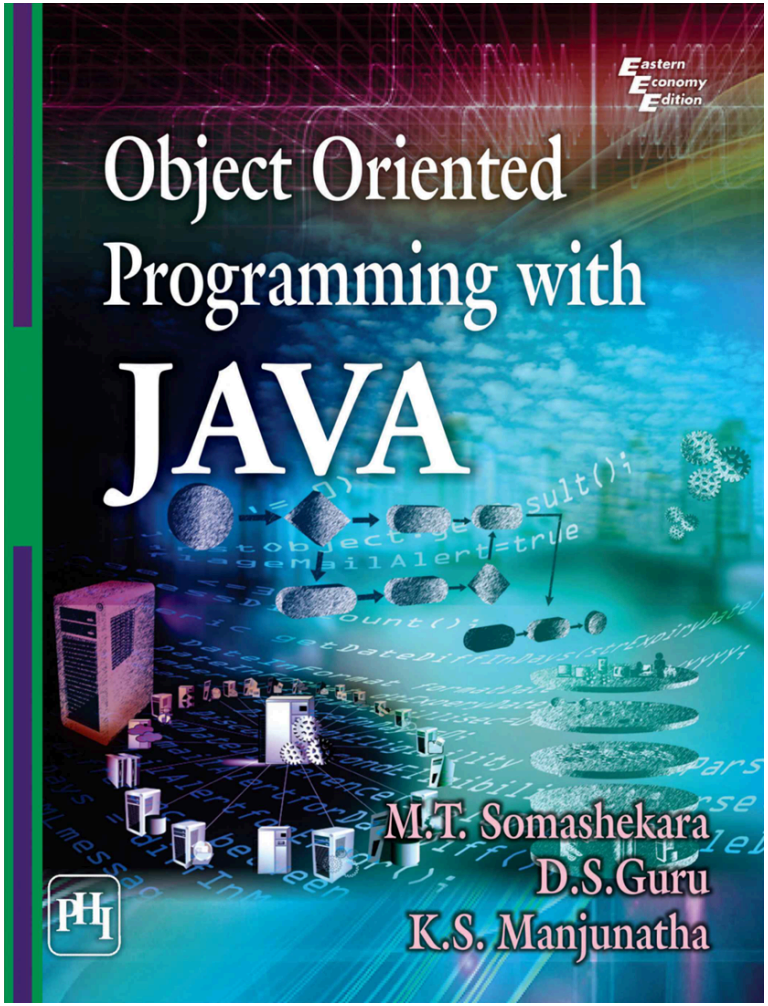
Chapter 1: Introduction to Computers, Programs, and Java™

1.1. Introduction

Textbook


Eastern Economy Edition
Object Oriented Programming with
JAVA
M.T. Somashekara
D.S.Guru
K.S. Manjunatha
PHI

- Welcome to the exciting world of programming!
- The central theme of this book is to help you learn how to solve problems by writing programs.
- Programming is the art of creating software, which is essentially a set of instructions that tell a computer or device what to do.
- This book will teach you how to create programs using the Java programming language.

Why Learn Programming?

- **Problem Solving**: Programming allows you to solve problems efficiently and creatively. You can automate repetitive tasks, analyze data, and build applications that make people's lives easier.
- **Creativity**: Programming is a creative process that allows you to express your ideas and build innovative solutions. You can create games, websites, mobile apps, and more.

