ALP II Objektorientierte Programmierung
SS 2012
Prof. Dr. Margarita Esponda

Klausur

Name: ..... **Vorname:**..... **Matrikel-Nummer:**.....

Ich bin mit der Veröffentlichung der Klausurergebnisse mit Matrikel-Nummer und Note im Internet einverstanden:

☐ ja ☐ nein

Unterschrift:.....

Die maximale Punktzahl ist 100.

Aufgabe	**A1**	**A2**	**A3**	**A4**	**A5**	**A6**	**Summe**	**Note**
Max. Punkte	18	20	12	10	26	14		
Punkte								

Viel Erfolg!

1. Aufgabe (18 Punkte)

Eine Mersenne-Zahl hat die Form $2^n - 1 \quad mit \quad n \in \mathrm{N}$

a) (6 Punkte) Schreiben Sie eine iterative Funktion in Python, die mit Hilfe einer **for**-Schleife bei Eingabe einer natürlichen Zahl ***n*** die entsprechende Mersenne-Zahl berechnet.

b) (6 Punkte) Schreiben Sie eine Funktion, die mit Hilfe einer rekursiven (nicht endrekursive) Funktion für die Potenz-Berechnung die Mersenne-Zahlen berechnet.

c) (6 Punkte) Schreiben Sie eine Funktion, die mit Hilfe einer endrekursiven Funktion die Berechnung realisiert.

In keinem der drei Fälle dürfen Sie den Potenz-Operator verwenden.

2. Aufgabe (20 Punkte)

Gegeben sei folgendes Python-Programm, welches die Quersumme der Ziffern einer beliebigen nicht-negativen ganzen Zahl berechnet:

```
n = int(input("n = ")) # Eingabe
```

$\#\{P\} \equiv \{n \geq 0\}$

```
hilf = n
qsum = 0
while hilf>0:
    qsum = qsum + hilf % 10
    hilf = hilf // 10
```

$\#\{Q\} \equiv \{qsum == q(n)\}$

Dabei sei $q : \mathbb{N}_0 \rightarrow \mathbb{N}_0$ die Funktion, die jeder nicht-negativen ganzen Zahl die Quersumme ihrer Dezimaldarstellung zuordnet.

Für $z = a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + ... + a_1 \cdot 10^1 + a_0$ mit $n \in \mathbb{N}_0$ und $a_0,...,a_n \in \{0,1,...,9\}$ ist

$q(z) = a_n + a_{n-1} + ... + a_1 + a_0$

$z \ // \ 10 = a_n \cdot 10^{n-1} + a_{n-1} \cdot 10^{n-2} + ... + a_1$ (ganzzahlige Division)

$z \ \% \ 10 = a_0$ (Modulo-Operation)

Für alle $z \in \mathbb{N}_0$ gilt daher $q(z) = q(z \ // \ 10) \ + \ z \ \% \ 10$

a) (12 Punkte) Beweisen Sie, dass $\{INV\} \equiv \{hilf \geq 0 \wedge q(n) == qsum + q(hilf)\}$ eine gültige Invariante der **while**-Schleife ist.

b) (7 Punkte) Beweisen Sie die partielle Korrektheit der vorgegebenen Programmformel.

c) (1 Punkte) Was muss noch bewiesen werden, um die totale Korrektheit des Programms zu beweisen?

3. Aufgabe (12 Punkte)

a) (8 Punkte) Erläutern Sie die einzelnen Schritte des Countingsort-Algorithmus, indem Sie folgende Zahlen damit sortieren. Der Wertebereich der Zahlen liegt, wie man sieht, nur zwischen 0 und 3.

0	2	3	1	0	1	2	0	0	2	3

b) (2 Punkte) Ist der Counting-Sort-Algorithmus stabil? Warum?

c) (2 Punkte) Analysieren Sie die Laufzeitkomplexität und den Speicherverbrauch des Algorithmus unter Verwendung der O-Notation.

4. Aufgabe (10 Punkte)

Die Klassen **Integer** und **Double** sind Unterklassen der Klasse **Number** innerhalb der Java-Standard-Bibliothek.

Welche der folgenden Zuweisungen sind 1) **legal**, 2) **legal, aber verursachen Laufzeitfehler** oder 3) **illegal**? Begründen Sie kurz Ihre Antwort.

Legal bedeutet hier, dass die Zuweisung vom Übersetzer akzeptiert wird.

a) Integer n = new Number(3);

b) ArrayList<Number> list = new ArrayList<Double>();

c) Number[] nums = new Integer[25];

d) ArrayList<? extends Number> list = new ArrayList<Integer>(100);

e) ArrayList<? super Object> list = new ArrayList<Double>(20);

5. Aufgabe (26 Punkte)

a) Vervollständigen Sie die **ArrayQueue**-Klasse, die mit Hilfe einen dynamischen Array und Wrap-around-Strategie eine Warteschlange simuliert, indem folgendes Interface implementiert wird:

```
public interface Queue<T> {
        public void enqueue( T newElement ) ;
        public T dequeue() throws EmptyQueueException;
        public boolean empty();
        public Iterator<T> iterator();
}
```

b) Die **EmptyQueueException**-Klasse für die entsprechende Behandlung von Ausnahmefällen soll auch definiert werden.

c) Die **Iterator**-Objekte sollen mit Hilfe einer inneren Klasse erzeugt werden, die wiederum folgendes Interface implementiert:

```
public interface Iterator <T> {
      public boolean hasNext();
      public Item next();
}
...
```

```
public class ArrayQueue<T> implements Queue<T> {

        private int head;
        private int tail;
        private T[] queue;

        public ArrayQueue( int size ) {
                head = tail = 0;
                queue = (T[]) new Object [ size ];
        }
        // . . .  diese Klasse vervollständigen!
}
```

6. Aufgabe (14 Punkte)

a) Ergänzen Sie die BinarySearchTree-Klasse um folgende Methoden:

```
/* liefert den kleinsten gespeicherten Schlüssel des Baums */
    public K min(TreeNode node);

/* berechnet die gesamte Anzahl der Knoten des Baums t = |t| */
    public int abs(TreeNode node);
```

Ein Binärbaum **t** heisst perfekt balanciert, wenn an jedem Knoten die Anzahl der Knoten im linken und im rechten Unterbaum sich um höchstens Eins unterscheidet.

b) Implementieren Sie folgende rekursive Funktion, die bei Eingabe eines beliebigen Knotens überprüft, ob der darunter liegende Baum perfekt balanciert ist:

```
        public boolean perfectlyBalanced(TreeNode node);


public class BinarySearchTree<K extends Comparable<K>> {

        private TreeNode root;

        public BinarySearchTree() {
                root = null;
        }
        . . .
        . . .
        . . .
        private class TreeNode {
                private K key;
                private TreeNode left, right;

                TreeNode(K key) {
                        this.key = key;
                }
        } // end of class TreeNode

} // end of class BinarySearchTree
```