Java: An Introduction to Problem Solving and Programming, 7th Edition

The method of dynamic binding in Java is considered a key feature of object-oriented programming. In this chapter, we will explore how Java implements dynamic binding and how it can be used to enhance program flexibility and reusability. We will also discuss the role of interfaces in implementing dynamic binding.

### 3.5.1 Dynamic Binding in Java

**Dynamic Binding** refers to the mechanism by which a method's implementation is determined at runtime, based on the actual object that is being operated on. In Java, dynamic binding is achieved through the use of **method dispatching** and **virtual method calls**.

**Method Dispatching** is the process of selecting the correct method implementation for a given method call at runtime. In Java, method dispatching is performed by the **runtime system** based on the type of the actual object passed as an argument.

**Virtual Method Calls** are method calls that are made through a method reference or a method handle. The implementation of the method is determined at runtime based on the actual object that is being operated on.

### 3.5.2 Interfaces and Dynamic Binding

In Java, **interfaces** are used to specify a contract for objects. An interface is a group of related methods and variables that are implemented by classes. When a class implements an interface, it is required to provide implementations for all the methods defined in the interface.

Interfaces are a powerful tool for implementing dynamic binding in Java. By defining an interface and specifying the methods that should be implemented by classes, you can enforce a certain level of behavior and make it easier to change the implementation of those methods without affecting the rest of the system.

### 3.5.3 Case Study: Dynamic Binding in a Calculator

In this case study, we will implement a simple calculator using interfaces. We will define an interface `Calculator` that specifies the methods for basic arithmetic operations. We will then create a class `SimpleCalculator` that implements the `Calculator` interface and provides the actual implementations for those methods.

```java
// Calculator interface
public interface Calculator {
    double add(double a, double b);
    double subtract(double a, double b);
    double multiply(double a, double b);
    double divide(double a, double b);
}

// SimpleCalculator class
public class SimpleCalculator implements Calculator {
    public double add(double a, double b) {
        return a + b;
    }
    public double subtract(double a, double b) {
        return a - b;
    }
    public double multiply(double a, double b) {
        return a * b;
    }
    public double divide(double a, double b) {
        return a / b;
    }
}
```

By using interfaces and dynamic binding, we can easily modify the behavior of the calculator without changing the client code. This makes the code more flexible and easier to maintain.

### Conclusion

In this chapter, we have learned about dynamic binding in Java, which is an important feature of object-oriented programming. We discussed how method dispatching and virtual method calls work in Java, and how interfaces can be used to implement dynamic binding. By understanding these concepts, you can write more flexible and maintainable code in Java.

---

**References**


---

**Appendix A: Java Environment Setup**

Java is a programming language that can be used on any computer with a Java Virtual Machine (JVM) installed. To set up your Java environment, you will need to follow these steps:

1. Download the latest version of the Java Development Kit (JDK) from the Oracle website.
2. Install the JDK on your computer.
3. Add the JDK to your system's path environment variable.
4. Verify that the JDK is installed correctly by running `java -version` in your terminal or command prompt.

---

**Appendix B: Java Code Examples**

Here are some Java code examples to help you get started with programming in Java:

```java
public class HelloJava {
    public static void main(String[] args) {
        System.out.println("Hello Java!");
    }
}
```

---

**Appendix C: Java Resources**

Here are some resources for learning more about Java:

- [Oracle's Java Tutorials](https://docs.oracle.com/javase/tutorial/)
- [Java Community Process](https://jcp.org)
- [JavaFX](https://openjfx.io)

---

**Appendix D: Java Standards**

Java is a language that is governed by a set of standards. The most important standards for Java are:

- [Java Language Specification](https://openjdk.java.net/projects/jdk/jdk8/jre/api/)
- [Java Platform Module System Specification](https://openjdk.java.net/projects/jdk/jdk8/jre/api/modules)

---

**Appendix E: Java Tools and Libraries**

Java is a language that is supported by a wide range of tools and libraries. Some of the most important tools and libraries for Java are:

- [NetBeans](https://netbeans.org)
- [Eclipse](https://www.eclipse.org)
- [IntelliJ IDEA](https://www.jetbrains.com/idea)

---

**Appendix F: Java Certification**

Java is a language that is supported by a number of certifications. The most important certification for Java is the [Java SE 8 Programmer](https://www.oracle.com/certification/learn/8-se-programmer.html)

---

**Appendix G: Java Community**

Java is a language that is supported by a number of communities. The most important community for Java is the [Java Community Process](https://jcp.org)

---

**Appendix H: Java Community Process**

Java is a language that is supported by a number of communities. The most important community for Java is the [Java Community Process](https://jcp.org)

