Chapter 3: Control Statements (Part I)

Java™ How to Program, 11th Edition

Instructor: Zhuozhao Li

Online Judge (OJ) Instructions

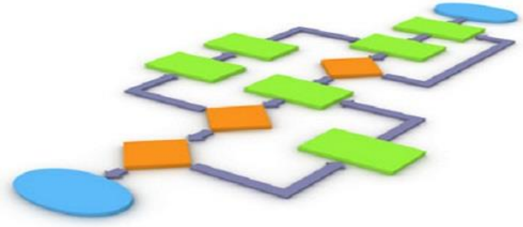
- Sakai
  - https://sakai.sustech.edu.cn/portal/site/3a72b64d-3574-49e2-978d-351b489b3ae7/page/0c0d82ed-9dec-48b7-80be-4ee6bb6b2a84
  - Resource
  - Assignment

- Declaration form submission
  - Assignments
  - One point in your final grade as your attendance score
  - Deadline -- Oct 12, 2021 8:00 PM

Objectives

- To learn and use basic problem-solving techniques
- To develop algorithms using pseudocode (伪代码)
- To use `if` and `if…else` selection statements
- To use `while` repetition statement

Algorithms



- Any computing problem can be solved by executing a series of actions in a specific order

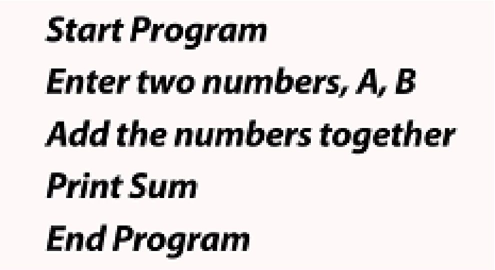
- An algorithm is a procedure for solving a problem in terms of
  - the actions to execute and
  - the order in which these actions execute

- 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) carpool to work.

- Specifying the order in which statements (actions) execute in a program is called program control.

Pseudocode （伪代码）

- Pseudocode is an informal language for developing algorithms



```
Start Program
Enter two numbers, A, B
Add the numbers together
Print Sum
End Program
```

- Similar to 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

Control Structures

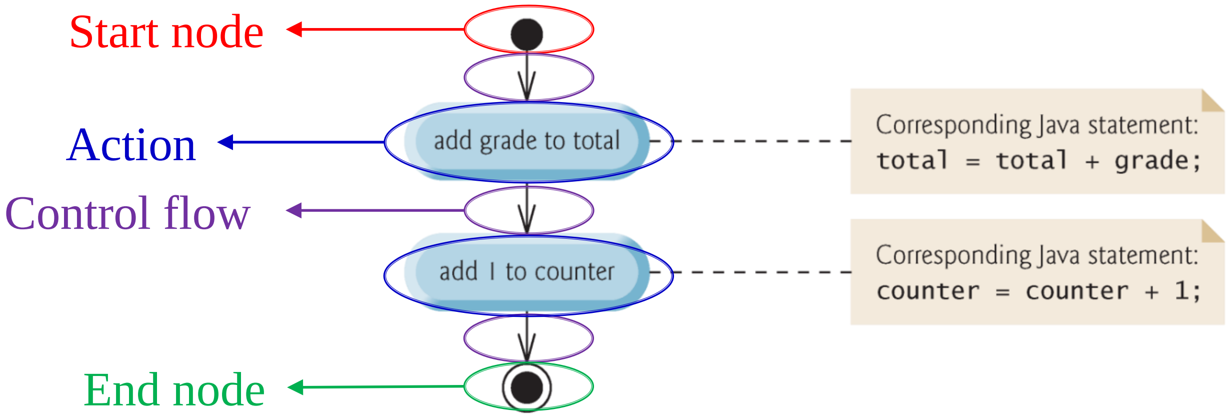
- Sequential execution: normally, statements in a program are executed one after the other in the order in which they are written.

- Transfer of control （控制转移）: 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 （顺序结构）, the selection structure （选择结构） and the repetition structure （循环结构）.

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) in UML （Unified Modeling Language，统一建模语言）， 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

Repetition Structure

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

if Single-Selection Statement

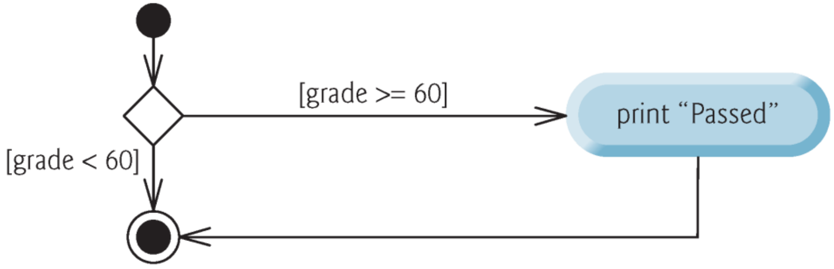
- Pseudocode:

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

- Java code:

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

if Single-Selection Statement



- Diamond, or decision symbol, indicates that a decision is to be made.
- Workflow continues along a path determined by the symbol's guard conditions （约束条件）, 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:

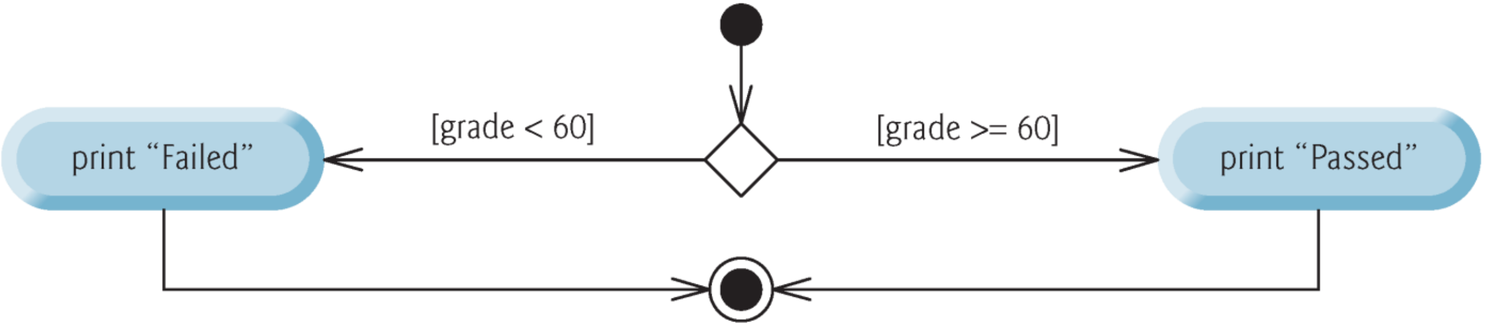
```
If student's grade is greater than or equal to 60
    Print "Passed"
Else
    Print "Failed"
```

- Java code:

```
if ( grade >= 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 ?:

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

The operands and `?:` form a conditional expression.

Shorthand of `if…else`

Conditional operator ?:

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

A boolean expression that evaluates to true or false

The conditional expression takes this value if the boolean expression evaluates to true

The conditional expression takes this value if the boolean expression evaluates to false

Equivalent to

```
String result;
if ( tudentGrade >= 60 )
   result = “Passed”;
else
   result = “Failed”;
```

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"
```

Nested `if…else` statements

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:

```
if ( student1 >= 60 )
   if ( student2 >= 60 )
      System.out.println( “Both students pass!" );
else
   System.out.println( “Student 1 fails" );
```

If-else Matching Rule

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

```
if ( student1 >= 60 )
  if ( student2 >= 60 )
     System.out.println( “Both students pass!" );
  else
     System.out.println( “Student 1 fails" );
```

If-else Matching Rule

What if you really want this effect?

```
if ( student1 >= 60 )
    if ( student2 >= 60 )
       System.out.println( “Both students pass!" );
else
    System.out.println( “Student 1 fails" );
```

Curly braces indicate that the 2nd `if` is the body of the 1st `if`

```
if ( student1 >= 60 ) {
    if ( student2 >= 60 )
       System.out.println( “Both students pass!" );
} else
    System.out.println( “Student 1 fails" );
```

Tip: always use {} to make the bodies of if and else clear.

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

- 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

```
if (x == 1) {
    ;
} else if (x == 2) {
    ;
} else {
    ;
}
```

```
if (x == 1); {
    System.out.println(“Always print”);
}
```

This program is valid, although meaningless.

while Repetition Statement

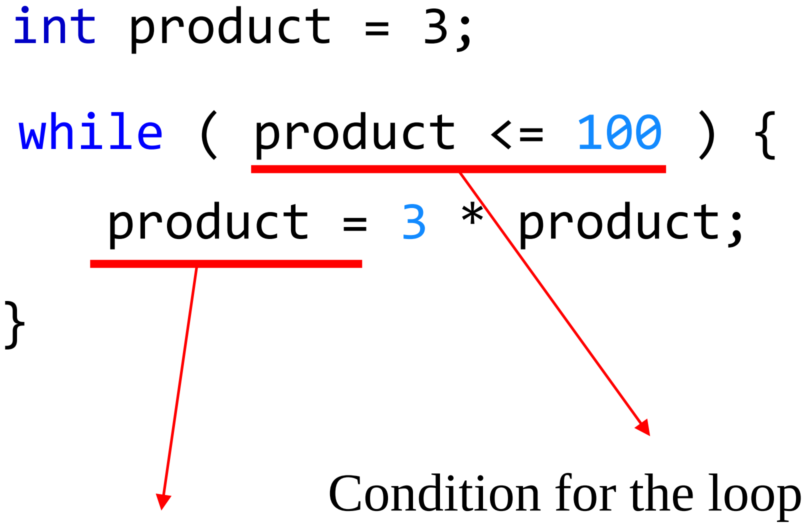
- Repeat an action while a condition remains true
- Pseudocode

```
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 （结束）, and the first statement after the repetition statement executes (otherwise, endless loop，死循环).

Example

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



```
int product = 3;

while ( product <= 100 ) {

    product = 3 * product;

}
```

Condition for the loop to continue

Assignment of a new value

product value

Example

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

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

product value
3

Initial value
Enter loop

Example

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

```
int product = 3;

