## **Spring Framework**

### **1. Spring Core Basics**

#### **Spring Dependency Injection (DI)**

Dependency Injection is a design pattern used to implement IoC, where the control of creating and binding the dependent objects is given to the Spring container.

**Types of Dependency Injection:**

* **Constructor Injection**
* **Setter Injection**

**Example:**

// Constructor Injection

public class TextEditor {

private SpellChecker spellChecker;

@Autowired

public TextEditor(SpellChecker spellChecker) {

this.spellChecker = spellChecker;

}

}

// Setter Injection

public class TextEditor {

private SpellChecker spellChecker;

@Autowired

public void setSpellChecker(SpellChecker spellChecker) {

this.spellChecker = spellChecker;

}

}

#### 

#### **Spring Inversion of Control (IoC)**

IoC is a principle in which the control flow of a program is inverted. In Spring, IoC is achieved through Dependency Injection.

**Example:**

// Bean configuration file (beans.xml)

<beans>

<bean id="textEditor" class="com.example.TextEditor">

<property name="spellChecker" ref="spellChecker"/>

</bean>

<bean id="spellChecker" class="com.example.SpellChecker"/>

</beans>

### **2. Aspect-Oriented Programming (AOP)**

AOP allows for the separation of cross-cutting concerns (like logging, transaction management) from the main business logic.

**Example:**

@Aspect

public class LoggingAspect {

@Before("execution(\* com.example.\*.\*(..))")

public void logBefore(JoinPoint joinPoint) {

System.out.println("Method called: " + joinPoint.getSignature().getName());

}

}

### 

### **3. Bean Scopes**

**Singleton:** Single instance per Spring IoC container. **Prototype:** New instance each time a bean is requested. **Request:** Single instance per HTTP request. **Session:** Single instance per HTTP session. **Application:** Single instance per ServletContext. **WebSocket:** Single instance per WebSocket lifecycle.

**Example:**

<bean id="myBean" class="com.example.MyBean" scope="singleton"/>

### **4. Auto Wiring**

Spring can automatically wire relationships between collaborating beans.

**Example:**

@Autowired

private SpellChecker spellChecker;

### **5. Annotations**

Annotations are used to provide metadata for the Spring container.

**Common Annotations:**

* @Component
* @Service
* @Repository
* @Controller
* @Autowired
* @Qualifier
* @Scope

**Example:**

@Component

public class MyComponent {

//...

}

### **6. Life Cycle Callbacks**

Spring provides lifecycle callbacks for beans using the @PostConstruct and @PreDestroy annotations.

**Example:**

java

@Component

public class MyBean {

@PostConstruct

public void init() {

// Initialization logic

}

@PreDestroy

public void destroy() {

// Cleanup logic

}

}

### 

### **7. Bean Configuration Styles**

**XML-Based Configuration:**

<bean id="myBean" class="com.example.MyBean"/>

**Java-Based Configuration:**

@Configuration

public class AppConfig {

@Bean

public MyBean myBean() {

return new MyBean();

}

}

**Annotation-Based Configuration:**

@Component

public class MyBean {

//...

}

## **Spring Boot**

### **1. Spring Boot Build Systems**

Spring Boot supports various build systems like Maven and Gradle.

**Example:** **Maven pom.xml**

xml

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

**Gradle build.gradle**

groovy

implementation 'org.springframework.boot:spring-boot-starter-web'

### **2. Spring Boot Code Structure**

A typical Spring Boot project has the following structure:

css

src/main/java

└── com/example/myapp

├── MyAppApplication.java

├── controller

├── service

└── repository

src/main/resources

├── application.properties

└── static

└── templates

### **3. Spring Boot Runners**

Spring Boot provides runners like CommandLineRunner to run specific pieces of code when the application starts.

**Example:**

java

@SpringBootApplication

public class MyAppApplication {

public static void main(String[] args) {

SpringApplication.run(MyAppApplication.class, args);

}

@Bean

public CommandLineRunner commandLineRunner() {

return args -> {

System.out.println("Application started!");

};

}

}

### **4. Logger**

Spring Boot uses SLF4J for logging by default.

**Example:**

java

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

@RestController

public class MyController {

private static final Logger logger = LoggerFactory.getLogger(MyController.class);

@GetMapping("/hello")

public String hello() {

logger.info("Hello endpoint was called");

return "Hello, World!";

}

}

### **5. Building RESTful Web Services**

**Creating a Rest Controller:**

java

@RestController

public class MyController {

@GetMapping("/hello")

public String hello() {

return "Hello, World!";

}

}

**Request Mapping:**

java

@RequestMapping("/api")

public class ApiController {

@GetMapping("/greet")

public String greet() {

return "Greetings!";

}

}

**Request Body:**

java

@PostMapping("/create")

public ResponseEntity<MyObject> create(@RequestBody MyObject myObject) {

// Process the request body

return ResponseEntity.ok(myObject);

}

**Path Variable:**

java

@GetMapping("/user/{id}")

public User getUser(@PathVariable Long id) {

return userService.getUserById(id);

}

**Request Parameter:**

java

@GetMapping("/search")

public List<User> searchUsers(@RequestParam String name) {

return userService.searchUsersByName(name);

}

**HTTP Methods (GET, POST, PUT, DELETE):**

java

@RestController

@RequestMapping("/users")

public class UserController {

@GetMapping("/{id}")

public User getUser(@PathVariable Long id) {

return userService.getUserById(id);

}

@PostMapping

public User createUser(@RequestBody User user) {

return userService.createUser(user);

}

@PutMapping("/{id}")

public User updateUser(@PathVariable Long id, @RequestBody User user) {

return userService.updateUser(id, user);

}

@DeleteMapping("/{id}")

public void deleteUser(@PathVariable Long id) {

userService.deleteUser(id);

}

}

**Building Web Applications:** Spring Boot allows easy building of web applications with embedded servers like Tomcat.

**Example:**

java

@SpringBootApplication

public class MyAppApplication {

public static void main(String[] args) {

SpringApplication.run(MyAppApplication.class, args);

}

}

### **Example Project Setup**

**Main Application Class:**

java

@SpringBootApplication

public class MyAppApplication {

public static void main(String[] args) {

SpringApplication.run(MyAppApplication.class, args);

}

}

**Controller Class:**

java

@RestController

@RequestMapping("/api")

public class MyController {

@GetMapping("/hello")

public String hello() {

return "Hello, World!";

}

@PostMapping("/create")

public ResponseEntity<MyObject> create(@RequestBody MyObject myObject) {

return ResponseEntity.ok(myObject);

}

}

**Model Class:**

java

public class MyObject {

private Long id;

private String name;

// Getters and Setters

}

**Service Class:**

java

@Service

public class MyService {

public MyObject process(MyObject myObject) {

// Business logic

return myObject;

}

}

**Repository Class:**

java

@Repository

public interface MyRepository extends JpaRepository<MyObject, Long> {

}

These notes provide a solid foundation for understanding and working with the Spring Framework and Spring Boot. They cover the basic concepts, important annotations, and typical configurations and examples.