Institut für Informatik der Ludwig-Maximilians-Universität München
Prof. Martin Hofmann, PhD
Dr. Steffen Jost

WiSe 2017/18
14.02.2018
Aschermittwoch

Erstklausur zur Vorlesung

Einführung in die Programmierung

Die Bearbeitungszeit beträgt 120 min. Hilfsmittel sind nicht erlaubt, auch der Besitz ausgeschalteter elektronischer Geräte wird als Betrug gewertet. Schreiben Sie Ihren vollständigen Namen und Ihre Matrikelnummer deutlich lesbar auf dieses Deckblatt, sowie Ihren Namen in die Kopfzeile auf **jedem Blatt** der Klausurangabe! Geben Sie alle Blätter ab, lassen Sie diese zusammengeheftet! Verwenden Sie nur dokumentenechte Stifte und weder die Farben rot noch grün.

Kontrollieren Sie, ob Sie alle Aufgabenblätter erhalten haben. Aufgabenstellungen befinden sich auf den Seiten 1–15. Sie dürfen die Rückseiten für Nebenrechnungen nutzen. Falls Sie die Rückseiten für Antworten nutzen, so markieren Sie klar, was zu welcher Aufgabe gehört und schreiben Sie in der entsprechenden Aufgabe, wo alle Teile Ihrer Antwort zu finden sind. Streichen Sie alles durch, was nicht korrigiert werden soll!

Lesen Sie die Aufgabenstellungen vollständig durch, bevor Sie mit der Bearbeitung der Aufgaben beginnen. Es gibt 6 unterschiedlich gewichtete Aufgaben zu insgesamt 68 Punkten. Mit 34 Punkten haben Sie sicherlich bestanden. Die Teilaufgaben können unabhängig voneinander bearbeitet werden. Für alle (Teil-)Aufgaben mit einer richtigen Antwort zum Ankreuzen (Single-Choice) gilt, dass fehlende oder mehrfache Antworten als falsche Antwort gewertet werden. Falsche Antworten reduzieren aber nicht die Punktzahl in anderen (Teil-)Aufgaben.

Mit Ihrer Unterschrift bestätigen Sie, dass Sie zu Beginn der Klausur in ausreichend guter gesundheitlicher Verfassung sind, und diese Klausurprüfung verbindlich annehmen.

Nachname:

Vorname:

Matrikelnummer:

Studiengang:

Hiermit erkläre ich die Richtigkeit der obigen Angaben: _____
(Unterschrift)

Aufgabe	1	2	3	4	5	6	$\sum$	Bonus	Note
Punkte	12	12	12	10	8	14	**68**		
erreicht									

Aufgabe 1 (UML & Vererbung): **(12 Punkte)**

Gegeben sind folgende Klassen:

```
public class Spielzeug {
  public void play(){
    System.out.println("Whee!");
} }

public class Katze implements Tier {
  private Spielzeug spielzeug;
  @Override
  public String gibLaut() {return "Miau";}
}

public class Canis implements Tier {
  private Integer kg;

  public Canis(int kg) { this.kg = kg; }

  @Override
  public String gibLaut() {return "Grr";}

  public String gewinnt(Canis h) {
    if (this.kg  > h.kg) return "ja";
    else                 return "nein";
  }

  public void eatPrey(Tier t) {
    this.kg = kg + t.gibLaut().length();
  }

  public Integer getKg() { return kg; }
}
```

```
public interface Tier  {
  public String gibLaut();
  public default String gewinnt(Tier t) {
    if(gibLaut().length() > t.gibLaut().length())
         return "ja";
    else return "nein";
} }

public class Hund extends Canis {
  private int kg;
  private String name;

  public Hund(int kg, String name) {
    super(kg); this.name = name;
  }

  @Override
  public String gibLaut() {return "Wau!Wau!";}

  public String gewinnt(Hund h) {
    if (this.name.length() > h.name.length())
          return "ja";
    else  return "nein";
  }

  @Override
  public void eatPrey(Tier t) { kg = kg+1; }
}
```

a) Zeichnen Sie zu diesem Code das UML-Klassendiagramm!

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 1:

b) Welche Ausgaben liefert folgendes Programm:

```
Katze jule  = new Katze();
Canis wolf  = new Canis(44);
Canis sammy = new Hund(12,"sammy");
Hund  pia   = new Hund(16, "pia");


Tier[] zoo = new Tier[] {jule, sammy, wolf, pia};
for(Tier t : zoo) { System.out.println(t.gibLaut()); }

System.out.println("Sammy vs Jule: " + sammy.gewinnt(jule));
System.out.println("Jule  vs Wolf: " + jule.gewinnt(wolf));
System.out.println("Sammy vs Wolf: " + sammy.gewinnt(wolf));
System.out.println("Sammy vs Pia:  " + sammy.gewinnt(pia));

sammy.eatPrey(jule);
System.out.println("Sammy kg: "+sammy.getKg());
wolf.eatPrey(sammy);
System.out.println("Wolf  kg: "+wolf.getKg());
```

Aufgabe 2 (Backus-Naur-Form): **(12 Punkte)**

Gegeben sei folgende BNF-Grammatik mit dem Startsymbol ⟨*Programm*⟩:

$$
\begin{aligned}
\langle \textit{Programm} \rangle &::= \langle \textit{Anweisung} \rangle \text{ ";" } [\langle \textit{Programm} \rangle] \\
\langle \textit{Anweisung} \rangle &::= \langle \textit{Verzweigung} \rangle \mid \langle \textit{Variable} \rangle \text{ "="} \langle \textit{Ausdruck} \rangle \\
\langle \textit{Verzweigung} \rangle &::= \text{"switch" "(" } \langle \textit{Ausdruck} \rangle \text{ ")" "\{" } (\langle \textit{Fall} \rangle)^+ \text{ "\}"} \\
\langle \textit{Fall} \rangle &::= \text{"case" } \langle \textit{Zahl} \rangle \text{ ":" } \langle \textit{Anweisung} \rangle \text{ "break" ";"} \\
\langle \textit{Ausdruck} \rangle &::= \langle \textit{Variable} \rangle \mid [\text{"-"}] \langle \textit{Zahl} \rangle \mid \langle \textit{Ausdruck} \rangle \text{ "+" } \langle \textit{Zahl} \rangle \mid \langle \textit{Zahl} \rangle \text{ "+" } \langle \textit{Ausdruck} \rangle \\
\langle \textit{Variable} \rangle &::= \text{"x"} \mid \text{"y"} \mid \text{"z"} \\
\langle \textit{Zahl} \rangle &::= \text{"0"} \mid \text{"1"} \mid \text{"2"} \mid \text{"3"} \mid \text{"4"} \mid \text{"5"} \mid \text{"6"} \mid \text{"7"} \mid \text{"8"} \mid \text{"9"}
\end{aligned}
$$

a) Entscheiden Sie für die folgenden Wörter jeweils, ob sie in der Sprache dieser Grammatik sind. Geben Sie klar an, ob das Wort in der Sprache ist oder nicht. Begründen Sie Ihre Antwort, falls das Wort nicht in der Sprache ist. Geben Sie für Wörter, welche in der Sprache sind, eine ausführliche Ableitung an! Führen Sie dabei jeden Schritt einzeln aus! Lediglich bei der Ersetzung eines Nichtterminals durch seine Definition dürfen Sie gleich eine Auswahl der Alternative treffen und optionale Teile weglassen. Die Anführungszeichen um Terminalsymbole dürfen Sie weglassen, wenn Sie möchten.

Die Ableitung des Wortes "x" "=" "x" "+" "1" ";" hier als Beispiel:

$$
\begin{aligned}
\langle \textit{Programm} \rangle &\rightarrow \langle \textit{Anweisung} \rangle \text{ ";"} \rightarrow \langle \textit{Variable} \rangle \text{ "="} \langle \textit{Ausdruck} \rangle \text{ ";"} \rightarrow \text{"x" "="} \langle \textit{Ausdruck} \rangle \text{ ";"} \\
&\rightarrow \text{"x" "="} \langle \textit{Ausdruck} \rangle \text{ "+" } \langle \textit{Zahl} \rangle \text{ ";"} \rightarrow \text{"x" "="} \langle \textit{Ausdruck} \rangle \text{ "+" "1" ";"} \\
&\rightarrow \text{"x" "="} \langle \textit{Variable} \rangle \text{ "+" "1" ";"} \rightarrow \text{"x" "=" "x" "+" "1" ";"}
\end{aligned}
$$

(i) "x" "=" "y" "+" "5" "+" "z" ";"

(ii) "y" "=" "3" "+" "-" "1" ";"

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 2:

(iii) “`switch`” “`(`” “`x`” “`=`” “`x`” “`+`” “`3`” “`)`” “{” “`case`” “`3`” “`:`” “`z`” “`=`” 4 “`break`” “`;`” “}” “`;`”

b) Ändern Sie die Grammatik ab, so dass ⟨*Ausdruck*⟩ beliebige ganze Zahlen als Konstanten ohne führende Nullen erlaubt, d.h. erlaubt sein sollen zum Beispiel die Wörter “`1`” “`2`” “`3`”, “`0`”, “`-`” “`0`” oder “`-`” “`5`” “`6`” “`7`” “`8`”, aber nicht “`0`” “`0`” “`1`” “`2`”, oder “`1`” “`2`” “`-`” “`2`” “`3`”

Aufgabe 3 (Hoare Logik): **(12 Punkte)**

a) Entscheiden Sie jeweils ohne Angabe eines Beweises über die Gültigkeit des gegebenen Hoare-Triples. Alle Variablen `i,j,x,y,z` sind deklariert und haben den Typ `int`, sowie `a` mit Typ `int[]`. Jeweils nur eine Antwort pro Teilaufgabe ankreuzen.

(i) $\{x > 0\}$ `x=y+z;` $\{x = z + y \land z + y > 0\}$

☐ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☐ ungültiges H.-T.

(ii) $\{x = 42 \land y < 0 \land z = 7\}$ `z++; y--; x = y * z;` $\{x > 69 \lor z < 69\}$

☐ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☐ ungültiges H.-T.

(iii) $\{a = [8, 4, 2] \land i \in \{0, 1, 2\} \land j \in \{0, 1, 2\}\}$ `if(a[i]<a[j]) r=j-i; else r=i-j;` $\{r \leq 0\}$

☐ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☐ ungültiges H.-T.

(iv) $\{x = 0, y = 2, z = 4\}$ `while(x<y) {x++; z++; x--;}` $\{z = 8\}$

☐ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☐ ungültiges H.-T.

b) Beweisen Sie mit Hilfe des Hoare-Kalküls die Gültigkeit des folgenden Hoare-Tripels:

$$\{x \geq 0 \land r = 0\}\ c\ \{r = x^2\}$$

wobei alle Variablen i, r, x den Typ `int` haben und c das folgende Programmstück ist:

```
int i = 0;
while (i!=x) {
  i++;
  r = r + i;
  r = i + r;
}
r = r - x;
```

Schreiben Sie nach Ihrem Beweis bitte hier die verwendete Invariante der `while`-Schleife hin:

Hinweise: Der berühmte deutsche Mathematiker Carl Friedrich Gauß fand im Alter von 9 Jahren die Formel $\sum_{k=1}^{n} k = \frac{n \cdot (n+1)}{2}$ für $n \in \mathbb{N}$.

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 3:

c) Vervollständigen Sie die Prämissen für die Regel des Hoare-Kalküls für `for`-Schleifen:

$$\frac{}{\{P\}\ \texttt{for(}c_1\texttt{;}\ b\texttt{;}\ c_3\texttt{)}\ c_2\ \{Q\}}$$

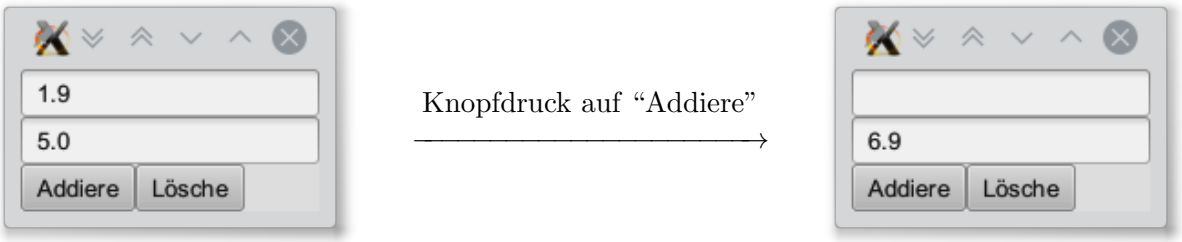
Hinweise:
Sie dürfen nur annehmen, dass die Abbruchbedingung b keinen Seiteneffekt hat. In der Vorlesung wurde diese Regel nicht behandelt! Stattdessen wurde empfohlen, eine `for`-Schleife zuerst in eine äquivalente `while`-Schleife umzuschreiben und dann die bekannte Regel dafür zu verwenden:

$$\frac{P \to I \qquad \{I \wedge b\}\ c\ \{I\} \qquad I \wedge \neg b \to Q}{\{P\}\ \texttt{while (}b\texttt{)}\ c\ \{Q\}}$$

Nutzen Sie dieses Wissen, um Ihre oben entworfene Regel für `for`-Schleifen zu überprüfen! Sie werden wahrscheinlich drei bis fünf Prämissen benötigen, mindestens zwei davon Hoare-Triple.

Aufgabe 4 (Benutzeroberflächen): **(10 Punkte)**

a) Wir wollen einen simplen Taschenrechner schreiben, welcher einfach nur Zahlen aufaddieren kann:



Die GUI besteht aus einem Textfeld zur Eingabe einer Zahl; ein Textfeld zur Anzeige der intern gespeicherten Summe; ein Knopf zum Addieren der im oberen Textfeld befindlichen Zahl zur Summe, welcher anschließend das obere Textfeld leert; und noch ein Knopf zum Rücksetzen der Summe auf 0.

Vervollständigen Sie dementsprechend das nachfolgende Programmgerüst. Folgendes ist noch zu tun:

- Der Knopf zum Addieren fehlt noch komplett. Fügen Sie diesen hinzu!
- Stellen Sie sicher, dass beide Knöpfe auch funktionieren!
  (Der in der Vorgabe enthaltene Knopf wird bis jetzt nur angezeigt, hat aber noch keine Wirkung.)

Hinweise: Zur Vereinfachung der Aufgabe wird fast alles in einer Methode implementiert. Zur vollständigen Lösung reicht es also, ein paar Zeilen an der mit `// *** TODO ***` gekennzeichneten Stelle einzufügen. Sie dürfen aber auch neue Methoden definieren, wenn Sie möchten. Die Anwendung der Entwurfsmuster MVC und/oder Observer sind nicht notwendig.

```
public class Main extends Application {
  private  double    summe;
  private  TextField eingabeFeld;
  private  TextField ausgabeFeld;
  private  Button    clrButton;
  private  Button    addButton;

  public static void main(String[] args) { launch(args); }
  public void start(Stage primaryStage) throws Exception{
    summe            = 0;
    eingabeFeld      = new TextField("");
    ausgabeFeld      = new TextField("0");
    clrButton        = new Button("Lösche");
    GridPane scheibe = new GridPane();
    scheibe.add(eingabeFeld,0,0,2,1);      // (Spalte 0, Zeile 0, Breite 2, Höhe 1)
    scheibe.add(ausgabeFeld,0,1,2,1);      // (Spalte 0, Zeile 1, Breite 2, Höhe 1)
    scheibe.add(clrButton,1,2);            // (Spalte 1, Zeile 2)
    // *** TODO ***
    primaryStage.setScene(new Scene(scheibe));
    primaryStage.show();
  }
  public double getEingabe() {  return Double.parseDouble(eingabeFeld.getText());  }
  public void   showSumme()  {  ausgabeFeld.setText(""+summe);  }
}
```

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 4:

Schreiben Sie hier Ihre Lösung zu Aufgabenteil **a)** hin:

b) Erklären Sie möglichst kurz und knapp das Entwurfsmuster “MVC”!
Ihre Antwort sollte erklären, wofür die Abkürzung steht, welche Abhängigkeiten zwischen den drei Komponenten bestehen und was Vor- und Nachteile dieses Entwurfsmusters sind.

Aufgabe 5 (Nebenläufigkeit): (8 Punkte)

Eine Variante eines beliebten Kindergeburtstagsspiels geht so: Die Kinder sitzen um einen Tisch, auf dem eine Gabel und eine Tafel Schokolade liegt. Jedes Kind hat einen Würfel und würfelt ununterbrochen. Hat es eine 6, darf es sich die Gabel nehmen—natürlich nur, wenn die Gabel frei ist. Wenn es die Gabel hat, isst es damit ein Stück der Schokolade und legt dann die Gabel wieder zurück.

