1 Allgemeine Aufgaben

Aufgabe 1.1: Begriffe

Was ist eine Klasse?

Was ist ein Objekt?

Was ist eine Oberklasse?

Was ist eine Unterklasse?

Was ist eine abstrakte Klasse?

Was ist eine abstrakte Methode?

Was ist ein Interface (in einem Java-Programm)?

Was versteht man unter Narrowing (in einem Java-Programm)?

Was versteht man unter Widening (in einem Java-Programm)?

Was versteht man unter „Späte Bindung“ (in einem Java-Programm)?

Was versteht unter „dynamische Erweiterung“ (in einem Java-Programm)?

Was versteht man unter Delegation (in einem Java-Programm)?

Was versteht man unter dem Überladen von Methoden?

Was versteht man unter dem Überschreiben von Methoden?

Was versteht man unter impliziter Typanpassung?

Was versteht man unter expliziter Typwandlung?

Unter welcher Bedigung ist eine Rekursion linear?

Unter welcher Bedingung ist eine Rekursion schlicht?

Was versteht man unter der Erweiterung einer Klasse?

Aufgabe 1.2: Rekursion

Betrachten Sie die folgende Methode.

```
int Fkt(int x)
{ System.out.println(x);
  if (x < 9)
    return x;
  if (Fkt(x/2) % 2 == 0)
    return Fkt(x - 10);
  else
    return Fkt(x - 20);
}
```

a) Ist die Rekursion der Methode `int Fkt(int x)` linear? Geben Sie bitte eine Antwort mit Begründung.

b) Ist die Rekursion der Methode `int Fkt(int x)` schlicht? Geben Sie bitte eine Antwort mit Begründung.

c) Welche Ausgabe liefert der Aufruf
`System.out.println(34 + Fkt(34));`

2 Listen und Arrays

Aufgabe 2.1: Zugfahrt

Ein Fernzug fährt täglich die gleiche Strecke, die 15 Städte verbindet. Gemäß Fahrplan benötigt er für jede der 14 Teilstrecken zwischen zwei Städten genau 19 Minuten. Die Aufenthaltsdauer in jedem der 13 Bahnhöfe unterwegs beträgt laut Fahrplan genau 2 Minuten. In allen Bahnhöfen (außer im Startbahnhof zu Beginn der Reise) kann jeweils ein Regionalzug erreicht werden, der gemäß Fahrplan 9 Minuten nach der fahrplanmäßigen Ankunft des Fernzugs abfährt. Fahrgäste können in den Regionalzug umsteigen, wenn nach Ankunft des Fernzugs mindestens 5 Minuten bis zur Abfahrt des Regionalzugs bleiben.

In den folgenden Teilaufgaben sollen Programmstücke als Teile der Hauptprogramm-Methode

```
public static void main (String [] unbenutzt)  { ... }
```

geschrieben werden. Alle Zeiten werden als ganze Zahlen in Minuten angegeben. Der fahrplanmäßige Abfahrtszeitpunkt des Fernzugs im Startbahnhof ist der Zeitpunkt 0.

a) Vereinbaren Sie drei eindimensionale Arrays `AnF`, `AbF` und `AbR`, um darin die fahrplanmäßigen Zeiten in jeder der 15 Städte zu speichern. Erstellen Sie ein Programmstück, das in `AnF` die Ankunftszeiten des Fernzugs, in `AbF` die Abfahrtszeiten des Fernzugs und in `AbR` die Abfahrszeiten des Regionalzugs einträgt. In allen drei Arrays soll jeder Index eine Stadt repräsentieren.

b) Durch verlängerte Fahrdauern auf den 14 Teilstrecken und verlängerte Aufenthaltsdauern in den Bahnhöfen entstehen Verspätungen. Wir nehmen an, dass Verspätungen bei der Weiterfahrt nicht wieder aufgeholt werden. Statt dessen pflanzt sich jede Verspätung bis zum Zielbahnhof fort. Auch die Abfahrt des Fernzugs im Startbahnhof und die Abfahrt der Regionalzüge kann sich verspäten. Eine Methode

```
int Zusatzdauer (char Zug, boolean Fahrt, int Stadt, int Tag, int Monat)
{ ... }
```

gebe die zusätzliche Dauer in Minuten an. Die Parameter haben folgende Bedeutung:

`Zug`	`'F'`	Fernzug
	`'R'`	Regionalzug
`Fahrt`	`false`	zusätzliche Aufenthaltsdauer im Bahnhof
	`true`	zusätzliche Fahrdauer von angegebener Stadt zur nächsten Stadt
`Stadt`	Index der betreffenden Stadt	
Tag	Tag und Monat bezeichen das Datum, an dem die zusätzliche Dauer aufgetreten ist	
Monat		

Die Methode `Zusatzdauer` kann als gegeben vorausgesetzt werden und muss hier nicht programmiert werden.

Vereinbaren Sie drei zweidimensionale Arrays `tAnF`, `tAbF` und `tAbR` und erstellen Sie ein Programmstück, das darin die tatsächlichen Zeiten der Züge in jeder der 15 Städte an allen Tagen des Monats Mai speichert. Rufen Sie dazu die Methode `Zusatzdauer` mit geeigneter Parametrisierung auf.

c) Erstellen Sie ein Programmstück, das die Anzahl der Städte ausgibt, in denen am 10. Mai ein Regionalzug mit ausreichender Umsteigedauer erreicht werden kann.

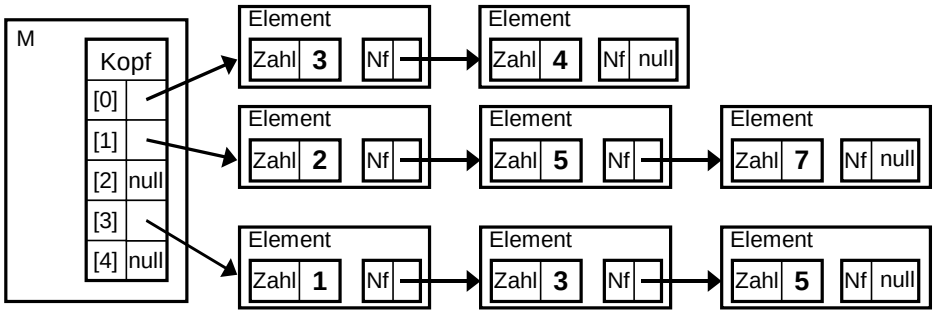
d) Erstellen Sie ein Programmstück, das die Stadt ausgibt, in der an den meisten Tagen des Monats Mai der Regionalzug mit ausreichender Umsteigedauer erreicht werden kann. Wenn dabei mehrere Städte gleich liegen, sollen alle diese Städte ausgegeben werden. Eine Stadt wird ausgegeben, indem der entsprechende Index ausgegeben wird.

Aufgabe 2.2: Zahlenmengen

Um eine Menge von Zahlen durch eine Datenstruktur zu repräsentieren werden Listen benutzt, in denen die Elemente aufsteigend sortiert sind. Ein Kopf-Verweis zeigt auf den Listenanfang. Einen Fuß-Verweis auf das Listenende gibt es nicht. Beachten Sie, dass in einer Menge ein Element nicht mehrfach vorkommen kann. Um fünf Mengen zu repräsentieren werden in der Klasse `M` fünf solche Listen verwendet, wobei das Array `Kopf` die fünf Verweise auf die Listenanfänge enthält. Die Klassen `Element` und `M` sind wie folgt definiert:

```
class Element
{ int Zahl;  Element Nf;

  Element (int Zahl, Element Nf)
  { this.Zahl = Zahl;  this.Nf = Nf;
  }
}

class M
{ Element[] Kopf = new Element[5];
  Element p(Element x, Element y)
  { if (x == null && y == null)
      return null;
    else if (x == null)
      return new Element(y.Zahl, p (null, y.Nf));
    else if (y == null)
      return new Element(x.Zahl, p (x.Nf, null));
    else if (x.Zahl == y.Zahl)
      return new Element(x.Zahl, p (x.Nf, y.Nf));
    else if (x.Zahl < y.Zahl)
      return new Element(x.Zahl, p (x.Nf, y));
    else
      return new Element(y.Zahl, p (x,    y.Nf));
  }  // Hinweis: In den Teilaufgaben e und f werden weitere
}    // Methoden in M eingefügt.
```

Durch Ausführung eines Programmstücks, das hier nicht dargestellt ist, sei bereits die folgende Datenstruktur aufgebaut, um 2 leere und 3 nicht leere Mengen zu repräsentieren:



a) Ist die Rekursion der Methode p linear? Ist sie schlicht?
Antworten mit Begründung.

b) Skizzieren Sie den Aufrufbaum, der durch Ausführung der Zuweisung

```
Kopf[2] = p(Kopf[1], Kopf[3]);
// Diese Zuweisung stehe in der Klasse M.
```

entsteht. Dabei wird empfohlen, Verweise auf Elemente einfach durch ein Pfeilsymbol und die Zahl des betreffenden Elements zu notieren, z.B. $\rightarrow$ 2.

c) Welche Elemente enthält die Liste, auf die `Kopf[2]` zeigt, nach Ausführung der Zuweisung aus Teilaufgabe b)? Notieren Sie die Zahlen der Elemente in der Reihenfolge, wie sie in der Liste auftreten.

d) Welche Operation auf Zahlenmengen wird durch die Methode p ausgeführt?

e) Programmieren sie in der Klasse `M` eine Methode `q`, welche die Differenz von zwei Zahlenmengen bildet. Es wird empfohlen, q ähnlich wie p aufzubauen. Hinweis: Die Differenz von zwei Mengen A und B enthält alle Elemente von A, die in B nicht vorkommen.

f) Programmieren Sie in der Klasse `M` eine Methode `s`, welche eine Zahl als zusätzliches Element in eine Menge einfügt. Geben Sie an, wie Ihre Methode `s` aufzurufen ist, um die Zahl 4 in die Menge einzufügen, auf die `Kopf[3]` verweist.

Aufgabe 2.3: Arrays aus Zeichen

Hinweis: In den folgenden Teilaufgaben ist die Anzahl der Elemente von Arrays unbekannt. Es darf jedoch davon ausgegangen werden, dass jedes Array mindestens ein Element besitzt.

a) Programmieren Sie zu dem Methoden-Kopf `boolean a(char[] p, char z)` einen Methoden-Rumpf, so dass `a` genau dann den Wert `true` liefert, wenn alle Elemente von `p` gleich dem als Parameter übergebenen Zeichen `z` sind.

b) Programmieren Sie zu dem Methoden-Kopf `boolean b(int[] s)` einen Methoden-Rumpf, so dass `b` genau dann den Wert `true` liefert, wenn jedes Elemente von `s` mit geradzahligem Index kleiner ist als jedes Element von `s` mit ungeradem Index.

c) Programmieren Sie zu dem Methoden-Kopf
`boolean c(char[] p, char[] q, int n)`
einen Methoden-Rumpf, so dass `c` genau dann den Wert `true` liefert, wenn `n` aufeinander folgende Elemente von `p` gleich `n` aufeinander folgenden Elementen von `q` sind. Dabei können die gleichen Elemente in `p` und `q` an verschiedenen Stellen stehen, d.h. ab verschiedenen Indizes beginnen.

Fünf Beispiele zur Verdeutlichung:

p	q	n	c liefert
{'F','i','n','g','e','r'}	{'W','e','i','n','g','l','a','s'}	3	true
{'S','c','h','m','e','r','z'}	{'h','e','r','z','l','i','c','h'}	4	false
{'o','r','g','a','n','i','s','c','h'}	{'V','o','r','g','a','n','g'}	5	true
{'N','a','s','e'}	{'N','a','s','e'}	5	false
{'F','r','e','i','z','e','i','t'}	{'w','e','i','c','h'}	2	true

Aufgabe 2.4: Zahlen aus Array in Liste einsortieren

Hinweis: In den folgenden Teilaufgaben ist die Anzahl der Elemente von Arrays unbekannt. Es darf jedoch davon ausgegangen werden, dass jedes Array mindestens zwei Elemente besitzt.

a) Programmieren Sie zu dem Methoden-Kopf `boolean g(int[] a)` einen Methoden-Rumpf, so dass `g` genau dann den Wert `true` liefert, wenn es in dem Array `a` zwei unmittelbar aufeinander folgende Elemente gibt, die gleich sind.

b) Programmieren Sie zu dem Methoden-Kopf `boolean m(int[] a)` einen Methoden-Rumpf, so dass `m` genau dann den Wert `true` liefert, wenn `a` ein Element enthält, das gleich dem arithmetischen Mittel (d.h. gleich dem durchschnittlichen Wert) aller Elemente von `a` ist.

c) Gegeben seien die folgenden Klassen:

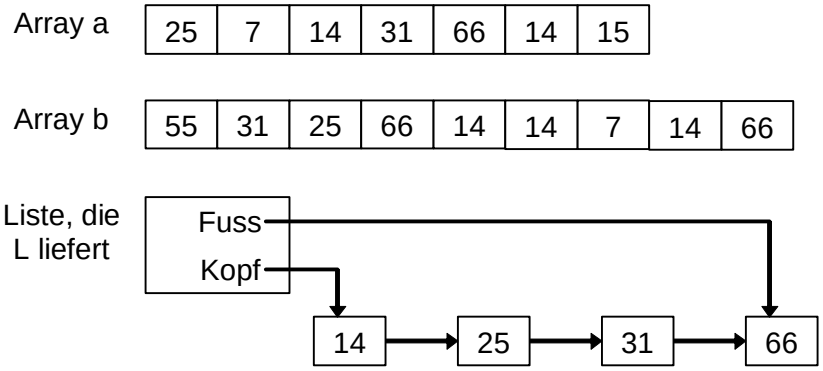
```
class Liste
{ Element Kopf, Fuss;  // Listen-Anfang und -Ende.
  Liste()  { Kopf = Fuss = null; }

}

class Element   // Ein Element der Liste.
{ int Zahl;     // In das Element eingetragene Zahl.
  Element Nf;   // Nachfolge-Element in der Liste.
  Element (int Zahl)  { this.Zahl = Zahl;  Nf = null; }
}
```

Programmieren Sie zu dem Methoden-Kopf `Liste L(int[] a, int[] b)` einen Methoden-Rumpf, so dass `L` eine aufsteigend sortierte Liste liefert, die alle Zahlen enthält, die

- im Array `a` vorkommen
- und im Array `b` vorkommen
- und positiv und zweiziffrig sind (d.h. im Intervall [10, 99] liegen).

In der Liste, die L liefert, darf keine Zahl mehrfach vorkommen. Beachten Sie, dass die Listenelemente aufsteigend sortiert sein müssen.

Ein Beispiel zur Verdeutlichung:



Aufgabe 2.5: LKW-Reifen

Ein australischer Fuhrunternehmer besitzt 53 baugleiche Lastzüge mit je 24 Rädern. Auf den unebenen Straßen kommt es immer wieder zu Reifenschäden. Wenn ein Fahrer einen Reifenschaden meldet, trägt ihn der Fuhrunternehmer in folgende Tabelle ein (in der einige Beispiel-Einträge zu sehen sind):

	`Lastzug`	`Rad`
1. Eintrag	19	4
2. Eintrag	3	21
3. Eintrag	44	9
weitere Einträge	...	...

In der Tabelle haben die beiden Spalten folgende Bedeutung:

`Lastzug` Nummer des Lastzugs, der einen Reifenschaden hat (Zahlenbereich 1, ... , 53)

`Rad` Nummer des Rads mit dem Reifenschaden (Zahlenbereich 1, ... , 24)

Am Ende des Jahres weist die Tabelle 121 Einträge auf. Sie werden von einem Java-Programm in zwei eindimensionale Arrays wie folgt eingelesen:

```
int[] Lastzug = new int[121];
```
nimmt die Angaben aus der ersten Spalte auf.

```
int[] Rad = new int[121];
```
nimmt die Angaben aus der zweiten Spalte auf.

