# Control Statement I

CS102A Lecture 3

James YU
yujq3@sustech.edu.cn

Department of Computer Science and Engineering
Southern University of Science and Technology

Sept. 21, 2020

# Objectives

- To learn and use basic problem-solving techniques.
- To develop algorithms using pseudo codes.
- To use selection statements.
  - `if`, `if...else`.
- To use repetition statement.
  - `while`, `do-while`, `for`.

# Programming like a professional

- Before writing a program, you should have a thorough understanding of the problem and a carefully planned approach to solving it.
- Understand the types of building blocks that are available and employ proven program-construction techniques.

# Algorithms

- Any computing problem can be solved by executing a series of actions in a specific order.
- An algorithm is a procedure for solving a problem in terms of
  - the actions to execute, and
  - the order in which these actions execute.
- The “rise-and-shine algorithm” for an executive: (1) get out of bed; (2) take off pajamas; (3) take a shower; (4) get dressed; (5) eat breakfast; (6) drive to work.
- Specifying the order in which statements (actions) execute in a program is called program control.

# Pseudocode

- Pseudocode is an informal language for developing algorithms.
  - Like everyday English.
  - Helps you “think out” a program.
- Pseudocode normally describes only statements representing the actions, e.g., input, output or calculations.
- Carefully prepared pseudocode can be easily converted to a corresponding Java program.

# Pseudocode

- Recall the time converter exercise in your lab2

```
Enter the number of seconds: 7402
The equivalent time is 2 hours 3 minutes and 22 seconds.
```

1. Get the number of seconds from user;
2. Compute the number of hours;
3. Compute the number of minutes;
4. Compute the remain seconds;
5. Print out result.

# Compound assignment operators

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Compound assignment operators simplify assignment expressions.
- `variable = variable operator expression;`
  - where operator is one of +, -, *, / or % can be written in the form `variable operator= expression;`
  - `c = c + 3;` can be written as `c += 3;`

# Compound assignment operators

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

| 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` |

# 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 post-fixed to (placed after) a variable is referred to as the *postfix increment* or *postfix decrement operator*, respectively.

```
1 int a = 6; int b = ++a; int c = a--;
```

# Pre-incrementing/pre-decrementing

- Using the prefix increment (or decrement) operator to add (or subtract) 1 from a variable is known as pre-incrementing (or pre-decrementing) the variable.
- Pre-incrementing (or pre-decrementing) a variable causes the variable to be **increment**ed (decremented) by 1; **then** the new value is **use**d in the expression in which it appears.

```
1 int a = 6; int b = ++a; // b gets the value 7
```

# Post-incrementing/post-decrementing

- Using the postfix increment (or decrement) operator to add (or subtract) 1 from a variable is known as post-incrementing (or post-decrementing) the variable.
- This causes the current value of the variable to be **use**d in the expression in which it appears; **then** the variable's value is **incremente**d (decremented) by 1.

```
1 int a = 6; int b = a++; // b gets the value 6
```

# Note the difference

```
1 int a = 6; int b = ++a; // b gets the value 7
```

```
1 int a = 6; int b = a++; // b gets the value 6
```

- In both cases, a becomes 7 after execution, but b gets different values. Be careful when programming.

# The operators introduced so far

| Operators | | | | | | Type |
|---|---|---|---|---|---|---|
| `++` | `--` | `(type)` | | | | Unary postfix |
| `*` | `/` | `%` | | | | Multiplicative |
| `+` | `-` | | | | | Additive |
| `<` | `<=` | `>` | `>=` | | | Relational |
| `==` | `!=` | | | | | Equality |
| `?:` | | | | | | Conditional |
| `=` | `+=` | `-=` | `*=` | `/=` | `%=` | Assignment |

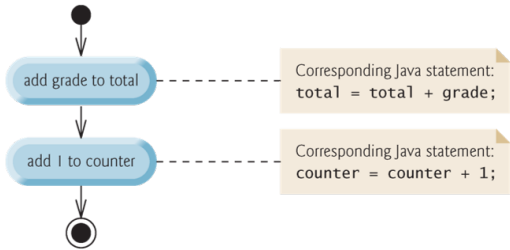
- Precedence: from top to down.

