DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe

Advanced Software Engineering

Programmentwurf - Protokoll

von

Vorname Nachname (123456)

Vorname Nachname (123456)

Vorname Nachname (123456)

Namen + Matrikelnummern einfügen

Abgabedatum 04. Mai 2025

Inhaltsverzeichnis

1 Einführung (4P)

1.1 Übersicht über die Applikation (1P)

[Was macht die Applikation? Wie funktioniert sie? Welches Problem löst sie/welchen Zweck hat sie?]

Bilanzius ist ein Finanzsystem, welches die Finanzen Mehrer Benutzter verwalten soll. Dazu zählen das Verwalten von mehreren Konten, Geld sowie Kategorien.

1.2 Starten der Applikation (1P)

[Wie startet man die Applikation? Was für Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

Gebaute App:

- Im Terminal *java –jar bilanzius.jar* ausführen

Code:

- Projekt in preferierter IDE öffnen
- Datei org.bilanzius.Main öffnen
- main() Funktion über die IDE starten

1.3 Technischer Überblick (2P)

[Nennung und Erläuterung der Technologien (z.B. Java, MySQL, ...), jeweils Begründung für den Einsatz der Technologien]

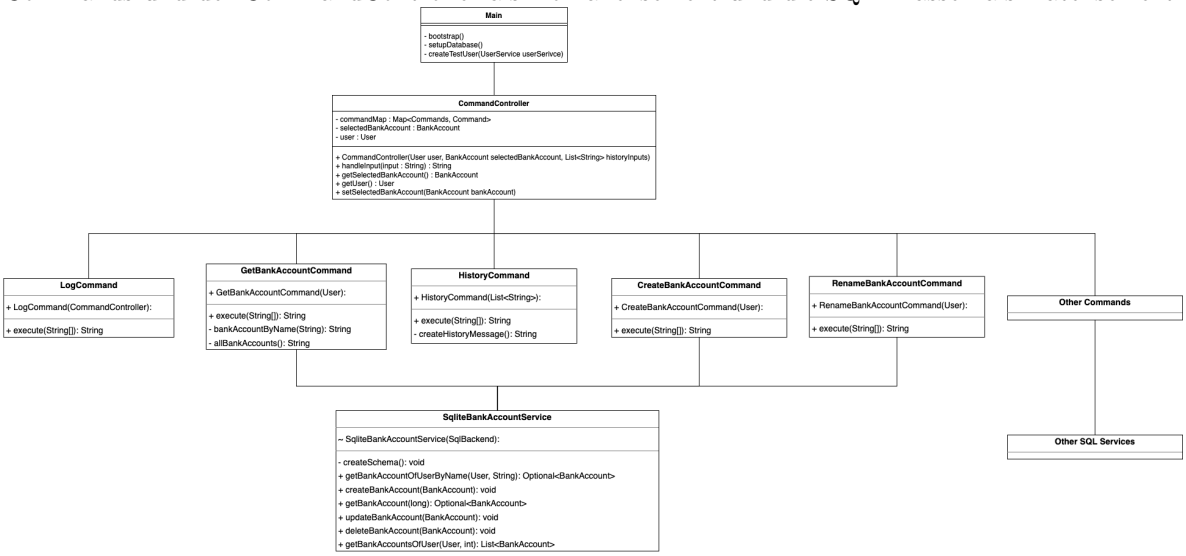
- Java – In der Vorlesung verwendete Programmiersprache.
- SQLite – Lokale Datenbank, die keine extra Installation benötigt.
- Maven – Dependency Manager sowie Build Tool
- GSON – Bibliothek, um JSON-Daten zu verwalten
- JUnit - Als Framework für automatisierte Softwaretests

2 Softwarearchitektur (8P)

2.1 Gewählte Architektur (4P)

[In der Vorlesung wurden Softwarearchitekturen vorgestellt. Welche Architektur wurde davon umgesetzt? Analyse und Begründung inkl. UML der wichtigsten Klassen, sowie Einordnung dieser Klassen in die gewählte Architektur]

Das Projekt basiert auf einem Schichtenmodel, wobei die Main Klasse als Präsentationschicht dient, die Commands und der CommandController als Domänenschicht und die SQL-Klassen als Datenschicht.



2.2 Domain Code (1P)

[kurze Erläuterung in eigenen Worten, was Domain Code ist – 1 Beispiel im Code zeigen, das bisher noch nicht gezeigt wurde]

Domain Code bezeichnet den Code, der die Geschäftslogik einer Anwendung enthält. Er beschreibt die Kernlogik eines Systems und bildet die Fachdomäne (Domain) des Unternehmens oder Projekts ab.

```
1  private String convertCurrency(String currencyCode, String currencyName)
2  {
3      JsonObject jsonObject = Requests.getRequest(currencyUrl);
4      BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, currencyCode));
5      BigDecimal balance;
6
7      try {
8          balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
       getBalance();
9      } catch (DatabaseException e) {
10         return localization.getMessage("database_error", e.toString());
11     }
12
13     return localization.getMessage("convert_balance", currencyName, (balance.multiply(exchangeRate)));
14 }
```

