Fakultät IV
Elektrotechnik/Informatik

Klausur Einführung in die Informatik II (Technikorientierung)

27. Juli 2007

Name: .....

Matr.-Nr. .....

Bearbeitungszeit: 120 Minuten

Bewertung

Aufgabe	Punkte	Erreichte Punkte
1	8	
2	2	
3	10	
4	5	
5	8	
6	6	
7	14	
8	8	
9	4	
10	5	
Summe	70	

- Benutzen Sie für die Lösung der Aufgaben **nur** das mit diesem Deckblatt ausgeteilte Papier. Lösungen, die auf anderem Papier geschrieben werden, können **nicht** bewertet werden. Schreiben Sie ihre Lösung auch auf die Rückseiten der Blätter; benötigen Sie für eine Lösung mehr als ein Blatt, finden Sie am Ende der Klausur Leerblätter. Zusätzliches Papier können Sie von den Tutoren bekommen.
- Tragen Sie jetzt (vor Beginn der eigentlichen Bearbeitungszeit!) auf **allen** Blättern ihren Namen und ihre Matrikelnummer ein.
- Schreiben Sie deutlich! Unleserliche oder mehrdeutige Lösungen können nicht gewertet werden.
- Schreiben Sie **nicht** mit Bleistift und **nicht** mit rotem oder grünem Stift (das sind die Farben für die Korrektur). Benutzen Sie blaue oder schwarze Kugelschreiber bzw. Füller.
- Lesen Sie die Aufgaben jeweils bis zum Ende durch; oft gibt es hilfreiche Hinweise!
- Als Hilfsmittel sind nur die beiden mitgebrachten beidseitig beschrifteten A4-Blätter erlaubt. Die Benutzung aller anderen Hilfsmittel ist untersagt.

Viel Erfolg!

Aufgabe 1 (8 Punkte) Allgemein.

Kreuzen Sie folgende Aussagen mit *ja* an, wenn sie zutreffen, andernfalls mit *nein*.
Beachten Sie folgende Punkteregelung:

- Für jede richtige Antwort erhalten Sie eine Punkteinheit.
- Für jede **falsche Antwort** wird Ihnen eine Punkteinheit **abgezogen**.
- Die minimal zu erzielende Punktzahl ist 0. Sie können also keine negativen Punkte erzielen, wenn die falschen Antworten die richtigen überwiegen.

Ja	**Nein**	
○	○	Ein Stack setzt die *LIFO-Strategie* um.
○	○	Es ist möglich in einem Java-Interface Methoden zu implementieren.
○	○	Es ist möglich in einem Java-Interface Methoden zu deklarieren.
○	○	Ist eine Java-Klasse mit dem Schlüsselwort `final` versehen, so dürfen von dieser Klasse keine Objekte erzeugt werden.
○	○	Die Java-Anweisung `String s;` erzeugt ein neues Objekt.
○	○	Selection-Sort ist ein stabiles Sortierverfahren.
○	○	Der Worst-Case für Heapsort ist $O(n^2)$.
○	○	In einem Binärbaum gilt für jeden Knoten, dass die Höhe des linken und des rechten Teilbaums sich höchstens um 1 unterscheiden.
○	○	In einem Suchbaum ist der Wert in einem Knoten mindestens so groß wie die Werte in allen seinen Nachfolgern.
○	○	Ein Blatt in einem Baum ist ein Knoten, der keine Nachfolger hat.
○	○	Sei G ein Graph. Dann ist G der einzige aufspannende Untergraph von G.
○	○	Jeder Teilgraph ist ein Untergraph.
○	○	Jeder kreisfreie Graph ist ein Baum.
○	○	Ein zusammenhängender gewichteter Graph kann mehrere minimale Spannbäume besitzen.
○	○	Die Höhe eines Baums ist gleich der Länge eines Wegs von der Wurzel bis zu einem beliebigen Blatt
○	○	Es sei T ein Suchbaum. Besitzen der rechte und linke Teilbaum der Wurzel die gleiche Höhe, so ist T stets ein AVL-Baum.

