Chapter 2
Control Statements (Part Ⅰ)

Yepang Liu

liuyp1@sustech.edu.cn

# **Agenda**

▸ Types of control structures (三种控制结构简介)

▸ Selection statements: if, if…else (选择/条件语句)

▸ Repetition statements: while (循环语句)

▸ Case Study (案例分析)

Control Structures

Control structures specify the flow of control in programs

Most simple case: Sequential Flow (顺序执行)

- Actions are executed one after the other in the order in which they are specified.

- Unless directed otherwise, computers execute Java statements one after the other in the order in which they're written.

- Such structures are called <u>sequence structures</u>.

Sequence

Statement 1

Statement 2
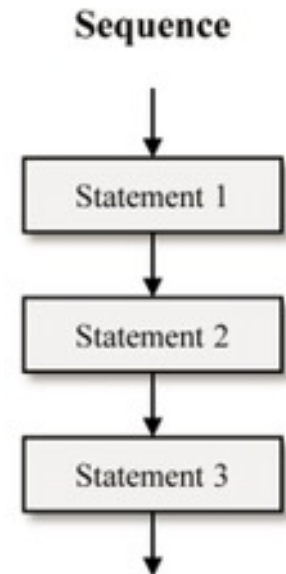
Statement 3
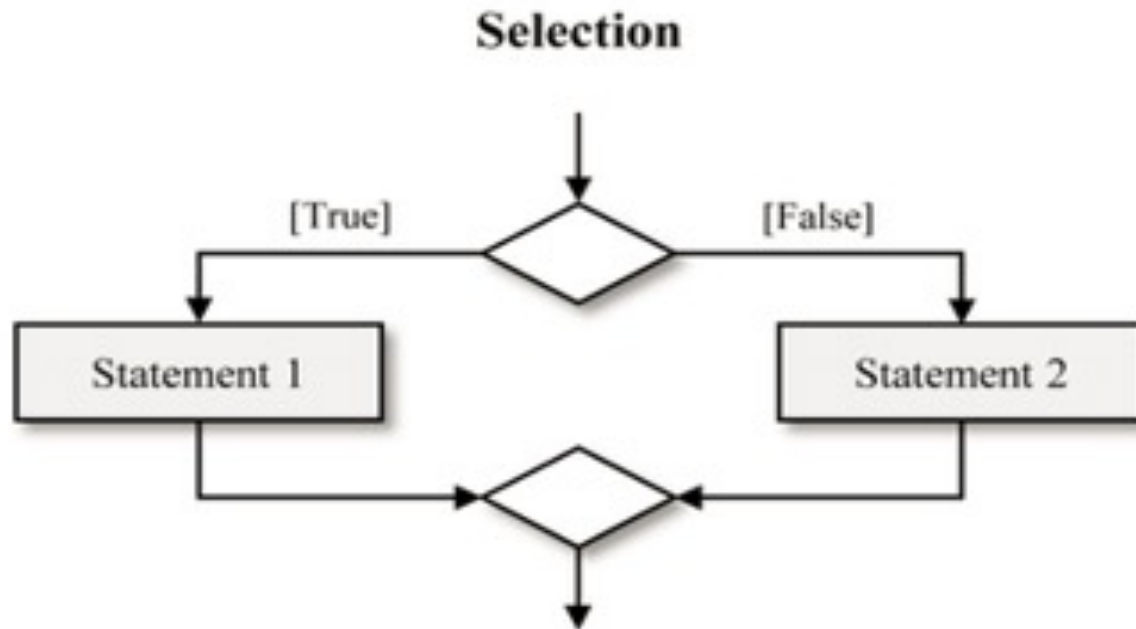
Control Structures

What if we need to make selections?

3

Control Structures

Conditional Flow (Selection Structure)

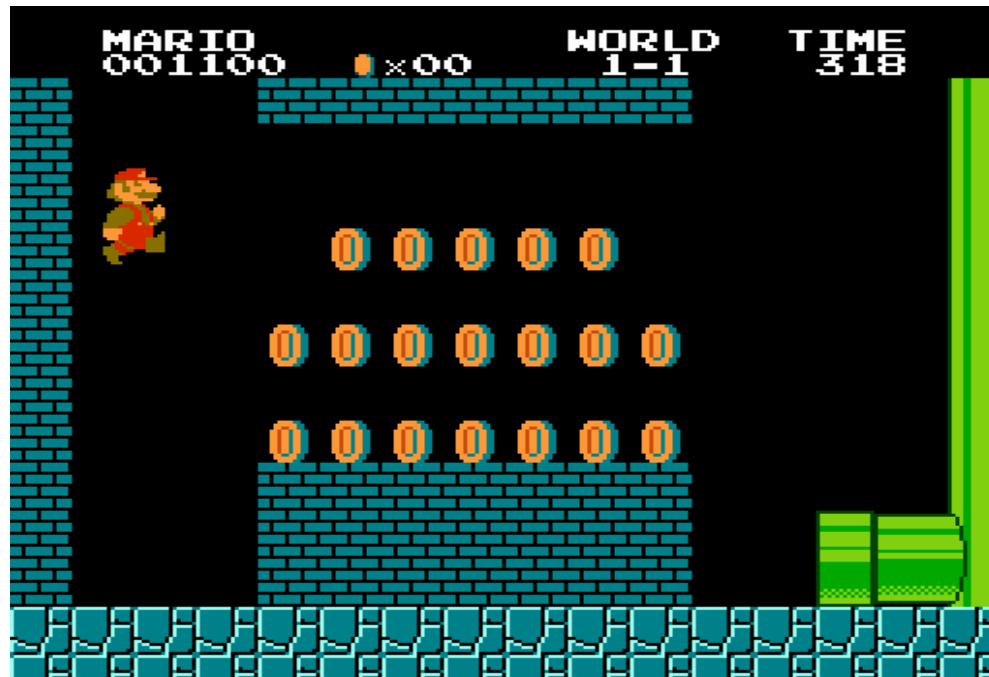Execute one or more statements when certain condition(s) are met

