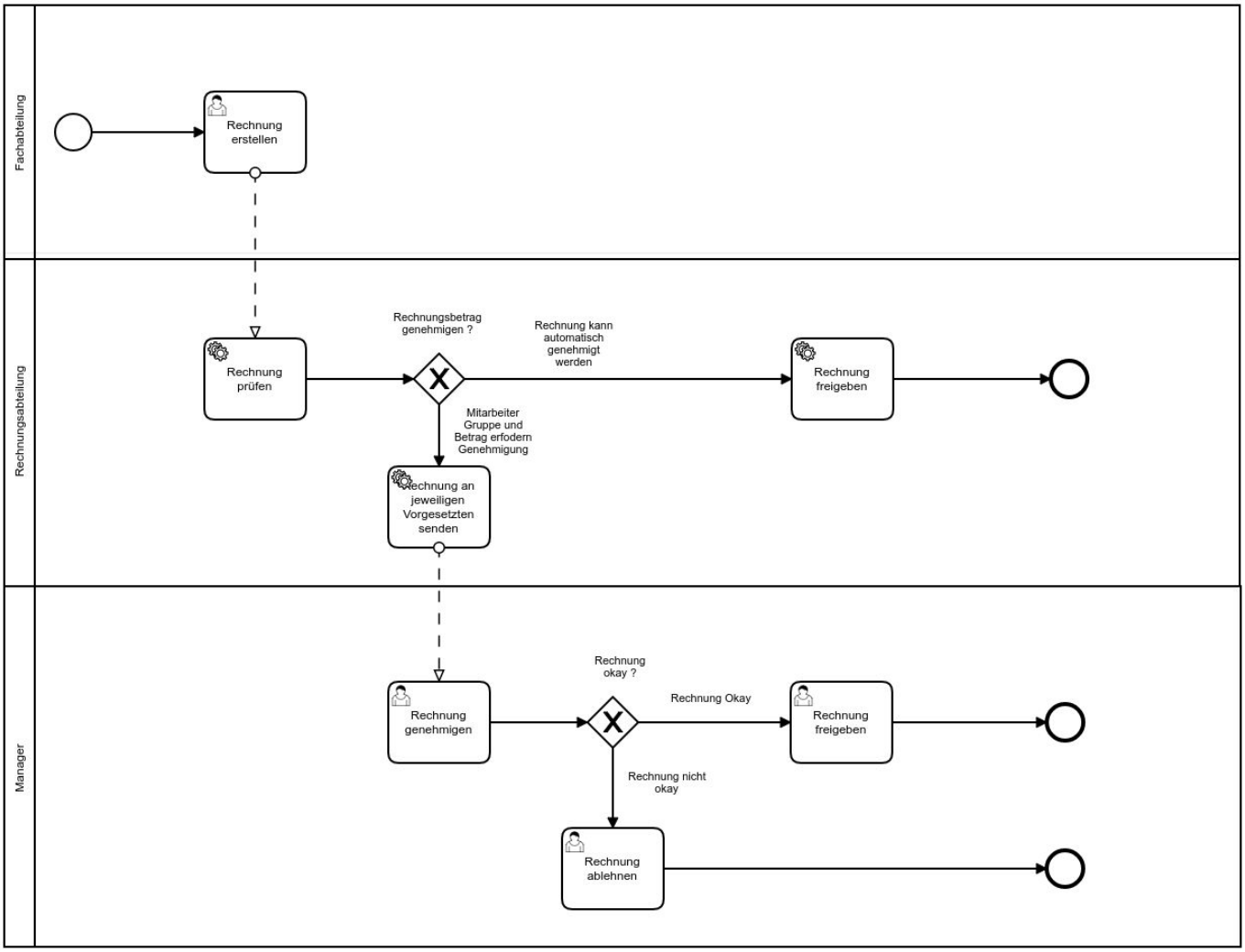
Dokumentenmanagement System **BPM Prozess** und Entscheidungen

In unserem Dokumenten Management System gibt es den Prozess für die Rechnungsfreigabe.
Rechnungen sind Mitarbeitergruppen zugeordnet und haben einen Rechnungsbetrag.
Der Prozess im Dokumentenmanagement System sieht für diesen Prozess wie folgt aus.



Die Rechnungen werden in der Fachabteilung von Mitarbeitern erstellt, die die Verträge aushandeln.
Die Rechnungen werden im Dokumentenmanagement System dann an die Rechnungsabteilung weitergeleitet durch einen automatischen Service, der in Camunda BPM realisiert ist.
Dieser Service prüft mit der DMN Tabelle die Eingabewerte und genehmigt die Rechnung entweder automatisch, oder leitet diese an den entsprechenden Vorgesetzten weiter.

In der DMN Entscheidungstabelle wird geprüft in welcher Mitarbeitergruppe der Mitarbeiter eingeteilt ist und welchen Betrag die Rechnung hat.
Durch diese Werte wird die Rechnung entweder automatisch genehmigt oder an den entsprechenden Vorgesetzten weitergeleitet, der die Rechnung im Prozess freigeben muss.

DMN Tabelle

BerechtigungsMatrixDocManagement

F	Input +		Output +	Annotation
	Mitarbeiter	Summe Rechnung	Verantowrtlicher zur Rechnungsfreigabe	
	string	integer	string	
1	"Mitarbeiter Gruppe A"	<= 1000	"genehmigt"	wird automatisch genehmigt
2	"Mitarbeiter Gruppe A"	> 1000	"Vorgesetzer A"	-
3	"Mitarbeiter Gruppe B"	<= 500	"genehmigt"	wird automatisch genehmigt
4	"Mitarbeiter Gruppe B"	> 500	"Vorgesetzter B"	-
5	"Mitarbeiter Gruppe C", "Mitarbeiter Gruppe D"	> 0	"Vorgesetzter C"	-

Die Eingabe sind die Mitarbeiter Gruppe und der Rechnungsbetrag.

