



# Apache Camel

Open-source, message oriented,  
rule-based routing and mediation engine

– **Avadhut**

# About Me

Apache Camel



- I'm Avadhut!
- Working as Senior Integration Engineer
- Open-source enthusiast
- Active community member for couple of open-source projects
- Worked with Integration, Fuse, Camel, Karaf, Kafka and messaging platforms for quite a bit
- Did full production deployments, architecture review and performance tuning for couple of employers and lot of Red Hat customers

**\*\* You can find me on:**

**GitHub:** <https://github.com/kodtodya>

**GitLab:** <https://gitlab.com/kodtodya>

**LinkedIn:** <https://www.linkedin.com/in/kodtodya/>

**Twitter:** <https://twitter.com/kodtodya>

<https://kodtodya.github.io/talks/>



# Pre-Requisite for Fuse Course

- Knowledge of Java or overall programming (We will use Java-8 for this course)
- Knowledge of Spring and Spring framework
- Eclipse/RHDS is strongly preferred, other IDEs are also OK
- Knowledge of Maven is mandatory too..
- Willingness to learn an awesome technology... 😊



# Who is this course for?

- **Developer:** who would like to learn how to write and run an Camel application that integrates various systems and servers
- **Architects:** who want to understand the role of Camel in the enterprise integration with/using micro-services
- **DevOps:** who want to understand how Camel works with regards to various systems, protocols, components and its infrastructural setup



# Welcome...!!!

Apache Camel

- A warm welcome to Apache Camel Rapid Track course..

<https://kodtodya.github.io/talks/>



# Course Structure

Apache Camel

## Part – 1 : Fundamentals

- What is Camel?
- Role of Camel in Fuse Integration
- Routes, Endpoints and Components
- Exchange and Message
- Camel context
- Processor
- Producer & Consumer
- Basic Data transformation
- Exception Handling



# Course Structure

Apache Camel

- **Part – 2 : Enterprise Integration Patterns**
  - EIP theory and Important EIP implementation in Camel
- **Part – 3 : Practicals**
  - Basic component handling
  - Important EIP implementation in Camel
  - Threading and transactions
  - Performance Tuning





