



# Java y NetBeans

# 0

Programación Java – NetBeans v. 7.0

RPC

## Contenido

0.1	Introducción .....	1
0.2	Historia de Java .....	1
0.2.1	Evolución de Java.....	3
0.3	Plataformas soportadas .....	4
0.4	Características del Lenguaje.....	5
0.5	Entornos de funcionamiento .....	8
0.6	Recursos.....	10
0.7	Sintaxis.....	11
0.8	Entornos de Desarrollo (Interface Developement Environment, IDE) .....	14
0.9	Historia de NetBeans .....	14
	Evolución de NetBeans .....	15
	Versiones.....	15
0.10	Instalación de NetBeans.....	16
0.11	Cuestionario .....	17
	Bibliografía.....	17

# 0 Java y NetBeans

“Write once, run anywhere<sup>1</sup>” James Gosling

## 0.1 Introducción

Java es un lenguaje de programación orientado a objetos, desarrollado por *Sun Microsystems* a principios de la década de 1990. Este lenguaje toma la estructura de sintaxis (escritura de código fuente) de C y C++ incorporando un modelo de objetos más simples y eliminando las herramientas de bajo nivel (que suele inducir muchos errores como la manipulación de punteros en C y C++). El surgimiento de Java es propiciado porque en la década de 1980 aparecieron varios lenguajes orientados a objetos y también se realizaron versiones orientadas a objetos (o semi-orientados a objetos) de lenguajes clásicos. Una de las más famosas fue el lenguaje orientado a objetos creado a partir del C tradicional; se le llamó C++ indicando con esa simbología que era un incremento del lenguaje C (en el lenguaje C, como en Java, los símbolos ++ significan incrementar) (Sánchez, 2004).

La Programación Orientada a Objetos (POO) permite desarrollar aplicaciones de forma más parecida al pensamiento humano, de hecho simplifica el problema de programación (principalmente en el desarrollo de aplicaciones de código largo) dividiéndolo en objetos, permitiendo centrarse en cada objeto, de esa manera se elimina la complejidad de escribir código largo.

En este capítulo se abordará la historia de Java, su evolución, su filosofía y entornos de desarrollo así como los distintos IDEs (interfaz de entorno de desarrollo) desde consola hasta las aplicaciones móviles haciendo referencia al IDE NetBeans pues este IDE constituye el enfoque sobre el cual se fundamenta este manual.

## 0.2 Historia de Java

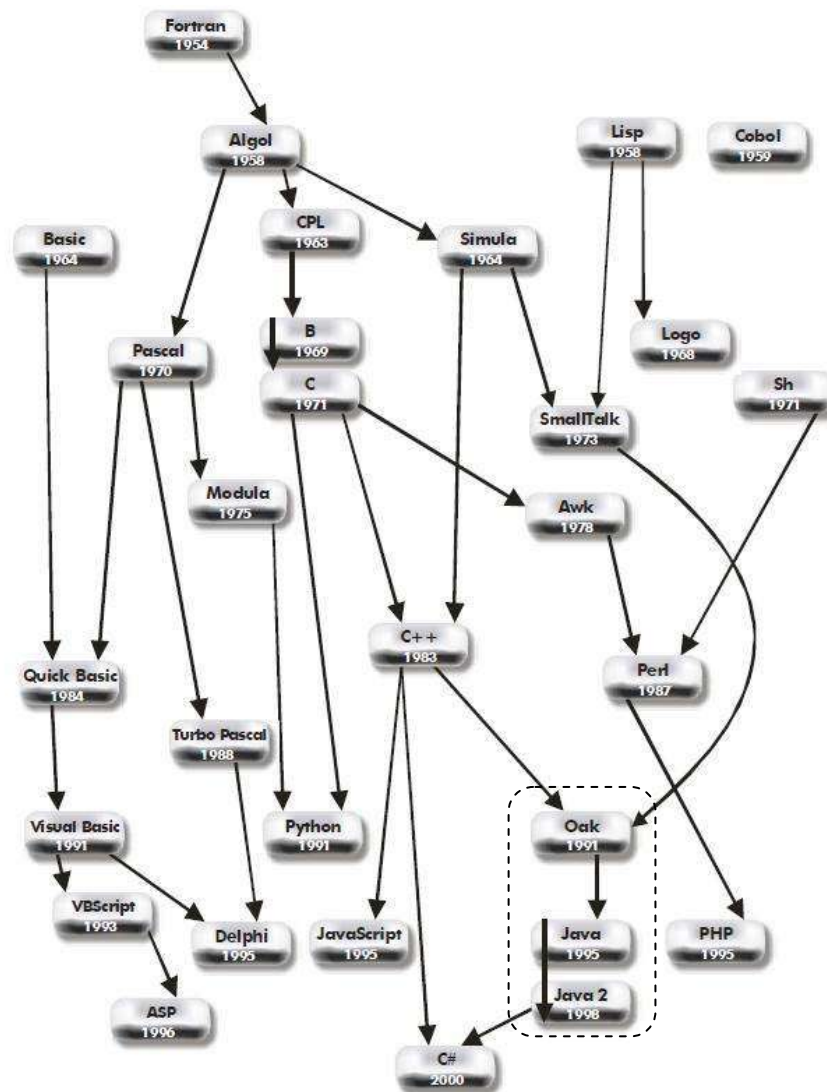
Java se creó como una herramienta de programación para ser usada en un proyecto de set-top-box en una pequeña operación denominada “*The Green Project*” en Sun Microsystems en el año 1991. El propósito de Sun Microsystems era introducirse en el mercado de la electrónica de consumo y desarrollar programas para pequeños dispositivos electrónicos (Coronel, 2010). El equipo “*Green Team*”, estaba compuesto por trece personas y dirigido por James Gosling, trabajaron durante 18 meses en su desarrollo. Los objetivos de Gosling eran implementar una máquina virtual y un lenguaje con una estructura y sintaxis similar a C++.

El lenguaje se denominó inicialmente ‘Oak’ (por un roble que había fuera de la oficina de Gosling), luego pasó a denominarse *Green* tras descubrir que *Oak* era ya una marca comercial registrada para adaptadores de tarjetas gráficas y finalmente se renombró a Java<sup>2</sup>.

---

<sup>1</sup> Filosofía de Java: “Escríbelo una vez, ejecútalo en cualquier lado”.

<sup>2</sup> El término Java fue acuñado en una cafetería frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus creadores: James Gosling, Arthur Van Hoff, y Andy Bechtolsheim. Otros abogan por el siguiente acrónimo, *Just Another Vague Acronym* (“sólo otro acrónimo ambiguo más”). La hipótesis que más fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana, de ahí que el icono de java sea una taza de café caliente.



**Figura 1 Evolución de Lenguajes de Programación**

Entre junio y julio de 1994, tras la poca aceptación de la plataforma en la industria electrónica, el equipo conformado por John Gaga, James Gosling, Joy Naughton, Wayne Rosing y Eric Schmidt, reorientó la plataforma hacia la Web (en tan solo tres días). Sintieron que la llegada del navegador web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo (tal como el pensaban era la televisión por cable). Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava. En 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. *Java 1.0a* pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web donde java soportaría los navegadores Netscape. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico.

