GameController.java

```java
package controller;

import java.awt.Color;

public class GameController {
    protected Board board; //An instance of the Board class
    protected Player[] players; //An array of Players

    public GameController(){
        board = new Board();
        GUIController.initializeBoard(board);
    }

    public void resetGame(int playerAmount, int balance){
        players = new Player[playerAmount];
        Color color = null;

        for (int i = 0; i < players.length; i++){

            switch (i){
            case 0: color = Color.red; break;
            case 1: color = Color.green; break;
            case 2: color = Color.yellow; break;
            case 3: color = Color.blue; break;
            case 4: color = Color.white; break;
            case 5: color = Color.black; break;
            default: System.exit(1);
            }

            players[i] = new Player(i+1,Messages.getGeneralMessages()[10]+(i+1),new Piece(color), new Account(balance));

            GUIController.addPlayer(players[i].getName(),players[i].getAccount().getBalance(),players[i].getPiece().getColor());

            for (Field j : board.getFields()){
                if (j instanceof Ownable){
                    //remove owner of field
                    ((Ownable) j).setOwner(null);
                    GUIController.removeFieldOwner(j.getId());

                    //remove houses and hotels
                    if (j instanceof Street){
                        ((Street) j).setHousesOwned(0);
                        GUIController.setHouses(j.getId(),0);
                    }
                }

            }

        }
    }

    public void playGame(){
        boolean winnerFound = false;
        Player currentPlayer;

        //first player is player 1
        currentPlayer = players[0];

        while (winnerFound == false){

            TurnController turn = new TurnController(currentPlayer,board);
            turn.playTurn();

            if (currentPlayer.getAccount().getBalance() < 0){
                removePlayer(currentPlayer);
            }

            if (players.length == 1){
                winnerFound = true;
                GUIController.showMessage((currentPlayer.getName()) + Messages.getGeneralMessages()[15]);
            }
            else{
                currentPlayer = defineNextPlayer(currentPlayer);
            }
```

```java
        }
    }

    protected void removePlayer(Player player){
        Player[] temp;
        temp = players;

        players = new Player[temp.length-1];

        int playerCount = 0;
        for (int i = 0; i<temp.length;i++){
            if (temp[i] != player){
                players[playerCount] = temp[i];
                playerCount++;
            }
        }

        GUIController.removeAllCars(player.getName());

        //puts players owned fields back on sale
        for (Ownable i : player.getAccount().getOwnedFields()){
            if(i != null){
                GUIController.removeFieldOwner(i.getId());
                i.setOwner(null);
                if (i instanceof Street){
                    ((Street) i).setHousesOwned(0);
                    GUIController.setHouses(i.getId(),0);
                }
            }
        }

    }

    protected Player defineNextPlayer(Player currentPlayer){
        Player nextPlayer;

        //current player threw two equal
        if(currentPlayer.getEqualCount() > 0 && currentPlayer.getAccount().getBalance() >= 0){
            nextPlayer = currentPlayer;
        }
        //current player is last player in array
        else if (currentPlayer == players[players.length-1]){
            nextPlayer = players[0];
        }
        //find next player in array
        else{
            int arrayIndex = 0;
            for (int i=0;i<players.length;i++){
                if (currentPlayer == players[i]){
                    arrayIndex=i;
                }
            }
            nextPlayer = players[arrayIndex+1];
        }

        return nextPlayer;
    }

}

```

```
package controller;

import java.awt.Color;

public class GUIController {

    public static void initializeBoard(Board board){
        Field[] fields = board.getFields();
        desktop_fields.Field[] graphicfields = new desktop_fields.Field[fields.length];

        for(int i = 0; i < fields.length; i++){
            if (fields[i] instanceof Brewery){
                graphicfields[i] = new desktop_fields.Brewery.Builder()
                        .setTitle(Messages.getFieldNames()[i])
                        .setDescription(Messages.getFieldNames()[i])
                        .setSubText(determineSubText(board, i))
                        .setRent(determineRent(board, i))
                        .build();
            }
            else if (fields[i] instanceof Fleet){
                graphicfields[i] = new desktop_fields.Shipping.Builder()
                        .setTitle(Messages.getFieldNames()[i])
                        .setDescription(Messages.getFieldNames()[i])
                        .setSubText(determineSubText(board, i))
                        .setRent(determineRent(board, i))
                        .build();
            }
            else if (fields[i] instanceof Tax){
                graphicfields[i] = new desktop_fields.Tax.Builder()
                        .setTitle(determineSubText(board, i))
                        .setDescription(Messages.getFieldNames()[i])
                        .setSubText(determineSubText(board, i))
                        .build();
            }
            else if (fields[i] instanceof GoToPrison || i == 10){
                    graphicfields[i] = new desktop_fields.Jail.Builder()
                            .setTitle(determineSubText(board, i))
```

```java
                            .setDescription(Messages.getFieldNames()[i])
                            .setSubText(determineSubText(board, i))
                            .build();
            }
            else if (fields[i] instanceof Chance){
                    graphicfields[i] = new desktop_fields.Chance.Builder()
//                          .setTitle(determineSubText(board, i))
//                          .setDescription(Messages.getFieldNames()[i])
//                          .setSubText(determineSubText(board, i))
                            .build();
            }
            else if (i == 20){
                graphicfields[i] = new desktop_fields.Refuge.Builder()
                        .setTitle(Messages.getFieldNames()[i])
                        .setDescription(Messages.getFieldNames()[i])
                        .setSubText(determineSubText(board, i))
                        .build();
            }
            else if (i == 0){
                graphicfields[i] = new desktop_fields.Start.Builder()
                        .setTitle(Messages.getFieldNames()[i])
                        .setDescription(Messages.getFieldNames()[i])
                        .setSubText(determineSubText(board, i))
                        .build();
            }
            else{
                graphicfields[i] = new desktop_fields.Street.Builder()
                        .setBgColor(fields[i].getColor())
                        .setTitle(Messages.getFieldNames()[i])
                        .setDescription(Messages.getFieldNames()[i])
                        .setSubText(determineSubText(board, i))
                        .setRent(determineRent(board, i))
                        .build();
            }


        }

        GUI.create(graphicfields);
        GUI.displayChanceCard();
    }

    public static void addPlayer(String name, int balance, Color pieceColor){
        Car car = new Car.Builder()
                .primaryColor(pieceColor)
                .build();
        GUI.addPlayer(name, balance, car);
        GUI.setBalance(name, balance);
        GUI.setCar(1,name);
    }

    public static void removeAllCars(String playerName){
        GUI.removeAllCars(playerName);
    }

    private static String determineSubText(Board board, int fieldNumber){
        Field[] fields = board.getFields();

        String text = "";
        if(fields[fieldNumber] instanceof Ownable){
            text += Messages.getBoardMessages()[0]; //Price:
            text += " " + String.valueOf(((Ownable) fields[fieldNumber]).getPrice());
        }
        else if(fields[fieldNumber] instanceof Tax){
            text += Messages.getBoardMessages()[3]; //Pay:
            text += " " + String.valueOf(((Tax) fields[fieldNumber]).getTaxAmount());
            if(((Tax) fields[fieldNumber]).getTaxRate() > 0){
                text += " " + Messages.getBoardMessages()[4];
                text += " " + String.valueOf(((Tax) fields[fieldNumber]).getTaxRate());
                text += "% " + Messages.getBoardMessages()[5];
            }
        }
        return text;
    }

    private static String determineRent(Board board, int fieldNumber){
        Field[] fields = board.getFields();

        String rent = "";
        if(fields[fieldNumber] instanceof Ownable){
            rent += Messages.getGeneralMessages()[22] + String.valueOf((((Ownable) fields[fieldNumber]).getRent()));
        }

        return rent;
    }

    public static String getUserChoice(String message, String ... options){
        return GUI.getUserSelection(message, options);
    }

    public static String getUserButtonPressed(String message, String ... buttons){
        return GUI.getUserButtonPressed(message, buttons);
    }

    public static void showMessage(String message){
        GUI.showMessage(message);
    }

    public static void setDice(int value1, int value2){
        GUI.setDice(value1, value2);
    }

    public static void setCar(int position, String playerName){
        GUI.setCar(position, playerName);
    }

    public static void setFieldOwner(String playerName, int fieldNumber){
            GUI.setOwner(fieldNumber, playerName);
    }

    public static void removeFieldOwner(int fieldNumber){
            GUI.removeOwner(fieldNumber);
    }

    public static void setPlayerBalance(String playerName, int balance){
        GUI.setBalance(playerName, balance);
    }

    public static String getUserSelection(String message, String ... options){
        return GUI.getUserSelection(message, options);
    }

    public static void setHouses(int fieldNumber, int houses){
        GUI.setHouses(fieldNumber, houses);
    }

    public static void setHotel(int fieldNumber){
        GUI.setHotel(fieldNumber, true);
    }

}