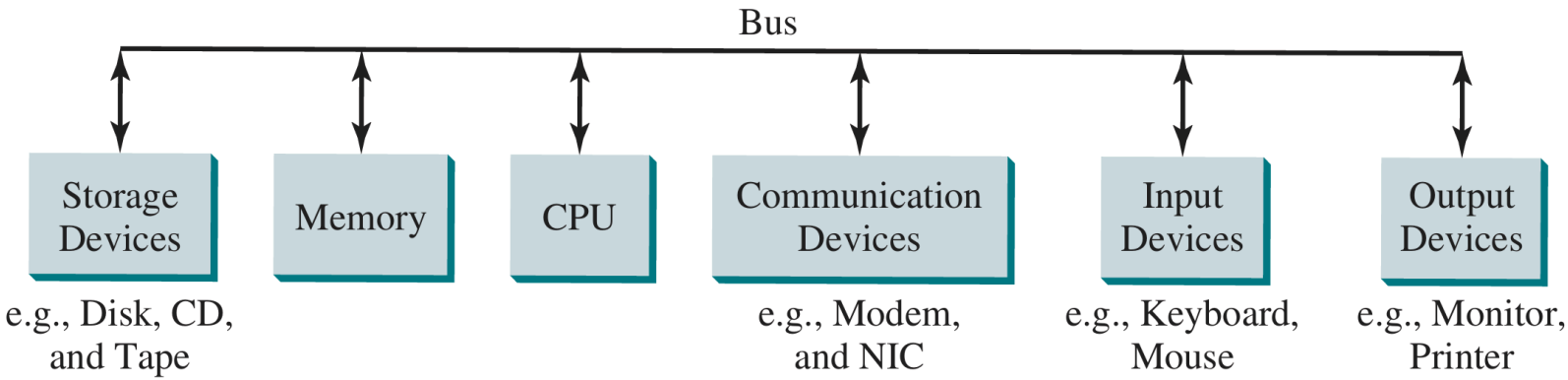
- **Career Opportunities**: Programming skills are in high demand in today's job market. Learning to program opens up a wide range of career opportunities in fields like software development, data science, artificial intelligence, and cybersecurity.
- **Empowerment**: Programming gives you the power to create tools and applications that can have a positive impact on the world. You can build software that helps people connect, learn, work, and play.

1.2. What is a Computer?

Definition

A computer is an electronic device that stores and processes data.

Architecture



Components

- **Hardware**: Physical components of the computer.
- **Software**: Programs that run on the computer.

Hardware Components

- **CPU (Central Processing Unit)**: The computer's brain; retrieves and executes instructions.
- **Memory (Main Memory)**: Stores data and programs for the CPU to access.
- **Storage Devices**: Devices like disks and CDs used for permanent data storage.
- **Input Devices**: Devices such as the mouse and keyboard that allow user interaction.

- **Output Devices**: Devices like monitors and printers that display results.
- **Communication Devices**: Devices like modems and network interface cards that enable data transfer.
- **Bus**: A subsystem that transfers data between computer components, similar to a system of roads.

Software Components

- **System Software**: Manages the computer's hardware and provides a platform for running applications.
- **Application Software**: Programs that perform specific tasks, such as word processing or web browsing.

1.3. Programming Languages

Definition

Programming languages are used to write instructions that tell a computer what to do. There are different types of programming languages, each with its own syntax and rules.

Types of Programming Languages

- **Machine Language**: The lowest-level programming language that uses binary code (0s and 1s) to represent instructions. It is specific to the computer's hardware and difficult to read and write.
- **Assembly Language**: A low-level language that uses mnemonics to represent machine language instructions. It is easier to read and write than machine language but still closely tied to the computer's hardware.
- **High-Level Language**: A programming language that is closer to human language and easier to read and write. Examples include Java, C++, Python, and JavaScript.

High-Level Languages

- **Source Program**: A program written in a high-level language.
- **Interpreter**: A programming tool that reads one statement from the source code, translates it into machine code, and executes it immediately.
- **Compiler**: A programming tool that translates the entire source code into a machine-code file, which is then executed.

1.4. Operating Systems

Definition

The operating system (OS) is the most important program that runs on a computer. It manages and controls a computer's activities.

Functions of an Operating System

- **Controlling and Monitoring System Activities**: The OS performs basic tasks like recognizing input from the keyboard, sending output to the monitor, and keeping track of files and folders.
- **Allocating and Assigning System Resources**: The OS determines what computer resources a program needs (CPU time, memory space, etc.) and allocates them.
- **Scheduling Operations**: The OS schedules programs' activities to make efficient use of system resources. Techniques like multiprogramming, multithreading, and multiprocessing help increase performance.

