Java Learning Journey

Chapter 1: Introduction to Computers, Programs, and Java

About This Chapter

This chapter introduces the **fundamentals of Java programming**.
It covers the essential building blocks that will form the foundation for OOP and Data Structures later.

Prepared by: Ahmed Elsifi

1.1 Introduction

- **Programming** is the process of creating software (programs) that give instructions to computers.
- Programs are everywhere: PCs, phones, cars, appliances, etc.
- Java is one of many programming languages; each has strengths and weaknesses.
- Learning to program in one language makes it easier to learn others.

1.2 What Is a Computer?

A computer is an electronic device that processes data. It consists of:

Hardware Components:

- **CPU**: The brain of the computer. Executes instructions.
- **Memory (RAM)**: Temporary storage for programs and data. Volatile.
- **Storage Devices**: Permanent storage (e.g., hard disks, SSDs, USB drives).
- **Input Devices**: Keyboard, mouse, touchscreen.
- **Output Devices**: Monitor, printer.
- **Communication Devices**: Modem, NIC, Wi-Fi adapter.

Key Concepts:

- **Bit**: Binary digit (0 or 1).
- **Byte**: 8 bits.
- **Units:** KB, MB, GB, TB.
- **Memory Address**: Each byte in memory has a unique address.
- **Bus**: Connects components; data travels via the bus.

1.3 Programming Languages

Machine Language:

- Binary code understood by the CPU.
- Difficult for humans to read/write.
- written in zeros and ones

Assembly Language:

- Uses mnemonics (e.g., `add`, `sub`).
- Translated to machine code via an **assembler**.
- Low-level and hardware-dependent.

High-Level Languages:

- English-like syntax (e.g., Java, Python, C++).
- Platform-independent.
- Source code must be translated to machine code using a **compiler** or **interpreter**.

1.4 Operating Systems (OS)

- Manages hardware and software resources.
- Examples: Windows, macOS, Linux.
- Responsibilities:
  - Controlling system activities.
  - Allocating resources (CPU, memory, etc.).
  - Scheduling operations (multiprogramming, multithreading, multiprocessing).

1.5 Java, the World Wide Web, and Beyond

- Java was developed by Sun Microsystems (now Oracle).
- Key features: platform independence, object-oriented, robust, secure.
- Used in web applications, mobile apps (Android), servers, and more.
- **Applets** (Java programs in browsers) are now deprecated due to security issues.

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

- **Java Language Specification**: Defines Java syntax and semantics.
- **API (Application Programming Interface)**: Predefined classes and interfaces.
- **JDK (Java Development Kit)**: Tools for developing Java programs.
- **JRE (Java Runtime Environment)**: Executes Java bytecode.
- **IDE (Integrated Development Environment)**: Tools like NetBeans or Eclipse for easier development.

1.7 A Simple Java Program

Example:

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

- **Class**: Must match the filename.
- **main method**: Entry point of the program.
- **System.out.println**: Displays output on the console.
- **Comments**: Use `//` for single-line or `/* */` for multi-line.
- **Case Sensitivity**: Java is case-sensitive.

1.8 Creating, Compiling, and Executing a Java Program

1. Write source code in a `.java` file.
2. Compile with `javac Welcome.java` → generates `Welcome.class` (bytecode).
3. Run with `java Welcome`.

- **JVM (Java Virtual Machine)**: Interprets bytecode and runs the program no matter what os you are using , as long as you have the JVM installed it will run your bytecode.

1.9 Programming Style and Documentation

- Use meaningful comments.
- Proper indentation and spacing.
- Consistent brace style (e.g., end-of-line or next-line).
- Follow coding conventions for readability.

1.10 Programming Errors

1. **Syntax Errors**: Compile-time errors (e.g., missing semicolon).
2. **Runtime Errors**: Occur during execution (e.g., division by zero).
3. **Logic Errors**: Program runs but produces wrong results.

Common Errors:

- Missing braces, semicolons, or quotation marks.
- Misspelling names (case sensitivity).

1.11 & 1.12 Developing Java Programs Using NetBeans and Eclipse

- IDEs provide integrated tools for coding, compiling, debugging.
- Steps:
    1. Create a project.
    2. Create a class.
    3. Write code.
    4. Compile and run.

Key Terms to Remember:

- **Hardware**: Physical components of a computer.
- **Software**: Programs that control the hardware.
- **CPU**: Executes instructions.
- **RAM**: Volatile memory for temporary storage.
- **Compiler**: Translates entire source code to machine code.
- **Interpreter**: Translates and executes one line at a time.
- **JVM**: Runs Java bytecode.
- **Syntax Error**: Mistake in code structure.
- **Logic Error**: Mistake in program logic.

Chapter Exercises (Examples):

1. Display messages using `System.out.println`.
2. Perform mathematical computations.
3. Solve equations using Java expressions.

`print` vs `println` in Java

`System.out.print()`

- **Does not** add a new line after output.
- Subsequent output appears on the **same line**.

Example:

```
System.out.print("Hello");
System.out.print("World");
```

Output: `HelloWorld`

`System.out.println()`

- **Adds a new line** after output.
- Subsequent output appears on the **next line**.

Example:

```
System.out.println("Hello");
System.out.println("World");
```

Output:

```
Hello
World
```

Key Difference:

Method	New Line After Output?	Usage Example
`print()`	❌ No	`print("Hello")`
`println()`	☑ Yes	`println("Hello")`

When to Use:

- Use `print()` when you want output to stay on the same line.
- Use `println()` when you want output to move to the next line.

Mixed Example:

```
System.out.print("Hello ");
System.out.println("World!");
System.out.print("Java");
```

Output:

```
Hello World!
Java
```

Note :

```
System.out.println("Welcome to Java!"); == System.out.print("Welcome to Java!\n");
// Gives the same result
```