Programación sobre redes

2018

FORMANDO TÉCNICOS DE CALIDAD PARA HACER LA DIFERENCIA

Introducción

por Pablo Ezequiel Jasinski

Es un hecho que en la actualidad la **computadora** se ha transformado en un elemento más en la vida cotidiana de las personas. La mayoría de los aparatos electrónicos cuentan con una computadora programada para realizar ciertas tareas referentes a su propósito. Desde dispositivos para realizar trabajos que antes realizaban los humanos, potentes calculadoras para diferentes especialidades, electrodomésticos en los hogares, consolas de juegos para el ocio y la diversión, ordenadores personales para múltiples fines, hasta un sinfín de dispositivos electrónicos son los que se encuentran habitualmente en el presente.

Toda esta tecnología que hoy se considera como algo normal, tuvo un inicio y una evolución, y desde el surgimiento de las primeras computadoras, fue fundamental (*y lo sigue siendo*) el papel del "**programador**", que es el encargado de escribir código entendible por la máquina para que la misma pueda realizar determinadas tareas.

Fig. 1. Telar de Jacquard

Los primeros programadores realizaban sus programas mediante **tarjetas perforadas**, idea basada en el **telar de Jacquard**, el cual utilizaba las mismas para representar el movimiento de brazos mecánicos, con el objetivo de generar patrones de dibujo en la tela. Estas tarjetas se usaban para introducir información e instrucciones en las computadoras mediante el *sistema binario*, en donde los orificios representaban al bit "0", mientras que los lugares que carecían de los mismos eran interpretados como el bit "1".

Por lo tanto hasta ese entonces no existían lenguajes de programación propiamente dichos, sino que se utilizaba el denominado lenguaje de máquina, el cual no es más que un sistema de códigos en forma de ceros y unos que la computadora interpreta como instrucciones o conjunto de datos.

Como programar directamente sobre lenguaje de máquina era una tarea tediosa, lenta, que conllevaba gran número de errores y una difícil depuración, a principios de los años 50 se introdujo el **código de ensamblaje**, el cual era una notación simbólica que representaba las operaciones que podía realizar la computadora. Al principio la traducción del código de ensamblaje a código de máquina se hacía manualmente, pero al ver que de esta tarea podía encargarse también la computadora se creó un programa traductor llamado **ensamblador**, punto donde comienza aparecer la abstracción en informática.

La abstracción ignora selectivamente ciertas partes de un todo para un mejor entendimiento. Por ejemplo, para sumar dos números en lenguaje ensamblador se usa la instrucción "*add destino, fuente*". Lo que hace esta instrucción es sumar dos operandos y guardar el

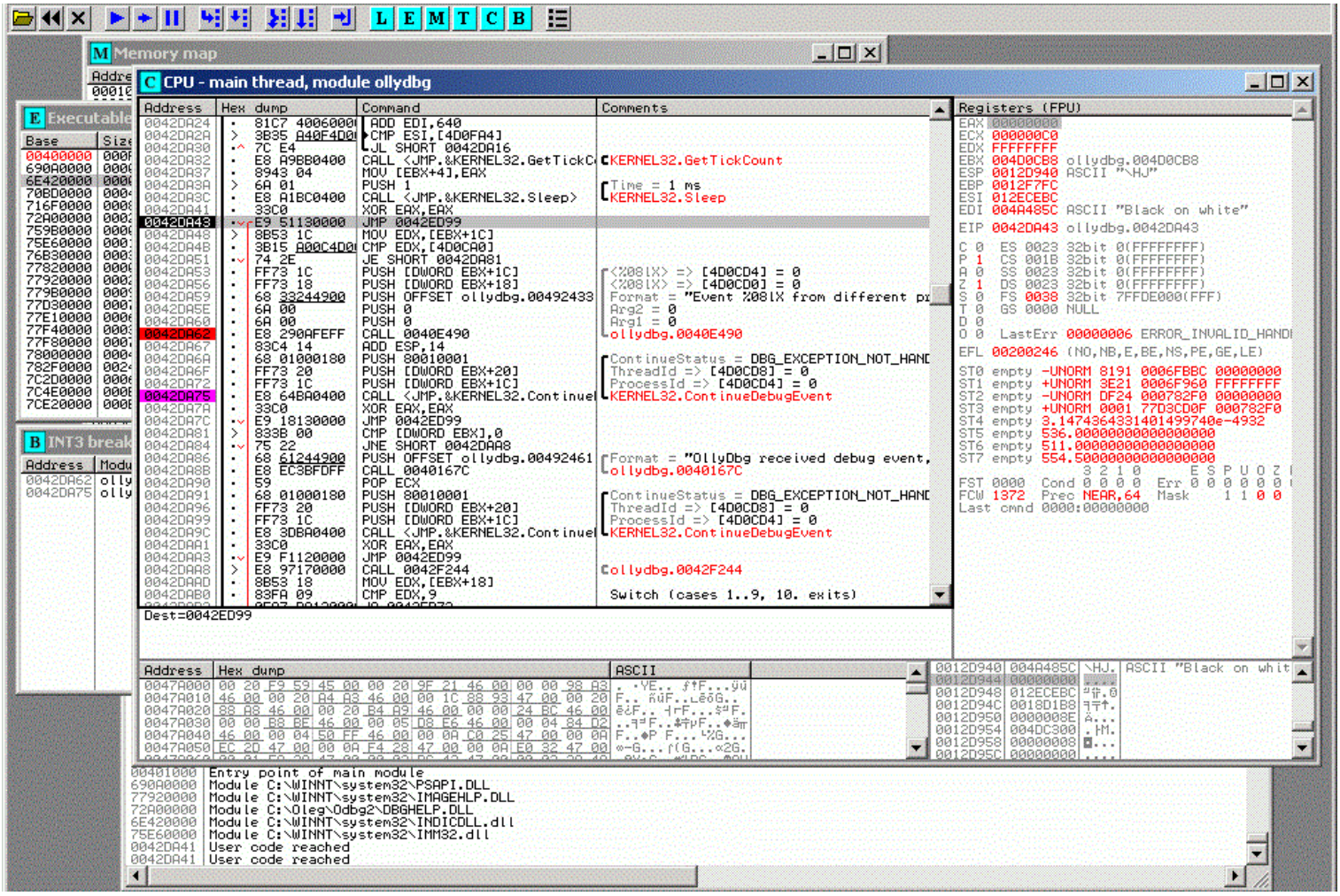


Fig. 2. Depurador de código ensamblador

resultado en el operando destino, por lo tanto si se cuenta con la instrucción "*add dl, 30h;*" se entiende que se suma el operando *dl* con el operando *30h*, y que el resultado se guarda en *dl*, y he aquí la esencia de la abstracción. Preste atención a este fragmento del último párrafo: *"se entiende que se suma el operando dl con el operando 30h, y que el resultado se guarda en dl"*, se conoce **que se hace**, pero no es relevante **como se hace**, debido a que se ignoran esos detalles, por lo tanto se usa un cierto nivel de **abstracción**.

Este lenguaje tiene las características de ser de bajo nivel de abstracción ya que permite controlar directamente el hardware del dispositivo electrónico y depende de la arquitectura del mismo. A partir de este punto los desarrolladores e investigadores comenzaron a pensar en lo que hoy se conoce como **lenguajes de programación**, con el objetivo de que el programador escriba líneas de código en un lenguaje más cercano al humano, y tenga una *flexibilidad* mayor a la hora de programar, ocupándose más del problema en sí a resolver que en el hardware sobre el cual tiene que programar, en otras palabras en un mayor nivel de abstracción.

Es en este punto donde comienzan a surgir los lenguajes de programación de medio nivel de abstracción y de alto nivel de abstracción, los cuales son muy populares en la actualidad, como por ejemplo *C, C++, Basic, Fortran, Cobol, Pascal, Lisp, Visual Basic, HTML, XML, PHP, Java*, etc.

