Enterprise Java with Spring Spring Core Lab 1

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)

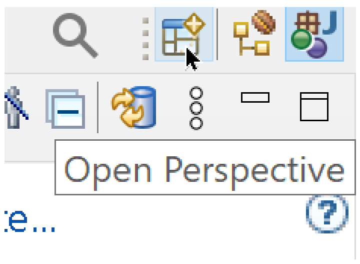
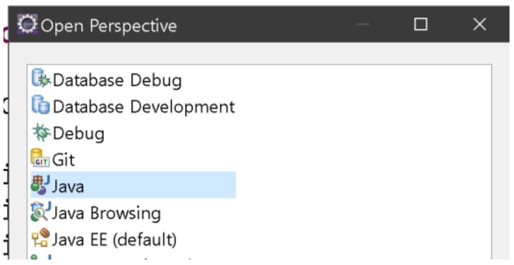
In each of the main lab folders, there are two subfolders: `changes` and `final`. The `changes` subfolder holds the source code and other related files for the lab, while the `final` subfolder holds the complete Eclipse project starting from its project root folder. We will use the code from the `changes` 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 Demonstrating IoC and DI

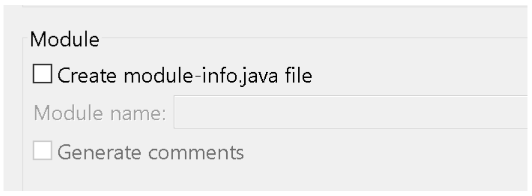
2.1 Demonstrating tightly coupled implementation

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

Switch to Java SE perspective, either using the icons at the upper right hand corner or via the menu: Window -> Perspective -> Open Perspective

	 Open Perspective Database Debug Database Development Debug Git Java Java Browsing Java EE (default)

Create a new Java project (File -> New -> Java Project).
For the project name, type: `BasicConcepts`
Ensure that the execution environment JRE is set to the correct version of Java that you have either installed locally or set as the JRE for Eclipse. Make sure to uncheck the Create module-info.java file option.


Module
☐ Create module-info.java file
Module name:
☐ Generate comments

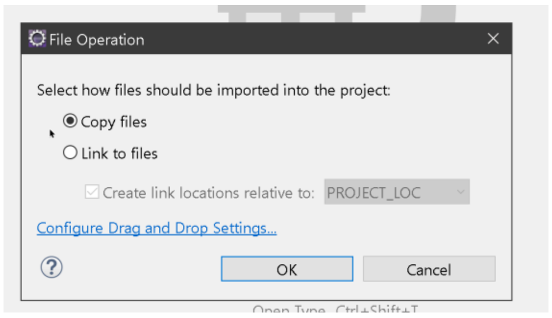
Then click Finish.

Create a new package (Right click on the project name, select New -> Package): `com.workshop.original`

Place these files from the same package name in `changes` into this new package in `src`:

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

You can drag and drop them, Eclipse will prompt you on whether you wish to copy or link these files; select Copy.


File Operation
Select how files should be imported into the project:
● Copy files
○ Link to files
☑ Create link locations relative to: PROJECT_LOC
Configure Drag and Drop Settings...
OK Cancel

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

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

Make the following changes to these files in `src` from the same package name in `changes`:

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

Run Student again to verify it works.

What happens if Student wants to do Jogging instead?

Make the following changes to these files in `src` from the same package name in `changes`:

```
Student-v3.java
```

Run Student again to verify it works.

Here we can see that the doSomeExercises method in Student is tightly coupled to the required functionality provided by methods in both SwimmingExercise and JoggingExercise (which are now dependencies for Student). In this situation, any changes in the methods of these 2 dependencies will require a change in Student as well, which makes maintaining the code base more difficult.

2.2 Using interfaces to achieve loose coupling

Create a new package `com.workshop.useinterface`

Place these files from the same package name in `changes` into `src`:

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

Run Student again to verify it works.

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

Make the following changes to these files in `src` from the same package name in `changes`:

```
SwimmingExercise-v2.java
```

Run Student again to verify it works. Notice that we did not need to make any changes to Student as we needed to in the previous example.

What happens if Student wants to do Jogging instead?

Make the following changes to these files in `src` from the same package name in `changes`:

```
Student-v2.java
```

Run Student again to verify it works. Notice that although we needed to modify Student to implement this change, the code change was limited to the single line of code in the constructor while doSomeExercises method did not need to be changed.