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

product value
~~3~~
9

Continue in loop

Example

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

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

product value
~~3~~
~~9~~
27

Continue in loop

Example

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

```
int product = 3;

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

product value
~~3~~
~~9~~
~~27~~
81

Continue in loop

Example

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

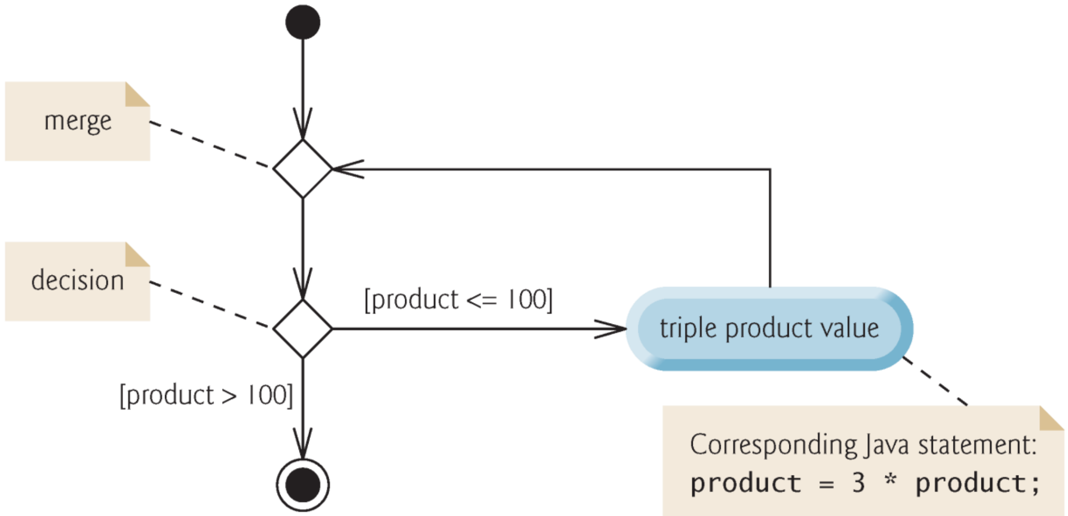
```
int product = 3;

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

product value	
~~3~~	
~~9~~	
~~27~~	
~~81~~	
243	← Loop terminates

while Statement Activity Diagram

- The UML represents both the merge symbol and the decision symbol as diamonds

- The merge symbol joins two flows of activity into one



Corresponding Java statement:
```
product = 3 * product;
```

Will this program terminate?

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

Formulating Algorithms: Counter-Controlled （计数器控制） Repetition

- *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*

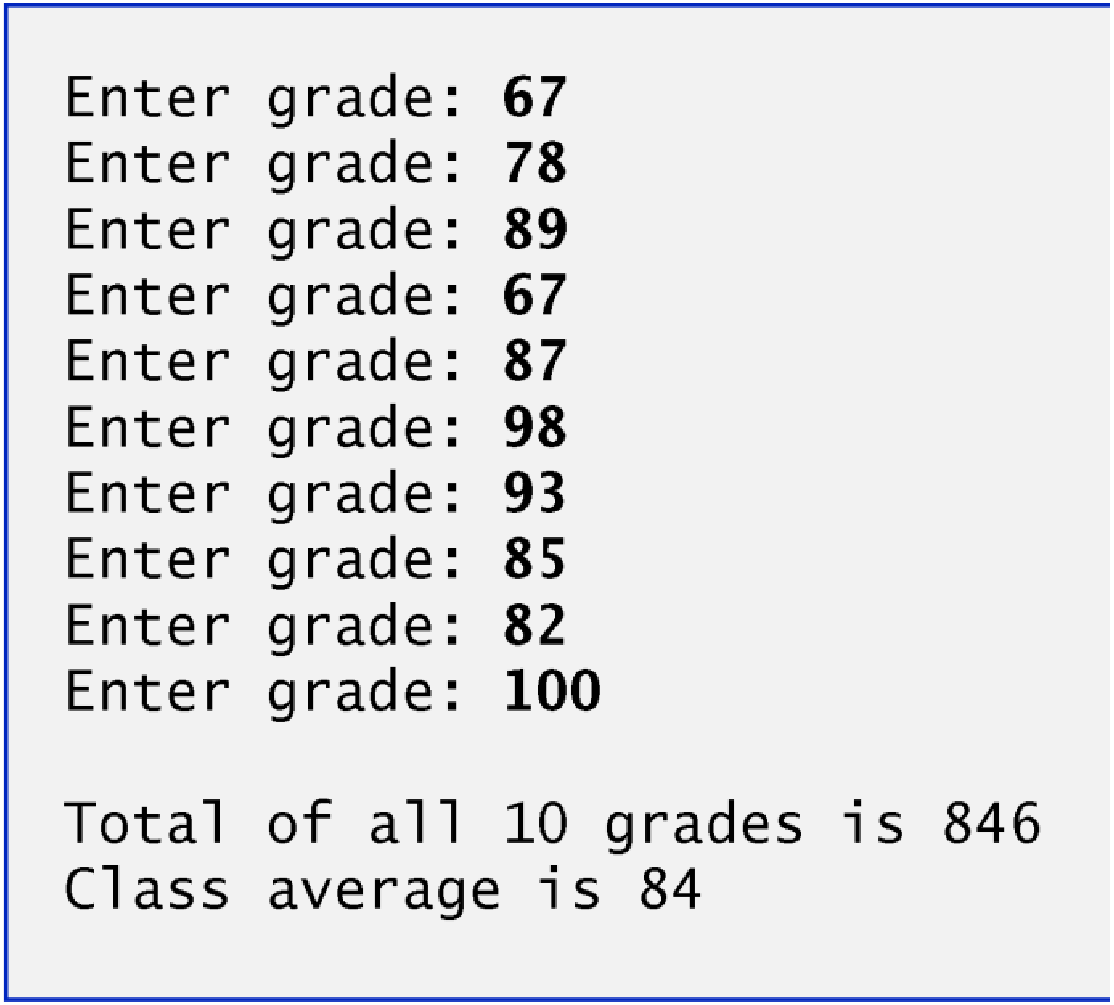
- *Analysis:* the algorithm for solving this problem on a computer must input each grade, keep track of the total of all grades input, perform the averaging calculation and print the result

