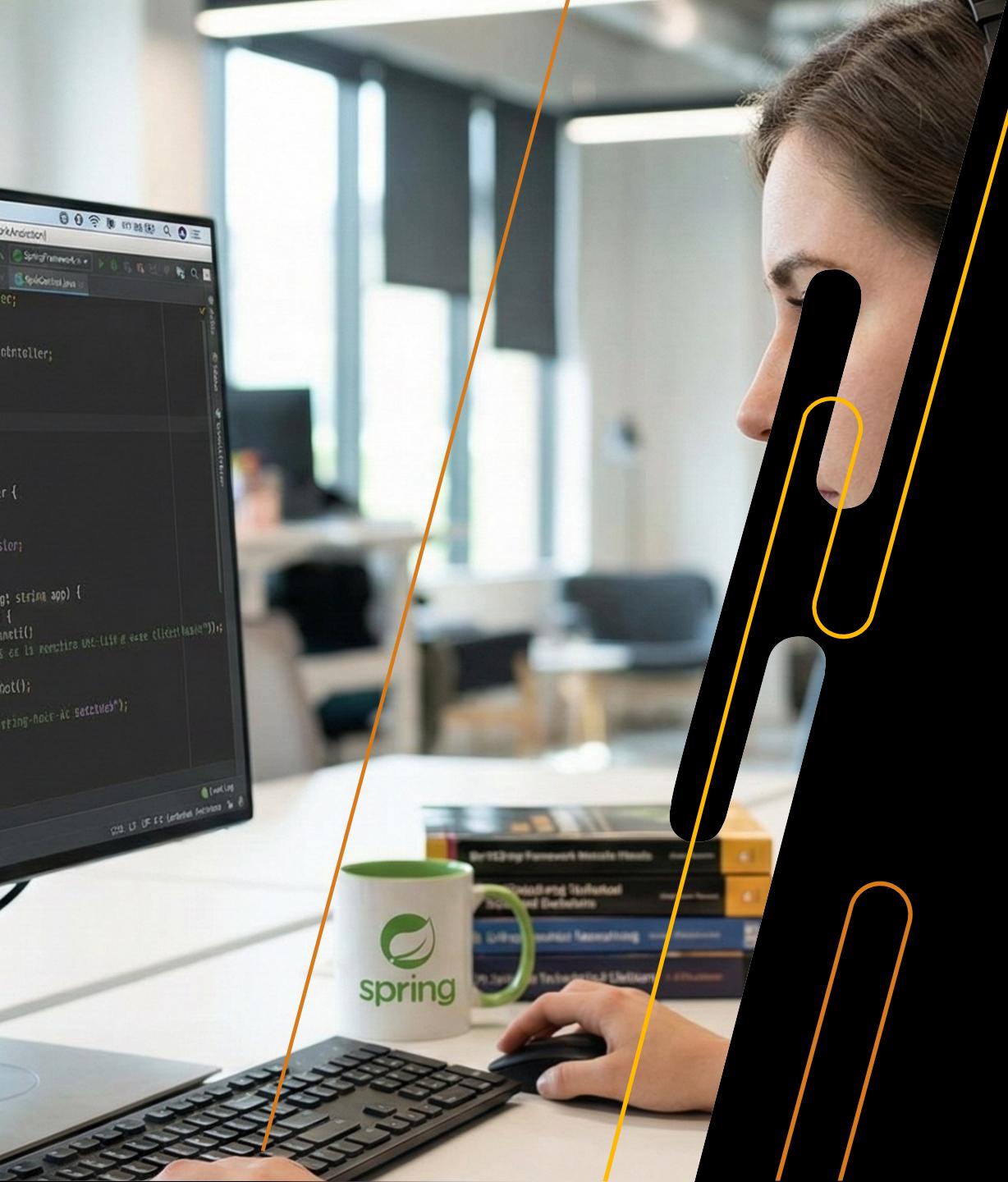




Wiring Beans
using the
Dependency
Injection Pattern

CHAPTER 3

PRESENTATION BY:
RMN

A photograph of a person working at a desk. On the desk is a computer monitor displaying code in a Java IDE, a white mug with a green leaf logo and the word "spring", and a stack of books about the Spring Framework. A hand is visible on a keyboard. A large black paperclip graphic is overlaid on the right side of the image, with orange outlines highlighting its ends.

