**Full Stack Development** Full Stack Development refers to the development of both front-end (client-side) and back-end (server-side) of a software application. A Full Stack Developer is responsible for working on the entire software stack, including databases, servers, APIs, and user interfaces.

**Software Project Architecture** Software architecture defines the structure, components, and interactions of a software system. Common architectural patterns include:

* **Monolithic Architecture**: A single unified codebase.
* **Microservices Architecture**: A collection of small, independent services.
* **Layered Architecture**: Divides the application into presentation, business logic, and data layers.
* **Event-Driven Architecture**: Components communicate via events.

**Roles & Responsibilities of a Full Stack Developer**

* Develop front-end and back-end applications.
* Design and manage databases.
* Work with version control systems.
* Implement security best practices.
* Deploy and maintain applications.

**What is a Database and Why Do We Need It?** A database is a structured collection of data that allows for efficient storage, retrieval, and management. Databases are needed for:

* Storing large amounts of data persistently.
* Ensuring data integrity and security.
* Supporting multi-user access.

**What is a Programming Language & Why Do We Need It?** A programming language is a set of instructions used to communicate with a computer. It is needed to:

* Automate tasks.
* Develop applications.
* Implement logic in software systems.

**What is Java?** Java is a high-level, object-oriented programming language developed by Sun Microsystems (now Oracle). It is known for platform independence (Write Once, Run Anywhere - WORA).

**Features of Java**

* Platform Independence
* Object-Oriented
* Automatic Memory Management
* Multi-threading Support
* Security Features

**Difference Between C and Java**

* **C** is procedural, while **Java** is object-oriented.
* **C** is compiled into machine code, whereas **Java** runs on JVM.
* **C** allows direct memory manipulation; **Java** provides automatic memory management.

**Types of Applications We Can Develop Using Java**

* **Web Applications** (Spring, JSP, Servlets)
* **Mobile Applications** (Android)
* **Desktop Applications** (Swing, JavaFX)
* **Enterprise Applications** (Java EE, Spring Boot)
* **Cloud-based Applications**

**Difference Between JDK, JRE, and JVM**

* **JDK (Java Development Kit)**: Includes tools for developing Java applications.
* **JRE (Java Runtime Environment)**: Provides libraries and JVM to run Java applications.
* **JVM (Java Virtual Machine)**: Executes Java bytecode.

**Execution Flow of Java Program**

1. Write Java Code.
2. Compile into Bytecode (.class file).
3. JVM executes Bytecode.

**Difference Between Interpreter and Compiler**

* **Compiler** translates the entire program before execution.
* **Interpreter** translates code line-by-line at runtime.

**JVM Architecture & Components**

* **Class Loader**: Loads class files.
* **Runtime Memory Areas**: Stores method areas, heap, stack, PC registers.
* **Execution Engine**: Converts bytecode into machine code.

**What is JIT?** Just-In-Time (JIT) compiler optimizes bytecode conversion to native code for faster execution.

**Java Data Types with Size, Range, and Default Values**

| **Data Type** | **Size** | **Range** | **Default Value** |
| --- | --- | --- | --- |
| byte | 1B | -128 to 127 | 0 |
| short | 2B | -32,768 to 32,767 | 0 |
| int | 4B | -2^31 to 2^31-1 | 0 |
| long | 8B | -2^63 to 2^63-1 | 0L |
| float | 4B | Varies | 0.0f |
| double | 8B | Varies | 0.0 |
| char | 2B | Unicode | '\u0000' |
| boolean | 1B | true/false | false |

**Java Program to Print Welcome Message**

public class Welcome {

public static void main(String[] args) {

System.out.println("Welcome to Java!");

}

}

**Java Comments**

* **Single-line (//)**
* **Multi-line (/*...*/)**
* **Javadoc (/**...\*/)\*\*

**Java Naming Conventions**

* **Packages**: lowercase (e.g., com.example)
* **Classes**: PascalCase (e.g., MyClass)
* **Methods & Variables**: camelCase (e.g., calculateSum)

**Operators in Java**

* Arithmetic (+, -, \*, /, %)
* Relational (==, !=, >, <, >=, <=)
* Logical (&&, ||, !)
* Bitwise (&, |, ^, ~)
* Assignment (=, +=, -=)

**Difference Between While & Do-While Loops**

* **while** checks condition before execution.
* **do-while** executes at least once before checking condition.

**Java Program on for Loop**

public class ForLoopExample {

public static void main(String[] args) {

for (int i = 1; i <= 5; i++) {

System.out.println(i);

}

}

}

**Object-Oriented Principles (OOPs)**

* **Encapsulation**: Wrapping data & methods into a single unit.
* **Inheritance**: Acquiring properties from another class.
* **Polymorphism**: Ability to take multiple forms (method overloading/overriding).
* **Abstraction**: Hiding implementation details.

**Access Modifiers in Java**

* **public**: Accessible everywhere.
* **private**: Accessible within class.
* **protected**: Accessible within package and subclasses.

**Difference Between Interface and Abstract Class**

| **Feature** | **Interface** | **Abstract Class** |
| --- | --- | --- |
| Methods | Only abstract | Can have both abstract & concrete methods |
| Variables | public, static, final | Can have instance variables |

**Packages & Import in Java**

* **Packages**: Organizes classes (e.g., package com.example;).
* **Import**: Imports classes (e.g., import java.util.List;).
* **Static Import**: Imports static members directly (e.g., import static java.lang.Math.\*;).

**Wrapper Classes**

* Provides Object representation of primitives (e.g., Integer, Double).
* Needed for collections like ArrayList<Integer>.

This document serves as a comprehensive guide to Java Full Stack Development.