CARL VON OSSIETZKY universität OLDENBURG

Dr. D. Boles, Dr. C. Schönberg
Carl von Ossietzky Universität Oldenburg, Fakultät II, Department für Informatik

Klausur – Gruppe A

Objektorientierte Modellierung und Programmierung (inf031)

Sommersemester 2019

- Lassen Sie die Blätter in jedem Fall zusammengeheftet.
- Geben Sie auf dieser Seite Name, Vorname und Matrikelnummer an und unterschreiben Sie die Erklärung.
- Notieren Sie die Lösungen zu den Aufgaben direkt auf dem Aufgabenzettel oder auf der vorhergehenden oder nachfolgenden Rückseite.
  Wenn Sie Lösungen an anderer Stelle notieren, markieren Sie dies deutlich sowohl dort als auch auf der Seite der Aufgabe. Zusätzliche Blätter erhalten Sie bei der Aufsicht. Verwenden Sie auf keinen Fall eigene Blätter.
- Für die Bearbeitung der Klausur stehen Ihnen 120 Minuten zur Verfügung.
- Schalten Sie Mobiltelefone und ähnliche Geräte aus und verstauen Sie sie sicher.
  Während der Klausur sichtbare technische Hilfsmittel gelten als Täuschungsversuch.
- Benutzen Sie nur dokumentenechte Stifte mit blauer oder schwarzer Tinte.
  Alles, was mit Bleistift oder in roter Tinte geschrieben ist, wird nicht gewertet.
- Als einziges Hilfsmittel ist ein doppelseitig *handbeschriebenes* DIN A4 Blatt zugelassen.

Name, Vorname: _____

Matrikelnummer: _____

Erklärung: Ich versichere, dass ich die Klausur selbstständig, ohne fremde Hilfe und ohne weitere Hilfsmittel bearbeitet habe.

_____
Unterschrift

Bewertung (vom Prüfer auszufüllen):

Aufgabe:	1	2	3	4	5	6	7	8	$\sum$
Punkte:	14	10	10	19	16	10	9	12	100
Erreicht:									

Aufgabe 1: *Klassen, Objekte, Exceptions* **(14 Punkte)**

In dieser Aufgabe wird ein stark abgespecktes soziales Netzwerk namens *Likebook* simuliert. Likebook besteht hierbei aus einer Menge an Nutzern (m/w/d) mit eindeutigen Namen. Jeder Nutzer hat eine Pinnwand, an der er Nachrichten hinterlassen kann. Nutzer können Nachrichten mit „Gefällt mir“ markieren.

Gegeben sei folgender Ausschnitt aus einer API, die die Abfrage bestimmter Likebook-Daten ermöglicht:

```
1  public class UnknownUserException extends Exception {
2
3    /** erzeugt eine neue Exception fuer einen unbekannten
4     * Nutzer mit dem gegebenen Namen */
5    public UnknownUserException(String name)
6
7    /** liefert den Namen des unbekannten Nutzers */
8    public String getName()
9  }
10
11 /** Likebook enthaelt eine Menge von Nutzern */
12 public class Likebook {
13
14   // Achtung: privater Konstruktor!
15   private Likebook()
16
17   /** laedt alle Likebook-Daten aus der Cloud und
18    * liefert ein Likebook-Objekt, ueber das diese Daten
19    * abgefragt werden koennen (Achtung: Methode ist static!) */
20   public static Likebook load()
21
22   /** liefert den Nutzer mit dem angegebenen Namen;
23    * wirft eine Exception, wenn ein Nutzer mit dem Namen nicht
24    * in Likebook existiert */
25   public User getUser(String name) throws UnknownUserException
26 }
27
28 /** jeder Nutzer hat einen eindeutigen Namen; jedem Nutzer gehoert
29  * eine Pinnwand, auf der er Nachrichten posten kann */
30 public class User {
31
32   /** zwei Nutzer sind gleich, wenn sie denselben Namen haben */
33   @Override
34   public boolean equals(Object obj) {
35     return (obj instanceof User) && name.equals(((User) obj).name);
36   }
37
38   /** liefert den Namen des Nutzers */
39   public String getName()
40
41   /** liefert die Pinnwand des Nutzers */
42   public Board getBoard()
43 }
44
45 /** jede Pinnwand gehoert einem Likebook-Nutzer; dieser kann auf
46  * seiner Pinnwand Nachrichten posten */
47 public class Board {
48
49   /** liefert den Eigentuemer der Pinnwand */
50   public User getOwner()
51
52   /** liefert die Nachrichten, die an der Pinnwand gepostet wurden */
```

```
53    public Message[] getMessages()
54  }
55
56  /** eine Nachricht besteht aus einem Text; jede Nachricht steht
57   * an genau einer Pinnwand; Nutzer koennen durch Druecken
58   * eines ”Gefaellt mir”-Buttons signalisieren, dass ihnen die
59   * Nachricht gefaellt */
60  public class Message {
61
62    /** liefert den Text der Nachricht */
63    public String getText()
64
65    /** liefert die Pinnwand, an der sich die Nachricht befindet */
66    public Board getBoard()
67
68    /** liefert die Nutzer, denen die Nachricht gefaellt */
69    public User[] getLikes()
70  }
```

