

# Apache Camel Exercises

## Setup

The setup steps are not required if using DaDesktop.

Download and install [Java](#). Download and install [Maven](#). Edit the environment file ~/.bashrc and add.

```
export JAVA_HOME=<path to JDK>
export M2_HOME to the Maven directory.
export M2=$M2_HOME/bin
export PATH=$PATH/$JAVA_HOME/bin:$M2
```

Download [Visual Studio Code](#).

Install Postgres client.

```
sudo apt update
sudo apt install postgresql-client
```

Install [Docker](<https://docs.docker.com/engine/install/>).

Add the student user to the docker group.

```
sudo usermod -a -G docker student
```

Log out and log in again to reload groups.

Pull the Postgres image.

```
docker pull postgres:latest
```

Fetch the exercises.

```
git clone https://github.com/Dr-Phill-Edwards/Camel.git
```

Open the **exercise1** folder in VSCode and enable Java extensions.

# Exercise 1: First Camel Application

Open Visual Studio Code and go to **File** → **Open Folder....** Select the **Camel** folder and open **exercise1**

Look at the dependencies in the POM file pom.xml. Read through the code in Camel.java.

Run the code from a terminal window. Fix any errors. Type control-c to terminate the program.

```
cd ~/Camel/exercise1  
mvn camel:run
```

End of exercise!

# Exercise 2: Camel Unit Testing

In a terminal window, copy exercise1 to exercise2.

```
cd ~/Camel
cp -r exercise1 exercise2
cd exercise2
```

Open the **exercise2** folder in Visual Studio Code.

Replace the configure method in src/main/java/training/Camel.java.

```
@Override
public void configure() {
    from("stream:in?promptMessage=Enter message: ")
        .routeId("camel1")
        .transform(simple("${body.toUpperCase()}"))
        .to("stream:out").id("out");
}
```

Add a test method to src/test/java/training/CamelTest.java.

```
@Test
void shouldChangeToUpperCase() throws Exception{
    AdviceWith.adviceWith(context.getRouteDefinition("camel1"),
        context, new AdviceWithRouteBuilder() {
            @Override
            public void configure() {
                replaceFromWith("direct:in");
                weaveById("out").replace().to("mock:out");
            }
        });
    context.start();
    result.expectedBodiesReceived("HELLO WORLD");
    result.expectedMessageCount(1);
    template.sendBody("direct:in", "hello world");
    result.assertIsSatisfied();
}
```

Compile the code.

```
cd ~/Camel/exercise2
mvn compile
```

Run the unit tests.

```
mvn test
```

Look at the test output. You should see the route displayed as XML before and after the advice.

Run the code. Type control-c to terminate the program.

```
mvn camel:run
```

End of Exercise.

## Exercise 3: Route Filters

There are some files in `data/input`. They contain the poem Jabberwocky from the book "Alice Through the Looking Glass" and a file for each verse.

In a terminal window, copy `exercise1` to `exercise3`.

```
cd ~/Camel
cp -r exercise1 exercise3
cd exercise3
```

Open the `exercise3` folder in Visual Studio Code.

Change the route in `Camel.java` to copy files.

```
@Override
public void configure() {
    from("file:../data/input?noop=true")
    .routeId("camel1")
    .to("file:../data/output").id("out");
}
```

Add the test method to `CamelTest.java`.

```
@Test
void filterOnFilenameTest() throws Exception {
    AdviceWith.adviceWith(context.getRouteDefinition("camel1"),
        context, new AdviceWithRouteBuilder() {
            @Override
            public void configure() {
                weaveById("out").replace().to("mock:out");
            }
        });
    context.start();
    result.message(0).header("CamelFileName").isEqualTo("jabberwocky.txt");
    result.message(1).header("CamelFileName").isEqualTo("jabberwocky1.txt");
    result.message(2).header("CamelFileName").isEqualTo("jabberwocky2.txt");
    result.expectedMessageCount(3);
    result.assertIsSatisfied();
}
```

Run the unit tests. Maven will compile the code automatically.

```
mvn test
```

The tests should fail as the files will not be processed in alphabetical order.

Add resequencing by filename header to the route.

```
.resequence(header("CamelFileName"))
```

Run the tests again, they should pass.

Change the **to** route to a filter which selects files 2, 3 and 5.

```
.filter(simple("${headers.CamelFileName} regex '.*[235].txt'"))  
    .to("stream:out").id("out")  
.end();
```

Change the test expectations to the required files.

Run the tests and they should pass.

Run the code. Type control-c to terminate the program.

```
mvn camel:run
```

You should see verses 2, 3 and 5 of Jabberwocky.

End of Exercise.

# Exercise 4: Multicast Routes

In a terminal window, copy exercise1 to exercise4.

```
cd ~/Camel
cp -r exercise1 exercise4
cd exercise4
```

Open the **exercise4** folder in Visual Studio Code.

Replace the route with the multicast route from the notes.

Replace the result attribute in CamelTest.java with result1 ... result4. Instantiate the mock endpoints to **mock:out1** ... **mock:out4**.

Add a test method. Create an advice that changes the from endpoint to **direct:in**. Weave **out1** to **mock:out1** and the other three.

Start the context. For each result set the expected receive body and set the message count to 1.

Use the template to send a message to **direct:in**.

Assert that each result is satisfied.

Run the test and fix any problems.

Modify Camel.java to add routes for each output.

```
from("direct:out1")
  .transform(simple("${body} #1"))
  .to("stream:out");
```

Run Camel and see the output. Is there anything unexpected? If so why?

