HOCHSCHULE
KONSTANZ
TECHNIK, WIRTSCHAFT
UND GESTALTUNG

PROJEKT DOKUMENTATION

WAGNER-WHITIN LP 1.3

TEAMMITGLIEDER:	Roland Kuczewski, 288958 Sonja Kordovan, 289719
MODUL:	Anwendung der linearen Optimierung
PROJEKTNUMMER:	ALO Projekt Nr. 2 Sommersemester 2017
PROJEKTNAME:	Wagner-Whitin LP 1.2
ABGABETERMIN:	29.06.2017
PROJEKTVERANTWORTLICHE:	Herr Prof. Dr. Grütz Serkan Önnisan

FEHLERSTATUS:

- Wenn bei bereits eingegebenem Modell im Menü unter Bearbeiten->alles ändern auf abbrechen geklickt wird, wird das eingegebene Modell gelöscht. Das Löschen soll hier verhindert werden.
- Erst nach dem Öffnen eines neuen Modells werden Defaultkosten übernommen. Dies soll aber auch bei einem bereits geöffneten Modell erfolgen.

Inhalt

Inhalt

1 Einleitung

Im Rahmen der Veranstaltung „Anwendung der linearen Optimierung“ sollen die in der Methodenbank OR-Alpha aufgeführten Methoden analysiert und verbessert werden. Um dieses Fach erfolgreich abschließen zu können, müssen drei Teilmodule der Methodenbank analysiert und die Ergebnisse vorgestellt werden. Weiterhin muss eine Projektarbeit in Zweierteams durchgeführt werden. Dieses Projekt beinhaltet die Entwicklung und Systemanalyse von Methoden im Bereich der logistischen Problemstellung.
Das Projekt beschäftigt sich mit der Methode „Wagner Whitin LP“. Diese hat zu Beginn des Semesters die oben aufgeführten Fehler, die bis Ende des Projekts behoben worden sind.

2 Projektarbeit

Das zu bearbeitende Projekt gliedert sich in zwei Hautaufgaben. Zuerst soll das Löschen des Modells verhindert werden. Hierbei muss erst der Bug lokalisiert werden und dann im Java Code der Fehler verbessert werden.

Des Weiteren sollen die Defaultkosten bei einem bereits geöffneten Modell übernommen werden. Auch hier muss der Bug lokalisiert werden und der Fehler im Java Code verbessert werden.

Bei einer zusätzlichen Codeoptimierung wurden nicht genutzter Methoden ausgeklammert und leere Catch-Blöcke befüllt, um ein unvorhersehbares Verhalten zu vermeiden.

3 Fehlerbehebung

3.1 Löschen verhindern

Sobald man Änderungen im Modell über *Bearbeiten* → *Alles ändern* vornehmen möchte und dann auf *Abbrechen* drückt, wurde bis her das bestehende Modell gelöscht. Um dies zu vermeiden musst der Schließaufruf **ref.dispose()** in der Klasse **plo_Hauptfenster.class** auskommentiert werden.

```
//*** Methode um das einlesen eines Modells neu zu starten ************************************************************
    public void allesAendern()
    {
        plo_Eingabemaske ref= new plo_Eingabemaske(this, 0, "temp", false, false, true, true);
        ref = this.getInternalFocus();
        this.getInternalFocus().getEm_ZugehoerigesErgebnisfenster().dispose();                                    //Alt
        plo_AnzahlNachfragenDialog dialog = new plo_AnzahlNachfragenDialog(this, 1, this.getInternalFocus().getTitle());/
        this.setAnzahlNachfragen(dialog.getInt_AnzahlNachfragen());                                                /
        /**
         * edited by roland
         * In ref wird der aktuelle Fokus erfasst, dieser ist das aktuell angeklickte Fenster.
         * Auskommentieren des Schließaufrufst (dispose) weil altes Fenster nicht geschlossen werden soll.
         * => Gelöster Bug für das Fach ALO.
         * ref.dispose();
         */
    }
//*** ende  ***************************************************************************************
```

