Lab Exercise

Enrichers - Lab 6

An enricher adds or enhances a Spring Integration (SI) message with more information or data. Some integration path processes may have to calculate or process some data in order to augment the message with data needed on the receiving end. Some messages may need to be augmented with data from a database. Whatever the objectives, the SI enricher is used to “enrich” the message with more information.

As with most of the SI componentry, there are several built-in enrichers and you can build a custom enricher. In this lab, you see an enricher mark a ship order as being shipped, and you’ll also see an enricher calculate the total price for an order. The later requires some customization.

Specifically, in this lab you will:

- Revisit Spring XML to object message transformation.
- Configure a SI enricher to enrich Shiporder objects with additional data.
- Enrich a Shiporder object with a boolean property indicating the order has shipped.
- Build and use a service activator to help the enricher calculate a total order value of a Shiporder.

Lab solution folder: ExpressSpringIntegration\lab6\lab6-enricher-solution

Scenario – Enrich the Shiporder object

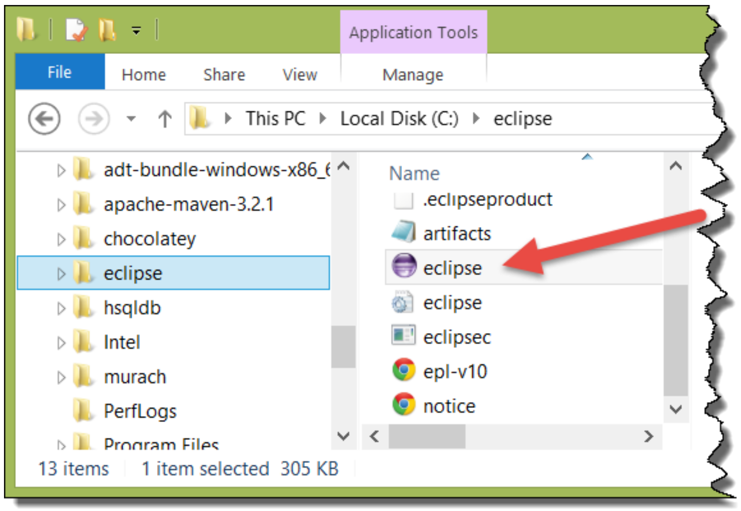
Step 1: Import the Maven Project

A Maven Project containing the base code for a message transforming and enriching application has already been created for you. You will use this project to begin your exploration of enrichers.

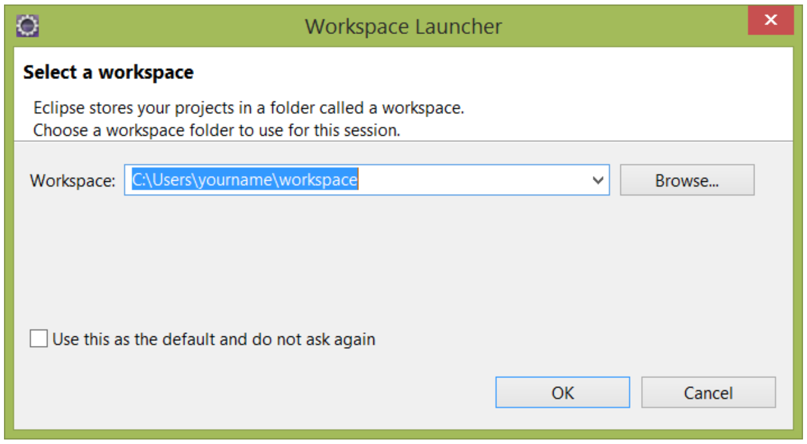
1.1 Start the Eclipse-based IDE. Locate and start Eclipse (or Eclipse-based) IDE.

