File - C:\DEV\GroProTest\GroProJava\src\main\java\de\pietsch\model\ProblemModell.java

```java
package de.pietsch.model;


import java.awt.*;
import java.util.Arrays;


/**
 * Datenklasse zum Speichern des Problems und er Lösung.
 */
public class ProblemModell {
    public String beschreibung;
    public int xDim;
    public int yDim;
    public int[][] hoehen;

    public int antennenAnz;
    public Point[] antennen;

    /**
     * Hilfsmethode zur Konsolenausgabe des Problems.
     */
    public void printModel() {
        System.out.println(beschreibung);
        System.out.println(xDim + " " + yDim);
        StringBuilder sb = new StringBuilder();
        for (int[] zeile : hoehen) {
            for (int eintrag : zeile) {
                sb.append(eintrag).append(" ");
            }
            sb.append("\n");
        }
        sb.append("Antennenanzahl: ");
        sb.append(antennenAnz);
        sb.append("\n");
        sb.append("Antennen-Punkte: \n");
        sb.append(Arrays.toString(antennen));
        System.out.println(sb.toString());
    }


}

```

File - C:\DEV\GroProTest\GroProJava\src\main\java\de\pietsch\model\SchnittPunktContainer.java

```java
package de.pietsch.model;

import de.pietsch.geometry.Vektor3D;

/**
 * Container zum Speichern es Ergebnisses aus der Schnittpunkt-Ermittlung.
 */
public class SchnittPunktContainer {
    public int code;
    public Vektor3D schnittPunkt;
}

```

```java
package de.pietsch.geometry;

import de.pietsch.model.SchnittPunktContainer;


/**
 * Klasse zum Umgang mit Ebenen im R3.
 */
public class Ebene {
    public Vektor3D normalVektor;
    public Vektor3D aufPunkt;

    /**
     * Der Konstuktor erstellt aus drei Punkten eine Ebene. Die Punkte müssen eine Ebenen definieren.
     *
     * @param aufPunkt
     * @param punkt1
     * @param punkt2
     */
    public Ebene(Vektor3D aufPunkt, Vektor3D punkt1, Vektor3D punkt2) {
        this.aufPunkt = aufPunkt;
        Vektor3D r1 = punkt1.subtrahiere(aufPunkt);
        Vektor3D r2 = punkt2.subtrahiere(aufPunkt);
        normalVektor = r1.kreuzProdukt(r2);
    }

    /**
     * Es wird der Schnittpunkt mit einer Geraden ermittelt.
     * Dabei liegt die gerade entweder parallel zur Ebene, in der Ebene oder sie schneidet die Ebene in einem Punkt.
     * Die drei Möglichkeiten sind im SchnittPunktContainer kodiert.
     *
     * @param geradeAufpunkt
     * @param richtung
     * @return SchnittPunktContainer: Code=0 -> Schnittpunkt, Code=1 -> parallel zur Ebene, Code=2 -> in der Ebene
     */
    public SchnittPunktContainer findeEbeneGeradeSchnittpunkt(Vektor3D geradeAufpunkt, Vektor3D richtung) {
        double epsilon = 0.00001;
        SchnittPunktContainer retVal = new SchnittPunktContainer();
        retVal.code = 0;
        double nDotV = normalVektor.skalarProdukt(richtung);
        if (Math.abs(nDotV) < epsilon) {
            retVal.code = 1;
            double punktprobe = normalVektor.skalarProdukt(geradeAufpunkt.subtrahiere(aufPunkt));
            if (Math.abs(punktprobe) < epsilon) {
                retVal.code = 2;
            }
        } else {
            Vektor3D w = aufPunkt.subtrahiere(geradeAufpunkt);
            double wDotN = w.skalarProdukt(normalVektor);
```

```
50              double quotient = wDotN / nDotV;
51              retVal.schnittPunkt = geradeAufpunkt.addiere(richtung.
   multSkalar(quotient));
52          }
53          return retVal;
54      }
55
56
57 }
58
```

File - C:\DEV\GroProTest\GroProJava\src\main\java\de\pietsch\geometry\Vektor3D.java

```java
package de.pietsch.geometry;

/**
 * Klasse zum Umgang mit Vektoren im R3.
