Estructuras de control condicionales

Por defecto,
las instrucciones de un programa se ejecutan secuencialmente:

> El orden secuencial de ejecución no altera el flujo de control del programa respecto al orden de escritura de las instrucciones.

Sin embargo, al describir la resolución de un problema, es normal que tengamos que tener en cuenta condiciones que influyen sobre la secuencia de pasos que hay que dar para resolver el problema:

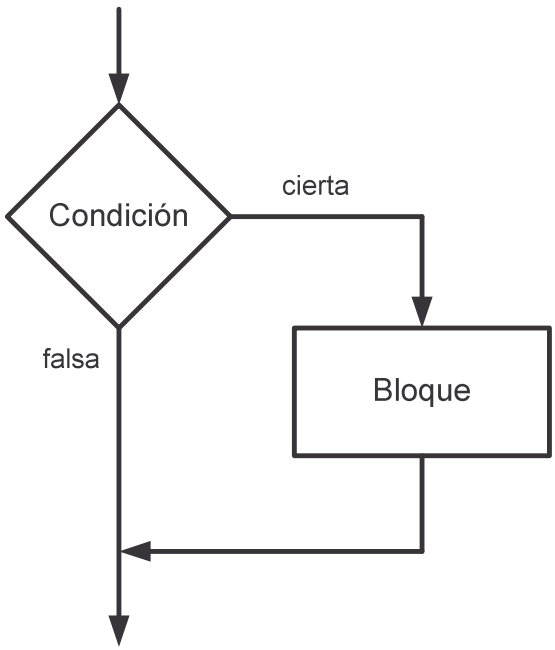
> Según se cumplan o no determinadas condiciones,
> la secuencia de pasos involucrada en la realización de una tarea será diferente

Las estructuras de control condicionales o selectivas nos permiten decidir qué ejecutar y qué no en un programa.

Ejemplo típico
Realizar una división sólo si el divisor es distinto de cero.

La estructura de control condicional `if`

La sentencia `if` nos permite elegir
si se ejecuta o no un bloque de instrucciones.



Sintaxis

```
if (condición)
   sentencia;


if (condición) {
   bloque
}
```

donde `bloque` representa un bloque de instrucciones.

Bloque de instrucciones: Secuencia de instrucciones encerradas entre dos llaves {....}

Consideraciones acerca del uso de la sentencia `if`

- Olvidar los paréntesis al poner la condición del `if` es un error sintáctico (los paréntesis son necesarios)

- No hay que confundir el operador de comparación ==
  con el operador de asignación =

- Los operadores de comparación ==, `!`=, <= y >= han de escribirse sin espacios.

- => y =< no son operadores válidos en Java.

- El fragmento de código afectado por la condición del `if` debe sangrarse para que visualmente se interprete correctamente el ámbito de la sentencia `if`:

```
if (condición) {
   // Aquí se incluye el código
   // que ha de ejecutarse
   // cuando se cumple la condición del if
}
```

- Aunque el uso de llaves no sea obligatorio cuando el `if` sólo afecta a una sentencia, es recomendable ponerlas siempre para delimitar explícitamente el ámbito de la sentencia `if`.

Error común:

```
if (condición);
   sentencia;
```

es interpretado como

```
if (condición)
   ;   // Sentencia vacía
sentencia;
```

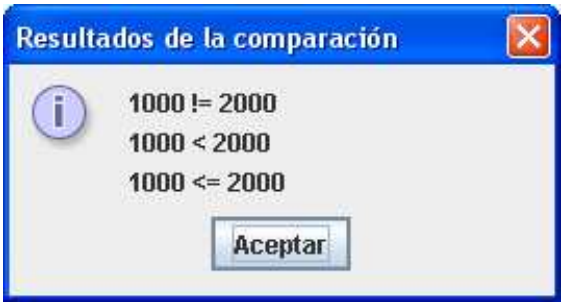
¡¡¡La sentencia siempre se ejecutaría!!!

