technische universität dortmund

fakultät für informatik

Elias Kuthe
Roman Kalkreuth
Bernd Zey

Dortmund, den 10. Januar 2020

Praktikum zur Vorlesung Einführung in die Programmierung WS 19/20

Blatt 11

Es können 30 Punkte erreicht werden.

Allgemeine Hinweise

1. Bitte lesen Sie vor der Bearbeitung **alle** Aufgaben sorgfältig durch! Dies erspart Ihnen unnötige Arbeit und somit auch Zeit!
2. Lassen Sie sich fertiggestellte Aufgaben bitte möglichst **frühzeitig** testieren. In der letzten halben Stunde vor Schluss werden bei diesem Blatt nur noch **zwei** Teil-Aufgaben testiert!
3. Wir akzeptieren ein Testat nur, wenn die Lösung eigenständig auf Anhieb erklärt werden kann. Andernfalls müssen wir die entsprechende Teilaufgabe mit 0 Punkten bewerten.

Aufgabe: Bankkonten (15 Punkte)

In dieser Aufgabe soll eine Klassenhierarchie für verschiedene Arten von Bankkonten implementiert werden.

- Alle *Bankkonten* haben eine Kontonummer (`unsigned long`), einen Inhaber (Klasse `string` aus dem gleichnamigen Header) und einen aktuellen Kontostand (`int`). Außerdem können auf alle Arten von Konten Geldbeträge gutgeschrieben werden. Alle Konten weisen zum Zeitpunkt der Eröffnung einen Kontostand von 0 € auf. Eine Ausnahme bildet das Sparkonto, welches (als Sparanreiz) zur Kontoeröffnung einen Kontostand von 1 € aufweist.

  Die Klasse `Bankkonto` soll folgende Methoden bereitstellen:

  - `void gutschreiben(int betrag)`: Gutschreiben eines positiven Betrags auf das Konto. Falls der Betrag nicht positiv ist, soll eine Fehlermeldung ausgegeben und der Kontostand nicht verändert werden.
  - `int kontostand() const`: Rückgabe des aktuellen Kontostands.
  - `void anzeigen() const`: Ausgabe aller Kontoinformationen auf der Konsole.
  - `unsigned long kontonummer() const`: Rückgabe der Kontonummer.

- Beim `Sparkonto` sind zusätzlich Auszahlungen über die Methode `int auszahlen(int betrag)` möglich, wobei nach einer Auszahlung wenigstens 1 € auf dem Konto verbleiben muss. Wird ein höherer Geldbetrag angefordert, wird nur der maximal mögliche Betrag ausgezahlt. Der tatsächliche Wert der Auszahlung soll zurückgegeben werden. Soll auch der letzte Euro ausgezahlt werden, muss das Konto aufgelöst werden (Methode `int aufloesen()`, analog zu `auszahlen`). Nach der Auflösung sind keine weiteren Gutschriften auf dieses Konto mehr möglich. Sollte dies dennoch versucht werden, soll eine Fehlermeldung ausgegeben und die Transaktion ignoriert werden.

- Beim `Girokonto` sind ebenso wie bei Sparkonten Auszahlungen und zudem Überweisungen (Methode `int ueberweisen(Bankkonto& zielKonto, int betrag)`) auf beliebige andere Bankkonten möglich. Im Gegensatz zu den Sparkonten dürfen bei Girokonten die Kontostände auch negativ werden (sogenannte Überziehung), jedoch nur bis -1000 €. Wenn durch eine Überweisung dieses Limit unterschritten würde, soll der Überweisungsbetrag entsprechend reduziert werden. Die Methode soll dann den Betrag zurückgeben, der tatsächlich überwiesen wurde. Girokonten können nicht aufgelöst werden.
- Beim `Tagesgeldkonto` sind Umbuchungen auf ein, bei der Eröffnung festgelegtes, Referenzkonto vom Typ `Girokonto` möglich. Die entsprechende Methode lautet `int umbuchen(int betrag)`. Ansonsten unterscheidet sich ein Tagesgeldkonto nicht von einem Bankkonto. Durch eine Umbuchung darf kein negativer Kontostand entstehen; daher wird bei Bedarf der Überweisungsbetrag automatisch entsprechend reduziert.

