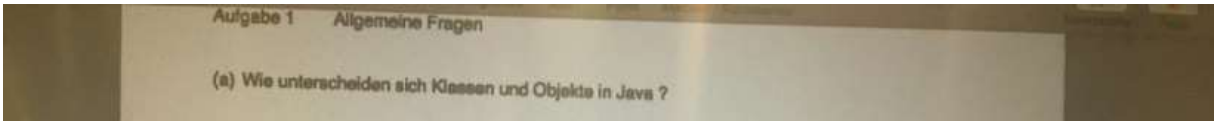
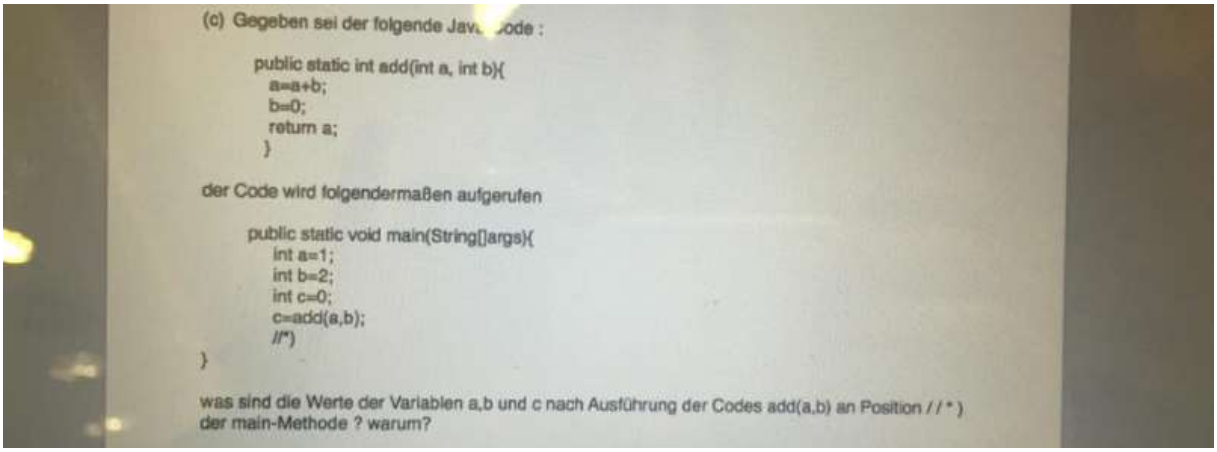
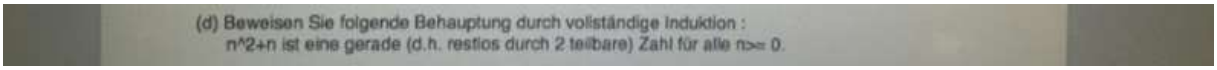


Eine Klasse ist ein abstrakter „Bauplan“, nach dem ein spezifisches Objekt dieser Klasse gebildet werden kann. Beispielsweise wäre das Blatt, auf dem diese Klausur steht, ein Objekt der abstrakten Klasse Papier, das verwendet werden kann (Indem man zB darauf schreibt), während auf der Klasse (= „Idee“) Papier, der abstrakten Version dieses Blattes, nicht geschrieben werden kann.

Der Konstruktor einer Klasse kann verwendet werden, um Eigenschaften zu initialisieren oder andere Konstruktoren aufzurufen, die diese Aufgabe übernehmen.



a = 1, da a in der main-Methode niemals verändert wird und die gleichnamige Variable aus add() ebendort definiert wird und dadurch nur dort existiert.
b = 2, Begründung siehe a.
c = 3, da c durch die Zeile c = add(a, b) den Wert 1+2 zugewiesen bekommt.



Behauptung: $(n^2 + n) \% 2 = 0$ für alle $n >= 0$

Induktionsanfang: n = 0:

$0^2 + 0 = 0$
$0 \% 2 = 0$
→ 0 ist durch 2 teilbar

Induktionsschritt: n = a + 1 unter der Bedingung, dass die Aussage $(n^2 + n) \% 2 = 0$ für n = a gilt.

