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.”

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).

Three repetition statements (also called looping statements)

- **while** and **for** statements perform the action(s) in their bodies zero or more times.
  - if, else, switch, while, do and for are keywords.
- **do...while** statement performs the action(s) in its body one or more times.

Appendix C: Complete list of Java keywords.
4.5 if Statement

- **Forms:**
  - if (condition) statement // cond is in parenthesis
  - if (condition) statement-1 else statement-2
  - Multiple statement are enclosed in braces ({}]
  - Make sure if components are properly matched!
  - Expression-1 ? Expression-2 : expression-3;

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.

4.6 if...else Double-Selection Statement

- **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.
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.
- Example:
  ```java
  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.

Conditional expression
- Conditional expression returns a value. Hence can be used in an expression or assignment
  ```java
  int number, x = 200;
  number = x > 100 ? 20 : 50;
  ```
  Is equivalent to
  ```java
  if (x>100)
      number = 20;
  else
      number = 50;
  ```
  Compact code:
  ```java
  System.out.println("your grade is: " + score > 60 ? "pass." : "fail.");
  ```
This pseudocode may be written in Java as

```java
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.

Most Java programmers prefer to write the preceding nested `if...else` statement as

```java
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.

### 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:

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

Common errors to avoid

- Using a = instead of == to compare primitive values
- Using == instead of the equals method to compare String objects
- Forgetting to enclose an if statement’s boolean expression in parenthesis
- writing semicolon at the end of an if clause (actually after the condition)
- Forgetting to express multiple conditionally executed statements in braces
- Omitting the trailing else in an if-else-if statement
- Not writing complete boolean expressions on both sides of the && and || operator (e.g., x > 0 && x < 10)

if...else Double-Selection Statement

- Syntax errors (e.g., when one brace in a block is left out) 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.

Looping Statements

- while statement
- for repetition statement
- do...while repetition statement
- switch multiple-selection statement
- break statement
- continue statement
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.
  
  ```java
  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.
- Beware of infinite loops

5.3 for Repetition Statement (Cont.)

- When the for statement begins executing, the control variable is declared and initialized.
- Next, the program checks the loop-continuation condition, which is between the two required semicolons.
- If the condition initially is true, the body statement executes.
- After executing the loop’s body, the program increments the control variable in the increment expression, which appears to the right of the second semicolon.
- Then the loop-continuation test is performed again to determine whether the program should continue with the next iteration of the loop.
- A common logic error with counter-controlled repetition is an off-by-one error.

5.3 for Repetition Statement (Cont.)

The general format of the for statement is

```java
for (initialization; loopContinuationCondition; increment) statement
```

- the initialization expression names the loop’s control variable and optionally provides its initial value
- loopContinuationCondition determines whether the loop should continue executing
- increment modifies the control variable’s value (possibly an increment or decrement), so that the loop-continuation condition eventually becomes false.
- The two semicolons in the for header are required.
In most cases, the for statement can be represented with an equivalent while statement as follows:

```
while (loopContinuationCondition) {
    statement
    increment;
}
```

Typically, for statements are used for counter-controlled repetition and while statements for sentinel-controlled repetition.

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

A variable’s scope defines where it can be used in a program.

- A local variable can be used only in the method that declares it and only from the point of declaration through the end of the method.

All three expressions in a for header are optional.
- If the loopContinuationCondition is omitted, the condition is always true, thus creating an infinite loop.
- You might omit the initialization expression if the program initializes the control variable before the loop.
- You might omit the increment if the program calculates it with statements in the loop’s body or if no increment is needed.

The increment expression in a for acts as if it were a standalone statement at the end of the for’s body, so

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

are equivalent increment expressions in a for statement.

The initialization, loop-continuation condition and increment can contain arithmetic expressions.

For example, assume that \( x = 2 \) and \( y = 10 \). If \( x \) and \( y \) are not modified in the body of the loop, the statement

```
for (int j = x; j <= 4 * x * y; j += y / x)
```

is equivalent to the statement

```
for (int j = 2; j <= 80; j += 5)
```

The increment of a for statement may be negative, in which case it’s a decrement, and the loop counts downward.

Although the value of the control variable can be changed in the body of a for loop, avoid doing so because this practice can lead to subtle errors.
5.4 Examples Using the for Statement

- a) Vary the control variable from 1 to 100 in increments of 1.
  ```java
  for ( int i = 1; i <= 100; i++ )
  ```

- b) Vary the control variable from 100 to 1 in decrements of 1.
  ```java
  for ( int i = 100; i >= 1; i-- )
  ```

- c) Vary the control variable from 7 to 77 in increments of 7.
  ```java
  for ( int i = 7; i <= 77; i += 7 )
  ```

- d) Vary the control variable from 20 to 2 in decrements of 2.
  ```java
  for ( int i = 20; i >= 2; i -= 2 )
  ```

- e) Vary the control variable over the values 2, 5, 8, 11, 14, 17, 20.
  ```java
  for ( int i = 2; i <= 20; i += 3 )
  ```

