Tema 2. Conceptos Básicos

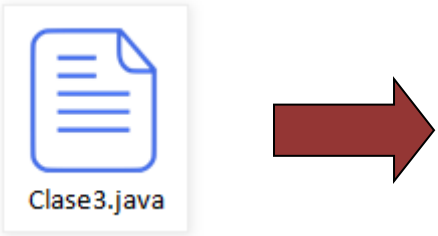
1. ESTRUCTURA DE UN PROGRAMA JAVA

Es muy importante tener en cuenta que Java es un Lenguaje Orientado a Objetos puro y no permite la posibilidad de programar mediante ninguna otra técnica que no sea ésta. Por esta razón la estructura general de un programa en Java tiene las siguientes características:

- Estará formado por uno o varios archivos fuente con extensión .java
- Cada uno de estos archivos puede contener una o varias clases.
- Si el archivo fuente contiene solo una clase y esta clase es pública, el nombre del archivo debe coincidir con el nombre de la clase.
- Si el archivo fuente contiene varias clases, solamente una de ellas podrá ser pública. En ese caso el nombre del archivo deberá coincidir con el nombre de la clase pública.
- Si el archivo fuente contiene una o varias clases y ninguna es pública, entonces el archivo se puede llamar de cualquier forma. No tiene por qué coincidir con el nombre de una de las clases que contiene.
- Para que un programa Java se pueda ejecutar debe contener una clase que llamaremos *Clase Principal* que tenga un método llamado *main*.
- El método main de un programa Java tiene la siguiente declaración:

```
public static void main(String [] args) {
        //Aquí se escribe el código del método main
}
```

Ejemplo: archivo fuente llamado *Clase3.java* que contiene tres clases, una de ellas pública.




```
class Clase1 {
  //Aquí se escribe el código de la Clase1
}

class Clase2 {
   //Aquí se escribe el código de la Clase2
}

public class Clase3 {
   //Aquí se escribe el código de la Clase3
}
```

Como la clase pública se llama Clase3, el nombre del archivo fuente que contiene estas clases debe llamarse Clase3.java. De otra forma obtendríamos un error de compilación.

Aunque un mismo archivo de código puede contener varias clases, lo más habitual es escribir cada una de las clases que componen el proyecto en un archivo fuente independiente:

Clase1.java

```
public class Clase1{
        //Código de Clase1
}
```

Clase2.java

```
public class Clase2{
        //Código de Clase2
}
```

Clase3.java

```
public class Clase3{
        //Código de Clase3
}
```

El concepto de clase es la base para la programación orientada a objetos y se estudia a fondo más adelante durante el curso, ahora no debes preocuparte si no tienes claro en qué consiste o porqué tienes que usarla. No es el momento de explicarlo ahora, solo debes conocer la estructura de un programa y empezar a escribir programas básicos.

Como ejemplo vamos a escribir el típico **programa *Hola Mundo***

El programa simplemente mostrará en pantalla el mensaje *"Hola Mundo!!!"* y acabará.

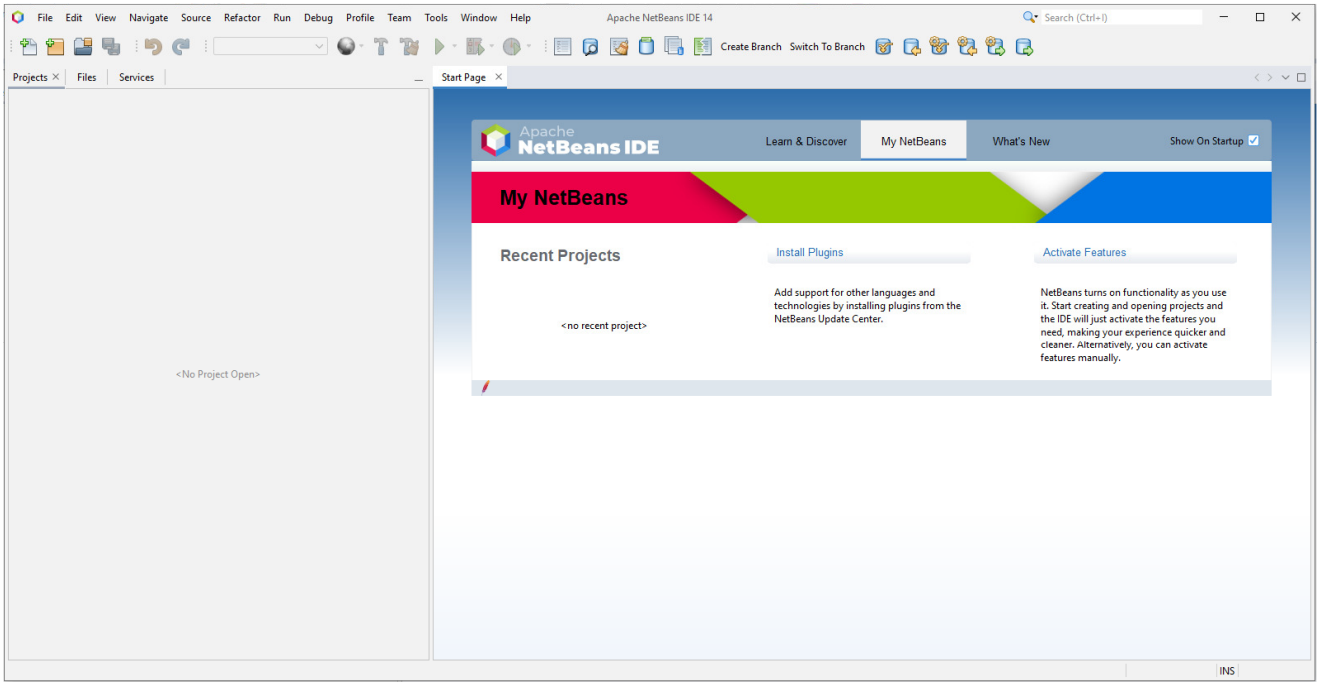
Como se trata de nuestro primer programa vamos a explicar los pasos a seguir para escribirlo y ejecutarlo.

2. PRIMER PROGRAMA JAVA: HolaMundo con NetBeans

1. Ejecutar NetBeans

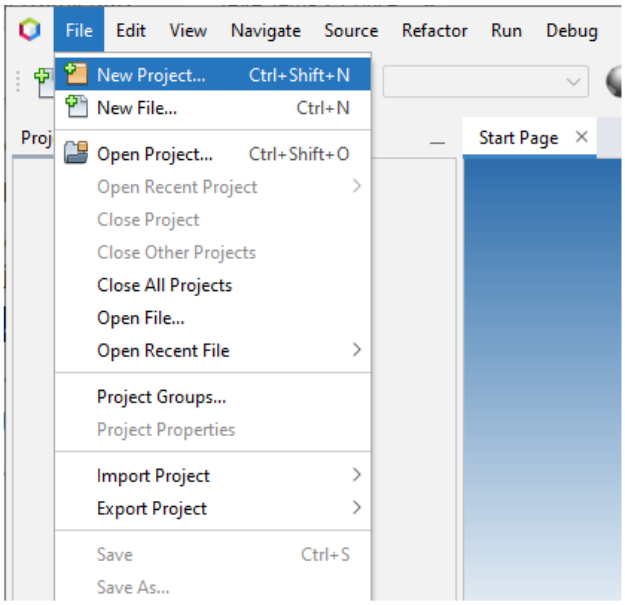
Haz doble click en el icono de NetBeans IDE en el escritorio para iniciar el IDE NetBeans.

Aparece la página de inicio de NetBeans:

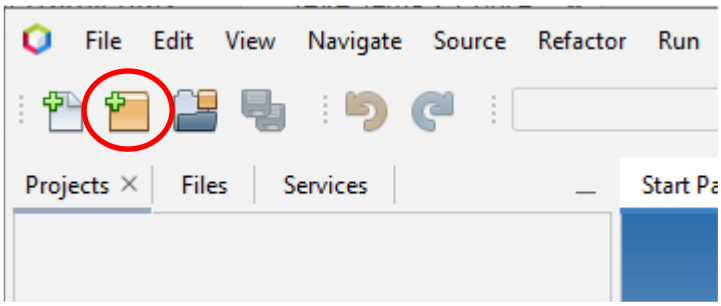


2. Crear un nuevo proyecto Java con NetBeans

Para crear un nuevo proyecto selecciona *File –> New Project*



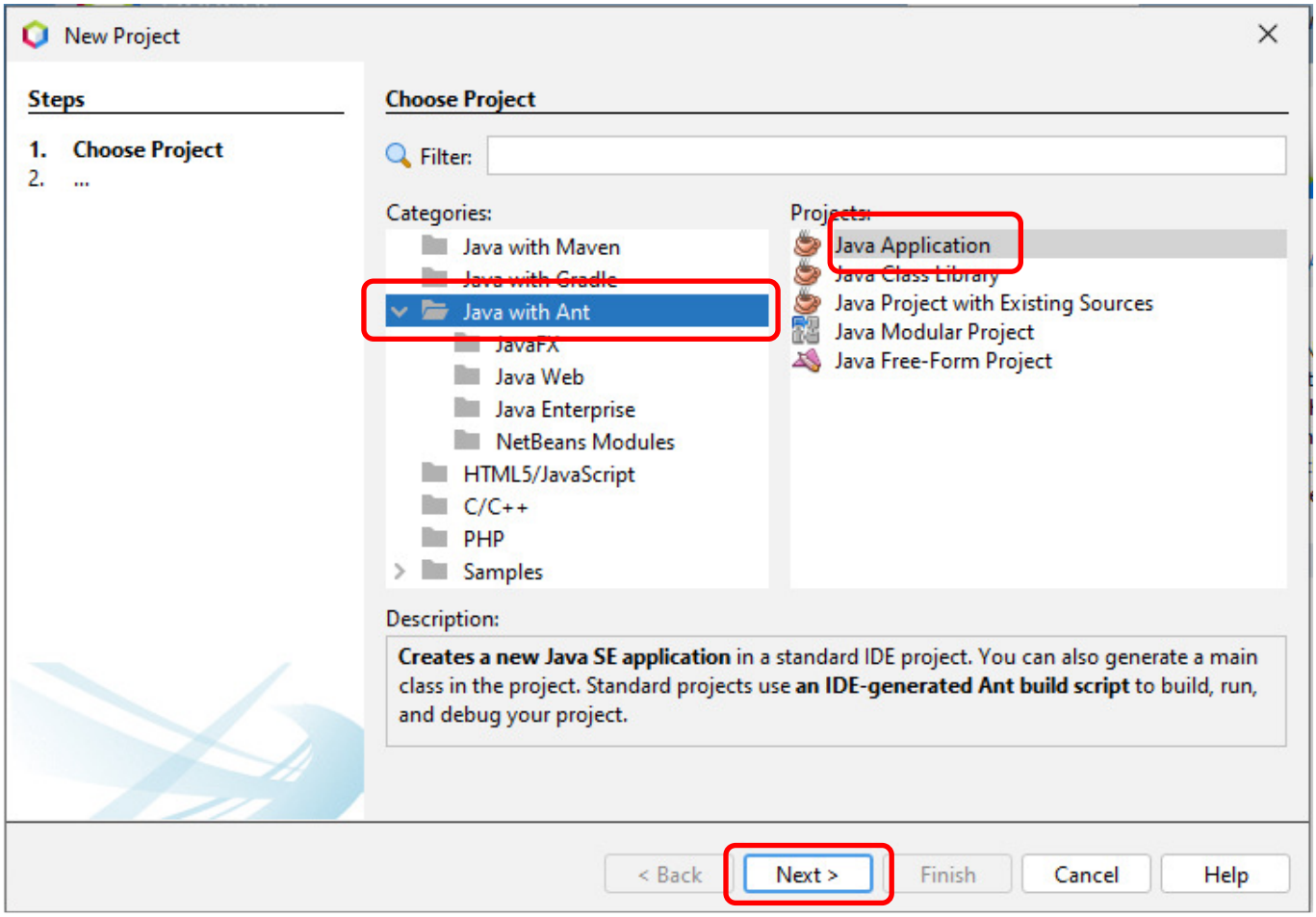
También se puede crear un proyecto nuevo pulsando sobre este icono:



