Elementos de un programa informático

Identificadores

Los identificadores son los nombres que asigna el programador a diferentes elementos del programa como las constantes, variables, métodos, clases, paquetes, ...

- Deben contener al menos un carácter
- Los caracteres válidos son letras (a-Z), dígitos (0-9), guión bajo (_) o dólar ($)

  double ***cant1***=0;

  ~~double ***mi%s***=5;~~
- El primer caracter no podrá ser un dígito

  ~~double ***1cant***=0;~~

Identificadores

Existen una serie de convenciones respecto a la nomenclatura a utilizar en los identificadores.

- No utilizar espacios entre palabras
  double ***mi numero***=1;
- Paquetes: Utilizar lowercase. La jerarquía de paquetes se representará con un “.”
- Clases e interfaces: UpperCamelCase
  class ***MiNuevaClase***
- Variables y métodos: lowerCamelCase
  public void ***calcularAreaTriangulo***(int base, int altura);
- Constantes: SCREAMING_SNAKE_CASE
  const ***VALOR_DE_PI***=3.1416

Identificadores

Existen una serie de convenciones respecto a la nomenclatura a utilizar en los identificadores.

- No utilizar espacios entre palabras
  double ***mi numero***=1;
- Paquetes: Utilizar lowercase. La jerarquía de paquetes se representará con un “.”
- Clases e interfaces: UpperCamelCase
  class ***MiNuevaClase***
- Variables y métodos: lowerCamelCase
  public void ***calcularAreaTriangulo***(int base, int altura);
- Constantes: SCREAMING_SNAKE_CASE
  const ***VALOR_DE_PI***=3.1416

Palabras reservadas

Son un conjunto de identificadores que tienen un significado propio para el compilador y por tanto no pueden ser utilizados libremente dentro del programa.

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

No podemos, por ejemplo, crear una variable con el nombre ***package***

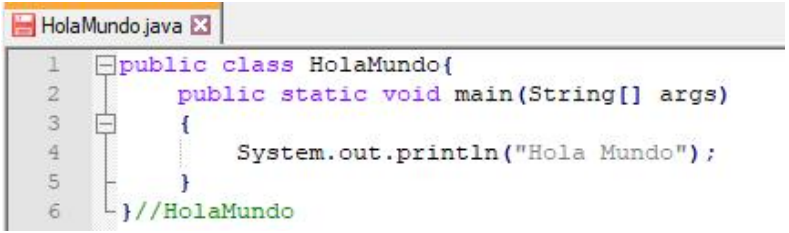
~~double ***package***=6;~~

Clases

Método Main

Es el punto de entrada del programa por el cual comienza la ejecución del mismo.

Debe declararse en alguna clase del programa y su formato debe ser siempre el mismo (ha de ser publico, estático y con la especificación de los parámetros de entrada). Además, el identificador de la clase en la que se declare debe coincidir con el nombre del archivo .java



Paquetes

Los paquetes sirven para conseguir modularidad y encapsulación agrupando clases o interfaces es una estructura de carpetas que puede seguir una estructura jerárquica.

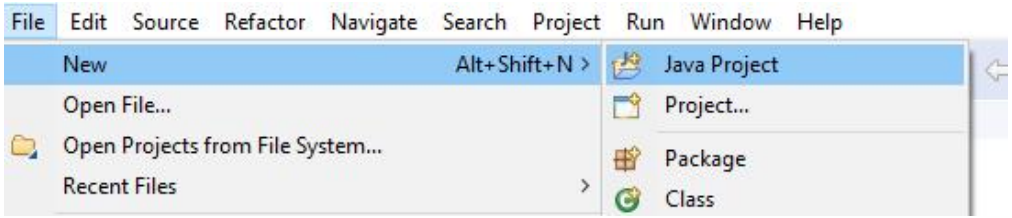
La definición de un paquete se realiza con la palabra reservada ***package***

Para acceder a los contenidos de un paquete desde una clase:

- Si la clase se encuentra dentro del mismo paquete basta con indicar el nombre de la clase.
- Si están en diferentes paquetes habrá que importar el paquete mediante la palabra reservada ***import*** o bien anteponer el nombre del paquete a la clase.