Zu beweisen: $((a + 1)^2 + (a + 1)) \% 2 = 0$, wenn $(a^2 + a) \% 2 = 0$
$(a^2 + 2a + 1 + a + 1) \% 2 = (a^2 + a + 2a + 2) \% 2 = (a^2 + a) \% 2 + (2a + 2) \% 2$
$(a^2 + a) \% 2 = 0$ (Siehe Induktionsbedingung), damit gilt
$(a^2 + a) \% 2 + (2a + 2) \% 2 = (2a + 2) \% 2$
$(2a + 2) \% 2 = 0$, da $(2a + 2) \% 2 = 2(a + 1) \% 2$, was immer null ist.
→ Aus $(a^2 + a) \% 2 = 0$ folgt $((a + 1)^2 + (a + 1)) \% 2 = 0$;

QED.

Name:

(5 Punkte)

Aufgabe 2 Eigenschaften von Algorithmen

Eigenschaften von Algorithmen

Gegeben sind vier verschiedene Algorithmen `test(int zahl, int[] liste)`, die jeweils überprüfen, ob die Zahl `zahl` in dem Array `liste` vorkommt oder nicht. Sie geben den entsprechenden booleschen Wert **`true`** oder **`false`** zurück. Sie können davon ausgehen, dass das Array `liste` initial nicht leer ist.

Die vordefinierte Methode `random(int max)` gibt einen zufälligen Wert zwischen `0` und `max-1` zurück.

Beantworten Sie die Fragen zu den jeweiligen Algorithmen. Eine richtige Antwort bedeutet 0,5 Punkte, eine falsche Antwort bedeutet 0,5 Punkte Abzug. Eine unbeantwortete Frage bedeutet 0 Punkte. Sie können in dieser Aufgabe maximal 5 Punkte und minimal 0 Punkte erreichen.

(a) Gegeben sei der folgende Algorithmus A:

```
public static boolean test(int zahl, int[] liste) {
    for (int i=0; i<liste.length; i++) {
        if (liste[i] == zahl) {
            return true;
        }
    }
    return false;
}
```

Ist Algorithmus A **total korrekt**? Ja ☒ Nein ☐ ✓

Ist Algorithmus A **deterministisch**? Ja ☒ Nein ☑ —

(b) Gegeben sei der folgende Algorithmus B:

```
public static boolean test(int zahl, int[] liste) {
    int i = random(liste.length);
    return liste[i] == zahl;
}
```

Ist Algorithmus B **determiniert**? Ja ☑ Nein ☒ —

Ist Algorithmus B **partiell korrekt**? Ja ☐ Nein ☒ ✓

Ist Algorithmus B **deterministisch**? Ja ☐ Nein ☒ ✓

(c) Gegeben sei der folgende Algorithmus C:

```
public static boolean test(int zahl, int[] liste) {
    while (true) {
        int i = random(liste.length);
        if (liste[i] == zahl) {
            return true;
        }
    }
}
```

Ist Algorithmus C **terminierend**? Ja ☑ Nein ☐

Ist Algorithmus C **partiell korrekt**? Ja ☐ Nein ☐

(d) Gegeben sei der folgende Algorithmus D:

```
public static boolean test(int zahl, int[] liste) {
    if (liste.length == 0) {
        return false;
    }

    int i = random(liste.length);
    if (liste[i] == zahl) {
        return true;
    } else {
        int[] neueListe = new int[liste.length - 1];
        for (int a = 0; a < i; a++) {
            neueListe[a] = liste[a];
        }
        for (int a = i + 1; a < liste.length; a++) {
            neueListe[a - 1] = liste[a];
        }
        return test(zahl, neueListe);
    }
}
```

Ist Algorithmus D **terminierend**? Ja ☐ Nein ☐

Ist Algorithmus D **total korrekt**? Ja ☐ Nein ☐

Ist Algorithmus D **determiniert**? Ja ☐ Nein ☑

(9 Punkte)

