FernUniversität in Hagen

FAKULTÄT FÜR
MATHEMATIK UND INFORMATIK

Lehrgebiet Programmiersysteme
Prof. Dr. Friedrich Steimann

[redacted]
(Name, Vorname)

[redacted]
(Straße, Nr.)

[redacted]
(PLZ) (Wohnort)

(Land, falls außerhalb Deutschlands)

Kurs 01618
Einführung in die objektorientierte Programmierung

(Kursdurchführung des Sommersemester 2015)

Hauptklausur am 12.09.2015

Dreistündige Klausur
(10.00 – 13.00 Uhr)

Lesen Sie zuerst die Hinweise auf der folgenden Seite!

Matrikelnummer: [redacted] **Geburtsdatum:** [redacted]

Klausurort: [redacted]

Aufgabe	1	2	3	4	5	6	7	8	9	10	Summe
habe bearbeitet	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	
maximal	10	10	10	10	10	10	10	10	10	10	100
erreicht	[redacted]										
Korrektur											

X Herzlichen Glückwunsch, Sie haben die Klausur bestanden. Note: [redacted]

○ Sie haben die Klausur leider nicht bestanden. Für den nächsten Versuch wünschen wir Ihnen viel Erfolg. Die nächste Klausur findet im Wintersemester 2015 / 2016 statt.

Hagen, den 30.09.2015

Im Auftrag

[redacted]

Hinweise zur Bearbeitung

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst auf insgesamt 21 Seiten:
   - 1 Deckblatt
   - Diese Hinweise zur Bearbeitung
   - 10 Aufgaben auf Seite 3-19
   - Zwei zusätzliche Seiten 20 und 21 für weitere Lösungen

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, umrahmten Leerraum. Benutzen Sie auf keinen Fall die Rückseiten der Aufgabenblätter. Versuchen Sie, mit dem vorhandenen Platz auszukommen; Sie dürfen auch stichwortartig antworten. Sollten Sie wider Erwarten nicht mit dem vorgegebenen Platz auskommen, benutzen Sie bitte die beiden dieser Klausur angefügten Leerseiten. **Es werden nur Aufgaben gewertet, die sich auf dem offiziellen Klausurpapier befinden.** Eigenes Papier ist nur für Ihre persönlichen Notizen erlaubt.

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

5. Geben Sie die gesamte Klausur ab. Lösen Sie die Blätter nicht voneinander.

6. Es sind *keine Hilfsmittel* zugelassen.

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

8. Es sind maximal 100 Punkte erreichbar. Wenn Sie mindestens 50 Punkte erreichen, haben Sie die Klausur bestanden.

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

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

Aufgabe 1: Begriffe der Objektorientierung (10 Punkte)

Gegeben sei das folgende Java-Programm:

```
1   public class Begriffe {
2
3       int blume = -1;
4
5       interface Ausdruck {
6           int baum = 42;
7       }
8
9       static void gießen(Integer i) {
10          Number erde = i;
11      }
12
13      static void gießen(Long l) {
14          Number erde = l;
15      }
16
17      void pflanzen() {
18          new Terminus() {
19              void wachsen() {
20                  gießen(blume);
21              }
22
23          }.wachsen();
24      }
25
26      class Terminus implements Ausdruck {
27          void wachsen() {
28              gießen(baum);
29          }
30      }
31
32  }
```

In diesem Programmfragment kommen verschiedene Elemente vor. Zählen Sie (unter Angabe der Zeilennummer) auf:

- die Namen aller aktuellen Parameter: (1 Punkt)

- die Namen aller formalen Parameter: (1 Punkt)

(Fortsetzung der Aufgabe auf der folgenden Seite)

(Fortsetzung von Aufgabe 1)

- die Namen aller inneren Klassen: (1 Punkt)

- alle anonymen Klassen (kürzen Sie das Innere der Klasse mit '...' ab) : (1 Punkt)

- alle dynamisch gebundenen Zugriffe auf Attribute: (1 Punkt)

- alle dynamisch gebundenen Referenzen an Methoden: (1 Punkt)

- alle überschreibenden Methoden sowie die zugehörigen überschriebenen Methoden: (1 Punkt)

- alle überladene Methoden: (1 Punkt)

- alle Stellen, an denen einen Subtyp-Beziehung ausgenutzt wird: (1 Punkt)

- alle Typen, die einen Default-Konstruktor haben: (1 Punkt)

Aufgabe 2: Klassenimplementierung (10 Punkte)

