| | |
|---|---|
| **Matrikelnummer:** | Beispielklausur |

| | |
|---|---|
| **Fachrichtung:** | Wirtschaftsinformatik |
| **Kurs:** | Beispiel Klausur |
| **Studienhalbjahr:** | 2 |

| | |
|---|---|
| **Zur Vorlesung** | Datenstrukturen & Algorithmen |
| **Dozent:** | Matthias Berg-Neels |
| **Datum:** | |
| **Bearbeitungszeit:** | 45 Minuten |
| **Hilfsmittel:** | keine |

| **Aufgabenblock:** | **1** | **2** | | | **Σ** |
|---|---|---|---|---|---|
| **Punkte:** | 22 | 23 | | | 45 |
| **Erreicht:** | | | | | |

**Weitere Hinweise:**

- Bitte schreiben Sie leserlich
- Keine Hilfsmittel
- Bitte unternehmen Sie keine Täuschungsversuche

**Viel Erfolg !!!**

## Aufgabenblock 1: Datenstrukturen

**Aufgabe 1 (4 Punkte)**
Beschreiben Sie die Funktionsweise einer Queue und deren wichtigsten Methoden.

**Aufgabe 2 (4 Punkte)**
In Ihrem Studium hören Sie immer wieder interessante neue Worte, die sie gerne nachschlagen möchten. Um dies zu tun, haben Sie sich einen Stapel angelegt. Immer wenn Sie ein neues Wort hören legen Sie dies auf den Stapel („push“). Wenn Sie Zeit haben nehmen Sie das oberste Wort vom Stapel herunter („pop“) um es nachzuschlagen. Wenn Sie nicht sicher sind ob Ihre Zeit gerade reicht schauen Sie sich das oberste Wort erst einmal nur an („peek“) und entscheiden ob sie es jetzt oder später nachschlagen.
Vervollständigen Sie das folgende Coding der Klasse „WordsIWantToLearn“ um die beschriebenen drei Methoden.

```
public class WordsIWantToLearn{

	Element peekElement = null;


	public void push(String data){



	}

	public String pop(){




	}

	public String peek(){



	}
```

```
	private class Element{
		private String data;
		private Element prevElement;

		public Element(String data, Element prevElement) {
			this.data = data;
			this.prevElement = prevElement;
		}

		public String getData(){
			return data;
		}

		public Element getPrevElement(){
			return prevElement;
		}
	}
}
```

**Aufgabe 3 (14 Punkte)**
Aus dem folgenden Codingblock wird der Binärbaum aus „Abbildung 1“ erzeugt. Die Implementierung der Klasse „Tree“ finden Sie in Anhang A.

```
	// Erzeuge Baum
	Tree<String> names = new Tree<String>();

	names.insert("Justus");
	names.insert("Tim");
	names.insert("Gabi");
```

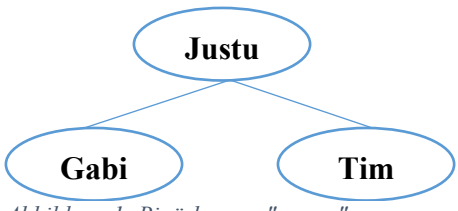

*Abbildung 1: Binärbaum - "names"*

a) Zeichnen Sie auf, wie der Baum nach Ausführung des folgenden Coding-Blockes aussieht.

```
// Coding Block Aufgabenteil a)
names.insert("Ina");
names.insert("Adam");
names.insert("Hans");
names.insert("Vicki");
```

b) Zeichnen Sie auf, wie der resultierende Baum aus Aufgabe a) nach dem ausführen des folgenden Coding-Blockes aussieht.

```
// Coding Block Aufgabenteil b)
names.insert("Isa");
names.delete("Tim");
names.insert("Dori");
```

c) Implementieren Sie die fehlende Methode „toString(Node<V> node)“ um alle Elemente in der korrekten Reihenfolge auszugeben. Nutzen Sie hierbei eine rekursive Implementierung.

```
@Override
public String toString() {
	String s = "";

	if (root != null) {
		s = toString(root);
	} else {
		s = "Empty Tree";
	}
	return s;
}

private String toString(Node<V> node) {









}
```

