Übungen II: Objektorientierte Programmierung

Aufgaben

Im Folgenden wird eine ganzheitliche Aufgabe zur Entwicklung einer objektorientierten Lösung für eine einfache Lohn- und Gehaltsabrechnung gestellt. Die einzelnen Aufgaben bauen aufeinander auf und sollten daher sukzessiv bearbeitet werden.

1. Definieren Sie eine Klasse `Mitarbeiter`. Ein Mitarbeiter besitzt eine eindeutige Nummer (`id`) und einen Namen (`name`). Die ID des Mitarbeiters soll durch die Klasse selbst fortlaufend nummeriert werden. Der erste Mitarbeiter hat also die ID 1, der zweite Mitarbeiter die ID 2 usw. Fügen Sie der Klasse sinnvolle `get`- und `set`-Methoden hinzu und definieren Sie zudem eine `toString()`-Methode, die die ID und den Mitarbeiternamen als String zurückliefert.
2. Schreiben Sie eine Klasse `PersonalVerwaltung`. Diese Klasse hat eine Mitarbeiterliste (`mitarbeiterListe`, Typ: `ArrayList <Mitarbeiter>`). Sie hält Methoden zum Hinzufügen und zum Entfernen von Mitarbeitern bereit. Außerdem benötigt sie eine Methode `listMitarbeiter()`, um alle Mitarbeiter auf der Konsole aufzulisten.
3. Fügen Sie der Klasse `PersonalVerwaltung` eine Methode `sortMitarbeiter()` hinzu. Diese Methode soll die Mitarbeiter mittels Bubblesort (s. Abschnitt *2.4. Arrays*, S. 32) sortieren. Zu diesem Zweck muss in der Klasse `Mitarbeiter` eine Methode `boolean istKleiner(Mitarbeiter m)` hinzugefügt werden. Sie ist von Bubblesort zu verwenden, um die Rangfolge unter den Mitarbeitern zu erkennen. Die `istKleiner()`-Methode soll dazu führen, dass die Mitarbeiter alphabetisch nach ihren Namen sortiert werden.

4. Implementieren Sie die abstrakte Klasse `Abrechnung` und ihre beiden Unterklassen `LohnAbrechnung` und `GehaltsAbrechnung` nach folgendem Grundriss:

```
public abstract class Abrechnung {

   private int periode;
   private Mitarbeiter mitarbeiter;

   public Abrechnung(int periode, Mitarbeiter m)
                                              { ... }
   public int getPeriode() { ... }
   public Mitarbeiter getMitarbeiter() { ... }
   public abstract double getVerdienst();
   public String toString() { ... }
}

public class GehaltsAbrechnung extends Abrechnung {

   private double gehalt;

   public GehaltsAbrechnung(int periode,
                            Mitarbeiter m,
                            double gehalt) { ... }
   public double getVerdienst() { ... }
}

public class LohnAbrechnung extends Abrechnung {

   private double stundenLohn;
   private double anzahlStunden;

   public LohnAbrechnung(int periode,
                         Mitarbeiter m,
                         double stundenlohn,
                         int stunden) { ... }
   public double getVerdienst() { ... }
}
```

Sowohl Lohn- als auch Gehaltsabrechnung erfolgen in einer Abrechnungsperiode (in der Regel eine fortlaufend durchnummerierte Periodenummer) und referenzieren einen Mitarbeiter. Die abstrakte Methode `getVerdienst()` in der Klasse `Abrechnung` gibt in dem konkreten Fall den Verdienst eines Mitarbeiters in der entsprechenden Periode zurück. Bei einer Gehaltsabrechnung ist dies das Gehalt, bei einer Lohnabrechnung ist es das Produkt aus Stundenlohn und

Anzahl der geleisteten Stunden. Die `toString()`-Methode in Abrechnung soll die Periodennummer, den Namen des Mitarbeiters und den Verdienst als String zurückgeben (Hinweis: Verwenden Sie für letzteres die `getVerdienst()`-Methode)

5. Erweitern Sie die Klasse `PersonalVerwaltung` dahingehend, dass analog zu den Mitarbeitern auch Abrechnungen hinzugefügt und entfernt werden können, und schreiben Sie eine Methode `listAbrechnungen()`, welche alle Abrechnungen einer bestimmten Abrechnungsperiode auf der Konsole ausgibt.

6. Java bietet zum Sortieren die statische Methode `Collections.sort()`. Verwenden Sie diese zum Sortieren der Mitarbeiterliste, sodass Sie auf Ihre eigene Bubblesort-Implementierung verzichten können. Damit dies funktioniert, muss die Klasse `Mitarbeiter` das generische Interface `Comparable<Mitarbeiter>` implementieren. Es ist demnach eine Methode `int compareTo(Mitarbeiter m)` erforderlich, deren Rückgabewert sich im Prinzip verhält wie die `compareTo()`-Methode der Klasse `String` (siehe Exkurs *Zeichenketten*, S. 82). Arbeiten Sie ggf. mit Hilfe der Java-Dokumentation. Im Anschluss können Sie die `istKleiner()`-Methode löschen, da sie quasi durch die `compareTo()`-Methode ersetzt wird.

   Sie können am Ende folgendes Testprogramm verwenden:

```
public static void main(String[] args) {

PersonalVerwaltung pv = new PersonalVerwaltung();
Mitarbeiter m1 = new Mitarbeiter("Josef Maier");
pv.addMitarbeiter(m1);
Mitarbeiter m2 = new Mitarbeiter("Franz Huber");
pv.addMitarbeiter(m2);
Mitarbeiter m3 = new Mitarbeiter("Werner Müller");
pv.addMitarbeiter(m3);

pv.sortMitarbeiter();
pv.listMitarbeiter();

pv.addAbrechnung(new LohnAbrechnung(1,m1,10,158));
pv.addAbrechnung(new GehaltsAbrechnung(1,m2,3010));
pv.addAbrechnung(new GehaltsAbrechnung(1,m3,2700));
```

```
pv.addAbrechnung(new LohnAbrechnung(2,m1,16,158));
pv.addAbrechnung(new GehaltsAbrechnung(2,m2,3010));
pv.addAbrechnung(new GehaltsAbrechnung(2,m3,2800));
pv.listAbrechnungen(2);
}
```

und sollten dann in etwa diese Ausgabe auf der Konsole erhalten:

```
Mitarbeiter
2, Franz Huber
1, Josef Maier
3, Werner Müller

Abrechnungen
2, Josef Maier, 2528.0
2, Franz Huber, 3010.0
2, Werner Müller, 2800.0
```