Chapter 3

Selections

3.1 Introduction 78
3.2 boolean Data Type, Values, and Expressions 78
3.3 if Statements 80
3.4 Two-Way if-else Statements 82
3.5 Nested if and Multi-Way if-else Statements 83
3.6 Common Errors and Pitfalls 85
3.7 Generating Random Numbers 89
3.8 Case Study: Computing Body Mass Index 91
3.9 Case Study: Computing Taxes 92
3.10 Logical Operators 95
3.11 Case Study: Determining Leap Year 99
3.12 Case Study: Lottery 100
3.13 switch Statements 102
3.14 Conditional Operators 105
3.15 Operator Precedence and Associativity 106
3.16 Debugging 108
Chapter 3

Selections

Objectives

- To declare `boolean` variables and write Boolean expressions using relational operators (§3.2).
- To implement selection control using one-way `if` statements (§3.3).
- To implement selection control using two-way `if-else` statements (§3.4).
- To implement selection control using nested `if` and multi-way `if` statements (§3.5).
- To avoid common errors and pitfalls in `if` statements (§3.6).
- To generate random numbers using the `Math.random()` method (§3.7).
- To program using selection statements for a variety of examples (SubtractionQuiz, BMI, ComputeTax) (§§3.7–3.9).
- To combine conditions using logical operators (`&&`, `||`, and `!`) (§3.10).
- To program using selection statements with combined conditions (LeapYear, Lottery) (§§3.11–3.12).
- To implement selection control using `switch` statements (§3.13).
- To write expressions using the `conditional expression` (§3.14).
- To examine the rules governing operator precedence and associativity (§3.15).
- To apply common techniques to debug errors (§3.16).
Chapter 3
Selections

3.1 Introduction 78

- Java provides selections that let you choose actions with two or more alternative courses.
- Selection statements use conditions. Conditions are Boolean expressions.
- Java has several types of selection statements:
  - if Statements, if … else statements, nested if statements
  - switch Statements
  - Conditional Expressions

3.2 boolean Data Type, Values, and Expressions 78

- Often in a program you need to compare two values, such as whether i is greater than j. Java provides six relational operators (also known as comparison operators) that can be used to compare two values. The result of the comparison is a Boolean value: true or false.

<table>
<thead>
<tr>
<th>Java Operator</th>
<th>Mathematics Symbol</th>
<th>Name</th>
<th>Example</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;</td>
<td>&lt;</td>
<td>less than</td>
<td>radius &lt; 0</td>
<td>false</td>
</tr>
<tr>
<td>&lt;=</td>
<td>≤</td>
<td>less than or equal to</td>
<td>radius &lt;= 0</td>
<td>false</td>
</tr>
<tr>
<td>&gt;</td>
<td>&gt;</td>
<td>greater than</td>
<td>radius &gt; 0</td>
<td>true</td>
</tr>
<tr>
<td>&gt;=</td>
<td>≥</td>
<td>greater than or equal to</td>
<td>radius &gt;= 0</td>
<td>true</td>
</tr>
<tr>
<td>==</td>
<td>=</td>
<td>equal to</td>
<td>radius == 0</td>
<td>false</td>
</tr>
<tr>
<td>!=</td>
<td>≠</td>
<td>not equal to</td>
<td>radius != 0</td>
<td>true</td>
</tr>
</tbody>
</table>

- Examples
  
  ```java
  System.out.println(1 < 2); // Displays true
  boolean b = (1 > 2);
  System.out.println("b is " + b); // Displays b is false
  ```
Problem: A Simple Math Learning Tool

- This example creates a program to let a first grader practice additions. The program randomly generates two single-digit integers number1 and number2 and displays a question such as “What is 7 + 9?” to the student. After the student types the answer, the program displays a message to indicate whether the answer is true or false.

LISTINT 3.1 AdditionQuiz.java

```java
import java.util.Scanner;

public class AdditionQuiz {
    public static void main(String[] args) {
        int number1 = (int)(System.currentTimeMillis() % 10);
        int number2 = (int)(System.currentTimeMillis() / 7 % 10);

        // Create a Scanner
        Scanner input = new Scanner(System.in);

        System.out.print(
            "What is " + number1 + " + " + number2 + "? ");

        int answer = input.nextInt();

        System.out.println(
            number1 + " + " + number2 + " = " + answer + " is " +
            (number1 + number2 == answer));
    }
}
```

What is 1 + 7? 8
1 + 7 = 8 is true

What is 4 + 8? 9
4 + 8 = 9 is false
3.3 if Statements 80

```java
if (booleanExpression) {
    statement(s);
} // execution flow chart is shown in Figure 3.1a
```

**Example**

```java
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
    radius + " is " + area);
} // if the Boolean expression evaluates to true, the statements in
the block are executed
```

**FIGURE 3.1** An if statement executes statements if the Boolean Expression evaluates as true

- **Note:**
  - The Boolean expression is enclosed in **parentheses** for all forms of the if statement. Thus, the outer parentheses in the previous if statements are required.

- The braces can be omitted if they enclose a **single** statement.

```java
if (i > 0) { 
  System.out.println("i is positive"); 
}  
(a) Wrong (b) Correct
```

```java
if (i > 0) { 
  System.out.println("i is positive"); 
}  
(a) Equivalent (b) 
```
• Write a program that prompts the user to enter an integer. If the number is a multiple of 5, print HiFive. If the number is divisible by 2, print HiEven.

LISTING 3.2 SimpleIfDemo.java

```java
import java.util.Scanner;

public class SimpleIfDemo {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Enter an integer: ");
        int number = input.nextInt();

        if (number % 5 == 0) {
            System.out.println("HiFive");
        }

        if (number % 2 == 0) {
            System.out.println("HiEven");
        }
    }
}
```

Enter an integer: 4
HiEven

Enter an integer: 30
HiFive
HiEven
3.4 Two-Way if-else Statements

```java
if (booleanExpression) {
    statement(s)-for-the-true-case;
} else {
    statement(s)-for-the-false-case;
}
```

FIGURE 3.2 An if-else statement executes statements for the true case if the Boolean expression evaluations are true; otherwise, statements for the false case are executed.

- if...else Example

```java
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area for the circle of radius " +
                       radius + " is " + area);
} else {
    System.out.println("Negative input"); // braces may be omitted
}
```

Note: If radius >= 0 is true, area is computed and displayed; if it is false, the message “Negative input” is printed.

- Using the if … else statement, you can rewrite the following code for determining whether a number is even or odd, as follows:

```java
if (number % 2 == 0)
    System.out.println(number + " is even.");
if (number % 2 != 0)
    System.out.println(number + " is odd.");
```

// rewriting the code using else

```java
if (number % 2 == 0)
    System.out.println(number + " is even.");
else
    System.out.println(number + " is odd.");
```

Note: This is more efficient because whether number % 2 is 0 is tested only once.
3.5 Nested if and Multi-Way if-else Statements 83

- The statement in an if or if ... else statement can be any legal Java statement, including another if or if ... else statement. The inner if statement is said to be nested inside the outer if statement.
- The inner if statement can contain another if statement.
- There is no limit to the depth of the nesting.

```java
if (i > k) {
    if (j > k)
        System.out.println("i and j are greater than k");
} else
    System.out.println("i is less than or equal to k");
// the if (j > k) is nested inside the if (i > k)
```

- The nested if-else statement can be used to implement multiple alternatives.

```java
if (score >= 90.0)
    System.out.print("A");
else if (score >= 80.0)
    System.out.print("B");
else if (score >= 70.0)
    System.out.print("C");
else if (score >= 60.0)
    System.out.print("D");
else
    System.out.print("F");
```

FIGURE 3.3 A preferred format for multiple alternatives is shown in (b) using a multi-way if-else statement.

