Table of Contents

1. Core Features Lab
   1.1. TypeConverter
   1.2. DataFormat
   1.3. Expressions
   1.4. Threads
   1.5. Exceptions

1. Core Features Lab

Goals

- Understand the DSL, CamelContext, Route, and Message features of the Camel Core engine.
- Manipulate different object types within a Camel Route, including the Bindy DataFormat object.
- Learn about managing threads and local and global exceptions.

Lab Assets

The lab exercises and solutions are available in the following zip archives:

- https://github.com/gpe-mw-training/camel-labs/archive/v0.3-exercise.zip.
- https://github.com/gpe-mw-training/camel-labs/archive/v0.3-solution.zip.

1.1. TypeConverter

The goal of this exercise is to demonstrate how the type conversion strategy in Apache Camel works, and how you can add a new converter using the TypeConverter interface that converts a Vector class into an ArrayList class. To complete the activities in this exercise, you will import the lab assets, review the code, and then run the project locally. You will then change the `typeConverter` and use the `@Converter` annotation.

1. Import the lab assets:
   a. Open **JBoss Developer Studio**.
   b. Select **File** → **Import** from the menu.

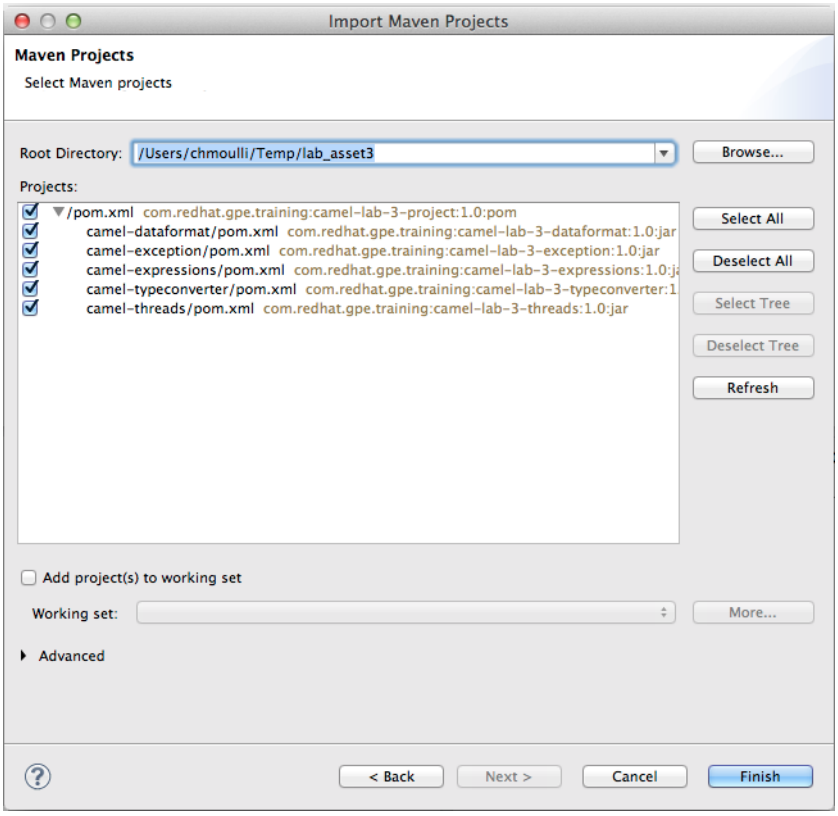

Figure 1. Import the Project

   c. Select **Maven** → **Existing Maven Projects**.
   d. Click **Finish**.
   e. Use **Project Explorer** to confirm that the `camel-lab-3` project was imported.

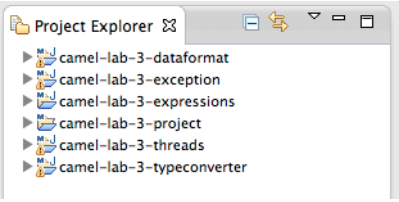

Figure 2. Project Explorer - Imported Lab files

2. Review the project contents:
   - `pom.xml`
   - Other files
   - The `MyRouteBuilder.class`
   - Two Routes exposed using the `direct` component

     First Route

     The route `from:direct/typeconverter` consumes an Exchange containing an Array of these String objects: "Charles", "Chad", "Jeff". When the type converter processor attempts to convert the Exchange Body from an Array to a Vector, these errors occur: `org.apache.camel.InvalidPayloadException` and `No body available of type: java.util.Vector but has value: [Charles, Chad, Jeff] of type: com.redha`

     These errors occur because no Conversion strategy exists for the type Array. As a result, the onException interceptor catches the exception and calls the route exposed by the `from("direct:continue")` statement.