Dieses Java-Programm sei bereits gegeben und muss hier nicht realisiert werden. Ergänzen Sie das Programm entsprechend den folgenden Teilaufgaben (Hinweis: Es sind nur die entsprechenden Programmstücke verlangt. Die Methodenköpfe müssen hier nicht angegeben werden):

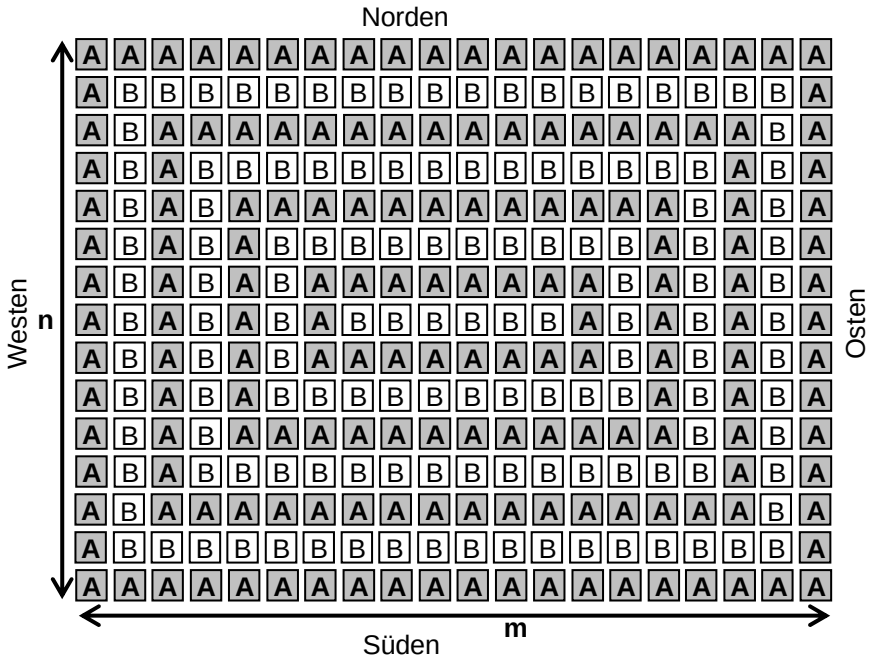
a) Vereinbaren Sie ein zweidimensionales Array `Schaden`, um für jedes Rad die Anzahl der Reifenschäden angeben zu können. Schreiben Sie ein Programmstück, das die Anzahl der Reifenschäden von Rad y des Lastzugs x in dem Array-Element `Schaden[x][y]` ablegt.

b) Schreiben Sie ein Programmstück, das die Nummer des Lastzugs ausgibt, welcher die meisten Reifenschäden hatte. Wenn mehrere Lastzüge den gleichen Höchstwert erreichen, sollen die Nummern all dieser Lastzüge ausgegeben werden.

c) Schreiben Sie ein Programmstück, das ausgibt, wie viele Räder insgesamt das ganze Jahr über keinen Reifenschaden aufgewiesen haben.

Aufgabe 2.6: Solarenergie

Zur Gewinnung von Sonnenenergie sind Solarzellen in einem rechteckigen Feld aufgestellt. Von Norden nach Süden des Feldes sind n, von Westen nach Osten des Feldes m Solarzellen angeordnet. Die Skizze zeigt den Fall n = 15 und m = 20.



Es werden zwei Typen A und B von Solarzellen verwendet, wobei die Aufstellungsorte „ringförmig“ gewählt sind. An den Rändern stehen nur Solarzellen vom Typ A, im nächst-inneren Ring nur solche vom Typ B, im innen anschließenden Ring nur solche vom Typ A, usw. (siehe Skizze). Wegen der ungleichmäßigen Sonneneinstrahlung weichen die Leistungen der Solarzellen leicht voneinander ab. In einem zweidimensionalen Array

```
float[][] s = new float[n][m]
```

seien die erzielten Leistungswerte jeder Solarzelle gespeichert worden. Die Methoden der folgenden Teilaufgabe werden mit den aktuellen Parametern `s`, `n` und `m` aufgerufen und sollen für beliebige $n \geq 1$ und $m \geq 1$ korrekt arbeiten.

a) Programmieren Sie zu dem Methodenkopf
`float Sum(float[][] s, int n, int m)`
einen Methodenrumpf, der die Summe der Leistungen aller Solarzellen liefert.

b) Programmieren Sie zu dem Methodenkopf
`int Abstand(float[][] s, int n, int m, int i, int j)`
einen Methodenrumpf, der den Abstand der Solarzelle mit Index `[i][j]` zum nächstgelegenen Rand des Feldes liefert. Für die Solarzellen des äußeren Rings (alle Typ A) ist dies der Wert 0, für

die Solarzellen des nächst-inneren Rings (alle Typ B) der Wert 1, für den nächsten weiter innen liegenden Ring (alle Typ A) der Wert 2, usw.

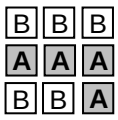
c) Programmieren Sie zu dem Methodenkopf
`char Typ(float[][] s, int n, int m, int i, int j)`
einen Methodenrumpf, der den Typ der Solarzelle mit Index `[i][j]` angibt (Rückgabewert entweder `'A'` oder `'B'`).

d) Programmieren Sie zu dem Methodenkopf
`float SumA(float[][] s, int n, int m)`
einen Methodenrumpf, der die Summe der Leistungen aller Solarzellen vom Typ A liefert.

e) Programmieren Sie zu dem Methodenkopf
`int AnzA(float[][] s, int n, int m)`
einen Methodenrumpf, der die Anzahl der Solarzellen vom Typ A liefert, für die gilt

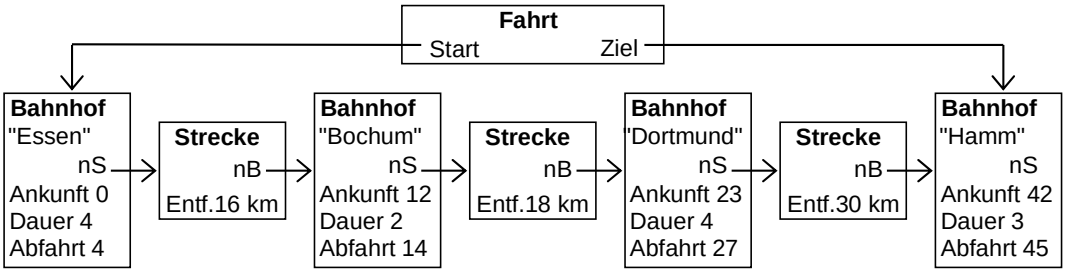
- dass sie nicht am Rand des rechteckigen Feldes liegen
- und dass sie eine höhere Leistung erzielen als jede der benachbarten Solarzellen vom Typ B.

Als benachbart gelten die an einer Kante oder eine Ecke angrenzenden Solarzellen. Die nebenstehende Abbildung zeigt für die Solarzelle vom Typ A in der Mitte des Bildes die 5 benachbarten Solarzellen vom Typ B.



Aufgabe 2.7: Liste für eine Zugfahrt

Ausgehend von einem Startbahnhof fährt ein Zug von Bahnhof zu Bahnhof und erreicht schließlich den Zielbahnhof. Die Fahrt wird durch Objekte der Klasse `Bahnhof` und Objekte der Klasse `Strecke` modelliert, die entsprechend der Zugfahrt in einer Liste angeordnet sind. In einem Objekt der Klasse `Fahrt` wird durch `Start` und `Ziel` auf den Start- bzw. Zielbahnhof verwiesen, wie das Beispiel einer Fahrt von Essen nach Hamm zeigt.



Gegebene Klassen:

```
class Fahrt   { Bahnhof Start, Ziel; }

class Bahnhof { String  Name;          // Name des Bahnhofs.
                Strecke nS;            // Strecke, die als nächste befahren wird.
                int     Ankunft;       // Zeitpunkt der Ankunft im Bahnhof (in Minuten).
                int     Dauer;         // Dauer des Halts im Bahnhof (in Minuten).
                int     Abfahrt; }     // Zeitpunkt der Abfahrt (in Minuten).
```

```
class Strecke { Bahnhof nB;          // nächster Bahnhof, zu dem der Zug hinfährt.
                int     Entfernung; } // Länge der Strecke (in Kilometern).
```

Alle Zeitpunkte werden ganzzahlig in Minuten seit der Bereitstellung des Zuges im Startbahnhof angegeben (folglich ist die Ankunft im Startbahnhof stets zum Zeitpunkt 0). Auf allen Strecken fährt der Zug exakt mit 120 km/h. Er legt also pro Minute zwei Kilometer zurück. Die Fahrdauer für eine Strecke wird ganzzahlig in Minuten angegeben (ggf. abgerundet).

a) Konstruktor: `Bahnhof(String Name, int Ankunft, int Dauer)`

   Programmieren Sie den Rumpf dieses Konstruktors, der die Variablen der Klasse Bahnhof entsprechend belegt. `Abfahrt` soll auf `Ankunft` + `Dauer` und `nS` auf `null` gesetzt werden.

b) Konstruktor: `Bahnhof(Fahrt f, int Entfernung, String Name, int Dauer)`

   Programmieren Sie den Rumpf dieses Konstruktors. Er soll zusätzlich ein neues `Strecke`-Objekt mit der angegebenen `Entfernung` erzeugen und an den bisherigen Zielbahnhof der Fahrt `f` als Nachfolger anfügen. Das neue `Bahnhof`-Objekt soll an die neue Strecke als Nachfolger angefügt werden. Die Ankunftszeit ist aus der Abfahrtszeit im bisherigen Zielbahnhof und der Fahrzeit für die neue Strecke zu errechnen. Um die Variablen des neuen Bahnhof-Objekts zu belegen, soll der in Teilaufgabe a) erstellte Konstruktor aufgerufen werden.

c) Bezeichnet man die Konstruktoren aus den Teilaufgaben a) und b) als „überladen“? (ja / nein)
   Bezeichnet man sie als „überschrieben“? (ja / nein)

d) Methode in der Klasse `Fahrt`: `void EssenHamm()`

   Programmieren Sie den Rumpf dieser Methode, welche die in dem obigen Bild skizzierte Zugfahrt von Essen nach Hamm mit den angegebenen Werten aufbaut.

e) Methode in der Klasse `Fahrt`: `boolean verspaetet(int Zusatzdauer)`

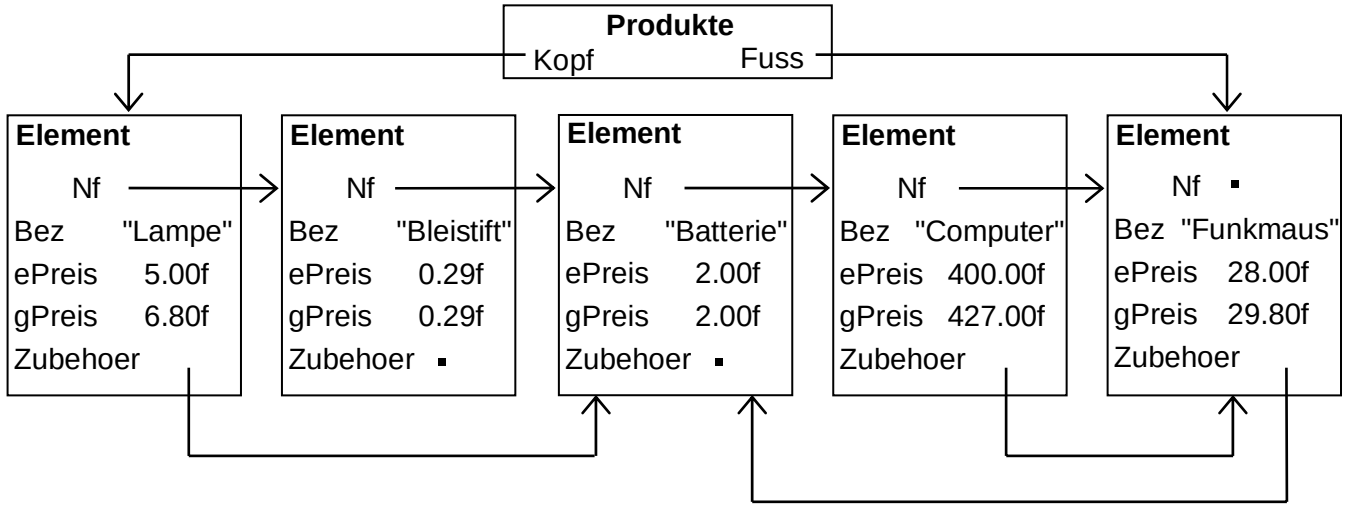
   Programmieren Sie den Rumpf dieser Methode. Es wird angenommen, dass sich bei einer Zugfahrt die Aufenthaltsdauer in jedem Bahnhof um die in Minuten angegebene `Zusatzdauer` erhöht. Die Geschwindigkeit auf allen Strecken ist unverändert 120 km/h. Die Methode soll genau dann `true` liefern, wenn sich durch die Zusatzdauern die Gesamtfahrzeit (gerechnet von der Abfahrt im Startbahnhof bis zur Ankunft im Zielbahnhof) um mindestens 10% erhöht.

f) Methode in der Klasse `Fahrt`: `boolean zielnah(Bahnhof b)`

   Programmieren Sie den Rumpf dieser Methode. Sie soll genau dann `true` liefern, wenn die Entfernung vom Bahnhof `b` zum Zielbahnhof geringer ist als die Entfernung vom Bahnhof `b` zum Startbahnhof. Es darf angenommen werden, dass der Bahnhof `b` einer der Bahnhöfe der betreffenden Fahrt ist.

Aufgabe 2.8: Produkte-Liste

Ein Händler bietet verschiedene Produkte an. Über jedes Produkt soll Information in einem `Element` einer `Produkte`-Liste gespeichert werden: die Bezeichnung `Bez`, der Einzelpreis `ePreis`, der Gesamtpreis `gPreis` sowie ggf. ein Verweis `Zubehoer` auf ein anderes Produkt, das als Zubehör gilt (z.B. eine Batterie für eine Taschenlampe). Der Fall, dass ein Produkt mehr als 1 Zubehör-Produkt besitzt, ist hier ausgeschlossen. Der Gesamtpreis `gPreis` wird in Teilaufgabe d) erläutert.