Chapter Three

We will cover these skills

- The dependency injection pattern
- Types of dependency injection patterns
- Resolving dependency using the Abstract Factory pattern
Lookup-method injection pattern
- Configuring beans using the Factory pattern
Configuring dependencies
- Common best practices for configuring dependencies in an application

The Dependency Injection Pattern

Dependency injection is a design pattern, which promotes the loosely coupled classes in the application.

This means that the classes in the system depend on the behavior of others, and do not depend on instantiation of object of the classes.

Dependency injection provides us with a decoupled and loosely coupled system.

It ensures construction of the dependent object.



Types of Dependency Injection Patterns

The following are the types of dependency injections that could be injected into your application:

- Constructor-based dependency injection
- Setter-based dependency injection



Constructor-based Dependency Injection

A constructor injection is one of the ways of fulfilling these object attributes at the time of creation to instantiate the object.

An object has a public constructor that takes dependent classes as constructor arguments to inject the dependencies.

Advantages:

- Constructor-based dependency injection is more suitable for mandatory dependencies, and it makes a strong dependency contract
- It favors the use of immutable objects, and does not break the information hiding principle

```
public class TransferServiceImpl implements TransferService {  
    AccountRepository accountRepository;  
    TransferRepository transferRepository;  
    public TransferServiceImpl(AccountRepository accountRepository, TransferRepository transferRepository) {  
        this.accountRepository = accountRepository;  
        this.transferRepository = transferRepository;  
    }  
    // ...  
}
```

Setter-based Dependency Injection

In setter injection, one of the ways to fulfil these dependencies is by providing a setter method in the dependent class.

Advantages:

- Setter injection is more readable than the constructor injection
- Setter injection solves the circular dependency problem in the application

Disadvantage:

- Security is lesser in the setter injection pattern, because it can be overridden

```
public class TransferServiceImpl implements TransferService {  
    AccountRepository accountRepository;  
    TransferRepository transferRepository;  
    public void setAccountRepository(AccountRepository accountRepository) {  
        this.accountRepository = accountRepository;  
    }  
    public void setTransferRepository(TransferRepository transferRepository) {  
        this.transferRepository = transferRepository;  
    }  
    // ...  
}
```

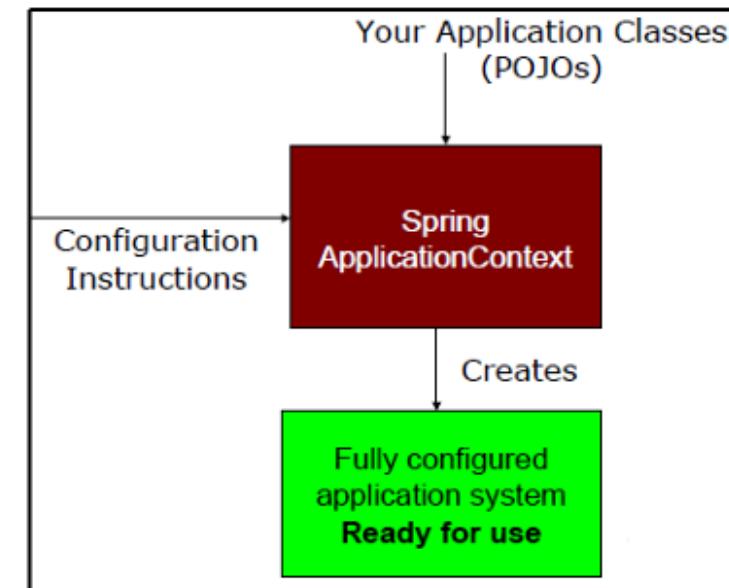
Configuring the Dependency Injection Pattern with Spring

Spring is very flexible in configuring the dependency of Spring beans.

The following are three ways to configure the metadata of your application:

- Dependency injection pattern with Java-based configuration-it is an explicit configuration in Java.
- Dependency injection pattern with Annotation-based configuration-it is an implicit bean discovery, and automatic wiring.

Dependency injection pattern with XML-based configuration-it is an explicit configuration in XML.



Dependency Injection Pattern with Java-based Configuration

As of Spring 3.0, it provides a Java-based Spring configuration to wire the Spring beans.

