Camel Exercise -1:

The following is the most basic example of data integration using **Camel**. We will show how to copy files from one folder to another using a simple **Camel** route.

Provided that you have downloaded **Camel**, launch your IDE and create a simple **Java** project containing this class:

```
import java.util.Scanner;

import org.apache.camel.CamelContext;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.impl.DefaultCamelContext;

public class App {
   public static void main(String args[]) throws Exception {
      CamelContext context = new DefaultCamelContext();
      context.addRoutes(new RouteBuilder() {
         public void configure() {
            from("file:/home/francesco?noop=true").to("file:/home/camel");
         }
      });
      context.start();
      System.out.println("Press enter to continue...");
      Scanner keyboard = new Scanner(System.in);
      keyboard.nextLine();
      context.stop();
   }
}
```

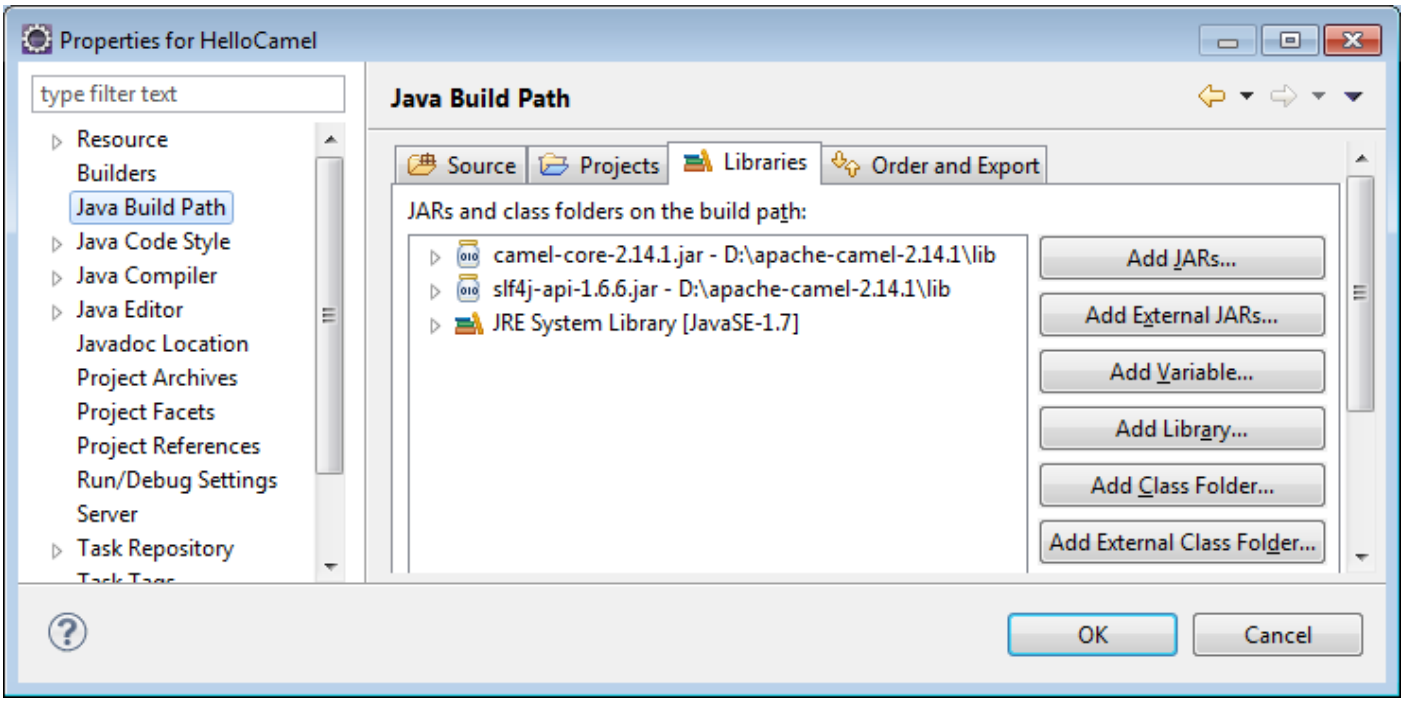
The purpose of this simple code is to route files from folder /home/francesco to /home/camel.

The most interesting part is the **CamelContext**: every Camel application uses a CamelContext that's subsequently started and then stopped. By starting the CamelContext, you will be able to access **Routes** in Camel are defined in such a way that they flow when read. This route can be read like this: route files from /home/francesco with the noop option set, and send to /home/camel.

The String *"file:/home/francesco?noop=true"* is the **Endpoint URI of a Route**. It is composed of three parts: The **Schema** (file), the **Context Path** (/home/francesco) and **Options** (noop=true)

The noop option tells Camel to leave the source file as is. If you didn't use this option, the file would be moved.

In order to compile this example, you will need to include the Camel core libraries and the slf4j log library:



Running with Maven

Creating the same project with Maven is pretty easy as well: start by setting up a quickstart project:

```
D:\maven>mvn archetype:generate -DgroupId=com.sample -DartifactId=hellocamel
-DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false
```

Now replace the com.sample.App class that is created with the quickstart archetype with the one above. Finally, include Camel core dependency in your pom.xml:

```
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-core</artifactId>
      <version>2.14.1</version>
    </dependency>
```

You can run your application using the exec task as follows:

```
1    mvn exec:java -Dexec.mainClass="com.sample.App"
```

3 Ways to run a Camel Application

What we have just seen is just an example to run a **Camel** based application. Actually there are three ways to run a Camel application:

1 - Running as a Java standalone application

Camel runs in a standalone application triggering the CamelContext initialization. This approach has been described at the beginning of this tutorial and just needs to initialize the CamelContext from your Java code and package the required Camel libraries in your application:

```
public class JavaRouter {

  public static void main(String[] args) throws Exception {

    context.addRoutes(new RouteBuilder() {

      public void configure() {
        from("test-jms:queue:test.queue").to("file://test");
      }

  });
}
```

2 - Running Camel with Spring and a Java starter class

An alternative way is to trigger the startup with the Spring Framework.

```
public class SpringRouter {

  public static void main(String[] args) throws Exception {

    AbstractXmlApplicationContext springCtx = new  ClassPathXmlApplicationContext("
    META-INF/camelContext.xml");

    springCtx.start();

    Thread.sleep(10000);

    springCtx.stop();

    springCtx.destroy();

  }
```

```
        }
```

Here is the example camelContext.xml file:

```
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:camel="http://camel.apache.org/schema/spring"
      xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
        http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-
spring.xsd">

  <camelContext id="camel5" xmlns="http://camel.apache.org/schema/spring">

      <routeBuilder ref="myBuilder" />

  </camelContext>

  <bean id="myBuilder" class="org.apache.camel.spring.example.test1.MyRouteBuilder"/>

</beans>
```

3 - Running Camel with Spring and Camel Maven plugin

The last option allows you to write and execute Camel routes without writing Java code at all. - You will define your route in Spring's application context file and use Camel's Maven plugin to compile and run the code. Here is the application-context.xml file:

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:camel="http://camel.apache.org/schema/spring"
   xmlns:util="http://www.springframework.org/schema/util"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
    http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/camel-spring.xsd
     http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.1.xsd">

   <camelContext xmlns="http://camel.apache.org/schema/spring">
      <route>
         <from uri="file://data/in" />
         <to uri="file://data/out" />
      </route>
   </camelContext>
```

```
</beans>
```

And this is the pom.xml file:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.esample.camel</groupId>
  <artifactId>camel-spring-maven</artifactId>
  <version>1.0.0.BUILD-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>Using Camel for Simple file copy - with Spring integration</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>3.1.2.RELEASE</spring.version>
    <camel.version>2.10.2</camel.version>
  </properties>

  <dependencies>
    <dependency>
      <groupId>org.apache.camel</groupId>
      <artifactId>camel-spring</artifactId>
      <version>${camel.version}</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>${spring.version}</version>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.camel</groupId>
        <artifactId>camel-maven-plugin</artifactId>
        <version>${camel.version}</version>
        <!-- the file must be in the classpath -->
        <configuration>
```

```
            <applicationContextUri>application-context.xml</applicationContextUri>
          </configuration>
        </plugin>
      </plugins>
    </build>

</project>
```

To run the above example, execute the following command in the shell:

```
1       mvn clean compile camel:run
```