

PROGRAMACIÓN (PRG). EUI. Curso 2001/2002
Tema 3. CONDICIONES Y ELECCIÓN

F. Marqués y N. Prieto

Índice General

1	Condiciones simples	1
2	El tipo lógico	2
2.1	Operadores de comparación	3
2.2	Comparación de cadenas (strings)	3
2.3	Operadores lógicos	4
2.4	Precedencia de operadores	5
3	Instrucciones condicionales	5
3.1	Instrucción " if...else... "	6
3.2	Instrucción " switch... "	7
4	El operador ternario	8
5	Tratamiento de errores. Introducción a las <i>excepciones</i>	9
5.1	<i>Excepciones</i>	10
5.2	Tratamiento básico de errores	10
6	Ejercicios	11

1 Condiciones simples

Una característica de los programas que se han visto hasta el momento es que consisten tan solo en una secuencia de instrucciones que se ejecuta inalterablemente. Sin embargo, en la resolución de un problema es a menudo necesario tomar decisiones en función de las características del mismo, esto es, se necesita poder alterar la secuencia de cálculos que se efectúan en función de los datos de entrada, o de resultados obtenidos con antelación. Las instrucciones condicionales, que se estudiarán seguidamente, permiten construir programas que tomen decisiones acerca de los cálculos a efectuar en función de las características del problema.

Supóngase, por ejemplo, que se desea determinar si cierta variable `temperatura` tiene un valor mayor o menor que 100 (escribiendo un mensaje según lo sea o no). Esto podría expresarse mediante las instrucciones:

```
if (temperatura>100)
    pantalla.writeln("La temperatura es mayor que 100!");
else pantalla.writeln("La temperatura es menor o igual que 100");
    .....
```

Para ejecutar esta instrucción se determina, en primer lugar, la verdad o falsedad de la condición. Si la variable `temperatura` tiene un valor mayor que 100 (la condición es cierta), entonces se escribirá el primer mensaje; si no es así, esto es si la variable tiene un valor menor o igual que 100 (la condición es falsa) se escribirá el segundo mensaje. Una vez escrito el mensaje correspondiente, la ejecución del programa continuará en la instrucción siguiente (que en el ejemplo aparece indicada mediante puntos suspensivos).

Tanto después de la condición, como después de la palabra `else`, es posible escribir cualquier instrucción, por ejemplo otra instrucción condicional. Así, en el ejemplo siguiente se clasifica más aun el valor de la variable `temperatura`:

```
if (temperatura>100)
    pantalla.writeln("Mayor que 100!");
else if (temperatura>50)
    pantalla.writeln("Mayor que 50, menor o igual que 100");
    else pantalla.writeln("Menor o igual que 50");
    .....
```

A continuación se examinarán con más profundidad algunas características relativas a las condiciones, y al agrupamiento de las instrucciones, con ello se podrá estudiar la forma general de las instrucciones condicionales, así como distintas variaciones de las mismas.

2 El tipo lógico

En el tema primero se han introducido algunos tipos de datos elementales que, como se ha visto, definen las características principales de variables y expresiones; al decir que un operando, por ejemplo, es de cierto tipo se están definiendo implícitamente las operaciones en las que puede intervenir dicho operando. Hasta el momento se han visto tipos numéricos y alfabéticos, se examinará a continuación el tipo asociado a las condiciones.

La condición anterior:

```
temperatura>100
```

adquiere al ser evaluada (del mismo modo que cualquier otra condición) uno de los dos posibles valores lógicos: *cierto* o *falso* que se denotarán en los programas como: `true` o `false`.

Se denomina expresión lógica, a cualquier expresión, secuencia de operandos y operadores, que al ser evaluada tiene uno de los valores: `true` o `false`. Naturalmente, cualquier condición es una expresión lógica.

Toda expresión lógica tiene además como tipo de datos el tipo *lógico* que se denotará en los programas como: `boolean`.

Como ocurre con los otros tipos ya vistos, es posible declarar variables como pertenecientes al tipo, así se podría reescribir las instrucciones iniciales del modo siguiente:

```
boolean alta;
.....
.....
alta = temperatura>100;
if (alta)
    pantalla.writeln("La temperatura es mayor que 100!");
else pantalla.writeln("La temperatura es menor o igual que 100");
.....
```

Al ejecutarse la primera instrucción se asigna a la variable `alta` el resultado de la evaluación de la expresión que hay a la derecha en la asignación. En función del valor lógico de dicha variable se escribirá uno u otro mensaje.

