Tipos de datos.

1 Introducción al Java

1.1 Características relevantes del Java

Desafortunadamente, no existe el lenguaje perfecto, sin ningún inconveniente y que sea ideal para crear cualquier tipo de programa. Siempre hay que llegar a un compromiso entre ventajas e inconvenientes. De hecho, la elección misma de qué lenguaje hay que usar puede llegar a condicionar enormemente el proceso de creación de un programa, y no es sensato usar siempre el mismo para resolver cualquier problema. En cualquier caso, vale la pena comentar los motivos por los que se considera interesante usar Java.

- **Popular**: Java fue creado en 1995 por la firma Sun Microsystems, que en 2009 fue comprada por la empresa de bases de datos Oracle. Su propósito era ofrecer un lenguaje lo menos ligado posible a la arquitectura sobre la que se ejecuta. Esto lo convirtió en sus inicios en el mecanismo más versátil existente para ejecutar aplicaciones sobre navegadores web. Desde entonces, su popularidad ha ido en aumento también como lenguaje para crear aplicaciones de escritorio, y actualmente es uno de los lenguajes más utilizados en este campo. Esto hace que la demanda de profesionales que lo dominen sea muy alta y que tenga una gran aceptación y cantidad de documentación disponible.

- **De nivel alto con compilador estricto**: Java es un lenguaje de nivel muy alto, con todas las ventajas que ello implica. Adicionalmente, su compilador es especialmente estricto a la hora de hacer comprobaciones sobre la sintaxis empleada y cómo se manipulan los datos que se están tratando en el programa. Si bien esto a veces puede parecer un poco molesto cuando se producen ciertos errores de compilación, en realidad se una ventaja, ya que enseña al programador a tener más grado de control sobre el código fuente que genera, de forma que sea correcto.

- **Multiplataforma**: uno de los factores decisivos en la popularidad de Java es que sus programas se pueden ejecutar en cualquier plataforma sin necesidad de volver a compilar. Una vez el código fuente se ha compilado una vez, el bytecode resultante puede ser llevado a otras plataformas basadas en otro tipo de procesador y continuará funcionando. Solamente hay que disponer del intérprete correspondiente para la nueva plataforma. Esto homogeneizar enormemente el aprendizaje del lenguaje independientemente de la plataforma que utiliza para estudiarlo.

- **Orientado a objetos**: este es el nombre de una metodología avanzada, muy popular y útil, para diseñar programas. Si bien sus detalles están totalmente fuera del alcance de este módulo, basta decir que es un mecanismo de nivel muy alto para acercar la forma en que se hacen los programas al método de pensamiento humano.

En este curso no veremos un enfoque orientado a objetos de la programación, pero el lenguaje Java le permitirá, cuando conozcáis esta tecnología, trabajar.

También será necesario ser conscientes de algunas particularidades:

- **Orientado a objetos**: algunas partes de la sintaxis y la nomenclatura formal de Java están íntimamente vinculadas a la metodología de la orientación a objetos y no se pueden separar. Por tanto, en algunos momentos será inevitable tener que hacer frente a aspectos ligados a la orientación a objetos, aunque se trate de una metodología avanzada que no sea objeto de estudio en este curso. La ventaja es que al pasar a programar con orientación a objetos ya conoceréis el lenguaje.
- **Interpretado con bytecode**: esta es una característica derivada de que sea multiplataforma. Al tratarse de un lenguaje interpretado, es necesario disponer del interprete correctamente instalado y configurado en cada máquina donde quiera ejecutar su programa. Esto quiere decir que hay un programa más que debe configurar correctamente en su sistema. Esto también hace que la ejecución de los programas en Java no siga el proceso típico de cualquier otra aplicación (por ejemplo, ejecutarlo desde línea de comandos o hacer doble clic en la interfaz gráfica). No hay archivo que se pueda identificar claramente como ejecutable.

1.2 Creación y ejecución de programas en Java

Este apartado se centra en mostrar detalladamente cómo se crea y ejecuta un programa en lenguaje Java.

Dado que Java es un lenguaje interpretado, las herramientas que necesita son:

- Un editor de texto simple cualquiera.
- Un compilador del lenguaje Java, para generar bytecode.
- Un intérprete de Java, para poder ejecutar los programas.

Disponer de un editor de texto es sencillo, ya que todos los sistemas operativos de propósito general suelen tener instalado algún predeterminado. Ahora bien, cuando se edita un archivo de código fuente, hay que asignarle una extensión específica de acuerdo con el lenguaje de programación empleado, de manera que pueda ser fácilmente identificado como tal.

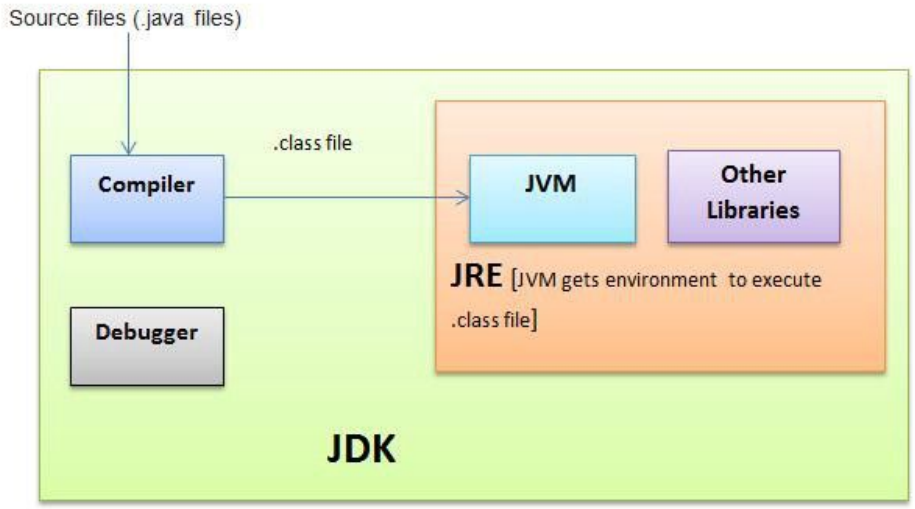
La extensión de los archivos de código fuente en el Java es la .java.

En el caso concreto de Java, hay una convención a la hora de dar nombre a un archivo de código fuente. Se suele usar **UpperCamelCase** (notación de camello con mayúsculas). Esta corresponde a usar sólo letras consecutivas sin acentos (ni espacios, subrayados o números), y en que la inicial de cada palabra usada sea siempre en mayúscula. Esto no es estrictamente imprescindible, pero sí muy recomendable, ya que es el estilo de nomenclatura que siguen todos los programadores de Java y la que se encuentra en

documentación, guías u otros programas. Además, en algunos sistemas, el uso de caracteres especiales, como los acentos, puede llevar a errores de compilación.

Algunos ejemplos de nombres de archivos de código fuente aceptables son: Prova.java, HolaMon.java, ElMeuPrograma.java, etc.

