

Andre Droschinsky
Roman Kalkreuth
Denis Kurz
Bernd Zey

Dortmund, den 17. Januar 2019

Praktikum zur Vorlesung Einführung in die Programmierung WS 18/19

Blatt 11

Es können 30 Punkte erreicht werden.

Aufgabe 1: 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:

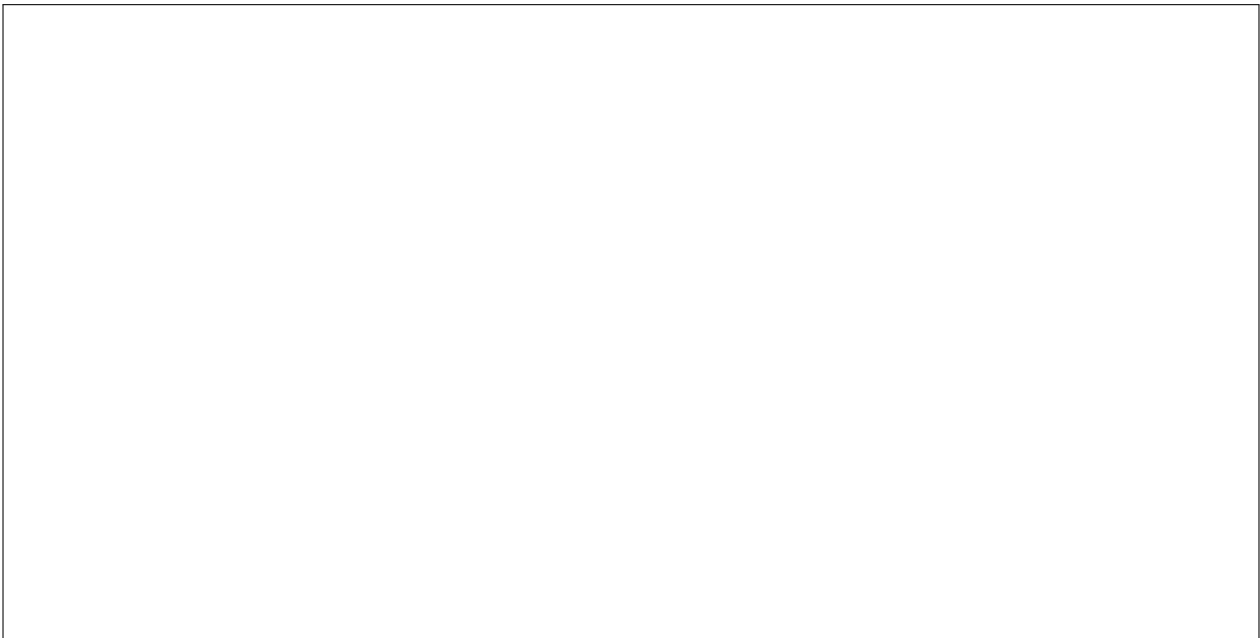
- `void gutschreiben(int betrag)`: Gutschreiben eines positiven Betrags auf das Konto. Falls der Betrag nicht positiv ist, soll eine Fehlermeldung ausgegeben und der Kontostand nicht verändert werden.
- `int kontostand() const`: Rückgabe des aktuellen Kontostands.
- `void anzeigen() const`: Ausgabe aller 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 auflösen()`, 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 Konten 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 Überweisungen (Umbuchungen) nur auf ein, bei der Eröffnung festgelegtes, Referenzkonto vom Typ *Girokonto* möglich (Methode `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. Überlegen Sie sich, ob es Eigenschaften gibt, die nur zwei der drei Kontoarten gemeinsam haben; Fügen Sie weitere Klassen zur Klassenhierarchie hinzu, um Redundanzen zu reduzieren.

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



_____ (5)

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).

_____ (3)

c) Implementieren Sie alle Methoden und Konstruktoren in den entsprechenden Quelldateien. Achten Sie darauf, dass die Parameter der Konstruktoren und Methodennamen kompatibel zur bereitgestellten Testumgebung sind (`konten_test.cpp`).

_____ (3)

d) 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)

e) Testen Sie Ihre Implementierung mit der bereitgestellten Testumgebung **konten_test.cpp**.

_____ (2)

Aufgabe 2: E-Books (15 Punkte)

In dieser Aufgabe soll eine Klassenhierarchie zur Beschreibung von E-Books (oder anderen Schriftstücken) erstellt werden. Die Basisklasse soll die Klasse **Buchbestandteil** darstellen; hiervon abgeleitet sind die Klassen **Text**, **Abbildung**, **Tabelle** und **Kapitel**. Es werden nur die zu implementierenden Methoden vorgegeben. Überlegen Sie sich, welche Attribute je Klasse zur Implementierung notwendig sind.

EINSCHRÄNKUNG: Implementieren Sie alle Methoden außerhalb der Klassendefinitionen in den entsprechenden Quelldateien (*.cpp). Machen Sie von den Spezifikationen **protected**, **private** und **public** sinnvoll Gebrauch.

a) Legen Sie ein neues Projekt an und deklarieren und implementieren Sie die abstrakte Basisklasse **Buchbestandteil**. Die Klasse soll folgende Methoden besitzen:

- **virtual unsigned int getAnzahlZeichen() const:**
Gibt die Anzahl der Zeichen in dem enthaltenen Text zurück.
- **virtual unsigned int getAnzahlAbbildungen() const:**
Gibt die Anzahl der enthaltenen Abbildungen zurück.
- **virtual unsigned int getAnzahlTabellen() const:**
Gibt die Anzahl der enthaltenen Tabellen zurück.
- **unsigned int getAnzahlFloats() const:**
Diese Methode soll die Summe der Abbildungen und Tabellen zurückgeben.

Die drei erstgenannten Methoden sollen rein virtuell sein.

_____ (3)

b) Deklarieren und implementieren Sie die von **Buchbestandteil** abgeleiteten Klassen **Abbildung** und **Tabelle**. Die Methoden dieser beiden Klassen sollen jeweils die korrekte Anzahl an Tabellen, Abbildungen und Zeichen zurückgeben (es ist immer nur der zugehörige Wert 1, alle anderen Werte 0).

_____ (3)

c) Deklarieren und implementieren Sie die von **Buchbestandteil** abgeleitete Klasse **Text**. Diese Klasse soll ein Attribut **inhalt** vom Typ **string** haben. In der Methode **getAnzahlZeichen()** sollen die Zeichen gezählt und zurückgegeben werden. Der **inhalt** wird per Konstruktor übergeben.

_____ (3)

d) Deklarieren und implementieren Sie die von **Buchbestandteil** abgeleitete Klasse **Kapitel**. Diese Klasse soll ein Attribut **Liste<Buchbestandteil*> bestandteile** besitzen, welches die Bestandteile dieses Kapitels enthält. Die Bestandteile sollen per Konstruktor übergeben werden können. Die beschriebenen **get**-Methoden dieser Klasse sollen die Summe der jeweiligen Werte der Bestandteile zurückgeben.

_____ (3)

e) Warum ist es in der Klasse **Kapitel** nicht notwendig, im Destruktor den Speicher freizugeben?

_____ (1)

f) Testen Sie Ihre Implementierung mit der bereitgestellten Testumgebung **ebook_test.cpp**.

_____ (2)