Lab 1

SwitchYard

Lab Steps

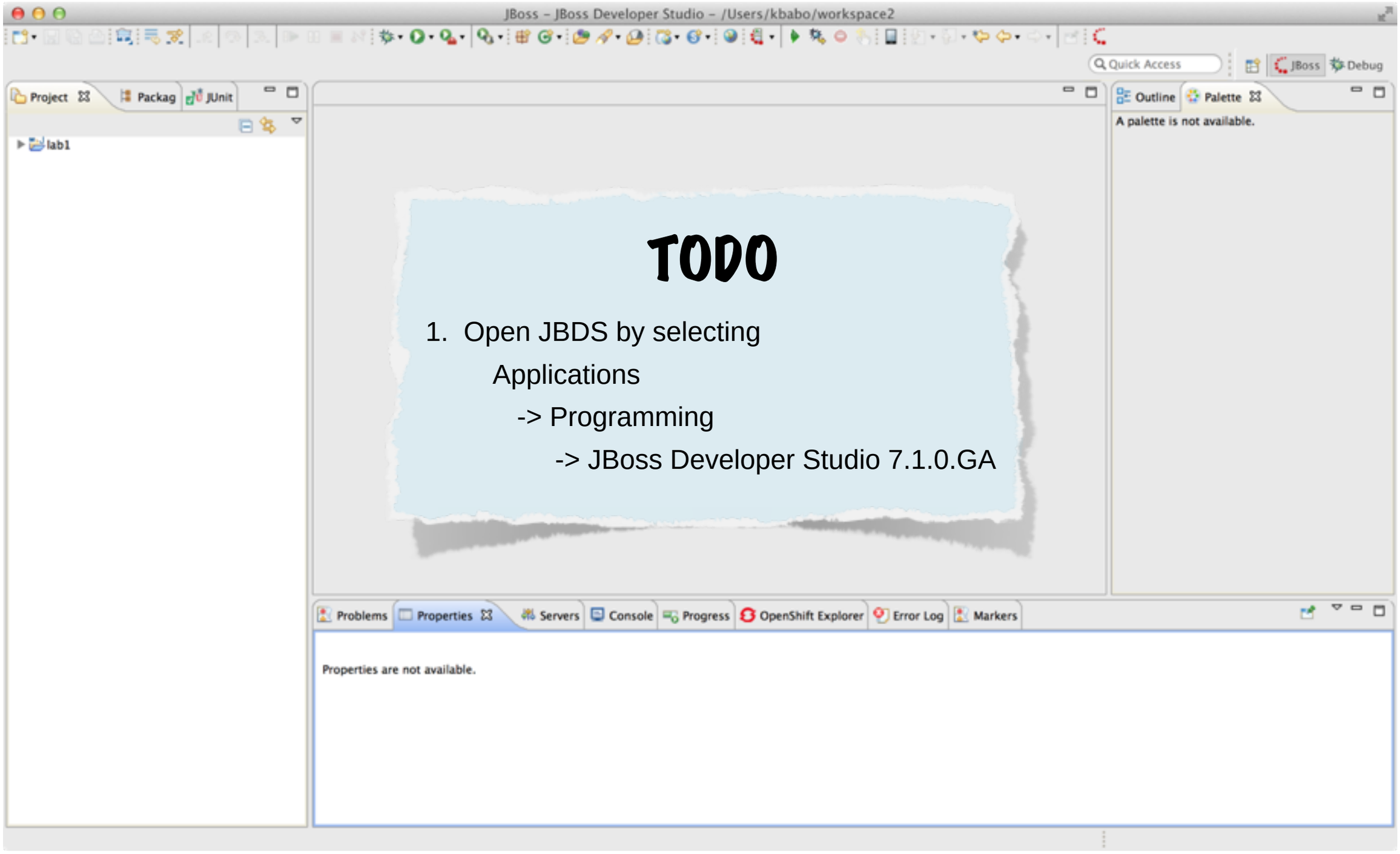
- Step 1 : Getting started
- Step 2 : Customer lookup with SQL binding
- Step 3 : Pre-qualification workflow
- Step 4 : Advanced routing with Camel
- Step 5 : Add a SOAP service binding
- Step 6 : Add a JMS service binding
- Step 7 : Implement RESTful status service

Step 1

Getting Started

Goals

- *Import the lab project*
- *Introduce key editor concepts*
- *Run a unit test to verify environment*


JBoss – JBoss Developer Studio – /Users/kbabo/workspace2
Quick Access
JBoss
Debug
Project
Packag
JUnit
lab1
Outline
Palette
A palette is not available.
TODO
1. Open JBDS by selecting
Applications
-> Programming
-> JBoss Developer Studio 7.1.0.GA
Problems
Properties
Servers
Console
Progress
OpenShift Explorer
Error Log
Markers
Properties are not available.

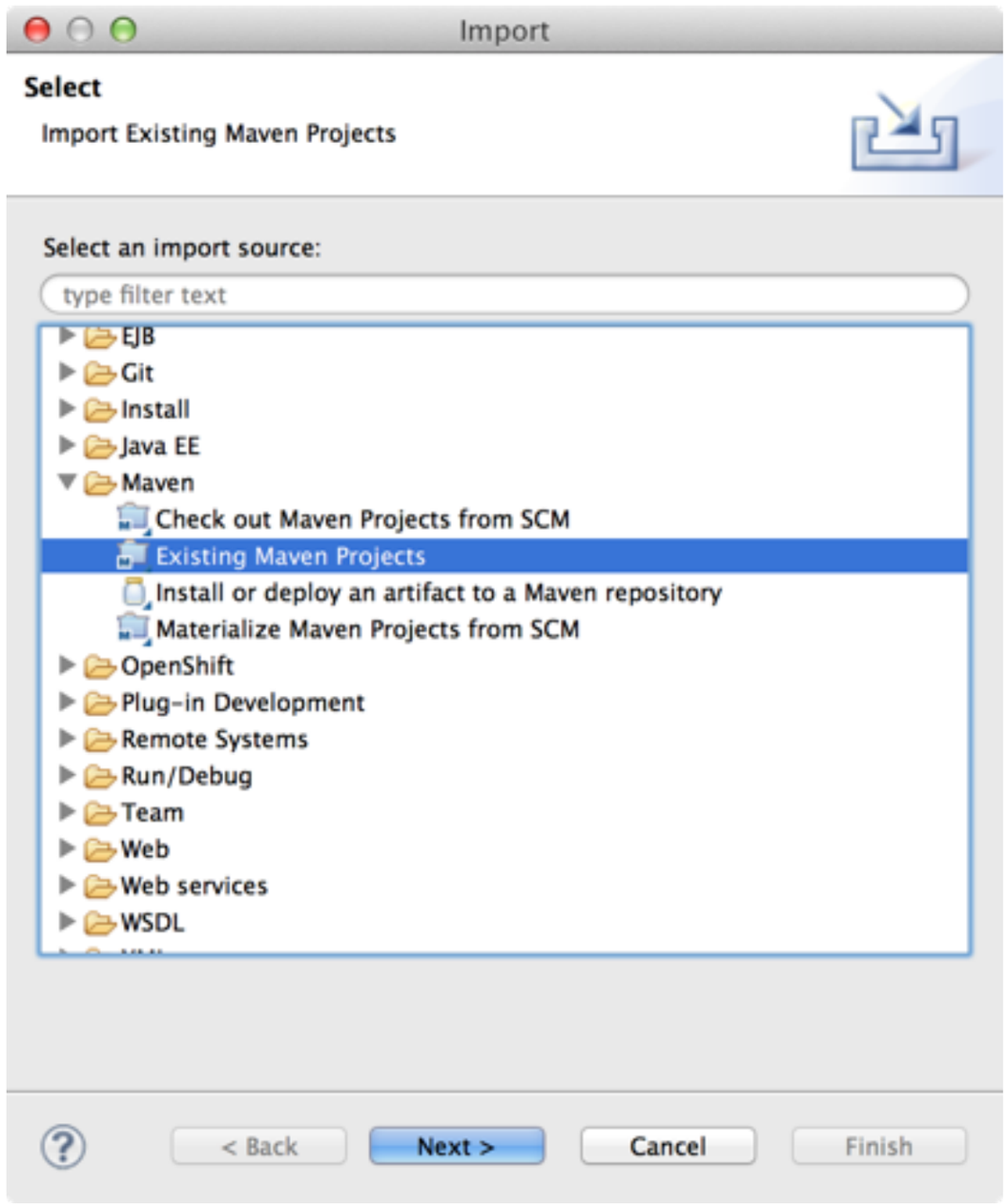
Importing Lab 1

TODO

1. File -> Import ... from the JBDS menu.

2. Select Maven -> Existing Maven Projects

3. Click Next



Importing Lab 1

TODO

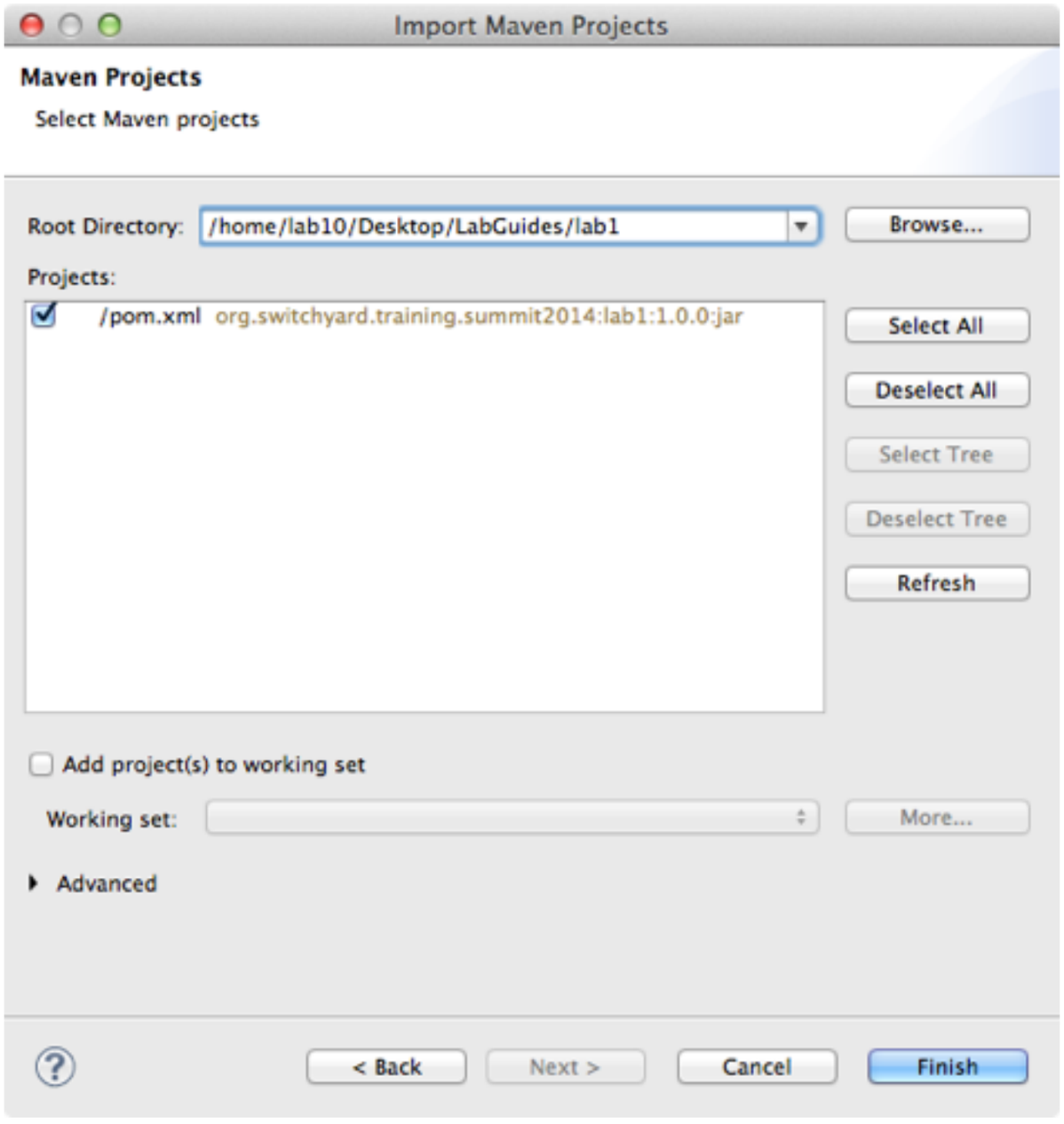
1. Click Browse ... and navigate to the location of lab1. For example:

```
/home/lab10/Desktop/LabGuides/lab1
```

2. Make sure the pom.xml is checked for:

```
org.switchyard.training.summit2014:lab1
```

3. Click Finish



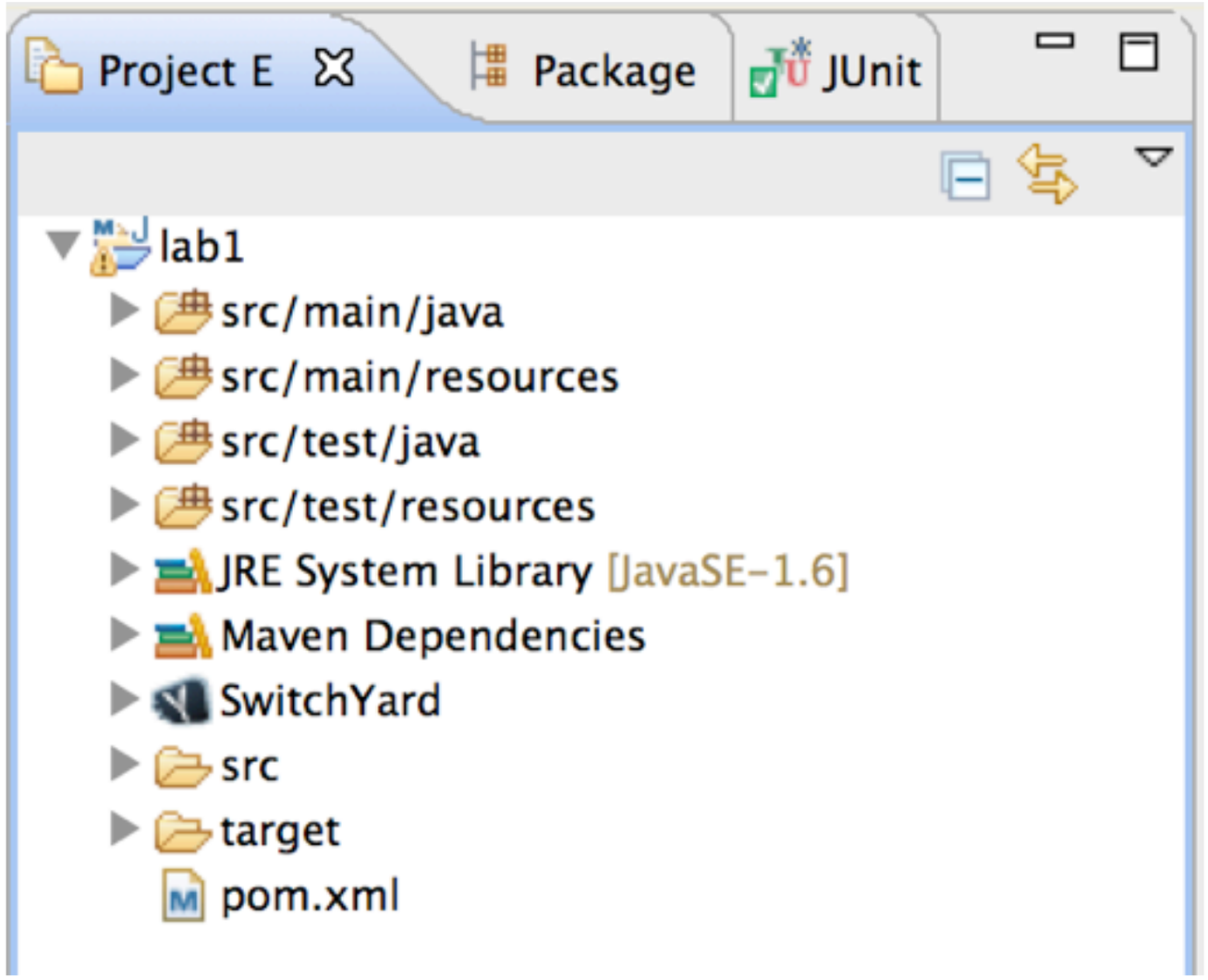
Application Structure

TODO

1. Go to the Project Explorer in the left frame and expand the lab1 project node.

FYI

The next set of slides will examine each section of the project in detail.



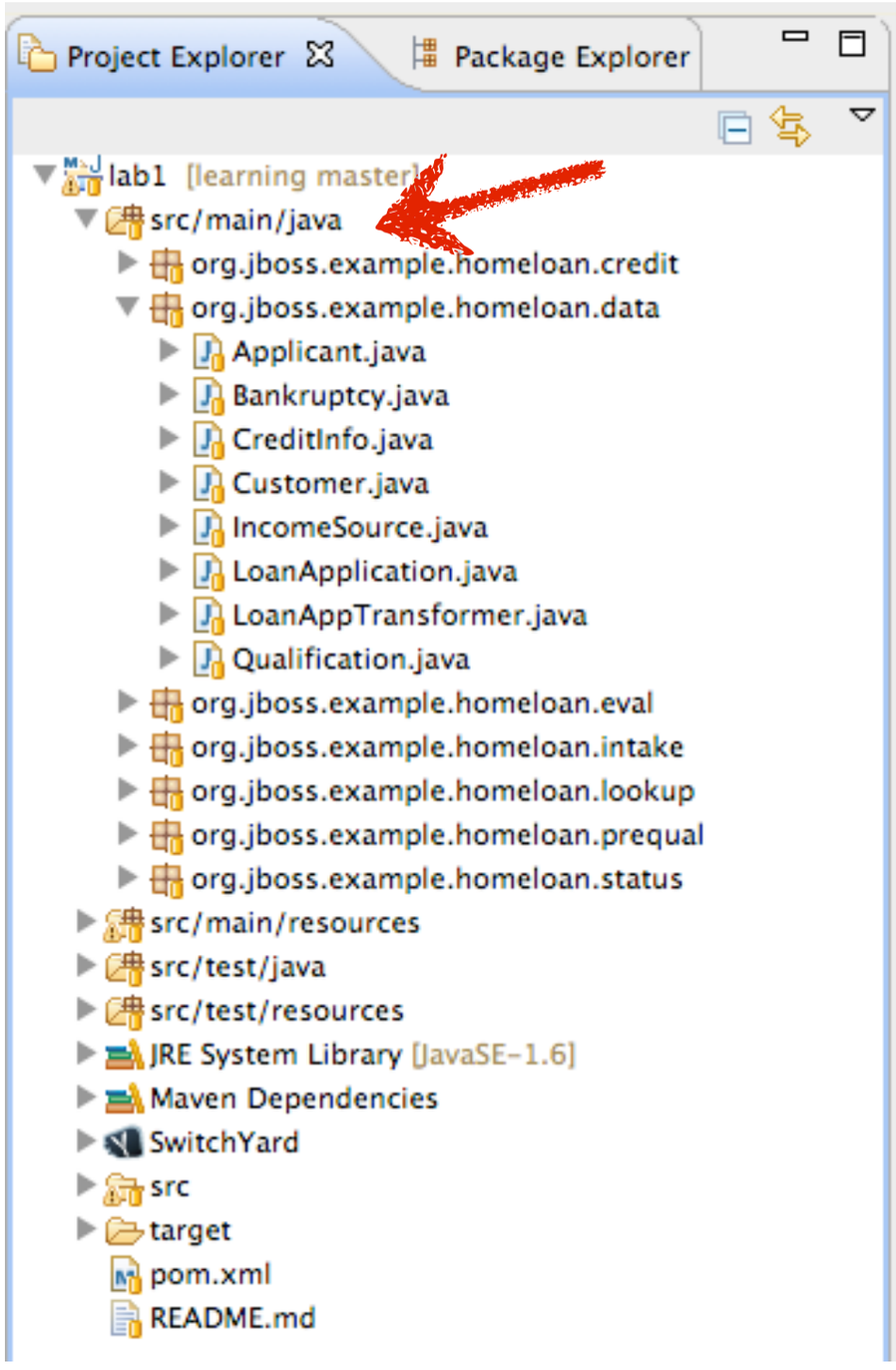
src/main/java

FYI

This is the home of Java files which provide your application's logic. These files are packaged with your application for deployment.

