



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

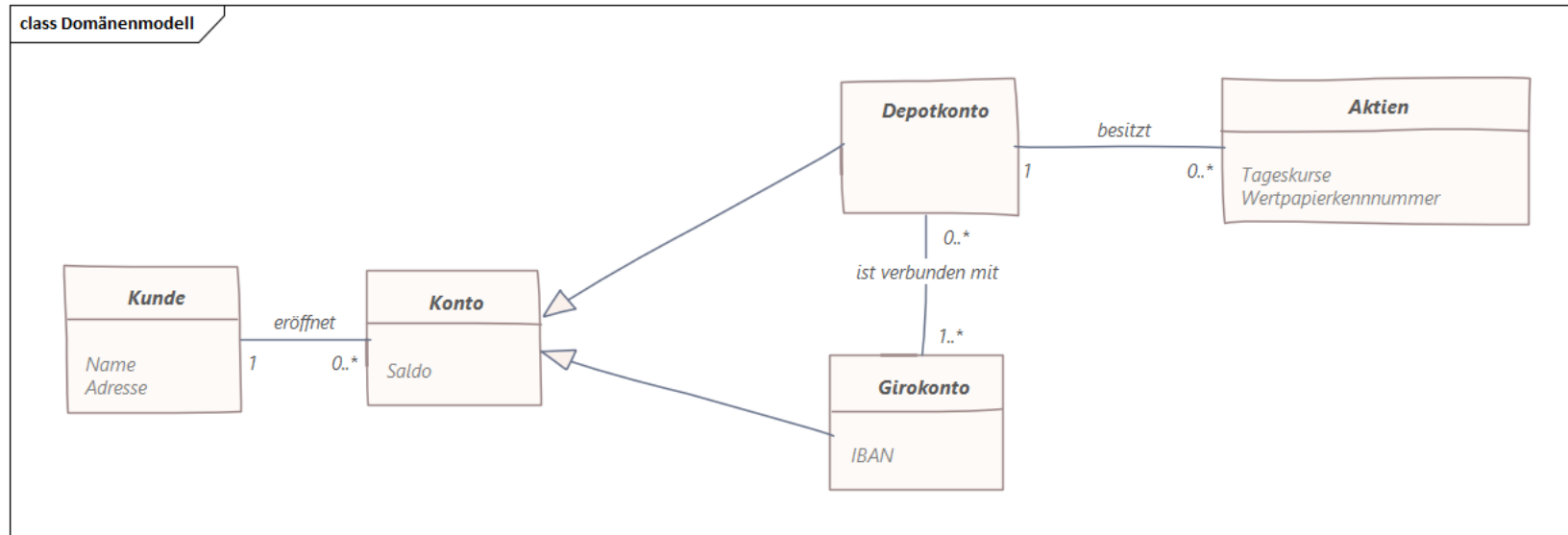
Gelb = Klassen

Grün = Attributen

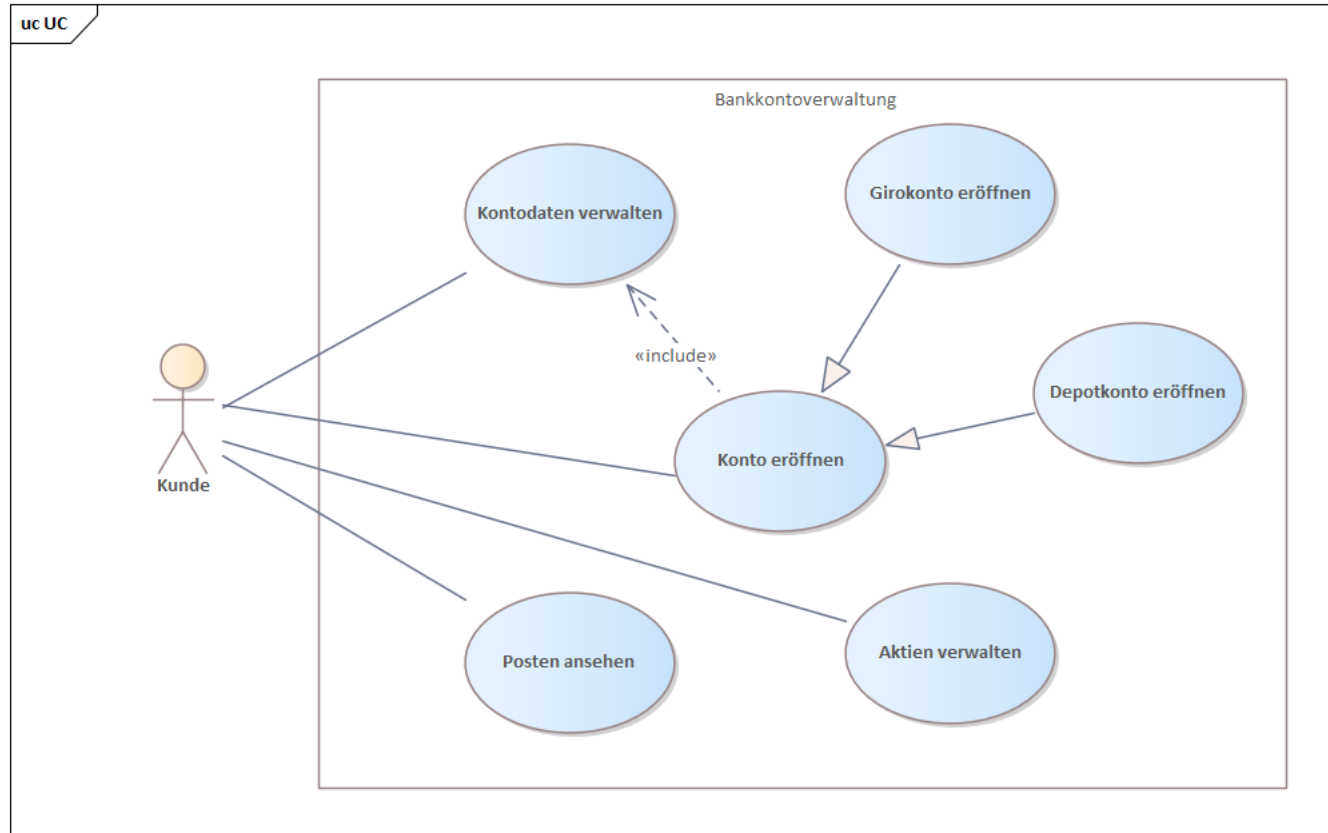
Blau = Use-Cases

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.

