Musterlösung zu Übungsblatt 5

Praktische Informatik I

Wintersemester 1999/2000

[Aufgabentext ist jeweils in kursiv gesetzt]

Aufgabe 1: Schrittweise Verfeinerung [4 Punkte]

Zu Semesterbeginn erstellen Sie sich üblicherweise einen Stundenplan, in dem Sie alle für Sie relevanten Veranstaltungen eintragen. Formulieren Sie dazu einen Algorithmus durch schrittweise Verfeinerung (in 2-3 Schritten) unter Angabe aller Randbedingungen, von denen Sie ausgehen (Studiengang, Semesterzahl etc.).

Randbedingungen:

- Studiengang: Wirtschaftsinformatik
- x-tes Semester
- nur Vorlesungsverzeichnis als Informationsgrundlage
- keine Änderungen, sondern nur Ersterstellung

1. Verfeinerung:

1. Vorlage vorbereiten
2. Falls *Grundstudium*
   (a) Veranstaltungen heraussuchen (Informatik)
   (b) Veranstaltungen heraussuchen (BWL)
   (c) Veranstaltungen heraussuchen (Statistik)
   (d) Veranstaltungen heraussuchen (Mathematik)
3. Falls *Hauptstudium*
   (a) Veranstaltungen heraussuchen (Informatik)
   (b) Veranstaltungen heraussuchen (WInf)
   (c) Veranstaltungen heraussuchen (Spez.BWL)
   (d) Veranstaltungen heraussuchen (Wahlpflichtfach)
4. Falls *Überschneidungen*
   (a) Überschneidungen auflösen

2. Verfeinerung:

1. Vorlage vorbereiten
   (a) Papier besorgen
   (b) Muster mit Blockzeiten vorbereiten

2. Veranstaltung heraussuchen(FACH)
   (a) Vorlesungen für FACH für x-tes Semester aus Vorlesungsverzeichnis herausschreiben und in Stundenplan eintragen
   (b) Falls *große Übung in FACH angeboten*
      i. Übung für FACH für x-tes Semester aus Vorlesungsverzeichnis herausschreiben und in Stundenplan eintragen
   (c) Falls *(kleine) Übungen in FACH angeboten*
      i. passende Übung(en) für FACH für x-tes Semester aus Vorlesungsverzeichnis herausschreiben und in Stundenplan eintragen
   (d) Falls *Tutorium in FACH nötig*
      i. passendes Tutorium für FACH für x-tes Semester aus Vorlesungsverzeichnis herausschreiben und in Stundenplan eintragen
   (e) Falls *Seminar in FACH nötig*
      i. passendes Seminar für FACH aus Vorlesungsverzeichnis herausschreiben und in Stundenplan eintragen

3. Überschneidungen auflösen
   (a) Lege Priorität für parallel stattfindende Veranstaltungen fest

(b) Prüfe, ob eine der Veranstaltungen auch an einen anderen, passenderen Zeitpunkt gelegt werden kann und verschiebe entsprechend
(c) Falls *Überschneidung nicht auflösbar*
   i. Streiche alle überschneidenden Veranstaltungen außer derjenigen mit der höchsten Priorität

