1. Caso Práctico.

Unidad didáctica VII

Caso Práctico.

Cuando se trata de construir una aplicación informática, siempre se piensa en un lenguaje de programación concreto, pero en ocasiones una aplicación puede estar desarrollada en diferentes lenguajes, cada uno de ellos más adecuado a determinadas facetas. Para una empresa como ***SI Andalucía*** *disponer de un experto en cada lenguaje o de una persona que domine varios lenguajes de programación es algo inviable, no sólo por lo que debería cobrar una "joya" como ésa, sino también porque se crearía una dependencia de la empresa poco recomendable para los tiempos que corren. Lo que se suele hacer por tanto es especializarse en un lenguaje concreto que se usará para cada programa de ordenadores que se realiza y si es necesario, en ocasiones puntuales, se desarrolla alguna rutina en otro lenguaje, por ejemplo C que se "defiende" muy bien en aplicaciones cercanas al hardware. Hay muchos lenguajes de programación, pero cada vez se parecen más, incluyendo todos ellos herramientas similares o funciones predefinidas que facilitan la programación. Pero en cualquier caso cuando un grupo de jóvenes emprendedores deciden crear su propia empresa de desarrollo de software, lo más recomendable es decantarse por uno de los lenguajes (mejor dicho, entornos de desarrollo) en el que basarán la mayor parte de su producción.*

Nuestros amigos de ***SI Andalucía*** *han consensuado que su elección será Java, básicamente porque dada la actual batalla por el sistema operativo más utilizado (Windows o Guadalinex) consideran que lo ideal es un lenguaje multiplataforma de modo que cualquiera de sus aplicaciones sea fácilmente portable de uno de estos sistemas al otro, pero además a cualquier otro sistema operativo existente o de nueva aparición.*

Unidad didáctica VII

Un poco de historia sobre el lenguaje Java.

*La principal defensora de Java es **María** que argumenta que se trata del lenguaje de programación más moderno y que por tanto incluye las mejores características de sus antecesores, así como también excluye los principales defectos detectados.*

***Víctor** opina que lo ideal sería adoptar como lenguaje de programación el que conozcan la mayoría de los componentes del grupo, ya que él precisamente no sabe nada de Java. Pero **María** no está de acuerdo y dice que es muy fácil de aprender y aporta una serie de ventajas que no presenta ningún otro. A **Víctor** le sorprende lo que le cuenta **María** sobre que fue creado para programar frigoríficos y lavadoras, desde donde saltó a los ordenadores personales cuando un antiguo navegador de Internet incorporó la JVM permitiendo mayor dinamismo e interactividad en los sitios de Internet. Lo que más le convence es que, según **María**, es un lenguaje que se comporta muy bien en aplicaciones para la Red, que es lo que siempre ha querido hacer **Víctor**.*

En 1991, la empresa **Sun Microsystems** financió un proyecto de investigación sobre el uso de los procesadores en los dispositivos electrónicos de consumo inteligentes. Se necesitaba un lenguaje de programación muy portable, capaz de correr en cualquier plataforma, debido a la diversidad de dispositivos electrónicos que podrían programarse.

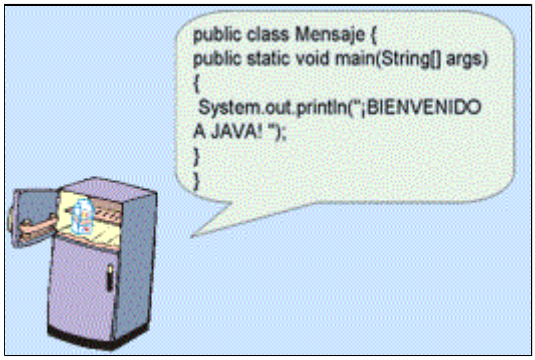
Uno de los resultados del proyecto fue la creación de un lenguaje de programación basado en C y C++, al que su creador (**James Gosling**) llamó **Oak** (roble en inglés) inspirado por un roble que crecía junto a la ventana de su trabajo en Sun. Como ya existía un lenguaje de programación que se llamaba así, hubo que cambiarle el nombre, y alguien propuso el nombre de **Java** (popularmente café en inglés) tras una reunión de un grupo de empleados de Sun en una cafetería. Y el nombre cobró fuerza. Nació el lenguaje **Java**.

Unidad didáctica VII

Necesidades que debía cumplir el nuevo lenguaje.

¿Qué fue lo que impulsó a los ingenieros de Sun a desarrollar un nuevo lenguaje?

Los electrodomésticos no se caracterizan precisamente por su gran potencia de cálculo ni por su gran cantidad de memoria. Sólo lo justo para almacenar y ejecutar el programa de lavado de la lavadora, o el tiempo cocción de un determinado plato en el horno, o la hora de encendido y apagado del horno.



¿Alguna vez habías pensado porqué hablamos de "programar" en el caso de las lavadoras o los hornos eléctricos, al igual que hacemos con los ordenadores? Realmente es programación, aunque el programa se almacena en un circuito integrado, sin que lo podamos modificar, siempre se ejecuta ese único programa y los datos se los damos con una rueda y unos botones, en vez de con un teclado, por ejemplo.

Cada día los fabricantes sacan nuevos modelos de electrodomésticos que incluyen funciones nuevas (frigoríficos que llevan un inventario de lo que has guardado en ellos, que avisan de que hay que comprar leche, lavadoras que detectan automáticamente la cantidad y el tipo de ropa o su grado de suciedad, y ajustan automáticamente el programa para minimizar el consumo de agua y energía, o cosas similares) pero todavía andan bastante lejos de las necesidades de memoria y procesamiento de un modesto ordenador de uso doméstico.

Por eso necesitaban desarrollar un lenguaje sencillo y capaz de generar código de tamaño muy reducido. Por otro lado los distintos electrodomésticos que debían ser programados disponían de distintos tipos de CPUs que además podían cambiar continuamente. Era importante conseguir una herramienta independiente del tipo de CPU utilizada. Ni C ni C++, los lenguajes más usados en esa época para el desarrollo de aplicaciones comerciales cumplían los requisitos. Les sobraban cosas y les faltaban otras para ser útiles en la programación de electrodomésticos.

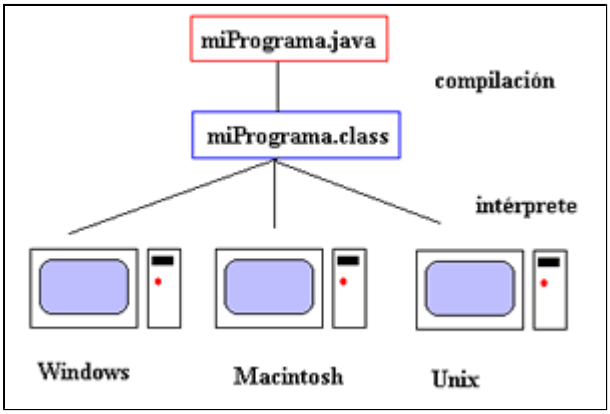
Por tanto, los ingenieros de Sun desarrollaron un código "neutro", llamado bytecodes, que no depende del tipo de procesador, (del tipo de electrodoméstico), que se ejecuta sobre una "máquina hipotética o virtual" denominada Java Virtual Machine (JVM). Es la JVM quien interpreta el código neutro (bytecodes) convirtiéndolo a código particular de la CPU utilizada (código máquina). Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: "Write Once, Run Everywhere". Podríamos traducirlo de forma más o menos libre como "escribe y compila una vez, ejecuta donde quieras".

La incorporación de esa gran portabilidad era fácilmente aplicable también a cualquier CPU, incluidas las de cualquier ordenador, y no sólo a las de electrodomésticos. Era sólo cuestión de tiempo.

Unidad didáctica VII

Causas de la difusión del lenguaje Java.

¿Y a qué se debe la enorme popularidad y la enorme difusión de un lenguaje pensado para programar electrodomésticos?



Aunque el proyecto original estuvo a punto de cancelarse, debido al poco interés mostrado por los fabricantes de electrodomésticos, en 1993 la popularidad de Internet iba en aumento e hizo que Sun advirtiera la potencialidad de Java para la creación de páginas Web con contenido dinámico, como consecuencia del modelo de traducción adoptado: el modelo de pseudocompilación a bytecodes.

La clave fue, a finales de 1995, la incorporación de un intérprete Java en el navegador Netscape Navigator, versión 2.0, produciendo una verdadera revolución en Internet.

Java 1.1 apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje y posteriormente han aparecido constantes actualizaciones.

Al ser un lenguaje altamente portable incluso después de haber sido compilado, era ideal para incluir aplicaciones incrustadas dentro del código HTML de las páginas Web que les dieran interactividad y dinamismo, ya que esas aplicaciones se descargarían y ejecutarían dentro de la página web, funcionando en cualquier ordenador que estuviese conectado a la red, con independencia de su procesador o de su sistema operativo, a condición de que el navegador tenga **instalada la máquina virtual Java (JVM- Java Virtual Machine)**. **La JVM es el intérprete encargado de traducir los bytecodes al código máquina concreto de cada ordenador**. Java es multiplataforma, ya que la fase de compilación desde el lenguaje Java a bytecodes es independiente de la máquina. Será el intérprete (la JVM) el que será específico de cada plataforma, y el que realizará la traducción final a código máquina cada vez que queramos ejecutar el programa. **Hemos ganado en portabilidad a costa de perder un poco de eficiencia.**

AUTOEVALUACIÓN

¿Qué requisito es necesario para que una aplicación desarrollada en Java se ejecute en una máquina? Señala la afirmación correcta

- a) Que la máquina tenga instalada un navegador
- b) Que la máquina tenga acceso a Internet.
- c) La máquina virtual Java(JVM).
- d) Todas las anteriores son correctas.

Comprobar

PARA SABER MÁS:
Aquí presentamos dos enlaces en los que puedes conocer más sobre la historia de Java y lo que aporta de novedoso al mundo de la programación de ordenadores.
En el primero de estos enlaces se comentan a grandes rasgos, los aspectos que desembocaron en la aparición de un sistema como Java, algo más que un lenguaje de programación. Puedes encontrar comentarios de visitantes y dejar tu comentario.
Nacimiento de Java. **[Versión en Caché]**
El segundo de estos enlaces añade, además de unos breves comentarios sobre la historia de Java, algunos apuntes sobre lo que puede ser su futuro, que se presenta prometedor.

Guía de Iniciación a Java. **[Versión en Caché]**
Como iniciado en Java una de los sitios de Internet que tienes que conocer es la página oficial de Sun Microsystem, empresa creadora de Java. Es una página en inglés en la que puedes encontrar todo lo relacionado con Sun y por supuesto con todo lo oficial de Java.
Sun Microsystems
Éste es el enlace a la página principal Sun Microsystems en español. Si no te defiendes muy bien con el inglés, ésta es tu alternativa al sitio anterior. Conoce más sobre los productos y herramientas disponibles.
Sun Microsystems español
Accede a este enlace en el que encontrarás un muy buen manual que te servirá de apoyo a la hora de programar en Java.
Manual de Apoyo **[Versión en Caché]**

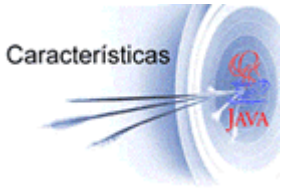
(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí

Unidad didáctica VII

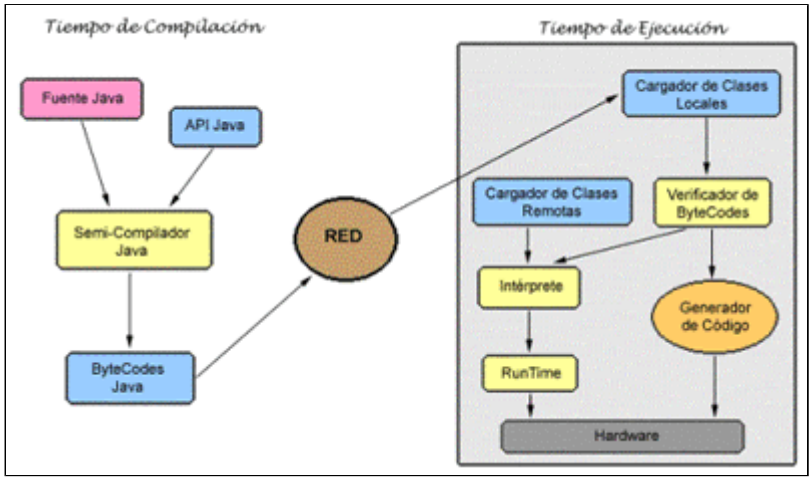
Características de Java.

Aunque aún no está muy convencido y teme que este lenguaje sea muy difícil, ***Víctor*** *se decide a aprender y tiene que empezar por conocer el lenguaje. Como siempre debe empezar por saber cuales son sus posibilidades a la hora de desarrollar aplicaciones y qué pueden aportar esas aplicaciones a los usuarios que las utilizan. Para comenzar,* ***María*** *le ha dejado un libro de iniciación a la programación con Java, así como unos cuantos sitios de Internet con manuales y artículos que pueden ayudarle a entender algunos aspectos más específicos de Java. Ahora sólo dependía de él estudiar y aprender Java cuanto antes, porque la empresa necesitaba su aportación de inmediato*

. Evidentemente tras el comienzo de la difusión del lenguaje Java y tras su incorporación en el navegador Netscape, ha sido revisado y mejorado en numerosas ocasiones. Y esto lo ha convertido en un lenguaje cada vez mejor, con una serie de características que lo hacen muy aconsejable tanto para su uso en el desarrollo de todo tipo de aplicaciones web, como para el desarrollo de todo tipo de aplicaciones "stand alone", y **también para aprender a programar** incorporando todos los conceptos novedosos de los lenguajes modernos.



