Chapter 3
Control Statements (Part Ⅰ)

Yepang LIU
liuyp1@sustech.edu.cn

Objectives

- To learn and use basic problem-solving techniques
- To develop algorithms using pseudo codes (伪代码)
- To use `if` and `if…else` selection statements (选择/条件语句)
- To use `while` repetition statement (循环语句)

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 (算法)

- Any computing problem can be solved by executing a series of actions in a specific order
- An algorithm describes 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 a typical white-collar worker: (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 (伪代码)

- Pseudocode is an informal language for developing algorithms
- Similar to everyday English
- Helps you "think out" a program

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

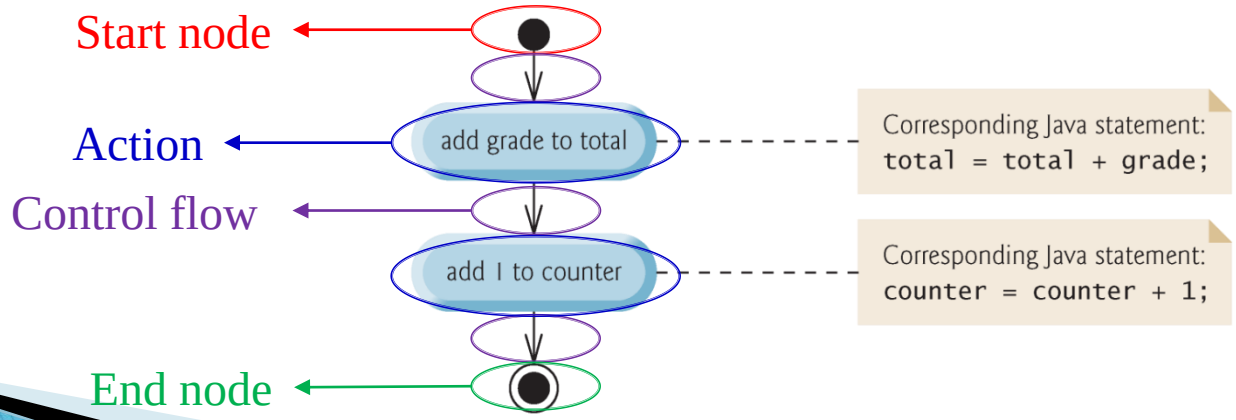
- Sequential execution (顺序执行): normally, statements in a program are executed one after the other in the order in which they are written.

- Transfer of control (控制跳转): 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 (顺序，选择，循环)

Sequence Structure (顺序)

- 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 (选择)

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

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

`if` Single-Selection Statement

- Pseudocode:

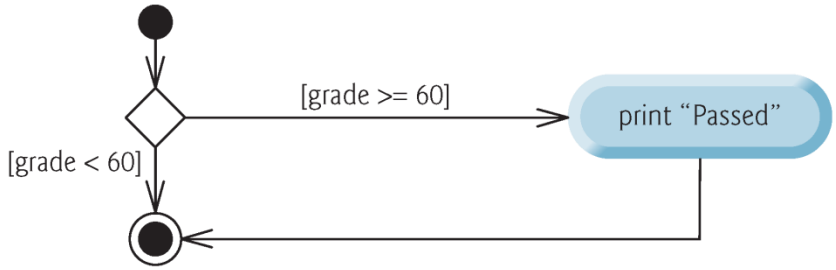
  If student's grade is greater than or equal to 60
  Print "Passed"

  Only one choice, so it is called "single-selection"

- Java code:

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

`if` Single-Selection Statement



Activity diagram in UML

(Unified Modeling Language

统一建模语言)

- 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"

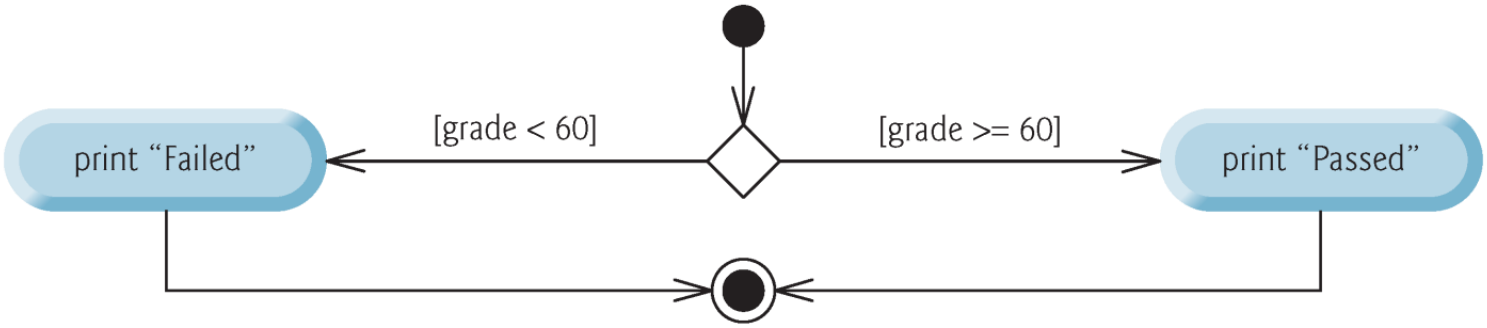
  Two choices, so it is called "double-selection"

- 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 ( studentGrade >= 60 )
    result = “Passed”;
else
    result = “Failed”;
```

A More Complex Example

- Pseudocode:

Nested `if…else` statements (嵌套)

```
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:

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

If-else Matching Rule

- Extra spaces are irrelevant in Java (only for formatting). The compiler actually interprets the statement as

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

If-else Matching Rule

What if you really want this effect?

