Los elementos del lenguaje Java

La sintáxis de un lenguaje define los elementos de dicho lenguaje y cómo se combinan para formar un programa. Los elementos típicos de cualquier lenguaje son los siguientes:

- Identificadores: los nombres que se dan a las variables
- Tipos de datos
- Palabras reservadas: las palabras que utiliza el propio lenguaje
- Sentencias
- Bloques de código
- Comentarios
- Expresiones
- Operadores

A lo largo de las páginas que siguen examinaremos en detalle cada uno de estos elementos.

1. Identificadores

Un identificador es un nombre que identifica a una variable, a un método o función miembro, a una clase. Todos los lenguajes tienen ciertas reglas para componer los identificadores:

- Todos los identificadores han de comenzar con una letra, el carácter subrayado (_) o el carácter dolar ($).
- Puede incluir, pero no comenzar por un número.
- No puede incluir el carácter espacio en blanco.
- Distingue entre letras mayúsculas y minúsculas.
- No se pueden utilizar las palabras reservadas como identificadores.

Además de estas restricciones, hay ciertas convenciones que hacen que el programa sea más legible, pero que no afectan a la ejecución del programa. La primera y fundamental es la de encontrar un nombre que sea significativo, de modo que el programa sea lo más legible posible. El tiempo que se pretende ahorrar eligiendo nombres cortos y poco significativos se pierde con creces cuando se revisa el programa después de cierto tiempo.

Tipo de identificador	**Convención**	**Ejemplo**
nombre de una clase	Comienza por letra mayúscula	String, Rectangulo, CinematicaApplet
nombre de función	comienza con letra minúscula	calcularArea, getValue, setColor
nombre de variable	comienza por letra minúscula	area, color, appletSize
nombre de constante	En letras mayúsculas	PI, MAX_ANCHO

2. Comentarios

Un comentario es un texto adicional que se añade al código para explicar su funcionalidad, bien a otras personas que lean el programa, o al propio autor como recordatorio. Los comentarios son una parte importante de la documentación de un programa. Los comentarios son ignorados por el compilador, por lo que no incrementan el tamaño del archivo ejecutable; se pueden, por tanto, añadir libremente al código para que pueda entenderse mejor.

La programación orientada a objetos facilita mucho la lectura del código, por lo que lo que no se precisa hacer tanto uso de los comentarios como en los lenguajes estructurados. En Java existen tres tipos de comentarios

- Comentarios en una sola línea
- Comentarios de varias líneas
- Comentarios de documentación

Como podemos observar un comentario en varias líneas es un bloque de texto situado entre el símbolo de comienzo del bloque **/***, y otro de terminación del mismo ***/**. Teniendo encuentra este hecho, los programadores diseñan comentarios como el siguiente:

```
 /*--------------------------------|
|   (C) Blanca Calderón         |
|   fecha: Octubre 2023              |
|   programa: PrimeroApp.java       |
|--------------------------------*/
```

Los comentarios de documentación es un bloque de texto situado entre el símbolo de comienzo del bloque **/****, y otro de terminación del mismo ***/.** El programa *javadoc* utiliza estos comentarios para generar la documentación del código.

```
/** Este es el primer programa de una
serie dedicada a explicar los fundamentos del lenguaje Java */
```

Habitualmente, usaremos comentarios en una sola línea //, ya que no tiene el inconveniente de aprendernos los símbolos de comienzo y terminación del bloque, u olvidarnos de poner este último, dando lugar a un error en el momento de la compilación. En la ventana de edición del Entorno Integrado de Desarrollo (IDE) los comentarios se distinguen del resto del código por el color del texto.

```
public class PrimeroApp{
    public static void main(String[] args) {
//imprime un mensaje
        System.out.println("El primer programa");
    }
}
```

Un procedimiento elemental de depuración de un programa consiste en anular ciertas sentencias de un programa mediante los delimitadores de comentarios. Por ejemplo, se puede modificar el programa y anular la sentencia que imprime el mensaje, poniendo delante de ella el delimitador de comentarios en una sola línea.

```
        //System.out.println("El primer programa");
```

Al correr el programa, observaremos que no imprime nada en la pantalla.

La sentencia *System.out.println()* imprime un mensaje en la consola. La función *println* tiene un sólo argumento una cadena de caracteres u objeto de la clase *String*.

3. Sentencias

Una sentencia es una orden que se le da al programa para realizar una tarea específica, esta puede ser: mostrar un mensaje en la pantalla, declarar una variable (para reservar espacio en memoria), inicializarla, llamar a una función, etc. Las sentencias acaban con **;**. este carácter separa una sentencia de la siguiente. Normalmente, las sentencias se ponen unas debajo de otras, aunque sentencias cortas pueden colocarse en una misma línea. He aquí algunos ejemplos de sentencias

```
int i=1;
import java.awt.*;
System.out.println("El primer programa");
rect.mover(10, 20);
```

En el lenguaje Java, los caracteres espacio en blanco se pueden emplear libremente. Como podremos ver en los sucesivos ejemplos, es muy importante para la legibilidad de un programa la colocación de unas líneas debajo de otras empleando tabuladores. El editor del IDE nos ayudará plenamente en esta tarea sin apenas percibirlo.

4. Bloques de código

Un bloque de código es un grupo de sentencias que se comportan como una unidad. Un bloque de código está limitado por las llaves de apertura **{** y cierre **}.** Como ejemplos de bloques de código tenemos la definición de una clase, la definición de una función miembro, una sentencia iterativa **for**, los bloques **try ... catch**, para el tratamiento de las excepciones, etc.

5. Expresiones

Una expresión es todo aquello que se puede poner a la derecha del operador asignación =. Por ejemplo:

```
x=123;
y=(x+100)/4;
area=circulo.calcularArea(2.5);
```

```
Rectangulo r=new Rectangulo(10, 10, 200, 300);
```

La primera expresión asigna un valor a la variable *x*.

La segunda, realiza una operación

La tercera, es una llamada a una función miembro *calcularArea* desde un objeto *circulo* de una clase determinada

La cuarta, reserva espacio en memoria para un objeto de la clase *Rectangulo* mediante la llamada a una función especial denominada constructor.

6. Variables

Una variable es un nombre que se asocia con una porción de la memoria del ordenador, en la que se guarda el valor asignado a dicha variable. Hay varios tipos de variables que requieren distintas cantidades de memoria para guardar datos.

Todas las variables han de declararse antes de usarlas, la declaración consiste en una sentencia en la que figura el tipo de dato y el nombre que asignamos a la variable. Una vez declarada se le podrá asignar valores.

Java tiene tres tipos de variables:

- de instancia
- de clase
- locales

Las variables de instancia o miembros dato como veremos más adelante, se usan para guardar los atributos de un objeto particular.