---

**Appendix I: Java Community Process**

Java is a language that is supported by a number of communities. The most important community for Java is the [Java Community Process](https://jcp.org)
Within these braces are typically one or more parts called methods. For example, the sample program in the text delighted you. Why not begin to learn a new programming language that might become Java—though it was not yet called that. This new language was called Smalltalk. It was a system-oriented language that combined object-oriented programming with the idea of running live programs in a virtual machine. This was a very different approach from the existing language offerings. It sounds like a humble engineering task, but in fact it's a very challenging one. Home appliances are controlled by a wide variety of microprocessors. A simple version of this controller might take on the form of a set of ROM instructions. This might be followed by a set of RAM instructions, the RAM instructions performing the actual computation, while the ROM instructions are used to control the execution of the RAM operations. A typical ROM contains about 256 bytes and is about 4 kilobytes in size. A typical RAM contains about 64 kilobytes to 1 megabyte. ROM is placed so that the computer can read, but not modify or write to it. RAM can be written, read, and altered many times. The possible contents of RAM are changed at the rate of one bit per 100 microsecond. The possible contents of ROM are changed at the rate of one bit per 0.5 second. The typical clock speed of a ROM is about 500 kilohertz. A computer program is a set of instructions to the machine. A typical program consists of 1000-5000 instructions. The first step is to get a list of the instructions. The second step is to examine a simple Java program. This introduction is simply an overview and a presentation of some terminology. We will consider computer systems in terms of a collection of components, including a CPU, memory, and input/output devices. We will also consider the flow of control through a computer, including the processing of instructions and the use of memory. We will then consider the flow of data through the computer, including the use of memory and the use of input/output devices. We will also consider the flow of control through a computer, including the processing of instructions and the use of memory. We will then consider the flow of data through the computer, including the use of memory and the use of input/output devices. We will also consider the flow of control through a computer, including the processing of instructions and the use of memory. We will then consider the flow of data through the computer, including the use of memory and the use of input/output devices.
A method in a computer program is a part of the program that performs a specific task. The input to a method can be called arguments or parameters, and the output is often called the result or return value. The number and types of arguments can be specified when the method is called, and the same method can be called multiple times with different arguments.

For example, consider the method of a computer program that calculates the total cost of a list of purchases. The method might be called `totalCost` and might have the following signature:

```
public double totalCost(List<Double> prices)
```

This method takes a list of `Double` values as an argument, which represent the prices of the items in the list. The method returns a `double` value, which is the total cost of all the items in the list.

In Java, methods are defined within classes. A class is a blueprint for creating objects, which are instances of the class. Each object has its own instance variables, which are values stored within the object, and methods, which are routines that can be called to perform actions on the object.

For example, consider a class `PizzaOrder` that represents an order for a pizza. The class might have instance variables for the size of the pizza, the toppings, and the customer's name, and methods for calculating the total cost of the order and printing a receipt.

```
public class PizzaOrder {
    private String size;
    private String[] toppings;
    private String customerName;

    public double totalCost() {
        // Calculate the cost of the pizza based on size and toppings
        return 10.0;  // Example cost calculation
    }

    public void printReceipt() {
        // Print a receipt for the pizza order
    }
}
```

Methods are an essential part of Java programming, and understanding how to write and use them is crucial for any Java developer.
When you assign an integer value to a variable of a floating-point type—double, for example—the statement double point = 7; is equivalent to double point = (double)7; The type cast (double) is implicit in the assignment. In addition, you can assign a value of type char to a variable of type int or to any of the numeric types that are covariant, such as float. However, we do not advise assigning a value of type char to a variable of type int when we discuss keyboard input. However, we do not advise assigning a value of type char to a variable of type int. You do not need to provide the type-casting operator when assigning a value of type char to a variable of type int.

The position of the decimal point. The number before the e may contain a decimal point, although it doesn’t have to. The number after the e tells you to move the decimal point to the right by the number of places indicated by the number after the e. The number before the e is a normal number, frequently called either e notation or floatingpoint notation. Because keyboards have no way of writing exponents, the 10 in the number 8.65 3 108 is written as 8.65e8. The e stands for exponent, since it is followed by a number that is thought of as a power of 10. For example, 4.563 105 means 4.563 10 5, which is 4,563,000. Mathematically, e notation is very convenient as a shorthand for exponential notation.

