FernUniversität in Hagen

FAKULTÄT FÜR
MATHEMATIK UND INFORMATIK

Lehrgebiet Programmiersysteme
Prof. Dr. Friedrich Steimann

FernUniversität in Hagen • 58084 Hagen

_____
(Vorname, Name)

_____
(Straße, Nr.)

_____
(PLZ, Ort)

_____
(Land, falls außerhalb von Deutschland)

Kurs:1618 SS 2008

„Einführung in die objektorientierte Programmierung“

Klausur am 07.02.2009

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	Summe
habe bearbeitet											
maximal	12	12	12	12	14	6	16	9	10	10	113
erreicht											
Korrektur											

O Herzlichen Glückwunsch, Sie haben die Klausur bestanden. Note: ……

O Sie haben die Klausur leider nicht bestanden. Für den nächsten Versuch wünschen wir Ihnen viel Erfolg.

Hagen, den

Im Auftrag

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
   – 1 Deckblatt,
   – 10 Aufgaben auf Seite 1 bis Seite 15.

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*) direkt in den bei den jeweiligen Aufgaben gegebenen Leerraum. Benutzen Sie auf keinen Fall die Rückseiten der Aufgabenblätter. Versuchen Sie, mit dem vorhandenen Platz auszukommen, sie dürfen auch stichwortartig antworten. Für den Notfall können Sie gekennzeichnetes Papier bei der Klausuraufsicht bekommen. **Es werden nur Aufgaben gewertet, die sich auf dem offiziellen Klausurpapier (Klausur + gekennzeichnetes Papier) befinden.** Eigenes Papier ist nur für Ihre persönlichen Notizen erlaubt und von der Benotung ausgeschlossen.

4. Kreuzen Sie die bearbeiteten Aufgaben auf dem Deckblatt an. Schreiben Sie unbedingt *auf jedes Blatt* Ihrer Lösungen Ihren Namen und Ihre Matrikelnummer, auf eventuelle Zusatzblätter auch die Nummer der Aufgabe.

5. Geben Sie die gesamte Klausur und, falls vorhanden, gekennzeichnete Zusatzblätter ab.

6. Es sind *keine Hilfsmittel* zugelassen.

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

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

9. Es sind maximal 113 Punkte erreichbar. Wenn Sie mindestens 46 Punkte erreichen, haben Sie die Klausur mit Sicherheit bestanden.

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

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

Aufgabe 1: Ausnahmebehandlung (12 Punkte)

Es soll eine Ausnahmebehandlung für Überläufe bei int-Additionen entwickeln werden (in Java werden solche Überläufe normalerweise nicht behandelt).

a) Welche Art von Anweisung verwendet man sinnvollerweise hierfür?

`Antwort:`

b) Die Klasse Ueberlauf sollte von einer Klasse der Java-Bibliothek erben. Welche Klasse ist das sinnvollerweise und wie implementiert man diese Beziehung?

`Antwort:`

c) Ergänzen Sie das folgende Programmfragment, das zwei Integervariablen addiert, die in Form von Strings in argf vorliegen.
Um einen Überlauf bei der Addition erkennen zu können, müssen diese Integervariablen vor der Addition zunächst in long-Werte konvertiert werden.
Die Variable `maxint` soll zum Test auf einen gegebenenfalls zu großen Integerwert dienen.
Die Variable `ergebnis` soll das Ergebnis der Addition aufnehmen, wenn es nicht größer als `maxint` ist.
Erweitern Sie das unten stehende Programmfragment so, dass bei Auftreten einer `IndexOutOfBoundsException` und einer `NumberFormatException` eine entsprechende Fehlermeldung ausgegeben wird. Auch Ihre Lösung aus Aufgabe 1b) sollte berücksichtigt werden.

```
String[] argf = {"2147483640", "3450336"};
long maxint = 2147483647L;


  int m, n, ergebnis ;
  m = Integer.parseInt( argf[0] );
  n = Integer.parseInt( argf[1] );
  long aux = (long)m + (long)n;
```

Aufgabe 2: Sichtbarkeit (12 Punkte)