# Control structures

- *Sequential execution*: normally, statements in a program are executed one after the other in the order in which they are written.
- *Transfer of control*: various Java statements enable you to specify the next statement to execute, which is not necessarily the next one in sequence.
- All programs can be written in terms of only three control structures — the sequence structure, the selection structure and the repetition structure.

# Sequence structure

- Unless directed otherwise, computers execute Java statements one after the other in the order in which they're written.
- The activity diagram (a flowchart showing activities performed by a system) below illustrates a typical sequence structure in which two calculations are performed in order.



# Selection structure

- Three types of selection statements:
  - `if` statement
  - `if...else` statement
  - `switch` statement

# `if` single-selection statement

- Pseudocode:

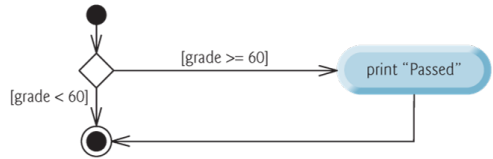
```
If student's grade is greater than or equal to 60
    Print "Passed"
```

- Java code:

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

# `if` single-selection statement

```
1 if (studentGrade >= 60) {
2   System.out.println("Passed");
3 }
```




- Diamond, or decision symbol, indicates that a decision is to be made.
- Workflow continues along a path determined by the symbol’s guard conditions, which can be `true` or `false`.
- Each transition arrow from a decision symbol has a *guard condition*.
- If a guard condition is `true`, the workflow enters the action state to which the transition arrow points.

# `if...else` double-selection statement

- Pseudocode:

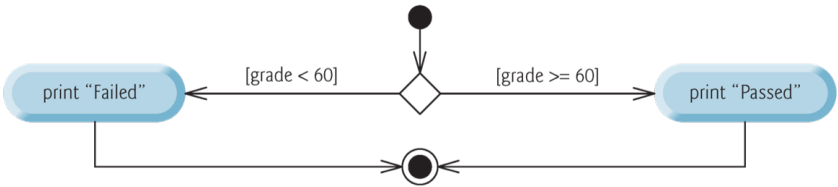
```
If student's grade is greater than or equal to 60
    Print "Passed"
Else
    Print "Failed"
```

- Java code:

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

# `if...else` double-selection statement

- The symbols in the UML activity diagram represent actions and decisions.



# Conditional operator ? :

```
1 String result = studentGrade >= 60 ? "Passed" : "Failed";
```

- The operands and `?:` form a conditional expression.
  - Shorthand of `if...else`
- `studentGrade >= 60`: a boolean expression that evaluates to `true` or `false`.
- `? "Passed"`: the conditional expression takes this value if the Boolean expression evaluates to `true`.
- `: "Failed"`: the conditional expression takes this value if the Boolean expression evaluates to `false`.

# A more complex example

- 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"
```

# A more complex example

- Translate the pseudocode to real Java code:

```java
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");
  }
}
```

# A more elegant version

- 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");
}
```

# `if-else` matching rule

- The Java compiler always associates an else with the immediately preceding if unless told to do otherwise by the placement of braces:
  - `and` .
- The following code does not execute like what it appears:

```
if (x > 5)
  if (y > 5)
    System.out.println("x and y are > 5");
else
  System.out.println("x is <= 5");
```

  - What is the program result when `x` is 7 and `y` is 3?

# `if-else` matching rule

- Recall that the extra spaces are irrelevant in Java. The compiler interprets the statement as

```
if (x > 5)
  if (y > 5)
    System.out.println("x and y are > 5");
  else
    System.out.println("x is <= 5");
```

```
if (x > 5)
  if (y > 5)
    System.out.println("x and y are > 5");
else
  System.out.println("x is <= 5");
```

# `if-else` matching rule

- What if you really want this effect?

```
if (x > 5) {
  if (y > 5)
    System.out.println("x and y are > 5");
} else
  System.out.println("x is <= 5");
```

- Curly braces indicate that the 2nd `if` is the body of the 1st `if`.
  - Tip: always use  to make the bodies of `if` and `else` clear.