La promesa inicial de Gosling era "Write Once, Run Anywhere" (Escríbelo una vez, ejecútalo en cualquier lugar), proporcionando un lenguaje independiente de la plataforma y un entorno de ejecución (la Java Virtual Machine) ligero y gratuito para las plataformas más populares de forma que los binarios (bytecode) de las aplicaciones Java pudiesen ejecutarse en cualquier plataforma.

El entorno de ejecución era relativamente seguro y los principales navegadores web pronto incorporaron la posibilidad de ejecutar *applets* Java incrustadas en las páginas web.

### 0.2.1 Evolución de Java

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar. Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (*Java Community Process*), que usa *Java Specification Requests* (JSRs) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en *Java Language Specification* (JLS). Los cambios en los JLS son gestionados en JSR 901. A continuación se muestra la evolución de java:

**JDK 1.0** (23 de enero de 1996); Primer lanzamiento.

**JDK 1.1** (19 de febrero de 1997); Principales adiciones incluidas: *i) reestructuración del modelo de eventos AWT* (Abstract Windowing Toolkit); *ii) clases internas* (inner classes); *iii) JavaBeans*; *iv) JDBC* (Java Database Connectivity), para la integración de bases de datos; *v) RMI* (Remote Method Invocation).

**J2SE 1.2** (8 de diciembre de 1998); Nombre clave Playground. Esta y las siguientes versiones fueron recogidas bajo la denominación Java 2 y el nombre "J2SE" (Java 2 Platform, Standard Edition), reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition). Otras mejoras añadidas incluían: *i) la palabra reservada* (keyword) *strictfp*; *ii) reflexión en la programación*; *iii) la API gráfica* (Swing) fue integrada en las clases básicas; *iv) la máquina virtual* (JVM) de Sun fue equipada con un compilador JIT (Just in Time) por primera vez; *v) Java Plug-in*; *vi) Java IDL*, una implementación de IDL (Lenguaje de Descripción de Interfaz) para la interoperabilidad con CORBA *y*; *vii) Colecciones* (Collections).

**J2SE 1.3** (8 de mayo de 2000); Nombre clave Kestrel. Los cambios más notables fueron: *i) la inclusión de la máquina virtual de HotSpot JVM* (la JVM de HotSpot fue lanzada inicialmente en abril de 1999, para la JVM de J2SE 1.2); *ii) RMI fue cambiado para que se basara en CORBA* (Common Object Request Broker Architecture); *iii) JavaSound*; *iv) se incluyó el Java Naming and Directory Interface (JNDI) en el paquete de bibliotecas principales* (anteriormente disponible como una extensión); *v) Java Platform Debugger Architecture (JPDA)*.

**J2SE 1.4** (6 de febrero de 2002) — Nombre Clave Merlin. Este fue el primer lanzamiento de la plataforma Java desarrollado bajo el Proceso de la Comunidad Java como JSR 59. Los cambios más notables fueron: *i) Palabra reservada assert* (Especificado en JSR 41); *ii) Expresiones regulares modeladas al estilo de las expresiones regulares Perl*; *iii) Encadenación de excepciones* Permite a una excepción encapsular la excepción de bajo nivel original; *iv) non-blocking NIO* (New Input/Output) (Especificado en JSR 51); *v) Logging API* (Specified in JSR 47); *vi) API I/O para la lectura y escritura de imágenes en formatos como JPEG o PNG*; *vii) Parser XML integrado y procesador XSLT (JAXP)* (Especificado en JSR 5 y JSR 63); *viii) Seguridad integrada y extensiones criptográficas* (JCE, JSSE, JAAS); *ix) Java Web Start incluido* (El primer lanzamiento ocurrió en marzo de 2001 para J2SE 1.3) (Especificado en JSR 56).

**J2SE 5.0** (30 de septiembre de 2004) — Nombre clave: Tiger. (Originalmente numerado 1.5, esta notación aún es usada internamente.[2]) Desarrollado bajo JSR 176, Tiger añadió un número significativo de nuevas características: *i) Plantillas*

(genéricos); provee conversión de tipos (type safety) en tiempo de compilación para colecciones y elimina la necesidad de la mayoría de conversión de tipos (type casting). (Especificado por JSR 14.); ii) Metadatos; también llamados anotaciones, permite a estructuras del lenguaje como las clases o los métodos, ser etiquetados con datos adicionales, que puedan ser procesados posteriormente por utilidades de proceso de metadatos. (Especificado por JSR 175.); iii) Autoboxing/unboxing; Conversiones automáticas entre tipos primitivos (Como los int) y clases de envoltura primitivas (Como Integer). (Especificado por JSR 201); iv) Enumeraciones — la palabra reservada enum crea una typesafe, lista ordenada de valores (como Dia.LUNES, Dia.MARTES, etc.). Anteriormente, esto solo podía ser llevado a cabo por constantes enteras o clases construidas manualmente (enum pattern). (Especificado por JSR 201); v) Varargs (número de argumentos variable); El último parámetro de un método puede ser declarado con el nombre del tipo seguido por tres puntos (e.g. void drawtext(String... lines)). En la llamada al método, puede usarse cualquier número de parámetros de ese tipo, que serán almacenados en un array para pasarlos al método; vi) Bucle for mejorado — La sintaxis para el bucle for se ha extendido con una sintaxis especial para iterar sobre cada miembro de un array o sobre cualquier clase que implemente Iterable, como la clase estándar Collection, de la siguiente forma:

```
void displayWidgets (Iterable<Widget> widgets) {
    for (Widget w : widgets) {
        w.display();
    }
}
```

Este ejemplo itera sobre el objeto Iterable widgets, asignando, en orden, cada uno de los elementos a la variable w, y llamando al método display() de cada uno de ellos. (Especificado por JSR 201.) }

**Java SE 6** (11 de diciembre de 2006); Nombre clave Mustang. Estuvo en desarrollo bajo la JSR 270. En esta versión, Sun cambió el nombre "J2SE" por Java SE y eliminó el ".0" del número de versión. Los cambios más importantes introducidos en esta versión son: i) Incluye un nuevo marco de trabajo y APIs que hacen posible la combinación de Java con lenguajes dinámicos como PHP, Python, Ruby y JavaScript; ii) Incluye el motor Rhino, de Mozilla, una implementación de Javascript en Java; iii) Incluye un cliente completo de Servicios Web y soporta las últimas especificaciones para Servicios Web, como JAX-WS 2.0, JAXB 2.0, STAX y JAXP; iv) Mejoras en la interfaz gráfica y en el rendimiento.

**Java SE 7**; Nombre clave Dolphin. El año 2006 aún se encontraba en las primeras etapas de planificación. Se estima su lanzamiento para julio de 2011: i) Soporte para XML dentro del propio lenguaje; ii) Un nuevo concepto de superpaquete; iii) Soporte para closures; iv) Introducción de anotaciones estándar para detectar fallos en el software.

