CONTROL DE FLUJO EN JAVA

El modo de ejecución de un programa en Java en ausencia de elementos de control de flujo es secuencial, es decir una instrucción se ejecuta detrás de otra y sólo se ejecuta una vez. Esto nos permite hace programas muy limitados; para evitarlo se introducen estructuras de control de flujo.

Las sentencias de control de flujo determinan el orden en que se ejecutarán las otras sentencias dentro del programa. El lenguaje Java soporta varias sentencias de control de flujo, incluyendo.

- Condicionales **if-else**, **switch**
- bucles **for**, **while**, **do-while**
- excepciones **try-catch-finally**, **throw**
- miscelaneas **break**, **continue**, **label:**, **return**

Las excepciones las veremos en un tema posterior.

SENTENCIAS CONDICIONALES

Sentencia if

```
if (expresion) {
    Bloque de código
}
```

La sentencia 'if' es la sentencia más básica de entre las de control de flujo. Le indica al programa que ejecute un bloque de código solo si una determinada expresión se evalúa a 'true'. Por ejemplo, la clase Bicicleta puede permitir a los frenos disminuir la velocidad de la bicicleta solo si la bicicleta ya está en movimiento. Una posible implementación de un método para frenar:

```
void frenar(){
  if (enMovimiento){  //  si la bicicleta se está moviendo
    velocidad--; //  entonces, decrementar la velocidad
  }
}
```

Si la expresión evalúa a 'false', el control salta al final de la sentencia 'if', no ejecutándose el bloque de código encerrado dentro de la misma.

Las llaves de apertura y cierre son opcionales en caso de que el código a ejecutar esté compuesto por una única sentencia:

```
void frenar(){
  if (enMovimiento)  //  si la bicicleta se está moviendo
    velocidad--; //  entonces, decrementar la velocidad
}
```

Decidir si omitir o no las llaves depende de las preferencias personales, pero el ponerlos ayuda a evitar posibles errores producidos al añadir posteriormente nuevas sentencias dentro del código a ejecutar en caso de cumplirse la expresión.

Sentencia if-else

```
if {
  Bloque de código1
}
else {
  Bloque de código2
}
```

La sentencia 'if-else' proporciona un camino secundario de ejecución cuando una cláusula 'if' se evalúa a 'false'. En este caso se ejecutarían las sentencias del 'Bloque de código2'.

Con el ejemplo anterior de los frenos, añadimos ahora la acción de que muestre un error si los frenos son aplicados con la bicicleta parada.

```
void frenar(){
  if (enMovimiento){  //  si la bicicleta se está moviendo
    velocidad--; //  entonces, decrementar la velocidad
  }
  else {
    System.out.println("La bicicleta ya estaba parada.");
  }
}
```

Las sentencias 'if-else' pueden anidarse, esto es, un 'else' puede llevar como código asociado otra sentencia 'if' o 'if-else'.

Un ejemplo. Asignamos una nota en función del porcentaje de respuestas correctas en un examen.

```
if (porcentaje >= 90) {
    nota = 'S';
} else if (porcentaje >= 80) {
    nota = 'N';
} else if (porcentaje >= 70) {
    nota = 'B';
} else if (porcentaje >= 60) {
    nota = 'A';
} else {
    nota = 'I';
}
System.out.println("Nota = " + nota);
```

La variable 'porcentaje' puede satisfacer más de una de las expresiones. Por ejemplo un porcentaje de 76, cumplirá la sentencia: 76 >= 70 y también: 76 >= 60. Sin embargo, cuando una de las condiciones se cumple, se ejecuta la sentencia o sentencias asociadas (en esta caso: nota = 'B') y las demás condiciones no serán evaluadas.

Sentencia switch

```
switch (expresionentera) {
  case valor1:
        Bloque sentencias1;
        break;
  case valor2:
       Bloque sentencias1;
       break;
  […]
  case valorN:
        Bloque sentenciasN;
        break;
  [default:
        Bloque sentencias default;
        break;]
}
```

También denominada condicional múltiple, una sentencia 'switch' no se limita a dos posibles caminos e ejecución ('true' o 'false'), permitiendo especificar cualquier cantidad de alternativas posibles.

La expresión a analizar en la sentencia 'switch' debe de evaluar a un tipo byte, short, char o int (en definitiva, un número entero). También permite trabajar con sus clases envoltorios: Byte, Short, Char, Integer. Y a partir de la versión 7 del jdk, también con la clase String.

La sentencia 'switch' evalúa su expresión, luego ejecuta todas las sentencias que siguen a la etiqueta 'case' que coincida con el resultado de la evaluación.

La sección opcional 'default', trata todos los valores que no hayan sido explícitamente indicados en las secciones 'case' anteriores.