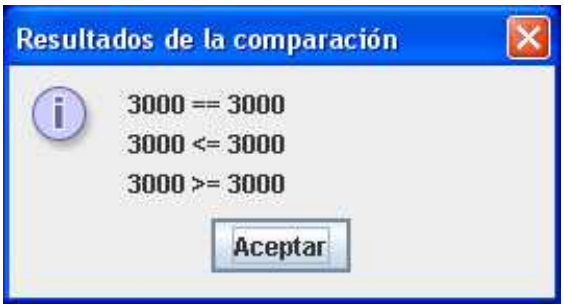
Ejemplo
Comparación de números (Deitel & Deitel)

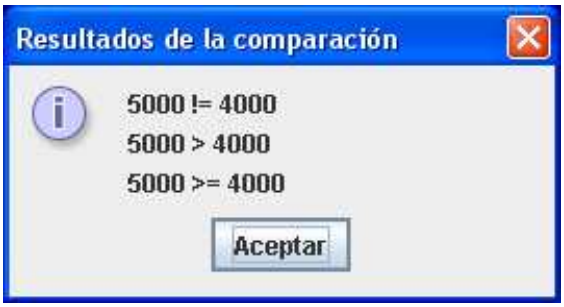
Operador	Significado
==	Igual
!=	Distinto
<	Menor
>	Mayor
<=	Menor o igual
>=	Mayor o igual

```
import javax.swing.JOptionPane;

public class Comparison
{
  public static void main( String args[] )
  {
    // Declaración de variables

    String primerDato, segundoDato;
    String resultado;
    int    dato1, dato2;


    primerDato  = JOptionPane.showInputDialog
                  ( "Primer dato:" );
    segundoDato = JOptionPane.showInputDialog
                  ( "Segundo dato:" );

    dato1 = Integer.parseInt( primerDato );
    dato2 = Integer.parseInt( segundoDato );

    resultado = "";

    if ( dato1 == dato2 )
       resultado += dato1 + " == " + dato2;

    if ( dato1 != dato2 )
       resultado += dato1 + " != " + dato2;
```

```
      if ( dato1 < dato2 )
         resultado += "\n" + dato1 + " < " + dato2;

      if ( dato1 > dato2 )
         resultado += "\n" + dato1 + " > " + dato2;

      if ( dato1 <= dato2 )
         resultado += "\n" + dato1 + " <= " + dato2;

      if ( dato1 >= dato2 )
         resultado += "\n" + dato1 + " >= " + dato2;

      JOptionPane.showMessageDialog
        ( null, resultado,
          "Resultados de la comparación",
          JOptionPane.INFORMATION_MESSAGE );

      System.exit( 0 );
   }
}
```

