Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Datenbanksysteme

Wintersemester 2025/2026
12. 12. 2025

Prof. Dr. Bernhard Seeger
Amir Tonta, M.Sc.

Vorlesung
Objektorientierte Programmierung

Abgabe: 12. 01. 2026,
bis **spätestens** 23:55 Uhr
über die ILIAS Plattform

Programmierprojekt

In diesem Projekt sollen Sie ein vereinfachtes Bankensystem modellieren und in Java implementieren. Das System umfasst Kunden, Konten sowie grundlegende Bankfunktionen wie Ein- und Auszahlungen sowie Überweisungen. Zusätzlich sollen moderne Zahlungs- und Transferdienste wie Google Pay, Apple Pay oder MoneyGram konzeptionell in das System integriert werden.

Bei der Implementierung ist die **strikte** Einhaltung der Java-Coding-Conventions erforderlich. Alle Methoden sind mit **Javadoc** zu dokumentieren, und längere Codeabschnitte sollten durch aussagekräftige Kommentare erläutert werden. Redundante Code-Wiederholungen sind zu vermeiden.

Laden Sie hierfür das bereitgestellte Java-Gerüst herunter. Bitte beachten Sie, dass im Aufgabenblatt nicht alle zu implementierenden Methoden vollständig aufgeführt sind. Maßgeblich sind die bereitgestellten Interfaces, die Sie nicht ändern dürfen, sowie die zugehörigen Javadoc-Kommentare. Implementieren Sie daher alle dort definierten Methoden vollständig.

Eine detaillierte Aufgabenbesprechung findet im Tutorium statt. Bitte wenden Sie sich bei Fragen direkt an Ihren Tutor oder Ihre Tutorin.

Aufgabe 1: Personalien und Konten (2+4+4) (10 Punkte)

a) Vervollständigen Sie die Klasse `banking.account.BankCustomer` und implementieren Sie die Funktionalitäten gemäß dem Interface `banking.account.Customer`.

b) Nutzen Sie die bereitgestellte Klasse `banking.account.BaseBankAccount`, um zwei konkrete Kontotypen zu ermöglichen. Dazu sollen Sie jeweils ein Feld vom Typ `banking.account.BaseBankAccount` in den zwei zu erstellenden Klassen definieren, um dort Basisfunktionalität darüber abzubilden. Konkret sollen Sie folgende Klassen erstellen:

1. Die Klasse `banking.account.CheckingAccount`, um ein **GiroKonto** zu realisieren. Dieses Konto soll bei der Abfrage `AccountType getAccountType()` entsprechend `Checking` zurückgeben. Zusätzlich wird beim Schließen das Restguthaben ausgezahlt.
2. Die Klasse `banking.account.SavingsAccount`, um ein **Sparkonto** hinzuzufügen. Das Sparkonto kann nur mit einem vorhandenen Girokonto geöffnet werden. Weiter soll dieses Konto bei AbKlassefrage `AccountType getAccountType()` entsprechend `Savings` zurückgegeben werden. Aus diesem Konto kann weder Geld abgehoben noch versendet noch das Terminal verwendet werden. Bei der Schließung wird das Guthaben nicht ausgezahlt (da dies nicht möglich ist), sondern auf das verknüpfte Girokonto überwiesen.

c) Vervollständigen Sie die Klasse `banking.service.BankStatementPrinter`, um eine Konsolenausgabe (Kontoauszug) zu generieren. Gehen Sie dabei wie folgt vor:

1. Fordern Sie den Kunden auf, seine Kontonummer mittels des Scanners einzugeben. Prüfen Sie nun, ob die eingetippte Kontonummer existiert. Ansonsten schreiben Sie eine Fehlermeldung und unterbrechen den Vorgang entsprechend.
2. Falls das Konto gefunden wurde, fordern Sie den Kunden auf, seine Pin mittels des Scanners einzugeben und validieren Sie die Pin. Bei falscher Pin darf die Eingabe noch maximal 3-mal wiederholt werden, bis der Vorgang abbricht und eine entsprechende Meldung in der Konsole erscheint.
3. Zuletzt prüfen Sie bei erfolgreichen Eingaben die Terminalfreigabe und drucken Sie nur dann den Kontoauszug, wenn diese verfügbar ist. Ansonsten wird mit einer Fehlermeldung der Vorgang nicht ausgeführt.

Aufgabe 2: Bankensysteme (4+6) (10 Punkte)

a) Fügen Sie der Klasse `banking.bank.BaseBank` Funktionalitäten gemäß dem Interface `banking.bank.Bank` hinzu. Verwalten Sie die Kunden und Bankkonten jeweils in Arrays. Diese beiden Felder sollen nicht als `private` deklariert werden, um einen späteren direkten Zugriff zu ermöglichen. Alle weiteren benötigten Felder bleiben `private` und sollen ausschließlich über die jeweiligen `Getter-` und `Setter-`Methoden erreichbar sein.

b) Vervollständigen Sie nun die Klasse `banking.bank.BICBank` und implementieren Sie Funktionalitäten gemäß dem Interface `banking.bank.SWIFTBank`. Beachten Sie, dass hier die Basisfunktionalität über ein Feld vom Typ `banking.bank.BaseBank` abgebildet werden soll. Bei der Methode `public String toString()` sollen neben Bankname, BIC auch alle Kunden und Konten als Information enthalten sein. Greifen Sie daher auf die Felder mit den jeweiligen Daten direkt zu (z.B. `this.baseBank.customersArr`), da diese nicht `private` markiert sind. Weiter kann eine `banking.bank.SWIFTBank` zusätzlich zu Überweisungen innerhalb der Bank selbst auch Überweisungen an andere Banken tätigen. Bei der Ausführung einer Transaktion via `boolean executeTransaction(Transaction)` sind folgende Punkte zu beachten:

- Wenn die Transaktion `null` ist oder ein Enddatum hat, soll keine Überweisung erfolgen.
- Falls die Transaktion innerhalb derselben Bank erfolgen soll, soll sie direkt über die Bank selbst abgewickelt werden.
- Bei verschiedenen Banken müssen beide Konten bei der jeweiligen Bank vorhanden sein, die Sender-Bank, die auszuführende Bank, sein, die Namen beider beteiligter Kunden überprüft werden sowie überprüft werden, ob der Sender ausreichende Deckung besitzt.
- Überweisungen von Sparkonten sind unabhängig von der Deckung nicht möglich. Jedoch ist es erlaubt, auf ein Sparkonto zu überweisen.
- Schließlich soll bei erfolgreicher Ausführung das Enddatum gesetzt werden, um die Transaktion als ausgeführt zu markieren und die Transaktion in die Historie der Bank eingetragen werden.

Aufgabe 3: Geldtransfersysteme (10+10) **(20 Punkte)**

a) Vervollständigen Sie die Klasse `banking.transfer.SWIFTSystem`, sodass gemäß dem Interface `banking.transfer.TransactionTransferSystem` die erforderlichen Funktionalitäten implementiert werden, um ein zentrales Überweisungssystem zu modellieren. Dieses System registriert Banken und koordiniert so eingehende Überweisungen. Speichen Sie die Registrierungen in einem Feld vom Typ `SWIFTBank[]` ab. Weiter soll dieses System Überweisungen ablehnen, falls in der Transaktion die Sender-Bank weder eine `banking.bank.SWIFTBank` ist noch im `SWIFTSystem` registriert. Ansonsten sind Ausführungen über die Banken selbst vorzunehmen. D.h. nur wenn die Sender-Bank eine `banking.bank.SWIFTBank` ist, muss sie im `banking.transfer.SWIFTSystem` eingetragen sein, um Transaktionen auszuführen. Beachten Sie außerdem, dass es sich hierbei eine Klasse (`banking.transfer.SWIFTSystem`) handelt, die wegen des einzigen privaten Konstruktors sich nirgendwo erzeugen lässt. Es existiert daher nur eine einzige Instanz als Konstante namens `SWIFT_INSTANCE`. Schließlich soll bei einer erfolgreichen register(SWIFTBank) Anfrage, der Bank eine BIC zugewiesen bekommen. Die BIC ist die Indexposition im Array `SWIFTBank[]` und somit immer eindeutig.

b) Implementieren Sie die Klasse `banking.transfer.PaymentService`, um zusätzliche Zahlungsdienste wie MoneyGram, Western Union und ähnliche Anbieter im Zahlungsverkehr zu unterstützen. Diese Services ermöglichen es, Überweisungen über das `banking.transfer.SWIFTSystem` durchzuführen, ohne ein eigenes Bankkonto zu besitzen. Dazu bekommt bei einer eingehenden Transaktion der Kunde kurzzeitig ein Bankkonto mit dem Geld bzw. Kundendaten, um diese Überweisung über das `banking.transfer.SWIFTSystem` zu tätigen. Beachten Sie hier ebenfalls, dass diese Klasse keinen öffentlichen Konstruktor besitzt, und daher auch nur die vordefinierten Konstanten zugreifbar sind. Diese Konstanten bilden die verfügbaren Dienste ab, z.B. `WERO` und `KLARNA`. All diese Dienste arbeiten mit dem `banking.transfer.SWIFTSystem`, d.h. eingehende Anfragen werden an das `SWIFTSystem.SWIFT_INSTANCE` weitergeleitet. Die Methoden `public SWIFTBank getByName(String)` und `public boolean submitTransaction(Transaction)` müssen angepasst werden und nicht nur direkt weiter delegiert werden. D.h. wenn eine Abfrage via getByName(KlarnaName) gestellt wird, soll auch der Dienst `KARNA` zurückgegeben werden, ansonsten falls es sich nicht um ein `PaymentService` handelt, soll weiter an das `SWIFTSystem.SWIFT_INSTANCE` delegiert werden. Um eine Transaktion mittels eines `PaymentService` auszuführen, wird explizit geprüft ob die Sender-Bank `null` ist, weil nur so kann der Dienst dann die Überweisung tätigen, indem er ein temporäres Bankkonto in einer ggf. temporären aber gültigen und regenerierten `banking.bank.SWIFTBank` anlegt und so die Überweisung abwickelt.

Aufgabe 4: Testing **(10 Punkte)**

1. Schreiben Sie für jede implementierte Funktionalität der Aufgabe 1 mindestens 2 **JUnit**-Tests. Testen Sie zum Beispiel, wenn ein Kunde versucht, das Terminal zu nutzen, obwohl er keine Befugnis besitzt, oder wenn ein Kunde versucht, Geld aus seinem Sparkonto auszuzahlen.
2. Prüfen Sie bei der Überweisung in der `banking.bank.BICBank` alle Fälle, z.B. wenn eine Überweisung ausgehend von einem Sparkonto angefragt wird oder wenn eine Transaktion schon bereits als ausgeführt markiert war. Machen Sie hierfür jeweils eine **JUnit**-Testmethode.
3. Bei den Geldtransfersystemen sollen Sie vorhandene Dienste mittels **JUnit**-Tests prüfen. Gehen Sie zum Beispiel auf die Fälle ein, in denen ausschließlich per Bargeldzahlung überwiesen wird oder eine Überweisung an einen unbekannten, nicht registrierten Dienst versucht wird.