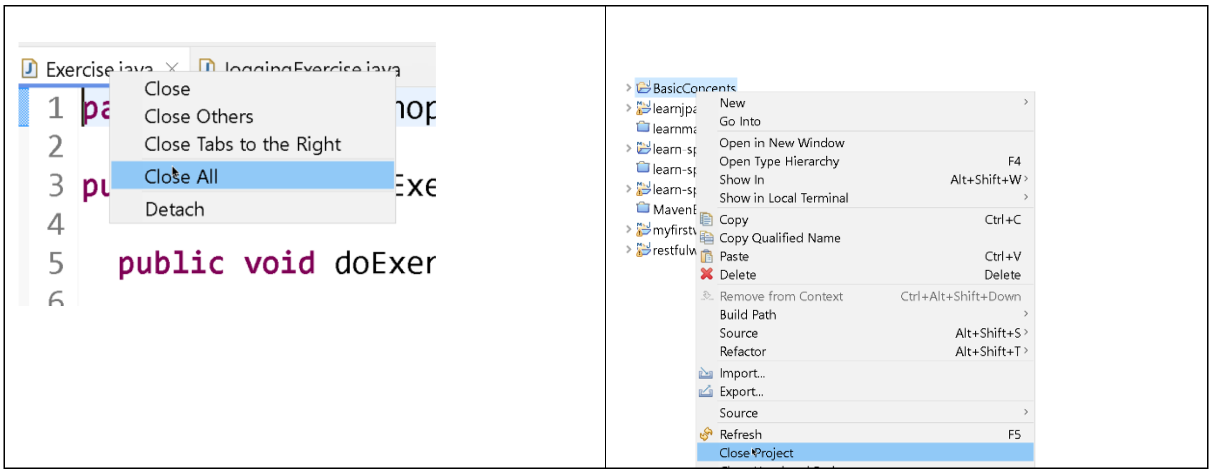
2.3 Demonstrating basic concept of Inversion of control (IoC)

Make the following changes to these files in `src` from the same package name in `changes`:

`Student-v3.java`

Run Student again to verify it works. This provides a basic demonstration of how IoC can be implemented manually by having the code that instantiates objects (in this case from the Student class) providing the dependency objects into the constructor of the original class.

When you are done, you can close all the files from this project as well as the project itself in the IDE to prevent accidentally opening the wrong files in future projects.



3 Setting up a Java Spring project with XML configuration

The primary issue with creating a Java project using the Spring framework is to ensure that the relevant Spring module JARs are on the build path of our application. There are two ways to accomplish this:

1) Create a basic Java project. Download and include relevant Spring module JARs on the build path of our project. This is almost never done in a real-life project since the 2nd approach using Maven will accomplish the same result in a faster and more reliable way as we have already seen in the previous Maven lab sessions.

2) Create a basic Maven project. Specify dependencies for relevant Spring modules into the project POM.xml. Java Spring applications are almost nearly always built as a Maven project.

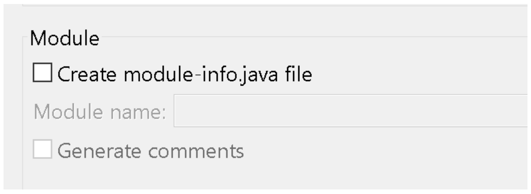
The source code for both approaches is found in `XML-Config-Basics/changes` folder.

We will start with the first approach just to revise and clarify the concept of ensuring that JARs containing the packages and libraries that our application requires (the dependencies) must be on the build path of the application.

3.1 Basic Java Project

Switch to Java SE perspective.

Create a new Java project (File -> New -> Java Project).
For the project name, type: `XMLConfigWithJARs`
Ensure that the execution environment JRE is set to the correct version of Java that you have either installed locally or set as the JRE for Eclipse. Make sure to uncheck the Create module-info.java file option.



Module
☐ Create module-info.java file
Module name:
☐ Generate comments

Then click Finish.

