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 3)

Topics Covered

1. Introduction and Learning Objectives
2. Boolean expressions
   - Boolean primitive data type
   - Relational Operators
   - Understanding relational operators
   - Logical operators
   - Precedence for logical operators
   - Short-circuit evaluation
   - Using Boolean expressions to control flow of execution
3. Control flow in selection statements
   - Simple selection statement
   - Blocks of statements
   - Local variables in a block
   - Selection statement if-else
   - Nested selection statements
   - Chaining if-else statements
4. Control flow in loop statements
   - Pre-test loop: While
   - Post-test loop: do-while
   - Nested loops
5. Assertions as a testing technique
   - Making assertions
   - Assertions as a testing technique

Introduction and Learning Objectives

- We used simple statements (executed sequentially)
- We would like to have more control the order in which statements are executed
- We will introduces selection statements and loops
- Selection statements enable us to define multiple actions that are guarded by conditions
- Loops enable us to execute the same statements repeatedly, dependent on a condition being satisfied
- Conditions controlling the execution flow are specified as Boolean expressions

Learning Objectives:

- Comparing values using relational operators
- Evaluation of expressions with logical operators
- Selecting statements to execute (if, if-else)
- Executing statements repeatedly using loops (while, do-while)
- Verifying expected program properties with assertions

Boolean primitive data type

- The primitive data type **boolean** in Java defines two Boolean values: true, and false
- We can declare variables of the type boolean and assign Boolean values to them:

boolean itemIsOnSale = true;

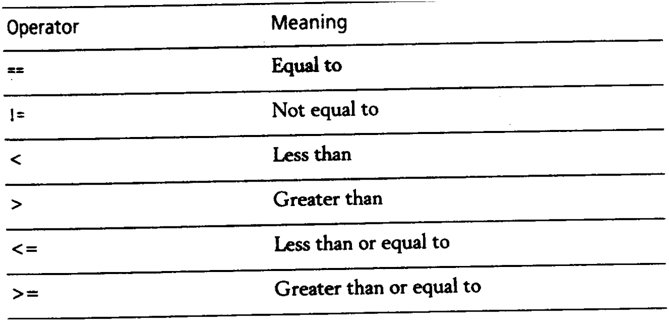
- We will use Boolean values to facilitate decision making over the actions that should be executed

Program Control Flow

Boolean expressions

Relational Operators

A relational operator enables us to compare values



Operator	Meaning
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

- The following expression in Java is a Boolean expression:

  numHours >= 37.5

- We can assign the value of a Boolean expression to a Boolean variable:

  boolean workedOvertime = numHours >= 37.5;

- Relational operators have higher precedence than the assignment operator =
  - first the Boolean expression is evaluated
  - then, the assignment is performed

heads + tails == tosses

- The operands of a relational operator can be any arithmetic expression
- Arithmetic operators have higher precedence than relational operators

((heads + tails) == tosses)

It is a common mistake to confuse the **equality operator** == with the **assignment operator** =

Program Control Flow
Boolean expressions
Understanding relational operators

```java
// Testing relational operators in boolean expressions.
public class TestRelationalOperators {
  public static void main(String[] args) {
    // Tests for integers.
    System.out.printf("Expression           Expected     Calculated%n");
    System.out.printf("%-20s%-12s%-12s%n", "3 == 3", true, (3 == 3));
    System.out.printf("%-20s%-12s%-12s%n", "3 != 3", false, (3 != 3));
    System.out.printf("%-20s%-12s%-12s%n", "7 > 4", true, (7 > 4));
    System.out.printf("%-20s%-12s%-12s%n", "7 < 4", false, (7 < 4));
    System.out.printf("%-20s%-12s%-12s%n", "6 <= 6", true, (6 <= 6));
    System.out.printf("%-20s%-12s%-12s%n", "6 >= 6", true, (6 >= 6));
  }
}
```

Program Output

```
Expression          Expected    Calculated
3 == 3              true        true
3 != 3              false       false
7 > 4               true        true
7 < 4               false       false
6 <= 6              true        true
6 >= 6              true        true
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Boolean expressions

Boolean primitive data type

Relational Operators

Understanding relational operators

Logical operators

Precedence for logical operators

Short-circuit evaluation

Using Boolean expressions to control flow of execution

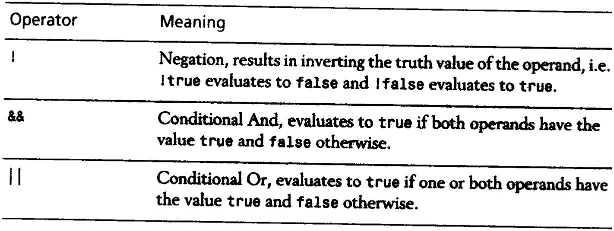
Control flow in selection statements

Control flow in

Program Control Flow
Boolean expressions
Logical operators

We can combine Boolean expressions by means of logical operators

boolean payBonus = workedOvertime || salesAboveAverage;



Operator	Meaning
!	Negation, results in inverting the truth value of the operand, i.e. !true evaluates to false and !false evaluates to true.
&&	Conditional And, evaluates to true if both operands have the value true and false otherwise.
||	Conditional Or, evaluates to true if one or both operands have the value true and false otherwise.

Truth tables for ||, && , and ! + Discuss DeMorgan Laws

Program Control Flow
Boolean expressions
Precedence for logical operators

- The precedence of the logical operators is as follows

! has higher precedence than &&

&& has higher precedence than ||

Example

```
boolean b1 = false, b2 = false, b3 = true;


System.out.printf(”b1 || ! b2 && b3 evaluates to
%s%n”, b1 || ! b2 && b3);


System.out.printf(”(b1 || (( ! b2) && b3)) evaluates
to %s%n”, (b1 || (( ! b2) && b3)));
```

Precedence for logical operators

- The logical operators have lower precedence than the relational operators

$$\textbf{weekday} >= \textbf{6} \; || \; \textbf{weekday} == \textbf{3}$$

$$(\textbf{weekday} >= \textbf{6}) \; || \; (\textbf{weekday} == \textbf{3})$$

Program Control Flow

Boolean expressions

Precedence for logical operators

```
public class RelationalOperatorsPrec {
  public static void main(String[] args) {
    int weekday1 = 4;
    int weekday2 = 6;

    System.out.printf("Weekday number is %d%n", weekday1);
    System.out.printf("Wednesday, Saturday or Sunday: %s%n", weekday1 >=
          6 || weekday1 == 3);
    System.out.println();
    System.out.printf("Weekday number is %d%n", weekday2);
    System.out.printf("Wednesday, Saturday or Sunday: %s%n", weekday2 >=
          6 || weekday2 == 3);
  }
}
```

Program Output

```
Weekday number is 4
Wednesday, Saturday or Sunday: false

Weekday number is 6
Wednesday, Saturday or Sunday: true
```

Short-circuit evaluation

- The operands in a Boolean expression are normally evaluated from left to right
- Note: the evaluation of a Boolean expression ends as soon as the value of the expression can be determined
- This is called **short-circuit evaluation**

  $$\mathbf{(4 == 3)}\ \&\&\ \mathbf{(3 < 4)}$$

  will first evaluate to: **false** && $\mathbf{(3 < 4)}$
- && returns false if one of its operands is false
- We do not need to evaluate $(3 < 4)$
- Another example: $\mathbf{(4 > 3)}\ ||\ \mathbf{(5 < 4)}$

Using Boolean expressions to control flow of execution

- We often want to perform different actions depending on whether a given condition is satisfied

- The condition can be formulated as a Boolean expression

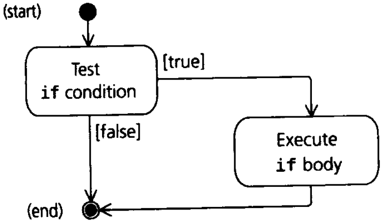
- The statement that allows the program to select a path of execution from others is called a **selection statement**

- Other types of problems require that certain actions be executed repeatedly (**repetition statement**)

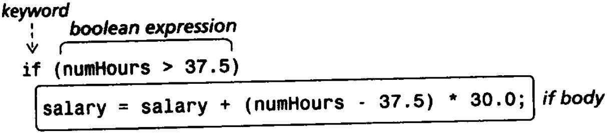
Simple selection statement

- A simple selection statement performs an action if a given condition is satisfied
- The condition is a Boolean expression
- If the Boolean expression evaluates to true, the action in the if body is executed
- If the expression evaluates to false, the action in the if body is skipped

Program Control Flow
Control flow in selection statements
Simple selection statement



(a) Control flow: if statement



```
if (numHours > 37.5)
    salary = salary + (numHours - 37.5) * 30.0;
```

Program Control Flow
Control flow in selection statements
Simple selection statement

```java
// Calculating weekly salary, version 1.
import java.util.Scanner;
public class Salary1 {
  public static void main(String[] args) {
    final double NORMAL_WORKWEEK = 37.5;

    // Read the number of hours worked this week.
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of hours worked [decimal number]:
          ");
    double numHours = keyboard.nextDouble();

    // Calculate the weekly salary and print it to the terminal window.
    double salary = 750.0;                                 // (1) weekly salary
    if (numHours > NORMAL_WORKWEEK)                                          //
         (2)
      salary = salary + (numHours - NORMAL_WORKWEEK) * 30.0;                 //
           (3)
    System.out.printf("Salary for %.1f hours is %.2f USD%n",
                        numHours, salary);                                   //
                              (4)
  }
}
```

Program Output

```
Enter the number of hours worked [decimal number]: 37.5
Salary for 37.5 hours is 750.00 USD
```

Program Control Flow
Control flow in selection statements
Simple selection statement

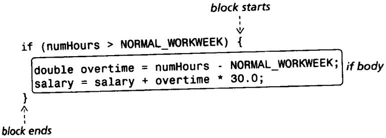
```java
// Calculating weekly salary, version 1.
import java.util.Scanner;
public class Salary1 {
  public static void main(String[] args) {
    final double NORMAL_WORKWEEK = 37.5;

    // Read the number of hours worked this week.
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of hours worked [decimal number]:
          ");
    double numHours = keyboard.nextDouble();

    // Calculate the weekly salary and print it to the terminal window.
    double salary = 750.0;                              // (1) weekly salary
    if (numHours > NORMAL_WORKWEEK)                                    //
         (2)
      salary = salary + (numHours - NORMAL_WORKWEEK) * 30.0;          //
           (3)
    System.out.printf("Salary for %.1f hours is %.2f USD%n",
                      numHours, salary);                               //
                            (4)
  }
}
```

Program Output

```
Enter the number of hours worked [decimal number]: 45.5
Salary for 45.5 hours is 990.00 USD
```

- A sequence of statements can be enclosed in curly brackets { }
- A sequence of statements between "{" and "}" is a block of statements
- A block is called a compound statement
- A compound statement can be used anywhere that a single statement can be used

Program Control Flow
Control flow in selection statements
Local variables in a block



```
if (numHours > NORMAL_WORKWEEK) {
   double overtime = numHours - NORMAL_WORKWEEK;
   salary = salary + overtime * 30.0;
}
```

block starts
if body
block ends

- We can define new variables inside a block
- It is then called a **local variable to the block**
- A local variables can only be accessed inside the block
- The part of the program where such a variable can be accessed is called its **scope**
- When it is not accessible, it is **out of scope**

Program Control Flow
Control flow in selection statements
Local variables in a block

```
// Calculating weekly salary, version 1b.
import java.util.Scanner;
public class Salary1b {
  public static void main(String[] args) {
    final double NORMAL_WORKWEEK = 37.5;

    // Read the number of hours worked this week.
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of hours worked [decimal number]:
         ");
    double numHours = keyboard.nextDouble();

    // Calculate the weekly salary and print it to the terminal window.
    double salary = 750.0;                           // (1) weekly salary
    if (numHours > NORMAL_WORKWEEK) {                // if body is a
        block
      double overtime = numHours - NORMAL_WORKWEEK; // local variable
      salary = salary + overtime * 30.0;
    }
    System.out.printf("Salary for %.1f hours is %.2f USD%n",
                      numHours, salary);
//  System.out.printf("Number of hours overtime: %.1f%n", overtime); //
     (1)
  }
}
```

Program Output

```
Enter the number of hours worked [decimal number]: 39.5
Salary for 39.5 hours is 810.00 USD
```

Program Output

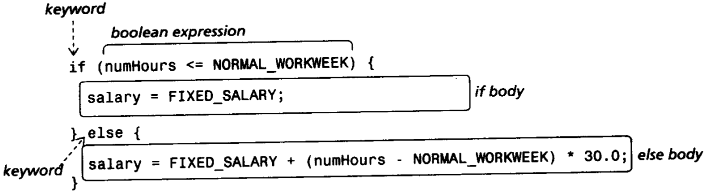
```
Enter the number of hours worked [decimal number]: 35.5
Salary for 35.5 hours is 750.00 USD
```

Program Control Flow

Control flow in selection statements

Selection statement if-else

- We often need to choose between two alternative actions
- Java offers an **if-else statement** for this purpose



```
if (numHours <= NORMAL_WORKWEEK) {
   salary = FIXED_SALARY;
} else {
   salary = FIXED_SALARY + (numHours - NORMAL_WORKWEEK) * 30.0;
}
```

SFWR ENG/COMP SCI 2S03 Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Boolean expressions

Control flow in selection statements

Simple selection statement

Blocks of statements

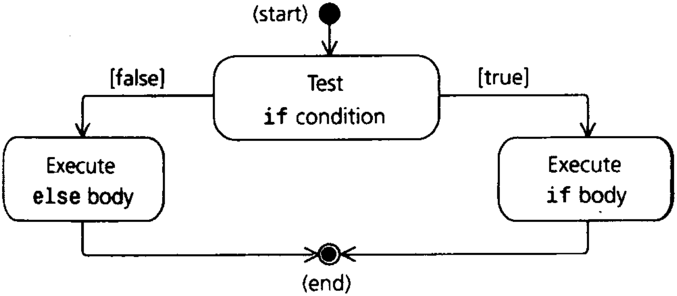
Local variables in a block

Selection statement if-else

Nested selection statements

Chaining if-else statements

Control flow in loop statements


(start)
[false]
Test
if condition
[true]
Execute
else body
Execute
if body
(end)

Program Control Flow

Control flow in selection statements

Selection statement if-else

```java
// Calculating weekly salary, version 2.
import java.util.Scanner;
public class Salary2 {
  public static void main(String[] args) {
    final double NORMAL_WORKWEEK = 37.5;
    final double FIXED_SALARY = 750.0;

    // Read the number of hours worked this week.
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of hours worked [decimal number]: ");
    double numHours = keyboard.nextDouble();

    //  Calculate the weekly salary and print it to the terminal window.
    double salary = 0.0;                                     // weekly salary
    if (numHours <= NORMAL_WORKWEEK) {                       // (1)
      salary = FIXED_SALARY;                                 // (2) if body
    } else {                                                 // (3)
      salary = FIXED_SALARY +
               (numHours - NORMAL_WORKWEEK) * 30.0;          // (4) else body
    }
    System.out.printf("Salary for %.1f hours is %.2f USD%n",
                      numHours, salary);
  }
}
```

Program Control Flow
Control flow in selection statements
Nested selection statements

```
// Calculating weekly salary, version 3.
import java.util.Scanner;
public class Salary3 {
  public static void main(String[] args) {
    final double NORMAL_WORKWEEK = 37.5;
    final double FIXED_SALARY = 750.0;

    // Read the number of hours worked this week.
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of hours worked [decimal number]:
         ");
    double numHours = keyboard.nextDouble();