Wesentliche Aspekte dieses Spiels sind in dieser Simulation repräsentiert (mit Zeilennummern):

```
10  public class Six {
11    private int forkOwner;
12
13    public void grabFork (int name) {
14        if (forkOwner == 0) {
15          System.out.println ("Kind " + name + " nimmt die Gabel.");
16          forkOwner = name;
17    }    }
18
19    public void eat (int name) {
20        if (name == forkOwner) {
21          System.out.println("Kind " + name + " isst und gibt die Gabel ab.");
22          forkOwner = 0;
23    }   }
24
25    private class Kid extends Thread {
26        private int name;
27        public Kid (int name) { this.name = name; }
28        public void run () {
29          while (true) {
30            grabFork (name);
31            eat (name);
32    }   } }
33
34    public void party() { for (int i=1; i <= 6; i++) { new Kid(i).start(); } }
35
36    public static void main (String[] args) { new Six().party(); }
37  }
```

a) Die Simulation ist noch nicht korrekt, denn bei einem Testlauf beobachten wir folgende Ausgabe:

```
Kind 2 nimmt die Gabel.
Kind 3 nimmt die Gabel.
```

Aber, nur ein Kind kann die Gabel halten! Erklären Sie, wie es zu dieser Ausgabe kommt.

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 5:

b) Um das Problem zu beheben, werden Zeilen 28–31 versuchsweise wie folgt geändert:

```
public void run() {
  while(true) {
    synchronized(this) {
      grabFork(name);
      eat(name);
} } }
```

Dies löst das Problem nicht! Erklären Sie in 2–3 Sätzen, warum obige Änderung nutzlos ist!

c) Beschreiben Sie die Programmänderungen, welche nötig sind, damit die Simulation korrekt ist.

d) Das Spiel wird nun so verändert, dass auch noch ein Messer auf dem Tisch liegt. Bei einem Sechser soll sich das Kind nun Messer und Gabel schnappen und das Stück Schokolade mit Messer und Gabel essen. Bei dieser Variante kann es zu einer Deadlock-Situation kommen. Beschreiben Sie diese:

Schlagen Sie eine simple Präzisierung der Regeln vor, die den Deadlock vermeidet.

Aufgabe 6 (Einfach verkettete Listen): (14 Punkte)

Gegeben ist folgende Klasse, welche die in der Vorlesung behandelten einfach verketteten Listen ohne Hilfsklassen implementiert (nach dem Muster der in der Vorlesung behandelten binären Bäume):

```
public class List {
  public static final List    EMPTY = null;
  public        final Integer data;
  public        final List    next;

  private List(int data, List next) { this.data = data;  this.next = next; }
  public static List add(int i, List l){ return new List(i,l); }
}
```

a) Skizzieren Sie den Speicherinhalt nach der Ausführung von folgendem Code, entweder durch ein UML-Objektdiagramm oder durch ein Speicherbild mit Pfeilen wie anfangs in den Übungen verwendet. Zur Vereinfachung dürfen `Integer` hier wie `int` ohne eigene Boxen gezeichnet werden. Etwaige vom Garbage Collector einzusammelnde Objekte müssen nicht eingezeichnet werden.

```
List l1 = List.EMPTY;
List l2 = List.EMPTY;
l1 = List.add(4,l1);
l1 = List.add(5,l1);
l1 = List.add(6,l1);
l2 = List.add(1,l2);
l2 = List.add(2,l1.next);
l2 = List.add(3,l2);
```

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 6:

b) Erweitern Sie die Klasse `List` um eine Methode `public static void print(List l)`, welche die Argumentliste der Reihe nach ausdruckt, z.B. `print(l1)` könnte ausdrucken `"6, 5, 4, "`. *Hinweis:* Wer möchte darf annehmen und verwenden, dass Teilaufgabe d richtig gelöst wurde.

c) Erweitern Sie die Klasse `List` um eine Methode

```
public static List add(int index, Integer element, List l)
```

Diese berechnet als Ergebnis eine neue Liste, bei der das gegebene Element am passenden Index einfügt wurde, z.B. `List.print(List.add(1,7,l1)` druckt `"4, 7, 5, 6, "`.
Hinweise: Es wird nur die Korrektheit bewertet, nicht die Effizienz. Die Instanzvariablen in `List` sind als `final` deklariert. Zur Vereinfachung der Aufgabe kann man dies missachten, verliert aber ein paar Punkte.

d) Wir möchten die For-Each-Schleifen-Syntax für Objekte der Klasse `List` erlauben. Dazu haben wir die erste Zeile der Klassendefinition wie folgt abgeändert:

```
public class List implements Iterable<Integer> {
```

Damit unser Programm wieder kompiliert und For-Each-Schleifen korrekt funktionieren, sind nun mehrere Dinge zu implementieren. Schreiben Sie zuerst in knappen Worten, was alles zu tun ist und tun Sie dies anschließend!

Zur Erinnerung:

```
public interface Iterable<T> {
  Iterator<T> iterator();
}
```

```
public interface Iterator<E> {
  boolean hasNext();
  E next();
}
```