# Apache Camel

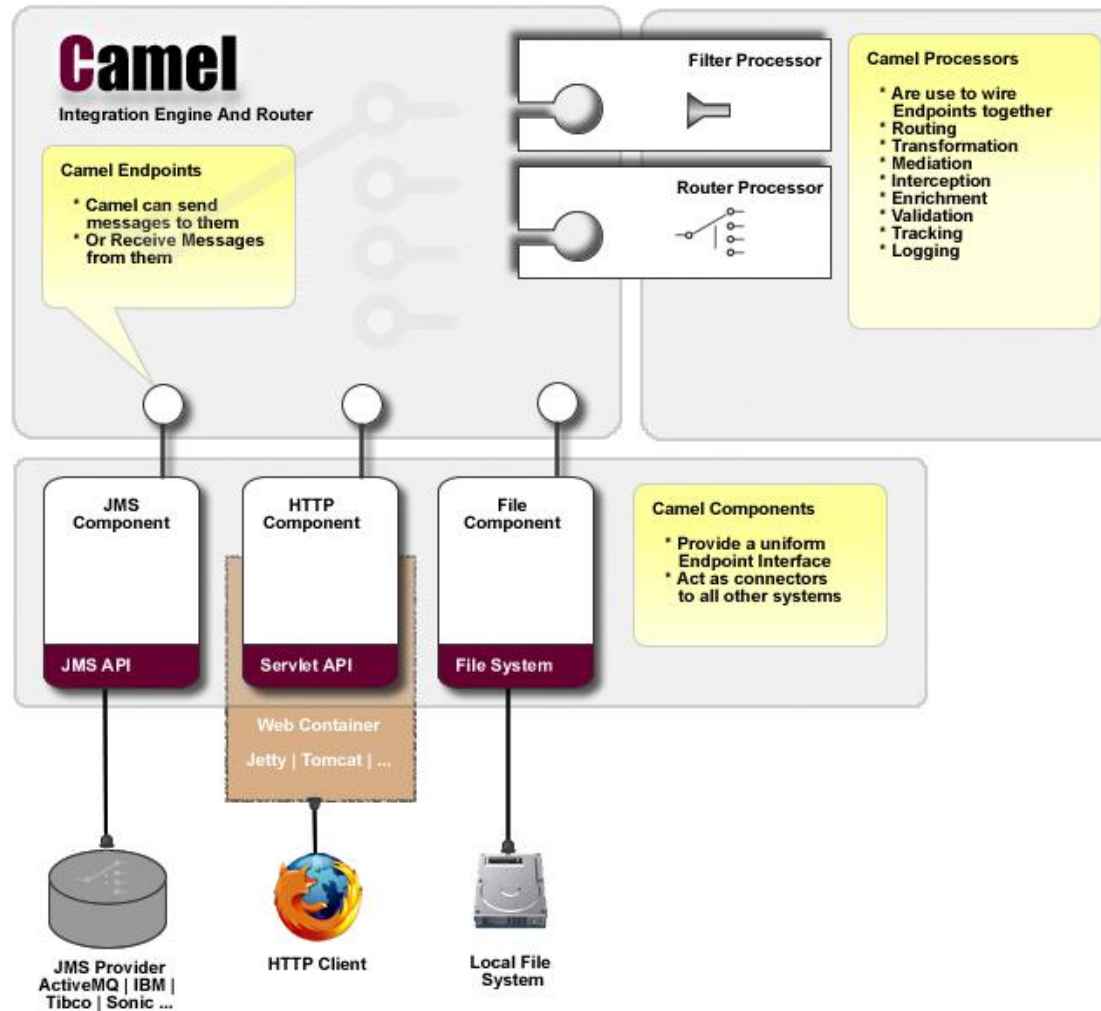
## Fundamentals





# What is Camel?

Apache Camel



<https://kodtodya.github.io/talks/>



# What is Apache Camel ?

- ❑ **Versatile open-source integration framework based on known Enterprise Integration Patterns**
- ❑ **Empowers you to define routing and mediation rules in a variety of domain-specific languages, including a Java API,**
- ❑ **Spring/Blueprint XML Configuration files, and a Scala DSL**
- ❑ **Open source framework for message-oriented middleware with a rule-based routing and mediation engine**
- ❑ **Camel provides a Java object-based implementation of the Enterprise Integration Patterns using an application programming interface**



# Where did the idea for Camel come from?

Apache Camel

- ◆ Camel was largely inspired by the book **Enterprise Integration Patterns**, a sort of academic textbook for integrating software systems.
- ◆ The authors of the book (now considered a 'classic'!) took dozens of common use cases and distilled them into reusable patterns, describing their use and, in some cases, providing suggested code for how to implement them.
- ◆ The Camel developers thought it would be a great idea to build a Java framework that represented the ideals of the book.
- ◆ ... And so Apache Camel borrows heavily from the book.

<https://kodtodya.github.io/talks/>



# Some facts worth noting about Camel

Apache Camel

- ◆ **It's built in Java**
- ◆ **Completely open source – nothing is hidden behind expensive closed-source products**
- ◆ **Not just for web services – it's a general integration framework**
- ◆ **Comes with a huge library of components**
- ◆ **It's pretty much mature now**



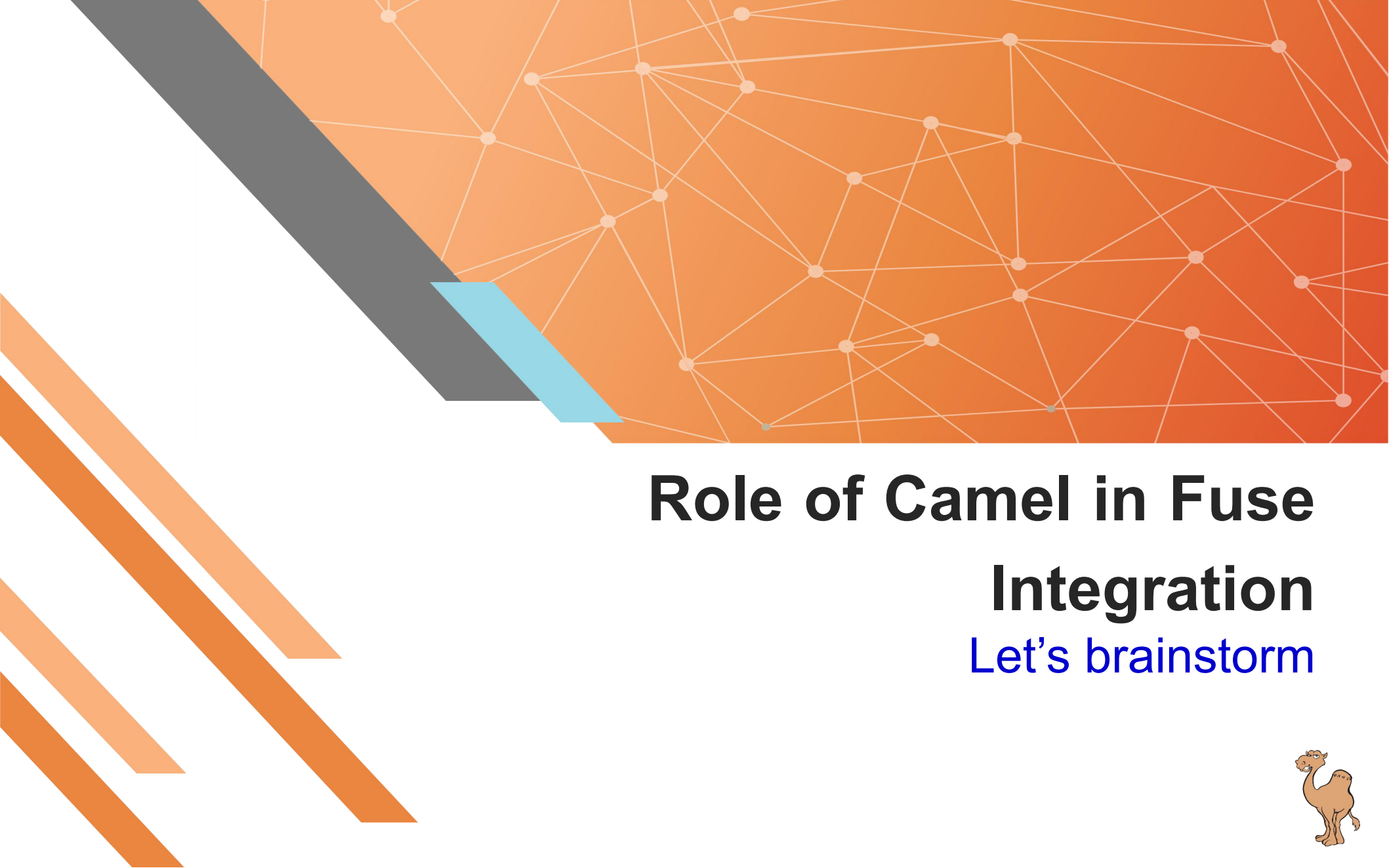
# What is Camel used for ?

Apache Camel

- ◆ Picking up data from some FTP server and emailing it to you
- ◆ Reading files from a folder and pushing them into Google Drive
- ◆ Reading JMS message and processed it with Rest service
- ◆ Getting data from Database and sending it over web-service response
- ◆ Collect data from different endpoints of IoT
- ◆ ... and lot more...

<https://kodtodya.github.io/talks/>





# **Role of Camel in Fuse**

## **Integration**

Let's brainstorm



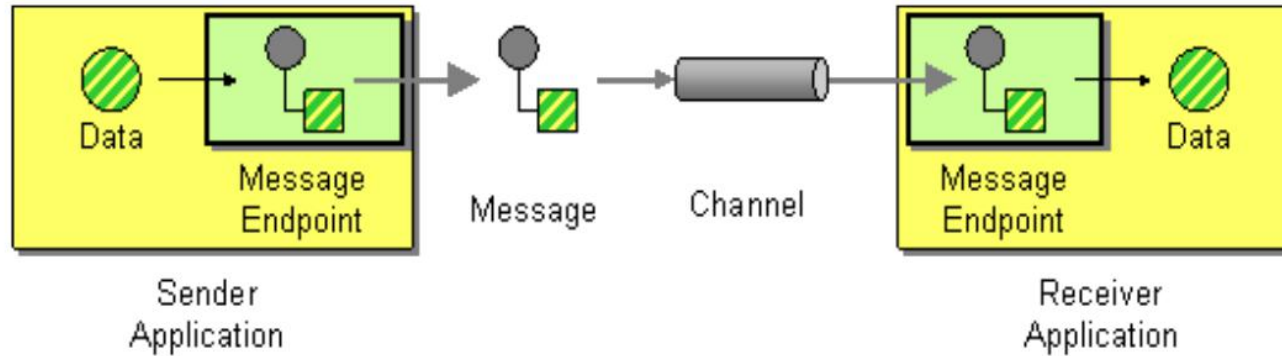


# Thinking in Camel

Let's think..



# What is Routes ?



- The basic concept in Camel is the Route. Routes are objects which you configure in Camel, which move your data from A to B
- Route is a pipe/channel that moves data from one place to another
- You can create routes in Camel using Java syntax or XML syntax
- Camel routes are small channels that move data between endpoints, using components

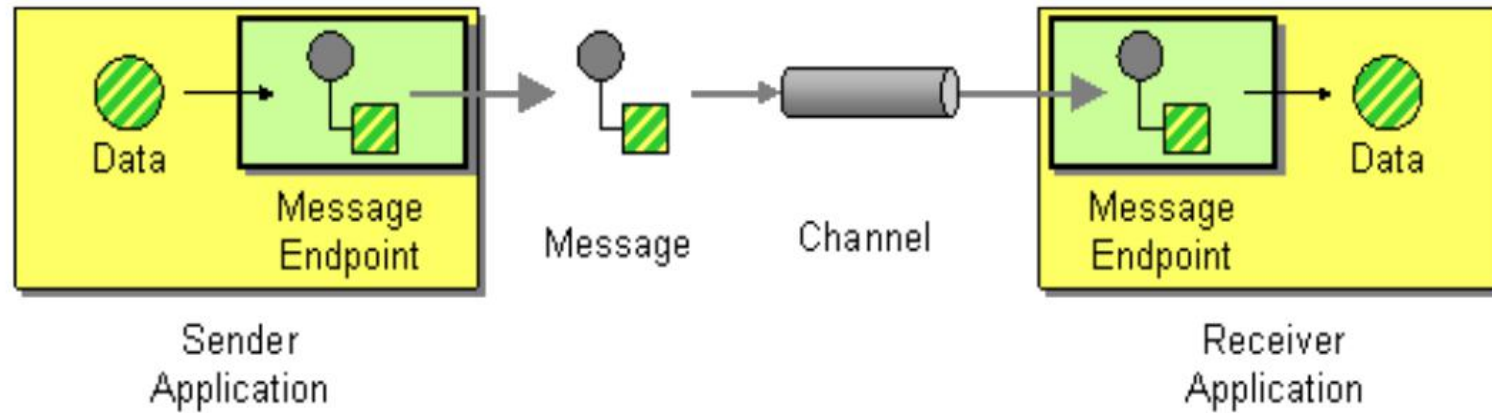
Example:

```
from("file:home/sample-files/myFile").to("activemq:queue:myQ");
```





# What is Endpoint?



- **Endpoints are external systems connected to Camel**
- **Camel receives message from “from” endpoint and sent to “to” endpoint**
- **Endpoints are either consumer or producer in general**
- **Message might be processed by EIP before sending to “to” endpoint**

Example:

```
from("file:home/sample-files/myFile").to("activemq:queue:myQ");
```



# What is Components ?



- To allow camel to connect to external systems/endpoints, Camel provides library of components
- Components are simply configurable plugs that allows us to connect to external systems
- Camel components are configurable, open-source, re-usable and you can even add your own component

Example:

```
from("file:home/sample-files/myFile").to("activemq:queue:myQ");
```



# What is Exchange ?

- **org.apache.camel.Exchange** is abstraction for an exchange of messages as part of a "conversation". Below are components of a such conversation, that make this abstraction more comprehensive.
- **Exchange ID**
- **MEP** a particular Message Exchange Pattern like InOnly or InOut. When the pattern is InOnly, the exchange contains only IN- Message.
- **Exception** when it occurs at any time during routing.
- **Properties** are similar to message headers, but they last for the duration of the entire exchange also camel may add some properties to a particular exchange.
- **IN-Message** or request message is mandatory.
- **OUT-Message** or replay message exists if the MEP is defined as InOut.



# What is Message ?

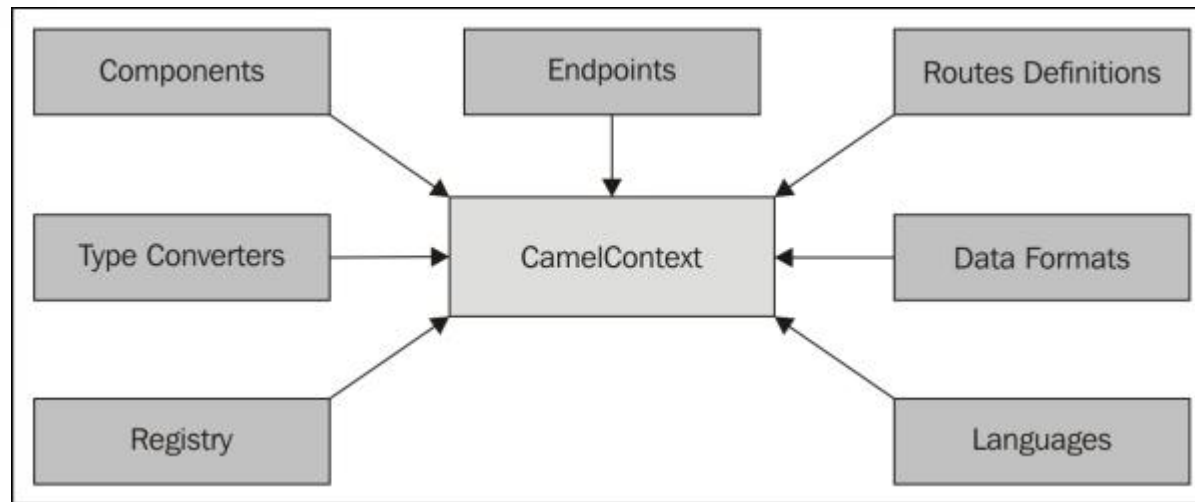
- Camel considers every data object passing through it as separate message
- Every message is separate unit
- Camel represents data object as *Message*
- Message is sub-set of Exchange
- Message body contains any kind of object
- Camel has a lot of built-in support for converting between different object types
- In fact, for many common file types, you might barely even have to write any code. (Less time writing boilerplate code? Sounds good, Isn't it?)



# What is Camel Context?



- Camel context is the runtime system and the loading container of all resources required for the execution of the routing
- It keeps everything together to allow the user to execute the routing logic. When the context starts, it also starts various components and endpoints, and activates the routing rules.

