Introduction to Computer Programming (CS102A)

Lecture 4: Control Statements (Part 2)

Yuxin Ma

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

Objectives

- To use `while` repetition statement
- To use `for` and while statements
- To use `switch` statement
- To use `continue` and `break` statements
- To use logical operators.

Repetition Structure

- There are many situations when you need to execute a block of code several number of times.
- Three repetition statements (a.k.a., looping statements)
  - `while` statement
  - `for` statement
  - `do...while` statement
- Perform statements repeatedly while a loop-continuation condition remains true.

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.

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

- **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?

Formulating Algorithms: Sentinel-Controlled Repetition

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

Formulating Algorithms: Sentinel-Controlled Repetition

- total stores the sum of grades
- counter stores the number grades

- Show a prompt
- Take the first input

- If no sentinel value seen, repeat the process

- Compute the average value

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

Formulating Algorithms: Sentinel-Controlled Repetition

- total stores the sum of grades
- counter stores the number grades

- Show a prompt
- Take the first input

- If no sentinel value seen, repeat the process

- Compute the average value

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

Why do we need this?

```java
// Sentinel-Controlled Repetition: Class-average problem

import java.util.Scanner; //program uses class Scanner

public class SentinelWhile {
    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++;
            System.out.print("Enter grade or -1 to quit: ");
            grade = input.nextInt();
        }
        if (gradeCounter != 0) {
            double average = (double) total / gradeCounter;
            System.out.printf("\nTotal of the %d grades is %d\n",
                    gradeCounter, total);
            System.out.printf("Class average is %.2f\n", average);
        } else {
            System.out.println("No grades were engered");
        }
    }
}

```

```
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

- 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

- 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!”

```java
// Analysis of examination results

import java.util.Scanner; //program uses class Scanner

public class NestedControl {
    public static void main(String[] args) {
        // create Scanner to obtain input
        Scanner input = new Scanner(System.in);
        int passes = 0;
        int fails = 0;
        int studentCounter = 1;
        int result; // one exam result obtained from user

        // loop until sentinel value read from user
        while (studentCounter <= 10) {
            System.out.print("Enter result (1 = pass, 2 = fail): ");
            result = input.nextInt();
            if (result == 1)
                passes++;
            else
                fails++;
            studentCounter++;
        }

        System.err.printf("Passed: %d\nFailed: %d\n", passes, fails);
        if (passes > 8)
            System.out.println("Bonus to instructor!");
    }
}

```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

```java
// Analysis of examination results

import java.util.Scanner; //program uses class Scanner

public class NestedControl {
    public static void main(String[] args) {
        // create Scanner to obtain input
        Scanner input = new Scanner(System.in);
        int passes = 0;
        int fails = 0;
        int studentCounter = 1;
        int result; // one exam result obtained from user

        // loop until sentinel value read from user
        while (studentCounter <= 10) {
            System.out.print("Enter result (1 = pass, 2 = fail): ");
            result = input.nextInt();
            if (result == 1)
                passes++;
            else
                fails++;
            studentCounter++;
        }

        System.err.printf("Passed: %d\nFailed: %d\n", passes, fails);
        if (passes > 8)
            System.out.println("Bonus to instructor!");
    }
}
```

An `if` block inside a `while` loop (hence *nested control*)

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 9
Failed: 1
Bonus to instructor!
```

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed: 6
Failed: 4
```

Counter-Controlled Repetition with `while`

```java
public class WhileCounter {
  public static void main(String[] args) {
    int counter = 1; // Control variable (loop counter)
    while ( counter <= 10 ) { // Loop continuation condition
      System.out.printf("%d", counter);
      ++counter; // Counter increment (or decrement) in each iteration
    }
    System.out.println();
  }
}
```

Counter-Controlled Repetition with `while`

```java
public class WhileCounter {
  public static void main(String[] args) {
    int counter = 1; // Control variable (loop counter)
    while ( counter <= 10 ) { // Loop continuation condition
      System.out.printf("%d", counter);
      ++counter; // Counter increment (or decrement) in each iteration
    }
    System.out.println();
  }
}
```

Any way of improving this?

The for Repetition Statement

- Specifies the counter-controlled-repetition details in a single line of code

