BPMN 2.0 – Flowable

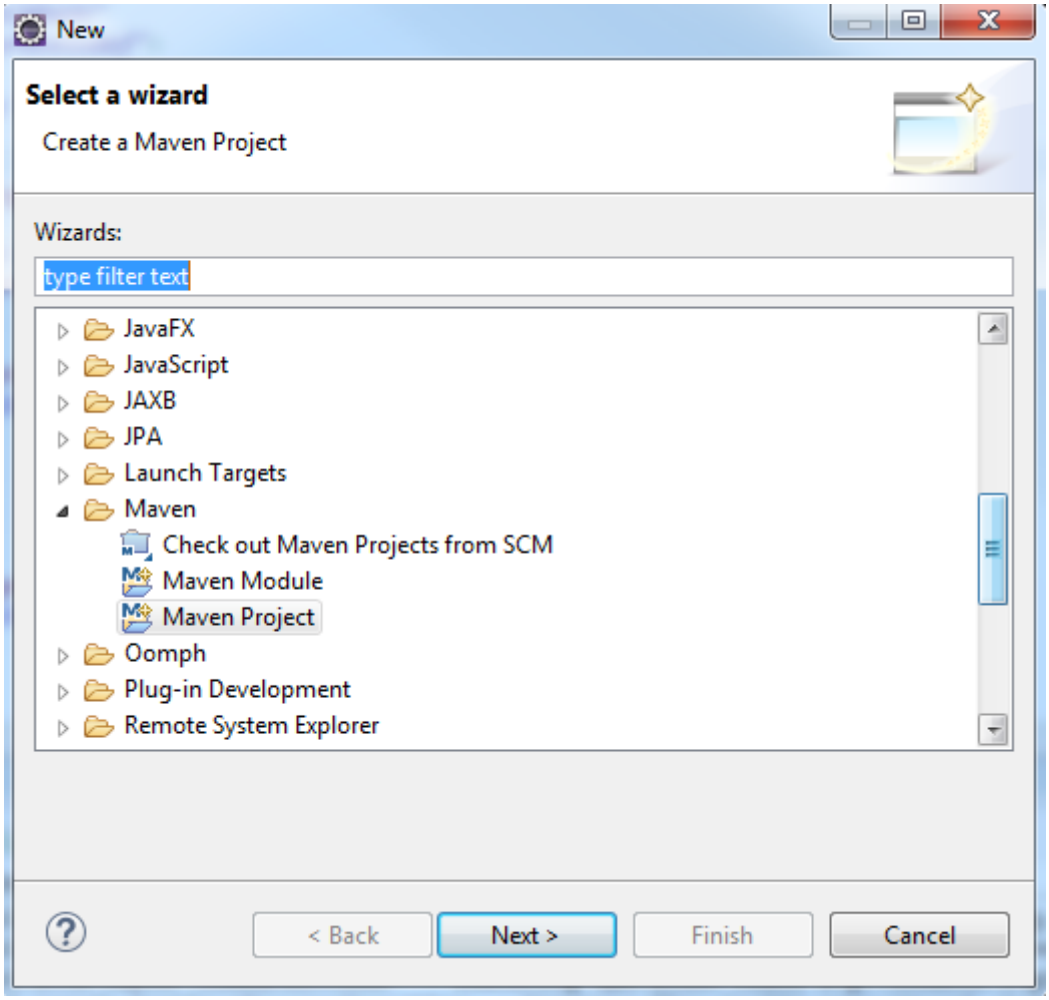
Ziel ist es, eine Java-Bibliothek zu nutzen, um Geschäftsprozesse, die in BPMN 2.0 modelliert sind, automatisiert umzusetzen.

Wir nutzen hier die Community Edition von Flowable.