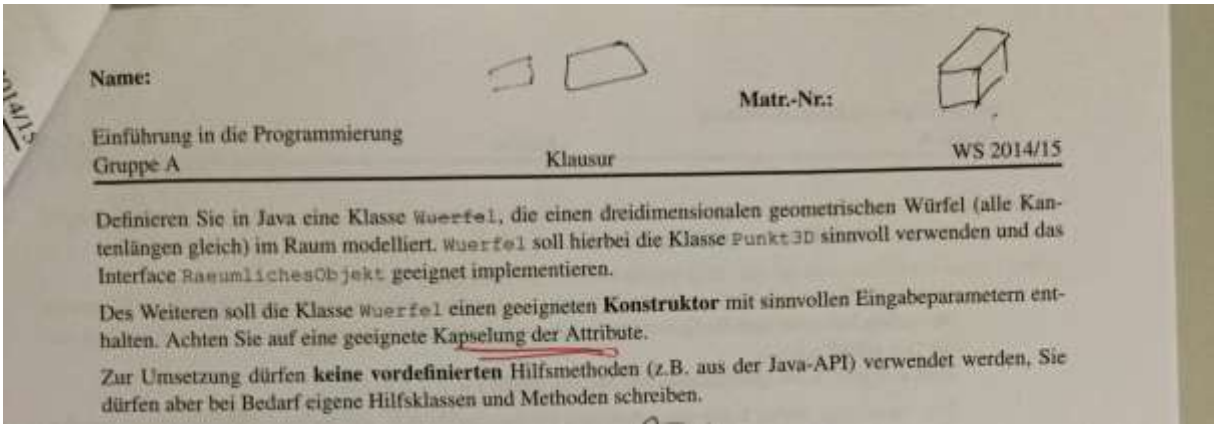
Aufgabe 3 Würfel

Modellierung

Gegeben ist die Implementierung eines Punktes im dreidimensionalen Raum sowie ein Interface für räumliche Objekte:

```
1  public class Punkt3D {
2          private double xCoord;
3          private double yCoord;
4          private double zCoord;
5
6          public Punkt3D(double x, double y, double z) {
7                  this.xCoord = x;
8                  this.yCoord = y;
9                  this.zCoord = z;
10         }
11
12         public double getX() {
13                 return this.xCoord;
14         }
15
16         public double getY() {
17                 return this.yCoord;
18         }
19
20         public double getZ() {
21                 return this.zCoord;
22         }
23
24         public void verschiebe(double deltaX, double deltaY, double deltaZ) {
25                 this.xCoord += deltaX;
26                 this.yCoord += deltaY;
27                 this.zCoord += deltaZ;
28         }
29 }
```

```
1  public interface RaeumlichesObjekt {
2          /**
3           * Berechnet die Oberfläche des räumlichen Objekts,
4           * also die Summe aller Seitenflächen.
5           */
6          public double getOberflaeche();
7          /**
8           * Berechnet den Volumeninhalt des räumlichen Objekts.
9           */
10         public double getVolumen();
11         /**
12          * Gibt den räumlichen Mittelpunkt des Objekts als Punkt3D.
13          */
14         public Punkt3D getMittelpunkt();
15         /**
16          * Verschiebt das Objekt um die Differenzwerte x, y und z im Raum.
17          */
18         public void verschiebe(double deltaX, double deltaY, double deltaZ);
19 }
```



Name: Matr.-Nr.:

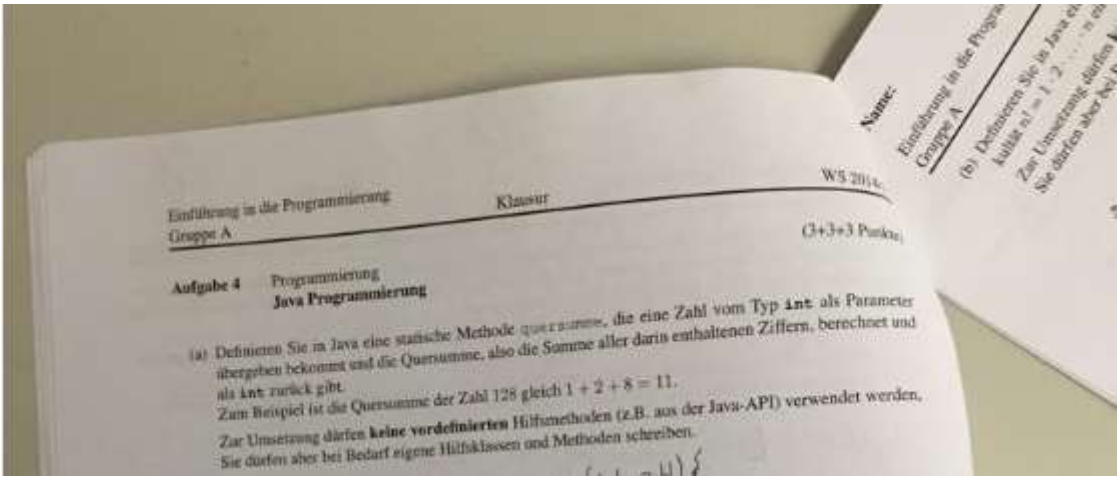
Einführung in die Programmierung
Gruppe A Klausur WS 2014/15

