

Bankwesen

Banken bieten eine Vielzahl an Sparprodukten für Ihre Kunden an. Im Rahmen dieser Aufgabe sollen einige davon (oberflächlich) simuliert werden.

Es sollen unterschiedliche Formen von **Konten** implementiert werden, nämlich ein **Sparbuch**, ein **Girokonto** und ein **Kreditkonto**. Für jedes dieser Konten soll eine eigene Klasse implementiert werden. Alle haben eine gemeinsame Oberklasse **Konto**, von der keine Objekte angelegt werden dürfen.

Überblick:

Allen Klassen gemeinsam sind folgende Eigenschaften:

- Kontonummer (grundsätzlich eine Zahl, die in manchen Fällen (siehe unten) 6-stellig mit führenden 0-ern dargestellt/verwendet werden soll) .. z.B.: 000001
- Kontostand (Fließkommawert)
- Zinssatz (Fließkommawert)
- Eine Liste von **Strings**, in der die einzelnen Buchungsvorgänge „gesammelt“ werden. (**ArrayList** (für volle Punktezahl) oder **String-Array** (mit Abzügen))

Alle Klassen haben folgende Methoden gemeinsam:

- **void verrechneZinsen()** – diese Methode berechnet aus Kontostand und Zinssatz den aktuellen Zinsertrag und verändert dementsprechend den Kontostand.
- **void speichereKontoauszug()** – diese Methode speichert den Inhalt der Liste mit den Buchungsvorgängen in einer Datei – der Dateiname soll „auszugnnnnn.txt“ sein, wobei nnnnnn die Kontonummer ist (z.B.: **auszug123301.txt**)

Sorge in der Klasse **Konto** dafür, dass alle abgeleiteten Klassen eine Methode **double einzahlung(double betrag)** implementieren MÜSSEN, mit der Einzahlungen auf das jeweilige Konto durchgeführt werden. Die Methode soll zurückliefern, wie viel eingezahlt werden konnte (Spezialfall bei Kreditkonto ...)

Klasse Konto (11,5 P)

- Ist die gemeinsame Oberklasse für alle weiteren Klassen
- Es dürfen keine Objekte dieser Klasse angelegt werden
- Lege hier alle sinnvollen Eigenschaften und Methoden (wie oben beschrieben) fest.
- Definiere für diese Klasse zwei Konstruktoren, die von den abgeleiteten Klassen verwendet werden sollen:
 - Ein Konstruktor mit zwei Parametern: Betrag (der Startkontostand) und Zinssatz
 - Ein Konstruktor mit einem Parameter: Zinssatz → Kontostand ist in diesem Fall 0.0
 - Diese beiden Konstruktoren sollen bei jedem Aufruf auch eine neue Kontonummer generieren, die jeweils um 1 höher ist als die letzte bisher vergebene. Die allererste Kontonummer soll 1 sein.

Klasse Sparbuch (5 P)

- Ein Sparbuch soll auf zwei Arten angelegt werden können (Konstruktoren)
 - Übergabe nur des Zinssatzes → Kontostand ist dann 0
 - Übergabe von Zinssatz UND Startkontostand
- Ein Sparbuch hat zusätzlich (zu den gemeinsamen Methoden aller Konten) eine Methode **boolean auszahlung(double betrag)**
 - Mit dieser Methode wird Geld vom Sparbuch abgehoben. Dies ist nur möglich, wenn der gewünschte Betrag auch auf dem Konto verfügbar ist!
- Die verpflichtende Methode **einzahlung(...)** erhöht den Kontostand um den übergebenen Betrag, wenn dieser Betrag nicht negativ ist!! ...
- Sorge dafür, dass eine Ausgabe des Objektes in folgendem Format durchgeführt wird:
000002 - Sparbuch: Zinssatz: 0.5%, Kontostand: 1000.0

Klasse Girokonto (7,5 P)