Place these files from `changes` into `src`:

```
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`.

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

- `commons-logging-1.3.4.jar`
- `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`

You can download these JAR files manually yourself by searching in the appropriate subcategories in the main Spring framework site at the Maven repository site:
https://mvnrepository.com/artifact/org.springframework
https://mvnrepository.com/artifact/commons-logging

On the project, select Properties -> Java Build Path
Select the Libraries Tab, then select Classpath. Click 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. The Spring framework specific class that we are using (`ClassPathXmlApplicationContext`) is found in the package `org.springframework.context.support` in the JAR `spring-context-x.y.z.jar`

The previously flagged syntax errors in `XMLConfigBasicMainApp` should now have disappeared.

Right click on this file and select Run As -> Java Application.

The Spring IOC Container ClassPathXmlApplicationContext in `XMLConfigBasicMainApp`, will initialize the bean in accordance with the XML configuration specifications in `beansDefinition.xml`
Once it starts up, it will read all the bean definitions in the XML configuration, whereupon these beans are said to be registered with the container. We then subsequently retrieve the registered beans using the `getBean` method of ClassPathXmlApplicationContext

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>
```

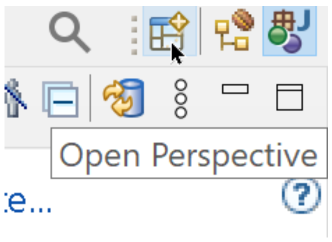
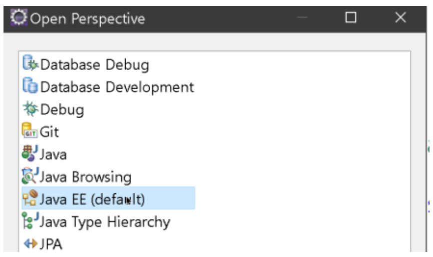
And again run `XMLConfigBasicMainApp` to verify that the correct bean is instantiated based on the console output to the screen.

Close all the open editor tabs to prepare for the next project creation.

NOTE: Manually downloading and referencing Spring module JARs on the build path of the project as we have done here almost never done in a real-life project since the 2nd approach described next using Maven will accomplish the same result in a faster and more reliable way as we have already seen in the previous Maven lab sessions.

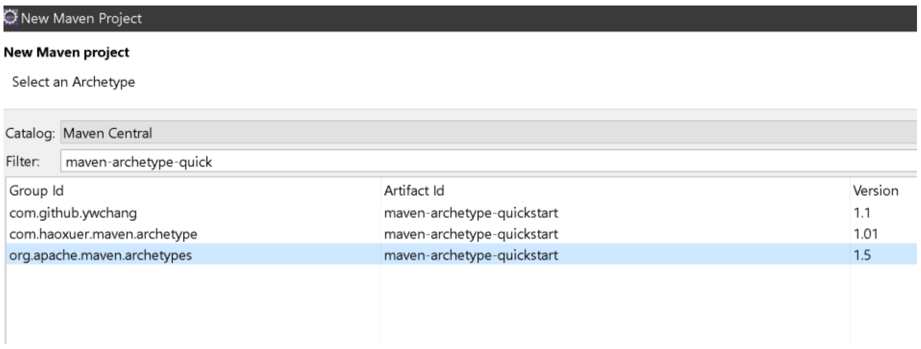
3.2 Basic Maven Project

Switch to Java EE perspective, either using the icons at the upper right hand corner or via the menu: Window -> Perspective -> Open Perspective

	 Open Perspective Database Debug Database Development Debug Git Java Java Browsing Java EE (default) Java Type Hierarchy JPA

Start with File -> New -> Maven Project. Make sure the checkbox for the Use Default Workspace location is ticked.

Select Next and choose the Maven Central catalog. 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



New Maven Project

New Maven project

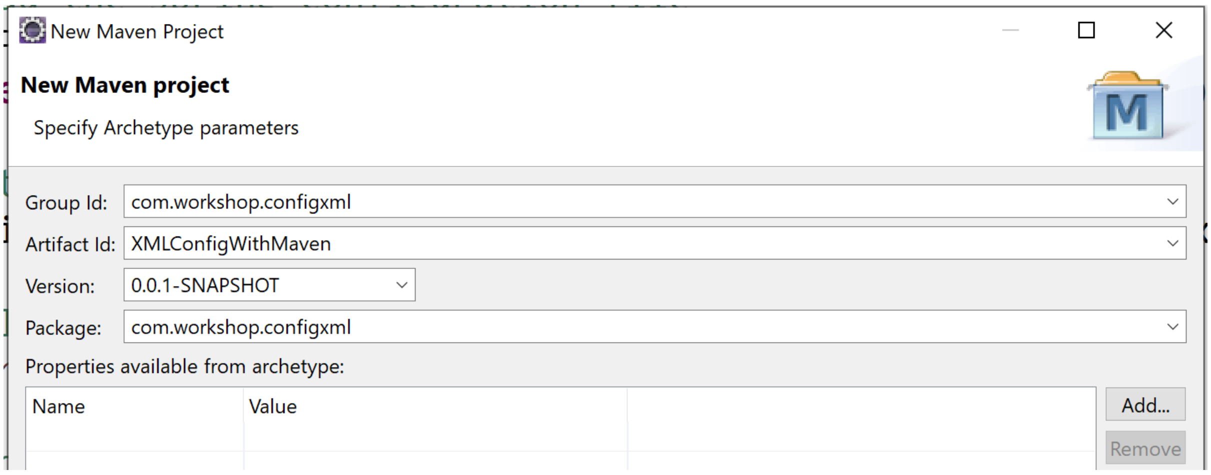
Select an Archetype

Catalog: Maven Central
Filter: maven-archetype-quick

Group Id	Artifact Id	Version
com.github.ywchang	maven-archetype-quickstart	1.1
com.haoxuer.maven.archetype	maven-archetype-quickstart	1.01
org.apache.maven.archetypes	maven-archetype-quickstart	1.5

Enter in the following details and click Finish.

GroupId: `com.workshop.configxml`
ArtifactId: `XMLConfigWithMaven`
Version: `0.0.1-SNAPSHOT`
Package: `com.workshop.configxml`


New Maven Project

New Maven project

Specify Archetype parameters

Group Id: com.workshop.configxml
Artifact Id: XMLConfigWithMaven
Version: 0.0.1-SNAPSHOT
Package: com.workshop.configxml
Properties available from archetype:

Name	Value

If this is the first time you are generating a Maven project, the creation of this project will download the required artifacts from Maven Central Repository (https://repo.maven.apache.org/maven2/), and messages to that effect will appear in the Console view in the stack of views at the bottom.

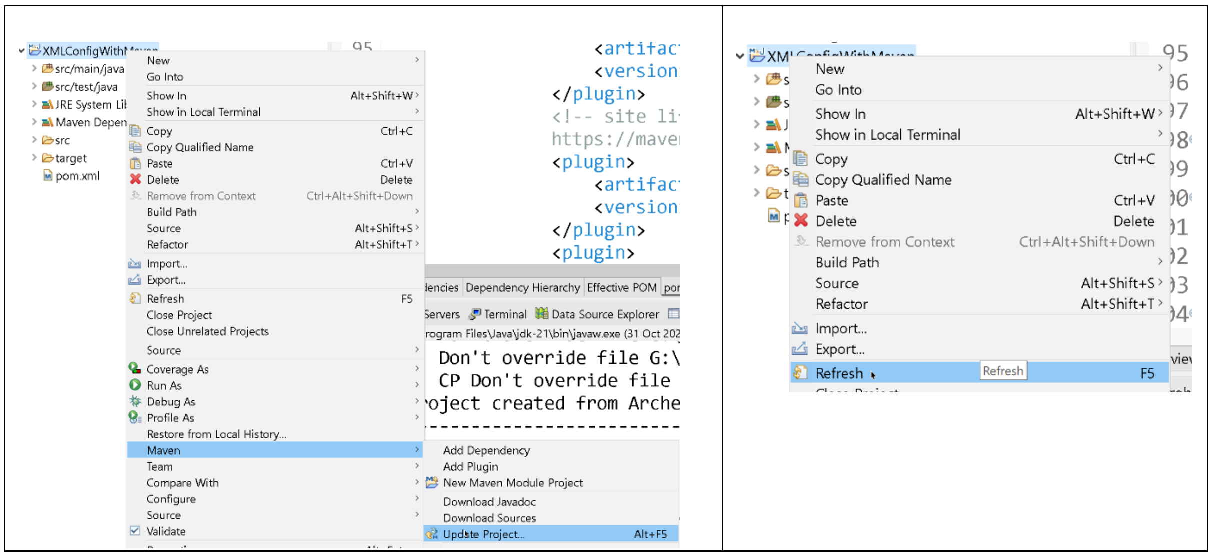
The Maven project creation will by default run interactively, so at some point, you will be asked to confirm the properties configuration in the Console view:

```
Confirm properties configuration:
javaCompilerVersion: 17
junitVersion: 5.11.0
groupId: com.workshop.basic
artifactId: XMLConfigWithMaven
version: 0.0.1-SNAPSHOT
package: com.workshop.basic
Y
```

Click in the Console view next to the Y and press enter. This should complete the installation process.

Replace the contents of the `pom.xml` in the newly generated project with `pom.xml` from `changes`.

Perform a Maven Project update followed by a refresh of the project to ensure the relevant Maven dependency JARs are downloaded.



```
<artifac
<version
</plugin>
<!-- site li
https://mave
<plugin>
<artifac
<version
</plugin>
<plugin>
```

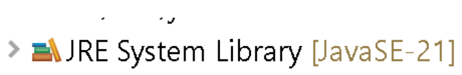
```
rogram Files\Java\jdk-21\bin\javaw.exe (31 Oct 202
Don't override file G:\
CP Don't override file
oject created from Arche
------------------------
```


We have made the following changes to the standard autogenerated POM:

- Updated the Java version to comply with the one in the IDE / JDK (double check on this)
- Added in the `spring-context` dependency at the latest version
- Added in the `commons-logging` dependency

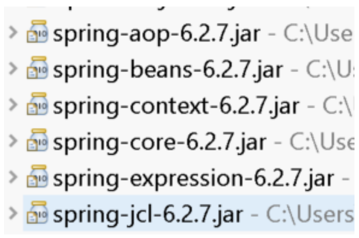
The latest version of Spring-Context should typically align with the core Spring Framework latest version.

You should see the JRE system library entry in the project list update to the version specified in the POM.



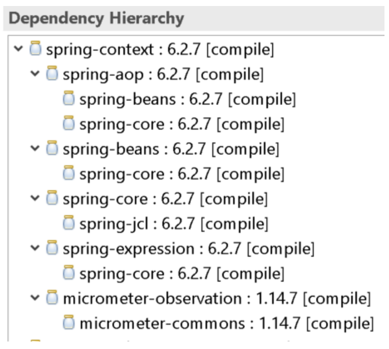
```
> JRE System Library [JavaSE-21]
```

You should also the see all the JARs that we had added manually in the previous lab now automatically added as Maven dependencies



```
> spring-aop-6.2.7.jar - C:\Use
> spring-beans-6.2.7.jar - C:\U
> spring-context-6.2.7.jar - C:\
> spring-core-6.2.7.jar - C:\Use
> spring-expression-6.2.7.jar -
> spring-jcl-6.2.7.jar - C:\Users
```

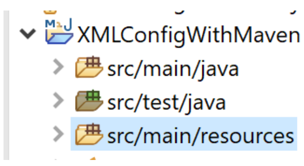
If you check in the Dependency Hierarchy tab in the IDE, you will notice that the single `spring-context` direct dependency itself has a number of transitive dependencies, which themselves in turn have transitive dependencies. Also notice that the multiple instances of transitive dependencies (such as spring-core) all have identical versions, so there is no need for Maven to perform dependency mediation and also no issue of the application code not working properly.



```
Dependency Hierarchy
v spring-context : 6.2.7 [compile]
  v spring-aop : 6.2.7 [compile]
      spring-beans : 6.2.7 [compile]
      spring-core : 6.2.7 [compile]
  v spring-beans : 6.2.7 [compile]
      spring-core : 6.2.7 [compile]
  v spring-core : 6.2.7 [compile]
      spring-jcl : 6.2.7 [compile]
  v spring-expression : 6.2.7 [compile]
      spring-core : 6.2.7 [compile]
  v micrometer-observation : 1.14.7 [compile]
      micrometer-commons : 1.14.7 [compile]
```

Right click on the project name and select New -> Source Folder. Name the source folder: `src/main/resources`

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).



If this does not happen, right click on the newly created folder, select Build Path -> Use as Source Folder. This will add the contents of this folder to the build path.

In `src/main/java`, there should already be a package: `com.workshop.configxml`
You can delete the autogenerated `App.java` in here.

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. You can do this by selecting all of these classes, click Copy and then paste into the destination package

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

Copy `beansDefinition.xml` from 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.

XML-based bean configuration was the original method for configuring beans in the earlier versions of the Spring framework. Since then, the framework has evolved to offer several modern alternatives that are more concise and type-safe. We will look at the two main types together: annotation-based configuration and Java-based configuration next.

Close all the open editor tabs to prepare for the next project creation.

Close this project to prevent confusion with the next project you are going to create (right click on project entry and select Close Project)