Notieren Sie in dem folgenden Programmcode für alle durch eine vorangestellte Nummer gekennzeichneten Anweisungen, ob der Zugriff auf das verwendete Attribut bzw. die verwendete Methode bzw. die verwendete Klasse erlaubt ist oder nicht. Begründen Sie jeweils Ihre Antwort kurz und in Stichpunkten.

```
package a;

public class A1 {
  private int i = 0;
  protected int j = 1;

  private void am1()  {
    System.out.println("Dies ist eine private Methode" + i);
  }
  protected void  am2() {
    System.out.println(
            "Dies ist eine Methode mit Modifikator 'protected'" + j);
  }

  boolean isEqualTo(A1 a) {
    /*   1)                           */
    if ( i == a.i ) return true;
    /* Zugriff erlaubt? Ja oder Nein */
    /* Begründung:                   */




    return false;
  }
  public boolean compareA1toB1 (B1 b) {
    if ( i == b.i ) return true;
    return false;
  }
}

class B1 {
  public int i = 2;
  int j = 3;
  void bm() {
    A1 a = new A1();
    A1 a2 = new A1();
    /*   2)                           */
    a.j = 10;
    /* Zugriff erlaubt? Ja oder Nein  */
    /* Begründung:                   */




    a.isEqualTo(a2);
```

```
      /*   3)                          */
      a.am1();
      /* Zugriff erlaubt? Ja oder Nein  */
      /* Begründung:                    */




    }
  }
/* ---------------------------------------------- */
  package b;
  import a.*;

  class D1 {
    private int i = 9;

    class D2 {
      private int i = 0;
    }

    D2 createD2Element () {
      return new D2();
    }
  }

  public class C1 extends A1 {
    void  cm( A1 a, A1 a2, C1 c, D2 d)  {
      /*   4)                          */
      B1 b = new B1();
      /* Zugriff erlaubt? Ja oder Nein */
      /* Begründung:                    */




      /*   5)                          */
      D1.D2 d2 = d.createD2Element();
      /* Zugriff erlaubt? Ja oder Nein */
      /* Begründung:                    */




      /*   6)                          */
      c.i = 15;
      /* Zugriff erlaubt? Ja oder Nein */
      /* Begründung:                    */




    }
  }
```

Aufgabe 3: Objektgeflechte (12 Punkte)

Gegeben ist die folgende Liste `ll`:

```
LinkedList ll  = new LinkedList();
StringBuffer s = new StringBuffer("Wasser");
StringBuffer t = new StringBuffer("bueffeln");
ll.add("Fleissig "); ll.add("mit "); ll.add(s); ll.add(t);
ll.add(" im "); ll.add(s); ll.add(" "); ll.add(t);
```

a) Welche Text ergibt sich, wenn man nach Ausführung aller `add`-Anweisungen sämtliche Elemente der Liste hintereinander auf der Systemausgabe ausgibt?

`Antwortsatz:`

b) Warum ist die Ausgabe von Objektgeflechten eine nichttriviale Aufgabe?
Nennen Sie zwei Gründe.

`Antwort:`
`Grund 1:`

`Grund 2:`

c) Mit welchen Anweisungen kann man in Java die obige Liste `ll` und alle referenzierten Objekte in eine Datei ausgeben?

`Programmfragment:`

Aufgabe 4: AWT (12 Punkte)

- Was bietet das AWT an, um automatisch die Darstellung eines Behälters B aus den Darstellungen der in B enthaltenen Komponenten zu berechnen?

  `Antwort:`

a) Was ist im Vergleich zu anderen Behältertypen im AWT das Besondere an Fenstern?

   `Antwort:`

b) Wozu dient die Klasse Panel?

   `Antwort:`

c) Was ist das Besondere an einer ScrollPane?

   `Antwort:`

d) Die Entwicklung von GUIs ist im Allgemeinen eine anspruchsvolle Aufgabe und zwar sowohl aus ergonomischer Sicht (wie organisiert man eine GUI so, dass sie möglichst einfach und komfortabel zu bedienen ist) als auch aus softwaretechnischer Sicht. Nennen Sie die zwei wesentlichen Ursachen für die softwaretechnische Komplexität.

