Lenguaje de Programación Orientada a Objetos

A partir de este punto, habiendo analizado las características del paradigma orientado a objetos y el lenguaje estándar que se utiliza para modelar (UML), estudiaremos el lenguaje de programación Java, que está basado en el paradigma orientado a objetos que estudiamos anteriormente.

Surgimiento del Lenguaje

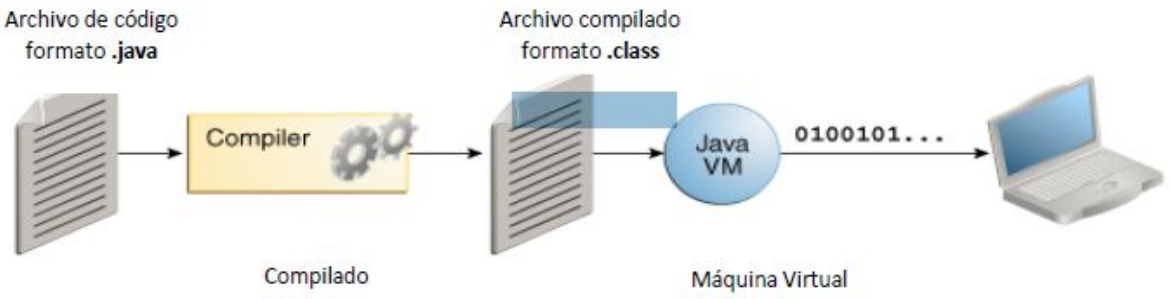
Java surgió en 1991 cuando un grupo de ingenieros de la empresa Sun Microsystems trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Es por esto que desarrollan un código “neutro” que no depende del tipo de electrodoméstico, el cual se ejecuta sobre una “máquina hipotética o virtual” denominada Java Virtual Machine (JVM). Es la JVM quien interpreta el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permitía lo que luego se ha convertido en el principal lema del lenguaje: “Write Once, Run Everywhere”. A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje. Java, como lenguaje de programación para computadoras, se introdujo a finales de 1995. Si bien su uso se destaca en la Web, sirve para crear todo tipo de aplicaciones (locales, intranet o internet).

Entendiendo Java

Java es un lenguaje compilado e interpretado; esto significa que el programador escribirá líneas de código utilizando un determinado programa que se denomina IDE (Ambiente de Desarrollo Integrado; Integrated Development Environment por sus siglas en inglés). Este programa le provee al programador un espacio de trabajo para crear código que se almacena en archivos de formato .java.
Luego, para poder correr el código y ejecutar el programa creado, se debe realizar primero lo que se denomina Compilación; para eso un programa denominado compilador será el encargado de revisar la sintaxis del código y si está correcto transformará el archivo .java en otro archivo de formato .class que se denominan: “bytecodes”. Una vez realizada la compilación, será la Máquina Virtual (Java Virtual Machine - JVM), la encargada de leer los archivos .class y transformarlos en el código entendido por la CPU de la computadora.

Fig. -Funcionamiento general del Lenguaje Java



El entorno de desarrollo de Java

Existen distintos programas comerciales que permiten desarrollar código Java. Oracle, quien compró a Sun (la creadora de Java), distribuye gratuitamente el Java Development Kit (JDK). Se trata de un conjunto de programas y librerías que permiten desarrollar, compilar y ejecutar programas en Java.
Incorpora además la posibilidad de ejecutar parcialmente el programa, deteniendo la ejecución en el punto deseado y estudiando en cada momento el valor de cada una de las variables (es el denominado Debugger). Es tarea muy común de un programador realizar validaciones y pruebas del código que construye, el debugger permite realizar una prueba de escritorio automatizada del código, lo cual ayuda a la detección y corrección de errores. En el momento de escribir este trabajo las herramientas de desarrollo: JDK, van por la versión 1.8. Estas herramientas se pueden descargar gratuitamente de http://www.oracle.com/technetwork/java.
Los IDEs (Integrated Development Environment), tal y como su nombre indica, son entornos de desarrollo integrados. En un mismo programa es posible escribir el código Java, compilarlo y ejecutarlo sin tener que cambiar de aplicación. Algunos incluyen una herramienta para realizar Debug gráficamente, frente a la versión que incorpora el JDK basada en la utilización de una Consola bastante difícil y pesada de utilizar. Estos entornos integrados permiten desarrollar las aplicaciones de forma mucho más rápida, incorporando en muchos casos librerías con componentes ya desarrollados, los cuales se incorporan al proyecto o programa. Estas herramientas brindan una interfaz gráfica para facilitar y agilizar el proceso de escritura de los programas. El entorno elegido para este módulo, por su relevancia en el mercado y su licencia gratuita se denomina Netbeans. El mismo se encuentra disponible para descargar de forma gratuita en www.netbeans.org. En la Guía Práctica abordaremos la forma de instalación y configuración de estas herramientas.

