Musterklausur für die Vorlesung Algorithmen und Graphentheorie

1. Aufgabe:

Lesen Sie die Datei „student.txt“ zeilenweise ein und fügen die drei Elemente pro Zeile in eine Instanz der Klasse Student. Die Namen der Studenten sind nicht komplett sortiert. Die Buchstaben sind sortiert, aber innerhalb eines Buchstabens ist keine Sortierung. Deshalb benötigt man zur schnelleren Suche eine Indexdatei. Also ab wann ein Buchstabe abfängt (siehe student-Lösung.txt).

Musterlösung: test-datei: 90 Zeilen, plus Klasse Student

a) Legen Sie ein neues Projekt an bzw. kopieren Sie die Vorlage

b) Kopieren Sie die beiden Dateien „student.txt“ und „student-Lösung.txt“ in das Projektverzeichnis. Nicht in src!

c) Erstellen Sie eine Methode „getStudentListe“, die die Datei „student.txt“ und in eine Liste von Studenten einträgt.

Benutzen Sie dazu die Methode „readFileAllLines“ oder readFileStringJDK1 der Klasse Basis

d) Erstellen Sie eine Methode „createIndizes“, die die Indizes der Liste erstellt. In der Datei „student-Lösung.txt „sind die Indizes aufgelistet.

e) Methode test1:

Aufruf einer Methode zum Lesen der Datei

Aufruf einer Methode zum Erstellen der Indizes

Lokales Feld der Methode test1:

```
String[] nachnamen = {
    "Anders","Baatz","Dali","Eckert","Freud","Goos","Heinath","Ingenhoven",
    "Jelinek","Körbs","Lade","Mahnkopf","Nöllen","Piper","Stansbury",
    "Turandot","Unger","Vetter","Windsor","Yildiz","Zorn",
    "Niemann","niemann"
};
```

Testen Sie die Methode „suche“ mit dem vorgegebenen Suchnamen.

f) Erstellen Sie die Methode „suche“, die ab einer Position einen Nachnamen sucht.

Parameter:

ArrayList<Student> liste,

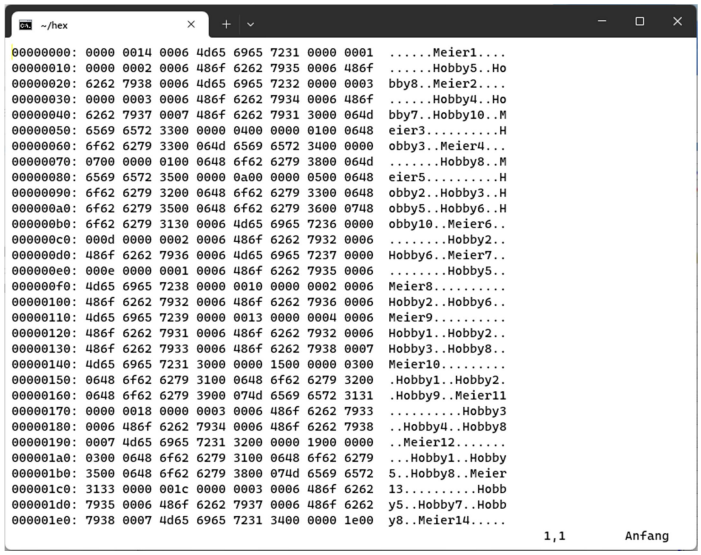
int[] abcIndizes,

String nachname

Geben Sie die Anzahl der Suchabfragen aus. Sind diese mit den in der Datei „student-Lösung.txt“ identisch?

Hinweise:

- Die zweite Aufgabe ist recht komplex. Man braucht ca. drei bis vier Stunden für die komplette Lösung.
- Falls Sie einen Aufgabenteil nicht sofort hinbekommen, können Sie ihn auch anders implementieren.
    - Beispiel: „unique der Hobbies“
- Bei Fragen wenden Sie sich an die „Hotline“.
- Im Anhang finden Sie wichtige Quellcode-Beispiele.
- Auf den letzten beiden Seiten finden Sie meine Test-Struktur (Nur ein Vorschlag).
- Folgende Dateien werden mitgeliefert:
    - student.txt Testausgabe
    - student1.bin Meine Lösung mit DataOutputStream
    - student1.text Meine Lösung mit DataOutputStream in einer Hex-Darstellung
    - student2.bin Meine Lösung mit ObjectOutputStream
    - student2.text Meine Lösung mit ObjectOutputStream in einer Hex-Darstellung
    - Hexeditor.exe Ein Hex-Viewer
    - Oder mit vim
        - vim student1.bin
        - dann :%!xxd



2. Aufgabe:

a) Erstellen Sie mit Hilfe der Klasse „Random“ n-Studenten mit maximal m-Hobbies.