### 0.3 Plataformas soportadas

Una versión del entorno de ejecución Java JRE (Java Runtime Environment) está disponible en la mayoría de equipos de escritorio. Sin embargo, Microsoft no lo ha incluido por defecto en sus sistemas operativos<sup>3</sup>. En el caso de Apple, éste incluye una versión propia del JRE en su sistema operativo, el Mac OS. También es un producto que por defecto aparece en la mayoría

---

<sup>3</sup> Debido a una disputa legal entre Sun y Microsoft donde este último pago 20 millones de dólares por daños y perjuicios debido a la falta de soporte a las interfaces RMI y JNI (compatibilidad de las implementaciones java). Como resultado, Microsoft dejó de usar java en su sistema operativo y en versiones recientes de Windows y su navegador Explorer no admite la ejecución de *applets* sin un *plugin* aparte.

de las distribuciones de GNU/Linux. Debido a incompatibilidades entre distintas versiones del JRE, muchas aplicaciones prefieren instalar su propia copia del JRE antes que confiar su suerte a la aplicación instalada por defecto. Los desarrolladores de *applets* de Java o bien deben insistir a los usuarios en la actualización del JRE, o bien desarrollar bajo una versión antigua de Java y verificar el correcto funcionamiento en las versiones posteriores.

#### 0.4 Características del Lenguaje

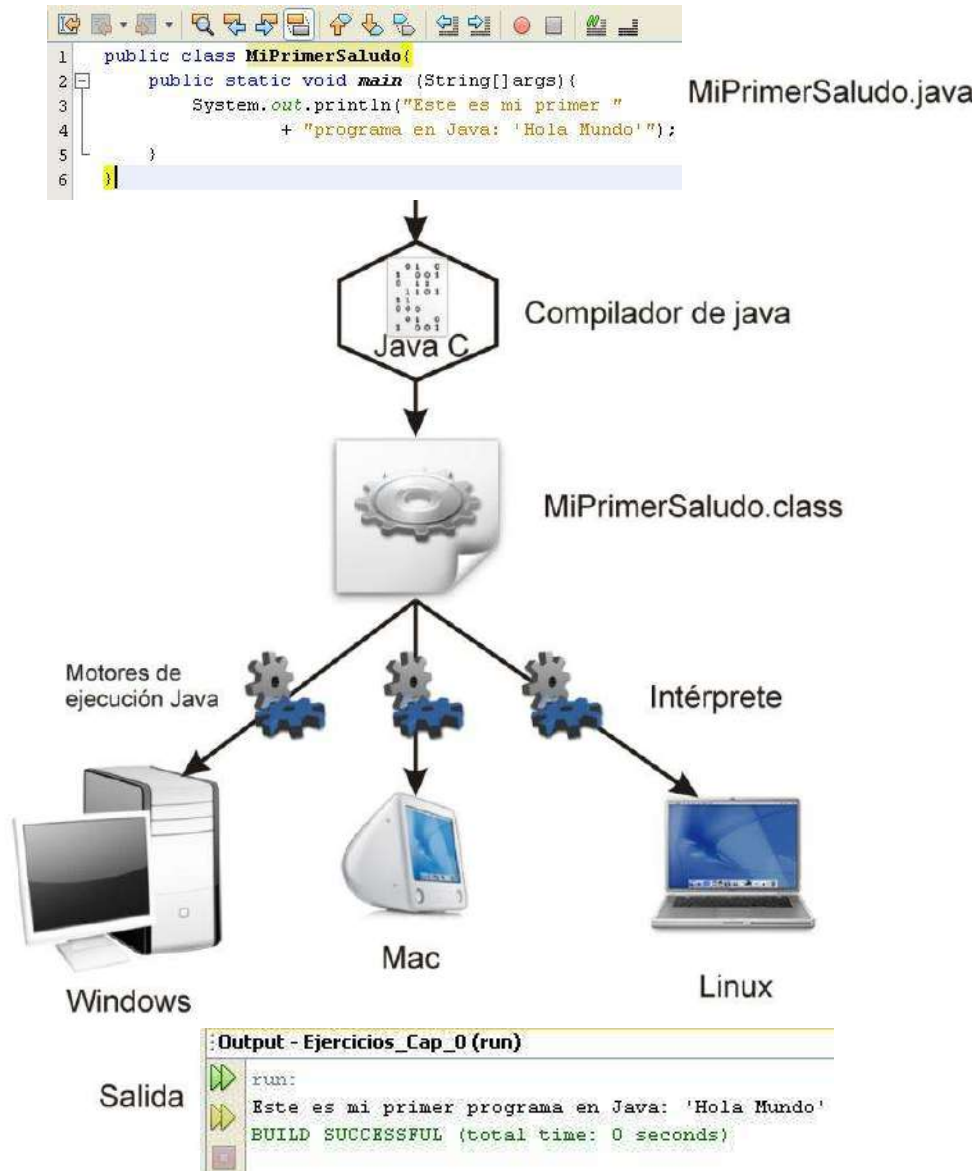
**Orientado a objetos;** La primera característica, orientado a objetos (“OO”), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que usen estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el “comportamiento” (el código) y el “estado” (datos). El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software. Un objeto genérico “cliente”, por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo.

**Simple;** Java reduce hasta en un 50% los errores más comunes de programación respecto a otros lenguajes (elimina la Aritmética de punteros, Registros, Macros, definición de tipos, liberar memoria entre otros) (Coronel, 2010).

**Distribuido;** Java proporciona librerías y herramientas para que los programas sean distribuidos, es decir, para que se ejecuten en varias máquinas e interactuando entre ellas (Coronel, 2010).

**Multiplataforma Plataforma (arquitectura neutral);** las aplicaciones java (.java) para poder ser ejecutadas, antes deberán ser compiladas dando como resultado un código intermedio denominado bytecode (.class). los bytecode son ejecutados por la Java Virtual Machine (JVM) donde son interpretados y luego traducidos al lenguaje máquina según la plataforma donde se ejecute. La ventaja de la JVM es la portabilidad del lenguaje (Coronel, 2007). Por ejemplo, el programa MiPrimerSaludo.java desarrollado en un entorno Windows (ver fig. 2) puede ser interpretado y ejecutado en cualquier, ya sea Linux o Mac, solo dependerá de que dichos entornos posean las JVM específicos para cada uno. Multiplataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Este es el significado de ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, “*write once, run anywhere*”. Para

establecer Java como parte principal de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura del ordenador donde se ejecutará. Cualquier ordenador que posea el sistema de ejecución Java Run Time (JRE) podrá ejecutar ese código objeto. Actualmente existen JRE para plataformas como Solaris 2.x, Sun Os 4.1x, Windows 95/98, Windows NT, Linux, Irix, Aix, Mac y Apple. El código fuente Java se 'compila' a código bytes (bytecode) de alto nivel independiente de la máquina donde se ejecute (Coronel, 2010).



**Figura 2 Multiplataforma**

**Es Robusto: recolección de basura;** Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución, estas comprobaciones ayudan a detectar errores en ciclo de desarrollo: obligando la declaración explícita de métodos. Además, se encarga de la liberación de memoria (Coronel, 2007). En Java el problema de las fugas de memoria se evita en gran medida gracias a la recolección de basura (*automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (*Java Runtime*) es el responsable de gestionar el ciclo de vida de los objetos. El

programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste. Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aun así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios, es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y puede que más rápida que en C++.

