Prof. Dr. Simone Strippgen

Klausur

Programmierung I – Konzepte

Datum: 21.03.2012

Name: [redacted] Vorname: [redacted]

Matr.-Nr.: [redacted] letzter Versuch: ☐ Ja ☒ Nein

Bitte zuerst den Kopf dieses Blattes vollständig und leserlich ausfüllen. Lassen Sie bei der Bearbeitung der Aufgaben die Blätter zusammen geheftet. Bitte schreiben Sie die Lösungen unter den jeweiligen Aufgabentext. Benutzen Sie kein eigenes Papier (außer zum Vorschreiben).

Viel Erfolg!

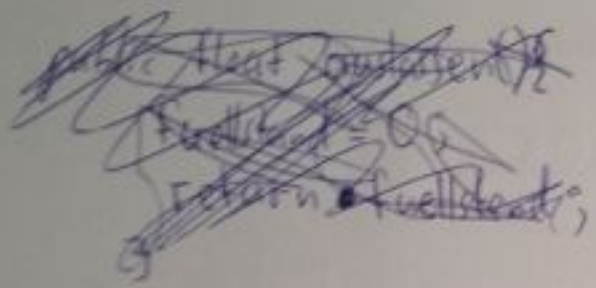
Aufgabe	Mögliche Punkte	Erstkorrektur	Zweitkorrektur
Aufgabe 1 : Aufwärmen	13	12	
Aufgabe 2: Sammlungen, Schleifen	19	8	
Aufgabe 3: HashMap	17	12,5	
Aufgabe 4: Klassenattribute/ -methoden	7	5,5	
Aufgabe 5: Vererbung, Überschreiben von Methoden	29	17,5	
Aufgabe 6: Interfaces	8	1	
Aufgabe 7: ArrayList, Substitution	7	4	
Punkte Gesamt	**100**	60,5	
Note		3	

Aufgabe 1: Aufwärmen

(12)

a) Schreiben Sie eine sondierende Methode *auslassen()*, die den Wert des Attributs *fuellstand* (vom Typ *float*) zurück liefert und das Attribut *fuellstand* auf 0 setzt.

(5 Punkte) 5



```
public float auslassen(){
    float i = fuellstand;
    fuellstand = 0;
    return i;
}
```

b) Schreiben Sie eine verändernde Methode *aendern*, die ein Zeichen übergeben bekommt. Bei '+' soll sie auf das Attribut *menge* den Betrag in *delta* aufaddieren, bei '-' soll sie den Betrag von *menge* abziehen. Der Wert von *menge* darf dabei aber nicht größer als 100 werden.

(6 Punkte) 5

```
int delta
public void aendern(String zeichen){
    if(zeichen.equals("+")){
        menge = menge + delta;
    } else if(zeichen.equals("-")){
        menge = menge - delta;
```

c) Dem Konstruktor der Klasse *Hund* kann bei der Initialisierung eines Objekts der Bezeichner des Hundes mitgegeben werden. Auf welchen Wert verweist die Variable *hund2* nach den drei Zuweisungen.

(2 Punkte) 2

```
Hund hund1 = new Hund("Waldi");
Hund hund2 = new Hund("Prinz");
hund2 = hund1;
hund1 = null;
```

der Wert von hund2 ist "Waldi"

(8)

Aufgabe 2: Sammlungen, Schleifen, Klassenkonstanten, -methoden

a) Ergänzen Sie die Klasse *UmsatzZahlen* um ein privates Attribut *woechentlicheWerte*, das die mittleren Umsatzzahlen für jede Woche eines Jahres in einem *Array* aufnehmen kann. Die Umsatzzahlen sollen als Gleitkommawerte gespeichert werden. Die Anzahl der Kalenderwochen im Jahr (53) soll als private Klassenkonstante angelegt werden. Ergänzen Sie den Konstruktor um entsprechende Zuweisungen. (5 Punkte) 5

```
public class UmsatzZahlen {

    private String jahr;
private double[]woechentliche Werte;
private static final int kalenderwochen;
```

```
public UmsatzZahlen(String jahr)
{
wochentliche Wert = 0;
kalenderjahr = 0;
this.jahr = jahr;
```

