

Einführung in die Programmierung
WS 2012/13

Übungsblatt 12: Typparameter, Datenstrukturen

Besprechung: 04./06./08.01.2013

Ende der Abgabefrist: Dienstag, 29.01.2013 14:00 Uhr.

Hinweise:

Die behandelten Inhalte sind Stoff der Hauptklausur, obwohl die Besprechung erst nach der Klausur stattfinden wird. Deshalb wird die Lösung zu diesem Übungsblatt am Dienstag, den 29.01.2013 nach Ablauf der Abgabefrist online gestellt. Beachten Sie, dass die Besprechung des Blattes zeitlich verschoben stattfindet, also von Montag, 04.02.2013 bis Freitag 08.02.2013.

Aufgabe 12-1 *Typparameter*

0 Punkte

Die Klasse `MinMax` enthält zwei Methoden, die das Maximum bzw. das Minimum aus einer Menge von `Comparable`-Objekten bestimmen. Die Schreibweise `Comparable... cs` in der Parameter-Liste steht für ein oder mehrere Objekte vom Typ `Comparable`. Man kann also auf diese Weise eine Methode definieren, die eine beliebige, nicht-leere Menge von Objekten eines bestimmten Typs als Parameter erhält. Diese Schreibweise kann allerdings nur beim letzten Parameter einer Methode auftreten. Im Methodenrumpf kann der entsprechende Parameter behandelt werden wie ein Array, er kann also auch in Schleifen verwendet werden. Hier wird eine erweiterte Form der `for`-Schleife verwendet, die nützlich ist, wenn ein Array lediglich durchlaufen werden soll, ohne dass eine Verwendung des Schleifenindex nötig ist.

Die `main`-Methode enthält nach zwei richtigen Verwendungen der Methoden (für `Integer`- und `String`-Objekte) auch eine Verwendung, in der beide Objekt-Typen gemischt vorkommen. Dieser Fehler wird aber nicht beim Übersetzen erkannt. Korrigieren Sie die Klasse `MinMax`, so dass eine solche gemischte Verwendung bereits beim Übersetzen als Fehler erkannt wird. Damit das korrigierte Programm übersetzbar ist, müssen Sie dann natürlich die letzte Anweisung in der `main`-Methode auskommentieren. Begründen Sie Ihre Korrektur der Klasse.

Aufgabe 12-2 *Datenstrukturen*

3+4+5 Punkte

Gegeben folgende Klasse zur Modellierung einer sortierten Liste:

```
public class SortierteListe<E extends Comparable<E>> {
    private int size;
    private Entry<E> head;

    public SortierteListe() {
        this.head = null;
    }

    public E get(int index) {
        if(index < 0 || index >= this.size) {
            throw new IndexOutOfBoundsException("Index: " + index
```

```

        + ", Size: " + this.size);
    }
    Entry<E> currentEntry = this.head;
    while(index > 0) {
        currentEntry = currentEntry.getNext();
        index--;
    }
    return currentEntry.getElement();
}

public int size() {
    return this.size;
}

private static class Entry<E> {
    private E element;
    private Entry<E> next;

    public Entry(E o, Entry<E> next) {
        this.element = o;
        this.next = next;
    }

    public E getElement() {
        return this.element;
    }

    public void setElement(E element) {
        this.element = element;
    }

    public Entry<E> getNext() {
        return this.next;
    }

    public void setNext(Entry<E> next) {
        this.next = next;
    }
}
}

```

Die Liste ist aufsteigend gemäß der natürlichen Ordnung ihrer Elemente sortiert, d.h. für zwei Elemente der Liste a und b gilt: Aus $a.compareTo(b) < 0$ folgt, dass das Element a in der Liste vor dem Element b steht. Ergänzen Sie diese Klasse um folgende Methoden, wobei Sie die Eigenschaft der Sortierung nutzen, um unnötige Operationen zu sparen:

- Die Methode **public int** `indexOf(E element)` gibt den Index des ersten Vorkommens des gegebenen Elementes in der Liste zurück (beginnend mit 0 für das erste Element der Liste) oder -1 , falls das Element in der Liste nicht vorkommt.
- Die Methode **public void** `add(E element)` fügt der Liste das gegebene Element hinzu, wobei die Eigenschaft der Sortierung erhalten bleiben soll.
- Die Methode **public void** `slice(E min, E max)` entfernt aus der Liste alle Elemente, die kleiner als min oder größer als max sind.

Aufgabe 12-3 Generics**0 Punkte**

Gegeben sind folgende Klassen zur Modellierung von Krapfen mit Füllung:

```
1 public class Geschmacksrichtung {
2     public static final String[] GESCHMACKSRICHTUNGEN =
3         {"Erdbeere", "Himbeere", "Hagebutte"};
4
5     private String geschmacksrichtung;
6
7     public Geschmacksrichtung(int geschmacksrichtung) {
8         if(geschmacksrichtung < 0
9             || geschmacksrichtung >= GESCHMACKSRICHTUNGEN.length) {
10             throw new IllegalArgumentException("Ungueltige Geschmacksrichtung");
11         }
12         this.geschmacksrichtung = GESCHMACKSRICHTUNGEN[geschmacksrichtung];
13     }
14
15     public String getGeschmacksrichtung() {
16         return this.geschmacksrichtung;
17     }
18 }
```

```
1 public class Fuellung {}
```

```
1 public class Marmelade extends Fuellung {
2     private Geschmacksrichtung geschmacksrichtung;
3
4     public Marmelade(Geschmacksrichtung geschmacksrichtung) {
5         this.geschmacksrichtung = geschmacksrichtung;
6     }
7
8     public Geschmacksrichtung getGeschmacksrichtung() {
9         return this.geschmacksrichtung;
10    }
11 }
```

```
1 public class Senf extends Fuellung {}
```

```
1 public class Krapfen {
2     private Fuellung fuellung;
3
4     public void fuellen(Fuellung fuellung) {
5         this.fuellung = fuellung;
6     }
7
8     public Fuellung schmecktNach() {
9         return this.fuellung;
10    }
11 }
```

Diese Modellierung wird von einer Reihe von Klassen verwendet:

```
1 import java.util.ArrayList;
2 import java.util.List;
3
4 public class Konditor {
5     public List<Krapfen> krapfenMachen(int anzahl, Fuellung fuellung)
6     {
```

```

7   List<Krapfen> krapfenliste = new ArrayList<Krapfen>(anzahl);
8   for(int i = 0; i < anzahl; i++)
9   {
10      Krapfen krapfen = new Krapfen();
11      krapfen.fuellen(fuellung);
12      krapfenliste.add(krapfen);
13  }
14  return krapfenliste;
15  }
16  }

```

```

1  public class Kunde {
2      public Geschmacksrichtung krapfenEssen(Krapfen krapfen){
3          return ((Marmelade) krapfen.schmecktNach()).getGeschmacksrichtung();
4      }
5  }

```

```

1  import java.util.List;
2
3  public class Krapfenkauf{
4      public static void main(String[] args){
5          Konditor grattler = new Konditor();
6          List<Krapfen> krapfenliste = grattler.krapfenMachen(10, new Senf());
7          Kunde kunde = new Kunde();
8          for(Krapfen krapfen : krapfenliste)
9          {
10             if(kunde.krapfenEssen(krapfen)
11                 .getGeschmacksrichtung()
12                 .equalsIgnoreCase("ERDBEERE"))
13             {
14                 System.out.println("Mmmmmh!");
15             }
16         }
17     }
18 }

```

- (a) Bei Ablauf des Programmes `Krapfenkauf` wird ein Fehler auftreten. Um was für einen Fehler handelt es sich, warum und an welcher Stelle (Klassenname, Zeilennummer) tritt er auf?
- (b) Verbessern Sie die Modellierung von Krapfen mit Füllung durch Typisierung so, dass bei Ihrer Verwendung Typsicherheit zur Übersetzungszeit erreicht wird.
(Es genügt, wenn Sie hier nur die geänderten Zeilen unter Angabe von Klassenname und Zeilennummer eintragen.)
- (c) Passen Sie die Klassen `Konditor`, `Kunde` und `Krapfenkauf` dieser neuen Modellierung an. Ein Kunde soll danach grundsätzlich nur Krapfen essen, die mit Marmelade gefüllt sind. Ansonsten soll sich aber im Ablauf und Ergebnis des Programmes nichts ändern.
(Es genügt, wenn Sie hier nur die geänderten Zeilen unter Angabe von Klassenname und Zeilennummer eintragen.)
- (d) Der Fehler aus Teilaufgabe (a) bleibt grundsätzlich bestehen, es ist aber nun ein Fehler anderer Art. Um welche Art Fehler handelt es sich jetzt, wo und warum tritt er auf? Was ist dennoch mit der geänderten Modellierung gewonnen?