Apéndice B

JAVA

1. Origen de Java

Sun Microsystems, líder en servidores para Internet, uno de cuyos lemas es *"the network is the computer"*, es quien ha desarrollado el lenguaje Java en un intento de resolver los problemas que se le plantean a los desarrolladores de software por la proliferación de sistemas incompatibles.

Hay versiones distintas sobre el origen, concepción y desarrollo de Java, desde la que dice que éste fue un proyecto que estuvo durante mucho tiempo por distintos departamentos de Sun sin que nadie le prestara atención hasta la más difundida, que presenta a Java como lenguaje de pequeños electrodomésticos.

Hace algunos años, Sun Microsystems decidió intentar introducirse en el mercado de la electrónica de consumo y desarrollar programas para pequeños dispositivos electrónicos. Tras unos comienzos dudosos, Sun decidió crear una filial, denominada FirstPerson Inc.. El mercado inicialmente previsto para los programas de FirstPerson eran los equipos domésticos: microondas, tostadoras y, fundamentalmente, televisores interactivos. En este mercado, dada la falta de pericia de los usuarios para el manejo de estos dispositivos, se requerían unos interfaces mucho más cómodos e intuitivos que los sistemas de ventanas del momento. También era importante la fiabilidad del código y la facilidad de desarrollo.

James Gosling, el miembro del equipo con más experiencia en lenguajes de programación, decidió que las ventajas aportadas por la eficiencia de C++ no compensaban el gran coste de pruebas y depuración. Gosling había estado trabajando en su tiempo libre en un lenguaje de programación que él había llamado Oak, el cual, aún partiendo de la sintaxis de C++, intentaba remediar las deficiencias que iba observando.

El primer proyecto en que se aplicó este lenguaje recibió el nombre de proyecto Green y consistía en un sistema de control completo de los aparatos electrónicos y el entorno de un hogar. Para ello se construyó un ordenador experimental denominado *7 (Star Seven). El sistema presentaba una interfaz basada en la representación de la casa de forma animada y el control se llevaba a cabo mediante una pantalla sensible al tacto. En el sistema aparecía Duke, la actual mascota de Java. Posteriormente se aplicó a otro proyecto denominado VOD (Video On Demand) en el que se empleaba como interfaz para la televisión interactiva. Ninguno de estos proyectos se convirtió nunca en un sistema comercial, pero fueron desarrollados enteramente en Java. Una vez que en Sun se dieron cuenta de que a corto plazo la televisión interactiva no iba a ser un gran éxito, urgieron a FirstPerson a desarrollar con rapidez nuevas estrategias que produjeran beneficios. No lo consiguieron y FirstPerson cerró en la primavera de 1994.

Pese a lo que parecía ya un olvido definitivo, Bill Joy, cofundador de Sun y uno de los desarrolladores principales del Unix de Berkeley, juzgó que Internet podría llegar a ser el terreno adecuado para disputar a Microsoft su primacía casi absoluta en el terreno del software, y vio en Oak el instrumento idóneo para llevar a cabo estos planes. Tras un cambio de nombre y modificaciones de diseño, el lenguaje Java fue presentado en sociedad en agosto de 1995.

2. Programación en Java

En donde la red sea algo crítico, como suele suceder en aplicaciones para empresas, Java le facilita tremendamente la vida al programador. Además, con un entorno de Java para cada una de las plataformas presentes en una empresa y una buena librería de clases ("packages" en Java), los programadores pueden encontrar muy interesante trabajar con este lenguaje.

Las aplicaciones que se crean en grandes empresas deben ser más efectivas que eficientes; es decir, conseguir que el programa funcione y el trabajo salga adelante es más importante que hacerlo eficientemente (aunque no tampoco hay que olvidar esta faceta). Al ser un lenguaje más simple, Java permite a los programadores concentrarse en la mecánica de la aplicación, en vez de pasarse horas y horas incorporando APIs para el control de las ventanas, controlando minuciosamente la memoria, etc.

Muchas de las implementaciones de Java actuales son puros intérpretes. Los byte-codes son interpretados por el sistema run-time de Java, la Máquina Virtual Java (JVM), sobre el ordenador del usuario. Aunque ya hay ciertos proveedores que ofrecen compiladores nativos Just-In-Time (JIT). Si la Máquina Virtual Java dispone de un compilador instalado, las secciones (clases) del byte-code de la aplicación se compilarán hacia la arquitectura nativa del ordenador del usuario. Los programas Java en ese momento rivalizarán con el rendimiento de programas en C++. Los compiladores JIT no se utilizan en la forma tradicional de un compilador; los programadores no compilan y distribuyen binarios Java a los usuarios. La compilación JIT tiene lugar a partir del byte-code Java, en el sistema del usuario, como una parte (opcional) del entorno run-time local de Java.

2.1 Internet & Java

Hasta ahora, la única forma de realizar una página web con contenido interactivo, era mediante la interfaz CGI (Common Gateway Interface), que permite pasar parámetros entre formularios definidos en lenguaje HTML y programas escritos en Perl o en C. Esta interfaz resulta muy incómoda de programar y es pobre en sus posibilidades.

El lenguaje Java y los navegadores con soporte Java, proporcionan una forma diferente de hacer que ese navegador sea capaz de ejecutar programas. Utilizando Java, se pueden eliminar los inconvenientes de la interfaz CGI y también se pueden añadir aplicaciones que vayan desde experimentos científicos interactivos de propósito educativo a juegos o aplicaciones especializadas para la televenta. Es posible implementar publicidad interactiva y periódicos personalizados. Además, con Java podemos estar seguros de que el código no contiene ningún trozo de código malicioso que dañe al sistema. El código que intente actuar destructivamente o que contenga errores, no podrá traspasar las medidas de seguridad de Java.

2.2 Características de Java

✓ Simplicidad

Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos: punteros, macros (#*define*), referencias y liberación de memoria dinámica (*free*). C++ adolece de falta de seguridad, pero C y C++ son de los lenguajes más difundidos, por ello Java se diseñó para ser parecido a C++ (se facilita el aprendizaje) y reducir los errores más comunes de programación en lenguajes como C.

Java elimina muchas de las características de otros lenguajes como C++ para mantener reducidas las especificaciones del lenguaje. Destaca el *garbage collector* ("recolector de basura"): no es necesario preocuparse de liberar memoria.

✓ Orientación a objetos

Java soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.

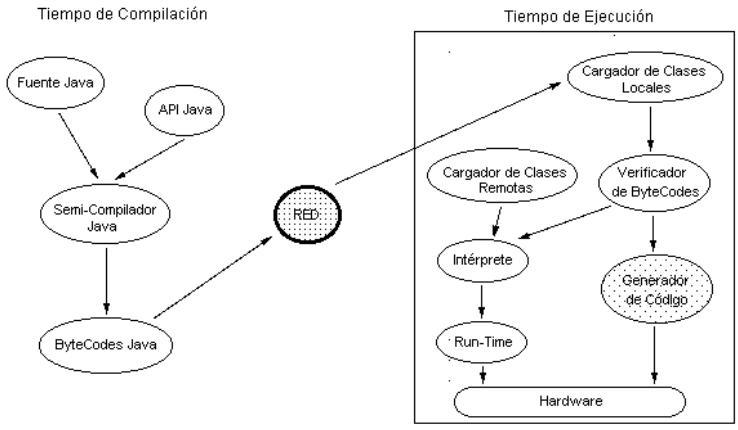
Java incorpora elementos inexistentes en C++ como por ejemplo, la resolución dinámica de métodos. Esta característica deriva del lenguaje Objective C, propietario del sistema operativo Next. En C++ se suele trabajar con librerías dinámicas (DLLs) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (RunTime Type Identification) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en el runtime que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda.

✓ Robustez

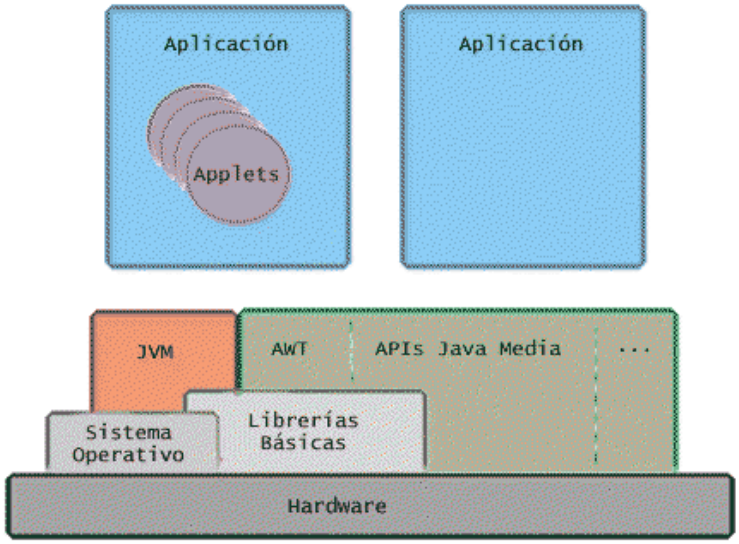
Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. Durante la compilación la comprobación de tipos en Java ayuda a detectar errores lo antes posible. Además, Java obliga la declaración explícita de métodos y maneja automáticamente la liberación de la memoria dinámica (*garbage collector*). Por otra parte, para asegurar el funcionamiento de la aplicación, realiza una verificación de los byte-codes (el resultado de la compilación de un programa Java: código de máquina virtual que es interpretado por el intérprete Java). Java proporciona comprobación de punteros, comprobación de límites de arrays, manejo de excepciones y verificación de byte-codes.

✓ Independencia de la arquitectura

El compilador de Java genera código de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado (Solaris 2.x, SunOs 4.1.x, Windows 95, Windows NT, Linux, Irix, Aix, Mac...).



El código fuente Java se "compila" a un código independiente de la máquina. Este byte-code está diseñado para ejecutarse en una máquina hipotética que es emuilada por el sistema run-time, que sí es dependiente de la máquina.



Lo único dependiente del sistema es la Máquina Virtual Java (JVM) y las librerías fundamentales, que nos permiten acceder directamente al hardware de la máquina. Además, habrá APIs de Java que también entren en contacto directo con el hardware y serán dependientes de la máquina, como ejemplo de este tipo de APIs podemos citar: Java 2D (gráficos 2D), Java Media Framework (audio, video...), Java Animation (animación de objetos en 2D), Java Telephony (telefonía), Java Share (interacción entre aplicaciones multiusuario) o Java 3D (gráficos 3D).

✓ Seguridad

La seguridad en Java tiene dos facetas: en el lenguaje (características como los punteros o el casting implícito que permiten los compiladores de C y C++ se eliminan para prevenir el acceso ilegal a la memoria) y cuando se usa en un navegador (se añaden las protecciones lógicas).

C tiene algunas lagunas de seguridad importantes que intenta solucionar Java. El código Java se pasa, antes de ejecutarse, a través de un verificador de byte-codes que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal. Si los byte-codes pasan la verificación sin generar ningún mensaje de error, entonces: el código no produce desbordamiento de operandos en la pila, el tipo de los parámetros de todos los códigos de operación son conocidos y correctos, no ha ocurrido ninguna conversión ilegal de datos, el acceso a los campos de un objeto es legal y no hay indicios de intentos de violación de las reglas de acceso y seguridad establecidas

El *Cargador de Clases* separa el espacio de nombres del sistema de ficheros local, del de los recursos procedentes de la red. Esto evita caballos de Troya (ya que las clases se buscan primero entre las locales y luego entre las procedentes del exterior), imposibilita que una clase suplante a una predefinida.

Java, para evitar modificaciones en el código, utiliza un método de autentificación mediante clave pública. El *Cargador de Clases* puede verificar una firma digital antes de crear una instancia de un objeto. Ningún objeto se crea sin que se validen los privilegios de acceso. La seguridad se integra en la compilación, con el nivel de detalle que sea necesario.

Java imposibilita abrir ningún fichero de la máquina local (siempre que se realizan operaciones con archivos, éstos residen en el disco duro de la máquina de donde partió el applet), no permite ejecutar ninguna aplicación nativa e impide que se utilicen otros ordenadores como puente, es decir, nadie puede utilizar nuestra máquina para hacer peticiones o realizar operaciones con otra. Además, los intérpretes que incorporan los navegadores de la Web son aún más restrictivos. Bajo estas condiciones se puede considerar que Java es un lenguaje seguro.

Respecto a la seguridad del código fuente, no ya del lenguaje, el JDK [*Java Development Kit*] de Sun proporciona un desemsamblador de byte-code, que permite que cualquier programa pueda ser convertido a código fuente. Utilizando *javap* no se obtiene el código fuente original, pero casi. La protección de los programadores ante esto es utilizar llamadas a programas nativos, externos (en C o C++), aunque se pierda portabilidad.

✓ Portabilidad

Más allá de la portabilidad consecuencia de ser de arquitectura independiente, Java fija estándares para facilitar la portabilidad: los enteros son siempre de 32 bits en complemento a 2, las interfaces de usuario se construyen a través de un sistema abstracto de ventanas (de forma que puedan ejecutarse en entornos Unix, PC o Mac)...

✓ **Lenguaje interpretado**

El intérprete Java (sistema run-time) ejecuta directamente el código objeto (byte-code) que genera el compilador de Java. Por ahora, Java es más lento que otros lenguajes de programación, como C++, ya que debe ser interpretado. La verdad es que Java, para conseguir ser un lenguaje independiente del sistema operativo y del procesador que incorpore la máquina utilizada, tiene que ser así. El código intermedio es de muy bajo nivel (pero sin alcanzar las instrucciones máquina propias de cada plataforma) y no tiene nada que ver con el p-code de Visual Basic. Con este sistema es fácil crear aplicaciones multiplataforma, pero para ejecutarlas es necesario que exista el run-time correspondiente a cada sistema.

✓ **Enlace dinámico**

Java no intenta enlazar 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).

✓ **Concurrencia**

Java permite hebras (threads) y, al estar incorporadas en el lenguaje, son más fáciles de usar que en C o C++.

✓ **Uso en sistemas distribuidos**

La verdad es que Java en sí no es distribuido, sino que proporciona las librerías y herramientas para que los programas puedan ser distribuidos. Los programadores pueden acceder a la información a través de la red con tanta facilidad como a los ficheros locales (interconexión TCP/IP).

El desarrollo de sistemas cliente/servidor en este lenguaje es mucho más sencillo en este lenguaje que en otros (como, por ejemplo, C). Especialmente importante para este tipo de aplicaciones es la portabilidad del código generado (bytecodes), así como las facilidades que el conjunto de clases estándar (vg: el paquete *java.awt* [*Abstract Window Toolkit*]) ofrecen al programador para el desarrollo rápido de aplicaciones [*RAD: Rapid Application Development*].

Otro aspecto interesante de Java es la posibilidad de utilizar CORBA [*Common Object Request Broker Architecture*] como sustituto de la combinación HTTP/CGI [*HyperText Transfer Protocol / Common Gateway Interface*], con lo que se consigue mayor flexibilidad, escalabilidad y un rendimiento superior.

2.3 Rendimiento

No hay que olvidar que Java es lento, realmente lento. Su lentitud se debe en gran parte a su nivel de abstracción superior al empleado en otros lenguajes: la interpretación de los bytecodes, la máquina virtual de tipo pila, el paquete *java.awt* [*Abstract Window Toolkit*]...

Esta característica ha dado lugar a la aparición de los *JIT* [*Just-In-time Compilers*], en realidad más parecidos a ensambladores que a compiladores. Estos programas convierten el código intermedio portable de Java en código objeto para una máquina concreta, con lo que se consiguen tiempos de ejecución más parecidos a los obtenidos con otros lenguajes, aunque siguen siendo peores que los logrados por compiladores de C/C++.

2.4 ¿Originalidad?

Java se ha convertido en muy poco tiempo en un lenguaje de programación muy popular (de hecho, en algunas universidades ya se utiliza en primero de carrera). En cierto modo su éxito es sorprendente, ya que los programadores experimentados suelen ser reacios a cambiar de lenguaje de programación (en muchos sitios se sigue empleando FORTRAN o COBOL).

El hecho de que un cambio así no se haya producido antes no quiere decir que no se hayan propuesto lenguajes alternativos (con nombres tan llamativos como ALLOY, COOL, PIZZA o SELF). En realidad, muchos de esos lenguajes tenían algunas de las características de Java (orientación a objetos, hebras, independencia de la arquitectura, etc.).

La característica más destacable de Java, su portabilidad, tampoco es ninguna novedad. Quince años antes de su aparición, la misma idea se empleó en el UCSD Pascal (una versión estándar de Pascal desarrollada en la Universidad de California en San Diego. UCSD Pascal es respecto a Pascal más o menos lo que Java supone respecto a C++. El UCSD Pascal generaba código intemedio independiente de la máquina (¿le suena?) que era interpretado por un intérprete que dependía de la máquina (¿le es familiar?). Sin embargo, UCSD Pascal fue un éxito técnico pero un fracaso de mercado: se convirtió de proyecto universitario en propiedad de una empresa y, además, su lentitud (¿sorprendido?) dificultó su venta. Cuando se construyó una máquina que interpretaba el código intermedio (¿déjà vu?) el interés en el proyecto se había desvanecido.

3. El lenguaje de programación JAVA

☞ Comentarios

En Java hay tres tipos de comentarios:

```
// Comentarios para una sola línea

/* Comentarios de una o más líneas */

/** Comentario de documentación, de una o más líneas */
```

Los dos primeros tipos de comentarios son los mismos que en C++. Los comentarios de documentación, colocados inmediatamente antes de una declaración (de variable o función), indican que ese comentario ha de ser colocado en la documentación que se genera automáticamente cuando se utiliza la herramienta de Java, javadoc. Dichos comentarios sirven como descripción del elemento declarado permitiendo generar la documentación de nuestras clases (en HTML) al mismo tiempo que se genera el código. La documentación del API de Java ha sido creada de este modo (el resultado obtenido no siempre es muy bueno).

☞ Identificadores

Los identificadores nombran variables, funciones, clases y objetos; cualquier cosa que el programador necesite identificar o usar. En Java, un identificador comienza con una letra, un subrayado (_) o un símbolo de dólar ($). Los siguientes caracteres pueden ser letras o dígitos. Se distinguen las mayúsculas de las minúsculas y no hay longitud máxima.

NOTA: Java utiliza el juego de caracteres Unicode, no el típico ASCII.

☞ Palabras clave

Las siguientes son las palabras clave que están definidas en Java y que no se pueden utilizar como indentificadores:

```
abstract    continue    for         new         switch
boolean     default     goto        null        synchronized
break       do          if          package     this
byte        double      implements  private     threadsafe
byvalue     else        import      protected   throw[s]
case        extends     instanceof  public      transient
catch       false       int         return      true
char        final       interface   short       try
class       finally     long        static      void
const       float       native      super       while
```

Además, el lenguaje se reserva unas cuantas palabras más, pero que hasta ahora no tienen un cometido específico:

```
cast        future      generic     inner
operator    outer       rest        var
```

☞ Literales y tipos básicos

Un valor constante en Java se crea utilizando una representación literal de él. Java utiliza cinco tipos básicos: enteros, reales, booleanos, caracteres y cadenas.

