Übung 4

In dieser Übung sollen Sie eine Klasse aus der Collection-Bibliothek nutzen.

a) Bank (*)

Die Klassen Konto, Girokonto, Sparbuch usw. kennen Sie ja schon. Sie sollen nun eine Klasse `Bank` schreiben, die diese Konten verwaltet.

Sie soll über folgende öffentliche Methoden/Konstruktoren verfügen:

- `public Bank(long bankleitzahl)`
  erstellt eine Bank mit der angegebenen Bankleitzahl
- `public long getBankleitzahl()`
  liefert die Bankleitzahl zurück
- `public long girokontoErstellen(Kunde inhaber)`
  erstellt ein Girokonto für den angegebenen Kunden. Dabei soll die Methode eine neue, noch nicht vergebene Kontonummer erzeugen, das neue Girokonto mit dieser Nummer in der Kontenliste speichern und die vergebene Kontonummer zurückgeben.
  - Man kann natürlich aus den gespeicherten Konten heraus eine neue noch nicht verwendete Kontonummer erzeugen. Leichter ist es aber, einfach eine zusätzliche Membervariable zu verwenden, in der man sich merkt, welches die größte bisher vergebene Nummer ist.
- `public long sparbuchErstellen(Kunde inhaber)`
  genauso mit Sparbuch
- `public String getAlleKonten()`
  liefert eine Auflistung von Kontoinformationen aller Konten (mindestens Kontonummer und Kontostand)
- `public List<Long> getAlleKontonummern()`
  liefert eine Liste aller gültigen Kontonummern in der Bank
- `public boolean geldAbheben(long von, double betrag)`
  hebt den Betrag vom Konto mit der Nummer `von` ab und gibt zurück, ob die Abhebung geklappt hat.
- `public void geldEinzahlen(long auf, double betrag)`
  zahlt den angegebenen Betrag auf das Konto mit der Nummer `auf` ein
- `public boolean kontoLoeschen(long nummer)`
  löscht das Konto mit der angegebenen `nummer` und gibt zurück, ob die Löschung geklappt hat (Kontonummer nicht vorhanden)
- `public double getKontostand(long nummer)`
  liefert den Kontostand des Kontos mit der angegebenen `nummer` zurück.
- `public boolean geldUeberweisen(long vonKontonr, long nachKontonr, double betrag, String verwendungszweck)`
  überweist den genannten Betrag vom Girokonto mit der Nummer `vonKontonr` zum Girokonto mit der Nummer `nachKontonr` und gibt zurück, ob die Überweisung geklappt hat (nur bankinterne Überweisungen!) Überlegen Sie gut, was dabei alles schief gehen kann, so dass Sie `false` zurückliefern müssen! (Vorsicht: Geld darf nicht einfach „verschwinden“.)

Wenn Sie möchten, können Sie die Signaturen der Methoden auch etwas abändern, um auf die unterschiedlichen Fehlersituationen hinzuweisen, z.B. könnte eine `KontonummerNichtVorhandenException` (oder so) geworfen werden.

Dorothea Hubrich
d_hubrich@web.de

Tipp: Denken Sie gut darüber nach, welche Speicherstruktur aus dem Collection-Framework sich hier zur Speicherung der Konten anbietet – List, Map oder Set. Sehen Sie sich die Methoden dieser drei Interfaces in der Dokumentation an.

Bitte laden Sie Ihre Lösung ab jetzt wieder in Moodle hoch. Git ist sehr schön, um gemeinsam zu arbeiten, nicht aber, um viele verschiedene Lösungen anzuschauen…

b) Kontoauszug

Schreiben Sie eine neue Klasse `Kontoaktion`. Sie stellt einen Eintrag dar, wie er üblicherweise auf einem Kontoauszug zu finden ist, z.B. „Abhebung von 100,- € am 13.5.2013“ oder auch „1.3.2013: Erhöhung der Dispozinsen auf 7,9 %“. Eine solche Kontoaktion hat folgende Membervariablen mit sinnvollen Zugriffsmethoden:

- ein beschreibender Text (z.B. „Abhebung“ oder „neuer Dispozinssatz von 7,9%“)
- ein Betrag (z.B. -100 oder auch 0, wenn keine Kontoaktion mit diesem Eintrag verbunden ist)
- ein Datum

Schreiben Sie zwei Konstruktoren, einer mit Parameterwerten für Text und Betrag, einer mit nur einem Parameterwert für den Text (Betrag wird dann auf 0 gesetzt). In beiden Fällen soll das heutige Datum gespeichert werden.
Alternative: Machen Sie Kontoaktion abstrakt und schreiben Sie für jede mögliche Aktion (Einzahlung, Abhebung, …) eine passende Unterklasse. Die geerbte Methode getMeldung() liefert dann den zur jeweiligen Aktion passenden Text.

Geben Sie der Klasse Konto eine Eigenschaft vom Typ `List<Kontoaktion>`.

- Sorgen Sie dafür, dass bei jeder Aktion (einzahlen, abheben, überweisen, Konto erstellen,…) ein Kontoaktion-Objekt erzeugt und in der Liste gespeichert wird. Überlegen Sie sich gut, welche Sichtbarkeit Ihre Eigenschaft haben soll.
- Schreiben Sie in Konto eine Methode
  `String getKontoauszug()`
  Die den gesamten Kontoauszug, also alle Kontoaktionen hintereinanderweg, zurückliefert.
- Schreiben Sie in Konto eine Methode
  `void alteEintraegeLoeschen(LocalDate vor)`
  Sie soll alle alten Kontoaktionen vor dem angegebenen Datum löschen.