Spring Framework Essentials

48 of 48 lessons completed

100%

Spring Overview Lab

Purpose

Welcome to *Core Spring*!

In this lab you will come to understand the basic workings of the *Reward Network* reference application and you will be introduced to the tools you will us throughout the course.

Learning Outcomes

What you will learn:

1. Project structure
2. Core *RewardNetwork* Domain and API
3. Basic interaction of the key components within the domain.

Specific subjects you will gain experience with:

- You will see that the application logic will be clean and not coupled with infrastructure APIs.
- You will understand that you can develop and unit test your business logic without using Spring.
- In later labs, you will see that the code you develop in this lab will be directly runnable in a Spring environment without change.

You will be using the *10-spring-intro* project.

Estimated time to complete: 30 minutes.

Prerequisites

The prerequisites are included as part of the *Introduction to the Spring Professional Learning Path* course *Lab Setup* lesson.

If you already completed it, you should be ready to do this lab. If not, assuming you already have JDK 11 or 17 installed and Java IDE, you will need to d following:

1. Download the **lab codebase zip file**.
2. Once you have downloaded the file, unzip it under a directory of your choice. The unzipped directory `core-spring-labfiles/lab` contains the lab projects.
3. Run `./mvnw clean install` (if you plan on using Maven as your build tool).

Use Case

Once you will have familarized yourself with the tools and the application domain, you will implement and test the *Rewards Network* application using Pla Java Objects (POJOs).

Have fun with the steps below, and remember the goal is to get comfortable with the tools and application concepts.

Instructions

This lab differs from the remainder of the course in that most of the steps are review tasks. It is imperative you follow *all* the review tasks to gain an understanding of the problem domain, and this existing codebase. It is the basis of the rest of the course.

Review Rewards Application

TODO-01: Review the *Introduction to the Spring Professional Learning Path* course *Rewards Reference Domain* lesson to gain background on the Rewa application.

Review Dependencies

TODO-02: Review project dependencies

- Open a terminal window, and type the following command under the `lab` directory - one for Maven and the other for Gradle.

```
<core-spring-labfiles/lab> ./mvnw -pl 10-spring-intro dependency:tree
```

```
<core-spring-labfiles/lab> ./gradlew 10-spring-intro:dependencies
```

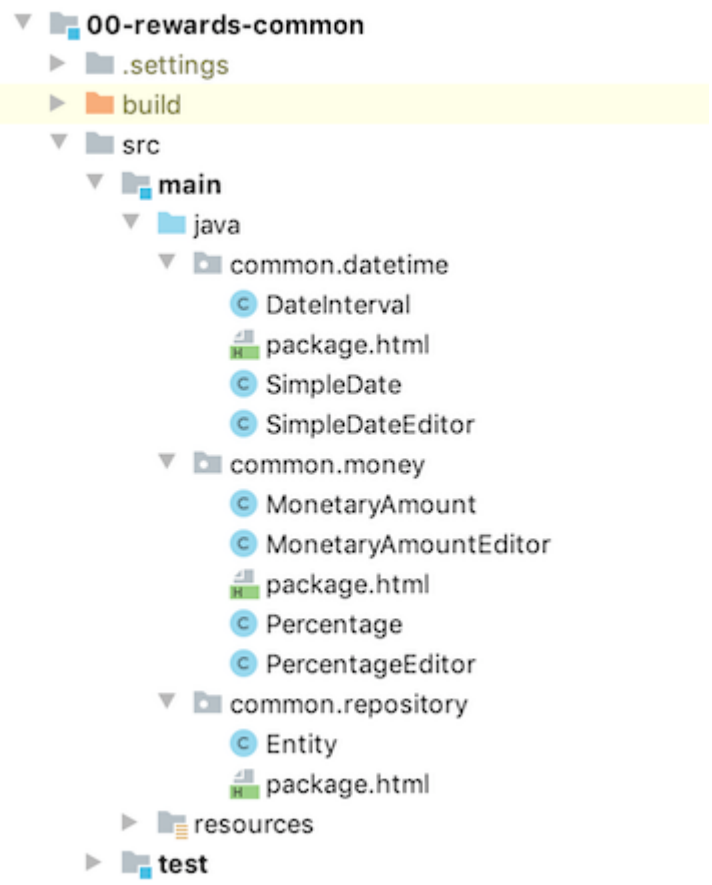
- Review the dependencies (including transitive dependencies)

  For the most part, these dependencies are straightforward and probably similar to what you're used to in your own projects. For example, there are dependencies on Spring Framework jars, on Hibernate, DOM4J, etc.

