

# Training

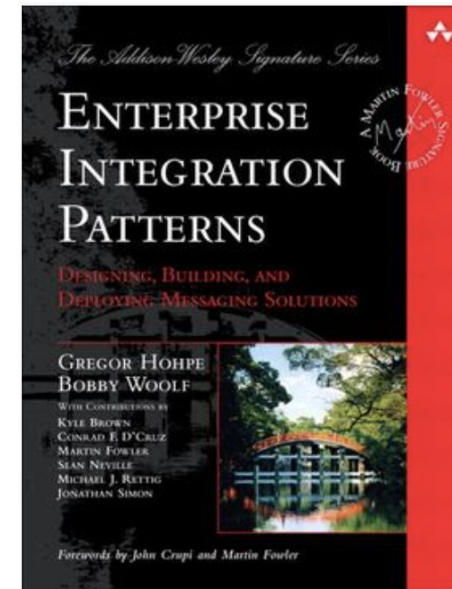
## Get Started with Apache Camel

April 2024



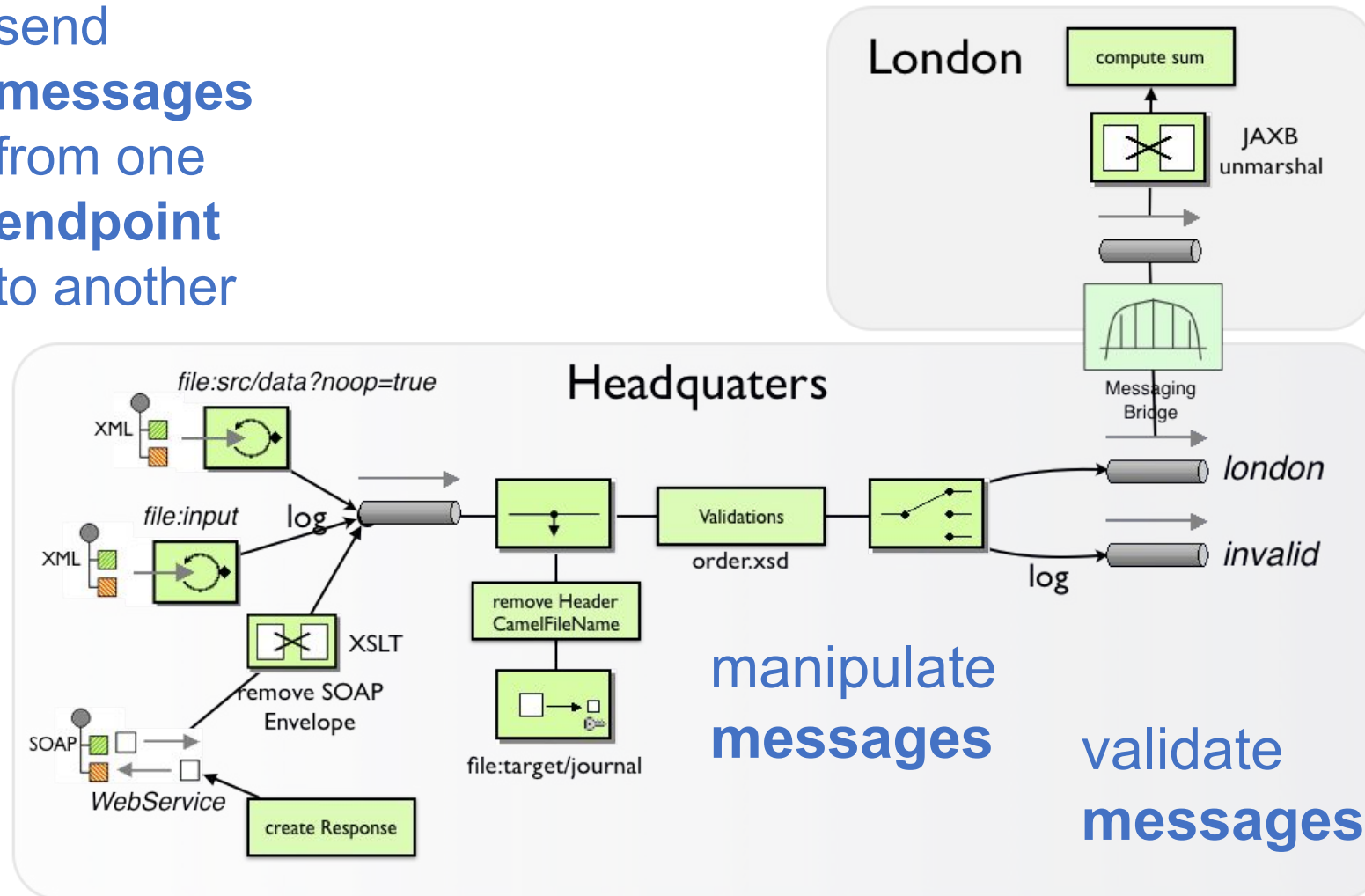
# What is Camel?

