Unidad 5

Estructura de control

Programación

5

Índice

Introducción

Debido que la gran diversidad de tipos de programas que existen, estos pueden tomar de varios lugares los datos con los que trabajan. Además, estos programas están diseñados para que nos muestren los resultados de maneras diversas. Pero la forma en la que obtienen los datos no afecta a su manera de ejecutarse, todos se ejecutan de manera secuencial, desde la primera línea del programa hasta la última. Las sentencias de control nos permitirán, dependiendo de la lógica que necesitemos aplicar, cambiar el orden en que se ejecutan.

En esta unidad veremos cuál es la forma básica para comunicarnos con un programa que se haya desarrollado en Java, y las sentencias de control con las que podremos trabajar con la información obtenida.

Al finalizar esta unidad

- Conoceremos los mecanismos básicos de entrada y salida de datos desde consola en un programa.
- Comprenderemos el manejo de las diferentes estructuras de selección, repetición y salto.
- Desarrollaremos un programa que evalúe datos de entrada y devuelva el resultado correspondiente.

5.1. Entrada y salida de información

Podemos comunicarnos con Java mediante distintas alternativas. A continuación, veremos la entrada-salida de información por el terminal y la entrada de información por argumentos (en la llamada).

5.1.1. Entrada por argumentos

Los programas Java utilizan el método main que declaramos en la clase principal como punto desde el que comenzar. Empezando por este método, el programa llevará a cabo las funciones que le correspondan, ya sea hasta terminar el trabajo o hasta que este se detenga, por causas ajenas o controladas por el programa.

Este método es declarado como estático (no hay necesidad de instanciar la clase en la que está contenido para poder invocarlo), público (se puede acceder desde cualquier sitio) y utilizando el parámetro args, cuando invoquemos el programa con argumentos, este parámetro los contendrá.

Por ahora solo hemos trabajado con programas cuyos datos eran establecidos desde el mismo programa, sin interaccionar con el usuario ni el traspaso de argumentos.

Necesitamos añadir a la llamada aquellos argumentos que queramos pasar a un programa (ya viniera dentro de un jar ejecutable o fuera una ejecución de forma simple). Los enviaremos separados por un espacio y sin cantidad límite. Para recuperarlos posteriormente en el método main, utilizaremos un String, en el que el primer argumento se escribirá como args[0], el segundo como args[1], etc.

Para trabajar con argumentos en NetBeans, debemos ir a propiedades del proyecto, luego entramos en la categoría Run, y especificamos, igual que haríamos por el terminal, los argumentos en la sección Arguments.

5.1.2. Entrada y salida por consola

A través de la invocación con argumentos, podemos transferir datos al programa, y este los recogerá para empezar la ejecución. No obstante, la implementación de programas interactivos necesita de algo más. Los flujos de datos (Streams) son la manera en que los datos entran y salen en Java. Ya en la primera versión de Java, este mecanismo estaba presente, y facilita un conjunto de flujos de datos estándar, los cuales estarán listos para darnos datos mientras que el programa se esté ejecutando. Encontramos los siguientes Streams:

- **Flujo estándar de salida System.out:** este objeto es de clase PrintStream, y es la pantalla por defecto. Si queremos que se muestren los datos por consola deberemos hacer uso de los siguientes métodos:
  - System.out.write (datos) → Este método consigue que los bytes que han sido enviados como argumento por el flujo estándar sean impresos.
  - System.out.print (obj) → Transforma a un conjunto de bytes aquella información que se ha enviado como argumento para después imprimirla mediante el método System.out.write.
  - System.out.println (obj) → Hace lo mismo que el método anterior, nada más que al final añade un salto de línea.
  - System.out.printf (String cadena, [tipo argumento1]...[,tipo argumentoN]) → Se imprime la cadena e incluye los argumentos en las especificaciones de formato. Si el número de especificaciones de formato es inferior al número de argumentos, los argumentos que sobren no se tendrán en cuenta.
- **Flujo estándar de salida de error System.err:** este método es el mismo que System.out, nada más que para salida de error.
- **Flujo estándar de entrada System.in:** este objeto es de clase InputStream, y es el teclado por defecto.
  - System.in.read (datos) → Los caracteres que se lean desde el teclado se guardaran en datos, el cual deberá haber sido declarado como un array de bytes con anterioridad.
  - System.in.read () → Devuelve el siguiente byte de datos desde el flujo de entrada en forma de número entero (de 0 a 255). Debe ser convertido en char a través de un casting.

