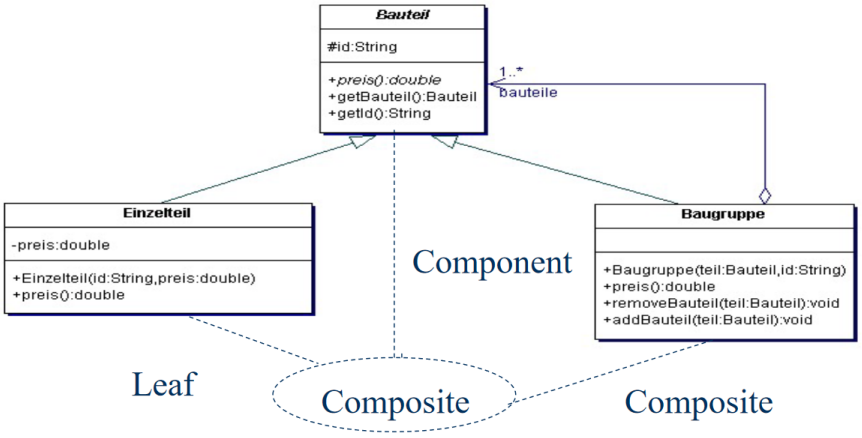
Softwaretechnologie, Übung 6

Henry Haustein

Aufgabe 1

(a) Composite

(b) Bauteil: Component, Baugruppe: Composite, Einzelteil: Leaf



(c) Datei `Bauteil.java`

```java
public abstract class Bauteil {
  protected String id;

  public abstract double preis();

  public String getId() {
    return id;
  }
}
```

Datei `Einzelteil.java`

```java
public class Einzelteil extends Bauteil {
  private double preis;

  public Einzelteil(String id, double preis) {
    this.preis = preis;
```

```
6       this.id = id;
7     }
8
9     @Override
10    public double preis() {
11      return preis;
12    }
13  }
```

Datei `Baugruppe.java`, gleich mit Verbesserungen

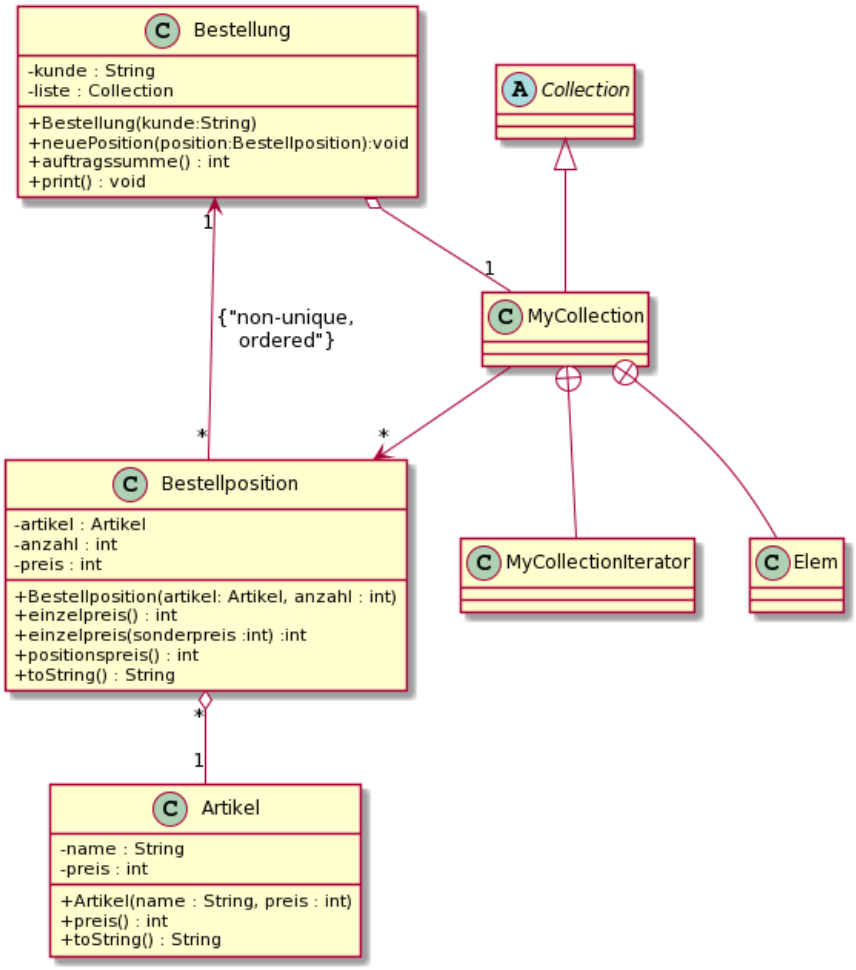
```
1   import java.util.Set;
2
3   public class Baugruppe extends Bauteil {
4     private Set<Bauteil> bauteile;
5
6     public Baugruppe (Bauteil teil, String id){
7       if(teil != null){
8         this.bauteile = new HashSet<Bauteil>();
9         this.id = id;
10      } else {
11        throw new Exception("Gib mir ein Bauteil !11!!11!1");
12      }
13    }
14
15    public double preis() {
16      double summe = 0;
17      for (Bauteil b : bauteile) {
18        summe += b.preis();
19      }
20      return summe;
21    }
22
23    // void Rueckgabe ist nicht so gut
24    // keine Info ueber Miss-/Erfolg des Loeschens
25    // void -> boolean
26    // Problem: was ist mit Bauteilen, die in Baugruppen
27    // versteckt sind? auch mit entfernen oder nicht? -> nur 1
28    // Ebene betrachten
29    // keine Rekursion
30    public boolean removeBauteil(Bauteil teil){
31      if (bauteile.size() == 1) {
32        return false;
33      } else {
34        return bauteile.remove(teil);
35      }
36    }
37
38    // void Rueckgabe ist nicht so gut
39    // keine Info ueber Miss-/Erfolg des Einfuegens
40    // + man erfaehrt nicht, ob Duplikat vorlag
41    // void -> boolean
```

```
42
43    // Refaktorisierte, Set-basierte Variante ohne Exception
44    // Ausnahmebehandlung sollte man nicht zur Behandlung
45    // normaler (d.h. haeufig auftretender) Programmsituationen
46    // einsetzen.
47    // (Siehe auch Tipp 34 aus "The Pragmatic Programmer":
48    // "Use Exceptions for Exceptional Problems" und Item 57
49    // aus "Effective Java": "Use exceptions only for
50    // exceptional conditions".)
51    public boolean addBauteil(Bauteil teil){
52      return bauteile.add(teil);
53    }
54  }
```

