Pivotal

Factory Beans

Spring Essentials

spring by Pivotal

Module Objectives

After completing this lesson, you should be able to do the following

- Describe the Factory Pattern
- Explain the purpose of Factory Beans and be able to use them

Agenda

- **Defining Bean using the Factory Pattern**
- Spring's FactoryBean Interface

Factory Pattern

- Object creation more complicated than a simple use of operator **`new`**
  - Different implementations
    - Depending on platform, configuration, user, …
- Wrap all creation code in a dedicated method or class
  - A “factory” for creating objects

```
public class AccountServiceFactory {
    public AccountService getInstance() {
        // Conditional logic – for example: selecting the right
        //  implementation or sub-class of AccountService to create
        return accountService;
    }
}
```

@Configuration Classes *are* Factories

- Spring's @Bean methods are factory methods
  - They create Spring Beans

```
@Configuration
public class AccountServiceFactory {
    @Bean
    public AccountService accountService() {
        // Conditional logic – for example: selecting the right
        //  implementation or sub-class of AccountService to create
        return accountService;
    }
}
```

Spring *is* a Factory

- Spring creates Spring Beans
  - No matter how you define the beans
- Bean definition options
  - **Java Configuration:** `@Configuration` classes
  - **Annotation-Based:** `@Component`, `@Autowired` and component-scanning
  - **XML Configuration:** `<bean>` elements
    - Common in existing applications
    - Not covered in this course
    - See optional sections at end of notes

Agenda

- Defining Bean using the Factory Pattern
- **Spring's FactoryBean Interface**

Complex Bean Instantiation

Note: XML *is not* in the certification exam – Factory Beans *are*

- Originally Spring only supported XML configuration
  - No easy way to do complex instantiation logic
  - `@Bean` methods can use *any* Java you need
- Instead Spring XML relied on the *Factory Pattern*
  - Use a factory to create the bean(s) we want
    - Implement Spring's **`FactoryBean`** interface
  - Put *any* complex Java code needed in factory's internal logic
  - *Spring FactoryBeans may be used in Java Config also*

The Spring *FactoryBean* interface

- Originally invented as a fall-back for complex configuration in XML
  - Used long before `@Bean` methods introduced

```
interface  FactoryBean<T> {
        // The factory method
        public T getObject() throws Exception;

        // Is this a singleton instance or not?
        public default boolean  isSingleton() { return true; }

        // What type of object is this – easier than introspecting T
        public Class<?> getObjectType();
}
```

Note: Sometimes convenient to use factory beans in @Bean methods

FactoryBean Example

```
public class AccountServiceFactoryBean
        implements FactoryBean <AccountService>{

      public AccountService getObject() throws Exception {
            // Conditional logic – for example: selecting the right
            //  implementation or sub-class of AccountService to create
            return accountService;
       }

       public Class<?> getObjectType() { return AccountService.class; }

       //  isSingleton defaults to returning true since Spring V5
}
```

FactoryBeans with Java Configuration

- Spring calls `getObject()` *automatically*

```
@Configuration
public class ServiceConfig {

    @Bean
    public AccountServiceFactoryBean accountService() {
        return new AccountServiceFactoryBean();
    }

    @Bean
    public CustomerService customerService(AccountService accountService) {
        return new CustomerService(accountService);
    }
```

getObject() called by Spring *internally*

creates

Do not call getObject() yourself

Spring often does *additional* setup *internally* first - such as invoking post-construct methods

Factory Beans in Spring

- FactoryBeans are widely used within Spring
  - `EmbeddedDatabaseFactoryBean`
    - Replaced by `EmbeddedDatabaseBuilder`
  - `JndiObjectFactoryBean`
    - One option for looking up JNDI objects
  - Creating Remoting proxies
  - Creating Caching proxies
  - For configuring data access technologies
    - JPA, Hibernate or MyBatis

Summary

- Factory Beans
  - Important configuration device
  - Understand how *getObject()* works

- XML configuration has existed in Spring since first release
  - Many existing applications use it
  - Optional sections at end of course
    - *XML Configuration*
    - *XML Best Practices*
    - *XML for Spring Security*