2.3 Analyse der Dependency Rule (3P)

[In der Vorlesung wurde im Rahmen der 'Clean Architecture' die s.g. Dependency Rule vorgestellt. Je 1 Klasse zeigen, die die Dependency Rule einhält und 1 Klasse, die die Dependency Rule verletzt; jeweils UML (mind. die betreffende Klasse inkl. der Klassen, die von ihr abhängen bzw. von der sie abhängt) und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]

Positiv-Beispiel: Dependency Rule

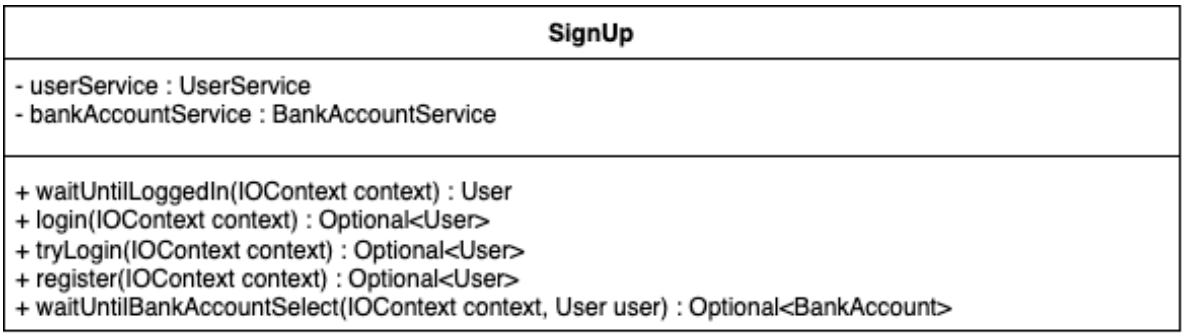
Negativ-Beispiel: Dependency Rule

3 SOLID (8P)

3.1 Analyse SRP (3P)

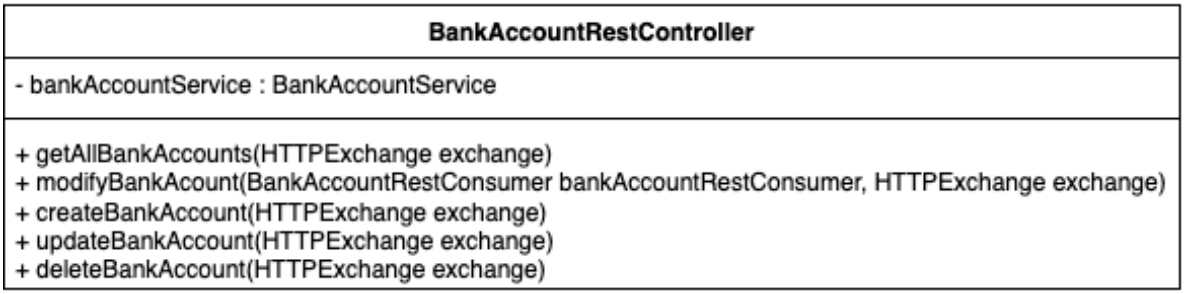
[jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

Positiv-Beispiel



Der BankAccountRestController ist eine Klasse, welche die CRUD-Endpunkte für die Bankkonten anbietet.

Negativ-Beispiel



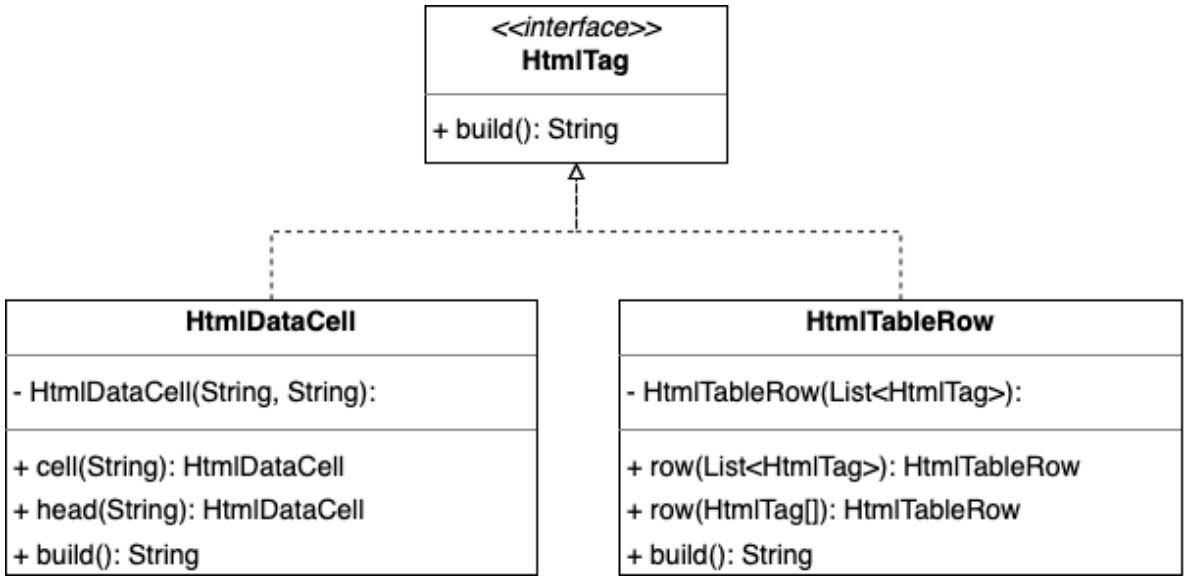
Die SingUp-Klasse ist für das Anmelden und Registrieren eines Benutzers vor dem Programm dar. Dadurch das die Methoden waitUntilLoggedIn() sowie waitUntilBankAccountIsSelect() eher weniger mit dem Anmelden zu tun haben, müsste man diese in eine neue Klasse wie zum Beispiel WaitUntil extrahieren.

