

TITEL	Leitende Aufgabenstellung Düsen-Passagierflugzeug Abgabedokumentation Teilaufgabe Softwareengineering
ERSTELLER	Marc Landolt, Pascal Jenni, Romino Florio
DATUM	26. November 2007
VERSION	0.0.4

Leitende Aufgabenstellung Düsen-Passagierflugzeug

Abgabedokumentation Teilaufgabe Softwareengineering

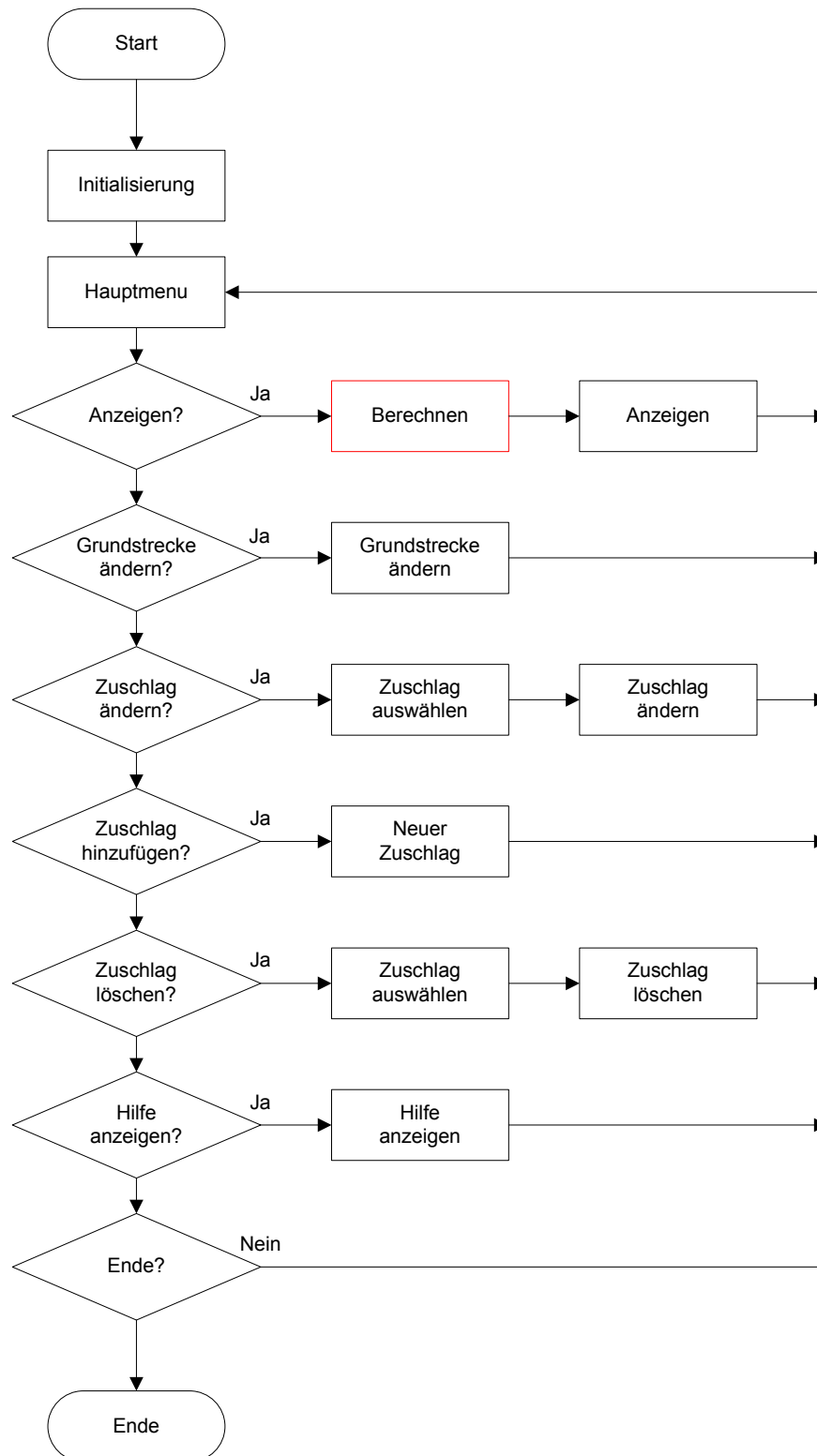
INHALTSVERZEICHNIS

1	FLUSSDIAGRAMM	3
1.1	ÜBERSICHT	3
1.2	DETAIL BERECHNUNG	4
1.2.1	Summe	4
1.2.2	Zuschlag	4
2	KLASSENDIAGRAMM	5
3	SOURCECODE	6
3.1	SOURCECODE AUF CD	6
3.2	SOURCECODE KLASSEN	7
3.2.1	Calculation	7
3.2.2	Help	10
3.2.3	Interface	10
3.2.4	Main	13
3.2.5	Menu	14
3.2.6	Tools	18
3.2.7	Tuple	21
4	PRINTSCREEN	23
5	BERICHT	24
5.1	TESTBERICHT	25
5.2	ERFAHRUNGEN	26
5.3	ERKENNTNISSE	26
5.4	STOLPERSTEINE	26

1 FLUSSDIAGRAMM

1.1 ÜBERSICHT

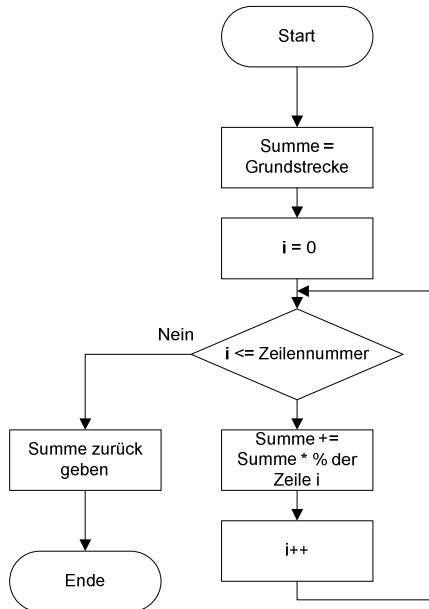
Die Berechnungen sind im Kapitel 1.2 detailliert dargestellt



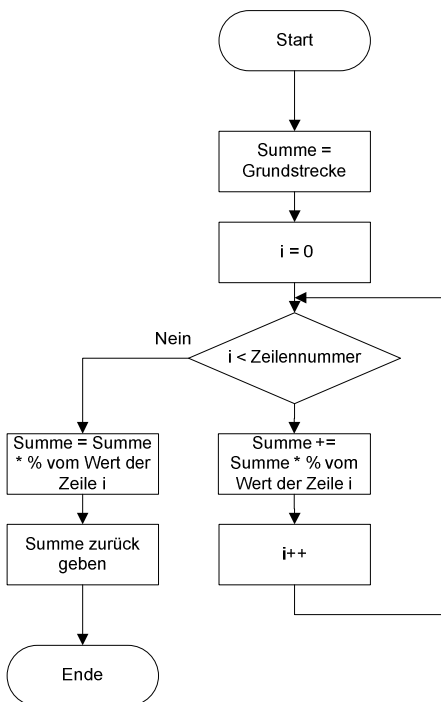
1.2 DETAIL BERECHNUNG

1.2.1 Summe

sum(int pNumber)



1.2.2 Zuschlag



2 KLASSENDIAGRAMM

3 SOURCECODE

Der Sourcecode teilt sich in die folgenden Klassen auf, welche in den Kapiteln ab 3.2 separat aufgeführt sind.