Examples of files you will find here:

- *Java service interfaces*
- *Camel Java DSL routes*
- *CDI Beans*
- *Java Transformers*



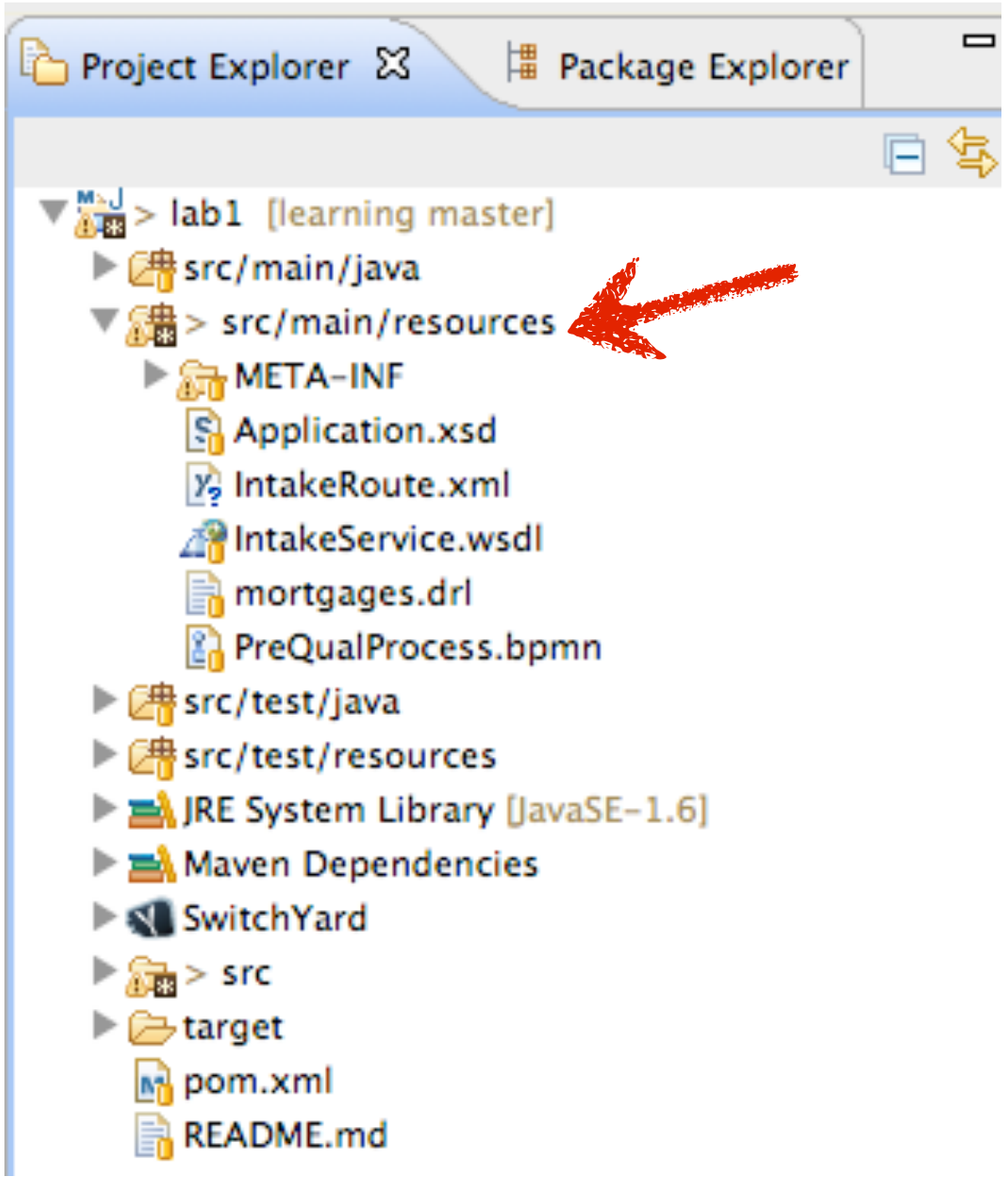
src/main/resources

FYI

Non-Java resources that will be packaged with your application go here.

Examples of files you will find here:

- *SwitchYard application descriptor (switchyard.xml)*
- *Camel XML route definitions*
- *Transformers (XSLT, Smooks)*
- *Drools rule definitions*
- *Process definitions (BPMN 2, BPEL)*
- *WSDL*
- *Schema*



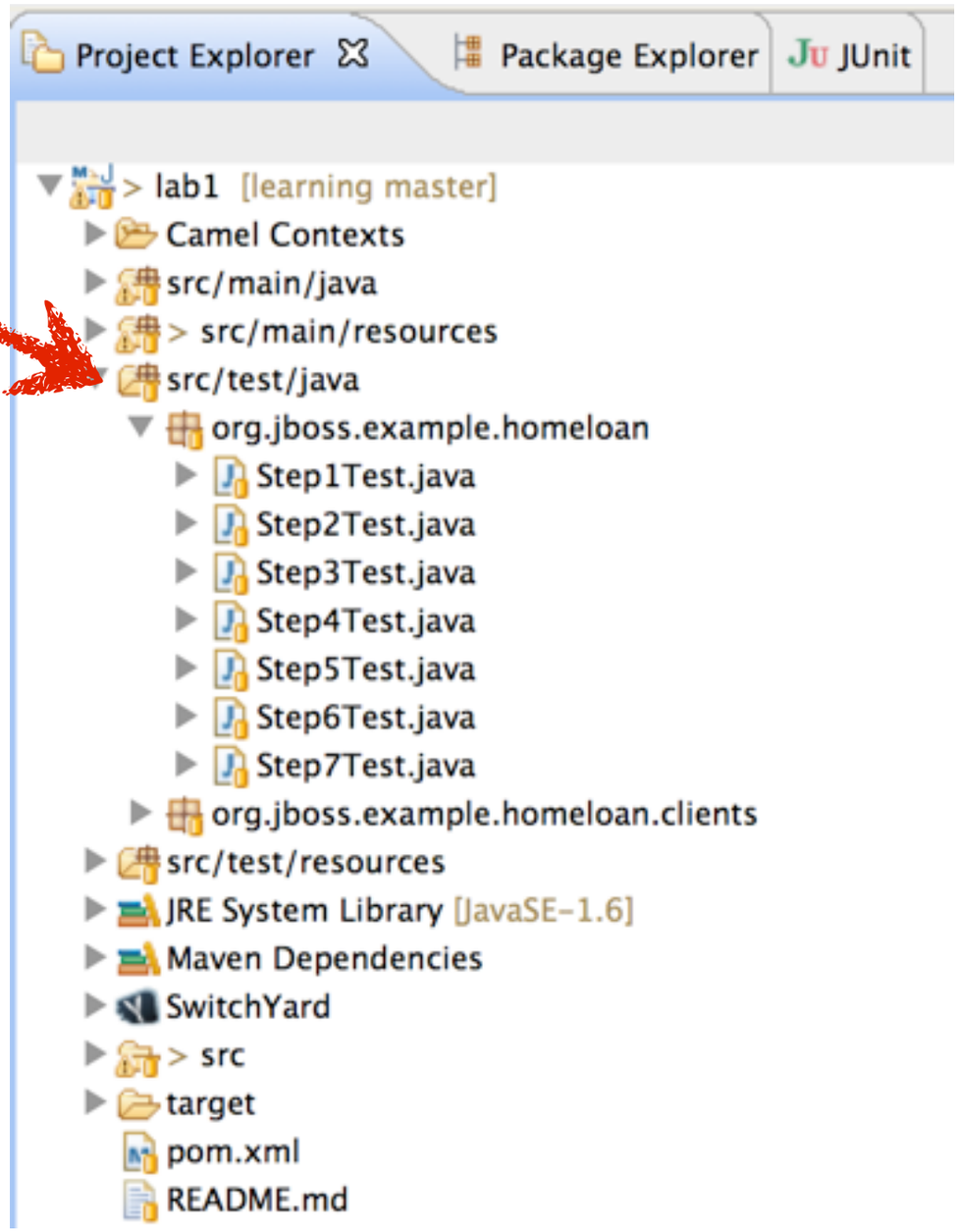
src/test/java

FYI

All Java classes related to testing your application go in src/test/java.

Examples of files you will find here:

- *Service unit tests*
- *Remote test clients*
- *Ancillary test classes (e.g. mock web services)*



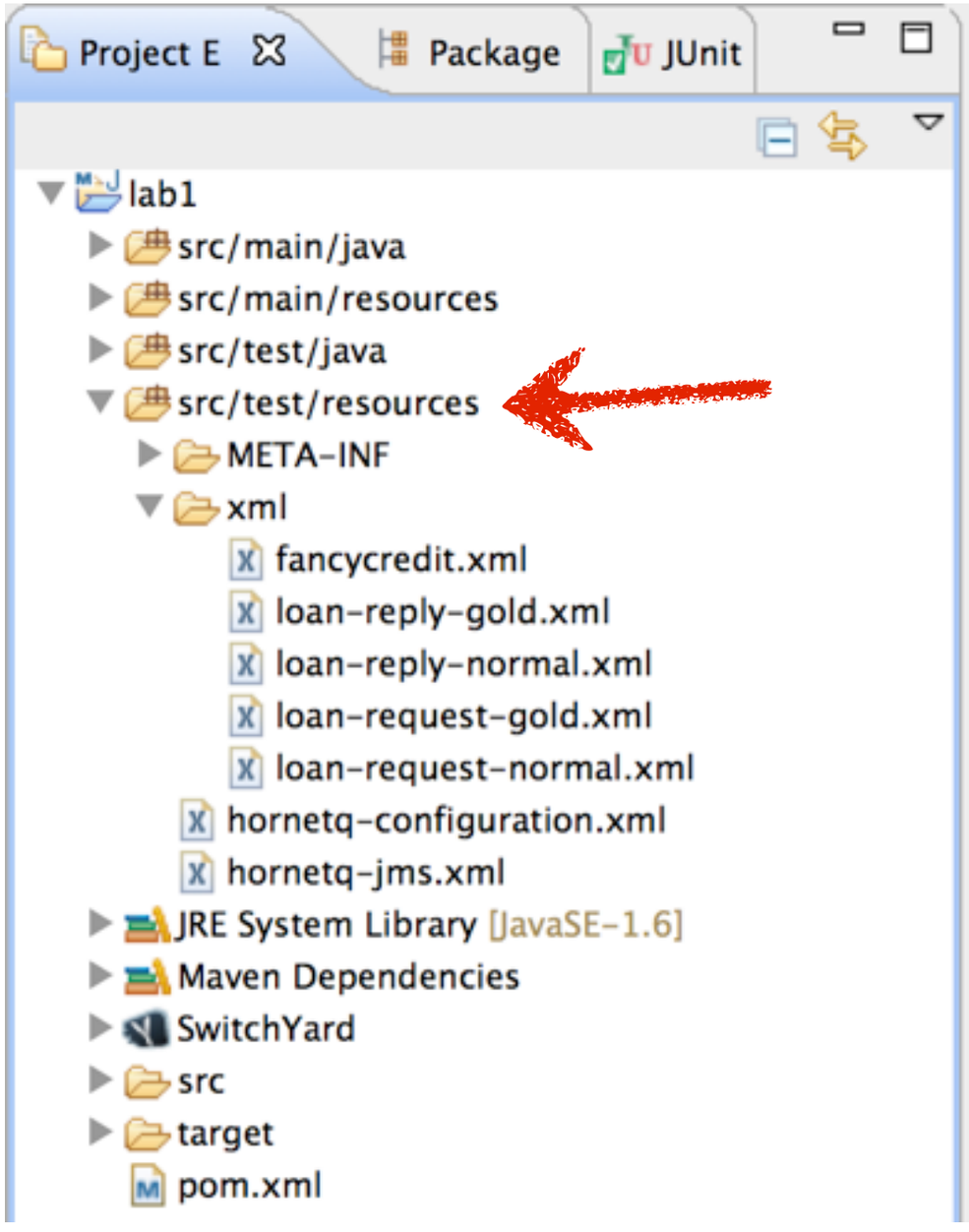
src/test/resources

FYI

Any resources that are used during test execution, but are not included in the application, belong in src/test/resources.

Examples of files you will find here:

- *Test payloads*
- *Expected results for test assertions*
- *JMS destinations used for testing*
- *log settings used for test execution*



SwitchYard!!!!

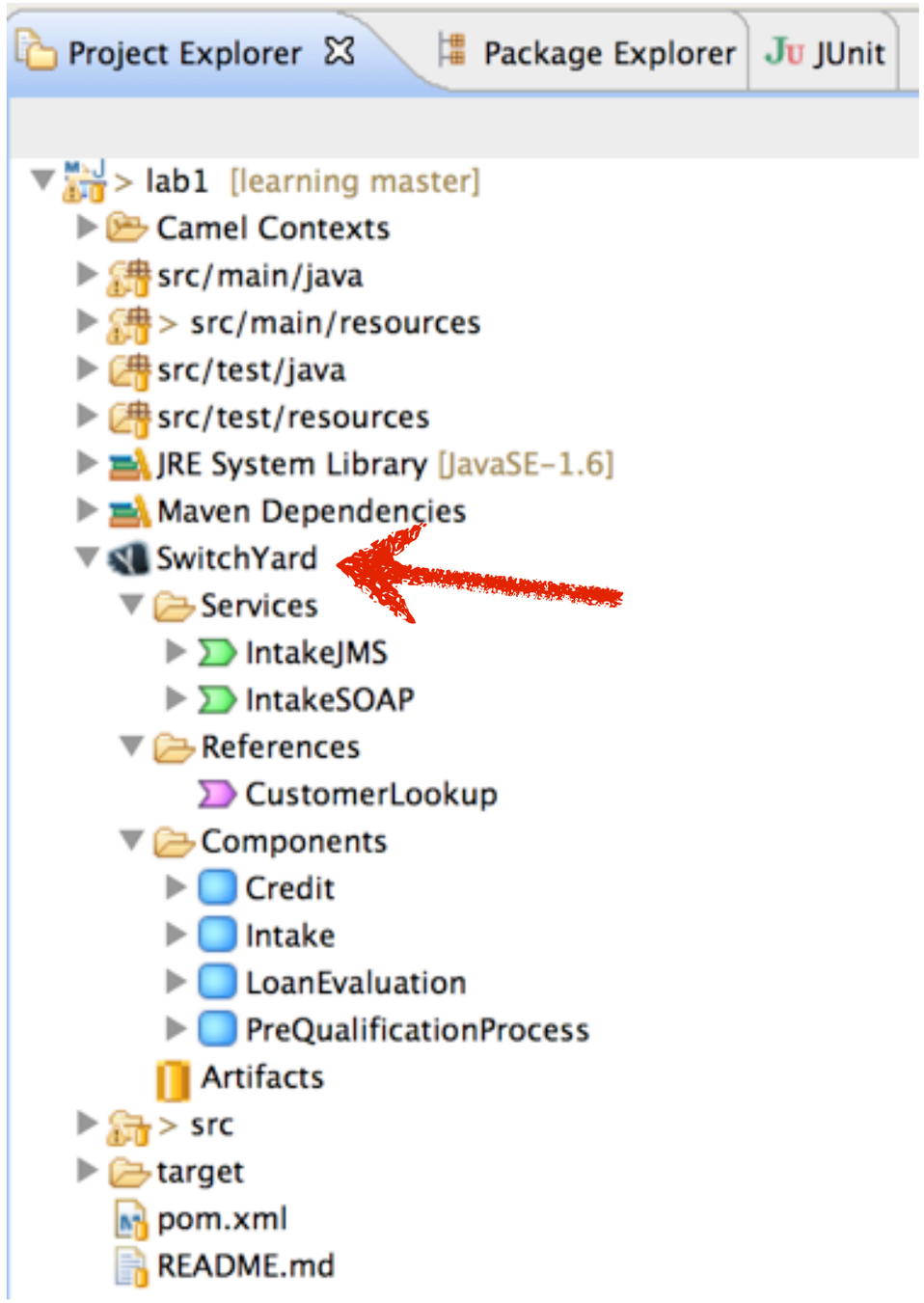
FYI

The SwitchYard node provides two functions in the Project Explorer:

- *An 'outline' style view of the application configuration (services, references, components, etc.)*
- *Shortcut to the SwitchYard visual application editor via a double-click.*

TODO

1. Double-click the SwitchYard node to begin your journey down the rabbit hole!



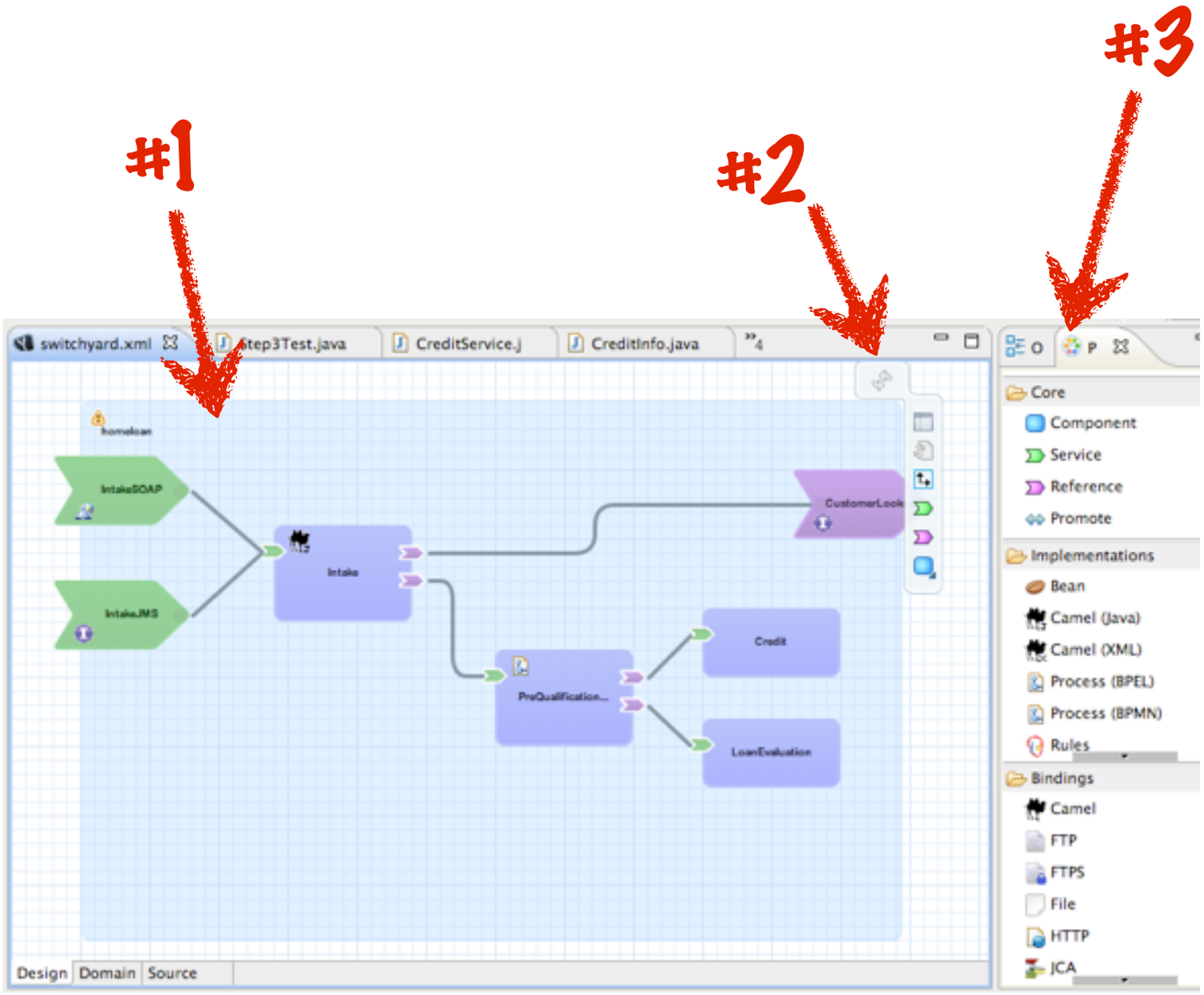
Visual Editor

FYI

*#1 : The **canvas** provides a visual representation of the structure and relationships in a SwitchYard application.*

*#2 : The **button bar** appears when you hover over items on the canvas and provides context-sensitive options for each item.*

*#3 : The **palette** provides a set of application building blocks which can be dragged onto the canvas to create an application.*



Living Large with the Button Bar

FYI

The button bar is really convenient when interacting with the visual editor. Just hover over the portion of the diagram you want to interact with and choose an option from the button bar.

This page provides a key to the more frequently used buttons on the button bar. The properties button is used the most by far, so definitely make note of that one. If the other buttons don't make sense now, don't worry they will soon.

NOTE: the focus for the button bar can be kinda touchy at times when there are multiple areas of focus close together. Do or do not. There is no try.

Opens properties view

Edit the interface for a service or reference.

Promotes a component service.

Generate WSDL from a Java interface.

Adds a reference to a component or composite.

Generate Java interface from a WSDL.

Adds a service to a component or composite.

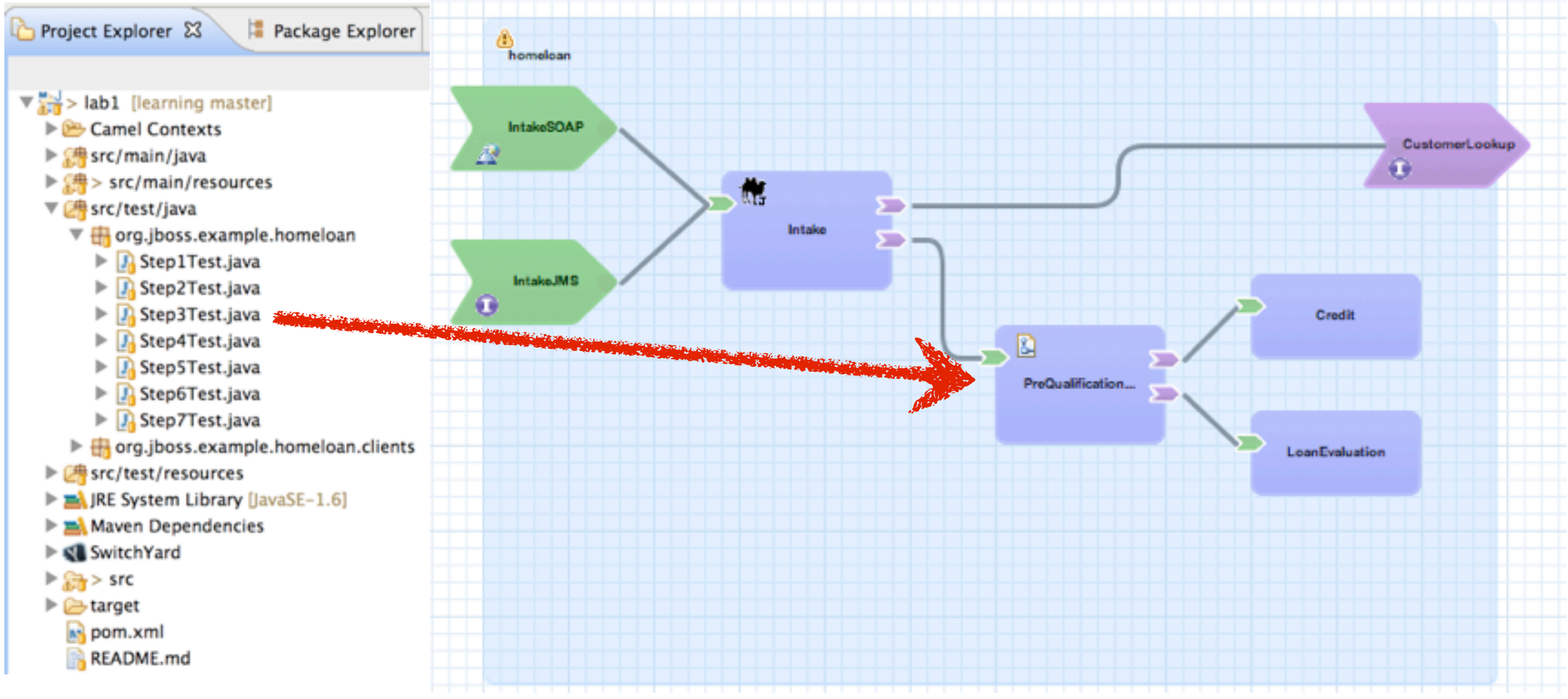
Create a unit test class for a service.

