Programmiertechnik II
Klausur SS 2020

Angewandte Informatik Bachelor

Name	
Matrikelnummer	

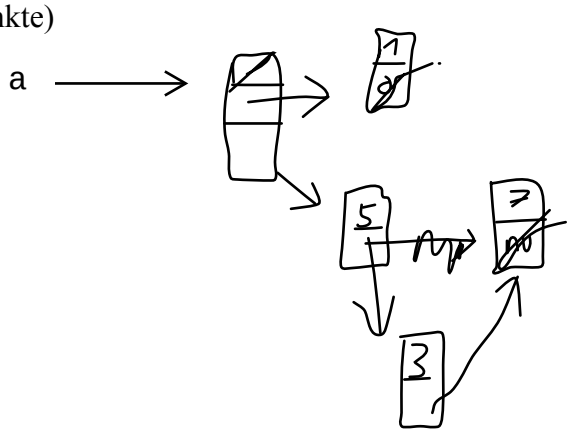
Aufgabe	Punkte
1	9
2	8
3	17
4	23
Zwischen-summe	**57**

Aufgabe	Punkte
5	16
6	22
7	9
8	16
Summe	**120**
Note	

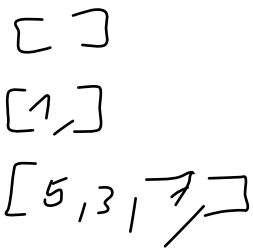
Aufgabe 1 ***(9 Punkte)***

a) Beschreiben Sie mit einem Speicherbelegungsbild (Referenzen als Pfeile!), was durch die main-Methode geleistet wird. Es genügt das Speicherbelegungsbild anzugeben, nachdem alle Anweisungen der main-Methode ausgeführt worden sind. (6 Punkte)

b) Was gibt die main-Methode auf die Konsole aus? (3 Punkte)



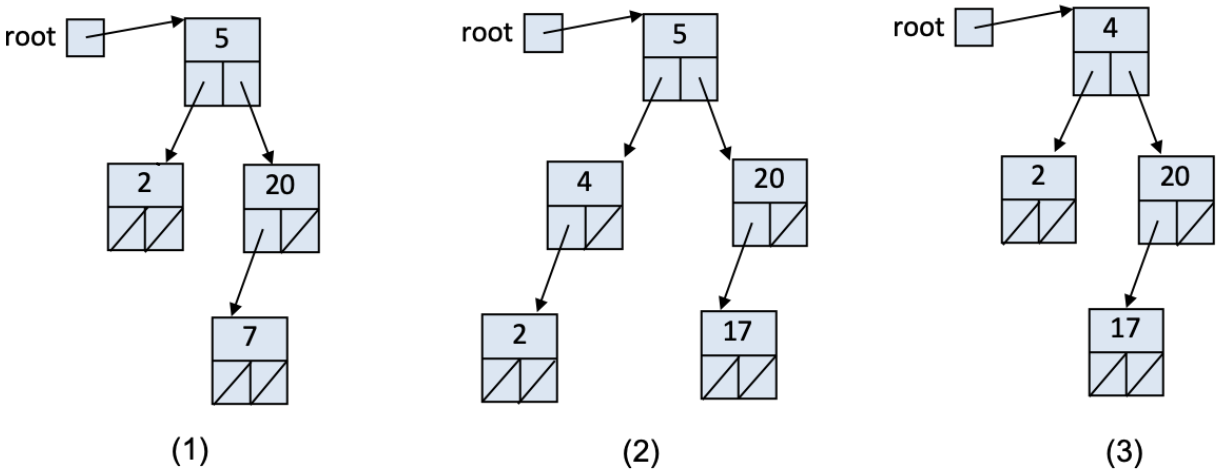

```
static class Node {
   Node next;
   int data;

   Node(int x, Node p) {
      next = p;
      data = x;
}

public static void main(String[] sf) {
   Node[] a = new Node[3];
   a[2] = new Node(7, null);
   a[2] = new Node(5,a[2]);
   a[2].next = new Node(3, a[2].next);
   Node q = new Node(1, null);
   a[1] = q;

   for (Node r : a) {
      System.out.print("[");
      for (Node s = r; s != null; s = s.next)
         System.out.print(s.data + ", ");
      System.out.println("]");

   }
}
```