El compilador de Java

Se trata de una de las herramientas de desarrollo incluidas en el JDK. Realiza un análisis de sintaxis del código escrito en los ficheros fuente de Java (con extensión *.java). Si no encuentra errores en el código genera los ficheros compilados (con extensión *.class). En otro caso muestra la línea o líneas erróneas. En el JDK de Sun dicho compilador se llama javac.exe si es para el sistema operativo Windows.

La Java Virtual Machine

Tal y como se ha comentado al comienzo, la existencia de distintos tipos de procesadores y ordenadores llevó a los ingenieros de Sun a la conclusión de que era muy importante conseguir un software que no dependiera del tipo de procesador utilizado. Se plantea la necesidad de conseguir un código capaz de ejecutarse en cualquier tipo de máquina. Una vez compilado no debería ser necesaria ninguna modificación por el hecho de cambiar de procesador o de ejecutarlo en otra máquina. La clave consistió en desarrollar un código "neutro" el cual estuviera preparado para ser ejecutado sobre una "máquina hipotética o virtual", denominada Java Virtual Machine (JVM). Es esta JVM quien interpreta este código neutro convirtiéndolo a código particular de la CPU o chip utilizada. Se evita tener que realizar un programa diferente para cada CPU o plataforma.
La JVM es el intérprete de Java. Ejecuta los "bytecodes" (ficheros compilados con extensión *.class) creados por el compilador de Java (javac.exe). Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado JIT (Just-In-Time Compiler), que puede mejorar entre 10 y 20 veces la velocidad de ejecución de un programa.

Características de Java

Java es un lenguaje:

- **Orientado a objetos:** Al contrario de otros lenguajes como C++, Java no es un lenguaje modificado para poder trabajar con objetos, sino que es un lenguaje creado originalmente para trabajar con objetos. De hecho, todo lo que hay en Java son objetos.
- **Independiente de la plataforma:** Debido a que existen máquinas virtuales para diversas plataformas de hardware, el mismo código Java puede funcionar prácticamente en cualquier dispositivo para el que exista una JVM.
- **Compilado e Interpretado:** La principal característica de Java es la de ser un lenguaje compilado e interpretado. Todo programa en Java ha de compilarse y el código que se genera bytecode es interpretado por una máquina virtual, como se vio anteriormente. De este modo se consigue la independencia de la máquina: el código compilado se ejecuta en máquinas virtuales que sí son dependientes de la plataforma. Para cada sistema operativo distintos, ya sea Microsoft Windows, Linux, OS X, existe una máquina virtual específica que permite que el mismo programa Java pueda funcionar sin necesidad de ser recompilado.
- **Robusto:** Su diseño contempla el manejo de errores a través del mecanismo de Excepciones y fuerza al desarrollador a considerar casos de mal funcionamiento para reaccionar ante las fallas.
- **Gestiona la memoria automáticamente:** La máquina virtual de Java gestiona la memoria dinámicamente como veremos más adelante. Existe un recolector de basura que se encarga de liberar la memoria ocupada por los objetos que ya no están siendo utilizados.
- **No permite el uso de técnicas de programación inadecuadas:** Como veremos más adelante, todo en Java se trata de objetos y clases, por lo que para crear un programa es necesario aplicar correctamente el paradigma de objetos.
- **Multihilos (multithreading):** Soporta la creación de partes de código que podrán ser ejecutadas de forma paralela y comunicarse entre sí.