```java
public class ForCounter {
  public static void main(String[] args) {
    for (int counter = 1; counter <= 10; counter++) {
      System.out.printf("%d", counter);
    }
    System.out.println();
  }
}
```

The for Repetition Statement

- Specifies the counter-controlled-repetition details in a single line of code

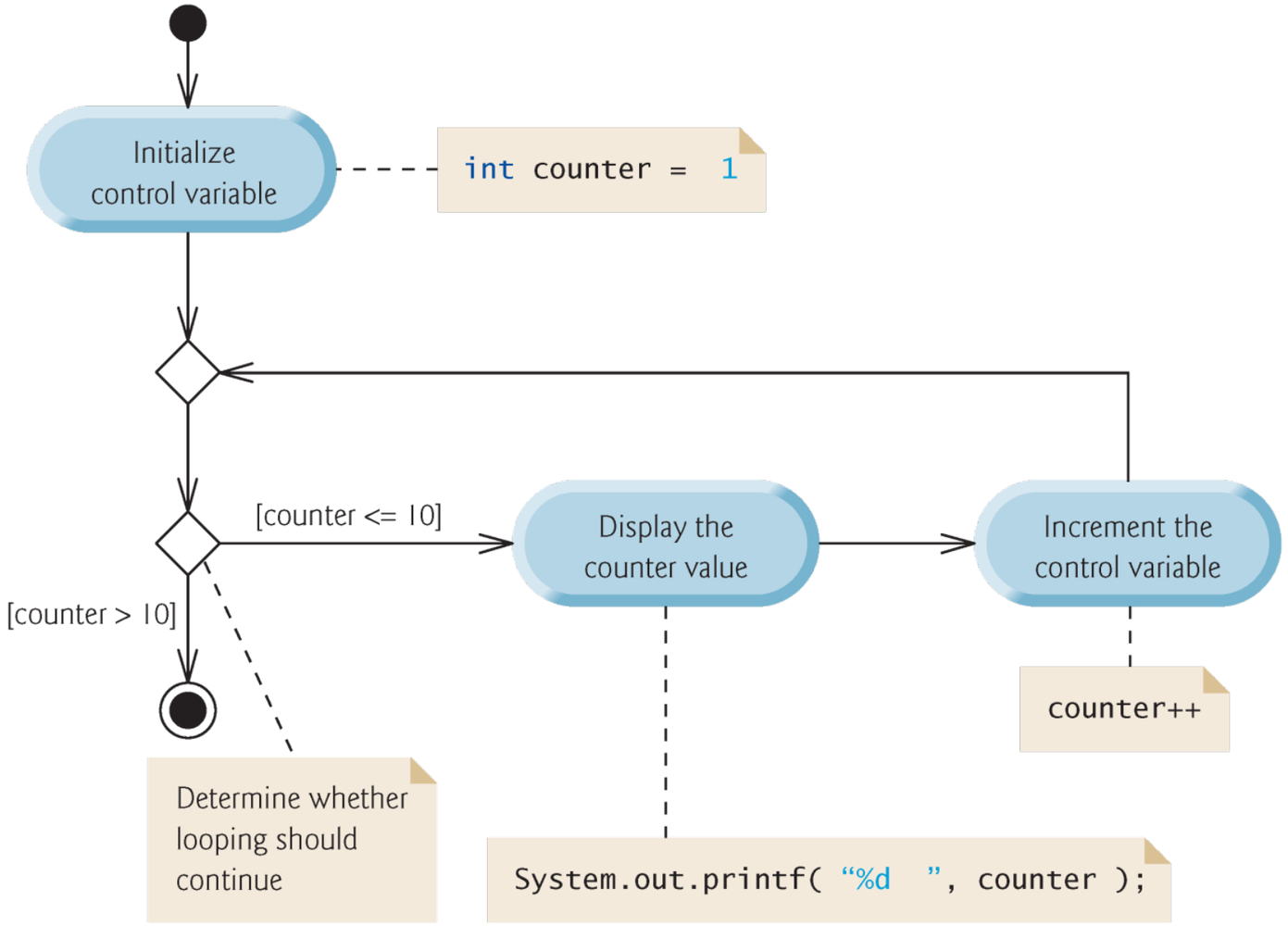
```
1 public class ForCounter {
2   public static void main(String[] args) {
3     for (int counter = 1; counter <= 10; counter++) {
4       System.out.printf("%d", counter);
5     }
6     System.out.println();
7   }
8 }
```

The magic of `for` loop

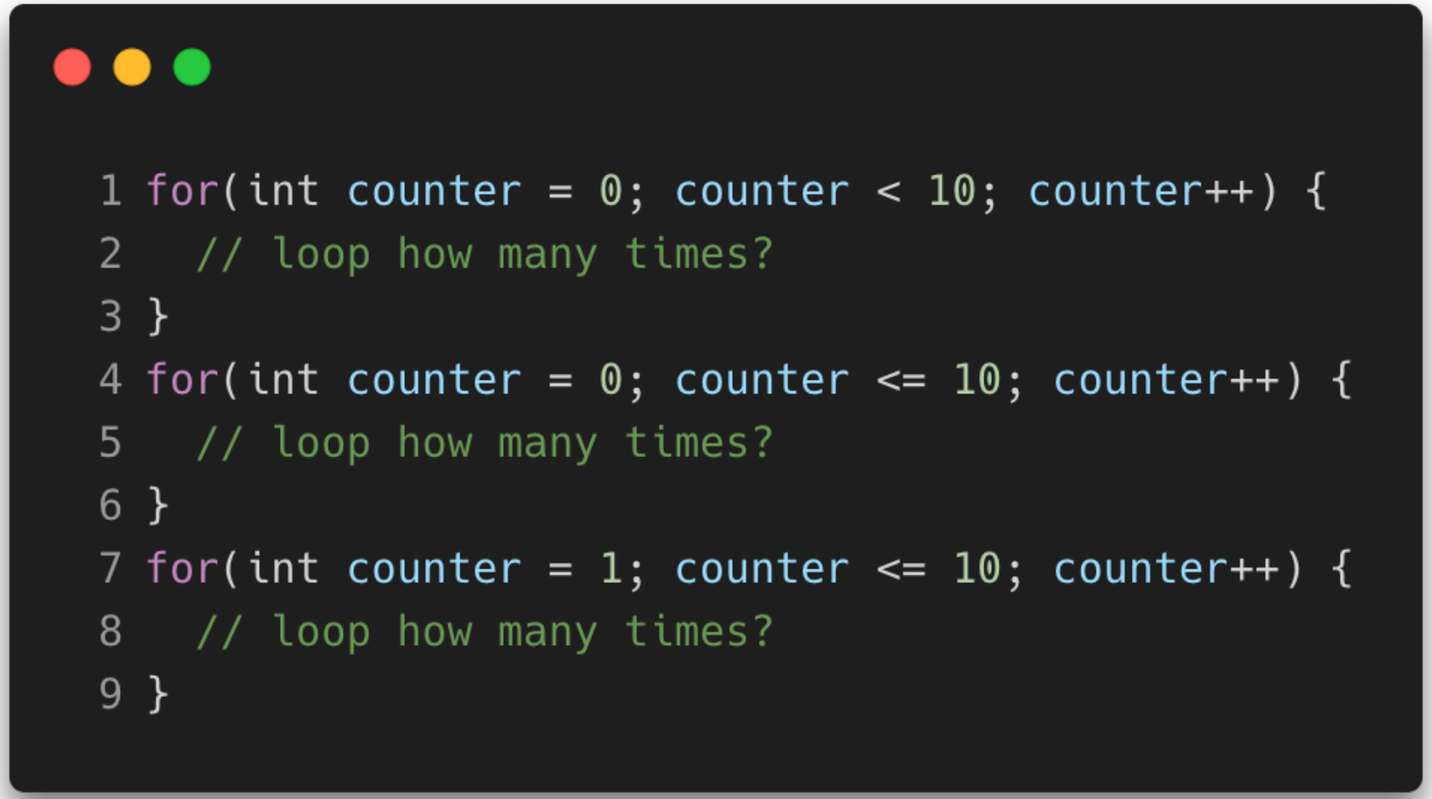
The for Repetition Statement

