🔷 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
2. **Appointment.java**
   - `bookAppointment()`: Book appointment
   - `cancelAppointment()`: Cancel it
3. **Billing.java**
   - `generateBill()`: Generate invoice
   - `sendBill()`: Email invoice

🧠 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:

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

📦 POJO Classes:

1. **Product.java**
    - `addProduct()`, `listProducts()`
2. **Order.java**
    - `createOrder()`, `cancelOrder()`
3. **Payment.java**

- `processPayment()`, `refundPayment()`

🧠 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:

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

📚 POJO Classes:

1. **Book.java**
   - `addBook()`, `searchBook()`
2. **Member.java**

- `registerMember()`, `viewMembers()`

3. **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.

✅ Comparison Summary:

Feature	XML-Based	Java-Based	Annotation-Based
Configuration	XML file	Java class with `@Bean`	Annotations (`@Component`, `@Autowired`)
Learning Use	Best to learn wiring and dependencies	Good for clean, Java-centric configuration	Ideal for real-world projects with less configuration
Flexibility	Verbose but explicit	Cleaner and testable	Lightweight and modern