HOCHSCHULE KONSTANZ TECHNIK, WIRTSCHAFT UND GESTALTUNG
UNIVERSITY OF APPLIED SCIENCES

SS16

Anwendung der linearen Optimierung

Leitstand Pflegestation &

Fallpauschalenoptimierung Krankenhausanalyse

Panagiotis Tambouridis 287535
Thorsten Ninnemann 288852

Konstanz den 29.06.2016

Inhaltsverzeichnis

1. Einleitung

Im Rahmen der Veranstaltung „Anwendung der linearen Optimierung“ findet an der HTWG Konstanz im Sommersemester 2016 das Projekt zur Verbesserung der beiden Methoden „Leitstand Pflegestation“ und „Fallpauschalenoptimierung Krankenhausanalyse“ statt.

Anhand dieser Dokumentation werden alle durchgeführten Veränderungen aufgezeigt.

2. Projektziele

2.1 Leitstand Pflegestation:

Ziel dieser Methode war zu einem eine vorhandene Exportfunktion zu optimieren, da bei der Speicherung nicht alle Parameter übernommen wurden.

Des Weiteren war das Ziel eine Importfunktion zu implementieren, um zuvor abgespeicherte Dateien öffnen zu können.

2.2 Fallpauschalenoptimierung Krankenhausanalyse:

Die uns zur Verfügung gestellte Methode wies beim Berechnen Fehler auf, welche behoben werden mussten.

Zusätzlich war es verlangt für den Benutzer eine Hilfefunktion zu implementieren.

3. Rahmenbedingungen

3.1 Leitstand Pflegestation

Das Tool muss unter Windows 7/8 und 8.1 Lauffähig sein und in Java entwickelt werden.

Für die Grafische Darstellung wurde Java Swing verwendet.

Berechnungen dieser Methode werden mit dem Solver „LP-Solve“ durchgeführt.

3.2 Fallpauschalenoptimierung Krankenhausanalyse

Das Tool muss unter Windows 7/8 und 8.1 Lauffähig sein und in C++ entwickelt werden.

Für die Entwicklung ist ein Borland Compiler notwendig.

Auch diese Methode greift zur Berechnung auf den „LP-Solve“ zurück.

4. Projektphase

Zu Beginn des Projektes mussten wir uns in die Funktionsweisen beider Methoden einarbeiten. Nachdem diese uns ersichtlich waren, haben wir uns intensiv mit unseren vorgegebenen Projektzielen beschäftigt und konnten somit mit der direkten Weiterentwicklung beider Methoden beginnen, welche wir im folgenden Kapitel 5 ,,Entwicklung“ beschreiben.

5. Entwicklung

5.1 Leitstand Pflegestation

Wir verwendeten die Sprache Java in der Entwicklungsumgebung Eclipse Mars.

Für die Grafische Oberfläche wurde Java Swing verwendet.

Aufgabe war es die vorhandene Exportfunktion zu optimieren, sowie eine neue Importfunktion zu implementieren.

Alte Version:

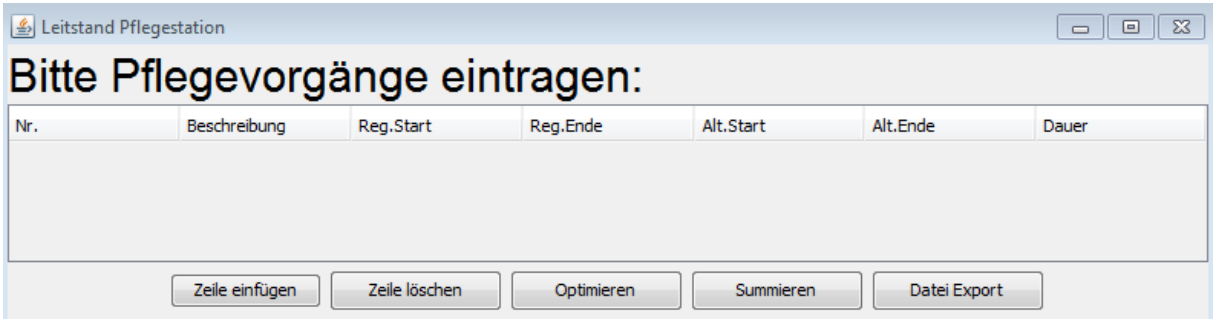

ABBILDUNG 1

Neue Version:

ABBILDUNG 2

In der neuen Version wurde ein zusätzlicher Button „Import“ implementiert.

