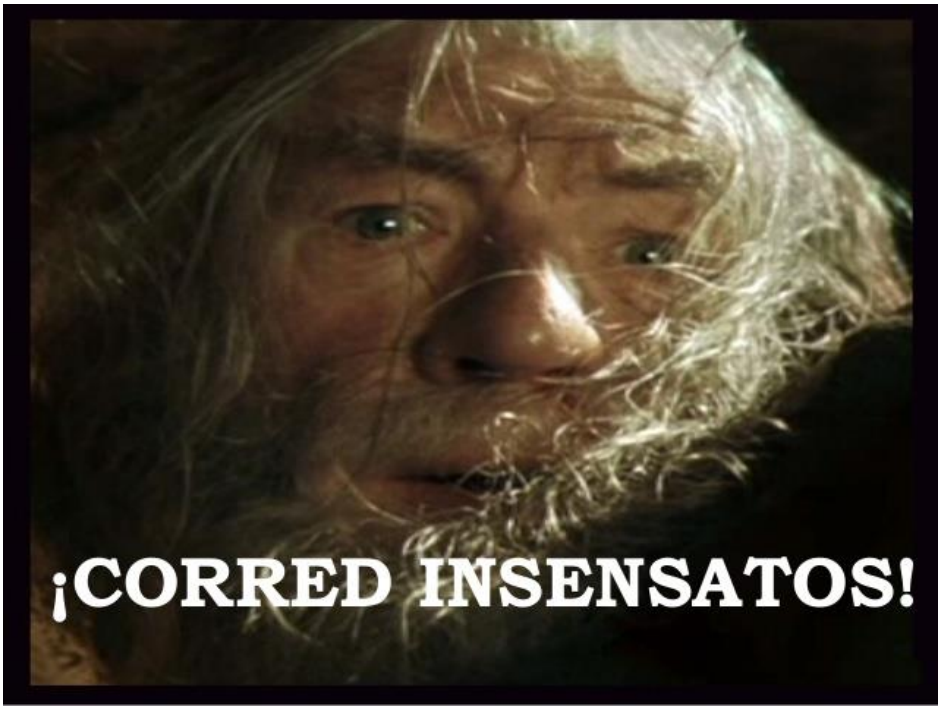
UNIDAD DIDÁCTICA 1: SINTAXIS BÁSICA

ÍNDICE



Visualiza este video y ve cambiando mentalmente la palabra Matrix en este video por las palabras "la programación". Esta será tu vida a partir de hoy. El mundo real de la programación puede que no te guste, pero es la realidad, sabrás qué se esconde detrás de todas esas Apps que han puesto delante de ti y que consumes sin conocer a fondo.

https://www.youtube.com/watch?v=SJrkhNskaUs

Todo el mundo debería aprender a programar:

https://www.youtube.com/watch?v=Y1HHBXDL9bg

1. ¿Qué es un programa? ¿Y un lenguaje de programación?

Seguro que alguna vez hemos usado el ordenador para escribir un texto o para entretenernos (demasiado tiempo) con algún juego. Para hacerlo hemos tenido que poner en marcha el procesador de texto o el juego en cuestión. Pues bien, las dos cosas son programas de ordenador.

Poner un programa en marcha es sinónimo de *ejecutarlo*. Cuando ejecutamos un programa sólo vemos los resultados, pero no el guion seguido por el ordenador para conseguir esos resultados. Este guion es realmente el programa.

A partir de ahora, seremos nosotros los que escribiremos el guion, es decir, sabremos cómo trabaja y por qué trabaja de esa forma. Vamos a pasar de usuarios (que utilizan) a programadores (que crean).

Si queremos enseñar a alguien a jugar a un juego de ordenador, le explicamos qué tiene que hacer, las normas y pasos a seguir. Básicamente esto es lo que hace un programa de ordenador. Un programa es una serie de instrucciones dadas al ordenador en un lenguaje entendido por él para decirle exactamente lo que queremos que haga. Si el ordenador no entiende alguna instrucción nos lo comunicará mediante mensajes visualizados por pantalla. Si esos pasos no son correctos puede pasar:

Preparad por parejas un papel con varias órdenes concisas para realizar una tarea en clase (puede ser escribir algo en la pizarra, coger algo, ir a algún compañero para pedir alguna cosa...). Tratad de ser lo más concretos posible.

Llenar vaso de agua:

https://www.youtube.com/watch?v=qHqDpUpbpJI

Bugs:

https://www.youtube.com/watch?v=xAfzQUqI_ac

Los pasos a seguir en la realización de un programa, una vez que ya tenemos claro qué queremos hacer (o hemos solucionado el problema que queremos solventar con un programa), son los siguientes:

a) Editar el programa. (Escribir el código).
b) Compilarlo.
c) Ejecutarlo.
d) Depurarlo.

Para todo ello, usaremos herramientas que facilitan el trabajo llamadas *Entornos de Desarrollo Integrado* (IDE). *(Se analizarán con profundidad en el correspondiente módulo del curso).*

¿Qué es un lenguaje de programación?

Un lenguaje de programación es un lenguaje formal diseñado para expresar procesos que pueden ser llevados a cabo por máquinas como los ordenadores.

Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana.

Está formado por un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Al proceso por el cual se escribe, se prueba, se depura, se compila (de ser necesario) y se mantiene el código fuente de un programa informático se le llama programación.

Veamos algunos listados de los lenguajes más usados hoy día.

TIOBE Index for September 2020

September Headline: Programming Language C++ is doing very well

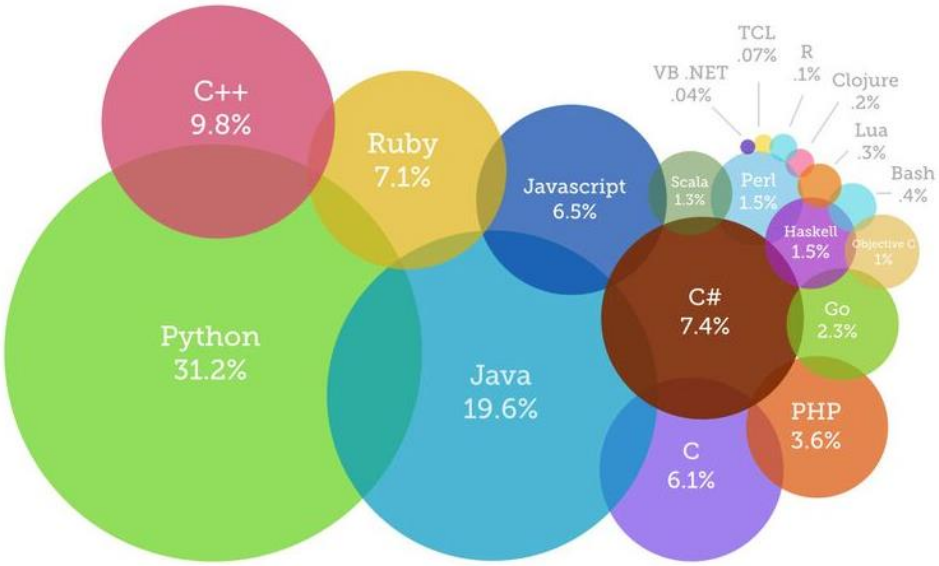
Back in 2003, the programming language C++ was a real winner. It peaked at 17.53% in August 2003, being close to the number #2 position and becoming winner of the programming language award of 2003. From then on C++ went downhill. After 2005 it didn't hit the 10% any more and in 2017 it scored an all time low of 4.55%. But if compared to last year, C++ is now the fastest growing language of the pack (+1.48%). I think that the new C++20 standard might be one of the main causes for this. Especially because of the new modules feature that is going to replace the dreadful include mechanism. C++ beats other languages with a positive trend such as R (+1.33%) and C# (+1.18%). On the other hand, Java is in real trouble with a loss of -3.18% in comparison to last year. - *Paul Jansen, CEO TIOBE Software*

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular search engines such as Google, Bing, Yahoo!, Wikipedia, Amazon, YouTube and Baidu are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	↑	C	15.95%	+0.74%
2	1	↓	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	↑	PHP	2.49%	+0.62%
9	19	⇈	R	2.37%	+1.33%
10	8	↓	SQL	1.76%	-0.19%

Java subió mucho en 2015 y desde 2008 no se veía tanta diferencia entre Java y el segundo. Los tres primeros no cambian desde el 2001. Esto se debe a que la versión 8 de java fue un éxito. También puede deberse a que, desde que Oracle compró la compañía ha trabajado mucho en Java.



¿Qué lenguajes deberíamos saber para trabajar como programadores?