Gegeben sei das folgende Programmfragment:

```
public class Imbissbude {

      public static void main(String[] args) {
            Pizza mista = new Pizza ("Mista");
            mista.istBelegtMit("Salami");
            mista.istBelegtMit("Pilzen");

            Gericht hawaii = new Pizza ("Hawaii");
            hawaii.istBelegtMit("Schinken");
            hawaii.istBelegtMit("Ananas");

            Burger hamburger = new Burger("Hamburger");
            hamburger.istBelegtMit("Hackfleisch");

            Gericht cheeseburger = new Burger("Cheeseburger");
            cheeseburger.istBelegtMit("Hackfleisch");
            cheeseburger.istBelegtMit("Käse");

            Gericht[] speisekarte = new Gericht[] { mista,
                        hawaii, hamburger, cheeseburger };

            for (Gericht g : speisekarte)
                  g.zubereitung();                        /* [1] */
      }
}
```

Geben Sie Implementierungen für die Klassen `Gericht`, `Pizza` und `Burger` an, sodass dieses Programmfragment kompiliert werden kann und durch die Ausführungen der mit `/* [1] */` markierten Zeile die folgenden Zeilen auf der Konsole ausgegeben werden.

```
Pizza Mista. Pizzaboden, belegt mit:
- Salami
- Pilzen
Pizza Hawaii. Pizzaboden, belegt mit:
- Schinken
- Ananas
Hamburger. Brötchen mit:
- Hackfleisch
Cheeseburger. Brötchen mit:
- Hackfleisch
- Käse
```

Diese Zeichenketten sollen selbstverständlich durch die Auswertung des zuvor zusammengesetzten Objektgeflechts entstehen. Nutzen Sie die Vererbung von Instanzvariablen und Instanzmethoden geschickt!

(Fortsetzung der Aufgabe auf der folgenden Seite)

(Fortsetzung von Aufgabe 2)

Aufgabe 3: Aufzählungstypen (10 Punkte)

Gegeben sei das folgende Java-Programmfragment:

```
public class MeinObstkorb {

      public static void main(String[] args) {
            alleFrüchteMitFarbe(Farbe.GELB);
            alleFrüchteMitFarbe(Farbe.GRÜN);
            alleFrüchteMitFarbe(Farbe.ROT);
      }
      /* [2] */

}
/* [1] */
```

Ergänzen Sie das Programm an den beiden markierten Stellen, sodass bei Terminierung der main-Methode die folgenden Zeilen auf der Konsole ausgegeben wurden:

```
Früchte mit Farbe GELB :
- BANANE
Früchte mit Farbe GRÜN :
- APFEL
Früchte mit Farbe ROT :
- ERDBEERE
- KIRSCHE
```

Ergänzen Sie hierzu das Programm an der mit `/* [1] */` markierten Stelle um Deklarationen der Aufzählungstypen `Farbe` (mit den Elementen `GELB`, `GRÜN` und `ROT`) und `Obst` (mit den Elementen `APFEL`, `BANANE`, `ERDBEERE` und `KIRSCHE`). Wählen Sie Ihre Implementierung so, dass zu jeder Obstsorte (als Element des Aufzählungstypen `Obst`) bereits bei ihrer Deklaration ihre Farbe angegeben wird und sie mit der Methode `farbe()` über sie Auskunft geben kann.

Ergänzen Sie zudem das Programm an der mit `/* [2] */` markierten Stelle um eine Implementierung der Methode `alleFrüchteMitFarbe(Farbe)`, sodass das Programm das oben beschriebene Verhalten aufweist.

(Fortsetzung der Aufgabe auf der folgenden Seite)

(Fortsetzung von Aufgabe 3)

Ergänzung an der mit `/* [1] */` markierten Stelle: (6 Punkte)

Ergänzung an der mit `/* [2] */` markierten Stelle: (4 Punkte)

Aufgabe 4: Überladen und Überschreiben (10 Punkte)

Gegeben sei das folgende Java-Programm:

```
class Person {}
class Erwachsener extends Person {}
class Kind extends Person {}

class Fahrzeug {
      void fahre (Person p1, Person p2) {
            System.out.println("Person und Person im Fahrzeug");          }
      void fahre (Person p, Erwachsener e) {
            System.out.println("Person und Erwachsener im Fahrzeug");     }
}

class Auto extends Fahrzeug {
      void fahre (Person p, Erwachsener e) {
            System.out.println("Person und Erwachsener im Auto");         }
      void fahre (Kind k, Erwachsener e) {
            System.out.println("Kind und Erwachsener im Auto");           }
}

class Boot extends Fahrzeug {
      void fahre (Person p, Erwachsener e) {
            System.out.println("Person und Erwachsener im Boot");         }
      void fahre (Kind k, Erwachsener e) {
            System.out.println("Kind und Erwachsener im Boot");           }
}

class Gummiboot extends Boot {
      void fahre (Person p, Erwachsener e) {
            System.out.println("Person und Erwachsener im Gummiboot");    }
}

class VonPersonenUndFahrzeugen {

      public static void main(String[] args) {

            Person person = new Person();
            Erwachsener erwachsener = new Erwachsener();
            Kind kind = new Kind();

            Fahrzeug fahrzeug1 = new Auto();
            Fahrzeug fahrzeug2 = new Gummiboot();
            Boot boot = new Gummiboot();

            /* [1] */
      }
}
```

