Control Statements: Part I

OBJECTIVES

In this chapter you'll learn:

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if...else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a program repeatedly.
- To use counter-controlled repetition and sentinel-controlled repetition.
- To use the compound assignment, increment and decrement operators.
- The portability of primitive data types.

4.1 Introduction

- Before writing a program to solve a problem, 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.
- This chapter introduces
  - The `if`, `if`…`else` and `while` statements
  - Compound assignment, increment and decrement operators
  - Portability of Java's primitive types

4.2 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" followed by one executive for getting out of bed and going to work:
  - (1) Get out of bed; (2) take off pajamas; (3) take a shower; (4) get dressed; (5) eat breakfast; (6) carpool to work.
- Suppose that the same steps are performed in a slightly different order:
  - (1) Get out of bed; (2) take off pajamas; (3) get dressed; (4) take a shower; (5) eat breakfast; (6) carpool to work.
- Specifying the order in which statements (actions) execute in a program is called program control.

4.3 Pseudocode

- Pseudocode is an informal language that helps you develop algorithms without having to worry about the strict details of Java language syntax.
- Particularly useful for developing algorithms that will be converted to structured portions of Java programs.
- Similar to everyday English.
- Helps you "think out" a program before attempting to write it in a programming language, such as Java.
- You can type pseudocode conveniently, using any text-editor program.
- Carefully prepared pseudocode can easily be converted to a corresponding Java program.
- Pseudocode normally describes only statements representing the actions that occur after you convert a program from pseudocode to Java and the program is run on a computer.
  - e.g., input, output or calculations.

4.4 Control Structures

- Sequential execution: Statements in a program execute one after the other in the order in which they are written.
- Transfer of control: Various Java statements, enable you to specify that the next statement to execute is not necessarily the next one in sequence.
- Bohm and Jacopini
  - Demonstrated that programs could be written *without any* `goto` *statements.*
  - All programs can be written in terms of only three control structures—the sequence structure, the selection structure and the repetition structure.
- When we introduce Java's control structure implementations, we'll refer to them in the terminology of the *Java Language Specification as "control statements."*

4.4 Control Structures (Cont.)

- Sequence structure
  - Built into Java.
  - Unless directed otherwise, the computer executes Java statements one after the other in the order in which they're written.
  - The activity diagram in Fig. 4.1 illustrates a typical sequence structure in which two calculations are performed in order.
  - Java lets you have as many actions as you want in a sequence structure.
  - Anywhere a single action may be placed, we may place several actions in sequence.

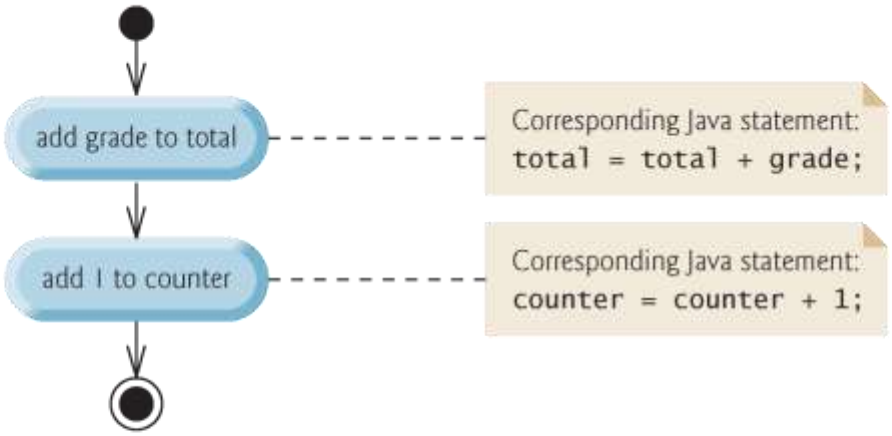

Fig. 4.1 | Sequence structure activity diagram.

4.4 Control Structures (Cont.)

- UML activity diagram
- Models the workflow (also called the activity) of a portion of a software system.
- May include a portion of an algorithm, like the sequence structure in Fig. 4.1.
- Composed of symbols
  - action-state symbols (rectangles with their left and right sides replaced with outward arcs)
  - diamonds
  - small circles
- Symbols connected by transition arrows, which represent the flow of the activity—the order in which the actions should occur.
- Help you develop and represent algorithms.
- Clearly show how control structures operate.

4.4 Control Structures (Cont.)

- Sequence structure activity diagram in Fig. 4.1.
- Two action states that represent actions to perform.
- Each contains an action expression that specifies a particular action to perform.
- Arrows represent transitions (order in which the actions represented by the action states occur).
- Solid circle at the top represents the initial state—the beginning of the workflow before the program performs the modeled actions.
- Solid circle surrounded by a hollow circle at the bottom represents the final state—the end of the workflow after the program performs its actions.