TODO-03: Review Rewards Common project

In addition to having dependencies on a number of libraries, all lab projects also have a dependency on a common project called *00-rewards-common*.

- Open this project now.



  This project is specific to Spring Training courseware, and contains a number of useful types such as *MonetaryAmount*, *SimpleDate* and *Percentage* will make use of these types throughout the course.

- Take a moment now to explore the contents of the project.

Review Reward Application Domain and API

Before we use Spring to configure an application, we need to understand the application itself.

This section will guide you through the background of the Reward Network application domain and provide context for the rest of the course.

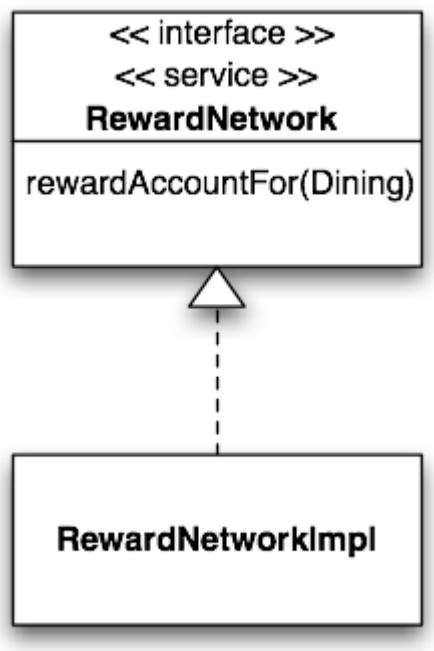
The rewards application consists of several pieces that work together to reward accounts for dining at restaurants.

In this lab, most of these pieces have been implemented for you. However, the central piece, the `RewardNetwork`, has not.

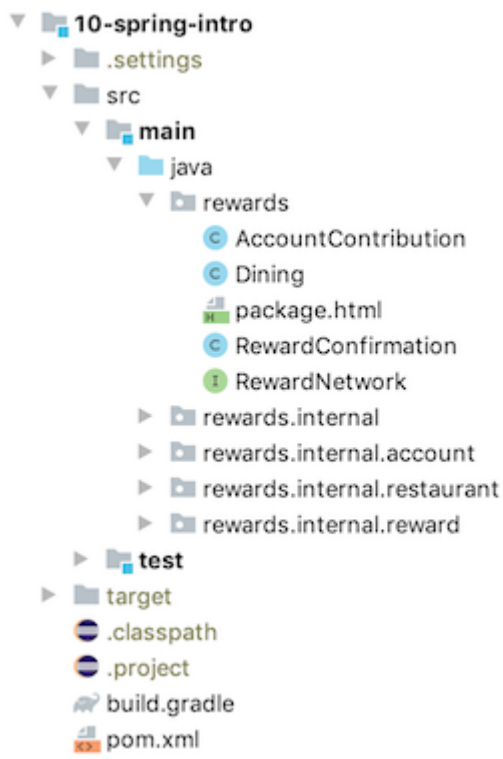
TODO-04: Review `RewardNetwork` interface and `RewardNetworkImpl` class

The `RewardNetwork` is responsible for carrying out `rewardAccountFor(Dining)` operations. It is an interface.