Popular Operating Systems

- **Microsoft Windows**: A widely used OS for personal computers.
- **Mac OS**: The operating system for Apple Macintosh computers.
- **Linux**: An open-source OS popular for servers and embedded systems.
- **Unix**: A powerful OS used in servers and workstations.
- **Android**: A mobile OS developed by Google.
- **iOS**: Apple's mobile OS for iPhones and iPads.
- **Chrome OS**: Google's OS for Chromebooks.

1.5. Java, the World Wide Web, and Beyond

What is Java?

Java is a popular programming language known for its portability, security, and versatility. It is widely used for developing web applications, mobile apps, and enterprise software.

History and Development

- Java was developed by a team led by James Gosling at Sun Microsystems, which was later acquired by Oracle.
- It was initially called Oak and was designed in 1991 for use in embedded chips in consumer electronic appliances.
- In 1995, it was renamed Java and redesigned for developing web applications.

Popularity and Characteristics

- Java promises "write once, run anywhere," making it a popular choice.
- It is simple, object-oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high-performance, multithreaded, and dynamic.

Usage and Applications

- Java is used for developing a wide range of applications, from desktop to server-side applications.
- It is also used in mobile devices, such as Android phones.
- Applets, which were once popular for web applications, are now less commonly used due to security issues, but Java remains popular for backend web development.

1.6. The Java Language Specification, API, JDK, JRE, and IDE

Java Language Specification

- The Java language syntax and semantics are defined in the Java language specification.
- The specification provides detailed rules for writing Java programs, including syntax, data types, control structures, and more.
- The Java language specification is maintained by Oracle and is available online.
- The specification ensures that Java programs are portable, secure, and reliable.

API (Application Programming Interface)

- The Java API is a collection of prewritten classes and interfaces that provide ready-to-use functionality for Java programs.
- The API includes classes for data structures, networking, file I/O, GUI development, and more.

JDK (Java Development Kit)

- The JDK is a software development kit that includes tools for developing Java applications.
- It includes the Java compiler (`javac`), the Java Virtual Machine (`java`), and other tools for compiling, running, and debugging Java programs.
- The JDK also includes the Java API and documentation.

JRE (Java Runtime Environment)

- The JRE is a runtime environment that allows Java programs to run on a computer.
- It includes the Java Virtual Machine (JVM) and the Java API.

IDE (Integrated Development Environment)

- An IDE is a software application that provides comprehensive facilities for software development.
- IDEs typically include a source code editor, build automation tools, and a debugger.

1.7. A Simple Java Program

Structure of a Java Program

Example:

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

Explanation:

- The program is saved in a file named `Welcome.java`.
- The program defines a class named `Welcome` with a `main` method that prints "Welcome to Java!" on the console.
- The `System.out.println` statement displays the message.

Class Definition

- Every Java program must have at least one class, and each class has a name.
- The class is defined using the `class` keyword.

Main Method

- The entry point for the program is the `main` method.
- The main method is defined as `public static void main(String[] args)`.

Statements and Syntax

- `System.out.println`: This method is used to display messages on the console. In the example, it prints "Welcome to Java!".
- `String`: A sequence of characters enclosed in double quotation marks.
- **Statement Terminator**: Every statement in Java ends with a semicolon (`;`).

Comments

- **Line Comments**: Use `//` to denote comments on a single line.
- **Block Comments**: Enclosed between `/*` and `*/`, used for comments spanning multiple lines.

Special Characters

TABLE 1.2 Special Characters

Character	*Name*	*Description*
`{}`	Opening and closing braces	Denote a block to enclose statements.
`()`	Opening and closing parentheses	Used with methods.
`[]`	Opening and closing brackets	Denote an array.
`//`	Double slashes	Precede a comment line.
`""`	Opening and closing quotation marks	Enclose a string (i.e., sequence of characters).
`;`	Semicolon	Mark the end of a statement.

Explanation:

- `{}` : Used to enclose blocks of code.
- `()` and `[]` : Used for method parameters and array indices, respectively.
- `;` : Statement terminator.
- `//` : Line comment.
- `/* */` : Block comment.

Keywords

Keywords are reserved words in Java that have specific meanings and cannot be used for other purposes.

List of 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`, `long`, `native`, `new`, `package`, `private`, `protected`, `public`, `return`, `short`, `static`, `strictfp`, `super`, `switch`, `synchronized`, `this`, `throw`, `throws`, `transient`, `try`, `void`, `volatile`, `while`, `true`, `false`, `null`, etc.

Case Sensitivity

- Java is case-sensitive, meaning that keywords, class names, and other identifiers must be used with consistent capitalization.

Example:

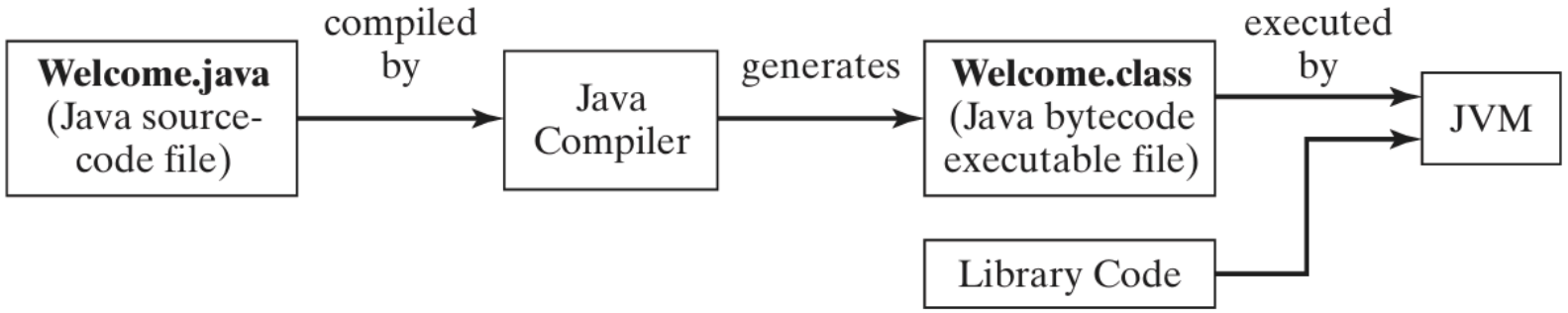
- `System.out.println` is different from `system.out.println`.
- `Welcome` is different from `welcome`.
- `main` is different from `Main`.
- `String` is different from `string`.
- `public` is different from `Public`.
- `class` is different from `Class`.
- `println` is different from `Println`.

Common Errors

- **Syntax Errors**: These occur when the program violates Java's syntax rules (e.g., missing semicolon, unmatched braces).
- **Compile Errors**: The compiler detects syntax errors and provides error messages indicating the problem.

1.8. Creating, Compiling, and Executing a Java Program

Architecture of a Java Program



Writing Source Code

Source code is written in a `.java` file using a text editor or an Integrated Development Environment (IDE).

Example: `Welcome.java`

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

Compiling

- `javac` is the Java compiler that translates Java source code into Java bytecode.
- Use the `javac` command to compile the source code into a bytecode file with a `.class` extension.
- The compiler checks the source code for syntax errors and generates bytecode if the code is error-free.

Example: `javac Welcome.java` compiles `Welcome.java` and creates `Welcome.class`.

Bytecode

- Java source code is translated into Java bytecode, a low-level language similar to machine instructions.
- Bytecode is platform-independent and can run on any system with a Java Virtual Machine (JVM).

Example: `Welcome.class` (Bytecode)

Running Bytecode

- `java` is the Java Virtual Machine (JVM) that executes Java bytecode.
- Use the `java` command to run the bytecode.
- The JVM loads the bytecode file and executes the program.

Example:

`java Welcome` runs the `Welcome.class` file.

- Bytecode is executed by the JVM, which interprets the individual instructions and translates them into the target machine language.

Handling Errors

- If compile errors occur, modify the source code and recompile.
- If runtime errors or incorrect results occur, modify the source code, recompile, and run the bytecode again.