- Calculation
- Help
- ICalculation
- Main
- Menu
- Tools
- Tuple

3.1 SOURCECODE AUF CD

CD in Hülle aufgeklebt (Hülle vorhanden)

3.2 SOURCECODE KLASSEN

3.2.1 Calculation

```
/*
 * Calculation.java
 * Created on 21. Oktober 2007, 10:34
 */

package la;

/**
 * In dieser Klasse wird die Berechnung durchgeführt. Zusätzlich enthält sie die Grundstrecke und die Datensätze vom Typ Tuple. Um die Daten zu sichern könnte man diese Klasse z.B. in
eine XML-Datei Serialisieren
 * @author Marc Landolt, Romino Florio, Pascal Jenni 48 3/i
 */
public class Calculation implements ICalculation{

    private final boolean DEBUG=false;

    private java.util.ArrayList<Tuple> list = new java.util.ArrayList<Tuple>();
    //private Tuple[] tuples; // eintrag damit ArgoUML die Linie direkt macht da es sich schwer tut mit der typisierten Klasse
    ArrayList
    private double mInitDistance=100;

    /**
     * Konstruktor der die vordefinierten Zuschläge initialisiert und readOnly setzt
     */
    public Calculation() {

        String[] name = { "Graspiste", "Rückenwind", "Pistenzustand", "Sicherheitsfaktor"}; //liesse sich als vorlagendatei speichern (eigene Klasse)
        double[] value = { 20.0, 10.0, 5.0, 33.0 };
        boolean[] ReadOnly = { true, true, true, true };

        for(int i = 0; i < name.length; i++)
            list.add(new Tuple(name[i], value[i], ReadOnly[i])); // Datensätze mit den Grundwerten initalisieren
    } // end method

    /**
     * Methode um die Grundstrecke zu setzen
     * @param pInitDistance Wert der Grundstrecke als Double
     * @return true wenn ein positiver Wert angegeben wurde
     */
    public boolean setInitDistance(double pInitDistance) {
        if(pInitDistance<0) return false;
        this.mInitDistance=pInitDistance;
        return true; // gibt true zurück, wenn ein gültiger (positiver) Wert eingegeben wurde
    } // end method

    /**
     * Methode um die Grundstrecke zurückzugeben
     * @return Die Grundstrecke als double
     */
    public double getInitDistance() {
        return this.mInitDistance;
    } // end method

    /**
     * Methode um einen neuen Datensatz hinzuzufügen
     * @param pName Die Bezeichnung dieses Datensatzes
     * @param pValue Der Wert in % dieses Datensatzes
     * @param pReadOnly Schreibschutz, so dass nur der Wert nicht aber die Bezeichnung geändert werden kann
     */
}
```

```

    * @return Gibt true zurück falls der Wert nicht negativ ist
    */
    public boolean addTuple(String pName, double pValue, boolean pReadOnly) {
        if (pValue<0) return false; // wird ein negativer Wert eingegeben, wird abgebrochen und false zurückgegeben
        list.add(new Tuple(pName, pValue, pReadOnly));
        return true;
    } // end method

    /**
     * Methode um den "pNumber" Datensatz zu löschen
     * @param pNumber Gibt die zu löschende Zeile an, falls -1 angegeben wird, wird abgebrochen und dennoch true zurückgegeben
     * @return ist der Wert von pNumber gültig oder -1 wird true zurückgegeben, ansonsten false
     */
    public boolean removeTuple(int pNumber) {
        if(pNumber>list.size()) return false;
        if(pNumber==-1) return true; // bei -1 wird nichts gelöscht
        if(list.get(pNumber).mReadOnly==true) return false;

        list.remove(pNumber);
        return true;
    } // end method

    /**
     * Methode um eine Referenz auf den Datensatz der Zeile "pNumber" zu erstellen
     * @param pNumber Die Nummer des Datensatzes
     * @return die Referenz zum ausgewählten Tuple
     * @see Siehe Tuple
     */
    public Tuple getTuple(int pNumber) // getter für Datensatz
    {
        if(list.size()<pNumber || pNumber<0) return null;
        else return list.get(pNumber);
    } // end method

    /**
     * Methode um die Grösse der Liste auszugeben
     * @return die Grösse der Liste als int
     */
    public int size() {
        return list.size();
    } // end method

    /**
     * Methode um die Summe für den gewählten Datensatz "pNumber" anhand der Grundstrecke und den vorhergehenden Zuschlägen zu berechnen
     * @param pNumber Gibt an bis zu welcher Zeile die Summe berechnet werden soll. Ist die Zahl ausserhalb der Datensätze wird Double.NaN zurückgeben
     * @return die Summe bis zur gewählten Zeile als double
     */
    public double sum(int pNumber) {
        if(pNumber>=size() || pNumber<0) return Double.NaN; // ist der Wert ausserhalb der Datensätze wird Double.NaN zurückgegeben

        double sum=mInitDistance; // mit der Grundstrecke beginnen

        for(int i = 0; i <= pNumber; i++) {
            sum+=sum*list.get(i).getValue()/100; // hier wird sukzessive jede Zeile prozentual dazugezählt
        }
        return sum;
    } // end method

    /**
     * Methode um den Zuschlag in Meter für den gewählten Datensatz "pNumber" anhand der Grundstrecke und den vorhergehenden Zuschlägen zu berechnen
     * @param pNumber Die gewünschte Zeile
     * @return der Zuschlag in Meter als double
     */
    public double supplementInMeter(int pNumber) { // sonst wie oben, ausser das nur die Teilsum angegeben wird
        if(pNumber>=size() || pNumber<0) return Double.NaN; // ist der Wert ausserhalb der Datensätze wird Double.NaN zurückgegeben

        double sum=mInitDistance; // mit der Grundstrecke beginnen

