Code-Review

Brian Frølund
Carl-M Höfer
Morten Carl Høj

20. maj 2010

Indhold

1 dao.*

DaoDb4o.java

```
1  package dao;
2
3
4  import java.util.List;
5
6  import model.*;
7  import model.Process;
8
9  import com.db4o.Db4oEmbedded;
10 import com.db4o.ObjectContainer;
11 import com.db4o.config.EmbeddedConfiguration;
12
13 /**
14  * @author Brian
15  */
16
17 public class DaoDb4o implements Dao {
18
19         private ObjectContainer db;
20         private static DaoDb4o dao = null;
21
22         private DaoDb4o(){
23                 EmbeddedConfiguration configuration = Db4oEmbedded.newConfiguration();
24                 configuration.common().activationDepth(6);
25                 configuration.common().updateDepth(6);
26
27                 db = Db4oEmbedded.openFile(configuration,"db.db4o");
28
29         }
30
31         public static Dao getDao(){
32                 if (dao == null)
33                         dao = new DaoDb4o();
34                 return dao;
35         }
36
37         //Depot
38         public List<Depot> getAllDepots() {
39                 return db.query(Depot.class);
40         }
41
42         public void store(Depot depot) {
43                 db.store(depot);
44                 db.commit();
45         }
46
47         public void delete(Depot depot) {
48                 db.delete(depot);
49                 db.commit();
50         }
51
52         //IntermediateProduct
53         public List<IntermediateProduct> getAllIntermediateProducts() {
54                 return db.query(IntermediateProduct.class);
55         }
56
57         public void store(IntermediateProduct intermediateProduct) {
58                 db.store(intermediateProduct);
59                 db.commit();
60         }
61
62         public void delete(IntermediateProduct intermediateProduct) {
63                 db.delete(intermediateProduct);
64                 db.commit();
65         }
66
67         //ProductType
68         public List<ProductType> getAllProductTypes() {
69                 return db.query(ProductType.class);
70         }
71
72         public void store(ProductType productType) {
73                 db.store(productType);
74                 db.commit();
75         }
```

```
76
77          public void delete(ProductType productType) {
78                  db.delete(productType);
79                  db.commit();
80          }
81
82
83          public void close(){
84                  db.close();
85          }
86
87  }
```

Dao.java

```
1  package dao;
2
3  import java.util.List;
4
5  import model.Depot;
6  import model.Drying;
7  import model.IntermediateProduct;
8  import model.Process;
9  import model.ProcessLine;
10 import model.ProcessLog;
11 import model.ProductType;
12 import model.StoringSpace;
13 import model.SubProcess;
14
15 /**
16  * @author Brian
17  */
18
19 public interface Dao {
20          //Depot
21          public List<Depot> getAllDepots();
22          public void store (Depot depot);
23          public void delete(Depot depot);
24
25          //IntermediateProduct
26          public List<IntermediateProduct> getAllIntermediateProducts();
27          public void store(IntermediateProduct interMediateProduct);
28          public void delete(IntermediateProduct interMediateProduct);
29
30          //ProductType
31          public List<ProductType> getAllProductTypes();
32          public void store(ProductType productType);
33          public void delete(ProductType productType);
34
35          public void close();
36 }
```

DaoList.java

```
1  package dao;
2
3  import java.util.ArrayList;
4  import java.util.List;
5
6  import model.Depot;
7  import model.Drying;
8  import model.IntermediateProduct;
9  import model.Process;
10 import model.ProcessLine;
11 import model.ProcessLog;
12 import model.ProductType;
13 import model.StoringSpace;
14 import model.SubProcess;
15
16 /**
17  * @author Brian
18  */
19
20 public class DaoList implements Dao {
21         private List<Depot> depots;
22         private List<IntermediateProduct> intermediateProducts;
23         private List<Process> processes;
24         private List<ProcessLine> processLines;
25         private List<ProductType> productTypes;
26
27         private static DaoList dao = null;
28
29         private DaoList() {
30                 depots = new ArrayList<Depot>();
31                 intermediateProducts = new ArrayList<IntermediateProduct>();
32                 processes = new ArrayList<Process>();
33                 processLines = new ArrayList<ProcessLine>();
34                 productTypes = new ArrayList<ProductType>();
35         }
36
37         public static Dao getDao() {
38                 if (dao == null)
39                         dao = new DaoList();
40                 return dao;
41         }
42
43         //Depot
44         public List<Depot> getAllDepots() {
45                 return depots;
46         }
47
48         public void store(Depot depot) {
49                 if (!depots.contains(depot))
50                         depots.add(depot);
51         }
52
53         public void delete(Depot depot) {
54                 depots.remove(depot);
55         }
56
57         //IntermediateProduct
58         public List<IntermediateProduct> getAllIntermediateProducts() {
59                 return intermediateProducts;
60         }
61
62         public void store(IntermediateProduct interMediateProduct) {
63                 if (!intermediateProducts.contains(interMediateProduct))
64                         intermediateProducts.add(interMediateProduct);
65         }
66
67         public void delete(IntermediateProduct interMediateProduct) {
68                 intermediateProducts.remove(interMediateProduct);
69         }
70
71         //ProductType
72         public List<ProductType> getAllProductTypes() {
73                 return productTypes;
74         }
75
```

```
76          public void store(ProductType productType) {
77                  if (!productTypes.contains(productType))
78                          productTypes.add(productType);
79          }
80
81          public void delete(ProductType productType) {
82                  productTypes.remove(productType);
83          }
84
85          @Override
86          public void close() {
87                  // do nothing here
88          }
89
90
91  }
```

2 gui.*

ChooseDepotsFrame.java

```
1  package gui;
2
3  import java.awt.Container;
4  import java.awt.Font;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.util.ArrayList;
8
9  import javax.swing.*;
10
11
12 /**
13  * Shows an Jlist that can be edited by the push of a button<br>
14  * the selcteble List will be handled as an ArrayList
15  * @author M. C. H j
16  */
17 @SuppressWarnings("serial")
18 public class ChooseDepotsFrame extends Container{
19
20         private JButton btnOk, btnValgVTilE, btnFjernFraE, btnOpenMiniFrame, btnVaelgAlle,
               btnVaelgIngen;
21         private JLabel lblSelectebleElements, lblSelctedElements;
22         private JScrollPane scSelctedElements, scSelctedElementsMF, scSelectebleElementsMF;
23         private JList jlSelectedElements, jlSelectedElementsMF, jlSelectebleElementsMF;
24         private JDialog miniFrame;
25
26         /**
27          * ArrayList over Elements that can be selected
28          */
29         private ArrayList<Object> selectebleElements = new ArrayList<Object>();
30         /**
31          * ArrayList over Elements that are selected
32          */
33         private ArrayList<Object> selectedElements = new ArrayList<Object>();
34         /**
35          * ArrayList over Elements that can be selected
36          */
37         private ArrayList<Object> origenList = new ArrayList<Object>();
38
39         /**
40          * DefaultListModel over Elements names that can be selected
41          */
42         private DefaultListModel selectebleElementsNames = new DefaultListModel();
43         /**
44          * DefaultListModel over Elements names that are selected
45          */
46         private DefaultListModel selectedElementsNames = new DefaultListModel();
47         /**
48          * DefaultListModel over Elements names that can be selected
49          */
50         private DefaultListModel origenListNames = new DefaultListModel();
51
52         /**
53          * actionsListener for our Jbuttons
54          */
55         private BtnController btnController = new BtnController();
56
57         // ***************************** Constructer ************************************
58         /**
59          * the constructer sets the grafical interface for the MultiSelectableList
60          * @param ElementNavn The names of the elements
61          */
62         public ChooseDepotsFrame(String ElementNavn){
63                 JFrameSetup(ElementNavn);
64
65                 jlSelectedElements = new JList(selectedElementsNames);
66                 jlSelectedElements.setSelectionMode(DefaultListSelectionModel.
                       MULTIPLE_INTERVAL_SELECTION);
67
68                 scSelctedElements = new JScrollPane(jlSelectedElements);
69                 scSelctedElements.setSize(150, 75);
70                 scSelctedElements.setLocation(0, 0);
71                 this.add(scSelctedElements);
72
73                 btnOpenMiniFrame = new JButton("Vaelg "+ ElementNavn);
```

```
74                  btnOpenMiniFrame.setSize(150, 25);
75                  btnOpenMiniFrame.setLocation(0, 75);
76                  btnOpenMiniFrame.addActionListener(btnController);
77                  this.add(btnOpenMiniFrame);
78
79                  super.setSize(150, 100);
80                  this.setVisible(true);
81
82          }
83          // ***************************** Container methods ************************************
84          /**
85           * Resizes this component so that it has width width and height height.
86           * @param width the new width of this component in pixels
87           * @param height the new height of this component in pixels
88           */
89          public void setSize(int width, int height){
90
91                  if (width<150){
92                          width=150;
93                  }
94                  if (height<100){
95                          height=100;
96                  }
97
98                  scSelctedElements.setSize(width, height-25);
99                  btnOpenMiniFrame.setSize(width, 25);
100                 btnOpenMiniFrame.setLocation(0, height-25);
101                 super.setSize(width, height);
102
103         }
104
105         // ***************************** List methods ************************************
106
107         /**
108          * adds an objekt that can be selected to the end of the list, and sets the objects name
109          * @param object The object that will be added to the list
110          * @param objectName The showed name of the object
111          */
112         public void add(Object object, String objectName,boolean isSelected){
113                 origenList.add(object);
114                 origenListNames.addElement(objectName);
115                 if (isSelected) {
116                         selectedElements.add(object);
117                         selectedElementsNames.addElement(objectName);
118                 } else {
119                         selectebleElements.add(object);
120                         selectebleElementsNames.addElement(objectName);
121                 }
122         }
123
124         /**
125          * adds an objekt that can be selected to the end of the list, and sets the objects name to the
                returned String from the toString method
126          * @param object The object that will be added to the list
127          */
128         public void add(Object object, boolean isSelected){
129                 origenList.add(object);
130                 origenListNames.addElement(object.toString());
131                 if (isSelected) {
132                         selectedElements.add(object);
133                         selectedElementsNames.addElement(object.toString());
134                 } else {
135                         selectebleElements.add(object);
136                         selectebleElementsNames.addElement(object.toString());
137                 }
138         }
139
140         /**
141          * adds an objekt that can be selected to a specific position, and sets the objects name
142          * @param index The position where the object will be added
143          * @param object The object that will be added to the list
144          * @param objectName The showed name of the object
145          */
146         public void add(int index, Object object, String objectName, boolean isSelected){
147                 origenList.add(index, object);
148                 origenListNames.add(index, objectName);
149                 if (isSelected) {
150                         selectedElements.add(index, object);
```

```
151                     selectedElementsNames.add(index, objectName);
152             } else {
153                     selectebleElements.add(index, object);
154                     selectebleElementsNames.add(index, objectName);
155             }
156     }
157
158     /**
159      * adds an objekt that can be selected to a specific position, and sets the objects name to the
             returned String from the toString method
160      * @param index The position where the object will be added
161      * @param object The object that will be added to the list
162      */
163     public void add(int index, Object object, boolean isSelected){
164             origenList.add(index, object);
165             origenListNames.add(index, object.toString());
166             if (isSelected) {
167                     selectedElements.add(index, object);
168                     selectedElementsNames.add(index, object.toString());
169             } else {
170                     selectebleElements.add(index, object);
171                     selectebleElementsNames.add(index, object.toString());
172             }
173     }
174
175     /**
176      * Removes object from the lists
177      * @param object Object to remove
178      */
179     public void remove(Object object){
180
181             if (origenList.contains(object)){
182                     int index = origenList.indexOf(object);
183                     origenList.remove(index);
184                     origenListNames.remove(index);
185                     if (selectebleElements.contains(object)){
186                             index = selectebleElements.indexOf(object);
187                             selectebleElements.remove(index);
188                             selectebleElementsNames.remove(index);
189                     }
190                     if (selectedElements.contains(object)){
191                             index = selectedElements.indexOf(object);
192                             selectedElements.remove(index);
193                             selectedElementsNames.remove(index);
194                     }
195             }
196
197     }
198
199     /**
200      * Removes object at a specific lokation
201      * @param index Lokation of the object to remove
202      */
203     public void remove(int index){
204             if (origenList.size()>index){
205                     Object object = origenList.get(index);
206                     origenList.remove(index);
207                     origenListNames.remove(index);
208                     if (selectebleElements.contains(object)){
209                             index = selectebleElements.indexOf(object);
210                             selectebleElements.remove(index);
211                             selectebleElementsNames.remove(index);
212                     }
213                     if (selectedElements.contains(object)){
214                             index = selectedElements.indexOf(object);
215                             selectedElements.remove(index);
216                             selectedElementsNames.remove(index);
217                     }
218             }
219     }
220
221     /**
222      * Clears all lists
223      */
224     public void Clear(){
225             origenList.clear();
226             origenListNames.clear();
227             selectebleElements.clear();
```

```
228             selectebleElementsNames.clear();
229             selectedElements.clear();
230             selectedElementsNames.clear();
231         }
232 
233         /**
234          * Clears the list over selected items
235          */
236         public void ClearSelctedItemsList(){
237             selectedElements.clear();
238             selectedElementsNames.clear();
239 
240             selectebleElements.clear();
241             selectebleElementsNames.clear();
242 
243             for (int i=0;i<origenList.size();i++){
244                 selectebleElements.add(origenList.get(i));
245                 selectebleElementsNames.addElement(origenListNames.elementAt(i));
246             }
247 
248         }
249 
250         /**
251          * Gets the list over objects that can be selcted
252          */
253         public ArrayList<Object> getSelectebleElements(){
254             return origenList;
255         }
256 
257         /**
258          * Gets the list over objects that isn't selected
259          */
260         public ArrayList<Object> getNonSelectedElements(){
261             return selectebleElements;
262         }
263 
264         /**
265          * Gets the list over selcted bjects
266          */
267         public ArrayList<Object> getSelectedElements(){
268             return selectedElements;
269         }
270 
271         /**
272          * Gets the indexes of the selceted objects
273          */
274         public int[] getSelectedIndexs(){
275             int[] integersToReturn = new int[selectedElements.size()];
276 
277             for (int i=0; i<selectedElements.size();i++){
278                 integersToReturn[i]=selectebleElements.indexOf(selectedElements.get(i));
279             }
280 
281             return integersToReturn;
282         }
283 
284         // ****************************** actionListner setup ************************************
285 
286         public void addActionToClose(ActionListener a){
287             btnOk.addActionListener(a);
288         }
289 
290         public void removeActionFromClose(ActionListener a){
291             btnOk.removeActionListener(a);
292         }
293 
294         public JButton getBtnCloseWindow(){
295             return btnOk;
296         }
297 
298 
299         // ****************************** JFrame setup ************************************
300 
301         /**
302          * sets the elements for the miniJframe
303          */
304         private void JFrameSetup(String ElementNavn){
305 
```

```
306
307                 miniFrame = new JDialog();
308                 miniFrame.setModal(true);
309                 miniFrame.setTitle("ValgListe over "+ ElementNavn);
310                 miniFrame.setLocation(300, 200);
311                 miniFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
312                 miniFrame.setLayout(null);
313                 miniFrame.setSize(450, 300);
314                 miniFrame.setResizable(false);
315                 miniFrame.setVisible(false);
316                 miniFrame.toFront();
317
318
319                 lblSelectebleElements = new JLabel("Tilgaengelige "+ ElementNavn);
320                 Font LabelFontH1 = new Font(lblSelectebleElements.getFont().getName(),Font.BOLD,13);
321                 lblSelectebleElements.setFont(LabelFontH1);
322                 lblSelectebleElements.setSize(190, 25);
323                 lblSelectebleElements.setLocation(20, 25);
324                 miniFrame.add(lblSelectebleElements);
325
326                 lblSelctedElements = new JLabel("Valgte " + ElementNavn);
327                 lblSelctedElements.setFont(LabelFontH1);
328                 lblSelctedElements.setSize(150, 25);
329                 lblSelctedElements.setLocation(250, 25);
330                 miniFrame.add(lblSelctedElements);
331
332                 btnOk = new JButton("Ok");
333                 btnOk.setName("btnOk");
334                 btnOk.setSize(150, 25);
335                 btnOk.setLocation(140, 240);
336                 btnOk.addActionListener(btnController);
337                 miniFrame.add(btnOk);
338
339                 btnValgVTilE = new JButton(">>");
340                 btnValgVTilE.setName("btnValgVTilE");
341                 btnValgVTilE.setSize(50, 25);
342                 btnValgVTilE.setLocation(195, 90);
343                 btnValgVTilE.addActionListener(btnController);
344                 miniFrame.add(btnValgVTilE);
345
346                 btnFjernFraE = new JButton("<<");
347                 btnFjernFraE.setName("btnFjernFraE");
348                 btnFjernFraE.setSize(50, 25);
349                 btnFjernFraE.setLocation(195, 140);
350                 btnFjernFraE.addActionListener(btnController);
351                 miniFrame.add(btnFjernFraE);
352
353                 jlSelectebleElementsMF = new JList(selectebleElementsNames);
354                 jlSelectebleElementsMF.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION)
                        ;
355
356                 scSelectebleElementsMF = new JScrollPane(jlSelectebleElementsMF);
357                 scSelectebleElementsMF.setSize(170, 150);
358                 scSelectebleElementsMF.setLocation(20, 50);
359                 miniFrame.add(scSelectebleElementsMF);
360
361                 btnVaelgAlle = new JButton("Vaelg Alle");
362                 btnVaelgAlle.setName("btnCloseWindow");
363                 btnVaelgAlle.setSize(170, 25);
364                 btnVaelgAlle.setLocation(20, 205);
365                 btnVaelgAlle.addActionListener(btnController);
366                 miniFrame.add(btnVaelgAlle);
367
368                 jlSelectedElementsMF = new JList(selectedElementsNames);
369                 jlSelectedElementsMF.setSelectionMode(ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
370
371                 scSelctedElementsMF = new JScrollPane(jlSelectedElementsMF);
372                 scSelctedElementsMF.setSize(170, 150);
373                 scSelctedElementsMF.setLocation(250, 50);
374                 miniFrame.add(scSelctedElementsMF);
375
376                 btnVaelgIngen = new JButton("Vaelg Ingen");
377                 btnVaelgIngen.setName("btnCloseWindow");
378                 btnVaelgIngen.setSize(170, 25);
379                 btnVaelgIngen.setLocation(250, 205);
380                 btnVaelgIngen.addActionListener(btnController);
381                 miniFrame.add(btnVaelgIngen);
382
```

```
383         }
384 
385         // ***************************** btnController ************************************
386         /**
387          * actionsListener for our Jbuttons
388          */
389         private class BtnController implements ActionListener {
390 
391                 public void actionPerformed(ActionEvent e) {
392 
393                         if (e.getSource() == btnOpenMiniFrame) {
394                                 miniFrame.setVisible(true);
395                         }
396 
397                         if (e.getSource() == btnOk) {
398                                 miniFrame.setVisible(false);
399                         }
400 
401                         if (e.getSource() == btnValgVTilE) {
402                                 int[] indexs = jlSelectebleElementsMF.getSelectedIndices();
403                                 for (int i=indexs.length-1; i>=0;i--){
404                                         selectebleElements.remove(indexs[i]);
405                                         selectebleElementsNames.removeElementAt(indexs[i]);
406                                 }
407 
408                                 selectedElements.clear();
409                                 selectedElementsNames.clear();
410 
411                                 for (int i=0;i<origenList.size();i++){
412                                         if (!selectebleElements.contains(origenList.get(i))){
413                                                 selectedElements.add(origenList.get(i));
414                                                 selectedElementsNames.addElement(origenListNames.
                                                         elementAt(i));
415                                         }
416                                 }
417 
418                         }
419 
420                         if (e.getSource() == btnFjernFraE) {
421                                 int[] indexs = jlSelectedElementsMF.getSelectedIndices();
422                                 for (int i=indexs.length-1; i>=0;i--){
423                                         selectedElements.remove(indexs[i]);
424                                         selectedElementsNames.removeElementAt(indexs[i]);
425                                 }
426 
427                                 selectebleElements.clear();
428                                 selectebleElementsNames.clear();
429 
430                                 for (int i=0;i<origenList.size();i++){
431                                         if (!selectedElements.contains(origenList.get(i))){
432                                                 selectebleElements.add(origenList.get(i));
433                                                 selectebleElementsNames.addElement(origenListNames.
                                                         elementAt(i));
434                                         }
435                                 }
436 
437                         }
438 
439                         if (e.getSource() == btnVaelgAlle) {
440                                 selectebleElements.clear();
441                                 selectebleElementsNames.clear();
442 
443 
444                                 selectedElements.clear();
445                                 selectedElementsNames.clear();
446 
447                                 for (int i=0;i<origenList.size();i++){
448                                         if (!selectebleElements.contains(origenList.get(i))){
449                                                 selectedElements.add(origenList.get(i));
450                                                 selectedElementsNames.addElement(origenListNames.
                                                         elementAt(i));
451                                         }
452                                 }
453 
454                         }
455 
456                         if (e.getSource() == btnVaelgIngen) {
457                                 selectedElements.clear();
```

```
458                                 selectedElementsNames.clear();
459
460                                 selectebleElements.clear();
461                                 selectebleElementsNames.clear();
462
463                                 for (int i=0;i<origenList.size();i++){
464                                         if (!selectedElements.contains(origenList.get(i))){
465                                                 selectebleElements.add(origenList.get(i));
466                                                 selectebleElementsNames.addElement(origenListNames.
                                                        elementAt(i));
467                                         }
468                                 }
469
470                         }
471                 }
472         }
473
474 }
```