El compilador y el intérprete de Java son dos programas que tendrá que instalar el su ordenador, ya que normalmente no están preinstalados por defecto.



- **JDK:** Java Development Kit es un software que provee herramientas de desarrollo para la creación de programas en Java.
- **JVM:** Java Virtual Machine o Máquina Virtual de Java. Es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el **bytecode** Java), el cual es generado por el compilador del lenguaje Java.

- **JRE:** Java Runtime Environment. Es un conjunto de utilidades que permite la ejecución de programas Java.

- **Compiler** o **compilador**: es un programa informático que traduce un programa que ha sido escrito en un lenguaje de programación a un lenguaje máquina o a un código intermedio (bytecode) como en el caso del Java.

- **Debugger** o **depurador**: es un programa usado para probar y depurar (eliminar) los errores de otros programas (el programa "objetivo").

- **Libraries** o **librerias**: En Java y en varios lenguajes de programación más, existe el concepto de librerías. Una librería en Java se puede entender como un conjunto de clases, que poseen una serie de métodos y atributos. Lo realmente interesante de estas librerías para Java es que facilitan muchas operaciones. De una forma más completa, las librerías en Java nos permiten reutilizar código, es decir que podemos

hacer uso de los métodos, clases y atributos que componen la librería evitando así tener que implementar nosotros mismos esas funcionalidades.

1.2.1 Instalación del JRE / JDK en Ubuntu

La opción más fácil para instalar Java es usar la versión empaquetada con Ubuntu. Por defecto, Ubuntu 18.04 incluye Open JDK, que es una variante de código abierto de JRE y JDK.

Este paquete instalará OpenJDK 10 u 11:

- Antes de septiembre de 2018, esto instalará OpenJDK 10.
- Después de septiembre de 2018, esto instalará OpenJDK 11.

Para instalar esta versión, primero actualice el índice del paquete:

```
$ sudo apt update
```

A continuación, comprueba si Java ya está instalado:

```
$ java -version
```

Si Java no está instalado actualmente, verá el siguiente resultado:

```
Output

openjdk version "10.0.1" 2018-04-17

OpenJDK Runtime Environment (build 10.0.1+10-Ubuntu-3ubuntu1)

OpenJDK 64-Bit Server VM (build 10.0.1+10-Ubuntu-3ubuntu1, mixed mode)
```

Es posible que necesite el Java Development Kit (JDK) además del JRE para compilar y ejecutar algún software específico basado en Java. Para instalar el JDK, ejecute el siguiente comando, que también instalará el JRE:

```
$ sudo apt install default-jdk
```

Verifica que el JDK está instalado verificando la versión de javac, el compilador de Java:

```
javac -version
```

Verás la siguiente salida:

```
Output

javac 10.0.1
```

1.2.2 Instalación del JRE / JDK en Windows

Normalmente en windows tienes instalado el JRE, pero no el JDK. Por lo tanto, podemos ejecutar programas creados en java (los .class que veremos más adelante), pero no podemos compilar los fuentes (los .java).

Para comprobar qué versión de java tenemos abriremos una ventana de línea de comandos. Cuando se abra la ventana, sólo tenemos que escribir el siguiente comando:

```
java -version
```

Si el sistema responde con un mensaje de error es que no tenemos instalado ni JRE, ni JDK.

En caso de que nos responda con la versión que tenemos instalada, se referirá a JRE, pero en ese caso, nos basta con ejecutar el compilador de Java para saber si también tenemos instalado JDK.

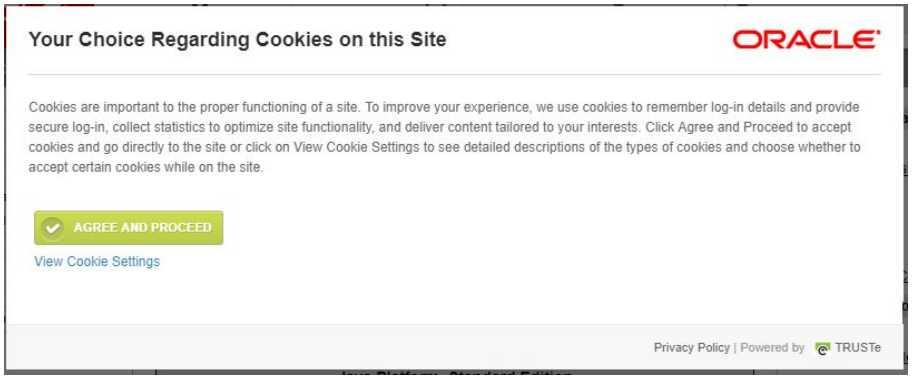
```
javac
```

Si da un error es que no tenemos instalado el JDK.

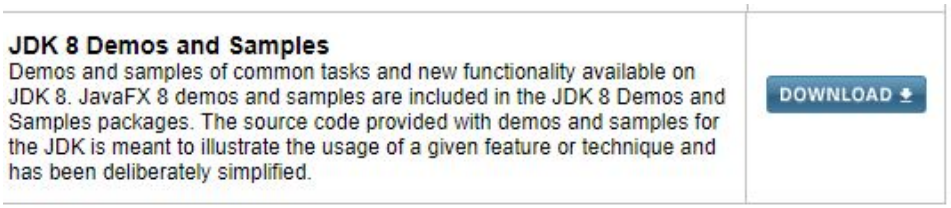
1.2.2.1 Descargar JDK para windows.

Supongamos que solo tenemos instalado el JRE, que es el caso más habitual. El primer paso será abrir el navegador y navegar hasta la página de descargas de Java (http://www.oracle.com/technetwork/java/javase/downloads).

Al llegar, lo primero que veremos será un aviso es un aviso sobre el uso de cookies, que deberemos aceptar antes de continuar.



Una vez en la página de descarga, deberemos desplazarnos hacia abajo hasta encontrar el botón que nos permite descargar JDK. Una vez localizado, hacemos clic sobre él.



En la página de destino encontramos una lista con las versiones para todos los sistemas operativos disponibles. Pero antes, en la cabecera de la lista encontramos un enlace al acuerde de licencia que debemos aceptar antes de iniciar la descarga.

Una vez lo hayamos leído, para entender a qué nos obliga la instalación del producto, podemos proceder a aceptarlo.

A continuación, ya podemos hacer clic sobre el enlace de descarga.En mi caso, me decanto por la versión de Windows para arquitecturas de 64 bits.