- Camel is
  - An integration framewok
  - A tool to implement integration patterns
  - A set of Java Libraries
- Camel is **not**
  - An Enterprise Service Bus (ESB)
  - A service runtime



# What is an integration Problem?

send  
messages  
from one  
endpoint  
to another



# Routes and Endpoints

- A **route** is a sequence of steps that do something with a **message**
- Example: Take a file and put it into a queue

```
<route>  
  <from uri="file:src/data?noop=true"/>  
  <to uri="activemq:queue_vienna"/>  
</route>
```

Endpoint

```
from("file:src/data?noop=true")  
.to("activemq:queue_vienna");
```

# Example: basics\_01\_first\_route

# Camel Context

- In the Camel Context we can define routes, endpoints and many other things.
- It can be a Spring-Context or CDI-Context depending on the Runtime

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

  <camelContext xmlns="http://camel.apache.org/schema/spring">
    <package>at.gepardec.trainings.camel</package>
    <route id="_route1">
      <from uri="file:src/data?noop=true"/>
      <to uri="file:src/result"/>
    </route>
  </camelContext>
</beans>
```

# Java DSL

- Define routes in Java instead of XML
- Use one or more **RouteBuilder** to define routes

```
package at.gepardec.trainings.camel;

import org.apache.camel.builder.RouteBuilder;

public class MyRouteBuilder extends RouteBuilder {

    public void configure() {

        from("file:target/messages/others?noop=true")
        .to("file:target/messages/somewhere");
    }

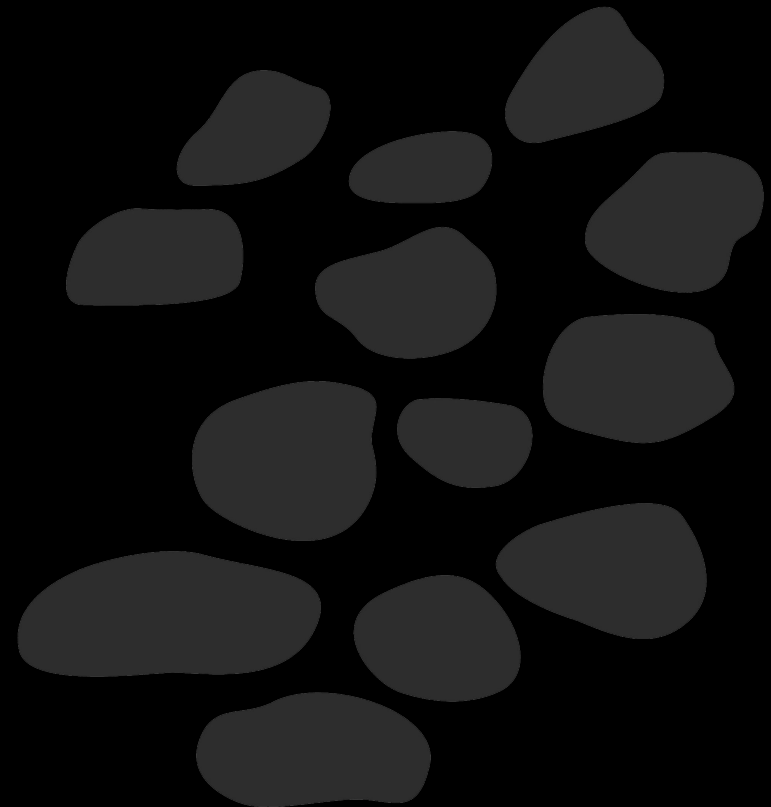
}
```

# Example: basics\_02\_1\_java\_dsl



*gepard*ec

Testen



# Testing Routes - Basic Tests

- With test library: `camel-test-spring`
- use JUnit Base Class: `extends CamelTestSupport`
- and override `createRouteBuilder()`

```
public class OrderProcessingTest extends CamelTestSupport {  
  
    @Override  
    protected RouteBuilder createRouteBuilder() {  
        return new RouteBuilder() {  
            public void configure() {  
                from("direct:start")  
                    .to(MyRoutes.URL_FILE_ORDERS_IN);  
  
                from(MyRoutes.URL_FILE_ORDERS_OUT)  
                    .to("mock:result");  
            }  
        };  
    }  
}
```

## Route to test

```
public void configure() {  
    from(MyRoutes.URL_FILE_ORDERS_IN)  
        .to(MyRoutes.URL_FILE_ORDERS_OUT);  
}
```

Mock component to check result

# Testing Routes - Basic Tests

- Get the `mock` endpoint from the route with `resolveMandatoryEndpoint()`
- Set expected values in mock endpoint
- Use Template to send message to endpoint
- Use `mock` endpoint to check results with `assertIsSatisfied()`

```
@Test
public void when_order_in_orders_message_is_in_processed() throws InterruptedException {
    String orderIn = "{\"partnerId\": 1, \"items\": [{ \"code\": 1, \"amount\": 110 }]}";
    String orderExpected = "{\"partnerId\":34,\"items\": [{ \"code\":1,\"amount\":110}]}";

    MockEndpoint resultEndpoint = resolveMandatoryEndpoint("mock:result", MockEndpoint.class);
    resultEndpoint.expectedMessageCount(1);
    resultEndpoint.expectedBodiesReceived(orderExpected);

    template.sendBody("direct:start", orderIn);
    resultEndpoint.assertIsSatisfied();
}
```

# Example: basics\_02\_2\_simple\_test

# Producer Templates

- Convenience class to send Messages to an endpoint.

## Direct Endpoint

- Endpoint for synchronous connection of routes within the same context.