- f) Vary the control variable over the values 99, 88, 77, 66, 55, 44, 33, 22, 11, 0.
  ```java
  for ( int i = 99; i >= 0; i -= 11 )
  ```

Note the use of >= or <= instead of ==

5.5 do...while Repetition Statement

- The do...while repetition statement is similar to the while statement.
- In the while, the program tests the loop-continuation condition at the beginning of the loop, before executing the loop’s body; if the condition is false, the body never executes.
- The do...while statement tests the loop-continuation condition after executing the loop’s body; therefore, the body always executes at least once.
- When a do...while statement terminates, execution continues with the next statement in sequence.
5.6 switch Multiple-Selection Statement

- switch multiple-selection statement performs different actions based on the possible values of a constant integral expression of type byte, short, int or char.

- You can also use it for enumerated or enum types as you will see later

- Strings have been added in Java SE 7
Fig. 5.9 Gradebook class uses switch to count letter grades (Part 5 of 6)

```java
    System.out.println("E: ", eCount); // display number of E grades
    System.out.println("D: ", dCount); // display number of D grades
    System.out.println("C: ", cCount); // display number of C grades
    System.out.println("B: ", bCount); // display number of B grades
    System.out.println("A: ", aCount); // display number of A grades

    System.out.println("No grades were entered.");
  } // end method displayGradeCount
}
} // end Class Gradebook
```

Fig. 5.9 Gradebook class uses switch to count letter grades (Part 6 of 6)

```java
// create Gradebook object and display grades and display grade report.
Gradebook gb = new Gradebook();
```

```java
// Create Gradebook object, input grades and display grade report.
public class GradebookTest

// make sure there are at least one 70
if (gradeCounter > 1
    System.out.println("Class average is 92.75", average =
```

```java
// String[] args) {
    public class GradebookTest

    // Create Gradebook object and display grades and display grade report.
    public static void main(String[] args)
```

```java
// Create Gradebook object, input grades and display grade report.
Gradebook gb = new Gradebook();
```

```java
// Create Gradebook object, input grades and display grade report.
Gradebook gb = new Gradebook();
```
switch does not provide a mechanism for testing ranges of values—every value must be listed in a separate case label.

Note that each case can have multiple statements.

switch differs from other control statements in that it does not require braces around multiple statements in a case.

Without break, the statements for a matching case and subsequent cases execute until a break or the end of the switch is encountered. This is called “falling through.”

If no match occurs between the controlling expression’s value and a case label, the default case executes.

If no match occurs and there is no default case, program control simply continues with the first statement after the switch.

It also helps you to see that none of the cases were satisfied which may throw some insight into debugging.
5.6 switch Multiple-Selection Statement (Cont.)

- When using the `switch` statement, remember that each `case` must contain a constant integral expression.
- An integer constant is simply an integer value.
- In addition, you can use `character constants`—specific characters in single quotes, such as `'A'`, `'?'`, or `$`—which represent the integer values of characters.
- The expression in each `case` can also be a `constant variable`—a variable that contains a value which does not change for the entire program. Such a variable is declared with the keyword `final`.
- Java has a feature called enumerations. Enumeration constants can also be used in `case` labels.

5.7 break and continue Statements

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

Example:

```
public class BreakTest {
    public static void main(String[] args) {
        int count = 1;
        while (count < 10) {
            if (count == 5) {
                System.out.print("Break out of loop at count = ");
                break;
            }
            // end if
            System.out.println("Loop count = "+count);
            // end for
            count++;
            // end while
        }
    }
}
```

Fig. 13 Break statement exiting a for statement.
5.7 break and continue Statements  
(Cont.)

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

```java
// Fig. 5.13: ContatinueTest.java
// continue statements terminating an iteration of a for statement.
public class ContinueTest
{
    public static void main(String[] args)
    {
        for (int count = 0; count < 10; count++) // loop 10 times
        {
            if (count  == 5) // if count is 5,
            {
                continue; // skip remaining code in loop
                System.out.print( "nd ", count );
            } // end if
            System.out.print( "lced continue to skip printing 5 ");
        } // end for
        System.out.println( "lasted continue to skip printing 5 ");
    } // end main
    if (3) // end class ContinueTest
}
```

---

**Software Engineering Observation 5.4**

Some programmers feel that break and continue violate structured programming. Since the same effects are achievable with structured programming techniques, these programmers do not use break or continue.

---

**Software Engineering Observation 5.5**

There is a tension between achieving quality software engineering and achieving the best-performing software. Often, one of these goals is achieved at the expense of the other. For all but the most performance-intensive situations, apply the following rule of thumb: First, make your code simple and correct, then make it fast and small, but only if necessary.
5.9 Structured Programming Summary

- Structured programming promotes simplicity.
- Bohm and Jacopini: Only three forms of control are needed to implement an algorithm:
  - Sequence
  - Selection
  - Repetition
- The sequence structure is trivial. Simply list the statements to execute in the order in which they should execute.

Thank You!