Gegebene Klassen:

```
class Produkte  { Element Kopf, Fuss; }  // Anfang und Ende der Liste der Produkte.

class Element { Element Nf;           // Verweis auf das Nachfolger-Produkt in der Liste.
                String  Bez;          // Bezeichnung eines Produkts.
                float   ePreis;       // Einzelpreis eines Produkts in Euro.
                float   gPreis;       // Gesamtpreis eines Produkts in Euro.
                Element Zubehoer; }   // Verweis auf ein benötigtes Zubehör-Produkt.
```

a) Konstruktor: `Element(String Bez, float ePreis, Produkte Liste)`

Programmieren Sie den Rumpf dieses Konstruktors, der die Variablen der Klasse `Element` entsprechend belegt. `gPreis` soll auf `0.0f` und `Zubehoer` auf `null` gesetzt werden. Außerdem soll das Produkt am Ende der angegebenen `Liste` eingefügt werden. Vor dem Einfügen kann die Liste noch leer sein. Nach dem Einfügen müssen der `Kopf`- und der `Fuss`-Verweis korrekt auf den Anfang bzw. das Ende der Liste zeigen.

b) Beschreiben Sie die Wirkung des `String`-Vergleichs durch == sowie die Wirkung des `String`-Vergleichs durch `equals`.

c) Methode in der Klasse Produkte: `void Zubehoer(String a, String b)`

Programmieren Sie den Rumpf dieser Methode. Sie soll in das mit a bezeichnete Produkt eintragen, dass Produkt b sein Zubehör ist. Es darf davon ausgegangen werden, dass alle Produkte verschiedene Bezeichnungen tragen. Wenn eine der Produktbezeichnungen nicht in der Liste vorkommt, soll kein Zubehör eingetragen werden.

d) Methode in der Klasse `Produkte`: `void Gesamtpreis()`

Programmieren Sie den Rumpf dieser Methode. Sie soll in jedes Produkt der `Produkte`-Liste den Gesamtpreis `gPreis` eintragen. Er ist die Summe aus dem Einzelpreis des Produkts und die um 10% verminderten Einzelpreise für alle seine Zubehör-Produkte (d.h. Zubehör, Zubehör des Zubehörs, Zubehör des Zubehörs des Zubehörs usw.). Das Bild oben zeigt Zahlenbeispiele zu dieser Teilaufgabe.

e) Methode in der Klasse `Produkte`: `void Angebot()`

Programmieren Sie den Rumpf dieser Methode, welche die Objektstruktur für genau das in dem obigen Bild skizzierte Angebot an Produkten aufbaut und die entsprechenden Variablenwerte einträgt.

f) Methode in der Klasse `Produkte`: `float zweitteuerstesProdukt()`

Programmieren Sie den Rumpf dieser Methode. Sie soll den Preis des zweitteuersten Produkts (bezogen auf den Einzelpreis `ePreis`) aus der `Produkt`-Liste angeben. In dem Beispiel aus dem obigen Bild ist dies der Einzelpreis `28.00f`. Sonderfälle: Wenn zwei verschiedene Produkte den gleichen höchsten Einzelpreis aufweisen, dann soll deren Preis angegeben werden. Sollte die `Produkt`-Liste leer sein oder nur ein Element enthalten, wird `0.0f` zurückgegeben.

Aufgabe 2.9: Listen und Arrays

Eine Liste sei (ähnlich wie in der Vorlesung) durch die folgenden Klassen definiert. Zusätzlich sind die Klassen `ElemA` und `ElemB` definiert. In jeder Klasse ist ein Konstruktor angegeben.

```
class Liste   { Element Kopf = null;   // Anfang der Liste.  Kein Fuß-Verweis.

                Liste(char[] a, int[] b)
                { int n = Math.max(a.length, b.length);
                  for (int i = 0;  i < n;  i++)
                  { if (i < a.length)  Kopf = new ElemA(a[i], Kopf);
                    if (i < b.length)  Kopf = new ElemB(b[i], Kopf);
                  }
                }

class Element { Element Nf;            // Nachfolger in der Liste.

                Element(Element e)
                { Nf = e;
                }
              }

class ElemA extends Element
              { char Zeichen;          // Zeichen in ElemA.

                ElemA(char x, Element e)
                { super(e);  Zeichen = x;
                }
              }

class ElemB extends Element
              { int Zahl;              // Zahl in ElemB.

                ElemB(int x, Element e)
                { super(e);  Zahl = x;
                }
              }
```

a) Geben Sie an, welche dieser Klassen Oberklassen und welche Unterklassen sind.

b) Skizzieren Sie die Struktur, die durch Ausführung des folgenden Programmstücks entsteht:

```
char[] c = {'g', 'b'};
int[]  z = {5, 7, 2};
Liste L = new Liste(c, z);
```

Zeichnen Sie für jedes Objekt ein Kästchen, tragen Sie die Werte ein und symbolisieren Sie Verweise durch Pfeile.

c) Methode in der Klasse `Liste`: **`int Summe()`**

Programmieren Sie den Rumpf dieser Methode, welche die Summe aus allen Listenelementen bildet. Dabei wird für Elemente, die eine Zahl enthalten (`ElemB`), die Zahl addiert. Für Elemente, die Zeichen enthalten (`ElemA`), wird stets 10 addiert. Ihre Lösung darf nicht `instanceof` enthalten. Dazu müssen Sie auch geeignete Methoden in den Klassen `ElemA` und `ElemB` programmieren. Geben Sie an, welche Methode in welcher Klasse angeordnet ist.

d) Welchen Vorteil weist die Lösung ohne `instanceof` aus Teilaufgabe c) gegenüber einer Lösung mit `instanceof` auf, wenn Element um zusätzliche Klassen erweitert wird

e) Methode in der Klasse `Liste`: **`char[] erzeugeArray()`**

Programmieren Sie den Rumpf dieser Methode, die ein Array liefert, das die Zeichen von allen Listenelementen des Typs `ElemA` enthält. Das Array soll genau so viele Elemente enthalten, wie die Liste Elemente vom Typ `ElemA` enthält.

3 Bäume

Aufgabe 3.1: Binäre Suchbäume

Die Knoten eines binären Suchbaum seien durch die folgende Klasse definiert:

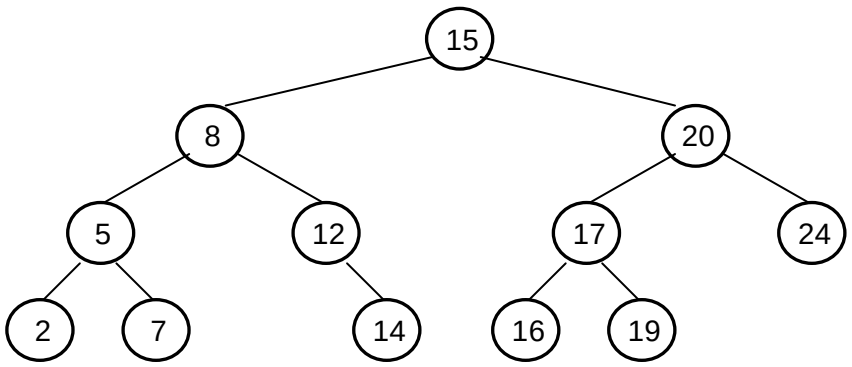
```
class Knoten
{ int Zahl;  Knoten links, rechts;
}
```

In der Klasse `Suchbaum` verweist die Variable `Wurzel` auf die Wurzel.

a) Zeichnen Sie den Suchbaum, der durch Einfügen der Werte 25, 36, 30, 28, 6, 44, 42, 40, 47, 24 (in dieser Reihenfolge) in einen leeren, binären Suchbaum entsteht.

b) Geben Sie eine Einfügereihenfolge der Werte aus Teilaufgabe a) an, die zu einem binären Suchbaum mit der schlechtesten durchschnittlichen Suchzeit führt.

c) In welcher Reihenfolge werden die Knoten des folgenden binären Suchbaums bei Tiefendurchlauf in PreOrder-, InOrder- und PostOrder-Reihenfolge sowie bei einem Breitendurchlauf besucht? Tragen Sie die Werte der Knoten entsprechend des jeweiligen Durchlaufs in die Tabelle ein.



PreOrder-Tiefendurchlauf												
InOrder-Tiefendurchlauf												
PostOrder-Tiefendurchlauf												
Breiten-durchlauf												

d) In der Klasse `Suchbaum` sei die folgende Methode gegeben;

```
int m(Knoten k, int x)
{ if      (k == null)                        return 0;
  else if (x <= k.Zahl && k.links  == null)  return k.Zahl;
  else if (x <= k.Zahl && k.links  != null)  return m(k.links, x);
  else if (x >  k.Zahl && k.rechts == null)  return k.Zahl;
  else
  { int r = m(k.rechts, x);
    if (x <= r)  return k.Zahl;
    else         return r;
  }
}
```

Welchen Werte liefert die Methode bei den drei folgenden Aufrufen:

`m(Wurzel, 15)`

`m(Wurzel, 16)`

`m(Wurzel, 19)`

e) Bei der in Teilaufgabe d) beschriebenen Methode lässt sich jede einzelne Rekursionsstufe wie folgt charakterisieren:

> Wenn x ≤ k.Zahl ist, wird versucht im Suchbaum nach links zu gehen,
> wenn x > k.Zahl ist, wird versucht im Suchbaum nach rechts zu gehen.
> Ist dies nicht möglich, weil der entsprechende Verweis null ist, wird k.Zahl zurückgegeben.
> Wenn nach rechts gegangen wird, gibt es eine Besonderheit:
> Sollte der im rechten Teilbaum gefundene Wert ≥ x sein, so wird k.Zahl zurückgegeben.

Beschreiben Sie, welche Wirkung ein Aufruf `m(Wurzel, x)` insgesamt hat.

f) Ist die Rekursion der Methode aus Teilaufgabe d) linear?
Ist die Rekursion der Methode aus Teilaufgabe d) schlicht?

Aufgabe 3.2: Vervollständigung von Bäumen

Gegeben seien die folgenden Klassen:

```
class Baum  // Ein Binärbaum.
{ Knoten Wurzel;    // Wurzel des Binärbaums.
  int  h() { ... }  // Höhe des Binärbaums.
  int  e() { ... }  // Anz. Knoten mit genau 1 Nachfolger.
  void v() { ... }  // Vervollständigung des Binärbaums.
  ...               // weitere Methoden.

}

class Knoten        // Ein Knoten des Binärbaums.
{ int Zahl;         // In den Knoten eingetragene Zahl.
  Knoten links, rechts;  // Nachfolger im Binärbaum.
```

```
  Knoten (int Zahl)
  { this.Zahl = Zahl;  links = rechts = null;
  }
}
```

Hinweis: Zur Lösung der folgenden Teilaufgaben können in der Klasse `Baum` weitere Methoden programmiert werden, die von den Methoden `h`, `e` und `v` aufgerufen werden können.

a) Programmieren Sie den Rumpf der Methode `h` in der Klasse `Baum`, so dass `h()` die Höhe des Baums liefert.

b) Programmieren Sie den Rumpf der Methode `e` in der Klasse `Baum`, so dass `e()`

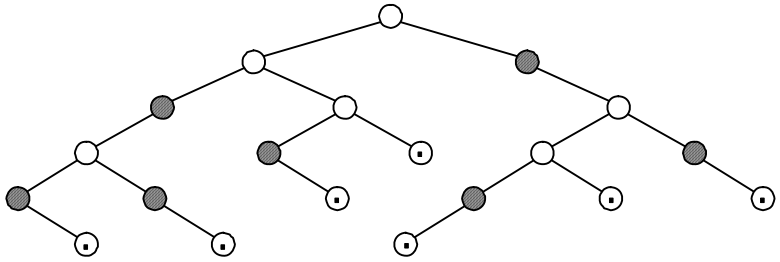
- die Anzahl aller Knoten mit genau einem Nachfolger liefert
- und alle Knoten mit genau einem Nachfolger in Pre-Order-Reihenfolge ausdruckt. Dabei soll für jeden auszudruckenden Knoten eine Ausgabezeile erzeugt werden, welche die Zahl des betreffenden Knotens enthält.

  Beispiel:

  ○ Knoten mit genau 2 Nachfolgern

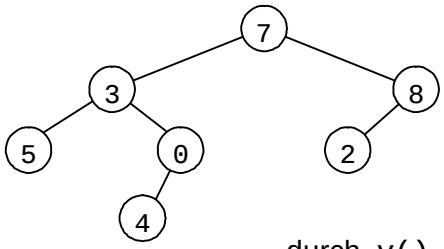
  ◍ Knoten mit genau einem Nachfolger

  ⊙ Knoten ohne Nachfolger (Blätter)

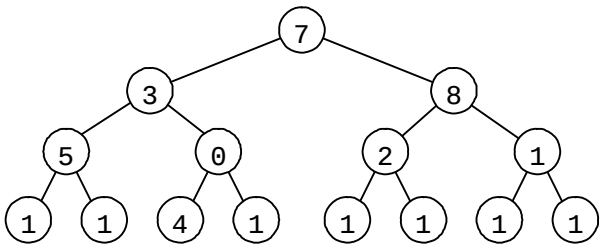


c) Programmieren Sie den Rumpf der Methode `v` in der Klasse `Baum`, so dass `v()` den Baum durch Hinzufügen von Knoten vervollständigt, ohne seine Höhe zu verändern. Ein vollständiger Binärbaum besitzt die maximale Anzahl von Knoten bei gegebener Höhe. In die hinzugefügten Knoten soll die Zahl 1 eingetragen werden.

Beispiel: unvollständiger Binärbaum



durch `v()` vervollständigter Binärbaum



Aufgabe 3.3: Inhalte von Blättern in Array übernehmen

Gegeben seien folgende Klassen:

```
class Baum  // Ein Binärbaum.
{ Knoten Wurzel;// Wurzel des Binärbaums.
  int b() { ... }     // Anzahl der Blätter.
  int[] z() { ... }  // Array, das Zahlen der Blätter enthält.
  boolean p(int x) { ... } // Prüfung, ob x in Blatt vorkommt.
  ...// weitere Methoden.

}

class Knoten  // Ein Knoten des Binärbaums.
{ int Zahl;              // Zahl eines Knotens.
  Knoten links, rechts;  // Linker, rechter Nachfolger-Knoten.
  Knoten (int Zahl)
  { this.Zahl = Zahl;  links = rechts = null;
  }
}
```

Hinweis: Zur Lösung der folgenden Teilaufgaben können in der Klasse `Baum` weitere Methoden programmiert werden, die von den Methoden `b`, `z` und `p` aufgerufen werden können.

a) Programmieren Sie den Rumpf der Methode `b` in der Klasse Baum, so dass `b()` die Anzahl der Blätter des Baums liefert.

b) Programmieren Sie den Rumpf der Methode `z` in der Klasse Baum, so dass `z()` ein Array liefert, wobei die Array-Elemente die Zahlen enthalten, die in den Blättern des Baums eingetragen sind. Die Reihenfolge der Blatt-Zahlen im Array soll der Reihenfolge der Blatt-Besuche beim In-Order-Durchlauf durch den Baum entsprechen. Das Array soll genau so viele Elemente enthalten, wie es Blätter im Baum gibt. Falls der Baum leer ist (d.h. keinen Knoten enthält), soll `z()` den Wert `null` liefern.

c) Programmieren Sie den Rumpf der Methode `p` in der Klasse Baum, so dass `p(x)` genau dann den Wert `true` liefert, wenn die Zahl `x` in einem Blatt des Baums eingetragen ist.

Aufgabe 3.4: Blätter entfernen

Gegeben seien die aus der Vorlesung bekannten Klassen `Baum` und `Knoten`, mit denen ein Binärbaum dargestellt werden kann. In der Klasse `Baum` sind zusätzlich eine Methode `a` und eine Methode `b` vereinbart.

```
class Baum
{ Knoten Wurzel;

  boolean a() // Siehe Teilaufgabe a.
  { .....
  }

  void b()   // Siehe Teilaufgabe b.
  { .....
  }
```

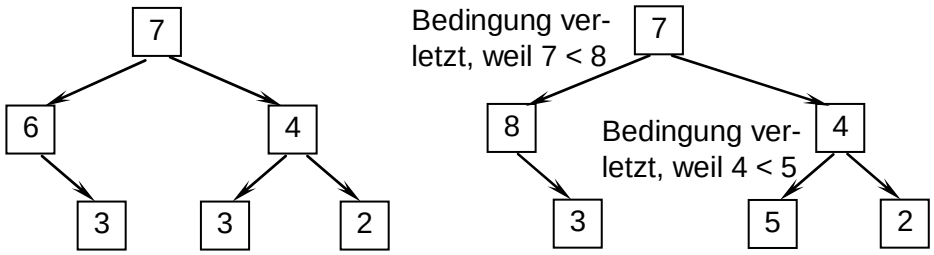
```
  .....
}
```

```
class Knoten
{ int Zahl;  Knoten links, rechts;
}
```

Zur Lösung der folgenden Teilaufgaben sind die Rümpfe der Methoden `a` und `b` zu formulieren. Bei Bedarf sind in der Klasse `Baum` weitere Methoden zu definieren.