```
from("file:data/in")
  .log("Got message: ${body}")
  .setBody(simple("my ${body}"))
  .to("file:data/out");
```



```
from("file:data/in")
  .log("Got message: ${body}")
  .to("direct:setBody");

from("direct:setBody")
  .setBody(simple("my ${body}"))
  .to("file:data/out");
```

# Testing Routes - Spring Runtime

- Use `org.apache.camel.test.spring.CamelSpringTestSupport` instead of `CamelTestSupport` (there are [other possibilities](#))
- Include the Spring context

```
@Override
protected AbstractXmlApplicationContext createApplicationContext() {
    return new ClassPathXmlApplicationContext("META-INF/spring/camel-context.xml");
}
```

- Inject `ProducerTemplate` and Endpoints in Tests

```
@Produce("direct:start")
private ProducerTemplate template;

@EndpointInject("mock:result")
private MockEndpoint resultEndpoint;
```

Example: basics\_02\_3\_spring\_test

# Extending Camel with Components

- Add ActiveMQ broker bean in Camel context:

```
<!-- Start an embedded artemis server, configured with broker.xml -->
<bean id="ActiveMQBroker" class="org.apache.activemq.artemis.core.server.embedded.EmbeddedActiveMQ"
      init-method="start" destroy-method="stop">
</bean>
```

- Configure the broker in broker.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<configuration xmlns="urn:activemq" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="urn:activemq /schema/artemis-configuration.xsd">
  <core xmlns="urn:activemq:core">
    <persistence-enabled>>false</persistence-enabled>
    <security-enabled>>false</security-enabled>
    <acceptors>
      <acceptor name="netty">tcp://0.0.0.0:61616</acceptor>
    </acceptors>
    </core>
  </configuration>
```

brokerURL



# Extending Camel with Components

- Add client component in context:

```
<bean id="activemq" class="org.apache.camel.component.jms.JmsComponent" depends-on="ActiveMQBroker">
  <property name="connectionFactory">
    <bean class="org.apache.activemq.artemis.jms.client.ActiveMQJMSConnectionFactory">
      <constructor-arg value="tcp://localhost:61616"/>
    </bean>
  </property>
</bean>
```

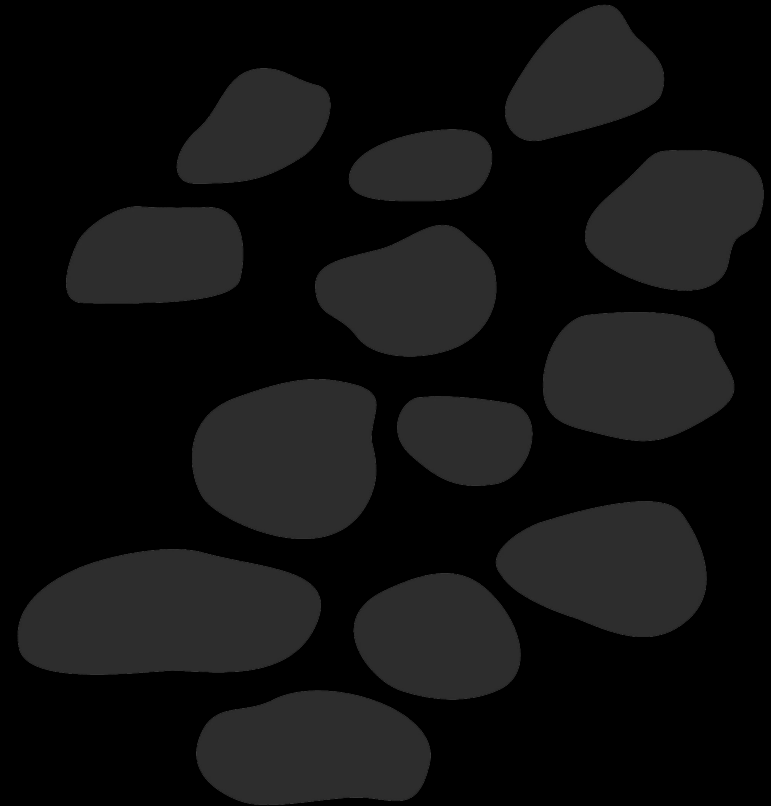
- Use the component in a route:

```
from("file:messages?noop=true")
  .to("activemq:wien");
```