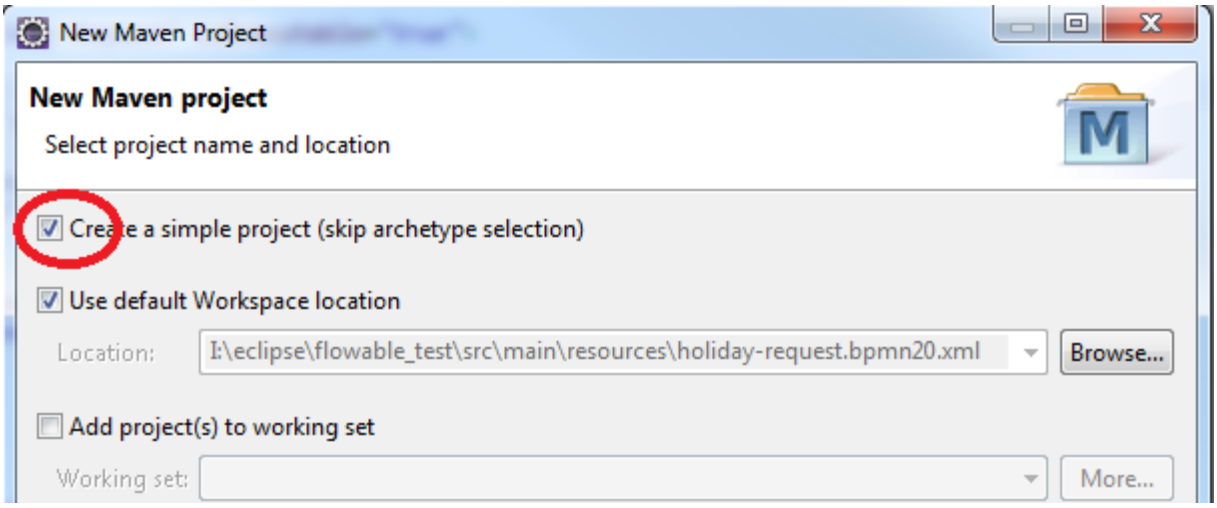
Viele BPMN 2.0-Software-Anbieter bieten sowohl Java-Apis als auch REST-Apis an.

Für die Nutzung der REST-Api ist jedoch ein Application Server (z. B. Tomcat) notwendig.

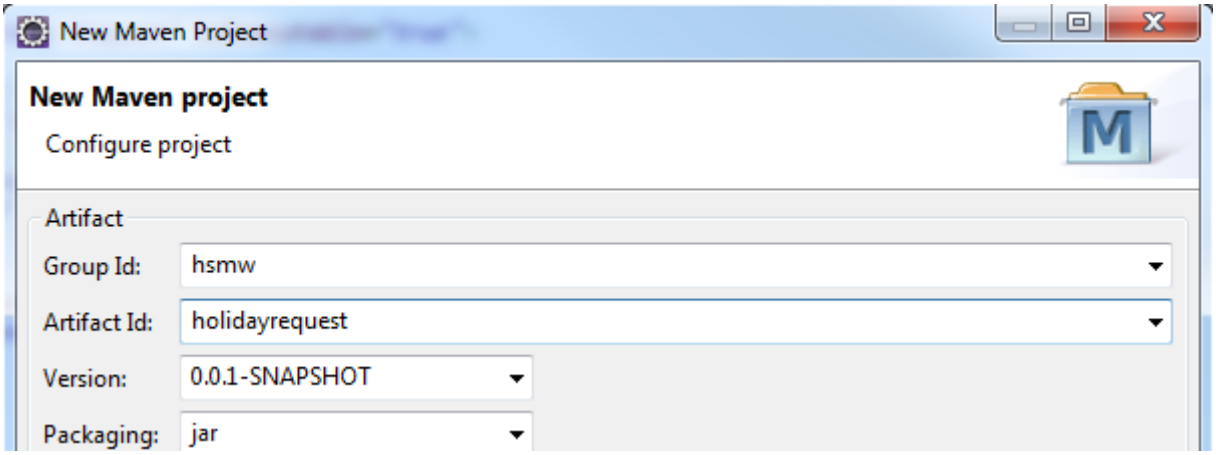
- Öffnen Sie Eclipse
- Legen Sie ein neues Maven-Projekt an
- File->New->Other