3.2 Analyse OCP (3P)

[jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

Positiv-Beispiel

Um auf der HTMLTag-Klasse aufzubauen wurde ein Interface daraus erstellt und Klassen wie HTMLTableRow oder HTMLTableCell erben von. HTMLTag die bestimmten Methoden. Somit wurde das OCP-Prinzip erfüllt.



Negativ-Beispiel

Negativ Beispiel

3.3 Analyse [LSP/ISP/DIP] (2P)

[jeweils eine Klasse als positives und negatives Beispiel für entweder LSP oder ISP oder DIP; jeweils UML und Begründung, warum hier das Prinzip erfüllt/nicht erfüllt wird; beim Negativ-Beispiel UML einer möglichen Lösung hinzufügen]

[Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP und ein negatives Beispiel für ISP genommen werden]

Positiv-Beispiel

Positiv Beispiel

Negativ-Beispiel

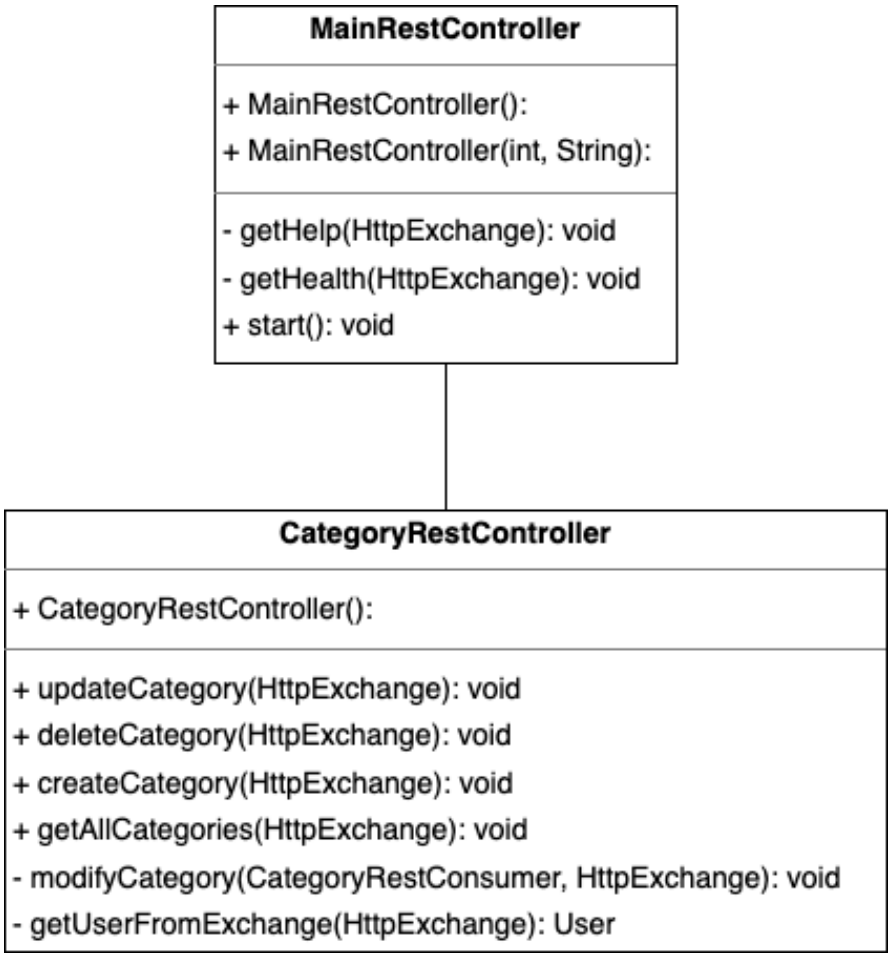
Negativ Beispiel

4 Weitere Prinzipien (8P)

4.1 Analyse GRASP: Geringe Kopplung (3P)

[eine bis jetzt noch nicht behandelte Klasse als positives Beispiel geringer Kopplung; UML mit zusammenspielenden Klassen, Aufgabenbeschreibung der Klasse und Begründung, warum hier eine geringe Kopplung vorliegt; es müssen auch die Aufrufer/Nutzer der Klasse berücksichtigt werden]

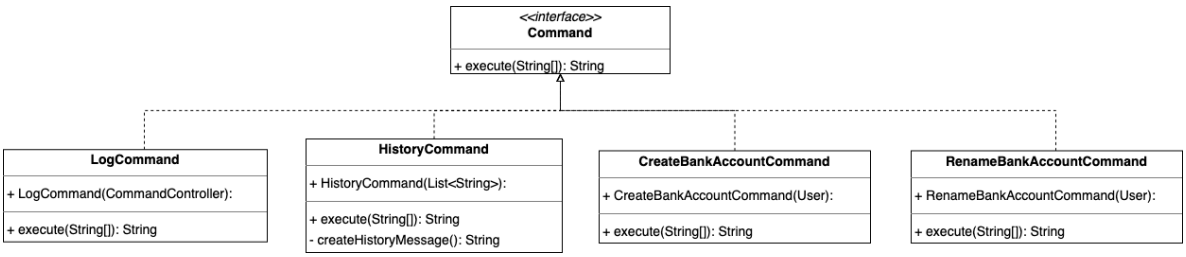
Die Klasse CategoryRestController ist für die Anbindung der CRUD-Endpunkte für den HTTP-Server für alle Kategorien eines Nutzers. Sie hat eine geringe Kopplung da sie nur für CRUD-Endpunkte für Kategorien dar sind. Davor waren alle Endpunkte im MainRestController.