Selection Structure (选择**)**

▸ Three types of selection statements：

- ▪ if statement (单路选择)

- ▪ if…else statement (双路选择)

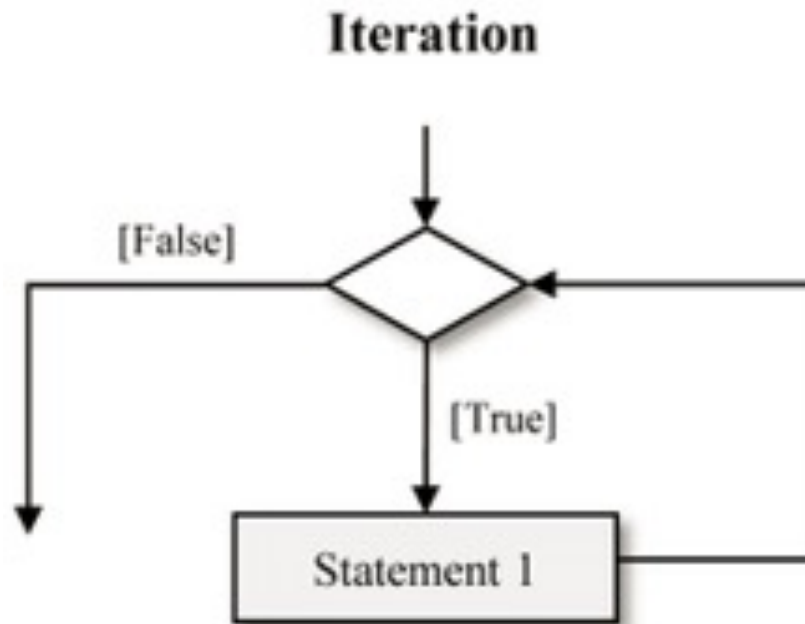- ▪ switch statement (多路选择)

Control Structures

What if we need to repeat an action?

Control Structures

Repetitive Flow (Repetition Structure)

Repeat an action a certain number of times or while a condition holds.

Repetition Structure (循环)

▸ 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

Agenda

▸ Types of control structures (三种控制结构简介)

▸ Selection statements: if, if…else (选择/条件语句)

▸ Repetition statements: while (循环语句)

▸ Case Study (案例分析)

if Single-Selection Statement

▸ If the condition is TRUE, execute the statement; if the condition is FALSE, nothing happens (i.e., one choice)
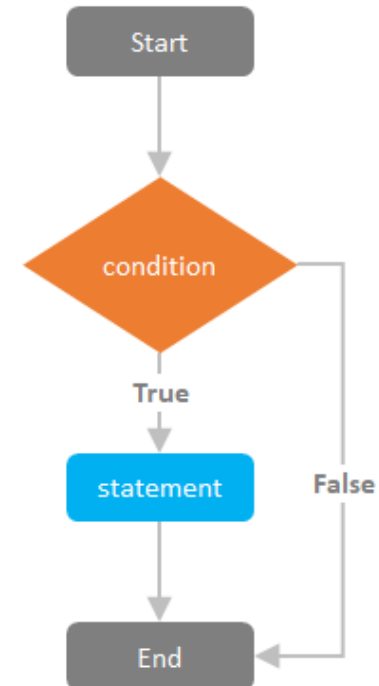
▸ Pseudocode (伪代码)

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

   Print "Passed"

▸ Java code

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

}
```

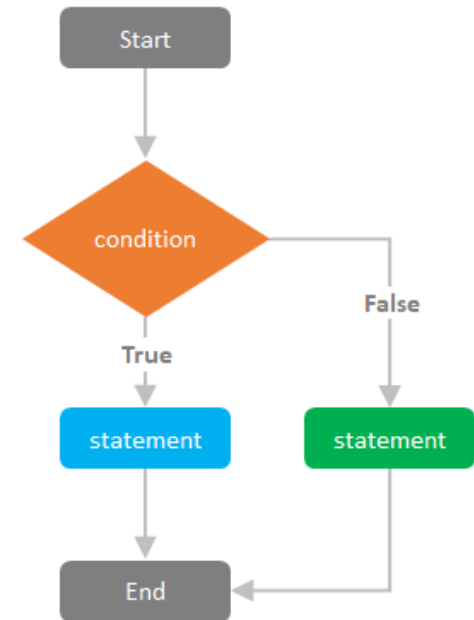# if…else Double-Selection Statement

▸ If the condition is TRUE, execute the statement; if the condition is FALSE, execute another statement (i.e., two choices)

▸ Pseudocode:

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

▸ Java code:

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

Conditional operator ?:

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

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

Shorthand of `if…else`

Ternary operator (三元操作符)：We need to specify 3 parts.

Equivalent to

```
String result;
if ( studentGrade >= 60 )
    result = "Passed";
else
    result = "Failed";
```

Nested `if` Statements (嵌套)

```java
if (score >= 90.0)
    System.out.print("A");
else
    if (score >= 80.0)
        System.out.print("B");
    else
        if (score >= 70.0)
            System.out.print("C");
        else
            if (score >= 60.0)
                System.out.print("D");
            else
                System.out.print("F");
```

① Condition: score >= 90

② Condition: score < 90

Nested `if` Statements (嵌套)

