Guía práctica de estudio 01:
Entorno y lenguaje de programación

Elaborado por:
M.C. M. Angélica Nakayama C.
Ing. Jorge A. Solano Gálvez

Autorizado por:
M.C. Alejandro Velázquez Mena

Guía práctica de estudio 01:
Entorno y lenguaje de programación

Objetivo:

Identificar y probar el entorno de ejecución y el lenguaje de programación orientado a objetos a utilizar durante el curso.

Actividades:

- Probar los conceptos básicos del entorno y lenguaje.
- Revisar la instalación y configuración el entorno de ejecución.
- Realizar un programa en el lenguaje de programación usando el entorno de ejecución, utilizando la sintaxis básica (notación, palabras reservadas, comentarios, etc.)

Introducción

Para una mejor comprensión de la **programación orientada a objetos**, se deben practicar los conceptos aprendidos en la teoría implementándolos en algún lenguaje de programación.

Lo primero que se debe hacer para comenzar a programar en dicho lenguaje es conocer sus **fundamentos** y el **entorno de ejecución** así como también las **herramientas** útiles con las que se cuenta para optimizar el desarrollo de programas.

Una vez que se han comprendido las bases del lenguaje, entorno y herramientas, se puede proceder a realizar un programa sencillo y realizar todos los pasos necesarios desde la codificación en el lenguaje hasta la ejecución del mismo en el entorno.

Nota: Elegir un lenguaje para aprender programación orientada a objetos es un tema de discusión entre programadores profesionales, ya que no existe un criterio unánime respecto a qué lenguaje es el ideal para aprender todos los conceptos necesarios. En esta guía se tomará como caso de estudio el lenguaje de programación JAVA, sin embargo, queda a criterio del profesor el uso de éste u otro lenguaje orientado a objetos.

El entorno de ejecución

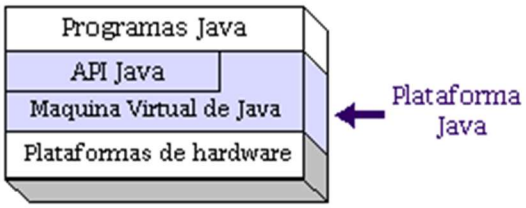
La tecnología Java es un lenguaje de programación y una plataforma comercializada por primera vez en 1995 por Sun Microsystems.

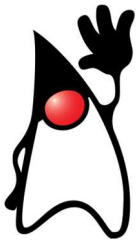
El lenguaje de programación Java fue originalmente desarrollado por James Gosling de Sun Microsystems (la cual fue adquirida por la compañía Oracle) y publicado en 1995 como un componente fundamental de la plataforma Java de Sun Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. El lenguaje de programación Java es un lenguaje de alto nivel, orientado a objetos. El lenguaje es inusual porque los programas Java son tanto compilados como interpretados.

La plataforma Java es una plataforma de software que se ejecuta sobre la base de varias plataformas de hardware. Está compuesto por la JVM (**Java Virtual Machine**) y la Interfaz de programación de aplicaciones Java o API (un amplio conjunto de componentes de software listos para usar, que facilitan el desarrollo y despliegue de aplicaciones).

Además de la API Java, toda implementación completa de la plataforma Java incluye:

- Herramientas de desarrollo para compilar, ejecutar, supervisar, depurar y documentar aplicaciones.
- Mecanismos estándar para desplegar aplicaciones para los usuarios.
- Kits de herramientas de interfaz de usuario que permiten crear interfaces gráficas de usuario (GUIs) sofisticadas.
- Bibliotecas de integración que permiten que los programas accedan a bases de datos y manipulen objetos remotos.



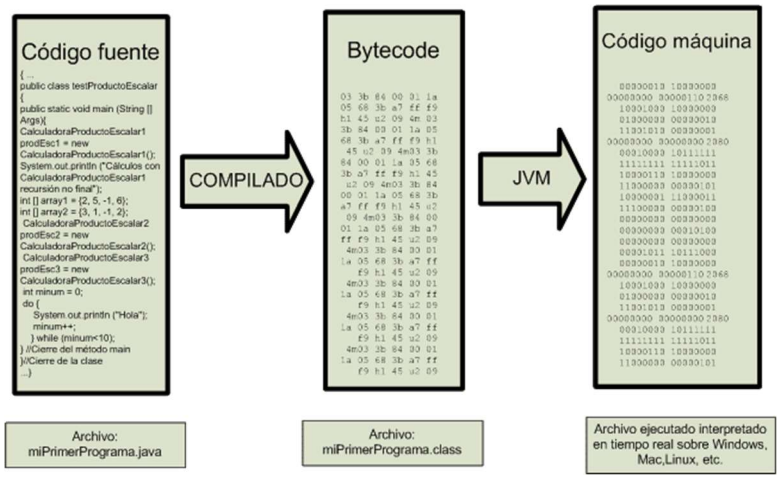


La Máquina Virtual Java (JVM)

Java se hizo independiente del sistema operativo añadiendo un paso intermedio al proceso de compilación (traducir el código escrito en "Lenguaje entendible por humanos" a un código en "Lenguaje Máquina" que entienden las máquinas):

Los programas Java no se ejecutan en nuestra máquina real (en nuestra computadora) sino que Java simula una **"máquina virtual"** con su propio hardware y sistema operativo.

Entonces en Java el proceso es: del código fuente, se pasa a un código intermedio denominado habitualmente *"bytecode"* entendible por la máquina virtual Java. Y es esta máquina virtual simulada, denominada **Java Virtual Machine** o **JVM**, la encargada de interpretar el *bytecode* dando lugar a la ejecución del programa.

