AL GHURAIR UNIVERSITY
College of Computing

CSC 209 JAVA I

week 3- Control Statements: Part I

Objectives:

- **To use the `if` and `if...else` selection statements to choose among alternative actions.**
- **To use the `while` repetition statement to execute statements in a program repeatedly.**
- **To use counter-controlled repetition.**

Examples:

Example 1

if Single-Selection Statement

Programs use selection statements to choose among alternative courses of action. For example, suppose that the passing grade on an exam is 60. The pseudocode statement

```
If student's grade is greater than or equal to 60
    Print "Passed"
```

determines whether the condition "student's grade is greater than or equal to 60" is true or false. If the condition is true, "Passed" is printed, and the next pseudocode statement in order is "performed." (Remember that pseudocode is not a real programming language.) If the condition is false, the Print statement is ignored, and the next pseudocode statement in order is performed. The indentation of the second line of this selection statement is optional, but recommended, because it emphasizes the inherent structure of structured programs.

The preceding pseudocode If statement may be written in Java as

```
if ( studentGrade >= 60 )
   System.out.println( "Passed" );
```

if...else Double-Selection Statement

The `if` single-selection statement performs an indicated action only when the condition is **`true`**; otherwise, the action is skipped. The `if...else` double-selection statement allows the programmer to specify an action to perform when the condition is true and a different action when the condition is false. For example, the pseudocode statement

```
If student's grade is greater than or equal to 60
    Print "Passed"
Else
    Print "Failed"
```

prints "Passed" if the student's grade is greater than or equal to 60, but prints "Failed" if it is less than 60. In either case, after printing occurs, the next pseudocode statement in sequence is "performed."

The preceding If...Else pseudocode statement can be written in Java as

```
if ( grade >= 60 )
   System.out.println( "Passed" );
else
   System.out.println( "Failed" );
```

Note that the body of the `else` is also indented. Whatever indentation convention you choose should be applied consistently throughout your programs. It is difficult to read programs that do not obey uniform spacing conventions.

while Repetition Statement

A **repetition statement** (also called a looping statement or a **loop**) allows the programmer to specify that a program should repeat an action while some condition remains true. The pseudocode statement

```
While there are more items on my shopping list
   Purchase next item and cross it off my list
```

describes the repetition that occurs during a shopping trip. The condition "there are more items on my shopping list" may be true or false. If it is true, then the action "Purchase next item and cross it off my list" is performed. This action will be performed repeatedly while the condition remains true. The statement(s) contained in the While repetition statement constitute the body of the While repetition statement, which may be a single statement or a block. Eventually, the condition will become false (when the last item on the shopping list has been purchased and crossed off the list). At this point, the repetition terminates, and the first statement after the repetition statement executes.

As an example of Java's `while` repetition statement, consider a program segment designed to find the first power of 3 larger than 100. Suppose that the `int` variable `product` is initialized to `3`. When the following `while` statement finishes executing, `product` contains the result:

```
int product = 3;

while ( product <= 100 )
   product = 3 * product;
```

When this `while` statement begins execution, the value of variable `product` is 3. Each iteration of the `while` statement multiplies `product` by 3, so `product` takes on the values 9, 27, 81 and 243 successively. When variable `product` becomes 243, the `while` statement condition `product <= 100`becomes false. This terminates the repetition, so the final value of `product` is 243. At this point, program execution continues with the next statement after the `while` statement.

Counter-Controlled Repetition

Consider the following problem statement:

A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz.

The class average is equal to the sum of the grades divided by the number of students. The algorithm for solving this problem on a computer must input each grade, keep track of the total of all grades input, perform the averaging calculation and print the result.

Problem Solving

Let's use pseudocode to list the actions to execute and specify the order in which they should execute. We use **counter-controlled repetition** to input the grades one at a time. This technique uses a variable called a **counter** (or **control variable**) to control the number of times a set of statements will execute. Counter-controlled repetition is often called **definite repetition**, because the number of repetitions is known before the loop begins executing. In this example, repetition terminates when the counter exceeds 10.

```
1  Set total to zero
2  Set grade counter to one
3
4  While grade counter is less than or equal to ten
5     Prompt the user to enter the next grade
6     Input the next grade
7     Add the grade into the total
8     Add one to the grade counter
9
10 Set the class average to the total divided by ten
11 Print the class average
```

```java
// GradeBook.java
// GradeBook class that solves class-average problem using
// counter-controlled repetition.
import java.util.Scanner; // program uses class Scanner

public class GradeBook
{
   private String courseName; // name of course this GradeBook represents

    // constructor initializes courseName
    public GradeBook( String name )
    {
       courseName = name; // initializes courseName
    } // end constructor

    // method to set the course name
    public void setCourseName( String name )
    {
       courseName = name; // store the course name
    } // end method setCourseName

    // method to retrieve the course name
    public String getCourseName()
    {
       return courseName;
    } // end method getCourseName

    // display a welcome message to the GradeBook user
    public void displayMessage()
    {
       // getCourseName gets the name of the course
       System.out.printf( "Welcome to the grade book for\n%s!\n\n",
          getCourseName() );
    } // end method displayMessage

    // determine class average based on 10 grades entered by user
    public void determineClassAverage()
    {
       // create Scanner to obtain input from command window
       Scanner input = new Scanner( System.in );

       int total; // sum of grades entered by user
       int gradeCounter; // number of the grade to be entered next
       int grade; // grade value entered by user
       int average; // average of grades

       // initialization phase
       total = 0; // initialize total
       gradeCounter = 1; // initialize loop counter

       // processing phase
       while ( gradeCounter <= 10 ) // loop 10 times
       {
          System.out.print( "Enter grade: " ); // prompt
          grade = input.nextInt(); // input next grade
          total = total + grade; // add grade to total
          gradeCounter = gradeCounter + 1; // increment counter by 1
       } // end while

```

```
60         // termination phase
61         average = total / 10; // integer division yields integer result
62
63         // display total and average of grades
64         System.out.printf( "\nTotal of all 10 grades is %d\n", total );
65         System.out.printf( "Class average is %d\n", average );
66      } // end method determineClassAverage
67
68   } // end class GradeBook
```

Class `GradeBook` contains a constructor (lines 11 14) that assigns a value to the class's instance variable `courseName` (declared in line 8). Lines 17 20, 23 26 and 29 34 declare methods `setCourseName`, `getCourseName` and `displayMessage`, respectively. Lines 37 66 declare method `determineClassAverage`, which implements the class-averaging algorithm described by the pseudocode.

Line 40 declares and initializes `Scanner` variable input, which is used to read values entered by the user. Lines 42 45 declare local variables `total`, `gradeCounter`, `grade` and `average` to be of type `int`. Variable `grade` stores the user input.

Note that the declarations (in lines 42 45) appear in the body of method `determinesClassAverage`. Recall that variables declared in a method body are local variables and can be used only from the line of their declaration in the method to the closing right brace (`}`) of the method declaration. A local variable's declaration must appear before the variable is used in that method. A local variable cannot be accessed outside the method in which it is declared.

In the class `GradeBook`, we simply read and process a set of grades. The averaging calculation is performed in method `determineClassAverage` using local variableswe do not preserve any information about student grades in instance variables of the class. We maintain the grades in memory using an instance variable that refers to a data structure known as an array. This allows a `GradeBook` object to perform various calculations on the same set of grades without requiring the user to enter the grades multiple times.

```
1  // GradeBookTest.java
2  // Create GradeBook object and invoke its determineClassAverage method.
3
4  public class GradeBookTest
5  {
6     public static void main( String args[] )
7     {
8        // create GradeBook object myGradeBook and
9        // pass course name to constructor
10        GradeBook myGradeBook = new GradeBook(
11           "CS101 Introduction to Java Programming" );
12
13        myGradeBook.displayMessage(); // display welcome message
14        myGradeBook.determineClassAverage(); // find average of 10 grades
15     } // end main
16
17  } // end class GradeBookTest
```

```
Welcome to the grade book for
CS101 Introduction to Java Programming!

Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

H/W Exercise:

Drivers are concerned with the mileage their automobiles get. One driver has kept track of five tankfuls times of gasoline by recording the miles driven and gallons used for each tankful. Develop a Java application that will input the miles driven and gallons used (both as integers) for each tankful.

The program should calculate and display the miles per gallon obtained for each tankful and print the combined miles per gallon obtained for all tankfuls up to this point. All averaging calculations should produce floating-point results.

Use class *Scanner* and *Counter -controlled* repetition to obtain the data from the user.

Ex.

miles	gallons
1200	21
1250	22
1150	21
1000	20
1350	22

Average miles = 1190.0

Average gallons = 19.4