The equal sign, =, is called the assignment operator when it is used in an assignment statement. It does not mean what it usually means in mathematics and the physical sciences, scientific notation. For instance, consider the number 865000000.0. This number is too large to write as a single number and characters and to perform single string printing. Why is it that assignment statements that accomplish the same result as a statement that prints the number in scientific notation? The reason is that Java provides a mechanism for producing such a value. The value is produced in a form that is output from a program. The statement int width, depth; System.out.println("Enter the height, width, and depth of cube: "); width = keyboard.nextInt(); depth = keyboard.nextInt(); double volume = width * depth * height; volume = volume / 10; System.out.println("The volume is "+ volume + " cubic units."); is equivalent to width = depth = height = 0; System.out.println("Enter the height, width, and depth of cube: "); System.out.println("The volume is "+ volume + " cubic units."); because the assignment operator puts the right value in the right variable. The numbers width, depth, and height are assigned to the variables width, depth, and height, respectively. These variables are declared in the program, which means that you must state some basic information about each one. The compiler—and so ultimately the computer—needs to know the name and the type of each variable. For example, the variables width, depth, and height are declared with the following statement: int width, depth, height;
Java language (although the compiler might guess that it is a quoted string with one missing quote or guess that you ... marks as part of the quoted string, unless you tell it that you mean to do so. You tell the compiler that you mean to hard! 012345678901234567890123 The word "hard" starts at index 19 The changed string is: TEXT PROCESSING IS EASY! ... Basic Computation Gotcha  String Index Out of Bounds The first character in a string is at index 0, not 1. So if a string ... will not discuss this class here because we do not need it. Although you cannot change the value of a String object, such ... a string to 'z'”. This was done intentionally to make the implementation of the ... This function returns true if this string is equal to the specified string a_string. It returns false if this string is not equal ... true equals IgnoreCase (a_string) boolean b = s.equals("java"); // b = true indexOf (a_string) int i = s.indexOf("va"); ... the index of the first occurrence of the substring a_string within this string or −1 if a_string is not found. Index ... double = count / 25; The decrement operator is similar, except that it subtracts 1 rather than adds 1 to ... an array of one character per element and the last one of the array is the character whose value is the last character in the ... operator, as we have done in our examples. Parentheses and Precedence Rules Parentheses can be used to group terms in an expression so that you can specify the order of evaluation. For example, $result = (5 + 3) * 2$; This expression evaluates $5 + 3$ first and then multiplies the result by 2. $result = 16$. What is the output produced by the following lines of program code in question 17? 

```java
String s1 = "00123456789001234567890123";
int num1 = s1.indexOf('0');
int num2 = s1.indexOf('1');
int num3 = s1.indexOf('2');
int num4 = s1.indexOf('3');
int num5 = s1.indexOf('4');
int num6 = s1.indexOf('5');
int num7 = s1.indexOf('6');
int num8 = s1.indexOf('7');
int num9 = s1.indexOf('8');
int num10 = s1.indexOf('9');
```

4. Output: Display the results. This structure, which we can abbreviate as PIZO, is particularly true of our ... as you design and code your programs. ■ Self-Test Question 17. Consider the following statement from the ... quarters = amount / 25; amount = amount % 25; We can treat dimes and nickels in a similar way, so we get the
program uses the method `JOptionPane.showInputDialog` to get input—as in Listing 2.11—the user must enter the input in the form of a text box. For example, if you ask the user for the radius of a circle, the method expects an integer to be entered, and the user enters 2,000 instead of 2000, your program will crash, because integers in Java are stored as primitive data types, not as objects. To be safe, you should always check the input value to ensure it is of the correct type.

`JOptionPane` is a standard, predefined class that comes with every installation of Java. To make it available to your program, you must import the class at the beginning of your program using the statement `import javax.swing.JOptionPane;`. This statement tells the computer where to find the definition of the `JOptionPane` class. You may recall that we used this statement in previous chapters.

Recap

- **The setSize Method**
  - The `setSize` method resizes a `JFrame` object. The `setSize` method is defined as `public void setSize(int x, int y)`. The parameters `x` and `y` specify the position of the `JFrame` on the screen.

- **The setLocationRelativeTo Method**
  - The `setLocationRelativeTo` method sets the location of a `JFrame` object relative to another `JFrame` object. The method signature is `public void setLocationRelativeTo(JFrame other)`. You can use this method to position your `JFrame` close to a specific location on the screen.

- **The setVisible Method**
  - The `setVisible` method sets the visibility of a `JFrame` object. The method signature is `public void setVisible(boolean b)`. If `b` is `true`, the `JFrame` is visible, and if `b` is `false`, the `JFrame` is hidden.

- **Using the `System.out.println` Method**
  - The `System.out.println` method is used to print output to the console. The method signature is `public void println(String arg)`. You can use this method to display text on the command line.

- **Using the `System.out.format` Method**
  - The `System.out.format` method is used to format output. The method signature is `public void format(String format, Object... objects)`. This method allows you to format your output using specific format specifiers.

### Self-Test Questions

1. Which of the following is used to display output to the console?
   - `System.out.println`
   - `System.out.format`

2. Which of the following is used to format output?
   - `System.out.println`
   - `System.out.format`

3. Which of the following is used to print an integer in a field of width six?
   - `System.out.println`
   - `System.out.format`

4. Which of the following is used to print a floating-point number in exponential format?
   - `System.out.println`
   - `System.out.format`

5. Which of the following is used to print a character string formatted to a field of 10 spaces?
   - `System.out.println`
   - `System.out.format`

6. Which of the following is used to print a floating-point number with 2 digits after the decimal?
   - `System.out.println`
   - `System.out.format`

7. Which of the following is used to print an integer in a field of 5 spaces?
   - `System.out.println`
   - `System.out.format`

### Output

```
Enter the radius of a circle in inches: 2.5
A circle of radius 2.5 inches has an area of 19.6349375 square inches.
```

### Constants

- `Math.PI`: Represents the mathematical constant pi.
- `Math.E`: Represents the mathematical constant e.