Die alte Version verfügte bereits über eine Exportfunktion. Jedoch konnten die Dateien nur in einem vordefinierten „temp“ Ordner abgespeichert werden. Es war somit nicht möglich den Speicherort selbst auszuwählen.

Die Dateien werden in einem „bdf“ Format abgespeichert, allerdings war die vorhandene Speicherfunktion fehlerhaft, da nicht alle Parameter übergeben wurden.

In der neuen Version wurde dieser Fehler behoben. Zusätzlich ist es jetzt möglich den Speicherort selbst auszuwählen. Was im folgenden Screenshot zu sehen ist.

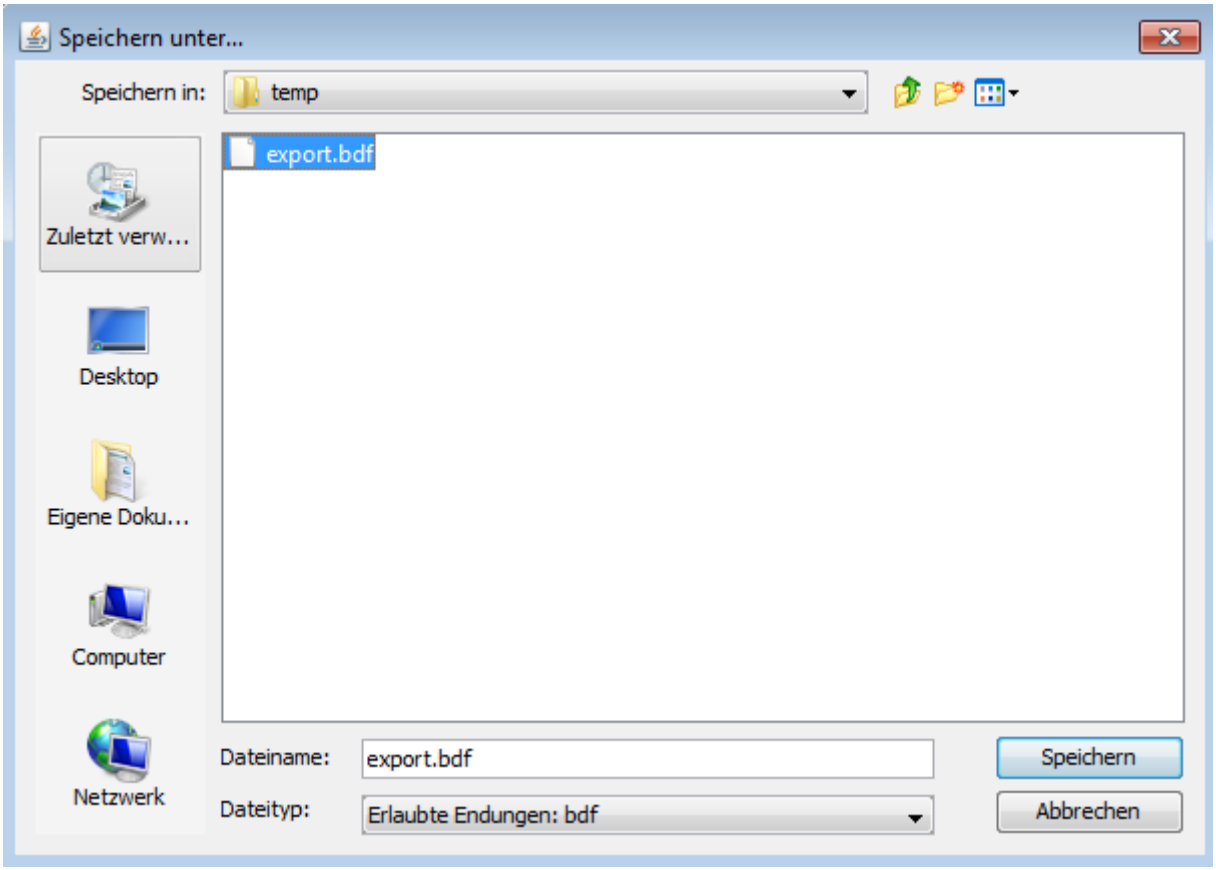

ABBILDUNG 3

Durch die Implementierung einer Importfunktion ist es nun möglich die zuvor abgespeicherten Dateien zu öffnen.