4.2 Analyse GRASP: [Polymorphismus/Pure Fabrication] (3P)

[eine Klasse als positives Beispiel entweder von Polymorphismus oder von Pure Fabrication; UML Diagramm und Begründung, warum es hier zum Einsatz kommt]

Damit alle Commands gleich sind und keine Methoden fehlen wurde ein Interface geschrieben, welches jeder Command erbt.



4.3 DRY (2P)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher) einfügen; begründen und Auswirkung beschreiben – ggf. UML zum Verständnis ergänzen]

Branch: *main/Commit bc6ad15*

Vorher

```
private String convertToBitcoin()
{

    JsonObject jsonObject = Requests.getRequest(currencyUrl);
    assert jsonObject != null;
    BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "btc"));
    BigDecimal balance;

    try {
        balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
     getBalance();
    } catch (DatabaseException e) {
        return localization.getMessage("database_error", e.toString());
    }

    return localization.getMessage("convert_balance", "Bitcoin", (balance.multiply(exchangeRate)));
}

private String convertToGermanDeutscheMark()
{

    JsonObject jsonObject = Requests.getRequest(currencyUrl);
    assert jsonObject != null;
    BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "dem"));
    BigDecimal balance;
       try {
        balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
     getBalance();
    } catch (DatabaseException e) {
        return localization.getMessage("database_error", e.toString());
    }

    return localization.getMessage("convert_balance", "Deutsche Mark", (balance.multiply(exchangeRate)))
     ;
}

private String convertToSwissFranc()
{

    String currency = "Swiss Franc";
    if (localization.getCurrentLanguageCode().equals("de")) {
        currency = "Schweizer Franken";
    }

    JsonObject jsonObject = Requests.getRequest(currencyUrl);
    assert jsonObject != null;
    BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "chf"));
    BigDecimal balance;
```

```
46
47      try {
48          balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
49      } catch (DatabaseException e) {
50          return localization.getMessage("database_error", e.toString());
51      }
52
53      return localization.getMessage("convert_balance", currency, (balance.multiply(exchangeRate)));
54  }
55
56  private String convertToDogecoin()
57  {
58
59      JsonObject jsonObject = Requests.getRequest(currencyUrl);
60      assert jsonObject != null;
61      BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "doge"));
62      BigDecimal balance;
63
64      try {
65          balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
66      } catch (DatabaseException e) {
67          return localization.getMessage("database_error", e.toString());
68      }
69
70      return localization.getMessage("convert_balance", "Deutsche Mark", (balance.multiply(exchangeRate)))
        ;
71  }
72
73  private String convertToSwissFranc()
74  {
75
76      String currency = "Swiss Franc";
77      if (localization.getCurrentLanguageCode().equals("de")) {
78          currency = "Schweizer Franken";
79      }
80
81      JsonObject jsonObject = Requests.getRequest(currencyUrl);
82      assert jsonObject != null;
83      BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "chf"));
84      BigDecimal balance;
85
86      try {
87          balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
88      } catch (DatabaseException e) {
89          return localization.getMessage("database_error", e.toString());
90      }
91
92      return localization.getMessage("convert_balance", currency, (balance.multiply(exchangeRate)));
93  }
94
95  private String convertToDogecoin()
96  {
97
98      JsonObject jsonObject = Requests.getRequest(currencyUrl);
99      assert jsonObject != null;
100     BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "doge"));
101     BigDecimal balance;
102
103     try {
104         balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
105     } catch (DatabaseException e) {
106         return localization.getMessage("database_error", e.toString());
107     }
108 }
```

Nachher

```
private String convertCurrency(String currencyCode, String currencyName) {
    JsonObject jsonObject = Requests.getRequest(currencyUrl);
    assert jsonObject != null;
    BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, currencyCode));
    BigDecimal balance;

    try {
        balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
     getBalance();
    } catch (DatabaseException e) {
        return localization.getMessage("database_error", e.toString());
    }

    return localization.getMessage("convert_balance", currencyName, (balance.multiply(exchangeRate)));
}

private String convertToBitcoin()
{
    return convertCurrency("btc", "Bitcoin");
}

private String convertToGermanDeutscheMark()
{
    return convertCurrency("dem", "Deutsche Mark");
}

private String convertToSwissFranc()
{
    String currency = localization.getCurrentLanguageCode().equals("de") ? "Schweizer Franken" : "Swiss
     Franc";
    return convertCurrency("chf", currency);
}

private String convertToDogecoin()
{
    return convertCurrency("doge", "Dogecoin");
}

private String convertToEthereum()
{
    return convertCurrency("eth", "Ethereum");
}

private String convertToHongKongDollar()
{
    return convertCurrency("hkd", "Hong Kong Dollar");
}

private String convertToJamaicanDollar()
{
    String currency = localization.getCurrentLanguageCode().equals("de") ? "Jamaikanische Dollar" : "
     Jamaican Dollar";
    return convertCurrency("jmd", currency);
}

private String convertToNorthKoreanWon()
{
    String currency = localization.getCurrentLanguageCode().equals("de") ? "Nordkoreanische Won" : "
     North Korean Won";
    return convertCurrency("kpw", currency);
}

private String convertToRussianRuble()
{
    String currency = localization.getCurrentLanguageCode().equals("de") ? "Russischer Rubel" : "Russian
      Ruble";
    return convertCurrency("rub", currency);
}

private String convertToUsDollar()
{
```

```
67      return convertCurrency("usd", "US Dollar");
68  }
```