```
3     for (int counter = 1; counter <= 10; counter++) {
4        System.out.printf("%d", counter);
5     }
```

Execution Flow



Common Logic Error: Off-by-One



```
1 for(int counter = 0; counter < 10; counter++) {
2   // loop how many times?
3 }
4 for(int counter = 0; counter <= 10; counter++) {
5   // loop how many times?
6 }
7 for(int counter = 1; counter <= 10; counter++) {
8   // loop how many times?
9 }
```

The for and while loops

- In most cases, a `for` statement can be easily represented with an equivalent `while` statement
- Typically,
  - `for` statements are used for counter-controlled repetition
  - `while` statements for sentinel-controlled repetition

Control Variable Scope in for

- If the initialization expression in the for header declares the control variable, the control variable can be used only in that for statement.

```
int i;
```

Declaration: stating the type and name of a variable

```
i = 3;
```

Assignment (definition): storing a value in a variable.
Initialization is the first assignment.

```
for(int i = 1; i <= 10; i++){
    // i can only be used
    // in the loop body
}
```

```
int i;
for(i = 1; i <= 10; i++){
    // i can be used here
}
// i can also be used
// after the loop until
// the end of the enclosing
block
```

More on for Repetition Statement

- **Pay attention to these issues when writing loops:**
  - If the loop-continuation condition is omitted, the condition is always true, thus creating an infinite loop.
  - You might omit the initialization expression if the program initializes the control variable before the loop.
    - E.g., `sum` should usually be initialized as 0
  - You might omit the increment if the program calculates it with statements in the loop's body or no increment is needed.
  - The increment expression in a `for` acts as if it were a standalone statement at the end of the `for`'s body, so

```
1 counter = counter + 1; counter += 1; ++counter; counter++;
```

are equivalent increment expressions in a `for` statement.

More on for Repetition Statement

- The *initialization* and *increment/decrement* expressions can contain multiple expressions separated by commas.

```
for ( int number = 2;  number <= 20;
    total += number, number += 2 )
    ; // empty statement

Equivalent to:

for ( int number = 2; number <= 20; number += 2 ) {
    total += number;
}
```

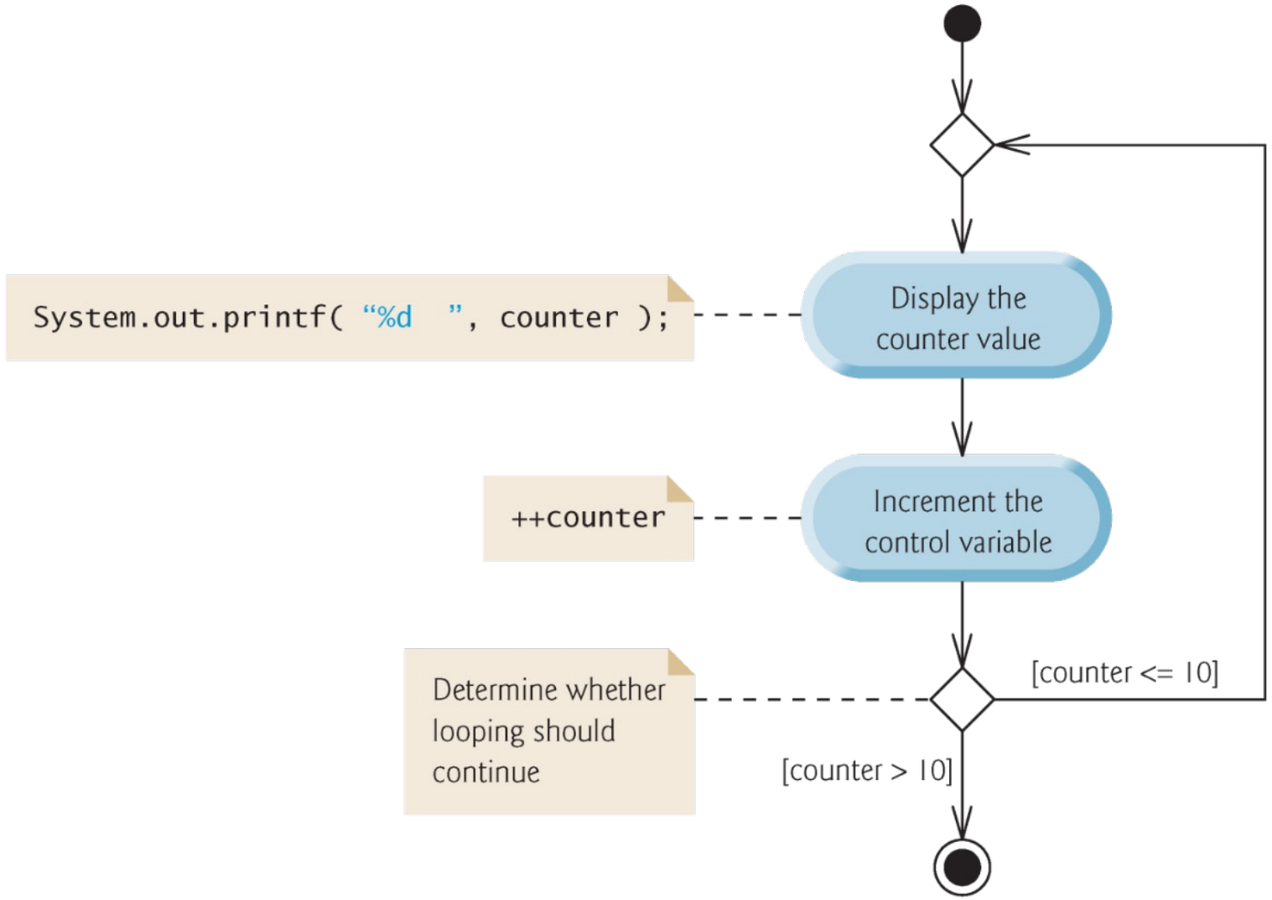
The `do...while` Repetition Statement

- `do...while` is similar to `while`
  - In `while`, the program tests the loop-continuation condition at the beginning of the loop
    - before executing the loop body; if the condition is false, the body never executes.
  - `do...while` tests the loop-continuation condition after executing the loop body.
    - The body always executes at least once.

Execution Flow

