Introduction to Spring

1. What is Spring Framework?

- **Spring** is a comprehensive **Java framework** for building robust, enterprise-grade applications.
- It simplifies **Java development** by offering a variety of tools for **dependency management**, **aspect-oriented programming**, **web development**, and more.

2. Why Spring?

- Other frameworks like **Struts** (web) and **Hibernate** (ORM) address specific concerns.
- **Spring** provides a **complete solution** for all application needs, including web, database, security, and transaction management.

3. Core Features of Spring:

- **POJO-Based Development**: Build applications using **Plain Old Java Objects** (POJOs), without relying heavily on frameworks.
- **Dependency Injection (DI)**: Handles object creation and dependency management, promoting **loose coupling**.
- **Aspect-Oriented Programming (AOP)**: Manages cross-cutting concerns (e.g., logging, security) separately from business logic.

4. Main Spring Modules:

- **Spring Core**: Provides the fundamental building blocks, including DI and BeanFactory.
- **Spring AOP**: Enables modularization of concerns like logging or transaction management.
- **Spring ORM**: Integration with Hibernate, JPA, and other ORM tools.
- **Spring MVC**: Facilitates web application development using the **Model-View-Controller** pattern.

- **Spring Boot**: A Spring extension for creating **standalone applications** with embedded servers (Tomcat, Jetty).

5. Why Spring?

- **Lightweight** and modular.
- Promotes **clean, maintainable code**.
- Supports a wide range of technologies (Web, ORM, Security).Spring has become a standard for Java development due to its flexibility and ease of integration with other frameworks and libraries.

Spring Docs

1. Community Support

The **Spring community** is one of the largest and most active in the Java ecosystem. It provides extensive resources, open-source projects, forums, and regular updates to the framework.

- **Community forums** and discussions help resolve queries quickly.
- Spring is backed by **Pivotal** (now part of VMware) and maintained by a large group of contributors.
- Active communities: **Stack Overflow**, **GitHub**, **Spring.io forum**.

2. Documentation

- The official **Spring documentation** provides thorough, well-organized guides and references for developers at all levels.
- It includes in-depth explanations, example projects, and best practices for using Spring features.
- Visit: Spring Documentation

3. Why Spring is a Great Framework

- **Modular and Flexible**: You can use just the components you need, like Spring Core, Spring Boot, Spring MVC, etc.
- **Lightweight**: Spring encourages POJO-based development, making it easy to manage without the overhead of heavy containers.
- **Wide Ecosystem**: It integrates seamlessly with various technologies (like Hibernate, JMS, RabbitMQ, etc.) and provides specialized modules for web, security, and data management.

Prerequisites

1. Core Java

- Fundamental features like **OOP concepts**, **Java APIs**, **basic I/O**, and **memory management**.
- Key topics: classes, objects, inheritance, polymorphism, encapsulation, abstraction.

2. Exception

- Mechanism to handle runtime errors using try, catch, finally, throw, and throws.
- Types: Checked and Unchecked exceptions.

3. Threads

- Enables multitasking; created using Thread class or Runnable interface.
- Synchronization for thread safety in shared resources.

4. Collections

- Data structures like **List**, **Set**, **Map**.
- Common classes: ArrayList, HashSet, HashMap.

5. JDBC

- Java API to interact with databases.
- Supports SQL operations: SELECT, INSERT, UPDATE, DELETE via Connection, Statement, PreparedStatement.

6. Maven/Gradle

- Build tools for managing dependencies, project lifecycle, and automation.
- **Maven** uses pom.xml, **Gradle** uses build.gradle for configurations.

7. Spring ORM/Hibernate

- **Spring ORM** integrates Hibernate to manage database operations.
- **Hibernate** is an ORM framework that maps Java objects to database tables.

8. Servlet

- Java class that handles HTTP requests and responses in web applications.
- Part of the **Java EE** framework; works with HttpServletRequest and HttpServletResponse.

IDE for Spring

To set up a Spring development environment, you need the **JDK** and a suitable **IDE** with Spring support. Below are instructions for using **VS Code**, **IntelliJ**, **Spring Tool Suite (STS)**, and **Eclipse**.

1. JDK Installation

Before working with Spring, make sure you have **JDK (Java Development Kit)** installed. You can download it from Oracle.

