Dr. Andreas Abel
Dr. Ulrich Schöpp
Dr. Steffen Jost

Ludwig-Maximilians-Universität München
Institut für Informatik
20. Dezember 2010

10. Musterlösung zur Vorlesung

Einführung in die Informatik: Programmierung und Software-Entwicklung

Hinweis: *Lediglich Aufgabe 10-7 und 10-11 dieses Übungsblattes werden benotet. Alle Aufgaben sollten Ihnen jedoch zur Klausurvorbereitung behilflich sein.*

Aufgabe 10-1 *Lernfragen* (0 Punkte) (Abgabeformat: `.txt` oder `.pdf`)

a) Welche Elemente können in einer Klasse vorkommen?

LÖSUNG
Eine Klasse enthält eine Menge von Attributen und Methoden. Zum Beispiel kann eine konkrete Klasse Instanzvariablen, Instanzmethoden (Mutatormethoden oder "Setter" sowie Accessormethoden oder "Getter") und auch ein oder mehrere Konstruktormethoden enthalten.

b) Welche Beziehungen zwischen Klassen, welche die UML-Notation beinhaltet, kennen Sie?

LÖSUNG
Assoziation, Aggregation (starke und schwache), Vererbung, Abhängigkeit.

c) Welches ist die allgemeinste Klasse in Java? Nennen Sie eine Methode, die in dieser Klasse definiert ist, und geben Sie die Signatur (Sichtbarkeit, Rückgabetyp, Name, Parameterliste) an.

LÖSUNG
Die Basisklasse der Java-Klassenhierarchie ist die Klasse **java.lang.Object**. Die Methode **toString()** dieser Klasse z.B. gibt eine String-Repräsentation des Objekts zurück. Die vollständige Signatur dieser Methode lautet **public String toString()**. Sie hat keine expliziten Parameter.

d) Welche Zeitkomplexität hat der Algorithmus Quicksort im durchschnittlichen bzw. schlechtesten Fall?

LÖSUNG
Im Worst Case hat Quicksort die Zeitkomplexität $O(n^2)$, im Average und im Best Case $O(n \log n)$.

e) Unter welchen Umständen muss eine Methode, in deren Implementierung die Anweisung `throw new MyException()` vorkommt, per `throws MyException` die Möglichkeit eines Auftretens der Ausnahme deklarieren?

LÖSUNG
Wenn die Exception `MyException` nicht in einem `try/catch`-Block innerhalb der Implementierung abgefangen und behandelt wird, sondern an die aufrufende Methode weitergegeben werden soll.

f) Enthält folgendes Java-Programm Syntaxfehler? Wenn ja, welche?

```
int i = 0;
Object o1 = "123" + i;
if (true == false) i /= 1;
```

LÖSUNG
Das angegebene Programmfragment enthält keine Syntaxfehler.

Aufgabe 10-2 *EBNF* (0 Punkte) (Abgabeformat: `.txt` oder `.pdf`)
Ein Text ist eine Sequenz von Sätzen, die jeweils durch einen Punkt (`.`) abgeschlossen werden. Ein Satz besteht aus einem Subjekt, gefolgt von einem Prädikat und einem Objekt. Das Objekt kann ein Wort oder ein in Anführungszeichen (» bzw. «) eingeschlossener Text sein.

a) Geben Sie eine EBNF-Grammatik für die oben beschriebenen Texte an. Dabei können Sie folgende Regeln als gegeben betrachten:

$$
\begin{aligned}
\mathit{Subjekt} &= \texttt{"Maus "} \mid \texttt{"Katze "} \\
\mathit{Prädikat} &= \texttt{"sagt "} \mid \texttt{"meint "} \\
\mathit{Wort} &= \texttt{"bla"} \mid \texttt{"foo "} \\
\mathit{Punkt} &= \texttt{"."} \\
\mathit{AnführungLinks} &= \texttt{"»"} \\
\mathit{AnführungRechts} &= \texttt{"«"}
\end{aligned}
$$

LÖSUNG

$$
\begin{aligned}
\mathit{Subjekt} &= \texttt{"Maus "} \mid \texttt{"Katze "} \\
\mathit{Prädikat} &= \texttt{"sagt "} \mid \texttt{"meint "} \\
\mathit{Wort} &= \texttt{"bla"} \mid \texttt{"foo "} \\
\mathit{Punkt} &= \texttt{". "} \\
\mathit{AnführungLinks} &= \texttt{"»"} \\
\mathit{AnführungRechts} &= \texttt{"«"} \\
\mathit{Objekt} &= \mathit{Wort} \mid \mathit{AnführungLinks}\ \mathit{Text}\ \mathit{AnführungRechts} \\
\mathit{Satz} &= \mathit{Subjekt}\ \mathit{Prädikat}\ \mathit{Objekt}\ \mathit{Punkt} \\
\mathit{Text} &= \mathit{Satz}\ \{\mathit{Satz}\}
\end{aligned}
$$

b) Leiten Sie den Text

`Maus sagt »Katze sagt bla.«.`

aus Ihrer Grammatik ab.

LÖSUNG:

Text
→ *Satz*
→ *Subjekt Prädikat Objekt Punkt*
→ "Maus" "sagt" *AnführungLinks Text AnführungRechts* "."
→ "Maus" "sagt" *AnführungLinks Satz AnführungRechts* "."

→ “Maus” “sagt” "»" *Satz* "«" “.”
→ “Maus” “sagt” "»" *Subjekt Prädikat Objekt Punkt* "«" “.”
→ “Maus” “sagt” "»" “Katze” “sagt” “bla” “.” "«" “.”

Aufgabe 10-3 *Basistypen* (0 Punkte) (Abgabeformat: `.txt` oder `.pdf`)
Gegeben seien folgende Zuweisungen:

```
int i1 = 12;
int i2 = 112;
byte b1 = 2;
byte b2 = 100;
float f1 = 10.1e30f;
float f2 = 0.3f;
double d1 = 0.456;
double d2 = 1e-60;
char c1 = 'a';
String str1 = "12";
```

Geben Sie den Typ der folgenden Ausdrücke an, d.h. vervollständigen Sie:

LÖSUNG:

```
double x1 = b1 * i1 / f1 * 3.0;
String x2 = f1 / 2 + b1 + str1 + d2;
String x3 = c1 + i1 + str1;
String x4 = c1 + str1 + i1;
double x5 = d2 / f1 * i2;
int    x6 = c1 * c1;
```

Fügen Sie in folgende Ausdrücke Typcasts ein, so dass das im Kommentar angegebene Ergebnis herauskommt:

```
int y1 = i1 * f2 + i1; // 12
float y2 = d2 * f1;   // 1.00999994E-29
String y3 = str1 + b2 + c1 + b1; // "12c972"
byte y4 = (((d1 * i2) + c1) * d1 / (11.0f + b1 + f2 * 10)); // 4
```

LÖSUNG:

```
int y1 = i1 * (int)f2 + i1; // 12
float y2 = (float)(d2 * (double)f1);   // 1.00999994E-29
String y3 = str1 + (char)b2 + (byte)c1 + b1; // "12d972"
byte y4 = (byte)((int)(((d1 * i2) + c1) * d1) / (int)(11.0f + b1 + f2 * 10)); // 4
```

Aufgabe 10-4 ***Rekursion und Iteration*** **(0 Punkte)** (Geben Sie alle Dateien Ihrer Lösung ab)
Sei die Prozedur static int f(int x, int y) wie folgt gegeben.

```
static int f(int x, int y) {
  int rest;
  do {
    rest = x%y; //Modulo-Operator
    x = y;
    y = rest;
  } while (y!=0);
        return x;
}
```

a) Seien x,y ganze Zahlen und es gelte $x \neq 0$ und $y \neq 0$. Was ist das Ergebnis des Prozeduraufrufs f(x,y), d. h. welche mathematische Funktion realisiert f?

LÖSUNG
Die Prozedur f realisiert den **größten gemeinsamen Teiler** (ggT).

b) Die Prozedur f stellt die iterative Lösung der gesuchten mathematischen Funktion dar. Wie können Sie die Prozedur kürzer als rekursive Variante schreiben?

