Bloque II. Elementos del lenguaje de programación Java

4. Instrucciones de control

Las instrucciones de un programa pueden ser:

- *simples*:
  - expresiones: de asignación, incremento o decremento
  - llamadas a métodos
  - creación de objetos
  - instrucciones de control: `if`, `switch`, `while`, `do-while`, `for`
- *compuestas*:
  - se encierran entre llaves `{}`, y también se llaman *bloques*
  - pueden contener muchas instrucciones y declaraciones;
  - las declaraciones del bloque sólo son visibles en él, y en los bloques contenidos en él

4.1. Instrucción condicional simple

La instrucción condicional simple permite tomar decisiones empleando una variable booleana:

Java	Pseudocódigo

`if (condición) {`
`  instrucciones;`
`}`
	`si condición entonces`
	`  instrucciones`
	`fsi`

`if (condición) {`
`  instrucciones;`
`} else {`
`  instrucciones;`
`}`
	`si condición entonces`
	`  instrucciones`
	`si no`
	`  instrucciones`
	`fsi`

La condición: expresión booleana (lógica o relacional)

La instrucción condicional simple (cont.)

También se puede escribir, aunque es menos recomendable (por ser menos visible el comienzo y final):

```
if (condición)
   instrucción;
else
   instrucción;
```

Ejemplo: poner un texto aprobado o suspenso según la nota

```
if (nota>=5.0) {
   System.out.println("Aprobado");
} else {
   System.out.println("Suspenso");
}
```

Instrucciones condicionales anidadas

Las instrucciones if también se pueden anidar:

- el `else` se asocia al `if` anterior más próximo que no tenga `else`, siempre que esté en el mismo bloque que el `else`.

Ejemplo: poner "cum laude" en el ejemplo anterior si nota>=9

```
if (nota>=5.0) {
   System.out.print("Aprobado");
   if (nota>=9.0) {
      System.out.println(" cum laude");
   } else {
      System.out.println("");
   }
} else {
   System.out.println("Suspenso");
}
```


Expresiones condicionales

Como expresión condicional se pueden usar operaciones relacionales y lógicas

Ejemplo: Intervalo: condición a en (5.0,6.3]

```
if (a>5.0 && a<=6.3) ...
```

Ejemplo: Intervalo contrario: condición a no está en (5.0,6.3]

```
if (a<=5.0 || a>6.3) ...
```

```
if (!(a>5.0 && a<=6.3)) ...
```

Ejemplo: año bisiesto

```
boolean esBisiesto;
int año=...;

if (año % 4 == 0) {
   if (año % 100 == 0) {
      if (año % 400 == 0) {
         esBisiesto=true;
      } else {
         esBisiesto=false;
      }
   } else {
      esBisiesto=true;
   }
} else {
   esBisiesto=false;
}
```

Son bisiestos los años múltiplos de 4, excepto los múltiplos de 100 que no sean múltiplos de 400

Ejemplo: año bisiesto (cont.)

```
if (esBisiesto) {
   System.out.println("El año "+año+" es bisiesto");
} else {
   System.out.println("El año "+año+" no es bisiesto");
}
```

4.2. Instrucción condicional múltiple

Permite tomar una decisión de múltiples posibilidades, en función de un valor no booleano

- **Si este valor es discreto (`byte`, `short`, `int`, `long`, `char`, o *enumerado*), podemos utilizar una instrucción `switch`**

Instrucción condicional múltiple (cont.)

Java	Pseudocódigo

`switch (expresión discreta) {`
`case valor1:`
`  instrucciones;`
`  break;`
`case valor2:`
`  instrucciones;`
`  break;`
`case valor3:`
`case valor4:`
`  instrucciones;`
`  break;`
`default:`
`  instrucciones;`
`}`
	`si`
	`  exp=valor1 -> instrucciones;`
	`  exp=valor2 -> instrucciones;`
	`  exp=valor3 | exp= valor4 ->`
	`  instrucciones;`
	`  exp=ninguno de los anteriores ->`
	`  instrucciones`
	`fsi`

Instrucción switch (cont.)

El funcionamiento es el siguiente:

- se compara la expresión con el primer valor
- si coincide, se ejecutan las instrucciones puestas bajo ese valor, y todas las siguientes que se encuentren, hasta encontrar un `break`.
- si no coincide, se compara con el segundo valor, y así sucesivamente
- si no coincide con ningún valor, se ejecutan las instrucciones que haya en la parte `default`, si existe.
- después de un `break`, la instrucción `switch` termina y seguimos por la siguiente instrucción
- los valores deben ser constantes, no variables,
- no puede haber ninguno coincidente.