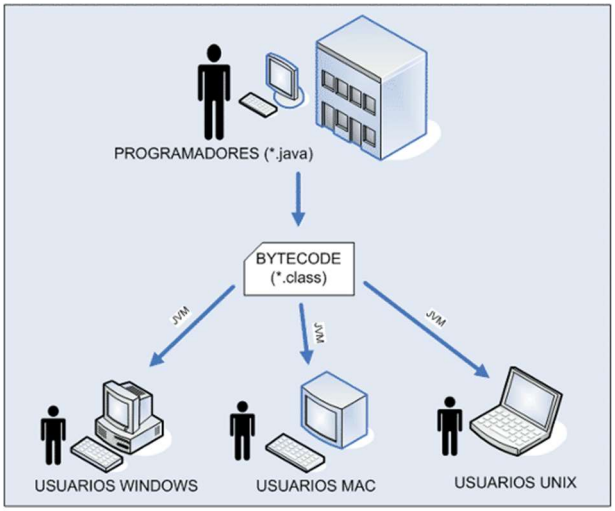


Esto permite que Java pueda ejecutarse en una máquina con sistema operativo Unix, Windows, Linux o cualquier otro, porque en realidad no va a ejecutarse en ninguno de los sistemas operativos, sino en su propia máquina virtual que se instala cuando se instala Java. El precio a pagar o desventaja de este esquema es que siempre que se quiera correr una aplicación Java se debe tener instalado Java con su máquina virtual.

Todo programa en Java está organizado en clases, estas se codifican en archivos de texto con extensión **.java**. Cada archivo de código fuente **.java** puede contener una o varias clases, aunque lo normal es que haya un archivo por clase.

Cuando se compila un **.java** se genera uno o varios archivos **.class** de código binario (*bytecodes*) que son independientes de la arquitectura. Esta independencia supone que los *bytecodes* no pueden ser ejecutados directamente por ningún sistema operativo.

Durante la fase de ejecución es cuando los archivos **.class** se someten a un proceso de interpretación, consistente en traducir los *bytecodes* a código ejecutable por el sistema operativo. Esta operación es realizada por la **JVM**.

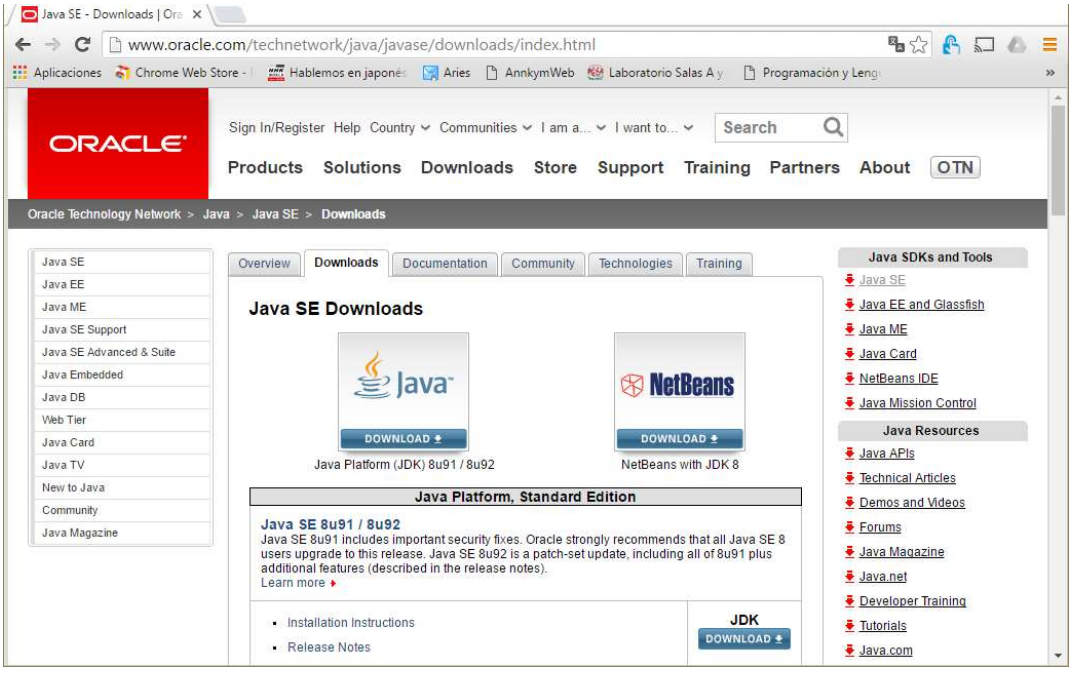


ESQUEMA COMPILADO – INTERPRETADO JAVA

Herramientas de desarrollo (JDK)