Schreiben Sie auf der Grundlage dieser API ein Programm `LikebookTest`, bei dem der Benutzer zunächst aufgefordert wird, zwei Namen einzugeben. Das Programm soll darauf die Likebook Nutzer A und B mit den entsprechenden Namen ermitteln und dann die Anzahl bestimmen und ausgeben, wie oft Nutzer B Nachrichten an der Pinnwand von Nutzer A mit „Gefällt mir“ markiert hat. Existiert kein Nutzer mit einem der eingegebenen Namen, soll eine Fehlermeldung ausgegeben werden. Die Ausgaben sollen die folgende Form haben:

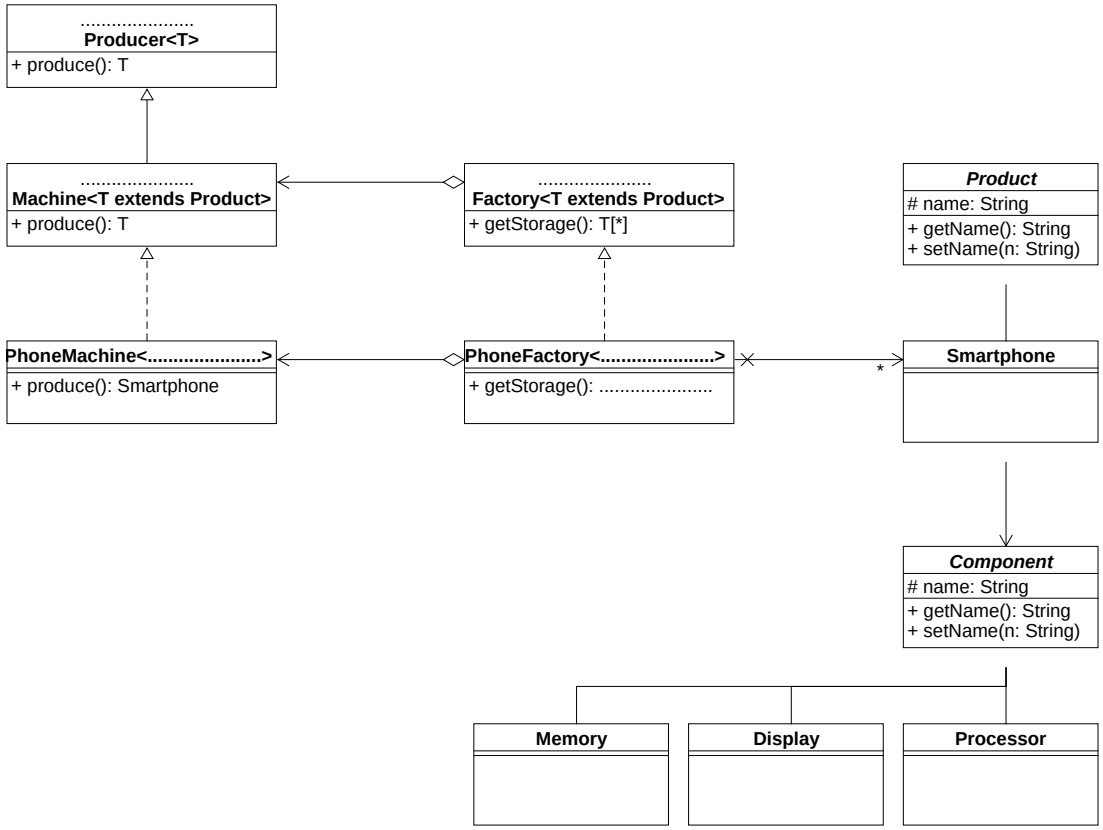
- `Dem Nutzer mit dem Namen otto gefallen 4 Nachrichten auf der Pinnwand des Nutzers mit dem Namen karl`
- `Ein Nutzer mit dem Namen maria existiert nicht in Likebook`

Zum Einlesen von Namen können Sie die statische Methode `IO.readString(String prompt)` verwenden, die einen eingelesenen `String` zurückgibt.

Aufgabe 2: *UML* **(10 Punkte)**

Im folgenden UML-Klassendiagramm fehlen einige Informationen. Ergänzen Sie

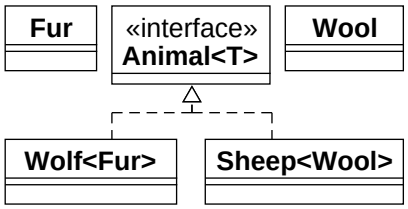
- drei Pfeil- oder Linienenden, wo die Linie nicht bis zu den Kästchen reicht
- sechs Begriffe, wo gepunktete Linien sind (…)
- Multiplizitäten und Rollenbezeichner, wo sie fehlen



Aufgabe 3: *Polymorphie* **(10 Punkte)**

Kreuzen Sie für jede Aussage entweder *Ja* oder *Nein* an. Korrekte Antworten werden positiv gewertet (+1 Punkt), falsche Antworten negativ (-1 Punkt), fehlende Antworten gehen nicht in die Bewertung ein (0 Punkte). Sie können insgesamt zwischen null und zehn Punkten erreichen, insbesondere aber *keine* negativen Punkte (auch bei überwiegend falschen Antworten).

Gegeben Sei folgende Klassenstruktur:




```
1 public class Fur { }
2 public class Wool { }
3 public interface Animal<T> { }
4 public class Wolf implements
      Animal<Fur> { }
5 public class Sheep implements
      Animal<Wool> { }
```

Wählen Sie für die folgenden Befehle aus, ob sie gültig (**Ja**) sind oder nicht (**Nein**). Befehle gelten als *gültig*, wenn sie weder zu einem Compilezeit- noch zu einem Laufzeitfehler führen.

Ja	**Nein**	
[ ]	[ ]	`Animal<Wool> a = new Sheep(); Sheep s = (Sheep)a;`
[ ]	[ ]	`Animal<Fur> a = new Wolf(); Sheep s = (Sheep)a;`
[ ]	[ ]	`Animal<Wool> a = new Sheep(); Sheep s = a;`
[ ]	[ ]	`Animal<?> a = new Wolf(); Sheep s = (Sheep)a;`
[ ]	[ ]	`Animal<Wool> a = new Wolf();`
[ ]	[ ]	`Wolf w = new Animal<Fur>();`
[ ]	[ ]	`Wolf w = new Wolf<Fur>();`
[ ]	[ ]	`Sheep s = new Sheep(); Animal<Wool> a = (Animal<Wool>)s;`
[ ]	[ ]	`Sheep s = new Sheep(); Animal<Wool> a = (Sheep)s;`
[ ]	[ ]	`Sheep s = new Sheep(); Animal<Wool> a = s;`

Aufgabe 4: *Klassen, Interfaces, Generics, Exceptions* **(19 Punkte)**

Gegeben seien folgende Interfaces und Klassen:

```
1  class Match<T> {
2    private T first;
3    private T second;
4
5    public Match(T first, T second) {
6      this.first = first;
7      this.second = second;
8    }
9
10   public T getFirst() {
11     return first;
12   }
13
14   public T getSecond() {
15     return second;
16   }
17 }
18
19 class Team {
20
21   private String name;
22
23   public Team(String name) {
24     this.name = name;
25   }
26
27   @Override
28   public String toString() {
29     return name;
30   }
31 }
32
33 class OddException extends Exception { }
34
35 interface Drawing<T> {
36   List<Match<T>> draw() throws OddException;
37 }
38
39 public class Football {
40   public static void main(String[] args) {
41     Lottery dfbPokal = new Lottery();
42     dfbPokal.add(new Team(”Bayern”));
43     dfbPokal.add(new Team(”BVB”));
44     dfbPokal.add(new Team(”Werder”));
45     dfbPokal.add(new Team(”Borussia”));
46     dfbPokal.add(new Team(”VFB”));
47     dfbPokal.add(new Team(”FC”));
48     dfbPokal.add(new Team(”Hansa”));
49     dfbPokal.add(new Team(”Eintracht”));
50     try {
51       List<Match<Team>> matches = dfbPokal.draw();
52       for (Match<Team> match : matches)
53         System.out.println(match.getFirst() + ” - ” + match.getSecond());
54     } catch (OddException e) {
55       System.out.println(”Ungerade Anzahl an Mannschaften!”);
56     }
57   }
58 }
```

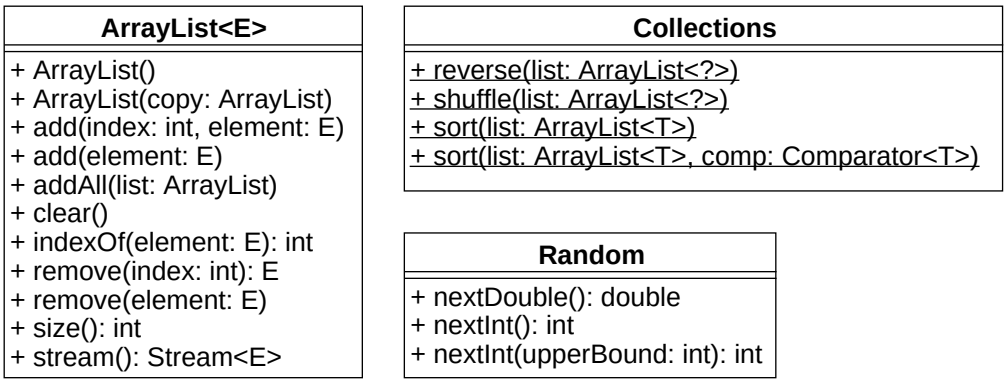
Implementieren Sie die fehlende Klasse `Lottery`. Bei einem Objekt dieser Klasse handelt es sich um eine `java.util.ArrayList`, die Objekte vom Typ `Team` speichern kann. Leiten Sie die Klasse `Lottery` also entsprechend von `ArrayList` ab.

Die Klasse `Lottery` soll weiterhin das Interface `Drawing` (dt. Auslosung) implementieren, und zwar auf folgende Art und Weise: Die Methode `draw` (dt. ziehen) soll eine `OddException` werfen, wenn sich aktuell eine ungerade Anzahl an Mannschaften in der Auslosung befindet. Befindet sich eine gerade Anzahl an Teams in der Auslosung, sollen aus den Teams per Zufall Paarungen (Klasse `Match`) gebildet und als Liste geliefert werden. Jedes Team darf dabei nur in einem einzelnen Match auftauchen.

Eine Beispielausgabe für das obige Programm wäre z.B.

```
FC - Bayern
BVB - VFB
Werder - Borussia
Hansa - Eintracht
```

Zur Erinnerung sehen Sie hier eine vereinfachte Übersicht über einige nützliche Klassen und Methoden:



Aufgabe 5: *Dynamische Programmierung* **(16 Punkte)**

Im alten China war Seide ein kostbares Gut. Ein Seidenhändler will auf einem Markt eine Seidenbahn für möglichst viel Geld verkaufen.

Gegeben ist eine Seidenbahn der Länge l. Unterschiedliche Längen von Seide kosten unterschiedlich viel, beispielsweise hat ein Stück Seide der Länge 2 den Wert 5, ein Stück Seide der Länge 3 hat den Wert 8. Die Preise sind also nicht gleichmäßig verteilt.

Gegeben ist außerdem folgendes Programm mit einer Methode
`public static int` `cutRecursive(`**`int`**`[] prices, `**`int`**` length)`,
die rekursiv den besten Preis für eine Zerstückelung einer Seidenbahn der gegebenen Länge zu den gegebenen Preisen berechnet. Dabei enthält `prices[l - 1]` den Wert für ein Stück Seide der Länge `l` (die Indizierung beginnt bei 0).

Die rekursive Implementierung ist allerdings für den praktischen Einsatz zu ineffizient. Ein erster Ansatz für eine Lösung mit Dynamischer Programmierung ist in der Methode
`public static int` `cutDynamic(`**`int`**`[] prices, `**`int`**` length)` gegeben.

Vervollständigen Sie diese Methode in den Zeilen 25, 27, 32, 34 und 39.

```
1  public class SilkCutting {
2    /**
3     * Berechnet den besten Preis fuer eine Zerstueckelung der Seidenbahn zu den
4     * gegebenen Preisen.
5     * @param prices Die Preise fuer Seiden-Stuecke verschiedener Laenge
6     * @param length Die Gesamtlaenge der verfuegbaren Seidenbahn
7     * @return den besten Preis fuer eine Zerstueckelung der Seidenbahn
8     */
9    public static int cutRecursive(int[] prices, int length) {
10     // Basisfall
11     if (length <= 0)
12       return 0;
13     int result = 0;
14     // zerschneide die Seidenbahn rekursiv in verschiedene Laengen
15     // und vergleiche
16     for (int i = 0; i < length; i++) {
17       result = Math.max(result, prices[i] +
18             cutRecursive(prices, length - i - 1));
19     }
20     return result;
21   }
22
23   public static int cutDynamic(int[] prices, int length) {
24     // Speicher fuer Memoisation: values[l] enthaelt den Wert fuer Laenge l
25     int[] values = new int[_______________];
26     // Basisfall
27     values[0] = ______;
28     // baue die Werte-Tabelle von unten her auf
29     for (int i = 1; i <= length; i++) {
30       // berechne den Wert fuer die Laenge i
31       int max = 0;
32       for (int j = ______; ____________; j++) {
33         max =
34               ____________________________________________________________;
35       }
36       values[i] = max;
37     }
38     // gib den Wert fuer die gewuenschte Laenge zurueck
39     return values[____________];
40   }
```

```
41
42    public static void main(String[] args) {
43      int[] prices = new int[] { 1, 5, 8, 9, 10, 17, 17, 20 };
44      System.out.println(cutRecursive(prices, 8)); // 22
45      System.out.println(cutDynamic(prices, 8)); // 22
46    }
47  }
```

Aufgabe 6: *Exceptions* **(10 Punkte)**

Gegeben sei folgendes Programm:

```
1  class Exc1 extends Exception {
2    public String toString() { return ”K”; }
3  }
4
5  class Exc2 extends Exception {
6    public String toString() { return ”H”; }
7  }
8
9  public class Exceptions {
10   public static int func1(int x) throws Exc1 {
11     try {
12       System.out.print(”G”);
13       x = func2(x);
14       System.out.print(”F”);
15     } catch (Exc2 exc) {
16       System.out.print(”E”);
17       return x + 3;
18     } finally {
19       System.out.print(”D”);
20     }
21     System.out.print(”C”);
22     return x + 6;
23   }
24
25   public static int func2(int x) throws Exc1, Exc2 {
26     if (x > 0) throw new Exc1();
27     if (x == 0) throw new Exc2();
28     System.out.print(”B”);
29     return -x;
30   }
31
32   public static void main(String[] args) {
33     try {
34       int i = IO.readInt();
35       int y = func1(i);
36       System.out.print(y);
37     } catch (Exc1 exc) {
38       System.out.print(exc.toString());
39     }
40     System.out.print(”A”);
41   }
42 }
```

Welche Konsolenausgabe produziert dieses Programm bei folgenden Benutzereingaben (**int**-Werte)?

