Java, Java, Java
Object-Oriented Problem Solving
Third Edition
Ralph Morelli | Ralph Walde
Trinity College
Hartford, CT
published by Prentice Hall
Objectives

- Know the basic steps involved in program development.
- Understand some basic Java language elements.
- Know how to use simple output operations.
- Be able to identify different types of errors in a program.
- Understand how a Java program is translated into machine language.
- Understand the difference between a Java application and a Java applet.
- Know how to edit, compile, and run programs.

Outline

- Designing Good Programs
- Designing a Riddle Program
- Java Language Elements
- Editing, Compiling, and Running a Java Program
- From the Java Library: The System and PrintStream classes.
Designing Good Programs

- Always precede coding with careful design.
- Remember: The sooner you begin to type code, the longer the program will take to finish.
- Design includes designing classes, data, methods, and algorithms.
- Design is followed by coding, testing, and revision.

The Program Development Process

- Problem Specification
- Problem Decomposition
- Design Specification
- Data, Methods, and Algorithms
- Coding into Java
- Testing, Debugging, and Revising
Problem Specification

• What exactly is the problem to be solved?
• How will the program be used?
• How will the program behave?

**Problem Specification:** Design a class that represent a riddle with a given question and answer. The definition of this class should make it possible to store different riddles and retrieve a riddle’s question and answer independently.
Problem Decomposition

- Divide-and-Conquer: What objects will be used and how will they interact?
- Nouns: In OOD, choosing objects means looking for nouns in the problem specification.

**Problem Specification:** Design a class that represent a riddle with a given question and answer. The definition of this class should make it possible to store different riddles and retrieve a riddle’s question and answer independently.

Object Design Questions

- What role will the object perform?
- What data or information will it need?
  - Look for nouns.
- Which actions will it take?
  - Look for verbs.
- What interface will it present to other objects?
  - These are public methods.
- What information will it hide from other objects?
  - These are private.
Design Specification for a Riddle

- Class Name: Riddle
- Role: To represent a riddle
- Attributes (Information or instance variables)
  - Question: A variable to store the question (private)
  - Answer: A variable to store the answer (private)
- Behaviors (public methods)
  - Riddle(): A method to set a riddle’s question and answer
  - getQuestion(): A method to return a riddle’s question
  - getAnswer(): A method to return a riddle’s answer

UML Design Specification

Instance variables -- memory locations used for storing the information needed.

Hidden information

Public interface

Methods -- blocks of code used to perform a specific task.

Figure 1.3. A UML Class Diagram
### Method Design

- What specific task will the method perform?
- What information will it need to perform its task?
- What result will the method produce?
- What *algorithm* (step-by-step description of the solution) will the method use?

### Method Specification: `getQuestion()`

- Method Name: `getQuestion()`
- Task: To return the riddle’s question
- Information Needed (variables)
  - Question: A variable to store the riddle’s answer (private)
- Algorithm: return question
- (This is a *very* simple method!)
Coding into Java

```java
public class Riddle extends Object  // Class header
    {
    private String question;    // Instance variables
    private String answer;

    public Riddle(String q, String a) // Constructor method
    {
        question = q;
        answer = a;
    }

    public String getQuestion()       // Access method
    {
        return question;
    } // getQuestion()

    public String getAnswer()         // Access method
    {
        return answer;
    } // getAnswer()
    } // Riddle class
```

Syntax

- The **syntax** of a programming language is the set of rules that determines whether its statements are correctly formulated.
- Example Rule: All Java statements must end with a semicolon.
- **Syntax error**: `question = q`
- Syntax errors can be detected by the compiler, which will report an error message.
Semantics

- The **semantics** of a programming language is the set of rules that determine the meaning of its statements.
- Example Rule: In $a + b$, the + operator will add $a$ and $b$.
- **Semantic error**: User intended to add $a$ and $b$ but coded $a - b$.
- Semantic errors cannot be detected by the compiler, because it can’t read the programmer’s mind.

