Name:_____ Matrikelnummer:_____

2. August 2005

Klausur zur Vorlesung
Grundlagen der Programmierung I und II (WS 2004/2005 und SS 2005)

Wichtig! Wichtig! Wichtig!

- Schreiben Sie zumindest Ihre Matrikelnummer auf jedes Blatt der Klausur!
- Blätter ohne Matrikelnummer werden nicht gewertet!
- Lassen Sie die Klausur zusammengeheftet!
- Die Klausur dauert 3 Stunden und umfasst 8 Aufgaben mit insgesamt 72 Punkten.
- Als Hilfe: Rechnen Sie pro Punkt mit etwa 2,5 Minuten Bearbeitungszeit.
- Die Klausur ist bestanden, wenn mindestens 50% der Punkte erreicht wurden.
- Abschreiben oder Abschreiben lassen führen zum Nichtbestehen der Klausur.
- Kennzeichnen Sie Ihre Lösung eindeutig! Es wird keine Lösung gewertet, wenn Sie zu einer Aufgaben mehr als eine Lösung abgeben.

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	∑
maximal erreichbare Punkte	5	7	10	5	9	12	12	12	72
erreichte Punkte									

Note	

Bitte kreuzen Sie Ihren Studiengang an:

	Bachelor-Studiengang Informatik (ab WS 04/05)
	Bachelor-Diplomstudiengang Informatik nach DPO4
	Wirtschaftsinformatik
	Ingenieurinformatik Schwerpunkt Informatik
	Ingenieurinformatik Schwerpunkt Maschinenbau
	Ingenieurinformatik Schwerpunkt Elektrotechnik
	Lehramt für Gymnasien und Gesamtschulen (Studienordnung 2004)
	Lehramt für Sekundarstufe 2
	Medienwissenschaften
	Sonstige: _____

1. Verständnisfragen I (1+2+1+1=5 Punkte)

a) Welche Typen haben folgende Literale (es können auch ungültige darunter sein)

1P

Literal	Typ
“GP1“	String
’2’	char
3	int/long
3.1	double
20i	ungültig
2e3	double
“2“	String
3.1D	double
true	boolean
“true“	String
030	int/long,**Oktalzahl**

b) **Die Zuweisung in der linken Spalte führt zu welchem Ergebnis rechts?** 2P

Zuweisungen	Ergebnis des System.out.println Befehls
`byte b1=-128;` `System.out.println(~b1);`	127
`int b3=4;` `System.out.println("b3 << 2 ="+(b3<<2));`	16
`byte secondsPerHour=(byte)1040;` `System.out.println(secondsPerHour);`	=10000010000 …. Abgeschnitten auf byte: 00010000 = 16
`int l=030;` `System.out.println(l);`	24

`int x=0;`
`boolean xx;`
`xx=x!=0&&1/x>0;`
	false

c) **`While` Schleifen:** 1 Punkt

Kann es passieren, dass die Anweisungen einer `while` Schleife nicht ausgeführt werden? Ja.

Kann es passieren, dass die Anweisungen einer `do-while` Schleife nicht ausgeführt werden? Nein.

d) **Programmiertechnik:** 1 Punkt

Das folgende Programmfragment:

```
sign=-17;
if (x<=0) if (x==0)sign=0; else sign=-1;
```

ergibt für `x` welches `sign`:

x	**sign**
-1	-1
0	0
1	-17

2. Verständnisfragen II (3+2+2 = 7 Punkte)

a) Schreiben Sie ein **komplettes Java Programm**, welches als Input von der **Tastatur** eine Zeile mit zwei int Werten erwartet (durch ein oder mehrere Leerzeichen getrennt), und danach die zwei Zahlen wieder **ausgibt**. 3 P.

Hinweise: Sie können **Stream Klassen** aus javagently verwenden. Sie müssen auch die wichtigsten **Ausnahmen** abfragen.

Anmerkung: Die Stream-Klasse (J. Bishop) aus javagently.*;

- Eingabefunktionen aus der Klasse Stream:

  public int **readInt** (); liest so lange, bis ein int gelesen wurde

  public double **readDouble** (); liest so lange, bis ein double gelesen wurde

  public String **readString** (); liest nur bis zum ersten “white space”!

  public char **readChar** (); liest genau ein Zeichen

- Konstuktoren:

  Stream (InputStream Dateiname) //z.B. für (System.in)

  Stream (String Dateiname, int how) //z.B. für Dateiname

```
import java.io.*;
import javagently.*;
class EinAusgabe
{
   public static void main (String [] args) throws IOException
   {
      int n1, n2;
      Stream tastatur = new Stream (System.in);
      System.out.print("Gib 2 Zahlen ein ");
      n1 = tastatur.readInt();
      n2 = tastatur.readInt();
      System.out.println ("Deine Zahlen waren "+n1+" "+n2);
   }
}
```

b) Vervollständigen Sie das untenstehende Programm, um zwei **boolsche** Arrays a und b mit der Länge 10 mit Zufallswerten aufzufüllen. Schreiben Sie auch die **Methode „sindGleich“**: diese gibt die Anzahl der gleichen boolschen Werte wieder, die im gleichen Index von a und b stehen, z.B. wäre „sindGleich“ bei

a: TFFFTTFTTF
b: FFFTTFFTTT

gleich 6. 2 Punkte

```
public static void main (String [] args){
  //erzeugen Sie die boolschen arrays a und b und füllen
  //Sie diese mit Zufallswerten.
    boolean [] a=new boolean[10];
    boolean [] b=new boolean[10];

      for (int i=0;i<a.length;i++)
        if (Math.random()<0.5)a[i]=true;
      for (int i=0;jlauf<a.length;i++)
        if (Math.random()<0.5)b[i]=true;
      sindGleich(a,b);
}
// fügen Sie hier die Methode sindGleich ein
    static int sindGleich(boolean a[], boolean b[]){
      int zaehler=0;
      for (int i=0;i<a.length;i++ )
        if (a[i]==b[i])zaehler++;
      return zaehler;
    }
```

Anmerkung: Klasse Math aus Package java.lang

Method Summary

static double	**random**() Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

c) Zeigen Sie **alle Fehler** in der/den folgenden Klasse/Methoden auf, **die wegen Doppeldeklarationen auftreten**. 2 Punkte (minus 1 Punkt bei falscher/fehlender Meldung einer Doppeldeklaration)

```
public class …
{
 int i;

 void meth1()
 {   int i=7;   }

 public static void main (String args[])
 {
   for (int i=0;i<5 ;i++ )
   { System.out.println("Drucke diese Zeile 5-mal");  }
   printar();
 }

 public static void printar()
 {
   int iprint=0;

   for (int iprint=0; iprint<20; iprint++) //Doppeldeklaration
   { System.out.println("Drucke diese Zeile 20mal");}

   for (int j=5; j<15; j++)
   {
     int iprint=10;    //Doppeldeklaration
     System.out.println("Drucke diese Zeile "+iprint+" mal");
   }
  }
}
```

3. (OO) Programmiertechniken 10 Punkte

Beantworten Sie die folgenden Fragen zum vollständigen Java Programm unten (die dabei verwendete Klasse „Display“ und ihre Methoden sind im Anhang unten erklärt):

a) Klassifizieren Sie jede Marke „_“ durch die passende Bezeichnung: **Konstruktor (K) / Methode mit Ergebnis (ME) / Methode ohne Ergebnis (MO) / Klassenmethode (KM)/Klassenvariable (KV) / keine dieser Begrifflichkeiten trifft zu (-)** 2 Punkte

b) Definieren Sie alle Objektvariablen der Klasse Curio, die mit **?? markiert sind, als „private“** und erweitern Sie das Programm so, dass es noch immer in der vorliegenden Form funktioniert. 4 Punkte

c) In der derzeitigen Form erhält man bei Eingabe von „ Masks “ (beachten Sie die Leerzeichen!) auf die Aufforderung „Kind of Curio sold“ die Ausgabe:

Masks is an invalid curio name

Korrigieren Sie bitte dieses Problem. (Hinweis: Die Methode trim() der Klasse String ist in Anmerkung2 erläutert.) 4 Punkte

Korrekturen:

1. siehe Programm
2. die beiden getter Methoden getName() und getPrice() werden eingefügt und insgesamt achtmal im Programm verwendet – es müssen nicht alle gefunden werden!
3. einfügen an der richtigen Stelle: curioName=curioName.trim();

```
import javagently.*;
class CurioStore3 {
  /* Curio Store Version 3        by J M Bishop April 2000
   * ---------------------
   * This shop has stock levels and a sell option
   * Illustrates working within an object-oriented program
   * Makes use of the Display class for input-output
   */
  public static void main (String [ ] args) _ KM{
    new CurioStore3 ();
  }
  // Declare objects
  Display display = new Display ("Polelo Curio Store");
  Curio   mugs, tshirts, carvings;
  boolean open_  -;
  //
  CurioStore3 ()_ K {
    mugs = new Curio("Traditional mugs", 6, "beaded in Ndebele style",20,
     display);
    tshirts = new Curio("T-shirts", 30, "sizes M to XL", 50, display);
    carvings = new Curio("Masks", 80, "carved in wood", 8, display);
    // print out the initial shop details
    report();

    // perform a sequence of actions
    stockTheStore();
    openTheStore();

    while (open) {
       sellCurios();
       open = display.getString("The store is").equals("Open");
       available();
    }
    display.println("Store closed.\nHave a good day");
  }


   void report ()_ MO {
    display.println("The Polelo Curio Store sells\n");
    // use the objects' access to toString to print their contents
    display.println(""+mugs);
    display.println(""+tshirts);
    display.println(""+carvings);
  }
  void stockTheStore () {
    display.ready("Press ready when new stock levels have been entered");
    mugs.addStock(display.getInt(mugs.getName()));
    tshirts.addStock(display.getInt(tshirts.getName()));
    carvings.addStock(display.getInt(carvings.getName()));
   }
   void openTheStore () {
     display.prompt("Kind of curio sold","                ");
     display.prompt("Number sold",0);
     display.prompt("The store is","Open");
     open = true;
   }
   void sellCurios () {
     Curio curio;

     display.ready("Press ready when sale completed");
     String curioName = display.getString("Kind of curio sold");
     curioName=curioName.trim();
```

```
      int curioSold = display.getInt("Number sold");

      if (curioName.equals(mugs.getName())) {
         curio = mugs;
      } else
      if (curioName.equals(tshirts.getName())) {
        curio = tshirts;
      } else
      if (curioName.equals(carvings.getName())) {
        curio = carvings;
      } else {
        display.println(curioName + " is an invalid curio name");
        return;
      }
      display.println("\nOrder for "+curioSold+" "+curioName);
      curio.sell(curioSold);
      display.println("RECEIPT: for "+ curioSold + " "
          + curioName +" at G"+ curio.getPrice()+" each = G" +
          curio.getPrice()*curioSold);
   }
   void available () {
     display.println("\nAvailable are "+mugs.stockLevel()+" "+
       tshirts.stockLevel()+" "+carvings.stockLevel()+
       " curios respectively\n");
   }
}
class Curio {

    String name_; - //??
    int price; //??
    String description; //??
    int stock_; - //??

    Curio (String n, int p, String d, int t, Display display) _ K {
      name = n;
      price = p;
      description = d;
      display.prompt(name,t);
    }
    void addStock (int n) _ MO {
      stock += n;
    }
    void sell (int n) {
        stock -= n;
    }
    int stockLevel ()_ ME {
      return stock;
    }
    // This method is accessed by println to turn
    // a Curio object into a String.
    public String toString ()_ ME {
      return name + " "+description+" for G" + price;
    }
    public String getName() {return name; }
    public String getPrice() {return price; }
}
```

4. Arrays sortieren - 5 Punkte

Schreiben sie eine Methode sort(), die ein Integer Array erhält und dieses mit dem Bubble-Sort Algorithmus aufsteigend sortiert und wieder zurückgibt.
Bubble-Sort:
Läuft über das gesamte Array der Länge n und betrachtet immer zwei direkt benachbarte Elemente x_i und x_{i+1}. Ist der Wert von x_i größer als der Wert von x_{i+1} werden die beiden Werte getauscht. Im nächsten Schritt betrachtet man die Werte von x_{i+1} und x_{i+2}. Nach dem ersten Durchlauf des Bubble-Sort Algorithmus befindet sich demnach das größte Element des Arrays an letzter Stelle. Nach dem zweiten Durchlauf sind die zwei größten Elemente am Ende des Arrays, und nach allen Durchläufen ist das Array sortiert.

Vorgabe:

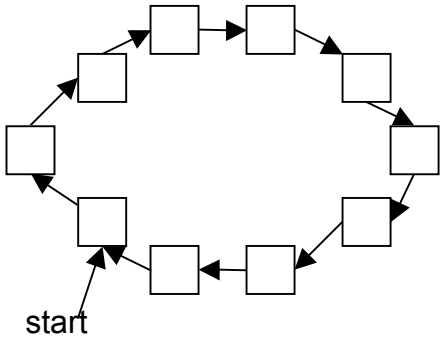
```
public class S1 {
  // Die Methode hier einfuegen:
  public static void main(String[] args) {
  int [] arr = {6,-5,2,-4,9,-3,-2,-1,0,1,2,3,4,15,5,6,7,8,-9};
  arr = sort(arr);
     for(int i = 0;i<arr.length;i++){
       System.out.println(arr[i]);
     }
  }
}
```

Musterlösung:

```
public class S1 {
public static int[] sort(int [] arr){
  int tausche;
    for (int i =0; i<arr.length-1; i++){
      for(int c= 0;c<arr.length-i-1;c++){
        if(arr[c]>arr[c+1]){
                tausche = arr[c];
                arr[c]=arr[c+1];
                arr[c+1]=tausche;
        }
      }
    }
  return arr;
}
  public static void main(String[] args) {
  int [] arr = {6,-5,2,-4,9,-3,-2,-1,0,1,2,3,4,15,5,6,7,8,-9};
  arr = sort(arr);
    for(int i = 0;i<arr.length;i++){
      System.out.println(arr[i]);
    }
  }
}
```

5. Dyn. Strukturen 9 Punkte

(a) Erweitern Sie die Klasse ZyklListe, die ihrerseits die Klasse Node verwendet, um eine Menge von **Integer Zahlen** in einer **zyklisch verketteten Liste** zu speichern, und dann auszudrucken. Fügen Sie die Erweiterungen (die Methoden „insert“ und „drucke“) in die Klasse ZyklListe an den vorgesehenen Stellen ein.
Zyklisch verkettet bedeutet, dass das jeweils letzte Element wieder auf das erste Element zeigt, sie stellen also eine Ringstruktur her. Beim Ausdrucken müssen die Integer Zahlen nicht ab/ oder aufsteigend sortiert sein. „insert“ 4 Punkte, „drucke“ 4 Punkte




```
class Node
{
   Node (Object d, Node n) {data = d; link = n;}
   Object data;
   Node link;
}

class ZyklListe
{
    Füge hier Methode "insert" ein
    Füge hier Methode "drucke" ein
   static Node start = null;

   public static void main (String [] args)
   {
     for (int i=0;i<10;i++)insert(i);
     drucke(start);
   }
}
      static void drucke(Node l)
      {
          Node head=l;
            do            //mindestens einmal ausführen
            {
                  System.out.print(l.data+" ");
                  l=l.link;
            }
          while(l!=head);

            System.out.println();
      }
static void insert(int i){
 if (start == null){
   start      = new Node(new Integer(i), null);
   start.link = start;}
 else start.link = new Node(new Integer(i), start.link);
}
```

(b) Die untenstehende Klasse BinTree (Binärbaum, der Integer Werte in den Blättern speichert) sei gegeben. Stellt die Methode check() eine InFix, PostFix oder Prefix Notation dar? 1 Punkte

```
class FinalBinTree
{
   private FinalBinTree left, right;
   private int value;

   FinalBinTree (FinalBinTree l, int v, FinalBinTree r)
   { left = l; value = v; right = r; }

   FinalBinTree (int v)
   { left = null; value = v; right = null; }

   void check()
   {
    if (left !=null) left.check();
    if (right !=null) right.check();
    System.out.print(value);
   }
}
```

Antwort: Postfix

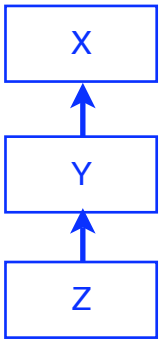
Name:	Matrikel-Nummer:

Aufgabe 6: Objekt-orientierte Programmierung (12 Punkte)

Ein hypothetisches Java-Programm besteht aus der folgenden Klassenhierarchie:

```
class X
{   void a() { b(); c(); }

    void b () { System.out.println("b aus Klasse X"); }
    void c () { System.out.println("c aus Klasse X"); }
}

class Y extends X
{   void c () { System.out.println("c aus Klasse Y"); }
}

class Z extends Y
{   void b () { System.out.println("b aus Klasse Z"); }
}
```

1. Stellen Sie die *Vererbungshierarchie* der drei Klassen grafisch als Baum dar. **(1 Pkt.)**



2. Welche *Ausgabe* erzeugen die folgenden Programmfragmente? **(3 Pkt.)**
   Berücksichtigen Sie dabei besonders die *dynamische Methodenbindung*.

Programmstück	Erzeugte Ausgabe
`X x1 = new X();` `x1.a();`	**b** aus Klasse **X** **c** aus Klasse **X**
`Y y1 = new Y();` `y1.a();`	**b** aus Klasse **X** **c** aus Klasse **Y**
`X x2 = new Z();` `x2.a();`	**b** aus Klasse **Z** **c** aus Klasse **Y**

Name:	Matrikel-Nummer:

3. Diese Teilaufgabe untersucht die *Polymorphie* bei *Variablentypen* anhand der Klassenhierarchie `X`, `Y` und `Z` von der vorherigen Seite. **(4 Pkt.)**
Welche der folgenden Programmfragmente sind korrekt, welche erzeugen Fehler zur Übersetzungszeit bzw. zur Laufzeit? Bei den fehlerhaften Programmstücken erläutern Sie bitte *kurz* die Fehlerursache.

Programmstück	korrekt?	Fehlerbeschreibung
`X x = new Y();`	**ja**	
`Y y = new X();`	**nein**	Fehler zur Übersetzungszeit: ein Objekt einer Oberklasse kann nicht an eine Variable vom Typ einer Unterklasse zugewiesen werden
`X x = new X();` `Y y = (Y) x;`	**nein**	Fehler zur Laufzeit: ein Objekt einer Oberklasse kann nicht per Typumwandlung / *type cast* in ein Objekts einer Unterklasse umgewandelt werden
`X x = new Z();` `Y y = (Y) x;`	**ja**	
`JPanel p = new JPanel();` `X x = (X) p;`	**nein**	Fehler zur Übersetzungszeit: die Variable `p` kann niemals ein Objekt der Klasse `X` oder einer Unterklasse von `X` enthalten, weil `JPanel` nicht mit `X` verwandt ist

4. Hier geht es um die *Vererbung* von *Methoden* bzw. damit *verwandte Konzepte*. **(2 Pkt.)**
Ergänzen Sie den Satz jeweils mit dem passenden *Fachbegriff* für die beschriebene Situation.

   a) Beide Methoden `m` sind *Objektmethoden* mit den *gleichen* Parameter- und Ergebnistypen.

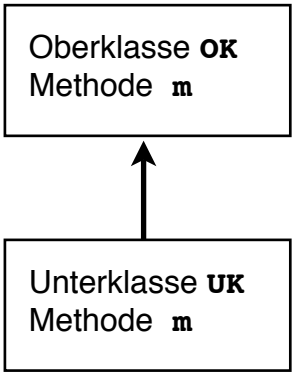
      Dann wird `m` aus `OK` in `UK` …überschrieben…

   b) Beide Methoden `m` sind *Objektmethoden* mit *unterschiedlichen* Parameter- und Ergebnistypen.

      Dann wird `m` aus `OK` in `UK` …überladen…

   c) Beide Methoden `m` sind *Klassenmethoden* mit den *gleichen* Parameter- und Ergebnistypen.

      Dann wird `m` aus `OK` in `UK` …verdeckt…



Name:	Matrikel-Nummer:

5. Das abgedruckte Interface `HasPrice` spezifiziert Objekte, deren Kaufpreis in Euro als `double`-Wert abgefragt werden kann. **(2 Pkt.)**

```
interface HasPrice
{   double getPrice ();
    void printPrice ();
}
```

Erweitern Sie die folgende Klasse `Computer`, so dass sie das *Interface* `HasPrice` *implementiert*.

```
class Computer implements HasPrice
{   private double kaufPreis;

    Computer (double k) { kaufPreis = k; }

    public void printPrice ()
    { System.out.println(getPrice()); }

    public double getPrice ()
    { return kaufPreis; }




}
```

Name:	Matrikel-Nummer:

Aufgabe 7: Ereignisbehandlung in Swing (12 Punkte)

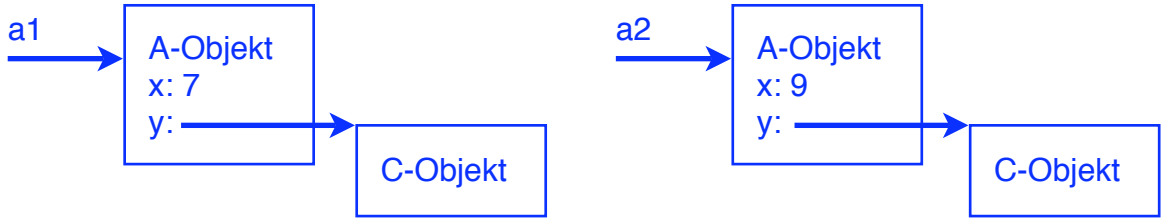
Betrachten Sie das folgende Programm, das eine benannte *innere Klasse* `C` verwendet.

```
class A extends B
{   private int x;
    private D y;

    class C extends D
    {
        void g (int p) { x = p; }
    }

    void m (int k) { y = new C(); y.g(k); }
}
```

1. Zeichnen Sie die *Objektstruktur*, die sich nach folgenden vier Anweisungen ergibt. Stellen Sie dabei auch die *Inhalte* aller *Objektvariablen* dar. **(2 Pkt.)**

```
A a1 = new A(); a1.m(7);
A a2 = new A(); a2.m(9);
```

Objektstruktur mit allen Objektvariablen:



2. Schreiben Sie die Klasse `A` so um, dass anstelle der inneren Klasse `C` eine *anonyme* innere Klasse verwendet wird. **(2 Pkt.)**

```
class A extends B
{   private int x;
    private D y;

    void m (int k)
    {   y = new D()
            {
                void g (int p) { x = p; }
            }

        y.g(k);
    }
}
```

Name:	Matrikel-Nummer:

3. In der folgenden Fensterklasse hat der Programmierer die Ereignisbehandlung für das `JButton`-Objekt vergessen. **(4 Pkt.)**

```
import javax.swing.*; import java.awt.*; import java.awt.event.*;

class CounterFrame extends JFrame
{   private JButton but;

    CounterFrame (String t)
    {   super(t);
        Container content = getContentPane();
        but = new JButton("0-mal betätigt");

        /* Stelle 1 */

        content.add(but);
        pack();
        setVisible(true);
    }

    /* Stelle 2 */
}
```

Erweitern Sie die Klasse `CounterFrame` um die Ereignisbehandlung für die Schaltfläche `but`, so dass die Anzahl Betätigungen gezählt und angezeigt wird. Das heisst, nachdem die Schaltfläche dreimal angeklickt wurde, soll die Beschriftung der Schaltfläche lauten: "3-mal betätigt".

Hinweis: Verwenden Sie die Objektmethode `void setText (String text)` aus der Klasse `JButton`, um die Beschriftung einer Schaltfläche zu verändern.

Notwendige *Ergänzung* an der markierten **Stelle 1**:

```
but.addActionListener (new Counter());
```

Notwendige *Ergänzung* an der markierten **Stelle 2**:

```
class Counter implements ActionListener
{   private int value = 0;

    public void actionPerformed (ActionEvent e)
    {
        value += 1;
        but.setText(value + "-mal betätigt");

        // alternativ:
        ((JButton) e.getSource()).setText(value + "-mal betätigt");
    }
}
```

Name:	Matrikel-Nummer:

4. Beim *Model-View*-Paradigma wird an zwei verschiedenen Stellen zu unterschiedlichen Zwecken das *Observer*-Prinzip eingesetzt. Beantworten Sie die folgenden Fragen jeweils getrennt für beide Anwendungen. **(3 Pkt.)**

   a) Welches Objekt ist der *Beobachtungsgegenstand* (wird beobachtet)?

1. Anwendung Observer-Prinzip	2. Anwendung Observer-Prinzip
die Swing-Komponente (das View-Objekt / der View)	die Datenstruktur (das Model-Objekt / das Modell)

   b) Welches Objekt ist der *Beobachter* (wird benachrichtigt)?

1. Anwendung Observer-Prinzip	2. Anwendung Observer-Prinzip
ein ActionListener-Objekt (ein Listener-Objekt / das Programm)	die Swing-Komponente (View-Objekt / ChangeListener-Obj.)

   c) Wodurch wird die *Benachrichtigung* des Beobachter-Objekts *ausgelöst*?

1. Anwendung Observer-Prinzip	2. Anwendung Observer-Prinzip
Benutzeraktion, z.B. Anklicken der Swing-Komponente, Texteingabe, Druck auf Return/Enter	Änderungen an der Datenstruktur (dem Modell), die das Programm vornimmt

5. Was ist eine *Adapterklasse* (wie z.B. `WindowAdapter`)? **(1 Pkt.)**

Eine Adapterklasse implementiert alle Methoden eines Listener-Interface als Methoden mit leerem Rumpf. Möchte ein Programm nur auf einige Methoden in einem Listener-Interface reagieren, verwendet es eine Unterklasse der Adapterklasse, die nur die interessierenden Methoden überschreibt.

[Adapterklassen sind nur sinnvoll für Listener-Interfaces, die mehr als eine Methode definieren, wie z.B. `WindowListener`, aber z.B. nicht `ActionerListener`.]

Name:	Matrikel-Nummer:

Aufgabe 8: Parallele Prozesse (12 Punkte)

1. Es geht um ein Anwendungsbeispiel für *unabhängige* parallele Prozesse. **(4 Pkt.)**
   Für die Herstellung eines Computer-generierten Zeichentrickfilms wird die folgende Klasse `SceneRenderer` verwendet. Jedes Objekt dieser Klasse repräsentiert eine Szene des zu erstellenden Films.

```
class SceneRenderer
{
    SceneRenderer (int nr) { … }

    void render () { … }
    void export (String fileName) { … }
}
```

Der fertige Film wird aus 10 Szenen mit den Nummern 1 bis 10 bestehen. Um die Fertigstellung des Films zu beschleunigen, soll die Herstellung der Szenen durch parallele Prozesse erfolgen.

Ergänzen Sie eine geeignete Prozessklasse `SceneThread`, sowie eine Klasse `FilmMaker` mit einem Hauptprogramm, das die 10 Szenen in unabhängigen Prozessen parallel berechnet (Methode `render`) und ausgibt (Methode `export`). Die erzeugten Dateien sollen die Namen `"Take1"` bis `"Take10"` tragen.

Hinweise: Das Hauptprogramm muss *nicht* auf die Fertigstellung der 10 Dateien warten. Für die Definition der Prozessklasse `SceneThread` gibt es zwei Möglichkeiten; suchen Sie sich eine aus.

Implementierung der Klassen `SceneThread` und `FilmMaker`:

```
class SceneThread extends Thread // oder: implements Runnable
{   private int sceneNr;

    SceneThread (int nr) { sceneNr = nr; }

    public void run ()
    {   SceneRenderer r = new SceneRenderer(sceneNr);
        r.render();
        r.export("Take" + sceneNr);
    }
}

class FilmMaker
{
    public static void main (String[] args)
    {
        for (int i = 1; i <= 10; i++)
            new SceneThread(i).start();

            // oder bei SceneThread implements Runnable:
            new Thread(new SceneThread(i)).start();
    }
}
```

Name:	Matrikel-Nummer:

2. In einer Klasse sollen *kritische Abschnitte* durch *gegenseitigen Ausschluss* geschützt werden. **(6 Pkt.)**
   Gegeben sei folgende Klasse aus einem Flugbuchungssystem. Jedes Objekt dieser Klasse speichert die noch verfügbaren Sitzplätze für eine bestimmte Flugverbindung.

```
class Flug
{   private int flugNr, freiePlätze;

    Flug (int nr, int sitze) { flugNr = nr; freiePlätze = sitze; }

    boolean buche (int p)
    {   if (freiePlätze < p)
            return false;   // Buchung nicht möglich
        else
        {   freiePlätze -= p;   // Buchung durchführen
            return true;        // Buchung hat geklappt
        }
    }

    void storniere (int p)
    {   int tmp = freiePlätze;
        tmp = tmp + p;
        freiePlätze = tmp;
    }

    int flugNummer () { return flugNr; }

    void flugAbsagen () { freiePlätze = 0; }
}
```

Objekte der Klasse `Flug` sollen in einem Programm mit *mehreren* parallelen Prozessen benutzt werden. Welche der Methoden in Flug müssen dazu in `synchronized`-Methoden umgewandelt werden, um ein korrektes Funktionieren des parallelen Programms sicher zu stellen?

Begründen Sie Ihre Entscheidungen! Dazu geben Sie für jede Methode, die `synchronized` sein muss, jeweils einen *verzahnten Ablauf* mit zwei Prozessen *p1* und *p2* an, der zu einem fehlerhaften Programmverhalten führt. Nehmen Sie an, dass *p1* und *p2* auf demselben `Flug`-Objekt `f` arbeiten:

```
Flug f = new Flug(1313, 100);
```

Beispiel-Begründung für Methode `m`**:** Prozess *p1* ruft `m` auf, *p2* ruft Methode … auf

Ablauf:

Fehlverhalten des Programms beschreiben

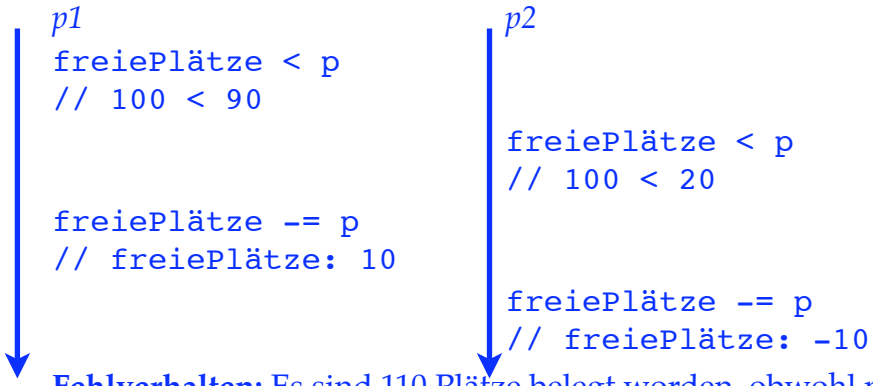
Name:	Matrikel-Nummer:

Folgende Methoden in der Klasse `Flug` müssen als `synchronized` definiert werden (jeweils mit Begründung wie oben skizziert):

Die Methoden **`buche`**, **`storniere`** und **`flugAbsagen`** müssen als `synchronized` definiert werden.

Begründung für Methode `buche`:
Prozess *p1* ruft `f.buche(90)` auf, *p2* ruft Methode `f.buche(20)` auf

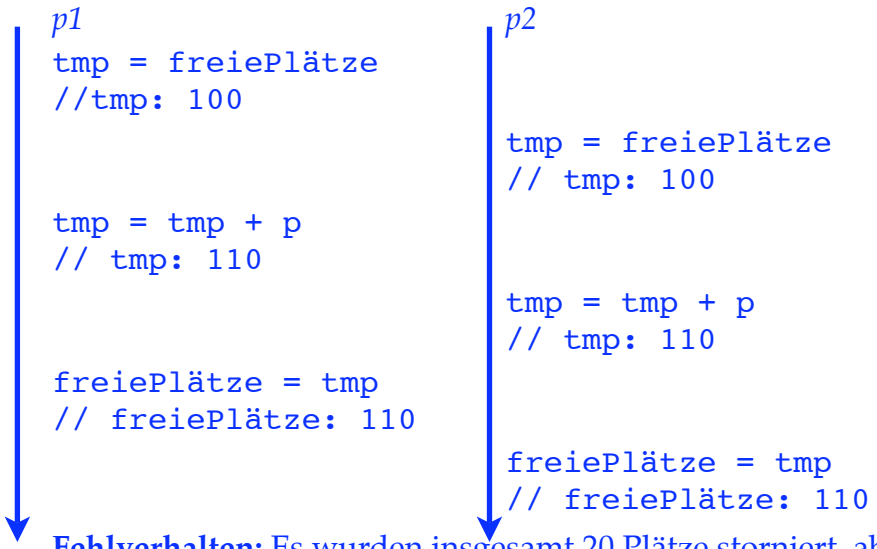
Ablauf:



Fehlverhalten: Es sind 110 Plätze belegt worden, obwohl nur 100 Plätze verfügbar waren.

Begründung für Methode `storniere`:
Prozess *p1* ruft `f.storniere(10)` auf, *p2* ruft Methode `f.storniere(10)` auf

Ablauf:

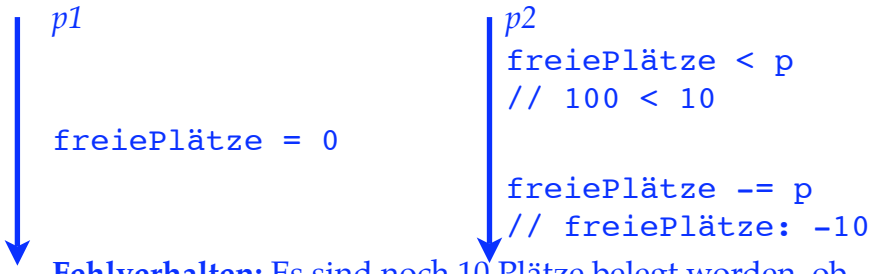


Fehlverhalten: Es wurden insgesamt 20 Plätze storniert, aber nur 10 Plätze davon sind als verfügbar gespeichert worden (*lost update*)

Name:	Matrikel-Nummer:

Begründung für Methode `flugAbsagen`:
Prozess *p1* ruft `f.flugAbsagen()` auf, *p2* ruft Methode `f.buche(10)` auf

Ablauf:



Fehlverhalten: Es sind noch 10 Plätze belegt worden, obwohl der Flug bereits kurz vorher abgesagt worden war.

[Die Methode `flugNummer()` und der Konstruktor müssen nicht `synchronized` sein:
Die Methode `flugNummer()` besteht nur aus einer einzelnen Leseoperation und ist deshalb unteilbar (und kann auch keine Konsistenzprobleme verursachen).
Ein Konstruktor arbeitet immer auf einem neu erzeugten Objekt, auf das noch kein anderer Thread Zugriff hat. Daher ist dort kein gegenseitiger Ausschluss (bezogen auf dieses Objekt) nötig.]

Name:	Matrikel-Nummer:

3. Was bedeutet der Begriff *Bedingungssynchronisation*? **(1 Pkt.)**

Ein Thread/Prozess wartet auf eine Bedingung, die durch Operationen in einem anderen Thread/Prozess hergestellt wird (eintritt). Zum Beispiel, ein Thread wartet bis ein Puffer begrenzter Größe nicht mehr komplett gefüllt ist, bevor er ein neues Element zum Puffer hinzufügt. Diese Bedingung tritt etwa dann ein, wenn ein anderer Thread ein Element aus dem Puffer entnimmt.

[Während ein Thread auf das Eintreten seiner Wartebedingung wartet, gibt er den Monitor frei.]

4. Warum muss die Methode `wait` immer in einer Schleife aufgerufen werden? **(1 Pkt.)**

Nach der Benachrichtigung des Threads [per `notifyAll()`] und bevor der Thread wieder Zugriff auf den Monitor erhält, kann ein anderer Thread die Wartebedingung bereits wieder invalidiert haben.