La Sénia
Institut d'Educació Secundària
Paiporta

GENERALITAT VALENCIANA
CONSELLERIA D'EDUCACIÓ, INVESTIGACIÓ, CULTURA I ESPORT

Fondo Social Europeo

Programación

UD 2: Introducción a JAVA

Ciclo Formativo de Grado Superior
Desarrollo de Aplicaciones Web

Jose Chamorro Molina

Introducción a Java

ORDEN 60/2012, de 25 de septiembre, de la Conselleria de Educación, Formación y Empleo por la que se establece para la Comunitat Valenciana el currículo del ciclo formativo de Grado Superior correspondiente al título de Técnico Superior en Desarrollo de Aplicaciones Web. [2012/9149]

Contenidos:

1.- Identificación de los elementos de un programa informático:
- 1.4.− Variables.
- 1.5.− Tipos de datos. Operaciones.
- 1.6.− Literales.
- 1.7.− Constantes.
- 1.8.− Operadores y expresiones.
- 1.9.− Conversiones de tipo.
- 1.10.− Comentarios.

Real Decreto 686/2010, de 20 de mayo, por el que se establece el título de Técnico Superior en Desarrollo de Aplicaciones Web y se fijan sus enseñanzas mínimas.

Resultados de aprendizaje:

1. Reconoce la estructura de un programa informático, identificando y relacionando los elementos propios del lenguaje de programación utilizado.

Criterios de evaluación:

1.d) Se han identificado los distintos tipos de variables y la utilidad específica de cada uno.
1.e) Se ha modificado el código de un programa para crear y utilizar variables.
1.f) Se han creado y utilizado constantes y literales.
1.g) Se han clasificado, reconocido y utilizado en expresiones los operadores del lenguaje.
1.h) Se ha comprobado el funcionamiento de las conversiones de tipo explícitas e implícitas.
1.i) Se han introducido comentarios en el código.

Competencias profesionales, personales y sociales:

q) Resolver situaciones, problemas o contingencias con iniciativa y autonomía en el ámbito de su competencia, con creatividad, innovación y espíritu de mejora en el trabajo personal y en el de los miembros del equipo.

Introducción a Java

1.- Variables e Identificadores

2.- Palabras reservadas

3.- Tipos de datos primitivos

4.- Declaración e inicialización

5.- Literales

6.- Constantes

7.- Operadores y expresiones

8.- Conversiones de tipo

9.- Comentarios

El lenguaje de programación Java

¿Qué es Java?

El lenguaje de programación Java es un lenguaje sencillo de aprender. Su sintaxis es la de C++ “simplificada”. Los creadores de Java partieron de la sintaxis de C++ y trataron de eliminar de este todo lo que resultase complicado o fuente de errores en este lenguaje.

Java es un lenguaje orientado a objetos, aunque no de los denominados puros; en Java todos los tipos, a excepción de los tipos fundamentales de variables (int, char, long...) son clases. Sin embargo, en los lenguajes orientados a objetos puros incluso estos tipos fundamentales son clases, por ejemplo en Smalltalk.

El lenguaje de programación Java

Historia

El 23 de Mayo de 1995, Java vio la luz de forma pública, durante la conferencia SunWorld.

La compañía Sun Microsystems presentó el lenguaje en el que había estado trabajando durante más de cinco años de forma interna el equipo de James Gosling (el padre de la criatura).

Un auténtico lenguaje moderno concebido para funcionar en cualquier dispositivo, esa fue la idea.

El lenguaje de programación Java

Características

Está diseñado para facilitar el trabajo en la WWW, mediante el uso de los programas navegadores de uso completamente difundido hoy en día. Los programas de Java que se ejecutan a través de la red se denominan applets (aplicación pequeña).

Inclusión en el lenguaje de un entorno para la programación gráfica (AWT y Swing)

Su ejecución es independiente de la plataforma, lo que significa que un mismo programa se ejecutará exactamente igual en diferentes sistemas.

Write Once, Run Anywhere
"Escríbelo una vez, ejecútalo en cualquier lugar"

El lenguaje de programación Java

Java Runtime Environment o JRE es un conjunto de utilidades que permite la ejecución de programas Java.