4.4 Control Structures (Cont.)

- UML notes
  - Like comments in Java.
  - Rectangles with the upper-right corners folded over.
  - Dotted line connects each note with the element it describes.
  - Activity diagrams normally do not show the Java code that implements the activity. We do this here to illustrate how the diagram relates to Java code.
- More information on the UML
  - see our optional case study (Chapters 12–13)
  - visit `www.uml.org`

4.4 Control Structures (Cont.)

- Three types of selection statements.
- `if` statement:
  - Performs an action, if a condition is true; skips it, if false.
  - Single-selection statement—selects or ignores a single action (or group of actions).
- `if...else` statement:
  - Performs an action if a condition is true and performs a different action if the condition is false.
  - Double-selection statement—selects between two different actions (or groups of actions).
- `switch` statement
  - Performs one of several actions, based on the value of an expression.
  - Multiple-selection statement—selects among many different actions (or groups of actions).

4.4 Control Structures (Cont.)

- Three repetition statements (also called looping statements)
  - Perform statements repeatedly while a loop-continuation condition remains true.
- `while` and `for` statements perform the action(s) in their bodies zero or more times
  - if the loop-continuation condition is initially false, the body will not execute.
- The `do`…`while` statement performs the action(s) in its body one or more times.
- `if`, `else`, `switch`, `while`, `do` and `for` are keywords.
  - Appendix C: Complete list of Java keywords.

4.4 Control Structures (Cont.)

- Every program is formed by combining the sequence statement, selection statements (three types) and repetition statements (three types) as appropriate for the algorithm the program implements.
- Can model each control statement as an activity diagram.
  - Initial state and a final state represent a control statement's entry point and exit point, respectively.
  - Single-entry/single-exit control statements
  - Control-statement stacking—connect the exit point of one to the entry point of the next.
  - Control-statement nesting—a control statement inside another.

4.5 if Single-Selection Statement

- Pseudocode

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

- If the condition is false, the Print statement is ignored, and the next pseudocode statement in order is performed.
- Indentation
  - Optional, but recommended
  - Emphasizes the inherent structure of structured programs
- The preceding pseudocode *If* in Java:

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

- Corresponds closely to the pseudocode.

4.5 `if` Single-Selection Statement (Cont.)

- Figure 4.2 `if` statement UML activity diagram.
- Diamond, or decision symbol, indicates that a decision is to be made.
- Workflow continues along a path determined by the symbol's guard conditions, which can be true or false.
- Each transition arrow emerging from a decision symbol has a guard condition (in square brackets next to the arrow).
- If a guard condition is true, the workflow enters the action state to which the transition arrow points.

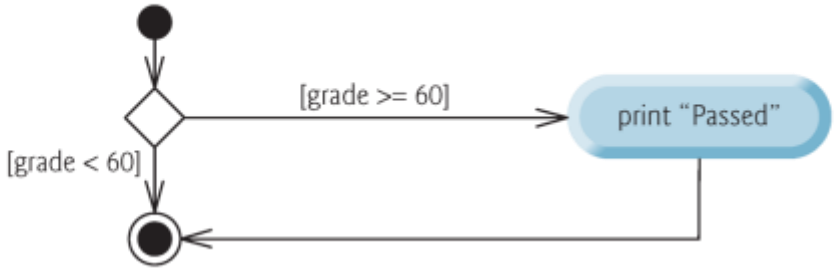

Fig. 4.2 | `if` single-selection statement UML activity diagram.

4.6 `if...else` Double-Selection Statement

- `if...else` double-selection statement—specify an action to perform when the condition is true and a different action when the condition is false.
- Pseudocode

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

- The preceding *If...Else pseudocode statement in Java:*

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

- Note that the body of the `else` is also indented.

Good Programming Practice 4.1

Indent both body statements of an `if...else` *statement. Many IDEs do this for you.*

Good Programming Practice 4.2

If there are several levels of indentation, each level should be indented the same additional amount of space.

4.6 `if...else` Double-Selection Statement (Cont.)

- Figure 4.3 illustrates the flow of control in the `if...else` statement.
- The symbols in the UML activity diagram (besides the initial state, transition arrows and final state) represent action states and decisions.