Definieren Sie in Java eine Klasse `Wuerfel`, die einen dreidimensionalen geometrischen Würfel (alle Kantenlängen gleich) im Raum modelliert. `Wuerfel` soll hierbei die Klasse `Punkt3D` sinnvoll verwenden und das Interface `RaeumlichesObjekt` geeignet implementieren.

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

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 Wuerfel implements RaeumlichesObjekt
{
	Punkt3D mitte;
	double kante;

	public Wuerfel(Punkt 3D mitte, double kante)
	{
		this.mitte = mitte;
		this.kante = kante;
	}

	public double getOberflaeche()
	{
		return 6*a*a;
	}

	public double getVolumen()
	{
		return a*a*a;
	}

	public Punkt3D getMittelpunkt()
	{
		return mitte;
	}

	public void verschiebe(double deltaX, double deltaY, double deltaZ)
	{
		mitte.verschiebe(deltaX, deltaY, deltaZ);
	}
}
```



Einführung in die Programmierung Klausur WS 2014/
Gruppe A

Name:

Einführung in die Programmierung
Gruppe A

Aufgabe 4 Programmierung
Java Programmierung

(3+3+3 Punkte)

(a) Definieren Sie in Java eine statische Methode quersumme, die eine Zahl vom Typ `int` als Parameter übergeben bekommt und die Quersumme, also die Summe aller darin enthaltenen Ziffern, berechnet und als `int` zurück gibt.
Zum Beispiel ist die Quersumme der Zahl 128 gleich 1 + 2 + 8 = 11.
Zur Umsetzung dürfen **keine vordefinierten** Hilfsmethoden (z.B. aus der Java-API) verwendet werden. Sie dürfen aber bei Bedarf eigene Hilfsklassen und Methoden schreiben.

(b) Definieren Sie in Java ei
kalität n! = 1 · 2 · ... · n

Zur Umsetzung dürfen
Sie dürfen aber bei

Lösung 1 (zahl als Integer, iterativ)

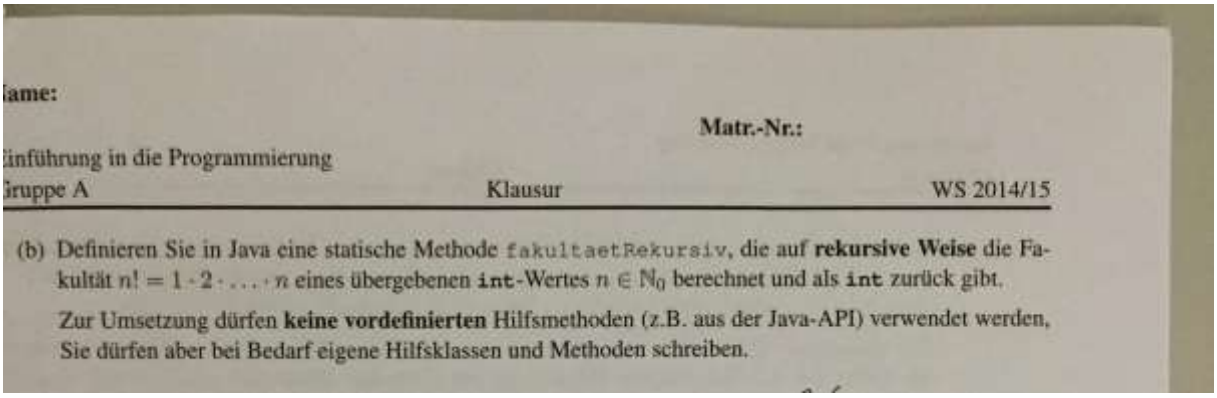
```
static int quersumme(int zahl)
{
	int q = 0;

	while(zahl > 0)
	{
		q += (zahl – 10*(zahl/10));
		zahl =  zahl/10;
	}

	return q;
}
```

Lösung 2 (zahl als Integer, rekursiv)

```
static int quersumme(int zahl)
{
	if(zahl < 10)
	{
		return zahl;
	}

	return (zahl – 10*(zahl/10)) + quersumme(zahl/10);
}
```

Lösung 3 (zahl als String, iterativ)

```
static int quersumme(int zahl)
{
	String z = zahl + „“;
	int q = 0;
	while(z.length() > 0)
	{
		q += Integer.parseInt(z.substring(z.length – 1));
		z = z.substring(0, z.length – 1);
	}

	return q;
}
```



lame:

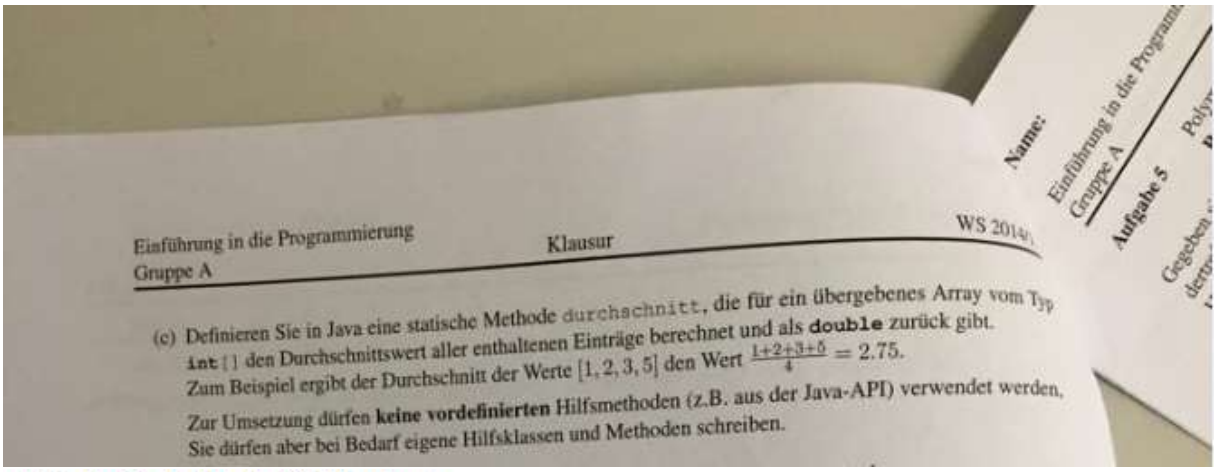
Matr.-Nr.:

inführung in die Programmierung
ruppe A | Klausur | WS 2014/15

(b) Definieren Sie in Java eine statische Methode `fakultaetRekursiv`, die auf **rekursive Weise** die Fakultät $n! = 1 \cdot 2 \cdot \ldots \cdot n$ 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.

```
static int fac(int i)
{
	if(i == 0)
	{
		return 1;
	}

	return i * fac(i – 1);
}
```



Name:

Einführung in die Programmierung
Gruppe A

Aufgabe 5

Einführung in die Programmierung | Klausur | WS 2014/

Gruppe A

(c) Definieren Sie in Java eine statische Methode `durchschnitt`, die für ein übergebenes Array vom Typ **`int`**`[]` den Durchschnittswert aller enthaltenen Einträge berechnet und als **`double`** zurück gibt.
Zum Beispiel ergibt der Durchschnitt der Werte $[1, 2, 3, 5]$ den Wert $\frac{1+2+3+5}{4} = 2.75$.

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

```
static double durchschnitt(int[] values)
{
	double d = 0;

	for(int v : values)
	{
		d += v;
	}

	return d / values.length;
}
```

Name: Matr.-Nr.:

Einführung in die Programmierung
Gruppe A — Klausur — WS 2014/15

Aufgabe 5 Polymorphismus (5 Punkte)

Polymorphismus

Gegeben sind die unten stehenden Klassen `Hund`, `Fuchs` und `Park`, in welchem Hunde und Füchse aufeinandertreffen und miteinander spielen.
Hinweis: Ein Fuchs ist ein Vertreter der Familie der Hunde.

```
public class Hund {

  public Hund(){

  }

  public String bellen(){
    return "Wuff";
  }

  public String spielen(Hund h) {
    return "Wuff " + h.bellen();
  }

}
```

```
public class Fuchs extends Hund {

  public Fuchs(){

  }

  public String bellen(){
    return "Ringding";
  }

  public String spielen(Fuchs f) {
    return "Ringding " + f.bellen();
  }

}
```

```
1  public class Park {
2
3    public static void main(String[] args) {
4
5      Hund bello = new Hund();
6      Fuchs foxi = new Fuchs();
7      Hund hybrid = new Fuchs();
8
9      System.out.println(bello.bellen());
10     System.out.println(hybrid.bellen());
11
12     System.out.println(bello.spielen(foxi));
13     System.out.println(hybrid.spielen(foxi));
14     System.out.println(foxi.spielen(hybrid));
15   }
16 }
```

Welche Ausgaben liefert das Programm bei Ausführung der Klasse `Park`? Kreuzen Sie die passenden Felder an. Im Fehlerfall ist keine Option anzukreuzen.

Zeile 9: ☒ Wuff ☐ Ringding ✓

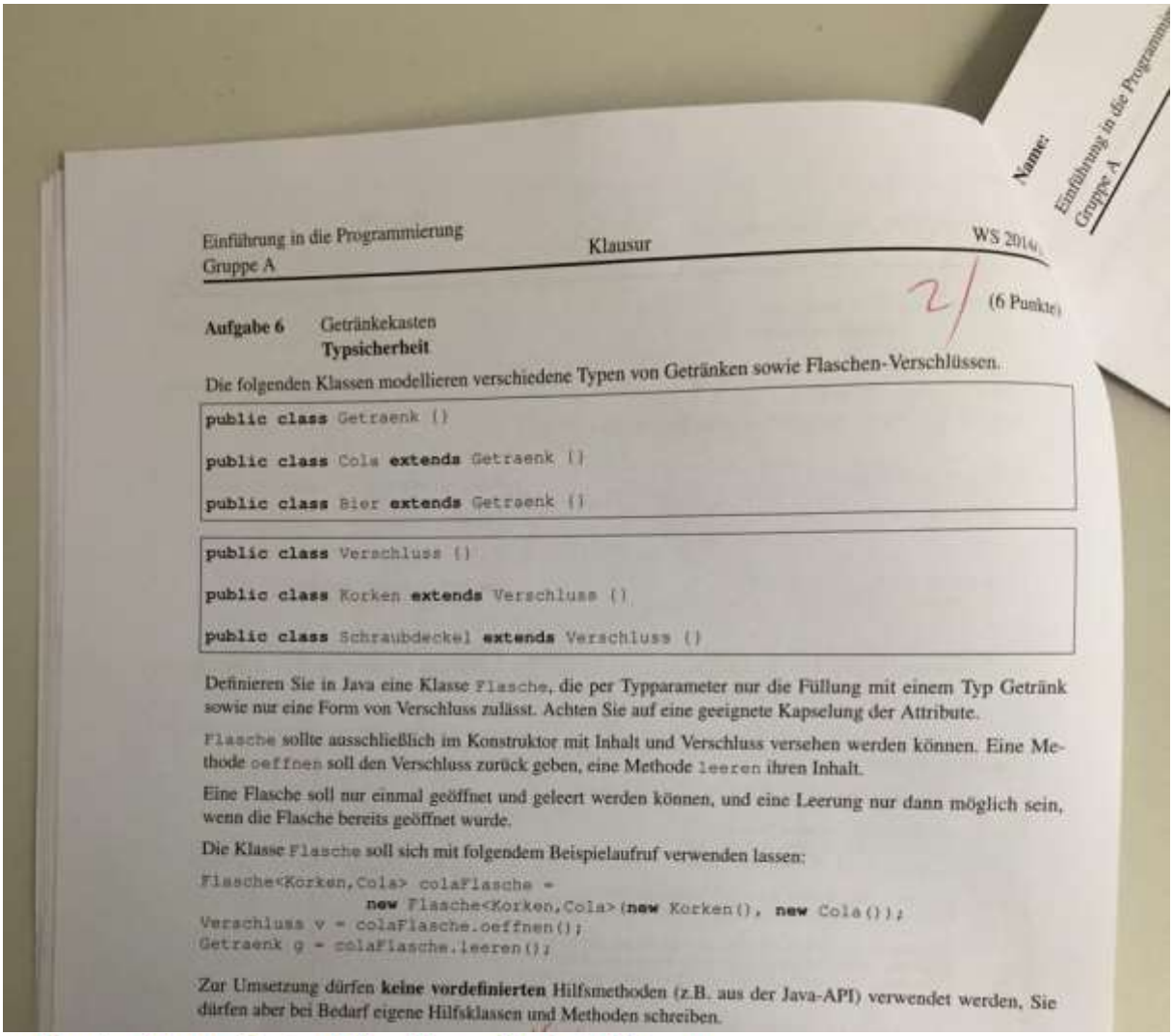
Zeile 10: ☐ Wuff ☒ Ringding ✓

Zeile 12: ☒ Wuff Ringding ☐ Wuff Wuff ☐ Ringding Wuff ☐ Ringding Ringding ✓

Zeile 13: ☒ Wuff Ringding ☐ Wuff Wuff ☐ Ringding Wuff ☑ Ringding Ringding —

Zeile 14: ☒ Wuff Ringding ☐ Wuff Wuff ☐ Ringding Wuff ☑ Ringding Ringding —

3 Pkt.



Name:

Einführung in die Programmierung
Gruppe A

Einführung in die Programmierung Klausur WS 2014
Gruppe A

2 / (6 Punkte)

Aufgabe 6 Getränkekasten
Typsicherheit

Die folgenden Klassen modellieren verschiedene Typen von Getränken sowie Flaschen-Verschlüssen.