Aparece el cuadro de diálogo de New Project.

En este cuadro de diálogo, en la sección **Categories** selecciona **Java with Ant** y en la sección **Projects** selecciona **Java Application**.

A continuación pulsa en el botón **Next**.



En la ventana que aparece a continuación:

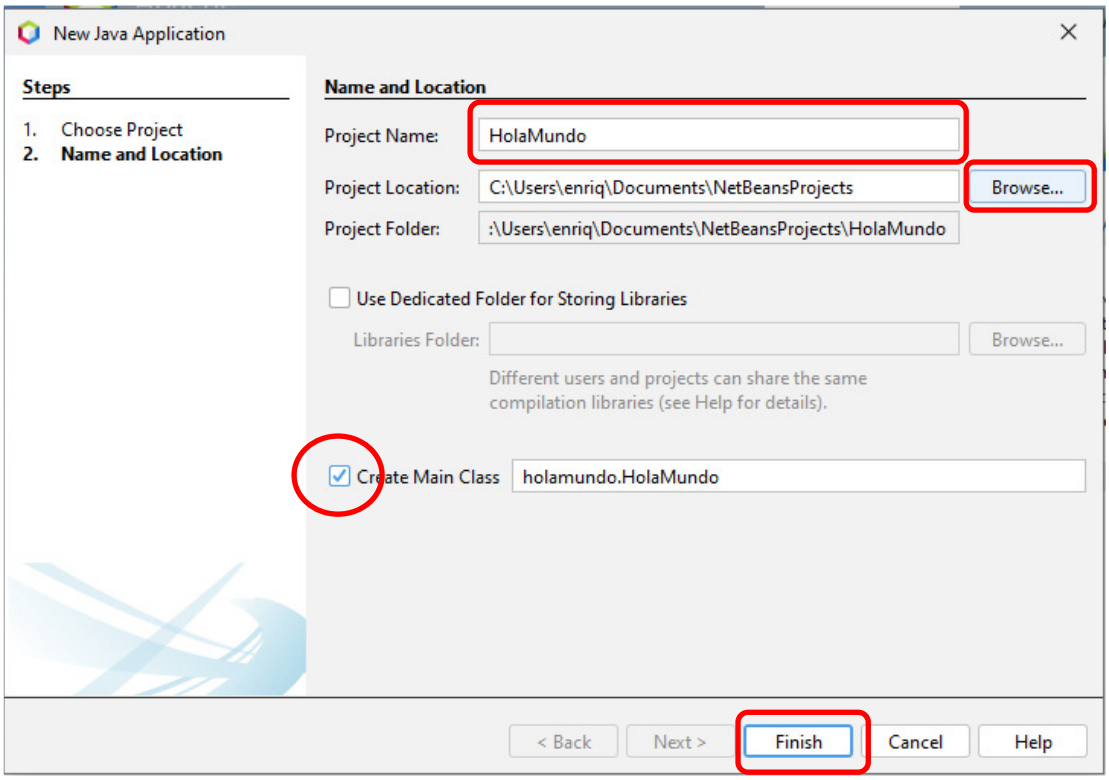
En el campo **Project Name** es donde escribimos el nombre del proyecto que estamos creando. En este caso escribimos ***HolaMundo***.

En **Project Location** se indica donde queremos guardar el proyecto. Por defecto, los proyectos se crean en la carpeta ***NetBeansProjects*** que se encuentra dentro de *Documents* (Carpeta Documentos del ordenador)

Si quieres cambiar el lugar donde se creará y guardará el proyecto pulsa en *Browse* y selecciona el lugar donde deseas guardarlo.

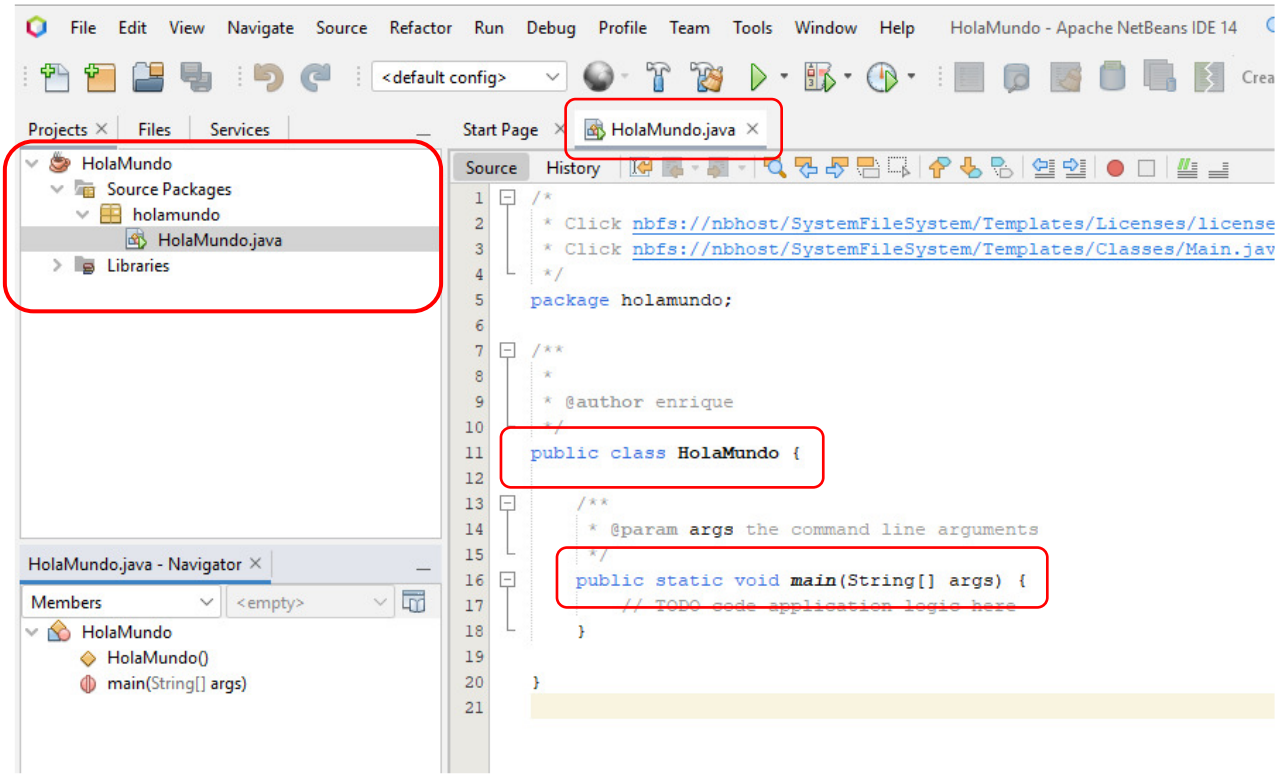
Deja seleccionada la opción *Create Main Class.* Seleccionando esta opción, NetBeans creará una clase principal para el proyecto que por defecto se llamará igual que el proyecto. Esta clase será la que contenga el método main. Si quieres cambiar el nombre de la clase principal, lo puedes modificar.

Finalmente, pulsa en *Finish* para crear el proyecto.



En la ventana que aparece a continuación vemos arriba a la izquierda el proyecto que acabamos de crear. El proyecto aparece dentro de la pestaña *Projects*.

La ventana grande de la derecha es la del **editor**. Aquí es donde se escribe el código del proyecto. Vemos el archivo fuente que nos ha creado NetBeans por defecto. Este archivo fuente se llama *HolaMundo.java*. Observa que el nombre del archivo coincide con el nombre de la clase pública que contiene. Observa también que dentro de esta clase se ha creado el método *main*.

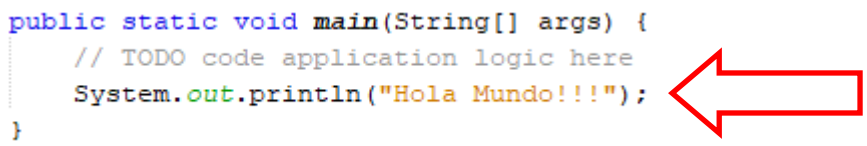


Lo que haremos ahora será escribir la instrucción que hará que aparezca el mensaje HolaMundo!!! por pantalla.