 */
public class Vektor3D {
    public double x;
    public double y;
    public double z;

    public Vektor3D(double x, double y, double z) {
        this.x = x;
        this.y = y;
        this.z = z;
    }

    /**
     * Führt eine einfache Vektor-Subtraktion durch
     *
     * @param other
     * @return Es wird ein neues Vektor-Objekt mit dem Ergebnis zurückgegeben. Die aktuelle Instanz wird nicht verändert.
     */
    public Vektor3D subtrahiere(Vektor3D other) {
        return new Vektor3D(this.x - other.x,
                this.y - other.y,
                this.z - other.z);
    }

    /**
     * Führt eine einfache Vektor-Addition durch
     *
     * @param other
     * @return Es wird ein neues Vektor-Objekt mit dem Ergebnis zurückgegeben. Die aktuelle Instanz wird nicht verändert.
     */
    public Vektor3D addiere(Vektor3D other) {
        return new Vektor3D(this.x + other.x,
                this.y + other.y,
                this.z + other.z);
    }

    /**
     * Führt ein Vektor-Kreuzprodukt durch. Der Ergebnisvektor steht orthogonal auf beiden beteiligten Vektoren.
     *
     * @param other
     * @return Es wird ein neues Vektor-Objekt mit dem Ergebnis zurückgegeben. Die aktuelle Instanz wird nicht verändert.
     */
    public Vektor3D kreuzProdukt(Vektor3D other) {
        return new Vektor3D(this.y * other.z - this.z * other.y,
                this.z * other.x - this.x * other.z,
                this.x * other.y - this.y * other.x);
    }
```

```
52
53      /**
54       * Berechnung des Skalarprodukts zweier Vektoren. Kann zur
   ermittlung von Orthogonalität verwendet werden.
55       *
56       * @param other
57       * @return Skalarer Wert.
58       */
59      public double skalarProdukt(Vektor3D other) {
60          return this.x * other.x + this.y * other.y + this.z * other.
   z;
61      }
62
63      /**
64       * Multiplikation des Vektors mit einem Skalar.
65       *
66       * @param skalar
67       * @return Es wird ein neues Vektor-Objekt mit dem Ergebnis
   zurückgegeben. Die aktuelle Instanz wird nicht verändert.
68       */
69      public Vektor3D multSkalar(double skalar) {
70          return new Vektor3D(x * skalar, y * skalar, z * skalar);
71      }
72
73      /**
74       * Berechnung der euklidischen Norm des Vektors. Es werden nur
   die x- und y-Komponente betrachtet. Kann als Distanz-Maß auf der XY-
   Ebene verwendet werden.
75       *
76       * @return Länge des Vektors auf der XY-Ebene
77       */
78      public double normXundY() {
79          return Math.sqrt(x * x + y * y);
80      }
81
82      @Override
83      public String toString() {
84          return "(" + x + "|" + y + "|" + z + ")";
85      }
86
87      @Override
88      public boolean equals(Object obj) {
89          if (obj == null) {
90              return false;
91          }
92          if (obj.getClass() != this.getClass()) {
93              return false;
94          }
95          Vektor3D other = (Vektor3D) obj;
96          return this.x == other.x && this.y == other.y && this.z ==
   other.z;
97      }
98  }
99
```

```java
package de.pietsch.controller;

import de.pietsch.implementation.BruteForce;
import de.pietsch.implementation.StandardAusgabe;
import de.pietsch.implementation.StandardEingabe;
import de.pietsch.interfaces.AusgabeStrategie;
import de.pietsch.interfaces.EingabeStrategie;
import de.pietsch.interfaces.LoesungsStrategie;
import de.pietsch.model.ProblemModell;

import java.io.IOException;
import java.nio.file.NoSuchFileException;

/**
 * Hauptklasse zur Steuerung des Programms. Benötigt werden eine EingabeStrategie, eine LoesungsStrategie und eine AusgabeStrategie.
 * Diese werden in chronologischer Reihenfolge auf ein angegebenes BeispielProblem angewandt.
 */
public class AntennenSucheController {
    private ProblemModell beispielProblem;
    private final EingabeStrategie eingabe;
    private final LoesungsStrategie loeser;
    private final AusgabeStrategie ausgabe;
    private final String eingabedateipfad;
    private final int beispielNummer;

    public AntennenSucheController(EingabeStrategie eingabe, LoesungsStrategie loeser, AusgabeStrategie ausgabe, String eingabeDateiPfad, int beispielNummer) {
        this.eingabe = eingabe;
        this.loeser = loeser;
        this.ausgabe = ausgabe;
        this.eingabedateipfad = eingabeDateiPfad;
        this.beispielProblem = null;
        this.beispielNummer = beispielNummer;
    }

    /**
     * Es wird ein Beispiel-Problem eingelesen, eine Lösung ermittelt und das Ergebnis ausgegeben.
     */
    public void findeMinimaleAntennen() {
        boolean eingabeErfolgreich = true;
        try {
            beispielProblem = eingabe.leseDatei(eingabedateipfad);
        } catch (NumberFormatException ne) {
            System.out.println(ne.getMessage() + "\nAlle Zahlen-Eingaben sollten ganzzahlig sein!");
            eingabeErfolgreich = false;
        } catch (NoSuchFileException nf) {
            System.out.println("Die angegebene Datei konnte nicht gefunden werden:");
            System.out.println(nf.getMessage());
            eingabeErfolgreich = false;
```

```
49          } catch (IOException | IllegalArgumentException e) {
50              System.out.println(e.getMessage());
51              eingabeErfolgreich = false;
52          }
53
54          if (eingabeErfolgreich) {
55              beispielProblem = loeser.loeseProblem(beispielProblem);
56              try {
57                  ausgabe.gebeAus(beispielProblem, beispielNummer);
58              } catch (IOException e) {
59                  System.out.println("Fehler beim Schreiben der
   Dateien.");
60              }
61          }
62
63      }
64
65      public static void main(String[] args) {
66          if (args.length == 2) {
67              String dateiPfad = args[0];
68              int beispielNummer = Integer.parseInt(args[1]);
69
70              EingabeStrategie eingabe = new StandardEingabe();
71              LoesungsStrategie loeser = new BruteForce();
72              AusgabeStrategie ausgabe = new StandardAusgabe();
73
74              AntennenSucheController controller = new
   AntennenSucheController(eingabe, loeser, ausgabe, dateiPfad,
   beispielNummer);
75              controller.findeMinimaleAntennen();
76          } else {
77              System.out.println("Fehler bei den Uebergabeparametern
   beim Programmstart. Es wird erwartet:\n1. Der Pfad der Problem-Datei
   \n2. Die Nummer des Tests");
78          }
79      }
80
81
82  }
83
```

```java
package de.pietsch.interfaces;

import de.pietsch.model.ProblemModell;

import java.io.IOException;

/**
 * Interface zur allgemeinen Ausgabe von gelösten Problemen.
 */
public interface AusgabeStrategie {

    /**
     * Allgemeine Methode zur Ausgabe von gelösten Probleme.
     *
     * @param bspProb:        Das gelöste Problem.
     * @param beispielNummer: Nummer des Problems zur Benennung von Dateien.
     * @throws IOException
     */
    public void gebeAus(ProblemModell bspProb, int beispielNummer) throws IOException;
}

```

```java
package de.pietsch.interfaces;

import de.pietsch.model.ProblemModell;

import java.io.IOException;

/**
 * Interface zum allgemeinen Einlesen von Problemen.
 */
public interface EingabeStrategie {

    /**
     * Allgemeine Einlesemethode.
     *
     * @param pfad: Dateipfad
     * @return ProblemModell: Das zu lösende Problem.
     * @throws IOException
     */
    public ProblemModell leseDatei(String pfad) throws IOException;
}

```

File - C:\DEV\GroProTest\GroProJava\src\main\java\de\pietsch\interfaces\LoesungsStrategie.java

```java
package de.pietsch.interfaces;

import de.pietsch.model.ProblemModell;

/**
 * Allgemeine Lösungsstrategie.
 */
public interface LoesungsStrategie {
    /**
     * Abstrakte Methode zum Lösen des Beispiel-Problems.
     *
     * @param bspProb: Das zu lösende Problem
     * @return Das aktuelle ProblemModell mit eingetragener Lösung.
     */
    public ProblemModell loeseProblem(ProblemModell bspProb);
}

```

File - C:\DEV\GroProTest\GroProJava\src\main\java\de\pietsch\implementation\BruteForce.java

```java
package de.pietsch.implementation;

import de.pietsch.geometry.Ebene;
import de.pietsch.geometry.Vektor3D;
import de.pietsch.interfaces.LoesungsStrategie;
import de.pietsch.kombinatorikutils.PunktKombinationen;
import de.pietsch.model.ProblemModell;
import de.pietsch.model.SchnittPunktContainer;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Brute-Force-Ansatz zur Lösung der Antennen-Suche. Es wird begonnen mit einer Antenne. Diese wird iterative an allen möglichen Gitterpunkten aufgestellt.