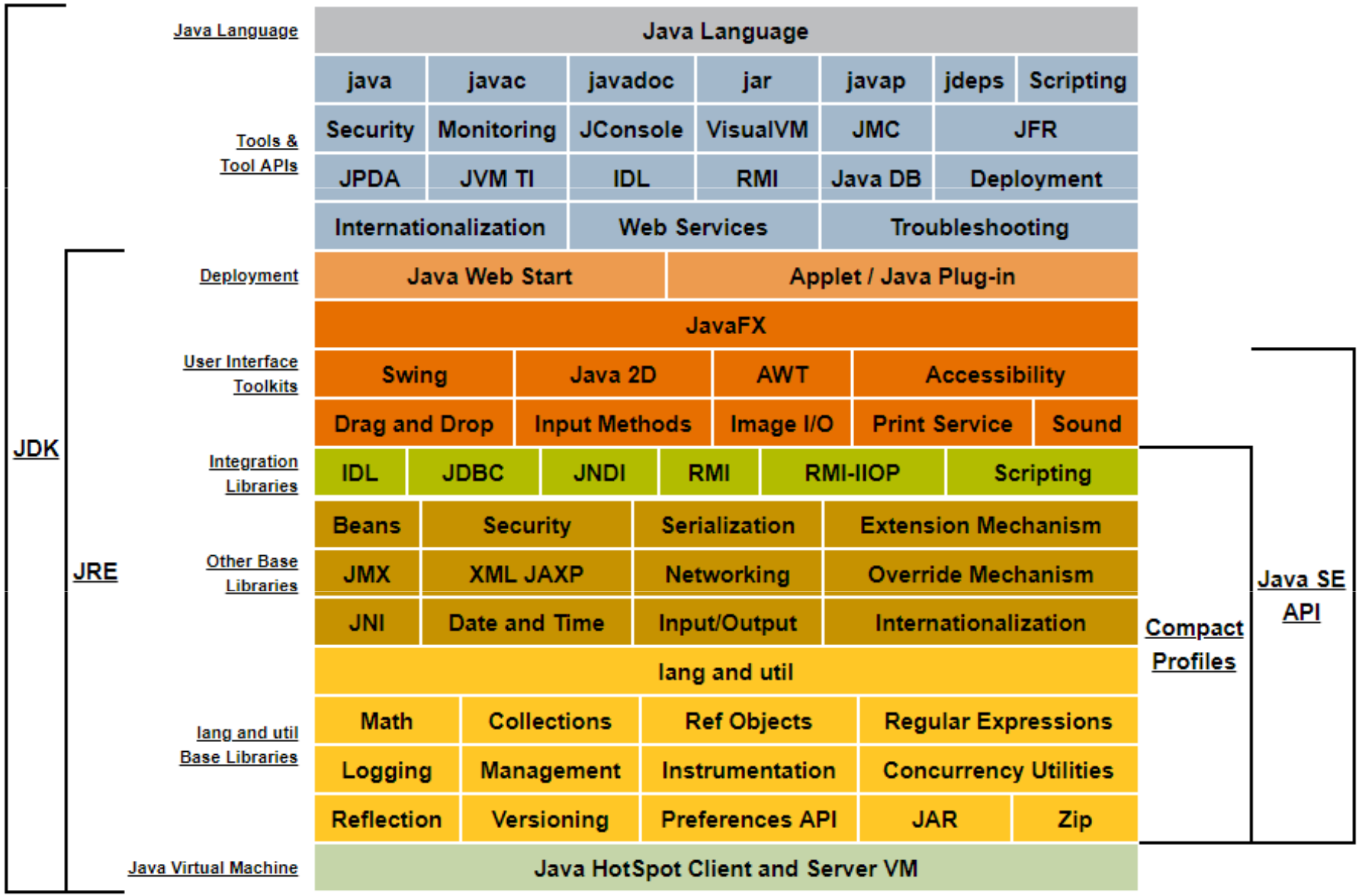
En su forma más simple, el entorno en tiempo de ejecución de Java está conformado por una Máquina Virtual de Java o JVM, un conjunto de bibliotecas Java y otros componentes necesarios para que una aplicación escrita en lenguaje Java pueda ser ejecutada. El JRE actúa como un "intermediario" entre el sistema operativo y Java.

La JVM es el programa que ejecuta el código Java previamente compilado (bytecode) mientras que las librerías de clases estándar son las que implementan el API de Java. Ambas JVM y API deben ser consistentes entre sí, de ahí que sean distribuidas de modo conjunto.

Un usuario sólo necesita el JRE para ejecutar las aplicaciones desarrolladas en lenguaje Java, mientras que para desarrollar nuevas aplicaciones en dicho lenguaje es necesario un entorno de desarrollo, denominado Java Development Kit o JDK, que además del JRE (mínimo imprescindible) incluye, entre otros, un compilador para Java.

