SYSC 4806 – Lab 3: Dependency Injection with Spring

In this lab we are first going to learn how to do dependency injection for a simple desktop application. Then we are going to use the power of dependency injection to reduce the amount of glue code and manual configuration that we had to do in our AddressBook persistence exercise in the previous lab. We're not going to give you as much detail as before (in part because going through the frustrations is educational), but we'll be around for help if you get stuck!

Part I – Dependency Injection Tutorial: the first thing you need to do is read this tutorial. 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, come up with a very simple Swing UI for your AddressBook application, where you populate a BuddyInfo and save it to your address book. Once you got it working, try replacing the "glue" code (e.g., where you add the subcomponents of the UI to the main frame, and where you register action listeners to the components) with configuration code, whether you use an XML configuration file or the more modern annotations. See how far you can go in replacing code with configuration!

Part II – Accessing Data with JPA using Spring Boot

Now let’s get to a situation where using Spring is actually helpful! Go through this other tutorial 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! All thanks to the magic of Java reflection and convention over configuration! So now let’s switch technologies and apply this tutorial to clean-up the AddressBook persistence work you did in the previous lab, using Spring Data JPA repositories and the in-memory H2 database.

When you’re done, show your work to the TA.