- 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

- Set total to 0 — total accumulates the sum of several values
- Set student counter to 0 — student counter counts the number of inputs
- While student counter is less than 10
  - Prompt the user to enter the next grade
  - Input the next grade
  - Add the grade to total
  - Add one to the student counter
- Calculate the class average by dividing total to 10
- Print the class average

```
// Counter-controlled repetition: Class-average problem
import java.util.Scanner; // program uses class Scanner

public class ClassAverage {
 // main method begins execution of Java application
    public static void main(String[] args) {
    // create a Scanner to obtain input from the command window
        Scanner input = new Scanner(System.in);

        int total;  // Sum of grades entered by user
        int average;  // Average of grades
        int newGrade;  // New grade value entered by user
        int studentCounter;  // Number of student grades entered

        // Initialization phase
        total = 0;
        studentCounter = 0;
        // Computation phase
        // Loop 10 times
        while ( studentCounter < 10 ) {
            System.out.print("Enter grade: ");  // promt
            newGrade = input.nextInt();  // Input next grade
            total = total + newGrade;  // Add grade to total
            studentCounter = studentCounter + 1; // Increment the student counter by 1
        } // End while

        // Termination phase
        average = total / 10;  // integer division yields integer result

        // Display the results
        System.out.printf( "\nTotal of all 10 grades is %d\n", total );
        System.out.printf( "\nClass average is %d\n", average );
    } // End method main
}
```



```
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
```

Formulating Algorithms: Sentinel-Controlled Repetition (边界值控制循环)

- ***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 know earlier, but here how can the program determine when to stop the input of grades?

Formulating Algorithms: Sentinel-Controlled Repetition

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”.

Marking the end of inputs

92, 77, 68, 84, 35, 72, 95, 79, 88, 84, **-1**

Formulating Algorithms: Sentinel-Controlled Repetition

- 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

欢 迎 光 临
请出示会员积分卡 Please Membership Card
欢 迎 光 临
请出示会员积分卡 Please Membership Card

One of the left items? Of course not…

```
Set total to 0
Set student counter to 0
```

total stores the sum of grades

counter stores the number grades

```
Prompt the user to enter the first grade
Input the first grade (possibly the sentinel)
```

Try to take an input

```
While the user has not entered the sentinel
    Add the grade to total
    Add one to the student counter
    Prompt the user to enter the next grade
    Input the next grade
```

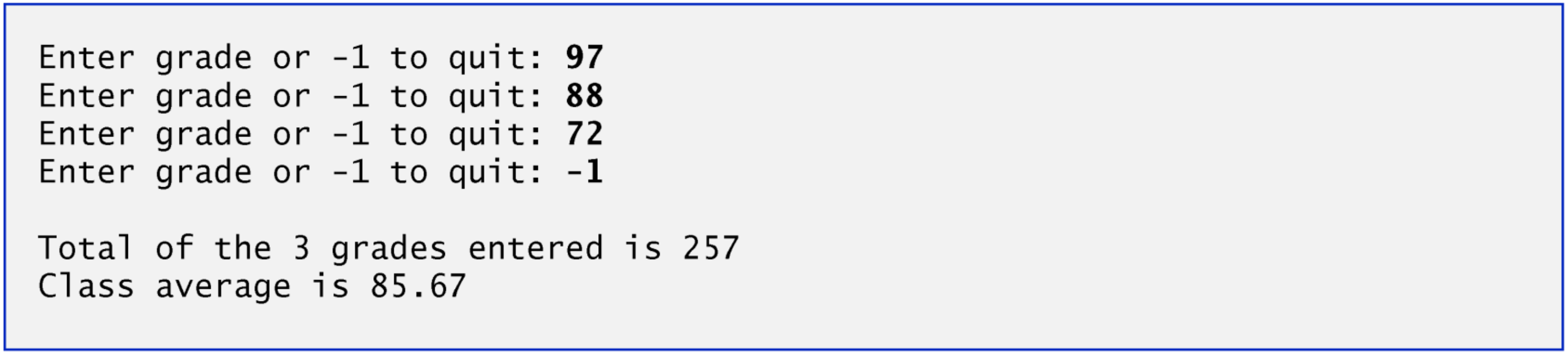
If no sentinel value seen, repeat the process

```
If the student counter is not 0
    Calculate the class average by dividing total to student counter
    Print the class average
Else
    Print "No grade was entered"
```