Enteros

byte	8 bits	en complemento a dos
short	16 bits	en complemento a dos
int	32 bits	en complemento a dos
long	64 bits	en complemento a dos

vg:	Decimal	21 (int), 21L (long)
	Octal	077
	Hexadecimal	0xDC00

Reales

float	32 bits	IEEE 754
double	64 bits	IEEE 754

vg: 3.14 (double), 3.14f (float), 3.14d (double), 2e12, 3.1E12 ...

Booleanos

```
true
false
```

Caracteres

vg: '\u????' (código UNICODE, en hexadecimal)
'\b' (backspace)
'\t' (tab)
'\n' (line feed)
'\f' (form feed)
'\r' (carriage return)
...

Cadenas

vg: "Esto es una cadena", "" (cadena vacía)...

NOTA: Para comparar cadenas se debe utilizar el método "equals". Los operadores == y != aplicados a objetos nos dicen si sus dos operandos se refieren a la misma instancia del objeto.

☞ Arrays

Se pueden declarar arrays de cualquier tipo, incluso arrays de arrays (como en C):

```
char s[];
int iArray[];
int tabla[][] = new int[4][5];
```

Los límites de los arrays se comprueban en tiempo de ejecución para evitar desbordamientos y la corrupción de la memoria. En Java un array es un objeto (tiene redefinido el operador []) que tiene una función miembro: length. Se puede utilizar este método para conocer la longitud de cualquier array.

```
int a[][] = new int[10][3];
a.length;               /* 10 */
a[0].length;            /*  3 */
```

Para crear un array en Java hay dos métodos básicos: crear un array vacío o crearlo especificando sus valores iniciales:

① `int lista[] = new int[50];`

② `String nombres[] = { "Juan","Pepe","Pedro","Maria" };`

Equivalente a:

```
String nombres[];
nombres = new String[4];
nombres[0] = new String( "Juan" );
nombres[1] = new String( "Pepe" );
nombres[2] = new String( "Pedro" );
nombres[3] = new String( "Maria" );
```

No se pueden crear arrays estáticos en tiempo de compilación. Tampoco se puede rellenar un array sin declarar el tamaño con el operador new.

Todos los arrays en Java son estáticos. Para conseguir arrays dinámicos se usa la clase Vector (java.util), que permite operaciones de inserción, borrado, etc.

☞ Operadores

Los operadores de Java son muy parecidos a los de C. En la siguiente tabla aparecen por orden de precedencia:

```
.     [ ]   ( )
++    --
!     ~     instanceof
new
*     /     %
+     -
<<    >>    >>>
<     >     <=    >=    ==    !=
&     ^     |
&&    ||
? :
=     op=   ,
```

Los operadores numéricos se comportan como en cualquier otro lenguaje. Los operadores relacionales devuelven un valor booleano. Se pueden utilizar los operadores + y += para concatenar cadenas.

✰ `new (tipo) expresión`

Se utiliza para crear instancias de una clase.

✰ `objeto instanceof clase`

Nos indica si un objeto es una instancia de una clase o no.

☞ Separadores

( ) PARÉNTESIS: Listas de parámetros en la definición y llamada a métodos, precedencia en expresiones, expresiones para control de flujo y conversiones de tipo.

{ } LLAVES: Inicialización de arrays, bloques de código, clases, métodos y ámbitos locales.

[ ] CORCHETES: Arrays.

; PUNTO Y COMA: Separador de sentencias.

, COMA: Identificadores consecutivos en una declaración de variables y sentencias encadenadas dentro de una sentencia for.

. PUNTO: Nombres de paquete, subpaquetes y clases; variables y métodos.

☞ Control de flujo

Muchas de las sentencias de control del flujo del programa se han tomado de C/C++

Estructuras condicionales

```
if ( Boolean ) {
   sentencias;
} [ else {
   sentencias;
} ]
```

```
switch ( expr_int ) {
  case valor:
       sentencias;
       [ break; ]
  ...
[ default:
       sentencias;
       [ break; ] ]
}
```

NOTA: Java también permite utilizar el operador condicional **?:** de C.

```
Test ? TrueResult : FalseResult
```

Estructuras repetitivas: Bucles

```
for ( expr1 inicio; expr2 test; expr3 incremento ) {
    sentencias;
}
```

```
while ( Boolean ) {
    sentencias;
}
```

```
do {
    sentencias;
} while ( Boolean );
```

NOTA: Al igual que C, Java permite salirnos de un bucle por las bravas (con la posibilidad de utilizar etiquetas). El lenguaje posee la palabra reservada goto, pero no se permite su uso en realidad (por suerte).

```
break [etiqueta];
continue [etiqueta];

etiqueta: sentencia;
```

Excepciones

```
try {
    // Código
} catch ( Exception ) {
    // Manejo de excepciones
} [ finally {
    // Se ejecuta haya o no excepción
} ]
```

· Las excepciones se pueden lanzar con "**throw** *Excepción*".

