Tasks and Assignments Course 02267

Hubert Baumeister

Version 1

1 Task (single/in pairs/using mob programming): Simple DTU Pay Integration with SOAP Bank Service TASK_SOAP

1.1 Introduction

Simple DTU Pay acts as a broker between customers and merchants, initiating money transfers via their bank accounts. This task enhances Simple DTU Pay from the previous task by integrating it with a SOAP-based bank service for real money transfers.

- **Bank Details**: The bank is provided by me, accessible via a SOAP interface[1].
- **Key Features to Implement**:
  1. Customer and merchant registration with bank integration.
  2. Payment handling using bank accounts.

1.2 Task Details

1.2.1 Use the Bank for Money Transfers

- Ensure customers and merchants have valid bank accounts prior to registration to Simple DTU Pay.
- **Important:** Simple DTU Pay is not responsible for creating an account with the bank. Bank accounts are created in their respective step-definitions.
- Also, remember which bank accounts have created and then release them after the tests have finished. **Don't** remove bank accounts that you have not created yourself. Those have been created, and are used, by other users of the bank.
- Simple DTU Pay must:
  - Use the bank's SOAP service to transfer money from a customer's bank account to a merchant's account.

1. Steps
   (a) **Bank Account Creation**:
   - Bank accounts must be created using the 'createAccount' service provided by the bank. This step is done in your test setup.
   - Each account must be initialized with a balance (e.g., 1000 kr).
   - Use the returned account number for further actions.

   (b) **Simple DTU Pay Registration**:
   - Customers and merchants must register with Simple DTU Pay using:

[1]: `http://fm-00.compute.dtu.dk`

- First and last names and CPR number (all required by the bank).
- Bank account number.

- Upon registration, Simple DTU Pay assigns a unique ID to each user.
- For simplicity, Simple DTU Pay does not need to check that the bank account numbers are valid. Invalid bank account numbers will cause payment failures.

(c) **Payment Handling**:

- Payments are initiated by a merchant specifying:
  - The customer's ID.
  - The payment amount.
- On successful transactions:
  - Customer's bank account balance decreases.
  - Merchant's bank account balance increases.

2. Example Cucumber Scenario

```
Feature: Payment
  Scenario: Successful Payment
    Given a customer with name "Susan", last name "Baldwin", and CPR "030154-4421"
    And the customer is registered with the bank with an initial balance of 1000 kr
    And the customer is registered with Simple DTU Pay using their bank account
    And a merchant with name "Daniel", last name "Oliver", and CPR "131161-3045"
    And the merchant is registered with the bank with an initial balance of 1000 kr
    And the merchant is registered with Simple DTU Pay using their bank account
    When the merchant initiates a payment for 10 kr by the customer
    Then the payment is successful
    And the balance of the customer at the bank is 990 kr
    And the balance of the merchant at the bank is 1010 kr
```

1.2.2 SOAP Web Service Integration

1. SOAP Service Access

   The bank's SOAP interface is accessible at `http://fm-00.compute.dtu.dk[fn:200]`. Use the provided WSDL file[2] to generate client stubs for communication.

2. Generating Stubs

   Use the 'jaxws-maven-plugin' in your 'pom.xml':

```
<plugin>
  <groupId>com.sun.xml.ws</groupId>
  <artifactId>jaxws-maven-plugin</artifactId>
  <version>4.0.3</version>
  <executions>
    <execution>
      <goals>
        <goal>wsimport</goal>
      </goals>
    </execution>
  </executions>
  <configuration>
    <wsdlUrls>
      <wsdlUrl>http://fm-00.compute.dtu.dk/services/BankService?wsdl</wsdlUrl>
```

[2]: `http://fm-00.compute.dtu.dk/services/BankService?wsdl`

```
    </wsdlUrls>
    <keep>true</keep>
    <packageName>dtu.ws.fastmoney</packageName>
    <sourceDestDir>src/main/java</sourceDestDir>
  </configuration>
</plugin>
```

Maven commands ‘mvn package‘ and ‘mvn test‘ will automatically generate the stubs. To only generate the stubs you can use ‘mvn jaxws:wsimport‘

Ensure the JAX-WS reference implementation is included in your project as the following dependency:

- SOAP:

```
<dependency>
  <groupId>com.sun.xml.ws</groupId>
  <artifactId>jaxws-ri</artifactId>
  <version>4.0.3</version>
  <type>pom</type>
</dependency>
```

You can use ‘mvn jaxws:help‘ and ‘mvn jaxws:help -Ddetail=true -Dgoal=<goal-name>‘ to get an overview of the goals provided by the plugin and their parameters. Note that the documentation of the plugin from MojoHaus, that you find on the Web, is outdated, but still can give some help understanding the plugin.

3. Accessing the Bank Service

   To call the bank services, you first have to create a stub implementing the `BankService` interface. This is done by executing:

```
BankService bank = new BankService_Service().getBankServicePort();
```

1.2.3 Account Creation and Account Deletion

For creating and deleting an account, an api-key is required. Each group gets its own api-key. Note that you can only delete bank accounts with the api-key you have used to create the bank account. This is to prevent that you delete by accidents accounts from other groups.

Note that the other bank operations don’t need an API key.

1.2.4 Account Cleanup After Tests

To ensure repeatable tests:

- Keep track of all created bank accounts.
- Delete accounts after each test run using Cucumber’s ‘@After‘ annotation.
- Note, both Cucumber and JUnit provide an @After annotation. Use the @After annotation from the package `io.cucumber.java` for Cucumber.

1. Example Cleanup Code

   Accounts should be added to a list during creation:

```
BankService bank = ...;
List<String> accounts = new ArrayList<>();
...
public String registerBankAccount(...) {
    ...
    var user = new User(...)
    ...
    var account = bank.createAccountWithBalance(user, amount);
    accounts.add(account);
    return account;
}
```

Then they will be deleted after running a scenario by the following code.

```
@io.cucumber.java.After
public void cleanupBankAccounts() throws BankServiceException_Exception {
    for (String account : accounts) {
        bank.retireAccount(account);
    }
}
```