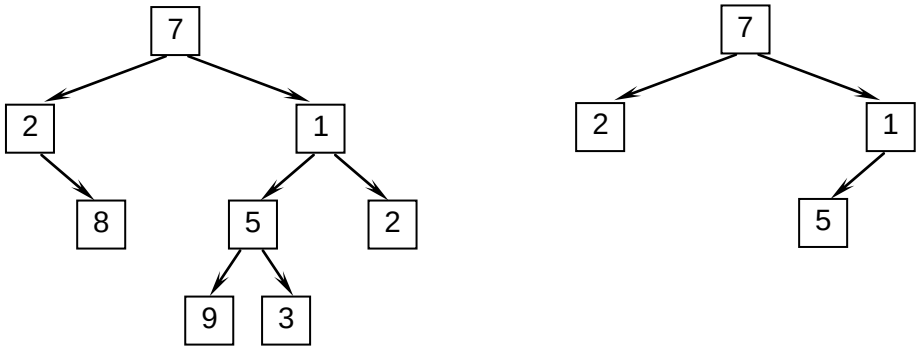
a) Schreiben Sie den Rumpf der Methode `a`, so dass ein Aufruf von `a` genau dann `true` liefert, wenn für alle Knoten des Baums gilt, dass ihre Variable `Zahl` echt größer ist als die Variablen `Zahl` ihrer direkten Nachfolger (soweit diese vorhanden sind). Auch für einen leeren Baum soll `true` geliefert werden.

   Ein Beispiel zu Teilaufgabe a: Für den links dargestellten Baum liefert ein Aufruf der Methode `a` den Wert `true`, für den rechts dargestellten Baum dagegen `false`.



b) Schreiben Sie den Rumpf der Methode `b`, so dass bei einem Aufruf von `b` alle Blätter des Baums entfernt werden.

   Ein Beispiel zu Teilaufgabe b: Ein Aufruf der Methode `b` soll den links dargestellten Baum so verändern, dass daraus der rechts dargestellte Baum wird.



Aufgabe 3.5: Array und Suchbaum

Zur sortierten Abspeicherung von ganzen Zahlen wird ein int-Array wie folgt benutzt: In Element 0 steht die Anzahl n der Einträge. In den Elementen 1 bis n stehen die eingetragenen Zahlen in aufsteigender Reihenfolge. Die restlichen Elemente n+1, n+2 usw. bleiben unbenutzt und können beliebige Werte aufweisen. Eine solches Array wird hier „sortiertes Array“ genannt.

Hier ein Beispiel eines sortierten Arrays mit 8 Elementen und 6 Einträgen (Das Zeichen ***** bezeichnet unbenutzte Elemente, die einen beliebigen Wert aufweisen können):

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
6	**3**	**11**	**18**	**21**	**21**	**30**	*	*

a) Programmieren Sie zu dem Methoden-Kopf `int[] neu()` einen Methoden-Rumpf, so dass der Aufruf `neu()` ein leeres sortiertes Array mit Platz für 4 Einträge liefert.

[0]	[1]	[2]	[3]	[4]
0	*	*	*	*

b) Programmieren Sie zu dem Methoden-Kopf `boolean sortiert(int[] a)` einen Methoden-Rumpf, so dass der Aufruf `sortiert(a)` genau dann `true` liefert, wenn a ein sortiertes Array ist.

c) Programmieren Sie zu dem Methoden-Kopf
`int[] sortiereein(int[] a, int e)`
einen Methoden-Rumpf, der die Zahl e in ein sortiertes Array a einsortiert. Dabei müssen evtl. einige Einträge in dem Array verschoben werden. In dieser Teilaufgabe darf davon ausgegangen werden, dass in a noch mindestens ein Element frei ist, um einen Eintrag e aufzunehmen. Der Aufruf `sortiereein(a, e)` soll einen Verweis auf das sortierte Array liefern, in das e einsortiert ist. Dazu ein Beispiel:

Parameter a:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
5	**3**	**4**	**8**	**8**	**14**	*	*	*

Parameter e: **7**

Gelieferter Rückgabewert:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]
6	**3**	**4**	**7**	**8**	**8**	**14**	*	*

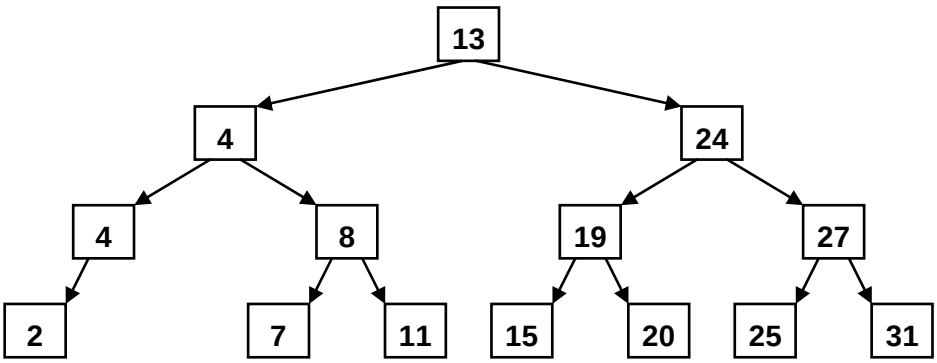
d) Programmieren Sie zu dem Methoden-Kopf `int suche(int[] a, int e)` einen Methoden-Rumpf, der den Index des Eintrags e liefert. Falls e in dem sortierten Array a nicht vorkommt, soll der Wert 0 geliefert werden.

e) Programmieren Sie zu dem Methoden-Kopf `Knoten Suchbaumwurzel(int[] a)` einen Methoden-Rumpf, der die Einträge des sortierten Arrays a in einen Suchbaum übernimmt und einen Verweis auf dessen Wurzel liefert. Dabei soll der Suchbaum nicht zu einer linearen Liste entarten, sondern möglichst vollständig sein. Aus diesem Grund soll ein Eintrag in der Mitte des sortierten Arrays a die Wurzel des Suchbaums bilden. Ein Eintrag in der Mitte der verbleibenden linken Hälfte des sortierten Arrays soll den linken Nachfolger der Wurzel bilden. Entsprechend soll ein Eintrag in der Mitte der verbleibenden rechten Hälfte des sortierten Arrays den rechten Nachfolger der Wurzel bilden. Dieses Verfahren wird fortgesetzt, bis der Suchbaum jedes Elemente des sortierten Arrays a genau einmal enthält. Es empfiehlt sich, zur Lösung dieser Aufgabe eine rekursive Methode zu programmieren

Dazu ein Beispiel. Das Array a sei:

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]	[16]
14	**2**	**4**	**4**	**7**	**8**	**11**	**13**	**15**	**19**	**20**	**24**	**25**	**27**	**31**	*	*

Daraus wird der folgende Suchbaum erzeugt:



Hinweis: Für den Suchbaum sei die Klasse `Knoten` wie üblich definiert:

```
class Knoten
{ int Zahl;  Knoten links, rechts;
  Knoten(int Zahl)
  { this.Zahl = Zahl;  links = rechts = null; }
}
```

f) Vergleichen Sie ein sortiertes Array, eine sortierte Liste und einen Suchbaum jeweils für eine große Anzahl von zufälligen Einträgen. Geben Sie für die folgenden Operationen und Datenstrukturen durch Ankreuzen an, ob sie vergleichsweise schnell oder langsam ausgeführt werden. Eine Operation gilt als langsam, wenn im Mittel die Hälfte der Elemente oder fast alle Elemente durchlaufen werden müssen.

	Sortiertes Array	Sortierte Liste	Suchbaum
Einsortieren	□ schnell □ langsam	□ schnell □ langsam	□ schnell □ langsam
Suchen	□ schnell □ langsam	□ schnell □ langsam	□ schnell □ langsam
Löschen	□ schnell □ langsam	□ schnell □ langsam	□ schnell □ langsam

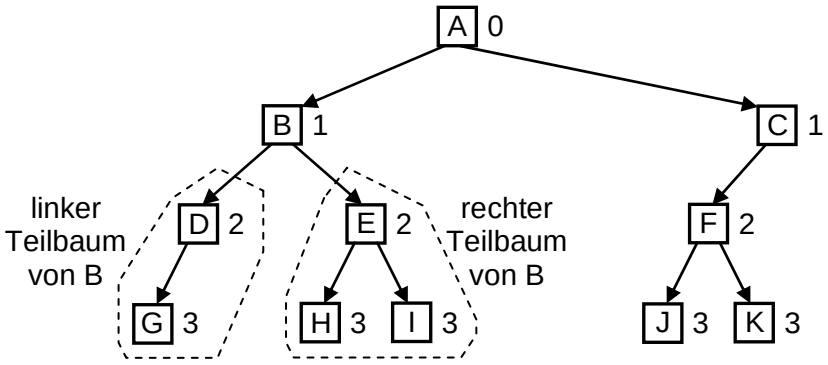
Aufgabe 3.6: Teilbäume

Ein binärer Baum wird durch die Klassen `Baum` und `Knoten` repräsentiert:

```
class Baum
{ Knoten Wurzel;  // Wurzel des Baums.
  ...  // Hier sind alle Methoden der folgenden Teilaufgaben angesiedelt.
}

class Knoten
{ String Bez;             // Bezeichnung des Knotens.
  Knoten links, rechts;   // Linker bzw. rechter Nachfolger.

  Knoten (String Bez)
  { this.Bez = Bez;  links = rechts = null;
```

```
  }
}
```

Beachten Sie, dass in dieser Aufgabe kein Suchbaum, sondern ein beliebiger binärer Baum betrachtet wird.

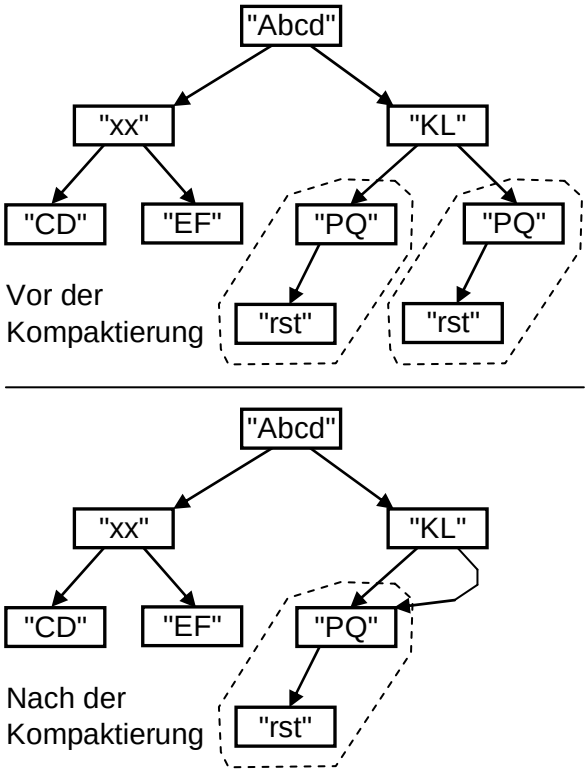


Unter dem linken Teilbaum eines Knotens k versteht man den Baum, dessen Wurzel `k.links` ist. Entsprechend ist `k.rechts` Wurzel des rechten Teilbaums von Knoten k.

a) Programmieren Sie zu dem Methoden-Kopf
`boolean TB(Knoten k1, Knoten k2)`
einen Methoden-Rumpf, der genau dann den Wert `true` liefert, wenn Knoten `k1` in einem der beiden Teilbäume von Knoten `k2` liegt.

b) Programmieren Sie zu dem Methoden-Kopf `boolean gleich(Knoten k)` einen Methoden-Rumpf, der angibt, ob der linke und der rechte Teilbaum von Knoten k gleich sind. Gleichheit besteht nur, wenn sowohl die Knotenstruktur der Teilbäume als auch die jeweils eingetragenen Bezeichnungen `Bez` übereinstimmen.

c) Programmieren Sie zu dem Methoden-Kopf `void kompaktiere(Knoten k)` einen Methoden-Rumpf, der prüft, ob der linke und der rechte Teilbaum von Knoten k gleich sind. Bei Gleichheit wird einer der beiden Teilbäume entfernt und sowohl `k.links` als auch `k.rechts` zeigen auf den verbleibenden Teilbaum. Die obere Abbildung zeigt ein Beispiel eines Baums, wobei in jeden Knoten der String `Bez` eingetragen ist. Die untere Abbildung zeigt den kompaktierten Baum nach dem Aufruf von `kompaktiere(k)`, wenn k auf den Knoten mit der Bezeichnung `"KL"` zeigt.



d) Programmieren Sie zu dem Methoden-Kopf `void kompaktiere()` einen Methoden-Rumpf, der den Baum an allen Stellen kompaktiert, wo dies möglich ist.

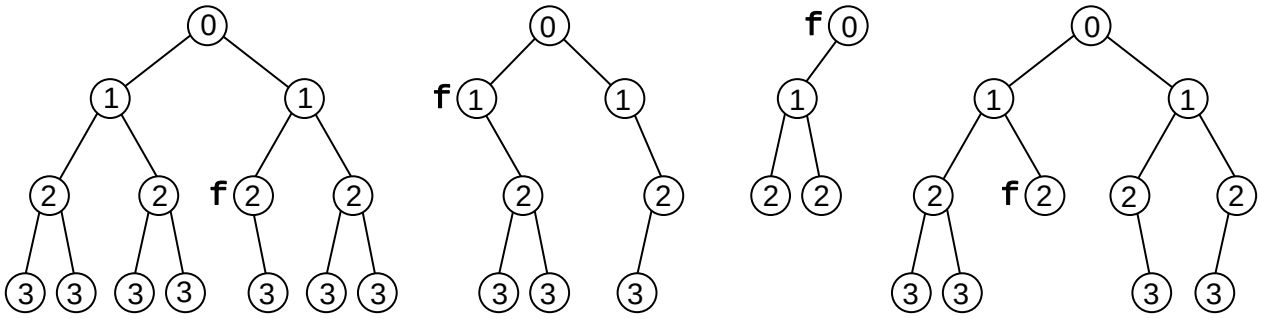
e) Programmieren Sie zu dem Methoden-Kopf `void expandiere()` einen Methoden-Rumpf, der aus einem kompaktierten Baum den ursprünglichen Baum wieder herstellt. Dazu sind alle kompaktierten Teilbäume zu duplizieren. Für den entsprechenden Knoten k verweisen dann `k.links` und `k.rechts` auf den ursprünglich vorhandenen Teilbaum sowie auf das erzeugte Duplikat.