Beschreibung

Es gab viele Methoden die, denn selben Code in einer kleinen abgeänderten Art ausgeführt haben. Die Lösung war eine Methode namens *convertCurrency(String currencyCode, String currencyName)* die in den einzelnen Methoden für die jeweilige Währung aufgerufen wird.

Das hatte zur Folge das der Code einfacherer zu lesen ist und unnötige Codezeilen gespart werden konnten.

Beschreiben warum die Namen der Währungen gehart-coded sind (ggf. Alternativen aufzeigen)

5 Unit Tests (8P)

5.1 10 Unit Tests (2P)

[Zeigen und Beschreiben von 10 Unit-Tests und Beschreibung, was getestet wird]

1. **Find User by Name Test (SqliteUserDatabaseServiceTest)**
   Erstelle einen neuen Nutzer mit dem Name Test und prüfe ob dieser über die findUserWithName Funktion gefunden werden kann.

```
@Test
void testFindUserByName()
{
    // Setup test
    var service = userService();
    service.createUser(User.createUser(USERNAME, DEFAULT_PASSWORD));
    var test = "hallo";
    // Find user and validate
    var result = service.findUserWithName(USERNAME).orElseThrow();

    Assertions.assertEquals(1, result.getId());
    Assertions.assertEquals(USERNAME, result.getUsername());
    Assertions.assertEquals(DEFAULT_PASSWORD, result.getHashedPassword());
}
```

2. **Update User Password Test (SqliteUserDatabaseServiceTest)**
   Erstelle einen neuen Nutzer mit dem Name Test und einem Standard-Passwort. Das Passwort wird daraufhin geupdatet. Anschließend wird geprüft das neue Passwort gesetzt wurde.

```
@Test
void testUpdateUserPassword()
{
    // Setup test
    var service = userService();
    var creatingUser = User.createUser(USERNAME, DEFAULT_PASSWORD);

    // user can't be updated before creation
    Assertions.assertFalse(creatingUser.canBeUpdated());

    service.createUser(creatingUser);

    // Find user and update password
    var user = service.findUser(1).orElseThrow();
    Assertions.assertTrue(user.canBeUpdated());
    user.setHashedPassword(OTHER_PASSWORD);
    service.updateUserPassword(user);

    // Find user again
    var sameUser = service.findUser(1).orElseThrow();
    Assertions.assertEquals(OTHER_PASSWORD, sameUser.getHashedPassword());
}
```

3. **Get Localized Message Test (LocalizationTest)**
   Prüfe ob die Übersetzungsfunktion richtig funktioniert. In diesem Test wird geprüft, ob ein existenter Language Key richtig aufgelöst werden kann.

```
@Test
void testGetMessage()
{
    var language = Localization.getInstance();
    language.setLocale("en");

    Assertions.assertEquals("en", language.getCurrentLanguageCode());
    Assertions.assertEquals(EXISTING_KEY_VALUE, language.getMessage(EXISTING_KEY));
    Assertions.assertEquals("MISSING KEY: " + NON_EXISTING_KEY, language.getMessage(
     NON_EXISTING_KEY));
}
```

4. **Get Localized Message with Parameters (LocalizationTest)**
   Prüfe ob eine Übersetzung mit Parametern gefüllt werden kann.

```
@Test
void testGetMessageWithParams()
{
    var language = Localization.getInstance();
    language.setLocale("en");

    Assertions.assertEquals("en", language.getCurrentLanguageCode());
    Assertions.assertEquals(EXISTING_KEY_WITH_PARAMS_VALUE, language.getMessage(
     EXISTING_KEY_WITH_PARAMS, "1"));
}
```

5. **Table Cell Tests (HtmlDataCellTest)**
   Die HtmlDataCell Klasse generiert HTML Tabellen Tags für das Finanzbericht Report Feature. Es wird getestet, ob die Klasse gültige HTML-Tags ausgibt.

```
@Test
void testTableElements()
{
    var headComponent = HtmlDataCell.head("head");
    var dataComponent = HtmlDataCell.cell("data");

    Assertions.assertEquals("<th>head</th>", headComponent.build());
    Assertions.assertEquals("<td>data</td>", dataComponent.build());
}
```