¿Será demasiado complejo o demasiado difícil para usarlo como primera aproximación a los lenguajes de programación? Todo lo contrario, os lo aseguro.



Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle se apoya en un gran número de clases que ya están disponibles. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre **hay un número muy importante de clases que forman parte del propio lenguaje (el API o Application Programming Interface - Interfaz del Programador de Aplicaciones- de Java)**. Java **incorpora muchos aspectos que en cualquier otro lenguaje son extensiones** propiedad de empresas de software o fabricantes de ordenadores (**threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.**). Por eso **es un lenguaje ideal para aprender la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro** que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

En los siguientes apartados vamos a ir enumerando las características del lenguaje Java, según la definición que aporta la propia Sun Microsystem, que es la empresa que lo desarrolla. Te mostramos el enlace a la página web del tutorial de Sun, que es la fuente de la que han obtenido esa información numerosos libros publicados sobre Java, e innumerables páginas web. No obstante, esa página web está en inglés, y por facilitarte las cosas te exponemos esos apartados en una versión algo más reducida, y traducida al español, disponible también en numerosas web y libros sobre Java. La descripción de estas características es casi un estándar, ya que seguramente la mejor descripción de las características de Java es la que ofrecen sus creadores, y por eso te las presentamos prácticamente tal y como las podrás encontrar en otros sitios web.

La información original, de la web oficial de Sun, la puedes encontrar en:

http://java.sun.com/docs/white/langenv/Intro.doc2.html#334

La versión en español, de donde hemos extraído los contenidos que desarrollamos en los apartados siguientes, la puedes encontrar de forma prácticamente idéntica, entre otros, en los siguientes enlaces, que ponemos a modo de ejemplo, como lugares destacados:

En el tutorial de Java traducido al español por Agustín Froufe de la versión en ingle´s del tutorial de Sun:

http://www.itapizaco.edu.mx/paginas/JavaTut/froufe/parte2/cap2-5.html
http://web.cica.es/formacion/JavaTut/Intro/carac.html

En la web del Departamento de Tratamiento de la Información y Codificación del Instituto de Física Aplicada del Consejo Superior de Investigaciones Científicas.

http://www.iec.csic.es/criptonomicon/java/quesjava.html

Formando parte de una memoria de la E.T.S. de la Universidad de Málaga (página 22 del documento)

http://www.lcc.uma.es/~ppgg/PFC/Derytas/DerytasDoc.pdf

Aunque puedes leer directamente los apartados siguientes, que es más directo, y obtendrás aproximadamente la misma información.

Unidad didáctica VII

Simple

Java es tan potente y funcional como cualquier otro lenguaje con el que se compare, pero sin las características menos usadas y más confusas de éstos. C y C++ eran lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje para un gran número de programadores.

Java elimina muchas de las características de otros lenguajes como C++, para mantener la simplicidad del lenguaje y añade características muy útiles como el garbage collector (reciclador de memoria dinámica o recolector automático de basura).

No es necesario preocuparse de liberar memoria, el recolector de basura se encarga de ello. Como es una aplicación que se ejecuta "en segundo plano" (programa que se ejecuta en memoria de forma concurrente a otras aplicaciones) y es de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que mejora la gestión de la memoria.

Java reduce en un 50% los errores más comunes de programación con lenguajes como C y C++ al eliminar muchas de las características de éstos, entre las que destacan:

1. aritmética de punteros
2. registros (struct)
3. definición de tipos (typedef)
4. macros (#define)
5. necesidad de liberar memoria (free)

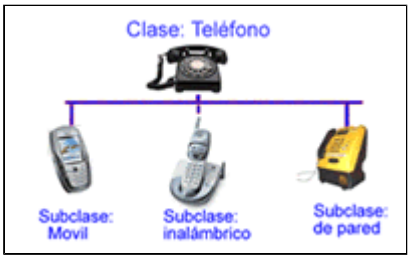
Aunque, en realidad, lo que hace es eliminar las palabras reservadas (struct, typedef), ya que las clases son algo parecido.

Unidad didáctica VII

Orientado a objetos

Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje.

- Java trabaja con sus datos como **objetos** y con interfaces a esos objetos.
- Soporta las tres características propias de la orientación a objetos: **encapsulación, herencia y polimorfismo**.
- Las plantillas de objetos son llamadas, como en C++, **clases** y sus copias, **instancias**.
- Estas instancias, como en C++, necesitan ser **construidas y destruidas en espacios de memoria**.
- Java incorpora funcionalidades inexistentes en C++ como por ejemplo, la **resolución dinámica de métodos**.
- En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada **RTTI (RunTime Type Identification-Identificación de Tipos en Tiempo de Ejecución)**. Las clases en Java tienen una representación que permite a los programadores interrogar en tiempo de ejecución por el tipo de clase y asociar dinámicamente la clase con el resultado de la búsqueda.



AUTOEVALUACIÓN

Teniendo en cuenta que Java es un lenguaje orientado a objetos y que implementa la tecnología básica de C++, señala la opción correcta:

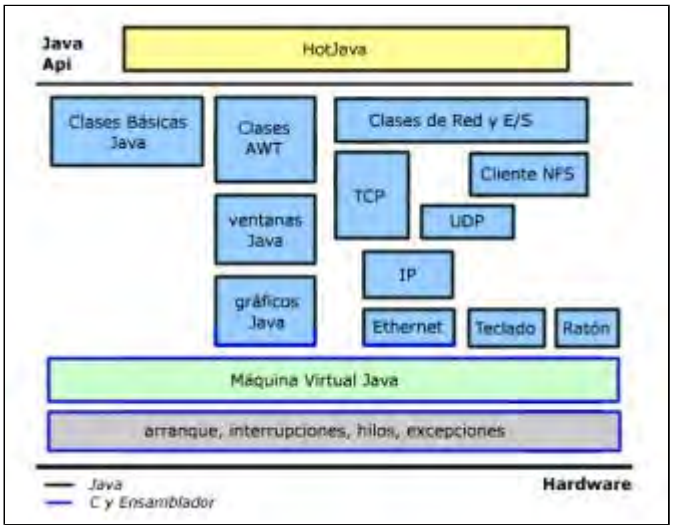
a) Las instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria.
b) Java incorpora funcionalidades inexistentes en C++ como por ejemplo, la resolución dinámica de métodos.
c) Las plantillas de objetos son llamadas, como en C++, clases y sus copias, instancias.
d) Todas las anteriores son correctas.

Comprobar

Unidad didáctica VII

Distribuido

Java se ha construido con **grandes capacidades de interconexión TCP/IP**. Existen librerías de rutinas para acceder e interactuar con protocolos como el protocolo http y el protocolo ftp. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los ficheros locales.

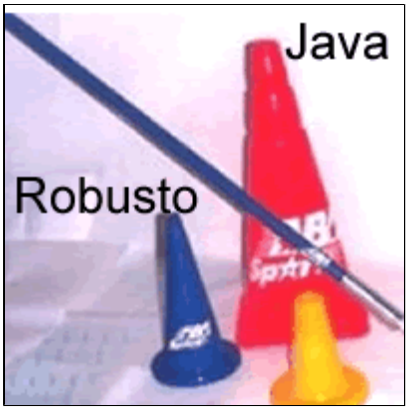


La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

Unidad didáctica VII

Robusto

Java realiza **verificaciones** en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución.



- La **comprobación de tipos** en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo.
- Java obliga a la **declaración explícita de métodos**, reduciendo así las posibilidades de error.
- **Maneja la memoria** para eliminar las preocupaciones por parte del programador de la liberación de memoria o de la corrupción de memoria.
- **Implementa los arrays auténticos**, en vez de listas enlazadas de punteros, con comprobación de límites, para evitar la posibilidad de sobrescribir o corromper memoria resultado de punteros que señalan a zonas equivocadas. Estas características reducen drásticamente el tiempo de desarrollo de aplicaciones en Java.
- Para asegurar el funcionamiento de la aplicación, realiza una **verificación de los bytecodes**. No es el código máquina directamente entendible por el hardware, pero ya ha pasado todas las fases del compilador: análisis de instrucciones, orden de operadores, etc., y ya tiene generada la pila de ejecución de órdenes.

AUTOEVALUACIÓN

Teniendo en cuenta que Java es un lenguaje robusto, señala la afirmación correcta

a) La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de depuración.

b) Java obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error.

c) El programador maneja la memoria para poder gestionar la liberación de memoria o la corrupción de la misma.

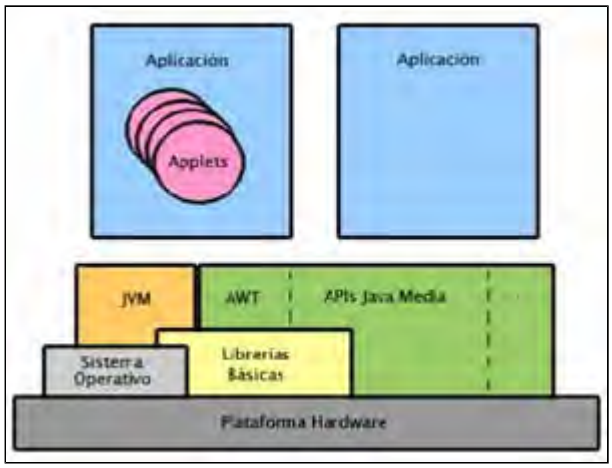
d) Todas las anteriores son correctas.

Comprobar

Unidad didáctica VII

Arquitectura neutral

Para establecer Java como parte integral de la red, el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el **sistema de ejecución (run-time o máquina virtual Java)** puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado. Actualmente existen sistemas run-time para Solaris, SunOs, Windows , Linux, Irix, Aix, Mac, Apple.



El código fuente Java se "compila" a un código de bytes de alto nivel independiente de la máquina. Este código (**bytecodes**) está diseñado para ejecutarse en una máquina hipotética que es implementada por **un sistema run-time, que sí es dependiente de la máquina**.

Lo verdaderamente dependiente del sistema es la Máquina Virtual Java (JVM) y las librerías fundamentales, que también nos permitirían acceder directamente al hardware de la máquina.

Unidad didáctica VII

Seguro

En el lenguaje Java, características como los punteros o el casting (cambio del tipo de un objeto) implícito que hacen los compiladores de C y C++ se eliminan para prevenir el acceso ilegal a la memoria. El lenguaje C, por ejemplo, tiene lagunas de seguridad importantes. Por ejemplo, los programadores de C utilizan punteros junto a operaciones aritméticas. Esto le permite al programador que un puntero haga referencia a un lugar conocido de la memoria y pueda sumar (o restar) algún valor, para referirse a otro lugar de la memoria. Si otros programadores conocen nuestras estructuras de datos pueden extraer información confidencial de nuestro sistema.

El código Java pasa muchos tests antes de ejecutarse en una máquina. El código se pasa a través de un verificador de bytecodes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal (código que falsea punteros, viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto).

Si los bytecodes pasan la verificación sin generar ningún mensaje de error, entonces sabemos que muchos errores no son posibles.

PARA SABER MÁS:
Lee este interesante enlace del Instituto de Física Aplicada del CSIC y profundiza sobre los aspectos de seguridad en Java.
Problemas de ser Multiplataforma.

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aqui

Unidad didáctica VII

Portable

Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. Además, **Java construye sus interfaces de usuario** a través de un sistema abstracto de ventanas de forma que las ventanas puedan ser implantadas en entornos Unix, Pc o Mac.

AUTOEVALUACIÓN

¿Qué entiendes ante la afirmación "una de las características más destacables de lenguaje Java es su portabilidad?.Señala la afirmación correcta.

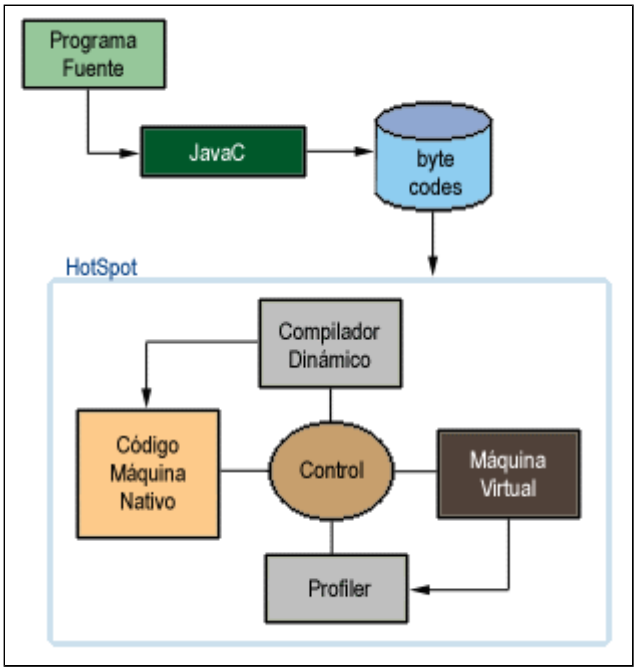
- a) Significa que cuando realizamos un programa en Java, podemos transportar los códigos fuentes del mismo de un ordenador a otro de la red mediante cualquier soporte de almacenamiento.
- b) Significa que en Java, los enteros son siempre enteros y además, enteros de 32 bits en complemento a 1.
- c) Significa que Java, además de la portabilidad básica, incluye un sistema que permite que las ventanas de interface pueden ser implantadas en entornos Unix, Pc o Mac.
- d) Todas las anteriores son correctas.