Abbildung 1 Schließaufruf auskommentiert

3.2 Defaultkosten

Möchte man in Wagner Whitin bei einem bestehenden Modell eine weitere Alternative einfügen, wurden bisher die Defaultkosten nicht automatisch eingetragen. Diesen Fehler sollte behoben werden. Hierfür wurde in der Klasse **nachfrage.class** ein neuer **Konstruktor** für den Defaultkosten-Bug eingefügt.

```
public nachfrage(double lagerkosten, double bestellkosten, int periode)
{
    tf_Nummer = new JTextField("0", 5);                 //Initialisiern der Textfelder
    tf_Menge = new JTextField("0", 10);
    tf_Periode = new JTextField(String.valueOf(periode), 5);
    tf_Bestellkosten = new JTextField(String.valueOf(bestellkosten), 10);
    tf_Lagerkosten = new JTextField(String.valueOf(lagerkosten), 10);
}
```

Abbildung 2 zusätzlicher Konstruktor für Defaultkosten-Bug

In der Klasse **plo_Hauptfenster.class** wird eine neue **Methode** eingefügt.

```
//***weitere Nachfrage einfügen ************************************************
    //Methode um in ein bestehendes Modell eine weitere Nachfrage einzufuegen
    public void nachfrageEinfuegen()
    {
        /**
         * edited by roland und sonja
         * bestell und lagerkosten abgreifen
         */
        double bestellkosten = this.defaultBestellkosten;
        double lagerkosten = this.defaultLagerkosten;
        this.getInternalFocus().nachfrageEinfuegen(bestellkosten, lagerkosten);
    }
// ***Ende Nachfrage einfügen ************************************************
```

Abbildung 3 neue Methode in plo_Hauptfenster.class

In der Klasse **plo_Eingabemaske.closs** wurde die Methode **nachfrageEinfeugen()** erweitert.

```
public void nachfrageEinfuegen(double bestellkosten, double lagerkosten)
{
   //Erstellen eines neuen, um 1 laengeren Nachfrage-Arrays
    nachfrage[] tempListe = new nachfrage[anzahlNachfragen+1];

   //Initialisiern des neuen Arrays
    for(int i = 0; i < (anzahlNachfragen+1); i++)
    {
        tempListe[i] = new nachfrage();
    }

   //Kopieren der alten Nachfragen in das neue Array,so dass die letzte Nachfrage leer bleibt
    for(int i = 0; i <= anzahlNachfragen; i++)
    {
        tempListe[i] = em_NachfragenListe[i];

        /**
         * edited by roland und sonja
         * neue Nachfrage mit lager- sowie bestellkosten, dazu muss der index angegeben werden welcher bei anzahlNachfragen+1 liegt.
         */
        if(i == anzahlNachfragen-1){
            tempListe[anzahlNachfragen] = new nachfrage(lagerkosten, bestellkosten, anzahlNachfragen+1);
            break;
        }
    }
```

Abbildung 4 Erweiterund der Methode

3.3 zusätzliche Codoptimierung

Um unvorhersehbares Verhalten zu vermeiden, wurden folgenden Leere Catch-Blöcke aus den entsprechenden Klassen entfernt oder befüllt.

Der folgende Catch-Block aus der Klasse **plo_Hauptfenster** wurde befüllt:

```
catch (IOException ioe)
{
        /**
         * edited by roland und sonja
         * niemals einen leeren catch block coden ;) !
         */
        ioe.printStackTrace();
}
```

Abbildung 5 befüllter Catch-Block

In der Klasse **plo_AnzahlNachfragenDialog** wurde folgender Catch-Block gelöscht:

```
public void keyReleased(KeyEvent kevt)
{

}
public void keyTyped(KeyEvent kevt)
{

}
```

Abbildung 6 leerer Catch-Block

Folgende Catch-Blöcke wurden in der Klasse **plo_Ef_SubwindowListener** entfernt:

```
public void internalFrameOpened(InternalFrameEvent ifevt)
{

}


public void internalFrameIconified(InternalFrameEvent ifevt)
{

}


public void internalFrameDeiconified(InternalFrameEvent ifevt)
{

}
```

Abbildung 7 leerer Catch-Block

```
public void internalFrameDeactivated(InternalFrameEvent ifevt)
{

}


public void adjustmentValueChanged(AdjustmentEvent aevt)
{
```

Abbildung 8 leerer Catch-Block

4 Abbildungsverzeichnis