```java
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

① Condition: score < 90 and score >= 80

② Condition: score < 90 and score < 80 (that is, score < 80)
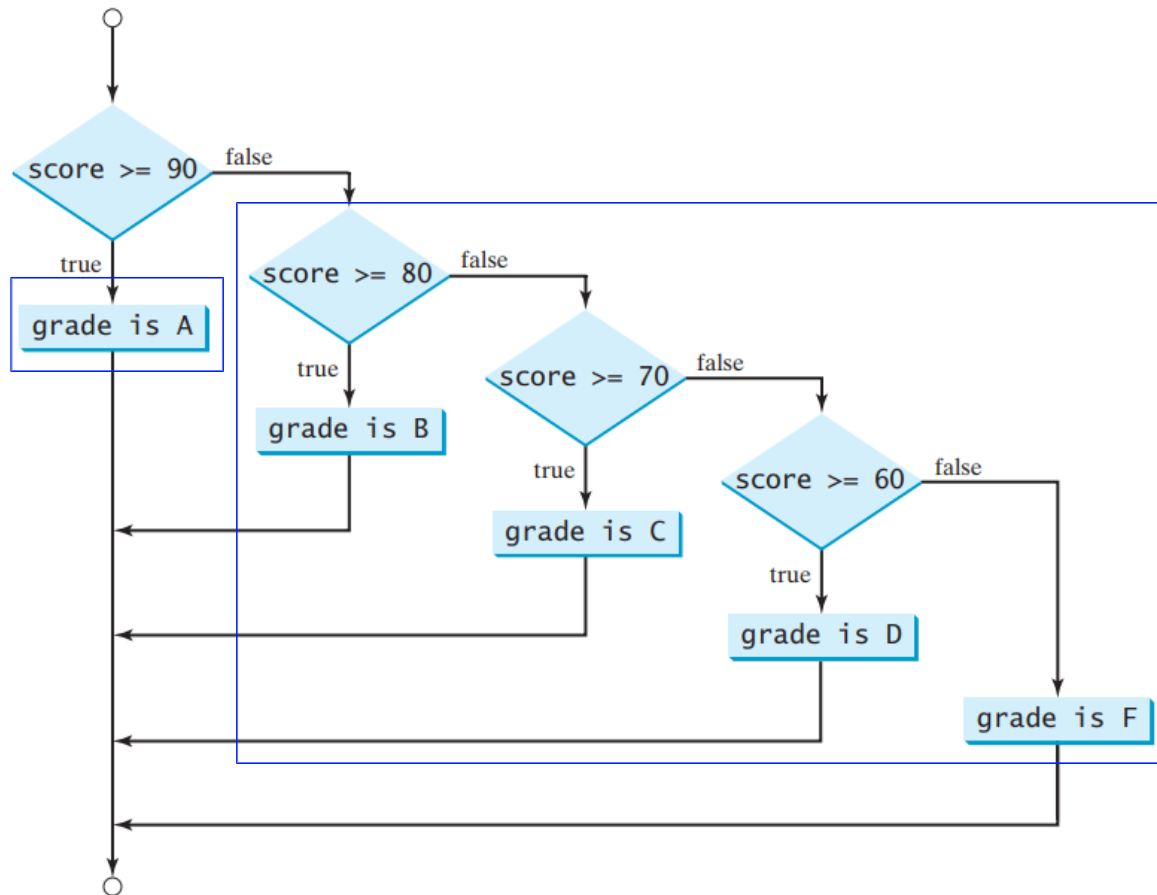
Nested `if` Statements (嵌套)

```java
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

① Condition: score < 80 and score >= 70

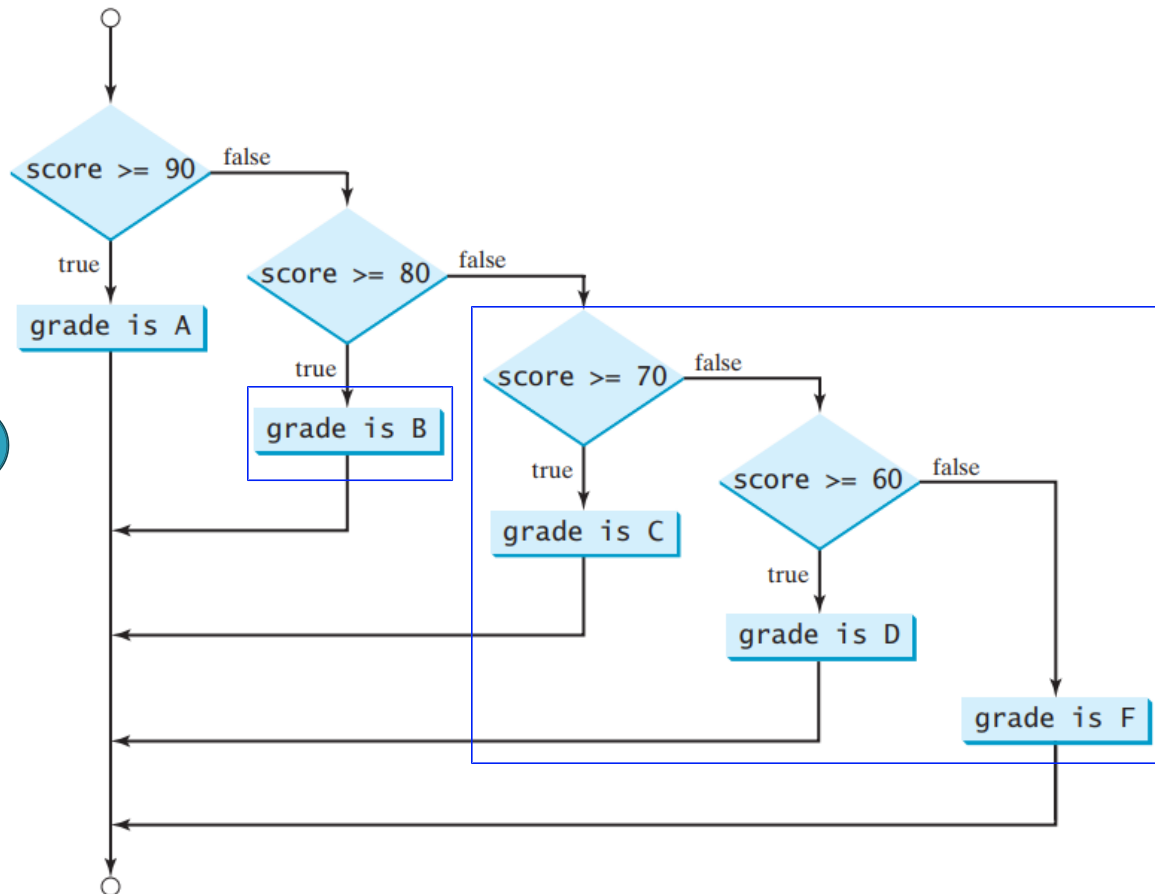**②** Condition: score < 80 and score < 70 (that is, score < 70)

Nested `if` Statements (嵌套)