Antes de nada, sabed que es imposible conocer en profundidad todos los lenguajes de programación, tecnologías, Frameworks y librerías que existen. Un listado para ir "enfocando" nuestro futuro es el siguiente.

https://www.adslzone.net/2017/12/05/lenguajes-de-programacion-trabajo-2018/

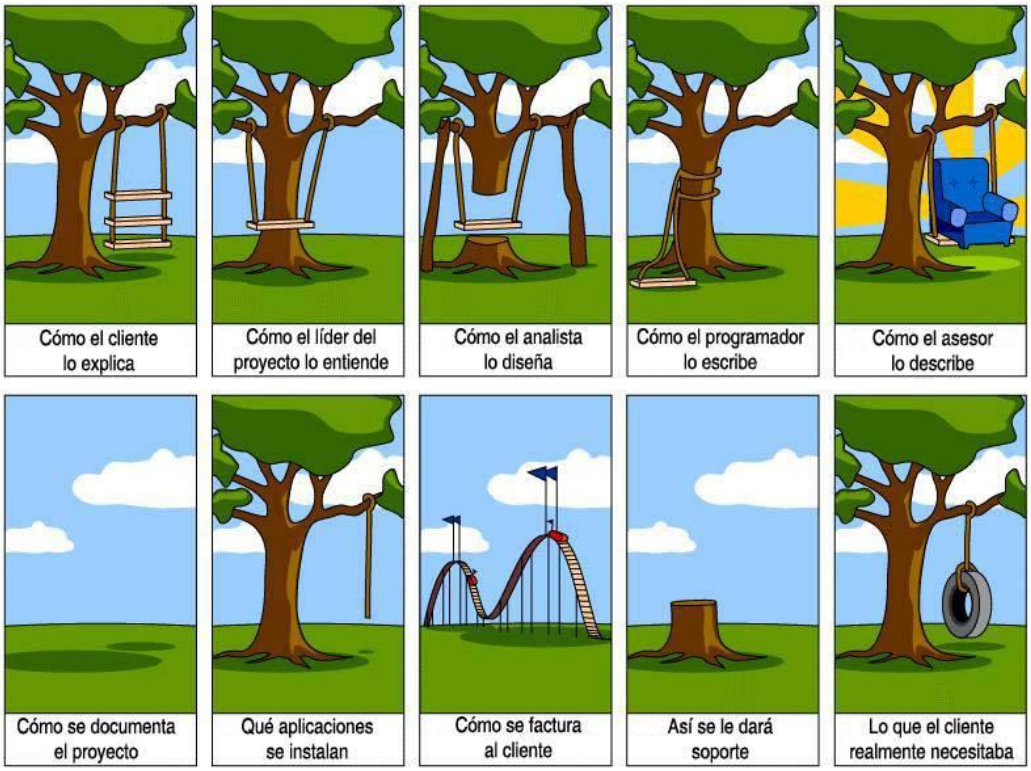
https://www.devsaran.com/blog/10-best-programming-languages-2019-you-should-know

¿Cómo crear un programa?

Abraham Lincoln dijo: "Si tuviera 8 horas para cortar un árbol, 6 de ellas las pasaría afilando el hacha". No me cansaré durante todo el año en repetirlo, llegando a la saciedad: el análisis del problema y el planteamiento de la solución de un problema es lo MÁS, MÁS, MÁS, MÁS... IMPORTANTE de cualquier programa.

Luego para empezar a crear un programa, primero debemos tener claro lo siguiente:

EL ANÁLISIS: Es importantísimo hacer un buen análisis de cuál es específicamente el problema a resolver. Para esto es bueno ayudarse mediante gráficos del problema o en caso de que no sea posible expresar con gráficos, también se puede resolver el problema para casos específicos y luego generalizarlo para todos los posibles casos.



También se deben observar cuáles serían los casos especiales, es decir, aquellos casos que no cumplan la norma general, y tratar de evaluarlos de otra forma.
Este paso es el que más tiempo debe llevarle a un buen programador, ya que de un buen análisis dependen los buenos resultados que arroje el algoritmo.

Preguntas como: **¿Qué es lo que realmente queremos resolver? ¿Cuáles son nuestros datos de entrada? ¿Cómo los vamos a pedir al usuario? ¿Cuándo un dato de entrada no será válido? ¿Cómo obtendremos la salida?...** nos pueden servir para aclararnos.

ESCRIBIR EL ALGORITMO: Después de haber analizado el problema en una forma abstracta, se debe llevar al papel, mediante instrucciones adecuadas al análisis. Si el problema fue bien analizado, este paso es muy rápido a comparación del anterior.

PRUEBA DE ESCRITORIO: Este paso es opcional y se aplica siguiendo paso por paso las instrucciones del algoritmo, anotando los diferentes valores que van tomando las variables, de forma que se pueda verificar si hay errores en alguna instrucción.
Obviamente, este método es muy engorroso para algoritmos muy extensos, por lo que en estos casos no sería aplicable.

Veremos algo de pseudocódigo y diagramas de flujo en la unidad 2 de este curso.

Por parejas, uno hará de cliente y otro de "programador" para una toma de datos real.

- **Queremos hacer un programa que avise al usuario cuando le queda menos de 1 € en la cuenta del banco.**
- **Programa que compruebe la contraseña para mostrar una página.**
- **Un programa que divida dos números enteros.**
- **Código para imprimir el nombre de un usuario en un texto, por ejemplo, Hola Ángel, ¡eres maravilloso!**
- **Programa que compruebe la mayoría de edad de un usuario.**
- **programa para comprobar que una persona no ha bebido, es decir, que no sobrepasa el 0,25 de alcohol en sangre.**
- **Aviso en máquina de refrescos que no se ha introducido el suficiente dinero para un producto**

2. Entorno de Desarrollo

Aunque este apartado se tratará profundamente en el módulo "Entornos de Desarrollo" de este ciclo, veremos qué programas son necesarios para comenzar a trabajar con código Java.

En primer lugar, comprobamos si en nuestro PC está instalado el JDK.

Es necesario descargar e instalar (si no está ya en el ordenador):

✓ **En la página web de Oracle**, iremos a "downloads" y descargaremos, de las Java SE downloads el JDK:

http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

- Dentro del apartado Java Platform Standard Edition descargamos la última versión JDK:

✓ **De la página web de "Eclipse" descargamos la última versión de Eclipse (normalmente detecta el sistema operativo de tu ordenador, pero debemos asegurarnos).**

https://eclipse.org/downloads/

- No descargar de la parte donde pone "Get it faster from our members". Meten pluggins que no nos hacen falta.

- No es necesario instalarlo, basta pinchar en el instalador, descomprimir el archivo y copiar la carpeta donde queramos. Luego podemos crear un acceso directo ya que lo usaremos a diario.

Instalamos primero el JDK y posteriormente, instalamos eclipse. En este caso, elegiremos el Eclipse IDE EE (Enterprise Edition).

3. Nuestro primer programa

Un programa se compone de muchas líneas escritas, cada una de las cuales realiza una función. Cuando se compila el programa, se comienza a leer línea a línea, desde la primera a la última, comenzando por arriba y hacia abajo.

Por ejemplo, en este caso, aunque de momento no entendamos qué significa cada cosa, la primera línea declara la clase pública de nombre **HelloWorld**, la siguiente es el método principal de la aplicación (de ahí el nombre de "**main**" y la tercera, llama al método que imprime por pantalla el texto que se encuentra entrecomillado dentro del paréntesis "**Hello World**".

Seguro que todos los programadores del mundo, han escrito alguna vez este programa y lo recuerdan con cariño. Nosotros no vamos a ser menos.

Vamos a hacer un minitutorial de cómo se hace un "Hola mundo" en consola usando eclipse. Hay que empezar desde cómo crear un proyecto, hasta cómo ejecutarlo enseñando el resultado. Se puede usar cualquier herramienta, presentación, en video, pdf, esquema en la pizarra y luego una foto, etc.

4. Datos y variables en el lenguaje Java.

Podemos definir dato como **un conjunto de símbolos que representan valores, hechos, objetos o ideas de forma adecuada para ser tratados**. Incluso, en el ámbito de la informática, podemos definir dato de forma similar, como cualquier "objeto" manipulable por el ordenador.

Un dato puede ser un carácter leído desde teclado, un número, la información almacenada en un CD, una foto almacenada en un fichero, una canción, el nombre de un alumno almacenado en la memoria del ordenador, etc.

Los ordenadores son máquinas creadas para **procesar automáticamente la información**, y para ello esa información no se almacena ni representa al azar, sino que **ha de organizarse y estructurarse de forma adecuada para que pueda almacenarse, procesarse y recuperarse de la forma más eficiente posible**.

a) Tipos de datos

Usaremos datos asociados normalmente a constantes y a variables.
Tanto constantes como variables son **zonas de la memoria del ordenador** a las que se le asigna un nombre (identificador), a modo de etiqueta, y a las que se les asocia un tipo de dato, de forma que sólo se podrán almacenar en esa zona de memoria (o posición de memoria) datos de ese tipo.

La diferencia:

Las *constantes* reciben el valor una vez, normalmente al principio del programa, y ya no puede volver a escribirse nada en la posición de memoria que ocupan (una vez que se les asigna el valor, no se puede modificar, permanece constante).

Las *variables* pueden cambiar de valor tantas veces como se desee a lo largo del programa (podemos volver a escribir otro valor en la zona de memoria que ocupan tantas veces como deseemos).

