

DHBW Mannheim

Programmieren in Java

TINF22CS2



Java-Banking-App -Dokumentation-

Teilnehmer:

Felipe Hidalgo Söhnlein (4612565),

Erjon Sejdiu (1358477),

Tobias Ruhland (8021463),

Merle Simon (6589957),

Christopher Frey(1881082),

Inhaltsangabe:

Java-Banking-App -Dokumentation-	1
1. Banking App: Die Idee dahinter	3
2. Installations Guide	4
3. Bedienungsanleitung	5
3.1 Perspektive des Bänkers	5
3.2 Perspektive des Kunden	6
4. Technische Dokumentation:	9
4.1 Interfaces	10
4.2 Beschreibung der Klassen	10
4.2.1 BankAccount	10
4.2.2 Customer	11
4.2.3 Login	13
4.2.4 Person	14
4.2.5 Settings	14
4.2.6 Register	15
4.2.7 GiroAccount	16
4.2.8 SavingAccount	17
4.2.9 NewKonto	18
4.2.10 DBProvider	18
4.2.11 UserInterface	20
4.2.12 Transfer	20
4.2.13 Customer	21
4.2.14 BankerUI	22
4.3 Datenbanken	23
5. Klassendiagramm	24
6. Ausblick	25
7. Initale Zugangsdaten zum Testen	25
8. Abbildungsverzeichnis	25

1. Banking App: Die Idee dahinter

Als Team haben wir uns zunächst mal an unseren eigenen Banking Apps auf dem Handy orientiert. Wir haben herausgearbeitet, was die grundlegenden Funktionen einer solchen App sein sollten und aufgeschrieben. Zu diesen gehören: individuelle Konten mit An- und Abmeldung und Bearbeitungsfunktion, spezielle Arten von Konten zugehörig zum Hauptkonto (Giro, Festgeld, Kreditkarte, Depot), Funktionen zum anzeigen des Kontostandes und des Dispos, Funktionen zum überweisen und ein System zum Beantragen von Sonderrechten (Höheres Dispo oder Kreditkartenlimit). Außerdem gibt es zusätzlich Funktionen auf Seiten des Bänkers der die Administration der Konten übernimmt. Dieser muss Konten manipulieren und löschen können, Anfragen genehmigen und ablehnen können, Sonderkonten hinzufügen oder löschen können, Dispo und Kreditkartenlimit erhöhen oder senken können. Dies haben wir zum Beispiel genommen und haben auf diesen Funktionalitäten unsere Banking App aufgebaut.

Die Logik der Funktionalitäten des Kunden sieht wie folgt aus: Der Kunde kann sich nach dem Starten der Anwendung registrieren oder auch anmelden. Nach erfolgreicher Authentisierung ist der Kunde dazu berechtigt, auf seine Konten zuzugreifen. Neben den Girokonten hat der Kunde Zugriff auf spezielle Kontotypen wie Festgeldkonten, die alle individuelle Merkmale und Funktionen haben. Das Festgeldkonto zeigt den aktuellen Betrag an, der dort angelegt ist. Mit dem Girokonto hat der Kunde die Möglichkeit, seinen aktuellen Saldo und sein verfügbares Dispolimit einzusehen.

Für Überweisungen bietet die App eine Funktion, bei der der Kunde die nötigen Details wie Empfänger, IBAN, Betrag und Verwendungszweck eingeben kann. Sollte der Kunde ein höheres Dispolimit benötigen, kann er dies per Kontakt via Mail oder Telefon beantragen. Außerdem werden alle Zahlungsein- und -ausgänge in einer Historie angezeigt. Die Logik der Funktionen auf der Seite des Bankangestellten beinhaltet das Manipulieren von Konten, das Genehmigen und Ablehnen von Anträgen, das Hinzufügen oder Löschen von Sonderkonten und das Erhöhen oder Senken von Dispolimits. Außerdem ist eine Suche mit Filter enthalten, die einem Kunden mit z.B. negativem Kontostand anzeigen kann. Dies ermöglicht eine umfassende Kontoverwaltung und -überwachung. Zusammenfassend lässt sich sagen, dass unsere Banking App die grundlegenden Funktionen von Banken aufbaut und diese Funktionen in einer benutzerfreundlichen und effizienten Weise zur Verfügung stellt. Es wurde darauf geachtet, dass sowohl die Kunden als auch die Bankmitarbeiter eine intuitive und effektive Benutzererfahrung haben.

Anmerkung: Die Initialen Zugangsdaten sind unter Punkt 6: Initiale Zugangsdaten zum Testen zu finden.

2. Installations Guide

Voraussetzungen:

- JRE Version 20 / JDK Version 20
- Zugriff auf die Datei BankingAppTeam4.zip im Moodle Verzeichnis
- Terminal App

Installationsschritte

1. Laden Sie die Zip-Datei Banking App Team 4.zip aus dem Moodle-Verzeichnis herunter.
2. Extrahieren Sie die Zip-Datei in einem Verzeichnis Ihrer Wahl auf Ihrem Computer. Die extrahierte Zip-Datei sollte alle benötigten Jar-Files für die Ausführung der Applikation inklusive Datenbank enthalten. Es ist wichtig, dass die Datenbank und die Files im Verzeichnis sind, um eine korrekte Ausführung mit den Testdaten zu gewährleisten. Ansonsten wird eine neue erstellt, die keine Daten und nur die zwei standard Bänker enthält
3. Öffnen Sie eine Konsole oder ein Terminal und navigieren Sie zu dem Verzeichnis, in das Sie die Jar-Files extrahiert haben.
4. Führen Sie das Programm aus, indem Sie den folgenden Befehl in der Konsole eingeben: `java -jar BankingAppTeam4.jar`. Stellen Sie sicher, dass Sie den Namen der Jar-Datei entsprechend anpassen, wenn die Jar-Datei einen anderen Namen hat.

Problemlösungen:

Falls bei der Ausführung der Jar-Datei Probleme auftreten, stellen Sie sicher, dass Sie über die richtige Java-Version verfügen und dass Ihre Systemvariablen korrekt gesetzt sind. Sie sollten in der Lage sein, die "java -Version" in Ihrer Konsole auszuführen und eine gültige Java-Version angezeigt zu bekommen. Falls das nicht der Fall ist, müssen Sie möglicherweise Ihr Java JDK installieren oder aktualisieren.

Hinweis

Die App enthält jeglichen Source Code und ist komplett lauffähig als Projekt. Alle benötigten Dateien sind in den Jar-Files enthalten. Für weitere Informationen oder bei spezifischen Problemen, wenden Sie sich bitte an uns oder sehen Sie in der spezifischen Dokumentation nach.

3. Bedienungsanleitung

3.1 Perspektive des Bänkers

Die Anwendung bietet einen allgemeinen für alle Zugänglichen Login, wo sich der Bänker auch anmelden kann.

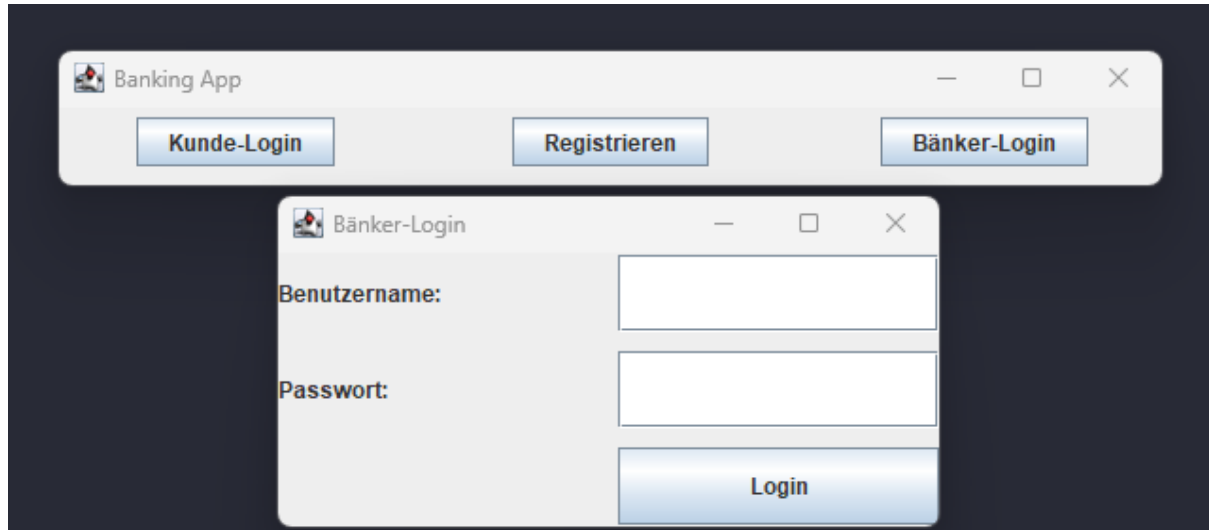


Abb. 1: Bänker-Login

Dadurch, dass der Bänker ein hoch privilegierter Account ist, haben wir uns ein Schutzkonzept überlegt, damit nicht jeder einen solchen Account erstellen kann. Dieser muss nämlich beantragt und durch die Systemadministratoren (Team 4) bereitgestellt werden.

Im Suchfeld kann nach Namen gesucht werden. Führt man das Dropdown-Menü links davon aus, stehen einem die Kategorien Alle, Ausstehend für noch zu genehmigende Anträge und Negativ für alle Konten mit negativem Kontostand zum Filtern zur Verfügung.

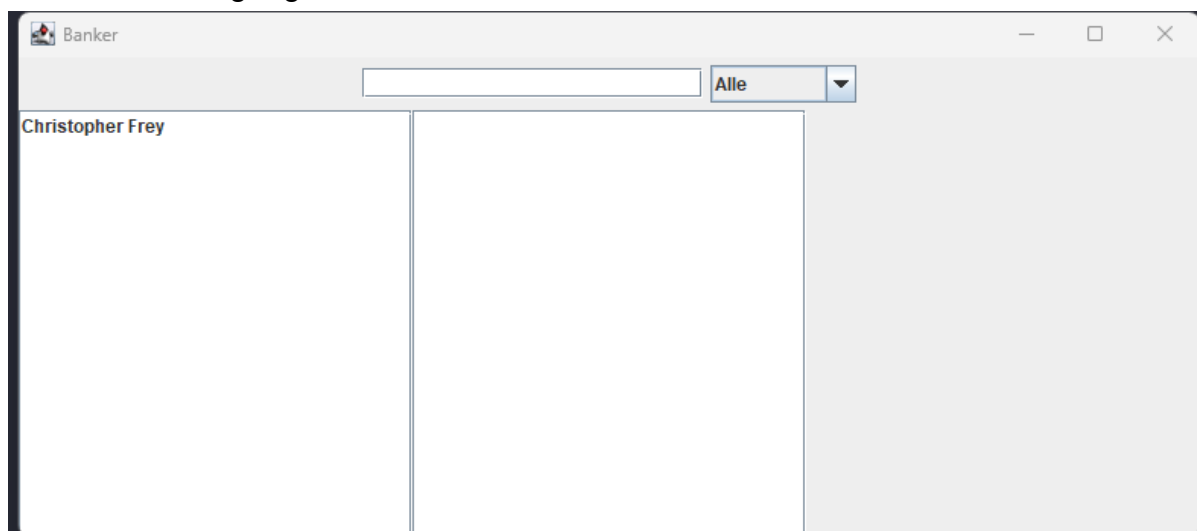


Abb. 2: Bänker-Interface

Durch das Drücken auf einen Namen erhält man als Bänker Zugriff auf die Konten eines Kunden. Man kann die Konten verändern, indem man das Limit durch den Button “Limit setzen” ändert.

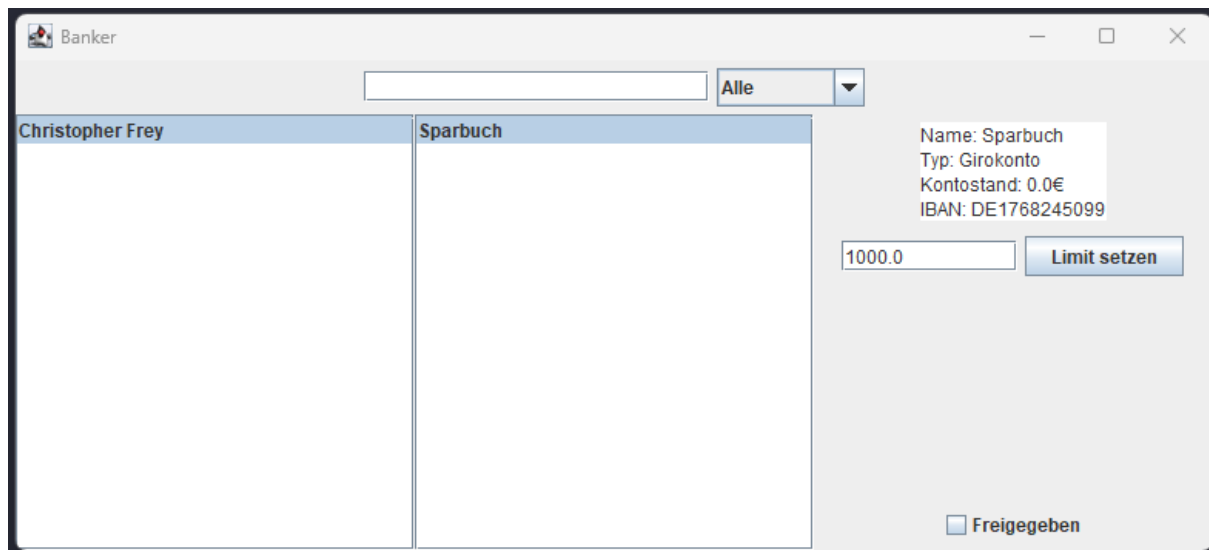


Abb. 3: Bänker Account Manipulation

3.2 Perspektive des Kunden

Der Kunde hat die Möglichkeit, sich normal zu registrieren und sich anzumelden. Die Registrierung läuft über den Registrieren Button im Haupt Login. Für bestehende Kunden

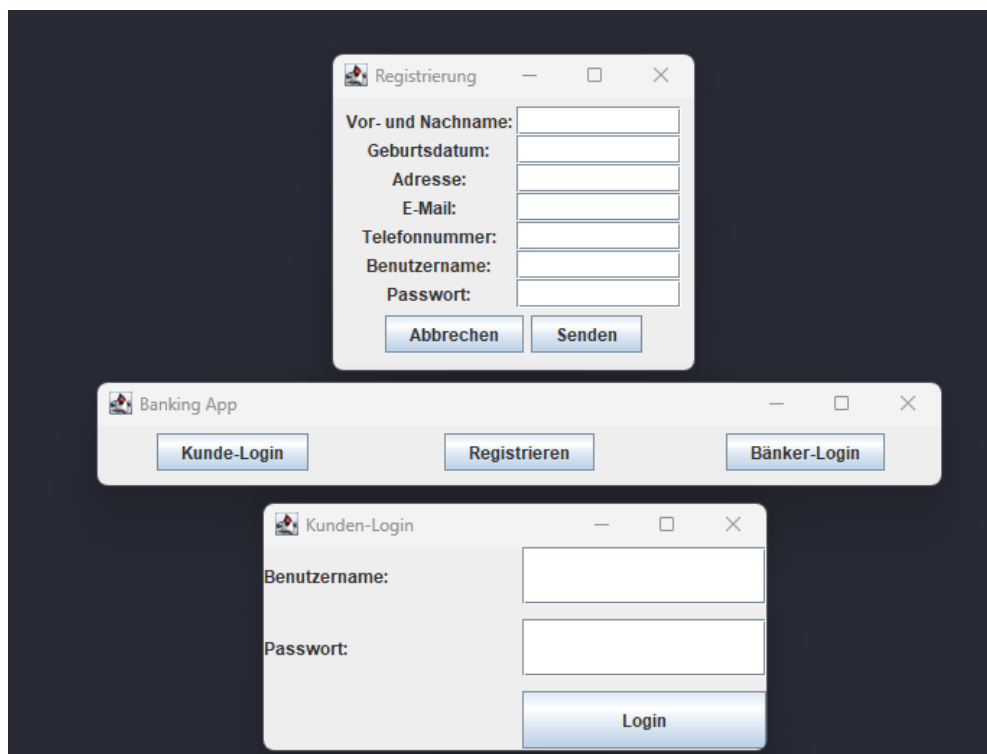


Abb. 4: Kunden Login

Wenn man einmal Angemeldet ist, sieht der Kunde folgendes vor sich.



