

## Übungsblatt 6 - Abgabe vor der Vorlesung 29.11.05

### Aufgabe 0 (Wiederholung Vererbung) [Optional]

- Vererbung in Haskell (Typklassen, Eq, Ord, Read)
- Generizität:
  - Syntax und Sinn
  - Unterschied parametrisierter Typ / generischer Typ
  - Autoboxing/Unboxing
  - Typverträglichkeit
  - Vererbung

### Aufgabe 1 (Vererbung)

Wir wollen diesmal eine Filiale der Bank aus der Vorlesung implementieren. Aus diesem Grund hier eine Auflistung der Anforderungen:

Eine Bankfiliale gehört zu einer Bank und bietet den Kunden Dienste für deren Wünsche an. Diese Wünsche umfassen: Geld abheben, Geld einzahlen, Konto eröffnen, beraten werden. Die Dienste der Bank können an verschiedenen Schaltern und Automaten der Bank in Anspruch genommen werden: Am Geldautomaten kann man Geld abheben, am Beratungsschaltern kann man ein Konto eröffnen und sich beraten lassen und an der Kasse kann man Geld einzahlen und auszahlen. Natürlich kann es an allen Schaltern und Automaten zu Schlangen kommen. Die Beratungsschalter haben hierbei eine gemeinsame Schlange, die Automaten und Kasse jeweils eigene Schlangen. Tritt bei einer Operation ein Fehler auf, so soll der Kunde ohne den Zustand der Bank zu ändern die Bank wieder verlassen.

Hier ein Testlauf, den eure Implementierung bestehen sollte:

```
Bank b = new Bank();
Filiale f = new Filiale("Dahlem", b);
assertTrue(b.filialen.contains(f));
Beratungsschalter b1 = new Beratungsschalter("Herr Kaiser", f);
Beratungsschalter b2 = new Beratungsschalter("Frau Maier", f);
assertEquals(2, f.beratungsschalter.size());
Geldautomat g1 = new Geldautomat("Eingangshalle", f);
Geldautomat g2 = new Geldautomat("Außenautomat", f);
Kasse k = new Kasse("Mrs. Money Penny", f);
Person p = new Person("Fritz Müller");
p.setWunsch(Wunsch.KONTOEROEFFNEN);
p.besuche(f);
b1.rufeNaechsten();
assertTrue(p.konto == null);
b1.bearbeite();
assertTrue(p.konto != null && p.konto.getKontostand() == 0);
assertTrue(p.getWunsch() == Wunsch.WUNSCHLOS);
assertEquals(0, b1.getSchlange().length());
```

```

assertEquals(0, f.kundenInFiliale());
assertEquals("Dahlem - Fritz Müller möchte ein Konto
eröffnen.\nDahlem - Herr Kaiser - Konto Eröffnung Fritz Müller.\n",
b.getProtokoll());

```

Abzugeben sind Quelltext und Tests (mindestens 5 Kunden gleichzeitig und ein Protokoll mit mehr als 20 Zeilen).

### Tipps:

- Wünsche (als Aufzählungstyp) lassen sich gut mit Enums (Java 1.5 Folge 2) implementieren.
- Eine geeignete Datenstruktur, die dynamisch mitwächst ist Vector, aus java.util (diese besitzt auch die Methoden size() und contains()).
- Nutzen Sie Vererbung um Funktionalität wiederzuverwenden, die in mehreren Klassen benötigt wird.
- Datenstrukturen sollten ab jetzt immer in ihren generischen Formen verwendet werden (d.h. z.B. Vector<String> und nicht mehr nur Vector).

Die Implementierungen von Schlange, Bank und Konto dürfen Sie natürlich von den Folien und aus vorangehenden Übungen übernehmen.

Wer zu schnell fertig wird, kann noch folgende Anforderungen in Angriff nehmen [optional]:

- Die Zuordnung zwischen Kunde und Konto sollte nicht über einen direkten Verweis in der Klasse Kunde, sondern durch ein geeignetes Schlüsselattribut (z.B. die Kontonummer) erfolgen.
- Sparkonten sollten getrennt von Girokonten zur Verfügung stehen.
- Die Geldautomaten sollten nur über eine EC-Karte und eine PIN zugänglich sein.
- Es wäre schön, wenn man den aktuellen Zustand der Bank graphisch dargestellt bekommen könnte. Dies erfordert natürlich eine Einarbeitung zum Thema Swing (Java GUI Bibliothek).

### Aufgabe 2 (Vergleichen)

Vergessen Sie, dass es in Java equals und Comparable gibt! In Haskell gibt es die vorbildlich formulierten Eq und Ord (Haskell Report 6.3.1/2 bzw. Haskell Prelude).

- a) Bilden Sie Eq und Ord so gut wie möglich als abstrakte Java-Klassen nach!

Die Lösung beginnt so:

```

abstract class Eq<A> {
    boolean equal(A a) {return !unequal(a); }
    ...
}

```

- b) Formulieren Sie - abgestützt auf Eq und Ord - eine Klasse Cardinal, die als Hüllklasse für natürliche Zahlen dient!

Hinweise:

- Verwenden Sie den Aufzählungstypen enum, um den Rückgabewert von compare darzustellen:

```

public enum Ordering { EQ, LT, GT }

```

- Wie bei equal angedeutet, sollen die Funktionen objektorientiert ausgelegt sein, also als Methoden, die auf einem Objekt aufgerufen werden, nicht als statische Methoden mit zwei Argumenten.

### **Java Wiederholung - Folge 6**

Wieder gibt es unter <http://projects.mi.fu-berlin.de/w/bin/view/SS/JavaWiederholung> neue Java-Aufgaben, welche mittwochs in Raum 055 um 16ct besprochen werden.