The `@Configuration` annotation, which indicates that it is a configuration class of the application that contains the details on bean definitions.

To declare a bean in a Java-based configuration, you have to write a method for the desired type object creation in the configuration class, and annotate that method with `@Bean`.

```
package  
com.packt.patterninspring.chapter4.bankapp.config;  
import org.springframework.context.annotation.Bean;  
import  
org.springframework.context.annotation.Configuration;  
;  
@Configuration  
public class AppConfig {  
    @Bean  
    public TransferService  
transferService(AccountRepository accountRepository,  
TransferRepository transferRepository) {  
    return new  
TransferServiceImpl(accountRepository,  
transferRepository);  
}  
    @Bean  
    public AccountRepository accountRepository() {  
    return new JdbcAccountRepository();  
}  
    @Bean  
    public TransferRepository transferRepository() {  
    return new JdbcTransferRepository();  
}  
}
```

Dependency Injection Pattern with XML-based Configuration

Spring provides dependency injection with XML-based configuration from the very beginning.

As with Java, we have to declare a class as a Spring bean into Spring's XML-based configuration by using an element of the Spring-beans schema as a `<bean>` element.

The `<bean>` element is the XML analogue to JavaConfig's `@Bean` annotation.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
    xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="http://www.springframework.org/s
    chema/beans
    http://www.springframework.org/schema/beans/spring-
    beans.xsd">
    <bean id="transferService"
        class="com.packt.patterninspring.chapter4.bankapp.
        service.TransferServiceImpl" p:accountRepository-
        ref="accountRepository" p:transferRepository-
        ref="transferRepository"/>
    <bean id="accountRepository"
        class="com.packt.patterninspring.chapter4.
        bankapp.repository.jdbc.JdbcAccountRepository"/>
    <bean id="transferRepository"
        class="com.packt.patterninspring.chapter4.
        bankapp.repository.jdbc.JdbcTransferRepository"/>
    <!-- more bean definitions go here -->
</beans>
```

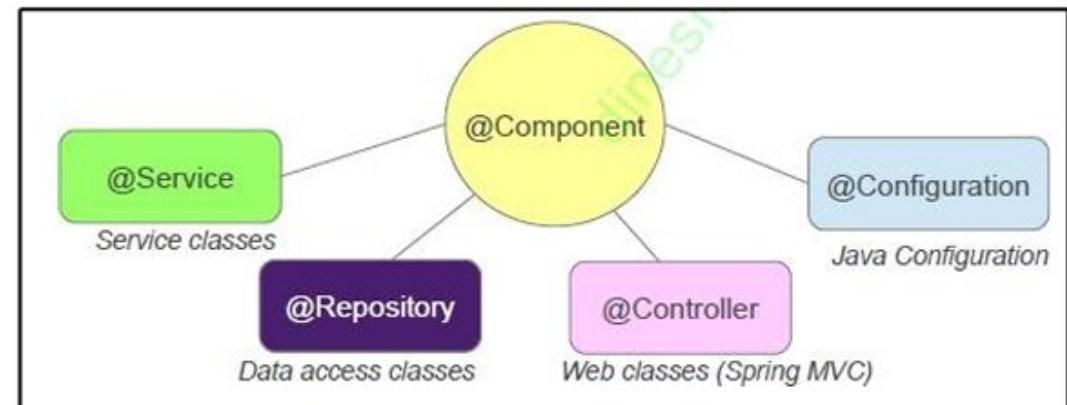
Dependency Injection Pattern with Annotation-based Configuration

The Spring Framework provides you with some special annotations.

The main stereotype annotation is `@Component`.

By using this annotation, Spring provides more Stereotype meta annotations such as:

- `@Service`, used to create Spring beans at the Service layer
- `@Repository`, which is used to create Spring beans for the repositories at the DAO layer
- `@Controller`, which is used to create Spring beans at the controller layer



Component Scanning and Autowiring

Component scanning: In this, Spring automatically searches the beans to be created in the Spring IoC container

Autowiring:

- Spring's @Autowired annotation is used for auto bean wiring.
- This @Autowired annotation indicates that autowiring should be performed for this bean.
- The @Autowired annotation is not limited to the construction; it can be used with the setter method, and can also be used directly in the field





Thank you

RMN