- **Cliente-servidor:** Java permite la creación de aplicaciones que pueden funcionar tanto como clientes como servidores. Además, provee bibliotecas que permiten la comunicación, el consumo y el envío de datos entre los clientes y servidores.
- **Con mecanismos de seguridad incorporados:** Java posee mecanismos para garantizar la seguridad durante la ejecución comprobando, antes de ejecutar código, que este no viola ninguna restricción de seguridad del sistema donde se va a ejecutar. Además, posee un gestor de seguridad con el que puede restringir el acceso a los recursos del sistema.
- **Con herramientas de documentación incorporadas:** Como veremos más adelante, Java contempla la creación automática de documentación asociada al código mediante la herramienta **Javadoc**.

Aplicaciones de java

Java es la base para prácticamente todos los tipos de aplicaciones de red, además del estándar global para desarrollar y distribuir aplicaciones móviles y embebidas, juegos, contenido basado en web y software de empresa. Java8 se encuentra aplicado en un amplio rango de dispositivos desde portátiles hasta centros de datos, desde consolas para juegos hasta súper computadoras, desde teléfonos móviles hasta Internet.

- El 97% de los escritorios empresariales ejecutan Java.
- Es la primera plataforma de desarrollo.
- 3 mil millones de teléfonos móviles ejecutan Java.
- El 100% de los reproductores de Blu-ray incluyen Java.
- 5 mil millones de Java Cards en uso.
- 125 millones de dispositivos de televisión ejecutan Java.

Antes de comenzar

Como se detalló en el apartado “Entendiendo Java”, ***los archivos de código escritos por el programador que se denominan: archivos de código fuente, en Java son los archivos*** `.java`. Éstos se podrían crear utilizando simplemente un editor de texto y guardándolo con la extensión `.java`. Por lo tanto, esto se puede hacer utilizando cualquier editor de texto como el bloc de notas de Windows.

Para facilitar la tarea de desarrollo en la práctica, como vimos, se utilizan IDEs (entornos de desarrollo integrados) que ofrecen herramientas como coloreado de palabras clave, análisis de sintaxis en tiempo real, compilador integrado y muchas otras funciones que usaremos.

Al ser un lenguaje multiplataforma y especialmente pensado para su integración en redes, la codificación de texto utiliza el **estándar Unicode**, lo que implica que los programadores y programadoras hispanohablantes podemos utilizar sin problemas símbolos de nuestra lengua como la letra “ñ” o las vocales con tildes o diéresis a la hora de poner nombre a nuestras variables.

Algunos detalles importantes son:

● En java (como en C) hay diferencia entre mayúsculas y minúsculas por lo que la variable nombrecompleto es diferente a nombreCompleto.

● Cada línea de código debe terminar con un; (punto y coma)

● Una instrucción puede abarcar más de una línea. Además, se pueden dejar espacios y tabuladores a la izquierda e incluso en el interior de la instrucción para separar elementos de la misma.
● A veces se marcan bloques de código, es decir código agrupado. Cada bloque comienza con llave que abre, "{" y termina con llave que cierra, "}"

TIPOS PRIMITIVOS DE DATO

Todo lenguaje de programación consta de elementos específicos que permiten realizar las operaciones básicas de la programación: tipos de datos, operadores e instrucciones o sentencias. En este apartado se introducen los distintos tipos de dato que pueden emplearse en la programación con Java. En concreto, se presentan los tipos primitivos en Java, así como las constantes y las variables. En los capítulos sucesivos se mostrarán el resto de elementos básicos de programación, incluyendo otras estructuras de datos más complejas.