**Es Seguro;** a nivel de lenguaje, se eliminan punteros y casting implícito de los compiladores de lenguajes convencionales para prevenir el acceso ilegal a la memoria. A nivel de ejecución, el código Java pasa una serie de pruebas (test) antes de ejecutarse en la máquina, pasando el código a través de un verificador de código bytecode que comprueba el formato de los fragmentos de código y aplica un verificador de teoremas para detectar fragmentos de código ilegal (código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo de clase de un objeto). Las aplicaciones Java son seguras porque no acceden a zonas delicadas de memoria o de sistema, por lo que evitan la interacción de ciertos tipos de virus (Coronel, 2010). A nivel de código fuente, el JDK proporciona un desensamblador de bytecode que permite que cualquier aplicación (programa desarrollado en java) pueda ser convertido a código fuente, esta es sin duda una desventaja para el desarrollador, no obstante, existen alternativas para impedir que el código se expuesto.

**Es portable;** más allá de la portabilidad intrínseca a la multiplataforma (arquitectura neutral), Java implementa otros estándares de portabilidad, p.e. los enteros son siempre enteros (32bits) y Java construye sus interfaces de usuario a través de un sistema abstracto de ventanas de manera que las ventanas pueden ser implementadas en entornos Unix, Mac o Windows (Coronel, 2010).

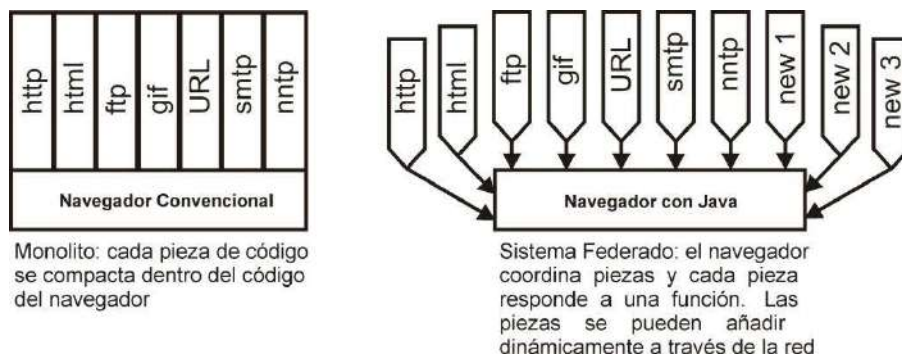
**Es interpretado;** El intérprete de java (sistema run time) puede ejecutar directamente el código objeto, sin embargo, debido a que no existen JDK específicos para las diversas plataformas, java es lento respecto a otros lenguajes ya que primero deberá ser interpretado y no ejecutado como en C++. El proceso consiste: i) primero se compila el código fuente escrito en lenguaje Java, para generar un código conocido como “bytecode” (específicamente Java bytecode)—instrucciones máquina simplificadas específicas de la plataforma Java; ii) Esta pieza está “a medio camino” entre el código fuente y el código máquina que entiende el dispositivo destino; iii) El bytecode es ejecutado entonces en la máquina virtual (JVM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran bibliotecas adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hilos o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

**Es Multihilo (multithreaded);** Java permite muchas funciones simultáneas en una aplicación, al respecto, los hilos son procesos o piezas independientes dentro de un gran proceso; al ser éstos hilos construidos en el mismo lenguaje, son más fáciles de usar y más robustos respecto a otros lenguajes. Al ser multihilo, se tiene mejor rendimiento interactivo y mejor comportamiento en tiempo real (pero limitado a las



capacidades del Sistema Operativo, OS subyacente), p.e. al navegar, java puede acceder a la información de la página sin tener que esperar a que el navegador cargue tediosamente figuras u animaciones (Coronel, 2010).

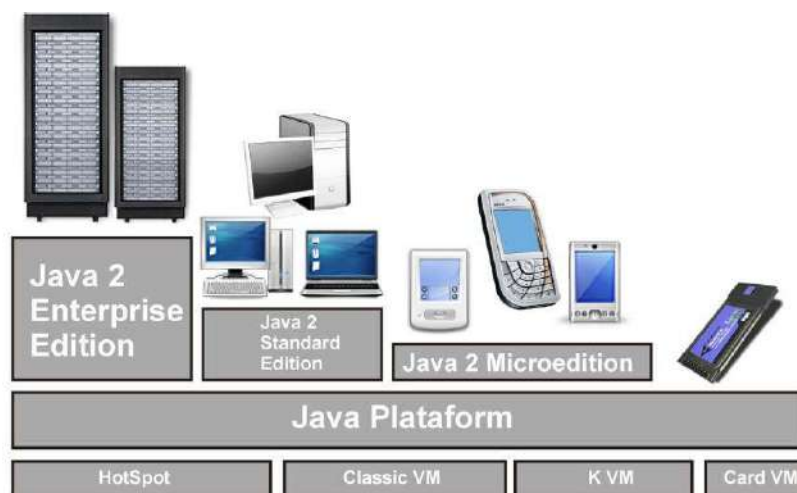
**Es Dinámico;** Java no intenta conectar todos los módulos que comprenden una aplicación hasta el mismo tiempo de ejecución. Las nuevas librerías no paralizan el tiempo de ejecución siempre que contengan el API anterior. También simplifica el uso de protocolos nuevos o actualizados; si uno ejecuta una aplicación java sobre la red, java es capaz de 'importar' automáticamente cualquier pieza que el sistema necesite para funcionar y, para evitar 'importar' cada vez, java implementa las opciones de persistencia para que no se eliminen cuando se limpie el caché de la máquina (Coronel, 2010).



**Figura 3 Sistema federado vs. monolito**

### 0.5 Entornos de funcionamiento

El diseño de Java, su robustez, el respaldo de la industria y su fácil portabilidad han hecho de Java uno de los lenguajes con un mayor crecimiento y amplitud de uso en distintos ámbitos de la industria de la informática.



**Figura 4 Entornos de funcionamiento**

**En dispositivos móviles y sistemas empotrados;** Desde la creación de la especificación J2ME (Java 2 Plataforma, Micro Edition), una versión del entorno de ejecución Java reducido y altamente optimizado, especialmente desarrollado para el

mercado de dispositivos electrónicos de consumo se ha producido toda una revolución en lo que a la extensión de Java se refiere.

Es posible encontrar microprocesadores específicamente diseñados para ejecutar bytecode Java y software Java para tarjetas inteligentes (JavaCard), teléfonos móviles, buscapersonas, set-top-boxes, sintonizadores de TV y otros pequeños electrodomésticos. El modelo de desarrollo de estas aplicaciones es muy semejante a las applets de los navegadores salvo que en este caso se denominan MIDlets.

**En el navegador web;** Desde la primera versión de java existe la posibilidad de desarrollar pequeñas aplicaciones (Applets) en Java que luego pueden ser incrustadas en una página HTML para que sean descargadas y ejecutadas por el navegador web. Estas mini-aplicaciones se ejecutan en una JVM que el navegador tiene configurada como extensión (plug-in) en un contexto de seguridad restringido configurable para impedir la ejecución local de código potencialmente malicioso.