6. **Create and Get Transaction (SqliteTransactionServiceTest)**
   Erstelle eine Zahlungstransaktion für einen Beispielnutzer und prüfe ob diese richtig gespeichert wurde.

```
@Test
void testCreateTransaction()
{
    var service = transactionService();
    service.saveTransaction(new Transaction(1, 1, 1, 1, BigDecimal.ZERO, Instant.now(), "1234"));
    service.saveTransaction(new Transaction(1, 1, 1, -1, BigDecimal.ZERO, Instant.now(), "1234"));

    var transactions = service.getTransactions(ModelUtils.existingUser(), ModelUtils.
     existingBankAccount(), 10, 0);
    Assertions.assertEquals(2, transactions.size());
}
```

7. **Create Transaction for Invalid User (SqliteTransactionServiceTest)**
   Wenn eine Transaktion für einen nicht existenten Benutzer erzeugt wird, soll ein Fehler geworfen werden.

```
@Test
void testCreateTransactionInvalidUser()
{
    var service = transactionService();

    Assertions.assertThrows(DatabaseException.class, () -> service.saveTransaction(new
     Transaction(1, 5, 1, 1, BigDecimal.ZERO, Instant.now(), "1234")));
}
```

8. **Database Provider Test (DatabaseProviderTest)**
   Prüfe ob die SqlDatabaseServiceRepository alle Services richtig erzeugt.

```
@Test
void testDatabaseProvider() throws SQLException
{
    DatabaseProvider.init(new SqlDatabaseServiceRepository(requestBackend()));
    Assertions.assertNotNull(DatabaseProvider.getBankAccountService());
    Assertions.assertNotNull(DatabaseProvider.getUserService());
    Assertions.assertNotNull(DatabaseProvider.getTransactionService());
    Assertions.assertNotNull(DatabaseProvider.getCategoryService());
```

```
9
10      Assertions.assertThrows(IllegalStateException.class, () -> DatabaseProvider.init(new
         SqlDatabaseServiceRepository(requestBackend())));
11  }
```

9. **Create and Get Bank Account (SqliteBankAccountServiceTest)**
   Legt einen Bankaccount für einen Benutzer an und prüft ob die Daten richtig gespeichert wurden.

```
1   @Test
2   void testCreateAndGet()
3   {
4       var service = bankAccountService();
5
6       service.createBankAccount(BankAccount.create(ModelUtils.existingUser(), NAME));
7       var account = service.getBankAccount(1).orElseThrow();
8
9       Assertions.assertEquals(NAME, account.getName());
10      Assertions.assertFalse(service.getBankAccountsOfUser(ModelUtils.existingUser(), 1).isEmpty());
11      Assertions.assertFalse(service.getBankAccountOfUserByName(ModelUtils.existingUser(), NAME).
         isEmpty());
12  }
```

10. **Delete Bank Account (SqliteBankAccountServiceTest)**
    Erzeugt ein Konto und löscht es.

```
1   @Test
2   void testDelete()
3   {
4       var service = bankAccountService();
5
6       service.createBankAccount(BankAccount.create(ModelUtils.existingUser(), NAME));
7       service.deleteBankAccount(service.getBankAccount(1).orElseThrow());
8
9       Assertions.assertTrue(service.getBankAccount(1).isEmpty());
10  }
```

5.2 ATRIP: Automatic, Thorough und Professional (2P)

[je Begründung/Erläuterung, wie 'Automatic', 'Thorough' und 'Professional' realisiert wurde – bei 'Thorough' zusätzlich Analyse und Bewertung zur Testabdeckung]

- **Automatic**
  Als Framework für die automatisierten Tests wird JUnit verwendet. Das Maven Surefire Plugin führt die Softwaretests direkt beim Kompilierungsprozess aus.
- **Thorough**
  Es werden Unit und Integrationstests verwendet. Da als Datenbank SQLite eingesetzt wird und die Datenbank dadurch lokal läuft, wird im Code selbst nicht zwischen Unit und Integrationstests unterschieden.

  Die Testabdeckung ist mit 19% nicht hoch, wobei der Datenbank-Layer mit einer Abdeckung von über 95% höher ist als der Rest. Die einzelnen Befehle werden aktuell nicht automatisiert getestet, dies liegt an dem hohen Aufwand
- **Professional**

5.3 Fakes und Mocks (4P)

[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten (die Fake/Mocks sind ohne Drittherstellter-Bibliothek/Framework zu implementieren); zusätzlich jeweils UML Diagramm mit Beziehungen zwischen Mock, zu mockender Klasse und Aufrufer des Mocks]

6 Domain Driven Design (8P)

6.1 Ubiquitous Language (2P)

[4 Beispiele für die Ubiquitous Language; jeweils Bezeichung, Bedeutung und kurze Begründung, warum es zur Ubiquitous Language gehört]

Bezeichung Bedeutung Begründung

6.2 Repositories (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist – NICHT, warum es nicht implementiert wurde]

6.3 Aggregates (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

6.4 Entities (1,5P)

[UML, Beschreibung und Begründung des Einsatzes einer Entity; falls keine Entity vorhanden: ausführliche Begründung, warum es keine geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

6.5 Value Objects (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

7 Refactoring (8P)

7.1 Code Smells (2P)

[jeweils 1 Code-Beispiel zu 2 unterschiedlichen Code Smells (die benannt werden müssen) aus der Vorlesung; jeweils Code-Beispiel und einen möglichen Lösungsweg bzw. den genommen Lösungsweg beschreiben (inkl. (Pseudo-)Code)]