# Example: basics\_03\_components

*gepard*ec

# Camel-Quarkus



# Camel Runtimes

- Dependency Injection frameworks
  - Spring
  - CDI
- Runtimes
  - Java SE / Unit Tests
  - JBoss EAP (Fuse, EOL June 2024)
  - Kubernetes (Camel K, Red Hat Build of)
  - Kafka (Camel Kafka Connector)
  - Quarkus (Camel Quarkus, Red Hat Build of)
  - Spring Boot (Camel Spring Boot, Red Hat Build of)
  - Karaf (Fuse, EOL June 2024)

# Camel Quarkus

- <https://camel.apache.org/camel-quarkus/latest>
- Create new Project:
  - Web: <https://code.quarkus.io/>
  - Examples: <https://github.com/apache/camel-quarkus-examples>
  - Maven:

```
mvn io.quarkus:quarkus-maven-plugin:3.8.3:create \
  -DprojectGroupId=at.gepardec.camel.quarkus \
  -DprojectArtifactId=myapp \
  -Dextensions=camel-quarkus-log,camel-quarkus-core,camel-quarkus-file
```

# Camel Quarkus

- Build and Run:
    - mvn compile
    - mvn quarkus:dev
    - mvn quarkus:build
    - mvn quarkus:run
- development mode
- normal mode

# Example: basics\_04\_quarkus

# XML DSL with Camel Quarkus

- Use [camel-quarkus-xml-io-dsl](#) in pom.xml:
- Configure location in application.properties:
- Configure routes in src/main/resources/routes/routes.xml

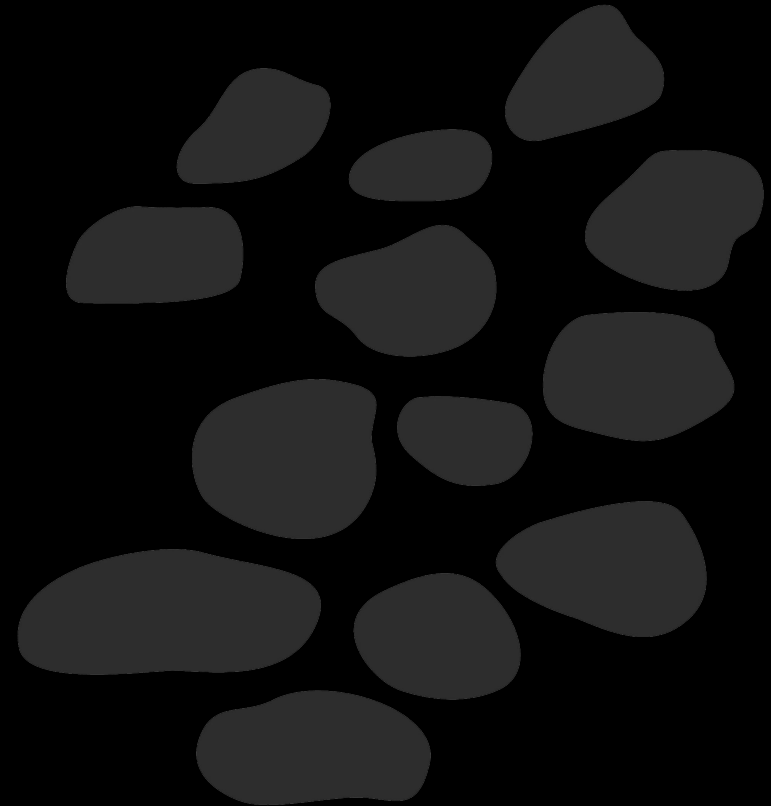
```
<dependency>  
  <groupId>org.apache.camel.quarkus</groupId>  
  <artifactId>camel-quarkus-xml-io-dsl</artifactId>  
</dependency>
```

```
camel.main.routes-include-pattern = routes/routes.xml
```

```
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xmlns="http://camel.apache.org/schema/spring"  
  xsi:schemaLocation="  
    http://camel.apache.org/schema/spring  
    http://camel.apache.org/schema/spring/camel-spring.xsd">  
  <route id="xml-route">  
    <from uri="file:target/messages/out"/>  
    <to uri="file:target/messages/next"/>  
  </route>  
</routes>
```



# Quarkus - Tests



# Testing Routes - Quarkus Tests

- With test library: camel-quarkus-junit5
- With JUnit Runner @QuarkusTest
- Inject Endpoints used in Test

```
<dependency>  
  <groupId>org.apache.camel.quarkus</groupId>  
  <artifactId>camel-quarkus-junit5</artifactId>  
  <scope>test</scope>  
</dependency>
```

```
@QuarkusTest  
public class FileRouteTest extends CamelQuarkusTestSupport {  
  
    @Produce("direct:start")  
    private ProducerTemplate startEp;  
  
    @EndpointInject("mock:result")  
    private MockEndpoint resultEndpoint;  
    ...  
}
```