Los dos únicos valores constantes que puede tener cualquier expresión de tipo `boolean` son los ya mencionados: `true` o `false`. Es posible en los programas hacer uso explícito de dichos valores.

2.1 Operadores de comparación

Los *operadores de comparación*, también denominados *operadores relacionales*, se utilizan para comparar entre sí valores de elementos que tienen el mismo *tipo simple*. Son tipos simples los tipos numéricos ya vistos, junto con el tipo `char` y el `boolean`. Los operadores, y su significado se listan a continuación:

OPERADOR	NOMBRE OPERACIÓN	EJEMPLO
<	menor	<code>temperatura < 100</code>
>	mayor	<code>minuto > 10</code>
<=	menor o igual	<code>hora <= 24</code>
>=	mayor o igual	<code>dia >= 1</code>
==	igual	<code>mes == 12</code>
!=	distinto	<code>temperatura != 100</code>

2.2 Comparación de cadenas (strings)

La comparación entre dos cadenas (`strings`) se realiza en Java de un modo distinto a como se efectúa la comparación entre valores pertenecientes a tipos elementales. Una `string` es en Java un objeto, por lo que las operaciones de comparación se efectúan mediante la *notación de punto*, que se estudiará más adelante.

Considérese, por ejemplo, el problema de solicitar un código de asignatura al usuario, para así comprobar si éste es o no el de una asignatura de primer curso, dando información sobre las que sí que lo son, y rechazando las que no lo sean. Un segmento de la posible solución sería:

```
String asig = teclado.readline();

if (asig.equals("PRG"))
    out.writeln("Si, es anual");
else if (asig.equals("MAD"))
    pantalla.writeln("Si, semestre A");
else if (asig.equals("AMA"))
    pantalla.writeln("Si, semestre B");
else if ...
.....
else if (asig.equals("MAD"))
    pantalla.writeln("Si, semestre A");
else pantalla.writeln("No es de primer curso");
```

Del mismo modo que es posible comparar dos `strings` para determinar si son o no iguales, es también posible realizar la comparación para determinar su *orden lexicográfico*. Este último depende del juego de caracteres subyacente, *UNICODE* en el caso del Java.

2.3 Operadores lógicos

Es posible construir una nueva expresión lógica yuxtaponiendo otras expresiones lógicas, ligadas entre sí mediante las denominadas *conectivas lógicas*.

Así, por ejemplo, se podría determinar si la variable temperatura antes considerada tiene un valor comprendido entre 15 y 20 grados, ambos inclusive, mediante las instrucciones siguientes:

```
if ((temperatura>=15) && (temperatura<=20)) ....
.....
```

Para que la condición anterior evalúe a `true` es necesario que el valor de la variable `temperatura` sea mayor o igual que 15 y, al mismo tiempo, menor o igual que 20, de lo contrario evaluará a `false`.

Además de la conectiva del ejemplo anterior para expresar la *conjunción o y-lógico* (`&&`), existen también conectivas para expresar la *disyunción o o-lógico* (`||`), la *o-exclusiva* (`^`) y la *negación* (`!`).

Por ejemplo, se puede determinar si la `temperatura` es mayor o igual que 15, mediante la instrucción:

```
if ((temperatura>15) || (temperatura=15)) ....
// similar a (temperatura >= 15)
.....
```

O determinar si el valor de `temperatura` no está comprendida entre 0 y 100 mediante:

```

if ((temperatura>=0) ^ (temperatura<=100)) ....
//similar a (temperatura>100) || (temperatura<0)
.....

```

En realidad, el lenguaje Java proporciona dos operadores diferentes para cada una de las conectivas. Los denominados *cortocircuitados* que tienen la particularidad de que en las subexpresiones donde aparecen se evalúa, siguiendo el recorrido de izquierda a derecha, la mínima parte de la subexpresión necesaria para conocer el valor lógico. Sin embargo, cuando se utiliza un operador no cortocircuitado se produce siempre la evaluación de toda la subexpresión donde éste aparece. En Java, los operadores no cortocircuitados para la *conjunción* y la *disyunción* son respectivamente: `&` y `|`.