CreateDrying.java

```
1  package gui;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.List;
6
7  import javax.swing.JButton;
8  import javax.swing.JDialog;
9  import javax.swing.JFrame;
10 import javax.swing.JLabel;
11 import javax.swing.JOptionPane;
12 import javax.swing.JTextField;
13
14 import model.Depot;
15 import model.Drying;
16 import model.ProcessLine;
17
18 import service.Service;
19
20 /**
21  * @author M. C. H j
22  */
23
24 public class CreateDrying extends JDialog {
25
26         private static final long serialVersionUID = 1L;
27         private JLabel lblMinTime;
28         private JTextField txfMinTime;
29         private JLabel lblIdealTime;
30         private JTextField txfIdealTime;
31         private JLabel lblMaxTime;
32         private JTextField txfMaxTime;
33         private JLabel lblDepot;
34
35         private JButton btnCreate;
36         private JButton btnChancel;
37         private ProcessLine pl;
38         private Drying thisDrying;
39         private BtnController btnController = new BtnController();
40         private ChooseDepotsFrame msl;
41
42         public CreateDrying(ProcessLine pl) {
43                 initComponents();
44
45                 this.pl=pl;
46                 this.setModal(true);
47                 this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
48                 this.setResizable(false);
49                 this.setLocationRelativeTo(null);
50
51                 this.setVisible(true);
52         }
53
54         private void initComponents() {
55                 setTitle("Opret Toerreing");
56                 setLayout(null);
57                 add(getLblMinTime());
58                 add(getLblIdealTime());
59                 add(getLblMaxTime());
60                 add(getTxfMaxTime());
61                 add(getTxfIdealTime());
62                 add(getTxfMinTime());
63                 add(getLblDepot());
64                 add(getBtnChancel());
65                 add(getBtnCreate());
66                 add(getMsl());
67                 setSize(400, 235);
68         }
69
70         private ChooseDepotsFrame getMsl() {
71                 if (msl == null) {
72                         msl = new ChooseDepotsFrame("Lagre");
73                         msl.setLocation(194, 34);
74                         msl.setSize(190, 127);
75
```

```
76                      List<Depot> depoter= Service.getService().getAllDepots();
77
78                      for (int i = 0; i < depoter.size(); i++) {
79                              msl.add(depoter.get(i), false);
80                      }
81
82              }
83              return msl;
84      }
85
86      private JButton getBtnChancel() {
87              if (btnChancel == null) {
88                      btnChancel = new JButton();
89                      btnChancel.setText("Annuller");
90                      btnChancel.addActionListener(btnController);
91                      btnChancel.setLocation(194, 174);
92                      btnChancel.setSize(190, 25);
93              }
94              return btnChancel;
95      }
96
97      private JButton getBtnCreate() {
98              if (btnCreate == null) {
99                      btnCreate = new JButton();
100                     btnCreate.setText("Opret toerring");
101                     btnCreate.addActionListener(btnController);
102                     btnCreate.setLocation(12, 174);
103                     btnCreate.setSize(170, 25);
104             }
105             return btnCreate;
106     }
107
108     private JLabel getLblDepot() {
109             if (lblDepot == null) {
110                     lblDepot = new JLabel();
111                     lblDepot.setText("Lagre hvor toerringen skal foregaa:");
112                     lblDepot.setLocation(194, 12);
113                     lblDepot.setSize(190, 20);
114             }
115             return lblDepot;
116     }
117
118     private JTextField getTxfMaxTime() {
119             if (txfMaxTime == null) {
120                     txfMaxTime = new JTextField();
121                     txfMaxTime.setLocation(12, 142);
122                     txfMaxTime.setSize(170, 20);
123             }
124             return txfMaxTime;
125     }
126
127     private JLabel getLblMaxTime() {
128             if (lblMaxTime == null) {
129                     lblMaxTime = new JLabel();
130                     lblMaxTime.setText("Maximum toerretid (dage):");
131                     lblMaxTime.setLocation(12, 120);
132                     lblMaxTime.setSize(170, 20);
133             }
134             return lblMaxTime;
135     }
136
137     private JTextField getTxfIdealTime() {
138             if (txfIdealTime == null) {
139                     txfIdealTime = new JTextField();
140                     txfIdealTime.setLocation(12, 88);
141                     txfIdealTime.setSize(170, 20);
142             }
143             return txfIdealTime;
144     }
145
146     private JLabel getLblIdealTime() {
147             if (lblIdealTime == null) {
148                     lblIdealTime = new JLabel();
149                     lblIdealTime.setText("Ideal toerretid (dage):");
150                     lblIdealTime.setLocation(12, 66);
151                     lblIdealTime.setSize(170, 20);
152             }
153             return lblIdealTime;
```

```
154         }
155
156         private JTextField getTxfMinTime() {
157                 if (txfMinTime == null) {
158                         txfMinTime = new JTextField();
159                         txfMinTime.setLocation(12, 34);
160                         txfMinTime.setSize(170, 20);
161                 }
162                 return txfMinTime;
163         }
164
165         private JLabel getLblMinTime() {
166                 if (lblMinTime == null) {
167                         lblMinTime = new JLabel();
168                         lblMinTime.setText("Minimums toerretid (dage):");
169                         lblMinTime.setLocation(12, 12);
170                         lblMinTime.setSize(170, 20);
171                 }
172                 return lblMinTime;
173         }
174
175         /**
176          * returnere den drying som vi laver. Vil returnere null, hvis vi annullere
177          * @return
178          */
179         public Drying getDrying(){
180                 return thisDrying;
181         }
182
183         private class BtnController implements ActionListener {
184
185                 @Override
186                 public void actionPerformed(ActionEvent e) {
187
188                         if (e.getSource().equals(btnChancel)){
189                                 CreateDrying.this.setVisible(false);
190                         } else if (e.getSource().equals(btnCreate)){
191
192                                 if (msl.getSelectedElements().isEmpty()){
193                                         JOptionPane.showMessageDialog(null,"Toerringsprocessen skal
                                                vaere tilknyttet mindst et lager!!!","Fejl!!!",JOptionPane.
                                                ERROR_MESSAGE);
194                                 } else {
195                                         long minTime;
196                                         long idealTime;
197                                         long maxTime;
198                                         try {
199                                                 minTime = Long.valueOf(getTxfMinTime().getText())
                                                        *24*60*60*1000;
200                                                 idealTime = Long.valueOf(getTxfIdealTime().getText())
                                                        *24*60*60*1000;
201                                                 maxTime = Long.valueOf(getTxfMaxTime().getText())
                                                        *24*60*60*1000;
202
203                                                 thisDrying = pl.createDrying(0, minTime, idealTime,
                                                        maxTime);
204
205                                                 for (int i = 0; i < msl.getSelectedElements().size(); i
                                                        ++) {
206                                                         thisDrying.addDepot((Depot)msl.
                                                                getSelectedElements().get(i));
207                                                 }
208
209                                                 System.out.println(thisDrying.getDepots());
210
211                                                 CreateDrying.this.setVisible(false);
212                                         }
213                                         catch (NumberFormatException exception){
214                                                 JOptionPane.showMessageDialog(null,"Toerringentiderne
                                                        kan ikke godtages!!!","Fejl!!!",JOptionPane.
                                                        ERROR_MESSAGE);
215                                         }
216                                         catch (RuntimeException exception){
217                                                 JOptionPane.showMessageDialog(null,"Toerringentiderne
                                                        kan ikke godtages!!!","Fejl!!!",JOptionPane.
                                                        ERROR_MESSAGE);
218                                         }
219                                 }
```

```
220                         }
221                     }
222                 }
223 }
```

CreateIntermediateProductFrame.java

```
1  package gui;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5  import java.util.List;
6
7  import javax.swing.DefaultComboBoxModel;
8  import javax.swing.JButton;
9  import javax.swing.JComboBox;
10 import javax.swing.JDialog;
11 import javax.swing.JFrame;
12 import javax.swing.JLabel;
13 import javax.swing.JOptionPane;
14 import javax.swing.JScrollPane;
15 import javax.swing.JTextArea;
16 import javax.swing.JTextField;
17
18 import model.IntermediateProduct;
19 import model.ProductType;
20 import service.Service;
21
22 /**
23  * @author M. C. H j
24  */
25
26 public class CreateIntermediateProductFrame extends JDialog {
27
28         private static final long serialVersionUID = 1L;
29         private JLabel lblProduktType;
30         private JComboBox cmbProductType;
31         private JLabel lblQuantity;
32         private JTextField txfQuantity;
33         private JLabel lblId;
34         private JTextField txfId;
35         private JLabel img;
36         private JScrollPane scpInfo;
37         private JTextArea txaInfo;
38         private JButton btnCreate;
39         private JButton btnChancel;
40         private BtnController btnController = new BtnController();
41         private IntermediateProduct thisProduct = null;
42
43         public CreateIntermediateProductFrame() {
44                 initComponents();
45
46                 this.setModal(true);
47                 this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
48                 this.setResizable(false);
49                 this.setLocationRelativeTo(null);
50                 setInfo();
51
52                 this.setVisible(true);
53         }
54
55         private void initComponents() {
56                 setTitle("Opret mellemvare");
57                 setLayout(null);
58                 add(getLblProductType());
59                 add(getCmbProductType());
60                 add(getLblQuantity());
61                 add(getTxfQuantity());
62                 add(getLblId());
63                 add(getTxfId());
64                 add(getscpInfo());
65                 add(getBtnCreate());
66                 add(getBtnChancel());
67                 add(getImg());
68                 setSize(442, 273);
69         }
70
71         /**
72          * s tter infoen som er tilknyttet den valgte produkttype.
73          * s tter desuden st rrelsen p  tekstfeltet alt efter om producttypen er tilknyttet et
               billed
74          */
```

```
75          private void setInfo(){
76                  ProductType thisProductType = (ProductType) cmbProductType.getSelectedItem();
77
78                  if (thisProductType.getPicture()==null){
79
80                          img.setVisible(false);
81                          scpInfo.setBounds(174, 12, 250, 226);
82
83                  } else {
84
85                          img.setVisible(true);
86                          scpInfo.setBounds(174, 124, 250, 114);
87
88                  }
89
90                  img.setIcon(thisProductType.getPicture());
91
92                  String pInfo = "Beskrivelse:\n";
93                  pInfo = pInfo+thisProductType.getProcessLine().getDescription();
94                  pInfo = pInfo+"\n\nBehandlinger:";
95
96                  List<model.Process> processes = thisProductType.getProcessLine().getProcesses();
97                  for (int i = 0; i < processes.size(); i++) {
98                          pInfo=pInfo+"\n"+processes.get(i);
99                  }
100
101                 txaInfo.setText(pInfo);
102
103
104
105         }
106
107         private JButton getBtnChancel() {
108                 if (btnChancel == null) {
109                         btnChancel = new JButton();
110                         btnChancel.setText("Annuller");
111                         btnChancel.setBounds(12, 212, 150, 25);
112                         btnChancel.addActionListener(btnController);
113                 }
114                 return btnChancel;
115         }
116
117         private JButton getBtnCreate() {
118                 if (btnCreate == null) {
119                         btnCreate = new JButton();
120                         btnCreate.setText("Opret mellemvare");
121                         btnCreate.setBounds(12, 180, 150, 25);
122                         btnCreate.addActionListener(btnController);
123                 }
124                 return btnCreate;
125         }
126
127         private JTextArea getTxaInfo() {
128                 if (txaInfo == null) {
129                         txaInfo = new JTextArea();
130                         txaInfo.setEditable(false);
131                 }
132                 return txaInfo;
133         }
134
135         private JScrollPane getscpInfo() {
136                 if (scpInfo == null) {
137                         scpInfo = new JScrollPane(getTxaInfo());
138                         scpInfo.setBounds(174, 124, 250, 114);
139                 }
140                 return scpInfo;
141         }
142
143         private JLabel getImg() {
144                 if (img == null) {
145                         img = new JLabel();
146                         img.setBounds(229, 12, 140, 100);
147                 }
148                 return img;
149         }
150
151         private JTextField getTxfQuantity() {
152                 if (txfQuantity == null) {
```

```
153                     txfQuantity = new JTextField();
154                     txfQuantity.setBounds(12, 148, 150, 20);
155             }
156             return txfQuantity;
157     }
158
159     private JLabel getLblQuantity() {
160             if (lblQuantity == null) {
161                     lblQuantity = new JLabel();
162                     lblQuantity.setText("Mellemvare maengde:");
163                     lblQuantity.setBounds(12, 125, 150, 20);
164             }
165             return lblQuantity;
166     }
167
168     private JTextField getTxfId() {
169             if (txfId == null) {
170                     txfId = new JTextField();
171                     txfId.setBounds(12, 93, 150, 20);
172             }
173             return txfId;
174     }
175
176     private JLabel getLblId() {
177             if (lblId == null) {
178                     lblId = new JLabel();
179                     lblId.setText("Mellemvarens id:");
180                     lblId.setBounds(12, 71, 150, 20);
181             }
182             return lblId;
183     }
184
185     private JComboBox getCmbProductType() {
186             if (cmbProductType == null) {
187                     cmbProductType = new JComboBox();
188                     cmbProductType.setModel(new DefaultComboBoxModel(new Object[] {}));
189                     cmbProductType.setDoubleBuffered(false);
190                     cmbProductType.setBorder(null);
191                     cmbProductType.setBounds(12, 34, 150, 25);
192
193                     List<ProductType> productTypes= Service.getService().getAllProductTypes();
194
195                     for (int i = 0; i < productTypes.size(); i++) {
196                             cmbProductType.addItem(productTypes.get(i));
197                     }
198
199                     cmbProductType.setSelectedIndex(0);
200
201                     cmbProductType.addActionListener(btnController);
202
203             }
204             return cmbProductType;
205     }
206
207     private JLabel getLblProductType() {
208             if (lblProduktType == null) {
209                     lblProduktType = new JLabel();
210                     lblProduktType.setText("Vaelg produkttype:");
211                     lblProduktType.setBounds(12, 12, 150, 20);
212             }
213             return lblProduktType;
214     }
215
216     private class BtnController implements ActionListener {
217
218             @Override
219             public void actionPerformed(ActionEvent e) {
220
221                     if (e.getSource().equals(btnChancel)){
222                             CreateIntermediateProductFrame.this.setVisible(false);
223
224                     } else if (e.getSource().equals(btnCreate)){
225
226                             try {
227
228                                     if (txfId.getText().isEmpty()){
229                                             JOptionPane.showMessageDialog(null,"Mellemvaren skal
                                                 have en id","Fejl!!!",JOptionPane.ERROR_MESSAGE);
```

```
230                                     } else {
231                                             thisProduct = new IntermediateProduct(txfId.getText(),
                                                    (ProductType) cmbProductType.getSelectedItem(),
                                                    Double.valueOf(txfQuantity.getText()));
232                                             CreateIntermediateProductFrame.this.setVisible(false);
233                                     }
234
235                                 }catch (NumberFormatException exception){
236                                     JOptionPane.showMessageDialog(null,"M ngden skal v re et tal"
                                            ,"Fejl!!!",JOptionPane.ERROR_MESSAGE);
237                                 }
238                                 catch (RuntimeException exception){
239                                     JOptionPane.showMessageDialog(null,"M ngden skal v re st rre
                                            end 0","Fejl!!!",JOptionPane.ERROR_MESSAGE);
240                                 }
241
242                             } else if (e.getSource().equals(cmbProductType)){
243
244                                     setInfo();
245
246                             }
247
248                     }
249             }
250
251             /**
252              * returnere den mellemvare som vi laver. Vil returnere null, hvis vi annullere
253              * @return
254              */
255             public IntermediateProduct getIntermediateProduct() {
256                     return thisProduct;
257             }
258
259     }
```

