Klassenattribute und -methoden, Vererbung

Michael Dienert

4. Februar 2014

1 Prüfungsaufgabe Anwendungsentwicklung Winter 2001

Die folgende Aufgabe stammt aus der Abschlussprüfung für Fachinformatiker Anwendungsentwicklung in Baden-Württemberg im Winter 2001. Die Aufgabe wurde leicht modifiziert. Z.B. wurden die Namen der Eigenschaften geändert und die Fragestellung im zweiten Teil exakter formuliert.

1.1 modifizierter Aufgabentext

Ihr Unternehmen hat die Aufgabe eine Kontoführungssoftware zu entwickeln.

1. Zu diesem Zweck hat ein Kollege im Vorfeld ein UML-Diagramm entworfen (Abb. 1).

   Setzen Sie dieses UML-Diagramm in Quellcode um.

   - Die Girokonten erhalten Kontonummern grösser 100000
   - Die Methoden haben folgende Funktion:

Konstruktor	erhöht die Klassenvariable `nextKontoNummer` um 1, initialisiert das Objekt und weist ihm dabei eine Kontonummer zu
`setNextKontoNummer(long)`	setzt die nächste zu vergebende Kontonummer vor dem Anlegen eines Objekts fest
`getKontoNummer()`	gibt die `kontoNummer` zurück
`saldo()`	gibt den aktuellen Kontostand zurück
`einzahlen(double)`	erhöht den Kontostand um eine Einzahlung

2. Das UML-Diagramm für ein Girokonto wurde erweitert: Abb. 2.

   Die zusätzlichen Methoden haben folgende Funktion:

`setZinssatz(double)`	legt den Zinssatz für alle Girokonten fest
`abheben(double)`	vermindert den Kontostand um eine Abhebung
`gebuehr(double)`	jedes Girokonto wird einmal im Monat mit einer Gebühr belastet
`zinsgutschrift()`	berechnet und schreibt den Zins für den abgelaufenen Monat gut; liefert den Gutschriftbetrag zurück

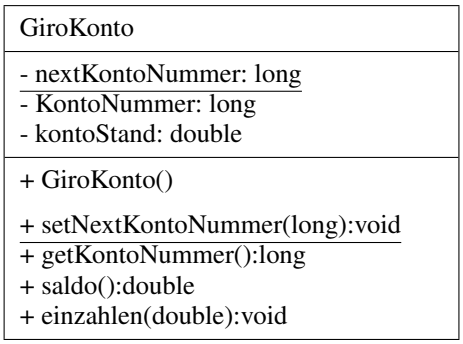
Die objektorientierte Analyse hat gezeigt, dass es neben den Girokonten auch Anlagekonten gibt.

Entwerfen Sie eine Vererbungshierarchie für die Klassen `GiroKonto` und `SparKonto`. Dabei sollen alle gemeinsamen Eigenschaften und Methoden dieser beiden Klassen von einer Superklasse `Konto` ererbt werden. [1]

Dabei ist für die Anlagekonten folgendes zu beachten:

- sie erhalten Kontonummern grösser 800000
- sie werden für eine bestimmte Laufzeit angelegt
- Anlagekonten können unterschiedliche Zinssätze haben
- dem Kunden wird drei Monate vor Ablauf eine Mitteilung zugesandt
- sie sind gebührenfrei

Die ursprüngliche Aufgabenstellung war mir zunächst völlig unklar. Erst ein Blick in die Musterlösung zeigte, dass hier als Lösung eine Vererbungsstruktur mit UML-Diagrammen erwartet wurde.



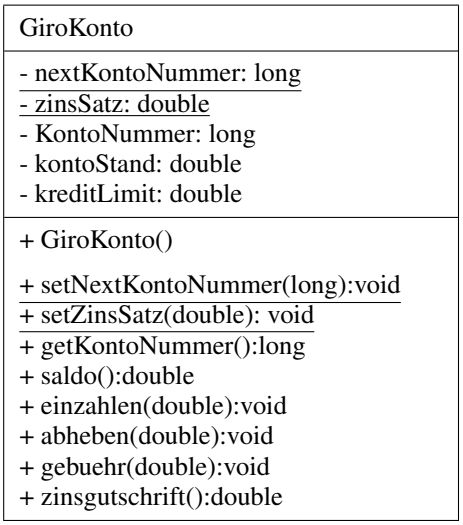


Abbildung 1: Klassendiagramm `GiroKonto` Abbildung 2: Erweiterte Klasse `GiroKonto`

---

2 Hinweise und Hilfen

Hier einige Hinweise zur Lösung der Aufgaben.

[1] Die originale Fragestellung war: Erweitern Sie das vorhandene UML-Diagramm und gestalten Sie es um, so dass auch die Anlagekonten abgebildet werden.

In den UML-Diagrammen sind einige Eigenschaften und Methoden unterstrichen. Was bedeutet das?

Unterstrichene Eigenschaften sind sogenannte *Klassenattribute*. Ein Klassenattribut beschreibt eine Eigenschaft *der Klasse*. Ein Klassenattribut wird immer dann verwendet, wenn der Wert dieser Eigenschaft (Attribut = Eigenschaft) für *alle* Objekte der Klasse der selbe ist. Ein typisches Beispiel für eine Klassenvariable wäre die Anzahl der erzeugten Objekte. In der Kontoaufgabe ist es die Eigenschaft `nextKontoNummer`, die ja auch die Anzahl der erzeugten Konten repräsentiert. Auch der Zinssatz, der für alle Girokonten der selbe sein soll, ist also ein Klassenattribut.

Klassenattribute werden mit dem Schlüsselwort `static` deklariert.

Analog zu den Klassenattributen, sind unterstrichene Methoden *Klassenmethoden*. Mit Klassenmethoden wird auf Klassenattribute zugegriffen. Eine Klassenmethode kann aufgerufen werden, bevor überhaupt Objekte der Klasse erzeugt worden sind. Damit dies möglich wird, werden die Klassenmethoden während der Laufzeit eines Java-Programms *sofort in den Speicher geladen* und verbleiben dort.

Die bekannteste statische Methode ist die `main`-Methode. Sie wird beim Start eines jeden Java-Programms in den Speicher geladen und dann aufgerufen.

Weitere wichtige statische Methoden sind z.B. alle Methoden der java.lang.Math - Klasse. Beispiel: Math.sqrt(2) berechnet die Quadratwurzel von 2.

3 Weitere Aufgaben

In der Prüfung nicht verlangt waren Testklassen, mit denen man Konto-Objekte erzeugt und testet. In dieser Übung sollen aber solche Testklassen erzeugt werden:

1. Um die Klasse GiroKonto zu testen, sollen Sie eine Klasse `GiroKontoTest` schreiben, in der ein GiroKonto-Objekt erzeugt wird.
2. In `GiroKontoTest` sollen alle Methoden von GiroKonto getestet werden.
3. Um die erweiterten Konto Klassen zu testen, soll das Programm interaktiv auf Benutzereingaben reagieren. Dazu ist ein Text-Menue möglicher Benutzeraktionen auszugeben. Die Auswahl eines Menuepunkts soll durch Eingabe eines Kennbuchstabens erfolgen. Das kann etwa so aussehen:

```
Was moechten Sie tun:
   (gk)   Girokonto anlegen
   (ak)   Anlagenkonto anlegen
   (ez)   einzahlen
   (ab)   abheben
   (sa)   Kontostand abfragen
   (zi)   Zinsen gutschreiben
   (kl)   Kreditlimit erfragen
   (li)   alle Konten auflisten
   (x)    beenden
          ...
```

Das Einlesen von Konsoleneingaben in Java ist ein bisschen umständlich. Grund sind die strenge Objektorientierung und die Verwendung von Unicode.

Um Zeichenketten (String) einzulesen, geht man so vor:

```
BufferedReader lesePuffer;
lesePuffer = new BufferedReader(
                new InputStreamReader(
                  System.in
                )
             )
String eingabe = lesePuffer.readLine();
int zahl = Integer.parseInt(eingabe);
```

Dabei haben die einzelnen Klassen folgende Funktionen:

- `System.in` ein Objekt, das die Standardeingabe (Tastatur) liest und einzelne Bytes liefert.

  `in` ist eine Objekt vom Typ `InputStream`. Die Klasse `System` definiert mit ihren Eigenschaften und Methoden eine plattformunabhängige Schnittstelle zum jeweiligen Betriebssystem.

- `InputStreamReader`: Objekte dieser Klasse lesen einzelne Bytes und codieren diese in Unicode.

- `BufferedReader`: ein `BufferedReader`-Objekt liest einzelne Unicode-Zeichen und legt diese in einem Pufferspeicher (*last in first out*, Stapelspeicher) ab.

  `BufferedReader`-Objekte besitzen die Methode `readLine()`, mit der eine ganze Textzeile aus dem Puffer ausgelesen und als String zurückgegeben wird.

Das heisst, um einen Eingabestring zu lesen, ruft man

```
String eingabe = lesePuffer.readLine();
```

auf. Möchte man numerische Eingaben entgegennehmen, kann man den String mit folgenden Methoden wandeln:

```
integer n = Integer.parseInt(eingabe);
long m = Long.parseLong(eingabe);
double x = Double.parseDouble(eingabe);
```

Diese parse-Methoden gibt es erst ab Java2 (ab jdk1.2). Da bei der Umwandlung eines Strings in einen numerischen Wert ein Ausnahmefehler auftreten kann - der String könnte ja z.B. Buchstaben enthalten - *müssen* die parse-Methoden innerhalb eines try-Blocks stehen. Beispiel:

```
try{
    BufferedReader lesePuffer = new BufferedReader(
                                    new InputStreamReader(
                                       System.in
                                    )
                                );

    System.out.println("Geben Sie eine ganze Zahl ein!");
    String eingabe = lesePuffer.readLine();
    int zahl = Integer.parseInt(eingabe);  //falls Ihr String einen
                                           //Dezimalpunkt enthaelt,
                                           //wird die Ausnahmebehandlung
                                           //ausgeloest
                          ...

} // ende des try-blocks

catch(Exception e){
   System.out.println("Es trat der Fehler " + e + "auf!");
}
```

Um eine ganze Liste von Kontoobjekten anzulegen, muss man ein *Array* von Objekten deklarieren. Das macht man mit folgender Zeile:

```
//anlegen eines feldes mit 100 konten
//kontoNummer darf nur aus diesem Bereich stammen

GiroKonto[] einGiroKonto = new GiroKonto[100];
```

Diese Zeile ist gleichwertig mit folgender Konstruktion:

```
GiroKonto einGiroKonto[0];
GiroKonto einGiroKonto[1];
          ...
GiroKonto einGiroKonto[99];
```

Also Vorsicht: obwohl oben der `new`-Operator verwendet wird, sind keinerlei Objekte erzeugt worden! Man hat lediglich 100 Objekte referenziert, also deren Namen als Zeiger auf die eigentlichen Objekte festgelegt.

Die Objekte müssen in einem zweiten Schritt in gewohnter Weise angelegt werden. Also z.B. so:

```
einGiroKonto[42] = new GiroKonto();
```

Statt der Zahl 42 kann natürlich auch eine int-Variable stehen.

Und ganz wichtig bei Objektarrays: runde und eckige Klammern nicht verwechseln!