· Se indica que un método puede lanzar una excepción con "**throws** *Excepción*" en la declaración del método. Véase java.lang.throwable.

Exclusión mutua

```
synchronized (Object)
{
  ...
}
```

☞ Métodos

Las funciones (métodos en Java ya que siempre deben pertenecer a alguna clase) se declaran y utilizan exactamente igual que las funciones en C. Para devolver el resultado de un método se utiliza la palabra reservada "return":

```
[tipoMétodo] tipoResultado idMétodo ( argumentos ) [throws Throwable]
{
  ...
  [return expr;]
}
```

Paso de parámetros

El paso de objetos como parámetros a los métodos se realiza siempre por referencia (incluyendo arrays). Las variables de tipos primitivos se pasan siempre por valor.

Sobrecarga de métodos

Java permite sobrecargar métodos: definir métodos con el mismo nombre para distinto número y tipo de argumentos, aunque no se permite sobrecargar métodos para distintos tipos de resultado.

☞ Variables

Declaración

Las variables se declaran exactamente igual que en C/C++. Una variable puede ser de cualquier tipo básico, clase o interface.

Ámbito de las variables

Los bloques de sentencias compuestas en Java se delimitan con llaves. Las variables de Java sólo son válidas desde el punto donde están declaradas hasta el final de la sentencia compuesta que la engloba. Se pueden anidar bloques y cada uno puede contener su propio conjunto de declaraciones de variables locales.

☒ No se puede, en teoría, declarar una variable con el mismo nombre que una de ámbito exterior. Al menos, no se debe (el compilador permite hacerlo aunque no es aconsejable).

REFERENCIAS: Las referencias en Java no son punteros ni referencias como en C++. Las referencias en Java son identificadores de instancias de las clases Java.

☞ Clases

Todo en Java forma parte de una clase, es una clase o describe como funciona una clase: todas las acciones de los programas Java se colocan dentro del bloque de una clase o un objeto, todos los métodos se definen dentro del bloque de la clase y Java no soporta funciones o variables globales.

La palabra clave import (equivalente al #include de C) puede colocarse al principio de un fichero, fuera del bloque de la clase. En realidad, el compilador reemplazará esa sentencia con el contenido del fichero que se indique, que consistirá, como es de suponer, en más clases.

Tipos de Clases

`public`	Accesible desde otras clases
`abstract`	Tiene, al menos, un método abstracto. No se instancia, se utiliza como clase base de la que heredan otras.
`final`	Termina una cadena de herencia. No se puede heredar de una clase final (vg: Math)
`synchronizable`	Todos sus métodos están sincronizados: no se puede acceder simultáneamente a dos de ellos desde distintas hebras.

Declaración

```
[tipo] class Nombre [ implements Interfaces ] [ extends SuperClase ]
{
  ...
}
```

Constructores

Cuando se declara una clase en Java, se pueden declarar uno o más constructores (con el mismo nombre de la clase, no devuelven valores de ningún tipo [void]) que realizan la inicialización cuando se instancia un objeto de dicha clase:

```
MiClase mc;
mc = new MiClase();
```

☆ El operador `new` se usa para crear una instancia de la clase:

```
new expresión

new (tipo) expresión
```

✯ Para copiar objetos se pueden utilizar los métodos `copy` y `clone`:

```
objeto_destino.copy (objeto_origen);

objeto_destino = (clase) objeto_origen.clone( );
```

NOTA: `clone` genera un Object, por lo que se hace necesario el casting.

Finalizadores

Java no utiliza destructores (al estilo de C++). No obstante, proporciona un método que, cuando se especifique en el código de la clase, el reciclador de memoria (garbage collector) llamará:

```
protected void finalize()
{
}
```

NOTA: En Java, la recolección y liberación de memoria es responsabilidad de un thread llamado automatic garbage collector (recolector automático de basura). Llamar manualmente al finalizador no implica que se libere la memoria ocupada por el objeto: la memoria se liberará cuando dejen de existir referencias al objeto.

Determinación de la clase a la que pertenece un objeto

① `String id = objeto.getClass( ).getName( );`

② `boolean x = objeto instanceof clase;`

Control de acceso

Cuando se crea una nueva clase en Java, se puede especificar el nivel de acceso que se quiere para las variables de instancia y los métodos de la clase: `public` (cualquier clase desde cualquier lugar puede acceder), `protected` (sólo las subclases pueden acceder [y las clases del mismo paquete]), `private` (sólo se puede acceder desde dentro de la clase) y `friendly` (se puede acceder desde cualquier clase del paquete).