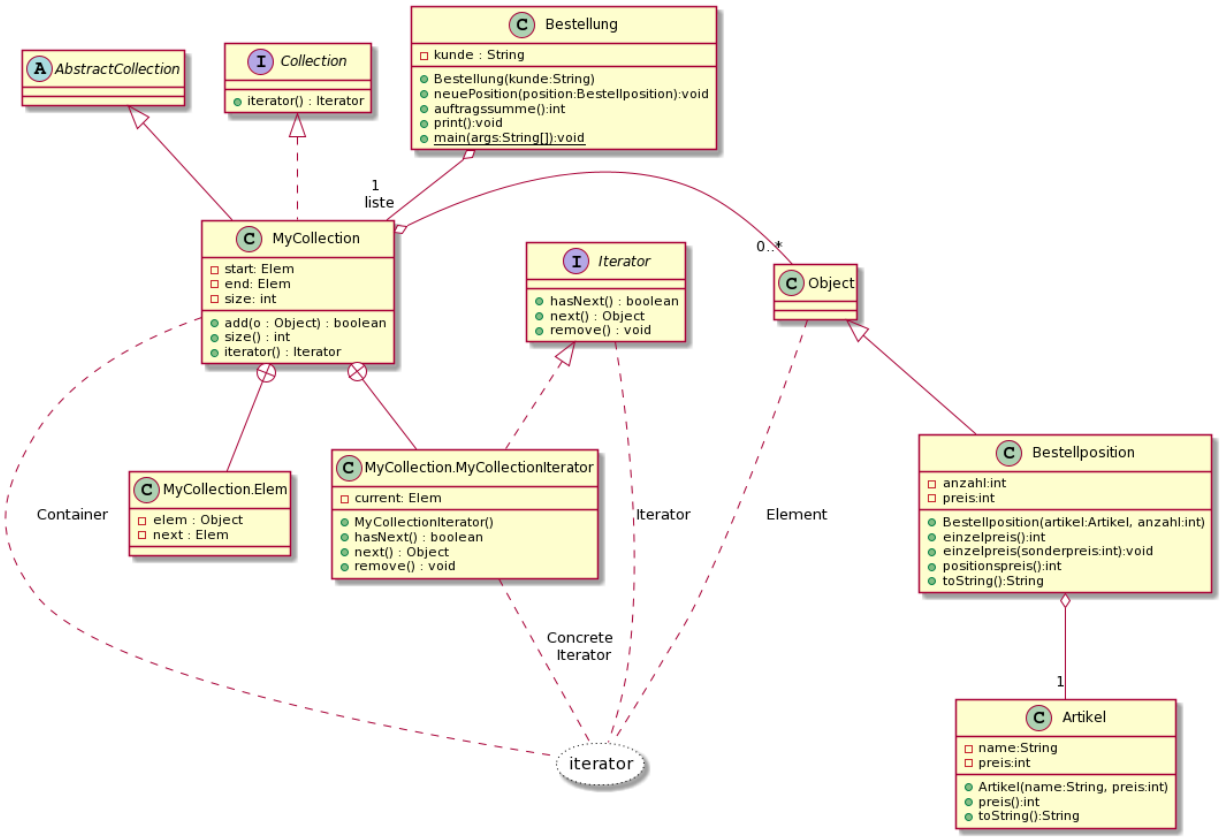
(d) Im Set würde das `t4` abgelehnt, in der Liste nicht.

Aufgabe 2

(a) UML-Diagramm



(b) UML-Diagramm mit Iterator-Pattern



(c) Datei `Bestellung.java`

```java
import java.util.Collection;
import java.util.Iterator;

public class Bestellung {
  private String kunde;
  private Collection<Bestellposition> liste;

  public Bestellung(String kunde) {
    this.kunde = kunde;
    liste = new MyCollection<Bestellposition>();
  }

  public void neuePosition(Bestellposition position) {
    liste.add(position);
  }

  public int auftragssumme() {
    Iterator iter = liste.iterator();
    int summe = 0;
```

```
20      while (iter.hasNext()) {
21        summe += iter.next().positionspreis();
22      }
23      return summe;
24    }
25
26    public void print() {
27      System.out.println("Bestellung fuer Kunde " + kunde);
28      Iterator iter = liste.iterator();
29      int pos = 0;
30      while (iter.hasNext()) {
31        System.out.println(pos + ". " + iter.next());
32        pos++;
33      }
34      System.out.println("Auftragssumme: " + auftragssumme());
35      System.out.println();
36    }
37
38    public static void main(String[] args) {
39      Artikel tisch = new Artikel("Tisch", 200);
40      Artikel stuhl = new Artikel("Stuhl", 100);
41      Artikel schrank = new Artikel("Schrank", 300);
42
43      Bestellung bestellung = new Bestellung("TUD");
44      bestellung.neuePosition(new Bestellposition(tisch, 1));
45      bestellung.neuePosition(new Bestellposition(stuhl, 4));
46      bestellung.neuePosition(new Bestellposition(schrank, 2));
47      bestellung.print();
48    }
49  }
```

Datei `MyCollection.java`

```
1   import java.util.AbstractCollection;
2   import java.util.Collection;
3   import java.util.Iterator;
4
5   public class MyCollection<T> extends AbstractCollection<T>
        implements Collection<T> {
6     private class Elem {
7       private T elem;
8       private Elem next;
9
10      public Elem(T elem, Elem next) {
11        this.elem = elem;
12        this.next = next;
13      }
14    }
15
16    private Elem start = null;
17    private Elem end = null;
18    private int size = 0;
```

```
19
20    @Override
21    public boolean add(T o) {
22      Elem e = new Elem(o, null);
23      if (end != null) {
24        end.next = e;
25      }
26      if (start == null) {
27        start = e;
28      }
29      end = e;
30      size++;
31      return true;
32    }
33
34    @Override
35    public int size() {
36      return size;
37    }
38
39    private class MyCollectionIterator implements Iterator<T> {
40      private Elem current;
41
42      public MyCollectionIterator() {
43        current = start;
44      }
45
46      @Override
47      public boolean hasNext() {
48        return current != null;
49      }
50
51      @Override
52      public T next() {
53        T o = current.elem;
54        current = current.next;
55        return o;
56      }
57
58      @Override
59      public void remove() {
60        throw new UnsupportedOperationException("Keine Lust zur
              Implementation");
61      }
62    }
63
64    @Override
65    public Iterator iterator() {
66      return new MyCollectionIterator();
67    }
68  }
```