Ejemplo: nota media (entera) con letra

```
public class NotaEntera {

    private int notaMedia;

    public NotaEntera (int nota) {
       notaMedia=nota;
    }

    public String convierte() {
        String notaLetra;

        switch (notaMedia) {
         case 0:
         case 1:
         case 2:
         case 3:
         case 4:
            notaLetra="Suspenso";
            break;
```

Ejemplo: nota media (entera) con letra (cont.)

```
            case 5:
            case 6:
               notaLetra="Aprobado";
               break;
            case 7:
            case 8:
               notaLetra="Notable";
               break;
            case 9:
            case 10:
               notaLetra="Sobresaliente";
               break;
            default:
               notaLetra="Error";
           }
           return notaLetra;
    }
}
```


Instrucción condicional múltiple no discreta

Cuando la decisión no es discreta, usamos una "escalera" de instrucciones `if`:

Java	Pseudocódigo

`if (condición1) {`
`  instrucciones;`
`} else if (condición2) {`
`  instrucciones;`
`} else if (condición3) {`
`  instrucciones;`
`} ...`
`} else {`
`  instrucciones;`
`}`
	`si`
	`  condición1 -> instrucciones;`
	`  condición2 -> instrucciones;`
	`  condición3 -> instrucciones;`
	`  ninguna de las anteriores ->`
	`  instrucciones;`
	`fsi`

Instrucción condicional múltiple no discreta (cont.)

- Las condiciones se examinan empezando por la de arriba
- Tan pronto como una se cumple, sus instrucciones se ejecutan y la instrucción se abandona.
- Si ninguna de las condiciones es cierta se ejecuta la última parte `else`.

La instrucción `switch` es mucho más eficiente que la instrucción condicional múltiple

- en `switch` sólo se toma una decisión
- en el `if` múltiple se evalúan muchas condiciones.

Ejemplo: nota media (real) con letra

```
public class NotaReal {

   private double notaMedia;

   public NotaReal(double nota) {
      notaMedia=nota;
   }

   public String convierte() {
      String notaLetra;

      if (notaMedia<0.0) {
         notaLetra="Error";
      } else if (notaMedia<5.0) {
         notaLetra="Suspenso";
      } else if (notaMedia<7.0) {
         notaLetra="Aprobado";
      } else if (notaMedia<9.0) {
         notaLetra="Notable";
```

Ejemplo: nota media (real) con letra (cont.)

```
        } else if (notaMedia<=10.0) {
            notaLetra="Sobresaliente";
        } else {
            notaLetra="Error";
        }
        return notaLetra;
    }
}
```


4.3. Instrucciones de lazo o bucle

Permiten ejecutar múltiples veces unas instrucciones

- **se corresponden a la *composición iterativa* de teoría**

La cantidad de veces se puede establecer mediante:

- ***una condición*:**
  - **se comprueba *al principio*: las instrucciones del lazo se hacen cero o más veces**
  - **se comprueba *al final*: las instrucciones del lazo se hacen una o más veces**
- ***un número fijo de veces*: se usa una variable de control**

4.3.1. Lazo con condición de permanencia al principio

Es el lazo `while`:

Java	Pseudocódigo

`while (condicion) {`
`  instrucciones;`
`}`
	`mientras condición hacer`
	`  instrucciones;`
	`fmientras`

Ejemplo

Calcular el primer entero positivo tal que la suma de él y los anteriores sea mayor que 100

```
public class SumaMayor100 {

  public static void main(String[] args) {
     int suma =0;
     int i=0;
     while (suma<=100) {
         i++;
         suma=suma+i;
     }
     System.out.println("La suma de i=1.."+i+" es "+suma);
  }
}
```

Ejemplo 2: lazo infinito o indefinido

Cálculo de las distancias entre dos puntos del globo terráqueo, múltiples veces.

```
import fundamentos.*;
public class Dist {

  public static void main(String[] args) {

    double dist; // Kilómetros
    double lon1,lat1,lon2,lat2; // grados

    Lectura pantalla = new Lectura("Círculo Máximo");

    pantalla.creaEntrada("Latitud  1",0.0);
    pantalla.creaEntrada("Longitud 1",0.0);
    pantalla.creaEntrada("Latitud  2",0.0);
    pantalla.creaEntrada("Longitud 2",0.0);
```


Ejemplo 2: lazo infinito o indefinido (cont.)

```
    while (true) {
        pantalla.espera("Introduce coordenadas y pulsa OK");
        lat1=pantalla.leeDouble("Latitud  1");
        lon1 =pantalla.leeDouble("Longitud 1");
        lat2 =pantalla.leeDouble("Latitud  2");
        lon2 =pantalla.leeDouble("Longitud 2");

        lat1=Math.toRadians(lat1);
        lat2=Math.toRadians(lat2);
        lon1=Math.toRadians(lon1);
        lon2=Math.toRadians(lon2);
        dist=Math.toDegrees(Math.acos(Math.sin(lat1)*
           Math.sin(lat2)+
           Math.cos(lat1)*Math.cos(lat2)*Math.cos(lon1-lon2)))*
           60.0*1.852;
        pantalla.println("La distancia es: "+dist+" Km");
    }
  }
}
```

