Chapter 6

More Complex Conditionals

Despite the initial simplicity of if/else statements as introduced in Chapter 5, things can get interesting when if/else statements are composed of other if/else statements. The result is that more exciting programs can be written.

6.1 Nested Conditionals

The statements in the body of the if or the else may be any Java statements, including other if statements. These nested if statements can be used to develop arbitrarily complex control flow logic. Consider CheckRange (6.1), whose check() method determines if a number is between 0…10, inclusive.

```java
public class CheckRange {
    public boolean check(int value) {
        if (value >= 0) {
            // First check
            if (value <= 10) {
                // Second check
                return true;
            }
        }
        return false;
    }
}
```

Listing 6.1: CheckRange—Checks to see if a value is in the range 0–10

CheckRange’s check() method behaves as follows:

- The first condition is checked. If `value` is less than zero, the second condition is not evaluated and the statement following the outer if is executed. The statement after the outer if is the return of false.
- If the first condition finds `value` to be greater than or equal to zero, the second condition is then checked. If the second condition is met, `true` is returned.
Both conditions of this nested if must be met for the message to be printed. This program can be rewritten to behave the same way with only one if statement, as NewCheckRange (Listing 6.2) shows.

NewCheckRange uses a logical and to check both conditions at the same time. Its logic is simpler, using only one if statement, at the expense of a slightly more complex Boolean expression in its condition. The second version is preferable here, but a slight variation of CheckRange would be impossible to convert so only one if is used.

EnhancedRangeCheck (Listing 6.3) provides more specific messages instead of a simple acceptance or rejection. Exactly one of three messages is returned based on the method’s parameter. A single if or if/else statement cannot choose from among more than two different execution paths.

BinaryConversion (Listing 6.4) uses a series of if statements to build a 10-bit binary string representing the binary equivalent of a decimal integer supplied by the user.
public String decimalToBinary(int value) {
    // Integer must be less than 1024
    String result = ""; // Result string initially empty
    if (value >= 0 && value < 1024) {
        if (value >= 512) {
            result += "1";
            value %= 512;
        } else {
            result += "0";
        }
        if (value >= 256) {
            result += "1";
            value %= 256;
        } else {
            result += "0";
        }
        if (value >= 128) {
            result += "1";
            value %= 128;
        } else {
            result += "0";
        }
        if (value >= 64) {
            result += "1";
            value %= 64;
        } else {
            result += "0";
        }
        if (value >= 32) {
            result += "1";
            value %= 32;
        } else {
            result += "0";
        }
        if (value >= 16) {
            result += "1";
            value %= 16;
        } else {
            result += "0";
        }
        if (value >= 8) {
            result += "1";
            value %= 8;
        } else {
            result += "0";
        }
        if (value >= 4) {
            result += "1";
            value %= 4;
        } else {
            result += "0";
        }
    }
}
Listing 6.4: BinaryConversion—Builds the 10-bit binary equivalent of an integer supplied by the user

In BinaryConversion:

- The outer if checks to see if the value provided is in the proper range. The program only works for nonnegative values, so the range is 0–1023.

- Each of inner if's compare the entered integer against decreasing powers of two. If the number is large enough:
  - a 1 is appended to the string, and
  - that power of two's contribution to the value is removed via the remainder operator.

- For the ones place, the remaining value can be appended without first checking it—value will be either 0 or 1 at that point.

SimplerBinaryConversion (6.5) simplifies BinaryConversion's logic using only one if statement.
The sole if statement in SimplerBinaryConversion ensures that the user provides an integer in the proper range. The other if statements originally found in BinaryConversion are replaced by a clever sequence of integer arithmetic operations. The two programs—BinaryConversion and SimplerBinaryConversion—behave identically but SimplerBinaryConversion’s logic is simpler.

### 6.2 Multi-way if/else Statements

What if exactly one of several actions should be taken? Nested if/else statements are required, and the form of these nested if/else statements is shown in DigitToWord (6.6).

```java
public class DigitToWord {
    public String convert(int value) {
        String result;
        if (value < 0) {
            result = "Too small";
        } else {
            if (value == 0) {
                result = "zero";
            } else {
                if (value == 1) {
                    result = "one";
                } else {
                    if (value == 2) {
                        result = "two";
                    } else {
                        if (value == 3) {
                            result = "three";
                        } else {
                            if (value == 4) {
                                result = "four";
                            } else {
                                if (value == 5) {
                                    // Additional cases...
                                }
                            }
                        }
                    }
                }
            }
        }
        return result;
    }
}
```

Listing 6.5: SimplerBinaryConversion—Re-implements BinaryConversion with only one if statement
Observe the following about DigitToWord’s convert() method:

- It returns exactly one of eight strings depending on the user’s input.