Las variables de clase o miembros dato estáticos son similares a las variables de instancia, con la excepción de que los valores que guardan son los mismos para todos los objetos de una determinada clase. En el siguiente ejemplo, *PI* es una variable de clase y *radio* es una variable de instancia. *PI* guarda el mismo valor para todos los objetos de la clase *Circulo*, pero el radio de cada círculo puede ser diferente

```
class Circulo{
        static final double PI=3.1416;
        double radio;
//...
}
```

Las variables locales se utilizan dentro de las funciones miembro o métodos. En el siguiente ejemplo *area* es una variable local a la función *calcularArea* en la que se guarda el valor del área de un objeto de la clase *Circulo*. Una variable local existe desde el momento de su definición hasta el final del bloque en el que se encuentra.

```
class Circulo{
//...
        double calcularArea(){
                double area=PI*radio*radio;
                return area;
        }
}
```

En el lenguaje Java, las variables locales se declaran en el momento en el que son necesarias. Es una buena costumbre inicializar las variables en el momento en el que son declaradas. Veamos algunos ejemplos de declaración de algunas variables

```
int x=0;
String nombre="Angel";
double a=3.5, b=0.0, c=-2.4;
boolean bNuevo=true;
int[] datos;
```

Delante del nombre de cada variable se ha de especificar el tipo de variable que hemos destacado en letra negrita. Las variables pueden ser

- Un tipo de dato primitivo
- El nombre de una clase
- Un array

El lenguaje Java utiliza el conjunto de caracteres Unicode, que incluye no solamente el conjunto ASCII sino también caracteres específicos de la mayoría de los alfabetos. Así, podemos declarar una variable que contenga la letra ñ

```
int año=2023;
```

Se ha de poner nombres significativos a las variables, generalmente formados por varias palabras combinadas, la primera empieza por minúscula, pero las que le siguen llevan la letra inicial en mayúsculas. Se debe evitar en todos los casos nombres de variables cortos como *xx*, *i*, etc.

```
double radioCirculo=3.2;
```

Las variables son uno de los elementos básicos de un programa, y se deben

- Declarar
- Inicializar

- Usar

7. Tipos de datos primitivos

Tipo	Descripcion
boolean	Tiene dos valores **true** o **false**.
char	Caracteres Unicode de 16 bits. Los caracteres alfa-numéricos son los mismos que los ASCII con el bit alto puesto a 0. El intervalo de valores va desde 0 hasta 65535 (valores de 16-bits sin signo).
byte	Tamaño 8 bits. El intervalo de valores va desde -2^7 hasta 2^7 -1 (-128 a 127)
short	Tamaño 16 bits. El intervalo de valores va desde -2^{15} hasta 2^{15}-1 (-32768 a 32767)
int	Tamaño 32 bits. El intervalo de valores va desde -2^{31} hasta 2^{31}-1 (-2147483648 a 2147483647)
long	Tamaño 64 bits. El intervalo de valores va desde -2^{63} hasta 2^{63}-1 (-9223372036854775808 a 9223372036854775807)
float	Tamaño 32 bits. Números en coma flotante de simple precisión. Estándar IEEE 754-1985 (de 1.40239846e–45f a 3.40282347e+38f)
double	Tamaño 64 bits. Números en coma flotante de doble precisión. Estándar IEEE 754-1985. (de 4.9406565841246544e–324d a 1.7976931348623157e+308d.)

Los tipos básicos que utilizaremos en la mayor parte de los programas serán **boolean**, **int** y **double**.

Caracteres

En Java los caracteres no están restringidos a los ASCII sino son Unicode. Un carácter está siempre rodeado de comillas simples como 'A', '9', 'ñ', etc. El tipo de dato **char** sirve para guardar estos caracteres.

Un tipo especial de carácter es la secuencia de escape, similares a las del lenguaje C/C++, que se utilizan para representar caracteres de control o caracteres que no se imprimen. Una secuencia de escape está formada por la barra invertida (\) y un carácter. En la siguiente tabla se dan las secuencias de escape más utilizadas.

Carácter	Secuencia de escape
retorno de carro	\r
tabulador horizontal	\t
nueva línea	\n
barra invertida	\\

Variables booleanas

En el lenguaje C/C++ el valor 0 se toma como falso y el 1 como verdadero. En el lenguaje Java existe el tipo de dato **boolean**. Una variable booleana solamente puede guardar uno de los dos posibles valores: true (verdadero) y false (falso).

```
boolean encontrado=false;
{...}
encontrado=true;
```

Variables enteras

Una variable entera consiste en cualquier combinación de cifras precedidos por el signo más (opcional), para los positivos, o el signo menos, para los negativos. Son ejemplos de números enteros:

12, -36, 0, 4687, -3598

Como ejemplos de declaración de variable enteras tenemos:

```
int numero=1205;
int x,y;
long m=30L;
```

int es la palabra reservada para declarar una variable entera. En el primer caso, el compilador reserva una porción de 32 bits de memoria en el que guarda el número 1205. Se accede a dicha porción de memoria mediante el nombre de la variable, *num*ero. En el segundo caso, las porciones de memoria cuyos nombres son *x* e *y*, guardan cualquier valor entero si la variable es local o cero si la variable es de instancia o de clase. El uso de una variaable local antes de ser convenientemente inicializada puede conducir a consecuencias desastrosas. Por tanto, declarar e inicializar una variable es una práctica aconsejable.

En la tercera línea 30 es un número de tipo **int** por defecto, le ponemos el sufijo **L** en mayúsculas o minúsculas para indicar que es de tipo **long.**

Existen como vemos en la tabla varios tipos de números enteros (**byte**, **short**, **int**, **long**), y también existe una clase denominada *BigInteger* cuyos objetos pueden guardar un número entero arbitrariamente grande.

Variables en coma flotante

Las variables del tipo **float** o **double** (coma flotante) se usan para guardar números en memoria que tienen parte entera y parte decimal.

```
double PI=3.14159;
double g=9.7805, c=2.9979e8;
```

El primero es una aproximación del número real π, el segundo es la aceleración de la gravedad a nivel del mar, el tercero es la velocidad de la luz en m/s, que es la forma de escribir $2.9979\ 10^8$. El carácter punto '.', separa la parte entera de la parte decimal, en vez del carácter coma ',' que usamos habitualmente en nuestro idioma.

Otros ejemplos son los siguientes

```
float a=12.5f;
float b=7f;
double c=7.0;
double d=7d;
```

En la primera línea 12.5 lleva el sufijo **f**, ya que por defecto 12.5 es **double**. En la segunda línea 7 es un entero y por tanto 7f es un número de tipo **float**. Y así el resto de los ejemplos.