Abb. 5: Kunden-Interface

Um nun ein Konto erstellen zu können, muss man auf "Neues Konto erstellen" klicken. Was man dann vor sich sieht, ist folgendes.

The screenshot shows a window titled "Neues Konto erstellen". It features a light gray background. At the top left, there is a hint: "Hinweis: Neue Kontoanträge müssen genehmigt wer...". Below this, there are three input fields: "Kontoname", "Limit", and "Girokonto" (which is a dropdown menu currently showing "Girokonto"). At the bottom of the form, there are two large buttons: "Abbrechen" on the left and "Absenden" on the right.

Abb. 6: Neues Konto erstellen

Hier kann man auswählen, was für eine Kontoart man will und welchen Namen und welches Limit man will. Dieser Antrag muss nach dem Absenden vom Banker genehmigt werden. Im Menü taucht nach erfolgter Genehmigung folgendes auf.

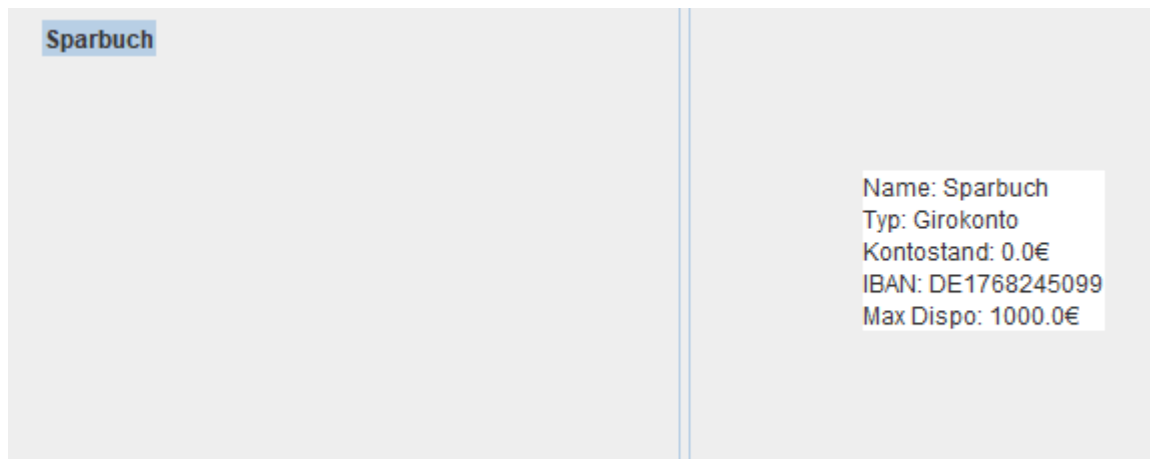


Abb 7: Konto in der Übersicht

Durch das Klicken auf das Konto erhält man auf der rechten Seite eine Übersicht von den Kontodetails. Man kann natürlich auch Konten löschen indem man dieses auswählt und auf den “Konto Löschen” Button drückt.

Des Weiteren kann man Überweisungen tätigen. Man drückt auf das Konto, mit dem man die Überweisung ausführen will. Bei Erstellung von einem Konto wird die IBAN automatisch erstellt. Beim Empfänger im folgenden Fenster gibt man diese mit anderen Informationen an.

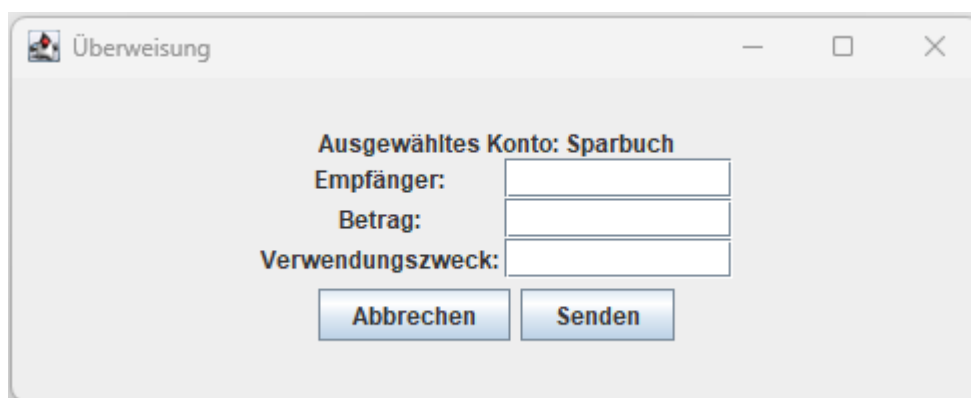


Abb 8: Überweisungs Fenster

Jeden Zahlungsein- und Ausgang kann man sich durch das Drücken des Transferhistorie-Buttons anzeigen lassen. Eine Transferhistorie sieht wie folgt aus:

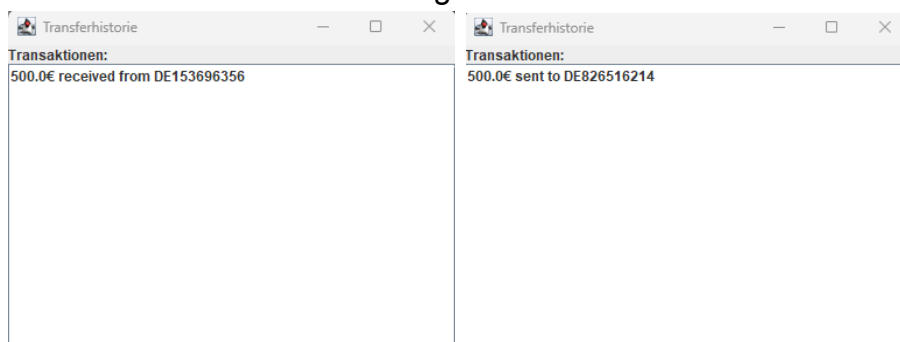
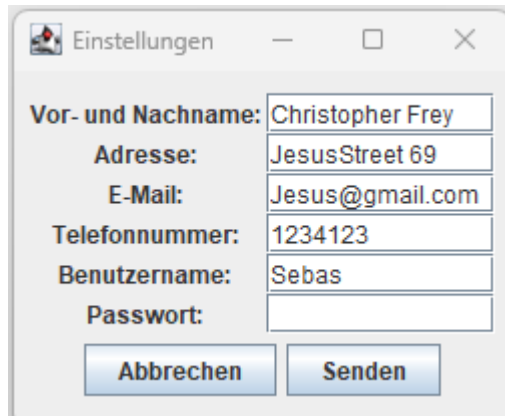


Abb 9: Transaktionsfenster erhalten (links) und gesendet (rechts)

Zu guter Letzt kann man die bei der Registrierung festgelegten Daten unter dem Punkt Einstellungen ändern. Die Einstellungen sehen wie folgt aus.



