Lab 2

Bean Services, Transformation, and JMS

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:
  - Use a reference to invoke another service
  - Define transformations to/from XML and JSON
  - Add JMS bindings to services and references

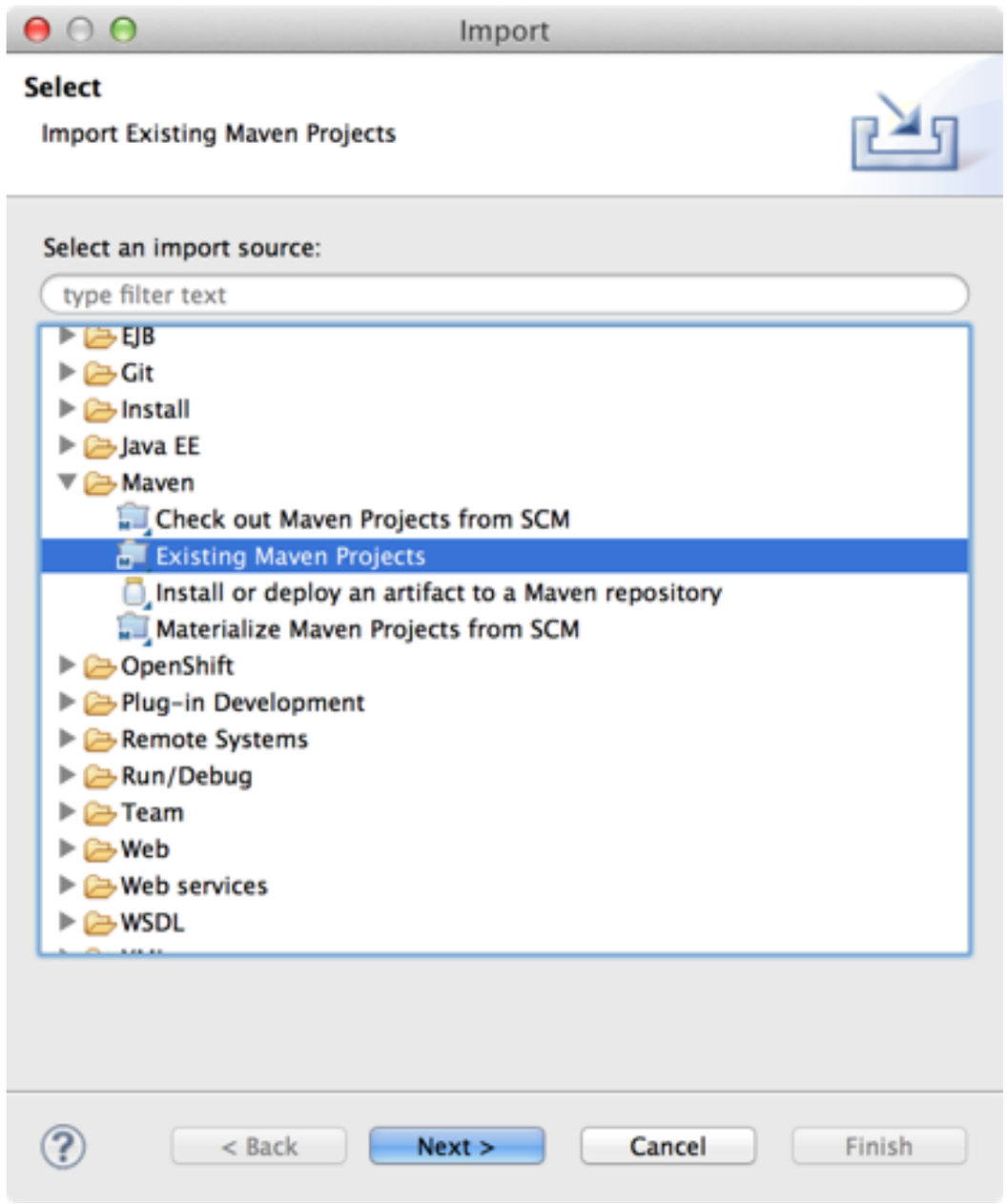
Importing Lab 2

TODO

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

2. Select Maven -> Existing Maven Projects

3. Click Next



Importing Lab 2

TODO

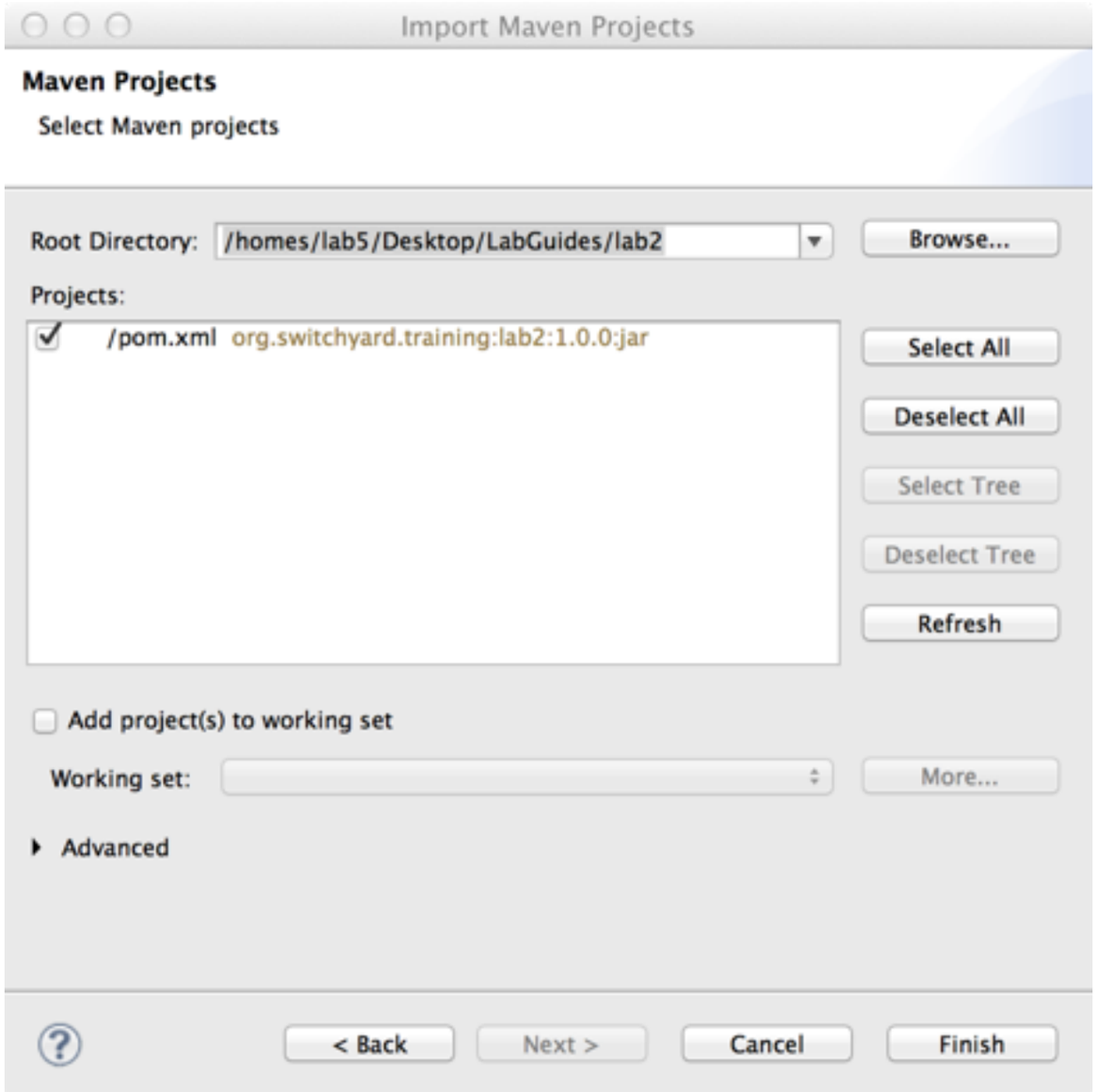
1. Click Browse ... and navigate to:

`/home/learning/summit2013/lab2`

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

`org.switchyard.training:lab2`

3. Click Finish



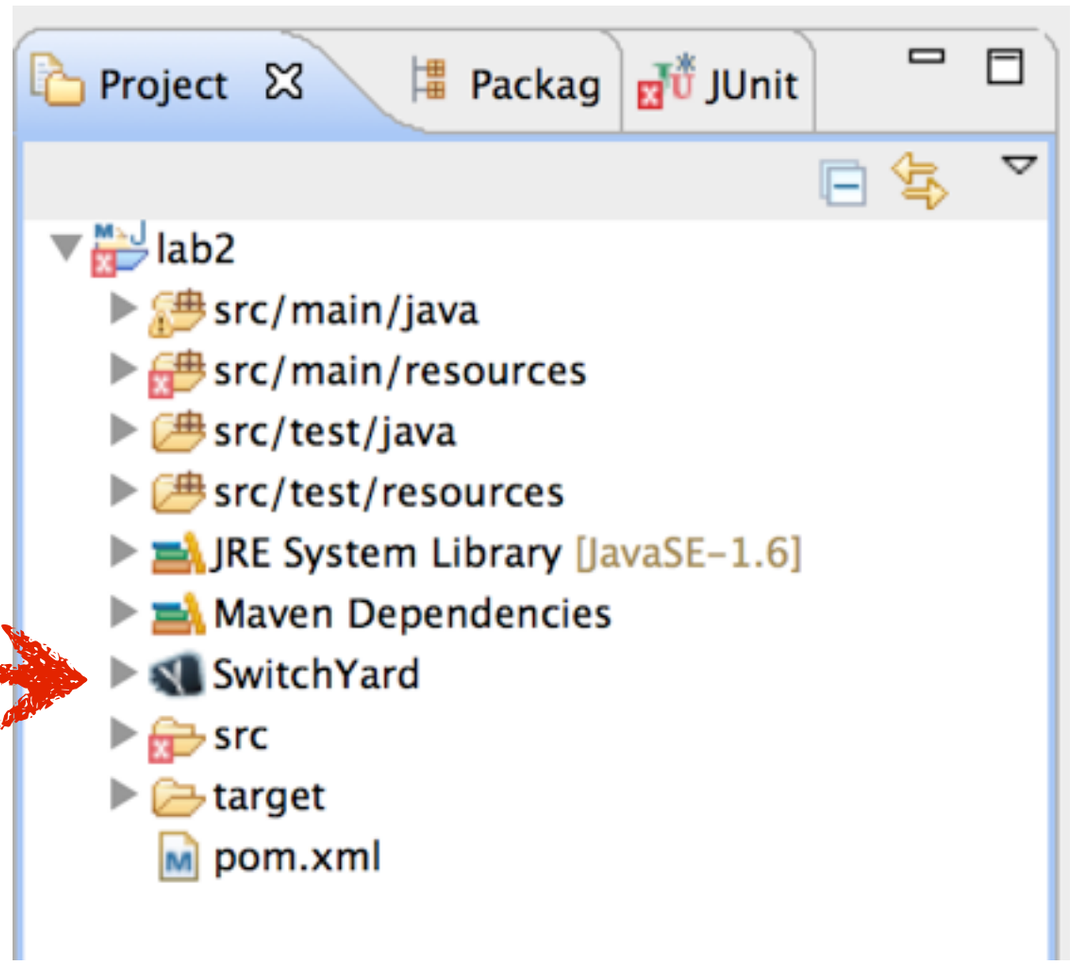
Lab 2

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

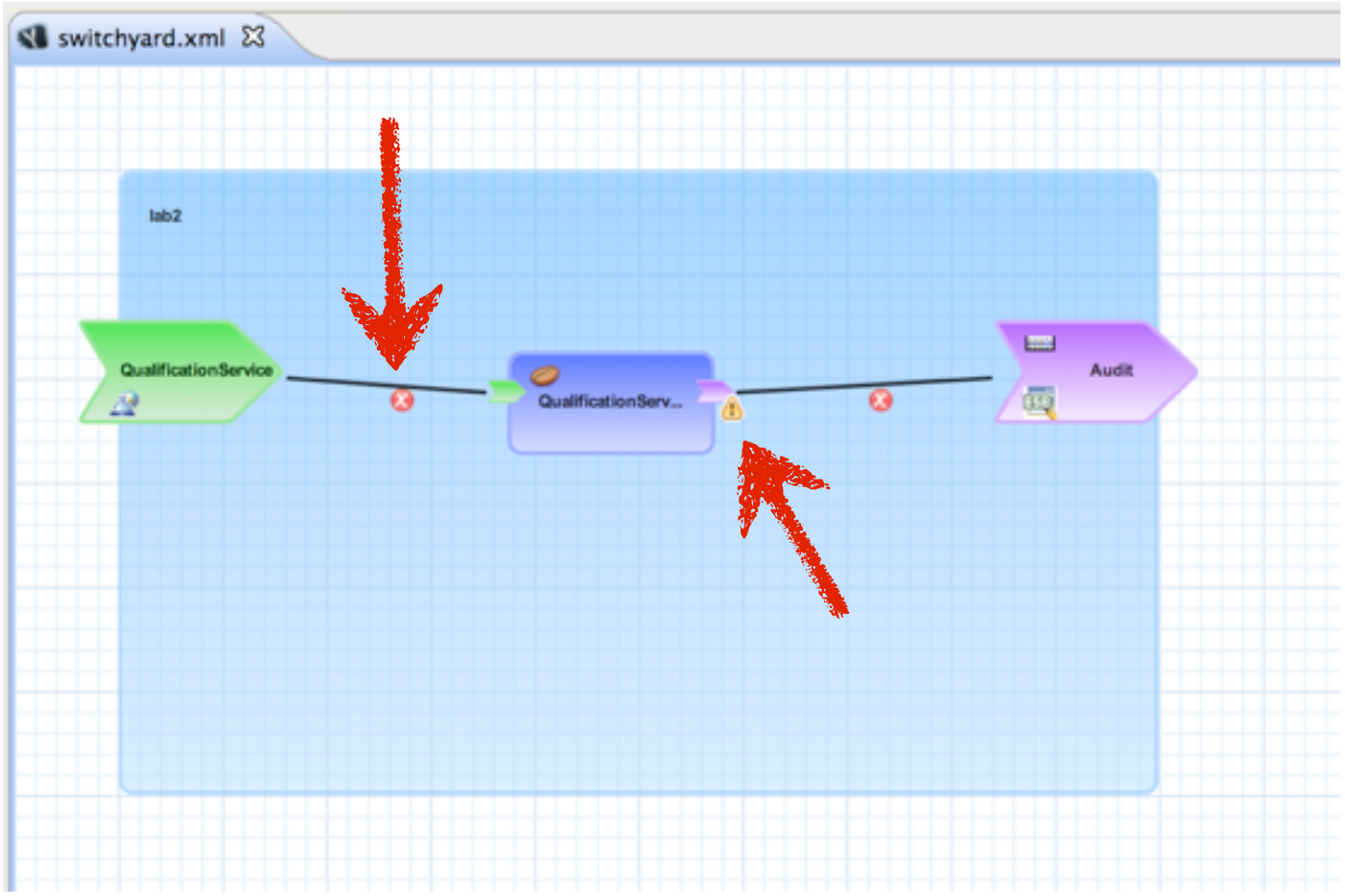
1. Double-click on the SwitchYard node in the explorer to open up the visual editor.



Lab 2 Application Model

FYI

Validation errors and warnings are displayed in the visual model as well. Hover over each warning/error to get details on what's wrong.



Step 1

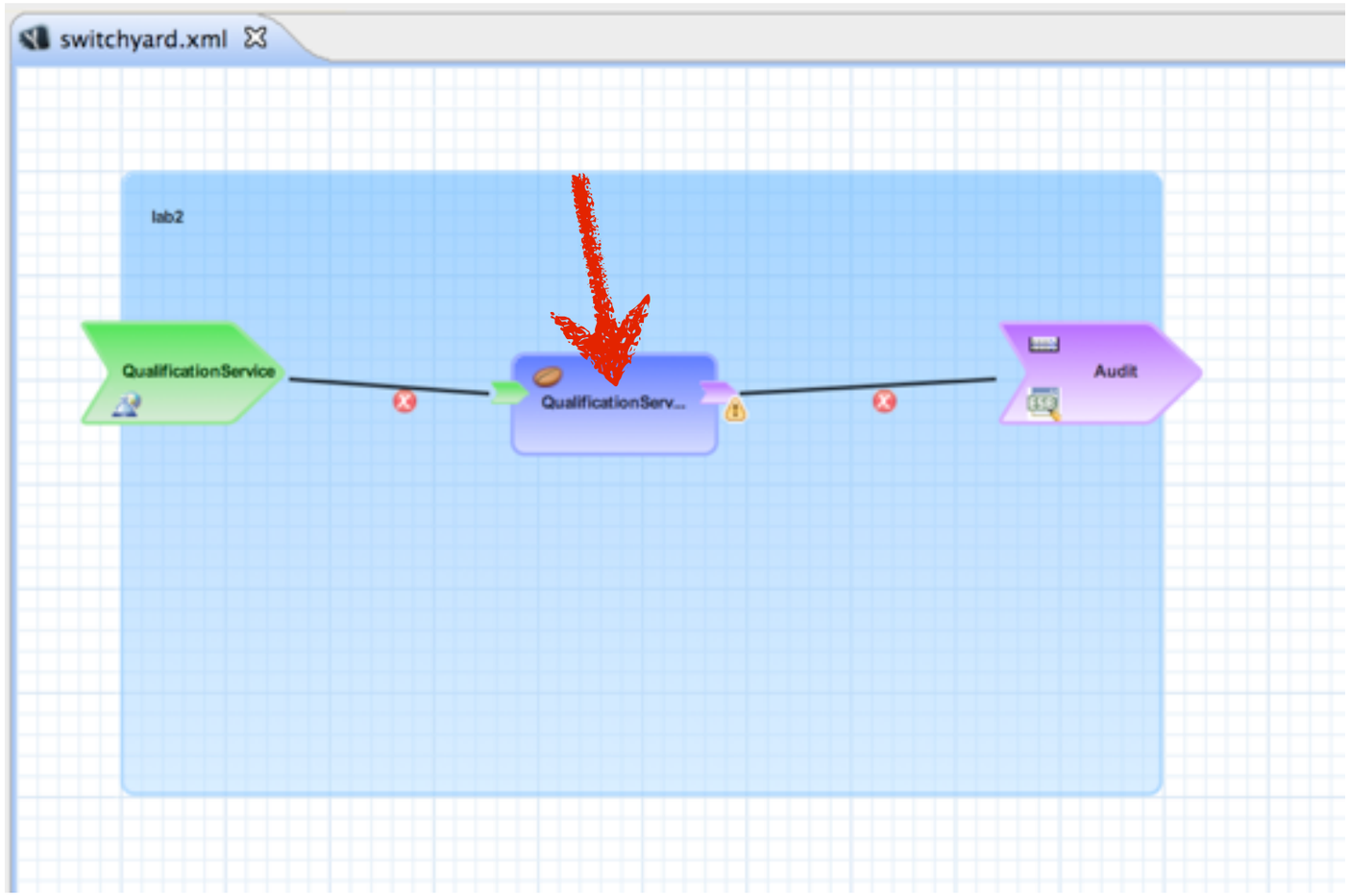
Invoking Audit Service

FYI

There is a validation warning on the QualificationService bean indicating that a reference is defined, but not used. Let's fix that.

TODO

1. Double-click on the QualificationService bean in the editor to open up the CDI Bean class in the editor.



Step 1

Update QualificationService.java

FYI

Invoking services from a Bean Service is done through a reference. We are going to inject a reference to the Audit service and invoke it from the qualify() method.

TODO

1. Add the injection for the Audit service:

```
@Inject @Reference
private Audit audit;
```

2. Invoke the Audit service at the end of the qualify method:

```
audit.recordApplication(applicant);
```

3. File - > Save

QualificationServiceBean.java

```
1  package mortgages;
2
3  import javax.inject.Inject;
8
9  @Service(QualificationService.class)
10 public class QualificationServiceBean implements QualificationService {
11
12     private static final String EASY = "easy";
13
14     @Property(name = "creditTerms")
15     private String creditTerms;
16
17     @Inject @Reference
18     private Audit audit;
19
20     @Override
21     public Applicant qualify(Applicant applicant) {
22
23         System.out.println("Qualifying applicant: " + applicant);
24
25         if (EASY.equals(creditTerms)) {
26             applicant.setApproved(true);
27         } else {
28             applicant.setApproved(applicant.getCreditScore() >= 600);
29         }
30
31         audit.recordApplication(applicant);
32
33         return applicant;
34     }
```

Step 1
Validate Changes

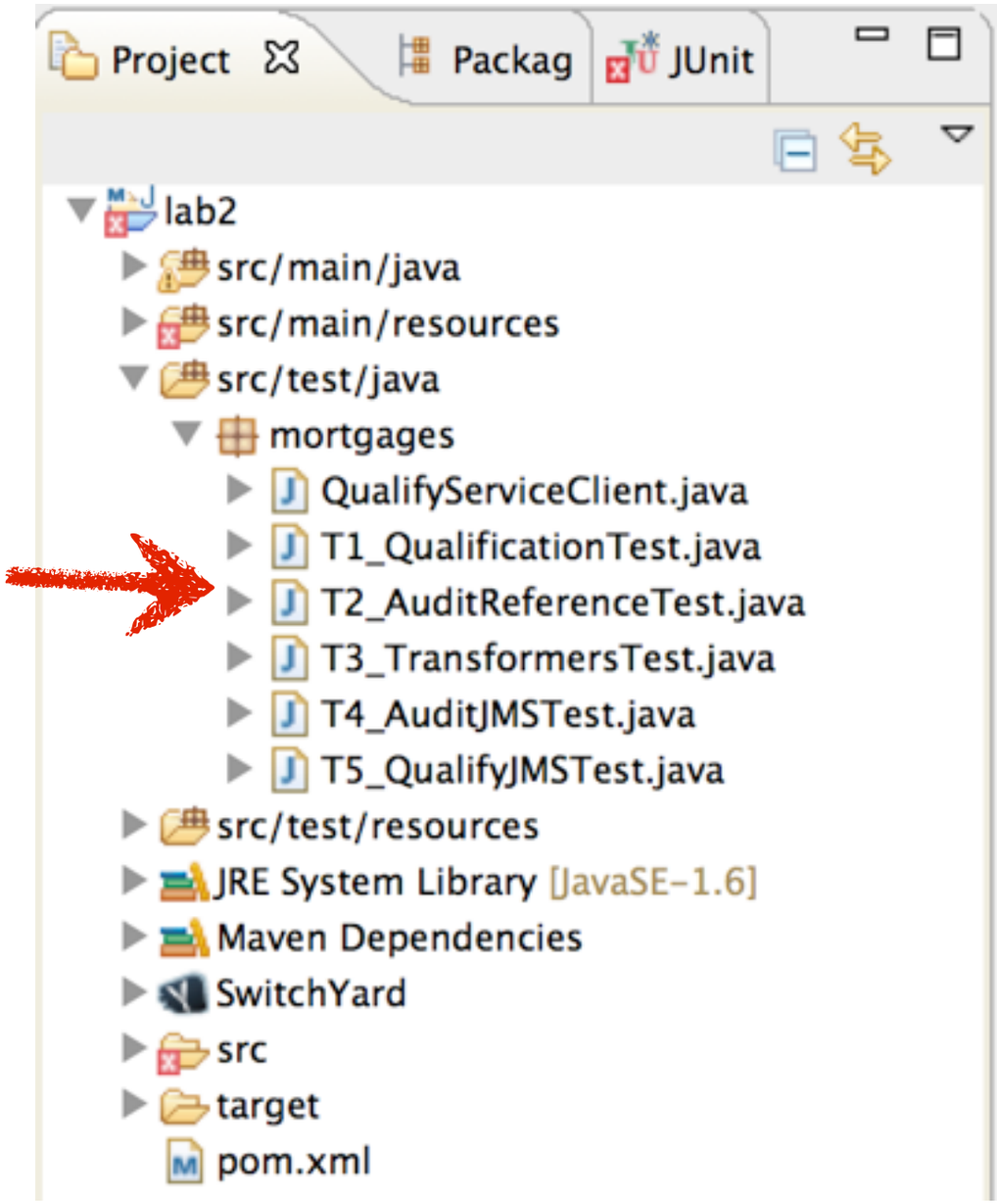
FYI

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

TODO

1. Double-click on T2_AuditReferenceTest 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.

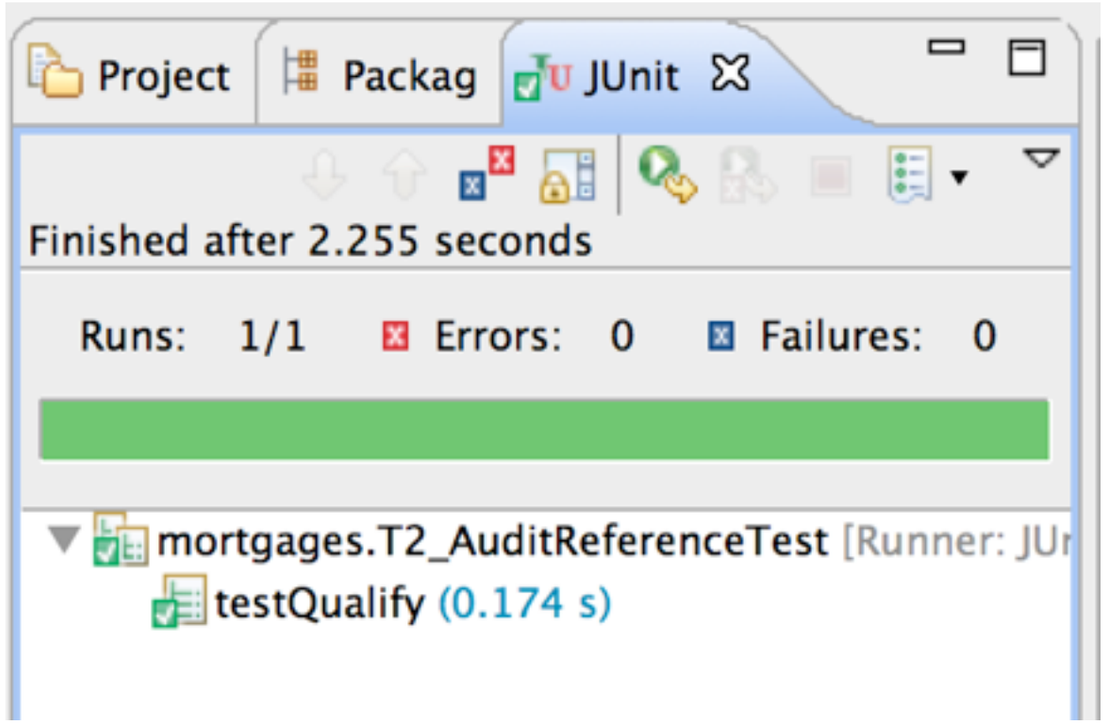


Step 1

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.

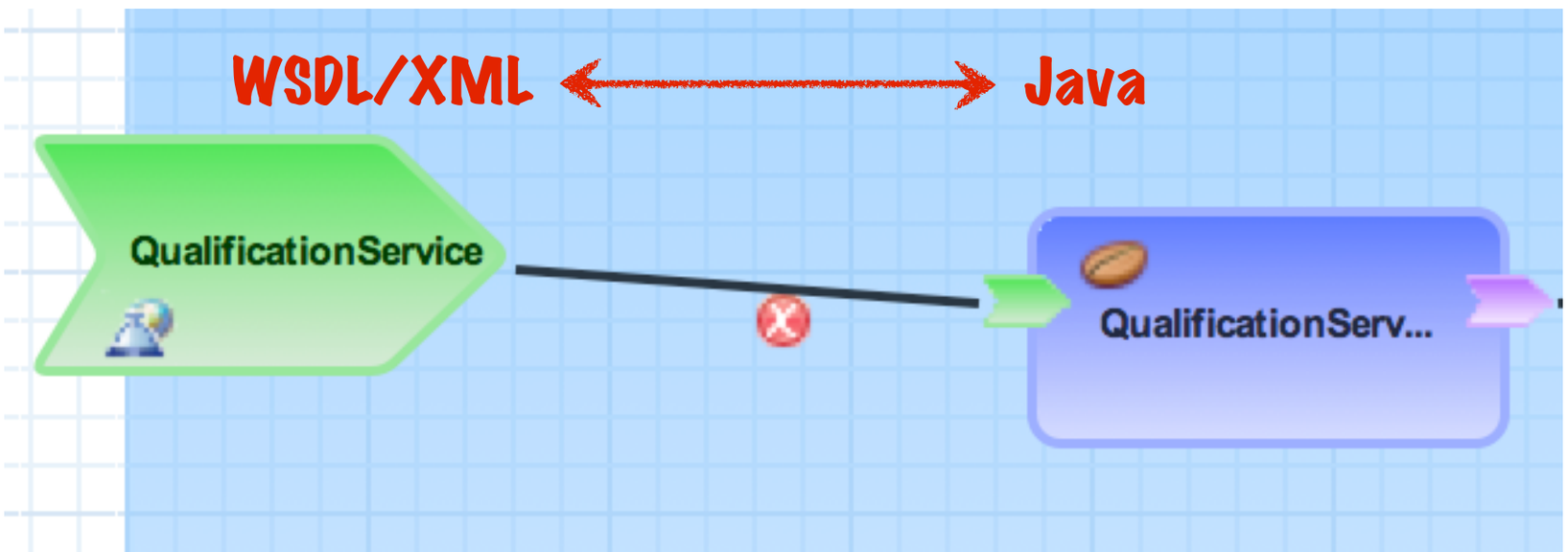


Step 2

Adding Transformers

FYI

Transformers allow you to transform between different data formats in a declarative manner outside of your implementation logic. In lab2, QualificationService is exposed to external consumers using WSDL, so the payload type is XML. The implementation of QualificationService is a CDI Bean and expects a Java type. We need to add a transformer which transforms between the XML and Java format of an applicant.



Step 2

Add XML to Java Transformer

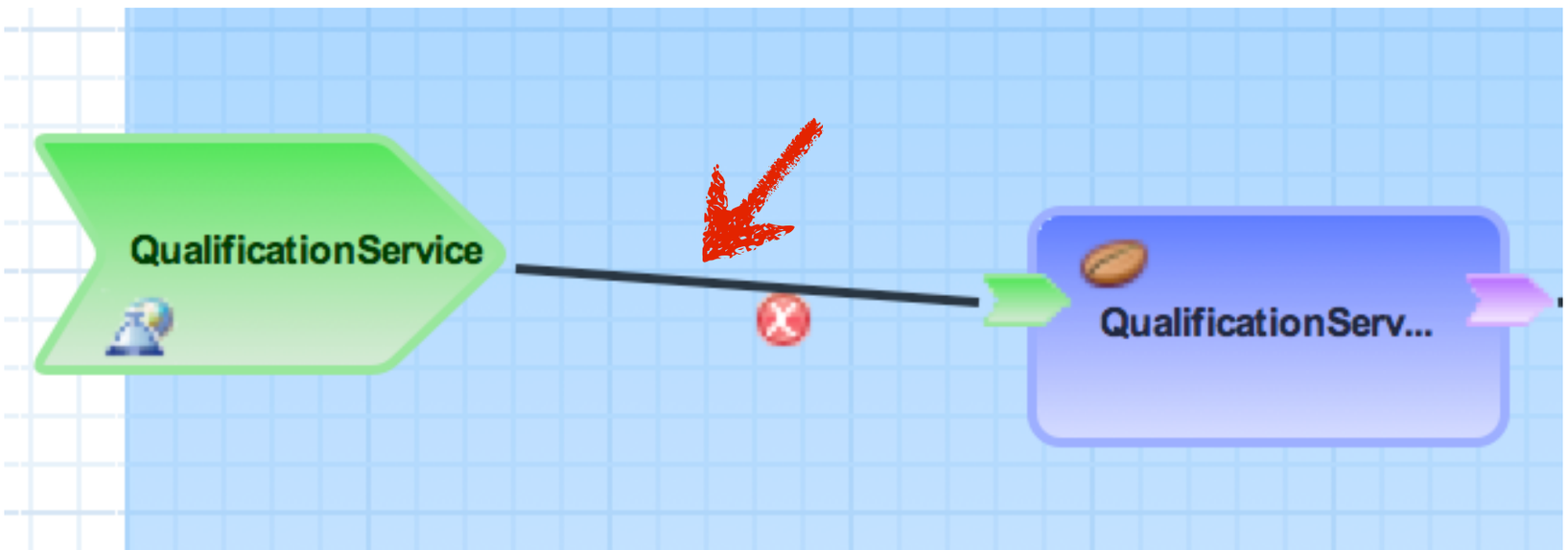
TODO

1. Right-click on the wire for QualificationService.

2. Select ‘Create Required Transformers’

FYI

Since all services have a contract, we know which data types are used for any message exchange on a wire. When you select a wire the tooling will prompt you to create transformers for that specific service.



Step 2

Add XML to Java Transformer

FYI

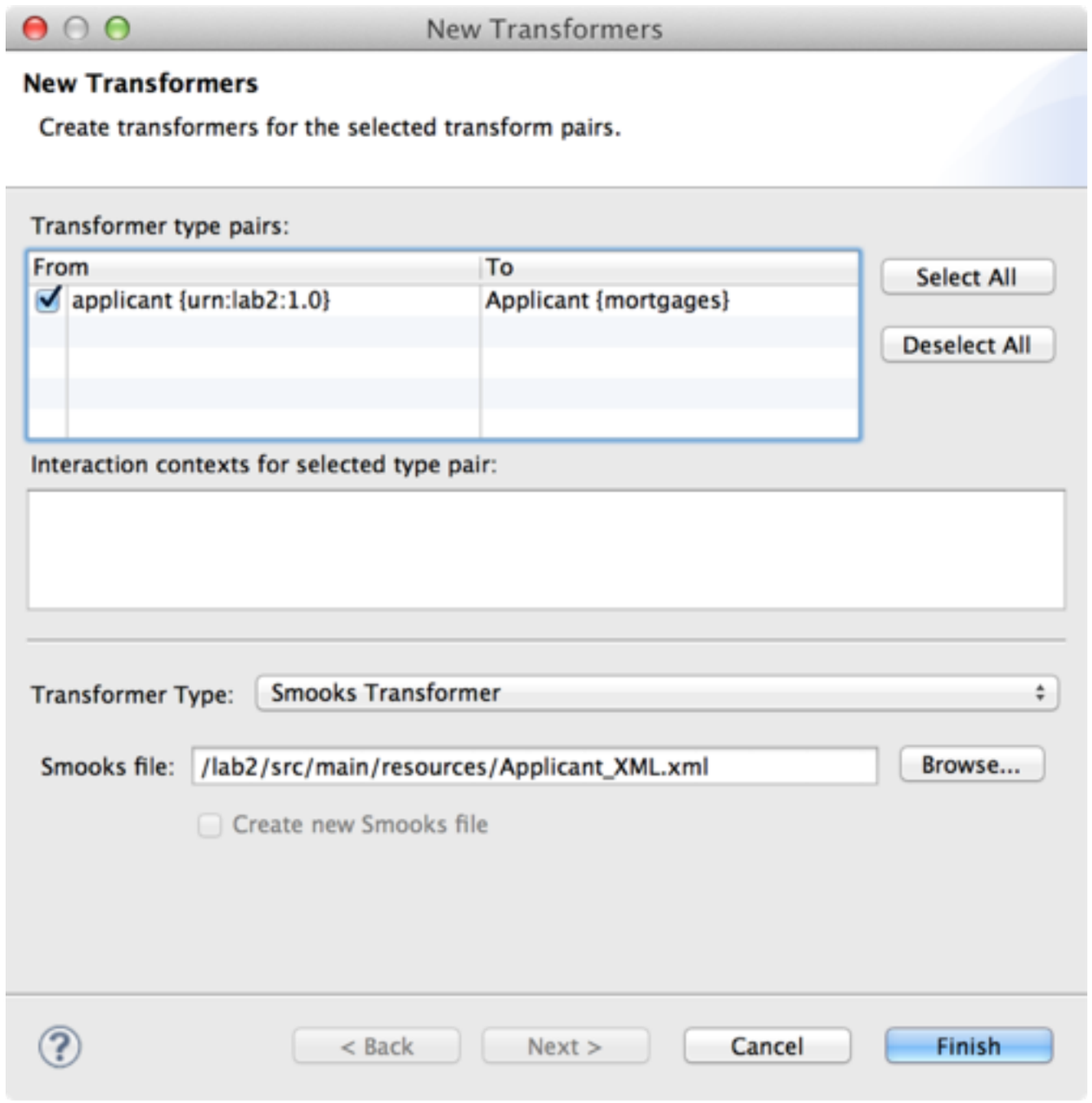
There should only be one transformer pair listed (see image at right). If you see more than one line then click Cancel and try the last step again (be sure to click on the wire).