Benutzen Sie dazu eine statische Factory-Methode.

Beispiel: n=20 Studenten, m=10 Hobbies

Projekt:

- Legen Sie ein neues Projekt an bzw. kopieren Sie die Vorlage.

Klasse Student:

- Erstellen Sie eine Klasse „Student“.
  - private Name: hier einfach Meier1 bis Meier1000 etc.
  - private Matrikelnummer: aufsteigend, aber mit Lücken !
  - private Liste der Hobbies: hier einfach: Hobby1 bis Hobby20 etc.
    - Es darf aber kein Hobby doppelt vorkommen!
    - Überlegen Sie, wie viele Hobbies ein Student maximal haben sollte.
- **Random**
  - Ich habe zwei Random-Instanzen erstellt.
  - Die erste Random-Instanz benutze ich für die weiteren Matrikelnummer:
    - random1.nextInt(3) + 1
  - Die zweite Random-Instanz benutze ich für die Anzahl der Hobbies:
    - random2.nextInt(m/2) + 1;
  - Beide Random-Instanzen initialisierte ich mit 12345679. Damit erhalte ich **immer** die gleichen Datensätze.
    - Ich habe die Namen der Studenten so benannt: „Meier“+i
    - Ich habe die Hobbies so benannt: „Hobby“+i

b) Erstellen einer Testliste:

- Erstellen Sie eine Testmethode, in der Sie n-Studenten mit maximal m-Hobbies erstellen. Die einzelnen Hobbies dürfen nicht doppelt vorkommen.
- Die Methode gibt eine Liste mit Studenten zurück.

c) Speichern der Liste

- Erzeugen Sie mit Hilfe der oberen Methode eine Studenten-Liste.
- Speichern Sie nun diese Liste mit Hilfe von:
  - FileOutputStream
  - DataOutputStream
  - Dateiname: student1.bin
  - Überlegen Sie sich das Dateiformat!
- Und dann mit Hilfe von
  - FileOutputStream
  - ObjectOutputStream (Serialize)
  - Dateiname: student2.bin

d) Lesen der Dateien:

- Lesen Sie nun die Datei „student1.bin“ mit Hilfe von:
    - FileInputStream
    - DataInputStream
    - Dateiname: student1.bin
    - Benutzen Sie dazu für die Klasse Student einen Default-Konstruktor.
    - In dieser Teilaufgabe sollen Sie auch die **Offset**sder einzelnen Datensätze ermitteln und ausgeben. Der erste Datensatz hat natürlich das Offset zwei oder vier. Der zweite Datensatz hat das Offset zwei oder vier plus der Länge des ersten Datensatzes.
        - Ausgabe der Offsets (wenn zu schwer, erstmal ignorieren)
        - Student: Meier1 Offset: 4 size: 32
        - Student: Meier2 Offset: 36 size: 41
        - Student: Meier3 Offset: 77 size: 24
        - Student: Meier4 Offset: 101 size: 24
        - Student: Meier5 Offset: 125 size: 57
        - Student: Meier6 Offset: 182 size: 32
        - Student: Meier7 Offset: 214 size: 24
        - Student: Meier8 Offset: 238 size: 32
        - Student: Meier9 Offset: 270 size: 48
        - Student: Meier10 Offset: 318 size: 41
        - Student: Meier11 Offset: 359 size: 41
        - Student: Meier12 Offset: 400 size: 41
        - Student: Meier13 Offset: 441 size: 41
        - Student: Meier14 Offset: 482 size: 26
        - Student: Meier15 Offset: 508 size: 58
        - Student: Meier16 Offset: 566 size: 41
        - Student: Meier17 Offset: 607 size: 33
        - Student: Meier18 Offset: 640 size: 34
        - Student: Meier19 Offset: 674 size: 33
        - Student: Meier20 Offset: 707 size: 41
        - letztes Offset: 748

e) Statistik

- Nachdem Einlesen der beiden Dateien werden die Hobbies ausgewertet:
- Ermitteln Sie:
    - das minimale Hobby
    - das maximale Hobby
    - das durchschnittliche Hobby
    - Liste der Anzahl pro Hobby
- Die Statistik wird mit zwei Methoden berechnet:
    - Methode groupHobbiesJava
        - Hier soll man nur pures Java nehmen (keine Streams)
    - Methode groupHobbiesStreams
        - Hier soll man möglichst alles mit Stream machen

