SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. Ridha Khedri

Department of Computing and Software, McMaster University
Canada L8S 4L7, Hamilton, Ontario

Acknowledgments: Material based on *Java actually: A Comprehensive Primer in Programming* (Chapter 5)

Topics Covered

1. Introduction and Learning Objectives
2. The extended assignment operators
3. The increment and decrement operatorss
4. Counter-controlled loops
   - Local variables in the *for(; ;)* loop
   - Nested *for(; ;)* loops
5. Changing control flow in loops
   - The *break* statement
   - The *continue* statement
6. Common pitfalls in loops
7. Multiple-selection statements
   - Falling through case labels

Introduction and Learning Objectives

Learning Objectives:

- Using extended assignment operators
- Using increment (++) and decrement (- -) operators
- Writing counter-controlled loops using the **for** statement
- Changing the control flow in a loop
- Avoiding common pitfalls when writing loops
- Using the multiple selection switch statement
- Changing the control flow in a switch statement

- A very common operation is computing the new value for a variable based on its old value

x= x + (valid_expression);

- We can use the extended assignment operator

x += (valid_expression);

- Java provides a number of such extended assignment operators (e.g., +=, -=, *=, /=, %=)

The extended assignment operators


```
1  // Illustrating use of extended assignment operators
   public class ExtendedAssignmentOperators {
3
     public static void main(String[] args) {
5      int i = 5, j = 10;
       double x = 5.0, y = 10.5, z = 10.0;
7      i += j;                 // i = i + (j)
       x -= y + 30.5;          // x = x - (y + 30.5)
9      j *= i + 20;            // j = j * (i + 20)
       y /= 5.0;               // y = y / (5.0)
11     z %= 3.0;               // z = z % (3.0)
       System.out.println("i : " + i);
13     System.out.println("x : " + x);
       System.out.println("j : " + j);
15     System.out.println("y : " + y);
       System.out.println("z : " + z);
17   }
   }
```

Program Output

```
i : 15
x : -36.0
j : 350
y : 2.1
z : 1.0
```

The increment and decrement operatorss

- Very often we need to increment the value in a variable by 1

```
x = x + 1;
x += 1;
x ++;
```

- We can use the decrement operator to decrement the value in a variable by 1

```
x --;
```

- The increment and decrement operators come in two flavours
  - **Post-operator:** x++ and x--
  - **Pre-operator:** ++x and --x

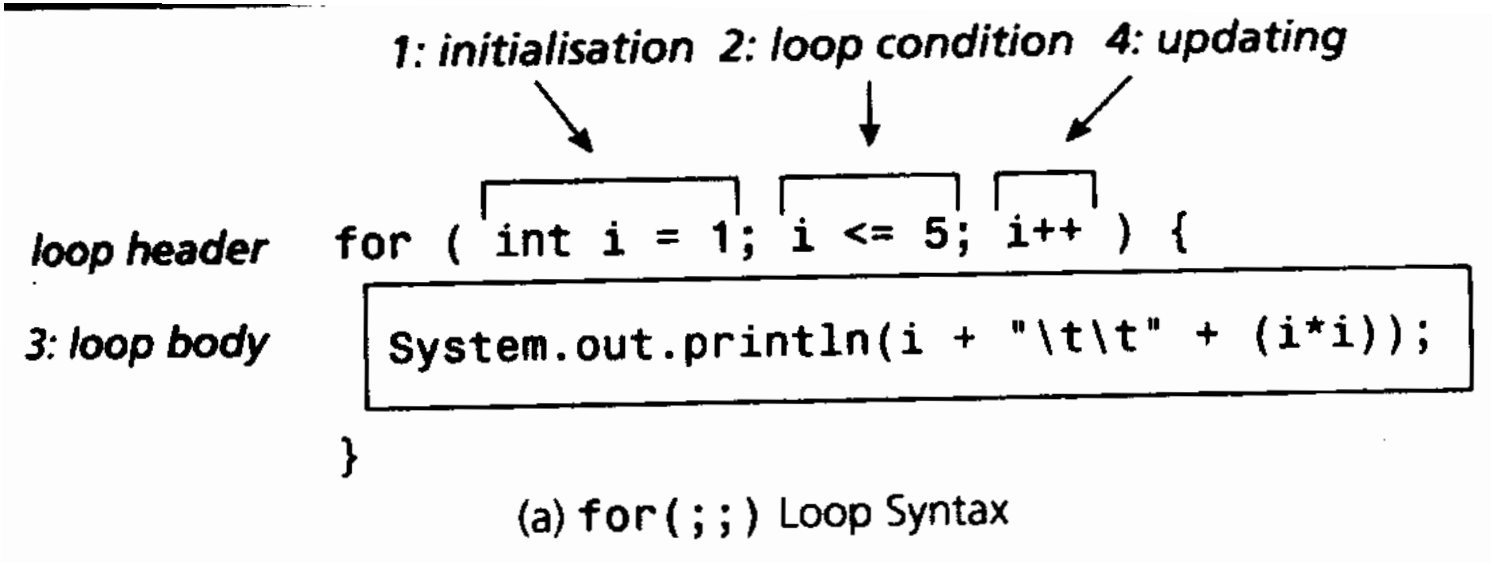
The increment and decrement operatorss


```
// Illustrating use of increment and
   decrement operators
public class IncrAndDecrOperators {

  public static void main(String[] args) {
    int i = 10, j;
    j = i++; // (1) Postfix: j gets the value
        10, and i gets the value 11
    System.out.println("j : " + j + " and i:
       " + i);
    i = 10;
    j = ++i; // (2) Prefix: j gets the value
       11, and i gets the value 11
    System.out.println("j : " + j + " and i:
       " + i);
  }
}
```