Aufgabe 2 (2 Punkte) Traversierung von Bäumen.

Betrachten Sie den folgenden Binärbaum aus Abbildung 1.

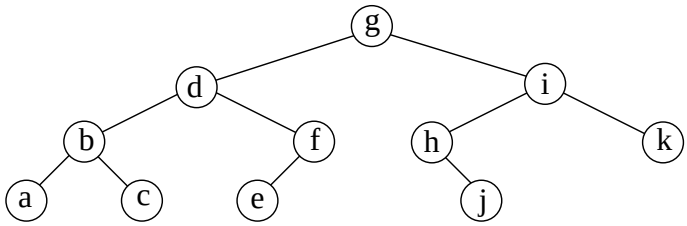


Abbildung 1: Baum T.

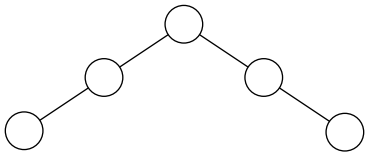
1. (1 Punkt) Geben Sie die Folge der Knoten von T in Preorder-Reihenfolge an.

2. (1 Punkt) Geben Sie die Folge der Knoten von T in Inorder-Reihenfolge an.

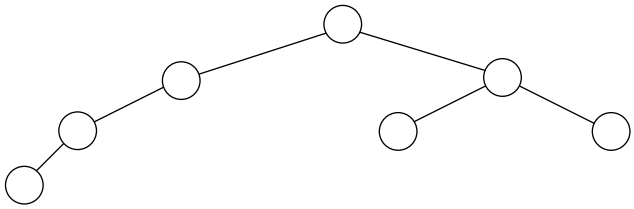
Aufgabe 3 (10 Punkte) AVL-Bäume.

Betrachten Sie die abgebildeten Bäume. Wir gehen davon aus, dass die Werte der Knoten derart sind, dass die Suchbaumeigenschaft erfüllt ist. Prüfen Sie, ob die abgebildeten Bäume AVL-Bäume sind und kreuzen Sie die entsprechende Antwort an.

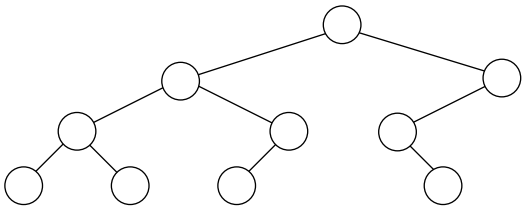
1. (2 Punkte)



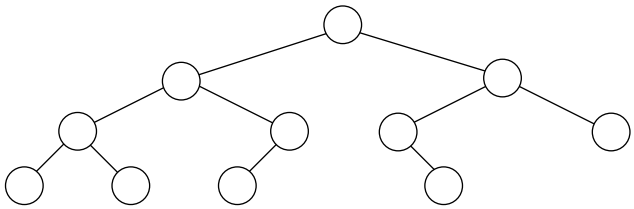
☐ AVL Baum
☐ kein AVL Baum



☐ AVL Baum
☐ kein AVL Baum



☐ AVL Baum
☐ kein AVL Baum



☐ AVL Baum
☐ kein AVL Baum

2. (8 Punkte) Fügen Sie in einen anfangs leeren AVL-Baum die Werte der Folge

$$F = 10, 15, 11, 4, 8, 7, 13.$$

nacheinander ein. Geben Sie dabei nach jedem Einfügen eines Elementes an, ob eine Rotation erforderlich ist. Wenn dies der Fall ist, kennzeichnen Sie den Bezugsknoten und geben an, ob eine Einfach- (E) oder Doppelrotation (D) durchgeführt wird. Wählen Sie als Bezugsknoten stets den Knoten, der am weitesten von der Wurzel entfernt ist.

Aufgabe 4 (5 Punkte) Grundlagen von Graphen.

Betrachten Sie den ungerichteten Graphen G aus Abbildung 2.

