Learning Consolidation

Structure, Package, and Build a Java Web Application Using Maven

Learning Objectives

- Building a Web Application
- Structuring a Java Application Using Maven
- Components of Maven
- Executing Maven Commands

Slide Note

Client Server Architecture

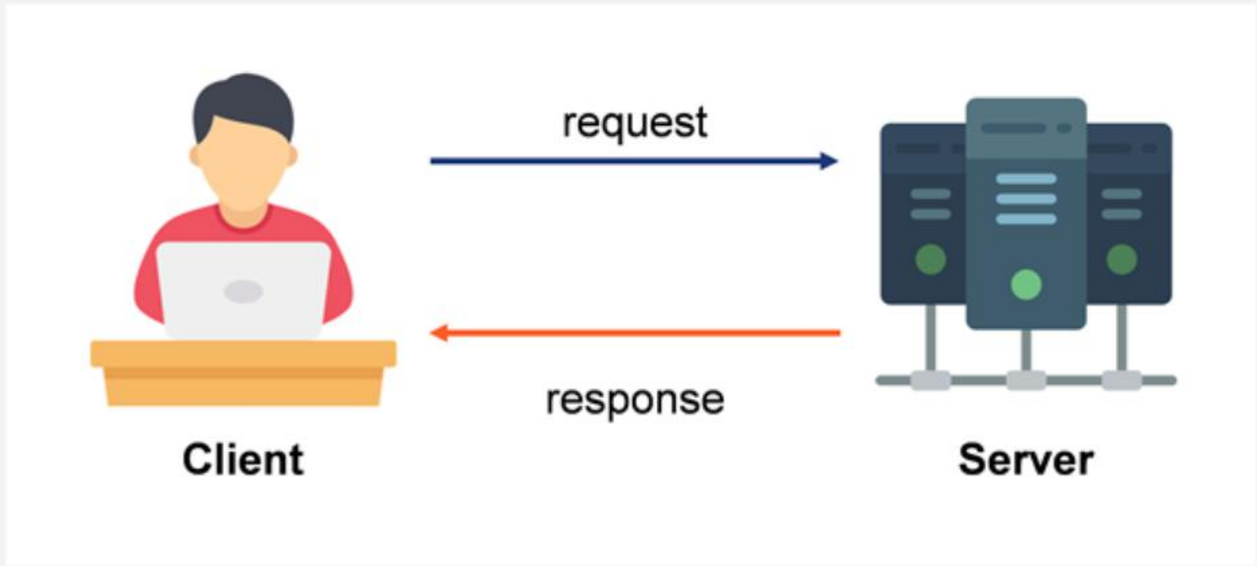
- A **server** is a computer program or device that provides a service to another computer program or device, also known as the **client**.
- This model is called the **client-server** architecture.

- In a web-based scenario:
  - the client is the device that runs on the browser and wishes to access the application
  - the server is where the application runs
- The client sends a request to the web application that executes on a server, and the server, after processing the request, sends a response back to the client.

Slide Note

Request - Response Model

- In the client-server architecture, communication between the client and the server takes place in a request-response model.
- In the request-response model:
  - A client computer requests data or services.
  - A server computer responds to the request by sending the requested data or service back.
- For example, when a login with the correct credentials is sent to a Gmail server, it returns a response by navigating the user to an inbox page.



Slide Note

The Spring Framework

- A web application can be built using any programming language.
- Java provides an easy and efficient way of building such applications using the Spring framework.
- The Spring framework is a well-defined mechanism that helps to build web applications using Java as a programming language.
- Before getting started with the Spring framework, it is important to learn how to manage the web application as it is being built.
- This can be done efficiently by using project management tools like Maven.

Note : Spring will be discussed in detail in later sessions.