CreateProductTypeFrame.java

```
1  package gui;
2
3  import java.awt.Color;
4  import java.awt.Font;
5  import java.awt.event.ActionEvent;
6  import java.awt.event.ActionListener;
7  import java.io.File;
8
9  import javax.swing.DefaultListModel;
10 import javax.swing.DefaultListSelectionModel;
11 import javax.swing.ImageIcon;
12 import javax.swing.JButton;
13 import javax.swing.JDialog;
14 import javax.swing.JFrame;
15 import javax.swing.JLabel;
16 import javax.swing.JList;
17 import javax.swing.JOptionPane;
18 import javax.swing.JScrollPane;
19 import javax.swing.JTextArea;
20 import javax.swing.JTextField;
21
22 import model.Process;
23 import model.ProcessLine;
24 import model.ProductType;
25
26 /**
27  * @author M. C. H j
28  */
29
30 public class CreateProductTypeFrame extends JDialog {
31
32         private static final long serialVersionUID = 1L;
33         private JLabel lblName;
34         private JTextField txfName;
35         private JButton btnCreateProductType;
36         private JLabel lblDescription;
37         private JScrollPane scpDescription;
38         private JTextArea txaDescription;
39         private JLabel lblProcesses;
40         private JButton btnCreateDrying;
41         private JButton btnCreateSubProcess;
42         private JButton btnDeleteProcess;
43         private JButton btnMoveUp;
44         private JButton btnMoveDown;
45         private JButton btnChancel;
46         private JList listProcesses;
47         private JScrollPane scpProcesses;
48         private BtnController btnController = new BtnController();
49         private ProductType thisProductType = null;
50         private ProductType productTypeToReturn;
51         private DefaultListModel listProcessesModel = new DefaultListModel();
52         private JButton btnPicture;
53
54
55         public CreateProductTypeFrame() {
56                 initComponents();
57
58                 this.setModal(true);
59                 this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
60                 this.setLocationRelativeTo(null);
61                 this.setResizable(false);
62
63                 thisProductType = new ProductType("");
64                 ProcessLine pL = new ProcessLine("", "", thisProductType);
65
66                 this.setProcessBtns(false);
67                 this.setVisible(true);
68         }
69
70         private void initComponents() {
71                 setTitle("Opret produkttype");
72                 setLayout(null);
73                 add(getLblName());
74                 add(getLblDescription());
75                 add(getBtnChancel());
```

```
76                  add(getTxfName());
77                  add(getBtnCreateDrying());
78                  add(getBtnCreateSubProcess());
79                  add(getBtnDeleteProcess());
80                  add(getBtnMoveDown());
81                  add(getBtnMoveUp());
82                  add(getScpProcesses());
83                  add(getLblProcesses());
84                  add(getBtnCreateProductType());
85                  add(getBtnPicture());
86                  add(getScpDescription());
87                  setSize(548, 312);
88          }
89
90          private JButton getBtnPicture() {
91                  if (btnPicture == null) {
92                          btnPicture = new JButton();
93                          btnPicture.setText("Tilknyt et billede");
94                          btnPicture.addActionListener(btnController);
95                          btnPicture.setLocation(12, 212);
96                          btnPicture.setSize(250, 25);
97                  }
98                  return btnPicture;
99          }
100
101         private JScrollPane getScpProcesses() {
102                 if (scpProcesses == null) {
103                         scpProcesses = new JScrollPane();
104                         scpProcesses.setViewportView(getListProcesses());
105                         scpProcesses.setLocation(280, 34);
106                         scpProcesses.setSize(158, 113);
107                 }
108                 return scpProcesses;
109         }
110
111         private JList getListProcesses() {
112                 if (listProcesses == null) {
113                         listProcesses = new JList();
114                         listProcesses.setSelectionMode(DefaultListSelectionModel.SINGLE_SELECTION);
115                         listProcesses.setModel(listProcessesModel);
116                 }
117                 return listProcesses;
118         }
119
120         private JButton getBtnDeleteProcess() {
121                 if (btnDeleteProcess == null) {
122                         btnDeleteProcess = new JButton();
123                         btnDeleteProcess.setText("Slet delbehandling");
124                         btnDeleteProcess.addActionListener(btnController);
125                         btnDeleteProcess.setLocation(280, 152);
126                         btnDeleteProcess.setSize(250, 25);
127                 }
128                 return btnDeleteProcess;
129         }
130
131         private JButton getBtnMoveUp() {
132                 if (btnMoveUp == null) {
133                         btnMoveUp = new JButton();
134                         btnMoveUp.setText("Ryk op");
135                         btnMoveUp.addActionListener(btnController);
136                         btnMoveUp.setLocation(450, 58);
137                         btnMoveUp.setSize(80, 25);
138                 }
139                 return btnMoveUp;
140         }
141
142         private JButton getBtnMoveDown() {
143                 if (btnMoveDown == null) {
144                         btnMoveDown = new JButton();
145                         btnMoveDown.setText("Ryk ned");
146                         btnMoveDown.addActionListener(btnController);
147                         btnMoveDown.setLocation(450, 98);
148                         btnMoveDown.setSize(80, 25);
149                 }
150                 return btnMoveDown;
151         }
152
153         private JButton getBtnChancel() {
```

```
154                 if (btnChancel == null) {
155                         btnChancel = new JButton();
156                         btnChancel.setText("Annuller");
157                         btnChancel.addActionListener(btnController);
158                         btnChancel.setLocation(280, 249);
159                         btnChancel.setSize(250, 25);
160                 }
161                 return btnChancel;
162         }
163
164         private JButton getBtnCreateSubProcess() {
165                 if (btnCreateSubProcess == null) {
166                         btnCreateSubProcess = new JButton();
167                         btnCreateSubProcess.setText("Opret behandling");
168                         btnCreateSubProcess.addActionListener(btnController);
169                         btnCreateSubProcess.setLocation(280, 182);
170                         btnCreateSubProcess.setSize(250, 25);
171                 }
172                 return btnCreateSubProcess;
173         }
174
175         private JButton getBtnCreateDrying() {
176                 if (btnCreateDrying == null) {
177                         btnCreateDrying = new JButton();
178                         btnCreateDrying.setText("Opret toerring");
179                         btnCreateDrying.addActionListener(btnController);
180                         btnCreateDrying.setLocation(280, 212);
181                         btnCreateDrying.setSize(250, 25);
182                 }
183                 return btnCreateDrying;
184         }
185
186         private JLabel getLblProcesses() {
187                 if (lblProcesses == null) {
188                         lblProcesses = new JLabel();
189                         lblProcesses.setText("Produkttypens delbehandlinger:");
190                         lblProcesses.setLocation(280, 12);
191                         lblProcesses.setSize(200, 20);
192                 }
193                 return lblProcesses;
194         }
195
196         private JScrollPane getScpDescription() {
197                 if (scpDescription == null) {
198                         scpDescription = new JScrollPane();
199                         scpDescription.setViewportView(getTxaDescription());
200                         scpDescription.setLocation(12, 88);
201                         scpDescription.setSize(250, 120);
202                 }
203                 return scpDescription;
204         }
205
206         private JTextArea getTxaDescription() {
207                 if (txaDescription == null) {
208                         txaDescription = new JTextArea();
209                 }
210                 return txaDescription;
211         }
212
213         private JLabel getLblDescription() {
214                 if (lblDescription == null) {
215                         lblDescription = new JLabel();
216                         lblDescription.setText("Beskrivelse:");
217                         lblDescription.setLocation(12, 66);
218                         lblDescription.setSize(250, 20);
219                 }
220                 return lblDescription;
221         }
222
223         private JButton getBtnCreateProductType() {
224                 if (btnCreateProductType == null) {
225                         btnCreateProductType = new JButton();
226                         btnCreateProductType.setText("Opret produkttype");
227                         btnCreateProductType.addActionListener(btnController);
228                         btnCreateProductType.setLocation(12, 249);
229                         btnCreateProductType.setSize(250, 25);
230                 }
231                 return btnCreateProductType;
```

```
232         }
233 
234         private JTextField getTxfName() {
235                 if (txfName == null) {
236                         txfName = new JTextField();
237                         txfName.setLocation(12, 34);
238                         txfName.setSize(250, 20);
239                 }
240                 return txfName;
241         }
242 
243         private JLabel getLblName() {
244                 if (lblName == null) {
245                         lblName = new JLabel();
246                         lblName.setText("Produkttypens navn:");
247                         lblName.setLocation(12, 12);
248                         lblName.setSize(250, 20);
249                 }
250                 return lblName;
251         }
252 
253         private class BtnController implements ActionListener {
254 
255                 @Override
256                 public void actionPerformed(ActionEvent e) {
257 
258                         if (e.getSource().equals(btnCreateDrying)){
259                                 CreateDrying cd = new CreateDrying(thisProductType.getProcessLine());
260                                 if (cd.getDrying()!=null){
261                                         listProcessesModel.addElement(cd.getDrying());
262                                         CreateProductTypeFrame.this.setProcessBtns(true);
263                                 }
264                         } else if (e.getSource().equals(btnCreateSubProcess)){
265                                 CreateSubProcess csp = new CreateSubProcess(thisProductType.
                                        getProcessLine());
266                                 if (csp.getSubProcess()!=null){
267                                         listProcessesModel.addElement(csp.getSubProcess());
268                                         CreateProductTypeFrame.this.setProcessBtns(true);
269                                 }
270                         } else if (e.getSource().equals(btnPicture)){
271 
272 
273                                 if (thisProductType.getPicture()==null){
274                                         File activeFile;
275 
276                                         EditorFileHandler choosenfil = new EditorFileHandler(
                                                EditorFileHandler.LOAD_FUNCTION, new File(System.
                                                getProperty("user.dir")+"\\gui\\icons"));
277                                         if (choosenfil.getIsOkPressed()){
278                                                 activeFile = choosenfil.getSelectedFile();
279 
280                                                 thisProductType.setPicture(new ImageIcon(activeFile.
                                                        getPath()));
281                                                 btnPicture.setText("Fjern billede ("+activeFile.getPath
                                                        ()+")");
282                                         }
283                                 } else {
284                                         thisProductType.setPicture(null);
285                                         btnPicture.setText("Tilknyt et billede");
286                                 }
287 
288 
289                         } else if (e.getSource().equals(btnDeleteProcess)){
290 
291                                 int index = listProcesses.getSelectedIndex();
292 
293                                 if (index>=0){
294 
295 
296                                         listProcessesModel.remove(index);
297                                         thisProductType.getProcessLine().getProcesses().remove(index);
298 
299                                         if (listProcessesModel.size()<1){
300                                                 CreateProductTypeFrame.this.setProcessBtns(false);
301                                         }
302 
303                                 }
304 
```

```
305
306
307                     } else if (e.getSource().equals(btnMoveDown)){
308                             int index = listProcesses.getSelectedIndex();
309
310                             if (index>=0 && index<listProcessesModel.size()-1){
311                                     Process elementTop = thisProductType.getProcessLine().
                                            getProcesses().get(index);
312                                     Process elementBottom = thisProductType.getProcessLine().
                                            getProcesses().get(index+1);
313
314                                     thisProductType.getProcessLine().getProcesses().set(index+1,
                                            elementTop);
315                                     listProcessesModel.set(index+1, elementTop);
316
317                                     thisProductType.getProcessLine().getProcesses().set(index,
                                            elementBottom);
318                                     listProcessesModel.set(index, elementBottom);
319
320                                     listProcesses.setSelectedIndex(index+1);
321                             }
322
323                     } else if (e.getSource().equals(btnMoveUp)){
324
325                             int index = listProcesses.getSelectedIndex();
326
327                             if (index>0){
328                                     Process elementTop = thisProductType.getProcessLine().
                                            getProcesses().get(index-1);
329                                     Process elementBottom = thisProductType.getProcessLine().
                                            getProcesses().get(index);
330
331                                     thisProductType.getProcessLine().getProcesses().set(index,
                                            elementTop);
332                                     listProcessesModel.set(index, elementTop);
333
334                                     thisProductType.getProcessLine().getProcesses().set(index-1,
                                            elementBottom);
335                                     listProcessesModel.set(index-1, elementBottom);
336                                     listProcesses.setSelectedIndex(index-1);
337                             }
338
339                     } else if (e.getSource().equals(btnCreateProductType)){
340
341                             if (getTxfName().getText().isEmpty()){
342                                     JOptionPane.showMessageDialog(null,"Produkttypen skal have et
                                            navn!!!","Fejl!!!",JOptionPane.ERROR_MESSAGE);
343
344                             } else {
345
346
347                                     for (int i=0; listProcessesModel.size()>i;i++){
348                                             Process p = (Process)listProcessesModel.get(0);
349                                             p.setProcessStep(i+1);
350                                     }
351
352                                     thisProductType.setName(txfName.getText());
353                                     ProcessLine pl = thisProductType.getProcessLine();
354                                     pl.setName(txfName.getText());
355                                     pl.setDescription(txaDescription.getText());
356
357                                     productTypeToReturn=thisProductType;
358
359                                     CreateProductTypeFrame.this.setVisible(false);
360                             }
361                     } else if (e.getSource().equals(btnChancel)){
362                             CreateProductTypeFrame.this.setVisible(false);
363                     }
364
365             }
366
367     }
368
369     /**
370      * returnere den producttype som vi laver. Vil returnere null, hvis vi annullere
371      * @return
372      */
373     public ProductType getProductType(){
```

```
374                     return productTypeToReturn;
375             }
376
377             private void setProcessBtns(boolean b){
378                     btnDeleteProcess.setEnabled(b);
379                     btnMoveUp.setEnabled(b);
380                     btnMoveDown.setEnabled(b);
381             }
382
383     }
```

CreateSubProcess.java

```
1  package gui;
2
3  import java.awt.event.ActionEvent;
4  import java.awt.event.ActionListener;
5
6  import javax.swing.JButton;
7  import javax.swing.JDialog;
8  import javax.swing.JFrame;
9  import javax.swing.JLabel;
10 import javax.swing.JOptionPane;
11 import javax.swing.JScrollPane;
12 import javax.swing.JTextArea;
13 import javax.swing.JTextField;
14
15 import model.ProcessLine;
16 import model.SubProcess;
17
18 /**
19  * @author M. C. H j
20  */
21
22 public class CreateSubProcess extends JDialog {
23
24         private static final long serialVersionUID = 1L;
25         private JLabel lblName;
26         private JTextField txfName;
27         private JLabel lblTemp;
28         private JTextField txfTemp;
29         private JLabel lblTime;
30         private JTextField txfTime;
31         private JLabel lblDescr;
32         private JButton btnCreate;
33         private JButton btnChancel;
34         private JTextArea txaDescr;
35         private JScrollPane scpDescr;
36         private ProcessLine pl;
37         private SubProcess thisSubProcess = null;
38         private BtnController btnController = new BtnController();
39         public CreateSubProcess(ProcessLine pl) {
40                 initComponents();
41
42                 this.pl=pl;
43                 this.setModal(true);
44                 this.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
45                 this.setResizable(false);
46                 this.setLocationRelativeTo(null);
47                 this.setVisible(true);
48         }
49
50         private void initComponents() {
51                 setTitle("Opret behandling");
52                 setLayout(null);
53                 add(getLblName());
54                 add(getLblTemp());
55                 add(getLblTime());
56                 add(getLblDescr());
57                 add(getBtnCreate());
58                 add(getBtnChancel());
59                 add(getScpDescr());
60                 add(getTxfTime());
61                 add(getTxfTemp());
62                 add(getTxfName());
63                 setSize(312, 378);
64         }
65
66         private JScrollPane getScpDescr() {
67                 if (scpDescr == null) {
68                         scpDescr = new JScrollPane();
69                         scpDescr.setViewportView(getTxaDescr());
70                         scpDescr.setLocation(12, 192);
71                         scpDescr.setSize(282, 120);
72                 }
73                 return scpDescr;
74         }
75
```

```
76          private JTextArea getTxaDescr() {
77                  if (txaDescr == null) {
78                          txaDescr = new JTextArea();
79                  }
80                  return txaDescr;
81          }
82
83          private JButton getBtnChancel() {
84                  if (btnChancel == null) {
85                          btnChancel = new JButton();
86                          btnChancel.setText("Annuller");
87                          btnChancel.addActionListener(btnController);
88                          btnChancel.setLocation(159, 316);
89                          btnChancel.setSize(135, 25);
90                  }
91                  return btnChancel;
92          }
93
94          private JButton getBtnCreate() {
95                  if (btnCreate == null) {
96                          btnCreate = new JButton();
97                          btnCreate.setText("Opret behandling");
98                          btnCreate.addActionListener(btnController);
99                          btnCreate.setLocation(12, 316);
100                         btnCreate.setSize(135, 25);
101
102                 }
103                 return btnCreate;
104         }
105
106         private JLabel getLblDescr() {
107                 if (lblDescr == null) {
108                         lblDescr = new JLabel();
109                         lblDescr.setText("Beskrivelse:");
110                         lblDescr.setLocation(12, 174);
111                         lblDescr.setSize(282, 20);
112                 }
113                 return lblDescr;
114         }
115
116         private JTextField getTxfTime() {
117                 if (txfTime == null) {
118                         txfTime = new JTextField("24");
119                         txfTime.setLocation(12, 142);
120                         txfTime.setSize(282, 20);
121                 }
122                 return txfTime;
123         }
124
125         private JLabel getLblTime() {
126                 if (lblTime == null) {
127                         lblTime = new JLabel();
128                         lblTime.setText("Behandlingens tid (timer):");
129                         lblTime.setLocation(12, 120);
130                         lblTime.setSize(282, 20);
131                 }
132                 return lblTime;
133         }
134
135         private JTextField getTxfTemp() {
136                 if (txfTemp == null) {
137                         txfTemp = new JTextField("15");
138                         txfTemp.setLocation(12, 88);
139                         txfTemp.setSize(282, 20);
140                 }
141                 return txfTemp;
142         }
143
144         private JLabel getLblTemp() {
145                 if (lblTemp == null) {
146                         lblTemp = new JLabel();
147                         lblTemp.setText("Behandlingengs temperatur (C):");
148                         lblTemp.setLocation(12, 66);
149                         lblTemp.setSize(282, 20);
150                 }
151                 return lblTemp;
152         }
153
```

```
154         private JTextField getTxfName() {
155                 if (txfName == null) {
156                         txfName = new JTextField();
157                         txfName.setLocation(12, 34);
158                         txfName.setSize(282, 20);
159                 }
160                 return txfName;
161         }
162 
163         private JLabel getLblName() {
164                 if (lblName == null) {
165                         lblName = new JLabel();
166                         lblName.setText("Behandlingsens navn:");
167                         lblName.setLocation(12, 12);
168                         lblName.setSize(282, 20);
169                 }
170                 return lblName;
171         }
172 
173         /**
174          * returnere den delbehandling som vi laver. Vil returnere null, hvis vi annullere
175          * @return
176          */
177         public SubProcess getSubProcess(){
178                 return thisSubProcess;
179         }
180 
181         private class BtnController implements ActionListener {
182 
183                 @Override
184                 public void actionPerformed(ActionEvent e) {
185 
186                         if (e.getSource().equals(btnChancel)){
187                                 CreateSubProcess.this.setVisible(false);
188                         } else if (e.getSource().equals(btnCreate)){
189 
190                                 if (getTxfName().getText().isEmpty()){
191                                         JOptionPane.showMessageDialog(null,"Behandlingen skal have et
                                                navn!!!","Fejl!!!",JOptionPane.ERROR_MESSAGE);
192 
193                                 } else {
194                                         long time;
195 
196                                         try {
197                                                 time = Long.valueOf(getTxfTime().getText())*60*60*1000;
198                                         }
199                                         catch (NumberFormatException exception){
200                                                 time = 24*60*60*1000;
201                                         }
202 
203                                         double temp;
204 
205                                         try {
206                                                 temp = Double.valueOf(getTxfTemp().getText());
207                                         }
208                                         catch (NumberFormatException exception){
209                                                 temp = 15.0;
210                                         }
211 
212 
213                                         thisSubProcess = pl.createSubProcess(0, getTxfName().getText(),
                                                getTxaDescr().getText(),time , temp);
214 
215                                         CreateSubProcess.this.setVisible(false);
216                                 }
217                         }
218 
219                 }
220         }
221 
222 
223 }
```

EditorFileFilter.java

```
1  package gui;
2
3  import java.io.File;
4
5  import javax.swing.filechooser.FileFilter;
6
7  /**
8   *  g r  os i stand til kun at se .java filer
9   * @author M. C.  H j
10  *
11  */
12 public class EditorFileFilter extends FileFilter{
13
14         @Override
15         public boolean accept(File f) {
16                 return f.getName().toLowerCase().endsWith(".jpg")||f.getName().toLowerCase().endsWith("
                       .gif")||f.getName().toLowerCase().endsWith(".jpeg")||f.getName().toLowerCase().
                       endsWith(".png") || f.isDirectory();
17         }
18
19         @Override
20         public String getDescription() {
21                 return "Billed filer";
22         }
23
24 }
```

EditorFileHandler.java

```
1  package gui;
2
3  import java.io.File;
4
5  import javax.swing.JFileChooser;
6  import javax.swing.JFrame;
7
8  @SuppressWarnings("serial")
9  /**
10  * vores jdialog som gemmer og loader filer
11  * @author M. C. H j
12  */
13 public class EditorFileHandler extends JFileChooser{
14
15         public static int LOAD_FUNCTION=0;
16         public static int SAVE_FUNCTION=1;
17         private int isOkPressed;
18
19         public boolean getIsOkPressed() {
20                 return isOkPressed == JFileChooser.APPROVE_OPTION;
21         }
22
23         /**
24          *
25          * @param function save or load function
26          * @param f file to choose
27          * @throws RuntimeException
28          */
29         public EditorFileHandler(int function, File f) throws RuntimeException{
30                 this.removeChoosableFileFilter(this.getChoosableFileFilters()[0]);
31                 this.setFileFilter(new EditorFileFilter());
32                 if (function == LOAD_FUNCTION){
33                         this.setCurrentDirectory(f);
34                         isOkPressed = this.showOpenDialog(new JFrame());
35                 } else if (function == SAVE_FUNCTION){
36                         this.setCurrentDirectory(f);
37                         this.setSelectedFile(f);
38                         isOkPressed = this.showSaveDialog(new JFrame());
39
40                 } else {
41                         throw new RuntimeException("Function must be save og load");
42                 }
43         }
44
45 }
```

IntermediateProductPanel.java

```
1  package gui;
2
3  import java.awt.*;
4
5  import javax.swing.BorderFactory;
6  import javax.swing.Box;
7  import javax.swing.BoxLayout;
8  import javax.swing.JComboBox;
9
10 import javax.swing.JLabel;
11 import javax.swing.JPanel;
12 import javax.swing.JProgressBar;
13
14 import model.Drying;
15 import model.StoringSpace;
16
17 /**
18  * @author Brian
19  */
20
21 public class IntermediateProductPanel extends JPanel{
22         private JPanel panel;
23         private JProgressBar progressBar;
24         private JLabel lblName, lblIcon;
25
26
27         private StoringSpace storingSpace;
28         private Boolean selected;
29
30
31
32         public IntermediateProductPanel(StoringSpace storingSpace) {
33                 this.storingSpace = storingSpace;
34                 this.setBorder(BorderFactory.createLineBorder(Color.black));
35                 this.setLayout(new BorderLayout());
36                 {
37                         panel = new JPanel();
38                         panel.setLayout(new BoxLayout(panel, BoxLayout.Y_AXIS));
39                         this.add(panel);
40                         {
41                                 lblName = new JLabel();
42                                 lblName.setAlignmentX(Component.CENTER_ALIGNMENT);
43                                 panel.add(lblName);
44                         }
45                         {
46                                 progressBar = new JProgressBar();
47                                 panel.add(progressBar);
48                         }
49                         {
50                                 panel.add(Box.createRigidArea(new Dimension(5,5)));
51                         }
52                         {
53                                 lblIcon = new JLabel();
54                                 lblIcon.setAlignmentX(JLabel.CENTER_ALIGNMENT);
55                                 panel.add(lblIcon);
56                         }
57                 }
58
59                 if(storingSpace.getIntermediateProduct() != null) {
60                         lblName.setText(storingSpace.getIntermediateProduct().getProductType().getName
                                ());
61                         lblIcon.setIcon(storingSpace.getIntermediateProduct().getProductType().
                                getPicture());
62                 }
63                 else {
64                         progressBar.setVisible(false);
65                         lblName.setVisible(false);
66                         lblIcon.setVisible(false);
67                 }
68
69                 updateTime();
70         }
71
72         public void updateTime() {
73                 if(storingSpace.getIntermediateProduct() != null) {
```

```
74                          Drying drying = (Drying) storingSpace.getIntermediateProduct().
                                getActivProcessLog().getProcess();
75                          progressBar.setMinimum(0);
76                          progressBar.setMaximum((int) drying.getMaxTime()/1000);
77                          long currentTime = System.currentTimeMillis()-storingSpace.
                                getIntermediateProduct().getActivProcessLog().getStartTime().getTime();
78                          progressBar.setValue((int) currentTime/1000);
79
80                          if (drying.getMinTime()>currentTime){
81                                  progressBar.setForeground(new Color(210,210,210));
82                          } else if ((drying.getMinTime()+(drying.getIdealTime()-drying.getMinTime())/2)>
                                currentTime){
83                                  progressBar.setForeground(Color.yellow);
84                          } else if ((drying.getIdealTime()+(drying.getMaxTime()-drying.getIdealTime())
                                /2)>currentTime){
85                                  progressBar.setForeground(Color.green);
86                          } else if (drying.getMaxTime()>currentTime) {
87                                  progressBar.setForeground(Color.red);
88                          } else {
89                                  panel.setOpaque(true);
90                                  panel.setBackground(Color.red);
91                                  progressBar.setForeground(Color.red);
92                          }
93
94                  } else {
95                          panel.setOpaque(false);;
96                  }
97          }
98
99          public void setSelected(Boolean bool) {
100                 if (bool) {
101                         this.setBorder(BorderFactory.createLineBorder(Color.orange, 3));
102                 }
103                 else {
104                         this.setBorder(BorderFactory.createLineBorder(Color.black));
105                 }
106         }
107
108         public StoringSpace getStoringSpace() {
109                 return this.storingSpace;
110         }
111
112 }
```

MainFrameApp.java

```
1  package gui;
2  import javax.swing.JFrame;
3  import javax.swing.UIManager;
4
5  /**
6   * @author Brian, Carl
7   */
8
9  public class MainFrameApp {
10
11         public static void main(String[] args) {
12                 try {
13                         UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
14                 }
15                 catch (Exception e) {
16                         System.out.println("Error setting look and feel: " + e.getMessage());
17                 }
18
19                 MainFrame frame = new MainFrame();
20                 frame.setExtendedState(frame.getExtendedState()|JFrame.MAXIMIZED_BOTH); //Maximer
                       framet hved starten
21
22                 frame.setVisible(true);
23         }
24 }
```

