



2. Creación de mi primer programa

| | |
|-------|----------------------|
| Autor | Xerach Casanova |
| Clase | Programación |
| Fecha | @Dec 8, 2020 8:34 AM |

1. [Introducción](#)
2. [Las variables e identificadores](#)
 - 2.1. [Identificadores](#)
 - 2.2. [Convenios y reglas para nombrar variables](#)
 - 2.3. [Palabras reservadas](#)
 - 2.4. [Tipos de variables](#)
3. [Los tipos de datos](#)
 - 3.1. [Tipos de datos primitivos I](#)
 - 3.1.1. [Tipos primitivos II](#)
 - 3.2. [Declaración e inicialización](#)
 - 3.3. [Tipos referenciados](#)
 - 3.2. [Tipos Enumerados](#)
4. [Literales de los tipos primitivos](#)
5. [Operadores y expresiones](#)
 - 5.1. [Operadores aritméticos](#)
 - [Operadores aritméticos básicos](#)
 - [Operadores incrementales en Java](#)
 - 5.2. [Operadores de asignación](#)
 - 5.3. [Operador condicional](#)
 - 5.4. [Operadores de relación](#)
 - 5.5. [Operadores lógicos](#)
 - 5.6. [Operadores de bits](#)
 - 5.7. [Trabajo con cadenas](#)
 - 5.8. [Precedencia de operadores](#)
6. [Conversión de tipo](#)
 - 6.1. [Anexo Conversión de tipos](#)
 - [Otras consideraciones con los tipos de datos](#)
7. [Comentarios](#)

1. Introducción

Los programas de ordenador deben resolver un problema y para ellos se debe utilizar de forma inteligente y lógica todos los elementos que nos ofrece el lenguaje. Java es un lenguaje multiplataforma, robusto y fiable y además es orientado a objetos.

2. Las variables e identificadores

Una variable es una zona de memoria en el ordenador con un valor que se almacena para ser utilizado más tarde en el programa. Se componen de:

- **Nombre.** Permite al programa acceder al valor que contiene en memoria y debe ser un identificador válido.
- **Tipo de dato.** Especifica qué tipo de información guarda.
- **Rango de valores** que puede admitir.

2.1. Identificadores

Un identificador es una secuencia ilimitada sin espacios de letras y dígitos unicode. El primer símbolo debe ser una letra, subrayado (_) o símbolo de dólar (\$), sin embargo, se desaconseja el uso distinto al del uso de letras.

Unicode es un código de caracteres que recoge los caracteres de prácticamente todos los idiomas importantes del mundo y las líneas de código de los programas se escriben usando unicode, por tanto java se puede utilizar en distintos alfabetos.

El estándar Unicode utilizaba 16 bits pudiendo representar hasta 65.536 caracteres distintos, sin embargo actualmente utiliza más o menos bits dependiendo del formato a utilizar: UTF-8, UTF-16 o UTF-32, en el cual a cada carácter le corresponde un número entero entre 0 a 2 elevado a n, siendo n el número de bits utilizados.

2.2. Convenios y reglas para nombrar variables

Aún no siendo obligatorias, existen una serie de normas de estilo de uso generalizado. Estas reglas son:

- Java distingue mayúsculas y minúsculas.
- No utilizar identificadores que comiencen con \$ o con _ además por convenio el \$ no se utiliza nunca.
- No se puede utilizar true o false ni null.
- Los identificadores deben ser descriptivos. Mejor usar palabras completas en vez de abreviaturas.

Además se recomienda:

- Las variables deben comenzar por letra minúsculas y si hay más de una palabra, se colocan juntas y comenzando por mayúsculas.
- Las constantes se declaran en letras mayúsculas, separando las palabras con guión bajo.
- Las clases comienzan por letra mayúscula y si hay más de una palabra, éstas comienzan por mayúscula.
- Las funciones comienzan en letra minúscula.

2.3. Palabras reservadas

También llamadas palabras clave o keywords. Su uso se reserva al lenguaje y no pueden utilizarse para crear identificadores.

| | | | | |
|----------|----------|------------|-------------|--------------|
| 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 | strictfcode | volatile |
| const | float | native | super | while |

Algunas palabras reservadas ya no se utilizan en la actualidad en Java **const** o **goto**. Tampoco puede utilizarse **true y false**, pero se consideran **literales booleanos** y tampoco **null**, considerado igualmente un literal.