4.3.2. Lazo con condición de permanencia al final

Es el lazo do-while:

Java	Pseudocódigo

`do {`
`  instrucciones;`
`} while (condicion);`
	`hacer`
	`  instrucciones;`
	`mientras condición;`

Ejemplo

Calcular el máximo de unos números positivos hasta que el introducido sea cero

```
import fundamentos.*;

public class Maximo {

  public static void main(String[] args) {

    double max = 0.0;
    double num;

    Lectura pantalla = new Lectura("Máximo");

    pantalla.creaEntrada("Número",0.0);
```

Ejemplo (cont.)

```
    do {

       pantalla.espera("Introduce número y pulsa OK (0 = fin)");
       num = pantalla.leeDouble("Número");
       if (num>max) {
          max=num;
       }
       pantalla.println("El máximo es: "+max);
     } while (num!=0);

     pantalla.println("Pulsa Cerrar");

  }
}
```


4.3.3 Lazo con variable de control

Es el lazo `for`:

```
for (decl-inicialización; cond-permanencia; expr-incremento;) {
   instrucciones;
}
```

Es equivalente a:

Java	Pseudocódigo

`{`
`  decl-inicialización;`
`  while (cond-permanencia) {`
`  instrucciones;`
`  expr-incremento;`
`  }`
`}`
	`decl-inicialización;`
	`mientras cond-permanencia hacer`
	`  instrucciones;`
	`  expr-incremento`
	`fmientras`

Ejemplo: suma de los 100 primeros enteros positivos

Existe también una sintaxis especial de pseudocódigo:

Java	Pseudocódigo

`int suma=0;`
`for (int i=1; i<=100; i++) {`
`  suma=suma+i;`
`}`
	`entero suma=0;`
	`para i=1 hasta i=100 hacer`
	`  suma:=suma+i;`
	`fpara`

También para incrementos distintos de uno (ej: nº pares):

Java	Pseudocódigo

`int suma=0;`
`for (int i=2; i<=100; i=i+2) {`
`  suma=suma+i;`
`}`
	`entero suma=0;`
	`para i=2 paso 2 hasta i=100 hacer`
	`  suma:=suma+i;`
	`fpara`

Recomendaciones sobre el lazo for

- Debe usarse para lazos con variable de control y de una manera uniforme
- Es conveniente declarar la variable de control en el lazo
- Es conveniente que la expresión de incremento sea eso
- Es conveniente que la expresión de permanencia sea simple
- Nunca cambiar el valor de la variable de control en las instrucciones.

Variantes de lazos

Hacia atrás:

```
for (int n=10; n>=-6; n--) ...
```

Vacío:

```
for (int n=0; n<finish; n++) ...//si finish<0
```

Anidado

```
for (int i=1; i<=10; i++) {
   for (int j=1; j<=20; j++) {
      ...
   }
}
```


Ejemplo: uso de la clase Grafica

Es una clase sencilla para hacer gráficos de funciones reales. Permite:

- almacenar puntos
- mostrarlos como puntos o líneas
- mostrar el gráfico
- puede mostrar varios gráficos en la misma ventana

Ejemplo (cont.)

```
import fundamentos.*;

public class FuncionesTrigonometricas {

  public static void main(String[] args) {

      // Gráficas de funciones trigonometricas

      Grafica g = new Grafica ("Seno y Coseno","x","y");
      double x;

      // El primer gráfico
      g.ponSimbolo(true);
      g.ponColor(Grafica.azul);
      g.ponTitulo("Seno");
      // Angulos desde 0 a 3*PI con incremento de PI/16
      for (double x1=0.0; x1<=Math.PI*3.0; x1=x1+Math.PI/16.0) {
         g.inserta(x1,Math.sin(x1));
      }
```


Ejemplo (cont.)

```
      // El segundo gráfico
      g.otraGrafica();
      g.ponSimbolo(true);
      g.ponColor(Grafica.rojo);
      g.ponTitulo("Coseno");
      // Angulos desde 0 a 10 radianes con incremento de 0.1 radianes
      for (int i=0; i<=100; i++) {
         x = i/10.0;
         g.inserta(x,Math.cos(x));
      }
      g.pinta();

  }
}
```

4.3.4. Instrucciones de salto en lazos

Hay tres instrucciones que permiten saltarse las instrucciones restantes del lazo:

- **`break`:**
  - termina el lazo
- **`continue`:**
  - termina las instrucciones del lazo, pero sigue en él
- **`return`:**
  - termina un método; si estamos en un lazo, lógicamente también lo termina