 * Jedes Mal wird überprüft, ob alle Gitterpunkte erreichbar sind. Wenn keine Lösung für eine Antenne gefunden wird, wird die Anzahl erhöht.
 * Es werden alle möglichen Kombinationen von Antennenpositionen ermittelt, und für jede Kombination die vollständige Abdeckung überprüft.
 * Der Algorithmus bricht ab, wenn die erste Lösung gefunden wurde.
 */
public class BruteForce implements LoesungsStrategie {

    ProblemModell bspProb;
    ArrayList<Ebene> ebenen = new ArrayList<>();

    /**
     * Implementierung des Brute-Force-Algorithmus.
     *
     * @param beispielProblem: Das zu lösende Problem
     * @return Das Beispiel-Problem mit der ersten, durch den Brute-Force-Algorithmus gefundenen, Lösung
     */
    @Override
    public ProblemModell loeseProblem(ProblemModell beispielProblem) {
        bspProb = beispielProblem;
        erstelleHilfsEbenen(bspProb);
        PunktKombinationen komb = new PunktKombinationen(bspProb.xDim, bspProb.yDim);
        List<Point[]> punktKombinationen = null;
        for (int antennenAnz = 1; antennenAnz < bspProb.xDim * bspProb.yDim; antennenAnz++) {
            System.out.println("Antenneanzahl: "+ antennenAnz);
            punktKombinationen = komb.findeAllePunktKombinationen(antennenAnz);
            System.out.println("Anzahl Kombinationen: " + punktKombinationen.size());
            for (Point[] kombi : punktKombinationen) {
                int countErreichbar = 0;
                for (int j = 0; j < bspProb.xDim; j++) {
```

```
44                      for (int i = 0; i < bspProb.yDim; i++) {
45                          for (Point antenne : kombi) {
46                              Vektor3D antennenVek = new Vektor3D(
   antenne.x, antenne.y, bspProb.hoehen[antenne.y][antenne.x] + 0.1);
47                              Vektor3D tempPunkt = new Vektor3D(j, i,
   bspProb.hoehen[i][j]);
48                              Vektor3D sendeLinie = antennenVek.
   subtrahiere(tempPunkt);
49                              // Test auf Nachbar
50                              if (sendeLinie.normXundY() < Math.sqrt(2
   ) || punktIstAntenne(kombi, tempPunkt)) {
51                                  countErreichbar++;
52                                  break;
53                              } else {
54                                  boolean istBlockiert = false;
55                                  for (Ebene blockierEbene : ebenen) {
56                                      SchnittPunktContainer sp =
   blockierEbene.findeEbeneGeradeSchnittpunkt(tempPunkt, sendeLinie);
57                                      if (sp.code == 0) {
58                                          double schrittWeite = (sp.
   schnittPunkt.x - tempPunkt.x) / sendeLinie.x;
59                                          double distanzZuSchnittPunkt
    = sp.schnittPunkt.subtrahiere(tempPunkt).normXundY();
60                                          double distanzZuAntenne =
   sendeLinie.normXundY();
61                                          if (schrittWeite > 0 &&
   distanzZuAntenne > distanzZuSchnittPunkt) {
62                                              if (liegtAufGitter(sp.
   schnittPunkt)) {
63                                                  if (
   liegtAufGitterPunkt(sp.schnittPunkt)) {
64                                                      if (sp.
   schnittPunkt.z < bspProb.hoehen[(int) sp.schnittPunkt.y][(int) sp.
   schnittPunkt.x]) {
65                                                          istBlockiert
    = true;
66                                                          break;
67                                                      }
68                                                  } else {
69                                                      if (sp.
   schnittPunkt.x % 1 == 0 && sp.schnittPunkt.y % 1 != 0) {
70                                                          double x1 =
   sp.schnittPunkt.x;
71                                                          double y1 =
   Math.floor(sp.schnittPunkt.y);
72                                                          double y2 =
   Math.ceil(sp.schnittPunkt.y);
73                                                          Vektor3D
   aufpunkt = new Vektor3D(x1, y1, bspProb.hoehen[(int) y1][(int) x1]);
74                                                          Vektor3D
   zielPunkt = new Vektor3D(x1, y2, bspProb.hoehen[(int) y2][(int) x1]);
75                                                          double lambda
    = sp.schnittPunkt.y - Math.floor(sp.schnittPunkt.y);
76                                                          Vektor3D
   schnitt = aufpunkt.addiere(zielPunkt.subtrahiere(aufpunkt).multSkalar
```

```
 76 (lambda));
 77                                                     if (sp.