Veamos unos ejemplos de cómo se gestiona la memoria cuando se usan variables:

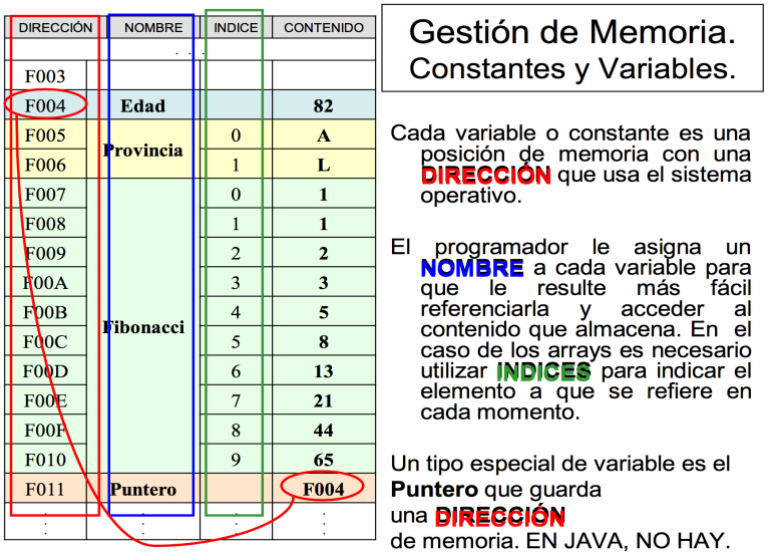
DIRECCIÓN	NOMBRE	INDICE	CONTENIDO
	. . .		
F003			
F004	**Edad**		**82**
F005	**Provincia**	0	**A**
F006		1	**L**
F007	**Fibonacci**	0	**1**
F008		1	**1**
F009		2	**2**
F00A		3	**3**
F00B		4	**5**
F00C		5	**8**
F00D		6	**13**
F00E		7	**21**
F00F		8	**44**
F010		9	**65**
F011	**Puntero**		**F004**
⋮	⋮	⋮	⋮

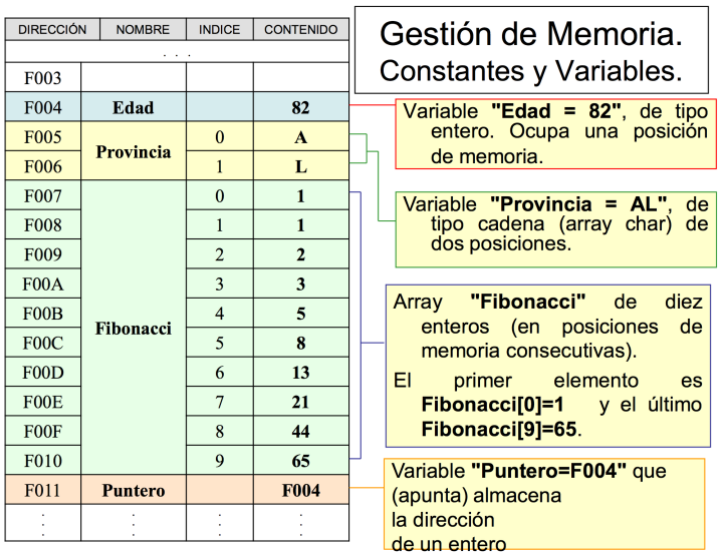
Gestión de Memoria. Constantes y Variables.

El contenido de las variables en memoria siempre son valores numéricos según el código UNICODE.

Así por ejemplo según la imagen, en la posición F005 que se ha llamado Provincia[0] realmente se almacena el valor:

(hexadecimal) $0041)_{16}$ =
(binario) $0000\ 0000\ 0100\ 0001)_2$ =
(decimal) $65)_{10}$ (A en Unicode es 65)





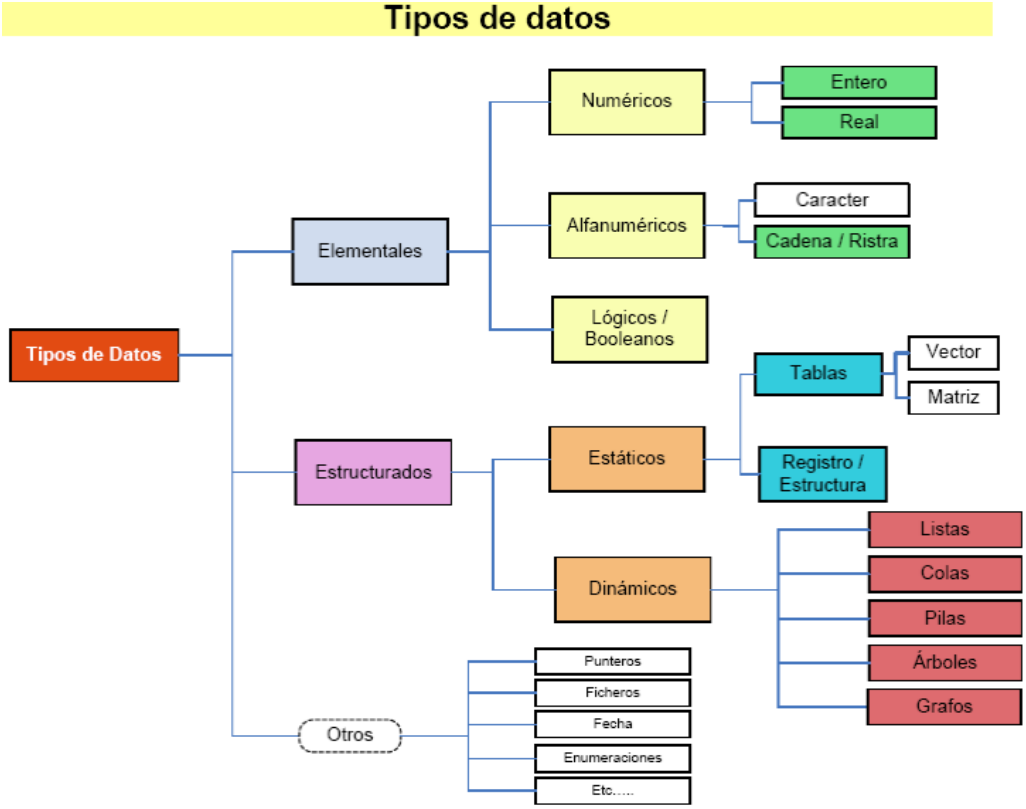
En definitiva, **para usar los datos, tenemos que disponer de ellos en la memoria del ordenador**. Es como si la memoria fuese un enorme casillero, en el que cada casilla puede almacenar un dato, y tiene una dirección asignada (realmente un número binario) y puedo referirme a una casilla en concreto y al valor que contiene a través de su dirección.

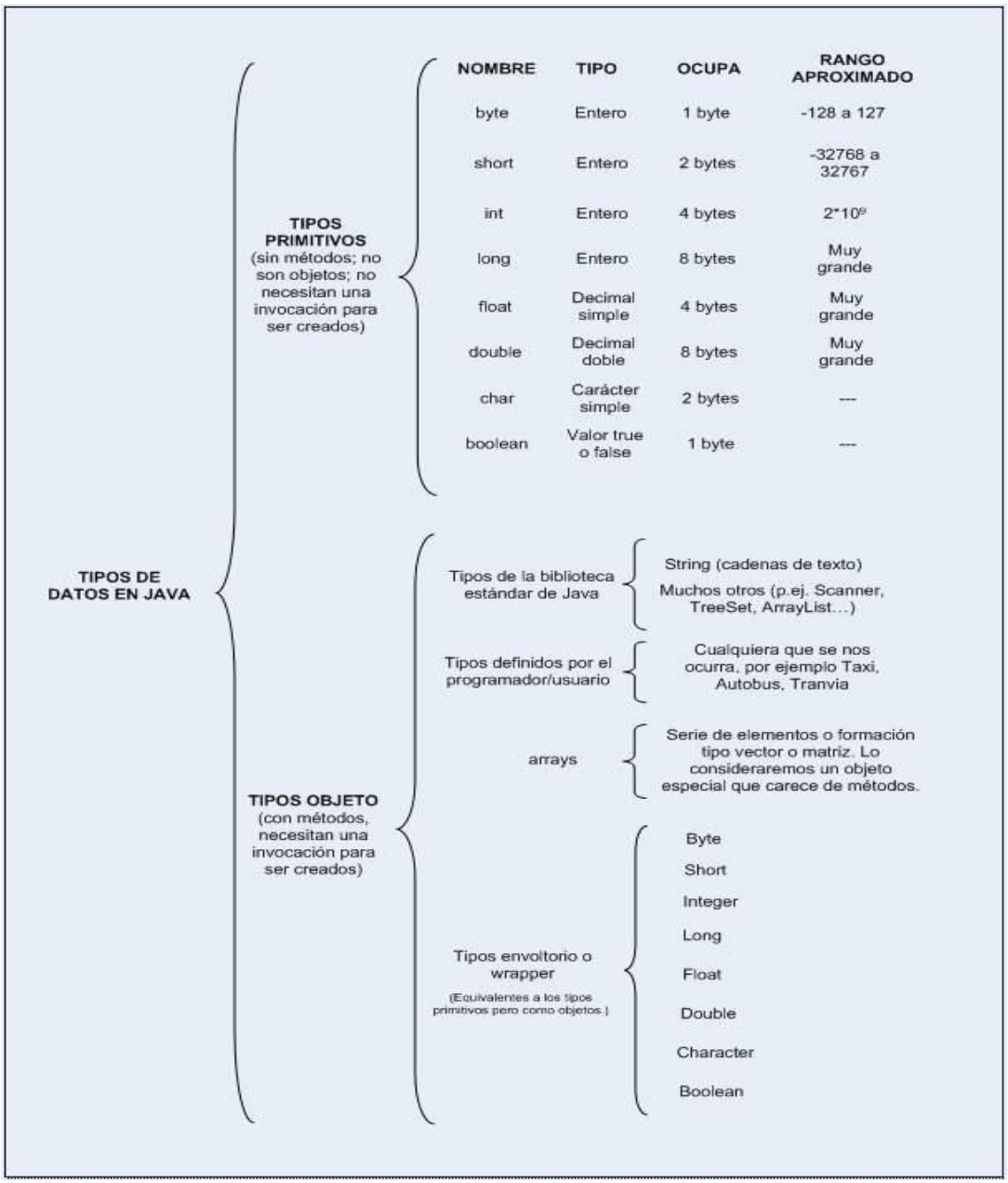
Así se hace de hecho cuando se programa en código maquina o ensamblador, pero eso es sumamente dificultoso, y requiere conocimiento de los detalles de la arquitectura del ordenador, de su funcionamiento interno, que no queremos manejar nosotros. Por eso, en los lenguajes de alto nivel, se asocia un nombre de variable o constante con una de estas direcciones de memoria. **Cada vez que en el programa hagamos mención al nombre de la**

variable o constante, el compilador sustituirá ese nombre (esa "etiqueta") por la dirección de memoria (en binario) que tenga asociada. Para nosotros es mucho más fácil de entender, por ejemplo, que a la variable *"edad"* se le suma 1 que entender que al dato almacenado en la posición de memoria 1000 1100 1010 1010 0100 1110 se le suma 1. Es muchísimo más descriptivo y más claro usar el nombre de la variable, al que llamamos ***identificador***, que la dirección de memoria asociada.