A continuación se muestra las tablas de verdad de los operadores lógicos vistos:

x	y	x && y x & y	x y x y	x ^ y	!x
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

2.4 Precedencia de operadores

La siguiente tabla muestra la precedencia relativa de todos los operadores vistos hasta el momento (junto con el operador ternario que se verá al final de este tema). Recuérdese que las expresiones se evalúan, por defecto, de izquierda a derecha.

```

grupo 0: ( )
grupo 1: ++ -- +(unario) -(unario) !
grupo 2: * / %
grupo 3: + -
grupo 5: > >= < <=
grupo 6: == !=
grupo 7: &
grupo 8: ^
grupo 9: |
grupo 10: &&
grupo 11: ||
grupo 12: ?: (operador ternario)
grupo 13: = op= (op es uno de: +,-,*,/,%,&,|,^)

```

3 Instrucciones condicionales

A continuación se examina la forma general que tienen las distintas instrucciones condicionales. Se utilizará la letra **B** para denotar una condición y la **S** para denotar una

instrucción (o bloque) cualquiera. Si existe más de una, de alguna de ellas, se hará uso de índices.

3.1 Instrucción "if...else..."

La forma más simple de una instrucción condicional es la siguiente:

```
if (B) S
```

donde **S** es una instrucción (por ej., una asignación seguida de ";") o bloque cualquiera. Su ejecución se efectúa evaluando la condición **B**, de forma que si es cierta se ejecuta **S**. Al acabar la ejecución de la instrucción (o bloque), ésta continua en la instrucción siguiente a la condicional.

La otra forma de instrucción condicional, que ya se ha visto a lo largo de los ejemplos anteriores, es:

```
if (B) S1 else S2
```

donde **S1** y **S2** son instrucciones simples (o bloques) cualesquiera. Su ejecución se efectúa evaluando la condición **B**, de forma que si ésta es cierta se ejecuta **S1**, por lo contrario, si no es cierta, se ejecuta **S2**. Al acabar la instrucción condicional la ejecución continúa en la instrucción siguiente.

El siguiente programa pide tres valores numéricos al usuario y los escribe ordenadamente de mayor a menor (incluso si hay varios iguales):

```
import nsIO.*;
class clasifica {
    public static void main(String args[]) {

        output pantalla = new output();
        input teclado = new input();
        int a,b,c;    // valores iniciales
        int max,med,min;    // valores clasificados

        pantalla.writeln("Valores a clasificar?:");
        a=teclado.readInt();
        b=teclado.readInt();
        c=teclado.readInt();

        if (a >= b)
            if (b >= c) {max=a; med=b; min=c;}
            else if (a>=c) {max=a; med=c; min=b;}
            else {max=c; med=a; min=b;}
    }
}
```

```

else if (a >= c) {max=b; med=a; min=c;}
    else if (b >= c) {max=b; med=c; min=a;}
        else {max=c; med=b; min=a;}

pantalla.writeln("máximo: "+max);
pantalla.writeln("medio: "+med);
pantalla.writeln("mínimo: "+min);

pantalla.close();
teclado.close();
}
}

```

3.2 Instrucción "switch..."

En la resolución de muchos problemas surge la necesidad de efectuar cálculos diferentes para distintos valores de una única variable o expresión simple. Si, por ejemplo, alguna variable de entrada tiene 10 posibles valores distintos, y para cada uno de los mismos es necesario un tratamiento especial, se podría utilizar una instrucción "if...else..." anidada; sin embargo ésta suele ser difícil de leer y mantener, por ello el Java introduce una instrucción condicional adicional con el objetivo de facilitar el tratamiento de casos como el anterior. Es la instrucción "switch ...", que tiene la siguiente forma general:

```

switch (expresion) {
    case val1: [SC1] [break;]
    case val2: [SC2] [break;]
    .....
    .....
    case valn: [SCn] [break;]
    [default: [SCn+1] ]
}

```

Donde, en primer lugar, hay que notar que aquellas componentes que aparecen entre corchetes indican opcionalidad (pueden o no aparecer), así por ejemplo, la parte etiquetada como `default` puede aparecer o no. Además, `expresion` es una expresión de un tipo simple cualquiera. La serie de valores `val1`, `val2`, ... `valn`, son todos valores compatibles con el de `expresion`. `SC1`, `SC2`, ... `SCn+1`, son instrucciones (o secuencias de instrucciones) cualesquiera. Nótese que tanto dichas instrucciones (o secuencias) como la instrucción `break` son opcionales.

