



# Einführung in die Informatik - Vertiefung Probeklausur

Sommersemester 2015

**Hinweis:** Diese Probeklausur enthält eine kleine Sammlung an Aufgaben, deren Schwierigkeitsgrad etwa dem der schriftlichen Prüfung des Moduls Einführung in die Informatik Vertiefung entspricht. Die Aufgaben decken nicht alle behandelten Themenbereiche ab und der Umfang der Probeklausur entspricht nicht dem der echten Prüfung.

Musterlösung  
Stand: 16. Juli 2015

---

**Aufgabe 1 Allgemeine Fragen.**

1. **Teilaufgabe:** Womit kann man einen Booleschen Ausdruck im Allgemeinen nicht in seine minimale Form umwandeln?
  - Ablesen aus Wahrheitstabellen.
  - Verfahren von Quine und McCluskey.
  - KV-Diagramme.
  - Anwendung Boolescher Axiome.
  
2. **Teilaufgabe:** Welche der folgenden Komplexitätsklassen ist so groß, dass die anderen drei angegebenen Klassen darin enthalten sind?
  - $O(n^2)$
  - $O(1^n)$
  - $O(n \log(n))$
  - $O(n)$
  
3. **Teilaufgabe:** Wonach werden Objekte von allgemeinen Datentypen üblicherweise sortiert?
  - Objekte allgemeiner Datentypen können nicht sortiert werden.
  - Nach der Reihenfolge des Operators  $\leq$ , welcher auch für allgemeine Datentypen definiert ist.
  - Nach einem zu definierenden Schlüssel.
  - Nach dem ersten Ganzzahl- oder Fließkomma-Attribut.
  
4. **Teilaufgabe:** Was ist eine generische Klasse?
  - Eine Klasse, von der nicht geerbt werden kann.
  - Eine Klasse, die einen Typen als Parameter besitzt, der zur Laufzeit verändert werden kann.
  - Eine Klasse, die einen Typen als Parameter besitzt, der bei der Instanziierung dieser Klasse festgelegt wird.
  - Ein primitiver Datentyp.
  
5. **Teilaufgabe:** Welche Interface(s) braucht man, um eine nicht-abstrakte iterierbare Klasse zu implementieren?
  - `Iterator<E>`, `Iterable<T>`
  - `Comparable<T>`, `Iterator<E>`
  - `Iterable<T>`
  - `Iterator<E>`

6. **Teilaufgabe:** Worin unterscheidet sich eine doppelt verkettete Liste im Vergleich zur einfach verketteten Liste?
- n
- Jeder Knoten zeigt auf seine zwei nachfolgenden Knoten.
  - Jeder Knoten speichert zusätzlich zu seinem eigenen Wert auch den Wert des Vorgängers.
  - Sie unterscheidet sich lediglich in einer Tail-Referenz, die auf das letzte Element der Liste zeigt.
  - Jeder Knoten zeigt auf seinen Vorgänger und Nachfolger.
7. **Teilaufgabe:** Wo steht in einem Max-Heap das größte Element?
- Im linken äußersten Blatt des Baumes.
  - Im rechten äußersten Blatt des Baumes.
  - Das ist in einem Heap nicht genau definiert.
  - In der Wurzel des Baumes.
8. **Teilaufgabe:** Wenn  $x$  der linke Nachfolger von  $y$  in einem binären Suchbaum ist, dann gilt:
- $\text{key}(x) \leq \text{key}(y)$
  - $\text{key}(x) < \text{key}(y)$
  - $\text{key}(x) \geq \text{key}(y)$
  - $\text{key}(x) > \text{key}(y)$
9. **Teilaufgabe:** Welche der folgenden Datenstrukturen ist linear?
- Graph.
  - Queue.
  - AVL-Baum.
  - Heap.
10. **Teilaufgabe:** Was gilt für AVL-Bäume?
- Es werden immer Rotationen beim Einfügen oder Löschen von Elementen benötigt.
  - Sie sind linksvoll.
  - Sie verhindern eine Degeneration zu einer Liste.
  - Sie verhalten sich wie Listen.
11. **Teilaufgabe:** Wie viele Einsen werden für den Ausdruck  $\bar{x} \cdot y \cdot z$  in eine KV-Tafel mit 5 Eingangsvariablen eingetragen?
- 8
  - 1
  - 4
  - 2
12. **Teilaufgabe:** Welche Aussage zu Heapsort ist richtig?
- Heapsort hat im Average-Case die Komplexität  $O(n)$ .
  - Bei Heapsort werden Sift-Down und Sift-Up benötigt.
  - Heapsort führt abwechselnd nacheinander Swaps und Heapifys aus.
  - Heapsort kann manche Arrays nicht sortieren, da diese als Heap vorliegen müssen.

