Lab 3

Web Services, Camel Routing

Lab Goals

- This lab begins with an incomplete application and walks you through the steps to complete it.
- Each step has an associated test so you can validate your application behavior as you proceed
- Lab steps:
  - Complete a proxy route between two web services
  - Add conditional routing to the proxy route
  - Configure SOAP binding for a service

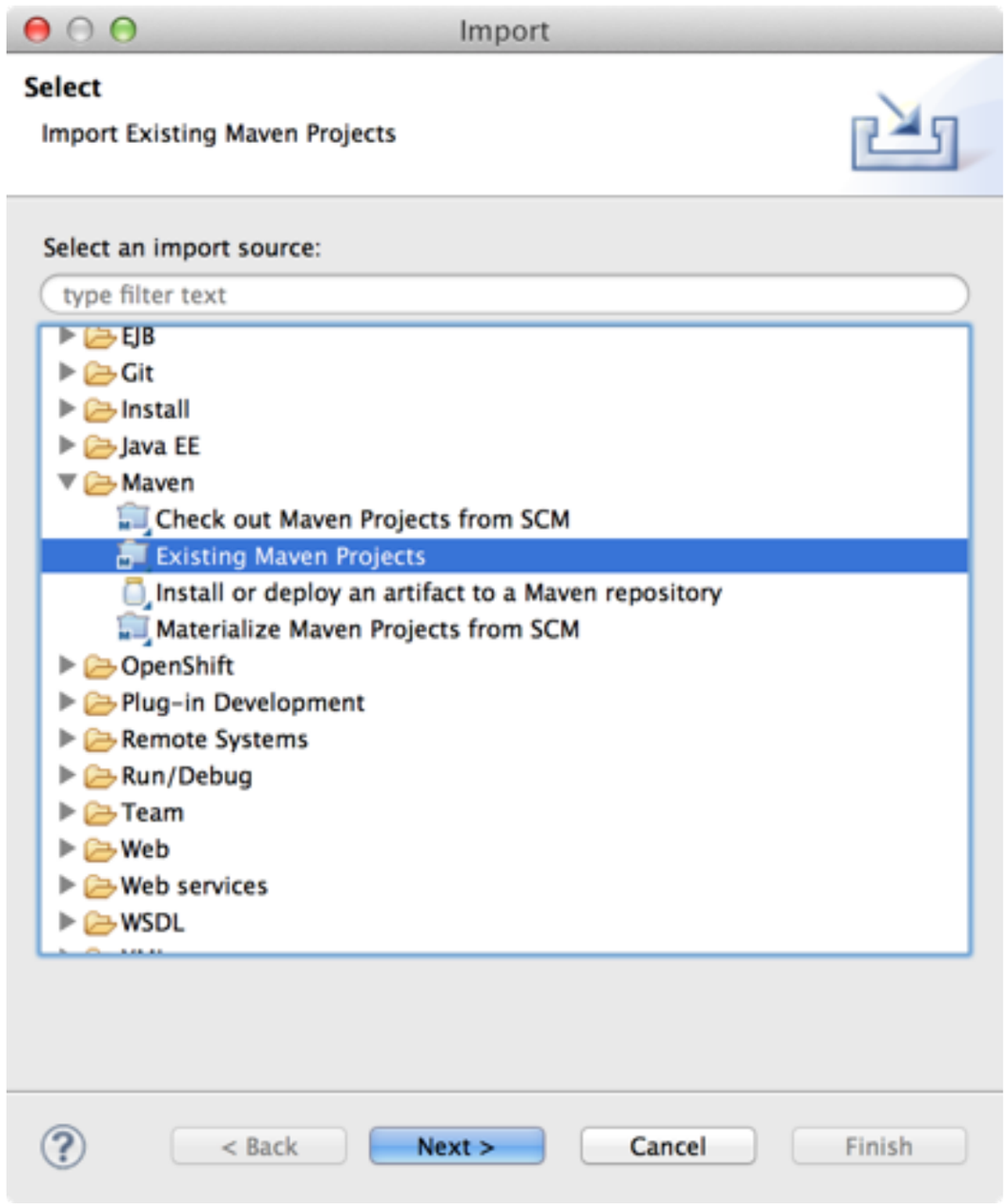
Importing Lab 3

TODO

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

2. Select Maven -> Existing Maven Projects

3. Click Next



Importing Lab 3

TODO

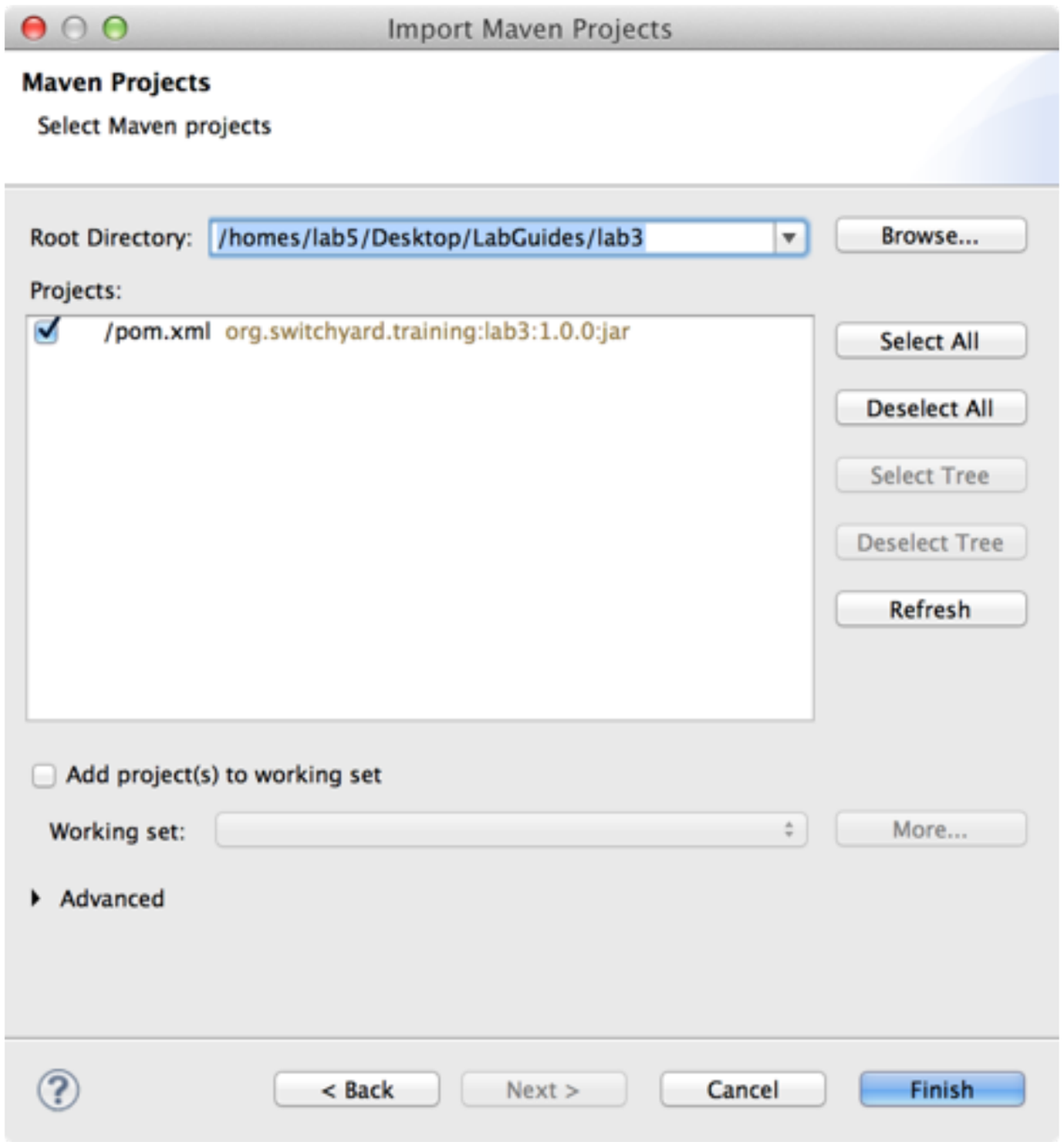
1. Click Browse ... and navigate to:

`/home/learning/summit2013/lab3`

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

`org.switchyard.training:lab3`

3. Click Finish



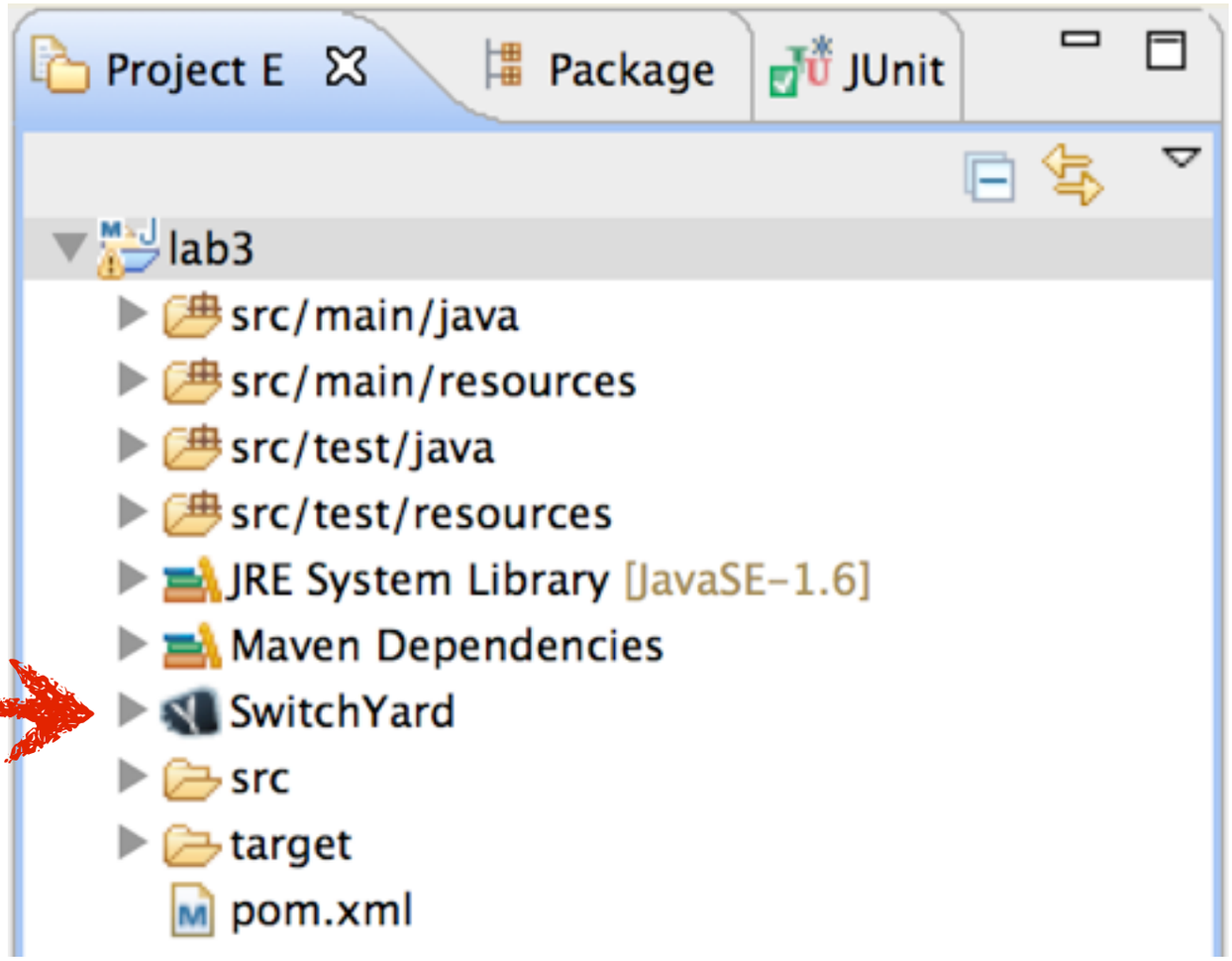
Lab 3

FYI

You will notice some red X's which represent validation errors. These are due to the fact that the application is incomplete at this stage. They will go away as you work through the lab.

TODO

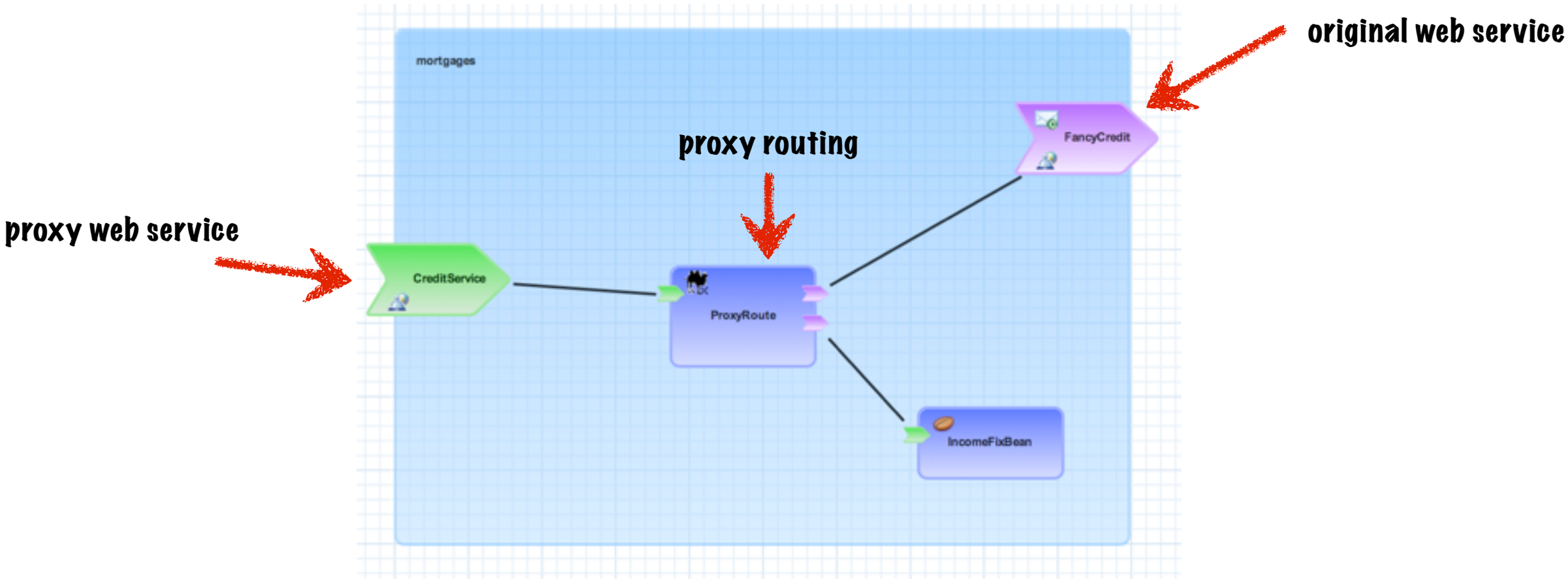
1. Double-click on the SwitchYard node in the explorer to open up the visual editor.



Lab 3 Application Model

FYI

The web service your organization used to assign credit scores for applicants has changed to require statement of income. Providing this information is currently optional for loan applicants, so it's necessary to sit a proxy in front of the changed web service which can deal with existing clients and still communicate with the changed credit web service. In lab 3, you will use a Camel route and the SOAP gateway to address this requirement.



Step 1

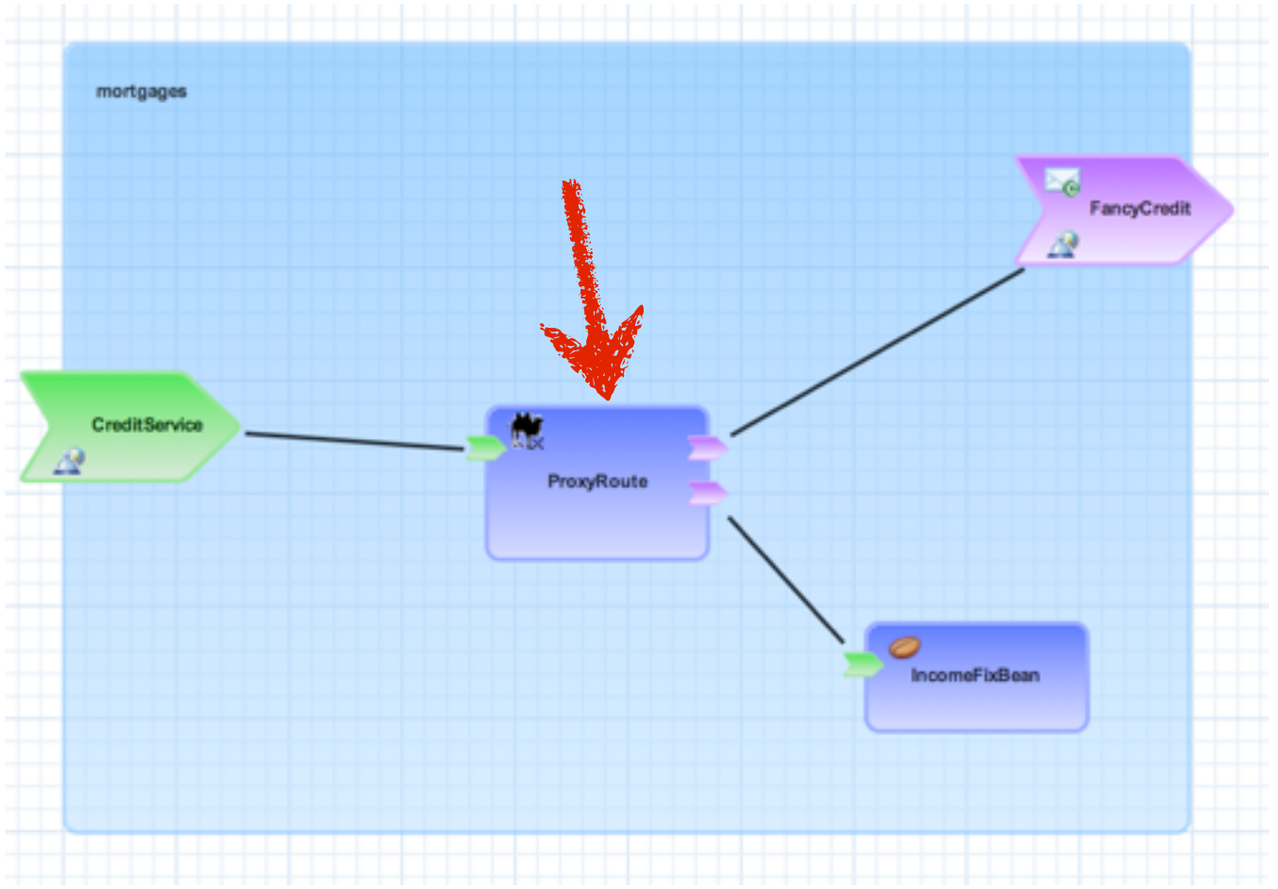
Connect Proxy Route

FYI

There is nothing unique about proxying in SOA-P 6. It can be implemented by defining a Camel route which routes from one SOAP binding to another.

