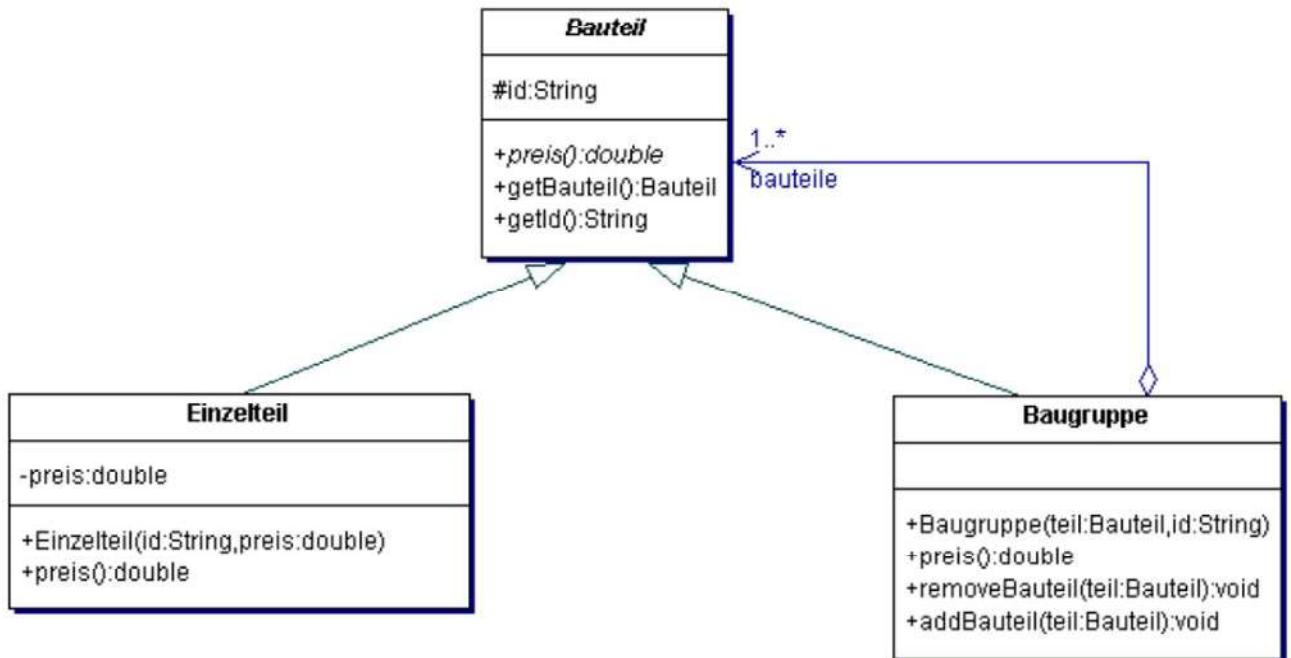


Ü6

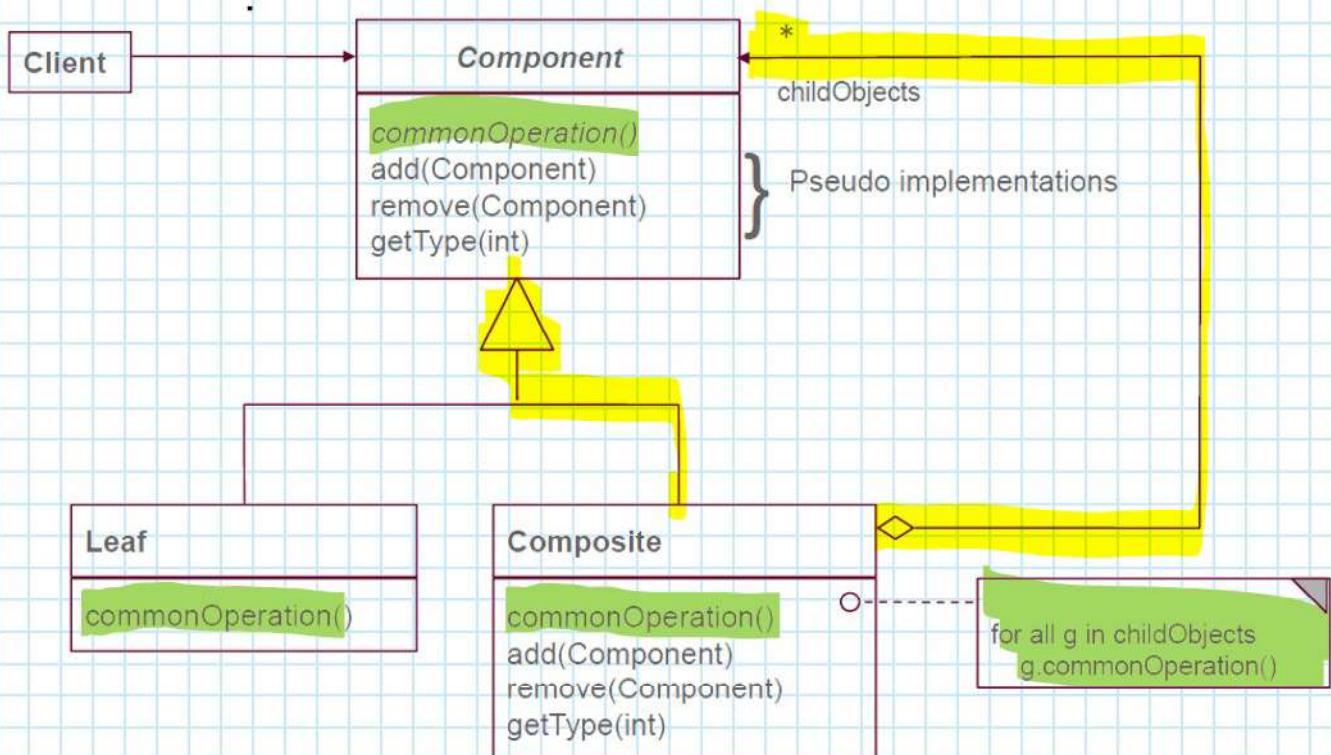
Mittwoch, 8. Mai 2019 17:56

Aufgabe 1 Bauteil

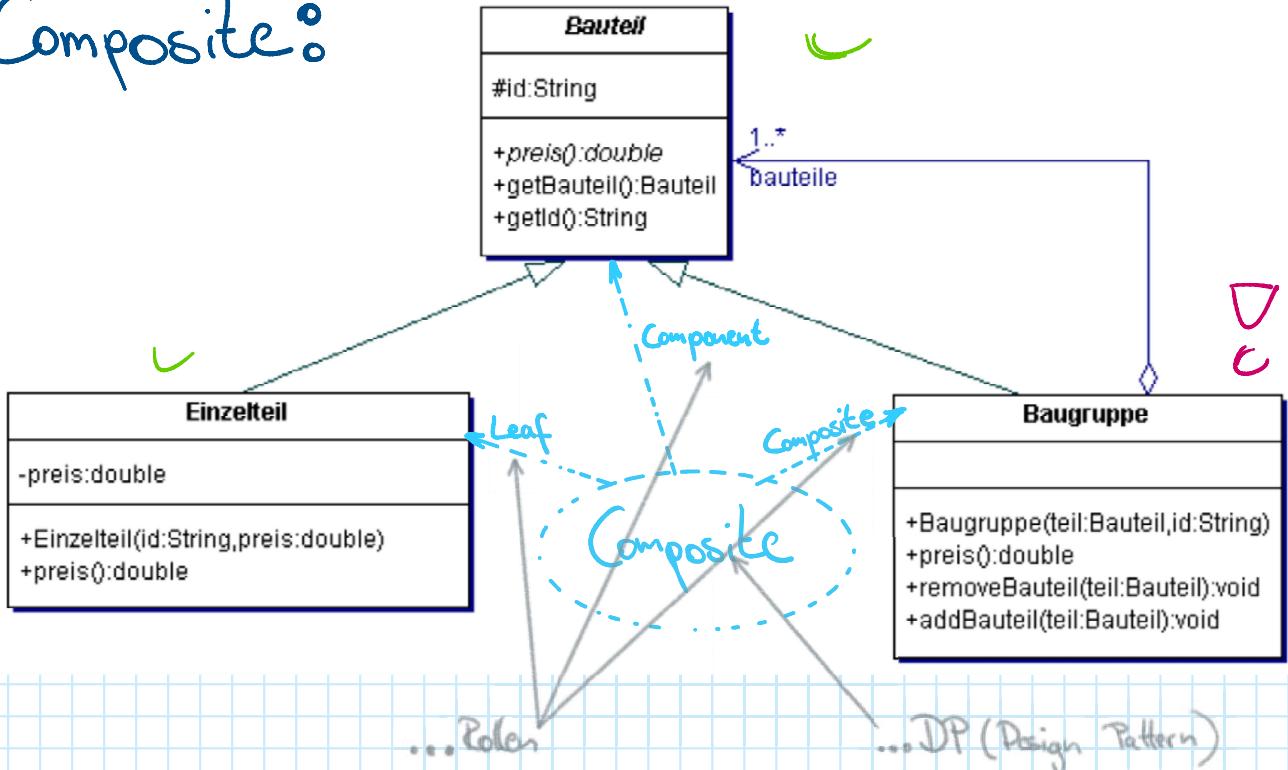


► Welches Entwurfsmuster? 2

→ Composite → rekursive n-Aggregation zur Superklasse



Composite:



```

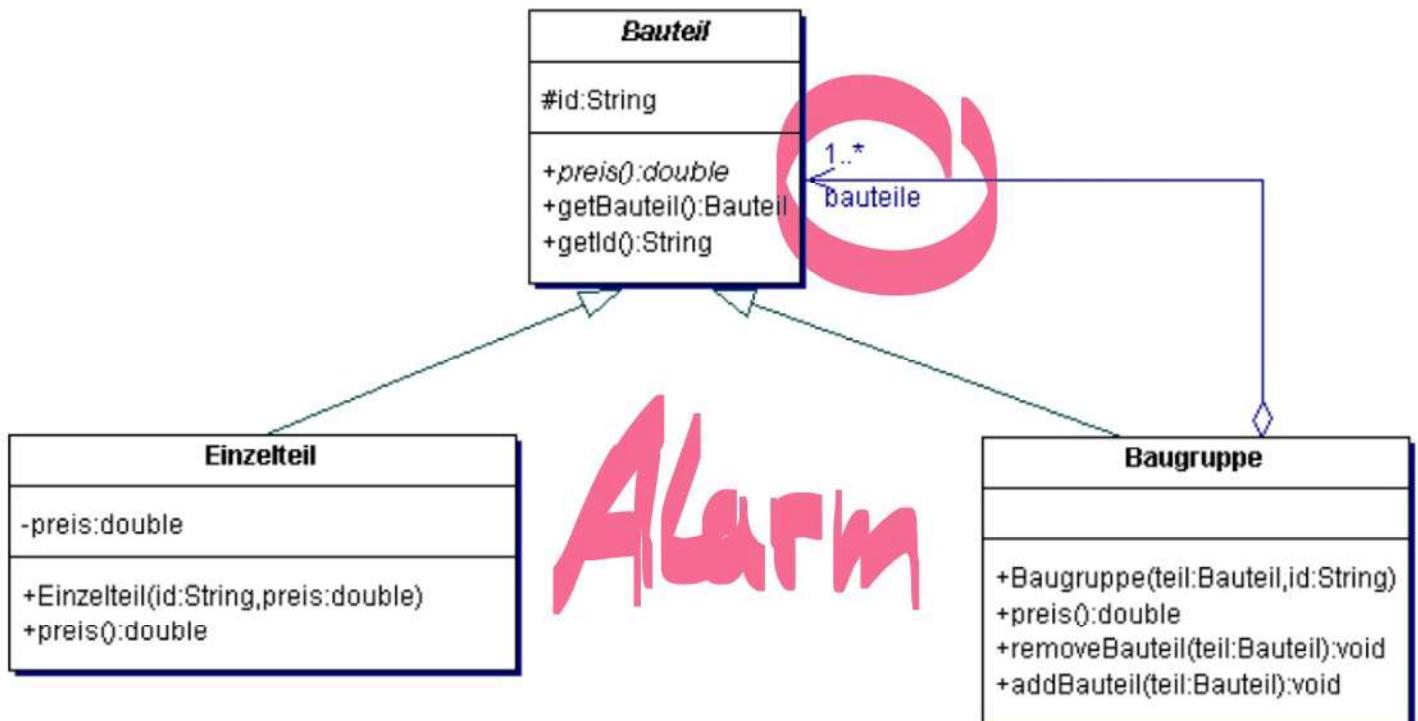
1 import java.util. Set
2 import java.util. HashSet
3
4 public class Baugruppe extends Bauteil {
5     private Set< Bauteile > bauteile;
6
7     public Baugruppe(Bauteil bauteil, String id) {
8         this.bauteile = new HashSet< Bauteile >();
9         this.addBauteil(bauteil);
10        this.id = id;
11    }
12
13    public double preis() {
14        double preis = 0;
15        for (Bauteil b : bauteile) {
16            preis += b.preis();
17        }
18        return preis;
    }
  