     Second Route

     This second route registers the type converter strategy `context.getTypeConverterRegistry().addTypeConverter(Vector.class, MyArray.class, new ArrayConverter`

     This type converter strategy converts the Array into a Vector. This means that the Exchange containing the Array is converted into a Vector, and the contents of the Exchange body are displayed in the log of the console.
3. Run the project locally:
   a. Expand the `src/main/resources/META-INF/spring` directory in the **Project Explorer** of `camel-lab-3-typeconverter` example.
   b. Right-click the file `spring-camel-context.xml`, and then select **Run As** → **Camel Local Context**.

      When you run the project, the following actions occur:
      - The CamelContext is instantiated by Spring.
      - The Routes are registered and started.
      - A Camel Exchange is created and sent to the `direct:typeconverter` endpoint.
      - A bean called `MessageProducer` implements the `ProducerTemplate` and communicates with the endpoint.

```
2014-10-20 14:30:52,685 [gMainApp.main()] INFO  MainSupport                    - Apache Camel 2.12.0.redhat-610379 starting
2014-10-20 14:30:52,738 [gMainApp.main()] INFO  ClassPathXmlApplicationContext - Refreshing org.springframework.context.support.ClassPathXmlApplicationContext@26d66c
2014-10-20 14:30:52,780 [gMainApp.main()] INFO  XmlBeanDefinitionReader        - Loading XML bean definitions from class path resource [META-INF/spring/spring-camel-
2014-10-20 14:30:53,594 [gMainApp.main()] INFO  DefaultListableBeanFactory     - Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultLis
2014-10-20 14:30:53,815 [gMainApp.main()] INFO  SpringCamelContext             - Apache Camel 2.12.0.redhat-610379 (CamelContext: my-context) is starting
2014-10-20 14:30:53,815 [gMainApp.main()] INFO  ManagedManagementStrategy      - JMX is enabled
2014-10-20 14:30:53,921 [gMainApp.main()] INFO  DefaultTypeConverter           - Loaded 176 type converters
2014-10-20 14:30:54,020 [gMainApp.main()] INFO  SpringCamelContext             - AllowUseOriginalMessage is enabled. If access to the original message is not needed,
2014-10-20 14:30:54,021 [gMainApp.main()] INFO  SpringCamelContext             - StreamCaching is not in use. If using streams then its recommended to enable stream
2014-10-20 14:30:54,064 [gMainApp.main()] INFO  SpringCamelContext             - Route: route1 started and consuming from: Endpoint[direct://typeconverter]
2014-10-20 14:30:54,066 [gMainApp.main()] INFO  SpringCamelContext             - Route: route2 started and consuming from: Endpoint[direct://continue]
2014-10-20 14:30:54,073 [gMainApp.main()] INFO  SpringCamelContext             - Total 2 routes, of which 2 is started.
2014-10-20 14:30:54,074 [gMainApp.main()] INFO  SpringCamelContext             - Apache Camel 2.12.0.redhat-610379 (CamelContext: my-context) started in 0.259 second
2014-10-20 14:30:54,091 [gMainApp.main()] INFO  route1                         - >> Exception should be throw as the typeconverter strategy is not defined from Array
2014-10-20 14:30:54,092 [gMainApp.main()] INFO  route1                         - >> Error : org.apache.camel.InvalidPayloadException: No body available of type: java
2014-10-20 14:30:54,092 [gMainApp.main()] INFO  route2                         - >> We will register the strategy to convert an Array to a Vector
2014-10-20 14:30:54,093 [gMainApp.main()] INFO  route2                         - >> Type looks good now
2014-10-20 14:30:54,094 [gMainApp.main()] INFO  MyRouteBuilder                 - Student : Charles
2014-10-20 14:30:54,094 [gMainApp.main()] INFO  MyRouteBuilder                 - Student : Chad
2014-10-20 14:30:54,094 [gMainApp.main()] INFO  MyRouteBuilder                 - Student : Jeff
```

Figure 3. Console - Sample Output

1.1.1. Change Type Converter

In this activity, you change the `typeConverter` selection and observe the output.