Aufgabe 2 ***(8 Punkte)***

Die Knoten eines binären Suchbaums sind durch folgende Klasse definiert:

```
class Node {
     int data;
     Node left;     // Referenz auf linkes Kind
     Node right;    // Referenz auf rechtes Kind

     Node(int x, Node l, Node r) {
          data = x;
          left = l;
          right = r
     }
}
```



a) Der binäre Suchbaum (2) entsteht aus dem binären Suchbaum (1), indem die Zahl „7“ durch „17“ ersetzt und ein neuer Knoten mit der Zahl „4“ eingefügt wird. Schreiben Sie mit Hilfe der Variablen `root` genau 2 Java-Anweisungen, die das leisten.

```
root.right.left = new Node(17,null,null);
root.left = new Nodew(4,root.left,null);
```

b) Der binäre Suchbaum (3) entsteht aus dem binären Suchbaum (2), indem der rechte Teilbaum der Wurzel `root` an den Knoten „4“als rechtes Kind eingehängt und dann der Wurzelknoten ausgehängt wird. Schreiben Sie genau 2 Java-Anweisungen, die das leisten.

```
root.left.right = root,right;
root = root.left;
```

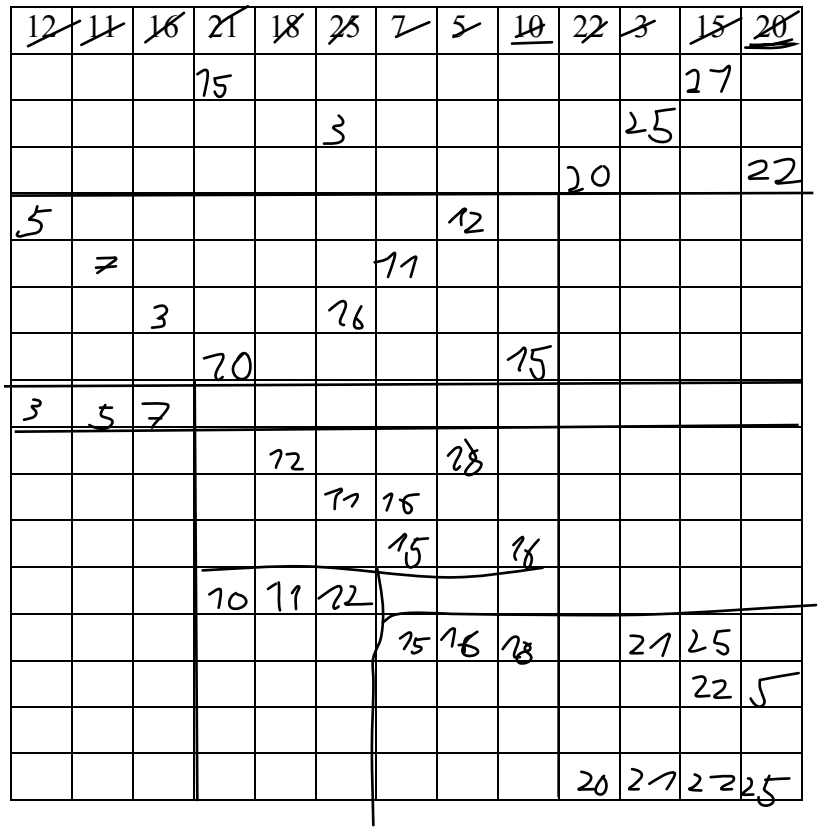
Aufgabe 3 QuickSort (17 Punkte)

a) Das 13-elementige Feld a = {12, 11, 16, 21, 18, 25, 7, 5, 10, 22, 3, 15, 20} wird mit Quicksort (ohne 3-Median-Strategie) sortiert. Tragen Sie in die Tabelle ein, wie sich das Feld a ändert. Geben Sie außerdem die Aufrufstruktur von Quicksort an. (11 Punkte)