```

```

18     return preis;
19   }
20
21   public void removeBauteil(Bauteil bauteil) {
22     this.bauteile.remove(bauteil);
23   }
24
25   public void addBauteil (Bauteil bauteil) {
26     this.bauteile.add (bauteil);
27   }

```

► Diskutiere removeBauteil()



Was passiert wenn wir nur ein Bauteil in einer Baugruppe haben und dann removeBauteil() aufrufen?

→ aktuell wird es einfach entfernt
⇒ Spezifikation nicht richtig umgesetzt

→ Ausweg
 ② return true/false
 ① Exception werfen

①

```
21 public void removeBauteil(Bauteil bauteil) throws IllegalStateException {  
22     if (this.bauteile.size() == 1) {  
23         throw new IllegalStateException(  
24             "Eine Baugruppe muss mindestens aus einem  
25             Bauteil bestehen.");  
26     } else {  
27         this.bauteile.remove(bauteil);  
28     }  
29 }
```

②

```
21 public boolean removeBauteil(Bauteil bauteil) throws IllegalStateException {  
22     if (this.bauteile.size() == 1) {  
23         return false;  
24     } else {  
25         this.bauteile.remove(bauteil);  
26         return true;  
27     }  
28 }
```

► Set oder List ?

→ Unterschiedliche Ergebnisse von BauteilTest,
da es in Sets keine Duplicatae gibt

↳ Entscheidung hängt von Aufgabe / Problem ab

Aufgabe 2 (My Collection)

Artikel

```
* Artikel.java X MyCollection.java Bestellung.java Bestellposition.java
1 class Artikel {
2     private String name;
3     private int preis;
4
5     public Artikel (String name, int preis) {
6         this.name = name;
7         this.preis = preis;
8     }
9
10    public int preis() {
11        return preis;
12    }
13
14    public String toString() {
15        return name;
16    }
17 }
```

Artikel

- name : String
- preis : int

- + Artikel (name: String, preis: int)
- + preis () : int
- + toString () : String

```

1  public class Bestellposition {
2      private Artikel artikel;
3      private int anzahl;
4      private int preis;
5
6      public Bestellposition (Artikel artikel, int anzahl) {
7          this.artikel = artikel;
8          this.anzahl = anzahl;
9          this.preis = artikel.preis();
10     }
11
12     public int einzelpreis() {
13         return preis;
14     }
15
16     public void einzelpreis (int sonderpreis) {
17         preis = sonderpreis;
18     }
19
20     public int positionspreis() {
21         return preis*anzahl;
22     }
23
24     public String toString() {
25         return anzahl + " x " + artikel +
26             " Einzelpreis: " + preis +
27             " Summe: " + positionspreis();
28     }
29 }