13. **Teilaufgabe:** Wozu dient die Methode `iterator()` des Interfaces `Iterable`?

- Sie ruft die `for-each` Schleife auf und sorgt somit dafür, dass die Iteration einmal vollständig durchgeführt wird.
- Sie erzeugt ein neues Iteratorobjekt und gibt dessen Referenz zurück.
- Sie gibt `true` zurück, falls ein Iterator noch Elemente enthält.
- Sie überprüft, welche der implementierten Iteratorklassen verwendet werden soll und gibt einen entsprechenden String zurück.

14. **Teilaufgabe:** Welche Wege findet der Dijkstra-Algorithmus ?

- die kürzesten Wege
- die schönsten Wege
- die längsten Wege
- einen optimalen Weg über alle Knoten im Graph

**Aufgabe 2 Boolesche Algebra.**

1. **Teilaufgabe:** Sind die folgenden Booleschen Ausdrücke äquivalent?

$$f(x, y) = (x \Rightarrow y) + (y \Rightarrow x)$$

$$g(x, y) = (x + y) \Rightarrow (x \cdot y).$$

Beweisen oder widerlegen Sie die Behauptung mit der Wahrheitstafelmethode. Die Zwischenschritte müssen erkennbar sein.

*Hinweis:* Es gilt  $x \Rightarrow y := \bar{x} + y$ .

**Lösung:**

x	y	$(x \Rightarrow y)$				$(y \Rightarrow x)$				$(x + y) \Rightarrow (x \cdot y)$				
0	0	0	1	0	1	0	1	0	0	0	1	0	0	0
0	1	0	1	1	1	1	0	0	0	1	1	0	0	1
1	0	1	0	0	1	0	1	1	1	1	0	0	1	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Wie man sieht, sind die Ausdrücke *nicht* äquivalent.

2. **Teilaufgabe:** Wandeln Sie den Booleschen Ausdruck der Funktion:

$$f(x, y, z) = \overline{(x \equiv y)} + z$$

mit Hilfe der algebraischen Umformung in eine aKNF um. Die Zwischenschritte müssen erkennbar sein.

*Hinweis:* Es gilt  $x \equiv y := (x \cdot y) + (\bar{x} \cdot \bar{y})$ .

**Lösung:**

$$f(x, y, z) = \overline{(x \equiv y)} + z$$

$$= \overline{((x \cdot y) + (\bar{x} \cdot \bar{y}))} + z$$

$$= ((\bar{x} \cdot y) + (x \cdot \bar{y})) + z$$

$$= ((\bar{x} + y) \cdot (x + \bar{y})) + z$$

$$= (\bar{x} + \bar{y} + z) \cdot (x + y + z)$$

3. **Teilaufgabe:** Wandeln Sie den Booleschen Ausdruck der Funktion:

$$f(x, y, z, w) = (((x \cdot y) + x + (z + w)) \cdot (w + z))$$

zunächst in eine DNF und anschließend mit Hilfe einer KV-Tafel in eine minimale DNF um. Die Zwischenschritte müssen erkennbar sein.

**Lösung:**

Für:  $f(x, y, z, w) = (((x \cdot y) + x + (z + w)) \cdot (w + z))$  ergibt sich folgendes KV-Diagramm:

$f(x, y, z, w) :$

		-----  $y$	
		-----  $x$	
		0	0
		0	0
		1	1
		1	1
	-----  $z$	1	1
	-----  $w$	1	1
		1	1
		1	1

Vereinfacht ergibt sich:

DNF:  $f(x, y, z, w) = (w + z)$

**Aufgabe 3 Komplexität.**

- Bestimmen Sie eine Formel für den Aufwand  $T_g(n)$  der folgenden Methode  $g(n)$ . Dabei soll für die Berechnung des Zeitaufwands nur in Zeile 7 der Funktionsaufruf  $\text{fun}(i)$  berücksichtigt werden. Die Funktion  $\text{fun}(i)$  weise hierbei einen Aufwand von  $T_{\text{fun}}(i) = 1$  für alle  $i$  auf.

```

1 public void g(int n) {
2     int i = 0;
3     int j = 0;
4     while (i < n) {
5         j = i;
6         while (j == i) {
7             fun(i);
8             j++;
9         }
10        i++;
11    }
12 }

```

- Nennen Sie die Komplexitätsklasse, in welcher sich  $T_g(n)$  befindet und markieren Sie in der folgenden Tabelle die Komplexitätsklassen zu denen  $T_g(n)$  gehört.