# Repetition structure

- Three repetition statements (a.k.a., looping statements). Perform statements repeatedly while a loop-continuation condition remains `true`.
  - `while` statement
  - `for` statement
  - `do...while` statement

# Loops

- There are many situations when you need to execute a block of code several number of times.
- Loops are used to execute a set of statements repeatedly until a particular condition is satisfied. In Java, we have three types of basic loops: `for`, `while` and `do-while`.

# The `for` loop in Java

```
for (initialization; condition; increment/decrement expression) {
  statements;
}
```

- In `for` loop, initialization happens first and only one time.
- Condition in `for` loop is evaluated on each iteration, if the condition is `true` then the statements inside `for` loop body gets executed. Once the condition returns `false`, the statements in `for` loop does not execute and the control gets transferred to the next statement in the program.
- After every execution of `for` loop's body, the increment/decrement part of `for` loop executes and updates the loop counter.
- After the third step, the control jumps to the second step and the condition is re-evaluated.

# `for` loop example

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
public class ForLoopExample {
  public static void main(String args[]) {
    for (int i = 10; i > 1; i--) {
      System.out.println("The value of i is : " + i);
    }
  }
}
```

```
The value of i is : 10
The value of i is : 9
The value of i is : 8
The value of i is : 7
...
The value of i is : 2
```

# `while` repetition statement

- Repeat 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 repetition statement's body may be a single statement or a block.
  Eventually, the condition should become `false`, and the repetition terminates, and the first statement after the repetition statement executes.

# Example

- Example of Java's while repetition statement: find the first power of 3 larger than 100.

```
1 int product = 3;
2 while (product <= 100) {
3     product = 3 * product
4 }
```

# Common mistakes

```
1 void rectifier(int x) {
2   if (x < 0);
3     x = 0;
4   return x;
5 }
```

# Syntax and logic errors revisited

- Syntax errors (e.g., when one brace in a block is left out of the program) are caught by the compiler.
- A logic error (e.g., when both braces in a block are left out of the program) has its effect at execution time.
  - A fatal logic error causes a program to fail and terminate prematurely.
  - A nonfatal logic error allows a program to continue executing but causes it to produce incorrect results.

# Empty statement

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Just as a block can be placed anywhere a single statement can be placed, it's also possible to have an empty statement.
- The empty statement is represented by placing a semicolon (`;`) where a statement would normally be.

```
// This program is valid, although meaningless.
if (x == 1) {
  ;
} else if (x == 2) {
  ;
} else {
  ;
}
```

# Formulating algorithms: Counter-controlled repetition

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Problem: 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.
- Analysis: 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.
- Solution: Use counter-controlled repetition 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.

# Formulating algorithms: Counter-controlled repetition

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
Set total score to zero
Set grade counter to one
While grade counter is less than or equal to ten
    Prompt the user to enter the next grade
    Input the next grade
    Add the grade into total score
    Add one to the grade counter
Set the class average to the total score divided by ten
Print the class average
```

# Formulating algorithms: Counter-controlled repetition

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
1  // Counter-Controlled Repetition
2  import java.util.Scanner; //program uses class Scanner
3  public class ClassAverage
4  {
5    public static void main(String[] args) {
6      // create Scanner to obtain input
7      Scanner input = new Scanner(System.in);
8      int total = 0; // sum of grades
9      int gradeCounter = 1; // number of the grade to be entered
10     int grade; // grade value entered by user
11     int average; // average of grades
12     // processing phase
13     while (gradeCounter <= 10) // loop 10 times
14     {
15       System.out.print("Enter grade: ");
16       grade = input.nextInt();
```

# Formulating algorithms: Counter-controlled repetition

```
17          total += grade;
18          gradeCounter++;
19       }
20       average = total / 10;
21       // display total and average of grades
22       System.out.printf("\nTotal of all 10 grades is %d\n", total);
23       System.out.printf("Class average is %d\n", average);
24    }
25 }
```

# Formulating algorithms: Counter-controlled repetition

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
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
```