Conceptualmente, hay infinitos números de valores entre dos números reales. Ya que los valores de las variables se guardan en un número prefijado de bits, algunos valores no se pueden representar de forma precisa en memoria. Por tanto, los valores de las variables en coma flotante en un ordenador solamente se aproximan a los verdaderos números reales en matemáticas. La aproximación es tanto mejor, cuanto mayor sea el tamaño de la memoria que reservamos para guardarlo. De este hecho, surgen las variables del tipo **float** y **double**. Para números de precisión arbitraria se emplea la clase *BigDecimal*.

Valores constantes

Cuando se declara una variable de tipo **final**, se ha de inicializar y cualquier intento de modificarla en el curso de la ejecución del programa da lugar a un error en tiempo de compilación.

Normalmente, las constantes de un programa se suelen poner en letras mayúsculas, para distinguirlas de las que no son constantes. He aquí ejemplos de declaración de constantes.

```
final double PI=3.141592653589793;
```

```
final int MAX_DATOS=150;
```

8. Las cadenas de caracteres o strings

Además de los ocho tipos de datos primitivos, las variables en Java pueden ser declaradas para guardar una instancia de una clase, como veremos más adelante.

Las cadenas de caracteres o strings son distintas en Java y en el lenguaje C/C++, en este último, las cadenas son arrays de caracteres terminados en el carácter '**\0**'. Sin embargo, en Java son objetos de la clase *String*.

```
String mensaje="El primer programa";
```

Empleando strings, el primer programa quedaría de la forma equivalente

```
public class PrimeroApp{
    public static void main(String[] args) {
//imprime un mensaje
        String mensaje="El primer programa";
        System.out.println(mensaje);
    }
}
```

En una cadena se pueden insertar caracteres especiales como el carácter tabulador '\t' o el de nueva línea '\n'

```
String texto="Un string con \t un carácter tabulador y \n un salto de línea";
```

9. Palabras reservadas

En el siguiente cuadro se listan las palabras reservadas, aquellas que emplea el lenguaje Java, y que el programador no puede utilizar como identificadores. Algunas de estas palabras le resultarán familiares al programador del lenguaje C/C++. Las palabras reservadas señaladas con un asterisco (*) no se utilizan.

abstract	boolean	break	byte	byvalue*
case	cast*	catch	char	class
const*	continue	default	do	double
else	extends	false	final	finally
float	for	future*	generic*	goto*
if	implements	import	inner*	instanceof
int	interface	long	native	new

null	operator*	outer*	package	private
protected	public	rest*	return	short
satatic	super	switch	synchronized	this
throw	transient	true	try	var*
void	volatile	while		

Las palabras reservadas se pueden clasificar en las siguientes categorías:

- Tipos de datos: **boolean, float, double, int, char**
- Sentencias condicionales: **if, else, switch**
- Sentencias iterativas: **for, do, while, continue**
- Tratamiento de las excepciones: **try, catch, finally, throw**
- Estructura de datos: **class, interface, implements, extends**
- Modificadores y control de acceso: **public, private, protected, transient**
- Otras: **super**, **null, this**.

10. Tipos Referenciados

¿No te parece que los tipos vistos hasta ahora son un poco limitados? Por ejemplo, no parece razonable que, para almacenar las notas de los 800 alumnos y alumnas de un centro escolar para la primera, segunda y tercera evaluación, tuviéramos que declarar en nuestro programa 3 x 800 = 2.400 variables, ¿no te parece? ¿Y qué pasa si el curso próximo el centro tiene 805 alumnos? Tendríamos que ir cambiando el programa cada vez que cambiara el número de alumnos y alumnas. O si quisiéramos meter una cuarta nota para la evaluación final, etc. ¡¡Se haría inmanejable!!

A partir de los ocho tipos de 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 (le llamamos array de enteros, vector de enteros). En la segunda instrucción estamos declarando la variable u objeto cuentaCliente como una referencia de tipo Cuenta. (Cuenta es una clase que habrá definido el usuario, y que especificará qué elementos forman parte de una cuenta, y qué operaciones se pueden hacer con ella).

Como comentábamos al principio del apartado de Tipos de datos, cualquier aplicación de hoy en día necesita no perder de vista una cierta cantidad de datos. 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.

Además de los ocho tipos de datos primitivos que ya hemos descrito, Java proporciona un tratamiento especial a los textos o cadenas de caracteres mediante el tipo de dato String (comentados anteriormente). En realidad se trata de objetos, y por tanto son tipos referenciados, pero **el lenguaje nos permite utilizarlos también de forma sencilla como si fueran variables de tipos primitivos**:

```
String primerMensaje;
primerMensaje="El primer mensaje";
String segundoMensaje="Otro mensaje más";
```

Hemos visto qué son las variables, cómo se declaran y los tipos de datos que pueden adoptar. Anteriormente hemos visto también un ejemplo de creación de variables. Ahora vamos a crear más variables, pero de distintos tipos primitivos y las vamos a mostrar por pantalla. Los tipos referenciados los veremos en unidades posteriores.

Aquí tienes otro ejemplo donde se declaran más variables, una de ellas de tipo referenciado:

```
public class EjemploTipos {

   public static void main(String[] args) {
      int i = 10;
      double d = 3.14;
      char c1 = 'a';
      char c2 = 65;
      boolean encontrado = true;
      String mensaje = "Bienvenido a Java";

      System.out.println("La variable i es de tipo entero y su valor es: " + i);
      System.out.println("La variable d es de tipo double y su valor es: " + d);
      System.out.println("La variable c1 es de tipo carácter y su valor es: " +
c1);
      System.out.println("La variable c2 es de tipo carácter y su valor es: " +
c2);
```

```
        System.out.println("La variable encontrado es de tipo booleano y su valor es: " + encontrado);
        System.out.println("La variable mensaje es de tipo String y su valor es: " + mensaje);
    }
}
```

El resultado que se obtendrá por pantalla al ejecutar el programa tendrá el siguiente aspecto:

La variable i es de tipo entero y su valor es: 10

La variable d es de tipo double y su valor es: 3.14

La variable c1 es de tipo carácter y su valor es: a

La variable c2 es de tipo carácter y su valor es: A

La variable “encontrado” es de tipo booleano y su valor es: true

La variable mensaje es de tipo String y su valor es: Bienvenido a Java

11. Tipos Enumerados

Los **tipos de datos enumerados** permiten 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.

Para declararlos se usa 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.

Al considerar Java este tipo de datos como si de una clase se tratara, no solamente podemos definir los valores de un tipo enumerado, sino que también podemos definir operaciones a realizar con él y otro tipo de elementos, lo que hace que este tipo de dato sea más versátil y potente que en otros lenguajes de programación. Pero eso se verá en unidades más avanzadas. Por el momento debemos quedarnos con que **los enum nos permiten definir un nuevo tipo cuyos valores posibles son los que nosotros definamos**.

Ejemplo:

```
public enum Dias {LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO};
```

12. Los operadores (aritméticos)

Java tiene cinco operadores aritméticos cuyo significado se muestra en la tabla adjunta

Operador	Nombre	Ejemplo
+	Suma	3+4
-	Diferencia	3-4
*	Producto	3*4
/	Cociente	20/7
%	Módulo	20%7

El cociente entre dos enteros da como resultado un entero. Por ejemplo, al dividir 20 entre 7 nos da como resultado 2.

El operador módulo da como resultado el resto de la división entera. Por ejemplo 20%7 da como resultado 6 que es el resto de la división entre 20 y 7.

El operador módulo también se puede emplear con números reales. Por ejemplo, el cociente entre 7.5 y 3.0 es 2.5 y el resto es cero, es decir, 7.5=3.0 x 2.5+ 0. El operador módulo, funciona de la siguiente forma 7.5=3.0 x 2+1.5, calcula la diferencia entre el dividendo (7.5) y el producto del divisor (3.0) por la parte entera (2) del cociente, devolviendo 1.5. Así pues, la operación 7.5%3.0 da como resultado 1.5.

Ejercicio 1:

Partiendo de una variable entera llamada x cuyo valor inicial es 6, haz que se muestre por pantalla la secuencia de números siguiente, aplicando una operación matemática sobre x:

6, 10, -4, 12, 3, 2, 9, -6, 0

Para hacer este ejercicio deberás realizar 9 operaciones matemáticas sobre la variable x, y deberás utilizar cada uno de los operadores vistos en este apartado al menos una vez.

Ejercicio 2:

Con el siguiente código se lee desde teclado el precio de un producto:

```
import java.util.Scanner;
```

```
public class CalcularIVA {
    public static void main(String args[]) {
      Scanner scanner=new Scanner(System.in);
      System.out.print("Introduce el precio del producto: ");
      double precioProducto=scanner.nextDouble();
    }
}
```

¿Podrías modificar el código anterior para que mostrara el precio con IVA de la siguiente forma? (ten en cuenta que el IVA es del 21%)

```
Introduce el precio del producto: 8,4
Importe del IVA: 1.764
Precio con IVA: 10.164
```

Ejercicio 3:

El siguiente código pregunta al usuario por la velocidad (en kilómetros por hora) y el tiempo (en segundos) de un vehículo.

```
import java.util.Scanner;


public class Velocidad {
    public static void main(String args[]) {
      Scanner scanner=new Scanner(System.in);
      System.out.print("Velocidad en kilometros por hora:");
      double velocidad=scanner.nextDouble();
      System.out.print("Tiempo en segundos:");
      int segundos=scanner.nextInt();
    }
}
```

¿Serías capaz de modificarlo para que calculase la distancia recorrida en kilómetros y en metros? Un ejemplo de salida sería:

```
Velocidad en kilometros por hora:30
Tiempo en segundos:90
Distancia recorrida en kilometros:0.75
Distancia recorrida en metros:750.0
```

El operador asignación

Nos habremos dado cuenta que el operador más importante y más frecuentemente usado es el operador asignación **=,** que hemos empleado para la inicialización de las variables. Así,

```
int numero;
numero=20;
```

la primera sentencia declara una variable entera de tipo **int** y le da un nombre (*num*ero). La segunda sentencia usa el operador asignación para inicializar la variable con el número 20.

Consideremos ahora, la siguiente sentencia.

```
a=b;
```

que asigna a *a* el valor de *b*. A la izquierda siempre tendremos una variable tal como *a*, que recibe valores, a la derecha otra variable *b*, o expresión que tiene un valor. Por tanto, tienen sentido las expresiones

```
a=1234;
double area=calculaArea(radio);
superficie=ancho*alto;
```

Sin embargo, no tienen sentido las expresiones

```
1234=a;
calculaArea(radio)=area;
```

Las asignaciones múltiples son también posibles. Por ejemplo, es válida la sentencia

```
c=a=b;          //equivalente a c=(a=b);
```

la cual puede ser empleada para inicializar en la misma línea varias variables

```
c=a=b=321;              //asigna 321 a a, b y c
```

El operador asignación se puede combinar con los operadores aritméticos

Operadores de asignación combinados en Java		
Operador	**Ejemplo en Java**	**Expresión equivalente**
+=	**op1 += op2**	**op1 = op1 + op2**
-=	**op1 -= op2**	**op1 = op1 - op2**

Operadores de asignación combinados en Java		
Operador	**Ejemplo en Java**	**Expresión equivalente**
*=	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

Ejercicio 1:

Dada la variable entera x, cuyo valor inicial es 10, haz un programa en el que se vaya modificando el valor de la variable x de manera que dicha variable tome la siguiente secuencia de valores:

5, 6, 12, 6, -4, 1

La variable x deberá modificarse haciendo una operación aritmética sobre el valor que ya posee, por ejemplo:

```
int x=10;
x=x+100; //La variable x toma el valor 110 (10 + 100)
```

Para hacer el ejercicio debes usar al menos una vez los operadores aritméticos de suma, resta, división y multiplicación. No olvides mostrar el valor de la variable x después de cada modificación.

Ejercicio 2:

Una factoría papelera confecciona **cuadernos** en los que se van alternando hojas de color **rojo** y **verde**. Siempre se comienza por el color rojo, siguiéndole el verde y comenzando nuevamente con el rojo.

Implementar un programa en Java que calcule, para un cuaderno de **20 hojas**, cuántas hojas contiene de cada color.

13. Concatenación de strings

En Java se usa el operador + para concatenar cadenas de caracteres o strings. Veremos en el siguiente apartado una sentencia como la siguiente:

```
System.out.println("la temperatura centígrada es " + tC);
```

El operador + cuando se utiliza con strings y otros objetos, crea un solo string que contiene la concatenación de todos sus operandos. Si alguno de los

operandos no es una cadena, se convierte automáticamente en una cadena. Por ejemplo, en la sentencia anterior el número del tipo **double** que guarda la variable *tC* se convierte en un string que se añade al string "la temperatura centígrada es ".

Como veremos más adelante, un objeto se convierte automáticamente en un string si su clase redefine la función miembro *toString* de la clase base *Object*.

Como vemos en el listado, para mostrar un resultado de una operación, por ejemplo, la suma de dos números enteros, escribimos

```
iSuma=ia+ib;
System.out.println("El resultado de la suma es " + iSuma);
```

Concatena una cadena de caracteres con un tipo básico de dato, que convierte automáticamente en un string.

El operador += también funciona con cadenas.

```
String nombre="Juan ";
nombre+="García";
System.out.println(nombre);
```

```
public class OperadorAp {
    public static void main(String[] args) {
        System.out.println("Operaciones con enteros");
        int ia=7, ib=3;
        int iSuma, iResto;
        iSuma=ia+ib;
        System.out.println("El resultado de la suma es "+iSuma);
        int iProducto=ia*ib;
        System.out.println("El resultado del producto es "+iProducto);
        System.out.println("El resultado del cociente es "+(ia/ib));
        iResto=ia%ib;
        System.out.println("El resto de la división entera es
"+iResto);

        System.out.println("**********************************");
        System.out.println("Operaciones con números decimales");
        double da=7.5, db=3.0;
        double dSuma=da+db;
        System.out.println("El resultado de la suma es "+dSuma);
        double dProducto=da*db;
        System.out.println("El resultado del producto es "+dProducto);
        double dCociente=da/db;
        System.out.println("El resultado del cociente es "+dCociente);
        double dResto=da%db;
        System.out.println("El resto de la división es "+dResto);
   }
}
```

14. La precedencia de operadores

El lector conocerá que los operadores aritméticos tienen distinta precedencia, así la expresión

```
a+b*c
```

es equivalente a

```
a+(b*c)
```

ya que el producto y el cociente tienen mayor precedencia que la suma o la resta. Por tanto, en la segunda expresión el paréntesis no es necesario. Sin embargo, si queremos que se efectúe antes la suma que la multiplicación tenemos de emplear los paréntesis

```
(a+b)*c
```

Para realizar la operación $\frac{a}{bc}$ escribiremos

```
a/(b*c);
```

o bien,

```
a/b/c;
```

En la mayoría de los casos, la precedencia de las operaciones es evidente, sin embargo, en otros que no lo son tanto, se aconseja emplear paréntesis. Como ejemplo, estudiemos un programa que nos permite convertir una temperatura en grados Fahrenheit en su equivalente en la escala Celsius. La fórmula de conversión es

$$tC = \frac{(tF - 32) * 5}{9}$$

cuya codificación es

```
tC=(tF-32)*5/9;
```

Las operaciones se realizan como suponemos, ya que, si primero se realizase el cociente 5/9, el resultado de la división entera sería cero, y el producto por el resultado de evaluar el paréntesis sería también cero. Si tenemos dudas sobre la precedencia de operadores podemos escribir

```
tC=((tF-32)*5)/9;
```

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

```
        int tF=80;
        System.out.println("la temperatura Fahrenheit es "+tF);
        int tC=(tF-32)*5/9;
        System.out.println("la temperatura centígrada es "+tC);
    }
}
```

15. La conversión automática y promoción (casting)

Cuando se realiza una operación, si un operando es entero (**int**) y el otro es de coma flotante (**double**) el resultado es en coma flotante (**double**).

```
int a=5;
double b=3.2;
double suma=a+b;
```

Cuando se declaran dos variables una de tipo **int** y otra de tipo **double**.

```
int entero;
double real=3.20567;
```

¿qué ocurrirá cuando asignamos a la variable *entero* el número guardado en la variable *real*?. Como hemos visto se trata de dos tipos de variables distintos cuyos tamaños en memoria es de 32 y 64 bits respectivamente. Por tanto, la sentencia

```
entero=real;
```

convierte el número real en un número entero eliminando los decimales. La variable *entero* guardará el número 3.

Se ha de tener cuidado, ya que la conversión de un tipo de dato en otro es una fuente frecuente de error entre los programadores principiantes. Ahora bien, supongamos que deseamos calcular la división 7/3, como hemos visto, el resultado de la división entera es 2, aún en el caso de que tratemos de guardar el resultado en una variable del tipo **double**, como lo prueba la siguiente porción de código.

```
int ia=7;
int ib=3;
double dc=ia/ib;
```

Si queremos obtener una aproximación decimal del número 7/3, hemos de promocionar el entero *ia* a un número en coma flotante, mediante un procedimiento denominado promoción o *casting*.

```
int ia=7;
int ib=3;
double dc=(double)ia/ib;
```

Como aplicación, consideremos el cálculo del valor medio de dos o más números enteros

```
int edad1=10;
int edad2=15;
double media=(double)(edad1+edad2)/2;
```

El valor medio de 10 y 15 es 12.5, sin la promoción se obtendría el valor erróneo 12.

Imaginemos ahora, una función que devuelve un entero **int** y queremos guardarlo en una variable de tipo **float**. Escribiremos

```
float resultado=(float)retornaInt();
```

Existen también conversiones implícitas realizadas por el compilador, por ejemplo, cuando pasamos un entero **int** a una función cuyo único parámetro es de tipo **long.**

16. Los operadores unarios

Los operadores unarios son:

- ++ Incremento
- -- Decremento

actúan sobre un único operando. Se trata de uno de los aspectos más confusos para el programador, ya que el resultado de la operación depende de que el operador esté a la derecha *i++* o a la izquierda *++i*.

Conoceremos, primero el significado de estos dos operadores a partir de las sentencias equivalentes:

```
i=i+1;          //añadir 1 a i
i++;
```

Del mismo modo, lo son

```
i=i-1;          //restar 1 a i
i--;
```

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

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

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

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

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

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

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

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

```
public class UnarioApp {
    public static void main(String[] args) {
        int i=8;
        int a, b, c;
        System.out.println("\tantes\tdurante\tdespués");
        i=8; a=i; b=i++; c=i;
        System.out.println("i++\t"+a+'\t'+b+'\t'+c);
        i=8; a=i; b=i--; c=i;
        System.out.println("i--\t"+a+'\t'+b+'\t'+c);

        i=8; a=i; b=++i; c=i;
        System.out.println("++i\t"+a+'\t'+b+'\t'+c);
        i=8; a=i; b=--i; c=i;
        System.out.println("--i\t"+a+'\t'+b+'\t'+c);

    }
}
```

La salida del programa es, la siguiente

```
              antes          durante        después
i++             8              8              9
i--             8              8              7
++i             8              9              9
--i             8              7              7
```

La primera columna (antes) muestra el valor inicial de *i*, la segunda columna (durante) muestra el valor de la expresión, y la última columna (después) muestra el valor final de *i*, después de evaluarse la expresión.

Se deberá de tener siempre el cuidado de inicializar la variable, antes de utilizar los operadores unarios con dicha variable.

17. Los operadores relacionales

Los operadores relacionales son símbolos que se usan para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa. Por ejemplo, 8>4 (ocho mayor que cuatro) es verdadera, se representa por el valor **true** del tipo básico **boolean**, en cambio, 8<4 (ocho menor que cuatro) es falsa, **false**. En la primera columna de la tabla, se dan los símbolos de los operadores relacionales, en la segunda, el nombre de dichos operadores, y a continuación su significado mediante un ejemplo.

Operador	**nombre**	**ejemplo**	**significado**
<	menor que	*a<b*	*a* es menor que *b*
>	mayor que	*a>b*	*a* es mayor que *b*
==	igual a	*a==b*	*a* es igual a *b*
!=	no igual a	*a!=b*	*a* no es igual a *b*
<=	menor que o igual a	*a<=5*	*a* es menor que o igual a *b*
>=	mayor que o igual a	*a>=b*	*a* es menor que o igual a *b*

Se debe tener especial cuidado en no confundir el operador asignación con el operador relacional igual a. Las asignaciones se realizan con el símbolo =, las comparaciones con ==.

En el programa *RelacionApp*, se compara la variable *i* que guarda un 8, con un conjunto de valores, el resultado de la comparación es verdadero (**true**), o falso (**false**).

```
public class RelacionApp {
    public static void main(String[] args) {
        int x=8;
       int y=5;
       boolean compara=(x<y);
       System.out.println("x<y es "+compara);
        compara=(x>y);
        System.out.println("x>y es "+compara);
        compara=(x==y);
       System.out.println("x==y es "+compara);
        compara=(x!=y);
        System.out.println("x!=y es "+compara);
```

```
        compara=(x<=y);
        System.out.println("x<=y es "+compara);
        compara=(x>=y);
        System.out.println("x>=y es "+compara);
    }
}
```

Hasta ahora hemos visto ejemplos que creaban variables y se inicializaban con algún valor. Pero ¿y si lo que queremos es que el usuario introduzca un valor al programa?

Entonces debemos agregarle interactividad a nuestro programa, por ejemplo utilizando la clase Scanner. Aunque no hemos visto todavía qué son las clases y los objetos, de momento vamos a pensar que la clase Scanner nos va a permitir leer los datos que se escriben por teclado, y que para usarla es necesario importar el paquete de clases que la contiene. El código del ejemplo lo tienes a continuación. El programa se quedará esperando a que el usuario escriba algo en el teclado y pulse la tecla intro. En ese momento se convierte lo leído a un valor del tipo int y lo guarda en la variable indicada. Además de los operadores relacionales, en este ejemplo utilizamos también el operador condicional, que compara si los números son iguales. Si lo son, devuelve la cadena iguales y si no, la cadena distintos.

```
import java.util.Scanner; //importamos el paquete necesario para poder usar la clase Scanner


public class EjemploRelacionales {


  public static void main( String args[] ){


    Scanner teclado = new Scanner( System.in );
    int x, y;
    String cadena;
    boolean resultado;


    System.out.print( "Introducir primer número: " );
    x = teclado.nextInt(); // pedimos el primer número al usuario
    System.out.print( "Introducir segundo número: " );
    y = teclado.nextInt(); // pedimos el segundo número al usuario


    // realizamos las comparaciones y las mostramos por pantalla
    cadena=(x==y)?"iguales":"distintos";
    System.out.printf("Los números %d y %d son %s\n",x,y,cadena);
```

```
        resultado=(x!=y);
        System.out.println("x != y  // es " + resultado);
        resultado=(x < y );
        System.out.println("x < y  // es " + resultado);
        resultado=(x > y );
        System.out.println("x > y  // es " + resultado);
        resultado=(x <= y );
        System.out.println("x <= y  // es " + resultado);
        resultado=(x >= y );
        System.out.println("x >= y  // es " + resultado);
    }
}
```

Ejercicio 1:

Señala cuáles son los operadores relacionales en Java.

1. = =, ! =, >, <, > =, < =.
2. =, !=, >, <, >=, <=.
3. ==, =!, >, <, =>, =<.
4. ==, !=, >, <, >=, <=.

Ejercicio 2

Dada una variable entera x cuyo valor inicial es 5, y otra variable entera z cuyo valor inicial es 6, haz que se muestre la secuencia de resultados: true, **false**, true, **false**, true, y false (6 en total); usando un operador de relación diferente en cada caso.

18. Los operadores lógicos

Los operadores lógicos son:

- && AND (el resultado es verdadero si ambas expresiones son verdaderas)
- || OR (el resultado es verdadero si alguna expresión es verdadera)
- ! NOT (el resultado invierte la condición de la expresión)

AND y OR trabajan con dos operandos y retornan un valor lógico basadas en las denominadas tablas de verdad. El operador NOT actúa sobre un operando. Estas tablas de verdad son conocidas y usadas en el contexto de la vida diaria, por ejemplo: "si hace sol Y tengo tiempo, iré a la playa", "si NO hace sol, me quedaré en casa", "si llueve O hace viento, iré al cine". Las tablas de verdad de los operadores AND, OR y NOT se muestran en las tablas siguientes

El operador lógico AND

x	**y**	**resultado**
true	true	true
true	false	false
false	true	false
false	false	false

El operador lógico OR

x	**y**	**resultado**
true	true	true
true	false	true
false	true	true
false	false	false

El operador lógico NOT

x	**resultado**

true	false
false	true

Los operadores AND y OR combinan expresiones relacionales cuyo resultado viene dado por la última columna de sus tablas de verdad. Por ejemplo:

```
(a<b) && (b<c)
```

es verdadero (**true**), si ambas son verdaderas. Si alguna o ambas son falsas el resultado es falso (**false**). En cambio, la expresión

```
(a<b) ||(b<c)
```

es verdadera si una de las dos comparaciones lo es. Si ambas, son falsas, el resultado es falso.

La expresión " NO *a* es menor que *b*"

```
!(a<b)
```

es falsa si *(a<b)* es verdadero, y es verdadera si la comparación es falsa. Por tanto, el operador NOT actuando sobre *(a<b)* es equivalente a

```
(a>=b)
```

La expresión "NO *a* es igual a *b*"

```
!(a==b)
```

es verdadera si *a* es distinto de *b*, y es falsa si *a* es igual a *b*. Esta expresión es equivalente a

```
(a!=b)
```

Ejercicio 1:

Dadas las variables siguientes:

```
int x1=10, x2=5, x3=0;
char c1='F', c2='S';
```

Crea una expresión lógica, que utilice operadores lógicos y relacionales, para cada uno de los siguientes casos (intenta evaluar mentalmente el resultado de la expresión antes de mostrarlo por pantalla):

- x1 es igual a x2

- c1 es distinto a c2
- x1 está entre 10 y **100**
- x2 no está entre 10 y **100**
- x3 no está entre 10 y **100**
- x1 es mayor que x2 y **c1** es igual a c2
- O x1 es mayor que x2, o **c1** es distinto a c2, cualquiera de los dos casos.
- x1 es menor o igual que 7 y c2 es igual a c1
- c2 es distinto de 'A' y **x2** es mayor que 0
- 'F' es distinto de c1 o **x1** es mayor que 20
- 'F' es distinto de c1 o **x1** es mayor que 20 o **x2** es mayor que 2
- 'F' es igual a c1 y **x3** es menor que x1
- 'F' es igual a c1 y **x3** es menor que x1 y **x2** es menor o igual que x3
- x2 está entre x3 y **x1**, y c2 es **'S'**
- x3 no está entre x2 y **x1**
- x2 no está entre x3 y **x1**, o c2 es igual a c1
- no se cumple que x3 es menor que x1
- ni x3 es mayor que x1, ni c2 es distinto a c1
- no se cumple que x1 es menor que 100 y **x2** es mayor que 10
- c2 es anterior alfabéticamente a c1

19. Operador Condicional

El **operador condicional** "? :" sirve para **evaluar una condición y devolver un resultado u otro** en función de si es verdadera o falsa dicha condición. Es el único **operador ternario** de Java, y como tal, **necesita tres operandos** para formar una expresión.

- El **primer operando** se sitúa **a la izquierda del símbolo de interrogación** (?), y siempre será una **expresión booleana**, también llamada condición.
- El **siguiente operando** se sitúa **a la derecha del símbolo de interrogación** (?) y **antes de los dos puntos** (:), y es el **valor que devolverá el operador condicional si la condición es verdadera**.
- El **tercer y último operando**, que aparece **después de los dos puntos** (:), es la **expresión cuyo resultado se devolverá si la condición evaluada es falsa**.

Operador condicional en Java

Operador	Expresión en Java
? :	condición ? exp1 : exp2

Por ejemplo, en la expresión:

(x>y) ? x : y ;

Se evalúa la condición de si x es mayor que y. En caso afirmativo se devuelve el valor de la variable x, y en caso contrario se devuelve el valor de y. Se trata de una manera muy elegante de calcular el **máximo** de esos dos valores.

El operador condicional se puede sustituir por la sentencia if-then-else que veremos en la siguiente unidad sobre las **estructuras de control**, de la que viene a ser una versión abreviada muy útil para algunos casos, permitiendo hacer operaciones potentes con una única sentencia bastante simple. Por ejemplo, la línea anterior de ejemplo, devuelve el mayor de los dos números que se comparan, de una manera sintética y elegante, aunque lo mismo se podría haber hecho usando una sentencia if-then-else.

Ejercicio 1:

El siguiente código generará un número aleatorio entre 0 y 100 (aunque el 100 no estará incluido):

```
//d contendrá un número aleatorio entre 0 y 100.
double d=Math.random()*100;
```

Usando exclusivamente el operador condicional, junto con operadores de relación, lógicos y de asignación, escribe un código en Java que muestre por pantalla si el número aleatorio generado está entre los rangos siguientes:

- d está entre 0 y **20**, 20 no incluido.
- d está entre 20 y **50**, ambos incluidos.
- d está entre 50 y **75**, ninguno incluido.
- d está entre 75 y **100**, ambos incluidos.

Ejercicio 2:

Sabemos si un número es **par** o **impar** (**divisible entre dos**) si **el resto de la división entera** de ese número entre dos es **cero** o **uno**.

Escribe un programa en Java que pida un número entero al usuario e indique si ese número es **par o impar**.

20. Operadores de bits

Los operadores a nivel de bits se caracterizan porque realizan operaciones sobre números enteros (o char) en su representación binaria, es decir, sobre cada dígito binario.

Aunque estos operadores no son de uso frecuente, sino más bien para aplicaciones muy específicas, no está de más que al menos sepas cuáles son. En la tabla tienes los operadores a nivel de bits que utiliza Java.

Operadores a nivel de bits en Java		
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 izquierda 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:

Operadores de bits

21. Trabajo con cadenas

Ya hemos visto que el objeto String se corresponde con una secuencia de caracteres entrecomillados, como por ejemplo “hola”. Este literal se puede utilizar en Java como si de un tipo de datos primitivo se tratase, y como caso especial, no necesita del operador new para ser creado.

No se trata aquí de que nos adentremos en lo que es una clase u objeto, puesto que lo veremos en unidades posteriores, y trabajaremos mucho sobre ello. Aquí sólo vamos a utilizar determinadas operaciones que podemos realizar con el objeto String, y lo verás mucho más claro con ejemplos descriptivos.

Para aplicar una operación a una variable de tipo String, escribiremos su nombre seguido de la operación, separados por un punto. Entre las principales operaciones que podemos utilizar para trabajar con cadenas de caracteres están las siguientes:

- **Creación**. Como hemos visto en el apartado de literales, podemos crear una variable de tipo **String** simplemente asignándole una cadena de caracteres encerrada entre comillas dobles.
- **Obtención del carácter** que se encuentra en una posición determinada de la cadena. Para ello se utiliza el método **charAt**.
- **Obtención de longitud**. Si necesitamos saber la longitud de un **String**, utilizaremos el método **length**.
- **Concatenación**. Se utiliza el operador + o el método **concat()** para concatenar cadenas de caracteres.
- **Comparación**. El método **equals** nos devuelve un valor booleano que indica si las cadenas comparadas son o no iguales. El método **equalsIgnoreCase** hace lo propio, ignorando las mayúsculas de las cadenas a considerar. Las comparaciones entre objetos **String** nunca deben hacerse con el operador ==.
- **Obtención de subcadenas**. Podemos obtener cadenas derivadas de una cadena original con el método **substring()**, al cual le debemos indicar el inicio y el fin de la subcadena a obtener.
- **Cambio a mayúsculas/minúsculas**. Los métodos **toUpperCase** y **toLowerCase** devuelven una nueva variable que transforma en mayúsculas o minúsculas, respectivamente, la variable inicial.
- **Conversiones.** Utilizaremos el método **valueOf** para convertir un tipo de dato primitivo (**int**, **long**, **float**, etc.) a una variable de tipo **String**.

A continuación, tienes varios ejemplos de las distintas operaciones que podemos realizar con cadenas de caracteres o String en Java:

```
public class EjemploCadenas {

    public static void main(String[] args) {

        // Cadenas de ejemplo con las que trabajar
        String cadena1 = "CICLO DAM-DAW";
        String cadena2 = "ciclo dam-daw";

        System.out.println ("EJEMPLOS DE OPERACIONES CON CADENAS");
        System.out.println ("-----------------------------------");

        // Mostramos las cadenas originales
        System.out.println ("La cadena cadena1 es " + cadena1);
        System.out.println ("La cadena cadena2 es " + cadena2);
        System.out.println ();

        System.out.println ("Longitud de cadena1: " + cadena1.length());
        System.out.println ("Longitud de cadena2: " + cadena2.length());

        // Concatenación de cadenas (concat o bien operador +)
        System.out.println ("Concatenación cadena1 y cadena2: " +
cadena1.concat(cadena2));
        System.out.println ("Concatenación cadena2 y cadena1: " + cadena2 + cadena1);

        // Comparación de cadenas
        System.out.println ("cadena1.equals(cadena2) es: " + cadena1.equals(cadena2));
        System.out.println ("cadena1.equalsIgnoreCase(cadena2) es: " +
cadena1.equalsIgnoreCase(cadena2));

        // Obtención de subcadenas
        System.out.println ("cadena1.substring(0,5) es: " + cadena1.substring(0, 5));

        // Pasar a minúsculas
        System.out.println ("cadena1.toLowerCase() es: " + cadena1.toLowerCase());
        System.out.println();
    }
```

```
}
```