Compute and print average (avoid division by 0)

```
// Counter-controlled repetition: Class-average problem
import java.util.Scanner; // program uses class Scanner

public class ClassAverage1 {
 // main method begins execution of Java application
   public static void main(String[] args) {
  // create a Scanner to obtain input from the command window
     Scanner input = new Scanner(System.in);

     int total;  // Sum of grades entered by user
     int newGrade;  // New grade value entered by user
     int studentCounter;  // Number of student grades entered
     double average;  // Average of grades with decimal point

     // Initialization phase
     total = 0;
     studentCounter = 0;

     // Computation phase
     // prompt for input and read grade from user
     System.out.print("Enter grade or -1 to quit: ");  // promt
     newGrade = input.nextInt();  // Input next grade
```

```
        while ( newGrade != -1 ) {
            total = total + newGrade;  // Add grade to total
            studentCounter = studentCounter + 1; // Increment the student counter by 1
            System.out.print("Enter grade or -1 to quit: ");  // promt
            newGrade = input.nextInt();  // Input next grade
        } // End while

        // Termination phase
        // if there is at least one grade
        if (studentCounter > 0) {
            average = (double) total / studentCounter;  // integer division yields integer
result

            // Display the results
            System.out.printf( "\nTotal of all %d grades is %d\n", studentCounter, total );
            System.out.printf( "\nClass average is %.2f\n", average );
        } else {
            System.out.println( "No grade was entered" );
        }
    } // End method main
}
```



```
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
```

Type Cast （类型转换）

```
int total;

int studentCounter;

double average;
```

```
average = (double) total / studentCounter;
```

The unary cast operator creates a temporary floating-point copy of its operand

- Cast operator performs explicit conversion (or type cast).
- This precedence is one level higher than the binary arithmetic operator, e.g., *, / and %.
- The value stored in the operand is unchanged (e.g., the value of total is unchanged!!!)

Type Promotion (类型提升)

```
int total;                average = (double) total / studentCounter;
int studentCounter;
double average;
```

Type promotion from int to double

- Java evaluates only arithmetic expressions in which the operands' types are identical.
- Promotion (or implicit 隐含 conversion) performed on operands.
- In the above expression, the `int` value of studentCounter is promoted to a `double` value for computation.
- byte->short->int->long->float->double

More on Cast Operators

- Cast operators are available for any type.
- Cast operator formed by placing parentheses around the name of a type.

The Scope of Variables （变量作用域）

- 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

Is this correct?

```
public class Scope {

    // main method begins execution of Java application
    public static void main(String[] args) {
        int a = 3;
    } // end method main


    public static void foo() {
         a = 3;  ✖
    }
}
```

"a" is a local variable in main method, cannot be used outside of main

Is this correct?

```
public class Scope {

    // main method begins execution of Java application
    public static void main(String[] args) {
        int a = 3;
        int a = 4;    ✖
    } // end method main


}
```

“a” cannot be defined twice because “a” has a method-level scope

Is this correct?

```
public class Scope {
 // main method begins execution of Java application
   public static void main(String[] args) {
      int a = 3;
      b = a + 4;   ✖
   } // end method main
}
```

“b” is not defined before use

Is this correct?

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

This is valid

Block Scope (块作用域)

- A variable can be declared inside a pair of braces “{” and “}”. It can be only used within the braces only.

```
int product = 3;
while ( product <= 100 ) {
    int x = 3 * product;
}
System.out.println(x);
```

✖

“x” is not defined

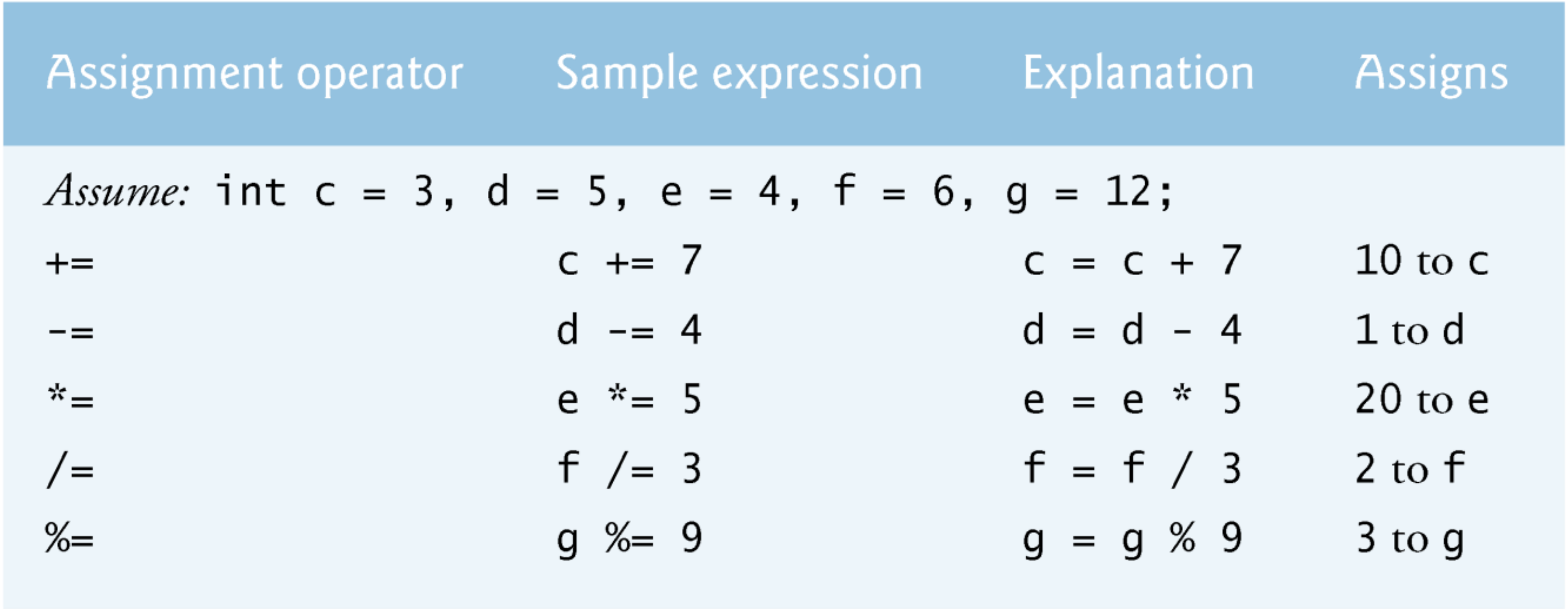
Compound Assignment Operators（组合赋值操作符）

- 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;`



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. 3.11 | Arithmetic compound assignment operators.

Increment and Decrement Operators (自增、自减运算符)

- 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 （前缀） 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 （后缀） increment or postfix decrement operator, respectively.

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

Preincrementing/Predecrementing

- 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.

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

Postincrementing/Postdecrementing

- 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.

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

Note the Difference

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

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

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

Note the Difference

```
int a = 6;
int b = a++;
```

Equivalent to

```
int a = 6;

int b = a;
a = a + 1;
```

```
int a = 6;
int b = ++a;
```

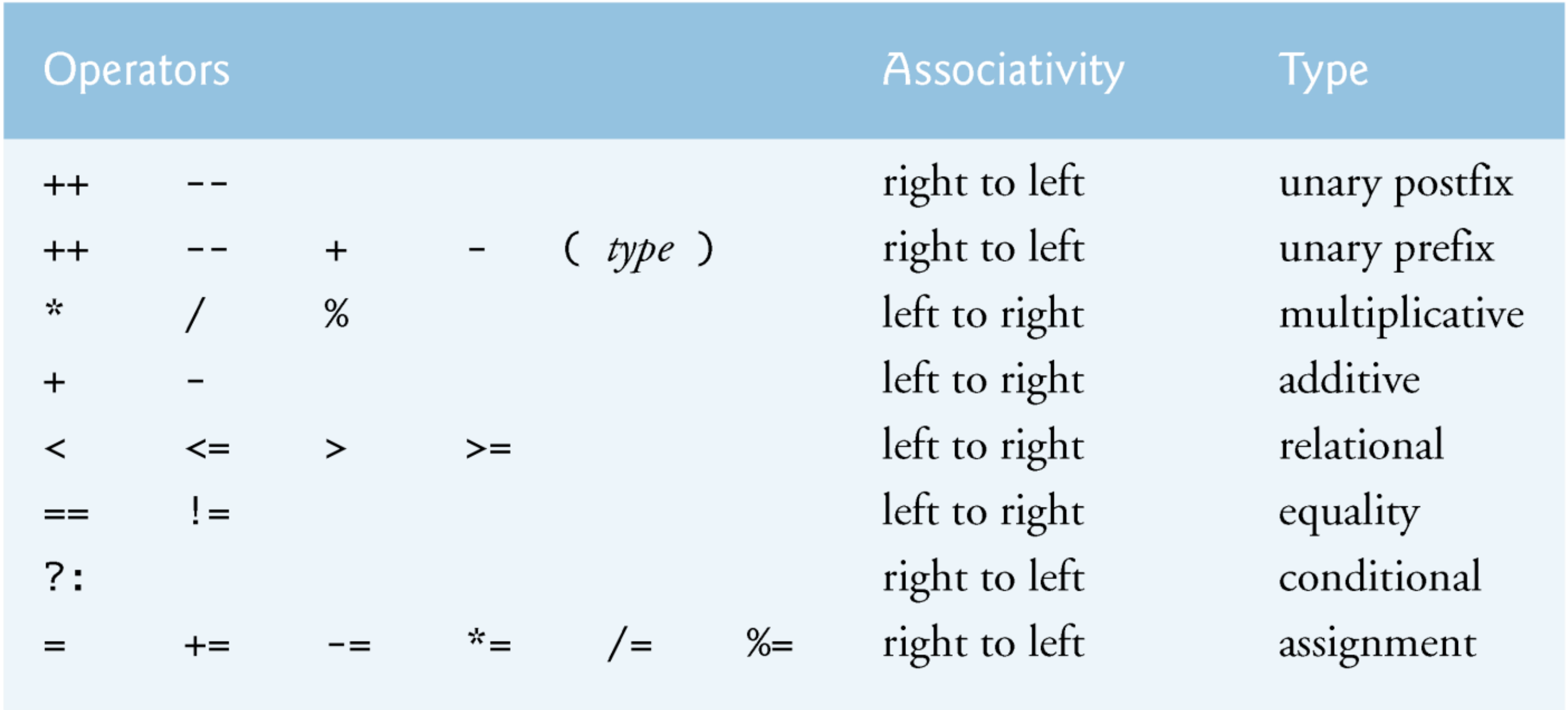
Equivalent to

```
int a = 6;

a = a + 1;
int b = a;
```

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

The Operators Introduced So Far

Precedence



Operators	Associativity	Type
`++ --`	right to left	unary postfix
`++ -- + - (` *type* `)`	right to left	unary prefix
`* / %`	left to right	multiplicative
`+ -`	left to right	additive
`< <= > >=`	left to right	relational
`== !=`	left to right	equality
`?:`	right to left	conditional
`= += -= *= /= %=`	right to left	assignment

Fig. 3.14 | Precedence and associativity of the operators discussed so far.

Please practice each of the operators by yourself ☺

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!"

```
Initialize passes to zero
Initialize failures to zero
Initialize student counter to one

While student counter is less than or equal to 10
    Prompt the user to enter the next exam result
    Input the next exam result

    If the student passed
        Add one to passes
    Else
        Add one to failures

    Add one to student counter

Print the number of passes
Print the number of failures

If more than eight students passed
    Print "Bonus to instructor!"
```

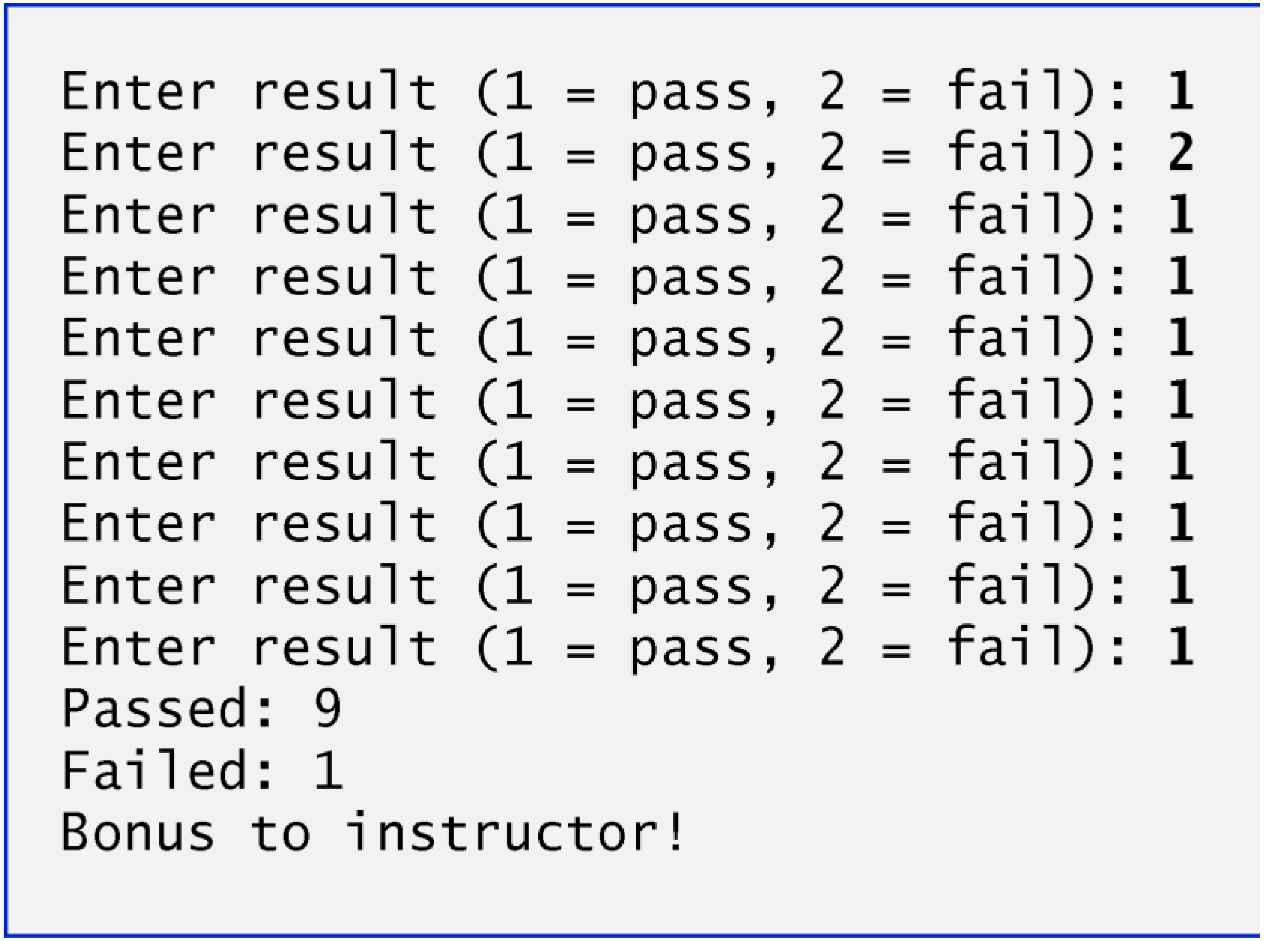
Two variables defined: passes and failures