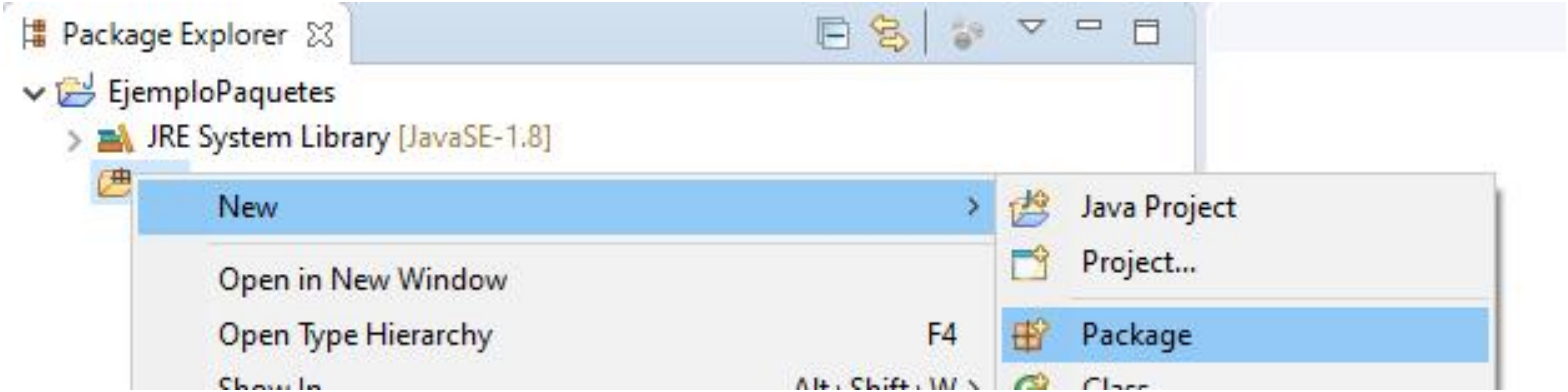
Ejemplo paquetes

Crear un nuevo proyecto de Java con el nombre ***EjemploPaquetes***



Ejemplo paquetes

Crear tres paquetes con los nombres “a”, “b” y “b.c”