Epics und User-Stories

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

Akzeptanzkriterium:

- 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.

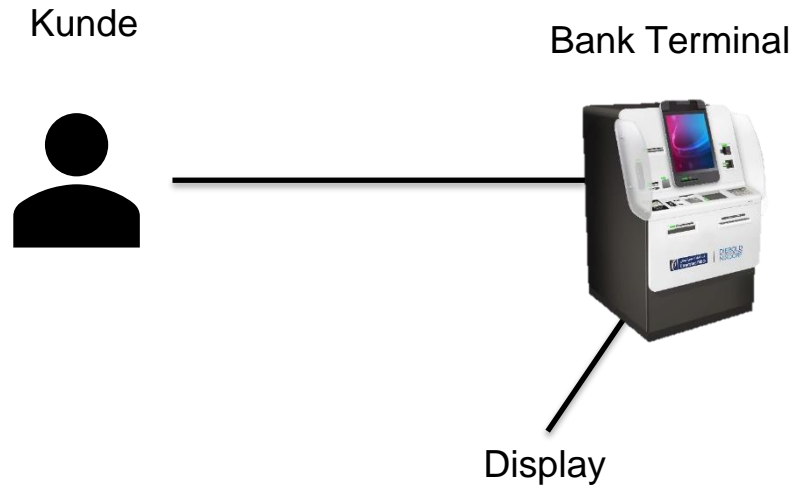
Epics und User-Stories

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

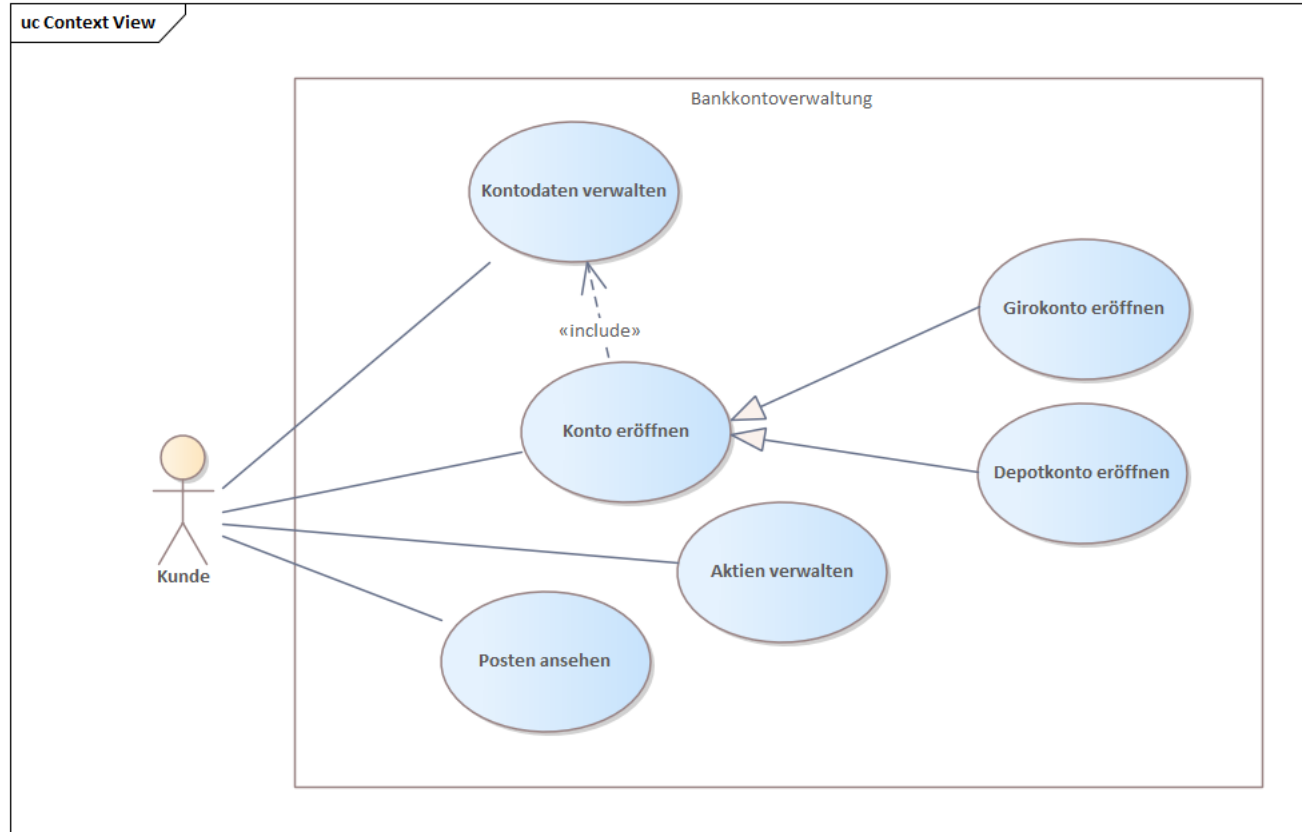
Akzeptanzkriterium:

- 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

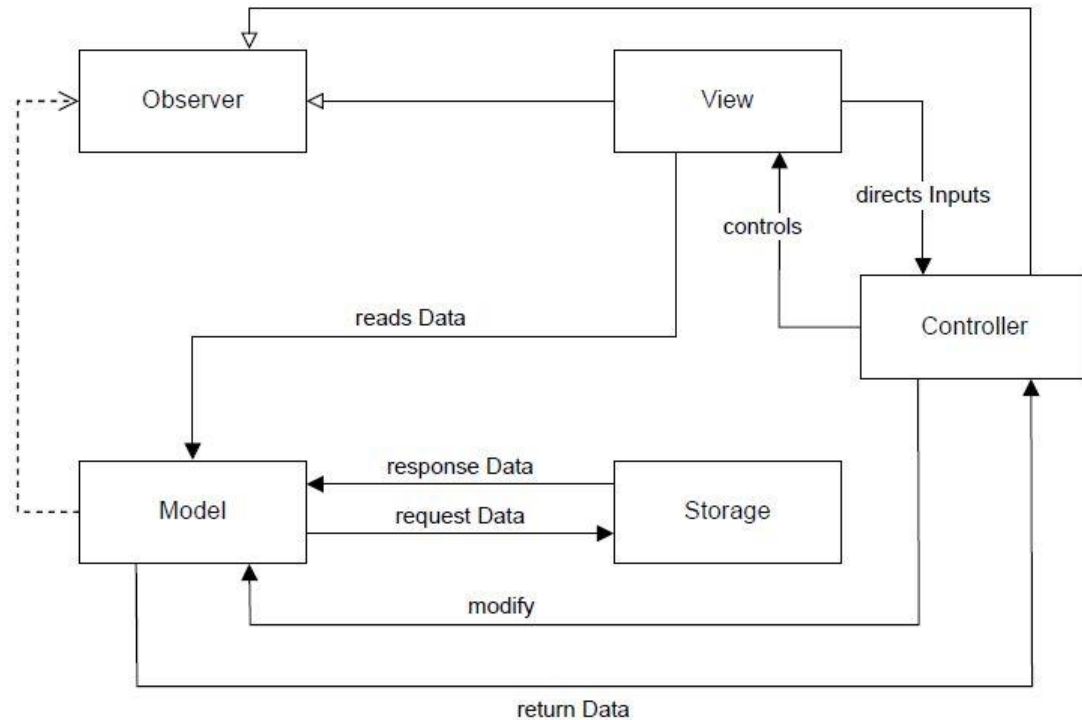


Context View: Fachlicher Überblick

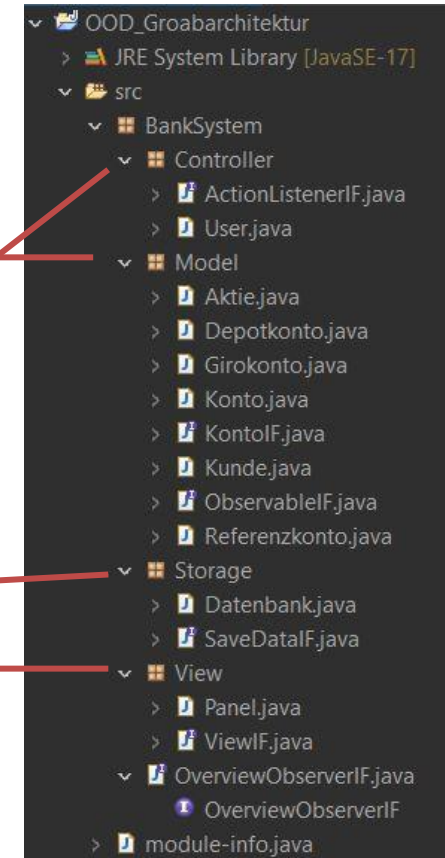
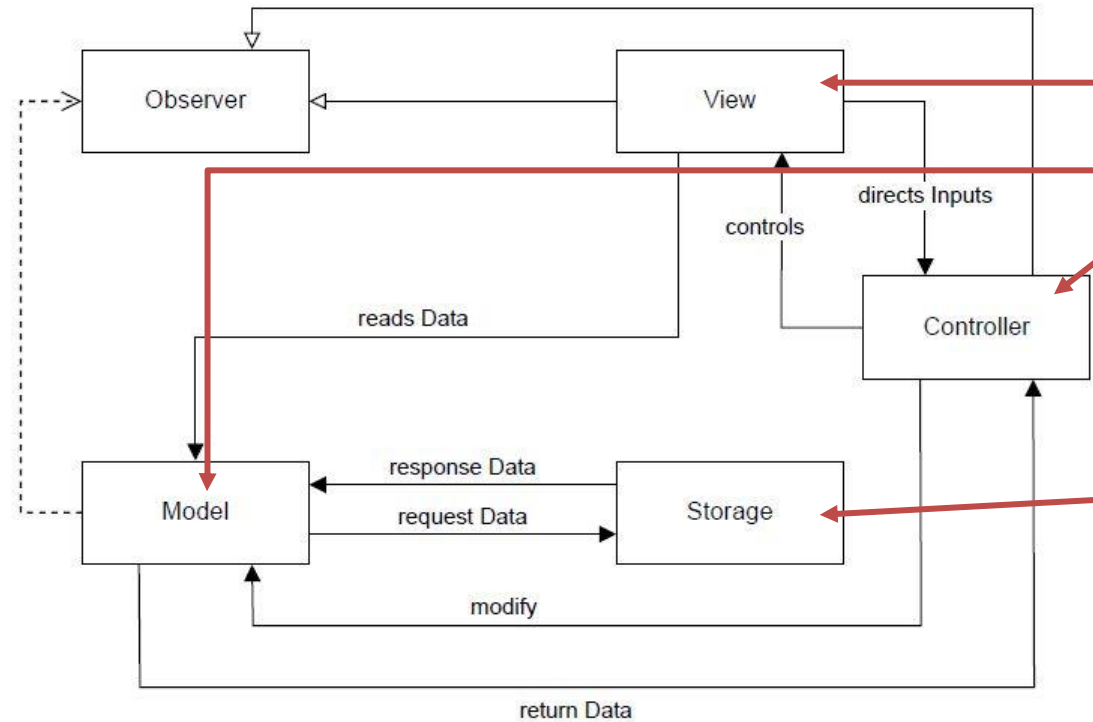


Grobarchitektur: Architekturstil

MVC + Storage

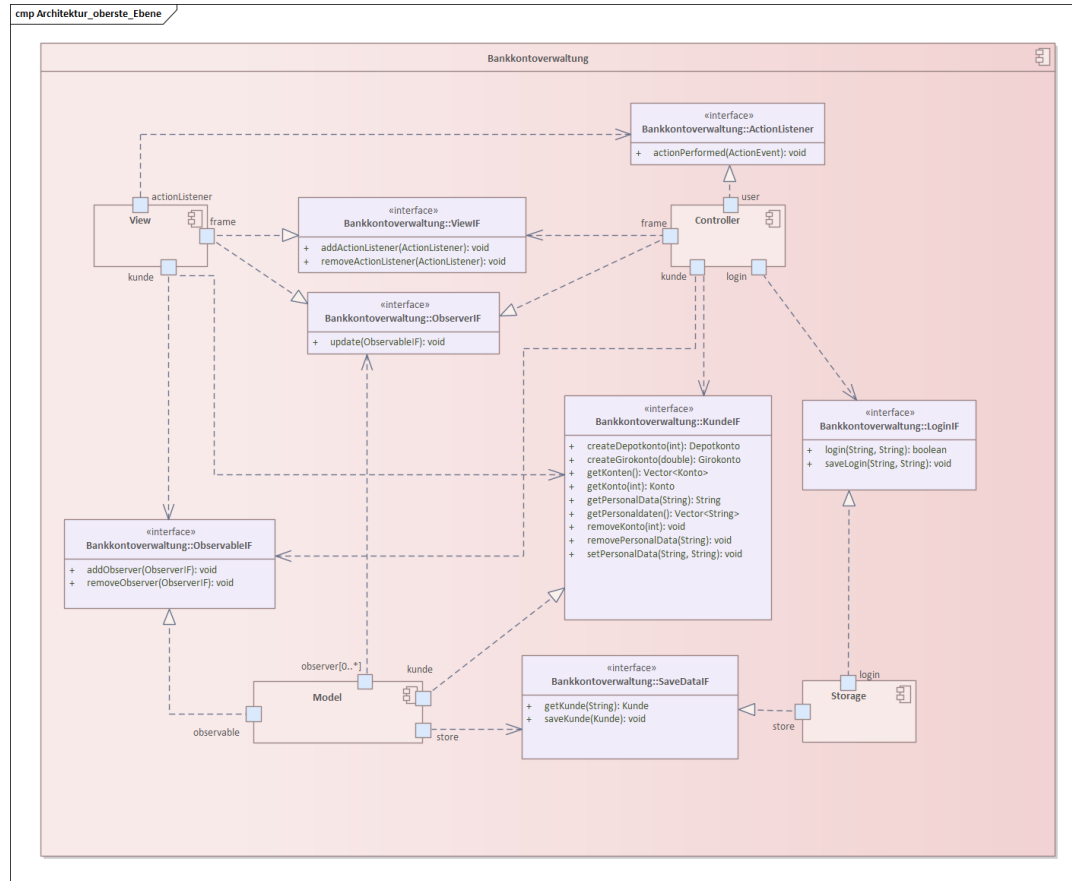


Grobarchitektur: Codeabbildung



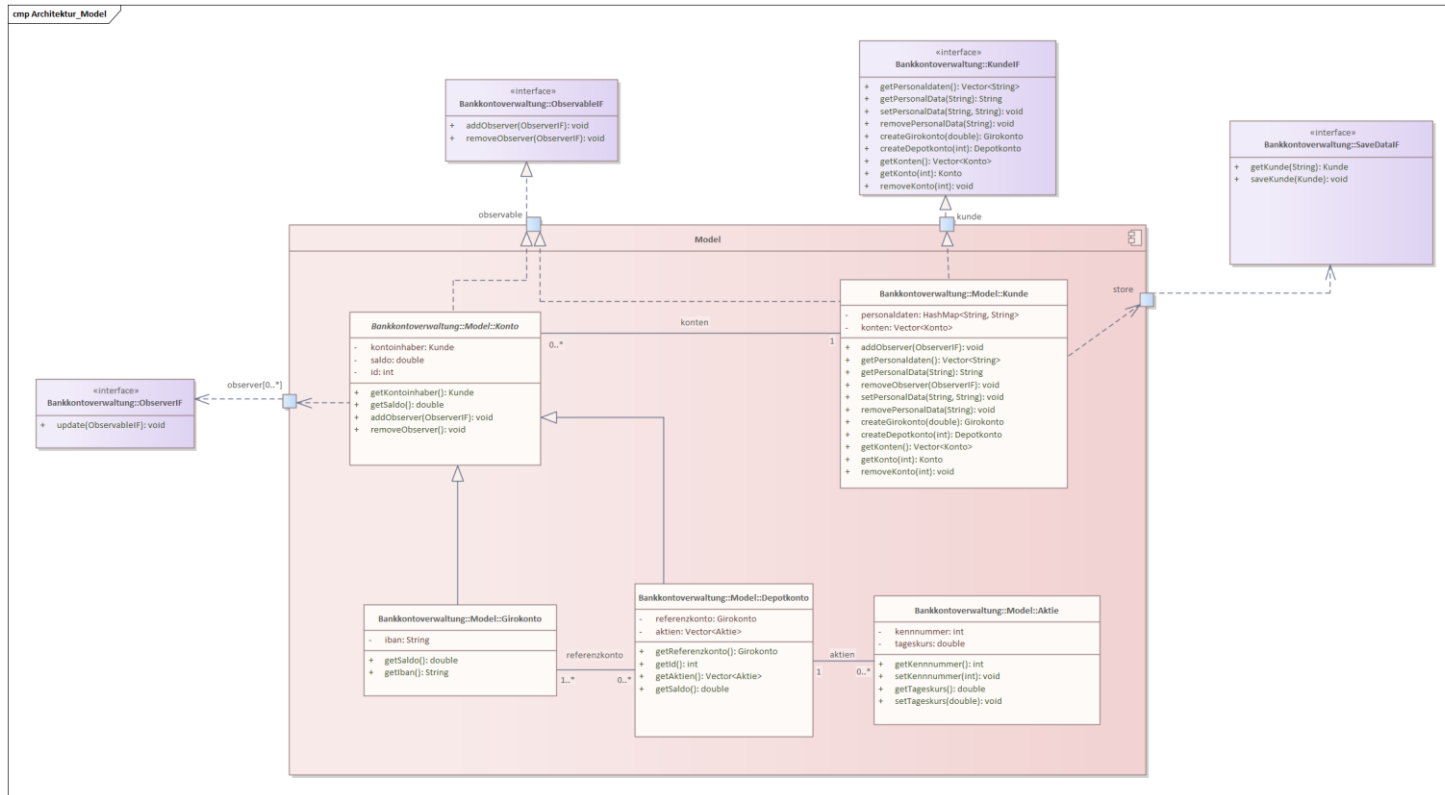
Fachliche Architekturebene: Structural View