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

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

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

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

Empty Statement

- 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

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

```
if (x == 1); {
   System.out.print("I always execute");
}
```

The two programs are valid, although not quite meaningful.

`while` Repetition Statement

- Repeat an action while a condition remains true
- Pseudocode

  While there are more items on my shopping list
  Purchase next item and cross it off my list

- The repetition statement's body may be a single statement or a block. Eventually, the condition should become false, and the repetition terminates, and the first statement after the repetition statement executes (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;
}
// 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

```
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

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

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

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
~~3~~
~~9~~
27

Condition true
Enter loop body

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

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

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~~
243

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~~
243

Condition false
Exit 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;
}
// other statements
```

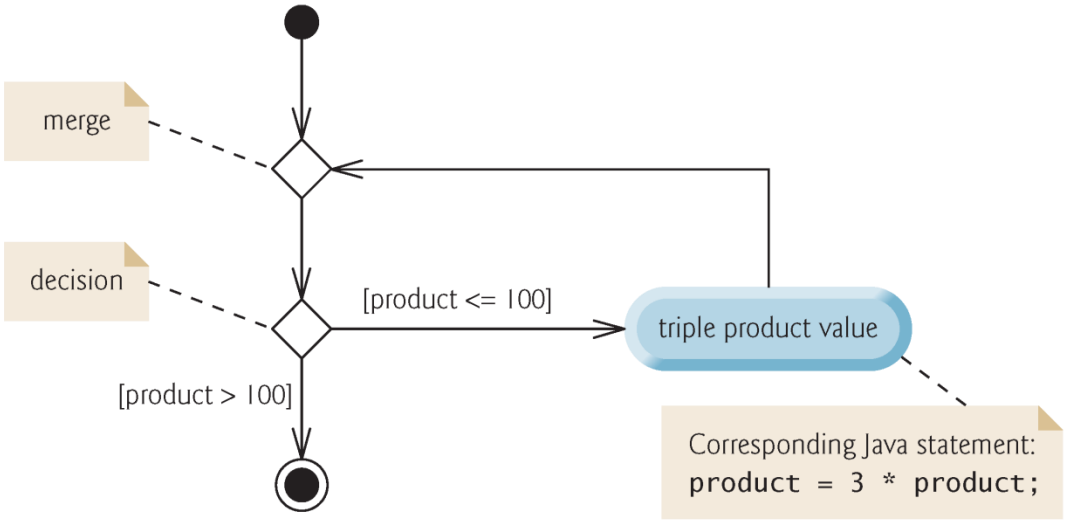
product value
~~3~~
~~9~~
~~27~~
~~81~~
243

The first statement after the while statement will be executed

`while` Statement Activity Diagram

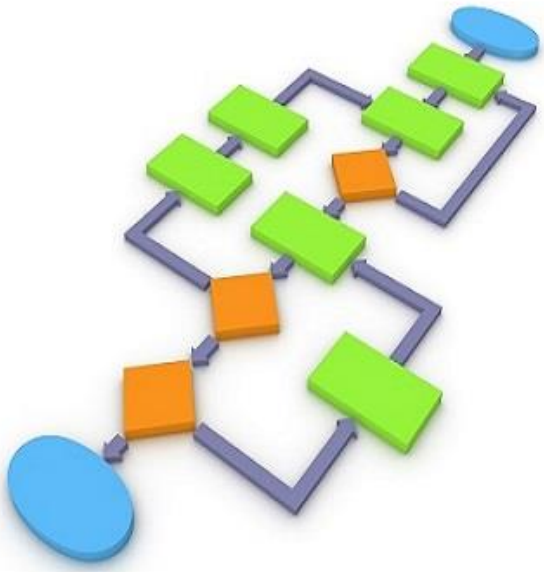
- The UML represents both the merge symbol and the decision symbol as diamonds