Escribe dentro del método *main* esta instrucción:

```
System.out.println("Hola Mundo!!!");
```

tal y como se muestra en la imagen:

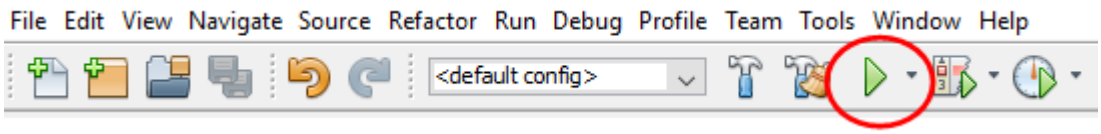


3. Ejecutar el proyecto

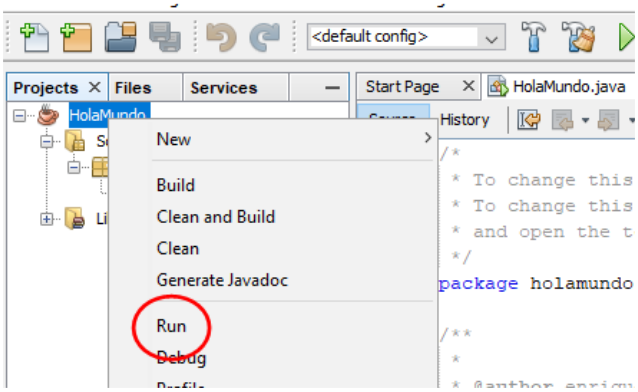
Para ejecutar el proyecto y ver el resultado de la ejecución del código lo podemos hacer de varias formas. Algunas de ellas son:

Pulsando la tecla F6.

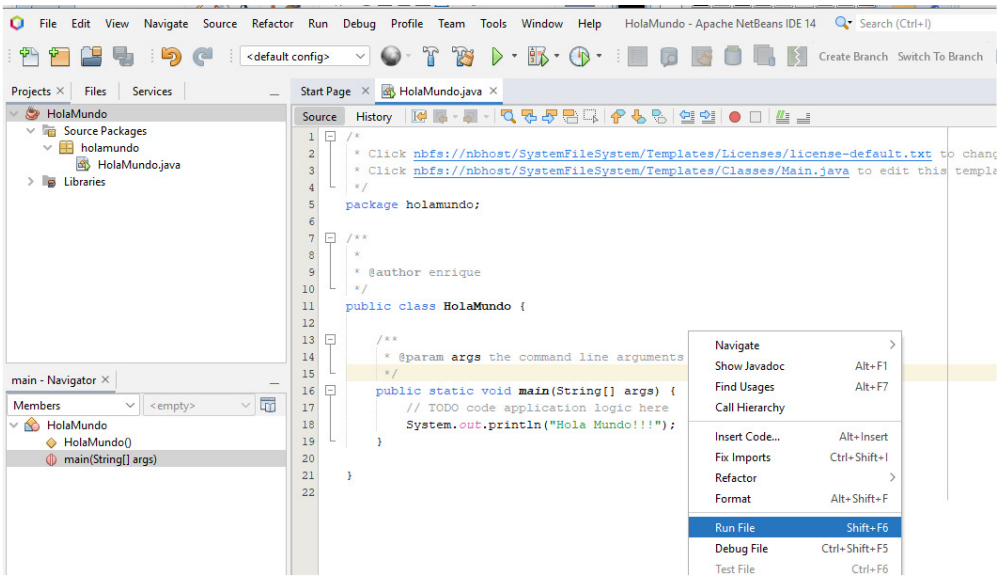
También lo podemos ejecutar pulsando sobre el icono *Run Project*



Otra opción es pulsar sobre el nombre del proyecto con el botón derecho del ratón y seleccionando *Run*.

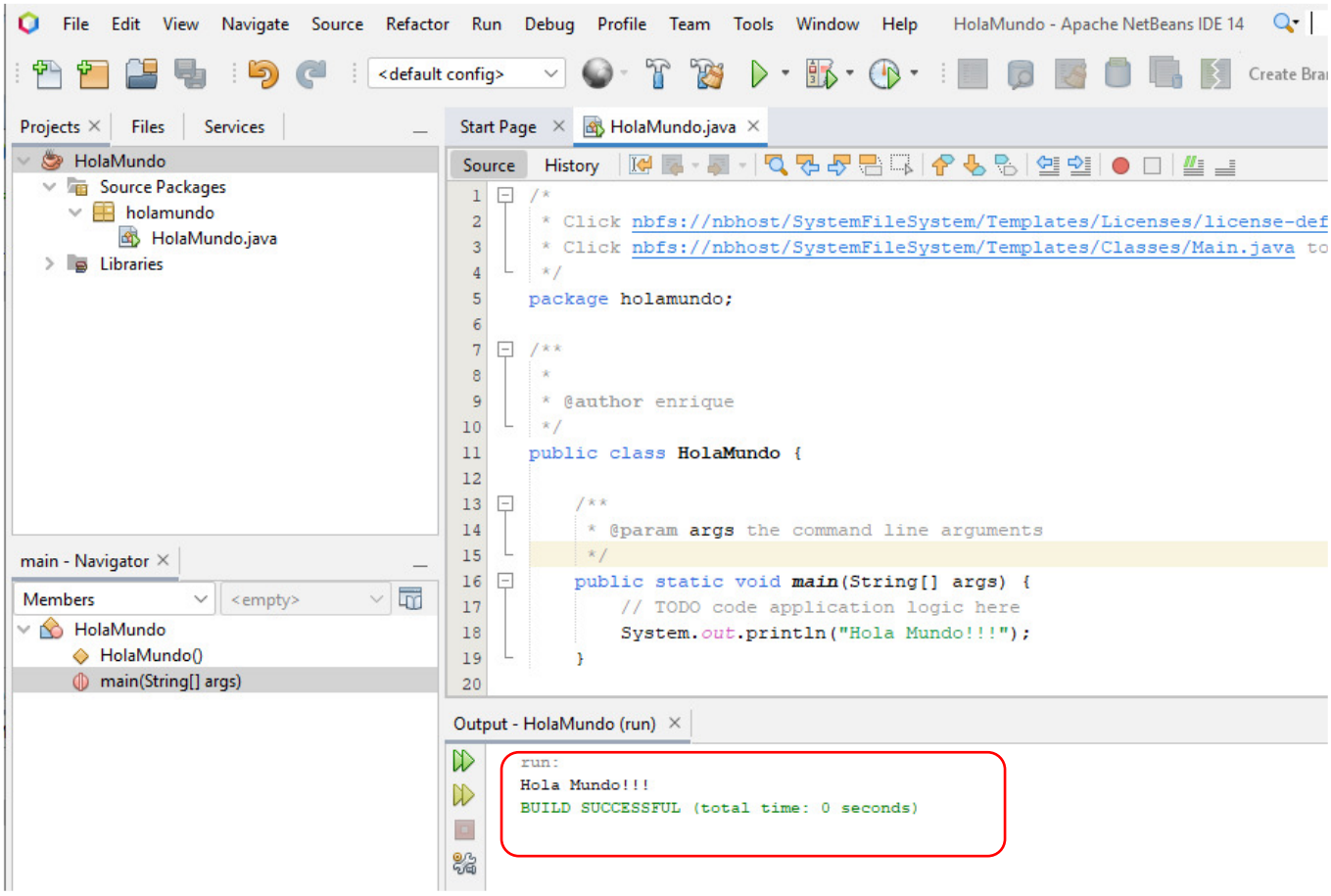


Otra forma de ejecutarlo es pulsar con el botón derecho del ratón sobre la clase principal y seleccionar la opción *Run File*



Elige cualquiera de ellas y verás el resultado de la ejecución del programa. En este caso se mostrará en la ventana de salida (*Output*) el mensaje *Hola Mundo!!!*

Además NetBeans nos indica que el programa se ha compilado y ejecutado de forma correcta mediante el mensaje *BUILD SUCCESSFUL*



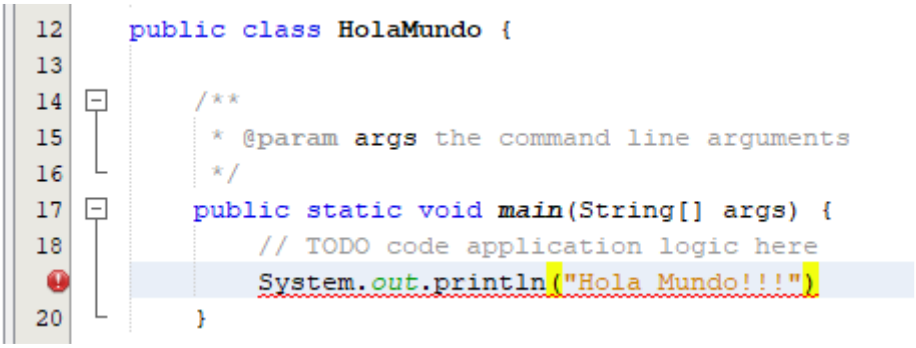
4. Corregir errores en un proyecto NetBeans

Si cometemos un error de sintaxis cuando escribimos el código, NetBeans enseguida lo detectará y nos avisará del error cometido para que lo solucionemos.

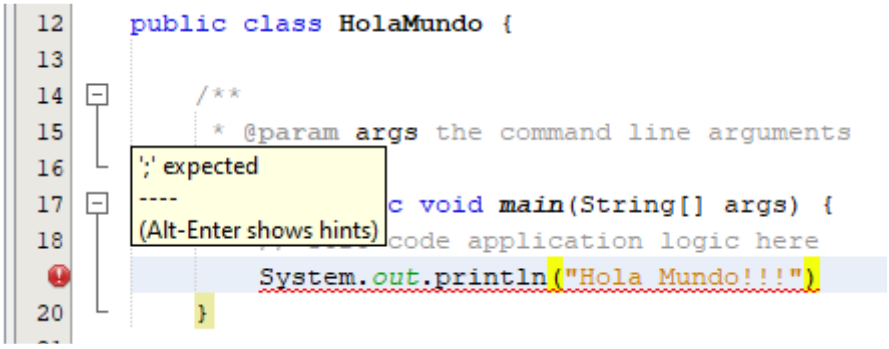
Vamos a ver cómo nos avisa NetBeans de los errores introduciendo un error sintáctico en el programa que acabamos de escribir.

El error que vamos a cometer es el siguiente: quita el punto y coma que aparece al final de la línea que has escrito. Este es un error muy común al empezar a programar en Java. Se nos olvida escribir el punto y coma con el que finalizan una gran mayoría de instrucciones Java.

Observa que NetBeans inmediatamente nos avisa del error colocando un círculo rojo con un signo de exclamación dentro en la línea donde se ha producido el error de sintaxis. Además la línea donde tenemos el error nos la subrayada en rojo.

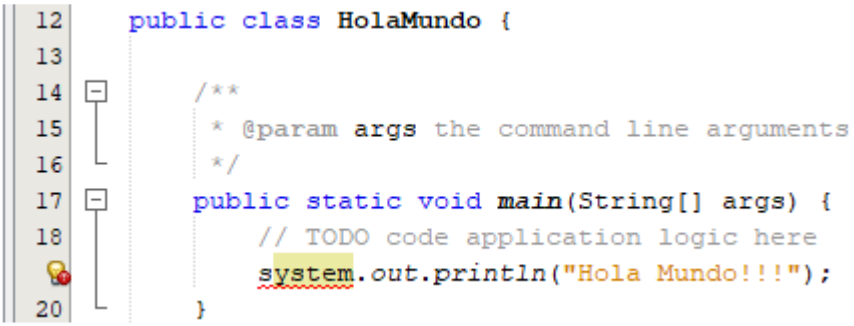


Si situamos el puntero del ratón sobre el círculo rojo sin pulsar en él, nos indica el tipo de error que estamos cometiendo, en este caso nos dice que *se espera un ;*

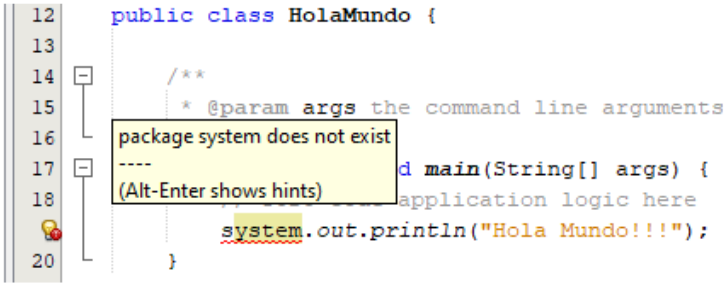


Corrige el error volviendo a escribir el punto y coma.

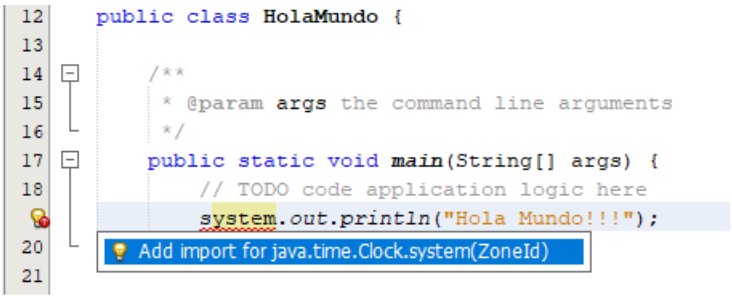
Vamos a cometer ahora otro error también común cuando empezamos a programar en Java. En este caso el error va a ser escribir la S de System en minúsculas. En este caso NetBeans nos avisa de una forma un poco diferente a como lo ha hecho en el caso anterior. Ahora nos aparece subrayada solo la palabra que está mal escrita y el símbolo de error al principio de la línea es una bombilla con un círculo rojo.



Igual que en el ejemplo anterior, si situamos el puntero sobre la bombilla sin pulsar nos da una idea de lo que está ocurriendo, en este caso nos dice que *system* no existe.



Pero este caso es diferente al anterior. Cuando aparece una bombilla como indicador de error podemos pulsar sobre ella y nos aparece una o varias acciones posibles que podemos realizar para corregirlo. En este caso NetBeans nos ofrece una posible solución para arreglar este error.



En este caso la solución que nos propone es importar una clase llamada system.

Importante: Mucho cuidado con hacer caso siempre a las sugerencias de NetBeans!! Es solo una posible solución que la mayoría de veces es la buena pero en este caso particular esta solución no nos sirve ya que esa clase no es la que necesitamos para mostrar un mensaje por pantalla. Para arreglarlo simplemente vuelve a cambiar la s por la S mayúscula.

Con esto ya hemos creado nuestro primer programa Java con NetBeans.

Aunque el programa es muy simple, contiene muchos de los conceptos de la programación orientada a objetos en Java.

La primera línea del programa:

```
public class HolaMundo{
```

declara una clase pública llamada *HolaMundo*.

Todo lo que se encuentre entre la llave abierta **{** y la llave cerrada **}** pertenece a la clase *HolaMundo*.

Este programa solo contiene la clase HolaMundo. La clase HolaMundo contiene el método main. En un programa Java, la clase que contiene el método main se le llama Clase principal.

Todo programa independiente escrito en Java empieza a ejecutarse a partir del método main().

El método main se declara de esta forma:

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

- **public**: indica que el método es público y, por tanto, puede ser llamado desde otras clases. **El método main debe ser público** para poder ejecutarse desde el intérprete Java.
- **static**: indica que no es necesario crear ningún objeto del la clase para poder utilizar el método. También indica que el método es el mismo para todas las instancias que pudieran crearse de la clase.
- **void**: indica que el método main no devuelve ningún valor.
- El método main tiene como parámetro un array de Strings, que contendrá los posibles argumentos que se le pasen al programa desde la línea de comandos, aunque como es nuestro caso, no se utilice.

No te preocupes si hay cosas que no has entendido. Todos estos conceptos se estudian en profundidad a lo largo del curso. Ahora solo debes saber que la cabecera del método main hay que escribirla siempre de esta forma.

La instrucción que realmente realiza el trabajo es:

```
System.out.println("Hola Mundo!!!");
```

Para mostrar información por pantalla se utiliza la clase System, que contiene un atributo *out* que a su vez contiene el método *println*.

println muestra el mensaje y realiza un salto de línea colocándose el cursor al principio de la línea siguiente.

Existe otro método (*print*) que muestra el mensaje pero no salta de línea.

Algunos aspectos a tener en cuenta a la hora de escribir código:

- Java diferencia entre mayúsculas y minúsculas.
- La mayoría de líneas de código terminan con punto y coma salvo excepciones que veremos durante el curso.
- Una instrucción puede ocupar más de una línea.

Vuelvo a repetir, no te preocupes si muchos de los conceptos teóricos que se han comentado no te han quedado claros. Lo importante ahora es que has conseguido crear tu primer programa Java.

3. CHARSET: EL CONJUNTO DE CARACTERES DE JAVA

Para escribir un programa Java como en cualquier otro lenguaje de programación utilizamos caracteres. Como es lógico, hay una serie de restricciones y no podemos utilizar cualquier carácter que queramos a la hora de escribir un programa. Cada lenguaje determina el conjunto de caracteres (character set o de forma más abreviada *charset*) que se pueden utilizar para escribir el código.

En muchos lenguajes de programación se utilizan los caracteres que contiene la tabla ASCII.

ASCII (*American Standard Code for Information Interchange* o Código Estándar Americano para el Intercambio de Información) es el estándar de codificación de caracteres más ampliamente utilizado. Asigna un valor numérico a cada letra, número, signo de puntuación y algunos otros caracteres especiales.

ASCII incluye **256 códigos** divididos en dos grupos: estándar y extendido, de 128 códigos cada uno. El conjunto **ASCII básico**, o estándar, utiliza **7 bits** para cada código, lo que da como resultado 128 códigos desde **0 hasta 127**.

ASCII básico

000	(nul)	016 ►	(dle)	032	sp	048	0	064	@	080	P	096	`	112	p
001 ☺	(soh)	017 ◄	(dc1)	033	!	049	1	065	A	081	Q	097	a	113	q
002 ☻	(stx)	018 ↕	(dc2)	034	"	050	2	066	B	082	R	098	b	114	r
003 ♥	(etx)	019 ‼	(dc3)	035	#	051	3	067	C	083	S	099	c	115	s
004 ♦	(eot)	020 ¶	(dc4)	036	$	052	4	068	D	084	T	100	d	116	t
005 ♣	(enq)	021 §	(nak)	037	%	053	5	069	E	085	U	101	e	117	u
006 ♠	(ack)	022 ▬	(syn)	038	&	054	6	070	F	086	V	102	f	118	v
007 •	(bel)	023 ↨	(etb)	039	'	055	7	071	G	087	W	103	g	119	w
008 ◘	(bs)	024 ↑	(can)	040	(	056	8	072	H	088	X	104	h	120	x
009	(tab)	025 ↓	(em)	041	)	057	9	073	I	089	Y	105	i	121	y
010	(lf)	026	(eof)	042	*	058	:	074	J	090	Z	106	j	122	z
011 ♂	(vt)	027 ←	(esc)	043	+	059	;	075	K	091	[	107	k	123	{
012 ♀	(np)	028 ∟	(fs)	044	,	060	<	076	L	092	\	108	l	124	|
013	(cr)	029 ↔	(gs)	045	-	061	=	077	M	093	]	109	m	125	}
014 ♫	(so)	030 ▲	(rs)	046	.	062	>	078	N	094	^	110	n	126	~
015 ☼	(si)	031 ▼	(us)	047	/	063	?	079	O	095	_	111	o	127	⌂

El conjunto **ASCII extendido** utiliza **8 bits** para cada código, dando como resultado 128 códigos adicionales, numerados desde el **128 hasta el 255**.

ASCII extendido

128	Ç	143	Å	158	₧	172	¼	186	‖	200	╚	214	╓	228	Σ	242	≥
129	ü	144	É	159	ƒ	173	¡	187	╗	201	╔	215	╫	229	σ	243	≤
130	é	145	æ	160	á	174	«	188	╝	202	╩	216	╪	230	µ	244	⌠
131	â	146	Æ	161	í	175	»	189	╜	203	╦	217	┘	231	τ	245	⌡
132	ä	147	ô	162	ó	176	░	190	╛	204	╠	218	┌	232	Φ	246	÷
133	à	148	ö	163	ú	177	▒	191	┐	205	═	219	█	233	Θ	247	≈
134	å	149	ò	164	ñ	178	▓	192	└	206	╬	220	▄	234	Ω	248	°
135	ç	150	û	165	Ñ	179	│	193	┴	207	╧	221	▌	235	δ	249	∙
136	ê	151	ù	166	ª	180	┤	194	┬	208	╨	222	▐	236	∞	250	·
137	ë	152	ÿ	167	º	181	╡	195	├	209	╤	223	▀	237	φ	251	√
138	è	153	Ö	168	¿	182	╢	196	─	210	╥	224	α	238	ε	252	ⁿ
139	ï	154	Ü	169	⌐	183	╖	197	┼	211	╙	225	ß	239	∩	253	²
140	î	155	¢	170	¬	184	╕	198	╞	212	╘	226	Γ	240	≡	254	■
141	ì	156	£	171	½	185	╣	199	╟	213	╒	227	π	241	±	255	
142	Ä	157	¥												

En el conjunto de caracteres ASCII básico, los primeros 32 valores (Códigos del 0 al 31) son caracteres no imprimibles y están asignados a códigos de control. Por ejemplo, el código 13 corresponde a la tecla

ENTER representada en la tabla por cr (CR por *carriage return* o "retorno de carro" término que provienen de las máquinas de escribir). También existen caracteres de control usados en teleprocesamiento como ACK (Acknowledge - aviso de mensaje recibido), BEL (bell - aviso por señal sonora), ETX (end of text – fin de texto), STX (start of text – comienzo de texto), etc.

Los 96 códigos restantes del código básico corresponden a los caracteres imprimibles y se asignan a los dígitos del 0 al 9, a las letras mayúsculas y minúsculas del alfabeto anglosajón, operadores aritméticos, etc.

Los códigos correspondientes al ASCII extendido, del 128 al 255, se asignan a aquellos caracteres que no pertenecen al alfabeto anglosajón, por ejemplo, las vocales con tilde, la ñ y en general todos los caracteres especiales que utilizan los distintos lenguajes.

Debido a lo limitado de su tamaño, el código ASCII no es suficiente para representar caracteres de alfabetos como el Japonés, Chino o árabe. La solución a este problema ha sido crear un código más grande con el que poder representar cualquier carácter de cualquier idioma: el código Unicode.

El código **UNICODE** proporciona una única representación numérica para cada símbolo, independientemente del ordenador, el programa o el lenguaje de programación que se use.

La codificación Unicode se ha transformado en un estándar adoptado por las principales empresas de hardware y software.

Java utiliza la codificación Unicode para la representación de caracteres. El código Unicode actualmente representa los caracteres de la mayoría de idiomas escritos en todo el mundo.

Los 127 primeros caracteres de Unicode corresponden al código ASCII básico.

La descripción completa del estándar y las tablas de caracteres están disponibles en la página web oficial de Unicode https://home.unicode.org/

Java soporta el sistema de codificación UNICODE y esto supone que el conjunto de caracteres del lenguaje sea muy amplio. ASCII es un subconjunto de UNICODE por lo tanto los caracteres ASCII siguen siendo válidos en Java.

Por lo tanto, una vez visto lo anterior, los caracteres que pueden aparecer en un programa Java para formar las constantes, variables, expresiones, etc., son estos:

- Las **letras mayúsculas y minúsculas** de la A(a) a la Z(z) de los alfabetos internacionales. La ñ y Ñ son válidas así como las vocales acentuadas.
- **Dígitos** (0, 1, 2, ..., 9)
- Los caracteres ' _ ' '$' y cualquier otro carácter considerado como letra en el sistema de codificación Unicode.
- Los **operadores y caracteres especiales** siguientes:
  `+ - * / = % & # ! ? ^ " ` ~ \ | < > ( ) [ ] { } :: ; . , ... @`
- **Separadores**: espacio, tabulador y salto de línea.
  Los separadores como el tabulador y el salto de línea ayudan a que el programa sea más legible por las personas.

  Por ejemplo, podemos escribir el método main de la forma:

  ```
  public static void main(String [] args){System.out.println("Hola Mundo!!!");}
  ```

  El programa se ejecuta de forma correcta pero queda mucho más claro si al escribirlo introducimos tabuladores y saltos de línea:

  ```
  public static void main(String [] args){
         System.out.println("Hola Mundo!!!");
  }
  ```

- **Secuencias de escape**: Una secuencia de escape está formada por una barra inversa seguida de una letra, un carácter o de una combinación de dígitos.

  Una secuencia de escape siempre representa un solo carácter aunque se escriba con dos o más caracteres.

  Se utilizan para realizar acciones como salto de línea o para usar caracteres no imprimibles. Algunas secuencias de escape definidas en Java son:

Secuencia de escape	Descripción
\n	Salto de línea. Sitúa el cursor al principio de la línea siguiente
\b	Retroceso. Mueve el cursor un carácter atrás en la línea actual.
\t	Tabulador horizontal. Mueve el cursor hacia adelante una distancia determinada para el tabulador.
\r	Mueve el cursor al principio de la línea actual.
\"	Comillas. Permite mostrar por pantalla el carácter *comillas dobles*
\'	Comilla simple. Permite mostrar por pantalla el carácter *comilla simple*
\\	Barra inversa.
\udddd	Carácter Unicode. d representa un dígito hexadecimal del carácter Unicode

Ejemplos de uso de las secuencias de escape:

Vemos algunos ejemplos de uso de las secuencias de escape para entender mejor para qué sirven:

Ejemplo 1: uso de la secuencia de escape \n o salto de línea. Provoca un salto al principio de la línea siguiente en el lugar donde se coloca.

```
System.out.println("Juan\nVictor\nLuis\nJordi");
```

Salida por pantalla:

```
Juan
Victor
Luis
Jordi
```

Ejemplo 2: Uso de la secuencia de escape \r. Provoca que el cursor se sitúe al principio de la línea siguiente.

```
System.out.println("Lunes\rMartes, Miércoles");
```

Salida por pantalla:

```
Martes, Miércoles
```

Ejemplo 3: Uso de la secuencia de escape \b. Provoca que el cursor retroceda un carácter.

```
System.out.println("Lunes\bMartes");
```

Salida por pantalla:

```
LuneMartes
```

Ejemplo 4: Uso de la secuencia de escape \t. Provoca que el cursor avance una distancia determinada.

```
System.out.println("Lunes\tMartes\tMiércoles");
```

Salida por pantalla:

```
Lunes	Martes	Miércoles
```

Ejemplo 5: Uso de las secuencias de escape \" y \'. Permite mostrar estos caracteres por pantalla.

```
System.out.println("\"Lunes\",\"Martes\",\'Miércoles\'");
```

Salida por pantalla:

```
"Lunes","Martes",'Miércoles'
```

4. IDENTIFICADORES

Los identificadores son los nombres que el programador asigna a variables, constantes, clases, métodos, paquetes, etc. de un programa.

Características de un identificador Java:

- Están formados por letras y dígitos.
- No pueden empezar por un dígito.
- No pueden contener ninguno de los operadores y caracteres especiales vistos en el punto anterior.
- No puede ser una palabra reservada del lenguaje. Las palabras reservadas de Java aparecen en el punto siguiente. Las palabras **true**, **false** y **null** aunque no son palabras reservadas tampoco pueden usarse como identificadores.

Ejemplo de identificadores válidos:

`Edad`	`nombre`	`_Precio`	`Año`	`año_nacimiento`
`AÑO0`	`$cantidad`	`_$cantidad`	`cantidad_10_1`	`PrecioVenta`
`num4`	`bl4nc0`	`miércoles`	`PrIvAdo`	`máximo`

En los ejemplos se puede comprobar que tanto los caracteres ñ y Ñ como las vocales acentuadas son caracteres válidos para crear identificadores en Java.

Java diferencia mayúsculas y minúsculas, por lo tanto, *nombre* y *Nombre* son identificadores distintos.

Ejemplo de identificadores NO válidos:

`4num` -> Identificador no válido porque comienza por un dígito

`z#` -> No válido porque contiene el carácter especial #

`"Edad"` -> No válido porque no puede contener comillas

`Tom's` -> No válido porque contiene el carácter '

`año-nacimiento` -> no válido porque contiene el carácter -

`public` -> no válido porque es una palabra reservada del lenguaje

`__precio@final` -> no válido porque contiene el carácter @

5. PALABRAS RESERVADAS

Las palabras reservadas son identificadores predefinidos que tienen un significado para el compilador y por tanto no pueden usarse como identificadores creados por el usuario en los programas.

Las palabras reservadas en Java son las siguientes:

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

* las palabras const y goto siguen siendo reservadas pero no se usan en la actualidad.

6. COMENTARIOS

Un comentario es un texto que se escribe dentro de un programa con el fin de facilitar la comprensión del mismo. Se utilizan para explicar el código fuente.

En Java se pueden utilizar tres tipos de comentarios:

Comentario tradicional

Son los mismos que se usan en C/C++

Las características de los comentarios tradicionales Java son las siguientes:

- Empieza con los caracteres /* y acaba con */.
- Pueden ocupar más de una línea y pueden aparecer en cualquier lugar donde pueda aparecer un espacio en blanco.
- No pueden anidarse.

Ejemplos:

```
/* Programa Ecuación segundo grado
   Calcula las soluciones de una ecuación de segundo grado */

/* Lectura de datos por teclado */

/*
    Salida de datos por pantalla
 */
```

Ejemplo de comentario no válido:

```
/* linea comentario 1
            /* linea comentario 2
               linea comemtario 3
            */
   linea comentario 4
*/
```

Comentario no válido. Los comentarios no se pueden anidar.

Comentarios de una sola línea.

Las características de los comentarios de una línea son las siguientes:

- Comienzan con una doble barra (//)
- Pueden escribirse al principio de la línea o a continuación de una instrucción.
- No tienen carácter de terminación.

Ejemplos:

```
// Programa Ecuación segundo grado
// Calcula las soluciones de una ecuación de segundo grado

//Salida de datos por pantalla

double p; // precio del producto
```

Comentarios de documentación.

Son comentarios especiales para generar documentación del programa.

También se conocen como comentarios Javadoc.

Comienza con /** y termina con */

Ejemplo:

```
/**
 *
 * @author Enrique García
 * @version 2.0
 *
 */
```

7. TIPOS DE DATOS

REPRESENTACIÓN INTERNA DE LOS DATOS

En el mundo real los datos que manejamos se representan mediante letras, números, símbolos, imágenes, sonidos, etc.

Esto se conoce como **representación externa** de los datos.

Pero si queremos introducirlos en un ordenador, todos estos elementos se deben transformar ó codificar.

Un ordenador está compuesto fundamentalmente por circuitos electrónicos digitales. Los datos circulan por estos circuitos en forma de impulsos eléctricos.

De forma muy simplificada podemos decir que por un circuito pasa o no pasa corriente y esto lo podemos representar con dos dígitos: 0 y 1.

Todos los datos e información que contiene un ordenador, están representados de forma interna mediante secuencias de ceros y unos.

Un sistema de representación que utiliza solamente dos símbolos 0 - 1 se llama **sistema binario.**

Por tanto, los datos tal y como los expresamos de forma natural se deben codificar de forma interna en binario para que puedan ser tratados por el ordenador.

El sistema binario utiliza solamente dos dígitos 0 y 1 llamados **bits**.

La palabra bit procede de la unión de las palabras **bi**nary digi**t**.

Un bit es la unidad mínima de representación de información.

Utilizando 1 bit podremos solamente representar dos valores posibles: 0, 1.

Utilizando 2 bits podemos representar 4 valores: 00, 01, 10, 11.

Utilizando 3 bits podemos representar 8 valores:

000, 001, 010, 011, 100, 101, 110, 111.

Utilizando 4 bits podemos representar 16 valores:

0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111,

1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

En general, utilizando **N bits** podremos representar **2^N valores**.