Aufgabe 3.7: Teilweise sortierter Baum

Zur Darstellung eines Binärbaums seien die folgenden Klassen `gegeben`:

```
class Baum   { Knoten Wurzel; }  // Verweis auf die Wurzel.

class Knoten
{ int    Zahl,              // Zahl als Inhalt des Knotens.
         Tiefe;             // Anzahl der Kanten von der Wurzel bis zu diesem Knoten.
  Knoten links, rechts;     // Verweis auf den linken bzw. rechten Nachfolger.

  Knoten(int z, int t) { Zahl = z;  Tiefe = t;  links = rechts = null; }
}
```

Bei den folgenden vier Bäumen ist für jeden Knoten seine `Tiefe` eingetragen.



a) Methode in der Klasse `Baum`: `Knoten frei()`

Programmieren Sie den Rumpf dieser Methode. Sie soll einen Verweis auf einen Knoten liefern, der nur einen oder keinen direkten Nachfolger-Knoten besitzt und eine möglichst geringe Tiefe aufweist. Wenn mehrere Knoten mit gleicher Tiefe diese Bedingung erfüllen, dann ist unter diesen der am weitesten links im Baum stehende Knoten auszuwählen. In den vier Beispiel-Bäumen aus dem Bild oben sind die Knoten mit „**f**“ markiert. Es darf davon ausgegangen werden, dass in allen Knoten ihre Tiefe korrekt eingetragen ist. Wenn der Baum leer ist, soll `null` zurückgegeben werden. Für diese Teilaufgabe empfiehlt sich eine rekursive Lösung.

b) Für das Einfügen in den Baum gelten die folgenden Regeln:

- Eine positive `Zahl` (sowie die Zahl 0) wird gemäß den Regeln eines *Suchbaums* eingefügt (Für eine kleinere Zahl wird ein neuer Knoten weiter links im Baum, für eine größere Zahl weiter rechts im Baum erzeugt).
- Eine negative Zahl wird dagegen in einen neuen Knoten eingetragen, welcher an der Stelle in den Baum eingefügt wird, den die Methode `frei()` anzeigt, siehe Teilaufgabe a). Der neue Knoten wird auf der Seite angefügt, wo der entsprechende Nachfolger-Verweis (`links` bzw. `rechts`) `null` ist. Falls beide Nachfolger-Verweise `null` sind, wird der neue Knoten links angefügt.
- Bei einem noch leeren Baum bildet der neue Knoten die Wurzel.

Zeichnen Sie den Baum, der entsteht, wenn in einen leeren Baum die folgenden Zahlen (in der gegebenen Reihenfolge) nacheinander eingefügt werden: 5, –8, 8, 2, 6, –9, 3, –2, –5.

c) Methode in der Klasse `Baum`: `void fuegeEin(int Zahl)`

Programmieren Sie den Rumpf dieser Methode. Sie soll die `Zahl` in einen neuen Knoten eintragen und diesen entsprechend den Regeln aus Teilaufgabe b) in den Baum einfügen. Nach dem Einfügen soll in den neuen Knoten seine `Tiefe` eingetragen werden.

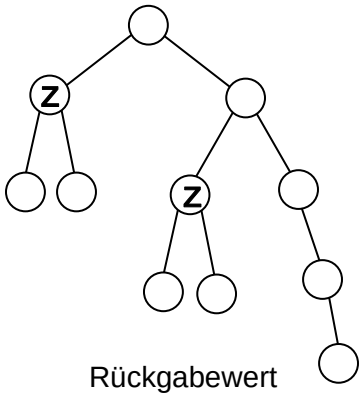
Aufgabe 3.8: Blätter im Baum

Zur Darstellung eines Binärbaums seien die folgenden Klassen `gegeben`:

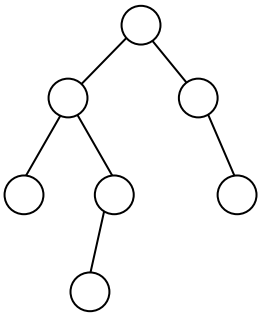
```
class Baum   { Knoten Wurzel; }  // Verweis auf die Wurzel.

class Knoten
{ int    Zahl;             // Zahl als Inhalt des Knotens.
  Knoten links, rechts;    // Verweis auf den linken bzw. rechten Nachfolger.

  Knoten(int z) { Zahl = z;  links = rechts = null; }
}
```

a) Methode in der Klasse `Baum`: `boolean zwei ()`

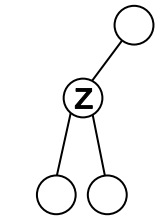
Programmieren Sie den Rumpf dieser Methode. Sie soll genau dann `true` liefern, wenn es in dem Baum einen Knoten gibt, dessen linker und rechter Nachfolger jeweils ein Blatt ist. Das Bild zeigt vier Beispiele, in denen alle Knoten mit „**z**“ markiert sind, denen rechts ud links ein Blatt nachfolgt. Hinweis: Es empfiehlt sich eine rekursive Lösung.



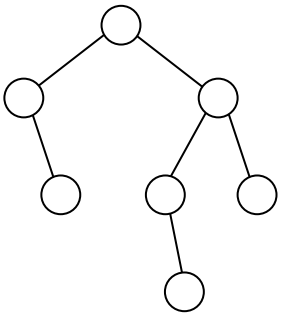
Rückgabewert `true`



Rückgabewert `false`



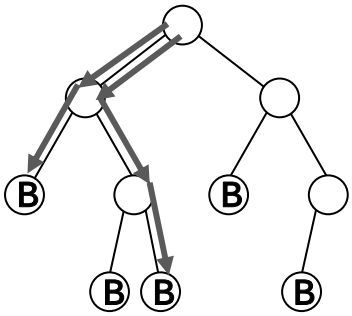
Rückgabewert `true`



Rückgabewert `false`

b) Methode in der Klasse `Baum`: `int Differenz()`

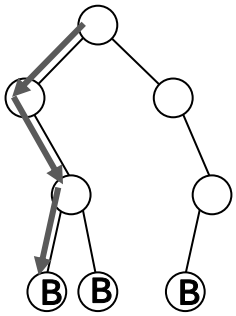
Programmieren Sie den Rumpf dieser Methode. Sie soll die Differenz liefern aus der Anzahl der Kanten, die von der Wurzel zum entferntesten Blatt führen, und der Anzahl der Kanten, die von der Wurzel zum nächstliegenden Blatt führen. Wenn der Baum leer ist, soll die Methode den Wert 0 liefern. Das Bild zeigt vier Beispiele, in denen alle Blätter mit „**B**“ markiert sind. Hinweis: Es empfiehlt sich eine rekursive Lösung.



Von der Wurzel zum entferntesten Blatt: 3 Kanten.

Von der Wurzel zum nächstliegenden Blatt: 2 Kanten.

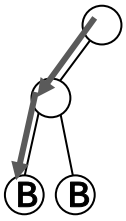
Differenz: 3 – 2 = 1.



Von der Wurzel zum entferntesten Blatt: 3 Kanten.

Von der Wurzel zum nächstliegenden Blatt: 3 Kanten.

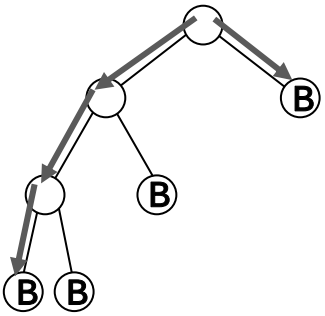
Differenz: 3 – 3 = 0.



Von der Wurzel zum entferntesten Blatt: 2 Kanten.

Von der Wurzel zum nächstliegenden Blatt: 2 Kanten.

Differenz: 2 – 2 = 0.



Von der Wurzel zum entferntesten Blatt: 3 Kanten.

Von der Wurzel zum nächstliegenden Blatt: 1 Kante.

Differenz: 3 – 1 = 2.

Aufgabe 3.9: Baum und Rekursion

Ein Binärbaum sei (wie in der Vorlesung) durch die folgenden Klassen definiert.

```
class Baum   { Knoten Wurzel;          // Verweis auf den Wurzel-Knoten.
               ...
             }
```

```
class Knoten { int Zahl;              // Wert des Knotens.
               Knoten links, rechts;  // Linker und rechter Nachfolger-Knoten im Baum.
               ...
             }
```

Wir gehen davon aus, dass bereits ein korrekt aufgebauter Baum existiert, in dessen Knoten die Variable `Zahl` jeweils mit einem Wert belegt ist.
Von einer Sortierung der Zahlen kann nicht ausgegangen werden.

a) Methode in der Klasse `Baum`: **`Knoten maxi()`**

Programmieren Sie den Rumpf dieser Methode, die einen Verweis auf den Knoten des Baums liefern soll, dessen `Zahl` die größte Zahl im Baum ist. Wenn es mehrere Knoten gibt, die die größte Zahl enthalten, dann soll ein Verweis auf irgendeinen dieser Knoten geliefert werden. Wenn der Baum leer ist, soll die Methode `null` liefern.

`maxi()` soll eine rekursive Methode `Knoten` **`maxi(Knoten k)`** aufrufen, die Sie ebenfalls programmieren sollen. `maxi(k)` liefert die größte Zahl in dem durch `k` gegebenen Teilbaum.

b) Methode in der Klasse `Baum`: **`Knoten mini()`**

Markieren Sie in Ihrer Lösung zu Teilaufgabe a) die Programmstellen mit ⊗, die man verändern müsste, um die Methoden `mini()` sowie `mini(Knoten k)` zu erhalten, die einen Verweis auf einen Knoten mit der kleinsten Zahl im Baum bzw. Teilbaum liefern.

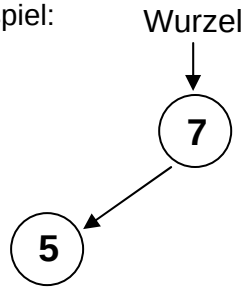
Sie müssen diese Methoden nicht programmieren.

c) Gegeben seien die folgenden Methoden in der Klasse `Baum`:

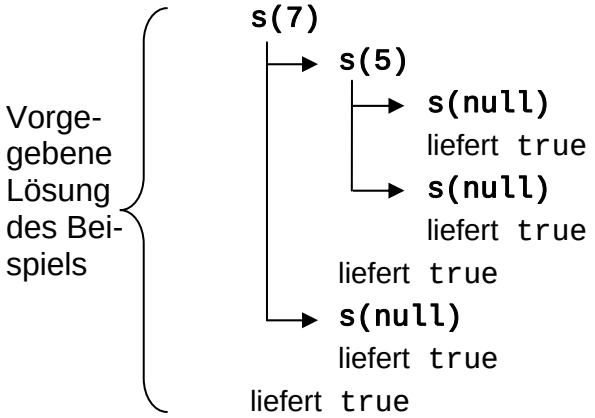
```
boolean s()
{ return s(Wurzel);
}

boolean s(Knoten k)
{ if (k != null)
    return     s(k.links)  &&  s(k.rechts)
           &&  (k.links == null  ||  maxi(k.links).Zahl <= k.Zahl)
           &&  (k.rechts == null  ||  mini(k.rechts).Zahl >= k.Zahl);
  else
    return  true;
}
```

Geben Sie für die beiden folgenden Bäume c1) und c2) jeweils alle Aufrufe der rekursiven Methode `s(Knoten k)` mit dem aktuellen Parameter und dem booleschen Rückgabewert an. Bezeichnen Sie dabei Verweise auf Knoten vereinfachend mit der `Zahl` des betreffenden Knotens bzw. mit `null`. Es empfiehlt sich, die Rekursionstiefe durch entsprechende Einrückung der Aufrufe auszudrücken (wie in der Vorlesung und in den eingebundenen Übungen und wie in dem folgenden Beispiel dargestellt). Die Aufrufe von `maxi(...)` und `mini (...)` müssen nicht dargestellt werden.

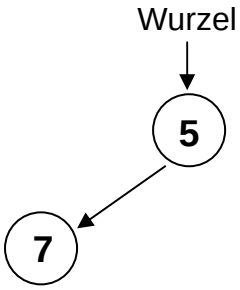
Beispiel:



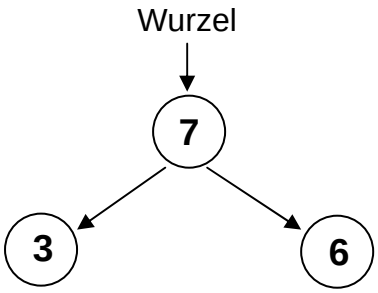
In die Knoten ist ihre `Zahl` eingetragen.



c1)



c2)



Zur Lösung der folgenden Teilaufgaben genügen jeweils ca. 1 bis 3 Sätze.

d) Beschreiben Sie die Funktion `s()` verbal: Unter welcher Bedingung liefert sie `true` ?

e) Ist die Rekursion der Funktion `s()` linear ? Ist sie schlicht ?

f) Erläutern Sie, warum die Funktion `s()` in großen Bäumen zu einer sehr großen Anzahl von Aufrufen der Funktionen `maxi(...)` und `mini(...)` führt, die unnötig hoch ist.

g) Geben Sie einen Ansatz an, wie sich die in Teilaufgabe f) beschriebene unnötig hohe Anzahl von Aufrufen der Funktionen `maxi(...)` und `mini(...)` reduzieren ließe, ohne die Funktion von `s()` zu verändern (Die Funktion `s(Knoten k)` spielt dabei keine Rolle).
Es genügt eine verbale Beschreibung der Idee. Sie muss nicht programmiert werden.

Aufgabe 3.10: Bäume

Ein Binärbaum sei (wie in der Vorlesung) durch die folgenden Klassen definiert.

```
class Baum   { Knoten Wurzel;          // Verweis auf den Wurzel-Knoten.
               ...
             }
class Knoten { int Zahl;               // Wert des Knotens.
               Knoten links, rechts;   // Linker und rechter Nachfolger-Knoten im Baum.
               ...
             }
```

Zur Lösung der Teilaufgaben a) bis d) genügen jeweils ca. 1 bis 3 Sätze bzw. Begriffe.

a) Durch welche Art des Durchlaufs durch einen Suchbaum können die eingetragenen Zahlen aufsteigend sortiert ausgegeben werden ?

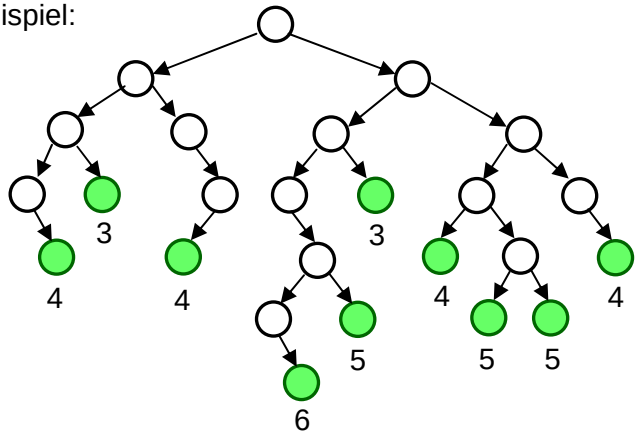
b) Was versteht man unter der Höhe eines Baums ?

c) Was versteht man unter einem vollständigen Baum ?

d) Methode in der Klasse `Baum`: **`int minBlattabstand()`**

Programmieren Sie den Rumpf dieser Methode, die den kürzesten Abstand zwischen der Wurzel und einem Blatt liefert. Dies bedeuet, dass die Anzahl der Knoten angegeben werden soll, die auf dem Weg von der Wurzel bis zum nächstliegenden Blatt liegen (einschließlich der Wurzel, aber ohne das Blatt selbst).

Beispiel:



Blätter sind grün.

Unter jedem Blatt ist sein Abstand zur Wurzel eingetragen.

Für diesen Baum muss die Methode den Wert 3 liefern.
(Es gibt zwei Blätter mit dem Abstand 3)

e) Methode in der Klasse `Baum`: **`int maxBlattabstand()`**

Markieren Sie in Ihrer Lösung zu Teilaufgabe e) die Programmstellen mit ⊗, die man verändern müsste, um die Methode `maxBlattabstand()` zu erhalten, die den größten Abstand zwischen der Wurzel und einem Blatt liefert. Sie müssen diese Methoden nicht programmieren.

f) Angenommen für einen bestimmten Baum liefern die beiden Methoden `minBlattabstand()` und `maxBlattabstand()` den gleichen Wert. Kann daraus geschlossen werden, dass der Baum vollständig ist ?