Por defecto, si no se especifica lo contrario, las variables y métodos de instancia se declaran *friendly*, lo que significa que son accesibles por todos los objetos dentro del mismo paquete, pero no por los externos al paquete. Los métodos protegidos (*protected*) pueden ser vistos por las clases derivadas, como en C++, y en Java, también por las de su paquete.

☞ Herencia

```
class subclase extends superclase
{
}
```

La palabra clave extends se usa para generar una subclase. Se heredan los métodos y variables de instancia de la superclase. Además, se pueden sustituir los métodos proporcionados por la clase base. En Java no se permite herencia múltiple.

☞ final

La única forma de especificar constantes en Java es utilizar la palabra reservada final, ya que el lenguaje no incluye ni el #define ni el const de C/C++. No se pueden declarar variables locales con final (sólo variables de instancia o variables estáticas).

La palabra reservada "final" sirve también para especificar que una clase no puede tener subclases o para indicar que un método no puede ser redefinido en subclases.

☞ Variables y métodos estáticos (*class variables&methods*): static

En un momento determinado se puede querer crear una clase en la que el valor de una variable de instancia sea el mismo (y de hecho sea la misma variable) para todos los objetos instanciados a partir de esa clase, es decir, que exista una única copia de la variable de instancia. Se usará para ello la palabra clave static. Si se cambia su valor, cambia para todos los objetos de la clase.

De la misma forma se puede declarar un método como estático, lo que impide que el método pueda acceder a las variables de instancia no estáticas.

☞ Clases abstractas: abstract

Cuando una clase contiene un método abstracto tiene que declararse abstracta. No obstante, no todos los métodos de una clase abstracta tienen que ser abstractos. Las clases abstractas no pueden tener métodos privados ni tampoco estáticos. Una clase abstracta tiene que derivarse obligatoriamente (no se puede hacer un new de una clase abstracta), equivale a una clase con funciones virtuales en C++.

☞ this

✔ Al acceder a variables de instancia de una clase, la palabra clave "this" hace referencia a los miembros de la instancia concreta.

```
this.método (argumentos);
```

✔ Dentro de un constructor se puede llamar a otro constructor de la clase usando "this":

```
this (argumentos);
```

☞ super

✔ Si se necesita llamar al método padre dentro de una clase que ha reemplazado ese método, se puede hacer referencia al método padre escribiendo:

```
super.método (argumentos);
```

✔ También se puede utilizar "super" para llamar al constructor del padre:

```
super (argumentos);
```

☞ Interfaces

Un interface contiene una colección de métodos que se implementan en otro lugar. La principal diferencia entre interfaces y clases abstractas es que un interface proporciona un mecanismo de encapsulación de los protocolos de los métodos sin tener que usar herencia:

```
public interface clase_interface
{
  ...       // Declaraciones
}


class clase_concreta implements clase_interface1 [,clase_interface2]...
{
  ...       // Código
}
```

☞ Métodos nativos

Permiten la llamada a funciones C y C++ desde nuestro código fuente Java. Para definir métodos como funciones C o C++ se utiliza la palabra clave native.

```
public class Fecha
{
  int ahora;

  public Fecha()
  {
     ahora = time();
  }

  private native int time();

  static
  {
    System.loadLibrary( "time" );
  }
}
```

Una vez escrito el código Java, se necesita utilizar *javah -stubs* para crear un fichero de cabecera (.h) y un fichero de stubs (que contiene la declaración de las funciones), escribir el código del método nativo en C o C++, compilar el fichero de stubs y el fichero .c en una librería de enlace dinámico (DLL en Windows o libXX.so en Unix).

☞ Paquetes

La palabra clave package permite agrupar clases e interfaces. Los nombres de los paquetes son palabras separadas por puntos y se almacenan en directorios que coinciden con esos nombres.

Los paquetes de clases se cargan con la palabra clave import, especificando el nombre del paquete como ruta y nombre de clase (es lo mismo que #include de C/C++). Se pueden cargar varias clases utilizando un asterisco. Si un fichero fuente Java no contiene ningún package, se coloca en el mismo directorio que el fichero fuente, y la clase puede ser cargada con la sentencia import.

El lenguaje Java proporciona una serie de paquetes que incluyen ventanas, utilidades, un sistema de entrada/salida general, herramientas y comunicaciones:

java.applet

Este paquete contiene clases diseñadas para usar applets. Hay una clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.

java.awt

El paquete Abstract Windowing Toolkit (awt) contiene clases para implementar la interfaz gráfica de usuario (GUI) y manejar imágenes (java.awt.Image). Incluye, entre otras, las clases Button, Checkbox, Choice, Component, Font, Graphics, Menu, Panel, TextArea, TextField y Window.

java.io

El paquete de entrada/salida contiene las clases de acceso a ficheros (streams): FileInputStream y FileOutputStream.

java.lang

Este paquete incluye las clases del lenguaje Java propiamente dicho: *Object*, *Thread*, *Exception*, System, Integer, Float, Math, String, etc.

java.net

Este paquete da soporte a las conexiones del protocolo TCP/IP y, además, incluye las clases Socket, URL y URLConnection.

java.sql

Permite acceder a bases de datos a través de JDBC (Java DataBase Connectivity).

java.util

Este paquete es una miscelánea de clases útiles para muchas cosas en programación. Se incluyen, entre otras: Date (fecha), Dictionary (diccionario), Random (números aleatorios), Stack (pila FIFO), Vector y Hashtable.