Ocho bits forman un **byte**.

El byte es la unidad básica de medida de la información.

Un byte es la cantidad más pequeña de información que el ordenador puede manejar.

Con un byte se pueden representar $2^8 = 256$ caracteres.

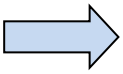
En el interior del ordenador los datos se transmiten y almacenan en grupos de bytes llamados **palabras.**

La longitud de palabra depende de cada tipo de ordenador: 8, 16, 32, 64.

La mayoría de ordenadores de propósito general utilizan palabras de 32 o 64 bits.

TIPOS DE DATOS EN JAVA

Java es un **lenguaje de programación fuertemente tipado**. Esto significa que, por ejemplo, antes de usar cualquier variable debemos indicar de qué tipo es. Este tipo determinará el conjunto de valores que puede guardar.



En Java existen dos tipos principales de datos:

- Datos de tipo básico o primitivo.
- Referencias a objetos.

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

Java soporta 8 tipos de datos básicos + void:

TIPOS DE DATOS BÁSICOS EN JAVA					
Tipo de dato	**Representación**	**Tamaño (Bytes)**	**Rango de Valores**	**Valor por defecto**	**Clase Asociada**
byte	Numérico Entero con signo	1	-128 a 127	0	Byte
short	Numérico Entero con signo	2	-32.768 a 32.767	0	Short
int	Numérico Entero con signo	4	-2147483648 a 2147483647	0	Integer
long	Numérico Entero con signo	8	-9223372036854775808 a 9223372036854775807	0	Long
float	Numérico en Coma flotante de precisión simple Norma IEEE 754	4	$\pm$ 3.4x10^{-38} a $\pm$ 3.4x10^{38}	0.0	Float
double	Numérico en Coma flotante de precisión doble Norma IEEE 754	8	$\pm$ 1.8x10^{-308} a $\pm$ 1.8x10^{308}	0.0	Double
char	Carácter Unicode	2	\u0000 a \uFFFF	\u0000	Character
boolean	Dato lógico	-	true ó false	false	Boolean
void	-	-	-	-	Void

Aunque void no es un tipo de dato básico en java, se utiliza para indicar que algo no tiene tipo. Por ejemplo, cuando se escribe un método se debe indicar el tipo del valor que devuelve. Si un método no devuelve nada se utiliza el *tipo* void.

Java también proporciona el tipo **String** para almacenar cadenas de caracteres.

El tipo String **no es un tipo primitivo.** Los String en java son objetos. Aunque los String se explicarán a fondo más adelante, su uso es muy común en cualquier programa y por eso es importante que se conozcan desde el principio.

Cuando necesitamos procesar enteros cuyo valor está fuera del rango de los tipos int o long, Java proporciona el tipo **BigInteger**.

El tipo BigInteger **no es un tipo primitivo.** Los números de tipo BigInteger son objetos.

Veremos algún ejemplo de uso durante el curso. En este enlace puedes ver documentación y ejemplos.

https://codegym.cc/es/groups/posts/es.648.clase-biginteger-de-java

Java también proporciona el tipo **BigDecimal** para trabajar con números reales con mayor precisión que la que ofrecen los tipos float y double.

DATOS NUMÉRICOS ENTEROS

En Java los representan los tipos: *byte, short, int, long*.

El tipo de dato numérico entero es un subconjunto finito de los números enteros.

Pueden ser positivos o negativos.

Ejemplo de declaración de variables de tipo entero:

```
int a;
byte n1, n2;
short x;
int valor = -7;
long numero = 4;
```

DATOS NUMÉRICOS REALES

En Java los representan los tipos: *float, double*.

El tipo de dato numérico real es un subconjunto finito de los números reales. Siempre llevan un punto decimal y también pueden ser positivos o negativos. Los números reales tienen una parte entera y una parte decimal.

Por ejemplo: 0.08 -54.0001

Ejemplo de declaración de variables reales:

```
float peso;
double longitud;
float altura = 2.5F;
double peso = 32.7;
double area = 1.7E4; // equivale a 1.7 * 10⁴
double z = .123; //si la parte entera es 0 se puede omitir
```

DATOS DE TIPO CARÁCTER

En Java se representa con el tipo *char*.

Un dato de tipo carácter se utiliza para representar un carácter dentro del rango \u0000 a \uFFFF (números desde 0 hasta 65535) en **Unicode**.

En realidad **un dato de tipo char contiene un número entero** dentro del rango anterior que representa al carácter.

Ejemplos de declaración de variables de tipo carácter:

```
char car;
char letra1 = 'z';
char letra = '\u0061'; //código unicode del carácter 'a'
```

DATOS DE TIPO LÓGICO

Se representan con el tipo *boolean*.

Los datos de este tipo sólo pueden contener dos valores: true (verdadero) ó false (falso).

Ejemplo de declaración de variables lógicas:

```
boolean primero;
boolean par = false;
```

Los tipos de datos lógicos son también conocidos como **booleanos** en honor del matemático inglés George Bool, que desarrolló la teoría conocida como álgebra de bool que fue la base para la representación de los circuitos lógicos.

8. LITERALES

Un literal en Java es un valor de tipo entero, real, lógico, carácter, cadena de caracteres o un valor nulo (null), que puede aparecer dentro de un programa.

Por ejemplo: 150, 12.4, "ANA", null, 't'.

Los literales suelen aparecer en la asignación de valores a las variables o formando parte de expresiones aritméticas y lógicas. Por ejemplo:

```
x = 25;
precio = 120.99;
mes = "enero";
descuento  = precio - (precio * 5 / 100);
if(a > 3 && a < 5)
```

Cuando se utilizan literales, es muy importante tener en cuenta que Java es un lenguaje **fuertemente tipado**, esto quiere decir que realiza un control estricto de los tipos de datos que pueden contener las variables.

Por ejemplo, si una variable se ha declarado como *int* no podremos asignarle un valor de tipo *double*.

```
int n = 2.5; //Esta asignación no es correcta
```

Esta asignación provocaría un error ya que estamos asignando un valor no entero (2.5) a una variable n declarada como int. Por lo tanto es muy importante saber de qué tipo es un literal para trabajar de forma correcta con ellos y no obtener resultados inesperados.

Los literales en Java pueden ser:

- Literales de tipo entero
  - Literales en base decimal
  - Literales en binario
  - Literales en base octal
  - Literales en base hexadecimal
- Literales de tipo real
- Literales de tipo carácter
- Literales de cadenas de caracteres

LITERALES DE TIPO ENTERO

El tipo del literal entero es **int** a no ser que su valor absoluto sea mayor que el de un int o se especifique el sufijo l (ele minúscula) ó L en cuyo caso será de tipo **long**.

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

Los literales de tipo entero pueden expresarse de cuatro formas distintas:

⇒ base decimal (base 10), binario (base 2), octal (base 8) y hexadecimal (base 16).

Literales de tipo entero en Base decimal

Están formados por 1 o más dígitos del 0 al 9.

El primer dígito debe ser distinto de cero.

Por ejemplo:

```
982                literal entero de tipo int
-25                literal entero de tipo int
+1200              literal entero de tipo int
1234               literal entero de tipo int
1234L              literal entero de tipo long. Lo indica la L final
123400000000       literal entero de tipo long porque supera el rango de los int
```

Literales de tipo entero en Binario

Están formados por 1 o más dígitos del 0 al 1.

Deben comenzar por 0b ó 0B.

Por ejemplo:

```
0b00100001              literal entero binario de tipo int
0B0010000101000101      literal entero binario de tipo int
0b1010000101000101101000010100010110100001010001011010000101000101L;
literal entero binario de tipo long
```

Literales de tipo entero en Base octal

Están formados por 1 o más dígitos del 0 al 7.

El primer dígito debe ser cero.

Por ejemplo:

```
01234
025
```

Literales de tipo entero en Base hexadecimal

Están formados por 1 o más dígitos del 0 al 9 y letras de la A a la F (mayúsculas o minúsculas).

Deben comenzar por 0x ó 0X.

Por ejemplo:

```
0x1A2
0x430
0Xf4
```

LITERALES DE TIPO REAL

Son números que deben llevar un parte entera, un punto decimal y una parte fraccionaria. Si la parte entera es cero, puede omitirse.

Los literales de tipo real solo se pueden expresar en base decimal.

El signo + al principio es opcional y el signo – será obligatorio si el número es negativo.

Por ejemplo:

```
12.2303
-3.44
+10.33
0.456
.96
```

También pueden representarse utilizando la notación científica. En este caso se utiliza una E (mayúscula o minúscula) seguida del exponente (positivo o negativo). El exponente está formado por dígitos del 0 al 9.

Por ejemplo, el número $14*10^{-3}$ en notación científica se escribe `14E-3`

Más ejemplos:

`2.15E2` -> $2.15 * 10^{2}$

`.0007E4` -> $0.0007 * 10^{4}$

`-50.445e-10` -> $-50.445 * 10^{-10}$

El tipo de estos literales es siempre **double**, a no ser que se añada el sufijo F ó f para indicar que es float.

Por ejemplo:

```
2.15              literal de tipo double
2.15F             literal de tipo float
.23e5             literal de tipo double
+74.336E-2F       literal de tipo float
```

También de forma opcional se pueden utilizar los sufijos d ó D para indicar que el literal es de tipo double:

```
12.002d    literal de tipo double
```

Los literales de tipo entero escritos en cualquier base y los literales de tipo real, pueden contener el carácter _ para facilitar la lectura del número.

Por ejemplo:

```
2_014_120                                   0xCAFE_BABE
1234_5678_9019_3456L                        0x7fff_ffff_ffff_ffffL
999_99_9999L                                0b0010_0101
3.14_15F                                    0b11010010_01101001_10010100_10010010
0xFF_EC_DE_5E
```

Este carácter solo puede aparecer entre dígitos. No puede aparecer en los siguientes lugares:

- Al principio o final de un número.
- Junto a un punto decimal en un número de tipo real.
- Antes de los sufijos F ó L.
- Entre el 0X hexadecimal o el 0B binario

Ejemplos de literales válidos y no válidos que contienen el carácter _

```
3_.1415F            No Válido
3._1415F            No Válido
999_99_9999_L       No Válido
_52                 No Válido
5_2                 Válido
52_                 No Válido
5_______2           Válido
0_x52               No Válido
0x_52               No Válido
0x5_2               Válido
0x52_               No Válido
0_52                Válido
05_2                Válido
052_                No Válido
```

LITERALES DE TIPO CARÁCTER

Contienen un solo carácter encerrado entre **comillas simples**.

Son de tipo **char**.