CATEGORÍAS DE TIPOS DE DATOS

Los tipos de datos utilizados en Java se pueden clasificar según diferentes categorías:
De acuerdo con el tipo de información que representan. Esta correspondencia determina los valores que un dato puede tomar y las operaciones que se pueden realizar con él. Según este punto de vista, pueden clasificarse en:

- Datos de tipo **primitivo**: representan un único dato simple que puede ser de tipo char, byte, short, int, long, float, double, boolean. Por ejemplo: 'a', 12345, 750.68, False,... Cada tipo de dato presenta un conjunto de valores o constantes literales.
- Variables *referencia* (variables *arrays*, de una clase/instancias, interfaces . ..). Se implementan mediante un nombre o referencia (puntero) que contiene la dirección en memoria de un valor o conjunto de valores (objeto creado con new).

Según cambie su valor o no durante la ejecución del programa. En este caso, se tienen:

- **Variables**: sirven para almacenar datos durante la ejecución del programa; el valor asociado puede cambiar varias veces durante la ejecución del programa.
- **Constantes o variables finales**: también sirven para almacenar datos pero una vez asignado el valor, éste no puede modificarse posteriormente.

Según su papel en el programa. Pueden ser:

- Variables *miembro* de una clase. Se definen dentro de una clase, fuera de los métodos. Pueden ser de tipos primitivos o referencias y también variables o constantes.
- Variables *locales*. Se definen dentro de un método o, en general, dentro de cualquier bloque de sentencias entre llaves {}. La variable desaparece una vez finalizada la ejecución del método o del bloque de sentencias. También pueden ser de tipos primitivos o referencias.

A continuación se describen cada uno de estos tipos de dato.

TIPOS DE DATO PRIMITIVOS EN JAVA

A todo dato (constante, variable o expresión) le corresponde un **tipo** específico en Java. Como se ha indicado anteriormente un tipo de dato determina los valores que pueden asignarse a un dato, el formato de representación correspondiente y las operaciones que pueden realizarse con dicho dato.

En Java casi todo es un objeto. Existen algunas excepciones como, por ejemplo, los tipos primitivos, tales como `int`, `char`, etc., que no se consideran objetos y se tratan de forma especial. Java tiene un conjunto de tipos *primitivos* para representar datos enteros (cuatro tipos diferentes), para datos numéricos reales en coma flotante (dos tipos diferentes), para caracteres y para datos lógicos o booleanos. Cada uno de ellos tiene **idéntico** tamaño y comportamiento en todas las versiones de Java y para cualquier tipo de ordenador. Esto implica que no hay directivas de compilación condicionales y asegura la **portabilidad** de los programas a diferencia de lo que ocurre, por ejemplo, con el lenguaje de programación C.

Por otro lado, a partir de estos tipos primitivos de dato pueden construirse otros tipos de datos compuestos, *arrays*, clases e interfaces. En la siguiente tabla se muestran los tipos de dato primitivos de Java con el intervalo de representación de valores que puede tomar y el tamaño en memoria correspondiente.

Tipo	Representación / Valor	Tamaño (en bits)	Valor mínimo	Valor máximo	Valor por defecto
`boolean`	`true` o `false`	1	N.A.	N.A.	`false`
`char`	Carácter Unicode	16	`\u0000`	`\uFFFF`	`\u0000`
`byte`	Entero con signo	8	-128	128	0
`short`	Entero con signo	16	-32768	32767	0
`int`	Entero con signo	32	-2147483648	2147483647	0
`long`	Entero con signo	64	-9223372036854775808	9223372036854775807	0
`float`	Coma flotante de precisión simple Norma IEEE 754	32	±3.40282347E+38	±1.40239846E-45	0.0
`double`	Coma flotante de precisión doble Norma IEEE 754	64	±1.79769313486231570E+308	±4.94065645841246544E-324	0.0

Figura 3.1 Tipos de dato primitivos de Java

Los tipos de datos numéricos en Java no pueden representar cualquier número entero o real. Por ejemplo, el tipo de dato entero `int` tiene un intervalo de representación entre -2147483648 y 2147483647. Si se desea representar el valor correspondiente a la población mundial del planeta (más de 6 mil millones de habitantes) no puede hacerse con dato de tipo `int`.

Los datos de tipo `double` no tienen tanta limitación. Pueden alcanzar un valor del orden de 10308, pero tienen otro problema: la precisión. En concreto, el tipo `double` tiene una precisión de 15 dígitos significativos cómo se detallará más adelante.

