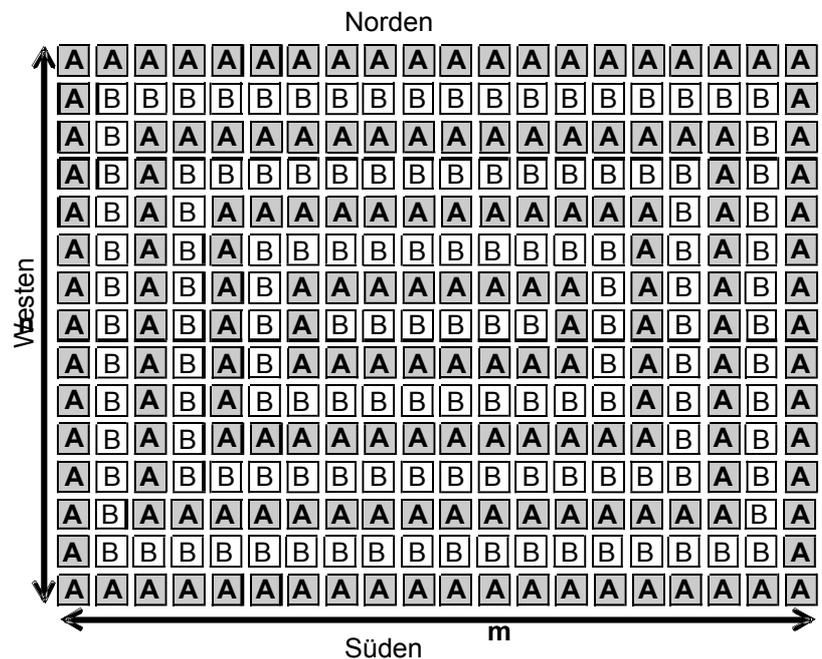


Hinweis: Für alle Aufgaben gilt, dass Sie in jeder Teilaufgabe Methoden benutzen können, die in einer anderen Teilaufgabe erstellt worden sind.

Aufgabe 1 [25 Punkte]

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 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.
[4 Punkte]

b) Programmieren Sie zu dem Methodenkopf `int Abstand(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. [5 Punkte]

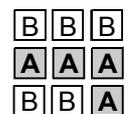
c) Programmieren Sie zu dem Methodenkopf `char Typ(int n, int m, int i, int j)` einen Methodenrumpf, der den Typ der Solarzelle mit Index `[i][j]` angibt (Rückgabewert entweder 'A' oder 'B'). [2 Punkte]

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. [4 Punkte]

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. [10 Punkte]



Aufgabe 2 [32 Punkte]

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

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

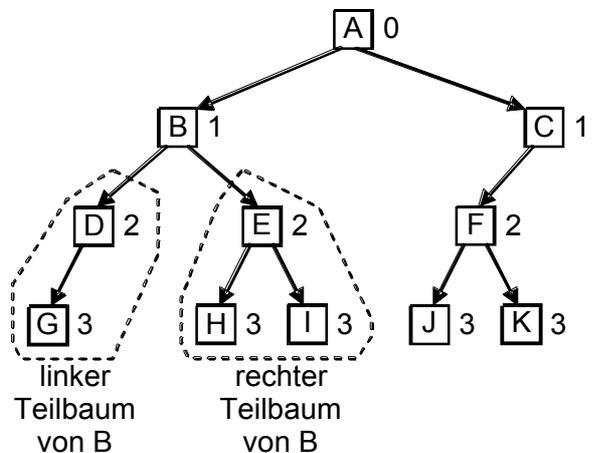
class Knoten
{ String Bez; // Bezeichnung des Knotens.
  Knoten links, rechts; // Linker bzw. rechter Nachfolger-Knoten.

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

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

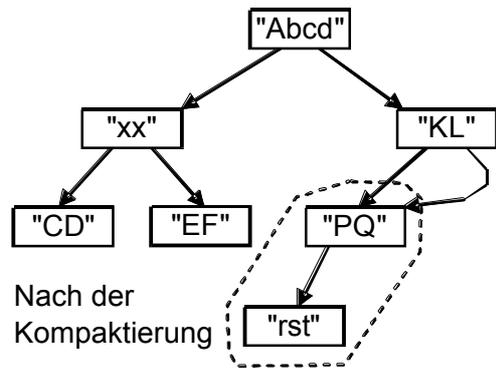
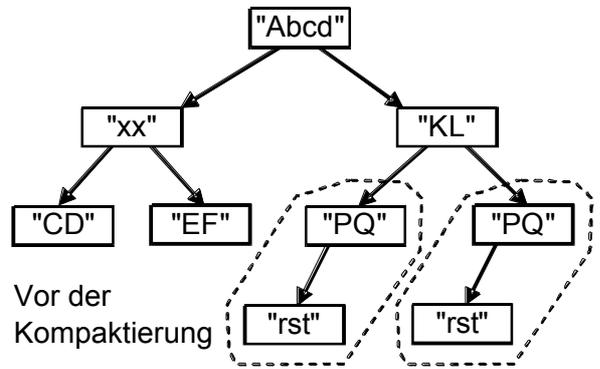
Unter dem linken Teilbaum eines Knotens `k` versteht man den Baum, dessen Wurzel `k.links` ist. Entsprechend ist `k.rechts` Wurzel des rechten Teilbaums von Knoten `k`.