Außerdem soll folgende Vereinfachung berücksichtigt werden: Besteht das zu sortierende Teilfeld nur aus 2 oder 3 Elementen, dann darf das Teilfeld durch einfache Vertauschungsschritte sortiert werden. Die Vertauschungen dürfen in einem Schritt durchgeführt werden (d.h. eine Zeile in der Tabelle).

b) Was ergibt die Partitionierung von QuickSort, wenn das Feld aus n = 2k+1 (d.h. n ist eine ungerade Zahl) gleichen Elementen besteht? (3 Punkte)

c) Wann führt eine Partitionierung von QuickSort zu Teilfeldern mit besonders starken Größenunterschiede? (3 Punkte)



Aufgabe 4 Sortierte, verkettete Liste mit Hilfskopfknoten (23 Punkte)

Die generische Klasse `MultiSet` speichert Elemente in einer linear verketteten Liste mit Hilfskopfknoten in einer absteigend sortierten Reihenfolge. Ergänzen Sie die Klasse um die folgenden Methoden (siehe auch nächste Seite).

a) insert(x) fügt ein Element x ein. x wird auch dann eingefügt, wenn x bereits vorkommt. Mehrfaches Vorkommen eines Elements ist also erlaubt. (6 Punkte)

b) isEmpty() prüft, ob Elemente vorkommen. (1 Punkt)

c) delMax() löscht das größte Element und liefert es zurück. Bei einer leeren Liste wird eine NullPointerException ausgelöst. Falls das größte Element nicht eindeutig ist, dann wird genau eines der größten Elemente gelöscht. (4 Punkte)

d) distinct() löscht alle mehrfachen Vorkommen eines Elements. Die Methode muss eine Laufzeit von O(n) haben, wobei n die Anzahl der Elemente ist. (8 Punkte)

e) Was gibt die main-Methode auf die Konsole aus? (4 Punkte)

```
class MultiSet<T extends Comparable<T>> {

   static private class Node<T> {
      private T data;
      private Node<T> next;
         Node(Node<T> p, T x) {
            data = x;
            next = p;
         }
   }
   private Node<T> head;


   public MultiSet() {
      head = new Node<>(null, null);
   }


   public boolean isEmpty() {
       return (head.next == null);


   }


   public void insert(T x) {
   for(Node<T> q = head, q != null, q = q.next) {
       if(q.next != null){
           if(q.next.data < x)}
                q.next = new Node(q.next, x);
                return;
           }
       continue;
       }
       q.next = new Node(null, x);
                return
   }
```

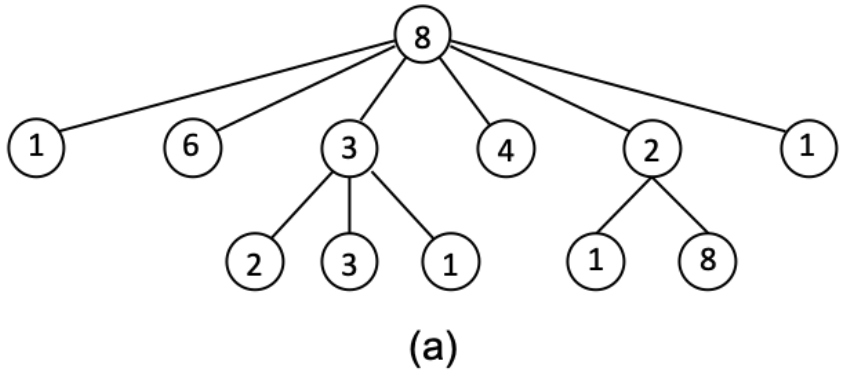
```
    public T delMax() {
    if (head.next == null)
        throw new NullPointerException();
    else {
        T data = head.next.data;
        head.next = head.next.next;
    return data;



    }


    public void distinct() {
    if (head.next == null)
        return;
    Node<T> p = head;
    while(p.next != null)
        if(p.data.compareTo(p.next.data) == 0)
            p.next = p.next.next;
        p = p.next;









    }


    public void fun(MultiSet<T> s) {
       for (Node<T> p = s.head.next; p != null; p = p.next)
          insert(p.data);
    }


    public static void main(String[] args) {
       MultiSet<Integer> s1 = new MultiSet<>();
       s1.insert(5);
       s1.insert(3);
       s1.insert(5);
       s1.insert(1);

       MultiSet<Integer> s2 = new MultiSet<>();
       s2.insert(5);
       s2.insert(1);
       s2.insert(8);
       s2.insert(5);

       s1.fun(s2);

       while (!s1.isEmpty())
          System.out.print(s1.delMax() + ", ");
       System.out.println("");
    }
}
```

