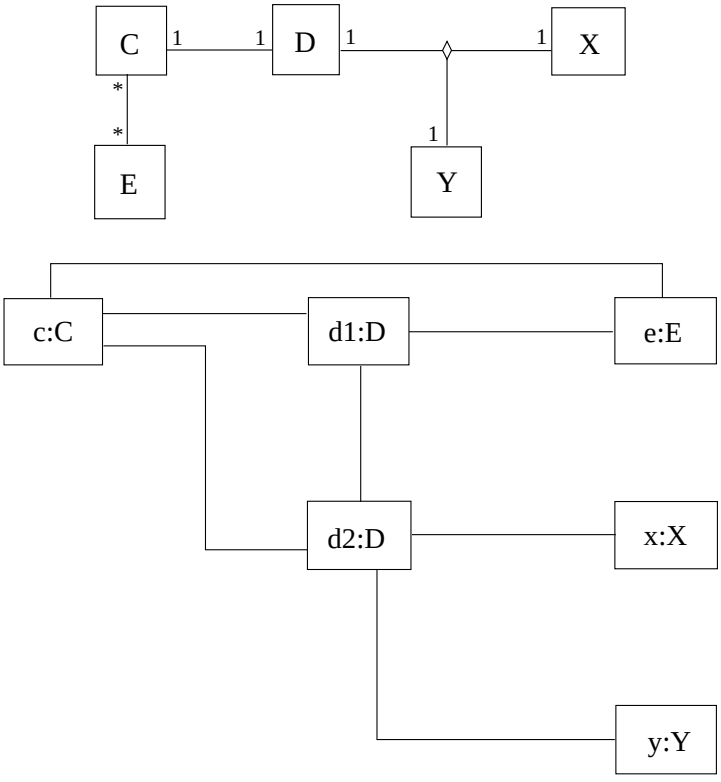
Ludwig-Maximilians-Universität München **WS 2013/14**
Institut für Informatik **Übungsblatt 11**

Softwaretechnik

Prof. Tomas Bures, PhD., Dipl. Inf. Lenz Belzner, Dipl. Inf. Christian Kroiß

Aufgabe 1

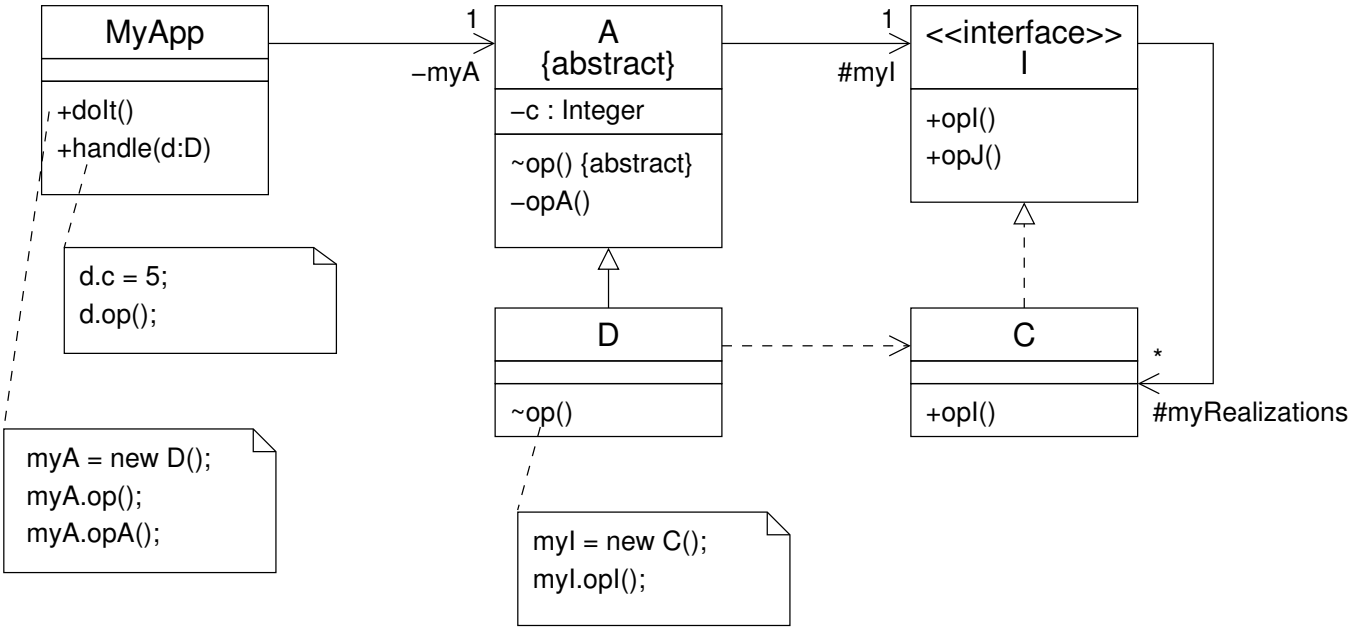
Folgendes Klassen- und Objektdiagramm seien gegeben. Das Objektdiagramm enthält Fehler. Beschreiben Sie diese kurz.