### Comments

- Use consistent indentation to show a level of nesting. For example, one part starts with `public static void main(String[] args) {` The body of this main method begins on the next line and is indented by a consistent number of spaces. The program in Listing 2.7 has three levels of indentation.
- If the last statement on a continuation line contains a symbol such as `*`, use the symbol that appears at the end of the previous line to continue the line. In the case of the `for` loop in Listing 2.7, the symbol `*` is used to indicate that the loop should continue on the next line.
- Don't end a line with a `*` symbol. Use the appropriate symbol to end the line and then start a new line with the indicated symbol. This inconsistency makes the code difficult to read.
- Do not end a line with a `*` symbol. Use the appropriate symbol to end the line and then start a new line with the indicated symbol. This inconsistency makes the code difficult to read.

### Self-Test Questions

1. Which of the following is used to display output to the console?
   - `System.out.println`
   - `System.out.format`

2. Which of the following is used to format output?
   - `System.out.println`
   - `System.out.format`

3. Which of the following is used to print an integer in a field of width six?
   - `System.out.println`
   - `System.out.format`

4. Which of the following is used to print a floating-point number in exponential format?
   - `System.out.println`
   - `System.out.format`

5. Which of the following is used to print a character string formatted to a field of 10 spaces?
   - `System.out.println`
   - `System.out.format`

6. Which of the following is used to print a floating-point number with 2 digits after the decimal?
   - `System.out.println`
   - `System.out.format`

7. Which of the following is used to print an integer in a field of 5 spaces?
   - `System.out.println`
   - `System.out.format`

### Output

```
Enter the radius of a circle in inches: 2.5
A circle of radius 2.5 inches has an area of 19.6349375 square inches.
```

### Constants

- `Math.PI`: Represents the mathematical constant pi.
- `Math.E`: Represents the mathematical constant e.

### Comments

- Use consistent indentation to show a level of nesting. For example, one part starts with `public static void main(String[] args) {` The body of this main method begins on the next line and is indented by a consistent number of spaces. The program in Listing 2.7 has three levels of indentation.
- If the last statement on a continuation line contains a symbol such as `*`, use the symbol that appears at the end of the previous line to continue the line. In the case of the `for` loop in Listing 2.7, the symbol `*` is used to indicate that the loop should continue on the next line.
- Don't end a line with a `*` symbol. Use the appropriate symbol to end the line and then start a new line with the indicated symbol. This inconsistency makes the code difficult to read.
- Do not end a line with a `*` symbol. Use the appropriate symbol to end the line and then start a new line with the indicated symbol. This inconsistency makes the code difficult to read.

### Self-Test Questions

1. Which of the following is used to display output to the console?
   - `System.out.println`
   - `System.out.format`

2. Which of the following is used to format output?
   - `System.out.println`
   - `System.out.format`

3. Which of the following is used to print an integer in a field of width six?
   - `System.out.println`
   - `System.out.format`

4. Which of the following is used to print a floating-point number in exponential format?
   - `System.out.println`
   - `System.out.format`

5. Which of the following is used to print a character string formatted to a field of 10 spaces?
   - `System.out.println`
   - `System.out.format`

6. Which of the following is used to print a floating-point number with 2 digits after the decimal?
   - `System.out.println`
   - `System.out.format`

7. Which of the following is used to print an integer in a field of 5 spaces?
   - `System.out.println`
   - `System.out.format`

### Output

```
Enter the radius of a circle in inches: 2.5
A circle of radius 2.5 inches has an area of 19.6349375 square inches.
```

### Constants

- `Math.PI`: Represents the mathematical constant pi.
- `Math.E`: Represents the mathematical constant e.

### Comments

- Use consistent indentation to show a level of nesting. For example, one part starts with `public static void main(String[] args) {` The body of this main method begins on the next line and is indented by a consistent number of spaces. The program in Listing 2.7 has three levels of indentation.
- If the last statement on a continuation line contains a symbol such as `*`, use the symbol that appears at the end of the previous line to continue the line. In the case of the `for` loop in Listing 2.7, the symbol `*` is used to indicate that the loop should continue on the next line.
- Don't end a line with a `*` symbol. Use the appropriate symbol to end the line and then start a new line with the indicated symbol. This inconsistency makes the code difficult to read.
- Do not end a line with a `*` symbol. Use the appropriate symbol to end the line and then start a new line with the indicated symbol. This inconsistency makes the code difficult to read.

