1. (16 Punkte)

Es sei die folgende (unvollständige) Klasse gegeben:

```
import java.util.Collection;
import java.util.Iterator;
public class IteratorFuncs {
    public static void printContents(Collection<String> col) {
        // Diese Methode muss ausimplementiert werden
    }
}
```

Die `printContents`-Methode, ist nicht ausimplementiert. Ihre Aufgabe soll aber sein, alle Elemente der gegebenen Collection auf der Konsole auszugeben. Implementieren Sie die `printContents`-Methode auf zwei unterschiedliche Weisen:

(a) Einmal, indem Sie sich eine Instanz von `Iterator` besorgen indem Sie die Methode `iterator()` von `Collection` aufrufen. Diesen Iterator benutzen Sie dann um über den Inhalt von `col` in einer Schleife zu iterieren.

(b) Und noch einmal, indem Sie möglichst elegant mit einer for-Schleife über den Inhalt von `col` iterieren.

Achten Sie in allen Fällen auf korrekte Anwendung generischer Konstrukte!

(a)
```
public static void printContents(Collection<String> col){
    Iterator<String> i = col.iterator();
    while(i.hasNext()){
        String s = i.next();
        System.out.println($s);
    }
}
```

b)
```
public static void printContents(Collection<String> col){
    for(String s : col){
        System.out.println(s);
    }
}
```

16/16

(16 Punkte)

2. Es sei die folgende Methode gegeben:

```
public int flaechenInhalt(int breite, int hoehe) {
    return breite * hoehe;
}
```

Die Methode berechnet den Flächeninhalt eines Rechtecks. Nun ist es relativ sinnlos, für ein Rechteck negative Seitenlängen zu spezifizieren. Insofern sollte der Code eigentlich daraufhin prüfen, ob `breite` oder `hoehe` negativ sind. Ergänzen Sie den Code, indem Sie

(a) Eine passende, sinnvolle Klasse definieren, die im Fehlerfall als Ausnahme geworfen werden kann. Fügen Sie der Klasse Attribute hinzu, die über den konkret aufgetretenen Fehlergrund Auskunft erteilen können. Fügen Sie einen Konstruktor hinzu, mit dem diese Attribute initialisiert werden. Überschreiben Sie die `getMessage()`-Methode, so dass die Ausnahme nach einer möglichst Aussagekräftigen Fehlermeldung befragt werden kann.

(b) Modifizieren Sie die Methode `flaechenInhalt`, so dass bei Eingabe eines negativen Parameters eine entsprechende Ausnahme erzeugt wird. Sorgen Sie dafür, dass die Methode dabei syntaktisch korrekt bleibt!

14/16

a)
```
public class NegativArgumentException extends Exception {
    private int errorData;
    public NegativArgumentException(int data){
        errorData = data;
    }
    public String getMessage(){
        return ("Fehler! Angegebener Parameter: " + errorData +
                " negativ! Nur positive Werte erlaubt!");
    }
}
```

b) Die Klasse ist zwar schon allgemein einsetzbar, beim Umgang mit (Höhe, Breite)-Paaren könnte aber auch die Information über den jeweils anderen Wert hilfreich sein. (-1)

b)
```
public int flaechenInhalt(int breite, int hoehe) throws
                                    NegativArgumentException {
    if (breite < 0){
        NegativArgumentException nae =
                new NegativArgumentException(breite);
        throw nae;
    }
    if(hoehe < 0){
        NegativArgumentException e =
                new NegativArgumentException(hoehe);
        throw e;
    }
    return breite * hoehe;
}
```

3. (5 Punkte)

Es sei die folgende Klasse gegeben:

```
class Zylinder {
    public int radius;
    public int hoehe;
    public boolean equals(Object other) {
        return this.hashCode() == other.hashCode();
    }
}
```

Inwieweit halten Sie die Methode `equals` dieser Klasse für sinnvoll? Was ist sinnvoll, was nicht und warum? Begründen Sie Ihre Aussage möglichst genau und verständlich!

3/5

Die equals-Methode ist nicht ~~sinnfo~~ sinnvoll, da die Methode hashCode() einen Integer zurückliefert. Es ist allerdings nicht möglich allein mit der Mächtigkeit von Integer jedes Objekt eindeutig zu referenzieren, da es deutlich mehr mögliche Objekte als Zahlen im Wertebereich von Integer gibt. Dadurch kann möglicherweise eine Variable der Klasse A den gleichen Hashcode wie eine Variable der Klasse B haben, wobei A und B sicher nicht die gleichen Objekte sind. (Also $A \neq B$)

Eindeutigkeit zum Hashcode von Object?

Es sollten deshalb noch weitere Identifikationsmerkmale überprüft werden (z.B. Attribute, die Klasse über .instanceof oder getClass())

4. (12 Punkte)

(a) Ist Java eine compilierte oder eine interpretierte Sprache? Oder keines von beidem? Oder gar beides?

(b) Welche Vor- und Nachteile hat die im Java-Konzept verwendete Vorgehensweise um vom Programmcode zur Programmausführung zu kommen?

das Java-Programm

a) Java ist beides. Zunächst wird es mithilfe eines Compilers in einen Zwischencode (auch Bytecode genannt) übersetzt. Dieser Zwischencode liegt in ~~der .class Datei~~ des einer Datei mit der Endung .class. ~~Aus dieser Datei liest~~ Den Code dieser Datei führt nun der ein Interpreter (die Java Virtual Machine) aus.

b) Java möchte sowohl plattformunabhängig als auch schnell sein. ~~Die Schnelligkeit~~ Da Compiler schnell aber maschinenabhängig und Interpreter maschinenunabhängig aber langsam sind, musste ein Kompromiss gefunden werden. Selbiger ist in a) beschrieben. So ist Java sowohl schneller als rein interpretierte Programmiersprachen, als auch plattformunabhängig.

