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 22.09.2007

Dauer: 3 Std., 10 – 13 Uhr

Lesen Sie zuerst die Hinweise auf der Rückseite!

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

Klausurort: .....

Aufgabe	1			2			3	4		5	6		7		8			9		10		
Teilaufgabe	a	b	c	a	b	c	-	a	b	-	a	b	a	b	a	b	c	a	b	a	b	c
habe bearbeitet																						
maximal	2	3	3	4	5	2	8	4	4	10	4	6	6	2	4	5	7	6	5	2	3	5
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 nächste Klausur findet am Sommersemester 2008 statt.

Hagen, den 02.10.2007 im Auftrag

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
   - 1 Deckblatt,
   - 10 Aufgaben auf Seite 1 bis Seite 13.

   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: Nette Kleinigkeiten **(8 Punkte)**

a) Bestimmen Sie das Ergebnis des im folgenden Programmcode durch ** gekennzeichneten Ausdrucks und begründen Sie Ihre Antwort. **(2 Punkte)**

```
public class TestVonAusdruecken {
  public static void main(String[] args) {
    int x = 256;
    ** long y = (x - 255.59) * 15 < 0 ? 12000L : x - 12256L;
  }
}
```

b) Sind die markierten Zuweisungen von Werten an die Attribute und Variablen ax, v2, ay korrekt? Begründen Sie Ihre Antwort. **(3 Punkte)**

```
class C {
  final int ax, ay = 9;
  C(){
    ax = 7;  // korrekt?
    m();
  }
  void m(){
    final int v1 = 4848, v2 = -3;
    v2 = 0;  // korrekt?
    ay = 34; // korrekt?
    ...
  }
}
```

c) Sind die folgenden Zuweisungen zulässig? Geben Sie eine kurze Begründung Ihrer Antwort an. **(3 Punkte)**
(`Person` ist in dem Beispiel ein Klassentyp, `Student` ist Subtyp von `Person`.)

```
Object   ovar = new String[3];
Person[]  pfv = new Student[2];
Student[] sfv = new Person[2];
```

Aufgabe 2: Gemischtes (11 Punkte)

a) Ist der folgende Programmausschnitt korrekt?
Begründen Sie Ihre Antwort. **(4 Punkte)**

```
(1)  String[] strfeld = new String[2];
(2)  Object[] objfeld = strfeld;
(3)  objfeld[0] =  new Object();

(4)  int strl   =  strfeld[0].length();
```

b) Gegeben ist der Schnittstellentyp `Lastwagen` mit den direkten Supertypen `Fahrbar` und `Fahrzeug` und den Methoden `fahren`, `getName`, `getFahrgestellnummer`, `isOldtimer`, `hatMautConsole`.
Geben Sie die Deklaration aller drei Schnittstellen an und ordnen Sie die Methoden sinnvoll den einzelnen Schnittstellen zu. Überlegen Sie sich für alle Methoden eine sinnvolle Signatur. **(5 Punkte)**

c) Definieren Sie eine Klasse `LinkedList`, die den Parametertyp `ET` hat, der durch die Schnittstellen `Druckbar` und `Konvertierbar` nach oben beschränkt ist. **(2 Punkte)**

Aufgabe 3: Aufzählungstypen (8 Punkte)

Gegeben ist die folgende Klasse Wochentag.

```
public class Wochentag {
  private int tag;

  public Wochentag() {
    tag = 0;
  }
  public void setTag(int i) throws KeinTagException {
  if (i < 0 || i > 6)
    throw new KeinTagException();
  else
     tag = i;
  }
  public int getTag() {
    return tag;
  }
  public void naechsterTag() {
    tag = (tag + 1) % 7;
  }
  public void vorhergehenderTag() {
    ...
  }
  public String toString() {
    switch(tag) {
      case 0: return "Montag";
      case 1: return "Dienstag";
      case 2: return "Mittwoch";
      case 3: return "Donnerstag";
      case 4: return "Freitag";
      case 5: return "Samstag";
      case 6: return "Sonntag";
      default: return "ERROR";
    }
  }
}

class KeinTagException extends Exception {
}
```

Implementieren Sie die Klasse `Wochentag` jetzt als Aufzählungstyp. Dabei soll das private Attribut `tag` seinen Namen in `aktuellerTag` ändern und seinen Typ in `Wochentag`. Die Signaturen der Methoden `setTag` und `getTag` sollen wie folgt abgeändert werden:

```
public void setTag(Wochentag tag) { ... }
public Wochentag getTag() { ... }
```

Beantworten Sie auch folgende Fragen:

1. Warum braucht die Methode `setTag` jetzt keine Ausnahme mehr zu werfen?
2. Was kann man zur Implementierung der Methode `toString` sagen?

Aufgabe 4: Autoboxing (8 Punkte)

a) Sind die Basisdatentypen Subtypen von `Object`? **(1 Punkte)**
Was versteht man unter Autoboxing? **(3 Punkte)**

b) Ist der folgende Programmausschnitt korrekt?
Begründen Sie Ihre Antwort. **(4 Punkte)**

```
      // Version ab Java 5.0
      List<Integer> ints = new ArrayList<Integer>();
(*)   ints.add(1);
(**)  int n = ints.get(0);
```

Aufgabe 5: Beobachter (10 Punkte)

Ergänzen Sie im folgenden Programmausschnitt in der Klasse `Fluss` die fehlenden Programmteile.

```
/*******************************************************************************/
/********                  Klasse Wasserbueffelbesitzer                 *********/
/*******************************************************************************/
/** Wenn der Wasserstand zu niedrig ist, kleinergleich 0,3 m, brauchen die Bueffel   **/
/** zusaetzliches Wasser.                                                            **/
/** Sobald die Wiesen ueberflutet werden, sollten die Bueffel in Sicherheit gebracht **/
/** werden, also bei einem Wasserstand zwischen 5 und 7 m.                           **/
/** Ab 7 m muessen die Bueffel schwimmen, was sie aber nur eine Weile aushalten.     **/
/** Daher sollten sie dann dringend aus dem Fluss gerettet werden.                   **/
/*******************************************************************************/
class Wasserbueffelbesitzer implements Beobachter{

  public void fallen(double wasserhoehe){
    if (wasserhoehe <= 0.3) {
      System.out.println("Wasserstand = " + wasserhoehe);
      System.out.println(" Achtung Bueffelbesitzer: Zu wenig Wasser im Fluss");
      System.out.println(" Bueffel zusaetzlich traenken!");
      System.out.println("");
    }
  }

  public void steigen(double wasserhoehe) {
    if ((wasserhoehe >= 5) && (wasserhoehe < 7)){
      System.out.println("Wasserstand = " + wasserhoehe);
      System.out.println(" Achtung Bueffelbesitzer: Wiesen ueberschwemmt");
      System.out.println(" Bueffel in den Stall bringen");
      System.out.println("");
    }
    else {
      if (wasserhoehe >=7){
        System.out.println("Wasserstand = " + wasserhoehe);
        System.out.println("Achtung: Bueffelbesitzer: Hochwasser");
        System.out.println("Bueffel aus Wasser retten!");
        System.out.println("");
      }
    }
  }
}



interface Beobachter {
  void steigen(double wasserhoehe);
  void fallen(double wasserhoehe);
}
```

```
/*****************************************************************************/
/**                        Klasse Fluss                                     **/
/*****************************************************************************/
/** Fuer diese Aufgabe benoetigen wir als reine Flussinformationen nur die  **/
/** Wasserhoehe. Auf sie greifen wir mit der Methode 'getHoehe' zu.         **/
/** Die Beobachter muessen verwaltet werden. Dazu werden sie in einer Liste namens **/
/** beobachterListe abgelegt. Dies geschieht mit der Methode 'anmelden-Beobachter'. **/
/** In der Methode 'setHoehe' werden die Beobachtungsmethoden 'steigen' und 'fallen'**/
/** fuer die jeweiligen Beobachter passend zum Flussverhalten aufgerufen.   **/
/*****************************************************************************/
import java.util.*;

public class Fluss{
  private double wasserhoehe; /** Wasserstand **/
  /*************************************************************/
  /**** Bitte ergaenzen!                                    ****/
  /**** Legen Sie eine Beobachter-Liste an;                 ****/
  /**** verwenden Sie dabei das Konzept der generischen Typen. ****/
  /*************************************************************/

  /** Fluss-Konstruktor **/
  Fluss(double h){
    wasserhoehe = h;
    /*************************************************************/
    /**** Bitte ergaenzen!                                    ****/
    /**** Erzeugen Sie eine Beobachter-Liste.                 ****/
    /*************************************************************/
  }

  /*************************************************************/
  /**** Bitte ergaenzen!                                    ****/
  /**** Definieren Sie eine Methode 'anmeldenBeobachter',   ****/
  /**** die Beobachter zur Beobachterliste hinzufügt.       ****/
  /*************************************************************/
  /** Ermittlung des Wasserstandes **/
  public double getHoehe(){
    return wasserhoehe;
  }

  /** Setzen des neuen Wasserstandes                **/
  /** Vergleich zwischen altem und neuem Wasserstand **/
  void setHoehe( double neueHoehe ){
    double alteHoehe = wasserhoehe;
    wasserhoehe = neueHoehe;
    /*****************************************************************/
    /**** Bitte ergaenzen!                                        ****/
    /**** Fuehren Sie eine Iteration ueber die Liste der Beobachter ****/
    /**** mittels der for-each Schleife durch, wobei Sie abhaengig  ****/
    /**** von der Wasserhoehe die Methoden 'steigen' und 'fallen' fuer ****/
    /**** jeden Beobachter aufrufen                               ****/
    /*****************************************************************/
  }
}
```

Aufgabe 6: Auflösen überladener Methodenaufrufe (10 Punkte)

a) Betrachten Sie folgenden Programmcode:

```
public class Objektgeflecht {
  Objektgeflecht a, b, c;

  public Objektgeflecht () {
    a = null;
    b = null;
    c = null;
  }

  public Objektgeflecht (Objektgeflecht a,
                         Objektgeflecht b,
                         Objektgeflecht c) {
    this.a = a;
    this.b = b;
    this.c = c;
  }

  public static void main (String argv[]) {
    Objektgeflecht u = new Objektgeflecht();
    Objektgeflecht v = new Objektgeflecht();
    Objektgeflecht w = new Objektgeflecht(u, v, null);

    (w.a).b = v;
    v.a = u.b;
    (u.b).c = w;
    w.c = v.c;
    u.c = (v.a).c;
    /* Markierung */
  }
}
```

Bei welchen Deklarationselementen tritt Überladung auf und bei welchen Anweisungen muss der Compiler eine Überladung auflösen?
Warum gelingt die Auflösung? **(4 Punkte)**

b) Betrachten Sie folgenden Programmcode:

```
class Haustier { ... }

class Hund extends Haustier { ... }
class Wachhund extends Hund { ... }

class Katze extends Haustier { ... }
class Leopard extends Katze { ... }

class TrautesHeim {
  void sichmoegen (Haustier a, Katze d) { ... }     // Deklaration #1
```

```
  void sichmoegen (Hund b, Haustier a) { ... }      // Deklaration #2
  void sichmoegen (Wachhund c, Katze d) { ... }     // Deklaration #3

  void SympathieTest() {
    Haustier a = new Haustier();
    Hund b = new Hund();
    Wachhund c = new Wachhund();
    Katze d = new Katze();
    Leopard e = new Leopard();

    sichmoegen(a,d);    // Aufruf-1
    sichmoegen(c,a);    // Aufruf-2
    sichmoegen(c,e);    // Aufruf-3
    sichmoegen(b,d);    // Aufruf-4
  }
}
```

Können die mit Aufruf-1 bis Aufruf-4 gekennzeichneten Methodenaufrufe erfolgreich aufgelöst werden? Wenn ja, geben Sie die jeweils zum Aufruf passende Deklaration an. Geben Sie für jeden Aufruf die Schritte an, die zur Auflösung des Aufrufs durchgeführt werden. **(6 Punkte)**

Aufgabe 7: Ausnahmebehandlung (8 Punkte)

Gegeben ist das folgende Programm:

```
class Klausurortfehler extends Exception { }

class Matrikelnummerfehler extends Exception { }

class Nachnamenfehler extends Exception { }

class Vornamenfehler extends Exception { }

public class TryKlausuranmeldung {
  public static void main(String[] argf){
   String Klausurort, Nachname, Vorname;
   String KlausurortAuswahl = "Hamburg";
   int i, LaengeNachname, LaengeVorname;
   int Matrikelnummer;
   char x;

   try{
       Klausurort = argf[0];
       System.out.println("Klausurort = " + Klausurort);
       if (!Klausurort.equals(KlausurortAuswahl)) throw new Klausurortfehler();

       System.out.println("Matrikelnummer = " + argf[1]);
       if (argf[1].length() != 8) throw new Matrikelnummerfehler();
       Matrikelnummer = Integer.parseInt(argf[1]);

       Nachname = argf[2];
       LaengeNachname = Nachname.length();
       System.out.println("Nachname = " + Nachname);
       for (i=0; i<LaengeNachname; i=i+1) {
            x = Nachname.charAt(i);
            if ( (int)x < 65 || (int)x > 122  || ((int)x > 90 & (int)x < 97))
            throw new Nachnamenfehler();
       }

       Vorname = argf[3];
       LaengeVorname = Vorname.length();
       System.out.println("Vorname = "  + Vorname);
       for (i=0; i<LaengeVorname; i=i+1) {
            x = Vorname.charAt(i);
            if ( (int)x < 65 || (int)x > 122 || ((int)x > 90 & (int)x < 97))
             throw new Vornamenfehler();
       }
       System.out.println();
       System.out.println("Eingabe fehlerfrei");
      }
      catch (IndexOutOfBoundsException e){
            System.out.println("Eingabe nicht vollstaendig");
      }
      catch (NumberFormatException e) {
```

```
            System.out.println("Eingabe der Matrikelnummer ist keine Integer-Zahl");
      }
      catch (Klausurortfehler e) {
           System.out.println("Klausurort ist nicht Hamburg");
      }
      catch (Matrikelnummerfehler e) {
             System.out.println("Die Matrikelnummer besteht nicht aus 8 Zeichen");
      }
      catch (Nachnamenfehler e) {
             System.out.println("Nachname enthaelt nicht nur Buchstaben");
      }
      catch (Vornamenfehler e) {
             System.out.println("Vorname enthaelt nicht nur Buchstaben");
      }
    finally {System.out.println("Anmelden ist immer schwer");
    }
 }
}
```

a) Beschreiben Sie mit zwei bis zu vier Sätzen, was dieses Programm macht. **(6 Punkte)**

b) Welches Ergebnis liefert das Programm bei der folgenden Eingabe? **(2 Punkte)**

```
java TryKlausuranmeldung Hamburger 111 Wolf 110
```

Aufgabe 8: Ereignissteuerung **(16 Punkte)**

a) Woraus besteht das abstrakte GUI-Modell des AWT?
Beschreiben Sie kurz seine drei Teile. **(4 Punkte)**

b) Was passiert, wenn an einer Komponente ein Ereignis auftritt, und wie kann eine GUI auf ein Ereignis reagieren? **(5 Punkte)**

c) Implementieren Sie ein Hauptfenster, in dem eine Schaltfläche platziert ist, die beim Anklicken das Hauptfenster schließt.
Verwenden Sie bitte den Layout-Manager `FlowLayout`. **(7 Punkte)**

Aufgabe 9: Threads (11 Punkte)

a) Die Aktionen, die ein Thread ausführen soll, werden in einer parameterlosen Methode mit Namen `run` beschrieben. Es gibt in Java zwei Möglichkeiten, diese run-Methode zu deklarieren. Geben Sie die beiden zugehörigen Muster an. **(6 Punkte)**

b) Ergänzen Sie die Klasse `Tausche` um Codestücke, so dass wechselweiser Ausschluss der beiden Vertauschungsvorgänge gewährleistet ist und keine Verklemmungen auftreten können.
Verändern Sie dabei die Klassen `Wert` und `Main` nicht.
Begründen Sie, warum Ihre Lösung das Gewünschte leistet. **(5 Punkte)**

```
class Wert {
  int wert;

  Wert(int wert) {
    this.wert = wert;
  }
}


class Tausche extends Thread {
  Wert a;
  Wert b;

  Tausche(Wert a, Wert b) {
    this.a = a;
    this.b = b;
  }

  public void run() {
    int h = a.wert;
    a.wert = b.wert;
    b.wert = h;
  }
}

class Main {
  public static void main(String[] argv) {
    Wert x = new Wert(0);
    Wert y = new Wert(1);
    Tausche tom = new Tausche(x,y);
    Tausche jerry = new Tausche(y,x);
    tom.start();
    jerry.start();
  }
}
```

Aufgabe 10: Programmierung verteilter Objekte (10 Punkte)

a) Was wird im Kurs 1618 unter einem *verteilten System* verstanden? **(2 Punkte)**

b) Sockets bilden eine Abstraktionsschicht zur Kommunikation zwischen verteilten Prozessen, die den Programmierer von einigen nicht trivialen Tätigkeiten befreit. Welche Tätigkeiten sind das? **(3 Punkte)**

c) Wie sieht das grundlegende Muster zur Realisierung eines einfachen Servers aus? Geben Sie das passende Programmfragment an oder beschreiben Sie das Muster textuell. **(5 Punkte)**