```

Bestellposition

- artikel : Artikel
- anzahl : int
- preis : int

+ Bestellposition (

 artikel: Artikel, int: anzahl)

+ einzelpreis () : int

+ einzelpreis (sonderpreis: int) : void

+ positionspreis () : int

```

1 import java.util.Collection;
2
3 public class Bestellung {
4     private String kunde;
5     private Collection liste;
6
7     public Bestellung(String kunde) {
8         this.kunde = kunde;
9         liste = new MyCollection();
10    }
11
12    public void neuePosition (Bestellposition b) {
13        liste.add(b);
14    }
15
16    public int auftragssumme() {
17        Iterator i = liste.iterator();
18        int s = 0;
19        while (i.hasNext())
20            s += ((Bestellposition)i.next()).positionspreis();
21        return s;
22    }
23
24    public void print () {
25        System.out.println("Bestellung fuer Kunde "+kunde);
26        Iterator i = liste.iterator();
27        int pos = 0;
28        while (i.hasNext()) {
29            System.out.println(pos+". "+i.next());
30            pos++;
31        }
32        System.out.println("Auftragssumme: "+auftragssumme());
33        System.out.println();
34    }
35
36
37    public static void main (String[] args) {
38        Artikel tisch = new Artikel("Tisch",200);
39        Artikel stuhl = new Artikel("Stuhl",100);
40        Artikel schrank = new Artikel("Schrank",300);
41
42        Bestellung b1 = new Bestellung("TUD");
43        b1.neuePosition(new Bestellposition(tisch,1));
44        b1.neuePosition(new Bestellposition(stuhl,4));
45        b1.neuePosition(new Bestellposition(schrank,2));
46        b1.print();
47    }
48 }

```

Bestellung

- kunde : String
- liste : Collection

+ Bestellung (kunde: String)

+ neue Position (b: Bestellposition) : void

+ auftragssumme () : int

+ print () : void

+ main (args: String[]) : void

```

  *Artikel.java   *MyCollection.java X  *Bestellung.java  Bestellposition.java
1 import java.util.Collection;
2
3 class MyCollection extends AbstractCollection implements Collection{
4     private Elem start;
5     private Elem end;
6     private int size;
7
8     //Innere Klasse
9     private class Elem {
10         private Object elem;
11         private Elem next;
12
13         public Elem (Object elem, Elem next) {
14             this.elem = elem;
15             this.next = next;
16         }
17     }
18
19
20     public boolean add (Object o) {
21         Elem e = new Elem(o,null);
22         if (end != null)
23             end.next = e;
24         if (start == null)
25             start = e;
26         end = e;
27         size++;
28         return true;
29     }
30
31
32     public int size() {
33         return size;
34     }
35
36     private class MyCollectionIterator implements Iterator {
37         private Elem current;
38         private Elem removeRef;
39
40         public MyCollectionIterator() {
41             current = start;
42         }
43
44         public boolean hasNext() {
45             return current != null;
46         }
47
48         public Object next() {
49             Object o = current.elem;
50             current = current.next;
51             return o;
52         }
53
54         public void remove() {
55             throw new UnsupportedOperationException();
56         }
57
58     }
59     public Iterator iterator() {
60         return new MyCollectionIterator();
61     }
62 }

```

→ Eltern und
MyCollection Iterator
 müssen nicht als
 Innere Klassen im UML
 dargestellt werden

My Collection

- start : Elem
- end : Elem
- size : int

Elem

- elem : Object
 - next : Elem
- + Elem (elem: Object, next: Elem)