JVM

- The JVM is a program that interprets Java bytecode.
- JVM dynamically loads necessary classes and verifies bytecode for security.

IDE

- **VS Code**: A lightweight code editor with support for Java development.
- **NetBeans**: An IDE with features like code completion, debugging, and project management.
- **Eclipse**: A popular Java IDE with a wide range of plugins and tools.
- **IntelliJ IDEA**: A powerful IDE with advanced features for Java development.

1.9. Programming Style and Documentation

Good Programming Style

- Ensure your programs are easy to read and understand.
- Use consistent formatting, including indentation and spacing.
- Align your code structure clearly, making the relationships between different parts obvious.

Appropriate Comments and Comment Styles

- Include a summary at the beginning of the program explaining what it does, its key features, and any unique techniques.
- Use comments to introduce each major step and explain complex code sections.
- Make comments concise and avoid cluttering the program.

Types of Comments

- **Line Comments**: Start with `//` for single-line comments.
- **Block Comments**: Enclosed between `/*` and `*/` for multi-line comments.
- **Javadoc Comments**: Begin with `/**` and end with `*/`, used for class-level and method-level comments.

Example: Line Comments

```java
// This program displays a welcome message.
public class Welcome {
    public static void main(String[] args) {
        // Display the message
        System.out.println("Welcome to Java!");
    }
}
```

Explanation:

- The line comment `// This program displays a welcome message.` provides a brief summary of the program.
- The line comment `// Display the message` explains the purpose of the `System.out.println` statement.

Example: Javadoc Comments

```java
/**
 * The Welcome class displays a welcome message.
 */
public class Welcome {
    /**
     * The main method prints the welcome message.
     */
    public static void main(String[] args) {
        // Display the message
        System.out.println("Welcome to Java!");
    }
}
```

Explanation:

- The Javadoc comment `/** ... */` provides a detailed description of the class and method.
- The Javadoc comment `/** ... */` is used for generating documentation.

Proper Indentation and Spacing

- Indentation helps illustrate the structural relationships between program components.
- Indent at least two spaces for each subcomponent.
- Add a single space on both sides of binary operators for readability.

Block Styles

- Blocks are groups of statements surrounded by braces `{}`.
- Two popular styles:
  - **Next-Line Style**: Align braces vertically.
  - **End-of-Line Style**: Place opening brace at the end of a line.
- Choose one style and use it consistently throughout your program.

Example: Next-Line Style

```
// Next-Line Style
public class Welcome
{
    public static void main(String[] args)
    {
        System.out.println("Welcome to Java!");
    }
}
```

Example: End-of-Line Style

```
// End-of-Line Style
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

1.10. Programming Errors

Types of Errors

Programming errors can be categorized into three types:

- **Syntax Errors**: Detected by the compiler and result from code construction mistakes like mistyped keywords, missing punctuation, or unmatched braces.
- **Runtime Errors**: Occur during program execution and cause abnormal termination. Typically caused by operations impossible to carry out, such as dividing by zero or incorrect input types.
- **Logic Errors**: Occur when a program doesn't perform as intended due to incorrect logic, such as using the wrong formula.

Example: Syntax Error

```
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!")
    }
}
```

Example: Runtime Error

```
public class Welcome {
    public static void main(String[] args) {
        int x = 5;
        int y = 0;
        int z = x / y; // Division by zero
        System.out.println(z);
    }
}
```

Example: Logic Error

```
public class Welcome {
    public static void main(String[] args) {
        int x = 5;
        int y = 3;
        int z = x - y; // Incorrect operation
        System.out.println(z);
    }
}
```

Detection and Fixing

- **Syntax Errors**: Easy to detect as the compiler indicates the error location and cause.
- **Runtime Errors**: Identified when the program crashes or gives unexpected results.
- **Logic Errors**: Challenging to identify as the program runs without errors but produces incorrect results. Requires thorough testing and debugging.

1.11. Developing Java Programs Using NetBeans

Practice.

1.12. Developing Java Programs Using Eclipse

Practice.