Los editores e IDE utilizan colores para diferenciar palabras reservadas. Estos colores pueden modificarse en las opciones de menú (netbeans: tools - options / fonts & colors)

2.4. Tipos de variables

Los distintos tipos de variables dependen del tipo de datos que representan, si su valor cambia o no durante ejecución o qué papel llevan a cabo.

- **Variables de tipos primitivos y variables referencia.**

Según el tipo de información que contengan. En función a que grupo pertenezca, tipos primitivos o tipos referenciados, se podrán realizar con ellas unas operaciones u otras.

- **Variables y constantes.**

Dependen de si su valor cambia o no durante la ejecución del programa.

- **Variable:** representa una zona de memoria del ordenador que contiene un determinado valor y al que se accede a través del identificador.

- **Constantes o variables finales:** lo mismo que las variables, pero su valor no cambia en tiempo de ejecución.
- **Variables miembro y variables locales.**
En función del lugar donde aparezcan en el programa.
 - **Variables miembro.** Se crean dentro de una clase, fuera de cualquier método. Pueden ser de tipos primitivos o referencia, variables o constantes.
 - **Variables locales:** se crean y se usan dentro de un método o dentro de cualquier bloque de código. La variable deja de existir cuando la ejecución del bloque o método finaliza.

Ejemplo de utilización de variables:

Constante: PI

Variable miembro: x

Variable local: valorantiguo

```
/**
 * Aplicación ejemplo de tipos de variables
 *
 * @author FMA
 */
public class ejemplovariables {
    final double PI = 3.1415926536; // PI es una constante
    int x;                          // x es una variable miembro
                                   // de clase ejemplovariables

    int obtenerX(int x) {           // x es un parámetro
        int valorantiguo = this.x; // valorantiguo es una variable local
        return valorantiguo;
    }

    // el método main comienza la ejecución de la aplicación
    public static void main(String[] args) {
        // aquí iría el código de nuestra aplicación
    } // fin del método main
} // fin de la clase ejemplovariables
```

3. Los tipos de datos

En los lenguajes fuertemente tipados a todo dato: constante, variable o expresión, le corresponde un tipo que es conocido antes de que ejecute el programa.

Un lenguaje fuertemente tipado es un lenguaje que no permite la utilización de un dato distinto al tipo de dato de la variable a menos que se haga una conversión.

Un tipo de dato es una especificación de los valores que son válidos para la variable y de las operaciones que se pueden realizar con ellos.

El tipo de dato de una variable se conoce durante la revisión del compilador para detectar errores.

Los tipos de datos en java se dividen en

- **Tipos sencillos o primitivos:** valores simples predefinidos en el lenguaje: caracter, número...
- **Tipo de datos referencia:** se definen con un nombre o referencia (puntero) que contiene la dirección en memoria de un valor o grupo de valores, por ejemplo arrays o clases.

3.1. Tipos de datos primitivos I

Constituyen los tipos de información más habituales: números, caracteres y valores lógicos o booleanos.

En Java, los datos primitivos no se consideran objetos.

El compilador optimiza mejor el uso de los tipos primitivos comparado con los objetos. Los tipos primitivos tienen idéntico tamaño y comportamiento en todas las versiones de java y en todo tipo de ordenador, por tanto se asegura la portabilidad.

