Java Project Development – Knowledge Base

Description: A practical beginner-to-intermediate guide to developing Java applications, covering requirements, Java fundamentals, database connectivity with JDBC, web development with Servlet/JSP, testing, build tools, version control, and deployment.

Java Project Development Flow

1. Requirement	2. Design	3. Development	4. Testing
Understand features and business rules	Choose architecture, database & modules	Java + JDBC / Servlet + JSP	JUnit/TestNG, API & UI testing
5. Build	**6. Version Control**	**7. CI/CD**	**8. Deployment**
Maven / Gradle	Git / GitHub	Jenkins / Azure Pipelines	Application server / Cloud

1. Java Fundamentals

Core Java is the foundation of a Java project. Important concepts include variables, data types, operators, conditions, loops, methods, arrays, strings, exception handling, and packages.

• OOP: class, object, inheritance, polymorphism, abstraction, encapsulation

• Collections: List, Set, Map and their common implementations

• Exception handling: try, catch, finally, throw and throws

• Java 8+ concepts: lambda expressions, streams and functional interfaces

2. Project Structure

A well-organized project separates responsibilities so that code is easier to maintain and test.

• src/main/java – Java source code

• src/main/resources – configuration/resources

• src/test/java – test code

• pom.xml – Maven dependencies and build configuration

• Packages such as controller, service, repository/DAO, model and utility

3. JDBC – Database Connectivity

JDBC (Java Database Connectivity) allows Java applications to communicate with relational databases such as MySQL.

• Load/configure the JDBC driver

• Create a Connection using URL, username and password

• Create Statement or PreparedStatement

• Execute SQL using executeQuery/executeUpdate

• Read ResultSet for SELECT queries

• Close resources; preferably use try-with-resources

• Use PreparedStatement for parameterized queries and safer database access

4. Servlet and JSP

For traditional Java web applications, Servlet and JSP can work together. JSP is mainly used for the presentation layer, while Servlet can receive HTTP requests and coordinate application logic.

• JSP – web page/presentation

• Servlet – request/response handling

• JDBC/DAO – database operations

• Typical flow: Browser → JSP/Form → Servlet → Service/DAO → JDBC → Database

5. Maven

Maven is a build and dependency-management tool commonly used in Java projects.

• pom.xml defines project information and dependencies

• Downloads and manages libraries

• Common lifecycle commands: clean, test, package, install

• Helps create repeatable builds

6. Testing

Testing verifies that the application behaves according to requirements.

• Unit testing – test individual classes/methods

• Integration testing – verify components working together

• API testing – verify REST endpoints

• UI testing – verify web application behavior

• Regression testing – confirm existing functionality still works after changes

• TestNG/JUnit can be used for Java test automation

7. Git and GitHub

Git tracks source-code changes. GitHub can host Git repositories and support collaboration.

• clone/pull – get code

• branch – create isolated work

• add/commit – save changes

• push – upload commits

• pull request – propose changes for review

8. CI/CD

CI/CD automates build, testing and delivery activities.

• Developer pushes code to Git

• CI server starts a build

• Maven compiles and runs tests

• Reports are generated

• Successful builds can be deployed to a target environment

• Common tools include Jenkins and Azure Pipelines

9. Typical Java Web Project Layers

A common layered design separates responsibilities.

• Controller – receives HTTP requests

• Service – business logic

• Repository/DAO – database access

• Model/Entity – represents application data

• Utility – reusable helper functions

• View – JSP or another frontend technology

10. Example Employee Insurance Project

A practical project could be an Employee Insurance Application. Employees can enroll in policies, manage dependents, renew policies, select add-ons, submit claims, and manage personal information.

• UI: employee login, enrollment, policy and claim screens

• Backend: Java/Servlet or Spring-based business logic

• Database: employee, policy, dependent and claim tables

• JDBC/DAO: CRUD operations and SQL queries

• Testing: functional, regression, database, API and UI automation

• Build/CI: Maven + Git + Jenkins/Azure Pipelines

11. Recommended Learning Order

For someone preparing for Java-based QA/automation roles, the following order is practical.

• 1. Core Java + OOP

• 2. Collections + Exception Handling

• 3. SQL + Database concepts

• 4. JDBC

• 5. Maven

• 6. Git/GitHub

• 7. Servlet/JSP basics

• 8. REST API concepts + REST Assured

• 9. Selenium + TestNG + Cucumber

• 10. Jenkins/CI-CD

Quick Reference

Technology	Main Purpose
Java	Application programming language
JDBC	Java ↔ relational database connectivity
Servlet	HTTP request/response handling
JSP	Server-side web presentation
SQL	Database queries and data manipulation
Maven	Build and dependency management
Git	Source-code version control
GitHub	Remote repository and collaboration
TestNG / JUnit	Java test execution
Selenium	Browser/UI automation
REST Assured	REST API automation
Jenkins	CI/CD automation

Key idea: JDBC and JSP are not the same thing. JDBC handles database connectivity; JSP handles the web presentation layer. In a traditional Java web application, Servlet commonly connects the web request to business logic and database-access code.