```

```
for(int i = 0; i < pNumber; i++) {
    sum+=sum*list.get(i).getValue()/100;
}

if(DEBUG) System.out.format("%10.2f<", sum);
// hier wird sukzessive jede Zeile prozentual dazugezählt
// dies ist keine Ausgabe im eigentlichen Sinne, es zeigt lediglich temporär Zwischenwerte an. Wird mit
der variabel "debug" zu Beginn ausgeschaltet

sum=sum*list.get(pNumber).getValue()/100;

return sum;
} // end method
} // end class
```

3.2.2 Help

```
/*
 * Help.java
 * Created on 23. November 2007, 21:13
 */

package la;

/**
 * Diese Klasse ist zuständig für die Ausgabe des Hilfetextes
 * @author Marc Landolt, Romino Florio, Pascal Jenni 48 3/i
 */
public class Help {

    /**
     * String Array mit dem kompletten Hilfetext
     */
    protected static String[] Text = { "Hilfe",
        "-----",
        "Mit dieser Software kann die Startstrecke eines Flugzeuges anhand der Grundstrecke aus dem Flughandbuch",
        "und Ihren individuellen Zuschlägen berechnet werden. Untenstehend werden die einzelnen Hauptmenüpunkte näher erläutert.",
        "",
        "Punkt 1: (Berechnung anzeigen) Hier ist die Komplette Berechnung ersichtlich.",
        "    Wenn noch keine Anpassungen vorgenommen wurden, wird diese anhand von Standardwerten erstellt.",
        "Punkt 2: (Grundstrecke ändern) Geben Sie hier die Grundstrecke ihres Flugzeuges aus dem Flughandbuch ein.",
        "Punkt 3: (Zuschläge ändern) Hier können Sie den prozentualen Anteil der vordefinierten Zuschläge editieren.",
        "    Die Bezeichnung (Graspiste, Rückenwind, Pistenzustand Sicherheitsfaktor) sind nicht editierbar.",
        "    Bereits erstellte freidefinierbare Zuschläge via Menüpunkt 4 können hier jedoch komplett editiert werden.",
        "Punkt 4: (Frei wählbarer Zuschlag hinzufügen) Hier können Sie individuelle Zuschläge erfassen.",
        "Punkt 5: (Frei wählbarer Zuschlag löschen) Hier können Sie die unter Menüpunkt 4 erstellten Zuschläge löschen.",
        "    Alle vordefinierten Zuschläge sind mit einem * gekennzeichnet und können nicht gelöscht werden.",
        "Punkt 9: (Hilfe) Hier gelangen Sie zu dieser Hilfe.",
        "Punkt 0: (Ende) Hiermit beenden Sie das Programm.",
        "",
        "Copyright (c) by Marc Landolt, Romino Florio, Pascal Jenni"};

    /**
     * Methode um die Hilfe am Bildschirm anzuzeigen
     * @param screenHeight in Abhängigkeit der Bildschirmhöhe werden genau soviele Zeilen angezeigt und dann auf Eingabe gewartet
     */
    public static void printHelp(int screenHeight) {
        Tools.clearScreen();
        for(int i=0; i<Text.length; i++) {
            System.out.println(Text[i]);
            if(i%(screenHeight+1)==screenHeight) Tools.pause(false); // ist der Bildschirm zu klein für die komplette Hilfe, werden mehrere Teilstücke angezeigt
        } // bei 9 Zeilen ist i%10 genau jedes 9. mal gleich 9
        Tools.pause(true);
    } // end method
} // end class
```

3.2.3 ICalculation

```
/*
 * ICalculation.java
 * Created on 21. Oktober 2007, 10:34
 */

package la;

/**
 * Interface für die Berechnungsklasse damit Sie austauschbar ist
 * @author Marc Landolt, Romino Florio, Pascal Jenni 48 3/i
 */
public interface ICalculation {

    /**
     * Methode um die Grundstrecke zu setzen
     * @param pInitDistance Wert der Grundstrecke als Double
     * @return true wenn ein positiver Wert angegeben wurde
     */
    public boolean setInitDistance(double pInitDistance);

    /**
     * Methode um die Grundstrecke zurückzugeben
     * @return Die Grundstrecke als double
     */
    public double getInitDistance();

    /**
     * Methode um einen neuen Datensatz hinzuzufügen
     * @param pName Die Bezeichnung dieses Datensatzes
     * @param pValue Der Wert in % dieses Datensatzes
     * @param pReadOnly Schreibschutz, so dass nur der Wert nicht aber die Bezeichnung geändert werden kann
     * @return Gibt true zurück falls der Wert nicht negativ ist
     */
    public boolean addTuple(String pName, double pValue, boolean pReadOnly);

    /**
     * Methode um den "pNumber" Datensatz zu löschen
     * @param pNumber Gibt die zu löschende Zeile an, falls -1 angegeben wird, wird abgebrochen und dennoch true zurückgegeben
     * @return ist der Wert von pNumber gültig oder -1 wird true zurückgegeben, ansonsten false
     */
    public boolean removeTuple(int pNumber);

    /**
     * Methode um eine Referenz auf den Datensatz der Zeile "pNumber" zu erstellen
     * @param pNumber Die Nummer des Datensatzes
     * @return die Referenz zum ausgewählten Tuple
     * @see Siehe Tuple
     */
    public Tuple getTuple(int pNumber);

    /**
     * Methode um die Grösse der Liste auszugeben
     * @return die Grösse der Liste als int
     */
    public int size();

    /**
     * Methode um die Summe für den gewählten Datensatz "pNumber" anhand der Grundstrecke und den vorhergehenden Zuschlägen zu berechnen
     * @param pNumber Gibt an bis zu welcher Zeile die Summe berechnet werden soll. Ist die Zahl ausserhalb der Datensätze wird Double.NaN zurückgegeben
     * @return die Summe bis zur gewählten Zeile als double
     */
    public double sum(int pNumber);
}
```

```
/**
 * Methode um den Zuschlag in Meter für den gewählten Datensatz "pNumber" anhand der Grundstrecke und den vorhergehenden Zuschlägen zu berechnen
 * @param pNumber Die gewünschte Zeile
 * @return der Zuschlag in Meter als double
 */
public double supplementInMeter(int pNumber);
} // end interface
```

3.2.4 Main

```
/*
 * Main.java
 * Created on 21. Oktober 2007, 10:06
 */

package la;

/**
 * Main Klasse die zum starten des Programmes benötigt wird
 * @author Marc Landolt, Romino Florio, Pascal Jenni 48 3/i
 */
public class Main {

    /**
     * Main Routine
     * @param args die übergebenen Argumente
     */
    public static void main(String[] args)
    {
        Tools.setScreenheight();           // setzen der Fensterhöhe
        new la.Menu();                     // Menu starten
    } // end method

} // end class
```

3.2.5 Menu

```
/*
 * Menu.java
 * Created on 21. Oktober 2007, 12:39
 */

package la;

import la.Tools;

/**
 * Diese Klasse ist die Benutzerschnittstelle
 * @author Marc Landolt, Romino Florio, Pascal Jenni 48 3/i
 */
public class Menu {

    ICalculation CC = new Calculation(); // hier wird die Berechnungsklasse (CalculationClass) initialisiert, welche die Berechnung übernimmt

    private static final boolean DEBUG=false; // falls true, werden zusätzliche Informationen während dem debuggen ausgegeben. Final, damit der Compiler
    im Falle false entsprechende Zeilen gar nicht kompiliert

    /**
     * Konstruktor der das Hauptmenu mainMenu() aufruft
     */
    public Menu() {
        mainMenu();
    } // end method

    /**
     * Methode um das Hauptmenu zu erzeugen
     */
    public void mainMenu() {
        boolean exitSignal=false;

        while(!exitSignal) // wiederholen bis beenden gewählt wird, exitSignal=true
        {
            Tools.clearScreen(); // den Bildschirm löschen

            System.out.println("Hauptmenu"); // Hauptmenu anzeigen
            System.out.println("-----");
            System.out.println();
            System.out.println("1: Berechnung anzeigen");
            System.out.println("2: Grundstrecke ändern");
            System.out.println("3: Zuschläge ändern");
            System.out.println("4: Frei wählbarer Zuschlag hinzufügen");
            System.out.println("5: Frei wählbarer Zuschlag löschen");
            System.out.println("9: Hilfe");
            System.out.println("0: Ende");

            Tools.feedScreen(Tools.sScreenheight-9); // Leerzeilen einfügen bis der Titel "Hauptmenu" zuoberst steht. Dies geschieht in Abhängigkeit des
            Tools.sScreenheight

            switch(Tools.intInput(true)) // Integerzahl einlesen und mit Switch auswerten
            {
                // 1: Anzeigen
                case 1: Tools.clearScreen(); // Bildschirm löschen
                    display(); // die Berechnungstabelle ausgeben
                    Tools.feedScreen(Tools.sScreenheight-CC.size()-3); // Leerzeilen einfügen bis die Liste zuoberst steht
                    Tools.pause(true); // pause damit die Liste sichtbar bleibt
                    break;

                // 2: Grundstrecke ändern
                case 2: changeInitDistance(); // Grundstrecke ändern starten. Details bei Methode weiter unten
            }
        }
    }
}
```

```

        break;

        // 3: Zuschläge ändern
        case 3: selectTupleToChange(); // Datensatz ändern starten. Details bei Methode weiter unten
        break;

        // 4: Frei wählbarer Zuschlag hinzufügen
        case 4: addRecord(); // Datensatz hinzufügen starten. Details bei Methode weiter unten
        break;

        // 5: Frei wählbarer Zuschlag löschen
        case 5: removeRecord(); // Datensatz löschen starten. Details bei Methode weiter unten
        break;

        // 9: Hilfe
        case 9: Help.printHelp(Tools.sScreenheight); // Hilfe Starten. Details in der Help-Klasse
        break;

        // 0: Ende
        case 0: exitSignal=true; // Beenden mit exitSignal=true
        break;

        default:
            System.out.println("Menupunkt nicht vorhanden."); // Fehlermeldung bei unbekanntem Menupunkt. Hier könnte man eine Pause einbauen, doch dies würde es träger
    } // end switch
} // end while
} // end method

/**
 * Methode um die aktuelle Berechnung in einer tabellarisch Liste auszugeben
 */
public void display() {
    System.out.format(" %-30s %10.1f", "Grundstrecke", CC.getInitDistance());
    System.out.println();
    for(int i = 0; i < CC.size(); i++) {
        System.out.format("%2d ", i+1);
        System.out.format("%-30s", CC.getTuple(i).getName());
        System.out.format("%9.1f", CC.getTuple(i).getValue()); System.out.print("%");
        System.out.format("%10.1f", CC.supplementInMeter(i)); // Zuschlag in Meter
        if(CC.getTuple(i).mReadOnly) System.out.print(" *");
        else System.out.print(" ");
        if(DEBUG) System.out.format("%10.1f", CC.sum(i)); // zeigt für das Verständnis die Summer bis zur aktuellen Zeile an
        System.out.println();
    } // end for
    System.out.format(" %-30s %10.1f", "Benötigte Strecke", CC.sum(CC.size()-1));
    System.out.println();
} // end method

/**
 * Methode um die Grundstrecke zu ändern
 */
public void changeInitDistance() {
    double eingabe=-1;
    Tools.clearScreen(); // Bildschirm löschen
    while(true) {
        System.out.println("Grundstrecke ändern [" + CC.getInitDistance() + "] (X = Wert behalten):");
        eingabe = Tools.doubleInput(true,CC.getInitDistance()); // Parameter true bedeutet, dass die Eingabe positiv sein muss. Das Zweite ist der Ursprungswert der
        wieder zurückgeschrieben wird falls x gedrückt wird
        if(CC.setInitDistance(eingabe)) break;
        System.out.println("Fehleingabe"); // Fehlermeldung bei falscher Eingabe
    } // end while
} // end method

/**
 * Methode um einen Datensatz zu löschen der nicht vordefiniert ist
 */

