Programmiertechnik II
Klausur WS 2020/21
Angewandte Informatik Bachelor

Prof. Dr. Oliver Bittel

Name	
Matrikelnummer	

Aufgabe	Punkte
1	8
2	8
3	11
4	22
5	14
6	16
7	11
8	18
9	12
Summe	**120**

1. Beachten Sie die fettgedruckten Textteile.
2. Viel Erfolg!

Aufgabe 1 (8 Punkte)

Die Klasse `Node` ist wie folgt definiert.

```
class Node {
   public Node next;
   public String[] data;

   public Node(Node p, String[] a) {
      this.next = p;
      this.data = a;
   }
}
```

a) Beschreiben Sie mit einem Speicherbelegungsbild für die Variable **`lines`**, was durch folgende Anweisungen geleistet wird: (6 Punkte)

```
String[] s = {"def", "ijk"};
Node lines = new Node(null, s);
s = new String[]{"xyz"};
lines = new Node(lines, s);
lines.next = new Node(lines.next, new String[]{"def", "ijk"});
```

b) Was wird auf die Konsole ausgegeben, wenn die Anweisungen aus a) und dann folgende Anweisungen durchgeführt werden. (2 Punkte)

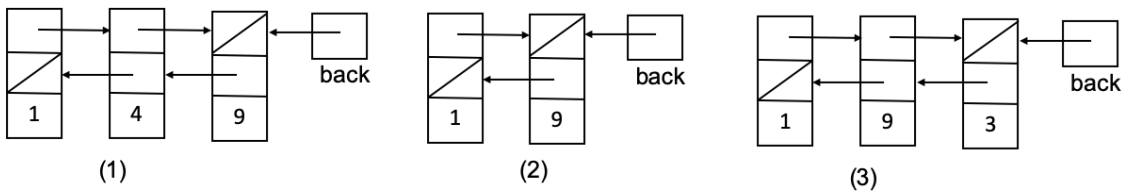
```
for (Node l = lines; l != null; l = l.next) {
   for (String w : l.data)
      System.out.print(w + ",␣");
   System.out.println("");
}
```

Aufgabe 2 (8 Punkte)

Für Knoten einer doppelt verketteten Liste sei die Klasse `Node` definiert.

```
class Node {
   int data;
   Node next;  // Referenz auf naechsten Knoten
   Node prev;  // Referenz auf vorhergehenden Knoten
   Node(int x, Node n, Node p) {data = x; next = n; prev = p}
}
```

Die Listen haben nur einen `back`-Zeiger auf den hinteren Knoten. Die folgende Abbildung zeigt drei verschiedene Zustände einer Liste.



a) Schreiben Sie eine Folge von Java-Anweisungen (keine Schleife!), die Liste 1 in Liste 2 überführt, indem der Knoten mit `data` = 4 gelöscht wird. (4 Punkte)

b) Schreiben Sie eine Folge von Java-Anweisungen (keine Schleife!), die Liste 2 in Liste 3 überführt, indem ein Knoten mit `data` = 3 eingefügt wird. (4 Punkte)

Aufgabe 3 (11 Punkte)

Das 11-elementige Feld a = {20, 19, 15, 14, 13, 10, 9, 8, 6, 5, 3} soll mit **Quicksort mit 3-Median-Strategie** sortiert werden.

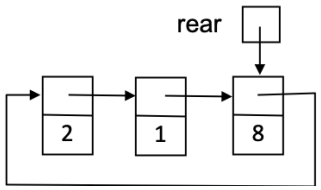
Beschreiben Sie, wie sich dabei das Feld a ändert. Benutzen Sie eine tabellenartige Darstellung wie in der Vorlesung. Geben Sie außerdem die Aufrufstruktur von Quicksort an.

Die **3-Median-Strategie** soll dabei wie folgt umgesetzt werden: Sortieren Sie die 3 Zahlen a[li], a[m] und a[re] mit m = (li+re)/2 (a[li] ist das Element am linken Rand, a[m] ist das Element in der Mitte und a[re] ist das Element am rechten Rand). Vertauschen Sie dann a[m] mit a[re]. Die 3-Median-Strategie darf in einem Schritt durchgeführt werden (d.h. eine Zeile in der Tabelle). 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).

Aufgabe 4 (22 Punkte)

Eine **Schlange (Queue)** soll als verkettete Ringliste realisiert werden. Folgende Abbildung zeigt eine 3-elementige Schlange, wobei 2 das vordere und 8 das hintere Element ist.



Gegeben ist eine rudimentäte Klasse `Queue`:

```
class Queue {
   private class Node {
      public int data;
      public Node next;
      public Node(int d, Node n) {
         data = d;
         next = n;
      }
   }

   // ...
}
```

a) Ergänzen Sie die Kasse um geeignete Instanzvariablen und definieren Sie einen Konstruktor. 3 Punkte)

b) Definieren Sie eine Methode `offer(x)`, die ein neues Element in die Schlange hinten anfügt. (4 Punkte)

c) Definieren Sie eine Methode `poll()`, die das vordere Element aus der Schlange entfernt und zurückliefert. Berücksichtigen Sie den Fehlerfall einer leeren Liste. (4 Punkte)

d) Definieren Sie eine Methode `offerAll(Queue q)`, die alle Elemente einer Schlange `q`, hinten anfügt. (5 Punkte)

e) Definieren Sie eine Methode `size()`, die die Anzahl der Elemente in der Schlange zurückliefert. Die Methode darf nur einen **Aufwand von O(1)** haben. (2 Punkte)

f) Leiten Sie die Klasse `Queue` zu einer instrumentierten Klasse `QueueInstr` ab, die die Anzahl der Methoden-Aufrufen von `offer(x)` zählt. Mit einer `get()`-Methode soll der Zählerstand zurückgeliefert werden. (4 Punkte)

Aufgabe 5 (14 Punkte)

Folgende statische Methode ermittelt alle Zahlenpaare in einem Feld, deren Summe 1971 ergibt:

```
public static void fun1971(int[] x) {
   for (int i = 1; i < x.length-1; i++)
      for (int j = i+1; j < x.length; j++) {
         if (x[i]+x[j] == 1971)
            System.out.println(x[i] + ",␣" + x[j]);
   }
}
```

a) Schätzen Sie die Laufzeit T(n) der Methode `fun1971(x)` ab (O-Notation). Gehen Sie dabei davon aus, dass das Feld `x` die Größe n hat. (3 Punkte)

b) Geben Sie eine effizientere Lösung an, indem `fun1971(x)` eine lokale Variable vom Typ `TreeSet` verwendet. (8 Punkte)

c) Von welcher Größenordnung ist die Laufzeit Ihrer Funktion aus c). (3 Punkte)

Aufgabe 6 (16 Punkte)

Es soll eine Klasse `TracingTable` zum Aufzeichnen von Variablenwerte realisiert werden. Die Klasse definiert die beiden Methoden `trace` und `get`, deren Arbeitsweise im folgenden Beispiel dargestellt wird:

```
public static void main(String[] args) {
   TracingTable tab = new TracingTable();

   double s = 0.0;
   for (int i = 1; i <= 10; i++) {
      s += i;
      tab.trace("i", i);   // Zeichne Variable i auf
      tab.trace("s", s);   // Zeichne Variable s auf
   }

   System.out.println(tab.get("i"));   // 1,2,3,4,5,6,7,8,9,10
   System.out.println(tab.get("s"));   // 1,3,6,10,15,21,28,36,45,55
}
```

a) Definieren Sie eine Klasse `TracingTable` mit Konstruktor. Sehen Sie für die Verwaltung der Variablenwerte den Datentyp **`Map`** vor. (5 Punkte)

b) Definieren Sie eine geeignete Methode `trace`. (8 Punkte)

c) Definieren Sie eine geeignete Methode `get`. (3 Punkte)

Aufgabe 7 (11 Punkte)

a) Fügen Sie in einem leeren binären Suchbaum die folgenden 9 Zahlen ein: 5, 3, 31, 29, 35, 9, 27, 21, 28. (2 Punkte)

b) Löschen Sie in dem Baum, der sich in a) ergeben hat, die Zahl 5 und dann die Zahl 9. (3 Punkte)

c) Die Klasse `BinarySearchTree` für binäre Suchbäume ist rudimentär definiert.

```
public class BinarySearchTree {
   static private class Node {
      private int data;
      private Node left;
      private Node right;
   }
   private Node root = null;
   // ...
}
```

Ergänzen Sie die Klasse um eine Methode `numSingleChildNodes()`, die die Anzahl der Knoten, die genau ein Kind haben, zurückliefert. Hinweis: definieren Sie zusätzlich eine private rekursive Methode. (6 Punkte)

Aufgabe 8 (18 Punkte)

Gegeben ist eine generische Klasse `Box`, die ein Datenelement `x` kapselt und eine `main`-Methode:

```
1  public class Box<T> {
2     private T x;
3
4     public void put(T x) {this.x = x;}
5     public T get() {return this.x;}
6
7     public void put(Box<T> b) {
8        this.x = b.get();
9     }
10    public void get(Box<T> b) {
11       b.put(this.x);
12    }
13
14    public static void main(String[] args) {
15       Box<Number> nbBox = new Box<>();
16       nbBox.put(17);
17       System.out.println(nbBox.get());
18
19       Box b = new Box();
20       b.put("abc");
21       int i = (int) b.get();
22
23       Box<Number> nbBox1 = new Box<>();
24       nbBox1.put(4);
25       Box<Number> nbBox2 = new Box<>();
26       nbBox2.put(nbBox1);
27       System.out.println(nbBox2.get());
28       nbBox2.put(9);
29       nbBox2.get(nbBox1);
30       System.out.println(nbBox1.get());
31
32       Box<Integer> intBox = new Box<>();
33       nbBox.put(intBox);
34       intBox.get(nbBox);
35    }
36 }
```

a) Begründen Sie, wieso der Typparameter in Zeile 16 korrekt ist. Was gibt Zeile 17 aus? (3 Punkte)

b) Was bedeutet in Zeile 19 der Typ `Box` ohne Typparameter? Wieso ist der Typparameter in Zeile 20 korrekt? Was passiert in Zeile 21? (4 Punkte)

c) Was wird in Zeile 27 und 30 ausgegeben. (4 Punkte)

d) Wieso sind die Parameter in Zeile 33 und 34 nicht korrekt? (3 Punkte).

e) Verändern Sie die Parametertypen in Zeile 7 und 10 so, dass die Aufrufe in Zeile 33 und 34 korrekt werden. (4 Punkte).

Aufgabe 9 (12 Punkte)

Gegeben ist eine Klasse `Stadt` und eine Liste von Städten:

```
class Stadt {
   public String name;     // Name der Stadt
   public int ewz ;        // Einwohnerzahl
   public String land;

   public Stadt(String name, String land, int ewz) {
      this.name = name;
      this.land = land;
      this.ewz = ewz;
   }

   public String toString() {
      return "name=" + name + ",␣ewz=" + ewz + ",␣land=" + land;
   }
}

List<Stadt> sLst = new LinkedList<>();
sLst.add(new Stadt("Muenchen","Deutschland",1_484_226 ));
sLst.add(new Stadt("Paris","Frankreich",2_175_601));
sLst.add(new Stadt("Berlin","Deutschland",3_669_491));
sLst.add(new Stadt("Mailand","Italien",1_396_059));
sLst.add(new Stadt("Konstanz","Deutschland",84_911));
// ...
```

Lösen Sie folgende Teilaufgaben durch **Stromoperationen**.

a) Sortieren Sie die Liste von Städten alphabetisch nach dem Land und bei gleichem Land absteigend nach der Einwohnerzahl. Geben Sie die sortierten Städte aus. (3 Punkte)

b) Geben Sie die Anzahl der Städte in Italien mit weniger als 100.000 Einwohner aus. (3 Punkte)

c) Bestimmen Sie die Stadt mit der größten Einwohnerzahl und geben Sie sie aus. (3 Punkte)

d) Was gibt `println` allgemein aus? (3 Punkte).

```
BinaryOperator<Integer> fun = (x, y) -> x >= y ? x : y;
System.out.println(sLst.stream()
   .map(s -> s.ewz)
   .reduce(Integer.MIN_VALUE,fun));
```