```java
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");  ①
      else
        System.out.print("F");  ②
```
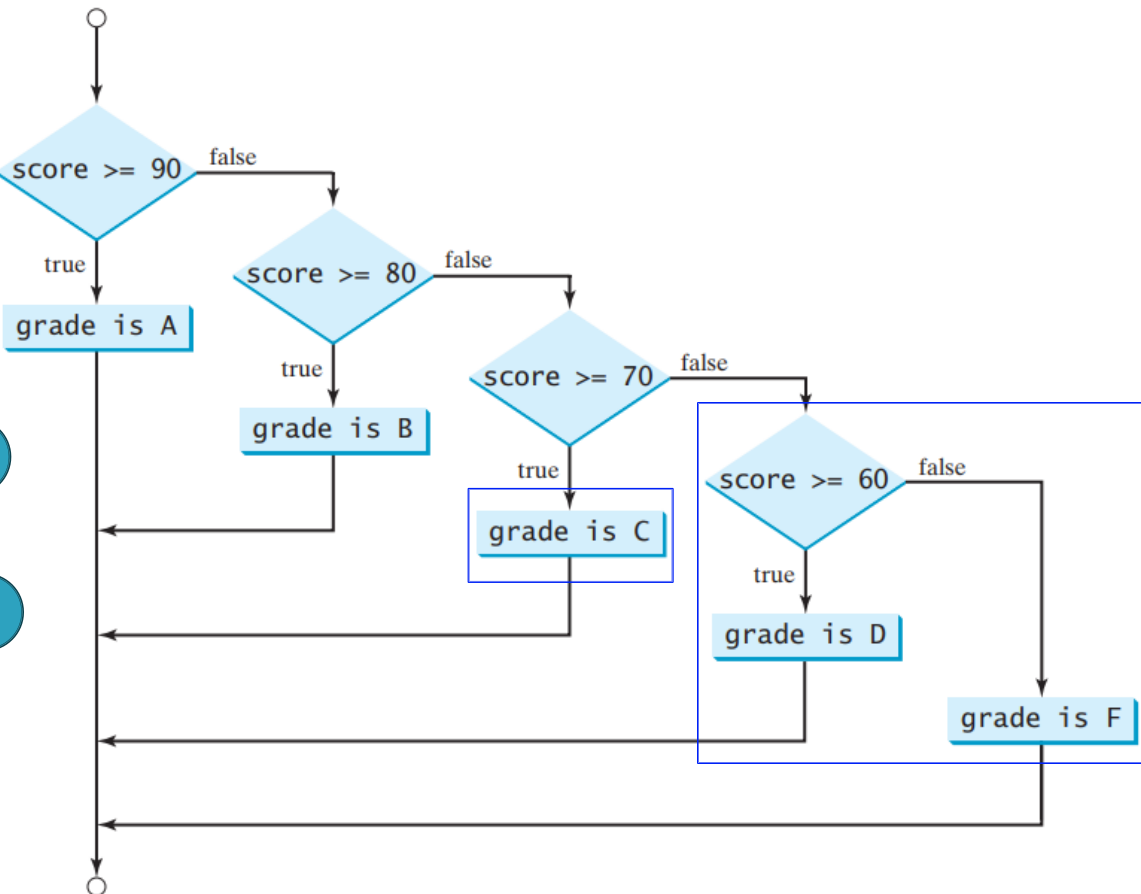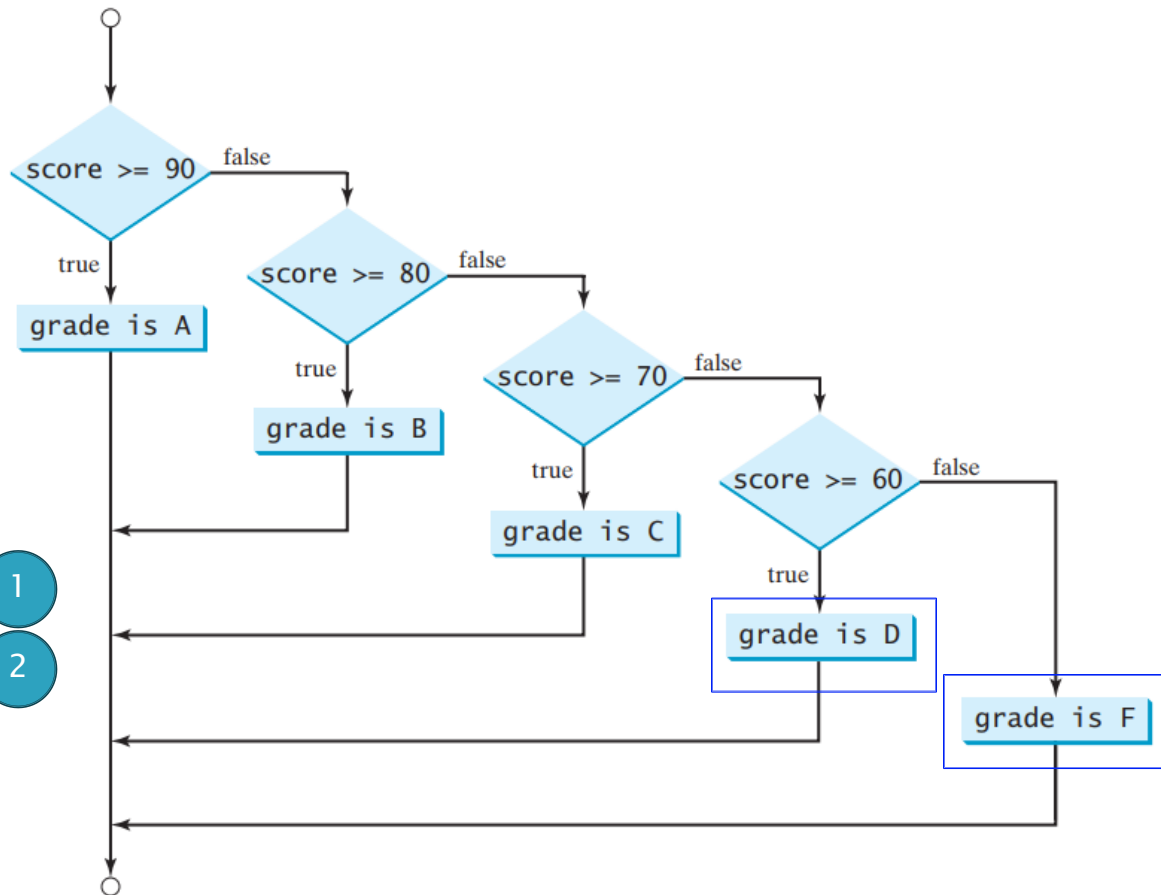

① Condition: score < 70 and score >= 60

② Condition: score < 70 and score < 60 (that is, score < 60)

A More Elegant Version

▸ Use *multi-way* *if...else* *statements* (多分支if-else语句) to specify a new condition to test, if the first condition is false

```
if (score >= 90.0)
  System.out.print("A");
else
  if (score >= 80.0)
    System.out.print("B");
  else
    if (score >= 70.0)
      System.out.print("C");
    else
      if (score >= 60.0)
        System.out.print("D");
      else
        System.out.print("F");
```

Equivalent
══════════

```
if (score >= 90.0)
  System.out.print("A");
else if (score >= 80.0)
  System.out.print("B");
else if (score >= 70.0)
  System.out.print("C");
else if (score >= 60.0)
  System.out.print("D");
else
  System.out.print("F");
```

This is better

Common Error 1: Unnecessary condition for `else`

```
String level = "";
if(grade>=90) {
        level = "A";
}
else if(grade>=75) {
        level = "B";
}
else if(grade>=60) {
        level = "C";
}
else (grade < 60) {
        level = "F";
}
System.out.println(level);
```

Can the code compile?

```
17  String level = "";
18  if(grade>=90) {
19      level = "A";
20  }
21  else if(grade>=75) {
22      level = "B";
23  }
24  else if(grade>=60) {
25      level = "C";
26  }
27  else (grade<60) {
28      level = "F";
29  }
30  System.out.println(level);
31
```

Common Error 2: Forgetting necessary braces

```java
if (radius >= 0) {
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

Correct version: The program prints the area of a "valid" circle after calculation.

```java
if (radius >= 0)
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
```

Incorrect version: The program prints the area regardless of the condition "radius >= 0".

Common Error 3: Dangling `else` ambiguity

```
int i = 1, j = 2, k = 3;

if (i > j)
    if (i > k)
        System.out.println("A");
else
        System.out.println("B");
```

(a)

You might think that the `else` part matches wit the first `if`.

Wrong! The indentation is misleading.

`if-else` matching rule

- Extra spaces are irrelevant in Java (only for formatting).
- The Java compiler always associates an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{` and `}`)

Common Error 3: Dangling `else` ambiguity

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
else
    System.out.println("B");
```
(a)

Equivalent

This is better
with correct
indentation

```
int i = 1, j = 2, k = 3;

if (i > j)
  if (i > k)
    System.out.println("A");
  else
    System.out.println("B");
```
(b)

Now `else` matches with
the first `if`!

```
int i = 1, j = 2, k = 3;

if (i > j) {
  if (i > k)
    System.out.println("A");
}
else
  System.out.println("B");
```

Common Error 4: Wrong semicolon at the `if` line

▸ Just as a block (代码块) can be placed anywhere a single statement can be placed, it's also possible to have an empty statement (空语句)

▸ The empty statement is represented by placing a semicolon (;) where a statement would normally be

Logic error

```
if (radius >= 0);
{
  area = radius * radius * PI;
  System.out.println("The area "
    + " is " + area);
}
```

(a)

Agenda

▸ Types of control structures (三种控制结构简介)

▸ Selection statements: if, if…else (选择/条件语句)

▸ Repetition statements: while (循环语句)

▸ Case Study (案例分析)

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
  Exit the shop

- 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
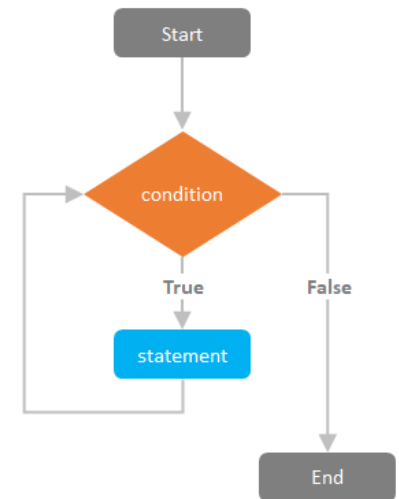
Example

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