Pero también hemos mencionado que la variable (o constante), además del nombre lleva asociado un tipo de dato. ¿Qué es eso? Un tipo de dato no es más que una especificación de los valores que son válidos para esa variable y de las operaciones que se pueden realizar con ellos

b) Datos primitivos.





¿Qué son los datos simples, elementales o primitivos? ¿Acaso son datos poco perfeccionados, o que no puedan usarse para cálculos complicados?
No exactamente. Son los datos que suele proporcionar el lenguaje de programación, y que por eso de ellos conocemos:

§ Cómo se representan internamente (en memoria).
§ El tamaño exacto que ocupan.
§ Los operadores que define el lenguaje para ellos.
§ Que están disponibles sin necesidad de definir nada por parte del usuario.

Son un conjunto mínimo de tipos de datos, de forma que **a partir de ellos se construirán los demás tipos de datos**, que por ello recibirán el nombre de datos estructurados.

En los siguientes apartados veremos cuáles son los principales tipos de datos simples (también llamados a veces *tipos primitivos* de datos).

CUADRO DE TIPOS DE DATOS PRIMITIVOS EN JAVA

Tipo	Descripción
boolean	Permite representar valores lógicos; Verdadero (V) o Falso (F).
char	Permite representar un símbolo o carácter UNICODE de 16 bits.
byte	Entero de 8 bits (1 byte) con signo (representado en complemento a dos). Su rango devalores va desde -128 a +127.
short	Entero de 16 bits con signo (complemento a dos). Rango de valores entre -32.768 y +32.767.
int	Entero de 32 bits con signo (complemento a dos). Rango de valores entre -2.147.483.648 y +2.147.483.647.
long	Entero de 64 bits con signo (complemento a dos). Rango de valores entre -9223372036854775808 y +9223372036854775807.
float	Número real (en coma flotante) de 32 bits, utilizando la representación IEEE 754-1985 (1 bit de signo, 8 para el exponente y 24 para la mantisa). Precisión aproximada de 7 dígitos.
double	Número real (en coma flotante) de 64 bits, utilizando la representación IEEE 754-1985 (1 bit de signo, 11 para el exponente y 52 para la mantisa). Precisión aproximada de 16 dígitos.

Veamos un ejemplo:

```
1  public class Primitivos{
2      public static void main(String arg[]){
3          /*Declaracion con primitivos*/
4          int datoInt=40;
5          double datoDob = 3.14159;
6          boolean bool1 = true;
7
8          /*Mostrar los valores en pantalla*/
9
10         System.out.println("el valor de datoInt es: "+datoInt);
11         System.out.println("el valor de datoDob es: "+datoDob);
12         System.out.println("el valor de bool1 es: "+bool1);
13     }
14 }
```

1. BUSCAR EN INTERNET INFORMACIÓN Y UNA TABLA CON EL JUEGO DE CARACTERES ASCII. DESPUÉS, BUSCA EN INTERNET CÓMO SE PUEDE ESCRIBIR UNA LETRA (TIPO DE DATO CHAR) EN JAVA USANDO DICHA TABLA.

2. HAZ UN PEQUEÑO PROGRAMA PARA IMPRIMIR TUS DATOS PERSONALES A MODO DE TARJETA DE VISITA. PON TODO EL TEXTO A IMPRIMIR POR LA PANTALLA DENTRO DE UNO O VARIOS MÉTODOS PRINTLN ().

EJEMPLOS DE ELECCIÓN DE TIPOS

Dato	Descripción	Tipo	Justificación
Estado_Civil	Queremos saber el estado civil de una persona	Char	Son posibles varias respuestas, C (casado/a), S (soltero/a), D (divorciado/a), V (viudo/a), etc.
Mayoría de edad	Queremos saber si alguien es mayor de edad	Boolean	Solo son posibles dos respuestas, por lo que boolean es el tipo adecuado.
Día de la semana	Se expresará como número del 1 al 7, pero tenemos que tener presente que **hay problemas de capacidad de memoria para el ordenador en que se va a ejecutar la aplicación**, por lo que debemos ahorrar todo el espacio de almacenamiento que podamos	byte	Permite representar números enteros entre -128 y127, por lo que es más que suficiente para los valores de 1 a 7 que queremos guardar. Además, sólo ocupa 8 bits. Es el tipo entero más pequeño que podemos usar.
Día del año	Se expresará como un número entre 1 y 366. Al igual que antes, **queremos aprovechar al máximo el espacio de almacenamiento.**	short	Permite representar números enteros entre-32.768y 32.767, por lo que es más que suficiente para los valores de 1 a 366 que queremos guardar. Sin embargo, el tipo byte no proporciona un rango suficiente. Además, sólo ocupa 16 bits. Es el tipo entero más pequeño que podemos usar para este conjunto de valores.
Milisegundos	Número de milisegundos que han trascurrido desde las 00:00:00 horas del día 1 de Enero de1970 hasta nuestros días.	long	Es una cantidad considerable, que no podemos almacenar en un int, pero que sigue siendo un número entero en el rango de long. De hecho, esa es la cantidad que usa Java para expresar una fechas.

Sueldo mensual	Queremos expresar el sueldo de cada empleado en euros, con precisión de céntimos. Suponemos que seguimos con limitaciones de memoria.	float	Es el tipo de número real (con decimales) de menor precisión y que menos memoria ocupa. No obstante, su rango de valores permite representar el sueldo de cualquier trabajador de la empresa, por elevado que éste sea. Además, tampoco necesitamos más precisión, ya que para los céntimos, con dos decimales nos basta.
Número π	Número Pi. Necesitamos almacenar el valor del número pi para una aplicación que realiza cálculos científicos de gran precisión. Necesitamos poder representar el mayor número posible de decimales del número Pi, para no perder precisión en los cálculos.	double	Es el tipo real que mayor precisión nos proporciona. El número Pi es un número real irracional, es decir, con infinitos decimales. Con double podremos representar de forma exacta el mayor número posible de ellos.

Escribe 3 posibles datos y pregunta a tu compañero qué tipo de variable elegiría para cada uno de ellos. Después haremos una puesta en común.

b.1) Enteros

¿Qué conjunto de números fue el primero que estudiaste en la escuela, cuando aprendías a contar? Si recuerdas bien, fueron los números Naturales, del cero en adelante. Pero rápidamente aprendiste que había operaciones como la resta que no siempre podían tener solución dentro de los naturales, así que se era necesario buscar un conjunto más grande, el de los números Enteros, que contenía a los Naturales, pero también a los números negativos. Y con ese conjunto ya eras capaz de resolver gran cantidad de problemas ¿no?

Los datos de tipo entero permiten representar números enteros (sin decimales) tanto positivos como negativos. La mayoría de los lenguajes definen varios tipos de enteros, que se diferencian fundamentalmente en el número de bytes que se usan para su representación. Mientras más bytes usemos para representar un número entero, mayor será el rango de números representable.

Por ejemplo, en Java, existen **cuatro tipos de números enteros** (en algunos libros también incluyen en el tipo entero a los char):

Nombre del tipo entero	Tamaño usado para su representación (bits)	Total de número distintos representables	Menor número representable para el tipo	Mayor número representable para el tipo
byte	8= 1 byte	2_8= 256	-128	+127
short	16=2 bytes	2_{16}=65536	-32.768	+32.767
int	32=4 bytes	2_{32}=4.294.967.296	-2.147.483.648	+2.147.483.647
long	64= 8 bytes	2_{64}= 1,844674407 E+19	-9.223.372.036.854.775.808	9.223.372.036.854.775.807

¿Cuándo definiremos una variable como de un tipo u otro? *Dependerá del tipo de datos que manejemos en nuestro problema. Si conocemos que el valor máximo o mínimo de nuestros datos nunca va a sobrepasar el máximo o el mínimo de un tipo, definiremos la variable de ese tipo. Mientras menor sea el número de bits que usa la representación del tipo, menos memoria ocupa, y por tanto mejor estaremos aprovechando los recursos del ordenador. Hay que pensar que en principio la memoria es siempre un bien escaso en un ordenador. De todas formas, en general,* ***la mayoría de los tipos enteros se definen como int y así lo haremos nosotros como normal general.***

Ejemplos:

Tipo de dato	Código
byte	byte a;
short	short b, c=3;
int	int d = -30; int e = 0xC125;
long	long b = 434123; long b = 5L; *//La L en este caso significa Long*

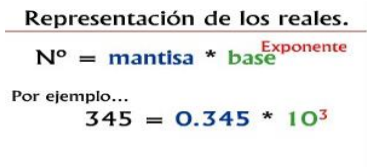
Crear un programa y declarar varias variables de tipo entero. Pueden ser las de los ejemplos anteriores. Imprimir los valores de las variables por pantalla con la sentencia:
System.out.println (nombre de la variable);

Repetir usando texto antes del valor de la variable.

b.2) Reales

Después de los enteros, en la escuela, estudiaste los números racionales y los números con decimales, es decir, los números reales. Básicamente nos permiten solucionar todos los problemas de nuestra vida diaria en cuanto a los números se refiere.
Los datos de tipo real se usan para representar números reales (con decimales). Al igual que ocurre con los enteros, la mayoría de los lenguajes definen más de un tipo de datos real, que se diferencian igualmente entre sí por el número de bits que se usan para representarlos y almacenarlos.