Top Level



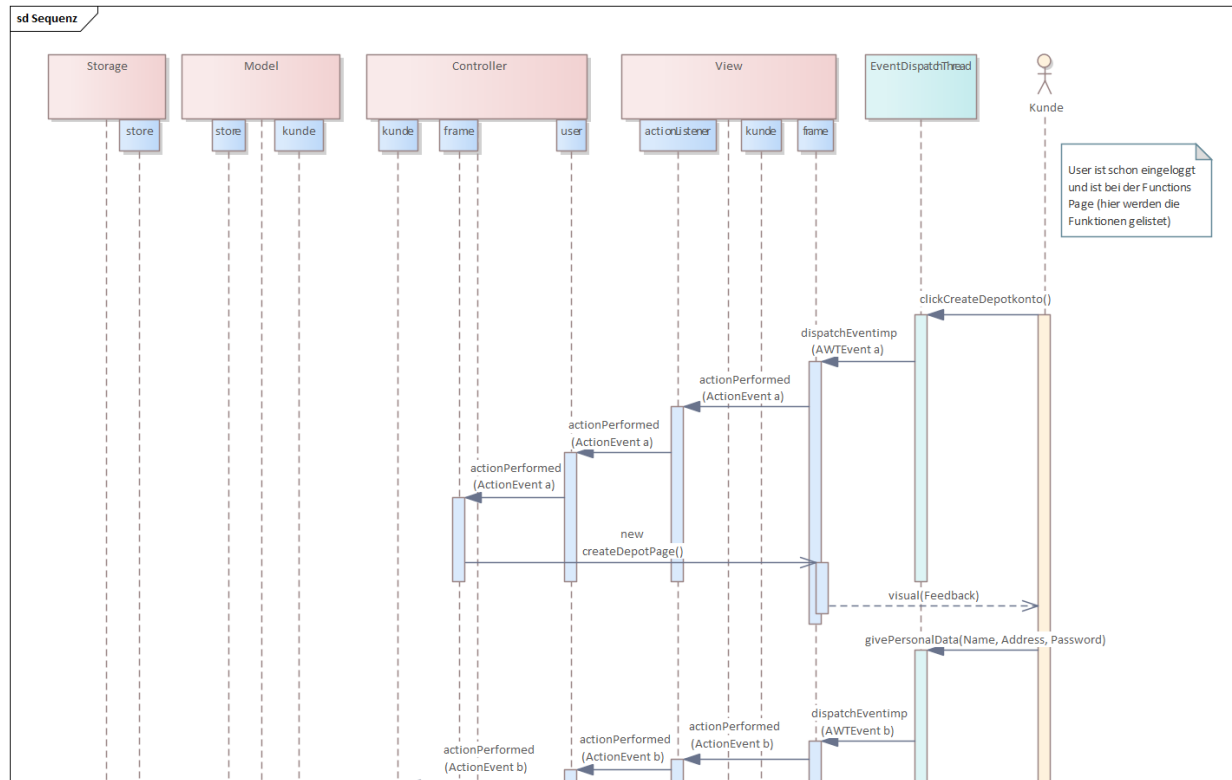
Fachliche Architekturebene: Structural View

Komponente Model



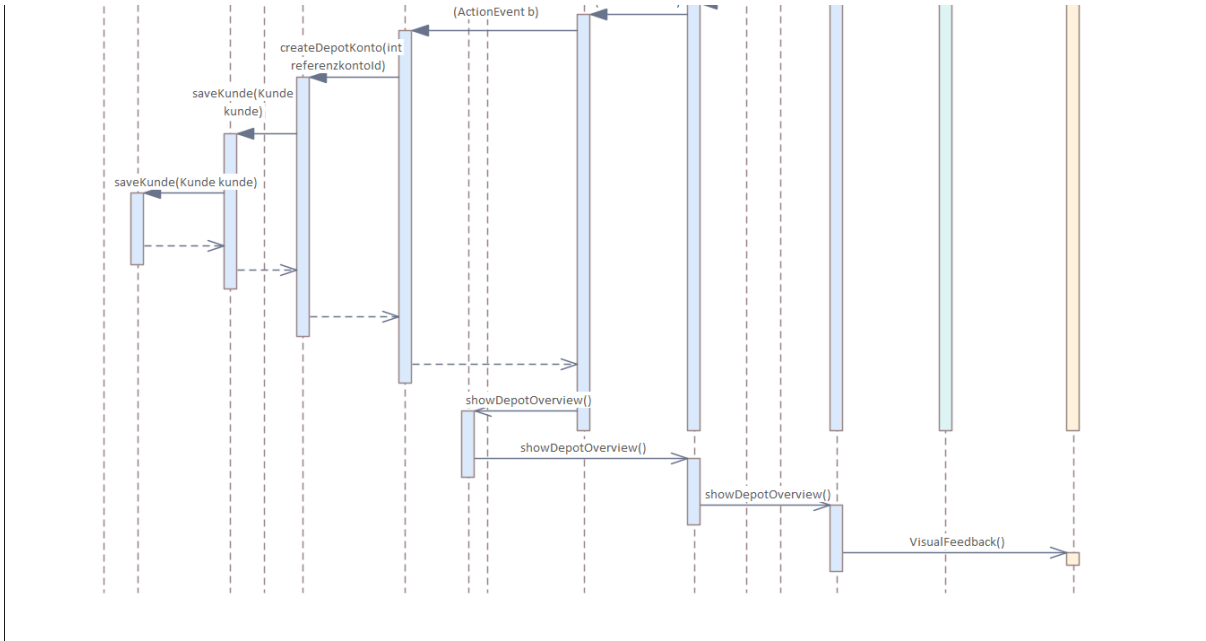
Fachliche Architekturebene: Behavioral View

