Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 8 + 6 Punkte)

a) Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Schreiben Sie hierzu jeweils die ausgegebenen Zeichen hinter den Kommentar "`OUT:`".

```
public class A {
    public static int x = 1;
    public A(int x) {
        A.x += x;
    }
    public int f(int x) {
        x++;
        return x;
    }
}

public class B extends A {
    public int y = 2;
    public B(int y) {
        super(y+2);
    }
    public int f(int x) {
        y += x;
        return y;
    }
}

public class M {
    public static void main(String[] args) {
        A a = new A(2);
        System.out.println(A.x);          // OUT: 3
        int z = a.f(3);
        System.out.println(z+" "+A.x);    // OUT: 4 3
        B b = new B(5);
        System.out.println(B.x+" "+b.y); // OUT: 10 2
        z = b.f(6);
        System.out.println(z+" "+b.y);    // OUT: 8 8
        a = b;
        z = a.f(7);
        System.out.println(z);            // OUT: 15
    }
}
```

Vorname	Name	Matr.-Nr.

3

b) Wir schreiben zusätzlich zu `A` und `B` eine neue Klasse `C`. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
public class C extends B {
    final static int z = 9;
    public C() {
        super(z,z);
        z = 10;
    }
    public double f(int z) {
        return x;
    }
}
```

- *Die Anweisung* `super(z,z);` *im Konstruktor* `C()` *ist nicht erlaubt, da es in der direkten Oberklasse* `B` *keinen zweistelligen Konstruktor mit zwei Argumenten vom Typ* `int` *gibt.*
- *Die Anweisung* `z = 10;` *im Konstruktor* `C()` *ist nicht erlaubt, da* `z` *als* `final` *deklariert wurde und sein Wert somit nicht mehr verändert werden darf.*
- *Das Überschreiben der Methode* `f` *aus der Oberklasse* `B` *durch die Methode* `public double f(int z)` *in der Klasse* `C` *ist nicht erlaubt, da der Rückgabetyp von* `f` *in* `B` *der Typ* `int` *und nicht* `double` *ist.*

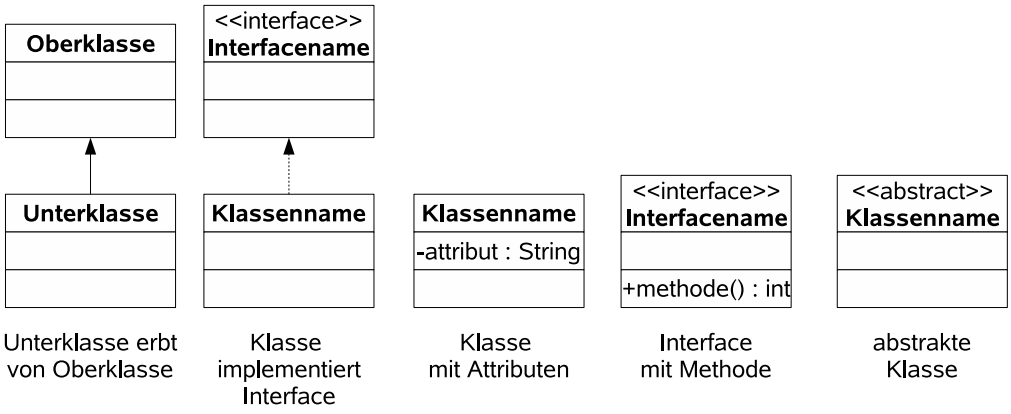
Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 6 + 8 Punkte**)**