1. Change the typeConverter selection from Array to Collection within the Route (`from:typeconverter`) instead of Vector.
2. Run the project using **Run as** → **Camel Local Context**.
3. Check the console for output.
4. Stop the **CamelContext**.

1.1.2. Use @Converter

In this activity, in order to dynamically register the type converter strategy, you use the `@Converter` annotation declared within the class containing the converters and the file `TypeConverter` created in the `META-INF/services/org/apache/camel/` directory.

1. Add `@Converter` annotations to the `ArrayConverter` class.

2. Remove the registration of the strategy (`//`).
3. Add a `TypeConverter` file and start the **CamelContext**.
4. Stop the **CamelContext**.
5. Close the project.

1.2. DataFormat

The goal of this exercise is to explore a more complex transformation process based on the DataFormat component. You use the Bindy DataFormat, which enables mapping Java Objects to CSV records as well as marshalling and unmarshalling the body of an Apache Camel Exchange.

Because you already imported the project during the previous exercise, you are ready to begin.

1. Open **JBoss Developer Studio** and expand the `camel-lab3-dataformat` project.
2. Review the project contents:
   - `Student` Model class, which contains Bindy Annotations.
3. Launch the **CamelContext** by clicking **Run As** → **Camel Local Context**.
4. Check the log results in the console.

```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: dataformat started and consuming from: Endpoint[file://./target/test-classes/camel/csv
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Total 1 routes, of which 1 is started.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.186 seconds
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : jeff,fuse-camel-traning,1
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : jeff,fuse-camel-traning,1,true,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : chad,fuse-camel-traning,2
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : chad,fuse-camel-traning,2,true,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : samuel,fuse-camel-traning,3
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : samuel,fuse-camel-traning,3,true,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : bernard,fuse-camel-traning,4
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : bernard,fuse-camel-traning,4,true,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : satya,fuse-camel-traning,5
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : satya,fuse-camel-traning,5,true,jboss-fuse-0123
```

Figure 4. Console - Dataformat Results 1

5. As you review the log data in the console, observe the following:
   - The file `students.txt` located in the `test/resources/camel/csv` directory was consumed.
   - The CSV records of the file were split into String objects.
   - Each String was transformed by the Bindy Dataformat into a `HashMap<String, Object>` where the String of the HashMap is the Model class name, and the object is the result of the Bindy processing. Next, the `MyBean` BeanProcessor is called, the content is enriched and a `Student` Object is returned. This `Student` Object forms part of the Exchange Body and is transformed into a String during the marshalling process.

1.2.1. Add a New Field for the Model

1. Add a **Date** field (with `Getter` and `Setter` methods) for the `Student` Model class with a Bindy annotation.
2. Define the input position and output position.
3. Add a pattern (`dd-MM-yyyy`).
4. Extend the file `students.txt` to include an end-of-record field with a date.
5. Launch the **CamelContext** using the command **Run As** → **Camel Local Context**.
6. Check the log results in the console.

```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: dataformat started and consuming from: Endpoint[file://./target/test-classes/camel/csv]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Total 1 routes, of which 1 is started.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.201 seconds
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : jeff,fuse-camel-training,1,5-10-2012
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : jeff,fuse-camel-training,1,05-10-2012,Follow,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : chad,fuse-camel-training,2,6-12-2011
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : chad,fuse-camel-training,2,06-12-2011,Follow,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : samuel,fuse-camel-training,3,30-8-2010
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : samuel,fuse-camel-training,3,30-08-2010,Canceled,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : bernard,fuse-camel-training,4,10-6-2009
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : bernard,fuse-camel-training,4,10-06-2009,Certified,jboss-fuse-0123

Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student : satya,fuse-camel-training,5,1-1-2008
Camel (camel-1) thread #0 - file://./target/test-classes/camel/csv INFO [dataformat] - >> Student Registered : satya,fuse-camel-training,5,01-01-2008,Registered,jboss-fuse-0123
```

Figure 5. Log Console - Data Format Results 2

7. Close the project.

1.3. Expressions

The goal of this exercise is to demonstrate how you can use some of the supported languages, such as `constant`, `simple` and OGNL within an Apache Camel project. These languages are used to set the value of the Exchange Body, the values of two Exchange Headers identified with the values `value1` and `value2`, as well as the result from the `addition`.

