Spring Framework

Container And wiring

Overview

Topics Covered:

- Recap of Coupling and Dependency Injection
- Introduction to Spring Container
- XML Configuration in Spring
- Accessing Beans from the Container
- Wiring in Spring
- Managing Dependencies in Code

Recap: Reducing Coupling

In the previous session, we learned that reducing coupling requires three steps:

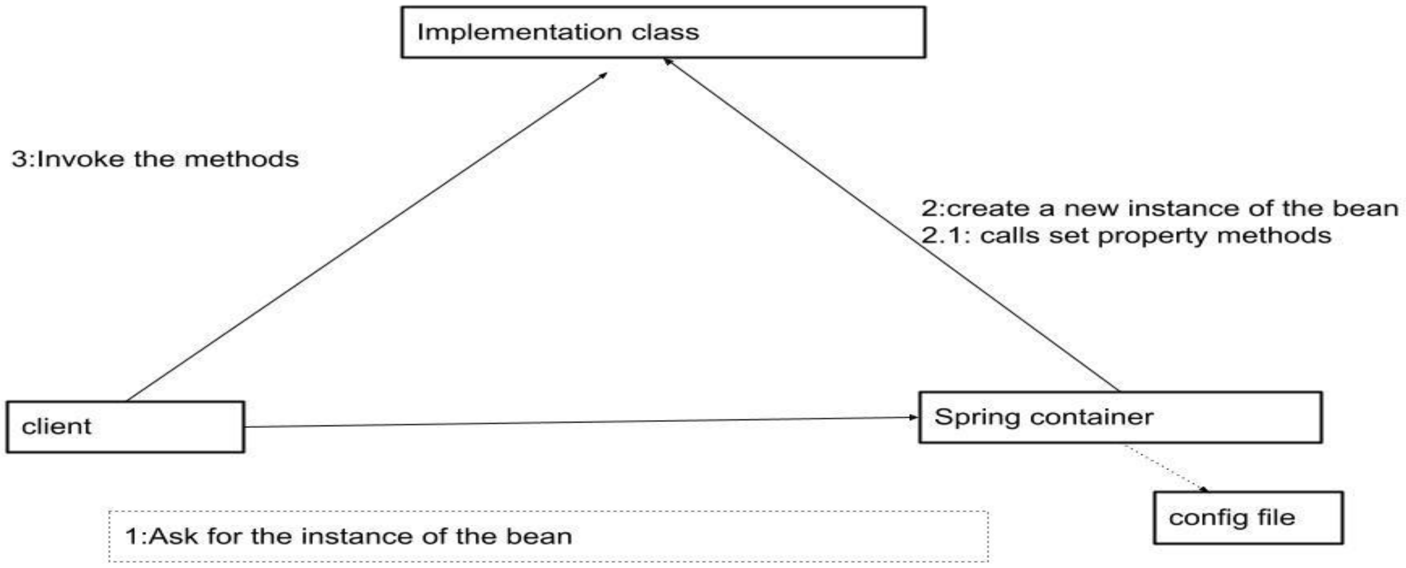
- Program to Interfaces
- Dependency Injection
- Centralise Configuration

Now, we explore how Spring Container helps in centralizing configurations.

What is Spring Container?

- The core of the Spring Framework.
- Responsible for creating, configuring, and managing objects.
- Uses Dependency Injection (DI) to inject dependencies.
- Decouples implementation classes → No need to modify code when switching implementations.
- Unlike traditional Java applications where objects are manually created, Spring Container manages object creation and wiring automatically.

General pattern that we use in Spring


Implementation class
3:Invoke the methods
2:create a new instance of the bean
2.1: calls set property methods
client
Spring container
config file
1:Ask for the instance of the bean

How Spring Creates Objects

Spring reads configuration from XML files or annotations.

Traditional Java:

```
InvoiceService service = new InvoiceService();
```

Spring Managed:

```
ClassPathXmlApplicationContext container = new
ClassPathXmlApplicationContext("application.xml");
```

The container handles object instantiation.

Configuring Spring Container (XML Approach)

Basic XML Configuration:

```xml
<beans>
    <bean id="msgService" class="se.yrgo.msgofday.MessageOfTheDayBasicImpl">
        <property name="message" value="Hello Spring"/>
    </bean>
</beans>
```

Key Elements in XML

Key Elements in XML:

- <beans> → Root element enclosing all bean definitions.
- <bean> → Defines a bean with:
  - id (unique identifier for the bean)
  - class (fully qualified class name of the implementation)
- <property> → Defines property values to be injected.

Spring injects the property message by calling setMessage().

Accessing Beans from Spring Container

Client Code to Retrieve Beans

```
ClassPathXmlApplicationContext container = new
         ClassPathXmlApplicationContext("application.xml");
MessageOfTheDayService helloSpring = container.getBean("msgService",
           MessageOfTheDayService.class);
System.out.println(helloSpring.getTodaysMessage());
container.close();
```

Accessing Beans from Spring Container

getBean("msgService", MessageOfTheDayService.class) retrieves the configured bean.