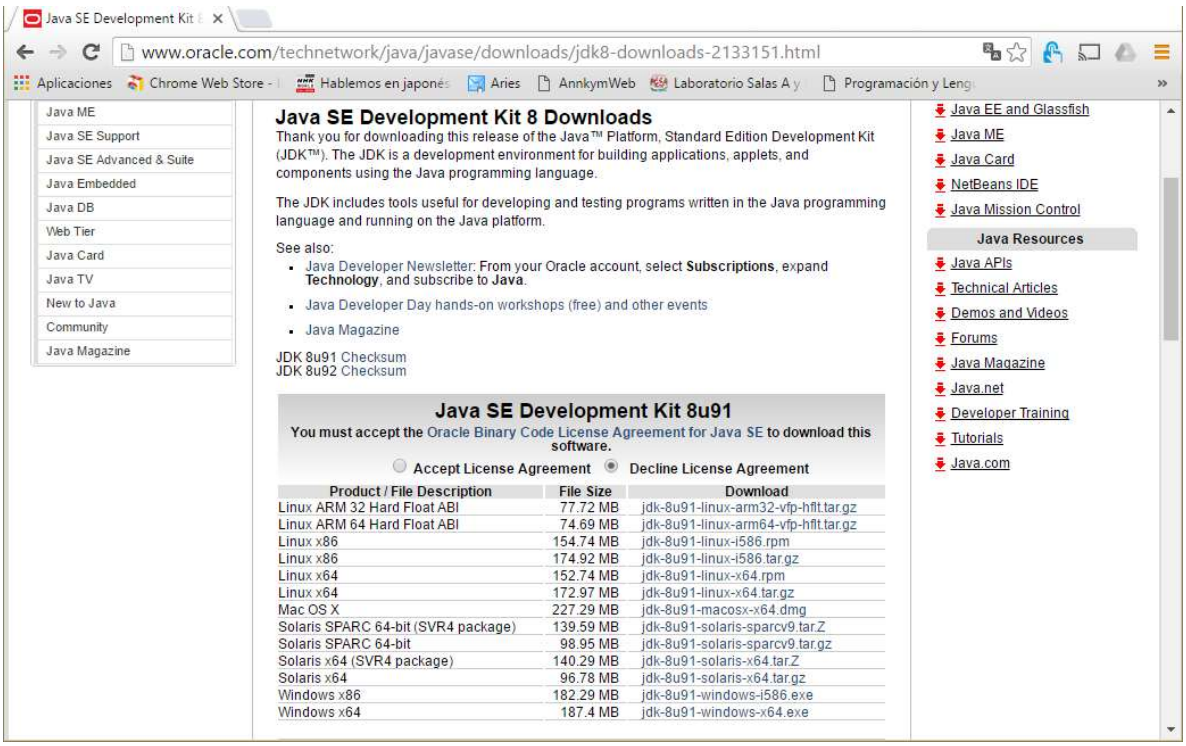
El **Java Development Kit (JDK)** proporciona el conjunto de herramientas básico para el desarrollo de aplicaciones con Java estándar. Se puede obtener de manera gratuita en internet, descargándola desde el sitio de Oracle.

http://www.oracle.com/technetwork/java/javase/downloads/index.html



En este sitio podemos encontrar varios recursos relacionados con java así como otras versiones y herramientas útiles.

Para este curso se utilizará la última versión de Java SE (Standard Edition).



Para obtener la descarga correcta se debe seleccionar el sistema operativo en donde se instalará y aceptar la licencia.

Instrucciones de instalación en distintos sistemas operativos en el sitio:

http://docs.oracle.com/javase/8/docs/technotes/guides/install/install_overview.html

Programa en JAVA

Aunque aún carecemos del conocimiento del lenguaje, podemos iniciar con un sencillo **programa en Java** que imprima un saludo. Este programa nos va a servir para conocer el procedimiento general que se debe seguir para crear, compilar y ejecutar programas Java.

Codificación

Utilizando cualquier **editor de texto** (Block de notas, notepad++, gedit, vi, etc.) procedemos a capturar el siguiente código (teniendo en cuenta que Java es *case sensitive*, es decir, sensible a mayúsculas y minúsculas):

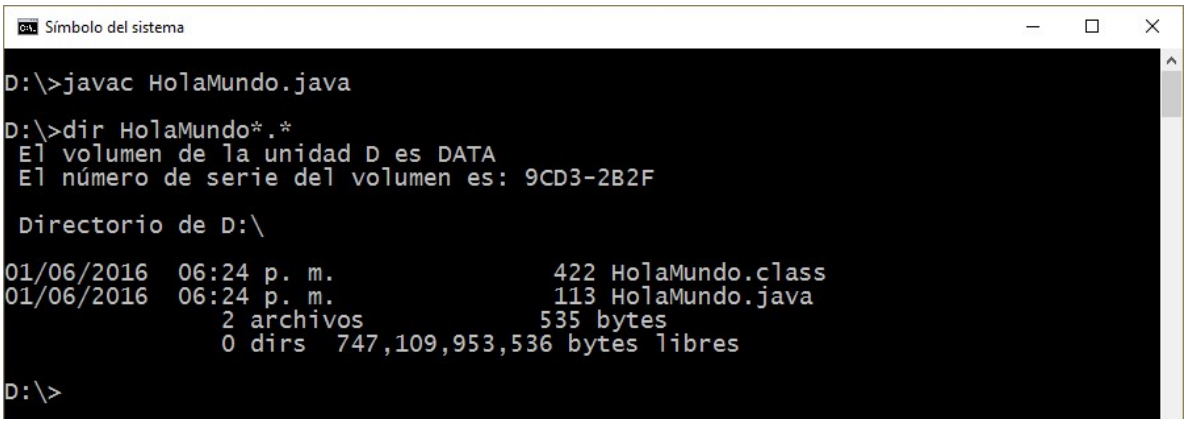
```
public class HolaMundo {
    public static void main(String[] args) {
        System.out.println("Hola Mundo");
    }
}
```

Después procedemos a guardar este programa en un archivo de texto llamado **HolaMundo.java** (el nombre del archivo debe ser el mismo que el de la clase)

Compilación

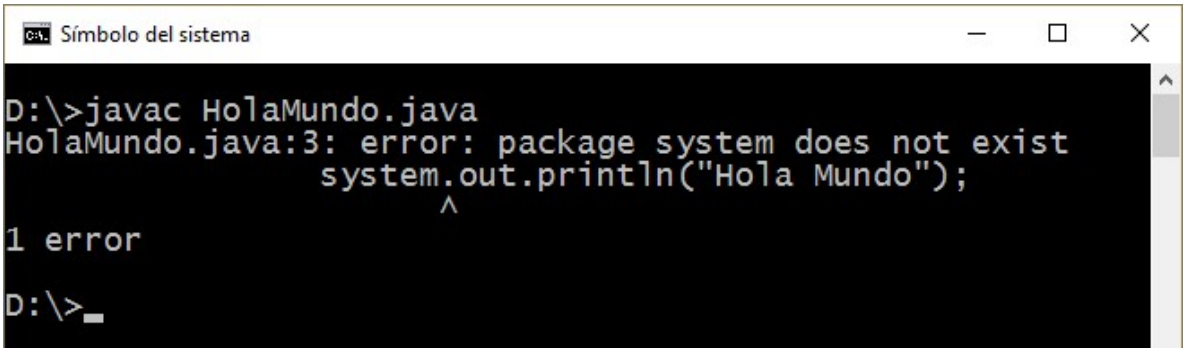
La compilación de un archivo de código fuente **.java** se realiza a través del comando **javac** del **JDK**. Si se ha instalado y configurado correctamente el **JDK**, entonces **javac** podrá ser invocado desde el directorio en el que se encuentre el archivo *HolaMundo.java* creado.

