Aufgabe 1 (Modellieren und Schreiben von Klassen)

Sie sollen in dieser Aufgabe Klassen entwickeln, die Wetterdaten repräsentieren.

a) Entwickeln Sie eine Klasse `Wetter`. Ein Objekt dieser Klasse repräsentiert Informationen über eine Wettermessung. Sie soll drei Eigenschaften haben: einen Namen des Ortes der Messstation, der durch eine Zeichenkette dargestellt ist, die gemessene Temperatur in Grad Celcius als ganze Zahl und den gemessenen Luftdruck in Gr[illegible]s als ganze Zahl.

Schreiben Sie einen Konstruktor, der alle Felder initialisiert.

Überschreiben Sie die Methode `equals` auf adequate Weise.

Überschreiben Sie die Methode `toString`.

```
class Wetter {
  String ort;
  int    temp;
  int    druck;
  Wetter(String o, int t, int d){
    ort = o; temp = t; druck = d;
  }
  @Override
  public boolean equals(Object o){
    if (o == null) return false;
    if(o instanceof Wetter that)
      return ort.equals(that.ort)
          && temp == that.temp
          && druck == that.druck;
    return false;
  }
```

Bachelorprüfung: Objektorientierte Softwareentwicklung

SS21

Studiengang (AI/AI Dual): _____

Erlaubte Hilfsmittel: Keine

Jeder Griff zu einem elektronischen Gerät (z.B. Smartphone, Smartwatch, Taschenrechner) wird geahndet.

Lesezeit ist auf den Klausurbögen angekreuzt. Sie müssen nicht in Lesezeit schreiben.

Bitte legen Sie den Studierendenausweis auf den Tisch.

Bearbeitungszeit: 90 Minuten

_____

Unterschrift

Benotung

Aufgabe	1	2	3	4	Gesamt	Note
Punkte	25	30	20	25	100	
erreicht						

Aufgabe 1 (Modellierung und Schreiben von Klassen)

Sie wollen in dieser Aufgabe Klassen entwickeln, die Warenkörbe repräsentieren.

a) Entwickeln Sie eine Klasse `Artikel`, die einen Artikel dieses Warenkorbes repräsentiert. Ein Artikel hat einen Namen, einen Preis und eine Anzahl. Implementieren Sie einen Konstruktor, der alle Attribute initialisiert. Implementieren Sie außerdem Getter-Methoden für alle Attribute. (7.5)

Schreiben Sie eine Klasse `Warenkorb`, die beliebig viele Artikel speichern kann. (2.5)

Überschreiben Sie die Methode `toString()`, sodass sie eine übersichtliche Auflistung der Artikel im Warenkorb und den Gesamtpreis ausgibt. (5)

Überschreiben Sie die Methode `toString()`.

3

3

```
@Override
public String toString() {
    return "Artikel in Warenkorb: " +
        + name + ": Gesamtpreis: "
        + anzahl + " mal Preis: "
        + preis;
}
```

b) Schreiben Sie eine Unterklasse `Alpenwetter` der Klasse `Wetter`. Diese Klasse soll ein zusätzliches Feld haben, das eine Lawinenwarnstufe aus einer der vier folgenden Aufzählungswerte speichert: keine, gering, erhöht, hoch.

```
enum Lawinenwarnstufe {
    keine, gering, erhöht, hoch;
}
class Alpenwetter extends Wetter {
    Lawinenwarnstufe lw;
    Alpenwetter(String o, int t, int d
                , Lawinenwarnstufe l) {
        super(o, t, d);
        lw = l;
    }
}
```

c) Gegeben Sei die Klasse `Zeitreihe`, die eine Liste von Wetterdaten ist:

```
class Zeitreihe extends java.util.ArrayList<Wetter>{
}
```

Listing 1: Zeitreihe.java

Schreiben Sie für diese Klasse eine Methode:

`double temperaturDurchschnitt()`,

die die Durchschnittstemperatur über alle Temperaturwerte der Liste darstellt.

```
double temperaturDurchschnitt() {
    double r = 0;
    for(Wetter w : this) {
        r = r + w.temp;
    }
    return r / this.size();
}
```

d) Schreiben Sie für die Klasse `Zeitreihe` eine Methode

```
int anzahlWarnungenHoch(){
```

/** die die Anzahl der in der Liste enthaltenen Alpenwettermessungen mit einer Lawinenwarnstufe >noch< berechnet. */

```
int r = 0;
for(var w : this){
  if(w instanceof Alpenwetter aw)
    if(aw.lw == Lawinenwarnstufe.hoch)
      r++;
}
return r;
}
```