Se representan como indica la figura:
Siempre, de tal forma que la mantisa tenga parte entera igual a cero. Así, tanto la parte entera como la base (que siempre es 10) no se almacenan.



Internamente hay algunos problemillas (veces que no se cumple la propiedad distributiva, etc.) con la representación de estos números, pero no lo estudiaremos. Sólo nos debe quedar claro, que el orden de las operaciones influye en el resultado.
Veremos con más detalle el orden cuando estudiemos los operadores.

En java, se definen dos tipos de números reales dependiendo de la cantidad de bits usados para su representación. Son estos:

Nombre del tipo real	Tamaño usado para su representación (bits)	Total de número distintos representables	Menor número representable para el tipo (en valor absoluto)	Mayor número representable para el tipo (en valor absoluto)
float	32= 4 bytes	2_{32}= 4.294.967.296	1.401298464324817 E-45	3.4028234663852886 E+38
double	64=8 bytes	2_{64}=1,844674407 E+19	4.9 E-324	1.7976931348623157 E+308

Como norma general, nosotros emplearemos siempre el tipo double.

La diferencia entre usar 8 bytes para cada variable double en vez de los 4 bytes de float en un programa que use 10.000 variables de tipo real (lo cual es muchísimo) supone usar 80.000 bytes en vez de 40.000, lo que supone una diferencia de menos de 40 Kbytes de memoria.

Ejemplos:

Tipo	Código
float	float pi = 3.1416; float pi = 3.1416F; *//La F en este caso indica Float* float medio = 1/2F; *//0.5*
double	double millón = 1e6; *// 1x10⁶* double medio = 1/2D; *// 0.5 en este caso la D significa Double*

Investiga la forma de poder imprimir por pantalla con formato determinados números, por ejemplo, poder imprimir solo dos decimales cuando el número a imprimir tiene más (hay varias formas y, de momento, solo usaremos la relacionada con el método printf).

b.3) Lógicos o Booleanos.

¿Cómo podríamos representar internamente dos únicos valores posibles?
Por otra parte, ¿se te ocurre alguna operación posible con esos dos valores? Lo bueno de estos datos es que sólo existen dos posibilidades, por lo que se pueden almacenar usando únicamente un bit. (0 ó 1)

Casi todos los lenguajes incluyen un tipo de datos lógicos o booleanos (C no los incluye como tipos básicos). Son datos que sólo pueden tomar dos valores distintos, (verdadero o falso, si o no, 0 ó 1) Se utilizan normalmente para comprobar el valor de una condición, o una comparación de valores o para distinguir la ocurrencia de un suceso de entre dos posibles. Estos tipos de datos se definen con la palabra ***boolean***.

Ejemplo:

boolean casado = true;//También puede asignarse el valor false.
boolean registrado= false;

b.4) Caracteres

Otro de los tipos son los datos de tipo carácter. ¿Qué nos representa el tipo carácter?
Los datos de tipo carácter representan elementos individuales del conjunto de caracteres usado como código de entrada-salida. El código más usual es el ASCII, que usa 8 bits (1 byte) para representar cada carácter, por lo que permite usar hasta 256 caracteres distintos, lo cual es bastante limitado a la hora de representar todos los posibles símbolos de todos los idiomas y alfabetos existentes. Aunque es el más extendido, no es el único posible. Por ejemplo, Java ha

incorporado como alfabeto el código Unicode, más reciente que el código ASCII (de hecho, ASCII es un subconjunto de Unicode), y que usa 16 bits para representar cada carácter, por lo que puede representar hasta 65.536 caracteres distintos, lo que hace posible representar todos los alfabetos de todas las lenguas, y todo tipo de símbolos gráficos, matemáticos, caracteres de control, etc. Además, el código Unicode es "compatible" con el código ASCII, ya que para los caracteres del código ASCII, Unicode asigna como código los mismos 8 bits, a los que les añade

Tipo	Código
char	char car1 ='c'; char car2 = 99; *//car1 y car2 son lo mismo porque el número 99 en unicode es la letra 'c'*

a la izquierda otros 8 bits todos a cero. La conversión de un carácter ASCII a Unicode es inmediata.

En Java, el tipo carácter se llama **char**, y el alfabeto o código usado es Unicode. Tiene además la peculiaridad de que es considerado por el compilador como un tipo de entero, por lo que puede operarse con caracteres como si fuesen números enteros, ya que realmente se opera con su código Unicode, que es un número.

Podéis leer más cosas sobre ASCII y Unicode en los siguientes enlaces de la Wikipedia:

http://es.wikipedia.org/wiki/ASCII
http://es.wikipedia.org/wiki/Unicode

NOTA: Hay lenguajes que incorporan otro tipo básico, que no vamos a estudiar aquí como "tipo básico", aunque si los veremos como otro tipo de datos. Este es: el tipo "fecha" (en java son gestionados con clases como Calendar, GregorianCalendar...).
Lo estudiaremos más adelante.

c) Nombres de las variables.

El programador es libre de denominar a sus variables con el nombre que considere oportuno, siempre que cumpla las siguientes normas:

- El primer carácter del nombre debe ser una letra (a-z, A-Z) o el carácter "-" o el carácter "$". No pueden empezar con números.
- No pueden usarse como nombres de las variables las palabras reservadas de Java (los token, ver apartado 8).
- Los nombres de las variables deben ser continuos, es decir, no pueden tener espacios en blanco.
- Los identificadores son sensibles a las mayúsculas y a las minúsculas. Por ejemplo, la variable "caso" es distinta a la variable "Caso".

- Es MUY recomendable, establecer un estilo de denominaciones. Por ejemplo, las variables con minúsculas, las constantes en mayúsculas y las clases con la primera letra en mayúscula (las demás en minúscula).
- En las palabras compuestas para variables, se puede tomar el criterio de poner el primer carácter de la segunda palabra en mayúscula. Por ejemplo, *miCasa, numeroLibros, etc.*
- No es recomendable el uso de nombres muy largos. Ejemplo:

  Sería muy largo: estaeslaultimateclapulsada
  Bien: uTecla

- Los nombres deben indicar qué dato almacenan y ser un sustantivo (los verbos que indican acción se dejan para los métodos).

Nombre	Estado	Comentario
Índice	Legal	Es recomendable escribir en minúsculas. El uso de tildes en los nombres de las variables está permitido, aunque no es recomendable cuando se van a realizar programas que se ejecuten en distintos sistemas operativos. NOSOTROS NO LOS PONDREMOS.
33ventas	Ilegal	El nombre no puede empezar con un número.
juan2	Legal	Un número puede intercalarse en un nombre.
const	Ilegal	const es una palabra reservada y no puede usarse.
juan&pedro	Ilegal	El carácter & no puede usarse en el nombre. Es un operador.
constl	Legal	Una palabra reservada puede usarse como parte de otra.
i	Legal	Es legal pero sólo recomendable en contadores.
INDICE	Legal	No recomendado escribirla en mayúsculas. Sólo para constantes.
MisCompras	Legal	Se recomienda el uso de palabras descriptivas
$miCasa	Legal	Se puede usar el carácter $ como primer carácter
mi casa	Ilegal	No se pueden dejar espacios en los nombres.
mi_Casa	Legal	Se puede utilizar el carácter "_".

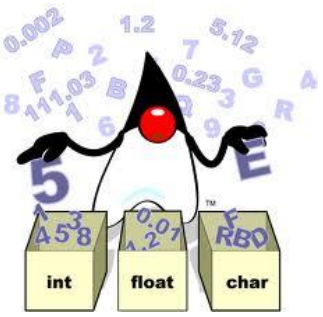
Ejemplos de nombres de variables

Inventa 5 nombres de variables, unos legales y otros ilegales. Compartiremos con el resto de compañeros algunos de ellos para comprobar si hemos aprendido esto.

d. Tipos de datos no básicos (solo de lectura excepto el apartado d.2 cadenas de caracteres)

Aunque a lo largo del curso veremos todos estos tipos de datos, se hace aquí una pequeña introducción de los mismos para tener una visión global de la cantidad de posibilidades que el lenguaje Java ofrece.

d.1) Datos estructurados



¿Se te ocurre algún tipo de dato de la vida real que sea difícil representar y procesar usando únicamente los tipos de datos básicos estudiados hasta ahora? Seguramente más de uno. Un empleado de la empresa es algo más que una serie de datos individuales, podemos verlo como una estructura que aglutina todos esos datos de tipo básico. Otro ejemplo podría ser una serie de valores que represente, con fines estadísticos, la altura en cm de una muestra de 5.000 personas, sobre los que tenemos que calcular variables estadísticas (media, moda, mediana, varianza, desviación típica, etc.) Usar 5.000 variables de tipo entero sería inmanejable. ¡Imagínate escribiendo la suma de las 5.000 variables en una expresión, cada una con su nombre! Es por eso que necesitamos estructuras de datos, que nos faciliten la tarea de gestionar gran cantidad de datos, o que sencillamente nos permitan representar los datos reales de una forma más fiel.