Tipos de datos primitivos en Java:

| Tipo | Descripción | Bytes | Rango | Valor por default |
|---------|--|-------|---|-------------------|
| byte | Entero muy corto | 1 | -128 a 127 | 0 |
| short | Entero corto | 2 | -32,768 a 32,767 | 0 |
| int | Entero | 4 | -2,147,483,648 a 2,147,483,647 | 0 |
| long | Entero largo | 8 | -9,223,372,036,854,775,808 a 9,223,372,036,854,775,807 | 0L |
| float | Numero con punto flotante de precisión individual con hasta 7 dígitos significativos | 4 | +/-1.4E-45 (+/-1.4 times 10 ⁻⁴⁵) a +/-3.4E38 (+/-3.4 times 10 ³⁸) | 0.0f |
| double | Numero con punto flotante de precisión doble con hasta 16 dígitos significativos | 8 | +/-4.9E-324 (+/-4.9 times 10 ⁻³²⁴) a +/-1.7E308 (+/-1.7 times 10 ³⁰⁸) | 0.0d |
| char | Carácter Unicode | 2 | \u0000 a \uFFFF | '\u0000' |
| boolean | Valor Verdadero o Falso | 1 | true o false | false |

Entre los tipos de datos primitivos existe una peculiaridad, esta es el tipo de dato char, el cual es considerado por el compilador como numérico, ya que los valores que guarda son el código unicode. Por tanto puede operarse con él como si se tratase de un número entero.

Para manejar datos mayores a la cifra de 2.17.483.647, no podemos utilizar int ya que es el número máximo de combinaciones posibles de 32 bits, por tanto tendremos que utilizar long o float, pero tienen un problema. **La precisión.**

3.1.1. Tipos primitivos II

El tipo de dato real permite representar cualquier número con decimales, en función del número de bits usado, hay más de un tipo de dato real. Cuanto mayor sea ese número:

- Más grande podrá ser el número real representado en valor absoluto.
- Mayor será la precisión de la parte decimal

El almacenamiento de los valores reales que son resultado de otros dos números reales, la gran mayoría se representará de forma aproximada, ya que en el ordenador solo se puede almacenar un número finito de bits.

Un número se expresa como :

$$\text{Valor} = \text{mantisa} * 2^{\text{exponente}}$$

Y **solo se almacena la mantisa y el exponente al que va elevada la base**. Los bits empleados por la mantisa representan la precisión del número real (el número de cifras decimales que puede tener, mientras que los bits del exponente expresan la diferencia entre el mayor y el menor número representable (intervalo de representación).

En java, float tiene precisión 32 bits (24 mantisa / 8 exponente). La mantisa es un valor entre -1.0 y 1.0 y el exponente representa la potencia de 2 para obtener el valor... double tiene precisión 64 bits (53 mantisa / 11 exponente).

Se suele utilizar tipo double, ya que se asegura que los errores cometidos en aproximaciones sean menores. Java considera los valores de coma flotante por defecto en tipo double.

Java utiliza el estándar internacional IEEE 754 para representación interna de números en coma flotante.

3.2. Declaración e inicialización

Para declarar una variable podemos hacerlo en cualquier bloque de código dentro de llaves. Se hace indicando identificador y tipo de dato, separadas por , si se declaran varias a la vez:

```
int alumnos = 15;
double radio = 3.14, importe = 102.95;
```

No es obligatorio dar valor a las variables, aunque dependiendo del caso, el compilador producirá error:

- **Variables miembro:** si no le damos valor, se inicializan de manera automática (numérico = 0, boolean = false y referenciado = null).
- **Variables locales:** no se inicializan automáticamente y debemos asignarles un valor antes de ser usadas. En el siguiente ejemplo daría error, porque p puede no ser inicializada:

```
int p;
if ( . . . )
    p = 5 ;
int q = p; // error
```


Para declarar constantes utilizamos la palabra reservada final:

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

3.3. Tipos referenciados

Los tipos referenciados se construyen a partir de los 8 tipos primitivos. Se utilizan para almacenar la dirección de los datos en la memoria.

```
int[] arrayDeEnteros; //array del tipo int

Cuenta cuentaCliente; //variable u objeto con una referencia de tipo cuenta.
```

Los arrays son datos agrupados en estructuras para facilitar el acceso a los mismos (datos estructurados).

Además de los 8 tipos primitivos, java trata a los textos o cadenas de caracteres mediante el tipo de dato String. Realmente son objetos y por tanto son tipos referenciados, pero se pueden utilizar como si fueran variables de tipos primitivos.

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

Para mostrar en pantalla se utiliza **System.out**, que es la salida estándar del programa. Este método escribe un conjunto de caracteres a través de la línea de comandos. Se puede utilizar:

- System.out.print
- System.out.println (sitúa el curso al principio de la línea siguiente).