The exercise uses a bean called `Calculation` which contains the method `addition` as well as the fields `value1` and `value2`. The simple and OGNL (like mvel) languages both offer the ability to access the `Setter` methods and their corresponding fields in an object.

Follow these steps to complete the exercise:

1. Expand the project `camel-lab3-expressions`.
2. Review the project content:
   - DSL `.setBody(), .setHeader()`
   - Expression languages used
3. Select **Run as** → **Camel Local Context** to launch the project.
4. Observe that every 10 seconds, Apache Camel logs the result of the addition `>> 10 + 20 = 30`.

```
2014-10-20 18:49:33,427 [ing.Main.main()] INFO  SpringCamelContext            - AllowUseOriginalMessage is enabled. If access to the original message is not needed, then its recomm
2014-10-20 18:49:33,427 [ing.Main.main()] INFO  SpringCamelContext            - StreamCaching is not in use. If using streams then its recommended to enable stream caching. See mor
2014-10-20 18:49:33,449 [ing.Main.main()] INFO  SpringCamelContext            - Route: route1 started and consuming from: Endpoint[timer://exercise?delay=2000&period=10s]
2014-10-20 18:49:33,454 [ing.Main.main()] INFO  SpringCamelContext            - Total 1 routes, of which 1 is started.
2014-10-20 18:49:33,455 [ing.Main.main()] INFO  SpringCamelContext            - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.245 seconds
2014-10-20 18:49:35,459 [imer://exercise] INFO  route1                        - >> This is a Camel exercise covering expression languages
2014-10-20 18:49:35,502 [imer://exercise] INFO  route1                        - >> 10 + 20 = 30
2014-10-20 18:49:45,449 [imer://exercise] INFO  route1                        - >> This is a Camel exercise covering expression languages
2014-10-20 18:49:45,451 [imer://exercise] INFO  route1                        - >> 10 + 20 = 30
```

Figure 6. Log Console - Addition results

5. Close the project.

1.3.1. Change the Calculation to a Subtraction

As an optional activity, you can perform subtraction instead of addition by using the `simple` language to set the different elements of the Exchange.

To assign a bean to an Exchange Body, use this `simple` language statement: `${type:FQN_of_the_class}`.

1.4. Threads

The goal of this exercise is to explore aspects of Threads management within an Apache Camel Project using both the `direct` component and the SEDA component in processing the exchanges synchronously and asynchronously.

1.4.1. Synchronous - One thread

This exercise involves one Thread and the `direct` component. An event is fired by the **timer** endpoint every five seconds.

1. In the **Project Explorer**, expand the project `camel-lab3-threads`.
2. Review the code of the class `MainAppSyncOneThread`.
3. To launch the project, right-click the Java Main Application class and select **Run as** → **Java Application**.

