Condicionales y Lógica

Programación de Computadoras II

Abdel G. Martínez L.

Agenda

1. Operadores Relacionales
2. Operadores Lógicos
3. Sentencias Condicionales
4. Anidando y Encadenando
5. Variables Bandera
6. Sentencia ’return’
7. Capturando y Mandejando Excepciones
8. Métodos Recursivos

Operadores Relacionales

- Los **operadores relacionales** se utilizan para validar condiciones donde dos valores son iguales, o donde uno es mayor que el otro.

```
x == y
x != y
x > y
x < y
x >= y
x <= y
```

- El resultado de estos operadores puede ser un valor booleano, es decir, *true* o *false*.

Operadores Relacionales

- Ambos operadores deben ser compatibles. Es decir, no se puede mezclar un valor númerico con una cadena.
- Para evaluar cadenas se recomienda utilizar el método *equals*.

```
String fruta1 = ”Manzana”;
String fruta2 = ”Naranja”;
System.out.println(fruta1.equals(fruta2));
```

Operadores Lógicos

- Java tiene operadores lógicos: &&, || y !, los cuales corresponden a las compuertas lógicas AND, OR y NOT, respectivamente.
- Ejemplo:

```
a = true;
b = false;
x = 5
y = 5
!(a && b) == !a || !b
!(x < 5 && y == 3)
x < 1 || y != 7
```

Sentencias Condicionales

- Para escribir programas útiles, debemos validar las condiciones y reaccionar apropiadamente.
- La sentencia condicional más simple en Java es if:

```
if (x > 0) {
   System.out.println("x es positivo");
}
```

- La expresión en paréntesis se le conoce como **condición**. Si es cierto lo que está entre llaves se ejecuta. De lo contrario, se salta ese bloque de código.

Sentencias Condicionales

- Las sentencias condicionales pueden tener dos posibilidades, utilizando if y else. Cada posibilidad se llama rama:

```
if (x % 2 == 0) {
   System.out.println(”x es par”);
} else {
   System.out.println(”x es non”);
}
```

- Las llaves son opcionales cuando se tiene una única sentencia por posibilidad, pero recomiendo utilizar las llaves para evitar confusión al momento de escribir el código.

Anidando y Encadenando

- En algunas ocasiones queremos validar condiciones relacionadas y elegir una de varias acciones. Se pueden **encadenar** sentencias:

```
if (x > 0) {
   System.out.println(”x es positivo”);
} else if (x < 0){
   System.out.println(”x es negativo”);
} else {
   System.out.println(”x es cero”);
}
```

- Estas cadenas pueden ser tan largas como se desee, pero pueden ser difíciles de leer si se escapa de nuestras manos.

Anidando y Encadenando

- Otra manera es crear decisiones complejas usando anidando una condición dentro de otra.

```
if (x > 0) {
   System.out.println(”x es positivo”);
} else {
   if (x < 0){
        System.out.println(”x es negativo”);
   } else {
   System.out.println(”x es cero”);
   }
}
```

Variables Bandera

- Para almacenar valores verdaderos o falsos, usamos variables booleanas. Los resultados de las comparaciones pueden almacenarse en variables:

```
boolean banderaPar = (n % 2 == 0);
boolean banderaPositivo = (x > 0);
```

- La **variable bandera** es aquella que define la presencia o ausencia de una condición. Se pueden utiizar en condiciones:

```
if (banderaPar) {
   System.out.println(”Se valido que es numero par”);

}
```

Sentencia ’return’

- La sentencia return permite terminar un método antes de llegar a su final. Usualmente es cuando se detecta una condición de error:

```
public static void imprimirLogaritmo(double x) {
   if (x <= 0.0) {
        System.err.println(”Error: x debe ser positivo.”);
        return;
   }
   double resultadod = Math.log(x)
   System.out.println(”El log de x es ” + resultado);
}
```

- En este ejemplo la sentencia return es para salir del método sin ejecutar el resto de líneas que lo contiene.

Capturando y Manejando Excepciones

- La sentencia try contiene un bloque de programa dentro el cual una excepción puede ocurrir.

```
try {
}
```

- La sentencia catch se asocia con un bloque try. Dicho bloque se ejecuta bajo un particular tipo de error.

```
try {
} catch (exception(type) e(object)) {
}
```

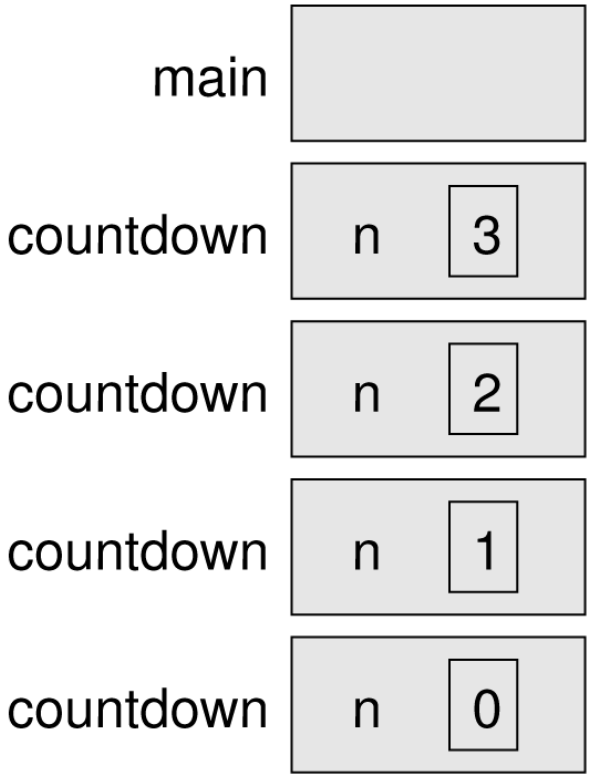
Capturando y Manejando Excepciones

```
class Ejemplo1 {
   public static void main(String args[]) {
     int num1, num2;
     try {
        num1 = 0;
        num2 = 62 / num1;
        System.out.println(”Sospechoso");
     } catch (ArithmeticException e) {
        System.out.println("Error: Division entre cero");
       }
     System.out.println(”Salvado");
   }
}
```

Métodos Recursivos

- La **recursividad** es un método que se llama a sí mismo.

```
public static void conteo(int n) {
   if (n == 0) {
        System.out.println(”Boom!”);
   } else {
        System.out.println(n);
        conteo(n - 1);
   }
}
```



¡HASTA LA PRÓXIMA CLASE!

Tema: Métodos con Valores