

Aufgabe 1 Allgemeine Fragen
Allgemeine Fragen

(2+3+1+1+2+2+3+2 Punkte)

- (a) Was versteht man in Java unter einem *Interface*? Wozu dienen *Interfaces* in Java?
(Als Antwort genügen Stichpunkte oder maximal 2-3 Sätze.)

Ein Interface ist eine Schnittstelle in Java. Es dient dazu, in Klassen implementiert zu werden, um Methoden ^{vorgend} dem Interface definieren zu können.

zu wenig

- (b) Gegeben sei der folgende Java-Code:

```
public static void verdoppeln(int[] a) {  
    for (int i=0; i<a.length; i++) {  
        a[i] = a[i]*2;  
    }  
    return;  
}  
public static void verdoppeln(int a) {  
    a = 2*a;  
}
```

der Code wird folgendermaßen aufgerufen:

```
public static void main(String[] args) {  
    int x = 4;  
    int[] y = {1, 2, 3};  
    verdoppeln(x);  
    verdoppeln(y);  
    // *)  
}
```

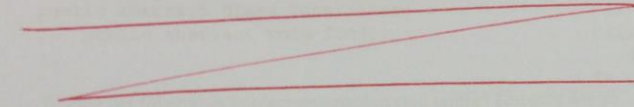
Was sind die Werte der Variablen x und y nach Ausführung an Position // *) der main-Methode?
Begründen Sie Ihre Antwort!

Ausgabe von x ist 4. Zwar wird die „verdoppeln“-Methode aufgerufen, jedoch wurde das x in der Methode nicht implementiert.

Begründung

Ausgabe von y ist {1, 2, 3}. Auch hier ändert sich nichts, da das y in den „verdoppeln“-Methoden nicht vorkommt.

- (c) Geben Sie die Menge $\{2^x \mid 0 \leq x \leq 4\}$ extensional an.

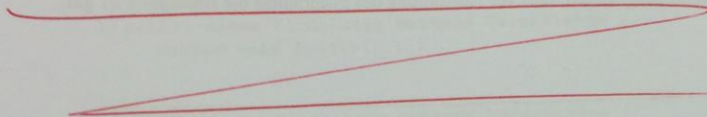


0

- (d) Betrachten Sie die Mengen $M_1 = a, b$, $M_2 = 1, 2, 3$.

Gegeben sei die Relation $R = \{(a, 1), (a, 2), (b, 2)\} \subseteq M_1 \times M_2$.

Ist die Relation R eine Funktion von M_1 nach M_2 ? Begründen Sie ihre Entscheidung.



0

- (e) Beweisen Sie folgende Behauptung durch vollständige Induktion:

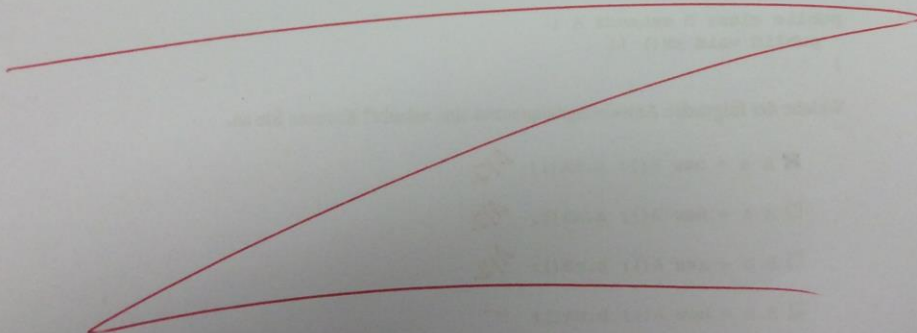
Für jedes $n \in \mathbb{N}_0$ gilt: $n \geq 0$.