Aquí podemos ver un ejemplo de Entrada/Salida utilizando flujos estándar:

```
int n1, n2, n3;
System.out.println("Introduce tres números");
n1 = System.in.read();
n2 = System.in.read();
n3 = System.in.read();
System.out.write(n1);
System.out.print((char)n2);
System.out.println((char)n3);
```

La clase Scanner es un tipo de mecanismo que nos permite leer los Stream de forma más cómoda aún. Fue introducido en la versión 1.5 de Java, este método analiza los tipos primitivos y divide la entrada en tokens, los cuales, por defecto, están separados por espacios. Entre los métodos más utilizados están:

- **nextLine ()** → Devuelve como un String la línea completa.
- **next ()** → Devuelve como un String el siguiente token.
- **nextXXXX ()** → Devuelve como si fuera de tipo XXXX el próximo token (siendo XXXX Byte, Double, Int, etc.).
- **hasNext ()** → En caso de que el escáner pueda devolver otro token, retorna true.
- **hasNextInt ()** → Si usando el método nextXXXX(), podemos interpretar el siguiente token como si fuera un valor del tipo XXXX, devuelve true.

Aquí podemos ver un ejemplo de entrada/salida que utiliza la clase Scanner:

```
package entradasalida ;
import java.util.Scanner;
public class EntradaSalida {
        public static void main(String[] args) {
                System.out.println(“Escriba su texto”);
                Scanner entradaEscaner = new Scanner (System.in);
                String entrada Teclado = entradaEscaner.nextLine ();
                System.out.println (entradaTeclado);
        }
}
```

Otra clase muy útil es la Console, que incorpora funciones para leer passwords ocultando aquellos caracteres que tecleemos. Se incorporó en Java 1.6 y entre sus métodos encontramos:

- **readLine ()** → Devuelve como un String la línea completa.
- **readPassword ()** → Mantiene ocultos los caracteres que se introduzcan utilizando el teclado y retorna un array de caracteres.
- **printf (String cadena, [tipo argumento 1]…[,tipo argumentoN])** → De forma parecida al método System.out.printf, este también imprime texto por consola.

5.2. Estructuras de selección (alternativa)

Para escribir el código, los lenguajes de programación cuentan con una serie de estructuras de selección y de control. En los siguientes apartados veremos las estructuras con las que cuenta Java.

5.2.1. if, else, else if

Para empezar, la sentencia de control más sencilla que podemos encontrarnos en todo tipo de lenguajes es la sentencia condicional if. Esta sentencia sirve para evaluar una condición, y dependiendo del resultado, actuar de una cierta manera. Sus variantes son:

- **if** → El programa evaluará la condición contenida entre paréntesis. En caso de retornar true, el grupo de instrucciones que se encuentra contenido entre llaves se ejecuta. En caso de retornar false, simplemente pasará a la siguiente instrucción.
- **If-else** → Esta sentencia se comporta igual que la anterior, excepto que al retornar false, esta ejecuta otro bloque de instrucciones en vez de pasar a la siguiente instrucción. A parte de las anteriores, cabe destacar el operador ternario que nos posibilita llevar a cabo un if-else simple en una sola línea de código.
- **If-else if** → En caso de que la primera sentencia if-else retorne false, se vuelve a analizar una condición, y para ello se encadenan varios bloques if-else.

Dentro de un bloque de instrucciones podemos encontrarnos con otras sentencias condicionales contenidas en ellos. Aquí algunos ejemplos de sentencias condicionales if:

EQUIVALENTES

```
if (s>t) {
        mayor = s;
        menor = t;
}
```

```
if (f>g)
        mayor = f;
else
        mayor = g;
```

```
mayor = (f>g)?f:g;
```

```
if (mes==”enero”) {
        …
} else if (mes==”febrero”) }
        …
} else if (mes==”marzo”) }
        …
} else {
        …
}
```

Imagen 1. Ejemplos de if.