Los comentarios se realizan con //.

En los entornos de desarrollo se incluyen funcionalidades para ayudar a escribir código.

```
7 public static void main(String[] args) {
8     // TODO code application logic here
9     int i= 10;
10    double d=3.24;
11
12    cannot find symbol
13    symbol: class string
14    location: class EjemplosTipos
15    (Alt-Enter shows hints)
16    String msj= "Bienvenido a Java";
17
18    System.out.println ("La variable i es de tipo entero y su valor es:" + i);
19
20 }
21
22 }
```

Si se coloca el ratón sobre el círculo rojo, Nebeans te informa sobre el error detectado.

También tiene las sugerencias de inserción de código y la generación automática de código, facilitando la inserción generando código automáticamente o sugiriéndote qué escribir en un determinado punto usando Ctrl + Espacio.

3.2. Tipos Enumerados

Es una forma de declarar una variable con un conjunto restringido de valores. Se utiliza la palabra reservada **enum** seguida del nombre de la variable y la lista de valores que puede tomar entre llaves. A estos valores se les considera constantes.

Las llaves se utilizan porque enum es una especie de clase en Java y todas las clases llevan su contenido entre llaves.

Al ser una clase, también podemos realizar operaciones con él.

A enum, al tener el tratamiento de clase, se le puede añadir métodos y campos o variables en la su declaración.

En el siguiente ejemplo se usa system.out.print. y a veces se utiliza la secuencia de escape llamada carácter de nueva línea (\n) que le indica al compilador que mueva el cursor al principio de la línea siguiente.

```

10 public class tiposenumerados {
11     public enum Dias {Lunes, Martes, Miercoles, Jueves, Viernes, Sabado, Domingo};
12
13     public static void main(String[] args) {
14         // codigo de la aplicacion
15         Dias diaactual = Dias.Martes;
16         Dias diasiguiente = Dias.Miercoles;
17
18         System.out.print("Hoy es: ");
19         System.out.println(diaactual);
20         System.out.println("Mañana\nes\n"+diasiguiente);
21
22     } // fin main
23
24 } // fin tiposenumerados

```

4. Literales de los tipos primitivos

Un literal, valor literal o constante literal es un valor concreto para los tipos primitivos, String o Null.

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

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

- **Decimal**
- **Octal:** Empieza siempre por cero seguido de dígitos octales de cero a siete.
- **Hexadecimal.** Empieza siempre por 0x seguido de dígitos hexadecimales (de 0 a 9 y de la a/A a la f/F).

Ejemplo de código:

```

public static void main(String[] args) {
    int value;
    value = 16;
    System.out.println( "16 decimal = " + value) ;
    value = 020;
    System.out.println("20 octal = " + value + " en decimal.");
    value = 0x10;
    System.out.println( "10 hexadecimal = " + value + " en decimal". ) ;
}

```

Mostrará en pantalla:

16 decimal = 16

20 octal = 16 en decimal.

10 hexadecimal = 16 en decimal.

Las **constantes literales** de tipo long se construyen añadiendo detrás l/L, si no, se considera por defecto tipo int (se suele utilizar en mayúsculas para no confundir con un 1).

Los **literales reales** o en coma flotante se expresan con coma decimal o en notación científica. El valor por defecto será double (D), para indicar que es Float se debe finalizar con una F. Ejemplos:

- 13.2D es lo mismo que 1.32e1 y lo mismo que 0.132E2
- .54 / 31,21E-5, 2.f, 6.0222137e23f...

Un **literal carácter** puede escribirse como un carácter entre comillas simples o por su código en tabla unicode anteponiendo la secuencia escape (\) si ponemos el valor en octal o (\u si lo ponemos en hexadecimal).

Ejemplo:

tanto en ASCII como en unicode, la letra A es el 65: en octal es 101 y en hexadecimal es 41: \101 en octal y \u0041 en hexadecimal.