Comprobar

Unidad didáctica VII

Interpretado

El intérprete Java (sistema run-time o Máquina Virtual Java) puede ejecutar directamente los bytecodes. No obstante, Java es más lento que otros lenguajes de programación, como C++, ya que debe ser interpretado antes de ser ejecutado en vez de ser ejecutado directamente como sucede en cualquier programa compilado de forma tradicional.



La verdad es que Java para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, **es tanto interpretado como compilado**. El código fuente escrito con cualquier editor se compila generando los bytecodes. Este código intermedio es de muy bajo nivel, pero sin alcanzar las instrucciones en código máquina propias de cada plataforma. Ese mismo código es el que se puede ejecutar sobre cualquier plataforma. Para ello hace falta la JVM, que sí es completamente dependiente de la máquina y del sistema operativo, que interpreta dinámicamente los bytecodes. Con este sistema es fácil crear aplicaciones multiplataforma, pero para ejecutarlas es necesario que exista la JVM correspondiente al sistema operativo utilizado.

AUTOEVALUACIÓN

¿Por qué crees que para la ejecución de una aplicación Java en una máquina es necesario que esté instalado la JVM?.Señala la afirmación correcta.

- a) Porque la máquina virtual Java compila el código fuente de la aplicación.
- b) Porque la máquina virtual Java interpreta dinámicamente los bytecodes generados con el compilador.
- c) No es necesario que esté instalada la máquina virtual Java en una máquina para ejecutar una aplicación Java.
- d) La afirmación a y b son correctas.

Comprobar

Unidad didáctica VII

Multihebrado

Al ser multithreaded (multihebrado), Java **permite muchas actividades simultáneas en un programa**. Los hilos (a veces llamados, procesos ligeros), son básicamente piezas independientes (líneas de ejecución) de un proceso. No ha de confundirse un hilo con un proceso. **Hilo es la unidad mínima ejecutable**, mientras que proceso es la unidad mínima planificable. **Un proceso está formado por hilos**. Al estar los hilos construidos en el lenguaje, son más fáciles de usar y más robustos que sus homólogos en C o C++.

Por ejemplo, es posible que el interfaz gráfico de una aplicación (las ventanas, botones, controles, etc.) se ejecuten en un hilo (**thread**) encargado de "dibujarlos" en la pantalla, mientras que las acciones que llevan asociadas se ejecutan en otro, consiguiendo una respuesta más rápida de la aplicación.

El beneficio de ser miltithebrado consiste en un **mejor rendimiento interactivo y mejor comportamiento en tiempo real**. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa (single-threaded) tanto en facilidad de desarrollo como en rendimiento.

AUTOEVALUACIÓN

Referente a los hilos en programación, señala la afirmación correcta.

- a) Hilo es la unidad mínima ejecutable.
- b) Los procesos están compuestos por hilos.
- c) La afirmación a y b son correctas.
- d) Ninguna de las anteriores es correcta.

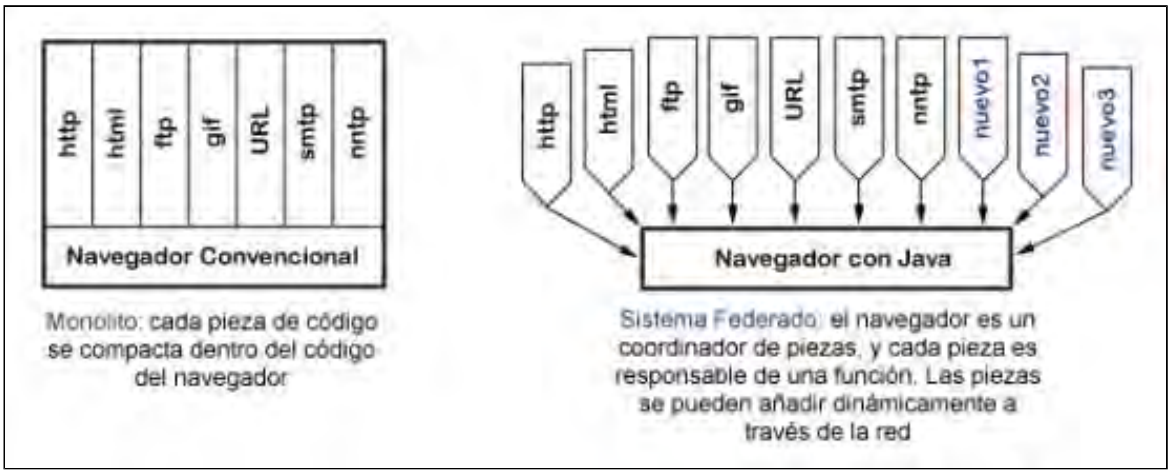
Comprobar

Unidad didáctica VII

Dinámico

Java se beneficia todo lo posible de la **tecnología orientada a objetos**. Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Las librerías nuevas o actualizadas no paralizarán las aplicaciones actuales (siempre que mantengan el API anterior).

Java también **simplifica el uso de protocolos nuevos o actualizados**. Si un sistema dado ejecuta una aplicación Java sobre la red y encuentra una pieza de la aplicación que no sabe manejar, Java es capaz de traer automáticamente cualquiera de esas piezas que el sistema necesita para funcionar.



AUTOEVALUACIÓN

De las siguientes, señala la afirmación correcta.

- a) podemos afirmar que Java es dinámico, entre otras razones, porque Java no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución.
- b) podemos afirmar que Java es dinámico, entre otras razones, porque Java conecta todos los módulos que comprenden una aplicación en el proceso de compilación, por lo cual los vínculos quedan creados en código máquina y es más rápida su llamada a la hora de la ejecución.
- c) Java no es dinámico.
- d) Todas las anteriores son falsas.

Comprobar

Unidad didáctica VII

Conceptos básicos sobre Java.

*Una vez que **Víctor** ha leído las bondades de Java, está impaciente por verlo en acción y ponerse a manejarlo para compilar sus primeros programas. **Carmen** es la compañera que le va guiar en sus primeros pasos con Java, de modo que se sientan ante el ordenador y le comenta que deben instalar en primer lugar el JDK. **Carmen** dice que se trata de un programa con el que compilar y ejecutar el código Java, que previamente se debe haber introducido en el ordenador mediante cualquier editor de texto plano, que se suministra con cualquier sistema operativo.*

Hasta ahora hemos visto un poco de historia y algunas características del lenguaje que lo hacen especialmente atractivo en comparación a otros lenguajes.

Ya va siendo hora de que empecemos a escribir algún pequeño programa Java, a compilarlo y a ejecutarlo. ¿Qué necesitaremos para empezar? ... Vamos a verlo.

- **JDK.**
- **API de Java.**
- **Variable de entorno PATH.**
- **Variable de entorno CLASSPATH.**

Unidad didáctica VII

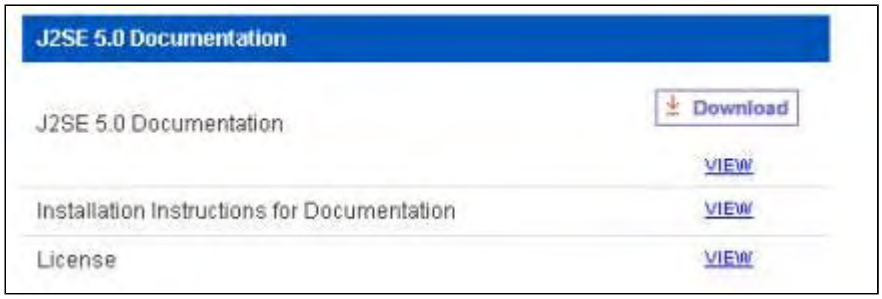
JDK

__Carmen__ explica que aunque podría tener un disco con el JDK e instalarlo, lo mejor es "bajarlo" de Internet porque así estará actualizado. Pues dicho y hecho __Carmen__ accede a la página descarga el J2SE_5.0 y lo instala en cuestión de unos minutos.

La herramienta básica para empezar a desarrollar aplicaciones o applets en Java es el JDK (Java Development Kit o **Kit de Desarrollo Java**), que consiste, básicamente, en **un compilador y un intérprete (JVM) para la línea de comandos**.No dispone de un entorno de desarrollo integrado (IDE), pero es suficiente para aprender el lenguaje y desarrollar pequeñas aplicaciones. **Es todo lo que necesitamos para poder compilar y ejecutar nuestros programas Java**.

Bueno, además necesitaremos un editor de texto plano (texto sin formato) que nos proporcionan todos los sistemas operativos, para poder escribir los programas. El JDK tiene la ventaja de que puede descargarse de forma gratuita de la propia página Web oficial de Sun (www.java.sun.com o http://java.sun.com). Actualmente recibe el nombre de **J2SE Development Kit (JDK) (Java 2 Standard Edition Development Kit)**. Este Kit de desarrollo incorpora todas las librerías de la **API** (Aplication Programer's Interface - **Interfaz para el Programador de Aplicaciones**), que también podremos utilizar sin coste alguno. También podemos descargarnos de forma gratuita toda **la documentación sobre el JDK**, incluyendo la documentación sobre las librerías de la API, pero eso sí, en inglés (o japonés).

Por ahora comenzaremos trabajando sólo con el editor de textos y con el JDK, en la consola de comandos, sin usar ningún IDE, para consolidar algunos conceptos, pero rápidamente recurriremos a usar algún entorno de desarrollo que facilite nuestra tarea.



Para descargar e instalar el JDK basta con acceder a la página de Sun, descargar el fichero adecuado a nuestro sistema operativo, y ejecutarlo. Si tienes problemas para esto puedes ver las siguientes simulaciones.

DEMO: Vea como descargar J2SE.
Pulsa en el cursor para ver los distintos pasos para descargar J2SE.

DEMO: Vea como instalar J2SE.
Pulsa en el cursor para ver los pasos a seguir para la instalación de J2SE.

Unidad didáctica VII

API de JAVA

Carmen *comenta que en el JDK se incluyen una serie de paquetes con cosas ya hechas. Dice que es algo así como las herramientas de los programadores y que las van a usar casi continuamente. Los nombra como API (Interfaz de Programación de Aplicaciones).*

La API de Java consiste en un j**uego de paquetes que se distribuyen con el JDK como bibliotecas de clases**. Estos paquetes proporcionan una interfaz común para desarrollar aplicaciones Java en todas las plataformas Java.

La API de Java está formada por varios paquetes de desarrollo principales y de un paquete de soporte de depuración. Estos paquetes son **colecciones de objetos relacionados**. Por ejemplo, vienen separados según se trate de programas de ventanas, mini-aplicaciones, software de conexión, etc.

PARA SABER MÁS:
Enlace a la página Web Oficial de Sun que incluye la documentación sobre la API de Java. La vas a encontrar en inglés, como la mayoría de los sitios de Internet, así que ve acostumbrándote.
API. Especificaciones de Java.

AUTOEVALUACIÓN

¿En qué consiste la API de Java? Señala la afirmación correcta:

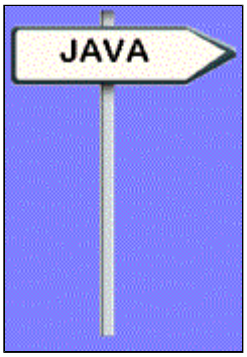
- a) En un editor de texto plano integrado.
- b) En un juego de paquetes que se distribuyen con el JDK como bibliotecas de clases que proporcionan una interfaz común para desarrollar aplicaciones Java en todas las plataformas Java.
- c) En un entorno de desarrollo integrado de Java.
- d) Ninguna de las anteriores es correcta.

Comprobar

Unidad didáctica VII

Variable de entorno PATH

***Carmen** le dice que ahora es necesario hacer unos pequeños ajustes para indicar al sistema operativo el lugar donde se encuentran los ficheros del JDK, para ello basta con añadir la ruta donde se ha instalado el JDK al PATH del sistema operativo.*



Ya hemos descargado e instalado el JDK, pero todavía falta un pequeño paso para que todo funcione sin problemas cuando trabajamos sin un IDE (entorno integrado de desarrollo). Tenemos que **señalarle al sistema operativo dónde encontrar los ficheros que le indiquen cómo compilar y ejecutar ficheros Java**.