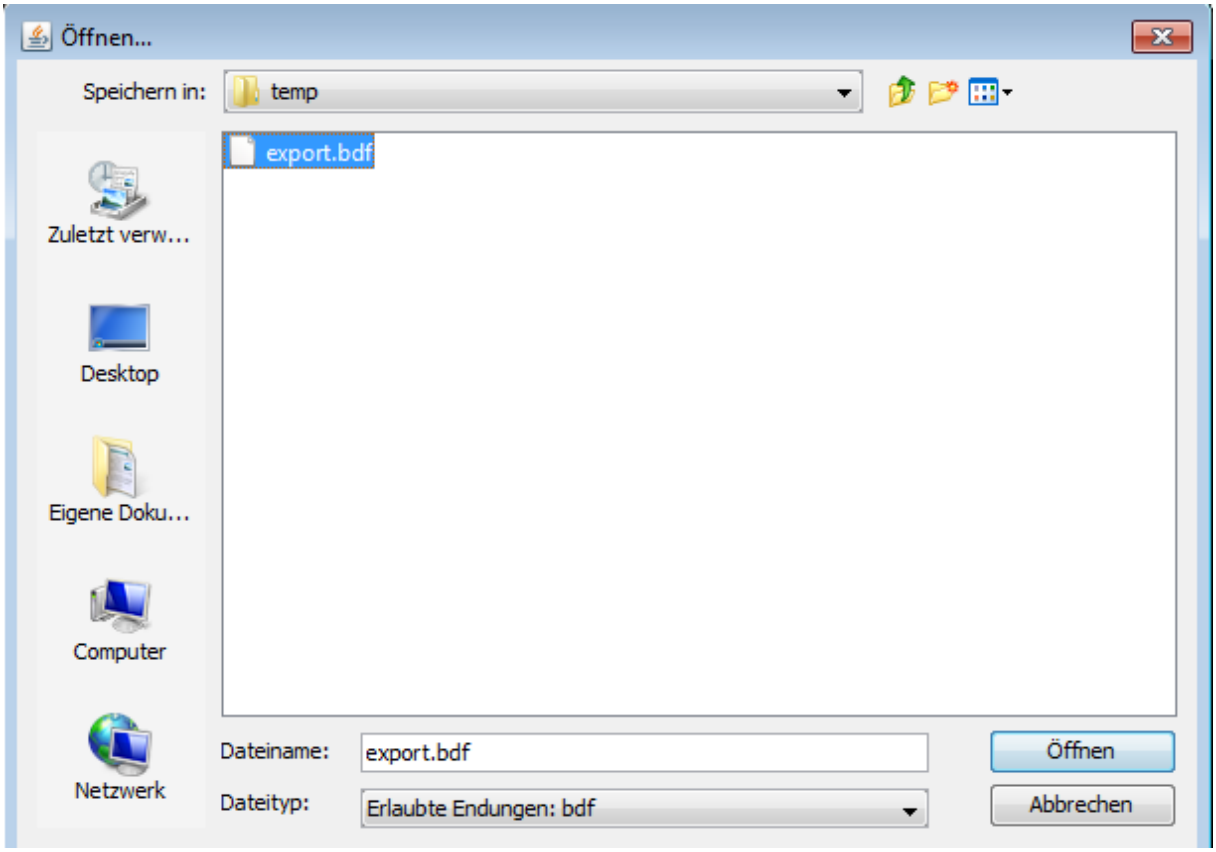

ABBILDUNG 4

5.1.1 Umsetzung Leitstand Pflegestation

```
String saveAs(String pfad) {
    JFileChooser chooser;
    if (pfad == null)
        pfad = System.getProperty("user.home");
    File file = new File(pfad.trim());

    chooser = new JFileChooser(pfad);
    chooser.setDialogType(JFileChooser.SAVE_DIALOG);
    FileNameExtensionFilter plainFilter = new FileNameExtensionFilter("Erlaubte Endungen: bdf", "bdf");
    chooser.removeChoosableFileFilter(chooser.getAcceptAllFileFilter());
    chooser.setFileFilter(plainFilter);
    chooser.setDialogTitle("Speichern unter...");
    chooser.setVisible(true);

    int result = chooser.showSaveDialog(this);

    if (result == JFileChooser.APPROVE_OPTION) {

        chooser.setVisible(false);
        String path = chooser.getSelectedFile().toString();
        if (path.endsWith(".bdf")) {
            return path;
        } else {
            return path + ".bdf";
        }
    }
    chooser.setVisible(false);
    return "";
}
```

ABBILDUNG 5

Abbildung 5 gibt den Code zur Pfadauswahl wieder. Es wurde der JFileChooser verwendet, mit dem es möglich ist den Speicherort selbst auszuwählen.