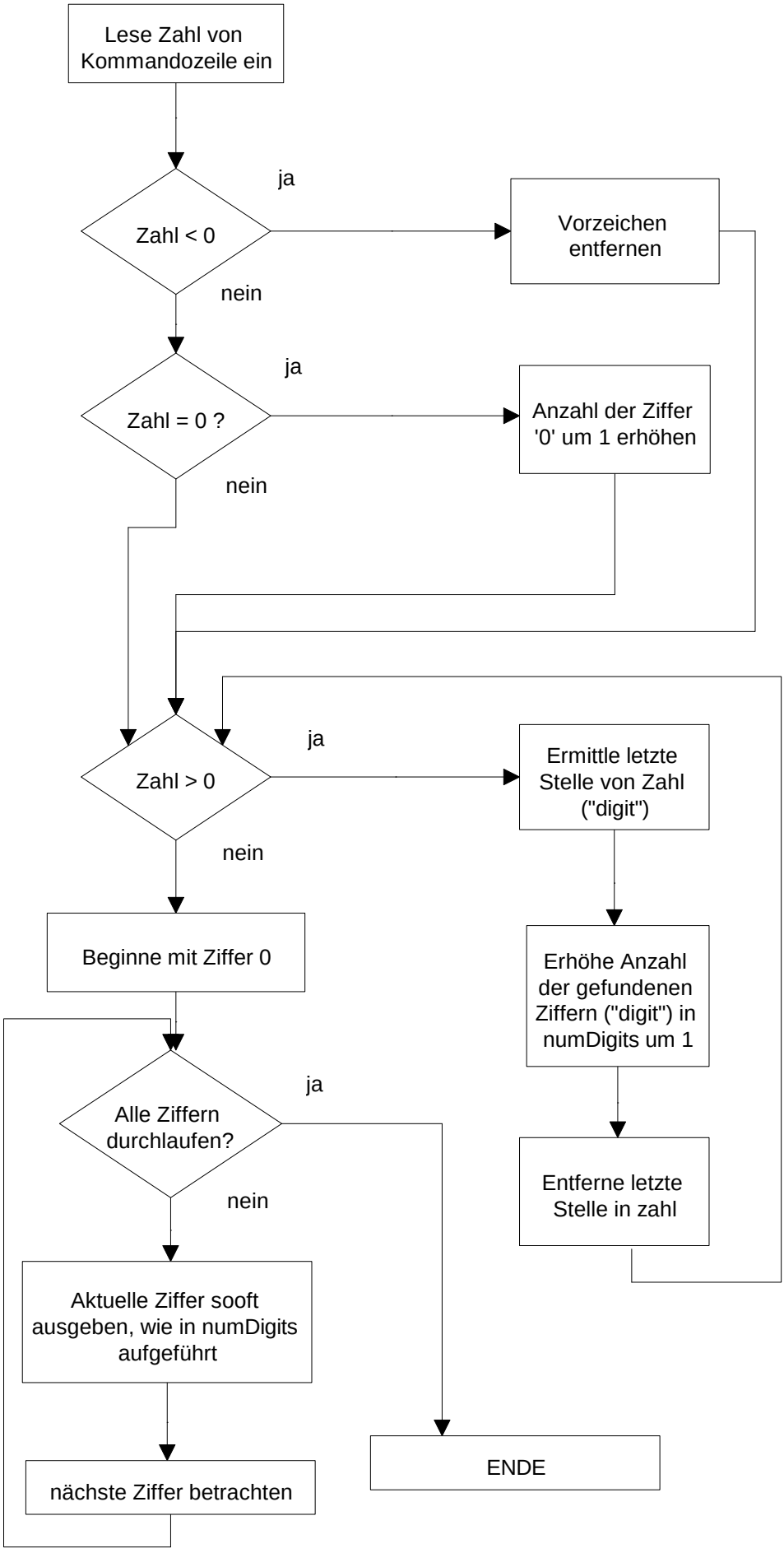
Aufgabe 2: Algorithmenentwurf und Java-Programmierung [8 Punkte]

Entwickeln Sie ein Verfahren, das eine beliebige Integerzahl einliest und die Ziffern aufsteigend sortiert ausgibt.

Beispiel: Eingabe 739811
Ausgabe 113789

(a) [4 Punkte] Formulieren Sie (auf Papier) einen geeigneten Algorithmus in Pseudocode und erstellen Sie ein Flußdiagramm.

Lege ein Feld mit 10 Elementen an, welches die Häufigkeit jeder Ziffer festhält
Lese die Zahl von der Kommandozeile ein
Entferne evtl. enthaltene Vorzeichen
Falls die angegebene Zahl 0 war, setze die Anzahl an „0“ im Feld auf 1
Solange die Zahl größer Null ist
    Betrachte die letzte Ziffer der Zahl
    Zähle zu der bisherigen Anzahl dieser Ziffer im Feld 1 dazu
    Schneide die letzte Ziffer der Zahl ab
Durchlaufe das Feld in aufsteigender Richtung (0 bis 9)
    Gebe die jeweilige Ziffer sooft aus, wie im Feld für diese Ziffer angegeben.