**Hinweis:** Ein Beweis ist nicht gefordert.

**Lösung:**

$$T_g(n) = \sum_{i=0}^{n-1} 1 \cdot T_{\text{fun}}(i) = n \cdot 1 = n$$

Ordnung	$T_g$ fällt in diese Klasse
$\mathcal{O}(1)$	
$\mathcal{O}(\log(n))$	
$\mathcal{O}(n)$	x
$\mathcal{O}(n \log(n))$	x
$\mathcal{O}(n^2)$	x
$\mathcal{O}(n^2 \log(n))$	x
$\mathcal{O}(n^3)$	x
$\mathcal{O}(n^p)$ mit $p > 3$	x
$\mathcal{O}(p^n)$	x

**Aufgabe 4 Heapsort.**

1. **Teilaufgabe:** Gegeben sei die Zahlenfolge

$$F_1 = 47, 21, 35, 34, 59, 22, 36, 58, 60, 23$$

Geben sie einen Binärbaum an, der die Elemente der Folge  $F_1$  enthält und die (Max-)Heap Eigenschaft erfüllt. Geben Sie zusätzlich dazu den zum Heap gehörigen Array an.

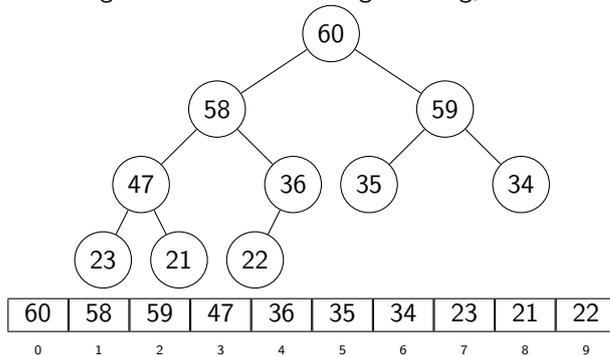
**Lösung:**

Wir lösen diese Aufgabe unter Beachtung der Heapbedingung für Binärheaps. Sie lautet für Maxheaps:

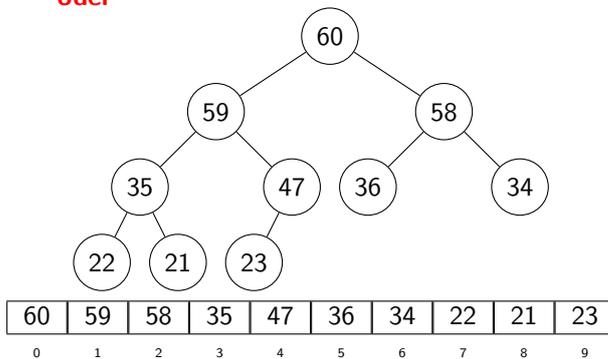
$$B \text{ ist Kindknoten von } A \Leftrightarrow \text{Key}(A) \geq \text{Key}(B)$$

Wir suchen uns also die größte Zahl der Zahlenfolge  $F_1$  (hier: 60) als erstes Element und bauen den Heap dann von oben nach unten (und dann in den einzelnen Ebenen von links nach rechts) auf. Gleichzeitig füllen wir das Array auf. Das erste Element hat den Index 0. Wenn wir immer die nächstkleinere noch nicht benutzte Zahl der Zahlenfolge  $F_1$  als nächstes Element fortlaufend in das Array (bzw. in den Heap) einfügen, wird die Heapbedingung nie verletzt. Es ist außerdem darauf zu achten, dass der fertige Heap ein linksvoller Baum sein sollte, damit die Arraydarstellung auch Sinn macht.

Diese Aufgabe hat keine eindeutige Lösung, denkbare Lösungen sind aber z.B:



oder



2. **Teilaufgabe:** Gegeben sei die Zahlenfolge

$$F_2 = 10, 9, 6, 8, 7, 2, 5, 1, 4, 3.$$

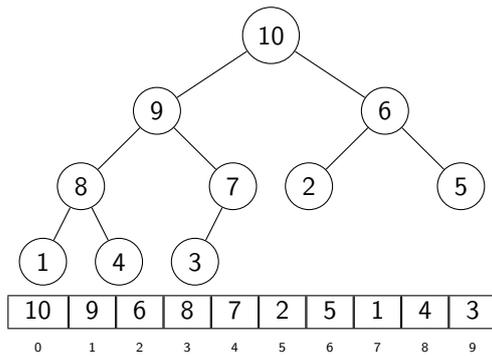
Sortieren Sie die Folge  $F_2$  mit Heapsort. Stellen Sie nach jeder Iteration den Restheap als Baum und die gesamte Zahlenfolge als Array dar.