```

```java
package controller;

import desktop_resources.GUI;

public class Manno1936 {

    public static void main(String[] args) {
        GameController game = new GameController(); //Initialize game

        while (GUIController.getUserButtonPressed(Messages.getGeneralMessages()[6] //do you want to create new game?
                , Messages.getGeneralMessages()[1] //yes
                        , Messages.getGeneralMessages()[2] //no
                ).equals(Messages.getGeneralMessages()[1]) //user chooses yes
            ){
            int playerAmount = Integer.parseInt(GUIController.getUserChoice(Messages.getGeneralMessages()[8],"3","4","5","6"));
            game.resetGame(playerAmount, 1500); //Reset the game with new amount of players and set start balance
            game.playGame();
            GUI.close();
        }
        GUI.close();
        System.exit(0);
    }

}
```

```java
package controller;

import java.awt.Color;

import entity.*;

public class TurnController {
    protected DiceCup dice; //An instance of the DiceCup class
    protected Field currentField;
    protected Player player;
    protected Board board;

    protected final int prisonEscapeFine = 50;
    protected final int payday = 200;

    protected boolean movingToPrison;
    protected boolean movingPiece;

    public TurnController(Player player, Board board){
        this.player = player;
        this.board = board;

        dice = new DiceCup(6,2);

    }

    public void playTurn(){
        //This used to be in the constructor, but problems arrived when trying to implement
        //a test class inheriting from this class as it requires the super class constructor to be used,
        //which in turn prevents us from doing any of our own determined inputs

        do{

            boolean buyHouseChoice = false;

            //we go through owned buildable streets
            for (Street ownedStreet : player.getAccount().getBuildableStreets()){

                Color fieldColor = ownedStreet.getColor();

                //we find number of streets in group
                int groupAmount = 0;
                for (Field boardField : board.getFields()){
                    if (boardField.getColor() == fieldColor && boardField instanceof Street){
                        groupAmount++;
                    }
                }

                //we find streets in group owned by player
                int ownedInGroup = 0;
                for(Ownable ownedField : player.getAccount().getOwnedFields()){
                    if (ownedField.getColor() == fieldColor && ownedField instanceof Street){
                        ownedInGroup++;
                    }
                }

                //we find out if player has all streets in group
                if (ownedInGroup == groupAmount){ //player owns all in group
                    buyHouseChoice = true;
                    break;
                }
            }

            //Player can buy houses
            if (buyHouseChoice == true){

                //Do you want to throw dice or buy houses/hotels
                player.setChoice(determineUserInput(new String[]{Messages.getGeneralMessages()[11] + player.getName() + Messages.getGeneralMessages()[12],
                    Messages.getGeneralMessages()[7],
                    Messages.getGeneralMessages()[21]}));

                //Player wants to buy house or hotel
                if (player.getChoice().equals(Messages.getGeneralMessages()[21])){
                    buyHouseHotel();
                }
            }
            //Player can only throw dice
            else{
                player.setChoice(determineUserInput(new String[]{Messages.getGeneralMessages()[11] + player.getName() + Messages.getGeneralMessages()[12],
                        Messages.getGeneralMessages()[7]}));
            }
        }
        while(!player.getChoice().equals(Messages.getGeneralMessages()[7])); //player does not want to throw dice

        movingPiece = false; // A boolean to define whether or not a piece shall move. I saw it necessary to simplify the code.

        // Player is not in prison
        if (player.getPrisonCount() == 0) {
            throwDice();
            movingPiece = true;
        }
        // If player is in Prison
        else if (player.getPrisonCount() > 0) {
            prisonEscape();
        }

        if (movingPiece) {
            movePiece();
            landOnField();
            if((currentField instanceof Chance) && (Chance.isMoveCard() == true)){
                GUIController.removeAllCars(player.getName());
                currentField = board.getFields()[player.getPiece().getPosition()-1];
                GUIController.setCar(player.getPiece().getPosition(), player.getName());
                landOnField();
            }
        }
    }

    protected void throwDice(){
```

```java
        dice.throwDice();
        player.setLastThrow(dice);
        GUIController.setDice(dice.getDice()[0].getValue(),dice.getDice()[1].getValue());

        //Are dice equal?
        if(dice.isEqual() == true){
            //Player has thrown equals less than 3 times in a row
            if (player.getEqualCount() != 2){
                player.setEqualCount(player.getEqualCount()+1);
                movingToPrison = false;
            }
            //Player has thrown equals 3 times in a row
            else{
                determineUserInput(new String[]{Messages.getGeneralMessages()[29], Messages.getGeneralMessages()[29]});
                movingToPrison = true;
                player.setEqualCount(0);
            }
        }
        else{
            player.setEqualCount(0);
            movingToPrison = false;
        }

    }

    protected void buyHouseHotel(){

        Street[] ownedStreets = player.getAccount().getBuildableStreets();

        //we go through owned buildable streets
        for (Street ownedStreet : player.getAccount().getBuildableStreets()){

            Color fieldColor = ownedStreet.getColor();

            //we find number of streets in group and min and max houses
            int groupAmount = 0;
            int maxHouses = 0;
            int minHouses = 5;
            for (Field boardField : board.getFields()){
                if (boardField.getColor() == fieldColor && boardField instanceof Street){
                    groupAmount++;

                    //we find current min amount of houses in group
                    if(((Street) boardField).getHousesOwned() < minHouses){
                        minHouses = ((Street) boardField).getHousesOwned();
                    }

                    //we find current max amount of houses in group
                    if(((Street) boardField).getHousesOwned() > maxHouses){
                        maxHouses = ((Street) boardField).getHousesOwned();
                    }
                }
            }

            //we find streets in group owned by player
            int ownedInGroup = 0;
            for(Ownable ownedField : player.getAccount().getOwnedFields()){
                if (ownedField.getColor() == fieldColor && ownedField instanceof Street){
                    ownedInGroup++;
                }
            }

            //we find out if player has all streets in group
            if (ownedInGroup == groupAmount //player owns all in group
                && (ownedStreet.getHousesOwned() < maxHouses //field has less houses than other field in group
                    || (ownedStreet.getHousesOwned() == maxHouses && minHouses == maxHouses))){ //all fields have same number of houses
                //everything is good
            }
            //we remove street from ownedStreets
            else{
                Street[] oldArray = ownedStreets;
                ownedStreets = new Street[oldArray.length-1];

                int counter = 0;
                for (Street i : oldArray){
                    if (i != ownedStreet){
                        ownedStreets[counter] = i;
                        counter++;
                    }
                }
            }
        }

        //we find names of the streets
        String[] fieldNames = new String[ownedStreets.length];

        for (int i = 0; i < fieldNames.length; i++){
            fieldNames[i] = Messages.getFieldNames()[ownedStreets[i].getId()-1];
        }

        //For which street do you want a house/hotel?
        String userChoice = GUIController.getUserSelection(Messages.getGeneralMessages()[30], fieldNames);

        Street chosenField = null;
        for (Field i : board.getFields()){
            if (Messages.getFieldNames()[i.getId()-1].equals(userChoice)){
                chosenField = (Street) i;
            }
        }

        //Are you sure?
        String areYouSure = determineUserInput(new String[]{Messages.getGeneralMessages()[23] + chosenField.getHousePrice() +
Messages.getGeneralMessages()[24]
                , Messages.getGeneralMessages()[1], Messages.getGeneralMessages()[2]});

        if (areYouSure.equals(Messages.getGeneralMessages()[1])){
            chosenField.setHousesOwned(chosenField.getHousesOwned()+1);

            if (chosenField.getHousesOwned() == 5){
                GUIController.setHotel(chosenField.getId());
```

```java
            }
            else{
                GUIController.setHouses(chosenField.getId(),chosenField.getHousesOwned());
            }

            player.getAccount().setBalance(player.getAccount().getBalance()-chosenField.getHousePrice());
            GUIController.setPlayerBalance(player.getName(), player.getAccount().getBalance());
        }

    }

    protected void movePiece(){
        int oldPosition = player.getPiece().getPosition();  // Save the old player position
        int position;

        /*
         * If there has already been placed a car, we remove it before placing a new one
         */
        GUIController.removeAllCars(player.getName());

        // Make sure the player did not throw 3 equals in a row.
        if (movingToPrison == false) {
            /*
             * Position is equal to the current position + the sum of the dice throw
             * We use modulus to calculate whether the player would end up outside the board
             */
            position = (player.getPiece().getPosition() + dice.getSum()) % board.getFields().length;
            //Since we use mod 40, we need to have a special case for field 40, or else we get position == 0
            if (position == 0){
                position = board.getFields().length;
            }

            //We set the car and piece position to the new values
            player.getPiece().setPosition(position);
            GUIController.setCar(player.getPiece().getPosition(),player.getName());
            currentField = board.getFields()[position-1];

            // Money when passing or landing on start
            if (oldPosition > position) {
                if (position == 1) {
                    determineUserInput(new String[]{
                            Messages.getGeneralMessages()[26] + Messages.getFieldNames()[0] +       // You landed on Start
                            Messages.getGeneralMessages()[28] + payday});                           // and receives the payday amount (200)
                }else {
                    determineUserInput(new String[]{Messages.getGeneralMessages()[27] + Messages.getFieldNames()[0] +       // You passed start
                            Messages.getGeneralMessages()[28] + payday});                                   // and receives the payday amount (200)

                }
                player.getAccount().setBalance(player.getAccount().getBalance() + payday);

                GUIController.setPlayerBalance(player.getName(), player.getAccount().getBalance());
            }
        }
        else{
            currentField = board.getFields()[10];
            moveToPrison(); // Moves the player and its piece straight to prison
        }
    }

    protected void landOnField(){
        // You landed on
        if (player.getPiece().getPosition()-1 != 0) { // No need to tell that you landed on start, when the MovePiece says it
            determineUserInput(new String[]{Messages.getGeneralMessages()[26] + Messages.getFieldNames()[(player.getPiece().getPosition())-1]});
        }

        if (player.getPiece().getPosition()-1 == 30) { // goToPrison field.
            GUIController.removeAllCars(player.getName());
            moveToPrison();
            determineUserInput(new String[]{Messages.getGeneralMessages()[29]});
        }

        int playerBalance = player.getAccount().getBalance();

    //Ownable
        if (currentField instanceof Ownable) {
            Player owner = ((Ownable) currentField).getOwner();
            int price = ((Ownable) currentField).getPrice();

        // Do you wish to buy it?
            if (owner == null && playerBalance >= price) {
                String playerChoice = determineUserInput(new String[]{
                        Messages.getGeneralMessages()[0] + ((Ownable) currentField).getPrice() + "?", //Do you want to buy field?
                        Messages.getGeneralMessages()[1],   // Yes
                        Messages.getGeneralMessages()[2]    // No
                                });

                player.setChoice(playerChoice);

                if (playerChoice.equals(Messages.getGeneralMessages()[1])) { // User chooses yes
                    GUIController.setFieldOwner(player.getName(), player.getPiece().getPosition());
                }
            }
            // You don't have enough money to buy field
            else if(owner == null && playerBalance < price){
                determineUserInput(new String[]{player.getName() + ": " + Messages.getGeneralMessages()[25]});
            }
        // You own the field
            else if (owner == player){
                determineUserInput(new String[]{player.getName() + ": " + Messages.getGeneralMessages()[20]});
            }
        // You have to pay rent
            else if (owner != player){
                int rent = 0;
                if (currentField instanceof Brewery){
                    //when brewery we should multiply dice sum with 4 or 10, depending on the amount of owned brewery from the 
same owner.
                    rent = ((Ownable) currentField).getRent()*player.getLastThrow().getSum();
        }
        else{
          rent = ((Ownable) currentField).getRent();
        }

        determineUserInput(new String[]{player.getName() + ": " + Messages.getGeneralMessages()[9] + rent + Messages.getGeneralMessages()[16]});
      }
    }

  //Tax
    else if( currentField instanceof Tax) {
      if ( ((Tax) currentField).getTaxRate() > 0) {

      String playerChoice = determineUserInput(new String[]{
          Messages.getGeneralMessages()[3],
          Messages.getGeneralMessages()[4] + ((Tax) currentField).getTaxRate() + Messages.getGeneralMessages()[5],    // Percent taxes of all assets
          Messages.getGeneralMessages()[4] + ((Tax) currentField).getTaxAmount()});   // Fixed amount of tax

      player.setChoice(playerChoice);
      }
    }
  //Chance
    else if( currentField instanceof Chance){
      determineUserInput(new String[]{Messages.getChanceMessages()[Chance.getCardID()]});
    }

    currentField.landOnField(player);
    GUIController.setPlayerBalance(player.getName(),player.getAccount().getBalance());
    if (currentField instanceof Ownable){
      if (((Ownable) currentField).getOwner() != null){
        GUIController.setPlayerBalance(((Ownable) currentField).getOwner().getName(),((Ownable) currentField).getOwner().getAccount().getBalance());
      }
    }
  }
```

```java


    protected String determineUserInput(String[] input){
        String text;
        switch(input.length) {
            case 1:
                GUIController.showMessage(input[0]);
                text = Messages.getGeneralMessages()[13]; //OK
                break;
            case 2:
                text = GUIController.getUserButtonPressed(input[0], input[1]);
                break;
            case 3:
                text = GUIController.getUserButtonPressed(input[0], input[1], input[2]);
                break;
            case 4:
                text = GUIController.getUserButtonPressed(input[0], input[1], input[2], input[3]);
                break;
            default:
                text = "";
                break;
        }
        return text;
    }

    protected void moveToPrison() {
        GUIController.removeAllCars(player.getName());
        player.getPiece().setPosition(11);
        player.setPrisonCount(3);
        GUIController.setCar(player.getPiece().getPosition(),player.getName());
        player.setEqualCount(0);
        movingToPrison = false;
    }

    protected void prisonEscape() {
        // Player is in prison and has several attempts to get out.
        if (player.getPrisonCount() > 1) {
            String playerChoice = determineUserInput(new String[]{
                    Messages.getGeneralMessages()[32],      //"De er i fængsel. For at komme ud kan de betale 50 kr. eller slå to ens"
                    Messages.getGeneralMessages()[31],      //"Betal 50 kr."
                    Messages.getGeneralMessages()[7]});     //"Slå med terningerne"

            player.setChoice(playerChoice);

            //Player wants to throw dice
            if(playerChoice.equals(Messages.getGeneralMessages()[7])){
                throwDice();

                if (dice.isEqual() == true) {
                    player.setEqualCount(1);
                    player.setPrisonCount(0);
                    movingPiece = true;
                }else {
                    player.setEqualCount(0);
                    player.setPrisonCount(player.getPrisonCount()-1);
                }
            }
            //Player wants to pay fine
            else if(playerChoice.equals(Messages.getGeneralMessages()[31])){
                player.setEqualCount(0);
                player.setPrisonCount(0);
                player.getAccount().setBalance(player.getAccount().getBalance()-prisonEscapeFine); // pay the fine for getting out of prison
                GUIController.setPlayerBalance(player.getName(), player.getAccount().getBalance());
                throwDice();
                movingPiece = true;
            }
        }
        // Player is using his last attempt to get out
        else if(player.getPrisonCount() == 1) {
            throwDice();
            if (dice.isEqual() == true) {
                player.setEqualCount(1);
                player.setPrisonCount(0);
            }
            else {
                player.setEqualCount(0);
                player.setPrisonCount(0);
                player.getAccount().setBalance(player.getAccount().getBalance()-prisonEscapeFine); // pay the fine for getting out of prison
                GUIController.setPlayerBalance(player.getName(), player.getAccount().getBalance());
            }
            movingPiece = true;
        }
    }

    protected void setDice(DiceCup dice){
        this.dice = dice;
    }
}