## Aufgabenblock 2: Algorithmen / Sortieren

**Aufgabe 4 (4 Punkte)**
Beschreiben Sie die Begriffe der Iteration und der Rekursion.

**Aufgabe 5 (6 Punkte)**
Gegeben sind die folgenden beiden Zahlenreihen. Sortieren Sie beide Reihen nach den dazugehörigen Angaben in dem Sie jeden einzelnen Tauschschritt entsprechend darstellen.

| Bubble-Sort-Verfahren - aufsteigen | Insertion-Sort-Verfahren - absteigend |
|---|---|
| 48 \| 52 \| 25 \| 92 \| 23 \| 51 | 27 \| 5 \| 43 \| 11 \| 51 \| 36 |

**Aufgabe 6 (13 Punkte)**
Die Klasse „Sort“ stellt zwei Methoden zum Sortieren von Integer-Arrays zur Verfügung und gibt das sortierte Array wieder zurück. Implementieren Sie die Methoden  für das Bubble-Sort- und für Selection-Sort-Verfahren. Alternativ können Sie auch die Struktogramme der Methoden aufzeichnen.
Hinweis: Nutzen Sie die Methode „swapElements“ zum Tausch von Array-Elementen.

```
public class Sort {

	private static void swapElements(int[] array, int a, int b){
		int buffer = array[a];
		array[a] = array[b];
		array[b] = buffer;
	}

	// Bubble-Sort-Verfahren: sortiert aufsteigend
	public static int[] bubbleSort(int[] array){









	}



	// Selection-Sort → nächste Seite
```

```
        // Selection-Sort-Verfahren: sortiert aufsteigend
        public static int[] selectionSort(int[] array){









        }
}
```

## Anhang A

```
public class Tree<V> {

	Node<V> root = null;

	public void insert(V data) {
		if (root == null) {
			root = new Node<V>(data);
		} else {
			insert(root, data);
		}
	}

	@SuppressWarnings("unchecked")
	private void insert(Node<V> node, V data) {
		if (((Comparable<V>)node.getData()).compareTo((V) data) > 0) {
			if (node.getLeft() == null) {
				node.setLeft(new Node<V>(data));
			} else {
				insert(node.getLeft(), data);
			}
		} else if (((Comparable<V>)node.getData())
					.compareTo((V) data) < 0) {
			if (node.getRight() == null) {
				node.setRight(new Node<V>(data));
			} else {
				insert(node.getRight(), data);
			}
		}
	}



	@Override
	public String toString() {
		String s = "";

		if (root != null) {
			s = toString(root);
		} else {
			s = "Empty Tree";
		}
		return s;
	}

	private String toString(Node<V> node) {
		// Aufgabenteil c)
	}
```

```java
public void delete(V data) {
	root = deleteNode(root, data);
}


@SuppressWarnings("unchecked")
private Node<V> deleteNode(Node<V> node, V data) {
	if (node != null) {
		if (data.equals(node.getData())) {
			if ((node.getLeft() == null)
				&& (node.getRight() == null)) {
				node = null;
			} else {
				if (node.getRight() == null)
					node = node.getLeft();
				else {
					if (node.getLeft() == null)
						node = node.getRight();
					else {
						Node<V> preRight = null;
						Node<V> right = node.getLeft();
						while (right.getRight() != null){
							preRight = right;
							right = right.getRight();
						}
						if (preRight != null) {
							preRight
							.setRight(right.getLeft());
							right
							.setLeft(node.getLeft());
						}
						right.setRight(node.getRight());
						node = right;
					}
				}
			}
		} else {
			if (((Comparable<V>) data)
						.compareTo((V) node.getData()) > 0)
					node.setRight(deleteNode(node
								.getRight(), data));
			else
				node
				.setLeft(deleteNode(node.getLeft(), data));
		}
	}
	return node;
}
```

```
	private class Node<T> {
		T data;
		Node<T> left;
		Node<T> right;

		public Node(T data) {
			this.data = data;
		}

		public T getData() {
			return data;
		}

		public Node<T> getLeft() {
			return left;
		}

		public Node<T> getRight() {
			return right;
		}

		public void setLeft(Node<T> left) {
			this.left = left;
		}

		public void setRight(Node<T> right) {
			this.right = right;
		}



	}
}
```