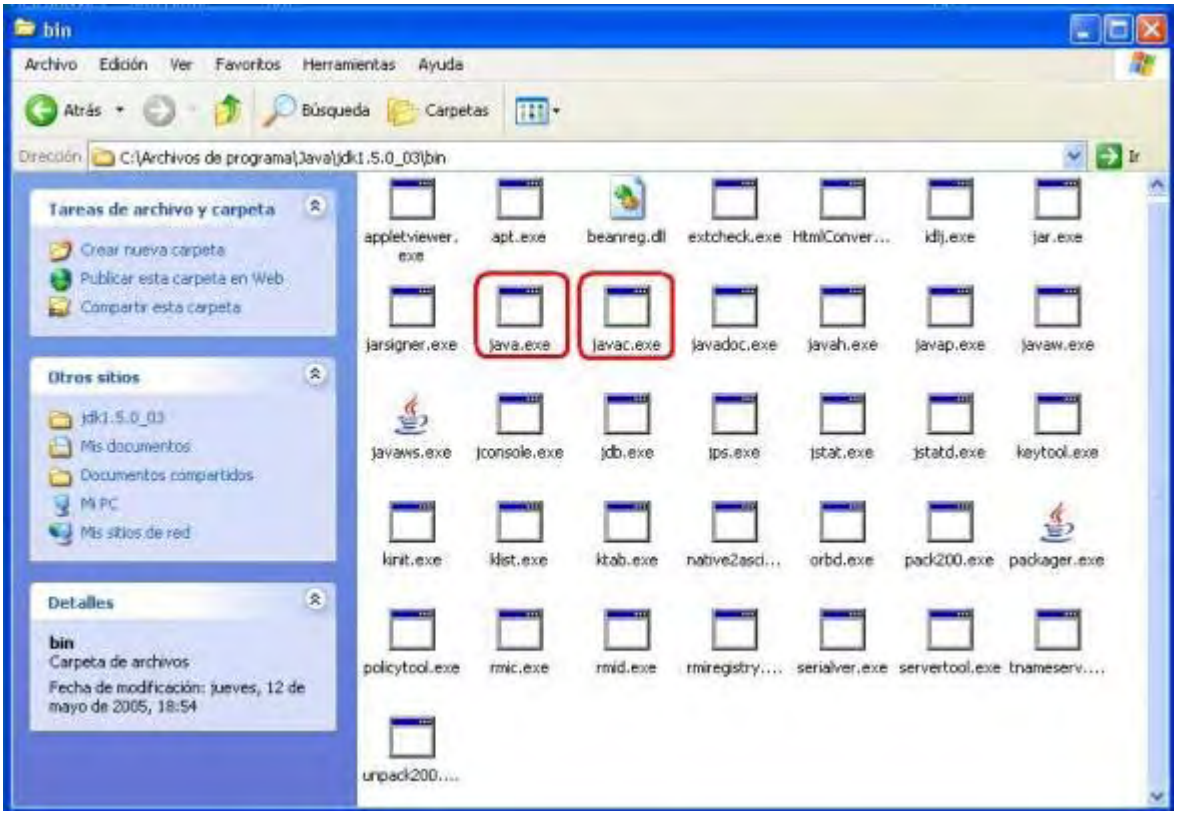
El desarrollo y ejecución de aplicaciones en Java exige que las **herramientas para compilar** (**javac.exe**) **y ejecutar** (**java.exe**) se encuentren accesibles. El ordenador, desde una ventana de comandos, sólo es capaz de **ejecutar los programas que se encuentran en los directorios indicados en la variable PATH** del ordenador. Si se desea compilar o ejecutar código en Java desde la consola de comandos, el directorio donde se encuentran estos programas (Java.exe y javac.exe) deberá encontrarse en el PATH.

Al instalar el JDK se crea una carpeta que lo contiene en el disco duro. Dependiendo de la versión del JDK que nos descarguemos e instalemos, el nombre será distinto. Por ejemplo, yo he descargado el J2SE 5.0 (versión 1.5.0_03 del JDK), y al instalarlo me ha creado en el directorio C:\Archivos de Programas\Java\ una carpeta con el nombre jdk1.5.0_03 cuyo contenido reproducimos a continuación junto al de la subcarpeta **bin**:

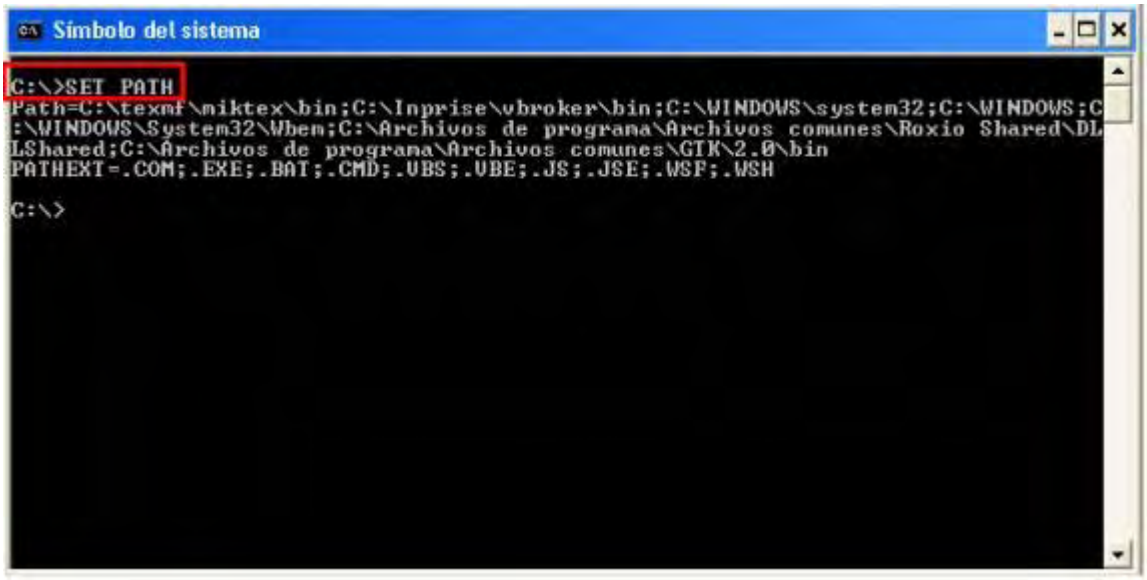
Pero esa carpeta, como decimos, puede ser distinta para distintas versiones de JDK, o incluso tú puedes elegir instalarlo en un carpeta diferente.

Vemos que contiene, entre otras cosas, una carpeta **bin** y otra carpeta **lib**.

La carpeta **bin** contiene los ejecutables, entre los que está el compilador **javac.exe** y la máquina virtual **java.exe**.



Ambos tienen que estar accesibles cuando queramos compilar un fichero con un programa Java o cuando queramos ejecutar el resultado de esa compilación, para poder ejecutarlos sin preocuparnos de la carpeta en la que estamos en ese momento. Para que todo funcione sin problemas, la ruta hasta la carpeta **bin** tiene que formar parte del PATH del sistema operativo. Tecleando SET PATH en una ventana de comandos se muestran los nombres de directorios incluidos en dicha variable de entorno. También podemos añadir rutas nuevas al PATH, entre ellas la de nuestro JDK.



PARA SABER MÁS:En este enlace aprenderás como modificar el PATH en Windows. De todas formas, recuerda que en Windows teclear variables de entorno en la propia ayuda del sistema operativo resulta bastante útil.

Cambio del Path en Windows. **[Versión en Caché]**
En este enlace correspondiente a una práctica de Conmutación de la Universidad de Navarra aprenderás cómo modificar el PATH en Linux.
Cambio del Path en Linux. **[Versión en Caché]**

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí

Unidad didáctica VII

Variable de entorno CLASSPATH

***Carmen** sugiere la creación de un fichero por lotes que le permita indicar al sistema operativo las rutas que necesita para localizar los ficheros de compilación y ejecución, así como los ficheros de clases. Han decidido llamar al fichero "**jdk.bat**".*

Al igual que por medio de la variable de entorno PATH le indicamos al sistema operativo dónde debe buscar los comandos **java.exe** y **javac.exe**, la variable de entorno CLASSPATH determina dónde buscar tanto las clases o librerías de Java (el API de Java) como otras clases de usuario, los ficheros "**.class**" que obtenemos tras la compilación de un fichero Java. **A partir de la versión 1.1.4 del JDK no es necesario indicar esta variable, salvo que se desee añadir conjuntos de clases de usuario que no vengan con dicho JDK**. Por defecto, las clases se buscarán cuando se necesiten en el directorio de trabajo activo y en el directorio **lib** del JDK.

La variable CLASSPATH puede incluir la ruta de directorios o ficheros comprimidos "***.zip**" o "***.jar**" en los que se encuentren los ficheros "***.class**" (Es capaz de encontrar las clases y usarlas aunque estén en esos ficheros comprimidos) .

En el caso de archivos "***.jar**" existe una herramienta (**jar.exe**), incorporada en el JDK, que permite **generar estos ficheros a partir de los archivos compilados** "***.class**". Los ficheros "***.jar**" son archivos comprimidos y por lo tanto ocupan menos espacio que los archivos "***.class**" por separado. El algoritmo de compresión que se usa para "***.jar**" es el mismo que para "***.zip**".

Una forma general de indicar estas dos variables es crear un fichero batch de comandos de consola (*.bat) donde se indiquen los valores de las variables PATH Y CLASSPATH. Cada vez que se abra una ventana de comandos será necesario ejecutar este fichero *.bat para asignar adecuadamente estos valores. Un posible fichero llamado **jdk.bat**, podría ser como sigue:

```
set JAVAPATH=C:\Archivos de Programa\Java\jdk1.5.0_03
set PATH=.;%JAVAPATH%\bin;%PATH%
set CLASSPATH=.\;%JAVAPATH%\lib\classes.zip;%CLASSPATH%
```

- En la primera línea decimos que el PATH del JDK de Java va a ser "**C:\jdk1.5.0**"
- En la segunda establecemos como rutas de búsqueda el **directorio activo (.)**, el directorio **bin** que hay dentro del directorio del JDK (**%JAVAPATH%\bin**) y cualquier otra ruta que estuviera establecida previamente (**%PATH%**).
- En la última línea, aunque normalmente no sería necesario hacerlo con el JDK 1.5.0, establecemos como CLASSPATH el **directorio activo (.)**, las clases contenidas en el fichero comprimido "**classes.zip**" de la carpeta **lib** que contiene la ruta del JDK (**%JAVAPATH%\lib\classes.zip**) y cualquier otra ruta de búsqueda de clases que tuviésemos establecida previamente (**%CLASSPATH%**)

Unidad didáctica VII

Probando el JDK con nuestros primeros programas.

*A continuación, **Carmen** le pide a su compañero que abra un editor de textos y escriba un primer programa de ejemplo. **Víctor** acaba de guardar su archivo como le ha indicado su compañera, con el nombre "Mensaje.java", en una carpeta de prácticas (llamada PRACTIJAVA) que ha situado en el directorio raíz del disco duro. Esto fue un consejo de su amigo **José**, de ese modo la localización de la carpeta de trabajo es inmediata y nunca se pierde nada. Ahora sólo le queda compilarlo y después ejecutarlo para comprobar que realmente hace lo que se había previsto al diseñar el programa.*

En el punto anterior hemos aprendido a instalar el JDK y configurar adecuadamente las variables de entorno. Es hora de que probemos nuestro primer programa Java. Va a incluir elementos que por ahora no hemos explicado con detalle, y que no terminarás de comprender del todo, pero de lo que se trata ahora es de aprender la mecánica de la escritura y depuración de un programa en Java, de su compilación y de su ejecución.

Lo primero será abrir el bloc de notas o el editor que prefieras, y escribir el código de nuestro programa en un fichero de extensión "**.java**". Comprueba que el editor no le añade automáticamente la extensión .txt. (Es posible que lo haga, pero que si las extensiones de archivos conocidos están ocultas no lo veas)

Unidad didáctica VII

Escritura, compilación, depuración y ejecución de un programa sencillo.

El fichero se va a llamar **Mensaje.java**. Ten cuidado con las mayúsculas y minúsculas. Java es "Sensible a Mayúsculas", que significa que distingue entre mayúsculas y minúsculas, por lo que para Java 'A' es una letra distinta de 'a'.

El contenido del fichero, como es corto, te lo vamos a escribir aquí, para que lo teclees igual. Más adelante te proporcionaremos el fichero para que no tengas que escribir tanto, pero ahora debes teclear y corregir los errores que cometas, para que entiendas el proceso de compilación en Java. Y guárdalo en una carpeta que identifiques fácilmente. Por ejemplo: C:\PruebasJava.

Contenido de Mensaje.java

```
public class Mensaje {
/*El método main es el principal, es por donde se empieza a ejecutar el programa*/
  public static void main(String[] args) {
    System.out.println("¡BIENVENIDOS AL FASCINANTE MUNDO DE JAVA! ");
    System.out.println("\tEstamos haciendo el primer programa Java de PLE.");
    System.out.println("\t\tY ya vamos por la unidad 7...");
  }
}
//Fin de la clase Mensaje
```

PARA SABER MÁS:
Cómo cambiar opciones de carpeta y archivos en Windows.
Cambiar archivos y carpetas.
Enlace que explica cómo ocultar o mostrar las extensiones de archivos en Windows.
Mostrar la Extensión. [Versión en Caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí

Ahora debes abrir la consola de comandos y asegurarte de que el PATH al JDK es correcto. Mira las explicaciones del punto anterior si no recuerdas cómo hacerlo.

