



Softwaretechnik I

Praktische Arbeit – OOP Gruppe 2

Jingrun Zhang

Silvia Wen

Mingwei Gao

Mona Amro

Mustafa Abdalla

Muhammad Daryl Rashad

30.01.2023

Anforderungstext – Bank

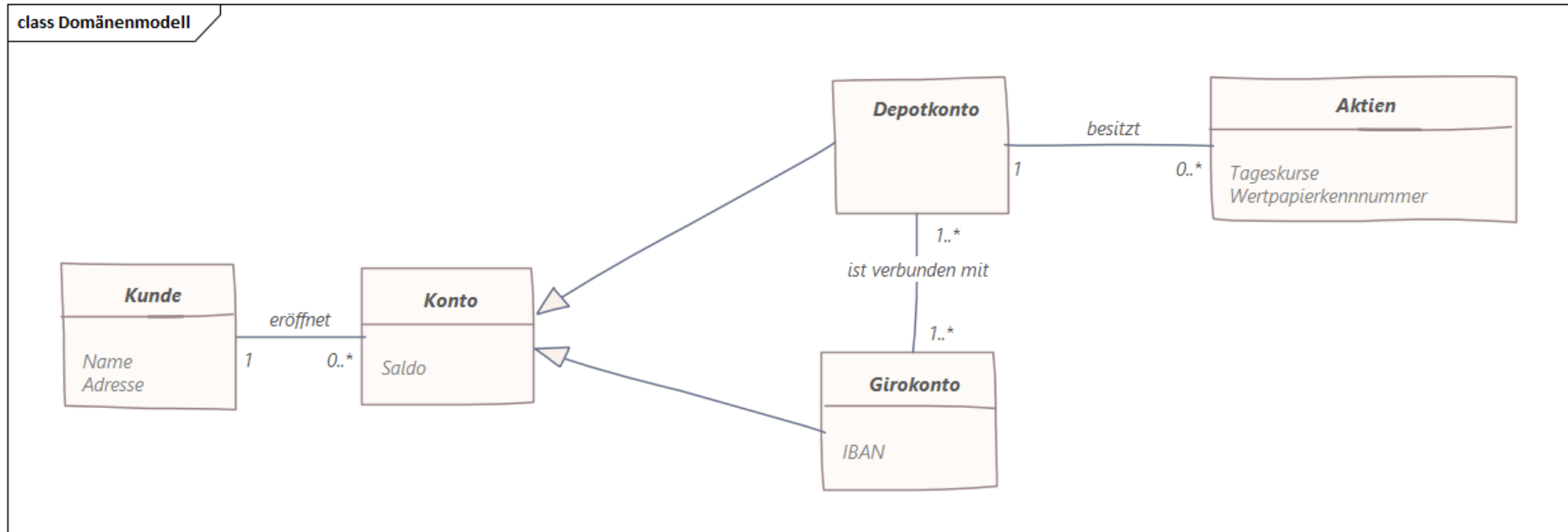
Gelb = Klassen

Grün = Attributen

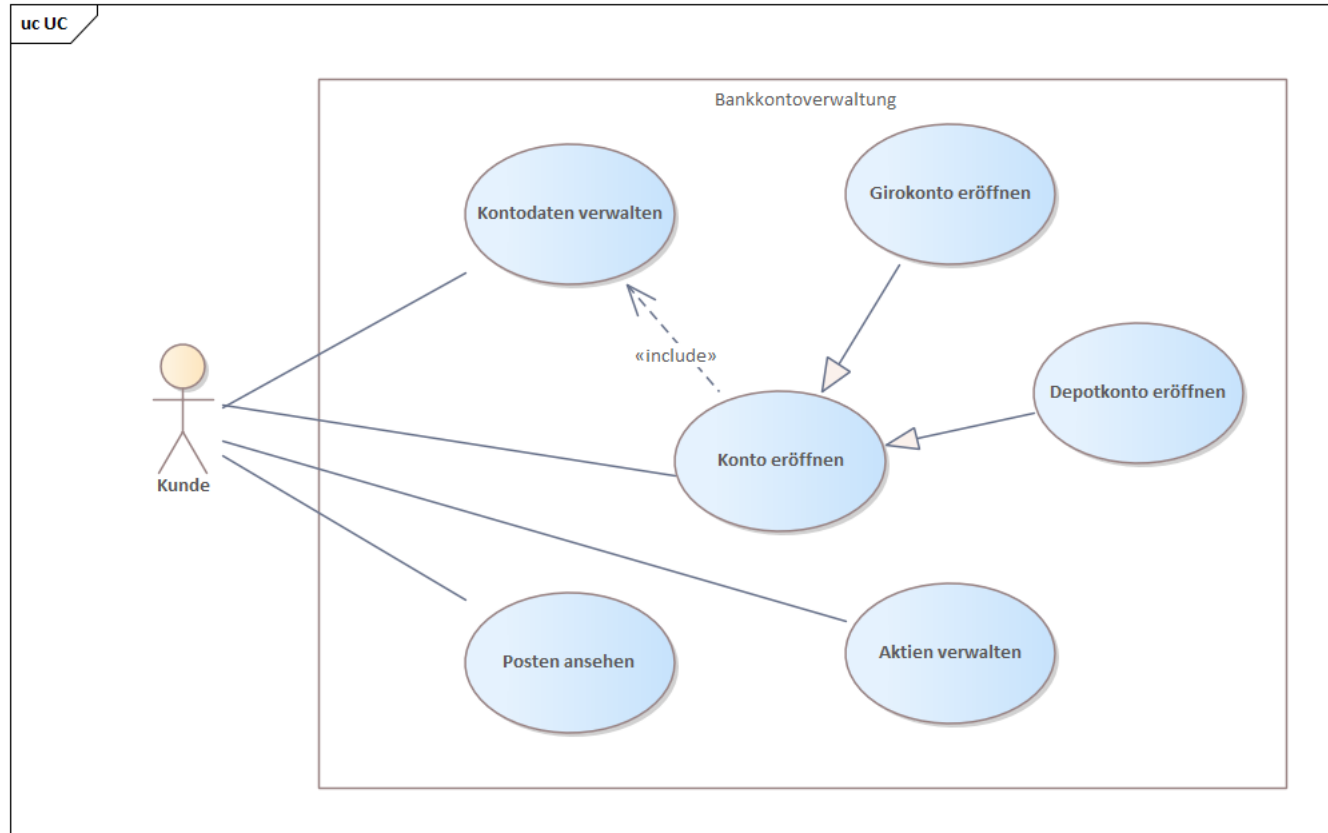
Blau = Use-Cases

In einer Bank können **Kunden** eine Reihe von **Konten** **eröffnen**. Hierzu müssen der **Name** und die **Adresse** **hinterlegt** werden. Alle Konten haben einen **Saldo**, welcher angibt, was für ein Wert sich aktuell auf dem Konto befindet. **Girokonten** sind die ganz normalen Konten, sie werden durch eine eindeutige **IBAN** identifiziert. Zudem gibt es **Depotkonten**, in dem die von den Kunden gekauften **Aktien** **abgelegt** werden. Der Kunde kann sich eine Auflistung der verschiedenen Posten in seinen Depots **anzeigen** lassen. Wichtig ist, dass zur Deckung von Käufen und zur Ausschüttung bei Verkäufen oder Dividendenzahlungen immer ein Referenzkonto angegeben werden muss. Zur Berechnung des Saldos eines Depots werden immer die **Tageskurse** der Aktien verwendet. Aktien werden durch die sogenannte **Wertpapierkennnummer** eindeutig identifiziert.

Domänenmodell



Use-Case Diagramm



Epics und User-Stories

Epic 1: Depotkonto eröffnen

Das System muss dem Kunden die Möglichkeit geben, eine Depotkonto eröffnen zu können.

(wird in Sequenzdiagramm dargestellt)

Epics und User-Stories

User Story 1: Als Kunde möchte ich Kontodaten wie Name und Adresse hinterlegen, um die bei der Eröffnung eines Depotkontos erforderlicher Informationen anzugeben.

Akzeptanzkriterium:

- Der Kunde muss seine Name und Adresse eingeben.
- Die eingegebene Name und Adresse muss zum Personalausweis übereinstimmen.

Epics und User-Stories

User Story 2: Als Kunde möchte ich Password des Depotkontos eingestellt, um ein zukünftiges Einloggen des Depotkontos zu ermöglichen.

Akzeptanzkriterium:

- Der Kunde muss zweimal das Password eingeben.
- Das Password muss aus mindestens 9 Zeichen bestehen.



Epics und User-Stories

Epic 2: Posten ansehen

Das System muss dem Kunden die Möglichkeit geben, die Posten in seinem Depot ansehen zu können.

Epics und User-Stories

User Story 1: Als Kunde möchte ich im Banksystem mein Depotkonto einloggen können, um die Posten im Konto zu checken.

Akzeptanzkriterium:

- Der Kunde muss seines Password eingeben.
- Wenn das Password mit dem Datenbank übereinstimmt, wird der Kunde eingeloggt sein.

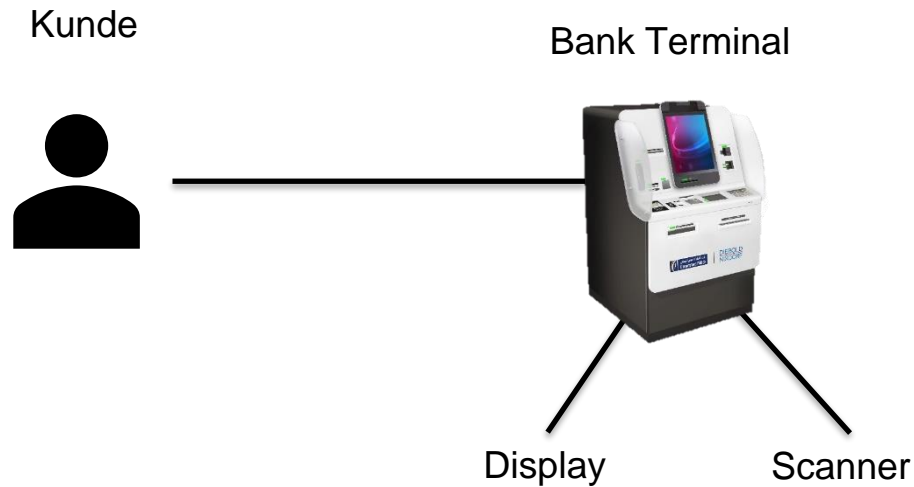
Epics und User-Stories

User Story 2: Als Kunde möchte ich die Liste der Posten in meinem Depot ansehen können, um die gesamte Werte meiner Aktien zu überwachen.

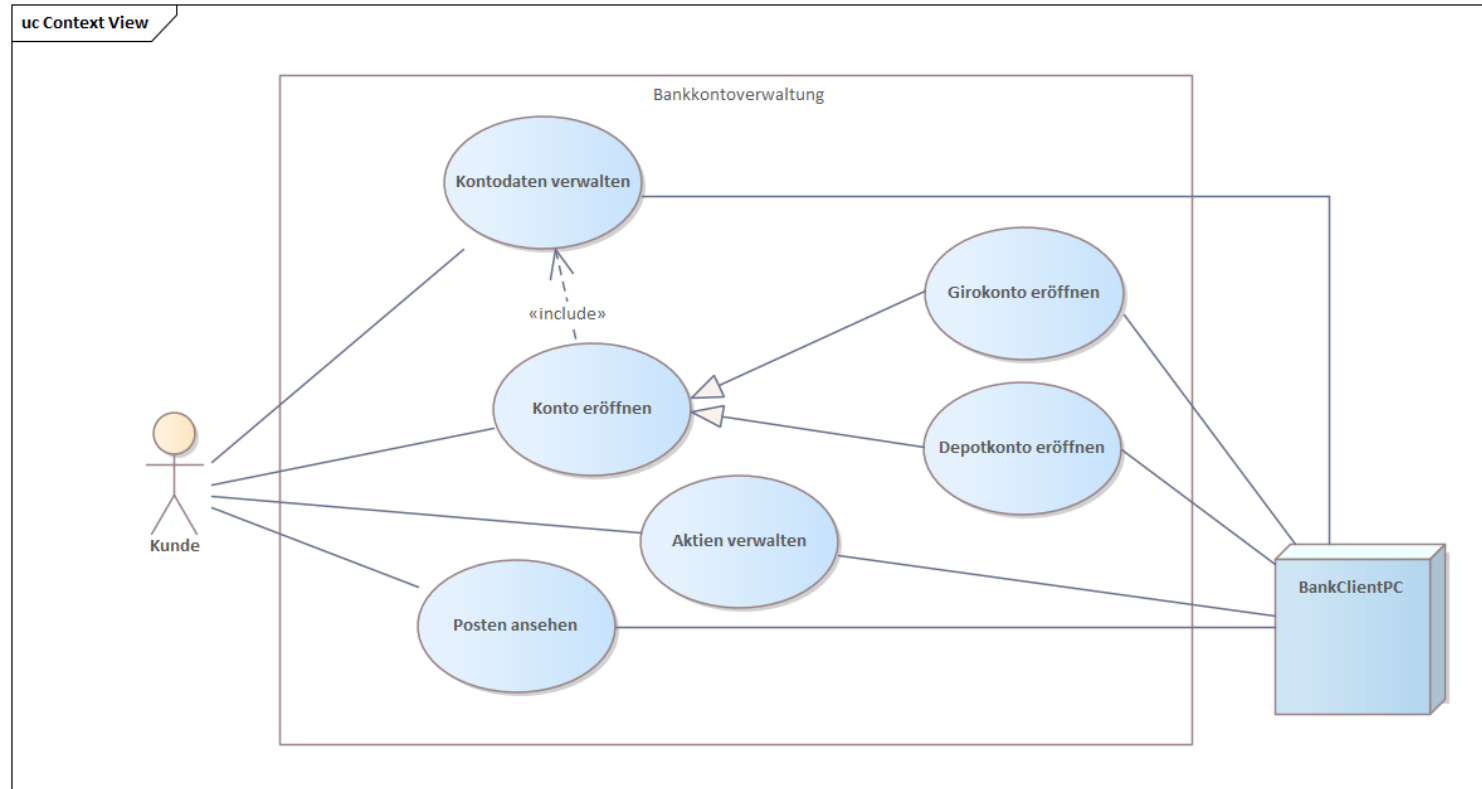
Akzeptanzkriterium:

- Der Kunde muss eingeloggt sein.
- Der Kunde bekommt die Liste der Posten angezeigt.
- Der Kunde bekommt die gesamte Werte des Depots angezeigt.

Context View: Technischer Überblick

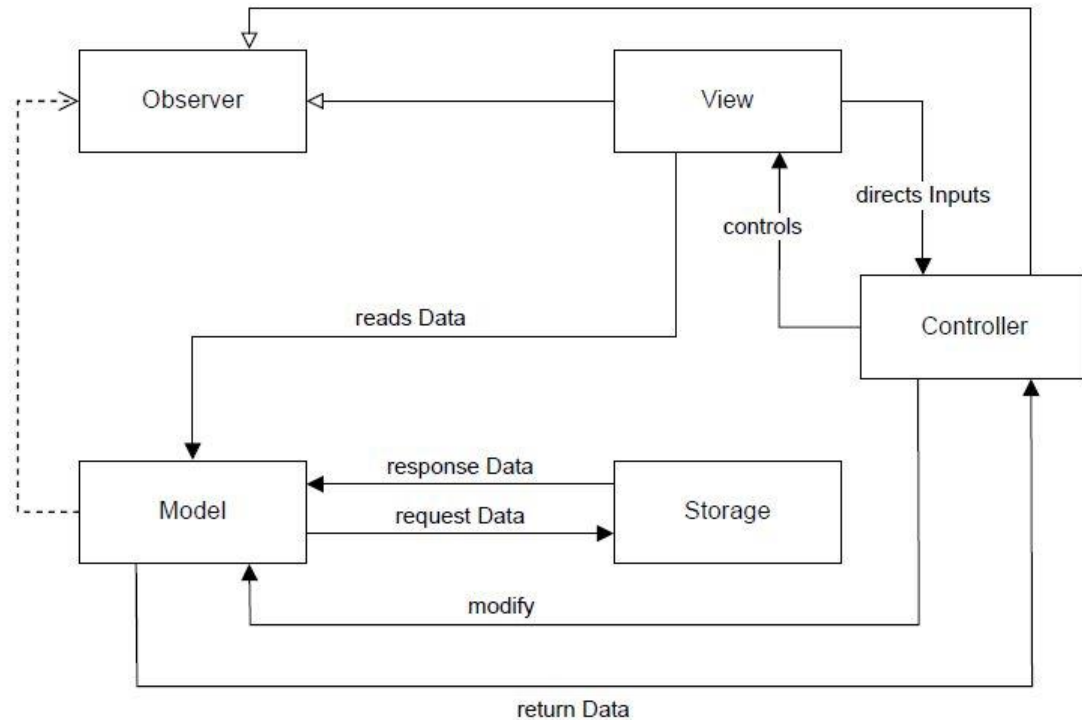


Context View: Fachlicher Überblick

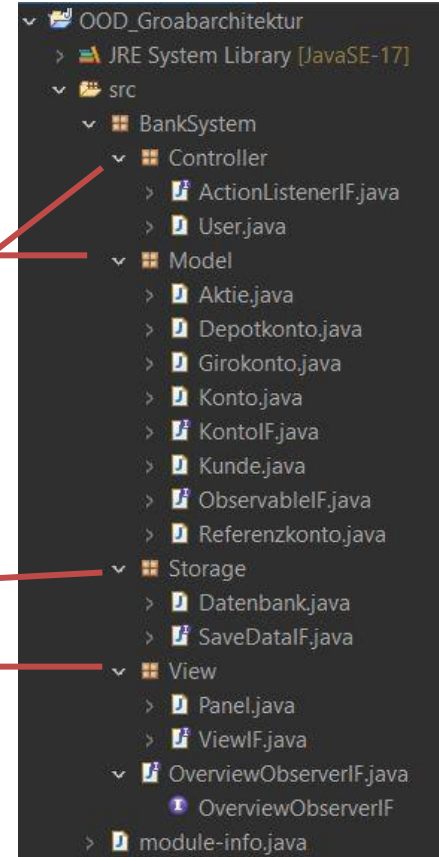
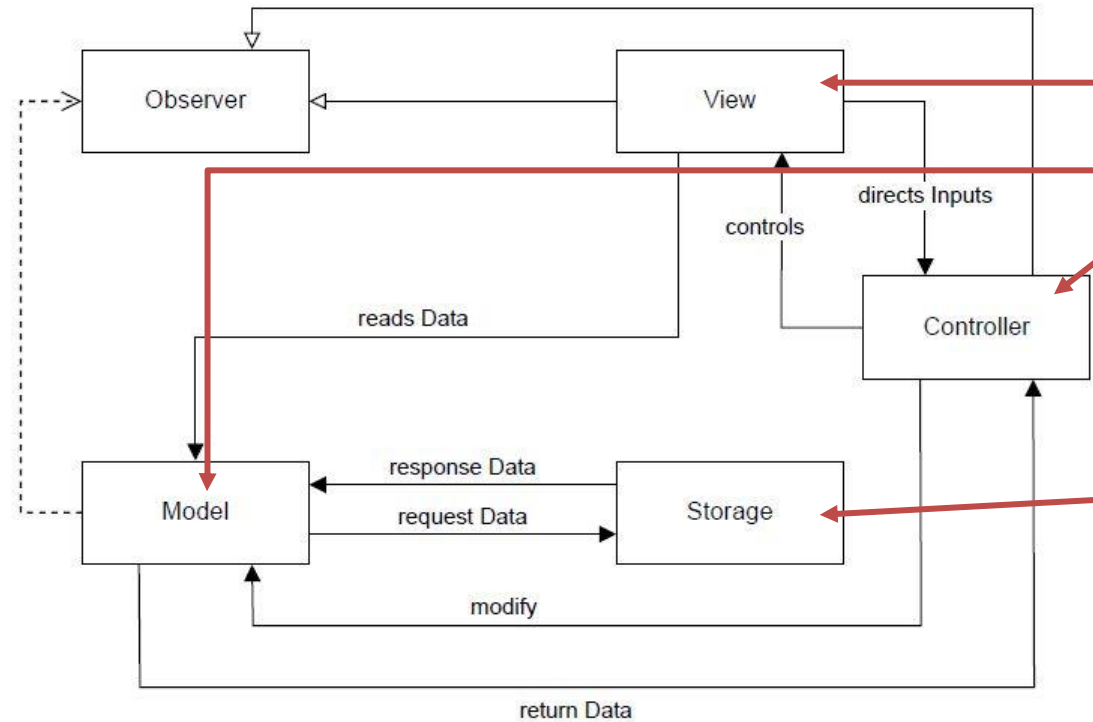