Tras ejecutar este comando, se generará un archivo **HolaMundo.class**.



En caso de que existieran errores sintácticos en el código fuente, el compilador nos habría informado de ello y el archivo **.class** no se generaría.

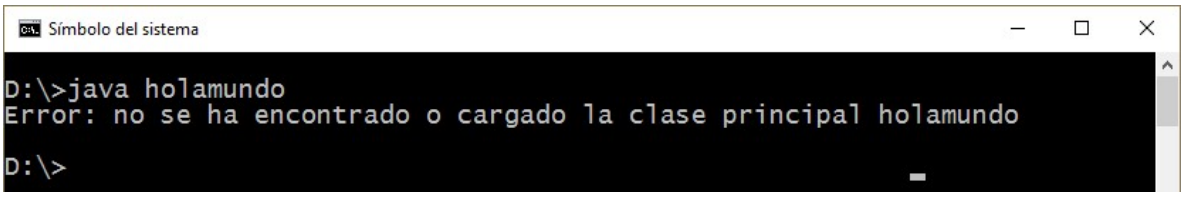
Por ejemplo, si en el código fuente escribimos **system** en vez de **System**, al intentar compilar obtendríamos:



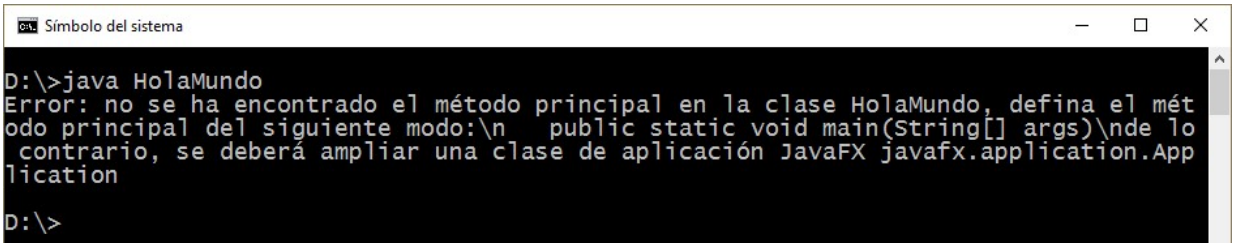
Ejecución

Para ejecutar el programa (una vez compilado correctamente), se utiliza el comando **java** seguido del nombre de la clase que contiene el método *main()*. En nuestro caso será **HolaMundo** ya que es la única clase existente.

La llamada al comando **java** insta a la máquina virtual a buscar en la clase indicada un método llamado *main( )* y procede a su ejecución. En caso que **java** no encuentre la clase ya sea porque el **JDK** esté mal configurado o porque el nombre de la clase sea incorrecto, se producirá un error como el siguiente.



Si el problema no es con la configuración ni el nombre de la clase si no que el formato del método ***main( )*** no es correcto, el programa compilará correctamente pero se producirá un error al ejecutar el programa con java.



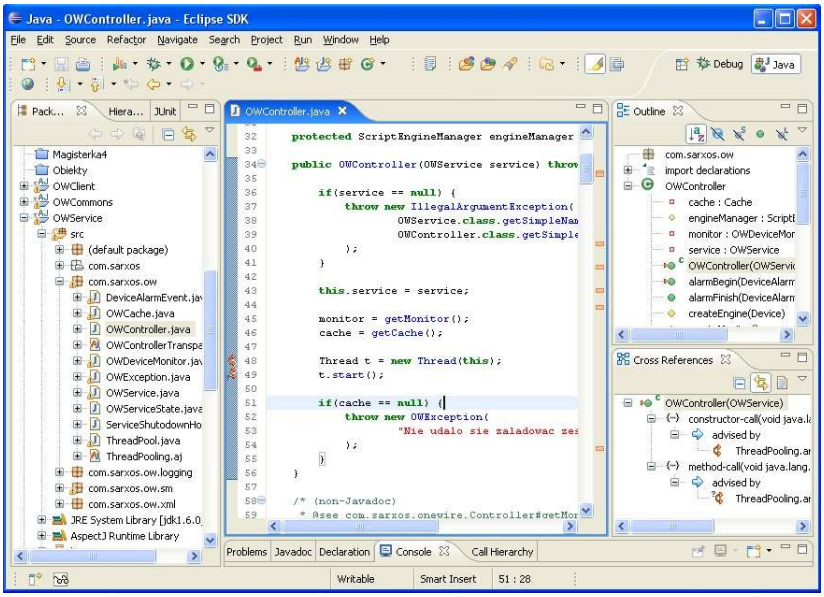
Este procedimiento para compilar y ejecutar la clase *HolaMundo* es el mismo que habrá de seguirse para **cualquier programa en Java**.

Entorno de desarrollo integrado (IDE)

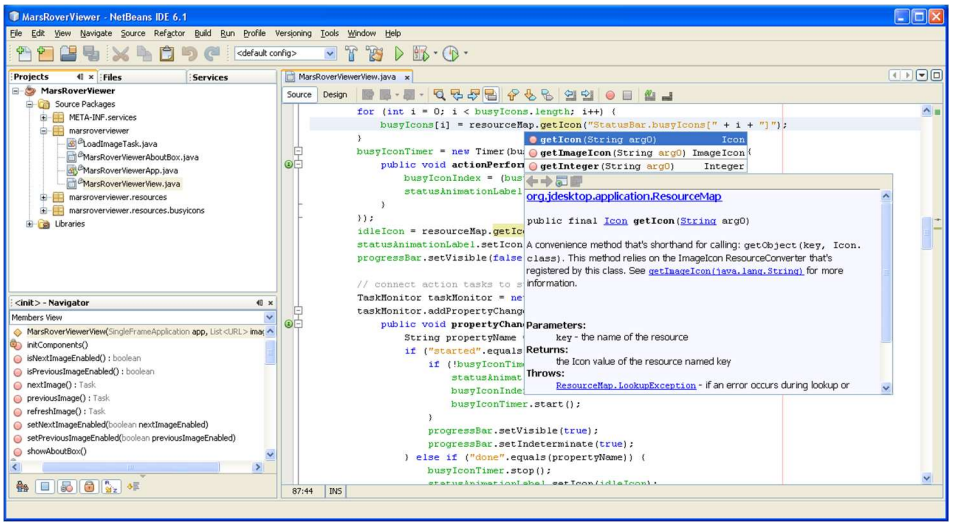
Para desarrollar un producto de software solo es necesario un editor de texto plano para capturar el código fuente y el compilador o el intérprete (según sea el caso) para transformar el lenguaje de alto nivel a lenguaje máquina. Sin embargo, también se puede hacer uso de una aplicación que contenga todas las herramientas en una interfaz, a este tipo de aplicaciones se les conoce como entorno de desarrollo integrado.

Un **entorno de desarrollo integrado** o **IDE** (**Integrated Development Environment**) es una aplicación que facilita el desarrollo de aplicaciones en algún lenguaje de programación. De manera general, un **IDE** es una interfaz gráfica de usuario diseñada para ayudar a los desarrolladores a construir aplicaciones de software proporcionando todas las herramientas necesarias para la codificación, compilación, depuración y ejecución.

Los **IDE** para Java utilizan internamente las herramientas básicas del **JDK** en la realización de estas operaciones, sin embargo, el programador no tendrá que hacer uso de la consola para ejecutar estos comandos, dado que el entorno le ofrecerá una forma alternativa de utilización, basada en menús y barras de herramientas.



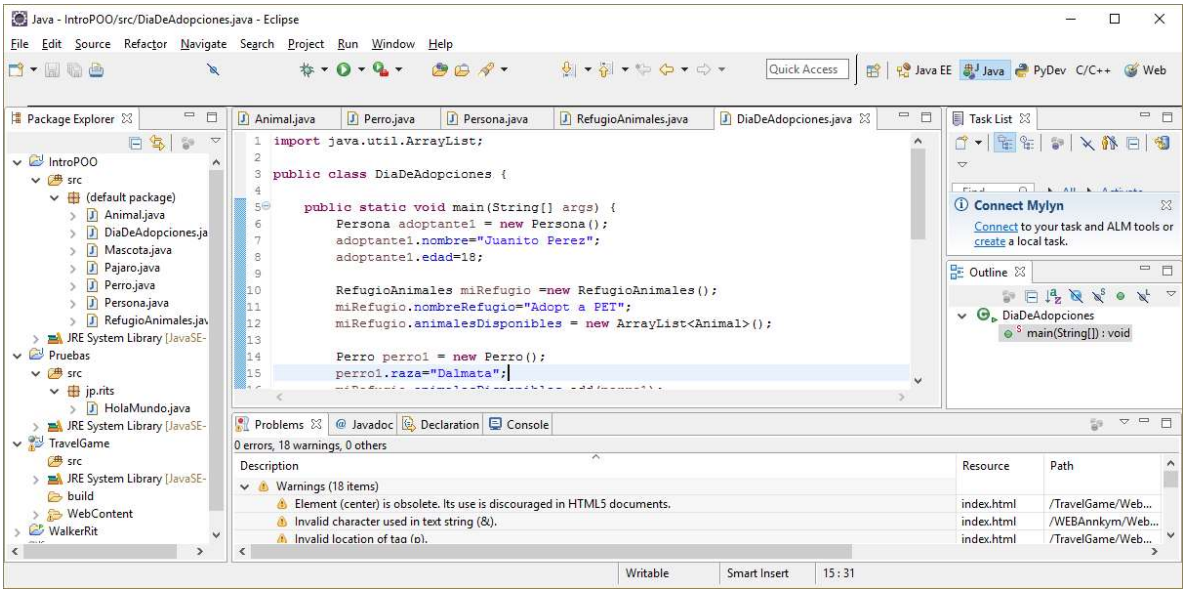
La escritura de código también resulta una tarea sencilla con un **IDE** ya que estos suelen contar con un editor de código que resalta las palabras reservadas del lenguaje para distinguirlas del resto del código. Algunos incluso permiten auto escritura de instrucciones utilizando la técnica *Intellisense*, que consiste en mostrar la lista completa de métodos de un objeto según se escribe la referencia al mismo.