TODO

1. Double-click on the ProxyRoute component in the editor to open up the Camel route in the editor.



Step 1

Update Camel Route

FYI

Note the use of "switchyard://" endpoints in the route definition. These are logical endpoints which map to services and references in your application, abstracting away the binding details from your routing logic. In this step, we are routing from our proxy service to the existing credit service.

TODO

1. Add the following line after the log statement in route.xml :

```
<to uri="switchyard://FancyCredit"/>
```

2. File - > Save

route.xml

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <route xmlns="http://camel.apache.org/schema/spring">
3    <from uri="switchyard://CreditService"/>
4    <log message="CreditService - message received: ${body}"/>
5    <to uri="switchyard://FancyCredit"/>
6  </route>
```

Step 1

Validate Changes

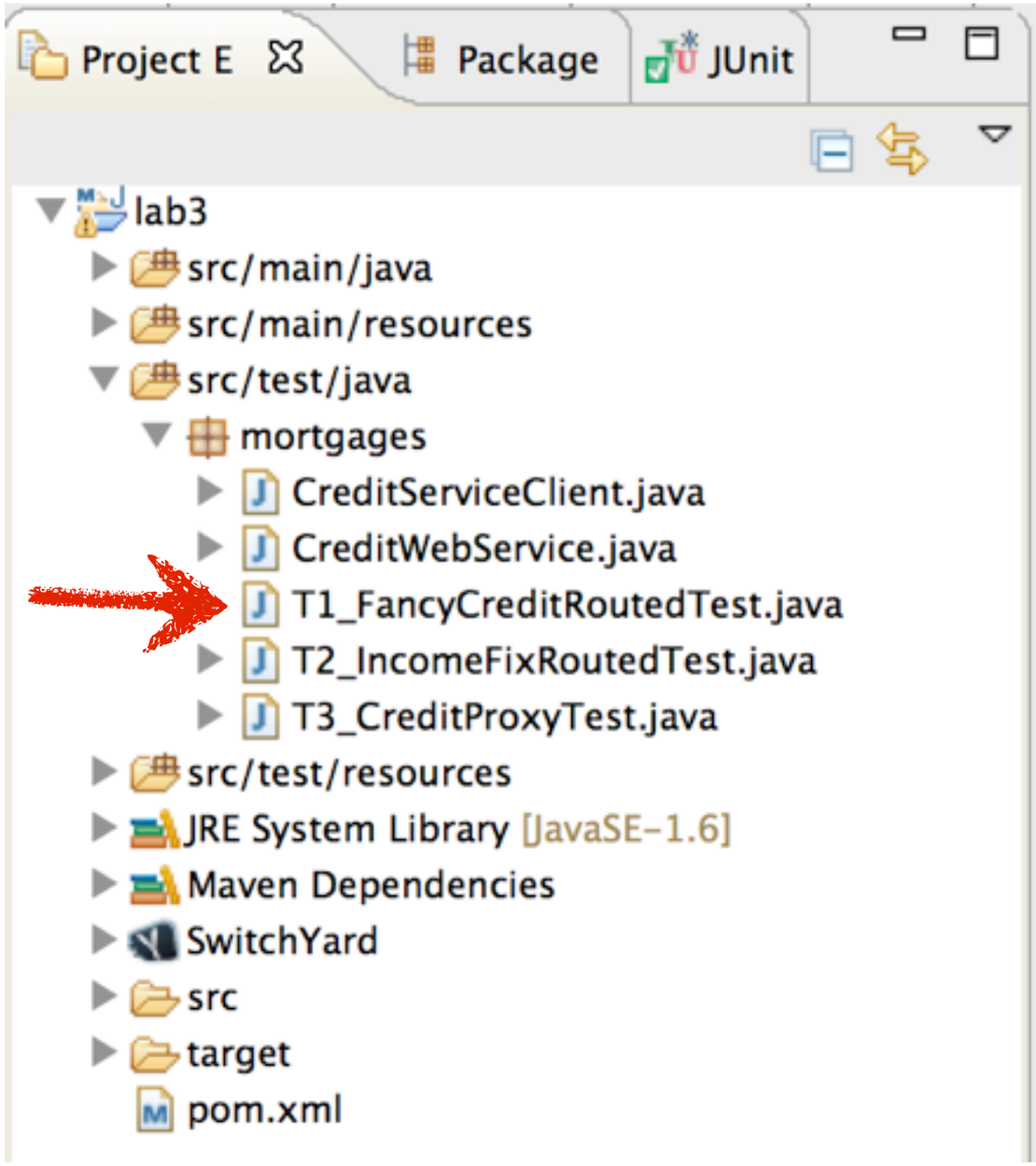
FYI

You have completed the changes required for step 1. Let's validate the changes using a service unit test.

TODO

1. Double-click on T1_FancyCreditRoutedTest in the explorer to open the unit test.

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

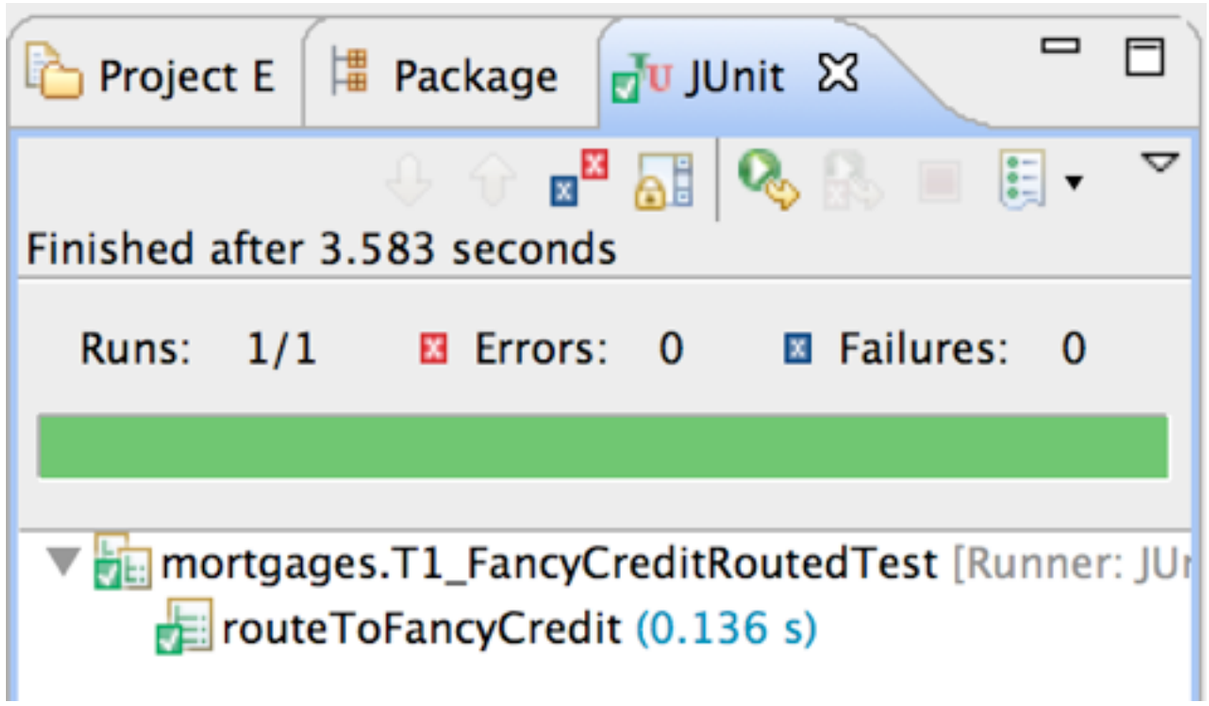


Step 1