It is implemented by the `RewardNetworkImpl` class:



- In the `10-spring-intro` project, navigate into the `src/main/java` source folder and you will see the root `rewards` package. Within that package y find the `RewardNetwork` Java interface definition:



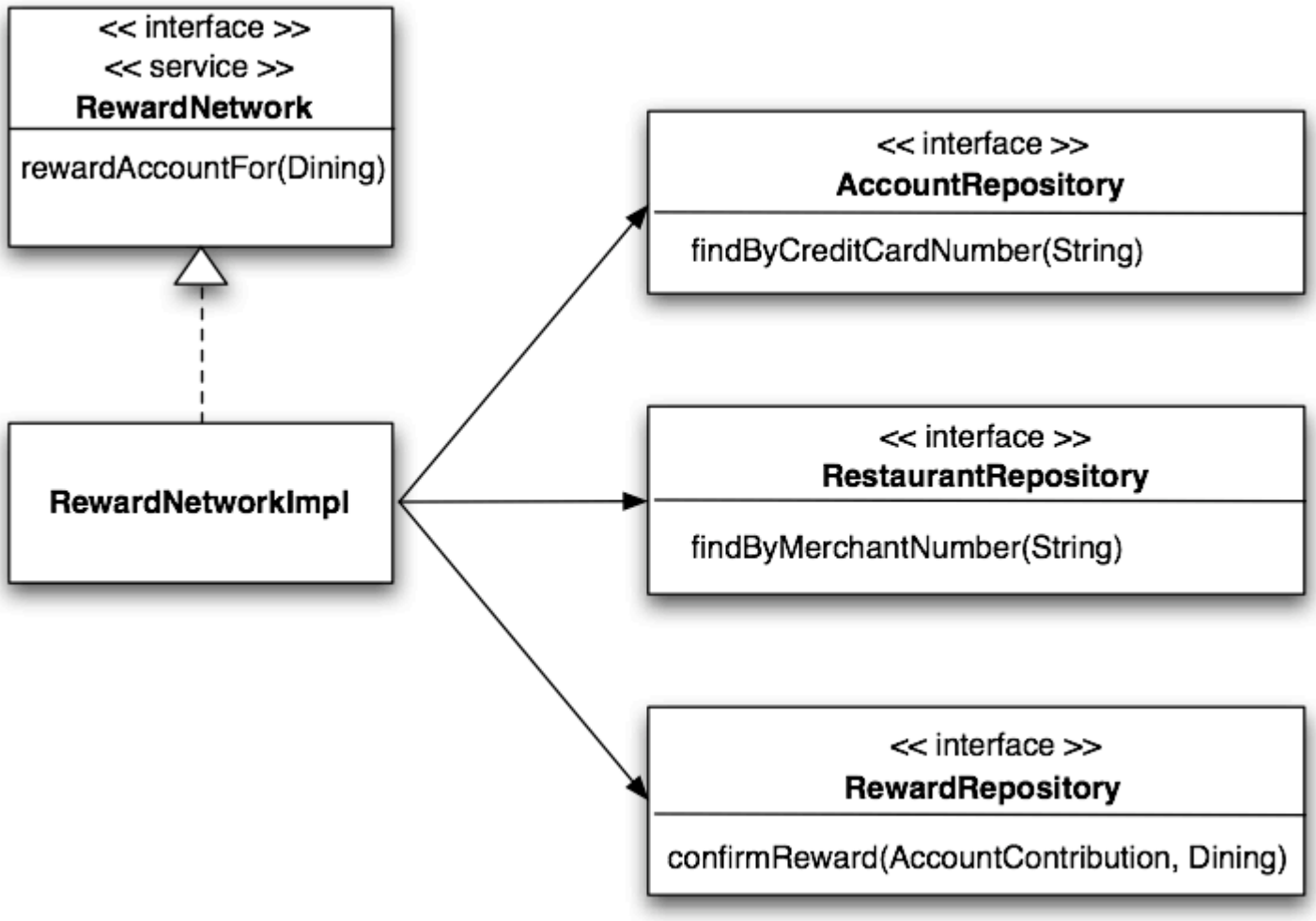
  The classes inside the root `rewards` package fully define the public interface for the application, with `RewardNetwork` being the central element.
- Open the `RewardNetwork` interface and review it.
- Expand the `rewards.internal` package and open the implementation class `RewardNetworkImpl`.