Las secuencias de escape se consideran literales de tipo carácter.

Por ejemplo:

```
'a'
'4'
'\n'
'\u0061'
```

LITERALES DE CADENAS DE CARACTERES

Están formados por 0 ó más caracteres encerrados entre **comillas dobles**.

Pueden incluir secuencias de escape.

Por ejemplo:

```
"Esto es una cadena de caracteres"
"Pulsa \"C\" para continuar"
""    -> cadena vacía
"T"   -> cadena de un solo carácter, es diferente a 't' que es de tipo char.
```

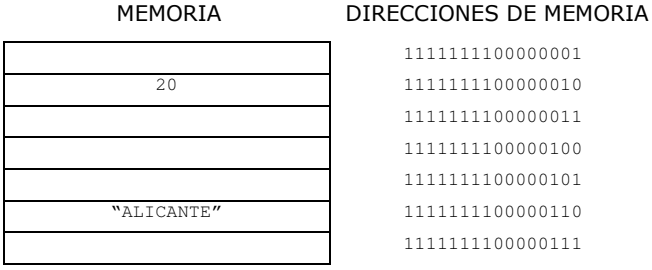
Las cadenas de caracteres en Java son objetos de tipo **String**. Se estudian a fondo más adelante.

9. VARIABLES

Los datos que maneja un programa se almacenan en la memoria del ordenador. A cada dato se le asigna una posición o dirección de memoria. Esta dirección será la que se use para acceder al dato.

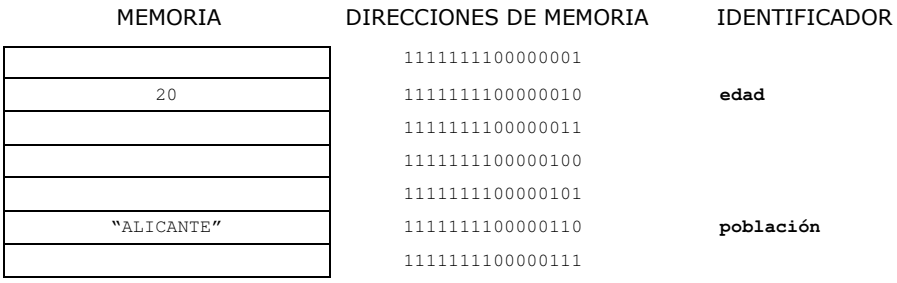
Las direcciones de memoria se expresan mediante un número binario o hexadecimal.

La siguiente figura muestra de una forma muy simplificada como se guardan los datos en memoria.

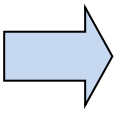


Podemos ver que la dirección 1111111100000010 contiene el valor 20 y la dirección 1111111100000110 contiene el valor "ALICANTE".

Para acceder a ellos necesitaríamos conocer la dirección de memoria o sea toda la secuencia de ceros y unos. Para facilitar el manejo de las posiciones de memoria dentro de un programa, se sustituye este número binario por un identificador. Tal como vimos en el punto 4 del tema, los identificadores son los nombres que el programador asigna a variables, constantes, clases, métodos, paquetes, etc. dentro de un programa.



De esta forma en lugar de utilizar en nuestro programa 1111111100000010 utilizaremos *edad*. El sistema se encargará de saber que *edad* realmente es la dirección de memoria donde se encuentra el dato.



Según esto podemos decir que una **variable** es una zona de memoria que se referencia mediante un identificador, conocido como nombre de la variable, donde se almacena un valor que puede cambiar durante la ejecución del programa.

Una variable tiene tres características básicas:

- **Nombre** o identificador de la variable.
- **Tipo**. Conjunto de valores que puede tomar la variable (int, double, char, etc.).
- **Valor**. Información que almacena.

Para poder utilizar una variable en un programa, primero tenemos que declararla.

Declarar una variable significa asignarle un nombre y un tipo.

Por ejemplo:

```
int a;      //declaramos la variable a de tipo int
char b;     //declaramos la variable b de tipo char
```

También podemos asignarle un valor inicial a la variable cuando se declara:

```
int x = 5;            //declaramos la variable x de tipo int y le asignamos el valor 5
char caracter = 'b'; //declaramos la variable caracter de tipo char y asignamos el valor 'b'
double altura = 6.25;//declaramos la variable altura tipo double y asignamos el valor 6.25
int contador = 0;     //declaramos la variable contador de tipo int y le asignamos el valor 0
```

Valor por defecto de las variables

A las variables que se declaran dentro de un método (por ejemplo, las variables que declaramos dentro del método main) java no les asigna un valor por defecto. Es nuestra responsabilidad asignarles valores iniciales válidos.

Las variables que son atributos de una clase sí toman valores iniciales por defecto.

Veremos todo esto con más detalle durante el curso.

Declaración de variables mediante var

A partir de Java 10 se incorporó al lenguaje la *inferencia de tipo para variables locales* (local variable type inference). Esto permite declarar variables sin necesidad de indicar el tipo. Java es capaz de deducir el tipo de la variable a partir del valor que se le asigna en el momento de declararla.

Se realiza mediante el identificador *var*.

Ejemplo:

```
var precio = 3.99;
```

En este ejemplo se declara una variable llamada *precio* y se le asigna el valor 3.99. Como 3.99 es un literal de tipo double, java asume que la variable *precio* es de tipo double.

Ejemplo:

```
var edad = 30;
```

Se declara una variable llamada *edad* y se le asigna el valor 30. Java asume que *edad* es de tipo int.

Las variables creadas mediante var funcionan igual que las creadas de forma tradicional. El uso de var no significa que las variables puedan cambiar de tipo durante la ejecución del programa. Una vez declarada una variable mediante var, el tipo asignado ya no se podrá cambiar.

Ejemplo:

```
var x = 1;
var x = "hola"; //error, x ya está declarada anteriormente
```

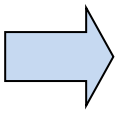
Debemos tener en cuenta una serie de restricciones a la hora de usar var:

- Solo se pueden declarar variables locales, es decir, variables que se encuentren dentro de un método.
- No se pueden declarar mediante var los parámetros de un método o el tipo de retorno de un método.
- Siempre se debe asignar un valor a la variable que se declara para poder inferir su tipo.
- A la variable no se le puede asignar el valor null.

El uso de var hace que el código sea más fácil de leer y escribir pero al mismo tiempo oculta información sobre los tipos de datos utilizados. Por lo tanto no hay un criterio a seguir para decidir cuando usarlo y cuando no.

10. CONSTANTES

Un programa puede contener ciertos valores que no deben cambiar durante su ejecución. Estos valores se llaman constantes.



Podemos decir que una constante es una zona de memoria que se referencia con un identificador, conocido como nombre de la constante, donde se almacena un valor que no puede cambiar durante la ejecución del programa.

Una constante se declara de forma similar a una variable, anteponiendo la palabra **final**

Ejemplo:

```
final double PI = 3.141591;
final int diasSemana = 7;
```

11. OPERADORES

OPERADORES JAVA						
ARITMÉTICOS	RELACIONALES	LÓGICOS	UNITARIOS	A NIVEL DE BITS	ASIGNACIÓN	CONDICIONAL
+	<	&&	+	&	=	? :
-	<=	||	-	|	+=	
*	>	!	++	^	-=	
/	>=		--	<<	*=	
%	!=		~	>>	/=	
	==		!	>>>	%=	
					<<=	
					>>=	
					>>>=	
					&=	
					|=	
					^=	

OPERADORES ARITMÉTICOS

+	Operador suma. Los operandos pueden ser enteros o reales
-	Operador Resta. Los operandos pueden ser enteros o reales
*	Operador Multiplicación. Los operandos pueden ser enteros o reales
/	Operador División. Los operandos pueden ser enteros o reales. -Si ambos operandos son enteros el resultado de la división es entero (sin decimales). -En cualquier otro caso el resultado de la división es real (se obtienen decimales).
%	Operador Resto de la división. Los números pueden ser enteros o reales.

El operador % con números reales actúa de la siguiente forma:

El resto de la división se obtiene calculando: `dividendo - divisor * parte entera del cociente`

Ejemplo:

Supongamos que tenemos las siguientes variables:

```
int a = 10, b = 3;
double v1 = 12.5, v2 = 2.0;
char c1='P', c2='T';
```

En la tabla se muestran distintas operaciones aritméticas que podemos hacer con ellas:

Operación	Valor	Operación	Valor	Operación	Valor
a+b	13	v1+v2	14.5	c1	80
a-b	7	v1-v2	10.5	c1 + c2	164
a*b	30	v1*v2	25.0	c1 + c2 + 5	169
a/b	3	v1/v2	6.25	c1 + c2 + '5'	217
a%b	1	v1%v2	0.5		

En las operaciones aritméticas donde intervienen variables de tipo char, el valor usado para realizar el cálculo es el valor numérico del carácter correspondiente según la tabla ASCII (o UNICODE en su caso).

Por eso, según vemos en el ejemplo: c1 + c2 = 164.

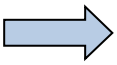
c1 = 'P' y el valor numérico de 'P' según la tabla ASCII es 80.

c2 = 'T' y el valor numérico de 'T' según la tabla ASCII es 84.

Por lo tanto cuando realizamos c1 + c2 el resultado es 80 + 84 = 164.

Como ya se ha comentado antes, Java es un lenguaje fuertemente tipado. Por eso **es importante saber de qué tipo es el resultado de una operación aritmética si intervienen datos de tipos distintos**.

En aquellas operaciones aritméticas en las que aparecen operandos de distinto tipo, java convierte los valores al tipo de dato de mayor precisión de todos los datos que intervienen y este será el tipo del resultado. Esta conversión se realiza de forma temporal, solamente para realizar la operación. Los tipos de datos originales permanecen igual después de la operación. Esto es muy importante tenerlo en cuenta para poder asignar el resultado de la operación a una variable del mismo tipo.

 Los tipos short, byte y char se convierten automáticamente a int.

Por ejemplo, dadas las siguientes variables:

```
int i = 7;
double f = 5.5;
char c = 'w';
byte b = 1;
long ln = 10;
```

En la tabla se puede ver el resultado que se obtiene al realizar operaciones esntre ellas y de qué tipo es este resultado (double, int, char, etc)

Operación	Valor	Tipo del resultado
i + f	12.5	double
i + c	126	int
i + c – '0'	78	int
(i + c) – (2 * f / 5)	123.8	double
b + c	120	int
ln + i	17	long
2.5 + ln	10.5	double

OPERADORES RELACIONALES

Los operadores relacionales comparan dos operandos y dan como resultado de la comparación verdadero ó falso.