- Setzen Sie den Haken bei Create simple Project



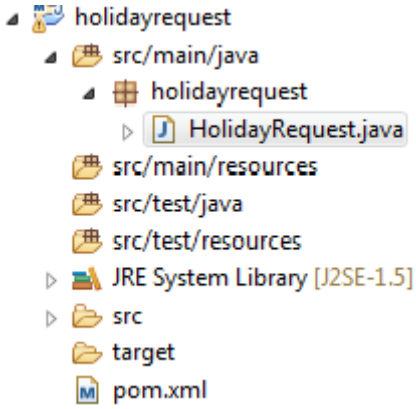
- Geben Sie als group ID an: hsmw
- Geben Sie als artifact an: holidayrequest



- Jetzt fügen wir die Dependencies ein
  1.) Flowable Process Engine
  2.) H2 Datenbank (RAM-Datenbank) zum Speichern der historischen Daten und der Ausführung
- Ergänzen Sie die pom.xml wiefolgt:

```
<dependencies>
  <dependency>
    <groupId>org.flowable</groupId>
    <artifactId>flowable-engine</artifactId>
    <version>6.4.2</version>
  </dependency>
  <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.3.176</version>
  </dependency>
</dependencies>
```

- Speichern Sie die pom.xml
- Legen Sie eine neue Klasse an in src/main/java




```
package holidayrequest;

public class HolidayRequest {

}
```

- Legen Sie eine Instanz einer **ProcessEngine** an.
  - Eine ProcessEngine-Instanz wird aus einer **ProcessEngineConfiguration** erzeugt. Diese benötigt mindestens einen JDBC-Connector zu einer Datenbank.
  - Legen Sie dafür eine main-Methode an und fügen Sie folgenden Quellcode ein.

```
package holidayrequest;

import org.flowable.engine.ProcessEngine;
import org.flowable.engine.ProcessEngineConfiguration;
import org.flowable.engine.impl.cfg.StandaloneProcessEngineConfiguration;

public class HolidayRequest {
	public static void main(String[] args){
	    ProcessEngineConfiguration cfg = new
StandaloneProcessEngineConfiguration()
	                .setJdbcUrl("jdbc:h2:mem:flowable;DB_CLOSE_DELAY=-1")
	                .setJdbcUsername("sa")
	                .setJdbcPassword("")
	                .setJdbcDriver("org.h2.Driver")

.setDatabaseSchemaUpdate(ProcessEngineConfiguration.DB_SCHEMA_UPDATE_TRUE);

	                ProcessEngine processEngine = cfg.buildProcessEngine();
	}

}
```

- Führen Sie die Klasse nun aus.

Es erscheinen ein paar Warnungen für log4j.

- Fügen Sie die entsprechenden Dependencies zur pom.xml hinzu.

```
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-api</artifactId>
```

```
  <version>1.7.21</version>
</dependency>
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-log4j12</artifactId>
  <version>1.7.21</version>
</dependency>
```