Vor- und Nachname:	Christopher Frey
Adresse:	JesusStreet 69
E-Mail:	Jesus@gmail.com
Telefonnummer:	1234123
Benutzername:	Sebas
Passwort:	

Abbrechen Senden

Abb 10: Einstellungen

4. Technische Dokumentation:

Im Folgenden wird der gesamte technische Rahmen und der Aufbau vollständig und präzise dargeboten, um so ein tieferes Verständnis bezüglich der Anwendung zu schaffen.

4.1 Interfaces

Die Interfaces basieren auf Java JFrame. JFrame ist eine grundlegende Klasse in der Java-Swing-Bibliothek zur Erstellung von grafischen Benutzeroberflächen (GUIs). Als Fenster auf dem Bildschirm kann ein JFrame verschiedene Swing-Komponenten wie Buttons, Labels und Textfelder enthalten. Bei der Erstellung eines JFrame-Objekts werden in der Regel Einstellungen wie Titel, Größe und Aktion beim Schließen des Fensters konfiguriert. Komponenten werden nicht direkt zum JFrame hinzugefügt, sondern zu seinem Inhaltsbereich, dem sogenannten ContentPane. Nachdem alle Komponenten hinzugefügt wurden, wird das JFrame-Objekt sichtbar gemacht. Die meisten Interfaces in diesem Projekt basieren auf einem GridLayout für eine bessere Strukturierung. Innerhalb des Gridlayouts sind in diesem Projekt mehrere Arten von Unterlayouts zu finden, darunter: Flowlayouts (alle Elemente von links nach rechts) und Borderlayouts (Mit Top-, East-, South-, West- Border).

4.2 Beschreibung der Klassen

4.2.1 BankAccount

Die Klasse BankAccount ist eine abstrakte Klasse, die als Grundlage für verschiedene Arten von Bankkonten dient. Sie implementiert das Serializable-Interface, um die Objekte dieser Klasse serialisierbar zu machen.

Attribute:

- **AccountNumber (String):** Die Kontonummer des Bankkontos.
- **AccountBalance (double):** Der Kontostand des Bankkontos.
- **AccountName (String):** Der Name des Bankkontos.
- **Customer (Customer):** Der Kunde, dem das Bankkonto gehört.
- **Approved (boolean):** Ein Flag, das angibt, ob das Bankkonto genehmigt ist

Konstruktor:

- **BankAccount(String accountNumber, double bankBalance, Customer customer, String accountName):** Ein Konstruktor, der eine Kontonummer, einen Anfangskontostand, einen Kunden und einen Kontonamen akzeptiert und ein Bank-Account-Objekt erstellt.

Methoden:

- **withdraw(double amount):** Eine abstrakte Methode, die den Abhebevorgang für das Bankkonto abgewickelt. Diese Methode sollte in den abgeleiteten Klassen implementiert werden.
- **deposit(double amount):** Eine Methode, die den Einzahlungsvorgang für das Bankkonto abgewickelt. Der angegebene Betrag wird zum aktuellen Kontostand addiert.
- **getAccountNumber():** Gibt die Kontonummer des Bankkontos zurück.
- **getBankBalance():** Gibt den aktuellen Kontostand des Bankkontos zurück.
- **setAccountBalance(double bankBalance):** Setzt den Kontostand des Bankkontos auf den angegebenen Wert.
- **getAccountName():** Gibt den Namen des Bankkontos zurück.
- **setApproved(boolean bool):** Setzt den Genehmigungsstatus des Bankkontos auf den angegebenen Wert.
- **getApproved():** Gibt den Genehmigungsstatus des Bankkontos zurück.
- **toString():** Gibt den Namen des Bankkontos als Zeichenkette zurück.

Diese abstrakte Klasse dient als gemeinsame Basis für verschiedene Arten von Bankkonten. Sie enthält grundlegende Methoden und Attribute, die in den abgeleiteten Klassen erweitert und spezialisiert werden können. Die withdraw-Methode muss in den abgeleiteten Klassen implementiert werden, um den spezifischen Abhebevorgang für das jeweilige Bankkonto durchzuführen.

4.2.2 Customer

Die Klasse Customer erweitert die abstrakte Klasse Person und repräsentiert einen Kunden in der Bankanwendung. Sie implementiert das Serializable-Interface, um die Objekte dieser Klasse serialisierbar zu machen.

Attribute:

- **login (String):** Der Benutzername des Kunden für den Login.
- **password (String):** Das Passwort des Kunden für den Login.
- **konten (List<BankAccount>):** Eine Liste der Bankkonten, die dem Kunden gehören.

Konstruktor:

- **Customer(String name, Date birthDate, String address, String email, String phoneNumber, String login, String password):** Ein Konstruktor, der den Namen, das Geburtsdatum, die Adresse, die E-Mail-Adresse, die Telefonnummer, den Benutzernamen und das Passwort des Kunden akzeptiert und ein Customer-Objekt erstellt. Es initialisiert auch die Liste der Bankkonten.

Methoden:

- **login(String login, String password):** Eine Methode, mit der der Kunde sich in das Bankkonto einloggen kann. Sie überprüft die eingegebenen Login-Daten und gibt entsprechende Meldungen aus.
- **logout():** Eine Methode, mit der der Kunde sich aus dem Bankkonto ausloggen kann.
- **allDebitAccounts():** Eine Methode, die eine Liste aller Girokonten des Kunden zurückgibt. Sie durchläuft die Liste der Bankkonten und fügt diejenigen Konten, die Instanzen von GiroAccount sind, zur Rückgabeliste hinzu.
- **allSavingAccounts():** Eine Methode, die eine Liste aller Sparkonten des Kunden zurückgibt. Sie durchläuft die Liste der Bankkonten und fügt diejenigen Konten, die Instanzen von SavingAccount sind, zur Rückgabeliste hinzu.
- **createAccount(BankAccount bankAccount):** Eine Methode, um dem Kunden ein neues Bankkonto hinzuzufügen. Das übergebene BankAccount-Objekt wird der Liste der Bankkonten hinzugefügt und die Daten werden in der Datenbank gespeichert.
- **deleteAccount(BankAccount account):** Eine Methode, um ein Bankkonto des Kunden zu löschen. Das übergebene BankAccount-Objekt wird aus der Liste der Bankkonten entfernt.
- **getAccount(String kontoNummer):** Eine Methode, die ein Bankkonto des Kunden anhand der Kontonummer zurückgibt. Sie durchläuft die Liste der Bankkonten und sucht nach einem Konto mit der angegebenen Kontonummer.
- **isAccountCoverd(String kontoNummer, double betrag):** Eine Methode, die überprüft, ob das Bankkonto mit der angegebenen Kontonummer ausreichend Guthaben für den angegebenen Betrag hat. Sie durchläuft die

Liste der Bankkonten und sucht nach einem Konto mit der entsprechenden Kontonummer. Wenn ein solches Konto gefunden wird, wird überprüft, ob der Kontostand größer oder gleich dem angegebenen Betrag ist.

- **getPassword():** Gibt das Passwort des Kunden zurück.
- **setPassword(String password):** Setzt das Passwort des Kunden auf den angegebenen Wert.
- **setUsername(String username):** Setzt den Benutzernamen des Kunden auf den angegebenen Wert.
- **getUsername():** Gibt den Benutzernamen des Kunden zurück.
- **getLogin():** Gibt den Benutzernamen des Kunden zurück.

Diese Klasse repräsentiert einen Kunden mit seinen Login-Daten und einer Liste von Bankkonten. Sie bietet Methoden zum Einloggen, Ausloggen, Verwalten der Bankkonten und zur Überprüfung von Kontoinformationen.