El éxito de este tipo de aplicaciones (la visión del equipo de Gosling) no fue realmente el esperado debido a diversos factores, siendo quizás el más importante la lentitud y el reducido ancho de banda de las comunicaciones en aquel entonces que limitaba el tamaño de las applets que se incrustaban en el navegador. La aparición posterior de otras alternativas (aplicaciones web dinámicas de servidor) dejó un reducido ámbito de uso para esta tecnología, quedando hoy relegada fundamentalmente a componentes específicos para la intermediación desde una aplicación web dinámica de servidor con dispositivos ubicados en la máquina cliente donde se ejecuta el navegador.

Las applets Java no son las únicas tecnologías (aunque sí las primeras) de componentes complejos incrustados en el navegador. Otras tecnologías similares pueden ser: ActiveX de Microsoft, Flash, Java Web Start, etc.

**En sistemas de servidor;** En la parte del servidor, Java es más popular que nunca, desde la aparición de la especificación de Servlets y JSP (Java Server Pages).

Hasta entonces, las aplicaciones web dinámicas de servidor que existían se basaban fundamentalmente en componentes CGI y lenguajes interpretados. Ambos tenían diversos inconvenientes (fundamentalmente lentitud, elevada carga computacional o de memoria y propensión a errores por su interpretación dinámica). Los servlets y las JSPs supusieron un importante avance ya que:

- El API de programación es sencillo, flexible y extensible.
- Los servlets no son procesos independientes (como los CGIs) y por tanto se ejecutan dentro del mismo proceso que la JVM mejorando notablemente el rendimiento y reduciendo la carga computacional y de memoria requeridas.
- Las JSPs son páginas que se compilan dinámicamente (o se pre-compilan previamente a su distribución) de modo que el código que se consigue una ventaja en rendimiento substancial frente a muchos lenguajes interpretados.
- La especificación de Servlets y JSPs define un API de programación y los requisitos para un contenedor (servidor) dentro del cual se puedan desplegar estos componentes para formar aplicaciones web dinámicas completas. Hoy día existen multitud de contenedores (libres y comerciales) compatibles con estas especificaciones.
- A partir de su expansión entre la comunidad de desarrolladores, estas tecnologías han dado paso a modelos de desarrollo mucho más elaborados con frameworks (p.e. Struts, Webwork) que se sobreponen sobre los servlets y las JSPs para conseguir un entorno de trabajo mucho más poderoso y segmentado en el que la

especialización de roles sea posible (desarrolladores y diseñadores gráficos,) y se facilite la reutilización y robustez de código. A pesar de todo ello, las tecnologías que subyacen (Servlets y JSPs) son substancialmente las mismas.

Este modelo de trabajo se ha convertido en uno de los estándar de-facto para el desarrollo de aplicaciones web dinámicas de servidor.

**En aplicaciones de escritorio;** Hoy en día existen multitud de aplicaciones gráficas de usuario basadas en Java. El entorno de ejecución Java (JRE) se ha convertido en un componente habitual en los PC de usuario de los sistemas operativos más usados en el mundo. Además, muchas aplicaciones Java lo incluyen dentro del propio paquete de la aplicación de modo que se ejecuten en cualquier PC.

En las primeras versiones de la plataforma Java existían importantes limitaciones en las APIs de desarrollo gráfico (AWT). Desde la aparición de la biblioteca Swing la situación mejoró substancialmente y posteriormente con la aparición de bibliotecas como SWT hacen que el desarrollo de aplicaciones de escritorio complejas y con gran dinamismo, usabilidad, etc. sea relativamente sencillo.

## 0.6 Recursos

**JRE** (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

**Bibliotecas de Java**, que son el resultado de compilar el código fuente desarrollado por quien implementa la JRE, y que ofrecen apoyo para el desarrollo en Java. Algunos ejemplos de estas bibliotecas son:

- Las bibliotecas centrales, que incluyen una colección de bibliotecas para implementar estructuras de datos como listas, arrays, árboles y conjuntos.
- Bibliotecas para análisis de XML.
- Seguridad: Bibliotecas de internacionalización y localización.
- Bibliotecas de integración, que permiten la comunicación con sistemas externos. Estas bibliotecas incluyen: La API para acceso a bases de datos JDBC (Java DataBase Connectivity). La interfaz JNDI (Java Naming and Directory Interface) para servicios de directorio. RMI (Remote Method Invocation) y CORBA para el desarrollo de aplicaciones distribuidas.
- Bibliotecas para la interfaz de usuario, que incluyen: El conjunto de herramientas nativas AWT (Abstract Windowing Toolkit), que ofrece componentes GUI (Graphical User Interface), mecanismos para usarlos y manejar sus eventos asociados. Las Bibliotecas de Swing, construidas sobre AWT pero ofrecen implementaciones no nativas de los componentes de AWT.
- APIs para la captura, procesamiento y reproducción de audio. Una implementación dependiente de la plataforma en que se ejecuta de la máquina virtual de Java

(JVM), que es la encargada de la ejecución del código de las bibliotecas y las aplicaciones externas.

- Plugins o conectores que permiten ejecutar applets en los navegadores Web.
- Java Web Start, para la distribución de aplicaciones Java a través de Internet.
- Documentación y licencia.

**APIs;** Sun define tres plataformas en un intento por cubrir distintos entornos de aplicación. Así, ha distribuido muchas de sus APIs (Application Program Interface) de forma que pertenezcan a cada una de las plataformas:

- **Java ME** (Java Platform, Micro Edition) o J2ME — orientada a entornos de limitados recursos, como teléfonos móviles, PDAs (Personal Digital Assistant), etc.
- **Java SE** (Java Platform, Standard Edition) o J2SE — para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio en un PC de escritorio.
- **Java EE** (Java Platform, Enterprise Edition) o J2EE — orientada a entornos distribuidos empresariales o de Internet.

Las clases en las APIs de Java se organizan en grupos disjuntos llamados paquetes. Cada paquete contiene un conjunto de interfaces, clases y excepciones relacionadas. La información sobre los paquetes que ofrece cada plataforma puede encontrarse en la documentación de ésta.

El conjunto de las APIs es controlado por Sun Microsystems junto con otras entidades o personas a través del programa JCP (Java Community Process). Las compañías o individuos participantes del JCP pueden influir de forma activa en el diseño y desarrollo de las APIs, algo que ha sido motivo de controversia.

**Extensiones y arquitecturas relacionadas;** Las extensiones de Java están en paquetes que cuelgan de la raíz javax: javax.\*. No se incluyen en la JDK o el JRE. Algunas de las extensiones y arquitecturas ligadas estrechamente al lenguaje Java es:

- Java EE (Java Platform, Enterprise Edition; antes J2EE) —para aplicaciones distribuidas orientadas al entorno empresarial

## 0.7 Sintaxis

La sintaxis de Java se deriva en gran medida de C++. Pero a diferencia de éste, que combina la sintaxis para programación genérica, estructurada y orientada a objetos, Java fue construido desde el principio para ser completamente orientado a objetos. Todo en Java es un objeto (salvo algunas excepciones), y todo en Java reside en alguna clase (recordemos que una clase es un molde a partir del cual pueden crearse varios objetos).

