```java
package game;

import game.Account.IllegalAmountException;

public class PlayerController {
    private Player[] players;
    private final int PASS_START_MONEY = 4000;
    private final int TOTAL_FIELDS;
    private TradeController tradeController = new TradeController();

    public PlayerController(Player[] players, int sTART_MONEY, int totalFields) {
        super();
        this.players = players;
        this.TOTAL_FIELDS = totalFields;
    }
    // Constructor
    public PlayerController(int startMoney, int totalfields) {
        this.TOTAL_FIELDS = totalfields;
    }
    //Ordinary commands
    public int getTotalAssets(BoardController boardController, Player player) {
        int totalAssets =0;
        totalAssets += player.getAccount().getBalance();
        System.out.println("totalAssets:" +totalAssets);
        for (Field field : BoardController.getFieldsbyPlayer(player)){
            totalAssets += ((Ownable)field).getPrice();
            System.out.println(totalAssets);
            if (field instanceof Street){
                totalAssets += ((Street)field).getBuildingBuyValue();
            }
        }

        return totalAssets;
    }


    //Getters and Setters
    public Player[] getPlayers() {
        return players;
    }
    public void setPlayers(Player[] players) {
        this.players = players;
    }

    // Takes a player and a sum and moves the player that sum forwards
    public void move(Player player, int sum,Decorator decorator,
            PlayerController playerController, CardController cardController, BoardController boardController) {
```

```java
        int currentField = player.getCurrentFieldNumber();
        if((currentField + sum) > TOTAL_FIELDS){
            passStart(player);
        }
        if((currentField + sum >= 1)){
            player.setCurrentFieldNumber(((currentField - 1 + sum)
                    % TOTAL_FIELDS)+1);
        } else {
            player.setCurrentFieldNumber(currentField + sum + 40);
        }
        decorator.updatePlayer(player);
        boardController.landOnField(player, player.getCurrentFieldNumber(), decorator, this, cardController);
    }
    // moves a player to a specific field
    public void moveTo(Player player, int fieldInt, BoardController boardController, Decorator decorator, CardController cardController, int rentModifier){
        if (player.getCurrentFieldNumber() > fieldInt){
            passStart(player);
        }
        player.setCurrentFieldNumber(fieldInt);
        decorator.updatePlayer(player);
        boardController.landOnField(player, fieldInt, decorator, this, cardController, rentModifier);
    }
    public void moveTo(Player player, int fieldInt, BoardController boardController, Decorator decorator, CardController cardController){
        moveTo(player, fieldInt, boardController, decorator, cardController, 1);
    }

    public void moveToJail(Player player, int jailField){
        player.setInJail(1);
        player.setCurrentFieldNumber(jailField);
        player.setTwoOfAKindCount(0);
    }

    // Adds START_MONEY to a players account
    public void passStart(Player player){
        try {
            player.getAccount().deposit(PASS_START_MONEY);
        } catch (Exception e) {
            System.err.println("fail in passStart");
            e.printStackTrace();
        }
    }
    public boolean payDebt(Player debitor, Player creditor, Decorator decorator, String[] msg, int debt) {
        boolean debtPayed = false;
        while (!debtPayed){
            decorator.showMessage(msg);
            try {
                debitor.getAccount().withdraw(debt);
                if (creditor != null) {
                    creditor.getAccount().deposit(debt); //Should be handled in a seperate statement - future
                }
                debtPayed = true;
            } catch (InsufficientFundsException e) {
                //insufficient funds to pay player
                System.out.println("Insufficient funds for transaction");
                //If player has any unpawned fields, he is forced to pawn them
                if(BoardController.hasAnyUnPawnedFields(debitor)){
                    String[] msg1 = new String[]{"YouMustAtLeastPawnAllFields"};
                    decorator.showMessage(msg1);
                    handleInsufficientFunds(debitor, debt, decorator);
                } else {
                    //Else he can choose between trying to raise money
                    String[] messageString = new String[] {"TradeOrGoBroke"};
                    String[] buttons = new String[] {"Trade","Bankrupt"};
                    int selection = decorator.getUserButtonPressed(messageString, buttons);
                    if (selection == 0){
                        handleInsufficientFunds(debitor, debt, decorator);
                    } else {