```

```

public void removeRecord() {
    while(true) {
        Tools.clearScreen(); // Bildschirm löschen
        display(); // die Berechnungstabelle ausgeben
        Tools.feedScreen(Tools.sScreenheight-CC.size()-3); // Leerzeilen einfügen bis die Liste zuoberst steht
        System.out.println("Welchen Datensatz möchten Sie löschen (0 = Zurück) :");
        int tuple=Tools.intInput(true);

        if(tuple<CC.size()+1) // prüft ob der Datensatz voranden ist
        {
            if(CC.removeTuple(tuple-1)) break; // versuchen zu löschen. Falls der Datensatz Fix also readOnly ist, Fehlermeldung ausgeben
        } else {
            if (tuple==0) break; // wird Null eingegeben, wird das Löschen abgebrochen
        }
        System.out.println("Sie versuchen einen falschen Datensatz zu löschen");
        Tools.pause(true);
    } // end while
} // end method

/**
 * Methode um einen Datensatz hinzuzufügen
 */
public void addRecord() {
    while(true) {
        Tools.clearScreen(); // Bildschirm löschen
        display(); // die Berechnungstabelle ausgeben
        Tools.feedScreen(Tools.sScreenheight-CC.size()-3); // Leerzeilen einfügen bis die Liste zuoberst steht
        CC.addTuple("Neuer Zuschlag", 0, false); // einen leeren tuple hinzufügen
        changeTuple(CC.size()); // den leeren tuple Tuple editieren (siehe unten: tupleAendern)
        System.out.println("Einen weiteren Datensatz hinzufügen? (j/n)");
        String next=Tools.stringInput();
        if(next.charAt(0)!='j' && next.charAt(0)!='J') break; // hier wird DeMorgan berücksichtigt
    } // end while
} // end method

/**
 * Methode um einen Datensatz auszuwählen für die Änderung
 */
public void selectTupleToChange() {
    while(true) {
        Tools.clearScreen(); // Bildschirm löschen
        display(); // die Berechnungstabelle ausgeben
        Tools.feedScreen(Tools.sScreenheight-CC.size()-3); // Leerzeilen einfügen bis die Liste zuoberst steht
        System.out.println("Welchen Datensatz möchten Sie ändern (0 = Zurück) :");
        int tuple=Tools.intInput(true);
        if (DEBUG) System.out.println("tuple: " + tuple);
        if(tuple<CC.size()+1) // den ausgewählten Datensatz ändern
        {
            if(changeTuple(tuple)) break; // falls erfolgreich, Schleife beenden
        } // sonst erneuter Versuch
        else {
            System.out.println("Falscher Datensatz!"); // Fehlermeldung ausgeben
            break;
        }
    } // end while
} // end method

/**
 * Methode um den gewählten Datensatz zu ändern. Ist er vordefiniert, wird nur der Wert verändert. Bei allen Anderen kann auch der Name geändert werden
 * @param pTuple ID des Datensatzes der geändert werden soll
 * @return gibt true zurück, wenn die Editierung erfolgreich war
 */
public boolean changeTuple(int pTuple) {
    if (pTuple==0) return true;
    pTuple--; // minus 1 weil der Mensch bei 1 und nicht bei Null zu zählen beginnt

    if (!CC.getTuple(pTuple).mReadOnly) {

```

```

        System.out.println("Bitte neue Bezeichnung angeben [" + CC.getTuple(pTuple).getName() + "]:"); // allenfalls leere Eingabe nicht akzeptieren
        CC.getTuple(pTuple).setName(Tools.stringInput());
    } else System.out.println("Bei diesem Datensatz darf nur der Wert des Zuschlags verändert werden, nicht aber dessen Name");

    System.out.println("Bitte neuen Wert von '" + CC.getTuple(pTuple).getName() + "' angeben [" + CC.getTuple(pTuple).getValue() + "%] (X = Wert behalten): ");

    //
    //          ABFRAGE          Ursprünglicher wert
    while( !CC.getTuple(pTuple).setValue( Tools.doubleInput( true, CC.getTuple(pTuple).getValue() ) ) ) // hier eine kleine Knacknuss... wir übergeben den
    Originalwert und im Falle der Eingabe eines x, wird der alte Wert wieder eingetragen
    {
        System.out.println();
    } // negativer Wert zulassen?
    return true;
} // end method
} // end class

```

3.2.6 Tools

```
/*
 * Tools.java
 * Created on 24. Oktober 2007, 19:15
 */

package la;

/**
 * Diese Klasse enthält die Hilfsroutinen
 * @author Marc Landolt, Romino Florio, Pascal Jenni 48 3/i
 */
public class Tools {

    private static final boolean DEBUG=false;
    private static java.util.Scanner myScanner = new java.util.Scanner(System.in); // wichtig ist, dass immer die ganze Linie abgefragt wird. Sonst bleiben noch irgendwelche
    Zeichenketten und Linefeeds im Scanner hängen, die einem das Leben schwer machen
    /**
     * Ermittelte Bildschirmhöhe
     */
    protected static int sScreenheight=10; // aufgrund dieser Variabel werden die Seitenvorschübe gemacht damit der Inhalt richtig
    ausgerichtet werden kann

    /**
     * Methode um durch Benutzereingabe die statische Variabel sScreenheight, für die Höhe des Terminals zu setzen
     */
    public static void setScreenheight()
    {
        for(int i = 100; i>0; i--) System.out.println(i);
        System.out.println("Um die Höhe ihres Terminals zu ermitteln geben Sie bitte die oberste sichtbare Zahl an (mind. 10)");
        sScreenheight = intInput(true);
    } // end method

    /**
     * Methode um einen String abzufragen und diesen zurückzugeben. So benötigen wir den Scanner nicht in jeder Klasse
     * @return die eingegebene Zeichenkette als String
     */
    public static String stringInput()
    {
        String string = myScanner.nextLine();
        return string;
    } // end method

    /**
     * Methode um einen Integer abzufragen und nach der Gültigkeitsprüfung zurückzugeben
     * @param mustBePositive wird hier true angegeben, wird solange wieder abgefragt, bis die Zahl positiv ist
     * @return die eingegebene Zahl als int
     */
    public static int intInput(boolean mustBePositive)
    {
        String string;
        int number;

        while(true) // diese Schleife wird solange wiederholt, bis ein korrekter Wert eingegeben wurde
        {
            try
            {
                string = myScanner.nextLine();
                number = Integer.parseInt(string);
            }
            catch(java.lang.Exception nfe)
            {
                System.out.println("    >>> Bitte nur natürliche Zahlen eingeben. z.B. '3' <<<");
                continue;
            }
        }
    }
}
```

```

    }
    if(mustBePositive&&number<0)
    {
        System.out.println("    >>>    Bitte nur positive Zahlen eingeben.    <<<");
        continue;
    }
    return number;
} // end while
} // end method

/**
 * Methode um einen double abzufragen und nach der Gültigkeitsprüfung zurückzugeben
 * @param pMustBePositive wird hier true angegeben wird solange wieder abgefragt bis die Zahl positiv ist
 * @return die eingegebene Zahl als double
 */
public static double doubleInput(boolean pMustBePositive)
{
    return doubleInput(pMustBePositive, 0);
}

/**
 * Methode um einen double abzufragen und nach der Gültigkeitsprüfung zurückzugeben. pOriginalValue ist der ursprüngliche Wert der im Falle des drückens von X verwendet wird
 * @param pMustBePositive wird hier true angegeben, wird solange wieder abgefragt, bis die Zahl positiv ist
 * @param pOriginalValue der ursprüngliche Wert der im Falle des drückens von X verwendet wird
 * @return die eingegebene Zahl als double
 */
public static double doubleInput(boolean pMustBePositive, double pOriginalValue)
{
    String string;
    double doubleNumber;

    while(true)
    {
        try
        {
            string = myScanner.nextLine();
            if(DEBUG) System.out.println(""+string+"");
            if(string.length()==0) continue; // verhindert Absturz bei Leereingabe
            if(string.charAt(0)=='x' || string.charAt(0)=='X') return pOriginalValue; // wurde ein X eingegeben, wird der Originalwert "pOriginalValue" zurückgegeben
            doubleNumber = Double.parseDouble(string);
        }
        catch(java.lang.NumberFormatException nfe)
        {
            System.out.println("    >>>Bitte nur rationale Zahlen eingeben. z.B. '3.53'<<<");
            continue;
        }
        if(pMustBePositive && doubleNumber < 0)
        {
            System.out.println("    >>>    Bitte nur positive Zahlen eingeben.    <<<");
            continue;
        }
    }
    return doubleNumber;
} // end while
} // end method

/**
 * Methode um den ganzen Bildschirm zu löschen
 */
public static void clearScreen()
{
    for(int i=0; i<screenheight; i++)
    {
        System.out.println();
        try { java.lang.Thread.sleep(10); } // just for fun...
        catch ( java.lang.InterruptedExceotion ie)
        { System.out.println("Threading error, YES you did something completely strange!"); }
    } // end for
}