Adds a binding to a composite service or reference.

Unit Testing Services

FYI

SwitchYard allows you to unit test individual services in your application as you build it. Step3Test contains test methods which invoke PreQualificationService directly to verify that it's behavior matches requirements.

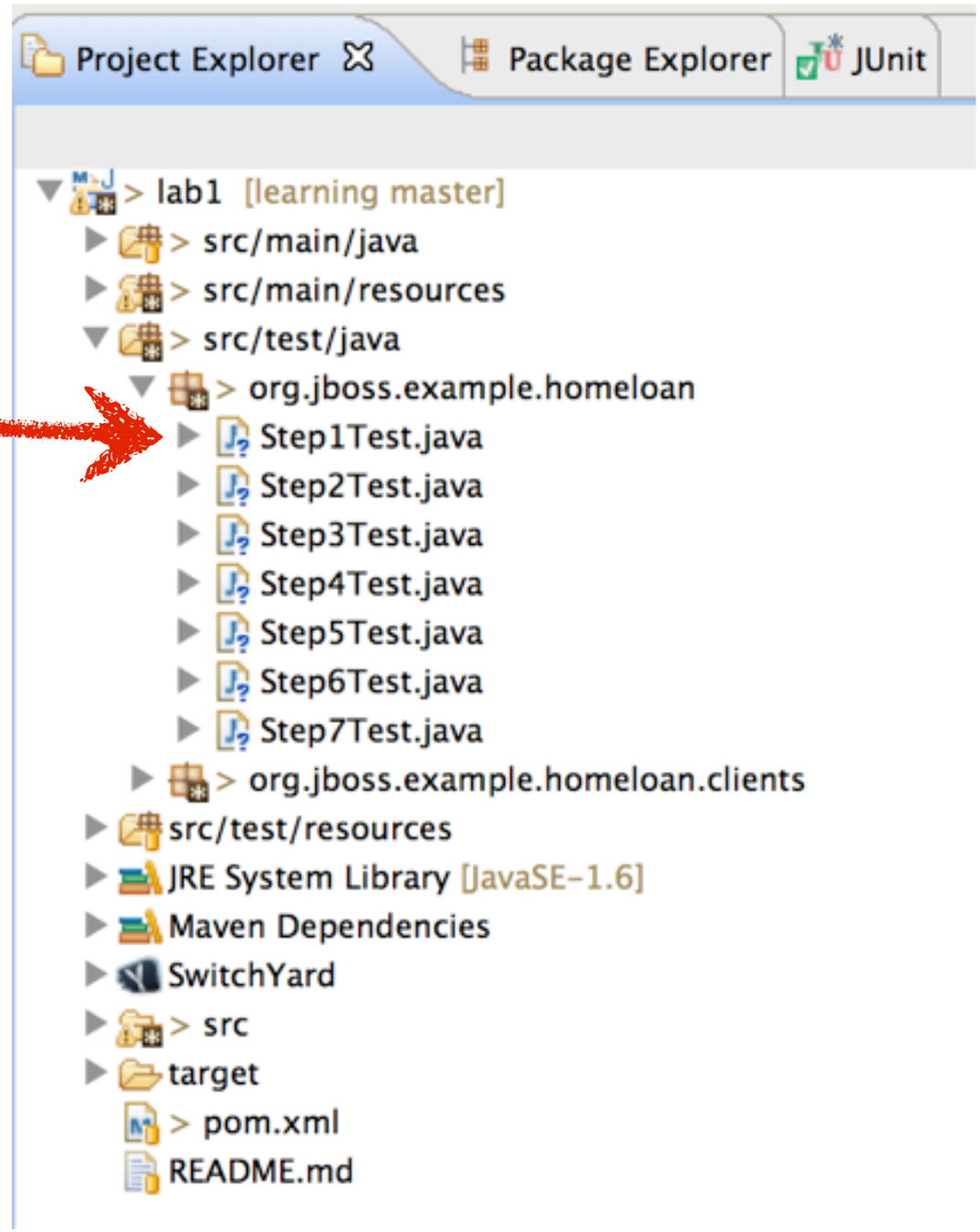


Step 1

Running a Unit Test

TODO

1. Double-click on Step1Test in the explorer to open the unit test.



Step 1

Running a Unit Test

FYI

The @ServiceOperation annotation allows you to inject an invoker for component services directly into unit tests.

TODO

1. Go to the Run menu in the main menu bar and select ‘Run As -> JUnit Test’ to run the unit test.

Step1Test.java

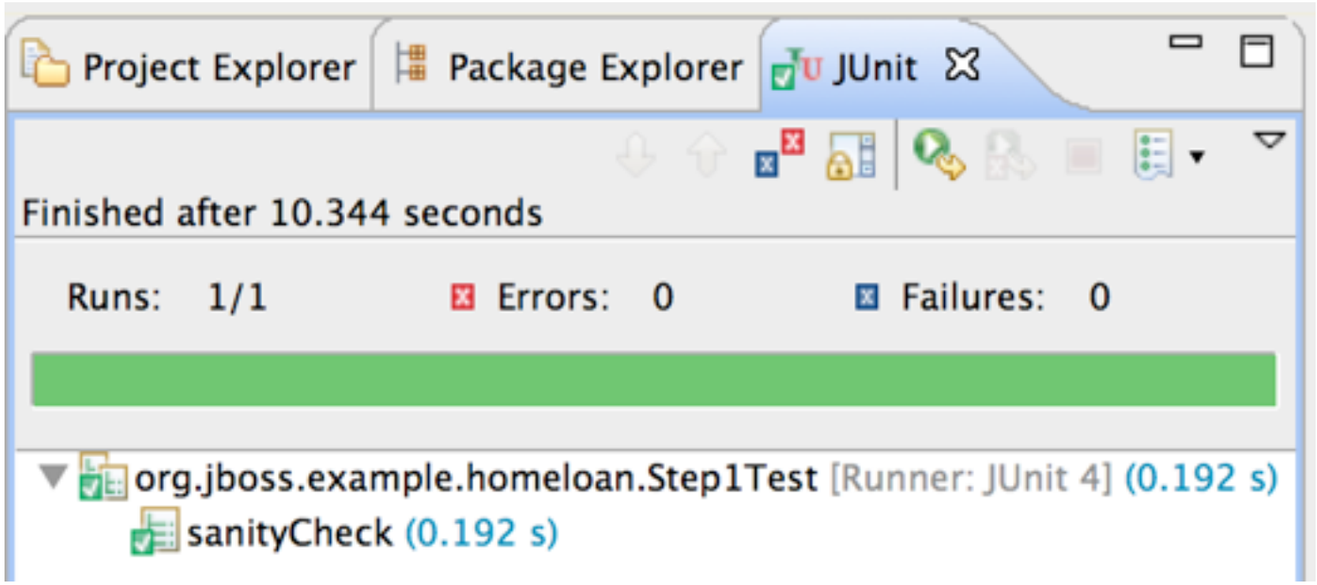
```
import junit.framework.Assert;

@RunWith(SwitchYardRunner.class)
@SwitchYardTestCaseConfig(
        config = SwitchYardTestCaseConfig.SWITCHYARD_XML,
        mixins = { CDIMixIn.class },
        exclude = "jms")
public class Step1Test {

    @ServiceOperation("IntakeService")
    private Invoker service;

    private SwitchYardTestKit testKit;

    @Test
    public void customerUpdate() throws Exception {

        // Mock providers for services called from IntakeService
        MockHandler lookUpService = testKit
            .registerInOutService("CustomerLookup")
            .replyWithOut(new Customer());
        MockHandler preQualService = testKit
            .registerInOutService("PreQualificationService")
            .forwardInToOut();

        service.operation("intake").sendInOnly(new LoanApplication());

        // validate that our downstream service references were called
        Assert.assertEquals(1, lookUpService.getMessages().size());
        Assert.assertEquals(1, preQualService.getMessages().size());
    }
```

Step 1

Success?

FYI

You should see a nice green bar indicating that the test passed. If you get a red bar indicating test failure, raise your hand and someone will be by to help.



Step 2

Customer Lookup

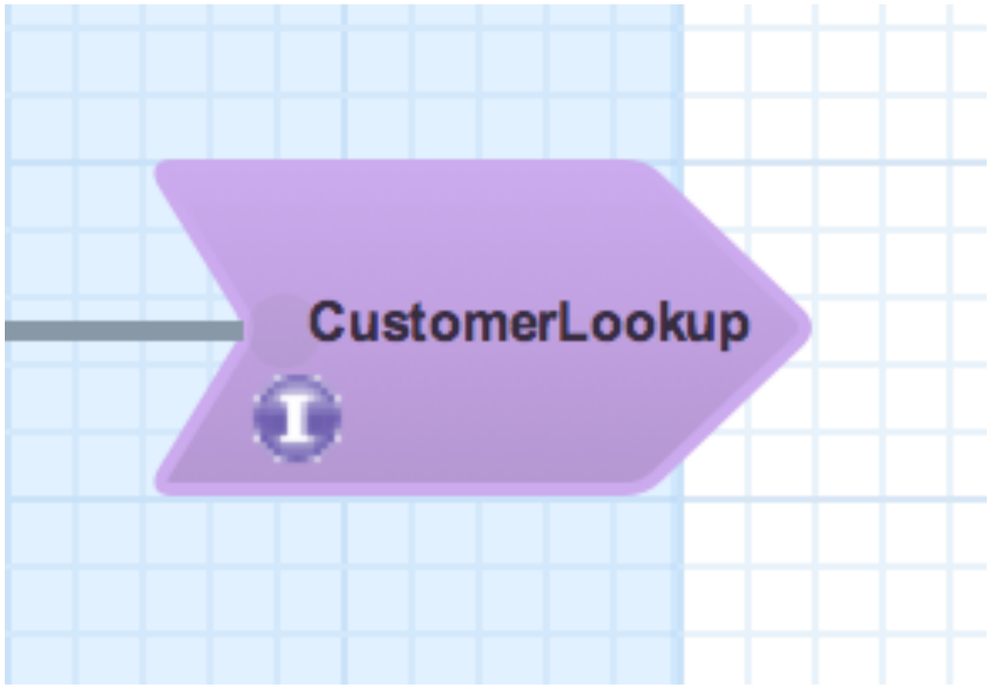
Goals

- *Add a SQL binding to the CustomerLookup reference to query the database.*
- *Enable a converter to convert between the SQL result and our domain object.*
- *Invoke CustomerLookup reference from Camel routing service.*
- *Run unit tests to verify changes.*

Composite Reference

FYI

A composite reference promotes a component reference to allow an implementation to invoke services outside the application through a binding. All composite references have a contract.



Step 2

Add SQL Binding

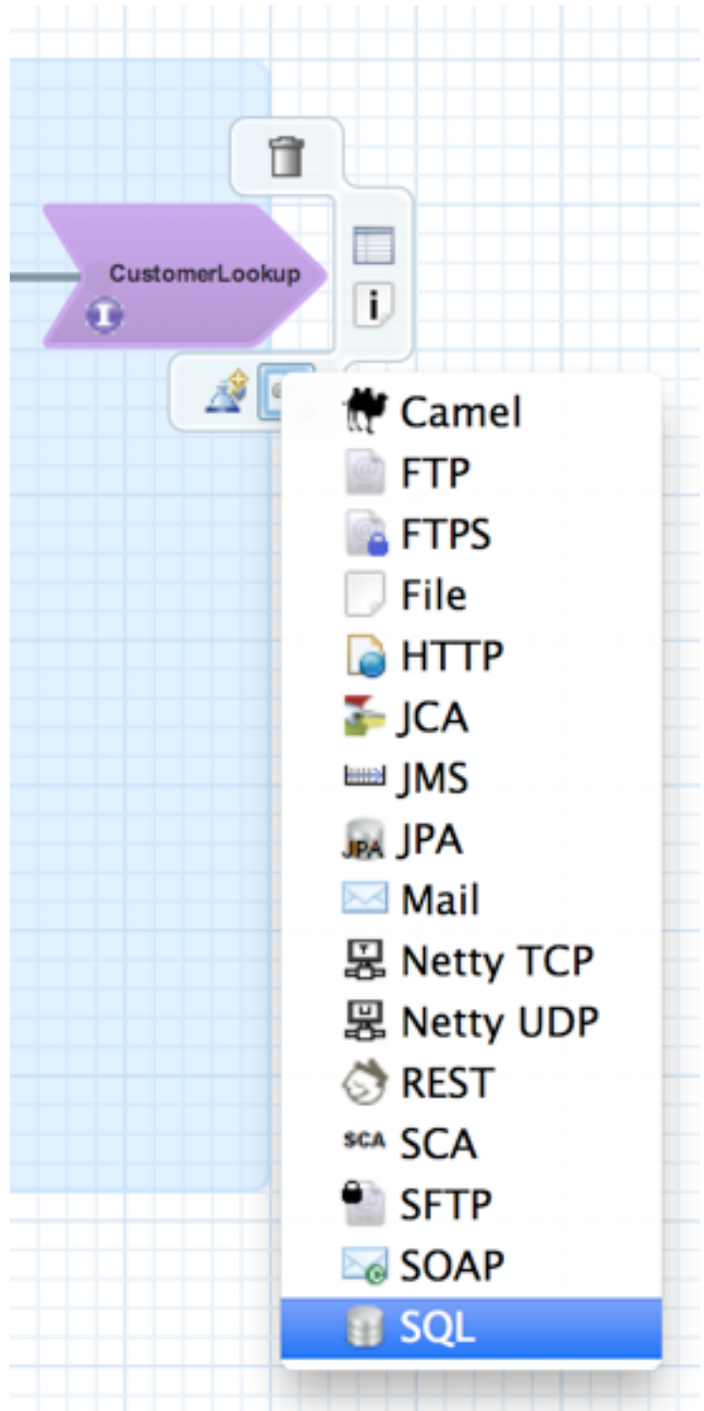
FYI

The button bar can be used to add bindings to composite services and references too. The button bar is awesome!

TODO

1. Hover over the CustomerLookup composite reference to access the button bar.

2. Click on the bindings button and select SQL.



Step 2

Configure SQL Binding

TODO

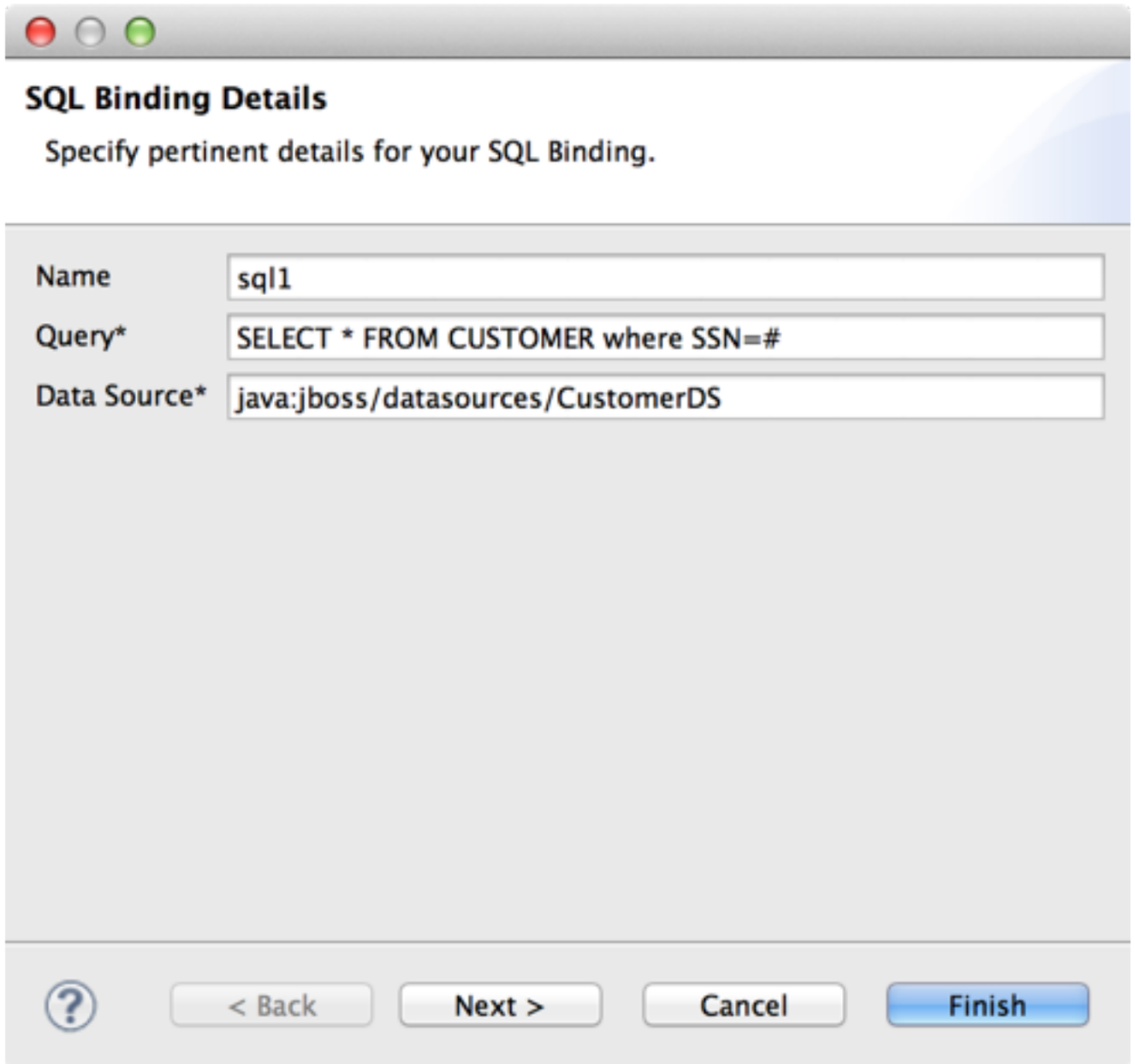
1. In the ‘Query’ field enter:

```
SELECT * FROM CUSTOMER where SSN=#
```

2. In the ‘Data Source’ field enter:

```
java:jboss/datasources/CustomerDS
```

3.Click Finish.



Step 2

Validate Changes

FYI

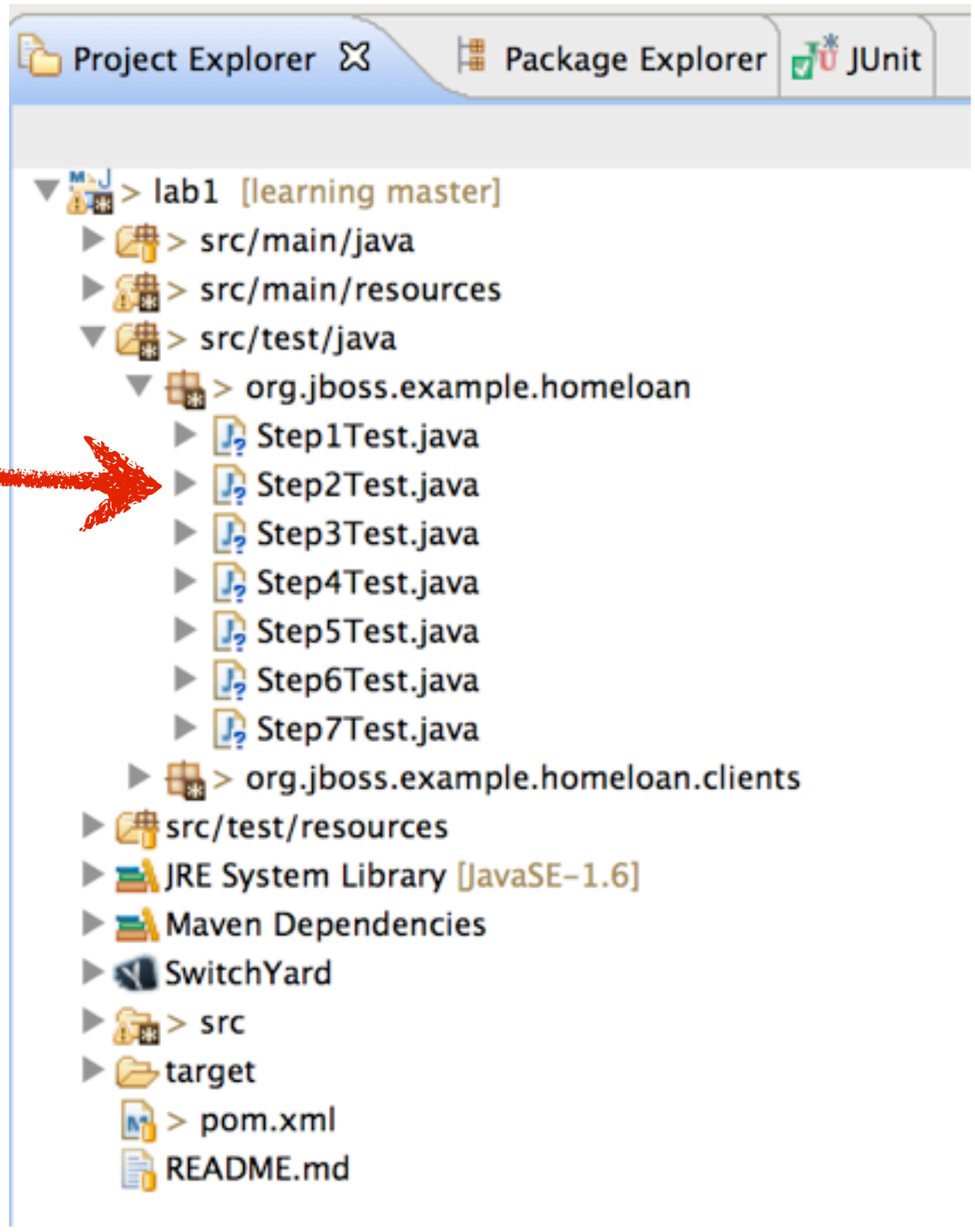
You have completed the changes required for step 2. Let's validate the changes using a service unit test.

TODO

1. Make sure the project is completely saved by selecting File -> Save All.

2. Double-click on Step2Test in the explorer to open the unit test.

3. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.

