Enterprise Java with Spring Spring Core Lab 2 Exercises

1 Lab setup

Make sure you have the following items installed

- Latest LTS JDK version (at this point: JDK 21)
- A suitable IDE (Eclipse Enterprise Edition for Java) or IntelliJ IDEA
- Latest version of Maven (at this point: Maven 3.9.9)
- A suitable text editor (Notepad ++)
- A utility to extract zip files (7-zip)

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder holds the source code and other related files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` subfolder to build up our applications from scratch and you can always fall back on the complete Eclipse project if you encounter any errors while building up the application.

2 Annotation-based configuration basics

Create a new project: `AnnotationConfigExercise`

Create a package: `com.annotation.exercise`
Place the main application class in it with the name of AnnotationConfigExerciseMainApp

Create an XML configuration file `beansAnnotationExercise.xml` to scan this package:

Create an interface `Payment` with a single method signature `makePayment()`

Create the following classes that implement this interface as well as providing their own unique implementation for `makePayment()`

- class `OnlinePayment` which should be annotated with @Component with a unique name
- class `DirectPayment` which should be annotated with just @Component (no unique name)
- two more classes `MobilePayment` and `BankPayment` with annotations of @Service and @Controller respectively (also no unique name)

In AnnotationConfigExerciseMainApp, initialize the IoC Container using the XML configuration file, and then:

- create a bean using the unique name of the @Component for `OnlinePayment`
- create a bean using the default names for `DirectPayment, MobilePayment` and `BankPayment`

call the implemented interface method on all these beans

3 Annotation-based DI configuration

3.1 Implementing constructor, setter and field injection

Create a new project: `AnnotationDIExercisePt1`

Create a package: `com.di.exercise`
Place the main application class in it with the name of AnnotationDIExerciseMainApp

Create an XML configuration file `beansAnnotationExercise.xml` to scan this package:

Create an interface `Payment` with a single method signature `makePayment()`

-

Create these following classes which should be marked with @Component

- class Supermarket that has a Payment object as a member field/property. This property will be initialized via constructor injection
- class GroceryStore that has a Payment object as a member field/property. This property will be initialized via setter injection
- class WetMarket that has a Payment object as a member field/property. This property will be initialized via field injection

Each of these classes should have a method `doBusiness`, which in turn invokes `makePayment` on their respective `Payment` member properties

Create another class `DirectPayment` that provides a suitable basic implementation for the method in `Payment.` Annotate this class with just @Component (no unique name)

In AnnotationDIExerciseMainApp, initialize the IoC Container using the XML configuration file, and then:

- Create a bean from the Supermarket class and call doBusiness on this bean
- Create a bean from the GroceryStore class and call doBusiness on this bean
- Create a bean from the WetMarket class and call doBusiness on this bean

Run AnnotationDIExerciseMainApp to verify the results are as expected

3.2 Choosing between multiple candidate beans to initialize

Create a new project: `AnnotationDIExercisePt2` which is a copy from `AnnotationDIExercisePt1` and extend from the code there

Create another class `OnlinePayment` that provides a suitable basic implementation for the method in `Payment.` Annotate this class with just @Component (no unique name)

Refactor all 3 existing classes: Supermarket and GroceryStore to initialize their Payment object member field/property through via field injection (just like WetMarket)

- For the following classes, initialize their Payment member properties in the following ways:
- For class GroceryStore, use @Qualifier to select the DirectPayment class to initialize its Payment member property
- For class Supermarket and WetMarket, use @Primary to select OnlinePayment class to initialize the Payment member properties

Create a properties file: `store.properties` with this key-value pair content

```
location=Petaling Jaya
rating=3
```

and modify `beansAnnotationExercise.xml` to provide access to this properties file

Create a new class ConvenienceStore which has two member fields/properties that are initialized from these properties file and provides a method to display these two properties

In AnnotationDIExerciseMainApp, create a bean from Convenience store and call this method to display these two properties

Run AnnotationDIExerciseMainApp to verify the results are as expected