**An introduction to Camel :**

**What is Camel :**

Apache Camel is an open source integration framework that aims to make

integrating systems easier. Camel provides simple, manageable abstractions for the complex systems you’re integrating . At the core of the Camel framework is a routing engine, or more precisely a routingengine

builder. It allows you to define your own routing rules, decide from which sources to accept messages, and determine how to process and send those messages to other destinations. Camel uses an integration language that allows you to define complex routing rules, akin to business processes. One of the fundamental principles of Camel is that it makes no assumptions about the type of data you need to process. This is an important point, because it gives you, the developer, an opportunity to integrate any kind of system, without the need to

convert your data to a canonical format.

Camel offers higher-level abstractions that allow you to interact with various systems using the same API regardless of the protocol or data type the systems are using.Components in Camel provide specific implementations of the API that target different protocols and data types. Camel comes with support for over 80 protocols and data types. Its extensible and modular architecture allows you to implement and seamlessly plug in support for your own protocols. These architectural choices eliminate the need for unnecessary conversions and make Camel not only faster but also very lean.

Camel doesn’t have container or a reliable message bus, but it can be deployed in one .

**Why Use Camel :**

Camel introduces a few novel ideas into the integration space. Please see below

■ Routing and mediation engine

■ Enterprise integration patterns (EIPs)

■ Domain-specific language (DSL)

■ Extensive component library

■ Payload-agnostic router

■ Modular and pluggable architecture

■ POJO model

■ Easy configuration

■ Automatic type converters

■ Lightweight core

■ Test kit ■ Vibrant community

**Routing and mediation engine :**

The core feature of Camel is its routing and mediation engine. A routing engine will

selectively move a message around, based on the route’s configuration.

**DOMAIN-SPECIFIC LANGUAGE (DSL) :**

Camel’s domain-specific language (DSL) is a major contribution to the integration

space. A few other integration frameworks currently feature a DSL (and some allow

you to use XML to describe routing rules), but unlike Camel their DSLs are based on

custom languages.Camel is unique because it offers multiple DSLs in regular programming

languages such as Java, Scala, Groovy, and it also allows routing rules to be

specified in XML. The purpose of the DSL is to allow the developer to focus on the integration problem

rather than on the tool.

Here are some examples of the DSL using different languages and staying functionally

equivalent:

■ Java DSL

from("file:data/inbox").to("jms:queue:order");

■ Spring DSL

<route>

<from uri="file:data/inbox"/>

<to uri="jms:queue:order"/>

**Routing With Camel**

Router’s function is to selectively move the message forward. In the context of enterprise messaging systems, routing is the process by which a message is taken from an input queue and, based on a set of conditions, sent to one

of several output queues.

In Apache Camel, routing is a more general concept. It’s defined as a step-by-step

movement of the message, which originates from an endpoint in the role of a consumer.

The consumer could be receiving the message from an external service, polling

for the message on some system, or even creating the message itself. This message

then flows through a processing component, which could be an enterprise integration

pattern (EIP), a processor, an interceptor, or some other custom creation. The message

is finally sent to a target endpoint . Camel use URI to communicate over

FTP and JMS. Through URI,You can decide to either send messages to the component configured by

this URI, or to consume messages from it. A Camel endpoint URI consists of three parts: a scheme, a context

path, and a list of options.

***Sending to a JMS queue :*** Camel provides extensive support for connecting to JMS-enabled providers.In JMS, message consumers and producers talk to one another through an intermediary a JMS destination. JMS also provides a ConnectionFactory that clients (like Camel) can use to create a connection with a JMS provider.

There are two types of JMS destinations: queues and topics.

**CONFIGURE CAMEL TO USE A JMS PROVIDER :**

To connect Camel to a specific JMS provider, you need to configure Camel’s JMS component

with an appropriate ConnectionFactory. Apache ActiveMQ is one of the most popular open source JMS providers.

You can create an ActiveMQConnectionFactory that points to the location of the running ActiveMQ broker:

ConnectionFactory **connectionFactory** = new ActiveMQConnectionFactory("vm://localhost");

create your CamelContext, you can add the JMS component as follows:

CamelContext context = new DefaultCamelContext();

context.addComponent("jms",JmsComponent.jmsComponentAutoAcknowledge(**connectionFactory**));

Needed Jar File: camel-core,camel-jms, activemq-core

Now that you’ve configured the JMS component to connect to an actual JMS broker,

it’s time to look at how URIs can be used to specify the destination.

**USING URIS TO SPECIFY THE DESTINATION**

jms:queue:incomingOrders

Using Camel’s Java DSL, you can send a message to the incomingOrders queue by

using the to keyword like this:

.to("jms:queue:incomingOrders")

***Creating routes in Java:***

RouteBuilders are used to create routes in Camel. Each RouteBuilder can create multiple routes.

CamelContext context = new DefaultCamelContext();

context.addRoutes(new RouteBuilder() {

public void configure() throws Exception {

...

}

});

Within the configure method, you define your routes using the Java DSL.

import javax.jms.ConnectionFactory;

import org.apache.activemq.ActiveMQConnectionFactory;

import org.apache.camel.CamelContext;

import org.apache.camel.builder.RouteBuilder;

import org.apache.camel.component.jms.JmsComponent;

import org.apache.camel.impl.DefaultCamelContext;

public class FtpToJMSExample {

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

CamelContext context = new DefaultCamelContext();

ConnectionFactory connectionFactory =

new ActiveMQConnectionFactory("vm://localhost");

context.addComponent("jms",

JmsComponent.jmsComponentAutoAcknowledge(connectionFactory));

context.addRoutes(new RouteBuilder() {

public void configure() {

from("ftp://rider.com/orders"

+ "?username=rider&password=secret")

process(new Processor() {

public void process(Exchange exchange) throws Exception {

System.out.println("We just downloaded: "

+ exchange.getIn().getHeader("CamelFileName"));

}

})

.to("jms:incomingOrders");

}

});

context.start();

Thread.sleep(10000);

context.stop();

}

}

**Creating routes with Spring :**

Spring is the most popular Inversion of Control (IoC) Java container out there. The

core framework allows to you “wire” beans together to form applications. This wiring

is done through an XML configuration file.

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://camel.apache.org/schema/spring

http://camel.apache.org/schema/spring/camel-spring.xsd">

<bean id="jms" class="org.apache.camel.component.jms.JmsComponent">

<property name="connectionFactory">

<bean class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="brokerURL" value="vm://localhost" />

</bean>

</property>

</bean>

<bean id="ftpToJmsRoute" class="camelinaction.FtpToJMSRoute"/>

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

<routeBuilder ref="ftpToJmsRoute"/>

</camelContext>

</beans>

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

This will automatically start a SpringCamelContext, which is a subclass of the

DefaultCamelContext you used for the Java DSL

public class FtpToJMSRoute extends RouteBuilder {

public void configure() {

from("ftp://rider.com" +

"/orders?username=rider&password=secret")

.to("jms:incomingOrders");

}

}

**USING SPRING DSL:**

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://camel.apache.org/schema/spring

http://camel.apache.org/schema/spring/camel-spring.xsd">

<bean id="jms" class="org.apache.camel.component.jms.JmsComponent">

<property name="connectionFactory">

<bean class="org.apache.activemq.ActiveMQConnectionFactory">

<property name="brokerURL" value="vm://localhost" />

</bean>

</property>

</bean>

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

<route>

<from

uri="ftp://rider.com/orders?username=rider&password=secret"/>

<to uri="jms:incomingOrders"/>

</route>

</camelContext>

</beans>

Reference link :

http:// manning.com/ibsen

http://code.google.com/p/camelinaction.