TODO

1. For Transformer Type, select “Smooks Transformer”.

2. Click Browse ... next to the Smooks file input and select ‘Applicant_XML.xml’ in the resulting window.

3. Click Finish.



Step 2

Add Java to JSON Transformer

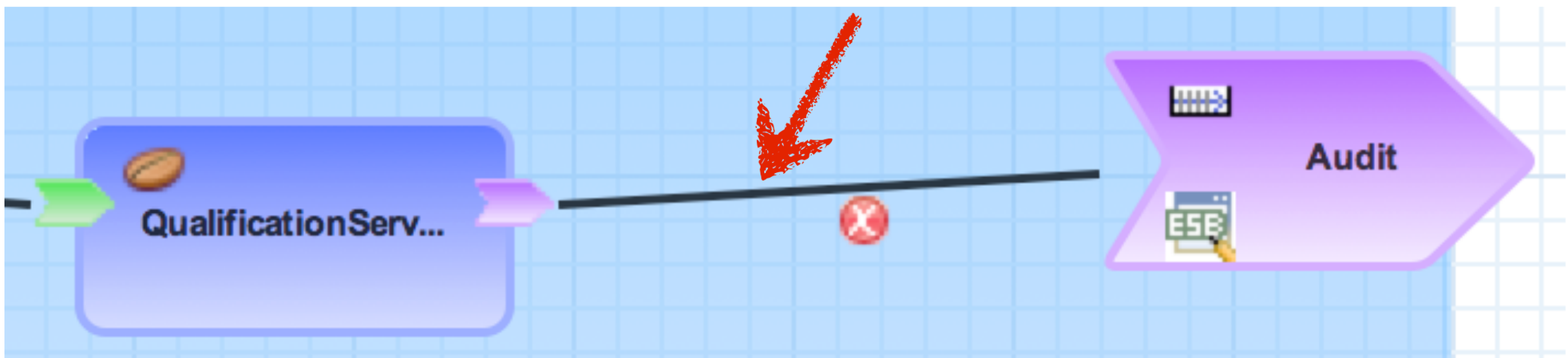
TODO

1. Right-click on the wire for the Audit service.

2. Select ‘Create Required Transformers’

FYI

The Audit service expects messages to be in JSON format, so another transformer is required to transform between the Java representation of Applicant to JSON.



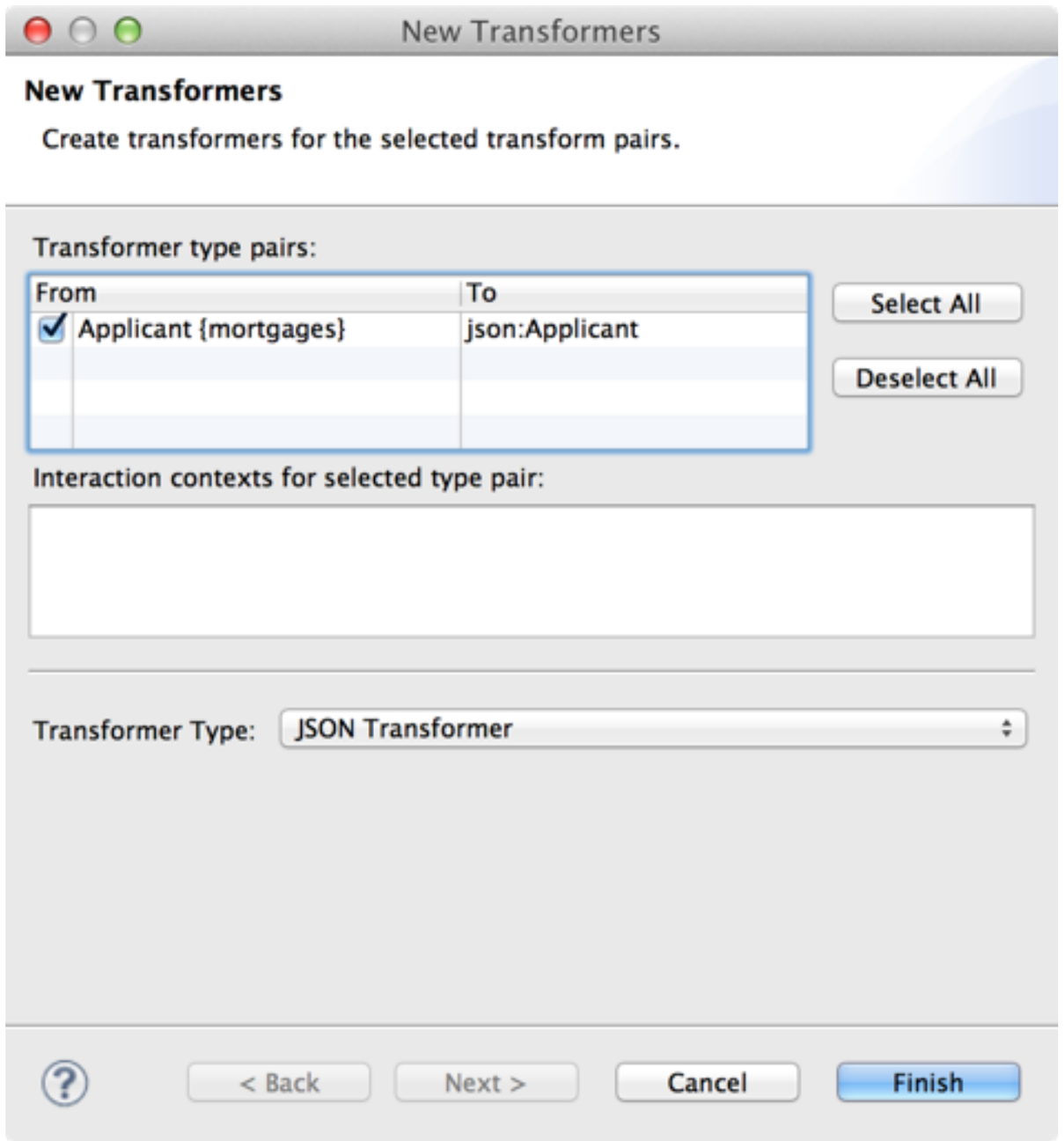
Step 2

Add Java to JSON Transformer

TODO

1. For Transformer Type, select “JSON Transformer”.

2. Click Finish.



Step 2
Validate Changes

FYI

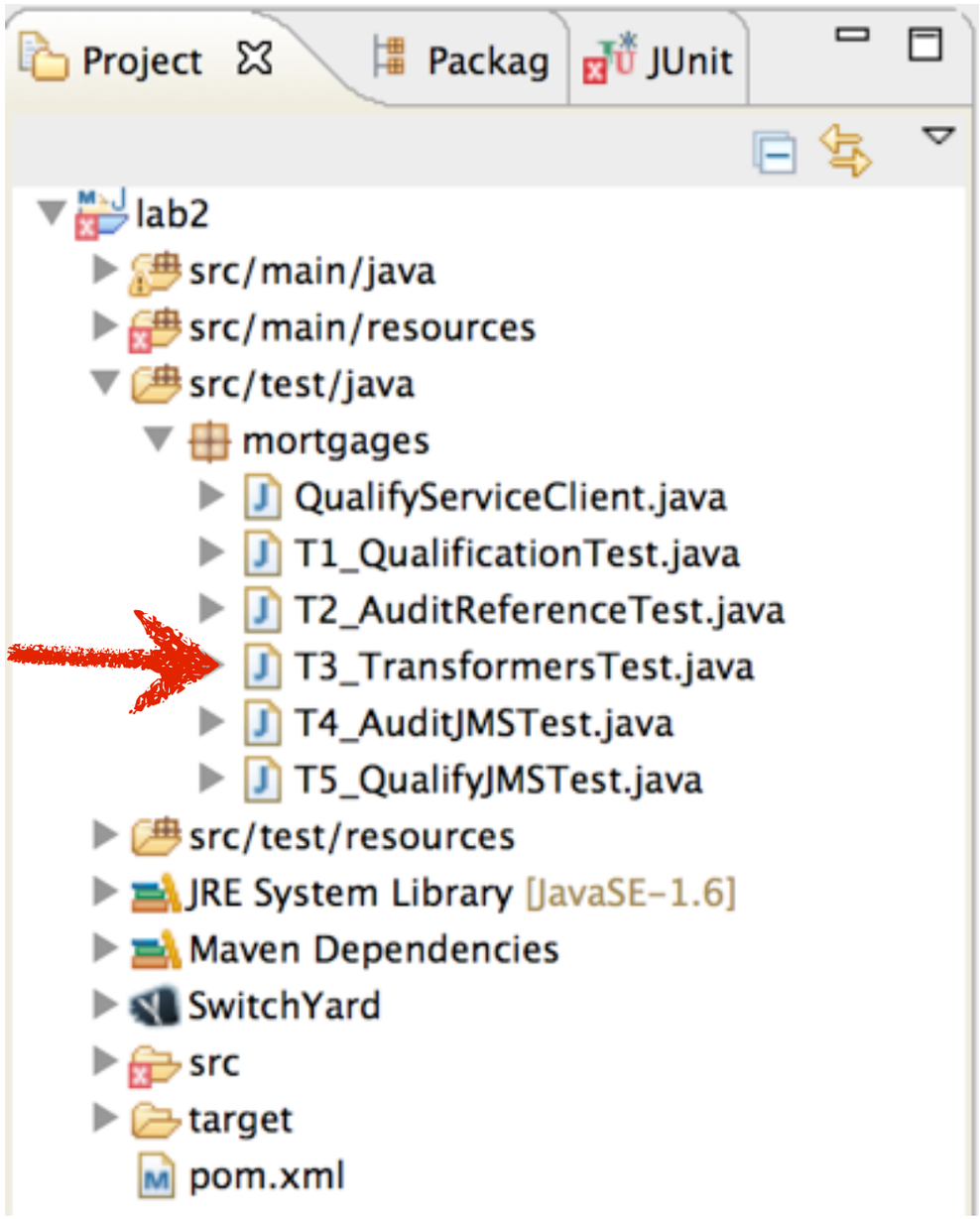
You have completed the changes required for step 2. Let's validate the changes using a service unit test.

TODO

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

2. Double-click on T3_TransformersTest 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.

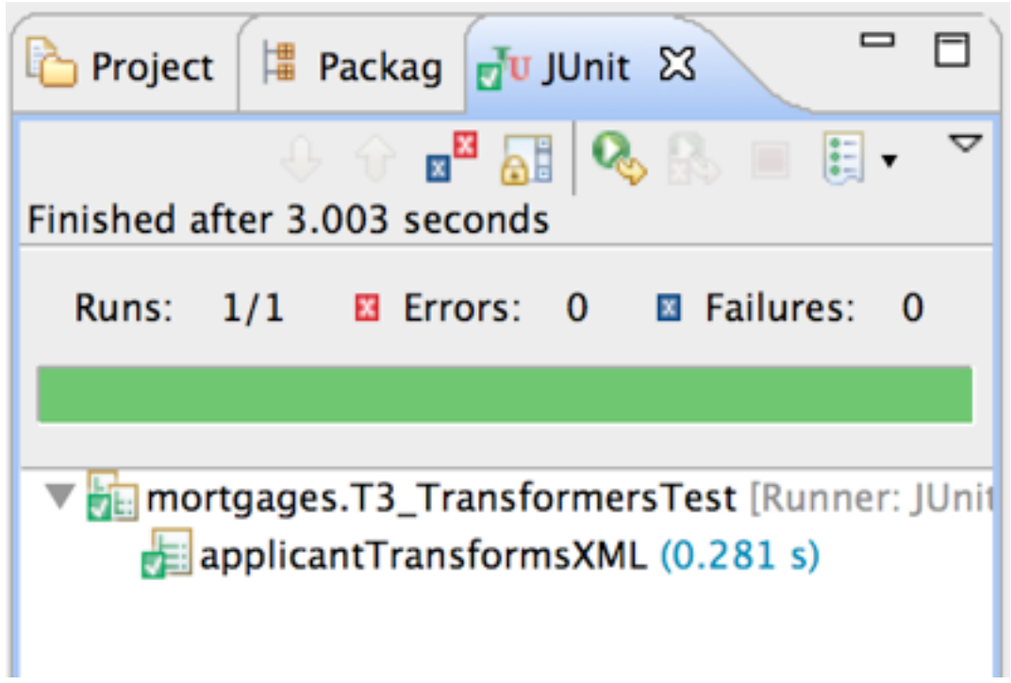


Step 2

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.

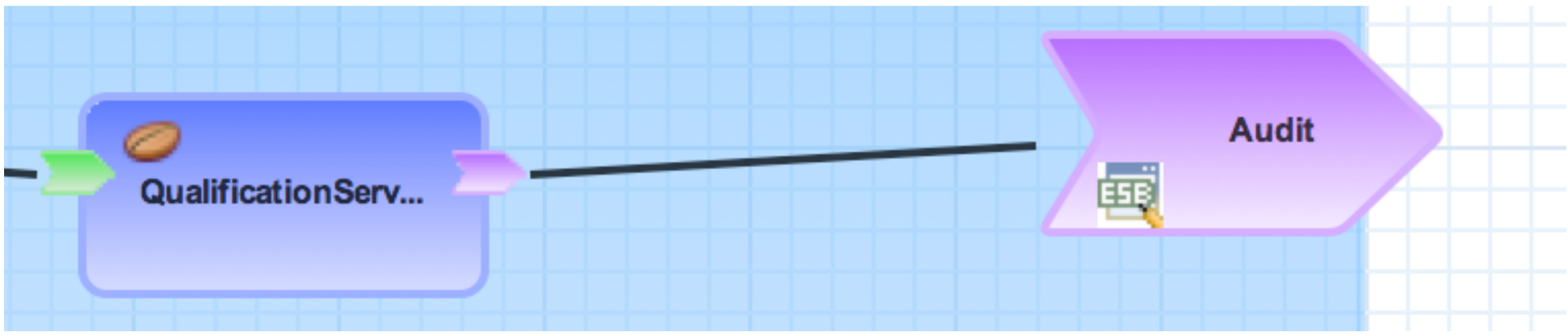


Step 3

Adding JMS Reference Binding

FYI

The QualificationService bean invokes the Audit service through a reference. This reference is promoted to a composite reference, which means the service being referenced is outside the application. We are going to add a JMS binding to the composite reference to access the Audit service.



Step 3

Add JMS Reference Binding

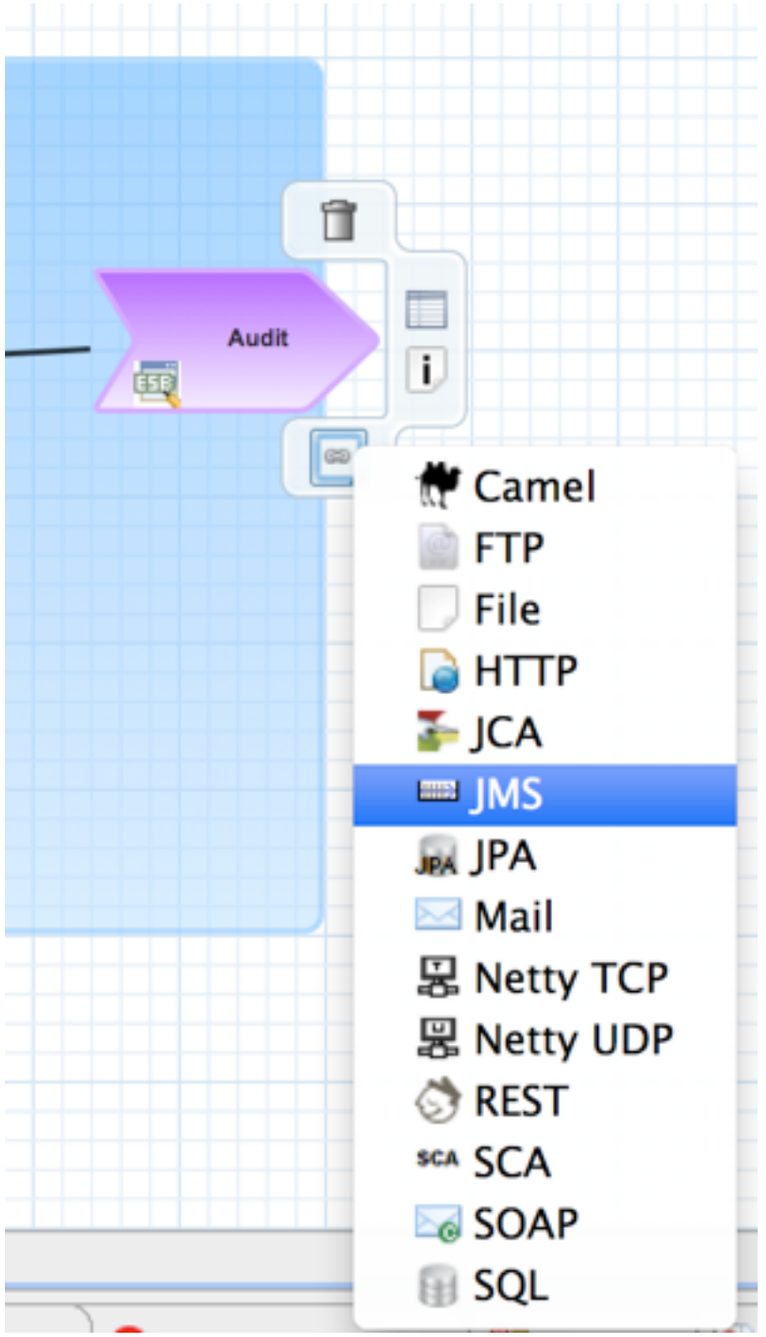
FYI

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 Audit composite reference to access the button bar.

2. Click on the bindings button and select JMS.



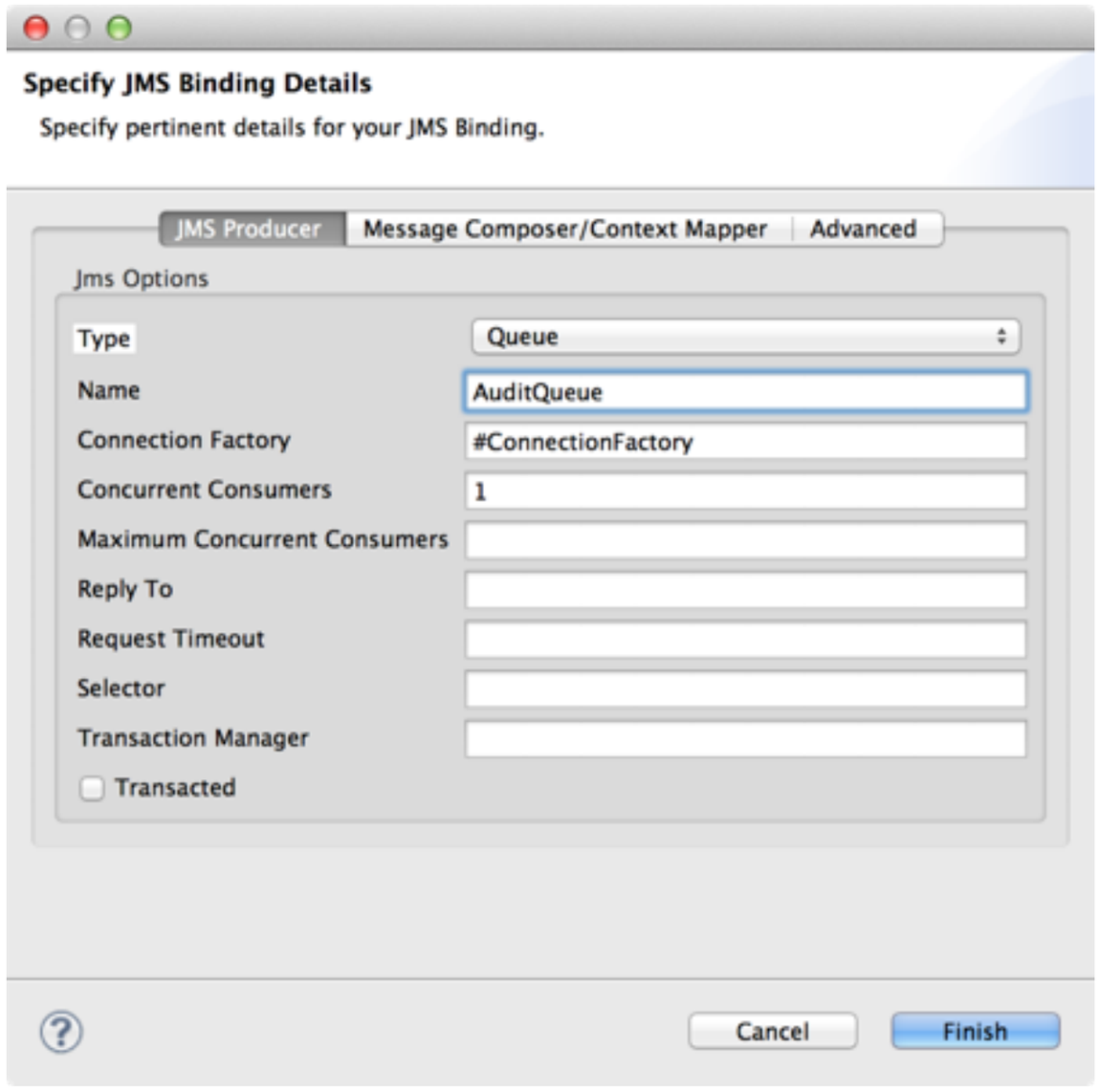
Step 3

Configure JMS Binding

TODO

1. `Name : AuditQueue`

2. Click Finish.



Step 3
Validate Changes

FYI

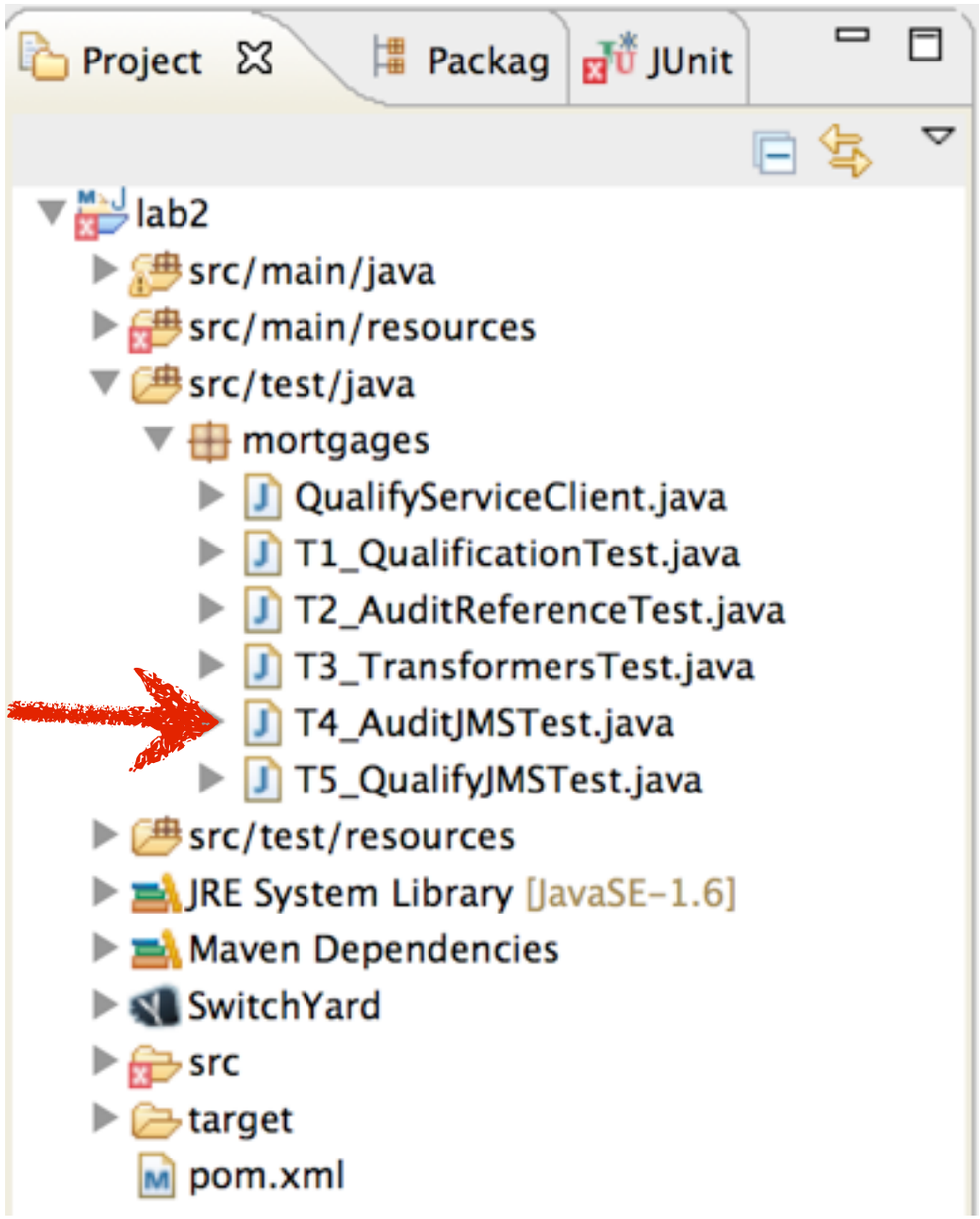
You have completed the changes required for step 3. Let's validate the changes using a service unit test.

TODO

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

2. Double-click on T4_AuditJMSTest 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.

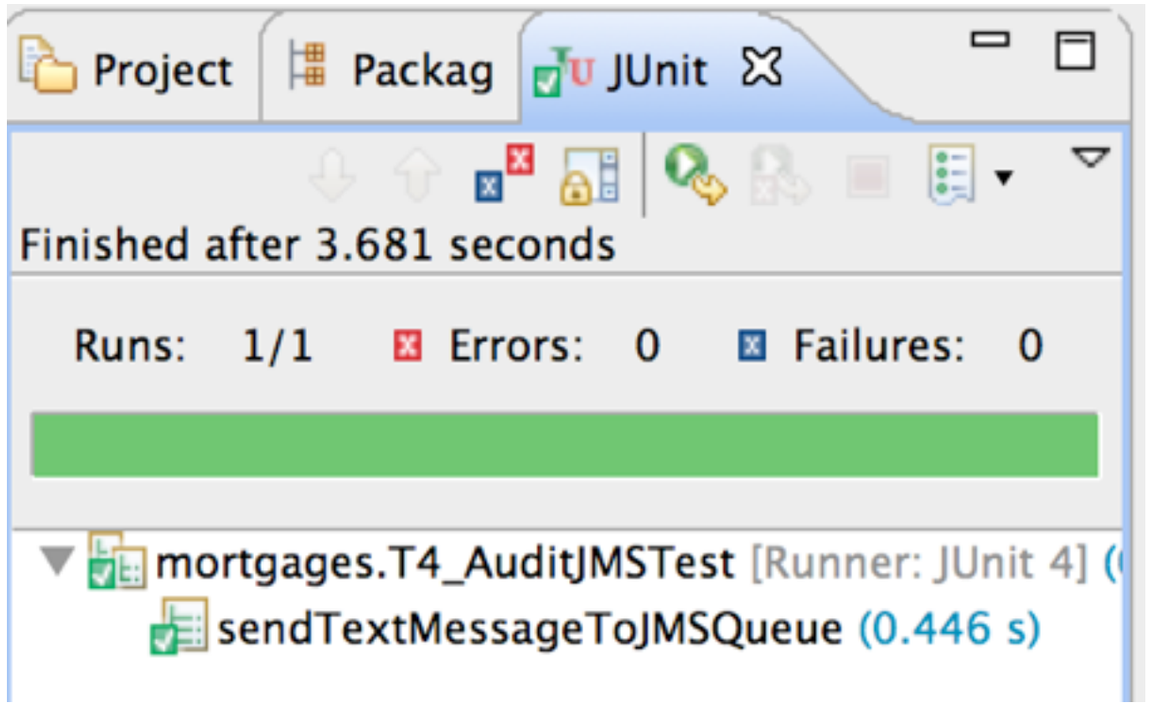


Step 3

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.

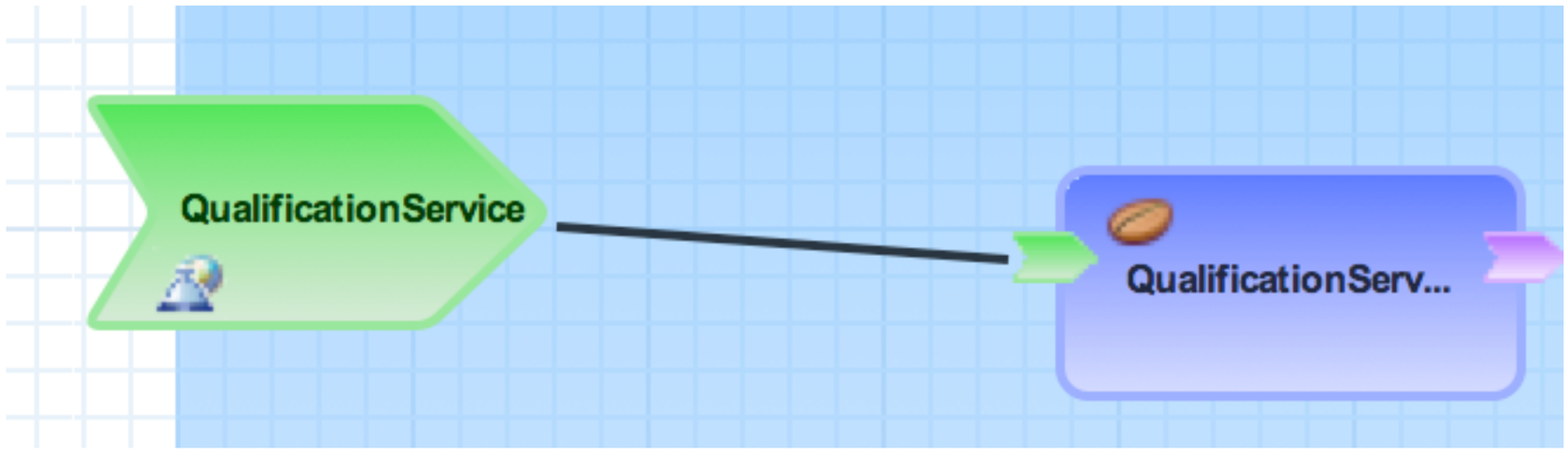


Step 4

Adding JMS Service Binding

FYI

QualificationService is promoted as a composite service, which means that it is exposed to service consumers outside the application. In order to invoke the composite service, we need to use one or more bindings to make it available. In Step 4, we will add a JMS binding to QualificationService.



Step 4

Add JMS Service Binding

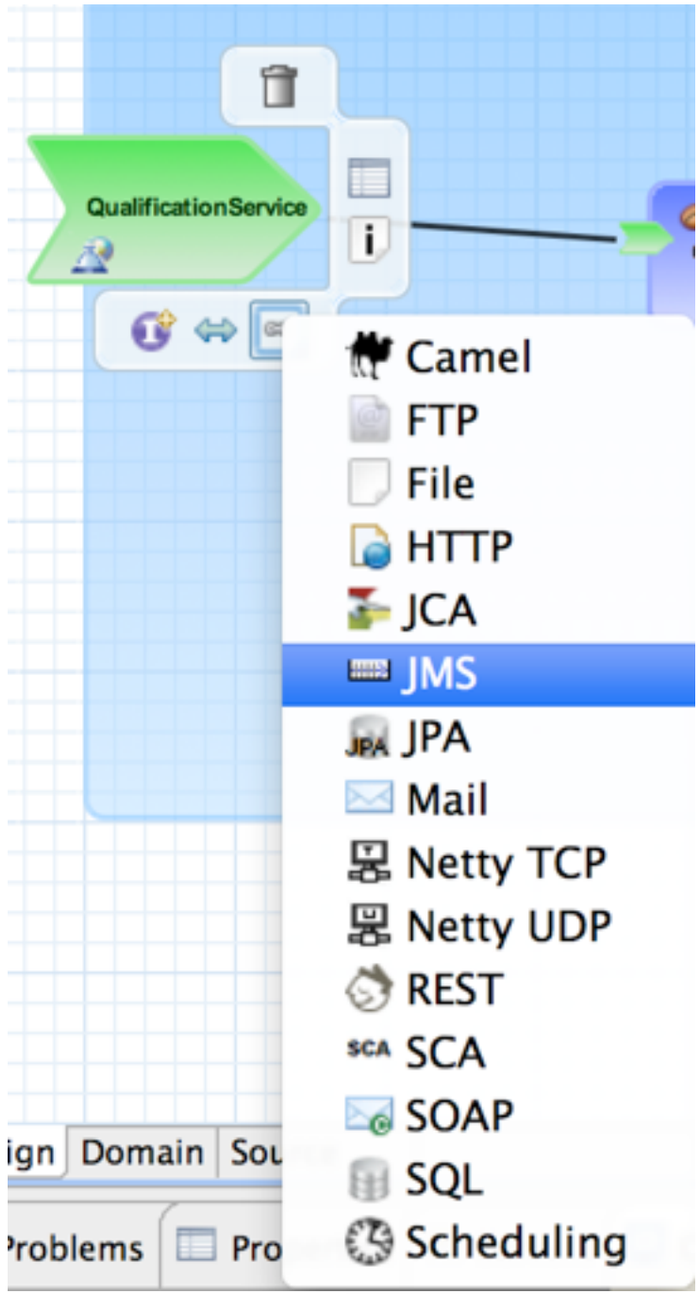
FYI

Very similar to Step 3, but we are adding the binding to a service instead of a reference.

TODO

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

2. Click on the bindings button and select JMS.



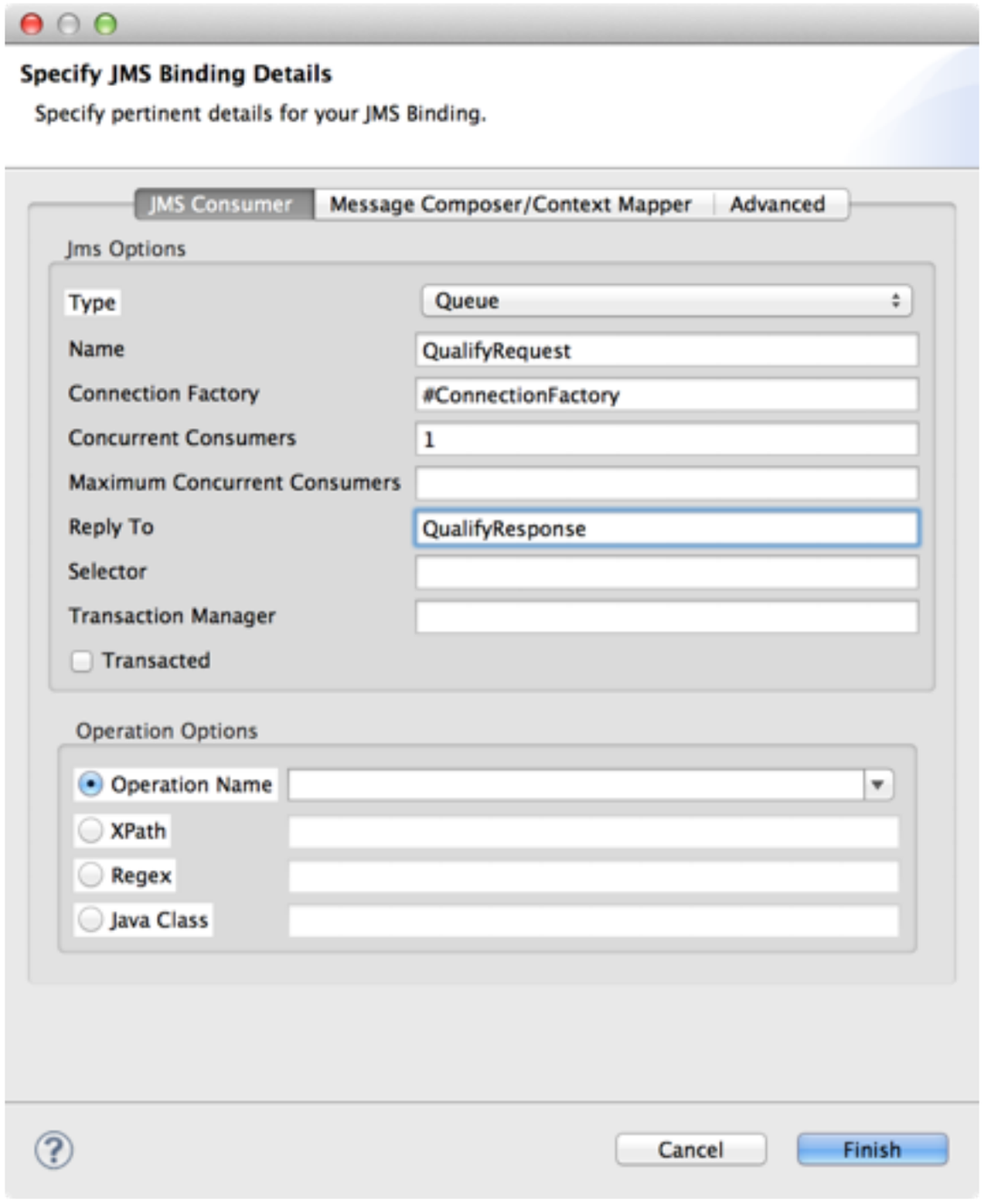
Step 3

Configure JMS Binding

TODO

1. Name : QualifyRequest
   ReplyTo : QualifyResponse

2. Click Finish.



Step 4

Validate Changes

FYI

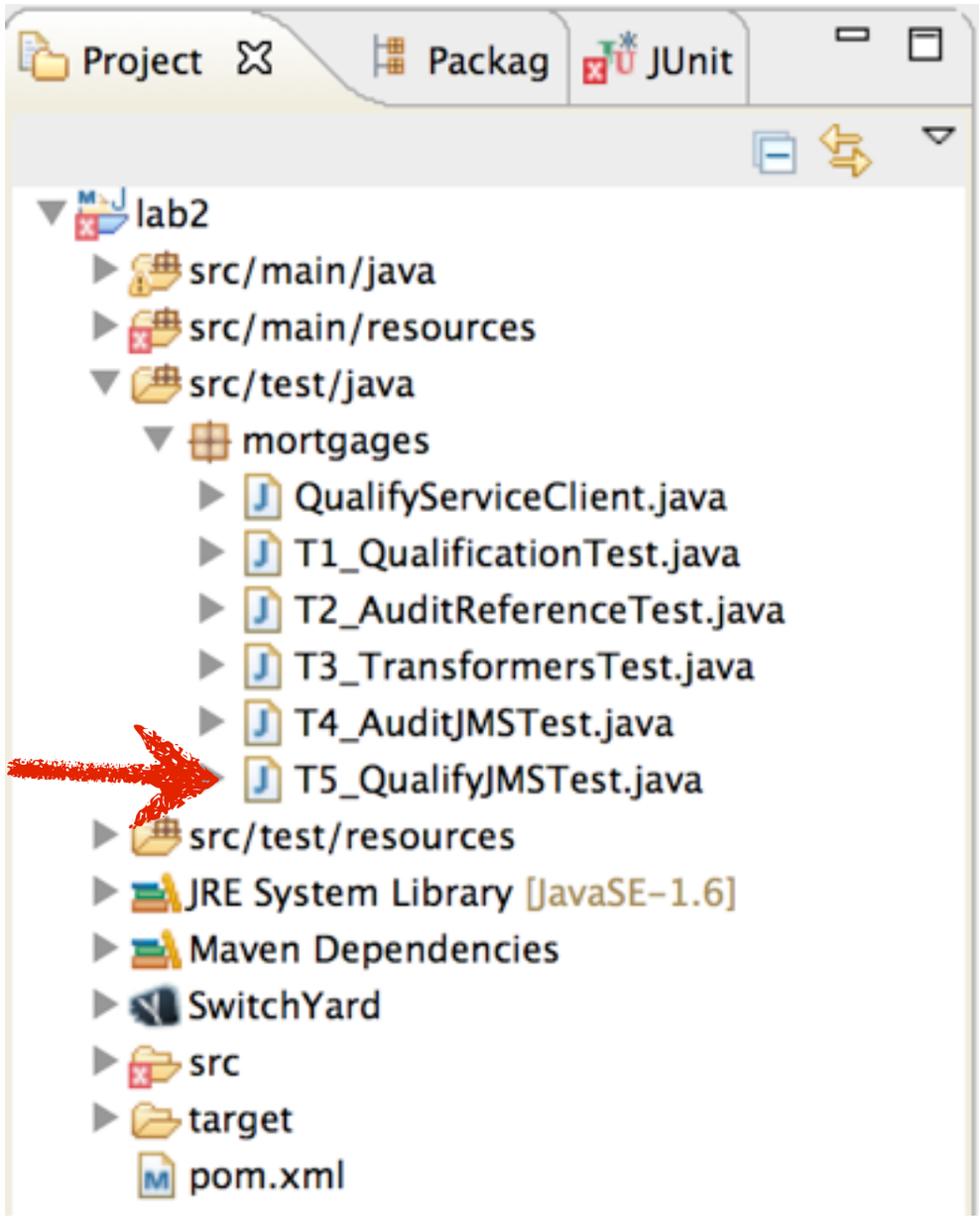
You have completed the changes required for step 4. Let's validate the changes using a service unit test.

TODO

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

2. Double-click on T5_QualifyJMSTest 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.

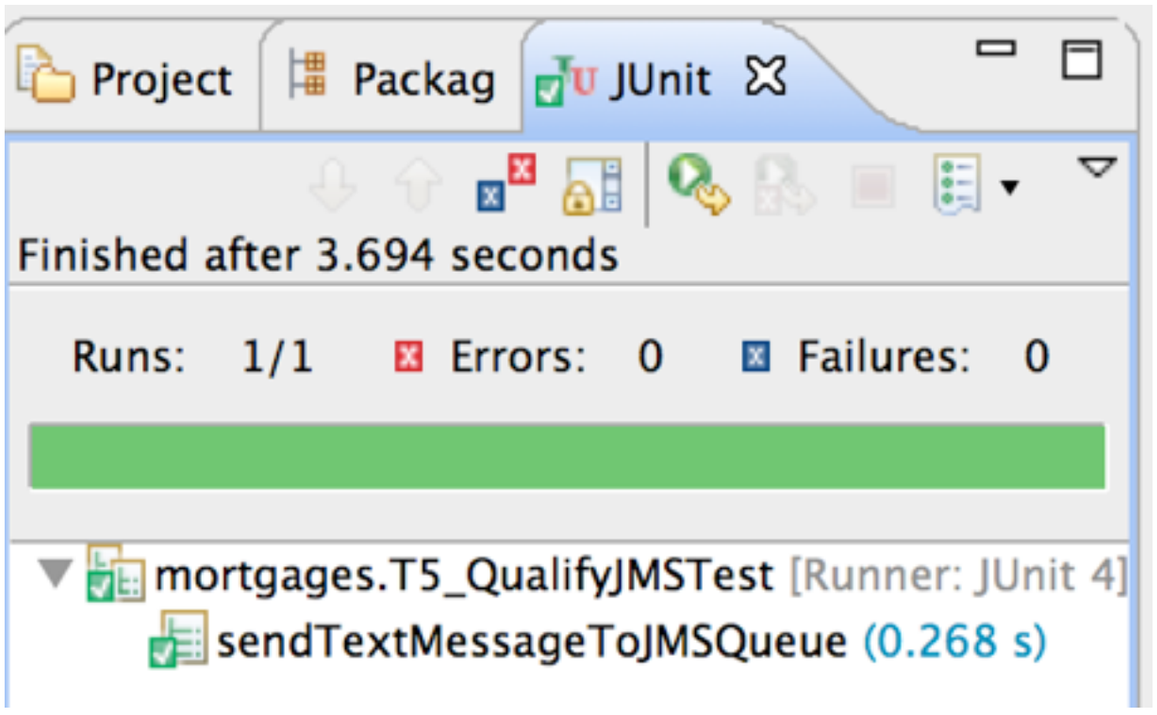


Step 4

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

- This is an optional step if you want to see lab2 deployed to the server.
- Step 5 includes:
  - Deploying required queues to SwitchYard
  - Deploy application
  - Test application with JMS client application

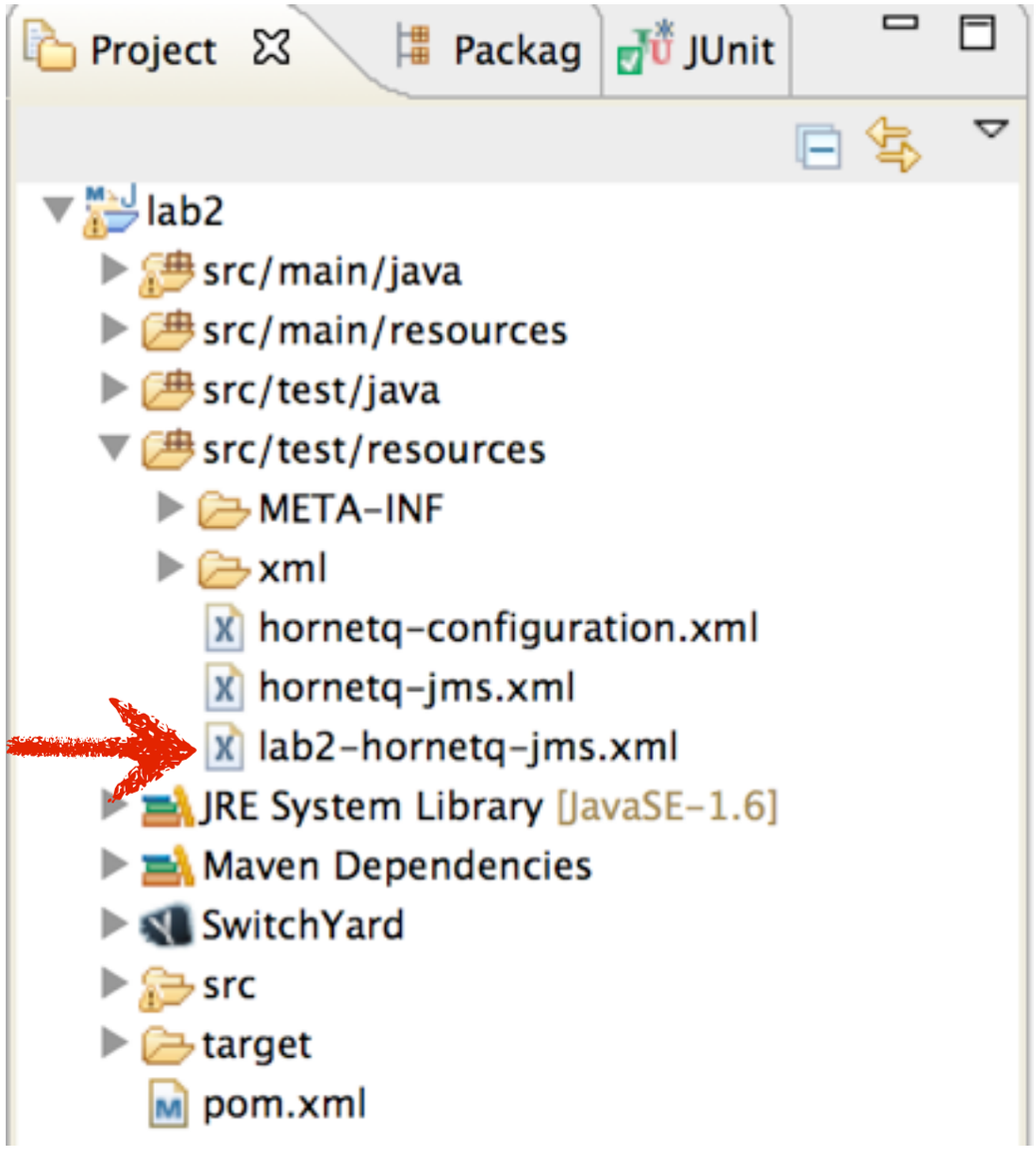
Step 5

Deploy Queue

TODO

1. Right-click on lab2-hornetq-jms.xml and select ‘Mark as Deployable’.

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

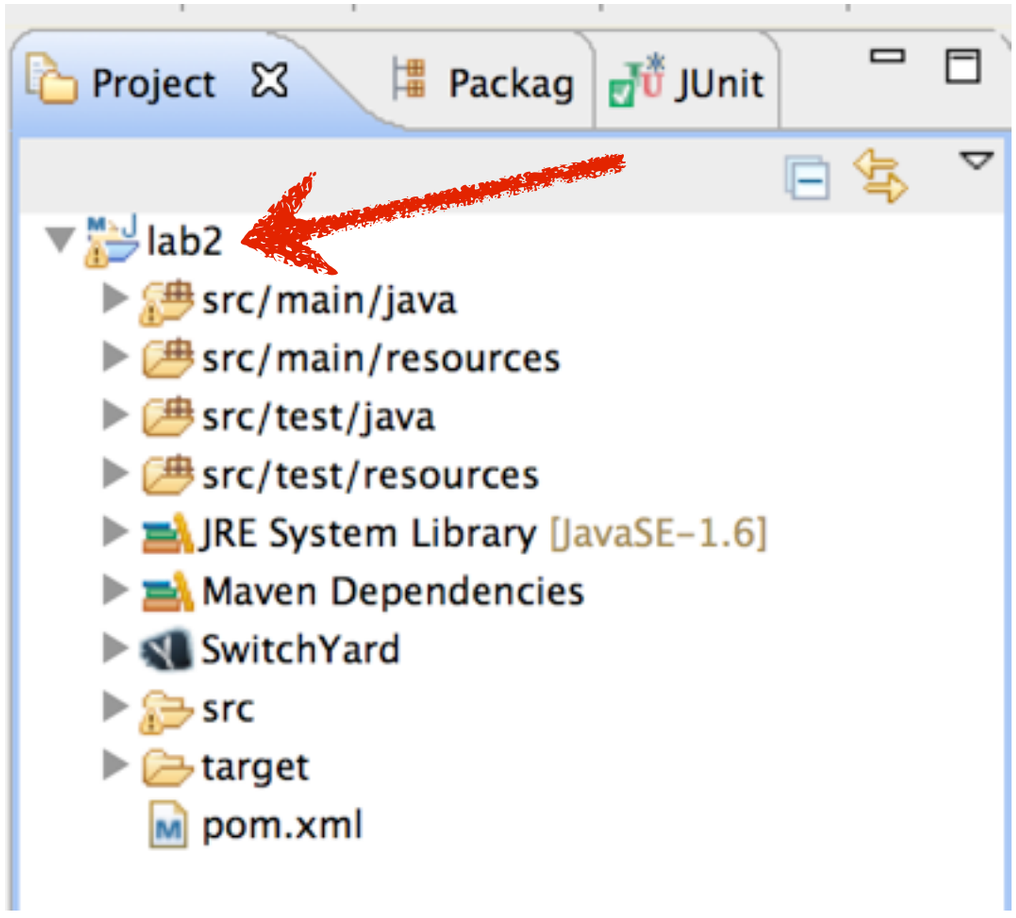


Step 5

Deploy Application

TODO

1. Right-click on the lab2 project and select Run As ... Run on Server

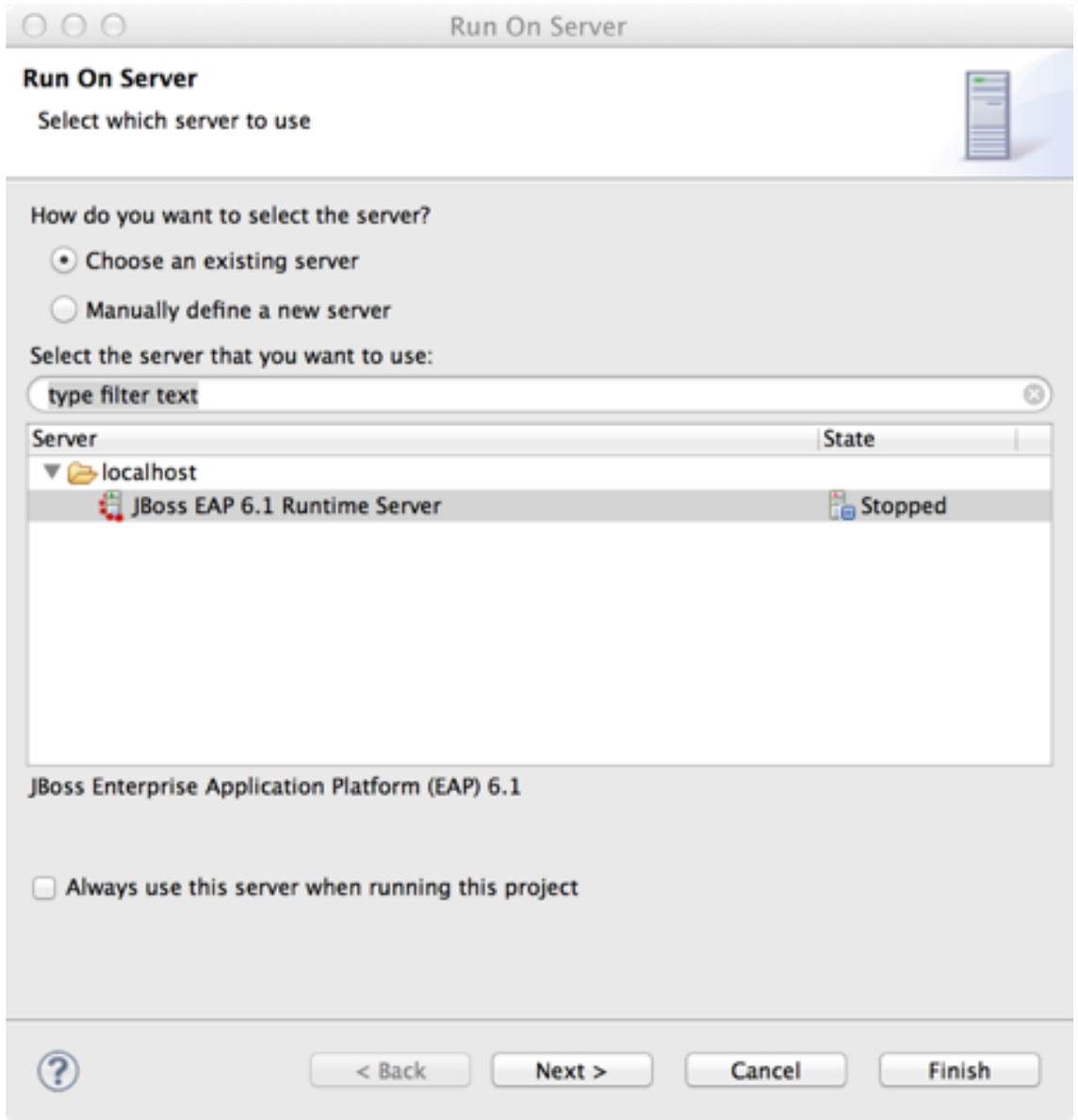


Select Server

TODO

1. Select the JBoss EAP 6.1 Runtime Server

2. Click Finish



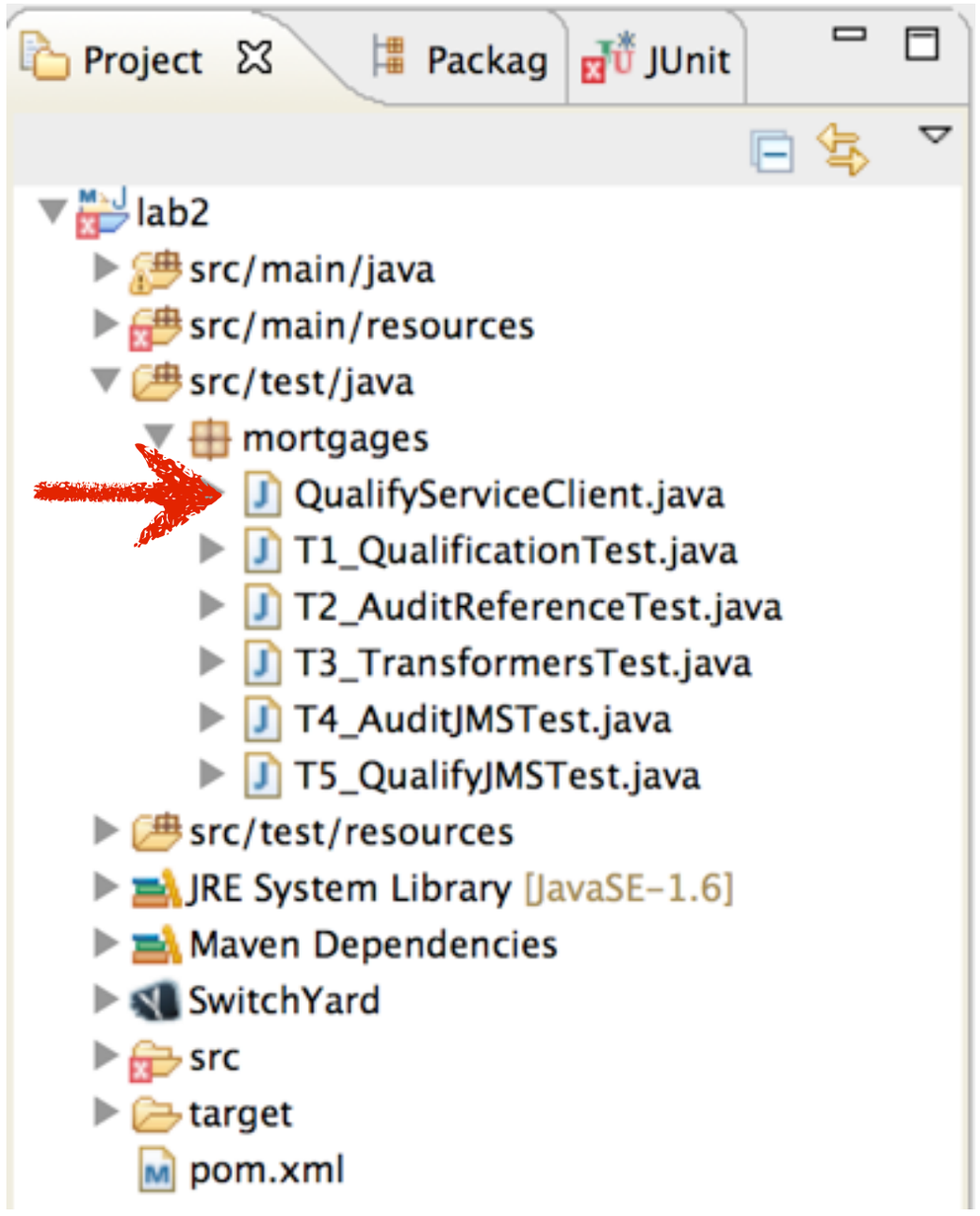
Step 5

Run Test Client

TODO

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

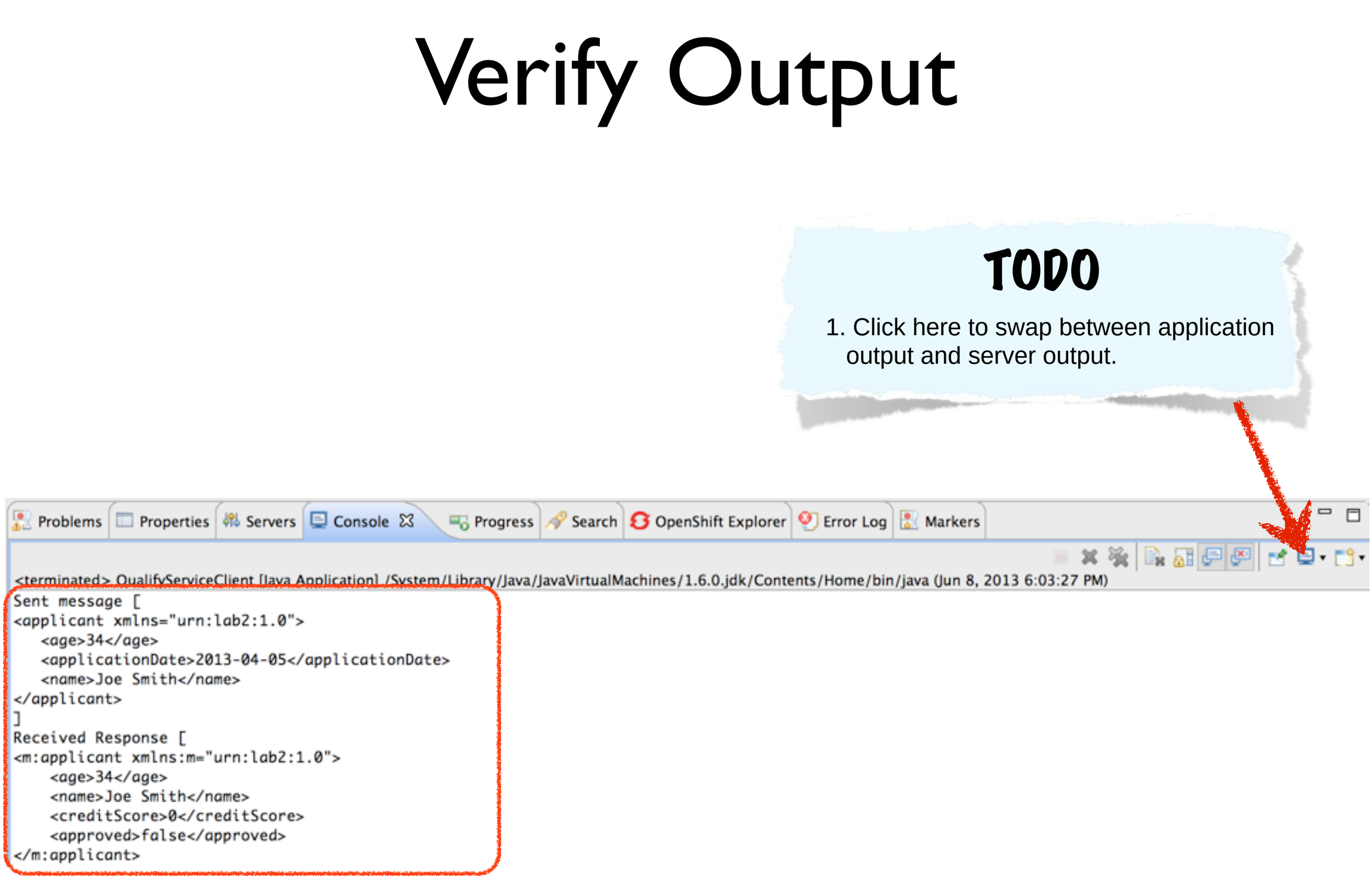
2. Go to the Run menu in the main menu bar and select ‘Run As -> Java Application’



Verify Output

TODO

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



Lab 2 Complete!