TODO-05: Review the `RewardNetworkImpl` configuration logic

`RewardNetworkImpl` should rely on three supporting data access services called 'Repositories' to do its job. These include:

1. An AccountRepository to load `Account` objects to make benefit contributions to.
2. A RestaurantRepository to load `Restaurant` objects to calculate how much benefit to reward an account for dining.
3. A RewardRepository to track confirmed reward transactions for accounting and reporting purposes.

This relationship is shown graphically as follows:

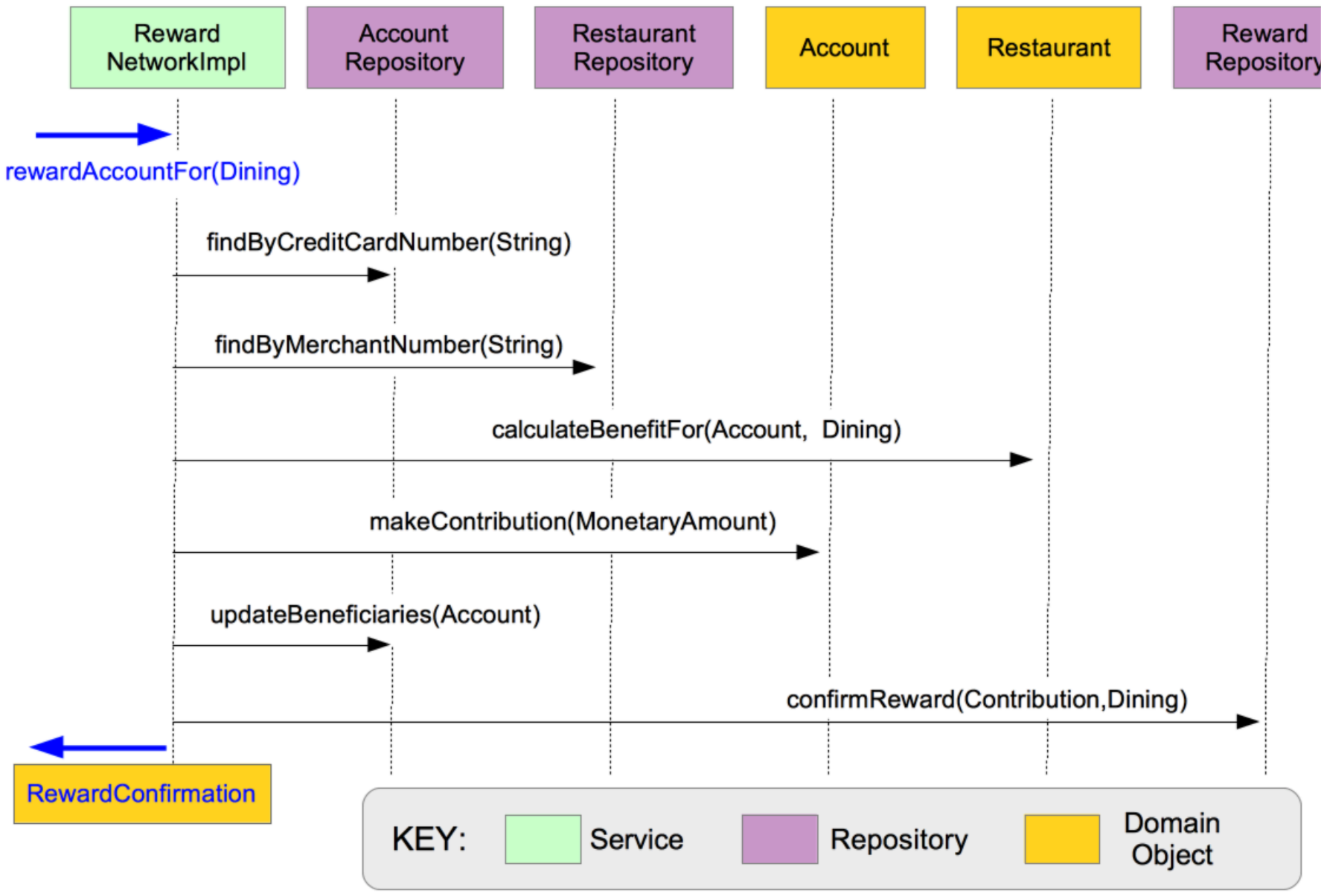


- Locate the single constructor and notice all three dependencies are injected when the `RewardNetworkImpl` is constructed.