```

```java
package entity;

public class Account {
    //not implemented private int jailFreeCounter;
    private int balance;
    private Ownable[] ownedFields;

    public Account(int startBalance){
        //jailFreeCounter = 0;
        this.balance = startBalance;

        ownedFields = new Ownable[0];
    }

    public int calculateAssets(){
        int assets = balance;

        for (Ownable i : ownedFields){
            if (i != null){
                assets += i.getPrice();
            }
        }
        return assets;
    }

    public void setBalance(int balance){
        this.balance = balance;
    }

    public int getBalance(){
        return balance;
    }

//  public void setJailFreeCounter(int jailFreeCounter){
//      this.jailFreeCounter = jailFreeCounter;
//  }

//  public int getJailFreeCounter(){
//      return jailFreeCounter;
//  }
```

```java

    public Ownable[] getOwnedFields(){
        return ownedFields;
    }

    public void setOwnedField(Ownable field){
        //add new Ownable to Ownable array

        Ownable[] temp = ownedFields;
        ownedFields = new Ownable[temp.length+1];

        //insert previous owned fields into new array
        for (int i = 0; i<temp.length;i++){
            ownedFields[i] = temp[i];
        }

        //insert new owned field into new array
        ownedFields[temp.length] = field;

    }

    public Street[] getBuildableStreets(){

        //find number of streets
        int streetCounter = 0;
        for (Ownable i : ownedFields){
            if (i instanceof Street
                    && ((Street) i).getHousesOwned() < 5
                    && balance >= ((Street) i).getHousePrice()){
                streetCounter++;
            }
        }

        //create new Street array
        Street[] ownedStreets = new Street[streetCounter];

        streetCounter = 0;
        for (int i = 0; i < ownedFields.length; i++){
            if (ownedFields[i] instanceof Street
                    && ((Street) ownedFields[i]).getHousesOwned() < 5
                    && balance >= ((Street) ownedFields[i]).getHousePrice()){
                ownedStreets[streetCounter] = (Street) ownedFields[i];
                streetCounter++;
            }
        }

        return ownedStreets;
    }
}

```

```java
package entity;

import java.awt.Color;

public class Board {
    private Field[] fields;

    public Board(){

        fields = new Field[40];

/**/        fields[0] = new Refuge(1, Color.white); //Start
        fields[1] = new Street(2, Color.cyan, 60, new int[]{2,10,30,90,160,250},50);        //Rødovrevej
/**/        fields[2] = new Chance(3, Color.white); //Chance
        fields[3] = new Street(4, Color.cyan, 60, new int[]{4,20,60,180,320,540},50);       //Hvidovre
        fields[4] = new Tax(5, Color.white, 10, 200);   //Betal Indkomstskat
        fields[5] = new Fleet(6, Color.white, 200);     //Øresund A/S
        fields[6] = new Street(7, Color.pink, 100, new int[]{6,30,90,270,400,550},50);      //Roskildevej
/**/        fields[7] = new Chance(8, Color.white); //Chance
        fields[8] = new Street(9, Color.pink, 100, new int[]{6,30,90,270,400,550},50);      //Valby Langgade
        fields[9] = new Street(10, Color.pink, 120, new int[]{8,40,100,300,450,600},50);        //Allégade
/*MISSING*/     fields[10] = new Refuge(11, Color.white);   //Fængsel
        fields[11] = new Street(12, Color.green, 140, new int[]{10,50,150,450,625,750},100);    //Frederiksberg Allé
        fields[12] = new Brewery(13, Color.white, 150); //Tuborg
        fields[13] = new Street(14, Color.green, 140, new int[]{10,50,150,450,625,750},100);    //Bülowsvej
        fields[14] = new Street(15, Color.green, 160, new int[]{12,60,180,500,700,900},100);    //Gl. Kongevej
        fields[15] = new Fleet(16, Color.white, 200);   //D.F.D.S
        fields[16] = new Street(17, Color.gray, 180, new int[]{14,70,200,550,750,950},100); //Bernstorffsvej
/**/        fields[17] = new Chance(18, Color.white);   //Chance
        fields[18] = new Street(19, Color.gray, 180, new int[]{14,70,200,550,750,950},100); //Hellerupvej
        fields[19] = new Street(20, Color.gray, 200, new int[]{16,80,220,600,800,1000},100);    //Strandvej
        fields[20] = new Refuge(21, Color.white);   //Helle
        fields[21] = new Street(22, Color.red, 220, new int[]{18,90,250,700,875,1050},150);     //Trianglen
/**/        fields[22] = new Chance(23, Color.white);   //Chance
        fields[23] = new Street(24, Color.red, 220, new int[]{18,90,250,700,875,1050},150);     //Østerbrogade
        fields[24] = new Street(25, Color.red, 240, new int[]{20,100,300,750,925,1100},150);        //Grønningen
        fields[25] = new Fleet(26, Color.white, 200);   //Ø.K.
        fields[26] = new Street(27, Color.magenta, 260, new int[]{22,110,330,800,975,1150},150);    //Bredgade
        fields[27] = new Street(28, Color.magenta, 260, new int[]{22,110,330,800,975,1150},150);    //Kongens Nytorv
        fields[28] = new Brewery(29, Color.white, 150); //Carlsberg
        fields[29] = new Street(30, Color.magenta, 280, new int[]{22,120,360,850,1025,1200},150);   //Østergade
        fields[30] = new GoToPrison(31, Color.white);   //Politi (GoToPrison)
        fields[31] = new Street(32, Color.yellow, 300, new int[]{26,130,390,900,1100,1275},200);    //Amagertorv
        fields[32] = new Street(33, Color.yellow, 300, new int[]{26,130,390,900,1100,1275},200);    //Vimmelskaftet
/**/        fields[33] = new Chance(34, Color.white);   //Chance
        fields[34] = new Street(35, Color.yellow, 320, new int[]{28,150,450,1000,1200,1400},200);   //Nygade
        fields[35] = new Fleet(36, Color.white, 200);   //D/S Bornholm
/**/        fields[36] = new Chance(37, Color.white);   //Chance
        fields[37] = new Street(38, Color.orange, 350, new int[]{35,175,500,1100,1300,1500},200);   //Frederiksberggade
        fields[38] = new Tax(39, Color.white, 0, 100);  //Ekstraordinær statsskat
        fields[39] = new Street(40, Color.orange, 400, new int[]{50,200,600,1400,1700,2000},200);   //Rådhuspladsen

    }

    public Field[] getFields(){
        return fields;
    }
}

```

```java
package entity;

import java.awt.Color;

public class Brewery extends Ownable {

    private final int FACTOR_1 = 4;
    private final int FACTOR_2 = 10;

    public Brewery(int id, Color color, int price) {
        super(id, color, price);
    }

    @Override
    public int getRent() {
        int ownedBreweries = 0;
        int rent = 0;

        if (this.getOwner() != null){
            for (Ownable i : this.getOwner().getAccount().getOwnedFields()){

                if (i instanceof Brewery){
                    ownedBreweries++;
                }

            }
        }

        switch (ownedBreweries){
        case 1: rent = FACTOR_1; break;
        case 2: rent = FACTOR_2; break;
        default: rent = 0;
        }

        return rent;

    }

}
```

```java
package entity;

import java.awt.Color;

public class Chance extends Field {

    public Chance(int id, Color color) {
        super(id, color);
        shuffleDeck();
    }
    //All cardtypes are identified by an integer
    private static int[] chanceCards = {

//not implemented// 0: "Ryk brikken frem til det nærmeste dampskibsselskab og betal ejeren to gange den leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen, kan De købe det af banken.", // Two of these
            1,  //  "Tag med Øresundsbåden - Flyt brikken frem, og hvis De passerer >>Start<<, indkassér kr. 200,00.",
            2,  //  "Ryk frem til Frederiksberg Allé. Hvis De passerer >>Start<<, indkassér kr. 200,00.",
            3,  //  "Ryk frem til Grønningen. Hvis De passerer >>Start<<, indkassér da kr. 200,00.",
            4,  //  "Tag ind på Rådhuspladsen.",
            5,  //  "Ryk frem til >>Start<<.",
            6, 6,   //  "Ryk tre felter tilbage.", // Two of these
            // Property charges
//not implemented// 7: "Ejendomsskatterne er steget, ekstraudgifterne er: kr. 50,00 pr. hus, kr. 125,00 pr. hotel.",
//not implemented// 8: "Kul- og kokspriserne er steget, og De skal betale: kr. 25,00 pr. hus, kr. 125,00 pr. hotel.",
            // Expenses
            9,  //  "De har kørt frem for >>Fuld Stop<<. Betal kr. 100,00 i bøde.",
            10, //  "De har anskaffet et nyt dæk til Deres vogn. Indbetal kr. 100,00.",
            11, //  "Betal kr. 75,00 for modtagne 2 kasser øl.",
            12, //  "De har måttet vedtage en parkeringsbøde. Betal kr. 20,00 til banken.",
            13, //  "Betal for vognvask og smøring kr. 10,00.",
            14, //  "De har været en tur i udlandet og haft for mange cigaretter med hjem. - Betal told kr. 20,00.",
            // Prison
            15, 15,//   "Gå i fængsel. Ryk direkte til fængslet. Selv om De passerer >>Start<<, indkasserer De ikke kr. 200,00.", // Two of these
            // Prison mercy
//not implemented// 16: "I anledning af Kongens fødselsdag benådes De herved for fængsel. Dette kort kan opbevares, indtil De får brug for det, eller De kan sælge det.", // Two of these
            // For the needy
            17, //  "De modtager >>Matador-legatet for værdig trængende<<, stort kr. 2000,00. Ved værdig trængende forstås, at Deres formue, d.v.s. Deres kontante penge + skøder + bygninger, ikke overstiger kr. 750,00.",
            // Bonuses
            18, //  "Deres præmieobigation er kommet ud. De modtager kr. 100,00 af banken.",
            19, //  "Værdien af egen avl fra nyttehaven udgør kr. 200,00, som De modtager af banken.",
            20, //  "Efter auktionen på Assistenhuset, hvor De havde pantsat Deres tøj, modtager De ekstra kr. 108,00.",
            21, //  "De har rettidigt afleveret Deres abonnementskort. Depositum kr. 1,00 udbetales Dem af banken.",
            22, //  "Modtag udbytte af Deres aktier: kr. 50,00.",
            23, //  "Manufakturvarerne er blevet billigere og bedre, herved sparer De kr. 50,00, som De modtager af banken.",
            24, //  "Kommunen har eftergivet et kvartals skat, hæv i banken til en glad aften kr. 150,00.",
            25, //  "De har solgt Deres gamle klude. Modtag kr. 20,00.",
            26, //  "Grundet på dyrtiden har De fået gageforhøjelse. Modtag kr 25,00.",
            // Money collector
//not implemented// 27: "De har lagt penge ud til sammenskudsgilde. Mærkværdigvis betaler alle straks. Modtag fra hver medspiller kr. 25,00."
    };

    @Override
    public void landOnField(Player player) {
        switch(chanceCards[0]){
//not implemented       case 0:
//              break;
        case 1: passedStart(player, 6);
                player.getPiece().setPosition(6);
                break;
        case 2: passedStart(player, 12);
                player.getPiece().setPosition(12);
                break;
        case 3: passedStart(player, 25);
                player.getPiece().setPosition(25);
                break;
        case 4: player.getPiece().setPosition(40);
                break;
        case 5: passedStart(player, 1);
                player.getPiece().setPosition(1);
                break;
        case 6: if(player.getPiece().getPosition() == 3){   //if the player draws this card on the first chanceCard field
                    player.getPiece().setPosition(40);      //move to position 40
                }
                else{
                    player.getPiece().setPosition(player.getPiece().getPosition()-3); //ryk 3 felter tilbage
                }
```

```java
                    break;
//not implemented       case 7:
//                  break;
//not implemented       case 8:
//                  break;
            case 9: player.getAccount().setBalance(player.getAccount().getBalance()-100);
                    break;
            case 10:player.getAccount().setBalance(player.getAccount().getBalance()-100);
                    break;
            case 11:player.getAccount().setBalance(player.getAccount().getBalance()-75);
                    break;
            case 12:player.getAccount().setBalance(player.getAccount().getBalance()-20);
                    break;
            case 13:player.getAccount().setBalance(player.getAccount().getBalance()-10);
                    break;
            case 14:player.getAccount().setBalance(player.getAccount().getBalance()-20);
                    break;
            case 15:player.getPiece().setPosition(11); //move player to prison
                    player.setPrisonCount(3);
                    break;
//not implemented       case 16:
//                  break;
            case 17:if(player.getAccount().calculateAssets() <= 750){
                        player.getAccount().setBalance(player.getAccount().getBalance()+2000);
                        }
                    break;
            case 18:player.getAccount().setBalance(player.getAccount().getBalance()+100);
                    break;
            case 19:player.getAccount().setBalance(player.getAccount().getBalance()+200);
                    break;
            case 20:player.getAccount().setBalance(player.getAccount().getBalance()+108);
                    break;
            case 21:player.getAccount().setBalance(player.getAccount().getBalance()+1);
                    break;
            case 22:player.getAccount().setBalance(player.getAccount().getBalance()+50);
                    break;
            case 23:player.getAccount().setBalance(player.getAccount().getBalance()+50);
                    break;
            case 24:player.getAccount().setBalance(player.getAccount().getBalance()+150);
                    break;
```

```java
        case 25:player.getAccount().setBalance(player.getAccount().getBalance()+20);
                break;
        case 26:player.getAccount().setBalance(player.getAccount().getBalance()+25);
                break;
//not implemented       case 27:
//              break;
        }
//      if(chanceCards[0] == 16){  //When a jailFree card is drawn, it should be removed from the deck
//          removeCard();
//      }
//      else{
            moveCardToBottom();
//      }

    }

    public void addCard(int cardID){ //adds a card to the bottom of the deck
        int[] tempDeck = new int[chanceCards.length+1];
        for(int i = 0; i < tempDeck.length; i++){
            if(i < tempDeck.length-1){
                tempDeck[i] = chanceCards[i];
            }
            else{
                tempDeck[i] = cardID;
            }
        }
        chanceCards = tempDeck;
    }

    private void removeCard(){  //removes the top card of the deck
        int[] tempDeck = new int[chanceCards.length-1];
        for(int i = 1; i < chanceCards.length; i++){
            tempDeck[i-1] = chanceCards[i];
            }
        chanceCards = tempDeck;
        }

    private void moveCardToBottom(){  //moves the top card of the deck to the bottom
        addCard(chanceCards[0]);
        removeCard();
        }

    private void shuffleDeck(){
        for(int i = 0; i < chanceCards.length; i++){
            int ranNum = ( (int) (Math.random()*(chanceCards.length)));
            int currentCard = chanceCards[i];
            chanceCards[i] = chanceCards[ranNum];
            chanceCards[ranNum] = currentCard;
        }
    }
    private void passedStart(Player player, int newPosition){
        int oldPosition = player.getPiece().getPosition();
        if(oldPosition > newPosition){                                              //did the player pass start?
            player.getAccount().setBalance(player.getAccount().getBalance()+200);   //The player receives 200
        }
    }
    public static int getCardID(){
        return chanceCards[0];

    }
    public static boolean isMoveCard(){
        boolean isMoveCard = false;
        switch(chanceCards[chanceCards.length-1]){
        case 1: isMoveCard = true;
                break;
        case 2: isMoveCard = true;
                break;
        case 3: isMoveCard = true;
                break;
        case 4: isMoveCard = true;
                break;
        case 5: isMoveCard = true;
                break;
        case 6: isMoveCard = true;
                break;
        case 15: isMoveCard = true;
                break;
        default: isMoveCard = false;
        }
        return isMoveCard;
    }

    public void setDeck(int[] data){
        chanceCards = data;
    }

}

```

```java
package entity;

public class Dice {
    protected int value;
    protected int sides;

//Constructor to set amount of dice-sides and the amount of dices
//furthermore uses the setAllValuesRandom method which simulates a roll
    public Dice(int diceSides){
        this.sides = diceSides;
        this.setRandom();
    }

    public int getValue(){
        return value;
    }

    //Simulates a roll of the chosen dice(s)
    public void setRandom(){
        value = ( (int) (Math.random()*sides)+1);
    }

    public void setValue(int value){
        this.value = value;
    }
}

```

```java
package entity;

public class DiceCup {
    protected Dice[] dice;

//Constructor to set amount of dice-sides and the amount of dices
//furthermore uses the setAllValuesRandom method which simulates a roll
    public DiceCup(int diceSides, int diceAmount){
        dice = new Dice[diceAmount];

        for (int i = 0; i<dice.length; i++){
            dice[i] = new Dice(diceSides);
        }

        throwDice();
    }

    public Dice[] getDice(){
        return dice;
    }

    public int getSum(){
        int sum = 0;
        for (Dice i : dice){
            sum += i.getValue();
        }
        return sum;
    }

    public void throwDice(){
        for (Dice i : dice){
            i.setRandom();
        }
    }

    public boolean isEqual(){
        boolean isEqual = true;
        int value = dice[0].getValue();

        //Checks if the first value of dice is equal to the rest
        for (Dice i : dice){
            if (value != i.getValue()){
                isEqual = false;
            }
        }

        return isEqual;
    }

}

```

```java
package entity;

import java.awt.Color;

public abstract class Field {
    protected int id;
    protected Color color;

    public Field(int id, Color color){
        this.id = id;
        this.color = color;
    }

    public Color getColor(){
        return color;
    }

    public int getId(){
        return id;
    }

    public abstract void landOnField(Player player);

}

```

```java
package entity;

import java.awt.Color;

public class Fleet extends Ownable {

    private final int[] RENTS = new int[]{25,50,100,200};

    public Fleet(int id, Color color, int price) {
        super(id, color, price);
    }

    @Override
    public int getRent() {
        int ownedFleets = 0;
        int rent = 0;

        if (this.getOwner() != null){
            for (Ownable i : this.getOwner().getAccount().getOwnedFields()){

                if (i instanceof Fleet){
                    ownedFleets++;
                }

            }
            rent = RENTS[ownedFleets-1];
        }

        return rent;
    }

}

```

```java
package entity;

import java.awt.Color;

public class GoToPrison extends Field{

    public GoToPrison(int id, Color color) {
        super(id, color);

    }
    @Override
    public void landOnField(Player player) {
            player.getPiece().setPosition(11);
            player.setPrisonCount(3);
    }


}

```

```java
package entity;

public class Messages {
    private static String[] chanceMessages = {
                // Cards resulting in new position of piece
            /*0*/   "Ryk brikken frem til det nærmeste dampskibsselskab og betal ejeren to gange den leje, han ellers er berettiget til. Hvis selskabet ikke ejes af nogen, kan De købe det af banken.", // Two of these
            /*1*/   "Tag med Øresundsbåden - Flyt brikken frem, og hvis De passerer >>Start<<, indkassér kr. 200,00.",
            /*2*/   "Ryk frem til Frederiksberg Allé. Hvis De passerer >>Start<<, indkassér kr. 200,00.",
            /*3*/   "Ryk frem til Grønningen. Hvis De passerer >>Start<<, indkassér da kr. 200,00.",
            /*4*/   "Tag ind på Rådhuspladsen.",
            /*5*/   "Ryk frem til >>Start<<.",
            /*6*/   "Ryk tre felter tilbage.", // Two of these
                // Property charges
            /*7*/   "Ejendomsskatterne er steget, ekstraudgifterne er: kr. 50,00 pr. hus, kr. 125,00 pr. hotel.",
            /*8*/   "Kul- og kokspriserne er steget, og De skal betale: kr. 25,00 pr. hus, kr. 125,00 pr. hotel.",
                // Expenses
            /*9*/   "De har kørt frem for >>Fuld Stop<<. Betal kr. 100,00 i bøde.",
            /*10*/  "De har anskaffet et nyt dæk til Deres vogn. Indbetal kr. 100,00.",
            /*11*/  "Betal kr. 75,00 for modtagne 2 kasser øl.",
            /*12*/  "De har måttet vedtage en parkeringsbøde. Betal kr. 20,00 til banken.",
            /*13*/  "Betal for vognvask og smøring kr. 10,00.",
            /*14*/  "De har været en tur i udlandet og haft for mange cigaretter med hjem. - Betal told kr. 20,00.",
                // Prison
            /*15*/  "Gå i fængsel. Ryk direkte til fængslet. Selv om De passerer >>Start<<, indkasserer De ikke kr. 200,00.", // Two of these
                // Prison mercy
            /*16*/  "I anledning af Kongens fødselsdag benådes De herved for fængsel. Dette kort kan opbevares, indtil De får brug for det, eller De kan sælge det.", // Two of these
                // For the needy
            /*17*/  "De modtager >>Matador-legatet for værdig trængende<<, stort kr. 2000,00. Ved værdig trængende forstås, at Deres formue, d.v.s. Deres kontante penge + skøder + bygninger, ikke overstiger kr. 750,00.",
                // Bonuses
            /*18*/  "Deres præmieobigation er kommet ud. De modtager kr. 100,00 af banken.",
            /*19*/  "Værdien af egen avl fra nyttehaven udgør kr. 200,00, som De modtager af banken.",
            /*20*/  "Efter auktionen på Assistenhuset, hvor De havde pantsat Deres tøj, modtager De ekstra kr. 108,00.",
            /*21*/  "De har rettidigt afleveret Deres abonnementskort. Depositum kr. 1,00 udbetales Dem af banken.",
            /*22*/  "Modtag udbytte af Deres aktier: kr. 50,00.",
            /*23*/  "Manufakturvarerne er blevet billigere og bedre, herved sparer De kr. 50,00, som De modtager af banken.",
            /*24*/  "Kommunen har eftergivet et kvartals skat, hæv i banken til en glad aften kr. 150,00.",
            /*25*/  "De har solgt Deres gamle klude. Modtag kr. 20,00.",
            /*26*/  "Grundet på dyrtiden har De fået gageforhøjelse. Modtag kr 25,00.",
                // Money collector
            /*27*/  "De har lagt penge ud til sammenskudsgilde. Mærkværdigvis betaler alle straks. Modtag fra hver medspiller kr. 
25,00."
    };
    private static String[] fieldNames = {
            "Start"                     //Field 1
            ,"Rødovrevej"               //Field 2
            ,"Prøv lykken"              //Field 3
            ,"Hvidovre"                 //Field 4
            ,"Betal Indkomstskat"       //Field 5
            ,"Øresund A/S"              //Field 6
            ,"Roskildevej"              //Field 7
            ,"Prøv lykken"              //Field 8
            ,"Valby Langgade"           //Field 9
            ,"Allégade"                 //Field 10
            ,"Fængsel"                  //Field 11
            ,"Frederiksberg Allé"       //Field 12
            ,"Tuborg"                   //Field 13
            ,"Bülowsvej"                //Field 14
            ,"Gl. Kongevej"             //Field 15
            ,"D.F.D.S"                  //Field 16
            ,"Bernstorffsvej"           //Field 17
            ,"Prøv lykken"              //Field 18
            ,"Hellerupvej"              //Field 19
            ,"Strandvej"                //Field 20
            ,"Helle"                    //Field 21
            ,"Trianglen"                //Field 22
            ,"Prøv lykken"              //Field 23
            ,"Østerbrogade"             //Field 24
            ,"Grønningen"               //Field 25
            ,"Ø.K."                     //Field 26
            ,"Bredgade"                 //Field 27
            ,"Kgs. Nytorv"              //Field 28
            ,"Carlsberg"                //Field 29
            ,"Østergade"                //Field 30
            ,"De sættes i fængsel"      //Field 31
            ,"Amagertorv"               //Field 32
            ,"Vimmelskaftet"            //Field 33
            ,"Prøv lykken"              //Field 34
            ,"Nygade"                   //Field 35
            ,"D/S Bornholm"             //Field 36
            ,"Prøv lykken"              //Field 37
            ,"Frederiksberggade"        //Field 38
            ,"Ekstraordinær Statsskat"  //Field 39
            ,"Rådhuspladsen"            //Field 40
    };
    private static String[] boardMessages = {
            "Pris:",                //0
            "Leje:",                //1
            "Modtag:",              //2
            "Betal:",               //3
            "eller",                //4
            "af alle ejendele",     //5
    };

    private static String[] generalMessages = {
            /*0*/   "Denne ejendom er ikke ejet af nogen spiller. Vil De købe den for ",
            /*1*/   "Ja",
            /*2*/   "Nej",
            /*3*/   "De har nu to muligheder",
            /*4*/   "Betal ",
            /*5*/   "% af alle ejendele ",
            /*6*/   "Vil De starte et nyt spil?",
            /*7*/   "Slå med terningerne",
            /*8*/   "Hvor mange spillere skal deltage i spillet?",
            /*9*/   "De er landet på en anden spillers ejendom. De skal betale ",
            /*10*/  "Spiller ",
            /*11*/  "Det er ",
            /*12*/  "'s tur.",
            /*13*/  "OK",
            /*14*/  "Tillykke ",
            /*15*/  ", De har vundet spillet!",
            /*16*/  " i leje.",
            /*17*/  "De skal betale ",
            /*18*/  " til skattefar.",
            /*19*/  "De modtager ",
            /*20*/  "De er landet på deres egen ejendom og nyder de dejlige omgivelser.",
            /*21*/  "Køb hus eller hotel",
            /*22*/  "Leje: ",
            /*23*/  "De skal betale ",
            /*24*/  " kr. Er De sikker?",
            /*25*/  "De har ikke nok penge til at købe dette felt.",
            /*26*/  "De er landet på ",
            /*27*/  "De har passeret ",
            /*28*/  ", og modtager ",
            /*29*/  "De er blevet stoppet af politiet og sendes direkte i fængsel!",
            /*30*/  "Hvilket felt vil De købe hus eller hotel til?",
            /*31*/  "Betal 50 kr.",
            /*32*/  "De er i fængsel. For at komme ud kan de betale 50 kr. eller slå to ens"
    };

    public static String[] getChanceMessages(){
        return chanceMessages;
    }

    public static String[] getFieldNames(){
        return fieldNames;
    }

    public static String[] getBoardMessages(){
        return boardMessages;
    }

    public static String[] getGeneralMessages(){
        return generalMessages;
    }
}
```

```java
package entity;

import java.awt.Color;

public abstract class Ownable extends Field{

    protected int price;
    protected Player owner;

    public Ownable(int id, Color color, int price){
        super(id, color);
        this.price = price;
    }

    @Override
    public void landOnField(Player player){
        int balance = player.getAccount().getBalance();

        //no owner of field
        if (owner == null){
            //player can afford field and wants to buy field
            if (balance >= price && player.getChoice().equals(Messages.getGeneralMessages()[1])){
                owner = player;
                player.getAccount().setOwnedField(this); //Field is added to Player's owned fields
                player.getAccount().setBalance(balance-price); //Balance of Player is changed
            }
        }
        //player owns field
        else if (owner == player){

        }
        //pay rent to owner if he is not bankrupt
        else if (owner.getAccount().getBalance() >= 0){
            int rent = 0;
            if (this instanceof Brewery){
                //Brewery is a special case as we need the player's last dice throw to calculate rent
                //Since we do not pass a player to getRent(), we need to do this multiplication on this level
                rent = getRent()*player.getLastThrow().getSum();
            }
            else{
                rent = getRent();
            }
            //if the player does not have enough money to pay all the rent, the owner should only be given what's left
            if(player.getAccount().getBalance() < rent){
                owner.getAccount().setBalance(owner.getAccount().getBalance() + player.getAccount().getBalance());
            }
            else{
                owner.getAccount().setBalance(owner.getAccount().getBalance() + rent);
            }
            player.getAccount().setBalance(balance - rent);
        }
    }

    public int getPrice(){
        return price;
    }

    public abstract int getRent();

    public Player getOwner(){
        return owner;
    }

    public void setOwner(Player owner){
        this.owner = owner;
    }

}