5.2.2. switch

Podemos implementar de forma más sencilla con switch bastantes de las soluciones que haríamos con if-else. Con switch es posible evaluar una variable, y en función del valor de esta, ejecutar un bloque de instrucciones u otro. También podemos incluir la opción default, que ejecuta otro bloque de instrucciones si el valor obtenido no coincide con ninguna de las anteriores opciones. Switch puede trabajar con:

- Strings.
- Tipos primitivos char, short, int, byte.
- Tipos enumerados.
- Clases envoltorio Byte, Short, Integer y Character.

Aquí podemos ver un ejemplo de condicional switch:

```
int mes = 1; // 1-J, 2-F, 3-M, 4-A, 5-Ma
switch (mes) {
        case 1: System.out.println(“January”);
        case 2: System.out.println(“February”);
        case 3: System.out.println(“March”);
        case 4: System.out.println(“April”);
        case 5: System.out.println(“May”);
                breack;
        default: System.out.println(“Resto del año”);
        break;
}
```

Como habréis observado, encontramos un break al final de cada opción. Esta sirve para interrumpir la ejecución de ese bloque y hace que se ejecute la siguiente. Sin ella, se siguen ejecutando las instrucciones siguientes.

5.3. Estructuras de repetición (iterativa)

A continuación, nos encontramos con las estructuras de repetición o bucles, los cuales nos permiten ejecutar repetidamente un bloque de instrucciones.

5.3.1. for

Si queremos iterar sobre un rango de valores, utilizaremos la sentencia de control for. Esta sentencia consta de una expresión de inicialización, la cual solo se ejecuta al comienzo, una expresión de finalización, que finaliza el bucle al retornar false, y una expresión de incremento/decremento que se invoca con cada iteración. En caso de no colocar ninguna de estas tres expresiones, el bucle se ejecutará de forma infinita. Ejemplos:

```
for (int i=1; i<4; i++) {
        System.out.println("Incrementando: " + i);
}
for (int j=3; j>0; j--) {
        System.out.println("Decrementando: " + j);
}
for ( ; ; ) {
        System.out.println("Bucle infinito");
}
```

```
Incrementando 1
Incrementando 2
Incrementando 3
Decrementando 3
Decrementando 2
Decrementando 1
Bucle infinito
Bucle infinito
...
```

Imagen 2. Ejemplos de sentencias de control for.

Podemos declarar la variable que utilicemos en la expresión de incremento/decremento, tanto en la propia expresión de inicialización como fuera de la sentencia iterativa. Si la declaramos fuera, esta permanecerá hasta que termine el bloque en el que se declaró el for, por otro lado, al declararse dentro del propio for, la variable se destruye al finalizar el bucle:

```
public static void main(String[] args) {
        int i=0;
        for (i=0; i<4; i++) ()
        System.out.println("Valor de i: " + i);
}

run:
Valor de i: 4
```

5.3.2. while

Mediante la sentencia while, la condición es evaluada, y en caso de devolver un valor true, ejecutará el bloque de instrucciones. Cabe la posibilidad de que si retorna un valor false, las sentencias contenidas no vayan a ejecutarse nunca. A no ser que lo que queramos sea la implementación de un bucle infinito, el resultado de la condición deberá ser alterado por las instrucciones contenidas en el bloque en algún momento. Ejemplo de sentencia while:

```
int i = 1;
while (i < 3) {
        System.out.println("Iteración: " + i);
        i++;
}
```

Iteración: 1

Iteración: 2

Imagen 3. Ejemplo de sentencia while.

5.3.3. do while

La sentencia do while es muy similar a la sentencia while, la única diferencia es que, en esta sentencia, el bloque de instrucciones es lo primero que se ejecuta, y después evalúa la condición. De manera que, el bloque de instrucciones al menos se ejecuta una vez siempre. Ejemplo de do while:

```
int i = 1;
do {
        System.out.println("Iteración: " + i);
        i++;
} while (i < 1);
```

Iteración: 1

Imagen 4. Ejemplo de sentencia do while.

5.4. Estructuras de salto incondicional

Debido a que estas estructuras nos dan la posibilidad de modificar el flujo natural de ejecución que sigue un programa para poder saltar a otras partes del código, su uso no se aconseja teniendo en cuenta que queremos seguir un tipo de programación estructurada, ya que estos saltos dificultan la trazabilidad y el seguimiento de los programas. Utilizando las estructuras antes vistas, es más que suficiente.

5.4.1. break continue

Utilizando la sentencia break, podemos parar la ejecución de un bucle para avanzar a la próxima instrucción que haya des-pués del bucle. A continuación, ejemplos de la sentencia de salto break:

```
while (true) {
        System.out.println("Dentro bucle 1");
        while (true) {
                System.out.println("Dentro bucle 2");
                break;
        }
}
```

Dentro bucle 1
Dentro bucle 2
Dentro bucle 1
Dentro bucle 2
…
…
…

```
int j=1;
while (true){
        System.out.println("Iteración " + j);
        if (j==3) break;
        j++;
}
System.out.println("Fin del bucle");
```

Iteración 1
Iteración 2
Iteración 3
Fin del bucle

Imagen 5. Ejemplos de la sentencia de salto break.

Mediante la sentencia continue, se detiene solamente la iteración actual y se salta a la próxima iteración del bucle, pero sin salir de este, a no ser que se haya terminado el bucle. Ejemplos de la sentencia de salto continue:

```
while (true) {
        System.out.println("Dentro bucle 1");
        while (true) {
                System.out.println("Dentro bucle 2");
                continue;
        }
}
```

```
Dentro bucle 1
Dentro bucle 2
Dentro bucle 1
Dentro bucle 2
…
…
…
```

```
for(int j=1; j<=3; j++){
        System.out.println("Iteración " + j);
        if (j<3) continue;
        System.out.println("Final del bucle");
}
System.out.println("Fuera del bucle");
```

```
Iteración 1
Iteración 2
Iteración 3
Final del bucle
Fuera del bucle
```

Imagen 6. Ejemplo de sentencia de salto continue.

Como norma general, tanto las instrucciones continue como las break suelen ir contenidas por una sentencia if, para poder indicarnos el momento en que se detendrá el bucle dependiendo de una condición. Las sentencias continue solo pueden ir dentro de un bucle, mientras que las break, además, pueden ir dentro de un switch.

5.4.2. break continue con etiquetas

Las etiquetas nos sirven para que cuando encontremos un continue o break, en vez de pasar a la siguiente iteración, se salte a la zona del código que viene identificada con dicha etiqueta. Estas etiquetas solamente se pueden utilizar para llevar a cabo saltos dentro de los bucles en los que nos encontremos. Para declarar una etiqueta se utiliza nombreEtiqueta:. Ejemplo de uso de etiquetas:

```
W: while (true) {
        System.out.priontln("Dentro de W");
        X: while (true) {
                System.out.println("Dentro de X");
                break W;
        }
}
Y: while (true) {
        System.outprintln("Dentro de Y");
        Z: while (true) {
                System.out.println("Dentro de Z");
                continue Y;
        }
}
```

```
Dentro de W
Dentro de X
Dentro de Y
Dentro de Z
Dentro de Y
Dentro de Z
Dentro de Y
Dentro de Z
Dentro de Y
Dentro de Z
Dentro de Y
Dentro de Z
…
```

Imagen 7. Ejemplo de uso de etiquetas.

5.5. Prueba y depuración de programas

Una vez completado un programa, este debe ser testado para asegurar su operatividad, cumpliendo con aquellas funciones para las que este se desarrolló, y que cuente con una cantidad de comentarios aclaratorios suficientes para permitir que otros desarrolladores lo comprendan con facilidad. Cuando se detecta un fallo en el programa, a este se le denomina bug.

La depuración es el proceso para eliminar estos errores, y para ellos se siguen una serie de pasos:

- Siguiendo el flujo del programa, e inspecciona el código mientras se trata de pronosticar el resultado, siempre que sea posible.
- Someter al programa a una gran cantidad de trabajo mediante baterías de prueba, para que, si consigue ejecutarlas correctamente, se pueda deducir que funcionará apropiadamente en un escenario real.
- Pruebas unitarias, es decir, probar por separado los diferentes módulos del programa.
- Y, por último, corregir todos aquellos errores que hayamos detectado, sabiendo que una vez corregidos, pueden generar nuevos errores.

www.universae.com