Los operadores relacionales son:

<	Menor que
>	Mayor que
<=	Menor o igual
>=	Mayor o igual
!=	Distinto
==	Igual

Los **operandos** tienen que ser de **tipo primitivo**.

Por ejemplo:

```
int a = 7, b = 9, c = 7;
```

Operación	Resultado
a == b	false
a >= c	true
b < c	false
a != c	false

OPERADORES LÓGICOS

Los operadores lógicos se utilizan con operandos de tipo boolean. Se utilizan para construir expresiones lógicas, cuyo resultado es de tipo true o false.

Los operadores lógicos son:

&& Operador AND.
El resultado es verdadero si los dos operandos son verdaderos.
El resultado es falso en caso contrario.
Si el primer operando es falso no se evalúa el segundo, ya que el resultado será falso.

|| Operador OR.
El resultado es falso si los dos operandos son falsos.
Si uno es verdadero el resultado es verdadero.
Si el primer operando es verdadero no se evalúa el segundo.

! Operador NOT.
Se aplica sobre un solo operando.
Cambia el valor del operando de verdadero a falso y viceversa.

Las definiciones de las operaciones OR, AND y NOT se recogen en unas tablas conocidas como **tablas de verdad.**

A	B	A OR B
F	F	F
F	V	V
V	F	V
V	V	V

A	B	A AND B
F	F	F
F	V	F
V	F	F
V	V	V

A	NOT A
F	V
V	F

Como ejemplo, en la siguiente tabla vemos una serie de expresiones lógicas y su valor:

```
int i = 7;
float f = 5.5F;
char c = 'w';
```

Expresión	Resultado
`i >= 6 && c != 'w'`	`false`
`i >= 6 || c != 'w'`	`true`
`f < 10 && i > 100`	`false`
`!(c != 'p') || i % 2 == 0`	`false`
`i + f <= 10`	`false`
`i >= 6 && c == 'w' && f == 5`	`false`
`c != 'p' || i + f <= 10`	`true`

Las expresiones lógicas se evalúan sólo hasta que se ha establecido el valor cierto o falso del conjunto. Cuando, por ejemplo, una expresión va a ser seguro falsa por el valor que ha tomado uno de sus operandos, java ya no evalúa el resto de expresión.

OPERADORES UNITARIOS.

– +	Operadores signo negativo y positivo
++ ––	Operadores incremento y decremento
~	Operador complemento a 1
!	Operador NOT. Negación

Estos operadores **afectan a un solo operando**.

El **operador ++** (incremento) incrementa en 1 el valor de la variable.

Ejemplo:

```
int i = 1;

i++; // Esta instrucción incrementa en 1 la variable i.
     // Es lo mismo que hacer i = i + 1; i toma el valor 2
```

El **operador ––** (decremento) decrementa en 1 el valor de la variable.

Ejemplo:

```
int i = 1;

i--; // decrementa en 1 la variable i.
     // Es lo mismo que hacer i = i - 1; i toma el valor 0
```

Los operadores incremento y decremento pueden utilizarse como prefijo o sufijo, es decir, pueden aparecer antes o después de la variable.

Por ejemplo:

```
i = 5;
i++; // i vale ahora 6
++i; // i vale ahora 7
```

Cuando estos operadores intervienen en una expresión, si preceden al operando (++i), el valor se modificará antes de que se evalúe la expresión a la que pertenece.

En cambio, si el operador sigue al operando (i++), entonces el valor del operando se modificará después de evaluar la expresión a la que pertenece.

Por ejemplo:

```
int x, i = 3;
x = i++;
```

En esta asignación a x se le asigna el valor de i (3) y a continuación i se incrementa, por lo tanto, después de ejecutarla:

x contiene 3, i contiene 4.

Si las instrucciones son:

```
int x, i = 3;
x = ++i;
```

En esta instrucción primero se incrementa el valor de i y el resultado se asigna a x. Por lo tanto, después de ejecutarla:

x contiene 4, i contiene 4.

Otro ejemplo:

```
int i = 1;

System.out.println(i);

System.out.println (++i);

System.out.println (i);
```

Estas instrucciones muestran por pantalla:

```
1
2
2
```

En cambio, si se cambia la posición del operador:

```
int i = 1;
System.out.println(i);
System.out.println (i++);
System.out.println (i);
```

Estas instrucciones mostrarían por pantalla:

```
1
1
2
```

El operador complemento a 1 ~ cambia de valor todos los bits del operando (cambia unos por ceros y ceros por unos). Solo puede usarse con datos de tipo entero. El carácter ~ es el ASCII 126.

Por ejemplo:

```
int a = 1, b;
b = ~a;
```

OPERADORES A NIVEL DE BITS

Realizan la manipulación de los bits de los datos con los que operan.

Los datos deben ser de tipo entero.

Los operadores a nivel de bits son:

&	and a nivel de bits
|	or a nivel de bits
^	xor a nivel de bits
<<	desplazamiento a la izquierda, rellenando con ceros a la derecha
>>	desplazamiento a la derecha, rellenando con el bit de signo por la izquierda
>>>	desplazamiento a la derecha rellenando con ceros por la izquierda

OPERADORES DE ASIGNACIÓN.

Se utilizan para asignar el valor de una expresión a una variable.

Los operandos deben ser de tipo primitivo.

=	Asignación
+=	Suma y asignación
– =	Resta y asignación
*=	Producto y asignación
/=	División y asignación
%=	Resto de la división entera y asignación
<<=	Desplazamiento a la izquierda y asignación
>>=	Desplazamiento a la derecha y asignación
>>>=	Desplazamiento a la derecha y asignación rellenando con ceros
&=	and sobre bits y asignación
|=	or sobre bits y asignación
^=	xor sobre bits y asignación

Ejemplos de asignaciones:

```
x += 3   -> es lo mismo que escribir x = x + 3;
y *= 3   -> es lo mismo que escribir y = y * 3;
```

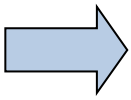
En general:

variable **op=** expresión equivale a: variable = variable **op** expresión

En la siguiente tabla vemos más ejemplos de asignaciones:

```
int i = 5, j = 7, x = 2, y = 2, z = 2;
float f = 5.5F, g = -3.25F;
```

Expresión	Expresión equivalente	Valor final
`i += 5`	`i = i + 5`	10
`f -= g`	`f = f - g`	8.75
`j *= i - 3`	`j = j * (i - 3)`	14
`f /= 3`	`f = f / 3`	1.833333
`i %= (j - 2)`	`i = i % (j - 2)`	0
`x *= -2 * (y + z) / 3`	`x = x * (-2 * (y + z) / 3)`	-4



Importante: Si los dos operandos de una expresión de asignación (el de la izquierda y el de la derecha) son de distinto tipo es posible que la asignación no se pueda realizar. **El valor de la expresión de la derecha se asignará a la variable que aparece a la izquierda del operador de asignación siempre que sean del mismo tipo o de tipos *compatibles*.**

Por ejemplo:

```
int n;
double m = 2.5;
n = m + 1; //Al intentar realizar esta asignación se producirá un error.
           //el resultado de la operación m + 1 es de tipo double y no se puede
           //asignar a una variable de tipo int
```

En este otro caso la asignación sí es posible:

```
double m;
int n = 2;
m = n+50; //El resultado de la operación n + 50 es de tipo int y m es de tipo double
          //aunque ambos tipos no coinciden, la asignación se puede realizar porque
          //el rango de un int es menor que el de un double y por lo tanto no
          //hay pérdida de información
```

En general, en una asignación del tipo A = B, si el rango de valores que puede almacenar A es mayor que el rango de valores que puede almacenar B (puedes consultar el rango de cada tipo en la tabla que aparece en el punto 7 del tema) entonces la asignación se podrá realizar. Por ejemplo, si A es de tipo double y B de tipo float la asignación A = B se puede hacer pero no se podrá realizar en el caso contrario (siendo A float y B double).

Por último, indicar que en Java están permitidas las **asignaciones múltiples**.

Ejemplo: a = b = c = 3; equivale a a = 3; b = 3; c = 3;

La asignación del valor a las variables en una asignación múltiple se realiza de derecha a izquierda. En el ejemplo, primero se asigna el valor 3 a c, a continuación el valor de c se asigna a b y finalmente el valor de b se le asigna a la variable a.

OPERADOR CONDICIONAL

Se puede utilizar en sustitución de la sentencia de control if-else, pero hace las instrucciones menos claras.

Los forman los caracteres **?** y **:**

Se utiliza de la forma siguiente:

expresión1 **?** expresión2 **:** expresión3

Si expresión1 es cierta entonces se evalúa expresión2 y éste será el valor de la expresión condicional. Si expresión1 es falsa, se evalúa expresión3 y éste será el valor de la expresión condicional.

Por ejemplo:

```
int i = 10, j;
j = (i < 0) ? 0 : 100;
```

Esta expresión asigna a j el valor 100. Su significado es: si el valor de i es menor que 0 asigna a j el valor 0, sino asigna a j el valor 100. Como i vale 10, a j se le asigna 100.

La instrucción anterior es equivalente a escribir:

```
if(i < 0)
   j = 0;
else
   j = 100;
```

Más Ejemplos:

```
int a = 1, b = 2, c = 3;
c += (a > 0 && a <= 10) ? ++a : a / b;    // c toma el valor 5
int a = 50, b = 10, c = 20;
c + =(a > 0 && a <= 10) ? ++a : a / b;    // c toma el valor 25
```

PRIORIDAD Y ORDEN DE EVALUACIÓN DE LOS OPERADORES

La siguiente tabla muestra todos los operadores de Java ordenados de mayor a menor prioridad. La primera línea de la tabla contiene los operadores de mayor prioridad y la última los de menor prioridad. Los operadores que aparecen en la misma línea tienen la misma prioridad.
Una expresión entre paréntesis siempre se evalúa primero y si están anidados se evalúan de más internos a más externos.

Operador	Asociatividad
`() [] .`	Izquierda a derecha
`++ -- ~ !`	Derecha a izquierda
`new`	Derecha a izquierda
`* / %`	Izquierda a derecha
`+ -`	Izquierda a derecha
`>> >>> <<`	Izquierda a derecha
`> >= < <=`	Izquierda a derecha
`== !=`	Izquierda a derecha
`&`	Izquierda a derecha
`^`	Izquierda a derecha
`|`	Izquierda a derecha
`&&`	Izquierda a derecha
`||`	Izquierda a derecha
`?:`	Derecha a izquierda
`= += -= *= ...`	Derecha a izquierda