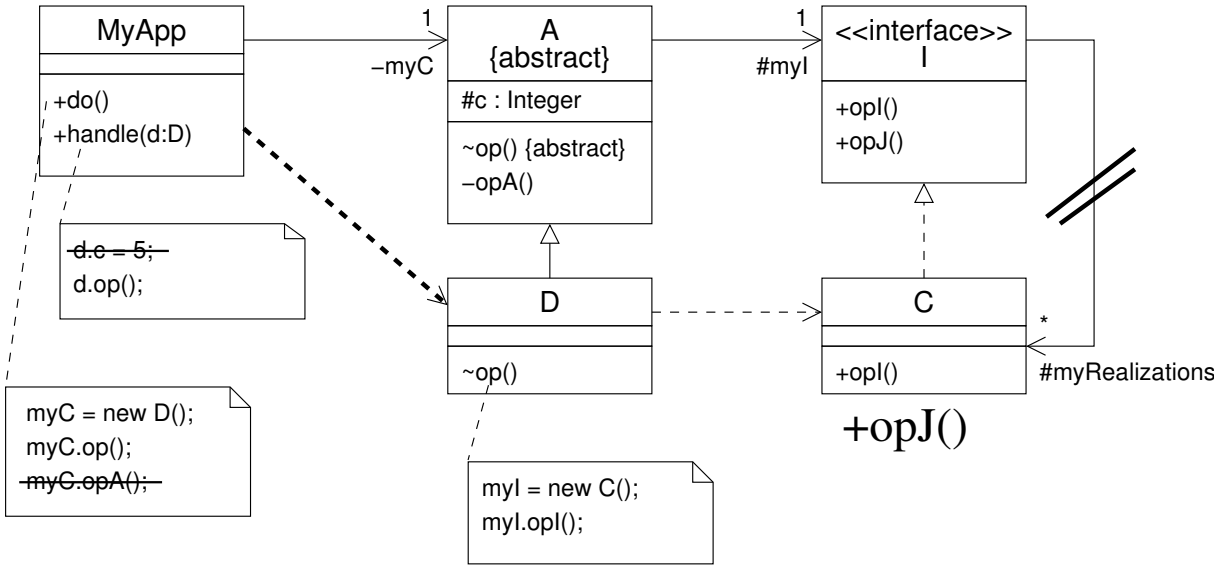
- Beschriftungen nicht unterstrichen
- Die Assoziation zwischen d1:D und e:E wird nicht vom Klassendiagramm erfasst.
- Die Assoziation zwischen d1:D und d2:D wird nicht vom Klassendiagramm erfasst.
- c:C ist mit zwei D-Objekten assoziiert.
- Die multiple Assoziation zwischen D, X und Y ist für d1 und d2 nicht nicht voll erfüllt.

Aufgabe 2

Das folgende Klassendiagramm mit Java-Annotationen enthält Fehler. Beschreiben Sie diese kurz.