   `Antwort:`

   Ursache 1:

   Ursache 2:

Aufgabe 5: Listen und Generics (14 Punkte)

Die Programmierin Ursula S. soll eine Menge geometrischer Figuren wie z.B. Kreise und Rechtecke in einer gemeinsamen Liste verwalten. Die Liste muss später auch andere Figuren wie z.B. Dreiecke oder Rauten aufnehmen können.
Helfen Sie ihr und implementieren Sie je eine Klasse für Kreise und Rechtecke sowie eine Klasse `FigurenListe` zur Verwaltung der Figuren mit den im Folgenden beschriebenen Eigenschaften:

- Die Figurenobjekte sollen jeweils über eine Methode `anzeigen` zur Darstellung der Figur verfügen.
- Die Darstellung einer Figur soll hier nur durch Ausgabe des Textes „Kreis anzeigen“, „Rechteck anzeigen“ etc. auf der Systemausgabe simuliert werden. (Instanzvariablen wie z.B. der Kreismittelpunkt müssen nicht deklariert werden.)
- Die Klasse `FigurenListe` soll ein Attribut `figurenListe` vom Typ `LinkedList<ET>` für einen geeigneten Elementtyp `ET` enthalten. Dieser Elementtyp muss sicherstellen, dass in der Liste nur Figuren mit den angegebenen Eigenschaften gespeichert werden können.
- Die Klasse `FigurenListe` soll die Methoden `figurAnfuegen`, `letzteFigurAuslesen` und `alleAnzeigen` zur Verfügung stellen.
  - Mit Hilfe der Methode `figurAnfuegen` soll ein Figurenobjekt am Ende von `figurenListe` eingefügt werden.
  - Die Methode `letzteFigurAuslesen` soll das letzte gespeicherte Figurenobjekt zurückgeben.
  - Mit Hilfe der Methode `alleAnzeigen` können alle Elemente der Liste nacheinander am Bildschirm anzeigt werden.

Die parametrische Klasse `LinkedList` ist in dem Bibliothekspaket `java.util` enthalten und braucht nicht implementiert zu werden.

Schreiben Sie bitte die Lösung auf das folgende Blatt.

Lösung von Aufgabe 5, Listen und Generics

Aufgabe 6: Autoboxing (6 Punkte)

Ist der folgende Programmausschnitt korrekt?
Begründen Sie Ihre Antwort.

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

Antwort:

Aufgabe 7: Statisches und dynamisches Binden: (16 Punkte)

Gegeben sei das folgende Java-Programm:

```
class A {
  int i = 1;
  int m() { return i * 100; }
  int n() { return i + m(); }
  void k() { System.out.println("" + this); }
}

class B extends A {
  int i = 10;
  int m() { return i * 1000; }
  int test2() { return n(); }
}


class Test {
  int test1() {
    A[] ar = { new A(), new B() };
    return ar[0].i + ar[0].m() + ar[1].i + ar[1].m();
  }

  public static void main(String[] argv) {
    A a = new B();
    System.out.println(a.m());              /* 1 */
    System.out.println(new B().test2());    /* 2 */
  }
}
```

a) Wie viele und welche Attribute hat ein Objekt vom Typ B?

Antwort:

b) In objektorientierten Sprachen muss man zwischen dem statischen und dynamischen Typ eines Ausdrucks unterscheiden. Welchen statischen Typ hat der Ausdruck `ar[1]` in der zweiten Zeile des Methodenrumpfes von `test1` und welchen dynamischen Typ hat er bei Programmausführung?

Antwort:

c) Welcher Wert wird an der mit „1“ gekennzeichneten Stelle ausgegeben?

`Antwort:`

Begründung:

d) Welcher Wert wird an der mit „2“ gekennzeichneten Stelle ausgegeben?

`Antwort:`

Begründung:

Aufgabe 8: Ko- und Kontravarianz (9 Punkte)

a) Ist folgendes Beispiel bezüglich der Parametertypen, der Ausnahmetypen und der Ergebnistypen ein korrektes Java-Programm?

```
class ExceptionT extends Exception{}

class ReturnType extends Exception{}

class ExceptionS extends ExceptionT { }

interface Supertyp {
  ReturnType meth( ParameterType p) throws ExceptionT;
}

class Subtyp implements Supertyp {
  public ReturnType meth( ParameterType p)
                              throws ExceptionS { ... }
}
```

Antwort:

Begründung:

b) Gegeben ist der Typ *Sub* mit der Methode $\texttt{meth}_{sub}$ und der Typ *Super* mit der Methode $\texttt{meth}_{super}$. *Sub* ist Subtyp von *Super* und $\texttt{meth}_{sub}$ soll $\texttt{meth}_{super}$ überschreiben.

- Was muss für jeden Parametertypen von $\texttt{meth}_{sub}$ in Bezug auf den entsprechenden Typ aus $\texttt{meth}_{super}$ gelten?

