Lecture 4: Control Statements (Part 2)

Object Oriented Concepts and Programming (CSC244)

By

Dr. Tabbasum Naz

tabbasum.naz@ciitlahore.edu.pk

for-statement

for (*initialization*; *termination*; *increment*) {

statement(s)

}

- When using this version of the for statement, keep in mind that:
  - The *initialization* expression initializes the loop; it's executed once, as the loop begins.
  - When the *termination* expression evaluates to false, the loop terminates.
  - The *increment* expression is invoked after each iteration through the loop; it is perfectly acceptable for this expression to increment *or* decrement a value.

for-statement (Contd)

```
class ForDemo {
    public static void main(String[] args){
        for(int i=1; i<11; i++){
            System.out.println("Count is: " + i);
        }
    }
}
```

- The **output** of this program is:

```
Count is: 1
Count is: 2
Count is: 3
Count is: 4
Count is: 5
Count is: 6
Count is: 7
Count is: 8
Count is: 9
Count is: 10
```

while-statement

```
while (expression)
{
   statement(s)
}
```

Program:

```
class WhileDemo {
    public static void main(String[] args){
    int count = 1;
      while (count < 11)
{
            System.out.println("Count is: " + count);
             count++;
      }
   }
}
```

do while statement

```
do {
 statement(s)
}
while (expression);
```

Program

```
class DoWhileDemo {
public static void main(String[] args)
{
        int count = 1;
        do
        {
            System.out.println("Count is: " + count);
            count++;
         } while (count <= 11);
   }
}
```

Note: do-while loop runs at-least once.

The Switch Statement

```
public class SwitchDemo {
   public static void main(String[] args) {
       int month = 2;
      String monthString;
      switch (month)
      {
         case 1:
                  monthString = "January";
      break;
       case 2:
                  monthString = "February";
       break;
         .
         .
         .
      case 12:
      monthString = "December";
       break;
      default:
      monthString = "Invalid month";
      break;
      }
      System.out.println(monthString);
   }
}
```

The Switch Statement (Contd)

```
class SwitchDemo2 {
    public static void main(String[] args) {

        int month = 2;
        int year = 2000;
        int numDays = 0;

        switch (month) {
            case 1:
            case 3:
            case 5:
            case 7:
            case 8:
            case 10:
            case 12:
                numDays = 31;
                break;
            case 4:
            case 6:
            case 9:
            case 11:
                numDays = 30;
                break;
            case 2:
                if ( year % 400 == 0)
                    numDays = 29;
                else
                    numDays = 28;
                break;
            default:
                System.out.println("Invalid  month.");
                break;
        }
        System.out.println("Number  of Days = " + numDays);
    }
}
```

The Switch Statement (Contd)

```
// An improved version of the season program.
class Switch {
    public static void main(String args[]) {
        int month = 4;
        String season;
        switch (month) {
          case 12:
          case 1:
          case 2:
            season = "Winter";
            break;
          case 3:
          case 4:
          case 5:
            season = "Spring";
            break;
          case 6:
          case 7:
          case 8:
            season = "Summer";
            break;
          case 9:
          case 10:
          case 11:
            season = "Autumn";
            break;
          default:
            season = "Bogus Month";
        }
        System.out.println("April is in the " + season + ".");
    }
}
```

break statement

- The break statement, when executed in for, while, do-while or switch, causes immediate exit from the statement.

```
1  // Fig. 5.12: BreakTest.java
2  // break statement exiting a for statement.
3  public class BreakTest
4  {
5     public static void main( String args[] )
6     {
7        int count; // control variable also used after loop terminates
8
9        for ( count = 1; count <= 10; count++ ) // loop 10 times
10       {
11          if ( count == 5 ) // if count is 5,
12             break;         // terminate loop
13
14          System.out.printf( "%d ", count );
15       } // end for
16
17       System.out.printf( "\nBroke out of loop at count = %d\n", count );
18    } // end main
19 } // end class BreakTest
```

```
1 2 3 4
Broke out of loop at count = 5
```

Fig. 5.12 | break statement exiting a for statement.

continue statement

- The continue statement, when executed in a while, for, do-while, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.
- In while and do while statements, the program evaluates the loop-continuation test immediately after the continue statement executes.
- In a for statement, the increment expression executes, then the program evaluates the loop-continuation test.

continue statement (Contd)

```
1  // Fig. 5.13: ContinueTest.java
2  // continue statement terminating an iteration of a for statement.
3  public class ContinueTest
4  {
5     public static void main( String args[] )
6     {
7        for ( int count = 1; count <= 10; count++ ) // loop 10 times
8        {
9           if ( count == 5 ) // if count is 5,
10             continue; // skip remaining code in loop
11
12          System.out.printf( "%d ", count );
13       } // end for
14
15       System.out.println( "\nUsed continue to skip printing 5" );
16    } // end main
17 } // end class ContinueTest
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

Fig. 5.13 | continue statement terminating an iteration of a for statement.

Logical Operators

- Conditional AND (&&) Operator

expression1	expression2	expression1 && expression2
false	false	false
false	true	false
true	false	false
true	true	true

Fig. 5.14 | && (conditional AND) operator truth table.

```
if ( gender == FEMALE && age >= 65 )
   ++seniorFemales;
```

Logical Operators (Contd)

- Conditional OR (||) Operator

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true

Fig. 5.15 | || (conditional OR) operator truth table.

```
if ( ( semesterAverage >= 90 ) || ( finalExam >= 90 ) )
   System.out.println ( "Student grade is A" );
```

Logical Operators

- Logical Negation (!) Operator

expression	!expression
false	true
true	false

Fig. 5.17 | ! (logical negation, or logical NOT) operator truth table.

```
if ( ! ( grade == sentinelValue ) )
   System.out.printf( "The next grade is %d\n", grade );
```

Boolean logical AND (&) and boolean inclusive OR (|)

The boolean logical AND (&) and boolean logical inclusive OR (|) operators work identically to the && (conditional AND) and || (conditional OR) operators, with one exception: The boolean logical operators always evaluate both of their operands (i.e., they do not perform short-circuit evaluation). Therefore, the expression

```
( gender == 1 ) & ( age >= 65 )
```

evaluates age >= 65 regardless of whether gender is equal to 1. This is useful if the right operand of the boolean logical AND or boolean logical inclusive OR operator has a required side effect—a modification of a variable's value. For example, the expression

```
( birthday == true ) | ( ++age >= 65 )
```

guarantees that the condition ++age >= 65 will be evaluated. Thus, the variable age is incremented in the preceding expression, regardless of whether the overall expression is true or false.

Boolean logical exclusive OR (^)

A simple condition containing the boolean logical exclusive OR (^) operator is `true` *if and only if one of its operands is* `true` *and the other is* `false`. If both operands are `true` or both are `false`, the entire condition is `false`. Figure 5.16 is a truth table for the boolean logical exclusive OR operator (^). This operator is also guaranteed to evaluate both of its operands.

expression1	expression2	expression1 ^ expression2
false	false	false
false	true	true
true	false	true
true	true	false

Fig. 5.16 | ^ (boolean logical exclusive OR) operator truth table.

Logical Operators (Summary)

```
Conditional AND (&&)
false && false: false
false && true: false
true && false: false
true && true: true

Conditional OR (||)
false || false: false
false || true: true
true || false: true
true || true: true

Boolean logical AND (&)
false & false: false
false & true: false
true & false: false
true & true: true

Boolean logical inclusive OR (|)
false | false: false
false | true: true
true | false: true
true | true: true

Boolean logical exclusive OR (^)
false ^ false: false
false ^ true: true
true ^ false: true
true ^ true: false

Logical NOT (!)
!false: true
!true: false
```

THANK YOU