Implement the `RewardNetworkImpl` Application Logic

TODO-06: Review sequence diagram

- Review the given `RewardNetworkImpl.rewardAccountFor(Dining)` implementation. It does not do anything at the moment.
- Review the following sequence diagram for implementing `RewardNetworkImpl.rewardAccountFor(Dining)`:



You should *not* need to use operator `new` in your code. Everything you need is returned by the methods you use. The interaction diagram does not s what each call returns, but most of them return something.

TODO-07: Write code for rewarding an account according to the sequence diagram above

- Implement the application logic necessary to complete the `rewardAccountFor(Dining)` operation, delegating to your dependencies (`accountRepository`, `restaurantRepository`, `rewardRepository`) as you go.
- Get the credit card and merchant number from the `Dining` object.

TODO-08: Return the corresponding reward confirmation

- Set the return value of the `rewardAccountFor(Dining)` operation to the `RewardConfirmation` returned by `rewardRepository.confirmReward()` operation.

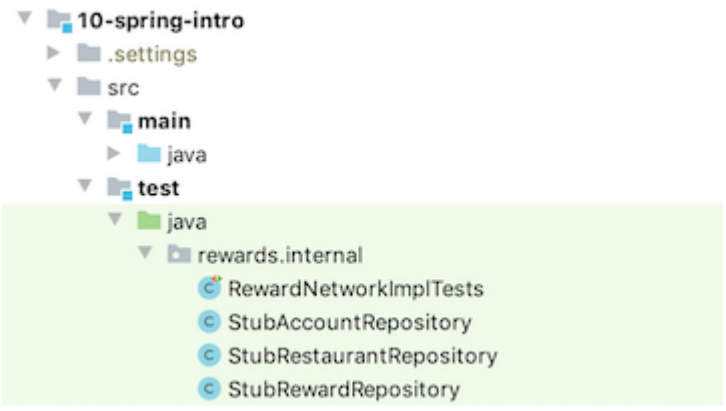
Unit Test the `RewardNetworkImpl` Application Logic

How do you know the application logic you just wrote actually works? You do not, not without a test that proves it. In this step you will review and run an automated JUnit test to verify what you just coded is correct.

TODO-09: Review the test setup

- Navigate into the `src/test/java` source folder and you will see the root `rewards` package.

  All tests for the rewards application reside within this tree at the same level as the source they exercise.
- Drill down into the `rewards.internal` package and you will see `RewardNetworkImplTests`, the JUnit test for your `RewardNetworkImpl` class.



- Review the `setUp()` method, the 'stub' repositories have been created and injected into the `RewardNetworkImpl` class using the constructor.

TODO-10: Run the `testRewardForDining()` test

- Review `testRewardForDining()` method. It is the only test method in the `RewardNetworkImplTests`. It is initially disabled.

  It calls `rewardNetwork.rewardAccountFor(Dining)` and then makes assert statements to evaluate the result of calling this method. In this way th test is able to construct an instance of `RewardNetworkImpl` using the stub objects as dependencies and verify that the logic you implemented functi expected.
- Enable `testRewardForDining()` by removing the `@Disabled` annotation.
- Run the `RewardNetworkImplTests` test within your IDE.
- You should have successful test. If you do not, review your `RewardNetworkImpl.rewardAccountFor(Dining)` implementation.

Summary

You have just developed and unit tested a component of a realistic Java application, exercising application behavior successfully in a test environment. Y used stubs to test your application logic in isolation, without involving external dependencies such as a database or Spring. And your application logic is c and decoupled from infrastructure APIs.

In the next lab, you will use Spring to configure this same application from all the *real* parts, including plugging in *real* repository implementations that acc relational database.

Summary **Instructors**

In this lab, you used Spring to configure the Reward Application, and test it.

Give Feedback

Help us improve by sharing your thoughts.

Give Feedback