```
1 int counter = 1;
2
3 do {
4   System.out.println(counter);
5   ++counter;
6 } while( counter <= 10 );
```

Don’t forget the semicolon



The `switch` Multiple-Selection Statement

- The `switch` statement performs different actions based on the values of a constant integral expression of type byte, short, int or char etc.
- It consists of a block that contains a sequence of `case` labels and an optional `default` case.

```java
switch (studentGrade) {
    case 'A':
        System.out.println("90 - 100");
        break;
    case 'B':
        System.out.println("80 - 89");
        break;
    case 'C':
        System.out.println("70 - 79");
        break;
    case 'D':
        System.out.println("60 - 69");
        break;
    default:
        System.out.println("score < 60");
}
```

The `switch` Multiple-Selection Statement

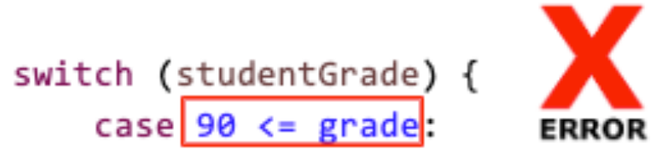
- The program compares the controlling expression's value with each case label.
- If a match occurs, the program executes that `case`'s statements.
- If no match occurs, the `default` case executes.
- If no match occurs and there is no `default` case, program simply continues with the first statement after switch.

```
1 switch (studentGrade) {
2     case 'A':
3         System.out.println("90 - 100");
4         break;
5     case 'B':
6         System.out.println("80 - 89");
7         break;
8     case 'C':
9         System.out.println("70 - 79");
10        break;
11    case 'D':
12        System.out.println("60 - 69");
13        break;
14    default:
15        System.out.println("score < 60");
16 }
```

The `switch` Multiple-Selection Statement

- `switch` does NOT provide a mechanism for testing ranges of values—every value must be listed in a separate case label.

- Each case can have multiple statements (braces are optional)




```
switch (studentGrade) {
    case 90 <= grade:
        System.out.println("A Level");
        break;
    case …:…
}
```

