FernUniversität in Hagen

FAKULTÄT FÜR
MATHEMATIK UND INFORMATIK

Lehrgebiet Programmiersysteme
Prof. Dr. Friedrich Steimann

_____
(Name, Vorname)
_____

_____
(Straße, Nr.)

_____ _____
(PLZ) (Wohnort)

_____
(Land, falls außerhalb Deutschlands)

Kurs 1853 SS 2013

„Moderne Programmiertechniken und -Methoden“

Klausur am 07.09.2013

Dauer: 3 Std., 10 – 13 Uhr

Lesen Sie zuerst die Hinweise auf der folgenden Seite!

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

Klausurort: _____

Aufgabe	1	2	3	4	5	6	7	8	9	10	Summe
habe bear-beitet											
Maximal	10	10	10	10	10	10	10	10	10	10	100
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. Die nächste Klausur findet im Sommersemester 2014 statt.

Hagen, den 20.09.2013

Im Auftrag

Hinweise zur Bearbeitung

1. Prüfen Sie die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst auf insgesamt 22 Seiten :
   – 1 Deckblatt
   – Diese Hinweise zur Bearbeitung
   – 10 Aufgaben auf Seite 1-18
   – Zwei zusätzliche Seiten 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 Leerraum.
   Benutzen Sie auf keinem 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 an dieser Klausur anhängenden 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 40 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: Interfaces (5 P + 5P = 10 Punkte)

a.) Im Kurstext wurden Interfaces in die folgenden Kategorien unterteilt:

- anbietendes Interface
- allgemeines Interface
- kontextspezifisches Interface
- ermöglichendes Interface.

Zu welcher oder zu welchen dieser Kategorien gehören

- idiosynkratische Interfaces
- Familieninterfaces
- Client/Server-Interfaces
- Server/Client-Interfaces
- Server/Item-Interfaces?

Antwort:

a.)
- Idiosynkratische Interfaces gehören zu:

- Familieninterfaces gehören zu:

- Client/Server-Interfaces gehören zu:

- Server/Client-Interfaces gehören zu:

- Server/Item-Interfaces gehören zu:

b.) Beschreiben und vergleichen Sie idiosynkratisches und Familien-Interface.

Antwort:

idiosynkratisches Interface:

Familien-Interface:

Aufgabe 2: Interfaces (4 + 4 + 2 = 10 Punkte)

a.) Gegeben sind die folgenden Arten von Interfaces und die beiden Interfaces `Runnable` und `Comparable` aus der Java-API:

- idiosynkratisches Interface
- Familien-Interface
- Client/Server-Interface
- Server/Client-Interface
- Server/Item-Interface.

```
*****************************************************
  public interface Runnable {
     void run();
  }
  class Client implements Runnable {
     ...
     public void run() {...}
  }
  Client client = new Client();
  new Thread(client).start();
*****************************************************

*****************************************************
  class Server {
     ...
        void sort(List<? extends Comparable> items) {
           ...
          if (a.compareTo(b) > 0) {...}
             ...
          }
  }

  class Client {
     Server server;
     List<Comparable> items;
        ...
        server.sort(items);
           ...
  }
*****************************************************
```

Welcher der oben genannten Kategorien von Interfaces kann `Runnable` und welcher `Comparable` zugeordnet werden? Begründen Sie Ihre Antwort.

Antwort:

`Runnable` ist ein:

Begründung:

`Comparable` ist ein:

Begründung;:

b.) Wir betrachten anbietende und ermöglichende Interfaces. Welche kommen vermehrt in Bibliotheken vor und welche in Blackbox-Frameworks?

Antwort:

In Bibliotheken:

In Blackbox-Frameworks:

Aufgabe 3: Design by Contract (6 + 4 = 10 Punkte)

a) Welche technischen Eigenschaften sollte eine Design-by-Contract-Sprache besitzen?

Antwort:

1.

2.