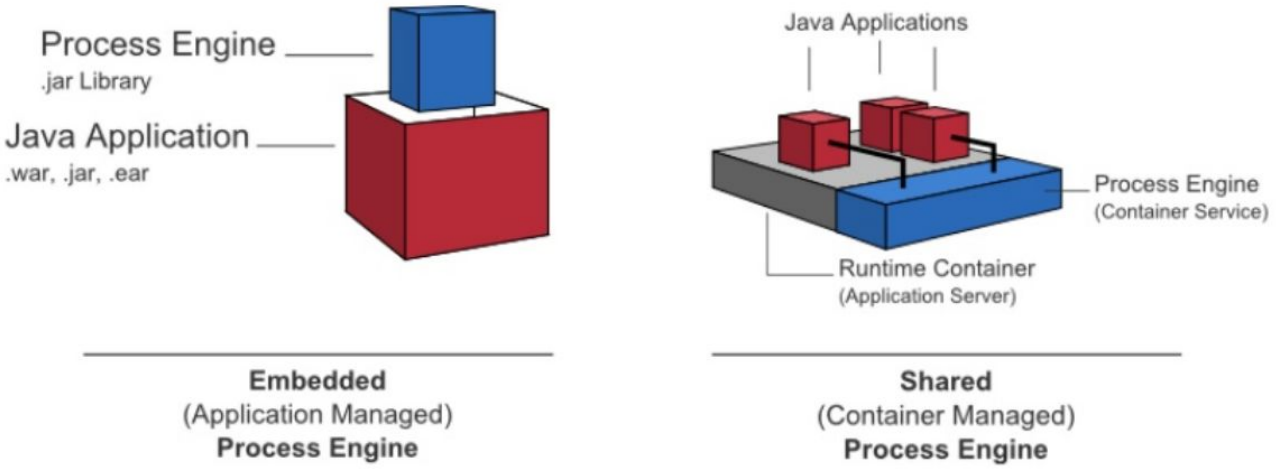
BerechtigungsMatrixDocManagement

F	Input +	
	Mitarbeiter	Summe Rechnung
	string	integer
1	"Mitarbeiter Gruppe A"	<= 1000
2	"Mitarbeiter Gruppe A"	> 1000
3	"Mitarbeiter Gruppe B"	<= 500
4	"Mitarbeiter Gruppe B"	> 500
5	"Mitarbeiter Gruppe C", "Mitarbeiter Gruppe D"	> 0

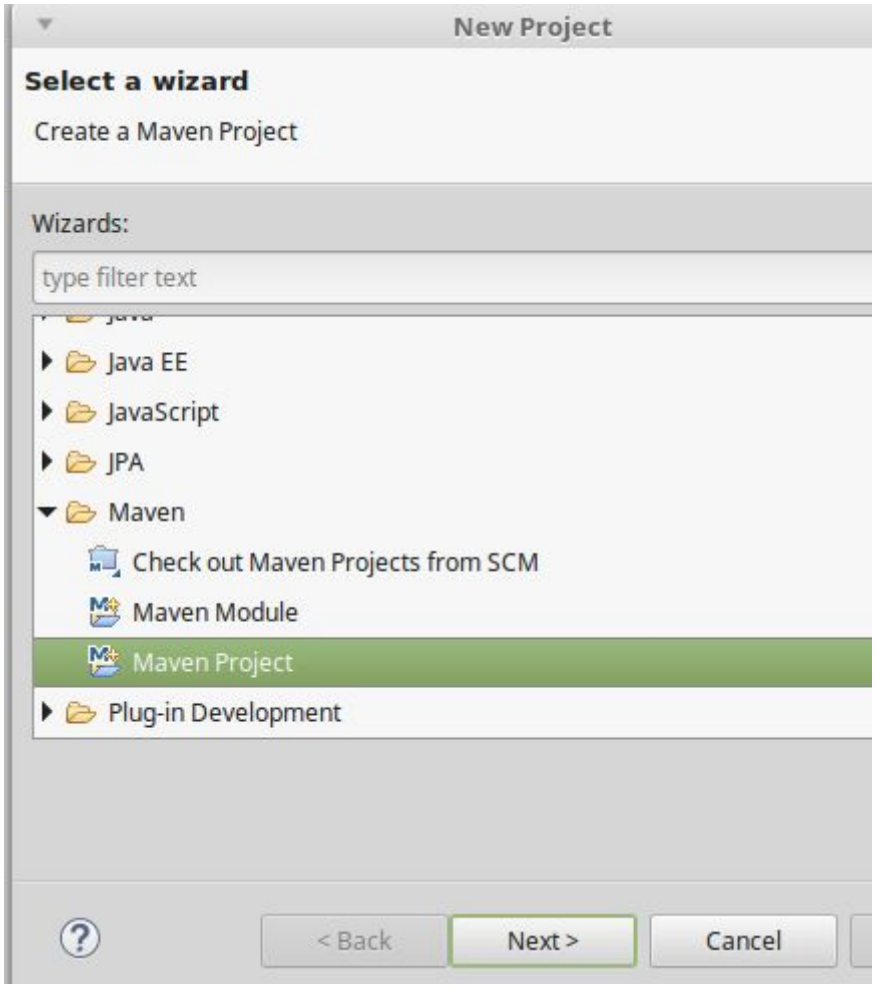
Output +	Annotation
Verantowrtlicher zur Rechnungsfreigabe	
string	
"genehmigt"	wird automatisch genehmigt
"Vorgesetzer A"	-
"genehmigt"	wird automatisch genehmigt
"Vorgesetzter B"	-
"Vorgesetzter C"	-

Zur Realisierung wird die BPM Engine von Camunda benutzt, die Möglichkeiten der BPM 2.0 Spezifikation benutzt. Das erstellte BPM wird in Eclipse importiert und es werden ie Camunda Bibliotheken als Plugin installiert und dann können wir Lanes aus dem Prozessmodell ausführbar einstellen und über die Programmiersprache Java Services aufrufen und automatische Logik in den Prozess integrieren.

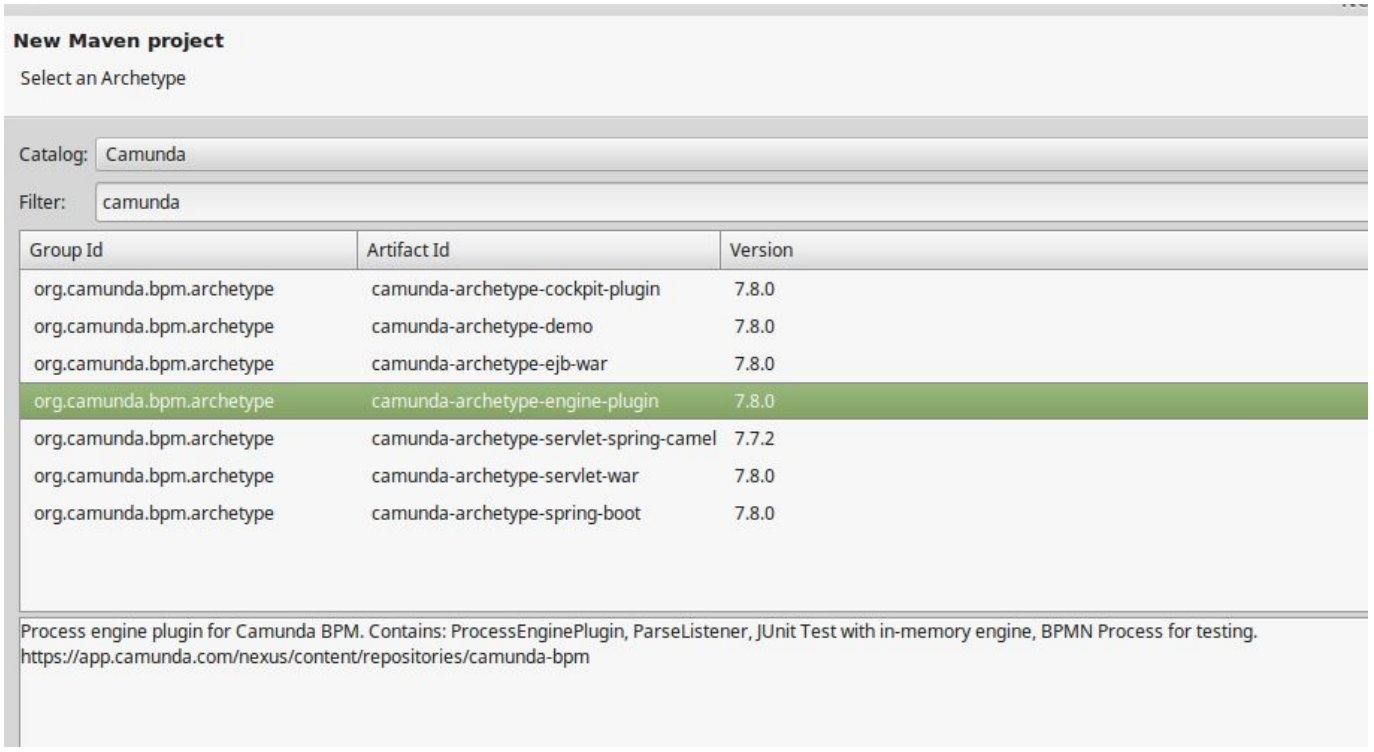
Features - Platform: Execute BPMN 2.0

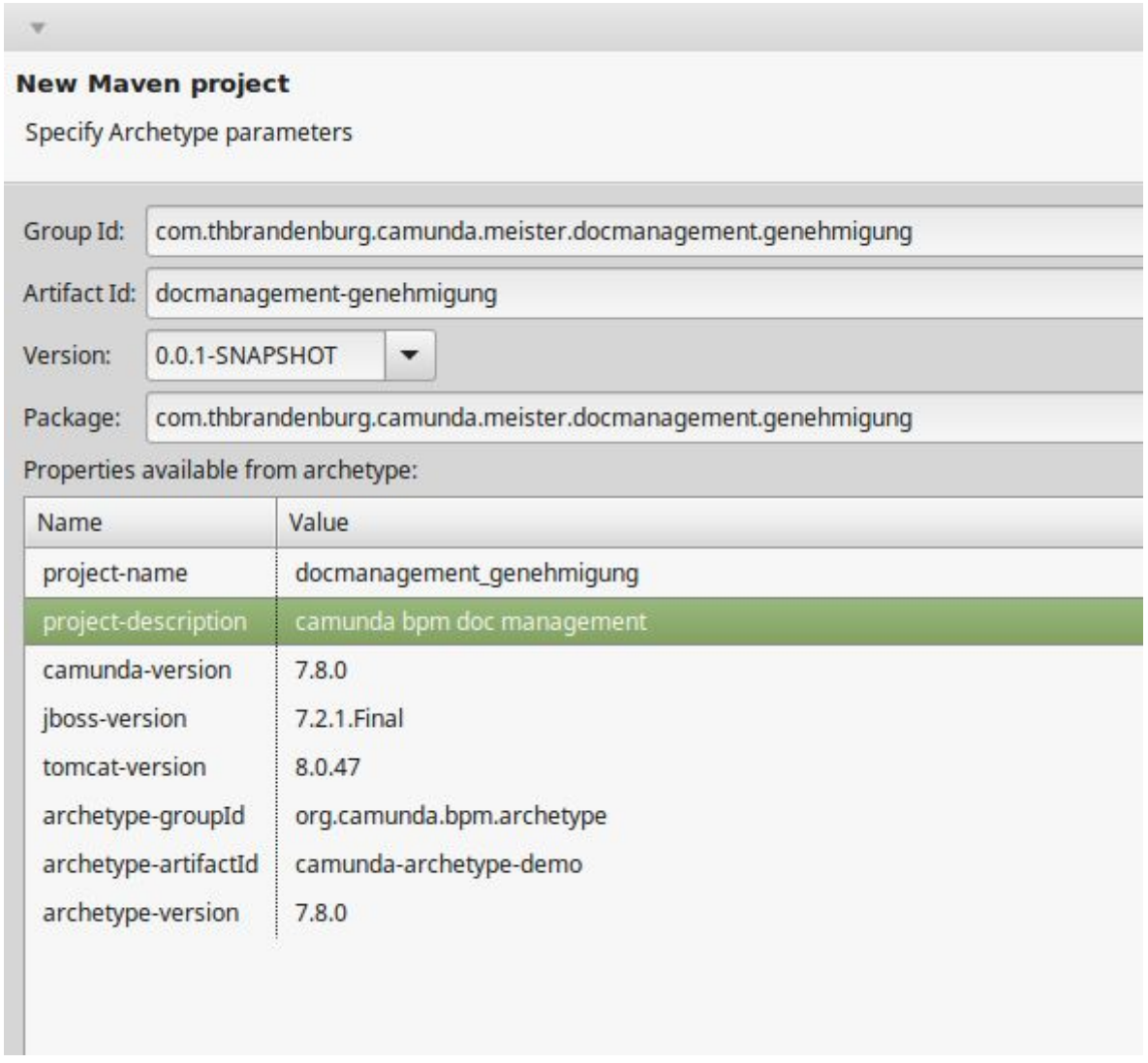


In Eclipse erstellen wir dazu ein Maven Projekt:



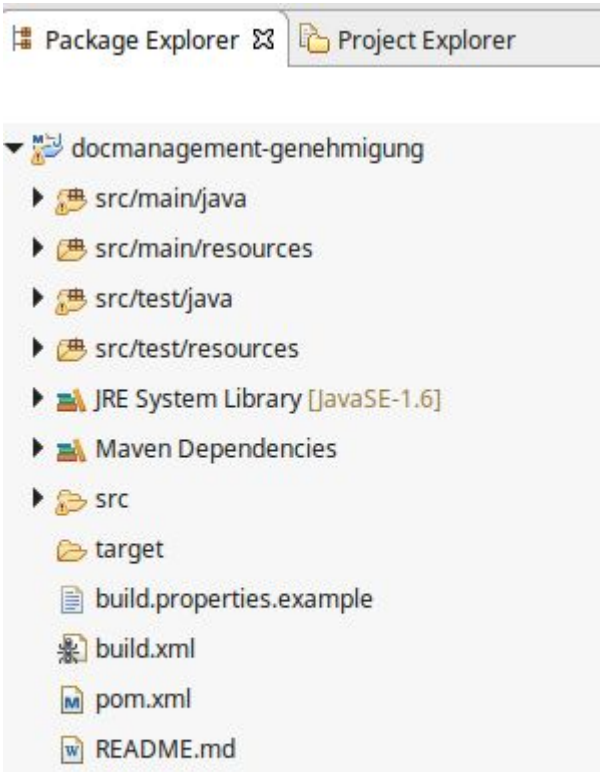
Und nutzen die Camunda Plugins um die Entsprechende Projektstruktur automatisch zu erstellen:



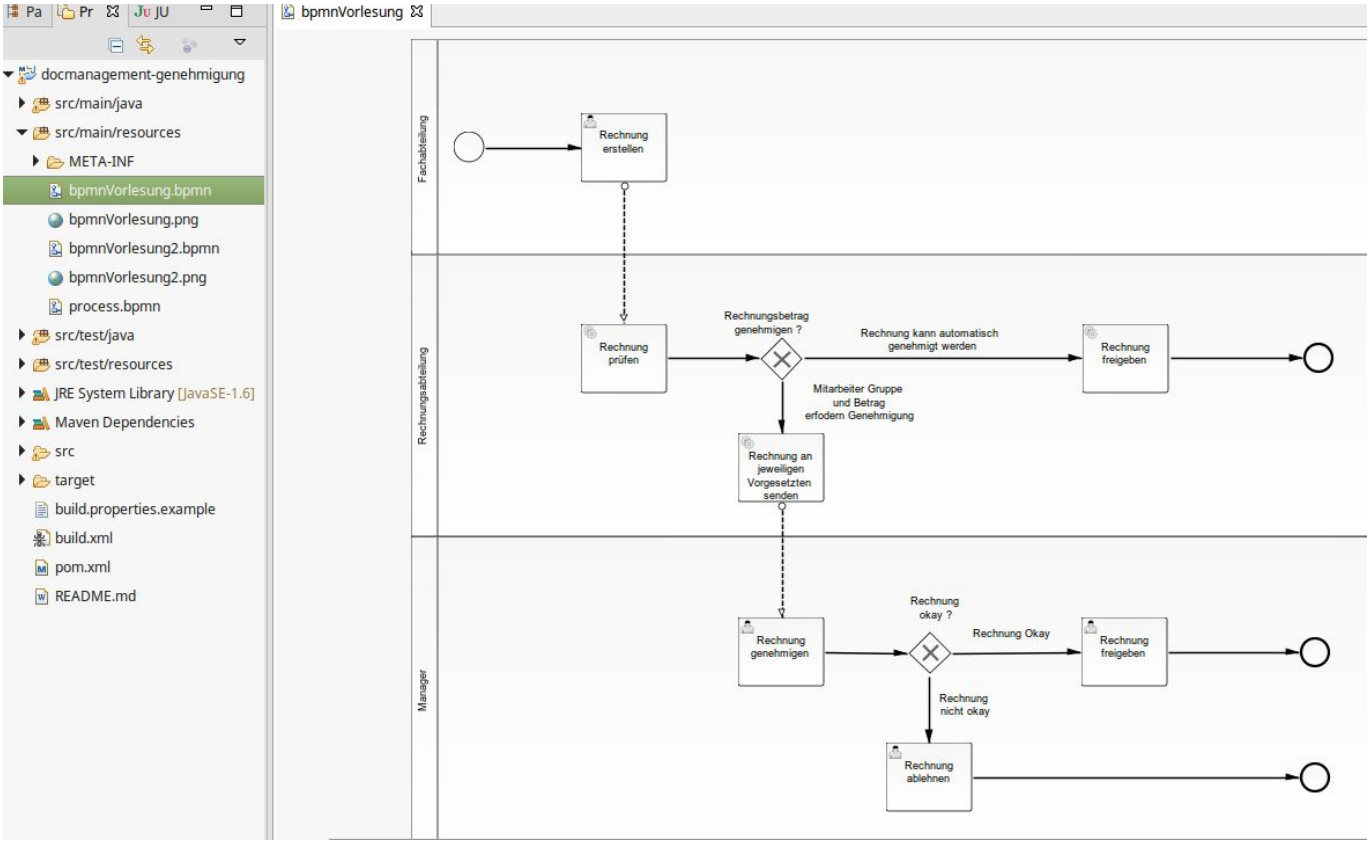


Die Camunda Maven Archetypes können hier bezogen werden, diese können dann in Eclipse als Software installiert werden:
https://docs.camunda.org/manual/7.4/user-guide/process-applications/maven-archetypes/