Slide Note

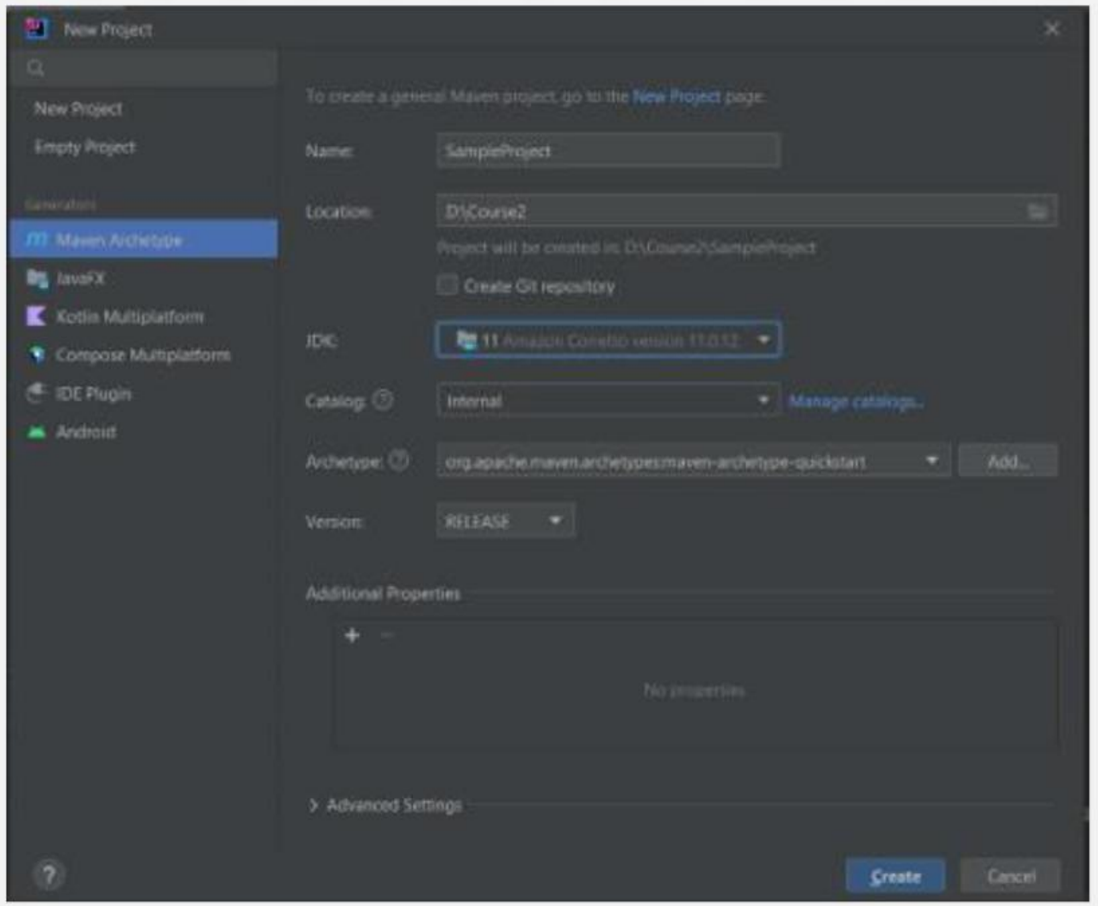
Structuring a Web Application Using Maven

- A web application is built as a project.
- Maven is a software project management tool.
- Maven can also be used to build and manage projects written in Java.
- Maven addresses two aspects of building software applications:
  - How is software built?
  - How are the dependencies managed?