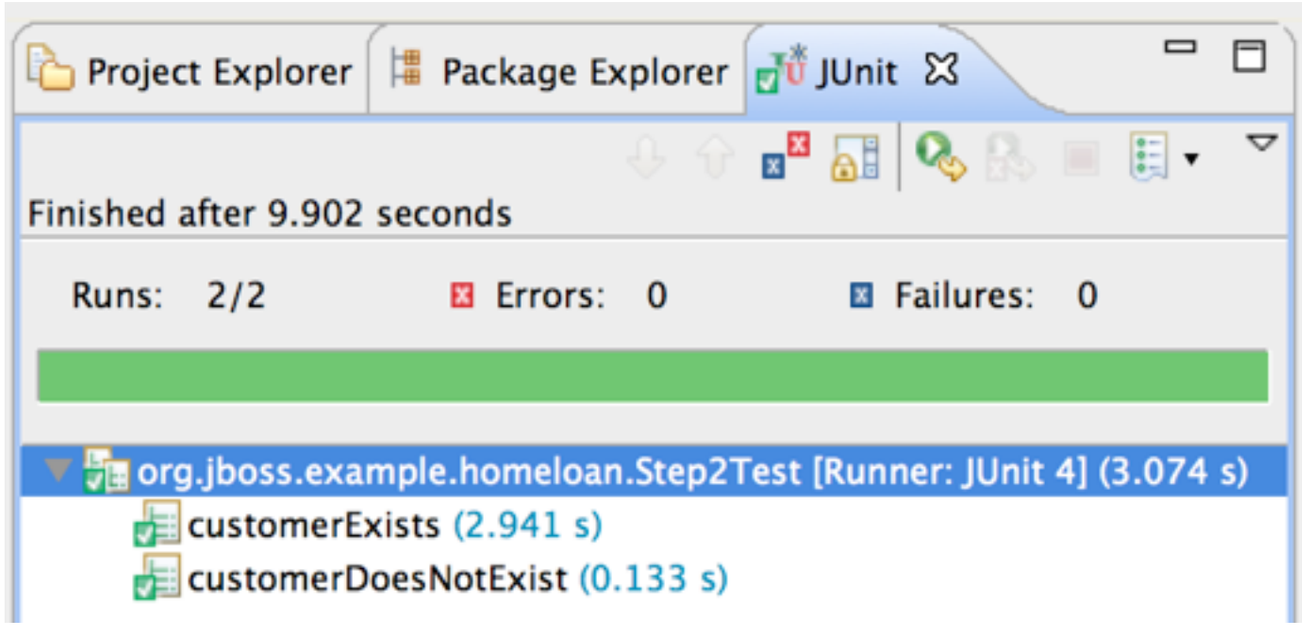


Step 2

Success?

FYI

You should see a nice green bar indicating that the test passed. If you get a red bar indicating test failure, raise your hand and someone will be by to help.



Step 3

Pre-qualification Workflow

Goals

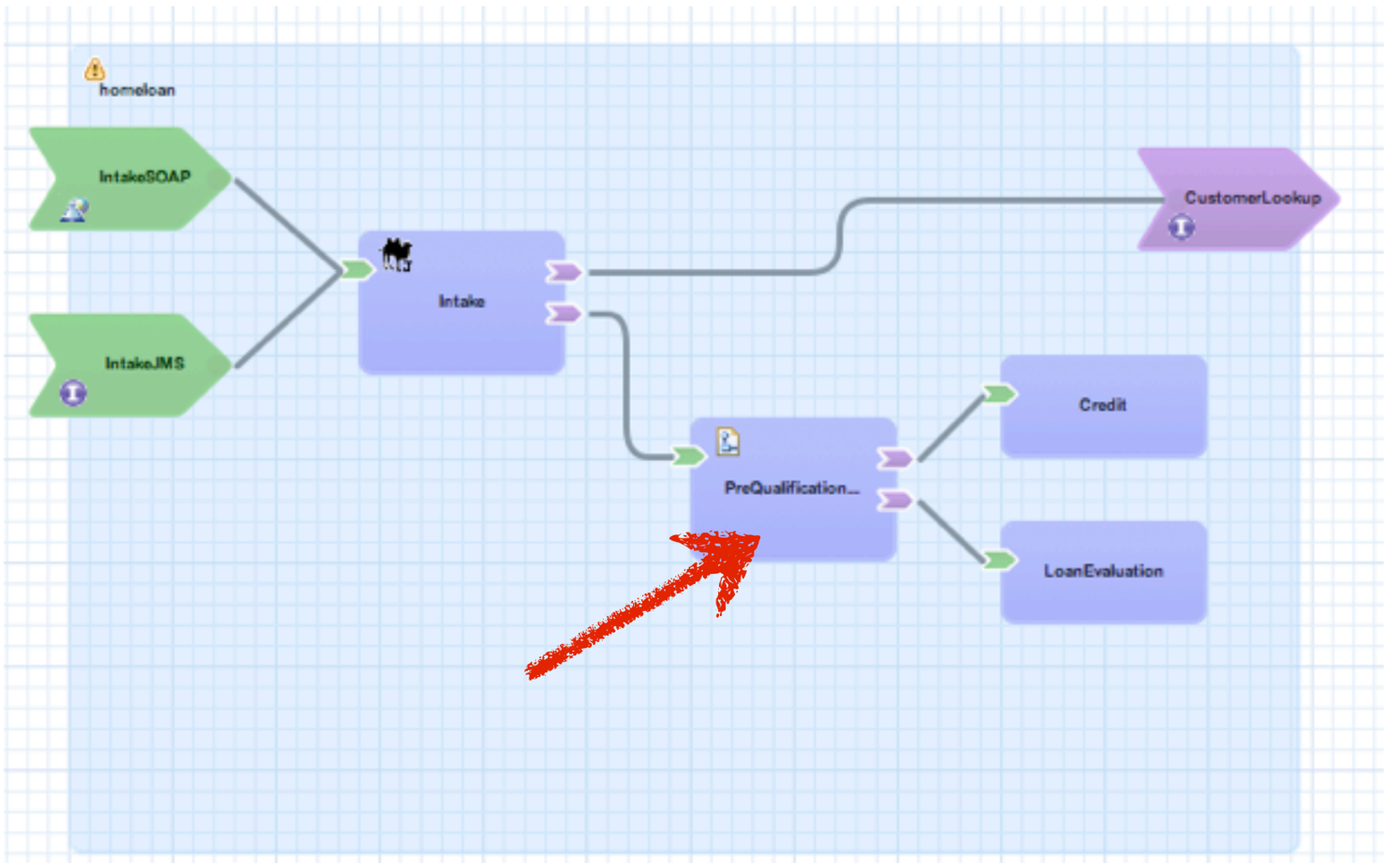
- *Inspect BPMN 2 workflow used for service orchestration.*
- *Create CreditService using CDI Bean.*
- *Create LoanEvaluationService using Drools business rules.*
- *Run unit tests to verify changes.*

Step 3

Open BPMN2 Process

TODO

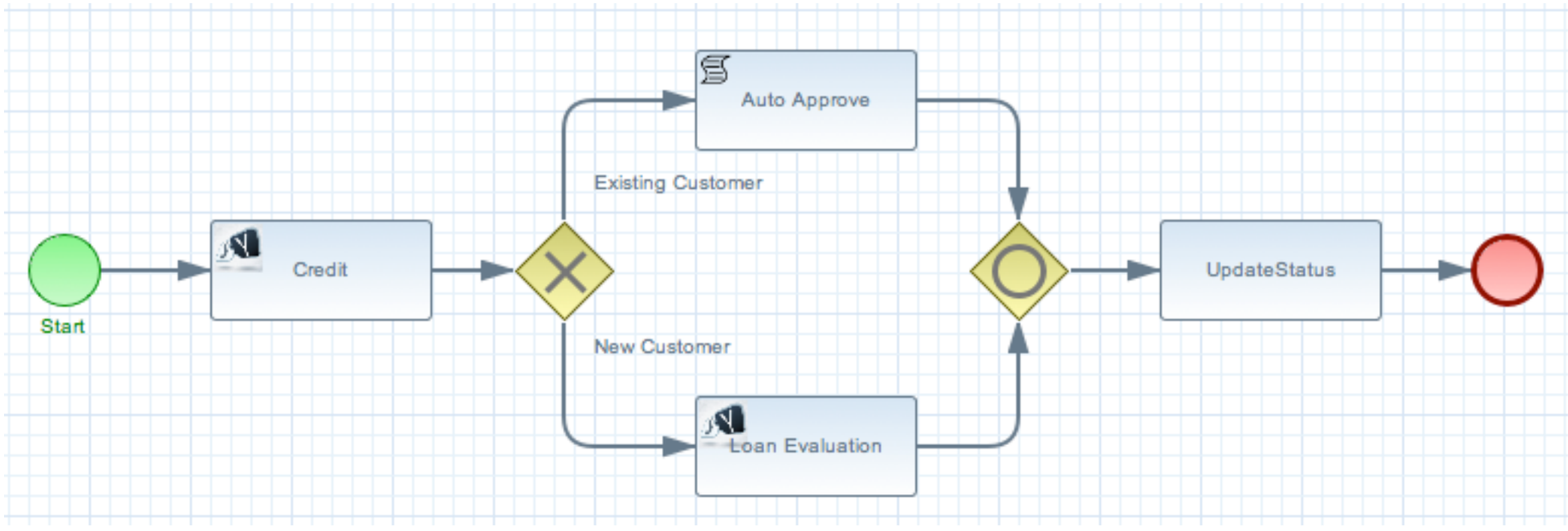
1. Double-click on the PreQualification component on the canvas.



Step 3

Inspect Process

FYI



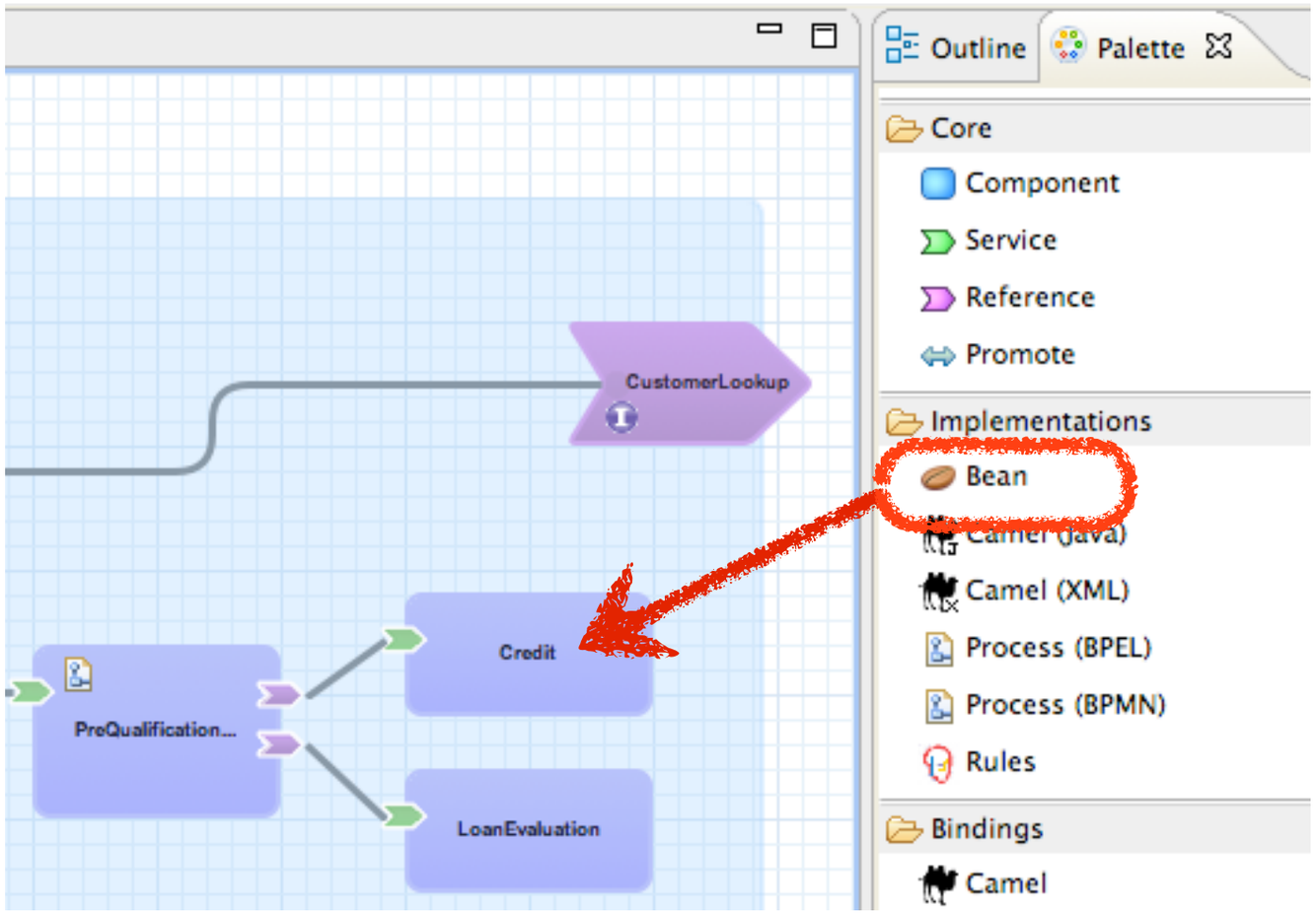
Step 3

Implement CreditService

FYI

TODO

1. Click and hold on the Bean implementation in the palette and drag it onto the Credit component on the canvas.
2. See next page for details on what to enter in the resulting dialog.



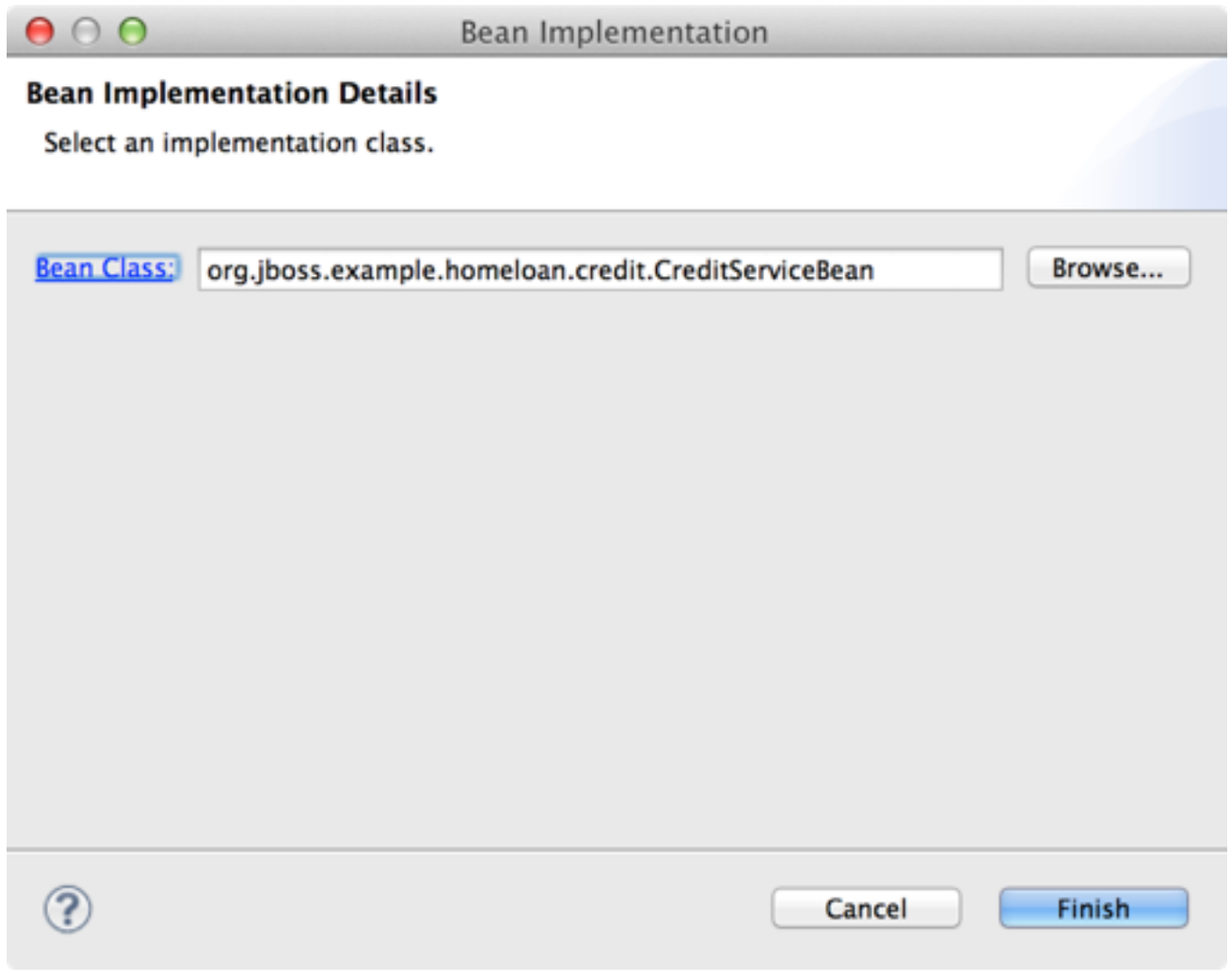
Step 3

Implement CreditService

FYI

TODO

1. Click on the Browse ... button to select the bean implementation.
2. Select CreditServiceBean from the list of beans.
3. Click Finish.



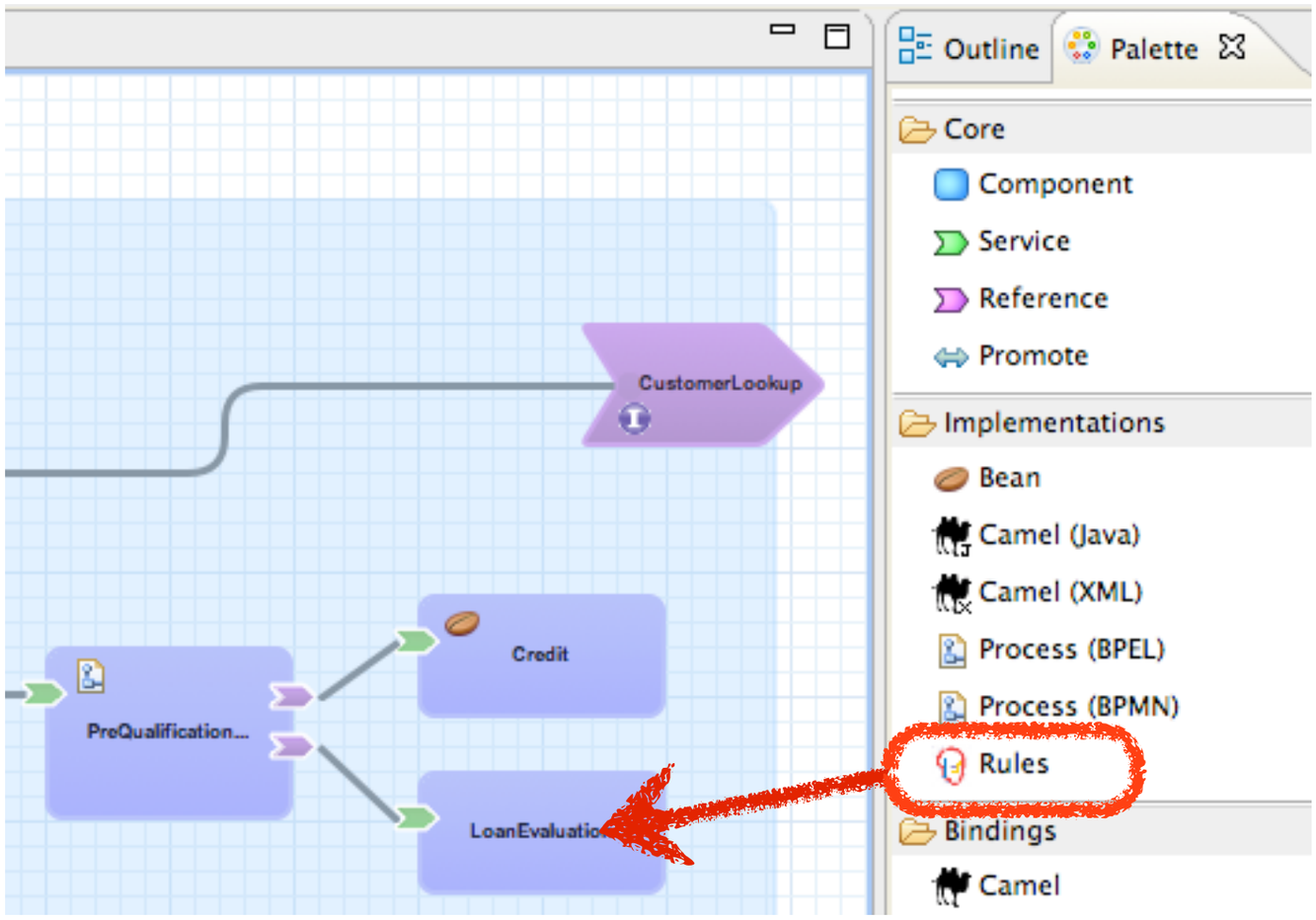
Step 3

Implement LoanEvaluation

FYI

TODO

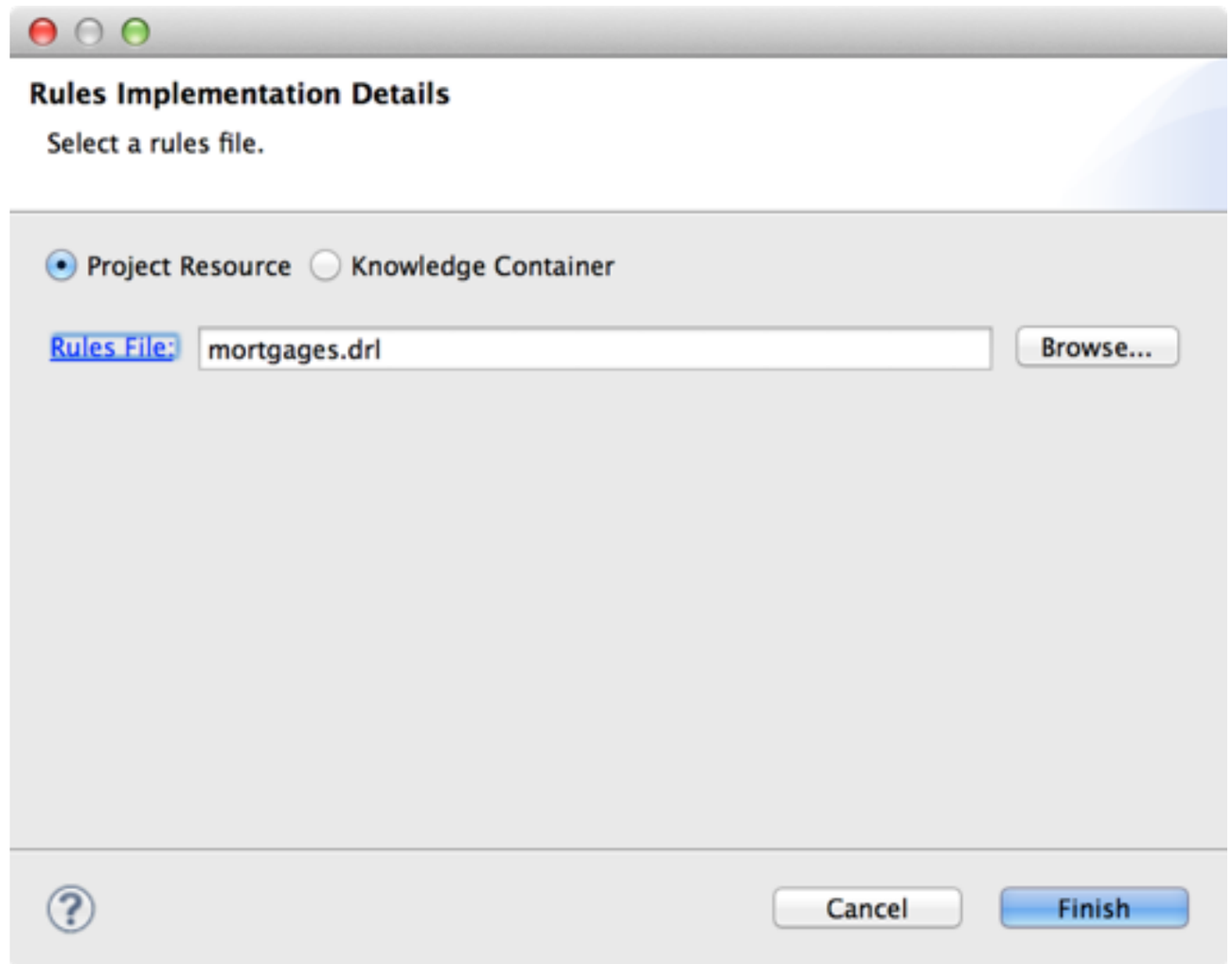
1. Click and hold on the Rules implementation in the palette and drag it onto the Credit component on the canvas.
2. See next page for details on what to enter in the resulting dialog.