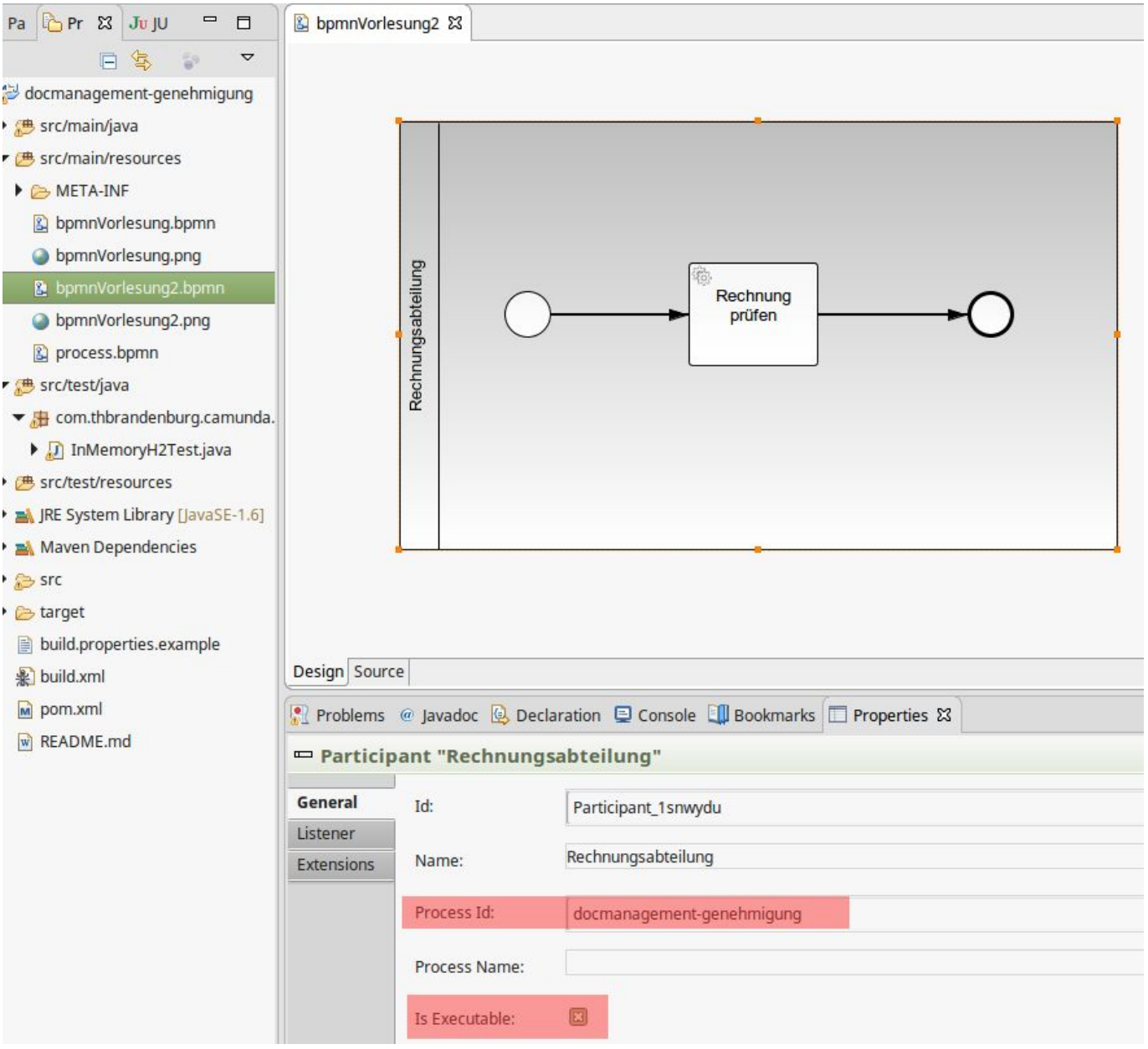
Dadurch wird die Projektstruktur automatisch erstellt



und wir können im nächsten Schritt unser BPM aus Camunda importieren:



Nun kommen die Eclipse spezifischen Einstellungen zum Tragen, durch die wir die Lane automatisch ausführbar machen.
Um diesen Prozess an einem einfachen Beispiel zu zeigen und ein funktionierenden Prototypen zu haben, **werden wir am Anfang mit einem vereinfachten Beispiel einer Rechnungsprüfung arbeiten.**

