Spring Framework

* What is spring
* When we should use it
* Why is it so popular
* What type of applications?

Open Source, Lightweight, DI Container, Framework 🡪 building java enterprise applications

<https://github.com/spring-projects/spring-framework>

<http://mvnrepository.com/artifact/org.springframework>

<http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlsingle>

maven use is recommended

lightweight – don’t need a runtime - don’t need an application server

not invasive – do not require you to extend framework classes or implement framework interfaces

low overhead – spring jars are relatively small

DI Container

* Servers as a lifecycle manager and instantiates and injects dependencies into your objects
* DI Container is sometimes called IOC Container
* Enterprise applications must deal with a wide variety of tech

Containerization

Cloud

JDBC, ORM

Streaming, events , JMS

Security

Monitoring

Provides classes to simplify working with lower-level techs

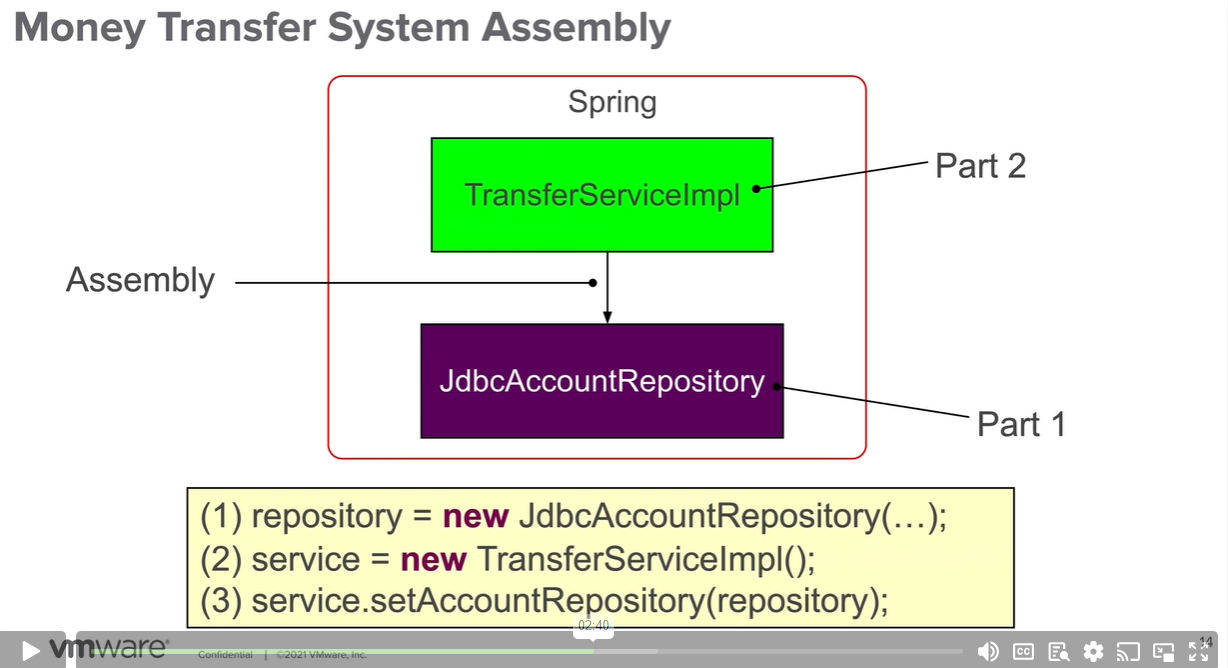
Extensible and customizable.

DI Container – heart of spring

* Don’t repeat yourself
* Separation of concerns
* Convention over configuration
* Testability

A typical application consists of several parts working together to carry out a use case

Eg. Service and Repository.



Above is the equivalent code

A screenshot of a computer

Description automatically generated

Notice that service depends on an interface and not an implementation.

For testing, when there is no container, you can create your own stub – so this simplifies testing a lot

Spring framework history

* 2003 – first release
* J2EE difficult to use and test
* Spring – dependency injection, jdbc data access, support for multiple envs
* Code quality
* OSS community
* Careful API design
* Adaptable to change – integration with other open source projects like hibernate
* Spring projects – sprint secutityy
* Spring boot
* 2003 – 0.9 version
* Spring - integration with web apps, messaging, persistence, batch, streaming
* Spring adaptability - jdk, nativa compilation, reactive , Kotlin, Kubernetes
* First lab.