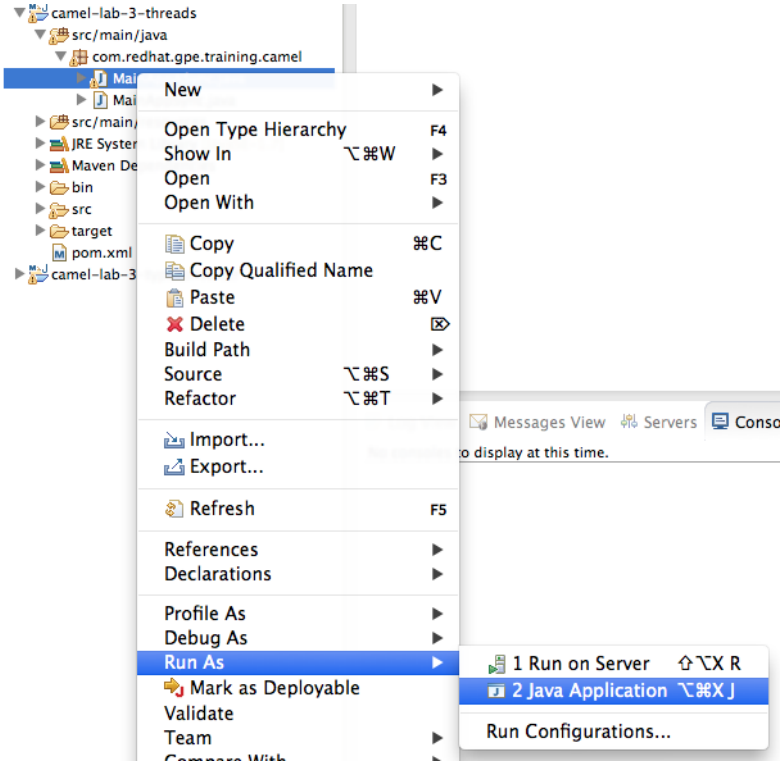

Figure 7. Run the Java Application

4. Observe that every five seconds, Apache Camel logs two messages:

```
>> Timer Direct thread : Camel (camel-1) thread #0 - timer://threadDirect
```

and

```
>> Direct thread : Camel (camel-1) thread #0 - timer://threadDirect
```

One thread was used to produce and process an Exchange.

5. Stop the **Java Application**.

1.4.2. Add Another Direct Endpoint

The goal of this exercise is to extend the previous Route in order to add a new endpoint, and to verify that the same Thread is involved.

Follow these steps to complete the exercise:

1. Call from the Route starting with the `from("direct:thread")` endpoint that another Route exposed with `from("direct:continueThread")`.
2. In the `simple` language, log the `threadName` header property.
3. To launch the modified class, right-click the `Java Main Application` class and select **Run as** → **Java Application**.
4. Check the console results to confirm that `ThreadName = 0` is still used: (`>> Direct continue thread: Camel (camel-1) thread #0 - timer://threadDirect`).
5. Stop the **Java Application**.

1.4.3. Synchronous - Parallel processing

This exercise involves multiple Threads and a customized Thread Pool with the `direct` component. An event is fired by the **timer** endpoint every five seconds.

1. Review the code for the `MainAppSyncMultiThread` class and observe that the Multicast EIP sends the same Exchange to multiple endpoints. In this case, two Exchanges are sent to the same direct endpoint that will process the Exchanges in parallel.
2. To launch the project, right-click the `Java Main Application` class and select **Run as** → **Java Application**.
3. In the console, observe that every five seconds Apache Camel logs a message indicating that threads #5 and #6 in the Thread Pool were used in the parallel processing of the two Exchanges.

```
2014-10-21 11:38:04,874 [main           ] INFO  DefaultCamelContext          - StreamCaching is not in use. If using streams then its recommended to enable stream caching. See more details at ht
2014-10-21 11:38:04,943 [main           ] INFO  DefaultCamelContext          - Route: # Timer for Direct Parallel# started and consuming from: Endpoint[timer://threadDirect?delay=1s&period=5s]
2014-10-21 11:38:04,945 [main           ] INFO  DefaultCamelContext          - Route: route1 started and consuming from: Endpoint[direct://threadparallel]
2014-10-21 11:38:04,951 [main           ] INFO  DefaultCamelContext          - Total 2 routes, of which 2 is started.
2014-10-21 11:38:04,953 [main           ] INFO  DefaultCamelContext          - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.414 seconds
2014-10-21 11:38:05,953 [://threadDirect] INFO  # Timer for Direct Parallel#  - >> Timer Direct parallel thread : Camel (camel-1) thread #0 - timer://threadDirect
2014-10-21 11:38:05,961 [ad #5 - Threads] INFO  route1                       - >> Direct parallel thread : Camel (camel-1) thread #5 - Threads
2014-10-21 11:38:05,961 [ad #6 - Threads] INFO  route1                       - >> Direct parallel thread : Camel (camel-1) thread #6 - Threads
```

Figure 8. Console - Direct Parallel Processing

4. Stop the **Java Application**.
5. Change the size of the Thread Pool to 1 (`pool.setPoolSize(1);`) and restart the Java application.
6. To launch the project, right-click the `Java Main Application` class and select **Run as** → **Java Application**.
7. Observe the processing details:
   - What happens when the Exchanges are consumed by the direct endpoint?
   - Is processing done in parallel or sequentially?
8. Stop the **Java Application**.

1.4.4. Asynchronous

The goal of this exercise is to use an alternative to the direct endpoint, and to observe what happens with threads when an Exchange is produced.