```
switch (studentGrade) {
    case 'A' : {
        System.out.println("90 - 100");
        break;
    }
    case …:…
}
```

The `switch` Multiple-Selection Statement

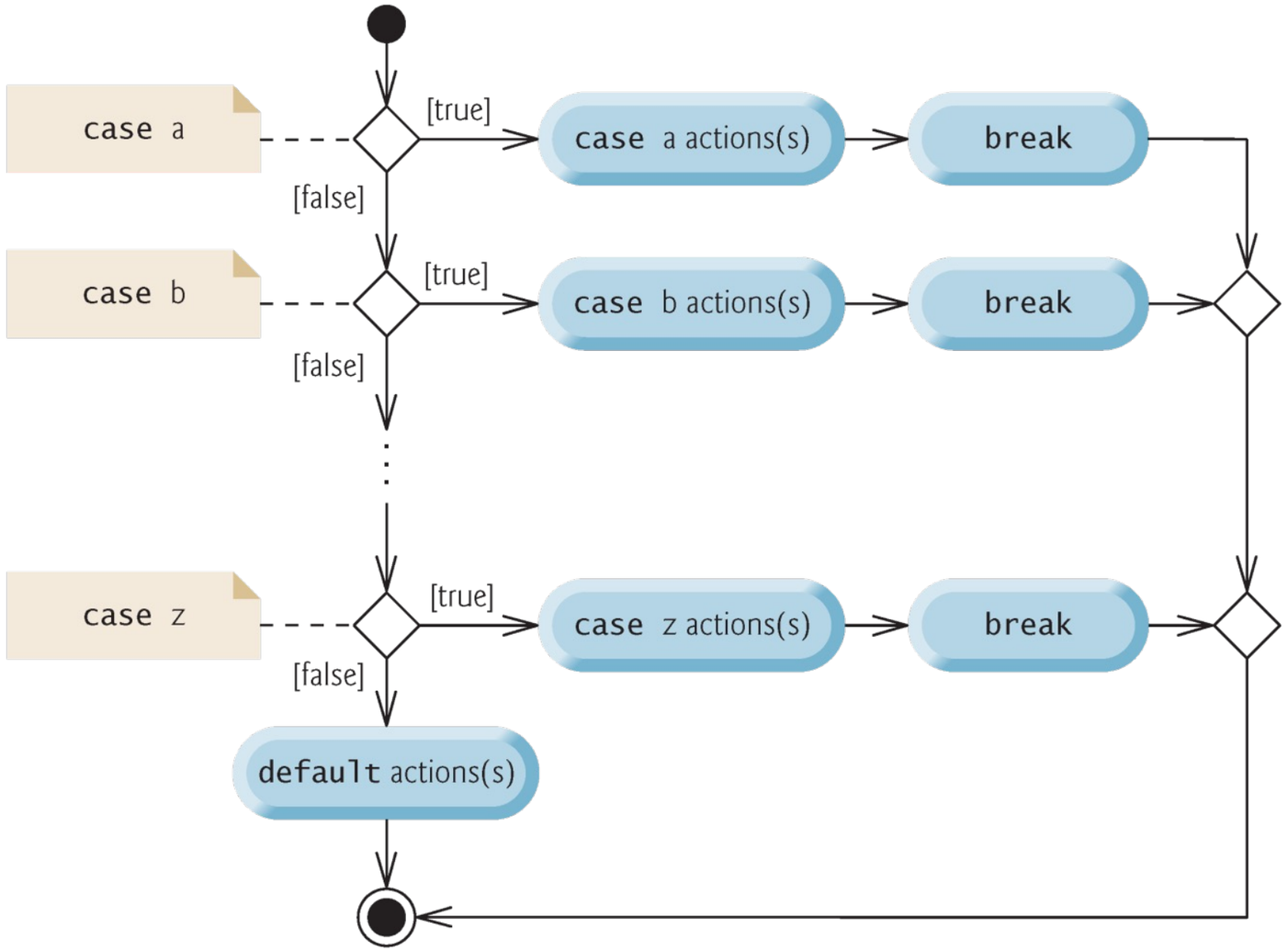
- **Falling through**: Without `break`, the statements for a matching case and subsequent cases execute until a break or the end of the switch is encountered.