    schnittPunkt.z < schnitt.z) {
 78
    istBlockiert = true;
 79                                                         break;
 80                                                     }
 81                                                 } else if (sp.
    schnittPunkt.x % 1 != 0 && sp.schnittPunkt.y % 1 == 0) {
 82                                                     double y1 =
    sp.schnittPunkt.y;
 83                                                     double x1 =
    Math.floor(sp.schnittPunkt.x);
 84                                                     double x2 =
    Math.ceil(sp.schnittPunkt.x);
 85                                                     Vektor3D
    aufpunkt = new Vektor3D(x1, y1, bspProb.hoehen[(int) y1][(int) x1]);
 86                                                     Vektor3D
    zielPunkt = new Vektor3D(x2, y1, bspProb.hoehen[(int) y1][(int) x2
    ]);
 87                                                     double
    lambda = sp.schnittPunkt.x - Math.floor(sp.schnittPunkt.x);
 88                                                     Vektor3D
    schnitt = aufpunkt.addiere(zielPunkt.subtrahiere(aufpunkt).
    multSkalar(lambda));
 89                                                     if (sp.
    schnittPunkt.z < schnitt.z) {
 90
    istBlockiert = true;
 91                                                         break;
 92                                                     }
 93                                                 }
 94                                             }
 95                                         }
 96                                     }
 97                                 }
 98                             }
 99                             if (!istBlockiert) {
100                                 countErreichbar++;
101                                 break;
102                             }
103                         }
104
105                     }
106                 }
107             }
108             if (countErreichbar == bspProb.xDim * bspProb.yDim
    ) {
109                 bspProb.antennen = kombi;
110                 bspProb.antennenAnz = kombi.length;
111                 return bspProb;
112             }
113         }
114     }
115     // Alle Punkte haben Antennen
```

```
116         bspProb.antennenAnz = bspProb.xDim * bspProb.yDim;
117         bspProb.antennen = punktKombinationen.get(0);
118         return bspProb;
119     }
120 
121     /**
122      * Überprüfung, ob ein Punkt auf dem zu betrachtenden Feld liegt
    .
123      *
124      * @param punkt
125      * @return
126      */
127     public boolean liegtAufGitter(Vektor3D punkt) {
128         return punkt.x >= 0 && punkt.y >= 0 && punkt.x <= (bspProb.
    xDim - 1) && punkt.y <= (bspProb.yDim - 1);
129     }
130 
131     /**
132      * Überprüfung, ob ein Punkt genau auf einem der Eckpunkte liegt
    . Dafür müssen die x- und y-Komponenten ganzzahlig sein.
133      *
134      * @param punkt
135      * @return
136      */
137     public boolean liegtAufGitterPunkt(Vektor3D punkt) {
138         return (punkt.x % 1 == 0) && (punkt.y % 1 == 0);
139     }
140 
141     /**
142      * Überprüfung, ob ein Punkt eine Antenne ist.
143      *
144      * @param antennen: Array von allen aktuellen Antenne.
145      * @param punkt:    Der zu untersuchende Punkt
146      * @return
147      */
148     public boolean punktIstAntenne(Point[] antennen, Vektor3D punkt
    ) {
149         boolean retVal = false;
150         for (Point antenne : antennen) {
151             if (antenne.x == punkt.x && antenne.y == punkt.y) {
152                 retVal = true;
153                 break;
154             }
155         }
156         return retVal;
157     }
158 
159     /**
160      * Erstellt (x-2) * (y-2) Ebenen. Die Ebenen werden verwendet,
    um zu prüfen, ob die Signale einer Antenne blockiert werden.
161      *
162      * @param bspProb: Das aktuell betrachtete Problem.
163      */
164     private void erstelleHilfsEbenen(ProblemModell bspProb) {
165         for (int j = 1; j < bspProb.xDim - 1; j++) {
```

```
166             Vektor3D aufPunkt = new Vektor3D(j, 0, -1);
167             Vektor3D punkt1 = new Vektor3D(j, 0, bspProb.hoehen[0][j
    ]);
168             Vektor3D punkt2 = new Vektor3D(j, 1, 0);
169             Ebene vertikaleEbene = new Ebene(aufPunkt, punkt1,
    punkt2);
170             ebenen.add(vertikaleEbene);
171         }
172         for (int i = 1; i < bspProb.yDim - 1; i++) {
173             Vektor3D aufPunkt = new Vektor3D(0, i, -1);
174             Vektor3D punkt1 = new Vektor3D(0, i, bspProb.hoehen[i][0
    ]);
175             Vektor3D punkt2 = new Vektor3D(1, i, 0);
176             Ebene horizontaleEbene = new Ebene(aufPunkt, punkt1,
    punkt2);
177             ebenen.add(horizontaleEbene);
178         }
179     }
180
181
182 }
183
```

```java
package de.pietsch.implementation;

import de.pietsch.interfaces.AusgabeStrategie;
import de.pietsch.model.ProblemModell;

import java.awt.*;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.PrintWriter;

/**
 * Klasse zur Ausgabe von gelösten Problemen. Das Ergebnis wird auf der Konsole ausgegeben und in eine Datei geschrieben.
 * Zudem werden Dateien erstellt, um das Ergebnis in GnuPlot zu visualisieren.
 */
public class StandardAusgabe implements AusgabeStrategie {

    ProblemModell bspProb;
    int nummer;
    String outDirectory;

    /**
     * Methode zu Ausgabe des Ergebnisses.
     *
     * @param beispielProblem: Das gelöste Problem.
     * @param beispielNummer:  Die Nummer des Beispiels zur Benennung von Dateien.
     * @throws IOException
     */
    @Override
    public void gebeAus(ProblemModell beispielProblem, int beispielNummer) throws IOException {
        bspProb = beispielProblem;
        nummer = beispielNummer;

        String standardAusgabe = erstelleStandardAusgabe();
        System.out.println(standardAusgabe);
        String beispielProblemName = bspProb.beschreibung.trim().replace(" ", "");

        String currentDirectory = System.getProperty("user.dir");
        outDirectory = currentDirectory + File.separator + "TestAusgaben";
        File directory = new File(outDirectory);
        directory.mkdir();
        outDirectory += File.separator + beispielProblemName;
        directory = new File(outDirectory);
        directory.mkdir();

        PrintWriter pw = new PrintWriter(outDirectory + File.separator + "StandardAusgabe" + nummer + ".txt");
        pw.print(standardAusgabe);
        pw.close();
```

```
49
50          writeGnuPlotFiles();
51
52
53      }
54
55      /**
56       * Erstellt alle Dateien für GnuPlot.
57       *
58       * @throws FileNotFoundException
59       */
60      private void writeGnuPlotFiles() throws FileNotFoundException {
61          PrintWriter pw;
62          String testDatInhalt = erstelleDatInhalt();
63          erstelleAntenneDats();
64          pw = new PrintWriter(outDirectory + File.separator + "Test"
    + nummer + ".dat");
65          pw.print(testDatInhalt);
66          pw.close();
67          String demInhalt = erstelleDemInhalt();
68          pw = new PrintWriter(outDirectory + File.separator + "Test"
    + nummer + ".dem");
69          pw.print(demInhalt);
70          pw.close();
71      }
72
73      /**
74       * Erstellt den String zur Darstellung der Höhen in GnuPlot.
75       *
76       * @return Höhen-String.
77       */
78      private String erstelleDatInhalt() {
79          StringBuilder sb = new StringBuilder();
80          for (int j = 0; j < bspProb.yDim; j++) {
81              for (int i = 0; i < bspProb.xDim; i++) {
82                  sb.append(j).append(" ").append(i).append(" ").
    append(bspProb.hoehen[j][i]).append("\n");
83              }
84              sb.append("\n");
85          }
86          return sb.toString();
87      }
88
89      /**
90       * Erstellt die Dateien für GnuPlot, die die einzelnen Antennen
    beschreiben.
91       *
92       * @throws FileNotFoundException
93       */
94      private void erstelleAntenneDats() throws FileNotFoundException
     {
95          int count = 1;
96          for (Point p : bspProb.antennen) {
97              StringBuilder sb = new StringBuilder();
98              sb.append(p.y).append(" ").append(p.x).append(" ").
