Einstieg in Java und OOP

Christian Silberbauer

Übungsblatt 3

Im Folgenden wird eine ganzheitliche Aufgabe zur Entwicklung einer objektorientierten Lösung für eine einfache Lohn- und Gehaltsabrechnung gestellt.

Aufgabe 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. Implementieren Sie einen Konstruktor, der als Parameter den Mitarbeiternamen verlangt.

Aufgabe 2

Schreiben Sie eine Klasse `MitarbeiterListe` zur Verwaltung von `Mitarbeiter`-Objekten. Intern verwaltet die Klasse ihre Mitarbeiter in einem Array, dessen Größe per Konstruktorparameter festgelegt wird. Die Klasse soll folgende Methoden implementieren:

- `int getSize()`: Gibt die Anzahl der aktuell verwalteten `Mitarbeiter` zurück
- `boolean add(Mitarbeiter m)`: Fügt einen `Mitarbeiter` hinzu; wenn kein Platz mehr ist, wird `false` zurückgegeben
- `Mitarbeiter get(int index)`: Gibt den `Mitarbeiter` an einer bestimmten Position zurück; bei unbekanntem `index` wird `null` zurückgegeben
- `Mitarbeiter set(int index, Mitarbeiter m)`: Setzt einen `Mitarbeiter` an eine bereits belegte Position; bei unbekanntem `index` wird `null` zurückgegeben, ansonsten der bisherige Wert
- `boolean remove(Mitarbeiter m)`: Entfernt einen `Mitarbeiter`; gibt `false` zurück, wenn der `Mitarbeiter` nicht existiert

Aufgabe 3

Schreiben Sie eine Klasse `PersonalVerwaltung`. Diese Klasse hat eine `MitarbeiterListe`. Sie hält Methoden zum Hinzufügen und zum Entfernen von

Mitarbeitern bereit. Außerdem benötigt sie eine Methode `listMitarbeiter()`, um alle Mitarbeiter auf der Konsole aufzulisten. Sie können zum Testen der Anwendung der Klasse `PersonalVerwaltung` folgende `main()`-Funktion hinzufügen:

```
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.listMitarbeiter();
}
```

Dies sollte zu folgender Ausgabe führen:

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

Aufgabe 4

Fügen Sie der Klasse `PersonalVerwaltung` eine Methode `sortMitarbeiter()` hinzu. Diese Methode soll die Mitarbeiter mittels Bubblesort (siehe Foliensatz des ersten Semesters, *8. Felder*, S. 17) 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. Sie können nun Ihrer `main()`-Funktion wiederum folgende zwei Zeilen hinzufügen:

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

Dies sollte schließlich zu folgender Ausgabe führen:

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

Aufgabe 5

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,
                              double 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)

Aufgabe 6

Erweitern Sie die Klasse `PersonalVerwaltung` dahingehend, dass analog zu den Mitarbeitern auch Abrechnungen hinzugefügt und entfernt werden können. Definieren Sie dafür eine Klasse `AbrechnungenListe`. schreiben Sie zudem eine Methode `listAbrechnungen()` in `PersonalVerwaltung`, welche alle Abrechnungen einer bestimmten Abrechnungsperiode auf der Konsole ausgibt. Wenn Sie Ihrer `main()`-Funktion nun folgenden Code hinzufügen:

```
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);
```

Sollten Sie folgende Ausgabe erhalten:

```
Abrechnungen
2, Josef Maier, 2528.0
2, Franz Huber, 3010.0
```

2, Werner Müller, 2800.0