```java
int product = 3;

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

The body of the while loop

Example

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

```java
int product = 3;

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

// other statements
```

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;

}

// other statements
```

product value
3

Condition true
Enter loop body
(1st time)

Example

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

```java
int product = 3;

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

// other statements
```

product value
~~3~~
9

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;
}

// other statements
```

product value
~~3~~
9

Condition true
Enter loop body
(2nd time)

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;

}

// other statements
```

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

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;
    }

    // other statements
```

product value
3̶
9̶
27

Condition true
Enter loop body
(3rd time)

Example

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

```java
int product = 3;

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

// other statements
```

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

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;
}
// other statements
```

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

Condition true
Enter loop body
(4th time)

33

Example

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

```java
int product = 3;

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

// other statements
```

product value
~~3~~
~~9~~
~~27~~
~~81~~
243

Example

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

```java
int product = 3;
while ( product <= 100 ) {
    product = 3 * product;
}
// other statements
```

product value
~~3~~
~~9~~
~~27~~
~~81~~
243

Condition false
Exit loop

Example

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

```java
int product = 3;

while ( product <= 100 ) {

    product = 3 * product;

}
```

⟶ `// other statements`

The first statement after the while statement will be executed

product value
3
9
27
81
243

Will This Program Terminate?
（下面程序的循环会终止吗？）

```
int product = 3;

while ( product <= 100 ) {

    int x = 3 * product;

}

// other statements
```

Endless Loop

▸ The condition remains TRUE and the loop never terminates

```
int product = 3;

  while ( product <= 100 ) {

      int x = 3 * product;

  }

  // other statements
```

The value of product never changes,
and the condition is always TRUE!
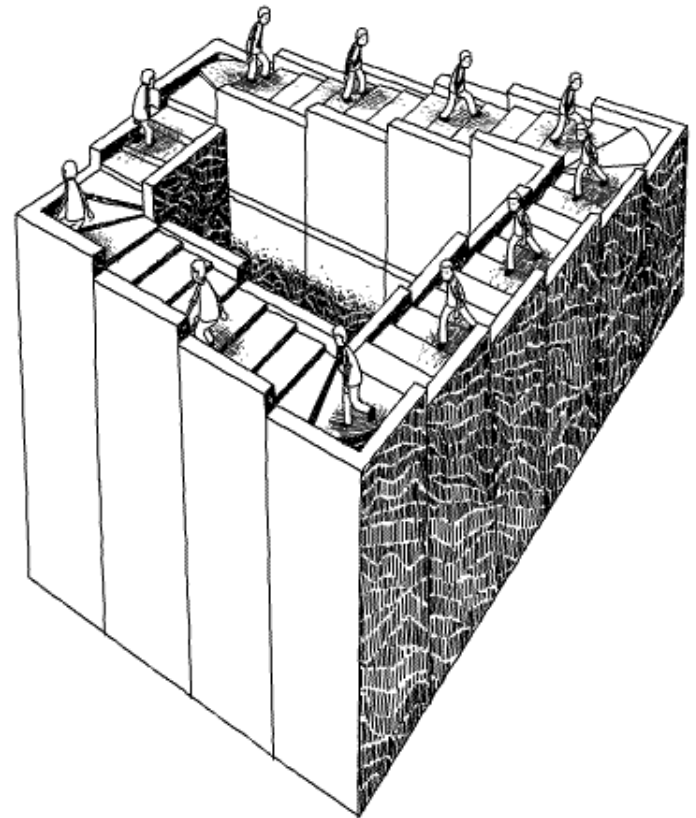
Agenda

▸ Types of control structures (三种控制结构简介)

▸ Selection statements: if, if…else (选择/条件语句)

▸ Repetition statements: while (循环语句)

▸ Case Study (案例分析)

Counter-Controlled Repetition
（计数器控制的循环）

- *Class-Average Problem: A class of **<u>ten students</u>** 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.

The Pseudo Code
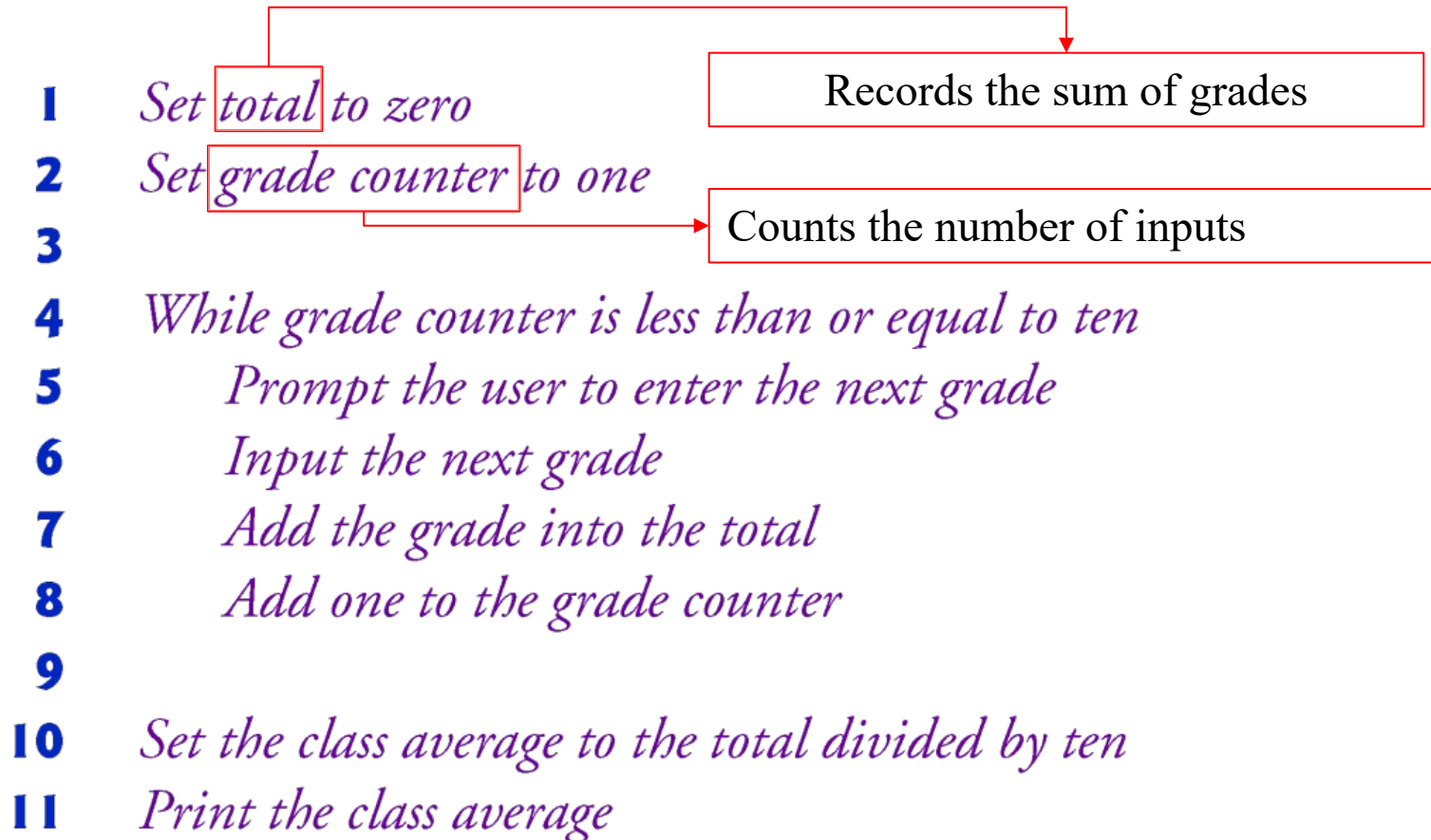
1. Set *total* to zero   Records the sum of grades
2. Set *grade counter* to one
3.   Counts the number of inputs
4. While grade counter is less than or equal to ten
5.   Prompt the user to enter the next grade
6.   Input the next grade
7.   Add the grade into the total
8.   Add one to the grade counter
9.
10. Set the class average to the total divided by ten
11. Print the class average

Translate to Java Code