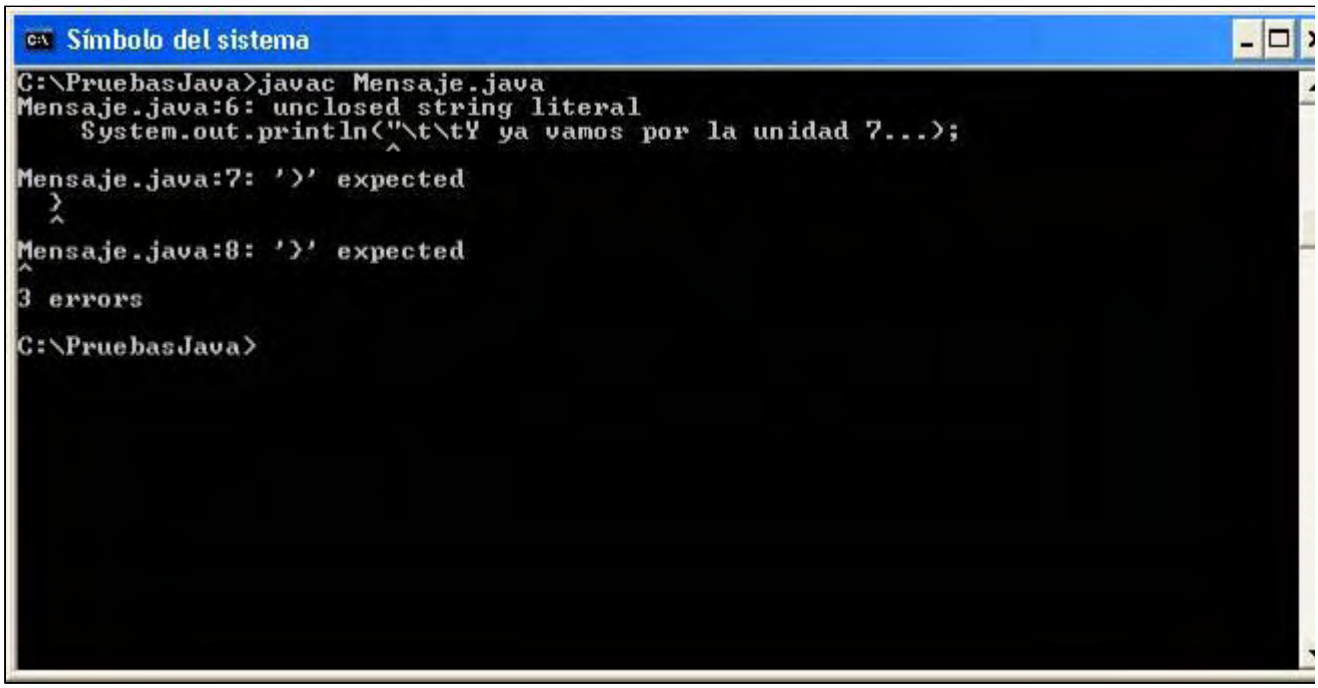
Una vez comprobado esto, vamos a compilar nuestro programa. Suponiendo que hemos fijado como directorio activo **C:\PruebasJava,** el prompt de la ventana de comandos será

C:\PruebasJava>

El cursor parpadeando espera que introduzcamos la orden que queramos, que en nuestro caso será compilar Mensaje.java

```
C:\PruebasJava> javac Mensaje.java
```

Si no nos hemos equivocado al teclear el contenido del fichero, y hemos tecleado exactamente el texto sugerido, todo debe haber ido bien, y el resultado de la compilación es que en nuestro directorio (o carpeta) de trabajo aparecerá un nuevo fichero de nombre **Mensaje.class**, que contiene los **bytecodes**, ejecutables por la máquina virtual. Si por el contrario has cometido algún error, el compilador te avisará indicándote el error que se ha producido y la línea en la que está. Por ejemplo, imagina que no hubiese puesto la última llave que cierra la definición de la clase, y que no hubiese cerrado unas comillas que encierran el texto del último mensaje. El resultado sería el siguiente:



Veamos los errores que aparecen.

- El primero me indica que en la línea 6 no he cerrado un literal de tipo String. Con el símbolo ^ marca las comillas del comienzo del literal que no se cerró. Los literales de tipo String son las cadenas de caracteres, que deben ir entre comillas. Efectivamente, le faltan las comillas de cierre tras los puntos suspensivos.
- El segundo error no es tal, y se produce porque el compilador intenta continuar adelante recuperándose del error para mostrar otros errores posibles. En este caso, la recuperación no ha sido buena, y encuentra un error que es el mismo de antes. Nos dice que en la línea 7, donde encuentra una llave de cierre (señalada con ^) esperaba un paréntesis de cierre. El paréntesis está en la línea anterior, pero como el literal String no se cerró, lo toma como un carácter más de ese literal, y al acabar la línea entiende que el paréntesis del método println no se ha cerrado. Le falta un paréntesis de cierre. Ése es el mensaje que da, pero no es el que realmente ha ocurrido. **Hay que mirar con cuidado los errores. Pueden aparecer errores que no son reales, como en este caso. Lo que haremos es mirar los primeros errores, y si los identificamos claramente, los corregimos y volvemos a compilar. Seguramente desaparecerán algunos de los errores que encontramos "extraños".**
- El tercer error me indica que en la línea 8 se esperaba una llave de cierre. Efectivamente, falta la llave de cierre.

Lo que debo hacer es abrir de nuevo el fichero **Mensaje.java** con el editor, corregir los fallos encontrados y guardarlo de nuevo para volver a compilarlo. Ese proceso tendré que repetirlo hasta que al compilar no obtenga ningún error. El fichero "**.class**" no se generará hasta entonces.

Para ejecutar nuestro programa, una vez que sea correctamente compilado, debemos teclear java seguido del nombre del fichero "**.class**" que queremos ejecutar, sin extensión, y sin olvidar que para Java las mayúsculas y las minúsculas son letras distintas (Mensaje es distinto de mensaje):

```
C:\PruebasJava> java Mensaje
```

El resultado debe ser la escritura de tres líneas de texto con los tres mensajes en la pantalla, incluyendo un tabulador más en cada una.

DEMO: Vea como ejecutar nuestro programa

Pulsa en el cursor para ver los distintos pasos a seguir para ejecutar nuestro programa.

Unidad didáctica VII

Compilación y ejecución de un fichero fuente que contenga la definición de más de una clase.

En el caso anterior en el fichero Mensaje.java sólo se definía una clase que se llamaba justamente igual que el fichero (si la clase es pública (public), obligatoriamente se tiene que llamar como el fichero que la contiene). Y al compilar obteníamos un fichero "**.class**" con el mismo nombre de la clase. Pero podemos tener varias clases definidas dentro del mismo fichero fuente. Sólo una de ellas podrá ser pública y llamarse igual que el fichero. Por cierto, es costumbre que los nombres de clase empiecen siempre por mayúscula. ¿Qué pasa al compilar? ¿Cuántos ficheros "**.class**" generará la compilación y qué nombres tendrán?

La respuesta es que lo que se compila, más que el fichero "**.java**", son las clases que contenga. Por tanto tendremos un fichero "**.class**" por cada una de las clases que tenga definidas en su interior el fichero fuente, y el nombre será el mismo que tengan las clases, con extensión "**.class**".

Veámoslo con un ejemplo.

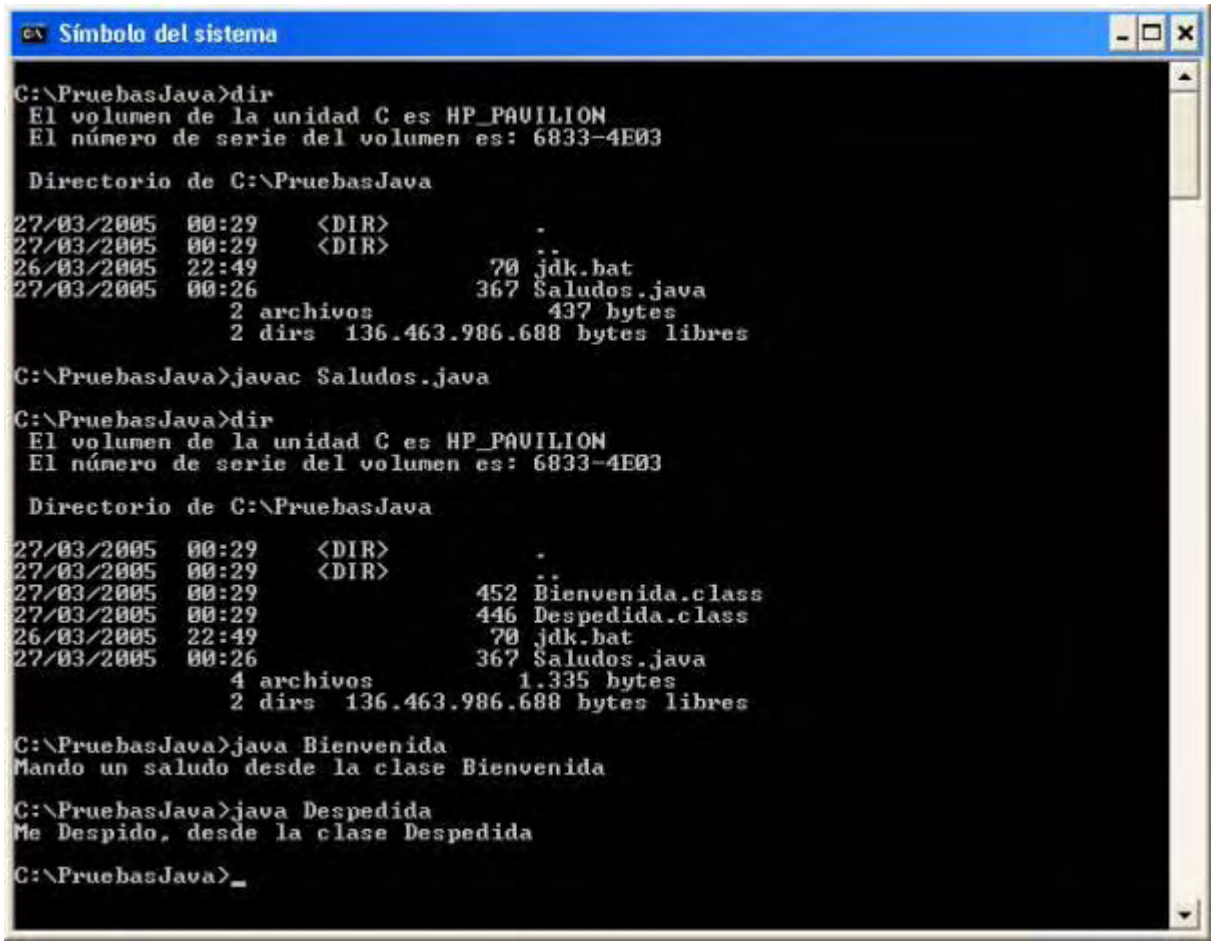
Tenemos un fichero llamado **Saludos.java**, que contiene la definición de una clase **Bienvenida** y de otra clase **Despedida**. A continuación tenemos el código que debe contener ese fichero, y la idea es que lo escribas, lo compiles, y veamos el resultado.

Saludos.java - Bloc de notas

Archivo Edición Formato Ver Ayuda

```
class Bienvenida {
  public static void main(String[] args) {
    System.out.println("Mando un saludo desde la clase Bienvenida");
  }
}
/*Ahora vamos a definir la segunda clase (Despedida)dentro del mismo fichero Saludos.java*/
class Despedida {
  public static void main(String[] args) {
    System.out.println("Me Despido, desde la clase Despedida");
  }
}
```

☐ **Descarga el archivo zip con Saludos.java**



Vemos que tras la compilación, (**C:\PruebasJava> javac Saludos.java**) aparecen dos nuevos ficheros en la carpeta:

- **Bienvenida.class**
- **Despedida.class**

Habrá un fichero "**.class**" por cada una de las clases que contenía el fichero fuente, en este caso **Saludos.java**.

Y además podemos ver que cada uno de los ficheros "**.class**" puede ejecutarse por separado, ya que ambas clases tenían definido un método main(), que es el que carga la máquina virtual para empezar la ejecución.

Unidad didáctica VII

Clases, objetos, variables y métodos.

Carmen *resume a* ***Víctor*** *los pasos que han realizado para habilitar el sistema básico de compilación de Java y le explica lo que han realizado con sus primeros programas Mensaje.java y Saludos.java: "Hemos descargado de Internet el último JDK disponible, junto a su documentación, lo hemos instalado y hemos aprendido la mecánica básica de la compilación y ejecución de programas Java. En definitiva hemos colocado las herramientas encima de la mesa y ahora es el momento de aprender a usarlas correctamente".*

Ha llegado el momento de presentar a nivel conceptual, sin entrar aún demasiado en detalles técnicos, las ideas básicas de Java.

Java es un lenguaje de programación totalmente orientado a objetos. Es imposible escribir un programa en Java que no use algún elemento de programación orientada a objetos. Incluso el ejemplo sencillo de escribir una línea de texto en la pantalla requiere de una definición de la **clase Mensaje**, de un **método main()**, del **objeto out** (que representa el dispositivo estándar de salida, y que suele ser la pantalla. Es una instancia de la clase PrintStream, que a su vez es una subclase de FilterOutputStream, que es a su vez subclase de OutputStream, que como cualquier clase de Java, es subclase de Object) perteneciente como **variable estática(static)** a la **clase System** al que se le envía el **mensaje (o método) println().**

¡Y sólo hemos hecho un simple programa para escribir una línea de texto en pantalla!

Pero esto que puede parecer muy complicado, realmente no lo es tanto. Y aunque pueda parecer que complica las cosas, realmente las simplifica. Lo irás observando a lo largo del curso. Por lo pronto, vamos a poner un ejemplo que sirva de ilustración a los conceptos de clase, objeto o instancia, variable y método o mensaje.

Unidad didáctica VII

Un ejemplo con clases, objetos, variables y métodos: Gestión de Depósitos.

__Víctor__ comenta que le gustaría empezar con el ejemplo que le dejó __José__ de la simulación del funcionamiento de los depósitos de la fábrica que fue su primer trabajo como programador. A __Carmen__ le parece bien y se deciden a emplear los programas que ya tienen para aprender el proceso básicamente y no detenerse en el diseño de las clases. __Carmen__ comienza constatando el contenido de cada fichero y anotando las clases que se definen en cada uno de ellos.

FICHEROS	Deposito.java	GestionDepositos.java	ES.java
CLASES	Deposito	GestionDepositos	ES

Una vez comprobado que la aplicación consta de tres clases (una por fichero), anota las variables definidas en cada clase e intenta identificar los métodos (operaciones) definidos sobre cada una de estas clases.