Además de los tipos simples, o básicos o primitivos, la mayoría de los lenguajes permiten usar una serie de estructuras de datos algo más complejas, que se construyen a partir de otros tipos, que pueden ser los propios tipos básicos o primitivos o ser a su vez tipos estructurados, o tipos definidos por el usuario. En este apartado se han incluido los tipos estructurados que suelen proporcionar ya definidos los lenguajes, aunque el usuario puede definir también sus propias estructuras de datos a través de la definición de clases, que son estructuras y podrían incluirse aquí.

NOTA: Gran parte del aprendizaje de un lenguaje consiste en saber usar sus tipos de datos tanto simples como estructurados, por lo que aquí sólo indicaremos brevemente en qué consisten estas estructuras, dejando una explicación más completa de su uso y en posteriores unidades.

d.2) Cadenas de caracteres

En general, **una cadena de caracteres es una sucesión de caracteres (letras, números y demás caracteres del alfabeto del lenguaje)**. Se utilizan normalmente como un tipo de dato predefinido, para palabras, frases o cualquier otra sucesión de caracteres. La palabra usada para definir una cadena es ***String***. Un **ejemplo** de valor posible para una cadena de caracteres es:

- el nombre de una persona.
- una frase de un libro.
- la contraseña de un usuario que accede a un curso a través de internet.
- el número de teléfono, que aunque no contenga más que dígitos entre sus caracteres, podemos almacenarlo como de tipo cadena de caracteres, ya que no vamos a realizar ningún tipo de operación matemática con él.

El valor se debe delimitar mediante comillas (") al principio y al final de la cadena. Como un **ejemplo** concreto podemos ver la cadena de caracteres “Esto es una cadena de caracteres”. Para poder mostrar, por ejemplo, una comilla (") dentro de la cadena y no tener problemas con las comillas que la delimitan, se usan **secuencias de escape** (Ver tabla siguiente). Esto se aplica a otros caracteres reservados o no imprimibles como el retorno de carro o salto de línea. No obstante, las expresiones para producir estas secuencias de escape dependen del lenguaje de programación que se esté usando.

Una forma común, en muchos, y en Java en concreto, de “escapar” un carácter es anteponiéndole un «\» (sin comillas). Por ejemplo:

"la cadena \"ejemplo\" contiene comillas "realmente será interpretada por el lenguaje como la cadena: la cadena "ejemplo" contiene comillas. Se muestran las principales secuencias de escape en la siguiente tabla:

NOTA: Las cadenas son un tipo de datos especial. La estudiaremos con más detenimiento más adelante en el tema de objetos y clases.

Secuencia de Escape	Descripción
\a	Sonido de alerta (Beep)
\b	Retroceso
\f	Salto de página
\n	Salto de línea
\r	Retorno de línea
\t	Tabulación horizontal
\v	Tabulación vertical
\\	Contra barra
\?	Interrogación
\'	Apóstrofe
\"	Comillas dobles
\0	Caracter nulo
\nnn	Número octal (nnn)
\xnnn	Número hexadecimal (nnn)

d.3) Arrays y matrices

Un *array* es un conjunto de variables o registros del mismo tipo que puede estar almacenado en memoria principal o en memoria auxiliar.

- Los arrays de 1 dimensión también se denominan ***vectores***.
- Los de 2 o más dimensiones se denominan ***matrices***.

En definitiva, un array es un conjunto de posiciones consecutivas de memoria, en la que se guardan datos del mismo tipo, y que recibe un único nombre, el nombre del array:

- Cada posición guarda un dato individual.
- Tiene un número asociado (un índice) que indica el lugar que ocupa dentro del array, comenzando **SIEMPRE** por el cero.
- Para acceder directamente a un dato individual (para darle valor, modificarlo o consultarlo…) lo hacemos mediante el nombre del array seguido del índice.
- Se guardan espacios consecutivos de memoria.

El nombre del array localiza la dirección de comienzo en memoria de la estructura, mientras que el índice representa el desplazamiento respecto a esa dirección de comienzo para llegar al elemento deseado.

Al declarar un array, deberemos decir cuántas dimensiones tiene, y cuántos elementos en cada dimensión. Algunos lenguajes obligan a hacer esto durante la compilación del programa (en tiempo de compilación), y otros permiten hacerlo durante la ejecución del programa (en tiempo de ejecución). En lo que suelen coincidir es en no limitar el número de dimensiones posibles. A nosotros nos cuesta imaginar más de tres dimensiones, porque lo asociamos al espacio, pero conceptualmente, no hay por qué limitarlo.

Por ejemplo, podemos representar las distintas mesas o puestos de trabajo de una empresa, de forma que:

1. La primera dimensión represente el edificio concreto de la empresa.
2. Dentro de cada edificio hay varias plantas, representadas por la segunda dimensión.
3. Dentro de cada planta, hay varios pasillos, representados por la tercera dimensión.
4. Dentro de cada pasillo, hay distintos despachos, representados por la cuarta dimensión, y
5. dentro de cada despacho hay varias mesas, representadas por la quinta dimensión.

Cuando hablamos de matrices, vectores y arrays, tendemos a limitar el concepto de array al de tabla, pero en informática el concepto de array o tabla, es mucho más amplio.

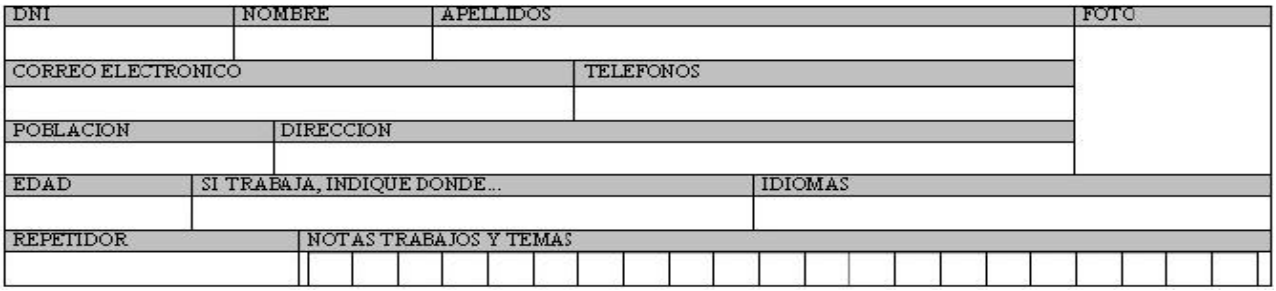
En Java, para definir un array, se pone el tipo de los elementos individuales, seguido de un corchete por cada dimensión que queramos darle, y del nombre del array. Posteriormente habrá que dimensionarlo, indicando cuántos elementos va a tener, y posteriormente se usará para almacenar valores con algún propósito. Operaciones típicas con arrays son *recorrerlo* procesando todos sus elementos (llenarlo, etc.), *consultar* el valor de un elemento, *modificar* un elemento (borrarlo, darle valor, actualizarlo).

En ese ejemplo se declara un array bidimensional de números enteros. A continuación, decimos que va a tener 3 filas y 4columnas, es decir, decimos cuántos elementos va a tener cada dimensión. Y finalmente le damos valor a algunos elementos individuales del array.

Ejemplo en Java:

```
…
int [ ][ ] arrayBidimensionalDeEnteros;
arrayBidimensionalDeEnteros = new int[3][4];
arrayBidimensionalDeEnteros[0][0]=23;
…
arrayBidimensionalDeEnteros[2][3]=376;
```

d.4) Registros



¿Para qué casos será apropiado usar registros? ¿Son los registros algo parecido a las fichas de cartulina donde recogen los datos de cada trabajador actualmente en la empresa? Efectivamente, ésa es la idea: llevar a formato digital esa ficha en cartulina, de forma que corregir los datos de una ficha, mantenerlas ordenadas, crear una nueva ficha, o consultar los datos de una de las fichas correspondiente a un trabajador concreto sea más fácil.

Un registro es una estructura de datos formada por yuxtaposición de elementos de distinto tipo que contiene información relativa a un mismo ente u objeto. A cada uno de los elementos que componen el registro se les llama campos. Los campos aparecen en un orden determinado y se identifican por un nombre. Para definir el registro, hay que darle un nombre y un tipo a cada campo. El tipo de cada campo puede ser una estructura de datos a su vez.

Por ejemplo, para representar la información de un alumno de este módulo profesional, que requiere asignarle una nota para el trabajo final de cada una de las 11 unidades de que consta, se puede usar un registro alumno que contenga los siguientes campos: apellidos (cadena de caracteres), nombre (cadena de caracteres), edad (entero), sexo (carácter), repetidor (lógico), notasTrabajosTemas (vector de números reales, de 11 elementos).

Posteriormente podré crear variables de tipo alumnoP que contendrán los datos de un alumno concreto.

No existen operaciones fijas a realizar con un registro, pero las habituales son:

- consultar sus valores,
- modificarlos y actualizarlos,
- crearlos y
- borrarlos.

NOTA: En Java no existen los registros como tales, pero se puede crear este tipo de estructuras como objetos de una clase. Se estudiarán más adelante.

d.5) Archivos o ficheros.

¿Pero cada registro es como una variable? ¿Si tengo miles de empleados tendré que gestionar miles de variables registro? ¿Si apago el ordenador perderé los datos de todos los registros que había creado en memoria? Claro que no. Para eso existen justamente otras estructuras de datos llamadas archivos o ficheros.