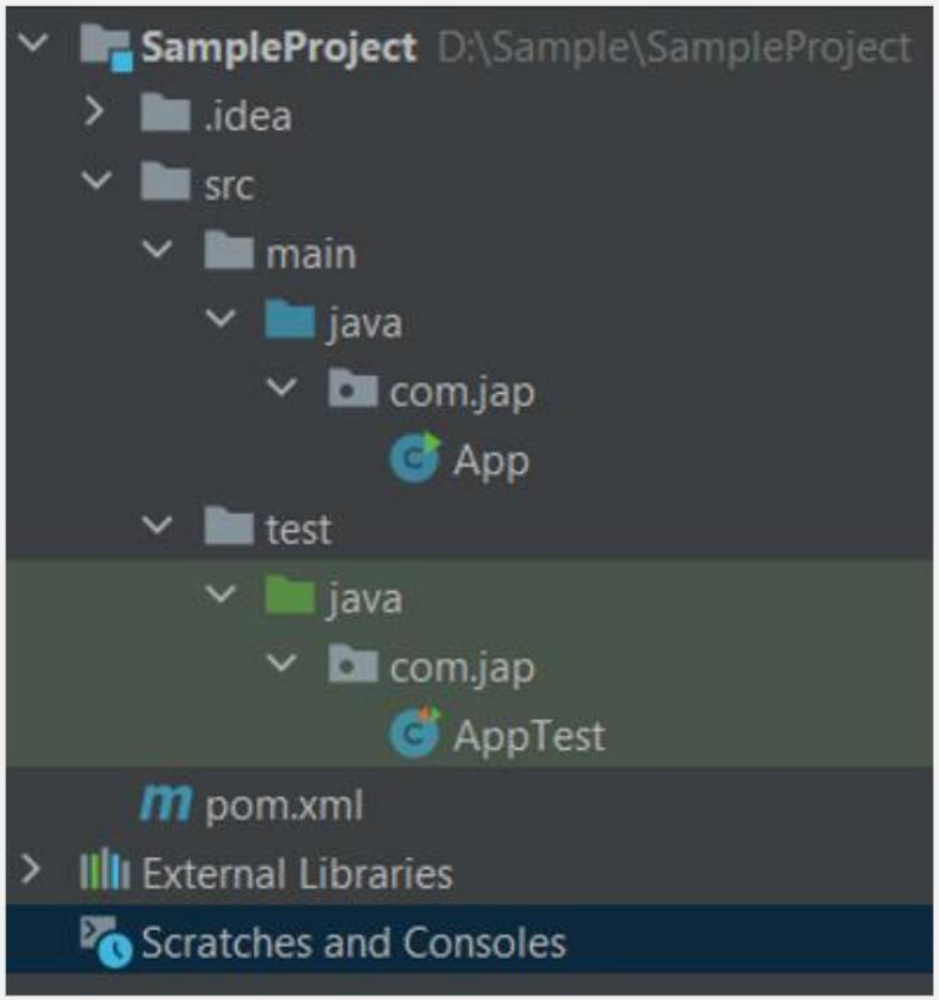
Slide Note

Creating a Java Maven Project

- Open IntelliJ.
- Navigate to File > New > Project.
- The dialog shown in the image pops up.
- Select the Maven Archetype.
- Enter the name of the project, location, and select the JDK version.
- Since we are creating a simple Java project, select the archetype as `quickstart`.
- In the advanced settings, `groupId`, `artifactId`, and `version` can be specified.



Slide Note



Decomposing the Maven Project

- The Maven project gives a default structure to the Java project.
- A src folder contains the Java classes.
- A **test** folder contains the test classes.
- A **pom.xml** that holds all the necessary dependencies that the project will require.

Slide Note

The Maven Archetype

- Archetype is a Maven project template toolkit.
- Archetypes provide templates for creating a Java project.
- A simple Java project can be created using the predefined quick start archetype, which provides structure to the project.
- A few examples of predefined archetypes:
  - `maven-archetype-quickstart` generates a sample Maven standalone project.
  - `maven-archetype-webapp` generates a sample Maven webapp project.
- User-defined archetypes can also be created.

Slide Note

The pom.xml File

- A Project Object Model, or POM, is the fundamental unit of work in Maven.
- It is an XML file that contains information about the project and configuration details used by Maven to build the project.
- It contains default values for most projects. For example, a default `App.java` file is created in the **src directory.**

Slide Note

A snapshot version is one where the project has not yet been released; snapshot versions are bound to change.

```xml
<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
  http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.jap</groupId>
  <artifactId>SampleProject</artifactId>
  <version>1.0-SNAPSHOT</version>
  <name>SampleProject</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>11</maven.compiler.source>
    <maven.compiler.target>11</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

Components of the pom.xml

- **project**: the top-level element in all Maven pom.xml files
- **groupId**: indicates the unique identifier of the organization or group that creates the project
- **artifactId**: indicates the unique base name of the primary artifact being generated by this project
- **packaging**: indicates the package type to be used by this artifact (e.g., JAR, WAR, EAR, etc.)
- **version**: specifies the version of the artifact under the given group
- **name**: indicates the display name used for the project. This is often used in Maven's generated documentation
- **dependencies**: defines the dependencies for this project. For example, the Junit dependency is used so that test cases can be written for the project

Slide Note

Build and Plugins

- Maven executes through the plugins defined in the build tag.
- The plugins are used to accomplish a specific goal.
- Maven plugins are generally used to:
  - Create jar or war files
  - Compile code files
  - Unit test code.
  - Create documentation
- For example,
  - A Java project can be compiled with the maven-compiler-plugin.
  - The surefire plugin runs the JUnit unit tests in an isolated class loader.

```
<build>
  <pluginManagement>
    <plugins>
      <plugin>
        <artifactId>maven-clean-plugin</artifactId>
        <version>3.1.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-resources-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.0</version>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.1</version>
      </plugin>
      <plugin>
        <artifactId>maven-jar-plugin</artifactId>
        <version>3.0.2</version>
      </plugin>
```

Slide Note

Maven Dependencies

- The project will download the Maven dependencies from the central Maven repository website, which is hosted on a cloud server.
- A copy of the dependencies is also maintained locally on the system.
- For example, the JUnit 4 dependencies are downloaded from the central repository and stored locally.
- When Maven builds the project, it first searches the local repository for the dependency; if it's unavailable, it pulls it from the remote or central repository.

Slide Note

Installing Maven

- Since we create Java Maven projects using the IntelliJ IDE, we can use the built-in features of the IDE to execute the project.
- But we can also initiate a build of the projects independently using Maven commands.
- To initiate the build of Maven-based projects, we need some commands that must be executed.
- The mvn tool must be installed before executing any command.
- The installation steps are provided here.

Slide Note

Maven Commands

- `mvn compiler:testCompile` –
  - This command compiles the test classes of the Maven project.
- `mvn package` –
  - This command builds the Maven project and packages it into a jar or war file.
    - It creates the target folder that contains the compiled class and jar file.
- `mvn install` –
  - This command builds the Maven project and installs the project files (JAR, WAR, pom.xml, etc.) to the local repository.

Slide Note

Maven Commands (Cont'd)

- `mvn validate` –
  - This command validates the Maven project by indicating that everything is correct and all the necessary information is available.
- `mvn test` –
  - This command is used to run the test cases of the project using the Maven-surefire-plugin.
- `mvn compile` –
  - This is another command to compile the Java source files.