LÖSUNG:

```
// Kuerzer
static int ggT(int x, int y) {
  int rest = x%y;
  if(rest==0) return y;
  else return ggT(y,rest);
}

// Noch kuerzer
static int ggT(int x, int y) {
  if(x%y==0) return y;
  else return ggT(y,x%y);
}
```

Aufgabe 10-5 ***Komplexität*** **(0 Punkte)** (Abgabeformat: `.txt` oder `.pdf`)
Die folgende Prozedur prüft für ein Array, ob sie ein Palindrom ist:

```
public static boolean isPalindrom(int[] a) {
  for (int i=0; i < a.length; i++) {
    if (a[i] != a[a.length - 1 - i]) {
      return false;
    }
  }
  return true;
}
```

a) Welche Komplexität hat diese Prozedur im schlechtesten Fall (in O-Notation)?

LÖSUNG
Im schlechtesten Fall wird die `for`-Schleife `a.length` mal durchlaufen, die Zeitkomplexität ist also $O(n)$.

b) Offensichtlich ist die Prozedur ineffizient: es genügt, die Schleife von `0` bis `a.length/2` laufen zu lassen. Wie ändert sich durch diese Optimierung die Komplexität?

LÖSUNG
Die Zeitkomplexität des Algorithmus ist immer noch linear.

Aufgabe 10-6 *Listen* (0 Punkte) (Geben Sie alle `.java`-Dateien Ihrer Lösung ab)
Gegeben seien folgende Klassen, die eine (minimale) verkettete Liste implementieren:

```
public class List {
  private ListElem anchor;

  public void addFirst(Object o) {
    anchor = new ListElem(o, anchor);
  }
}

public class ListElem {
  private ListElem next;
  private Object data;

  public ListElem(Object data, ListElem next) {
    this.data = data;
    this.next = next;

  }
}
```

Ergänzen Sie die Klasse `List` um eine Methode `public int size()`, die die Länge der Liste zurückgibt. Sie dürfen dazu auch die Klasse `ListElem` anpassen. Machen Sie deutlich, welche Änderung an welcher Klasse vorgenommen werden soll. Nennen und erklären Sie einen Begriff aus dem Bereich der objektorientierten Programmierung, der Ihre Anpassungen bezüglich der Klassen `List` und `ListElem` bezeichnet.

LÖSUNG:

```
public class List {
  private ListElem anchor;

  public void addFirst(Object o) {
    anchor = new ListElem(o, anchor);
  }
```

```
  public int size() {
    if (anchor == null)
      return 0;
    else
      return anchor.size();
  }
}


public class ListElem {
  private ListElem next;
  private Object data;

  public ListElem(Object data, ListElem next) {
    this.data = data;
    this.next = next;
  }

  public int size() {
    int sum = 0;
    if (next != null)
      sum = next.size();
    return 1 + sum;
  }
}
```

In der objektorientierten Programmierung bezeichnet der Begriff der **Delegation** eine Interaktion von Klassen. Im Gegensatz zum Konzept der Vererbung gibt dabei eine Klasse die Verantwortlichkeit für einen Teil ihrer Funktionalität an eine andere Klasse ab. Im Beispiel "delegiert" die Klasse List im Fall der nicht-leeren Liste die Berechnung der Anzahl der Listenelemente an die Hilfsklasse `ListElem` durch den Aufruf von `ListElem.size()`. Die Verkettung der Listenelemente ist Teil der Funktionalität dieser Klasse und nicht originär der Klasse List.

Aufgabe 10-7 *Klassen* (4 Punkte) (Geben Sie alle `.java`-Dateien Ihrer Lösung ab)
In der Praxis muss man oft mit Klassenbibliotheken arbeiten, die man nicht verändern kann. Gegeben seien die folgenden Klassen `Text`, `Wort`, `Nicht` und `Gut`, die Sie nicht verändern können.

Klasse `Wort`:

```
public class Wort {
    @Override
    public String toString() {
        return "";
    }
}
```

Klasse `Gut`:

```
public class Gut extends Wort {
    @Override
    public String toString() {
        return "gut";
    }
}
```

Klasse `Nicht`**:**

```
public class Nicht extends Wort {
    @Override
    public String toString() {
       return "nicht";
    }
}
```

Klasse `Text`**:**

```
import java.util.LinkedList;

public class Text {
    private LinkedList<Wort> characters;

    public Text() {
        characters = new LinkedList<Wort>();
        initWoerter();
    }

    public void initWoerter() {
        addWort(new Nicht());
        addWort(new Gut());
    }

    public void addWort(Wort w) {
        characters.add(w);
    }

    public void print() {
        for (Wort c : characters) {
            System.out.print(c.toString() + " ");
        }
        System.out.println();
    }
}
```

Mit diesen Klassen gibt folgendes Beispielprogramm den Text “nicht gut” aus:

```
public class Main {
    public static void main(String[] args) {
        Text t = new Text();
        t.print();
    }
}
```

Verändern Sie die Klasse `Main` so, dass sie den Text "sehr gut" ausgibt. Sie dürfen dabei **keine Methoden zur Ausgabe von Text verwenden**, d.h. Methoden wie `System.out.print` oder `System.out.println` dürfen **nicht** verwendet werden. Im gesamten Programm dürfen nur in einer Methode Anweisungen zum Ausgeben von Text vorkommen, nämlich in der Methode `print` der Klasse `Text`. Weiterhin dürfen Sie die Klassen `Text`, `Wort`, `Nicht` und `Gut` **nicht** verändern. Sie dürfen jedoch neue Klassen definieren und die `main`-Methode in der Klasse `Main` verändern.

LÖSUNG: Wir definieren folgende neue Klasse, welche fast alles von der Klasse Text erbt. Lediglich die Methode `initWoerter` wird in unserem Sinn überschrieben.

```
public class TextGut extends Text {

    public void initWoerter() {
        addWort(new Sehr());
        addWort(new Gut());
    }

}
```

Nun definieren wir noch eine neue Klasse Sehr für das Wort "sehr", analog zu der gegebenen Klasse Gut.

```
public class Sehr extends Wort {
    @Override
    public String toString() {
        return "sehr";
    }
}
```

In der dritte Zeile der Klasse Main erstellen wir dann einfach ein Objekt der Klasse `TextGut`, um die gewünschte Ausgabe zu erzielen.

```
public class Main {
    public static void main(String[] args) {
        Text t = new TextGut();
        t.print();
    }
}
```

Aufgabe 10-8 ***Vererbung*** **(0 Punkte)** (Abgabeformat: `.txt` oder `.pdf`)
Welche Ausgabe erzeugt folgendes Programm?

```
public class A {
  public int f() {
    return 1;
  }
```

```
  public int g() {
    return h();
  }

  public int h() {
    return 20;
  }

  public int h(Object o) {
    return 5;
  }

  public int h(String s) {
    return 15;
  }
}

public class B extends A {
  public int f() {
    return 2;
  }

  public int g() {
    return super.g();
  }

  public int h() {
    return 30;
  }

  public int h(Object o) {
    return 25;
  }
}

public class Main1 {
  public static void main(String[] args) {
    A a = new A();
    System.out.println("a.f() = " + a.f());
    System.out.println("a.g() = " + a.g());
    a = new B();
    System.out.println("a.f() = " + a.f());
    System.out.println("a.g() = " + a.g());
    System.out.println("a.h(x) = " + a.h("x"));
  }
}
```

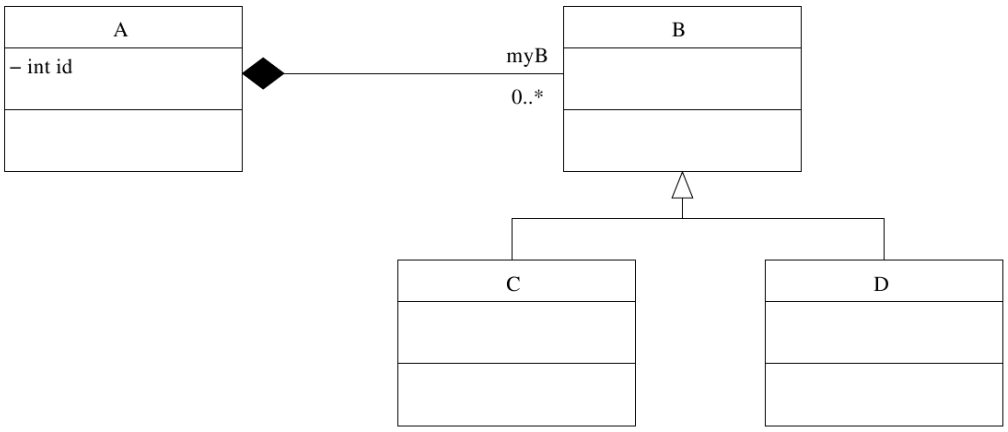
LÖSUNG
Das Programm erzeugt folgende Ausgabe:

```
a.f() = 1
a.g() = 20
a.f() = 2
a.g() = 30
a.h(x) = 15
```

Für die letzten drei Ausgaben ist anzumerken, dass die Variable der Klasse A nun auf ein Objekt der Klasse B zeigt. Es werden also alle von B neu implementierten Methoden bevorzugt, sofern in A keine Methode gleichen Namens mit einer besser passenden Signatur existiert.

In diesem Beispiel gibt es die Methode `h` in der Klasse B mit dem Argumenttyp Object und auch noch die geerbter Methode `h` mit Argumenttyp String. Da der letzte Aufruf mit einem String Argument ausgeführt wird, wird die geerbte Methode aus Klasse A gewählt, den String ist spezieller als Objekt.

Aufgabe 10-9 *UML I* (0 Punkte) (Geben Sie alle Dateien Ihrer Lösung ab)
Gegeben sei das folgende UML-Diagramm:



a) Implementieren Sie die Klasse `A` wie im Diagramm angegeben.

LÖSUNG:

```
public class A {

  public A() {
  }

  private int id;

  private ArrayList myB = new ArrayList();
```

```
}
```

b) Ergänzen Sie Ihre Implementierung um Getter- und Setter-Methoden für die Attribute von `A`.

LÖSUNG:

```
public class A {

  public A() {
  }

  public int getID() {
    return id;
  }

  public void setID(int i) {
    id = i;
  }

  public ArrayList getMyB() {
        return myB;
  }

  public void setMyB(ArrayList b) {
    myB = b;
  }


  private int id;
  private ArrayList myB = new ArrayList();
}
```

Hinweis: *Durch das Hinzufügen der Setter-Methoden wird das UML Diagramm aus Teil a. verletzt. Durch die Existenz einer Setter-Methode kann die Abhängigkeit der Lebensdauer nicht mehr gewährleistend werden, weshalb dann nur noch eine schwache Aggregation vorliegt.*

c) Wie nennt man die Beziehung zwischen `A` und `B`? Beschreiben Sie kurz, welche Eigenschaften eine solche Beziehung hat.

LÖSUNG
Die Beziehung zwischen `A` und `B` nennt man **Starke Aggregation**. Instanzen der Klasse B existieren nur innerhalb des Aggregats "A"; d.h. die Lebensdauer eines "B"-Objekts wird durch die Lebensdauer des zugehörigen Aggregats beschränkt. Weitere Eigenschaften der Starken Aggregation siehe Vorlesung.

Aufgabe 10-10 *UML II* (0 Punkte) (Geben Sie alle Dateien Ihrer Lösung ab)
Hinweis: Diese Aufgabe stellt eine für die Wiederholung des Stoffes hilfreiche, aber recht umfangreiche Aufgabe dar. Wir bitten um Verständnis, wenn insbesondere die Lösung für die Teilaufgabe c, nicht ausführlich in den Übungen besprochen werden kann. Wir werden jedoch die Lösung zusammen mit den anderen Aufgaben am Ende der vorlesungsfreien Zeit ins Netz stellen.

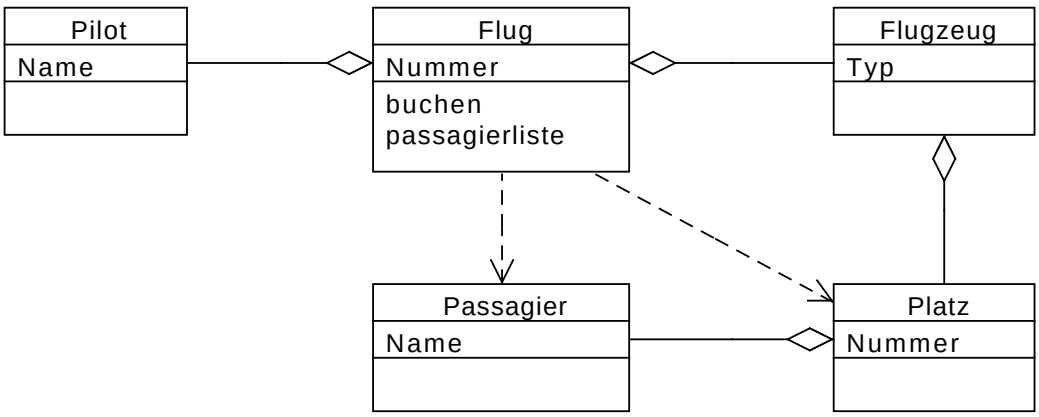
Gegeben sei folgendes Szenario: Es gibt eine Menge von Flügen. Jeder Flug wird von einem Piloten durchgeführt. Außerdem kommt dafür ein Flugzeug zum Einsatz, in dem Passagieren bestimmte Plätze zugewiesen sind. Es ist möglich, einen Flug zu buchen, sowie eine Passagierliste für den Flug auszugeben. Die Passagierliste zeigt die Flugnummer, den Namen des Piloten, die Anzahl der Passagiere sowie eine Liste aller Passagiere mit Namen und zugewiesenen Plätzen.

a) Identifizieren Sie die beteiligten Klassen.

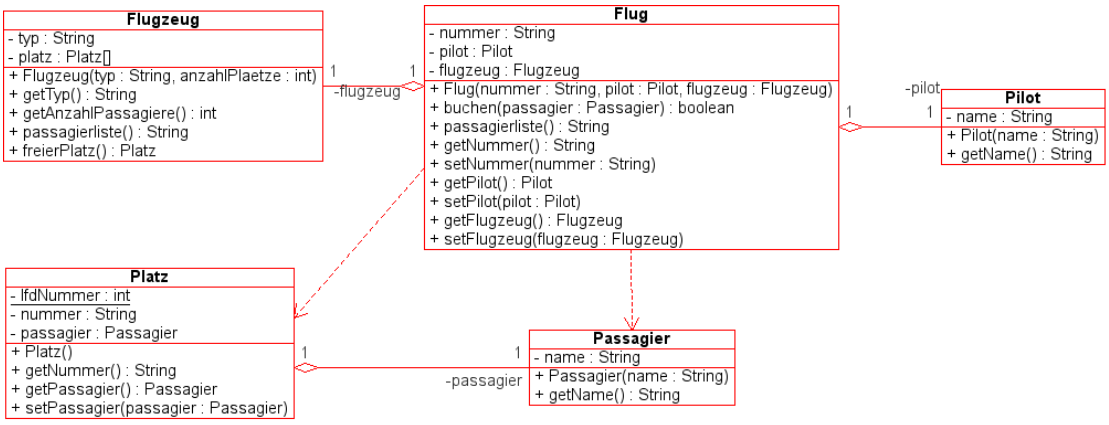
LÖSUNG
Flug, Pilot, Flugzeug, Passagier und Platz

b) Zeichnen Sie ein UML-Diagramm, das die Beziehungen zwischen den einzelnen Klassen darstellt.

LÖSUNG:



Beachten Sie, dass dieses Diagramm nur die grob die Beziehungen der Klassen modelliert. Es enthält nicht alle Methoden und Attribute der späteren Implementierung. Ein vollständiges Diagramm wäre wie folgt.



c) Implementieren Sie das Programm.

LÖSUNG:

```
public class Main {
  public static void main(String[] args) {
    Pilot kirk = new Pilot("James T. Kirk");
    Flugzeug enterprise = new Flugzeug("U.S.S. Enterprise", 400);
    Flug flug = new Flug("1701", kirk, enterprise);
    Passagier zaphod = new Passagier("Zaphod Beeblebrox");
    Passagier bogart = new Passagier("Humphrey Bogart");
    flug.buchen(zaphod);
    flug.buchen(bogart);
    System.out.println(flug.passagierliste());
  }
}
```

```
public class Flug {
  private String nummer;
  private Pilot pilot;
  private Flugzeug flugzeug;

  public Flug(String nummer, Pilot pilot, Flugzeug flugzeug) {
    this.nummer = nummer;
    this.pilot = pilot;
    this.flugzeug = flugzeug;
  }

  /**
   * bucht einen Platz fuer diesen Flug
   *
   * @param passagier
   *            der Passagier, fuer den ein Platz gebucht wird
```

```
   * @return ob ein Platz gebucht werden konnte
   */
  public boolean buchen(Passagier passagier) {
    Platz platz;
    if ((platz = flugzeug.freierPlatz()) != null) {
      platz.setPassagier(passagier);
      return true;
    } else {
      return false;
    }
  }

  /**
   * gibt die Passagierliste dieses Fluges aus. Darin enthalten sind auch die
   * Flugnummer, der Name des Piloten sowie die Anzahl der Passagiere.
   */
  public String passagierliste() {
    return "Passagierliste fuer Flug Nr. " + nummer + ":\n" + "Pilot: "
        + pilot.getName() + "\n" + "Anzahl Passagiere: "
        + flugzeug.getAnzahlPassagiere() + "\n"
        + flugzeug.passagierliste();
  }
}


public class Flugzeug {
  private String typ;
  private Platz[] platz;

  public Flugzeug(String typ, int anzahlPlaetze) {
    this.typ = typ;
    platz = new Platz[anzahlPlaetze];
    for (int n = 0; n < anzahlPlaetze; n++) {
      platz[n] = new Platz();
    }
  }

  public String getTyp() {
    return typ;
  }

  public int getAnzahlPassagiere() {
    int anzahlPassagiere = 0;
    for (int n = 0; n < platz.length; n++) {
      if (platz[n].getPassagier() != null) {
        anzahlPassagiere++;
      }
```

```
    }
    return anzahlPassagiere;
  }

  public String passagierliste() {
    String passagierliste = "";
    for (int n = 0; n < platz.length; n++) {
      if (platz[n].getPassagier() != null) {
        passagierliste += platz[n].getNummer() + ": "
            + platz[n].getPassagier().getName() + "\n";
      }
    }
    return passagierliste;
  }

  public Platz freierPlatz() {
    for (int n = 0; n < platz.length; n++) {
      if (platz[n].getPassagier() == null) {
        return platz[n];
      }
    }
    return null;
  }
}


public class Platz {
  private static int lfdNummer = 666;
  private String nummer;
  private Passagier passagier;

  public Platz() {
    nummer = String.valueOf(lfdNummer++);
  }

  public String getNummer() {
    return nummer;
  }

  public Passagier getPassagier() {
    return passagier;
  }

  public void setPassagier(Passagier passagier) {
    this.passagier = passagier;
  }
}
```

```
public class Pilot {
  private String name;

  public Pilot(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
}


public class Passagier {
  private String name;

  public Passagier(String name) {
    this.name = name;
  }

  public String getName() {
    return name;
  }
}
```

Aufgabe 10-11 *Ausnahmen* (4 Punkte) (Geben Sie alle `.java`-Dateien Ihrer Lösung ab)

a) Schreiben Sie eine Prozedur `public static boolean neighbour(int[] arr)`, die prüft, ob in einem `int[]`-Array zwei benachbarte Elemente gleich sind. Die Prozedur soll also genau dann `true` zurückgeben, wenn `arr[i] == arr[i+1]` für wenigstens ein `i` gilt.

LÖSUNG:

```
public static boolean neighbour(int[] arr){
  for (int i = 0; i < arr.length - 1; i++) {
    if (arr[i] == arr[i + 1]) {
        return true;
    }
  }
  return false;
}
```

b) Schreiben Sie eine Ausnahme `ArrayTooShortException`, die ausgelöst werden soll, wenn das Array weniger als zwei Elemente enthält und insofern eine sinnvolle Prüfung auf

benachbarte Elemente nicht möglich ist. Geben Sie an, wie Ihre im vorigen Aufgabenteil formulierte Prozedur modifiziert werden muss, damit die Ausnahme ausgelöst wird, wenn das Eingabearray nicht lang genug ist.

LÖSUNG: Definieren einer neuen Unterklasse von `Exception`:

```
public class ArrayTooShortException extends Exception {
}
```

Änderungen an der Prozedur `neighbour`:

```
public static boolean neighbour(int[] arr) throws ArrayTooShortException {
  if (arr.length < 2)
    throw new ArrayTooShortException();
  for (int i = 0; i < arr.length - 1; i++) {
    if (arr[i] == arr[i + 1]) {
        return true;
    }
  }
  return false;
}
```

Abgabe: Sie können ihre Lösungen bis Montag, den 10.01.11, 14 Uhr über UniWorX abgeben.

Wie immer gilt das Dateien, welche die Tutoren nicht ohne Probleme öffnen können, nicht beachtet werden! Java Dateien, welche nicht kompilieren, werden ebenfalls ignoriert.