Lese Zahl von Kommandozeile ein
Zahl < 0
ja
nein
Vorzeichen entfernen
Zahl = 0 ?
ja
nein
Anzahl der Ziffer '0' um 1 erhöhen
Zahl > 0
ja
nein
Ermittle letzte Stelle von Zahl ("digit")
Beginne mit Ziffer 0
Erhöhe Anzahl der gefundenen Ziffern ("digit") in numDigits um 1
Alle Ziffern durchlaufen?
ja
nein
Entferne letzte Stelle in zahl
Aktuelle Ziffer sooft ausgeben, wie in numDigits aufgeführt
ENDE
nächste Ziffer betrachten

(b) [4 Punkte] Implementieren Sie den Algorithmus als Java-Application und geben Sie das Programm über abox ab.

```
/*****************************************************************************
*
* Project name: Musterloesung Zahlensortierung
*
* Date: 29.11.1999
*
* Name: Zahlensortierung.java
*
* Description:
*   Reads in a number given as command line argument, sorts the digits in
*   ascending order and displays the result on standard output.
*
*****************************************************************************/

class Zahlensortierung {

    /*****************************************************************************
     *
     * Name: main
     *
     * Description:
     *   This method reads in the first command line parameter, converts it
     *   to int, possibly removes a minus-sign, and counts the digits in the
     *   remaining number as follows:
     *     - as long as we have a number above 0, consider the last digit
     *       by calculating the remainder of a division by 10.
     *     - this digit is taken as index to an array which counts the occurances
     *       of every 10 digits (numDigits) and the corresponding array value is
     *       being incremented by 1.
     *     - we divide our number by 10, thus cutting off the current (last) digit.
     *   Once we have finished couting our number's digits, we find the number
     *   of each digit in the numDigits-Array.
     *   We then simply go through the numDigits-Array - starting at 0 - and print
     *   the actual digit we are working on, as long as there are more than 0
     *   occurances of this digit.
     *
     *   The algorithm is based on "counting sort".
     *
     * Parameters:
     *   The method requires one parameter to be passed from command line, which
     *   is the number whose digits are to be sorted in ascending order.
     *
     *****************************************************************************/

  public static void main (String[] args) {

    int number;                         // number to be read in as command line argument
    int digit;                          // digit we are currently working on
    int[] numDigits = new int[10];      // array, which counts the occurances of each digit

    if (args.length != 1) {
       System.out.println ("Start with: java Zahlensortierung IntegerNumber");
       System.exit (1);
    }

    number = Integer.parseInt (args[0]);     // read in command line argument and
                                             // convert it to int

    if (number < 0) {                   // negative number ?
      number = -number;                 // remove minus-sign
    } else {
      if (number == 0) {                // number is already zero ?
        numDigits[0] = 1;               // adapt the number of zeros in numDigits array
      }
    }

    // Fill numDigits-Array

    while (number > 0) {          // still some work to do?
      digit = number % 10;        // get the last digit (rightmost)
      numDigits[digit]++;         // now that we've found a digit, adapt numDigits accordingly
      number = number / 10;       // cut off rightmost digit
    }
```

```
    // Print out the results...

    for (digit = 0; digit < 10; digit++)               // go through array in ascending order
      for ( ; numDigits[digit] > 0; numDigits[digit]-- )   // as long as this digit occurs...
        System.out.print (digit);                          // ... print it out!
    System.out.println ("");
  }
}
```

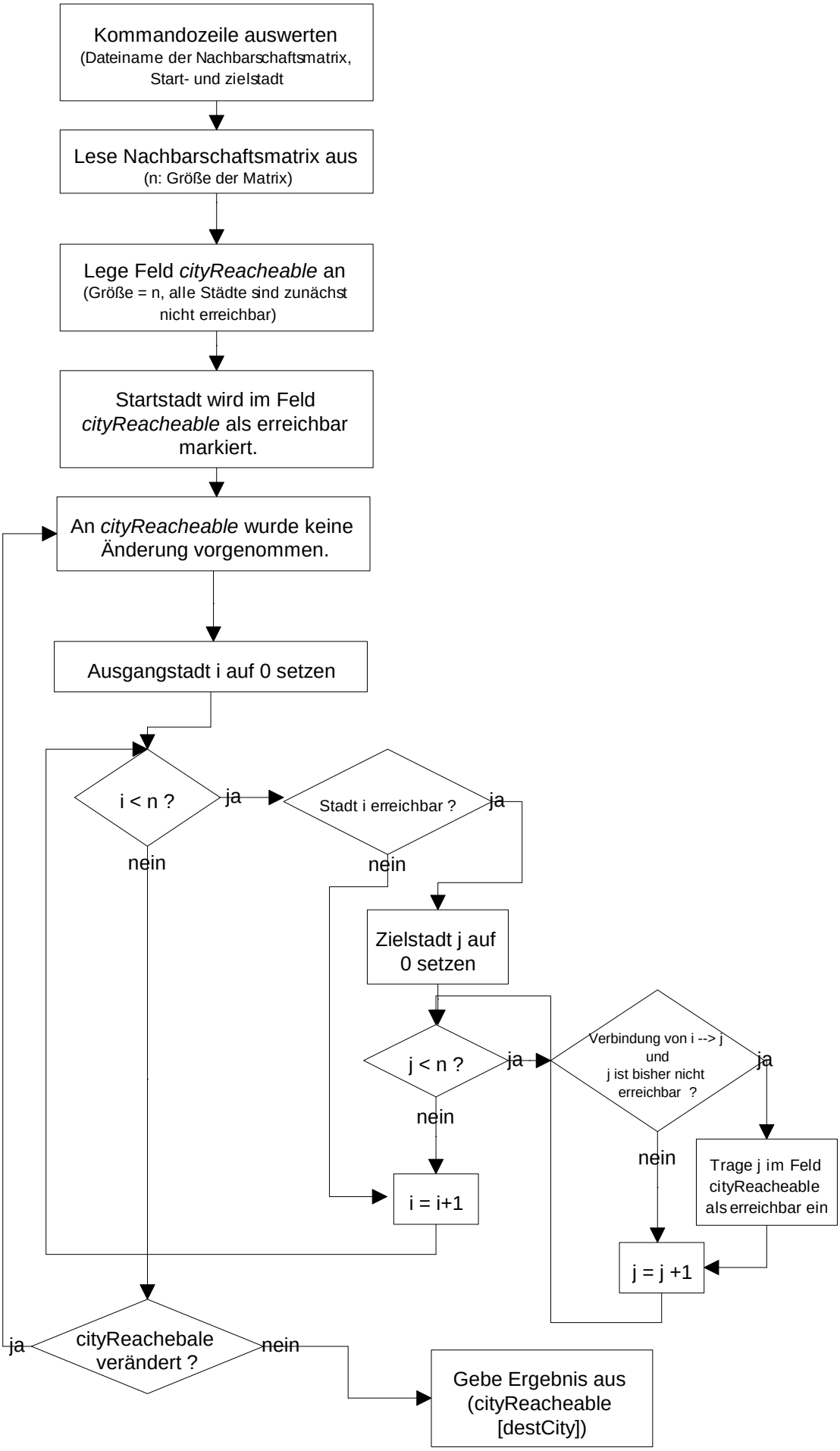
Aufgabe 3: Algorithmenentwurf und Java-Programmierung [8 Punkte]

Mehrere Städte sind durch ein Straßennetz miteinander verbunden (siehe Abbildung auf Übungsblatt 5). Zur Repräsentation dieses Sachverhalts im Computer wird eine Nachbarschaftsmatrix M verwendet, aus der sich erkennen läßt, ob zwei benachbarte Städte i und j eine direkte Verbindung haben (M[i][j] = 1) oder nicht (M[i][j] = 0). Diese Verbindungen sind immer bidirektional, d.h. es gibt in unserem Modell keine Einbahnstraßen.

Entwerfen Sie einen Algorithmus, mit dem sich überprüfen läßt, ob zwei Städte i und j, i ! = j,
durch das Straßennetz verbunden sind.

(a) [4 Punkte] Formulieren Sie (auf Papier) einen geeigneten Algorithmus in Pseudocode
und erstellen Sie ein Flußdiagram.