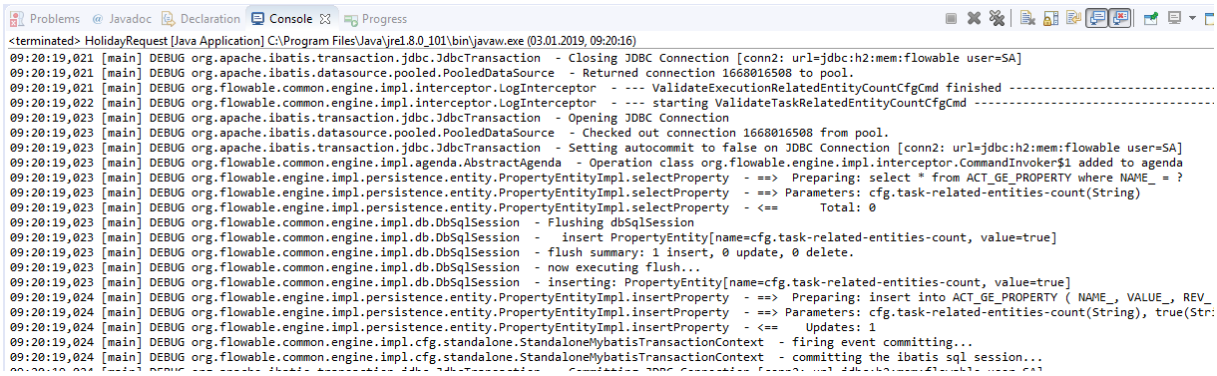
Log4j benötigt eine Konfigurationsdatei.

- Legen Sie eine neue Datei mit dem Namen *log4j.properties* unter *src/main/resources* an.
- Fügen Sie den folgenden Inhalt in die Datei log4j.properties.

```
log4j.rootLogger=DEBUG, CA
log4j.appender.CA=org.apache.log4j.ConsoleAppender
log4j.appender.CA.layout=org.apache.log4j.PatternLayout
log4j.appender.CA.layout.ConversionPattern= %d{hh:mm:ss,SSS} [%t] %-5p %c
%x - %m%n
```

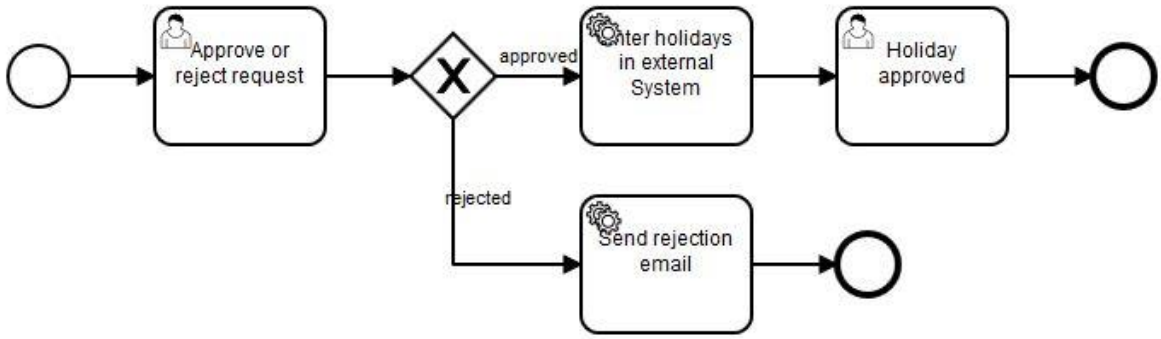
- Starten Sie die Anwendung erneut.

Die Konsolenausgabe sollte jetzt ungefähr so aussehen.



Prozessdefinition deployen

Es soll folgender Prozess umgesetzt werden



Damit die ProcessEngine damit umgehen kann, benötigen wir die XML-Repräsentation des Prozesses.

Diese ist hier gegeben oder in der Datei flow_xml.txt.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
  xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
  xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
  xmlns:flowable="http://flowable.org/bpmn"
  typeLanguage="http://www.w3.org/2001/XMLSchema"
  expressionLanguage="http://www.w3.org/1999/XPath"
  targetNamespace="http://www.flowable.org/processdef">

  <process id="holidayRequest" name="Holiday Request" isExecutable="true">

    <startEvent id="startEvent"/>
    <sequenceFlow sourceRef="startEvent" targetRef="approveTask"/>

    <userTask id="approveTask" name="Approve or reject request"/>
    <sequenceFlow sourceRef="approveTask" targetRef="decision"/>