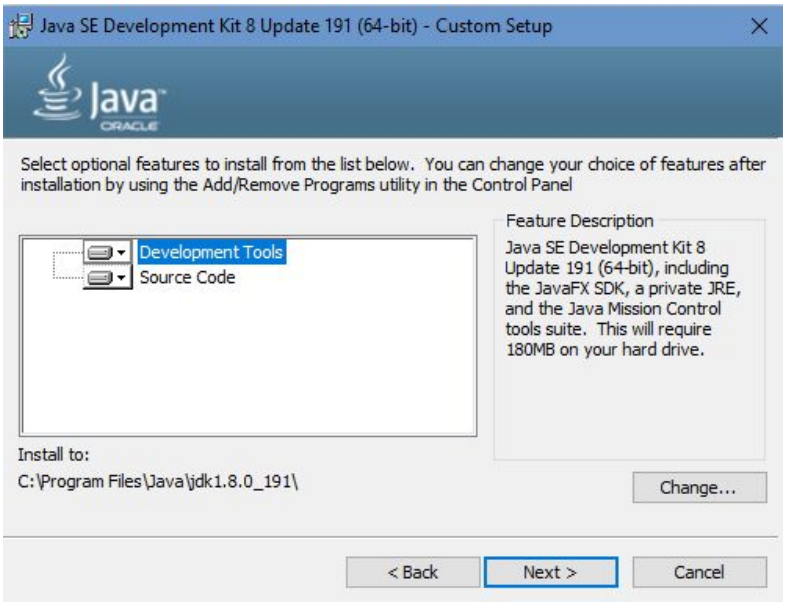
1.2.2.2 Instalar JDK para Windows

Cuando concluya la descarga, sólo tendrás que usar tu explorador de archivos para abrir la carpeta donde almacenes habitualmente los archivos descargados. Una vez allí, puedes iniciar la instalación haciendo doble clic sobre el archivo que acabas de obtener.

En la ventana del Control de cuentas de usuario nos avisará de que se va a ejecutar un programa que puede hacer cambios en nuestro equipo. Hacemos clic sobre el botón Sí y empezará la instalación.

La primera pantalla del asistente de instalación que nos ofrece es sólo de bienvenida, por lo que nos limitaremos a hacer clic sobre el botón Next.

En el siguiente paso, debemos indicar los componentes que queremos instalar. Se encuentran divididos en tres categorías: Development Tools, Source Code y Public JRE (si lo tenemos instalado no aparecerá este último). En cualquier caso, nosotros dejaremos los valores predeterminados.



A partir de aquí, el asistente se dedica a descomprimir todos los archivos que forman el programa y colocarlos en la ubicación correcta. También nos muestra una barra de progreso para mostrarnos el avance de la tarea.

Durante el proceso, también se instala JRE (si no estaba ya instalado). Recuerda que es la aplicación multiplataforma que permite la ejecución de programas y que incluye JVM (Java Virtual Machine).

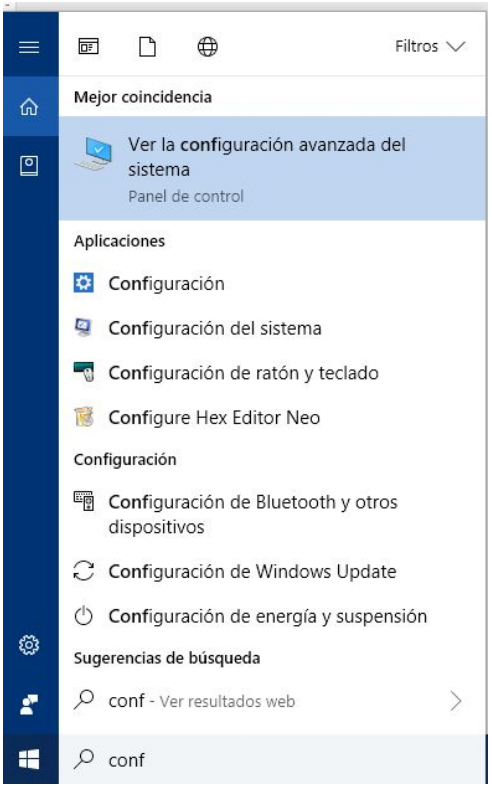
Por ese motivo, aparecerá una ventana pidiéndonos que decidamos en qué carpeta queremos instalarlo (recordad que si ya está instalado no dirá nada). Puedes utilizar el botón Cambiar para elegir la carpeta que prefieras o dejar la que te ofrece el asistente de forma predeterminada.

Al final, el asistente nos informa de que la instalación ha terminado y nos ofrece un botón botón (Next Steps) que nos facilitará el acceso a información complementaria con tutoriales, documentación sobre el API, guía para el desarrollador, notas de la versión que hemos instalado, etc.

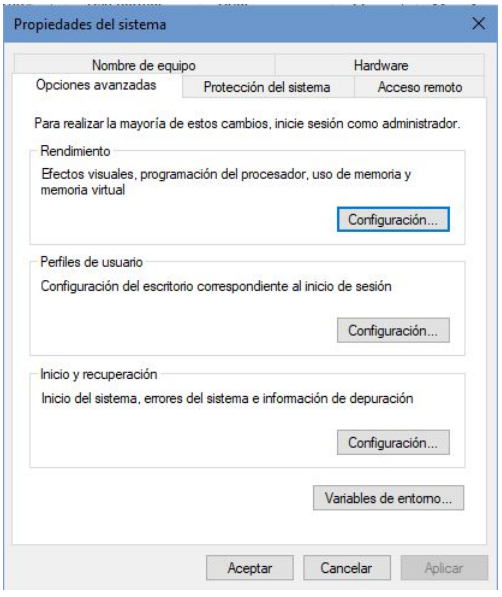
1.2.2.3 Ajustes después de la instalación de JDK

Curiosamente, el proceso de instalación de JDK no añade la carpeta donde se ha instalado, al conjunto de rutas donde Windows busca sus programas ejecutables. Y si no lo hacemos a mano, cuando necesitemos los diferentes componentes de JDK, no será capaz de encontrarlos.

Buscamos la **Configuración avanzada del sistema.** Lo más sencillo es usar la búsqueda de Cortana:



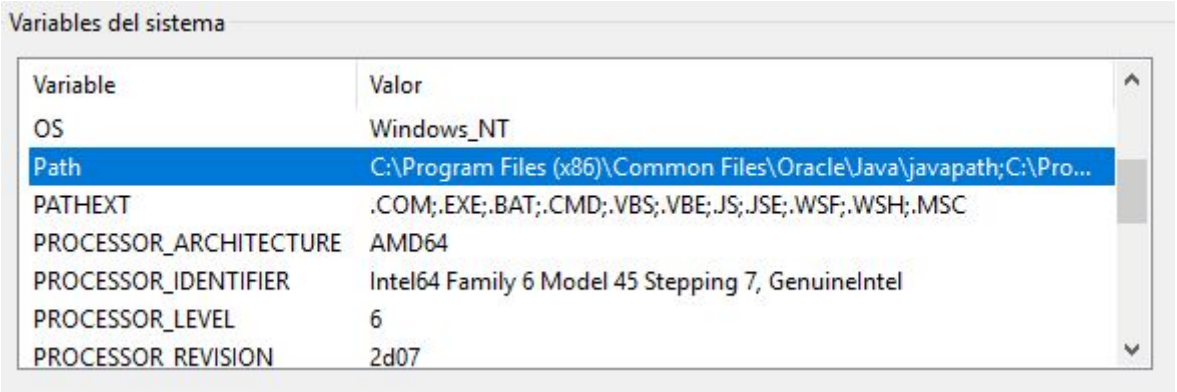
En pantalla la ventana **Propiedades del sistema**, nos debemos asegurar que se encuentra seleccionada la solapa **Opciones avanzadas**.