Step 3

Implement CreditService

FYI



TODO

1. Click on the Browse ... button to select the rules file to use.
2. Select mortgages.drl from the list.
3. Click Finish.

Step 3

Validate Changes

FYI

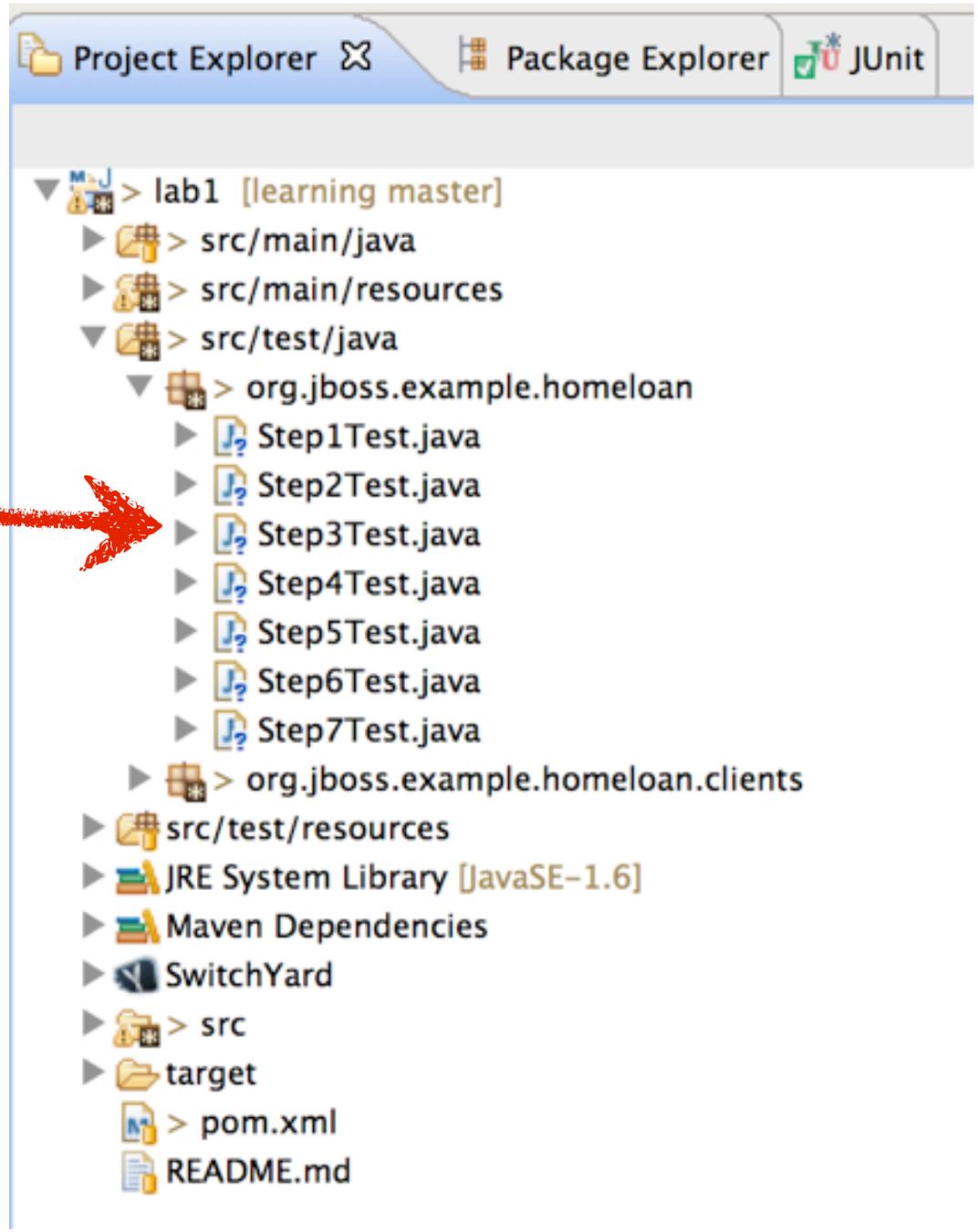
You have completed the changes required for step 3. Let's validate the changes using a service unit test.

TODO

1. Make sure the project is completely saved by selecting File -> Save All.

2. Double-click on Step3Test in the explorer to open the unit test.

3. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.

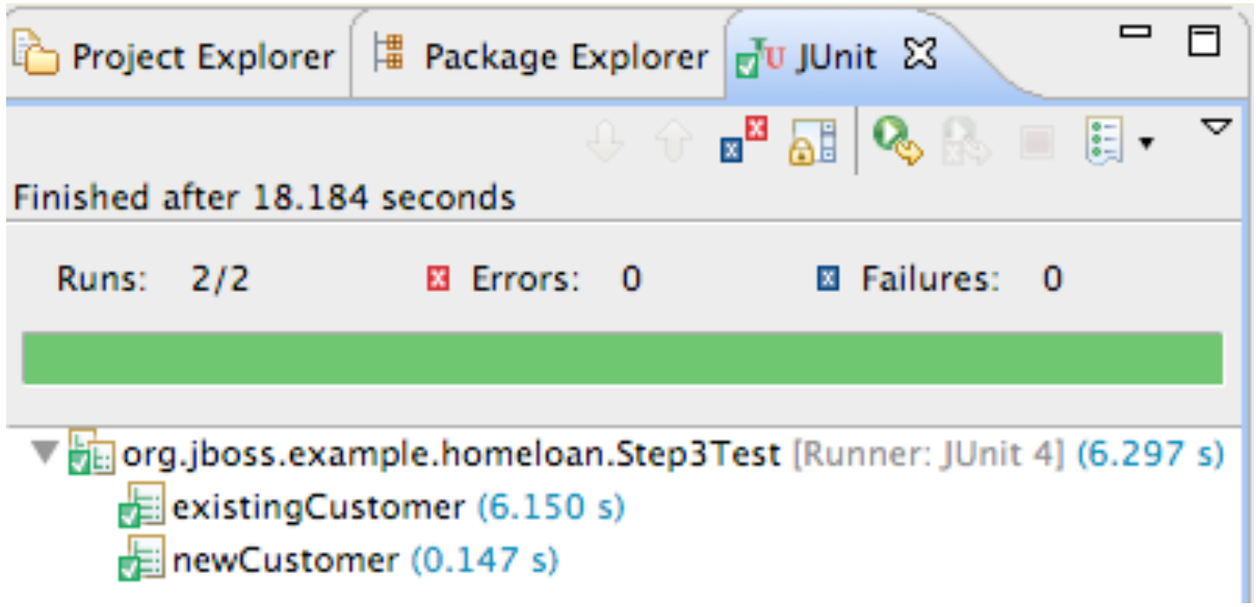


Step 3

Success?

FYI

You should see a nice green bar indicating that the test passed. If you get a red bar indicating test failure, raise your hand and someone will be by to help.



Step 4

Advanced Routing

Goals

- *Invoke Bean from Camel route.*
- *Add message filtering to Camel routing logic.*
- *Run unit tests to verify changes.*

Step 4

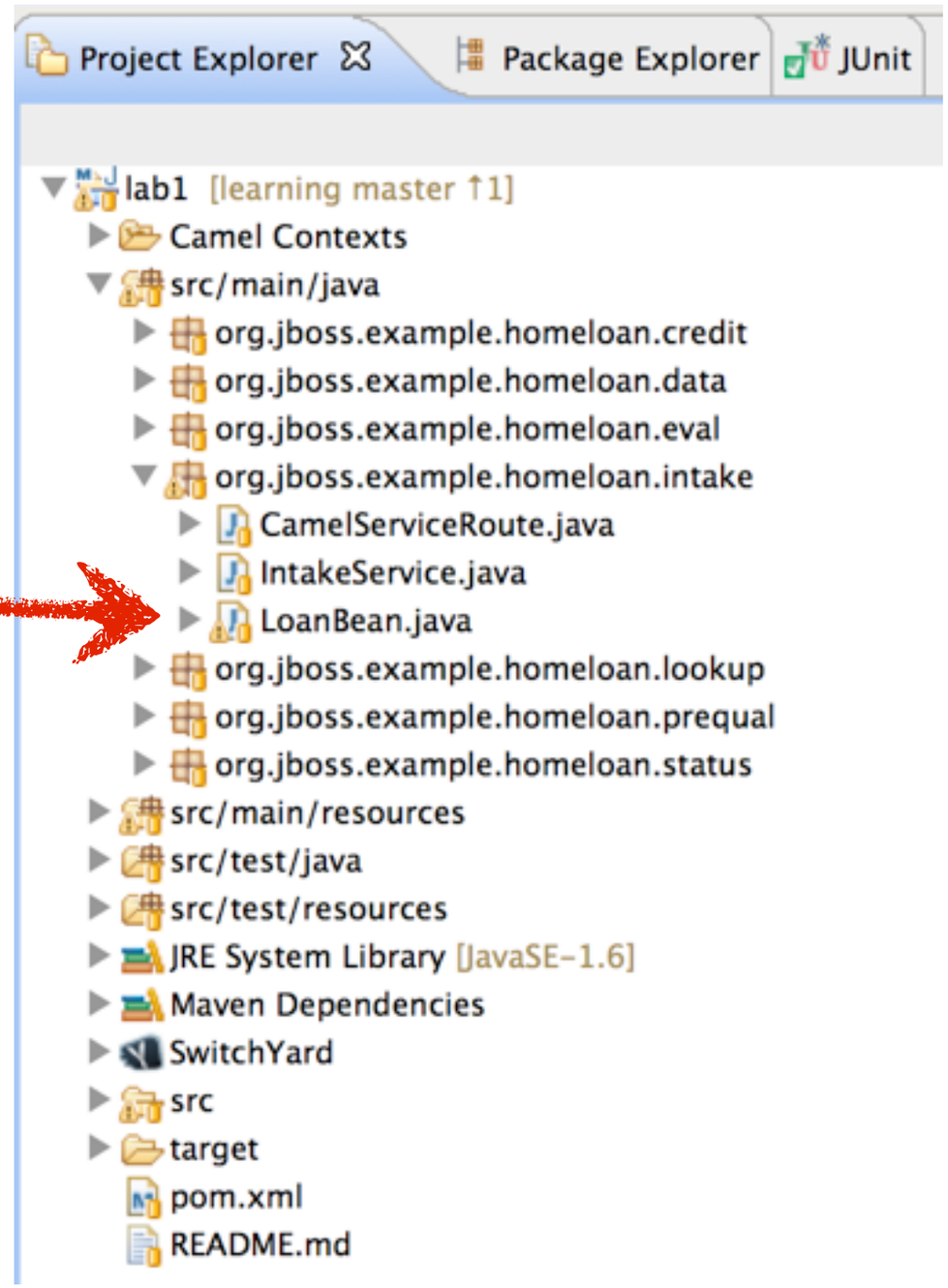
Calling a Bean from Camel

FYI

Bea

TODO

1. Double-click on LoanBean.java to open up the LoanBean object.



Step 4

Calling a Bean from Camel

FYI

TODO

1. Add the following annotation above the class definition for LoanBean:

```
@Named("Loan")
```

LoanBean.java

```
package org.jboss.example.homeloan.intake;

import java.text.DecimalFormat;

@Named("Loan")
public class LoanBean {

    public void customerUpdate(LoanApplication app, Customer custo
        app.getApplicant().setFirstName(customer.getFirstName());
        app.getApplicant().setLastName(customer.getLastName());
        app.getApplicant().setPostalCode(customer.getPostalCode())
        app.getApplicant().setStreetAddress(customer.getStreetAddr
        app.getApplicant().setSavingsBalance(customer.getSavingsBa
        app.getApplicant().setCheckingBalance(customer.getChecking
    }

    public void summary(LoanApplication app) {
        DecimalFormat df = new DecimalFormat("###,###.00");
```

Step 4

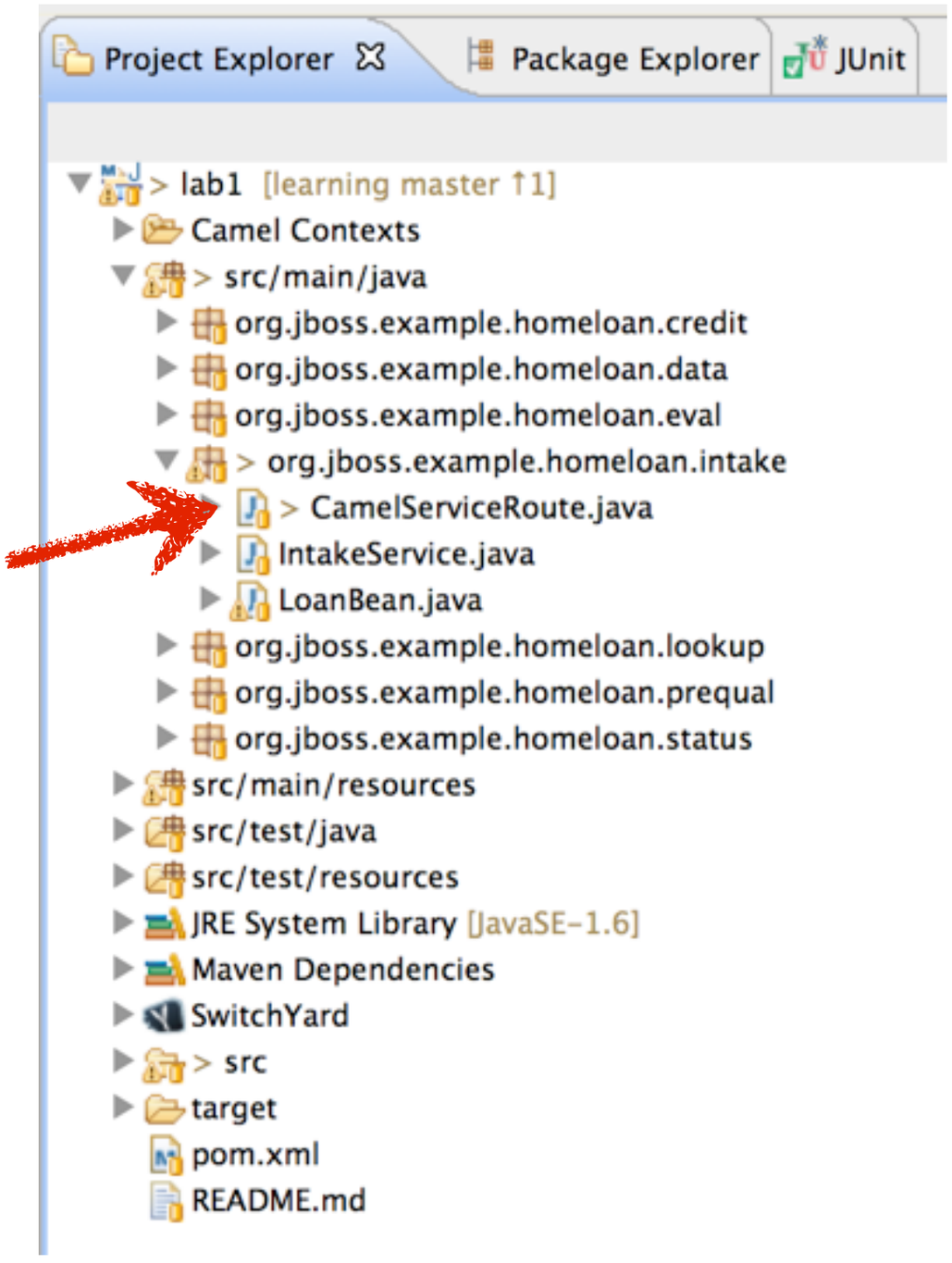
Add Routing Logic to Camel Route

FYI

Bea

TODO

1. Double-click on CamelServiceRoute.java to open up the Camel route.



Step 4

Add Routing Logic to Camel Route

FYI

TODO

1. Add additional routing logic between the BEGIN and END comments. See next page for details on what to add.

CamelServiceRoute.java

```
package org.jboss.example.homeloan.intake;

import org.apache.camel.builder.RouteBuilder;

public class CamelServiceRoute extends RouteBuilder {

    public void configure() {
        from("switchyard://IntakeService")
            .setProperty("LoanApplication").simple("${body}")
            .setBody().simple("${body.applicant.ssn}")
            .to("switchyard://CustomerLookup")
            // BEGIN - additional routing logic

            // END - additional routing logic
            .setBody().property("LoanApplication")
            .to("switchyard://PreQualificationService");
    }

}
```

Step 4

Add Routing Logic to Camel Route

FYI

```
// BEGIN - additional routing logic
.filter(simple("${body} != null && ${body.size} > 0"))
    .beanRef("Loan", "customerUpdate(${property.LoanApplication}, ${body[0]})")
    .setHeader("ExistingCustomer").constant(true)
.end()
.beanRef("Loan", "summary(${property.LoanApplication})")
// END - additional routing logic
```

FYI

You can copy and paste the text from this page into your editor.

Step 4

Add Routing Logic to Camel Route

FYI

This is the completed route definition.

CamelServiceRoute.java

```
package org.jboss.example.homeloan.intake;

import org.apache.camel.builder.RouteBuilder;

public class CamelServiceRoute extends RouteBuilder {

    public void configure() {
        from("switchyard://IntakeService")
            .setProperty("LoanApplication").simple("${body}")
            .setBody().simple("${body.applicant.ssn}")
            .to("switchyard://CustomerLookup")
            // BEGIN - additional routing logic
            .filter(simple("${body} != null && ${body.size} > 0"))
                    .beanRef("Loan", "customerUpdate(${property.LoanApplication}, ${body})")
                    .setHeader("ExistingCustomer").constant(true)
            .end()
            .beanRef("Loan", "summary(${property.LoanApplication})")
            // END - additional routing logic
            .setBody().property("LoanApplication")
            .to("switchyard://PreQualificationService");
    }

}
```

Step 4
Validate Changes

FYI

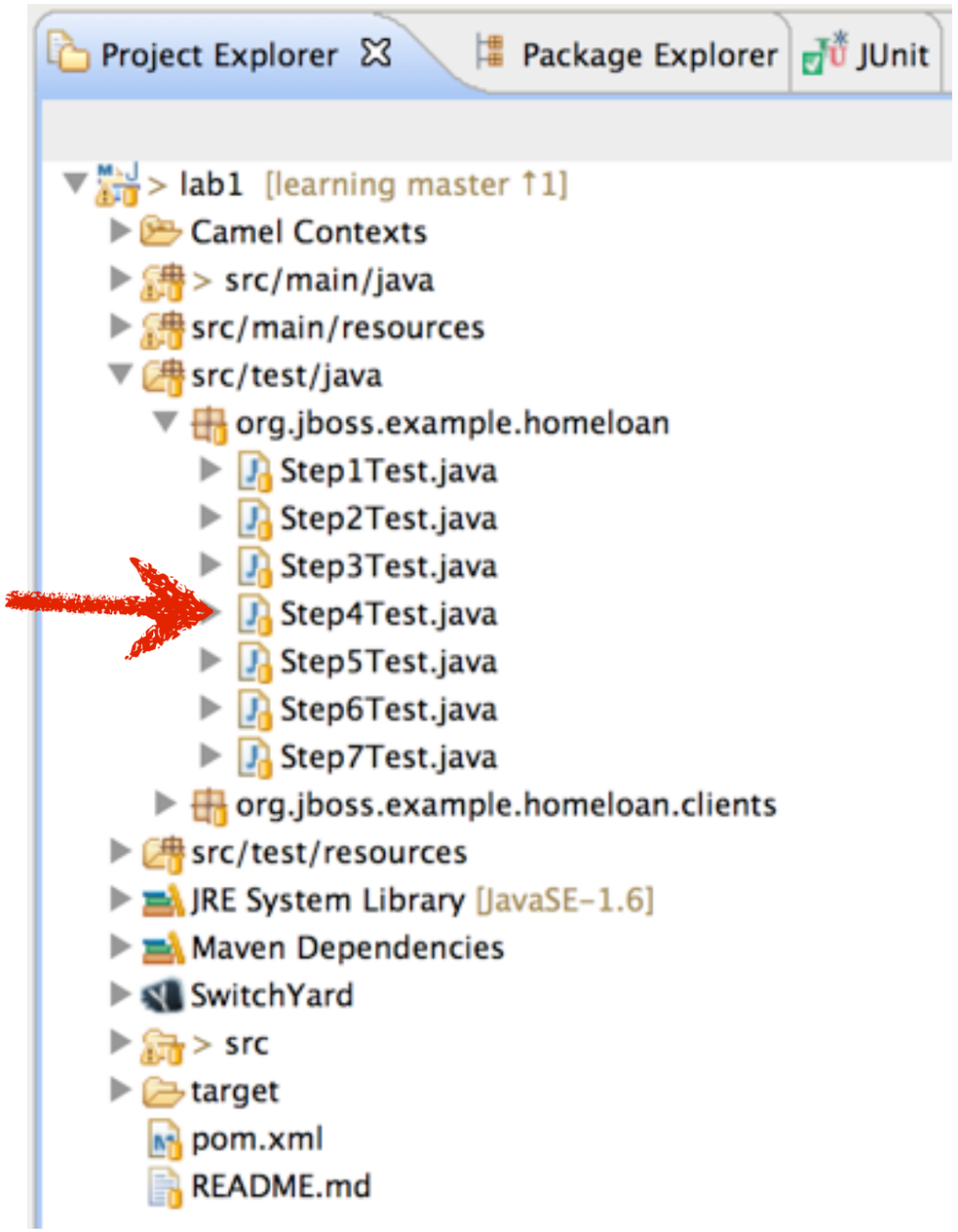
You have completed the changes required for step 4. Let's validate the changes using a service unit test.