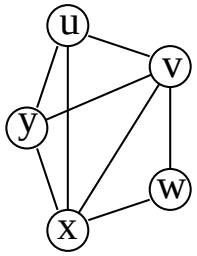


Abbildung 2: Graph G.

1. (1) Geben Sie die mathematische Darstellung von G als Menge von Knoten und Kanten an.

2. (1 Punkt) Geben Sie die Adjazenzmatrix von G an.

3. (1 Punkt) Geben Sie die Adjazenzliste von G an.

4. (2 Punkte) Zeichnen Sie einen nicht-aufspannenden Untergraphen U von G mit der folgenden Eigenschaft: Kein nicht-aufspannender Teilgraph von G hat mehr Kanten als U.

Aufgabe 5 (8 Punkte) Algorithmen auf Graphen.

1. (3 Punkte)

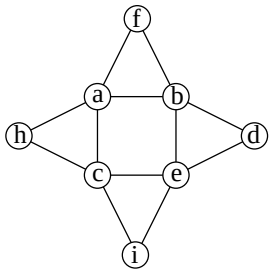


Abbildung 3: Graph G

Traversieren Sie den Graphen G aus Abbildung 3 mittels Tiefensuche. Führen Sie dazu eine Handsimulation mit Hilfe der untenstehenden Tabelle durch. Dabei bezeichne

- t die Anzahl des aktuellen Schleifendurchlaufs.
- EK den zu expandierenden Knoten.

Befolgen Sie bei der Simulation folgende Regeln:

- Startknoten ist der Knoten mit Bezeichner *a*.
- Geben Sie für $t = 0$ den Zustand des Stacks unmittelbar *nach* Initialisierung an.
- Geben Sie für $t > 0$ den Zustand des Stacks jeweils am Ende des aktuellen Schleifendurchlaufs an.
- Fügen Sie die weißen Nachfolger von EK stets in alphabetisch aufsteigender Reihenfolge in den Stack ein.

t	EK	Stack

2. (2 Punkte) Betrachten Sie den Pseudocode der Breitensuche:

```
1. Initialisiere Queue Q mit einem Knoten aus V
2. Solange Q nichtleer, wiederhole
   a. Waehle erstes Element v von Q
   b. Entferne v aus Q
   c. Fuege alle weissen Nachfolger von v an das Ende von Q ein
```

Gegeben sei ein ungerichteter Graph $G = (V, E)$, ein Startknoten $a \in V$ und ein Zielknoten $z \in V$. Es soll nun überprüft werden, ob z von a aus erreichbar ist. Modifizieren Sie den oben angegebenen Algorithmus der Breitensuche so, dass er als Rückgabe eine Aussage über die Erreichbarkeit von a und z liefert. Gesucht ist eine Variante, die für beliebig Start- und Zielknoten nicht notwendig alle Knoten von G traversiert.

3. (3 Punkte) Bestimmen Sie für den ungerichteten Graphen H aus Abbildung 4 einen minimalen Spannbaum.

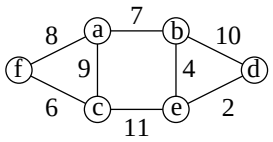


Abbildung 4: Graph H

Führen Sie mit Hilfe der untenstehenden Tabelle eine Handsimulation des Algorithmus von Kruskal durch. Dabei bezeichne

- t die Anzahl des aktuellen Schleifendurchlaufs.
- AK die aktuelle Kante.
- E' die Teillösung im t-ten Schritt.

Beachten Sie bei der Simulation die folgenden Regeln:

- Betrachten Sie Kanten zwischen zwei Knoten nur einmalig. Schreiben Sie dazu kurz $\{i, j\}$ für eine ungerichtete Kante (i, j), (j, i) zwischen Knoten i und j.
- Geben Sie für $t = 0$ den Zustand von E' unmittelbar nach Initialisierung an.
- Geben Sie für $t > 0$ den Zustand von E' jeweils am Ende eines Schleifendurchlaufs an.

t	AK	E'
0	—	

Aufgabe 6 (6 Punkte) Komplexität.