# CamelQuarkusTestSupport

```
@QuarkusTest
public class FileRouteTest extends CamelQuarkusTestSupport {
    @Produce("direct:start")
    private ProducerTemplate startProducer;
    @EndpointInject("mock:result")
    private MockEndpoint resultEndpoint;
    @Override
    protected RouteBuilder createRouteBuilder() {
        return new RouteBuilder() {
            public void configure() throws Exception {
                from("direct:start").log("Got message: ${body}")
                    .to("file:target/messages/in");

                from("file:target/messages/out")
                    .to("mock:result");
            }
        };
    }
    @Test
    public void test_message_goes_from_in_to_out() throws InterruptedException {
        String msgIn = "Gepardec";
        String msgExpected = "Hello Gepardec!";
        resultEndpoint.expectedMessageCount(1);
        resultEndpoint.expectedBodiesReceived(msgExpected);
        startProducer.sendBody("direct:start", msgIn);
        resultEndpoint.assertIsSatisfied();
    }
}
```

# Change Routes with Advice / weave

- Identify route by routeld
- Add Mock-Endpoint, replace endpoint by Mock-Endpoint or replace from-endpoint

```

@QuarkusTest
@TestInstance(Lifecycle.PER_CLASS)
@TestProfile(EggOrderRouteBuilderTest.class)
public class EggOrderRouteBuilderTest extends CamelQuarkusTestSupport {
...
    @BeforeAll
    public void beforeAll() throws Exception {
        AdviceWith.adviceWith(context, EntryRouteBuilder.REST_ENDPOINT_ROUTE_ID,
            a -> a.weaveByToUri(EggOrderRouteBuilder.OUTPUT_JMS_ENDPOINT_URI)
                .replace()
                .to("direct:EggOrderRouteBuilderTestDirectResult")
        );
    }
}

```

setup a new context after each test-class

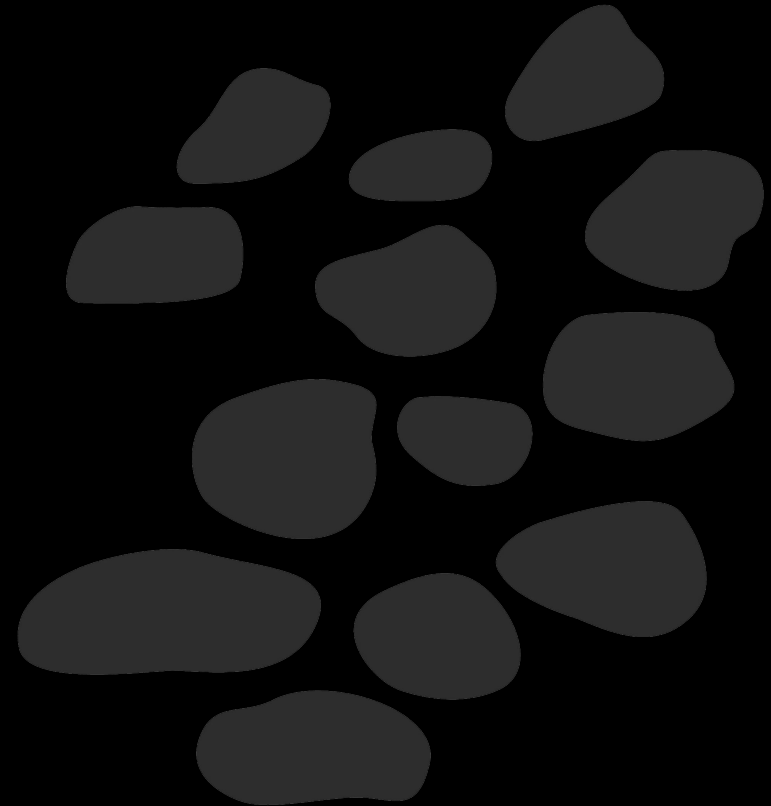
from(fromEndpoint).routeId(**REST\_ENDPOINT\_ROUTE\_ID**)  
.to(**OUTPUT\_JMS\_ENDPOINT\_URI**)