### Self-Test Questions

1. Which of the following is used to display output to the console?
   - `System.out.println`
   - `System.out.format`

2. Which of the following is used to format output?
   - `System.out.println`
   - `System.out.format`

3. Which of the following is used to print an integer in a field of width six?
   - `System.out.println`
   - `System.out.format`

4. Which of the following is used to print a floating-point number in exponential format?
   - `System.out.println`
   - `System.out.format`

5. Which of the following is used to print a character string formatted to a field of 10 spaces?
   - `System.out.println`
   - `System.out.format`

6. Which of the following is used to print a floating-point number with 2 digits after the decimal?
   - `System.out.println`
   - `System.out.format`

7. Which of the following is used to print an integer in a field of 5 spaces?
   - `System.out.println`
   - `System.out.format`
String s1 = "Hello"; String s2 = "hello";

String s3 = "Java";

Listing 3.2 also demonstrates the String method `equalsIgnoreCase`. This method returns `true` if the two strings contain equal values ignoring the case of the characters. For example, to check if `s1`, `s2` and `s3` are equal:

```java
if (s1.equals(s2)) // false
if (s1.equalsIgnoreCase(s2)) // true
if (s1.equalsIgnoreCase(s3)) // false
```

**Boolean AND** can also be used in apps. For example, the boolean expression `s1.equals(s2) && s3.equals(s3)` will return `true` if both strings are equal. A compound statement is a statement that contains other statements. For example, the following compound statement contains two statements:

```java
if (balance >= 0) {
    System.out.println("Good for you. You earned interest.");
    balance = balance + INTEREST_RATE * balance;
} else {
    System.out.println("You have a negative balance.");
    balance = balance - (-balance);
}
```

The compound statement contains two statements, the first one when there is a positive balance and the second one when the balance is negative. When the user enters a balance value, a different statement will be executed. For example, if the user enters a positive balance, the program will print a message and update the balance. If the user enters a negative balance, the program will print a message and calculate the interest charged on the account. The compound statement can be used in many different scenarios, such as handling exceptions or performing complex calculations.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
    System.out.println("Number is greater than 5.");
} else {
    System.out.println("Number is greater than 5.");
    System.out.println("Number is less than or equal to 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.

```java
if (number <= 5) {
    System.out.println("Number is less than or equal to 5.");
} else {
    System.out.println("Number is greater than 5.");
}
```