FICHEROS	Deposito.java	GestionDepositos.java	ES.java
VARIABLES	nombreDeposito	d1	
	SustanciaQueContiene	d2	
	CantidadQueContiene		
	CapacidadDeposito		
	TotalDepositosCreados		
MÉTODOS	meterEnDeposito()	main()	leeDeTeclado()
	sacarDeDeposito()	listadoDepositos()	
	vaciarDeposito()	llenarElDeposito()	
	llenarDeposito()	meterAlgoEnDeposito()	
	cosultaTotalDepositosCreados()	vaciarElDeposito()	
		sacarAlgoDeDeposito()	
		seleccionarDeposito()	

Además dice que esta aplicación crea dos __OBJETOS__ (instancias de la clase Depósito) o depósitos asociados a las variables d1 y d2.
Ahora tienen una visión más general de la aplicación que utiliza un menú de opciones para gestionar dos depósitos (objetos) de la fábrica. Aunque pueden definir todos los que quieran.
Pero hay algo que __Víctor__ no termina de entender, se trata de los constructores. __Carmen__ le explica que equivale a crear el objeto (reserva memoria para un depósito en este caso) y cabe la posibilidad de construirlo de dos modos diferentes uno que al crearlo se llena de la sustancia completamente (en nuestro caso el de agua) y el otro que puede llenar la cantidad que desee sin pasarse de la capacidad del depósito (eso ocurre en el de aceite).
Aún tenemos que introducir muchos conceptos antes de que puedas entender todo lo que aparece en el ejemplo anterior. De hecho es posible que no entiendas casi nada. Pero de la misma manera que para aprender inglés es muy conveniente ir a estudiarlo a Inglaterra, "sumergiéndote" en el idioma aunque no lo entiendas todo, con un lenguaje de programación ocurre algo similar. A fin de cuentas también es un lenguaje. Y aunque no seas consciente de ello, habrás aprendido bastantes cosas de forma intuitiva.

Se trata por ahora de que "hagas un acto de buena fe"

- compiles y ejecutes el ejemplo,
- pruebes su funcionamiento, y
- una vez que hayas visto como funciona,
- mires el código, leyendo los comentarios que acompañan a las instrucciones para explicarlas, e intentando relacionarlo con las estructuras de control de flujo que hemos visto en unidades anteriores.

No te preocupes demasiado de los detalles de sintaxis. Ya los estudiaremos. Por ahora sólo nos va a servir de base para una primera aproximación a los **conceptos de clase, objeto, variable y método**. En las unidades dedicadas a **Programación Orientada a Objetos** se definirán más conceptos y de forma más amplia. Por ahora nos vale como aproximación que tanto **clase** como **objeto** son conceptos de programación orientada a objetos, que lo que pretende es que nuestros programas se asemejen a la realidad que pretenden representar. En la vida corriente casi todo lo que nos rodea son objetos de alguna clase. Hagamos que los programas los puedan representar tal y como son en la vida real (o al menos de forma parecida)

Unidad didáctica VII

Ejemplos de Clases.

Nuestro ejemplo consta de 3 ficheros fuente: **GestionDepositos.java, Deposito.java y ES.java**.

En cada uno de ellos se define una clase, y al ser compilados generarán las correspondientes clases compiladas, de igual nombre pero extensión ".class" (los Bytecodes). Además se ejemplifican los distintos usos que tienen las clases en Java.

- **Deposito.java.** Se usa para **definir una estructura de datos, un tipo de datos** podríamos decir. Definimos lo que va a ser un Depósito, indicando dos aspectos, que ya estudiamos que básicamente son los que definen a un tipo de datos:
  - Los datos que va a contener, los valores que forman parte de un Depósito (**Nombre**, de tipo cadena de caracteres, **Sustancia que contiene**, de tipo cadena de caracteres, **Cantidad que contiene**, de tipo entero y **Capacidad**, de tipo entero) Además especificamos que el Nombre, la capacidad y la sustancia que va a contener es algo que decidimos cuando creamos el depósito (cuando lo construimos), pero que no va a poder cambiarse nunca más durante la vida útil de ese contenedor. Sin embargo la cantidad que contiene en cada momento sí que va a poder cambiar según vaciemos o llenemos el depósito. Estamos definiendo **QUÉ ES** un Depósito.
  - Las operaciones que se pueden hacer sobre los objetos de esa clase, que definen en buena medida el comportamiento de los objetos de ese tipo. Estamos definiendo qué vamos a poder hacer con un depósito. Podremos **construirlo, meter algo en él, sacar de él alguna cantidad, llenarlo completamente o vaciarlo completamente**, además de **representarlo en un listado**. Y además definimos qué se va a permitir y qué no (por ejemplo, no permito que se meta en un depósito una sustancia distinta a la que se decidió que iba a contener cuando se construyó. Estamos definiendo **QUÉ podemos HACER** con un Depósito**, CÓMO SE COMPORTA**.

☐ **Descarga el archivo zip con Deposito.java**

- **GestionDepositos.java.** Se usa para **contener el código de la aplicación** que nos interesa. Cualquier trozo de código que yo quiera ejecutar debe estar en una clase. Nosotros lo que realmente queremos es crear un par de Depósitos con unas características concretas, y gestionarlos (meter y sacar de ellos lo que proceda). Esta clase es la que contiene el método main(), que es el comienzo de la aplicación, la que usa la estructura de datos definida en Depósito.java, la clase de **control de la aplicación**, podríamos decir. Crea los depósitos y nos ofrece el menú para que digamos lo que queremos hacer con ellos. Al ser la que contiene el método main (), es la que debemos compilar, y automáticamente se compilarán también las otras dos a las que se hace referencia desde ella. Por tanto debemos compilar con el comando javac GestionDepositos.java y ejecutar con el comando Java GestionDepositos .

☐ **Descarga el archivo zip con GestionDepositos.java**

- **ES.java.** En los ejemplos hemos visto que Java nos proporciona directamente mecanismos para escribir datos en pantalla. Sin embargo, a pesar de que es un lenguaje que facilita mucho la programación, no nos da ninguna instrucción que nos permita leer sin más desde teclado frases, números enteros o números reales. Nos lo tenemos que construir nosotros. Evidentemente, lo hacemos una vez, lo incluimos en una clase y a partir de ahí usamos esta clase en todos nuestros programas para solucionar el problema. La clase ES (Entrada/Salida) se limita a contener una serie de funciones (métodos en terminología de programación orientada a objetos) que permiten leer números enteros, reales y cadenas de caracteres desde el teclado. Sirve como recipiente para contener estos métodos y poder reutilizarlos en todos nuestros ejemplos. Es otro de los usos de las clases en Java, contener métodos para que puedan referenciarse desde otras clases. Así, por ejemplo, en la sentencia

```
operacion = ES.leeNº("Elige una opcion del menu: ");
```

...estamos invocando (llamando, ejecutando) al **método leeNº** de la **claseES**, que nos escribe un mensaje en pantalla solicitando una opción y se queda a la espera de que se lo indiques con el teclado, en resumen **lee un número entero desde el teclado**. Y esta clase y sus métodos podremos usarlos en cualquier programa que en el que necesitemos leer datos desde teclado.

☐ **Descarga el archivo zip con ES.java**

La estructura básica del lenguaje Java. Parte I: Tokens, tipos básico y literales

Unidad didáctica VII

Ejemplos de Objetos.

Al principio de la clase GestionDepositos, lo primero que hacemos es declarar dos variables de tipo Deposito. Aún no se han creado los depósitos, pero cuando los creemos, los usaremos a través de esas variables. La palabra static significa que van a ser dos variables globales, comunes a todos los métodos que yo defina en la clase GestionDepositos, con las que todos van a poder trabajar. Tanto d1 como d2 van a "contener" dos objetos de tipo Deposito. Es como si la clase fuese el tipo y el objeto o instancia el valor concreto.

```
static Deposito d1, d2;
```

Más adelante, en el método main(), invocamos a los constructores de la clase Depósito a través del operador **new**, que busca espacio en memoria donde quepa un objeto de tipo Deposito, para alojar ahí la **nueva instancia de la clase depósito**, y le asignamos cada uno de esos nuevos objetos a una de las variables antes declaradas.

```
d1 = new Deposito("DEPOSITO1", "AGUA", 200);
d2 = new Deposito("DEPOSITO2", "ACEITE", 500, 250);
```

Una vez que tenemos creados y referenciados los dos nuevos depósitos, podemos "enviarles un mensaje llenarDeposito()" para que se ejecute ese método con el objeto que deseemos. Así **d1.llenarDeposito() lo que hará será llenar el depósito d1**. Es una operación que estamos haciendo con un "dato", sólo que ese dato es un objeto Depósito.

```
if (dep = = 1){
        d1.llenarDeposito();
   }else{
        d2.llenarDeposito();
   }
```

Unidad didáctica VII

Ejemplos de variables.

Ya sabemos lo que es una variable por unidades anteriores. Lo que aquí procede comentar es que en Java existen distintos tipos de variables:

- **Variables** con el significado visto en las unidades anteriores, que se correspondían con los tipos básicos. Por ejemplo, la variable operacion de GestionDepositos es una variable entera que se usa en las operaciones del programa, pero que no pertenece a los objetos, en cierta manera no es un dato fundamental de la aplicación, sino una variable auxiliar.
- **Referencias a objetos**, que no guardan el objeto en sí, sino la dirección de memoria donde está ese objeto. Por ejemplo, cuando el operador new llama al constructor para que cree un nuevo objeto, busca en memoria un espacio libre donde quepa, y cuando el resultado lo asignamos a la variable d1, lo que realmente se guarda en d1 es la dirección donde está ese depósito recién creado. **d1 es una referencia**. Ya volveremos sobre la utilidad de que esto sea así.

Desde otro punto de vista es posible otra clasificación:

- **Variables miembro de objeto**. Son las que forman parte de cada objeto, de forma que cada objeto tiene su propia copia, su propio valor, su propia zona de memoria reservada para ese valor. En nuestro ejemplo, son variables miembro de objeto el **nombreDepósito**, **sustanciaQueContiene**, **capacidadDepósito**, y **cantidadQueContiene**. Cada Depósito tendrá su propio nombre, su propia capacidad, etc.

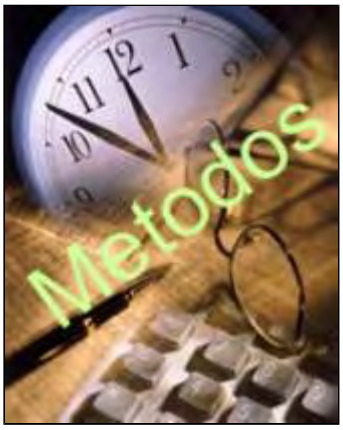
- **Variables miembro de clase**. No son propias de cada objeto, sino de la clase. Son globales a todos los objetos, y en cada momento tomarán un único valor, ya que hay una única zona de memoria reservada para ellas. En la clase Deposito, **totalDepositosCreados** es una variable miembro de clase, y en GestionDepositos, **d1** y **d2** también lo son.

Unidad didáctica VII

Ejemplos de métodos.

Nos hacemos una primera idea de lo que son las clases, de lo que son los objetos y de los usos de distintos modelos de variables que podemos definir en Java. Pero falta por definir dónde se usan todos esos elementos dentro de nuestro programa.

La respuesta es que se usan desde dentro de los métodos. Empezando por el método **main()**, que es el primero que se ejecuta, cualquier trozo de código que queramos ejecutar se llama o invoca desde algún método. En nuestro ejemplo, el método **main()** al comenzar a ejecutarse muestra un menú, pregunta la operación a realizar, y dependiendo de la elección del usuario, invoca al método adecuado. (**llenarElDeposito()**, **listadoDepositos()**, **vaciarElDeposito()**, etc.)



Los métodos de Java también reciben otros nombres en otros lenguajes, como procedimientos o funciones. Incluso en programación orientada a objetos se usa mucho la terminología mensaje, que se entenderá plenamente al estudiar con más detalle conceptos de Programación Orientada a Objetos como el polimorfismo.

- Un método es de forma general un trozo de código al que se le pone un nombre, y que podemos invocar (ejecutar) sin más que escribir ese nombre.
- El compilador sustituirá esa llamada, ese nombre, por el código que contiene el método.
- Y para permitir que la ejecución del código pueda incluir adaptaciones a distintas situaciones, el método tiene una lista de parámetros que son valores que se le indican al método en la llamada para que los use en su ejecución de forma que ésta dependa de esos valores suministrados.

Pensemos en la función matemática f(x) = 2x. podemos verla como la definición de un método encargado de calcular el doble de un número. 2x representa las instrucciones de ese método (multiplicar por dos el valor de x que se pase como parámetro. X es el parámetro formal, el número que habrá que multiplicar por dos. Y f(5) será una llamada concreta para un valor concreto, que es 5. De esta forma y =f (5) tendrá como efecto que y tomará el valor 10. De la misma manera funcionan los métodos, pero éstos son más generales que las funciones matemáticas, admiten mucha más variedad de parámetros y pueden realizar cualquier tipo de operaciones.

Unidad didáctica VII

Niveles léxico, gramatical y semántico de los lenguajes de programación.

Victor *se revela un poco y dice que la gestión de depósitos de la fábrica es algo muy sencillo y que con la aplicación informática lo que estamos haciendo es complicarlo.* ***Carmen*** *le explica que esto no puede hacerse de otro modo ya que el ordenador debe recibir las instrucciones de forma clara y sin ambigüedades. De otra forma entraría en conflictos de los que no sabría salir o realizaría operaciones que no tienen nada que ver con lo esperado, por lo que se hace necesario ajustarse a ciertos patrones que es necesario conocer bien y que están ahí para facilitar la comunicación entre el programador y la máquina.*

Seguramente te habrás preguntado cómo es posible que un ordenador, que es una máquina a fin de cuentas, sea capaz de entender las instrucciones que nosotros le proporcionamos en un lenguaje de programación. Ya estudiamos que a fin de cuentas los lenguajes de programación son bastante parecidos al lenguaje natural, pero evitando ambigüedades y con algo más de rigidez en su sintaxis. Pero siguen siendo lenguajes, al fin y al cabo, y Java no es la excepción, evidentemente.