- 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. [7 Punkte]



- 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. [10 Punkte]

- 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. [3 Punkte]



- d) Programmieren Sie zu dem Methoden-Kopf
 void kompaktiere() einen Methoden-Rumpf, der den Baum an allen Stellen kompaktiert,
 wo dies möglich ist. [4 Punkte]

- 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. [8 Punkte]

Aufgabe 3 [30 Punkte]

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

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

class Knoten
{ String Bez; Knoten Nf; // Nf zeigt auf den Nachfolger-Knoten.
  Kante Kopf, Fuss; // Kanten, die von einem Knoten ausgehen.
}

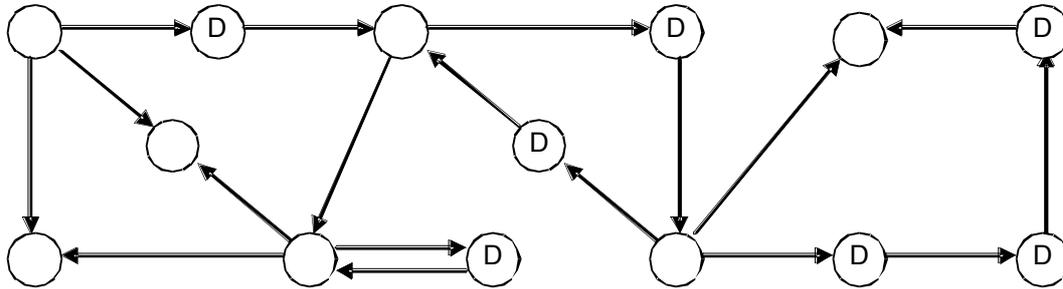
class Kante
{ String Bez; Kante Nf; // Nf zeigt auf die Nachfolger-Kante.
  Knoten Kante; // Knoten, zu dem die Kante hinführt.
}
```

- 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. [4 Punkte]

- 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. [8 Punkte]

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.

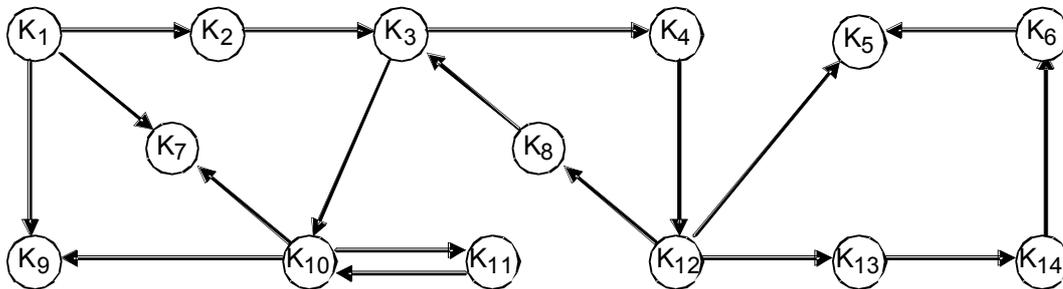


- c) Programmieren Sie zu dem Methoden-Kopf `boolean Du(Knoten k)` einen Methoden-Rumpf, der angibt, ob Knoten k ein Durchgangsknoten ist. [8 Punkte]

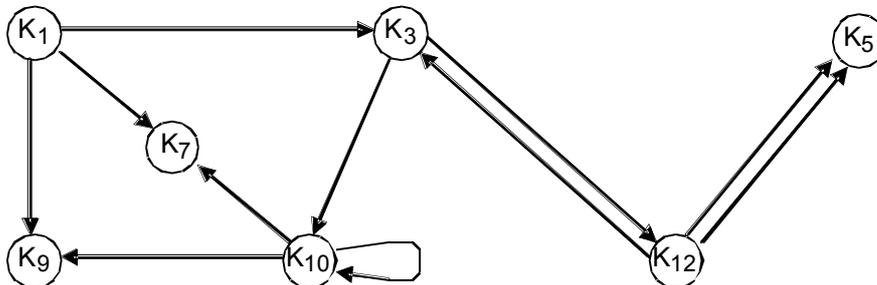
d) Programmieren Sie zu dem Methoden-Kopf `void ueberbr()` einen Methoden-Rumpf, der aus einem beliebigen Graphen alle Durchgangsknoten durch eine Kante überbrückt und dann löscht. Dies bedeutet: Wenn k ein Durchgangsknoten ist und es Kanten gibt von Knoten x zu Knoten k und von Knoten k zu Knoten y , 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}).

[10 Punkte] Als Beispiel zu dieser Teilaufgabe ein gegebener Graph:



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



Aufgabe 4 [13 Punkte]

Welche Ausgabe erzeugt das folgende Programm? Es wird empfohlen, aber nicht verlangt, zur Beantwortung dieser Frage eine Skizze anzufertigen, welche die erzeugte Objektstruktur darstellt. [13 Punkte]