Un ejemplo de código que a partir del número de día de la semana, imprime el nombre del mismo.

```
switch (dia) {
  case 1:
    diaString = "Lunes";
    break;
  case 2:
    diaString = "Martes";
    break;
  case 3:
    diaString = "Miércoles";
    break;
  case 4:
    diaString = "Jueves";
    break;
  case 5:
    diaString = "Viernes";
    break;
  case 6:
    diaString = "Sábado";
    break;
  case 7:
    diaString = "Domingo";
    break;
  default:
    diaString = "dia no válido";
}
System.out.println(diaString);
```

Otro punto importante es la sentencia 'break'. La sentencia 'break' es necesaria debido a que sin su uso, todas las sentencias escritas a continuación de la etiqueta que

coincida con la evaluación de la expresión, serán ejecutadas en secuencia hasta el final del 'switch' o bien hasta que un 'break' se encontrado. Si en el ejemplo anterior no incluyésemos las sentencias 'break' el valor final de 'diaString' sería siempre "día no válido" independientemente del valor numérico del día.

Aunque un break después de la última sentencia del 'switch' no es necesario, puede ser recomendable su uso para evitar errores al modificar el código.

El mismo programa se podría haber realizado anidando sentencias 'if-else':

```
if (dia == 1) {
  diaString = "Lunes";
}
else if (dia == 2) {
  diaString = "Martes";
}
else if (dia == 3) {
  diaString = "Miércoles";
}
else if (dia == 4) {
  diaString = "Jueves";
}
else if (dia == 5) {
  diaString = "Viernes";
}
else if (dia == 6) {
  diaString = "Sábado";
}
else if (dia == 7) {
  diaString = "Domingo";
}
else {
  diaString = "dia no válido";
}
System.out.println(diaString);
```

En principio la legibilidad al utilizar 'switch' en lugar de 'if-else' es mayor. Pero hay que tiene en cuenta que una sentencia 'if-else' puede funcionar con expresiones basadas en rangos de valores, caso que no es posible con 'switch'.

SENTENCIAS ITERATIVAS

Sentencia while

```
while (expresion) {
    Bloque de código
}
```

La sentencia 'while' ejecuta repetidamente un bloque de instrucciones, mientras la expresión indicada sea 'true'.

La expresión debe devolver un valor booleano. Si la expresión se evalúa a 'true' se ejecuta el Bloque de código encerrado, a continuación se vuelve a evaluar la expresión y así sucesivamente, hasta que la expresión devuelva 'false' momento en que la ejecución del programa seguirá por la siguiente sentencia posterior a la sentencia 'while'.

Un programa que imprime los números entre 1 y 10:

```
int num = 1;
while (num < 11) {
  System.out.println(num);
  num++;
}
```

Para crear un bucle infinito:

```
while (true) {
  // código a ejecutar
}
```

Sentencia do-while

```
do {
    Bloque de código
} while (expresion);
```

La diferencia entre 'while' y 'do-while' está en que 'do-while' evalúa la expresión al final del bucle, mientras 'while' lo hace al principio. Por tanto las sentencias dentro

de ‘do-while’ serán siempre ejecutadas al menos 1 vez. Con ‘while’ pueden no ser ejecutadas nunca.

El programa que escribe del 1 al 10 utilizando ‘do-while’:

```
int num = 1;
do {
  System.out.println(num);
  num++;
} while (num < 11);
```

Sentencia for

La sentencia ‘for’ proporciona una manera compacta de iterar sobre un rango de valores. La forma general se pude expresar como:

```
for (inicialización; terminación; incremento) {
   bloque de código
}
```

La expresión ‘inicialización’ inicializa el bucle. Es ejecutada una única vez al comienzo del bucle.

Cuando la expresión ‘terminación’ evalúa a ‘false’, el bucle finaliza.

La expresión ‘incremento’ es llamada después de cada iteración.

El programa que escribe los números del 1 al 10:

```
for (int num = 1; num < 11; ++num) {
  System.out.println(num);
}
```

Como se ve en el código anterior, es posible declarar variables en la expresión ‘inicialización’. El alcance de esta variable es el de la propia sentencia ‘for’.

Las tres expresiones son opcionales. Un bucle infinito podría crearse así:

```
for (;;) {
  // código a ejecutar
}
```

La sentencia ‘for’ tiene otra forma, diseñada para iterar a través de Collections y arrays. Esta forma puede hacer los bucles más compactos y fáciles de leer.

Un ejemplo iterando sobre un array de números.