Lab

A screenshot of a computer

Description automatically generated

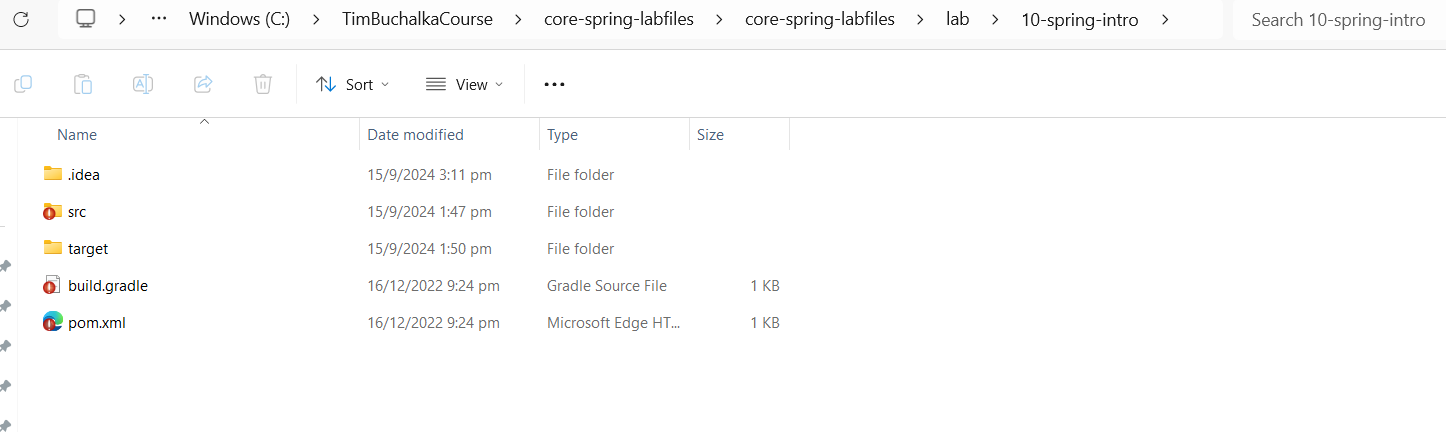
We need to run mvn clean install – so that it will be available as a library to be used.



A screenshot of a computer program

Description automatically generated

We open the following project



This project is dependent upon the following

00-rewards-common

A screenshot of a computer program

Description automatically generated

Where is the 00-rewards-common jar coming from ?

Lets look at the m2 repository

A screenshot of a computer

Description automatically generated

Its coming from there!

