```java
package weightsimulator.controller;

import java.util.ArrayList;

/**
 * MainController - integrating input from socket and ui. Implements ISocketObserver and IUIObserver to handle this.
 * @author Christian Budtz
 * @version 0.1 2017-01-24
 *
 */
public class MainController implements IMainController, ISocketObserver, IWeightInterfaceObserver {

    private ISocketController socketHandler;
    private IWeightInterfaceController weightController;
    private KeyState keyState = KeyState.K4;
    private Connector conn;

    private Double total = 0.0;
    private List<Character> numbers = new ArrayList<Character>();
    private int numbersPointer = 0;
    private String numberMessage;
    private boolean allowCommands = true;
    private int tempOutput = 0;
    private boolean toleranceFail = true;
    private double upperTolerance, lowerTolerance;
    private boolean key0 = true, key1 = false;

    private int opr_id, rb_id, pb_id;
    private double weight = 0.0;
    private double tarWeight = 0.0;

    public MainController(ISocketController socketHandler, IWeightInterfaceController weightInterfaceController) {
        this.init(socketHandler, weightInterfaceController);
    }

    @Override
    public void init(ISocketController socketHandler, IWeightInterfaceController weightInterfaceController) {
        this.socketHandler = socketHandler;
        this.weightController=weightInterfaceController;
        try {
            this.conn = new Connector();
        } catch (InstantiationException e) {
            e.printStackTrace();
        } catch (IllegalAccessException e) {
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void start() {
        if (socketHandler!=null && weightController!=null){
            //Makes this controller interested in messages from the socket
            socketHandler.registerObserver(this);
            //Starts socketHandler in it's own thread
```

```
 97            new Thread(socketHandler).start();
 98            //weightController setup
 99            weightController.registerObserver(this);
100            //Starts weightController in it's own thread
101            new Thread(weightController).start();
102        } else {
103            System.err.println("No controllers injected!");
104        }
105    }
106
107    //Listening for socket input
108    //When we notify observers, this is the controller that gets the input
109    @Override
110    public void notify(SocketInMessage message) {
111        if(allowCommands){
112            switch (message.getType()) {
113            case B:
114                try{
115                    if (Double.parseDouble(message.getMessage()) < -weight){
116                        weightController.showMessageSecondaryDisplay("Cant withdraw more
    weight than currently on weight");
117                    } else{
118                        weight = weight + Double.parseDouble(message.getMessage());
119                        weightController.showMessagePrimaryDisplay(weight+"kg");
120                        weightController.showMessageSecondaryDisplay("Unmodified total
    weight:");
121                    }
122                    break;
123                }
124                catch(NumberFormatException e){
125                    weightController.showMessageSecondaryDisplay("Error: Wrong format " +
    e.getMessage());
126                    break;
127                }
128            case D:
129                weightController.showMessagePrimaryDisplay(message.getMessage());
130                socketHandler.sendMessage(new SocketOutMessage("D A"));
131                break;
132            case Q:
133                quit();
134                break;
135            case RM204: //Not implemented. Same functionality as RM208
136                break;
137            case RM208:
138                pb_id = 0;
139                key0 = false;   //Disable ZERO button
140                allowCommands = false;
141                synchronized(this){
142                    try {
143                        try {
144                            doName();                          //User identification
145                        } catch (DALException e1) {
146                            e1.printStackTrace();
147                        }
148                        try {
149                            doPB();                            //Productbatch identification
150                            setPbStatus(pb_id, 1);
151                        } catch (DALException e1) {
152                            e1.printStackTrace();
153                        }
154                        try {
155                            int raavareCount = getRowCount(pb_id);
```

```
156                             List<String> names = getProductName(pb_id);
157                             ReceptKompDAO rkDAO = new MySQLReceptKomponentDAO();
158                             List<ReceptKompDTO> rkList = rkDAO.getReceptKompList();
159                             List<Integer> rbIdList = getRaavareBatchID(pb_id);
160 
161                             for(int i = 0; i < raavareCount ; i++){
162                                 if(i == 0){
163                                     weightController.showMessageSecondaryDisplay("Productba
    tch ID set. Place container on weight and tara.");
164                                 } else {
165                                     weightController.showMessageSecondaryDisplay("Productba
    tchcomponent set. Place new container on weight and tara.");
166                                 }
167                                 key1 = true;
168                                 this.wait();
169                                 weightController.showMessageSecondaryDisplay("Tara set.