Halten Sie sich bei Ihrer Implementierung an die folgenden **Einschränkungen:**

- Nutzen Sie die Vererbung aus, um redundanten Code zu vermeiden. Eine Implementierung aller Methoden in allen Klassen ist daher nicht erlaubt.
- Implementieren Sie alle Methoden außerhalb der Klassendefinitionen in den entsprechenden Quelldateien (`*.cpp`).
- Machen Sie von den Spezifikationen `protected`, `private` und `public` sinnvoll Gebrauch.
- Verwenden Sie immer Cent-Beträge.

a) Skizzieren Sie für die beschriebenen Kontoarten eine sinnvolle Klassenhierarchie, welche Gemeinsamkeiten zwischen den verschiedenen Kontoarten so gut wie möglich ausnutzt. Geben Sie dabei für jede Klasse ihre Attribute und Methoden an — in Unterklassen nur die neuen Attribute, sowie die neuen oder überladenen Methoden. Orientieren Sie sich an dem Klassenhierarchie-Diagramm auf Folie 3 in Kapitel 11.

HINWEIS: Es gibt Eigenschaften, die nur zwei der drei Kontoarten gemeinsam haben. Fügen Sie eine weitere Klasse zur Klassenhierarchie hinzu, um Redundanzen zu reduzieren.

HINWEIS: Lassen Sie sich diese Aufgabe testieren, bevor Sie mit den folgenden Aufgaben weiter machen.

(7)

b) Legen Sie ein neues Projekt an und erstellen Sie für jede Klasse eine Header- und eine Quelldatei (Format: `<klassenname>.h` und `<klassenname>.cpp`) im Ordner `src`. Implementieren Sie die Klassenhierarchie (nur Klassendefinitionen, keine Methodendefinitionen).

_____ (5)

c) Implementieren Sie alle Methoden und Konstruktoren in den entsprechenden Quelldateien.

_____ (8)

d) Legen Sie eine `.cpp`-Datei mit der `main`-Funktion an. Testen Sie hier Ihre Implementierung indem Sie je ein Giro-, Spar- und Tagesgeldkonto erzeugen (das Referenzkonto vom Tagesgeldkonto soll das Girokonto sein). Geben Sie zu Beginn die Informationen aller Konten aus. Führen Sie die folgenden Operationen durch und geben Sie nach jeder Transaktion den neuen Kontostand aus. Bei Auszahlungen, Überweisungen oder Umbuchungen soll auch der tatsächliche Betrag ausgegeben werden. Strukturieren Sie Ihre Ausgaben übersichtlich und geben Sie aus, welches Konto gemeint ist.

- Schreiben Sie dem Sparkonto 9 € gut und zahlen Sie nacheinander einmal 2 € und und anschließend 20€ aus.
- Schreiben Sie dem Girokonto 100 € gut. Zahlen Sie nun 2000 € aus.
- Schreiben Sie dem Girokonto 3000 € gut und überweisen Sie einmal 2000 € auf das Sparkonto und einmal 2000 € auf das Tagesgeldkonto
- Buchen Sie vom Tagesgeldkonto 2000 € auf das Referenzkonto

_____ (5)

e) Stellen Sie sich folgendes Szenario vor: In einer Kontenverwaltung werden alle Konten einer Bank in einer Liste geführt, die Zeiger auf Objekte vom Typ `Bankkonto` enthält. Nun sollen für einen Statusbericht alle Kontoinformationen zu einem Report zusammengefasst werden. Dazu wird für jedes Konto die Methode `anzeigen()` aufgerufen.
Warum ist hierfür eine dynamische Bindung notwendig?

_____

_____

_____

Ändern Sie Ihren Code so ab, dass dynamische Bindung/Polymorphie voll unterstützt wird.

_____ (2)

f) Erweitern Sie Ihre `main`-Funktion indem Sie eine Liste (`eidpliste.h` von Blatt 10) mit Zeigern auf Bankkonten anlegen. Fügen Sie die Konten aus Aufgabe **d)** zur Liste hinzu. Durchlaufen Sie die Liste und geben Sie die Konteninformationen aus.

_____ (3)