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.

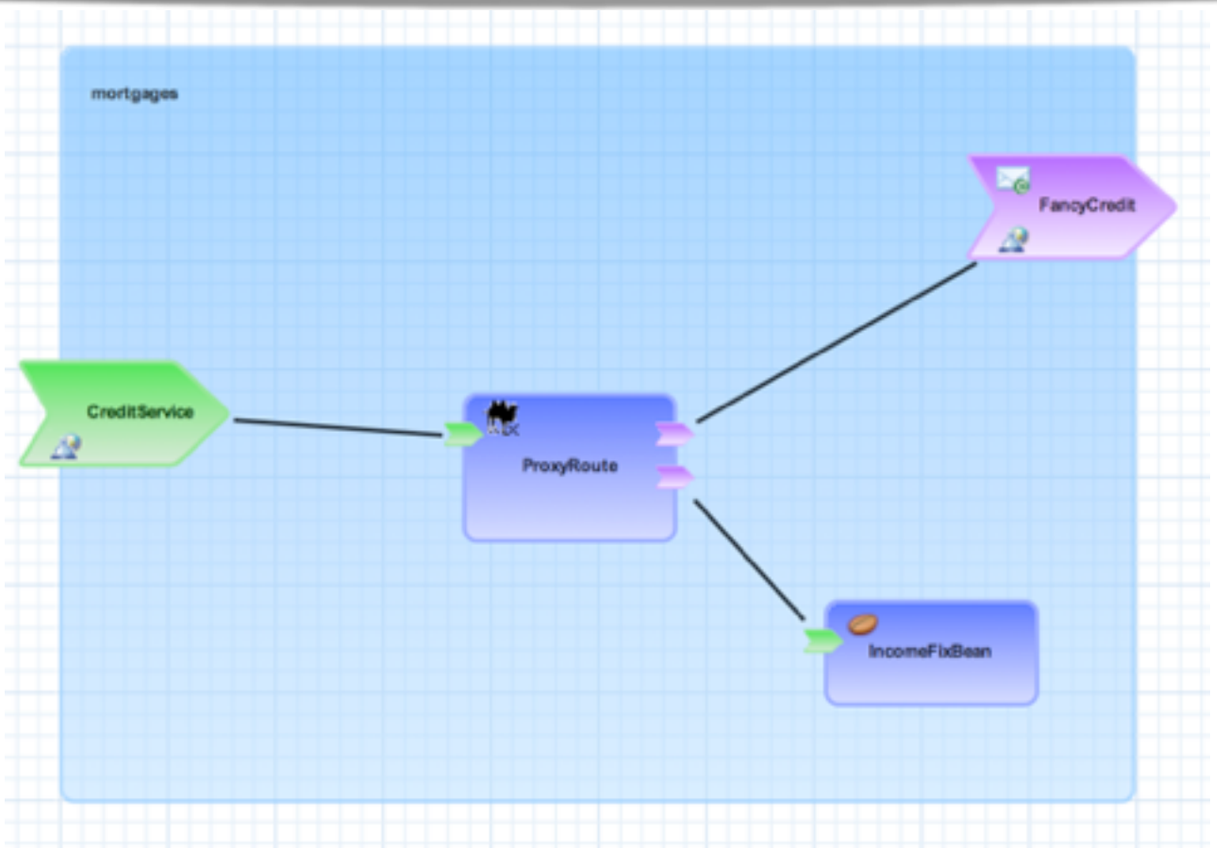


Step 2

The Power of the Proxy

FYI

In step 1, we connected our proxy web service with the existing credit service, but this doesn't solve the issue with applicants that don't have income defined. Luckily, we have an in-house service "IncomeFix" which invents an income value based on the required minimum for a mortgage. We need to route to this new service if an incoming request does not have an income value before sending the application to the credit service.

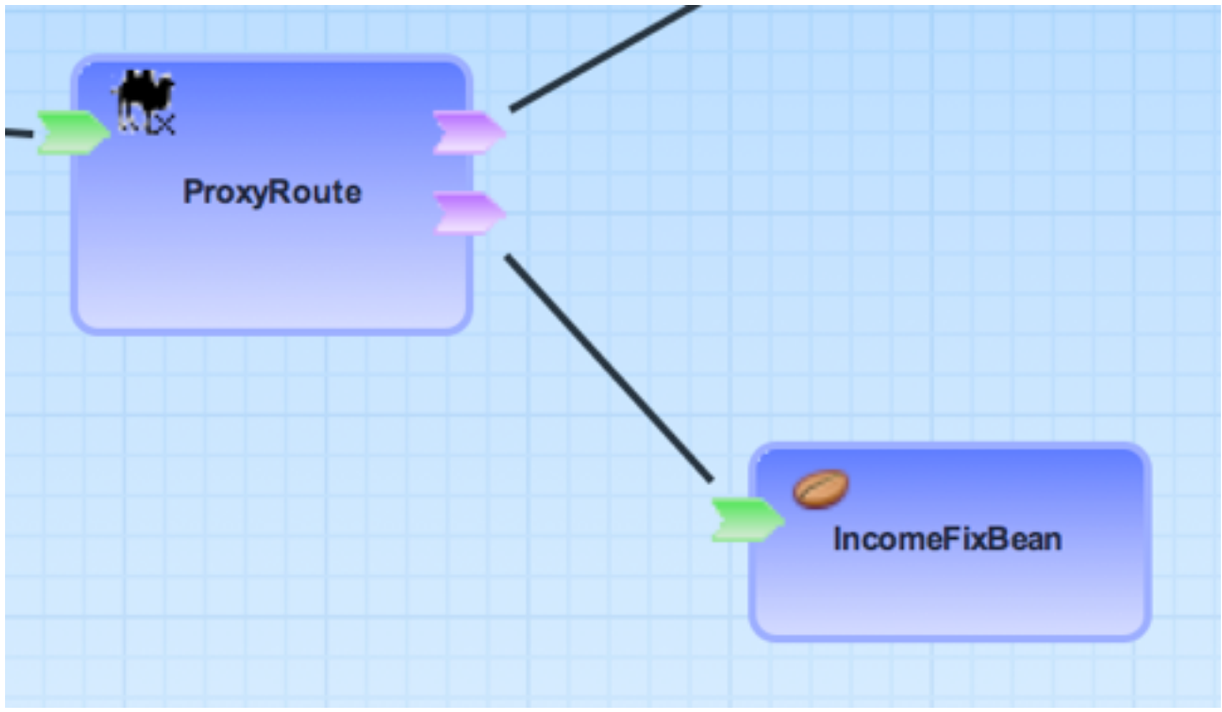


Step 2

Update Proxy Route to Call IncomeFix

FYI

Our application model already has a reference defined between the proxy route and the IncomeFix service. We just need to use that reference in our routing logic.



TODO

1. If you closed route.xml from step 1, double-click on the ProxyRoute component in the editor to open up the Camel route in the editor.

Step 2

Connect Proxy Route

FYI

We can use a message filter in Camel to provide conditional routing in the case that an income value is not provided by an applicant.

TODO

1. Add the following filter definition above the line routing to FancyCredit (copy and paste recommended) :