Wir legen die folgenden Definitionen für die Komplexität zugrunde: Es seien $f, g : \mathbb{N} \to \mathbb{R}^+$.

$$\begin{aligned} O(g(n)) &= \{f(n) \mid \exists c > 0, n_0 \in \mathbb{N} : f(n) \leq c \cdot g(n)\ \forall n \geq n_0\} \\ \Omega(g(n)) &= \{f(n) \mid \exists c > 0, n_0 \in \mathbb{N} : c \cdot g(n) \leq f(n)\ \forall n \geq n_0\} \end{aligned}$$

Beweisen oder widerlegen Sie die folgenden Aussagen.

1. (2.5 Punkte) Es gilt: $a_0 + a_1 n + a_2 n^2 \in O(n^2)$ für alle $a_0, a_1, a_2 \in \mathbb{R}$.

2. (3.5 Punkte) Es gilt: $n^2 \in \Omega(2^n)$

Aufgabe 7 (14 Punkte) Sortierverfahren.

1. (1.5 Punkte) Abgebildet ist die Sortierung eines Arrays mit einem einfachen Sortierverfahren. Geben Sie für den unten abgebildeten Ablauf an, mit welchem Sortierverfahren die Sortierung erfolgt ist. Es ist jeder Aufruf der Methode `swap(int[] a, int i, int j)` dargestellt.

4	1	7	6	3	11

4	1	7	3	6	11

4	1	3	7	6	11

1	4	3	7	6	11

1	4	3	6	7	11

1	3	4	6	7	11

Lösung:

2. (1.5 Punkte) Welches Sortierverfahren wird durch die abgebildete Methode `void sort( int[]f )` implementiert?

```
void sort(int[] f){
  int a; int b;
  for(int i = 0; i < f.length; i++){
    a = i;
    for(int j = i; j < f.length; j++){
      if(f[j] < f[a]){ a = j; }
    }
    swap(f, i, a);
  }
}
```

Lösung:

3. (2 Punkte) Geben Sie an, wann der Best-Case und wann der Worst-Case der folgenden Sortierverfahren auftritt:

Sortierverfahren	Best-Case	Worst-Case
Insertion-Sort		
Quicksort (Pivotelement ist das letze Element das Arrays)		

4. (9 Punkte) Stellen Sie den Verlauf der Sortierung für die Zahlenfolge

$$F = (2, 5, 6, 3, 7, 9, 1)$$

mittels Heapsort dar.

Beachten Sie dabei folgende Regeln.

(a) Die Simulation des Verlaufs soll sowohl im Baum/Heap als auch im Array dargestellt werden.

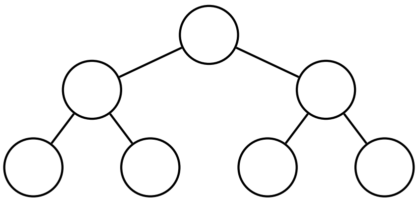
(b) Kennzeichnen Sie **alle** Elementvertauschungen. Markieren Sie zu tauschende Elemente mit einem doppelseitigen Pfeil.

(c) Kennzeichnen Sie, wann die erste Phase (initialer Heapaufbau) abgeschlossen ist.

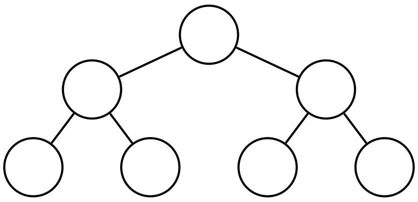
(d) Während der zweiten Phase unterstreichen Sie den bereits sortierten Teil des Arrays immer dann, wenn dieser Bereich sich um ein Element vergrößert.

(e) Elemente dieses sortierten Bereiches können im Heap weggelassen werden.