1. Review the code for the `MainAppASync` class.
2. To launch the project, right-click the `Java Main Application` class and select **Run as** → **Java Application**.
3. In the console, observe that every five seconds Apache Camel logs a message indicating that the thread used to produce the message (`#0`) is different from the thread used by the `seda` endpoint to consume the Exchange received (`Thread #1`).

```
2014-10-21 11:51:06,787 [main           ] INFO  DefaultCamelContext          - StreamCaching is not in use. If using streams then its recommended to enable stream caching. See more d
2014-10-21 11:51:06,837 [main           ] INFO  DefaultCamelContext          - Route: # Timer for SEDA # started and consuming from: Endpoint[timer://threadSeda?delay=1s&period=5s]
2014-10-21 11:51:06,846 [main           ] INFO  DefaultCamelContext          - Route: route1 started and consuming from: Endpoint[seda://thread]
2014-10-21 11:51:06,851 [main           ] INFO  DefaultCamelContext          - Total 2 routes, of which 2 is started.
2014-10-21 11:51:06,853 [main           ] INFO  DefaultCamelContext          - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.393 seconds
2014-10-21 11:51:07,849 [er://threadSeda] INFO  # Timer for SEDA #           - >> Timer thread : Camel (camel-1) thread #0 - timer://threadSeda
2014-10-21 11:51:07,853 [- seda://thread] INFO  route1                       - >> Seda thread : Camel (camel-1) thread #1 - seda://thread
```

Figure 9. Console - Asynchronous Processing

4. Stop the **Java Application**.

1.4.5. Asynchronous - Concurrent

In this exercise, you enable parallel processing for the Exchanges by setting a property in the `seda` component: `concurrentConsumer`. As in the last exercise, different threads are used to consume the Exchanges in the SEDA blocking queue in parallel.

1. Review the code for the `MainAppASyncConcurrent` class.

2. To launch the project, right-click the `Java Main Application` class and select **Run as** → **Java Application**.
3. Check the console for the names of the threads.

```
2014-10-21 11:55:32,409 [main           ] INFO  DefaultCamelContext           - Route: # Timer for SEDA Concurrent # started and consuming from: Endpoint[timer://threadSedaConcurre
2014-10-21 11:55:32,414 [main           ] INFO  DefaultCamelContext           - Route: route1 started and consuming from: Endpoint[seda://thread-concurrent?concurrentConsumers=3]
2014-10-21 11:55:32,419 [main           ] INFO  DefaultCamelContext           - Total 2 routes, of which 2 is started.
2014-10-21 11:55:32,421 [main           ] INFO  DefaultCamelContext           - Apache Camel 2.12.0.redhat-610379 (CamelContext: camel-1) started in 0.426 seconds
2014-10-21 11:55:33,420 [dSedaConcurrent] INFO  # Timer for SEDA Concurrent #  - >> Timer thread : Camel (camel-1) thread #0 - timer://threadSedaConcurrent
2014-10-21 11:55:33,430 [read-concurrent] INFO  route1                        - >> Seda concurrent thread : Camel (camel-1) thread #1 - seda://thread-concurrent
2014-10-21 11:55:33,430 [read-concurrent] INFO  route1                        - >> Seda concurrent thread : Camel (camel-1) thread #2 - seda://thread-concurrent
2014-10-21 11:55:33,430 [read-concurrent] INFO  route1                        - >> Seda concurrent thread : Camel (camel-1) thread #3 - seda://thread-concurrent
```

Figure 10. Console Results - Asynchronous Concurrent Processing

4. Stop the **Java Application**.

1.5. Exceptions

This goal of this exercise is to demonstrate how Exceptions and ErrorHandlers can augment the management of messages within an Apache Camel Route. The exercise contains different Camel Routes that are used to generate various exception types in order to demonstrate how the exceptions are handled either by the `onException` clause, or by one of the Apache Camel ErrorHandlers (`DefaultErrorHandler` or `DeadLetterChannel`).

This is the scenario of the exercise:

- Different files are consumed by the `from("file")` endpoint.
- If the XML file contains a payment in `EUR`, (Euro dollars) then a `MyFunctional Exception` occurs, which is handled by the `onException()` clause. The exception appears only once in the log of the console.
- If the file contains a payment in `US` (US dollars), then an `Exception` occurs, but it is **not** a `MyFunctional Exception` and it is **not** handled by the `onException` clause. The `DeadLetterChannel` attempts two redeliveries to the last Camel processor where the error occurred, and then sends the redelivery result to the Camel route that logs to the console.