In this example, the program prints a message depending on whether the number is less than or equal to 5 or greater than 5. This is a simple example of a compound statement with two statements.
would be on top of the eyes, nose, M03_SAVI1833_07_SE_C03.indd 219 20/05/14 3:12 PM 220 Chapter 3 /  Flow of Control: ... setColor Method Color.BLACK Color.BLUE Color.CYAN Color.DARK_GRAY Color.GRAY Color.GREEN Color.LIGHT_GRAY The order in
[34x206]no questions. We did not present this window in Chapter 2, because you need the branching mechanisms introduced in this ... material about JOptionPane, you can skip the description of this dialog window. None of the JOptionPane material in this
[34x468]System.out.println("Balcony."); price = 15.00; break; default: System.out.println("Unknown ticket code."); break; } ... statement and it executes two cases when you expect it to execute only one case, you probably have forgotten to include a
[34x564]for example, you wanted to take the same action for the values 1 through 4, you would need to write a case for each ... a switch statement. The controlling expression in a switch statement can be more complicated than a single variable; for
[34x592]System.out.println("Grade A"); break; case 'C': case 'c': System.out.println("Grade C"); break; default: ... A and grade C eggs can be indicated by either uppercase or lowercase letters. Other values of eggGrade are handled by the
[34x813]Java program, the constants true or false must be spelled with all lowercase letters, but the input method nextBoolean is ... with the user: false Enter a boolean value: true You entered true As you can see from this example, the class Scanner has ... an expression is called short-circuit evaluation, or sometimes lazy evaluation, and is what Java does with expressions
[34x1047]adding parentheses, as follows: score 90 Of the remaining operators in the expression, the subtraction operator has the ... clarify operator precedence in a boolean expression score 90 Of the remaining operators in the expression, the
[34x1103]precedence and performs the operation with the higher precedence before performing the other one. Some operators have ... Binary operators of equal precedence are performed in left-to-right order. Unary operators of equal precedence are
[34x1185]might change. Such is more likely to be the case in the context of a loop, which we will study in the next chapter. More ... our opening example that checks whether we are ready for a launch would be preceded by a statement such as boolean
[34x1310]Calculation Program (part 2 of 2) Sample Screen Output Enter your weight in pounds. 150 Enter your height in feet and inches. 65 Enter your age in years. 32 The BMI is 24.93. You are in the overweight category. Your ... category is Overweight. S e l f -Te s t Q u e s t i o n s 8. What output is produced by the following code? int time = 2,
[34x1448]know that the BMI is not less than 18.5. In other words, the BMI is greater than or equal to 18.5. This makes the check ... applies to line 10. Consequently, we can change lines 9 and 10 of the algorithm to 9. Else if BMI< 25 then output
[34x1599]M03_SAVI1833_07_SE_C03.indd 196 20/05/14 3:12 PM 3.1 The if-else Statement 197 Figure 3.8  The Semantics of a Multibranch ... else if (Boolean_Expression_2) Action_2 . . . else if (Boolean_Expression_n) Evaluate Boolean_ Expression_1 True Action_n
[34x1793]== 0) System.out.println("Zero balance"); An equivalent, clearer, and preferred way to write this statement is as ... 0) System.out.println("Negative balance"); else if (balance < 0) System.out.println("Negative balance"); else if (balance
[34x1944]interest."); } else balance = balance − OVERDRAWN_PENALTY; In this case, the braces are an aid to clarity but are not, ... PM 3.1 The if-else Statement 193 In other cases, braces are needed. If we omit an else, for example, things get a bit
[34x1986]statement can contain any sort of statements within it. In particular, you can nest one if-else statement within another ... statement can be nested in another if-else statement if (balance >= 0) if (INTEREST_RATE >= 0) balance = balance +
[34x2096]before using the method compareTo to compare the strings. You can use either of the String methods toUpperCase or ... would compare lexicographic order instead of alphabetic order. ■ S e l f -Te s t Q u e s t i o n s 1. Suppose goals is a
[34x2207]the method call s1.compareTo(s2) compares the lexicographic ordering of the two strings and returns • A negative number ... if s1 comes before s2 Thus, the boolean expression s1.compareTo(s2) < 0 is true if s1 comes before s2 in lexicographic
[34x2234]compareTo, which we described in Figure 2.5 of Chapter 2. The method compareTo tests two strings to determine their lexicographic order. Lecxicographic order is similar to alphabetical order and is sometimes, but not always, the same as lexicographic order. In lexicographic ordering, the letters and other characters are compared lexicographically and the string with the lower value wins. If two strings have the same value for the first character, the method will call its compareTo() method to compare the lexicographic ordering of the two strings. and as a result the two strings are equal. If s1.compareTo (s2) returns 0, the two strings are equal. If s1.compareTo(s2) returns a positive number, s1 is lexicographically greater than s2, and s2 is lexicographically less than s1. If s1.compareTo(s2) returns a negative number, s1 is lexicographically less than s2, and s2 is lexicographically greater than s1. If s1.compareTo(s2) returns 0, the two strings are equal. If s1.compareTo(s2) returns a positive number, s1 is lexicographically greater than s2, and s2 is lexicographically less than s1. If s1.compareTo(s2) returns a negative number, s1 is lexicographically less than s2, and s2 is lexicographically greater than s1. Compare two strings lexicographically. Return a positive number if the first string is lexicographically greater than the second string; return a negative number if the first string is lexicographically less than the second string; return 0 if the two strings are equal. A String object s1 compares lexicographically to another String object s2 if s1.compareTo(s2) is less than zero. A String object s1 compares lexicographically to another String object s2 if s1.compareTo(s2) is greater than zero. The order in
[950x1075]s +, Fourth: the boolean operators , = Fifth: the boolean operators ==, != Sixth: the boolean operator & Seventh: the boolean operator |
A hacker can damage the clone, but the private data will not be affected. A brief introduction to cloning is given in ... may wish to look at that appendix. Do not get the impression that instance variables of a class type are a bad idea. They

```java
PetPair(goodDog, buddy); System.out.println("Our pair:"); pair.writeOutput( ); Pet badGuy = pair.getFirst(); buddy = pair.getSecond(); System.out.println("The pet wasn't so private!"); System.out.println("Looks like a security breach."); }
```
public class Vehicle { ...

Do Practice Program 5 from Chapter 5 but add a...
```java
private void loadSpecies() throws IOException
{
    try {
        inputStream = new ObjectInputStream(new FileInputStream("species.records"));
        while (true) {
            readTwo = (Species) inputStream.readObject();
            inputStream.close();
        }
    } catch (IOException e) {
        System.out.println("Error opening input file "+ fileName + ".");
        System.exit(0);
    }
}
```

```java
public static void main(String[] args) throws IOException
{
    Scanner keyboard = new Scanner(System.in);
    int number = keyboard.nextInt();
    int i = 0;
    while (number >= 0) {
        list[i] = number;
        i++;
    }
}
```

```java
public class ArrayOfTemperatures2 {
    public static void main(String[] args) throws IOException
    {
        Scanner keyboard = new Scanner(System.in);
        int size = keyboard.nextInt();
        double[] temperature = new double[size];
        // Read temperatures and compute their average:
        System.out.println("Enter "+ size + " temperatures:");
        for (int index = 0; index < temperature.length; index++)
        {
            temperature[index] = keyboard.nextDouble();
            System.out.print(temperature[index] + " ");
            sum = sum + temperature[index];
        }
        double average = sum / temperature.length;
        System.out.println("The average temperature is " + average);
    }
}
```

```java
public class Species {
    private String name;
    private int animal;
    private double weight;
    private String species;
    private int id;
    private String gender;
    private char age;
    private String color;  
    public Species() {
        id = 0;
    }
    public Species(String name, int animal, double weight, String species, int id, String gender, char age, String color) {
        this.name = name;
        this.animal = animal;
        this.weight = weight;
        this.species = species;
        this.id = id;
        this.gender = gender;
        this.age = age;
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public int getAnimal() {
        return animal;
    }
    public double getWeight() {
        return weight;
    }
    public String getSpecies() {
        return species;
    }
    public int getId() {
        return id;
    }
    public String getGender() {
        return gender;
    }
    public char getAge() {
        return age;
    }
    public String getColor() {
        return color;
    }
    public Species clone() {
        return (Species) super.clone();
    }
}
```

```java
public class SpeciesDB {
    private static Species[] entries = new Species[20];
    public static void main(String[] args) throws IOException
    {
        Scanner keyboard = new Scanner(System.in);
        int size = keyboard.nextInt();
        double[] temperature = new double[size];
        // Read temperatures and compute their average:
        System.out.println("Enter "+ size + " temperatures:");
        for (int index = 0; index < temperature.length; index++)
        {
            temperature[index] = keyboard.nextDouble();
            System.out.print(temperature[index] + " ");
            sum = sum + temperature[index];
        }
        double average = sum / temperature.length;
        System.out.println("The average temperature is " + average);
    }
}
```

```java
public class SpeciesDB {
    private static Species[] entries = new Species[20];
    public static void main(String[] args) throws IOException
    {
        Scanner keyboard = new Scanner(System.in);
        int size = keyboard.nextInt();
        double[] temperature = new double[size];
        // Read temperatures and compute their average:
        System.out.println("Enter "+ size + " temperatures:");
        for (int index = 0; index < temperature.length; index++)
        {
            temperature[index] = keyboard.nextDouble();
            System.out.print(temperature[index] + " ");
            sum = sum + temperature[index];
        }
        double average = sum / temperature.length;
        System.out.println("The average temperature is " + average);
    }
}
```

```java
public class SpeciesDB {
    private static Species[] entries = new Species[20];
    public static void main(String[] args) throws IOException
    {
        Scanner keyboard = new Scanner(System.in);
        int size = keyboard.nextInt();
        double[] temperature = new double[size];
        // Read temperatures and compute their average:
        System.out.println("Enter "+ size + " temperatures:");
        for (int index = 0; index < temperature.length; index++)
        {
            temperature[index] = keyboard.nextDouble();
            System.out.print(temperature[index] + " ");
            sum = sum + temperature[index];
        }
        double average = sum / temperature.length;
        System.out.println("The average temperature is " + average);
    }
}
```

```java
public class SpeciesDB {
    private static Species[] entries = new Species[20];
    public static void main(String[] args) throws IOException
    {
        Scanner keyboard = new Scanner(System.in);
        int size = keyboard.nextInt();
        double[] temperature = new double[size];
        // Read temperatures and compute their average:
        System.out.println("Enter "+ size + " temperatures:");
        for (int index = 0; index < temperature.length; index++)
        {
            temperature[index] = keyboard.nextDouble();
            System.out.print(temperature[index] + " ");
            sum = sum + temperature[index];
        }
        double average = sum / temperature.length;
        System.out.println("The average temperature is " + average);
    }
}
```

```java
public class SpeciesDB {
    private static Species[] entries = new Species[20];
    public static void main(String[] args) throws IOException
    {
        Scanner keyboard = new Scanner(System.in);
        int size = keyboard.nextInt();
        double[] temperature = new double[size];
        // Read temperatures and compute their average:
        System.out.println("Enter "+ size + " temperatures:");
        for (int index = 0; index < temperature.length; index++)
        {
            temperature[index] = keyboard.nextDouble();
            System.out.print(temperature[index] + " ");
            sum = sum + temperature[index];
        }
        double average = sum / temperature.length;
        System.out.println("The average temperature is " + average);
    }
}
```
1. a String object is an object that contains a sequence of characters. The class String is the class for objects that represent strings. String objects are created with the constructor String(String s), which creates a new String object containing the same sequence of characters as the string literal s.


3. char key = 'a';

4. double speciesMass = 2.0;

5. int length = 10;

6. int sum = 0;

7. boolean isTrue = true;

8. char[] characters = {'a', 'b', 'c'};


10. if (species == "human")

11. double speciesAge = 30.0;


example, the following statements effectively double the size of our array:

```java
OrderItem[] orderItems = new OrderItem[100];
orderItems = Arrays.copyOf(orderItems, orderItems.length * 2);
```

Doubling the size of an array is a common practice, particularly when dealing with dynamic data structures. However, it's important to consider the implications, such as the potential for increased memory usage.

In the context of a mail-order house, storing customer orders is a critical task. Suppose you are designing a program to record customer orders, and you need to store information about each order item. You could ask the user for the number of items in an order, store the number in a variable, and use that variable to create an array of `OrderItem` objects. This approach allows for efficient storage and retrieval of order item information.