![Diagram](image)

FIGURE 3.4 You can use a multi-way if-else statement to assign a grade.
3.6 Common Errors and Pitfalls 85

- **Common Error 1: Forgetting Necessary Braces**

```
if (radius >= 0)
    area = radius * radius * PI;
System.out.println("The area "+ " is "+ area);
```

(a) Wrong

```
if (radius >= 0)
{
    area = radius * radius * PI;
    System.out.println("The area "+ " is "+ area);
}
```

(b) Correct

- **Common Error 2: Wrong Semicolon at the if Line**
  - Adding a semicolon at the end of an if clause is a common mistake.
  - This mistake is hard to find, because it is not a compilation error or a runtime error, it is a logic error.
  - This error often occurs when you use the next-line block style.

```
if (radius >= 0) ;
{
    area = radius * radius * PI;
    System.out.println("The area "+ " is "+ area);
}
```

(a) Wrong

```
if (radius >= 0) {
    area = radius * radius * PI;
    System.out.println("The area "+ " is "+ area);
}
```

(b) Correct

- **Common Error 3: Redundant Testing of Boolean Values**
  - To test whether a Boolean variable is true or false in a test condition, it is redundant to use the equality comparison operator like this:

```
if (even == true)
    System.out.println("It is even.");
```

(a) Equivalent

```
if (even)
    System.out.println("It is even.");
```

(b) Equivalent

**Caution**
- What’s wrong with the following?

```
if (even == true)
    System.out.println("It is even.");
```

This statement does not have syntax errors. It assigns true to even so that even is always true.
• **Common Error 4: Dangling else Ambiguity**
  o The *else* clause matches the most recent **unmatched** *if* clause in the same block. For example, the following statement:

```java
int i = 1; int j = 2; int k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
else
    System.out.println("B");
```

is equivalent to:

```java
int i = 1; int j = 2; int k = 3;
if (i > j)
    if (i > k)
        System.out.println("A");
else
    System.out.println("B");
```

o Nothing is printed from the preceding statement because the compiler ignores indentation. To force the *else* clause to match the first *if* clause, you must add a pair of braces:

```java
int i = 1; int j = 2; int k = 3;
if (i > j) {
    if (i > k)
        System.out.println("A");
} else
    System.out.println("B");
```

This statement prints B.

• **Common Error 5: Equality Test of Two Floating-Point Values**
  o You expect the following code to display true, but surprisingly it displays false.

```java
double x = 1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1;
System.out.println(x == 0.5);
```

Here, x is not exactly 0.5, but is 0.5000000000000001.
• **Common Pitfall 1: Simplifying Boolean Variable Assignment**
  o Often new Programmers write that assigns a test condition to a Boolean variable like the code in (a).

  ```
  if (number % 2 == 0) {
    even = true;
  } else {
    even = false;
  }
  ```

  o The code can be simplified by assigning the test value directly to the variable, as shown in (b). This is not an error, but it should be better written as shown in (b).

  ```
  boolean even = number % 2 == 0;
  ```

  (a)  
  (b)

• **Common Pitfall 2: Avoiding Duplicate Code in Different Cases**
  o Often, new programmers write the duplicate code in different cases that should be combined in one place. For example, the highlighted code in the following statement is duplicated.

  ```
  if (inState) {
    tuition = 5000;
    System.out.println("The tuition is " + tuition);
  } else {
    tuition = 15000;
    System.out.println("The tuition is " + tuition);
  }
  ```

  o The new code removes the duplication and makes the code easy to maintain, because you only need to change in one place if the print statement is modified.
3.7 Generating Random Numbers 89

- This example creates a program to teach a first grade child how to learn subtractions. The program randomly generates two single-digit integers number1 and number2 with number1 > number2 and displays a question such as “What is 9 – 2?” to the student, as shown in the figure. After the student types the answer in the input dialog box, the program displays a message dialog box to indicate whether the answer is correct.

LISTING 3.3 SubtractionQuiz.java

```java
import java.util.Scanner;

public class SubtractionQuiz {
    public static void main(String[] args) {
        // 1. Generate two random single-digit integers
        int number1 = (int)(Math.random() * 10);
        int number2 = (int)(Math.random() * 10);

        // 2. If number1 < number2, swap number1 with number2
        if (number1 < number2) {
            int temp = number1;
            number1 = number2;
            number2 = temp;
        }

        // 3. Prompt the student to answer “what is number1 - number2?”
        System.out.print("What is "+ number1 + " - " + number2 + "? ");
        Scanner input = new Scanner(System.in);
        int answer = input.nextInt();

        // 4. Grade the answer and display the result
        if (number1 - number2 == answer)
            System.out.println("You are correct!");
        else
            System.out.println("Your answer is wrong. \n" + number1 + " - " + number2 + " should be " + (number1 - number2));
    }
}
```

What is 6 - 6? 0
You are correct!

What is 9 - 2? 5
Your answer is wrong.
9 - 2 should be 7
3.8 Case Study: Computing Body Mass Index 91

- Body Mass Index (BMI) is a measure of health on weight. It can be calculated by taking your weight in kilograms and dividing by the square of your height in meters. The interpretation of BMI for people 16 years or older is as follows:

<table>
<thead>
<tr>
<th>BMI</th>
<th>Interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMI &lt; 18.5</td>
<td>Underweight</td>
</tr>
<tr>
<td>18.5 ≤ BMI &lt; 25.0</td>
<td>Normal</td>
</tr>
<tr>
<td>25.0 ≤ BMI &lt; 30.0</td>
<td>Overweight</td>
</tr>
<tr>
<td>BMI ≥ 30.0</td>
<td>Obese</td>
</tr>
</tbody>
</table>

LISTING 3.4 ComputeAndInterpretBMI.java

```java
import java.util.Scanner;

public class ComputeAndInterpretBMI {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter weight in pounds
        System.out.print("Enter weight in pounds: ");
        double weight = input.nextDouble();

        // Prompt the user to enter height in inches
        System.out.print("Enter height in inches: ");
        double height = input.nextDouble();

        final double KILOGRAMS_PER_POUND = 0.45359237; // Constant
        final double METERS_PER_INCH = 0.0254; // Constant

        // Compute BMI
        double weightInKilograms = weight * KILOGRAMS_PER_POUND;
        double heightInMeters = height * METERS_PER_INCH;
        double bmi = weightInKilograms / (heightInMeters * heightInMeters);

        // Display result
        System.out.println("BMI is " + bmi);
        if (bmi < 18.5)
            System.out.println("Underweight");
        else if (bmi < 25)
            System.out.println("Normal");
        else if (bmi < 30)
            System.out.println("Overweight");
        else
            System.out.println("Obese");
    }
}
```

Enter weight in pounds: 146
Enter height in inches: 70
BMI is 20.948603801493316
Normal
3.9 Case Study: Computing Taxes

- The US federal personal income tax is calculated based on the filing status and taxable income. There are four filing statuses: single filers, married filing jointly, married filing separately, and head of household. The tax rates for 2009 are shown below.

**TABLE 3.2 2009 U.S. Federal Personal Tax Rates**

<table>
<thead>
<tr>
<th>Marginal Tax Rate</th>
<th>Single</th>
<th>Married Filing Jointly or Qualifying Widow(er)</th>
<th>Married Filing Separately</th>
<th>Head of Household</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>$0–$8,350</td>
<td>$0–$16,700</td>
<td>$0–$8,350</td>
<td>$0–$11,950</td>
</tr>
<tr>
<td>15%</td>
<td>$8,351–$33,950</td>
<td>$16,701–$67,900</td>
<td>$8,351–$33,950</td>
<td>$11,951–$45,500</td>
</tr>
<tr>
<td>35%</td>
<td>$372,951+</td>
<td>$372,951+</td>
<td>$186,476+</td>
<td>$372,951+</td>
</tr>
</tbody>
</table>