(Fortsetzung der Aufgabe auf der folgenden Seite)

(Fortsetzung von Aufgabe 4)

An der mit `/* [1] */` markierten Stelle werden nun (einzeln und nacheinander) die folgenden Befehle eingefügt. Kann das Programm nun kompiliert werden? Wenn ja, geben Sie die durch diese Anweisung ausgegebene Zeile an; wenn nein: Warum nicht?

```
fahrzeug1.fahre(kind, erwachsener);
```
(1 Punkt)

```
fahrzeug1.fahre(person, erwachsener);
```
(1 Punkt)

```
fahrzeug2.fahre(kind, kind);
```
(1 Punkt)

```
boot.fahre(erwachsener, erwachsener);
```
(1 Punkt)

```
boot.fahre(kind, erwachsener);
```
(1 Punkt)

(Fortsetzung der Aufgabe auf der folgenden Seite)

(Fortsetzung von Aufgabe 4)

```
new Gummiboot(){
      void fahre(Person p, Erwachsener e) {
            System.out.println("Person und Erwachsener im knallroten Gummiboot.");
      }
}.fahre(kind, erwachsener);
```

(2 Punkte)

```
new Auto(){
      Integer fahre (Kind k, Erwachsener e) {
            System.out.println("Kind und Erwachsener im Sportauto.");
            return 0;
      }
}.fahre(kind, erwachsener);
```

(3 Punkte)

Aufgabe 5: Polymorphie (10 Punkte)

Welche vier Arten von Polymorphie kennen Sie? (2 Punkte)

Charakterisieren Sie jede Art kurz! (jeweils 2 Punkte)

Aufgabe 6: (Beschränkt) Parametrische Polymorphie(10 Punkte)

Gegeben sei das folgende Programmfragment:

```
public class Restaurant {
	public static void main(String[] args) {
		Herd herd = new Herd();
		Ofen ofen = new Ofen();
		Suppe erbsensuppe = new Suppe(herd);
		Pizza pizzaMista = new Pizza(ofen);
		Gericht<?> einGericht = erbsensuppe;

		Zubereitungsgerät gerät = einGericht.zubereitungsgerät();
		Herd suppenherd = erbsensuppe.zubereitungsgerät();
		Ofen pizzaofen = pizzaMista.zubereitungsgerät();
		/* [1] */
	}
}

class Zubereitungsgerät { }
class Herd extends Zubereitungsgerät { }
class Ofen extends Zubereitungsgerät { }
```

Geben Sie eine Implementierung für die Klassen `Gericht`, `Suppe` und `Pizza` an, sodass dieses Programmfragment kompiliert. Die Methode `zubereitungsgerät()` soll insgesamt nur ein einziges Mal implementiert werden. (5 Punkte)

(Fortsetzung der Aufgabe auf der folgenden Seite)

(Fortsetzung von Aufgabe 6)

Die `main`-Methode wird nun an der mit `/* [1] */` markierten Stelle um die folgenden Zeilen ergänzt:

```
Herd andererHerd = new Herd();
Suppe hühnersuppe = new Suppe(andererHerd);
erbsensuppe.platzTauschen(hühnersuppe);

Ofen andererOfen = new Ofen();
Pizza pizzaHawaii = new Pizza(andererOfen);
pizzaMista.platzTauschen(pizzaHawaii);
```

Während diese zwei Aufrufe von `platzTauschen(.)` durch den Compiler akzeptiert werden sollen, soll der Aufruf von

```
erbsensuppe.platzTauschen(pizzaMista);
```

zurückgewiesen werden.

Geben Sie eine (einzige) Implementierung der Methode `platzTauschen(.)` an, die diese Kriterien erfüllt. In welcher Klasse steht diese? (2,5 Punkte)

Erklären Sie, warum der Compiler den letztgenannten Aufruf zurückweist. (2,5 Punkte)

Aufgabe 7: Aufbau eines Java-Prgramms (10 Punkte)

Der Einstiegspunkt in ein Java-Programm ist die so genannte `main`-Methode. Sie muss innerhalb einer Klasse definiert werden und die in der mit `/* [1] */` markierten Zeile dargestellte erweiterte Signatur haben.

```
public class Programmstarter {
      public static void main(String[] args) {          /* [1] */
            System.out.println("Hallo Welt!");
      }
}
```

Warum muss die `main`-Methode als Klassenmethode (also mit dem Schlüsselwort `static`) deklariert werden? (2 Punkte)