b) Schreiben Sie für die Klasse *UmsatzZahlen* eine öffentliche Klassenmethode *berechneMittlerenUmsatz*, die ein *Array* mit Gleitkommawerten übergeben bekommt. Das *Array* enthält für eine beliebige Menge von Tagen den Tagesumsatz. Die Methode soll aus den Werten im *Array tagesumsaetze* das arithmetische Mittel der darin befindlichen Umsätze berechnen und diesen Wert als Rückgabewert liefern. (9 Punkte)

```
public double berechneMittlerenUmsatz(double[] tagesumsatz){
    double wert;
    for(int i=0; 0<=tagesumsatz.length; i++){
        wert = tagesumsatz[i];
        return wert;
    }
    return wert;
}
```

c) Schreiben Sie für die Klasse *UmsatzZahlen* eine öffentliche Methode *setzeWochenMittel*, die als Argumente eine Kalenderwoche (Zahl zwischen 1 und 53) und ein *Array* mit den Tagesumsätzen für jeden Tag der angegebenen Woche übergeben bekommt. Die Methode soll das arithmetische Mittel der Umsätze der entsprechenden Woche berechnen und ihn in dem Array *woechentlicheWerte* für die angegebene Woche setzen. Sorgen Sie dafür, dass die *Array*-Grenzen eingehalten werden. (5 Punkte)

```
public double setzeWoche_Mittel(int kalenderwoche, double[] tagesumsatz){
    return 0;
}
```

Aufgabe 3: HashMap

Es sei die folgende Klasse gegeben:

```
public class Buch {
   private String isbn;
   private float autor;
   ...

   public String gibIsbn() {
      return isbn;
   }

   public float gibAutor() {
     return autor;
   }
   ...
}
```

a) Deklarieren und Initialisieren Sie eine Variable *buecher* vom Typ *HashMap* in der man Objekte vom Typ *Buch* unter der ISBN des *Buch*-Objekts abspeichern kann. (2 Punkte)

```
HashMap<String, Buch> buecher = new HashMap<String, Buch>();
```

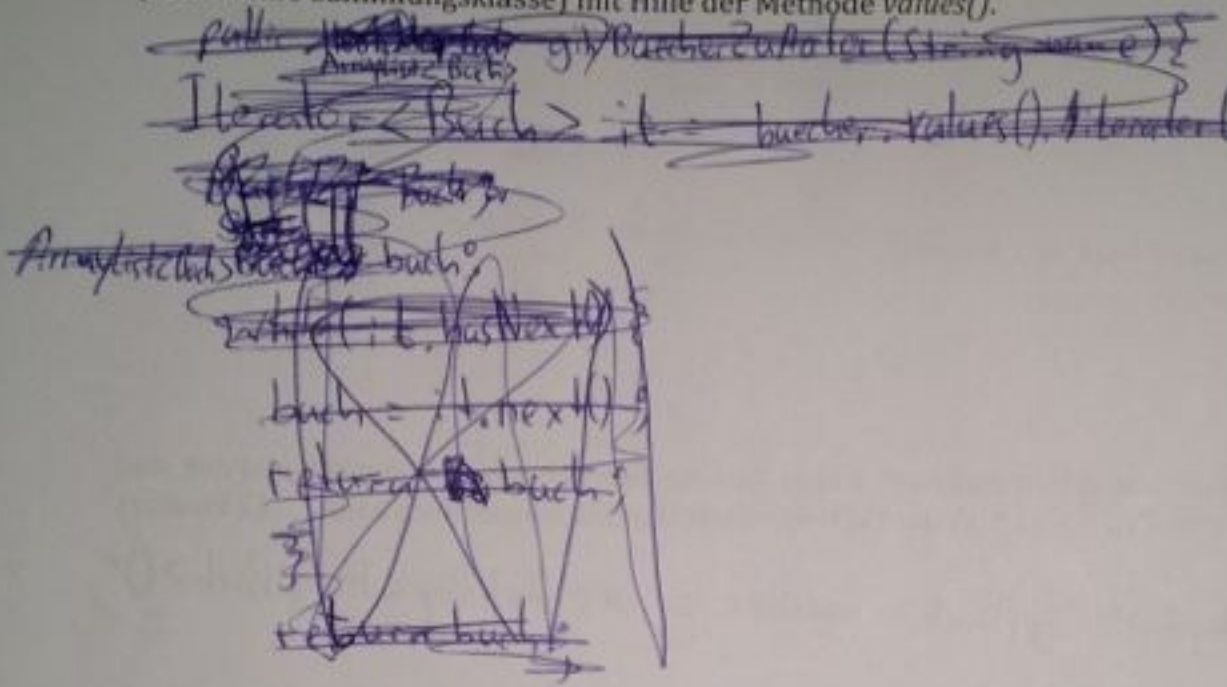
b) Schreiben Sie eine Methode *einfuegen(..)*, die ein *Buch*-Objekt übergeben bekommt und das *Buch*-Objekt in die HashMap *buecher* einfügt. Sorgen Sie dafür, dass es keine *NullPointerException* gibt und kein Buch eingefügt wird, dessen Autor den Wert "" hat. (5 Punkte)

```
public void einfuegen(Buch neuesBuch){
    if(!containsKey(neuesBuch.gibIsbn){
    buecher.put( neuesBuch.gibIsbn, neuesBuch);
    }
}
```

