CMPE 101
Object Oriented Programming

İstanbul
Bilgi University

Dr. Emel Küpçü

Department of Computer Engineering

İstanbul Bilgi University

Week-3: Conditional Statements

Control Structures

Selection Statements in Java

- Three types of selection statements.
- **`if`** statement:
  - Performs an action, if a condition is *true*; skips it, if *false*.
  - Single-selection statement—selects or ignores a single action (or group of actions).
- **`if…else`** statement:
  - Performs an action if a condition is *true* and performs a different action if the condition is *false*.
  - Double-selection statement—selects between two different actions (or groups of actions).
- **`switch`** statement
  - Performs one of several actions, based on the value of an expression.
  - Multiple-selection statement—selects among *many different actions* (or *groups of actions*).

Control Structures (Cont.)

Iteration Statements in Java

- Four iteration statements (also called iteration statements or looping statements)
  - Perform statements repeatedly while a loop-continuation condition remains *true*.
- **`while`** and **`for`** statements perform the action(s) in their bodies zero or more times
  - if the loop-continuation condition is initially false, the body will *not* execute.
- The **`do…while`** statement performs the action(s) in its body *one or more* times.
- `if`, `else`, `switch`, `while`, `do` and `for` are keywords.

`if` Single-Selection Statement

- Pseudocode

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

- If the condition is false, the Print statement is ignored, and the next pseudocode statement in order is performed.
- Indentation
  - Optional, but recommended
  - Emphasizes the inherent structure of structured programs
- ***If*** in Java:

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

  - Corresponds closely to the pseudocode.

Note: **Pseudocode** is an informal language that helps you develop algorithms without having to worry about the strict details of Java language syntax.

if…else Double-Selection Statement

- `if…else` double-selection statement—specify an action to perform when the condition is true and a different action when the condition is false.
- Pseudocode

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

- ***If…Else*** *statement in Java:*

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

Good Programming Practice 4.1

Indent both body statements of an `if...else` statement. Most IDEs do this for you.

Good Programming Practice 4.2

If there are several levels of indentation, each level should be indented the same additional amount of space.

if…else Double-Selection Statement (Cont.)

Nested if…else Statements

- A program can test multiple cases by placing `if…else` statements inside other **`if…else`** statements to create nested *`if…else` statements*.
- Pseudocode:

```
If student's grade is greater than or equal to 90
    Print "A"
else
    If student's grade is greater than or equal to 80
        Print "B"
    else
        If student's grade is greater than or equal to 70
            Print "C"
        else
            If student's grade is greater than or equal to 60
                Print "D"
            else
                Print "F"
```

if...else Double-Selection Statement (Cont.)

- This pseudocode may be written in Java as

```
if (studentGrade >= 90) {
   System.out.println("A");
}
else {
   if (studentGrade >= 80) {
      System.out.println("B");
   }
   else {
      if (studentGrade >= 70) {
         System.out.println("C");
      }
      else {
         if (studentGrade >= 60) {
            System.out.println("D");
         }
         else {
            System.out.println("F");
         }
      }
   }
}
```

Error-Prevention Tip 4.1

In a nested `if...else` statement, ensure that you test for all possible cases.

if…else Double-Selection Statement (Cont.)

- Most Java programmers prefer to write the preceding nested `if…else` statement as

```
if (studentGrade >= 90) {
   System.out.println("A");
}
else if (studentGrade >= 80) {
   System.out.println("B");
}
else if (studentGrade >= 70) {
   System.out.println("C");
}
else if (studentGrade >= 60) {
   System.out.println("D");
}
else {
   System.out.println("F");
}
```

4.6 if…else Double-Selection Statement (Cont.)

Blocks

- The `if` statement normally expects only one statement in its body.
- To include several statements in the body of an `if` (or the body of an `else` for an `if…else` statement), enclose the statements in braces.
- Statements contained in a pair of braces (such as the body of a method) form a block.
- A block can be placed anywhere in a method that a single statement can be placed.
- Example: A block in the `else` part of an `if…else` statement:

```
if (grade >= 60) {
   System.out.println("Passed");
}
else {
   System.out.println("Failed");
   System.out.println("You must take this course again.");
}
```

4.6 if…else Double-Selection Statement (Cont.)

Conditional operator (?:)

- Conditional operator (`?:`)—shorthand `if…else`.
- Ternary operator (takes *three* operands)
- Operands and `?:` form a conditional expression
- **Operand (1.)** to the left of the `?` is a `boolean` expression—evaluates to a `boolean` value (`true` or `false`)
- **Second operand (2.)** (between the `?` and `:`) is the value if the `boolean` expression is `true`
- **Third operand (3.)** (to the right of the `:`) is the value if the `boolean` expression evaluates to `false`.
- Example:

```
                                  1.                    2.          3.
System.out.println( studentGrade >= 60 ? "Passed" : "Failed" );
```

- Evaluates to the string `"Passed"` if the `boolean` expression `studentGrade >= 60` is true and to the string `"Failed"` if it is false.

Student Class: Nested if…else Statement

```
1  // Fig. 4.4: Student.java
2  // Student class that stores a student name and average.
3  public class Student {
4     private String name;
5     private double average;
6
7     // constructor initializes instance variables
8     public Student(String name, double average) {
9        this.name = name;
10
11       // validate that average is > 0.0 and <= 100.0; otherwise,
12       // keep instance variable average's default value (0.0)
13       if (average > 0.0) {
14          if (average <= 100.0) {
15             this.average = average; // assign to instance variable
16          }
17       }
18    }
```

Class

Instance Variables

Constructor

Fig. 4.4 | Student class that stores a student name and average. (Part 1 of 4.)

Student Class: Nested if…else Statement (cont.)

```
19
20      // sets the Student's name
21      public void setName(String name) {
22         this.name = name;
23      }
24
25      // retrieves the Student's name
26      public String getName() {
27         return name;
28      }
29
```

Setter method for “name”

Getter method for “name”

Fig. 4.4 | Student class that stores a student name and average. (Part 2 of 4.)

Student Class: Nested if…else Statement (cont.)

```
30      // sets the Student's average
31      public void setAverage(double average ) {
32         // validate that average is > 0.0 and <= 100.0; otherwise,
33         // keep instance variable average's current value
34         if (average > 0.0) {
35            if (average <= 100.0) {
36               this.average = average; // assign to instance variable
37            }
38         }
39      }
40
41      // retrieves the Student's average
42      public double getAverage() {
43         return average;
44      }
45
```

Setter method for “average”

Getter method for “average”

Fig. 4.4 | Student class that stores a student name and average. (Part 3 of 4.)

Student Class: Nested if…else Statement (cont.)

```
46      // determines and returns the Student's letter grade
47      public String getLetterGrade() {
48         String letterGrade = ""; // initialized to empty String
49
50         if (average >= 90.0) {
51            letterGrade = "A";
52         }
53         else if (average >= 80.0) {
54            letterGrade = "B";
55         }
56         else if (average >= 70.0) {
57            letterGrade = "C";
58         }
59         else if (average >= 60.0) {
60            letterGrade = "D";
61         }
62         else {
63            letterGrade = "F";
64         }
65
66         return letterGrade;
67      }
68   }
```

Method

Fig. 4.4 | Student class that stores a student name and average. (Part 4 of 4.)

Class StudentTest

```
1  // Fig. 4.5: StudentTest.java
2  // Create and test Student objects.
3  public class StudentTest {
4     public static void main(String[] args) {
5        Student account1 = new Student("Jane Green", 93.5);
6        Student account2 = new Student("John Blue", 72.75);
7
8        System.out.printf("%s's letter grade is: %s%n",
9           account1.getName(), account1.getLetterGrade());
10       System.out.printf("%s's letter grade is: %s%n",
11          account2.getName(), account2.getLetterGrade());
12    }
13 }
```

```
Jane Green's letter grade is: A
John Blue's letter grade is: C
```

Fig. 4.5 | Create and test `Student` objects.

while Iteration Statement

- Iteration statement—repeats an action while a condition remains true.
- Pseudocode

  While there are more items on my shopping list

  Purchase next item and cross it off my list

- The iteration statement's body may be a single statement or a block.
- Eventually, the condition will become false. At this point, the iteration terminates, and the first statement after the iteration statement executes.

while Iteration Statement (Cont.)

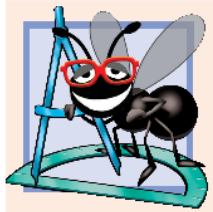
- Example :
  - What will be the final value of the variable **product** after execution?

```
Int product = 3 ;
while (product <= 100){
      product = 3 * product;}
```

- Each iteration multiplies `product` by 3, so `product` takes on the values **9**, **27**, **81** and **243** successively.
- When `product` becomes 243, `product <= 100` becomes false.
- Iteration terminates. The final value of `product` is 243.
- Program execution continues with the next statement after the `while` statement.
- If you don't include an action in a while loop that makes the condition false eventually, it can cause an infinite loop, meaning the loop never ends.

Formulating Algorithms: Example Program

- *A class of **ten students** took a quiz. The grades (integers in the range 0-100) 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.
  - A **total** is a variable used to accumulate the sum of several values.
  - Variables used to store totals are normally initialized to zero before being used in a program.
  - A **counter** is a variable used to count



Software Engineering Observation 4.4

Many programs can be divided logically into three phases: an initialization phase that initializes the variables; a processing phase that inputs data values and adjusts program variables accordingly; and a termination phase that calculates and outputs the final results.

Formulating Algorithms: Counter-Controlled Iteration

Pseudocode Algorithm with Counter-Controlled Iteration

- Use counter-controlled iteration to input the grades one at a time.
- A variable called a counter (or control variable) controls the number of times a set of statements will execute.
- Counter-controlled iteration is often called definite iteration, because the number of iterations is known *before* the loop begins executing.

Formulating Algorithms: Example Program Pseudocode

```
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
```

Fig. 4.7 | Pseudocode algorithm that uses counter-controlled iteration to solve the class-average problem.

Formulating Algorithms: Example Program (cont.)

```
1  // Fig. 4.8: ClassAverage.java
2  // Solving the class-average problem using counter-controlled iteration.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class ClassAverage {
6     public static void main(String[] args) {
7        // create Scanner to obtain input from command window
8        Scanner input = new Scanner(System.in);
9
10       // initialization phase
11       int total = 0;  // initialize sum of grades entered by the user
12       int gradeCounter = 1; // initialize # of grade to be entered next
13
14       // processing phase uses counter-controlled iteration
15       while (gradeCounter <= 10) { // loop 10 times
16          System.out.print("Enter grade: "); // prompt
17          int grade = input.nextInt(); // input next grade
18          total = total + grade; // add grade to total
19          gradeCounter = gradeCounter + 1; // increment counter by 1
20       }
```

Fig. 4.8 | Solving the class-average problem using counter-controlled iteration. (Part 1 of 3.)

Formulating Algorithms: Example Program (cont.)

```
21
22          // termination phase
23          int average = total / 10; // integer division yields integer result
24
25          // display total and average of grades
26          System.out.printf("%nTotal of all 10 grades is %d%n", total);
27          System.out.printf("Class average is %d%n", average);
28       }
29    }
```

Fig. 4.8 | Solving the class-average problem using counter-controlled iteration. (Part 2 of 3.)

Formulating Algorithms: Example Program (cont.)

```
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
```

Fig. 4.8 | Solving the class-average problem using counter-controlled iteration. (Part 3 of 3.)

Error-Prevention Tip 4.3

Initialize each total and counter, either in its declaration or in an assignment statement. Totals are normally initialized to 0. Counters are normally initialized to 0 or 1, depending on how they're used (we'll show examples of when to use 0 and when to use 1).

Formulating Algorithms: Counter-Controlled Iteration (Cont.)

Notes on Integer Division and Truncation

- The program's output indicates that the sum of the grade values in the sample execution is 846, which, when divided by 10, should yield the floating-point number 84.6.
- The result of the calculation `total / 10` (Fig. 4.8) is the integer 84, because `total` and `10` are both integers.
- Dividing two integers results in integer division—any fractional part of the calculation is truncated (i.e., *lost*).

Formulating Algorithms: Sentinel-Controlled Iteration

- ***Develop a class-averaging program that processes grades for an arbitrary number of students each time it is run.***
- Sentinel-controlled iteration is often called indefinite iteration because the number of iterations is not known before the loop begins executing.
- A special value called a sentinel value (also called a signal value, a dummy value or a flag value) can be used to indicate "end of data entry."
- A sentinel value must be chosen that cannot be confused with an acceptable input value.