Testausgabe:

Hobbies:

Hobby: Hobby8 Anzahl: 8

Hobby: Hobby2 Anzahl: 7

Hobby: Hobby7 Anzahl: 6

Hobby: Hobby6 Anzahl: 6

Hobby: Hobby3 Anzahl: 6

Hobby: Hobby5 Anzahl: 5

Hobby: Hobby10 Anzahl: 5

Hobby: Hobby4 Anzahl: 4

Hobby: Hobby1 Anzahl: 3

Hobby: Hobby9 Anzahl: 1

Minimale Anzahl für Hobby9 1

Maximale Anzahl für Hobby8 8

Anzahl Mittelwert: 5.1

f) Gruppieren mittels einfachem Java: Methode groupHobbiesJava

- Parameter: Liste der Studenten
- Bestimmen Sie die minimale, maximale und durchschnittliche Anzahl der Hobbies pro Student.

Ausgabe:

Hobbies:

Minimale Anzahl für Hobby9 1

Maximale Anzahl für Hobby8 8

Anzahl Mittelwert: 5.1

g) Gruppieren mittels Streams: Methode groupHobbiesStreams

- Parameter: Liste der Studenten
- Bestimmen Sie die minimale, maximale und durchschnittliche Anzahl der Hobbies pro Student.

Ausgabe:

Hobbies:

Minimale Anzahl für Hobby9 1

Maximale Anzahl für Hobby8 8

Anzahl Mittelwert: 5.1

3. Aufgabe Graphentheorie:

In der Datei „graph.txt“ stehen alle Nachfolger der Knoten im gegebenen nicht gerichteten Graph. Das Minuszeichen bedeutet, dass der Nachfolger nicht vorhanden ist.

Ablauf:

- **Projekt:**
  - Legen Sie ein neues Projekt an bzw. kopieren Sie die Vorlage.
- Erstellen Sie die Klasse Knoten.
  - Attribute:
    - public bez
    - private ArrayList<Knoten> nextKnoten
  - Methoden (Vorschlag):
    - public void addKnoten(Knoten knoten)
    - public int getAnzahlNextKnoten()
    - public boolean checkLoop(String bez)
    - public String toString()
- Erstellen Sie die Klasse Kanten.
  - Attribute:
    - public knotenVon
    - public knotenBis
  - Methoden (Vorschlag):
    - public String toString()
- Lesen Sie die Datei in eine Kantenliste ein.
  - Eventuelle Fehler bitte berücksichtigen!
- Ausgabe der Kantenliste.
- Erzeugen Sie die Knotenliste (auch die Nachfolgerliste)
- Ausgabe der Knotenliste.
- Statistik
  - Aus der Knotenliste geben Sie aus:
    - Ausgabe der Endpunkte
    - Ausgabe der freien Knoten (haben keine Nachfolger, oder Vorgänger)
    - Ausgabe der Knoten mit Loop-Beziehung
    - Ausgabe der Vorgänger pro Knoten

Ausgabe der Musterlösung:

```
readKanten


Ausgabe der Kanten
Kante{knotenVon='a', knotenBis='c'}
Kante{knotenVon='a', knotenBis='b'}
Kante{knotenVon='b', knotenBis='f'}
Kante{knotenVon='b', knotenBis='b'}
Kante{knotenVon='c', knotenBis='e'}
Kante{knotenVon='c', knotenBis='d'}
Kante{knotenVon='d', knotenBis='g'}
Kante{knotenVon='g', knotenBis='i'}
Kante{knotenVon='g', knotenBis='j'}
Kante{knotenVon='f', knotenBis='h'}
Kante{knotenVon='g', knotenBis='h'}
Kante{knotenVon='h', knotenBis='k'}
Kante{knotenVon='i', knotenBis='z'}
Kante{knotenVon='i', knotenBis='j'}
Kante{knotenVon='n', knotenBis=''}
Kante{knotenVon='m', knotenBis='m'}



Ausgabe der Knoten
bez:a  Next: c   b
bez:b  Next: a   f   b
bez:c  Next: a   e   d
bez:d  Next: c   g
bez:e  Next: c
bez:f  Next: b   h
```

bez:g Next: d i j h

bez:h Next: f g k

bez:i Next: g z j

bez:j Next: g i

bez:k Next: h

bez:m Next: m

bez:n Next:

bez:z Next: i

Ausgabe der Statistik

Ausgabe der Endpunkte:

Knoten: e

Knoten: k

Knoten: m

Knoten: z

Ausgabe der freien Knoten:

Knoten: n

Ausgabe der Knoten mit Loop-Beziehung:

Knoten: b

Knoten: m

Ausgabe der Vorgänger pro Knoten:

Knoten: a Nachfolger: b c

Knoten: b Nachfolger: a b f

Knoten: c Nachfolger: a d e

Knoten: d Nachfolger: c g

Knoten: e Nachfolger: c

Knoten: f Nachfolger: b h

Knoten: g Nachfolger: d h i j

Knoten: h Nachfolger: f g k

Knoten: i Nachfolger: g j z

Knoten: j Nachfolger: g i

Knoten: k Nachfolger: h

Knoten: m Nachfolger: m

Knoten: n Nachfolger:

Knoten: z Nachfolger: i

Anhang

I/O- Data

Datei Öffnen zum Schreiben:

```
FileOutputStream fout = new FileOutputStream(filename);
BufferedOutputStream bout = null;  // optional
DataOutputStream dout = new DataOutputStream(fout);
Oder
DataOutputStream dos = new DataOutputStream(fout);
…
dout.close();
fout.close();
```

Datei Öffnen zum Lesen:

```
FileInputStream fin = new FileInputStream(filename);
DataInputStream din = new DataInputStream(fin);
…
din.close();
fin.close();
```

Methoden zum Schreiben:

```
dout.writeInt(…);
dout.writeUTF(…);          // Bitte schauen Sie mit dem HexViewer, wie Java einen String speichert!
dout.writeDouble(…);
etc.
```

Methoden zum Lesen:

```
int n = din.readInt();

String dummy = din.readUTF();

double number = din.readDouble();

etc.
```

IO-Serialize

Wenn man Klassen mit dieser Technik speichert oder liest, benötigt man eine SerialiteID.

Beispiel:

- private static final long serialVersionUID = 1L;

Datei Öffnen zum Schreiben:

```
FileOutputStream fout = new FileOutputStream(filename);
BufferedOutputStream bout = null;  // optional
ObjectOutputStream oos = new ObjectOutputStream(???);
…
oos.close();
fout.close();
```

Datei Öffnen zum Lesen:

```
FileInputStream fin = new FileInputStream(filename);
BufferedInputStream bin = null;
ObjectInputStream ios = new ObjectInputStream(???);
…
ios.close();
fin.close();
```

Methoden zum Schreiben:

```
oos.writeObject(student);
oos.writeObject(liste);
```

Methoden zum Lesen:

```
liste = (ArrayList<Object>) ios.readObject();
Student student = (Student) ios.readObject();
```

HashMap

Eine Hashmap speichert Objekte mit Hilfe einer internen Hashfunktion. Das heißt, das zwei Objekte niemals den gleichen Hashwert haben sollen.

Vorteil:

- Bei einer unsortierten Liste muss man die Liste maximal komplett durchsuchen. Mit der HashMap wird mit dem Objekt nur die Hashfunktion, also der interne Index, errechnet und dann das Objekt aus der internen Liste geholt.

Erstellen:

- HashMap<Key-datentyp, Objekt-Datentyp> hashMap = new HashMap <>(???);

Einfügen:

- Student student = new Student("Schmidt", 1000);
- hashMap.put(student. hashCode(),student) ;
- oder
- hashMap.put(student.getMnr(), student);
- oder
- hashMap.put(1000, 2000);

Holen eines Werte:

- Student student = (Student)hashMap.put(student.getMnr())
- Integer nr = (Integer)hashMap.put(1000)
- Falls der Wert noch nicht in der Liste ist, gibt es null zurück.

Holen aller Objekte(nicht die Keys):

- Collection<Student> values = hashMap.values();
- for (Student student:values) {
    - sout(student);
- }

Holen aller Keys:

- Set set = hashMap.keySet();

Sortieren

Eine Klasse wird sortiert, indem Sie das Interface „Comparable“ implementiert:

```
@Override
   public int compareTo(MyObject myobj) {
    int result = 0;
            ….
     return –result;
   }
```

Rückgabewerte:

<0: Das Object ist kleiner als das per Parameter übergegebenen

0: Das Object ist gleich dems per Parameter übergegebenen

>0: Das Object ist größer als das per Parameter übergegebenen

Falls es mehrere Sortierkriterien gibt, könnte man das mit einer statischen Variablen lösen.