Counter-controlled repetition

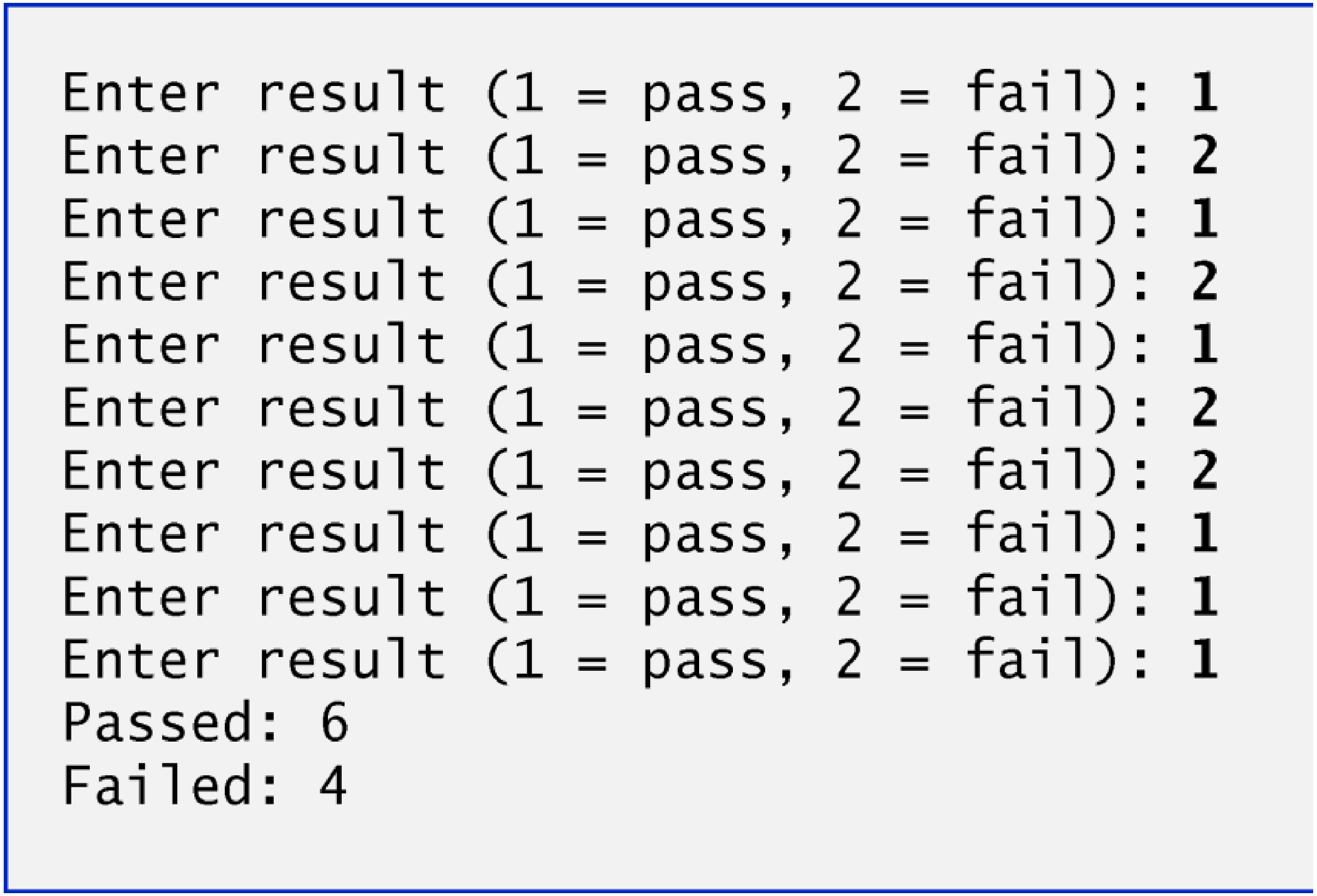
`if…else` nested in `while`

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

public class Analysis {
 // main method begins execution of Java application
   public static void main(String[] args) {
   // create a Scanner to obtain input from the command window
      Scanner input = new Scanner(System.in);

      int passes = 0;  // Number of passes
      int failures = 0;  // Number of failures
      int studentCounter = 0;  // Number of student grades entered
      int result;  // One exam result entered from user

      // Computation phase
      // Loop 10 times
      while ( studentCounter < 10 ) {
         System.out.print("Enter result (1 = pass, 2 = fail): ");  // prompt
         result = input.nextInt();  // Input next result
```

```
            // if ... else nested in while
            if (result == 1) {
                passes += 1;
            } else {
                failures += 1;
            }

            // Increment the studentCounter to make sure the loop terminates
            studentCounter += 1;

        } // End while

        // Display the results
        System.out.printf( "Passed: %d\nFailed: %d\n", passes, failures );
        // Determine whether there are more than 8 students passed
        if ( passes > 8 ){
            System.out.println( "Bonus to instrcutor" );
        }
    } // End method main
}
```



```
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
```

Online Judge (OJ) Instructions

- Sakai
  - https://sakai.sustech.edu.cn/portal/site/3a72b64d-3574-49e2-978d-351b489b3ae7/page/0c0d82ed-9dec-48b7-80be-4ee6bb6b2a84
  - Resource
  - Assignment

- Declaration form submission
  - Assignments
  - One point in your final grade as your attendance score
  - Deadline -- Oct 12, 2021 8:00 PM