- Here the JMS-endpoint will be replaced by a direct endpoint in a test-route

# Example: basics\_06\_1\_unit\_tests

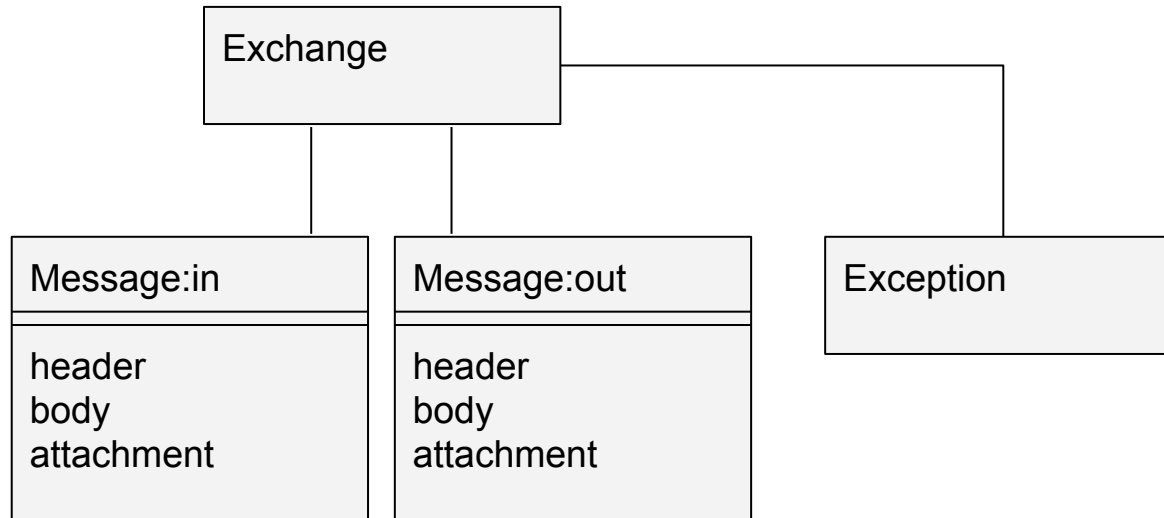
*gepard*ec

# Exchange - Processors



# Exchange - A Core Structure

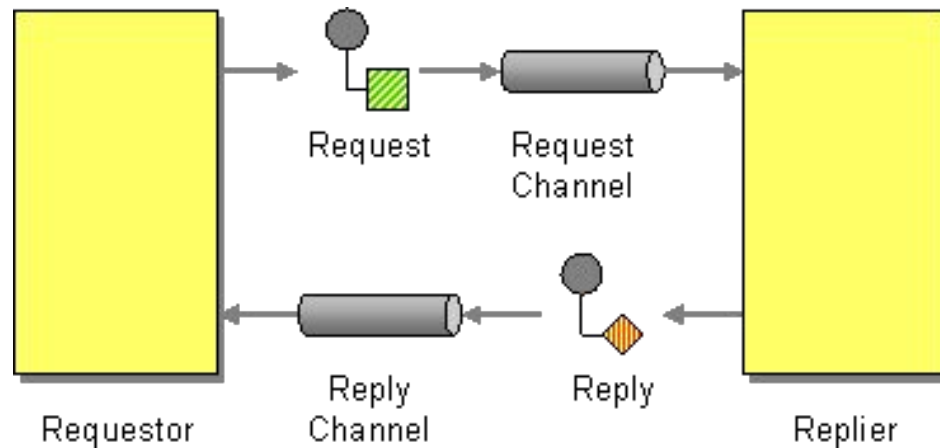
- See <https://camel.apache.org/manual/latest/faq/using-getin-or-getout-methods-on-exchange.html>
- Camel passes an Exchange object to endpoints, not messages!
- `org.apache.camel.ExchangePattern`



- There are different Exchange-Patterns:
  - InOnly (Event Messages) e.g for JMS messaging
  - InOut (Request-Reply) e.g. For webservice calls

# Exchange Patterns

- Property of an exchange. Usually set in the first endpoint
- InOnly: Send a message and don't expect a reply (e.g. JMS endpoints)
- InOut: Send a message and expect a reply (e.g. REST endpoints)
- What happens when you send an InOut exchange to a JMS-Endpoint?



- If you don't want to use this Request-Reply Pattern, change the ExchangePattern in the route
  - `setExchangePattern(ExchangePattern.InOnly)`



# Processor

- Interface to manipulate an exchange

A processor is used to implement the Event Driven Consumer and Message Translator patterns and to process message exchanges.

```
public class SimpleProcessor implements Processor {  
    @Override  
    public void process(Exchange exchange) throws Exception {  
        String body = exchange.getIn().getBody(String.class);  
        exchange.getIn().setBody("Hello " + body);  
    }  
}
```



# Why?

// Implement a Consumer for the Message Exchange

// Change the Message before the to-part

// Do something that's not possible with the Camel-EIP

**DO NOT MISUSE IT!!!**



# How?

// Use this inside a route by declaring the bean in Spring

```
<bean id="myProcessor" class="com.acme.MyProcessor" />
```

// Define a variable

// Define an anonymous inner class

// Define a lambda

# Code/Syntax

registered in Camel context

```
from("activemq:myQueue").process("myProcessor");
```