Code Smell 1: Duplicated Code *main/Commit bc6ad15*

Sollte eigentlich OK sein, aber ich bin kein Fän davon, dass wir hier das gleiche Beispiel von DRY nutzen

Vorher

```
1  private String convertToBitcoin()
2  {
3
4      JsonObject jsonObject = Requests.getRequest(currencyUrl);
5      assert jsonObject != null;
6      BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "btc"));
7      BigDecimal balance;
8
9      try {
10         balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
       getBalance();
11     } catch (DatabaseException e) {
12         return localization.getMessage("database_error", e.toString());
13     }
14
15     return localization.getMessage("convert_balance", "Bitcoin", (balance.multiply(exchangeRate)));
16 }
17
18 private String convertToGermanDeutscheMark()
19 {
20
21     JsonObject jsonObject = Requests.getRequest(currencyUrl);
22     assert jsonObject != null;
23     BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "dem"));
24     BigDecimal balance;
25        try {
26         balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
       getBalance();
27     } catch (DatabaseException e) {
28         return localization.getMessage("database_error", e.toString());
29     }
30
31     return localization.getMessage("convert_balance", "Deutsche Mark", (balance.multiply(exchangeRate)))
       ;
32 }
33
34 private String convertToSwissFranc()
35 {
36
37     String currency = "Swiss Franc";
38     if (localization.getCurrentLanguageCode().equals("de")) {
39         currency = "Schweizer Franken";
40     }
41
42     JsonObject jsonObject = Requests.getRequest(currencyUrl);
43     assert jsonObject != null;
44     BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "chf"));
45     BigDecimal balance;
46
47     try {
48         balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
       getBalance();
49     } catch (DatabaseException e) {
```

```
50          return localization.getMessage("database_error", e.toString());
51      }
52
53      return localization.getMessage("convert_balance", currency, (balance.multiply(exchangeRate)));
54  }
55
56  private String convertToDogecoin()
57  {
58
59      JsonObject jsonObject = Requests.getRequest(currencyUrl);
60      assert jsonObject != null;
61      BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "doge"));
62      BigDecimal balance;
63
64      try {
65          balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
66      } catch (DatabaseException e) {
67          return localization.getMessage("database_error", e.toString());
68      }
69
70      return localization.getMessage("convert_balance", "Deutsche Mark", (balance.multiply(exchangeRate)))
        ;
71  }
72
73  private String convertToSwissFranc()
74  {
75
76      String currency = "Swiss Franc";
77      if (localization.getCurrentLanguageCode().equals("de")) {
78          currency = "Schweizer Franken";
79      }
80
81      JsonObject jsonObject = Requests.getRequest(currencyUrl);
82      assert jsonObject != null;
83      BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "chf"));
84      BigDecimal balance;
85
86      try {
87          balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
88      } catch (DatabaseException e) {
89          return localization.getMessage("database_error", e.toString());
90      }
91
92      return localization.getMessage("convert_balance", currency, (balance.multiply(exchangeRate)));
93  }
94
95  private String convertToDogecoin()
96  {
97
98      JsonObject jsonObject = Requests.getRequest(currencyUrl);
99      assert jsonObject != null;
100     BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, "doge"));
101     BigDecimal balance;
102
103     try {
104         balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
105     } catch (DatabaseException e) {
106         return localization.getMessage("database_error", e.toString());
107     }
108 }
```

Lösung

Der Code Smell wurde gelöst, indem die Methode *convertCurrency(String currencyCode, String currencyName)* eingeführt wurde. Diese wird in den einzelnen Methoden für die jeweilige Währung aufgerufen.

```
1  private String convertCurrency(String currencyCode, String currencyName) {
2      JsonObject jsonObject = Requests.getRequest(currencyUrl);
```

```
        assert jsonObject != null;
        BigDecimal exchangeRate = BigDecimal.valueOf(getCurrencyFromJson(jsonObject, currencyCode));
        BigDecimal balance;

        try {
            balance = bankAccountService.getBankAccount(selectedBankAccount.getAccountId()).orElseThrow().
        getBalance();
        } catch (DatabaseException e) {
            return localization.getMessage("database_error", e.toString());
        }

        return localization.getMessage("convert_balance", currencyName, (balance.multiply(exchangeRate)));
    }

    private String convertToBitcoin()
    {
        return convertCurrency("btc", "Bitcoin");
    }

    private String convertToGermanDeutscheMark()
    {
        return convertCurrency("dem", "Deutsche Mark");
    }

    private String convertToSwissFranc()
    {
        String currency = localization.getCurrentLanguageCode().equals("de") ? "Schweizer Franken" : "Swiss
        Franc";
        return convertCurrency("chf", currency);
    }

    private String convertToDogecoin()
    {
        return convertCurrency("doge", "Dogecoin");
    }

    private String convertToEthereum()
    {
        return convertCurrency("eth", "Ethereum");
    }

    private String convertToHongKongDollar()
    {
        return convertCurrency("hkd", "Hong Kong Dollar");
    }

    private String convertToJamaicanDollar()
    {
        String currency = localization.getCurrentLanguageCode().equals("de") ? "Jamaikanische Dollar" : "
        Jamaican Dollar";
        return convertCurrency("jmd", currency);
    }

    private String convertToNorthKoreanWon()
    {
        String currency = localization.getCurrentLanguageCode().equals("de") ? "Nordkoreanische Won" : "
        North Korean Won";
        return convertCurrency("kpw", currency);
    }

    private String convertToRussianRuble()
    {
        String currency = localization.getCurrentLanguageCode().equals("de") ? "Russischer Rubel" : "Russian
          Ruble";
        return convertCurrency("rub", currency);
    }

    private String convertToUsDollar()
    {
        return convertCurrency("usd", "US Dollar");
    }
```

