## **Case Study 1: XML-Based Configuration**

# Case Study Title: Hospital Management System Scenario:

A hospital wants a simple system to manage patient information, appointments, and billing. You need to implement these features using Spring's XML-based configuration.

#### **Folder Structure:**

```
hospital-management-xml/

src/
main/
java/
com/example/hospital/
Patient.java
Appointment.java
Billing.java
HospitalService.java
resources/
applicationContext.xml
pom.xml
```

#### **POJO Classes:**

## 1. Patient.java

```
• registerPatient(): Register a new patient
```

• getPatientDetails(): View details

```
package com.example.hospital;
public class Patient {
    private String name;
    private int age;

public void setName(String name) {
        this.name = name;
    }

public void setAge(int age) {
        this.age = age;
    }

public void registerPatient() {
        System.out.println(" Patient registered: " + name + ", Age: " + age);
    }

public void getPatientDetails() {
        System.out.println(" Patient Details: " + name + " (Age: " + age + ")");
    }
}
```

```
2. Appointment.java
bookAppointment(): Book appointment
• cancelAppointment(): Cancel it
package com.example.hospital;
public class Appointment {
  public void bookAppointment() {
    System.out.println("Appointment booked successfully!");
  }
  public void cancelAppointment() {
    System.out.println(" Appointment cancelled!");
}
3. Billing.java
• generateBill(): Generate invoice
• sendBill(): Email invoice
package com.example.hospital;
public class Billing {
  public void generateBill() {
    System.out.println("Bill generated successfully!");
  }
  public void sendBill() {
    System.out.println(" Bill sent to patient's email!");
  }
}
//HospitalService.java
package com.example.hospital;
public class HospitalService {
  private Patient patient;
  private Appointment appointment;
  private Billing billing;
  public void setPatient(Patient patient) {
    this.patient = patient;
  public void setAppointment(Appointment appointment) {
    this.appointment = appointment;
```

```
public void setBilling(Billing billing) {
    this.billing = billing;
  public void manageHospital() {
    patient.registerPatient();
    patient.getPatientDetails();
    appointment.bookAppointment();
    billing.generateBill();
    billing.sendBill();
  }
}
//applicationContext.xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"</pre>
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans.xsd">
  <!-- Patient Bean -->
  <bean id="patient" class="com.example.hospital.Patient">
    property name="name" value="John Doe"/>
    your name="age" value="30"/>
  </bean>
  <!-- Appointment Bean -->
  <bean id="appointment" class="com.example.hospital.Appointment"/>
  <!-- Billing Bean -->
  <bean id="billing" class="com.example.hospital.Billing"/>
  <!-- HospitalService Bean (Dependency Injection) -->
  <bean id="hospitalService" class="com.example.hospital.HospitalService">
    cproperty name="patient" ref="patient"/>
    property name="appointment" ref="appointment"/>
    cproperty name="billing" ref="billing"/>
  </bean>
</beans>
//MainApp.java
package com.example.hospital;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
```

```
public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        HospitalService hospitalService = context.getBean("hospitalService", HospitalService.class);
        hospitalService.manageHospital();
    }
}
```

## **Key Learning:**

- Use of XML to wire beans.
- applicationContext.xml manages object creation and dependencies.
- Beans injected using <bean> and <property> tags.

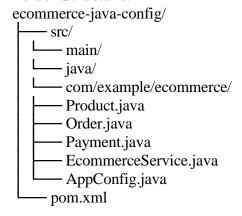
## Case Study 2: Java-Based Configuration

## **Case Study Title: E-Commerce Order Processing**

#### **Scenario:**

An e-commerce application handles product orders, payments, and inventory. We implement the service using Spring's Java configuration (@Configuration, @Bean).