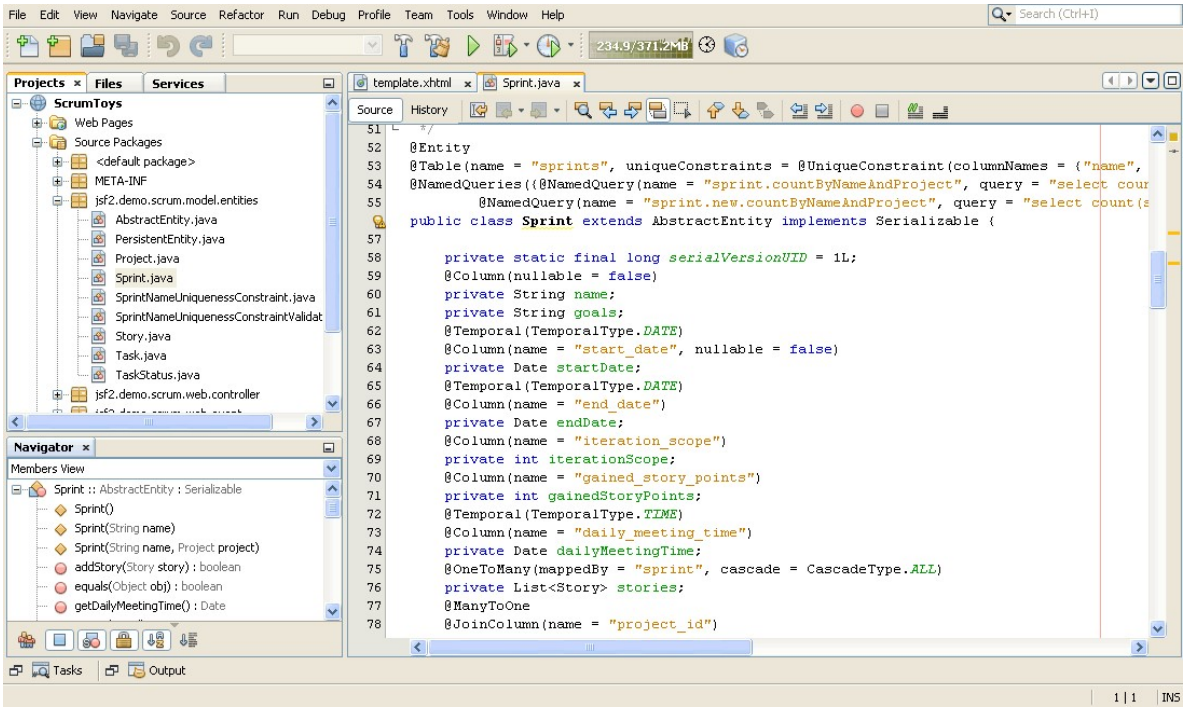
En el mercado existen diversos tipos de **IDE**, cada uno con características propias, empero, una constante es que permiten manejar las etapas para generar un programa dependiendo del tipo de lenguaje utilizado. La mecánica de utilización de estos programas es muy similar, todos ellos se basan en el concepto de proyecto como conjunto de clases que forman una aplicación.

Algunos de los IDE más populares para Java son:

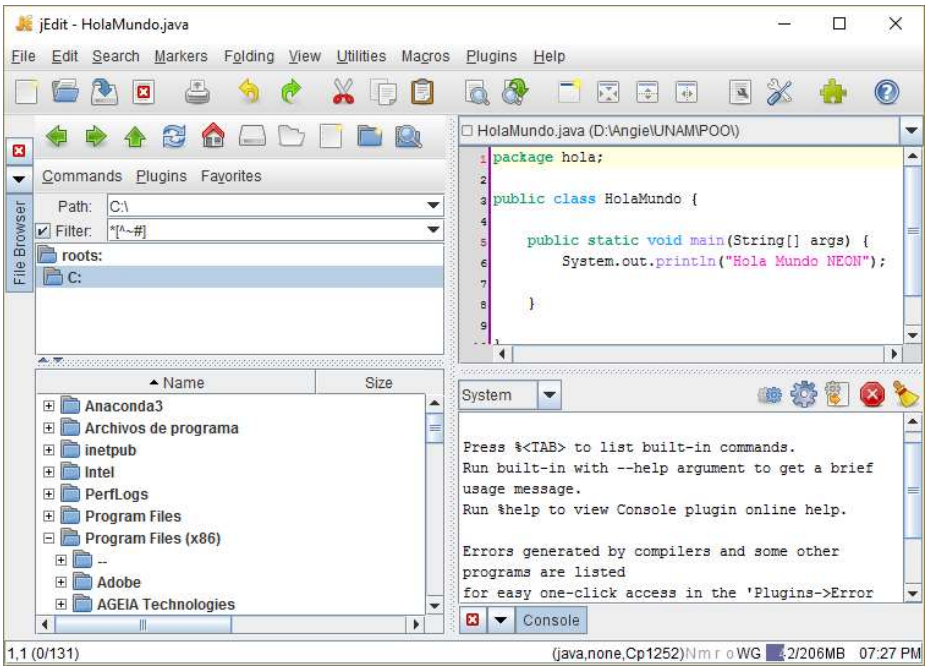
Eclipse



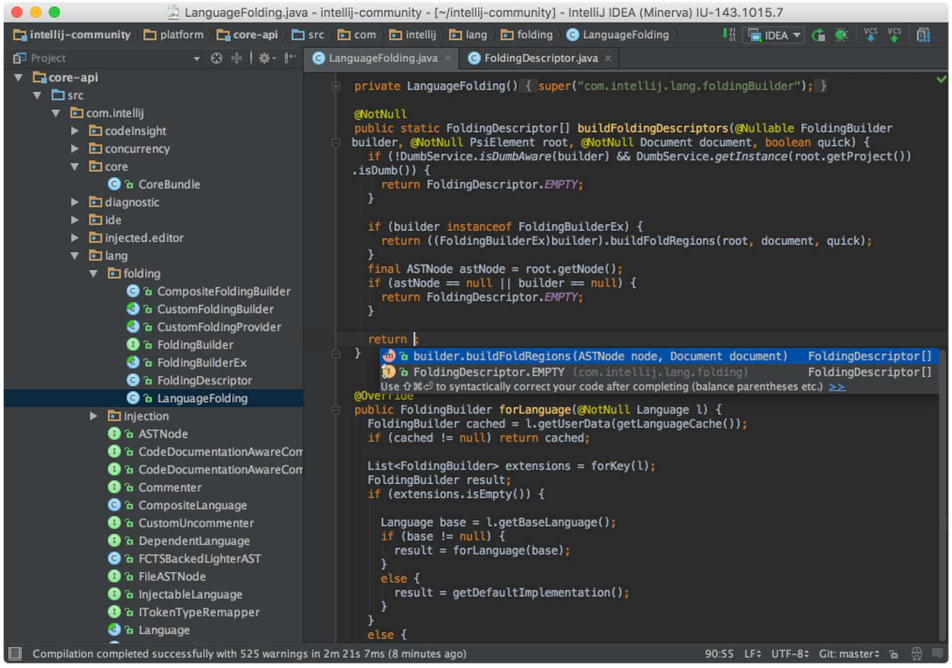
NetBeans



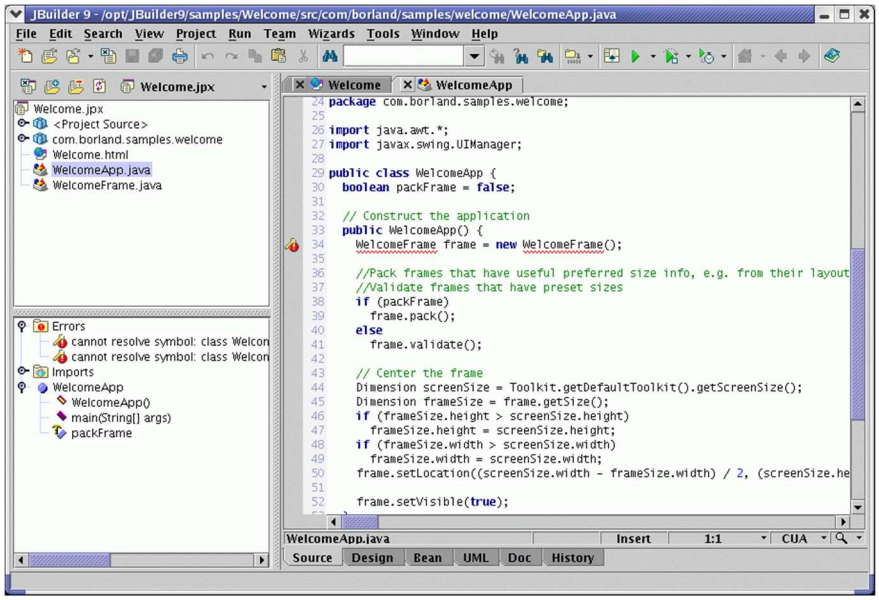
jEdit



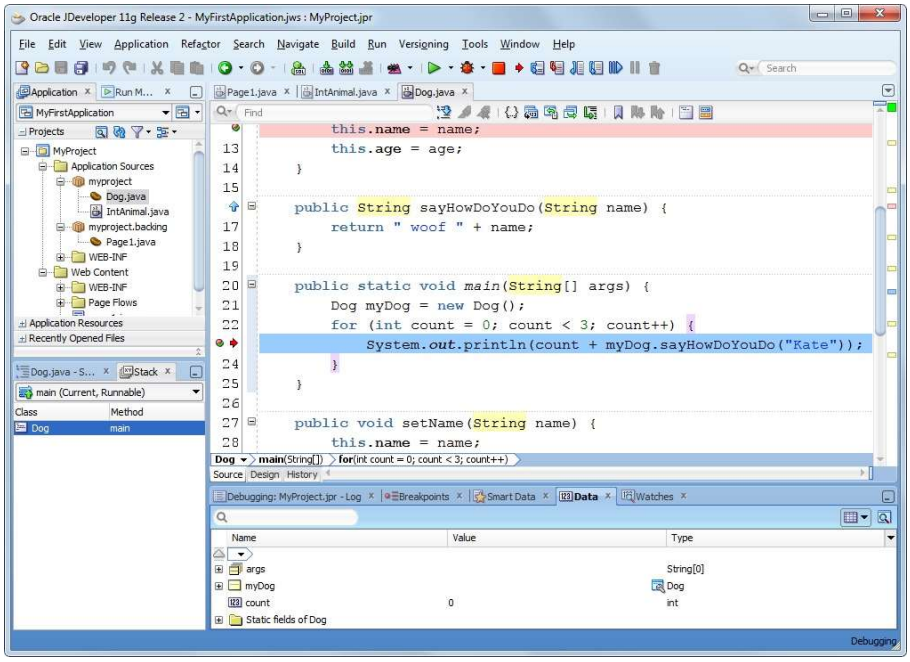
IntelliJ IDEA



JBuilder



JDeveloper



Queda a juicio del programador elegir si utiliza un IDE o no y en caso de hacerlo, también decidir cuál de acuerdo a sus necesidades y gustos.

Estructura general de un programa Java

Una de las principales características de Java es que es un lenguaje totalmente orientado a objetos. Como tal, todo programa Java debe estar escrito en una o varias **clases**, dentro de las cuales se podrá hacer uso del amplio conjunto de paquetes y clases prediseñadas.

Un programa en Java consta de una **clase** principal (que contiene el método **main)** y algunas clases de usuario (las específicas de la aplicación que se está desarrollando) que son utilizadas por el programa o clase principal.

La clase principal debe ser declarada con el modificador de acceso **public** y la palabra reservada **class**, seguida del nombre de la clase iniciando con mayúscula.

Un archivo fuente **(*.java**) puede contener más de una clase, pero sólo una puede ser **public**. El nombre del archivo fuente debe **coincidir** con el de la clase **public** (con la extensión *.java).