Ihre Aufgabe ist es, eine objektorientierte Datenstruktur zur Verwaltung von Pflanzen zu entwerfen. Bei der vorangehenden Analyse wurden folgende Eigenschaften der verschiedenen Pflanzen ermittelt.

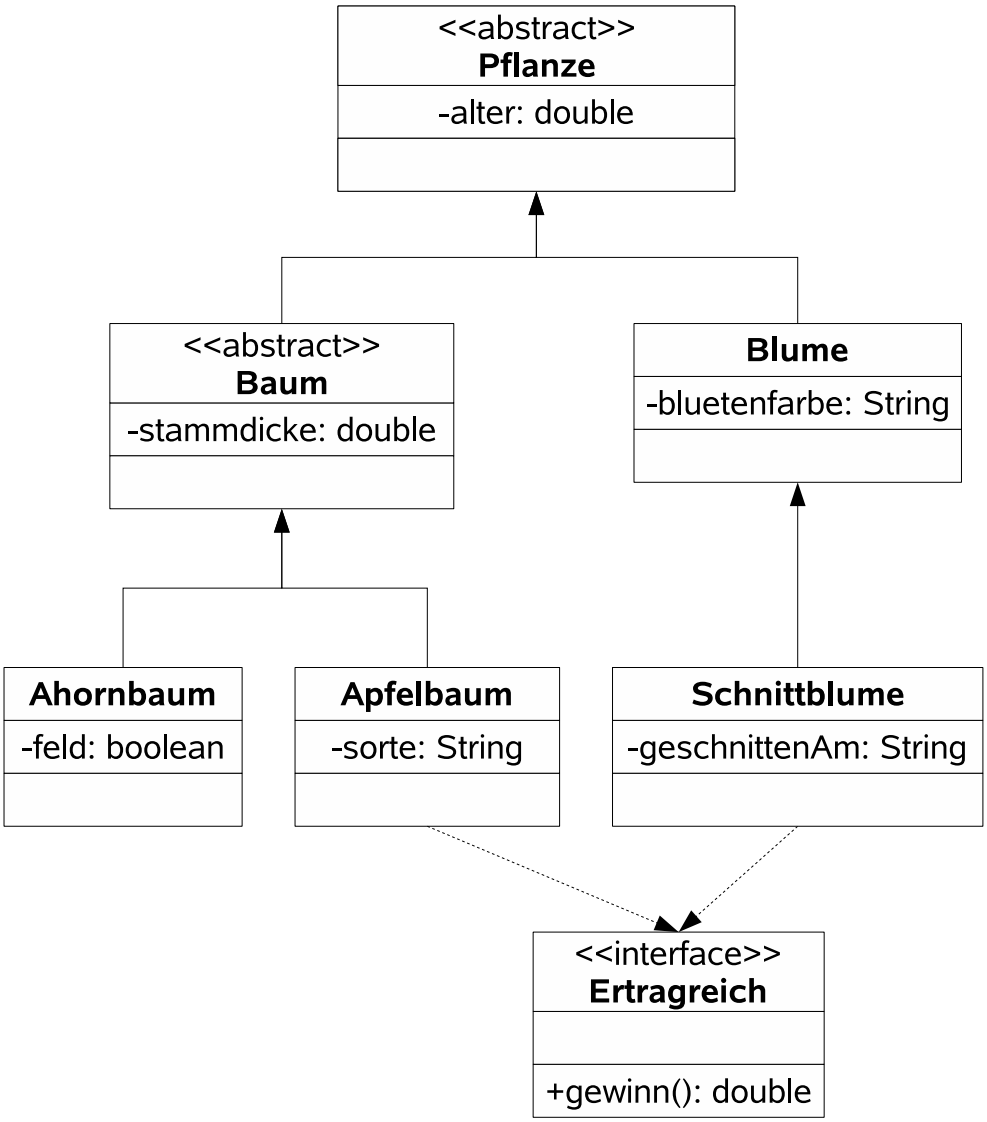
- Ein Apfelbaum ist eine Pflanze. Jeder Apfelbaum hat ein bestimmtes Alter und eine Stammdicke. Zudem gibt es zu jedem Apfelbaum eine Sorte von Äpfeln, die an ihm wachsen.
- Ahornbäume kennzeichnen sich durch ihre Stammdicke und ihr Alter. Man unterscheidet zwischen Feld- und Spitzahornbäumen. Natürlich ist ein Ahornbaum auch eine Pflanze.
- Eine Blume ist eine Pflanze und hat ein bestimmtes Alter. Bei Blumen ist vor allem die Blütenfarbe interessant.
- Schnittblumen sind Pflanzen mit einer bestimmten Blütenfarbe. Neben dem Alter der Schnittblume ist auch die Dauer seit dem Abschneiden interessant.
- Apfelbäume und Schnittblumen sind ertragreiche Pflanzen. Für diese kann man berechnen, wieviel Gewinn man erzielt (durch den Verkauf der jeweiligen Äpfel eines Apfelbaums oder durch den Verkauf der Schnittblume).

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Pflanzen. Achten Sie darauf, dass gemeinsame Merkmale in (evtl. abstrakten) Oberklassen zusammengefasst werden. Notieren Sie Ihren Entwurf graphisch und verwenden Sie dazu die folgende Notation:



Geben Sie für jede Klasse ausschließlich den jeweiligen Namen und die Namen ihrer Attribute an. Methoden von Klassen müssen nicht angegeben werden. Geben Sie für jedes Interface ausschließlich den jeweiligen Namen sowie die Namen seiner Methoden an.

Vorname	Name	Matr.-Nr.



Vorname	Name	Matr.-Nr.

b) Implementieren Sie in Java eine Methode `schlechterBaum`. Die Methode bekommt als Parameter ein Array von ertragreichen Pflanzen übergeben. Sie soll den ersten Apfelbaum liefern, der einen geringeren Gewinn hat als der durchschnittliche Gewinn der ertragreichen Pflanzen im Array. Ansonsten soll die Methode `null` zurückliefern.

Gehen Sie dabei davon aus, dass das übergebene Array nicht der `null`-Wert ist und dass es keine `null`-Werte enthält. Kennzeichnen Sie die Methode mit dem Schlüsselwort "`static`", falls angebracht.

```
public static Apfelbaum schlechterBaum(Ertragreich[] array) {
  double durchschnitt = 0.0;
  int anzahl = array.length;

  if (anzahl == 0) return null;

  for (int i=0; i<anzahl; i++) {
    durchschnitt += array[i].gewinn();
  }
  durchschnitt /= anzahl;

  for (int i=0; i<anzahl; i++) {
    if (array[i] instanceof Apfelbaum &&
        array[i].gewinn() < durchschnitt) {
      return (Apfelbaum) array[i];
    }
  }

  return null;
}
```

Vorname	Name	Matr.-Nr.

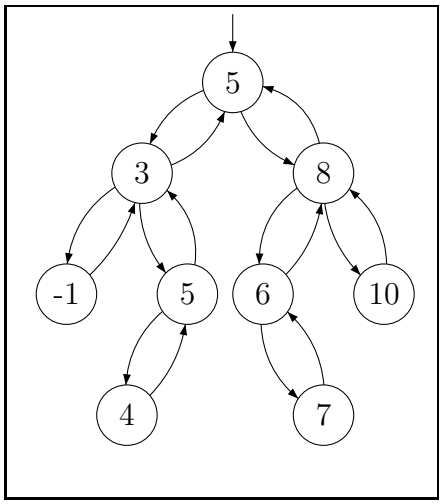
Aufgabe 4 (Programmierung in Java, 7 + 7 + 12 Punkte)

Gegeben ist das folgende Interface.

```
public interface Vergleichbar {
    public boolean groesser(Vergleichbar other);
}
```

Für zwei Objekte `s` und `t` vom Typ `Vergleichbar` liefert `s.groesser(t)` den Wert `true` zurück, falls `s` größer als `t` ist und ansonsten liefert `s.groesser(t)` den Wert `false`.

In dieser Aufgabe betrachten wir einen doppelt verketteten sortierten Binärbaum. Das folgende Bild zeigt einen solchen Baum schematisch:



Ein solcher Baum heißt *sortiert*, wenn die folgenden Bedingungen erfüllt sind:

- jeder Knoten enthält einen Wert `wert` vom Typ `Vergleichbar`
- falls ein Knoten einen linken Unterbaum hat, so sind alle Werte im linken Unterbaum kleiner oder gleich `wert`
- falls ein Knoten einen rechten Unterbaum hat, so sind alle Werte im rechten Unterbaum größer als `wert`

Die Datenstruktur heißt doppelt verkettet, da jeder Knoten sowohl Verweise auf seine Nachfolger (`links` und `rechts`) als auch auf seinen Vorgänger (`vater`) besitzt.

Die Datenstruktur sei folgendermaßen in Java implementiert:

```
public class Baum {
    private Knoten wurzel;

    ...
}
```

Vorname	Name	Matr.-Nr.

10

```
public class Knoten {
    Vergleichbar wert;
    Knoten links, rechts, vater;

    public Knoten (Vergleichbar wert) {
        this.wert = wert;
    }
}
```

Implementieren Sie die folgenden Methoden in der Klasse `Baum`. Dabei dürfen Sie direkt (ohne Verwendung von Selektoren) auf die Attribute aus der Klasse `Knoten` zugreifen. Selbstverständlich dürfen Sie auch weitere benötigte Hilfsmethoden implementieren. Verwenden Sie die Schlüsselworte `public` und `private` auf sinnvolle Weise und kennzeichnen Sie Methoden als `static`, falls angebracht.

a) Implementieren Sie eine Methode `public int hoehe()`, die die Höhe des Binärbaums berechnet. Die *Höhe* eines Binärbaums sei dabei wie folgt definiert:

- Der leere Binärbaum besitzt die Höhe 0.
- Der Binärbaum, der nur einen Knoten besitzt (an der Wurzel), hat die Höhe 1.
- Jeder andere Binärbaum besitzt eine Höhe von $1 + \max$(Höhe des linken Unterbaums, Höhe des rechten Unterbaums).

Der Binärbaum im Beispiel hat also die Höhe 4.

Beachten Sie, dass bei der Implementierung dieser Methode keine Schleifen verwendet werden dürfen. Sie dürfen aber Rekursion benutzen.

b) Implementieren Sie eine Methode `public boolean balanciert()`, die berechnet, ob der Binärbaum balanciert ist. Ein Binärbaum heißt *balanciert*, wenn für jeden seiner Knoten gilt, dass sich die Höhen der Unterbäume dieses Knotens maximal um 1 unterscheiden. Sie können hier die Funktion `public static int abs (int zahl)` der Klasse `Math` verwenden, die den absoluten Betrag einer `int`-Zahl `zahl` zurückliefert.

c) Implementieren Sie eine Methode `public void einfuegen(Vergleichbar w)`, die einen gegebenen Wert `w` in den Binärbaum einfügt. Gehen Sie hierbei davon aus, dass `w` nicht `null` ist und dass auch in jedem Knoten des Baums der jeweilige `wert` nicht `null` ist. Der Binärbaum muss nach diesem Einfügen wieder *sortiert* sein. Achten Sie darauf, dass auch die `vater`-Verweise auf die jeweiligen Vorgänger entsprechend aktualisiert werden.

Vorname	Name	Matr.-Nr.


```
private static int hoehe(Knoten current){
    if    (current == null)
          return 0; // der leere Baum hat die Hoehe 0
    else {int hoeheLinks = hoehe(current.links);
          int hoeheRechts = hoehe(current.rechts);
          return 1 + (hoeheLinks >= hoeheRechts ? hoeheLinks : hoeheRechts);
    }
}

public int hoehe(){
    return hoehe(wurzel);
}

private static boolean balanciert(Knoten current){
    if    (current == null)
          return true; //der leere Baum ist per Definition balanciert
    else  return balanciert(current.links) &&
                 balanciert(current.rechts) &&
                 Math.abs(hoehe(current.links) - hoehe(current.rechts)) <= 1;
}


public boolean balanciert(){
    return balanciert(wurzel);
}

private static void einfuegen(Knoten current, Vergleichbar w){
    boolean groesser = w.groesser(current.wert);
    Knoten kind = groesser ? current.rechts : current.links;
                 //Der Wert w muss in den Teilbaum eingefuegt werden,
                 //der mit kind beginnt.
    if   (kind != null)
         einfuegen(kind,w);
    else {kind = new Knoten(w);
          kind.vater = current;
          if  (groesser)
               current.rechts = kind;
          else current.links = kind;
    }
}