Induktionsanfang: $n = 0 \Rightarrow$ $n \geq 0$ gilt, da 0 in \mathbb{N}_0 vorkommt. ✓ 1

Induktionsschluss: $n+1 \stackrel{?}{=} \in \mathbb{N}_0$

$\Rightarrow n+1$ ist ein Element von \mathbb{N}_0 und es gilt auch $n \geq 0$

Wenn die Aussage für $n+1$ gilt, gilt es auch für $n+2$,
und immer weiter bis $n+n$. HJS



(f) Gegeben sei folgender abstrakter Java-Code:

```
// Vorbedingung x1
if (a) Anweisung1;
else Anweisung2;
// Nachbedingung x2
```

Welche Beweisverpflichtungen verlangt das Hoare-Kalkül, um die Korrektheit obigen Programmes zu zeigen? Hinweis: x_1 , x_2 und a sind Bedingungen; Anweisung1 und Anweisung2 sind Anweisungen. Weitere Beweisverpflichtungen, die sich aus den Anweisungen Anweisung1 und Anweisung2 ergeben, brauchen nicht berücksichtigt werden. Hierbei bedeutet Korrektheit, dass nach Ausführung des Programms die Bedingung x_2 gilt, falls vor Ausführung des Programms die Bedingung x_1 gilt.

(g) Gegeben seien die folgenden Java Klassen:

```
public class A {
    public void mA() {}
}

public class B extends A {
    public void mB() {}
}
```

Welche der folgenden Anweisungssequenzen sind erlaubt? Kreuzen Sie an.

- ☒ A a = new B(); a.mA(); $\frac{1}{2}$
- ☐ A a = new B(); a.mB(); $\frac{1}{2}$
- ☐ B b = new A(); b.mB(); $\frac{1}{2}$
- ☒ B b = new A(); b.mA(); $-$
- ☐ A a = new A(); a.mB(); $\frac{1}{2}$
- ☒ B b = new B(); b.mA(); $\frac{1}{2}$

(h) Gegeben sei folgende abstrakte Klasse:

```
public abstract class Vaterklasse {  
    public abstract void foo();  
}
```

Welche der folgenden Implementierungen sind gültig? Kreuzen Sie an.

☒ `public class Kindklasse extends Vaterklasse {
 public void foo() { }`

1/2

☐ `public class Kindklasse extends Vaterklasse {
 public void foodle() { }`

1/2

☐ `public class Kindklasse extends Vaterklasse {
 public void foo(int a) { }`

1/2

☐ `public abstract class Kindklasse extends Vaterklasse {
 public void foo(int a) { }`

—
1,5

Definieren Sie in Java eine Klasse `Kugel`, die eine dreidimensionale geometrische Kugel im Raum modelliert. `Kugel` soll hierbei die Klasse `Punkt3D` sinnvoll verwenden und das Interface `RaeumlichesObjekt` geeignet implementieren.

Des Weiteren soll die Klasse `Kugel` einen geeigneten **Konstruktor** mit sinnvollen Eingabeparametern enthalten. Achten Sie auf eine geeignete Kapselung der Attribute.

Hinweis: Die Oberfläche einer Kugel mit Radius r lässt sich mit Hilfe der Formel $O = 4\pi r^2$ berechnen. Das Volumen einer Kugel mittels der Formel $V = \frac{4}{3}\pi r^3$. Für π können Sie im Programm den `double`-Wert `Math.PI` verwenden.

Zur Umsetzung dürfen **keine vordefinierten** Hilfsmethoden (z.B. aus der Java-API) verwendet werden, Sie dürfen aber bei Bedarf eigene Hilfsklassen und Methoden schreiben.

```
public class Kugel implements RaeumlichesObjekt { 3/9
    private Punkt3D mitte;
    private double radius;
}
public Kugel (Punkt3D mitte, double radius) {
    this.mitte = mitte;
    this.radius = radius;
}

public double getOberflaeche();
return 4 * Math.PI * (radius * radius);
}

public double getVolumen();
return (4/3) * Math.PI * (radius * radius * radius);
}

public Punkt3D getMittelpunkt();
return this.mitte;
}

public void verschiebe (double deltaX, double deltaY, double deltaZ) {
    mitte.x += deltaX;
    mitte.y += deltaY;
    mitte.z += deltaZ;
}
```

Aufgabe 3 **Programmierung**
Java Programmierung

(3+3+2 Punkte)

- (a) Definieren Sie in Java eine statische Methode `anzahlGleich`, die für zwei übergebene Arrays der selben Länge vom Typ `int[]` prüft, an wie vielen Stellen die beiden Arrays gleiche Elemente enthalten und einen entsprechenden `int` Wert zurückgibt. Ein Beispielaufruf könnte wie folgt aussehen:

`anzahlGleich([1, 2, 3, 4], [4, 2, 3, 1])` ergibt 2 als Ergebnis.

Achten Sie auf eine geeignete Fehlerbehandlung für den Fall, dass die übergebene Arrays nicht die gleiche Länge haben.

Zur Umsetzung dürfen **keine** vordefinierten Hilfsmethoden (z.B. aus der Java-API) verwendet werden, Sie dürfen aber bei Bedarf eigene Hilfsklassen und Methoden schreiben.

```
public static int anzahlGleich(int[] a, int[] b) {
```

```
    int erg = 0;
```

```
    for (i = 0; i < a.length; i++) {
```

```
        if if erg++ a[i] == b[i] {
```

```
            erg++;
```

```
        }
```

2

- (b) Definieren Sie in Java eine statische Methode `enthaeltZiffer`, die zwei `int`-Parameter `z` und `n` übergeben bekommt und prüft, ob die Zahl `z` die Ziffer `n` enthält und einen entsprechenden `boolean` Wert zurückgibt. Achten Sie auf eine geeignete Fehlerbehandlung für den Fall, dass `n` keine einstellige Ziffer ist. Beispielaufrufe könnten wie folgt aussehen:

`enthaeltZiffer(123, 3)` ergibt `true` als Ergebnis.

`enthaeltZiffer(123, 7)` ergibt `false` als Ergebnis.

Zur Umsetzung dürfen **keine** vordefinierten Hilfsmethoden (z.B. aus der Java-API) verwendet werden, Sie dürfen aber bei Bedarf eigene Hilfsklassen und Methoden schreiben.


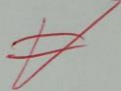
~~`public static boolean enthaeltZiffer(int z, int n) {`~~

~~`if (`~~

- (c) Definieren Sie in Java eine statische Methode `sumQuadrateRek`, die auf **rekursive** Weise die Summe der Quadrate $0^2 + 1^2 + \dots + n^2$ eines übergebenen `int`-Wertes $n \in \mathbb{N}_0$ berechnet und als `int` zurück gibt.

Zur Umsetzung dürfen **keine** vordefinierten Hilfsmethoden (z.B. aus der Java-API) verwendet werden, Sie dürfen aber bei Bedarf eigene Hilfsklassen und Methoden schreiben.

```
public static int sumQuadrateRek (int eingabe) {  
    if (eingabe < 0) {  
        throw new IllegalArgumentException ("Dies ist keine gültige Zahl");  
    }  
    if (eingabe >= 0)  
    {  
        return eingabe * eingabe;  
    }  
}
```



Aufgabe 4 Polymorphismus
Polymorphismus

(6 Punkte)

Gegeben sind die unten stehenden Klassen Hase, Osterhase und Hasenstall, in welchem Hasen und Osterhasen aufeinandertreffen und herumhoppeln.

```
public class Hase {
    public Hase() { }

    public String hopp() {
        return "Spring";
    }

    public String hoppeln(int x) {
        String erg = "Spring";
        for (int i = 0; i < x; i++) {
            erg += this.hopp();
        }
        return erg;
    }
}
```

```
public class Osterhase extends Hase {
    public Osterhase() { }

    public String hopp() {
        return "Hopp";
    }

    public String hoppeln(double x) {
        String erg = "Hopp";
        for (int i = 0; i < x; i++) {
            erg += this.hopp();
        }
        return erg;
    }
}
```

```
1 public class Hasenstall {
2
3     public static void main(String[] args) {
4
5         Hase hasi = new Hase();
6         Osterhase osterhasi = new Osterhase();
7         Hase speziell = new Osterhase();
8
9         System.out.println(hasi.hopp());
10        System.out.println(speziell.hopp());
11
12        System.out.println(hasi.hoppeln(1));
13        System.out.println(osterhasi.hoppeln(1));
14        System.out.println(speziell.hoppeln(1));
15        System.out.println(speziell.hoppeln(1.0));
16    }
17 }
```

Welche Ausgaben liefert das Programm bei Ausführung der Klasse Hasenstall? Betrachten Sie die Konsolenausgaben in den Zeilen 9-15 unabhängig voneinander, das heißt, falls eine Zeile einen Fehler verursacht, sind die anderen Zeilen dennoch so zu beantworten als wäre kein Fehler aufgetreten. Kreuzen Sie für jede dieser Zeilen die entsprechende Ausgabe an.

