

Lösung 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!";}

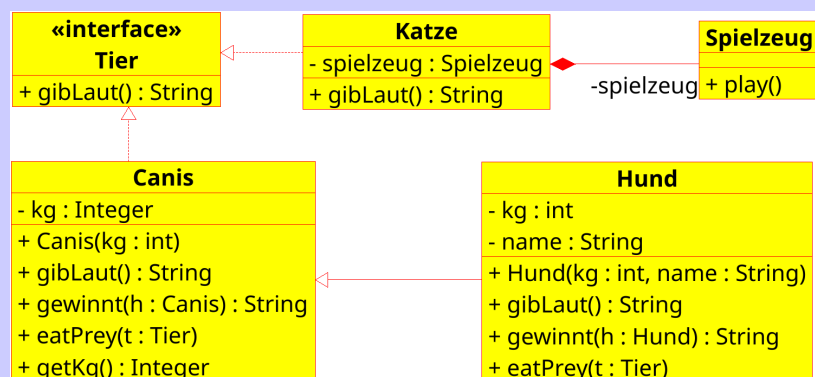
    public String gewinnt(Hund h) {
        if (this.name.length() > h.name.length())
            return "ja";
        else return "nein";
    }

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

```

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

LÖSUNG: 5 Boxen mit 4 Pfeilen: 1x Aggregation, 1x Vererbung, 2x Implementierung; Details wie Sichtbarkeit oder die Angabe von Konstruktoren war optional, musste aber einheitlich sein!



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());
```

LÖSUNG: Beim Gebell am Anfang wurde fälschlicherweise oft die Definitionsreihenfolge der Variablen benutzt, doch die Reihenfolge for-Each-Schleife war hier maßgeblich:

```
Miau
Wau!Wau!
Grr
Wau!Wau!
Sammy vs Jule: ja // Vergleich nach Klasse Tier (Laut)
Jule vs Wolf: ja // Vergleich nach Klasse Tier (Laut)
Sammy vs Wolf: nein // Vergleich nach Klasse Canis (kg)
Sammy vs Pia: nein // Vergleich nach Klasse Canis (kg)
Sammy kg: 12
Hund kg: 52
```

`sammy.eatPrey(jule);` änderte nur den `int`-Wert von `Hund`, aber `getKg()` liefert den `Integer` aus `Canis`, deshalb bleibt es bei 12!

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

Gegeben sei folgende BNF-Grammatik mit dem Startsymbol $\langle \text{Programm} \rangle$:

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

- 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 $\text{"x" "=" "x" "+" "1" ";"}$ hier als Beispiel:

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

- (i) $\text{"x" "=" "y" "+" "5" "+" "z" ";"}$

LÖSUNG: Nicht ableitbar, da für jedes "+" eine eigene Zahl vorkommen muss, aber hier ist nur eine Zahl für zwei "+" vorhanden.

- (ii) $\text{"y" "=" "3" "+" "-" "1" ";"}$

LÖSUNG: Ableitbar:

$$\begin{aligned}\langle \text{Programm} \rangle &\rightarrow \langle \text{Anweisung} \rangle \text{";" } \rightarrow \langle \text{Variable} \rangle \text{"=" } \langle \text{Ausdruck} \rangle \text{";" } \rightarrow \text{"y" "=" } \langle \text{Ausdruck} \rangle \text{";" } \\ &\rightarrow \text{"y" "=" } \langle \text{Zahl} \rangle \text{"+" } \langle \text{Ausdruck} \rangle \text{";" } \rightarrow \text{"y" "=" } \text{"3" "+" } \langle \text{Ausdruck} \rangle \text{";" } \\ &\rightarrow \text{"y" "=" } \text{"3" "+" } [\text{"-"}] \langle \text{Zahl} \rangle \text{";" } \rightarrow \text{"y" "=" } \text{"3" "+" "-" } \langle 1 \rangle \text{";" }\end{aligned}$$

Fortsetzung von Aufgabe 2:

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

LÖSUNG: Keine Ableitung möglich, da nach `switch` `(` nur ein Ausdruck stehen darf, aber keine Zuweisung.

- b) Ändern Sie die Grammatik ab, so dass $\langle \text{Ausdruck} \rangle$ 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`

LÖSUNG: Es reicht das Nonterminal $\langle \text{Zahl} \rangle$ wie folgt zu ändern:

$$\begin{aligned}\langle \text{Zahl} \rangle &::= \text{0} \mid \langle \text{ZahlN} \rangle (\langle \text{Zahl} \rangle)^* \\ \langle \text{ZahlN} \rangle &::= \text{1} \mid \text{2} \mid \text{3} \mid \text{4} \mid \text{5} \mid \text{6} \mid \text{7} \mid \text{8} \mid \text{9}\end{aligned}$$

Viele andere Lösungen waren hier möglich.

Lösung 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) $\{x > 0\} \text{ x=y+z; } \{x = z + y \wedge z + y > 0\}$
☐ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☒ ungültiges H.-T.
- (ii) $\{x = 42 \wedge y < 0 \wedge z = 7\} \text{ z++; y--; x = y * z; } \{x > 69 \vee z < 69\}$
☒ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☐ ungültiges H.-T.
- (iii) $\{a = [8, 4, 2] \wedge i \in \{0, 1, 2\} \wedge j \in \{0, 1, 2\}\} \text{ if(a[i]<a[j]) r=j-i; else r=i-j; } \{r \leq 0\}$
☒ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☐ ungültiges H.-T.
- (iv) $\{x = 0, y = 2, z = 4\} \text{ while(x<y) } \{x++; z++; x--;\} \{z = 8\}$
☒ gültiges Hoare-Tripel ☐ partiell-gültiges H.-T. ☐ ungültiges H.-T.

LÖSUNG: *Hinweis:* Der Begriff eines „partiell-gültigen Hoare-Triple“ ist komplett unsinnig! Ein gültiges Hoare-Triple $\{P\} c \{Q\}$ besagt: Wenn P gilt, dann gilt nach dem erfolgreichem Ablauf von c danach Q .

In der Vorlesung wurde auch $\{P\} c \{\text{false}\}$ besprochen, dessen Gültigkeit impliziert, dass c nicht terminiert (oder abbricht).

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

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

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:

— $x \geq 0 \wedge r = 2 \cdot \sum_{k=0}^i k \text{ ODER } r = i \cdot i(+1) \text{ ODER } r - i = i^2$

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 $n \in \mathbb{N}$.

LÖSUNG: Deutlich ausführlicher als gefordert:

$\{x \geq 0 \wedge r = 0\}$	<code>int i = 0;</code>
$\{x \geq 0 \wedge r = 0 \wedge i = 0\}$	
$\{x \geq 0 \wedge r = 2 \sum_{k=0}^i k\}$	<code>while (i!=x){</code>
$\{x \geq 0 \wedge r = 2 \sum_{k=0}^i k \wedge i! = x\}$	<code> i++;</code>
$\{x \geq 0 \wedge r = 2 \sum_{k=0}^{i-1} k\}$	<code> r = r + i;</code>
$\{x \geq 0 \wedge r = i + 2 \sum_{k=0}^{i-1} k\}$	<code> r = i + r;</code>
$\{x \geq 0 \wedge r = 2i + 2 \sum_{k=0}^{i-1} k\}$	
$\{x \geq 0 \wedge r = 2 \sum_{k=0}^i k\}$	
$\{x \geq 0 \wedge r = 2 \sum_{k=0}^i k \wedge \neg(i < x)\}$	<code> r = r - x;</code>
$\{x \geq 0 \wedge r + x = 2 \sum_{k=0}^i k \wedge \neg(i \neq x)\}$	
$\{x \geq 0 \wedge r + x = 2 \sum_{k=0}^i k \wedge i = x\}$	
$\{r + x = 2 \left(\frac{x \cdot (x+1)}{2} \right)\}$	
$\{r + x = x^2 + x\}$	
$\{r = x^2\}$	

Je nach Ansicht kann die Verwendung der „Wunsch“-Invariante $r - i = i^2$ den Beweis vielleicht etwas einfacher aussehen lassen?