El resultado que se debería obtener es el siguiente:

```
EJEMPLOS DE OPERACIONES CON CADENAS
----------------------------------
La cadena cadena1 es CICLO DAM-DAW
La cadena cadena2 es ciclo dam-daw

Longitud de cadena1: 13
Longitud de cadena2: 13
Concatenación cadena1 y cadena2: CICLO DAM-DAWciclo dam-daw
Concatenación cadena2 y cadena1: ciclo dam-dawCICLO DAM-DAW
cadena1.equals(cadena2) es: false
cadena1.equalsIgnoreCase(cadena2) es: true
cadena1.substring(0,5) es: CICLO
cadena1.toLowerCase() es: ciclo dam-daw
```

Ejercicio 1:

Dada la variable cadena tipo String, haz que vaya mostrando por pantalla la secuencia siguiente:

```
La casa de
La casa de Juan es
La casa de Juan es el número
La casa de Juan es el número 25.
```

Para ello tienes que ir modificando el valor de la variable cadena, partiendo del valor que tiene con anterioridad. Para ello puedes usar una de las dos formas siguientes:

```
cadena=cadena+"texto añadido";
cadena+="texto añadido"
```

Fíjate por último que 25 es un número, y no un texto.

Ejercicio 2:

¿Cuál de las siguientes formas es la forma aconsejada para comparar si dos cadenas son iguales entre sí?

```
cad1.equals(cad2)
cad1==cad2
```

```
cad1===cad2
cad1.compareTo(cad2)
```

Ejercicio 3:

El siguiente código Java pregunta al usuario por una palabra y la muestra por pantalla:

```
import java.util.Scanner;


public class Palabra {
   public static void main(String args[]) {
     Scanner teclado=new Scanner(System.in);
     System.out.print ("Introduzca una palabra: ");
     String palabra= teclado.nextLine();
     System.out.print ("La palabra introducida es " + palabra);
   }
}
```

Para obtener el carácter ubicado en una determinada posición de una cadena en Java disponemos del método charAt teniendo en cuenta que la primera posición es cero y que la última es la del tamaño de la cadena menos uno.

Por ejemplo, dada la siguiente cadena:

```
String palabra= "ejemplo";
```

podríamos obtener el carácter ubicado en la primera posición aplicando a la variable palabra el método charAt con el valor 0:

```
char primeraLetra= palabra.charAt (0);
```

en tal caso obtendríamos el carácter 'e' ¿Cómo obtendríamos el último? De manera similar, pero accediendo a la posición palabra.length() - 1:

```
char ultimaLetra= palabra.charAt (palabra.length()-1);
```

en este caso obtendríamos el carácter 'o'.

Amplía el fragmento de código anterior para que además de solicitar una palabra se muestren por pantalla la primera y la última letra de esa palabra.

Por ejemplo, si la palabra introducida es "prueba" el programa debería mostrar por pantalla los caracteres 'p' y **'a'**.

22. Conversión de tipos

Imagina que queremos dividir un número entre otro: ¿tendrá decimales el resultado de esa división?

Podemos pensar que siempre que el denominador no sea divisible entre el divisor, tendremos un resultado con decimales, pero no es así.

Si el denominador y el divisor son variables de tipo entero, el resultado será entero y no tendrá decimales. Para que el resultado tenga decimales necesitaremos hacer una **conversión de tipo**.

Las conversiones de tipo se realizan para hacer que el resultado de una expresión sea del tipo que nosotros deseamos. En el ejemplo anterior, para hacer que el resultado de la división sea de tipo real, con decimales, debemos hacer una conversión de tipo. Existen dos tipos de conversiones:

- **Conversiones automáticas**. 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**).

- **Conversiones explícitas**. 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 derecha a izquierda. Las conversiones explícitas también suelen ser conocidas como **casting**.

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**. Por 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. El compilador detecta que a es "un recipiente más grande", y que por tanto si intentamos volcar su contenido en b que es un recipiente más pequeño, puede no caber, así que mejor avisar del error, y no dejar ni intentarlo, no sea que se pierda el contenido "por desbordamiento del recipiente".

Lo correcto es forzar la conversión de a al tipo de datos byte, (puesto que su contenido, 12, realmente puede ser visto como un literal byte también) y entonces asignarle su valor a la variable b.

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	-

Explicación de los símbolos utilizados:

- **N:** Conversión no permitida (un **boolean** no se puede convertir a ningún otro tipo y viceversa).
- **CI:** Conversión implícita o automática. Un asterisco indica que puede haber posible pérdida de datos.
- **C:** Casting de tipos o conversión explícita.
- -: Mismo tipo. No hay que convertir nada.

El asterisco indica que puede haber **una posible pérdida de datos**, por ejemplo al convertir un número de tipo **int** que usa los 32 bits posibles de la representación, a un tipo **float**, que también usa 32 bits para la representación, pero 8 de los cuales son para el exponente.

En cualquier caso, las conversiones de números en coma flotante a números enteros siempre necesitarán un **casting**, y deberemos tener mucho cuidado debido a la pérdida de precisión que ello supone.

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:**No se encuentran entradas de índice.**

- Si uno de los operandos es de tipo **double**, el otro es convertido a **double**.
- En cualquier otro caso:
    - Si 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, (ningún operando es **double**, ni **float**, ni **long**) entonces ambos operandos son convertidos al tipo **int**.

<table>
<tr><th colspan="3">Tabla sobre otras consideraciones con los Tipos de Datos</th></tr>
<tr><th>Conversiones de números en Coma flotante (float, double) a enteros (int)</th><th>Conversiones entre caracteres (char) y enteros (int)</th><th>Conversiones de tipo con cadenas de caracteres (String)</th></tr>
<tr>
<td>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(numero) : Redondeo al siguiente número entero.
• Math.ceil(numero): El menor de los enteros que sigue siendo mayor o igual a numero.
• Math.floor(numero): El mayor de los enteros que sigue siendo inferior o igual a numero.

double numero=3.5;
x=Math.round(numero); // x = 4
y=Math.ceil(numero); // y = 4
z=Math.floor(numero); // z = 3</td>
<td>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 numero;
char c;
numero = (int) 'A'; // numero = 65
c = (char) 65; // c = 'A'
c = (char) ((int) 'A' + 1); // c = 'B'</td>
<td>Para convertir cadenas de texto a otros tipos de datos se utilizan las siguientes funciones:

numero=Byte.parseByte(cadena);
numero=Short.parseShort(cadena);
numero=Integer.parseInt(cadena);
numero=Long.parseLong(cadena);
numero=Float.parseFloat(cadena);
numero=Double.parseDouble(cadena);

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);</td>
</tr>
</table>