Secuencias de escape en java.

| Secuencia de escape | Significado | Secuencia de escape | Significado |
|---------------------|-----------------|---------------------|---------------------------|
| \b | Retroceso | \r | Retorno de carro |
| \t | Tabulador | \" | Carácter comillas dobles |
| \n | Salto de línea | \' | Carácter comillas simples |
| \f | Salto de página | \\ | Barra diagonal |

Los **literales de cadenas** de caracteres se indican entre comillas simples. Al construir cadenas de caracteres se puede incluir cualquier carácter unicode, excepto un carácter de retorno de carro.

```
String texto = "Juan dijo: \"Hoy hace un día fantástico...\"";
```

Los literales String no necesitan ser creados con la orden new, a pesar de ser objetos, ya que java los crea implícitamente.

5. Operadores y expresiones

Los operadores llevan a cabo operaciones sobre un conjunto de datos u operandos. Pueden ser unarios, binarios o terciarios, dependiendo del número de operandos que participen. Actúan sobre tipos de datos primitivos y devuelven datos primitivos.

Estos operadores se combinan con literales y/o identificadores para formar expresiones, que no es otra cosa que una combinación de operadores y operandos el cual después de evaluarse devolverá un único resultado de un tipo determinado.

Las expresiones combinadas con algunas palabras reservadas o por sí mismas forman sentencias o instrucciones: `i+1`; `sum = i+1`;

5.1. Operadores aritméticos

Operadores aritméticos básicos

| Operador | Operación Java | Expresión Java | Resultado |
|----------|------------------------------------|----------------|-----------|
| - | Operador unario de cambio de signo | -10 | -10 |
| + | Adición | 1.2 + 9.3 | 10.5 |
| - | Sustracción | 312.5 - 12.3 | 300.2 |
| * | Multiplicación | 1.7 * 1.2 | 1.02 |
| / | División (entera o real) | 0.5 / 0.2 | 2.5 |
| % | Resto de la división entera | 25 % 3 | 1 |

El resultado de expresiones depende de los operandos que intervengan:

| Tipo de los operandos | Resultado |
|---|---------------------|
| Un operando de tipo <code>long</code> y ninguno real (<code>float</code> o <code>double</code>) | <code>long</code> |
| Ningún operando de tipo <code>long</code> ni real (<code>float</code> o <code>double</code>) | <code>int</code> |
| Al menos un operando de tipo <code>double</code> | <code>double</code> |
| Al menos un operando de tipo <code>float</code> y ninguno <code>double</code> | <code>float</code> |

Operadores incrementales en Java

Son operadores unarios y podemos utilizarlos con notación prefija si aparece antes del operando o postfija si aparece después.

| Tipo operador | Expresión Java | |
|-------------------------|--|---|
| ++ (incremental) | Prefija: x=3; y=++x; // x vale 3 e y vale 4 | Postfija: x=3; y=x++; // x vale 4 e y vale 3 |
| --(decremental) | 5-- // el resultado es 4 | |

Ejemplo:

```

10 public class operadoresaritmeticos {
11     public static void main(String[] args) {
12         short x = 7;
13         int y = 5;
14         float f1 = 13.5f;
15         float f2 = 8f;
16         System.out.println("El valor de x es " + x + " y el valor de y es " + y);
17         System.out.println("El resultado de x + y es " + (x + y));
18         System.out.println("El resultado de x - y es " + (x - y));
19         System.out.printf("%s\n%s\n", "División entera:", "x / y = ", (x/y));
20         System.out.println("Resto de la división entera: x % y = " + (x % y));
21         System.out.printf("El valor de f1 es %f y el de f2 es %f\n", f1, f2);
22         System.out.println("El resultado de f1 / f2 es " + (f1 / f2));
23     } // fin de main
24 } // fin de la clase operadoresaritmeticos

```

5.2. Operadores de asignación

El principal operador de asignación es "=", pero existen algunos compuestos:

| Operador | Ejemplo en Java | Expresión equivalente |
|----------|-----------------|-----------------------|
| += | op1 += op2 | op1 = op1 + op2 |
| -= | op1 -= op2 | op1 = op1 - op2 |
| *= | op1 *= op2 | op1 = op1 * op2 |
| /= | op1 /= op2 | op1 = op1 / op2 |
| %= | op1 %= op2 | op1 = op1 % op2 |

5.3. Operador condicional

El operador condicional "?" evalúa una condición y devuelve un resultado en función de si es verdadera o falsa. Es el único operador ternario.

El primer operando es el que va a ser evaluado y debe ser una expresión booleana o condición. A continuación se pone el operador "?" y seguidamente las dos expresiones se paradas por ":", las cuales se devuelven dependiendo de si la condiciones verdadera (izquierda de los dos puntos) o falsa (derecha de los dos puntos).

| Operador | Expresión en Java |
|----------|-------------------------|
| ?: | condición ? exp1 : exp2 |

Por ejemplo: `z = (x>y) x:y;`

Si x es mayor que y se devuelve variable z almacena x y si no es mayor almacena y.

5.4. Operadores de relación

Los operadores relacionales se utilizan para comparar datos de tipo primitivo (numérico, carácter y booleano). El resultado se utiliza en otras expresiones o sentencias que se ejecutarán en función de si se cumple o no la relación.

| Operador | Ejemplo en Java | Significado |
|--------------------|----------------------------|---------------------------|
| <code>==</code> | <code>op1 == op2</code> | op1 igual a op2 |
| <code>!=</code> | <code>op1 != op2</code> | op1 distinto de op2 |
| <code>></code> | <code>op1 > op2</code> | op1 mayor que op2 |
| <code><</code> | <code>op1 < op2</code> | op1 menor que op2 |
| <code>>=</code> | <code>op1 >= op2</code> | op1 mayor o igual que op2 |
| <code><=</code> | <code>op1 <= op2</code> | op1 menor o igual que op2 |

Para introducir valores en nuestro programa se puede utilizar la clase **Scanner** la cual nos permite leer datos que se escriben por teclado. Para ello debemos importar el paquete de clases que la contiene.

El programa espera se queda esperando a que el usuario escriba y pulse la tecla intro. Seguidamente continua la ejecución del programa:

```
public class ejemplorelacionales {  
    // método principal que inicia la aplicación  
    public static void main( String args[] )  
    {  
        // clase Scanner para petición de datos  
        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  
        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);  
    } // fin método main  
} // fin clase ejemplorelacionales
```

5.5. Operadores lógicos

Realizan operaciones sobre valores booleanos o resultados de expresiones relacionales, dando como resultado un valor booleano.

En algunos casos el segundo operando de una expresión lógica no se evalúa ahorrando tiempo de ejecución, por ejemplo en la comparativa `a&& b`, si `a` es falso no se analizará `b` porque ya no se cumple. En el caso contrario `a|| b`, si `a` es verdadero, `b` no se analiza porque ya se cumple.

En estos casos es favorable colocar de primer operando el valor que sabemos que nos va a ahorrar tiempo de ejecución.

| Operador | Ejemplo en Java | Significado |
|----------|-----------------|---|
| ! | !op | Devuelve true si el operando es false y viceversa. |
| & | op1 & op2 | Devuelve true si op1 y op2 son true |
| | op1 op2 | Devuelve true si op1 u op2 son true |
| ^ | op1 ^ op2 | Devuelve true si sólo uno de los operandos es true |
| && | op1 && op2 | Igual que &, pero si op1 es false ya no se evalúa op2 |
| | op1 op2 | Igual que , pero si op1 es true ya no se evalúa op2 |

```

public class operadoreslogicos {
    public static void main(String[] args) {
        System.out.println("OPERADORES LÓGICOS");

        System.out.println("Negacion:\n ! false es : " + (! false));
        System.out.println(" ! true es : " + (! true));

        System.out.println("Operador AND (&):\n false & false es : " + (false & false));
        System.out.println(" false & true es : " + (false & true));
        System.out.println(" true & false es : " + (true & false));
        System.out.println(" true & true es : " + (true & true));

        System.out.println("Operador OR (|):\n false | false es : " + (false | false));
        System.out.println(" false | true es : " + (false | true));
        System.out.println(" true | false es : " + (true | false));
        System.out.println(" true | true es : " + (true | true));

        System.out.println("Operador OR Exclusivo (^):\n false ^ false es : " + (false ^ false));
        System.out.println(" false ^ true es : " + (false ^ true));
        System.out.println(" true ^ false es : " + (true ^ false));
        System.out.println(" true ^ true es : " + (true ^ true));

        System.out.println("Operador &&:\n false && false es : " + (false && false));
        System.out.println(" false && true es : " + (false && true));
        System.out.println(" true && false es : " + (true && false));
        System.out.println(" true && true es : " + (true && true));

        System.out.println("Operador ||:\n false || false es : " + (false || false));
        System.out.println(" false || true es : " + (false || true));
        System.out.println(" true || false es : " + (true || false));
        System.out.println(" true || true es : " + (true || true));
    } // fin main
} // fin operadoreslogicos