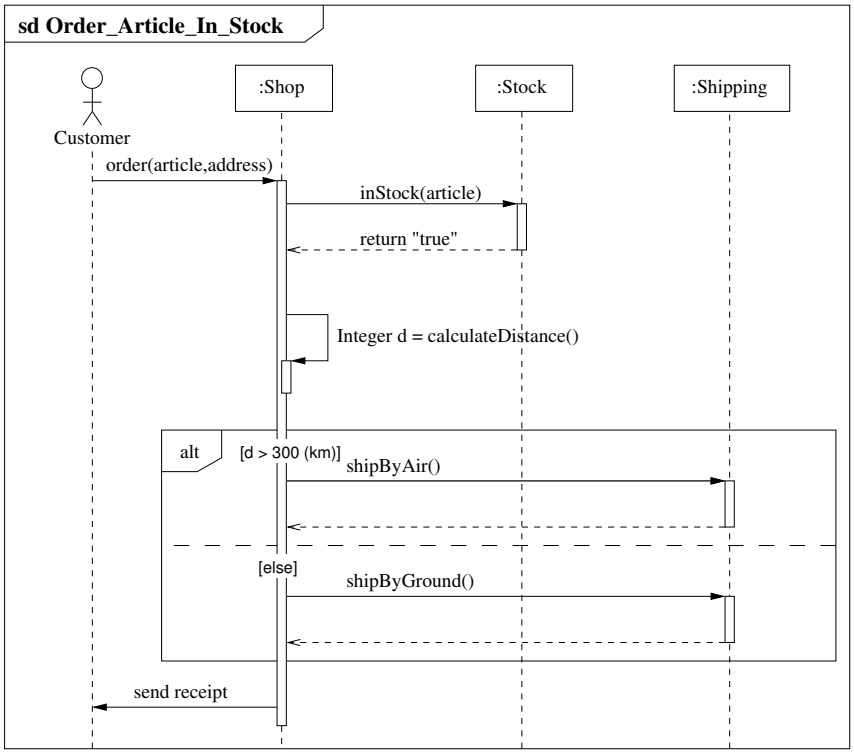
Tip: Auch nach fehlenden Abhängigkeiten suchen!

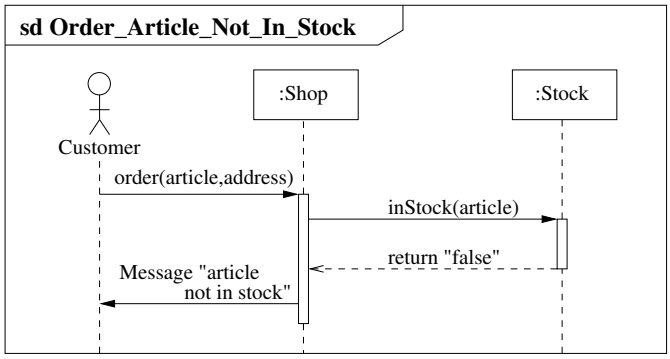


5 errors overall

Aufgabe 3

Gegeben sei das folgende Klassendiagramm für einen Shop mit einer Versandabteilung (Shipping) und einem Artikellager (Stock). Die beiden Sequenzdiagramme beschreiben mögliche Szenarien wenn ein Kunde einen Artikel bestellt. Beide Szenarien können beliebig oft wiederholt werden.