Y, para seguir, hacemos clic sobre el botón Variables de entorno. Cuando aparezca la ventana **Variables de entorno**, comprobaremos que tenemos dos zonas bien delimitadas.

En la parte superior, aparece una lista con las Variables de usuario para, seguida del nombre de la cuenta de usuario actual. Y en la parte inferior,la lista contiene las **Variables del sistema**.

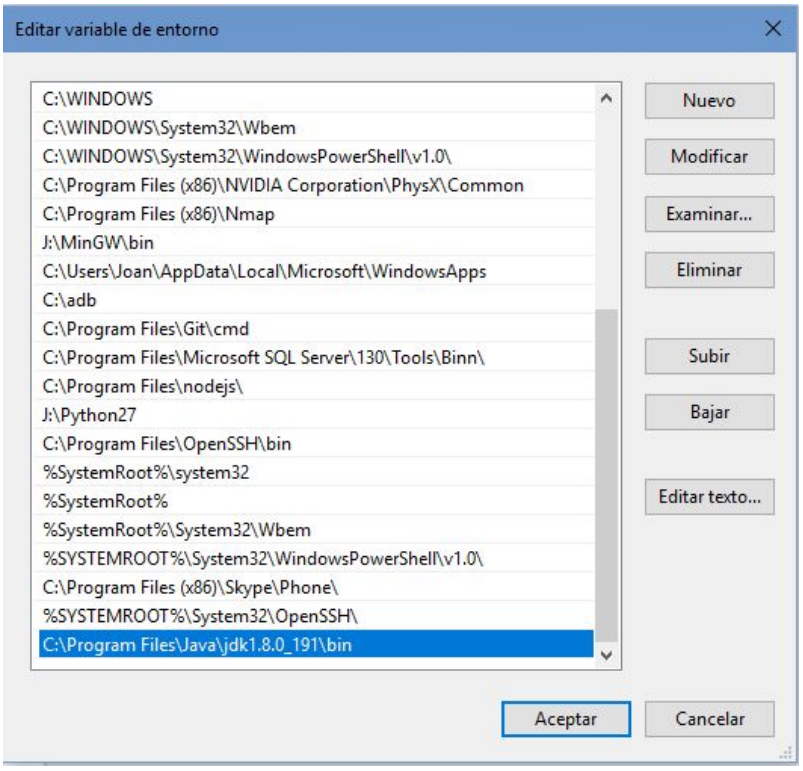
Esta última es la que contiene la variable que nos interesa. Su nombre es **Path** y, cuando la localicemos, debemos hacer clic sobre ella.



Una vez seleccionada, hacemos clic sobre el botón Editar.

De este modo, conseguiremos una nueva ventana titulada **Editar la variable del sistema**, con una lista de los distintos path almacenados en nuestro sistema. Ahora de debemos añadir la ruta donde se ha instalado JDK.

Si no la conoces, basta con abrir el explorador de archivos y buscarlo dentro de Archivos de programas. Por ejemplo, en mi equipo, la ruta es: **C:\Program Files\Java\jdk1.8.0_191\bin**.



Probablemente, en tu caso sea muy parecido, aunque puede cambiar el número de versión de JDK.

Una vez que tengas la ruta, debes hacer clic sobre el botón **Nuevo** y, a continuación, escribimos la ruta anterior.

Cuando estemos listos, hacemos clic sobre el botón Aceptar.

De vuelta en la ventana Propiedades del sistema, sólo nos queda validar los cambios haciendo clic sobre el botón Aceptar.

Y por último, podemos cerrar todas las ventanas.

1.2.2.4 Comprobar que todo funciona

Para comprobar que JDK está funcionando correctamente, basta con reiniciar el sistema y volver a tratar de ejecutar el compilador de java:

```
javac
```

1.3 Ejemplo de programa: ¡Hola, mundo!

Dentro del ámbito de la programación es tradición que el primer programa que se escribe y se ejecuta cuando se inicia el estudio de un nuevo lenguaje sea el llamado "Hola, mundo!" (Originalmente en inglés, " Hello, world! "). Esta tarea es un simple ejercicio de copiar el código fuente del programa, por lo que ni siquiera hay que entender aún la sintaxis del lenguaje. El objetivo principal de este programa es ver que el entorno de trabajo se encuentra correctamente instalado y configurado, ya que por su sencillez es difícil que dé problemas. Además, también os hace servicio como plantilla de la estructura básica de un programa en el lenguaje escogido y permite repasar la estructura y algunos de los elementos básicos.

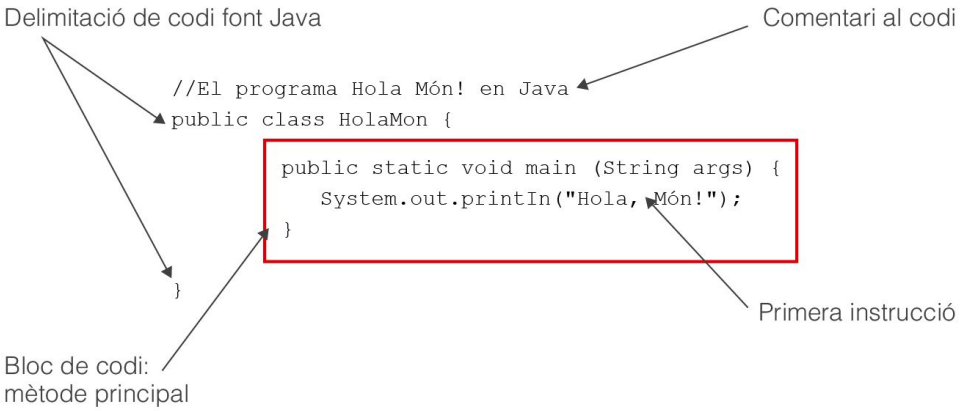
El código fuente para la versión en Java es el siguiente:

```
//El programa "Hola, mundo!" en Java
   public class HolaMundo {
       public static void main(String[] args) {
       System.out.println("Hola, món!");
   }
}
```

El archivo se guardará con el mismo nombre que la clase más la extensión .java. En este ejemplo será **HolaMundo.java**

1.4 Elementos de un programa en Java.

La siguiente figura muestra los elementos del programa “Hola, Mundo!”