```
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));
  }
}
```


Lösung 1

- a)

```
{ float Sum = 0.0f;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      Sum += s[i][j];
  return Sum;
}
```
- b)

```
{ return Math.min(Math.min(i, j),
  Math.min(n - 1 - i, m - 1 - j));
}
```
- c)

```
{ if (Abstand(n, m, i, j) % 2 == 0) return 'A';
  else return 'B';
}
```
- d)

```
{ float Sum = 0.0f;
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++)
      { if (Typ(n, m, i, j) == 'A') Sum += s[i][j];
      }
  return Sum;
}
```
- e)

```
{ int Anz = 0;
  for (int i = 2; i < n - 2; i++)
    for (int j = 2; j < m - 2; j++)
      if (Typ(n, m, i, j) == 'A')
        { int Summand = 1;
          for (int x = i - 1; x <= i + 1; x++)
            for (int y = j - 1; y <= j + 1; y++)
              { if (Typ(n, m, x, y) == 'B' && s[x][y] >= s[i][j])
                Summand = 0;
              }
          Anz += Summand ;
        }
  }
  return Anz;
}
```

Lösung 2

```
a) { return k2 != null && (tb(k1, k2.links) || tb(k1, k2.rechts));  
  }
```

```
boolean tb(Knoten k1, Knoten k2)  
{ return k2 != null  
  && (k1 == k2 || tb(k1, k2.links) || tb(k1, k2.rechts));  
}
```

```
b) { return k != null && gl(k.links, k.rechts); }
```

```
boolean gl(Knoten x, Knoten y)  
{ return x != null && y != null  
  && x.Bez.equals(y.Bez)  
  && gl(x.links, y.links) && gl(x.rechts, y.rechts)  
  || x == null && y == null;  
}
```

```
c) { if (gleich(k)) k.rechts = k.links;  
  }
```

```
d) { komp(Wurzel); }
```

```
void komp(Knoten k)  
{ kompaktiere(k);  
  if (k != null)  
  { komp(k.links); komp(k.rechts);  
  }  
}
```

```
e) { expan(Wurzel); }
```

```
void expan(Knoten k)  
{ if (k != null)  
  { if (k.links != null && k.links == k.rechts)  
    k.rechts = dupl(k.links);  
    expan(k.links); expan(k.rechts);  
  }  
}
```

```
Knoten dupl(Knoten k)
```

```
{ if (k != null)  
  { Knoten neu = new Knoten(k.Bez);  
    neu.links = dupl(k.links); neu.rechts = dupl(k.rechts);  
    return neu;  
  }  
  else return null;  
}
```

Lösung 3

- a) { if (a == null) return null;
 for (Kante e = a.Kopf; e != null; e = e.Nf)
 if (e.Kante == b) return e;
 return null;
}
- b) { Knoten x, vx = null;
 for (x = Kopf; x != null; vx = x, x = x.Nf)
 if (x == k) break;
 if (x != null && x == k)
 { if (vx == null) Kopf = Kopf.Nf;
 else vx.Nf = k.Nf;
 if (Fuss == k) Fuss = vx;
 }
}
- c) { int Anz = 0;
 if (k.Kopf == null || k.Kopf.Nf != null) return false;
 for (Knoten x = Kopf; x != null; x = x.Nf)
 if (exKante(x, k) != null) Anz++;
 return Anz == 1;
}
- d) { Knoten k, x; Kante e;
 for(k = Kopf; k != null; k = k.Nf)
 if (Du(k))
 { x = Kopf;
 while ((e = exKante(x, k)) == null) x = x.Nf;
 e.Kante = k.Kopf.Kante;
 loescheKn(k);
 }
}

Lösung 4

Himmelskörper: Titan

Mond: Scheibe

Himmelskörper: Saturn

Planet: Punkt

Scheibe

schwierig

unbekannt

Punkt (schwach)