Fidelity INVESTMENTS

MASTERING SPRING AND MYBATIS

EXERCISE MANUAL

This page intentionally left blank.

Table of Contents

This page intentionally left blank.

Chapter 1: Introducing the Spring Framework

Exercise 1.1: A Simple Spring Application

In this exercise, you will examine and run a Spring application. You will then modify the greeter-beans.xml Spring configuration file to control what managed beans Spring will create and use in the application.

Time: 30 minutes
Format: Programming exercise

1. Open the `Greeter` project.

2. Examine the sources in `com.fidelity.greeter`

3. Examine the configuration file `src/main/resources/greeter-beans.xml`
    a. Click the **Source** tab at the bottom of the xml editor window to view and edit the raw XML.

4. Run the Driver as a Java application.
    a. Right-click `Driver.java` and select **Run as Java Application**.
    b. Examine the output from the application.
    c. Is this what you expected?

5. Modify the configuration file `greeter-beans.xml`.
    a. Change the value of the String injected into the AmarilloVisitor.
    b. Suggested change: Replace "Joe Bob Springsteen" with "Batman".

6. Run the Driver application again.
    a. Do you see the new message displayed?

7. In the package `com.fidelity.greeter`, add a new class that implements the Visitor interface.
    a. Suggested class: WindyCityVisitor.
    b. Note that you will need to define a setName(String name) method even though it is not in the interface.

8. Use dependency injection to set the name and greeting properties.
    a. Suggested change: Set the name to your instructor's name, or your name, or your favorite literary or movie character's name. Or set the name to "Godot" and then wait…
    b. Suggested change: Set the greeting to "Love Da Bears" or another greeting appropriate to the new Visitor you have created.

9. In the file `config/greeter-beans.xml`, configure Spring to create an instance of your new class.

10. Inject your new Visitor instance into the PopupGreeter instead of the AmarilloVisitor.

11. Run the Driver application again.
    a. Do you see the message from your new Visitor class?

12. Notice the code in the main method did not change even though one of the two classes being used was changed.

Exercise 1.2: Spring with Annotations

In this exercise, you will use annotations to provide Spring configuration information instead of doing this with xml settings.

Time: 30 minutes
Format: Programming exercise

1. Open the `GreeterAnnotations` project.

2. Examine the sources in `com.fidelity.greeter`

3. Examine the configuration file `src/main/resources/greeter-beans-annotations.xml`

4. Run the Driver application.
    a. Is the message output what you expected?

5. Change the value of the String injected into the AmarilloVisitor using the `@Value` annotation.
    a. Suggested change: Set the name to Superman, leave the greeting unchanged.

6. Run the Driver application again.
    a. Does the new name appear in the message output?

7. In the package `com.fidelity.greeter`, add a new class that implements the Visitor interface.
    a. Suggested change: Create the class BostonVisitor.

8. Use dependency injection to set the name and greeting properties.
    a. Suggested change: Set the name to "Abby Johnson" and the greeting to "Happy to Meet You".
    b. Set `@Component("bostonVis")`.

9. Run the Driver application again.
    a. Do you see the message that you expected to be displayed?
    b. *Note:* Because now two classes implement the Visitor interface, there is a problem. Bravely march on to correct this issue in the following steps.

10. Spring uses type-based injection in this instance.

11. We need a way of distinguishing which implementation of the Visitor interface to use.
    a. `@Qualifier` allows us to do this.
    b. *Hint:* `@Qualifier("bostonVis")`.
    c. *Hint:* See slide 1-21 for an example.

12. Use the above annotation to specify which Visitor bean to use.

13. Run the Driver application.
    a. Is the output what you expect?

Exercise 1.3: Java Configuration

In this exercise, you will work with a Spring application that is complete without any xml configuration file. All of the Spring configuration information will be provided in Java files and annotations.