3.

b) Welche der im Kurstext behandelten Design-by-Contract-Sprachen eignet sich dazu, Interfaces mit Semantik zu versehen? Begründen Sie Ihre Antwort.

Antwort:

Begründung:

Aufgabe 4: Design by Contract (3 + 4 + 3 = 10 Punkte)

JAVA hat mit der Version 1.4 ein Schlüsselwort `assert` spendiert bekommen. Mit ihm können beliebige boolesche JAVA-Ausdrücke zu Zusicherungen (engl. assertions) gemacht werden.

a) Ein Problem dieser JAVA-Assertions zeigt das folgende Programmstück.
Um welches Problem handelt es sich?
Gibt es eine andere Design-by-Contract-Sprache, die für dieses Problem eine Lösung anbietet?

```
boolean hinein_in_die_Kartoffeln() {
     assert raus_aus_den_Kartoffeln();
     return true;
}

boolean raus_aus_den_Kartoffeln() {
     assert hinein_in_die_Kartoffeln();
     return true;
}
```

Antwort:

b) Welche Unzulänglichkeiten gibt es bei JAVA-`Asserts` zu beachten, die sich besonders bei der Formulierung von Nachbedingungen auswirken können?

Antwort:

1.

2.

c) Kann die Programmiersprache EIFFEL das Problem der Seiteneffekte und das Problem der mangelnden Redundanz lösen?

Antwort:

Aufgabe 5: Testen (4 + 6 P = 10 Punkte)

a) Beschreiben Sie bitte die Rolle von Mock-Objekten beim Testen.

Antwort:

b) Wie kann man dem zu testenden Objekt die Mock-Objekte unterschieben?
Geben Sie drei Möglichkeiten an und beschreiben Sie sie kurz.

Antwort:

1.)

2.)

3.)

Aufgabe 6: Entwurfsmuster, Refaktorisierungen (6 + 4 P = 10 Punkte)

a) Bei dem COMPOSITE PATTERN in seiner klassischen Form folgen Blätter als auch Komposita demselben Protokoll. Erläutern Sie, warum das nicht ohne Probleme ist.

Antwort:

b) Beschreiben Sie kurz das Refactoring EXTRACT INTERFACE:

Antwort:

Aufgabe 7: Refactoring durchführen (10 Punkte)

Es sei folgender Java-Code gegeben, der refaktorisiert werden soll. Wählen Sie fünf Refactorings aus der folgenden Liste aus und geben Sie an, wo und wie Sie diese Refactoring für diesen Code sinnvoll einsetzen können.

Liste der Refactorings:

- VERSCHACHTELTE BEDINGUNG DURCH WÄCHTER ERSETZEN
- INTRODUCE PARAMETEROBJECT
- PULL UP METHOD
- BEDINGUNG DURCH POLYMORPHISMUS ERSETZEN
- BEDINGUNGEN ZERLEGEN
- PULL UP FIELD
- EXTRACT CLASS
- ENCAPSULATE FIELD

Programmcode:

```
 1  abstract class Figure {
 2      protected boolean isVisible;
 3      protected boolean isDrawable;
 4      protected boolean printWhenInvisible;
 5      private String d;

 6      public void setDescription(String description) {
 7          this.d = description;
 8      }
 9      public String toString() {
10          String result = d;
11           if (!isDrawable) {
12               // nothing
13           } else {
14               if (isVisible) {
15                   result = d + " (will be drawn)";
16               } else {
17                   if (printWhenInvisible) {
18                       result = d + " (will be drawn but invisible)";
19                   }
20                }
21           }
22           return result;
23       }
24   }
25   class Rectangle extends Figure {
26       public int x_position;
```

```
27      public int y_position;
28      public int width;
29      public int length;

30      public int getPerimeter()
31          return 2 * width + 2 * length;
32      }
33      public void setPosition(int x, int y) {
34          this.x_position = x;
35          this.y_position = y;
36      }
37      public void draw(int foreground_red, int foreground_green,
38              int foreground_blue, int background_red, int background_green,
39              int background_blue) {
40          if (isDrawable && (isVisible || printWhenInvisible)) {
41              /* ... draws Rectangle with given fore- and background color ... */
42          }
43      }
44  }
45  class Square extends Figure {
46      public int x_position;
47      public int y_position;
48      public int width;

49      public int getPerimeter() {
50          return 4 * width;
51      }
52      public void setPosition(int x, int y) {
53          this.x_position = x;
54          this.y_position = y;
55      }
56      public void draw(int foreground_red, int foreground_green,
57          int foreground_blue, int background_red, int background_green,
58          int background_blue) {
59          if (isDrawable && (isVisible || printWhenInvisible)) {
60              /* ... draws Square with given fore- and background color ... */
61          }
62      }
63  }
```

Antwort:

1.

2.

3.

4.

5.

Aufgabe 8: Vererbung und offene Rekursion (4 + 4 + 2 P = 10 Punkte)

a) Folgende Konstellation zweier Klassen enthält einen Aufruf einer geerbten Methode und eine offene Rekursion. Identifizieren Sie beide.

```
 1 class Pferd {
 2   void wiehern() {
 3        fressen();
 4   }

 5   void fressen() {
 6        System.out.println("Superfutter");
 7   }
 8 }

 9 class Haflinger extends Pferd {
10    void mitreden() {
11         wiehern();
12    }

13    void fressen() {
14         System.out.println("Hafifutter");
15    }
16 }
```

Antwort:

b.) Erläutern Sie den Unterschied zwischen Forwarding und Delegation, indem Sie auf die Bindung von `this` eingehen.

Antwort:

c.) Bei welchem Refactoring ist es wichtig, den Unterschied zwischen **Forwarding** und **Delegation** zu kennen? Begründen Sie Ihre Antwort.

Antwort:

Aufgabe 9: Metaprogrammierung (4 + 2 + 4 P = 10 Punkte)

a) Bei der Metaprogrammierung unterscheidet man zwischen **reflexiver** und **nicht- reflexiver** Metaprogrammierung. Die reflexive Metaprogrammierung unterteilt sich noch in die Bereiche **Introspektion**, **Intersession**, **Modifikation**. Um welche Art von Metaprogrammierung handelt es sich bei

- Java Reflexion API,
- aspektorientierter Programmierung,
- Lisp oder Prolog,
- JUNIT 3.8,
- Debugger,
- Compiler,
- Refactoring-Tools,
- genetische Algorithmen?

Antwort:

- Java Reflexion API:
- aspektorientierter Programmierung:
- Lisp oder Prolog:
- JUNIT 3.8:
- Debugger:
- Compiler:
- Refactoring-Tools:
- genetische Algorithmen:

b) Werden Annotationen nur in reflexiven Programmen eingesetzt? Begründen Sie Ihre Antwort.

Antwort:

c) Ihre Firma schreibt ein Programm, um die Stellwerke der Deutschen Bundesbahn besser verwalten zu können. Ihre Programmierer wollen dabei für typische Cross-Cutting-Concerns aspektorientierte Programmierung einsetzen.
Halten Sie dies für eine gute oder eher schlechte Idee? Begründen Sie Ihre Antwort.

Antwort:

Aufgabe 10: Extreme Programming (10 Punkte)

Benennen Sie fünf der häufigsten Risiken in der Softwareentwicklung, für die Extreme Programming eine (gute) Lösung anbietet. Beschreiben Sie diese Lösung kurz.

Antwort:

- Risiko 1:
- Risiko 2:
- Risiko 3:
- Risiko 4:
- Risiko 5:
- Antwort auf Risiko Nr. 1:
- Antwort auf Risiko Nr. 2:
- Antwort auf Risiko Nr. 3:

- Antwort auf Risiko Nr. 4:

- Antwort auf Risiko Nr. 5:

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.