TODO

1. Make sure the project is completely saved by selecting File -> Save All.

2. Double-click on Step4Test in the explorer to open the unit test.

3. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.

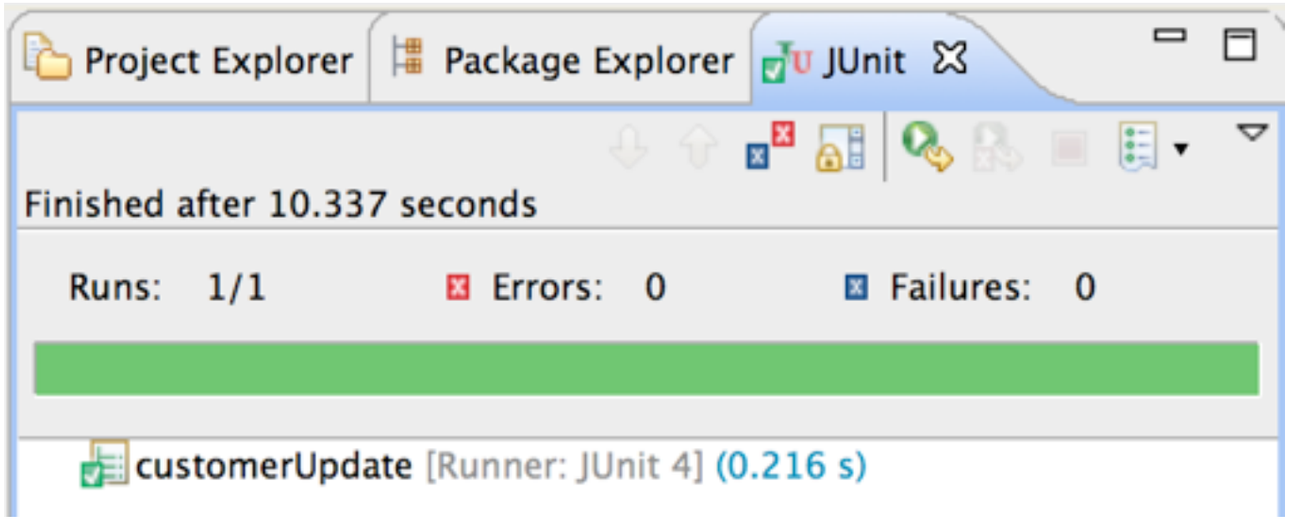


Step 4

Success?

FYI

You should see a nice green bar indicating that the test passed. If you get a red bar indicating test failure, raise your hand and someone will be by to help.



Step 5

SOAP Service Binding

Goals

- *Add SOAP binding to composite service.*
- *Run unit tests to verify changes.*

Step 5

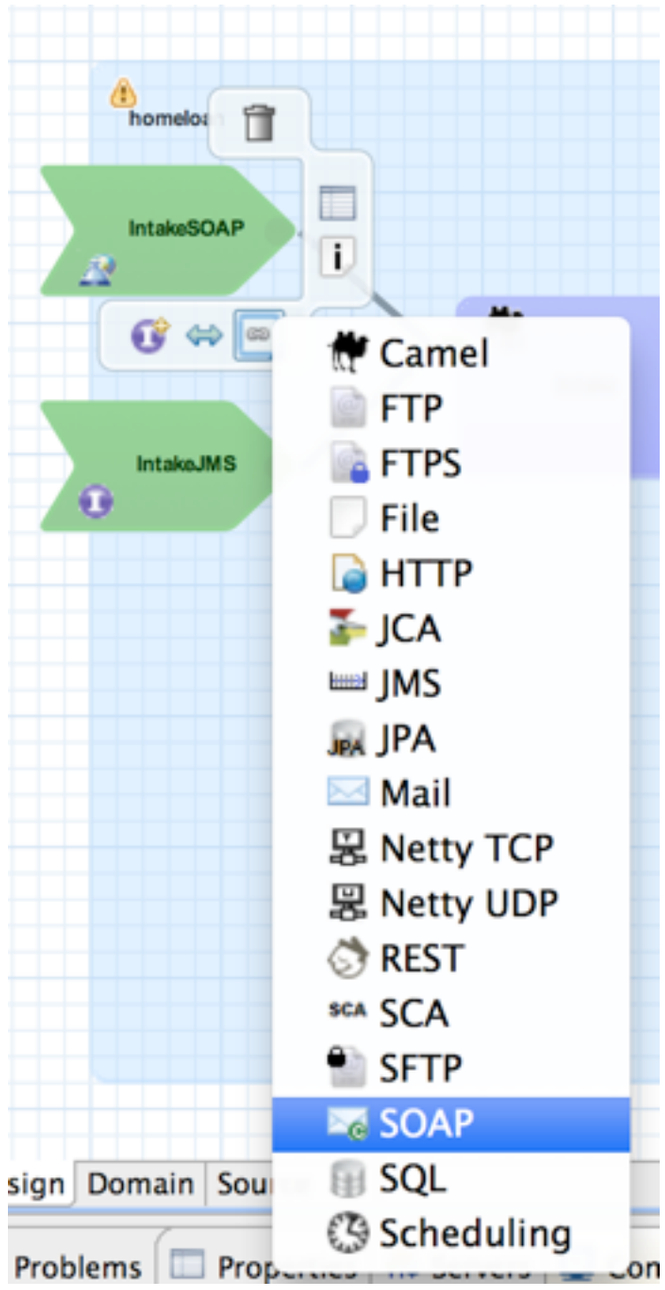
Add SOAP Service Binding

FYI

TODO

1. Hover over the IntakeSOAP composite service to access the button bar.

2. Click on the bindings button and select SOAP.



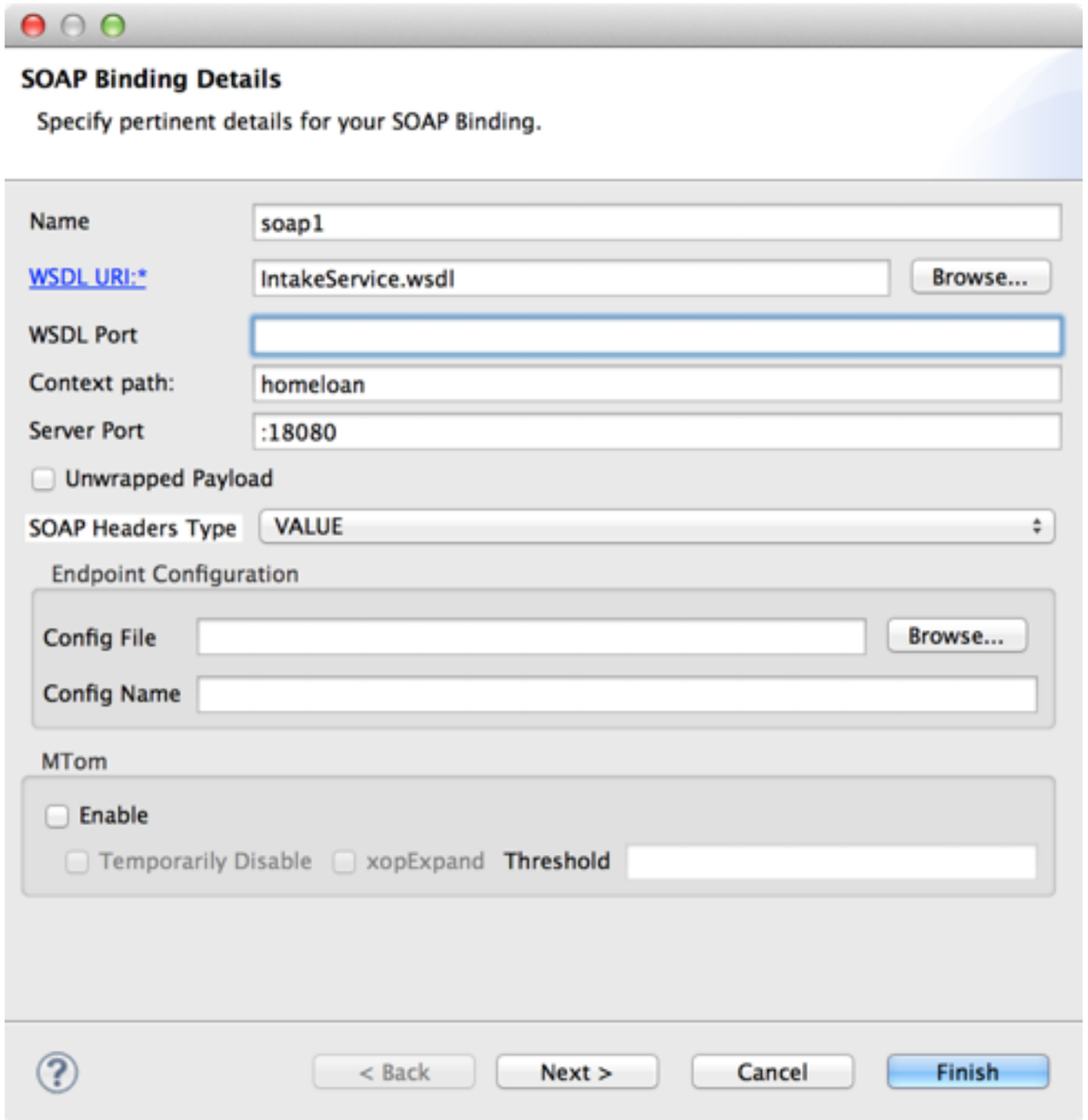
Step 5

Configure SOAP Binding

TODO

1. Context path : homeloan
   Server Port : 18080

2. Click Finish.



Step 5
Validate Changes

FYI

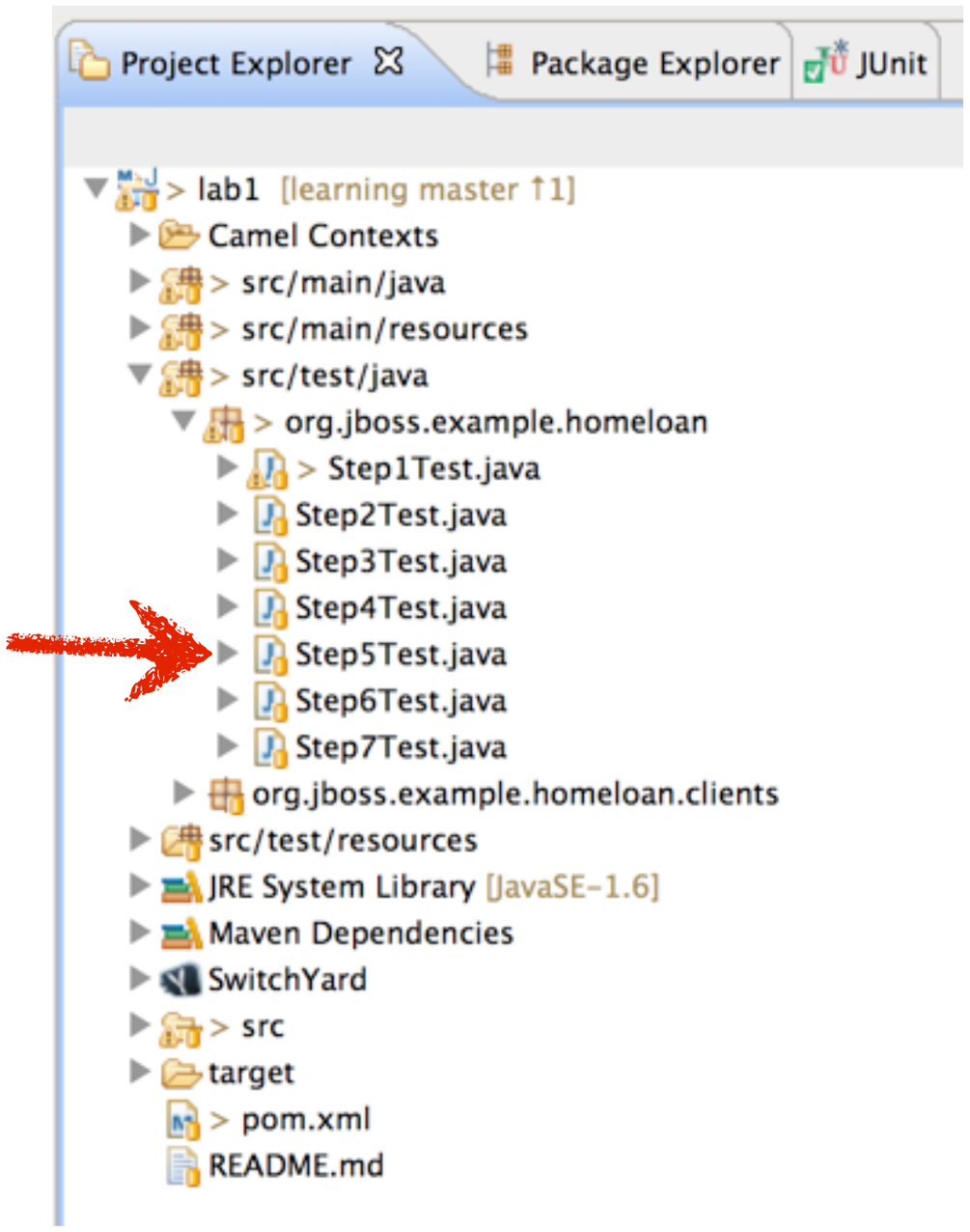
You have completed the changes required for step 5. Let's validate the changes using a service unit test.

TODO

1. Make sure the project is completely saved by selecting File -> Save All.

2. Double-click on Step5Test in the explorer to open the unit test.

3. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.

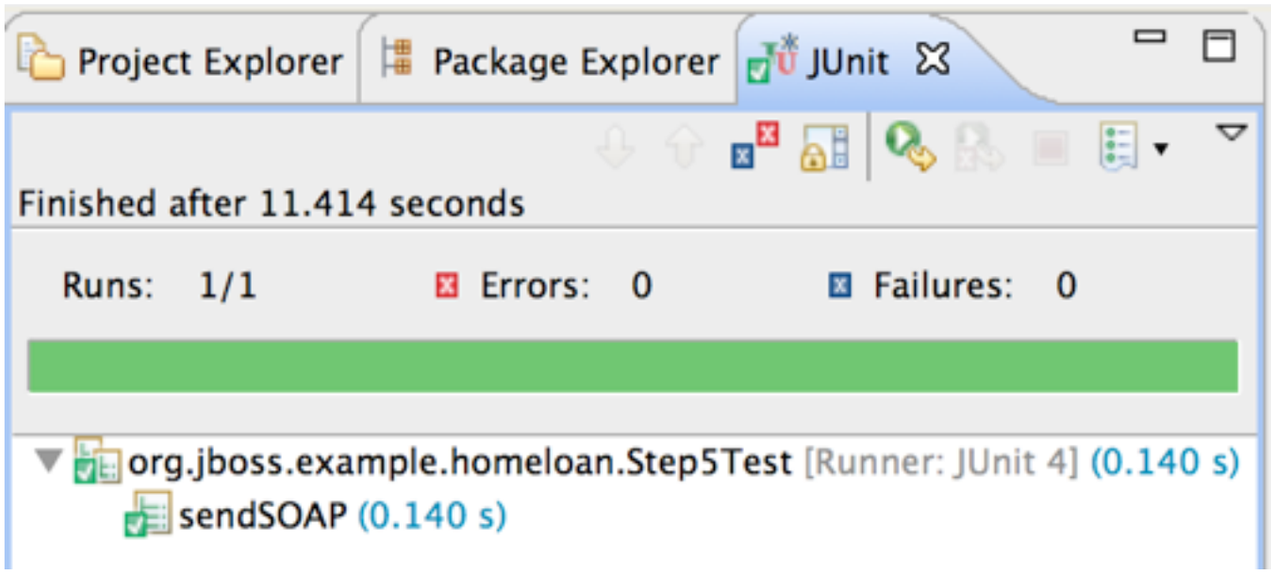


Step 5

Success?

FYI

You should see a nice green bar indicating that the test passed. If you get a red bar indicating test failure, raise your hand and someone will be by to help.