Java Language Elements

- A Java program is made up of **class definitions**.
- A class definition contains a **header** and a **body**.
- A **method** is a named section of code that can be called by its name.
- **Multi-line** and **single-line comments** are used to document the code.
Java Language Elements: Class Definition

```java
public class HelloWorld extends Object { // Class header
    private String greeting = "Hello World!"; // Instance variable

    public void greet() { // Method definition header
        System.out.println(greeting); // Output statement
    } // greet() // End of greet method body

    public static void main(String args[]) { // Method definition header
        HelloWorld helloworld; // Reference variable
        helloworld = new HelloWorld(); // Object instantiation stmt
        helloworld.greet(); // Method call
    } // main() // End of method body
} // HelloWorld // End of class body
```

Java Keywords

- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- final
- finally
- float
- for
- goto
- if
- implements
- import
- instanceof
- int
- interface
- instanceof
- interface
- long
- native
- new
- package
- private
- protected
- public
- return
- short
- strictfp
- static
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- try
- void
- volatile
- while

Copyright 2000 by Ralph Morelli. All rights reserved.
Java Language Elements: Identifiers

- An **identifier** must begin with a letter (A to Z, a to z) and may be followed by any number of letters or digits (0 to 9) or underscores (_). An identifier may not be identical to a Java keyword.
- Style: Class names begin with a capital letter and use capitals to distinguish words within the name: `HelloWord`, `TextField`
- Style: Variable and method names begin with a lowercase letter and use capitals to distinguish words within the name: `main()`, `getQuestion()`

Data Types

- Java data are classified according to **data type**.
- Types of Objects (Riddle, Rectangle, String) versus **primitive data types** (int, double, boolean).
- A **literal value** is an actual value (“Hello”, 5, true).

<table>
<thead>
<tr>
<th>Type</th>
<th>Keyword</th>
<th>Size in Bits</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>boolean</td>
<td>-</td>
<td>true or false</td>
</tr>
<tr>
<td>character</td>
<td>char</td>
<td>16</td>
<td>‘A’, ‘5’, ‘4’</td>
</tr>
<tr>
<td>byte</td>
<td>byte</td>
<td>8</td>
<td>-128 to +127</td>
</tr>
<tr>
<td>integer</td>
<td>short</td>
<td>16</td>
<td>-32768 to +32767</td>
</tr>
<tr>
<td>integer</td>
<td>int</td>
<td>32</td>
<td>-2147483648 to +2147483647</td>
</tr>
<tr>
<td>integer</td>
<td>long</td>
<td>64</td>
<td>really big numbers</td>
</tr>
<tr>
<td>real number</td>
<td>float</td>
<td>32</td>
<td>21.3, -0.45, 1.67e28</td>
</tr>
<tr>
<td>real number</td>
<td>double</td>
<td>64</td>
<td>21.3, -0.45, 1.67e28</td>
</tr>
</tbody>
</table>

Java’s Primitive Data Types.
Variables

- A variable is a typed container that can store a value of a certain type.

```
num: int 5
```

Figure 1.6. A variable is a typed container.

Statements

- A statement is a segment of code that takes some action in a program.
- A declaration statement declares a variable and has the general form:

  ```
  Type VariableName;
  ```

```
// Type variableName;
HelloWorld helloworld;
int int1, int2;
```
Assignment Statements

- An assignment statement stores a value in a variable and has the general form:
  \[ \text{VariableName} = \text{Value} \; ; \]

- Figure 1.7

Expressions and Operators

- An expression is Java code that specifies or produces a value in the program.

- Expressions use operators (=, +, <, …)
  
  ```java
  num1 + num2 // An addition of type int
  num1 < num2 // A comparison of type boolean
  num = 7 // An assignment expression of type int
  square(7) // A method call expression of type int
  num == 7 // An equality expression of type boolean
  ```

- Expressions occur within statements:
  
  ```java
  num = square(7) ; // An assignment statement
  ```
Class Definition

- A Java program consists of one or more class definitions.
- A class definition contains a class header:
  \[ \textbf{ClassModifiers} \textbf{class} \textbf{ClassName} \textbf{Pedigree} \]

  ```java
  public class HelloWorld extends Object
  ```

- And a class body, which is code contained within curly brackets: `{…}

Declaring an Instance Variable

- In general an instance variable declaration takes the following form:
  \[ \textbf{Modifiers} \textbf{Type} \textbf{VariableName} \textbf{InitializerExpression} \]

  ```java
  private String greeting = "Hello World";
  int num;
  double realNum = 5.05;
  static int count = 0;
  ```

- In these examples the types are String, int, double and the variable names are greeting, num, realNum, and count.
Declaring an Instance Method

- A method definition consists of a **method header**: 
  \[\text{Modifiers}_{\text{opt}} \text{ReturnType MethodName} (\text{ParameterList}_{\text{opt}})\]
- Followed by a **method body**, which is executable code contained within curly brackets: \{…\}

```java
public void greet() // Method definition header
{  // Start of method body
    System.out.println(greeting); // Executable statement
} // greet() // End of greet method body
```

```java
public static void main(String args[]) // Method definition header
{  HelloWorld helloworld;         // Reference variable
    helloworld = new HelloWorld(); // Object instantiation stmt
    helloworld.greet();            // Method call
} // main()
```

Applications vs. Applets

<table>
<thead>
<tr>
<th><strong>Java Applications</strong></th>
<th><strong>Java Applets</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Stand-alone program</td>
<td>Embedded program.</td>
</tr>
<tr>
<td>Runs independently</td>
<td>Runs in a Web browser</td>
</tr>
<tr>
<td>Has a main() method</td>
<td>No main() method.</td>
</tr>
<tr>
<td>No HTML file</td>
<td>Requires an HTML file</td>
</tr>
<tr>
<td>Run using JDK’s java interpreter</td>
<td>Run using JDK’s appletviewer</td>
</tr>
</tbody>
</table>

Copyright 2000 by Ralph Morelli. All rights reserved.
The HelloWorld Application

```java
public class HelloWorld extends Object { // Class header
    private String greeting = "Hello World!"; // Instance variable

    public void greet() { // Method definition header
        System.out.println(greeting); // Output statement
    } // greet() // End of greet method body

    public static void main(String args[]) { // Method definition header
        HelloWorld helloworld; // Reference variable
        helloworld = new HelloWorld(); // Object instantiation stmt
        helloworld.greet(); // Method call
    } // main() // End of method body
} // HelloWorld // End of class body
```

The HelloWorldApplet Program

```java
/**
 * HelloWorldApplet program
 */

import java.applet.Applet; // Import the Applet class
import java.awt.Graphics; // and the Graphics class

public class HelloWorldApplet extends Applet { // Class header
    public void paint(Graphics g) { // The paint method
        g.drawString("HelloWorld!",10,10); // End of paint
    } // End of paint
} // End of HelloWorld
```

These statements import Java class names.

This statement displays “HelloWorld!” on the browser window.

Copyright 2000 by Ralph Morelli. All rights reserved.
Qualified Names

• A qualified name takes the form
  
  reference.elementName

  where reference refers to some object (or class or package) and elementName is the name of one of the object’s (or class’s or package’s) elements.

• Use: To refer to elements in Java’s package, class, element hierarchy.

• Context dependent.

  ```
  System.out.println(); // println() method in System.out class
  pet1.eat();           // eat() method in pet1 object
  java.awt.Button      // Button class in java.awt package
  ```
The Java Development Process

- Step 1: Editing the Program
  - Software: Any text editor will do.
- Step 2: Compiling the Program
  - Software: Java Development Kit (JDK)
  - JDK: `javac HelloWorld.java`
- Step 3: Running the Program
  - JDK: `java HelloWorld` (Application)
  - JDK: `appletviewer file.html` (Applet)

Editing a Java Program

- Software: A text editor (vi, emacs, BBEdit).
- Program source code must be saved in a text file named `HelloWorld.java` where `HelloWorld` is the name of the public class contained in the file.

- **Remember**: Java class names and file names are case sensitive.
Compiling a Java Program

• Compilation translates the source program into Java bytecode.
  – Bytecode is platform-independent
• JDK Command: javac HelloWorld.java
• Successful compilation will create the bytecode class file: HelloWorld.class

Running a Java Application

• The class file (bytecode) is loaded into memory and interpreted by the Java Virtual Machine (JVM)
• JDK Command: java HelloWorld
Running a Java Applet

- Running an applet requires an HTML file containing an `<applet>` tag:

```html
<HTML>
...<APPLET CODE="HelloWorldApplet.class"
WIDTH=200 HEIGHT=200>
</APPLET>
...</HTML>
```

- JDK Command: `appletviewer file.html`
- Browser: Open the applet’s HTML file

Running a HelloApplet

- Try running `HelloWorldApplet`
Coding into Java

- **Stepwise Refinement** is the right way to code.
  - Code small stages at a time, testing in between.
  - Errors are caught earlier.
- **Syntax** rules must be followed.
  - Syntax is the set of rules that determine whether a particular statement is correctly formulated.
- **Semantics** must be understood.
  - Semantics refers to the meaning (effect on the program) of each Java statement.

Testing, Debugging, and Revising

- Coding, testing, and revising a program is an iterative process.
- The java compiler catches *syntactic errors*, producing error messages.
- The programmer must test thoroughly for *semantic errors*.
  - Semantic errors are errors which manifest themselves through illogical output or behavior.
  - Errors are corrected in the debugging phase.
Writing Readable Programs

- Style, in addition to working code, is the mark of a good programmer. Style consists of:
  - *Readability*.
    - Code should be well-documented and easy to understand.
  - *Clarity*.
    - Conventions should be followed and convoluted code avoided.
  - *Flexibility*.
    - Code should be designed for easy maintenance and change.

Java Library: System and PrintStream

- The `java.lang.System` class contains `PrintStream` objects that perform Input/Output (I/O).

- The `java.lang.PrintStream` class contains the `print()` and `println()` methods that perform output.
Example: OldMacDonald Program

```java
public class OldMacDonald // Class header
{
    // Start of body
    public static void main(String argv[]) // Main method
    {
        System.out.println("Old MacDonald had a farm.");
        System.out.println("E I E I O.");
        System.out.println("And on his farm he had a duck.");
        System.out.println("E I E I O.");
        System.out.println("With a quack quack here.");
        System.out.println("And a quack quack there.");
        System.out.println("Here a quack, there a quack.");
        System.out.println("Everywhere a quack quack.");
        System.out.println("Old MacDonald had a farm");
        System.out.println("E I E I O.");
    } // End of main
} // End of OldMacDonald
```

Summary: Technical Terms

- algorithm
- applet
- application program
- assignment statement
- comment
- compound statement (block)
- data type
- declaration statement
- default constructor
- executable statement
- expression
- identifier
- keyword
- literal value
- object instantiation
- operator
- package
- parameter
- primitive data type
- pseudocode
- qualified name
- semantics
- stepwise refinement
- syntax
Summary: Key Points

- A Java applet is an embedded program that runs within the context of a WWW browser. Java applets are identified in HTML documents by using the `<applet>` tag.

- A Java application runs in stand-alone mode. Applications must have a `main()` method.

- Java programs are first compiled into bytecode and then interpreted by the Java Virtual Machine (JVM).

---

Summary: Key Points

- A Java source program must be stored in a file that has a `.java` extension.
- A Java bytecode file has the same name as the source file but a `.class` extension.
- The name of the source file must be identical to the name of the public `class` defined in the file.
- Java is case sensitive.
Summary: Key Points

- Good program design requires that each object and each method have a well-defined task.
- Coding Java should follow the *stepwise refinement* approach.
- A *stub method* is a method with a complete header and an incomplete body.

Summary: Key Points

- A *syntax error* results when a statement violates one of Java’s grammar rules.
- A *semantic error* or *logic error* is an error in the program’s design and cannot be detected by the compiler.
- Testing a program can only reveal the presence of bugs, not their absence.
- Good programs should be designed for readability, clarity, and flexibility.
Questions & Discussion