Spring and Hibernate (SPH)

Exercise SPH.2 – Spring Open Session in View

The Setup:

The goal of this exercise is to test Springs open session in view support. Start by downloading the **W2D4-SpringHibernate** project from Sakai. This is my solution to the W2D3-HibernateApp-2 project. Alternately you can also make a copy your own solution and use that.

I've removed the dependencies from the pom.xml (doing so keeps me from having to update the version numbers all the time in every exercise). Add the same dependencies as that were in your W2D3-HibernateApp-2 project, and then also add:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>6.0.8</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-tx</artifactId>
    <version>6.0.8</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>6.0.8</version>
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>6.0.8</version>
</dependency>
```

The Exercise:

a) Web Application with Spring and Hibernate:

Create a WebApplicationInitializer (or web.xml) implementation as shown in the slides for adding spring to a web application (don't include any servlets or filters yet).

Then create a spring configuration (Config.class or springconfig.xml),turn StudentsService and StudentDao into spring bean classes, and inject the StudentDao into the StudentService. Note: if you use XML configuration your config file should be located inside webapp/WEB-INF/ not inside resources.

Configure Hibernate inside of Spring, as shown in the "Combining Spring and Hibernate" slides. The slide showing the Java configuration does not show the import statements needed. Be sure to use the javax.sql.DataSource and the java.util.Properties.

Some common configuration mistakes: in the entityManagerFacotory bean configuration

it specifies the packages that Hibernate should scan in order to find the @Entity classes. Make sure this is set to cs544. Also check that your mysql url has ?useSSL=false.

While you're modifying the Hibernate configuration inside Spring also make sure that you've enabled the format_sql option.

Delete the persistence.xml file and the EntityManagerHelper class. Update the StudentDao to use @PersistenceContext to get a Thread Local entityManager injected into it, and remove the transaction code from StudentService.

Update the Servlet to get the StudentService from the Spring Web Application Context, which it gets from the ServletContext (as shown in the slides). Then run your application to check if everything works.

b) OpenEntityManagerInView Filter:

Currently the DAO retrieves both the student and his courses by using an EntityGraph. You can see this based on the single select it executes inside the Tomcat logs when it retrieves student 12345. Lets switch to using the OpenEntityManagerInView filter to lazy load the collection instead.

Remove the EntityGraph code from the StudentDao, and add the OpenEntityManagerInView to the your WebApplicationInitializer as shown in the S&H OpenEntityManagerInView slides.

Run the application again, and you should see two SELECT statements in the Tomcat log when you retrieve student 12345 this time.

c) Our last step will be to add Spring Transactions.

Add the transactionManager bean to your spring configuration. Then add @Transactional(propagation = Propagation.REQUIRES_NEW) to the StudentService class, and add @Transactional(propagation = Propagation.MANDATORY) to the StudentDao. Be sure to use Spring's version of @Transactional, not javax.transaction.Transactional!

Run your code one last time to check that everything works.