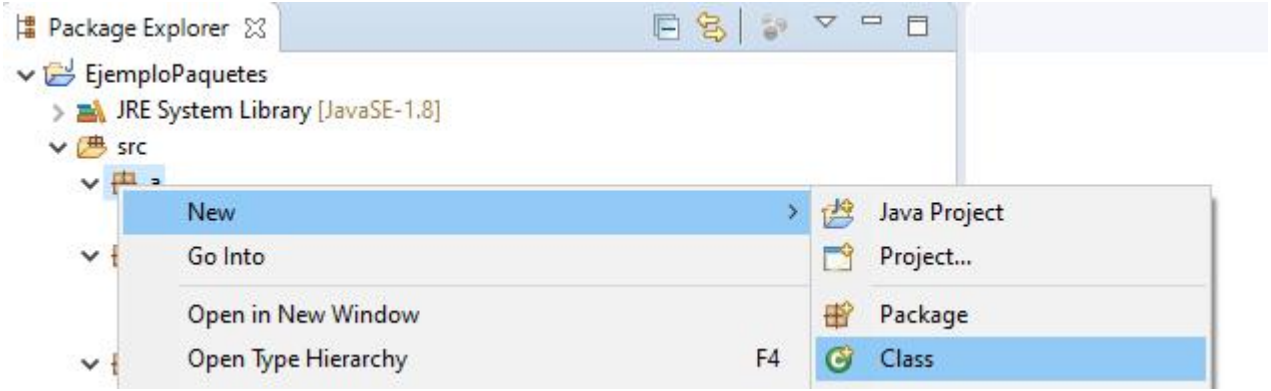


Ejemplo paquetes

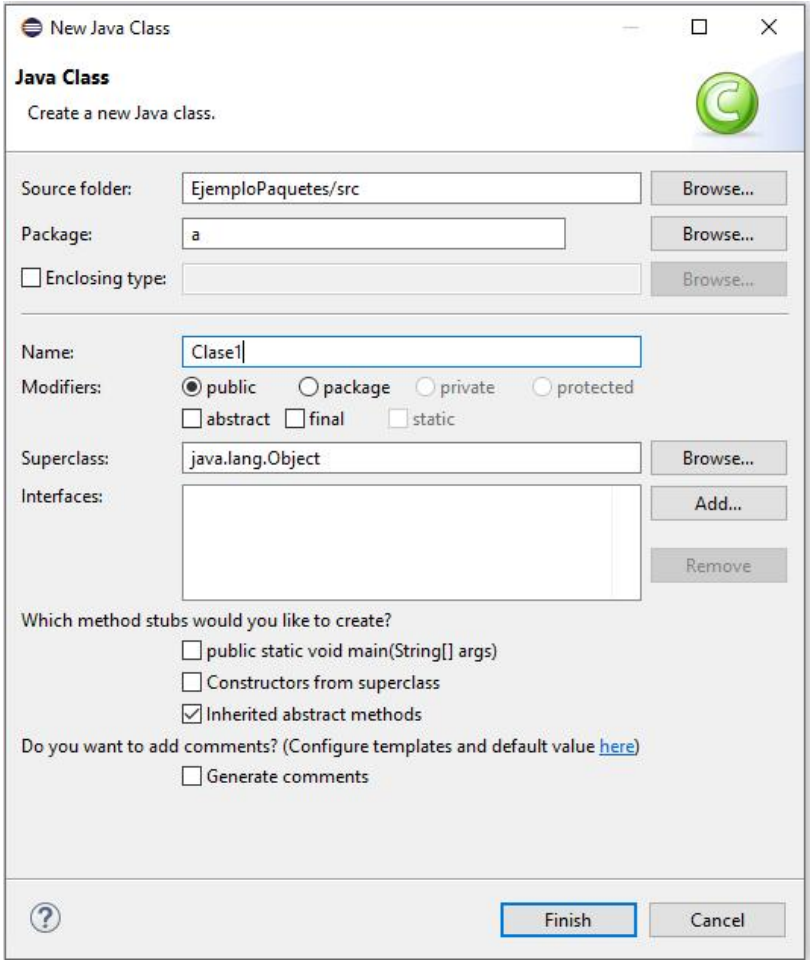
Crear tres paquetes con los nombres “a”, “b” y “b.c”

Ejemplo paquetes

Creamos clases dentro de los paquetes de la siguiente forma (segundo botón del ratón encima del paquete donde se creará la clase)