Code Smell 2: Switch-Statement *main/Commit c46c962*

Vorher

```java
public CommandController()
{
}

public String handleInput(String input)
{

    String output;

    switch (input) {
        case "/exit":
            System.exit(0);
            output = "Goodbye User";
            break;
        case "/help":
            HelpCommandService helpCommandService = new HelpCommandService();
            output = helpCommandService.getAllCommands();
            break;
        default:
            output = "Something went wrong :( ";
            break;
    }

    return output;
}
```

Lösung

Erklären was gemacht wurde und warum es gemacht wurde. Warum ist da jetzt gefühlt 7x mehr Code, was bringt das?

```java
// CommandController.java
private final Map<Commands, CommandService> commandMap;

public CommandController()
{

    commandMap = new HashMap<>();

    commandMap.put(Commands.EXIT, new ExitCommandService());
    commandMap.put(Commands.HELP, new HelpCommandService());
    commandMap.put(Commands.BILANZIUS, new BilanziusCommandService());

}

public String handleInput(String input)
{

    String[] parts = input.split(" ", 2);
    String commandStr = parts[0];
    String[] arguments = parts.length > 1 ? parts[1].split(" ") : new String[0];

    Commands command = Commands.fromString(commandStr);
    CommandService commandService = commandMap.get(command);

    if (commandService != null) {
        return commandService.execute(arguments);
    }

    return "Unknown command :( . Type /help for a list of commands.";
}

// Commands.java
public enum Commands
{
    EXIT("/exit", "Exit the application", null),
    HELP("/help", "Show all commands", null),
    BILANZIUS("/bilanzius", "Get information about the application", BilanziusCommandArguments.
     getAllArguments());

```

```
    // Hier werden die einzelnen Befehle hinzugefügt

    private final String command;
    private final String description;
    private final String arguments;

    Commands(String command, String description, String arguments) {
        this.command = command;
        this.description = description;
        this.arguments = arguments;
    }

    public String getCommand() {
        return command;
    }

    public String getDescription() {
        return description;
    }

    public String getArguments() {
        return arguments;
    }

    public static String getAllCommands() {

        return Arrays.stream(Commands.values()).map(
                c -> c.getCommand()
                        + " - " +
                        c.getDescription()
                        +
                        (
                                c.getArguments()
                                        != null ?
                                        " | " + (c.getArguments())
                                        : ""
                        )
                ).reduce(
                        (a, b) -> a + "\n" + b
                ).orElse("");
    }

    public static Commands fromString(String command) {
        for (Commands c : Commands.values()) {
            if (c.command.equals(command)) {
                return c;
            }
        }
        return null;
    }
}

public interface CommandService
{
    String execute(String[] arguments);
}
```

7.2 2 Refactorings (6P)

[2 unterschiedliche Refactorings aus der Vorlesung jeweils benennen, anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen – die Refactorings dürfen sich nicht mit den Beispielen der Code überschneiden]

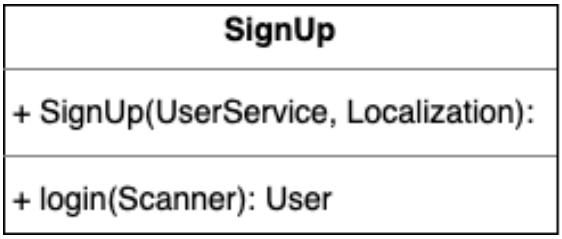
Refactoring 1: Replace Error Code with Exceptions *main/Commit 2cc1fce*

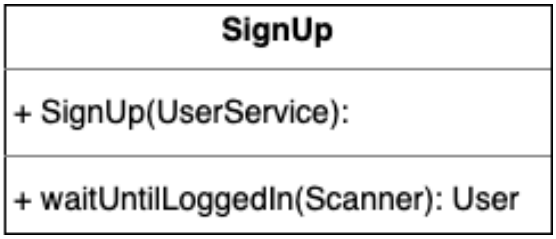
Alle Methoden von UserService werfen eine DatabaseException die nicht abgefangen wurde. Daher wurden try-catch-Block um die jeweiligen Methoden hinzugefügt.

UML? Vergleich zu vorher, bzw. was war der Error Code?

Refactoring 2: Rename Methods *main/Commit d3be454*

Die Methode login() in der Klasse SignUp wurde zu waitUntilLoggedIn() umbennant da in der Methode gerwartet wird bis der Benutzter eingeloggt ist und daraufhin dann entscheidet.





8 Entwurfsmuster (8P)

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils benennen, sinnvoll einsetzen, begründen und UML-Diagramm]

8.1 Entwurfsmuster : Singleton (Erzeugungsmuster) (4P)

8.2 Entwurfsmuster: Strategy (Verhaltensmuster) (4P)