Grobarchitektur: Architekturstil

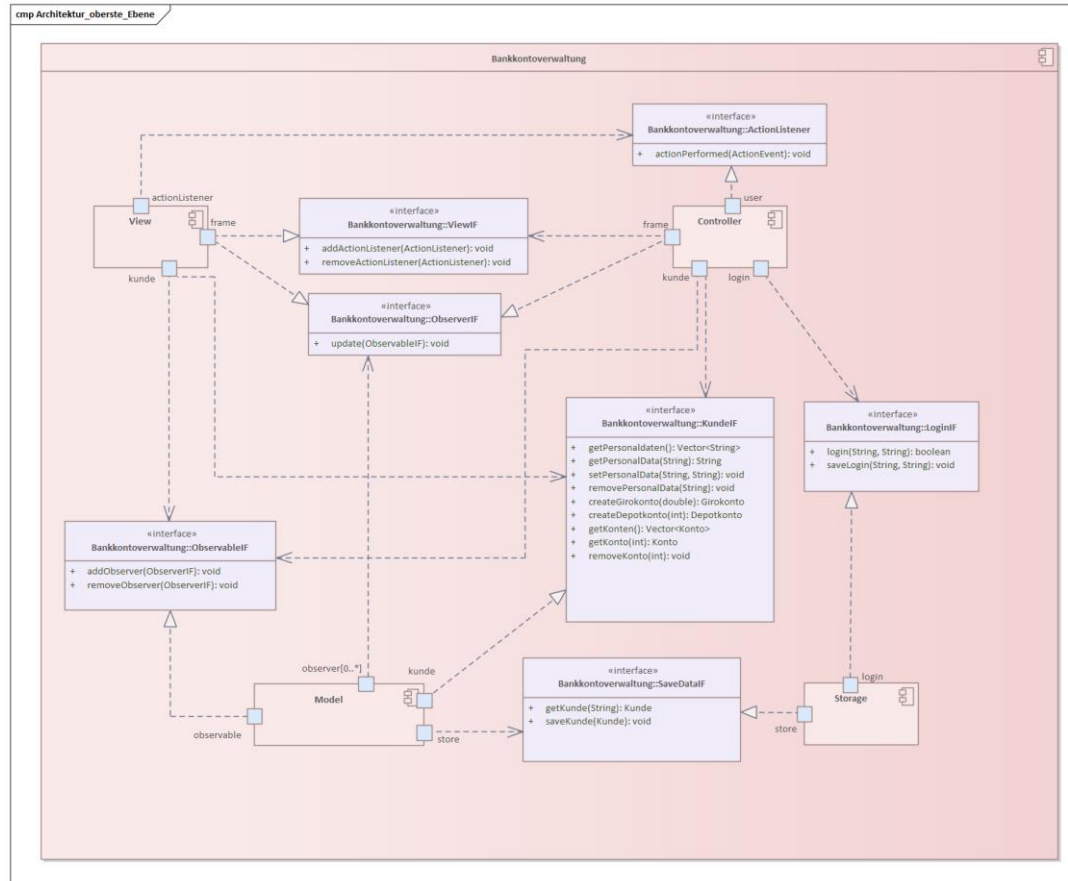
MVC + Storage



Grobarchitektur: Codeabbildung

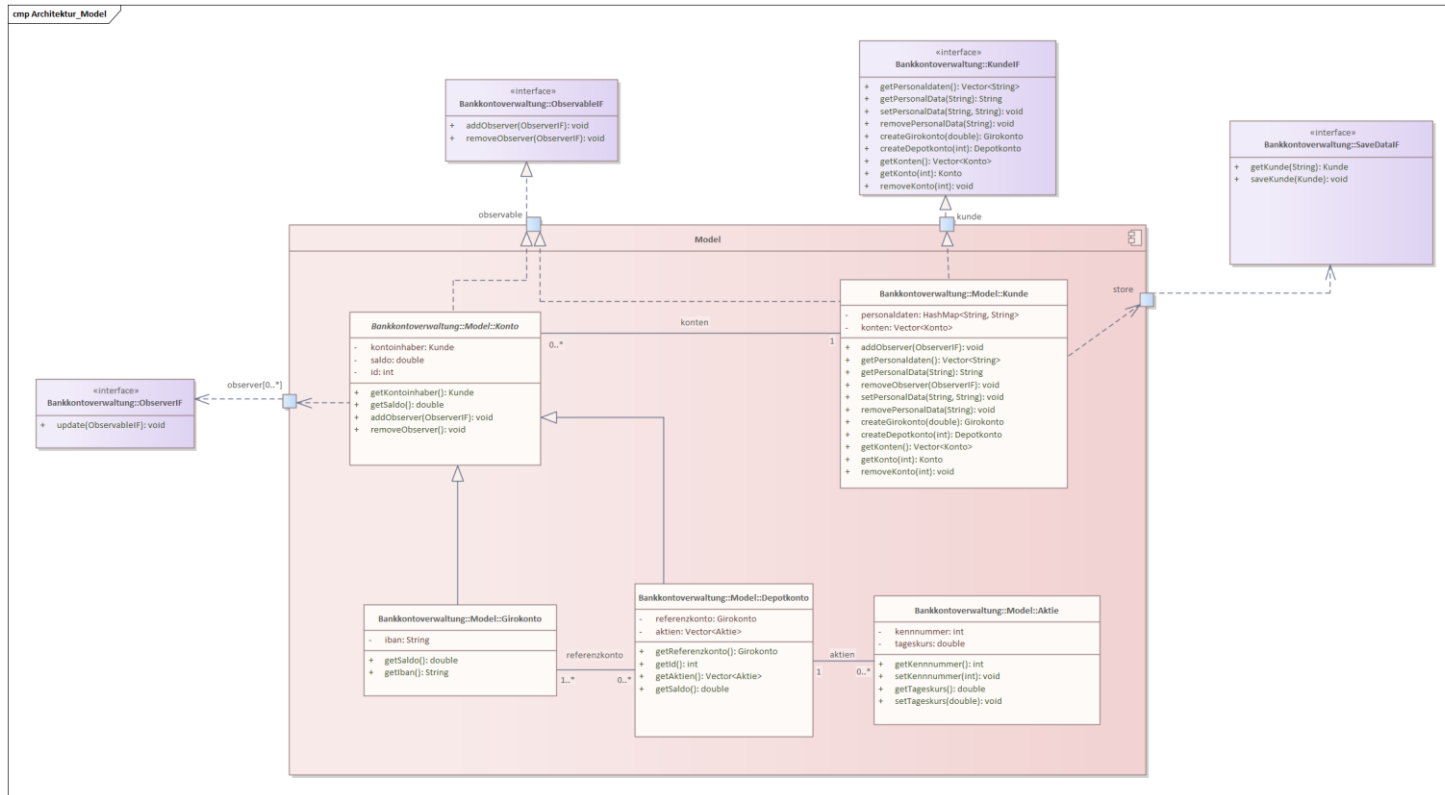