```
public class Getraenk {}

public class Cola extends Getraenk {}

public class Bier extends Getraenk {}
```

```
public class Verschluss {}

public class Korken extends Verschluss {}

public class Schraubdeckel extends Verschluss {}
```

Definieren Sie in Java eine Klasse `Flasche`, die per Typparameter nur die Füllung mit einem Typ Getränk sowie nur eine Form von Verschluss zulässt. Achten Sie auf eine geeignete Kapselung der Attribute.

`Flasche` sollte ausschließlich im Konstruktor mit Inhalt und Verschluss versehen werden können. Eine Methode `oeffnen` soll den Verschluss zurück geben, eine Methode `leeren` ihren Inhalt.

Eine Flasche soll nur einmal geöffnet und geleert werden können, und eine Leerung nur dann möglich sein, wenn die Flasche bereits geöffnet wurde.

Die Klasse `Flasche` soll sich mit folgendem Beispielaufruf verwenden lassen:

```
Flasche<Korken,Cola> colaFlasche =
                    new Flasche<Korken,Cola>(new Korken(), new Cola());
Verschluss v = colaFlasche.oeffnen();
Getraenk g = colaFlasche.leeren();
```

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 Flasche<V extends Verschluss><G extends Getraenk>
{
	private V verschluss;
	private G getraenk;

	public Flasche(V verschluss, G getraenk)
	{
		this.verschluss = verschluss;
		this.getraenk = getraenk;
	}

	public V oeffnen()
	{
		if(verschluss != null)
		{
			V v = verschluss;
			verschluss = null;
			return v;
		}

		return null;
	}
```

```
	public G leeren()
	{
		if(verschluss == null && getraenk != null)
		{
			G g = getraenk;
			getraenk = null;
			return g;
		}

		return null;
	}
}
```

(6 Punkte)

Aufgabe 7 Einfach-verkettete Liste
Datenstrukturen

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;
    }
}
```

Name:

Matr.-Nr.:

Einführung in die Programmierung
Gruppe A | Klausur | WS 2014/15

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 void addAt(T element, int k).
```

die das übergebene Element `element` an der `k`-ten Stelle der Liste einfügt. Achten Sie dabei auf sinnvolle Fehlerbehandlungen, falls nötig.
Hinweis: Beachten Sie, dass das Element an erster Stelle der Liste den Index 0 hat.

```
public void addAt(T element, int k)
{
	Entry<T> e = new Entry(element, null);

	if(k == 0)
	{
		e.setNext(this.firstEntry);
		this.firstEntry = e;
		size = 1;
	}
	else
	{
		int i;
		Entry<T> c = this.firstEntry;

		try
		{
			for(i = 1; i < k; k++)
			{
				c = c.getNext();
			}

			e.setNext(c.getNext());
			c.setNext(e);
			size++;
		}
		catch(NullPointerException e)
		{
			Entry<T> c = this.firstEntry;

			while(c.getNext() != null)
			{
				c = c.getNext();
			}

			c.setNext(e);
		}
	}

	try
	{
		size++;
	}
	catch(Exception e)
	{
		size = 0;
	}
}
```