c) Schreiben Sie eine Methode *gibBuecherZuAutor(..)*, die als Argument den Namen eines Autors bekommt und als Rückgabewert eine *ArrayList* mit Objekten vom Typ *Buch* aus der HashMap *buecher* liefert, deren Autor den übergebenen Namen *name* trägt.

(10 Punkte) 7,5

Tipp: Hier müssen sie mit einem *Iterator* über die Wert-Objekte in der *HashMap* iterieren. Die Wert-Objekte der *HashMap* bekommen Sie als Instanz der Klasse *Collection* (selber eine Sammlungsklasse) mit Hilfe der Methode *values()*.



```
ArrayList<Buch>
public  g.b Bucher Zu Autor (String name){ 2
Iterator<Buch> it = buecher.values().iterator(); 2
ArrayList<Buch> buch = new ArrayList<Buch>(); 1
while(it.hasNext()) { ^
    if(containsKey(name)){
    buch = it.next(); ✗ f 0,5 buch ist Arraylist
    return Buch; f
    }
    }
    return buch; } ^
}
```

Aufgabe 4: Klassenattribute/-methoden (5,5)

Versehen Sie die Klasse *Mitarbeiter* mit einem Mechanismus, der es erlaubt, die Anzahl der *Mitarbeiter*-Instanzen abzufragen.
(7 Punkte) 5,5

```
public class Mitarbeiter {
    private String name;
    private String abteilung;
    private String gehaltsklasse;
    private ArrayList<Mitarbeiter> marb;   2
    ~~public Mitarbeiter~~
```

```
public Mitarbeiter(String name, String abteilung, String gehaltsklasse){   1
    this.name = name;
    this.abteilung = abteilung;
    this.gehaltsklasse = gehaltsklasse;
}
```

fehlt Auffüllen Arraylist

```
public int anzahl(){
    int anz=0;
    for(Mitarbeiter m: marb){                    1,5
        if(m != null){
            anz++;
        }
    } return anz;                                !
    return anz;
}
```

Aufgabe 5: Vererbung / Überschreiben von Methoden

Gegeben seien folgende Klassen *Spieler* und *Monster* aus einem „Ballerspiel". Modellieren Sie eine Klassenhierarchie, die die Gemeinsamkeiten der Spielfiguren so abbildet, dass sich leicht weitere Figurenarten integrieren lassen. Gehen Sie davon aus, dass sich alle Figuren mit mindestens einer Waffe ausstatten lassen und alle Figuren von den Waffen getroffen werden können. Versuchen Sie, möglichst viele allgemeine Eigenschaften und Verhaltensweisen bereits in der gemeinsamen Superklasse anzubieten.

Schreiben Sie die drei Klassen auf und definieren Sie sowohl die Konstruktoren (deren Rümpfe hier ausgelassen wurden) als auch die entsprechenden Methoden. Sie müssen keine getter- und setter-Methoden für die Attributwerte hinzufügen. Überlegen Sie, ob es abstrakte Methoden gibt.

```
public class Spieler {

    private String name;
    private int xPos;
    private int yPos;
    private String typ;
    private boolean sichtbar;
    private int energie;
    private ArrayList<Waffe> waffen;

    public Spieler(int x, int y, String typ, String name){
      ...
    }

    public void bewegen(int x, int y) {
        this.xPos = x;
        this.yPos = y;
        neuZeichnen();
    }

    private void neuZeichnen(){
        // Code zum Zeichnen
    }

    public void getroffen() {
        energie --;
    }

    public void addWaffe(Waffe waffe) {
        if(waffe != null) waffen.add(waffe);
    }
}
```

```
public class Monster {

    private String gattung;
    private Waffe waffe;
    private int xPos;
    private int yPos;
    private boolean getroffen;


    public Monster(String gattung, Waffe waffe, int x, int y){
     ...
    }

    public void bewegen(int x, int y) {
        this.xPos = x;
        this.yPos = y;
        neuZeichnen();
    }

    private void neuZeichnen(){
        // Code zum zeichnen
    }

    public void getroffen() {
        getroffen = true;
    }
}
```

a) Hier kommt die Superklasse hin: 8,5 (14 Punkte)

```
public abstract class Teilnehmer {                                  1
    protected Waffe waffe;                                          1
    protected String name;                                          1
    protected int xpos;                                             0,5
    protected int ypos;                                             0,5
    public Teilnehmer(String name, int xpos, int ypos, Waffe waffe) 0,5
        this.waffe = new Waffe();                                   1
        this.name = name;
        this.xpos = xpos;                                           0,5
        this.ypos = ypos;                                           0,5
    }
    public void bewegen(int x, int y) {                             1
        this.xpos = x;
        this.ypos = y;
    }                                                               1 d.ll.
    public void neuZeichnen();                                      1
    public void getroffen();                                        1
}                                                                   8

2 für addWaffe(..) in Spieler
```

b) Hier kommt die Klasse *Spieler* hin: (6 Punkte) 4,5

```
public class Spieler extends Teilnehmer{
 private String typ;
 private boolean sichtbar;
 private int energie;
 private ArrayList<Waffe> waffen;
  public Spieler(int x, int y, String typ, String name, Waffe waffe){
  super(name, x, y, waffe);
  this.typ = typ;
 }
 public void neuZeichnen(){
  super.bewegen();
 }
 public void getroffen(){
     energie--;
 }
 public void addWaffe(Waffe waffe){
     if(waffe != null) waffen.add(waffe);
 }
}
}
```

c) Hier kommt die Klasse *Monster* hin: 4,5 (9 Punkte)

```
public class Monster extends Teilnehmer{
~~private Waffe waffe;~~
private boolean getroffen;

public Monster (String gattung, Waffe waffe, int x, int y){
super(gattung, x, y, waffe);
~~this.waffe = new Waffe();~~
}

public void neuZeichnen(){
  super.bewege(xPos, xPos);
}
public void getroffen(){
    getroffen = true;
}

}
```

Aufgabe 3: Interfaces

Unten sehen Sie zwei Interfaces.

```
public interface A
{
    int zeit = 9;

    boolean isMethode1();

    void methode5(boolean gueltig);
}


public interface B extends A
{
    void methode3(int x, String name);

    float methode6(float b);
}
```

Schreiben Sie eine Klasse *EineKlasse*, die das Interface *B* implementiert. Die Methoden sollen lediglich ihren Namen (z.B. „methode5") auf der Konsole ausgeben. (8 Punkte)

```
pullic class EineKlasse implements B {

    public void methode5(boolean gueltig);
    public void methode3(int x, String name);
    public float methode6(float b);

    System.out.println("Name der Methode:"+"methode5");
    System.out.println("Name der Methode:"+"methode3");
    System.out.println("Name der Methode:"+"methode6");

}
```

Aufgabe 7: ArrayList, Substitution

(4)

Hier sehen Sie eine *Testlauf*-Klasse, die mit den Typen aus *Aufgabe6* arbeitet. Ergänzen Sie die main-Methode um eine konventionelle For-Schleife (mit Laufvariable *i*), in der auf den Objekten in *liste* alle Methoden aufgerufen werden, die das Objekt anbietet.
(7 Punkte) 4

```
public class Testlauf {

    public static void main(String[] args){

        ArrayList<B> liste = new ArrayList<A>();

        EineKlasse obj1 = new EineKlasse();

        EineKlasse obj2 = new EineKlasse();

        EineKlasse obj3 = new EineKlasse();

        liste.add(obj1);

        liste.add(obj2);

        liste.add(obj3);

       // Hier kommt Ihre For-Schleife hin
```

```
for(int i = 0; i <= liste.size(); i++){
    liste.get(i).methode5(true);
    liste.get(i).methode3( 3, "machen");
    liste.get(i).methode6( 3.4f);
}
}
}
```