MainFrame.java

```
1  package gui;
2
3  import java.awt.BorderLayout;
4  import java.awt.Component;
5  import java.awt.Dimension;
6  import java.awt.FlowLayout;
7  import java.awt.Font;
8  import java.awt.GridLayout;
9  import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.awt.event.KeyEvent;
12 import java.awt.event.MouseEvent;
13 import java.awt.event.MouseListener;
14 import java.awt.event.WindowEvent;
15 import java.awt.event.WindowListener;
16 import java.util.ArrayList;
17 import java.util.List;
18
19 import javax.swing.BorderFactory;
20 import javax.swing.BoxLayout;
21 import javax.swing.DefaultListModel;
22 import javax.swing.JButton;
23 import javax.swing.JFrame;
24 import javax.swing.JLabel;
25 import javax.swing.JList;
26 import javax.swing.JMenu;
27 import javax.swing.JMenuBar;
28 import javax.swing.JMenuItem;
29 import javax.swing.JOptionPane;
30 import javax.swing.JPanel;
31 import javax.swing.JScrollPane;
32 import javax.swing.JTextField;
33 import javax.swing.ListModel;
34 import javax.swing.ListSelectionModel;
35 import javax.swing.event.DocumentEvent;
36 import javax.swing.event.DocumentListener;
37 import javax.swing.event.ListSelectionEvent;
38 import javax.swing.event.ListSelectionListener;
39
40 import model.Depot;
41 import model.Drying;
42 import model.IntermediateProduct;
43 import model.Process;
44 import model.StoringSpace;
45 import service.Service;
46 import sun.awt.WindowClosingListener;
47
48 /**
49  * @author Carl, Brian
50  */
51
52 public class MainFrame extends JFrame {
53
54         private static final long serialVersionUID = -1752000392799508369L;
55         {
56                 //Set Look & Feel
57                 try {
58                         javax.swing.UIManager.setLookAndFeel("com.jgoodies.looks.plastic.
                            Plastic3DLookAndFeel");
59                 } catch(Exception e) {
60                         e.printStackTrace();
61                 }
62         }
63         //MENU
64         private JMenuBar mnbBar;
65         private JMenu mnuView, mnuCreate;
66         private JMenuItem mitCreateProductType, mitCreateIntermediateProduct, mitViewDepot;
67         private ArrayList<JMenuItem> mitDepots = new ArrayList<JMenuItem>();
68         // WEST
69         private JPanel pnlWest;
70         private JLabel lblIntermediateProduct;
71         private JTextField txfSearch;
72         private JScrollPane scpIntermediateProducts;
73         private JList lstIntermediateProducts;
74         private JButton btnCreateIntermediateProduct;
```

```
75          //CENTER
76          private JPanel pnlCenter;
77          private JPanel pnlSelectedDepot;
78          private JLabel lblSelectedDepot;
79          private JPanel pnlIntermediateProductMap;
80          private GridLayout lytIntermediateProductMap = new GridLayout();
81          private ArrayList<IntermediateProductPanel> intermediateProductPanels = new ArrayList<
                IntermediateProductPanel>();
82          //EAST
83          private JPanel pnlEast;
84          private JLabel lblInformation;
85          private JLabel lblID, lblQuantity, lblProductType, lblDepot, lblCoordinates;
86          private JTextField txfID, txfQuantity, txfProductType, txfDepot, txfCoordinates;
87          private JPanel pnlProcessOverView;
88          private ArrayList<ProcessPanel> processPanels = new ArrayList<ProcessPanel>();
89          private JButton btnSendToNextProcess, btnDeleteIntermediateProduct;
90
91          private IntermediateProductPanel selectedIntermediateProductPanel = null;
92          private CreateProductTypeFrame createProductTypeFrame;
93          private CreateIntermediateProductFrame createIntermediateProduct;
94          private Controller controller = new Controller();
95          private UpdateTimer updateTimer;
96
97          public MainFrame() {
98                  this.addWindowListener(controller);
99                  this.setTitle("Carletti v1.1");
100                 BorderLayout thisLayout = new BorderLayout();
101                 getContentPane().setLayout(thisLayout);
102                 this.setResizable(true);
103                 this.setPreferredSize(new Dimension(600,600));
104                 {
105                         pnlWest = new JPanel();
106                         getContentPane().add(pnlWest, BorderLayout.WEST);
107                         pnlWest.setPreferredSize(new Dimension(170,700));
108                         pnlWest.setLayout(new FlowLayout());
109                         {
110                                 {
111                                         lblIntermediateProduct = new JLabel("Mellemvarer:");
112                                         lblIntermediateProduct.setFont(lblIntermediateProduct.getFont()
                                                .deriveFont(lblIntermediateProduct.getFont().getStyle() ^
                                                Font.BOLD));
113                                         lblIntermediateProduct.setPreferredSize(new Dimension(160,25));
114                                         pnlWest.add(lblIntermediateProduct);
115                                 }
116                                 {
117                                         txfSearch = new JTextField();
118                                         txfSearch.setPreferredSize(new Dimension(160,25));
119                                         txfSearch.getDocument().addDocumentListener(controller);
120                                         pnlWest.add(txfSearch);
121                                 }
122                                 {
123                                         scpIntermediateProducts = new JScrollPane();
124                                         pnlWest.add(scpIntermediateProducts);
125                                         scpIntermediateProducts.setPreferredSize(new Dimension(160,200)
                                                );
126
127                                         {
128                                                 ListModel lstIntermediateProductsModel = new
                                                        DefaultListModel();
129                                                 lstIntermediateProducts = new JList();
130                                                 lstIntermediateProducts.setSelectionMode(
                                                        ListSelectionModel.SINGLE_SELECTION);
131                                                 scpIntermediateProducts.setViewportView(
                                                        lstIntermediateProducts);
132                                                 lstIntermediateProducts.setModel(
                                                        lstIntermediateProductsModel);
133                                                 lstIntermediateProducts.addListSelectionListener(
                                                        controller);
134                                         }
135                                 }
136                                 {
137                                         btnCreateIntermediateProduct = new JButton();
138                                         btnCreateIntermediateProduct.setText("Opret Mellemvare");
139                                         btnCreateIntermediateProduct.setPreferredSize(new Dimension
                                                (160,25));
140                                         btnCreateIntermediateProduct.addActionListener(controller);
141                                         pnlWest.add(btnCreateIntermediateProduct);
142                                         btnCreateIntermediateProduct.setMnemonic(KeyEvent.VK_M);
```

```
143                             }
144
145                     }
146             }
147             {
148                     pnlEast = new JPanel();
149                     getContentPane().add(pnlEast, BorderLayout.EAST);
150                     pnlEast.setPreferredSize(new Dimension(170, 700));
151                     pnlEast.setLayout(new FlowLayout());
152                     {
153                             {
154                                     lblInformation = new JLabel();
155                                     pnlEast.add(lblInformation);
156                                     lblInformation.setFont(lblInformation.getFont().deriveFont(
                                            lblInformation.getFont().getStyle() ^ Font.BOLD));
157                                     lblInformation.setText("Information:");
158                                     lblInformation.setPreferredSize(new Dimension(160,25));
159                             }
160                             {
161                                     lblID = new JLabel();
162                                     pnlEast.add(lblID);
163                                     lblID.setText("ID:");
164                                     lblID.setPreferredSize(new Dimension(80,25));
165                             }
166                             {
167                                     txfID = new JTextField();
168                                     pnlEast.add(txfID);
169                                     txfID.setPreferredSize(new Dimension(80,25));
170                                     txfID.setEditable(false);
171                             }
172                             {
173                                     lblQuantity = new JLabel();
174                                     pnlEast.add(lblQuantity);
175                                     lblQuantity.setText("Antal:");
176                                     lblQuantity.setPreferredSize(new Dimension(80,25));
177                             }
178                             {
179                                     txfQuantity = new JTextField();
180                                     pnlEast.add(txfQuantity);
181                                     txfQuantity.setPreferredSize(new Dimension(80,25));
182                                     txfQuantity.setEditable(false);
183                             }
184                             {
185                                     lblProductType = new JLabel();
186                                     pnlEast.add(lblProductType);
187                                     lblProductType.setText("Produkt type:");
188                                     lblProductType.setPreferredSize(new Dimension(160,25));
189                             }
190                             {
191                                     txfProductType = new JTextField();
192                                     pnlEast.add(txfProductType);
193                                     txfProductType.setPreferredSize(new Dimension(160,25));
194                                     txfProductType.setEditable(false);
195                             }
196                             {
197                                     lblDepot = new JLabel();
198                                     pnlEast.add(lblDepot);
199                                     lblDepot.setText("Lager:");
200                                     lblDepot.setPreferredSize(new Dimension(80,25));
201                             }
202                             {
203                                     txfDepot = new JTextField();
204                                     pnlEast.add(txfDepot);
205                                     txfDepot.setPreferredSize(new Dimension(80,25));
206                                     txfDepot.setEditable(false);
207                             }
208                             {
209                                     lblCoordinates = new JLabel();
210                                     pnlEast.add(lblCoordinates);
211                                     lblCoordinates.setText("Position:");
212                                     lblCoordinates.setPreferredSize(new Dimension (80,25));
213                             }
214                             {
215                                     txfCoordinates = new JTextField();
216                                     pnlEast.add(txfCoordinates);
217                                     txfCoordinates.setPreferredSize(new Dimension(80,25));
218                                     txfCoordinates.setEditable(false);
219                             }
```

```
220                             {
221                                     pnlProcessOverView = new JPanel();
222                                     BoxLayout pnlProcessOverViewLayout = new BoxLayout(
                                            pnlProcessOverView, BoxLayout.Y_AXIS);
223                                     pnlProcessOverView.setLayout(pnlProcessOverViewLayout);
224                                     pnlEast.add(pnlProcessOverView);
225                             }
226                             {
227                                     btnSendToNextProcess = new JButton();
228                                     pnlEast.add(btnSendToNextProcess);
229                                     btnSendToNextProcess.setText("Viderebehandle");
230                                     btnSendToNextProcess.setPreferredSize(new Dimension(160,25));
231                                     btnSendToNextProcess.addActionListener(controller);
232                                     btnSendToNextProcess.setMnemonic(KeyEvent.VK_I);
233                             }
234                             {
235                                     btnDeleteIntermediateProduct = new JButton();
236                                     pnlEast.add(btnDeleteIntermediateProduct);
237                                     btnDeleteIntermediateProduct.setText("Kassere mellemvare");
238                                     btnDeleteIntermediateProduct.setPreferredSize(new Dimension
                                            (160,25));
239                                     btnDeleteIntermediateProduct.addActionListener(controller);
240                                     btnDeleteIntermediateProduct.setMnemonic(KeyEvent.VK_K);
241                             }
242                     }
243             }
244             {
245                     pnlCenter = new JPanel();
246                     getContentPane().add(pnlCenter, BorderLayout.CENTER);
247                     pnlCenter.setLayout(new BorderLayout());
248                     {
249                             pnlSelectedDepot = new JPanel();
250                             pnlCenter.add(pnlSelectedDepot, BorderLayout.NORTH);
251                             pnlSelectedDepot.setLayout(new FlowLayout());
252                             {
253                                     lblSelectedDepot = new JLabel();
254                                     pnlSelectedDepot.add(lblSelectedDepot, BorderLayout.NORTH);
255                                     lblSelectedDepot.setPreferredSize(new Dimension(100,25));
256                                     lblSelectedDepot.setFont(lblSelectedDepot.getFont().deriveFont(
                                            lblSelectedDepot.getFont().getStyle() ^ Font.BOLD));
257
258                             }
259                     }
260                     {
261                             pnlIntermediateProductMap = new JPanel();
262                             pnlCenter.add(pnlIntermediateProductMap, BorderLayout.CENTER);
263                             lytIntermediateProductMap.setHgap(4);
264                             lytIntermediateProductMap.setVgap(4);
265                             pnlIntermediateProductMap.setLayout(lytIntermediateProductMap);
266                             pnlIntermediateProductMap.setBorder(BorderFactory.createEtchedBorder())
                                    ;
267                     }
268             }
269             {
270                     mnbBar = new JMenuBar();
271                     setJMenuBar(mnbBar);
272                     {
273                             mnuCreate = new JMenu();
274                             mnbBar.add(mnuCreate);
275                             mnuCreate.setText("Opret");
276                             mnuCreate.setMnemonic(KeyEvent.VK_O);
277                             {
278                                     mitCreateProductType = new JMenuItem();
279                                     mnuCreate.add(mitCreateProductType);
280                                     mitCreateProductType.setText("Opret Produkttype");
281                                     mitCreateProductType.addActionListener(controller);
282                             }
283                             {
284                                     mitCreateIntermediateProduct = new JMenuItem();
285                                     mnuCreate.add(mitCreateIntermediateProduct);
286                                     mitCreateIntermediateProduct.setText("Opret Mellemvare");
287                                     mitCreateIntermediateProduct.addActionListener(controller);
288                             }
289                     }
290                     {
291                             mnuView = new JMenu();
292                             mnbBar.add(mnuView);
293                             mnuView.setText("Vis");
```

```
294                                 mnuView.setMnemonic(KeyEvent.VK_V);
295                                 {
296                                         mitViewDepot = new JMenu();
297                                         mnuView.add(mitViewDepot);
298                                         mitViewDepot.setText("Vis lager");
299                                         {
300                                                 fillChooseDepotMenu();
301
302                                         }
303                                 }
304                         }
305                 }
306                 pack();
307
308                 controller.fillLstIntermediateProducts();
309                 updateTimer = new UpdateTimer(2, intermediateProductPanels);
310
311         }
312         public void fillChooseDepotMenu() {
313                 mitViewDepot.removeAll();
314                 mitDepots.clear();
315                 mitViewDepot.updateUI();
316                 for (Depot depot : Service.getService().getAllDepots()) {
317                         JMenuItem mitDepot = new JMenuItem();
318                         mitDepot.setText(depot.getName());
319                         mitDepot.addActionListener(controller);
320                         mitDepots.add(mitDepot);
321                         mitViewDepot.add(mitDepot);
322                 }
323                 if(mitDepots.size()>0) {
324                         updateDepotMap(Service.getService().getAllDepots().get(0));
325                 }
326
327         }
328         // Denne metode kr ver at arraylisten med StoringSpaces i depot er efter I se systemet
329         public void updateDepotMap(Depot depot) {
330                 lblSelectedDepot.setText(depot.getName());
331                 pnlIntermediateProductMap.removeAll();
332                 intermediateProductPanels.clear();
333                 pnlIntermediateProductMap.updateUI();
334                 lytIntermediateProductMap.setColumns(depot.getMaxX());
335                 lytIntermediateProductMap.setRows(depot.getMaxY());
336
337                 for (StoringSpace storingSpace : depot.getStoringSpaces()) {
338                         IntermediateProductPanel intermediateProductPanel = new
                                IntermediateProductPanel(storingSpace);
339                         intermediateProductPanel.addMouseListener(controller);
340                         intermediateProductPanels.add(intermediateProductPanel);
341                         pnlIntermediateProductMap.add(intermediateProductPanel);
342                 }
343         }
344
345         /**
346          * Denne methode bliver udfoert ver gang man klicker paa en storingspace med musen i guien
347          * @param intermediateProductPanel
348          */
349         public void updateInfoFromPanel(IntermediateProductPanel intermediateProductPanel) {
350                 if (selectedIntermediateProductPanel != intermediateProductPanel) {
351                         if (selectedIntermediateProductPanel != null) { //unselecter den gamle
                                storingspace
352                                 selectedIntermediateProductPanel.setSelected(false);
353                         }
354                         intermediateProductPanel.setSelected(true);
355                         selectedIntermediateProductPanel = intermediateProductPanel;
356                         lstIntermediateProducts.setSelectedValue(intermediateProductPanel.
                                getStoringSpace().getIntermediateProduct(), true);
357                         updateInfo();
358                 }
359         }
360
361         public void updateInfoFromList(IntermediateProduct intermediateProduct) {
362                 if(intermediateProduct != null) {
363                         if(intermediateProduct.getActivProcessLog() != null) {
364                                 if(intermediateProduct.getActivProcessLog().getProcess().getClass().
                                        getName().equals("model.Drying")) {
365                                         updateDepotMap(intermediateProduct.getStoringSpace().getDepot()
                                                );
```

```
366                                     for (IntermediateProductPanel intermediateProductPanel :
                                            intermediateProductPanels) {
367                                         if (intermediateProductPanel.getStoringSpace().
                                                getIntermediateProduct() == intermediateProduct) {
368                                             updateInfoFromPanel(intermediateProductPanel);
369                                         }
370                                     }
371                             }
372                             else {
373                                     updateInfo();
374                             }
375                     }
376                     else {
377                             updateInfo();
378                     }
379             }
380     }
381
382     private void updateProcessOverView(IntermediateProduct intermediateProduct) {
383             if(intermediateProduct != null) {
384                     pnlProcessOverView.removeAll();
385                     processPanels.clear();
386                     pnlProcessOverView.updateUI();
387
388                     for (Process process : intermediateProduct.getProductType().getProcessLine().
                            getProcesses()) {
389                             ProcessPanel processPanel = new ProcessPanel(intermediateProduct,
                                    process);
390                             processPanel.addMouseListener(controller);
391                             processPanels.add(processPanel);
392                             pnlProcessOverView.add(processPanel);
393                     }
394             }
395     }
396     private void updateInfo() {
397             if((IntermediateProduct)lstIntermediateProducts.getSelectedValue() != null) {
398                     IntermediateProduct intermediateProduct = (IntermediateProduct)
                            lstIntermediateProducts.getSelectedValue();
399                     updateProcessOverView(intermediateProduct);
400
401                     btnDeleteIntermediateProduct.setVisible(true);
402                     btnSendToNextProcess.setVisible(true);
403                     txfID.setText(intermediateProduct.getId());
404                     txfProductType.setText(intermediateProduct.getProductType().getName());
405                     txfQuantity.setText(intermediateProduct.getQuantity()+ "");
406                     if(intermediateProduct.getStoringSpace() != null) {
407                             txfCoordinates.setText("( " + intermediateProduct.getStoringSpace().
                                    getPositionX()+ ":" + intermediateProduct.getStoringSpace().
                                    getPositionY() + " )");
408                             txfDepot.setText(intermediateProduct.getStoringSpace().getDepot().
                                    getName());
409                     }
410                     else {
411                             txfDepot.setText("N/A");
412                             txfCoordinates.setText("N/A");
413                     }
414             }
415             else {
416                     btnDeleteIntermediateProduct.setVisible(false);//delete btn skal ikke vises
                            hvis man trykker paa et tomt feld
417                     btnSendToNextProcess.setVisible(false);
418                     txfID.setText("N/A");
419                     txfQuantity.setText("N/A");
420                     txfProductType.setText("N/A");
421                     txfQuantity.setText("N/A");
422             }
423     }
424     private class Controller implements ActionListener, ListSelectionListener, MouseListener,
            DocumentListener, WindowListener {
425
426             public void actionPerformed(ActionEvent e) {
427                     if (e.getSource() == mitCreateProductType) {
428                             createProductTypeFrame = new CreateProductTypeFrame();
429                             if (createProductTypeFrame.getProductType()!=null){
430                                     Service.getService().storeProductType(createProductTypeFrame.
                                            getProductType());
431                             }
432                     }
```

```
433
434                     else if (e.getSource() == mitCreateIntermediateProduct || e.getSource() ==
                            btnCreateIntermediateProduct) {
435                         createIntermediateProduct = new CreateIntermediateProductFrame();
436                         if (createIntermediateProduct.getIntermediateProduct()!=null){
437                                 Service.getService().StoreIntermediateProduct(
                                        createIntermediateProduct.getIntermediateProduct());
438                         }
439                         fillLstIntermediateProducts();
440                     } else if (e.getSource().equals(btnDeleteIntermediateProduct)){
441                         IntermediateProduct selectedIntermediateProduct=(IntermediateProduct)
                                lstIntermediateProducts.getSelectedValue();
442                         if (selectedIntermediateProduct!=null){
443                                 Depot selectedIntermediateProductDepot =null;
444                                 if (selectedIntermediateProduct.getStoringSpace()!=null){
445                                         selectedIntermediateProductDepot =
                                                selectedIntermediateProduct.getStoringSpace().
                                                getDepot();
446                                 }
447                                 selectedIntermediateProduct.discardThisIntermediateProduct();
448                                 Service.getService().StoreIntermediateProduct(
                                        selectedIntermediateProduct);
449                                 fillLstIntermediateProducts();
450                                 if (selectedIntermediateProductDepot!=null){
451                                         updateDepotMap(selectedIntermediateProductDepot);
452                                 }
453                                 updateInfo();
454                         } else {
455                                 JOptionPane.showMessageDialog(null, "Ingen mellemvare valgt", "
                                        Fejl", JOptionPane.ERROR_MESSAGE);
456                         }
457                     } else if (e.getSource().equals(btnSendToNextProcess)){
458                         IntermediateProduct selectedIntermediateProduct=(IntermediateProduct)
                                lstIntermediateProducts.getSelectedValue();
459
460                         if (selectedIntermediateProduct!=null){
461                                 Depot currentDepot =null;
462                                 if (selectedIntermediateProductPanel!=null){
463                                         currentDepot = selectedIntermediateProductPanel.
                                                getStoringSpace().getDepot();
464                                 }
465
466                                 if (selectedIntermediateProduct.getNextProcess()==null ||
                                        selectedIntermediateProduct.getNextProcess().getClass().
                                        equals(model.SubProcess.class)){
467                                     if (selectedIntermediateProduct.getActivProcessLog()!=
                                            null && selectedIntermediateProduct.
                                            getActivProcessLog().getProcess().getClass().equals
                                            (model.Drying.class)){
468                                         selectedIntermediateProductPanel=null;
469                                     }
470                                     selectedIntermediateProduct.sendToNextProcess(null);
471                                     Service.getService().StoreIntermediateProduct(
                                            selectedIntermediateProduct);
472                                 } else if (selectedIntermediateProduct.getNextProcess().
                                        getClass().equals(model.Drying.class)){
473                                     if (selectedIntermediateProductPanel==null){
474                                             JOptionPane.showMessageDialog(null, "V lg et
                                                    lagerplads hvor mellemvaren skal l gges",
                                                    "Fejl", JOptionPane.ERROR_MESSAGE);
475                                     } else if (selectedIntermediateProductPanel.
                                            getStoringSpace().getIntermediateProduct()!=null){
476                                         JOptionPane.showMessageDialog(null, "Der ligger
                                                allerede en mellemvare paa den valgte
                                                placering", "Fejl", JOptionPane.
                                                ERROR_MESSAGE);
477                                     } else if (!((Drying)selectedIntermediateProduct.
                                            getNextProcess()).getDepots().contains(
                                            selectedIntermediateProductPanel.getStoringSpace().
                                            getDepot())){
478                                         JOptionPane.showMessageDialog(null, "
                                                Mellemvaren kan ikke ligge p  det valgte
                                                lager,f lgende lagre er gyldige "+((Drying
                                                )selectedIntermediateProduct.getNextProcess
                                                ()).getDepots(), "Fejl", JOptionPane.
                                                ERROR_MESSAGE);
479                                     } else {
```

```
480                                                     selectedIntermediateProduct.sendToNextProcess(
                                                            selectedIntermediateProductPanel.
                                                            getStoringSpace());
481                                                     Service.getService().StoreIntermediateProduct(
                                                            selectedIntermediateProduct);
482                                             }
483
484                                     }
485                                     fillLstIntermediateProducts();
486                                     if (currentDepot!=null){
487                                             updateDepotMap(currentDepot);
488                                     }
489                                     lstIntermediateProducts.setSelectedValue(
                                            selectedIntermediateProduct, true);
490                                     updateInfo();
491
492                             } else {
493                                     JOptionPane.showMessageDialog(null, "Ingen mellemvare valgt", "
                                            Fejl", JOptionPane.ERROR_MESSAGE);
494                             }
495                     }
496
497                     for (int i = 0; i < mitDepots.size(); i++) {
498                             if (e.getSource() == mitDepots.get(i)) {
499                                     updateDepotMap(Service.getService().getAllDepots().get(i));
500                             }
501                     }
502             }
503             @Override
504             public void valueChanged(ListSelectionEvent e) {
505                     if (e.getSource() == lstIntermediateProducts) {
506                             if (!e.getValueIsAdjusting()) {
507                                     updateInfoFromList((IntermediateProduct)lstIntermediateProducts
                                            .getSelectedValue());
508                             }
509                     }
510
511             }
512
513             @Override
514             public void mouseClicked(MouseEvent e) {
515                     if(e.getSource().getClass().equals(IntermediateProductPanel.class)) {
516                             updateInfoFromPanel((IntermediateProductPanel)e.getSource());
517                     }
518             }
519
520             @Override
521             public void mouseEntered(MouseEvent e) {
522                     // TODO Auto-generated method stub
523
524             }
525
526             @Override
527             public void mouseExited(MouseEvent e) {
528                     // TODO Auto-generated method stub
529
530             }
531
532             @Override
533             public void mousePressed(MouseEvent e) {
534                     // TODO Auto-generated method stub
535
536             }
537
538             @Override
539             public void mouseReleased(MouseEvent e) {
540                     // TODO Auto-generated method stub
541
542             }
543             @Override
544             public void changedUpdate(DocumentEvent e) {
545                     fillLstIntermediateProducts();
546
547             }
548             @Override
549             public void insertUpdate(DocumentEvent e) {
550                     fillLstIntermediateProducts();
551
```

```
552                 }
553                 @Override
554                 public void removeUpdate(DocumentEvent e) {
555                         fillLstIntermediateProducts();
556
557                 }
558
559                 public void fillLstIntermediateProducts() {
560
561                         List<IntermediateProduct> allIntermediateProducts = Service.getService().
                                getActiveIntermediateProducts();
562                         List<IntermediateProduct> searchedIntermediateProducts = new ArrayList<
                                IntermediateProduct>();
563
564                         for (int i = 0; i < allIntermediateProducts.size(); i++) {
565                                 String idLower = allIntermediateProducts.get(i).getId().toLowerCase();
566                                 String productTypeLower = allIntermediateProducts.get(i).getProductType
                                        ().getName().toLowerCase();
567                                 if (idLower.indexOf(txfSearch.getText().toLowerCase())!=-1 ||
                                        productTypeLower.indexOf(txfSearch.getText().toLowerCase())!=-1){
568                                         searchedIntermediateProducts.add(allIntermediateProducts.get(i)
                                                );
569                                 }
570                         }
571                         lstIntermediateProducts.setListData(searchedIntermediateProducts.toArray());
572                 }
573                 @Override
574                 public void windowActivated(WindowEvent e) {
575                         // TODO Auto-generated method stub
576
577                 }
578                 @Override
579                 public void windowClosed(WindowEvent e) {
580                         // TODO Auto-generated method stub
581
582                 }
583                 @Override
584                 public void windowClosing(WindowEvent e) {
585                         updateTimer.cancel();
586                         service.Service.getService().closeDao();
587                         System.exit(0);
588                 }
589                 @Override
590                 public void windowDeactivated(WindowEvent e) {
591                         // TODO Auto-generated method stub
592
593                 }
594                 @Override
595                 public void windowDeiconified(WindowEvent e) {
596                         // TODO Auto-generated method stub
597
598                 }
599                 @Override
600                 public void windowIconified(WindowEvent e) {
601                         // TODO Auto-generated method stub
602
603                 }
604                 @Override
605                 public void windowOpened(WindowEvent e) {
606                         // TODO Auto-generated method stub
607
608                 }
609         }
610 }
```

Mellemvarelabel.java

```
1  package gui;
2
3  import javax.swing.JLabel;
4
5  public class Mellemvarelabel extends JLabel {
6
7          public Mellemvarelabel() {
8                  // TODO Auto-generated constructor stub
9          }
10
11
12
13 }
```

UpdateTimer.java

```
1  package gui;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import java.util.TimerTask;
6
7  import javax.swing.JDialog;
8  import javax.swing.JOptionPane;
9
10 import model.Drying;
11 import model.IntermediateProduct;
12 import model.ProcessLog;
13
14 /**
15  * @author M. C. H j
16  */
17
18 public class UpdateTimer extends java.util.Timer {
19
20         private TimerAction timerAction = new TimerAction();
21         private ArrayList<IntermediateProductPanel> intermediateProductPanels;
22         private ArrayList<ProcessLog> logsRecievedAnWarning = new ArrayList<ProcessLog>();
23         private ArrayList<ProcessLog> logsToOld = new ArrayList<ProcessLog>();
24
25         public UpdateTimer(int updateIntervalInSeconds, ArrayList<IntermediateProductPanel>
               intermediateProductPanels){
26             this.schedule(timerAction, 100l, updateIntervalInSeconds*1000l);
27             this.intermediateProductPanels = intermediateProductPanels;
28         }
29
30         class TimerAction extends TimerTask{
31             @Override
32             public void run() {
33
34                     // opdatere prograssbars
35                     for (int i = 0; i < intermediateProductPanels.size(); i++) {
36                             intermediateProductPanels.get(i).updateTime();
37                     }
38
39                     List<IntermediateProduct> intermediateProducts = service.Service.getService().
                           getActiveIntermediateProducts();
40
41                     //leder efter mellemvare hvor der skal gives en advarsel
42                     for (int i = 0; i < intermediateProducts.size(); i++) {
43
44                             if (intermediateProducts.get(i).getActivProcessLog()!=null){
45                                     if (intermediateProducts.get(i).getActivProcessLog().getProcess
                                           ().getClass().equals(Drying.class)){
46
47                                             Drying drying = (Drying) intermediateProducts.get(i).
                                                   getActivProcessLog().getProcess();
48                                             ProcessLog log = intermediateProducts.get(i).
                                                   getActivProcessLog();
49                                             long currentTime = System.currentTimeMillis()-log.
                                                   getStartTime().getTime();
50                                             if ((drying.getIdealTime()+(drying.getMaxTime()-drying.
                                                   getIdealTime()))/2)<currentTime && drying.getMaxTime
                                                   ()>currentTime && !logsRecievedAnWarning.contains(
                                                   log)){
51                                                     logsRecievedAnWarning.add(log);
52                                                     JDialog dialogWarning = new JOptionPane("
                                                           Mellemvaren "+intermediateProducts.get(i)+"
                                                            p   placeringengen "+log.getStoringSpace()
                                                           .getDepot()+" "+log.getStoringSpace()+" har
                                                            snat overskredet sin t rretid!",
                                                           JOptionPane.WARNING_MESSAGE,JOptionPane.
                                                           CLOSED_OPTION).createDialog("Advarsel");
53                                                     dialogWarning.setModal(false);
54                                                     dialogWarning.setAlwaysOnTop(true);
55                                                     dialogWarning.setVisible(true);
56                                             } else if (drying.getMaxTime()<currentTime && !
                                                   logsToOld.contains(log)) {
57                                                     logsToOld.add(log);
58                                                     JDialog dialogToOld = new JOptionPane("
                                                           Mellemvaren "+intermediateProducts.get(i)+"
```

```
                                                                    p   placeringengen "+log.getStoringSpace()
                                                                .getDepot()+" "+log.getStoringSpace()+" har
                                                                 overskredet sin t rretid!",JOptionPane.
                                                                ERROR_MESSAGE,JOptionPane.CLOSED_OPTION).
                                                                createDialog("Advarsel");
59                                                          dialogToOld.setModal(false);
60                                                          dialogToOld.setAlwaysOnTop(true);
61                                                          dialogToOld.setVisible(true);
62                                                  }
63
64                                          }
65                                  }
66
67                          }
68
69                  }
70          }
71  }
```

3 model.*

Depot.java

```
1  package model;
2
3  import java.util.ArrayList;
4
5  /**
6   * @author Carl
7   */
8
9  public class Depot {
10         private String name;
11         private String description;
12         private int maxX;
13         private int maxY;
14         private ArrayList<StoringSpace> storingspaces = new ArrayList<StoringSpace>();
15         private ArrayList<Drying> dryings = new ArrayList<Drying>();
16
17         public Depot(String name, String desciption, int maxX, int maxY) throws RuntimeException{
18
19                 if (maxY<=0 || maxX<=0){
20                         throw new RuntimeException("maxY and maxX can't be a negatic number");
21                 } else {
22                 this.maxX=maxX;
23                 this.maxY=maxY;
24
25                 for (int y = 1; y <=maxY; y++) {
26                         for (int x = 1; x <=maxX; x++) {
27                                 StoringSpace ss =new StoringSpace(x,y,this);
28                                 this.storingspaces.add(ss);
29                         }
30                 }
31
32                 this.setName(name);
33                 this.setDescription(desciption);
34                 }
35         }
36
37         public String getName(){
38                 return this.name;
39         }
40
41         public void setName(String name){
42                 this.name=name;
43         }
44
45         public String getDescription(){
46                 return this.description;
47         }
48
49         public void setDescription(String description){
50                 this.description=description;
51         }
52
53         public ArrayList<StoringSpace> getStoringSpaces(){
54                 return this.storingspaces;
55         }
56
57         public ArrayList<Drying> getDryings(){
58                 return this.dryings;
59         }
60
61         public void addDrying(Drying drying){
62                 this.dryings.add(drying);
63                 if(!drying.getDepots().contains(this)){
64                         drying.addDepot(this);
65                 }
66         }
67
68         public void removeDrying(Drying drying){
69                 this.dryings.remove(drying);
70                 if(drying.getDepots().contains(this)){
71                         drying.removeDepot(this);
72                 }
73         }
74
75         public int getMaxX(){
```

```
76                      return this.maxX;
77              }
78
79              public int getMaxY(){
80                      return this.maxY;
81              }
82
83              public String toString(){
84                      return name;
85              }
86
87      }
```

Drying.java

```
1  package model;
2
3  import java.util.ArrayList;
4
5  /**
6   * @author Carl
7   */
8
9  public class Drying extends model.Process {
10         private long minTime;
11         private long idealTime;
12         private long maxTime;
13         private ArrayList<Depot> depots  = new ArrayList<Depot>();
14
15         public Drying(long minTime, long idealTime, long maxTime, int processStep, ProcessLine
               processLine) throws RuntimeException{
16                 super(processStep, processLine);
17                 this.setMinTime(minTime);
18                 this.setIdealTime(idealTime);
19                 this.setMaxTime(maxTime);
20         }
21
22         public long getMinTime(){
23                 return this.minTime;
24         }
25
26         public void setMinTime(long minTime) throws RuntimeException{
27                 if (minTime<=0){
28                         throw new RuntimeException("minTime can't be a negative number");
29                 } else {
30                         this.minTime=minTime;
31                 }
32         }
33
34         public long getIdealTime(){
35                 return this.idealTime;
36         }
37
38         public void setIdealTime(long idealTime) throws RuntimeException{
39                 if (idealTime<=0){
40                         throw new RuntimeException("idealTime can't be a negative number");
41                 } else if(idealTime <= minTime){
42                         throw new RuntimeException("idealTime can't be less then minTime");
43                 } else {
44                         this.idealTime=idealTime;
45                 }
46         }
47
48         public long getMaxTime(){
49                 return this.maxTime;
50         }
51
52         public void setMaxTime(long maxTime) throws RuntimeException{
53                 if (maxTime<=0){
54                         throw new RuntimeException("maxTime can't be a negative number");
55                 } else if(maxTime <= idealTime){
56                         throw new RuntimeException("maxTime can't be less then idealTime");
57                 } else {
58                         this.maxTime=maxTime;
59                 }
60         }
61
62         public ArrayList<Depot> getDepots(){
63                 return this.depots;
64         }
65
66         public void addDepot(Depot depot){
67                 this.depots.add(depot);
68                 if(!depot.getDryings().contains(this)){
69                         depot.addDrying(this);
70                 }
71         }
72
73         public void removeDepot(Depot depot){
74                 this.depots.remove(depot);
```

```
75                  if(depot.getDryings().contains(this)){
76                          depot.removeDrying(this);
77                  }
78          }
79
80          public String toString(){
81                  return "Toerring";
82          }
83
84  }
```

IntermediateProduct.java

```
1  package model;
2
3  import java.util.ArrayList;
4
5  /**
6   * @author M. C. H j
7   */
8
9  public class IntermediateProduct {
10         private boolean finished = false;
11         private boolean discarded = false;
12         private String id;
13         private double quantity;
14         private ProductType productType;
15         private ArrayList<ProcessLog> processLogs  = new ArrayList<ProcessLog>();
16         private StoringSpace storingSpace = null;
17
18         /**
19          * Creates an Intermediate product
20          * @param id - the id of the product
21          * @param productType - type of product
22          * @param quantity - how much of the product there is
23          * @throws RuntimeException
24          */
25         public IntermediateProduct(String id, ProductType productType, double quantity) throws
               RuntimeException{
26                 this.setId(id);
27                 this.setProductType(productType);
28                 this.setQuantity(quantity);
29         }
30
31         /**
32          * Tels if the product hav been throught al its processes
33          * @return Boolean
34          */
35         public boolean isFinished(){
36                 return this.finished;
37         }
38
39         /**
40          * Tels if the product af been discarded
41          * @return
42          */
43         public boolean isDiscarded(){
44                 return this.discarded;
45         }
46
47         /**
48          * Product id
49          * @return String
50          */
51         public String getId(){
52                 return this.id;
53         }
54
55         /**
56          * Sets an nes id for the product
57          * @param id
58          */
59         public void setId(String id){
60                 this.id=id;
61         }
62
63         /**
64          * Gets how mutch of the product there is
65          * @return double
66          */
67         public double getQuantity(){
68                 return this.quantity;
69         }
70
71         /**
72          * Tells the quantity of the product
73          * @param quantity
74          * @throws RuntimeException - throw an exeption if quantity is smaller then 0
```

```
75          */
76          public void setQuantity(double quantity) throws RuntimeException{
77                  if (quantity<0){
78                          throw new RuntimeException("quantity can't be a negative number");
79                  } else {
80                          this.quantity=quantity;
81                  }
82          }
83
84          /**
85           * Information about the type of product
86           * @return ProductType
87           */
88          public ProductType getProductType(){
89                  return this.productType;
90          }
91
92          /**
93           * sets the type of product
94           * @param productType
95           * @throws RuntimeException throws an exeption if productType is null
96           */
97          public void setProductType(ProductType productType) throws RuntimeException{
98                  if (productType==null){
99                          throw new RuntimeException("productType can't be null");
100                 } else {
101                         if (this.productType != null){
102                                 this.productType.removeIntermediateProduct(this);
103                         }
104                         this.productType=productType;
105                         if (!productType.getIntermediateProducts().contains(this)){
106                                 productType.addIntermediateProduct(this);
107                         }
108                 }
109         }
110
111         /**
112          * Gets the processes that the intermediate product have been throught
113          * @return ArrayList<ProcessLog>
114          */
115         public ArrayList<ProcessLog> getProcessLogs(){
116                 return this.processLogs;
117         }
118
119         /**
120          * Creates an ProcessLog.
121          * Warning dont call this method directly, use {@link #sendToNextProcess(StoringSpace)} or {
                @link #discardThisIntermediateProduct()}
122          * @param process - the process that is started
123          * @param storingSpace - the Storing space where the process is executed
124          * @return ProcessLog
125          * @throws RuntimeException throughts an exeption if process is null
126          */
127         public ProcessLog createProcessLog(Process process, StoringSpace storingSpace) throws
              RuntimeException {
128                 ProcessLog p =new ProcessLog(process,storingSpace,this);
129                 this.processLogs.add(p);
130                 return p;
131         }
132
133         /**
134          * Deletes an ProcessLog
135          * Warning dont call this method, it will mess up the data that this class contains
136          * @param processLog
137          */
138         public void deleteProcessLog(ProcessLog processLog){
139                 this.processLogs.remove(processLog);
140                 if (processLog.getStoringSpace()!=null){
141                         processLog.unsetStoringSpace();
142                 }
143                 processLog.getProcess().removeProcessLog(processLog);
144         }
145
146         /**
147          * returns the storingspace where this IntermediateProduct is
148          * @return
149          */
150         public StoringSpace getStoringSpace(){
```

```
151                 return this.storingSpace;
152         }
153 
154         /**
155          * Sets the storingSpace where this product is.
156          * Warning dont call this method directly, use {@link #sendToNextProcess(StoringSpace)} or {
                 @link #discardThisIntermediateProduct()}
157          * @param storingSpace
158          */
159         public void setStoringSpace(StoringSpace storingSpace){
160                 if (this.storingSpace != null){
161                         this.storingSpace.unsetIntermediateProduct();
162                 }
163                 this.storingSpace=storingSpace;
164                 if (storingSpace.getIntermediateProduct()!=this){
165                         storingSpace.setIntermediateProduct(this);
166                 }
167         }
168 
169         /**
170          * remove this IntermediateProduct from its StoringSpace.
171          * Warning dont call this method directly, use {@link #sendToNextProcess(StoringSpace)} or {
                 @link #discardThisIntermediateProduct()}
172          */
173         public void unsetStoringSpace(){
174                 StoringSpace oldStoringSpace = this.storingSpace;
175                 this.storingSpace = null;
176                 if (oldStoringSpace.getIntermediateProduct()!=null){
177                         oldStoringSpace.unsetIntermediateProduct();
178                 }
179         }
180 
181         /**
182          * IntermediateProduct description
183          */
184         public String toString() {
185                 return id+" "+productType.getName();
186         }
187 
188         /**
189          * Method used to start, send to, and finish the processes the IntermediateProduct have to go
                 throught
190          * This method automaticly sts and unsets the storingspace og this product
191          * @param StoringSpace if the process needs an storingspace set, his atribute, otherwise set it
                 to null
192          */
193         public void sendToNextProcess(StoringSpace storingSpace){
194                 //tests if the IntermediateProduct is finished or discarded
195                 if (!isDiscarded() && !isFinished()){
196                         //tests if we are going to start the first process
197                         if (processLogs.size()==0){
198 
199                                 createProcessLog(this.productType.getProcessLine().getProcesses().get
                                         (0), storingSpace);
200                                 if (storingSpace==null){
201                                         if (this.storingSpace!=null){
202                                                 unsetStoringSpace();
203                                         }
204                                 } else {
205                                         this.setStoringSpace(storingSpace);
206                                 }
207 
208                                 //test if we are at the last process
209                         } else if (processLogs.size()>=this.productType.getProcessLine().getProcesses()
                                 .size()) {
210                                 if (this.storingSpace!=null){
211                                         unsetStoringSpace();
212                                 }
213                                 processLogs.get(processLogs.size()-1).endProcess();
214                                 finished=true;
215 
216                                 //neither first or last process
217                         } else {
218                                 int i = processLogs.size()-1;
219                                 processLogs.get(i).endProcess();
220                                 createProcessLog(this.productType.getProcessLine().getProcesses().get(i
                                         +1), storingSpace);
221 
```

```
222                                 if (storingSpace==null){
223                                         if (this.storingSpace!=null){
224                                                 unsetStoringSpace();
225                                         }
226                                 } else {
227                                         this.setStoringSpace(storingSpace);
228                                 }
229
230                         }
231                 }
232         }
233
234         /**
235          * returns the active ProcessLog
236          * will be null if the IntermediateProduct is finished, discared or that we havent started any
                processes yet
237          * @return
238          */
239         public ProcessLog getActivProcessLog(){
240
241                 if (processLogs.size()==0 || isFinished() || isDiscarded()) {
242                         return null;
243                 } else {
244                         return processLogs.get(processLogs.size()-1);
245                 }
246         }
247
248         /**
249          * returns the next process
250          *  will be null if the IntermediateProduct is finished, discared or if the active process is
                the last
251          * @return
252          */
253         public Process getNextProcess(){
254                 if (processLogs.size()>=this.productType.getProcessLine().getProcesses().size() ||
                    isFinished() || isDiscarded()) {
255                         return null;
256                 } else {
257                         return productType.getProcessLine().getProcesses().get(processLogs.size());
258                 }
259
260         }
261
262         /**
263          * Discards this IntermediateProdict
264          */
265         public void discardThisIntermediateProduct(){
266                 if (!isDiscarded() && !isFinished()){
267                         discarded = true;
268                         if (processLogs.size()!=0){
269                                 int i = processLogs.size()-1;
270                                 processLogs.get(i).endProcess();
271                         }
272                         if (this.storingSpace!=null){
273                                 unsetStoringSpace();
274                         }
275                 }
276         }
277 }
```

IntermediateProductTest.java

```
1  package model;
2
3  import static org.junit.Assert.*;
4
5  import org.junit.Before;
6  import org.junit.Test;
7
8  /**
9   * @author M. C. H j
10  */
11
12 public class IntermediateProductTest {
13
14          private ProductType pT;
15          private IntermediateProduct iP;
16          private Depot dp ;
17
18          @Before
19          public void setUp() throws Exception {
20                  pT = new ProductType("test");
21                  ProcessLine pL = new ProcessLine("test", "none", pT);
22
23                  pL.createSubProcess(1, "ProcessTest", "none", 10, 10);
24                  pL.createDrying(2, 1, 2, 3);
25                  pL.createSubProcess(3, "ProcessTest", "none", 10, 10);
26                  pL.createSubProcess(4, "ProcessTest", "none", 10, 10);
27                  pL.createDrying(5, 1, 2, 3);
28                  pL.createSubProcess(6, "ProcessTest", "none", 10, 10);
29                  pL.createDrying(7, 1, 2, 3);
30                  pL.createSubProcess(8, "ProcessTest", "none", 10, 10);
31                  pL.createDrying(9, 1, 2, 3);
32
33                  iP = new IntermediateProduct("", pT, 0);
34
35                  dp = new Depot("", "", 2, 2);;
36          }
37
38          @Test
39          public void testConstructor() {
40                  IntermediateProduct iP1 = new IntermediateProduct("", pT, 0);
41                  assertEquals(pT,iP1.getProductType());
42                  assertTrue(pT.getIntermediateProducts().contains(iP1));
43                  assertEquals(0, iP1.getQuantity(),0.1);
44                  IntermediateProduct iP2 = new IntermediateProduct("", pT, 1.2);
45                  assertEquals(pT,iP2.getProductType());
46                  assertTrue(pT.getIntermediateProducts().contains(iP2));
47                  assertEquals(1.2, iP2.getQuantity(),0.1);
48                  ProductType pT1 = new ProductType("");
49                  IntermediateProduct iP3 = new IntermediateProduct("asd", pT1, 3000);
50                  assertEquals(pT1,iP3.getProductType());
51                  assertTrue(pT1.getIntermediateProducts().contains(iP3));
52                  assertEquals(3000, iP3.getQuantity(),0.1);
53          }
54
55          @Test (expected = RuntimeException.class)
56          public void testConstructor1() {
57                  new IntermediateProduct("", null, 0);
58          }
59
60          @Test (expected = RuntimeException.class)
61          public void testConstructor2() {
62                  new IntermediateProduct("", pT, -1);
63          }
64
65          @Test (expected = RuntimeException.class)
66          public void testConstructor3() {
67                  new IntermediateProduct("", pT, -301);
68          }
69
70          @Test (expected = RuntimeException.class)
71          public void testConstructor4() {
72                  new IntermediateProduct("", pT, -0.000001);
73          }
74
75          @Test (expected = RuntimeException.class)
```

```
76          public void testConstructor5() {
77                  new IntermediateProduct("", null, 12.5);
78          }
79
80          @Test
81          public void testSetId(){
82                  iP.setId("asdf");
83                  assertEquals("asdf", iP.getId());
84                  iP.setId("");
85                  assertEquals("", iP.getId());
86          }
87
88          @Test
89          public void testSetProductType(){
90                  ProductType pT1 = new ProductType("asdf");
91                  iP.setProductType(pT1);
92                  assertEquals(pT1, iP.getProductType());
93                  assertTrue(pT1.getIntermediateProducts().contains(iP));
94                  iP.setProductType(pT);
95                  assertEquals(pT, iP.getProductType());
96                  assertFalse(pT1.getIntermediateProducts().contains(iP));
97                  assertTrue(pT.getIntermediateProducts().contains(iP));
98          }
99
100         @Test (expected = RuntimeException.class)
101         public void testSetProductType1(){
102                 iP.setProductType(null);
103         }
104
105         @Test
106         public void testSetQuantity(){
107                 iP.setQuantity(2.2);
108                 assertEquals(2.2, iP.getQuantity(),0.1);
109                 iP.setQuantity(67.3);
110                 assertEquals(67.3, iP.getQuantity(),0.1);
111                 iP.setQuantity(500);
112                 assertEquals(500, iP.getQuantity(),0.1);
113                 iP.setQuantity(14);
114                 assertEquals(14, iP.getQuantity(),0.1);
115                 iP.setQuantity(3004.6);
116                 assertEquals(3004.6, iP.getQuantity(),0.1);
117                 iP.setQuantity(0);
118                 assertEquals(0, iP.getQuantity(),0.1);
119
120         }
121
122         @Test (expected = RuntimeException.class)
123         public void testSetQuantity1(){
124                 iP.setQuantity(-0.0001);
125         }
126
127         @Test (expected = RuntimeException.class)
128         public void testSetQuantity2(){
129                 iP.setQuantity(-14.1);
130         }
131
132         @Test (expected = RuntimeException.class)
133         public void testSetQuantity3(){
134                 iP.setQuantity(-16);
135         }
136
137         @Test (expected = RuntimeException.class)
138         public void testSetQuantity4(){
139                 iP.setQuantity(-300.3);
140         }
141
142         @Test
143         public void testSetStoringSpace (){
144                 iP.setStoringSpace(dp.getStoringSpaces().get(0));
145                 assertEquals(dp.getStoringSpaces().get(0), iP.getStoringSpace());
146                 assertEquals(iP,dp.getStoringSpaces().get(0).getIntermediateProduct());
147
148                 iP.setStoringSpace(dp.getStoringSpaces().get(1));
149                 assertEquals(dp.getStoringSpaces().get(1), iP.getStoringSpace());
150                 assertEquals(iP,dp.getStoringSpaces().get(1).getIntermediateProduct());
151                 assertNull(dp.getStoringSpaces().get(0).getIntermediateProduct());
152
153                 iP.unsetStoringSpace();
```

```
154                 assertNull(iP.getStoringSpace());
155                 assertNull(dp.getStoringSpaces().get(1).getIntermediateProduct());
156         }
157
158         @Test
159         public void testCreateProcessLog(){
160                 ProcessLog pLog1 = iP.createProcessLog(pT.getProcessLine().getProcesses().get(0), null)
                        ;
161                 assertEquals(1,iP.getProcessLogs().size());
162                 assertTrue(pT.getProcessLine().getProcesses().get(0).getProcessLogs().contains(pLog1));
163                 ProcessLog pLog2 = iP.createProcessLog(pT.getProcessLine().getProcesses().get(1), dp.
                        getStoringSpaces().get(0));
164                 assertEquals(2,iP.getProcessLogs().size());
165                 assertTrue(pT.getProcessLine().getProcesses().get(1).getProcessLogs().contains(pLog2));
166                 assertTrue(dp.getStoringSpaces().get(0).getProcessLogs().contains(pLog2));
167                 iP.deleteProcessLog(pLog1);
168                 assertFalse(pT.getProcessLine().getProcesses().get(0).getProcessLogs().contains(pLog1))
                        ;
169                 assertEquals(1,iP.getProcessLogs().size());
170                 iP.deleteProcessLog(pLog2);
171                 assertFalse(pT.getProcessLine().getProcesses().get(1).getProcessLogs().contains(pLog2))
                        ;
172                 assertFalse(dp.getStoringSpaces().get(0).getProcessLogs().contains(pLog2));
173                 assertEquals(0,iP.getProcessLogs().size());
174         }
175
176         @Test
177         public void TestProcessHandling(){
178                 //before start
179                 assertFalse(iP.isFinished());
180                 assertFalse(iP.isDiscarded());
181                 assertNull(iP.getActivProcessLog());
182                 assertEquals(pT.getProcessLine().getProcesses().get(0), iP.getNextProcess());
183                 assertEquals(0, iP.getProcessLogs().size());
184
185                 //first process
186                 iP.sendToNextProcess(null);
187
188                 assertFalse(iP.isFinished());
189                 assertFalse(iP.isDiscarded());
190                 assertEquals(pT.getProcessLine().getProcesses().get(0), iP.getActivProcessLog().
                        getProcess());
191                 assertEquals(pT.getProcessLine().getProcesses().get(1), iP.getNextProcess());
192                 assertEquals(1, iP.getProcessLogs().size());
193                 assertNull(iP.getStoringSpace());
194                 assertTrue(iP.getActivProcessLog().isActive());
195
196                 //sekond prcess
197                 iP.sendToNextProcess(dp.getStoringSpaces().get(0));
198
199                 assertFalse(iP.isFinished());
200                 assertFalse(iP.isDiscarded());
201                 assertEquals(pT.getProcessLine().getProcesses().get(1), iP.getActivProcessLog().
                        getProcess());
202                 assertEquals(pT.getProcessLine().getProcesses().get(2), iP.getNextProcess());
203                 assertEquals(2, iP.getProcessLogs().size());
204                 assertEquals(dp.getStoringSpaces().get(0), iP.getStoringSpace());
205                 assertEquals(iP, dp.getStoringSpaces().get(0).getIntermediateProduct());
206                 assertTrue(iP.getActivProcessLog().isActive());
207                 assertFalse(iP.getProcessLogs().get(0).isActive());
208
209                 //third prcess
210                 iP.sendToNextProcess(null);
211
212                 assertFalse(iP.isFinished());
213                 assertFalse(iP.isDiscarded());
214                 assertEquals(pT.getProcessLine().getProcesses().get(2), iP.getActivProcessLog().
                        getProcess());
215                 assertEquals(pT.getProcessLine().getProcesses().get(3), iP.getNextProcess());
216                 assertEquals(3, iP.getProcessLogs().size());
217                 assertNull(iP.getStoringSpace());
218                 assertNull(dp.getStoringSpaces().get(0).getIntermediateProduct());
219                 assertTrue(iP.getActivProcessLog().isActive());
220                 assertFalse(iP.getProcessLogs().get(1).isActive());
221
222                 //Skipping the middle processes
223                 iP.sendToNextProcess(null);
224                 iP.sendToNextProcess(null);
```

```
225                 iP.sendToNextProcess(null);
226                 iP.sendToNextProcess(null);
227                 //second to last process
228                 iP.sendToNextProcess(null);
229 
230                 assertFalse(iP.isFinished());
231                 assertFalse(iP.isDiscarded());
232                 assertEquals(pT.getProcessLine().getProcesses().get(7), iP.getActivProcessLog().
                        getProcess());
233                 assertEquals(pT.getProcessLine().getProcesses().get(8), iP.getNextProcess());
234                 assertEquals(8, iP.getProcessLogs().size());
235                 assertFalse(iP.getProcessLogs().get(0).isActive());
236                 assertFalse(iP.getProcessLogs().get(1).isActive());
237                 assertFalse(iP.getProcessLogs().get(2).isActive());
238                 assertFalse(iP.getProcessLogs().get(3).isActive());
239                 assertFalse(iP.getProcessLogs().get(4).isActive());
240                 assertFalse(iP.getProcessLogs().get(5).isActive());
241                 assertFalse(iP.getProcessLogs().get(6).isActive());
242                 assertTrue(iP.getActivProcessLog().isActive());
243 
244                 //last process
245                 iP.sendToNextProcess(null);
246 
247                 assertFalse(iP.isFinished());
248                 assertFalse(iP.isDiscarded());
249                 assertEquals(pT.getProcessLine().getProcesses().get(8), iP.getActivProcessLog().
                        getProcess());
250                 assertNull(iP.getNextProcess());
251                 assertEquals(9, iP.getProcessLogs().size());
252                 assertTrue(iP.getActivProcessLog().isActive());
253                 assertFalse(iP.getProcessLogs().get(7).isActive());
254 
255                 //intermediateproduct is finished
256                 iP.sendToNextProcess(null);
257 
258                 assertTrue(iP.isFinished());
259                 assertFalse(iP.isDiscarded());
260                 assertNull(iP.getActivProcessLog());
261                 assertNull(iP.getNextProcess());
262                 assertEquals(9, iP.getProcessLogs().size());
263                 assertFalse(iP.getProcessLogs().get(8).isActive());
264         }
265 
266         @Test
267         public void testDiscardMethod(){
268                 //before start
269                 assertFalse(iP.isFinished());
270                 assertFalse(iP.isDiscarded());
271                 assertNull(iP.getActivProcessLog());
272                 assertEquals(pT.getProcessLine().getProcesses().get(0), iP.getNextProcess());
273                 assertEquals(0, iP.getProcessLogs().size());
274 
275                 //first process
276                 iP.sendToNextProcess(null);
277 
278                 assertFalse(iP.isFinished());
279                 assertFalse(iP.isDiscarded());
280                 assertEquals(pT.getProcessLine().getProcesses().get(0), iP.getActivProcessLog().
                        getProcess());
281                 assertEquals(pT.getProcessLine().getProcesses().get(1), iP.getNextProcess());
282                 assertEquals(1, iP.getProcessLogs().size());
283                 assertNull(iP.getStoringSpace());
284                 assertTrue(iP.getActivProcessLog().isActive());
285 
286                 //sekond prcess
287                 iP.sendToNextProcess(dp.getStoringSpaces().get(0));
288 
289                 assertFalse(iP.isFinished());
290                 assertFalse(iP.isDiscarded());
291                 assertEquals(pT.getProcessLine().getProcesses().get(1), iP.getActivProcessLog().
                        getProcess());
292                 assertEquals(pT.getProcessLine().getProcesses().get(2), iP.getNextProcess());
293                 assertEquals(2, iP.getProcessLogs().size());
294                 assertEquals(dp.getStoringSpaces().get(0), iP.getStoringSpace());
295                 assertTrue(iP.getActivProcessLog().isActive());
296                 assertFalse(iP.getProcessLogs().get(0).isActive());
297 
298                 //testing discard method
```

```
299                     iP.discardThisIntermediateProduct();
300                     assertFalse(iP.isFinished());
301                     assertTrue(iP.isDiscarded());
302                     assertNull(iP.getActivProcessLog());
303                     assertNull(iP.getNextProcess());
304                     assertEquals(2, iP.getProcessLogs().size());
305                     assertNull(iP.getStoringSpace());
306                     assertFalse(iP.getProcessLogs().get(0).isActive());
307                     assertFalse(iP.getProcessLogs().get(1).isActive());
308
309             }
310
311     }
```

Process.java

```
1  package model;
2
3  import java.util.ArrayList;
4
5  /**
6   * @author M. C. H j
7   */
8
9  public abstract class Process {
10         private int ProcessStep;
11         private ProcessLine processLine;
12         private ArrayList<ProcessLog> processLogs  = new ArrayList<ProcessLog>();
13
14         public Process(int processStep, ProcessLine processLine) throws RuntimeException{
15                 if (processLine==null){
16                         throw new RuntimeException("processLine can't be set to null");
17                 } else {
18                         this.setProcessStep(processStep);
19                         this.processLine=processLine;
20                 }
21         }
22
23         public int getProcessStep(){
24                 return this.ProcessStep;
25         }
26
27         public void setProcessStep(int processStep)throws RuntimeException{
28                 if (processStep<0) {
29                         throw new RuntimeException("processStep can't be a negative number");
30                 } else {
31                         this.ProcessStep=processStep;
32                 }
33         }
34
35         public ProcessLine getProcessLine(){
36                 return this.processLine;
37         }
38
39         public ArrayList<ProcessLog> getProcessLogs(){
40                 return this.processLogs;
41         }
42
43         public void addProcessLog(ProcessLog processLog){
44                 this.processLogs.add(processLog);
45                 if (processLog.getProcess()!=this){
46                         processLog.setProcess(this);
47                 }
48         }
49
50         /**
51      * always call this method trougth setProcess method in ProcessLog
52      * @param intermediateProduct
53      */
54         public void removeProcessLog(ProcessLog processLog) throws RuntimeException{
55                 this.processLogs.remove(processLog);
56         }
57
58 }
```

ProcessLine.java

```
1  package model;
2
3  import java.util.ArrayList;
4
5  /**
6   * @author M. C. H j
7   */
8
9  public class ProcessLine {
10         private String name;
11         private String description;
12         private ProductType productType;
13         private ArrayList<Process> processes  = new ArrayList<Process>();
14
15         public ProcessLine(String name, String description, ProductType productType) throws
               RuntimeException{
16                 if ( productType==null){
17                         throw new RuntimeException("produktType can't be null");
18                 } else {
19                         this.setName(name);
20                         this.setDescription(description);
21                         this.productType=productType;
22                         productType.setProcessLine(this);
23                 }
24         }
25
26         public String getName(){
27                 return this.name;
28         }
29
30         public void setName(String name){
31                 this.name=name;
32         }
33
34         public String getDescription(){
35                 return this.description;
36         }
37
38         public void setDescription(String description){
39                 this.description=description;
40         }
41
42         public ProductType getProductType(){
43                 return this.productType;
44         }
45
46         public ArrayList<Process> getProcesses(){
47                 return this.processes;
48         }
49
50         public SubProcess createSubProcess(int processStep, String name, String desciption, long
               treatmentTime, double temperature){
51                 SubProcess sp = new SubProcess(name, desciption, treatmentTime, temperature,
                       processStep, this);
52
53                 this.processes.add(sp);
54
55
56                 return sp;
57
58         }
59
60         public Drying createDrying(int processStep, long minTime, long idealTime, long maxTime) throws
               RuntimeException{
61                 Drying d = new Drying(minTime, idealTime, maxTime, processStep, this);
62
63                 this.processes.add(d);
64
65                 return d;
66         }
67
68         public void deleteProcess(Process process){
69                 this.processes.remove(process);
70         }
71
```

```
72 }
```

ProcessLog.java

```
1  package model;
2
3  import java.sql.Date;
4
5  /**
6   * @author M. C. H j
7   */
8
9  public class ProcessLog{
10         private long startTime=0;
11         private long endTime=0;
12         private Process process;
13         private StoringSpace storingSpace;
14         private IntermediateProduct intermediateProduct;
15
16         public ProcessLog(Process process, StoringSpace storingSpace, IntermediateProduct
               intermediateProduct) throws RuntimeException{
17             if (intermediateProduct==null){
18                     throw new RuntimeException("intermediateProduct can't be set to null");
19             } else {
20                     this.setProcess(process);
21                     if (storingSpace==null){
22                             this.storingSpace=null;
23                     } else {
24                             this.setStoringSpace(storingSpace);
25                     }
26                     this.intermediateProduct=intermediateProduct;
27                     this.startTime =System.currentTimeMillis();
28             }
29         }
30
31         public Date getStartTime(){
32                 return  new Date(this.startTime);
33         }
34
35         public Date getEndTime(){
36                 return  new Date(this.endTime);
37         }
38
39         public void endProcess(){
40                 this.endTime = System.currentTimeMillis();
41         }
42
43         public boolean isActive(){
44                 return this.endTime==0;
45         }
46
47         public Process getProcess(){
48                 return this.process;
49         }
50
51         public void setProcess(Process process) throws RuntimeException{
52                 if (process==null){
53                         throw new RuntimeException("process can't be null");
54                 } else {
55                         if (this.process != null){
56                                 this.process.removeProcessLog(this);
57                         }
58                         this.process=process;
59                         if (!process.getProcessLogs().contains(this)){
60                                 process.addProcessLog(this);
61                         }
62                 }
63         }
64
65         public StoringSpace getStoringSpace(){
66                 return this.storingSpace;
67         }
68
69         public void setStoringSpace(StoringSpace storingSpace){
70                 if (this.storingSpace != null){
71                         this.storingSpace.removeProcessLog(this);
72                 }
73                 this.storingSpace=storingSpace;
74                 if (!storingSpace.getProcessLogs().contains(this)){
```

```
75                          storingSpace.addProcessLog(this);
76                  }
77          }
78
79          public void unsetStoringSpace(){
80                  StoringSpace oldStoringSpace = this.storingSpace;
81                  this.storingSpace = null;
82                  if (oldStoringSpace.getProcessLogs().contains(this)){
83                          oldStoringSpace.removeProcessLog(this);
84                  }
85          }
86
87          public IntermediateProduct getIntermediateProduct(){
88                  return this.intermediateProduct;
89          }
90
91          public String toString() {
92                  return process.toString();
93          }
94
95  }
```

ProductType.java

```
1  package model;
2
3  import java.util.ArrayList;
4
5  import javax.swing.Icon;
6  import javax.swing.ImageIcon;
7
8  import com.sun.image.codec.jpeg.ImageFormatException;
9  import com.sun.image.codec.jpeg.JPEGCodec;
10 import com.sun.image.codec.jpeg.JPEGImageEncoder;
11
12 /**
13  * @author M. C. H j
14  */
15
16
17 public class ProductType {
18         private String name;
19         private ProcessLine processLine = null;
20         private ImageIcon picture = null;
21         private ArrayList<IntermediateProduct> intermediateProducts  = new ArrayList<
               IntermediateProduct>();
22
23         public ProductType(String name){
24                 this.setName(name);
25         }
26
27         public String getName(){
28                 return this.name;
29         }
30
31         public void setName(String name){
32                 this.name=name;
33         }
34
35         public ProcessLine getProcessLine(){
36                 return this.processLine;
37         }
38
39         public ImageIcon getPicture() {
40                 return picture;
41         }
42
43         /**
44          * not recomended to call this method
45          * @param processLine
46          */
47         public void setProcessLine(ProcessLine processLine){
48                 this.processLine=processLine;
49         }
50
51         public ArrayList<IntermediateProduct> getIntermediateProducts(){
52                 return this.intermediateProducts;
53         }
54
55         public void setPicture(ImageIcon picture) {
56                 this.picture = picture;
57         }
58
59         public void addIntermediateProduct(IntermediateProduct intermediateProduct){
60                 this.intermediateProducts.add(intermediateProduct);
61                 if (intermediateProduct.getProductType()!=this){
62                         intermediateProduct.setProductType(this);
63                 }
64         }
65
66         /**
67          * always call this method trougth setProductType method in IntermediateProduct
68          * @param intermediateProduct
69          */
70         public void removeIntermediateProduct(IntermediateProduct intermediateProduct){
71                 this.intermediateProducts.remove(intermediateProduct);
72         }
73
74         public String toString() {
```

```
75                  return this.name;
76          }
77
78  }
```

StoringSpace.java

```
1  package model;
2
3  import java.util.ArrayList;
4
5  /**
6   * @author M. C. H j
7   */
8
9  public class StoringSpace {
10         private int positionX;
11         private int positionY;
12         private IntermediateProduct intermediateProduct;
13         private Depot depot;
14         private ArrayList<ProcessLog> processLogs  = new ArrayList<ProcessLog>();
15
16         public StoringSpace(int positionX, int positionY, Depot depot) throws RuntimeException{
17                 if (depot==null){
18                         throw new RuntimeException("depot can't be null");
19                 } else {
20                         this.setPositionX(positionX);
21                         this.setPositionY(positionY);
22                         this.depot=depot;
23                 }
24         }
25
26         public int getPositionX(){
27                 return this.positionX;
28         }
29
30         public void setPositionX(int positionX){
31                 if (positionX<0){
32                         throw new RuntimeException("positionX can't be a negativ number");
33                 } else {
34                         this.positionX=positionX;
35                 }
36         }
37
38         public int getPositionY(){
39                 return this.positionY;
40         }
41
42         public void setPositionY(int positionY){
43                 if (positionY<0){
44                         throw new RuntimeException("positionY can't be a negativ number");
45                 } else {
46                         this.positionY=positionY;
47                 }
48         }
49
50         public IntermediateProduct getIntermediateProduct(){
51                 return this.intermediateProduct;
52         }
53
54         public void setIntermediateProduct(IntermediateProduct intermediateProduct){
55                 if (this.intermediateProduct != null){
56                         this.intermediateProduct.unsetStoringSpace();
57                 }
58                 this.intermediateProduct=intermediateProduct;
59                 if (intermediateProduct.getStoringSpace()!=this){
60                         intermediateProduct.setStoringSpace(this);
61                 }
62         }
63         public void unsetIntermediateProduct(){
64                 IntermediateProduct oldIntermediateProduct = this.intermediateProduct;
65                 this.intermediateProduct = null;
66                 if (oldIntermediateProduct.getStoringSpace()!=null){
67                         oldIntermediateProduct.unsetStoringSpace();
68                 }
69     }
70
71         public Depot getDepot(){
72                 return this.depot;
73         }
74
75         public ArrayList<ProcessLog> getProcessLogs(){
```

```
76                  return this.processLogs;
77          }
78
79          public void addProcessLog(ProcessLog processLog){
80                  this.processLogs.add(processLog);
81                  if (processLog.getStoringSpace() != this){
82                          processLog.setStoringSpace(this);
83                  }
84          }
85
86          public void removeProcessLog(ProcessLog preocessLog){
87                  this.processLogs.remove(preocessLog);
88                  if (preocessLog.getStoringSpace() != null){
89                  preocessLog.unsetStoringSpace();
90                  }
91          }
92
93          public String toString() {
94                  return "("+positionX+","+positionY+")";
95          }
96
97  }
```

4 service.*

Service.java

```
1  package service;
2
3  import java.io.File;
4  import java.util.ArrayList;
5  import java.util.Collection;
6  import java.util.Iterator;
7  import java.util.List;
8  import java.util.ListIterator;
9  import java.util.Random;
10
11 import javax.swing.ImageIcon;
12
13 import model.*;
14 import model.Process;
15
16 import dao.Dao;
17 import dao.DaoDb4o;
18 import dao.DaoList;
19
20 /**
21  * @author Brian, M. C. H j
22  */
23
24 public class Service {
25         private static Service service = null;
26         private Dao dao = null;
27
28         private Service() {
29
30                 dao = DaoList.getDao();
31                 createTestListData();
32
33 //              boolean isCreated = new File("db.db4o").exists();
34 //              dao = DaoDb4o.getDao();
35 //              if (!isCreated) {
36 //                      createTestListData();
37 //              }
38
39         }
40
41         public static Service getService() {
42                 if (service == null) {
43                         service = new Service();
44                 }
45                 return service;
46         }
47
48         //Depot
49         public List<Depot> getAllDepots() {
50                 return dao.getAllDepots();
51         }
52
53         public Depot createDepot(String name, String description,int maxX, int maxY) {
54                 Depot depot = new Depot(name, description,maxX,maxY);
55                 dao.store(depot);
56                 return depot;
57         }
58
59         public void deleteDepot(Depot depot) {
60                 dao.delete(depot);
61         }
62
63         //IntermediateProduct
64         public List<IntermediateProduct> getAllIntermediateProducts() {
65                 return dao.getAllIntermediateProducts();
66         }
67
68         public IntermediateProduct createIntermediateProduct(String id, ProductType productType, double
               quantity) {
69                 IntermediateProduct intermediateProduct = new IntermediateProduct(id, productType,
                       quantity);
70                 dao.store(intermediateProduct);
71                 return intermediateProduct;
72         }
73
```

```
74          public void StoreIntermediateProduct(IntermediateProduct intermediateProduct) {
75                  dao.store(intermediateProduct);
76          }
77
78          public void deleteIntermediateProduct(IntermediateProduct intermediateProduct) {
79                  dao.delete(intermediateProduct);
80          }
81
82          public List<IntermediateProduct> getActiveIntermediateProducts(){
83                  List<IntermediateProduct> activeP= new ArrayList<IntermediateProduct>();
84                  List<IntermediateProduct> allP= dao.getAllIntermediateProducts();
85                  for (int i = 0; i < allP.size(); i++) {
86                          if (!allP.get(i).isFinished() && !allP.get(i).isDiscarded()){
87                                  activeP.add(allP.get(i));
88                          }
89                  }
90
91                  return activeP;
92          }
93
94          //ProductType
95          public List<ProductType> getAllProductTypes() {
96                  return dao.getAllProductTypes();
97          }
98
99          public ProductType createProductType(String name) {
100                 ProductType productType = new ProductType(name);
101                 dao.store(productType);
102                 return productType;
103         }
104
105         public void storeProductType(ProductType productType){
106                 dao.store(productType);
107         }
108
109         public void deleteProductType(ProductType productType) {
110                 dao.delete(productType);
111         }
112
113         public void closeDao(){
114                 dao.close();
115         }
116
117         /**
118          * oprettelse af test data
119          */
120         public void createTestListData() {
121
122                 Depot depot1 = createDepot("Lager 1","Hovedlageret",5,8);
123                 Depot depot2 = createDepot("Lager 2","Lager til lort",4,5);
124
125                 ProductType pteSkumbananer = createProductType("Skumbananer");
126                 pteSkumbananer.setPicture(new ImageIcon("gui/icons/skumbananer.jpg"));
127                 ProcessLine plSkumbananer = new ProcessLine("Skumbananer", "Skum", pteSkumbananer);
128                 plSkumbananer.createSubProcess(1, "Tilfoej skum", "siger sig selv", 2, 24);
129                 Drying d1 = plSkumbananer.createDrying(2, 1, 2*60*10000, 3*60*10000);
130                 d1.addDepot(depot1); d1.addDepot(depot2);
131                 plSkumbananer.createSubProcess(3, "Tilsaet chokolade", "siger sig selv", 1, 100);
132                 Drying d2 = plSkumbananer.createDrying(4, 1, 2*60*10000, 3*60*10000);
133                 d2.addDepot(depot2);
134
135                 ProductType pteChokoKaramelLys = createProductType("Choko Karamel Lys");
136                 pteChokoKaramelLys.setPicture(new ImageIcon("gui/icons/choko karamel lys.jpg"));
137                 ProcessLine plChokoKaramelLys = new ProcessLine("Choko Karamel Lys", "asdf",
                        pteChokoKaramelLys);
138                 plChokoKaramelLys.createSubProcess(1, "Tilsaet karamel", "Lys karamel", 1, 2);
139                 Drying d3 = plChokoKaramelLys.createDrying(21, 1*60*10000, 2*60*10000, 3*60*10000);
140                 d3.addDepot(depot1); d3.addDepot(depot2);
141
142                 ProductType pteChokoKaramelMoerk = createProductType("Choko Karamel Moerk");
143                 pteChokoKaramelMoerk.setPicture(new ImageIcon("gui/icons/choko karamel moerk.jpg"));
144                 ProcessLine plChokoKaramelMoerk = new ProcessLine("Choko Karamel Moerk", "asdf2",
                        pteChokoKaramelMoerk);
145                 plChokoKaramelMoerk.createSubProcess(1, "Tilsaet karamel", "Moerk karamel", 1, 2);
146                 Drying d4 = plChokoKaramelMoerk.createDrying(2, 3*60*10000, 4*60*10000, 5*60*10000);
147                 d4.addDepot(depot1); d4.addDepot(depot2);
148
149                 ProductType pteChokoladelinser = createProductType("Chokoladelinser");
```

```
150                 pteChokoladelinser.setPicture(new ImageIcon("gui/icons/chokoladelinser.jpg"));
151                 ProcessLine plChokoladelinser = new ProcessLine("Chokoladelinser", "nam",
                        pteChokoladelinser);
152                 plChokoladelinser.createSubProcess(1, "Tilsaetter chokolade", "tilfoejer chokolade",24,
                        -3);
153                 Drying d5 = plChokoladelinser.createDrying(2, 30*10000, 60*10000, 120*10000);
154                 d5.addDepot(depot1); d5.addDepot(depot2);
155                 plChokoladelinser.createSubProcess(3, "Tilsaetter linser", "Tilsaetter linser fra
                        optikkeren", 3, 13);
156                 Drying d6 = plChokoladelinser.createDrying(4, 1*60*10000, 2*60*10000, 3*60*10000);
157                 d6.addDepot(depot1); d6.addDepot(depot2);
158
159                 ProductType pteCitronDrage = createProductType("Citron Drage");
160                 pteCitronDrage.setPicture(new ImageIcon("gui/icons/citron drage.jpg"));
161                 ProcessLine plCitronDrage = new ProcessLine("Citron Drag ", "ild", pteCitronDrage);
162                 plCitronDrage.createSubProcess(1, "Tilsaetter citron", "press en eller to citroner og
                        put dem i",21, -45);
163                 Drying d7 = plCitronDrage.createDrying(2, 1*60*10000, 2*60*10000, 3*60*10000);
164                 d7.addDepot(depot1); d7.addDepot(depot2);
165                 plCitronDrage.createSubProcess(3, "Tilsaetter linser", "Tilsaetter linser fra
                        optikkeren", 17, 87);
166                 Drying d8 = plCitronDrage.createDrying(4, 4*60*10000, 7*60*10000, 8*60*10000);
167                 d8.addDepot(depot1); d8.addDepot(depot2);
168
169                 createIntermediateProduct("011", pteSkumbananer, 80);
170                 createIntermediateProduct("012", pteCitronDrage, 80);
171                 createIntermediateProduct("013", pteChokoladelinser, 100);
172                 createIntermediateProduct("014", pteChokoKaramelMoerk, 100);
173                 createIntermediateProduct("015", pteChokoKaramelLys, 140);
174
175                 createIntermediateProduct("021", pteSkumbananer, 80);
176                 createIntermediateProduct("022", pteCitronDrage, 80);
177                 createIntermediateProduct("023", pteChokoladelinser, 100);
178                 createIntermediateProduct("024", pteChokoKaramelMoerk, 100);
179                 createIntermediateProduct("025", pteChokoKaramelLys, 140);
180
181
182                 for (int i = 0; i < 5; i++) {
183                         getAllIntermediateProducts().get(i).sendToNextProcess(null);
184                         getAllIntermediateProducts().get(i).sendToNextProcess(depot1.getStoringSpaces()
                                .get(i));
185                 }
186
187                 for (int i = 5; i < 10; i++) {
188                         getAllIntermediateProducts().get(i).sendToNextProcess(null);
189                         getAllIntermediateProducts().get(i).sendToNextProcess(depot2.getStoringSpaces()
                                .get(i-5));
190                 }
191
192         }
193
194         public void createTestDB40Data() {
195                 Depot depot1 = createDepot("Lager 1","Hovedlageret",5,8);
196                 Depot depot2 = createDepot("Lager 2","Lager til lort",4,5);
197
198                 ProductType pteSkumbananer = createProductType("Skumbananer");
199                 pteSkumbananer.setPicture(new ImageIcon("gui/icons/skumbananer.jpg"));
200                 ProcessLine plSkumbananer = new ProcessLine("Skumbananer", "Skum", pteSkumbananer);
201                 plSkumbananer.createSubProcess(1, "Tilfoej skum", "siger sig selv", 2, 24);
202                 Drying d1 = plSkumbananer.createDrying(2, 1, 2*60*1000, 3*60*1000);
203                 d1.addDepot(depot1); d1.addDepot(depot2);
204                 plSkumbananer.createSubProcess(3, "Tilsaet chokolade", "siger sig selv", 1, 100);
205                 Drying d2 = plSkumbananer.createDrying(4, 1, 2*60*1000, 3*60*1000);
206                 d2.addDepot(depot2);
207
208                 storeProductType(pteSkumbananer);
209
210                 ProductType pteChokoKaramelLys = createProductType("Choko Karamel Lys");
211                 pteChokoKaramelLys.setPicture(new ImageIcon("gui/icons/choko karamel lys.jpg"));
212                 ProcessLine plChokoKaramelLys = new ProcessLine("Choko Karamel Lys", "asdf",
                        pteChokoKaramelLys);
213                 plChokoKaramelLys.createSubProcess(1, "Tilsaet karamel", "Lys karamel", 1, 2);
214                 Drying d3 = plChokoKaramelLys.createDrying(21, 1*60*1000, 2*60*1000, 3*60*1000);
215                 d3.addDepot(depot1); d3.addDepot(depot2);
216
217                 storeProductType(pteChokoKaramelLys);
218
219                 ProductType pteChokoKaramelMoerk = createProductType("Choko Karamel Moerk");
```

```
220                 pteChokoKaramelMoerk.setPicture(new ImageIcon("gui/icons/choko karamel moerk.jpg"));
221                 ProcessLine plChokoKaramelMoerk = new ProcessLine("Choko Karamel Moerk", "asdf2",
                        pteChokoKaramelMoerk);
222                 plChokoKaramelMoerk.createSubProcess(1, "Tilsaet karamel", "Moerk karamel", 1, 2);
223                 Drying d4 = plChokoKaramelMoerk.createDrying(2, 3*60*1000, 4*60*1000, 5*60*1000);
224                 d4.addDepot(depot1); d4.addDepot(depot2);
225
226                 storeProductType(pteChokoKaramelMoerk);
227
228                 ProductType pteChokoladelinser = createProductType("Chokoladelinser");
229                 pteChokoladelinser.setPicture(new ImageIcon("gui/icons/chokoladelinser.jpg"));
230                 ProcessLine plChokoladelinser = new ProcessLine("Chokoladelinser", "nam",
                        pteChokoladelinser);
231                 plChokoladelinser.createSubProcess(1, "Tilsaetter chokolade", "tilfoejer chokolade",24,
                        -3);
232                 Drying d5 = plChokoladelinser.createDrying(2, 30*1000, 60*1000, 120*1000);
233                 d5.addDepot(depot1); d5.addDepot(depot2);
234                 plChokoladelinser.createSubProcess(3, "Tilsaetter linser", "Tilsaetter linser fra
                        optikkeren", 3, 13);
235                 Drying d6 = plChokoladelinser.createDrying(4, 1*60*1000, 2*60*1000, 3*60*1000);
236                 d6.addDepot(depot1); d6.addDepot(depot2);
237
238                 storeProductType(pteChokoladelinser);
239
240                 ProductType pteCitronDrage = createProductType("Citron Drage");
241                 pteCitronDrage.setPicture(new ImageIcon("gui/icons/citron drage.jpg"));
242                 ProcessLine plCitronDrage = new ProcessLine("Citron Drag ", "ild", pteCitronDrage);
243                 plCitronDrage.createSubProcess(1, "Tilsaetter citron", "press en eller to citroner og
                        put dem i",21, -45);
244                 Drying d7 = plCitronDrage.createDrying(2, 1*60*1000, 2*60*1000, 3*60*1000);
245                 d7.addDepot(depot1); d7.addDepot(depot2);
246                 plCitronDrage.createSubProcess(3, "Tilsaetter linser", "Tilsaetter linser fra
                        optikkeren", 17, 87);
247                 Drying d8 = plCitronDrage.createDrying(4, 4*60*1000, 7*60*1000, 8*60*1000);
248                 d8.addDepot(depot1); d8.addDepot(depot2);
249
250                 storeProductType(pteCitronDrage);
251
252         }
253
254 }
```