  Antwort:

- Was muss für den Ergebnistyp von $\texttt{meth}_{sub}$ und von $\texttt{meth}_{super}$ gelten?

  Antwort:

Aufgabe 9: Parallelität (10 Punkte)

Gegeben ist das folgende Programm. Was ist die Hauptaufgabe des Programms und welche vier wesentlichen Aufgaben erfüllt es?

```
import java.io.*;
import java.net.*;
import java.util.*;

class FileServer {
  public static void main(String args[]) throws IOException {
    System.out.println("Starting FileServer...");
    DateiServer myServer = new DateiServer();
  }
}

class DateiServer {
  public DateiServer()  throws IOException {
    ServerSocket serversocket = new ServerSocket (8181);
    while (true) {
      Socket socket = serversocket.accept();
      Thread bearbeiteAnfrage = new BedienerThread(socket);
      bearbeiteAnfrage.start();
    }
  }
}

class BedienerThread extends Thread {
  private Socket mySocket;

  public BedienerThread(Socket so) {
    mySocket = so;
  }

  public void run() {
    BufferedReader fromClient;
    BufferedWriter toClient;
    BufferedReader dateiLeser;

    try {
      // Oeffne den Eingabestrom vom Client
      fromClient = new BufferedReader(new InputStreamReader
                                      (mySocket.getInputStream()));

      // Oeffne den Ausgabestrom zum Client
      toClient = new BufferedWriter(new OutputStreamWriter
                                    (mySocket.getOutputStream()));

      // Oeffnen der uebertragenen Datei
      String liesDatei = fromClient.readLine();
      StringTokenizer strToken = new StringTokenizer
                                      (liesDatei, " \r\n");

      // Clientnamen ermitteln
```

```
      String clientName = strToken.nextToken();
      String dateiName = strToken.nextToken();
      dateiLeser = new BufferedReader(new FileReader (dateiName));
      System.out.println("Bearbeite Anfrage Client " + clientName);

      int c = dateiLeser.read();
      while( c != -1) {
        toClient.write(c);
        c = dateiLeser.read();
      }
      dateiLeser.close();
      toClient.flush();

      System.out.println("Beende Anfrage Client " + clientName);
      mySocket.close();
    } catch (Exception e) {
        System.out.println("Fehler beim Lesen der Datei");
    }
  }
}
```

Tragen Sie hier Ihre Lösung ein:

Hauptaufgabe des Programms:

Das Programm löst die folgenden Aufgaben:

A1:

A2:

A3:

A4:

Aufgabe 10: Beobachter (10 Punkte)

Gegeben ist das folgende Programm:

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

class Addierer extends Frame {
  protected Button b1;
  protected TextField t1,t2,t3;

  public Addierer() {
    setLayout(new FlowLayout());
    setSize(300,100);
    b1 = new Button("addiere");
    t1 = new TextField("4",4);
    t2 = new TextField("2",4);
    t3 = new TextField(4);
    t3.setEditable(false);
    add(t1);
    add(t2);
    add(t3);
    add(b1);
  }
}

class AddiererTest {
  public static void main(String argv[]) {
    Addierer f = new Addierer();
    f.addWindowListener(new WindowAdapter () {
      public void windowClosing (WindowEvent e) {
        System.exit(0);
      }
    });
    f.setVisible(true);
  }
}
```

Erweitern Sie das Programm so, dass beim Drücken des Knopfes `b1` die Inhalte der Textfelder `t1` und `t2` in Integer-Zahlen umgewandelt, addiert und in das Textfeld `t3` geschrieben werden. Achten Sie darauf, dass fehlerhafte Eingaben abgefangen werden.
Implementieren Sie zur Lösung dieser Aufgabe eine geeignete Beobachterklasse. Wählen Sie dazu die folgende Variante:

- Die Beobachterklasse spezialisiert die Klasse `Addierer` so, dass sie als ihr eigener Beobachter für an `b1` auftretende Ereignisse verwendet werden kann.

Ergänzen Sie dazu die folgende Klasse `AddiererSpez`:

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

class AddiererSpez                                                    {

  AddiererSpez () {









  }

  public void actionPerformed (ActionEvent e) {
```

```
   }

class AddiererTest {
  public static void main(String argv[]) {
    Addierer f = new AddiererSpez();
    f.addWindowListener(new WindowAdapter () {
      public void windowClosing (WindowEvent e) {
        System.exit(0);
      }
    });
    f.setVisible(true);
  }
}
```