- The merge symbol joins two flows of activity into one



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
```



Formulating Algorithms

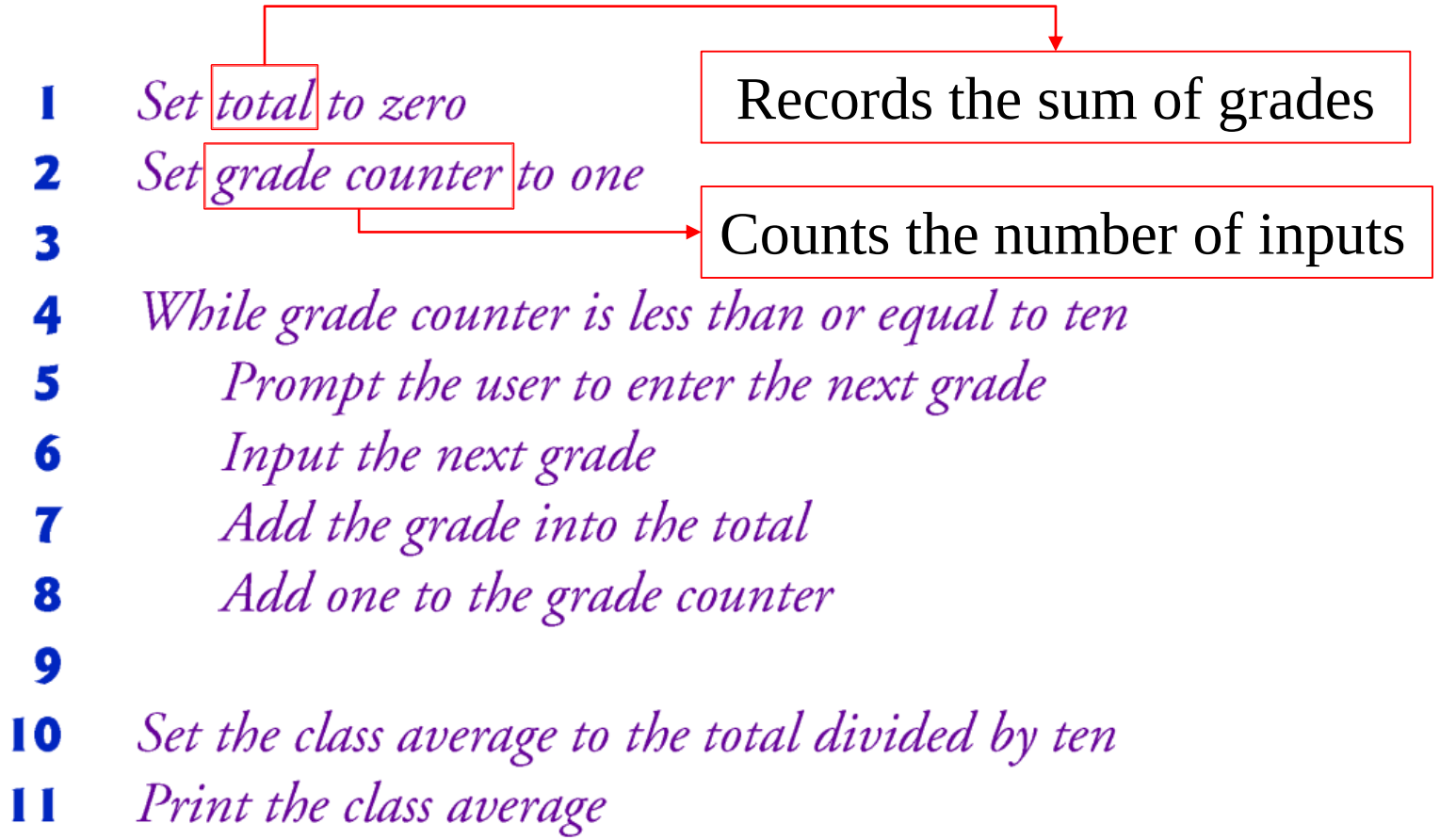
Counter-Controlled Repetition (计数器控制的循环)

- ***Class-Average 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.

The Pseudo Code



Translate to Java Code

```
// 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

```
    // 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);

    // close Scanner to avoid resource leak
    input.close();
  } // 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**

Sentinel-Controlled Repetition

- Sentinel-controlled repetition is often called indefinite repetition (不确定循环) 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

One of the left items? Of course not…

Pseudo Code

```
1   Initialize total to zero
2   Initialize counter to zero
3
4   Prompt the user to enter the first grade
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
10      Prompt the user to enter the next grade
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
16  else
17      Print "No grades were entered"
```

total stores the sum of grades
counter stores the number grades

Try to take an input

If no sentinel value seen, repeat the process

Compute and print average (avoid division by 0)

Java Code

```
// 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

```
// 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
```

```
    // close Scanner to avoid resource leak
    input.close();
  } // 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;
int gradeCounter;
double average;
```

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

The unary cast operator creates a temporary 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;
int gradeCounter;
double average;
```

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

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 **promoted** (implicit conversion) to a `double` value for computation.

Why it is called promotion? double is more expressive

The Scope (作用域) 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's declaration must appear before the variable is used in that method

https://www.geeksforgeeks.org/variable-scope-in-java/

Is the code correct?

```
public class Scope {

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

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

}
```

a is a local variable in `main`, cannot be used outside `main`

Is the code correct?

```
public class Scope {

  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

Is the code correct?

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

b must be defined before use

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

```
// 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

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

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;`

Compound Assignment Operators

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

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; // 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
```

Note the Difference

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

```
int b = ++a;

Equivalent to:

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

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

```
int b = a++;

Equivalent to:

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

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

Please practice each of the operators by yourself ☺