Einzelprüfung „Automatentheorie / Komplexität / Algorithmen (vertieft)“

Einzelprüfungsnummer 66112 / 2002 / Herbst

Thema 1 / Aufgabe 4 (*Banksystem*)

Stichwörter: Vererbung, Abstrakte Klasse, Implementierung in Java, Klassendiagramm

In einer Anforderungsanalyse für ein Banksystem wird der folgende Sachverhalt beschrieben:

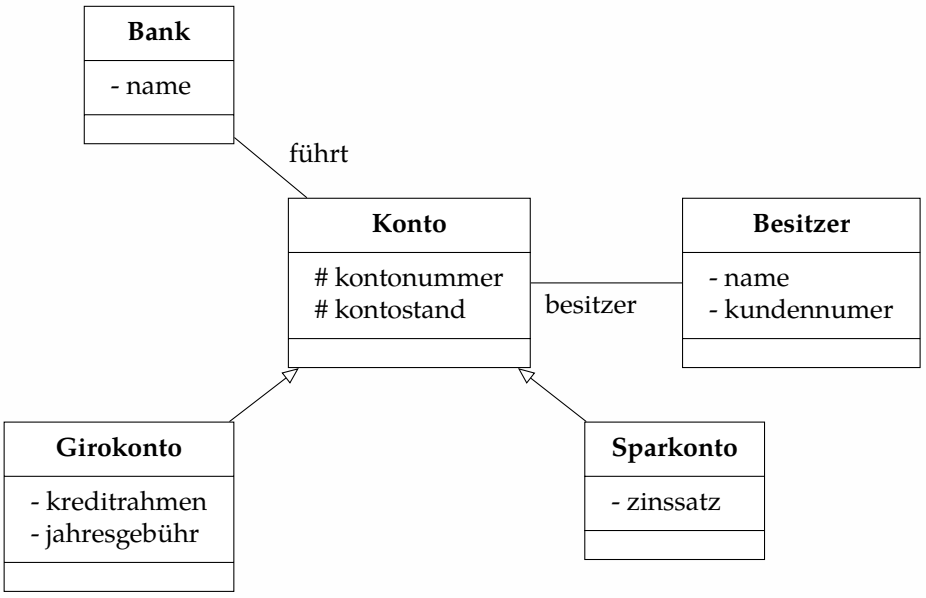
Eine Bank hat einen Namen und sie führt Konten. Jedes Konto hat eine Kontonummer, einen Kontostand und einen Besitzer. Der Besitzer hat einen Namen und eine Kundennummer. Ein Konto ist entweder ein Sparkonto oder ein Girokonto. Ein Sparkonto hat einen Zinssatz, ein Girokonto hat einen Kreditrahmen und eine Jahresgebühr.

(a) Deklarieren Sie geeignete Klassen in Java [oder in C++], die die oben beschriebenen Anforderungen widerspiegeln! Nutzen Sie dabei das Vererbungskonzept aus, wo es sinnvoll ist! Gibt es Klassen, die als abstrakt zu verstehen sind?

[Hinweis: Geben Sie sowohl ein Klassendiagramm als auch den Quellcode für die beteiligten Klassen inkl. Attributen, ohne Methoden an! Achtung: in Teilaufgabe b) werden anschließend die benötigten Konstruktoren verlangt; Um die verschiedenen Konten in einer Bank zu verwalten, eignet sich das Array NICHT als Datenstruktur. Informieren Sie sich über die Datenstruktur „ArrayList“ und verwenden Sie diese.]

Lösungsvorschlag

Die Klasse Konto ist abstrakt, da ein Konto immer entweder ein Spar- *oder* ein Girokonto ist. Ein Objekt der Klasse Konto ist deshalb nicht sinnvoll.



(b) Geben Sie für alle nicht abstrakten Klassen benutzerdefinierte Konstruktoren an mit Parametern zur Initialisierung der folgenden Werte: der Name einer Bank, die Kontonummer, der Kontostand, der Besitzer und der Zinssatz (bzw. Kreditrahmen und Jahresgebühr) eines Sparkontos (bzw. Girokontos), der Name und die Kundennummer eines Kontobesitzers.

Ergänzen Sie die Klassen um Methoden für die folgenden Aufgaben! Nutzen Sie wann immer möglich das Vererbungskonzept aus und verwenden Sie ggf. abstrakte (bzw. virtuelle) Methoden!

[Achtung: Ergänzen Sie ggf. alle benötigten Getter und Setter in den beteiligten Klassen!]

Lösungsvorschlag

Konstruktoren ergänzen:

Bemerkung: Auch eine abstrakte Klasse kann einen Konstruktor besitzen, dieser kann nur nicht ausgeführt werden. In den abgeleiteten Klassen kann dieser Super-Konstruktor aber verwendet werden.

(c) Auf ein Konto soll ein Betrag eingezahlt und ein Betrag abgehoben werden können. Soll von einem Sparkonto ein Betrag abgehoben werden, dann darf der Kontostand nicht negativ werden. Bei einer Abhebung von einem Girokonto darf der Kreditrahmen nicht überzogen werden.

Lösungsvorschlag

Bemerkung: Die Methode `einzahlen` ist in der Klasse `Konto` implementiert, da sie sich für Spar- und Girokonten nicht unterscheidet im Gegensatz zur Methode `abheben`, die in beiden Klassen unterschiedlich implementiert ist. [Sie wird als abstrakte Methode in der Klasse Konto angegeben, um ihre Implementierung (Überschreiben) zu gewährleisten.] Kreditrahmen wird als negativer Wert gespeichert. Die Methoden zum Abheben liefern zusätzlich zur Änderung des Kontostandes eine Rückmeldung bezüglich Erfolg oder Misserfolg der Abbuchung.

(d) Ein Konto kann eine Jahresabrechnung durchführen. Bei der Jahresabrechnung eines Sparkontos wird der Zinsertrag gutgeschrieben, bei der Jahresabrechnung eines Girokontos wird die Jahresgebühr abgezogen (auch wenn dadurch der Kreditrahmen überzogen wird).

Lösungsvorschlag

Anmerkung: Im Folgenden nur noch Angabe der gesuchten Methoden, alle vorherigen Implementierungen (Attribute, Konstruktoren, Methoden, s. o.) wurden nicht nochmals aufgeführt.

(e) Eine Bank kann einen Jahresabschluss durchführen. Dieser bewirkt, dass für jedes Konto der Bank eine Jahresabrechnung durchgeführt wird.

(f) Eine Bank kann ein Sparkonto eröffnen. Die Methode soll die folgenden fünf Parameter haben: den Namen und die Kundennummer des Kontobesitzers, die Kontonummer, den (anfänglichen) Kontostand und den Zinssatz des Sparkontos. Alle Parameter sind als String-Objekte oder als Werte eines Grunddatentyps zu übergeben! Das Sparkonto muss nach seiner Eröffnung in den Kontenbestand der Bank aufgenommen sein.

[Hinweis: Falls der Kunde schon mit einem Konto in der Bank geführt ist, können die Werte für das `Besitzer`-Objekt übernommen werden. Schreiben Sie daher eine Hilfsmethode `Besitzer schonVorhanden(String name, int kunr)`, die prüft, ob der Kunde mit `name` und `kunr` schon in der Liste vorhanden ist und diesen bzw. andernfalls einen neu erzeugten Besitzer zurückgibt.]