1.1.1 Locate the Eclipse folder.

1.1.2 Start Eclipse by double clicking on the eclipse.exe icon (as highlighted in the image below).



1.1.3 Open your workspace. Type in *C:\Users\<your username >\workspace* and click the *OK* button.


Workspace Launcher

Select a workspace

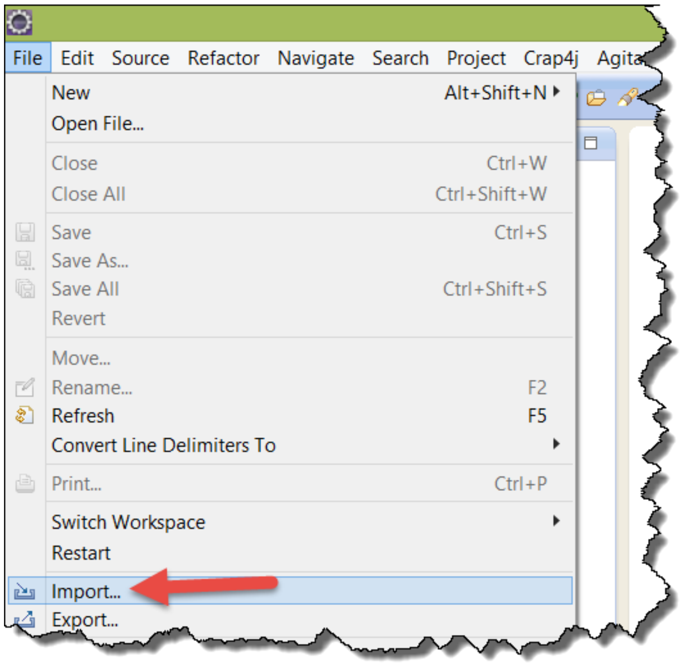
Eclipse stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: C:\Users\yourname\workspace

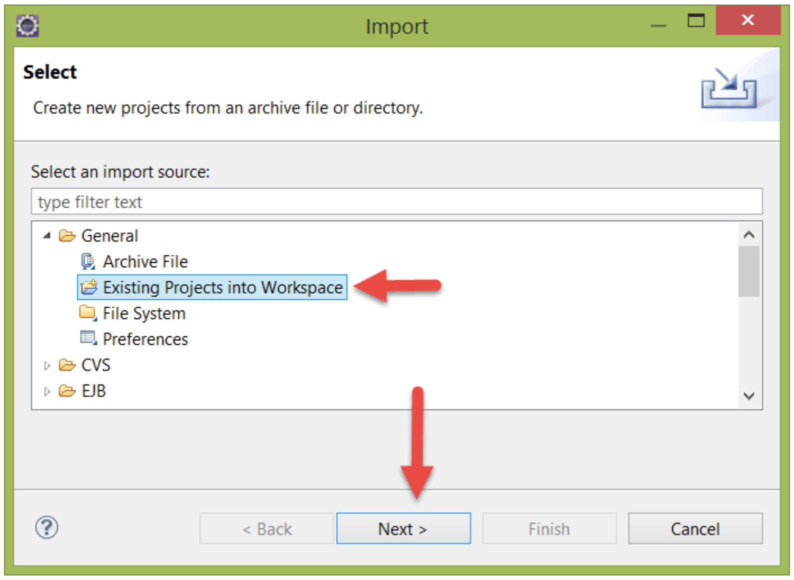
Note: As noted in the past labs, you may use an alternate location for your workspace, but the labs will always reference this location (c:\users\[your username]\workspace) as the default workspace location.

1.2 Import the new Maven Project.

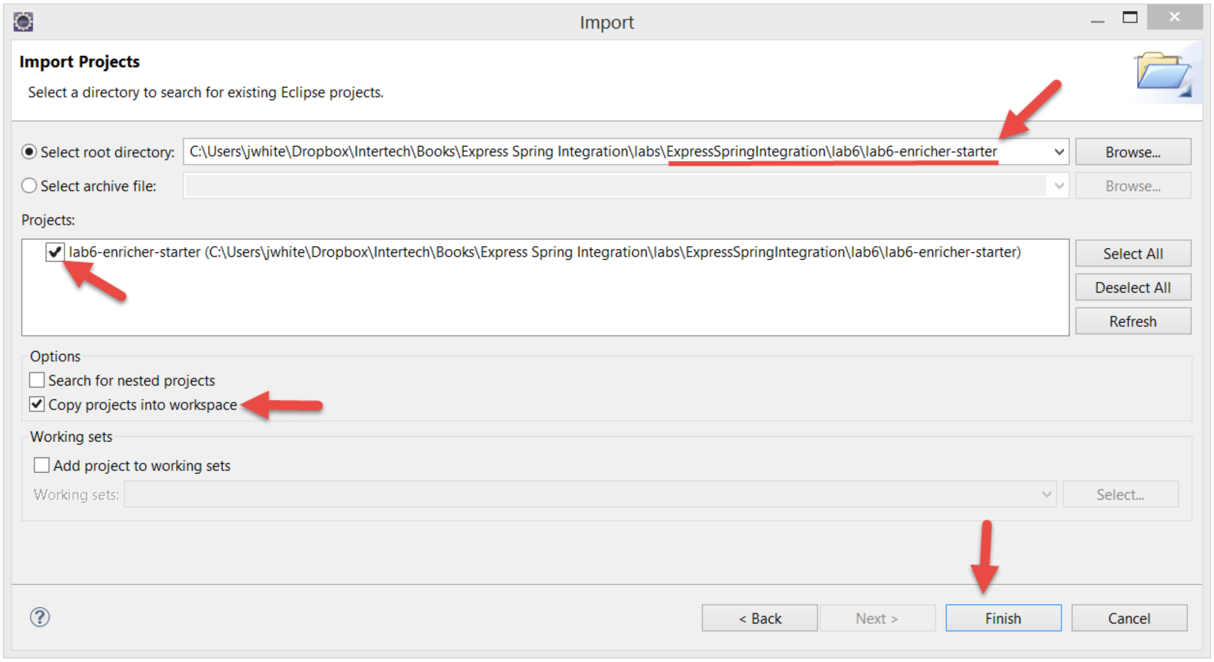
1.2.1 Select File>Import... from the Eclipse menu bar.



1.2.2 Locate and open the General folder in the Import window, and then select *Existing Projects into Workspace Project* from the options. Now push the *Next>* button.



1.2.3 In the "Import" window, use the *Browse…* button to locate the lab6-enricher-starter project folder located in ExpressSpringIntegration\lab6 (this folder is located in the lab downloads). Make sure the project is selected, and the *"Copy projects into workspace"* checkbox is also checked before you hit the *Finish* button (as shown below).



(●) Select root directory: C:\Users\jwhite\Dropbox\Intertech\Books\Express Spring Integration\labs\ExpressSpringIntegration\lab6\lab6-enricher-starter

( ) Select archive file:

Projects:

[✓] lab6-enricher-starter (C:\Users\jwhite\Dropbox\Intertech\Books\Express Spring Integration\labs\ExpressSpringIntegration\lab6\lab6-enricher-starter)

[ ] Search for nested projects

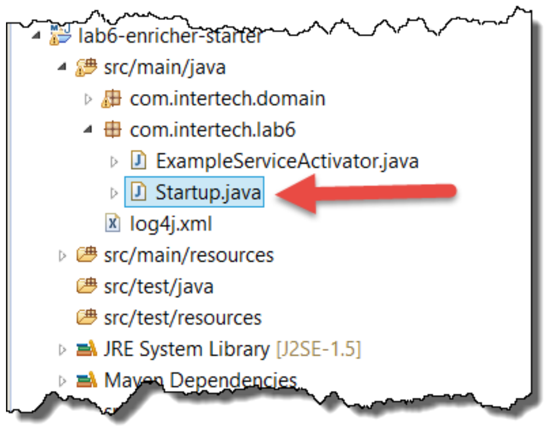
[✓] Copy projects into workspace

[ ] Add project to working sets

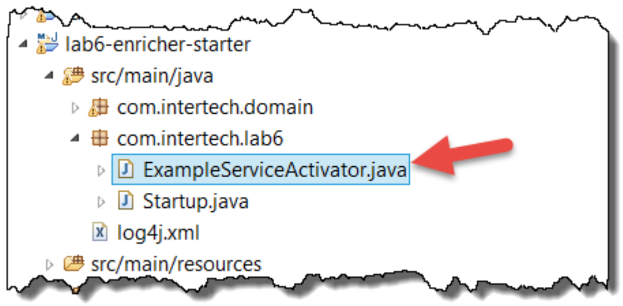
Working sets:

1.3 Explore the project. Examine the project for the Spring Integration components that are already present.

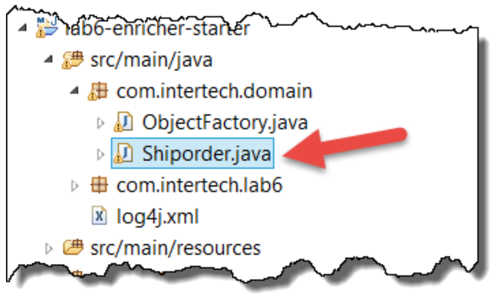
1.3.1 Examine the Startup.java. Expand the src/main/java folder in the Project Explorer view, and open the Startup.java file by double clicking on it. As in past labs, the Startup.java class is used to put the application in an infinite loop so that the SI components can do their work.



1.3.1 Note the presence of ExampleServiceActivator. Again, its job will be to display the contents of a SI message to the Console view.

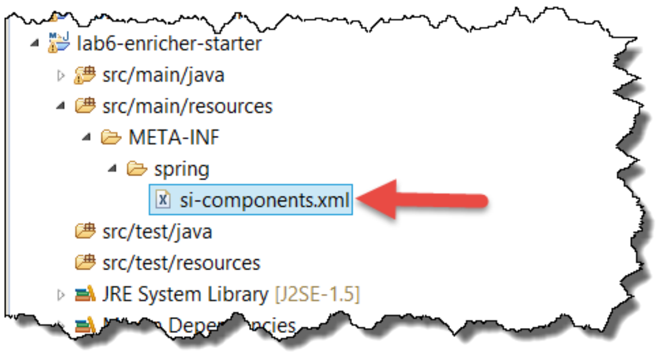


1.3.2 Locate and open the Shiporder.java class found in the com.intertech.domain package in src/main/java folder by double clicking on the file in the Project Explorer.

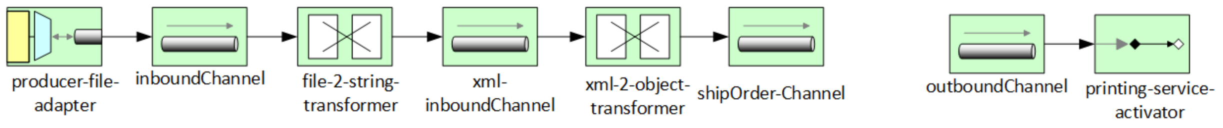


As you learned earlier in this tutorial class, there are three classes defined in this file: Shiporder and two inner classes Item and Shipto. XML messages will again be read and transformed into Shiporder objects with this lab. In particular, the Shiporder objects will be enriched with additional data – the total cost of the order based on the price and quantity for each item and the fact that each order has been shipped.

1.3.3 Examine the SI configuration. Expand the src/main/resources/META-INF/spring folder in the Project Explorer view, and open si-components.xml file by double clicking on it. The si-components.xml file contains the Spring configuration that includes the definitions for several SI components already.



1.3.4 In particular, note that the application already contains an inbound file adapter and a file-to-string transformer to take XML files and get them to String form. It also includes an unmarshalling transformer to turn the XML messages into Shiporder object payload messages (just like the work you completed in Lab 4). Below is an EIP model representing what is currently defined in the si-components.xml file. The adapter gets XML files to the inboundChannel and then transformers convert the file messages to string messages first and Shiporder object payload messages second. A disconnected outboundChannel and printing SA are also provided.



Step 2: Create an Enricher

Uses a built-in SI enricher to mark each Shiporder message as “shipped.” Shipped is represented by a simple boolean instance variable in the Shiporder payload object (true for shipped, false for not shipped).

2.1 Add a SI enricher component.

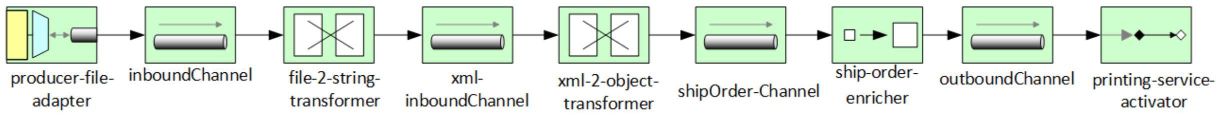
2.1.1 In the same si-components.xml file, add a SI enricher that sets the shipped property of the Shiporder object payload to the value of true. Note: if you examine Shiporder.java, you will see the value of the shipped property has a value of false by default. The enricher should take messages from the shipOrder-Channel, enrich them, and deposit the enriched messages to the outboundChannel.

```
<int:enricher id="ship-order-enricher" input-channel="shipOrder-
Channel"
  output-channel="outboundChannel">
    <int:property name="shipped" expression="true"/>
</int:enricher>
```

Note: if you get stuck or feel like not typing in all the code yourself, you will find a working copy of the final si-component.xml file at ExpressSpringIntegration\lab6\lab6-enricher-solution.

2.1.2 With the addition of this enricher component, you have linked the inbound and outbound channels.

2.1.3 Wolfe and Hohpe call this type of component a “content enricher.” It is represented by the icon above the “ship-order-enricher” text below.



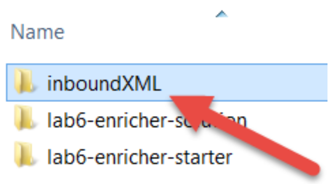
2.2 Save the configuration file and make sure there are no errors in the file.

Step 3: Test the Enricher

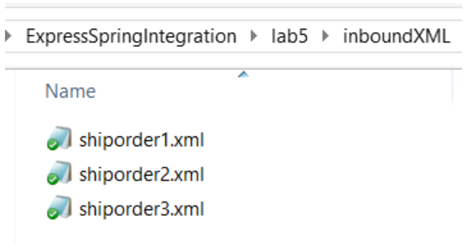
3.1 Add the files to the inbound message folder.

3.1.1 In the si-components.xml, find the producer-file-adapter. Note the location of the directory. It is set, by default, to file:c://inboundXML. This is the location where the XML ship order messages will be taken into the application by the adapter. It is the same folder used in the last lab (lab 5). If it does not already exist, create this message folder - changing the location to suit your needs and your file system (change the producer-file-adapter to reflect your location).

3.1.2 Again, some sample XML ship order messages have been provided to you. Find them in an inboundXML folder in ExpressSpringIntegration\lab6.



Copy the messages from this folder to the c:\\inboundXML folder (or whatever folder you created per 3.1.1. above).



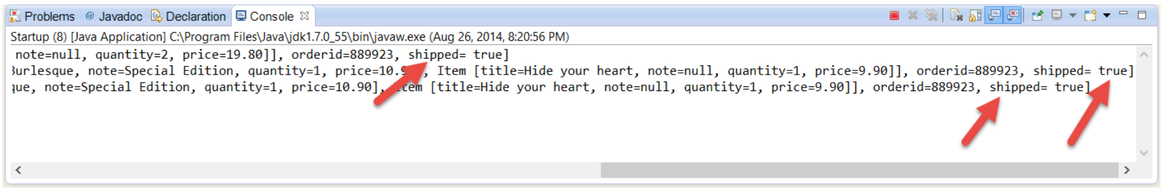
Note: the messages and locations did not change for the last lab. You can use the same messages if they remain from the last lab (lab 5) without the need to copy anew.

3.2 Test the application. Test the application to see the Files in the inbound file folder are enriched – specifically test that each Shiporder object's shipped instance variable is set to true.

3.2.1 Locate the Startup.java file in the source folder. Right click on file and select Run As > Java Application from the resulting menu. Nothing should display in the Console view.

Note: depending on the speed of your system, you may have to give the application a couple of seconds to complete its work.

3.3 Check the Console view for results. The ExampleServiceActivator takes each message from the outboundChannel and completes a toString( ) call to the object in the message payload after performing the enrichment. This causes the object's contents to be displayed in the Console view. Note that the value for the shipped property for each message is "true".



```
Startup (8) [Java Application] C:\Program Files\Java\jdk1.7.0_55\bin\javaw.exe (Aug 26, 2014, 8:20:56 PM)
 note=null, quantity=2, price=19.80]], orderid=889923, shipped= true]
Burlesque, note=Special Edition, quantity=1, price=10.9[illegible], Item [title=Hide your heart, note=null, quantity=1, price=9.90]], orderid=889923, shipped= true]
que, note=Special Edition, quantity=1, price=10.90], Item [title=Hide your heart, note=null, quantity=1, price=9.90]], orderid=889923, shipped= true]
```

3.4 Stop the application. Recall the application is running in an infinite loop to allow for the constant publishing and consuming of messages. Stop the application now.

