

Lab 4

Administration, Debugging, and General Hackery

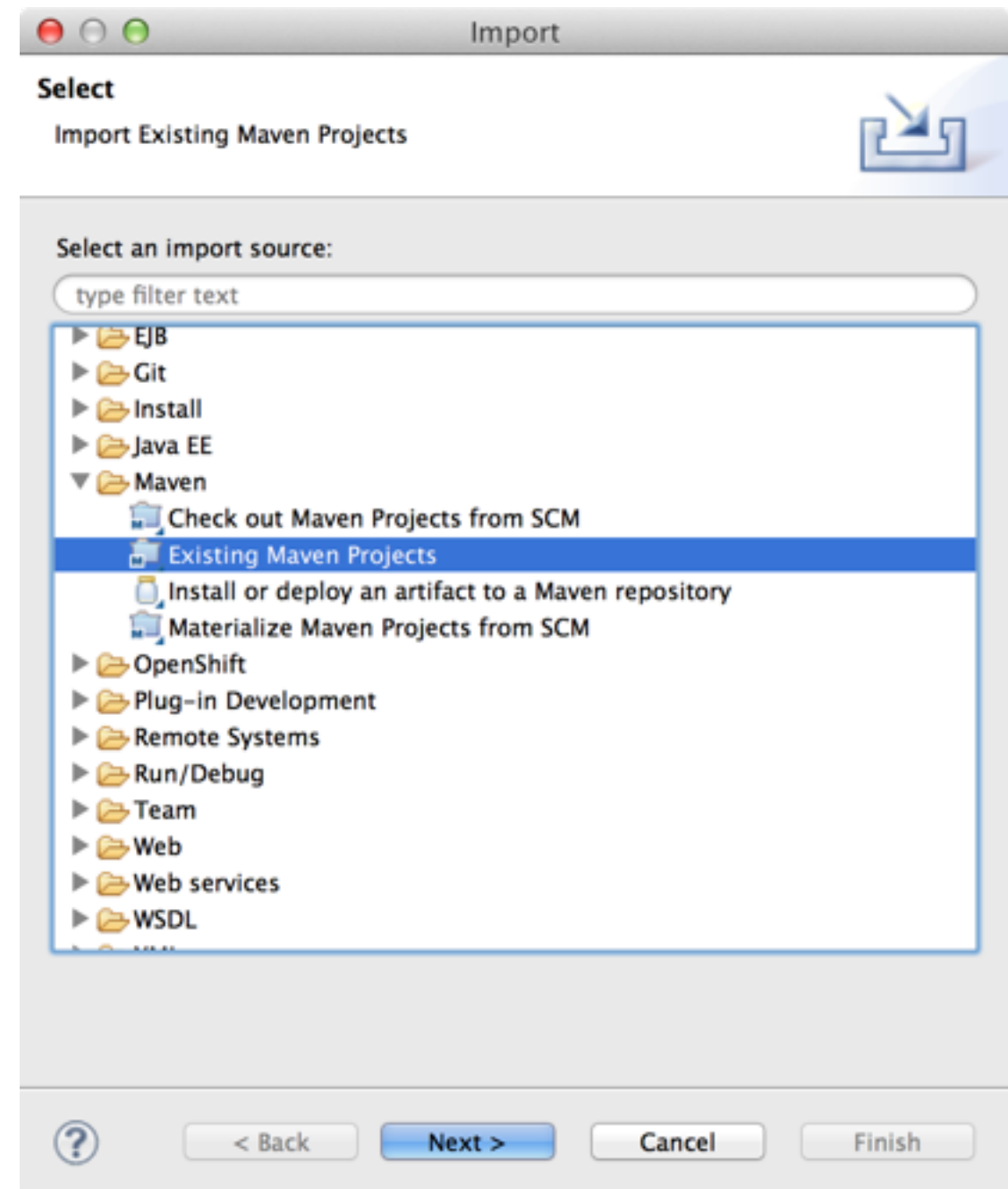
Lab Goals

- Cruise around the administration console and find some goodies
- Introduce message tracing to diagnose problems in poorly behaved applications
- Black belt debugging with auditors

Importing Lab 4

TODO

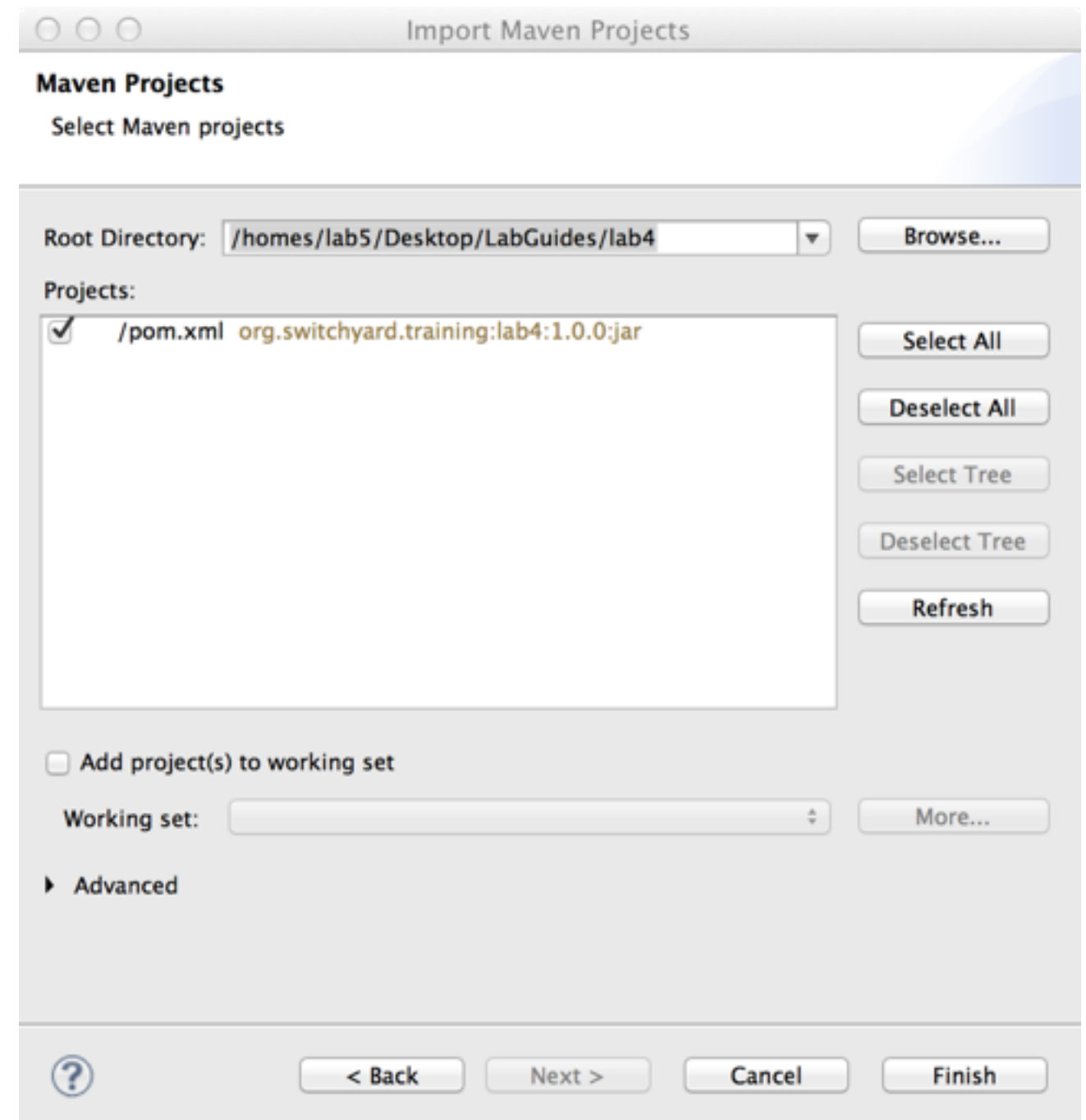
1. File -> Import ... from the JBDS menu.
2. Select Maven -> Existing Maven Projects
3. Click Next



Importing Lab 4

TODO

1. Click Browse ... and navigate to:
`/home/learning/summit2013/lab4`
2. Make sure the pom.xml is checked for:
`org.switchyard.training:lab4`
3. Click Finish



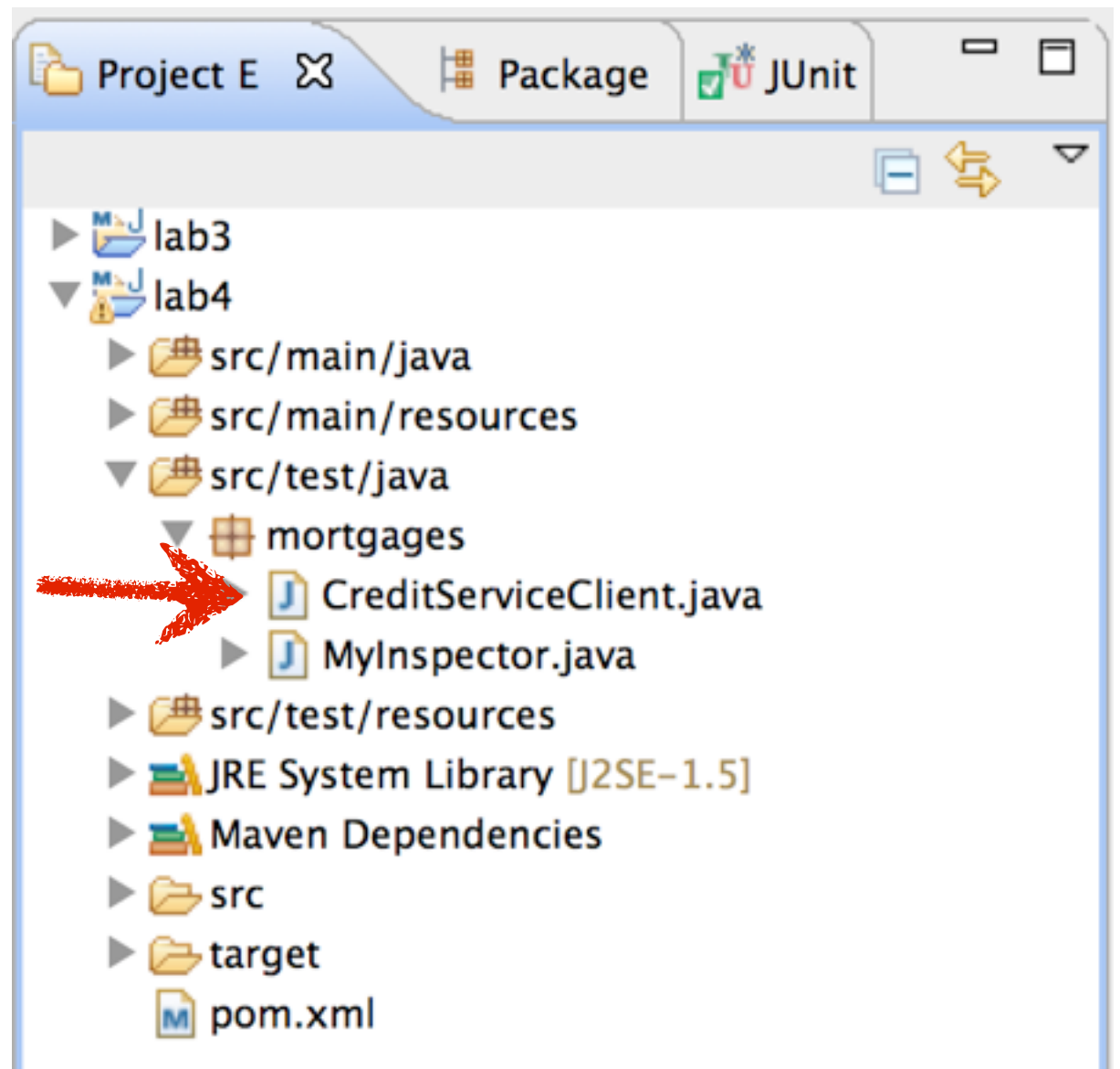
Lab 4

FYI

Lab 4 is not a SwitchYard-based application. It is simply a test client project for the application you deployed in Lab 3. Make sure the lab3 application is deployed and running.

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'



Seeding Monitoring Data

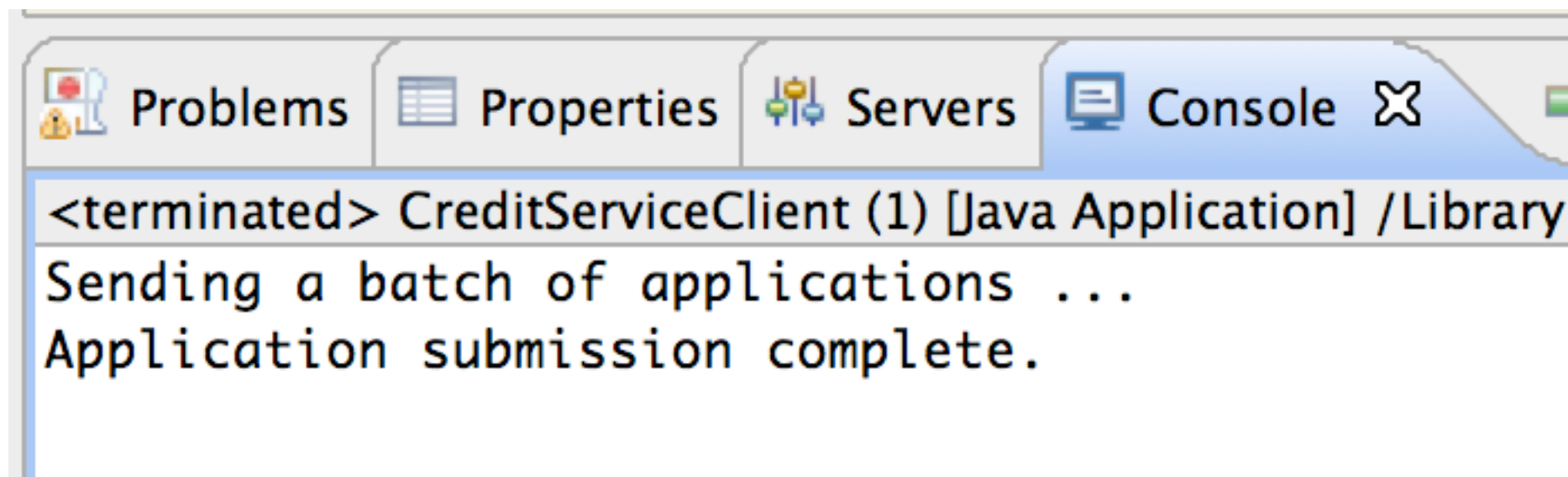
FYI

Once the test client completes its run, we are headed to the administration console to check out the lab 3 application.

TODO

1. Open up Firefox using the icon in the desktop menu bar.
2. Visit the SwitchYard section of the administration console. Here's a direct link to the page:

<http://localhost:9990/console/App.html#sy-apps>



Step 1

Cruise around the administration console and find some goodies

Application View

FYI

You will likely have more applications displayed in the list at this point in the lab.

TODO

1. Click on 'lab3' in the Applications table.

JBoss Application Server 7.2

(0) Messages ▾

Profile Runtime

Server

Overview

Manage Deployments

Status

Platform

JVM

Environment

Subsystems

Datasources

JPA

JMS Destinations

JNDI View

SwitchYard

Transactions

Web

Webservices

Runtime Operations

SwitchYard

Applications

Services

Artifacts

Applications

Displays a list of deployed SwitchYard applications. Select an application to see more details.

Applications

Name	Target Namespace
lab3	urn:lab3:1.0

1-1 of 1

Application Details

Displays details for a specific application. Select an application to see its implementation details.

Application Name:

Application Namespace:

Services

Artifact References

Transformers

Validators

Services

Name	Promoted Service
No Items!	

Application Details

FYI

If you scroll down, you'll find a view of the application metadata for the selected application.

TODO

1. Click on CreditService under the Services section.

Services

Artifact References

Transformers

Validators

Services

▲ Name	Promoted Service
CreditService	CreditService

1-1 of 1

Component Services

▲ Name	Interface	Implementation
CreditService	CreditService.wsdl#wsdl.porttype(CreditServicePortType)	View Details...
IncomeFix	mortgages.IncomeAdjustment	View Details...

1-2 of 2

Service View

FYI

This is a drill-down page for CreditService in lab3. Note you can view bindings for the service under the Gateways section.

TODO

1. Click on the soap binding for CreditService to view the binding's configuration.

Applications

Services

Artifacts

Services

Displays a list of deployed SwitchYard services. Select a service to see more details.

Services

Name	Target Namespace
CreditService	urn:lab3:1.0

1-1 of 1

Service Details

Service Name: CreditService Application: lab3

Service Namespace: urn:lab3:1.0

Implementation Details

Promoted Service: CreditService Interface: CreditService.wsdl#wSDL.porttype(Cre

Gateways

Type	Configuration
soap	View Configuration...

1-1 of 1



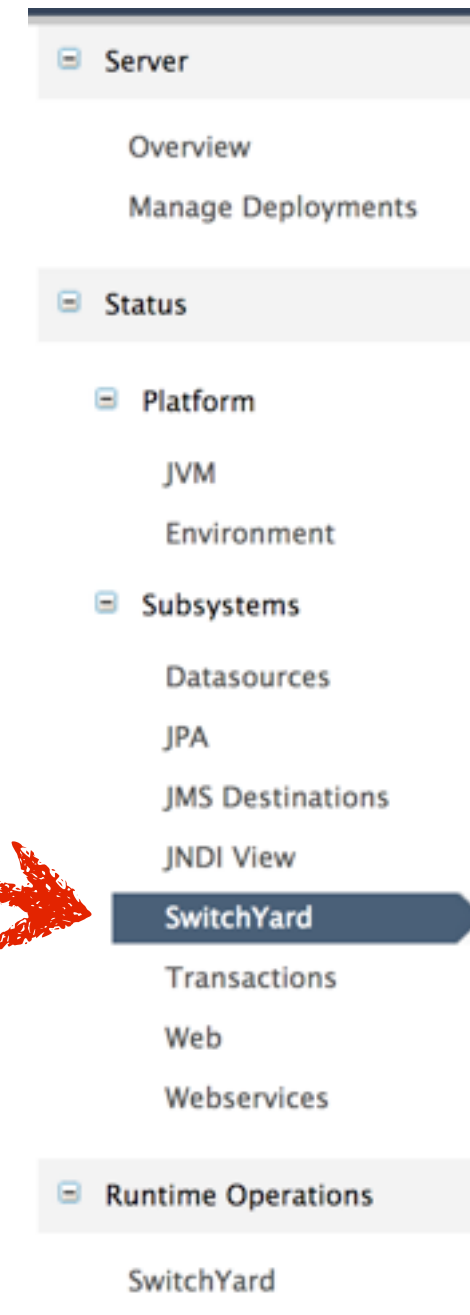
Monitoring

FYI

In addition to application metadata, service metrics are also available in the admin console.

TODO

1. Click on the SwitchYard entry in the Status section of the left frame of the console.



System Metrics

FYI

At the top of the Message Metrics page are metrics for the entire system.

TODO

1. Scroll down to the Service metrics section.

SwitchYard Message Metrics

System

Displays message metrics for the SwitchYard subsystem.

Message Counts

Metric	Actual	
Total Count:	100	
Success Count:	100	100%
Fault Count:	0	0%

Processing Times

Metric	Actual
Total Processing Time:	1250
Average Processing Time:	12.5
Min. Processing Time:	8
Max. Processing Time:	187

Service Metrics


FYI

Detailed metrics are available for each service in a SwitchYard runtime. Note that service metrics can be isolated by the operation invoked (Operation Metrics tab).

TODO

1. Click on CreditService under Service Metrics to see metrics for this service.
2. Click on the ReferenceMetrics tab to see metrics on services called from this service.

Service Metrics



Name	Target Namespace	Message Count	Average Time	▼ Time %	Fault %
► CreditService	urn:lab3:1.0	100	12.5	100%	0%

« 1-1 of 1 »

Service Metrics Operation Metrics Reference Metrics

Message Counts

Metric	Actual	
Total Count:	100	100%
Success Count:	100	100%
Fault Count:	0	0%

Processing Times

Metric	Actual	
Total Processing Time:	1250	100%
Average Processing Time:	12.5	
Min. Processing Time:	8	
Max. Processing Time:	187	

Reference Metrics

FYI

Reference metrics allow you to view the execution time for every service invoked from a given service. This can be extremely useful when debugging things like SLA violations where your service execution time is slow but the actual root cause is in a downstream service.

The screenshot below shows that for 100 invocations of CreditService, FancyCredit was invoked every time and accounted for 48% of total execution time of the service. IncomeFix was invoked 50 times since the routing to IncomeFix was conditional on income not being specified in an application.



Service Metrics

Operation Metrics

Reference Metrics

Referenced Service Metrics

▲ Name	Message Count	Average Time	Time %	Fault %
FancyCredit	100	5.98	48%	0%
IncomeFix	50	1.8	7%	0%

Step 2

Introduce message tracing to diagnose problems in poorly behaved applications

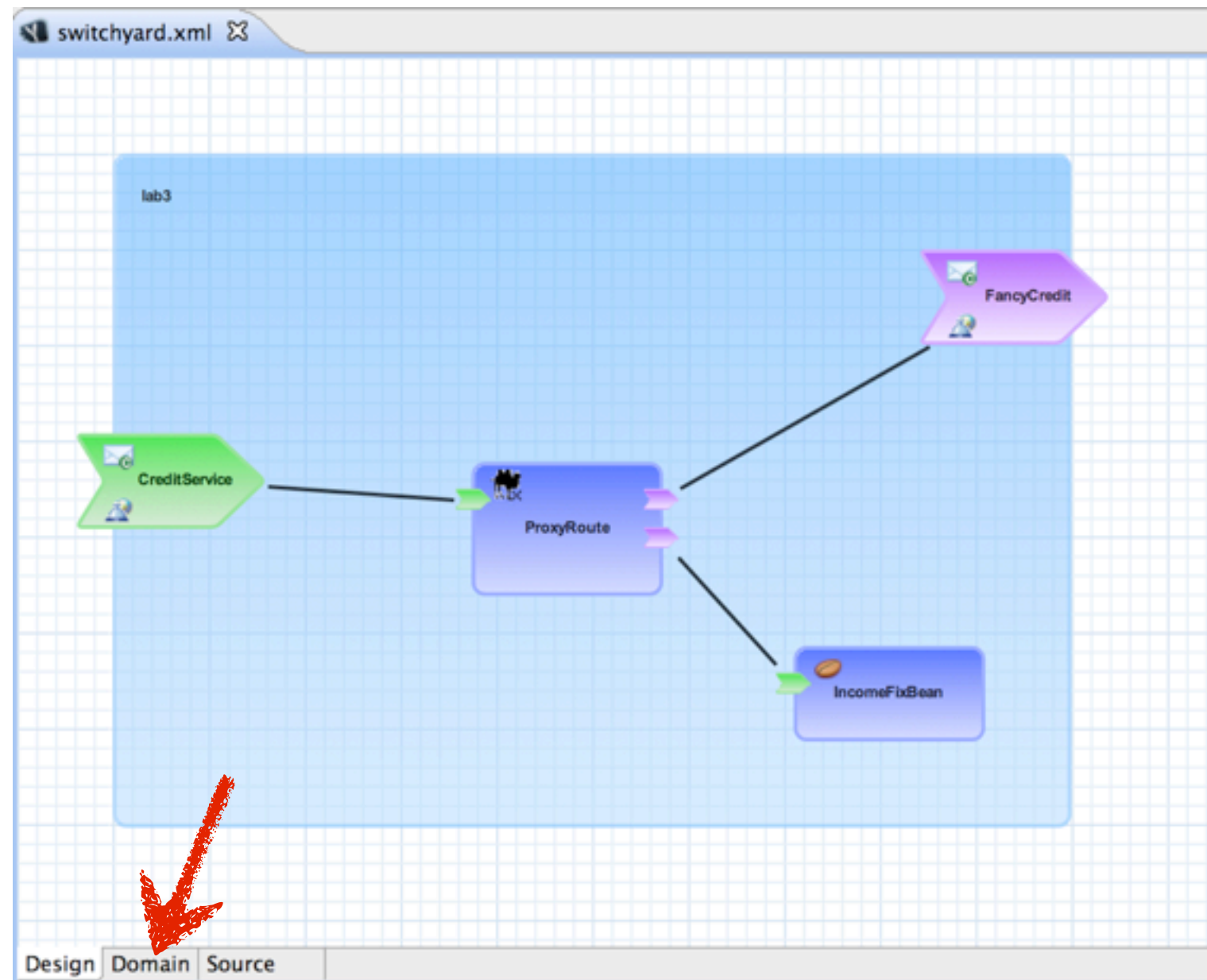
Enabling Message Tracing

FYI

Message Tracing is a configurable option for applications that provides visibility into messages as they are routed on the bus.

TODO

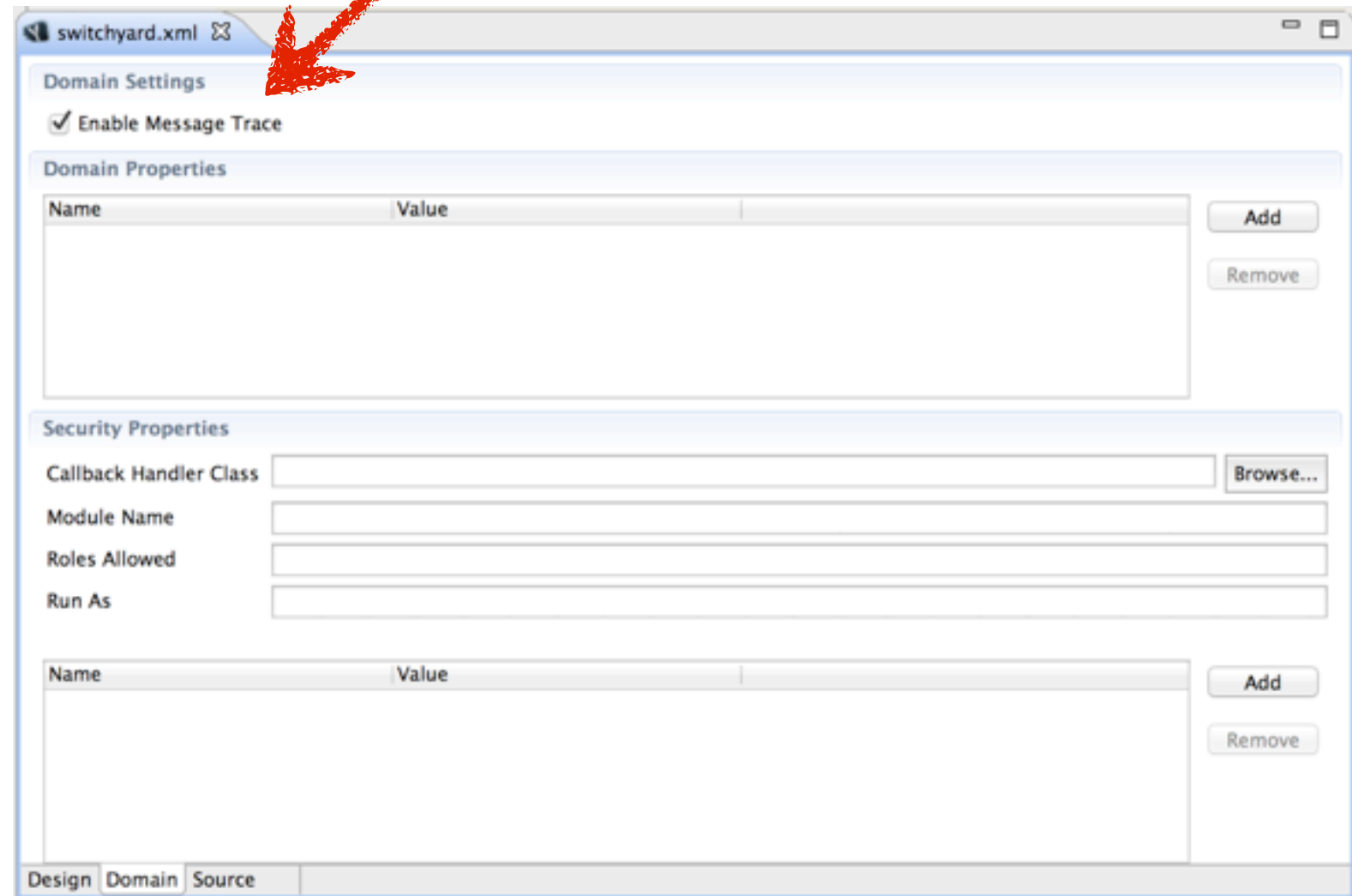
1. Open up the visual editor for **lab3**.
2. Click on the Domain tab.



Enabling Message Tracing

TODO

1. Check the box next to 'Enable Message Trace' at the top of the editor.
2. File -> Save.



The screenshot shows the 'switchyard.xml' editor window. The 'Domain Settings' tab is active, and the 'Enable Message Trace' checkbox is checked. A red arrow points to this checkbox. Below the settings are sections for 'Domain Properties' and 'Security Properties', each with a table for Name and Value, and buttons for Add and Remove. The 'Security Properties' section also includes fields for Callback Handler Class, Module Name, Roles Allowed, and Run As, with a Browse... button next to the Callback Handler Class field. The bottom of the window has tabs for Design, Domain, and Source.

Name	Value
------	-------

Add Remove

Security Properties

Callback Handler Class Browse...

Module Name

Roles Allowed

Run As

Name	Value
------	-------

Add Remove

Design Domain Source

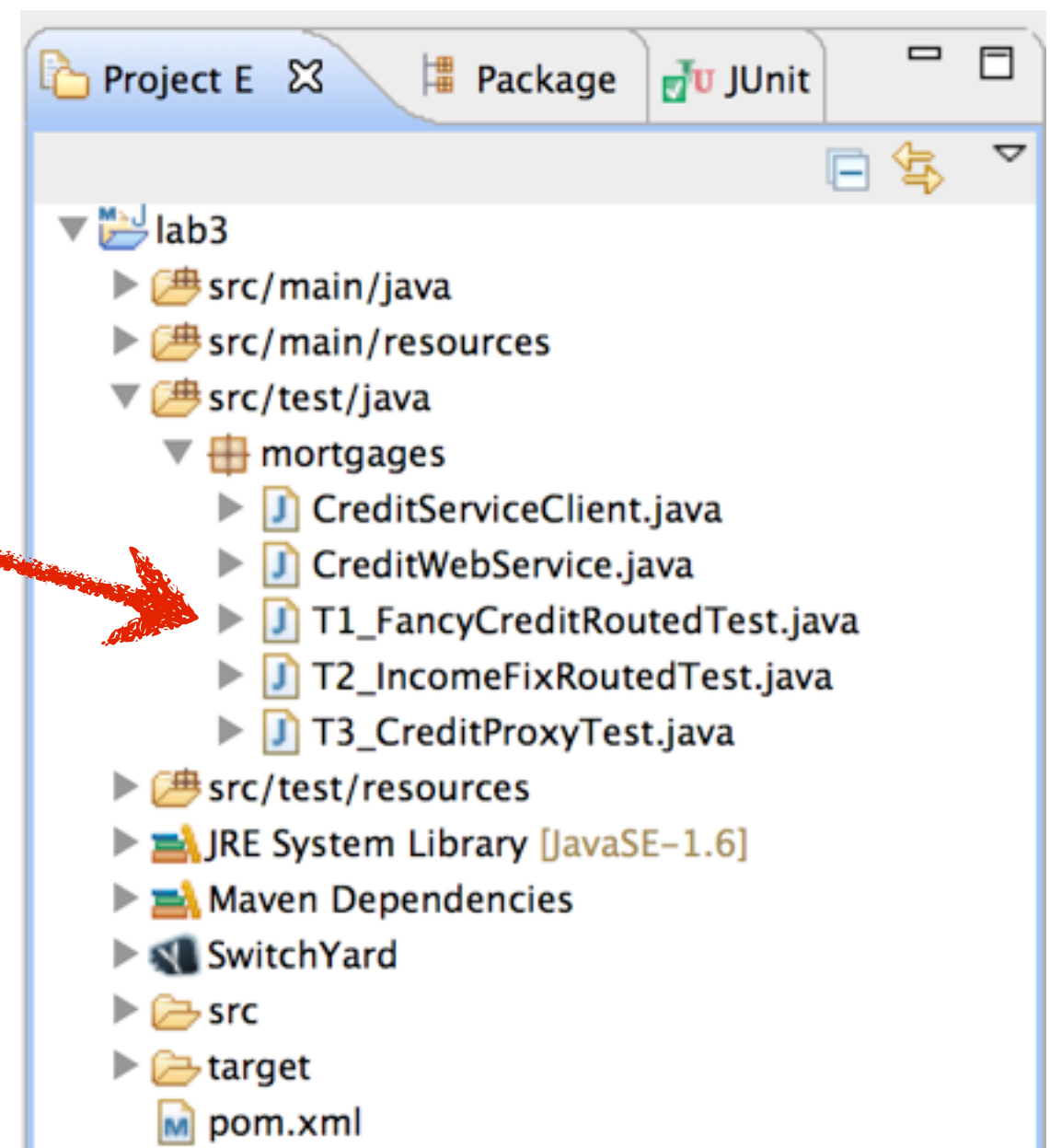
Tracing In Action

FYI

MessageTracing works in unit tests and inside a deployed SwitchYard runtime. Let's see what it looks like in a test.

TODO

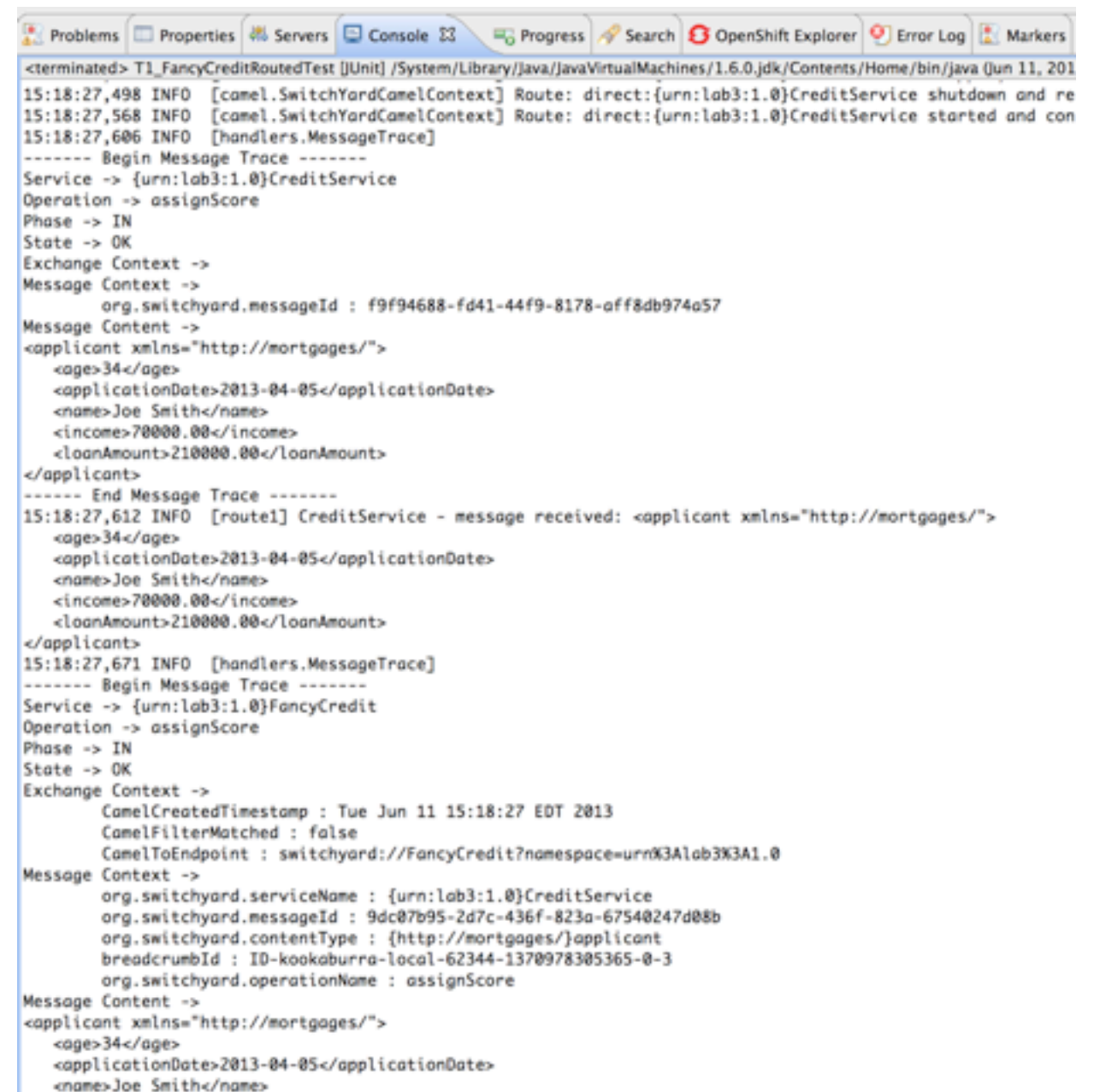
1. Double-click on T1_FancyCreditRoutedTest in the explorer to open the unit test.
2. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.



Tracing In Action

FYI

The trace data can be verbose, so you may want to expand the Console view to take it all in. These details can be a lifesaver when something is going wrong and you want a better idea of what's happening "on the bus".



```
<terminated> T1_FancyCreditRoutedTest [JUnit] /System/Library/Java/JavaVirtualMachines/1.6.0.jdk/Contents/Home/bin/java (Jun 11, 2013 15:18:27,498 INFO [camel.SwitchYardCamelContext] Route: direct:{urn:lab3:1.0}CreditService shutdown and re
15:18:27,568 INFO [camel.SwitchYardCamelContext] Route: direct:{urn:lab3:1.0}CreditService started and con
15:18:27,606 INFO [handlers.MessageTrace]
----- Begin Message Trace -----
Service -> {urn:lab3:1.0}CreditService
Operation -> assignScore
Phase -> IN
State -> OK
Exchange Context ->
Message Context ->
    org.switchyard.messageId : f9f94688-fd41-44f9-8178-aff8db974a57
Message Content ->
<applicant xmlns="http://mortgages/">
  <age>34</age>
  <applicationDate>2013-04-05</applicationDate>
  <name>Joe Smith</name>
  <income>70000.00</income>
  <loanAmount>210000.00</loanAmount>
</applicant>
----- End Message Trace -----
15:18:27,612 INFO [route1] CreditService - message received: <applicant xmlns="http://mortgages/">
  <age>34</age>
  <applicationDate>2013-04-05</applicationDate>
  <name>Joe Smith</name>
  <income>70000.00</income>
  <loanAmount>210000.00</loanAmount>
</applicant>
15:18:27,671 INFO [handlers.MessageTrace]
----- Begin Message Trace -----
Service -> {urn:lab3:1.0}FancyCredit
Operation -> assignScore
Phase -> IN
State -> OK
Exchange Context ->
    CamelCreatedTimestamp : Tue Jun 11 15:18:27 EDT 2013
    CamelFilterMatched : false
    CamelToEndpoint : switchyard://FancyCredit?namespace=urn:lab3:1.0
Message Context ->
    org.switchyard.serviceName : {urn:lab3:1.0}CreditService
    org.switchyard.messageId : 9dc07b95-2d7c-436f-823a-67540247d08b
    org.switchyard.contentType : {http://mortgages/}applicant
    breadcrumbId : ID-kookaburra-local-62344-1370978305365-0-3
    org.switchyard.operationName : assignScore
Message Content ->
<applicant xmlns="http://mortgages/">
  <age>34</age>
  <applicationDate>2013-04-05</applicationDate>
  <name>Joe Smith</name>
```

Step 3

Introduce message tracing to diagnose problems in poorly behaved applications

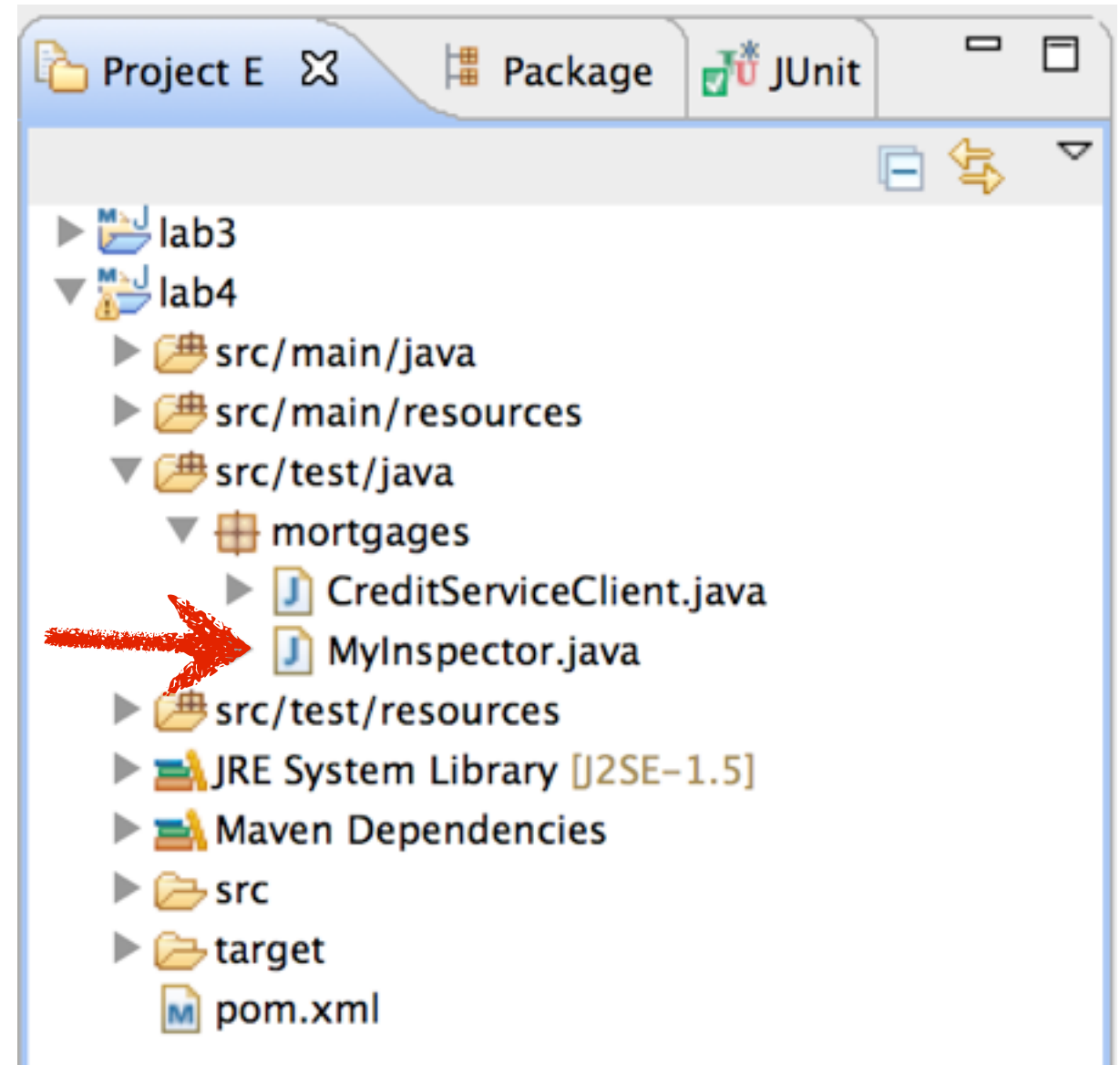
Auditor Unleashed!

FYI

Under the covers, every message exchange in SwitchYard happens over an internal Camel bus. Service execution is a route with a set of runtime processors that handle things like policy, transformation, validation, etc. Auditors allow you to inject code to surround each of these processors at runtime. This is a serious debugging tool and is only recommended for crazy situations and for testing only. Also, if asked, you didn't hear about this from me.

TODO

1. Find MyInspector.java in src/test/java directory of lab4.
2. Copy MyInspector to the src/test/java directory of lab3.



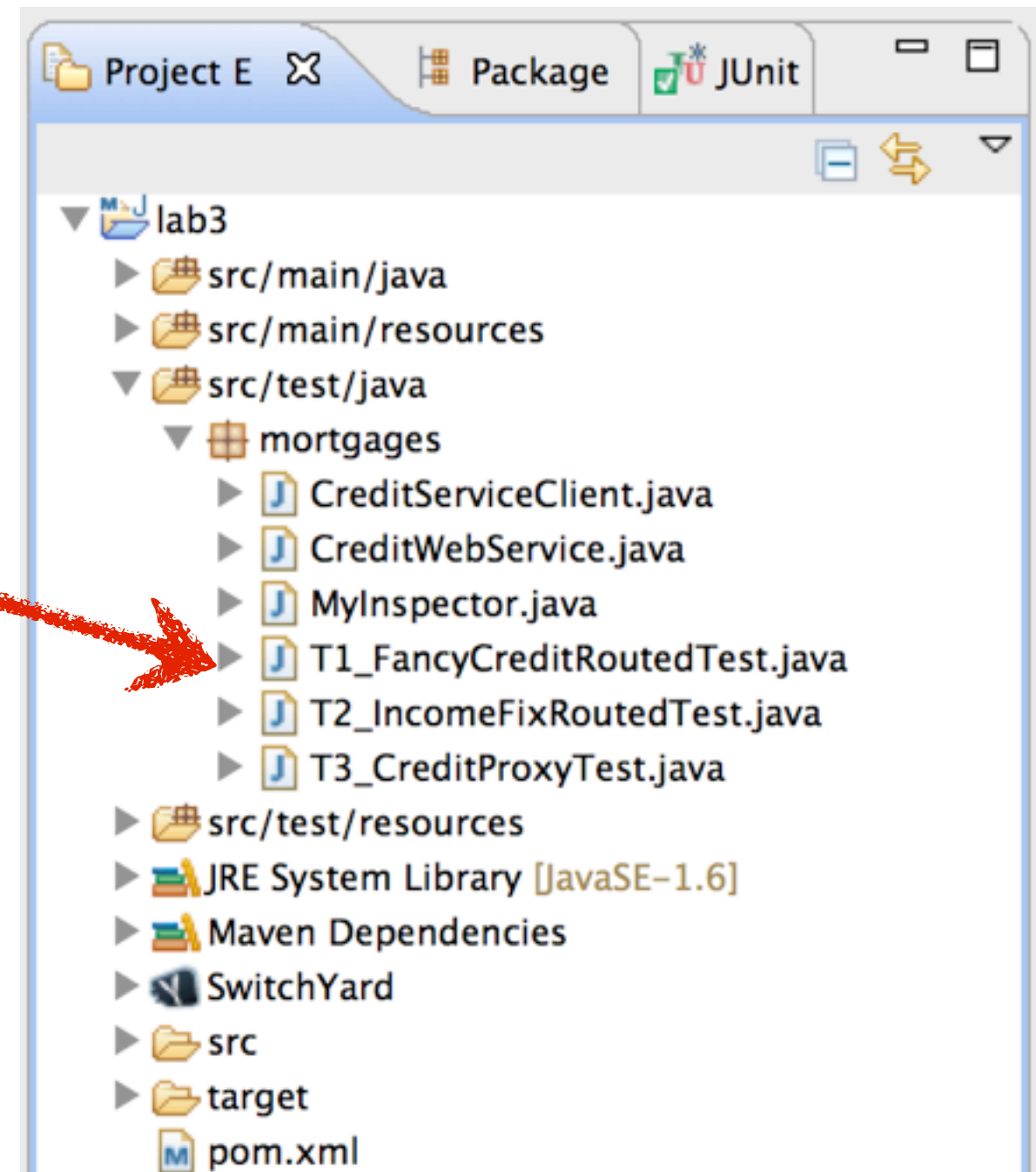
Auditing In Action

FYI

The SwitchYard deployer automatically picks up Inspectors that it finds in the classpath. Note that we put it in src/test/java so that it's not packaged with the application.

TODO

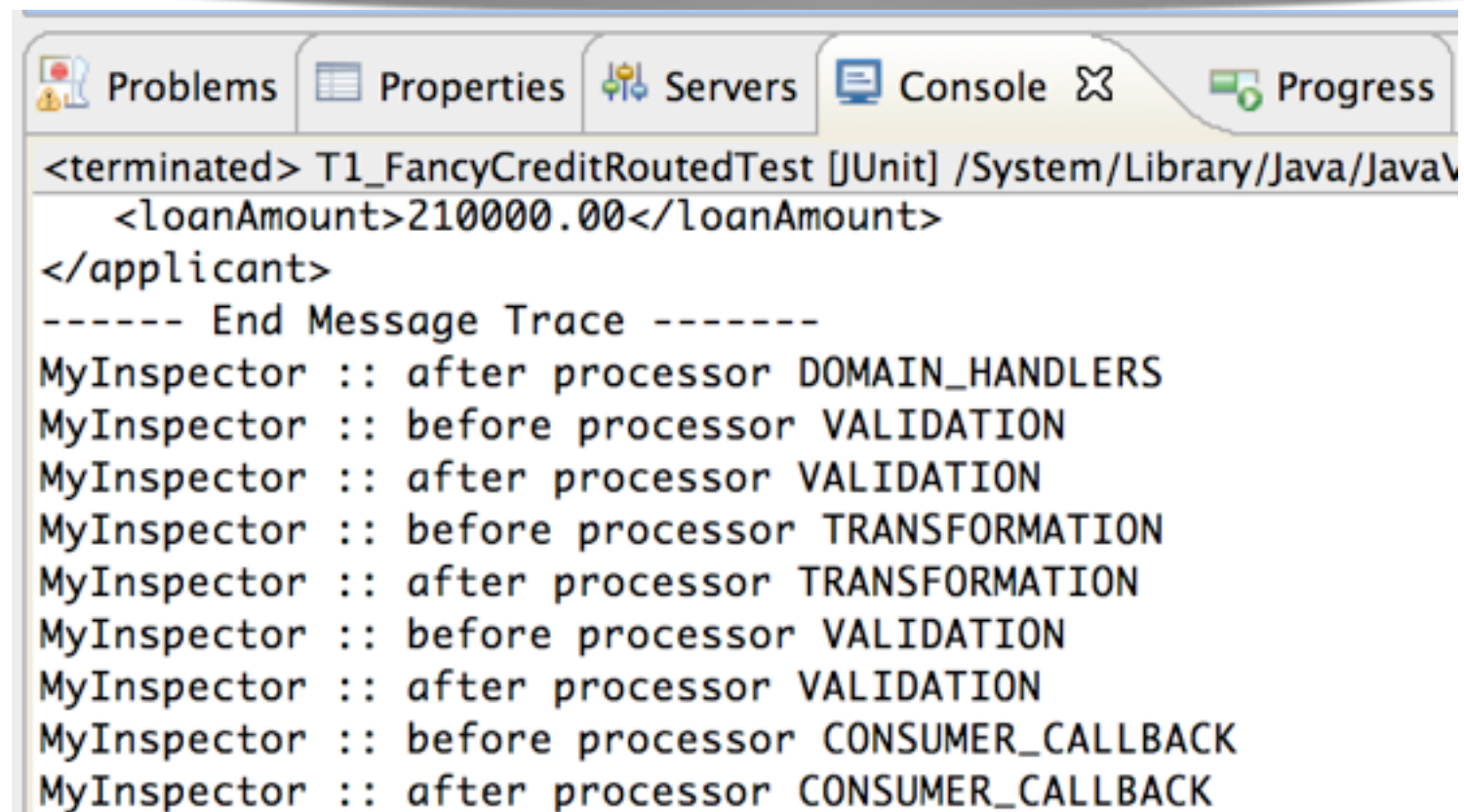
1. Double-click on T1_FancyCreditRoutedTest in the explorer to open the unit test.
2. Go to the Run menu in the main menu bar and select 'Run As -> JUnit Test' to run the unit test.



Auditing In Action

FYI

The trace data can be verbose, so you may want to expand the Console view to take it all in. These details can be a lifesaver when something is going wrong and you want a better idea of what's happening "on the bus".



The screenshot shows the Eclipse IDE's Console view. The top toolbar includes icons for Problems, Properties, Servers, Console (which is active), and Progress. The console output displays a terminated test run for T1_FancyCreditRoutedTest [JUnit] in the System/Library/Java/JavaV directory. It shows an XML-like structure with <loanAmount>210000.00</loanAmount> and </applicant>. Below this, a message trace is shown, starting with "----- End Message Trace -----". The trace consists of several lines of log messages from MyInspector, detailing the execution flow through various processors: DOMAIN_HANDLERS, VALIDATION, TRANSFORMATION, and CONSUMER_CALLBACK, with "before" and "after" markers for each.

```
<terminated> T1_FancyCreditRoutedTest [JUnit] /System/Library/Java/JavaV
  <loanAmount>210000.00</loanAmount>
</applicant>
----- End Message Trace -----
MyInspector :: after processor DOMAIN_HANDLERS
MyInspector :: before processor VALIDATION
MyInspector :: after processor VALIDATION
MyInspector :: before processor TRANSFORMATION
MyInspector :: after processor TRANSFORMATION
MyInspector :: before processor VALIDATION
MyInspector :: after processor VALIDATION
MyInspector :: before processor CONSUMER_CALLBACK
MyInspector :: after processor CONSUMER_CALLBACK
```

Lab 4 Complete!