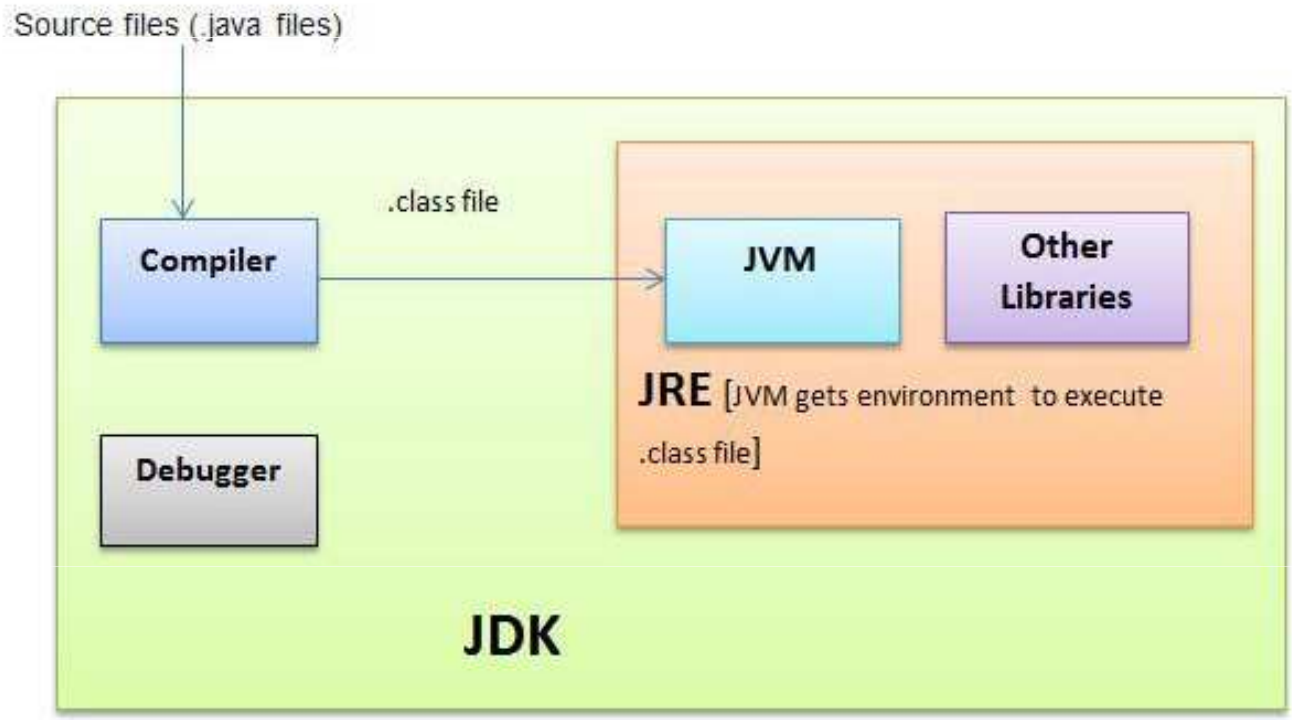
El lenguaje de programación Java

Description of Java Conceptual Diagram



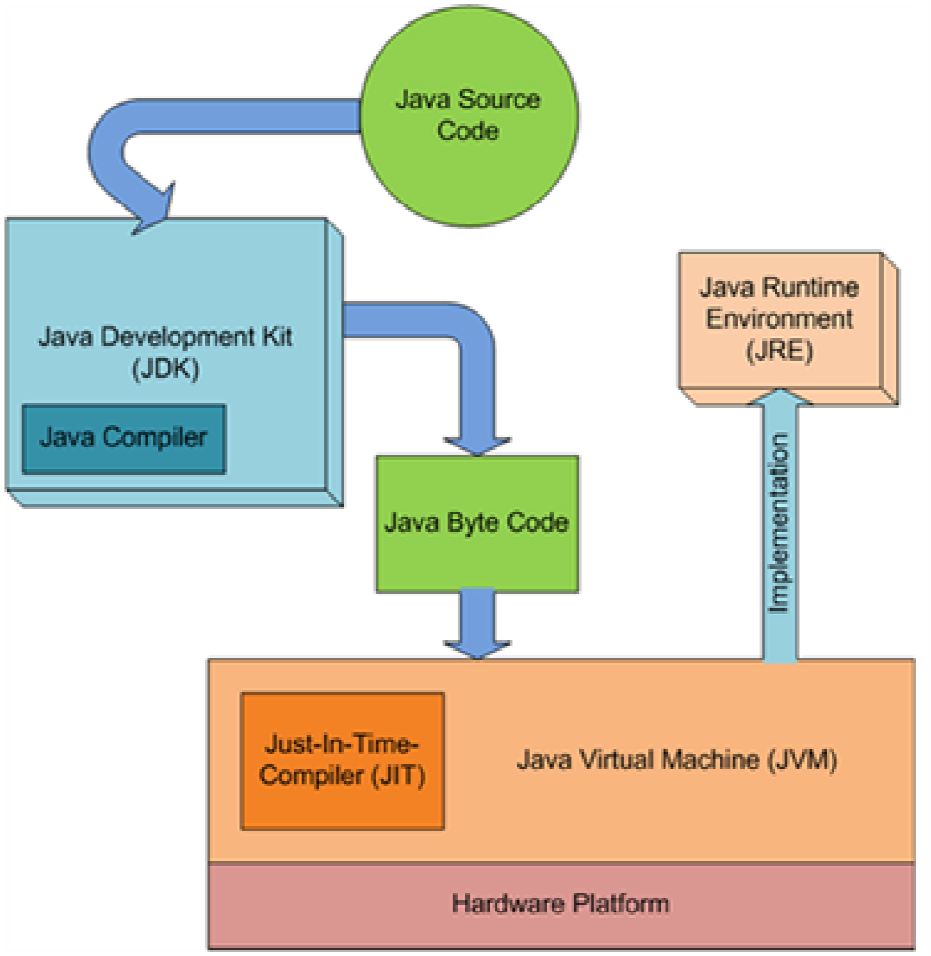
El lenguaje de programación Java

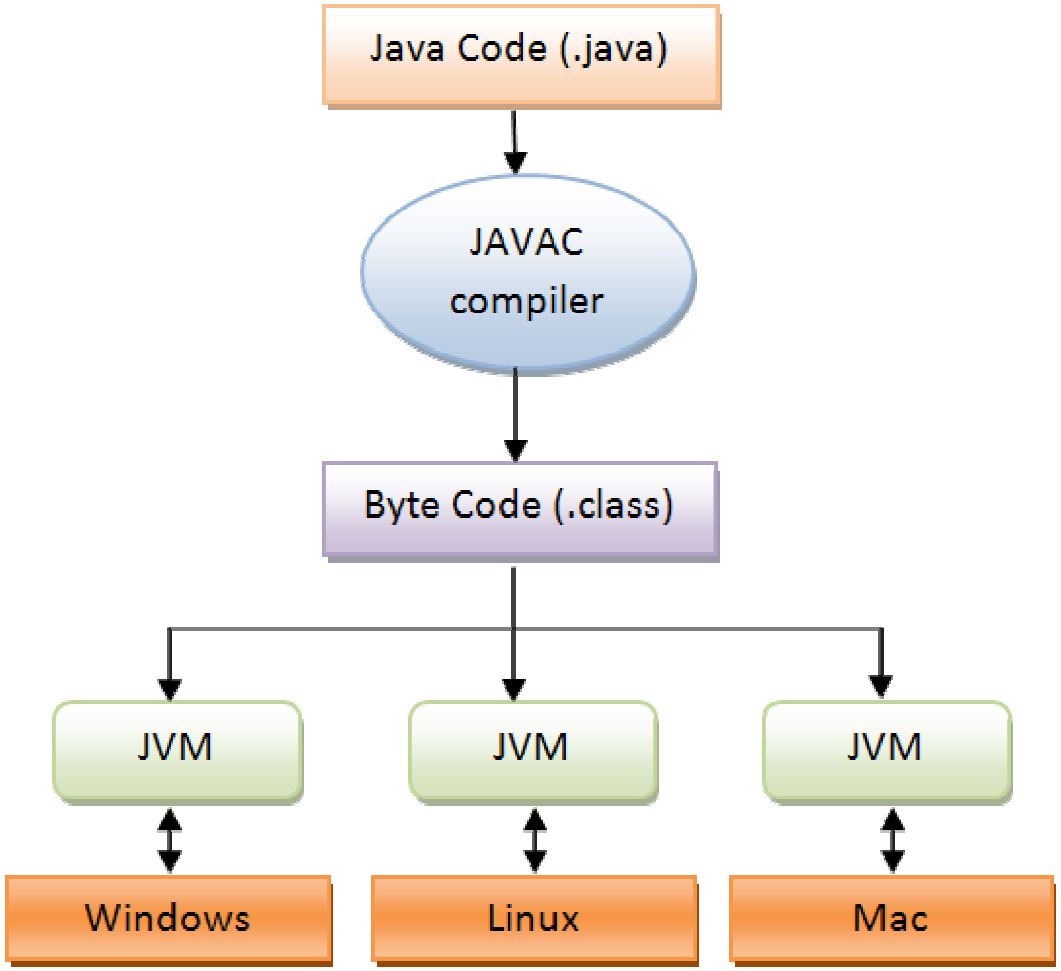
Simplificando…



El lenguaje de programación Java

De codificar a ejecutar…





El lenguaje de programación Java

¿Por qué Java?