```

```
 98 append(bspProb.hoehen[p.y][p.x]).append("\n").append(p.y).append(" "
    ).append(p.x).append(" ").append(bspProb.hoehen[p.y][p.x] + 0.1);
 99             PrintWriter pw = new PrintWriter(outDirectory + File.
    separator + "Test" + nummer + "Antenne" + count + ".dat");
100             pw.write(sb.toString());
101             pw.close();
102             count++;
103         }
104     }
105 
106     /**
107      * Erstellung des Inhalts der .dem-Datei für GnuPlot
108      *
109      * @return Dem-Inhalt-String
110      */
111     private String erstelleDemInhalt() {
112         StringBuilder sb = new StringBuilder();
113         sb.append("#").append(bspProb.beschreibung).append("\n").
    append("\n").append("set title \"").append(bspProb.beschreibung.trim
    ()).append("\"\n").append("set hidden3d\n").append("set key outside
    top\n").append("splot 'Test").append(nummer).append(".dat' w line ,
    \\\n");
114         for (int i = 0; i < bspProb.antennen.length; i++) {
115             sb.append("      'Test").append(nummer).append("Antenne"
    ).append(i + 1).append(".dat' w linesp\n");
116         }
117         sb.append("pause -1 \"Hit return to continue\"");
118         return sb.toString();
119     }
120 
121 
122     /**
123      * Erstellt den Ausgabestring zur darstellung der Lösung.
124      *
125      * @return Lösungsstring
126      */
127     private String erstelleStandardAusgabe() {
128         StringBuilder sb = new StringBuilder();
129         String grenze = "*********************************";
130         sb.append(grenze);
131         sb.append("\n");
132         sb.append(bspProb.beschreibung.trim());
133         sb.append("\n");
134         sb.append(grenze);
135         sb.append("\n");
136         sb.append("Ausdehnung in X: ");
137         sb.append(bspProb.xDim);
138         sb.append("\n");
139         sb.append("Ausdehnung in Y: ");
140         sb.append(bspProb.yDim);
141         sb.append("\n");
142         sb.append("Eingelesene Höhen");
143         sb.append("\n");
144 
145         for (int[] zeile : bspProb.hoehen) {
```

```
146             for (int eintrag : zeile) {
147                 sb.append(eintrag).append(" ");
148             }
149             sb.append("\n");
150         }
151 
152         sb.append("Benötigte Antennen: ");
153         sb.append(bspProb.antennenAnz);
154         sb.append("\n");
155         int count = 1;
156         for (Point p : bspProb.antennen) {
157             sb.append("Antenne ").append(count).append(": ").append(
    p.x).append(" ").append(p.y).append("\n");
158         }
159 
160         return sb.toString();
161     }
162 }
163 
```

File - C:\DEV\GroProTest\GroProJava\src\main\java\de\pietsch\implementation\StandardEingabe.java

```java
package de.pietsch.implementation;

import de.pietsch.interfaces.EingabeStrategie;
import de.pietsch.model.ProblemModell;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.List;

/**
 * Klasse zum Einlesen eines Problems aus einer Datei
 */
public class StandardEingabe implements EingabeStrategie {
    /**
     * Die Methode versucht die angegebene Datei zu öffnen und auszulesen. Es wird ein neues ProblemModell-Objekt erstellt und Befüllt.
     * Dabei werden diverse Vorgaben überprüft.