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

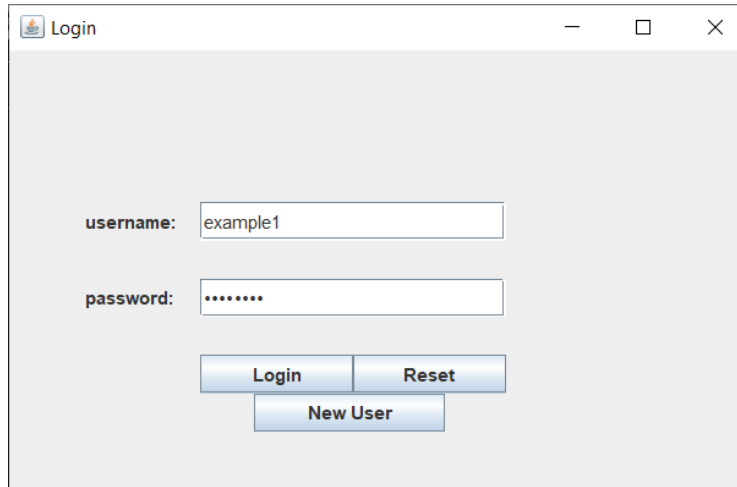
- Only the following functions have been implemented:
 - Create Giro
 - Create Depot
 - Accounts overview
- Example user credentials:
 - user: example1
 - password: password12345
- Javadoc located in doc folder

Credits

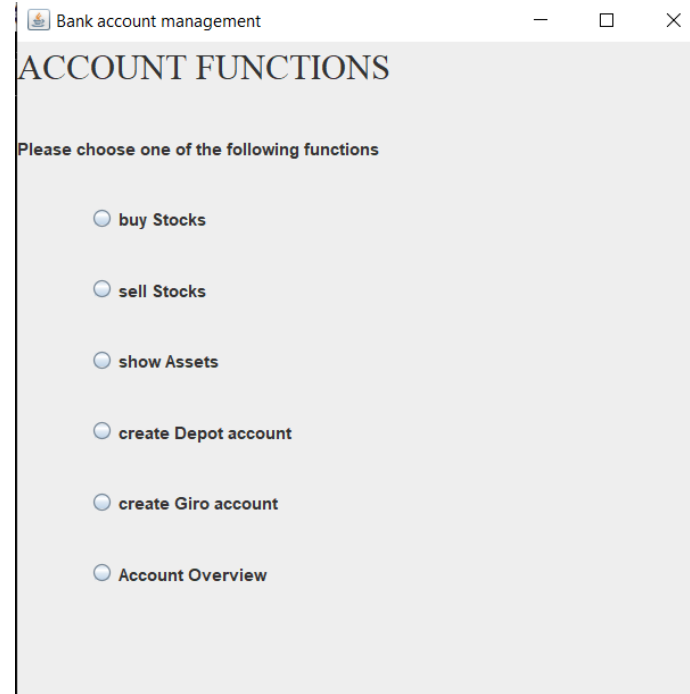
Made by SWT Group 2:

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

Demonstration des Prototypen und des Readme



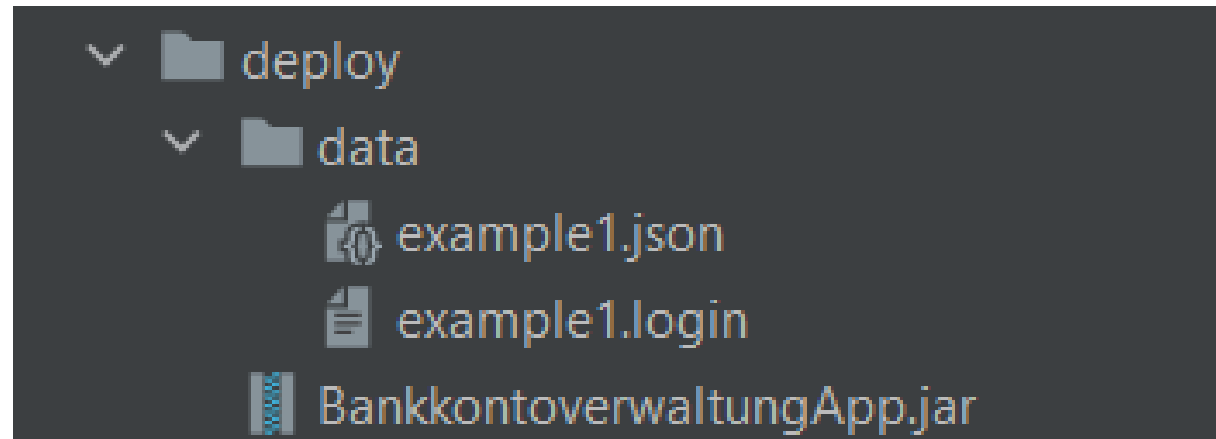
A screenshot of a web browser window titled "Login". The window has a light gray background. It contains two input fields: "username:" with the text "example1" and "password:" with masked characters ".....". Below the password field are three buttons: "Login" and "Reset" side-by-side, and "New User"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



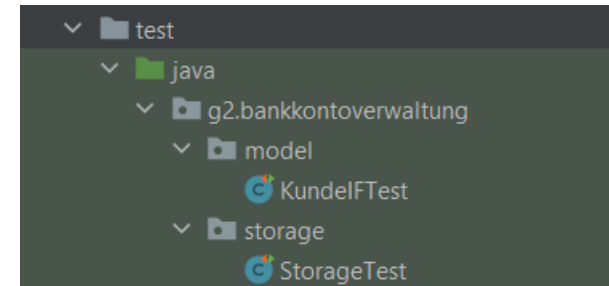
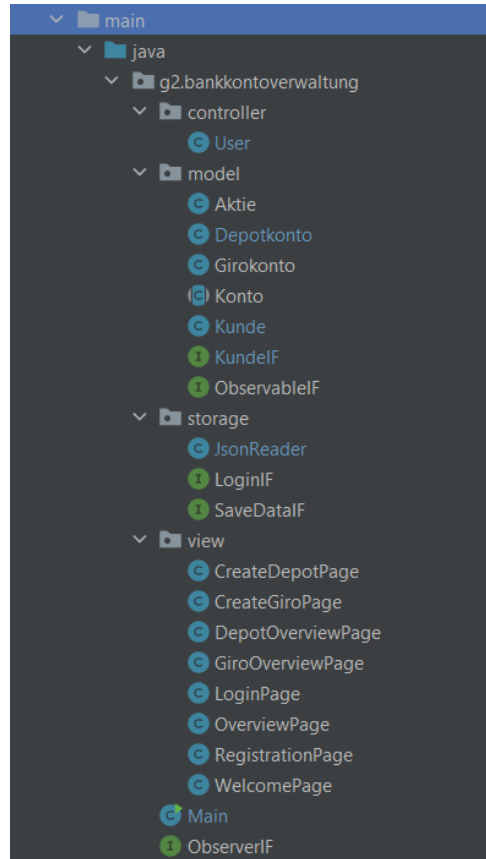
A screenshot of a web browser window titled "Bank account management". The window has a light gray background. It features a heading "ACCOUNT FUNCTIONS" and a prompt "Please choose one of the following functions". Below this are six radio button options: "buy Stocks", "sell Stocks", "show Assets", "create Depot account", "create Giro account", and "Account Overview".

Demonstration des Prototypen und des Readme

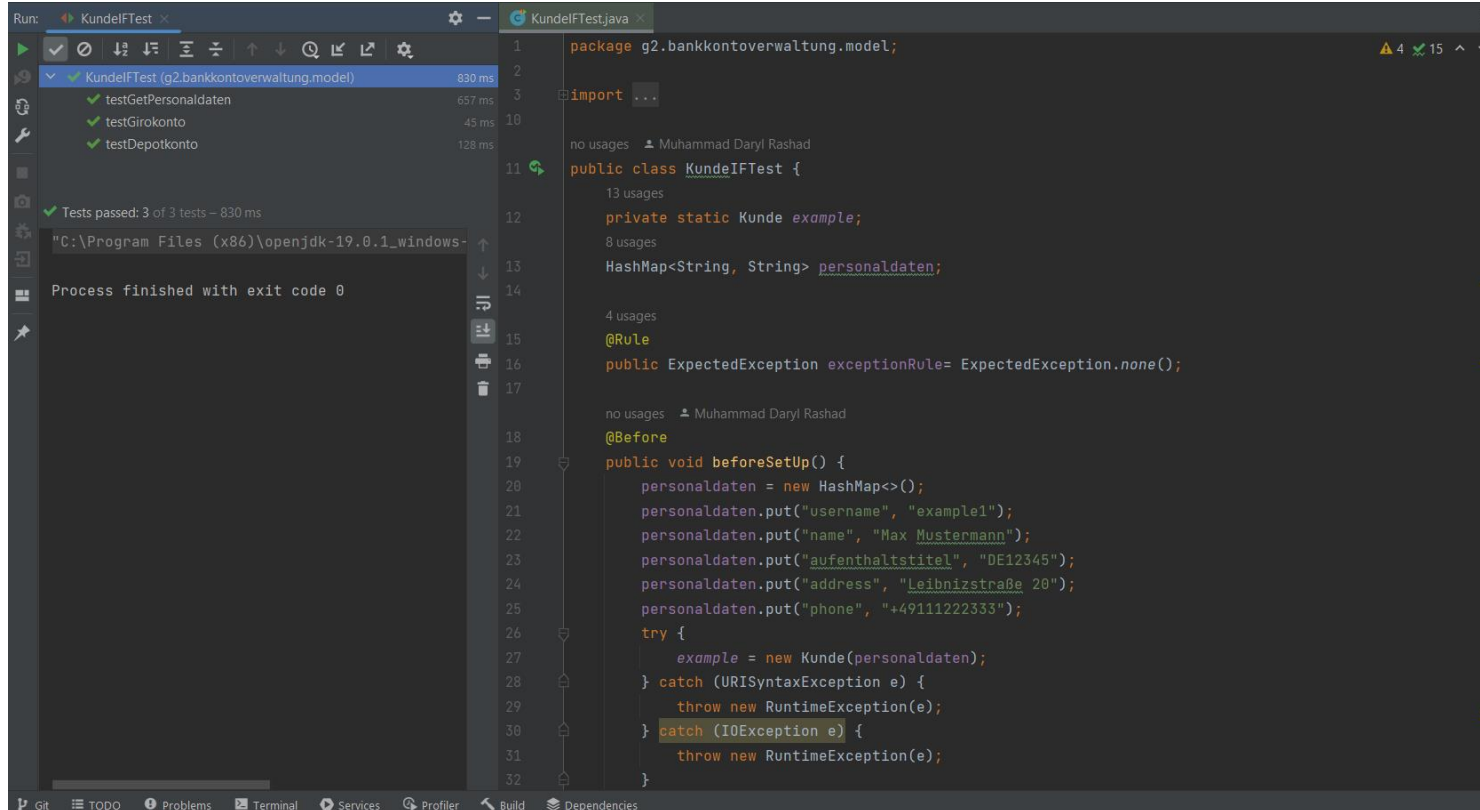
- Die Daten werden im Ordner data gespeichert
- JSON für Kundendaten
- .login für Password



Demonstration der Klassen- und Paketstruktur



Präsentation der Testfälle



The screenshot shows an IDE with two main panels. The left panel displays the test results for `KundelFTest` (package `g2.bankkontoverwaltung.model`). The tests passed are:

- `testGetPersonaldaten` (657 ms)
- `testGirokonto` (45 ms)
- `testDepotkonto` (128 ms)

Overall, 3 of 3 tests passed in 830 ms. The terminal below shows the process finished with exit code 0.

The right panel shows the source code for `KundelFTest.java`:

```
package g2.bankkontoverwaltung.model;

import ...

no usages  Muhammad Daryl Rashad

public class KundeIFTest {
    13 usages
    private static Kunde example;
    8 usages
    HashMap<String, String> personaldaten;

    4 usages
    @Rule
    public ExpectedException exceptionRule= ExpectedException.none();

    no usages  Muhammad Daryl Rashad
    @Before
    public void beforeSetUp() {
        personaldaten = new HashMap<>();
        personaldaten.put("username", "example1");
        personaldaten.put("name", "Max Mustermann");
        personaldaten.put("aufenthaltstitel", "DE12345");
        personaldaten.put("address", "Leibnizstraße 20");
        personaldaten.put("phone", "+49111222333");
        try {
            example = new Kunde(personaldaten);
        } catch (URISyntaxException e) {
            throw new RuntimeException(e);
        } catch (IOException e) {
            throw new RuntimeException(e);
        }
    }
}
```


Präsentation der Testfälle

```
KundelfTest.java x
no usages  Muhammad Daryl Rashad
35  @Test
36  public void testGetPersonalDaten() {
37      HashMap<String, String> exampleDaten = example.getPersonalDaten();
38
39      assert exampleDaten.get("username") == personalDaten.get("username");
40      assert exampleDaten.get("reiseepass") == null;
41  }
42
43  no usages  Muhammad Daryl Rashad
44  @Test
45  public void testGirokonto() throws IOException, URISyntaxException {
46      // Test creation of Girokonto
47      Girokonto testGirokonto = example.createGirokonto( anfangssaldo: 100);
48      assert example.getKonto( id: 0).equals(testGirokonto);
49
50      // Test to get non-existent Konto ID
51      exceptionRule.expect(ArrayIndexOutOfBoundsException.class);
52      example.getKonto( id: 1);
53
54      // Test remove konto
55      exceptionRule.expect(ArrayIndexOutOfBoundsException.class);
56      example.removeKonto( id: 0);
57      example.getKonto( id: 0);
58      example.removeKonto( id: 0);
59  }
60
```

```
KundelfTest.java x
60
61  no usages  Muhammad Daryl Rashad
62  @Test
63  public void testDepotkonto() throws IOException, URISyntaxException {
64      // Test creation of Depotkonto
65      example.createGirokonto( anfangssaldo: 100);
66      Depotkonto testDepotkonto = example.createDepotkonto( referenzkontoid: 0);
67      assert example.getKonto( id: 1).equals(testDepotkonto);
68
69      // Test invalid Anfangssaldo
70      exceptionRule.expect(IllegalArgumentException.class);
71      example.createGirokonto( anfangssaldo: -100);
72
73      // Test invalid id
74      exceptionRule.expect(IllegalArgumentException.class);
75      example.createDepotkonto( referenzkontoid: -1);
76  }
77
78  no usages  Muhammad Daryl Rashad
79  @After
80  public void cleanUp() {
81      File[] allContents = new File( pathname: "./data").listFiles();
82      for (File file : allContents) {
83          file.delete();
84      }
85  }
86
```

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>	
Depotkonto	<code>createDepotkonto(int referenzkontoId)</code>	Create a new Depotkonto for the user
Girokonto	<code>createGirokonto(double anfangssaldo)</code>	Create a new Girokonto for the user
Vector[Ⓔ]<Konto>	<code>getKonten()</code>	Get all Konto belonging to user
Konto	<code>getKonto(int id)</code>	Get a specific Konto belonging to the user
HashMap[Ⓔ]<String[Ⓔ],String[Ⓔ]>	<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 java.lang.Object[Ⓔ]		
<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

Präsentation der Code-Dokumentation und Metrik- Auswertung in GitLab

Analysis of prototype

General Information

Total lines of code: 697

Number of classes: 21

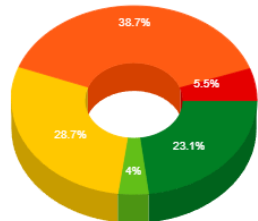
Number of packages: 5

Number of external packages: 16

Number of external classes: 63

Number of problematic classes: 0

Number of highly problematic classes: 0

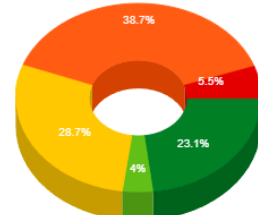


Lack of Cohesion of Methods

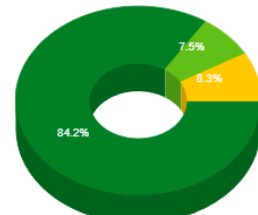
- Very High
- High
- Medium-high
- Low-medium
- Low

Distribution of Quality Attributes

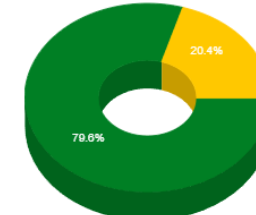
Complexity, Coupling, Cohesion, and Size



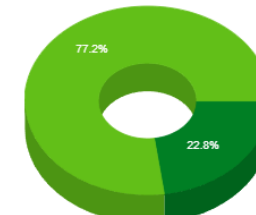
Lack of Cohesion of Methods



Lack of Cohesion



Coupling



Complexity