2. IDE Options for Spring Development

- **VS Code**
- **IntelliJ IDEA** (Community Edition - Free)
- **Spring Tool Suite (STS)** (Based on Eclipse)
- **Eclipse IDE**

Steps to Set Up Spring in Different IDEs

a) Visual Studio Code (VS Code)

1. **Install VS Code** from here.
2. **Install Java Extension Pack**:
    - Go to the **Extensions** marketplace.
    - Search for **Java Extension Pack** and click **Install**. This will install support for Java, including **Maven**, **Debugger**, and **Spring Boot**.
3. **Install Spring Boot Extension**:
    - Search for **Spring Boot Extension Pack** and install it.
4. **Start Spring Boot Project**:
    - Press Ctrl + Shift + P and select **Spring Initializr: Generate a Maven Project**.
    - Configure your project (dependencies, packaging, Java version, etc.).

b) IntelliJ IDEA (Community Edition - Free)

1. **Install IntelliJ IDEA** from here.
2. **Enable Spring Support**:
    - Go to **File > New > Project**.
    - Choose **Spring Initializr** to create a Spring Boot project.
3. **Install Spring Plugin** (if not installed by default):
    - Go to **File > Settings > Plugins**.
    - Search for **Spring Boot** and install the plugin.
4. **Generate a Spring Project**:
    - After installing the plugin, create a Spring Boot project using **Spring Initializr**.

c) Spring Tool Suite (STS)

1. **Install STS** from here.
    - STS is based on Eclipse and comes with built-in Spring support.
2. **Create a New Spring Project**:
    - Open STS and go to **File > New > Spring Starter Project**.
    - Follow the wizard to create a Spring Boot project using **Spring Initializr**.
3. **Install Additional Plugins**:
    - You can install any required plugins via **Eclipse Marketplace** in STS for more capabilities (e.g., for testing or ORM integration).

d) Eclipse IDE

1. **Install Eclipse IDE** from here.
2. **Add Spring Support**:
    - Open **Eclipse Marketplace** (Help > Eclipse Marketplace).
    - Search for **Spring Tools (STS)** and click **Install**.
3. **Create a Spring Project**:
    - Go to **File > New > Spring Starter Project**.
    - Follow the wizard to set up the project.

Steps to Use Spring Project in Any IDE

1. **Start a Spring Project** using **Spring Initializr**.
    - Choose dependencies like **Spring Web**, **Spring Data JPA**, **Thymeleaf**, etc.
2. **Import Project**:
    - For Eclipse/STS, use **File > Import > Existing Maven Project**.
    - In IntelliJ, simply open the project as a Maven/Gradle project.
3. **Run the Project**:
    - For Eclipse/STS, right-click on the project and select **Run As > Spring Boot App**.
    - In IntelliJ, click on the **Run** button in the top-right corner.

Inversion of Control (IoC) and Dependency Injection (DI)

Inversion of Control (IoC)

Definition: IoC is a design principle in which the control of object creation and management is transferred from the application code to a framework or container. This allows developers to focus on the core business logic while the framework takes care of the lifecycle and interactions of the objects.

Example with Laptop and Alien Classes:

- In the given example, the Alien class has a dependency on the Laptop class. Instead of Alien being responsible for creating an instance of Laptop, the Spring framework manages this for us.
- When the application starts, the Spring container creates the Laptop instance and injects it into the Alien class. This is a prime example of IoC, where the control of object creation is inverted from the user code to the framework.

```
@Component
public class Alien {
    @Autowired
    Laptop laptop;  // Dependency on Laptop

    public void code() {
        laptop.compile();  // Calling the compile method on the injected Laptop
instance
    }
}
```

Dependency Injection (DI)

Definition: DI is a specific implementation of IoC that involves providing an object's dependencies from an external source rather than having the object create

them internally. This fosters loose coupling between classes and enhances testability and maintainability.

Example with Laptop and Alien Classes:

- In the provided example, the Alien class is dependent on the Laptop class. By using the @Autowired annotation, Spring automatically injects an instance of Laptop into Alien.
- This eliminates the need for the Alien class to instantiate a Laptop object directly, promoting loose coupling.

```
@Component
public class Laptop {
    public void compile() {
        System.out.println("Compiling...");
    }
}
```

```
@Component
public class Alien {
    @Autowired
    Laptop laptop;  // Dependency Injection occurs here

    public void code() {
        laptop.compile();  // Utilizing the Laptop instance
    }
}
```

Summary

- **IoC** allows the Spring framework to manage the lifecycle and relationships of objects, freeing developers from having to write boilerplate code for object management.
- **DI** is a specific implementation of IoC. It focuses on how dependencies are injected into a class rather than being created by the class itself. For example, the Alien class doesn't create a Laptop instance on its own, instead Spring injects it. This leads to loose coupling, better separation of concerns, and easier testing.

Spring vs Spring Boot

Spring Framework

- **Definition**: Spring is a comprehensive framework for enterprise Java development. It provides support for building Java applications with features such as dependency injection (DI), aspect-oriented programming (AOP), transaction management, and more.
- **Key Features**:
    - **Inversion of Control (IoC)**: Manages the creation and lifecycle of application objects.
    - **Dependency Injection (DI)**: Promotes loose coupling by allowing dependencies to be injected rather than hard-coded.
    - **Aspect-Oriented Programming (AOP)**: Supports cross-cutting concerns like logging and security.
    - **Web Application Development**: Provides support for building web applications using the MVC (Model-View-Controller) architecture.