```
If studentGrade == ‘A’, then output is

90 – 100
80 – 89
70 – 79
```

```java
switch (studentGrade) {
    case 'A':
        System.out.println("90 - 100");
        // break;
    case 'B':
        System.out.println("80 - 89");
        // break;
    case 'C':
        System.out.println("70 - 79");
        break;
    case 'D':
        System.out.println("60 - 69");
        break;
    default:
        System.out.println("score < 60");
}
```

Execution Flow



The `break` Statement

- The `break` statement, when executed in a `while`, `for`, `do...while` or `switch`, causes immediate exit from that statement.
- Execution continues with the first statement after the control statement.
- Common uses of the `break` statement are to escape early from a loop or to skip the remainder of a `switch`.

The break Statement

```java
// break statement exiting a for statement
public class BreakTest
{
    public static void main(String[] args) {
        int count; // control variable also used after loop terminates
        for (count = 1; count <= 10; count++) { // loop 10 times
            if (count == 5) // if count is 5
                break; // terminate loop
            System.out.printf("%d ", count);
        }
        System.out.printf("\nBroke out of loop at count = %d\n", count);
    }
}
```

```
1 2 3 4
Broke out of loop at count = 5
```

The `continue` Statement

- The `continue` statement, when executed in a `while`, `for` or `do...while`, skips the remaining statements in the loop body and proceeds with the next iteration of the loop.
- In `while` and `do...while` statements, the program evaluates the loop-continuation test immediately after the `continue` statement executes.
- In a `for` statement, the increment expression executes, then the program evaluates the loop-continuation test.

The `continue` Statement

```java
// continue statement terminating an iteration of a for statement
public class ContinueTest
{
    public static void main(String[] args) {
        for (int count = 1; count <= 10; count++) { // loop 10 times
            if (count == 5) // if count is 5
                continue; // skip remaining code in loop
            System.out.printf("%d ", count);
        }
        System.out.println("\nUsed continue to skip printing 5");
    }
}
```

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

Logical Operators

- Logical operators help form complex conditions by combining simple ones:
  - && (conditional AND)
  - || (conditional OR)
  - & (boolean logical AND)
  - | (boolean logical inclusive OR)
  - ^ (boolean logical exclusive OR)
  - ! (logical NOT)
- &, | and ^ are also bitwise operators when applied to integral operands
  - Pay attention to the operand types

The && (Conditional AND) Operator

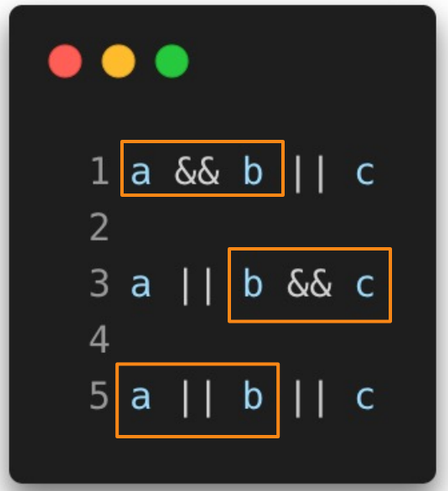
- && ensures that two conditions are both true before choosing a certain path of execution.
- Java evaluates to `false` or `true` for all expressions that include relational operators, equality operators or logical operators.

expression I	expression2	expression I && expression2
false	false	false
false	true	false
true	false	false
true	true	true

The || (Conditional OR) Operator

- || ensures that either or both of two conditions are true before choosing a certain path of execution.
  - Operator && has a higher precedence than operator ||.
  - Both operators associate from left to right.

expression1	expression2	expression1 || expression2
false	false	false
false	true	true
true	false	true
true	true	true



Short-Circuit Evaluation of && and ||

- The expression containing && or || operators are evaluated only until it’s known whether the condition is `true` or `false`.

```
(gender == FEMALE) && (age >= 65)
// Evaluation stops if the first part is false, the whole expression’s value is false

(gender == FEMALE) || (age >= 65)
// Evaluation stops if the first part is true, the whole expression’s value is true
```

The & and | operators

- The boolean logical AND (&) and boolean logical inclusive OR (|) operators are identical to the && and || operators, except that the & and | operators always evaluate both of their operands (they do not perform short-circuit evaluation).
- This is useful if the right operand of the & or | has a required side effect—a modification of a variable's value.