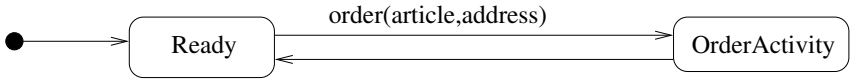


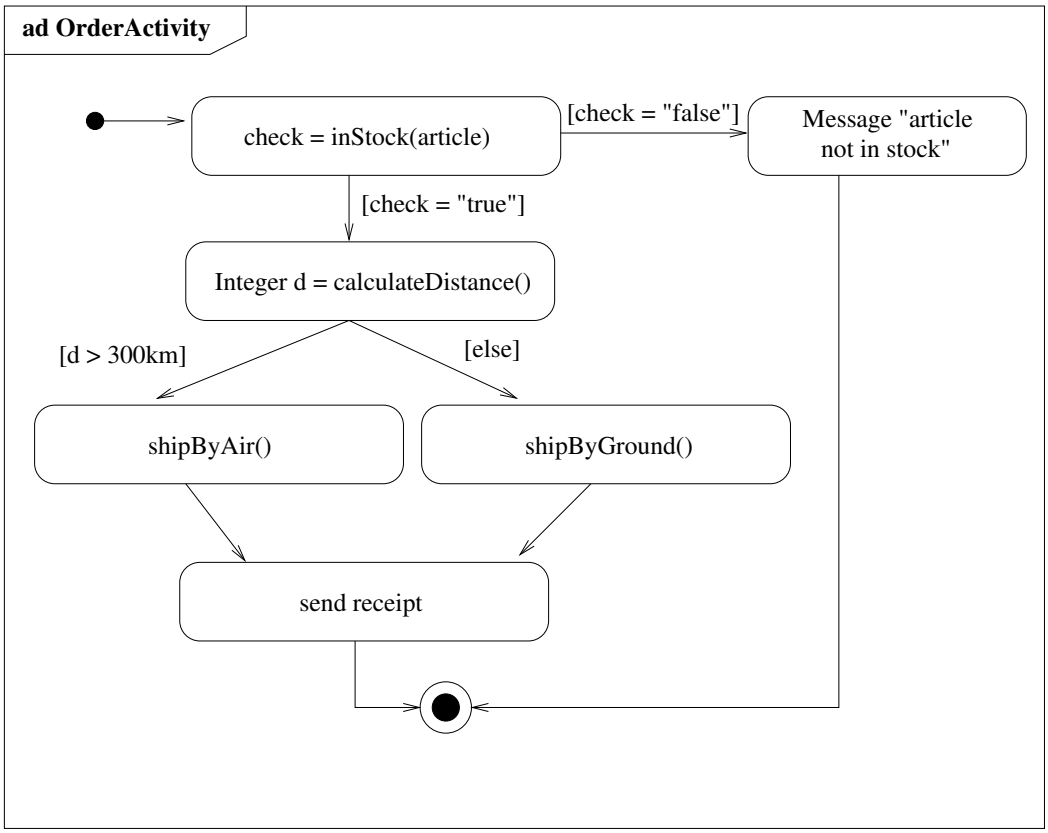


1. Leiten Sie ein Zustandsdiagramm für das Verhalten der Klasse Shop von den zwei Sequenzdiagrammen ab. Verwenden Sie dazu das Verfahren aus der Vorlesung mit stabilen und Aktivitätszuständen.

2. Entwerfen Sie ein Aktivitätsdiagramm für jeden Aktivitätszustand.

3. Welche Operationen müssen gemäß der Sequenzdiagramme beim Objektdesign welcher Klasse hinzugefügt werden?


order(article,address)
Ready
OrderActivity


ad OrderActivity
check = inStock(article)
[check = "false"]
Message "article not in stock"
[check = "true"]
Integer d = calculateDistance()
[d > 300km]
[else]
shipByAir()
shipByGround()
send receipt

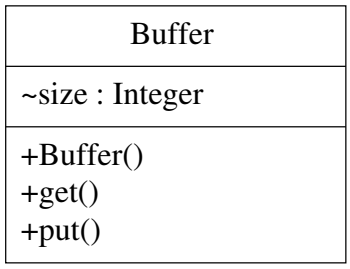
Shop: order(article:Integer,address:String), calculateDistance():Integer

Stock: inStock(article:Integer):Boolean

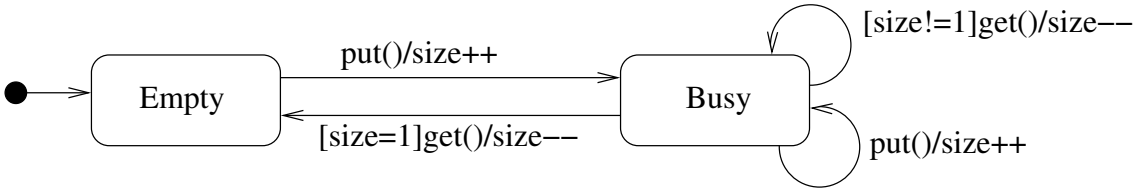
Shipping: shipByAir(), shipByGround()

Aufgabe 4

Die folgende Klasse *Buffer* modelliert einen Buffer mit unbeschränkter Kapazität.



Das Verhalten der Klasse ist durch folgendes UML-Zustandsdiagramm gegeben.



Implementieren Sie das Zustandsdiagramm mit Zustandsobjekten.