Program Output

```
j : 10 and i: 11
j : 11 and i: 11
```

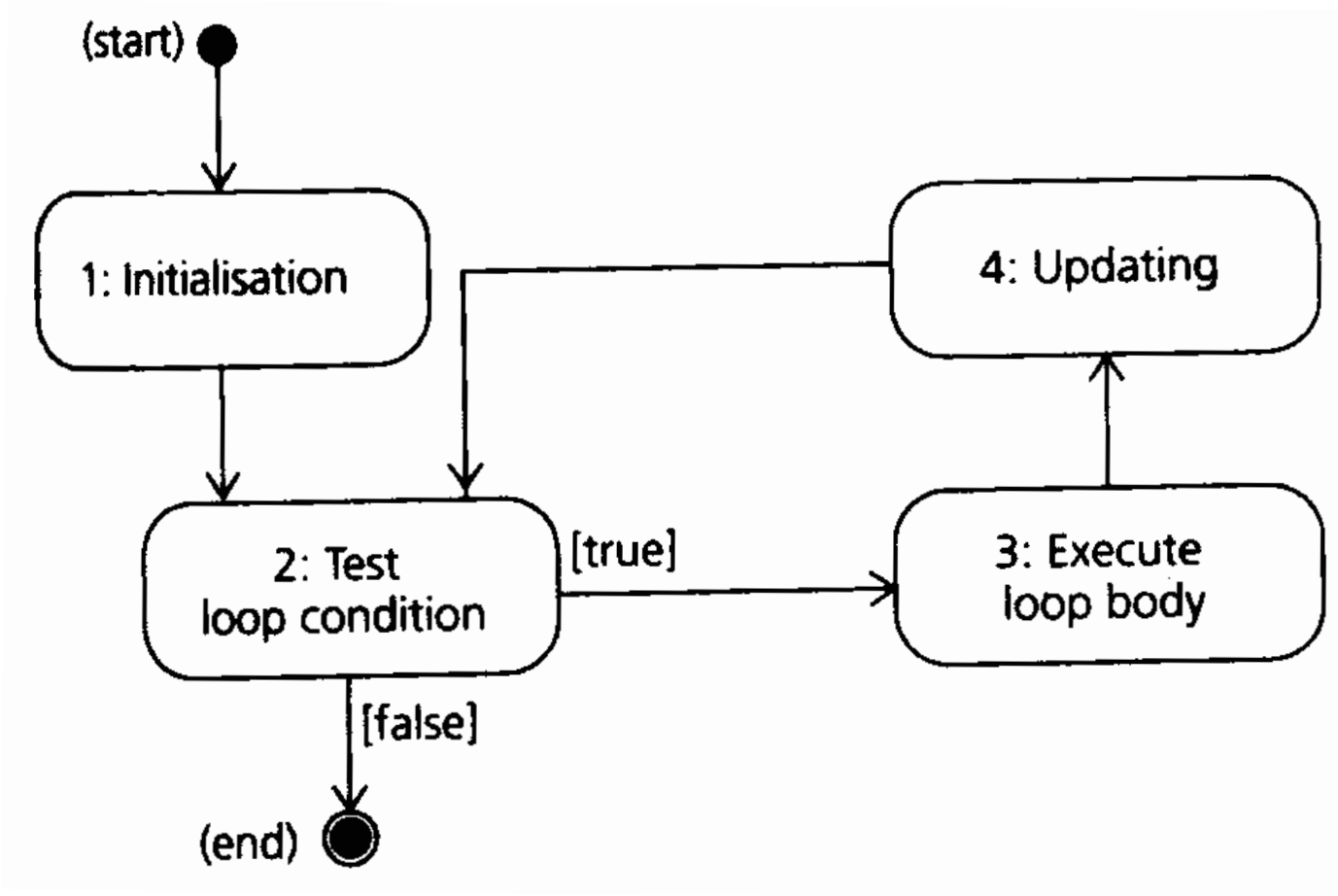
- The while statement is suitable for implementing conditional loops
- There are situations in which the number of iterations is known beforehand
- We use counter-controlled loops
- In Java, the **for(; ;)** statement is specifically designed to implement counter-controlled loops
- The loop consists of a loop header and a loop body