Time: 20 minutes
Format: Programming exercise

1. Open the `GreeterJavaConfiguration` project.

2. Run the Driver application.

3. Examine the sources in `com.fidelity.greeter` in particular:
    a. `AppConfig.java` and `Driver.java`

4. In the package `com.fidelity.greeter`, add a new class that implements the Visitor interface.
    a. Suggested change: Create the class IndiaVisitor.
    b. Set your name and greeting to be whatever you desire.

5. In the `AppConfig.java` class, add a method that will create a bean of the new Visitor implementation.

6. Make it so that the PopupGreeter uses this second visitor not the original one.

7. Run the Driver application again.
    a. Do you see the output that you expected?

This page intentionally left blank.

Chapter 2: Understanding Spring

Exercise 2.1: Using Spring as a Factory

In this exercise, you will use Spring's ApplicationContext to create some managed beans. You will configure Spring to use dependency injection to create a completely initialized bean for you to use in your application.

You will test your code with JUnit to verify that everything works as expected.

Time: 20 minutes
Format: Programming exercise

1. Open the `Library` project.

2. Examine the `Book` class in the `com.fidelity.business` package.

3. Examine the `BookDao` and `MockBookDao` in the `com.fidelity.integration` package.

4. Examine the `BookService` in the `com.fidelity.services` package.
    a. Notice that the `BookService` has a dependency on a `BookDao`.

5. Complete the Spring configuration defined in `library-beans.xml` in the `src/main/resources` folder.
    a. Define a `BookDao` bean.
    b. Define a `BookService` bean.
    c. Configure Spring to inject a `BookDao` into the `BookService`.

6. Write a JUnit test to verify this all works.
    a. Create the Spring `ApplicationContext` using the `library-beans.xml` file.
    b. Get the `BookService` bean from the `ApplicationContext`
    c. Verify that `queryAllBooks` works correctly.

Exercise 2.2: Dependency Injection with Constructors

In this exercise, you will configure Spring to do dependency injection by using a constructor.

Time: 20 minutes
Format: Programming exercise

1. Open the `Library` project in Eclipse.
    a. This is the project that you worked on in the previous exercise.

2. Modify the `BookService` to include a constructor with a `BookDao` argument.

3. Modify the `library-beans.xml` file to instruct Spring to use this constructor to inject the `BookDao` into the `BookService`.

4. Run the JUnit test to verify this still passes.
    a. No modification should be required for this test to pass.

Exercise 2.3: Integration Testing with Spring

In this exercise, you will use the Spring TestContext Framework to simplify your JUnit tests.

Time: 10 minutes
Format: Programming exercise

1. Open the `Library` project in Eclipse.
    a. This is the project that you worked on in the previous exercise.

2. Modify the `BookServiceTest` to use the Spring TestContext Framework.
    a. *Hint:* you will not need a @BeforeEach, so everything that is done there should be done through annotations.

3. Run the JUnit test to verify this still passes.

4. Write your tests this way from now on.

Exercise 2.4: Create and Access a `HashMap`

In this exercise, you will write Java code that uses a `HashMap` to store birthday information for a collection of people. Your code will add entries into the map and retrieve information from the map.

Time: 20 minutes
Format: Programming exercise

1. Open the `Birthdays` project.

2. In `com.fidelity.birthday`, explore the `Birthday` class.

3. Create a new class named `Driver` that contains a `main()` method.

4. In the main method, create a `HashMap` collection to store the birthdays of five different persons.
    a. Key is String to hold name.
    b. Value is the Birthday for that person.
    c. Use the birthdays in the `Birthday.java` file or make up your own.

5. Write code to display the names and birthdays of all the entries in your map.

Chapter 3: Advanced Spring Configuration

Exercise 3.1: Bean Destroy

In this exercise, you will specify a destroy method that Spring will call before shutting the bean factory down.