Ermittle Dateiname der Nachbarschaftsmatrix, sowie Start- und Zielstadt aus den Kommandozeilenparametern
Lese die Nachbarschaftsmatrix aus der angegeben Datei ein
Lege ein Feld (cityReacheable) an, das darüber Auskunft gibt, ob eine Stadt von der Startstadt aus erreichbar ist
Markiere die Startstadt zunächst als erreichbar (cityReacheable = true)
Tue
    Bisher wurden keine Änderungen am Feld (cityRecheable vorgenommen)
    Für alle Städte i von 0 bis n-1 tue
        Ist die Stadt i von der Startstadt aus erreichbar ?
            Für alle Städte j von 0 bis n-1 tue
                Gibt es von der Stadt i zu der Stadt j eine direkte Verbindung,
                die bisher im Feld noch nicht berücksichtigt wurde ?
                    Markiere die Stadt j im Feld (cityReacheable) als erreichbar (true)
                    Es wurden jetzt Änderungen am Feld vorgenommen
Solange Änderungen am Feld (cityReacheable) vorgenommen wurden
Gebe das Ergebnis aus, in dem im Feld nachgeschaut wird, ob die Zielstadt erreichbar ist.


Kommandozeile auswerten
(Dateiname der Nachbarschaftsmatrix, Start- und zielstadt
Lese Nachbarschaftsmatrix aus
(n: Größe der Matrix)
Lege Feld *cityReacheable* an
(Größe = n, alle Städte sind zunächst nicht erreichbar)
Startstadt wird im Feld *cityReacheable* als erreichbar markiert.
An *cityReacheable* wurde keine Änderung vorgenommen.
Ausgangstadt i auf 0 setzen
i < n ?
ja
nein
Stadt i erreichbar ?
ja
nein
Zielstadt j auf 0 setzen
j < n ?
ja
nein
Verbindung von i --> j und j ist bisher nicht erreichbar ?
ja
nein
Trage j im Feld cityReacheable als erreichbar ein
i = i+1
j = j +1
cityReachebale verändert ?
ja
nein
Gebe Ergebnis aus (cityReacheable [destCity])

