```java
package game.cards;

import game.BoardController;

public class CardController {

    private Card[] deck; // array of card objects
    private static final int NUMBER_OF_CARDS = 44; //constant number of cards
    private int NUMBER_OF_GET_OUT_OF_JAIL_CARDS = 2;
    private Card[] getOutOfJailCards;
    private int payAmount;

    // constructor fills deck of cards
    public CardController() {
        deck = new Card[NUMBER_OF_CARDS];
        deck[0] = new GetOutOfJailCard("theKingsBirthday");
        deck[1] = new GetOutOfJailCard("theKingsBirthday2");
        deck[2] = new PayCard("payParkingTicket", 200);
        deck[3] = new PayCard("payBeer", 200);
        deck[4] = new PayCard("payCar", 3000);
        deck[5] = new PayCard("payCar2", 3000);
        deck[6] = new PayCard("payDentist", 2000);
        deck[7] = new PayCard("payCarInsurance", 1000);
        deck[8] = new PayCard("payToll", 200);
        deck[9] = new PayCard("payCarWash", 300);
        deck[10] = new PayCard("payNewTires", 1000);
        deck[11] = new PayCard("paySpeedingTicket", 1000);
        deck[12] = new PayForBuildingsCard("payPropertyTax", 800, 2300);
        deck[13] = new PayForBuildingsCard("payOilPrices", 500, 2000);
        deck[14] = new MoveToCard("moveToRådhuspladsen", 40); //TODO Når vi har tid, skal alle moveTo rettes til at flytte til det bestemte felt, hellere end feltnummer.
        deck[15] = new MoveToNearestCard("moveToNearestFerry", 1);
        deck[16] = new MoveToCard("moveToVimmelSkaftet", 33);
        deck[17] = new MoveToNearestCard("moveToNearestShipping", 2);
        deck[18] = new MoveToCard("moveToMolsLinien", 16);
        deck[19] = new GoToJailCard("moveToJail1");
        deck[20] = new MoveToCard("moveToFrederiksberg", 12);
        deck[21] = new GoToJailCard("moveToJail2");
        deck[22] = new MoveToCard("moveToStrandvejen", 20);
        deck[23] = new MoveToCard("moveToGrønningen", 25);
        deck[24] = new MoveFieldsCard("moveForward", 3);
        deck[25] = new MoveToCard("moveToStart", 1);
        deck[26] = new MoveToCard("moveToStart2", 1);
        deck[27] = new MoveFieldsCard("moveBackwards", -3);
        deck[28] = new MoveFieldsCard("moveBackwards2", -3);
        deck[29] = new ReceiveCard("receiveLottery", 500);
        deck[30] = new ReceiveCard("receiveLottery2", 500);
        deck[31] = new ReceiveFromPlayersCard("receiveBirthday", 200);
```

```
57          deck[32] = new ReceiveCard("receiveDivivendsStocks", 1000);
58          deck[33] = new ReceiveCard("receiveDividends", 1000);
59          deck[34] = new ReceiveCard("receiveDividends2", 1000);
60          deck[35] = new ReceiveCard("receivePaymentTaxes", 3000);
61          deck[36] = new ReceiveFromPlayersCard("receiveParty", 500);
62          deck[37] = new ReceiveCard("receiveGageRaise", 1000);
63          deck[38] = new ReceiveCard("receiveFromBets", 1000);
64          deck[39] = new ReceiveCard("receiveFromFurniture", 1000);
65          deck[40] = new ReceiveCard("receiveFromBond", 1000);
66          deck[41] = new ReceiveCard("receiveFromBond2", 1000);
67          deck[42] = new ReceiveFromPlayersCard("receiveFamParty", 500);
68          deck[43] = new ReceiveCard("receiveNyttehaven", 200);
69          getOutOfJailCards = new Card[]{null, null};
70          //
71          Shuffler.Shuffle(deck);
72      }
73      public void drawCard(Player player, Decorator decorator,
    PlayerController playerController, BoardController boardController){
74          Card card = getNextCard();
75          decorator.showChanceCard(new String[] {card.getCardDescription()});
76          decorator.showMessage(new String[] {"LandedOnChance"});
77
78          if (card instanceof ReceiveCard){
79              try {
80                  player.getAccount().deposit(((ReceiveCard)
    card).getAmount());
81              } catch (Exception e) {
82                  System.err.println("deposit error - drawCard");
83                  e.printStackTrace();
84              }
85          }
86          if (card instanceof ReceiveFromPlayersCard){
87
88              for(Player payingPlayer : playerController.getPlayers()){