1.4.1 Importación de bibliotecas

Una biblioteca es un conjunto de extensiones al conjunto de instrucciones disponibles cuando genere un programa. Para poder usar estas instrucciones adicionales es necesario declarar la importación. En caso contrario, las extensiones no están disponibles por defecto dentro del lenguaje. En el caso del programa "Hola, mundo!", al ser muy sencillo, no es necesario, pero en Java se usaría la sintaxis:

```
import <nombreBiblioteca>;
```

1.4.2 Indicador de inicio de código

El código fuente donde empieza realmente el programa en Java comienza con el texto de declaración que se muestra a continuación, donde <nombrearchivo> puede variar pero siempre debe corresponder exactamente con el nombre del archivo que lo contiene. Esto es esencial. En otro caso, el compilador nos dará un error. El código del programa se escribirá a continuación, siempre entre llaves, {...}.

```
public class <nombrearchivo> {
...
  }
```

Este texto declara que este archivo es el inicio de lo que en nomenclatura Java llama una clase. Este es un término estrechamente vinculado a la orientación a objetos, pero por ahora lo usaremos simplemente para referirnos a un archivo que contiene código fuente de Java.

1.4.3 Comentarios al código

Opcionalmente, también se pueden escribir comentarios dentro del código fuente. Se trata de texto que representa anotaciones libres al programa, pero que el compilador no procesa. Los comentarios pueden estar en cualquier parte del archivo, ya que el compilador los ignora.

Sirven para comentar el código fuente para explicar qué hace cada parte del programa, especialmente en aquellas más complejas, es una tarea muy importante que demuestra si un programador es cuidadoso o no.

En el programa Hola mundo está el comentario siguiente:

```
// El programa "Hola, mundo!" en Java
```

En lenguaje Java los comentarios se escriben o bien precediéndolos con dos barras, en caso de que tengan una sola línea, o bien en el formato siguiente si ocupan más de una línea.

```
/ **
* Este es el programa "Hola, mundo!"
* en el lenguaje de programación Java
** /
```

1.4.4 Indicador de la primera instrucción a ejecutar

Para que el ordenador sepa por dónde empezar a ejecutar instrucciones, ante todo debe saber cuál es la primera de todas. En algunos lenguajes esto se hace implícitamente, ya que la primera instrucción es directamente la primera línea de texto que aparece en el código fuente. En el caso del Java, hay un texto que lo indica claramente.

En el Java, el bloque de instrucciones en el que se engloba la primera instrucción del programa en la mayoría de lenguajes de programación se denomina el **método principal.**

Este método principal engloba todas las instrucciones del programa dentro de un bloque de instrucciones, entre llaves, {...}. Antes de las llaves hay una serie de texto que debe escribirse exactamente tal como se muestra. Si no, el intérprete de Java será incapaz de encontrarlo y de iniciar correctamente la ejecución del programa.

Concretamente, dirá que *"No encuentra el método principal" (main method not found)*.

```
public static void main (String [] args) {
    ...
}
```

La primera instrucción es la primera que hay escrita justo después de la llave abierta, {. La última instrucción es la escrita inmediatamente antes de la llave cerrada,}.

1.4.5 Bloque de código o de instrucciones

Las instrucciones o sentencias del programa están escritas una detrás de la otra, normalmente en líneas separadas para hacer el código más fácil de entender. En algunos lenguajes, al final de cada línea hay un **delimitador especial**, que sirva para indicar cuando termina una sentencia y empieza otra. Con el salto de línea no es suficiente. **En el caso del Java, se trata del punto y coma,;**.

Las diferentes instrucciones se suelen agrupar en **bloques de instrucciones**. El inicio y el fin de cada bloque diferente quedan identificados en Java para que las instrucciones **están rodeadas por llaves, {...}.**

Por tanto, en el programa HolaMundo sólo hay una única vez primera y última, instrucción.

```
System.out.println ( "Hola, mundo!");
```

2 Manipulación básica de datos

El propósito principal de todo programa de ordenador, en última instancia, es procesar datos de todo tipo. Para lograr esta tarea, el ordenador debe almacenar los datos en la memoria, por lo que posteriormente el procesador las pueda leer y transformar de acuerdo con los propósitos del programa.

> El término **dato** indica toda información que utiliza el ordenador en las ejecuciones de los programas.

Aunque a veces se usa el término datos, en plural, como si fuera algo en general, hay que tener en cuenta que dentro de un programa cada dato que se quiere tratar es un elemento individual e independiente. Por ejemplo, en un programa que suma dos números cualesquiera hay tres datos con los que trabaja: los dos operandos iniciales y el resultado final.

2.1 Tipos de datos

Estrictamente hablando, cualquier información se puede transformar en datos que pueda entender y manipular un ordenador. Un dato individual dentro de su programa puede ser un documento de texto, una imagen, un plano de un edificio, una canción, etc.

> Un tipo de dato es la definición del conjunto de valores válidos que pueden tomar unos datos y el conjunto de transformaciones que se puede hacer.

Como cada dato dentro de su programa siempre debe pertenecer a algún tipo, precisamente parte de su labor como programadores es **identificar cuál es el tipo que se acerca más a la información que desea representar y procesar.** Establecer cuál es el tipo de un dato implica que se establecen un conjunto de condiciones sobre ese dato a lo largo de todo el programa. Una de las más importantes es el efecto sobre la forma en que este dato se representará, tanto internamente dentro del hardware del ordenador como a la hora de representarla en el código fuente de sus programas.

2.1.1 Tipos de datos primitivos

Cada lenguaje de programación incorpora sus tipos de datos propios y, aparte, casi siempre ofrece mecanismos para definir otros nuevos partiendo de tipo de datos ya existentes. Por lo tanto, como programadores, sólo tiene que elegir entre todos los que ofrece el lenguaje cuál se acerca más a la clase de información que desea tratar.

> Los tipos primitivos de datos son los que ya están incorporados directamente dentro de un lenguaje de programación, y son usados como piezas básicas para construir otras más complejos.

El apoyo a diferentes tipos primitivos puede variar también entre lenguajes. De todos modos, hay cuatro que se puede considerar que, de una manera o de otra, todos los lenguajes los soportan. Se trata de los **números enteros, los reales, los caracteres y los booleanos.**

2.1.2 Literales

Un literal es un texto usado para representar un valor fijo dentro del código fuente de un programa.

2.1.3 El tipo de dato booleano

El tipo de dato booleano representa un valor de tipo lógico para establecer la certeza o falsedad de un estado o afirmación.

- Ejemplos de literales de un dato booleana: true (cierto) o false (falso). No hay más.
- Ejemplos de datos que se suelen representar con un booleano: interruptor encendido o apagado, estar casado, tener derecho de voto, disponer de carnet de conducir B1, la contraseña es correcta, etc.

Un literal de tipo booleano se representa simplemente escribiendo el texto tal como se ha descrito arriba. En un IDE, normalmente este texto queda resaltado en un color especial para que quede claro que se ha escrito un literal de tipo booleano.

La palabra clave para identificar este tipo de dato en Java es **boolean**.

2.1.4 El tipo de dato entero

El tipo de dato entero representa un valor numérico, positivo o negativo, sin decimal.

- Ejemplos de literales enteros: 3, 0, -345, 138764, -345.002, etc.
- Ejemplos de datos que se suelen representar con un entero: edad, día del mes, año, número de hijos, etc.

La palabra clave para identificar este tipo de dato en Java es **int**. Un literal de tipo entero se representa simplemente escribiendo un número sin decimales.