```java
// Counter-controlled repetition: Class-average problem
import java.util.Scanner;
public class ClassAverage {

  public static void main(String[] args) {

    // create Scanner to obtain input from command window
    Scanner input = new Scanner(System.in);

    int total; // sum of grades entered by user
    int gradeCounter; // number of the grade to be entered next
    int grade; // grade value entered by user
    int average; // average of grades

    // initialization phase
    total = 0; // initialize total
    gradeCounter = 1; // initialize loop counter
```

Translate to Java Code

```java
   // processing phase
   while(gradeCounter <= 10) { // loop 10 times
      System.out.print("Enter grade: "); // prompt
      grade = input.nextInt(); // input next grade
      total = total + grade; // add grade to total
      gradeCounter = gradeCounter + 1; // increment counter by 1

   } // end while

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

   // display total and average of grades
   System.out.printf("\nTotal of all 10 grades is %d\n", total);
   System.out.printf("Class average is %d\n", average);

} // end main
} // end class ClassAverage
```

A Sample Run

```
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
```

Sentinel-Controlled Repetition
（边界值控制的循环）

▶ *A new class-average problem:* *Develop a program that processes grades for an* **arbitrary number** *of students and output the average grade.*

▶ **Analysis:** In the earlier problem, the number of students was known in advance, but here how can the program determine when to stop the input of grades?

Sentinel-Controlled Repetition

We can use **a special value** called a sentinel 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**

Pseudo Code

1   Initialize total to zero   *total* stores the sum of grades
2   Initialize counter to zero   *counter* stores the number grades

3

4   Prompt the user to enter the first grade   Try to take an input
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   If no sentinel value seen,
10   Prompt the user to enter the next grade   repeat the process
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   Compute and print average
16   else   (avoid division by 0)
17   Print "No grades were entered"

```java
// Sentinel-controlled repetition: Class-average problem
import java.util.Scanner;
public class ClassAverage2 {
  public static void main(String[] args) {

    // create Scanner to obtain input from command window
    Scanner input = new Scanner(System.in);

    int total; // sum of grades
    int gradeCounter; // number of grades entered
    int grade; // grade value
    double average; // number with decimal point for average

    // initialization phase
    total = 0; // initialize total
    gradeCounter = 0; // initialize loop counter

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

Sentinel value

```java
        // loop until sentinel value read from user
        while(grade != -1) {

            total = total + grade; // add grade to total
            gradeCounter = gradeCounter + 1; // increment counter
            // prompt for input and read next grade from user
            System.out.print("Enter grade or -1 to quit: ");
            grade = input.nextInt();

        } // end while

        // termination phase
        if(gradeCounter != 0) { // if user entered at least one grade

            // calculate average of all grades entered
            average = (double) total / gradeCounter;
            // display total and average (with two digits of precision)
            System.out.printf("\nTotal of the %d grades entered is
%d\n",  gradeCounter, total);
            System.out.printf("Class average is %.2f\n", average);

        } else { // no grades were entered, output appropriate message

            System.out.println("No grades were entered");

        } // end if

    } // end main
} // end class ClassAverage2
```

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

int gradeCounter;   The unary cast operator creates a temporary

double average;     floating-point copy of its operand
```
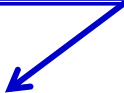
- Cast operator performs explicit conversion (or type cast). It has a higher precedence than the binary arithmetic operators (e.g., /).

- The value stored in the operand is unchanged (e.g., `total`'s value is not changed, `total`'s type is also not changed).

Type Promotion (类型提升)

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

int gradeCounter;

double average;     Type promotion from int to double
```

- Java evaluates only arithmetic expressions in which the operands' types are identical.