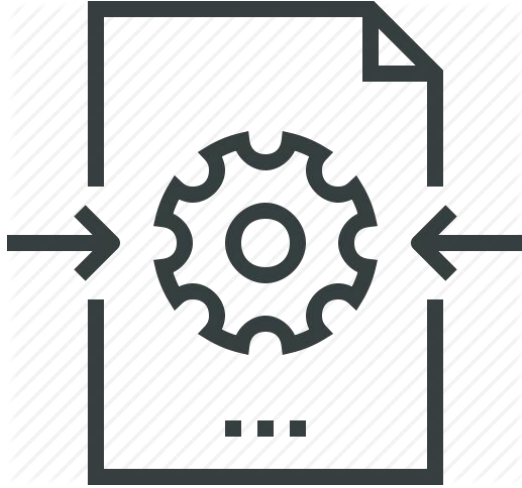


# What is Camel Context?

- **As same like Spring, Camel has it's own tiny container that help us to run and manage our routes**
- **You can consider it as small container as our routes run inside this engine**
- **When camel starts, it adds all the routes from application and add it to Camel context**
- **Camel context contains following:**
  - The components and endpoints used in the routing (see later for the details about components and endpoints)
  - The type converters used to transform a message of one type to another
  - The data formats used to define the format of a message body
  - The registry where Camel will look for the beans used in the routing
  - The languages describing expressions and predicates used in the routing by a language (xpath, xquery, PHP, and so on)
  - The routes definition itself allowing you to design...



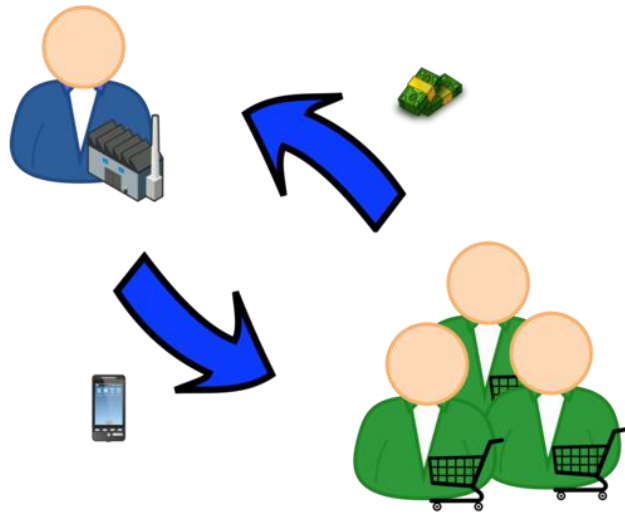
# What is Processor?



- Message are processed by an Enterprise Integration Pattern
- Processors are entities which processes data in Camel
- The *Processor* interface is used to implement consumers of message exchanges or to implement a Message Translator
- Processor is piece of code which goes in between routes.
- You can write code to manipulate (transform/enrich/extract etc.) the message or have some integration logic inside processor



# Producer and Consumer



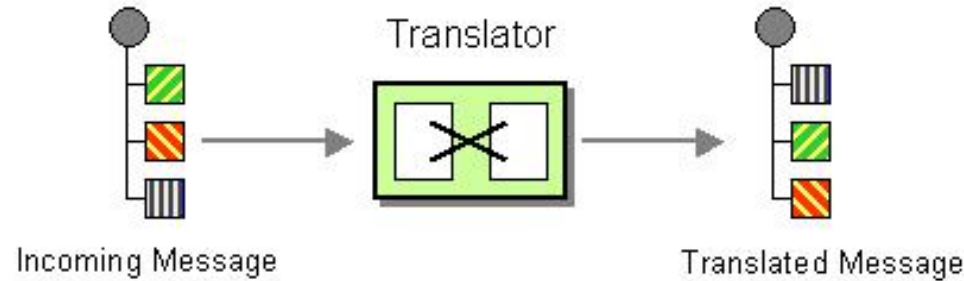
Producer - Consumer

- **Component that is configured to write something is called a producer — for example, writing to a file on disk, or writing to a message queue.**
- **A component that is configured to read something is called a consumer — for example, reading a file from disk, or receiving a REST request.**





# Data Transformation



- **Camel supports the Message Translator from the EIP patterns by using an arbitrary Processor in the routing logic, by using a bean to perform the transformation, or by using `transform()` in the DSL.**
- **You can also use a Data Format to marshal and unmarshal messages in different encodings.**



# Exception Handling

- Try...Catch...Finally
- doTry...doCatch...doFinally...end
- Dead Letter Channel as ErrorHandler
- DefaultErrorHandler
- Spring transactions
- onException()
  - handled()
  - continued()
  - useOriginalMessage()



