Aufgabe 1: Interfaces (5 + 5 Punkte)

a) In der Kurseinheit 1 des Kurses 1853 werden drei Prinzipien der Interfacebasierten Programmierung genannt:

1. Das erste Prinzip wiederverwendbaren objektorientierten Designs
2. Interface Segregation Principle
3. Dependency Inversion Principle

Was besagen diese Prinzipien?

Antwort:

1.

2.

3.

b) Das Dependency Inversion Principle hat zwei Schönheitsfehler:

- Da ist die Abhängigkeit, die durch die Objekterzeugung entsteht und die sich nur mit einigem Aufwand beseitigen lässt (z. B. durch der Dependency injection).
- Den zweiten Schönheitsfehler erkennt man an folgendem Beispiel:
  Das links stehende Programmstück ohne Interfaces wird in das rechts stehende mit den beiden Interfaces IB und IC transformiert:

```
package a;

import b.B;
import c.C;

class A {
      B b;
      C c;
      ...
         b.m(c);
         c.n();
      ...
 }
```

wird geändert in

```
package a;

interface IB { void m(C c); }
interface IC { void n(); }

class A {
       IB b;
       IC c;
       ...
          b.m(c);
          c.n();
       ...
```

```
 package b;

 import c.C;

 class B {
      void m(C c){...}
 }
```

```
  package b;

  import c.C;
  import a.IB;

  class B implements IB {
       void m(C c){...}
  }
```

```
 package c;

 class C {
      void n(){...}
 }
```

```
  package c;

  import a.IC;

  class C implements IC {
       void n(){...}
  }
```

Was ist am Interface-Beispiel falsch, was nicht optimal?

Antwort:

Aufgabe 2: Interfaces (2 + 5 + 3 Punkte)

1.) Das Vorliegen interfacebasierter Programmierung ist hauptsächlich an zwei Anzeichen erkennbar. Welche beiden Anzeichen sind das?

Antwort:

A1:

A2:

2.) Welche Art des Gebrauchs von Interfaces kommt im Zusammenhang mit

a) Factories
b) Black-Box-Frameworks
c) Bibliotheken
d) Threads
e) Event-Listener-Mechanismus

häufig vor?

Antwort:

a) **Factories:**

b) **Black-Box-Frameworks:**

c) **Bibliotheken:**

d) **Threads:**

e) **Event-Listener-Mechanismus:**

3.) Woraus besteht das Klasseninterface der folgenden JAVA-Klasse A?

```
public class A {
     int i;
     public m() {...}
     public n() {...}
     protected o() {...}
     private p() {...}
     q() {...}
}
```

Antwort:

Aufgabe 3: Dependency Injection (6 + 3 + 1 Punkte)

a) Gegeben sei das Java-Interface `IServer` und die davon abhängige Java-Klasse `Client`. Beschreiben Sie, wie Interface injection funktioniert: Geben Sie bitte dazu den Programm-Code des zugehörigen Interfaces und der Klasse `Client` an und erläutern Sie den Code kurz.

Antwort:

b) Sie möchten Dependency Injection bei dem folgenden Codesegment einsetzen. Welches Problem könnte sich dabei stellen?

```
if (...)
     server = Server();
else
     server = Server("www.fernuni-hagen.de");
```

Antwort:

c) Welche Alternative zur Dependency Injection gibt es?

Antwort:

Aufgabe 4: Design by contract (3 + 7 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?

```
boolean ping() {
     assert pong();
     return true;
}

boolean pong() {
     assert ping();
     return true;
}
```

Antwort:

b) Ein Standardbeispiel für Objekte mit Zustand ist die Klasse Stack.

```
1 class Stack<T> {
2
3    private static final int MAXIMUM_CAPACITY = 100;
4    private int capacity = 0;
5    private StackElement top = null;
6
7    public void push(T element) {
8       top = new StackElement(top, element);
9       capacity++;
10    }
11
12    public T pop() {
13       T removedElement = top();
14       top = top.getNext();
15       capacity--;
16       return removedElement;
17    }
18
19    public T top() {
```

```
20        return top.getElement();
21     }
22
23     private class StackElement {
24        private T element;
25        private StackElement next;
26
27        public StackElement(StackElement next, T element) {
28        this.next = next;
29        this.element = element;
30        }
31
32      public T getElement() {
33        return element;
34       }
35
36     public StackElement getNext() {
37        return next;
38      }
39     }
40 }
```

Für die Methoden push, pop und top können die folgenden Nachbedingungen formuliert werden:

- **Nachbedingung für die Methode push:** Der Stack ist nicht (mehr) leer, er kann sogar voll sein, seine Größe ist um 1 angewachsen, das mit push auf den Stack gelegte Element liegt oben, ansonsten hat sich nichts verändert.
- **Nachbedingung für die Methode pop:** Der Stack ist nicht (mehr) voll, er kann sogar leer sein, seine Größe ist um 1 geschrumpft, das zuletzt mit push auf den Stack gelegte Element liegt nicht mehr darauf.
- **Nachbedingung für die Methode top:** Das zuletzt mit push auf den Stack gelegte Element wird zurückgegeben, der Stack hat sich nicht verändert

Auf welche Probleme stoßen Sie, wenn Sie die Nachbedingungen von push und pop mittels Assert-Statements überprüfen mögen?
Halten Sie es für sinnvoll, auch Nachbedingungen für top mittels assert zu prüfen?