The `MyBean` POJO is used to generate the various Exception types (`MyFunctional Exception` or just `Exception`) and to increase an internal counter that measures how often the `DeadLetterChannel` attempts redelivery. The log output generated in the console of JBoss Developer Studio documents all the steps described in the scenario.

```
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: cbr started and consuming from:
Endpoint[file://./target/test-classes/camel/in/xml]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: direct-error-handler-with-exception
started and consuming from: Endpoint[direct://errorHandlerWithException]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: direct-error-handler started and
consuming from: Endpoint[direct://errorHandler]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Route: DLQ started and consuming from:
Endpoint[direct://directDLQError]
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Total 4 routes, of which 4 is started.
org.apache.camel.spring.Main.main() INFO [org.apache.camel.spring.SpringCamelContext] - Apache Camel 2.12.0.redhat-610379
(CamelContext: camel-1) started in 0.236 seconds
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [cbr] - Message to be handled: EUPayments.xml, body:
<?xml version="1.0" encoding="UTF-8"?>
<tns:Payments xmlns:tns="http://www.fusesource.com/training/payment"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.fusesource.com/training/payment xsd/Payment.xsd ">

    <tns:Currency>EUR</tns:Currency>

    <tns:Payment>
        <tns:from>ade</tns:from>
        <tns:to>jack</tns:to>
        <tns:amount>1000000.0</tns:amount>
    </tns:Payment>
    <tns:Payment>
        <tns:from>jack</tns:from>
        <tns:to>jill</tns:to>
        <tns:amount>20.0</tns:amount>
    </tns:Payment>
    <tns:Payment>
        <tns:from>ade</tns:from>
        <tns:to>jill</tns:to>
        <tns:amount>42.0</tns:amount>
    </tns:Payment>

</tns:Payments>

Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [org.apache.camel.builder.xml.XPathBuilder] - Created
default XPathFactory com.sun.org.apache.xpath.internal.jaxp.XPathFactoryImpl@66ffaea7
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [cbr] - This is an Euro XML Payment: EUPayments.xml
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [direct-error-handler-with-exception] - Message will be
processed only 1 time.
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [com.redhat.gpe.training.camel.MyBean] - >>>> Exception
created for : EUR, counter = 1
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [direct-error-handler-with-exception] - %%%
MyFunctional Exception handled.
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [cbr] - Message to be handled: USPayments.xml, body:
<?xml version="1.0" encoding="UTF-8"?>
<tns:Payments xmlns:tns="http://www.fusesource.com/training/payment"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://www.fusesource.com/training/payment ../xsd/Payment.xsd ">

    <tns:Currency>USD</tns:Currency>

    <tns:Payment>
        <tns:from>paul</tns:from>
        <tns:to>ade</tns:to>
        <tns:amount>1000000.0</tns:amount>
    </tns:Payment>
    <tns:Payment>
        <tns:from>daan</tns:from>
        <tns:to>jack</tns:to>
        <tns:amount>78.0</tns:amount>
    </tns:Payment>
    <tns:Payment>
        <tns:from>pat</tns:from>
        <tns:to>jill</tns:to>
        <tns:amount>13.0</tns:amount>
    </tns:Payment>

</tns:Payments>

Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [cbr] - This is an USD XML Payment: USPayments.xml
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [direct-error-handler] - Message will be processed 2
times.
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [com.redhat.gpe.training.camel.MyBean] - >>>> Exception
created for : USD, counter = 1
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [com.redhat.gpe.training.camel.MyBean] - >>>> Exception
created for : USD, counter = two
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [com.redhat.gpe.training.camel.MyBean] - >>>> Exception
created for : USD, counter = 3
Camel (camel-1) thread #0 - file://./target/test-classes/camel/in/xml INFO [DLQ] - >>> Info send to DLQ
```

Follow these steps to complete the exercise:

1. Review the code for the `camel-lab-3-exception project` and note the following:
    - Interceptor
    - Configuration of the ErrorHandler (DLQ, …)
    - How exceptions are generated within `MyBean`
2. Expand the `src/main/resources/META-INF/spring` directory in the **Project Explorer** of the `camel-lab-3-exception` example.
3. Right-click `spring-camel-context.xml` and select **Run As -→ Camel Local Context**.
4. After the two files are consumed, review the information logged in the console.
5. Stop the **CamelContext**.

Last updated 2015-11-12 12:04:12 EST