public void einfuegen(Vergleichbar w){
    if   (wurzel == null)
         wurzel = new Knoten(w);
    else einfuegen(wurzel, w);
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 1 (Programmanalyse, 8 + 6 Punkte)

a) Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Schreiben Sie hierzu jeweils die ausgegebenen Zeichen hinter den Kommentar "`OUT:`".

```
public class A {
  public int z = 1;

  public void f(int x) {
    z = x + z;
  }
}
public class B extends A {
  public static int n = 0;
  public B() {
      n++;
  }
  public void f(int x) {
    n = x + n;
  }
}
public class M {
  public static void main(String[] args) {
    A a = new A();
    a.f(5);
    System.out.println(a.z+", " + B.n); // OUT: 6, 0
    B b = new B();
    b.f(5);
    System.out.println(b.z+", " + B.n); // OUT: 1, 6
    a = b;
    b.z = a.z + 1;
    a.f(5);
    System.out.println(a.z+", " + B.n); // OUT: 2, 11
    a.z = b.z + 1;
    b.f(5);
    System.out.println(b.z+", " + B.n); // OUT: 3, 16
  }
}
```

b) In der Klasse `B` wird die Methode `f` durch folgende Methode ersetzt. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
  protected void f(int a) {
    A a = new B();
    B b = new B();
    b.z = new A().z;
    A.n = 1;
  }
```

Vorname	Name	Matr.-Nr.

3

1. `protected void f(int a)` *ist falsch, da die überschriebene Methode* `void f(int a)` *der Klasse* `A` *als* `public` *definiert ist.*

2. `A a = ...` *ist falsch, da* `a` *bereits durch den Methodenkopf* `void f(int a)` *definiert ist.*

3. `A.n` *ist falsch, da die Klasse* `A` *nicht über ein statisches Attribut* `n` *verfügt.*

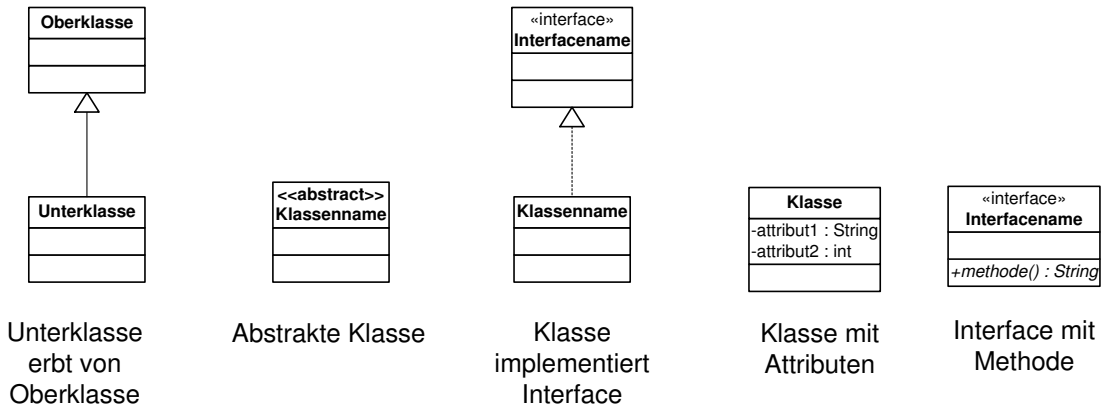
Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 6 + 8 Punkte**)**

Ihre Aufgabe ist es, eine objektorientierte Datenstruktur zur Verwaltung von Datenspeichern zu entwerfen. Bei der vorangehenden Analyse wurden folgende Eigenschaften der verschiedenen Datenspeicher ermittelt:

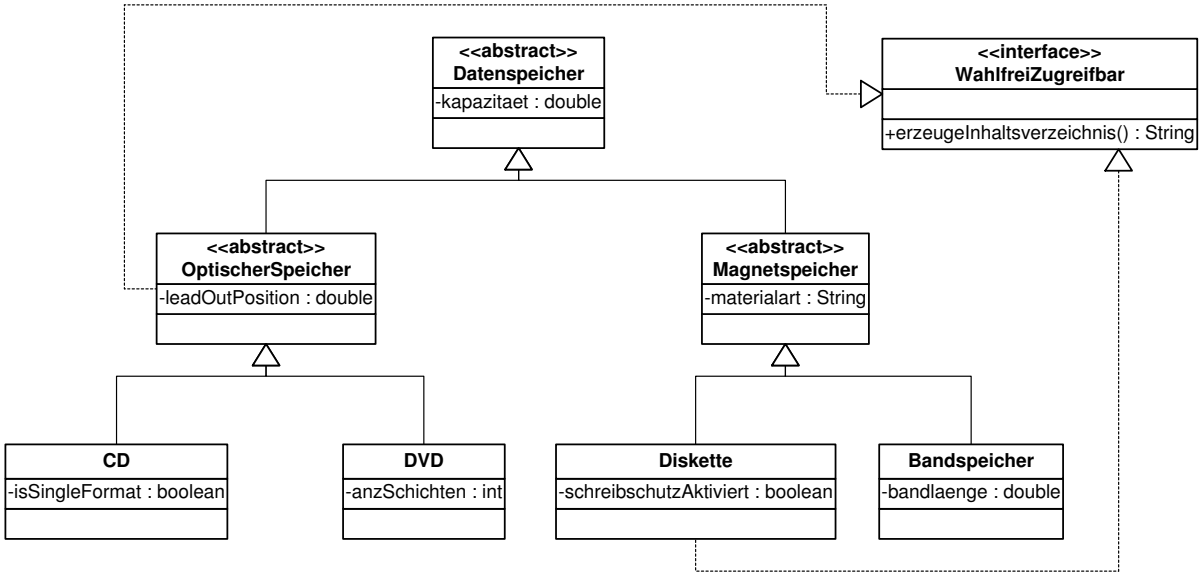
- Eine CD ist ein optischer Speicher, der über eine bestimmte Kapazität verfügt und eine Lead-Out-Position besitzt (d.h., eine Zeitangabe als Gleitkommazahl), die das Ende der Aufzeichnung kennzeichnet. Darüberhinaus kann eine CD im Single-Format vorliegen oder nicht.
- Eine DVD ist ein optischer Speicher mit einer bestimmten Kapazität und einer Lead-Out-Position. Darüberhinaus besitzt eine DVD eine bestimmte Anzahl von Schichten.
- Eine Diskette ist ein magnetischer Speicher mit einer bestimmten Kapazität, der durch eine Materialart (dargestellt durch einen `String`) gekennzeichnet ist. Darüberhinaus kann bei einer Diskette ein Schreibschutz aktiviert sein oder nicht.
- Ein Bandspeicher ist ein magnetischer Speicher mit einer bestimmten Kapazität, der durch eine Materialart und die Bandlänge gekennzeichnet ist.
- Alle Datenspeicher außer dem Bandspeicher sind Datenspeicher mit wahlfreiem Zugriff. Für Datenspeicher mit wahlfreiem Zugriff soll es möglich sein, ein Inhaltsverzeichnis (als `String`) berechnen zu lassen.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Datenspeichern. Achten Sie darauf, dass gemeinsame Merkmale in (evtl. abstrakten) Oberklassen zusammengefasst werden. Notieren Sie Ihren Entwurf graphisch und verwenden Sie dazu die folgende Notation:



Geben Sie für jede Klasse ausschließlich den jeweiligen Namen und die Namen ihrer Attribute an. Methoden von Klassen müssen nicht angegeben werden. Geben Sie für jedes Interface ausschließlich den jeweiligen Namen sowie die Namen seiner Methoden an.

Vorname	Name	Matr.-Nr.



Vorname	Name	Matr.-Nr.

b) Implementieren Sie in Java eine Methode `getKapazitaeten`, die ein Array von Datenspeichern übergeben bekommt. Die Methode soll ein zwei-elementiges Array zurückliefern, welches die Gesamtkapazität aller optischen Speicher und aller magnetischen Speicher enthält. Die einzelnen Positionen des Arrays sind dabei wie folgt festgelegt:

- Position 0: Gesamtkapazität aller optischen Speicher
- Position 1: Gesamtkapazität aller magnetischen Speicher

Gehen Sie dabei davon aus, dass für alle Attribute geeignete Selektoren existieren und verwenden Sie für den Zugriff auf die benötigten Attribute die passenden Selektoren. Kennzeichnen Sie die Methode mit dem Schlüsselwort "`static`", falls angebracht.

```
public static double[] getKapazitaeten(Datenspeicher[] dt){
    double[] kaps = {0,0};
    int pos;
    if(dt != null){
        for(int i = 0; i < dt.length; i++){

            if   (dt[i] instanceof OptischerSpeicher) pos = 0;
            else pos = 1;

            kaps[pos] = kaps[pos] + dt[i].getKapazitaet();

        }
    }
    return kaps;
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 5 + 4 + 2 + 7 + 8 Punkte**)**

In dieser Aufgabe sollen Programme unter Beachtung der Prinzipien der Datenkapselung entworfen werden. Kennzeichnen Sie Methoden mit dem Schlüsselwort "`static`", falls angebracht.

a) Gegeben ist das folgende Interface.

```
public interface Test {
    public boolean check(Object obj);
}
```

Falls `t` ein Objekt vom Typ `Test` ist und `obj` ein beliebiges `Object` ist, so sagen wir, dass "`obj` den Test `t` erfüllt", falls `t.check(obj)` den Wert `true` liefert. Schreiben Sie zwei Klassen `SmallerTest` und `InverseTest` in Java, die beide das Interface `Test` implementieren. Objekte der Klasse `SmallerTest` kapseln eine ganze Zahl. Falls `s` ein Objekt der Klasse `SmallerTest` ist, so soll `obj` genau dann den Test `s` erfüllen, wenn `obj` eine ganze Zahl vom Typ `Integer` ist, deren Wert kleiner als die in `s` gekapselte Zahl ist. Hierbei können Sie die Methode `public int intValue()` der Klasse `Integer` verwenden.

Objekte der Klasse `InverseTest` kapseln ein Objekt vom Typ `Test`. Falls `i` ein Objekt der Klasse `InverseTest` ist, so sollen genau die Objekte `obj` den Test `i` erfüllen, die den in `i` gekapselten Test nicht erfüllen. Sie brauchen keine Konstruktoren für die Klassen zu schreiben.

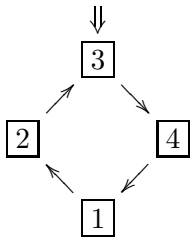
```
public class SmallerTest implements Test {

    private int bound;

    public boolean check(Object obj) {
        if   (obj instanceof Integer)
             return ((Integer) obj).intValue() < bound;
        else return false;
    }
}
```

```
public class InverseTest implements Test {

    private Test t;

    public boolean check(Object obj) {
        return (!t.check(obj));
    }
}
```

Vorname	Name	Matr.-Nr.

10

b) Die folgenden beiden Klassen dienen zur Darstellung von *Ringen* (zyklischen Listen). In diesen Ringen können beliebige Objekte (vom Typ `Object`) gespeichert werden. Ein Beispiel ist der folgende Ring `r1`:



Die einzelnen Ringelemente werden durch Objekte der Klasse `Element` implementiert, die jeweils ein Attribut `value` für den Wert und ein Attribut `next` für das nächste Element des Rings besitzen. Ein Objekt der Klasse `Ring` besitzt nur ein Attribut `position`, das auf das Element an der *aktuellen* Position zeigt (im Beispiel durch einen Pfeil "⇓" gekennzeichnet). Für den Ring `r1` wäre `r1.position` das Element `e` mit dem `value` 3. Es muss sicher gestellt sein, dass das "letzte" Ring-Element wieder auf das erste Element des Rings verweist. Wenn man also vom Element `e` aus viermal dem `next`-Verweis folgt, so erhält man wieder dasselbe Element, d.h., es gilt `e.next.next.next.next == e`. Die Klassen `Ring` und `Element` sind wie folgt implementiert. Der Konstruktor `Ring()` dient hierbei zur Erzeugung des leeren Rings.