- | | | | | |
|-----------|----------------------------------------------|--------------------------------------------|--------------------------------------------------|------------------------------------------------|
| Zeile 9: | <input type="checkbox"/> Hopp | <input checked="" type="checkbox"/> Spring | <input type="checkbox"/> Fehler | <input checked="" type="checkbox"/> |
| Zeile 10: | <input checked="" type="checkbox"/> Hopp | <input type="checkbox"/> Spring | <input type="checkbox"/> Fehler | <input checked="" type="checkbox"/> |
| Zeile 12: | <input type="checkbox"/> HoppHopp | <input type="checkbox"/> HoppSpring | <input checked="" type="checkbox"/> SpringSpring | <input type="checkbox"/> SpringHopp |
| | | | | <input type="checkbox"/> Fehler |
| Zeile 13: | <input checked="" type="checkbox"/> HoppHopp | <input type="checkbox"/> HoppSpring | <input type="checkbox"/> SpringSpring | <input type="checkbox"/> SpringHopp |
| | | | | <input checked="" type="checkbox"/> Fehler |
| Zeile 14: | <input type="checkbox"/> HoppHopp | <input type="checkbox"/> HoppSpring | <input type="checkbox"/> SpringSpring | <input checked="" type="checkbox"/> SpringHopp |
| | | | | <input type="checkbox"/> Fehler |
| Zeile 15: | <input checked="" type="checkbox"/> HoppHopp | <input type="checkbox"/> HoppSpring | <input type="checkbox"/> SpringSpring | <input type="checkbox"/> SpringHopp |
| | | | | <input type="checkbox"/> Fehler |

4/6

Aufgabe 5 Osterzeit Typsicherheit

Dekorieren und verstecken Sie Ihr Osternest!

Die folgenden Klassen modellieren verschiedene Typen von Eiern und Dekorationen, um ein Osternest auszuschnücken. Des Weiteren seien unterschiedliche Verstecke gegeben.

```
public class Versteck {}
public class Hecke extends Versteck {}
public class Busch extends Versteck {}
```

```
public class Ei {}
public class Schokoei extends Ei {}
public class Huehnerei extends Ei {}
```

```
public class Dekoration {}
public class Ostergras extends Dekoration {}
public class Stroh extends Dekoration {}
```

Definieren Sie in Java eine Klasse Osternest, die per Typparameter nur Orte mit dem Typ Versteck annimmt. Um das Nest auszuschnücken seien darüber hinaus eine Form von Dekoration und ein Ei mit anzugeben. Achten Sie auf eine geeignete Kapselung der Attribute.

Das Versteck, sowie der Inhalt (Dekoration + Ei) von Osternest sollen ausschließlich im Konstruktor angegeben werden können.

Definieren Sie sich eine Methode suche(), die das Versteck des Osternests zurückgibt. Eine Methode naschen() soll das Ei aus dem Nest zurückgeben.

Beachten Sie, dass ein Nest nur einmal gefunden werden kann und der Inhalt nur dann gegessen werden kann, wenn das Nest bereits gefunden wurde und das Ei noch im Nest liegt. Es liegt nur ein Ei im Korb.

Die Klasse Osternest soll sich mit folgendem Beispielaufrufen verwenden lassen:

```
Osternest<Busch, Schokoei, Ostergras> nest =
    new Osternest<Busch, Schokoei, Ostergras>(new Busch(),
                                                new Schokoei(),
                                                new Ostergras());
```

```
Versteck v = nest.suche();
Ei ei = nest.naschen();
```

Zur Umsetzung dürfen **keine** vordefinierten Hilfsmethoden (z.B. aus der Java-API) verwendet werden, Sie dürfen aber bei Bedarf eigene Hilfsklassen und Methoden schreiben.

```
public class Osternest<H extends Hecke> (H extends Busch) {
    private H hecke;
    private Ei ei;
    public Osternest (H hecke, H busch, Ei s, Dekoration o)
    this.hecke = h;
    this.busch = b;
}
```

~~public class Osterneest~~ ^{< V extends Versteck, E extends Ei} ~~extends Versteck~~ ~~extends Busch~~ [>] { ^{0,5}
~~private H heute;~~
~~private B busch;~~
~~public Osterneest (H heute, B busch, Ei ei, Dekoration dekoration);~~
~~this.heute = heute;~~ ^{this.versteck = versteck;}
~~this.busch = busch;~~ ^{this.ei = ei;}
~~versteck v = v;~~

^V
~~public suche();~~
~~{~~
~~if (v == null) {~~
~~v = nest;~~
~~nest == null;~~
~~return nest;~~
~~}~~
~~return null;~~
~~}~~

^{E Folgefehler}
~~public maschen();~~
~~{~~
~~if (nest != null) {~~
~~if (ei != null) {~~
~~return maschen;~~
~~}~~
~~return null;~~
~~}~~

Aufgabe 6 Einfach-verkettete Liste
Datenstrukturen

(6 Punkte)

Gegeben ist eine Klasse `Entry<T>`, die die Elemente einer typisierten Liste implementiert.

```
public class Entry<T> {  
    /**  
     * Eigentliches Element  
     */  
    private T element;  
    /**  
     * Verweis auf das naechste Element  
     */  
    private Entry<T> next;  
    /**  
     * Erzeugt und initialisiert ein Listen-Element  
     * @param o      Wert fuer das eigentliche Element  
     * @param next   Wert fuer den Verweis auf das naechste Element  
     */  
    public Entry(T o, Entry<T> next) {  
        this.element = o;  
        this.next = next;  
    }  
    /**  
     * Liefert den Wert fuer das eigentliche Element zurueck  
     * @return Wert fuer das eigentliche Element  
     */  
    public T getElement() {  
        return this.element;  
    }  
    /**  
     * Weist den Wert fuer das eigentliche Element zu  
     * @param element Wert fuer das eigentliche Element  
     */  
    public void setElement(T element) {  
        this.element = element;  
    }  
    /**  
     * Liefert den Wert fuer den Verweis auf das naechste Element zurueck  
     * @return Wert fuer den Verweis auf das naechste Element  
     */  
    public Entry<T> getNext() {  
        return this.next;  
    }  
    /**  
     * Weist den Wert fuer den Verweis auf das naechste Element zu  
     * @param next   Wert fuer den Verweis auf das naechste Element  
     */  
    public void setNext(Entry<T> next) {  
        this.next = next;  
    }  
}
```

Des Weiteren ist eine Klasse `Liste` gegeben, die eine **einfach-verkettete** Liste realisiert.

```
public class Liste<T> {  
    /**  
     * Anzahl der Elemente in der Liste  
     */  
    private int size;  
    /**  
     * Verweis auf das erste Element der Liste  
     */  
    private Entry<T> firstEntry;  
  
    /**  
     * Erzeugt eine leere Liste  
     */  
    public Liste() {  
        this.firstEntry = null;  
    }  
  
    /**  
     * Liefert die Anzahl der Elemente in der Liste zurueck  
     * @return Anzahl der Elemente in der Liste  
     */  
    public int getSize() {  
        return this.size;  
    }  
}
```

Erweitern Sie die Klasse `Liste` um eine Methode

```
public T removeAt(int k),
```

die den Eintrag an der Stelle mit Index k aus der Liste entfernt und das entsprechende Element als Ergebnis zurückliefert. Achten Sie auf sinnvolle Fehlerbehandlungen, falls nötig.
Hinweis: Beachten Sie, dass das Element an erster Stelle der Liste den Index 0 hat.

```
public T removeAt(int k) {
```