4.2.3 Login

Die Klasse Login erweitert die JFrame-Klasse und stellt das Anmeldefenster für die Banking-App dar. Es ermöglicht Kunden und Bankern, sich einzuloggen oder sich zu registrieren.

Attribute:

- **customerLoginButton (JButton):** Ein Button für den Kunden-Login.
- **bankerLoginButton (JButton):** Ein Button für den Banker-Login.
- **registerButton (JButton):** Ein Button für die Registrierung.

Konstruktor:

- **Login():** Ein Konstruktor, der das Anmeldefenster initialisiert. Es setzt den Titel, die Größe, das Layout und erstellt und fügt die Komponenten hinzu. Es setzt auch die ActionListener für die Buttons.

Methoden:

- **createAndAddComponents():** Eine Methode, die die Komponenten für das Anmeldefenster erstellt und hinzufügt. Sie erstellt Panels für den Kunden-Login, die Registrierung und den Banker-Login und fügt sie dem Frame hinzu. Sie setzt auch ActionListener für die Buttons.
- **createBankerLoginWindow():** Eine Methode, die ein neues Fenster für den Banker-Login erstellt. Sie zeigt ein Textfeld für den Benutzernamen, ein Passwort Feld für das Passwort und einen Login-Button an. Sie überprüft die eingegebenen Login-Daten und öffnet das BankerUI-Fenster, wenn die Daten korrekt sind.
- **createCustomerLoginWindow():** Eine Methode, die ein neues Fenster für den Kunden-Login erstellt. Sie zeigt ein Textfeld für den Benutzernamen, ein

Passwort Feld für das Passwort und einen Login-Button an. Sie überprüft die eingegebenen Login-Daten und öffnet das `UserInterfaceTest`-Fenster, wenn die Daten korrekt sind.

- **`authenticateUser(String username, String password)`**: Eine statische Methode, die die eingegebenen Login-Daten für den Kunden überprüft, indem sie die `customerLogin`-Methode des `DBProvider` aufruft. Sie gibt `true` zurück, wenn die Authentifizierung erfolgreich ist, andernfalls `false`.
- **`authenticateBanker(String username, String password)`**: Eine statische Methode, die die eingegebenen Login-Daten für den Banker überprüft, indem sie die `bankerLogin`-Methode des `DBProvider` aufruft. Sie gibt `true` zurück, wenn die Authentifizierung erfolgreich ist, andernfalls `false`.

Diese Klasse erstellt das Anmeldefenster für die Banking-App und ermöglicht Kunden und Bankern den Login. Es enthält auch die Logik zur Überprüfung der Login-Daten mithilfe des `DBProvider`.

4.2.4 Person

Die Klasse `Person` ist eine einfache Datenklasse, die Informationen über eine Person enthält. Sie implementiert das `Serializable`-Interface, um die Objekte serialisierbar zu machen.

Attribute:

- **`name (String)`**: Der Name der Person.
- **`birthDate (Date)`**: Das Geburtsdatum der Person.
- **`address (String)`**: Die Adresse der Person.
- **`email (String)`**: Die E-Mail-Adresse der Person.
- **`phoneNumber (String)`**: Die Telefonnummer der Person.

Getter und Setter:

- **`getName()`**: Gibt den Namen der Person zurück.
- **`setName(String name)`**: Setzt den Namen der Person.
- **`getBirthDate()`**: Gibt das Geburtsdatum der Person zurück.
- **`setBirthDate(Date birthDate)`**: Setzt das Geburtsdatum der Person.
- **`getAddress()`**: Gibt die Adresse der Person zurück.
- **`setAddress(String address)`**: Setzt die Adresse der Person.
- **`getEmail()`**: Gibt die E-Mail-Adresse der Person zurück.
- **`setEmail(String email)`**: Setzt die E-Mail-Adresse der Person.
- **`getPhoneNumber()`**: Gibt die Telefonnummer der Person zurück.
- **`setPhoneNumber(String phoneNumber)`**: Setzt die Telefonnummer der Person.

Diese Klasse dient als Grundlage für die Klassen, die von einer Person abgeleitet sind, z. B. die Customer-Klasse. Sie speichert grundlegende Informationen über eine Person wie Name, Geburtsdatum, Adresse, E-Mail-Adresse und Telefonnummer. Durch die Implementierung des Serializable-Interface kann ein Objekt der Person-Klasse in eine serialisierbare Form umgewandelt und gespeichert werden.

4.2.5 Settings

Die Klasse Settings ist eine GUI-Klasse, die ein Fenster für die Einstellungen eines Kunden in der Banking-App erstellt. Das Fenster enthält verschiedene Eingabefelder, in denen der Kunde seine persönlichen Informationen bearbeiten kann.

Konstruktor:

- **Settings():** Der Konstruktor initialisiert das Einstellungsfenster. Es wird ein Titel gesetzt, das Layout wird auf GridBagLayout festgelegt und die Größe des Fensters wird festgelegt. Die Position des Fensters wird auf die Mitte des Bildschirms gesetzt und die Schließ-Operation wird auf DISPOSE_ON_CLOSE gesetzt.

Methoden:

- **createAndAddComponents():** Diese Methode erstellt die Komponenten für das Einstellungsfenster und fügt sie dem GridBagLayout hinzu. Sie enthält Labels und Textfelder für Vor- und Nachname, Adresse, E-Mail, Telefonnummer, Benutzername und Passwort. Die Textfelder werden mit den aktuellen Werten des aktuellen Kunden aus der Datenbank initialisiert.
- **createButtonPanel():** Diese Methode erstellt ein JPanel, das die Abbrechen- und Senden-Buttons enthält. Sie fügt die Buttons zum Panel hinzu und gibt das Panel zurück.
- **submitButtonActionPerformed(ActionEvent e):** Diese Methode wird aufgerufen, wenn der Senden-Button geklickt wird. Sie liest die eingegebenen Werte aus den Textfeldern aus und aktualisiert die entsprechenden Eigenschaften des aktuellen Kunden in der Datenbank. Falls ein Feld leer gelassen wird oder der eingegebene Wert mit dem aktuellen Wert übereinstimmt, wird keine Aktualisierung durchgeführt. Nach der Aktualisierung werden die Änderungen gespeichert und eine Erfolgsmeldung angezeigt.

Die Klasse Settings ermöglicht es dem Kunden, seine persönlichen Informationen wie Vor- und Nachname, Adresse, E-Mail, Telefonnummer, Benutzername und Passwort zu bearbeiten. Wenn der Kunde Änderungen vornimmt und den Senden-Button klickt, werden die geänderten Informationen in der Datenbank aktualisiert und gespeichert.

4.2.6 Register

Die Klasse Register ist eine GUI-Klasse, die ein Registrierungsfenster für neue Kunden in der Banking-App erstellt. Das Fenster enthält verschiedene Eingabefelder, in denen der Kunde seine persönlichen Informationen für die Registrierung eingeben kann.

Konstruktor:

- **Register():** Der Konstruktor initialisiert das Registrierungsfenster. Es wird ein Titel gesetzt, das Layout wird auf GridBagLayout festgelegt und die Größe des Fensters wird festgelegt. Die Position des Fensters wird auf die Mitte des Bildschirms gesetzt und die Schließ-Operation wird auf DISPOSE_ON_CLOSE gesetzt.

Methoden:

- **createAndAddComponents():** Diese Methode erstellt die Komponenten für das Registrierungsfenster und fügt sie dem GridBagLayout hinzu. Sie enthält Labels und Textfelder für Vor- und Nachname, Geburtsdatum, Adresse, E-Mail, Telefonnummer, Benutzername und Passwort.
- **createButtonPanel():** Diese Methode erstellt ein JPanel, das die Abbrechen- und Senden-Buttons enthält. Sie fügt die Buttons zum Panel hinzu und gibt das Panel zurück.
- **submitButtonActionPerformed(ActionEvent e):** Diese Methode wird aufgerufen, wenn der Senden-Button geklickt wird. Sie liest die eingegebenen Werte aus den Textfeldern aus, validiert das Geburtsdatum und erstellt ein neues Customer-Objekt mit den eingegebenen Informationen. Das Customer-Objekt wird dann über den DBProvider zur Registrierung an die Datenbank übergeben. Eine Erfolgsmeldung wird angezeigt und das Registrierungsfenster wird geschlossen.