Step 6

JMS Service Binding

Goals

- *Add JMS binding to composite service.*
- *Run unit tests to verify changes.*

Step 6

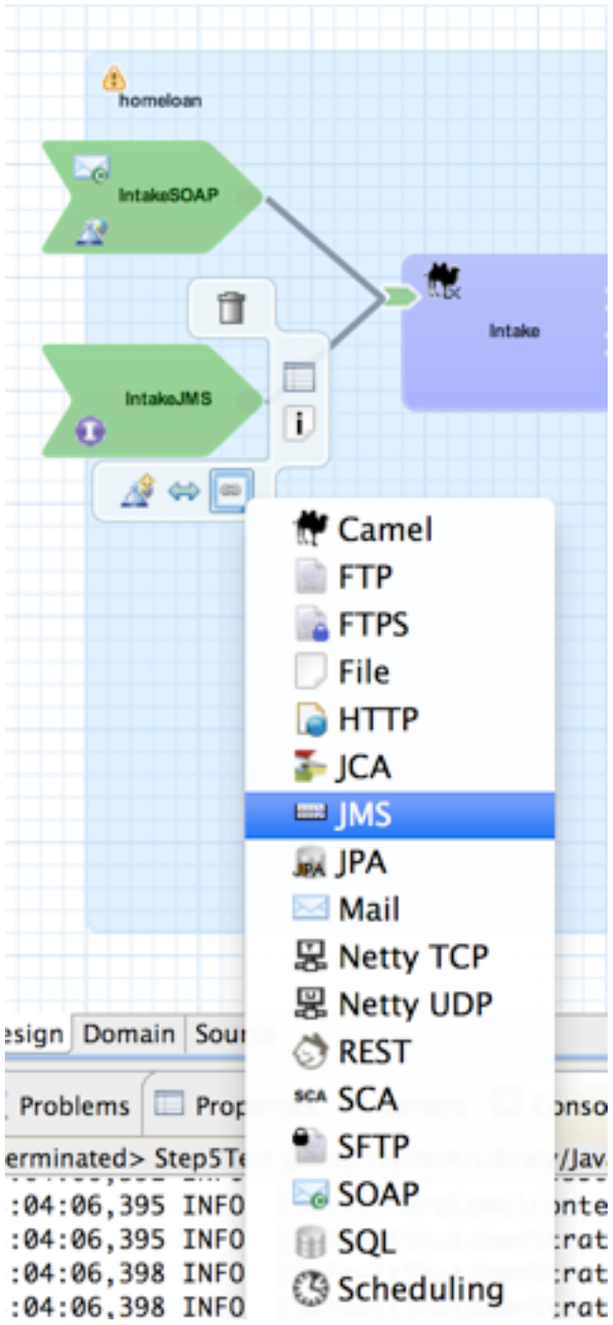
Add JMS Service Binding

FYI

TODO

1. Hover over the IntakeJMS composite service to access the button bar.

2. Click on the bindings button and select JMS.



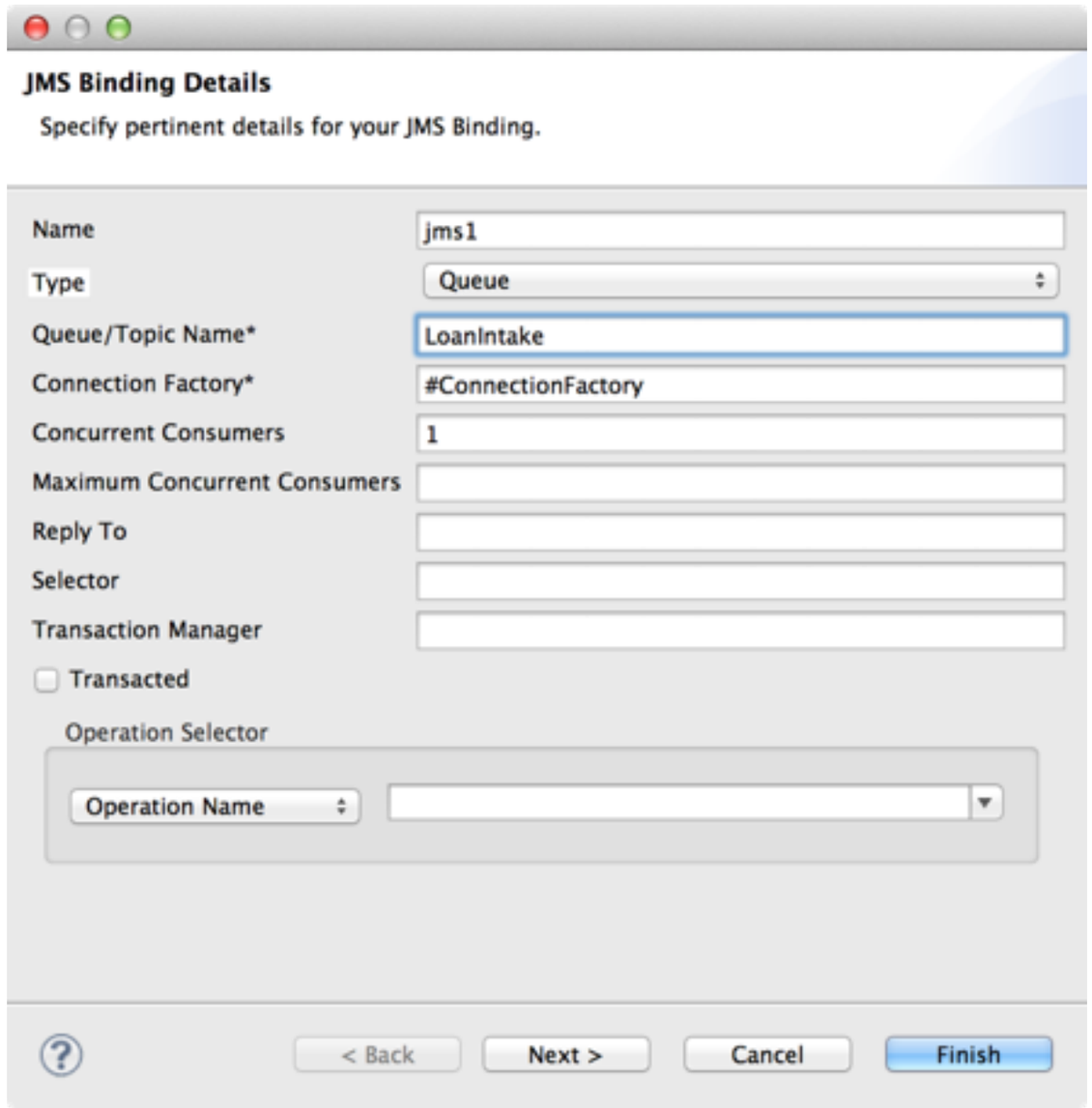
Step 6

Configure JMS Binding

TODO

1. `Queue Name : LoanIntake`

2. Click Finish.



Step 6

Validate Changes

FYI

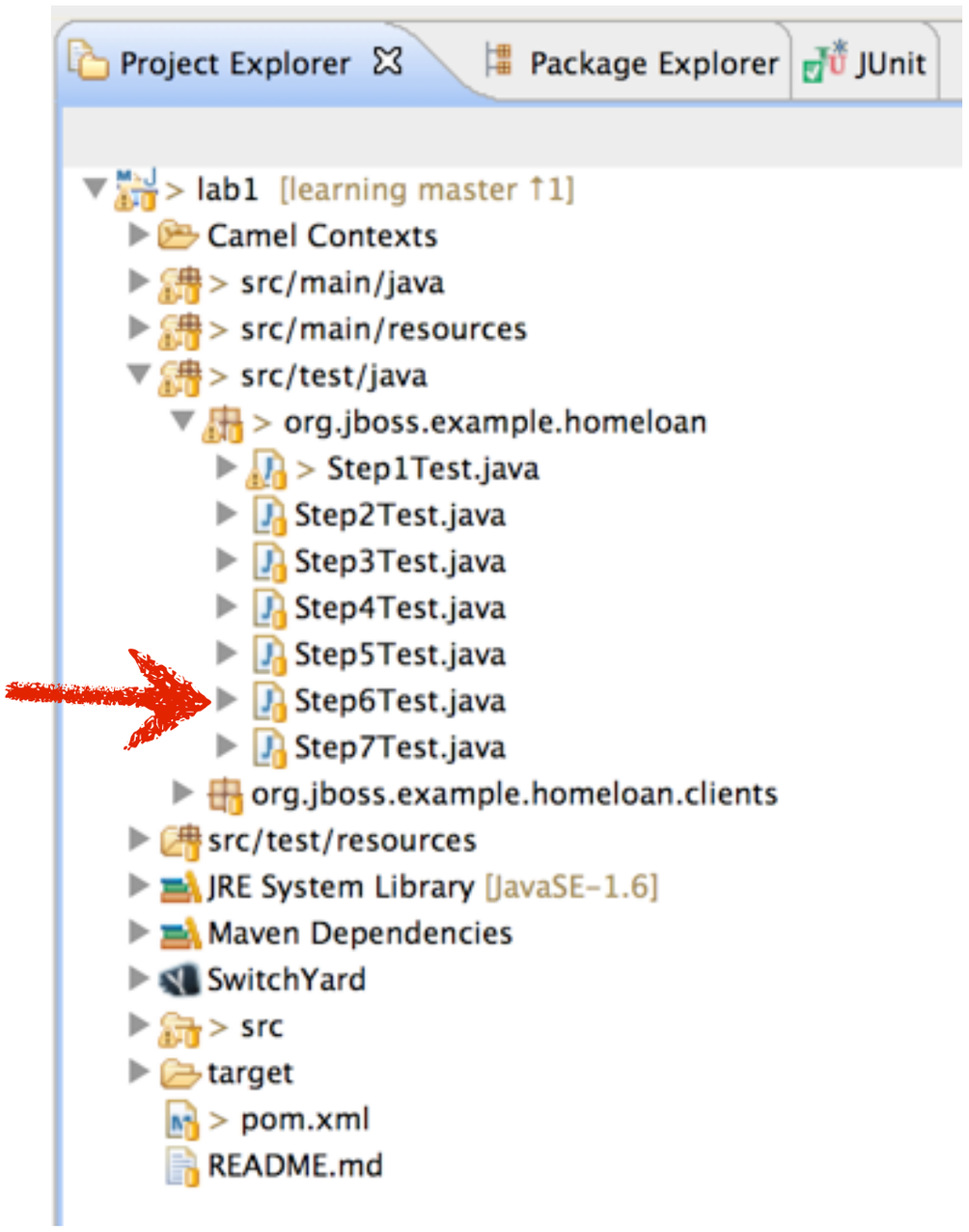
You have completed the changes required for step 6. Let's validate the changes using a service unit test.

TODO

1. Make sure the project is completely saved by selecting File -> Save All.

2. Double-click on Step6Test in the explorer to open the unit test.

3. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.

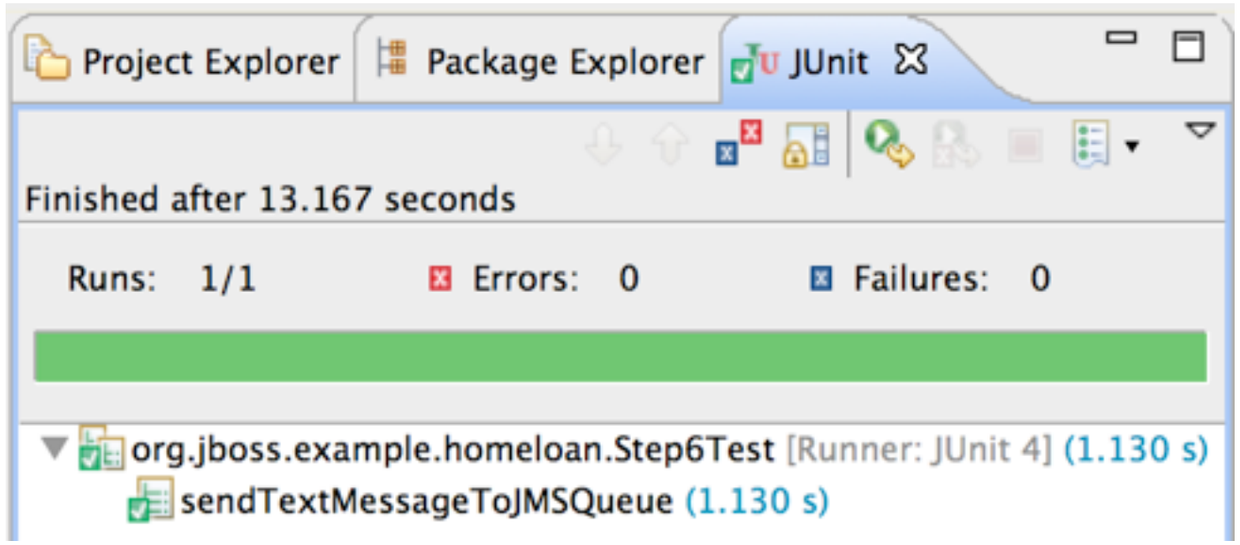


Step 6

Success?

FYI

You should see a nice green bar indicating that the test passed. If you get a red bar indicating test failure, raise your hand and someone will be by to help.



Step 7

REST Status Service

Goals

- *Create status service implementation using CDI Bean.*
- *Promote StatusService using Java contract.*
- *Define RESTful interface using JAX-RS.*
- *Add REST binding to StatusService.*
- *Run unit tests to verify changes.*

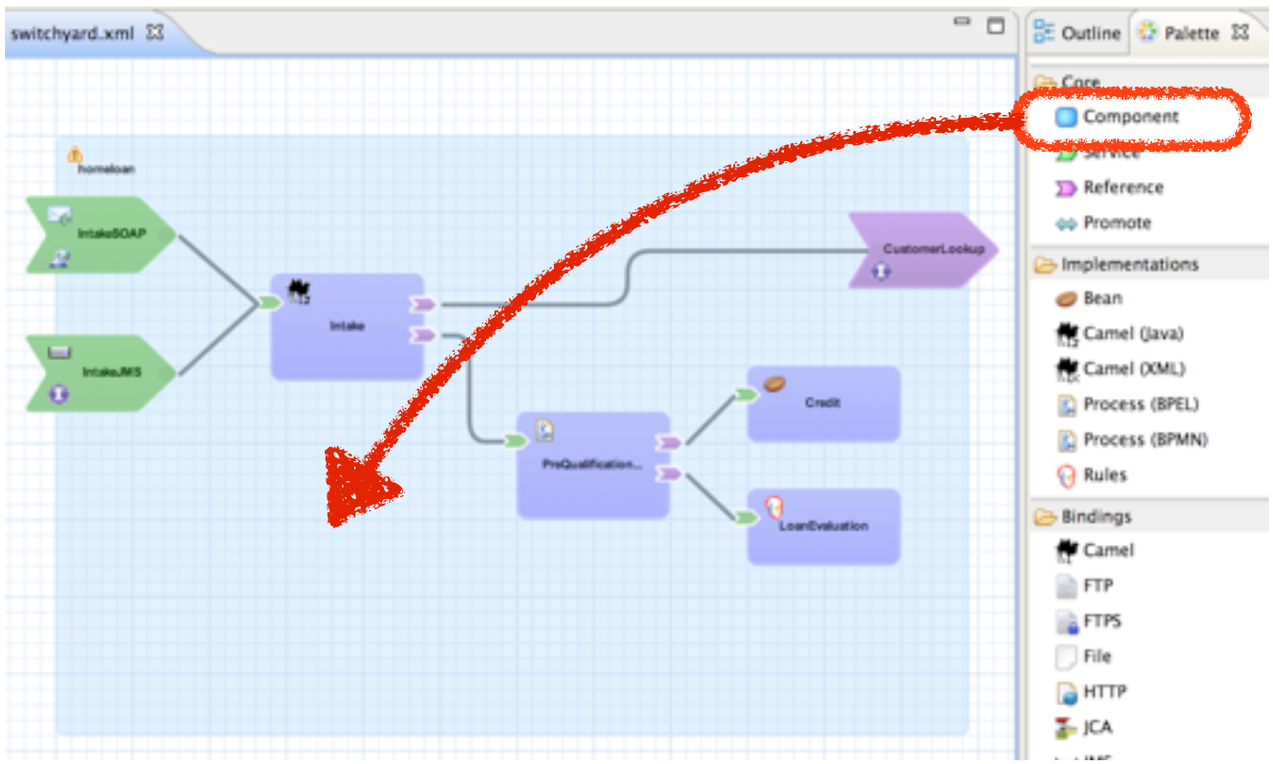
Step 7

StatusService Implementation

FYI

TODO

1. Click and hold on the Component icon in the palette and drag it onto an empty area of the canvas.



Step 7

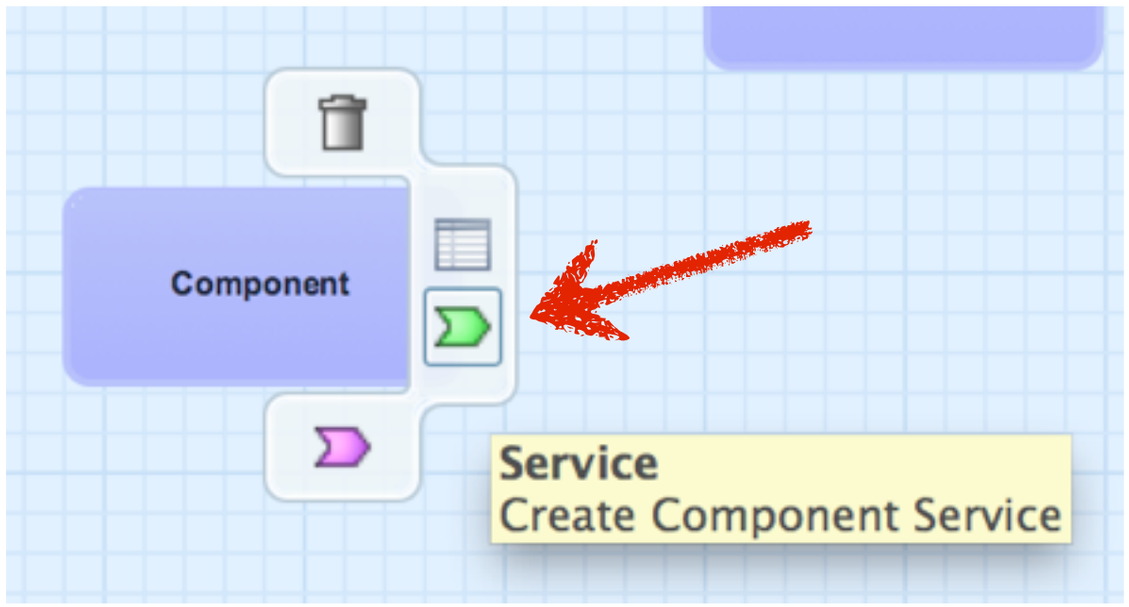
StatusService Implementation

FYI

TODO

1. Hover over the new Component to access the button bar.

2. Click on the green Service icon to add a service to the component.



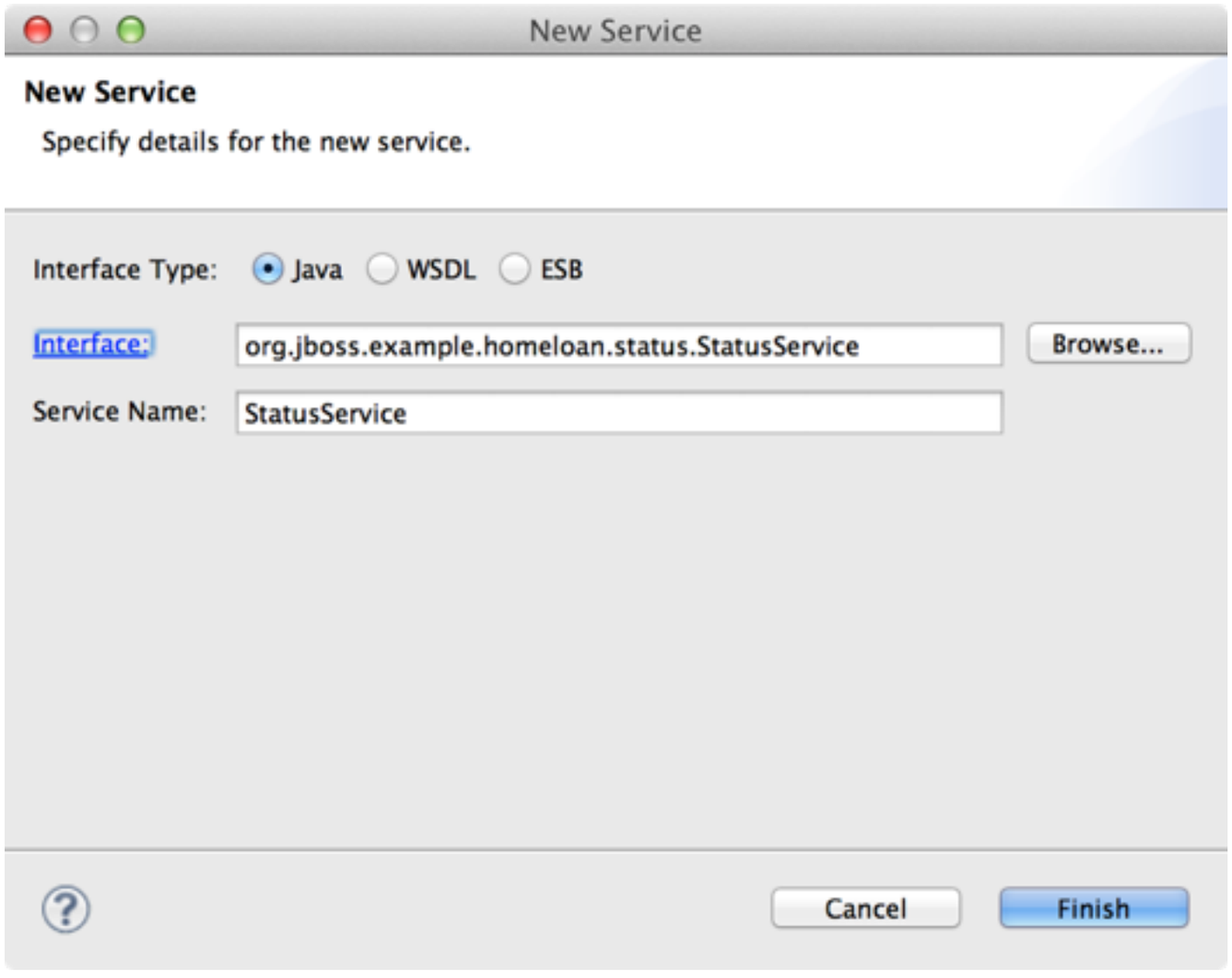
Step 7

StatusService Implementation

FYI

TODO

1. Click on the Browse ... button to select the service interface.
2. Begin typing ‘StatusService’ in the resulting dialog and select the StatusService interface when you see it in the list.
3. Click Finish.