2.1.5 El tipo de dato real

El tipo de dato real representa un valor numérico, positivo o negativo, con decimales.

- Ejemplos de literales reales: 2.25, 4.0, -9653.3333, 100.0003, etc.

- Ejemplos de datos que se suelen representar con un real: un precio en euros, el récord mundial de los 100 m lisos, la distancia entre dos ciudades, etc.

Para referirse a un dato de tipo real, esta incluye siempre sus decimales con un punto (.). En los ejemplos nótese el detalle del valor 4.0. Los números reales representan valores numéricos con decimales, pero en su definición nada impide que el decimal sea 0. Por lo tanto, estrictamente, el valor 4 y el valor 4.0 corresponden a tipos de datos diferentes. El primero es un valor para un tipo de dato entero y el segundo para uno real.

La palabra clave para identificar este tipo de dato a Java es **double**.

Una pregunta que quizás le pueden plantear ahora es qué sentido tiene el tipo entero si con el tipo real ya podemos representar cualquier número, con decimales y todo. No es un poco redundante? Bueno, de entrada puede parecer que sí, pero hay un motivo para diferenciarlos. Sin entrar en detalles muy técnicos, representar y realizar operaciones con enteros internamente dentro del ordenador (en binario) es mucho más sencillo y rápido que con reales. Además, un entero requiere menos memoria. Evidentemente, un programa no será perceptiblemente más lento o más rápido por el simple hecho de hacer una operación entre dos datos de tipo real en lugar de entero, pero es una buena costumbre usar siempre el tipo de dato que se adapte exactamente a sus necesidades. Si un dato no tiene sentido que tenga decimales, como un número de año o de mes, es mejor usar el tipo entero.

2.1.6 El tipo de dato carácter

El tipo de dato carácter representa una unidad fundamental de texto usada en cualquier alfabeto, un número o un signo de puntuación o exclamación.

- Ejemplos de literales carácter: 'a', 'A', '4', '>', '?', etc.
- Ejemplos de datos que se suelen representar con un carácter: cada uno de los símbolos individuales de un alfabeto.

Para referirse a un dato de tipo carácter, ésta se rodea de comillas simples ('). Por tanto, no es lo mismo el carácter '4' y que el valor entero 4, ya que pertenecen a tipos diferentes. El primero lo usará para hacer referencia a una representación textual, mientras que el segundo es el concepto propiamente matemático. Tenga cuidado, ya que entre las comillas simples sólo puede haber un solo carácter, o Java dirá que la sintaxis no es correcta.

La palabra clave para identificar este tipo de dato en Java es **char**.

2.1.6.1 Sistemas de representación de los caracteres

A la hora de decidir cómo se representa internamente un carácter, se utilizan diferentes tablas de códigos establecidos, entre los que destacan los códigos ASCII y UNICODE.

El código ASCII tiene una pega, y es que sólo permite representar alfabetos occidentales, por lo que los programas que la usan para representar sus datos de tipo carácter son incompatibles con sistemas con otros alfabetos, como todos los asiáticos, el cirílico, etc.

Por este motivo, posteriormente se creó la tabla de codificación UNICODE, que permite codificar hasta 65.536 caracteres, pero manteniendo la compatibilidad con la codificación ASCII. Esto permite apoyar cualquier lengua actual, e incluso de antiguas, como los jeroglíficos egipcios. Este sistema es el que actualmente usan la mayoría de aplicaciones modernas.

El Java representa sus caracteres internamente usando la tabla UNICODE.

2.2 Operaciones

2.2.1 Operaciones entre booleanos

Este tipo de dato debe su nombre al álgebra de Boole, que es el conjunto de normas y operaciones que rigen cómo se pueden combinar los valores true y false.

2.2.1.1 Tabla de verdad de las operaciones entre booleanos

Operandos		Resultados de la operación		
A	B	A && B	A || B	! A
false	false	false	false	true
true	false	false	true	false
false	true	false	true	true
true	true	true	true	false

2.2.1.2 Tabla de verdad de las operaciones relacionales entre booleanos

Operandos		Resultados de la operación	
A	B	A == B	A !=| B
false	false	true	false
true	false	false	true
false	true	false	true
true	true	true	false

2.2.2 Operaciones entre enteros

Las operaciones básicas que se pueden hacer sobre datos de este tipo son de dos tipos: aritméticas y relacionales.

Los operadores aritméticos generalmente soportados son el cambio de signo (-, operador unario), la suma (+), resta (-, operador binario), multiplicación (*) y división (/).

Cualquier operación entre dos datos de tipo entero siempre da como resultado un nuevo dato también de tipo entero. Ahora bien, como que los números enteros no disponen de decimales, hay que tener presente que, en el caso de la división, el resultado se trunca y se pierden todos los decimales. En algunos lenguajes de programación también hay una quinta operación denominada módulo (%).

La siguiente tabla muestra ejemplos de resultados de la operación de división entera y módulo en datos de tipo entero.

Operandos		Resultado de las operaciones	
A	B	A / B	A % B
17	4	4	1
9	5	1	4
23	10	2	3
17689	1000	17	689

La siguiente tabla muestra ejemplos de las operaciones relacionales sobre enteros. Las operaciones son menor (<), mayor (>), menor o igual (<=), mayor o igual (>=), igual (==) y distinto (!=). El resultado de estas operaciones es un booleano (True o False).

Operandos		Resultados de las operaciones					
A	B	==	!=	>	<	>=	<=
4	3	False	True	True	False	True	False
14	-2	False	True	True	False	True	False
-78	34	False	True	False	True	False	True
12	12	True	False	False	True	True	True

2.2.3 Operaciones entre reales

Las operaciones que se pueden hacer entre datos de tipo real son exactamente las mismas que entre enteros, tanto aritméticas como relacionales. La única excepción es la operación módulo, que deja de tener sentido, ya que ahora la división sí se hace con cálculo de decimales.

2.3 Expresiones

Hasta ahora hemos vistos operaciones sobre uno o dos números, según si la operación es unaria o binaria. Ahora bien, en muchos casos, es útil poder aplicar de una sola vez un conjunto de operaciones diferentes sobre una serie de datos.

Una **expresión** es una combinación cualquiera de operadores y operandos.

Para construir una expresión, es necesario que ésta sea correcta en dos niveles, sintáctica y semánticamente, de forma que se respete el significado de los datos usados como operandos y sus operadores.

Desde el punto de vista sintáctico, las normas básicas de construcción de expresiones usando literales son las siguientes:

1. Se considera que un literal sólo es en sí mismo una expresión.
2. Dada una expresión correcta E, también lo es escribirla entre paréntesis: (E).
3. Dada una expresión correcta E y un operador unario op, opE es una expresión correcta.
4. Dadas dos expresiones correctas E1 y E2 y un operador binario op, E1 op E2 es una expresión correcta.