12/12

5. (14 Punkte)

Es sei die folgende Methode gegeben:

```
static Object getLastElemEqualTo(List li, Object elem) {
    if (elem == null) return null;
    Object last = null;
    for (Object ob : li) {
        if (elem.equals(ob)) last = ob;
    }
    return last;
}
```

Die Methode ist insofern etwas unschön, als sie zum einen den Inhalt der Argumentliste `li` nicht näher spezifiziert und zum anderen sehr allgemein immer einen Wert des Typs `Object` zurückgibt. Schreiben Sie eine **generische** Methode, die ganz ähnlich wie die gegebene funktioniert, in der aber dafür gesorgt wird, dass in der Liste `li` ausschließlich Objekte einer beim Aufruf der Methode bestimmbaren Klasse enthalten sind. Außerdem soll die Methode als Rückgabetyp auch nicht `Object` besitzen, sondern der Rückgabetyp soll dem Typ der Listenelemente entsprechen. Sorgen Sie zusätzlich dafür, dass der beim Aufruf bestimmbare generische Typ immer eine Unterklasse von `Number` sein muss.

```
static <T extends Number> T getLastElemEqualTo(List<T> li, T elem){
    if (elem == null) return null;
    T last = null;
    for (T ob : li){
        if (elem.equals(ob)) last = ob;
    }
    return last;
}
```

14/14

6. (8 Punkte)

Schreiben Sie eine Methode, die eine (nach Möglichkeit generische) Instanz vom Typ `List` erzeugt und zurückgibt, die drei Elemente vom Typ `Integer` enthält.

```
static public List<Integer> generateIntList(){
    List<Integer> li = new LinkedList<Integer>();
    li.add(400);
    li.add(7235);
    li.add(17000);
    return li;
}
```

8/8

7. (10 Punkte)

Gegeben sei die folgende Klasse:

```
1     class Bla {
2         static boolean contains(int[] buf, int val) {
3            return binSearch(buf, val, 0, buf.length - 1);
4         }
5
6         static boolean binSearch(int[] buf, int val, int left, int right) {
7             if (left > right) return false;
8             int m = (left + right) / 2;
9             if (m < 0 || m >= buf.length)
10                thrown new RuntimeException("Illegal index: " + m);
11            if (buf[m] < 0)
12                throw new IllegalArgumentException("No negative values!");
13            if (buf[m] == val) return true;
14            else {
15                if (binSearch(buf, val, left, m - 1)) return true;
16                if (binSearch(buf, val, m + 1, right)) return true;
17            }
18            return false;
19        }
20
21
22        public static void main(String[] a) {
23            contains(new int[]{-3, 5, 9, 15}, 7);
24        }
25    }
```

Wenn man sie startet, wird dann eine Ausnahme erzeugt? Falls ja, geben Sie die Stack-Trace an. Dabei ist der ganz genaue Wortlaut der Stack-Trace nicht entscheidend, wichtig sind die Zeilennummern, die Methode in der eine Weiterleitung der Ausnahme jeweils erfolgt sowie die eigentliche Fehlermeldung. Achten Sie auf die Reihenfolge, in der die einzelnen Zeilen der Stack-Trace ausgegeben werden!

IllegalArgumentException : No negative values!
at Bla.binSearch 12
at Bla.binSearch 15
at Bla.contains 3
at Bla.main 23

✓

10/10

8. (14 Punkte)

Gegeben seien die folgende unvollständige Klasse:

```
public class MethodOps {
    /**
     * Ein Interface für eine einstellige Funktion auf
     * rationalen Zahlen. Dabei wird die Eingabe x auf
     * den jeweiligen Funktionswert abgebildet. Auf welchen
     * konkreten Funktionswert dabei abgebildet wird, ist
     * vom Programmierer der konkreten Klasse zu bestimmen.
     */
    interface Function {
        public double getValue(double x);
    }

    static Function getAdditionFunc(Function f, Function g) {
        // Diese Methode muss ausimplementiert werden!
    }
}
```

In der Klasse `MethodOps` ist das Interface `Function` für ganz allgemeine Funktionen gegeben. In Funktionen werden ja immer Werte (in diesem Fall vom Typ `double`) auf andere Werte abgebildet. Z.B. bildet die Funktion $f_1(x) \stackrel{\text{def}}{=} x+1$ alle Zahlen auf den um eins größeren Wert ab. Zur Berechnung des Bilds von x steht dabei im Interface die Methode `getValue` zur Verfügung. Eine Klasse, die die oben beschriebene Funktion f_1 mit Hilfe des Interface ausimplementiert, könnte also wie folgt aussehen:

```
class f1 implements Function {
    public double getValue(double x) {
        return x + 1;
    }
}
```

Vervollständigen Sie die Klasse `MethodOps`, indem Sie die Methode `getAdditionFunc` ausimplementieren. Sie soll das folgende leisten: Auf Eingabe der beiden Funktionen f und g soll eine neue Funktion zurückgegeben werden, die jedes x im Wertebereich von `double` auf den Wert $f(x) + g(x)$ abbildet. Benutzen Sie in Ihrer Implementation eine anonyme innere Klasse; insbesondere dürfen Sie keinen zusätzlichen Code außerhalb der Methode `getAdditionFunc` schreiben.

```
static Function getAdditionFunc(Function f, Function g) {
    return ( new Function() {
        public double getValue(double x) {
            double erg;
            erg = f.getValue(x);
            erg = erg + g.getValue(x);   // Rückgabe von erg
        }
    });
}
```

Sie können hier keine nichtfinalen, lokalen Variablen einsetzen!

11/14