Desarrollo de aplicaciones móviles I

Tema Nº1:Introducción al lenguaje de programación Java.

Indicador de logro Nº1:Aplica los conceptos básicos del Lenguaje de Programación Java a través de casos prácticos

**TEMA 01 Teoría de los**

Imagen que contiene Icono

Descripción generada automáticamente

**TEMA Nº1:**

Introducción al lenguaje de programación Java.

**Subtema 1.1:**

Variables y tipos de datos en Java

La sintaxis 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.

### 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 dólar ( $ ).
* 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 |

### 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) Angel Franco García |

| fecha: Marzo 1999 |

| 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, una ventana DOS que se abre en el escritorio de Windows 95. La función *println* tiene un sólo argumento una cadena de caracteres u objeto de la [clase *String*.](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/string.htm)

### 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.

### 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.

### 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.

### 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;

//...

}

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=1999;**

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

### Tipos de datos primitivos

|  |  |
| --- | --- |
| **Tipo** | **Descripción** |
| **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 -27 hasta 27-1 (128 a 127) |

|  |  |
| --- | --- |
| **short** | Tamaño 16 bits. El intervalo de valores va desde -215 hasta 215-1 (- 32768 a 32767) |
| **int** | Tamaño 32 bits. El intervalo de valores va desde -231 hasta 231-1 (-2147483648 a 2147483647) |
| **long** | Tamaño 64 bits. El intervalo de valores va desde -263 hasta263-1 (-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 4.94065645841246544e–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 PI el segundo es la aceleración de la gravedad nivel del mar, el tercero es la velocidad de la luz en m/s, que es la forma de escribir 2.9979 108. 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 arbitararia 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;

### 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 en el siguiente capítulo ([Clases y](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/clases.htm#Los%20objetos) [objetos](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/clases.htm#Los%20objetos)).

Las [cadenas de caracteres o strings](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/clases1/string.htm) 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";

### 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](http://www.sc.ehu.es/sbweb/fisica/cursoJava/fundamentos/introduccion/#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 | by value\* |
| Case | cast\* | catch | char | class |
| const\* | continue | default | do | double |
| Else | extends | false | final | finally |
| Float | for | future\* | generic\* | goto\* |
| If | implements | import | inner\* | instance of |
| Int | interface | long | native | new |
| Null | operator\* | outer\* | package | private |
| Protected | public | rest\* | return | short |
| Static | 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**.

**Subtema 1.2:**

Conversiones en Java.

**Ejemplos:**

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 cuyo tamaño 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.**

**Subtema 1.3:**

Constantes en Java.

**Ejemplos:**

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;

**Subtema 1.4:**

Salida de datos en Java.

**Ejemplos:**

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;

}

}

**Subtema 1.5:**

Entrada de datos en Java.

**Ejemplos:**

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 escribiremos a/(b\*c); 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



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);

}

}

**Subtema 1.6:**

Operadores matemáticos de Java.

**Ejemplos:**

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.

### 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

|  |  |
| --- | --- |
| **Expresión** | **Significado** |
| x+=y | x=x+y |
| x-=y | x=x-y |
| x\*=y | x=x\*y |
| x/=y | x=x/y |

Así, la sentencia

x=x+23;

evalúa la expresión *x+23*, que es asignada de nuevo a *x*. El compilador lee primero el contenido de la porción de memoria nombrada *x*, realiza la suma, y guarda el resultado en la misma porción de memoria. Se puede escribir la sentencia anterior de una forma equivalente más simple

x+=23;

**Subtema 1.7:**

Operadores relacionales de Java.

**Ejemplos:**

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);

}

}

**Subtema 1.8:**

Operadores lógicos de Java.

**Ejemplos:**

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) Ejemplo:

package relacion;

public class RelacionApp {

public static void main(String[] args) { int x=8;

int y=5;

boolean compara=(xy);

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);

try {

//espera la pulsación de una tecla y luego RETORNO System.in.read();

}catch (Exception e) { }

}

}

### Prioridad y Orden de evaluación

|  |  |
| --- | --- |
| **Operador** | **Acción** |
| ( ) [ ] . | Paréntesis, corchetes y el punto |
|   ++ -- | Cambio signo, Negación, incremento, decremento |
| new (cast) | Asignación de memoria, cambio de tipo |
| \* % | División, multiplicación, resto de una división |
| + - | Suma y resta |
| < > <= >= | Menor que, mayor que, menor o igual, mayor o igual |
| == = | Igual a, diferente de |
| && | Y lógico |
| ǁ | O lógico |
| = \*= /= %=  += -= | Igual, \* igual, División igual, resto igual, suma igual resta  igual |

**Actividad:**

a) CUESTIONARIO TÉCNICO

* Indique cuáles son los operadores lógicos en Java
* ¿Cuál es la utilidad de los operadores unarios?
* Cite un ejemplo práctico de cuándo utilizar la conversión de tipo automática y promoción (casting)
* Mencione cuál es la diferencia entre una variable y una constante
* Proponga un ejemplo donde se evidencie la utilidad del operador aritmético Módulo

b) CONCLUSIONES DE LA EXPERIENCIA

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_