[Sinnvolle/notwendige Methoden der Klasse ArrayList für diese Aufgabe:
`public int size()` : Returns the number of elements in this list.
`public boolean isEmpty()` : Returns true if this list contains no elements.
`public E get(int index)` : Returns the element at the specified position in this list.
`public boolean add(E e)` : Appends the specified element to the end of this list.
(siehe auch Java-Api: https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html)]

Lösungsvorschlag

```java
import java.util.ArrayList;

public class Bank {
  @SuppressWarnings("unused")
  private String name;
  private ArrayList<Konto> konten;

  public Bank(String name) {
    this.name = name;
    konten = new ArrayList<Konto>();
  }

  public void rechneAb() {
    for (int i = 0; i <= konten.size(); i++) {
      konten.get(i).rechneAb();
    }
  }

  public void legeAn(String name, int kundenNummer, int kontoNummer, float
  ↪ kontoStand, float zinssatz) {
    Besitzer besitzer = schonVorhanden(name, kundenNummer);
    konten.add(new Sparkonto(kontoNummer, kontoStand, besitzer, zinssatz));
  }

  public Besitzer schonVorhanden(String name, int kundenNummer) {
    if (!konten.isEmpty()) {
      for (int i = 0; i < konten.size(); i++) {
        if (konten.get(i).besitzer.gibKundenNummer() == kundenNummer) {
          return konten.get(i).gibBesitzer();
        }
      }
    }
    return new Besitzer(name, kundenNummer);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Bank.java`

```java
public abstract class Konto {
  protected int kontoNummer;
  protected float kontoStand;
  protected Besitzer besitzer;

  public Konto(int kontoNummer, float kontoStand, Besitzer besitzer) {
    this.kontoNummer = kontoNummer;
    this.kontoStand = kontoStand;
```

```
    this.besitzer = besitzer;
  }

  public void zahleEin(float betrag) {
    kontoStand += betrag;
  }

  public Besitzer gibBesitzer() {
    return besitzer;
  }

  public abstract boolean hebeAb(float betrag);

  public abstract void rechneAb();

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Konto.java`

```
public class Sparkonto extends Konto {
  private float zinssatz;

  public Sparkonto(int kontoNummer, float kontoStand, Besitzer besitzer, float
↪   zinssatz) {
    super(kontoNummer, kontoStand, besitzer);
    this.zinssatz = zinssatz;
  }

  public boolean hebeAb(float betrag) {
    if (kontoStand - betrag >= 0) {
      kontoStand -= betrag;
      return true;
    } else {
      return false;
    }
  }

  public void rechneAb() {
    kontoStand += kontoStand * (1 + zinssatz);
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Sparkonto.java`

```
public class Girokonto extends Konto {
  private float kreditRahmen;
  private float jahresGebühr;

  public Girokonto(int kontoNummer, float kontoStand, Besitzer besitzer, float
↪   kreditRahmen, float jahresGebühr) {
    super(kontoNummer, kontoStand, besitzer);
    this.kreditRahmen = kreditRahmen;
    this.jahresGebühr = jahresGebühr;
  }
```

```
  public boolean hebeAb(float betrag) {
    if (kontoStand - betrag >= kreditRahmen) {
      kontoStand -= betrag;
      return true;
    } else {
      return false;
    }
  }

  public void rechneAb() {
    kontoStand -= jahresGebühr;
  }
}
```

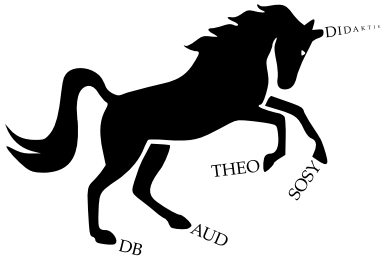
Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Girokonto.java`

```
@SuppressWarnings("unused")
public class Besitzer {
  private String name;
  private int kundenNummer;
  private Konto hatKonto;

  public Besitzer(String name, int kundenNummer) {
    this.name = name;
    this.kundenNummer = kundenNummer;
  }

  public int gibKundenNummer() {
    return kundenNummer;
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66112/jahr_2002/herbst/Besitzer.java`



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.Der TEX-Quelltext dieser Aufgabe kann unter folgender URL aufgerufen werden: `https://github.com/bschlangaul-sammlung/examens-aufgaben-tex/blob/main/Examen/66112/2002/09/Thema-1/Aufgabe-4.tex`