Control Statement I

CS102A Lecture 3

James YU

Sept. 21, 2020

南方科技大学
SOUTHERN UNIVERSITY OF SCIENCE AND TECHNOLOGY

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

- 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

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 **increment**ed (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

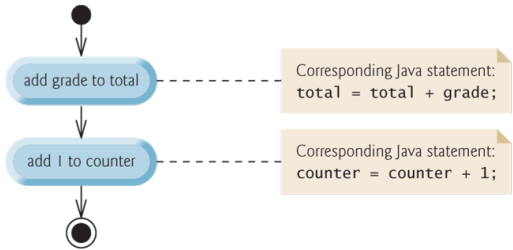
- Precedence: from top to down.

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:

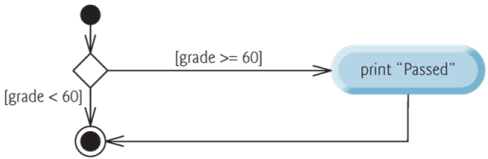
```
If student's grade is greater than or equal to 60
    Print "Passed"
```

- Java code:

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

`if` single-selection statement

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



- 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:

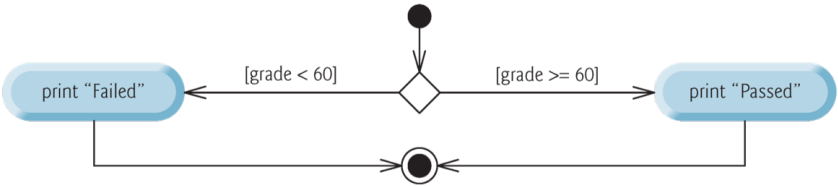
```
If student's grade is greater than or equal to 60
    Print "Passed"
Else
    Print "Failed"
```

- 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:

```
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:

```
1 if (x > 5)
2   if (y > 5)
3     System.out.println("x and y are > 5");
4 else
5   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

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

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