    Bring: \"" + names.get(i) + "\" and enter commodity batch ID");
170                                 do{
171                                     this.wait();
172                                     if(tempOutput != rbIdList.get(i)){
173                                         weightController.showMessageSecondaryDisplay("Given
    Id does not match the Id for \"" + names.get(i) + "\". Try again.");
174                                         tempOutput = 0;
175                                     } else {
176                                         rb_id = tempOutput;
177                                     }
178                                 } while(tempOutput == 0);
179                                 weightController.showMessageSecondaryDisplay("Weight \"" +
    names.get(i) + "\" and press send.");
180                                 do{
181                                     this.wait();
182                                     upperTolerance = 1.0+rkList.get(i).getTolerance();
183                                     lowerTolerance = 1.0-rkList.get(i).getTolerance();
184                                     System.out.println("rkList.get(i).getNomNetto(): " +
    rkList.get(i).getNomNetto());
185                                     System.out.println("UpperTolerance: " +
    upperTolerance);
186                                     System.out.println("LowerTolerance: " +
    lowerTolerance);
187                                     System.out.println("Weight: " + weight);
188                                     if(((weight-tarWeight) > rkList.get(i).getNomNetto() *
    upperTolerance) ||
189                                             ((weight-tarWeight) <
    rkList.get(i).getNomNetto() * lowerTolerance)){
190                                         weightController.showMessageSecondaryDisplay("Weigh
    t should be within " + rkList.get(i).getNomNetto()*lowerTolerance + "kg and " +
    rkList.get(i).getNomNetto()*upperTolerance + "kg. Try again");
191                                         toleranceFail = true;
192                                     } else {
193                                         toleranceFail = false;
194                                     }
195                                 }while(toleranceFail);
196                                 weightController.showMessageSecondaryDisplay("You are about
    to commit a productbatchcomponent. Press send to continue.");
197                                 wait();
198                                 commitPBK(opr_id, rb_id, pb_id, weight, tarWeight);
199                                 tarWeight = 0;
200                             }
201                             setPbStatus(pb_id, 2);
202                         } catch (DALException e) {
203                             e.printStackTrace();
204                         }
```

```
205                         key0 = true;
206                         allowCommands = true;
207                     } catch (InterruptedException e) {
208                         e.printStackTrace();
209                     }
210                 }
211                 break;
212             case S:
213                 total = weight - tarWeight;
214                 weightController.showMessageSecondaryDisplay("The current weight is:");
215                 weightController.showMessagePrimaryDisplay(total.toString());
216                 break;
217             case T:
218                 tara();
219                 break;
220             case DW:
221                 weightController.showMessagePrimaryDisplay(message.getMessage());
222                 break;
223             case K:
224                 handleKMessage(message);
225                 break;
226             case P111:
227                 weightController.showMessageSecondaryDisplay(message.getMessage());
228                 break;
229             }
230         }
231         else{
232             weightController.showMessageSecondaryDisplay("RM20 is currently active. Send a
    number before proceeding.");
233         }
234 
235     }
236 
237     private void handleKMessage(SocketInMessage message) {
238         switch (message.getMessage()) {
239         case "1" :
240             this.keyState = KeyState.K1;
241             break;
242         case "2" :
243             this.keyState = KeyState.K2;
244             break;
245         case "3" :
246             this.keyState = KeyState.K3;
247             break;
248         case "4" :
249             this.keyState = KeyState.K4;
250             break;
251         default:
252             socketHandler.sendMessage(new SocketOutMessage("ES"));
253             break;
254         }
255     }
256     //Listening for UI input
257     @Override
258     public void notifyKeyPress(KeyPress keyPress) {
259         switch (keyPress.getType()) {
260         case SOFTBUTTON:
261             break;
262         case TARA:
263             tara();
264             break;
265         case TEXT:
```

```
266             numbers.add(keyPress.getCharacter());
267             numbersPointer++;
268             numberMessage = "";
269             for(int i = 0; i<numbersPointer; i++){
270                 numberMessage += numbers.get(i);
271             }
272             weightController.showMessageSecondaryDisplay(numberMessage);
273             break;
274         case ZERO:
275             if (key0 == true){
276                 weight = 0.0;
277                 tarWeight = 0.0;
278                 total = 0.0;
279                 weightController.showMessagePrimaryDisplay(total.toString());
280                 weightController.showMessageSecondaryDisplay("");
281             }
282             break;
283         case C:
284             numbers = new ArrayList<Character>();
285             numbersPointer = 0;
286             System.out.println("C");
287             break;
288         case EXIT:
289             quit();
290             break;
291         case SEND:
292             synchronized (this){
293                 if (/*keyState.equals(KeyState.K4) ||*/ keyState.equals(KeyState.K3) ){