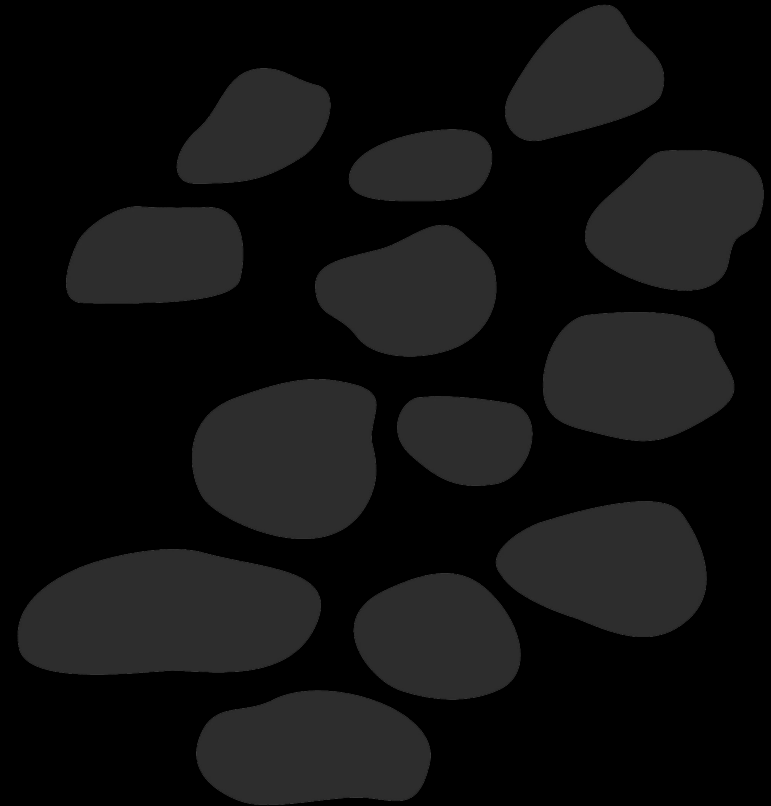
```
Processor myProcessor = new MyProcessor();  
...  
from("activemq:myQueue").process(myProcessor);
```

```
from("activemq:myQueue").process(new Processor() {  
    public void process(Exchange exchange) throws Exception {  
        String payload = exchange.getIn().getBody(String.class);  
        // do something with the payload and/or exchange here  
        exchange.getIn().setBody("Changed body");  
    }  
}).to("activemq:myOtherQueue");
```

```
from("activemq:myQueue")  
    .process(exchange -> exchange.getIn().setBody("Changed body"))  
    .to("activemq:myOtherQueue");
```

# Example: basics\_07\_1\_processor

# Basic Endpoints



# Direct Endpoint

```
<dependency>
  <groupId>org.apache.camel.quarkus</groupId>
  <artifactId>camel-quarkus-direct</artifactId>
</dependency>
```

- Endpoint to connect routes synchronously in the same camel context
- Used to bring more structure into complex routes

```
from("direct:startProcessorTest")
  .log("start")
  .process(new SimpleProcessor())
  .choice().
    when(xpath("/person/city = 'London'"))
      .to("activemq:london.records").
    otherwise()
      .to("activemq:invalid.records")
  .log("end")
  .to(resultEndpoint);
```



```
from("direct:startProcessorTest")
  .log("start")
  .process(new SimpleProcessor())
  .to("direct:choice");

from("direct:choice")
  .choice().
    when(xpath("/person/city = 'London'"))
      .to("activemq:london.records").
    otherwise()
      .to("activemq:invalid.records")
  .log("end")
  .to(resultEndpoint);
```

## Seda Endpoint

- Similar to 'direct', but asynchronously

```
<dependency>  
  <groupId>org.apache.camel.quarkus</groupId>  
  <artifactId>camel-quarkus-seda</artifactId>  
</dependency>
```

## Log Endpoint

- Log exchange infos

```
<dependency>  
  <groupId>org.apache.camel.quarkus</groupId>  
  <artifactId>camel-quarkus-log</artifactId>  
</dependency>
```

```
to("log:org.apache.camel.example?level=DEBUG")
```

- Alternative:

```
.log("This is the body: ${body}")
```



# Example: basics\_07\_2\_direct\_seda

*gepard*ec

# Data Formats and Transformations



# Data Formats, e.g. JSON

- Use a library to process json: `camel-jackson`
- Marshal: Transform Java Object to String
- Unmarshal: Transform String to Java Object

## to marshal

aufstellen

arrangieren

ordnen

einweisen [[Fahrzeuge etc.](#)]

```
JacksonDataFormat orderFormat = new JacksonDataFormat(Order.class);
```

```
from(URL\_FILE\_ORDERS\_IN) Message body is String
```

```
.unmarshal(orderFormat)
```

```
.to(DIRECT\_ORDER\_IN);
```

Message body is Java Object (Order)

```
from(DIRECT\_ORDER\_IN)
```

```
.marshal(orderFormat)
```

```
.to(URL\_FILE\_ORDERS\_OUT);
```

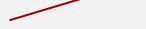
Message body is Java Object (Order)

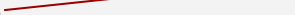
Message body is String

# Marshal with predefined DataFormats

- Use marshal() and unmarshal() without Arguments
- Use function for datatype (e.g. arvo(), bindy(), csv(), json()...)
- Adding data format library (e.g. jackson) is still needed

or

```
from(URL_FILE_ORDERS_IN)
  .unmarshal().json(Order.class)  unmarshal class
  .to(DIRECT_ORDER_IN);

from(DIRECT_ORDER_IN)
  .marshal().json(true)  prettyPrint
  .to(URL_FILE_ORDERS_OUT);
```

```
JsonDataFormat jsonOrderToProducer =
  dataFormat()
  .json()
    .unmarshalType(OrderToProducer.class)
    .prettyPrint(true)
  .end();
```