La ejecución de una instrucción así es como sigue: se evalúa en primer lugar la *expresion*, comparándose el valor resultante con el de cada uno de los valores asociados a las etiquetas `case`, si coincide con alguno entonces se ejecuta todo el código que sigue a la etiqueta `case` correspondiente, incluso el asociado a las etiquetas `case` posteriores, hasta que, o bien se finaliza toda la instrucción `switch`, o bien se encuentra una instrucción

break, en cuyo caso toda la instrucción **switch** se acaba inmediatamente. Si ninguno de los valores coincide con el de la **expresión** entonces se ejecuta, si existe, la instrucción o bloque asociado a la etiqueta **default**. Toda vez finalizada la ejecución del **switch** el programa continúa a partir de la instrucción subsiguiente.

Véase, por ejemplo, su uso para determinar, dado un mes representado como un entero, el número de días totales de dicho mes. Se supone que la variable **bisiesto** es de tipo **boolean** y está debidamente inicializada.

```
{
boolean bisiesto; int mes, num;
.....
.....
if ((mes<1) || (mes>12))
    out.writeln("Mes no válido");
else
switch (mes) {
    case 2: if (bisiesto) num = 29;
            else num = 28; break;
    case 4: //continuar
    case 6: //continuar
    case 9: //continuar
    case 11: num = 30;
            break;
    default: num = 31;
}
}
```

4 El operador ternario

El lenguaje Java (al igual que los lenguajes C y C++) introduce un operador especial, ternario, con un uso parecido al de una instrucción condicional. La forma general de una expresión en la que aparece dicho operador es:

```
exprbool ? expr1 : expr2
```

Donde **exprbool** es cualquier expresión de tipo **boolean** y **expr1** y **expr2** son expresiones cualesquiera del mismo tipo. Recuérdese que lo que se presenta es un operador y no una instrucción y que un operador sólo puede aparecer formando parte de una expresión y no tiene sentido de forma aislada. La evaluación de toda la expresión anterior se efectúa del modo siguiente: en primer lugar se evalúa la **exprbool** que se encuentra a la izquierda del interrogante. Si el resultado de dicha evaluación es **true** entonces el valor de toda la expresión es el valor de la **expr1**, en caso contrario, el valor de toda la expresión es el valor de la **expr2**.

Por ejemplo, la siguiente instrucción, en la que interviene el operador ternario, asigna a la variable **max** el valor mayor de entre los de **a** y **b**.

```
int a,b, max;
.....
max = a>b ? a : b;
.....
```

En el siguiente ejemplo se utiliza el operador ternario para redefinir la suma de dos números enteros a y b:

```
int a, b, suma_ab;
.....
suma_ab = (a==2 && b==2) ? 5 : a+b;
.....
```

5 Tratamiento de errores. Introducción a las *excepciones*

Frecuentemente existen secuencias de instrucciones que pueden provocar efectos no deseados, o incluso errores de ejecución debido a que los datos a ser tratados no cumplen las condiciones necesarias para que las instrucciones actúen correctamente.

Ejemplos habituales de este tipo de problemas se encuentran fácilmente en el ámbito de las operaciones matemáticas. La instrucción siguiente:

```
double frecuencia, período;
// inicialización de las variables
...
frecuencia = 1/período;
...
```

tiene sólo sentido si la variable `período` tiene valor distinto de cero cuando se intente efectuar la división. Ejemplos parecidos en el mismo ámbito pueden darse cuando se intenten efectuar operaciones como raíces cuadradas, cálculo de logaritmos, algunas funciones trigonométricas, etc.

Naturalmente, errores similares a los comentados, pueden provocarse en ámbitos no matemáticos, por ejemplo al tratar de manipular un objeto que no existe o cuando se intenta efectuar una operación con parámetros no válidos (acceso a una dirección remota inexistente, escritura en un fichero erróneo, no inicializado, etc). O incluso, aún peor, puede ocurrir que no llegue a producirse un error pero si un funcionamiento inapropiado del segmento en ejecución provocándose, tal vez, un comportamiento anómalo o errático.

Por ejemplo, considérese:

```
...
input teclado = new input();
output pantalla = new output();
char c;
...
```

```

pantalla.write(">Desea seguir?, introduzca S o N: ");
c = teclado.read();
if (c=='S' || c=='s') pantalla.writeln("Has escrito que SI");
else pantalla.writeln("Has escrito que NO");
...
// cuidado, lo que escribe el programa no siempre es cierto!!!.

```

El lenguaje Java introduce un nuevo término: *excepción* para definir y también poder tratar errores de ejecución.

5.1 Excepciones

Una *excepción* es una condición anormal que ocurre durante la ejecución de un segmento de código. Siendo excepciones frecuentes, como ya se ha dicho, las de índole matemático, como son la división por cero, o el intento de obtener la raíz cuadrada de un número negativo, otros tipos de excepción serán tratadas en capítulos posteriores. Al ocurrir una excepción la ejecución de la secuencia de código se detiene dando un error en tiempo de ejecución.

