Spring Core Workshop
Lab 1

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 Demonstrating IoC and DI

The source code for this lab is found in `Basic-Concepts/changes` folder.

Switch to Java Perspective.

Create a new Java project: `DemoBasicConcepts`

Create a new package `com.workshop.original`

Create these 3 original Java files in this package.

```
SwimmingExercise.java
JoggingExercise.java
Student.java
```

In `Student.java`, right click and select Run as -> Java Application

What happens if SwimmingExercise wants to change its implementation of doSwimming ?

```
SwimmingExercise-v2.java
Student-v2.java
```

What happens if Student wants to do Jogging instead?

```
Student-v3.java
```

Create a new package `com.workshop.useinterface`

Create these 4 original Java files in this package.

```
Exercise.java
JoggingExercise.java
Student.java
SwimmingExercise.java
```

What happens if SwimmingExercise wants to change its implementation of doSwimming ?

```
SwimmingExercise-v2.java
```

What happens if Student wants to do Jogging instead?

```
Student-v2.java
```

Demonstrating Inversion of control (IoC)

```
Student-v3.java
```

3 XML-based configuration basics

The primary issue with creating an initial Spring project is to ensure that the relevant Spring module JARs are on the build path of our application. There are two main ways to ensure this:

1) Create a basic Java project. Download and include relevant Spring module JARs on the build path of our project

2) Create a basic Maven project. Specify dependencies for relevant Spring modules into POM.xml

The source code for both approaches is found in `XML-Config-Basics/changes` folder.

We will start with the first approach.

3.1 Basic Java Project

Switch to Java SE perspective.

Create a new Java project: `XMLConfigWithJARs`

In the `src` folder, create a file `beansDefinition.xml`

Create a new package: `com.workshop.configxml`

In this package, create 5 classes:

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
XMLConfigBasicMainApp.java
```

Notice that there is a syntax error registered on `XMLConfigBasicMainApp` as the relevant Spring module classes are not on the build class path yet.

In the project, create a new folder `lib`.

If you haven't already done so, download the latest Spring release distribution from the download link given in the workshop installation instructions and unzip into a suitable folder.

Copy and paste the following JAR files from the `lib` folder into the `lib` folder of your project.

- `spring-aop-x.y.z.jar`
- `spring-beans-x.y.z.jar`
- `spring-context-x.y.z.jar`
- `spring-core-x.y.z.jar`
- `spring-expression-x.y.z.jar`
- `spring-jcl-x.y.z.jar`

On the project, select Properties -> Java Build Path
Right click Libraries Tab, select Add JARs. Select all the 6 JAR files that you just pasted into `lib`. Click Apply and Close.
Notice that you can expand any of these JAR files to see its contents in the Referenced Libraries entry in the Package Explorer.

Right click on `XMLConfigBasicMainApp` and select Run As -> Java Application. Verify that the correct bean is created and its console log displayed in the Console view.

Change the contents of `beansDefinition.xml` to reflect different classes for `favoriteExercise`, for e.g.

```
<bean id="favouriteExercise"
    class="com.workshop.configxml.CyclingExercise">
</bean>
```

```
<bean id="favouriteExercise"
    class="com.workshop.configxml.JoggingExercise">
</bean>
```

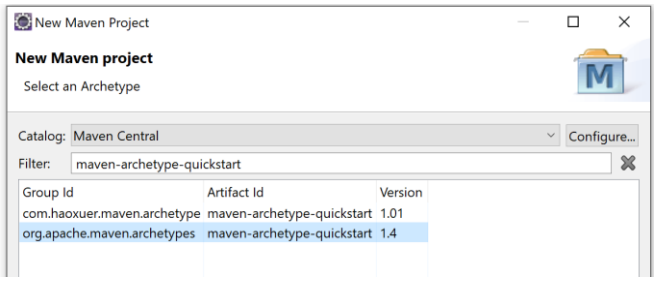
And again run `XMLConfigBasicMainApp` to verify that the correct bean is instantiated based on the console output to the screen.

We now continue with the second approach

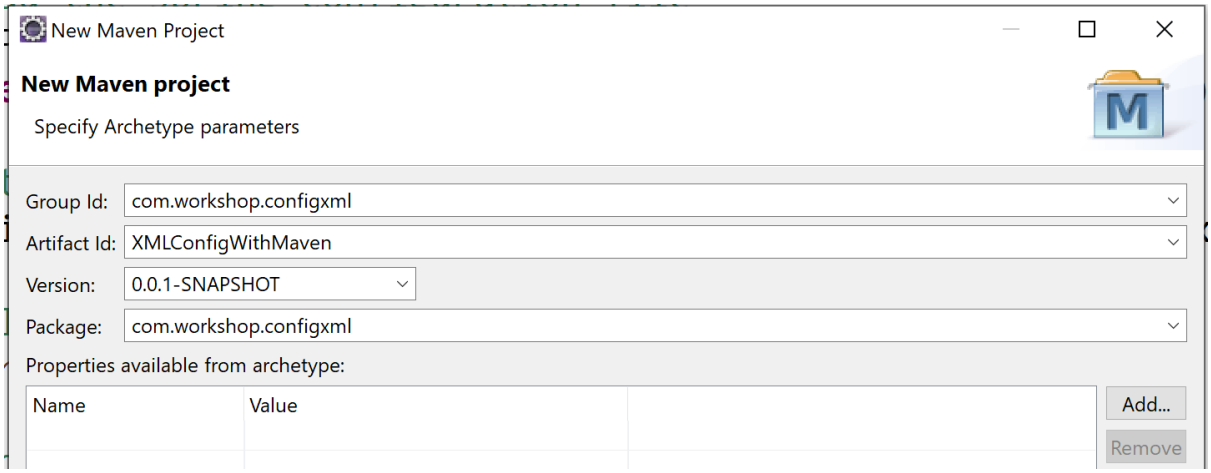
3.2 Basic Maven Project

Switch to Java EE perspective.

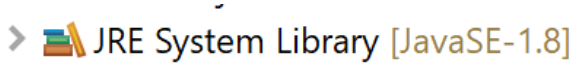
Start with File -> New -> Maven Project. Select Next and type `maven-archetype-quickstart` in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes available at Maven Central. Select the entry with group id: `org.apache.maven.archetypes` and click Next



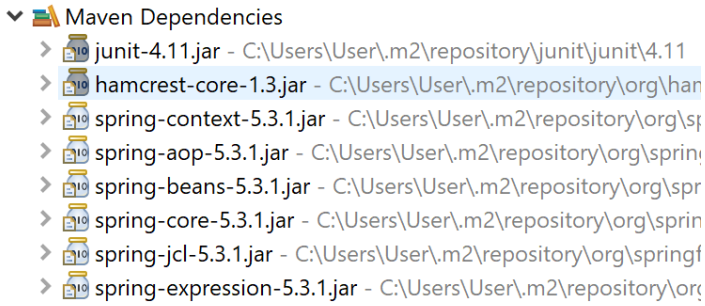
Enter in the following details and click Finish.



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. You should see the JRE system library entry in the project list update to JavaSE-1.8.

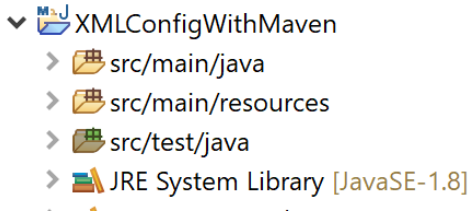


You should also the following JARs automatically added as Maven dependencies



Right click on project and select New -> Folder. Create a new folder named `resources` in `src/main`

Right click on project and select Properties. Select Java Build Path and click on the Source tab. Select Add Folder. Tick on the checkbox for `src/main/resources` and click ok. Click Apply and Close. You should now see this folder being registered in the entries below the project name (all these entries indicate folders which are on the application build path).



In `src/main/java`, there should already be a package: `com.workshop.configxml`

Copy all the previous 5 classes from the previous project `XMLConfigWithJars/src` and place them in the same package `com.workshop.configxml` in this Maven project.

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
XMLConfigBasicMainApp.java
```

Copy `beansDefinition.xml` the previous project `XMLConfigWithJars/src` and paste it into `src/main/resources` in this Maven project

Open and right click on `XMLConfigBasicMainApp` and select Run As -> Java Application. Verify that the correct bean is created and its console log displayed in the Console view.

Change the contents of `beansDefinition.xml` to reflect different classes for `favoriteExercise`, for e.g.

```
<bean id="favouriteExercise"
    class="com.workshop.configxml.CyclingExercise">
</bean>
```

```
<bean id="favouriteExercise"
    class="com.workshop.configxml.JoggingExercise">
</bean>
```

And again run `XMLConfigBasicMainApp` to verify that the correct bean is instantiated based on the console output to the screen.

We can configure the IoC container `ClassPathXmlApplicationContext` to read from more than one XML configuration file at once in order to obtain bean definitions. Let's demonstrate this:

Make the following changes:

- Add `backupDefinition.xml` to `src/main/resources`
- Make the change `XMLConfigBasicMainApp-v2.java`

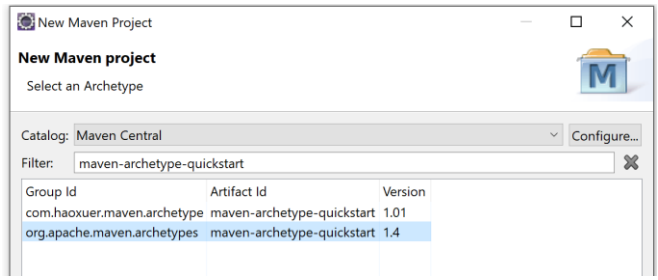
4 XML-based constructor DI

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

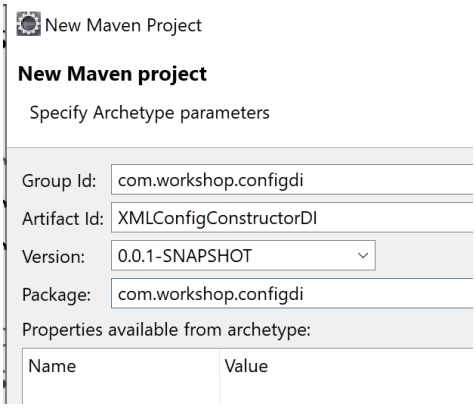
Switch to Java EE perspective.

Start with File -> New -> Maven Project. Select Next and type `maven-archetype-quickstart` in the filter. Eclipse may freeze temporarily while it attempts to filter through all the archetypes

available at Maven Central. Select the entry with group id: `org.apache.maven.archetypes` and click Next



Enter in the following details and click Finish.



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.

Right click on project and select New -> Folder. Create a new folder named `resources` in `src/main`

Right click on project and select Properties. Select Java Build Path and click on the Source tab. Select Add Folder. Tick on the checkbox for `src/main/resources` and click ok. Click Apply and Close.

There are 2 approaches to specify constructor injection:

- Nest the child bean within the parent bean as a child element of the of the parent bean using the **<constructor-arg>** element
- Define the child bean separately and then reference it via the **ref** attribute of the **<constructor-arg>** element

4.1 Approach 1: Nest child bean approach

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

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

Create the following classes:

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
Student.java
HighSchoolStudent.java
CollegeStudent.java
XMLConfigConstructorDIMainApp.java
```

Open and right click on `XMLConfigConstructorDIMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

4.2 Approach 2: Reference child bean separately

Make the following changes:

```
beansDefinition-v2.xml
XMLConfigConstructorDIMainApp-v2.java
```

Right click on `XMLConfigConstructorDIMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

5 XML-based setter DI

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

We can create a Maven project from scratch, or we can make a copy from an existing one.
We will make a copy of the previous lab project: `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: `XMLConfigSetterDI`

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.

There are 2 approaches to specify setter injection

- Nest the child bean within the parent bean as a child element of the of the parent bean using the **<property>** element
- Define the child bean separately and then reference it via the **ref** attribute of the **<property>** element

5.1 Approach 1: Nest child bean approach

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

Create the package `com.workshop.setterdi` and place these classes in it:

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
Student.java
HighSchoolStudent.java
CollegeStudent.java
XMLConfigSetterDIMainApp.java
```

Open and right click on `XMLConfigSetterDIMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

5.2 Approach 2: Reference child bean separately

Make the following changes:

```
beansDefinition-v2.xml
XMLConfigSetterDIMainApp-v2.java
```

Right click on `XMLConfigSetterDIMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

6 XML-based setter DI - literal values

The source code for this lab is found in `XML-Config-Setter-LiteralValues-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: `XMLConfigSetterLiteralValues`

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.

There are 2 approaches to specify setter injection with literal values

- Specify literal values directly in XML configuration file
- Load values from an external property file

6.1 Approach 1: Specify literal values directly in XML configuration

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

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

```
SwimmingExercise.java
JoggingExercise.java
CyclingExercise.java
Exercise.java
Student.java
HighSchoolStudent.java
CollegeStudent.java
XMLConfigSetterLiteralValuesMainApp.java
```

Open and right click on `XMLConfigSetterLiteralValuesMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.

6.2 Approach 2: Load values from an external property file

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

Make the following changes:

```
beansDefinition-v2.xml
XMLConfigSetterLiteralValuesMainApp-v2.java
```

Open and right click on `XMLConfigSetterLiteralValuesMainApp` and select Run As -> Java Application. Verify that the correct parent and child bean have been created and their output logged to the console correctly.