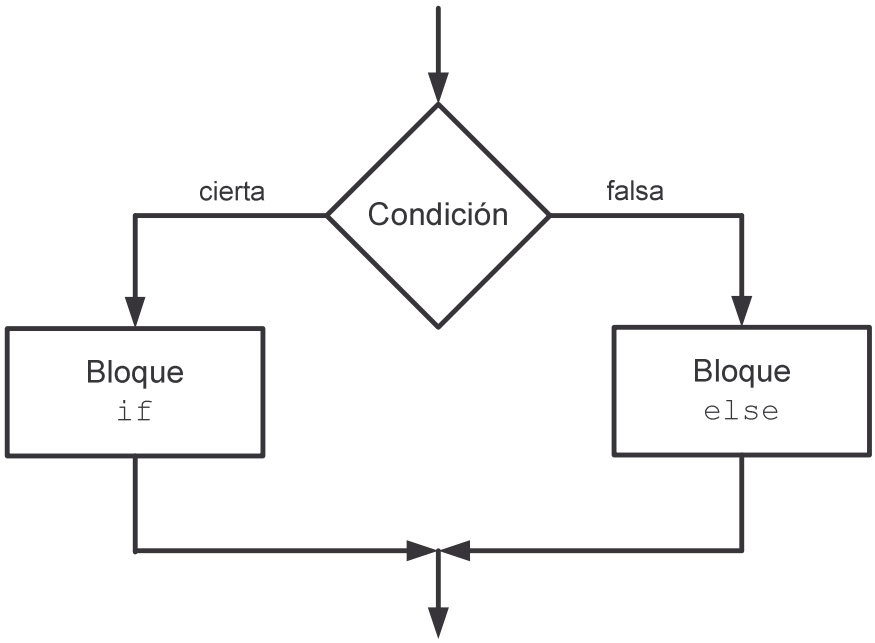






La cláusula `else`

Una sentencia `if`, cuando incluye la cláusula `else`, permite ejecutar un bloque de código si se cumple la condición y otro bloque de código diferente si la condición no se cumple.



Sintaxis

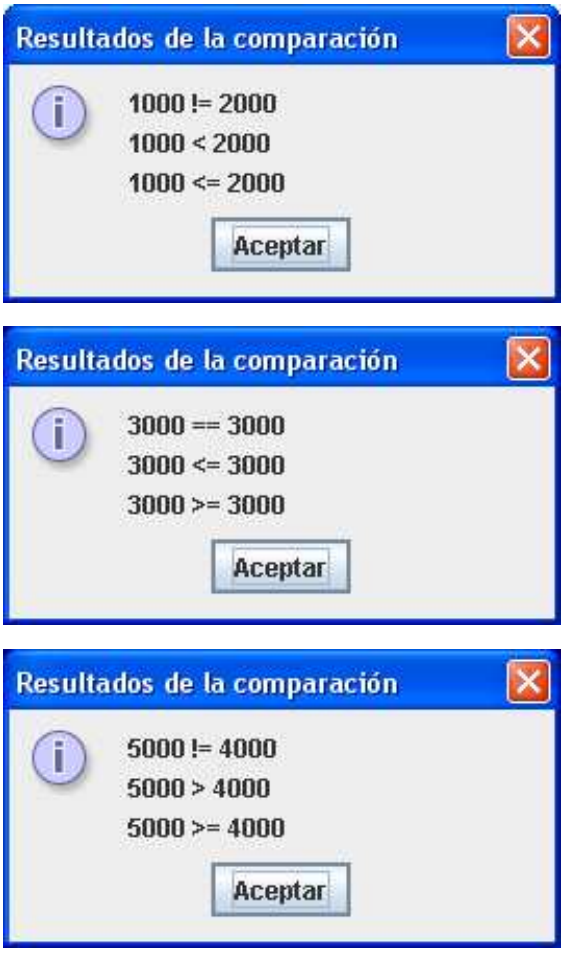
```
if (condición)
   sentencia1;
else
   sentencia2;


if (condición) {
   bloque1
} else {
   bloque2
}
```

Los bloques de código especificados
representan dos alternativas complementarias y excluyentes

Ejemplo

```
import javax.swing.JOptionPane;

public class IfElse
{
  public static void main( String args[] )
  {
    String primerDato, segundoDato;
    String resultado;
    int    dato1, dato2;

    primerDato  = JOptionPane.showInputDialog
                  ( "Primer dato:" );
    segundoDato = JOptionPane.showInputDialog
                  ( "Segundo dato:" );

    dato1 = Integer.parseInt( primerDato );
    dato2 = Integer.parseInt( segundoDato );

    resultado = "";

    if ( dato1 == dato2 )
       resultado += dato1 + " == " + dato2;
    else
       resultado += dato1 + " != " + dato2;

    if ( dato1 < dato2 )
       resultado += "\n" + dato1 + " < " + dato2;
    else
       resultado += "\n" + dato1 + " >= " + dato2;

    if ( dato1 > dato2 )
       resultado += "\n" + dato1 + " > " + dato2;
    else
       resultado += "\n" + dato1 + " <= " + dato2;

    JOptionPane.showMessageDialog
      ( null, resultado,
        "Resultados de la comparación",
        JOptionPane.INFORMATION_MESSAGE );

    System.exit( 0 );
  }
}
```