Un archivo es un conjunto de información sobre un mismo tema, tratada como unidad de almacenamiento y organizada de forma estructurada para la búsqueda y recuperación de un dato individual. Es por tanto la estructura que necesitamos usar para poder guardar los datos e informaciones en soportes de almacenamiento masivo o permanente, tales como discos duros, Cd's, memorias Flash, etc. y poder recuperarlos cada vez que quiera usarlos en una aplicación sin tener que volver a introducirlos de nuevo. Si sólo se guardan en la memoria principal del ordenador, se perderán cuando éste se apague.

Habitualmente los ficheros o archivos están compuestos por una colección de registros homogéneos. Así, por ejemplo, lo lógico es que los datos de los alumnos de este módulo profesional se guardaran en memoria permanente, en el disco duro del ordenador de secretaría. Para ello lo normal es que se cree un fichero de alumnos que sería una colección de registros alumnoP, como los definidos en el apartado anterior.

Las operaciones típicas a realizar con un fichero son:

- Creación del fichero.
- Inserción de un registro.
- Eliminación o borrado de un registro.
- Consulta de uno, varios o todos los registros.
- Modificación o actualización de un registro.
- Borrado del fichero.

- Búsqueda y localización de un registro concreto (normalmente habrá que hacerlo antes de cualquier operación de recuperación o actualización de un registro.)
- Duplicado o copia de seguridad del fichero.
- Ordenación del fichero.
- Mezcla de los datos de varios ficheros para formar uno nuevo.

NOTA: Más adelante veremos más cosas sobre ficheros, aunque no es parte del temario estudiarlo a fondo.

NOTA: Existen gran cantidad de tipos de datos, no básicos que todavía nos quedan por nombrar, como punteros (aunque java no tiene punteros, tiene referencias que son algo parecido conceptualmente y de manejo más sencillo), listas, árboles, enumerados, etc. Algunos de ellos, prácticamente no se usan aquí y otros los estudiaremos en su momento.

5. Operadores.

Los operadores son símbolos especiales que realizan operaciones concretas sobre uno, dos o tres *operandos* y devuelven un resultado.

A medida que exploremos los operadores del lenguaje Java será de utilidad conocer con antelación cuáles de ellos poseen la precedencia más alta. Los operadores de la siguiente tabla se muestran por orden de precedencia. Cuanto más alto esté el operador en la tabla, mayor es su precedencia. Los operadores con mayor precedencia se evalúan antes que los que posean una menor. Los operadores que estén en la misma línea tienen igual precedencia. Cuando en la misma expresión aparecen operadores de igual precedencia, debe haber una regla que indique cuál se evalúa primero. Todos los operadores binarios, excepto los de asignación, se evalúan de izquierda a derecha; los operadores de asignación se evalúan de derecha a izquierda.

Hay muchos tipos de operadores en Java. No veremos aquí algunos, como los operadores de bits.

Cada tipo se usa para una cosa, por ejemplo, los aritméticos se utilizan para realizar operaciones matemáticas como sumar, restar, etc.

Operadores	Precedencia
postfix	*expr*++ *expr*--
unarios	++*expr* --*expr*

multiplicativos	* / %
aditivos	+ -
de movimiento (shift)	<< >> >>>
relacionales	< > <= >= instanceof
de igualdad	== !=
AND a nivel de bit (bitwise AND)	&
OR exclusivo a nivel de bit (bitwise exclusive OR)	^
OR inclusivo a nivel de bit (bitwise inclusive OR)	|
AND lógico	&&
OR lógico	||
ternarios	? :
de asignación	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Operadores de asignación, aritméticos y unarios

El operador de asignación sencillo

Uno de los operadores más habituales que se encontrará es el operador de asignación sencillo "="; asigna el valor de su derecha al operando a su izquierda:

```
int cadence = 0;
int speed = 0;
int gear = 1;
```

Este operador también se puede utilizar en objetos para asignar *referencias a objetos* (se verá en la unidad correspondiente)

Los operadores aritméticos

El lenguaje de programación Java proporciona operadores que realizan la suma, resta, multiplicación y división. Probablemente los reconozcas por sus homólogos en la matemática básica. El único símbolo que te podrá parecer nuevo es «%», que divide un operando por el otro y devuelve el resto como su resultado (se denomina módulo).

+ operador de adición (también se utiliza para concatenar Strings)
- operador de sustracción
***** operador de multiplicación
/ operador de división
% operador de resto o también llamado módulo

Veamos un ejemplo:

```
public class ArithmeticDemo {

   public static void main (String[] args){

      int result = 1 + 2; // result es ahora 3
      System.out.println(result);

      result = result - 1; // result es ahora 2
      System.out.println(result);

      result = result * 2; // result es ahora 4
      System.out.println(result);

      result = result / 2; // result es ahora 2
      System.out.println(result);

      result = result + 8; // result es ahora 10
      result = result % 7; // result es ahora 3
      System.out.println(result);

   }
}
```

También puede combinar los operadores aritméticos con el operador de asignación sencillo para crear *asignaciones compuestas*. Por ejemplo, x+=1; y x=x+1; ambos incrementan el valor de x en 1.

Operador	Uso	Equivalente a
+=	*op1 += op2*	op1 = op1 + op2
-=	*op1 -= op2*	op1 = op1 - op2
*=	*op1 *= op2*	op1 = op1 * op2
/=	*op1 /= op2*	op1 = op1 / op2
%=	*op1 %= op2*	op1 = op1 % op2
&=	*op1 &= op2*	op1 = op1 & op2

El operador + también se puede utilizar para concatenar (unir) dos cadenas, como se muestra en el siguiente código:

```
public class ConcatDemo {
   public static void main(String[] args){
      String firstString = "Esto es";
      String secondString = " una cadena concatenada.";
      String thirdString = firstString+secondString;
      System.out.println(thirdString);
   }
}
```

Nota: Las cadenas son en realidad "objetos especiales". Estudiaremos la programación orientada a objetos en la unidad 3.

Los operadores unarios

Los operadores unarios solamente necesitan un operando; realizan diferentes operaciones como incrementar o disminuir un valor en una unidad, negar una expresión o invertir el valor de un booleano.

+ Operador unario «más», indica un valor positivo (sin embargo, los números son positivos sin el operador).
- Operador unario «menos»; niega una expresión.
++ Operador de incremento; incrementa un valor en 1.
-- Operador de decremento; decrementa un valor en 1.
! Operador de complemento lógico; invierte el valor de un booleano.

El siguiente programa muestra su uso:

```
public class UnaryDemo {

   public static void main(String[] args){
      int result = +1; // result es ahora 1
      System.out.println(result);
      result--;  // result es ahora 0
      System.out.println(result);
      result++; // result es ahora 1
      System.out.println(result);
      result = -result; // result es ahora -1
      System.out.println(result);
      boolean success = false;
      System.out.println(success); // falso/false
      System.out.println(!success); // verdadero/true
   }
}
```

Los operadores de incremento y decremento se pueden aplicar delante (prefix) o detrás (postfix) del operando. Tanto el código result++; como ++result; resultarán en que result será incrementado en uno. La única diferencia es que la versión prefix (++result) evalúa según el valor incrementado, mientras que la versión postfix (result++) evalúa según el valor original. Si solamente está realizando un incremento o decremento sencillo, realmente no importa cuál de las dos formas utiliza. Pero si utiliza este operador como parte de una expresión más larga, la forma que elija puede influir significativamente en el resultado. Veamos un ejemplo:

```
public class PrePostDemo {
   public static void main(String[] args){
      int i = 3;
      i++;
      System.out.println(i);      // "4"
      ++i;
      System.out.println(i);      // "5"
      System.out.println(++i);    // "6"
      System.out.println(i++);    // "6"
      System.out.println(i);      // "7"
   }
}
```

Probad vosotros qué ocurre cuando se usan el incremento y decremento mezclados con asignaciones, por ejemplo, j=i++; o j=++i;

Veamos un ejemplo de lo que se pide en el ejercicio anterior:

Examinemos ahora, la posición del operador respecto del operando. Consideremos en primer lugar que el operador unario ++ está a la derecha del operando. La sentencia

```
j=i++; //primero asigna el valor de i a j y luego incremento i en uno
```

asigna a *j*, el valor que tenía *i*. Por ejemplo, si *i* valía 3, después de ejecutar la sentencia, *j* toma el valor de 3 e *i* el valor de 4. Lo que es equivalente a las dos sentencias:

```
j=i;
i++;
```

Un resultado distinto se obtiene si el operador ++ está a la izquierda del operando

```
j=++i; //Primero incrementa i en uno y ese nuevo valor lo asigna a j.
```

asigna a *j* el valor incrementado de *i*. Por ejemplo, si *i* valía 3, después de ejecutar la sentencia *j* e *i* toman el valor de 4. Lo que es equivalente a las dos sentencias:

```
++i;
  j=i;
```

Operadores de igualdad, relacionales y condicionales

Los operadores de igualdad y relacionales

Los operadores de igualdad y relacionales determinan si un operando es mayor, menor, igual a o distinto de otro. Probablemente la mayoría de estos operadores también le serán familiares. Tenga en mente que debe utilizar «==», no «=» cuando compruebe si dos valores primitivos son iguales.

```
==   igual a
!=   distinto de
>    mayor que
>=   mayor que o igual a
<    menor que
<=   menor que o igual a
```

El siguiente programa, comprueba los operadores de comparación (ya explicaremos qué significa la palabra **if**, de momento comprueba si es cierta o no la condición del paréntesis):

```
public class ComparisonDemo {

   public static void main(String[] args){
      int value1 = 1;
      int value2 = 2;
      if(value1 == value2) System.out.println("value1 == value2");
      if(value1 != value2) System.out.println("value1 != value2");
      if(value1 > value2) System.out.println("value1 > value2");
      if(value1 < value2) System.out.println("value1 < value2");
      if(value1 <= value2) System.out.println("value1 <= value2");
   }
}
```

Resultado:
value1 != value2
value1 < value2
value1 <= value2

Los operadores condicionales

Los operadores && y || realizan las operaciones *AND-Condicional* y *OR-Condicional* sobre dos expresiones booleanas. Estos operadores muestran un comportamiento de «cortocircuito», lo que significa que el segundo operando solamente se evalúa si es necesario.