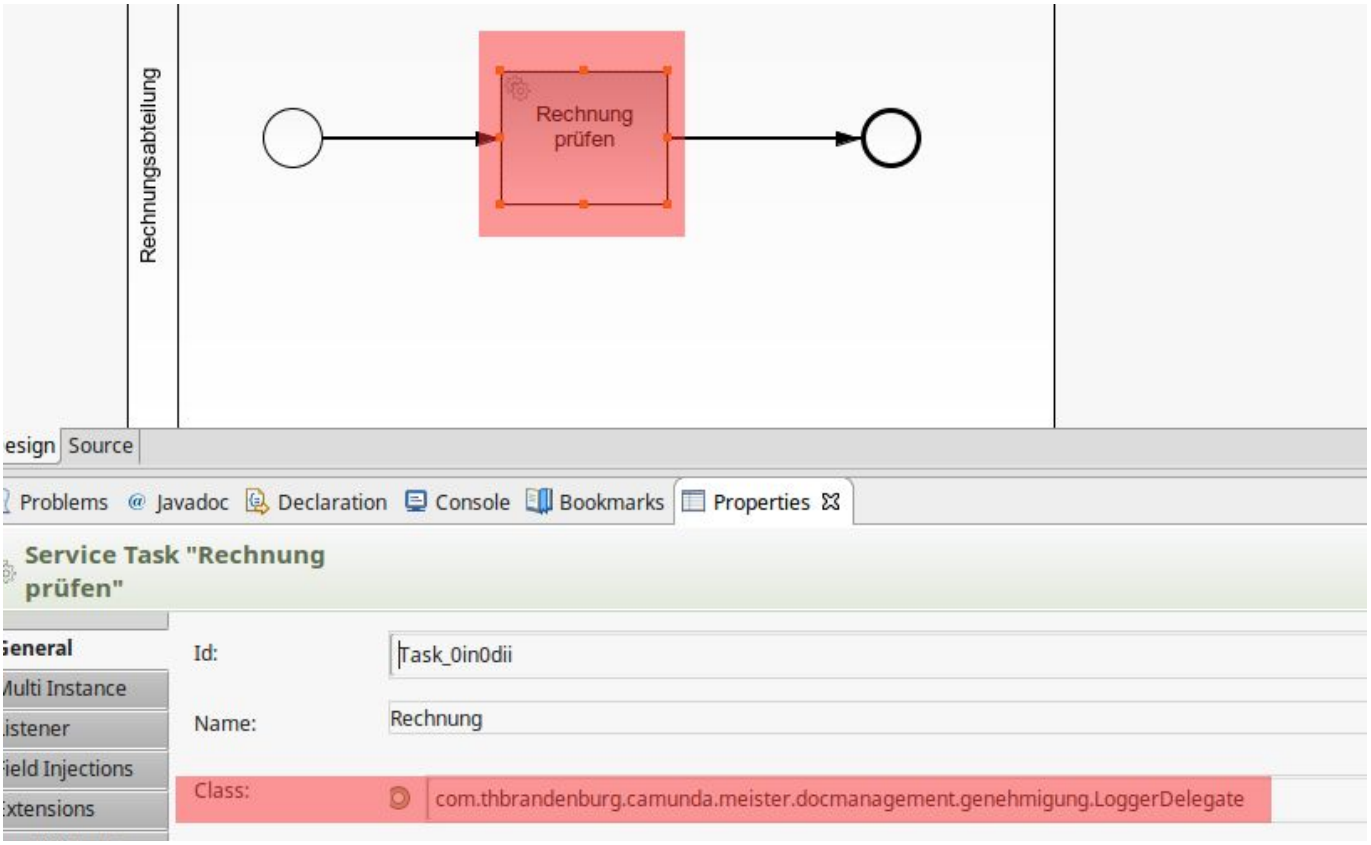


Die Lane wird als “ausführbar” markiert und die Prozess ID ist wichtig, da wir diese nun im Java Code benutzen, um eine Instanz der prozess Engine zu erstellen.

Prozess ID: docmanagement-genehmigung
BPM Datei: bpmnVorlesung2.bpmn

Diese Informationen werden benutzt um nun in einem Test zu schauen, ob das BPM okay ist und eine Prozess Engine daraus erstellt werden kann. In Eclipse wird ein sogenannter Unit Test ausgeführt, der auf das BPM verweist und unsere Prozess Id benutzt um die Engine zu erstellen.

Darüber hinaus müssen allen Service Aktivitäten im Prozess direkte Java Klassen zugeordnet werden, die dann bei der Aktivität aufgerufen werden und Java Logik ausführen können.



Die verknüpfte Klasse muss die Camunda Bibliotheken importieren, von der Camunda Klasse JavaDelegate erben und die “public void” Methode execute mit dem Parameter für die Ausführung haben:

bpmnVorlesung2 | InMemoryH2Test.java | *LoggerDelegate.java

```java
package com.thbrandenburg.camunda.meister.docmanagement.genehmigung;

import java.util.logging.Logger;
import org.camunda.bpm.engine.delegate.DelegateExecution;
import org.camunda.bpm.engine.delegate.JavaDelegate;
/**
 * This is an empty service implementation illustrating how to use a plain Java
 * class as a BPMN 2.0 Service Task delegate.
 */
public class LoggerDelegate implements JavaDelegate {

  private final Logger LOGGER = Logger.getLogger(LoggerDelegate.class.getName());

  public void execute(DelegateExecution execution) throws Exception {
    LOGGER.info("\n\n  ... LoggerDelegate invoked by "
            + "processDefinitionId=" + execution.getProcessDefinitionId()
            + ", activtyId=" + execution.getCurrentActivityId()
            + ", activtyName='" + execution.getCurrentActivityName() + "'"
            + ", processInstanceId=" + execution.getProcessInstanceId()
            + ", businessKey=" + execution.getProcessBusinessKey()
            + ", executionId=" + execution.getId()
            + " \n\n");
  }
}
```

Die Methode wird dann von der Prozess Engine aufgerufen.

Für unsere Tests benutzen wir nun erst einmal die Java Unit Test Methoden, die prüfen, ob alles okay ist.