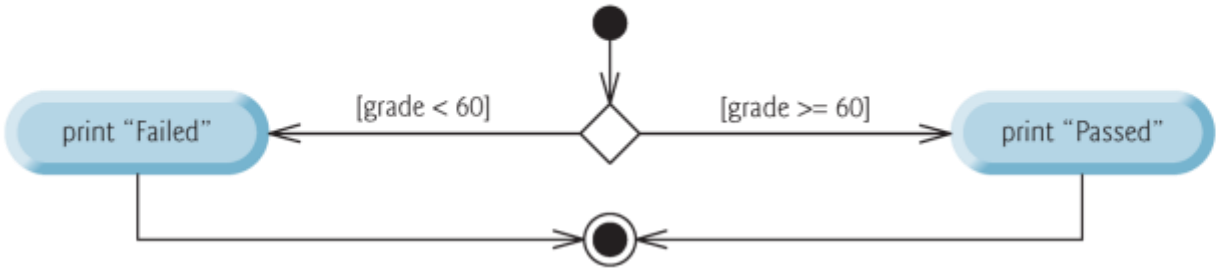

Fig. 4.3 | `if...else` double-selection statement UML activity diagram.

4.6 `if...else` Double-Selection Statement (Cont.)

- Conditional operator (`?:`)—shorthand `if...else`.
- Ternary operator (takes three operands)
- Operands and `?:` form a conditional expression
- Operand to the left of the `?` is a boolean expression—evaluates to a `boolean` value (`true` or `false`)
- Second operand (between the `?` and `:`) is the value if the `boolean` expression is `true`
- Third operand (to the right of the `:`) is the value if the `boolean` expression evaluates to `false`.

4.6 if...else Double-Selection Statement (Cont.)

- Example:

```
System.out.println(
   studentGrade >= 60 ? "Passed" : "Failed" );
```

- Evaluates to the string `"Passed"` if the `boolean` expression `studentGrade >= 60` is true and to the string `"Failed"` if it is false.

Good Programming Practice 4.3

Conditional expressions are more difficult to read than `if...else` *statements and should be used to replace only simple* `if...else` *statements that choose between two values.*

4.6 `if...else` Double-Selection Statement (Cont.)

- Can test multiple cases by placing `if...else` statements inside other `if...else` statements to create nested *if...else statements*.
- 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"
```

4.6 if...else Double-Selection Statement (Cont.)

- This pseudocode may be written in Java 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 `studentGrade` >= 90, the first four conditions will be true, but only the statement in the `if` part of the first `if`...`else` statement will execute. After that, the `else` part of the "outermost" `if`...`else` statement is skipped.

4.6 if...else Double-Selection Statement (Cont.)

- 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" );
```

- The two forms are identical except for the spacing and indentation, which the compiler ignores.

4.6 if...else Double-Selection Statement (Cont.)

- The Java compiler always associates an `else` with the immediately preceding `if` unless told to do otherwise by the placement of braces (`{` and `}`).
- Referred to as the dangling-else problem.
- The following code is not what it appears:

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

- Beware! This nested `if...else` statement does not execute as it appears. 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" );
```

4.6 if...else Double-Selection Statement (Cont.)

- To force the nested `if...else` statement to execute as it was originally intended, we must write it as follows:

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

- The braces indicate that the second `if` is in the body of the first and that the `else` is associated with the *first* `if`.
- Exercises 4.27–4.28 investigate the dangling-`else` problem further.

4.6 if...else Double-Selection Statement (Cont.)

- The `if` statement normally expects only one statement in its body.
- To include several statements in the body of an `if` (or the body of an `else` for an `if...else` statement), enclose the statements in braces.
- Statements contained in a pair of braces form a block.
- A block can be placed anywhere that a single statement can be placed.
- Example: A block in the `else` part of an `if...else` statement:

```
if ( grade >= 60 )
   System.out.println("Passed");
else
{
   System.out.println("Failed");
   System.out.println("You must take this course again.");
}
```

4.6 if...else Double-Selection Statement (Cont.)

- 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.

4.6 `if...else` Double-Selection Statement (Cont.)

- 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.

Common Programming Error 4.1

Placing a semicolon after the condition in an `if` *or* `if...else` *statement leads to a logic error in single-selection* `if` *statements and a syntax error in double-selection* `if...else` *statements (when the* `if`*-part contains an actual body statement).*

4.7 while Repetition Statement

- Repetition statement—repeats 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 will become false. At this point, the repetition terminates, and the first statement after the repetition statement executes.

4.7 while Repetition Statement (Cont.)

- Example of Java's while repetition statement: find the first power of 3 larger than 100. Assume `int` variable `product` is initialized to `3`.

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

- Each iteration multiplies `product` by 3, so `product` takes on the values 9, 27, 81 and 243 successively.
- When variable `product` becomes 243, the `while`-statement condition—`product <= 100`—becomes false.
- Repetition terminates. The final value of `product` is 243.
- Program execution continues with the next statement after the `while` statement.

Common Programming Error 4.2

Not providing in the body of a `while` *statement an action that eventually causes the condition in the* `while` *to become false normally results in a logic error called an* ***infinite loop*** *(the loop never terminates).*

4.7 `while` Repetition Statement (Cont.)

- The UML activity diagram in Fig. 4.4 illustrates the flow of control in the preceding `while` statement.
- The UML represents both the merge symbol and the decision symbol as diamonds.
- The merge symbol joins two flows of activity into one.

4.7 while Repetition Statement (Cont.)

- The decision and merge symbols can be distinguished by the number of "incoming" and "outgoing" transition arrows.
  - A decision symbol has one transition arrow pointing to the diamond and two or more pointing out from it to indicate possible transitions from that point. Each transition arrow pointing out of a decision symbol has a guard condition next to it.
  - A merge symbol has two or more transition arrows pointing to the diamond and only one pointing from the diamond, to indicate multiple activity flows merging to continue the activity. None of the transition arrows associated with a merge symbol has a guard condition.

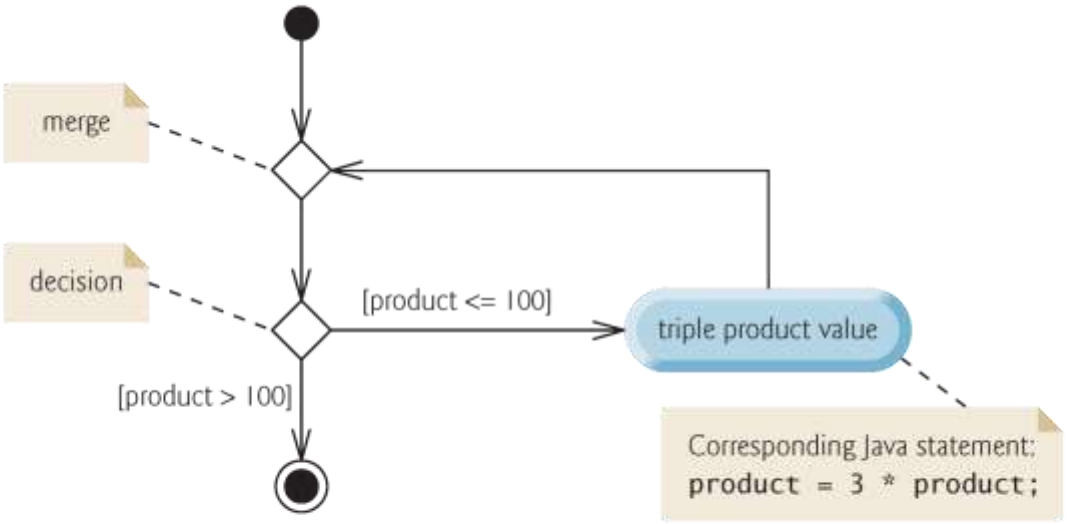

Fig. 4.4 | `while` repetition statement UML activity diagram.

4.8 Formulating Algorithms: Counter-Controlled Repetition

- *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.*
- The class average is equal to the sum of the grades divided by the number of students.
- 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.
- 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.
- Counter-controlled repetition is often called definite repetition, because the number of repetitions is known before the loop begins executing.

Software Engineering Observation 4.1

Experience has shown that the most difficult part of solving a problem on a computer is developing the algorithm for the solution. Once a correct algorithm has been specified, producing a working Java program from the algorithm is usually straightforward.

4.8 Formulating Algorithms: Counter-Controlled Repetition (Cont.)

- A total is a variable used to accumulate the sum of several values.
- A counter is a variable used to count.
- Variables used to store totals are normally initialized to zero before being used in a program.

4.8 Formulating Algorithms: Counter-Controlled Repetition (Cont.)

- 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.
- A local variable cannot be accessed outside the method in which it's declared.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition

- *Develop a class-averaging program that processes grades for an arbitrary number of students each time it is run.*
- Sentinel-controlled repetition is often called indefinite repetition because the number of repetitions is not known before the loop begins executing.
- A special value called a sentinel value (also called a signal value, a dummy value or a flag value) can be used to indicate "end of data entry."
- A sentinel value must be chosen that cannot be confused with an acceptable input value.

4.9 Formulating Algorithms: Sentinel-Controlled Repetition (Cont.)

- Program logic for sentinel-controlled repetition
  - Reads the first value before reaching the `while`.
  - This value determines whether the program's flow of control should enter the body of the `while`. If the condition of the `while` is false, the user entered the sentinel value, so the body of the `while` does not execute (i.e., no grades were entered).
  - If the condition is true, the body begins execution and processes the input.
  - Then the loop body inputs the next value from the user before the end of the loop.