Con esta versión del programa de comparación de números obtenemos los mismos resultados que antes, si bien nos ahorramos tres comparaciones:



La sentencia

```
if (condición)
   sentencia1;
else
   sentencia2;
```

es equivalente a

```
if (condición)
   sentencia1;

if (!condición)
   sentencia2;
```

Nota acerca de la comparación de objetos

Cuando el operador == se utiliza para comparar objetos,
lo que se compara son las referencias a los objetos
y no el estado de los objetos en sí.

Ejemplo

```
public class Complex
{
  private double real;
  private double imag;

  // Constructor

  public Complex (double real, double imag)
  {
    this.real = real;
    this.imag = imag;
  }

  // equals se suele definir para comparar objetos

  public boolean equals (Complex c)
  {
    return (this.real == c.real)
        && (this.imag == c.imag);
  }
}

...
Complex c1 = new Complex(2,1);
Complex c2 = new Complex(2,1);

if (c1 == c2)
   System.out.println("Las referencias son iguales");
else
   System.out.println("Las referencias no son iguales");

if (c1.equals(c2))
   System.out.println("Los objetos son iguales");
else
   System.out.println("Los objetos no son iguales");
```

Encadenamiento

Las sentencias `if` se suelen encadenar:

if … else if …

```
import javax.swing.JOptionPane;

public class IfChain
{
  public static void main( String args[] )
  {
    String entrada;
    String resultado;
    float  nota;


    entrada = JOptionPane.showInputDialog
              ( "Calificación numérica:" );

    nota = Float.parseFloat( entrada );


    if ( nota >= 9 )
       resultado = "Sobresaliente";
    else if ( nota >= 7 )
       resultado = "Notable";
    else if ( nota >= 5 )
       resultado = "Aprobado";
    else
       resultado = "Suspenso";


    JOptionPane.showMessageDialog
      ( null, resultado,
        "Calificación final",
        JOptionPane.INFORMATION_MESSAGE );

    System.exit( 0 );
  }
}
```

El `if` encadenado anterior equivale a:

```
import javax.swing.JOptionPane;

public class IfChain2
{
  public static void main( String args[] )
  {
    String entrada;
    String resultado;
    float  nota;


    entrada = JOptionPane.showInputDialog
              ( "Calificación numérica:" );

    nota = Float.parseFloat( entrada );


    resultado = “”;

    if ( nota >= 9 )
       resultado = "Sobresaliente";

    if ( (nota>=7) && (nota<9) )
       resultado = "Notable";

    if ( (nota>=5) && (nota<7) )
       resultado = "Aprobado";

    if ( nota < 5 )
       resultado = "Suspenso";


    JOptionPane.showMessageDialog
      ( null, resultado,
        "Calificación final",
        JOptionPane.INFORMATION_MESSAGE );

    System.exit( 0 );
  }
}
```

Anidamiento

Las sentencias `if` también se pueden anidar unas dentro de otras.

Ejemplo: Resolución de una ecuación de primer grado **`ax+b=0`**

```
import javax.swing.JOptionPane;

public class IfNested
{
  public static void main( String args[] )
  {
    String entrada;
    String resultado;
    float  a,b,x;

    entrada = JOptionPane.showInputDialog
              ( “Coeficiente a” );
    a = Float.parseFloat( entrada );

    entrada = JOptionPane.showInputDialog
              ( “Coeficiente b” );
    b = Float.parseFloat( entrada );

    if (a!=0) {
       x = -b/a;
       resultado = "La solución es " + x;
    } else {
       if (b!=0) {
          resultado = "No tiene solución.";
       } else {
          resultado = "Solución indeterminada.";
       }
    }

    JOptionPane.showMessageDialog
      ( null, resultado,
        "Solución de la ecuación de primer grado",
        JOptionPane.INFORMATION_MESSAGE );

    System.exit( 0 );
  }
}
```

El `if` anidado anterior equivale a …

```
import javax.swing.JOptionPane;

public class IfNested2
{
  public static void main( String args[] )
  {
    String entrada;
    String resultado;
    float  a,b,x;

    entrada = JOptionPane.showInputDialog
              ( “Coeficiente a” );
    a = Float.parseFloat( entrada );

    entrada = JOptionPane.showInputDialog
              ( “Coeficiente b” );
    b = Float.parseFloat( entrada );

    resultado = "";

    if (a!=0) {
       x = -b/a;
       resultado = "La solución es " + x;
    }

    if ( (a==0) && (b!=0) ) {
       resultado = "No tiene solución.";
    }

    if ( (a==0) && (b==0) ) {
       resultado = "Solución indeterminada.";
    }

    JOptionPane.showMessageDialog
      ( null, resultado,
        "Solución de la ecuación de primer grado",
        JOptionPane.INFORMATION_MESSAGE );

    System.exit( 0 );
  }
}
```

En este caso, se realizan 5 comparaciones en vez de 2.

El operador condicional **`?:`**

Java proporciona una forma de abreviar una sentencia `if`

> El operador condicional `?:`
> permite incluir una condición dentro de una expresión.

Sintaxis

```
variable = condición? expresión1: expresión2;
```

"equivale" a

```
if (condición)
   variable = expresión1;
else
   variable = expresión2;
```

Sólo se evalúa una de las expresiones,
por lo que deberemos ser cuidadosos con los efectos colaterales.

Ejemplos

```
max = (x>y)? x : y;

min = (x<y)? x : y;

med = (x<y)? ((y<z)? y: ((z<x)? x: z)):
             ((x<z)? x: ((z<y)? y: z));

nick = (nombre!=null)? nombre : "desconocido";
```

Selección múltiple con la sentencia **`switch`**

Permite seleccionar entre varias alternativas posibles

Sintaxis

```
switch (expresión) {

  case expr_cte1:
       bloque1;
       break;

  case expr_cte2:
       bloque2;
       break;
  ...
  case expr_cteN:
       bloqueN;
       break;

  default:
       bloque_por_defecto;
}
```

- Se selecciona a partir de la evaluación de una única expresión.
- La expresión del `switch` ha de ser de tipo entero (`int`).
- Los valores de cada caso del `switch` han de ser constantes.
- En Java, cada bloque de código de los que acompañan a un posible valor de la expresión entera ha de terminar con una sentencia `break;`
- La etiqueta `default` marca el bloque de código que se ejecuta por defecto (cuando al evaluar la expresión se obtiene un valor no especificado por los casos anteriores del `switch`).
- En Java, se pueden poner varias etiquetas seguidas acompañando a un único fragmento de código si el fragmento de código que ha de ejecutarse es el mismo para varios valores de la expresión entera que gobierna la ejecución del `switch`.

Ejemplo

```
public class Switch
{
  public static void main( String args[] )
  {
    String resultado;
    int    nota;

    // Entrada de datos
    …
    switch (nota) {
      case 0:
      case 1:
      case 2:
      case 3:
      case 4:
           resultado = "Suspenso";
           break;

      case 5:
      case 6:
           resultado = "Aprobado";
           break;

      case 7:
      case 8:
           resultado = "Notable";
           break;

      case 9:
      case 10:
           resultado = "Sobresaliente";
           break;

      default:
           resultado = "Error";
    }

    // Salida de resultados
    …
  }
}
```

Si tuviésemos que trabajar con datos de tipo real, no podríamos usar `switch` (usaríamos `if`s encadenados).