Compare-Methoden:

- Integer k
  - Integer.compare(k,myobj.k);
- Double k
  - Double.compare(k,myobj.k);
- Short k
  - Short.compare(k,myobj.k);
- Byte k
  - Byte.compare(k,myobj.k);
- Long k
  - Long.compare(k,myobj.k);
- Float k
  - Float.compare(k,myobj.k);
- String bez
  - bez.compareTo(myobj.bez)

Streams

Summieren einer ArrayList:

- List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
- int summe = numbers.stream().reduce(0, (number1, number2) -> number1 + number2);

- List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
- int summe = numbers.stream().reduce(0, ArithmeticUtils::**add**);

- List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
- int summe = numbers.stream().mapToInt(Integer::intValue).sum();

Summieren mit Map (zu simpel):

- Map<Double, Integer> map = new HashMap<>();
- map.put(1.0, 100);
- map.put(2.0, 200);
- map.put(3.0, 300);
- map.put(4.0, 400);
- int summe = map.values().stream().reduce(0, Integer::sum);

Summieren mit Map mit eigenen Klassen:

- List<Employee> emps = Arrays.asList()
- emps.add(new Employee(100, "A", 25));
- emps.add(new Employee(200, "B", 35));
- emps.add(new Employee(300, "C", 45));
- 1. Variante
    - int summe = emps.stream().map(emp -> emp.getAge()).reduce(0, (a, b) -> a + b);
- 2. Variante
    - int summe = emps.stream().map(emp -> emp.getAge()).mapToInt(Integer::intValue).sum();
- 2. Variante
- int summe = emps.stream().map(emp -> emp.getAge()).collect(Collectors.summingInt(Integer::intValue));

Klasse Basis

```
import java.io.*;
import java.nio.file.*;
import java.util.List;


public class Basis {


  // jdk 11
  public static List<String> readFileAllLines(String filename) {
     try {
        Path path = Paths.get(filename);
        List<String> lines = Files.readAllLines(path);
        return lines;
     } catch (IOException e) {
        Error("Fehler", "Fehler beim Lesen der Datei:\n" + filename);
        e.printStackTrace();
        return null;
     }
  }


   // ab jdk 1
  public static String readFileStringJDK1(String filename) {
     StringBuilder sb = new StringBuilder();
     try {
        FileInputStream fin;
        InputStreamReader iin;
        LineNumberReader din;
        String line;
        fin = new FileInputStream(filename);
```

```
            iin = new InputStreamReader(fin);
            din = new LineNumberReader(iin); // oder BufferedReader br = new BufferedReader(ir);
            while (din.ready()) {
               line = din.readLine();
               sb.append(line);
               sb.append("\n");
            }
         } catch (FileNotFoundException e) {
            Error("Fehler", "Fehler beim Lesen der Datei:\n" + filename);
            e.printStackTrace();
            return "";
         } catch (IOException e) {
            Error("Fehler", "Fehler beim Lesen der Datei:\n" + filename);
            e.printStackTrace();
            return "";
         }
         return sb.toString();
      }


}
```

Quellcode von Tests.java für Aufgabe2 (kann als Vorlage benutzt werden)

```
import java.io.*;
import java.util.*;


public class Tests {


    public void test1(int n, int m) {
        // erstellen der Studentenliste und Ausgabe
        testWrite1("student1.bin",liste);
        testWrite2("student2.bin",liste);
    }


    public void testWrite1(String filename, ArrayList<Student> liste) {
        System.out.println("in testwrite1");
        // speichern der Liste
    }  // testWrite1


    public void testWrite2(String filename, ArrayList<Student> liste) {
        System.out.println("in testwrite2");
        // speichern der Liste
    }  // testWrite2
```

```
    public void test2() {
        System.out.println("\nReadTest1");
        // lesen von "student1.bin"  testRead1
        // Ausgabe

        System.out.println("\nReadTest2");
        // lesen von "student2.bin"  testRead2
        // Ausgabe

        // nun Statistik
        groupHobbiesJava(liste);
        groupHobbiesStreams(liste);
    }  // test2


    private ArrayList<Student> testRead1(String filename) {
    }  // testRead1

    private ArrayList<Student> testRead2(String filename) {
    } // testRead2

    private void groupHobbiesJava(ArrayList<Student> liste) {
        System.out.println("groupHobbiesJava");
    } // groupHobbiesStreams

    private void groupHobbiesStreams(ArrayList<Student> liste) {
        System.out.println("\ngroupHobbiesStreams");
    }
} // class Tests
```