```
<filter>
  <xpath>count(//*[local-name() = 'income']) = 0</xpath>
  <to uri="switchyard://IncomeFix"/>
</filter>
```

2. File -> Save All

route.xml

```
1  <?xml version="1.0" encoding="ASCII"?>
2  <route xmlns="http://camel.apache.org/schema/spring">
3    <from uri="switchyard://CreditService"/>
4    <log message="CreditService - message received: ${body}"/>
5    <filter>
6      <xpath>count(//*[local-name() = 'income']) = 0</xpath>
7      <to uri="switchyard://IncomeFix"/>
8    </filter>
9    <to uri="switchyard://FancyCredit"/>
10 </route>
```

Step 2
Validate Changes

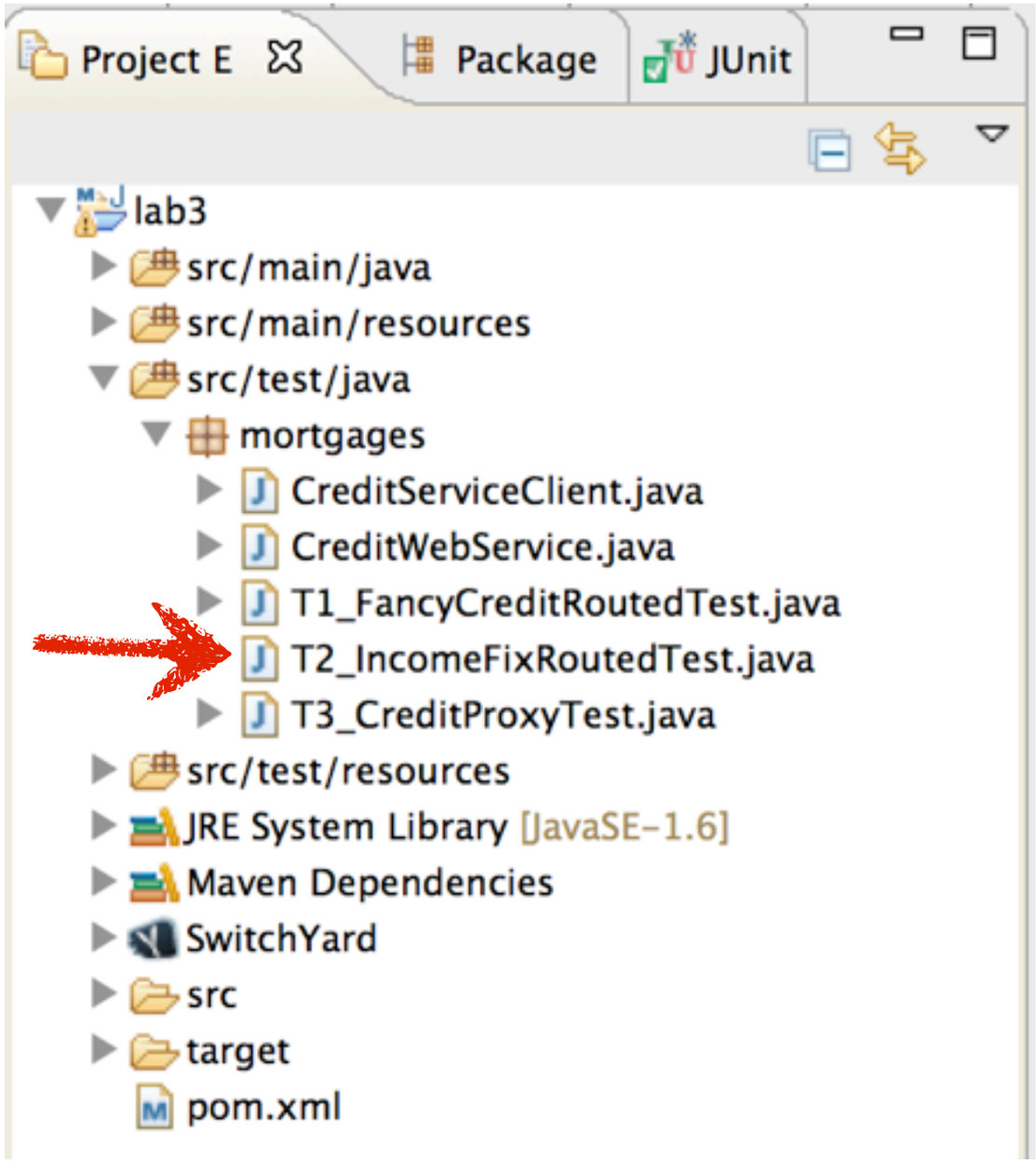
FYI

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

TODO

1. Double-click on T2_IncomeFixRoutedTest in the explorer to open the unit test.