LITERALES O CONSTANTES LITERALES DE TIPOS DE DATO PRIMITIVOS

Un **literal**, valor literal ó constante literal es una constante cuyo nombre o identificador es la representación escrita de su valor y tiene ya ese significado en el código fuente de un programa Java. Seguidamente se muestran algunos ejemplos de **valores o constantes literales** pertenecientes a los tipos primitivos que pueden utilizarse directamente en un programa fuente de Java.

- Las constantes literales **booleanas:** son `false` y `true`.
- Las constantes literales de tipo carácter: aparecen entre comillas simples. Como ya se ha comentado anteriormente, los caracteres, cadenas e identificadores en Java se componen de caracteres pertenecientes al conjunto de caracteres **Unicode** (http://www.unicode.org). Un dato de tipo caracter representa un único carácter. El formato de Unicode utiliza 16 bits (2 bytes) para poder codificar un total de 65536 caracteres diferentes que incluyen caracteres y símbolos procedentes de distintas lenguas del mundo. En este conjunto de caracteres encontramos letras mayúsculas (‘A’, ‘B’, ‘C’,...), letras minúsculas (‘a’, ‘b’, ‘c’,...), signos de puntuación (‘,’ ‘;’ ‘:’ ...), dígitos (‘0’, ‘1’, ‘2’,...), símbolos especiales (‘#’, ‘&’, ‘%’,...) y caracteres de control (tabulador, retorno de carro,...).

Hay algunos caracteres que pueden causar algún problema en el código fuente de un programa Java debido a que se utilizan para tareas específicas dentro del lenguaje. Por ejemplo, como se verá más adelante, el carácter correspondiente a las comillas dobles (") se emplea para delimitar una cadena de caracteres. Para solucionar este inconveniente, existen diferentes formas de indicar una constante literal de tipo carácter. Por ejemplo:

```
‘t’ // Representa la letra minúscula, con comillas simples
\t // Con la barra de pendiente negativa: secuencia de escape
\u0234 // Con la barra y la letra u: secuencia Unicode
```

Una *secuencia de escape* es una serie de caracteres que comienza por el carácter \ seguido de una letra, un conjunto de dígitos o la letra u seguida de un conjunto de dígitos y que representa a un carácter. En este último caso, la secuencia de caracteres o secuencia de escape `\uxxx` puede emplearse en cualquier lugar en un programa de Java para representar un carácter Unicode, siendo `xxxx` una secuencia de cuatro dígitos hexadecimales. En la siguiente tabla se muestran algunos ejemplos de secuencias de escape.

Secuencia de escape	Valor
\b	Retroceso o *backspace* (equivalente a \u0008)
\t	Tabulador (equivalente a \u0009)
\n	Nueva línea (equivalente a \u000A)
\f	Salto de página (equivalente a \u000C)
\r	Retorno de carro (equivalente a \u000D)
\"	Doble comilla (equivalente a \u0022)
\'	Comilla simple (equivalente a \u0027)

Figura 3.2 Caracteres representados por secuencias de escape

Aunque en realidad una cadena no es un tipo primitivo de Java pueden construirse constantes literales de **cadenas** de caracteres. Las constantes literales de cadenas de texto se indican entre comillas **dobles**. Por ejemplo:

```
"Hola, mundo" // Con comillas dobles: indica una cadena
```

Al construir una cadena de caracteres se puede incluir cualquier carácter Unicode excepto un carácter de nueva línea. Si se desea incluir un salto de línea en una cadena de caracteres debe utilizarse la secuencia de escape \n.

Las constantes **enteras** son secuencias de dígitos octales, decimales o hexadecimales en las que no se emplea el punto o coma decimal. Si comienza por un 0 indica un formato **octal**. Si comienza por un 0x ó 0X indica un formato **hexadecimal**. El resto se considera en formato decimal. Las constantes de tipo long se indican con una l o una L al final. Normalmente se emplea la L para no confundir con el 1 (uno). Si no se indica ninguna de estas terminaciones se supone un tipo int. A continuación se muestran algunos ejemplos de constantes enteras:

```
34 // de tipo int, solo digitos
-78 // digitos sin punto decimal
034 // en octal (equivale al 28 decimal)
0x1C // en hexadecimal (equivale al 28 decimal)
875L // de tipo long
```

Las constantes **reales** o en coma flotante se expresan con coma decimal y opcionalmente seguidos de un exponente. El valor puede finalizarse con una f o una F para indica el formato float (por defecto es double). Por ejemplo:

```
15.2 // de tipo double
15.2D // el mismo valor
1.52e1 // el mismo valor
0.152E2 // el mismo valor
.8e10 // de tipo double
15.8f // de tipo float
15.8F // tambien de tipo float
```

Como se verá más adelante cada tipo de dato primitivo tiene una clase correspondiente (Boolean, Character, Byte, Short, Integer, Long, Float y Double), llamadas *wrappers*, que definen también constantes y métodos útiles.

FORMATO DE REPRESENTACIÓN DE LOS DATOS NUMÉRICOS REALES

Los números reales se representan en el ordenador según un formato de coma flotante. Los bits empleados para la representación de un valor real se dividen en dos componentes: **mantisa y exponente**. En el sistema binario, la expresión matemática que relaciona la mantisa, el exponente y el valor del número real representado es:

Valor = mantisa · 2exponente

Un número real de tipo float utiliza 32 bits de los cuales 24 son para la mantisa y 8 para el exponente. La mantisa representa un valor entre –1.0 y 1.0 y el exponente representa la potencia de 2 necesaria para igualar la mantisa al valor que se quiere representar conjuntamente.

Los números reales se caracterizan por dos magnitudes: la **precisión** y el **intervalo de representación**. La precisión es el número de dígitos significativos con los que se puede representar un número y el intervalo de representación es la diferencia entre el mayor y el menor número que se pueden representar.

La precisión de un número real depende del número de bits de su mantisa mientras que el intervalo depende del número de bits de su exponente.

Una mantisa de 24 bits como en el caso del tipo float puede representar aproximadamente +/-223, o sea, cerca de 7 dígitos decimales significativos.

Un exponente de 8 bits puede representar multiplicadores entre 2-128 y 2127 de forma que el intervalo de representación es de 10-38 a 1038 aproximadamente. En consecuencia los datos de tipo float tienen un intervalo de representación mucho mayor en el mayor de los enteros pero con sólo siete dígitos significativos de precisión.

Cuando un valor con más de siete cifras significativas se almacena en una variable de tipo float sólo se consideran los siete dígitos decimales más significativos. Los dígitos restantes se pierden. Por ejemplo si se almacena el valor 1234,56789 en una variable de tipo float, el valor se redondeará a 1234,568. Esta diferencia entre el valor original y el valor representado en el ordenador se denomina error de redondeo. Por este motivo, es muy importante seleccionar el tipo de dato numérico real con la precisión suficiente con el fin de considerar las cifras necesarias para resolver un problema particular correctamente.

Por su parte, un número real de tipo double utiliza 64 bits en la memoria del ordenador dividida en 53 bits para la mantisa y 11 para el exponente. Una mantisa de 53 bits permite representar entre 15 y 16 dígitos decimales significativos así que la precisión es de 15 dígitos decimales. Un exponente de 11 bits permite representar multiplicadores entre 2-1024 y 21024, de forma que el intervalo de representación va de 10-308 y 10308 aproximadamente.

Una constante real se distingue de un entero porque contiene un punto decimal (no una coma) y/o un exponente. Si la constante es positiva puede escribirse con o sin el signo +. El tipo de una constante real se puede especificar añadiendo la letra `F` para el tipo `float` o la letra `D` para `double`. Por defecto, una constante real es de tipo `double`. En este caso debe estar en el intervalo de representación comprendido entre los valores numéricos reales -1.79769313486231570E+308 y 1.79769313486231570E+308.

