Spring Core Workshop
Lab 2

1 Lab setup

Make sure you have the following items installed

- Latest version of JDK 8 / 11 (note: labs are tested with JDK 8 but should work on JDK 11 with no or minimal changes)
- Eclipse Enterprise Edition for Java (or a suitable alternative IDE for Enterprise Java)
- Latest version of Maven
- A suitable text editor (Notepad ++)
- A utility to extract zip files

In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder just holds the source code 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

The source code for this lab is found in `Annotation-Config-Basics/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.
Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select Copy and then right click in any empty space in the Explorer and select Paste.

For the new project name, type: `AnnotationConfigBasics`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

2.1 Using explicit component name

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.annotation`

```
SwimmingExercise.java
Exercise.java
AnnotationConfigBasicMainApp.java
```

Open and right click on `AnnotationConfigBasicMainApp` and select Run As -> Java Application. Verify that the correct bean has been created and its output logged to the console correctly.

2.2 Using default component name

Create the following changes:

```
SwimmingExercise-v2.java
AnnotationConfigBasicMainApp-v2.java
```

Open and right click on `AnnotationConfigBasicMainApp` and select Run As -> Java Application. Verify that the correct bean has been created and its output logged to the console correctly.

3 Annotation-based DI

The source code for this lab is found in `Annotation-Config-DI/changes` folder.

We can create a Maven project from scratch, or we can make a copy from any of the existing Maven projects.

Choose any previous Maven lab project to make a copy from, for e.g.: `XMLConfigConstructorDI`

In the Project Explorer, right click on `XMLConfigConstructorDI`, select Copy and then right click in any empty space in the Explorer and select Paste.

For the new project name, type: `AnnotationConfigDI`

Replace the contents of the `pom.xml` in the project with `pom.xml` from `changes`. Right click on the project, select Maven -> Update Project, and click OK.

Delete all the packages and files in `src/main/java` and `src/main/resources`. We will start populating the project from scratch.

3.1 Constructor injection

Place `beansDefinition.xml` into `src/main/resources`

Create the following classes in `com.workshop.annotation`

```
SwimmingExercise.java
Exercise.java
CollegeStudent.java
Student.java
AnnotationConfigDIMainApp.java
```

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Verify that the correct bean has been created and its output logged to the console correctly.

3.2 Setter / method injection

Make the following changes:

`CollegeStudent-v2.java`

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Verify that the correct bean has been created and its output logged to the console correctly.

To demonstrate method injection, make the following changes:

`CollegeStudent-v3.java`

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. There are 3 methods (including the setter) which have been marked with @Autowired. All these methods are called in a random sequence after the default constructor is executed. Try running the application multiple times to confirm this.

3.3 Field injection

Make the following changes:

`CollegeStudent-v4.java`

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Verify that the correct bean has been created and its output logged to the console correctly.

3.4 Problems with multiple candidate beans

Make the following changes:

`CyclingExercise.java`
`JoggingExercise.java`

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Notice the exception regarding multiple matching beans (double click on the Console view to enlarge it for a better view)

```
org.springframework.context.support.AbstractApplicationContext refresh
WARNING: Exception encountered during context initialization - cancelling refresh attempt:
org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name
'collegeStudent': Unsatisfied dependency expressed through field 'myExercise'; nested exception is
org.springframework.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type
'com.workshop.annotation.Exercise' available: expected single matching bean but found 3:
cyclingExercise,joggingExercise,swimmingExercise
```

3.5 Using autowiring by name to select between multiple candidate beans

Make the following changes:

`CollegeStudent-v5.java`

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Verify that the correct bean has been created and its output logged to the console correctly.

3.6 Using @Qualifier to select between multiple candidate beans

Make the following changes:

`CollegeStudent-v6.java`

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Notice that the bean is now instantiated with the correct dependency due to the use of the @Qualifier annotation.

When working with constructor injection (as opposed to field injection), @Qualifier must be applied to the specific arguments in the constructor method signature.

Make the following changes:

`CollegeStudent-v7.java`

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Notice that the bean is now instantiated with the correct dependency due to the use of the @Qualifier annotation.

3.7 Using @Primary to give higher preference to a bean

Make the following changes:

```
CollegeStudent-v8.java
CyclingExercise-v2.java
```

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Notice that the bean is now instantiated with the correct dependency due to the use of the @Primary annotation.

3.8 Injecting values from a properties file with @Value

Place `highSchool.properties` into `src/main/resources`

Make the following changes:

```
HighSchoolStudent.java
AnnotationConfigDIMainApp-v2.java
beansDefinition-v2.xml
```

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. Notice that the correct values are read for HighSchoolStudent's fields.

3.9 Marking optional dependencies with @Autowired(required = false)

Make the following changes:

```
Study.java
HighSchoolStudent-v2.java
```

Open and right click on `AnnotationConfigDIMainApp` and select Run As -> Java Application. The application runs fine even if there is no implementation available for the myStudy dependency in HighSchoolStudent, as this is marked as optional. In HighSchoolStudent, change

```
@Autowired(required = false)
private Study myStudy;
```

to

```
@Autowired
```

```
private Study myStudy;
```

and this time attempting to run the application results in an error, as the IoC container will now throw an exception if it is not able to find an implementation for the given dependency.