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

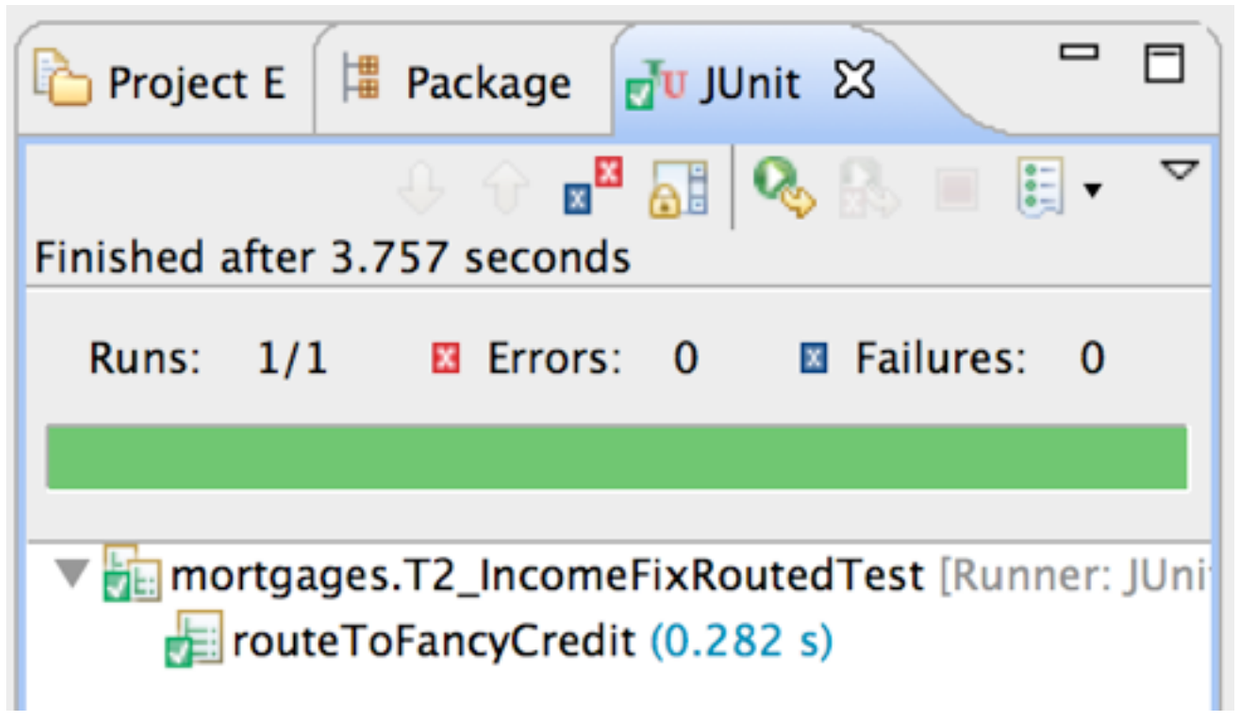


Step 2

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.

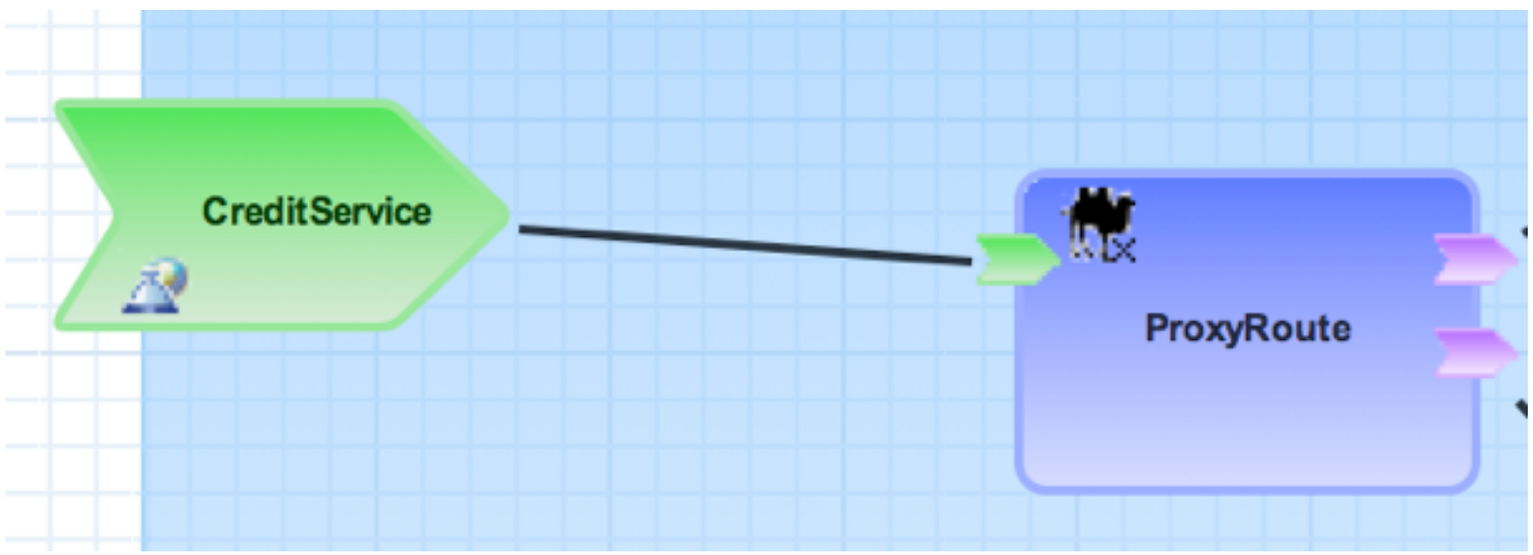


Step 3

Adding SOAP Service Binding

FYI

We need to add a SOAP binding to the composite service CreditService so that requests can be routed through our proxy logic and into the original service FancyCredit.



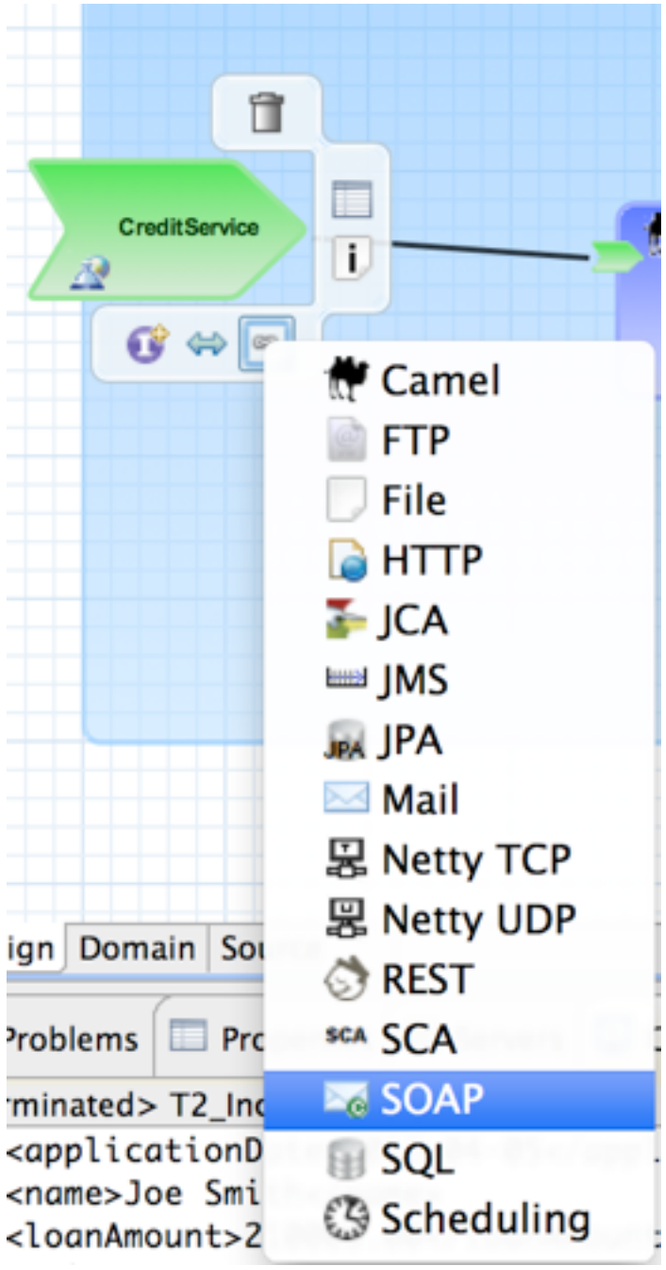
Step 3

Add SOAP Service Binding

TODO

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

2. Click on the bindings button and select SOAP.



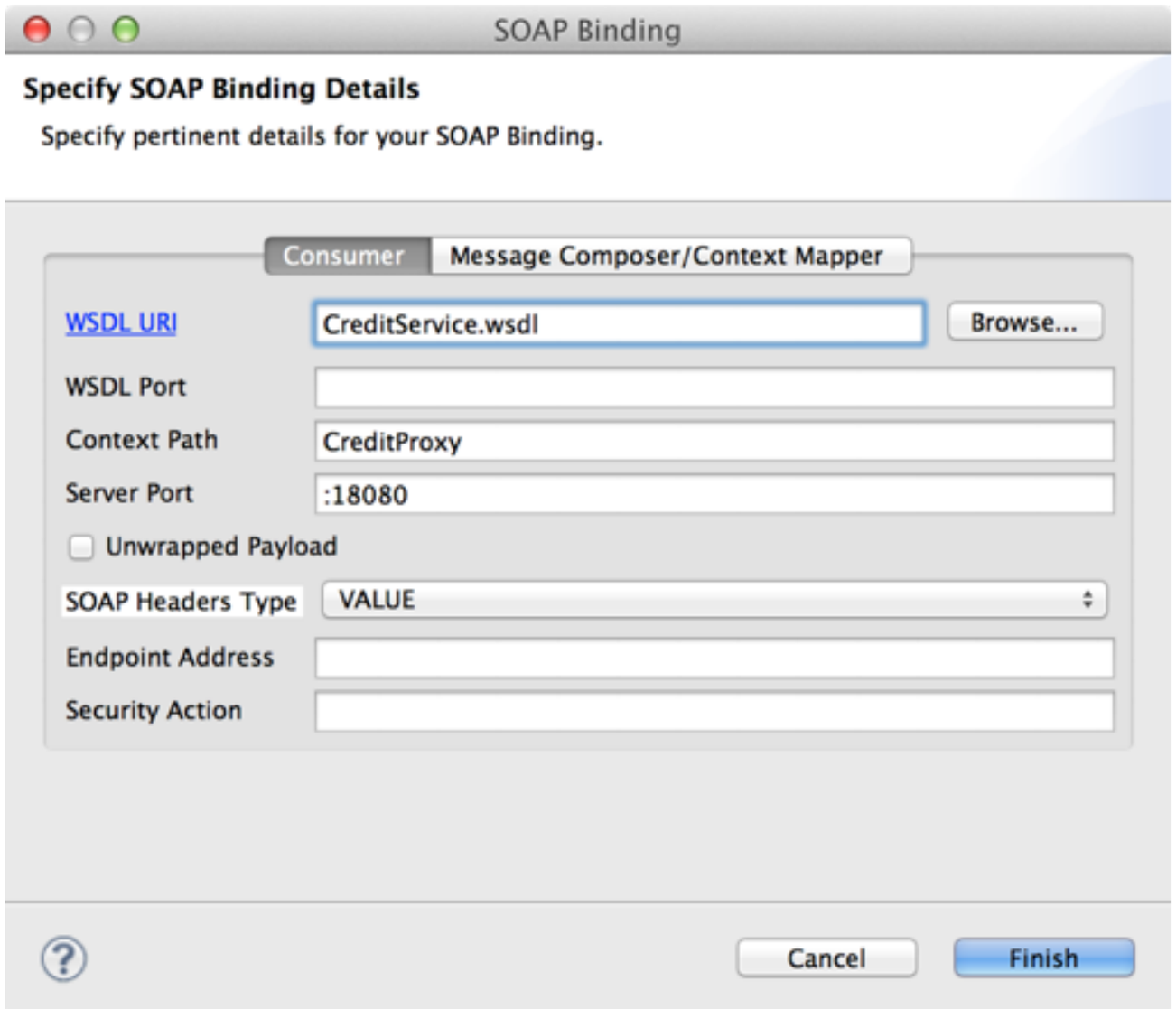
Step 3

Configure SOAP Binding

TODO

1. ContextPath : “CreditProxy”