                        hostileTakeOver(creditor, debitor);
                        break;
                    }
                }
            } catch (IllegalAmountException e) {
                System.err.println("IllegalAmount LandOnOwnable");
                e.printStackTrace();
                break;
            }
        }
        return debtPayed;
    }
    public boolean handleInsufficientFunds(Player player, int amount, Decorator decorator) {
        String[] msg1 = new String[]{"YouMustPawnTrade"};
        String[] msg2 = new String[]{"TradeOrNot"};
        String[] opt0 = new String[]{"Trade","Buildings", "Cancel"};
        while(player.getAccount().getBalance() < amount){
            String[] msg0 = new String[]{"NotEnoughMoneyYouNeed", Integer.toString(amount)};
            decorator.showMessage(StringTools.add(msg0, msg1));
            int choice = decorator.getUserButtonPressed(msg2, opt0);
            if(choice == 0){
                tradeController.trade(player, decorator, this);
            }
            if(choice == 1){
                tradeController.buildings(player, this, decorator);
            }
            else{
                break;
            }
            decorator.updatePlayer(player);
        }
        return player.getAccount().getBalance() > amount;
    }

    public int getNumberOfPlayersLeft(){
        int numberOfPlayersLeft = 0;
        for(Player p: players){
            if(!p.isBroke()){
                numberOfPlayersLeft++;
            }
        }
        return numberOfPlayersLeft;
    }

    // some kind of error
    public void hostileTakeOver(Player kreditor, Player debitor){
        Field[] fieldsToReset = BoardController.getFieldsbyPlayer(debitor);
        if(kreditor == null){
            //TODO make auktion available then this method can be built properly
            for(Field f: fieldsToReset){
                if(f instanceof Ownable){
                    ((Ownable) f).setOwner(null);
                }

            }
        }
        else{
            for(Field f: fieldsToReset){
                if(f instanceof Ownable){
                    ((Ownable) f).setOwner(kreditor);
                }
            }
        }
        try {
            debitor.getAccount().setBalance(0);
        } catch (Exception e) {
            e.printStackTrace();
        }
        debitor.setIsBroke(true);
    }
    public void trade(Player activePlayer, Decorator decorator) {
        tradeController.trade(activePlayer, decorator, this);

    }
    public void buyHouse(Player activePlayer, Decorator decorator) {
        tradeController.buildings(activePlayer, this, decorator);

    }
    public void playerSetup(GameController gameController, Decorator decorator, int startBalance) {
        String[] options = {"2 Players","3 Players","4 Players","5 Players","6 Players"};
        int numberOfPlayers = (decorator.getUserSelection(new String[] {"SelectNumberOfPlayers"}, options))+2;
        this.players = new Player[numberOfPlayers];
        String PlayerName = null;
        for (int i = 0;i<players.length;i++){
            PlayerName = nameCheck(gameController, decorator, players, PlayerName, i);
            players[i] = new Player(PlayerName, startBalance);
        }
    }
    private String nameCheck(GameController gameController, Decorator decorator, Player[] players, String PlayerName, int i) {
        boolean sameName = true;
        while (sameName  == true){
            PlayerName = decorator.getUserString(new String[] {"EnterPlayerName", "Player",String.valueOf(i+1)});
            sameName = false;
            for (int j = 0;j<i;j++){
                if (PlayerName.toLowerCase().equals(players[j].getPlayerName().toLowerCase())){
                    decorator.showMessage(new String[] {"NameTaken"});
                    sameName= true;
                    break;
                } else {
                    sameName = false;
                }
            }
        }
        return PlayerName;
    }


    //  public static void main(String[] args){
    //      //Test - move()
    //      Player p = new Player("ChristiansMor", 6000);
    //      PlayerController testController = new PlayerController(30000,
    40);
    //      testController.move(p, 4, decorator,
    //              playerController,  cardController,  boardController);
    //      System.out.println(p.getCurrentFieldNumber() + "\t" +
            p.getAccount());
    //
    //      //Test - moveTo()
    ////        testController.moveTo(p, 0);
    ////        System.out.println(p.getCurrentFieldNumber() + "\t" +
            p.getAccount());
    //
    //      //Test - moveToJail()
    //      testController.moveToJail(p, 3);
    //      System.out.println(p.getCurrentFieldNumber() + "\t" + p + "\t"
            + p.getInJail());
    //  }

            }

```