    // Calculate the weekly salary and print it to the terminal window.
    double salary = 0.0;
    if (numHours <= NORMAL_WORKWEEK) {                 // (1) if statement
      salary = FIXED_SALARY;                           // (2) if body
    } else {                                           // (3) else body
      salary = FIXED_SALARY + (numHours - NORMAL_WORKWEEK) * 30.0; //
           (4)
      if (numHours > 42.0) {                           // (5) nested if
           statement
        salary = salary + 100.0;                       // (6)
      }
    }                                                  // (7)
    System.out.printf("Salary for %.1f hours is %.2f USD%n",
                      numHours, salary);
  }
}
```

SFWR ENG/COMP SCI 2S03
Principles of Programming

Dr. R. Khedri

Intro. & Learning Objectives

Boolean expressions

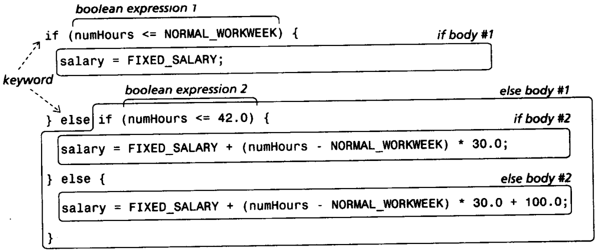
Control flow in selection statements

- Simple selection statement
- Blocks of statements
- Local variables in a block
- Selection statement if-else
- **Nested selection statements**
- Chaining if-else statements

Control flow in loop statements

Program Control Flow
Control flow in selection statements
Chaining if-else statements

- **Chaining if-else:** The else (, or true) body in an if-else statement can be another if-else statement

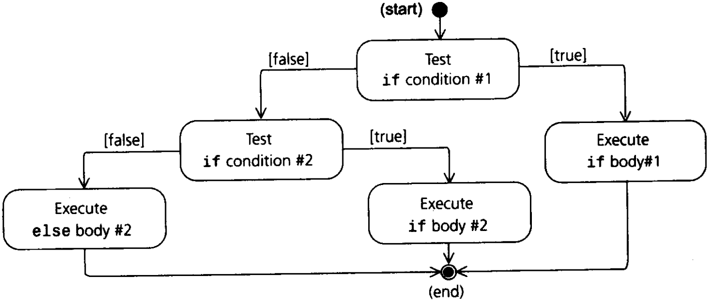


```
if (numHours <= NORMAL_WORKWEEK) {
    salary = FIXED_SALARY;
} else if (numHours <= 42.0) {
    salary = FIXED_SALARY + (numHours - NORMAL_WORKWEEK) * 30.0;
} else {
    salary = FIXED_SALARY + (numHours - NORMAL_WORKWEEK) * 30.0 + 100.0;
}
```

Program Control Flow

Control flow in selection statements

Chaining if-else statements



Program Control Flow

Control flow in selection statements

Chaining if-else statements

- When nesting selection statements, we must be careful to ensure that the program logic is correct

```java
import java.util.Scanner;

public class NestingIfElse {
  public static void main(String[] args) {
    int temperature;
    // Read the temperate.
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the temperate: ");
    temperature = keyboard.nextInt();

    if (temperature < 0) { // Temperature in ]MinValue, 0[
      if (temperature < −200) {// Temperature in ]MinValue, −200[
        System.out.println("It is really too cold");
    }
      else { // Temperature in [−200, 0[
        System.out.println("It is cold");
      }

  }
    else { // Temperature in [0, MaxValue[
      if (temperature < 70) {// Temperature in [0, 70[
        System.out.println("It is warm");
      }
      else { // Temperature in [70, 10000[
        if (temperature < 10000) {
          System.out.println("It is too hot");
        }
        else {// Temperature in [10000, MaxValue[
          System.out.println("It is hell");
        }

      }

    }
  }
}
```

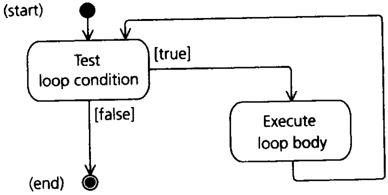
Program Control Flow
Control flow in selection statements
Chaining if-else statements

```java
// Calculating weekly salary, version 4.
import java.util.Scanner;
public class Salary4 {
  public static void main(String[] args) {
    final double NORMAL_WORKWEEK = 37.5;
    final double FIXED_SALARY = 750.0;

    // Read the number of hours worked this week.
    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of hours worked [decimal number]: ");
    double numHours = keyboard.nextDouble();