8, 5, 5, 5, 3, 1,

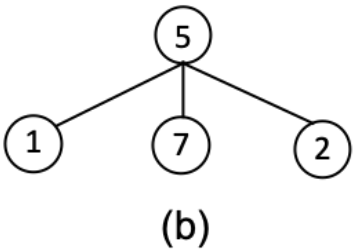
Aufgabe 5 Bäume (16 Punkte)

Die Klasse `Tree` definiert Bäume mit int-Daten in jedem Knoten und einer beliebigen Anzahl von Kindern. `root` ist die Wurzel des Baums.

In der Abbildung (a) und (b) sind zwei Beispielbäume gezeigt.



(a)



(b)

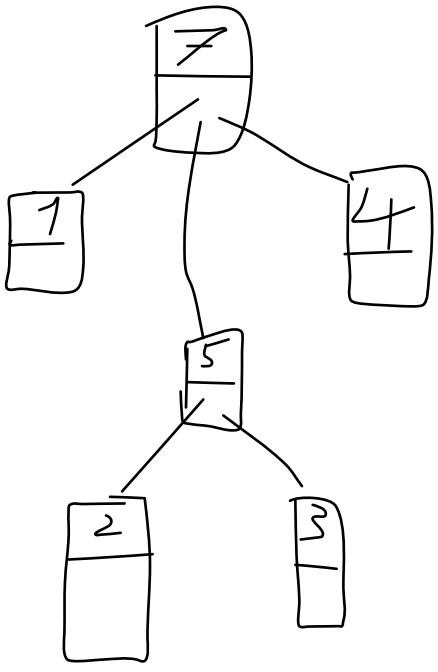
Ein Teil der Klasse `Tree` ist auf dieser und der nächsten Seite bereits vorgegeben. Lösen Sie folgende Aufgaben.

a) Die statische Methode `buildTree` erzeugt einen Baum. Stellen Sie den Baum grafisch dar. (4 Punkte)

b) Schreiben Sie eine Methode `sum`, die die Summe aller Zahlen im Baum zurückliefert. Beispielsweise ergibt im Baum (b) die Summe 15. Hinweis: Eventuell ist es hilfreich, wenn `sum` eine private Methode aufruft, die dann rekursiv definiert wird. (6 Punkte)

c) Schreiben Sie eine Methode `height`, die die Höhe des Baums zurückliefert. Beispielsweise hat der Baum in (a) die Höhe 2 und in (b) die Höhe 1. Beachten Sie den Hinweis aus b). (6 Punkte)

```
class Tree {

      private static class Node {
            int data;
            List<Node> children;
            Node(int d) {
                  data = d;
                  children = new LinkedList<>();
            }
      }
      private Node root = null;

      public static Tree buildTree() {
            Node p = new Node(5);
            p.children.add(new Node(2));
            p.children.add(new Node(3));
            Tree t = new Tree();
            t.root = new Node(7);
            t.root.children.add(new Node(1));
            t.root.children.add(p);
            t.root.children.add(new Node(4));
            return t;
      }
```



```
    public int sum() {

    public int height() {

}
```

Aufgabe 6 Collection (22 Punkte)