Si uno de los objetivos de los lenguajes de programación de alto nivel era que fuesen fáciles de entender, y cercanos a nuestra forma de pensar, lo que se ha hecho evidentemente es imitar el lenguaje natural. De esta manera, todos los lenguajes de programación de alto nivel (Java, C, etc.) tienen una estructura similar a la del lenguaje natural (español, inglés, etc.). Las similitudes podrían resumirse así:

- **Estructura lingüística**: El lenguaje de programación imita (aunque simplificando la estructura y eliminando la ambigüedad) la estructura lingüística de los lenguajes, lo que supone la aparición de **los niveles léxico, gramatical y semántico.**

- **Implementación ejecutable**: Necesitamos algo que funcione en un ordenador, que sea "ejecutable". Para ello existen normas que definen:

  1. **qué símbolos son correctos** en el lenguaje,
  2. **qué combinaciones de dichos símbolos forman palabras correctas** en el lenguaje (**nivel léxico**),
  3. **qué combinaciones de palabras forman sentencias (nivel gramatical) y**
  4. **qué combinaciones de sentencias tienen sentido y constituyen un programa correcto en el lenguaje (nivel semántico).**

Por otro lado, existen las construcciones concretas en dicho lenguaje, que las normas dividen en construcciones correctas (léxica, gramatical y semánticamente bien formadas en dicho lenguaje) e incorrectas (con errores en alguno de los niveles anteriores). Y debe existir algo que sea capaz no sólo de reconocer la corrección de las construcciones, sino además de ejecutar lo que la semántica del lenguaje asocia con cada construcción correcta.

En el caso de los lenguajes de programación, ese algo que necesitábamos se concreta en la forma de compiladores o intérpretes, que traducen el texto escrito en lenguaje de alto nivel a código máquina, ejecutable por el ordenador. Pero no sólo hacen eso, sino que nos muestran los errores que encuentran en cualquiera de los niveles vistos anteriormente, para que los corrijamos.

En el caso de Java, ya sabemos que este proceso está partido en dos fases:

- compilación a bytecodes, que es la fase donde se detectarán los errores léxicos, gramaticales y semánticos.

- interpretación de los bytecodes para pasarlos a código máquina. Es un proceso de pseudocompilación.

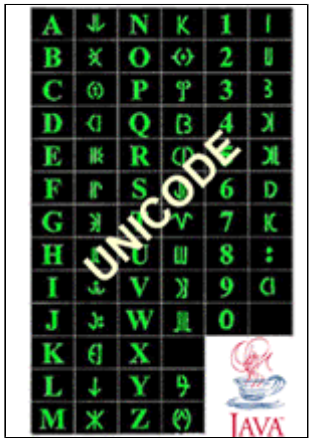
Unidad didáctica VII

El alfabeto de Java: Unicode.

Carmen *comenta que es importante conocer el código UNICODE, ya que es el que se utiliza en Java. De este modo es más acertado el uso de identificadores que crea el programador. Al poder incluir todo tipo de caracteres especiales, puede usar nombres que sean más descriptivos para él, o que estén más relacionados con el uso del elemento que se quiere nombrar. En algunos casos puede usar símbolos específicos de alfabetos de otras lenguas que no era posible utilizar cuando se empleaba el código ASCII. Incluso tiene un amigo Japonés al que le gusta ponerles nombres en japonés a sus variables. A ella misma le parece adecuado guardar el número π (aproximadamente 3.141592654) en una variable que se llame así que en otra que se llame pi. Mejor escribir π = 3.141592654, que es como se suele escribir en matemáticas, que pi = 3.141592654.*

Aunque Carmen también comenta que hay programadores, sobre todo los de la antigua escuela, que renuncian a esta posibilidad y mantienen el uso de los caracteres o símbolos habituales, lo cual es posible porque UNICODE incorpora todo ese conjunto de caracteres y no les va mal.

Como ya hemos visto, Java no es muy distinto de cualquier otro lenguaje, incluido el castellano. ¿Recuerdas cómo empezaste a aprender a escribir en tu idioma en la escuela? Con independencia de que pueda haber métodos pedagógicos distintos, e incluso más efectivos, casi todos hemos empezado aprendiendo las letras que forman parte del alfabeto de nuestra lengua, y en todo caso, usemos el método que usemos para aprender a leer, seguro que tenemos que usarlas de una en una al escribir un texto. Igual ocurre en Java. **El lenguaje se construye sobre un alfabeto**, un **conjunto de símbolos** que son los que podemos emplear para escribir nuestros programas.



Tradicionalmente, el alfabeto que venía usándose para todos los lenguajes de programación era el código ASCII de 8 bits, lo que permitía un conjunto de 256 símbolos distintos, que más o menos coincidían con los caracteres usados en la mayoría de los países occidentales. Pero eso imposibilitaba el uso de caracteres propios de los idiomas de muchos países.

Para evitar ese problema, **Java ha incorporado como alfabeto el código Unicode**, que es un nuevo código estándar de E/S que usa 16 bits para representar cada carácter, lo que da un número suficientemente amplio (2^{16} = 65.536 posibles caracteres distintos) como para permitir la representación de cualquier carácter o símbolo de cualquier idioma, incluyendo símbolos científicos, etc., por extraño y raro que sea. Además el código Unicode es totalmente compatible con ASCII, ya que respeta el código de los caracteres ASCII, a los que sencillamente les coloca 8 bits a cero delante para convertirlos en Unicode.

Así, en Java es posible que una variable se llame por ejemplo π, o que use caracteres cirílicos, o chinos o japoneses, lo que permite que los programas estén más adaptados a los lenguajes e idiomas locales, al menos en los nombres de los identificadores, que así serán más significativos para los programadores que escriban y lean el código.

PARA SABER MÁS:
Enlace en el que encontrar la tabla del código ASCII completa.
Código ASCII [Versión en Caché]
Enlace a la página principal del proyecto UNICODE. La vas a encontrar en inglés.
UNICODE
Puedes consultar todas las tablas de todos los alfabetos conocidos en...
Tablas UNICODE

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí

AUTOEVALUACIÓN

Respecto al código Unicode, señala la afirmación correcta.

- a) Se pueden representar 216 = 65.536 posibles caracteres distintos.
- b) Es el usado por Java debido a que se pueden representar muchos más caracteres distintos que con el código ASCII.
- c) El código Unicode es totalmente compatible con ASCII.
- d) Todas la anteriores son correctas

Comprobar

Unidad didáctica VII

Nivel léxico: Tokenización.

***José**, que pasaba por allí, interviene y les comenta que lo importante es utilizar adecuadamente las palabras reservadas del lenguaje Java, y decidir convenientemente los nombres de los identificadores a utilizar en cada programa. Si eso lo hacemos bien, poco importa el alfabeto que utilicemos para ello.*

Hasta ahora hemos visto los caracteres que podemos usar, el alfabeto. Pero ¿Cómo podemos unir esos símbolos para formar palabras correctas en ese lenguaje? ¿Cómo identifica el compilador qué palabras componen el texto de nuestro programa?

Inicialmente el compilador percibe nuestro fichero de texto con el código fuente en Java como una secuencia de caracteres, que le van siendo suministrados uno a uno (tal y como se escriben) y de los que tiene que ir extrayendo las palabras con significado propio (los tokens, en la jerga informática referida a compiladores.) Este proceso es el análisis léxico de nuestro programa, y se denomina tokenización. Para ello, algunos caracteres tendrán que interpretarse de forma especial por el compilador, indicándole que ahí termina un token y comienza otro. Son los separadores, y así tenemos la primera noción léxica que introduciremos, la de **separador**.

En Java, son separadores los siguientes caracteres o grupos de caracteres:

1. **Espacio en blanco**: Se trata del separador típico. Es un carácter más del alfabeto (Unicode), pero cuya misión es introducir un corte entre dos palabras.
2. **Return**: El símbolo de return, intro o nueva línea (como se le suele denominar indistintamente) es un separador con exactamente las mismas propiedades que el espacio en blanco.
3. **Tabulador**: podemos decir lo mismo, se trata de otro posible separador.
4. **Comentario**: Un comentario (de cualquiera de los tres tipos posibles en Java, que se verán más adelante) cumple asimismo las funciones de separador. Los comentarios poseen asimismo la función de documentación sobre el código fuente, pero desde el punto de vista léxico del lenguaje, no son más que separadores.
5. **Operador**. Son un caso especial, porque actúan como separadores que permiten diferenciar tokens distintos, pero con la peculiaridad de que se mantienen a sí mismos como tokens significativos, por lo que **no podrán ser sustituidos por ningún otro separador.** Son la excepción a la regla que mencionamos en los párrafos siguientes.

Por ejemplo, la expresión a=3+4 incluye dos operadores (= y +) que permiten separar el nombre de la variable a del primer operando 3 y éste del segundo operando 4, generando la secuencia de tokens a , =, 3, +, 4. Veremos con más detalle la lista completa de operadores de Java.

Durante la primera fase de análisis del código fuente, el análisis léxico, el compilador de Java aplica la siguiente regla:

Toda secuencia de uno o más separadores se sustituye por un único espacio en blanco que establece una división entre tokens.

Por ejemplo, si tenemos el ejemplo de Mensaje.java, veamos en qué consistiría el proceso de tokenización:

Contenido de Mensaje.java tal y como lo hemos escrito:

```
public class Mensaje {
/*El método main es el principal, es por donde se empieza a ejecutar el programa*/
  public static void main(String[] args) {
    System.out.println("¡BIENVENIDOS AL FASCINANTE MUNDO DE JAVA! ");
    System.out.println("\tEstamos haciendo el primer programa Java de PLE.");
    System.out.println("\t\tY ya vamos por la unidad 7...");
```

```
  }
}
//Fin de la clase Mensaje
```

Resultado de la tokenización de Mensaje.java, después de haber aplicado la regla antes mencionada:

```
public class Mensaje { public static void main ( String [ ] args ) {
  System.out.println ( " ¡BIENVENIDOS AL FASCINANTE MUNDO DE JAVA! " ) ;
  System.out.println ( " \tEstamos haciendo el primer programa Java de PLE. " ) ;
  System.out.println ( " \t \t Y ya vamos por la unidad 7... " ) ;
  } }
```

Tenemos que decir que realmente el texto se escribe en tres líneas para que se vea bien en la página, pero realmente para el compilador los cambios de línea también serían sustituidos por espacios en blanco, de forma que todo estaría en una única línea.

AUTOEVALUACIÓN

De los siguientes, señala los que son separadores en Java:

- a) Return.
- b) Operador.
- c) Comentario.
- d) Todas las anteriores son correctas.

Comprobar

Unidad didáctica VII

Tipos de Tokens.

José *insiste y comenta los convenios que él adoptó en su momento y a los que se ajusta en todo momento, de modo que localiza rápidamente una variable o identificador en cualquiera de sus programas. Dice que las palabras reservadas del lenguaje son inalterables y hay que usarlas siempre del mismo modo, respetando mayúsculas y minúsculas.*

Los operadores y literales son similares a los de cualquier otro lenguaje y las expresiones siguen los patrones de la mayoría de los lenguajes de programación. Así que los identificadores, que son los que debe decidir el programador (por ejemplo nombres de variables, clases u objetos), él siempre los escribe en minúsculas y si une varias palabras la inicial de las palabras interiores la pone en mayúscula. Por ejemplo si tiene que nombrar un archivo de copia de seguridad de la fecha 5 de mayo de 2005, el nombre que le daría podría ser;
archivoCopiaDeSeguridadDeFecha050505

Ya sabemos qué hace el compilador para aislar los tokens de un programa. Pero una vez que tenemos esa lista de "palabras", ¿qué podemos hacer con ellas para entender lo que quieren expresar, lo que nos están pidiendo que hagamos?

Lo primero será determinar el tipo de esos tokens, ya que no todos son iguales ni todos deben ser interpretados de la misma forma. Los tipos de tokens en Java son :

- **Palabras reservadas**
- **Operadores**
- **Literales**
- **Identificadores**

Pongamos por ejemplo la sentencia siguiente escrita en lenguaje Java:

```
int numero = 8 ;
```

En ella aparecen los cuatro tipos de tokens que hemos mencionado. Veámoslo por partes.

- **int** se usa para indicar que lo que ponemos a continuación va a ser una variable de tipo entero. El lenguaje le da un significado concreto, y no podemos usarla con ningún otro sentido que no sea el reservado para ella por el lenguaje. Por eso es una **palabra reservada.**
- **numero** es el nombre que yo, como programador, he decidido darle a la variable. Cualquier elemento que yo use en el lenguaje debe tener un nombre (variables, constantes, clases, métodos, etc.) y ese nombre lo elige el programador que lo crea, a su gusto, sin más que preocuparse de que sea significativo, que nos recuerde el uso que vamos a hacer de él, de forma que nos permita identificarlo y distinguirlo de otros elementos. Es un **identificador**.
- **=** es un **operador**, concretamente el operador de asignación, e indica una operación o acción a realizar, en este caso asignarle a la variable que lleva delante (numero) el valor que lleva detrás (8).
- El término 8 representa un valor concreto de los muchos que forman la categoría de los enteros en Java. Es un valor literal, **un literal** del tipo entero.

AUTOEVALUACIÓN

De los siguientes, señala los que son tipos de token en Java:

a) Palabras reservadas, operadores, comparadores.

b) Operadores, identificadores, return.

c) Literales, operadores, identificadores.

d) Todos los anteriores son correctos.

Comprobar

La estructura básica del lenguaje Java. Parte I: Tokens, tipos básico y literales

Unidad didáctica VII

Palabras reservadas.