Spring Boot

- **Definition**: Spring Boot is an extension of the Spring framework that simplifies the process of building Spring-based applications. It offers a set of conventions and defaults that help developers get started quickly without extensive configuration.
- **Key Features**:
    - **Convention Over Configuration**: Reduces the need for boilerplate code and configuration.
    - **Embedded Servers**: Supports embedded servers (e.g., Tomcat, Jetty) for easy deployment and testing.
    - **Auto-Configuration**: Automatically configures Spring components based on the dependencies present in the project.
    - **Standalone Applications**: Simplifies the deployment of applications as standalone JAR files.

Transition from Servlets to Spring

Issues with Servlets

- **Boilerplate Code**: Servlet-based applications often require a significant amount of boilerplate code for tasks like managing the request and response cycle.
- **Tight Coupling**: The coupling between different components can lead to a lack of flexibility and difficulty in testing.
- **Configuration Complexity**: Managing configurations for various components can be complex and cumbersome.
- **Lack of Built-in Features**: Servlets do not provide built-in features for cross-cutting concerns like security, transactions, and logging.

Need for Spring

- **Simplified Development**: Spring addresses the complexities of Java EE development by providing features like DI and AOP, leading to cleaner, more maintainable code.
- **Modularity**: Encourages a modular architecture, making it easier to manage and test different components.
- **Integration**: Spring easily integrates with various technologies (e.g., JPA for database access, Spring Security for authentication) to enhance application capabilities.

Need for Spring Boot When We Have Spring

- **Rapid Development**: Spring Boot streamlines the development process, enabling developers to create production-ready applications quickly with minimal configuration.
- **Microservices Support**: Spring Boot is designed to support microservices architecture, making it easier to build and deploy independent services.
- **Embedded Server**: Unlike traditional Spring applications that require an external server, Spring Boot applications can run as standalone JARs with an embedded server.

- **Simplified Configuration**: Auto-configuration features eliminate the need for complex XML or Java-based configurations, reducing setup time.
- **Community Support**: Spring Boot has a large community and extensive documentation, making it easier for developers to find solutions and best practices.

Summary

Spring provides a powerful framework for building enterprise applications, while Spring Boot simplifies and accelerates the development process, especially for microservices and modern cloud-based applications. Transitioning from servlets to Spring enhances code maintainability, and Spring Boot further streamlines this by offering rapid development and deployment capabilities.

First Spring Boot App

Creating our first Spring Boot project using Spring Initializr is straightforward and efficient. Here are the detailed steps to set it up without any additional dependencies, along with the default Spring Starter dependencies included in the pom.xml.

Step 1: Access Spring Initializr

1. Open your web browser and navigate to Spring Initializr.

Step 2: Configure Project Metadata

2. Fill in the project metadata:
    - **Project**: Select **Maven Project**.
    - **Language**: Select **Java**.
    - **Spring Boot**: Choose the latest stable version (e.g., 3.0.0).
    - **Group**: Enter your group name (e.g., com.telusko).
    - **Artifact**: Enter your artifact name (e.g., SpringBootFirstApplication).
    - **Name**: Enter your project name (e.g., Spring Boot First Application).
    - **Package Name**: Usually auto-filled based on your group and artifact names.
    - **Packaging**: Choose **Jar**.
    - **Java**: Select your Java version (e.g., 21 or 17).

Step 3: Add Dependencies

3. Since you want to create a project with no dependencies, simply skip the dependency selection. If you wanted to add dependencies later, you would do so here.

Step 4: Generate the Project

4. Click on the **Generate** button. This will download a ZIP file containing your new Spring Boot project.

Step 5: Extract and Open the Project

5. Extract the downloaded ZIP file.
6. Open the project in your preferred IDE (e.g., IntelliJ IDEA, Eclipse, or Visual Studio Code).

Step 6: Review the pom.xml

7. Open the pom.xml file in the root directory of your project. Here's what the default Spring Starter dependencies would look like:

```
<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.telusko</groupId>
    <artifactId>SpringBootFirstApplication</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>Spring Boot First Application</name>
    <description>Demo project for Spring Boot</description>
    <properties>
        <java.version>17</java.version> <!-- Ensure the Java version matches your
installation -->
    </properties>

    <dependencies>
        <!-- Spring Boot Starter Web dependency -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter</artifactId>
        </dependency>
        <!-- Spring Boot Starter Test for testing -->
        <dependency>
```

```
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-maven-plugin</artifactId>
      </plugin>
    </plugins>
  </build>
</project>
```

Step 7: Create the Main Application Class

8. Inside the src/main/java/com/telusko/app directory, create a new Java class named SpringBootFirstApplication.java. Here's the code:

```
package com.telusko.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringBootFirstApplication {

  public static void main(String[] args) {
    SpringApplication.run(SpringBootFirstApplication.class, args);
```

```
    }
}
```

Step 8: Run the Application

9. To run the application, you can use the command line or your IDE:

Command Line: Navigate to the project directory and run:

```
./mvnw spring-boot:run
```

IDE: Right-click on the SpringBootFirstApplication class and select Run.