### Arrays as Data Structures

**Section 12.3: Exception Handling with Linked lists**

Chapters 1 through 7 and Chapter 9 are devoted to the study of exception handling with linked lists. Chapter 12.4 covers array-based data structures, and Chapter 12.5 discusses recursion. The text mentions that recursion can be a versatile tool, allowing for the definition of a method in terms of itself. However, it's essential to design such recursive algorithms carefully to avoid infinite recursion.

#### Recursive Methods and Trees

Recursion is a powerful method in computer science, particularly in the context of trees. For example, a binary search tree can be recursively traversed using methods like `inOrder`, `preOrder`, and `postOrder`. These methods work by recursively visiting the left subtree, then the root node, and finally the right subtree. This approach efficiently explores the tree structure and allows for tasks such as searching, inserting, and deleting nodes.

#### Example: Merge Sort

Listing 11.8 gives the complete Java implementation of the merge sort algorithm, including the method `merge`. Listing 11.9 contains the `mergeSort` method that demonstrates how to use `merge` to sort an array. The code snippet provided shows how the algorithm works by dividing the array into smaller subarrays, sorting them, and then merging them back.

#### Programming Projects

Programming Projects require the implementation of methods that solve specific problems. Projects like `countIntegers`, `sumIntegers`, and `cumulativeWages` require recursion. For instance, the `cumulativeWages` function calculates the cumulative sum of salaries from a given list of salaries. The recursive method `cumulativeWages` calls itself repeatedly to compute the sum, starting from the first salary and adding each subsequent salary to the total.

#### Recursion in Practice

The text notes that recursion can be a versatile tool, allowing for the definition of a method in terms of itself. However, it's essential to design such recursive algorithms carefully to avoid infinite recursion. The text provides guidelines for designing recursive algorithms, such as using a base case and a recursive case. It also mentions that recursion can be a useful tool for solving problems that can be broken down into smaller subproblems, such as sorting algorithms like merge sort.
12.3 Linked Data Structures

Listing 12.9 A Linked List with an Iterator (part 2 of 4)

```java
/** Sets iterator to begin at the start of the list. */
public void setStart() {
    current = head;
}

/** Advances iterator to the next element. */
public void next() {
    current = current.next;
}

/** Advances iterator to the previous element. */
public void previous() {
    current = current.prev;
}

/** Returns iterator to the start of the list. */
public void reset() {
    current = head;
}

/** Returns iterator to one node after the start of a list. */
public void skipTo(int index)
```
HashMap students = new HashMap(); After all data is input, iterate through the map and output the student ID and all ... list of classes organized by student ID. M12_SAVI1833_07_SE_C12.indd 969 20/05/14 5:11 PM 970 Chapter 12 /  Dynamic Data reference, which is to the list's last node. 13. Suppose that we would like to perform a bird survey to count the number ... linked-list classes given in this chapter. (The linked list you use will affect what your new class can do, so give some

... a data structure known as a queue. A queue can be thought of as a line. Items are added at the end of the line and are removed from the head. It is very useful for building data structures to store connections, such as in a phone book. A queue can be implemented using a linked list. Another common and

... are several more linked data structures that you will surely encounter in later studies. A stack is an ADT whose ... A stack organizes its data so that items are removed in the reverse of the order in which they were inserted. So if you

... link = linkValue; } } Because an object of any class type is also of type Object, you can store any kinds of objects in a linked list. A linked list is superior to an array for this reason. An ArrayList is primeval but is not tailored to this particular application. By using a generic data type, you can create a class like this:

public class StringLinkedListWithIterator { private StringNode head; public StringLinkedListWithIterator() { head = null; } public String getRepresentative() { String value = ""; for (StringNode current = head; current != null; current = current.link) { if (current.data[0] == 'a') { value = current.data; } break; } return value; } public int position() { int count = 0; for (StringNode current = head; current != null; current = current.link) { if (current.data[0] == 'a') { return count; } count++; } return count; } public void addANodeToStart(String addData) { StringNode newNode = new StringNode(addData); newNode.link = head; head = newNode; } public void addANodeToEnd(String addData) { StringNode newNode = new StringNode(addData); if (head == null) { head = newNode; } else { StringNode current = head; while (current.link != null) { current = current.link; } current.link = newNode; } } public void deleteANode() { if (head == null) { System.out.println("List is empty."); return; } StringNode current = head; head = head.link; } public String toString() { String result = "["; StringNode current = head; while (current != null) { result += current.data + ", " + current.position + ""; current = current.link; } result += "]"; return result; } public Object getRepresentative() { throw new IllegalArgumentException("List is empty."); } public int position() { throw new IllegalArgumentException("List is empty."); } public void addANodeToStart(String addData) { throw new IllegalArgumentException("List is empty."); } public void addANodeToEnd(String addData) { throw new IllegalArgumentException("List is empty."); } public void deleteANode() { throw new IllegalArgumentException("List is empty."); } public String toString() { throw new IllegalArgumentException("List is empty."); } }