Poco tenemos que decir en esta unidad sobre las palabras reservadas, salvo que el estudio de cualquier lenguaje dedica gran parte del tiempo a conocer el uso de sus palabras reservadas, y será lo que haremos en próximos capítulos.

La lista completa de palabras reservadas de Java es:

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

Unidad didáctica VII

Tipos básicos o primitivos.

Cada uno de los tipos básicos tiene una palabra reservada asociada que permite declarar variables como pertenecientes a ese tipo. En la unidad 2, donde hablábamos de los tipos de datos y sus características, ya hicimos una introducción a los tipos básicos de Java. ¿Los recuerdas? Por si acaso, los vamos a enumerar de nuevo, indicando la palabra reservada que los identifica.

La declaración de una variable consiste básicamente en indicar el tipo de los valores que va a almacenar seguido del nombre de esa variable, que nos va a permitir referirnos de forma cómoda al valor que contiene.

Java es un lenguaje fuertemente tipado, que quiere decir que no podremos usar ninguna variable si previamente no le hemos asignado un tipo. Además el lenguaje se preocupa de comprobar exhaustivamente que cualquier valor que se intenta guardar en una variable sea exactamente del mismo tipo que la variable. Además, cada tipo nos proporciona una serie de operaciones disponibles para usar con los datos de ese tipo.

No todos los lenguajes son así, pero que lo sean a menudo evita errores imprevistos por parte del programador.

PARA SABER MÁS:
Enlace donde podrás profundizar sobre los sistemas de tipos: tipificación estática y dinámica.
Presentación sobre la herencia en clases. [Versión en Caché]

(Si tienes problemas para acceder a algún enlace, pulsa en "Versión en Caché" para visualizar una copia de esa página web)
Para descargar el programa Acrobat Reader pulsa aquí

Los tipos primitivos en Java son:

Tipo	Descripción
boolean	Permite representar valores lógicos; Verdadero (**true**) o Falso (**false**).
char	**Permite representar cualquier símbolo o carácter UNICODE de 16 bits.**
byte	Entero de **8 bits** con signo (representado en complemento a dos). Su rango de valores va desde **-128 (-2^7)** a **+127 (+2^7-1)**.
short	**Entero de 16 bits con signo (complemento a dos). Rango de valores entre -32.768 (-2^{15}) y +32.767 (+2^{15}-1).**
int	Entero de **32 bits** con signo (complemento a dos). Rango de valores entre **-2.147.483.648 (-2^{31})** y **+2.147.483.647 (+2^{31}-1)**.
long	**Entero de 64 bits con signo (complemento a dos). Rango de valores entre -2^{63}y +2^{63}-1.**
float	Número real (en coma flotante) de 32 bits, utilizando la representación IEEE 745-1985.
double	**Número real (en coma flotante) de 64 bits, utilizando la representación IEEE 745-1985.**

En gran medida, ya explicamos lo que representa cada uno de ellos al hablar de tipos de datos en la unidad 2. Básicamente cada tipo lo que nos define es un rango de valores permitidos, y una serie de operaciones que se pueden efectuar. Y esta es la tabla que usábamos como ejemplo:

DATO	DESCRIPCION	TIPO	BREVE JUSTIFICACION
Casado	Queremos saber si el	**boolean**	Sólo son posibles dos respuestas:

	trabajador está casado o no, a fin de elaboración de horarios que faciliten la conciliación de la vida laboral y familiar		Sí o No (Verdadero o Falso)
Sexo	Queremos almacenar el sexo del empleado, y distinguir entre Hombre, Mujer y Desconocido, ya que existe la posibilidad de que el trabajador sea extranjero, con un nombre que no nos deje claro su sexo, y que haya olvidado indicarlo al rellenar la ficha de sus datos personales.	**char**	Con una sóla letra podemos distinguir entre las 3 situaciones posibles, asignando 'H' a Hombre, 'M' a mujer y 'D' a Desconocido
Día de la semana	Se expresará como número del 1 al 7, pero tenemos que tener presente que hay problemas de capacidad de memoria para el ordenador en que se va a ejecutar la aplicación, por lo que debemos ahorrar todo el espacio de almacenamiento que podamos	**byte**	Permite representar números enteros entre -128 y 127, por lo que es más que suficiente para los valores de 1 a 7 que queremos guardar. Además sólo ocupa 8 bits. Es el tipo entero más pequeño que podemos usar.
Día del año	Se expresará como un número entre 1 y 366. Al igual que antes, queremos aprovechar al máximo el espacio de almacenamiento.	**short**	Permite representar números enteros entre -32.768y 32.767, por lo que es más que suficiente para los valores de 1 a 366 que queremos guardar.Sin embargo el tipo byte no proporciona un rango suficiente. Además sólo ocupa 16 bits. Es el tipo entero más pequeño que podemos usar para este conjunto de valores.
Total Factura	Será una cantidad redondeada a euros, ya que nuestros clientes nos pagan grandes sumas, en las que incluir céntimos parece ridículo.	**int**	Permite representar números enteros, sin decimales, entre -2.147.483.648 y 2.147.483.647. Por mucho que nuestros clientes nos paguen grandes sumas, no tendremos la suerte de que nos paguen más de dos mil millones de euros en una sola factura.
Milisegundos	Número de milisegundos que han trascurrido desde las 00:00:00 horas del día 1 de Enero de 1970 hasta nuestros días.	**long**	Es una cantidad considerable, que no podemos almacenar en un int, pero que sigue siendo un número entero en el rango de long. De hecho esa es la cantidad que usa Java en la clase Date para expresar una fecha.
Sueldo Mensual	Queremos expresar el sueldo de cada empleado en euros, con precisión de céntimos. Suponemos que seguimos con limitaciones de memoria.	**float**	Es el tipo de número real (con decimales) de menor precisión y que menos memoria ocupa. No obstante, su rango de valores permite representar el sueldo de cualquier trabajador de la empresa, por elevado que éste sea. Además tampoco necesitamos más precisión, ya que para los céntimos, con dos decimales nos basta.
π	Número Pi. Necesitamos almacenar el valor del número pi para una aplicación que realiza cálculos científicos de gran precisión. Necesitamos poder representar el mayor	**double**	Es el tipo real que mayor precisión nos proporciona. El número Pi es un número real irracional, es decir, con infinitos decimales. Con double podremos representar de forma exacta el mayor número posible de

número posible de decimales del número Pi, para no perder precisión en los cálculos.

ellos.

En el siguiente apartado complementaremos la información al hablar de los literales de cada tipo.

AUTOEVALUACIÓN

Respecto a los tipos primitivos en Java, señala la afirmación correcta:

- a) int es un entero de 32 bits con signo (complemento a dos). Rango de valores entre -2.147.483.648 (-231) y +2.147.483.647 (+231-1).
- b) short es un entero de 64 bits con signo (complemento a dos). Rango de valores entre -263 y +263-1.
- c) long es un entero de 16 bits con signo (complemento a dos). Rango de valores entre -32.768 (-215) y +32.767 (+215-1).
- d) Todos los anteriores son correctos.

Comprobar

Respecto a los tipos primitivos en Java, señala la afirmación correcta:

- a) byte en un entero de 8 bits con signo (representado en complemento a dos). Su rango de valores va desde -128 (-27) a +127 (+27-1.
- b) double es un número real (en coma flotante) de 64 bits, utilizando la representación IEEE 745-1985.
- c) float es un número real (en coma flotante) de 32 bits, utilizando la representación IEEE 745-1985.
- d) Todos los anteriores son correctos.

Comprobar

De las siguientes, señala la afirmación correcta:

- a) La declaración de una variable consiste básicamente en indicar el tipo de los valores que va a almacenar seguido del nombre de esa variable, que nos va a permitir referirnos de forma cómoda al valor que contiene.
- b) Java es un lenguaje fuertemente tipado.
- c) En Java, cada tipo nos proporciona una serie de operaciones disponibles para usar con los datos de ese tipo.
- d) Todos los anteriores son correctos.

Comprobar

Unidad didáctica VII

Literales de los tipos primitivos.

Cada tipo tiene por tanto su conjunto finito de valores válidos. ¿Podré referirme a cada uno de esos valores concretos para cada tipo de forma explícita? ¿Qué diferencias habrá entre los valores de un tipo u otro?

Eso es lo que vamos a aprender a hacer en este apartado, justamente.

- **El tipo boolean**

Sólo permite dos valores posibles, que son true (verdadero) y false (falso). Es el tipo lógico estudiado en la unidad 2.

Estos dos valores, aunque son literales, funcionan en la práctica como palabras reservadas, ya que no pueden usarse con ningún otro significado en Java. Por ejemplo no se puede crear una variable llamada true, o llamada false.

Ejemplo:

```
class LiteralBoolean {
  public static void main(String[] args) {
    boolean apto = false;
    int a = ES.leeNº;("introduce una nota desde teclado");
    if (a >= 5)
      apto = true;
    System.out.println("¿Ha aprobado? : " + apto);
        }
}
```

- **Los literales del tipo char**

Se representan mediante comillas simples. Cualquier símbolo Unicode situado entre comillas simples es un literal de tipo carácter.

Ejemplos:

- 'A'
- 'a'
- '7'
- '?'
- '+'

Algunos caracteres se pueden representar utilizando secuencias de escape:

- '\n' Nueva línea (\u000A)
- '\t' Tabulador (\u0009)
- '\b' Espacio hacia atrás (\u0008)
- '\\' La barra hacia atrás (\u005C)
- '\'' La comilla simple (\u0027)
- '\"' La comilla doble (\u0022)

Los símbolos de Unicode correspondientes al ASCII (recuérdese que se trata del primero de los dos bloques de 8 bits que forman Unicode, concretamente los 8 bits menos significativos de Unicode), se

pueden representar asimismo en notación octal. Para ello, conociendo el valor octal de un carácter (N), el literal correspondiente se representa como '\N'. Por ejemplo, si sabemos que tanto en ASCII como en Unicode, la letra A (mayúscula) es el símbolo número 65, y que 65 en octal es 101, podemos representar esta letra como '\101'.

Finalmente, cualquier símbolo Unicode se puede especificar utilizando su valor hexadecimal, precedido por la secuencia de escape \u. Por ejemplo, si sabemos que el símbolo Pi mayúscula del griego se representa en Unicode con el valor hexadecimal 0370, dicho carácter se especificará en Java como el literal '\u0370'.

Ejemplo:

```
class LiteralChar {
        public static void main (String[] args) {
                char a1 = 'A';
                char a2 = '\115';
                char a3 = '\u0055';
                System.out.print("a1 = " + a1);
                System.out.print('\n');
                System.out.print("a2 = " + a2);
                System.out.print('\n');
                System.out.print("a3 = " + a3);
                System.out.print('\n');
         }
}
```

- **Literales de tipo entero.**

Un número entero se define como una secuencia de dígitos (0-9) que puede llevar delante signo o no. La única precaución que hay que tener es que para cada tipo primitivo de enteros (byte, short, int y long) se deben utilizar valores comprendidos en su rango de valores permitidos.

Los números enteros se pueden especificar en tres notaciones: decimal, octal y hexadecimal.

Por defecto, todos los números se consideran que están escritos en base decimal, a excepción de los números que comienzan por un 0, que automáticamente se consideran que están en base octal, y los que comienzan por Ox o 0X (es decir, de un cero seguido de una x, mayúscula o minúscula), que se tratan como números en hexadecimal, pudiendo contener en dicho caso, además de dígitos (los números del 0 al 9), las letras A, B, C, D, E o F (en mayúscula o minúscula).

Así, por ejemplo, los siguientes literales representan todos ellos el mismo valor numérico:

- 90
- 0132
- 0X5A
- 0x5A
- 0X5a
- 0x5a

- **Literales de tipo real.**

En Java hay dos tipos de números reales, que son float y double. La diferencia básica es la precisión de la representación y el rango de valores representables, debido al mayor tamaño de double, que usa 64 bits en vez de los 32 de float para representar cada valor.

La diferencia entre float y double es que la utilización de un mayor número de bits en el caso de double permite aumentar tanto la precisión de la mantisa (número de dígitos significativos) como los valores extremos del exponente.

En cualquier caso, lo que aquí nos interesa es la especificación de literales de este tipo.

Un número se considera real si posee decimales (lo que supone que se utilice el punto de separación entre la parte entera y la decimal, aunque cualquiera de las dos puede estar vacía), o posee un exponente (el exponente se introduce mediante la letra e ó E), o va precedido por una letra identificativa de tipo (que es f ó F para float y d ó D para double).

Según el criterio de utilización del punto (en inglés se utiliza el punto para la separación de los decimales, por lo que a estos números se los denomina de punto flotante), los siguientes números son reales y no enteros:

- 137.56
- 137.
- .56

Si la parte entera o decimal es nula (valor igual a 0) se puede omitir. Ahora bien, mientras que 137 es un número entero, 137. es un número real.

Según el criterio de exponente, un número real se puede representar como NeM (equivalente NEM), donde N es un número real y M es un entero. Esta notación, denominada científica, genera el siguiente valor:

$$NeM = N * 10^{M}$$

Por ejemplo, los siguientes números son todos reales y poseen el mismo valor:

- 1584.0
- 158.4E1 = 158.4e1 = 158.4e+1
- 15.84E2
- 1.584E3
- .1584E4

Y finalmente, según el criterio de la letra postfija de tipo, los siguientes valores serán de tipo float:

- 16f
- 16F
- 115.94f

y los siguientes de tipo double:

- 56823d
- 56823D
- 98.234d

Es importante tener en cuenta que en caso de no especificar la letra postfija correspondiente al tipo float, todos los números reales se consideran de tipo double por defecto.