Week 4 - Full Stack Application Development

Task 4.1: Basic Servlet-based Spring Boot Application

Steps:
1. Create Spring Boot project with spring-boot-starter-web and devtools.
2. Main Class:

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

3. Controller:

```
@RestController
public class HelloController {
    @GetMapping("/")
    public String home() {
        return "Spring Boot is Running Successfully!";
    }
}
```

4. application.properties:

```
server.port=9090
```

Outcome: Embedded Tomcat runs on port 9090.

Task 4.2: Field Injection using @Autowired

```
@Service
public class StudentService {
    public String getMessage() {
        return "Service Layer Called";
    }
}

@RestController
public class StudentController {

    @Autowired
    private StudentService service;

    @GetMapping("/student")
    public String getStudent() {
        return service.getMessage();
    }
}
```

Outcome: REST endpoint calls service layer using field injection.

Task 4.3: Constructor Injection

Interface:

```
public interface PaymentService {
    String pay();
}
```

Implementation:

```
@Service
public class PaymentServiceImpl implements PaymentService {
    public String pay() {
        return "Payment Successful";
    }
}
```

Controller:

```java
@RestController
public class PaymentController {

    private final PaymentService paymentService;

    @Autowired
    public PaymentController(PaymentService paymentService) {
        this.paymentService = paymentService;
    }

    @GetMapping("/pay")
    public String processPayment() {
        return paymentService.pay();
    }
}
```

Outcome: Constructor injection ensures better dependency management.

Task 4.4: Multiple Beans with @Qualifier

Interface:

```java
public interface NotificationService {
    String notifyUser();
}
```

Email Service:

```java
@Service("emailService")
public class EmailNotificationService implements NotificationService {
    public String notifyUser() {
        return "Email Notification Sent";
    }
}
```

SMS Service:

```java
@Service("smsService")
public class SMSNotificationService implements NotificationService {
    public String notifyUser() {
        return "SMS Notification Sent";
    }
}
```

Controller:

```java
@RestController
public class NotificationController {

    @Autowired
    @Qualifier("emailService")
    private NotificationService service;

    @GetMapping("/notify")
    public String sendNotification() {
        return service.notifyUser();
    }
}
```

Outcome: @Qualifier resolves ambiguity between multiple beans.

Task 4.5: Optional Dependency Injection

```java
@Component
public class OptionalComponent {
    public String message() {
        return "Optional Bean Present";
    }
}

@RestController
public class OptionalController {
    @Autowired(required = false)
    private OptionalComponent component;

    @GetMapping("/optional")
    public String check() {
        if (component != null)
            return component.message();
        else
            return "Optional Bean Not Available";
    }
}
```

Outcome: Application runs even if bean is missing.

Task 4.6: Employee Management using BeanFactory

```java
@Component
public class Employee {
    private String name = "John";
    public String getName() { return name; }
}

@Component
public class EmployeeService {

    @Autowired
    private Employee employee;

    public String getEmployee() {
        return employee.getName();
    }
}
```

Main Class:

```java
public static void main(String[] args) {
    ApplicationContext context =
        new AnnotationConfigApplicationContext("com.example");

    EmployeeService service = context.getBean(EmployeeService.class);
    System.out.println(service.getEmployee());
}
```

Outcome: Demonstrates IoC and DI using Spring Core.

Task 4.7: Spring MVC Application

Controller:

```java
@Controller
public class EmployeeController {

    @GetMapping("/employee")
    public String getEmployee(Model model) {
        model.addAttribute("name", "Padmini");
        return "employee";
    }
}
```

employee.html (Thymeleaf):

```html
<html>
<body>
<h2>Employee Name: <span th:text="${name}"></span></h2>
</body>
</html>
```

Outcome: MVC flow without XML configuration.

End of Week 4 Solutions