294                     socketHandler.sendMessage(new SocketOutMessage("K A 3"));
295                 }
296                 else if(keyState.equals(KeyState.K4)){
297                     socketHandler.sendMessage(new SocketOutMessage(numbers.toString()));
298                     numbersPointer = 0;
299                     tempOutput = 0;
300                     for(int i = 0; i < numbers.size(); i++){
301                         tempOutput = (tempOutput*10+(numbers.get(i)-48));       //-48 to
    convert from ASCII to integer
302                     }
303                     numbers = new ArrayList<Character>();
304                 }
305                 else{
306                     weightController.showMessageSecondaryDisplay("No command was expecting
    an input. Input discarded.");
307                     System.out.println("No command was expecting an input. Input
    discarded.");
308                 }
309                 if(!key1){
310                     notify();
311                 }
312             }
313             break;
314         }
315
316     }
317
318     @Override
319     public void notifyWeightChange(double newWeight) {
320         this.weight = newWeight; //Set the weight to be equal to the new weight
321         weightController.showMessagePrimaryDisplay(weight-tarWeight+"kg"); //Print this to
    the GUI
322
323     }
```

```
324
325     public void tara(){
326         tarWeight = 0;
327         tarWeight = weight;
328         weightController.showMessagePrimaryDisplay(total.toString());
329         System.out.println("Tarweight is: " + tarWeight);
330         synchronized (this){
331             if(key1){
332                 notify();
333                 key1 = false;
334             }
335         }
336     }
337
338     public void quit(){
339         System.exit(0);
340     }
341
342     public boolean getCommandStatus(){
343         return allowCommands;
344     }
345
346     private void commitPBK(int opr_id, int rb_id, int pb_id, double weight, double
    tarWeight) throws DALException{
347         ProduktBatchKompDAO pbkDAO = new MySQLProduktBatchKomponentDAO();
348         ProduktBatchKompDTO pbkDTO = new ProduktBatchKompDTO(pb_id, rb_id, tarWeight,
    weight, opr_id);
349         pbkDAO.createProduktBatchKomp(pbkDTO);
350     }
351
352     private void setPbStatus(int pb_id, int newStatus) throws DALException{
353         ProduktBatchDAO pbDAO = new MySQLProduktBatchDAO();
354         ProduktBatchDTO productBatch = pbDAO.getProduktBatch(pb_id);
355         productBatch.setStatus(newStatus);
356         pbDAO.updateProduktBatch(productBatch);
357     }
358
359     private void doName() throws DALException{
360         OperatoerDAO oprDAO = new MySQLOperatoerDAO();
361         List<OperatoerDTO> oprList = oprDAO.getOperatoerList();
362         weightController.showMessageSecondaryDisplay("Enter your operator ID: ");
363         do{
364             try {
365                 wait();
366             } catch (InterruptedException e) {
367                 e.printStackTrace();
368             }
369             if((tempOutput < oprList.size()) && (tempOutput > 0)){
370                 weightController.showMessagePrimaryDisplay(oprList.get(tempOutput).getOprNa
    vn());
371                 opr_id = tempOutput;
372             } else{
373                 weightController.showMessageSecondaryDisplay("Invalid user ID. Enter new
    ID.");
374                 tempOutput = 0;
375             }
376         }while(tempOutput < 1);
377     }
378
379     private List<Integer> getRaavareBatchID(int pb_id) throws DALException{
380         ViewDAO view = new MySQLViewDAO();
381         List<ViewRaavareNavneDTO> viewList = view.getRaavareNavneListPbId(pb_id);
```

```
382         List<Integer> rbID = new ArrayList<Integer>();
383         for (int i = 0; i < viewList.size(); i++){
384             System.out.println("RaavareBatch ID " + i + ": " + viewList.get(i).getRbId());
385             rbID.add(viewList.get(i).getRbId());
386         }
387         return rbID;
388     }
389 
390 
391     private void doPB() throws DALException{
392         ProduktBatchDAO pbDAO = new MySQLProduktBatchDAO();
393         List<ProduktBatchDTO> pbList = pbDAO.getProduktBatchList();
394         weightController.showMessageSecondaryDisplay("Enter the ID for the productbatch you
    want to weight");
395         do{
396             try {
397                 wait();
398             } catch (InterruptedException e) {
399                 e.printStackTrace();
400             }
401             if (tempOutput <= pbList.size() && tempOutput > 0){
402                 if(pbList.get(tempOutput-1).getStatus() == 0){
403                     pb_id = tempOutput;
404                 } else{
405                     weightController.showMessageSecondaryDisplay("The productbatch of the
    ID is finished or in progress, submit new ID.");
406                     tempOutput = 0;
407                 }
408 
409             } else {
410                 weightController.showMessageSecondaryDisplay("Invalid ID, submit new ID.");
411                 tempOutput = 0;
412             }
413         }while(tempOutput < 1);
414         System.out.println("pb_id is: " + pb_id);
415     }
416 
417     private int getRowCount(int pb_id) throws DALException{
418         ViewDAO view = new MySQLViewDAO();
419         List<ViewRaavareNavneDTO> viewList = view.getRaavareNavneListPbId(pb_id);
420         System.out.println("viewList.size: " + viewList.size());
421         return viewList.size();
422     }
423 
424     private List<String> getProductName(int pb_id) throws DALException{
425         ViewDAO view = new MySQLViewDAO();
426         List<ViewRaavareNavneDTO> viewList = view.getRaavareNavneListPbId(pb_id);
427         List<String> names = new ArrayList<String>();
428         for (int i = 0; i < viewList.size(); i++){
429             System.out.println("Raavare " + i + ": " + viewList.get(i).getRaavareNavn());
430             names.add(viewList.get(i).getRaavareNavn());
431         }
432         return names;
433     }
434 }
435 
```