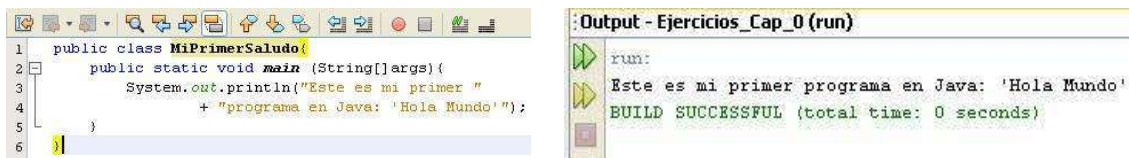
### 0.7.1 Aplicaciones Autónomas

Todo en Java está dentro de una clase, incluyendo programas autónomos.

- El código fuente se guarda en archivos con el mismo nombre que la clase que contienen y con extensión “.java”. Una clase (class) declarada pública (public) debe

seguir este convenio. En el ejemplo anterior, la clase es **MiPrimerSaludo**, por lo que el código fuente debe guardarse en el fichero "MiPrimerSaludo.java"

- El compilador genera un archivo de clase (con extensión ".class") por cada una de las clases definidas en el archivo fuente. Una clase anónima se trata como si su nombre fuera la concatenación del nombre de la clase que la encierra, el símbolo "\$", y un número entero.
- Los programas que se ejecutan de forma independiente y autónoma, deben contener el método "main()".
- La palabra reservada "void" indica que el método main no devuelve nada.
- El método main debe aceptar un array de objetos tipo String. Por acuerdo se referencia como "args", aunque puede emplearse cualquier otro identificador.
- La palabra reservada "static" indica que el método es un método de clase, asociado a la clase en vez de una instancia de la misma. El método main debe ser estático o "de clase".
- La palabra reservada public significa que un método puede ser llamado desde otras clases, o que la clase puede ser usada por clases fuera de la jerarquía de la propia clase. Otros tipos de acceso son "private" o "protected".
- La utilidad de impresión (en pantalla por ejemplo) forma parte de la biblioteca estándar de Java: la clase 'System' define un campo público estático llamado 'out'. El objeto out es una instancia de 'PrintStream', que ofrece el método 'println (String)' para volcar datos en la pantalla (la salida estándar).



The screenshot shows two windows from an IDE. The left window displays the source code for a class named `MiPrimerSaludo`. The code is as follows:

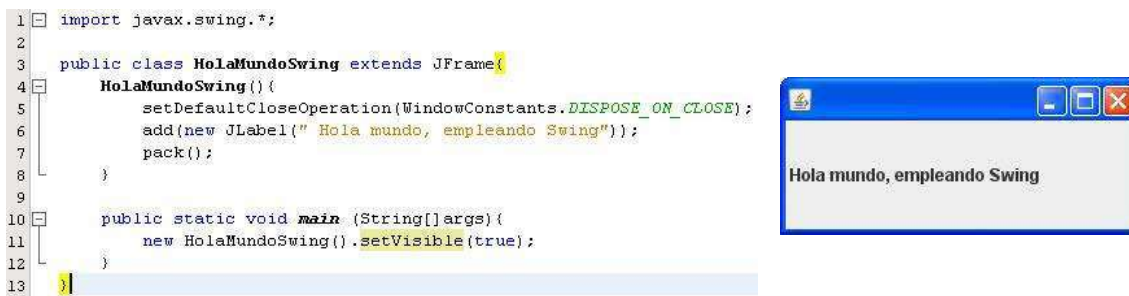
```
1 public class MiPrimerSaludo{
2     public static void main (String[] args){
3         System.out.println("Este es mi primer "
4             + "programa en Java: 'Hola Mundo'");
5     }
6 }
```

The right window, titled "Output - Ejercicios\_Cap\_0 (run)", shows the execution output:

```
run:
Este es mi primer programa en Java: 'Hola Mundo'
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 0.7.2 Aplicaciones con ventanas Swing

Swing es la biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE.



The screenshot shows an IDE with a code editor on the left and a running window on the right. The code editor shows the following code:

```
1 import javax.swing.*;
2
3 public class HolaMundoSwing extends JFrame{
4     HolaMundoSwing(){
5         setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
6         add(new JLabel(" Hola mundo, empleando Swing"));
7         pack();
8     }
9
10    public static void main (String[] args){
11        new HolaMundoSwing().setVisible(true);
12    }
13 }
```

The running window on the right has a blue title bar and contains the text "Hola mundo, empleando Swing".

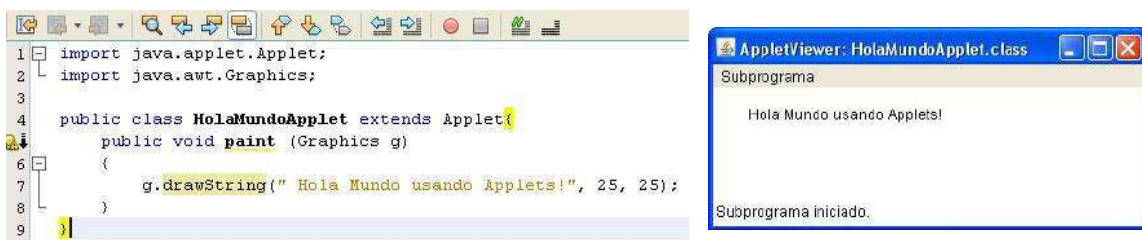
- Las instrucciones import indican al compilador de Java que las clases e interfaces del paquete javax.swing se incluyan en la compilación.
- La clase Hola extiende (extends) la clase javax.swing.JFrame, que implementa una ventana con una barra de título y un control para cerrarla.
- El constructor HolaMundoSwing() inicializa el marco o frame llamando al método setDefaultCloseOperation (int) heredado de JFrame para establecer las operaciones por defecto cuando el control de cierre en la barra de título es seleccionado al valor WindowConstants.DISPOSE\_ON\_CLOSE. Esto hace que se liberen los recursos tomados por la ventana cuando es cerrada, y no simplemente ocultada, lo que permite a la máquina virtual y al programa acabar su ejecución. A continuación se crea un objeto de tipo JLabel con el texto "Hola, mundo!", y se añade al marco mediante el

método add (Component), heredado de la clase Container. El método pack(), heredado de la clase Window, es invocado para dimensionar la ventana y distribuir su contenido.

- El método main() es llamado por la JVM al comienzo del programa. Crea una instancia de la clase Hola y hace la ventana sea mostrada invocando al método setVisible (boolean) de la superclase (clase de la que hereda) con el parámetro a true. Véase que, una vez el marco es dibujado, el programa no termina cuando se sale del método main(), ya que el código del que depende se encuentra en un hilo de ejecución independiente ya lanzado, y que permanecerá activo hasta que todas las ventanas hayan sido destruidas.

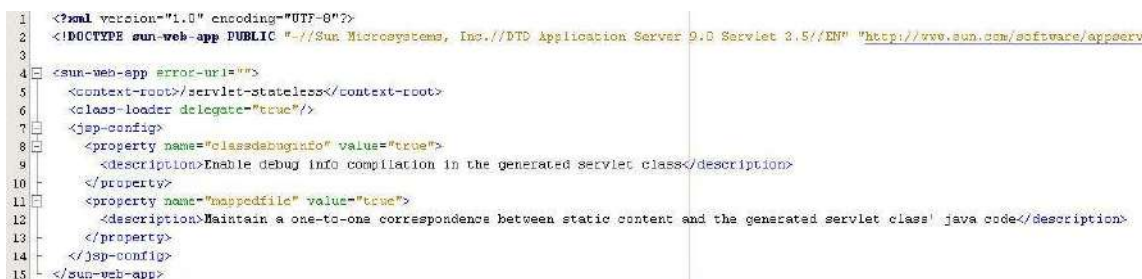
### 0.7.3 Aplicaciones Web: Applets

Son programas independientes que pueden ser interpretados por cualquier navegador con capacidades Java pero, al estar 'dentro' de una página web, las reglas de éstas le afectan (Sánchez, 2004).



### 0.7.4 Aplicaciones de Servidor: Servlets

Un servlet "es un programa escrito en Java que se ejecuta en el marco de un servicio de red, (un servidor HTTP, por ejemplo), y que recibe y responde a las peticiones de uno o más clientes". Un servlet es una clase de java, por tanto tiene todas las características del lenguaje (como portabilidad y seguridad). Una ventaja de los servlets es que quedan activos en la memoria del servidor hasta que el programa que controla el servidor los desactiva (García et. al. 1999).



## 0.8 Entornos de Desarrollo (Interface Development Environment, IDE)

Java no cuenta con un entorno de desarrollo propio, por esa razón, se puede utilizar desde un bloc de notas hasta entornos de desarrollo avanzados como NetBeans. Netbeans es un poderoso entorno de desarrollo que permite desarrollar aplicaciones complejas con interacción web, UML, base de datos, aplicaciones para telefonía móvil (Fig. 4) e inclusive Inteligencia Artificial (IA). Debido a su performance, se abordará la programación de aplicaciones en este IDE.

## 0.9 Historia de NetBeans

NetBeans comenzó en 1996 como un proyecto estudiantil en República Checa (originalmente llamado *Xelfi*), bajo la tutoría de la Facultad de Matemáticas y Física en la Universidad Carolina en Praga. La meta era escribir un entorno de desarrollo integrado (IDE) para Java parecida a la de *Delphi*. *Xelfi* fue el primer entorno de desarrollo integrado escrito en Java, con su primer pre-release en 1997. El proyecto atrajo suficiente interés, por lo que los estudiantes, después de graduarse, decidieron que lo podían convertir en un proyecto comercial. Prestando espacios web de amigos y familiares, formaron una compañía alrededor de esto. Casi todos ellos siguen trabajando en NetBeans.

Tiempo después, fueron contactados por Roman Stanek, un empresario que ya había estado relacionado con varias iniciativas en la República Checa. Estaba buscando una buena idea en la que invertir, y encontró en *Xelfi* una buena oportunidad. Así, tras una reunión, el negocio surgió. El plan original era desarrollar unos componentes JavaBeans para redes. Jarda Tulach, quien diseñó la arquitectura básica de la IDE, propuso la idea de llamarlo NetBeans, a fin de describir este propósito. Cuando las especificaciones de los Enterprise JavaBeans salieron, decidieron trabajar con este estándar, ya que no tenía sentido competir contra él, sin embargo permaneció el nombre de NetBeans.