Time: 20 minutes
Format: Programming exercise

1. Open the `BeanDestroy` project.

2. Run the exhibits.sql file in SQL Developer against the Scott database.

3. Examine the `ExhibitsDAOJDBCImpl` class.
   a. The connection is established by the `getConnection()` method, which is called from `getExhibits()`.
   b. The connection is only closed if the `close()` method on the DAO is called.

4. To guarantee that Spring will call the `close()` method before the bean factory is shut down, address the TODO items in the project. You can find them in `ExhibitsDaoTest` and `museum-beans.xml`.

5. Run the test to verify the DAO works correctly and that the database connection is closed.

Exercise 3.2: Debugging Spring Configuration Problems

In this exercise, you will correct problems with a Spring application. You will determine what the source of the problems are and correct them.

Time: 20 minutes
Format: Programming exercise

1. Open the `SpringProblems` project.

2. Run the JUnit test.
    a. What error is reported?
    b. Correct that error.

3. Keep working until you get that green bar!

4. Verify that the service works correctly with both DAO beans.
    a. The test should pass using either DAO.

Now, doesn't that feel better?

Chapter 4: Introduction to MyBatis and Spring

Exercise 4.1: Configure MyBatis with Spring

In this exercise, you will use Spring to make working with MyBatis much easier.

Time: 15 minutes
Format: Programming exercise

1. Open the `MyBatisSpring` project.
2. Open the `beans.xml` file.
3. Complete the TODO steps that are listed in the file.
4. Run the test and see that it completes with a green bar.

Exercise 4.2: Query the Database with MyBatis and Spring

In this exercise, you will complete the project that you started in the previous exercise. Working TDD, you will use Spring and MyBatis to query the database for Department information.

Time: 30 minutes
Format: Programming exercise

1. Continue working with the `MyBatisSpring` project from the previous exercise.

2. Modify the test class to use the annotations illustrated in the notes:
   a. `@ExtendWith(SpringExtension.class)`
   b. `@ContextConfiguration("classpath:beans.xml")`

3. Still in the test class, delete the temporary test that is shown there.

4. Declare a field to hold a DAO, mark it with `@Autowired`.
   a. *Note:* For this exercise, there is no interface. Make a variable for the DAO `impl` class and `@Autowire` it. This is not the preferred approach—we would generally use an interface.

5. Also write a test for a method in the DAO that queries the database for all departments.

6. Now implement the method in the DAO. It should delegate execution to an `@Autowired DepartmentMapper`. To do this, you must declare a suitable method in the `DepartmentMapper.java` interface.
   a. *Note:* Use SQLDeveloper to see the names of the columns in the `dept` table. Also, examine `Department.java` to see how to make the `DepartmentMapper.xml` entries.

7. In `DepartmentMapper.xml`, define the database operation that corresponds to the interface method in `DepartmentMapper.java`.

8. Use the test you wrote earlier to check if your query is working correctly.

9. Keep working on it until you get a green bar.

Exercise 4.3: Query for Complex Object Relationships

In this exercise, you will modify the Department class to define a one-to-many relationship with the Employee class. You will then define the one-to-many relationship for MyBatis to use in querying for Department and Employee records from the database.

Time: 45 minutes
Format: Programming exercise

1. Continue working with the `MyBatisSpring` project from the previous exercise.

2. Modify the Department class by defining a one-to-many relationship with the Employee class.
    a. In this exercise, we will leave the department referred to by Employee as an id rather than a Department object.

3. Write a JUnit test for a method that queries the database for all Departments and all of the Employees in each department.
    a. The method should return a collection of Department objects.
    b. Each Department object should contain a collection of all the Employees in that Department

4. Implement the method, making changes to the mapper and DAO, as needed.

This page intentionally left blank.

Chapter 5: Working Effectively with MyBatis

Exercise 5.1: DML with MyBatis and Spring

In this exercise, you will use MyBatis and Spring to perform DML operations on the database.