Fachliche Architekturebene: Structural View



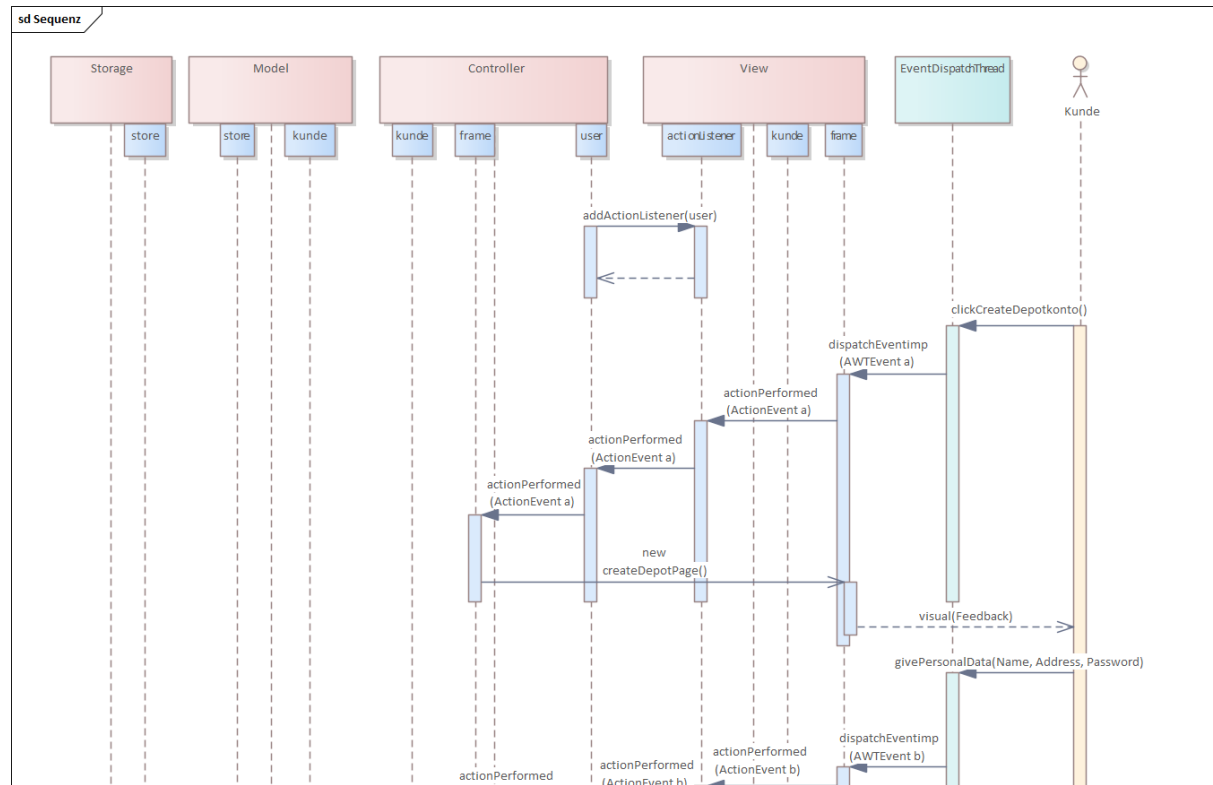
Fachliche Architekturebene: Structural View

Komponente Model



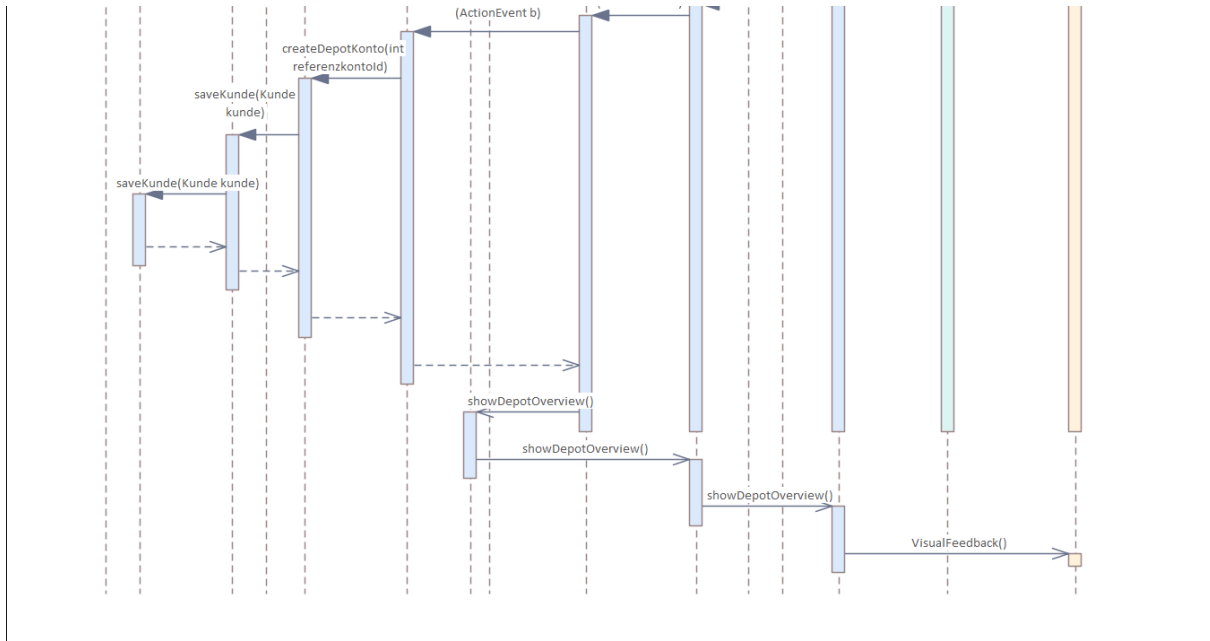
Fachliche Architekturebene: Behavioral View

Sequenzdiagramm für Epic 1: Depotkonto eröffnen



Fachliche Architekturebene: Behavioral View

Sequenzdiagramm für Epic 1: Depotkonto eröffnen



Demonstration des Prototypen und des Readme

Bankkontoverwaltung App

Simple bank account management app.

Usage

Prerequisites

- JDK 11 or newer [here](#)

Instructions

- Run BankkontoverwaltungApp.jar in the deploy folder by double-clicking it or using the following command:

```
cd deploy  
java -jar BankkontoverwaltungApp.jar
```

- Create account or login
- Use the available funtions

Demonstration des Prototypen und des Readme

Notes

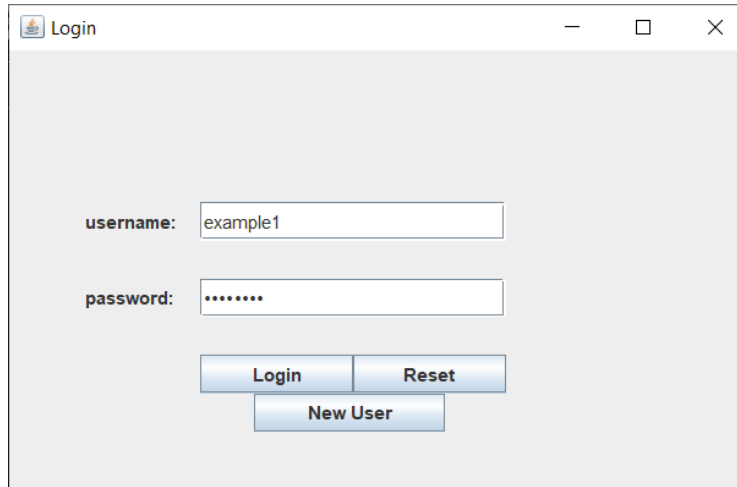
- Only the following functions have been implemented:
 - Create Giro
 - Create Depot
 - Accounts overview
- Example user credentials:
 - user: example1
 - password: password12345
- Javadoc located in doc folder

Credits

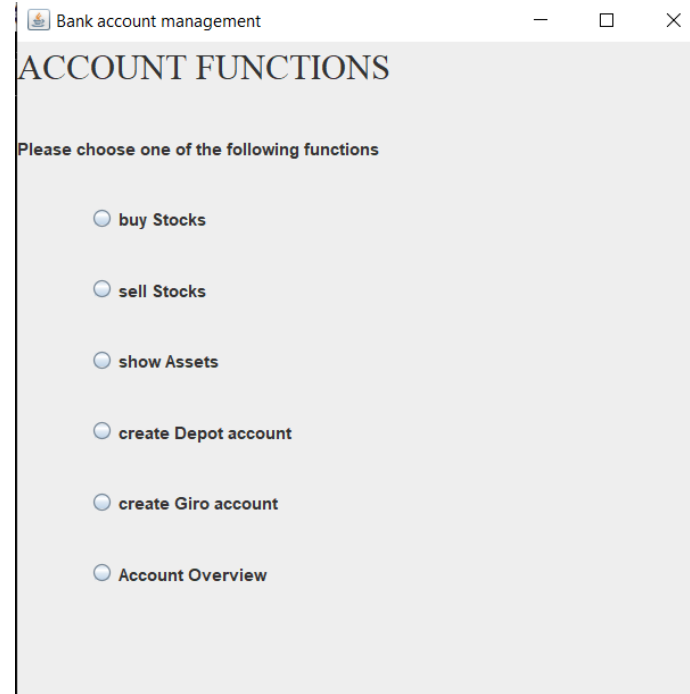
Made by SWT Group 2:

- Mona Amro
- Jingrun Zhang
- Mingwei Gao
- Muhammad Daryl Rashad
- Mustafa Abdalla
- Silvia Wen