*Hinweis:* Die Zahlenfolge  $F_2$  ist ein Heap.

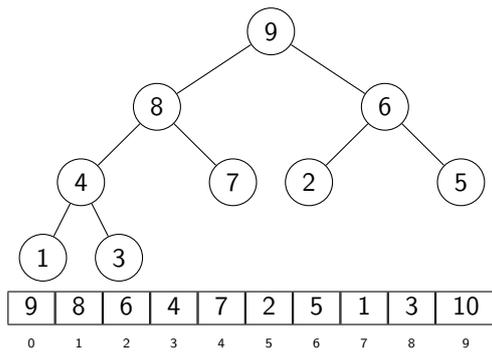
**Lösung:**

Heapsort kann nur sinnvoll auf einen Heap angewendet werden. Der Hinweis versichert, dass  $F_2$  einer ist. Der Heapsort-Algorithmus ist ausführlich in Tutoriumsvorbereitung # 8 zu finden.

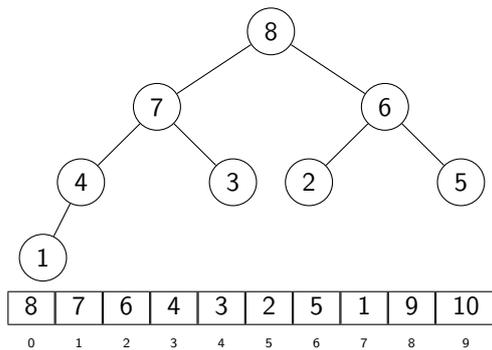
Ausgangsheap:



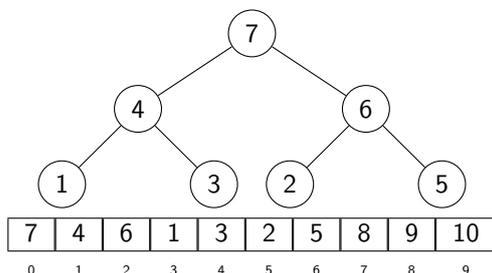
Tausche 3 mit 10 und führe Sift-Down auf die neue Wurzel durch ( tausche sie also mit 9,8,4 ).



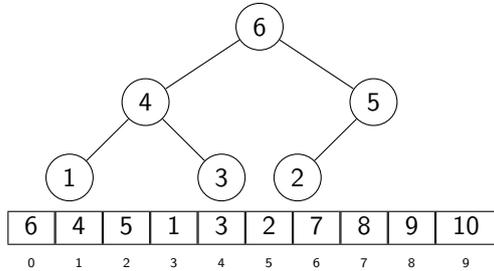
Tausche 3 mit 9 und führe Sift-Down auf die neue Wurzel durch ( tausche sie also mit 8,7 ).



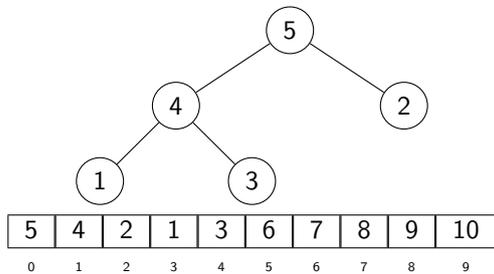
Tausche 1 mit 8 und führe Sift-Down auf die neue Wurzel durch ( tausche sie also mit 7,4 ).



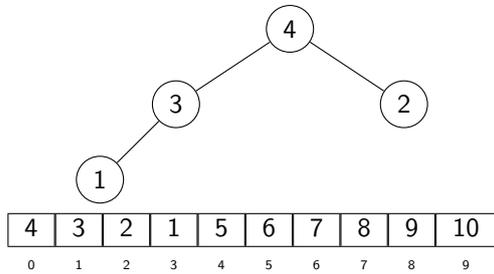
Tausche 5 mit 7 und führe Sift-Down auf die neue Wurzel durch ( tausche sie also mit 6 ).



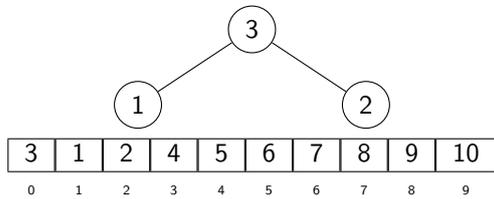
Tausche 2 mit 6 und führe Sift-Down auf die neue Wurzel durch ( tausche sie also mit 5 ).



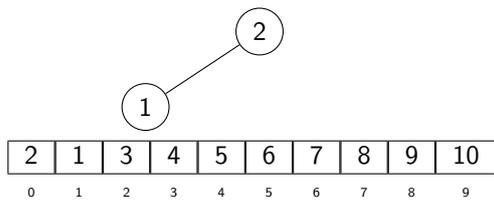
Tausche 3 mit 5 und führe Sift-Down auf die neue Wurzel durch ( tausche sie also mit 4 ).



Tausche 1 mit 4 und führe Sift-Down auf die neue Wurzel durch ( tausche sie also mit 3 ).



Tausche 2 mit 3 und führe Sift-Down auf die neue Wurzel durch ( kein Tausch notwendig ).



Tausche 1 mit 2 und führe Sift-Down auf die neue Wurzel durch ( kein Tausch notwendig ).



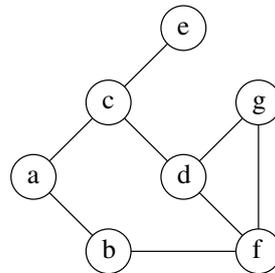
①

1	2	3	4	5	6	7	8	9	10
0	1	2	3	4	5	6	7	8	9

Fertig!

**Aufgabe 5** Tiefensuche.

Betrachten Sie den folgenden Graphen:



1. **Teilaufgabe:** Welche Datenstruktur verwendet die Tiefensuche? Wie lautet das Speicherprinzip dieses Datentyps?

**Lösung:**

Stack, LIFO (last in, first out)

2. **Teilaufgabe:** Traversieren Sie den Graphen  $G$  mit Tiefensuche. Führen Sie dazu eine Handsimulation mit Hilfe der untenstehenden Tabelle durch. Dabei bezeichne *Schritt* die Nummer des aktuellen Schleifendurchlaufs und *AK* den aktuellen Knoten. Beachten Sie bei der Handsimulation Folgendes:

- Startknoten ist der Knoten mit Bezeichner  $a$ , welcher sich nach Initialisierung (Schritt 0) im Stack befindet.
- Geben Sie für  $Schritt > 0$  den Inhalt des Stacks jeweils am Ende des aktuellen Schleifendurchlaufs an.
- Fügen Sie pro Schleifendurchlauf jeweils alle weißen Nachfolger von  $AK$  stets in *alphabetisch aufsteigender* Reihenfolge in den Stack ein.
- Die schwarze Liste enthält alle Knoten, die schon abgearbeitet worden sind. Fügen Sie einen Knoten in dem selben Schleifendurchlauf in die Schwarze Liste ein, in welchem alle seine Nachfolger-Knoten in den Stack eingefügt wurden.

Schritt	AK	Stack	schwarze Liste
0	-	a	-

**Lösung:**

Schritt	AK	Stack	schwarze Liste
0		a	
1	a	c,b	a
2	c	e,d,b	a,c
3	e	d,b	a,c,e
4	d	g,f,b	a,c,e,d
5	g	f,b	a,c,e,d,g
6	f	b	a,c,e,d,g,f
7	b		a,c,e,d,g,f,b

**Aufgabe 6 Mergesort.**

Implementieren Sie eine Java-Methode `public void mergesort(int[] arr)` die das Sortierverfahren Mergesort für ganze Zahlen realisiert und gegebenenfalls benötigte Hilfsmethoden.

**Lösung:**

```
1
2 public void mergesort(int [] arr) {
3     divide(arr, 0, arr.length - 1);
4 }
5
6 /*
7  Methode divide: "Teilt" das Array in Teilarrays durch rekursiven Aufruf
8  @param arr Array der zu sortierenden Elemente
9  @param start Startindex
10 @param end Endindex
11 */
12 private static void divide(int[] arr, int start, int end) {
13     if (start < end) {
14         int mid = (start + end + 1) / 2; //Mitten-Element wird bestimmt
15         divide(arr, start, mid - 1);
16         divide(arr, mid, end);
17         merge(arr, start, end, mid);
18     }
19 }
20
21 /*Methode merge: Fügt zwei sortierte Arrays zusammen
22 @param arr: zweigeteiltes Array (Teilung durch Teilungsindex realisiert)
23 @param start: Startindex der ersten Menge
24 @param end: Endindex der zweiten Menge
25 @param part: Der Teilungsindex der beiden Sets*/
26
27 private static void merge(int [] arr, int start, int end, int part) {
28     if( start < end ){
29         int i, j, k;
30         i = 0; //Variable für Indizes im Hilfsarray b
31         j = start; //Variable für Indizes im Array arr
32
33         int [] b = new int[part - start]; //Neues Array mit der Laenge des 1.Sets
34         //wird angelegt
35         while (j < part){ //Kopiere das erste Set in Hilfsarray b um
36             //Ueberschreiben zu vermeiden
37             b[i] = arr[j];
38             j++;
39             i++;
40         }
41         //j = part, also nun Variable auf das 2.Set von arr
42         i = 0; //weiterhin Variable für Indizes im Hilfsarray b
43         k = start; //Variable der von start ab durchzählt (Index in dem das
44         //kleinere Element aus den ursprünglich beiden Sets gespeichert wird)
45         //Zurueckkopieren der Elemente, die kleiner sind
46         while (k < j && j <= end){
47             if (b[i]< arr[j]){
48                 arr[k++] = b[i++];
49             }
50             else{
51                 arr[k++] = arr[j++];
52             }
53         }
54     }
55 }
```

```
46         }
47
48         else {
49             arr[k++] = arr[j++];
50         }
51     }
52
53     //Zurueckkopieren des ueberbleibenden Rests
54     while (k < j){
55         arr[k++] = b[i++];
56     }
57 }
58 }
```

**Aufgabe 7 Generische Klassen.**

1. **Teilaufgabe:** Es seien  $\mathcal{X}$  und  $\mathcal{Y}$  zwei Mengen beliebigen Typs. Implementieren Sie ein generisches Interface mit dem Namen `Function`, das eine Funktion  $f : \mathcal{X} \rightarrow \mathcal{Y}$  repräsentiert. Dabei sollen die Mengen  $\mathcal{X}$  und  $\mathcal{Y}$  generisch verschiedenen Datentypen angehören können. Ausserdem fordert das Interface eine Methode `apply`, die bei Eingabe eines Elementes  $x \in \mathcal{X}$  den Funktionswert  $y = f(x) \in \mathcal{Y}$  zurück liefert.

*Hinweis:* mit der Schreibweise  $\langle T_1, T_2, \dots, T_n \rangle$  kann man ein generischer Datentyp mit  $n$  Typvariablen  $T_1, T_2, \dots, T_n$  definiert werden.

**Lösung:**

Wir unterteilen die Aufgabenstellung in drei Schritte

- Zuerst erstellen wir ein Interface mit dem Namen `Function`,
- dann erweitern wir das Interface um die generischen Datentypen  $X$  und  $Y$
- und zuletzt erweitern wir das Interface um die Methode `apply`, die als Parameter den Datentypen  $X$  und als Rückgabetypen den Datentypen  $Y$  bekommt:

```
1 public interface Function<X,Y>{  
2     Y apply(X value);  
3 }
```

2. **Teilaufgabe:** Implementieren Sie eine Klasse `Length`, die eine Funktion  $f : \text{String} \rightarrow \text{Integer}$  repräsentiert. Die Klasse implementiert das Interface `Function`. Die Methode mit dem Name `apply` erhält als Eingabe ein Objekt vom Typ `String` und liefert die Länge des Strings als Objekt der Klasse `Integer` zurück.

*Hinweis:* Für ein Objekt `str` der Klasse `String` liefert die Methode `int length()` die Länge (Anzahl der Zeichen) von `str` zurück.

**Lösung:**

Wir können die Aufgabenstellung wieder in mehrer Schritte zerlegen.

- (a) Eine Klasse `Length` anlegen.
- (b) Klasse `Length` erweitern, damit es das Interface `Function` implementieren muss.
- (c) Die Methode `apply` implementieren

```
1 public class Length implements Function<String,Integer>{  
2     public Integer apply(String value){  
3         return new Integer(value.length());  
4     }  
5 }
```

3. **Teilaufgabe:** Implementieren Sie eine Klasse `Test`. Diese Klasse besitzt eine `main()`-Methode, die folgenden Ablauf realisiert:

- Es wird ein Objekt der Klasse `Length` erzeugt.
- Das Objekt ruft die Methode `apply()` für die Eingabe des Strings "Inftech" auf.
- Der resultierende Funktionswert wird auf der Konsole ausgegeben.

**Lösung:**

Hier können wir direkt nach Stichpunkten gehen die wir vorgegeben haben.

```
1 public class Test{
2     public static void main(String[] args){
3         Length len = new Length();
4         System.out.println("Laenge des Objektes: "+len.apply("Inftech"))
5     }
6 }
```

*Als Ausgaben würden wir dies hier erhalten. War aber nicht in der Aufgabe explizit gefordert.*

```
1 user@PC:~/Probeklausur/code$ java Test
2 Laenge des Objektes: 7
```

**Aufgabe 8 Listen.**

Betrachten Sie das folgende unvollständige Java-Programm für die doppelt verkettete Liste:

```
1 public class DoppeltVerketteteListe<T> {
2
3     private class ListElem {
4
5         T data;
6
7         ListElem(T data) {
8             this.data = data;
9         }
10    }
11
12 }
```

**1. Teilaufgabe:**

Ergänzen Sie die Klasse `DoppeltVerketteteListe` um Referenzen auf das erste und letzte Element der Liste (head und tail). Ergänzen Sie weiter die innere Klasse `ListElem` um die benötigten Referenzen auf Vorgänger- und Nachfolge-Elemente.

**2. Teilaufgabe:**

Implementieren Sie eine Methode `public T get(int i)`, die das Datenobjekt `data` des Listerlements an der  $i$ -ten Stelle zurückgibt.

**Hinweise:**

1. Es sei  $n$  die Anzahl der Listenelemente. Der Kopf der Liste befindet sich an der 0-ten Stelle. Das letzte Listenelement befindet sich an der  $(n-1)$ -ten Stelle.

2. Gehen Sie davon aus, dass stets  $0 \leq i < n$  gilt.

**3. Teilaufgabe:**

Implementieren Sie eine Methode `public T addFirst(T data)`, die an der ersten Stelle der Liste das übergebene Datenobjekt einfügt.

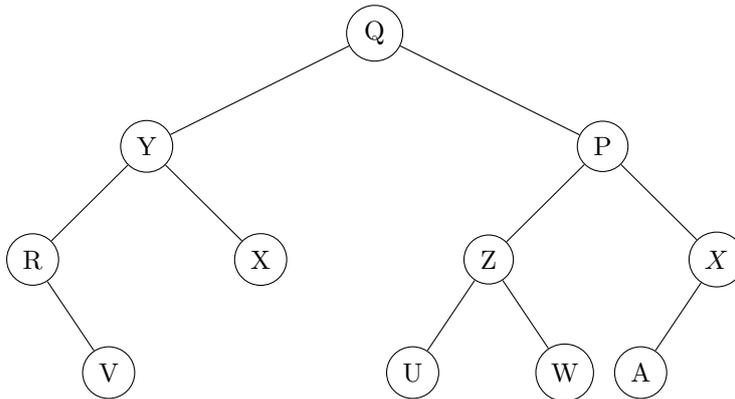
**Lösung:**

```
1 public class DoppeltVerketteteListe<T> {
2
3     private class ListElem {
4
5         T data;
6         ListElem next;
7         ListElem prev;
8
9         public ListElem(T data) {
10            this.data = data;
11        }
12    }
13 }
14 }
```

```
15 private ListElem head;
16 private ListElem tail;
17
18 public Liste() {
19     head = null;
20     tail = null;
21 }
22
23 public T get(int i) {
24     //leere Liste, kann nach Aufgabenstellung auch weggelassen werden
25     if(head == null) {
26         return null;
27     }
28     //Liste hat mind. 1 Element
29     else {
30         ListElem current = head;
31         int currentIdx = 0;
32         while(currentIdx < i) {
33             current = current.next;
34             currentIdx++;
35         }
36         return current.data;
37     }
38 }
39
40 public void addFirst(T data) {
41     Listelem neu = new Listelem(data);
42     if(head == null) {
43         head = neu;
44         tail = head;
45     }
46     //Liste hat mind. 1 Element
47     else {
48         neu.next = head;
49         neu.next.prev = neu;
50         head = neu;
51     }
52 }
53
54 }
```

**Aufgabe 9 Traversierung von Bäumen.**

1. **Teilaufgabe:** Geben Sie die entstehende Buchstabenfolge aus, wenn Sie den folgenden Binärbaum in postorder-Reihenfolge traversieren.



**Lösung:**

V, R, X, Y, U, W, Z, A, X, P, Q

2. **Teilaufgabe:** Implementieren Sie eine Java-Klasse `Tree` mit einer Unterklasse `Node` für einen Baum und für Baumknoten an. Dabei sollen folgende Bedingungen erfüllt sein:
- Die in den Knoten abgespeicherten Nutzdaten sind Elemente eines generischen Datentyps.
  - Die Klasse `Node` ist außerhalb der Klasse `Tree` unsichtbar.
  - Jeder Knoten des Baums hat beliebig viele Nachfolger.