Zusätzlich wurde definiert, dass es nur erlaubt ist eine „.bdf“ Datei zu speichern.

Falls der Dateiname ohne „.bdf“ Endung abgespeichert wird, wird „.bdf“ automatisch angehängt.

```
void jButtonFileExport_actionPerformed(ActionEvent e) {

    String strPath = saveAs("c:/temp/");
    if (!strPath.isEmpty()) {
        try {
            FileWriter out_file = new FileWriter(strPath);
            BufferedWriter writer = new BufferedWriter(out_file);

            for (int i = 0; i < objInputModel.getRowCount(); i++) {
                writer.write(objInputModel.getValueAt(i, 0) + ";" + objInputModel.getValueAt(i, 1) + ";"
                        + objInputModel.getValueAt(i, 2) + ";" + objInputModel.getValueAt(i, 3) + ";"
                        + objInputModel.getValueAt(i, 4) + ";" + objInputModel.getValueAt(i, 5) + ";"
                        + objInputModel.getValueAt(i, 6) + ";" + objInputModel.getValueAt(i, 7));
                writer.newLine();
            }
```

ABBILDUNG 6

Als Standard Speicherort ist weiterhin der „temp“ Ordner festgelegt.
Die Parameter werden mit BufferedWriter in die abzuspeichernde Datei geschrieben.

```
    void jButtonFileImport_actionPerformed(ActionEvent e) {

        String strPath = saveAs("c:/temp/");
        if (!strPath.isEmpty()) {
            for(int i = 0; i < objInputModel.getRowCount(); i++){
                objInputModel.removeElement(i);
            }
            try {
                FileReader in_file = new FileReader(strPath);
                BufferedReader reader = new BufferedReader(in_file);
                int i = 0;
                String str = "", line = "";
                while ((line = reader.readLine()) != null) {
                    WorkingStep ws = new WorkingStep(0);
                    String[] elements = line.split(";");
                    ws.setNumber(Integer.parseInt(elements[0]));
                    ws.setDescription(elements[1]);
                    ws.setRegularStartHour(elements[2].split(":")[0]);
                    ws.setRegularStartMinute(elements[2].split(":")[1]);
                    ws.setRegularEndHour(elements[3].split(":")[0]);
                    ws.setRegularEndMinute(elements[3].split(":")[1]);
                    ws.setAlternativeStartHour(elements[4].split(":")[0]);
                    ws.setAlternativeStartMinute(elements[4].split(":")[1]);
                    ws.setAlternativeEndHour(elements[5].split(":")[0]);
                    ws.setAlternativeEndMinute(elements[5].split(":")[1]);
                    ws.setDuration(elements[6]);
                    ws.setExecutionTime(elements[7]);

                    objInputModel.addElement(ws);
                }
                JOptionPane.showMessageDialog(this, "Datei wurde aus " + strPath + " importiert");
            } catch (Exception ex) {

                JOptionPane.showMessageDialog(this, "Datei konnte nicht importiert werden!");
            }
        }

    }
```

ABBILDUNG 7

In der Abbildung 7 wird die Methode aufgezeigt, welche bei einem Klick auf den Button „Import“ ausgeführt wird und somit die Datei öffnet.

```
jButtonFileImport.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(ActionEvent e) {
        jButtonFileImport_actionPerformed(e);
    }
});
jButtonFileImport.setText("Import");
jButtonFileImport.setPreferredSize(new Dimension(113, 27));
jButtonFileImport.setActionCommand("Datei Import");
jButtonFileImport.setMinimumSize(new Dimension(113, 27));
jButtonFileImport.setMaximumSize(new Dimension(113, 27));
contentPane.add(jPanelMain, new GridBagConstraints(0, 0, 1, 1, 1.0, 1.0, GridBagConstraints.CENTER,
        GridBagConstraints.BOTH, new Insets(0, 0, 0, 0), 323, -460));
jPanelMain.add(jLabelHeader, BorderLayout.NORTH);
jPanelMain.add(jScrollPaneInput, BorderLayout.CENTER);
jPanelMain.add(jPanelButtons, BorderLayout.SOUTH);
jScrollPaneInput.getViewport().add(jTableInput, null);
jPanelButtons.add(jButtonAddRow, null);
jPanelButtons.add(jButtonRemoveRow, null);
jPanelButtons.add(jButtonOptimize, null);
jPanelButtons.add(jButtonVisualize, null);
jPanelButtons.add(jButtonFileExport, null);
jPanelButtons.add(jButtonFileImport, null);
```