Die Klasse Register ermöglicht es neuen Kunden, sich in der Banking-App zu registrieren, indem sie ihre persönlichen Informationen eingeben. Nach der Registrierung werden die Informationen in der Datenbank gespeichert und der Kunde erhält eine Erfolgsmeldung.

4.2.7 GiroAccount

Die Klasse GiroAccount erweitert die Klasse BankAccount und stellt ein Girokonto dar, das spezifische Eigenschaften und Funktionen für ein solches Konto enthält. Ein Girokonto ist ein Bankkonto, das es dem Kontoinhaber ermöglicht, Zahlungen vorzunehmen, Geld abzuheben und über einen Dispositionskredit ("Dispo") zu verfügen.

Konstruktor:

- **GiroAccount(String accountnumber, double balance, Customer customer, String accountname, double dispo):** Dieser Konstruktor erstellt ein GiroAccount-Objekt mit den übergebenen Informationen. Er ruft den Konstruktor der Superklasse BankAccount auf und setzt zusätzlich den Wert für den Dispositionskredit (dispo) des Kontos.

Methoden:

- **withdraw(double amount):** Diese Methode überschreibt die Methode withdraw der Superklasse. Sie ermöglicht es dem Kunden, Geld vom Girokonto abzuheben. Zunächst wird überprüft, ob der angegebene Betrag das verfügbare Guthaben und den Dispositionskredit überschreitet. Wenn ja, wird eine Fehlermeldung angezeigt und false zurückgegeben. Andernfalls wird der angegebene Betrag vom Kontostand abgezogen. Wenn der Kontostand negativ wird, wird der Betrag aus dem Dispositionskredit verwendet. Eine Information über die Nutzung des Dispositionskredits wird angezeigt, und true wird zurückgegeben.
- **getDispo():** Diese Methode gibt den aktuellen Wert des Dispositionskredits (Dispo) des Kontos zurück.

Die Klasse GiroAccount repräsentiert ein Girokonto in der Banking-App. Es ermöglicht Kunden, Geld abzuheben und den Dispositionskredit zu nutzen, wenn das Guthaben nicht ausreicht. Der Dispositionskredit wird automatisch genutzt, wenn der Kontostand negativ ist. Die Klasse bietet Funktionen zur Überprüfung und Verwaltung des Guthabens und des Dispositionskredits des Kontos.

4.2.8 SavingAccount

Die Klasse SavingAccount erweitert die Klasse BankAccount und repräsentiert ein Sparkonto in der Banking-App. Ein Sparkonto ist ein Bankkonto, das es dem Kontoinhaber ermöglicht, Geld zu sparen und Zinsen auf das Guthaben zu verdienen. Im Gegensatz zum Girokonto ermöglicht ein Sparkonto normalerweise keine direkten Auszahlungen oder Überweisungen.

Konstruktor:

- **SavingAccount(String accountNumber, double bankBalance, Customer customer, String accountName):** Dieser Konstruktor erstellt ein SavingAccount-Objekt mit den übergebenen Informationen. Er ruft den Konstruktor der Superklasse BankAccount auf und setzt die entsprechenden Werte für Kontonummer, Kontostand, Kundenobjekt und Kontoname.

Methoden:

- **withdraw(double amount):** Diese Methode überschreibt die Methode withdraw der Superklasse. Sie gibt eine Fehlermeldung aus, dass von einem Sparkonto kein Geld abgehoben werden kann, und gibt false zurück, um anzuzeigen, dass die Transaktion fehlgeschlagen ist.

Die Klasse SavingAccount repräsentiert ein Sparkonto in der Banking-App. Es bietet die Möglichkeit, Guthaben zu speichern und zu verwalten, jedoch keine direkten Zahlungen oder Überweisungen vorzunehmen. Die Klasse bietet eine Methode zur Fehlerbehandlung, um zu verhindern, dass Geld von einem Sparkonto abgehoben wird.

4.2.9 NewKonto

Die Klasse NewKonto erweitert JFrame und stellt ein Fenster dar, das es dem Benutzer ermöglicht, ein neues Konto zu erstellen. Das Fenster enthält Eingabefelder für den Kontonamen und das Dispolimit (nur für Girokonten). Der Benutzer kann auswählen, ob er ein Girokonto oder ein Festgeldkonto erstellen möchte. Nach dem Ausfüllen der Informationen und dem Klicken auf "Absenden" wird das neue Konto erstellt und dem aktuellen Kunden hinzugefügt.

Konstruktor:

- NewKonto(UserInterfaceTest parentComponent): Der Konstruktor erstellt ein NewKonto-Objekt und nimmt einen UserInterfaceTest-Parameter entgegen, der die übergeordnete Benutzeroberfläche Komponente darstellt. Das Fenster für das neue Konto wird initialisiert und angezeigt.

Methoden:

- Keine zusätzlichen öffentlichen Methoden vorhanden.

Die Klasse NewKonto bietet die folgenden Funktionen:

- Anzeigen eines Fensters zur Erstellung eines neuen Kontos.
- Auswahl zwischen Girokonto und Festgeldkonto.
- Eingabe des Kontonamens und des Dispolimits (nur für Girokonto).
- Validierung der eingegebenen Informationen.
- Erstellung des entsprechenden Konto-Typs (Girokonto oder Festgeldkonto) mit den angegebenen Informationen.
- Hinzufügen des neuen Kontos zum aktuellen Kunden.
- Speichern der Datenbank.
- Aktualisieren der übergeordneten Benutzeroberfläche, um die neuen Konten anzuzeigen.

Das NewKonto-Fenster bietet eine benutzerfreundliche Möglichkeit für Kunden, neue Konten zu erstellen und ermöglicht die einfache Verwaltung von verschiedenen Kontotypen in der Banking-App.

4.2.10 DBProvider

Die Klasse DBProvider stellt die Datenbank für das Banking System dar und bietet Funktionen zum Speichern, Laden und Verwalten der Kundendaten und Kontoinformationen.

Konstruktor:

- **DBProvider(DBContent content):** Der Konstruktor ist privat und wird nur einmal aufgerufen, um eine Instanz der Klasse zu erstellen. Er nimmt ein DBContent-Objekt entgegen, das die vorhandenen Datenbankinhalte repräsentiert.

Statische Methoden:

- **getInstance():** Diese Methode liefert eine einzige Instanz des DBProvider-Objekts zurück. Wenn bereits eine Instanz vorhanden ist, wird diese zurückgegeben. Andernfalls wird überprüft, ob eine Datenbankdatei existiert. Wenn keine Datei vorhanden ist, wird eine neue Datenbank erstellt. Andernfalls wird die Datenbank aus der Datei geladen und eine Instanz des DBProvider-Objekts erstellt.
- **serializeObject(Object object, String filePath):** Diese Methode serialisiert ein Objekt und speichert es in einer Datei mit dem angegebenen Dateipfad. Sie gibt true zurück, wenn der Vorgang erfolgreich war, andernfalls false.
- **deserializeObject(String filePath):** Diese Methode lädt und deserialisiert ein Objekt aus der angegebenen Datei und gibt es zurück. Bei einem Fehler wird null zurückgegeben.