```

```

} // end method

/**
 * Methode um den Inhalt an die richtige Position zu schieben
 * @param pLines um soviel Linien wird verschoben
 */
public static void feedScreen(int pLines)
{
    for(int i=0; i<pLines; i++)
        System.out.println();
} // end method

/**
 * Methode für Pause (wartet auf eingabe)
 * @param message wird hier true angegeben, wird dem Benutzer eine Meldung angezeigt, er solle Enter drücken
 */
public static void pause(boolean message)
{
    if(message) System.out.println("    >>> Bitte Enter-Taste drücken um zurück zu kehren <<<");
    stringInput();
} // end method

} // end class

```

3.2.7 Tuple

```
/*
 * Tuple.java
 * Created on 21. Oktober 2007, 10:08
 */

package la;

/**
 * In dieser Klasse werden die einzelnen Datensätze gespeichert
 * @author Marc Landolt, Romino Florio, Pascal Jenni 48 3/i
 */
public class Tuple {

    /**
     * Beschreibung des Zuschlages
     */
    private String mName;
    /**
     * Wert in % dieses Zuschlages
     */
    private double mValue;
    /**
     * Schreibschutz für den Namen des Datensatzes damit er nicht verändert werden kann
     */
    protected boolean mReadOnly;

    /**
     * Konstruktor
     * @param pName Name des Zuschlags
     * @param pValue Wert des Zuschlags
     * @param pReadOnly Schreibschutz für den Namen des Datensatzes
     */
    public Tuple(String pName, double pValue, boolean pReadOnly)
    {
        this.mReadOnly=pReadOnly;
        this.mName=pName;
        this.mValue=pValue;
    } // end method

    /**
     * Methode um die Bezeichnung des Datensatzes zu setzen
     * @param pName Bezeichnung des Datensatzes
     * @return gibt true Zurück nachdem der Name geändert wurde wenn kein Schreibgeschützt gesetzt ist
     */
    public boolean setName(String pName)
    {
        if (this.mReadOnly) return false; // ist der Datensatz ReadOnly wird abgebrochen
        this.mName=pName;
        return true;
    } // end method

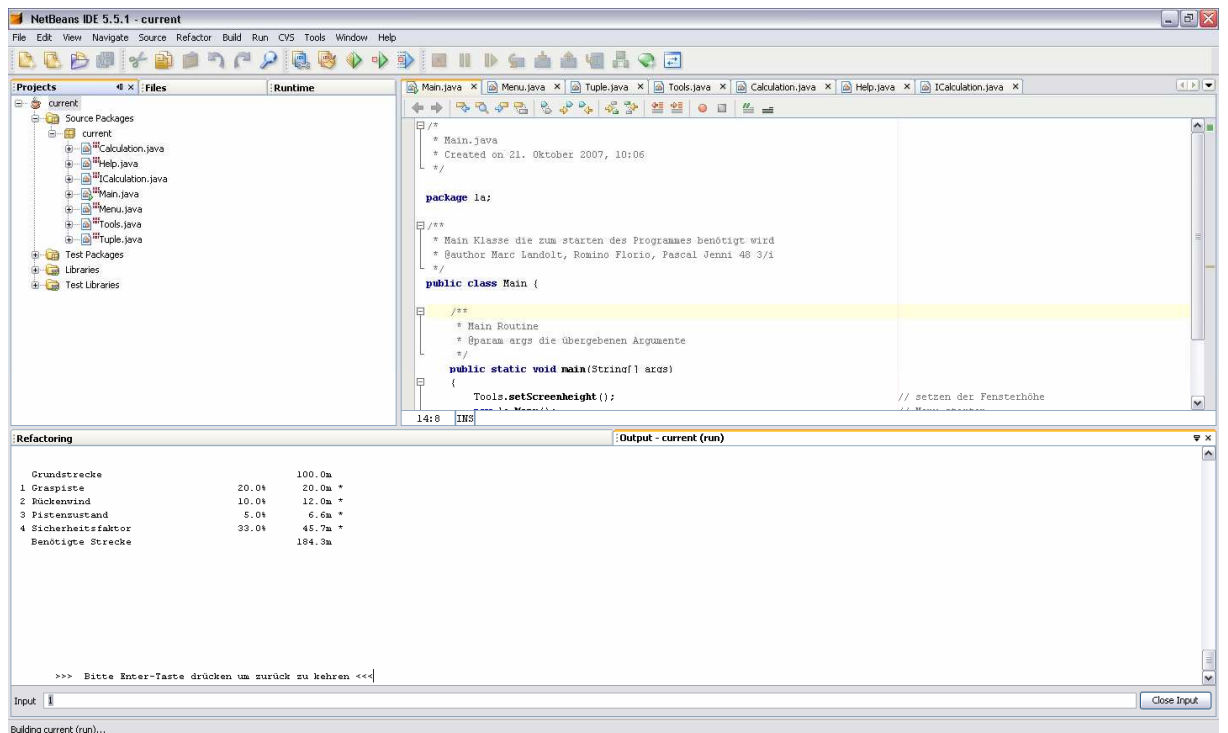
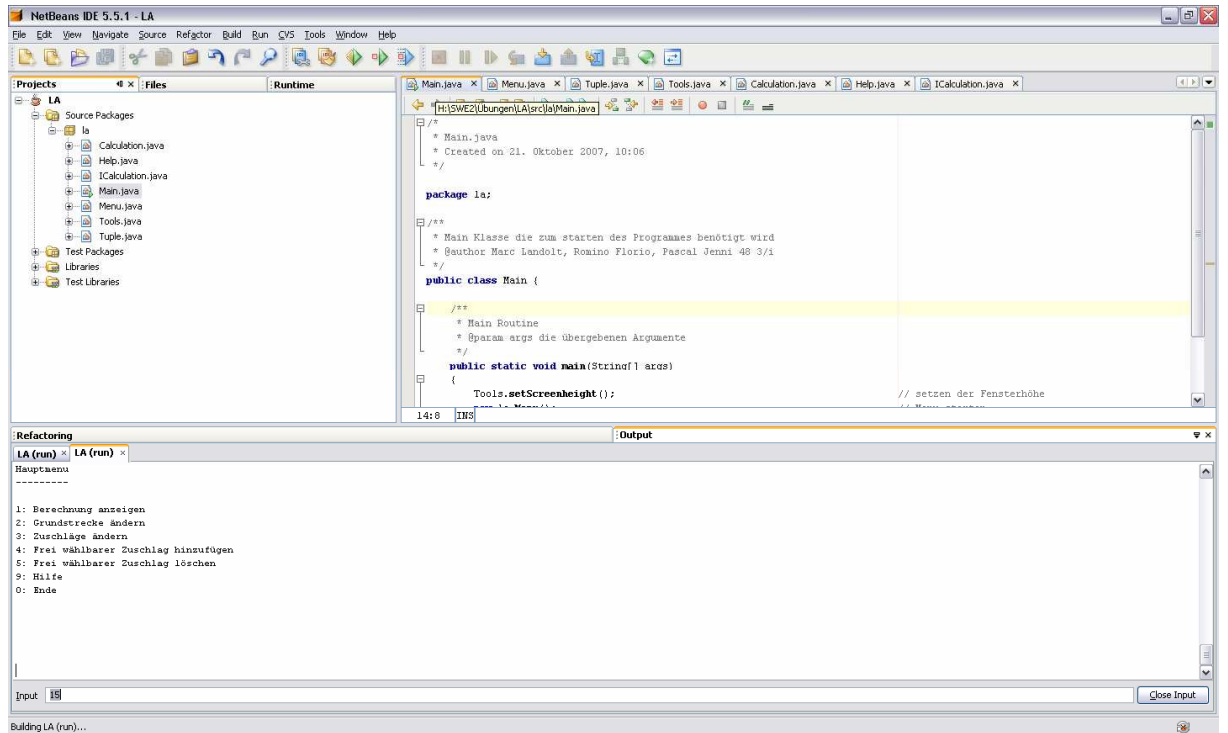
    /**
     * Methode um die Beschreibung zurückzugeben
     * @return die entsprechende Bezeichnung als String
     */
    public String getName()
    {
        return this.mName;
    } // end method

    /**
     * Methode um den Wert eines Zuschlages zu setzen (inklusive Gültigkeitsüberprüfung)
     * @param pValue Der neue Wert
     */
}
```

```
* @return gibt true zurück, wenn alles ok war
*/
public boolean setValue(double pValue)
{
    if(pValue>=0)
    {
        this.mValue=pValue;
        return true;
    }
    return false;
} // end method

/**
 * Methode um den Wert eines Zuschlages zurückzugeben
 * @return den Wert als double
 */
public double getValue()
{
    return this.mValue;
} // end method
} // end class
```