a) -2 Lösung: _____

b) 0 Lösung: _____

c) 3 Lösung: _____

Aufgabe 7: *SML* **(3 + 3 + 3 Punkte)**

a) Schreiben Sie eine Funktion `max = fn : int * int -> int`, die zwei `int`-Werte als Parameter bekommt, und den größeren der beiden Werte zurückgibt. Sie dürfen annehmen, dass beide Werte **positiv** oder 0 sind, d.h. Sie müssen sich um negative Werte keine Gedanken machen.
Schreiben Sie diese Funktion **rekursiv**: Sie dürfen **keine Vergleiche** mit <, <=, >, >= oder = verwenden und Sie dürfen **keine** `if then else` Konstrukte verwenden. Sie dürfen stattdessen Rekursion und die üblichen arithmetischen Operationen (+, -, …) verwenden.

b) Schreiben Sie eine Funktion `maxList = fn : int list -> int`, die das größte Element einer Liste von `int`-Werten zurückgibt. Ist die Liste leer, soll der Wert 0 zurückgegeben werden.
Hier dürfen Sie beliebige Operationen und Funktionen verwenden, inklusive <, `if then else` und der Funktion `max` aus Teilaufgabe a), unabhängig davon, ob Sie diese Teilaufgabe gelöst haben oder nicht.

c) Gegeben sei die unten angegebene Datenstruktur `tree` für Binärbäume sowie die Funktion `depth` zur Berechnung der Tiefe eines Binärbaums.
Definieren Sie eine Funktion `balanced = fn : tree -> bool`, die angibt, ob der als Parameter übergebene Binärbaum balanciert ist, d.h. ob die Höhe des linken Teilbaums der Höhe des rechten Teilbaums entspricht.

```
- datatype tree = Leaf | Node of tree * tree;
- fun depth(Leaf) = 1
    | depth(Node(l,r)) = 1 + max(depth(l), depth(r));
val depth = fn : tree -> int
- depth(Leaf);
val it = 1 : int
- depth(Node(Node(Leaf, Leaf), Node(Leaf, Leaf)));
val it = 3 : int
```

Aufgabe 8: *I/O* **(12 Punkte)**

Gegeben sei folgende Klasse `Database`, die Namen/Wert-Paare speichern kann. Namen und Werte sind dabei `String`s, wobei die Werte (nicht jedoch die Namen) auch **`null`** sein dürfen.

Die Klasse verfügt über eine `save`-Methode zum Speichern der Daten in einer Datei und über eine `load`-Methode zum Laden der gespeicherten Daten aus einer Datei. Die `load`-Methode ist allerdings noch nicht ganz fertig – vervollständigen Sie die Methode so, dass damit eine mit der `save`-Methode geschriebene Datei gelesen werden kann.

```
1  public class Database {
2
3    private Map<String, String> values = new HashMap<>();
4    private int size = 0;
5
6    public void add(String name, String value) {
7      if (name == null) throw new IllegalArgumentException();
8      values.put(name, value);
9      size++;
10   }
11
12   public String get(String name) {
13     return values.get(name);
14   }
15
16   public Collection<String> getNames() {
17     return values.keySet();
18   }
19
20   public int getSize() {
21     return size;
22   }
23
24   public boolean save(File file) {
25     try(DataOutputStream out = new DataOutputStream(
26           new BufferedOutputStream(new FileOutputStream(file)))) {
27       out.writeInt(size);
28       for (String name : getNames()) {
29         out.writeUTF(name);
30         String value = get(name);
31         if (value == null) {
32           out.writeBoolean(false);
33         } else {
34           out.writeBoolean(true);
35           out.writeUTF(value);
36         }
37       }
38       return true;
39     } catch (FileNotFoundException e) {
40       System.err.println("File '" + file.getName() + "' could not be found!");
41       return false;
42     } catch (IOException e) {
43       System.err.println("Error writing to file '" + file.getName() + "'!");
44       return false;
45     }
46   }
47
48
49
50
```

```
51    public boolean load(File file) {
52      values.clear();
53      try(DataInputStream in = new DataInputStream(
54            new BufferedInputStream(new FileInputStream(file)))) {
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81      } catch (FileNotFoundException e) {
82        System.err.println("File '" + file.getName() + "' could not be found!");
83        return false;
84      } catch (IOException e) {
85        System.err.println("Error reading from file '" + file.getName() + "'!");
86        return false;
87      }
88    }
89  }
```