ABBILDUNG 8

Der Import Button wird definiert und zur GUI hinzugefügt. Zusätzlich werden Größen bzw. Anordnungsparameter festgelegt.

5.2 Fallpauschalenoptimierung Krankenhausanalyse

Für die Entwicklung dieser Methode wird die Sprache C++ verwendet. Der Einsatz von einem Borland Compiler ist zwingend notwendig. Zur Berechnung wird der Solver ;,LP-Solve“ verwendet.

Aufgabe war es für den Benutzer eine Hilfefunktion zu implementieren sowie die Berechnung funktionsfähig zu machen

5.2.1 Umsetzung Fallpauschalenoptimierung Krankenhausanalyse

```
//÷ffnen der Hilfe
//Fehler, Hilfe hinter Hauptfenster, bug leider nicht gefunden
ShellExecute(Handle, "Open", "Hilfefunktion.pdf", NULL, NULL, SW_SHOW);
Application->HelpCommand(HELP_FINDER, 0);
```

ABBILDUNG 9

Hier wird definiert das sich eine PDF-Datei Namens ,,Hilfefunktion.pdf“ öffnet, wenn man auf den Button ,,Hilfe“ klickt. Der folgende Screenshot zeigt die geöffnete Hilfefunktion.

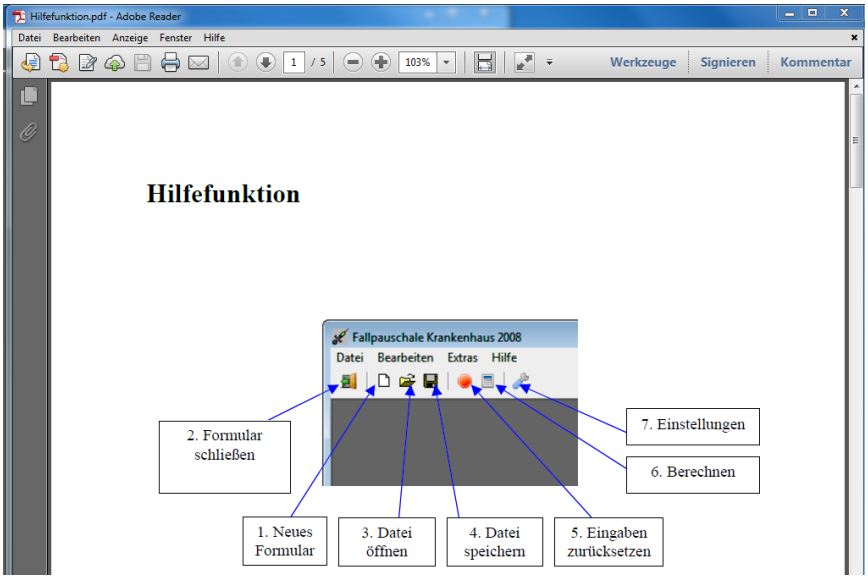

ABBILDUNG 10

In der alten Version wurde bei der Durchführung einer Berechnung keine Ergebnisse geliefert. Lediglich eine Meldung ,,Nicht berechnet“, wie im nachfolgenden Screenshot zusehen.