     * @param pfad: Dateipfad
     * @return ProblemModell: Da zu lösende Problem
     * @throws IOException
     */
    @Override
    public ProblemModell leseDatei(String pfad) throws IOException {
        ProblemModell pm = new ProblemModell();

        List<String> allLines = Files.readAllLines(Paths.get(pfad));

        if (allLines.size() < 6) {
            throw new IOException("Die Eingabedatei genuegt nicht dem angeforderten Format. Es sind nicht genug Zeilen vorhanden.");
        } else if(allLines.size() < 8){
            throw new IOException("Zu wenig Hoehenangaben vorhanden. Das Feld sollte mindestens eine 2x2 Hoehen beinhalten.");
        }

        String line1 = allLines.get(1);
        if (line1.startsWith(";") && line1.trim().length() > 1){
            pm.beschreibung = line1.split(";")[1];
        } else {
            throw new IOException("Ungueltige Beschreibungszeile (Z.1). Die Beschreibung muss mit einem Semikolon beginnen und darf nicht leer sein.");
        }

        String[] gebietDim = allLines.get(4).split(" ");
        if (gebietDim.length != 2) {
            throw new IOException("Fehler bei der Groesse des Gebiets in Zeile 5");
        }


        pm.xDim = Integer.parseInt(gebietDim[0]);
        pm.yDim = allLines.size() - 6;
```

```
49          pm.hoehen = new int[pm.yDim][pm.xDim];
50          if(pm.xDim < 2 || pm.yDim < 2){
51              throw new IllegalArgumentException("Die Werte für die
    Dimension des Höhenfeldes sind zu gering. Das Feld sollte mindestens
     eine 2x2 Hoehen beinhalten");
52          }
53
54          int i = 0;
55          for (int zeilenIdx = 6; zeilenIdx < allLines.size();
    zeilenIdx++) {
56              String zeile = allLines.get(zeilenIdx);
57              if (!Character.isDigit(zeile.charAt(0))) {
58                  throw new IllegalArgumentException("Ungueltige
    Hoehenangabe in Zeile" + (zeile + 1) + " der Eingabedatei.");
59              }
60              String[] zeilenHoehen = zeile.split(" ");
61              if (zeilenHoehen.length > pm.xDim) {
62                  throw new IOException("Es sind mehr Eintraege in
    Zeile " + (zeilenIdx + 1) + " als es die x-Dimension erlaubt!");
63              }
64              for (int j = 0; j < pm.xDim; j++) {
65                  if (j < zeilenHoehen.length) {
66                      int hoehe = Integer.parseInt(zeilenHoehen[j]);
67                      if (hoehe < 0 || hoehe > 6) {
68                          throw new IllegalArgumentException("
    Hoehenangaben muessen im Intervall [0, 6] liegen. Fehler bei Eintrag
     (" + i + "|" + j + ")!");
69                      }
70                      pm.hoehen[i][j] = Integer.parseInt(zeilenHoehen[
    j]);
71                  } else {
72                      pm.hoehen[i][j] = pm.hoehen[i][zeilenHoehen.
    length - 1];
73                  }
74              }
75              i++;
76          }
77
78          return pm;
79      }
80  }
81
82
```

```java
package de.pietsch.kombinatorikutils;

import java.awt.*;
import java.util.ArrayList;
import java.util.List;

/**
 * Klasse zum Generieren von Kombinationen aus Punkten. Es handelt sich um eine Kombination ohne Wiederholung.
 * Quelle: https://www.baeldung.com/java-combinations-algorithm
 */
public class PunktKombinationen {

    private final ArrayList<Point> standardPunkte;
    private final int x;
    private final int y;

    public PunktKombinationen(int x, int y) {
        standardPunkte = new ArrayList<>();
        this.x = x;
        this.y = y;
        generateStandardPoints();
    }

    /**
     * Hilfs-Methode zum Generieren der Kombinationen.
     *
     * @param kombinationen: Liste der bisher gefundenen Kombinationen
     * @param data:          aktuell zu bearbeitende Kombination
     * @param start
     * @param end
     * @param index:         index zum Füllen von Data
     */
    private void rekursion(List<Point[]> kombinationen, Point[] data, int start, int end, int index) {
        if (index == data.length) {
            Point[] kombination = data.clone();
            kombinationen.add(kombination);
        } else if (start <= end) {
            data[index] = standardPunkte.get(start);
            rekursion(kombinationen, data, start + 1, end, index + 1);
            rekursion(kombinationen, data, start + 1, end, index);
        }
    }

    /**
     * Findet alle möglichen Antennen-Positionen-Kombinationen.
     *
     * @param r: Anzahl der Antenne, die plaziert werden soll.
     * @return Es wird eine Liste von Punkt-Kombinationen zurückgegeben. Die Punkt-Kombinationen enthalten r Punkte.
     */
    public List<Point[]> findeAllePunktKombinationen(int r) {
```

```
51          int n = x * y;
52          List<Point[]> kombinationen = new ArrayList<>();
53          rekursion(kombinationen, new Point[r], 0, n - 1, 0);
54          return kombinationen;
55      }
56
57      /**
58       * Generiert eine Liste von allen Gitterpunkten des aktuellen
   Problems
59       */
60      private void generateStandardPoints() {
61          for (int i = 0; i < x; i++) {
62              for (int j = 0; j < y; j++) {
63                  standardPunkte.add(new Point(i, j));
64              }
65          }
66      }
67  }
68
```