Case Study 1: Hospital Management System (XML-Based Configuration)

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

- Patient.java – registerPatient(), getPatientDetails()
- Appointment.java – bookAppointment(), cancelAppointment()
- Billing.java – generateBill(), sendBill()

Key Learning

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

Code Implementation

Patient.java

```
package com.example.hospital;
public class Patient {
    public void registerPatient() {
        System.out.println("Patient Registered");
    }
    public void getPatientDetails() {
        System.out.println("Patient Details");
    }
}
```

Appointment.java

```
package com.example.hospital;
public class Appointment {
    public void bookAppointment() {
        System.out.println("Appointment Booked");
    }
    public void cancelAppointment() {
        System.out.println("Appointment Cancelled");
    }
}
```

Billing.java

```
package com.example.hospital;
public class Billing {
    public void generateBill() {
        System.out.println("Bill Generated");
    }
    public void sendBill() {
        System.out.println("Bill Sent");
    }
}
```

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();
        appointment.bookAppointment();
        billing.generateBill();
    }
}
```

applicationContext.xml

```
<beans xmlns="http://www.springframework.org/schema/beans"
    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">

    <bean id="patient" class="com.example.hospital.Patient"/>
    <bean id="appointment" class="com.example.hospital.Appointment"/>
```

```
    <bean id="billing" class="com.example.hospital.Billing"/>

    <bean id="hospitalService"
class="com.example.hospital.HospitalService">
        <property name="patient" ref="patient"/>
        <property name="appointment" ref="appointment"/>
        <property name="billing" ref="billing"/>
    </bean>
</beans>
```

Case Study 2: E-Commerce Order Processing (Java-Based Configuration)

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

Folder Structure

```
ecommerce-java-config/
├── src/
│   └── main/
│       └── java/
│           └── com/example/ecommerce/
│               ├── Product.java
│               ├── Order.java
│               ├── Payment.java
│               ├── EcommerceService.java
│               └── AppConfig.java
└── pom.xml
```

POJO Classes

- Product.java – addProduct(), listProducts()
- Order.java – createOrder(), cancelOrder()
- Payment.java – processPayment(), refundPayment()

Key Learning

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

Code Implementation

Product.java

```
package com.example.ecommerce;
public class Product {
    public void addProduct() {
```

```
        System.out.println("Product Added");
    }
    public void listProducts() {
        System.out.println("Listing Products");
    }
}
```

Order.java

```
package com.example.ecommerce;
public class Order {
    public void createOrder() {
        System.out.println("Order Created");
    }
    public void cancelOrder() {
        System.out.println("Order Cancelled");
    }
}
```

Payment.java

```
package com.example.ecommerce;
public class Payment {
    public void processPayment() {
        System.out.println("Payment Processed");
    }
    public void refundPayment() {
        System.out.println("Payment Refunded");
    }
}
```

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 processEcommerce() {
        product.addProduct();
        order.createOrder();
        payment.processPayment();
    }
}
```

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());
    }
}
```

Case Study 3: Library Management System (Annotation-Based Configuration)

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

Folder Structure

```
library-annotation-config/
├── src/
│   └── main/
│       └── java/
│           └── com/example/library/
│               ├── Book.java
│               ├── Member.java
│               ├── Loan.java
│               ├── LibraryService.java
│               └── MainApp.java
└── pom.xml
```

POJO Classes

- Book.java – addBook(), searchBook()
- Member.java – registerMember(), viewMembers()
- Loan.java – issueBook(), returnBook()

Key Learning

- Use of annotations like @Component, @Autowired, @Service, @Repository.
- Spring automatically wires beans.
- Clean, decoupled structure without XML or manual bean declaration.

Code Implementation

Book.java

```
package com.example.library;
import org.springframework.stereotype.Component;

@Component
public class Book {
    public void addBook() {
        System.out.println("Book Added");
    }
    public void searchBook() {
        System.out.println("Searching Book");
    }
}
```

Member.java

```
package com.example.library;
import org.springframework.stereotype.Component;

@Component
public class Member {
    public void registerMember() {
        System.out.println("Member Registered");
    }
    public void viewMembers() {
        System.out.println("Viewing Members");
    }
}
```

Loan.java

```
package com.example.library;
import org.springframework.stereotype.Component;

@Component
public class Loan {
    public void issueBook() {
        System.out.println("Book Issued");
```

```
    }
    public void returnBook() {
        System.out.println("Book Returned");
    }
}
```

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 manageLibrary() {
        book.addBook();
        member.registerMember();
        loan.issueBook();
    }
}
```

MainApp.java

```
package com.example.library;
import org.springframework.context.ApplicationContext;
import
org.springframework.context.annotation.AnnotationConfigApplicationConte
xt;
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.manageLibrary();
    }
}
```