*Es sollen nur die Attribute (und keine Methoden) der Klassen angegeben werden.*

**Lösung:**

```

1 class Tree<T> {
2     private class Node {
3         Node[] children;
4         T data;
5     }
6
7     Node root;
8 }
  
```

3. **Teilaufgabe:** Implementieren Sie für die Klasse `Tree` aus Aufgabenteil 2 eine *rekursive* Methode `int countNodes()`, welche die Anzahl der Knoten dieses Baumes zurückgibt.

**Lösung:**

```

1 int countNodes() {
2     return countNodes(root);
3 }
4
5 int countNodes(Node pos){
6     int count = 0;
7     if (pos.children != null)
  
```

```
8     for (int i=0;i<pos.children.length;i++)
9         count += countNodes(pos.children[i]);
10    return count+1;
11 }
```

**Aufgabe 10 AVL-Bäume.**

Gegeben sei eine Klasse AVLBaum, die die spezifischen Unterklassen für innere Knoten (Fork) und Blätter (Leaf) sowie eine Referenz auf das Wurzelement enthält.

```
1 public class AVLBaum<T>{
2
3     private abstract class Node{
4         public int key;
5         public int hoehe;
6
7         Node(int key){
8             this.key = key;
9             hoehe = 0;
10        }
11
12        public abstract boolean checkAVLCondition();
13
14    }
15
16    private class Fork extends Node{
17        Node links;
18        Node rechts;
19
20        Fork(int key, Node links, Node rechts){
21            super(key);
22            this.links = links;
23            this.rechts = rechts;
24            hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
25        }
26
27        public void setRechts(Node rechts){
28            this.rechts = rechts;
29            hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
30        }
31
32        public void setLinks(Node links){
33            this.links = links;
34            hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
35        }
36
37    }
38
39    private class Leaf extends Node{
40        T daten;
41
42        Leaf(int schluessel, T daten){
43            super(schluessel);
44            this.daten = daten;
45        }
46
47    }
48
49    // Wurzel des AVL-Baums
50    private Node root;
```

51 }  
52 }

1. **Teilaufgabe:** Erweitern Sie die Klasse Fork um eine Methode `public Fork rotateLeft()`, die eine Linksrotation am aufrufenden Knoten durchführt.
2. **Teilaufgabe:** Erweitern Sie die Klasse AVLTree um eine Methode `public boolean checkAVLCondition()`, die die AVL-Eigenschaft des gesamten Baumes testet.  
**Hinweis:** Für Ihre Implementierung müssen Sie die Methode `public boolean checkAVLCondition()` innerhalb der Fork- und Leaf-Klasse implementieren.

**Lösung:**

```
1 public class AVLBaum<T>{
2
3     private abstract class Node{
4         public int key;
5         public int hoehe;
6
7         Node(int key){
8             this.key = key;
9             hoehe = 0;
10        }
11
12        public abstract boolean checkAVLCondition();
13
14    }
15
16    private class Fork extends Node{
17        Node links;
18        Node rechts;
19
20        Fork(int key, Node links, Node rechts){
21            super(key);
22            this.links = links;
23            this.rechts = rechts;
24            hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
25        }
26
27        public void setRechts(Node rechts){
28            this.rechts = rechts;
29            hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
30        }
31
32        public void setLinks(Node links){
33            this.links = links;
34            hoehe = Math.max(links.hoehe, rechts.hoehe)+1;
35        }
36
37        private Fork rotateLeft() {
38            Fork b = (Fork) this.rechts;
39            this.setRechts(b.links);
40            b.setLinks(this);
41            return b;

```

```
42     }
43
44     public boolean checkAVLCondition(){
45         if((Math.abs(links.hoehe-rechts.hoehe))>1){
46             return false;
47         } else{
48             return links.checkAVLCondition() && rechts.checkAVLCondition();
49         }
50     }
51
52 }
53
54 private class Leaf extends Node{
55     T daten;
56
57     Leaf(int schluessel, T daten){
58         super(schluessel);
59         this.daten = daten;
60     }
61
62     public boolean checkAVLCondition(){
63         return true;
64     }
65 }
66
67 // Wurzel des AVL-Baums
68 private Node root;
69
70 //pruefe auf AVL-Bedingung
71 public boolean checkAVLCondition(){
72     return root.checkAVLCondition();
73 }
74
75 }
```