Camel Processor

Camel Processor is one of the key building blocks, and it gives access to the full message transferred, including body, headers, and properties associated with the message. By creating a processor, you can modify any elements or properties of the message body; for example, you can add custom properties to the header, transforming the message from CSV to XML, etc.

Camel Processor is interface defined in **org.apache.camel.processor,** having a single method.

```
public void process(Exchange exchange) throws Exception
```

Configuring routes startup ordering and autostartup

Available as of Camel 2.1

Camel now supports configuring two aspects:

- auto startup
- order of starting routes

By default a route is auto started.

Configuring whether a route should be started or not in Java DSL

You can use the autoStartup option to configure if a given route should be started when Camel starts. By default a route is auto started.

You can disable or enable it as follows:
from("activemq:queue:special").noAutoStartup().to("file://backup");

Why do you want to control the starting order?

It can help in cases where routes are inter dependent on each other and also help with graceful shutting down Camel as Camel can stop the routes in the correct order as well.

Stopping routes

Camel 2.2: Camel will stop the routes in the **same** order that they were started.
Camel 2.3: Camel will stop the routes in the **reverse** order that they were started.

Simple example

from("seda:foo").startupOrder(1).to("mock:result");

from("direct:start").startupOrder(2).to("seda:foo");

And the same example with XML DSL:

```
<route startupOrder="1">

   <from uri="seda:foo"/>

   <to uri="mock:result"/>

</route>


<route startupOrder="2">

   <from uri="direct:start"/>

   <to uri="seda:foo"/>

</route>
```

What is CamelContext

The CamelContext is the runtime system of Apache Camel and connects its different concepts such as routes, components or endpoints.
Below are the steps that we need to consider while configuring the CamelContext:

1. Create CamelContext.
2. Add endpoints or components.
3. Add Routes to connect the endpoints.
4. Invoke `CamelContext.start()` – This starts all the camel-internal threads which are responsible for receiving, sending and processing messages in the endpoints.
5. Lastly, invoke `CamelContext.stop()` when all the messages are exchanged and processed. This will gracefully stop all the camel-internal threads and endpoints.

```
public class CamelStarter {
public static void main(String[] args) throws Exception {
   CamelContext context = new DefaultCamelContext();
   context.addRoutes(new IntegrationRoute());
   context.start();
   Thread.sleep(30000);
   context.stop();
   }
}
```

Camel Core Components

The `camel-core` module ships in with some built-in components. We have listed few important ones below:

1. Bean – Invokes a Java bean in the registry.
2. Direct – Allows you to synchronously call another endpoint with little overhead.
3. File – Allows you to work with files, to reads or write to files.
4. Log – Logs messages to a number of different logging providers.
5. Mock – Tests that messages flow through a route as expected.
6. SEDA – Allows you to asynchronously call another endpoint in the same `CamelContext`
7. Timer – Sends out messages at regular intervals