89                  int cardAmount = ((ReceiveFromPlayersCard)
    card).getAmount();
90                  //all players except the one that shall receive money
91                  if (!player.equals(payingPlayer)) {
92                      boolean amountPaid = false;
93                      while(!amountPaid){
94                          try {
95                              payingPlayer.getAccount().withdraw(cardAmount);
96                              player.getAccount().deposit(cardAmount);
97                              decorator.updatePlayer(payingPlayer);
98                              decorator.updatePlayer(player);
99                              amountPaid = true;
100                         } catch (Exception e) {
101                             // as long the player has unpawned fields, he
    is forced to pawn fields until he can afford to pay
102                             if(BoardController.hasAnyUnPawnedFields(payingP
```

```
layer)){
                                String[] msg0 = new
String[]{"YouHaveToPawn"};
                                decorator.showMessage(msg0);
                                playerController.handleInsufficientFunds(pa
yingPlayer, cardAmount, decorator);
                            }
                            // when his only choice is to sell off his
fields, he is being offered to go bankrupt
                            else{
                                String[] msg1 = new
String[]{"TradeOrBankrupt"};
                                String[] opt1 = new String[]{"Trade",
"Bankrupt"};
                                int choice =
decorator.getUserButtonPressed(msg1, opt1);
                                if(choice == 1){
                                    playerController.hostileTakeOver(player,
payingPlayer);
                                    break;
                                }
                                else{
                                    playerController.handleInsufficientFund
s(payingPlayer, cardAmount, decorator);
                                }
                            }
                            e.printStackTrace();
                        }
                    }
                }
            }
        }
        if (card instanceof GetOutOfJailCard){
            player.setNumberOfJailCards(player.getNumberOfJailCards() + 1);
        }
        if (card instanceof GoToJailCard){
            player.setInJail(1);
            player.setCurrentFieldNumber(11);
            decorator.updatePlayer(player);
        }
        if(card instanceof PayForBuildingsCard){
            try {
                payAmount =
player.getAccount().withdraw(getBuildingExpenses(player,
((PayForBuildingsCard) card).getHousePrice(), ((PayForBuildingsCard)
card).getHotelPrice(), boardController));
            } catch (Exception e) {
                playerController.handleInsufficientFunds(player, payAmount,
decorator);
                e.printStackTrace();
```

```
140             }
141         }
142         if(card instanceof PayCard){
143             try {
144                 player.getAccount().withdraw(((PayCard) card).getAmount());
145             } catch (Exception e) {
146                 playerController.handleInsufficientFunds(player, ((PayCard)
    card).getAmount(), decorator);
147                 e.printStackTrace();
148             }
149         }
150         if(card instanceof MoveFieldsCard){
151             playerController.move(player, ((MoveFieldsCard)
    card).getNumberOfFields(), decorator, playerController, this,
    boardController);;
152         }
153         // TODO is broken
154         if(card instanceof MoveToNearestCard){
155             Field[] allFields = BoardController.getBoard().getFields();
156             int playerField = player.getCurrentFieldNumber();
157             int moveTo = 0;
158             for(int i = 0; i < allFields.length; i++){
159                 if(allFields[i] instanceof game.fields.Shipping &&
    playerField < i){
160                     moveTo = i+1;
161                     break;
162 
163                 }
164             }
165             if(moveTo == 0){
166                 for(int i = 0; i < allFields.length; i++){
167                     if(allFields[i] instanceof game.fields.Shipping){
168                         moveTo = i+1;
169                         break;
170                     }
171                 }
172             }
173             playerController.moveTo(player, moveTo, boardController,
    decorator, this, ((MoveToNearestCard) card).getRentModifier());
174         }
175         if(card instanceof MoveToCard){
176             playerController.moveTo(player, ((MoveToCard)
    card).getFieldNumber(), boardController, decorator, this);
177         }
178         decorator.updatePlayer(player);
179     }
180 
181 
182 
183     //Flytter alle kort et til venstre i deck
```

```
184     public Card getNextCard() {
185         Card currentCard = deck[0];
186         if (currentCard instanceof GetOutOfJailCard)
187         {
188             moveCards(deck, NUMBER_OF_CARDS);
189             if (getOutOfJailCards[1] != null)
190             {
191                 moveCards(getOutOfJailCards,
    NUMBER_OF_GET_OUT_OF_JAIL_CARDS);
192                 getOutOfJailCards[1] = currentCard;
193             } else {
194                 getOutOfJailCards[0] = currentCard;
195             }
196         }
197         else {
198             moveCards(deck, NUMBER_OF_CARDS);
199             deck[NUMBER_OF_CARDS-1] = currentCard;
200         }
201         return currentCard;
202     }
203 
204     public void useJailCard(Player player){
205         // TODO smide fængselskortet ind i deck igen
206         player.setInJail(0);
207         if (getOutOfJailCards[0] != null){
208             Card currentCard = getOutOfJailCards[0];
209             getOutOfJailCards[0] = null;
210             deck[deck.length - 1] = currentCard;
211         } else if(getOutOfJailCards[1] != null){
212             Card currentCard = getOutOfJailCards[1];
213             getOutOfJailCards[1] = null;
214             deck[deck.length] = currentCard;
215 
216         } else{
217             System.err.println("Player has no jailcard!");
218         }
219         player.setNumberOfJailCards(player.getNumberOfJailCards() - 1);
220     }
221 
222     private void moveCards(Card[] card, int numberOfCards){
223         for(int i = 1; i < numberOfCards; i++){
224             card[i-1] = card[i];
225         }
226     }
227 
228     private int getBuildingExpenses(Player player, int houseExpense, int
    hotelExpense, BoardController boardController){
229         int buildingExpenses = 0;
230         Field[] playersFields = BoardController.getFieldsbyPlayer(player);
231         for(Field field: playersFields){
```

```
232             if(field instanceof Street){
233                 int fieldBuildings = ((Street)field).getBuildings();
234                 if (fieldBuildings == 5){
235                     buildingExpenses += hotelExpense;
236                 }else {
237                     buildingExpenses += fieldBuildings*houseExpense;
238                 }
239             }
240 
241         }
242 
243         return buildingExpenses;
244     }
245     public static void main(String [] args){
246         CardController testCardController = new CardController();
247         BoardController testBoardController = new BoardController(40);
248         Player[] players = new Player[3];
249         PlayerController testPlayerController = new
    PlayerController(players, 30000, 40);
250         Decorator testDecorator = new Decorator("danish");
251         players[0] = new Player("BootStrap", 30000);
252         players[1] = new Player("Sparrow", 30000);
253         players[2] = new Player("Barbossa", 30000);
254 
255         testDecorator.setupGUI(testBoardController.getBoard().getFields(),
    players);
256         for(int i = 0; i < 40; i++){
257         testDecorator.updatePlayer(players[0]);
258         testDecorator.updatePlayer(players[1]);
259         testDecorator.updatePlayer(players[2]);
260         testCardController.drawCard(players[0] , testDecorator,
    testPlayerController, testBoardController);
261         }
262 
263     }
264 }
265 
```