```

5.6. Operadores de bits

Realizan operaciones sobre números enteros o char en su representación binaria (sobre cada dígito binario)

| 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 |

5.7. Trabajo con cadenas

Para aplicar una operación a una variable de tipo String, se escribe su nombre seguido de la operación, separados por un punto. Estas son las operaciones que podemos realizar con String

- **Creación:** Simplemente asignando la cadena de caracteres entre comillas dobles.
- **Obtención de longitud:** método `length()`.
- **Concatenación:** se utiliza el operador `+` o el método `concat()`.
- **Comparación.** Se utiliza el método `equals()`, el cual devuelve un valor booleano. `equalsIgnoreCase()` hace lo mismo pero ignorando las mayúsculas.
- **Obtención de subcadenas.** Con el método `substring()`, indicándole el inicio y el fin de la subcadena a obtener.
- **Cambio a mayús/minus.** Con los métodos `toUpperCase()` y `toLowerCase()`.
- **Valueof.** Se utiliza para convertir un tipo de dato primitivo `int`, `long`, `float`... en variable de tipo String.

```

public class ejemplocadenas {
    public static void main(String[] args)
    {
        String cad1 = "CICLO DAM";
        String cad2 = "ciclo dam";

        System.out.printf( "La cadena cad1 es: %s y cad2 es: %s", cad1,cad2 );

        System.out.printf( "\nLongitud de cad1: %d", cad1.length() );

        // concatenación de cadenas (concat o bien operador +)
        System.out.printf( "\nConcatenación: %s", cad1.concat(cad2) );

        //comparación de cadenas
        System.out.printf("\ncad1.equals(cad2) es %b", cad1.equals(cad2) );
        System.out.printf("\ncad1.equalsIgnoreCase(cad2) es %b", cad1.equalsIgnoreCase(cad2) );
        System.out.printf("\ncad1.compareTo(cad2) es %d", cad1.compareTo(cad2) );

        //obtención de subcadenas
        System.out.printf("\ncad1.substring(0,5) es %s", cad1.substring(0,5) );

        //pasar a minúsculas
        System.out.printf("\ncad1.toLowerCase() es %s", cad1.toLowerCase() );

        System.out.println();
    } // fin main
} // fin ejemplocadenas

```

5.8. Precedencia de operadores

Las reglas de precedencia de operadores coinciden con las expresiones de álgebra convencional:

- Multiplicación, división y resto se evalúan primero.
- Suma y resta se aplican después de las anteriores.
- Si dentro de la misma operación existen varias operaciones de este tipo se evalúan de izquierda a derecha.

Cuando se evalúa una expresión es necesario tener en cuenta la asociatividad, la cual indica qué operador se evalúa antes, en condiciones de igualdad de precedencia.

| Operador | Tipo | Asociatividad |
|-------------------------|---------------------------|---------------|
| ++ -- | Unario, notación postfija | Derecha |
| ++ -- + - (cast) ! ~ | Unario, notación prefija | Derecha |
| * / % | Aritméticos | Izquierda |
| + - | Aritméticos | Izquierda |
| << >> >>> | Bits | Izquierda |
| < <= > >= | Relacionales | Izquierda |
| == != | Relacionales | Izquierda |
| & | Lógico, Bits | Izquierda |
| ^ | Lógico, Bits | Izquierda |
| | Lógico, Bits | Izquierda |
| && | Lógico | Izquierda |
| | Lógico | Izquierda |
| ?: | Operador condicional | Derecha |
| = += -= *= /= %= | Asignación | Derecha |

Ejemplos:

- En una operación de $10 / 2 * 5$, primero se realiza la división y después la multiplicación.
- En $x=y=z=1$; se comienza asignando el valor 1 a la variable z, luego a la y y por último a x. Además al contrario sería imposible de realizar.