Los lenguajes de programación de alto nivel de abstracción como *Fortran, Cobol, Pascal, Basic, Python, Perl, PHP, Java, C++, C#,* entre otros, son lenguajes en los cuales la mera lectura del código puede ser entendida por cualquier persona ya que se asimila más al lenguaje natural humano que al lenguaje de máquina, y que están **orientados a objetos**, los cuales emergen de **procedimientos**.

Los lenguajes de programación de medio nivel de abstracción como *BCPL, C y Basic*, están en un intermedio entre los lenguajes de bajo nivel y los de alto nivel. Si bien estos lenguajes a menudo se los considera como de alto nivel, poseen la característica de poder programar directamente sobre el hardware como se hacía en lenguajes de bajo nivel y al mismo tiempo poder realizar operaciones de alto nivel. A diferencia de los lenguajes de alto nivel que están orientados a objetos, estos están orientados a **procedimientos**, los cuales se componen de **procesos**.

Más allá de los diferentes tipos de lenguajes que existen y las características que tenga cada uno, todos ellos tienen algo en común: Un **intérprete** o un **compilador**. Como se mencionó anteriormente el ordenador comprende un solo lenguaje, el lenguaje de máquina, conformado por bits, por lo tanto debe existir un programa informático que se encargue de traducir el código hecho en algún lenguaje en particular al código de máquina, en ceros y unos. Este programa traductor es el intérprete o bien el compilador.

Un intérprete es un programa que realiza la traducción a medida que sea necesaria, línea por línea, y por lo general no guardan el resultado de la traducción. La ventaja del mismo es que un mismo

código podrá ejecutarse en cualquier sistema, así sean muy diferentes entre ellos, pero la desventaja es que el intérprete se encontrará cargado en memoria todo el tiempo para poder hacer esta traducción, factor que hace que un programa interpretado sea más lento que uno compilado.

Un compilador es un programa que, a diferencia del intérprete, realiza toda la traducción de un archivo fuente de una sola vez. La ventaja es que al no tener cargado un intérprete en memoria, el programa será más rápido, pero la desventaja es que el ejecutable generado por el compilador solo funcionará en el sistema específico para el que se haya compilado el código.

Un ejemplo sencillo para comprender la función de un intérprete y de un compilador es el de un traductor. Imagine que desea ir a otro país cuya lengua desconoce, y necesita comunicarse con personas de ese país que desconozcan su lengua. Si usted fuese a *Rusia* y necesita comunicarse con una persona que habla en ruso y desconoce su idioma, una manera de poder hacerlo es contar con una persona que haga de interprete, así todo lo que usted diga será traducida y comunicada a la persona de ese país a medida que vaya hablando, y viceversa. Por lo tanto mientras exista el intérprete se podrá entablar una conversación. De forma parecida funciona un intérprete en informática.

Ahora suponga que en vez de establecer una comunicación oral desea establecer la misma de forma escrita, escribiendo un documento, el cual podría llegar a muchas personas en *Rusia*. En este caso no tiene sentido que cada persona de ese país cuente con un intérprete para entender su documento, y que cada vez que desee leerlo necesite del mismo. Sería mejor que usted cuente con una persona que se encargue de traducir su documento al idioma ruso, obteniendo como resultado una copia en su idioma original y otra en el idioma ruso, así todas las personas en Rusia que no entiendan su idioma, podrán entender el documento enviado. Análogamente funciona un compilador en informática.

De la mano con el surgimiento de los lenguajes de programación, surgían los **paradigmas de programación** que muy a grandes rasgos son *maneras o estilos de programar* empleados. Estos paradigmas son enfoques propuestos para resolver problemas, los cuales tienen una aceptación por una comunidad de programadores. El surgimiento de nuevos paradigmas de programación supone mejoras en la forma de resolver problemas respecto a ya existentes, o bien el planteo de resolución de problemas para situaciones específicas.

Entre los paradigmas de programación se puede encontrar el imperativo, representado por lenguajes como *C, Basic o Pascal*, el lógico representado por *Prolog*, el declarativo representado por lenguajes como *Lisp, SQL, HTML o XML*, el orientado a eventos representado por lenguajes como *Visual Basic, Visual C#, Visual C++, Javascript o Actionscript*, y el orientado a objetos representado por lenguajes como *Smalltalk, Simula 67 o Java*.

El paradigma orientado a objetos fue tomando popularidad y se convirtió en el paradigma dominante a mediados de los años ochenta. Este paradigma se basa en el manejo de datos mediante una entidad llamada objetos la cual a su vez puede relacionarse con otras entidades de este tipo, y hoy en día es el paradigma más usado por la gran mayoría de los programadores dadas sus ventajas frente a la resolución de problemas ya que este paradigma es muy flexible, presenta una manera de resolver problemas muy grandes en forma ordenada y sencilla, y presenta características para la reutilización de código.

En este documento se abordara la enseñanza del lenguaje de programación **JAVA** basándose en los pilares fundamentales del paradigma orientado a objetos para su completa comprensión.

El lenguaje JAVA

por Pablo Ezequiel Jasinski

Fig. 3. James Gosling, creador de Java

JAVA, que inicialmente fue nombrado **Green**, es un lenguaje orientado a objetos multi-propósito creado en 1990 por *James Gosling*, un licenciado en ciencias de la computación, quien era empleado de *Sun Microsystems* hasta ese entonces, empresa que luego sería comprada por *Oracle*.

El lenguaje fue diseñado para que sus programas puedan ejecutarse en cualquier plataforma sin tener que volver a recompilar el código por cada plataforma en la que se quiera ejecutar. El mismo fue creado tomando las mejores características de lenguajes como *C y C++*, brindando un gran soporte para trabajar en red. Por este motivo hoy en día es un lenguaje muy utilizado para **tecnología web**.

Pero en la época en la que se creó este lenguaje internet no existía tal y como se conoce hoy en día. Para ese entonces se estaba desactivando el software de *ARPANET*, e internet estaba limitada a universidades, por lo que la pregunta es: ¿Cómo es que JAVA es sumamente útil para tecnologías webs tal y como se las conocen hoy en día si en 1990 recién se estaban estableciendo las bases de la red de redes? Hay una explicación para esto y la realidad es que lo que *Sun Microsystems* pretendía hacer con *JAVA*,

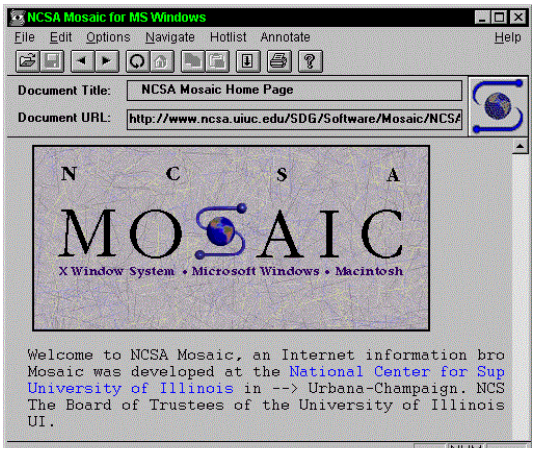


Fig. 4. Mosaic – Primer navegador web gráfico

no tenía nada que ver con internet, sino que su objetivo era crear una interfaz atractiva e intuitiva para *electrónica de consumo*, entendiendo a esta como aquellos equipos electrónicos que nos rodean cotidianamente, como por ejemplo, *televisores, radios, calculadoras, videocámaras*, etc.

Pero al ver que la electrónica de consumo no evolucionó como se esperaba y que por ese motivo, el lenguaje de programación hasta ese entonces llamado Green estuvo congelado por unos años, el equipo desarrollador de este lenguaje decidió juntarse para repuntar el proyecto.

Tras el lanzamiento del reciente navegador web gráfico **Mosaic**, el equipo apostó a que internet evolucionaría como habían pensado que lo haría la tecnología de consumo, y fue así que en 1994 reorientaron el lenguaje a plataformas web y renombraron finalmente el lenguaje, llamándose ahora JAVA.

Características del lenguaje

por Pablo Ezequiel Jasinski

JAVA es un lenguaje utilizado por más de 10 millones de usuarios en la actualidad gracias a las prestaciones y características del mismo. Entre las más importantes se encuentran:

- **Paradigma orientado a objetos:** Es el paradigma por excelencia, adoptado por la mayoría de programadores del mundo en la actualidad. Como ya se ha mencionado anteriormente, este paradigma trata sobre el manejo de datos conjunto a sus operaciones en una entidad llamada objeto, la cual puede relacionarse con otros objetos. El mismo proporciona independencia de código y por lo tanto la posibilidad de reutilizar el mismo, lo cual permite construir grandes proyectos en una manera sencilla y ordenada.

- **Facilidad de uso:** Java combina ser un poderoso lenguaje y tener una gran facilidad de uso ya que el mismo está basado en lenguajes como *C y C++*. Como estos lenguajes son muy populares, y por tanto conocidos por la mayoría de programadores, hacen que al sumergirse en JAVA el código resulte familiar, sumado a las facilidades de uso que proporciona el propio lenguaje.

- **Independencia de plataforma:** Java es un lenguaje que está diseñado para que sus programas se ejecuten bajo una máquina virtual. Gracias a esto es posible que un mismo código compilado pueda ser ejecutado en múltiples plataformas sin tener que recompilar el código para cada plataforma en la cual se quiera ejecutar un programa.

- **Soporte para trabajo en red:** Java ofrece una total transparencia a la hora de trabajar en red. Para eso se abstrajeron las características y detalles específicos en referencia a los aspectos técnicos de redes de comunicaciones. Con el objetivo de simplificar la tarea del programador a la hora de trabajar con redes a bajo nivel, Java proporciona formas básicas de trabajar en red mediante *Sockets y URL*.

- **Seguridad en la ejecución de aplicaciones en sistemas remotos:** Al ser Java un lenguaje multiplataforma, ofrece la posibilidad de programar aplicaciones que pueden ser ejecutadas en navegadores web. Una aplicación programada para ejecutarse en un navegador web podrá por lo tanto ejecutarse en sistemas remotos. A diferencia de Microsoft, Java presenta un sistema de seguridad por capas, el cual se basa en certificados y restringe acciones como el acceso directo al disco rígido y al portapapeles.

La máquina virtual JAVA

por Pablo Ezequiel Jasinski

Luego de explicar conceptos como compilador e intérprete, puede surgir la duda de si Java trabaja con un compilador o bien con un intérprete. Al remitirse a las definiciones anteriores puede pensarse que trabaja con un intérprete, ya que un mismo ejecutable en JAVA puede correr bajo cualquier plataforma, pero por otra parte JAVA es un lenguaje eficiente en cuanto a rapidez y eso contradice a los lenguajes que utilizan intérpretes, ya que estos se caracterizan por no ser muy rápidos. Entonces ¿Cómo hace JAVA para ser multiplataforma y ser eficiente al mismo tiempo?

En realidad la solución a estos temas fue crear una **máquina virtual**, lo que se interpreta como un microprocesador virtual que trabaja con un código binario especial, llamado **bytecode**, el cual es generado por un compilador del lenguaje. El código resultante compilado por el lenguaje es ejecutado por la máquina virtual de JAVA, la cual por un proceso de interpretación o bien mediante un compilador JIT (*Just in Time*), convierte el bytecode en código nativo de la plataforma en la que se vaya a ejecutar, lo que hace a la ejecución mucho más rápida, y que permite entonces que un mismo programa pueda ser ejecutado en diferentes plataformas como *Windows*, *Linux*, *Mac*, etc.

Resumidamente la *máquina virtual de JAVA* (**JVM**) es el puente entre el resultado de la compilación y la plataforma en donde se vaya a ejecutar la aplicación.

Las desventajas son que la máquina virtual de JAVA debe estar instalada en las plataformas sobre las cuales se quieran ejecutar el código, aunque en la actualidad esto no presenta mucho problema ya que la mayoría de las plataformas presentan compatibilidad con ésta, y que el inicialmente se requiere de un tiempo necesario mayor para la compilación del código respecto a otros lenguajes.

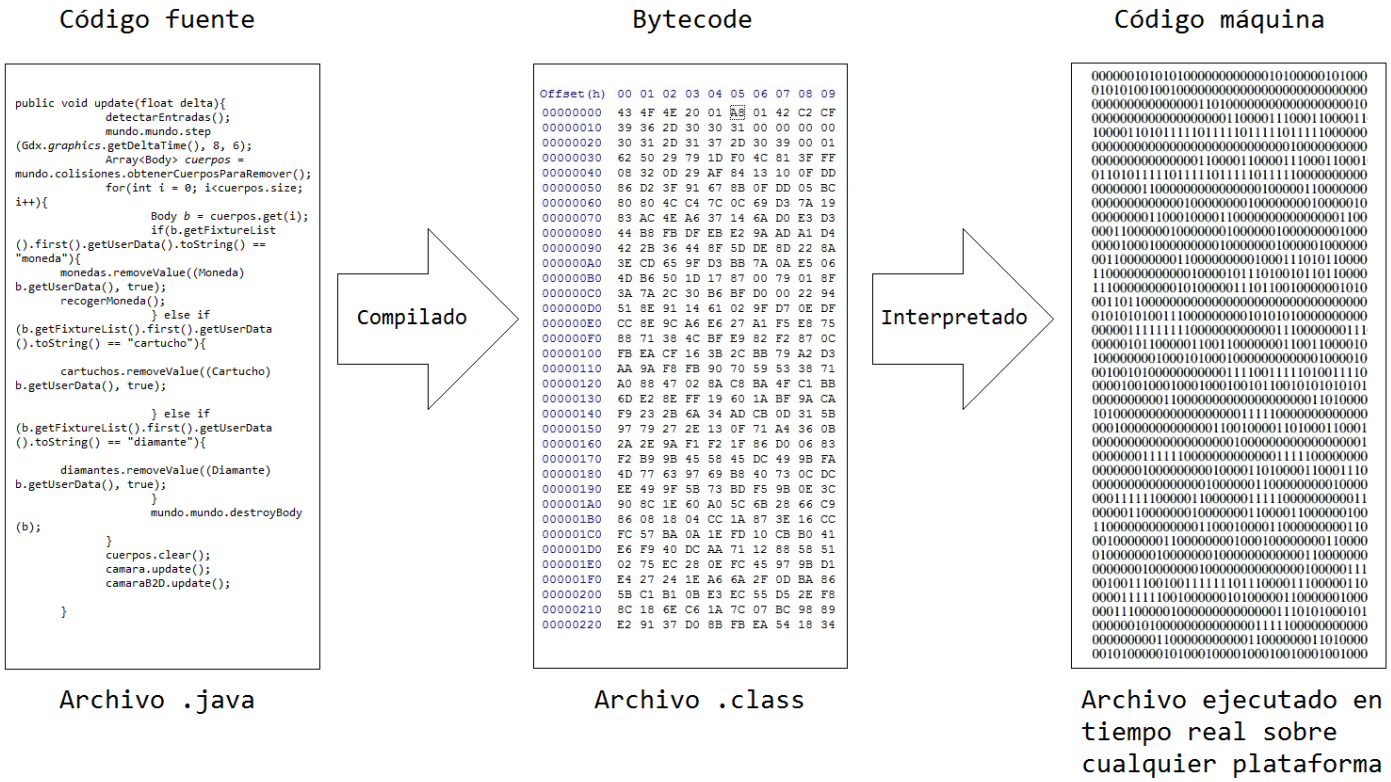


Fig. 4. Funcionamiento de la máquina virtual de Java

Herramientas necesarias para poder programar en JAVA

por Pablo Ezequiel Jasinski

A continuación se detallarán las herramientas necesarias que se deben tener instaladas para poder programar en JAVA.

1) **Máquina virtual de JAVA (JVM):**
   El primer paso es contar con la máquina virtual de JAVA (JVM) instalada, de no ser así debe descargarse desde la página oficial de JAVA. Debe asegurarse de que la versión que descargue sea la última estable y que sea compatible con su sistema operativo.

2) **Configurar la variable de entorno JAVA_HOME:**

   Para sistemas UNIX:

   Para **shells korn** y **bash** ejecutar los siguientes comandos

```
export JAVA_HOME=jdk-install-dir
export PATH=$JAVA_HOME/bin:$PATH
```

   Para **shells bourne** ejecutar los siguientes comandos

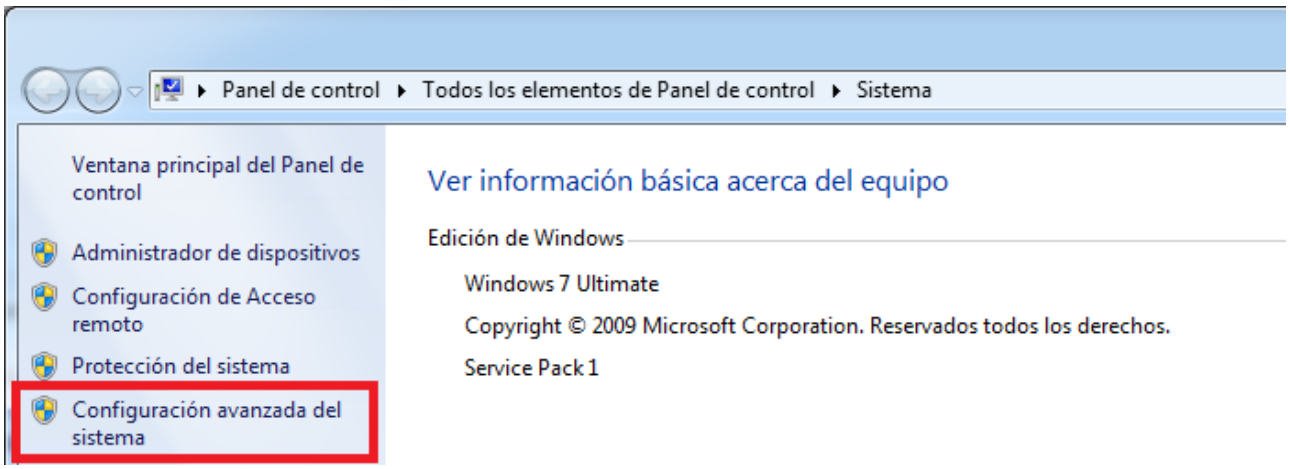
```
JAVA_HOME=jdk-install-dir
export JAVA_HOME
PATH=$JAVA_HOME/bin:$PATH
export PATH
```

   Para **shells C** ejecutar los siguientes comandos

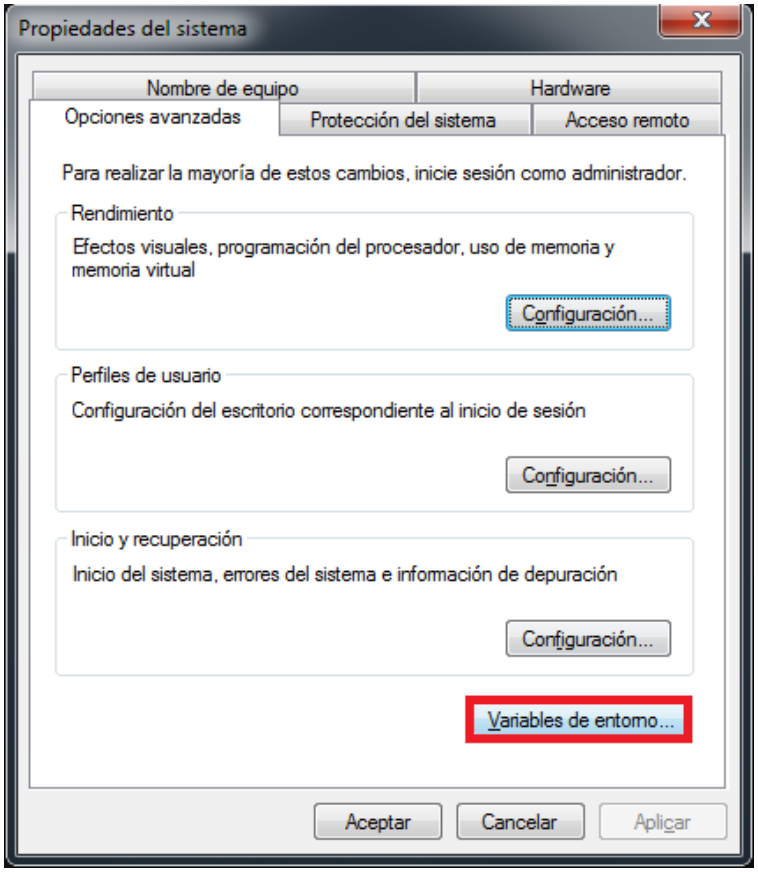
```
setenv JAVA_HOME jdk-install-dir
setenv PATH $JAVA_HOME/bin:$PATH
export PATH=$JAVA_HOME/bin:$PATH
```

   Para sistemas Windows:

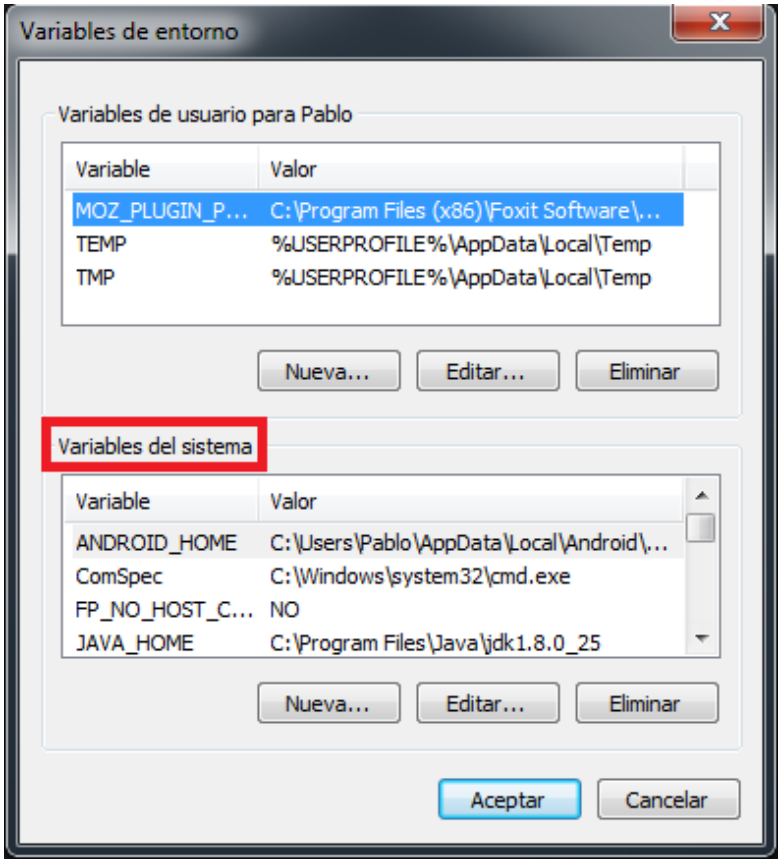
   Ir a "Equipo" hacer clic derecho sobre el mismo y acceder a la opción "Propiedades". Luego se debe hacer clic sobre la opción "Configuración avanzada del sistema"



Una vez hecho esto aparecerá una ventana como la siguiente:



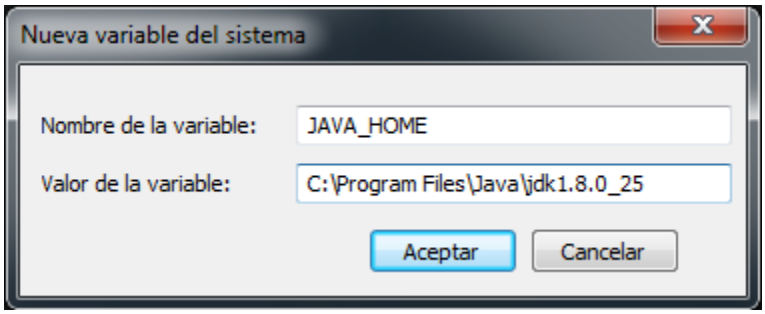
En esta ventana se debe hacer hacer clic sobre el botón "Variables de entorno..."