- ✓ Es un lenguaje sencillo de aprender.
- ✓ Es un lenguaje Orientado a Objetos.
- ✓ Gran comunidad de desarrolladores.
- ✓ Su ejecución es independiente de la plataforma.
- ✓ Se pueden desarrollar todos los contenidos del currículo de 1º DAW.
- ✓ Incorporación al mundo laboral actual.

El lenguaje de programación Java

¿Por qué Java en 1ºDAW?

Hay que escoger un Lenguaje Orientado a Objetos…

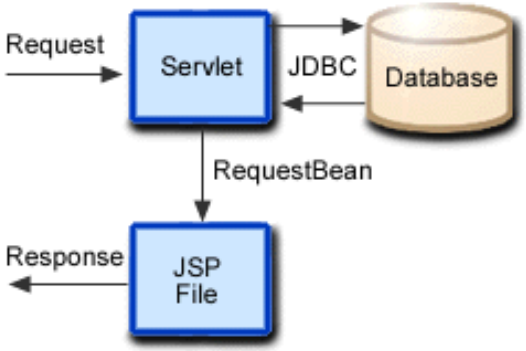
Concurso nacional ciclos formativos ProgramaMe: Java o C++

Entonces… ¿Java o C++?

El módulo Programación es común para 1º DAW y 1º DAM

2º DAW

Servlets en Java (MVC):



2º DAM

Programar para Android:

El lenguaje de programación Java

TIOBE Index for June 2018

TIOBE Programming Community Index

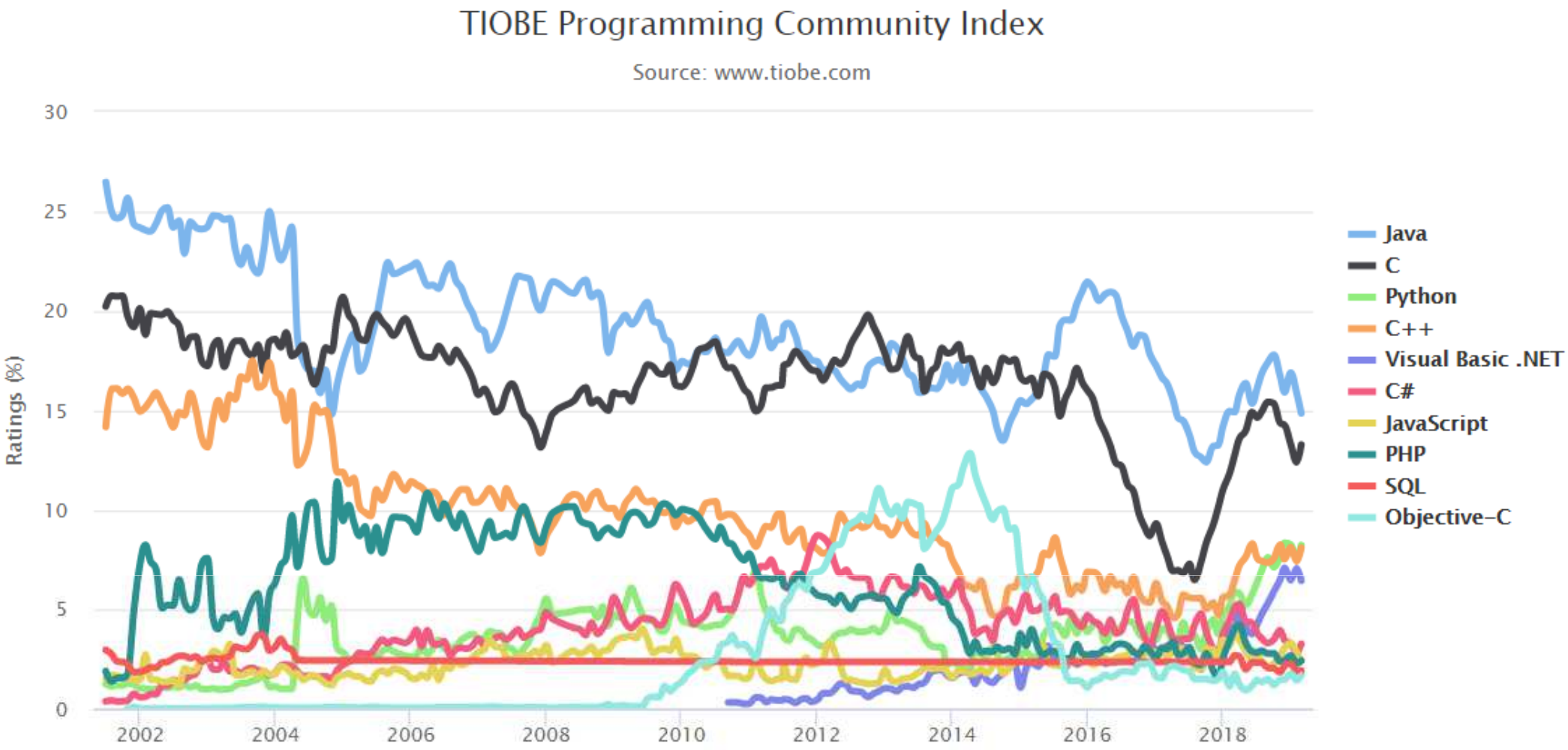
Source: www.tiobe.com

June Headline: TypeScript finally enters the TIOBE Index Top 100

Jun 2018	Jun 2017	Change	Programming Language	Ratings	Change
1	1		Java	15.368%	+0.88%
2	2		C	14.936%	+8.09%
3	3		C++	8.337%	+2.61%
4	4		Python	5.761%	+1.43%
5	5		C#	4.314%	+0.78%
6	6		Visual Basic .NET	3.762%	+0.65%
7	8	↑	PHP	2.881%	+0.11%
8	7	↓	JavaScript	2.495%	-0.53%
9	-	⇈	SQL	2.339%	+2.34%
10	14	⇈	R	1.452%	-0.70%
11	11		Ruby	1.253%	-0.97%
12	18	⇈	Objective-C	1.181%	-0.78%
13	16	↑	Visual Basic	1.154%	-0.86%

https://www.tiobe.com/tiobe-index/

El lenguaje de programación Java



1.- Variables e Identificadores

1.- Variables e identificadores

Una variable es una zona en la memoria del ordenador con un valor que puede ser almacenado para ser usado más tarde en el programa.