3.4.1 In the Console view, click on the red square to terminate the Startup application.

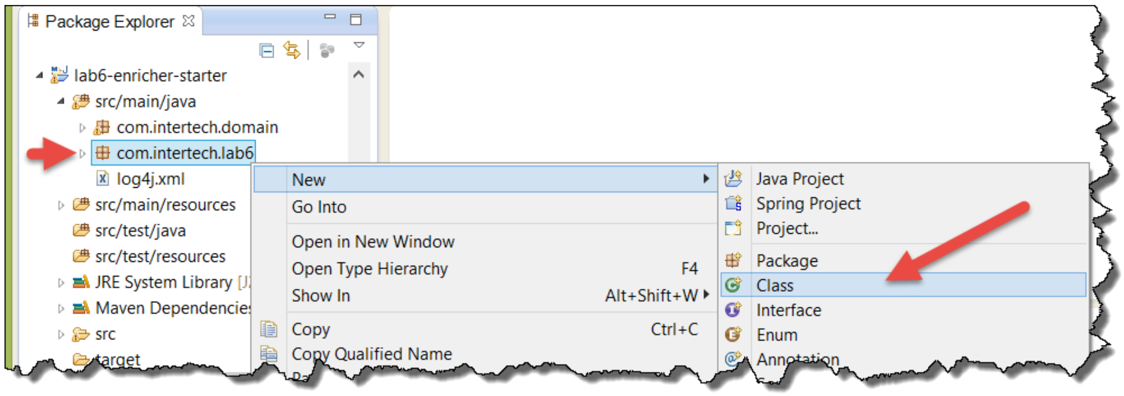
3.4.2 The Console view should now indicate that the application is "<terminated>".

Step 4: Create an Enriching Service Activator

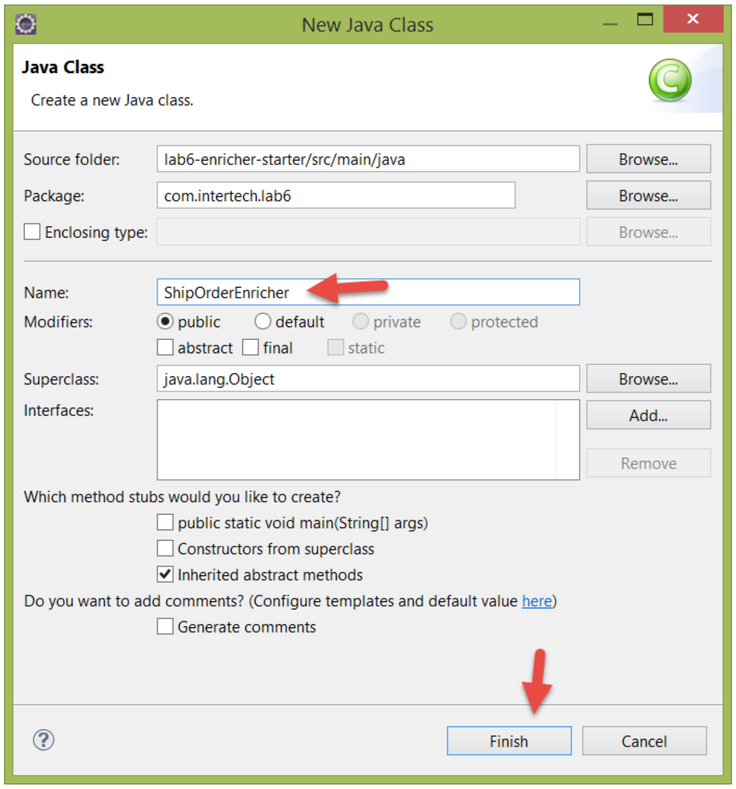
While built-in capability for completing simple enrichment of messages is provided by SI, more complex enrichment of either the payload or header, may require customization. Specifically, it may require sending the message to a special service activator component from the enricher that provides information back to the enricher to complete its message augmenting work. In this step, you create a service activator to calculate the cost of the entire Shiporder (based on the cost and quantity associated with each of the associated Item objects in the Shiporder). The service activator provides the total cost back to the enricher so that it can update a property of the Shiporder object.

4.1 Create the enriching service activator.

4.1.1 Right click on the com.intertech.lab6 package of the project in the Package Explorer view and select New > Class.



4.1.2 In the New Java Class window that opens, enter ShipOrderEnricher in the Name field and then press the Finish button.



Source folder: lab6-enricher-starter/src/main/java
Package: com.intertech.lab6
☐ Enclosing type:
Name: ShipOrderEnricher
Modifiers: ● public ○ default ○ private ○ protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
☐ public static void main(String[] args)
☐ Constructors from superclass
☑ Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
☐ Generate comments

The class should be created and an editor opened for code entry for the new class.

4.1.3 Create a computeTotal( ) method. This method should receive a Message<Shiporder> as a parameter and return a double – the total cost of all items in the Shiporder. The code for this method is below.

```
public double computeTotal(Message<Shiporder> order) {
  double sum = 0;
  for (Item item : order.getPayload().getItem()) {
    sum = sum + (item.getPrice().doubleValue() *
      item.getQuantity().intValue()) ;
  }
  return sum;
 }
```

Note: if you get stuck or feel like not typing in all the code yourself, you will find a working copy of the final ShipOrderEnricher class at ExpressSpringIntegration\lab6\lab6-enricher-solution.

4.1.4 The code added, will require imports. Add the following imports to the top of the class.

```
import org.springframework.messaging.Message;
import com.intertech.domain.Shiporder;
import com.intertech.domain.Shiporder.Item;
```

4.1.5 Save the class and make sure there are no compile errors.

4.2 Add the ShipOrderEnricher bean to the Spring configuration.

4.2.1 If not already open, expand the src/main/resources/META-INF/spring folder in the Project Explorer view, and open si-components.xml file by double clicking on it.

4.2.2 Add a Spring bean of the ShipOrderEnricher type, as shown below, into the configuration.

```
<bean id="shipOrderEnricher"
  class="com.intertech.lab6.ShipOrderEnricher"/>
```

4.2.3 Add a new channel and service activator SI component to the configuration. The enricher will pass all messages to be enriched to the new service activator through the total-price-enricher-channel. The SI bean (above) will calculate the total price of the order and pass the resulting total (a double) back to the enricher through a built-in default channel.

```
<int:channel id="total-price-enricher-channel" />
<int:service-activator id="enriching-service-activator"
  ref="shipOrderEnricher" input-channel="total-price-enricher-
channel"/>
```

4.3 Update the enricher to use the service activator to enrich the Shiporder with the total price for the order.

4.3.1 Add a request-channel attribute to the enricher. Specify the new total-price-enricher-channel as the request channel. This tells the enricher to send all "to-be-enriched" messages to the service activator through this channel before completing its enrichment duties.

```
<int:enricher id="ship-order-enricher" input-channel="shipOrder-
Channel"
    output-channel="outboundChannel"
    request-channel="total-price-enricher-channel">
  <int:property name="shipped" expression="true"/>
</int:enricher>
```

4.3.2 Add an additional property element to the enricher. The property should cause the orderTotal property on the Shiporder to be updated or enriched by the payload of the return message from the service activator.

```
<int:enricher id="ship-order-enricher" input-channel="shipOrder-
Channel"
    output-channel="outboundChannel"
    request-channel="total-price-enricher-channel">
  <int:property name="orderTotal" expression="payload" />
  <int:property name="shipped" expression="true"/>
</int:enricher>
```

4.3.3 Save the configuration file and make sure there are no errors in the file.

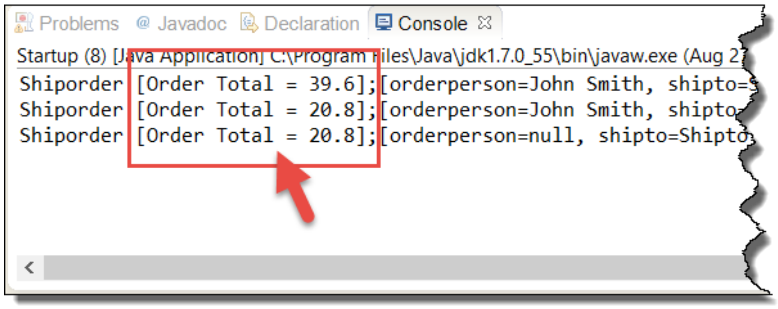
Step 5: Retest the Enricher

5.1 Test the application. Retest the application to see the Files in the inbound file folder are enriched – this time to see them enriched with the order total as calculated by the service activator.

5.1.1 Locate the Startup.java file in the source folder. Right click on file and select Run As > Java Application from the resulting menu. Nothing should display in the Console view.

Note: depending on the speed of your system, you may have to give the application a couple of seconds to complete its work.

5.1.2 Check the Console view for results. The ExampleServiceActivator takes each message from the outboundChannel and completes a toString( ) call to the object in the message payload after performing the enrichment. This causes the object's contents to be displayed in the Console view. Note the order total near the beginning of the output for each Shiporder (see below).



```
Startup (8) [Java Application] C:\Program Files\Java\jdk1.7.0_55\bin\javaw.exe (Aug 2
Shiporder [Order Total = 39.6];[orderperson=John Smith, shipto=
Shiporder [Order Total = 20.8];[orderperson=John Smith, shipto
Shiporder [Order Total = 20.8];[orderperson=null, shipto=Shipto
```

5.2 Stop the application. Recall the application is running in an infinite loop to allow for the constant publishing and consuming of messages. Stop the application now.

5.2.1 In the Console view, click on the red square to terminate the Startup application.

The Console view should now indicate that the application is "<terminated>".

Lab Solution

si-components.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:int="http://www.springframework.org/schema/integration"
  xmlns:int-
file="http://www.springframework.org/schema/integration/file"
  xmlns:int-
mail="http://www.springframework.org/schema/integration/mail"
  xmlns:int-
xml="http://www.springframework.org/schema/integration/xml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:int-
stream="http://www.springframework.org/schema/integration/stream"
  xsi:schemaLocation="
  http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/integration
http://www.springframework.org/schema/integration/spring-
integration.xsd
  http://www.springframework.org/schema/integration/stream
http://www.springframework.org/schema/integration/stream/spring-
integration-stream.xsd
  http://www.springframework.org/schema/integration/file
http://www.springframework.org/schema/integration/file/spring-
integration-file.xsd
  http://www.springframework.org/schema/integration/xml
http://www.springframework.org/schema/integration/xml/spring-
integration-xml.xsd">

  <!-- Adapter for reading files -->

  <int-file:inbound-channel-adapter id="producer-file-adapter"
    channel="inboundChannel" directory="file:c://inboundXML"
    prevent-duplicates="true">
    <int:poller fixed-rate="5000" />
  </int-file:inbound-channel-adapter>

  <int:channel id="inboundChannel" />

  <int-file:file-to-string-transformer
    id="file-2-string-transformer" input-channel="inboundChannel"
    output-channel="xml-inboundChannel" charset="UTF-8" />

  <int:channel id="xml-inboundChannel" />

  <int-xml:unmarshalling-transformer
    id="xml-2-object-transformer" input-channel="xml-inboundChannel"
    output-channel="shipOrder-Channel" unmarshaller="jaxbMarshaller"/>

  <bean id="jaxbMarshaller"
class="org.springframework.oxm.jaxb.Jaxb2Marshaller">
    <property name="contextPath" value="com.intertech.domain" />
  </bean>
  <int:channel id="shipOrder-Channel" />

  <int:enricher id="ship-order-enricher" input-channel="shipOrder-
Channel"
    output-channel="outboundChannel" request-channel="total-price-
enricher-channel">
    <int:property name="orderTotal" expression="payload" />
    <int:property name="shipped" expression="true"/>
  </int:enricher>
  <int:channel id="total-price-enricher-channel" />
  <int:service-activator id="enriching-service-activator"
    ref="shipOrderEnricher" input-channel="total-price-enricher-
channel"/>

  <bean id="shipOrderEnricher"
class="com.intertech.lab6.ShipOrderEnricher"/>

  <int:channel id="outboundChannel" />

  <int:service-activator id="printing-service-activator"
    input-channel="outboundChannel" ref="serviceActivator" />
  <bean id="serviceActivator"
class="com.intertech.lab6.ExampleServiceActivator" />

</beans>
```

ShipOrderEnricher.java

```
package com.intertech.lab6;

import org.springframework.messaging.Message;
import com.intertech.domain.Shiporder;
import com.intertech.domain.Shiporder.Item;

public class ShipOrderEnricher {

  public double computeTotal(Message<Shiporder> order) {
    double sum = 0;
    for (Item item : order.getPayload().getItem()) {
      sum = sum
          + (item.getPrice().doubleValue() * item.getQuantity()
              .intValue());
    }
    return sum;
  }

}
```