegimos la opción Java Class y le ponemos el nombre que deseemos (por convención arrancan con una letra mayúscula).
- El punto de entrada se puede identificar buscando el método main. Si no está, lo podemos escribir nosotros.

# MiPrimerAppJava.java

- En el archivo aparecen varias cosas (quitando los comentarios que aparecen en gris):
  - a. en la primer línea aparece el nombre del paquete donde está el archivo
  - b. luego el nombre de la clase MiPrimerAppJava
  - c. dentro de la clase, el método **main**, que es el punto de entrada de la app.

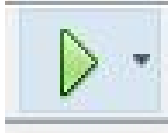
```
package miprimerappjava;  
  
public class MiPrimerAppJava {  
  
    public static void main(String[] args) {  
  
    }  
}
```



# El método **main**

- Es el punto de entrada de una aplicación Java.
- Solo puede haber **un main por clase** (puede haber varios main por proyecto).
- La máquina virtual necesita saber en qué clase tiene que **buscar el método main** de nuestra aplicación.
- **NetBeans nos simplifica** esto al hacer click en el botón Run (automáticamente busca una clase que tenga un main o toma la clase actual)

# Ejecutar el código en NetBeans

- Entonces, antes de continuar, probemos que el código que tenemos, funciona.
- Hacer click en el botón “Run” 
- Comienza la compilación
- Inmediatamente después, el código ejecuta en la máquina virtual de Java.
- Todo este proceso lo hace NetBeans, usando los ejecutables que vienen en el JDK (por ej, el compilador javac.exe y la máquina virtual java.exe).

MiPrimerAppJava - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

<default config>

Projects Files Services

MiPrimerAppJava

- Source Packages
  - miprimerappjava
    - MiPrimerAppJava.java
- Libraries

Navigator

Members

MiPrimerAppJava

- main(String[] args)

MiPrimerAppJava.java

Source History

```
1  /*
2  * To change this license header, choose License Headers in Project Properties
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package miprimerappjava;
7
8  /**
9   *
10  * @author acorbellini
11  */
12  public class MiPrimerAppJava {
13
14      /**
15       * @param args the command line arguments
16      */
```

Output - MiPrimerAppJava (run)

```
run:
BUILD SUCCESSFUL (total time: 0 seconds)
```

1:1 INS

# No sucede nada...

- El código que tenemos **no muestra nada por pantalla, ni por ningún otro medio** (un archivo por ej.).
- Mostremos un texto arbitrario por “consola”, es decir, por la interfaz de texto de NetBeans (el panel que dice Output).
- Vamos a usar una API de Java que permite imprimir cosas por pantalla. Ya existen algunas utilidades que hacen la tarea simple, siendo una de ellas la **clase System**.
- Dentro de System, tenemos el **objeto “out”**, que es la salida de texto. Dicho objeto, tiene un método que se llama **println**, es decir, **imprimir una línea**.

# println

El método pertenece a la clase **PrintStream** (es la clase del objeto “out”) y tiene una API definida. **Recibe un texto** definido por el programador (un objeto del tipo String, es decir, una cadena de caracteres) e **imprime dicho texto por la salida de texto** en donde estemos ejecutando la aplicación, en este caso NetBeans.

En resumen, vamos a hacer que aparezca un texto en el panel Output de NetBeans.

MiPrimerAppJava - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config>

Projects Files Services

- MiPrimerAppJava
  - Source Packages
    - miprimerappjava
      - MiPrimerAppJava.java
  - Libraries

main - Navigator

Members

- MiPrimerAppJava
  - main(String[] args)

MiPrimerAppJava.java

```
12 public class MiPrimerAppJava {
13
14     /**
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         System.out.println(";Hola Mundo!");
19     }
20
21 }
22
```

Output - MiPrimerAppJava (run)

```
run:
;Hola Mundo!
BUILD SUCCESSFUL (total time: 0 seconds)
```

18:44 | INS

# Argumentos del programa

- Dentro de los paréntesis del método **main** viene una variable de tipo `String[]`.
- Los argumentos de programa permiten recibir datos externos del usuario, tanto datos a procesar como preferencias o configuraciones.
- Por ejemplo, la variable de tipo `String[]` puede contener **[“¡Hola Mundo!”, “en\_mayúsculas”, “sin\_espacios”]**. En este ejemplo, el primer argumento es “¡Hola Mundo!”, que es el mensaje a imprimir, y los siguientes pueden ser opciones del programa.
- Para acceder a cada argumento:
  - Identificar la variable, por ejemplo, si la declaración es **`String[] argumentos`**, la variable se llama **argumentos**.
  - `argumentos[0]` accede al 1er argumento, `argumentos[1]` accede al 2do argumento.



