Spring Labs – Part 1

Lab 0 – Setup

You will need the following to do these labs:

1. Java SE 8
   - http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html
2. Eclipse: Eclipse IDE for Java EE Developers
   - http://www.eclipse.org/downloads/
3. Maven
   - https://maven.apache.org/download.cgi

Maven may already be loaded on your machine. This can be checked by:

1. Going to a command prompt and typing **mvn**. This will tell you if it is on the system's path.
2. Open Eclipse and go **Help -> Preferences**. On the left-hand side of the **Preferences** panel, there should be a **Maven** menu. See if there is one that is embedded, or if there is another one that has been selected.
3. The labs have been tested with Maven 3.3.9, however if you get something newer, there should not be much difference as far as we are concerned.

Note: If you are already using Maven for work purposes, let the instructor know. There is a way to separate what you are using for the class from what you are using for work that they can show you.

Lab 1 – Hello Spring!

For this lab, we will make a simple HelloWorld application, using Spring. There will be three classes, the interface, the Spring bean and the application class. We will do two versions of this. One using the `application-config.xml` for configuration, the other using `JavaConfig`.

1. Define a new Maven project.
   - **File** > **New** > **Other** > **Maven** and select **Maven Project**
   - Click **Next** and select the archetype **maven-archetype-quickstart**
   - Select **Next** and fill in as follows:

Group Id:	`com.di`
Artifact Id:	`hello`
Package:	`com.di.helloworld`

   - Select **Finish**

2. Fix the `pom.xml` file to use Spring by replacing the one defined by the wizard with the one found in the **resources** directory.
3. We will make 3 classes – the interface, the bean and the application in the `src/main/java` directory.
   - Put them in the package `com.di.helloworld`
   - Suggested names are `HelloSpring`, `HelloSpringBean` and `MainApplication`
   - The bean should have two methods – `setMessage` and `getMessage`
   - `MainApplication` should get your `ApplicationContext`, call both methods, and print your message to the console
4. Define the `application-config.xml` for this project.
   - Right click on the project and go **New > Other > XML** and select **XML File**
   - Select our project and name the file `application-config.xml`
   - Select **Finish**
5. Move the `application-config.xml` file to be under the `src/main/java` directory.
6. Modify application-config.xml to wire your `HelloSpringBean`.
   - Note: In the **resources** directory, there is a file called `spring-config-template.xml`. It has the main beans tag that you can copy into your `application-config.xml` file
7. Run the application.
8. Copy the **HelloWorld** project and name the new one **HelloWorld2**.
9. Define a `JavaConfig` file to wire your `HelloSpringBean` using Java.
10. Modify `MainApplication` to now use the `JavaConfig` file instead of `application-config.xml`.
11. Run the application.

Lab 2 – Initial Wiring of PhoneBookApplication

In this and many future labs, we are going to work on a **PhoneBook** application. You will be given the three base classes – `Contact.java`, `ContactTable.java` and `ContactRepository.java`. The `ContactTable` will represent your contacts, the `ContactRepository` will be the interface for this.

1. Import the initial project by going to **File > Import > Maven** then selecting **Existing Maven Projects**. Select **Next** then find the **PhoneBookApplication** which should be under your Resources directory. Select **Finish**.
2. Define a new package called `com.di.phonebook.service`. In this package, we will define two new classes – `PhoneService` and `PhoneServiceBean`.

    - The interface should have the same methods in it that `ContactRepository` has
    - The bean should have a constructor that takes the `ContactRepository` as an argument
    - Override the inherited methods to call their corresponding method in `ContactRepository`
3. Write the `JavaConfig` file for this project and place it in a new package called `com.di.phonebook.config`.
4. Write the application class, named whatever you like.
5. Run the application.

Lab 3 – Using Property Files

In unit 2 we saw several different configuration items. We used a few of them with our **PhoneBook** application but using property files isn't appropriate with that example. To get practice with them, we are going to return to our **HelloSpring** application.

1. Make a copy of **HelloSpring2** and call it **HelloSpring3**.
2. Modify this application to read a message from a properties file.
    - The file should be called **messages.properties**
    - The file has one key – **messages.greeting**. You may set its value as you wish
3. Modify `JavaConfig` as appropriate to load the property file and perform component scanning.
4. Modify `HelloSpringBean` so that it the message field will be able to be set as part of the startup of the application.
5. Modify `MainApplication` to no longer call `setMessage`
6. Run the application.