    <exclusiveGateway id="decision"/>
    <sequenceFlow sourceRef="decision" targetRef="externalSystemCall">
      <conditionExpression xsi:type="tFormalExpression">
        <![CDATA[
          ${approved}
        ]]>
      </conditionExpression>
    </sequenceFlow>
    <sequenceFlow  sourceRef="decision" targetRef="sendRejectionMail">
      <conditionExpression xsi:type="tFormalExpression">
        <![CDATA[
          ${!approved}
        ]]>
      </conditionExpression>
    </sequenceFlow>

    <serviceTask id="externalSystemCall" name="Enter holidays in external
system"
        flowable:class="here_package_name.CallExternalSystemDelegate"/>
    <sequenceFlow sourceRef="externalSystemCall"
targetRef="holidayApprovedTask"/>

    <userTask id="holidayApprovedTask" name="Holiday approved"/>
    <sequenceFlow sourceRef="holidayApprovedTask" targetRef="approveEnd"/>

    <serviceTask id="sendRejectionMail" name="Send out rejection email"
        flowable:class="here_package_name.SendRejectionMail"/>
    <sequenceFlow sourceRef="sendRejectionMail" targetRef="rejectEnd"/>

    <endEvent id="approveEnd"/>

    <endEvent id="rejectEnd"/>

  </process>

</definitions>
```

- Machen Sie sich mit dem xml vertraut
    - Die Attribute im definition tag sind boilerplate

- Legen Sie eine Datei mit dem Namen *holiday-request.bpmn20.xml* im Ordner *src/main/resources* an.
- Fügen Sie das oben angegeben xml ein.
- Das Deployment wird wiefolgt durchgeführt. Kopieren Sie dafür den Code in die main-Methode.

```
RepositoryService repositoryService = processEngine.getRepositoryService();
Deployment deployment = repositoryService.createDeployment()
  .addClasspathResource("holiday-request.bpmn20.xml")
  .deploy();


ProcessDefinition processDefinition =
repositoryService.createProcessDefinitionQuery()
  .deploymentId(deployment.getId())
  .singleResult();
System.out.println("Prozessdefinition gefunden: " +
processDefinition.getName());
```

- Importieren Sie die notwendigen Bibliotheken

```
import org.flowable.engine.impl.cfg.StandaloneProcessEngineConfiguration;
import org.flowable.engine.repository.Deployment;
import org.flowable.engine.repository.ProcessDefinition;
```

- Starten Sie die Anwendung und prüfen Sie, ob der Name des Prozesses angezeigt wird.

```
> Prozessdefinition gefunden: Holiday Request
```

Eine Prozessinstanz starten

Beim Start soll eine Abfrage verschiedener Daten gemacht werden. Dies geschieht hier über die Konsole mithilfe von java.util.Scanner.

- Fügen Sie den folgenden Code in die main-Methode ein.

```
Scanner scanner= new Scanner(System.in);

System.out.println("Who are you?");
String employee = scanner.nextLine();

System.out.println("How many holidays do you want to request?");
Integer nrOfHolidays = Integer.valueOf(scanner.nextLine());

System.out.println("Why do you need them?");
String description = scanner.nextLine();
```

- Importieren Die die notwendige Bibliothek

```
import java.util.Scanner;
```

- Die Daten sollen in einer java.util.Map gespeichert werden. Fügen Sie folgenden Quellcode in die Main-Methode ein.

```
Map<String, Object> variables = new HashMap<String, Object>();
variables.put("employee", employee);
variables.put("nrOfHolidays", nrOfHolidays);
variables.put("description", description);
```

- Importieren Sie die notwendigen Bibliotheken:

```
import java.util.HashMap;
import java.util.Map;
```

Der Start der Process-Engine erfolgt über den RuntimeService. Diesem wird als key die id des Prozesses (siehe xml) übergeben.

- Passen Sie den folgenden Quellcode entsprechend an und fügen Sie ihn in die main-Methode ein.