   Server Port : “:18080”

2. Click Finish.



Step 3

Validate Changes

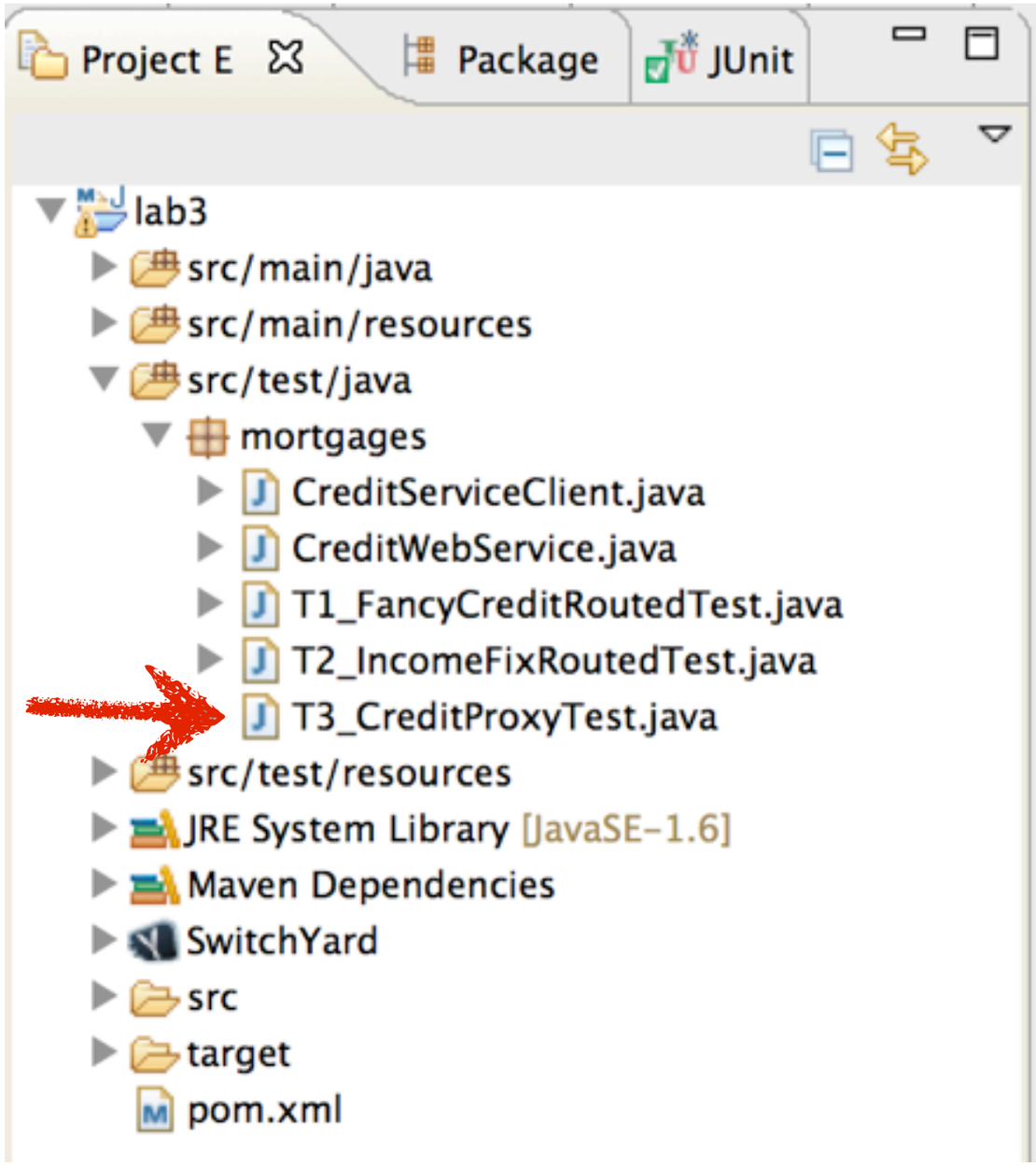
FYI

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

TODO

1. Double-click on T3_CreditProxyTest in the explorer to open the unit test.

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

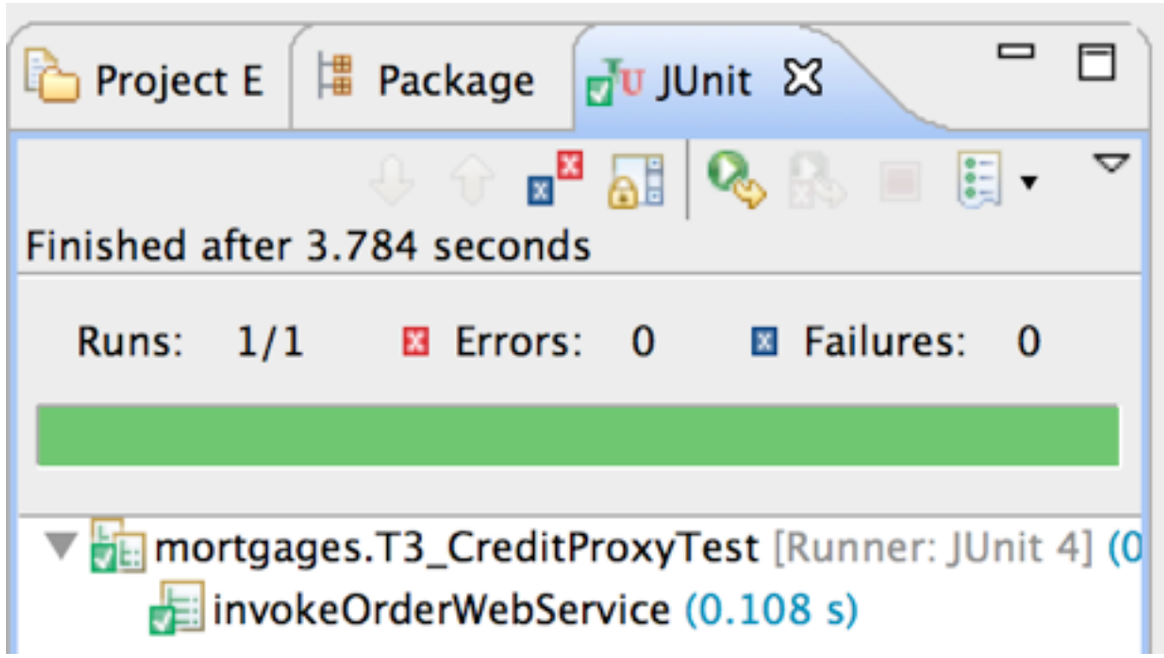


Step 3

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

- This is an optional step if you want to see lab3 deployed to the server.
- Step 4 includes:
  - Deploying original credit service to SwitchYard
  - Deploy application
  - Test application with SOAP client application

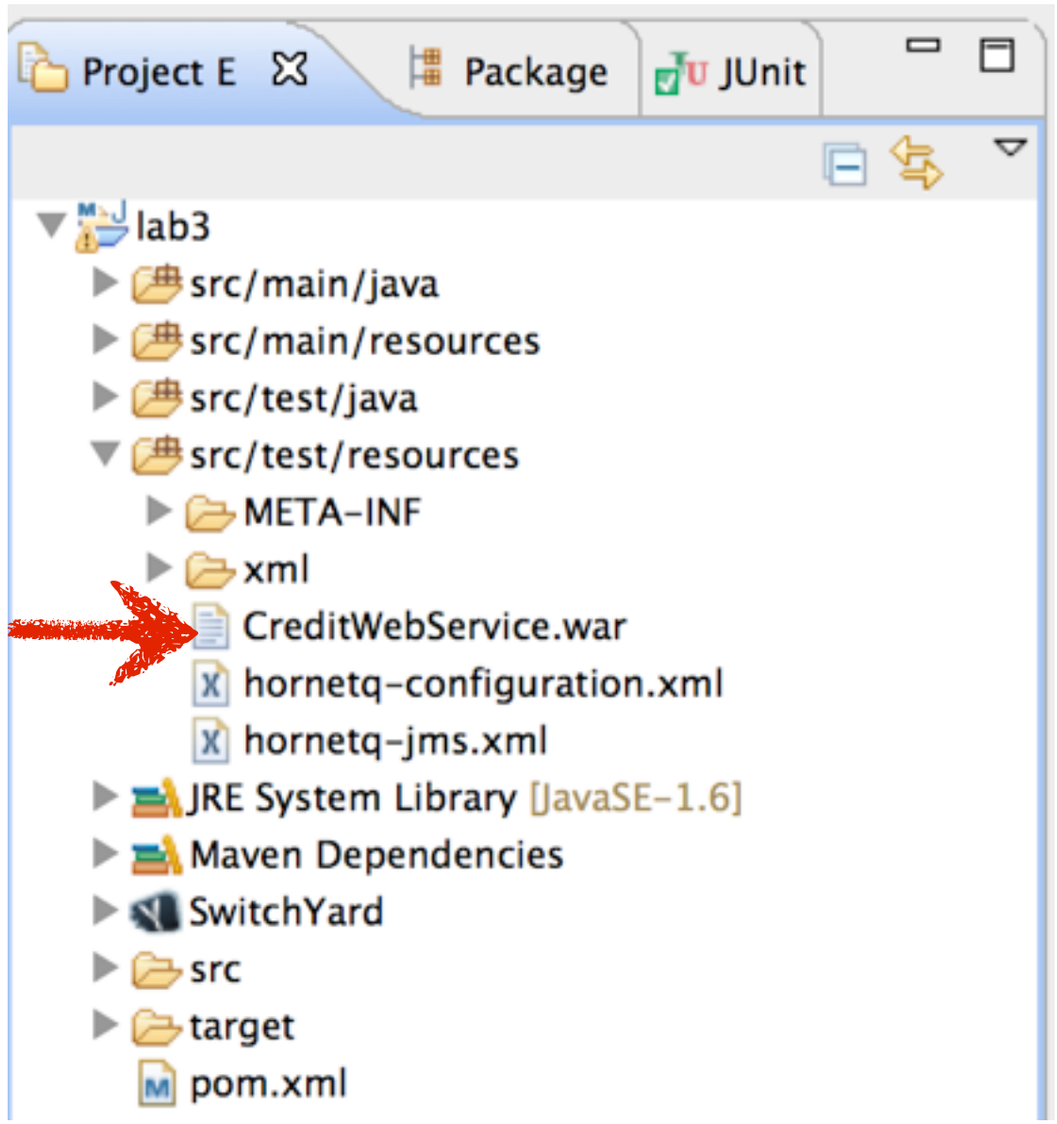
Step 4

Deploy Original CreditService

TODO

1. Right-click on CreditWebService.war and select ‘Mark as Deployable’.

2. When prompted with “Really mark these as deployable?”, click OK.

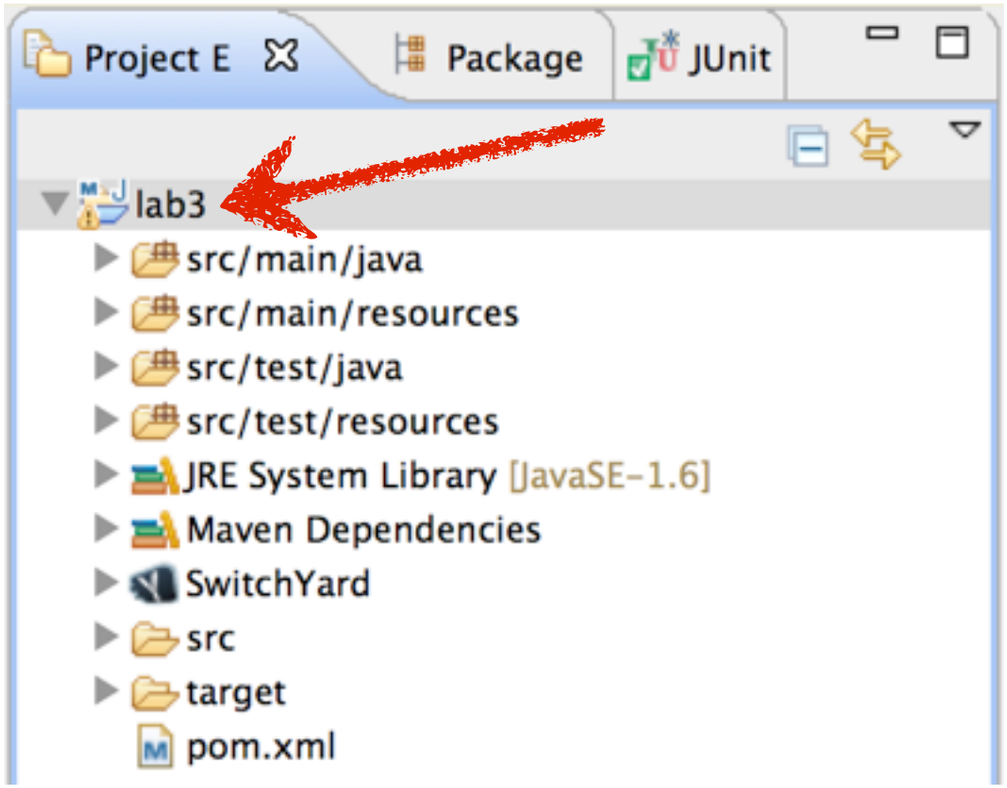


Step 4

Deploy Application

TODO

1. Right-click on the lab3 project and select Run As ... Run on Server

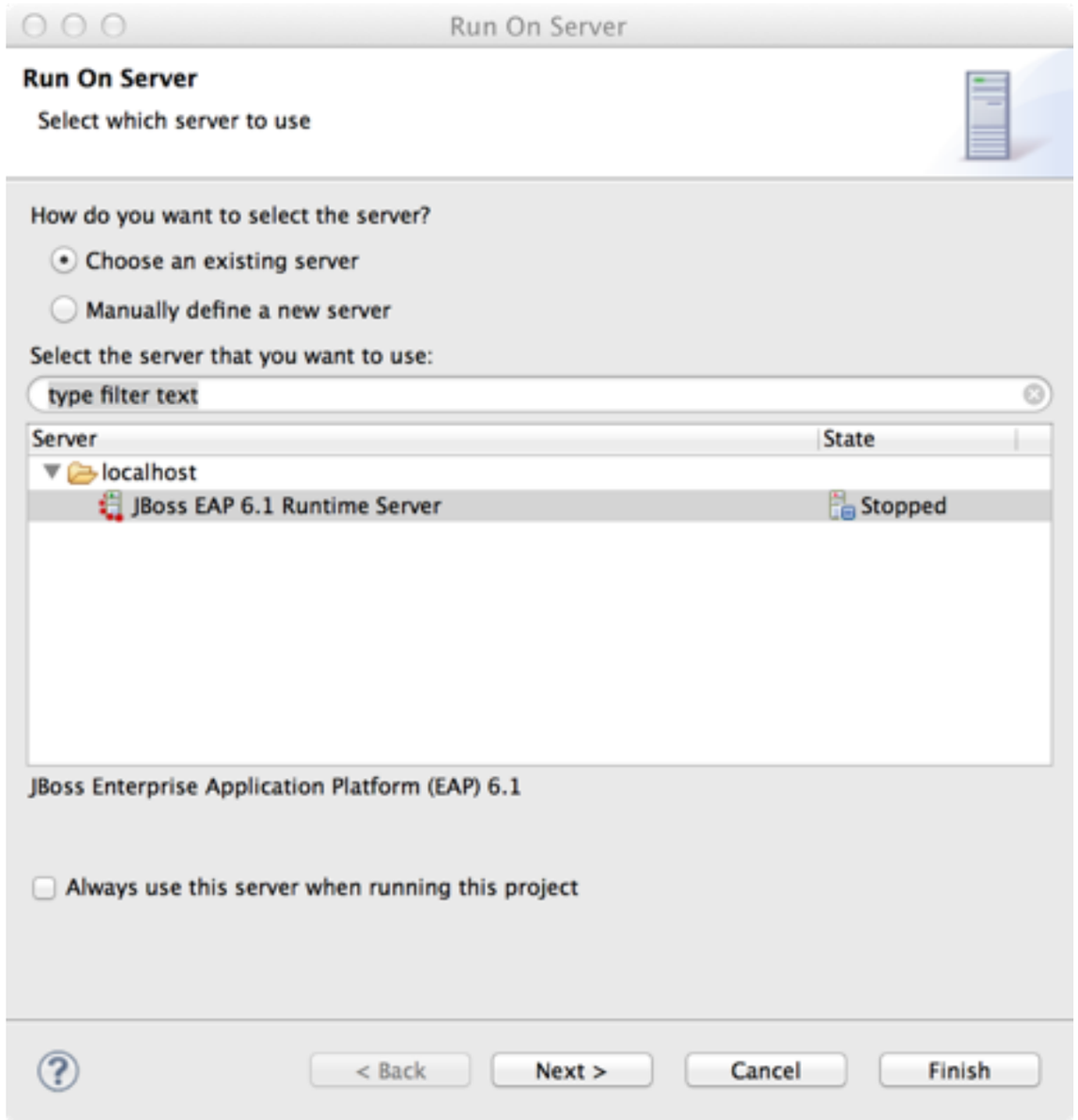


Select Server

TODO

1. Select the JBoss EAP 6.1 Runtime Server

2. Click Finish



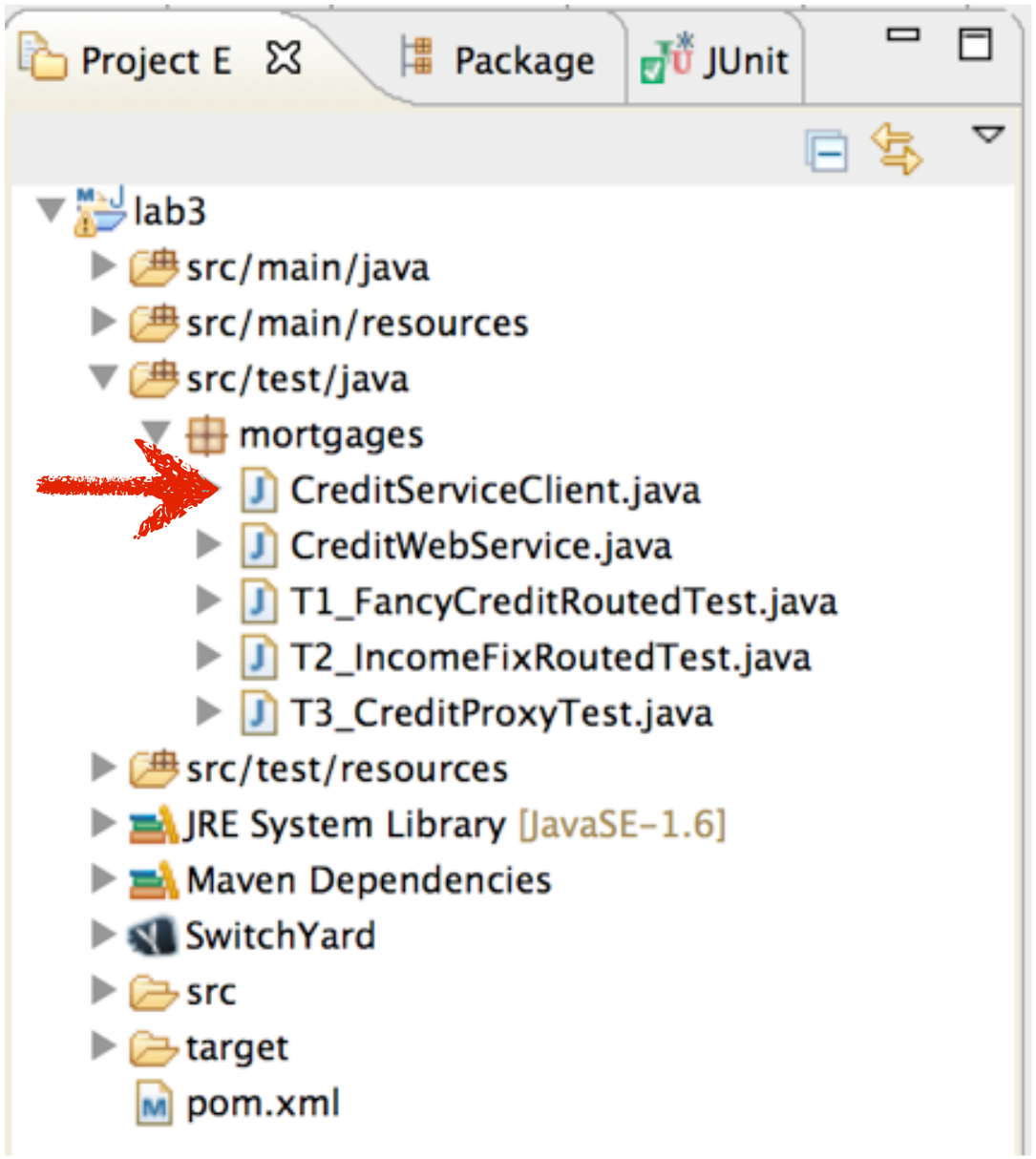
Step 4

Run Test Client

TODO

1. Open CreditServiceClient.java from the Project Explorer view.

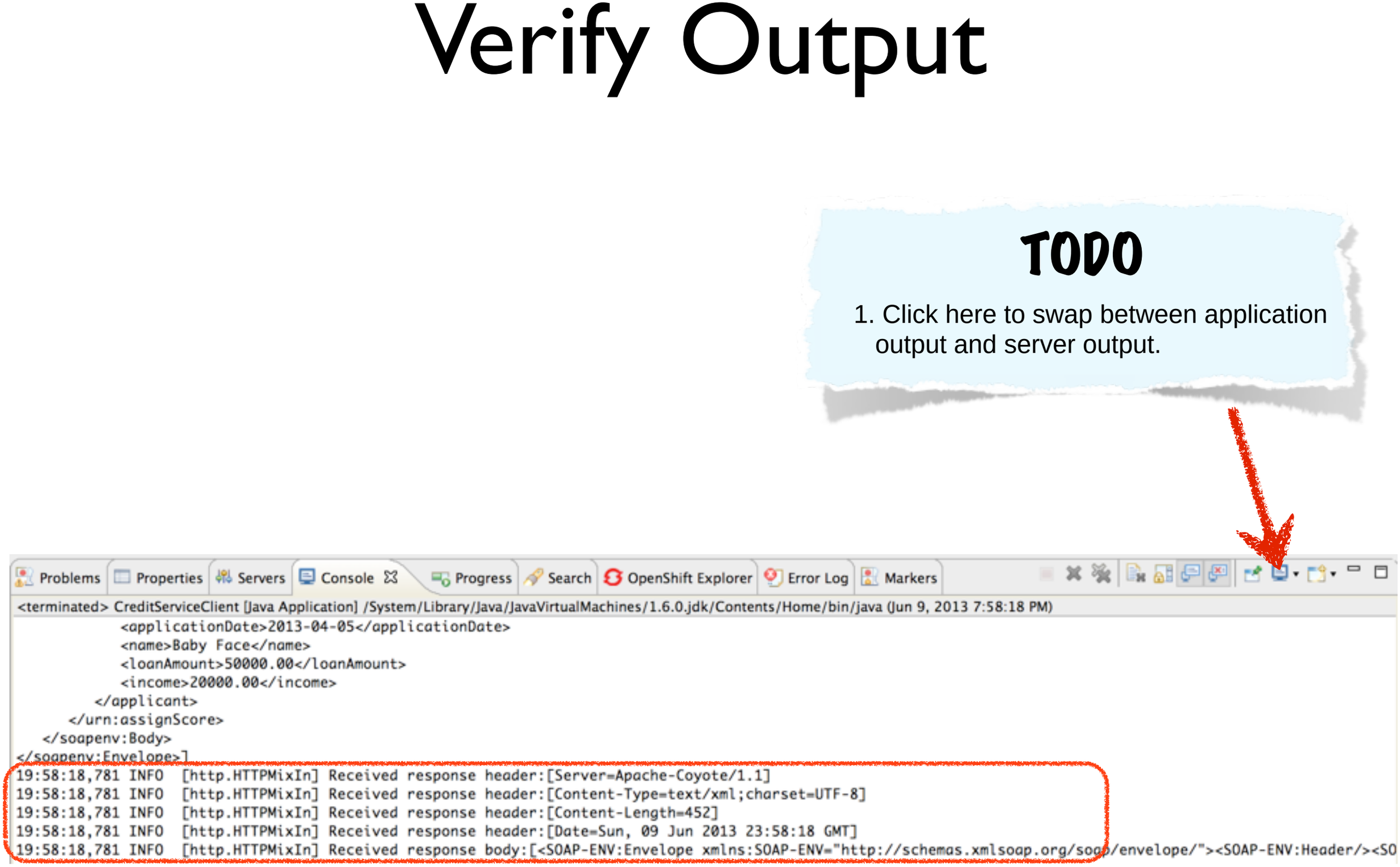
2. Go to the Run menu in the main menu bar and select ‘Run As -> Java Application’



Verify Output

TODO

1. Click here to swap between application output and server output.



Lab 3 Complete!