Las variables vienen determinadas por:

- un nombre, que permite al programa acceder al valor que contiene en memoria. Debe ser un identificador válido.

- un tipo de dato, que especifica qué clase de información guarda la variable en esa zona de memoria

- un rango de valores que puede admitir dicha variable.

1.- Variables e identificadores

Sirven para referirse tanto a objetos como a tipos primitivos.

Tienen que declararse antes de usarse:

```
tipo identificador;
```

```
int posicion;
```

Se puede inicializar mediante una asignación:

```
tipo identificador = valor;
```

```
int posicion = 0;
```

Definición de constantes:

```
static final float PI = 3.14159f;
```

1.- Variables e identificadores

Se llama identificador al nombre que le damos a la variable.

Los identificadores:

- Nombran variables, funciones, clases y objetos.

- Comienza con una letra. Los siguientes caracteres pueden ser letras o dígitos.

- Se distinguen las mayúsculas de las minúsculas.

- No hay una longitud máxima establecida para el identificador.

1.- Variables e identificadores

Tipos de variables:

- ✓ Variables de tipos primitivos y variables referencia.
- ✓ Variables y constantes.
- ✓ Variables miembro y variables locales.

1.- Variables e identificadores

Variables de tipos primitivos y variables referencia, según el tipo de información que contengan. En función de a qué grupo pertenezca la variable, tipos primitivos o tipos referenciados, podrá tomar unos valores u otros, y se podrán definir sobre ella unas operaciones u otras.

Variables y constantes, dependiendo de si su valor cambia o no durante la ejecución del programa. La definición de cada tipo sería:

- ✓ Variables. Sirven para almacenar los datos durante la ejecución del programa, pueden estar formadas por cualquier tipo de dato primitivo o referencia. Su valor puede cambiar varias veces a lo largo de todo el programa.
- ✓ Constantes o variables finales. Son aquellas variables cuyo valor no cambia a lo largo de todo el programa.

1.- Variables e identificadores

Variables miembro y variables locales, en función del lugar donde aparezcan en el programa. La definición concreta sería:

Variables miembro. Son las variables que se crean dentro de una clase, fuera de cualquier método. Pueden ser de tipos primitivos o referencias, variables o constantes. En un lenguaje puramente orientado a objetos como es Java, todo se basa en la utilización de objetos, los cuales se crean usando clases.

Variables locales. Son las variables que se crean y usan dentro de un método o, en general, dentro de cualquier bloque de código. La variable deja de existir cuando la ejecución del bloque de código o el método finaliza. Al igual que las variables miembro, las variables locales también pueden ser de tipos primitivos o referencias.

2.- Palabras reservadas

2.- Palabras reservadas

En el lenguaje de programación Java se puede hacer uso de las palabras clave (*keywords*), también llamadas palabras reservadas, mostradas en la siguiente tabla. Dichas palabras, no pueden ser utilizadas como identificadores por los programadores para definir variables, constantes, etc.

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

true, false y null no son considerados palabras clave de Java, sino literales. Ahora bien, tampoco se pueden utilizar como indentificadores.

La descripción de la funcionalidad de todas las palabras reservadas se puede encontrar en:
https://www.abrirllave.com/java/palabras-clave.php

3.- Tipos de datos primitivos

3.- Tipos de datos primitivos

Tipo	Descripción	Bytes	Rango	Valor por defecto
byte	Entero muy corto	1	-128 a 127	0
short	Entero corto	2	-32.768 a 32.767	0
int	Entero	4	-2.147.486.648 a 2.147.486.647 (-2^{31} a $2^{31}-1$)	0
long	Entero largo	8	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	0L
float	Número con punto flotante de precisión individual con hasta 7 dígitos significativos	4	+/-1.4E-45 (+/-1.4 times 10^{-45}) a +/-3.4E38 (+/-3.4 times 10^{38})	0.0f
double	Número con punto flotante de precisión doble con hasta 16 dígitos significativos	8	+/-4.9E-324 (+/-4.9 times 10^{-324}) a +/-1.7E308 (+/-1.7 times 10^{308})	0.0d
char	Carácter Unicode	\u0000 a \uFFFF	'\u0000'	
boolean	Valor verdadero o false	1	true o false	false

3.- Tipos de datos primitivos

Tipos referenciados

A partir de los ocho tipos datos primitivos, se pueden construir otros tipos de datos. Estos tipos de datos se llaman **tipos referenciados** o **referencias**, porque se utilizan para almacenar la dirección de los datos en la memoria del ordenador.

```
int[] arrayDeEnteros;
Cuenta cuentaCliente;
```

En la primera instrucción declaramos una lista de números del mismo tipo, en este caso, enteros. En la segunda instrucción estamos declarando la variable u objeto cuentaCliente como una referencia de tipo Cuenta.

Cuando el conjunto de datos utilizado tiene características similares se suelen agrupar en estructuras para facilitar el acceso a los mismos, son los llamados **datos estructurados**.

Son datos estructurados los **arrays, listas, árboles**, etc. Pueden estar en la memoria del programa en ejecución, guardados en el disco como ficheros, o almacenados en una base de datos.

3.- Tipos de datos primitivos

Tipos enumerados

Los **tipos de datos enumerados** son una forma de declarar una variable con un conjunto restringido de valores. Por ejemplo, los días de la semana, las estaciones del año, los meses, etc. Es como si definiéramos nuestro propio tipo de datos.

La forma de declararlos es con la palabra reservada **enum**, seguida del nombre de la variable y la lista de valores que puede tomar entre llaves. A los valores que se colocan dentro de las llaves se les considera como constantes, van separados por comas y deben ser valores únicos.

La lista de valores se coloca entre llaves, porque un tipo de datos **enum** no es otra cosa que una especie de clase en Java, y todas las clases llevan su contenido entre llaves.

4.- Declaración e inicialización

4.- Declaración e inicialización

Las declaraciones de variables pueden ir en cualquier parte del programa pero siempre antes de que la variable sea usada. Hay que tener cuidado con el rango de validez (scope) de la declaración.

Ejemplos:

```
int i;
int j = 1;
double pi = 3.14159;
char c = 'a';
boolean estamosBien = true;
```

5.- Literales

5.- Literales

Un **literal**, **valor literal** o **constante literal** es un valor concreto para los tipos de datos primitivos del lenguaje, el tipo **String** o el tipo **null**.

Los distintos tipos de literales son:

- ✓ Literales booleanos
- ✓ Literales enteros
- ✓ Literales reales
- ✓ Literales carácter
- ✓ Literales cadenas de caracteres

5.- Literales

Literales booleanos

Los **literales booleanos** tienen dos únicos valores que puede aceptar el tipo: **true** y **false**.

Por ejemplo, con la instrucción:

```
boolean encontrado = true;
```

estamos declarando una variable de tipo booleana a la cual le asignamos el valor literal **true**.

Literales enteros

Los **literales enteros** se pueden representar en tres notaciones:

Decimal: por ejemplo 20. Es la forma más común.

Octal: por ejemplo 024. Un número en octal siempre empieza por cero, seguido de dígitos octales (del **0** al **7**).

Hexadecimal: por ejemplo 0x14. Un número en hexadecimal siempre empieza por **0x** seguido de dígitos hexadecimales (del **0** al **9**, de la '**a**' a la '**f**' o de la '**A**' a la '**F**').

Las constantes literales de tipo **long** se le debe añadir detrás una **l** ó **L**, por ejemplo 873L, si no se considera por defecto de tipo **int**. Se suele utilizar **L** para evitar la confusión de la ele minúscula con 1.

5.- Literales

Literales reales

Los **literales reales** o en coma flotante se expresan con coma decimal o en notación científica, o sea, seguidos de un exponente **e** ó **E**.

El valor puede finalizarse con una **f** o una **F** para indica el formato **float** o con una **d** o una **D** para indicar el formato **double** (por defecto es **double**).

Por ejemplo, podemos representar un mismo literal real de las siguientes formas:

13.2, **13.2D**, **1.32e1**, **0.132E2**.

Otras constantes literales reales son por ejemplo:

.54, **31.21E-5**, **2.f**, **6.022137e+23f**, **3.141e-9d**.

5.- Literales

Literal carácter

Un **literal carácter** puede escribirse como un carácter entre comillas simples como 'a', 'ñ', 'Z', 'p', etc. o por su código de la tabla Unicode, anteponiendo la secuencia de escape '****' si el valor lo ponemos en octal o '**\u**' si ponemos el valor en hexadecimal.

Por ejemplo, si sabemos que tanto en ASCII como en Unicode, la letra A (mayúscula) es el símbolo número 65, y que 65 en octal es 101 y 41 en hexadecimal, podemos representar esta letra como '**\101**' en octal y '**\u0041**' en hexadecimal.

Existen unos caracteres especiales que se representan utilizando secuencias de escape:

Secuencia de escape	Significado	Secuencia de escape	Significado
\b	Retroceso	\r	Retorno de carro
\t	Tabulador	**\”**	Carácter comillas dobles
\n	Salto de línea	**\’**	Carácter comillas simples
\f	Salto de página	****	Barra diagonal

5.- Literales

Literales de cadenas de caracteres

Los **literales de cadenas de caracteres** se indican entre comillas dobles.

En el ejemplo anterior “El primer programa” es un literal de tipo cadena de caracteres. Al construir una cadena de caracteres se puede incluir cualquier carácter Unicode excepto un carácter de retorno de carro, por ejemplo en la siguiente instrucción utilizamos la secuencia de escape **\”** para escribir dobles comillas dentro del mensaje:

```
String texto = “Pedro dijo: \"Hoy hace un día fantástico…\"";
```

En el ejemplo anterior de tipos enumerados ya estábamos utilizando secuencias de escape, para introducir un salto de línea en una cadena de caracteres, utilizando el carácter especial **\n**.

*Normalmente, los objetos en Java deben ser creados con la orden new. Sin embargo, los literales **String** no lo necesitan ya que son objetos que se crean implícitamente por Java.*

6.- Constantes

6.- Constantes

Constantes o variables finales

Son aquellas variables cuyo valor no cambia a lo largo de todo el programa.

Declaración de constantes en Java

```
final double PI = 3.1415926536;
```

En nombre de las constantes se deben ser en mayúsculas.

7.- Operadores y expresiones

7.- Operadores y expresiones

Operadores Aritméticos:

Suma +

Resta -

Multiplicación *

División /

Resto de la División %

7.- Operadores y expresiones

Operadores de Asignación:

El principal es '=' pero hay más operadores de asignación con distintas funciones.

'+=': 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

7.- Operadores y expresiones

Operadores Relacionales:

Permiten comparar variables según relación de igualdad/desigualdad o relación mayor/menor. Devuelven siempre un valor boolean.

```
'>':  Mayor que
'<':  Menor que
'==': Iguales
'!=': Distintos
'>=': Mayor o igual que
'<=': Menor o igual que
```

7.- Operadores y expresiones

Operadores Lógicos:

Nos permiten construir expresiones lógicas.

'&&' : devuelve true si ambos operandos son true.

'||' : devuelve true si alguno de los operandos son true.

'!' : Niega el operando que se le pasa.

A	B	A && B	A || B	! A
true	**true**	true	true	false
true	**false**	false	true	false
false	**true**	false	true	true
false	**false**	false	false	true

7.- Operadores y expresiones

Operador de Concatenación:

Operador de concatenación con cadena de caracteres '+':

Ejemplo:

```
System.out.println("El total es " + result + " unidades.");
```

Operadores Incrementales:

Son los operadores que nos permiten incrementar las variables en una unidad. Prefija ó sufija.

'++'
'--'

7.- Operadores y expresiones

Operadores de Desplazamiento de bits:

Operador	Ejemplo en Java	Significado
~	~op	Realiza el complemento binario de op (invierte el valor de cada bit)
&	op1 & op2	Realiza la operación AND binaria sobre op1 y op2
|	op1 | op2	Realiza la operación OR binaria sobre op1 y op2
^	op1 ^ op2	Realiza la operación OR-exclusivo (XOR) binaria sobre op1 y op2
<<	op1 << op2	Desplaza op2 veces hacia la iquierda los bits de op1
>>	op1 >> op2	Desplaza op2 veces hacia la derecha los bits de op1
>>>	op1 >>> op2	Desplaza op2 (en positivo) veces hacia la derecha los bits de op1

Para saber más:

Los operadores de bits raramente los vas a utilizar en tus aplicaciones de gestión. No obstante, si sientes curiosidad sobre su funcionamiento, puedes ver el siguiente enlace dedicado a este tipo de operadores:

http://www.zator.com/Cpp/E4_9_3.htm

7.- Operadores y expresiones

Operador Condicional:

condición ? exp1 : exp2

Se explica con detalle en la UD04 - Uso de estructuras de control

7.- Operadores y expresiones

Operadores en orden de precedencia

Operadores	Asociatividad	Tipo
()	izquierda a derecha	paréntesis
++ -- + - !	derecha a izquierda	unarios
* / %	izquierda a derecha	multiplicativos
+ -	izquierda a derecha	aditivos
< <= > >=	izquierda a derecha	relacionales
== !=	izquierda a derecha	de igualdad
&	izquierda a derecha	AND lógico booleano
^	izquierda a derecha	OR exclusivo lógico booleano
|	izquierda a derecha	OR inclusivo lógico booleano
&&	izquierda a derecha	AND lógico
||	izquierda a derecha	OR lógico
?:	derecha a izquierda	condicional expresion ? sentencia1 : sentencia2
= += -= *= /= %=	derecha a izquierda	asignación ej. x += y ⇔ x = x + y;

8.- Conversiones de tipo

8.- Conversiones de tipo

El casting es un procedimiento para transformar una variable primitiva de un tipo a otro.

También se utiliza para transformar un objeto de una clase a otra clase siempre y cuando haya una relación de herencia entre ambas.

*En este tema nos centraremos en el primer tipo de casting.

Dentro de este casting de variables primitivas se distinguen dos clases:

- Casting implícito
- Casting explícito

Las conversiones de tipo se realizan para hacer que el resultado de una expresión sea del tipo que nosotros deseamos.

8.- Conversiones de tipo

Casting implícito o automático

Cuando a una variable de un tipo se le asigna un valor de otro tipo numérico con menos bits para su representación, se realiza una conversión automática.

En ese caso, el valor se dice que es promocionado al tipo más grande (el de la variable), para poder hacer la asignación.

También se realizan conversiones automáticas en las operaciones aritméticas, cuando estamos utilizando valores de distinto tipo, el valor más pequeño se promociona al valor más grande, ya que el tipo mayor siempre podrá representar cualquier valor del tipo menor (por ejemplo, de int a long o de float a double).

En este caso no se necesita escribir código para que la conversión se lleve a cabo. Ocurre cuando se realiza lo que se llama una conversión ancha (*widening casting*), es decir, cuando se coloca un valor pequeño en un contenedor grande.

Ejemplo:

```
int  num1 = 100;
long num2 = num1; //Un int "cabe" en un long
```

8.- Conversiones de tipo

Casting explícito

Cuando hacemos una conversión de un tipo con más bits a un tipo con menos bits.

En estos casos debemos indicar que queremos hacer la conversión de manera expresa, ya que se puede producir una pérdida de datos y hemos de ser conscientes de ello. Este tipo de conversiones se realiza con el **operador cast**.

El operador cast es un operador unario que se forma colocando delante del valor a convertir el tipo de dato entre paréntesis. Tiene la misma precedencia que el resto de operadores unarios y se asocia de izquierda a derecha. El formato general para indicar que queremos realizar la conversión es:

(tipo) valor_a_convertir

En el casting explícito sí es necesario escribir código. Ocurre cuando se realiza una conversión estrecha (*narrowing casting*), es decir, cuando se coloca un valor grande en un contenedor pequeño.

```
int num1   = 100;
short num2 = (short) num1;  //Casting explícito:
                            //short tiene menor rango que int
```

8.- Conversiones de tipo

Debemos tener en cuenta que un valor numérico nunca puede ser asignado a una variable de un tipo menor en rango, si no es con una conversión explícita.

Ejemplo:

```
int a;
byte b;

a = 12; // no se realiza conversión alguna
b = 12; // se permite porque 12 está dentro del rango permitido de valores para b
b = a;  // error, no permitido (incluso aunque 12 podría almacenarse en un byte)

byte b = (byte) a; // Correcto, forzamos conversión explícita
```

En el ejemplo anterior vemos un caso típico de error de tipos, ya que estamos intentando asignarle a **b** el valor de **a**, siendo **b** de un tipo más pequeño. Lo correcto es promocionar **a** al tipo de datos **byte**, y entonces asignarle su valor a la variable **b**.

8.- Conversiones de tipo

Tabla de Conversión de Tipos de Datos Primitivos									
		Tipo destino							
		boolean	char	byte	short	int	long	float	double
Tipo origen	boolean	-	N	N	N	N	N	N	N
	char	N	-	C	C	CI	CI	CI	CI
	byte	N	C	-	CI	CI	CI	CI	CI
	short	N	C	C	-	CI	CI	CI	CI
	int	N	C	C	C	-	CI	CI*	CI
	long	N	C	C	C	C	-	CI*	CI*
	float	N	C	C	C	C	C	-	CI
	double	N	C	C	C	C	C	C	-

N: Conversión no permitida (un boolean no se puede convertir a ningún otro tipo y viceversa).