```
1 int b = 0, c = 0;
2 if(true || b == (c = 6)) System.out.println(c);
3 // Prints 0
4
5 int b = 0, c = 0;
6 if(true | b == (c = 6)) System.out.println(c);
7 // Prints 6
```

The ^ Operator

- A simple condition containing the boolean logical exclusive OR (^) operator is true if and only if one of its operands is true and the other is false.
- This operator evaluates both of its operands.

expression1	expression2	expression1 ^ expression2
false	false	false
false	true	true
true	false	true
true	true	false

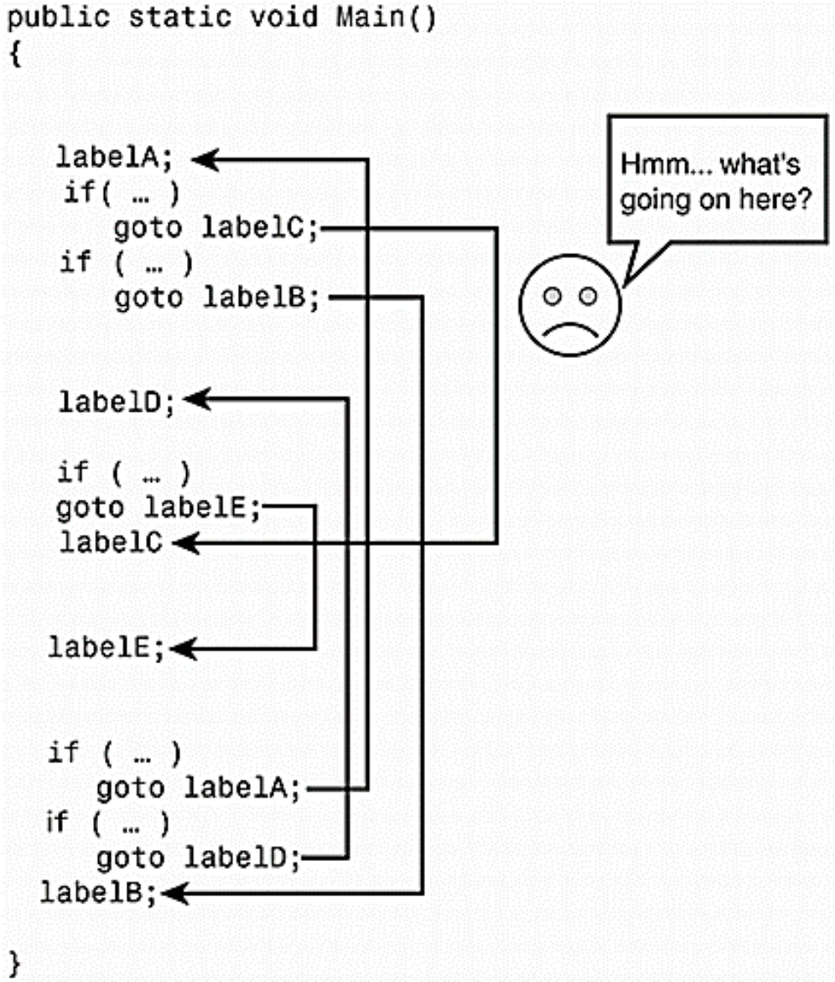
The ! (Logical Not) Operator

- ! (a.k.a., logical negation or logical complement) unary operator “reverses” the value of a condition.

expression	!expression
`false`	`true`
`true`	`false`

Summary: Structured Programming

- Structured programming makes extensive use of controls structures to produce programs with high quality and clarity (in contrast to using simple jumps such as the `goto` statement).
  - `goto` is not supported by Java



Summary: Structured Programming

- Selection is implemented in one of three ways:
  - `if` statement (single selection)
  - `if…else` statement (double selection)
  - `switch` statement (multiple selection)
- The simple `if` statement is sufficient to provide any form of selection—everything that can be done with the if…else and switch can be implemented by combining if statements.

Summary: Structured Programming

- Repetition is implemented in one of three ways:
  - `while` statement
  - `do...while` statement
  - `for` statement
- The `while` statement is sufficient to provide any form of repetition. Everything that can be done with do…while and for can be done with the while statement.