Probeklausur
SS 2018
Hochschule Zittau/Görlitz,
Prüfer: Prof. Dr. Georg Ringwelski

Objekt-Orientierte Programmierung

II/Wb17

Name:

Matrikelnummer:

Punkte:

1	2	3	4	5	Gesamt
/30	/25	/25	/20	/20	/120

Spielregeln:

- Die Bearbeitungszeit beträgt **120 Minuten** ab Ausgabe der Klausur
- Die maximale Punktzahl ist 120, für jede Minute einen. Achten Sie bei der Bearbeitung darauf die **Zeit pro Aufgabe** an den zu erreichenden Punkten auszurichten.
- Abzugeben ist **ausschließlich das ausgegebene Papier**. Lösungen werden direkt unter die Fragen und ggf. auf Rückseiten der Blätter geschrieben. Lösungen auf eigenem Papier werden nicht bewertet.
- Zur Bearbeitung sind **dokumentenechte Stifte** zu verwenden (kein Bleistift, Füller o.ä.)
- Wer die **Prüfung verlässt** um zur Toilette zu gehen, gibt währenddessen seine Klausur ab. Es kann immer nur eine Person gehen.
- Jeder **Betrugsversuch** führt zum sofortigen Ende der Klausur für alle Beteiligten und deren Beurteilung mit der Note 5.
- Die Klausur besteht aus **6 Seiten, prüfen Sie den vollständigen Erhalt**. Alle Seiten sind abzugeben und Name und Matrikelnummer sind auf jedem separaten Blatt einzutragen.
- Es sind **keine elektrischen Hilfsmittel** zugelassen! (Hilfsmittel aus Papier (Skripte etc) sind erlaubt)

Viel Erfolg !

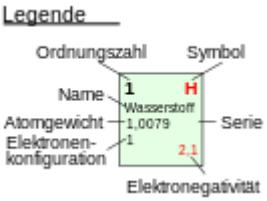
Name: MatNr:

1. Aufzählungstypen, Sichtbarkeit und Dokumentation

a) (20 Punkte) Definieren Sie einen Aufzählungstyp in Java für chemische Elemente, dessen Werte die unten dargestellten chemischen Symbole sind. Ihr Typ soll die Schnittstelle `Element` implementieren und zu dessen package gehören, wobei die Werte, die die Getter liefern ebenfalls unten dargestellt sind. Zusätzlich soll der Bezeichner (also z.B. „Helium“) des Elements für andere Sprachen mit einem Setter (nur) aus dem `chemistry`-package heraus änderbar sein.

b) (10 Punkte) Dokumentieren Sie Ihren Code mit geeigneten Annotations und Javadoc-Kommentaren.

```
package chemistry;
interface Element{
  String getBezeichner();  // deutscher Name des Elements
  int getOrdnungszahl(); // die Ordnungszahl
}
```



Name: MatNr:

2. Exceptions und Streams

(25 Punkte) Ergänzen Sie die Klasse `Person` um zwei Methoden zum Schreiben und Lesen der Daten in eine Datei. Verarbeiten Sie die Exceptions, die dabei auftreten können durch Exception Chaining und erzeugen und werfen im Fall aller auftretenden Laufzeitfehler eine `PersonException`.
Implementieren Sie dazu auch die Klasse `PersonException`.

Hinweis: Öffnen, Lesen, Schreiben und Schließen von Dateien kann unter anderem die `java.io.IOExceptions` oder die `java.io.FileNotFoundException` werfen, die Sie abfangen müssen.

Achtung: Beim Lesen kann es passieren, dass das gelesene Datenformat nicht passt und es dann zu einer `ClassCastException` kommt. Fangen Sie auch diese Exception ab und verpacken Sie sie in einer `PersonException`, bevor Sie sie weiterleiten.

```
import java.io.*;
import java.util.Date;
class Person implements Serializable{
	String name;
	Date geburtstag;
	Person(String n, Date d){name = n; geburtstag = d;}

	public void writeToFile(String filename) throws PersonException{
		//TODO
	}
	public void readFromFile(String filename) throws PersonException{
		//TODO
	}
}
```

Name: MatNr:

3. Generics

(25 Punkte) Benutzen Sie `java.util.Comparator<T>` um Objekte auf unterschiedliche Weise zu sortieren.

- Schreiben Sie eine Java Anwendung, in der eine Liste mit 3 Instanzen der Klasse Student (s.u.) erstellt wird.
- Sortieren Sie diese Liste mit der statischen Methode `Collections.sort(List<T> l, Comparator<? super T> c)` nach `name` alphabetisch aufsteigend und geben Sie sie aus.
- Sortieren Sie die selbe Liste danach mit der Methode `sort` nochmals, aber nach `matrikel` aufsteigend und geben Sie sie erneut aus.

Hinweis: Außer der Anwendung müssen Sie zwei `Comparator`- Klassen implementieren. Das dürfen auch innere Klassen sein.

```
class Student{
   String name;
   int matrikel;
   Student(String n, int m){name = n; matrikel = m;}
   public String toString(){return name;}
}

interface Comparator<T>{
   /** Compares its two arguments for order. Returns a negative integer, zero, or a positive integer as the
    first argument is less than, equal to, or greater than the second.
   */
   int compare(T o1, T o2);
}
```

Name: MatNr:

4. Nebenläufige Threads

(20 Punkte) In einer Anwendung zum Umrechnung von Geldbeträgen in unterschiedliche Währungen (wie in der Übungsaufgabe) sollen immer aktuelle Wechselkurse verwendet werden. Diese können aus dem Internet über die Methode `static Set<Rate> getRates()` der Klasse `Rates` geladen werden.
Implementieren Sie eine Klasse `RatesThread`,

- in der die Wechselkurse als `Set<Rate>` abgespeichert werden.
- die alle 500ms die Wechselkurse nebenläufig aktualisiert, indem sie sie mit `Rates.getRates()` neu lädt und dann lokal abspeichert. Das nebenläufige Aktualisieren beginnt sofort nach der Erzeugung eines Objekts der Klasse.
- in der das ständige aktualisieren der Kurse mit einer Methode `stopLoading()` beendet werden kann.

Name: MatNr:

5. Innere Klassen und Lambda-Ausdrücke

(20 Punkte) Implementieren Sie den Code der notwendig ist, damit sich folgendes Programm übersetzen lässt.

```
class Test{
      public static void main(String[] a){ new Test();}
      Test(){
            X x = new X(2);
            X.Y xy = new X.Y(1.0);
            X.Z xz = x.new Z(”abc”);
            foo(new A(){public void m(){}});
            goo((X xx) -> 5);
      }
      void foo(A a){}
      void goo(B b){}
}
```