Zeile	x	y	i	Ausdruck
3, 4, 6, 7	14	13	13	"A: 13 14 13"
16, 17	[illegible]	16	12	
6, 7			8	"A: 8 14 16"
9, 11	15			"weiter"
9, 14	16			"I: 8 16 16"
16		12	7	
6, 7			3	"A: 3 16 14"
9, 14	17			"I: 3 17 17"
9, 11	18			"weiter"
9, 14	19			"I: 3 17 17"
16, 17		22	2	
6			-2	

b) Betrachten Sie folgende Klasse:

```
1  public class Aufgabe2b{
2      public int e(int a, int b, int c) {
3          if (a <= 0) return c;
4          return e(a-1,b,c*b);
5      }
6
7      public static void main(String[] args){
8          System.out.println(e(4,3,1));
9      }
10 }
```

Listing 3: Aufgabe2b.java

Berechnen Sie schrittweise das Ergebnis des Ausdrucks `e(4,3,1)`

Was berechnet die Funktion?

e(4,3,1)
→ e(3,3,3)
→ e(2,3,9)
→ e(1,3,27)
→ e(0,3,81)
→ 81

Es wird $b^a \cdot c$ berechnet.

a) Schreiben Sie in der Klasse `AL` eine Methode `startsWith`:

Es soll getestet werden, ob die übergebene Liste ein Präfix der this-Liste ist.

```
startsWith(AL<A> prefix){
    if (prefix.size() > this.size())
        return false;
    for (int i=0; i< prefix.size(); i++)
        if(!prefix.get(i).equals( get(i)))
            return false;

    return true;
}
```

Listing 5: AL.java

b) Schreiben Sie in der Klasse `AL` eine Methode `reverse`:

Die this-Liste soll so verändert werden, dass sie anschließend die Elemente in umgekehrter Reihenfolge enthält.

```
reverse(){
    for(int i=0; i<size()/2; i++) {
        var tmp = array[i];
        array[i] = array[size()-1-i];
        array[size()-1-; ] = tmp;
    }
}
```

Listing 6: AL.java

c) Gegeben sei zusätzlich folgende Klasse:

```
1  class Pair<A,B>{
2    A fst;
3    B snd;
4    Pair(A a, B b){ fst=a; snd=b; }
5  }
```

Listing 7: Pair.java

Sie soll verwendet werden, um Paare von Objekten zu bilden.

Schreiben Sie für die Klasse `AL` eine Methode `makeIndex()`.

Es soll eine neue Liste erstellt werden, die aus der ursprünglichen Liste entsteht, indem für jedes Element mit seinem Index ein Paar gebildet wird.

```
<A> AL<Pair<Integer, A>> makeIndex(){
  var rs = new AL<Pair<Integer, A>>();
  for(var i=0; i<size(); i++){
    rs.add(new Pair<>(i, get(i)));
  }
  return rs;
}
```

Listing 8: AL.java

d) Schreiben Sie eine Methode `getValue`, die aus einer Liste von Paaren, das Objekt im Feld `snd` aus dem Paar zurück gibt, in dem das Feld `fst` ein Objekt gleich zum Parameter `a` ist.

Ist kein solches Element enthalten, soll `null` als Ergebnis zurück gegeben werden.

```
<A,B> B lookUp(AL<Pair<A,B>> ps,A a){
  for (var i= 0; i< size(); i++) {
    if (get(i).fst.equals(a))
      return get(i).snd;
  }
  return null;
}
```

Listing 9: AL.java

e) Wie lässt sich ein Ausdruck der Form `lookUp(xs makeIndex(),x)` klüger umsetzen?

```
get(x)
```

Aufgabe 4 Zusammenhänge

Erklären Sie in kurzen Worten.

a) Können Schnittstellen andere Schnittstellen erweitern und wenn ja wie viele? Wie geht das syntaktisch?

Ja, beliebig viele. Hierzu werden die Oberschnittstellen nach dem Schlüsselwort 'extends' kommagetrennt aufgelistet.

b) Was kann bei der Implementierung eines Konstruktors die erste Anweisung sein. Was passiert, wenn keine dieser Anweisungen die erste Anweisung des Konstruktors ist?

Es kann der Aufruf eines Konstruktors der Oberklasse mit 'super' sein oder der eigenen Klasse mit 'this'. Wenn keine dieser Aufrufe im Quelltext steht generiert der Compiler: super().

c) Was versteht man unter der *diamond notation*? Wofür ist diese nützlich?

Beim Aufruf des Konstruktors einer generischen Klasse darf leere Spitze Klammern gesetzt <>. Die Typparameter werden inferiert.

d) Was ist der Unterschied zwischen Überladen und Überschreiben von Methoden?

Überladen ist eine Methode mit gleichen Namen und unterschiedlichen Parametern.
Geerbte Methode wird in Unterklasse neu definiert bezeichnet man als Überschreiben. Die Parametertypen sind dann gleich.

e) Welche Eigenschaft haben funktionale Schnittstellen? Wozu kann man diese Eigenschaft nutzen?

Sie haben nur eine abstrakte Methode. Diese kann mit einen Lambda-Ausdruck implementiert werden.