Beans are referenced by ID and type.

Always close the container after use.

Wiring in Spring

- Wiring refers to configuring object dependencies.
- XML configuration is often called “wiring”.

Example: Wiring a Service and DAO:

```
<bean id="invoiceDao" class="com.example.dao.JdbcInvoiceDao"/>
<bean id="invoiceService" class="com.example.service.InvoiceService">
    <property name="invoiceDao" ref="invoiceDao"/>
</bean>
```

Here, InvoiceService depends on InvoiceDao.

Spring injects invoiceDao into invoiceService.

What is component

Component is a class that contains business logics.

A Service interface for example is a component.

Components are configured in Spring Container, making them accessible using getBean().

Managing Dependencies in Code

Spring helps manage dependencies only where required, not in every class.

Where NOT to Use Spring for Dependency Management:

- Domain Classes (e.g., Book & Author) → These relationships are dynamic and managed at runtime.

Where to Use Spring for Dependency Management:

- Service Layer
- Data Access Layer
- Database Connection Pools

Spring manages configurations that change due to architecture changes, not runtime behavior.

Why Centralized Configuration Matters

- Centralizing configuration ensures flexibility.
- Example: If we switch from JDBC to Hibernate, only one change is needed.

Configuration Before Using Spring:

```
InvoiceService service = new InvoiceService();

service.setInvoiceDao(new JdbcInvoiceDao());
```

This requires manual modifications when changing implementations.

Why Centralized Configuration Matters

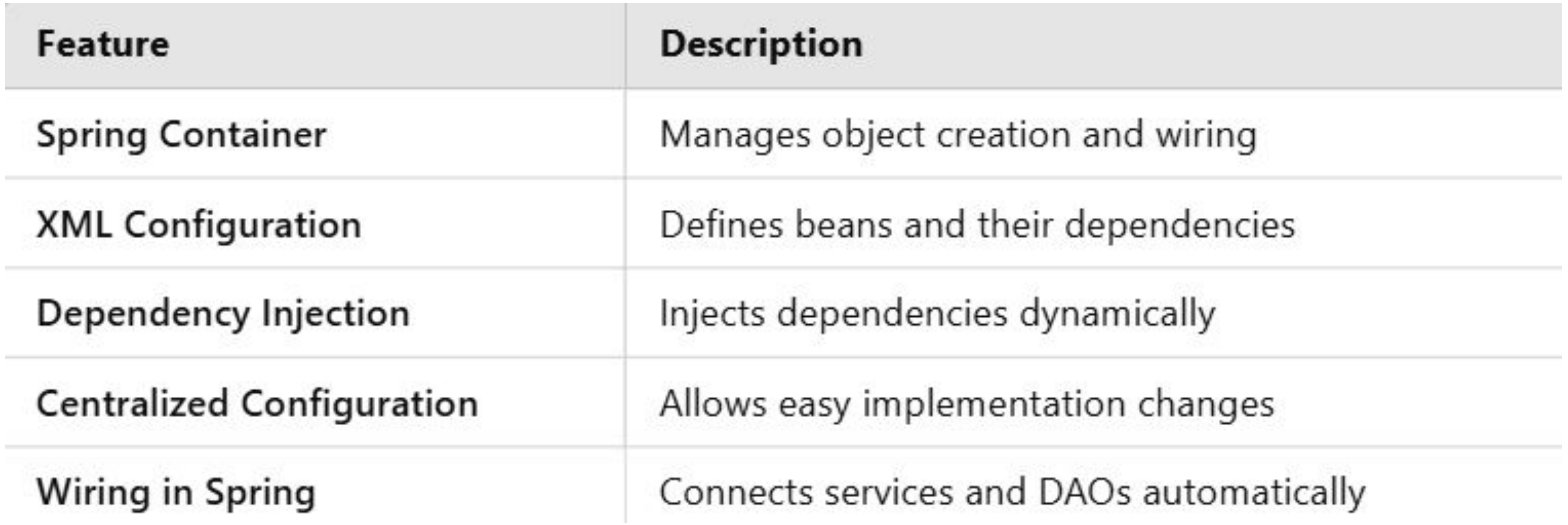
Configuration Using Spring:

```
ClassPathXmlApplicationContext container = new
ClassPathXmlApplicationContext("application.xml");

InvoiceService service = container.getBean("invoiceService",
InvoiceService.class);
```

With Spring, only the XML configuration changes, keeping the client code intact.

Summary



Feature	Description
Spring Container	Manages object creation and wiring
XML Configuration	Defines beans and their dependencies
Dependency Injection	Injects dependencies dynamically
Centralized Configuration	Allows easy implementation changes
Wiring in Spring	Connects services and DAOs automatically