4.5 `if` Single-Selection Statement

Programs use selection statements to choose among alternative courses of action. For example, suppose that the passing grade on an exam is 60. The *pseudocode* statement

```
If student’s grade is greater than or equal to 60
Print “Passed”
```

determines whether the *condition* “student’s grade is greater than or equal to 60” is *true*. If so, “Passed” is printed, and the next pseudocode statement in order is “performed.” If the condition is *false*, the *Print* statement is ignored, and the next pseudocode statement in order is performed. The indentation of the second line of this selection statement is optional, but recommended, because it emphasizes the inherent structure of structured programs.

The preceding pseudocode *If* statement may be written in Java as

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

The Java code corresponds closely to the pseudocode. This is one of the properties of pseudocode that makes it such a useful program development tool.

UML Activity Diagram for an `if` Statement

Figure 4.2 illustrates the single-selection `if` statement. This figure contains the most important symbol in an activity diagram—the *diamond*, or **decision symbol**, which indicates that a *decision* is to be made. The workflow continues along a path determined by the symbol's associated **guard conditions**, which can be *true* or *false*. Each transition arrow emerging from a decision symbol has a guard condition (specified 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. In Fig. 4.2, if the grade is greater than or equal to 60, the program prints "Passed," then transitions to the activity's final state. If the grade is less than 60, the program immediately transitions to the final state without displaying a message.

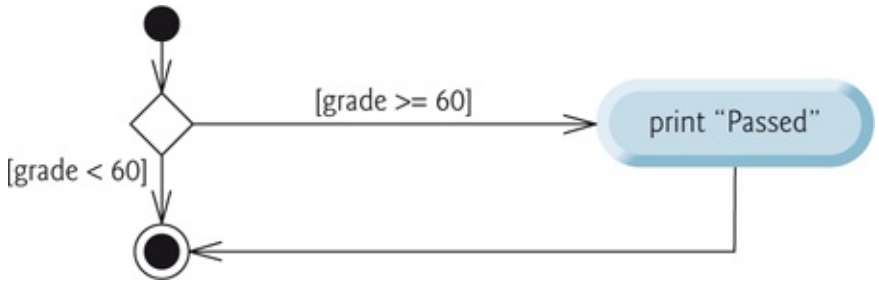

Fig. 4.2

`if` single-selection statement UML activity diagram.

The `if` statement is a single-entry/single-exit control statement. We'll see that the activity diagrams for the remaining control statements also contain initial states, transition arrows, action states that indicate actions to perform, decision symbols (with associated guard conditions) that indicate decisions to be made, and final states.

4.6 `if…else` Double-Selection Statement

The `if` single-selection statement performs an indicated action only when the condition is `true`; otherwise, the action is skipped. The `if…else` **double-selection statement** allows you to specify an action to perform when the condition is *true* and another action when the condition is *false*. For example, the pseudocode statement

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

prints "Passed" if the student's grade is greater than or equal to 60, but prints "Failed" if it's less than 60. In either case, after

printing occurs, the next pseudocode statement in sequence is "performed."

The preceding *If…Else* pseudocode statement can be written in Java as

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

The body of the `else` is also indented. Whatever indentation convention you choose should be applied consistently throughout your programs.

Good Programming Practice 4.1

Indent both body statements of an `if…else` *statement. Most 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.

UML Activity Diagram for an `if…else` Statement

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

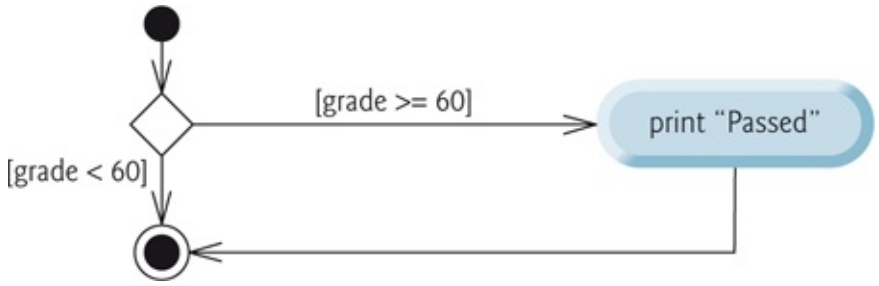

Fig. 4.3

`if…else` double-selection statement UML activity diagram.

Description

4.6.1 Nested `if…else` Statements

A program can test multiple cases by placing `if…else` statements inside other `if…else` statements to create **nested `if…else` statements**. For example, the following pseudocode represents a nested `if…else` that prints `A` for exam grades greater than or equal to 90, `B` for grades 80 to 89, `C` for grades 70 to 79, `D` for grades 60 to 69 and `F` for all other grades:

```
If student’s grade is greater than or equal to 90
      Print “A”
else
      If student’s grade is greater than or equal to
            Print “B”
      else
            If student’s grade is greater than or equ
                  Print “C”
            else
                  If student’s grade is greater than
                        Print “D”
                  else
                        Print “F”
```

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");
            }
         }
      }
   }
```

Error-Prevention Tip 4.1

In a nested `if`…`else` *statement, ensure that you test for all possible cases.*

If variable `studentGrade` is greater than or equal to 90, the first four conditions in the nested `if`…`else` statement will be true, but only the statement in the `if` part of the first `if`…`else` statement will execute. After that statement executes, the `else` part of the "outermost" `if`…`else` statement is skipped. Many 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. The latter form avoids deep indentation of the code to the right. Such indentation often leaves little room on a line of source code, forcing lines to be split.

4.6.2 Dangling-`else` Problem

Throughout the text, we always enclose control statement bodies in braces (`{` and `}`). This avoids a logic error called the "dangling-`else`" problem. We investigate this problem in Exercises 4.27–4.29.

4.6.3 Blocks

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 braces (such as a method's body) form a **block**. A block can be placed anywhere in a method that a single statement can be placed.

The following example includes a block of multiple statements 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 agai
}
```

In this case, if `grade` is less than 60, the program executes *both* statements in the body of the `else` and prints

```
Failed
You must take this course again.
```

Note the braces surrounding the two statements in the `else` clause. These braces are important. Without the braces, the statement

```
System.out.println("You must take this course again."
```

would be outside the body of the `else` part of the `if…else` statement and would execute *regardless* of whether the grade

was less than 60.

Syntax errors (such as a missing brace) 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.

Just as a block can be placed anywhere a single statement can be placed, it's also possible to have an empty statement. Recall from Section 2.8 that 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).*

4.6.4 Conditional Operator (?:)

Java provides the **conditional operator** (**?:**) that can be used in place of simple `if`…`else` statements. This can make your code shorter and clearer. The conditional operator is the only **ternary operator** (i.e., it takes *three* operands). Together, the operands and the `?:` symbol form a **conditional expression.** The first operand (to the left of the `?`) is a **boolean expression** (i.e., a *condition* that evaluates to a `boolean` value—**true** or **false**), the second operand (between the `?` and `:`) is the value of the conditional expression if the condition is `true` and the third operand (to the right of the `:`) is the value of the conditional expression if the condition is `false`. For example, the following statement prints the value of `println`'s conditional-expression argument:

```
System.out.println(studentGrade >= 60 ? "Passed" : "F
```

The conditional expression in this statement evaluates to `"Passed"` if the `boolean` expression `studentGrade >= 60` is true and to `"Failed"` if it's false. Thus, this statement with the conditional operator performs essentially the same function as the `if`…`else` statement shown earlier in this section. The precedence of the conditional operator is low, so the entire conditional expression is normally placed in parentheses. Conditional expressions can be used as parts of larger expressions where `if`…`else` statements cannot.

Error-Prevention Tip 4.2

Use expressions of the same type for the second and third operands of the `?:` *operator to avoid subtle errors.*