Veamos algunos ejemplos:

- La siguientes expresiones se pueden considerar correctas: 6, -6, (6), 6+9, (6+9), (6+9)*67, -(6*78+34)
- No son correctas las siguientes: +66, 78*, 6*(), 98*(67+5, 98*(67+)

Semánticamente, las normas que una expresión debe cumplir son las siguientes:

1. Cualquier operación siempre debe ser entre datos del mismo tipo.
2. La operación usada debe existir para el tipo de dato.

No serían correctas semánticamente las siguientes expresiones: 5.3=='d', true+false, -'g', 5==false, 5 || 4.0

2.3.1 Orden de precedencia

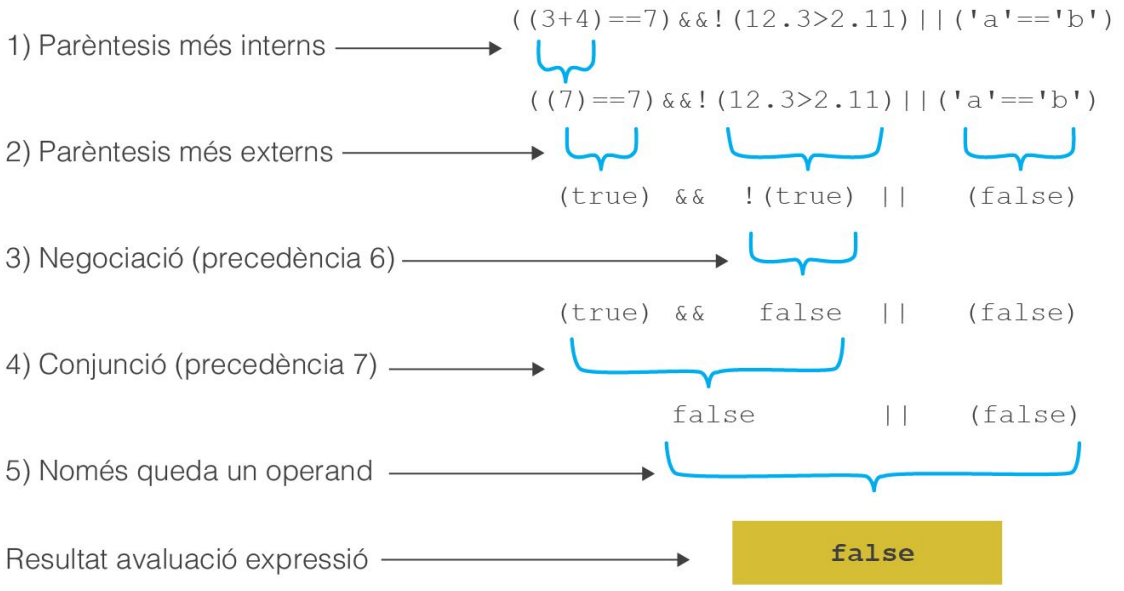
El orden de precedencia de un conjunto de operadores es la regla usada para establecer de manera no ambigua el orden en que deben resolver las operaciones dentro de una expresión.

Las operaciones con orden de precedencia mayor evalúan antes que las de orden menor. Para las operaciones que se han descrito hasta ahora, este orden es el siguiente.

En caso de empate, se resuelve la expresión ordenadamente de izquierda a derecha. La tabla siguiente muestra este orden, de más prioritario (1) a menos (8).

Orden	Operación	Operador
1	Cambio de signo	-(unario)
2	Producto, división y módulo	* / %
3	suma y resta	+ -(binario)
4	Relacionales de comparación	> < <= >=
5	Relacionales de igualdad	== !=
6	Negación	! (unario)
7	Conjunción	&&
8	Disyunción	||

Ejemplo de evaluación de expresiones



2.3.2 Rangos y palabras clave de los tipos primitivos en Java

Los tipos de datos básicos o primitivos no son objetos y se pueden utilizar directamente en un programa sin necesidad de crear objetos de este tipo. La biblioteca Java proporciona clases asociadas a estos tipos que proporcionan métodos que facilitan su manejo.

Los tipos de datos primitivos que soporta Java son:

Tipo	Palabra clave Java	Tamaño (Bytes)	Rango de Valores	Valor por defecto
byte	byte	1	-128 a 127	0
entero corto	short	2	-32768 a 32767	0
entero simple	int	4	-2147483648 a 2147483647	0
entero largo	long	8	-9223372036854775808 a 9223372036854775807	0
real simple precisión	float	4	$\pm$ 3.4x10^{-38} a $\pm$ 3.4x10^{38}	0.0
real doble precisión	double	8	$\pm$ 1.8x10^{-308} a $\pm$ 1.8x10^{308}	0.0
carácter	char	2	\u0000 a \uFFFF	\u0000
booleano	boolean	-	true ó false	false

Los literales que usa Java por defecto para los enteros es el int y para los reales usa por defecto double. Si se quiere usar un literal indicando que que es de un tipo diferente se puede indicar añadiendo un carácter al final del número. Por ejemplo:

- 268 → indica un int
- 268L → indica un long
- 5.0 → indica un double
- 5D → indica un double
- 5F → indica un float

Los números que introducimos están en base decimal. Si queremos usar otra base se hace de la siguientes forma:

- Números en binario: empiezan por 0b o 0B. Por ejemplo:
  - 0b00100001
  - 0B0010000101000101
- Números en hexadecimal: empiezan por 0x o 0X. Por ejemplo:
  - 0x1A2
  - 0x430
  - 0Xf4

2.3.3 Desbordamiento y errores de precisión

Como se ha visto en la tabla anterior, los números que se pueden representar usando los tipos primitivos son limitados. Por ejemplo, usando un tipo de dato byte no podemos representar números superiores a 127 ni inferiores a -128. En caso de salirme de estos límites se produce lo que se denomina un desbordamiento u overflow.

El compilador suele detectar si se asigna directamente un número fuera de rango a una variable, pero no puede controlar todas las operaciones que se realizan sobre esa variable y que pueden llevar a un desbordamiento.

En el caso de los reales además de que el número sea demasiado grande (tanto en positivo como en negativo) también puede darse el caso de que la parte decimal sea demasiado pequeña para representarse. En este caso se producirán errores de redondeo o de precisión en las operaciones.

2.4 Variables

Las variables son una de las características fundamentales de los lenguajes de programación, permiten acceder a la memoria para almacenar y recuperar los datos con los que nuestros programas van a trabajar. Son por tanto el mecanismo que los lenguajes de programación ponen a nuestra disposición para acceder a la memoria.

Se trata de un mecanismo de lo más sencillo, sólo tenemos que dar un nombre a nuestras variables, a partir de ese momento el compilador traducirá de forma automática ese nombre en un acceso a memoria. Por ejemplo:

```
//Almacenamos un dato en memoria referenciado por el nombre edad
edad = 5;
//Recuperamos el dato almacenado y lo modificamos
edad = edad + 1;
```

2.4.1 Declaración de variables

Java es un lenguaje tipado y nos obliga a declarar nuestras variables antes de poder hacer uso de ellas, con esta declaración le indicamos al compilador el espacio en memoria que debe de reservar para almacenar la información. Por ejemplo:

```
int numero;
```

Aquí estamos reservando memoria para una variable de tipo *int* y la identificamos con el nombre “numero”. De ahora en adelante si en el programa hablamos de numero, estamos haciendo referencia a esa porción de memoria y al valor que contiene.