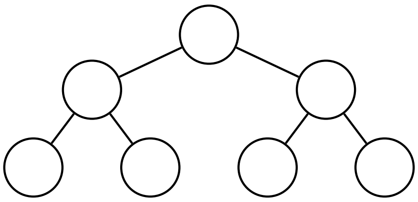
(f) Verwenden Sie für diese Aufgabe die Vorlage auf den folgenden Seiten.



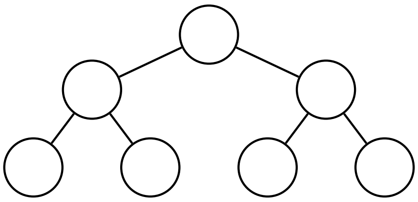
0	1	2	3	4	5	6
2	5	6	3	7	9	1



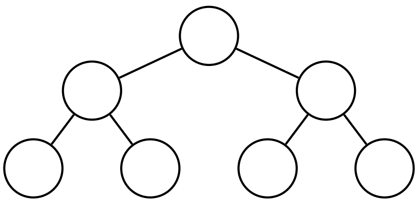
0	1	2	3	4	5	6



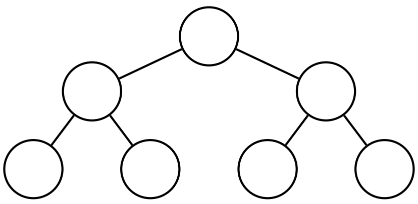
0	1	2	3	4	5	6



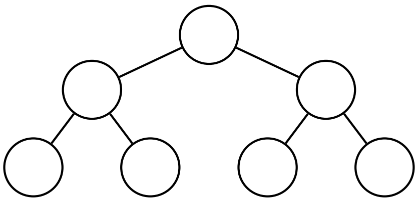
0	1	2	3	4	5	6



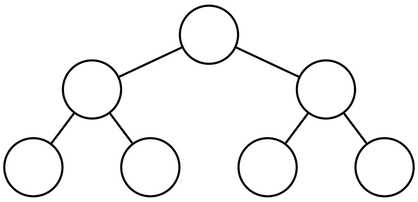
0	1	2	3	4	5	6



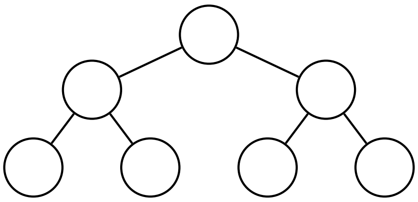
0	1	2	3	4	5	6



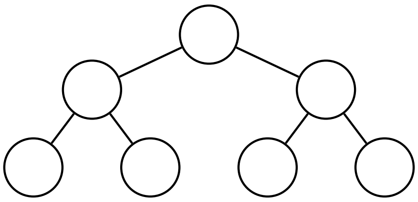
0	1	2	3	4	5	6



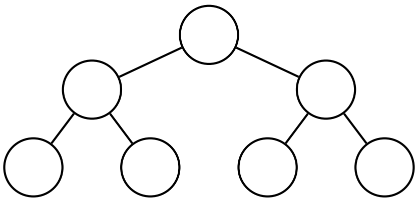
0	1	2	3	4	5	6



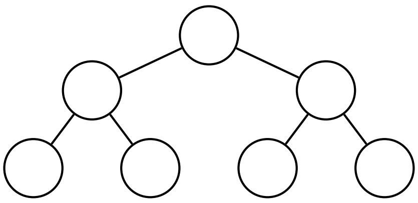
0	1	2	3	4	5	6



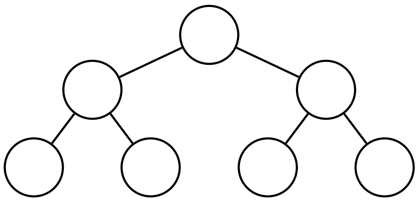
0	1	2	3	4	5	6



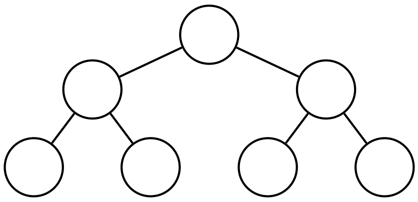
0	1	2	3	4	5	6



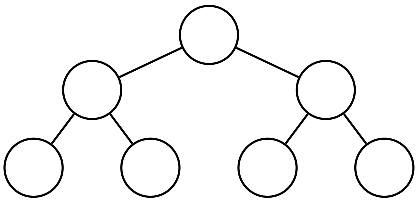
0	1	2	3	4	5	6



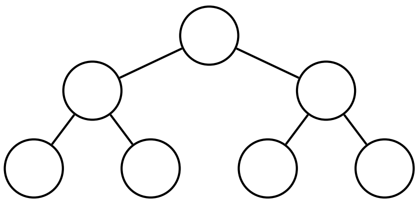
0	1	2	3	4	5	6



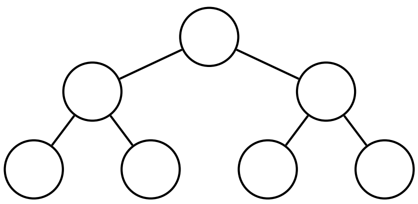
0	1	2	3	4	5	6



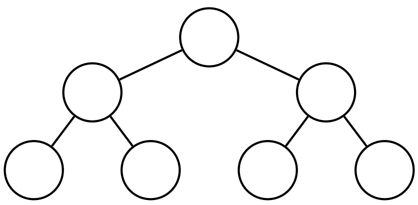
0	1	2	3	4	5	6



0	1	2	3	4	5	6



0	1	2	3	4	5	6



0	1	2	3	4	5	6

Aufgabe 8 (8 Punkte) Java.

1. (4 Punkte) Betrachten Sie die unten abgebildeten Java-Klassen. Was wird durch die `main` Methode der Klasse `FlaschenTest` ausgegeben? Achten Sie auf die korrekte Reihenfolge der Ausgaben!

```
class Flasche{
  int volumen; //Angabe in Litern
  Flasche(int volumen){ this.volumen = volumen; }
  void print(){
    System.out.println("Eine Flasche mit " + volumen + " Litern.");
  }
}

class ColaFlasche extends Flasche{
  ColaFlasche(int volumen){ super(volumen); }
  void print(){
    System.out.println("Eine Colaflasche mit " + volumen + " Litern.");
  }
  void print(String s){}
}

class MilchFlasche extends Flasche{
  MilchFlasche(int volumen){ super(volumen); }
  void print(String s){
    System.out.println("Eine " + s +  " mit " + volumen + " Litern.");
  }
}

class FlaschenTest{
  public static void main(String args[]){
    MilchFlasche f1 = new MilchFlasche(2);
    Flasche f2 = new Flasche(3);
    Flasche f3 = new ColaFlasche(4);
    Flasche f[] = {f1, f2, f3};
    for(int i = 0; i < f.length; i++){
      f[i].print();
      if( f[i] instanceof ColaFlasche )
          ((ColaFlasche)f[i]).print("kleine Colaflasche");
      if( f[i] instanceof MilchFlasche )
          ((MilchFlasche)f[i]).print("grosse Milchflasche");
    }
  }
}
```

Lösung:

2. (4 Punkte) Betrachten Sie die folgenden Java-Klassen und das gegebene Interface. Ihre Aufgabe ist es, die Klassen `GeometrischesObjekt`, `Kreis` und `Quadrat` so zu vervollständigen, dass die `main`-Methode der Klasse `GeomObjektTest` ausgeführt werden kann. Für die im Array `geomObj` vorhandenen Objekte soll in der `for`-Schleife jeweils der Flächeninhalt ausgegeben werden.

   Ein Kreis soll durch seinen Mittelpunkt und den Radius beschrieben werden. Ein Quadrat durch seinen linken unteren Eckpunkt und die Seitenlänge. Vervollständigen Sie die Konstruktoren der Klassen `Kreis` und `Quadrat`.

   Da der Flächeninhalt eines geometrischen Objektes nicht berechnet werden soll, wenn nur der Verankerungspunkt bekannt ist, soll Ihre Implementierung sicherstellen, dass keine Objekte der Klasse `GeometrischesObjekt` erzeugt werden können. Es soll aber sichergestellt sein, dass alle von dieser Klasse abgeleiteten Klassen über eine Methode `berechneFlaeche` verfügen.

   Hinweis: Die Flächenberechung für einen Kreis erfolgt mit der Formel $A = r^2 * \pi$.

```
class GeomObjektTest{
  public static void main(String args[]){
    Point pk = new Point(3.0,  4.0);
    Point pr = new Point(2.0, 6.0);
    HatFlaeche geomObj[] = { new Kreis(pk, 3), new Quadrat(pr, 4) };
    for(int i = 0; i < geomObj.length; i++){
      System.out.println(geomObj[i].berechneFlaeche());
    }
  }
}

class Point{
  double x; double y;
  Point(double x, double y){ this.x = x; this.y = y; }
}

interface HatFlaeche{
  double berechneFlaeche();
}


class GeometrischesObjekt implements HatFlaeche{
  Point verankerung;
  GeometrischesObjekt(Point verankerung){
    this.verankerung = verankerung;
  }









}
```

```
class Kreis extends GeometrischesObjekt{
  double radius;
  Kreis(Point verankerung, double radius){









}

class Quadrat extends GeometrischesObjekt {
  double seitenlaenge;
  Quadrat(Point verankerung, double seitenlaenge){









}
```

Aufgabe 9 (4 Punkte) Referenzen.

Analysieren Sie die abgebildeten Java-Klassen `Value` und `ValueTest`. Geben Sie an, was durch die `main`-Methode der Klasse `ValueTest` an den Punkten 1 – 8 ausgegeben wird und tragen die Ausgaben in die Tabelle ein.

```
1  class Value{
2    String string;
3    int integer;
4    Value(){
5      this.string = "AAA"; this.integer = 0;
6    }
7    Value(String string, int integer){
8      this.string = string; this.integer = integer;
9    }
10   Value m(Value v, int w){
11     w = 8;
12     v.string += w;
13     v.integer += w;
14     return this;
15   }
16   void print(){
17     System.out.println(string + " " + integer);
18   }
19 }
20
21 class ValueTest{
22   public static void main(String args[]){
23     Value object1 = new Value();
24     Value object2 = new Value("BBB", 1);
25     Value object3 = new Value("ZZZ", 25);
26     int a = 6;
27     object1.print();    //1.
28     object2.print();    //2.
29     object3.print();    //3.
30     System.out.println( a );//4.
31     object3 = object1.m(object2, a);
32     object1.print();    //5.
33     object2.print();    //6.
34     object3.print();    //7.
35     System.out.println( a );//8.
36   }
37 }
```

1.	
2.	
3.	
4.	
5.	
6.	
7.	
8.	

Aufgabe 10 (5 Punkte) Numerik.

Es seien

$$\mathcal{S} = \{(x_0, f_0), (x_1, f_1), \ldots, (x_n, f_n)\}$$

$n-1$ verschiedene Stützpunkte. Die Lagrange'sche Interpolationsformel zur Lösung des Interpolationsproblem von $\mathcal{S}$ ist von der Form

$$\ell(y) = \sum_{i=0}^{n} f_i \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{y - x_k}{x_i - x_k}$$

Implementieren Sie die Lagrange'sche Interpolationsformel in Java. Dabei sei `double[] x` ein Array von Stützstellen und `double[] f` ein Array mit zugehörigen Stützwerten. Der Parameter y spezifiert die Stelle an der die unbekannte Funktion f interpoliert werden soll. Sie können davon ausgehen, dass die Arrays `double[] x` und `double[] f` jeweils mindestens ein Element besitzen und gleich lang sind. Des weiteren sind die Elemente des Arrays `double[] x` paarweise verschieden.

```
double l(double[] x, double[] f, double y) {




}
```