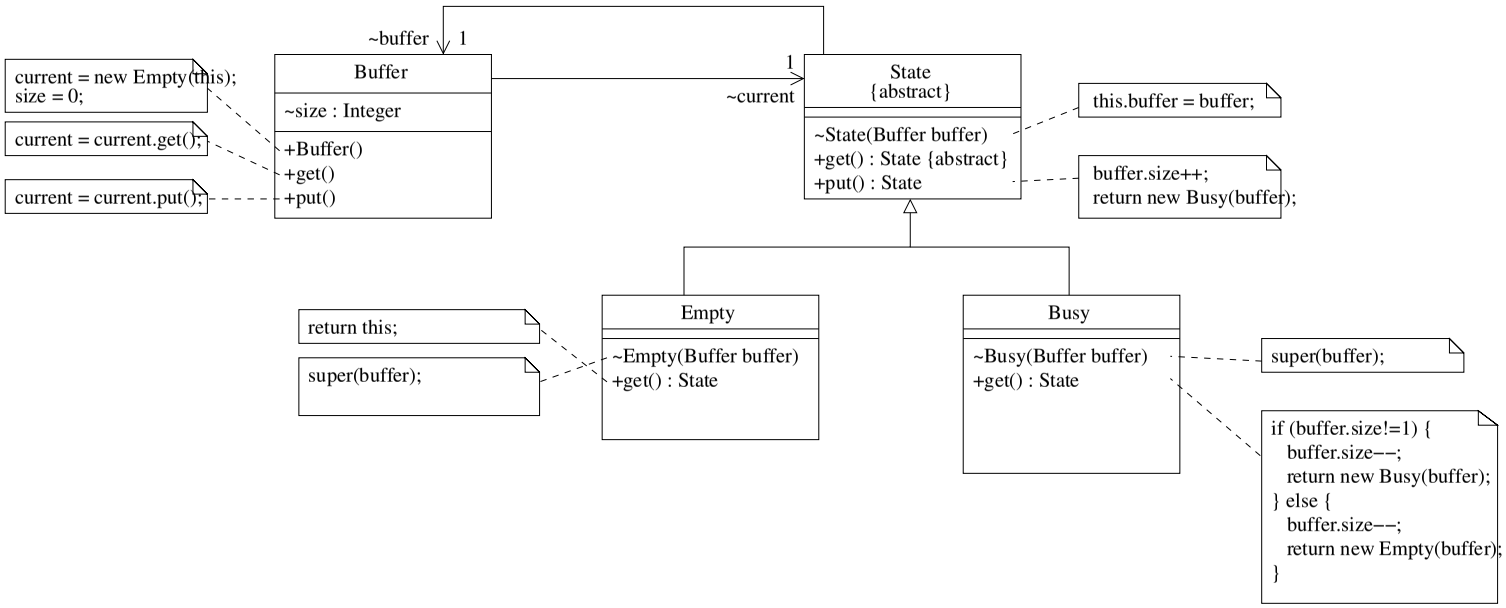
```
public class Buffer {
  int size;
  State current;
  public Buffer() {
    current = new Empty(this);
    size = 0;
  }
  public void get() {
    current = current.get();
  }
  public void put() {
    current = current.put();
  }
}

abstract class State {
  Buffer buffer;
  State(Buffer buffer) {
    this.buffer = buffer;
  }
  public abstract State get();
  public State put() {
    buffer.size ++;
    return new Busy(buffer);
  }
}

class Empty extends State {
  Empty(Buffer buffer) {
    super(buffer);
  }
  public State get() {
    return this;
  }
}

class Busy extends State {
  Busy(Buffer buffer) {
```

```
    super(buffer);
  }
  public State get() {
    if (buffer.size != 1) {
      buffer.size--;
      return new Busy(buffer);
    } else {
      buffer.size--;
      return new Empty(buffer);
    }
  }
}
```


~buffer
1
Buffer
~size : Integer
+Buffer()
+get()
+put()
current = new Empty(this);
size = 0;
current = current.get();
current = current.put();
1
~current
State
{abstract}
~State(Buffer buffer)
+get() : State {abstract}
+put() : State
this.buffer = buffer;
buffer.size++;
return new Busy(buffer);
Empty
~Empty(Buffer buffer)
+get() : State
return this;
super(buffer);
Busy
~Busy(Buffer buffer)
+get() : State
super(buffer);
if (buffer.size!=1) {
buffer.size−−;
return new Busy(buffer);
} else {
buffer.size−−;
return new Empty(buffer);
}