Formulating Algorithms: Sentinel-Controlled Iteration - Example

- The second refinement of the preceding **pseudocode** statement is then

Initialize variables
Prompt the user to enter the first grade
Input the first grade (possibly the sentinel)
While the user has not yet entered the sentinel
    Add this grade into the running total
    Add one to the grade counter
    Prompt the user to enter the next grade
    Input the next grade (possibly the sentinel)
Calculate and print the class average

4.10 Formulating Algorithms: Sentinel-Controlled Iteration (Cont.)

Proceeding to the Second Refinement

- Second refinement: commit to specific variables.
- The pseudocode statement

  Initialize variables

- can be refined as follows:

  Initialize total to zero

  Initialize counter to zero

Formulating Algorithms: Sentinel-Controlled Iteration (Cont.)

- The pseudocode statement

  Calculate and print the class average

- can be refined as follows:

  If the counter is not equal to zero

  Set the average to the total divided by the counter

  Print the average

  else

  Print "No grades were entered"

- Test for the possibility of *division by zero*—a *logic error* that, if undetected, would cause the program to fail or produce invalid output.

Formulating Algorithms: Example's Pseudocode

```
1   Initialize total to zero
2   Initialize counter to zero
3
4   Prompt the user to enter the first grade
5   Input the first grade (possibly the sentinel)
6
7   While the user has not yet entered the sentinel
8       Add this grade into the running total
9       Add one to the grade counter
10      Prompt the user to enter the next grade
11      Input the next grade (possibly the sentinel)
12
13  If the counter is not equal to zero
14      Set the average to the total divided by the counter
15      Print the average
16  Else
17      Print "No grades were entered"
```

Fig. 4.9 | Class-average pseudocode algorithm with sentinel-controlled iteration.

Formulating Algorithms: Example's Code

```
1  // Fig. 4.10: ClassAverage.java
2  // Solving the class-average problem using sentinel-controlled iteration.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class ClassAverage {
6     public static void main(String[] args) {
7        // create Scanner to obtain input from command window
8        Scanner input = new Scanner(System.in);
9
10       // initialization phase
11       int total = 0; // initialize sum of grades
12       int gradeCounter = 0; // initialize # of grades entered so far
13
```

Fig. 4.10 | Solving the class-average problem using sentinel-controlled iteration. (Part 1 of 4.)

Formulating Algorithms: Example's Code (cont.)

```
14          // processing phase
15          // prompt for input and read grade from user
16          System.out.print("Enter grade or -1 to quit: ");
17          int grade = input.nextInt();
18
19          // loop until sentinel value read from user
20          while (grade != -1) {
21             total = total + grade; // add grade to total
22             gradeCounter = gradeCounter + 1; // increment counter
23
24             // prompt for input and read next grade from user
25             System.out.print("Enter grade or -1 to quit: ");
26             grade = input.nextInt();
27          }
28
```

Fig. 4.10 | Solving the class-average problem using sentinel-controlled iteration. (Part 2 of 4.)

Formulating Algorithms: Example's Code (cont.)

```
29          // termination phase
30          // if user entered at least one grade...
31          if (gradeCounter != 0) {
32             // use number with decimal point to calculate average of grades
33             double average = (double) total / gradeCounter;
34
35             // display total and average (with two digits of precision)
36             System.out.printf("%nTotal of the %d grades entered is %d%n",
37                gradeCounter, total);
38             System.out.printf("Class average is %.2f%n", average);
39          }
40          else { // no grades were entered, so output appropriate message
41             System.out.println("No grades were entered");
42          }
43       }
44    }
```

Fig. 4.10 | Solving the class-average problem using sentinel-controlled iteration. (Part 3 of 4.)

Formulating Algorithms: Example's Code (cont.)

```
Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67
```

Fig. 4.10 | Solving the class-average problem using sentinel-controlled iteration. (Part 4 of 4.)