Wenn ja, geben Sie eine Begründung dafür an.
Wenn nein, skizzieren Sie einen nicht vollständigen Baum, für den
`minBlattabstand() == maxBlattabstand()` gilt.

g) Methode in der Klasse `Baum`: **`int Blattanzahl()`**

Programmieren Sie den Rumpf dieser Methode, welche die Anzahl der Blätter eines Baums angibt. Für den in Teilaufgabe d) dargestellten Baum liefert sie den Wert 10.

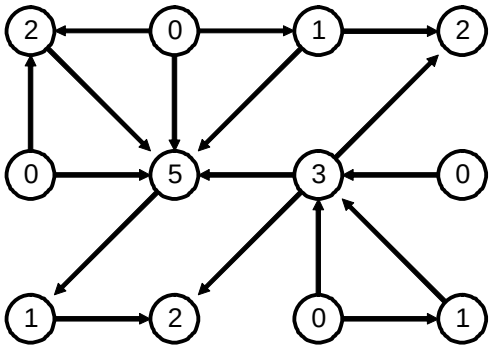
4 Graphen

Aufgabe 4.1: Kanten in Graphen

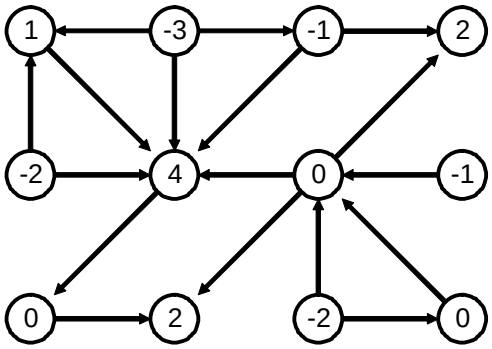
Ein gerichteter Graph wird durch die aus der Vorlesung bekannten Klassen `Graph`, `Knoten` und `Kante` repräsentiert. Hier wurde zur Klasse Knoten die `int`-Variable `Anz` hinzugefügt.

```
class Graph
{ Knoten Kopf, Fuss;
  ... // Methoden von Teilaufg. a), b) hier einfügen.
}

class Knoten
{ String Bez;  Knoten Nf;  Kante Kopf, Fuss;
  int Anz;
  ...
}

class Kante
{ String Bez;  Kante Nf;  Knoten Kante;
  ...
}
```

a) Programmieren Sie in der Klasse `Graph` eine Methode `h`, die für einen gegebenen Graphen in jeden Knoten die Anzahl der zu ihm hinführenden Kanten in die Variable `Anz` einträgt. Ansonsten soll der Graph unverändert bestehen bleiben. Für den folgenden Beispiel-Graphen sind die Werte von `Anz` in jeden Knoten eingetragen.



b) Programmieren Sie in der Klasse `Graph` eine Methode `d`, die für einen gegebenen Graphen in jeden Knoten die Differenz aus der Anzahl der zu ihm hinführenden Kanten und der Anzahl der von ihm wegführenden Kanten in die Variable `Anz` einträgt. Ansonsten soll der Graph unverändert bestehen bleiben. Die nachfolgende Abbildung zeigt die in Anz einzutragende Differenz für einen Beispiel-Graphen. Hinweis: Die Methode `d` kann ähnlich aufgebaut werden wie die Methode `h` aus Teilaufgabe a).



Aufgabe 4.2: Doppelt verbundene Knoten

Ein gerichteter Graph wird durch die aus der Vorlesung bekannten Klassen `Graph`, `Knoten` und `Kante` repräsentiert.

```
class Graph
{ Knoten Kopf, Fuss;  // Liste der Knoten des Graphen.
  ...  // Teilaufg. a und b: Methoden hier einfügen.
}

class Knoten
{ String Bez;  Knoten Nf;  // Nf in der Liste der Knoten.
  Kante Kopf, Fuss;        // Liste der Kanten eines Knotens.
  ...
}

class Kante
{ String Bez;  Kante Nf;  // Nf in der Liste der Kanten.
  Knoten Kante;           // Knoten, zu dem die Kante hinführt.
  ...
}
```

Wie üblich nehmen wir an, dass es in einem Graphen keine Kante gibt, die von einem Knoten zu sich selbst führt. Ebenso nehmen wir an, dass von einem Knoten K_i nicht mehrere Kanten zu einem Knoten K_j führen (keine parallelen Kanten).

a) In einem Graphen wird ein Knoten K_i als Senke bezeichnet, wenn keine Kanten von K_i wegführen (Anmerkung: Hinführende Kanten sind erlaubt). Programmieren Sie in der Klasse `Graph` eine Methode `s`, welche die Anzahl der Senken im Graphen liefert.

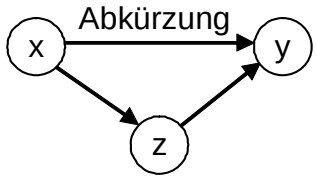
b) In einem Graphen wird ein Knotenpaar (K_i, K_j) als doppelt verbunden bezeichnet, wenn es sowohl eine Kante von K_i nach K_j als auch eine Kante von K_j nach K_i gibt. Programmieren Sie in der Klasse `Graph` eine Methode `d`, welche die Anzahl der Knotenpaare liefert, die doppelt verbunden sind.

Aufgabe 4.3: Abkürzungen in Graphen

Ein gerichteter Graph wird durch die aus der Vorlesung bekannten Klassen `Graph`, `Knoten` und `Kante` repräsentiert:

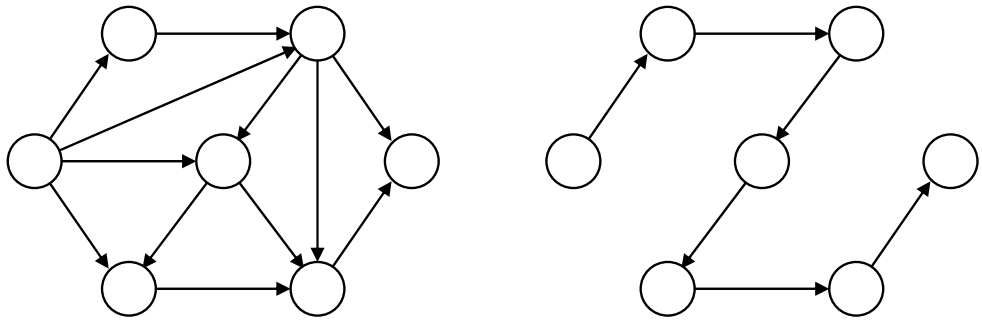
```
class Graph
{ Knoten Kopf, Fuss;  // Liste der Knoten des Graphen.
  ...  // Methoden zur Bearbeitung des Graphen.
}

class Knoten
{ String Bez;  Knoten Nf;  // Nachfolger-Knoten.
  Kante Kopf, Fuss;        // vom Knoten ausgehende Kanten.
}

class Kante
{ String Bez;  Kante Nf;  // Nachfolger-Kante.
  Knoten Kante;           // Knoten, zu dem die Kante hinführt.
}
```

Hier wird definiert: Eine Kante von einem Knoten x zu einem Knoten y heißt „Abkürzung“, wenn es einen Knoten z und Kanten von x nach z sowie von z nach y gibt.



a) Programmieren Sie zu dem Methoden-Kopf
`boolean Abk(Knoten k, Kante e)` einen Methoden-Rumpf, der angibt, ob die von Knoten k ausgehende Kante e eine Abkürzung ist. Die Methode boolean `Abk(Knoten k, Kante e)` sei in der Klasse `Graph` angeordnet.

b) Programmieren Sie zu dem Methoden-Kopf `void loescheAbk()` einen Methoden-Rumpf, der aus einem beliebigen Graphen alle Abkürzungen löscht. Die Methode `void loescheAbk()` sei in der Klasse `Graph` angeordnet. Ein Beispiel zu dieser Teilaufgabe: Links der gegebene Graph, rechts der Graph, nachdem durch den Aufruf `loescheAbk()` alle Abkürzungen entfernt worden sind.



Hinweis: Je nach dem von Ihnen eingeschlagenen Lösungsweg kann es nötig sein, dass Sie in die Klasse `Kante` eine zusätzliche Variable einfügen müssen.

Aufgabe 4.4: Durchgangsknoten

Ein gerichteter Graph wird durch die Klassen `Graph`, `Knoten` und `Kante` repräsentiert:

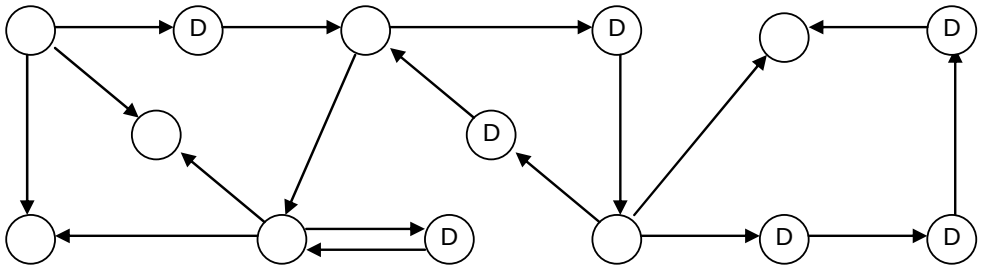
```
class Graph
{ Knoten Kopf, Fuss;  // Liste der Knoten des Graphen.
  ...  // Hier sind alle Methoden der folgenden Teilaufgaben angesiedelt.
}

class Knoten
{ String Bez;  Knoten Nf;  // Nachfolger-Knoten.
  Kante Kopf, Fuss;        // vom Knoten ausgehende Kanten.
}

class Kante
{ String Bez;  Kante Nf;  // Nachfolger-Kante.
  Knoten Kante;           // Knoten, zu dem die Kante hinführt.
}
```

a) Programmieren Sie zu dem Methoden-Kopf
`Kante exKante(Knoten a, Knoten b)`
einen Methoden-Rumpf, der einen Verweis auf die Kante von Knoten a zu Knoten b liefert. Falls keine Kante von Knoten a zu Knoten b führt, soll die Methode den leeren Verweis `null` liefern.

b) Programmieren Sie zu dem Methoden-Kopf `void loescheKn(Knoten k)` einen Methoden-Rumpf, der den Knoten k löscht. Es darf davon ausgegangen werden, dass es keine zum Knoten k hinführende Kante gibt.

Hier wird definiert: Ein Knoten x, zu dem genau eine Kante hinführt und von dem genau eine Kante wegführt, heißt Durchgangsknoten.

In dem folgenden Beispiel sind alle Durchgangsknoten mit „D“ markiert.

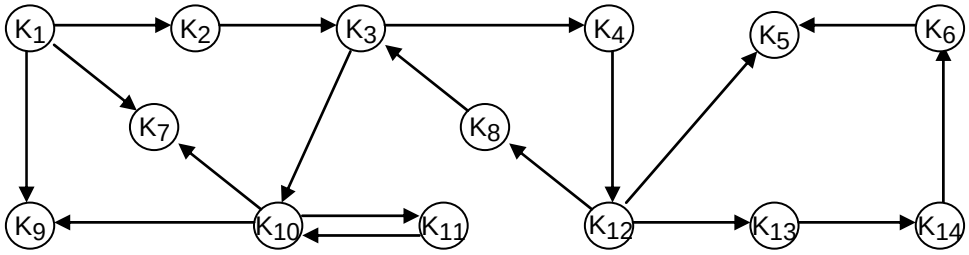


c) Programmieren Sie zu dem Methoden-Kopf `boolean Du(Knoten k)` einen Methoden-Rumpf, der angibt, ob Knoten k ein Durchgangsknoten ist.

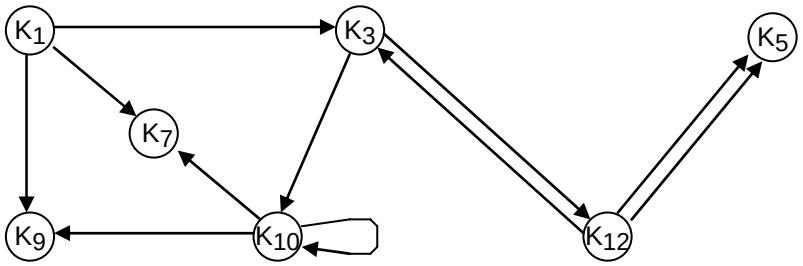
d) Programmieren Sie zu dem Methoden-Kopf `void ueberbr()` einen Methoden-Rumpf, der aus einem beliebigen Graphen alle Durchgangsknoten durch eine Kante überbrückt und dann löscht. Dies bedeutet: Wenn k ein Durchgangsknoten ist und es Kanten gibt von Knoten x zu Knoten k und von Knoten k zu Knoten y, dann soll eine Kante von Knoten x zu Knoten y eingefügt werden. Außerdem sind Knoten k und die mit ihm verbundenen Kanten zu entfernen. Sonderfälle:

- Auch wenn der Graph bereits eine Kante von Knoten x zu Knoten y enthält, ist eine zusätzliche Kante von x nach y zu erzeugen (in dem Beispiel unten: siehe zusätzliche Kante von Knoten K_{12} zu Knoten K_5 zur Überbrückung der Knoten K_{13}, K_{15} und K_6).
- Auch wenn die Knoten x und y gleich sind, ist eine Kante zu erzeugen, die von Knoten x zu sich selbst führt (in dem Beispiel unten: Kante von Knoten K_{10} zu sich selbst zur Überbrückung des Knotens K_{11}).

Als Beispiel zu dieser Teilaufgabe ein gegebener Graph:



Daraus entsteht durch den Aufruf von `ueberbr()` der folgende Graph:



Aufgabe 4.5: Graphen und Arrays

Ein gerichteter Graph sei (ähnlich wie in der Vorlesung) durch die folgenden Klassen definiert. Beachten Sie die zusätzlichen int-Variablen `Anzahl`, `Index` und `Zahl`.

```
class Graph  { Knoten Kopf, Fuss; // Liste der Knoten.
               int    Anzahl;     // Anzahl der Knoten im Graphen.
               ...
             }
class Knoten { String Bez;        // Bezeichnung des Knotens.
               int    Index;      // Index des Knotens = Position in der Liste der Knoten.
               Knoten Nf;         // Verweis auf den Nachfolger in der Liste der Knoten.
               Kante  Kopf, Fuss; // Liste der Kanten, die von dem Knoten ausgehen.
               ...
             }
class Kante  { int    Zahl;       // Wert der Kante.
               Kante  Nf;         // Verweis auf den Nachfolger in der Liste der Kanten.
               Knoten Kante;      // Verweis auf den Zielknoten der Kante („Pfeilspitze“).
               ...
             }
```

Wir gehen davon aus, dass bereits ein korrekt aufgebauter Graph existiert, in dessen Kanten die Variable `Zahl` jeweils mit einer natürlichen Zahl belegt ist.

a) Methode in der Klasse `Graph`: **`void indiziere()`**

Programmieren Sie den Rumpf dieser Methode, die den Knoten entsprechend ihrer Reihenfolge in der Liste der Knoten einen fortlaufenden Index zuordnen soll (der erste Knoten in der Liste: 0, der zweite Knoten: 1, der dritte Knoten: 2, usw.). Der Index ist in die Variable `Index` einzutragen. Außerdem ist die Anzahl der Knoten in die Variable `Anzahl` des Graphen einzutragen.

b) Methode in der Klasse `Graph`: **`int[][] Matrix()`**

Programmieren Sie den Rumpf dieser Methode, die den Graphen durch eine zweidimensionale int-Matrix repräsentieren soll. Die Zeilen- und die Spaltenanzahl der Matrix sind gleich der Knotenanzahl des Graphen. Wenn eine Kante von einem Knoten mit Index i zu dem Knoten mit Index j führt, wird in das Element `[i][j]` der Matrix die `Zahl` dieser Kante eingetragen. Andernfalls wird eine 0 eingetragen.

Hinweis für die folgenden Teilaufgaben: Sie können (aber müssen nicht) die Methode `Matrix()` aufrufen und dann mit der Matrix-Darstellung des Graphen arbeiten.

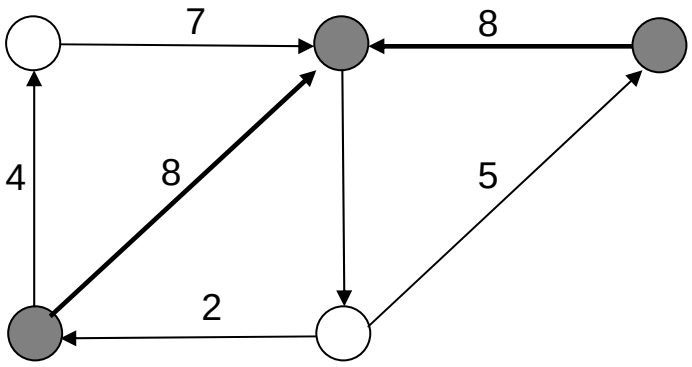
c) Methode in der Klasse `Graph`: **`int Summe(Knoten k)`**

Programmieren Sie den Rumpf dieser Methode, die für den durch `k` gegebenen Knoten die Summe der Zahlen in allen Kanten (jeweils Variable `Zahl`) liefert, die von k ausgehen oder die in k enden (d.h. „Pfeilanfang“ oder „Pfeilspitze“ der betreffenden Kanten ist mit dem Knoten k verbunden).

d) Methode in der Klasse `Graph`: **`int AnzMax()`**

Programmieren Sie den Rumpf dieser Methode, welche die Anzahl der Knoten liefert, von denen eine Kante ausgeht oder in die eine Kante mündet, deren `Zahl` maximal ist (d.h. „Pfeilanfang“ oder „Pfeilspitze“ einer solchen Kante ist mit dem Knoten verbunden). Die `Zahl` einer Kante ist maximal, wenn die `Zahl` keiner anderen Kante im Graphen größer ist.

Hinweis: Es kann eine oder mehrere Kanten im Graphen geben, deren `Zahl` maximal ist. Wenn es mehrere solche Kanten gibt, dann sind deren Zahlen gleich. Der folgende Beispielgraph zeigt zwei Kanten mit maximalen Zahlen (jeweils 8). Sie sind mit drei Knoten verbunden (grau hervorgehoben), so dass `AnzMax()` den Wert 3 liefern muss.



Aufgabe 4.6: Graphen

Ein gerichteter Graph sei (ähnlich wie in der Vorlesung) durch die folgenden Klassen definiert.
Knoten und Kanten werden jeweils in der Farbe dargestellt, die durch die Variable `Farbe` gegeben ist.
Dabei bedeuten: `'r'` = rot, `'b'` = blau, `'s'` = schwarz. Weitere Farben gibt es nicht.
Die Variablen `n` und `f` in der Klasse `Knoten` werden nur in Teilaufgabe c) benötigt.

```
class Graph  { Knoten Kopf, Fuss; // Liste der Knoten.
               ...
             }

class Knoten { char   Farbe;      // Farbe des Knotens.
               int    n;         // Anzahl der Farben hinführender Kanten.
               char   f;         // Farbe einer beliebigen hinführenden Kante.
```

```
    Knoten Nf;          // Verweis auf den Nachfolger in der Liste der Knoten.
    Kante  Kopf, Fuss; // Liste der Kanten, die von dem Knoten ausgehen.
    ...
  }
class Kante  { char   Farbe;       // Farbe der Kante.
    Kante  Nf;          // Verweis auf den Nachfolger in der Liste der Kanten.
    Knoten Kante;       // Verweis auf den Zielknoten der Kante („Pfeilspitze“).
    ...
  }
```

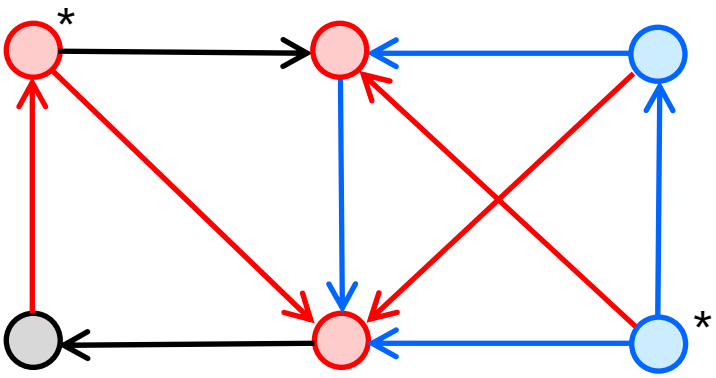
Wir gehen davon aus, dass bereits ein korrekt aufgebauter Graph existiert.

a) Methode in der Klasse `Graph`: **`int AnzRot()`**

Programmieren Sie den Rumpf dieser Methode, welche die Anzahl der roten Knoten angibt (`Farbe == 'r'`).

b) Methode in der Klasse `Graph`: **`int Anz_gleichfarbig()`**

Programmieren Sie den Rumpf dieser Methode, welche die Anzahl der Knoten angibt, von denen eine Kante ausgeht, die die gleiche Farbe wie der Knoten aufweist und die zu einem Knoten führt, der ebenfalls die gleiche Farbe aufweist. Bespiel: Ein roter Knoten ist zu zählen, wenn von ihm eine rote Kante zu einem anderen roten Knoten führt.



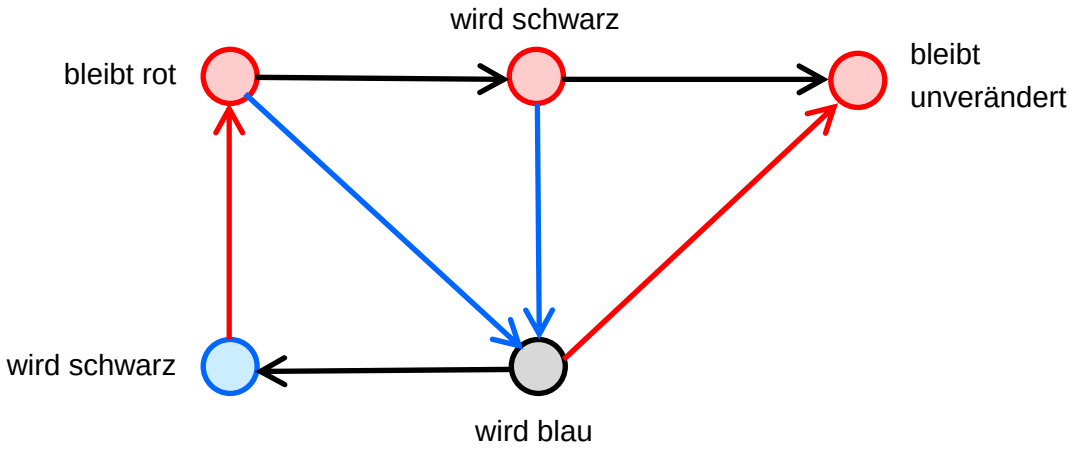
Für diesen Graphen muss die Methode den Wert 2 liefern (mit * gekennzeichnete Knoten)

c) Methode in der Klasse `Graph`: **`void faerbe_hinfuehrend()`**

Programmieren Sie den Rumpf dieser Methode, die für alle Knoten prüft, ob alle zu ihm hinführenden Kanten die gleiche Farbe haben. Ist dies der Fall, so nimmt der Knoten die Farbe der zu ihm hinführenden Kanten an. Wenn zu einem Knoten keine Kante hinführt, bleibt seine Farbe unverändert.

Es wird empfohlen (aber nicht zwingend verlangt) die Variablen `n` und `f` in der Klasse `Knoten` zu benutzen. Dabei soll `n` die Anzahl der Farben hinführender Kanten und `f` die Farbe einer hinführenden Kante ausdrücken.

Hinweis: Da nur entschieden werden muss, ob alle hinführenden Kanten gleichfarbig sind, muss n nicht unterscheiden, ob es zwei oder drei Farben hinführender Kanten gibt („2“ würde genügen um „mehr als 1 Farbe“ auszudrücken).



d) Angenommen, ein Graph besteht aus x Knoten, wobei von jedem Knoten genau 2 Kanten ausgehen. Wir nehmen außerdem an, dass der Durchlauf durch jeder Programmschleife genau eine Zeiteinheit benötigt sonst kein weiterer Zeitbedarf für die Programmausführung besteht. Wie viele Zeiteinheiten benötigt die Ausführung der Methode, die Sie in Teilaufgabe c) erstellt haben ?

5 Klassenhierarchien und Verweisstrukturen

Aufgabe 5.1: Erweiterung von Klassen

Entwerfen Sie Klassen und eine Klassenhierarchie mit den folgenden relevanten Daten. Gegeben seien diee Begriffe:

`PKW` (Personenkraftwagen), `Zug, Flugzeug, LKW` (Lastkraftwagen), `Transportmittel, Kraftwagen.`

Ferner seien folgende Eigenschaften gegeben:

`Wagonanzahl, Bezeichnung, Fluggesellschaft, Kennzeichen, Gewicht, Ladekapazität.`

a) Ordnen Sie die Eigenschaften den Begriffen sinnvoll zu und skizzieren Sie eine Vererbungshierarchie zwischen den Klassen.

b) Implementieren Sie die in Teilaufgabe a) entwickelte Hierarchie mit Hilfe von Klassenerweiterungen in Java. Fügen Sie den Klassen Ausgabemethoden hinzu, die alle Eigenschaften der Klasse ausgeben. Zur Ausgabe von allgemeinen Eigenschaften sollen die Oberklassen verwendet werden.

c) Schreiben Sie jeweils einen Konstruktor für die Klassen `Transportmittel` und `Zug`. Die Konstruktoren sollen jeweils alle Eigenschaften der Klasse als Argumente entgegennehmen und diese in die entsprechenden Variablen der Klassen eintragen. Der Konstruktor von `Zug` soll den Konstruktor von `Transportmittel` verwenden, um die Transportmitteleigenschaften anzulegen.

d) Schreiben Sie ein Interface `Fahrzeug`, welches die Methode `Ausgabe()` enthält. Wie muss die Deklaration bei der Klasse `Transportmittel` aussehen, wenn diese Klasse das Interface implementieren soll?

Aufgabe 5.2: Waren und Preise

Wie lautet die Ausgabe des folgenden Programms? Es wird empfohlen, aber nicht verlangt, zur Beantwortung dieser Frage eine Skizze anzufertigen, welche die erzeugte Objektstruktur darstellt.

```
interface Zubehoer  { Ware Hauptware(); }

interface Verpackung  { int Kosten(int Einpackdauer); }

abstract class Ware
{ String Art;  int Preis;  // Preis in Cent.

  int Preis()  { return Preis * (100 - Rabatt()) / 100; }

  Ware(String Art, int Preis)
  { this.Art = Art;  this.Preis = Preis;
    System.out.println("Ware " + Art + " " + Preis + " Cent");
  }

  abstract int Rabatt();  // Rabatt in Prozent.
}
```

```
class Birne extends Ware implements Zubehoer
{ Ware Haupt;   int Leistung;  // Leistung in Watt.

  Birne(int P, Ware H)
  { super("Birne", 2 * P);
    Leistung = P;  Haupt = H;
    System.out.println("Birne " + P + " Watt, " + H.Art);
  }

  int Rabatt()  { return 0; }

  public Ware Hauptware()  { return Haupt; }
}

class Kiste extends Ware implements Verpackung, Zubehoer
{ Ware Haupt;

  Kiste(Ware Haupt)
  { super("Kiste", 400);  this.Haupt = Haupt;
    System.out.println("Kiste, " + Haupt.Art);
  }

  public int Kosten(int Dauer) { return 40 * Dauer; }
                    // Dauer in Minuten.

  public Ware Hauptware()  { return Haupt; }

  int Rabatt()  { return 10; }

  int Preis()  { return 10 + Kosten(5); }
}

class Lampe extends Ware
{ Ware Verpackung;  Ware[] Birnen = new Ware[2];

  Lampe()
  { super("Lampe", 8000);
    Verpackung = new Kiste(this);
    for (int i = 0; i < 2; i++)
      Birnen[i] = new Birne(100 + 50*i, this);
  }

  int Rabatt()  { return 20; }
}

public class Haupt
{ public static void main (String[] leer)
  { Lampe Verkauf = new Lampe();
    System.out.println("A " + Verkauf.Birnen[1].Preis());
    System.out.println("B " + Verkauf.Verpackung.Preis());
    System.out.println("C " +
        ((Birne)(Verkauf.Birnen[1])).Hauptware().Preis());
  }
}
```

Aufgabe 5.3: Ober- und Unterklasse

Wie lautet die Ausgabe des folgenden Programms? Es wird empfohlen, aber nicht verlangt, eine Skizze anzufertigen, welche die jeweils aktuelle Belegung der Variablen darstellt.

```
class p
{ int x;

  p()
  { x = 0;  System.out.println("p " + x);
  }

  int m (int i)
  { return 2 * i;
  }
}

class q extends p
{ int x;

  q()
  { x = 1;  System.out.println("q " + x);
  }

  int m (int k)
  { return k - 1;
  }
}

public class Haupt
{ public static void main (String[] leer)
  { int[] a = {3, 1, 2, 8};
    p r;  q s = new q();
    r = (p) s;
    for (int i = 1;  i < 4;  i++)
    { if (a[i] - a[i - 1] > r.x)   r.x = a[i] - a[i - 1];
      a[i] = a[i - 1] + a[i];
      if (a[i] - a[i - 1] > s.x)   s.x = a[i] - a[i - 1];
      r.x = r.m(r.x);
      s.x = s.m(s.x);
      for (int j = 0;  j < 4;  j++)
        System.out.print(a[j] + " ");
      System.out.println(“[” + i + "] " + r.x + ", " + s.x);
    }
  }
}
```

Aufgabe 5.4: Veweise auf Vater, Mutter und Kinder

Gegeben sei die folgende Klasse `Person`. Ein Objekt der Klasse stellt eine Person dar und drückt durch Verweise ihre verwandschaftlichen Beziehungen zu Eltern und Kindern aus.

```
class Person
{ String Name;           // Name der Person.
  Person Vater, Mutter;  // Verweise auf Person-Objekte, die
                         // Eltern darstellen.
  Person [] Kinder;      // Verweise auf Person-Objekte, die
                         // Kinder darstellen.

  void trage_Kind_ein (Person Kind)
  { Person [] k = this.Kinder;
    if (k == null)  Kinder = new Person [1];
    else            Kinder = new Person [k.length + 1];
    for (int i = 0;  i < Kinder.length - 1;  i++)  Kinder[i] = k[i];
    Kinder [Kinder.length - 1] = Kind;
  }
}
```

Bei den Teilaufgaben a) und b) ist zu beachten, dass `Vater` und `Mutter` leere Verweise sein können, weil nicht die gesamte Menschheit, sondern nur eine begrenzte Personengruppe durch Objekte dargestellt wird.

a) Schreiben Sie den Rumpf eines Konstruktors
`Person (String Name, Person Vater, Person Mutter)`,
der in das erzeugte `Person`-Objekt den Namen sowie die als Parameter angegebenen Verweise auf Vater und Mutter einträgt. Der Konstruktor soll außerdem in den betreffenden Vater- und Mutter-Objekten (soweit durch Parameter angegeben) je einen Verweis in `Kinder` eintragen, der auf das erzeugte `Person`-Objekt verweist. Hinweis: Dabei kann die Methode `trage_Kind_ein` verwendet werden.

b) Silke und Sven sind ein Paar, das zwei gemeinsame Kinder Tina und Theo hat. Schreiben Sie ein Programmstück, das die Familie durch vier Objekte der Klasse `Person` und entsprechende Verweise zwischen diesen darstellt.