Nota de interés

En la programación con Java es recomendable emplear el tipo double cuando se trabaja con tipos de dato reales, considerando el funcionamiento de los algoritmos y métodos matemáticos y de los recursos de los sistemas informáticos que se emplean en la actualidad.

DECLARACIONES DE VARIABLES

Una variable es un espacio de la memoria correspondiente a un dato cuyo valor puede modificarse durante la ejecución de un programa y que está asociado a un identificador. Toda variable ha de declararse antes de ser usada en el código de un programa en Java. En la declaración de una variable debe indicarse explícitamente el identificador de la variable y el tipo de dato asociado. El tipo de dato determina el espacio reservado en memoria, los diferentes valores que puede tomar la variable y las operaciones que pueden realizarse con ella. La declaración de una variable en el código fuente de un programa de Java puede hacerse de la siguiente forma:

```
tipo_de_dato identificador_de_la_variable;
```

O bien, la declaración de múltiples variables (con los correspondientes identificadores separados por comas) del mismo tipo:

```
tipo_de_dato ident_1, ident_2, . . . , ident_n;
```

Por ejemplo:

```
int n;
double x, y;
```

En el primer ejemplo se declara n como una variable de tipo `int`. En el segundo ejemplo se declaran dos variables `x` e `y` de tipo `double`**.** En Java una variable queda definida únicamente dentro del bloque de sentencias (entre llaves { }) en el que ha sido declarada. De esta forma queda determinado su ámbito o alcance (scope) en el que puede emplearse.

El identificador elegido para designar una variable debe respetar las normas de construcción de identificadores de Java. Además, por convención:

- los identificadores de las variables comienzan con una letra minúscula. Por ejemplo: `n, x2, mes, clave, suma, ó nombre`.

- si el identificador es una palabra compuesta, las palabras restantes comienzan por una letra mayúscula. Por ejemplo: `esDivisible`.
- El carácter del subrayado puede emplearse en cualquier lugar del identificador de una variable pero suele emplearse para separar nombres en identificadores de constantes.

La declaración e inicialización de una variable de tipo primitivo puede realizarse de forma simultánea en la misma línea empleando el operador asignación =. Por ejemplo:

```
int n = 15;
```

Independientemente de haber inicializado o no, el valor asignado a la variable puede modificarse las veces que se quiera durante la ejecución del programa. También puede realizarse la declaración e inicialización de varias variables del mismo tipo primitivo en la misma línea separándolas por comas. Por ejemplo:

```
double x = 12.5, y = 25.0;
```

En el caso de que no se inicialice explícitamente la variable, ésta toma el valor 0 si es numérica, `false` si es booleana y `'\0'` si es de tipo carácter.

Como se verá más adelante, la declaración de un objeto o instancia equivalente se realiza empleando la palabra `new`. Por ejemplo:

```
Integer n = new Integer(15);
```

Esta declaración y el tipo de dato (clase) `Integer` se verá más adelante con detenimiento.

Ejercicio: CONVERSIONES ENTRE TIPOS DE DATO

```
public class Conversiones {
    public static void main (String [] args) {
        int a = 2;
        double b = 3.0;
        float c = (float) (20000*a/b + 5);
        System.out.println("Valor en formato float: " + c);
        System.out.println("Valor en formato double: " + (double) c);
        System.out.println("Valor en formato byte: " + (byte) c);
        System.out.println("Valor en formato short: " + (short) c);
        System.out.println("Valor en formato int: " + (int) c);
        System.out.println("Valor en formato long: " + (long) c);
    }
}
```

El proceso consiste en almacenar el valor de una variable de un determinado tipo primitivo en otra variable de distinto tipo. Suele ser una operación más o menos habitual en un programa y la mayoría de los lenguajes de programación facilitan algún mecanismo para llevarla a cabo.

En cualquier caso, no todas las conversiones entre los distintos tipos de dato son posibles. Por ejemplo, en Java no es posible convertir valores booleanos a ningún otro tipo de dato y viceversa.

Además, en caso de que la conversión sea posible es importante evitar la pérdida de información en el proceso. En general, existen dos categorías de conversiones:

- De ensanchamiento o promoción. Por ejemplo: pasar de un valor entero a un real.
- De estrechamiento o contracción. Por ejemplo: pasar de un valor real a un entero.