- Notice that each if body contains a single assignment statement and each else body, except the last one, contains an if statement. The control logic forces the program execution to check each condition in turn. The first condition that matches wins, and its corresponding if body will be executed. If none of the conditions are true, the last else’s “too large” string will be returned.

- None of the curly braces used to delimit the if and else bodies are required since each body contains only a single statement (although a single deeply nested if/else statement is a mighty big statement).

DigitToWord is formatted according to the conventions used in earlier examples. As a consequence the mass of text drifts to the right as more conditions are checked. A commonly used alternative style, shown in RestyledDigitToWord (§6.7), avoids this rightward drift.
The formatting of RestyledDigitToWord does indeed hide the true structure of its logic, but it is so commonly used that it is regarded as acceptable by most programmers. The sequence of else if lines all indented to the same level identifies this construct as a multi-way if/else statement.

DateTransformer (6.8) uses a multi-way if/else to transform a numeric date in month/day (United States) format to an expanded English form, as in 2/14 → February 14. It also transforms the international day/month form to Español, as in 14-2 → 14 febrero.
result += day;

return result;
}

public String toSpanish(int day, int month) {
    String result = day + " ";
    // Translate month
    if (month == 1) {
        result += "enero";
    } else if (month == 2) {
        result += "febrero";
    } else if (month == 3) {
        result += "marzo";
    } else if (month == 4) {
        result += "abril";
    } else if (month == 5) {
        result += "mayo";
    } else if (month == 6) {
        result += "junio";
    } else if (month == 7) {
        result += "julio";
    } else if (month == 8) {
        result += "agosto";
    } else if (month == 9) {
        result += "septiembre";
    } else if (month == 10) {
        result += "octubre";
    } else if (month == 11) {
        result += "noviembre";
    } else {
        result += "diciembre";
    }
    return result;
}

Listing 6.8: DateTransformer—Transforms a numeric date into an expanded form

Valentine’s Day and Christmas would be transformed as:

Welcome to DrJava. Working directory is /Users/rick/java
> DateTransformer dt = new DateTransformer();
> dt.toEnglish(2, 14)
"February 14"
> dt.toSpanish(14, 2)
"14 febrero"
> dt.toEnglish(12, 25)
6.3 Errors in Conditional Expressions

Carefully consider each compound conditional used, such as

\[ \text{value} \geq 0 \land \text{value} \leq 10 \]

found in NewCheckRange (6.2). Confusing logical and and logical or is a common programming error. The Boolean expression

\[ \text{value} \geq 0 \lor \text{value} \leq 10 \]

is known as a tautology. A tautology is a Boolean expression that is always true. What value could the variable \text{value} assume that would make this Boolean expression false? No matter its value, one or both of the subexpressions will be true, so the compound expression is always true. The or expression here is just a complicated way of expressing the value true.

Another common error is contriving compound Boolean expressions that are always false, known as contradictions. Suppose you wish to exclude values from a given range; that is, reject \(0 \ldots 10\) and accept all other numbers. Is the Boolean expression in the following code fragment up to the task?

```java
// All but 0, 1, 2, ..., 10
if ( value < 0 && value > 10 ) {
    /* Code to execute goes here . . . */
}
```

A closer look at the condition reveals it can never be true. What number can be both less than zero and greater than ten at the same time? None can, of course, so the expression is a contradiction and a complicated way of expressing false. To correct this code fragment, replace the && operator with ||.

6.4 Summary

- The relational operators (==, !==, <, >, <=, and >=) evaluate to Boolean values.
- ! is the unary not operator.
- Boolean expressions can be combined via && and ||.
- The if statement can be used to optionally execute statements.
- A block groups a series of statements within a pair of curly braces ({}).
- The if statement has an optional else clause to require the selection between two alternate paths of execution.
- The if/else statements can be nested to achieve arbitrary complexity.
- Complex Boolean expressions require special attention, as they are easy to get wrong.
6.5 Exercises

1. Rewrite the following Boolean expressions in simpler form:
   a. !(x == 2)  
   b. x < 2 || x == 2  
   c. !(x < y)  
   d. !(x <= y)  
   e.  
   f.  
   g.  
   h.  
   i.  
   j.  
   k.  
   l.  
   m.  
   n.  
   o.  

2. What is the simplest tautology?

3. What is the simplest contradiction?