&& AND-Condicional
|| OR-Condicional

Ejemplo:

```
class ConditionalDemo1 {

   public static void main(String[] args){
         int value1 = 1;
         int value2 = 2;

                  System.out.println("¿Es value1 igual a 1? "+(value1==1));
                  System.out.println("¿Es value1 igual a 2? "+(value1==2));
   }
}
```

Os dejo por último una tabla con el orden de precedencia de todos los operadores (para consultarla cuando lo necesitéis). Aunque no es necesario saber de memoria el orden de todos los operadores y, con el tiempo, lo aprenderéis, si es importante conocer el orden de importancia a la hora de realizar una operación u otra pues es muy común, encontrar varias operaciones en una misma línea de código.

Priority	Operator	Operation	Order of Evaluation
1	[ ]	Array index	Left to Right
	()	Method call	

	.	Member access	
2	++	Prefix or postfix increment	Right to Left
	--	Prefix or postfix decrement	
	+ -	Unary plus, minus	
	~	Bitwise NOT	
	!	Boolean (logical) NOT	
	(type)	Typecast	
	new	Object creation	
3	* / %	Multiplication, division, remainder	Left to Right
4	+ -	Addition, subtraction	Left to Right
	+	String concatenation	
5	<<	Signed bit shift left to right	Left to Right
	>>	Signed bit shift right to left	
	>>>	Unsigned bit shift right to left	
6	<<=	Less than, less than or equal to	Left to Right
	>>=	Greater than, greater than or equal to	
	instanceof	Reference test	
7	==	Equal to	Left to Right
	!=	Not equalt o	
8	&	Bitwise AND	Left to Right
	&	Boolean (logical) AND	
9	^	BitwiseXOR	Left to Right
	^	Boolean (logical) XOR	
10	|	Bitwise OR	Left to Right
	|	Boolean (logical) OR	
11	&&	Boolean (logical) AND	Left to Right
12	||	Boolean (logical) OR	Left to Right
13	? :	Conditional	Right to Left
14	=	Assignment	Right to Left
	*= /= += -= %= <<= >>= >>>= &= ^= |=	Combinated assignment (operation and assignment)	

6. Constantes y comentarios

a) Constantes

Las constantes en java, por cuestión de estilo, se suelen declarar en MAYÚSCULA, mientras que las variables se declaran en minúscula. Las constantes se declaran de la siguiente manera:

final <tipo de dato> <nombre de la constante> = valor;

Ejemplo: ***final double PI = 3.141592;***

El calificador ***final*** identificará que es una constante.

IMPORTANTE: Las constantes se utilizan en datos que nunca varían (PI, e, etc.) Usando constantes y no variables, aseguramos que su valor no va a poder ser modificado nunca. También, permite centralizar el valor de un dato en una sola línea de código, es decir, si se quiere cambiar el valor del IVA se hará solamente en una sola línea sin tener que hacerlo una por una, en todas las partes del programa donde se ha utilizado. Hoy día pocas variables serán constantes, pues prácticamente todo cambia.
En java, no existen las variables globales, como en otros lenguajes.

b) Comentarios

Un comentario en java es aquella parte de código que el compilador no tendrá en cuenta y que sirven para dar información adicional sobre el código.
Existen 3 tipos de comentarios en Java:

1. Doble barra (//): Para comentarios de una sola línea. Desde donde se encuentran las barras hasta el final de esa línea.
2. Barra y asterisco (/* */): Para comentarios de varias líneas. Este empezará desde la barra y el primer asterisco hasta el cierre del comentario con */.
3. Barra y doble asterisco (/ */):** Son comentarios usados para generar la documentación del programa. Se verá más adelante, pero diremos que Java proporciona una herramienta para la documentación del programa en formato **html**, similar al API.

Veamos algunos ejemplos de comentarios reales (no hacer bajo ningún concepto).

https://www.genbetadev.com/desarrolladores/comentarios-de-codigo-algunos-de-los-mas-divertidos

7. Ejemplos

a) Las variables deben ser declaradas antes o en el momento de ser usadas (normalmente se declaran al principio del programa, son pues, las primeras líneas dentro del main). En la declaración **se determina el *tipo* de variable de que se trata y sólo puede realizarse una vez.**

b) **Ejemplo 1:**

```
//Ejemplo de asignación

public class T2EjemploAsignacion{
        public static void main ( String arg[ ] ){
                short enteroCorto=0;
                int entero=0;
                long enteroLargo=0;
                enteroCorto = 23000; //Ya no es necesario poner short otra vez
                entero = 2300000;
                enteroLargo = 109999915;
                System.out.println ("Entero corto= "+ enteroCorto);
                System.out.println ("Entero normal = "+ entero);
                System.out.println ("Entero largo= "+ enteroLargo);
        }
}
```

Un valor numérico sin decimales, definido por sus dígitos, es decir, en formato literal, java lo toma como un entero del tipo más corto que lo pueda contener, partiendo como mínimo de un tipo short. Así, de esta forma, el número 23000 del ejemplo T2EjemploAsignacion, se toma de tipo *short*. El número 87 se tomaría igualmente de tipo short aun cuando cabría en un entero de tipo *byte.* Si queremos que se tome de tipo *byte* debemos poner un *casting o molde* (byte) para convertirlo en un entero de tipo *byte.* Los castings se verán con más detenimiento en temas posteriores.

De momento, nosotros asignaremos un valor a las variables justo en el momento de declararlas, para evitar que contengan "basura", es decir, algún valor que pueda perjudicar futuros cálculos con ellas, por ejemplo, cero:

c) **Ejemplo 2:**

```
//Ejemplo T2AsignaReales

public class T2AsignaReales{
        public static void main (String args[ ] ){
                float interes=0.0;
                double capital=0.0;
                interes=6.25F; //Para especificar que es float pero no es obligatorio
                capital = 3.72E+9D; //La D para double, tampoco se pone
                System.out.println ("Interés =  " + interes);
                System.out.println("Capital =  " + capital);
        }
}
```

En los datos con decimales, escritos de forma decimal, la coma decimal debe ser un punto. Por defecto, toma el espacio de un dato decimal de tipo double. Si queremos que se tome como un dato de tipo float hay que ponerle a la derecha la letra "F" (aunque esto último no es necesario en algunos IDEs).

Si se pone a la derecha la letra "D" se toma como un dato de tipo double, pero esto no es necesario, ya que el tipo double es por defecto. Para especificar que es Float se pone al final la letra F. La letra "E" en el centro seguida de un carácter "+" o "-", indica que se trata de un dato en potencias de 10, es decir, el valor de la izquierda de la letra "E" se multiplica por 10 elevado a la cantidad que hay a la derecha de la letra "E". O bien, se divide, si se trata del carácter "-".

d) Ejemplo 3: T2AsignaOtras

Ejemplo T2AsignaOtras

```
public class T2AsignaOtras {
	public static void main( String arg[ ] ) {
		boolean esCierto;
		char letra;
		esCierto = false; //Dos posibles valores, true o false
		letra = 'a'; //Entre comillas simples, la de la interrogación
		System.out.println ("Es cierto = "+ esCierto);
		System.out.println ("Letra = "+ letra);
		letra = '\r';
		// La /n significa salto de línea
		System.out.println ("Este texto\n se ve en\n tres líneas");
	}
}
```

Los datos de tipo carácter se escriben entre comillas simples, así el carácter “a” se tiene que escribir como ‘a’. Si se escribe como “a”, es tomado como una cadena de caracteres, aunque contenga sólo una letra.

Como ya hemos visto, en las secuencias de escape o caracteres especiales, también conocidos como caracteres de control, o no imprimibles, se identifican con el carácter ‘\’ y una letra que identifica el carácter, por ejemplo, el salto de línea se identifica como ‘\n’.

OJO: VISIBILIDAD Y VIDA DE LAS VARIABLES: LA VISIBILIDAD DE UNA VARIABLE (O SCOPE) ES LA PARTE DEL CÓDIGO DE UNA APLICACIÓN DONDE LA VARIABLE ES ACCESIBLE Y PUEDE SER UTILIZADA. EN JAVA, LAS VARIABLES NO PUEDEN DECLARARSE FUERA DE UNA CLASE Y POR LO GENERAL, TODAS LAS VARIABLES QUE ESTÁN DENTRO DE UN BLOQUE, ES DECIR, ENTRE DOS LLAVES {}, SON VISIBLES Y EXISTEN DENTRO DE DICHO BLOQUE. AL ACABAR EL TROZO DE CÓDIGO CON LA LLAVE DE CIERRE, MUEREN COMO BELLACAS.

8. Palabras reservadas o Tokens

Las palabras clave son las órdenes del lenguaje de programación. El compilador espera esos identificadores para "comprender" el programa, compilarlo y ejecutarlo. Queda por tanto **PROHIBIDO** utilizar palabras reservadas como ***boolean, double, float, if, private, class, main***, etc. para otras cosas como nombres de variables. Tampoco se pueden usar caracteres especiales para nombrar variables como ***+, -, /***, etc.

Tabla de Tokens