So we have to mvn clean build this project first

Lets do a mvn clean and try

Deleted the library

A screenshot of a computer

Description automatically generated

Now the project got errors

A screenshot of a computer program

Description automatically generated

Lets jus mvn cleanA screenshot of a computer

Description automatically generated build the commons module

Npw check the m2 again

A screenshot of a computer

Description automatically generated

Errors in proj should have disappeared

Now we will test this method

A screenshot of a computer program

Description automatically generated

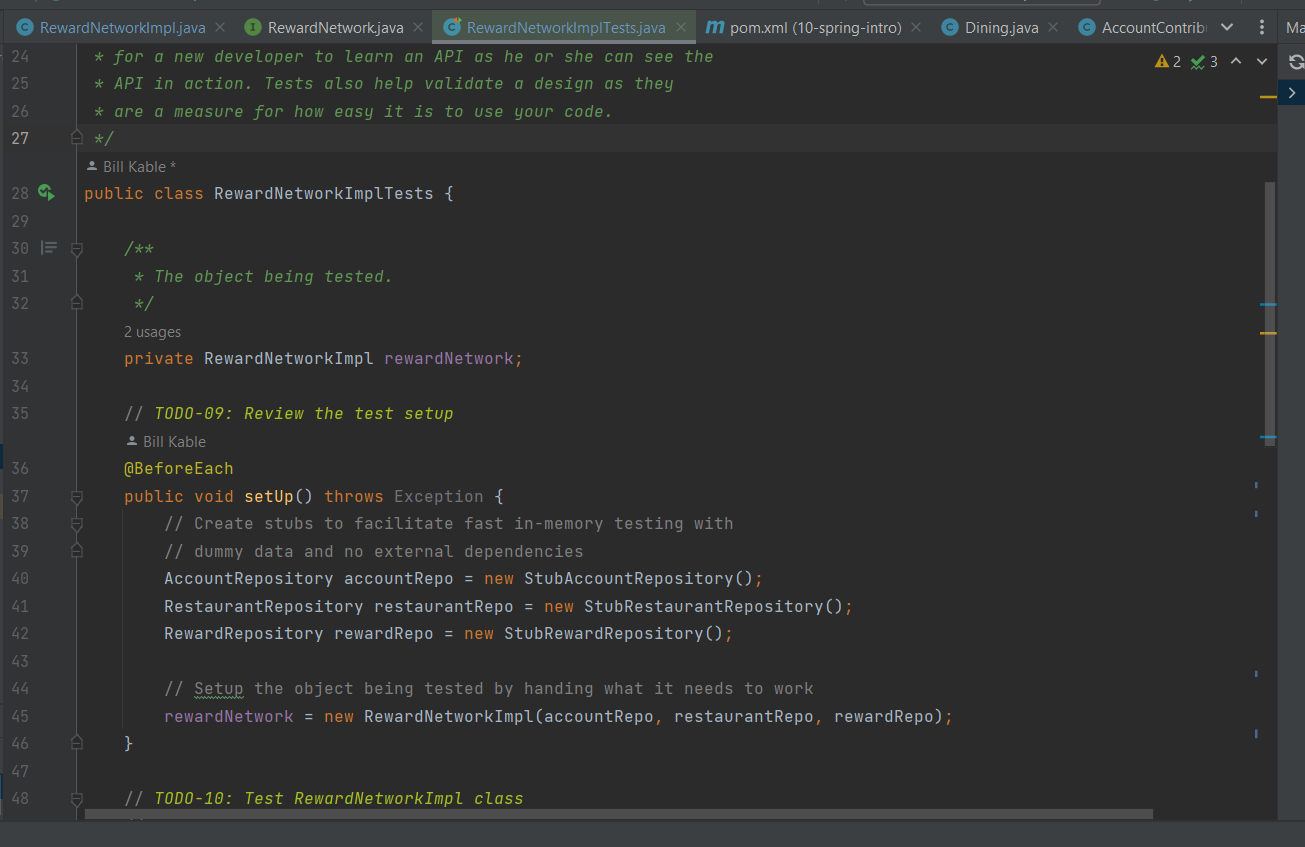
This method belongs to the class

RewardNetworkImpl

To test this method, we need a @Test Annotation

As per the below code

package rewards.internal;  
  
import common.money.MonetaryAmount;  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Disabled;  
import org.junit.jupiter.api.Test;  
import rewards.AccountContribution;  
import rewards.Dining;  
import rewards.RewardConfirmation;  
import rewards.internal.account.AccountRepository;  
import rewards.internal.restaurant.RestaurantRepository;  
import rewards.internal.reward.RewardRepository;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
import static org.junit.jupiter.api.Assertions.*assertNotNull*;  
  
*/\*\*  
 \* Unit tests for the RewardNetworkImpl application logic.  
 \* Configures the implementation with stub repositories  
 \* containing dummy data for fast in-memory testing without  
 \* the overhead of an external data source.  
 \*   
 \* Besides helping catch bugs early, tests are a great way  
 \* for a new developer to learn an API as he or she can see the  
 \* API in action. Tests also help validate a design as they  
 \* are a measure for how easy it is to use your code.  
 \*/*public class RewardNetworkImplTests {  
  
 */\*\*  
 \* The object being tested.  
 \*/* private RewardNetworkImpl rewardNetwork;  
  
 // *TODO-09: Review the test setup* @BeforeEach  
 public void setUp() throws Exception {  
 // Create stubs to facilitate fast in-memory testing with  
 // dummy data and no external dependencies  
 AccountRepository accountRepo = new StubAccountRepository();  
 RestaurantRepository restaurantRepo = new StubRestaurantRepository();  
 RewardRepository rewardRepo = new StubRewardRepository();  
  
 // Setup the object being tested by handing what it needs to work  
 rewardNetwork = new RewardNetworkImpl(accountRepo, restaurantRepo, rewardRepo);  
 }  
  
 // *TODO-10: Test RewardNetworkImpl class* // - Remove the @Disabled annotation below.  
 // - Run this JUnit test. Verify it passes.  
 @Test  
 //@Disabled  
 public void testRewardForDining() {  
 // create a new dining of 100.00 charged to credit card '1234123412341234' by merchant '123457890' as test input  
 Dining dining = Dining.*createDining*("1500.00", "1234123412341234", "1234567890");  
  
 // call the 'rewardNetwork' to test its rewardAccountFor(Dining) method  
 RewardConfirmation confirmation = rewardNetwork.rewardAccountFor(dining);  
  
 // assert the expected reward confirmation results  
 *assertNotNull*(confirmation);  
 *assertNotNull*(confirmation.getConfirmationNumber());  
  
 // assert an account contribution was made  
 AccountContribution contribution = confirmation.getAccountContribution();  
 *assertNotNull*(contribution);  
  
 // the account number should be '123456789'  
 *assertEquals*("123456789", contribution.getAccountNumber());  
  
 // the total contribution amount should be 8.00 (8% of 100.00)  
 *assertEquals*(MonetaryAmount.*valueOf*("300.00"), contribution.getAmount());  
  
 // the total contribution amount should have been split into 2 distributions  
 *assertEquals*(2, contribution.getDistributions().size());  
  
 // each distribution should be 4.00 (as both have a 50% allocation)  
 *assertEquals*(MonetaryAmount.*valueOf*("120.00"), contribution.getDistribution("Annabelle").getAmount());  
 *assertEquals*(MonetaryAmount.*valueOf*("180.00"), contribution.getDistribution("Corgan").getAmount());  
 }  
}



Notice that the rewardNetworkImpl object’s dependencies, could easily be stubbed because of the loose coupling!

The setup method has a BeforeEach annotation, which means it will run before the execution of each test

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated