FernUniversität in Hagen

Fakultät für Mathematik und Informatik

Lehrgebiet Programmiersysteme
Prof. Dr. Friedrich Steimann

FernUniversität in Hagen D-58084 Hagen

_____
(Name, Vorname)
_____
_____
(Straße, Nr.)
_____ _____
(PLZ) (Wohnort)
_____
(Land, falls außerhalb Deutschlands)

Kurs 1618 SS 2007

„Einführung in die objektorientierte Programmierung“

Klausur am 28.07.2007

Dauer: 3 Std., 10 – 13 Uhr

Lesen Sie zuerst die Hinweise auf der Rückseite!

Matrikelnummer: | | | | | | | | Geburtsdatum: | | | . | | . | | |

Klausurort: .....

Aufgabe	1					2			3		4					5	Summe
Teilaufgabe	a	b	c	d	e	a	b	c	a	b	a	b	c	d	e		
habe bearbeitet																	
maximal	5	4	6	3	7	2	12	16	3	7	6	2	2	2	8	15	100
erreicht																	
Korrektur																	

- ○ Herzlichen Glückwunsch, Sie haben die Klausur bestanden. Note: .....
- ○ Sie haben die Klausur leider nicht bestanden. Für den nächsten Versuch wünschen wir Ihnen viel Erfolg. Die Nachklausur findet am 22.09.2007 statt; wenn Sie teilnehmen möchten, melden Sie sich bitte bis zum 10.09.2007 an. (Näheres siehe unser Informationsheft vom 29.05.2007).

Hagen, den 10.08.2007 im Auftrag

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
   - 1 Deckblatt,
   - 5 Aufgaben auf Seite 1 bis Seite 10.

   Geben Sie diese Unterlagen zusammen mit Ihren Lösungen später bitte vollständig ab, *einschließlich* Aufgabenstellung.

2. Füllen Sie jetzt bitte zuerst das Deckblatt aus:
   - Name, Vorname und Adresse,
   - Matrikelnummer, Geburtsdatum und Klausurort.

3. Schreiben Sie Ihre Lösungen mit Kugelschreiber oder Füllfederhalter (*kein Bleistift*) auf eigenes Papier. Kreuzen Sie die bearbeiteten Aufgaben auf dem Deckblatt an. Schreiben Sie unbedingt *auf jedes Blatt* Ihrer Lösungen die Aufgabennummer, Ihren Namen und Ihre Matrikelnummer.

4. Es sind *keine Hilfsmittel* zugelassen.

5. Lesen Sie vor der Bearbeitung einer Aufgabe den *gesamten* Aufgabentext sorgfältig durch.

6. Achten Sie darauf, dass Sie bei Programmieraufgaben Ihre Lösungen sinnvoll kommentieren; es könnten Ihnen sonst Punkte abgezogen werden.

7. Es sind maximal 100 Punkte erreichbar. Wenn Sie mindestens 40 Punkte erreichen, haben Sie die Klausur mit Sicherheit bestanden.

8. Sie erhalten die korrigierte Klausur zurück zusammen mit einer Bescheinigung für das Finanzamt und ggf. dem Übungsschein.

9. Legen Sie jetzt noch Ihren Studentenausweis und einen amtlichen Lichtbildausweis bereit, dann kann die Arbeit beginnen. Viel Erfolg!

Aufgabe 1: Kurz gefasst (25 Punkte)

a) Zeichnen Sie das Objektgeflecht, das am Ende der `main`-Methode entstanden ist. (5 Punkte)

```
public class TestFeldInitialisierung {
  public static void main(String[] args) {
    char[] vor;
    char[] Fliegen = {'F','l','i','e','g','e','n'};

    vor = new char[3];
    vor[0] = 'v';
    vor[1] = 'o';
    vor[2] = 'r';

    char[][] satz = { Fliegen,
                      {'f','l','i','e','g','e','n'},
                      vor,
                      Fliegen };

  }
}
```

b) Im Kurs haben Sie zwei Möglichkeiten kennengelernt, mit Hilfe von for-Schleifen über Felder zu iterieren. Wenden Sie diese beiden Varianten an, um den Inhalt der Variablen `satz` aus Aufgabenteil a) auf der Standardausgabe auszugeben. (4 Punkte)

c) Realisieren Sie einen Aufzählungstyp für Farben mit Hilfe eines Java-Interfaces und des, ab der Java-Version 5.0 verfügbaren, Programmierkonstrukts für Aufzählungstypen. Welchen Vorteil bietet eine Realisierung mit Hilfe der neuen Aufzählungstypen gegenüber der Realisierung mit Hilfe von Interfaces ? (6 Punkte)

d) Bestimmen Sie das Ergebnis des im folgenden Programmcode durch (*) gekennzeichneten Ausdrucks und begründen Sie Ihre Antwort. (3 Punkte)

```
public class TestVonAusdruecken {
  public static void main(String[] args) {
    int x = 26;

(*) int i = 10 + (i = x) / 7 + 3 * i - 13;
  }
}
```

e) In dieser Teilaufgabe soll innerhalb der main-Methode der Klasse `ObjektErzeugung` ein Objekt vom Typ `B` erzeugt werden (`B b = new B(ia1, ia2);`) . Kann die Objekterzeugung erfolgreich durchgeführt werden? Wenn ja, geben Sie den Inhalt aller Attribute des von b referenzierten Objekts an. Wenn nein, erläutern Sie, an welcher Stelle ein Problem auftritt und warum es auftritt. (7 Punkte)

```
class A {
  private int [] ia;
  private int numberElements;

  public A (int[] ia) {
    this.ia = ia;
    numberElements = numberOfElements ();
  }

  public int numberOfElements () {
    return ia.length;
  }

  /* ...*/
}

class B extends A {
  private int[] ia2;
  private int noElements;

  public B (int[] ia, int[] ia2) {
    super(ia);
    this.ia2 = ia2;
    noElements = numberOfElements ();
  }

  public int numberOfElements () {
    return ia2.length;
  }
}

class ObjektErzeugung {
  public static void main(String[] args) {
    int[] ia1 = {1, 2, 3, 4};
    int[] ia2 = {2, 4, 6, 8, 10, 12};
    B b = new B(ia1, ia2);
  }
}
```

Aufgabe 2: Polymorphie (30 Punkte)

In dieser Aufgabe geht es um die verschiedenen Arten von Polymorphie, die Sie im Kurs kennengelernt haben.

a) Erläutern Sie zunächst allgemein den Begriff der *Polymorphie*. Was versteht man darunter im Kontext von Programmiersprachen? (2 Punkte)

b) Nennen Sie die vier im Kurs genannten Arten von Polymorphie und erklären Sie deren Bedeutung. (12 Punkte)

c) Geben Sie je ein in Java geschriebenes Beispiel an für jede der vier Arten von Polymorphie. Ihr Beispiel soll die im Folgenden angegebene Klasse `Stack` dazu geeignet modifizieren. Das Beispiel soll, wo nötig, weiterhin zusätzlichen Programmcode enthalten, der dazu dient, für die gerade diskutierte Art der Polymorphie zu verdeutlichen, wo Polymorphie zum Tragen kommt. (16 Punkte)

```
import java.util.*;
class Stack {
  ArrayList stack = new ArrayList();

  public boolean isEmpty() {
    return stack.isEmpty();
  }

  public void push (Object elem) {
    stack.add(elem);
  }

  public Object pop () {
    if (!stack.isEmpty())
      return stack.remove(stack.size()-1);
    else
      return null;
  }

  public Object top () {
    if (!stack.isEmpty())
      return stack.get(stack.size()-1);
    else
      return null;
  }
}
```

Erläuterung: Die Klasse `Stack` ist ein Container für Elemente vom Typ `Object` mit der folgenden Charakteristik: Die `push`-Methode legt das als Parameter übergebene Objekt oben auf dem Stack ab, die `pop`-Methode nimmt das oberste Element vom Stack und gibt es zurück. D.h. das Element, das zuletzt auf dem Stack abgelegt wurden, wird mittels `pop` als erstes wieder vom Stack herunter genommen. Die `top`-Methode der Klasse Stack liefert das oberste Stackelement zurück, lässt es aber, im Gegensatz zu `pop`, auf dem Stack. Die Methode `isEmpty` prüft, ob der Stack leer ist.

Aufgabe 3: Verhaltenskonformität (10 Punkte)

Im Kurs wurde neben den syntaktischen Regeln, die ein Subtyp einhalten muss, auch von konformem Verhalten eines Subtyps gesprochen.

a) Erläutern Sie, was man unter *Verhaltenskonformität* im Kontext von Subtyping versteht. (3 Punkte)

b) Geben Sie ein Beispiel für einen Subtyp an, der sich nicht konform zu seinem Supertyp verhält. Ihr Beispiel soll lauffähigen Java-Code enthalten, der den Sachverhalt demonstriert sowie eine Erklärung, warum und wo im Beispiel die Verhaltenskonformität verletzt wird. (7 Punkte)

   Hinweis: Sie dürfen auch auf dem Stack-Beispiel aus Aufgabe 2 aufbauen.

Aufgabe 4: AWT und Ereignissteuerung (20 Punkte)

a) Nennen Sie drei der im Kurs genannten Ereignissorten und erläutern Sie deren Bedeutung. (6 Punkte)

b) Erläutern Sie, wie eine AWT-Komponente bekannt gibt, Ereignisse welcher Ereignissorten an ihr auftreten können, und geben Sie ein Beispiel dafür an. (2 Punkte)

c) Welche Bedingungen müssen Klassen erfüllen, deren Instanzen als Beobachter von Ereignissen einer bestimmten Ereignissorte *ESEvent* fungieren sollen? Geben Sie ein passendes Beispiel an. (2 Punkte)

d) Was passiert, wenn an einer Komponente ein Ereignis auftritt? Beschreiben Sie die durch das Ereignis ausgelösten Aktivitäten und erläutern Sie Ihre Ausführungen an einem Beispiel. (2 Punkte)

e) In dieser Teilaufgabe sollen Sie die unten angegebene Implementierung einer Klasse vervollständigen, die einen Zähler darstellen soll, der durch Drücken entsprechender Schaltflächen erhöht bzw. erniedrigt werden kann. (8 Punkte)

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

class CountFrame extends Frame {
    private int counter = 0;
(1) private Panel p = new Panel(...);
    private TextField tCounter = new TextField(5);
    private Button bInc = new Button(">");
    private Button bDec = new Button("<");

    public CountFrame() {
        tCounter.setText("" + counter);

(2)     // Komponenten in Fenster einfügen


(3)     // Beobachter registrieren


        this.setSize(100, 100);
        this.setLocation(100, 100);
        this.setVisible(true);
    }
}
```

1. Fügen Sie die Komponenten `p, tCounter, bInc` und `bDec` so in das Fenster ein (siehe (2)), dass das Fenster die in Abbildung 1 gezeigte Darstellung auf dem Bildschirm hat. Dabei sollen Textfenster und Button-Objekte im `Panel`-Objekt platziert werden. Wählen Sie einen geeigneten Layoutmanager für die Behälterkomponente `p` aus (siehe (1)), der die Darstellung aus Abbildung 1 gewährleistet.

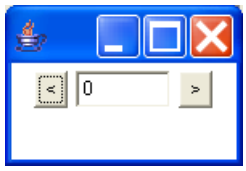

Abbildung 1: Benutzungsoberfläche von `CountFrame`

2. Registrieren Sie Beobachter-Objekte in geeigneter Weise, sodass der Zählerstand per Mausklick ereignis-gesteuert um eins erhöht bzw. erniedrigt werden kann (siehe (3)).
3. Das Programm soll beendet werden, wenn der Benutzer die Anforderung zum Schließen des Fensters gibt (unter Windows durch Anklicken der rechten oberen Schaltfläche ☒ des Fensters).

Aufgabe 5: Kooperierende Threads (15 Punkte)

In dieser Aufgabe simulieren wir einen Verpackungsprozess, der aus drei anderen Prozessen besteht:

- einem Kartonzufuhrprozess,
- einem Produktablageprozess und
- einem Prozess zum Abtransport der befüllten Kartons.

Immer dann, wenn kein Karton mehr an der Befüllposition steht, soll einer dorthin gefahren werden, anschließend mit einem Produkt befüllt und danach weggefahren werden. Im Anschluss daran beginnt der ganze Zyklus von vorn. Die drei Prozesse synchronisieren sich über 3 Signale, von denen eins (`ablageposFrei`) signalisiert, dass ein neuer Karton zur (Ablage-/)Befüllposition gefahren werden kann, ein zweites, das signalisiert, ob der Karton befüllt werden kann (`produktablageErlaubt`) und ein drittes, das anzeigt, ob der Karton mit einem Produkt befüllt wurde und damit weggefahren werden kann (`produktAbgelegt`). Nach dem Wegfahren des Kartons ist die Ablageposition wieder frei. Die benötigten Signale werden durch Attribute, die in einer gemeinsamen Klasse `Signale` deklariert sind, simuliert. Der Programmrahmen sieht wie folgt aus:

```
class Verpackungsprozess {
  public static void main(String[] args) {
    Signale signale = new Signale();

    /* Die Prozesse 'Kartonzufuhrprozess', */
    /* 'Produktablageprozess' und          */
    /* 'Kartonabtransportprozess' starten. */
  }
}

class Signale {
  private boolean ablageposFrei = true;
  private boolean produktablageErlaubt = false;
  private boolean produktAbgelegt = false;

  // Darauf warten, dass Signal auf 'true' gesetzt wird
  public synchronized void WarteAblageposFrei() throws InterruptedException {
    /* ... */
  }
  public synchronized void WarteProduktablageErlaubt() throws InterruptedException {
    /* ... */
  }
  public synchronized void WarteProduktAbgelegt() throws InterruptedException {
    /* ... */
  }

  // Signalzustand auf 'value' setzen
  public synchronized void setAblageposFrei(boolean value) {
    /* ... */
  }
```

```
  public synchronized void setProduktablageErlaubt(boolean value) {
    /* ... */
  }
  public synchronized void setProduktAbgelegt(boolean value) {
    /* ... */
  }
}

class Kartonzufuhrprozess extends Thread {
  private Signale signale;

  public Kartonzufuhrprozess(Signale signale) {
    this.signale = signale;
  }

  private void fahreKartonVor() throws InterruptedException {
    sleep(1000); // Verfahrzeit bis zur Ablageposition
  }

  private void fahreKartonZurAblageposition() throws InterruptedException {

    signale.WarteAblageposFrei();
    System.out.println("-1- Karton wird vorgefahren");
    fahreKartonVor();
    /* Relevante Signale (zurueck) setzen */
  }

  public void run() {
    while (true) {
      try {
        fahreKartonZurAblageposition();
      }
      catch (InterruptedException e) {
        System.out.println("Unzulaessige Unterbrechung");
        System.exit(0);
      }
    }
  }
}


class Produktablageprozess extends Thread {
  private Signale signale;

  public Produktablageprozess(Signale signale) {
    this.signale = signale;
  }

  private void legeProduktAb() throws InterruptedException {
    /* Produktablage simulieren analog zum Vorfahren des Kartons */
  }
```

```
  private void legeProduktAbFallsErlaubt() throws InterruptedException {

    /* Auf Signal zum Ablegen warten */
    System.out.println("-2- Produkt wird abgelegt");
    legeProduktAb();
    /* Relevante Signale (zurueck) setzen */
  }

  public void run() {
    while (true) {
      try {
        legeProduktAbFallsErlaubt();
      }
      catch (InterruptedException e) {
        System.out.println("Unzulaessige Unterbrechung");
        System.exit(0);
      }
    }
  }
}

class Kartonabtransportprozess extends Thread {
  private Signale signale;

  public Kartonabtransportprozess(Signale signale) {
    this.signale = signale;
  }

  private void fahreKartonWeg() throws InterruptedException {
    /* Wegfahren des Kartons simulieren analog zum Vorfahren des Kartons */
  }

  private void fahreKartonVonAblageposWeg() throws InterruptedException {

    /* Auf Signal warten, das signalisiert, dass Produkt     */
    /* abgelegt und damit der Karton weggefahren werden kann */
    System.out.println("-3- Karton wird weggefahren");
    fahreKartonWeg();
    /* Relevante Signale (zurueck) setzen */
  }

  public void run() {
    while (true) {
      try {
        fahreKartonVonAblageposWeg();
      }
      catch (InterruptedException e) {
        System.out.println("Unzulaessige Unterbrechung");
        System.exit(0);
      }
    }
  }
}
```

Aufgabe: Ergänzen Sie den durch Kommentare angedeuteten Programmtext so, dass der Ablageprozess wie oben beschrieben abläuft. Setzen Sie nach Erledigung einer Teilaufgabe, wie z.B. `fahreKartonVor();`, alle notwendigen Signale zurück und setzen Sie das Signal auf `true`, das die darauf folgende Teilaufgabe triggert. Verwenden Sie zur korrekten Koordinierung über die Signale Java's wait/notify-Mechanismus.