Mehr über Vererbung

Lernziele

- Sie können Unsauberkeiten bei einer gegebenen Vererbungshierarchie erkennen und korrigieren.
- Sie können für eine einfache Problemstellung eine geeignete Vererbungshierarchie konzipieren und umsetzen und wenden dabei das Überschreiben von Methoden und Polymorphie korrekt an.

Aufgabe 1 (auf Papier)

Auf den nächsten beiden Seiten ist eine Vererbungshierarchie mit vier Klassen für verschiedene Tickets angegeben. Das Programm funktioniert bestens und wenn folgende Codezeilen aufgerufen werden

```
System.out.println(new Ticket("Mister T."));
System.out.println(new GratisTicket("James Bond", "Universal Studios"));
System.out.println(new BezahltesTicket("Robin Hood", 12));
System.out.println(new GruppenTicket("Superman", 38, 4));
```

erfolgt folgende Ausgabe:

```
Ausgestellt auf Mister T.
Ausgestellt auf James Bond, gesponsort von Universal Studios
Ausgestellt auf Robin Hood, Preis: CHF 12
Ausgestellt auf Superman, Gruppengroesse: 4, Gesamtpreis: CHF 38
```

Die Klassen enthalten aber einige Design-Unschönheiten – korrigieren Sie diese. Die Ausgabe muss dabei in der genau gleichen Art erfolgen wie oben angegeben.

```
/**
 * Ein Ticket hat einen Namen, auf den das Ticket ausgestellt ist.
 */
public class Ticket
{
  protected String name;


  public Ticket(String name)
  {
    this.name = name;
  }


  public String toString()
  {
    return "Ausgestellt auf " + name;
  }
}

/**
 * Ein Gratisticket hat zusaetzlich einen Sponsor.
 */
public class GratisTicket extends Ticket
{
  private String sponsor;


  public GratisTicket(String name, String sponsor)
  {
    super(name);
    this.sponsor = sponsor;
  }

  public String toString()
  {
    return "Ausgestellt auf " + name + ", gesponsort von " + sponsor;
  }
}



/**
 * Ein BezahltesTicket hat zusaetzlich einen Preis.
 */
public class BezahltesTicket extends Ticket
{
  protected int preis;
```

```
  public BezahltesTicket(String name, int preis)
  {
    super(name);
    this.preis = preis;
  }

  @Override
  public String toString()
  { return super.toString() + ", Preis : CHF " + getPreis();
    return "Ausgestellt auf " + name + ", Preis: CHF " + preis;
  }
}



/**
 * Ein GruppenTicket hat zusaetzlich eine Anzahl Personen.
 */
public class GruppenTicket extends BezahltesTicket
{
  private int gruppenGroesse;


  public GruppenTicket(String name, int preis, int gruppenGroesse)
  {
    super(name, preis);
    this.gruppenGroesse = gruppenGroesse;
  }

  @Override
  public String toString()
  {                                  getName()
    return "Ausgestellt auf " + name + ", Gruppengroesse: " +
           gruppenGroesse + ", Gesamtpreis: CHF " + preis;
  }                                                  getPreis()
}
```

Aufgabe 2

Forken Sie für diese Aufgabe das (leere) Projekt https://github.engineering.zhaw.ch/prog1-kurs/09_Praktikum_Bank. Nutzen Sie Eclipse um die eigene Projektkopie auf Ihren Computer zu holen und zu bearbeiten.

Eine Bank bietet drei Arten von Konten an: „normale“ Bankkonten, Salärkonten und Nummernkonten. Die Eigenschaften sind nachfolgend beschrieben:

- Jedes Konto hat einen Inhaber (ein String) und einen aktuellen Kontostand.
- Es darf höchstens ein Maximalbetrag von CHF 100'000 auf einem Konto sein.
- Das Salärkonto darf als einziges überzogen werden. Dazu hat es eine Überzugslimite, welche im Bereich 0 bis maximal CHF 10'000 liegt.
- Es kann Geld auf ein Konto einbezahlt werden. Würde der Kontostand dadurch den Maximalbetrag überschreiten, wird nur soviel einbezahlt, wie noch erlaubt ist.
- Es kann Geld von einem Konto abgehoben werden. Der Kontostand darf dadurch nicht negativ bzw. beim Salärkonto nicht unter die Überzugslimite fallen. Falls dies so wäre, so wird nur die erlaubte Geldmenge abgehoben.
- Ein Nummernkonto hat nebst dem Inhaber zusätzlich eine Nummer. Diese beginnt für das erste Nummernkonto bei 1000 und wird für weitere Nummernkonten fortlaufend um 1 erhöht (1001, 1002...).
- Bei allen Konten kann der Inhaber und der Kontostand abgefragt werden. Beim Nummernkonto wird bei der Abfrage des Inhabers nicht der Name, sondern die Nummer geliefert. Bei Salärkonten kann man auch die Überzugslimite abfragen.
- Beim Erzeugen eines Kontos wird der Inhaber angegeben und optional eine Ersteinlage (Kontostand zu Beginn). Bei Salärkonten wird zudem die Überzugslimite spezifiziert.

Ihre Aufgabe ist es, die angebotenen Konten unter Berücksichtigung der Anforderungen in einer geeigneten Vererbungshierarchie umzusetzen. Schreiben Sie zudem eine Klasse, welche eine Bank simuliert. Dabei sollen mehrere Konten der verschiedenen Typen erzeugt werden, Geld einbezahlt und abgehoben werden und die Konten sollen ausgegeben werden. Verwalten Sie alle erzeugten Konten in derselben Datenstruktur. Überprüfen Sie insbesondere auch, ob die Limiten korrekt eingehalten werden. Die Ausgabe soll in etwa wie folgt aussehen (jeweils für ein Bankkonto, ein Salärkonto und ein Nummernkonto):

```
Inhaber: Jolly Jumper, Kontostand: 30000.0

Inhaber: Donald Duck, Kontostand: -2000.0, Ueberzugslimite: 5000.0

Inhaber: 1000, Kontostand: 100000.0
```

Hinweis zur Umsetzung:

- Verwalten Sie Konto-intern den Kontostand am besten in Rappen (int). Damit vermeiden Sie allfällige Präzisionsprobleme mit Gleitkommazahlen. Die Methoden und Konstruktoren sollen aber bei den Parametern und Rückgabewerten Beträge in Franken verwenden. Um effizient zwischen Franken und Rappen zu konvertieren, verwenden Sie am besten zwei Konto-interne Methoden.

Aufgabe 3 (optional)

Erweitern Sie Aufgabe 2 um folgendes:

- Überschreiben Sie die Methode `toString`, damit ein Konto Informationen über sich selbst liefert, ähnlich wie die Ausgabe in Aufgabe 2.
- Fügen Sie JUnit-Tests hinzu, um die Konten zu testen.