4 PRINTSCREEN



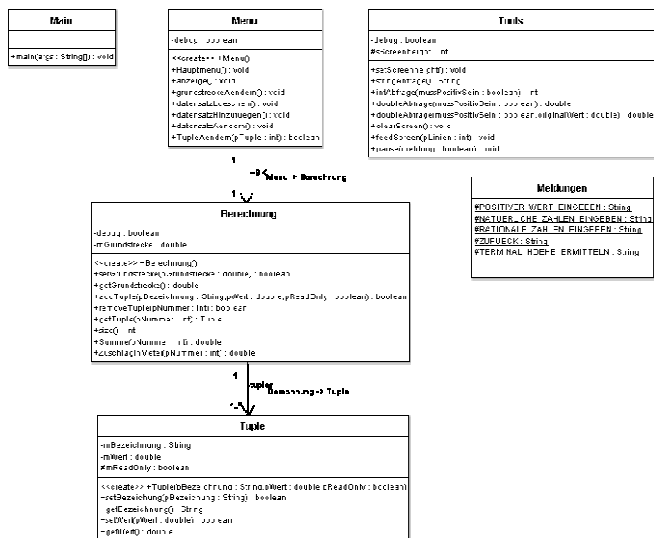
5 BERICHT

5.1 VARIANTEN

In unseren Grundüberlegungen sind wir von einem Menu gesteuerten Programm ausgegangen, welches wir uns in verschiedenen Varianten grob ausarbeitet haben. Die Varianten sind nachfolgend kurz beschrieben.

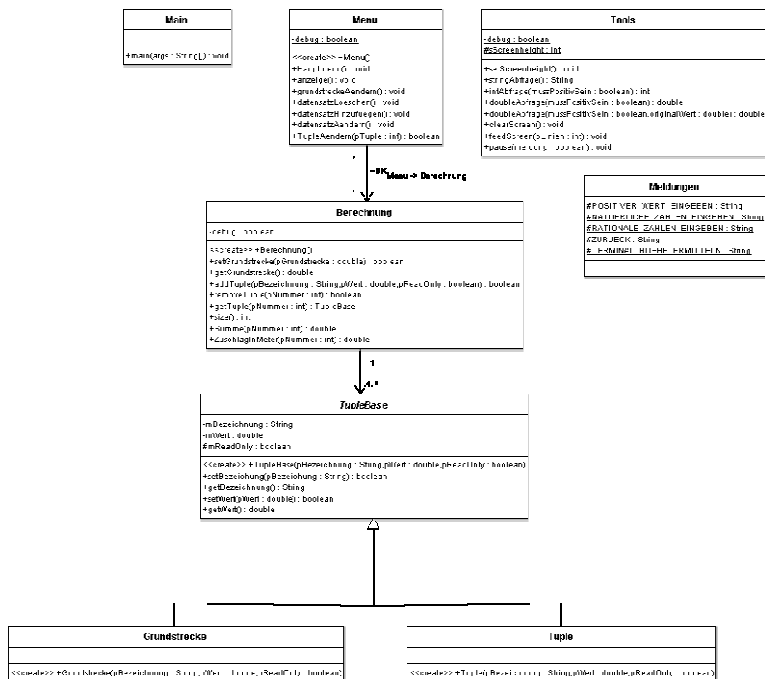
5.1.1 Variante 1

ohne Vererbung, ohne Interface



5.1.2 Variante 2

mit Vererbung, ohne Interface



5.1.3 Variante 3

mit Vererbung, mit Interface → analog Variante 2, jedoch mit Interface

5.1.4 Variante 4

ohne Vererbung, mit Interface (Klassendiagramm siehe Kapitel 2)