Las conversiones de promoción transforman un dato de un tipo a otro con el mismo o mayor espacio en memoria para almacenar información. En estos casos puede haber una cierta pérdida de precisión al convertir un valor entero a real al desechar algunos dígitos significativos. Las conversiones de promoción en Java se resumen en la siguiente tabla.

Tipo de origen	Tipo de destino
byte	short, int, long, float, double
short	int, long, float, double
char	int, long, float, double
int	long, float, double
long	double
float	float, double
double	-

Las conversiones de contracción son más comprometidas ya que transforman un dato de un tipo a otro con menor espacio en memoria para almacenar información. En estos casos se corre el riesgo de perder o alterar sensiblemente la información. Las conversiones de contracción en Java se resumen en la siguiente tabla.

Tipo de origen	Tipo de destino
byte	char
short	byte, char
char	byte, short
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Tanto las conversiones de promoción como las de contracción se realizan por medio de los siguientes mecanismos:

- Por **asignación**: cuando una variable de un determinado tipo se asigna a una variable de otro tipo. Sólo admite conversiones de **promoción**. Por ejemplo: si `n` es una variable de tipo `int` que vale `25` y `x` es una variable de tipo `double`, entonces se produce una conversión por asignación al ejecutarse la sentencia
- `x = n;`
- la variable x toma el valor `82.0` (valor en formato real). El valor de `n` no se modifica.

- Por **promoción aritmética**: como resultado de una operación aritmética. Como en el caso anterior, sólo admite conversiones de **promoción**. Por ejemplo, si `producto` y `factor1` son variables de tipo `double` y `factor2` es de tipo `int` entonces la ejecutarse la sentencia
- `producto = factor1 * factor2;`
- el valor de `factor2` se convierte internamente en un valor en formato real para realizar la operación aritmética que genera un resultado de tipo double. El valor almacenado en formato entero en la variable `factor2` no se modifica.
- Con **casting o "moldes"**: con operadores que producen la conversión entre tipos. Admite las conversiones de promoción y de contracción indicadas anteriormente. Por ejemplo: si se desea convertir un valor de tipo `double` a un valor de tipo `int` se utilizará el siguiente código
- `int n; double x = 82.4; n = (int) x;`
- la variable `n` toma el valor `82` (valor en formato entero). El valor de `x` no se modifica.□ El código fuente del siguiente programa ilustra algunas de las conversiones que pueden realizarse entre datos de tipo numérico.

Ejercicio:

Dado un número generar su tabla de multiplicar desde el 1 hasta el 10, como se puede modificar el algoritmo para que genere también las tablas que están antes de la que se pidió: Por ejemplo:

A- indica la tabla de multiplicar: 3
3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30

B- indica la tabla de multiplicar: 3
1 X 1 = 1
1 X 2 = 2
1 X 3 = 3
1 X 4 = 4
1 X 5 = 5
1 X 6 = 6
1 X 7 = 7
1 X 8 = 8
1 X 9 = 9
1 X 10 = 10

2 X 1 = 2
2 X 2 = 4
2 X 3 = 6
2 X 4 = 8
2 X 5 = 10
2 X 6 = 12
2 X 7 = 14
2 X 8 = 16
2 X 9 = 18
2 X 10 = 20

3 X 1 = 3
3 X 2 = 6
3 X 3 = 9
3 X 4 = 12
3 X 5 = 15
3 X 6 = 18
3 X 7 = 21
3 X 8 = 24
3 X 9 = 27
3 X 10 = 30

```
import java.util.Scanner;

public class TablaMultiplicar {
	public static void main(String[] args){
		Scanner obtenerNumero = new Scanner(System.in);
		int numero,i,j;

		System.out.print("indica la tabla de multiplicar: ");
		numero = obtenerNumero.nextInt();


		for(j = 1; j <= 10; j++){
			System.out.println(numero + " X " + j + " = " + numero*j);
		}
		System.out.println();

	}
}
```