CI: Conversión implícita o automática.

CI*: Conversión implícita o automática. Puede haber posible pérdida de datos.

C: Casting de tipos o conversión explícita.

8.- Conversiones de tipo

Reglas de Promoción de Tipos de Datos

Cuando en una expresión hay datos o variables de distinto tipo, el compilador realiza la promoción de unos tipos en otros, para obtener como resultado el tipo final de la expresión. Esta promoción de tipos se hace siguiendo unas reglas básicas en base a las cuales se realiza esta promoción de tipos, y resumidamente son las siguientes:

• Si uno de los operandos es de tipo double, el otro es convertido a double.

• En cualquier otro caso:

- Si el uno de los operandos es float, el otro se convierte a float
- Si uno de los operandos es long, el otro se convierte a long
- Si no se cumple ninguna de las condiciones anteriores, entonces ambos operandos son convertidos al tipo int.

8.- Conversiones de tipo

Conversión de números en Coma flotante (float, double) a enteros (int)

Cuando convertimos números en coma flotante a números enteros, la parte decimal se trunca (redondeo a cero). Si queremos hacer otro tipo de redondeo, podemos utilizar, entre otras, las siguientes funciones:

Math.round(num): Redondeo al siguiente número entero.
Math.ceil(num): Mínimo entero que sea mayor o igual a num.
Math.floor(num): Entero mayor, que sea inferior o igual a num.

Ejemplo:

```
double num = 3.5;

x = Math.round(num);  // x = 4
y = Math.ceil(num);   // y = 4
z = Math.floor(num);  // z = 3
```

8.- Conversiones de tipo

Conversiones entre caracteres (char) y enteros (int)

Como un tipo char lo que guarda en realidad es el código Unicode de un carácter, los caracteres pueden ser considerados como números enteros sin signo.

Ejemplo:

```
int num;
char c;

num = (int) 'A';                    // num = 65
c   = (char) 65;                    // c = 'A'
c   = (char) ((int) 'A' + 1);       // c = 'B'
```

8.- Conversiones de tipo

Conversiones de tipo con cadenas de caracteres (String)

Para convertir cadenas de texto a otros tipos de datos se utilizan las siguientes funciones:

```
num = Byte.parseByte(cad);
num = Short.parseShort(cad);
num = Integer.parseInt(cad);
num = Long.parseLong(cad);
num = Float.parseFloat(cad);
num = Double.parseDouble(cad);
```

Por ejemplo, si hemos leído de teclado un número que está almacenado en una variable de tipo String llamada cadena, y lo queremos convertir al tipo de datos byte, haríamos lo siguiente:

```
byte n = Byte.parseByte(cadena);
```

9.- Comentarios

9.- Comentarios

Los comentarios son muy importantes a la hora de describir qué hace un determinado programa.

A lo largo de la unidad los hemos utilizado para documentar los ejemplos y mejorar la comprensión del código. Para lograr ese objetivo, es normal que cada programa comience con unas líneas de comentario que indiquen, al menos, una breve descripción del programa, el autor del mismo y la última fecha en que se ha modificado.

Todos los lenguajes de programación disponen de alguna forma de introducir comentarios en el código. En el caso de Java, nos podemos encontrar los siguientes tipos de comentarios:

- ✓ **Comentarios de una sola línea**
- ✓ **Comentarios de múltiples líneas**
- ✓ **Comentarios Javadoc**

9.- Comentarios

Comentarios de una sola línea:

Utilizaremos el delimitador // para introducir comentarios de sólo una línea.

Ejemplo:

```
// comentario de una sola línea
```

Comentarios de múltiples líneas:

Para introducir este tipo de comentarios, utilizaremos una barra inclinada y un asterisco (/*), al principio del párrafo y un asterisco seguido de una barra inclinada (*/) al final del mismo.

Ejemplo:

```
/* Esto es un comentario
 * de varias líneas.
 * En concreto, de 3 líneas. */
```

9.- Comentarios

Comentarios Javadoc:

Utilizaremos los delimitadores /** y */. Al igual que con los comentarios tradicionales, el texto entre estos delimitadores será ignorado por el compilador.

Este tipo de comentarios se emplean para generar documentación automática del programa. A través del programa javadoc, incluido en JavaSE, se recogen todos estos comentarios y se llevan a un documento en formato .html.

Ejemplo:

```
/** Comentario de documentación.
 * Javadoc extrae los comentarios del código y
 * genera un archivo html a partir de este tipo de comentarios */
```

9.- Comentarios

Comentarios Javadoc:

Cuando se crea una clase nueva el código debe venir precedido por un comentario de documentación que incluye la descripción de la clase y, precedidas por las etiquetas `@author` y `@version` respectivamente, el nombre del autor o autores y el número de versión o fecha de creación de la clase.

En los comentarios de documentación en Java también se puede usar código html (por ejemplo, para resaltar texto en negrita   o para incluir un cambio de línea
).

Desde Eclipse se puede producir automáticamente la documentación html de un código comentado de esta forma. El resultado es un fichero html que se puede abrir con cualquier navegador y cuyo resultado es idéntico a la documentación online de Oracle sobre Java.

Bibliografía

Bibliografía

ORACLE Java Documentation:

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

¿Qué es Java?:

https://www.java.com/es/about/

Características del lenguaje de programación Java:

https://es.wikibooks.org/wiki/Programación_en_Java/Características_del_lenguaje

Utilización de los distintos lenguajes de programación:

https://www.tiobe.com/

20 años de Java: ¿En qué quedó el sueño de programar una vez, ejecutar en cualquier lugar?

https://www.xataka.com/aplicaciones/20-anos-de-java-celebramos-su-tremenda-influencia-en-el-mundo-del-software-y-la-programacion