Die beiden folgenden Teilaufgaben c) und d) gehen nicht von einer bestimmten Familie aus, sondern von beliebig vielen `Person`-Objekten und Verweisen zwischen diesen aus. Außerdem darf davon ausgegangen werden, dass bei keinem der verwendeten Objekte für `Vater`, `Mutter` oder Elemente von `Kinder` leere Verweise eingetragen sind.

c) Schreiben Sie den Rumpf einer Methode `Person [] Oma (Person p)`, die für eine Person `p` Verweise auf ihre Großmütter liefert.

d) Schreiben Sie den Rumpf einer Methode `int Anzahl (Person p)`, die für eine Person `p` die Anzahl ihrer Halbgeschwister liefert (d.h. Geschwister, mit denen sie entweder nur den Vater oder nur die Mutter, aber nicht beide Elternteile gemeinsam haben).

Aufgabe 5.5: Benzinverbrauch

In einem Projekt soll der Benzinverbrauch von verschiedenen Fahrzeugen untersucht werden. Dabei ist jedes Fahrzeug durch die folgenden Eigenschaften charakterisiert:

E_1: den Typ des Fahrzeugs, z.B. “PKW-Kombi Fabrikat X”

E_2: die Höchstgeschwindigkeit des Fahrzeugs (ganzzahlig in km/h), z.B. 172

E_3: die Bezinverbrauchsfunktion b(v), die für jede Geschwindigkeit v (ganzzahlig in km/h) den Bezinverbauch (in Liter / 100 km als Gleitkommazahl) angibt, z.B.

$$b(v) = 6 + \frac{(v - 91)^2}{1920} + 0{,}024 \cdot v$$

Ein Sportwagen könnte dagegen folgende Eigenschaften aufweisen (als Beispiel):

- Typ “Sportwagen Fabrikat Y“,
- Höchstgeschwindigkeit 210,
- Bezinverbrauchsfunktion
  $b(v) = 7{,}8 \cdot 10^{-6} \cdot (v - 111)^2 \cdot (v + 90) + 0{,}09 \cdot v - 2{,}6 \cdot 10^{-4} \cdot v^2$

Untersuchungsgegenstände des Projekts sind:

U_1: Bestimmung der Geschwindigkeit mit dem geringsten Bezinverbrauch
(Wenn mehrere Geschwindigkeiten den gleichen geringsten Benzinverbrauch aufweisen, soll die größte dieser Geschwindigkeiten bestimmt werden).

U_2: Bestimmung der größten Geschwindigkeit, deren Benzinverbauch nicht mehr als 10% über dem geringsten Benzinverbauch des jeweiligen Fahrzeugs liegt.

Alle Programmteile zur Behandlung der Untersuchungsgegenstände U_1 und U_2 sind in einer Oberklasse `Fahrzeug` anzusiedeln. Die Programmteile zur Realisierung der Fahrzeugeigenschaften E_1, E_2 und E_3 sollen dagegen in Unterklassen davon angesiedelt sein, die spezifisch für das jeweilige Fahrzeug sind, z.B. `PKWKombiX` oder `SportwagenY`.

Die Anzahl der untersuchten Fahrzeuge kann beliebig groß sein. Für jedes Fahrzeug wird ein entsprechendes Objekt erzeugt.

Die Methoden, die U_1 und U_2 behandeln, sollen als Rückgabewert die entsprechenden Ergebnisse liefern.

a) Programmieren Sie die Oberklasse `Fahrzeug` mit den Methoden, die für die Untersuchungsgegenstände U_1 und U_2 erforderlich sind.

b) Programmieren Sie die Unterklasse `PKWKombiX` für ein Fahrzeug mit den folgenden Eigenschaften:
   - Typ “PKW-Kombi Fabrikat X“,
   - Höchstgeschwindigkeit 172,
   - Bezinverbrauchsfunktion $b(v) = 6 + \frac{(v - 91)^2}{1920} + 0{,}024 \cdot v$

Aufgabe 5.6: Himmelskörper

Welche Ausgabe erzeugt das folgende Programm? Es wird empfohlen, aber nicht verlangt, zur Beantwortung dieser Frage eine Skizze anzufertigen, welche die erzeugte Objektstruktur darstellt.

```
class Himmelskoerper
{ String Name, Sicht;

  Himmelskoerper(String Name)
  { this.Name = Name;  Sicht = "unbekannt";
    System.out.println("Himmelskörper: " + Name);
```

```
  }
  String sichtbar(boolean Tag)
  { if (Tag) return "unsichtbar";
    else     return Sicht;
  }
}

class Planet extends Himmelskoerper
{ String Sicht = "Punkt";  Mond[] Monde;

  Planet(String n, int a, Mond m)
  { super(n);  Monde = new Mond[a];  Monde[0] = m;
    System.out.println("Planet: " + Sicht);
  }
  String sichtbar(boolean Tag)
  { if (Tag) return Sicht + " (schwach)";
    else     return "sichtbar";
  }
}

class Mond extends Himmelskoerper
{ String Sicht = "Scheibe";  Planet zentral;

  Mond(String n, Planet p)
  { super(n);  zentral = p;
    System.out.println("Mond: " + Sicht);
  }
  String sichtbar(boolean Tag)
  { if (super.Sicht.equals("Scheibe")) return "sehr gut";
    else                               return "schwierig";
  }
}

public class Haupt
{ public static void main(String[] unbenutzt)
  { Himmelskoerper[] Stern = new Himmelskoerper[2];
    Mond m;
    Stern[0] = m = new Mond("Titan", null);
    Stern[1] = new Planet("Saturn", 7, m);
    System.out.println(((Planet) Stern[1]).Monde[0].Sicht);
    System.out.println(m.sichtbar(true));
    System.out.println(Stern[1].Sicht);
    System.out.println(Stern[1].sichtbar(true));
  }
}
```

Aufgabe 5.7: Sportliche Klassen und Arrays

In der neuen Sportart „Snowboardweitsprung“ wird ein Wettbewerb ausgetragen, an dem Amateure und Profis teilnehmen. Um das zu entrichtende Startgeld (ganzzahlig in Euro) zu bestimmen, die erzielten Sprungweiten (ganzzahlig in Metern) zu erfassen, daraus Punkte zu errechnen, den Sieger zu ermitteln und die Siegprämie (ganzzahlig in Euro) zu bestimmen, wurde das folgende Programm erstellt:

```
abstract class Sportler
{ String Name;  int[] Weite;

  Sportler(String n, int[] w)  { Name = n;  Weite = w; }

  abstract int Startgeld();

  int Punkte()
  { if (Weite == null  ||  Weite.length < 1)  return 0;
    int m = Weite[0];
    for (int i = 1;  i < Weite.length;  i++)
      if (Weite[i] > m)  m = Weite[i];
    return 2 * m;
  }
}

class Amateur extends Sportler
{ Amateur(String n, int[] w)  { super(n, w); }
  int Startgeld()  { return  10; }
}

class Profi extends Sportler
{ Profi(String n, int[] w)  { super(n, w); }
  int Startgeld()  { return 600; }
}

public class Snowboardweitsprung
{ public static void main(String[] unbenutzt)
  { Sportler[] s = new Sportler[5];
    int[] s0 = { 9, 14, 16};      s[0] = new Amateur("Sven", s0);
    int[] s1 = {13, 15, 11};      s[1] = new Amateur("Lisa", s1);
    int[] s2 = {26, 30, 21, 19};  s[2] = new Profi  ("Vera", s2);
    int[] s3 = {33, 12, 24, 27};  s[3] = new Profi  ("Erik", s3);
    int[] s4 = {14,  8, 12};      s[4] = new Amateur("Jens", s4); // ***
    int g = 0,  Sieger = 0;
    for (int i = 0;  i < s.length;  i++)
    { g += s[i].Startgeld();
      if (s[i].Punkte() > s[Sieger].Punkte())  Sieger = i;
      System.out.println(  s[i].Name + " " + s[i].Startgeld()
                         + " " + s[i].Punkte());
    }
    System.out.println(  s[Sieger].Name + " hat gewonnen"
                       + " und erhaelt " + g + " Euro.");
  }
}
```

a) Wie lautet die Ausgabe des Programms?

b) Ist die Anweisung `Sportler x = new Sportler(0);` zulässig?
(Antwort mit Begründung)

c) Für Profis sollen nun die Punkte auf eine andere Art berechnet werden: Die Punkteanzahl ergibt sich als Summe der Weiten der zwei besten Sprünge. Programmieren Sie für diese Punkteberechnung in der Klasse `Profi` eine geeignete Methode. Sie soll so realisiert sein, dass das restliche Programm unverändert bleiben kann. Die Punkteberechnung für Profis soll für eine beliebige Sprunganzahl korrekt arbeiten, wobei von mindestens zwei Sprüngen ausgegangen werden darf.

d) Das gegebene Programm ist „ungerecht“: Wie verhält es sich, wenn die höchste Punktanzahl von mehreren Sportler erreicht wird? Wer erhält die Siegprämie?

e) Programmieren Sie die Methode `main` ab der Stelle `... ("Jens", s4); // ***` neu, so dass die Siegprämie entsprechend den folgenden Regeln gerechter aufgeteilt wird:
   - Der beste Amateur soll als Siegprämie die Summe der Startgelder aller Amateure erhalten. Wenn mehrere Amateure die beste Punktzahl der Amateure erreichen, soll der Betrag unter diesen zu gleichen Teilen aufgeteilt werden (abgerundet auf ganze Euro).
   - Der beste Profi soll als Siegprämie die Summe der Startgelder aller Profis erhalten. Wenn mehrere Profis die beste Punktzahl der Profis erreichen, soll der Betrag unter diesen zu gleichen Teilen aufgeteilt werden (abgerundet auf ganze Euro).

   Ändern Sie auch die Ausgabeanweisung `System.out.println( ... hat gewonnen ... )` entsprechend ab.

Aufgabe 5.8: Klassenhierarchie zur Zeiterfassung

Zur Zeiterfassung bei der Feuerwehr wird das folgende Programm geschrieben. Jedes Objekt repräsentiert eine bestimmte Arbeit. Die Arbeitsdauer wird durch die Zeitpunkte `Beginn` und `Ende` bestimmt (ganzzahlig in Stunden des Tages).

```
abstract class Zeit
{ int Beginn, Ende;
  Zeit(int b, int e)  { Beginn = b;  Ende = e; }
  int Dauer()  { return Ende – Beginn; }
  float Bezahlung()  { return 20.0f * Dauer(); }
}

class Einsatz extends Zeit
{ String Art;
  Einsatz(String a, int b, int e)  { super(b, e);  Art = a; }
}

abstract class Bereitschaft extends Zeit
{ boolean Pausen;   // gibt an, ob Pausen während der Bereitschaft erlaubt sind.
  Bereitschaft(int b, int e)  { super(b, e);  Pausen = false; }
}

class PraesenzBereitschaft extends Bereitschaft   // in der Feuerwache.
{ PraesenzBereitschaft(int b)  { super (b, b + 8); }
}   // Hinweis: Die Präsenzbereitschaft dauert 8 Stunden.

class RufBereitschaft extends Bereitschaft   // telefonisch rufbereit zuhause.
{ int Rufnummer;
```

```
  RufBereitschaft(int r, int b, int e)  { super(b, e);  Rufnummer = r; }
  float Bezahlung()  { return super.Bezahlung() / 2; }
}

public class Arbeitszeiten
{ public static void main(String[] unbenutzt)
  { Zeit[] z = new Zeit[5];
    z[0] = new PraesenzBereitschaft(7);
    z[1] = new Einsatz("Feuer geloescht", 11, 14);
    z[2] = new PraesenzBereitschaft(7);
    z[3] = new Einsatz("Kaetzchen vom Dach geholt", 9, 10);
    z[4] = new RufBereitschaft(12345, 18, 24);
    ((Bereitschaft) z[2]).Pausen = true;
    float Bezahlung = 0;
    for (int i = 0;  i < z.length;  i++)
    { Bezahlung += z[i].Bezahlung();
      System.out.println(z[i].Dauer() + " h, " + z[i].Bezahlung() + " Euro");
    }
    System.out.println("gesamt: " + Bezahlung + " Euro");
  }
}
```

a) Wie lautet die Ausgabe des Programms?

b) Welche der gegebenen Methoden nimmt eine *dynamische Erweiterung* vor?

Für die Teilaufgaben c), d) und e) gilt:

- Die Klasse `Arbeitszeiten` darf nicht verändert werden.
- Schreiben sie zu den Teilaufgaben die entsprechenden Programmstücke und markieren Sie sie mit (1), (2), (3), ... Markieren Sie in dem gegebenen Programm jeweils die Stelle, wo das entsprechende Programmstück einzufügen ist, ebenfalls mit (1), (2), (3), ... , so dass zweifelsfrei klar ist, was wo einzufügen ist. Wenn an der Einfügestelle Programmstücke wegfallen sollten, sind diese durchzustreichen.

c) Ändern bzw. ergänzen Sie das Programm so, dass bei Erzeugung eines neuen Objekts des Typs `Rufbereitschaft` Pausen zunächst erlaubt sind, d.h. `Pausen` auf `true` gesetzt wird.

d) Die `Dauer` soll nun anders berechnet werden: Im Falle der `PraesenzBereitschaft` soll zur Differenz aus `Ende` und `Beginn` noch eine Stunde für die Fahrt zur Feuerwache dazugezählt werden.

e) Ändern bzw. ergänzen Sie das Programm so, dass sich die Bezahlung jeder Bereitschafts-Zeit um 10% erhöht, wenn Pausen nicht erlaubt sind.

f) Methode in der Klasse `Arbeitszeiten`: `static boolean gleich(Zeit[] z)`

Programmieren Sie den Rumpf dieser Methode. Sie soll genau dann `true` liefern, wenn alle im Array z gespeicherten Bereitschaftszeiten (egal, ob Präsenz- oder Rufbereitschaft) genau die gleiche Dauer aufweisen. Falls in z überhaupt keine Bereitschaftszeiten gespeichert sind, soll ebenfalls `true` zurückgegeben werden.

Aufgabe 5.9: Erweiterung von Klassen

Zur Beantwortung der Fragen in den Teilaufgaben a) bis d) genügen jeweils ca. 1 bis 3 Sätze.

a) Was versteht man unter dem Überschreiben von Methoden ?

b) Was versteht man unter dem Überladen von Methoden ?

c) Was versteht man unter Widening ?

d) Was versteht man unter einer abstrakten Methode ?

Gegeben seien die folgenden Klassen- und Variablenvereinbarungen:

```
class A
{ int x = 4;
  A(int x)  { this.x = x; }
  int m(int x)  { return x + this.x; }
}

class B extends A
{ int x = 5;
  B()  { super(6);  this.x = x; }
  int m(float x)  { return (int) x + super.x; }
  int m(int c)  { return x – c; }
}

B b = new B();
A a = b;
```

e) Angenommen die zuvor angegebenen Zuweisungen seien ausgeführt worden. Welche Werte liefern dann die folgenden fünf Ausdrücke ?

Hier die Lösung eintragen:

```
a.x
b.x
a.m(3)
b.m(3.0f)
b.m(3)
```

f) Angenommen in einer Methode ist nur der Verweis a (vom Datentyp A) verfügbar, nicht aber der Verweis b. Schreiben Sie ein (kurzes) Programmstück, das Folgendes erreicht:
Die Variable x der Unterklasse B wird auf 0 gesetzt, wenn das Objekt, auf das a verweist, mit B erweitert wurde. Andernfalls wird die Variable x der Oberklasse auf 0 gesetzt).