# Ejemplo en NetBeans

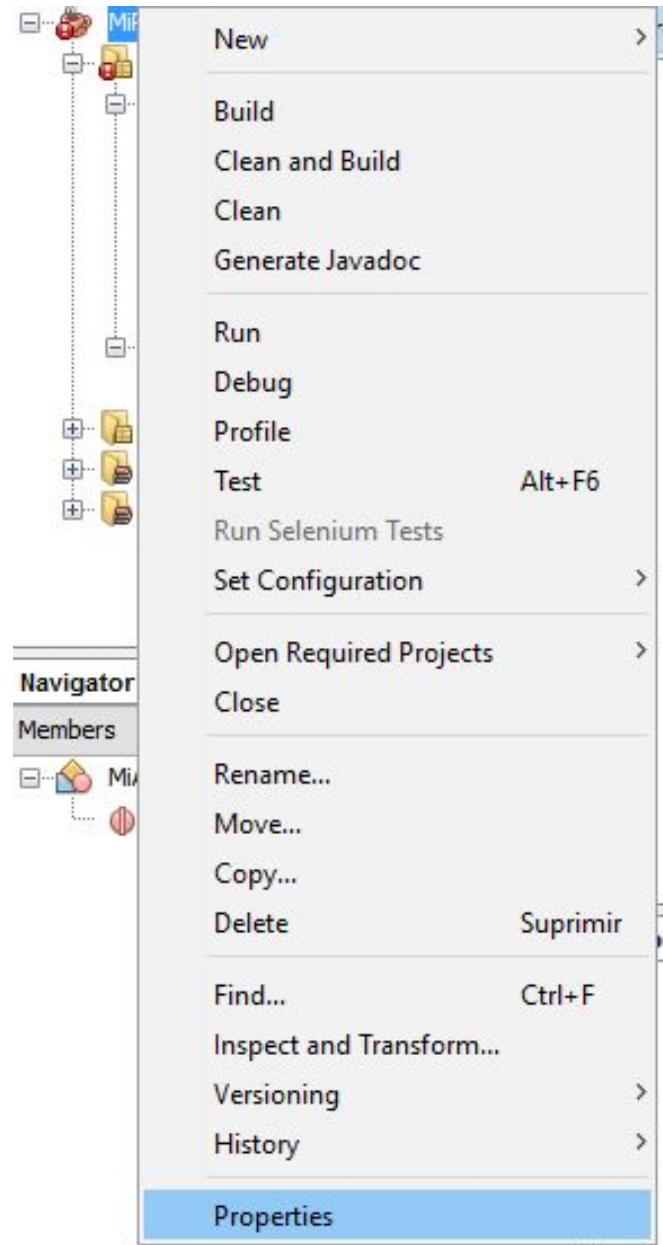
```
public class MiAplicacion {  
    public static void main(String[] args) {  
        System.out.println(args[0]);  
    }  
}
```

- En este ejemplo, la variable que contiene los argumentos del programa se llama **args** (de **argumentos**).
- De esa manera, el primer argumento es args[0] (se lee “args” sub cero). Los arreglos comienzan en la posición 0, como se verá más adelante.



# ¿Dónde se ponen los argumentos en NetBeans?

- En general, los argumentos de un programa pueden ser establecidos desde consola o, por ejemplo, en Windows en las opciones de un acceso directo.
- Debido a que NetBeans es quien invoca a la máquina virtual con nuestra aplicación, se deben poner los argumentos desde la interfaz.
- Ir a las **Propiedades del proyecto** (click derecho en el proyecto y luego en Propiedades).
- Una vez en la pantalla, ir a la sección **Run**, y colocar en el campo **Arguments** los argumentos deseados (tener cuidado de poner aquellos argumentos con espacios entre comillas ya que los espacios separan argumentos - args[0], args[1], etc. ).



Project Properties - MiPrimerAppJava

Categories:

- Sources
- Libraries
- [-] Build
  - Compiling
  - Packaging
  - Deployment
  - Documenting
- [-] Run
  - Application
    - Web Start
  - License Headers
  - Formatting
  - Hints

Configuration: <default config>

New...

Delete

Runtime Platform: Project Platform

Manage Platforms...

Main Class: edu.otro.MiAplicacion

Browse...

Arguments: "¡Hola Mundo!"

Working Directory:

Browse...

VM Options:

Customize...

(e.g. -Xms10m)

☐ Run with Java Web Start

(To run and debug the application with Java Web Start, first enable Java Web Start)

OK

Cancel

Help

# Resumen

- Vimos cómo se compone la plataforma Java (JDK, JRE, máquina virtual, bytecode, el lenguaje Java, etc.)
- Vimos cómo crear un proyecto en NetBeans.
- Vimos cómo ejecutar una aplicación Java en la máquina virtual de Java, todo a través de NetBeans.
- Mostramos “¡Hola Mundo!” en el panel de salida de NetBeans usando las utilidades de la plataforma.
- Además, vimos cómo utilizar los argumentos recibidos en el método main.