Prof. Martin Hofmann, PhD
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
5. Februar 2018

Lösungsvorschlag zur 14. Übung zur Vorlesung

Einführung in die Programmierung

Auf den folgenden Seiten finden Sie eine alte Klausur aus einem vergangenen Semester.

Bearbeiten Sie diese **vor Ihrer Übung**,

so dass in der Übungsstunde genug Zeit bleibt, diese Probeklausur ausführlich zu besprechen!

Diese Probeklausur ist natürlich nur von beispielhafter Natur. Eine Klausur kann immer nur eine Auswahl der behandelten Themen abfragen. Diese Klausur berührt Themen wie Exceptions oder Interfaces kaum und Themen wie OO-Design, GUI, Sortieren oder verkettete Listen überhaupt nicht — alle Themen der Vorlesung können in den echten Klausuren drankommen! Natürlich fallen dann andere Aufgaben weg, damit der Gesamtumfang gleich bleibt.

Auch zu diesem Übungsblatt wird eine Musterlösung per UniWorX herausgegeben werden.

Organisatorische Hinweise zur Klausur

- Eine Klausuranmeldung per UniworX ist zur Teilnahme zwingend erforderlich. Die Anmeldefirst ist bereits abgelaufen. Eine Abmeldung ist bis 12.2. möglich. Eine komplette Abmeldung von der Veranstaltung ist nicht notwendig, da wir ohnehin nur die zur Klausur angemeldeten Teilnehmer an das Prüfungsamt melden. Es werden allerdings alle zur Klausur angemeldeten Teilnehmer gemeldet, welche unentschuldigt nicht erscheinen!
- Jeder Student muss einen gültigen Lichtbildausweis **und** Studentenausweis mitbringen.
- Bitte beachten Sie die Hinweise zur Klausur auf der Vorlesungshomepage und auf der Klausur selbst (siehe nächste Seite)! Es ist immer wieder verwunderlich, wenn Teilnehmer trotz all dieser Hinweise z.B. wegen Einsatz falscher Stifte Punktabzug bekommen!
- Papier wird von uns gestellt und darf nicht mitgebracht werden.

  Am Platz darf sich nur ein paar Stifte ohne Mäppchen/Etui und eventuell ein Getränk in einer durchsichtigen Flasche befinden. Wir übernehmen keinerlei Haftung für Ihre Garderobe am Rand des Hörsaals.
- Taschen und Jacken müssen vorne an der Tafel abgelegt werden. Sollte jemand ein Telefon, mp3-Player, oder Ähnliches am Platz haben, ist das ein Täuschungsversuch, der dem Prüfungsausschuss gemeldet wird. Sollte ein Telefon klingeln, ist das eine Störung des Prüfungsablaufs und hat den Ausschluss von der weiteren Teilnahme zur Folge.
- Gehen Sie rechtzeitig vor Beginn in den zugewiesenen Raum! Die Raumeinteilung wird erst 2–3 Tage vor Klausurbeginn auf der Vorlesungshomepage bekanntgegeben.

Keine Abgabe

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

WiSe 2015/16
22.03.2016

Lösung zur Nachklausur 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 1–11. 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 72 Punkten. Mit 36 Punkten haben Sie sicherlich bestanden. Die Teilaufgaben können unabhängig voneinander bearbeitet werden. Für alle (Teil-)Aufgaben mit Antworten zum Ankreuzen (Multiple-Choice) gilt, dass falsche Antworten die erzielte Punktzahl dieser Aufgabe reduziert. Unbeantwortete Fragen zählen 0 Punkte und reduzieren die Punktzahl nicht. Im schlimmsten Fall bekommt man 0 Punkte auf die jeweilige Aufgabe.

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	12	12	12	**72**		
erreicht									

LÖSUNG:

A1: UML-Klassendiagramm und ein Speicherbild sollte nicht überraschend gewesen sein. Bei Aufgabenteil b) hatten erschreckend viele Probleme damit, einen einfachen Methodenaufruf in Java hinzuschreiben: weder `insert(Node 32);` noch `taa.insert(32).insert(64);` sind korrekt! Verbotene Aufrufe der *privaten* Methoden `place` oder der direkte Zugriff auf die *privaten* Instanzvariablen waren häufige Fehler. Das es sich bei der Datenstruktur um einen missgebildeten Binär-Baum handelte, war dagegen zur Lösung der Aufgabe weitgehend irrelevant.

A2: Diese Aufgabe behandelte nacheinander die Themen: Methodenaufruf und Seiteneffekte, Überladen, Arrays und Exceptions. Jedes richtige Kreuz bringt 3 Punkte, jedes falsche Kreuz oder ungültige Mehrfach-Kreuze -1. Für das Erkennen der vier "schwergewichtigen" Bands innerhalb der Antwortmöglichkeiten gab es leider keine Bonuspunkte.

A3: In dieser Aufgabe erzielten die meisten die volle Punktzahl. Überraschenderweise wurden die meisten Punkte in den ersten beiden Aufgabenteilen verschenkt. Die von uns als "schwerer" eingestufte dritte Frage wurde dagegen fast immer richtig gelöst.

A4: Kommentar zur Aufgabe befindet sich innerhalb der Lösung.

A5: Der schwierigste Teil der ersten Teilaufgabe war offenbar (ii), da hier die Verzahnung recht verzwickt ist. (iv) ist offensichtlich unmöglich, da ja die Ausgaben immer vor einer Zustandsänderung erfolgen. (v) ist genau wegen der Synchronisierung von `umpacken` unmöglich.

Enttäuschend war die geringe Anzahl an richtigen Lösungen zum zweiten und vierten Aufgabenteil. Vielleicht sollte man in der Vorlesung zuerst und primär `synchronized(object){...}` verwenden, anstatt Methoden als Synchronisiert zu deklarieren?

A6: Ankreuzen gültiger Hoare-Tripel kam sehr gut an; die verschachtelte `while`-Schleife eher weniger. Allerdings sieht die nur schwer aus, den wer eine `while`-Schleife behandeln kann, schafft auch zwei - es ist ja schließlich genau das gleiche zu tun! Die gesuchten Invarianten waren ja praktisch vorgegeben: man musste nur die beiden 0 in der Vorbedingung durch i und k ersetzen, sobald diese im Scope waren.

Die Bedingung $z = 42$ musste man einfach nur überall mit hinschreiben: Der Code berührt die Variable z ja nirgends, d.h. man muss den Zustand von z einfach nur Durchschleifen. Für diese grundlegende Erkenntnis des Hoare-Kalküls gab es 1 Punkt.

Lösung Aufgabe 1 (UML & Speicherdiagramm): **(12 Punkte)**

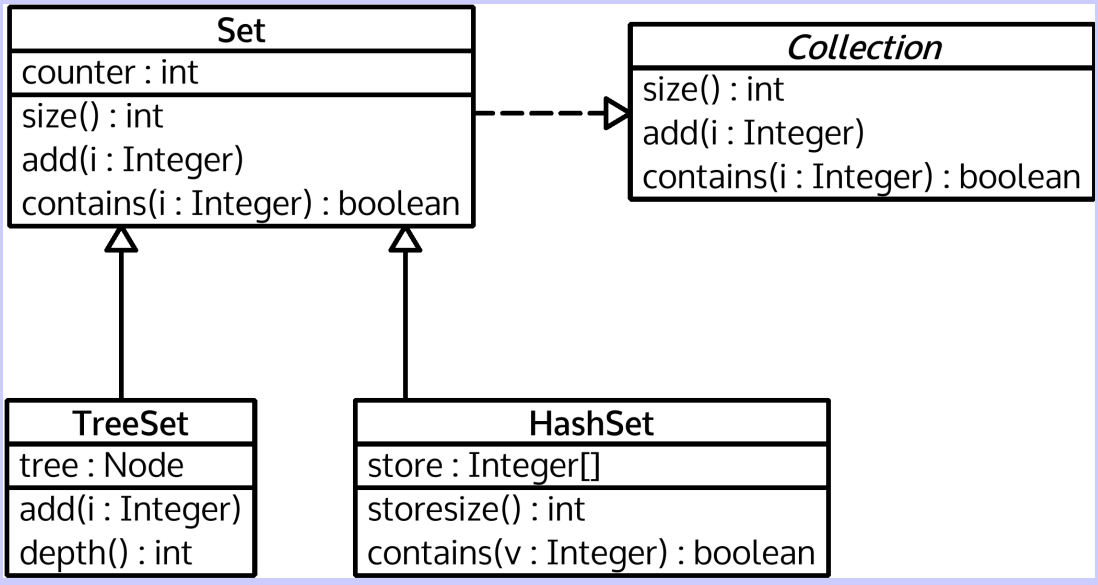
a) Zeichnen Sie das UML Klassendiagramm zu folgenden 4 Klassen:

```
public interface Collection {
  int     size();
  void    add(Integer i);
  boolean contains(Integer i);
}


public class Set implements Collection {
  private int counter;

  public Set() { counter=0; }

  @Override
  public int size() { return counter; }

  @Override
  public void add(Integer i) {
    counter++;  // INCOMPLETE
  }

  @Override
  public boolean contains(Integer i) {
    return false;  // INCOMPLETE
  }
}
```

```
public class HashSet extends Set {
  private Integer[] store;

  public HashSet() { store = new Integer[42]; }
  public int storesize() { return 42; }
  @Override
  public boolean contains(Integer v) {
    for (int i=0; i<store.length; i=i+1){
      if (v.equals(store[i])) return true;
    }
    return false;
} }

public class TreeSet extends Set {
  private Node tree;

  public TreeSet() { super(); }
  @Override
  public void add(Integer i) {
    super.add(i);
    tree.insert(i);
  }
  public int depth() {
    return 0;  //UNFINISHED
} }
```

Hinweis: Klasse `Node` aus Teilaufgabe b soll im Klassendiagramm nicht mit abgebildet werden.

LÖSUNG:



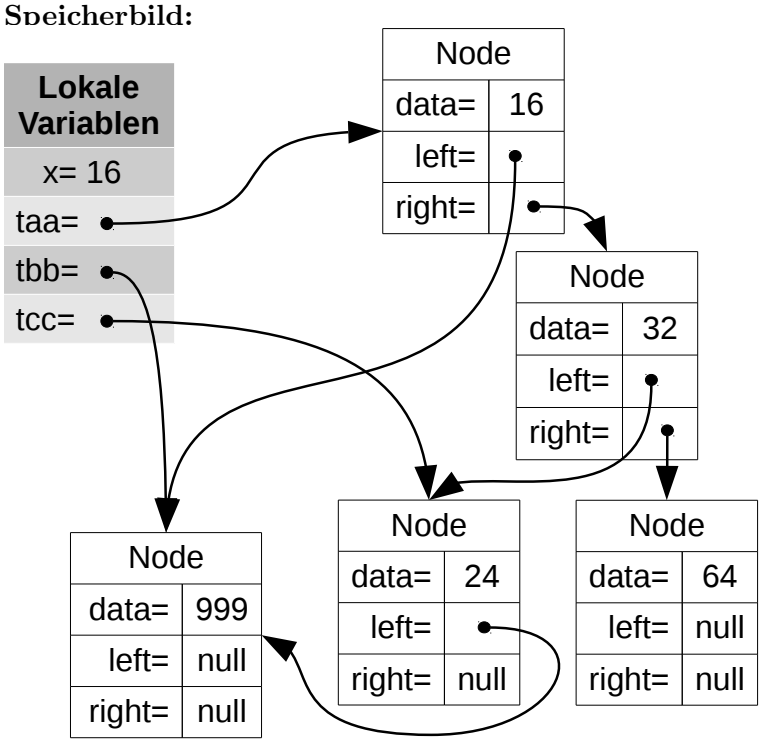
b) Gegeben ist die neben angegebene Definition der Klasse `Node`, eine unvollständige `main`-Methode aus einer anderen Klasse, sowie das unten gezeigte Speicherdiagramm.
Fügen Sie Anweisung zur `main`-Methode hinzu, so dass das gegebene Speicherdiagramm zu der markierten Stelle passt!
Genauer: Beginnend mit einem leeren Speicher soll nach Ablauf der von ihnen erweiterten `main`-Methode der Speicherinhalt mit dem Speicherdiagramm übereinstimmen. Den gegebenen Code dürfen Sie nicht verändern; Sie dürfen nur Anweisungen in `main` hinter dem Kommentar einfügen. Verwenden Sie möglichst wenige Anweisungen. Achten Sie drauf, dass auch die angegebenen lokalen Variablen exakt[a] übereinstimmen.

```
public class Main {
  public static void
    main(String[] args) {
      int   x  = 2 * 8;
      Node taa = new Node(x);
      // Anweisungen ab hier anfügen!


      taa.insert(32);

      taa.insert(64);

      Node tbb =
        new Node(Node.UNDEFINED);

      taa.setLeft(tbb);

      Node tcc = new Node(24);

      tcc.setLeft(tbb);

      taa.getRight().setLeft(tcc);


      // !!! Speicherbild hier !!!
      System.out.println("So isses!");
}   }
```

[a] Parameter `args` zur Vereinfachung ignoriert.

```
public class Node {
  public final int UNDEFINED = 999;
  private int data;
  private Node left;
  private Node right;

  public Node(int data) {
    this.data  = data;
    this.left  = null;
    this.right = null;
  }

  public Node getRight() { return right; }

  public void setLeft(Node node) { left = node; }

  public void insert(int value) {
    if (value > data) right = place(value,right);
    else              left  = place(value, left);
  }

  private Node place(int value, Node node) {
    if (node==null) { return new Node(value); }
    else {
      node.insert(value);
      return node;
    }
} }
```

Speicherbild:



Lösung Aufgabe 2 (Programmverständnis): **(12 Punkte)**

Welche Ausgabe wird jeweils von dem gegebenen Programm erzeugt? (Jeweils max. 1 von 4 ankreuzen.)

a)
```
public class Main {
  public static String c(){
      System.out.print("b");
      return "e";
  }
  public static void main(String[] args) {
      String c = "c";
      String d = "d";
      System.out.print(("a"+c())+((d)+"c"));
} }
```

Ausgabe ist:

- [ ] bacdc
- [ ] aedc
- [X] baedc
- [ ] acdc

---

b)
```
public class Main {
 public static String foo(Integer i,Double d){return "Hallo";}
 public static String foo(Double d, Object o){return "Hello";}
 public static String foo(Object d){ return " world"; }
 public static String foo(String s){ return "ween"; }
 public static void main(String[] args) {
   String res = foo(6.0,9.0)+foo(3.0+"")+"";
   System.out.println(res);
}}
```

Ausgabe ist:

- [ ] Hallo world
- [ ] Halloween
- [ ] Hello world
- [X] Helloween

---

c)
```
public class Main {
  public static void main(String[] args) {
    int[]idx  =
      new int[]{  6 , 5 , 2 , 7 , 9 ,10 , 1 , 8 , 4 , 3 ,10 };
    char[]val =
      new char[]{'d','a','i','m','e','m','a','d','d','i','n'};
    int i = 1;
    String result = "";
    while (i < idx.length) { result += val[idx[i]];
                             i = idx[i]+1;          }
    System.out.println(result);
} }
```

Ausgabe ist:

- [ ] mai
- [ ] admin
- [ ] aaimdm
- [X] maiden

---

d)
```
public class Fehler extends Exception { }
public class Main {
  public static void main(String[] args) {
    try {
      System.out.print("Mot");
      if (0==0.0) throw new Fehler();
      System.out.print("kraft");
    } catch (NumberFormatException n) {
      System.out.print("ten");
    } catch (Fehler f) {
      System.out.print("or");
    } catch (Exception e) {
      System.out.print("örhead");
    } finally {
      System.out.print("rad");
} } }
```

Ausgabe ist:

- [X] Motorrad
- [ ] Motorkraftrad
- [ ] Motten
- [ ] Motörhead

Lösung Aufgabe 3 (Backus-Naur-Form): **(12 Punkte)**

Gegeben sei folgende BNF-Grammatik mit dem Startsymbol $\langle Ausdruck \rangle$:

$$\begin{aligned}
\langle Ausdruck \rangle &::= \langle BinOp \rangle \mid [\text{“!”}]\,\langle Term \rangle \\
\langle Term \rangle &::= \langle Atom \rangle \mid \text{“(”}\ \langle Ausdruck \rangle\ \text{“)”} \\
\langle BinOp \rangle &::= \langle Ausdruck \rangle\ \text{“\&\&”}\ \langle Term \rangle \mid \langle Ausdruck \rangle\ \text{“||”}\ \langle Term \rangle \\
\langle Atom \rangle &::= (\langle Buchstabe \rangle)^+ \mid \text{“true”} \mid \text{“false”} \\
\langle Buchstabe \rangle &::= \text{“x”} \mid \text{“y”} \mid \text{“z”} \\
\langle Funktion \rangle &::= \text{“f”} \mid \text{“g”}
\end{aligned}$$

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. Die Anführungszeichen um Terminalsymbole dürfen Sie weglassen, wenn Sie möchten.

Die Ableitung des Wortes “true” “&&” “x” hier als Beispiel:

$$\begin{aligned}
&\langle Ausdruck \rangle \rightarrow \langle BinOp \rangle \rightarrow \langle Ausdruck \rangle\ \text{“\&\&”}\ \langle Term \rangle \rightarrow [\text{“!”}]\,\langle Term \rangle\ \text{“\&\&”}\ \langle Term \rangle \\
&\quad \rightarrow \langle Term \rangle\ \text{“\&\&”}\ \langle Term \rangle \rightarrow \langle Term \rangle\ \text{“\&\&”}\ \langle Atom \rangle \rightarrow \langle Term \rangle\ \text{“\&\&”}\ (\langle Buchstabe \rangle)^+ \\
&\quad \rightarrow \langle Term \rangle\ \text{“\&\&”}\ \langle Buchstabe \rangle \rightarrow \langle Term \rangle\ \text{“\&\&”}\ \text{“x”} \rightarrow \langle Atom \rangle\ \text{“\&\&”}\ \text{“x”} \rightarrow \text{“true”}\ \text{“\&\&”}\ \text{“x”}
\end{aligned}$$

a) “(” “!” “true” “)”

LÖSUNG:

$$\begin{aligned}
\langle Ausdruck \rangle &\rightarrow [\text{“!”}]\,\langle Term \rangle \rightarrow \langle Term \rangle \rightarrow \text{“(”}\ \langle Ausdruck \rangle\ \text{“)”} \rightarrow \text{“(”}\ [\text{“!”}]\,\langle Term \rangle\ \text{“)”} \\
&\rightarrow \text{“(”}\ \text{“!”}\ \langle Term \rangle\ \text{“)”} \rightarrow \text{“(”}\ \text{“!”}\ \langle Atom \rangle\ \text{“)”} \rightarrow \text{“(”}\ \text{“!”}\ \text{“true”}\ \text{“)”}
\end{aligned}$$

b) “!” “y” “||” “z”

LÖSUNG:

$$\begin{aligned}
&\langle Ausdruck \rangle \rightarrow \langle BinOp \rangle \rightarrow \langle Ausdruck \rangle\ \text{“||”}\ \langle Term \rangle \rightarrow [\text{“!”}]\,\langle Term \rangle\ \text{“||”}\ \langle Term \rangle \\
&\quad \rightarrow \text{“!”}\ \langle Term \rangle\ \text{“||”}\ \langle Term \rangle \rightarrow \text{“!”}\ \langle Atom \rangle\ \text{“||”}\ \langle Term \rangle \rightarrow \text{“!”}\ \langle Atom \rangle\ \text{“||”}\ \langle Atom \rangle \\
&\quad \rightarrow \text{“!”}\ (\langle Buchstabe \rangle)^+\ \text{“||”}\ (\langle Buchstabe \rangle)^+ \rightarrow \text{“!”}\ \langle Buchstabe \rangle\ \text{“||”}\ \langle Buchstabe \rangle \\
&\quad \rightarrow \text{“!”}\ \text{“y”}\ \text{“||”}\ \text{“z”}
\end{aligned}$$

c) “true” “&&” “false” “||” “true”

LÖSUNG:

$$\begin{aligned}
&\langle Ausdruck \rangle \rightarrow \langle BinOp \rangle \rightarrow \langle Ausdruck \rangle\ \text{“||”}\ \langle Term \rangle \rightarrow \langle BinOp \rangle\ \text{“||”}\ \langle Term \rangle \\
&\quad \rightarrow \langle Ausdruck \rangle\ \text{“\&\&”}\ \langle Term \rangle\ \text{“||”}\ \langle Term \rangle \rightarrow [\text{“!”}]\,\langle Term \rangle\ \text{“\&\&”}\ \langle Term \rangle\ \text{“||”}\ \langle Term \rangle \\
&\quad \rightarrow \langle Term \rangle\ \text{“\&\&”}\ \langle Term \rangle\ \text{“||”}\ \langle Term \rangle \rightarrow \langle Atom \rangle\ \text{“\&\&”}\ \langle Atom \rangle\ \text{“||”}\ \langle Atom \rangle \\
&\quad \rightarrow \text{“true”}\ \text{“\&\&”}\ \text{“false”}\ \text{“||”}\ \text{“false”}
\end{aligned}$$

Lösung Aufgabe 4 (Vererbung): **(12 Punkte)**

Gegeben sind folgende Klassen:

```
public class Bankkonto implements Comparable {
  private int kontonummer;

  public Bankkonto(int knr) {
    this.kontonummer = knr;
  }
  public int getKontonummer() {
    return kontonummer;
  }
  @Override
  public int compareTo(Object o) {  // TODO
    return 0;
  }
  @Override
  public String toString() {
    return "Bankkonto("+ kontonummer + ")";
} }
```

```
public class Girokonto extends Bankkonto {
  public Girokonto(int kontonummer) {
    super(kontonummer);
} }

public class Sparkonto extends Bankkonto {
  private int zinssatz;

  public Sparkonto(int knr, int zinssatz){
    super(knr);
    this.zinssatz = zinssatz;
} }
```

a) Welche der folgenden Anweisung erzeugen Typfehler:

(i)	`Object o = new Bankkonto(4);`	[X]	Okay	[ ] Typfehler
(ii)	`Bankkonto b = new Girokonto(7);`	[X]	Okay	[ ] Typfehler
(iii)	`Sparkonto s = new Bankkonto(5);`	[ ]	Okay	[X] Typfehler
(iv)	`Girokonto g = new Sparkonto(2,7);`	[ ]	Okay	[X] Typfehler

b) Die Anweisung `System.out.println(new Girokonto(9));` gibt `"Bankkonto(9)"` aus. Ändern Sie die Klasse `Girokonto` so ab, dass `"Girokonto(9)"` ausgegeben wird. Die Klasse `Bankkonto` dürfen Sie nicht verändern!

LÖSUNG: Wir fügen zur Klasse `Girokonto` hinzu:

```
@Override
public String toString() {
  return "Girkonto("+ this.getKontonummer() + ")";
}
```

Wichtig: `return "Girkonto("+ this.kontonummer + ")";` geht hier gerade nicht!

Gefährlich umständliche Lösungen, bei denen `Girokonto` eine eigene Instanzvariable mit der Kontonummer spendiert wurde, haben wir auch akzeptiert, sofern dann auch der Konstruktor entsprechend korrekt mit abgeändert wurde.

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 4:

c) Vervollständigen Sie die Methode `compareTo` innerhalb der Klasse `Bankkonto`, so dass Bankkonten nach Ihrer Kontonummer verglichen werden.

Beispiel: `(new Bankkonto(7)).compareTo(new Bankkonto(3))` evaluiert zu -1.

LÖSUNG:

```
@Override
public int compareTo(Object o) {
  if (o instanceof  Bankkonto){
      Integer otherknr = ((Bankkonto) o).kontonummer;
      return otherknr.compareTo(kontonummer);
  } else {
    throw new IllegalArgumentException();
} }
```

Die Verwendung von `instanceof`, `compareTo` und das Werfen der Ausnahme wurden nicht gefordert. Auch folgende Lösung erzielt die vollen Punktzahl:

```
public int compareTo(Object o) {
  Bankkonto other = (Bankkonto) o;
  if      (kontonummer == other.kontonummer) return  0;
  else if (kontonummer >  other.kontonummer) return -1:
  else return 1;
}
```

Für die Bewertung spielte es auch keine Rolle, ob die implementierte Ordnung "größer als" oder "kleiner als" implementierte, d.h. $+1$ und -1 durfte man vertauschen. Wichtig war, das überhaupt eine Ordnung implementiert wurde: `b1.compareTo(b2) == -b2.compareTo(b1)`. In Java ergibt `compareTo` auf `Integer`-Objekten mit Werten 7 und 3 auch $+1$, doch eine Ordnung auf Konten können wir natürlich auch umgekehrt definieren – je nachdem, was wir modellieren wollen (z.B. `compareTo` als "älter" – eine kleinere Kontonummer bedeutet meistens auch "älter").

d) Gegeben sind zwei korrekte Methoden zum Sortieren von Array-Listen. Die beiden Methoden unterscheiden sich nur durch Ihre Typ-Signaturen:

```
public static void barSort(ArrayList<          Bankkonto> bls) { }
public static void fooSort(ArrayList<? extends Bankkonto> bls) { }
```

Erklären Sie den Unterschied zwischen `barSort` und `fooSort`!

LÖSUNG: Die Methode `barSort` akzeptiert nur Listen der Klasse `Bankkonto`. Dagegen akzeptiert die Methode `fooSort` auch Listen von Erben von `Bankkonto`, also zum Beispiel eine Liste über `Girokonto`.

Lösung Aufgabe 5 (Nebenläufigkeit): **(12 Punkte)**

```
public class Packer implements Runnable {
  String name;
  String ding;
  Koffer koffer;

  public Packer(
    String n, String d, Koffer k) {
      name = n; ding = d; koffer = k;
  }
  @Override
  public synchronized void run() {
    while (true) {
      if (koffer.auspacken().equals(""))
      { System.out.println(name +
          " packt " + ding + " hinein.");
        koffer.einpacken(ding);
      } else {
        System.out.print(name +
          " packt " + ding +" ein und ");
        ding = koffer.umpacken(ding);
        System.out.println(ding+" aus.");
      }
    }
  }
}
```

```
public class Koffer {
  private String inhalt;

  public Koffer() { inhalt = ""; }
  public String auspacken() {return inhalt;}
  public void einpacken(String sache) {
   inhalt = sache;
  }
  public synchronized
    String umpacken(String herein) {
      String heraus = inhalt;
      inhalt = herein;
      return heraus;
} }
public class Main {
  Koffer k = new Koffer();
  Packer pa = new Packer("A","hut",k);
  Packer pb = new Packer("B","ski",k);
  Packer pc = new Packer("C","axt",k);
  (new Thread(pa)).start();
  (new Thread(pb)).start();
  (new Thread(pc)).start();
  while(true){
    System.out.println("Koffer: "+k.auspacken());
} }
```

a) Welche der folgenden Ausgaben sind jeweils nach einem frischem Programmstart möglich:

(i)
```
C packt axt hinein.
B packt ski ein und axt aus.
Koffer: ski
A packt hut ein und ski aus.
```
Ausgabe ist
[X] möglich [ ] unmöglich.

(ii)
```
A packt hut hinein.
B packt ski hinein.
Koffer: hut
C packt axt ein und ski aus.
```
Ausgabe ist
[X] möglich [ ] unmöglich.

(iii)
```
A packt hut hinein.
A packt hut ein und hut aus.
Koffer: hut
A packt hut ein und hut aus.
```
Ausgabe ist
[X] möglich [ ] unmöglich.

(iv)
```
B packt ski ein und axt aus.
C packt axt hinein.
A packt hut ein und ski aus.
Koffer: hut
```
Ausgabe ist
[ ] möglich [X] unmöglich.

(v)
```
A packt hut hinein.
B packt ski ein und hut aus.
C packt axt ein und hut aus.
Koffer: axt
```
Ausgabe ist
[ ] möglich [X] unmöglich.

(vi)
```
A packt hut hinein.
C packt axt hinein.
A packt hut ein und axt aus.
```
Ausgabe ist
[X] möglich [ ] unmöglich.

Die Aufgabe wird auf dem nächsten Blatt fortgesetzt.

Fortsetzung von Aufgabe 5:

b) Eine seltene Race Condition verursacht bei einem Test folgende hässliche Ausgabe:

```
Koffer: ski
B packt axt ein und A packt hut ein und axt aus.
ski aus.
Koffer: hut
```

Die mittleren beiden Zeilen werden von einer Methode erzeugt, welche bereits `synchronized` ist! Warum ist diese Ausgabe dennoch möglich? *Genauer:* Wie kann man dies mit korrekter Synchronisierung verhindern? (Die `print`-Anweisungen bleiben unverändert an gleicher Stelle.)

LÖSUNG: Die Synchronisierung findet auf *verschiedenen* Lock-Objekten statt: Das Schlüsselwort `synchronized` synchronisiert eine Methode auf `this`, also das Objekt, mit dem diese Methode aufgerufen wird. In diesem Fall also jeweils eines der `Packer`-Objekte. Um dies zu verhindern, müssten alle Packer auf ein gemeinsames Objekt synchronisieren.

c) Kann es in dem gegeben Programm zu einem Deadlock kommen, so dass keine Ausgabe mehr stattfindet? Begründen Sie anschließend Ihre Anwort mit 2–4 Sätzen!

☐ Ja, Deadlock ist möglich, weil... ☒ Nein, es kann kein Deadlock auftreten, weil...

LÖSUNG: Nein, kein Deadlock, es findet eine kontinuierliche Ausgabe statt.

Die einzige Möglichkeit, dass die `run`-Methode der `Packer`-Objekte angehalten wird, ist durch Aufruf der synchronisierten Methode `umpacken`. Diese Methode kann jedoch immer bis zum Ende durchlaufen, so dass das Lock des `Koffer`-Objekts immer wieder frei wird.

d) Die Methode `wait()` darf nur innerhalb einer synchronisierten Methode (oder innerhalb eines synchronisierten Blocks) aufgerufen werden. Warum? Was ist der Sinn eines Aufrufs von `wait()`?

LÖSUNG: Der Sinn von `wait` besteht darin, ein gehaltenes Lock temporär frei zu gegeben. Außerhalb eines synchronisierten Methode/Programmblocks wird kein Lock gehalten, das frei gegeben werden könnte.

Lösung Aufgabe 6 (Hoare Logik): **(12 Punkte)**

a) Entscheiden Sie jeweils ohne Angabe eines Beweis, ob es sich um ein Hoare-Tripel handelt[b], und falls es ein Hoare-Tripel ist, ob dieses gültig ist oder nicht.

