Erläuterung des Demo Codes

Klasse CamelStarter

```java
package camel;
import org.apache.camel.CamelContext;
import org.apache.camel.impl.DefaultCamelContext;

public class CamelStarter {

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

        // Create a CamelContext (which is Camel's runtime system)
        CamelContext context = new DefaultCamelContext();
        context.addRoutes(new IntegrationRoute());

        // After starting, you have 30 seconds to copy files to the inbox.
        // They are automatically processed by the route.
        context.start();

        Thread.sleep(30000);

        context.stop();

    }

}
```

Die Klasse CamelStarter initialisiert ein Objekt des Typs CamelContext und lädt die in der Klasse „IntegrationRoute“ definierte Route in den Camel Context. Der Camel Context ist das Runtime System von Camel und wird deshalb in der main-Methode des Projekts deklariert und initialisiert. Anschließend wird der Context und somit die Abarbeitung der Route gestartet und nach 30 Sekunden wieder getoppt, da die Route andernfalls unendlich lange weiterlaufen würde.

Klasse IntegrationRoute

```java
package camel;
import org.apache.camel.builder.RouteBuilder;

public class IntegrationRoute extends RouteBuilder {

    @Override
    public void configure() throws Exception {

        // Consumer Endpoint
        from("file:orders/inbox?noop=true")
            // if tracing is turned on, you get very detailed i
            //.tracing()
            // a processor does custom integration logic
            .process(new LoggingProcessor())
            // a bean also does custom integration logic
            .bean(new TransformationBean(), "makeUpperCase")
            // Translator EIP (using the camel-csv component)
            .unmarshal().csv()
            // Splitter EIP
            .split(body().tokenize(","))
            // Content-based Router EIP
            .choice()
                .when(body().contains("DVD"))
                    // Producer Endpoint
                    .to("file:orders/outbox/dvd")
                .when(body().contains("CD"))
                    .to("file:orders/outbox/cd")
                .otherwise()
                    .to("mock:others");
    }

}
```

Die Klasse IntegrationRoute beinhaltet die Integrationslogik in Form einer Camel Route. Diese wird zur Ausführung in den Camel Context geladen.

Zur Definition einer Route in einer Java Klasse, muss die Klasse von der Klasse RouteBuilder erben und die Methode „public void configure“ überschreiben. In dieser Methode wird die eigentliche Integrationslogik implementiert. Durch „from(…)“, definiert man den Anfang der Route und somit von wo die Daten zur Verarbeitung in der Route konsumiert werden sollen. In diesem Projekt wird eine Datei, was durch das Schlüsselwort „file“ gekennzeichnet wird, aus dem Ordner „orders/inbox“ konsumiert. Der Parameter „?noop=true“ sorgt dafür, dass die Datei anschließend nicht durch Camel verschoben wird.

Die konsumierte Datei wird anschließend an einen Processor geleitet, welcher in der Klasse LoggingProcesor definiert ist. Anschließend wird die konsumierte Datei zur weiteren Verarbeitung an die in der Klasse TransformationBean definierten Bean weitergeleitet. Danach wird der Inhalt der Datei mit dem Befehl „.unmarshal().csv()“ konvertiert und mit dem Befehl „.split(body().tokenize(„,“)) in neue Teil Strings zerlegt. Die so entstehenden Teilstrings werden anschließend anhand ihres Inhalts in eine Datei im Ordner „DVD“, „CD“ oder einem Mock „others“ geschrieben.

Klasse LoggingProcessor

```java
package camel;
import org.apache.camel.Exchange;
import org.apache.camel.Processor;

public class LoggingProcessor implements Processor {

    @Override
    public void process(Exchange exchange) throws Exception {

        System.out.println("Received Order: " + exchange.getIn().getBody(String.class));

    }

}
```

Die Klasse LoggingProcessor implementiert das Interface Processor. Somit kann die Klasse in der Camel Route durch den Befehl „.process(new LoggingProcessor())“ dazu verwendet werden den Inhalt der sich in der Route befindenden Daten zu „manipulieren“ und somit eigene Implementierungslogik in die Route einfließen lassen.

Die Klasse dient in diesem Beispiel dazu den Inhalt der konsumierten Datei auf der Konsole auszugeben. Dazu wird zunächst die eingehende Nachricht über das Objekt Exchange und anschließend der Nachrichteninhalt abgerufen.
Die Variable „Exchange exchange“ repräsentiert den Exchange Container, welcher sämtliche Informationen über die aktuellen Daten in der Route enthält, wie beispielsweise eine eindeutige ID, Exceptions sowie die ein- und ausgehenden Nachrichten.

Klasse TransformationBean

```java
package camel;
public class TransformationBean {

    public String makeUpperCase(String body) {
        return body.toUpperCase();
    }

}
```

Die Klasse TransformationBean, wird als Bean in der Camel Route mit der Methode „public String makeUpperCase(String body)“ aufgerufen. Obwohl wir zu Beginn der Route eine Datei konsumieren, können wir die Methode mit einem String aufrufen. Dies ist aufgrund der zahlreichen automatischen Typkonvertierungen von Camel möglich.

Die Methode gibt den Inhalt der zu Beginn der Route eingelesenen Datei in Großbuchstaben zurück. Eine Bean gilt hierbei als Alternative zu einem Processor.