(a) `for(;;)` Loop Syntax

```
1: initialisation    int i = 1;

2: loop condition    while ( i <= 5 ) {

3: loop body           System.out.println(i + "\t\t" + (i*i));

4: updating            i++;

                     }
```

(b) Equivalent **while** Loop



Counter-controlled loops


```
import java.util.Scanner;
// Program spells out a telephone number: if statement version
public class Telephone {
  public static void main(String[] args) {
    System.out.print("Enter the telephone number (as 55584152): ");
    Scanner keyboard = new Scanner(System.in);
    String inputPhoneString = keyboard.next(); // (1)
    String outputPhoneString = "";
    for (int i = 0; i < inputPhoneString.length(); i++) { // (2)
      char aChar = inputPhoneString.charAt(i);
      if (aChar == '1') {
        outputPhoneString += "one ";
      } else if (aChar == '2') {
        outputPhoneString += "two ";
      } else if (aChar == '3') {
        outputPhoneString += "three ";
      } else if (aChar == '4') {
        outputPhoneString += "four ";
      } else if (aChar == '5') {
        outputPhoneString += "five ";
      } else if (aChar == '6') {
        outputPhoneString += "six ";
      } else if (aChar == '7') {
        outputPhoneString += "seven ";
      } else if (aChar == '8') {
        outputPhoneString += "eight ";
      } else if (aChar == '9') {
        outputPhoneString += "nine ";
      } else if (aChar == '0') {
        outputPhoneString += "zero ";
      } else {
        System.out.println(aChar + " is not a digit!");
      }
    }
    System.out.println(outputPhoneString);
  }
}
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

```
1 for (int j = 1; j<=5; j++){
    int m = 20;
3   System.out.println(j + "\t" + (j*m));
  }
5
  System.out.println("j has the value : " + j); //Compile-time error
7 System.out.println("m has the value : " + m); //Compile-time error
```

```
1
  int j, m;
3
  for (int j = 1; j<=5; j++){
5   m = 20;
    System.out.println(j + "\t" + (j*m));
7 }

9 System.out.println("j has the value : " + j); // Prints 6
  System.out.println("m has the value : " + m); // Prints 20
```

- We are not limited to increments of 1
- We can specify other increments using extended assignment operators
- Care must be taken to initialize and increment the loop variable correctly, and to ensure that the loop condition is formulated correctly
- We can write **for(; ;)** loops that can count backwards
- We can have nested **for(; ;)** loops

```
// Illustrating various uses of the for(;;) loop
public class ForExamples {
  public static void main(String[] args) {

    // (1) Simple for loop
    // Print the square of numbers from 1 to 5.
    System.out.println("Square of numbers from 1 to 5");
    for (int i = 1; i <= 5; i++) {
      System.out.println(i + "\t\t" + (i*i));
    }

    // (2) Increment by other than 1
    // Print the square of odd numbers from 1 to 5.
    System.out.println("Square of odd numbers from 1 to 5");
    for (int i = 1; i <= 5; i += 2) {
      System.out.println(i + "\t\t" + (i*i));
    }

    // (3) "Backwards" for loop
    // Print the square of numbers from 5 to 1.
    System.out.println("Square of numbers from 5 to 1");
    for (int i = 5; i > 0; i--) {
      System.out.println(i + "\t\t" + (i * i));
    }

    // (4) Nested for loops
    // Print the multiplication tables from 1 to 5.
    for (int i = 1; i <= 5; i++) {
      System.out.println("Multiplication table for " + i);
      for (int j = 1; j <= 10 ; j++) {
        System.out.printf("%2d x %2d = %2d%n", i, j, (i*j));
      }
    }
  }
}
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Program Output

```
Square of numbers from 1 to 5
1 1
2 4
3 9
4 16
5 25
Square of odd numbers from 1 to 5
1 1
3 9
5 25
Square of numbers from 5 to 1
5 25
4 16
3 9
2 4
1 1
Multiplication table for 1
 1 x  1 =  1
 1 x  2 =  2
 1 x  3 =  3
 1 x  4 =  4
 1 x  5 =  5
 1 x  6 =  6
 1 x  7 =  7
 1 x  8 =  8
 1 x  9 =  9
 1 x 10 = 10
Multiplication table for 2
 2 x  1 =  2
 2 x  2 =  4
 2 x  3 =  6
```

- An **if-else** statement can be used to control which statements are executed (in each iteration)

```
1
  boolean done = false;
3
  for (int i = 1; !done && i <=5; i++){
5   if (i==4) {
      System.out.println("Terminates the loop when i is equal to " + i);
7     done = true;
    } else {
9     system.out.println(i + "\t\t" + (i*i));
      }
11 }
```

- The **break** and the **continue** statements in Java have special significance when they are executed in any loop
- **break** and the **continue** statements change the normal execution of a loop
- Execution of the break statement in a loop body results
  - in the rest of the loop body being skipped
  - the loop terminates
  - execution continues after the loop
- Using a **break** statement in a loop introduces another exit from the loop
- HOWEVER, it makes the program unstructured

- Execution of the **continue** statement in a loop body results in the rest of the loop body being skipped
- What happens next depends on the loop statement
- In a While-loop, execution will continue with the testing of the loop condition
- In a *for (; ;)* loop, execution will continue with the updating of the loop variable