```
RuntimeService runtimeService = processEngine.getRuntimeService();
runtimeService.startProcessInstanceByKey("hier Prozess id eingeben", variables);
```

- Probieren Sie die Eingabe der Variablen, indem Sie die Anwendung starten

Tasks abfragen und abschließen

Tasks dürfen nur von bestimmten Personen oder Gruppen ausgeführt werden.

Passen Sie die bpmn xml so an, dass nur Manager die Task approveTask ausführen können.

```
<userTask id="approveTask" name="Approve or reject request"
flowable:candidateGroups="managers"/>
```

Passen Sie die bpmn exm so an, dass nur Employees die Task holidayApproveTask ausführen können.

```
<userTask id="holidayApprovedTask" name="Holiday approved"
flowable:assignee="${employee}"/>
```

- Um die eigentliche Task-Liste zu erhalten, wird eine TaskQuery erstellt. Diese soll nur die Tasks der Manager ausgeben.

```
TaskService taskService = processEngine.getTaskService();
List<Task> tasks =
taskService.createTaskQuery().taskCandidateGroup("managers").list(); //import
org.flowable.task.api.Task;
System.out.println("You have " + tasks.size() + " tasks:");
for (int i=0; i<tasks.size(); i++) {
      System.out.println((i+1) + ") " + tasks.get(i).getName());
}
```

Mit dem Task-Identifizierer können die spezifischen Prozess-Instanz-Variablen ausgelesen werden.

```
System.out.println("Welche task möchten Sie bearbeiten?");
int taskIndex = Integer.valueOf(scanner.nextLine());
Task task = tasks.get(taskIndex - 1);
Map<String, Object> processVariables = taskService.getVariables(task.getId());
```

```
System.out.println(processVariables.get("employee") + " will " +
processVariables.get("nrOfHolidays") + " Tage Urlaub. Genehmigen Sie dies?");
```

Der Manager kann nun die Task abschließen. Meist wird ein Formular an einen Nutzer geschickt. Wir geben wir die Map mit der approved-Variable weiter.

```
boolean approved = scanner.nextLine().toLowerCase().equals("y");
variables = new HashMap<String, Object>();
variables.put("approved", approved);
taskService.complete(task.getId(), variables);
```

Die Task ist nun abgeschlossen. Das Gateway wird nun auf einem der zwei ausgehenden Pfade verlassen.

Die Service Task

Ziel ist es jetzt, eine Logig zu implementieren, die ausgeführt wird, wenn die vorangehende User-Task abgeschlossen ist. Dies könnte der Call einer REST Api sein, das Versenden einer E-Mail, Speichern in einer Datenbank usw.

Wir geben uns mit einer Konsolenausgabe zufrieden.

Wenn Sie die obige Anwendung ausführen, müsste folgende Fehlermeldung erscheinen:

```
Exception in thread "main" org.flowable.common.engine.api.FlowableException: couldn't instantiate class here_package_name.CallExternalSystemDelegate
        at org.flowable.common.engine.impl.util.ReflectUtil.instantiate(ReflectUtil.java:140)
        at org.flowable.engine.impl.bpmn.helper.AbstractClassDelegate.defaultInstantiateDelegate(AbstractClassDelegate.java:64)
```

- Passen Sie die bpmn-xml entsprechend an.

Legen Sie eine neue Klasse mit dem Namen CallExternalSystemDelegate an und ergänzen Sie den Code wiefolgt:

```
package holidayrequest;

import org.flowable.engine.delegate.DelegateExecution;
import org.flowable.engine.delegate.JavaDelegate;

public class CallExternalSystemDelegate implements JavaDelegate{
    public void execute(DelegateExecution execution) {
        System.out.println("Calling the external system for employee "
            + execution.getVariable("employee"));
    }
}
```

- Führen Sie nun die Anwendung erneut aus.

Es solle in den Logausgaben folgende Meldung erscheinen:

```
10:33:19,736 [main] DEBUG org.flowable.engine.impl.age
Calling the external system for employee Dozent
10:33:19,743 [main] DEBUG org.flowable.engine.impl.age
```