Fortsetzung von Aufgabe 3:

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

$$\frac{\{P\} \ c_1 \ \{I\} \qquad \{I \wedge b\} \ c_2; c_3 \ \{I\} \qquad I \wedge \neg b \rightarrow Q}{\{P\} \ \mathbf{for}(c_1; \ b; \ c_3) \ c_2 \ \{Q\}}$$

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:

$$\frac{P \rightarrow I \quad \{I \wedge b\} \ c \ \{I\} \quad I \wedge \neg b \rightarrow Q}{\{P\} \ \mathbf{while} \ (b) \ c \ \{Q\}}$$

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.

LÖSUNG: Äquivalent auch ohne eingebaute Konsequenz-Regel

$$\frac{\{P\} \ c_1 \ \{P'\} \quad P' \rightarrow I \quad \{I \wedge b\} \ c_2; c_3 \ \{I\} \quad I \wedge \neg b \rightarrow Q}{\{P\} \ \mathbf{for}(c_1; \ b; \ c_3) \ c_2 \ \{Q\}}$$

oder auch noch ohne Kompositions-Regel

$$\frac{\{P\} \ c_1 \ \{P'\} \quad P' \rightarrow I \quad \{I \wedge b\} \ c_2 \ \{R\} \quad \{R\} \ c_3 \ \{I\} \quad I \wedge \neg b \rightarrow Q}{\{P\} \ \mathbf{for}(c_1; \ b; \ c_3) \ c_2 \ \{Q\}}$$

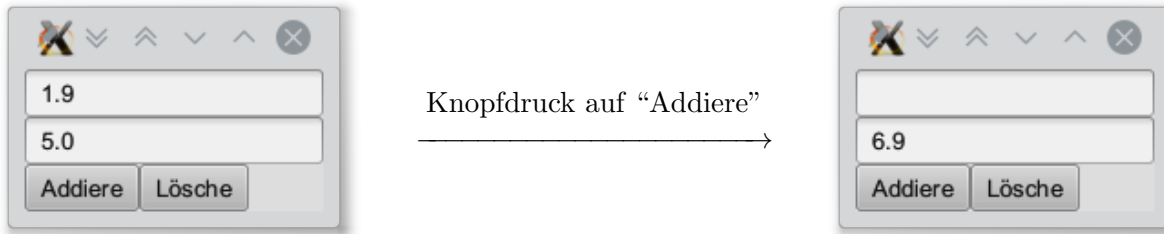
Die Idee: Zuerst wird einmalig c_1 ausgeführt. Danach muss die Invariante gelten. Vor Durchlauf des Schleifenrumpfes gelten die Invariante und Abbruchbedingung. Nach Durchlauf des Schleifenrumpfes c_2 wird aber zuerst noch c_3 ausgeführt, bevor die Abbruchbedingung b erneut geprüft wird; aber unmittelbar vor Prüfung der Abbruchbedingung muss die Schleifeninvariante wieder gelten. Wie bei der **while**-Schleife auch, muss ganz am Ende die Invariante und die negierte Abbruchbedingung stark genug sein, um die Nachbedingung zu implizieren.

Schlaue Köpfe erzielten hier auch volle Punkte mit dieser völlig korrekten Lösung:

$$\frac{\{P\} \ c_1 \ \{R\} \quad \{R\} \ \mathbf{while} \ (b) \ c_2; c_3 \ \{Q\}}{\{P\} \ \mathbf{for}(c_1; \ b; \ c_3) \ c_2 \ \{Q\}}$$

Lösung 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:

LÖSUNG: Folgende Zeilen sind bei `// *** TODO ***`

```
addButton = new Button("Addiere"); // Neuer Knopf (speichern in Instanz kein muss)
scheibe.add(addButton,0,2);        // Zur Scheibe bei Spalte 0, Zeile 2 einfügen
addButton.setOnAction(event -> {   // Event-Handler für addButton
    summe += getEingabe();           // Interne Summe erhöhen
    showSumme();                    // Summen-Anzeige aktualisieren
    eingabeFeld.setText("");        // Eingabe-Feld leeren (auch zuerst möglich)
});
clrButton.setOnAction(event -> {    // Event-Handler für clrButton
    summe=0;                        // Summe zurücksetzen
    showSumme();                    // Summen-Anzeige aktualisieren
});
```

Wer die Hausübungen zu dem Thema GUI gemacht hat, sollte hier leicht Punkte verdienen. Aber auch ohne Ahnung von GUIs konnte man zumindest mit Copy-Paste von `clrButton` zu `addButton` einen Punkt erzielen.

Es gab keinen Punktabzug, wenn man sich nicht mehr genau an `setOnAction` erinnerte, sondern `button.setIrgendwas` hinschrieb. Der Lambda-Ausdruck war kein Zwang (z.B. Anonyme Klasse).

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.

LÖSUNG: MVC steht für Modell-View-Controller. Jeder Teil des Codes gehört zu einer dieser drei Komponenten:

Das **Modell** kümmert sich um die Datenhaltung und die darauf erlaubten Operationen. Das Modell sollte völlig unabhängig von der anderen Komponenten sein.

Die **View** kümmert sich um die Darstellung der Daten des Modells und hängt davon ab. Die View kann mit Hilfe des Entwurfsmuster „Beobachter“ auch selbständig auf Änderungen am Modell reagieren.

Der **Controller** kümmert sich um den Kontrollfluss und verarbeitet Benutzereingaben, Netzwerknachrichten, etc. Entsprechend werden Methoden des Modells zu Änderung aufgerufen, und die Darstellung angepasst. Der **Controller** hängt von Modell und der View ab.

Ein großer Vorteil von MVC ist es, dass man z.B. leicht die Darstellung ändern kann, da das Modell davon unabhängig ist.

Die Trennung der Komponenten erzwingt eine klar definierte Schnittstelle zwischen diesen Komponenten und kann daher die Entwicklung durch ein mehrköpfiges Team beschleunigen.

Ein Nachteil sind ggf. Duplikation zwischen Controller und View, z.B. für JavaFX gelingt zwischen diesen beiden Komponenten keine saubere Trennung mehr (Model-View-Presenter).

Interessanterweise schrieben hier einige einen nutzlosen Aufsatz über das Observer-Pattern!?!

Lösung 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.

LÖSUNG: Kid(2) betritt Methode `grabFork`, es kommt zur ersten Ausgabezeile. Dort wird der Thread von Kid(3) unterbrochen, der auch `grabFork` betritt. Da `forkOwner` noch auf 0 steht, wird der Rumpf des Konditionals betreten und es kommt zur zweiten Ausgabezeile.

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!

LÖSUNG: Jedes Objekt der Klasse `Kid` synchronisiert nur auf sich selbst; es muss aber eine Synchronisierung auf ein gemeinsames Objekt gemacht werden.

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

LÖSUNG: Das Schlüsselwort `synchronized` in die Methodenköpfe von `grabFork` (Zeile 13) und `eat` (Zeile 19) aufnehmen.

```
public synchronized void grabFork ...
public synchronized void eat ...
```

Alternativ geht auch eine Synchronisierung wie in der vorangegangenen Teilaufgabe gezeigt, jedoch mit einem gemeinsamen Objekt. In Klasse `Six` zum Beispiel:

```
12 private static final Object LOCK = new Object();
    ...
28 public void run() {
29     while(true) {
30         synchronized(Six.LOCK) {
31             grabFork(name);
32             eat(name);
33         } } }
}
```

- 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:

LÖSUNG: Ein Kind schnappt sich das Messer, gleichzeitig schnappt sich ein anderes Kind die Gabel. Kein Kind kann Schokolade essen.

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

LÖSUNG: Bei den ersten Sechser muss sich das Kind zuerst das Messer nehmen, dann die Gabel. Ist das Messer schon weg, darf die Gabel nicht genommen werden.

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

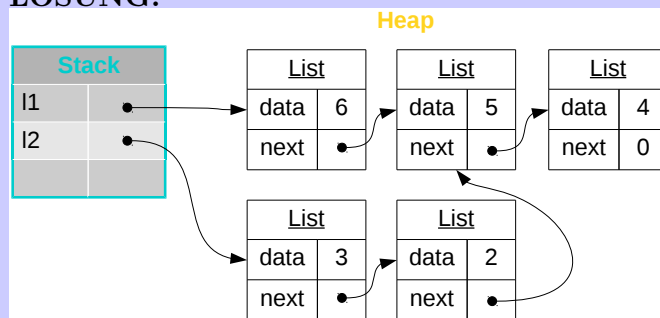
Gegeben ist folgende Klasse, welche die in der Vorlesung behandelten einfach verketteten Listen ohne Hilfsklassen implementiert (nach dem Muster der in der Vorlesung behandelten binären Bäume):

```
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);
```

LÖSUNG:

Bemerkung: In der Vorlesung wurde bei der Besprechung der binären Bäume mehrfach bemerkt, dass man auch Listen nach diesem Muster (statische Methoden und dafür keine Wrapper-Klasse) implementieren kann. Wer beides verfolgt hatte, sollte damit eigentlich keine Probleme mehr haben.

Doch schon das Speicherbild bereitete erhebliche Probleme, obwohl der extrem kurze gegebene Code viel einfacher ist (nur wenige Sprach-Features) als der Code zu dem in der Übung komplexe Speicherbilder gefordert waren!

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 ausdrückt, z.B. `print(l1)` könnte ausdrucken "6, 5, 4, ".
Hinweis: Wer möchte darf annehmen und verwenden, dass Teilaufgabe d richtig gelöst wurde.

LÖSUNG: Direkte Lösung:

```
public static void print(List l) {
    while(l != null) {
        System.out.print(l.data, i+", ");
        l = l.next;
    }
}
```

Mit Hilfe der Lösung von d:

```
public static void print(List l) {
    for(Integer i : l) { System.out.print(i+", "); }
}
```

- 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 eingefügt 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.

LÖSUNG:

```
public static List add(int index, Integer element, List l){
    if(index==0) { return new List(element,l); }
    else {
        List tail = add(index-1, element, l.next);
        return new List(l.data,tail);
        // Hinweis: Diese recht einfache Lösung ist nicht besonders effizient,
        // wegen dem hohen Stack-Verbrauch (Stichwort "endrekursion", nicht in EiP behandelt)
        // Eine bessere Lösung mit Schleife ist aber auch nicht so schwer zu finden.
    }
}
```

Die Schwierigkeit mit `final` erfordert, dass man den Anfang der Liste kopieren muss, aber dies konnte man ja zur Vereinfachung ignorieren, wie in der Aufgabenstellungen vorgeschlagen.

- 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();  
}
```

LÖSUNG: In der Klasse `List` ist die Methode `iterator` zu implementieren. Dazu müssen wir eine neue Klasse anlegen, welche das Interface `Iterator` passend implementiert, wozu die beiden Methoden `hasNext` und `next` zu implementieren sind.

Innerhalb von `List`:

```
public Iterator iterator() { return new Iter(this); }
```

Neue Klasse `Iter`:

```
public class Iter implements Iterator<Integer> {  
  
    List position;  
  
    public Iter(List position) { this.position = position; }  
  
    @Override  
    public boolean hasNext() { return position != null; }  
    @Override  
    public Integer next() {  
        if (position == null) throw new NoSuchElementException();  
        List current = position; position = position.next; return current.data;  
    } }  
}
```