Con la forma tradicional de la sentencia ‘for’:

```
int[] numeros = {1,2,3,4,5,6,7,8,9,10};
for (int i = 0; i < 10; ++i) {
  System.out.println("Núm: " + numeros[i]);
}
```

Con la nueva forma:

```
int[] numeros = {1,2,3,4,5,6,7,8,9,10};
for (int item : numeros) {
  System.out.println("Núm: " + item);
}
```

En este ejemplo, la variable ‘item’ almacena el valor de la posición del array en cada momento. No se indica un incremento explícitamente, se recorre todas las posiciones desde la primera (0) a la última (9).

Limitaciones:

- Al menos Java 5.
- Solo hacia adelante, y elemento a elemento.
- Sólo acceso. Los elementos no pueden ser asignados.
- Sólo estructuras simple. Por ejemplo no se pueden comparar 2 arrays.
- Sólo elementos simples. Por ejemplo no se pueden comparar elementos contiguos.

Tanto en las sentencias ‘while’ como en las ‘do-while’ como en las ‘for’, si el cuerpo del bucle está formado por una única sentencia, las llaves no son necesarias.

SENTENCIAS DE CORTE

Sentencia break

La sentencia 'break' tiene dos formas: etiquetada y no etiquetada. La no etiquetada es la que se usa, por ejemplo, en la sentencia 'switch'. También se puede usar para terminar un bucle 'for', 'while' o 'do-while'.

Un ejemplo:

```
int[] numeros = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
int buscar = 12;
boolean encontrado = false;

int i;
for (i = 0; i < numeros.length; i++) {
  if (numeros[i] == buscar) {
    encontrado = true;
    break;
  }
}

if (encontrado) {
  System.out.println("Encontrado " + buscar + " en posición " + i);
} else {
  System.out.println(buscar + " no está en el array");
}
```

Este programa busca el número 12 en el array. La sentencia 'break' fuerza la finalización del bucle en caso de haber encontrado el número. Como la variable 'i' se va a utilizar después de bucle, no debe ser declarada dentro del 'for' si no antes.

Una sentencia 'break' no etiquetada termina el 'switch', 'for', 'while' o 'do-while' más interno. Mediante un 'break' etiquetado podemos salir de sentencias anidadas.

```
etq: for (int i = 1; i < 6; i++ ) {
       System.out.print(" i="+i);
       for (int j = 1; j < 6; j++ ){
         if (j == 3) {
           break etq;
         }
         System.out.print(" j="+j);
       }
     }

     System.out.println(" FIN");
```

Imprimiría:

```
i=1 j=1 j=2 FIN
```

En mismo programa con un 'break' no etiquetado imprimiría:

```
i=1 j=1 j=2 i=2 j=1 j=2 i=3 j=1 j=2 i=4 j=1 j=2 i=5 j=1 j=2 FIN
```

La sentencia 'break' termina la sentencia etiquetada; no transfiere el flujo a la etiqueta. El control de flujo se transfiere a la sentencia inmediatamente posterior a la sentencia etiquetada.

Sentencia continue

La sentencia 'continue' fuerza una nueva iteración de un bucle. Admite forma etiquetada y no etiquetada. La forma no etiquetada fuerza la siguiente iteración del bucle más interno que la contenga, evaluando de nuevo la expresión booleana que controla el bucle.

Un ejemplo con un programa que busca las letras 'p' existentes en una cadena de caracteres, indicando cuantas hay:

```
String cadena = "peter piper picked a peck of pickled peppers";
int max = cadena.length();
int numPs = 0;

for (int i = 0; i < max; i++) {
    // busamos las 'p' que haya en la cadena
    if (cadena.charAt(i) != 'p')
        continue;
    numPs++;
}
System.out.println("Encontradas " + numPs + " p's en la cadena.");
```

El programa imprimirá:

```
Encontradas 9 p's en la cadena.
```

Si no utilizásemos la sentencia 'continue' el resultado (erróneo) sería:

```
Encontradas 44 p's en la cadena.
```

Una sentencia ‘continue’ etiquetada fuerza la siguiente iteración del bucle marcado por la etiqueta.

```
etq: for (int i = 1; i < 6; i++ ) {
       System.out.print(" i="+i);
       for (int j = 1; j < 6; j++ ){
         if (j == 3) {
           continue etq;
         }
         System.out.print(" j="+j);
       }
     }

     System.out.println(" FIN");
```

El programa imprimirá:

```
i=1 j=1 j=2 i=2 j=1 j=2 i=3 j=1 j=2 i=4 j=1 j=2 i=5 j=1 j=2 FIN
```

En mismo programa con un ‘continue’ no etiquetado imprimiría:

```
i=1 j=1 j=2 j=3 j=4 j=5 i=2 j=1 j=2 j=3 j=4 j=5 i=3 j=1 j=2 j=3 j=4 j=5
i=4 j=1 j=2 j=3 j=4 j=5 i=5 j=1 j=2 j=3 j=4 j=5 FIN
```