bpmnVorlesung2 — InMemoryH2Test.java

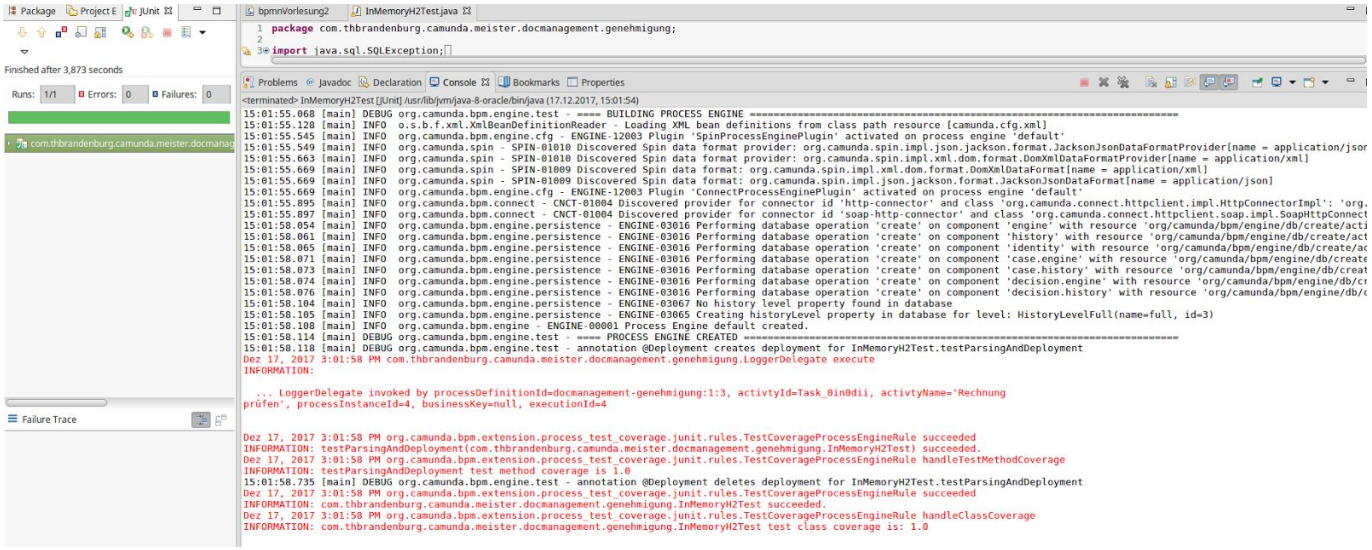
```
1  package com.thbrandenburg.camunda.meister.docmanagement.genehmigung;
2
3  import java.sql.SQLException;
17
18 /**
19  * Test case starting an in-memory database-backed Process Engine.
20  */
21 public class InMemoryH2Test {
22
23   @ClassRule
24   @Rule
25   public static ProcessEngineRule rule = TestCoverageProcessEngineRuleBuilder.create().build();
26
27   private static final String PROCESS_DEFINITION_KEY = "docmanagement-genehmigung";
28
29   static {
30     LogFactory.useSlf4jLogging(); // MyBatis
31   }
32
33   @Before
34   public void setup() {
35     init(rule.getProcessEngine());
36   }
37
38   /**
39    * Just tests if the process definition is deployable.
40    */
41   @Test
42   @Deployment(resources = "bpmnVorlesung2.bpmn")
43   public void testParsingAndDeployment() throws SQLException {
44     // nothing is done here, as we just want to check for exceptions during deployment
45
46       ProcessInstance processInstance = processEngine().getRuntimeService().startProcessInstanceByKey(PROCESS_DEFINITION_KEY);
47
48
49
50       //assertThat(processInstance).isEnded();
51
52       // To inspect the DB, run the following line in the debugger
53       // then connect your browser to: http://localhost:8082
54       // and enter the JDBC URL: jdbc:h2:mem:camunda
55       //org.h2.tools.Server.createWebServer("-web").start();
56
57   }
```

Code der Test Klasse:

```
package com.thbrandenburg.camunda.meister.docmanagement.genehmigung;

import java.sql.SQLException;

import org.apache.ibatis.logging.LogFactory;
import org.camunda.bpm.engine.runtime.ProcessInstance;
import org.camunda.bpm.engine.test.ProcessEngineRule;
import
org.camunda.bpm.extension.process_test_coverage.junit.rules.TestCoverageProcessEngineRuleBuilder;
import org.camunda.bpm.engine.test.Deployment;
import org.junit.Before;
import org.junit.ClassRule;
import org.junit.Rule;
import org.junit.Test;

import static org.camunda.bpm.engine.test.assertions.ProcessEngineTests.*;
import static org.junit.Assert.*;

/**
 * Test case starting an in-memory database-backed Process Engine.
 */
public class InMemoryH2Test {

  @ClassRule
```

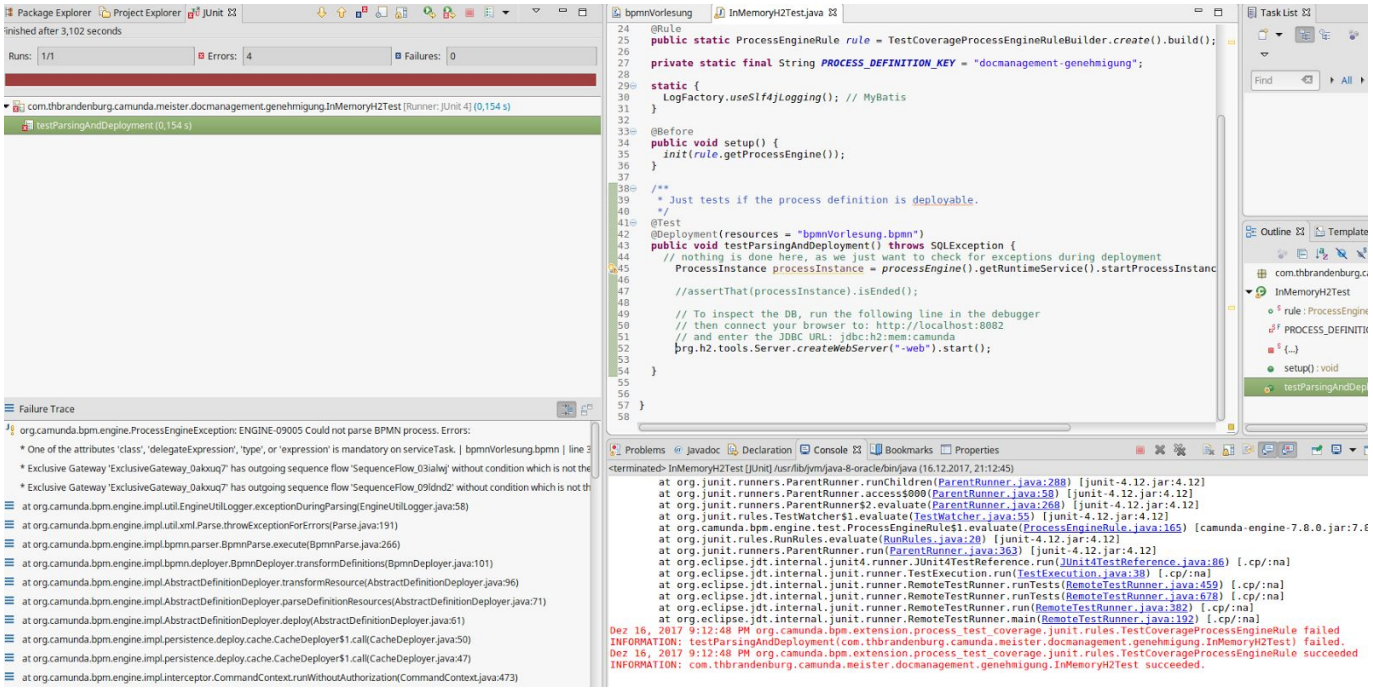
```
  @Rule
  public static ProcessEngineRule rule = TestCoverageProcessEngineRuleBuilder.create().build();

  private static final String PROCESS_DEFINITION_KEY = "docmanagement-genehmigung";

  static {
    LogFactory.useSlf4jLogging(); // MyBatis
  }

  @Before
  public void setup() {
    init(rule.getProcessEngine());
  }

  /**
   * Just tests if the process definition is deployable.
   */
  @Test
  @Deployment(resources = "bpmnVorlesung2.bpmn")
  public void testParsingAndDeployment() throws SQLException {
    // nothing is done here, as we just want to check for exceptions during deployment

        ProcessInstance processInstance =
processEngine().getRuntimeService().startProcessInstanceByKey(PROCESS_DEFINITION_KEY);

        //assertThat(processInstance).isEnded();

        // To inspect the DB, run the following line in the debugger
        // then connect your browser to: http://localhost:8082
        // and enter the JDBC URL: jdbc:h2:mem:camunda
        //org.h2.tools.Server.createWebServer("-web").start();
  }
}
```