- Ein Girokonto hat eine zusätzliche Eigenschaft: den **Namen** des Kontoinhabers (**String**)
- Ein Girokonto soll auf zwei Arten angelegt werden können (Konstruktoren)
 - Übergabe von Name und Zinssatz → Kontostand ist dann 0
 - Übergabe von Name, Zinssatz UND Startkontostand
- Die Methoden **auszahlung(...)** und **einzahlung(...)** sind wie beim Sparbuch umzusetzen
- Von einem Girokonto können Überweisungen auf jeden anderen Kontotyp durchgeführt werden: schreibe dafür die Methode
double ueberweisung(Konto empfaenger, double betrag) { .. }
Diese Methode soll den übergebenen Betrag, der weder negativ noch größer als der Kontostand sein darf an das übergebene andere Konto überweisen. (Du solltest dafür die Methoden **einzahlung()** und **auszahlung()** sinnvoll einsetzen)
- Sorge dafür, dass eine Ausgabe des Objektes in folgendem Format durchgeführt wird:
000003 - Girokonto von Max Muster: Zinssatz: 0.05%, Kontostand: 1000.0

Klasse Kreditkonto (6 P)

Ein Kredit wird von Banken wie ein Sparbuch oder Konto geführt, nur dass der Kontostand bei Aufnahme des Kredites negativ ist (z.B.: -50.000 bei einem Kredit von 50.000 EUR und die Zinsen nicht gutgeschrieben werden sondern die Schulden noch weiter vergrößern. Ziel ist es, durch Einzahlung der Kreditraten den Kontostand auf 0 zu bringen

- Ein Kredit hat eine zusätzliche Eigenschaft: den **Namen** des Kreditnehmers (String)
- Ein Kredit kann nur auf eine Art angelegt werden (Konstruktor)
 - Übergabe von Name, Zinssatz UND Kreditbetrag .. dieser soll als positiver Wert übergeben werden, der Kontostand dann entsprechend negativ gesetzt werden
- Auf ein Kreditkonto kann **nur eingezahlt** werden → Methode **einzahlung(...)** .. hier ist folgendes zu berücksichtigen:
 - Nach einer Einzahlung darf der (Kredit)Kontostand nicht größer als 0 sein, denn dann ist das „Ziel“ erreicht und der Kredit ausbezahlt (getilgt).
 - Sollte dieser Fall eintreten, so wird nur so viel eingezahlt, dass der Kontostand letztlich 0 beträgt → Rückgabewert!!
- Sorge dafür, dass eine Ausgabe des Objektes in folgendem Format durchgeführt wird:
000001 - Kreditkonto von Max Muster: Zinssatz: 3.0%, Kontostand: -500.0

Die Buchungen

Alle Transaktionen (Kontoeröffnung, Einzahlung, Auszahlung, Überweisung, Kredittilgung-Spezialfall einer Einzahlung) in einem Konto sollen in der Liste der Buchungen gespeichert werden.

Folgende Einträge sollten (z.B.: für einen Kredit) in dieser Liste gespeichert werden:

“Kontoeröffnung: Betrag = -500.0, Zinssatz = 3.0%“

“Zinsen verrechnet: -15.0“

“Einzahlung: 150.0“

“Kredit getilgt mit: 365.0“

weitere mögliche Buchungen:

“Auszahlung: 200.0“

“Überweisung an 000001: 365.0“

Kontoauszug speichern

Die Liste der Buchungen eines Kontos soll durch Aufruf der Methode **speichereKontoauszug()** im jeweiligen Objekt zeilenweise in eine Textdatei übertagen werden.

Testklasse

Lege in einer Testklasse mehrere unterschiedliche Kontoobjekte an und teste ausgiebig die geforderten Methoden.

Schreibe in der Testklasse noch eine zusätzliche Methode **(2 P)**

```
double berechneSumme(...) { .. }
```

, der eine **beliebige Anzahl** von Konten übergeben werden kann und die die Gesamtsumme der Kontostände der übergebenen Konten berechnet und zurückliefert.