Demonstration des Prototypen und des Readme



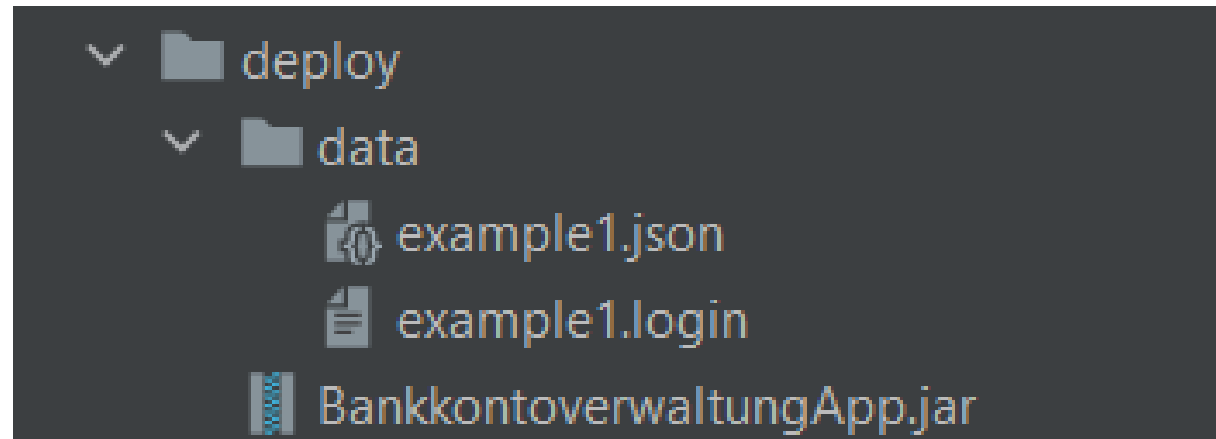
A screenshot of a web browser window titled "Login". The window has a light gray background. It contains two input fields: "username:" with the text "example1" and "password:" with masked characters ".....". Below the input fields are three buttons: "Login" and "Reset" side-by-side, and "New User" centered below them.



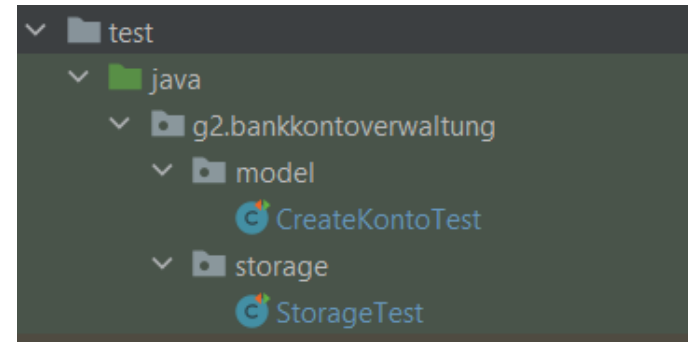
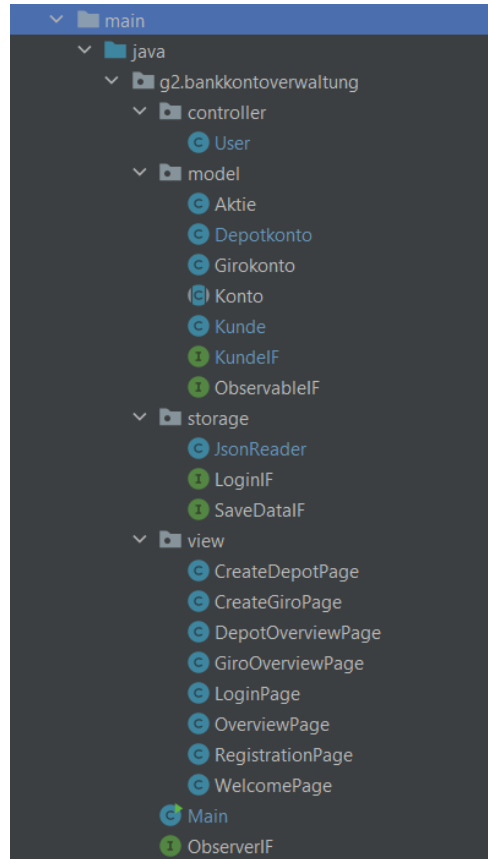
A screenshot of a web browser window titled "Bank account management". The window has a light gray background. It displays the heading "ACCOUNT FUNCTIONS" in a serif font. Below the heading is the text "Please choose one of the following functions". There are six radio button options listed vertically: "buy Stocks", "sell Stocks", "show Assets", "create Depot account", "create Giro account", and "Account Overview".

Demonstration des Prototypen und des Readme

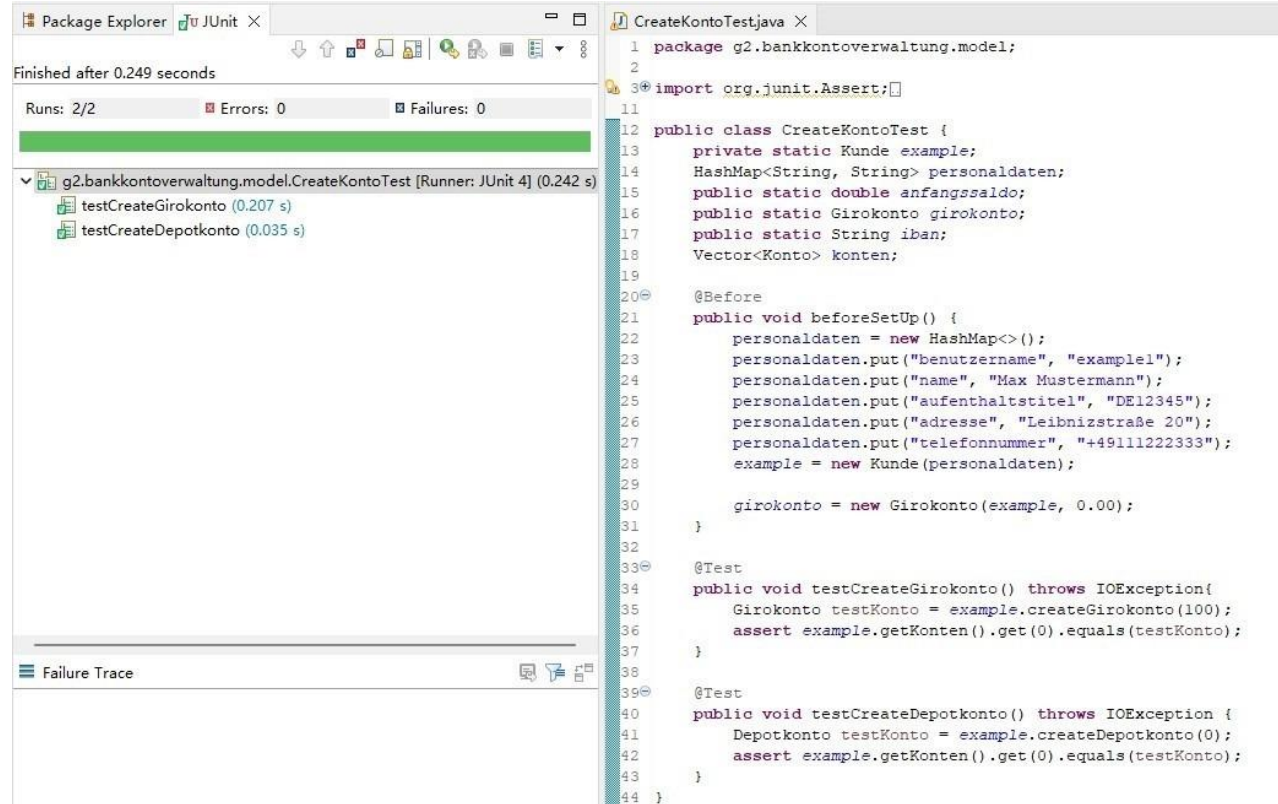
- Die Daten werden im Ordner data gespeichert
- JSON für Kundendaten
- .login für Password



Demonstration der Klassen- und Paketstruktur



Präsentation der Testfälle



The screenshot displays an IDE with two main panels. The left panel shows the 'JUnit' test runner results for the package 'g2.bankkontoverwaltung.model'. It indicates that the tests finished after 0.249 seconds, with 2 runs, 0 errors, and 0 failures. The test list includes 'testCreateGirokonto' (0.207 s) and 'testCreateDepotkonto' (0.035 s). The right panel shows the source code for 'CreateKontoTest.java'. The code defines a test class with static fields for an example customer, personal data, starting balance, giro account, and iban. It includes a '@Before' method for setup and two '@Test' methods: 'testCreateGirokonto' and 'testCreateDepotkonto', both of which create an account and assert the returned list of accounts.

```
1 package g2.bankkontoverwaltung.model;
2
3 import org.junit.Assert;
4
5
6
7
8
9
10
11
12 public class CreateKontoTest {
13     private static Kunde example;
14     HashMap<String, String> personaldaten;
15     public static double anfangssaldo;
16     public static Girokonto girokonto;
17     public static String iban;
18     Vector<Konto> konten;
19
20     @Before
21     public void beforeSetUp() {
22         personaldaten = new HashMap<>();
23         personaldaten.put("benutzername", "example1");
24         personaldaten.put("name", "Max Mustermann");
25         personaldaten.put("aufenthaltstitel", "DE12345");
26         personaldaten.put("adresse", "Leibnizstraße 20");
27         personaldaten.put("telefonnummer", "+49111222333");
28         example = new Kunde(personaldaten);
29
30         girokonto = new Girokonto(example, 0.00);
31     }
32
33     @Test
34     public void testCreateGirokonto() throws IOException{
35         Girokonto testKonto = example.createGirokonto(100);
36         assert example.getKonten().get(0).equals(testKonto);
37     }
38
39     @Test
40     public void testCreateDepotkonto() throws IOException {
41         Depotkonto testKonto = example.createDepotkonto(0);
42         assert example.getKonten().get(0).equals(testKonto);
43     }
44 }
```


Präsentation der Code-Dokumentation und Metrik-Auswertung in GitLab

[OVERVIEW](#) [PACKAGE](#) [CLASS](#) [TREE](#) [INDEX](#) [HELP](#)

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package g2.bankkontoverwaltung.model

Class Kunde

java.lang.Object[Ⓜ]
g2.bankkontoverwaltung.model.Kunde

All Implemented Interfaces:
KundeIF, ObservableIF

public class Kunde
extends Object[Ⓜ]
implements KundeIF, ObservableIF

Represents a Bank Customer

Author:
Muhammad Daryl Rashad

Constructor Summary

Constructors

Constructor	Description
Kunde()	
Kunde(HashMap [Ⓜ] <String [Ⓜ] ,String [Ⓜ] > personaldaten)	Basic constructor for a Kunde

Präsentation der Code-Dokumentation und Metrik-Auswertung in GitLab

All Methods	Instance Methods	Concrete Methods
Modifier and Type	Method	Description
void	<code>addObserver(ObserverIF observer)</code>	
<code>Depotkonto</code>	<code>createDepotkonto(int referenzkontoId)</code>	Create a new Depotkonto for the user
<code>Girokonto</code>	<code>createGirokonto(double anfangssaldo)</code>	Create a new Girokonto for the user
<code>Vector[Ⓔ]<Konto></code>	<code>getKonten()</code>	Get all Konto belonging to user
<code>Konto</code>	<code>getKonto(int id)</code>	Get a specific Konto belonging to the user
<code>HashMap[Ⓔ]<String[Ⓔ],String[Ⓔ]></code>	<code>getPersonaldaten()</code>	Get all personal data
void	<code>removeKonto(int id)</code>	Remove a Konto
void	<code>removeObserver(ObserverIF observer)</code>	
void	<code>removePersonalData(String[Ⓔ] key)</code>	Remove a specific personal data
void	<code>setPersonalData(String[Ⓔ] key, String[Ⓔ] value)</code>	Set/add a specific personal data
Methods inherited from class <code>java.lang.Object[Ⓔ]</code>		
<code>clone[Ⓔ], equals[Ⓔ], finalize[Ⓔ], getClass[Ⓔ], hashCode[Ⓔ], notify[Ⓔ], notifyAll[Ⓔ], toString[Ⓔ], wait[Ⓔ], wait[Ⓔ], wait[Ⓔ]</code>		

Präsentation der Code-Dokumentation und Metrik-Auswertung in GitLab

Method Details

getPersonaldaten

```
public HashMap<String, String> getPersonaldaten()
```

Get all personal data

Specified by:

getPersonaldaten in interface KundeIF

Returns:

HashMap containing personal data

setPersonalData

```
public void setPersonalData(String key,  
                             String value)
```

Set/add a specific personal data

Specified by:

setPersonalData in interface KundeIF

Parameters:

key - Type of personal data

value - Value of personal data