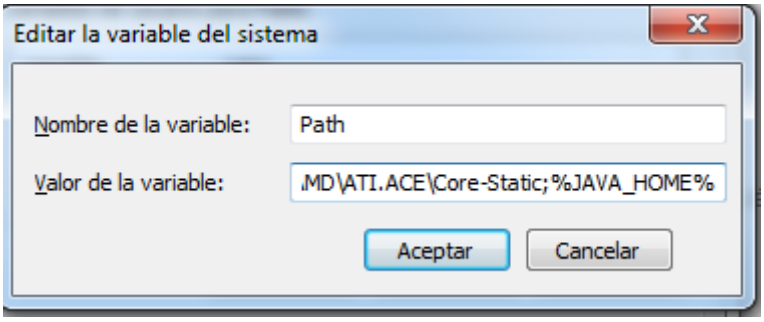
Antes de proseguir se debe verificar que versión de Java se encuentra instalada en el equipo, para eso ir al directorio donde se ha instalado, por ejemplo:

C:\Program Files\Java\jdk1.8.0_25 , y copiar la ruta.

Luego se debe regresar a la ventana anterior prestando atención en "Variables del sistema" y se debe hacer clic sobre el botón "Nueva...", en la nueva ventana visible se debe ingresar el nombre de la variable de entorno y la ruta.



Se debe hacer clic sobre el botón aceptar "Aceptar" y luego hay que buscar en "Variables del sistema" la variable "Patch". Una vez que encontrada se debe hacer clic sobre el botón "Editar..." y en el campo "Valor de variable" al final del texto agregar el carácter ";" para luego escribir %JAVA_HOME%\bin



Al hacer clic sobre botón "Aceptar" se tendrán las **variables de entorno** de **Java** configuradas en el equipo para poder trabajar.

3) Instalar un IDE:

Existen varios IDEs para trabajar en JAVA como BlueJ, Eclipse, InteliJ IDEA, JBuilder, JCreator, Netbeans, etc. Entre los más populares se encuentran Elipse y Netbeans. Este apunte se basará en torno al IDE Eclipse para el desarrollo de conceptos acerca del lenguaje y para la realización de ejemplos.

Para descargar eclipse se debe ir a la página oficial del IDE, asegurarse de escoger la última versión del programa y de que éste sea compatible con el sistema operativo con el que se cuente. Una vez descargado el mismo puede ejecutarse sin antes realizar una instalación del mismo, ya que el programa viene en forma portable.

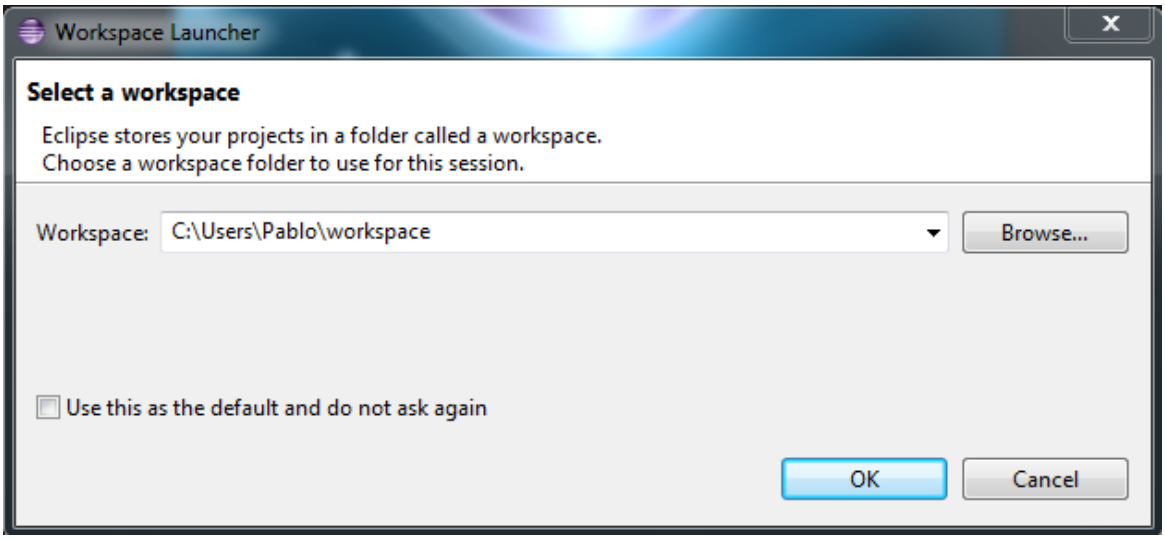
Conociendo el entorno de trabajo

por Pablo Ezequiel Jasinski

Antes de comenzar a programar se desarrollará brevemente los conceptos básicos acerca del IDE **Eclipse**.

El workspace

El **workspace** o espacio de trabajo es el directorio en donde se van a encontrar los proyectos que se cargarán al momento de iniciar *Eclipse*. Al ejecutar el programa aparecerá una pantalla que ofrece la opción de escoger el directorio de trabajo.

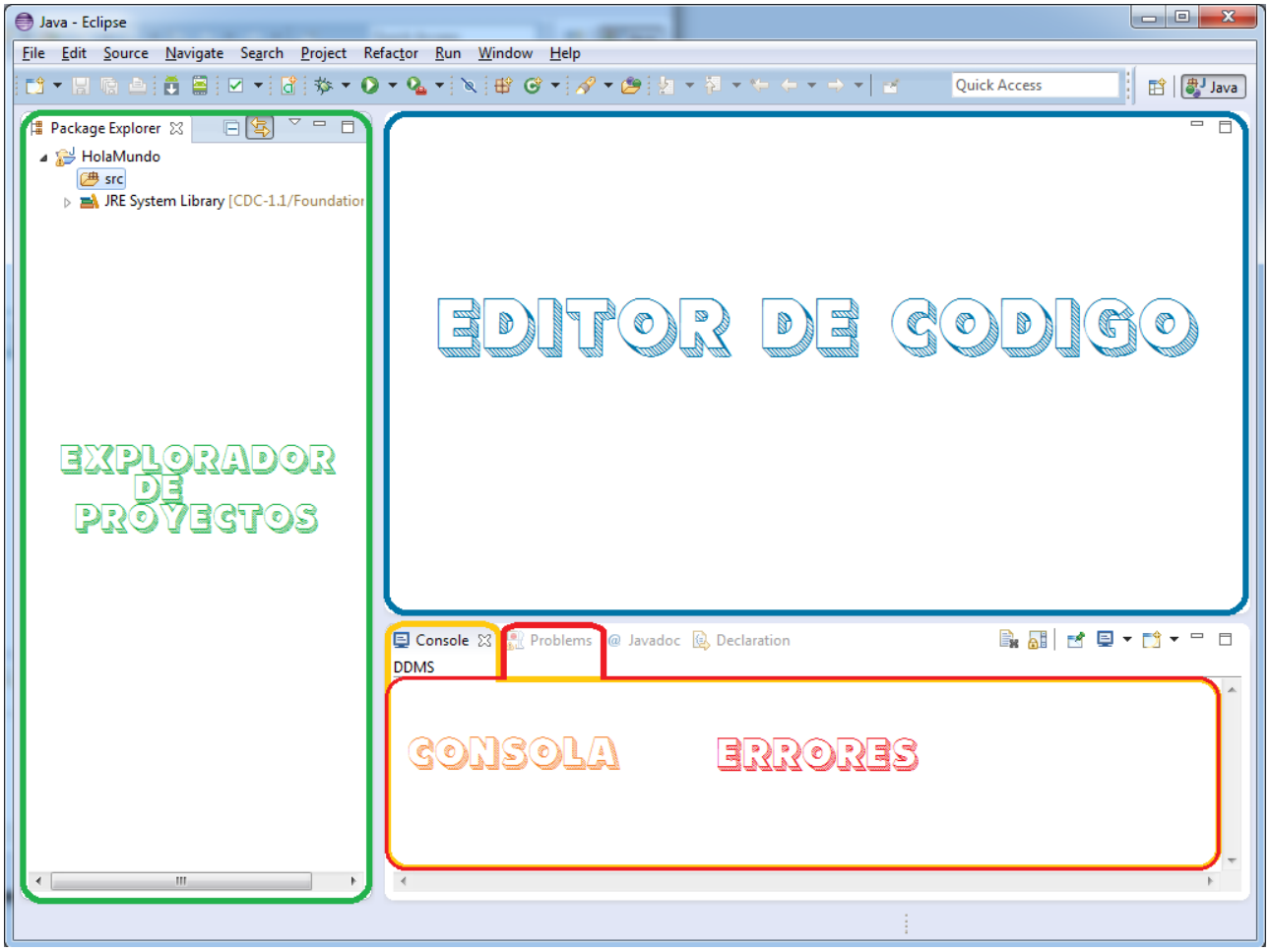


Se elegirá un directorio en donde existan todos los proyectos que se desarrollen o se pueden crear varios directorios para organizar los proyectos de una forma más ordenada, eso queda en cada usuario. Si se desea escoger un directorio por defecto el cual no se cambiará en un futuro se puede marcar la opción "*Use this as the default and do not ask again*".

Secciones básicas del IDE de Eclipse

Al presionar el botón "OK", una vez seleccionado el espacio de trabajo, cargará el IDE de eclipse y al crear un nuevo proyecto se podrá apreciar un conjunto de secciones que se utilizarán a la hora de programar.

A la izquierda se puede observar el **explorador de proyectos**, en el cual se encontrarán todos los archivos del proyecto que se haya creado. Luego está el **editor de código**, en donde aparecerá todo el código de la clase que se vaya a editar o programar, y finalmente se encuentra la **consola**, en donde se puede mostrar o pedir que se ingresen datos, y la sección de **errores** en donde se mostraran todos los errores que existan en el proyecto si así fuera el caso.



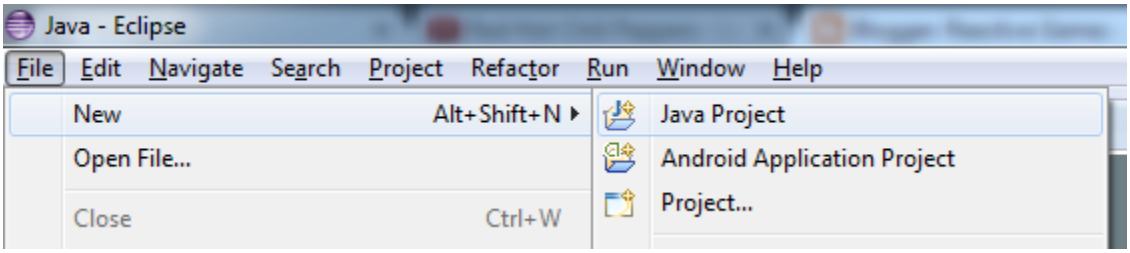
Hay muchos más aspectos para contemplar en el **IDE de Eclipse**, pero por el momento no son relevantes para comenzar a aprender el lenguaje JAVA.

Primer programa en JAVA

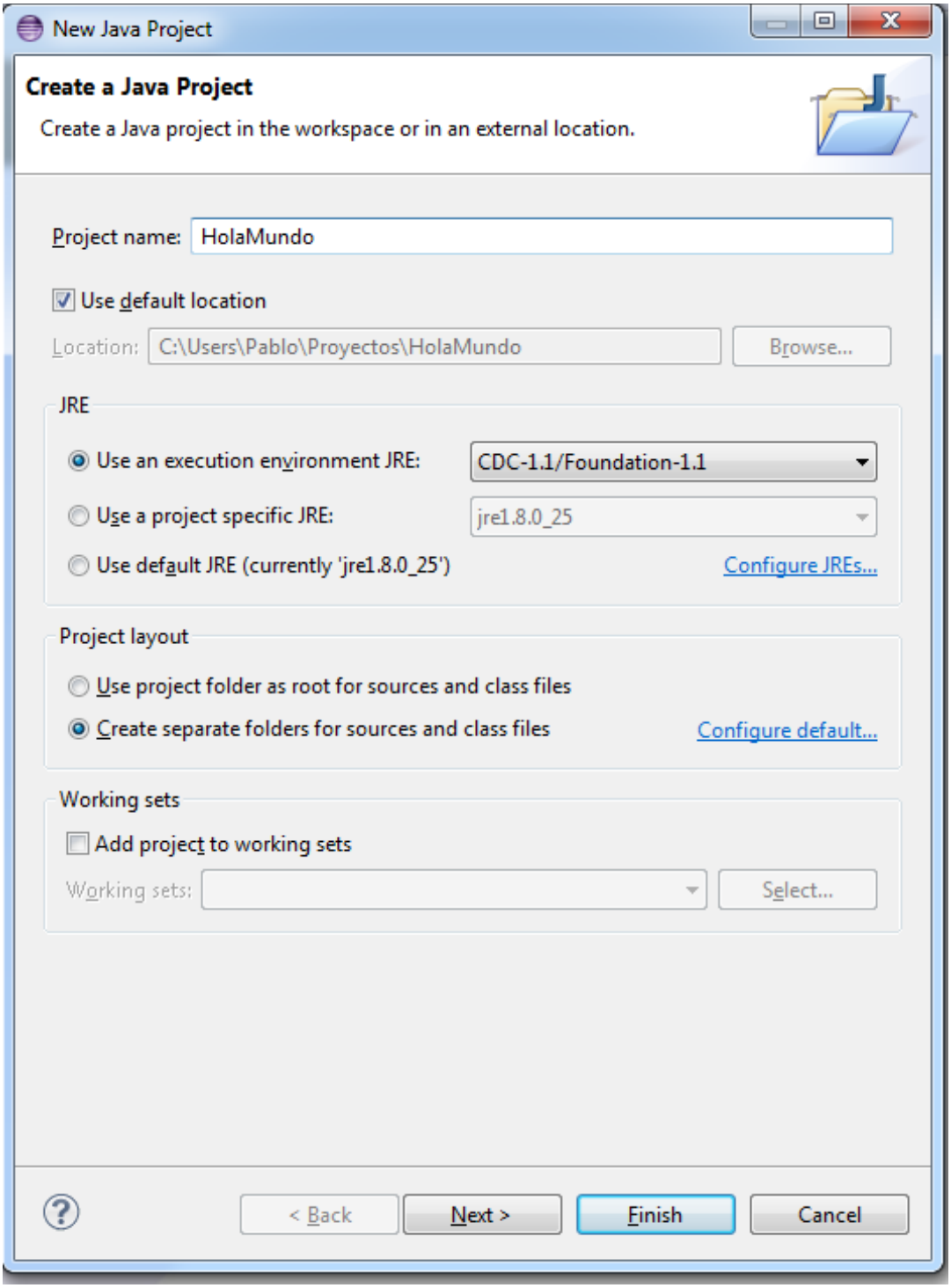
por Pablo Ezequiel Jasinski

Creación de un nuevo proyecto

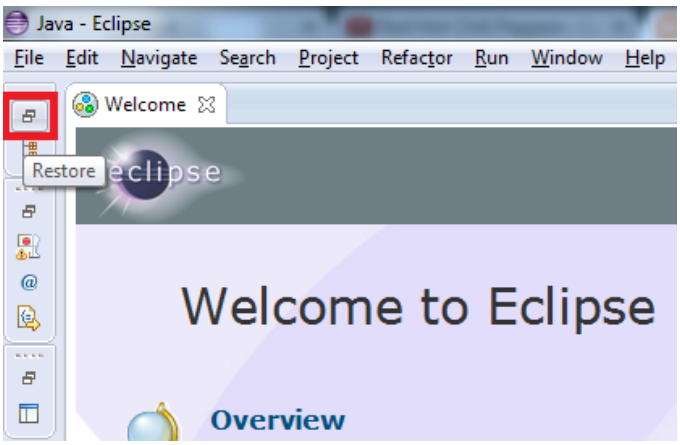
Para comenzar a programar en JAVA se debe generar un nuevo proyecto, la forma de hacerlo es dirigiéndose a la barra de herramientas, luego seleccionar "**File**", "**New**" y finalmente "**Java Project**".