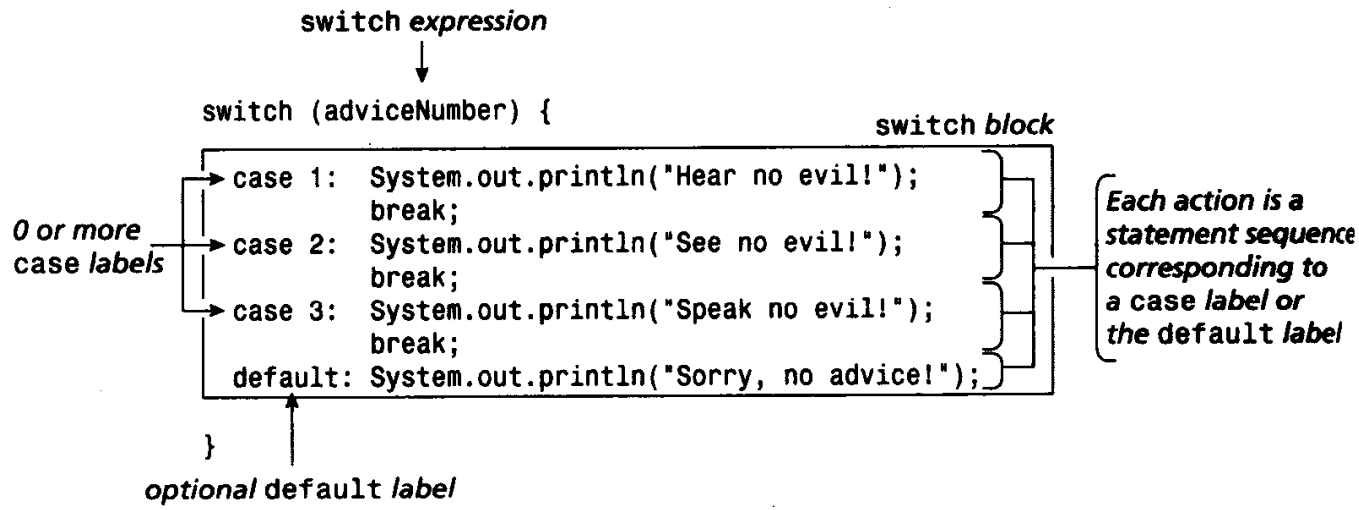
Changing control flow in loops

The *continue* statement

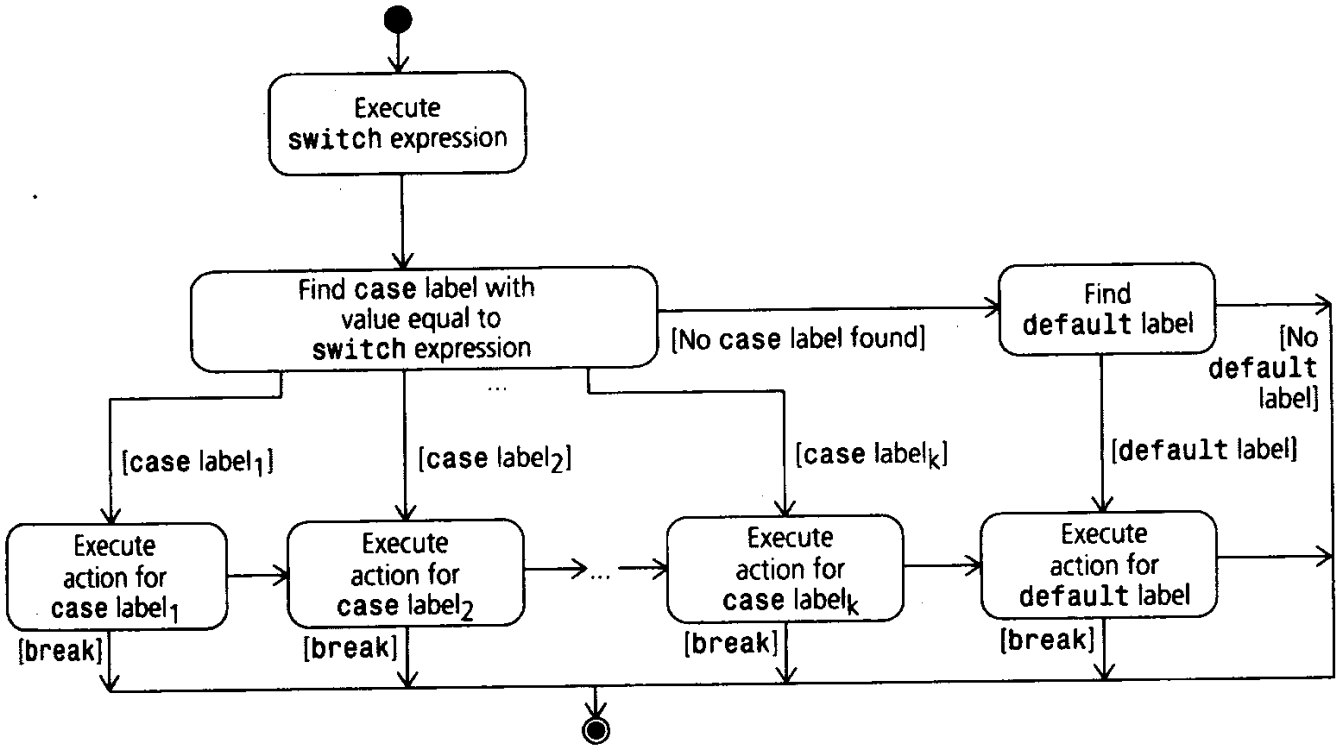

```
1  // Illustrating use of break and continue statements
   public class BreakAndContinue {
3    public static void main(String[] args) {

5      // for loop with a break statement.
       System.out.println("Square of numbers from 1 to 5");
7      for (int i = 1; i <= 5; i++) {
                                        // (1)
         if (i == 4) {
                                                  // (2)
9          System.out.println("Terminates the loop when i is
                 equal to " + i);
           break;

                // (3)
11       }
         // Never executed when i == 4, as the loop
              terminates.
13       System.out.println(i + "\t\t" + (i * i));
                           // (4)
       }
15
       // for loop with a continue statement.
17     System.out.println("Square of numbers from 1 to 5");
              // (5)
       for (int i = 1; i <= 5; i++) {
                                        // (6)
19       if (i == 3) {
                                                  // (7)
           System.out.println("Skips rest of the loop body"
                +
21                            " when i is equal to " + i);
           continue;
                                                        //
                (8)
23       }
         // Skipped when i == 3, otherwise executed.
25       System.out.println(i + "\t\t" + (i * i));
                           // (9)
       }
27   }
}
```

```
Square of numbers from 1 to 5
1    1
2    4
3    9
Terminates the loop when i is equal to 4
Square of numbers from 1 to 5
1    1
2    4
Skips rest of the loop body when i is equal to 3
4    16
5    25
```