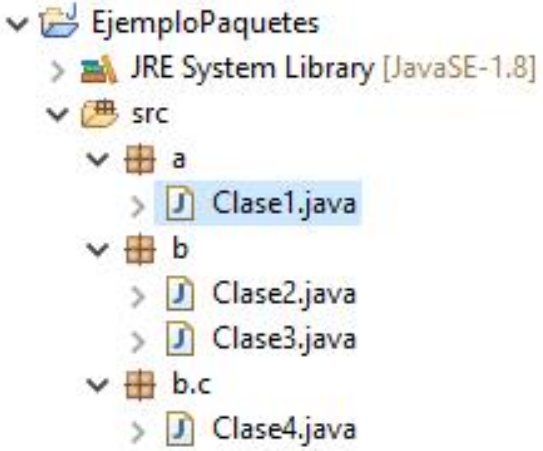


Ejemplo paquetes



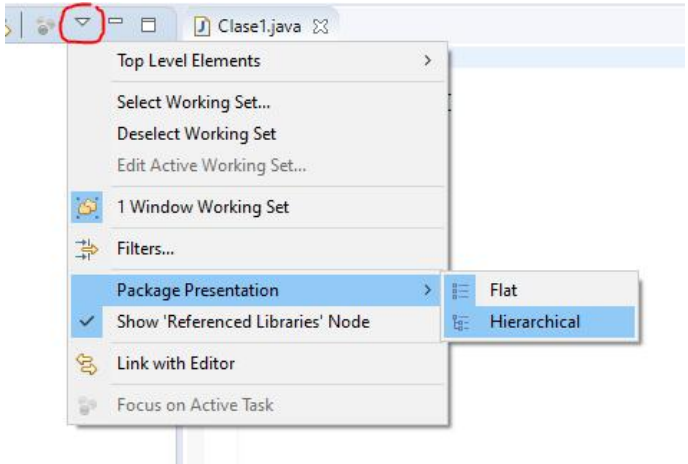
Ejemplo paquetes

Crearemos clases en cada paquete para que nos quede de esta forma



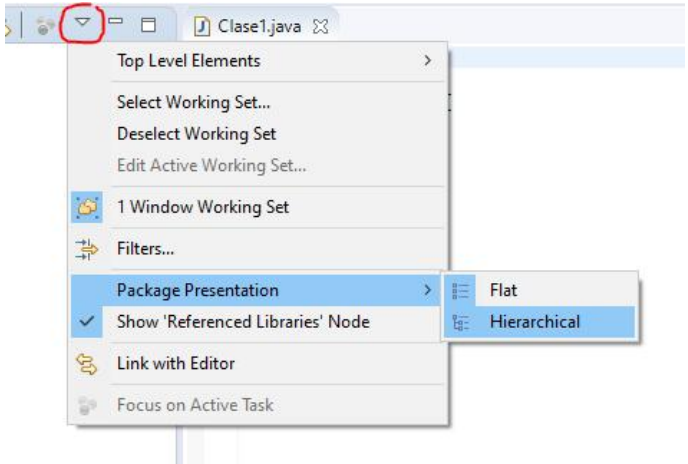
Ejemplo paquetes

Una vez que los paquetes tienen elementos (las clases) podemos ver la estructura jerárquica

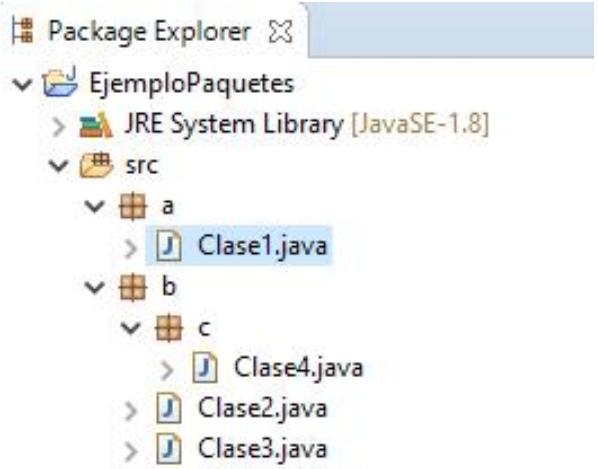


Ejemplo paquetes

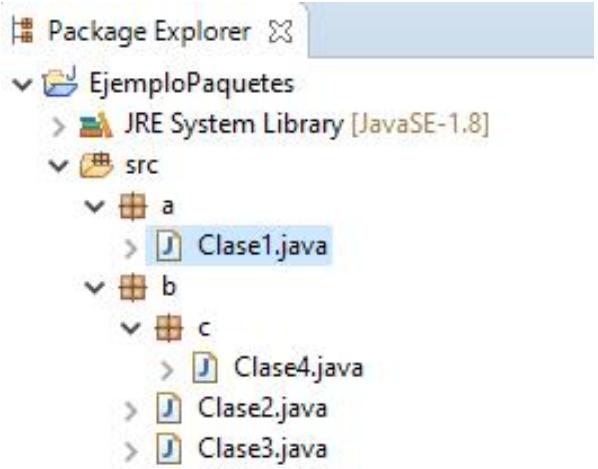
Una vez que los paquetes tienen elementos (las clases) podemos ver la estructura jerárquica