Ejemplo:

```
public class MiClase {
    public static void main(String[] args) {
        System.out.println("Esta es mi clase");
    }
}
```

El nombre del archivo tiene que ser **exactamente el mismo** que el de la clase, en este caso debe ser **MiClase.java**. Es importante que coincidan mayúsculas y minúsculas ya que *MiClase.java* y *miclase.java* serían clases diferentes para Java.

Normalmente una aplicación está constituida por varios archivos ***.class**. Cada clase realiza funciones particulares, permitiendo construir las aplicaciones con gran modularidad e independencia entre clases. La aplicación se ejecuta por medio del nombre de la clase que contiene el método **main**( ) (sin la extensión *.class).

Para el ejemplo:

El método main

En la clase principal debe existir una **función o método** estático llamado **main** cuyo formato debe ser:

public static void main(String[ ] args)

El método **main** debe cumplir las siguientes características:

- Debe ser un método púbico (**public**).
- Debe ser un método estático (**static**).
- No puede devolver ningún resultado (tipo de devolución **void**).
- Debe declarar un arreglo de cadenas de caracteres en la lista de parámetros o un número variable de argumentos).

El método **main** es el punto de arranque de un programa Java, cuando se invoca al comando **java** desde la línea de comandos, la **JVM** busca en la clase indicada un método estático llamado **main** con el formato indicado.

Dentro del código de **main** pueden crearse objetos de otras clases e invocar sus métodos, en general, se puede incluir cualquier tipo de lógica que respete la sintaxis y estructura de java.

Sintaxis básica

Una de las primeras cosas que hay que tener en cuenta es que Java es un lenguaje **sensitivo** a mayúsculas/minúsculas. El compilador Java hace distinción entre mayúsculas y minúsculas, esta distinción no solo se aplica a palabras reservadas del lenguaje sino también a nombres de variables y métodos.

La sintaxis de Java es muy parecida a la de C y C++, por ejemplo, las sentencias finalizan con **;** , los bloques de instrucciones se delimitan con llaves **{** y **}**, etc. A continuación se explican los puntos más relevantes de la sintaxis de Java.

Comentarios

Los comentarios son muy útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. Además permite que cualquier persona distinta al programador original pueda comprender el código escrito de una forma más rápida. Se recomienda acostumbrarse a comentar el código desarrollado. De esta forma se simplifica también la tarea de estudio y revisión posteriores.

En Java existen tres tipos de comentarios:

• Comentarios de una sola línea.

// Esta es una línea comentada.

• Comentarios de bloques.

/ Aquí empieza el bloque comentado*

*y aquí acaba */*

• Comentarios de documentación (**JavaDoc**).

*/** Los comentarios de documentación se realizan de este modo */*

Identificadores

En Java los identificadores comienzan por una letra del alfabeto inglés, un subrayado _ o el símbolo de dólar **$**, los siguientes caracteres del identificador pueden ser letras o dígitos. No se debe nunca iniciar con un digito. No hay un límite en lo concerniente al número de caracteres que pueden tener los identificadores.

Las reglas del lenguaje respecto a los nombres de variables son muy amplias y permiten mucha libertad al programador, pero es habitual seguir ciertas normas que facilitan la lectura y el mantenimiento de los programas.

En Java es habitual utilizar nombres con minúsculas, con las excepciones que se indican en los puntos siguientes.

- Los nombres de **clases** e **interfaces** comienzan siempre por mayúscula. Ejemplos: *Geometria*, *Rectangulo*, *Dibujable*, *Graphics*, *Iterator*.
- Cuando un nombre consta de varias palabras es habitual poner una a continuación de otra, poniendo con mayúscula la primera letra de la palabra que sigue (**CammelCase**). Ejemplos: *elMayor( )*, *VentanaCerrable*, *RectanguloGrafico*, *addWindowListener( )*.
- Los nombres de **objetos**, **métodos, variables miembro** y **variables locales** de los métodos, comienzan siempre por minúscula. Ejemplos: *main( )*, *dibujar( )*, *numRectangulos*, *x*, *y*, *r*.
- Los nombres de las **variables finales**, es decir de las **constantes**, se definen siempre con mayúsculas. Ejemplo: *PI*

Sin embargo, éstas no son las únicas normas para la nomenclatura en Java, también existen las **convenciones de código**, las cuales son importantes para los programadores dado que mejoran la lectura del programa, permitiendo entender código nuevo mucho más rápidamente y más a fondo.

Para que funcionen las convenciones, todos los programadores deben seguir la convención, por esta razón, es muy importante generar el hábito de aplicar convenciones y estándares de programación ya que de esta manera no solo se enfoca en las funcionalidades sino que también se aporta a la calidad de los desarrollos.

Las convenciones de código o **Java Code Conventions**, pueden ser consultadas en el siguiente link.

http://www.oracle.com/technetwork/java/codeconvtoc-136057.html

Palabras reservadas

Las siguientes son palabras reservadas utilizadas por Java y no pueden ser usadas como identificadores.

abstract	boolean	break	byte	case	catch
char	class	const	continue	default	do
double	else	extends	final	finally	float
for	goto	if	implements	import	instanceof
int	interface	long	native	new	package
private	protected	public	return	short	static
strictfp	super	switch	synchronized	this	throw
throws	transient	try	void	volatile	while
assert	enum				

También existen las literales reservadas *null*, *true* y *false*, las cuales tampoco pueden ser usadas como identificadores.

Bibliografía

Martín, Antonio
Programador Certificado Java 2.
Segunda Edición.
México
Alfaomega Grupo Editor, 2008

Sierra Katy, Bates Bert
SCJP Sun Certified Programmer for Java 6 Study Guide
Mc Graw Hill

Dean John, Dean Raymond.
Introducción a la programación con Java
Primera Edición.
México
Mc Graw Hill, 2009