**LISTING 3.5 ComputeTax.java**

```java
import java.util.Scanner;

public class ComputeTax {
    public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);

        // Prompt the user to enter filing status
        System.out.print("0-single filer, 1-married jointly or " + "qualifying widow(er), 2-married separately, 3-head of " + "household) Enter the filing status: ");
        int status = input.nextInt();

        // Prompt the user to enter taxable income
        System.out.print("Enter the taxable income: ");
        double income = input.nextDouble();

        // Compute tax
        double tax = 0;

        if (status == 0) { // Compute tax for single filers
            if (income <= 8350)
                tax = income * 0.10;
            else if (income <= 33950)
                tax = 8350 * 0.10 + (income - 8350) * 0.15;
            else if (income <= 82250)
                tax = 8350 * 0.10 + (33950 - 8350) * 0.15 + (income - 33950) * 0.25;
            else if (income <= 171550)
                tax = 8350 * 0.10 + (33950 - 8350) * 0.15 + (82250 - 33950) * 0.25 + (income - 82250) * 0.28;
```
else if (income <= 372950)
tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
(82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
(income - 171550) * 0.33;
else
tax = 8350 * 0.10 + (33950 - 8350) * 0.15 +
(82250 - 33950) * 0.25 + (171550 - 82250) * 0.28 +
(372950 - 171550) * 0.33 + (income - 372950) * 0.35;
}
else if (status == 1) { // Compute tax for married file jointly
  // Left as exercise
}
else if (status == 2) { // Compute tax for married separately
  // Left as exercise
}
else if (status == 3) { // Compute tax for head of household
  // Left as exercise
}
else {
  System.out.println("Error: invalid status");
  System.exit(1);
}

// Display the result
System.out.println("Tax is "+(int)(tax * 100) / 100.0);

(0-single filer, 1-married jointly or qualifying widow(er), 2-
marrid separately, 3-head of household) Enter the filing status: 0
Enter the taxable income: 40000
Tax is 6187.5
3.10 Logical Operators 95

- Logical operators, also known as Boolean operators, operate on Boolean values to create a new Boolean value.

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>!</td>
<td>not</td>
<td>logical negation</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>and</td>
<td>logical conjunction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>^</td>
<td>exclusive or</td>
<td>logical exclusion</td>
</tr>
</tbody>
</table>

- Examples

  ```
  && (and)  
  (1 < x) && (x < 100)
  || (or)   
  (lightsOn) || (isDayTime)
  ! (not)   
  !(isStopped)
  ```

**TABLE 3.4 Truth Table for Operator !**

<table>
<thead>
<tr>
<th>p</th>
<th>!p</th>
<th>Example (assume age = 24, weight = 140)</th>
</tr>
</thead>
<tbody>
<tr>
<td>true</td>
<td>false</td>
<td>!(age &gt; 18) is false, because (age &gt; 18) is true.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>!(weight == 150) is true, because (weight == 150) is false.</td>
</tr>
</tbody>
</table>

**TABLE 3.5 Truth Table for Operator &&**

<table>
<thead>
<tr>
<th>p₁</th>
<th>p₂</th>
<th>p₁ &amp;&amp; p₂</th>
<th>Example (assume age = 24, weight = 140)</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &lt;= 18) &amp;&amp; (weight &lt; 140) is false,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>because (age &lt;= 18) and (weight &lt; 140) are both false.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>(age &lt;= 18) &amp;&amp; (weight &lt;= 140) is false,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>because (age &lt;= 18) is false.</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 18) &amp;&amp; (weight &gt; 140) is false,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>because (weight &gt; 140) is false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 18) &amp;&amp; (weight &gt;= 140) is true,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>because both (age &gt; 18) and (weight &gt;= 140) are true.</td>
</tr>
</tbody>
</table>

**TABLE 3.6 Truth Table for Operator ||**

| p₁  | p₂  | p₁ || p₂ | Example (assume age = 24, weight = 140)                      |
|-----|-----|--------|-------------------------------------------------------------|
| false| false| false  | (age > 34) || (weight >= 150) is false,                      |
|      |     |        | because (age > 34) and (weight >= 150) are both false.      |
| false| true | true   | (age > 34) || (weight <= 150) is true,                       |
|      |     |        | because (age > 34) is false, but (weight <= 150) is true.   |
| true | false| true   | (age < 34) || (weight >= 150) is false,                      |
|      |     |        | because (age < 34) is true.                                 |
| true | true | true   | (age < 34) || (weight <= 150) is true,                       |
|      |     |        | because both (age < 34) and (weight <= 150) are true.        |
### TABLE 3.7 Truth Table for Operator ^

<table>
<thead>
<tr>
<th>p₁</th>
<th>p₂</th>
<th>p₁ ^ p₂</th>
<th>Example (assume age = 24, weight = 140)</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>(age &gt; 34) ^ (weight &gt; 140) is false, because (age &gt; 34) and (weight &gt; 140) are both false.</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>true</td>
<td>(age &gt; 34) ^ (weight &gt;= 140) is true, because (age &gt; 34) is false but (weight &gt;= 140) is true.</td>
</tr>
<tr>
<td>true</td>
<td>false</td>
<td>true</td>
<td>(age &gt; 14) ^ (weight &gt; 140) is true, because (age &gt; 14) is true but (weight &gt; 140) is false.</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>false</td>
<td>(age &gt; 14) ^ (weight &gt;= 140) is false, because (age &gt; 14) and (weight &gt;= 140) are both true.</td>
</tr>
</tbody>
</table>

### LISTING 3.6 TestBooleanOperators.java

```java
import java.util.Scanner;

public class TestBooleanOperators {
    public static void main(String[] args) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);

        // Receive an input
        System.out.print("Enter an integer: ");
        int number = input.nextInt();

        if (number % 2 == 0 && number % 3 == 0)
            System.out.println(number + " is divisible by 2 and 3.");

        if (number % 2 == 0 || number % 3 == 0)
            System.out.println(number + " is divisible by 2 or 3.");

        if (number % 2 == 0 ^ number % 3 == 0)
            System.out.println(number + " divisible by 2 or 3, but not both.");
    }
}
```

Enter an integer: 4
4 is divisible by 2 or 3.
4 divisible by 2 or 3, but not both.

Enter an integer: 18
18 is divisible by 2 and 3.
18 is divisible by 2 or 3.
3.11 Case Study: Determining Leap Year 99

- This program first prompts the user to enter a year as an int value and checks if it is a leap year.
- A year is a leap year if it is divisible by 4 but not by 100, or it is divisible by 400.

\[(\text{year} \% 4 == 0 && \text{year} \% 100 != 0) || (\text{year} \% 400 == 0)\]

LISTING 3.7 LeapYear.java

```java
import java.util.Scanner;

public class LeapYear {
    public static void main(String args[]) {
        // Create a Scanner
        Scanner input = new Scanner(System.in);
        System.out.print("Enter a year: ");
        int year = input.nextInt();

        // Check if the year is a leap year
        boolean isLeapYear =
            (year % 4 == 0 && year % 100 != 0) || (year % 400 == 0);

        // Display the result in a message dialog box
        System.out.println(year + " is a leap year? " + isLeapYear);
    }
}
```

Enter a year: 2008
2008 is a leap year? true

Enter a year: 1900
2002 is a leap year? false
3.12 Case Study: Lottery 100

- Write a program that randomly generates a lottery of a two-digit number, prompts the user to enter a two-digit number, and determines whether the user wins according to the following rule:
  - If the user input matches the lottery in exact order, the award is $10,000.
  - If the user input match all the digits in the lottery, the award is $3,000.
  - If one digit in the user input matches a digit in the lottery, the award is $1,000.