Nach dem ausführen bekommen wir hier ein grünes Ergebnis und alles scheint okay zu sein:



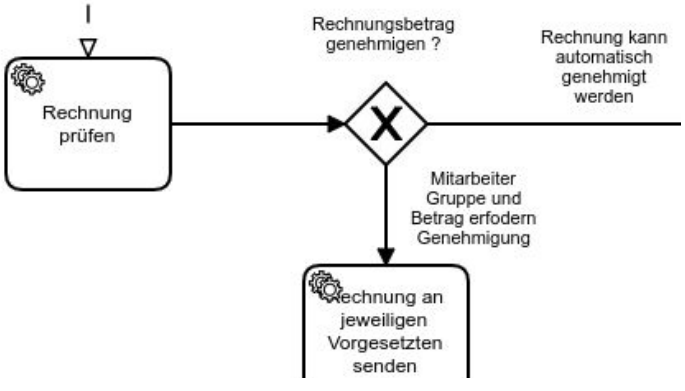
Auf dem Weg hierhin haben wir auch versucht den komplexen Prozess zum Laufen zu bringen, aber dabei sind wir auch auf viele Probleme gestoßen, da Variablen referenziert werden müssen und durch den Prozess die Informationen benutzt werden müssen, was doch komplizierter ist.

Mögliche Fehler beim Testen:




```
3:09:33.235 [main] DEBUG org.camunda.bpm.engine.test - ==== PROCESS ENGINE CREATED ==========
3:09:33.242 [main] DEBUG org.camunda.bpm.engine.test - annotation @Deployment creates deployment for InMemoryH2Test.testParsingAndDeployment
3:09:33.362 [main] INFO  org.camunda.bpm.engine.bpmn.parser - ENGINE-01002 Ignoring non-executable process with id 'Process_14n2h6e'. Set the attribute isExecutable="true" to
3:09:33.805 [main] ERROR org.camunda.bpm.engine.context - ENGINE-16006 BPMN Stack Trace:
       ExclusiveGateway_0akxuq7 (activity-leave, ProcessInstance[4])
       ExclusiveGateway_0akxuq7, name=Rechnungsbetrag genehmigen ?
         ^
         |
       Task_0in0dii, name=Rechnung
rüfen
         ^
         |
       StartEvent_1, name=Rechnung |
st Eingegangen

3:09:33.810 [main] ERROR org.camunda.bpm.engine.context - ENGINE-16004 Exception while closing command context: Unknown property used in expression: #{betragRechnung <= 1000}
rg.camunda.bpm.engine.ProcessEngineException: Unknown property used in expression: #{betragRechnung <= 1000}. Cause: Cannot resolve identifier 'betragRechnung'
       at org.camunda.bpm.engine.impl.el.JuelExpression.getValue(JuelExpression.java:60) ~[camunda-engine-7.8.0.jar:7.8.0]
       at org.camunda.bpm.engine.impl.el.UelExpressionCondition.evaluate(UelExpressionCondition.java:47) ~[camunda-engine-7.8.0.jar:7.8.0]
```

Wir hatten Probleme die Gateway Variablen zu referenzieren, die beim Aufruf der DMN Tabelle benutzt werden und sind dadurch hier nicht weiter gekommen.
In der Service Aktivität Rechnung Prüfen soll die DMN Tabelle geprüft werden:



Mit den Parametern aus dem Rechnungseingang:

BerechtigungsMatrixDocManagement

F	Input +	
	Mitarbeiter	Summe Rechnung
	string	integer
1	"Mitarbeiter Gruppe A"	<= 1000
2	"Mitarbeiter Gruppe A"	> 1000
3	"Mitarbeiter Gruppe B"	<= 500
4	"Mitarbeiter Gruppe B"	> 500
5	"Mitarbeiter Gruppe C", "Mitarbeiter Gruppe D"	> 0

und danach die Entscheidung ans Gateway weitergegeben werden.
Hierfür wird aber mehr Erfahrung mit der Camunda Engine in Eclipse benötigt.