Step 9: Verify the Application

10. Open your web browser and go to (default **port)** http://localhost:8080/. You should see a default error page since there are no endpoints defined yet, but it confirms that your application is running successfully.

Summary

You have now created your first Spring Boot application using Spring Initializr. The default pom.xml file includes essential dependencies such as spring-boot-starter and spring-boot-starter-test. This setup provides a solid foundation for building and running Spring Boot applications effortlessly.

DI Using Spring Boot

Inversion of Control (IoC)

Inversion of Control (IoC) is a design principle where the control of object creation and management is transferred from the application code to a framework or container, allowing developers to focus on business logic rather than dependency management.

Dependency Injection (DI)

Dependency Injection (DI) is a specific implementation of IoC that provides an object's dependencies from an external source, rather than having the object create them internally. This promotes loose coupling, making code more modular, testable, and maintainable.

Step 1: Without Spring (Basic Java Implementation)

```
public class Alien {
    public void code() {
        System.out.println("Coding");
    }
}
```

- The Alien class has a simple method code() that prints "Coding".

Step 2: Main Class without Spring

```
public class SpringBootDemoApplication {
    public static void main(String[] args) {
        Alien obj = new Alien();  // Manually creating the object
        obj.code();
    }
```

```
}
```

- The Alien object is instantiated **manually** and the method code() is called.

Output:

```
Coding
```

Step 3: Using Spring Framework for Dependency Injection

1. Annotate the Alien class with @Component to let Spring manage it as a bean:

```
@Component
public class Alien {
    public void code() {
        System.out.println("Coding");
    }
}
```

2. Update the main class to use **Spring's ApplicationContext** to get the Alien bean:

```
public class SpringBootDemoApplication {
    public static void main(String[] args) {
        ApplicationContext context =
SpringApplication.run(SpringBootDemoApplication.class, args);
        Alien obj = context.getBean(Alien.class);  // Getting bean from Spring
context
        obj.code();
    }
}
```

Output:

```
Coding
```

- Now the Alien object is managed by Spring, not manually instantiated.

Step 4: Multiple Bean Calls (Optional)

- You can call the Alien bean multiple times using context.getBean():

```
Alien obj1 = context.getBean(Alien.class);
obj1.code();

Alien obj2 = context.getBean(Alien.class);
obj2.code();
```

- Spring will manage the lifecycle and instantiation, making DI easier.

Summary:

- **Step 1 & 2**: Manual object creation without Spring.
- **Step 3**: Object creation is managed by Spring using @Component and ApplicationContext.

Autowiring

Autowiring is a feature in the Spring Framework that allows the Spring container to automatically inject dependencies into a bean. It simplifies the process of wiring together beans by eliminating the need for explicit setter or constructor calls. When a bean requires a dependency, Spring resolves and injects it into the bean automatically.

1. Main Application Class (SpringBootDemoApplication)

This class starts the Spring Boot application and retrieves the Alien bean from the Spring context.

```
package com.telusko.app;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;

@SpringBootApplication
public class SpringBootDemoApplication {
   public static void main(String[] args) {
      ApplicationContext context =
SpringApplication.run(SpringBootDemoApplication.class, args);

      Alien obj = context.getBean(Alien.class); // Retrieve Alien bean
      obj.code();  // Call method on Alien bean
   }
}
```

2. Alien Class (Alien.java)

This class represents a component that depends on a Laptop bean, which is injected automatically using the @Autowired annotation.

```
package com.telusko.app;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

@Component
public class Alien {

    @Autowired
    Laptop laptop;  // Autowiring of Laptop bean

    public void code() {
        laptop.compile();  // Calling Laptop's compile method
    }
}
```

3. Laptop Class (Laptop.java)

This is the class whose instance (bean) is injected into the Alien class. The compile method is called when the Alien object invokes its code() method.

```
package com.telusko.app;

import org.springframework.stereotype.Component;

@Component
```

```
public class Laptop {

    public void compile() {
        System.out.println("Compiling...");  // Output message
    }
}
```

Output

When you run the SpringBootDemoApplication class, the Alien bean's code() method is called, which in turn calls the Laptop bean's compile() method. The output will be:

```
Compiling...
```