Time: 60 minutes
Format: Programming exercise

1. Open the `MyBatisWithSpringDML` project.

2. The project currently provides the capability of querying the database for Department and Employee information.

3. Confirm that it is working correctly by running the tests.

4. Your task now is to provide the capability of inserting a new Department into the database and to update an existing Department.

5. Start by writing a JUnit test for an insert method. Set up the test with transactional control so that it automatically rolls back. Consider what assertions you need to apply to confirm the method is working correctly.

6. Now implement the method.

7. When the insert method is complete, write a similar test for the update.

8. And implement the update method itself.

Optional Exercise 5.2: Using an Embedded Database

In this exercise, you will convert your Department application to work with an embedded database.

Time: 20 minutes
Format: Programming exercise

1. Continue working with the `MyBatisWithSpringDML` project from the previous exercise.

2. Modify the beans.xml file to invoke scripts to initialize the database and populate the data.
    a. You can generate starting scripts by using the export facility in SQL Developer.
    b. Or use the files provided in the SQL folder of the project.

3. Set up an alias for the datasource so it will work with the Oracle database or your new embedded database. Switch over to the embedded database.

4. Confirm that it is working correctly by running the tests.

Optional Exercise 5.3: Calling a Stored Procedure with MyBatis

In this exercise, you change your project to work with a stored procedure.

Time: 20 minutes
Format: Programming exercise

1. Continue working with the `MyBatisWithSpringDML` project from the previous exercise.

2. Create a new JUnit test for a method that calls the stored procedure in the proc.sql file (there is also a version for hsqldb).
    a. If you plan to use the hsqldb version, you will need to change the command separator so that Spring does not interpret the semicolons in the stored procedure as the end of executable statements. Use the following:

```
<jdbc:script location="classpath:departments-hsqldb-procedure.sql"
     separator="#" />
```

3. Now implement the method.

Exercise 5.4: Caching with MyBatis

In this exercise, you will explore how you can control how MyBatis performs caching.

Time: 30 minutes
Format: Programming exercise

1. Open the `MyBatisCaching` project.

2. Open the `ExhibitDAOTest.java` file and inspect the `testGetAllExhibits` test.

3. Run the test as a JUnit test.

4. Examine the output in the Console window.

5. How many times is the `SELECT` statement prepared?
    a. Look in the Console for the MyBatis `DEBUG` message that says it is preparing the SQL statement.

6. Open the `ExhibitMapper.xml` file.

7. Uncomment the `<cache >` statement.

8. Run the `testGetAllExhibits` test again.

9. How many times is the `SELECT` statement prepared now?
    a. Look in the Console for the MyBatis `DEBUG` message that says it is preparing the SQL statement.

10. Modify the `flushinterval` attribute of the cache element to be `flushInterval="2000"`

11. Run the `testGetAllExhibits` test again.

12. How many times is the `SELECT` statement prepared now?
    a. Look in the Console for the MyBatis `DEBUG` message that says it is preparing the SQL statement.

Optional Exercise 5.5: Using Annotations with MyBatis

In this exercise, you will configure MyBatis mapping information by using annotations instead of an XML file.

Time: 20 minutes
Format: Programming exercise

1. Open the `MyBatisWithSpringDML` that you worked on in previous exercises.

2. Convert one of the queries from XML to annotations.

3. Re-run your JUnit tests and confirm that it still works.

4. If you have time, convert more of the queries.

This page intentionally left blank.

Chapter 6: Functional Programming

Exercise 6.1: Lambda Expressions

Time: 20 minutes
Format: Individual programming exercise

1. Open the `FunctionalProgramming` project and examine the classes in the `com.fidelity.lambda` package.
    a. Complete the tasks described in the TODO comments.
    b. Make sure that the unit tests pass.

Exercise 6.2: Working with Streams

Time: 20 minutes
Format: Individual programming exercise

1. Open the `FunctionalProgramming` project and examine the classes in the `com.fidelity.streams` package.
    a. Complete the steps described in the TODO comments.
    b. Make sure that the unit tests pass.