Warum ist das Schlüsselwort `void` notwendig? (1 Punkt)

(Fortsetzung der Aufgabe auf der folgenden Seite)

(Fortsetzung von Aufgabe 7)

Wozu dient der Parameter `args`? Warum ist er mit einem String-Array typisiert? (2 Punkte)

Aus welchen Einheiten besteht ein Java-Programm und woher können Sie stammen? (1 Punkt)

Welche Möglichkeiten gibt es, diese Einheiten zu organisieren? Welche Vorteile hat eine sorgfältige Organisation dieser Einheiten? (4 Punkte)

Aufgabe 8: Fehlersuche (10 Punkte)

In das folgende Java-Programm haben sich einige Fehler eingeschlichen.

```
1  interface EineSchnittstelle {
2        void gehWeg();
3  }
4
5  final abstract class MeineKlasse {
6        abstract void tuWas();
7  }
8
9  class AndereKlasse implements EineSchnittstelle {}
10
11 public class Fehlerteufel {
12       public static void main(String[] args) {
13             EineSchnittstelle einI = new MeineKlasse();
14             springHoch();
15       }
16       void springHoch(){
17       }
18
```

Identifizieren Sie die fünf Fehler anhand Ihrer Zeilennummern und erklären Sie sie.

Fehler (1) (2 Punkte)

Fehler (2) (2 Punkte)

Fehler (3) (2 Punkte)

Fehler (4) (2 Punkte)

Fehler (5) (2 Punkte)

Aufgabe 9: Lokale und anonyme Klassen (10 Punkte)

Geben Sie eine kompilierbare Java-Klasse an, in der eine *lokale* Klasse deklariert, eine Instanz von ihr erzeugt und diese einer Variablen zugewiesen wird. (2 Punkte)

Geben Sie eine kompilierbare Java-Klasse an, in der eine Instanz einer *anonymen* Klasse erzeugt und einer Variablen zugewiesen wird. (2 Punkte)

In welchem Kontext werden anonyme Klassen in Java häufig verwendet? Warum eignen sie sich hier besonders gut? (2 Punkte)

Welche Gemeinsamkeiten und welche wesentlichen Unterschiede bestehen zwischen einer lokalen und einer anonymen Klasse? (4 Punkte)

Aufgabe 10: Stromklassen & Objektströme (10 Punkte)

Geben Sie an, ob die folgenden Aussagen wahr oder falsch sind.

Achtung! Falsche Antworten geben Minuspunkte, eine negative Gesamtpunktzahl bei dieser Aufgabe wird Ihnen aber auf null Punkte aufgerundet.

Stromklassen dienen dazu, Folgen von Daten zu lesen bzw. zu schreiben.	Wahr	Falsch
Stromklassen können immer nur einzeln verwendet werden; um sie zu verbinden, sind sog. Adapter notwendig.	Wahr	Falsch
Klassen, die die Schnittstelle `java.io.Reader` implementieren, signalisieren das Erreichen des Endes des Stromes in der Methode mit dem Namen `read` durch Rückgabewert `null`.	Wahr	Falsch
Die `Reader`- und `Writer`-Klassen stellen Stromklassen auf Grundlage des Typs `byte` dar, während die `InputStream`- und `OutputStream`-Klassen auf Grundlage des Typs `char` arbeiten.	Wahr	Falsch
Die Stromklassen zum Lesen und die zum Schreiben von Daten sind zu großen Teilen symmetrisch; es gibt z.B. Klassen `FileInputStream` und `FileOutputStream` oder `PipedReader` und `PipedWriter`.	Wahr	Falsch
Stromklassen können auch dazu verwendet werden, Objektgeflechte zwischen verschiedenen Prozessen auszutauschen.	Wahr	Falsch
Wird aus einem Objektgeflecht ein Objektstrom zu erzeugt, werden die Referenzen zwischen Objekten durch ihre unveränderten Speicheradressen abgebildet.	Wahr	Falsch
Um ein Objektgeflecht wieder einlesen zu können, müssen die Klassen der enthaltenen Objekte in dem einlesenden Programm zur Verfügung stehen.	Wahr	Falsch
Durch eine unvorsichtige Implementierung können sich durch das Einlesen eines zuvor erzeugten Objektstroms Duplikate von Objekten „einschleichen".	Wahr	Falsch
An die Klasse eines Objektes, welches einem Objektstrom hinzugefügt werden soll, werden keine Anforderungen gestellt.	Wahr	Falsch

Zusätzlicher Platz für Ihre Lösungen

Ergänzung zu Aufgabe Nr.

Ergänzung zu Aufgabe Nr.

Zusätzlicher Platz für Ihre Lösungen

Ergänzung zu Aufgabe Nr.

Ergänzung zu Aufgabe Nr.