6. Conversión de tipo

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

Existen dos tipos de conversiones:

- **Automáticas:** si a una variable de un tipo se le asigna un valor de otro tipo numérico con menos bits, se realiza conversión automática (valor promocionado). En conversiones automáticas en operaciones aritméticas, el valor más pequeño promociona al valor más grande, ya que el tipo mayor

siempre puede representar cualquier tipo del menor de int a long o de float a double).

- **Explícitas.** Cuando hacemos una conversión de un tipo con más bits a uno con menos. En estos casos debemos indicar la conversión de manera expresa. Se realiza con el operador **cast**. Por ejemplo byte b = (byte) a;. 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
```

6.1. Anexo Conversión de tipos

| | | Tipo destino | | | | | | | |
|-------------|---------|--------------|------|------|-------|-----|------|-------|--------|
| | | boolean | char | byte | short | int | long | float | double |
| Tipo origen | boolean | - | N | N | N | N | N | N | N |
| | char | N | - | C | C | CI | CI | CI | CI |
| | byte | N | C | - | CI | CI | CI | CI | CI |
| | short | N | C | C | - | CI | CI | CI | CI |
| | int | N | C | C | C | - | CI | CI* | CI |
| | long | N | C | C | C | C | - | CI* | CI* |
| | float | N | C | C | C | C | C | - | CI |
| | double | N | C | C | C | C | C | C | - |

N: Conversión no permitida.

CI: Conversión implícita o automática. Un asterisco indica que puede haber posible pérdida de datos.

C: Casting de tipos o conversión explícita.

Convertir tipos de coma flotante en tipos enteros implica utilizar siempre un casting, sabiendo que esto implica pérdida de datos.

Otras consideraciones con los tipos de datos

| Conversiones de números en Coma flotante (float, double) a enteros (int) | Conversiones entre caracteres (char) y enteros (int) | Conversiones de tipo con cadenas de caracteres (String) |
|---|---|---|
| <p>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:</p> <ul style="list-style-type: none">✓ Math.round(num): Redondeo al siguiente número entero.✓ Math.ceil(num): Mínimo entero que sea mayor o igual a num.✓ Math.floor(num): Entero mayor, que sea inferior o igual a num. <pre>double num=3.5; x=Math.round(num); // x = 4 y=Math.ceil(num); // y = 4 z=Math.floor(num); // z = 3</pre> | <p>Como un tipo <code>char</code> lo que guarda en realidad es el código Unicode de un carácter, los caracteres pueden ser considerados como números enteros sin signo.</p> <p>Ejemplo:</p> <pre>int num; char c; num = (int) 'A'; // num = 65 c = (char) 65; // c = 'A' c = (char) ((int) 'A' + 1); // c = 'B'</pre> | <p>Para convertir cadenas de texto a otros tipos de datos se utilizan las siguientes funciones:</p> <pre>num=Byte.parseByte(cad); num=Short.parseShort(cad); num=Integer.parseInt(cad); num=Long.parseLong(cad); num=Float.parseFloat(cad); num=Double.parseDouble(cad);</pre> <p>Por ejemplo, si hemos leído de teclado un número que está almacenado en una variable de tipo <code>String</code> llamada <code>cadena</code>, y lo queremos convertir al tipo de datos <code>byte</code>, haríamos lo siguiente:</p> <pre>byte n=Byte.parseByte(cadena);</pre> |

7. Comentarios

Todos los lenguajes disponen de alguna forma de introducir comentarios en el código. En java se encuentran los siguientes:

- **Comentarios de una sola línea:** `//Comentario`
- **Comentarios de múltiples líneas:** `/* Comentario */`
- **Comentarios Javadoc:** `/** Comentario. Se emplean para generar documentación automática del programa.`

Mapa Conceptual