Gegeben ist eine Klasse Buch und eine List von Büchern buchListe:

```
class Buch {
        public String autor;
        public String titel;
        public int jahr;            // Erscheinungsjahr
        public Buch(String a, String t , int j) { autor = a; titel = t; jahr = j;}
        public toString() {...}
}

List<Buch> buchListe = new LinkedList<>();
buchList.add(new Buch("Grisham","Die Jury",1993));
buchList.add(new Buch("Crichton","Airframe",1997));
// ...
```

a) Sortieren Sie buchListe aufsteigend nach dem Erscheinungsjahr. Benutzen Sie die sort-Methode aus dem Java-Interface List. (3 Punkte)

b) Definieren Sie eine statische Methode list2Set(bList), der eine Liste von Büchern bList übergeben wird und die die Menge der Autoren zurückliefert, die in bList vorkommen. (5 Punkte)

c) Definieren Sie eine statische Methode list2Map(bList), der eine Liste von Büchern bList übergeben wird und die eine Map zurückliefert, die jedes Jahr einer Liste der erschienenen Bücher zuordnet. (7 Punkte)

d) Definieren Sie eine statische Methode map2List(jahr2BuchList, j), der eine SortedMap jahr2BuchList, die jedes Jahr einer Liste der erschienenen Bücher zuordnet, und ein Jahr j übergeben wird und die eine Liste aller Bücher zurückliefert, die nach dem Jahr j (einschließlich) erschienen sind.

Benutzen Sie die Methode tailMap aus dem Java-Interface SortedMap. (7 Punkte)

Aufgabe 7 Subtyping (9 Punkte)

Es werden folgende Variablen definiert, wobei die Initialisierungen weggelassen sind.

```
Collection<Double> colDb;
Collection<Number> colNb;
Collection<Integer> colInt;
Collection<Object> colObj;
LinkedList<String> llStr;
ArrayList<Object> arrListObj;
TreeSet<Number> treeSetNb;
ArrayDeque<Double> arrQueueDb;
```

Geben Sie in folgender Tabelle für jeden Aufruf an, ob er korrekt ist („+“) oder ob er nicht korrekt ist („-“). Falsche Antworten geben Abzüge!

`treeSetNb.addAll(arrQueueDb);`	+
`colInt.addAll(arrListObj);`	−
`colObj.addAll(colObj);`	+
`colObj.add(colObj);`	−
`arrQueueDb.containsAll(llStr);`	−
`treeSetNb.removeAll(colObj);`	+
`colInt.add(colNb);`	−
`colInt.remove(colDb);`	−
`llStr.addAll(arrListObj);`	−

Aufgabe 8 Lambda-Ausdrücke und Ströme (16 Punkte)

Gegeben ist eine Klasse Einkauf und eine Liste von Einkäufen ekList:

```
class Einkauf {
        public String name;        // Name des gekauften Artikels
        public int jahr;           // Anschaffungsjahr
        public double kosten;      // Kosten in Euro
        public Einkauf (String s, int j, double k) { name = s; jahr = j; kosten = k;}
}
List<Einkauf> ekList = new LinkedList<>();
ekList.add(new Einkauf("PC", 2018, 799.00));
ekList.add(new Einkauf("Drucker", 2017, 195.00));
ekList.add(new Einkauf("Drucker2", 2017, 95.00));
// ...
```

a) Definieren Sie einen Comparator c (mit Typ) für Einkäufe, der den Vergleich von Einkäufen auf ihre Kosten zurückführt. Benutzen Sie die statische Methode comparing. (2 Punkte)

b) Erzeugen Sie aus ekList einen Strom und weisen Sie einer Variablen sum mit Hilfe von Strom-Operationen die Summe der Kosten der im Zeitraum [2015,2019) angeschafften Artikeln zu. (4 Punkte)

c) Erzeugen Sie aus ekList einen Strom und geben Sie alle Artikel, die mit „D“ beginnen, nach den Kosten absteigend sortiert aus. Hinweis: Verwenden Sie a). (4 Punkte)
(4 Punkte)

d) Was wird der Variablen m zugewiesen? Sie müssen den Wert des Ausdrucks nicht ausrechnen. Die Benutzung einer mathematischen Funktion genügt. (3 Punkte)

```
int n = 10;
int m = IntStream.range(1, n+1).reduce(1, (x,y) -> x*y);
```

e) Was gibt folgende Strom-Operation aus? (3 Punkte)

```
IntStream.iterate(1, x -> 2*x+1)
        .limit(8)
        .forEach(x -> System.out.print(x + ", "));
```