# Exception Handling

## ● Provided Error Handlers

- Default Error Handler
- `LoggingErrorHandler`
- `NoErrorHandler`
- `TransactionErrorHandler`
- `TransactionalErrorHandler`

## ● Advance usage of Exception Clause

- `onRedelivery()`
- `onWhen()`
- `retryWhile()`
- `Custom ExceptionPolicyStrategy`
- Global and Per Route Exception Clauses



# doTry...doCatch...doFinally

- ❑ When we use doTry .. doCatch .. doFinally, then the regular Camel Error Handler does not apply
- ❑ That means any onException or the equivalent exception handler does not trigger
- ❑ The reason is that doTry .. doCatch .. doFinally is, in fact, its own error handler and that it aims to mimic and work like how try/catch/finally works in Java
- ❑ you can define multiple exceptions to catch in a single block
- ❑ Camel will check in exception hierarchy when it matches a thrown exception against the doCatch blocks
- ❑ you can attach a onWhen predicate to signal if the catch should trigger or not at runtime



# onException

- Camel uses `DefaultExceptionPolicyStrategy` to determine a strategy how an exception being thrown should be handled by which `onException` clause. The strategy is:
- the order in which the `onException` is configured takes precedence. Camel will test from first...last defined.
- Camel will start from the bottom (nested caused by) and recursive up in the exception hierarchy to find the first matching `onException` clause.
- `instanceof` test is used for testing the given exception with the `onException` clause defined exception list. An exact `instanceof` match will always be used, otherwise the `onException` clause that has an exception that is the closest super of the thrown exception is selected (recurring up the exception hierarchy).



## Features Support by Various **Error Handlers**

Here is a breakdown of which features is supported by the **Error Handler(s)**:

Feature	Supported By The Following <b>Error Handler</b>
<i>all scopes</i>	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
onException	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
onWhen	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
continued	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
handled	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
Custom ExceptionPolicy	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
useOriginalBody	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
retryWhile	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
onRedelivery	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
RedeliveryPolicy	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
asyncDelayedRedelivery	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
redeliverWhileStopping	DefaultErrorHandler, TransactionErrorHandler, Dead Letter Channel
<i>dead letter queue</i>	Dead Letter Channel
onPrepareFailure	DefaultErrorHandler, Dead Letter Channel

See **Exception Clause** documentation for documentation of some of the features above. <https://dodotodaya.github.io/talks/>



# Error Handler

- ❑ Camel supports pluggable ErrorHandler strategies to deal with errors processing an Event Driven Consumer
- ❑ An alternative is to specify the error handling directly in the DSL using the Exception Clause
- ❑ Using Error Handler combined with Exception Clause is a very powerful combination
- ❑ We encourage end-users to use this combination in your error handling strategies
- ❑ We roughly distinguish Error Handler in 2 major categories:
  1. Non-Transacted
  2. Transacted



**DefaultErrorHandler** is the default error handler in Camel which does not support a dead letter queue, it will propagate exceptions back to the caller, as if there where no error handler at all. It has a limited set of features.

**Dead Letter Channel** which supports attempting to redeliver the message exchange a number of times before sending it to a dead letter endpoint

**LoggingErrorHandler** for just catching and logging exceptions

**NoErrorHandler** for no error handling





- ✓ **TransactionErrorHandler** is the default error handler in Camel for transacted routes.
- ✓ These error handlers can be applied in the DSL to an entire set of rules or a specific routing rule.
- ✓ Error handling rules are inherited on each routing rule within a single RouteBuilder.
- ✓ More details can be found in Transactional Client EIP pattern.



# Let's revise – Part-1

Apache Camel

## ● Part – 1 : Fundamentals

- What is Camel?
- Role of Camel in Fuse Integration
- Routes, Endpoints and Components
- Exchange and Message
- Camel context
- Processor
- Producer & Consumer
- Basic Data transformation
- Exception Handling

<https://kodtodya.github.io/talks/>



# Questions ?

Apache Camel



<https://kodtodya.github.io/talks/>





# Thank you..!!!

LinkedIn, GitHub, GitLab, Twitter: [@kodtodya](#)

<https://kodtodya.github.io/talks/>