**LISTING 3.8 Lottery.java**

```java
import java.util.Scanner;

public class Lottery {
    public static void main(String[] args) {
        // Generate a lottery
        int lottery = (int)(Math.random() * 100);

        // Prompt the user to enter a guess
        Scanner input = new Scanner(System.in);
        System.out.print("Enter your lottery pick (two digits): ");
        int guess = input.nextInt();

        // Get digits from lottery
        int lotteryDigit1 = lottery / 10;
        int lotteryDigit2 = lottery % 10;

        // Get digits from guess
        int guessDigit1 = guess / 10;
        int guessDigit2 = guess % 10;

        System.out.println("The lottery number is " + lottery);

        // Check the guess
        if (guess == lottery)
            System.out.println("Exact match: you win $10,000");
        else if (guessDigit2 == lotteryDigit1 && guessDigit1 == lotteryDigit2)
            System.out.println("Match all digits: you win $3,000");
        else if ((guessDigit1 == lotteryDigit1 || guessDigit1 == lotteryDigit2)
                  || (guessDigit2 == lotteryDigit1 || guessDigit2 == lotteryDigit2))
            System.out.println("Match one digit: you win $1,000");
        else
            System.out.println("Sorry, no match");
    }
}
```

Enter your lottery pick (two digits): 45
The lottery number is 12
Sorry, no match

Enter your lottery pick (two digits): 23
The lottery number is 34
Match one digit: you win $1,000
3.13 switch Statements

- One can write a switch statement to replace a nested if statement. For example,

```java
switch (status) {
    case 0: compute taxes for single filers;
        break;
    case 1: compute taxes for married file jointly;
        break;
    case 2: compute taxes for married file separately;
        break;
    case 3: compute taxes for head of household;
        break;
    default: System.out.println("Errors: invalid status");
        System.exit(0);
} // checks if status matches the values 0, 1, 2, or 3 respectively.
```

The switch Statement Rules:
- The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.
- The value1... and valueN must have the same data type as the value of the switch-expression. value1... and valueN are constant expressions, meaning that they cannot contain variables in the expression, such as 1 + x.
- When the value in a case statement matches the value of the switch-expression, the statements starting from this case are executed until either a break statement or the end of the switch statement is reached.
• The keyword **break** is optional. The **break** statement **immediately ends** the **switch** statement.

• The **default** case, which is optional, can be used to perform actions when **none** of the specified cases matches the **switch-expression**.

• The cases statements are checked in **sequential** order, but the **order** of the cases (including the default case) **does not** matter. However, it is a good programming style to follow the logical sequence of the cases and place the default case at the end.

**Caution**

• **Do not** forget to use a **break** statement when one is needed. Once a case is matched, the statements starting from the matched case are executed until a break statement or the end of the switch statement is reached. This is referred to as fall-through behavior.

• For example, the following code displays Weekdays for day of 1 to 5 and Weekends for day 0 and 6.

```java
switch (day) {
    case 1:
    case 2:
    case 3:
    case 4:
    case 5: System.out.println("Weekday"); break;
    case 0:
    case 6: System.out.println("Weekend");
}
```
• Problem (Chinese Zodiac): Write a program that prompts the user to enter a year and displays the animal for the year.

![Chinese Zodiac Diagram]

FIGURE 3.6 The Chinese Zodiac is based on a 12-year cycle

LISTING 3.9 ChineseZodiac.java

```java
import java.util.Scanner;

public class ChineseZodiac {
    public static void main(String[] args) {
        Scanner input = new Scanner(System.in);

        System.out.print("Enter a year: ");
        int year = input.nextInt();

        switch (year % 12) {
            case 0: System.out.println("monkey"); break;
            case 1: System.out.println("rooster"); break;
            case 2: System.out.println("dog"); break;
            case 3: System.out.println("pig"); break;
            case 4: System.out.println("rat"); break;
            case 5: System.out.println("ox"); break;
            case 6: System.out.println("tiger"); break;
            case 7: System.out.println("rabbit"); break;
            case 8: System.out.println("dragon"); break;
            case 9: System.out.println("snake"); break;
            case 10: System.out.println("horse"); break;
            case 11: System.out.println("sheep"); break;
        }
    }
}
```

Enter a year: 1963
rabbit

Enter a year: 2014
horse
3.14 Conditional Operators 105

- Conditional expressions are in different style, which no explicit if in the statement. The syntax is shown below:

  BooleanExpression ? expression1 : expression2;

  The result of this conditional expression expression1 if BooleanExpression is true; otherwise the result is expression2.

- For example:

  ```java
  if (x > 0)
      y = 1
  else
      y = -1;
  ```

  is equivalent to

  ```java
  y = (x > 0) ? 1 : -1;
  ```

- For example:

  ```java
  if (num % 2 == 0)
      System.out.println(num + " is even");
  else
      System.out.println(num + " is odd");
  ```

  is equivalent to

  ```java
  System.out.println((num % 2 == 0) ? num + " is even" : num + " is odd");
  ```

- For example:

  ```java
  Max = (num1 > num2) ? num1 : num2;
  ```

**Note**

- The symbols ? and : appear together in a **conditional expression**. They form a condition operator. The operator is called a **ternary operator** because it uses three operands.
### 3.15 Operator Precedence and Associativity

How to evaluate?

\[ 3 + 4 * 4 > 5 * (4 + 3) - 1 \]

- The precedence rule defines precedence for operators as shown below.
- If operators with the same precedence are next to each other, their associativity determines the order of evaluation.
- All binary operators except assignment operators are **left-associative**. For example:

\[
a - b + c - d \text{ is equivalent to } ((a - b) + c) - d
\]

Assignment operators are **right-associative**. Therefore, the expression

\[
a = b += c = 5 \text{ is equivalent to } a = (b += (c = 5))
\]

**TABLE 3.8 Operator Precedence Chart**

<table>
<thead>
<tr>
<th>Operator</th>
<th>Precedence</th>
</tr>
</thead>
<tbody>
<tr>
<td>var++, var- (Postfix)</td>
<td></td>
</tr>
<tr>
<td>+, - (Unary plus and minus), ++var, --var (Prefix)</td>
<td></td>
</tr>
<tr>
<td>(type) (Casting)</td>
<td></td>
</tr>
<tr>
<td>! (Not)</td>
<td></td>
</tr>
<tr>
<td>*, /, % (Multiplication, division, and remainder)</td>
<td></td>
</tr>
<tr>
<td>+, - (Binary addition and subtraction)</td>
<td></td>
</tr>
<tr>
<td>&lt;, &lt;=, &gt;, &gt;= (Comparison)</td>
<td></td>
</tr>
<tr>
<td>==, != (Equality)</td>
<td></td>
</tr>
<tr>
<td>&amp; (Unconditional AND)</td>
<td></td>
</tr>
<tr>
<td>^ (Exclusive OR)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp;= (Conditional AND) Short-circuit AND</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>=, +=, -=, *=, /=, %= (Assignment operator)</td>
<td></td>
</tr>
</tbody>
</table>

- Example
  Applying the operator precedence and associativity rule, the expression \(3 + 4 * 4 > 5 * (4 + 3) - 1\) is evaluated as follows:

\[
3 + 4 * 4 > 5 * (4 + 3) - 1
\]

(1) inside parentheses first

\[
3 + 4 * 4 > 5 * 7 - 1
\]

(2) multiplication

\[
3 + 16 > 5 * 7 - 1
\]

(3) multiplication

\[
3 + 16 > 35 - 1
\]

(4) addition

\[
19 > 35 - 1
\]

(5) subtraction

\[
19 > 34
\]

(6) greater than

false
3.16 Debugging 108

- Logic errors are called **bugs**. The process of finding and correcting errors is called **debugging**. A common approach to debugging is to use a combination of methods to narrow down to the part of the program where the bug is located. You can hand-trace the program (i.e., catch errors by reading the program), or you can insert print statements in order to show the values of the variables or the execution flow of the program. This approach might work for a short, simple program. But for a large, complex program, the most effective approach for debugging is to use a debugger utility.

- **Debugger** is a program that facilitates debugging. You can use a debugger to
  - Executing a single statement at a time.
  - Tracing into or stepping over a method.
  - Setting breakpoints.
  - Displaying variables.
  - Displaying call stack.
  - Modifying variables.