Öffentliche Methoden:

- **register(Customer kunde):** Diese Methode registriert einen neuen Kunden, indem er ihn zur Datenbank hinzufügt und die Datenbank speichert. Sie gibt true zurück, wenn die Speicherung erfolgreich war, andernfalls false.
- **bankerLogin(String username, String password):** Diese Methode ermöglicht einem Bankangestellten das Einloggen in das System, indem der Benutzername und das Passwort überprüft werden. Wenn die Anmeldedaten korrekt sind, wird ein Banker-Objekt zurückgegeben, andernfalls null.
- **customerLogin(String uname, String password):** Diese Methode ermöglicht einem Kunden das Einloggen in das System, indem der Benutzername und das Passwort überprüft werden. Wenn die Anmeldedaten

korrekt sind, wird der entsprechende Customer zurückgegeben und als aktueller Benutzer festgelegt. Andernfalls wird null zurückgegeben.

- **getCurrentCustomer():** Diese Methode gibt den aktuellen eingeloggten Kunden zurück.
- **save():** Diese Methode speichert die Datenbank, indem sie den aktuellen Datenbankinhalt serialisiert und in einer Datei speichert. Sie gibt true zurück, wenn die Speicherung erfolgreich war, andernfalls false.
- **findBankAccountbyIban(String iban):** Diese Methode sucht anhand der IBAN nach einem Bankkonto in der Datenbank und gibt das entsprechende BankAccount-Objekt zurück. Wenn kein Konto mit der angegebenen IBAN gefunden wird, wird null zurückgegeben.
- **getCustomers():** Diese Methode gibt eine Liste der in der Datenbank gespeicherten Kunden zurück.

Die Klasse DBProvider ermöglicht das Laden und Speichern von Kundendaten und Kontoinformationen in einer Datenbank-Datei. Sie stellt auch Funktionen zum Registrieren von Kunden, zum Einloggen von Bankangestellten und Kunden sowie zur Verwaltung von Bankkonten bereit. Die Datenbank kann über die DBProvider-Instanz abgefragt werden, um auf die gespeicherten Daten zuzugreifen und sie zu aktualisieren.

4.2.11 UserInterface

Die Klasse UserInterface stellt ein Benutzerinterface für das Banking System dar. Es erbt von der Klasse JFrame und bietet Funktionen zum Anzeigen von Kontoinformationen, zur Aktualisierung der Kontoliste, zum Erstellen neuer Konten, zur Durchführung von Überweisungen und zum Löschen von Konten.

Methoden:

- **refreshAccountsCallback():** Diese Methode aktualisiert die Kontoliste, indem sie eine neue DefaultListModel erstellt und alle vorhandenen Sparkonten und Girokonten des aktuellen Kunden aus der Datenbank hinzufügt. Die aktualisierte Liste wird dann der AccountListElement-JList zugewiesen.
- **refreshAccountInfo():** Diese Methode aktualisiert die angezeigten Kontodetails, basierend auf dem ausgewählten Konto in der AccountListElement-JList. Sie ruft die relevanten Informationen des ausgewählten Kontos ab und zeigt sie im AccountDetailsText-JTextArea an.
- **UserInterfaceTest():** Der Konstruktor der Klasse erstellt das Benutzerinterface für die Banking-App. Es erstellt ein JFrame mit verschiedenen Panels und Buttons, um die Kontoliste, Kontodetails, Kontoaktionen und Einstellungen anzuzeigen. Die ActionListener werden für die Buttons definiert, um entsprechende Aktionen wie das Erstellen eines

neuen Kontos, das Durchführen einer Überweisung oder das Löschen eines Kontos auszuführen.

Das `UserInterfaceTest`-Objekt ermöglicht es den Benutzern, Kontoinformationen anzuzeigen, Konten zu erstellen, Überweisungen durchzuführen und Konten zu löschen. Es interagiert mit der Datenbank über den `DBProvider`, um auf Kundendaten und Kontoinformationen zuzugreifen und diese zu aktualisieren.

4.2.12 Transfer

Die Klasse `Transfer` stellt ein Fenster dar, das es dem Benutzer ermöglicht, eine Überweisung von einem ausgewählten Bankkonto durchzuführen. Das Fenster enthält Eingabefelder für den Empfänger, den Betrag und den Verwendungszweck der Überweisung. Es enthält auch Abbrechen- und Senden-Buttons, um die Überweisung zu steuern.

Konstruktor:

- **`Transfer(BankAccount selectedAccount, UserInterfaceTest parentComponent)`**: Der Konstruktor erstellt das Fenster für die Überweisung. Er erhält das ausgewählte Bankkonto (`selectedAccount`), von dem aus die Überweisung durchgeführt wird, sowie eine Referenz auf das übergeordnete `UserInterfaceTest`-Objekt (`parentComponent`). Das Fenster wird entsprechend konfiguriert und mit den erforderlichen Komponenten wie Beschriftungen, Textfeldern und Buttons initialisiert. Die Aktionen der Buttons werden definiert, um die Überweisung abzuberechnen oder die Überweisung durchzuführen.

Methoden:

- Keine zusätzlichen öffentlichen Methoden vorhanden.

Die `Transfer`-Klasse ermöglicht es dem Benutzer, eine Überweisung von einem Bankkonto zum anderen durchzuführen. Die eingegebenen Informationen wie Empfänger, Betrag und Verwendungszweck werden abgerufen und validiert. Das ausgewählte Bankkonto wird entsprechend debitiert, und der Betrag wird dem Zielkonto gutgeschrieben, sofern das Zielkonto in der Datenbank vorhanden ist. Die Aktualisierung der Kontoinformationen wird über die Methode `refreshAccountInfo()` des übergeordneten `UserInterfaceTest`-Objekts durchgeführt. Nachdem die Überweisung abgeschlossen ist, wird das Fenster geschlossen und der Benutzer kehrt zum Hauptfenster zurück.

4.2.13 Customer

Die Klasse DBContent ist eine serialisierbare Klasse, die den Inhalt der Datenbank für das Bankanwendungssystem repräsentiert. Sie enthält eine Liste von Customer-Objekten, die die Kundendaten und deren Bankkonten darstellen.

Attribute:

- **CustomerAccounts:** Eine ArrayList vom Typ Customer, die die Kundendaten und deren Bankkonten enthält.

Die Klasse DBContent dient als Container für die Kundendaten und deren Bankkonten. Sie wird für die Serialisierung und Deserialisierung der Datenbank verwendet. Das Attribut CustomerAccounts enthält eine Liste von Customer-Objekten, die alle Kundendaten und deren Bankkonten repräsentieren. Durch die Serialisierung der DBContent-Instanz können die Datenbank Informationen gespeichert und geladen werden, um den Zustand der Anwendung zwischen verschiedenen Ausführungen beizubehalten.

4.2.14 BankerUI

Die Klasse BankerUI ist eine Swing-basierte Benutzeroberfläche für die Bankanwendung des Bankers. Sie ermöglicht dem Banker die Verwaltung von Kundenkonten und bietet verschiedene Funktionen zur Filterung und Anzeige von Kundendaten und Bankkontoinformationen.

Attribute:

- **searchField:** Ein JTextField, das für die Eingabe des Suchbegriffs für die Kunden Filterung verwendet wird.
- **filterComboBox:** Eine JComboBox, die die verfügbaren Filteroptionen für die Kundendaten anzeigt.
- **customerList:** Eine JList, die die Kundenliste darstellt und die Kundenauswahl ermöglicht.
- **bankAccountList:** Eine JList, die die Bankkontoliste des ausgewählten Kunden darstellt und die Auswahl eines Bankkontos ermöglicht.
- **accountLimitField:** Ein JTextField, das für die Eingabe des Limits für ein Girokonto verwendet wird.
- **setLimitButton:** Ein JButton, der zum Festlegen des Limits für ein Girokonto verwendet wird.
- **approveCheckBox:** Eine JCheckBox, die den Freigabestatus eines Bankkontos anzeigt und bearbeitet.
- **accountPanel:** Ein JPanel, das die Informationen zu einem ausgewählten Bankkonto anzeigt.