El siguiente ejemplo muestra un programa donde se produce una excepción durante su ejecución.

```

class ejem1 {
    public static void main(String args[]) {
        int den=0;
        int res = 100/den;
    }
}

```

Al ser ejecutado el programa anterior se obtendría lo siguiente

```

java ejem1
java.lang.ArithmeticException: / by zero
    at ejem1.main(ejem1.java:4)

```

El lenguaje Java permite un tratamiento especial de las excepciones, de forma que no supongan siempre la finalización del programa, sino la ejecución de un código de tratamiento diseñado por el programador. Este tipo de gestión se estudiará en temas posteriores, donde se ampliará el concepto de *excepción*, así como su posible tratamiento.

5.2 Tratamiento básico de errores

Para evitar las excepciones, los errores, y en general los posibles funcionamientos incorrectos (tal como todos los señalados anteriormente) será necesario garantizar que cualquier secuencia de instrucciones se ejecute siempre bajo las condiciones debidas para su correcta actuación. Lo que se podrá conseguir haciendo uso de las instrucciones condicionales estudiadas; así, en cada uno de los casos anteriores se podría escribir:

```

double frecuencia, período;
// inicialización de las variables
...
if (período!=0) frecuencia = 1/período;
    else pantalla.writeln("Valor de período incorrecto (0)");
...

    O bien,

...
input teclado = new input();
output pantalla = new output();
char c;
...
pantalla.write(">Desea seguir?, introduzca S o N: ");
c = teclado.read();
if (c=='S' || c=='s') pantalla.writeln("Has escrito que SI");
else if (c=='N' || c=='n') pantalla.writeln("Has escrito que NO");
    else pantalla.writeln("No has introducido bien los datos");
...
// ahora, lo que escribe el programa es cierto!!!.

```

6 Ejercicios

1. Dados dos números enteros, `num1` y `num2`, realizar un programa que escriba uno de los dos mensajes: "el producto de los dos números es positivo o nulo" o bien "el producto de los dos números es negativo". No hay que calcular el producto.
2. Considérese el texto siguiente: "En una empresa el cálculo de las vacaciones pagadas se efectúa de la manera siguiente: si una persona lleva menos de un año en la empresa, tiene derecho a dos días por cada mes de presencia, si no, al menos a 28 días. Si es un directivo y si tiene menos de 35 años y si su antigüedad es superior a 3 años, obtiene 2 días suplementarios. Si tiene menos de 45 años y si su antigüedad es superior a los 5 años, se le conceden 4 días suplementarios.
 - Reformular el texto suprimiendo toda ambigüedad.
 - Escribir el programa correspondiente, considerando que los datos son: la antigüedad expresada en meses `ant`, la edad en años `edad`, y la condición de ser o no directivo `dir`. El resultado es el número de días de vacaciones pagadas.
 - Completar el programa introduciendo verificaciones sobre la coherencia de los datos. Por ejemplo, la edad ha de ser inferior a 65 años y superior a 18 años, etc.

3. Considérese el juego siguiente: un jugador (jug. A) elige un número entre 1 y 16. Otro jugador (jug. B) puede hacer cuatro preguntas de la forma: ¿es "13" el número?, a lo que el primer jugador (jug. A) responderá diciendo: **igual**, **mayor**, o **menor**, según el número propuesto sea igual menor o mayor que el elegido.
 - Estudiar una estrategia ganadora para el juego.
 - Realizar un programa en que el usuario sea el jugador A y el ordenador el B.
4. Se desea escribir un programa para calcular la raíz cuadrada real de un número real cualquiera pedido inicialmente al usuario. Como dicha operación no está definida para los números negativos es necesario tratar, de algún modo, dicho posible error sin que el programa detenga su ejecución. Escribanse distintas soluciones al problema anterior, haciendo uso de: operadores cortocircuitados, instrucciones condicionales, el operador ternario, etc.
5. Escribese un programa para simular una calculadora. Considérese que los cálculos posibles son del tipo `num1 operador num2`, donde `num1` y `num2` son dos números reales cualesquiera y `operador` es uno de entre: `+`, `-`, `*`, `/`. El programa pedirá al usuario en primer lugar el valor `num1`, a continuación el `operador` y finalmente el valor `num2`. Resuélvase utilizando tanto instrucciones `"if...else"`, como `"switch"`.