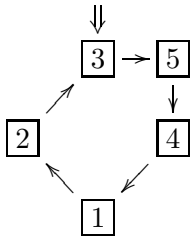
```
public class Element {

    private Object value;
    private Element next;

    public Element(Object value, Element next) {
        this.next = next;
        this.value = value;
    }

    public void setNext(Element next) {
        this.next = next;
    }

    public Element getNext() {
        return next;
    }

    public Object getValue() {
        return value;
    }
}

public class Ring {

    private Element position;

    public Ring() {
        position = null;
    }
}
```

Vorname	Name	Matr.-Nr.

Ergänzen Sie die Klasse `Ring` um eine Methode

```
public void insert(Object v) { ... }
```

die ein neues `Element` mit dem Wert `v` in den Ring einfügt und zwar an die Stelle hinter der aktuellen Position. Sofern der Ring nicht leer war, bleibt die aktuelle Position unverändert. Falls `v` der Wert `5` (als `Integer`-Objekt) ist, so verändert der Aufruf `r1.insert(v)` den Ring `r1` also zu folgendem Ring `r2`.




```
public void insert(Object v) {
    if (position == null) {
        position = new Element(v,null);
        position.setNext(position);
    }
    else {
        Element e = new Element(v,position.getNext());
        position.setNext(e);
    }
}
```

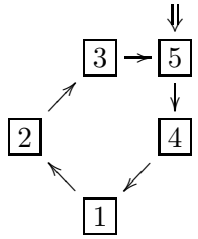
Vorname	Name	Matr.-Nr.

12

c) Ergänzen Sie die Klasse `Ring` um eine Methode

```
public void rotate() { ... }
```

die die aktuelle Position um eins weitersetzt. Die Anwendung von `rotate` auf `r2` ergibt also den folgenden Ring `r3`.



Die Anwendung von `rotate` auf einen leeren Ring oder einen Ring mit nur einem Element ändert den Ring nicht.

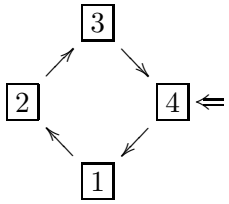
```
public void rotate() {
    if (position != null) position = position.getNext();
}
```

Vorname	Name	Matr.-Nr.

d) Ergänzen Sie die Klasse `Ring` um eine Methode

```
public void delete() { ... }
```

die das Element an der aktuellen Position löscht. Die neue aktuelle Position ist dann die darauffolgende Position. Die Anwendung von `delete` auf `r3` ergibt also den folgenden Ring `r4`.



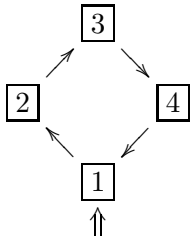

```
public void delete() {
    if (position != null) {
        if   (position.getNext() == position) position = null;
        else {
             Element prev = position;
             while (prev.getNext() != position) prev = prev.getNext();
             position = position.getNext();
             prev.setNext(position);
       }
    }
}
```

Vorname	Name	Matr.-Nr.

e) Ergänzen Sie die Klasse `Ring` um eine Methode

```
public boolean find(Test t) { ... }
```

die überprüft, ob es einen Wert im Ring gibt, der den Test `t` erfüllt. Falls kein Wert des Rings den Test erfüllt, wird nichts geändert und `false` zurückgegeben. Ansonsten wird die aktuelle Position auf das erste Element gesetzt, das den Test erfüllt und `true` zurückgegeben. (Die Suche soll hierbei an der aktuellen Position beginnen.) Falls `t` der Test der Klasse `SmallerTest` ist, der überprüft, ob ein Wert kleiner als `3` ist, so hat `r4.find(t)` das Ergebnis `true` und der Ring wird wie folgt geändert:



Die Methode `find` und ggf. benötigte Hilfsmethoden sollen ohne Verwendung von Schleifen realisiert werden. Sie dürfen aber Rekursion benutzen.

```
public boolean find(Test t) {
    if   (position == null) return false;
    else return find(position,t);
}

private boolean find(Element current, Test t) {
    if      (t.check(current.getValue())) {position = current; return true;}
    else if (current.getNext() == position) return false;
    else    return find(current.getNext(),t);
}
```