En la siguiente ventana se debe escribir el nombre de proyecto, por ejemplo "*HolaMundo*", y presionar sobre el botón "**Finish**".

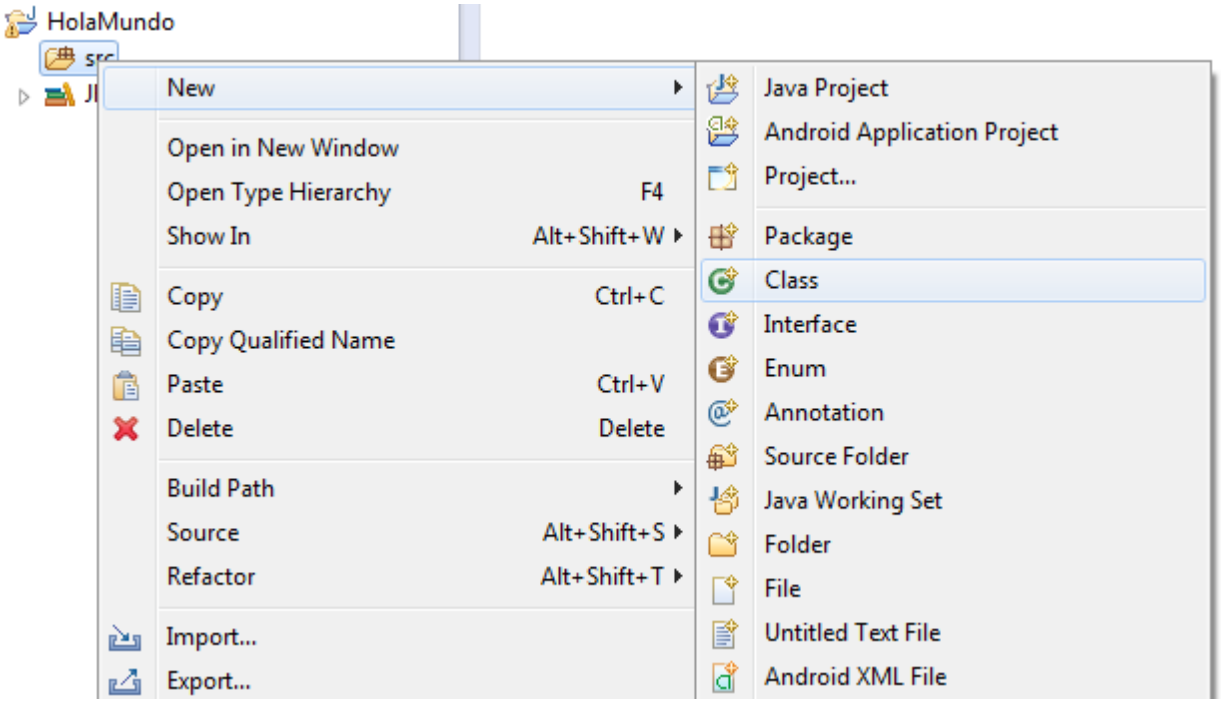


Una vez hecho esto se dispondrá del proyecto creado. En el caso de que solo se muestre la pantalla de bienvenida de eclipse y no se vea el proyecto creado se deberá hacer clic sobre el botón que se muestra a continuación.

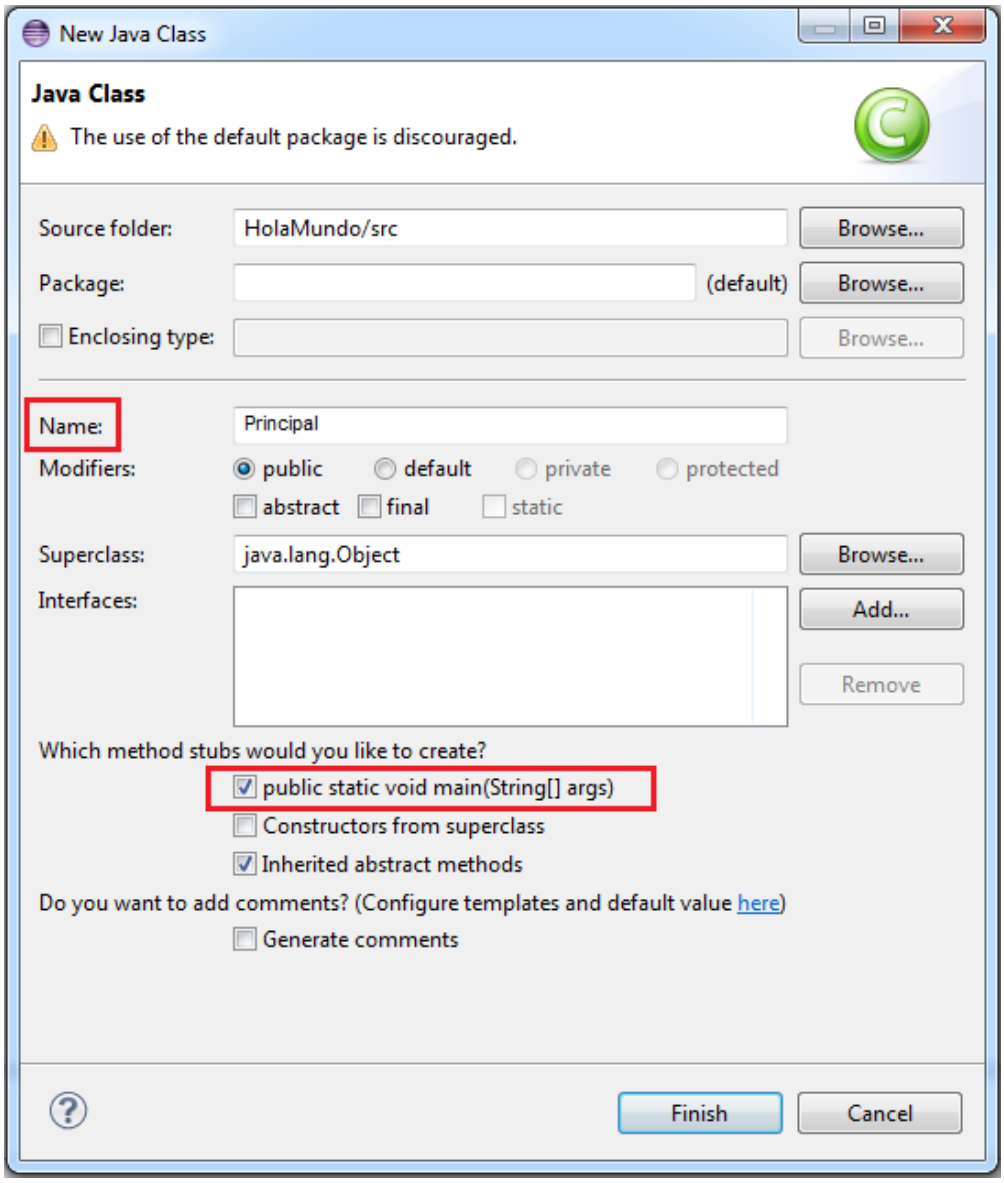


La clase principal de un programa

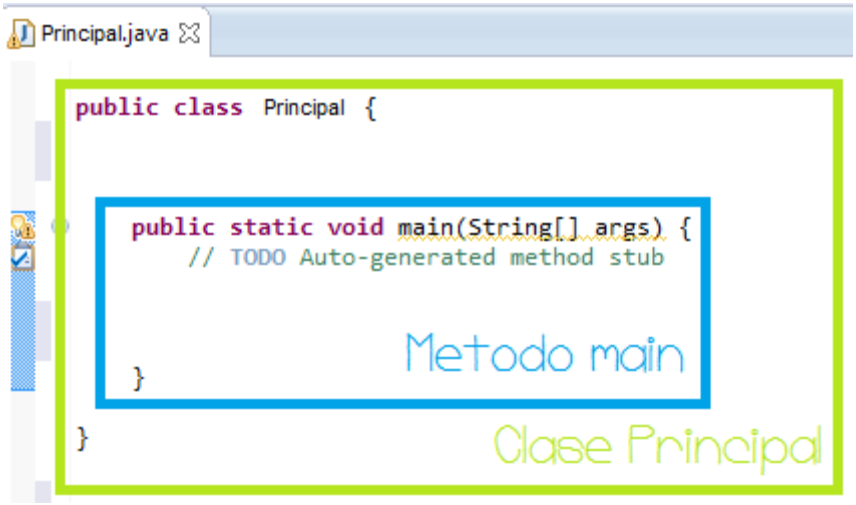
El siguiente paso a realizar es crear la clase principal. Para crear ésta se debe posicionar el ratón sobre la carpeta *"src"*, luego se debe hacer clic con el botón derecho del mismo, escoger la opción "**New**" y seguido de eso se debe seleccionar "**Class**".



En la siguiente ventana que aparece, se debe poner un nombre a la clase, que la identifique como principal, por ejemplo "**Principal**", y luego seleccionar la casilla "*public static void main(String[] args)*", para así convertirla en principal y de esta manera JAVA auto generare el código necesario para comenzar a trabajar con ella sin preocuparse por los aspectos relacionados a la escribir el método principal del programa en forma manual. Seguido de esto se debe hacer clic al botón "**Finish**" y finalmente se creará la clase principal "**Principal**" en la cual se programará.



Al generarse la clase principal, se podrá observar algo de código.



Por un lado se encuentra la **clase** Principal. Si bien todavía no se han abordado los conceptos de la programación orientada a objetos, inicialmente se definirá clase como un espacio en el que se podrán escribir todos los **métodos** y **atributos** que sean necesarios.
Por otro lado se encuentra el método **main** que es el primer método que se va a ejecutar al correr el programa. Aquí irá toda la **lógica del programa**, podrán definirse variables y escribir sentencias de control, bucles, operaciones, etc.

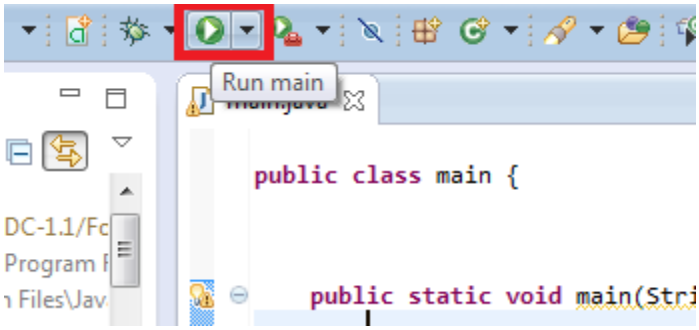
*En programación orientada a objetos, a las **funciones** se les llaman **métodos** (de una clase), y a las **variables** se les llaman **atributos** (de una clase).*

Escribiendo código en JAVA

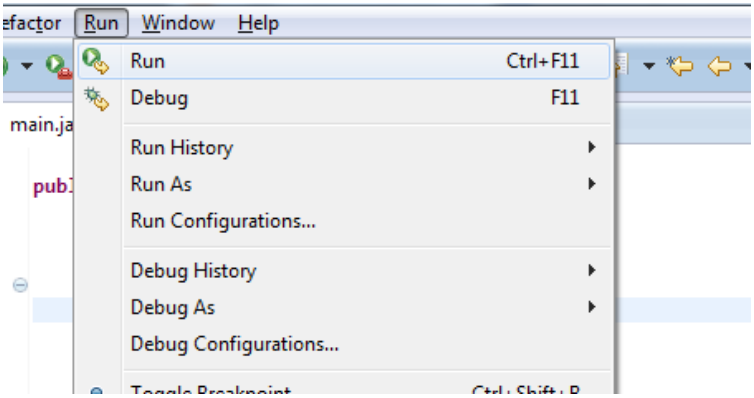
En todo *libro, curso, tutorial o artículo* que enseñe a programar siempre se comienza por el muy conocido "*Hola mundo*" y en este caso no se hará la excepción.
Java dispone de una **consola** en donde se puede mostrar información. Para mostrar información en la consola se debe poner, dentro del método main, el siguiente código.

```
1   // HolaMundo.java
2   // Programa para imprimir texto por pantalla
3
4   public class Principal {
5
6     // El método main comienza la ejecución de la aplicación
7     public static void main(String[] args) {
8
9         System.out.println("Hola mundo");
10
11    }// Fin del método main
12
13  } // Fin de la clase Principal
```

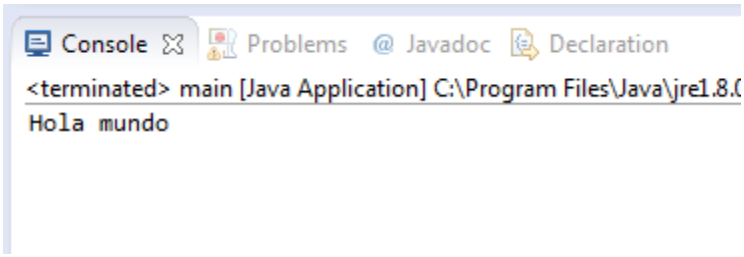
De esta manera se mostrará en la consola el mensaje "*Hola mundo*". Para probar que funciona, se debe hacer clic en el botón "**Run**"



O bien en la barra de herramientas seleccionar el menú "**Run**" y la opción "**Run**"



O bien presionar **Ctrl+F11**, y si no existe ningún error en el proyecto se debería ejecutar el programa y se podrá visualizar en la consola el mensaje que se ha ingresado.



*Java distingue entre mayúsculas y minúsculas, por lo cual no es lo mismo poner "**System.out.println("");**" que "**system.out.println("");**", ya que si se escribe de la segunda manera Java acusará error.*

***Observación:** Para mostrar un mensaje por pantalla mediante el método println siempre debe anteponerle System.out y enlazarla mediante un punto ".", ya que de otra manera no funcionará y se tendrán errores en el programa.*

Tip: Tener que escribir el comando System.out.println(); cada vez que se necesita mostrar algo por consola puede resultar un trabajo muy fastidioso y la repetición de esta operación puede conllevar pérdidas de tiempo innecesarias.

Una solución rápida a esta problemática consiste en utilizar los atajos de eclipse: Un atajo muy útil de Eclipse es el de autocompletar. Para invocar a este atajo se debe presionar la combinación de teclas "Ctrl + Barra espaciadora", por lo tanto para implementar el atajo al escribir el comando correspondiente para mostrar información por consola, se debe escribir la palabra "syso" y seguido la combinación de teclas indicada anteriormente. Como se puede apreciar Eclipse se encargó de escribir el comando entero.