Podemos asignarle algún valor en el momento de declarar una variable. Por ejemplo:

```
int numero = 25;
```

La sentencia para declarar una variable se resume como:

```
Tipo_de_Dato Nombre_de_Variable [= Valor_inicial];
```

Definimos el tipo de dato, el nombre y opcionalmente su valor inicial.

2.4.2 Identificadores

Se llaman identificadores los nombres de las variables, clases, objetos y métodos de los programas Java.

No pueden ser identificadores las palabras claves ni las palabras reservadas del lenguaje JAVA.

La siguiente tabla resume los nombre que se les puede asignar a las variables:

<table>
<tr><th colspan="3">VÁLIDO</th><th>NO VÁLIDO</th></tr>
<tr><td rowspan="3">Comienzo con letra</td><td rowspan="3">Comienzo con guión bajo _</td><td rowspan="3">Comienzo con $</td><td>Palabras claves o reservadas</td></tr>
<tr><td>No debe contener los símbolos que se utilicen como operadores (+ , - , ?, etc)</td></tr>
<tr><td>Cualquier palabra que empieze con símbolos distintos de letras, guión bajo _ ó $</td></tr>
</table>

Las **palabras claves** son: abstract, assert, boolean, break, byte, by value, char, case, catch, class, const, continue, default, do, double, else, enum, extends, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, thread, safe, super, throw, throws, transient, try, volatile, while

Además de estas palabras clave, tampoco se pueden usar las siguientes palabras por estar **reservadas**: true, false y void.

Convenciones de nomenclatura. Aunque se tiene libertad total para elegir los identificadores de las variables, al igual que ocurre con los nombres de las clases de Java, hay una convención de código para nombrar las variables en Java. En este caso hay que usar **lowerCamelCase** (notación de camello minúscula). Esta notación es como UpperCamelCase, pero en este caso **la primera palabra siempre en minúscula** y no en mayúscula. Ejemplos de identificadores de variables que siguen la convención son: divisor, resultadoDivision, miNumeroEntero, etc.

2.4.4 Constantes

Una constante es una variable del sistema que mantiene un valor inmutable a lo largo de toda la vida del programa. Las constantes en Java se definen mediante el modificador final. La estructura sería:

```
static final nombreConstante = valor;
```

De esta forma si queremos definir las constantes DIAS_SEMANA o DIAS_LABORABLES, que sabemos que son variables que no cambiarán su valor a lo largo del programa, generaremos el siguiente código:

```
static final int DIAS_SEMANA = 7;
static final int DIAS_LABORABLES = 5;
```

Por convención, como se puede ver en los ejemplos anteriores, las variables se escriben en mayúsculas, separando las palabras con barra baja.

Tened también en cuenta que las constantes solo se pueden declarar dentro de la clase, pero fuera de cualquiera de los métodos que hay (de momento el único método que estamos usando es el principal *main*).

Por ejemplo:

```
public class ConversionError {
    public static final double CONVERSIO_EURO_A_DOLAR = 1.3656;
    public static void main(String[] args) {
        double valor = 12.0;
        System.out.println(valor*CONVERSIO_EURO_A_DOLAR);
        valor = 300.0;
        System.out.println(valor*CONEVRSIO_EURO_A_DOLAR);
    }
}
```

2.4.5 Conversión de tipos

Una conversión de tipo es la transformación de un tipo de dato a otro tipo diferente.

Dentro de los lenguajes de programación hay dos tipos diferentes de conversiones de tipo: las implícitas y explícitas. El primer caso corresponde a aquellas conversiones fáciles de resolver y que el lenguaje de programación es capaz de gestionar automáticamente sin problemas. El segundo caso es más complejo y lo realiza el programador, si es posible.

2.4.5.1 Conversión implícita

La conversión de implícita, también llamada automática o de ampliación, tiene lugar cuando dos tipos de datos se convierten automáticamente. Esto sucede cuando:

- Los dos tipos de datos son compatibles.
- Cuando asignamos el valor de un tipo de datos más pequeño a un tipo de datos más grande.

Las posibles conversiones implícitas son:

Nuevo tipo	Tipos origen
short	byte
int	byte, short, char
long	byte, short, char, int
float	byte, short, char, int, long
double	byte, short, char, int, long, float

Hay que tener en cuenta el tipo de dato char, que aunque no es estrictamente numérico, si tiene asignado el valor de la posición del carácter dentro de la tabla unicode, y se puede convertir directamente en un int, long, float o double.

2.4.5.2 Conversión explícita

En la conversión explícita, también llamada *cast* o *casting*, es tarea del programador especificar el nuevo tipo al que se va a transformar el dato. Se escribe de forma explícita entre paréntesis delante del dato. Por ejemplo:

```
byte a = 20;
int x = (int) a;
```

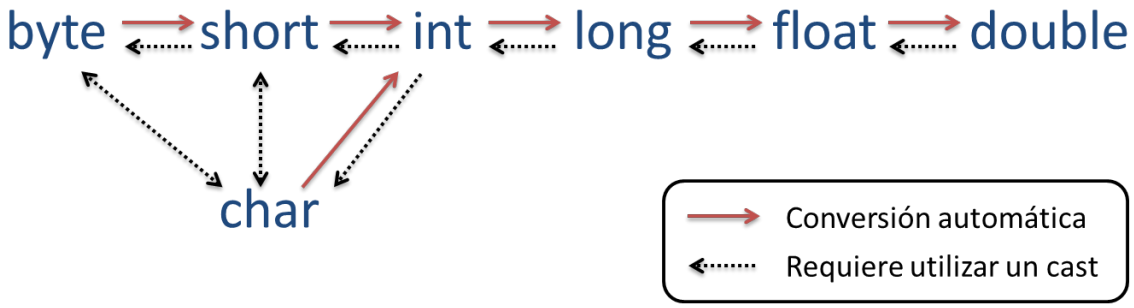
Al escribir int entre paréntesis se fuerza a cambiar el dato de tipo byte a int.

Hay que tener cuidado al realizar esta conversión ya que se puede aplicar a tipos no compatibles, lo que puede derivar en pérdidas de información e incluso errores en ejecución. Por ejemplo, en la siguiente conversión:

```
float x = 5.7F;
int y = (int) x;
```

Se realizará la conversión, pero se perderá la parte decimal del número con punto flotante al guardarlo en y. Solo se guardará 5 en y. Aunque parezca ilógico, habrá ocasiones en las que este tipo de conversiones nos pueden resultar útiles.

La siguientes imagen resume los distintos tipos de conversiones que se pueden realizar entre los tipos primitivos de datos.



Visualización y entrada de datos en Java

Instrucciones de salida de datos por pantalla

Cadenas de texto

Operadores de cadenas de texto

Caracteres de control

Entrada simple de datos por teclado