5.2 VARIANTENENTSCHEID

Ursprünglich hatten wir uns für die einfachste Variante 1 (ohne Vererbung und ohne Interface) entschieden. Aufgrund von wirtschaftlichen Überlegungen, der guten Lesbarkeit und des geringen Umfangs der vorgegebenen Arbeit wäre diese Variante 1 durchaus berechtigt gewesen. Mit fortlaufendem Unterricht und aufgrund von Empfehlung haben wir uns dann trotzdem entschieden, die Variante 4 (mit Interface) durchzuführen.

Für den Variantenentscheid haben wir uns einem Stärkendiagramm beholfen, welches im Anhang ersichtlich ist.

5.3 VERSIONSBEZEICHNUNG

Die Versionsbezeichnung wurde nach den üblichen Mechanismen der Open Source Gemeinde erstellt.

5.4 TESTBERICHT

5.4.1 Funktion

Die Funktion entspricht den in der Aufgabenstellung gestellten Anforderungen. Teilweise gehen die Funktionen über die gestellten Anforderungen hinaus.

5.4.2 Berechnung

Die Berechnungen wurden anhand von Tests, welche im Anhang ersichtlich sind, verifiziert und für korrekt befunden.

Die Berechnung der Zwischenergebnisse kann in der Berechnungsansicht in einer dritten Spalte dargestellt werden. Während dem erstellen des Programmes und in der Testphase waren diese Zwischenresultate von Vorteil, so konnten z. B. Fehler bei der Berechnung leichter erkannt werden.

Die Spalte mit den Zwischenergebnissen kann über eine Debug-Variable in der Klasse Menu aktiviert werden.

5.4.3 Fehleingaben / Fehlerbehandlung

Getestet wurde die Eingabe von negativen und positiven Zahlen, reellen Zahlen, Sonderzeichen, Text, Gross-/Kleinschreibung, etc. Ebenfalls leere Eingaben wurden getestet.

Das Programm hat in sämtlichen Fällen so reagiert wie von uns erwartet und eine der Situation angepasste Meldung ausgegeben. In einzelnen Fällen wird keine Meldung ausgegeben, sondern das Programm „reagiert mit keiner Aktion“ auf die Eingabe.

Eine detaillierte Auswertung ist im Anhang ersichtlich.

5.4.4 Hilfe

Dem Benutzer steht im Menu 9 eine Übersicht zur Verfügung. Die meisten Eingaben und auch Fehleingaben werden mit einem Hinweis begleitet.

Sämtliche Meldungen sind korrekt ausgegeben und entsprechen der Situation in der sich der Benutzer befindet. Eine detaillierte Auswertung ist im Anhang ersichtlich.

5.4.5 Navigation / Bedienerfreundlichkeit

Die Navigation könnte in folgenden Punkten verbessert werden:

- Zurück immer mit den gleichen Eingaben.
- Dort wo das Programm mit *keiner Aktion* nach einer Fehleingabe reagiert könnte eine Meldung generiert werden.

5.5 ERFAHRUNGEN

Wir haben die Erfahrung gemacht, dass es auch bei kleineren Projekten sinnvoll ist, in der Gruppe verschiedene Varianten zu diskutieren.

Zusammen in der Gruppe den Code schreiben, geht schlecht, deshalb und auch weil das Wissen in der Gruppe unterschiedlich ist, macht es Sinn die verschiedenen Teilarbeiten aufzuteilen und einen groben Zeitplan zu erstellen.

Es hat sich als günstig erwiesen, früh mit der Arbeit zu beginnen. Trotz Zeitplan mussten wir gelegentlich ein bisschen „korrigieren“, damit die verschiedenen Teilarbeiten zur richtigen Zeit fertig wurden. Der frühe Beginn hat uns auch erlaubt, zum Teil mehr ins Detail zu gehen, auch wenn es nicht geplant war.

5.6 ERKENNTNISSE

Mit fortlaufendem Unterricht kamen weitere Ideen dazu, welche unser Programm noch verbessert oder erweitert hätten. Der Versuch, Abstrakte Klassen, Vererbung sowie Interfaces einzubinden erwies sich als in diesem Projekt nicht sehr wirtschaftlich. Die investierte Zeit stand (wirtschaftlich) in keinem Verhältnis zum Ertrag. Aus schulischer Sicht und für das persönliche Verständnis der Materie haben wir uns dann trotzdem entschieden Teile davon zu implementieren.

Damit wir Fehler im Code während der Programmierung besser beheben konnten haben wir Debug-Variablen eingebaut. Ein Beispiel dafür ist im Kapitel 5.4.2 erwähnt.

5.7 STOLPERSTEINE

Das tabellarische darstellen der Streckentabelle erwies sich als nicht ganz einfach. Wir haben das Problem schlussendlich mit `System.out.format(String, Var...)` gelöst und sind mit dem Resultat zufrieden.

Java Doc erwies sich als aufwändig wenn es während der Programmierphase erstellt werden soll. Wir haben daher diesen Teil zurückgestellt bis der Code fertig programmiert war.

ANHANG

Entscheidungsmatrix

Test Berechnungen

	Test 1	Test 2		Test 3		
Grundstrecke	100.00		240.00		340.00	
Graspiste	20.00%	120.0	2.00%	244.8	22.50%	416.5
Rückenwind	10.00%	132.0	44.50%	353.7	4.00%	433.2
Pistenzustand	5.00%	138.6	77.00%	626.1	56.90%	679.6
Sicherheitsfaktor	33.00%	184.3	55.00%	970.5	33.00%	903.9
Zuschlag 1	55.00%	285.7	22.20%	1185.9	0.00%	903.9
Zuschlag 2	44.00%	411.4	1.10%	1199.0	0.00%	903.9
Zuschlag 3		411.4	334.70%	5211.9	0.00%	903.9
Total		411.4		5211.9		903.9
Programm		411.4		5211.4		903.9