Explicitly and Implicitly Converting Between Primitive Types

- Integer division yields an integer result.
  - ```
    int a = 5;
    int b = 2;
    int result = a / b;  // result will be 2, not 2.5, because integer division truncates the decimal part.
    System.out.println(result);  // Output: 2
    ```

- To perform a floating-point calculation with integers, *temporarily* treat these values as floating-point numbers for use in the calculation.
- The unary cast operator `(double)` creates a temporary floating-point copy of its operand.
  - ```
    int c = 5;
    double result = (double) c;  // The cast creates a temporary double copy of the integer 'c'
    System.out.println(result);  // Output: 5.0
    ```

- **Cast** operator performs explicit conversion (or type cast).
  - The value stored in the operand is unchanged.
  - ```
    double d = 5.6;
    int e = (int) d;  // 'd' is cast to an int, but its value remains 5.6
    System.out.println(d);  // Output: 5.6 (original value of 'd' is unchanged)
    System.out.println(e);  // Output: 5 (after casting to int)
    ```
    ```

Explicitly and Implicitly Converting Between Primitive Types

- Java evaluates only arithmetic expressions in which the operands' types are *identical*.
- In an expression containing values of the types `int` and `double`, the `int` values are promoted to `double` values for use in the expression.

  - ```
    int f = 5;
    double g = 2.5;
    // Java will automatically promote 'f' to double to match the type of 'g'
    double result = f + g;  // result will be 7.5
    System.out.println(result);  // Output: 7.5
    ```
    ```

Compound Assignment Operators

- Compound assignment operators abbreviate assignment expressions.
- Example:

```
c = c + 3;
```

  can be written with the addition compound assignment operator, +=, as

```
c += 3;
```

- The += operator adds the value of the expression on its right to the value of the variable on its left and stores the result in the variable on the left of the operator.
- Statements like

  variable = variable operator expression;

  where operator is one of the binary operators +, -, *, / or % can be written in the form

  variable operator= expression;

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

Fig. 4.13 | Arithmetic compound assignment operators.

Increment and Decrement Operators

- Unary increment operator, ++, adds one to its operand
- Unary decrement operator, --, subtracts one from its operand
- An increment or decrement operator that is prefixed to (placed before) a variable is referred to as the prefix increment or prefix decrement operator, respectively.
- An increment or decrement operator that is postfixed to (placed after) a variable is referred to as the postfix increment or postfix decrement operator, respectively.

Operator	Sample expression	Explanation
++ (prefix increment)	`++a`	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
++ (postfix increment)	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
-- (prefix decrement)	`--b`	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
-- (postfix decrement)	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

Fig. 4.14 | Increment and decrement operators.

Increment and Decrement Operators (Cont.)

- Using the prefix increment (or decrement) operator to add (or subtract) 1 from a variable is known as preincrementing (or predecrementing) the variable.
- Preincrementing (or predecrementing) a variable causes the variable to be incremented (decremented) by 1; then the new value is used in the expression in which it appears.
- Using the postfix increment (or decrement) operator to add (or subtract) 1 from a variable is known as postincrementing (or postdecrementing) the variable.
- This causes the current value of the variable to be used in the expression in which it appears; then the variable's value is incremented (decremented) by 1.

```
1  // Fig. 4.15: Increment.java
2  // Prefix increment and postfix increment operators.
3
4  public class Increment {
5     public static void main(String[] args) {
6        // demonstrate postfix increment operator
7        int c = 5;
8        System.out.printf("c before postincrement: %d%n", c); /
9        System.out.printf("    postincrementing c: %d%n", c++);
10       System.out.printf(" c after postincrement: %d%n", c); /
11
12       System.out.println(); // skip a line
13
14       // demonstrate prefix increment operator
15       c = 5;
16       System.out.printf(" c before preincrement: %d%n", c); //
17       System.out.printf("     preincrementing c: %d%n", ++c); /
18       System.out.printf("  c after preincrement: %d%n", c); //
19    }
20 }
```

Fig. 4.15 | Prefix increment and postfix increment operators. (Part 1 of 2.)

```
c before postincrement: 5
    postincrementing c: 5
 c after postincrement: 6

 c before preincrement: 5
     preincrementing c: 6
  c after preincrement: 6
```

Fig. 4.15 | Prefix increment and postfix increment operators. (Part 2 of 2.)

Common Programming Error 4.8

Attempting to use the increment or decrement operator on an expression other than one to which a value can be assigned is a syntax error. For example, writing `++(x + 1)` is a syntax error, because `(x + 1)` is not a variable.

Questions?