SYSC 4806 – Lab 3: Dependency Injection with Spring

Part I – Accessing Data with JPA using Spring Boot

Go through https://spring.io/guides/gs/accessing-data-jpa/ first (see https://spring.io/guides for a whole slew of very nice and simple tutorials that will be very helpful for your projects). The tutorial uses Spring and the H2 database for JPA support and persistence instead of EclipseLink and SQLite. It also uses the notion of Repositories to provide some built-in methods to access the database via Java (thus reducing the need for JPQL). Finally, you'll notice that there's no persistence.xml file involved (and so you should delete it to avoid any possible undesired interaction and confusion)! All thanks to the magic of Java reflection and convention over configuration! So now let's switch technologies and apply this tutorial to replace the AddressBook persistence work you did in the previous lab, but this time using Spring Data JPA repositories and the in-memory H2 database. It should look much nicer!

When you're done, show your work to the TA, or if you weren't able to, upload a zip file of the work.

Part II – OPTIONAL: no need to show to the TAs, but they can help you

Part I shouldn't take too long to do, so with the time you have left you can attempt this more challenging exercise, which is optional but will get you to understand in more depth what Spring's Dependency injection can do, by applying it in the simpler context of a desktop application that uses Swing for its UI.

So first, in a separate project (you won't be reusing what you do here in future labs) come up with a very simple Swing UI for your AddressBook application, where you can interactively populate a BuddyInfo and save it to your address book. Use MVC principles as per SYSC3110! The gluing together of the various UI components, as well the registration of the View to the Model, the Controller to the View, etc., should happen, as much as possible, in some `main()` method. This is done in order to reduce the dependency of the various components to one another. In the next part, this *configuration* that happens in the `main()` method will be done using Dependency Injection instead.

Now read this tutorial (it is very old, and was rescued by TA John from the wayback machine!) You can skip the environment setup parts, since you're going to use Maven and just add the Spring dependency in the pom.xml file there. The meat of the article starts at the section entitled "Creating the to-do list". It shows how dependency injection can be used to assemble the components of a Swing application. It's not necessarily a great idea to use Spring and XML files for such a purpose (one could assemble these components using plain Java in a separate class), but it will help give you a concrete feel for what DI does. You can skip the last part of the article that talks about the Spring Rich Client Project.

Now to try out what is described in the tutorial: try gradually replacing the "glue" code in your above-mentioned `main()` method (e.g., where you put together the UI and the MVC) with configuration code, using an XML configuration file or the more modern annotations. See how far you can go in replacing code with configuration! Important: don't try to blindly copy-paste what you find in the tutorial!!! Don't try to mimic the UI of the tutorial!!! That will only create frustration.