☞ Aplicaciones y applets

Una aplicación en Java consiste en un conjunto de clases. Se debe definir un método que determine el punto de arranque del programa (main). Este método debe declararse public y static, tal como se muestra en el ejemplo.

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

La aplicación "Hola mundo"

Un applet permite ejecutar código Java desde un navegador WWW. El ejemplo siguiente muestra cómo se puede construir uno:

Fichero JAVA:

```
import java.awt.Graphics;

public class HelloWorld extends java.applet.Applet
{
  public void paint (Graphics g)
  {
    g.drawString("Hello world!", 5,25);
  }
}
```

Fichero HTML:

```
<HTML>
<HEAD>
<TITLE>
Applet básico en Java
</TITLE>
</HEAD>
<BODY>
<APPLET CODE="HelloWorld.class" WIDTH=150 HEIGHT=25>
</APPLET>
</BODY>
</HTML>
```

El applet "Hola mundo"

4. Bibliografía

FROUFE, Agustín
"Tutorial de Java en Español"
Sevilla, 1996.

Este tutorial de Java se encuentra en formato HTML y puede encontrarse en Internet. Sirve de introducción para el que no sepa programar en Java e incluye algunos ejemplos interesantes con su código correspondiente (aunque éste corresponde a la versión 1.0 del JDK).

GLASS, Robert L.
"Everything old is new again"
Practical Programmer
Communications of the ACM, May 1998

En este curioso artículo, se revisan algunas de las supuestas novedades (como Java, el NC [*Network Computer*] o la situación de Microsoft) que no lo son tanto. Ideas que ahora tienen éxito en ocasiones anteriores no lo tuvieron. Por ejemplo, Java y el UCSD Pascal.

"JDK 1.x.x Documentation"
JavaSoft.

Documentación del JDK (Java Development Kit) que incluye ayuda de los paquetes incluidos en cada una de las versiones desarrolladas por Sun, incluyendo información acerca del paquete **java.sql** que permite acceder a bases de datos programando en Java a través de JDBC.

LEMAY, Laura & PERKINS, Charles L.
"Teach yourself JAVA in 21 days"
USA: Sams.net Publishing, 1996. [1ª edición].

Libro, ya clásico, para aprender Java. Existe una versión en español, pero la traducción no es muy buena. El libro se escribió cuando el API de Java estaba aún en versión beta, por lo que algunas cosas han cambiado (como los applets), pero el lenguaje sigue siendo el mismo. No obstante, los navegadores de Internet aún utilizan el API descrito en el libro (la versión 1.0), por lo que puede valer si lo único que nos interesa es construir programas que puedan ejecutarse desde una página HTML.

LEWANDOSWSKI, Scott M.
"Frameworks for Component-Based Client/Server Computing"
ACM Computing Surveys, March 1998

Este artículo, realizado por un estudiante de la Brown University (Providence, RI), revisa distintos estándares para el desarrollo de aplicaciones distribuidas tales como CORBA (abierto), COM (de Microsoft). Así mismo, trata del uso de Java para el desarrollo de aplicaciones cliente/servidor y de su integración con CORBA.

"Oracle Lite JAVA Developer's Guide"
Oracle Corporation, 1997.

En él se comenta (sin demasiados detalles) cómo trabajar con Oracle en Java: procedimientos, triggers, JDBC (Java Database Connectivity) y JAC (Java Access Classes).

RADA, Roy
"Corporate Shortcut to Standarization"
Communications of the ACM, January 1998

Artículo en el que se comenta toda la burocracia que rodea a la creación de un estándar, proceso que ha de acelerarse en temas de informática (donde de lo que hoy está de moda mañana nadie se acordará). Como ejemplo se expone la "estandarización" del lenguaje Java promovida por Sun (y no muy respetada por Microsoft).

SINGHAL, Sandeep & NGUYEN, Bihn, eds.
"The Java Factor"
Communications of the ACM, June 1998

Una sección especial de la publicación más importante de la ACM dedicada al ascenso meteórico del lenguaje de programación Java, ascenso ligado a la creciente popularidad de la WWW [*World Wide Web*]. En ella se discuten algunas carencias del lenguaje, como su lentitud o su poca idoneidad para el desarrollo de aplicaciones en tiempo real.