- **accountDetailsText:** Ein JTextArea, das die Details des ausgewählten Bankkontos anzeigt.

Methoden:

- **getCurrentFilterType():** Eine private Methode, die den ausgewählten Filtertyp aus der filterComboBox zurückgibt.
- **filterByParameters(List<Customer> customers, FilterType filter):** Eine private Methode, die die Kundenliste anhand des Suchbegriffs und des Filtertyps filtert und die gefilterte Liste zurückgibt.
- **filterAccountsByParams(List<BankAccount> accounts, FilterType filter):** Eine private Methode, die die Bankkontoliste anhand des Filtertyps filtert und die gefilterte Liste zurückgibt.
- **refreshAll():** Eine private Methode, die alle Komponenten der Benutzeroberfläche aktualisiert, einschließlich der Kundenliste, der Bankkontoliste und der Bankkontoinformationen.
- **refreshBankAccountInfo():** Eine private Methode, die die Informationen zu einem ausgewählten Bankkonto aktualisiert und in der accountDetailsText anzeigt.
- **refreshCustomers():** Eine private Methode, die die Kundenliste aktualisiert und anhand des aktuellen Suchbegriffs und Filters filtert.
- **refreshBankAccounts():** Eine private Methode, die die Bankkontoliste des ausgewählten Kunden aktualisiert und anhand des aktuellen Filters filtert

Konstruktor:

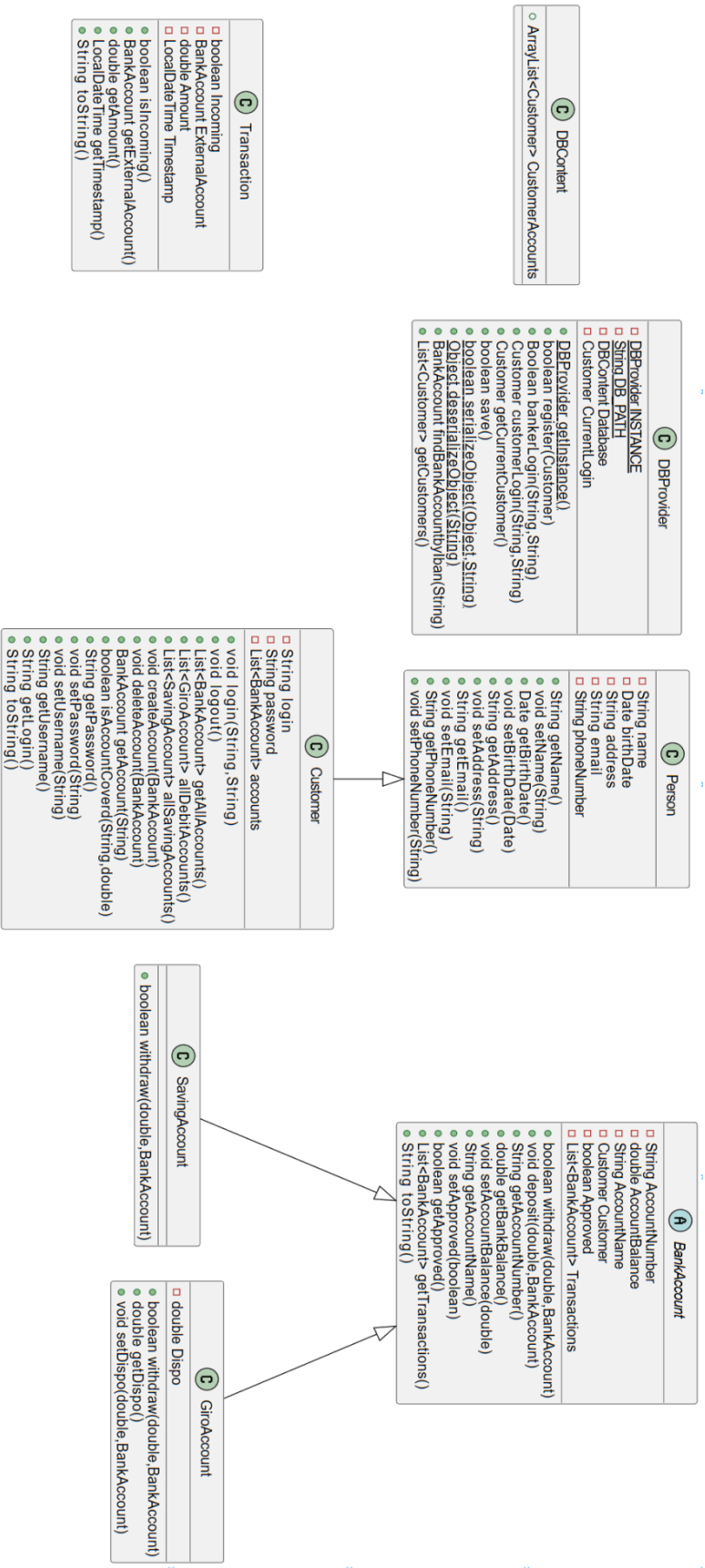
- Der Konstruktor der BankerUI-Klasse initialisiert die Benutzeroberfläche und ihre Komponenten. Er erstellt das Hauptfenster, die Panels für die obere, mittlere und untere Ebene der Benutzeroberfläche und fügt die Komponenten zu den entsprechenden Panels hinzu. Es werden auch die Listener für die Aktionen wie die Auswahl von Kunden oder Bankkonten, das Festlegen des Limits für ein Girokonto und das Aktualisieren der Suchanfrage und des Filters hinzugefügt.

Die Klasse BankerUI bietet dem Banker eine Benutzeroberfläche zur Verwaltung von Kundenkonten. Sie ermöglicht das Filtern von Kundendaten, die Anzeige von Kundenlisten und Bankkontoinformationen sowie die Bearbeitung von Bankkontodetails wie Limits und Freigabestatus.

4.3 Datenbanken

In diesem Projekt hält die aktuelle DBContent Instanz die Daten und wird in einen Objektstream umgewandelt und in eine Datei geschrieben. Die Datei, welche das Produkt dieser Umwandlung der DBContent Instanz darstellt, ist eine generische Binarydatei.

5. Klassendiagramm



6. Ausblick

Die App bietet schon einiges an Funktionen und ermöglicht die Simulation einer Banking App mit ähnlichen Funktionen. Möglich sind noch Kundensupport, andere für den Kunden angenehme Funktionen wie z.B. Daueraufträge, die bei Bedarf implementiert werden können. Als Team 4 sind wir aber sehr zufrieden mit dem letztendlichen Produkt, das rausgekommen ist, da wir dadurch viele Erfahrungen in der Entwicklung mit Java machen konnten, die wir vorher nicht hatten. Generell war es für uns alle das erste große Java Projekt, weshalb es für uns besonders interessant war, daran zu arbeiten.

7. Initiale Zugangsdaten zum Testen

<u>Art des Accounts</u>	<u>Benutzername</u>	<u>Passwort</u>
Bänker	<u>admin</u>	<u>1234</u>
Bänker	<u>admin2</u>	<u>12345</u>
Kunde	<u>Sebas</u>	<u>Sebas</u>
Kunde	<u>Christopher</u>	<u>Jesus1234</u>
Kunde	<u>Kartoffelpuffer</u>	<u>Kartoffelbrei</u>

8. Abbildungsverzeichnis

1. Abbildung 1: Bänker-Login
2. Abbildung 2: Bänker-Interface
3. Abbildung 3: Bänker Account Manipulation
4. Abbildung 4: Kunden Login
5. Abbildung 5: Kunden-Interface
6. Abbildung 6: Neues Konto erstellen
7. Abbildung 7: Konto in der Übersicht
8. Abbildung 8: Überweisungs Fenster
9. Abbildung 9: Transaktionsfenster erhalten (links) und gesendet (rechts)
10. Abbildung 10: Einstellungen