(i) $\{i < j\}$ `i = i + 7;` $\{i + 7 < j\}$

☐ kein Hoare-Tripel ☒ ungültiges Hoare-Tripel ☐ gültiges Hoare-Tripel

(ii) $\{n = a + b \text{ und } a \leq b\}$ `{b = b + 1; n = n + 1;}` $\{a < b\}$

☐ kein Hoare-Tripel ☐ ungültiges Hoare-Tripel ☒ gültiges Hoare-Tripel

(iii) $\{z = x + y\}$ `if (x > 0) {z = z - x;} else {y = y - x;}` $\{x + y \geq z\}$

☐ kein Hoare-Tripel ☐ ungültiges Hoare-Tripel ☒ gültiges Hoare-Tripel

b) Vervollständigen Sie die Regel des Hoare-Kalküls für `while`-Schleifen:

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

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

$$\{z = 42 \text{ und } r = (0 \cdot y) + 0\}\ \texttt{c}\ \{z = 42 \text{ und } r = x \cdot y\}$$

wobei all Variablen i, k, r, x, y, z Typ `int` haben und c das folgende Programmstück ist:

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

Nachdem Sie Ihren Beweis vollendet haben, schreiben Sie hier die benutzten Invarianten hin:

Invariante äußere Schleife I_1: $z = 42 \text{ und } r = i \cdot y$

Invariante innere Schleife I_2: $z = 42 \text{ und } r = (i \cdot y) + k$

Option zu Vereinfachung: $z = 42$ überall weglassen, dafür 1 Punkt Abzug.

[b] Falls es kein Hoare-Tripel ist, dann liegt es *nicht* an einer Spitzfindigkeit wie z.B. ein Semikolon zu viel/zu wenig.

LÖSUNG:

Zur Abkürzung schreiben wir $\wedge$ für "und" und $\vee$ für "oder". Die Gültigkeit des Hoare-Triples

$$\{z = 42 \wedge r = 0 \cdot y\}\ \texttt{int i = 0;}\ \{z = 42 \wedge r = i \cdot y\}$$

folgt nach der Hoare-Regel für Zuweisungen. Die Nachbedingung dieses Triples ist bereits identisch zu I_1, somit haben wir bereits die erste Prämisse $Q \to I_1$ der While-Regel. Die Negation der Schleifenbedingung `i!=x` ergibt $i = x$; es gilt also $I_1 \wedge i = x \to r = x \cdot y \wedge z = 42$ wie benötigt für die dritte Prämisse der While-Regel.

Für die zweite Prämisse der While-Regel haben wir noch zu zeigen:
$\{I_1 \wedge i \neq x\}$ *Rumpf äußere Schleife* $\{I_2\}$ Mit der Konsequenz-Regel verändern wir $I_1 \wedge i \neq x$ zuerst zu $z = 42 \wedge r = (i \cdot y) + 0$. Wir haben hier $i \neq x$ gleich wieder vergessen, dass muss man aber nicht tun. Es gilt

$$\{z = 42 \wedge r = (i \cdot y) + 0\}\ \texttt{int k = 0;}\ \{z = 42 \wedge r = (i \cdot y) + k\}$$

womit wir bereits die Invariante der inneren Schleife direkt erhalten haben. Nach inneren Schleife gilt $I_2 \wedge k = y$, was wir umformen zu $(z = 42 \wedge r = (i \cdot y) + y)$ und dann zu $(z = 42 \wedge r = (i+1) \cdot y)$. Die Gültigkeit des Hoare-Triples

$$\{z = 42 \wedge r = (i + 1) \cdot y\}\ \texttt{i++;}\ \{I_1\}$$

beendet den Beweis für die äußere Schleife.

Zuletzt müssen wir noch den Rumpf der inneren Schleife durchrechnen:

$$\begin{array}{l}
\{I_2 \wedge k \neq y\} \\
\{z = 42 \wedge r = (i \cdot y) + k\} \\
\{z = 42 \wedge r + 1 = (i \cdot y) + (k + 1)\} \quad \texttt{r++;} \quad \{z = 42 \wedge r + 1 = (i \cdot y) + (k + 1)\} \\
\{z = 42 \wedge r + 1 = (i \cdot y) + (k + 1)\} \quad \texttt{k++;} \quad \{z = 42 \wedge r + 1 = (i \cdot y) + k \quad \equiv I_2\}
\end{array}$$