Antwort:

Antwort-Fortsetzung:

Aufgabe 5: Testen (4 + 6 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: Interfaces und Testen (5 + 5 Punkte)

a) Beschreiben Sie kurz das Testframework JUNIT. Beantworten Sie dazu die folgenden Fragen:

- Aus welchen drei Phasen sollte jeder Testfall bestehen?
- Wie erfolgt der Aufruf der Testmethoden?
- Wie erfolgt die Zuordnung von Testfällen zu den zu testenden Programmeinheiten?

Antwort:

- Jeder Testfall sollte aus den folgenden Phasen bestehen:
  *
  *
  *
- Der Aufruf der Testmethoden
- Die Zuordnung von Testfällen

b) Wie kann man in Java mit JUnit Interfaces testen?

Antwort:

Antwort Fortsetzung:

Aufgabe 7: Factories (4 + 2 + 4 Punkte)

Die Einführung von Interfaces oder abstrakten Klassen in ein Programm kann der Entkopplung dienen. Doch erst wenn in einem Teilprogramm (oder einer Komponente) ein Klassenname gar nicht mehr erwähnt wird, ist es von der Klasse wirklich unabhängig, also entkoppelt. Eine Möglichkeit, dies zu erreichen, sind Factory-Methoden.

a) Schreiben Sie bitte das folgende Programmsegment so um, dass statt des Konstruktoraufrufs eine Factory-Methode verwendet wird, und geben Sie die Klasse `Factory` an.

```
class Client {
     IServer x = new Server();
}
```

Antwort:

b) Eine Factory-Methode rentiert sich erst dann, wenn alle Objekte von einer Methode erzeugt werden und der Typ des erzeugten Objekts von den Parametern dieser Methode (oder irgendeiner anderen Bedingung) abhängt.
Wie könnte in diesem Fall die Klasse `Factory` schematisch aussehen?

Antwort:

Aufgabe 8: Entwurfsmuster (3 + 1 + 2 + 1 + 3 Punkte)

a) Nennen Sie bitte drei Vorteile der Entwurfsmuster.

Antwort:

1.

2.

3.

b) Welchen Sachverhalt beschreibt das OBSERVER Pattern?

Antwort:

c) Was ist ein klassisches Einsatzgebiet für das OBSERVER Pattern?

Antwort:

d) Was versteht man in Zusammenhang mit dem OBSERVER Pattern unter einem *Publish-subscribe-Verfahren*?

Antwort:

e) Sollte man für das OBSERVER Pattern die Art und Anzahl der Observer eines Objektes kennen?

Antwort:

Aufgabe 9: Refactoring (6 + 4 Punkte)

Es sei folgender Java-Code gegeben:

```
abstract class Figure {
    protected boolean isVisible;
    protected boolean isDrawable;
    protected boolean printWhenInvisible;
    private String d;

    public void setDescription(String description) {
        this.d = description;
    }
    public String toString() {
        String result = d;
        if (!isDrawable) {
            // nothing
        } else {
            if (isVisible) {
                result = d + " (will be drawn)";
            } else {
                if (printWhenInvisible) {
                    result = d + " (will be drawn but invisible)";
                }
           }
        }
        return result;
    }
}
class Rectangle extends Figure {
    public int x_position;
    public int y_position;
    public int width;
    public int length;

    public int getPerimeter()
        return 2 * width + 2 * length;
    }
    public void setPosition(int x, int y) {
        this.x_position = x;
        this.y_position = y;
    }
    public void draw(int foreground_red, int foreground_green,
            int foreground_blue, int background_red, int background_green,
            int background_blue) {
        if (isDrawable && (isVisible || printWhenInvisible)) {
            /* ... draws Rectangle with given fore- and background color ... */
        }
    }
}
class Square extends Figure {
    public int x_position;
    public int y_position;
    public int width;
```

```
    public int getPerimeter() {
        return 4 * width;
    }
    public void setPosition(int x, int y) {
        this.x_position = x;
        this.y_position = y;
    }
    public void draw(int foreground_red, int foreground_green,
        int foreground_blue, int background_red, int background_green,
        int background_blue) {
        if (isDrawable && (isVisible || printWhenInvisible)) {
            /* ... draws Square with given fore- and background color ... */
        }
    }
}
```

a) Finden Sie für jedes der unten aufgelisteten Refactorings eine Stelle, an der Sie es einsetzen können.
Refactorings:

1. VERSCHACHTELTE BEDINGUNG DURCH WÄCHTER ERSETZEN
2. INTRODUCE PARAMETER OBJECT
3. PULL UP FIELD
4. EXTRACT CLASS

Antwort:

1. VERSCHACHTELTE BEDINGUNG DURCH WÄCHTER ERSETZEN:

2. INTRODUCE PARAMETER OBJECT:

3. PULL UP FIELD:

4. EXTRACT CLASS:

b) Der redundante Code innerhalb der beiden Methoden `draw` in `Rectangle` und `Square` lädt ebenfalls zur Refaktorisierung ein. Was könnte man hier tun und was ist dabei zu beachten?

Antwort:

Aufgabe 10: Extreme Programming (5 + 5 Punkte)

Benennen Sie fünf der zehn häufigsten Risiken in der Softwareentwicklung nach Boehm.

Antwort:

- Risiko 1:
- Risiko 2:
- Risiko 3:
- Risiko 4:
- Risiko 5:

Extreme Programming hält für die meisten dieser Risiken eine Antwort bereit. Beschreiben Sie zu den von Ihnen genannten Risiken, ob und wie Extreme Programming hilft, sie zu vermeiden.

Antwort:

- 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: