```java
package game;

import game.fields.Field;

public class TradeController {

    private final String cancelButton = "Cancel";
    final private int NUMBER_OF_GROUPS = 8;
    final private int UNPAWN_RENT = 10; //TODO should probably be moved to Ownable

    // Constructor
    public TradeController(){

    }

    public void trade(Player player, Decorator decorator, PlayerController playerController) {

        Ownable[] ownedFields = new Ownable[BoardController.getFieldsbyPlayer(player).length];
        ownedFields = BoardController.getFieldsbyPlayer(player);
        String[] msg0 = new String[]{"TradeOrPawn"};
        String[] msg1 = new String[]{"Trade", "Pawn","UnPawn", "Cancel"};
        String[] msg2 = new String[]{"TradeOrCancel"};
        String[] msg3 = new String[]{"NoFieldsToTrade"};
        int choice;

        if(ownedFields.length == 0){
            decorator.showMessage(msg3);
        }
        else if(BoardController.hasAnyUnPawnedFields(player)){
            choice = decorator.getUserButtonPressed(msg0, msg1);
            if(choice == 0){
                sell(player, ownedFields, decorator, playerController);
            }
            // handles the case where the player wants to pawn a field
            if(choice == 1){
                pawn(player, ownedFields, decorator, playerController);
            }
            // if a player wants to unpawn a field
            if(choice == 2){
                unPawn(player, ownedFields, decorator, playerController);
            }
        }
        else{
            String[] msg4 = new String[]{"OnlyTradeOrUnPawn"};
            String[] msg5 = new String[]{"Trade", "UnPawn", "Cancel"};
            decorator.showMessage(msg4);
            choice = decorator.getUserButtonPressed(msg2, msg5);
```

```
51              if(choice == 0){
52                  sell(player, ownedFields, decorator, playerController);
53
54              }
55              if(choice == 1){
56                  unPawn(player, ownedFields, decorator, playerController);
57              }
58          }
59      }
60
61      private void sell(Player seller, Ownable[] ownedFields, Decorator
  decorator, PlayerController playerController){
62
63          Ownable[] sellableFields = sellAndPawnArray(ownedFields);
64          String[] ownedFieldsTitle = new String[sellableFields.length+1];
65          ownedFieldsTitle[0] = cancelButton;
66
67          // makes list of owned fields titles for the dropdown list
68          for(int i = 0; i < (sellableFields.length); i++){
69              ownedFieldsTitle[i+1] = sellableFields[i].getTitle();
70          }
71
72          String[] msg0 = new String[]{"ChooseFieldToSell"};
73          String[] msg1 = new String[]{"ChooseFieldPrice"};
74          String[] msg2 = new String[]{"NotValidPrice"};
75          String[] msg3 = new String[]{"ChoosePlayerToSellTo"};
76          String[] msg4 = new String[]{"ContinueTrade"};
77          String[] opt1 = new String[]{"Yes", "No"};
78          Player[] allPlayers = playerController.getPlayers();
79          Player[] buyingPlayers = new
  Player[playerController.getNumberOfPlayersLeft()];
80          String[] buyingPlayersName = new
  String[playerController.getNumberOfPlayersLeft()];
81          buyingPlayersName[0] = cancelButton;
82
83          int chosenFieldNumber = decorator.getUserSelection(msg0,
  ownedFieldsTitle);
84
85          if(chosenFieldNumber != 0){
86              // players choose field to sell
87              Field chosenField = sellableFields[chosenFieldNumber-1];
88
89              String userInput;
90              int sellingPrice = 0;
91              while(sellingPrice == 0){
92                  userInput = decorator.getUserString(msg1);
93                  try{
94                      sellingPrice = Integer.parseInt(userInput);
95                  } catch(NumberFormatException e){
96                      decorator.showMessage(msg2);
```

```
 97                 }
 98             }
 99
100             // buyingPlayers[0] is the cancel button therefore is j = 1
101             int j = 1;
102             // makes list of all possible buyers
103             for(int i = 0; i < allPlayers.length; i++){
104                 if(!allPlayers[i].isBroke() &&
    !allPlayers[i].equals(seller)){
105                     buyingPlayers[j] = allPlayers[i];
106                     buyingPlayersName[j] = allPlayers[i].getPlayerName();
107                     j++;
108                 }
109             }
110             int chosenPlayerNumber = decorator.getUserSelection(msg3,
    buyingPlayersName);
111             if(chosenPlayerNumber != 0){
112                 Player chosenPlayer = buyingPlayers[chosenPlayerNumber];
113
114                 while(sellingPrice >
    chosenPlayer.getAccount().getBalance()){
115                     playerController.handleInsufficientFunds(chosenPlayer,
    sellingPrice, decorator);
116                     if(sellingPrice >
    chosenPlayer.getAccount().getBalance()){
117                         if(decorator.getUserButtonPressed(msg4, opt1) ==
    1){
118                             break;
119                         }
120                     }
121                 }
122                 if(sellingPrice < chosenPlayer.getAccount().getBalance()){
123                     try {
124                         chosenPlayer.getAccount().withdraw(sellingPrice);
125                         seller.getAccount().deposit(sellingPrice);
126                         if(chosenField instanceof Ownable){
127                             ((Ownable) chosenField).setOwner(chosenPlayer);
128                         }
129                     } catch (Exception e) {
130                         System.err.println("Faaaaaame stort problem
    tradeController.sell");
131                         // while loop should make sure that this never
    happens
132                     }
133
134                     if(chosenField instanceof Ownable &&
    ((Ownable)chosenField).getOwner().equals(chosenPlayer)){
135                         // update field owner in GUI
136                         decorator.updateFieldOwner((Ownable) chosenField);
137
```

```
138                     }
139                     decorator.updatePlayer(chosenPlayer);
140                     decorator.updatePlayer(seller);
141                 }
142             }
143         }
144     }
145
146     private void pawn(Player player, Ownable[] ownedFields, Decorator
    decorator, PlayerController playerController){
147         if(ownedFields.length != 0){
148
149             Ownable[] ownedPawneble = sellAndPawnArray(ownedFields);
150             String[] ownedPawnebleTitle = new String[ownedPawneble.length +
    1];
151             ownedPawnebleTitle[0] = cancelButton;
152             // makes String[] for the GUI dropdown list
153             for(int i = 0; i < (ownedPawneble.length); i++){
154                 ownedPawnebleTitle[i+1] = ownedPawneble[i].getTitle();
155             }
156
157             // makes a dropdown list with only those fields that can be
    pawned and sets pawningField to the chosen field
158             String[] msg2 = new String[]{"ChooseFieldToPawn"};
159             int chosenFieldNumber = decorator.getUserSelection(msg2,
    ownedPawnebleTitle);
160             if(chosenFieldNumber != 0){
161                 chosenFieldNumber--;
162                 Ownable pawningField =
    ((Ownable)ownedPawneble[chosenFieldNumber]);
163                 try {
164                     pawningField.getOwner().getAccount().deposit(pawningFie
    ld.getPawnValue()); //gives the owner of that field, the fields pawn value
165                     pawningField.setPawned(true); // sets the field
    isPawned to true
166                 } catch (Exception e) {
167                     e.printStackTrace();
168                 }
169                 decorator.updatePlayer(player);
170                 decorator.updatePawned(pawningField);
171             }
172         }
173     }
174
175     private void unPawn(Player player, Ownable[] ownedFields, Decorator
    decorator, PlayerController playerController){
176         Ownable[] rawUnPawnableFields = new Street[ownedFields.length];
177
178         for(int i = 0; i < ownedFields.length; i++){
179             if(((Ownable)ownedFields[i]).isPawned()){
```

```
180                 rawUnPawnableFields[i] = ((Ownable)ownedFields[i]);
181             }
182         }
183         Ownable[] unPawnableFields =
    removeNullOwnable(rawUnPawnableFields);
184         String[] unPawnableFieldsTitle = new
    String[unPawnableFields.length+1];
185 
186         if(unPawnableFields.length > 0){
187             unPawnableFieldsTitle[0] = cancelButton;
188             for(int i = 0; i < unPawnableFields.length; i++){
189                 unPawnableFieldsTitle[i+1] =
    unPawnableFields[i].getTitle();
190             }
191             int choice = decorator.getUserSelection(new
    String[]{"ChooseFieldToUnPawn"}, unPawnableFieldsTitle);
192             if(choice != 0){
193                 choice--;
194                 int unPawnCost = (unPawnableFields[choice].getPawnValue() +
    unPawnableFields[choice].getPawnValue()/UNPAWN_RENT);
195                 boolean paid = false;
196                 while(!paid){
197                     try {
198                         player.getAccount().withdraw(unPawnCost);
199                         unPawnableFields[choice].setPawned(false);
200                         decorator.updatePawned(unPawnableFields[choice]);
201                         paid = true;
202                     } catch (Exception e) {
203                         playerController.handleInsufficientFunds(player,
    unPawnCost, decorator);
204                         e.printStackTrace();
205                     }
206                 }
207             }
208 
209         } else{
210             decorator.showMessage(new String[]{"NoFieldsToUnPawn"});
211         }
212         decorator.updatePlayer(player);
213     }
214 
215     public void buildings(Player activePlayer, PlayerController
    playerController, Decorator decorator){
216         //************************************************************
    ******
217         //      Field[] fields = BoardController.getBoard().getFields();
218         //      ((Ownable)fields[1]).setOwner(activePlayer); // for test
    must be deleted
219         //      ((Ownable)fields[3]).setOwner(activePlayer); // for test
    must be deleted
```

```
220         //      ((Ownable)fields[5]).setOwner(activePlayer);
221         //      ((Ownable)fields[6]).setOwner(activePlayer);
222         //      ((Ownable)fields[8]).setOwner(activePlayer);
223         //      ((Ownable)fields[9]).setOwner(activePlayer);
224         //      ((Ownable)fields[11]).setOwner(activePlayer);
225         //      ((Ownable)fields[13]).setOwner(activePlayer);
226         //      ((Ownable)fields[14]).setOwner(activePlayer);
227         //      ((Ownable)fields[15]).setOwner(activePlayer);
228         //      ((Ownable)fields[16]).setOwner(activePlayer);
229         //      ((Ownable)fields[18]).setOwner(activePlayer);
230         //      ((Ownable)fields[19]).setOwner(activePlayer);
231         //      ((Ownable)fields[21]).setOwner(activePlayer);
232         //      ((Ownable)fields[23]).setOwner(activePlayer);
233         //      ((Ownable)fields[25]).setOwner(activePlayer);
234         //      ((Ownable)fields[26]).setOwner(activePlayer);
235         //      ((Ownable)fields[28]).setOwner(activePlayer);
236         //      ((Ownable)fields[29]).setOwner(activePlayer);
237         //      ((Ownable)fields[31]).setOwner(activePlayer);
238         //      ((Ownable)fields[32]).setOwner(activePlayer);
239         //      ((Ownable)fields[34]).setOwner(activePlayer);
240         //      ((Ownable)fields[37]).setOwner(activePlayer);
241         //      ((Ownable)fields[39]).setOwner(activePlayer);
242         //
243         //      decorator.updateFieldOwner(((Ownable)fields[1]));
244         //      decorator.updateFieldOwner(((Ownable)fields[3]));
245         //      decorator.updateFieldOwner(((Ownable)fields[5]));
246         //      decorator.updateFieldOwner(((Ownable)fields[6]));
247         //      decorator.updateFieldOwner(((Ownable)fields[8]));
248         //      decorator.updateFieldOwner(((Ownable)fields[9]));
249         //      decorator.updateFieldOwner(((Ownable)fields[11]));
250         //      decorator.updateFieldOwner(((Ownable)fields[13]));
251         //      decorator.updateFieldOwner(((Ownable)fields[14]));
252         //      decorator.updateFieldOwner(((Ownable)fields[15]));
253         //      decorator.updateFieldOwner(((Ownable)fields[16]));
254         //      decorator.updateFieldOwner(((Ownable)fields[18]));
255         //      decorator.updateFieldOwner(((Ownable)fields[19]));
256         //      decorator.updateFieldOwner(((Ownable)fields[21]));
257         //      decorator.updateFieldOwner(((Ownable)fields[23]));
258         //      decorator.updateFieldOwner(((Ownable)fields[25]));
259         //      decorator.updateFieldOwner(((Ownable)fields[26]));
260         //      decorator.updateFieldOwner(((Ownable)fields[28]));
261         //      decorator.updateFieldOwner(((Ownable)fields[29]));
262         //      decorator.updateFieldOwner(((Ownable)fields[31]));
263         //      decorator.updateFieldOwner(((Ownable)fields[32]));
264         //      decorator.updateFieldOwner(((Ownable)fields[34]));
265         //      decorator.updateFieldOwner(((Ownable)fields[37]));
266         //      decorator.updateFieldOwner(((Ownable)fields[39]));
267
268         //*************************************************************
    ******
```

```
269
270
271         Field[] playersField =
    BoardController.getFieldsbyPlayer(activePlayer);
272         String[] msg0 = new String[]{"NoFieldsToBuildHousesOn"};
273         String[] buyBuildingFieldsTitle = null;
274         String[] sellBuildingFieldsTitle = null;
275         Street[] buildableFields = getBuildableFields(playersField);
276
277
278         if(buildableFields.length == 0){
279             decorator.showMessage(msg0);
280         }
281
282         else{
283             Street[] buyBuildingFields =
    getBuyBuildingFields(buildableFields);
284             Street[] sellBuildingFields =
    getSellBuildingFields(buildableFields);
285
286             if(buyBuildingFields.length !=0){
287                 buyBuildingFieldsTitle = new
    String[buyBuildingFields.length+1];
288                 buyBuildingFieldsTitle[0] = cancelButton;
289
290                 for(int i = 0; i < buyBuildingFields.length; i++){
291                     buyBuildingFieldsTitle[i+1] =
    buyBuildingFields[i].getTitle();
292                 }
293             }
294             if(sellBuildingFields.length != 0){
295                 sellBuildingFieldsTitle = new
    String[sellBuildingFields.length+1];
296                 sellBuildingFieldsTitle[0] = cancelButton;
297
298                 for(int i = 0; i < sellBuildingFields.length; i++){
299                     sellBuildingFieldsTitle[i+1] =
    sellBuildingFields[i].getTitle();
300                 }
301             }
302
303             String[] msg2 = new String[]{"BuyOrSellBuilding"};
304             String[] buttons0 = new String[]{"Buy", "Sell", "Cancel"};
305             int sellORbuy = decorator.getUserButtonPressed(msg2, buttons0);
306
307             if(sellORbuy != 2){
308                 boolean regretBuying = false;
309                 // if player wants to buy
310                 if(sellORbuy == 0){
311                     String[] msg3 = new String[]{"ChooseFieldToBuildOn"};
```

```
312                     if(buyBuildingFields.length == 0){
313                         String[] msg4 = new String[]{"NoFieldsToBuildOn"};
314                         decorator.showMessage(msg4);
315                         return;
316                     }
317                     else{
318                         int buyOnField = (decorator.getUserSelection(msg3,
    buyBuildingFieldsTitle));
319                         if(buyOnField != 0){
320                             buyOnField--;
321                             int buildingPrice =
    buyBuildingFields[buyOnField].getBuildingPrice();
322 
323                             if(activePlayer.getAccount().getBalance() <
    buildingPrice){
324                                 regretBuying =
    !playerController.handleInsufficientFunds(activePlayer, buildingPrice,
    decorator);
325                             }
326                             if(!regretBuying){
327                                 try {
328                                     activePlayer.getAccount().withdraw(buyB
    uildingFields[buyOnField].getBuildingPrice());
329                                     buyBuildingFields[buyOnField].addBuildi
    ng();
330                                     decorator.updateHouses(buyBuildingField
    s[buyOnField]);
331                                     decorator.updatePlayer(activePlayer);
332                                 } catch (Exception e) {
333                                     // should not get this far because of
    the previous if statement
334                                     e.printStackTrace();
335                                 }
336                             }
337                         }
338                     }
339                 }
340                 if(sellORbuy == 1){
341                     String[] msg5 = new
    String[]{"ChooseFieldToSellBuilding"};
342                     if(sellBuildingFields.length == 0){
343                         String[] msg4 = new
    String[]{"NoFieldsToSellBuilding"};
344                         decorator.showMessage(msg4);
345                         return;
346                     }
347 
348                     else{
349                         int sellOnField = (decorator.getUserSelection(msg5,
    sellBuildingFieldsTitle));
```

```
350                         if(sellOnField != 0){
351                             sellOnField--;
352 
353                             if(!regretBuying){
354                                 try {
355                                     activePlayer.getAccount().deposit(sellB
    uildingFields[sellOnField].getBuildingSellValue());
356                                     sellBuildingFields[sellOnField].removeB
    uilding();
357                                     decorator.updateHouses(sellBuildingFiel
    ds[sellOnField]);
358                                     decorator.updatePlayer(activePlayer);
359                                 } catch (Exception e) {
360                                     // only if you have more money than int
    can handle
361                                     e.printStackTrace();
362                                 }
363                             }
364                         }
365                     }
366                 }
367             }
368         }
369     }
370 
371 
372     private Street[] getBuyBuildingFields(Street[] buildableFields){
373 
374         Street[] buyHouseFields = new Street[buildableFields.length];
375         for(int i = 0; i < buildableFields.length; i++){
376             boolean legitField = true;
377             if(buildableFields[i].getBuildings() <
    Street.MAX_NUMBER_OF_BUILDINGS){
378                 for(int j = 0; j < buildableFields.length; j++){
379                     if(buildableFields[i].getGroup().equals(buildableFields[
    j].getGroup()) && !buildableFields[i].equals(buildableFields[j])){
380 
381                         if(buildableFields[i].getBuildings() >
    buildableFields[j].getBuildings()){
382                             legitField = false;
383                             break;
384                         }
385                     }
386                 }
387             }
388             else {
389                 System.out.println("LegitField = false");
390                 legitField = false;
391             }
392             if(legitField){
```

```
393                 buyHouseFields[i] = buildableFields[i];
394             }
395         }
396         return removeNullStreets(buyHouseFields);
397     }
398 
399     private Street[] getSellBuildingFields(Street[] buildableFields){
400 
401         Street[] sellBuildingFields = new Street[buildableFields.length];
402         for(int i = 0; i < buildableFields.length; i++){
403             boolean legitField = true;
404             if(buildableFields[i].getBuildings() > 0){
405                 for(int j = 0; j < buildableFields.length; j++){
406                     if(buildableFields[i].getGroup().equals(buildableFields[
    j].getGroup()) && !buildableFields[i].equals(buildableFields[j])){
407 
408                         if(buildableFields[i].getBuildings() <
    buildableFields[j].getBuildings()){
409                             legitField = false;
410                             break;
411                         }
412                     }
413                 }
414             }
415             else {
416                 legitField = false;
417             }
418             if(legitField){
419                 sellBuildingFields[i] = buildableFields[i];
420             }
421         }
422 
423         return removeNullStreets(sellBuildingFields);
424     }
425 
426     private Street[] removeNullStreets(Street[] list){
427         int nullCounter = 0;
428         for(int i = 0; i < list.length; i++){
429             if(list[i] == null){
430                 nullCounter++;
431             }
432         }
433         Street[] newArray = new Street[list.length - nullCounter];
434         int j = 0;
435         System.out.println("nullcounter: " + nullCounter);
436         for(int i = 0; i < list.length; i++){
437             if(list[i] != null){
438                 newArray[j] = list[i];
439                 System.out.println("newArray: " + newArray[j].getTitle());
440                 j++;
```

```
441             }
442         }
443         return newArray;
444     }
445 
446     private Ownable[] sellAndPawnArray(Ownable[] ownedFields){
447 
448         Ownable[] rawSellAndPawnArray = new Ownable[ownedFields.length];
449         for(int i = 0; i < ownedFields.length; i++){
450             boolean addStreet = true;
451             if(ownedFields[i] instanceof Street){
452                 if(((Street)ownedFields[i]).getBuildings() == 0){
453                     for(int j = 0; j < ownedFields.length; j++){
454                         if(ownedFields[j] instanceof Street){
455                             if(((Street)ownedFields[i]).getGroup() ==
    ((Street)ownedFields[j]).getGroup()){
456                                 if(((Street)ownedFields[j]).getBuildings()
    > 0){
457                                     addStreet = false;
458                                     break;
459                                 }
460                             }
461                         }
462                     }
463                 }
464                 else{
465                     addStreet = false;
466                 }
467             }
468             if(addStreet){
469                 rawSellAndPawnArray[i] = ownedFields[i];
470             }
471         }
472         return removeNullOwnable(rawSellAndPawnArray);
473     }
474     private Ownable[] removeNullOwnable(Ownable[] list){
475         int nullCounter = 0;
476 
477         for(int i = 0; i < list.length; i++){
478 
479             if(list[i] == null){
480                 nullCounter++;
481             }
482         }
483         Ownable[] newArray = new Ownable[list.length - nullCounter];
484         int j = 0;
485         System.out.println("nullcounter: " + nullCounter);
486         for(int i = 0; i < list.length; i++){
487             if(list[i] != null){
488                 newArray[j] = list[i];
```

```
489                 System.out.println("newArray: " + newArray[j].getTitle());
490                 j++;
491             }
492         }
493         return newArray;
494     }
495     private Street[] getBuildableFields(Field[] playersField){
496 
497         Field[] allFields = BoardController.getBoard().getFields();
498         Group[] availableGroups = new Group[NUMBER_OF_GROUPS];
499         int numberOfAvailableFields = 0;
500         int numberOfAvailableGroup = 0;
501         for(int i = 0; i < playersField.length; i++){
502             if(playersField[i] instanceof Street){
503                 boolean allFieldsInGroup = true;
504                 // checks if owner has all fields matching playersField[i] group
505                 int j = 0;
506                 for(; j < allFields.length; j++){
507                     System.out.println("checker tilgængelige felter at bygge huse på"); // for test
508                     // should always get in this statement. Only check for if the field is instance of the class Street
509                     if(allFields[j] instanceof Street){
510                         // if loop avoids this statement, then all fields in playersField[i].getGroup is owned by same player
511                         if(((Street)playersField[i]).getGroup().equals(((Street)allFields[j]).getGroup()) &&
512                                 !((Street)playersField[i]).getOwner().equals(((Street)allFields[j]).getOwner())){
513                             allFieldsInGroup = false;
514                             break;
515                         }
516                     }
517                 }
518                 // if enters this statement it means that the player is the owner of all fields in one group
519                 if(allFieldsInGroup){
520                     availableGroups[numberOfAvailableGroup] = ((Street)playersField[i]).getGroup();
521 
522                     if(numberOfAvailableGroup > 0){
523                         if(availableGroups[numberOfAvailableGroup].equals(availableGroups[numberOfAvailableGroup-1])){
524                             numberOfAvailableGroup--;
525                         }
526                     }
527                     numberOfAvailableGroup++;
528                     numberOfAvailableFields++;
```

```
529                 }
530             }
531         }
532         Street[] buildableFields = new Street[numberOfAvailableFields];
533 
534         // loop to do array of fields that can have houses built on
535         int k = 0; // starts with 1 because of the cancel button
536         for(int j = 0; j < availableGroups.length; j++){
537 
538             for(int i = 0; i < playersField.length; i++){
539                 if(playersField[i] instanceof Street){
540                     if(((Street)playersField[i]).getGroup().equals(availabl
    eGroups[j])){
541                         buildableFields[k] = ((Street)playersField[i]);
542                         k++;
543                         // have to get rid of this statement should not be
    necessary
544                         if(numberOfAvailableFields == k){
545                             break;
546                         }
547                     }
548                 }
549             }
550             if(numberOfAvailableFields == k){
551                 break;
552             }
553         }
554         return buildableFields;
555     }
556 
557     // TODO future feature to be done
558     //  public void auktion(Ownable auktionField, Player activePlayer,
    PlayerController playerController){
559     //
560     //      Player[] bidders = new
    Player[playerController.getNumberOfPlayersLeft()];
561     //      Player highestBidder;
562     //      Player currentBidder;
563     //      int j = 0;
564     //      for(int i = 0; i < playerController.getPlayers().length; i++){
565     //          if(!playerController.getPlayers()[i].getIsBroke()){
566     //              bidders[j] = playerController.getPlayers()[i];
567     //          }
568     //          if(playerController.getPlayers()[i].equals(activePlayer)){
569     //              if(i == 0){
570     //
571     //              }
572     //              currentBidder = playerController.getPlayers()[i];
573     //          }
574     //      }
```

```
575     //      for(Player p: b)
576     //      currentBidder = activePlayer;
577     //          do{
578     //
579     //
580     //          }
581     //      while(highestBidder.equals(currentBidder));
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599 }
600
```