    // Calculate the weekly salary and print it to the terminal window.
    double salary = 0.0;
    if (numHours <= NORMAL_WORKWEEK) {                                    // (1)
      salary = FIXED_SALARY;                                              // (2)
    } else if (numHours <= 42.0) {                                        // (3)
      salary = FIXED_SALARY + (numHours - NORMAL_WORKWEEK) * 30.0;        // (4)
    } else {
      salary = FIXED_SALARY +
               (numHours - NORMAL_WORKWEEK) * 30.0 + 100.0;               // (5)
    }
    System.out.printf("Salary for %.1f hours is %.2f USD%n",
                      numHours, salary);
  }
}
```

Program Output

```
Enter the number of hours worked [decimal number]: 42.5
Salary for 42.5 hours is 1000.00 USD
```

- A loop statement can-be used to execute an action repeatedly
- The action is specified in the loop body
- The action can consist of zero or more statements
- Each execution of the loop body is called an iteration



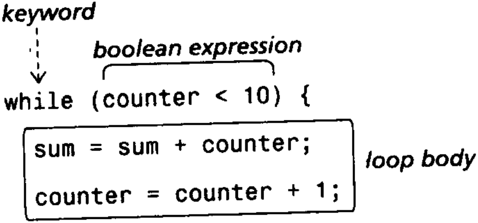
(b) Control flow: loop

Program Control Flow

Control flow in loop statements

Pre-test loop: While

- The loop executes the loop body for as long as a given loop condition is satisfied
- The condition is specified as a Boolean expression
- In a while statement, the loop condition is tested before the loop body is executed
- This kind of loop called a pre-test loop

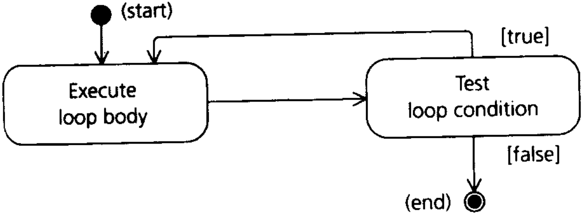


```
while (counter < 10) {
  sum = sum + counter;
  counter = counter + 1;
```

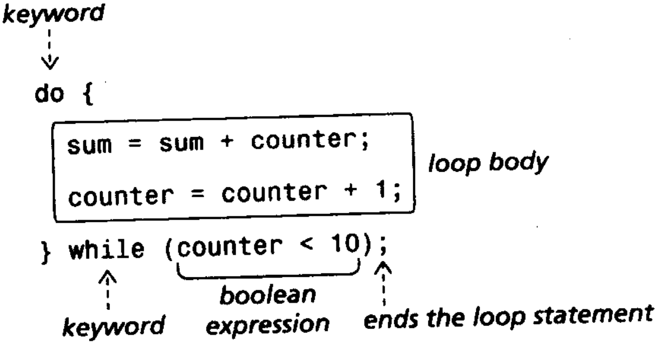
- The execution of a loop body MUST at some point affect the loop condition
- OTHERWISE, the loop **will never terminate**: infinite loop
- Should an undesirable infinite loop occur in a program, the program will need to be terminated explicitly
- On most platforms, pressing the key combination **CTRL - C** terminates program execution

- A **do-while** loop evaluates the loop condition after the loop body has been executed

- It is a post-test loop



Program Control Flow
Control flow in loop statements
Post-test loop: do-while



```
do {
    sum = sum + counter;
    counter = counter + 1;
} while (counter < 10);
```

Program Control Flow
Control flow in loop statements
Post-test loop: do-while

```
// Adding a series of integers read from the keyboard.
import java.util.Scanner;
public class IntegerAddition {
  public static void main(String[] args) {

    Scanner keyboard = new Scanner(System.in);
    System.out.print("Enter the number of integers to add [integer]: ");
    int totalNumbers = keyboard.nextInt();   // (1) No. of integers to
         add
    keyboard.nextLine();                     // Skip rest of input

    int numberCounter = 0;                   // Numbers read so far
    int sum = 0;                             // Sum of numbers so far

    while (numberCounter < totalNumbers) {                   // (2)
      System.out.print("Enter the next number [integer]: ");
      int nextInteger = keyboard.nextInt();  // Read the next number
      keyboard.nextLine();
      sum = sum + nextInteger;
      numberCounter = numberCounter + 1;
    }                                                        // (3)

