1. Verständnisfragen I: (7 Punkte)

a) Sind die folgenden Variablendeklarationen/Zuweisungen gültig oder ungültig? 1.5P

Zuweisung	Gültig/ungültig
`int i=127`	g
`int l=300000000`	g
`byte b=127`	g
`byte b2=-128`	g
`byte b1=300`	u
`int j=300`	g
`String a='a'`	u
`Boolean istRentner=(i>65)`	g
`boolean istGleich=(b= =i)`	g
`int i2=067`	g
`double d3=6.0e+3`	g

b) Sind die folgenden Namen als Variablennamen in Java zulässig? 1.5 P

Variablennamen	zulässig	Nicht zulässig	Bei Nicht zulässig: Begründung
rate1	x		
1stPlayer		x	Alphazeichen muss zuerst sein
myprogram.java		x	Punkt nicht zulässig
long		x	long ist Schlüsselwort
TimeLimit	x		erlaubt, aber schlechte Wahl: timeLimit besser
numberOfWindows	x		

c) Welchen Output generiert der folgende Code? 1.5 P

```
int i;
int[] a = new int[10];
for (i=0; i<a.length; i++)
  a[i]=2*i;
for (i=0; i<a.length; i++)
  System.out.print(a[i] + " ");
  System.out.println();
```

0 2 4 6 8 10 12 14 16 18

Hinweis: auch wenn die letzte Zeile eingerückt ist, sie ist außerhalb der for-Schleife.

d) Was macht die folgende Schleife? Schreiben Sie die Formel als mathematischen Ausdruck auf. 1 P

```
for (i=1; i<10; i++)
{
   if (i%2==0) d = d - 1.0/i;
```

```
    else        d = d + 1.0/i;
}
```

1 – 1/2 + 1/3 – 1/4 + 1/5 – 1/6 + 1/7 – 1/8 + 1/9 =0.745634….

e) Schreiben Sie die `for` Schleife aus Aufgabe 1.d als `while` Ausdruck auf (1.5 P.)

```
int ilauf=1;
while (ilauf<10)
{
     if (ilauf%2==0) d = d - 1.0/ilauf;
         else        d = d + 1.0/ilauf;
         ilauf=ilauf+1;
}
```

2. Verständnisfragen II: (6 Punkte)

a) **Im folgenden Programm soll vor Ziehen einer Wurzel geprüft werden, ob die Zahl zwischen 0 und 400 liegt.** Ist sie kleiner als 0, soll sie auf 0 gesetzt werden; ist sie größer als 400, soll sie auf 400 gesetzt werden. Welche der Lösungen sind richtig, welche falsch**?** **(1P.)**

if (zahl < 0) zahl = 0; **if** (zahl > 400) zahl = 400;	**R**
if (zahl < 0) zahl = 0; **else** **if** (zahl > 400) zahl = 400;	**R**
if (zahl >= 0) **if** (zahl > 400) zahl = 400; **else** zahl = 0;	**F**

if (zahl >= 0)
  if (zahl <= 400)
  else zahl = 400;
else zahl = 0;
	F

b) **Gegeben sei das folgende Java Programm. Welchen Output generiert das Programm? (2 P):**

```
public class Rek {
```

```
    public static void main (String[] args) {
      rek(4);
    }
  private static void rek(int z) {
     z--;
     System.out.println(z);
     if (z!=0) rek(z);
     System.out.println(z);
  }
}
```

3
2
1
0
0
1
2
3

c) **Schreiben Sie eine Methode „`reziprok`“,** die als Eingabe einen Integer Wert n nimmt, und die die Summe (also $1 + {}^1/_2 + {}^1/_3 + \ldots + {}^1/_n$), **das Produkt** (also $1 \cdot {}^1/_2 \cdot {}^1/_3 \cdot \ldots \cdot {}^1/_n$) und den **Durchschnitt** der **ersten n** Reziproken Werte **ausdruckt**. **3 Punkte**

z.b. ergibt der Aufruf

```
reziprok (2)
```

das Ergebnis:

```
Summe 1.5
Produkt 0.5
Durchschnitt 0.75
```

und der Aufruf

```
reziprok (4)
```

das Ergebnis:

```
Summe 2.0833…
Produkt 0.4166…
Durchschnitt 0.520833…
```

```
static void reziprok(int n)
{
  double sum = 0;
  double product = 1;
  for (int i=1; i<=n; i++)
  {
     sum += 1.0/i;
     product *= 1.0/i;
  }
  double average = sum / n;

  // output results
  System.out.println(sum + "\tSum");
  System.out.println(product + "\tProduct");
  System.out.println(average + "\tAverage");
```

3. Arrays (8 Punkte)

a) Schreiben Sie eine Methode `int minGleich(int[] a)`, welche die minimale Anzahl (ungleich 0) gleicher int-Werte ermittelt. **(2.5 P)**

Beispiel: Im Array {7,3,5,2,5,3,2,3} ist die minimale Anzahl gleicher Werte 1, da die Zahl 7 einmal vorkommt und alle anderen Zahlen zwei oder dreimal.
Im Array {1001,1002,1003,1004, 1001,1002,1003,1004} ist die minimale Anzahl 2, da alle Zahlen zweimal vorkommen.

```
static int minGleich(int []a) {
     int min=a.length+1;
     for (int i=0;i<a.length;i++) {
          int m=0;
          for (int j=0;j<a.length;j++){
               if (a[i]==a[j]) m++;
          }
          if (m <min)min=m;
     }
     return min;
}
```

b) Die folgende Schleife füllt ein zweidimensionales Array:

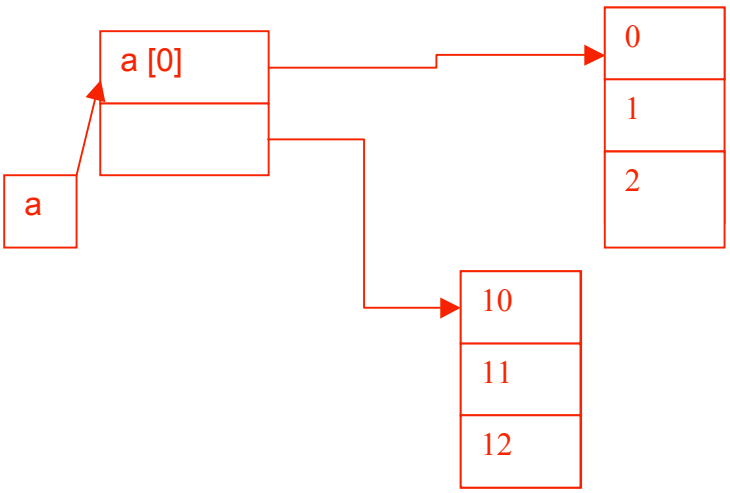
```
for (int i=0; i<2; i++)
   for (int j=0; j<3;j++)
      a[i][j]=10*i+j;
```

i) Wie würde ein System.out.println Befehl lauten, der die zweite Dimension des Arrays (Dimension j) ausgibt? 1 P

```
System.out.println (a[0].length);
```

ii) Zeichnen Sie die Datenstruktur dieses Arrays auf: 1 P

Lösung:



iii) Um die korrekte Indizierung im Array zu prüfen und gegebenenfalls eine Ausnahme auszulösen würde man welchen try-catch Befehl verwenden? Füllen Sie die „????“ unten entsprechend ein. 0.5 P.

try { … a[i][j] … }

catch (???????????????? e)

```
catch (e) { ... }
```

ArrayIndexOutOfBoundsException

c) Schreiben Sie eine Methode `objektvergleich` des Typs `boolean`, welche die Objekt-Elemente zweier eindimensionaler Arrays a und b vergleicht und bei Gleichheit „`equal`“ und ansonsten „`not equal`“ ausgibt.
Beachten Sie dass die Arrays mit Objekten statt mit einfachen Elementen gefüllt sind. (3 P)

```
class …
{
   static boolean objektvergleich (Integer a [], Integer b[]) {
   //Methode hier ausführen

   boolean eq=(a.length == b.length);

   for (int i=0; eq&& i<a.length; i++)
    eq=a[i].equals(b[i]);
   return eq;
   }

   }
   public static void main (String [] args) {
   Integer [] a=new Integer[10];
```

```
    Integer [] b=new Integer[10];
    //Schleife zum Füllen der Arrays

    boolean eq;
    eq=elementvergleich(a,b);
    System.out.println(eq? „equal“:“not equal“);
    }
}
```

4. Einfache Grundlagen der OO Programmierung (10 P.)

Das folgende objektorientiertes Programm verwaltet Buchdokumente. Es besteht aus mehreren Klassen und mehreren Methoden. Beantworten Sie die Fragen, die im Programm mit **//???** gekennzeichnet sind.

```
import java.util.*;

public class DocMan {
```
//??? Welche Bedeutung hat “static” in der folgenden Zeile und warum ist es nötig
Die Methode showAll wird in der Methode main aufgerufen; da main eine Klassenmethode ist (…static main…) muss auch showAll als Klassenmethode deklariert werden.

```
      public static void showAll(Dokument[] dokumente) {
            for (int i = 0; i < dokumente.length; i++) {
                  System.out.println("-----------------------------------");
                  dokumente[i].show();
            }
      }

      public static void main(String[] args) {
            Buch b1 = new Buch(new Autor("Monson-Haefel", "Richard"), "Enterprise
JavaBeans", "", 2, 2000, "1-56592-869-5");
            Buch b2 = new Buch(new Autor("Knudsen", "Jonathan"), "Java 2D
Graphics", "", 1, 1999, "1-56592-484-3");

            Autor autor = new Autor("Sycara", "Katja");
            Artikel a1 = new Artikel(autor, "Multiagent Systems", "", "AI
magazine", "2/1998", "79-92");
            Artikel a2 = new Artikel(autor, "Negotiation planning", "An AI
approach", "European Journal of Operational Research", "2/1990", "216-234");

            Dokument[] dokumente = { b1, b2, a1, a2 };
            showAll(dokumente);
      }
}

public class Dokument
{
      private Autor     autor;
      private String    titel;
      private String    untertitel;


      public Dokument(Autor autor, String titel, String untertitel) {
```
//??? füllen Sie den Inhalt dieses Konstruktors aus: es ist ein reiner Kopierkonstruktor
```
            this.autor = autor;
```

```
            this.titel = titel;
            this.untertitel = untertitel;
      }

      public Autor      getAutor()              { return autor; }
      public String     getTitel()              { return titel; }
      public String     getUntertitel()         { return untertitel; }

      public void show() {
            ///??? Schreiben Sie hier eine Methode show, die Autor, Titel und Untertitel auf dem
Bildschirm ausgibt.
            System.out.println("Autor: " + autor.getName() + ", " +
              autor.getVorname());
            System.out.println("Titel: " + getTitel() + ", Untertitel: " +
              getUntertitel());
            }
      }

}

public class Buch extends Dokument {
      private int             auflage;
      private int             jahr;
      private String    isbn;

      public Buch(Autor autor, String titel, String untertitel, int auflage, int
jahr, String isbn) {
            super(autor, titel, untertitel);
            this.auflage = auflage;
            this.jahr = jahr;
            this.isbn = isbn;
      }

      public int              getAuflage()            { return auflage; }
      public int              getJahr()                     { return jahr; }
      public String     getIsbn()                 { return isbn; }

      public void             setAuflage(int auflage)                         {
this.auflage = auflage; }
      public void       setJahr(int jahr)
      { this.jahr = jahr; }
      public void       setIsbn(String isbn)                                  {
this.isbn = isbn; }

      public void show() {
            super.show();
            System.out.println("Auflage: " + getAuflage());
            System.out.println("Jahr: " + getJahr());
            System.out.println("ISBN: " + getIsbn());
      }
}
public class Artikel extends Dokument {
      private String    zeitschrift;
      private String    ausgabe;
      private String    seiten;

      public Artikel(Autor autor, String titel, String untertitel, String
zeitschrift, String ausgabe, String seiten) {
            super(autor, titel, untertitel);
            this.zeitschrift = zeitschrift;
            this.ausgabe = ausgabe;
            this.seiten = seiten;
      }
```

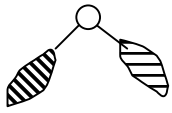
```
	public String	getZeitschrift()		{ return zeitschrift; }
	public String	getAusgabe()			{ return ausgabe; }
	public String	getSeiten()			{ return seiten; }

	public void		setZeitschrift(String zeitschrift)  {
this.zeitschrift = zeitschrift; }
	public void		setAusgabe(String ausgabe)		{ this.ausgabe =
ausgabe; }
	public void		setSeiten(String seiten)				{
this.seiten = seiten; }

	public void show() {
		super.show();
		System.out.println("Zeitschrift: " + getZeitschrift());
		System.out.println("Ausgabe: " + getAusgabe());
		System.out.println("Seiten: " + getSeiten());
	}
}
public class Autor {
```

//??? Vervollständigen Sie die Klasse Autor mit dem entsprechenden Konstruktor und den getter und setter Methoden.

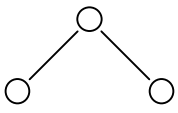
```
public Autor(String name, String vorname) {
		this.vorname = vorname;
		this.name = name;
	}

	public String getVorname() { return vorname; }
	public String getName() { return name; }

	public void setVorname(String vorname){   this.vorname = vorname; }
	public void setName(String name)    { this.name = name; }

}
```

5. Dynamische Datenstrukturen (5 P.)

Ergänzen Sie die Klasse `Autumn`, welche die Klasse `LeaveTree` verwendet, um den folgenden Baum aufzubauen, und als postfix auslesen.
Anmerkung: `LeaveTree` ist ähnlich der Klasse `BinTree` aus der Vorlesung, nur speichert `LeaveTree` in den äusseren Blättern Strings und die inneren Knoten sind nur Verzweigungen von weiteren Binärbäumen. Die folgenden Konstruktionen werden durch LeaveTree erlaubt:



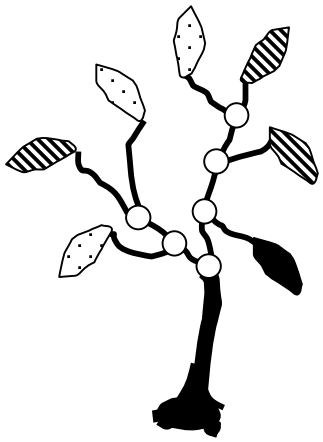
Knoten mit 2 bunten Blättern als Söhnen



Knoten mit 2 Knoten als Söhnen

Knoten mit je einem Knoten und einem bunten Blatt als Söhnen

Gefragter Baum:



 Grünes Blatt

 Braunes Blatt

 Rotes Blatt

○ Knoten

```
class LeaveTree // Binäre Bäume mit Strings in den Blättern
{
   private LeaveTree left, right;
   private String colorLeft,colorRight;

  // Konstruktor für innere Knoten

   LeaveTree (String cl, LeaveTree r) { //links ein Blatt
       right = r; colorLeft = cl;  }
   LeaveTree (LeaveTree l, String cr) { //rechts ein Blatt
       left = l; colorRight = cr; }
   LeaveTree (LeaveTree l, LeaveTree r) { //nur Astverzweigung
       left = l; right=r; }

  // Konstruktor für Blattknoten
   LeaveTree (String cl,String cr)
   { colorRight=cr; colorLeft=cl; }    //links und rechts je ein
Blatt

 void postFix() {   // Ausgabe in Postfix-Form ohne Klammern
     if (left !=null) left.postFix();  // Drucke linken Teilbaum
     if (right !=null) right.postFix(); // Drucke rechten Teilbaum
     if (colorLeft!=null)System.out.print(colorLeft+" ");
                                          //  Drucke eigenen Wert
     if (colorRight!=null)System.out.print(colorRight+" ");
   }
void inFix() { // Ausgabe in Infix-Form, geklammert

     System.out.print("(");

     if (left !=null) left.inFix();    // Drucke linken Teilbaum

     if (colorLeft!=null)System.out.print(colorLeft+" ");
                                        // Drucke eigenen Wert
     if (colorRight!=null)System.out.print(colorRight+" ");
```

```
      if (right !=null) right.inFix();   // Drucke rechten Teilbaum

      System.out.print(")");

    }

}
class Autumn
{
      public static void main (String args[]) {

      //Aufbau und Ausgabe eines Bintrees mit bunten Blättern:

      LeaveTree b1=new LeaveTree("gruen","braun");
      LeaveTree b2=new LeaveTree("braun", "gruen");
      LeaveTree b3=new LeaveTree(b1,"braun");
      LeaveTree b4=new LeaveTree(b3,"rot");

      LeaveTree b5=new LeaveTree("gruen",b2);

      LeaveTree b6=new LeaveTree(b5,b4);


      System.out.print("\n\nBaum b6, Postfix\n"); b6.postFix();
      System.out.print("\n\nBaum b6, InFix\n"); b6.inFix();
  }
}
```

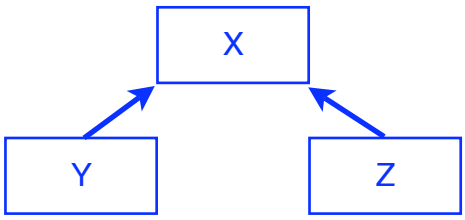
Name:	Matrikel-Nummer:

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 X"); }
    void c () { b(); }
}

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

class Z extends X
{   void b () { System.out.println("b aus 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 **X** **b** aus **X**
`Y y1 = new Y();` `y1.a();`	**b** aus **X** **c** aus **Y**
`X x2 = new Z();` `x2.a();`	**b** aus **Z** **b** aus **Z**

Name:	Matrikel-Nummer:

3. Hier geht es um die *Vererbung* von *Methoden* bzw. damit *verwandte Konzepte*. Beschreiben Sie die Beziehung zwischen den Methoden `p`, `q` und `r` in der folgenden Klassenhierarchie jeweils mit dem passenden *Fachbegriff*. **(3 Pkt.)**

```
class Ober
{   int p (int x, int y) { return x+y; }
    int q (int x) { return x-1; }
    static int r (int x) { return 2*x; }
}

class Unter extends Ober
{   int p (int x) { return x+1; }
    int q (int x) { return x-2; }
    static int r (int x) { return x/2; }
}
```

In der Klasse `Unter` wird die Methode **`p`** überladen.

In der Klasse `Unter` wird die Methode **`q`** überschrieben.

In der Klasse `Unter` wird die Methode **`r`** verdeckt.

4. Das abgedruckte Interface `Wearable` spezifiziert Kleidungsstücke, deren Farbe und Konfektionsgröße abgefragt werden können. **(2 Pkt.)**

```
interface Wearable
{   Color getColor();
    int getSize ();
    void printSize ();
}
```

Erweitern Sie die folgende Klasse `Shirt`, so dass sie das *Interface* `Wearable` *implementiert*.

```
class Shirt implements Wearable
{   private Color farbe;
    private int größe;

    Shirt (Color f, int g) { farbe = f; größe = g; }

    public void printSize () { System.out.print(getSize()); }

    public Color getColor () { return farbe; }

    public int getSize ()    { return größe; }




}
```

Name:	Matrikel-Nummer:

5. Welchen Vorteil hätte es, das Interface `Wearable` in eine *abstrakte Klasse* umzuwandeln? Denken Sie insbesondere an die Methode `printSize`. **(1 Pkt.)**

Die Methode `printSize` könnte direkt in der abstrakten Klasse implementiert werden [unter Benutzung von `getSize`]. Unterklassen würden diese Implementierung erben und unverändert übernehmen können. [Bei Verwendung eines Interfaces muß die Methode `printSize` in jeder Klasse separat implementiert werden.]

6. Geben Sie die *Signatur* (Parameterliste und Ergebnistyp) einer Methode `findCloth` an, die in einem Array nach einem beliebigen Kleidungsstück mit einer vorgegebenen Farbe und Größe sucht und dieses zurückgibt. **(1 Pkt.)**

Hinweis: Die Methode `findCloth` soll *nicht* komplett programmiert werden!

```
Wearable findCloth      ( Wearable[] sachen, Color farbe,
                          int größe )
{ … }
```

7. Begründen Sie kurz, warum das folgende Programmstück *nicht* korrekt ist. **(1 Pkt.)**

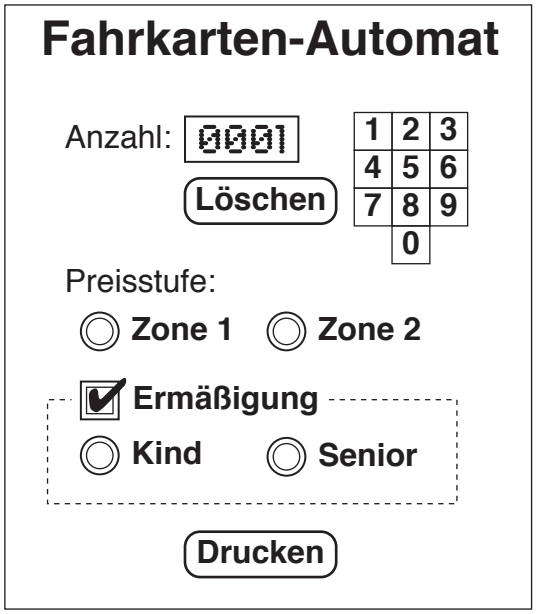
```
Wearable w = new Shirt(Color.red, 42);
Shirt s = w;
```

Objekte vom Typ des Interfaces `Wearable` bieten nur die im Interface definierten Methoden an: sie sind eine Verallgemeinerung von z.B. `Shirt`-Objekten. [oder: `Shirt`-Objekte sind eine Spezialisierung von `Wearable`-Objekten]. Das heißt, bei der zweiten Zuweisung oben findet eine Typeinengung statt, die vom Java-Compiler nur mit einem expliziten einengenden *type cast* akzeptiert wird [`Shirt s = (Shirt) w;`]

Name:	Matrikel-Nummer:

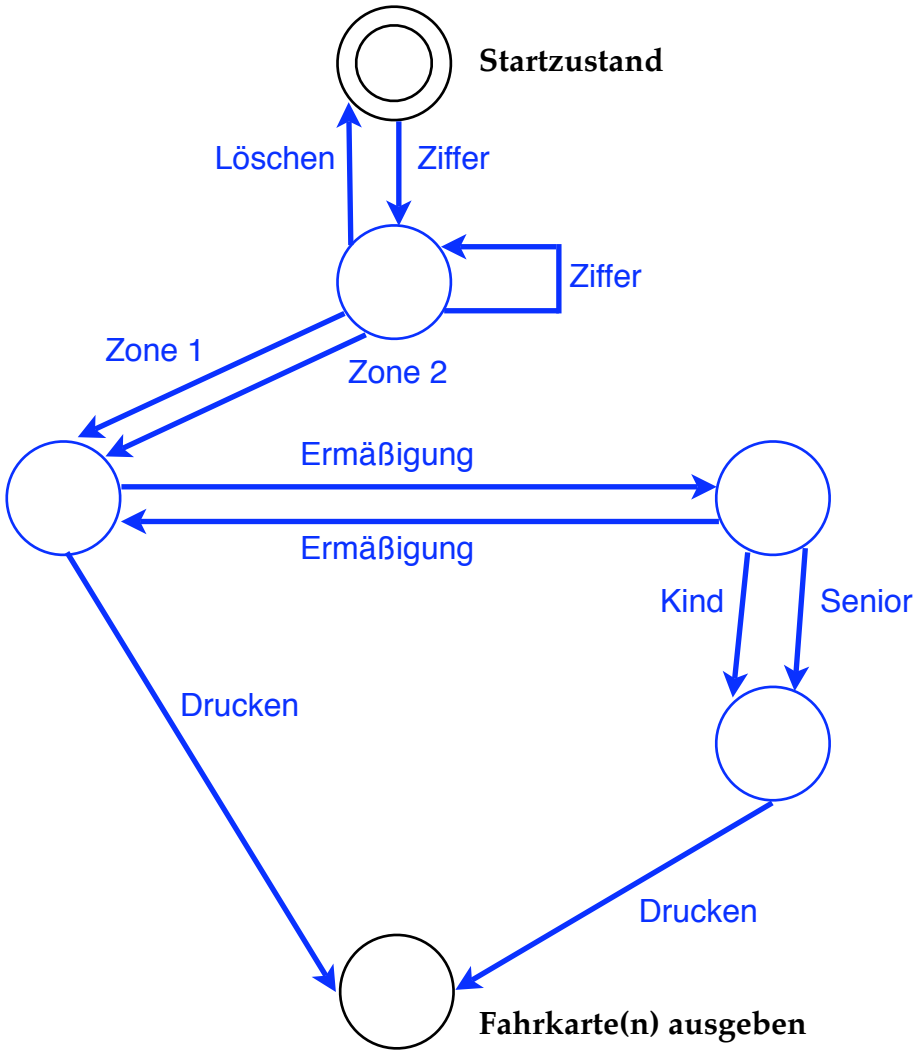
Aufgabe 7: Swing-Programmierung (12 Punkte)

Dieses Bild zeigt die Benutzungsoberfläche eines Fahrkartenautomaten, der wie folgt bedient wird:



- Zuerst legt man die Anzahl der gewünschten Fahrkarten fest. Dazu ist *mindestens* eine Ziffer einzugegeben. Korrekturen erfolgen mit der Taste "Löschen": diese löscht alle bereits eingegebenen Ziffern.
- Dann *muss* eine der beiden Preisstufen (Zone 1 oder 2) gewählt werden.
- Jetzt *kann* die Option "Ermäßigung" aktiviert werden (wie im Bild dargestellt). Wenn die Option aktiviert ist, *muss* man eine der beiden Ermäßigungsarten (Kind oder Senior) wählen. Alternativ kann man auch die Option "Ermäßigung" wieder deaktivieren.
- Zuletzt startet die Taste "Drucken" die Ausgabe der Fahrkarten. (Bezahlt wird bei diesem Fahrkartenautomaten nicht.)

1. Modellieren Sie die zulässigen *Ereignisfolgen* (Folgen von Bedienereignissen) durch einen *endlichen Automaten*. Start- und Endzustand sind bereits vorgegeben. **(4 Pkt.)**



Name:	Matrikel-Nummer:

2. In der folgenden Fensterklasse hat der Programmierer die *Ereignisbehandlung* für die Swing-Komponenten vergessen. **(4 Pkt.)**

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

class TextLengthFrame extends JFrame
{   private JTextField input;
    private JLabel result;

    TextLengthFrame (String t)
    {   super(t);
        Container content = getContentPane();
        content.setLayout(new GridLayout(2, 1));
        input = new JTextField();
        result = new JLabel("Länge der Eingabe: 0 Zeichen");

        /* Stelle 1 */

        content.add(input); content.add(result);
        pack(); setVisible(true);
    }

    /* Stelle 2 */
}
```

Erweitern Sie die Klasse `TextLengthFrame` um die Ereignisbehandlung für folgende Funktionalität: Wenn der Benutzer im Eingabefeld `input` die Return-Taste betätigt, wird die Länge des Textes im Eingabefeld in der Textzeile `result` angezeigt. Beispiel: Nach Eingabe von "Hallo" plus Return-Taste im Textfeld lautet die Anzeige:
"Länge der Eingabe: 5 Zeichen".

Hinweis: Verwenden Sie die Objektmethoden `String getText()` und `void setText(String text)`, um den Text einer Swing-Komponente abzufragen bzw. zu verändern. Benutzen Sie eine innere Klasse für die Ereignisbehandlung.

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

```
input.addActionListener(new LengthCounter());
```

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

```
class LengthCounter implements ActionListener
{
    public void actionPerformed (ActionEvent e)
    {
        int len = input.getText().length();
        result.setText("Länge der Eingabe: " + len + " Zeichen");
    }
}
```

Name:	Matrikel-Nummer:

3. Bei der Ereignisbehandlung sind verschiedene Konstellationen zwischen Swing-*Komponenten* und *Listener*-Objekten möglich. Beschreiben Sie kurz, in welchen Situationen die folgenden Objektstrukturen sinnvoll sind. **(2 Pkt.)**

   a) *Ein* Listener-Objekt wird gleichzeitig *mehreren* Swing-Komponenten zugeordnet, …

wenn mehrere Komponenten die gleiche Funktionalität im Programm auslösen können. [Zum Beispiel: Eingabe von Return in einem Textfeld und das Anklicken einer Schaltfläche oder: dasselbe Listener-Objekt für alle Schließknöpfe von Fenstern eines Programms.]

   b) *Mehrere* Listener-Objekte werden gleichzeitig *einer* Swing-Komponente zugeordnet, …

wenn eine Komponente mehrere Einzelfunktionen unmittelbar nacheinander vornimmt. [Zum Beispiel: Schaltfläche zum Schließen aller Fenster eines Programms besitzt ein Listener-Objekt für jedes momentan geöffnete Fenster eines Programms.] [Die Reihenfolge in der die einzelnenen Listener-Objekte benachrichtigt werden ist dabei undefiniert.]

4. In welcher *Reihenfolge* werden die vier zentralen *Applet-Methoden* `destroy()`, `init()`, `start()` und `stop()` aufgerufen? Zu welchen Zeitpunkten werden sie aufgerufen? **(2 Pkt.)**

Nr.	Methode	wird aufgerufen, wenn das Applet …
1	`init()`	geladen wird
2	`start()`	seine Ausführung (wieder) beginnen soll
3	`stop()`	seine Ausführung unterbrechen soll
4	`destroy()`	beendet wird

Name:	Matrikel-Nummer:

Aufgabe 8: Parallele Prozesse (12 Punkte)

1. Welche zwei Möglichkeiten gibt es, in Java eine *Prozessklasse* `P` zu definieren? **(2 Pkt.)**
   Nennen Sie beide Möglichkeiten und geben Sie zusätzlich an, wie in beiden Fällen ein Objekt der Klasse `P` erzeugt und als paralleler Prozess gestartet wird.

Man kann die Klasse `P` das Interface `Runnable` implementieren lassen oder `P` als Unterklasse der vordefinierten Klasse `Thread` definieren:

```
// 1. Möglichkeit
class P implements Runnable { … }

// Objekt erzeugen und als Prozess starten
new Thread(new P()).start();
```

```
// 2. Möglichkeit
class P extends Thread { … }

// Objekt erzeugen und als Prozess starten
new P().start();
```

2. Sei `t` eine Variable vom Typ `Thread`. **(1 Pkt.)**
   Was bewirkt dann der Aufruf `t.join()` ?

Der Prozess (*thread*), der den Aufruf ausführt wartet bis der durch die Variable `t` referenzierte Prozess terminiert ist. [Es sind also zwei Prozesse beteiligt: der aufrufende Prozess und der Prozess `t`.]

Name:	Matrikel-Nummer:

3. Es geht um *gegenseitigen Ausschluss* im Monitor. **(4 Pkt.)**
   In der folgenden Monitorklasse `Mon` gibt jede der drei Methoden am Anfang und am Ende ihren Namen am Bildschirm aus.

```
class Mon
{   synchronized void s ()
    {   System.out.println("Anfang s");
        System.out.println("Ende s");
    }

    synchronized void t ()
    {   System.out.println("Anfang t");
        System.out.println("Ende t");
    }

    void u ()
    {
        System.out.println("Anfang u");
        System.out.println("Ende u");
    }
}
```

Zwei Prozesse *p1* und *p2* rufen nun jeweils über eine Variable `v` Methoden für **dasselbe** `Mon`-Objekt auf. Welche der folgenden Bildschirmausgaben können dabei entstehen, welche Reihenfolgen sind nicht möglich? Begründen Sie ihre Antworten kurz.

(a)

`run`-Methode in *p1*	**`run`-Methode in *p2***	**Bildschirmausgabe**

`v.s();`	`v.t();`	`Anfang t`
		`Ende t`
		`Anfang s`
		`Ende s`

Ist diese Reihenfolge bei der Ausgabe möglich? (ja/nein)		ja
Begründung: Die beiden `synchronized`-Methoden werden nacheinander (unter gegenseitigem Ausschluss) durchgeführt.		

Name:	Matrikel-Nummer:

(b)

`run`-Methode in *p1*	**`run`-Methode in *p2***	**Bildschirmausgabe**

`v.s();`	`v.t();`	`Anfang s`
		`Anfang t`
		`Ende t`
		`Ende s`

Ist diese Reihenfolge bei der Ausgabe möglich? (ja/nein)		nein
Begründung: Die Ausführung der `synchronized`-Methode `s` kann nicht durch den Aufruf der `synchronized`-Methode `t` auf demselben Objekt unterbrochen werden.		

(c)

`run`-Methode in *p1*	**`run`-Methode in *p2***	**Bildschirmausgabe**

`v.s();`	`v.u();`	`Anfang s`
		`Anfang u`
		`Ende s`
		`Ende u`

Ist diese Reihenfolge bei der Ausgabe möglich? (ja/nein)		ja
Begründung: Die Methode `u` ist nicht als `synchronized` definiert und kann daher die Ausführung der `synchronized`-Methode `s` auf demselben Objekt unterbrechen.		

(d)

`run`-Methode in *p1*	**`run`-Methode in *p2***	**Bildschirmausgabe**

`v.s();`	`v.t();`
	`v.t();`
		`Anfang t`
		`Ende t`
		`Anfang s`
		`Ende s`
		`Anfang t`
		`Ende t`

Ist diese Reihenfolge bei der Ausgabe möglich? (ja/nein)		ja
Begründung: Das Prinzip des gegenseitigen Ausschlusses wird für die drei Aufrufe von `synchronized`-Methoden eingehalten. Der Prozess *p2* wird genau zwischen zwei solchen Aufrufen durch *p1* unterbrochen, was zulässig ist.		

Name:	Matrikel-Nummer:

4. Es geht um *Bedingungssynchronisation* im Monitor. **(5 Pkt.)**
   Folgende, noch unvollständige Klasse `Konto` soll in einer Bank-Software verwendet werden.

```
class Konto
{   private int kontoNr;
    private double kontoStand, maxKredit;

    Konto (int nr, double limit)
    { kontoNr = nr; kontoStand = 0.0; maxKredit = limit; }

    …
}
```

Ein Objekt für ein Konto mit einem maximalen Überziehungskredit von 1000 Euro wird z.B. so erzeugt:

```
Konto maxMustermann = new Konto(123456, 1000.0);
```

Erweitern Sie die Klasse `Konto` um zwei Operationen `einzahlen` und `abheben`, die einen als Parameter übergebenen Geldbetrag auf das jeweilige Konto einzahlen bzw. von diesem abheben. Verwenden Sie *Bedingungssynchronisation*, um sicher zu stellen, dass das Konto niemals über den maximalen Überziehungskredit hinaus überzogen wird. Für das obige Beispiel dürfte der Kontostand also niemals unter –1000 Euro fallen.

Zusätzliche Methoden in der Klasse `Konto`:

```
synchronized void einzahlen (double betrag)
{
    kontoStand += betrag;
    notifyAll();
}

synchronized void abheben (double betrag)
{
    while (kontoStand - betrag < -maxKredit)
    {
        try
        { wait(); }
        catch (InterruptedException e)
        { /* kann nicht auftreten */ }
    }
    kontoStand -= betrag;
}
```