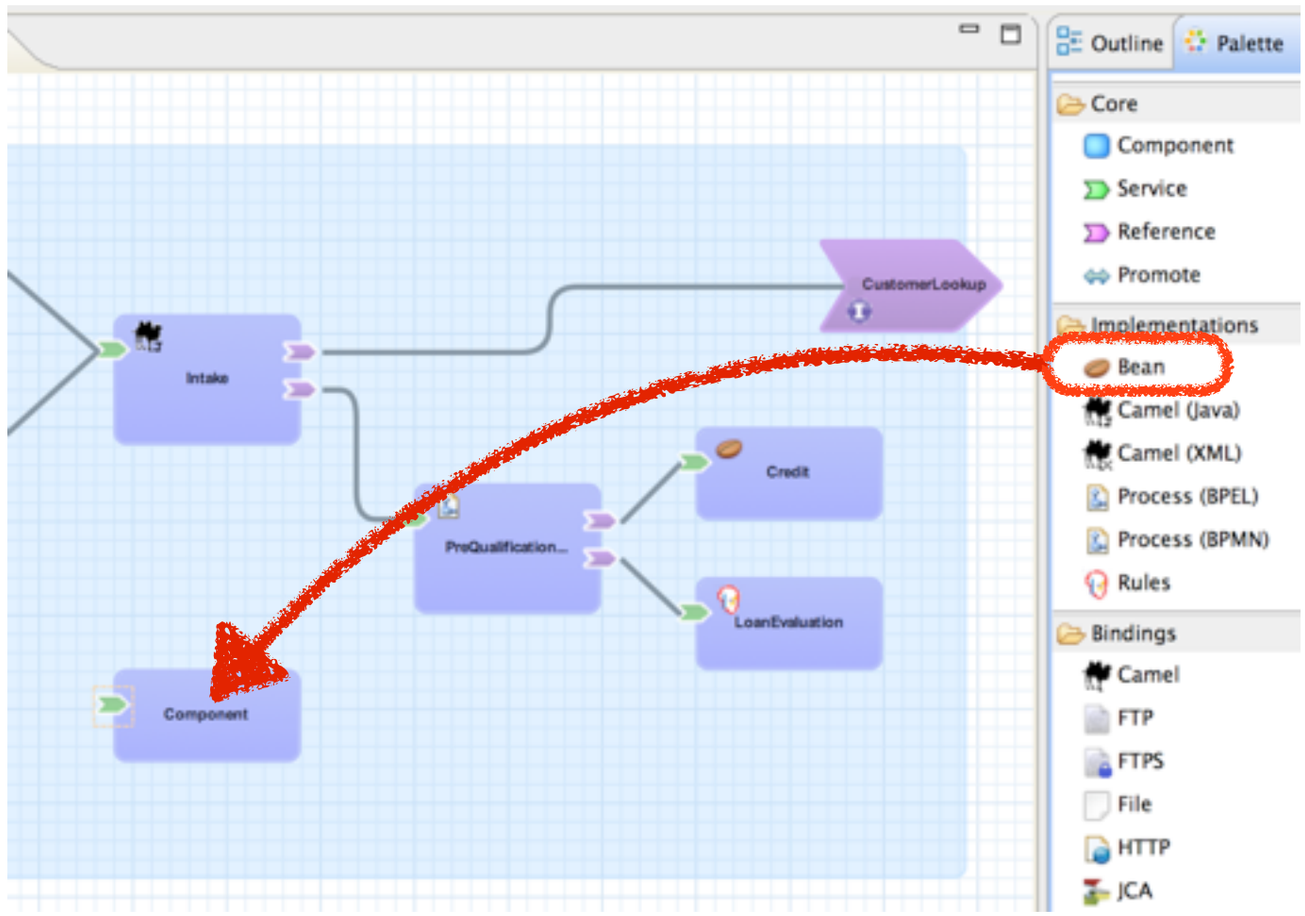
Step 7

StatusService Implementation

FYI

TODO

1. Click and hold on the Bean implementation in the palette and drag it on top of the new Component.



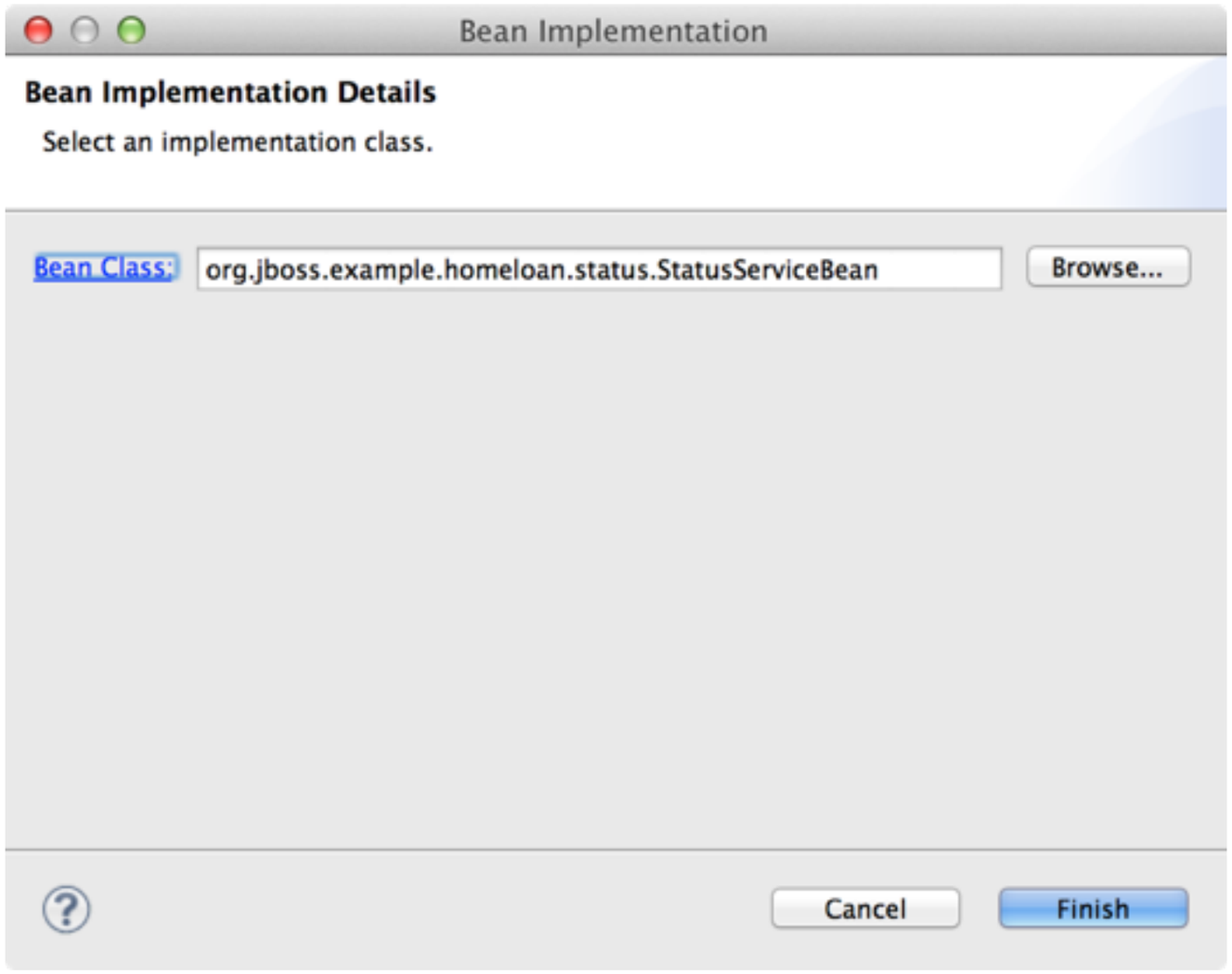
Step 7

StatusService Implementation

FYI

TODO

1. Click on the Browse ... button to select the bean implementation to use.
2. Select StatusServiceBean from the list.
3. Click Finish.



Step 7

Service Promotion

FYI

TODO

1. Hover over the green component service you just created.
2. Click on the promotion icon to promote the component service to a composite service.

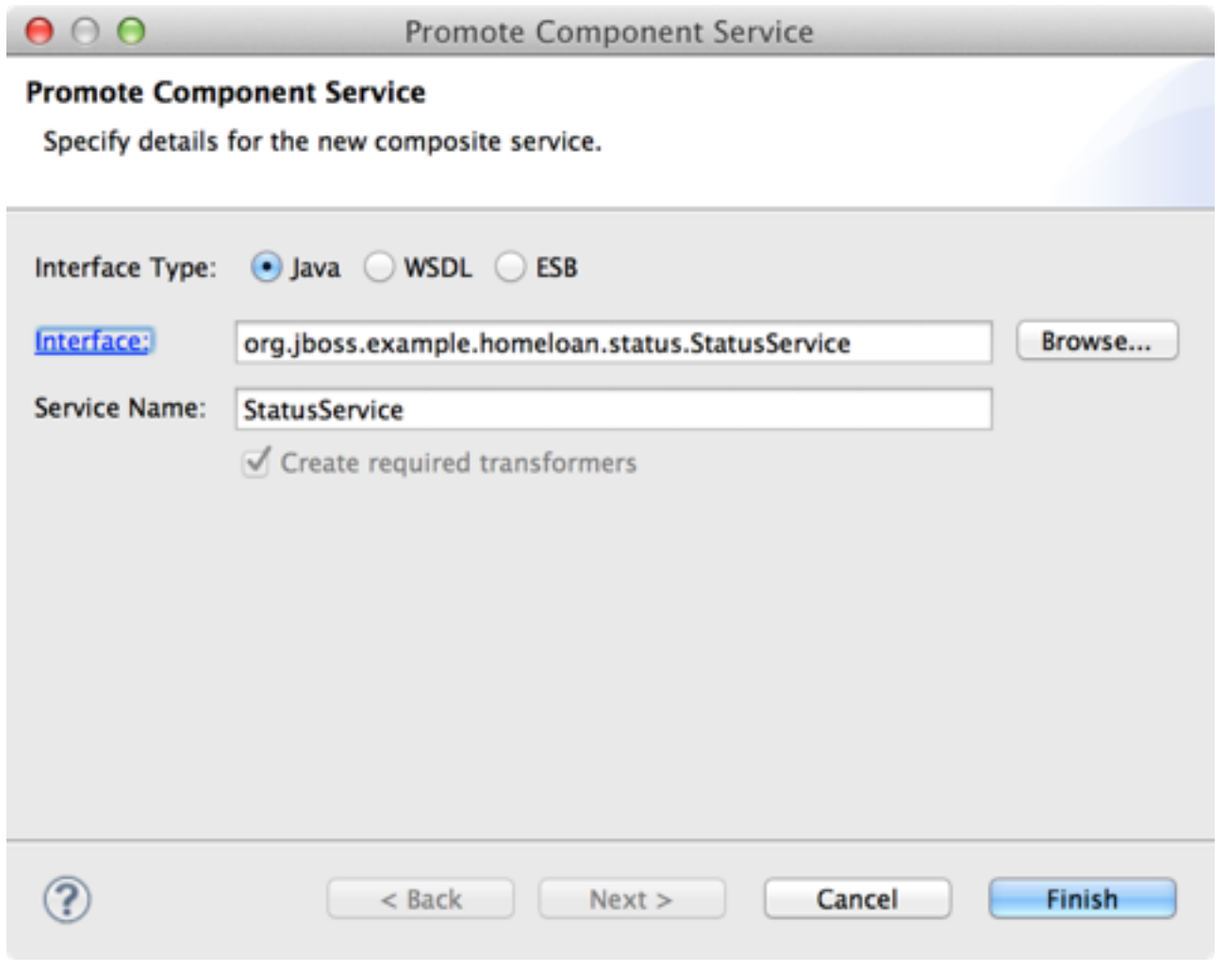
Step 7

Service Promotion

FYI

TODO

1. Click Finish.



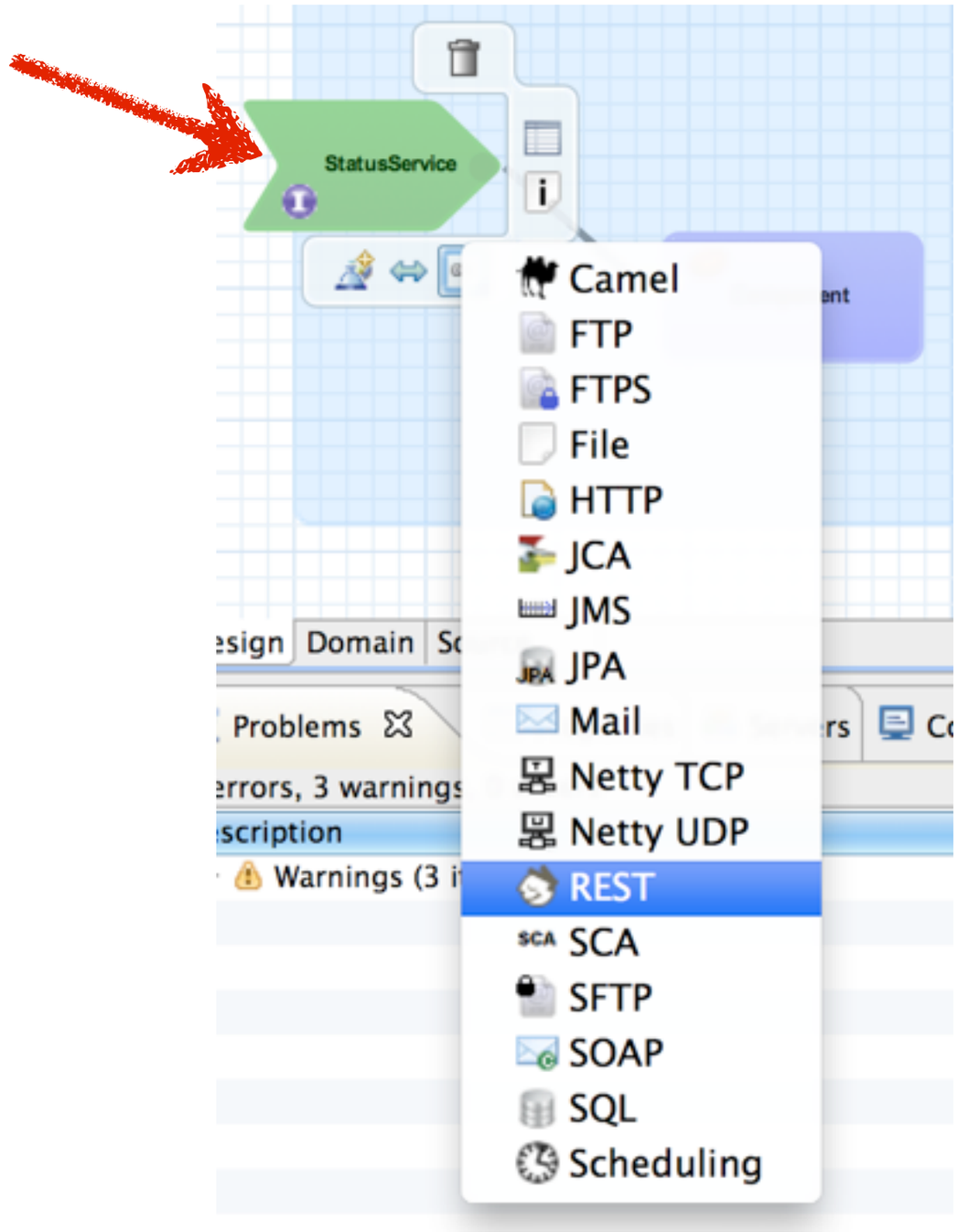
Step 7

Adding a REST Binding

FYI

TODO

1. Hover over the new composite service StatusService to bring up the button bar.
2. Click on the binding icon and select REST.



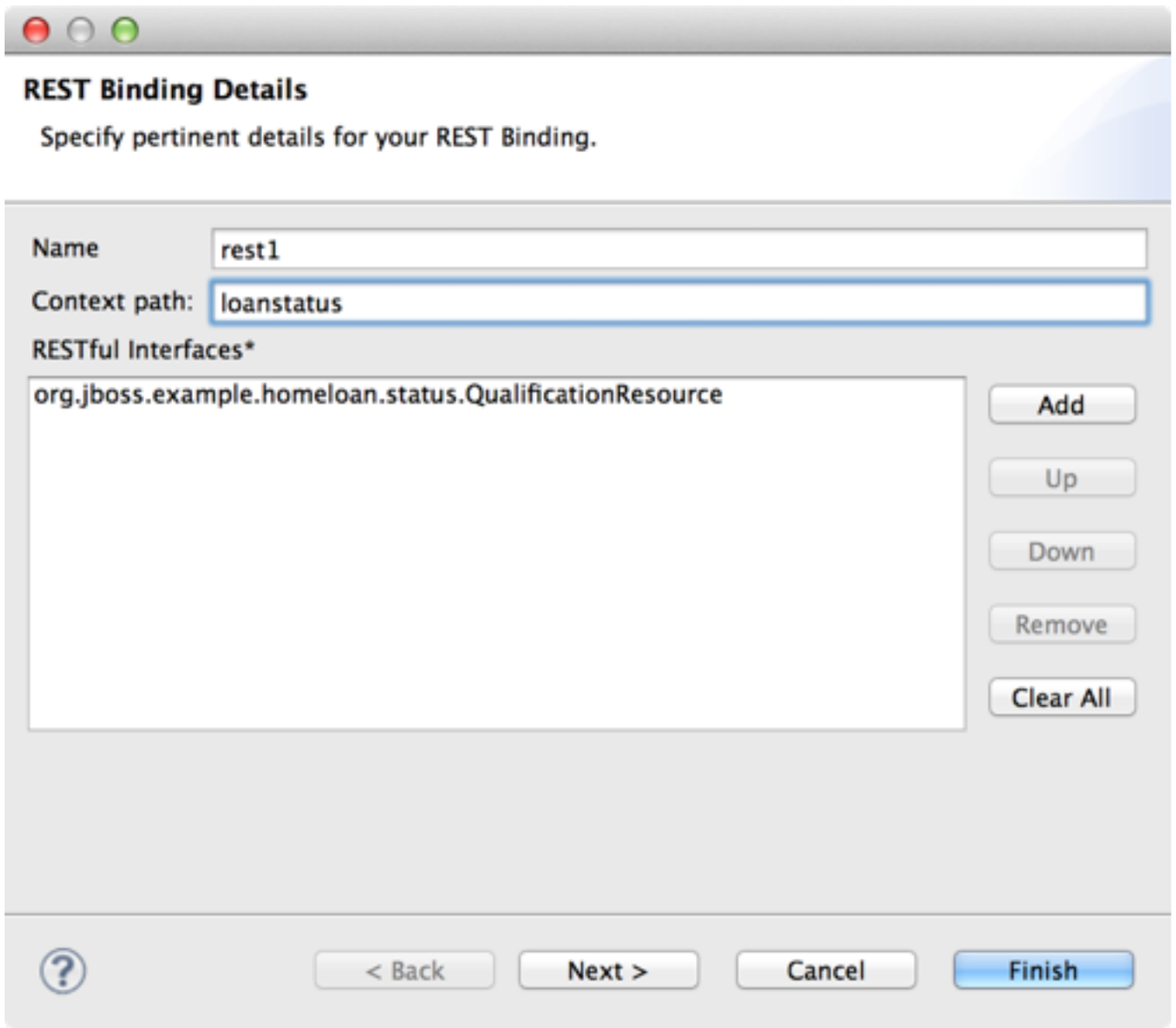
Step 7

Adding a REST Binding

FYI

TODO

1. `Context path : loanstatus`
2. Click Add to select the JAX-RS interface definition for the RESTful endpoint. Select QualificationResource in the resulting dialog.
3. Click Finish.



Step 7

Validate Changes

FYI

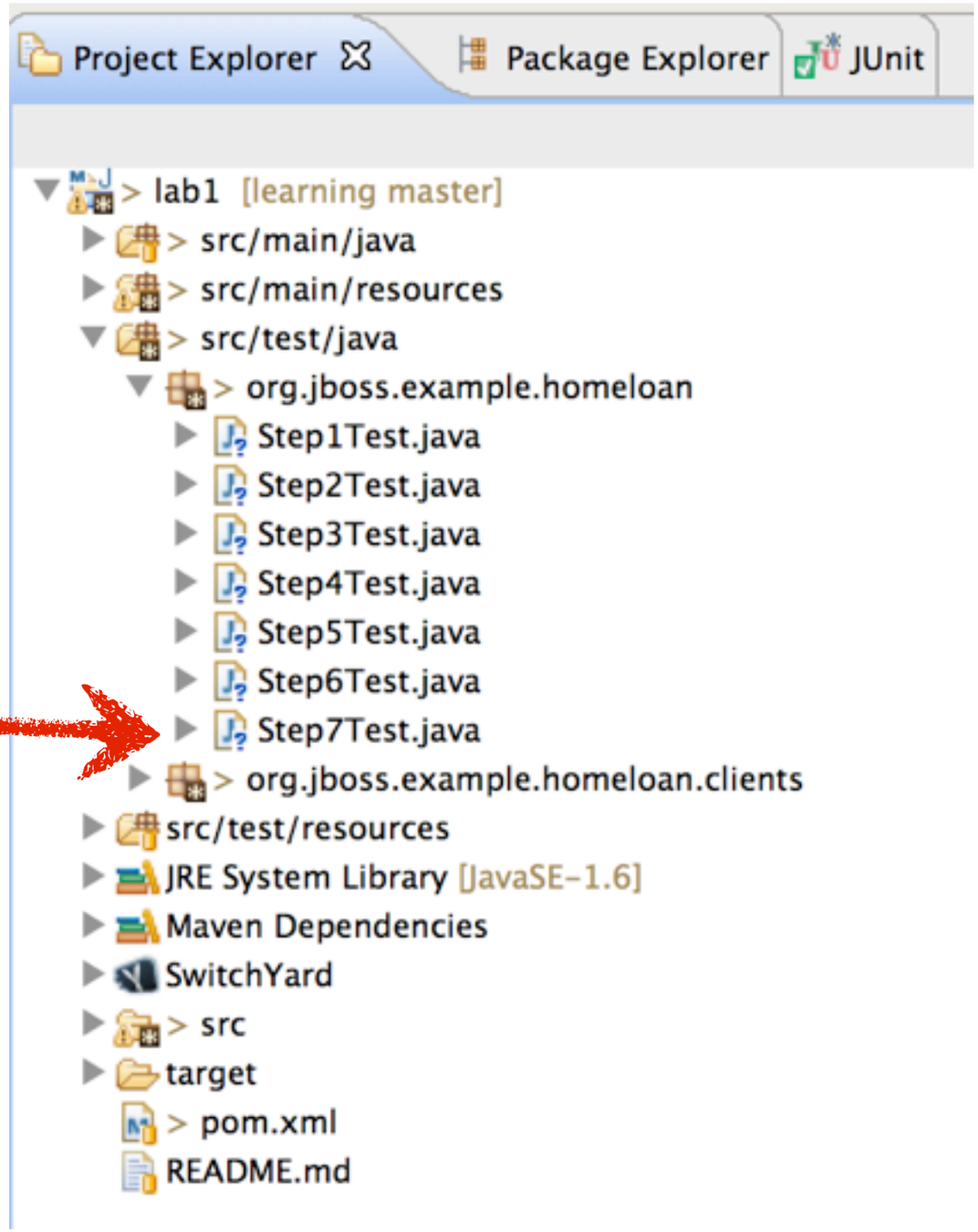
You have completed the changes required for step 7. Let's validate the changes using a service unit test.

TODO

1. Make sure the project is completely saved by selecting File -> Save All.

2. Double-click on Step7Test in the explorer to open the unit test.

3. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.

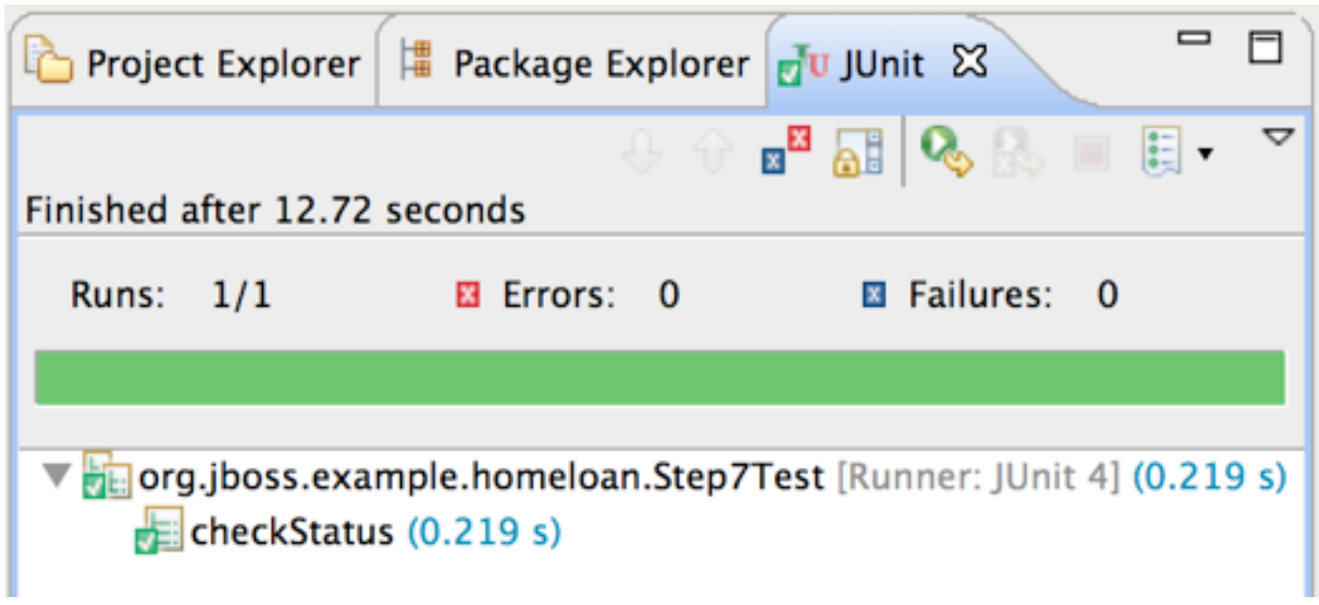


Step 7

Success?

FYI

You should see a nice green bar indicating that the test passed. If you get a red bar indicating test failure, raise your hand and someone will be by to help.



Lab 1 Complete!