End of Exercise.

# Exercise 5a: Aggregator

In a terminal window, copy exercise1 to exercise5.

```
cd ~/Camel  
cp -r exercise1 exercise5  
cd exercise5
```

Open the **exercise5** folder in Visual Studio Code.

Create a new file StringAggregator.java in the training package using the code in the notes.

Create a new file AggregatorTest.java in the test training package using the code in the notes.

Run the tests.

Try adding more tests with different header values. You will need to modify the expected message count and the expected bodies received.

End of Exercise.



# Exercise 5b: Splitter

This exercise still uses the exercise5 directory.

Create a new file SplitterTest.java in the test training package using the code in the notes.

Run the tests.

Add a string tokeniser route to the test and a test method to test it.

End of Exercise.

# Exercise 5c: Routing Slip

This exercise still uses the exercise5 directory.

Create a new file RoutingSlipTest.java in the test training package using the code in the notes.

Run the tests.

Send one or more additional messages in the test. Modify the mock end point expected message counts and bodies received to pass the tests.

End of Exercise.

# Exercise 5d: Dynamic Router

This exercise still uses the exercise5 directory.

Create a new file RouterBean.java in the main training package using the code in the notes.

Create a new file DynamicRouterTest.java in the test training package using the code in the notes.

Run the tests.

Modify the bean to add another route. Modify the mock end point expected message counts and bodies received to pass the tests.

End of Exercise.

# Exercise 5e: Bean Transformer

This exercise still uses the exercise5 directory.

Use a Java Bean Transformer to reverse the characters of the message body string

Create a new file ReverserBean.java in the main training package based on the code in the notes.

Create a new file ReverserBeanTest.java in the test training package based on the code in the notes.

Run the tests.

End of Exercise.

## Exercise 6: CSV, XML and JSON

The file `data/csv/animals.csv` contains data about animals including their body mass in kilograms and their brain mass in grams.

In a terminal window, copy exercise1 to exercise6.

```
cd ~/Camel  
cp -r exercise1 exercise6  
cd exercise6
```

Open the `exercise6` folder in Visual Studio Code.

Create a new file `Animal.java` containing the `Animal` class with Bindy annotations for CSV using the code in the notes.

Add routes for reading the CSV file and displaying the animals using the code in the notes.

Run Camel and see the list of animals. Type control-C to exit.

Create the wrapper class `Animals.java` using the code in the notes.

Annotate `Animal.java` for JAXB.

Create `AnimalListProcessor.java`.

Add the XML route to display the XML.

Run Camel and see the list of animals as XML. Type control-C to exit.

End of Exercise.

# Exercise 7a: Components

In a terminal window, copy exercise1 to exercise7.

```
cd ~/Camel  
cp -r exercise1 exercise7  
cd exercise7
```

Open the **exercise7** folder in Visual Studio Code.

Create StringProcessor.java using the code from the notes.

Create MINATest.java using the code from the notes.

Run the tests.

Create a POJO class of your choice which getters, setters, and a toString() method.

Create a processor class that takes a name string and returns an instance of the POJO.

Add a route to send a message to MINA and send it to the processor.

Add tests for the the route.

Run the tests.

End of Exercise.

# Exercise 7b: Database

This exercise still uses the exercise7 directory.

In a terminal window, check that the database is running. If Postgres is not listed run the `rundb` script. If the database is running when the script executes it will stop and restart it.

```
cd ~/Camel
docker ps
./rundb
```

Take a look at the file `animal.sql` it creates a database table in Postgres. If you created your own POJO for the MINA exercise, create an SQL file to define a table for it and load that into Postgres.

The Postgres client is `psql`. The password is `camel`. You can run SQL statements or execute Postgres commands which are a backslash followed by a letter. Load the table into Postgres and list get the table details. This starts a pager and hit `q` to exit. Exit `psql` using `\q`.

```
psql -h localhost -U camel cameldb
\i animal.sql
\d animal
q
\q
cd exercise7
```

Create `JdbcTest.java` using the code from the notes.

Add the initial setup method to define the data source.

Add the setup method to set up the context.

Create the processor `CreatureInsertProcessor.java`.

Create the route inner class and define the insert route.

Add a test method to send to the insert endpoint in a try - catch block.

Run the tests and fix any problems. The stack traces can be very long and cryptic.

Verify that the database inserts worked.

```
psql -h localhots -U camel cameldb
select * from animal;
\q
```

Add the delete route and send an sql delete statement to the route. Run the tests and fix any issues.

Create CreatureSelectProcessor.java. There are some nasty casts. The guy who implemented generics has apologised!

Implement the select route. Set up mock expectations and send the SQL select to the route. Run the tests and fix any issues.

End of Exercise.



# Project

This project can be done individually or in pairs.

The file `data/csv/penguins.csv` contains data about penguin types and the island (hopefully tariff free) that they inhabit. It the dimensions of penguins' beaks and flippers, body mass and sex. Read the header line for details. Write a Camel application.

- Create a Penguin POJO.
- Create a penguin database table.
- Read in the CSV file.
- Split the CSV file into individual penguin data.
- Write the penguin data to a database.
- Query the database for the type and sex of the heaviest penguin and the one with the longest flipper.

You can get the row with the largest value of a column by ordering the column in reverse order and limiting the returned rows to one.

```
SELECT * FROM penguins ORDER BY flipper DESC LIMIT 1;
```

End of Project.