- In the above expression, the `int` value of `gradeCounter` will be **implicitly promoted** (widening) to a `double` value for computation.

Why is it called promotion? Why is it implicit?

Promotion Rules

Besides arguments passed to methods, the rules also apply to expressions containing values of two or more primitive types

```
2 * 2.0 becomes 4.0

int x = 2;
double y = x * 2.0;
// is x 2.0 or 2 now?
```

Answer: x is still of `int` type, the expression uses a temporary copy of x's value for promotion

Promotion Rules

Specify which conversions are allowed (which conversions can be performed without losing data)

Type	Valid promotions
double	None
float	double
long	float or double
int	long, float or double
char	int, long, float or double
short	int, long, float or double (but not char)
byte	short, int, long, float or double (but not char)
boolean	None (boolean values are not considered to be numbers in Java)

Variables: Declaration, Assignment, and Usage

▸ A variable must be declared before it can be assigned a value

```java
public static void main(String[] args) {
    int a = 3;
    b = a + 4;
}
```

✗ b must be declared before assigned a value

Variables: Declaration, Assignment, and Usage

▸ A variable must be declared before it can be assigned a value
▸ A variable declared in a method must be assigned a value before it can be used.

```
public static void main(String[] args) {

    int a;

    System.out.println(a);

}          ✗       a must be initialized before being used
```

Variables: Declaration, Assignment, and Usage

- A variable must be declared before it can be assigned a value
- A variable declared in a method must be assigned a value before it can be used.
- A variable can be declared only once inside its scope (more on this later)

```java
public static void main(String[] args) {
    int a = 3;
    int a = 5;   ✗
}
```

a cannot be defined twice because the
first a has a method-level scope

Block Scope (块作用域)

▸ A variable declared inside a pair of braces "{" and "}" in a method has a scope within the braces only

```java
// generates a random number in [0, 1)
double a = Math.random();
System.out.println(a);
if(a > 0.5) {
  double b = 2 * a;
}
System.out.println(b);
```
✗ b can be used only in the `if` block

Block Scope (块作用域)

▸ Due to the rule of variable scope, we often define counters before repetition statements

```java
int counter = 0;
while(counter < 10) {
    // do something and increase counter
    // ...
    counter = counter + 1;
}
System.out.printf("repeated %d times\n", counter);
```

Recall case study I

```
int total; // sum of grades entered by user
int gradeCounter; // number of the grade to be entered next
int grade; // grade value entered by user
int average; // average of grades

// initialization phase
total = 0; // initialize total
gradeCounter = 1; // initialize loop counter

// processing phase
while(gradeCounter <= 10) { // loop 10 times
  System.out.print("Enter grade: "); // prompt
  grade = input.nextInt(); // input next grade
  total = total + grade; // add grade to total
  gradeCounter = gradeCounter + 1; // increment counter by 1
} // end while

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

// display total and average of grades
System.out.printf("\nTotal of all 10 grades is %d\n", total);
System.out.printf("Class average is %d\n", average);
```

Can grade be declared inside of the while loop?

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;`

    61

Compound Assignment Operators
（组合赋值操作符）

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

Recall case study I

```
int total; // sum of grades entered by user
int gradeCounter; // number of the grade to be entered next
int grade; // grade value entered by user
int average; // average of grades

// initialization phase
total = 0; // initialize total
gradeCounter = 1; // initialize loop counter

// processing phase
while(gradeCounter <= 10) { // loop 10 times
    System.out.print("Enter grade: "); // prompt
    grade = input.nextInt(); // input next grade
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1; // increment counter by 1
} // end while

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

// display total and average of grades
System.out.printf("\nTotal of all 10 grades is %d\n", total);
System.out.printf("Class average is %d\n", average);
```

Where can we apply the compound assignment operators?

total += grade;

gradeCounter += 1;

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 placed before a variable is called prefix increment or prefix decrement operator (前缀自增自减操作符).

▸ An increment or decrement operator placed after a variable is called postfix increment or postfix decrement operator (后缀自增自减操作符).

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

Preincrementing/Predecrementing
（前缀自增/自减）

▸ 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; // a, b gets the value 7
```

Postincrementing/Postdecrementing
（后缀自增/自减）

‣ 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, a gets the value 7
```

Note the Difference

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

```
int b = a++;

Equivalent to:

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

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

```
int b = ++a;

Equivalent to:

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

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

Recall case study

```
int total; // sum of grades entered by user
int gradeCounter; // number of the grade to be entered next
int grade; // grade value entered by user
int average; // average of grades

// initialization phase
total = 0; // initialize total
gradeCounter = 1; // initialize loop counter

// processing phase
while(gradeCounter <= 10) { // loop 10 times
    System.out.print("Enter grade: "); // prompt
    grade = input.nextInt(); // input next grade
    total = total + grade; // add grade to total
    gradeCounter = gradeCounter + 1; // increment counter by 1
} // end while

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

// display total and average of grades
System.out.printf("\nTotal of all 10 grades is %d\n", total);
System.out.printf("Class average is %d\n", average);
```

Where can we apply the increment operators?

gradeCounter++;
or
++gradeCounter;

The Operators Introduced So Far
(Take a close look by yourself)

Precedence (downward arrow)

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

Associativity is not relevant for some operators. For example, `x <= y <= z` and `x++--` are invalid expressions in Java.