# Example: basics\_08\_1\_marshall

# Type Converters

- Convert e.g: Domain Objects to DTOs, XML to JSON,...

```
import org.apache.camel.Converter;  
import org.apache.camel.TypeConverters;
```

```
public class OrderTypeConverters implements TypeConverters {
```

```
    @Converter
```

```
    public static OrderItemDto toDto(OrderItem item) {  
        return new OrderItemDto(item.getCode(), item.getAmount());  
    }
```

```
    @Converter
```

```
    public static OrderItem fromDto(OrderItemDto item) {  
        return new OrderItem(item.getCode(), item.getAmount());  
    }
```

```
}
```

Marker Interface

annotate each converter

# Type Converters

- Register type converters in CamelContext

```
context().getTypeConverterRegistry().addTypeConverters(new OrderTypeConverters());
```

- explicit transformation

```
from("direct:convert")  
  .convertBodyTo(OrderItemDto.class)  
  .to("mock:result");
```

Body is OrderItem.class

Body is OrderItemDto.class

```
from("direct:inputTypeConverter").inputType(OrderItemDto.class)  
  .to("mock:result");
```

- implicit transformation

```
from("direct:bean")  
  .bean(new OrderDtoProcessor())  
  .to("mock:result");
```

```
public class OrderDtoProcessor {  
  
    @Handler  
    public OrderItemDto processDto(OrderItemDto dto) {  
        dto.amount++;  
        return dto;  
    }  
}
```

# Example: basics\_08\_2\_type\_converter



# Auto-Register Type Converters in Quarkus

- Annotate class with @Converter

```
import org.apache.camel.Converter;
import org.apache.camel.TypeConverters;

@Converter
public class OrderTypeConverters implements TypeConverters {

    @Converter
    public static OrderItemDto toOrderItemDto(OrderItem item) {
        return new OrderItemDto(item.getCode(), item.getAmount());
    }

    ...
}
```

# Transformers

- Convert types (similar to Type Converters)
  - from specified inputType
  - to specified inputType
- Register in CamelContext
- Use implicitly

```
transformer()
    .fromType(OrderItem.class)
    .toType(OrderItemDto.class)
    .withJava(OrderItemTransformer.class);
```

```
from("direct:inputTypeTransformer").inputType(OrderItemDto.class)
    .to("mock:result");
```

- Or use explicitly

```
from(...) // In comes "json:OrderToProducer"
    .transform(new DataType("json:OrderToProducer"), new DataType(OrderToProducer.class))
    .to("mock:result");
```

# Transformers

- Data Format Transformer

```
JsonDataFormat jsonOrderToProducer =  
    dataFormat()  
    .json()  
    .unmarshalType(OrderToProducer.class)  
    .prettyPrint(true)  
    .end();  
  
transformer()  
    .fromType("json:OrderToProducer")  
    .toType(OrderToProducer.class)  
    .withDataFormat(jsonOrderToProducer);
```

# Transformers

- Endpoint Transformer

```
transformer()
    .fromType("xml:ABCOrder")
    .toType("xml:XYZOrder")
    .withUri("xslt:transform.xsl");
```

- Custom Transformer

```
transformer()
    .fromType(OrderItem.class)
    .toType(OrderItemDto.class)
    .withJava(OrderItemTransformer.class);
```

```
public class OrderItemTransformer extends Transformer{

    @Override
    public void transform(Message message, DataType from, DataType to) throws Exception {
        if ( from.getName().equals(OrderItem.class.getName()) &&
            to.getName().equals(OrderItemDto.class.getName())
        ){
            message.setBody(OrderTypeConverters.toDto(message.getBody(OrderItem.class)));
        }
    }
}
```

# DataTypeAware Messages

- Problem: Java type of body is not sufficient to determine the message type:
  - e.g: Order in JSON format is `String.class` and Order in XML is `String.class`
  - Want transform to Order from JSON to XML!
- Solution:
  - Add explicit `DataType` to a Message
  - Messages in exchange (e.g: `ex.getIn()`) implement `DataTypeAware`

# DataTypeAware Messages

```
DataType type = new DataType("json:OrderToProducer");
```

```
private void sendTo(String endpointUri, Object msgIn, DataType type) {  
    startProducer.send(endpointUri,  
        ex -> {  
            ((DataTypeAware)ex.getIn())  
                .setBody(msgIn, type);  
        }  
    );  
}
```

```
public interface DataTypeAware {  
    void setDataType(DataType type);  
    DataType getDataType();  
    boolean hasDataType();  
    void setBody(Object body, DataType type);  
}
```

Example: best\_13\_2\_transform\_with\_datatype