(b) [4 Punkte] Implementieren Sie den Algorithmus als Java-Application und geben Sie das Programm über abox ab.

```
/*****************************************************************************
*
* Project name: Musterloesung Strassennetz
*
* Date: 29.11.1999
*
* Name: Strassennetz.java
*
* Description:
*   Finds out, if two cities are connected (in)directly given a neighbourhood
*   matrix and shows the result on standard output.
*****************************************************************************/

import java.io.*;
import java.util.StringTokenizer;

class Strassennetz {

    /*****************************************************************************
     *
     * Name: main
     *
     * Description:
     *   - Reads in the neighbourhood matrix given as filename in [0]
     *   - Takes command line arguments [1] and [2] as start and stop city
     *     respectively
     *   - creates an array cityReacheable. Every city marked as true herein
     *     can be reached from the start city. At first every city except for
     *     the start city is marked unreacheable
     *   - step through the cityReachebale array and for every reacheable city
     *     try to find any neighbourhood cities, which aren't marked as reacheable
     *     in cityReacheable yet. If we find a new neighbour, we set a flag
     *     "changed" to indicate that the contents of cityReacheable changed and
     *     change cityRecheable accordingly.
     *   - we repeat the last step as long as we find any changes in the array
     *
     *   NOTE: All cities start with index "0", so if you state a connection from 1 to 2,
     *         actually a connection from indices 0 to 1 is being checked!
     *         Only [1] and [2] are directly dependant on this
     *
     * Parameters:
     *   command line parameters:
     *     [0] File name from which the neighbourhood-matrix is being read
     *     [1] Number of the city to start from
     *     [2] Number of the destination city
     *****************************************************************************/

   public static void main (String[] args) {

      int         i, j;                   // variables needed within loops only
      int         n = 0;                  // size of the neighbourhood-matrix
      boolean     changed;                // Indicates any changes on the array cityReacheable
      boolean[]   cityReacheable;         // Array, which indicates if a city with a
                                          // given index (0...n-1) can be reached from
                                          // the start city

      int[][]     adjacentMatrix;         // neighbourhood matrix adjacentMatrix [index1][index2]
                                          // which indicates whether there is a connection from city
                                          // index1 to city index2.

      String      zeile = "";             // needed for reading in the file

      if (args.length != 3) {             // program called with wrong argument number?
         System.out.println ("Start with: java Strassennetz FileName.txt startCity destCity");
         System.exit (1);
      }

      // NOTE that actual city numbers start with 1 but our array-indices use 0 as
      //      lowest number, therefore startCity and destCity are adjusted accordingly.

      int startCity = Integer.parseInt(args[1]) - 1;    // read in number of start-city
      int destCity  = Integer.parseInt(args[2]) - 1;    // read in number of destination-city

      if (startCity == destCity) {                      // this case is simply undefined as the
        System.out.println ("(nicht definiert)");       // text states that i and j to be differnet
        System.exit (0);
      }
```

```
// Make file in args[0] available for reading through a BufferedReader

BufferedReader in = null;

try {                          //attempt to open input stream from file
  in = new BufferedReader(new FileReader(args[0]));
} catch(FileNotFoundException fnfe) {
  System.out.println ("Datei " + args[0] + " nicht gefunden!");
  System.exit(1);
}

// Read in size of neighbourhood matrix "n" from file

try {
  zeile = in.readLine ();                    // read in first line of file
  n = Integer.parseInt(zeile);
} catch (IOException ioe) {
  System.out.println ("n konnte nicht aus der Datei gelesen werden");
  System.exit(1);
}

if (n < 1) {                                 // no use in considering this...
  System.out.println ("Adjazenzmatrix hat Größe < 1");
  System.exit(1);
}

// Create neighbourhood matrix with the given argument n (size of neighbourhood matrix)

adjacentMatrix = new int[n][n];

// Read in neighbourhood matrix from file

for (i = 0; i < n; i++) {            // Read "n" Lines
  try {
    zeile= in.readLine();            // read one row
  } catch (IOException ioe) {        // We've encountered an error
    System.out.println ("Fehler beim Lesen von " + args[0]);  // show error msg
    System.exit (1);                 // and leave program
  }

  StringTokenizer st= new StringTokenizer(zeile);
  for (j = 0; j < n; j++)            // and extract "n" Integer Values from each line
    {
      String s= st.nextToken();      // different tokens are separated by whitespaces
      int wert = Integer.parseInt(s);
      adjacentMatrix[i][j] = wert;
    }
}

cityReacheable = new boolean[n];      // create new Array cityReachebale with every city
                                      // marked as unreacheable.
cityReacheable[startCity] = true;     // we need to define the current (starting) city
                                      // as reacheable so that our algorithm has a point
                                      // to start from
do {
  changed = false;                    // initialize our changed-flag to false, i.e. we didn't
                                      // modify the cityReacheable-array yet.
  for (i = 0; i < n; i++) {           // step through all the cities (0...n-1)
     if (cityReacheable[i]) {         // YEP! We found a city i, which is reacheable from the
                                      // startCity
                                      // (according to what we've found out so far)
        for (j = 0; j < n; j++) {     // looking from this one city i we then try to
                                      // find a neighbour j
                                      // (= a connection to any other city).
           if ((adjacentMatrix[i][j] == 1) && !cityReacheable[j]) {
                                           // BINGO! City i has a neighbour j
                                           // and j wasn't known to be reacheable from
                                           // startCity so far - this is something new to
                                           // include in our cityReacheable-array.

             cityReacheable[j] = true;     // the neighbour j we found is reacheable from i and
                                           // consequently from our startCity
             changed = true;               // we did some changes in the array...
           }
        }
     }
  }
} while (changed);   // once we didn't encounter any more changes in the array, we actually found
                     // every city which can be reached from the start city  - otherwise there are
```

```
                        // still cities which are reacheable from the start city but whose
                        // neighbours are not (yet) included in cityReacheable...

      System.out.println (cityReacheable[destCity] ? "ja" : "nein");   // show result
    }
}
```