- Infinite loop
- One-off errors (the loop body is executed either one more or one less time)
- Errors in initialization
- Errors in the loop condition
- Optimizing loops



- The **default** label is optional in a switch statement
- If no case label value is found and no default label is specified, the switch statement is skipped


Execute switch expression
Find case label with value equal to switch expression
[No case label found]
Find default label
[No default label]
[case label1]
[case label2]
[case labelk]
[default label]
Execute action for case label1
Execute action for case label2
Execute action for case labelk
Execute action for default label
[break]
[break]
[break]
[break]

```
1  import java.util.Scanner;
   // Program spells out a telephone number: switch
        statement version
3  public class TelephoneII {
     public static void main(String[] args) {
5      System.out.print("Enter the telephone number (as
           55584152): ");
       Scanner keyboard = new Scanner(System.in);
7      String inputPhoneString = keyboard.next();
       String outputPhoneString = "";
9      for (int i = 0; i < inputPhoneString.length(); i++) {
         char aChar = inputPhoneString.charAt(i);
11       switch (aChar) {
           case '1':
13           outputPhoneString += "one ";
             break;
15         case '2':
             outputPhoneString += "two ";
17           break;
           case '3':
19           outputPhoneString += "three ";
             break;
21         case '4':
             outputPhoneString += "four ";
23           break;
           case '5':
25           outputPhoneString += "five ";
             break;
27         case '6':
             outputPhoneString += "six ";
29           break;
           case '7':
31           outputPhoneString += "seven ";
             break;
33         case '8':
             outputPhoneString += "eight ";
35           break;
           case '9':
37           outputPhoneString += "nine ";
             break;
39         case '0':
             outputPhoneString += "zero ";
41           break;
           default:
43           System.out.println(aChar + " is not a digit!");
         } // end switch
45     } // end for
       System.out.println(outputPhoneString);
47   }
}
```

```
1  Enter the telephone number (as 55584152): 9058787754
2  nine zero five eight seven eight seven seven five four
```

- We **need not use** the block notation, { }, to enclose the sequence of statements for an action corresponding to a case label

- Using { } for the statement of a case label does not affect the control flow

- HOWEVER, the execution of a **break** statement as part of an action can terminate the switch statement **(when it is the last statement for an action)**
- *break* is not part of the syntax for a switch
- However, it helps to prevent unintentional fall through in a switch **(when used well)**

```
import java.util.Scanner;
public class Seasons {
  public static void main(String[] args) {
    System.out.print("Enter a month number [1-12]: ");
    Scanner keyboard = new Scanner(System.in);
    int monthNumber = keyboard.nextInt();
    switch (monthNumber) {
      case 12: case 1: case 2:// Common action for several case labels
        System.out.println("It's winter!");
        break;
      case 3: case 4: case 5:
        System.out.println("It's spring!");
        break;
      case 6: case 7: case 8:
        System.out.println("It's summer!");
        break;
      case 9: case 10: case 11:
        System.out.println("It's autumn!");
        break;
      default:
        System.out.println(monthNumber + " is not a valid month number!"
            );
    } // end switch
  }
}
```

Program Output

```
Enter a month number [1-12]: 6
It's summer!
```