# The scope of variables

- Variables declared in a method body are local variables and can be used only from the line of their declaration to the closing right brace of the method declaration.
- A local variable cannot be accessed outside the method in which it's declared.
- A local variable's declaration must appear before the variable is used in that method

# Formulating algorithms: Sentinel–controlled repetition

- A new problem: Develop a class-averaging program that processes grades for an arbitrary number of students each time it is run.
- Analysis: The number of grades was known earlier, but here how can the program determine when to stop the input of grades?
  - We can use a special value called a sentinel value (a.k.a, signal value, dummy value or flag value) can be used to indicate “end of data entry”.
  - Sentinel-controlled repetition is often called indefinite repetition because the number of repetitions is not known before the loop begins executing.
  - A sentinel value must be chosen that cannot be confused with an acceptable input value.


# Formulating algorithms: Sentinel–controlled repetition

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
Set total score to zero
Set grade counter to zero
Prompt the user to enter the next 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)
If the counter is not equal to zero
    Set the average to the total divided by the counter
else
    Print ``No grades were entered''
```

# Formulating algorithms: Sentinel–controlled repetition

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```java
// Sentinel-Controlled Repetition: Class-average problem
import java.util.Scanner; //program uses class Scanner
public class ClassAverage
{
  public static void main(String[] args) {
    // create Scanner to obtain input
    Scanner input = new Scanner(System.in);
    int total = 0; // sum of grades
    int gradeCounter = 0; // number of the grade to be entered
    System.out.print("Enter grade or -1 to quit: ");
    int grade = input.nextInt(); // grade value entered by user
    // loop until sentinel value read from user
    while (grade != -1) {
      total += grade;
      gradeCounter++;
```

# Formulating algorithms: Sentinel–controlled repetition

```
16        System.out.print("Enter grade or -1 to quit: ");
17        grade = input.nextInt();
18      }
19      if (gradeCounter != 0) {
20        double average = (double) total / gradeCounter;
21        System.out.printf("\nTotal of the %d grades is %d\n",
22                          gradeCounter, total);
23        System.out.printf("Class average is %.2f\n", average);
24      } else {
25        System.out.println("No grades were engered");
26      }
27    }
28  }
```

# Formulating algorithms: Sentinel–controlled repetition



```
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 is 257
Class average is 85.67
```

# Case study: Nested control statements

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- A college offers a course that prepares students for the state licensing exam for real estate brokers. Last year, ten of the students who completed this course took the exam. The college wants to know how well its students did on the exam.
- You've been asked to write a program to summarize the results. You've been given a list of these 10 students. Next to each name is written a 1 if the student passed the exam or a 2 if the student failed.

# Case study: Nested control statements

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

- Your program should analyze the exam results as follows:
  - Input each test result (i.e., a 1 or a 2). Display the message "`Enter result`" on the screen each time the program requests another test result.
  - Count the number of test results of each type (`pass` or `fail`).
  - Display a summary of the test results, indicating the number of students who passed and the number who failed.
  - If more than eight students passed the exam, print the message "`Bonus to instructor!`".

# Case study: Nested control statements

南方科技大学 SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

```
1  // Analysis of examination results
2  import java.util.Scanner; //program uses class Scanner
3  public class Analysis
4  {
5    public static void main(String[] args) {
6      // create Scanner to obtain input
7      Scanner input = new Scanner(System.in);
8      int passes = 0;
9      int fails = 0;
10     int studentCounter = 1;
11     int result; // one exam result obtained from user
12     // loop until sentinel value read from user
13     while (studentCounter <= 10) {
14       System.out.print("Enter result (1 = pass, 2 = fail): ");
15       result = input.nextInt();
```

# Case study: Nested control statements

```
16        if (result == 1)
17          passes++;
18        else
19          fails++;
20        studentCounter++;
21      }
22        System.err.printf("Passed: %d\nFailed: %d\n", passes, fails);
23      if (passes > 8)
24        System.out.println("Bonus to instructor!");
25    }
26  }
```