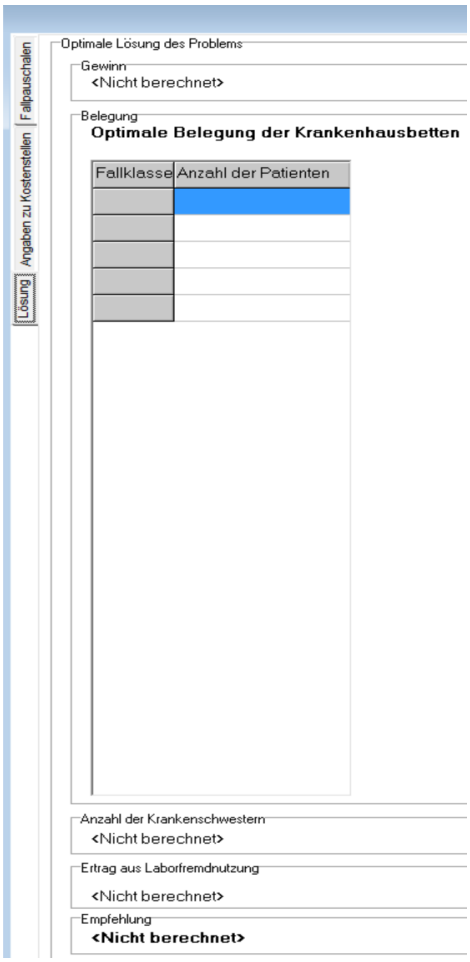

ABBILDUNG 11

Das Problem bestand darin, dass der Solverpfad nicht richtig eingestellt war. Nach dem dieser geändert wurde konnten nun Berechnungen durchgeführt werden. Anhand der OR Aufgabe 45 Teil 3 wurde der LP-Ansatz überprüft. Welcher wie im folgenden Screenshot korrekte Ergebnisse liefert.

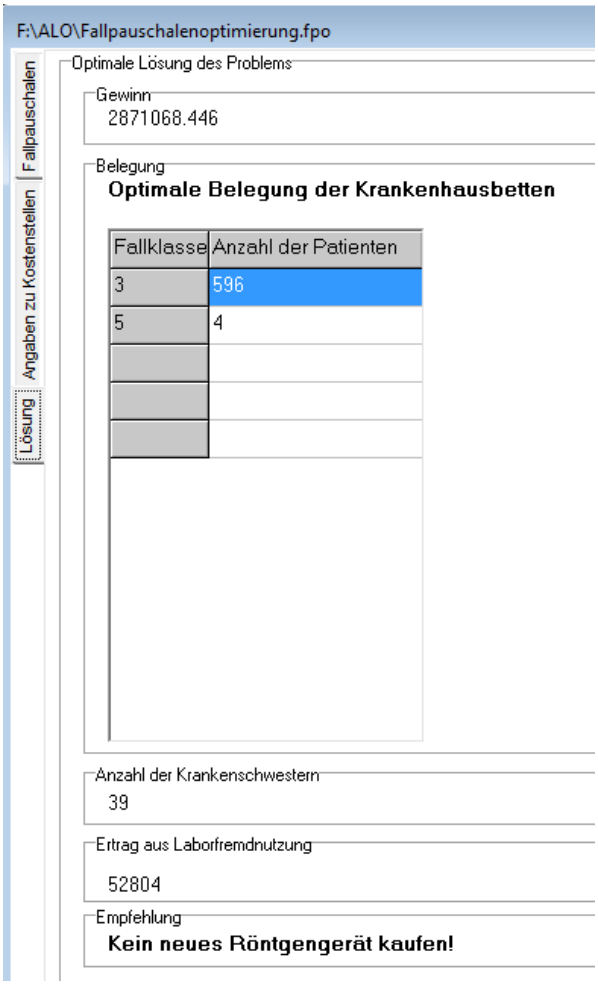

ABBILDUNG 12

6. Fazit

Der Projektverlauf hatte seine Höhen und Tiefen. Nach dem wir mit der Methode Leitstand Pflegestation gut vorangekommen sind und alles reibungslos verlief, sind wir bei der Methode Fallpauschelenoptierung auf die erste große Hürde gestoßen.

Die Methode Fallpauschalenoptimierung wurde mit C++ entwickelt. Da wir im Rahmen unseres Studiums überwiegend mit Java arbeiten, mussten wir uns erstmal auf diese für uns relativ neue Sprache einarbeiten beziehungsweise umstellen.

Das nächste Problem bestand darin die Methode in einem Projekt zu öffnen. Dies ist erst mit dem Einsatz eines Borland Compilers gelungen.

Durch den Projektverlauf konnten wir unsere Kenntnisse in Java, Java Swing sowie C++ verbessern.

Wir sind froh die an uns gestellten Aufgaben bewältigt zu haben.