    System.out.printf("The sum of %d integers is %d%n",
                      numberCounter, sum);                   // (4)
  }
}
```

Program Output

```
Enter the number of integers to add [integer]: 3
Enter the next number [integer]: 12
Enter the next number [integer]: 34
Enter the next number [integer]: 567
The sum of 3 integers is 613
```

Program Control Flow
Control flow in loop statements
Nested loops

```
// Printing a multiplication table using nested loops.
public class NestedLoops {
  public static void main(String[] args) {
    int number = 1, limit = 10;
    while (number <= limit) {            // Outer loop
      int times = 1;
      while (times <= limit) {           // Inner loop
        int product = number * times;
        System.out.println(number + " x " + times + " = " + product);
        times = times + 1;
      }
      number = number + 1;
    }
  }
}
```

Program Output

```
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
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
```

- Sometimes, we want to make sure that a program satisfies a certain assumption at a given point
- The code for an assumption defines an **assertion**
- The **assert** statement allows us to specify an assertion about the program's behaviour
- The assumption is written as a Boolean expression
- The Boolean expression is evaluated during program execution
- If the expression evaluated to **false**, then
  - an error message is generated
  - the execution is aborted

Using **java -ea FloatingPointArea3**

```java
// Using assertions to verify user input and calculated values.
import java.util.Scanner;
public class FloatingPointArea3 {
  public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);

    // Read rectangle dimensions
    System.out.print("Enter the rectangle length [decimal number]: ");
    double length = keyboard.nextDouble();
    keyboard.nextLine();
    System.out.print("Enter the rectangle width [decimal number]: ");
    double width = keyboard.nextDouble();

    // Validate user input
    assert length > 0.0 : "The length of the rectangle must be > 0.0";//
          (1)
    assert width > 0.0 : "The width of the rectangle must be > 0.0";  //
          (2)

    double area = length * width; // Calculate area of the rectangle

    // Print the correct answer
    System.out.printf(
      "A rectangle of length %.2f cm. and width %.2f cm. has" +
      " area %.2f sq. cm.%n",
      length, width, area);
  }
}
```

Program Output

```
Enter the rectangle length [decimal number]: -4
Enter the rectangle width [decimal number]: 1.4
Exception in thread "main" java.lang.AssertionError: The length of the rectangle must be > 0.0
at FloatingPointArea3.main(FloatingPointArea3.java:15)
```

Using **java FloatingPointArea3**

```
// Using assertions to verify user input and calculated values.
import java.util.Scanner;
public class FloatingPointArea3 {
  public static void main(String[] args) {
    Scanner keyboard = new Scanner(System.in);

    // Read rectangle dimensions
    System.out.print("Enter the rectangle length [decimal number]: ");
    double length = keyboard.nextDouble();
    keyboard.nextLine();
    System.out.print("Enter the rectangle width [decimal number]: ");
    double width = keyboard.nextDouble();

    // Validate user input
    assert length > 0.0 : "The length of the rectangle must be > 0.0";//
          (1)
    assert width > 0.0 : "The width of the rectangle must be > 0.0";  //
          (2)

    double area = length * width; // Calculate area of the rectangle

    // Print the correct answer
    System.out.printf(
      "A rectangle of length %.2f cm. and width %.2f cm. has" +
      " area %.2f sq. cm.%n",
      length, width, area);
  }
}
```

Program Output

```
Enter the rectangle length [decimal number]: -4
Enter the rectangle width [decimal number]: 1.4
A rectangle of length -4.00 cm. and width 1.40 cm. has area -5.60 sq. cm.
```

Assertions as a testing technique

- Assertions provide a useful testing technique that can help us detect errors early
- Assertions can be turned on when running the program for test purposes, and turned off when the program is shipped to the user
- The assertions can be turned on again by means of the ”**-ea**” flag