+ add (o: Object) : boolean

+ size () : int

MyCollectionIterator

- current : Elem
- removeRef : Elem

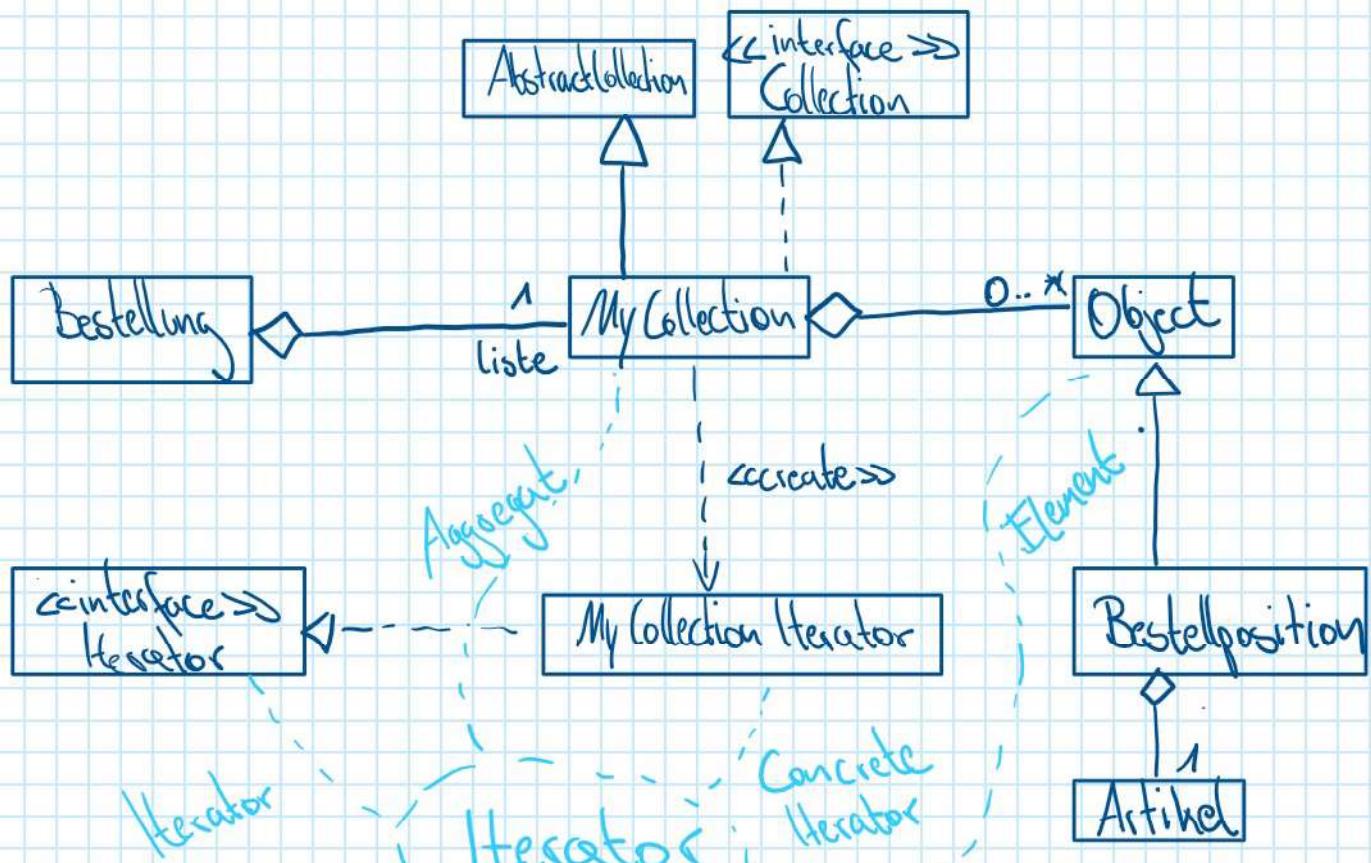
+ MyCollectionIterator()

+ hasNext () : boolean

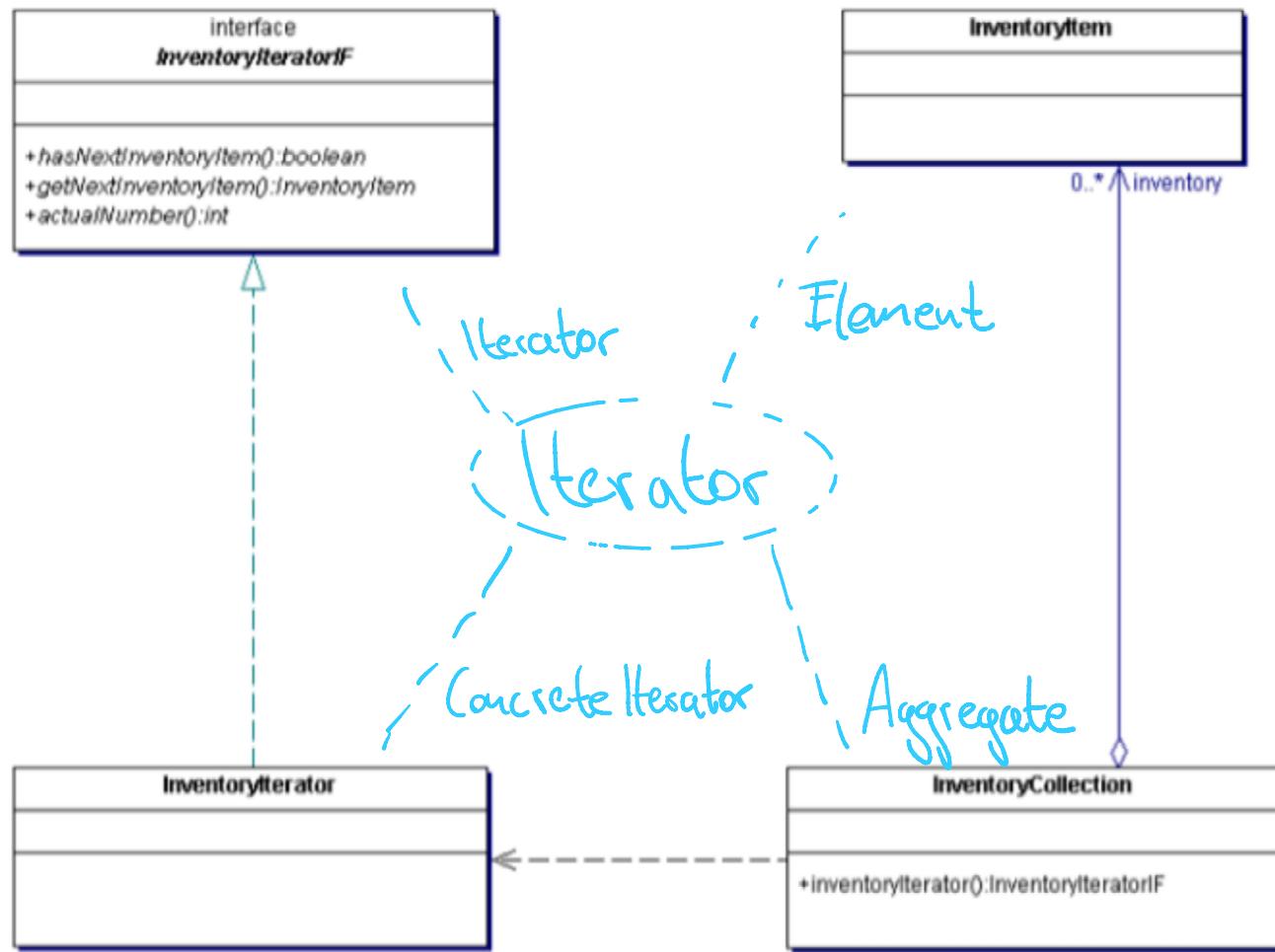
+ next () : Object

+ remove () : void

+ iterator : Iterator



Aufgabe 3. aus Klausur



2. Entwurfsmuster 2
o