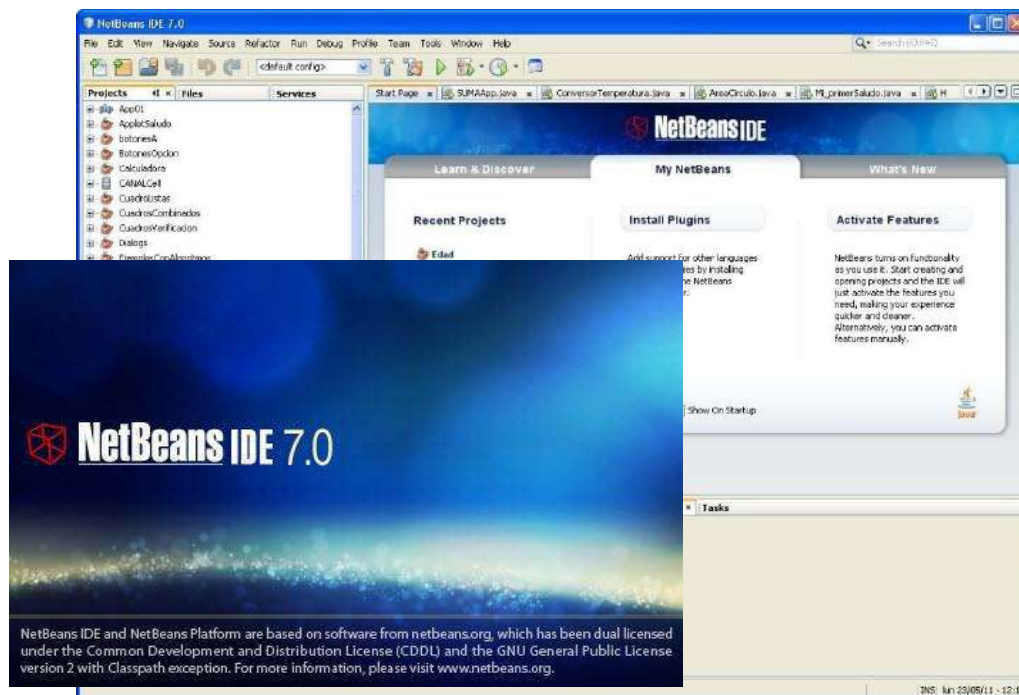


Figura 5 IDE NetBeans 7.0

El año 1999 Netbeans DeveloperX2 fue lanzado, soportando Swing. Las mejoras de rendimiento que llegaron con el JDK 1.3, lanzado en otoño de 1999, hicieron de NetBeans una alternativa realmente viable para el desarrollo de herramientas. En el verano de 1999, el equipo trabajó duro para rediseñar DeveloperX2 en un NetBeans más modular, lo que lo convirtió en la base de NetBeans hoy en día. El verano de 1999. Sun Microsystems quería una herramienta mejor de desarrollo en Java, y comenzó a estar interesado en NetBeans. El otoño de 1999, con la nueva generación de NetBeans Beta, se llegaría a un acuerdo. Sun adquirió otra compañía de herramientas al mismo tiempo, Forté, y decidió renombrar NetBeans a Forté for Java. El nombre de NetBeans desapareció por un tiempo.

Seis meses después, se tomó la decisión de hacer a NetBeans código abierto (open source). Mientras que Sun había contribuido considerablemente con líneas de código en varios proyectos de código abierto a través de los años, NetBeans se convirtió en el primer proyecto de código abierto patrocinado por ellos. En Junio del 2000 NetBeans.org fue lanzado.

## Evolución de NetBeans

Un proyecto de código abierto es un proceso que toma tiempo encontrar el equilibrio. El primer año, fue crucial como inicio. Los dos años siguientes, se orientó hacia código abierto. Como muestra de lo abierto que era, en los primeros dos años había más debate que implementación.

Con **NetBeans 3.5** se mejoró enormemente en desempeño, y con la llegada de **NetBeans 3.6**, se re-implementó el sistema de ventanas y la hoja de propiedades, y se limpió enormemente la interfaz. **NetBeans 4.0** fue un gran cambio en cuanto a la forma de funcionar del IDE, con nuevos sistemas de proyectos, con el cambio no solo de la experiencia de usuario, sino del reemplazo de muchas piezas de la infraestructura que había tenido NetBeans anteriormente. NetBeans **IDE 5.0** introdujo un soporte mucho mejor para el desarrollo de nuevos módulos, el nuevo constructor intuitivo de interfaces Matisse, un nuevo y rediseñado soporte de CVS, soporte a Sun ApplicationServer 8.2, Weblogic9 y JBoss 4.

Con **Netbeans 6.01** y **6.8** Se dio soporte a frameworks comerciales como son Struts, Hibernate.

## Versiones

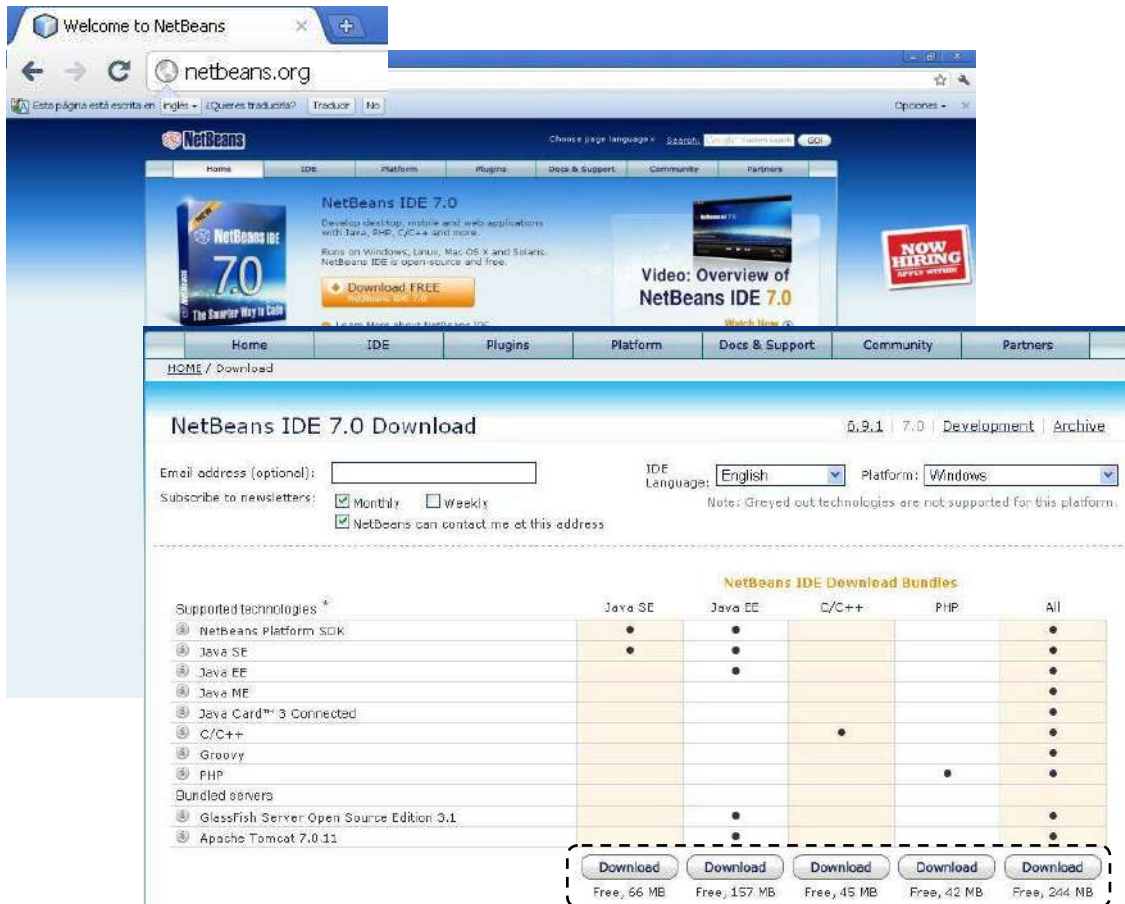


Versión	Fecha de Lanzamiento
NetBeans 6.9.1	4 de agosto de 2010
NetBeans 6.9	15 de Junio de 2010
NetBeans 6.8	10 de Diciembre de 2009
NetBeans 6.7.1	27 de Julio de 2009
NetBeans 6.7	29 de Junio de 2009
NetBeans 6.5	Noviembre 25 de 2008
NetBeans 6.1	Abril 28 de 2008
NetBeans 6.0	Diciembre 3 de 2007
NetBeans 5.5.1	Mayo 24 de 2007
NetBeans 5.5	Octubre 30 de 2006
NetBeans 5.0	Enero de 2006
NetBeans 4.1	Mayo de 2005
NetBeans 4.0	Diciembre de 2004
NetBeans 3.6	Abril de 2004
NetBeans 3.5	Junio de 2003



## 0.10 Instalación de NetBeans

Primero descargue el IDE de su página web: [www.netbeans.org](http://www.netbeans.org) en ella encontrará la última versión de su conveniencia (existen varias opciones desde la plataforma SDK + el IDE hasta opciones avanzadas que incluyen Java ME, C/C++ y PHP).



Después de descargar NetBeans, ejecute el instalador y proceda a instalar en su máquina.



Por lo general, la instalación no reviste mayor dificultad; en caso de tener alguna, consulte la página web en la sección FAQ.

## 0.11 Cuestionario

1. Estado actual de la tecnología Java
2. Paradigma o fundamento filosófico de Java
3. Qué significa Multiplataforma
4. Cuáles son las diferencias entre JSDK, JRE, J2SE, J2EE
5. Qué es un IDE? Es cierto que java no posee ningún IDE?

## Bibliografía

CORONEL, E. (2010) "Lenguaje de Programación Java" Editorial Macro, Lima, Perú. p. 1 – 40.

DEITEL. H.; DEITEL, P. (1998) "Cómo programar en Java" Editorial Mac GrawHill, D.F., México. p. 23.

GALVEZ, S.; GARCÍA, I. (2006) "Java a tope: Java mail" Universidad de Málaga, España. p.14.

GARCÍA, J.; IGNACIO, J.; IMAZ, A. (1999) "Aprenda Servlets de Java como si estuviera en segundo" Universidad de Navarra, p. 13-14.

NORIEGA, A. (2007) "Programación en Java 2" Editorial Megabyte, Lima, Perú. p.

SÁNCHEZ, J. (2004) [en línea] "Java 2" <[www.jorgesanchez.net](http://www.jorgesanchez.net)> [Acceso: 10, mar. 2011].

WIKIPEDIA. (2011) [en línea] "Java (Lenguaje de Programación)" <[www.wikipedia.org](http://www.wikipedia.org)> [Acceso: 17, mar. 2011].