#### **Folder Structure:**



#### **POJO Classes:**

//pom.xml

```
<artifactId>ecommerce-java-config</artifactId>
  <version>1.0-SNAPSHOT</version>
  cproperties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  <dependencies>
    <!-- Spring Core -->
    <dependency>
      <groupId>org.springframework
      <artifactId>spring-core</artifactId>
      <version>5.3.39</version>
    </dependency>
    <!-- Spring Context (Java-based config support) -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.39</version>
    </dependency>
  </dependencies>
</project>
1. Product.java
addProduct(), listProducts()
//Product.java
package com.example.ecommerce;
import java.util.ArrayList;
import java.util.List;
public class Product {
  private List<String> products = new ArrayList<>();
  public void addProduct(String productName) {
    products.add(productName);
    System.out.println("Product added: " + productName);
  }
  public void listProducts() {
    System.out.println("Available Products:");
    for (String p : products) {
      System.out.println(" - " + p);
    }
  }
```

```
}
2. Order.java
createOrder(), cancelOrder()
package com.example.ecommerce;
public class Order {
  public void createOrder(String product) {
    System.out.println("Order created for product: " + product);
  public void cancelOrder(String product) {
    System.out.println("Order cancelled for product: " + product);
  }
}
3. Payment.java
o processPayment(), refundPayment()
package com.example.ecommerce;
public class Payment {
  public void processPayment(double amount) {
    System.out.println("Payment processed: $" + amount);
  public void refundPayment(double amount) {
    System.out.println("Payment refunded: $" + amount);
}
//EcommerceService.java
package com.example.ecommerce;
public class EcommerceService {
  private Product product;
  private Order order;
  private Payment payment;
  public EcommerceService(Product product, Order order, Payment payment) {
    this.product = product;
    this.order = order;
    this.payment = payment;
  }
  public void runEcommerceFlow() {
    product.addProduct("Laptop");
    product.addProduct("Smartphone");
```

```
product.listProducts();
    order.createOrder("Laptop");
    payment.processPayment(1500.00);
    payment.refundPayment(1500.00);
    order.cancelOrder("Laptop");
  }
}
//AppConfig.java
package com.example.ecommerce;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
@Configuration
public class AppConfig {
  @Bean
  public Product product() {
    return new Product();
  @Bean
  public Order order() {
    return new Order();
  }
  @Bean
  public Payment payment() {
    return new Payment();
  @Bean
  public EcommerceService ecommerceService() {
    return new EcommerceService(product(), order(), payment());
  }
}
//MainApp.java
package com.example.ecommerce;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationContext;
public class MainApp {
  public static void main(String[] args) {
    ApplicationContext context = new AnnotationConfigApplicationContext(AppConfig.class);
    EcommerceService = context.getBean(EcommerceService.class);
    service.runEcommerceFlow();
}
```

## **Key Learning:**

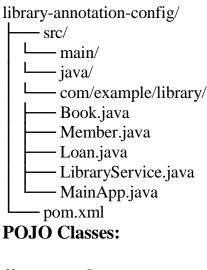
- Uses @Configuration and @Bean to define dependencies.
- No need for XML.
- AnnotationConfigApplicationContext is used instead of ClassPathXmlApplicationContext.

## Case Study 3: Annotation-Based Configuration

## Case Study Title: Library Management System Scenario:

A small community library wants a system to manage books, members, and loans. You implement this using annotation-based Spring (@Component, @Autowired).

#### **Folder Structure:**



### //pom.xml

```
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-core</artifactId>
      <version>5.3.39</version>
    </dependency>
    <!-- Spring Context (Annotation-based config support) -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.39</version>
     </dependency>
  </dependencies>
</project>
1. Book.java
addBook(), searchBook()
package com.example.library;
import org.springframework.stereotype.Component;
@Component
public class Book {
  public void addBook(String title) {
    System.out.println(" Book added: " + title);
  public void searchBook(String title) {
    System.out.println(" Searching for book: " + title);
  }
}
2. Member.java
registerMember(), viewMembers()
package com.example.library;
import org.springframework.stereotype.Component;
import java.util.ArrayList;
import java.util.List;
@Component
public class Member {
  private List<String> members = new ArrayList<>();
  public void registerMember(String name) {
    members.add(name);
```

```
System.out.println("

✓ Member registered: " + name);
  }
  public void viewMembers() {
    System.out.println("
                         Registered Members:");
    for (String m : members) {
      System.out.println(" - " + m);
    }
  }
3. Loan.java
issueBook(), returnBook()
package com.example.library;
import org.springframework.stereotype.Component;
@Component
public class Loan {
  public void issueBook(String title, String member) {
    System.out.println(" Book issued: " + title + " to " + member);
  }
  public void returnBook(String title, String member) {
    System.out.println(" Book returned: " + title + " by " + member);
}
LibraryService.java
package com.example.library;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class LibraryService {
  @Autowired
  private Book book;
  @Autowired
  private Member member;
  @Autowired
  private Loan loan;
  public void libraryOperations() {
    // Step 1: Add and Search Book
```

```
book.addBook("Java Programming");
     book.addBook("Spring Framework");
     book.searchBook("Java Programming");
     // Step 2: Register and View Members
     member.registerMember("Sravani");
     member.registerMember("John");
     member.viewMembers();
     // Step 3: Issue and Return Books
     loan.issueBook("Java Programming", "Sravani");
     loan.returnBook("Java Programming", "Sravani");
   }
 }
MainApp.java
 package com.example.library;
 import org.springframework.context.ApplicationContext;
 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
 import org.springframework.context.annotation.ComponentScan;
 import org.springframework.context.annotation.Configuration;
 @Configuration
 @ComponentScan(basePackages = "com.example.library")
 public class MainApp {
   public static void main(String[] args) {
     ApplicationContext context = new AnnotationConfigApplicationContext(MainApp.class);
```

LibraryService libraryService = context.getBean(LibraryService.class);

libraryService.libraryOperations();

}