Ejemplo paquetes



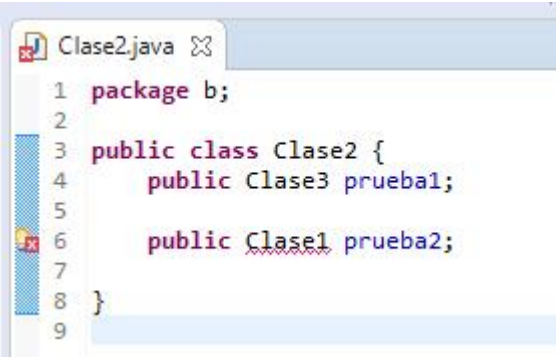
Ejemplo paquetes



Ejemplo paquetes

Accesibilidad entre paquetes:

Elementos dentro del mismo paquete son directamente accesibles.



Ejemplo paquetes

Accesibilidad entre paquetes:

Para hacer uso de la clase de otro paquete podemos realizar un import o indicar el paquete en el que se encuentra.

```
package b;
import a.*;

public class Clase2 {
    public Clase3 prueba1;

    public Clase1 prueba2;

}
```

Realizando ***import a.*;*** cargamos en memoria todas las clases del paquete a para su uso dentro de este fichero (Clase2.java)

Ejemplo paquetes

Accesibilidad entre paquetes:

```
1  package b;
2  import a.Clase1;
3
4  public class Clase2 {
5      public Clase3 prueba1;
6
7      public Clase1 prueba2;
8
9  }
```

Si sólo necesitamos cargar una clase podemos realizar un import sólo de la clase que vamos a usar.

Ejemplo paquetes

Accesibilidad entre paquetes:

```
1  package b;
2
3  public class Clase2 {
4      public Clase3 prueba1;
5
6      public a.Clase1 prueba2;
7
8  }
9
```

Otra forma de acceder a la clase deseada es indicar la ruta poniendo el nombre del paquete antes del nombre de la clase.

Variables

Permiten almacenar en la memoria diferentes valores a lo largo de la vida del programa.

En java pueden distinguirse las de tipo primitivo (representaciones de números, booleanos,caracteres, ...) y las de tipo objeto (String, Integer, objetos instanciados de Clases definidas por el usuario,...).

La declaración (crear la variable) y su inicialización (darle un valor inicial) puede realizarse o bien en la misma instrucción o en instrucciones diferentes.

Sintaxis (los parámetros entre corchetes son opcionales)

[Visibilidad] tipo nombreVariable1;

nombreVariable1=valor1;

[Visibilidad] tipo nombreVariable2= valor;

Variables

Ejemplo:

```
public static void main(String args) {
    int a;
    a=2;
    int b=8;
}
```

Variables

Ámbito de variables:

Una variable puede ser accesible dentro de una sección de código dependiendo de cómo y dónde se declare. Esto se conoce como ámbito, alcance o scope de la variable. A considerar:

- El código entre llaves “{}” se encuentra en el mismo ámbito
- Una variable declarada en un ámbito puede ser accedida desde ese ámbito o desde subámbitos contenidos dentro de este.
- Una vez se sale del ámbito en el que fue declarada la variable, esta se destruye.

Variables

```
public class Ambitos {

    public static void main(String[] args) {
        {
            {
                int a =2;
                System.out.println(a);

            }

            System.out.println(a);

            int a=1;
            int b=1000;

            System.out.println(a);

            int b=2;
        }

        System.out.println(b);

        {
            int a=2;
            {
                System.out.println(a+1);
                {
                    System.out.println(a+2);
                }
            }
        }

    }//main

}//Ambitos
```

Constantes

Una constante es un elemento que mantiene un valor inalterable a lo largo de la vida del programa.

Se declaran de forma similar a una variable pero con las palabras reservadas ***static final***

[Visibilidad] static final tipo NOMBRE_CONSTANTE=valor;

```
public static final double VALOR_PI=3.1416;
```

Tipos de datos primitivos

VARIABLES DE TIPOS PRIMITIVOS.					
Nombre	**Tipo**	**Tamaño**	**Valor por defecto**	**Forma de inicializar**	**Rango**
Boolean	Lógico	1 bit	False	Boolean a=true	True-false
Char	Carácter	16 bits	Null	Char a='Z'	Unicode
Byte	Numero entero	8 bits	0	Byte a =0	-128 a 127
Short	Numero entero	16 bits	0	Short a =12	-32.768 a 32.767
Int	Numero entero	32 bit	0	Int a= 1250	-2.147.483.648 a 2.147.483.649
Long	Numero entero	64 bits	0	Long a= 125000	-9*10^18 a 9*10^18
Float	Numero real	32 bits	0	Float a =3.1	-3,4*10^38 a 3,4*10^38
Double	Numero real	64 bits	0	Double a = 125.2333	-1,79*10^308 a 1,79*10^308

Conversiones de Tipo

Existe la posibilidad de guardar información de un tipo en una variable definida con un tipo diferente, aunque no siempre es posible. A esta acción se le denomina ***conversión***, ***cast*** o ***tipado***.

Al realizar la conversión se debe evitar la pérdida de información (por ejemplo convirtiendo un valor real 3.14 a entero 3 perdemos la parte decimal).

Como norma general las conversiones seguras se realizan de los tipos que almacenan menos datos a los que almacenan más datos.

Conversiones de Tipo

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

Conversiones de tipo

Implícitas

Se realizan de forma automática con operaciones de asignación.

```
byte b=1;
short s=b;
int i=s;
long l=i;
float f=l;
double d =f;
```

Conversiones de tipo

Explícitas

Fuerzan la conversión anteponiendo el tipo de valor destino entre paréntesis. Con ellas se pueden realizar conversiones que no son seguras por la pérdida de información.

```
int i=127;
byte b=i;

byte c=(byte)i;

byte d=(byte) 128;
System.out.println(d);

double j=1.3698;
int z=(int)j;
```

Conversiones de tipo

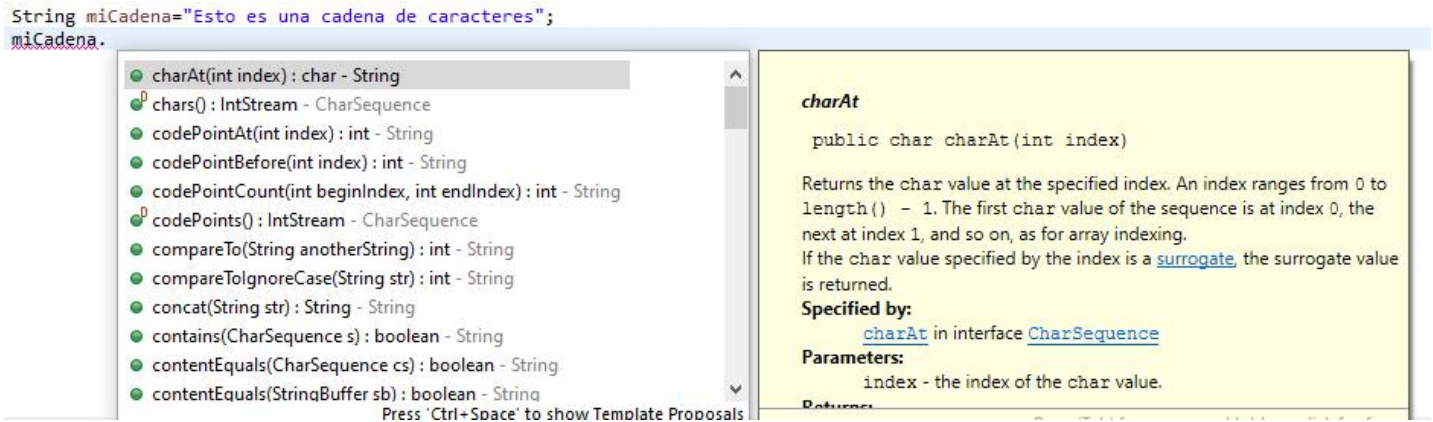
Explícitas

```
int i=127;
byte b=i;        //No es posible realizar una conversión simple

byte c=(byte)i;   //Tanto i como c tienen el valor 127

byte d=(byte) 128;
System.out.println(d);  //d tiene el valor -128

double j=1.3698;
int z=(int)j;        //z tiene el valor 1
```

String

El tipo primitivo ***char*** almacena únicamente un caracter con lo que su uso resulta muy limitado.

Para trabajar con cadenas de caracteres (“Hola”) se utilizan objetos de la clase String, que además provee de una serie de métodos para la manipulación de cadenas.



Clases envoltorio

Las clases envoltorio son clases especiales cuyo cometido es crear objetos que contengan un tipo de dato primitivo y proveer métodos que faciliten su manejo.

Primitivo	boolean	byte	char	double	float	int	long	short
Wrapper	Boolean	Byte	Character	Double	Float	Integer	Long	Short

```
Integer miInteger=5;
miInteger.toString();   //Devuelve "5"
```

Secuencias de escape

En java existen una serie de secuencias especiales para utilizar con Strings y char:

\b	Retrocede un espacio	\r	Retorno de carro
\t	Tabulación	\"	Comilla doble
\n	Nueva línea	\'	Comilla simple
\f	Salto de página	\\	Barra invertida

Comentarios

Es posible, incluso necesario, realizar comentarios sobre el código.

Estos comentarios son ignorados por el compilador.

Existen tres tipos:

- De una línea: Comienzan por //
- De varias líneas: Comienzan por /* y terminan por */
- Documentación: Tipo especial de varias líneas. Comienzan por /** y terminan con */. El programa javadoc utiliza estos comentarios para generar automáticamente documentación del código.

Operadores y Expresiones

Binarios

Aritméticos		Relacionales		Lógicos	
+	Suma en números y concatenación en Strings	==	Igual	&&	y lógico
-	Resta	<	Menor	||	o lógico
*	Multiplicación	<=	Menor o igual	Especiales	
/	División real	>	Mayor	=	Asignación
%	Resto o módulo	>=	Mayor o igual	instanceof	Objeto es de tipo
		!=	Distinto		

Operadores y Expresiones

Unarios

+	Indica un valor positivo	++	Incrementa en 1	!	Negación lógica
-	Niega la expresión	--	Decrementa en 1		

Operadores Unarios

Operador de incremento (++): Aumenta el valor de una variable en 1.

- Pre-incremento (++variable): Incrementa la variable antes de usarla en una expresión
- Post-incremento (variable++): Usa la variable primero y luego la incrementaPost-incremento (variable++): Usa la variable primero y luego la incrementa

Operador de decremento (--): Disminuye el valor de una variable en 1 (existe el uso pre-decremental y post-decremental)

Negación numérica (-): Cambia el signo de un número, es decir, convierte un número positivo en negativo o viceversa.

Operadores Unarios (2)

Operador unario positivo (+): No cambia el valor del operando; simplemente reafirma su positividad.

+= (Suma y asignación): Este operador suma el valor del operando a la derecha al operando a la izquierda, y luego asigna el resultado a la variable de la izquierda.

Existen equivalentes con otras operaciones aritméticas: *= -= /= %=