```

```java
package entity;

import java.awt.Color;

public class Piece {
    private int position;
    private Color color;

    // Constructor - In order to create the piece, you will need to give the vehicle a color using java.awt.Color;
    public Piece(Color color) {
        this.color = color;
        position = 1; //player starts on field 1
    }

    public Color getColor() {
        return this.color;
    }

    // Move the piece to a position using an integer
    public void setPosition(int position) {
        this.position = position;
    }

    public int getPosition() {
        return position;
    }

}

```

```java
package entity;

public class Player {

    private int id;
    private String name;
    private Piece piece;
    private Account account;
    private String choice;
    private int prisonCount; //counts number of turns in prison. If 0 then player is not in prison.
    private int equalCount; //counts number of equal diceValues in DiceCup in a row.
    private DiceCup lastThrow;

    public Player(int id, String name, Piece piece, Account account){
        this.id = id;
        this.name = name;
        this.piece = piece;
        this.account = account;
        prisonCount = 0;
        equalCount = 0;
    }

    public String getName(){
        return name;
    }

    public Account getAccount(){
        return account;
    }

    public Piece getPiece(){
        return piece;
    }

    public int getID(){
        return id;
    }

    public String getChoice(){
        return choice;
    }

    public void setChoice(String choice){
        this.choice = choice;
    }

    public int getPrisonCount(){
        return prisonCount;
    }

    public void setPrisonCount(int prisonCount){
        this.prisonCount = prisonCount;
    }

    public int getEqualCount(){
        return equalCount;
    }

    public void setEqualCount(int equalCount){
        this.equalCount = equalCount;
    }

    public DiceCup getLastThrow(){
        return lastThrow;
    }

    public void setLastThrow(DiceCup lastThrow){
        this.lastThrow = lastThrow;
    }


}

```

```java
package entity;

import java.awt.Color;

public class Refuge extends Field{

    public Refuge(int id, Color color) {
        super(id, color);

    }

    @Override
    public void landOnField(Player player) {

    }

}

```

```java
package entity;

import java.awt.Color;

public class Street extends Ownable {

    private final int[] rents;
    private int housesOwned;
    private final int housePrice;

    public Street(int id, Color color, int price, int[] rents, int housePrice) {
        super(id, color, price);

        this.rents = rents;
        housesOwned = 0;
        this.housePrice = housePrice;
    }

    @Override
    public int getRent() {
        int rent = 0;

        if (housesOwned == 0 && owner != null){
            //find amount of streets in group
            int groupAmount;
            if (color == Color.cyan || color == Color.orange){
                groupAmount = 2;
            }
            else{
                groupAmount = 3;
            }

            //find streets in group owned by player
            int ownedInGroup = 0;
            for(Ownable i : owner.getAccount().getOwnedFields()){
                if (i.getColor() == color && i instanceof Street){
                    ownedInGroup++;
                }
            }

            //double rent
            if (ownedInGroup == groupAmount){
                rent = rents[0]*2;
            }
            else{
                rent = rents[0];
            }
        }
        else{
            rent = rents[housesOwned];
        }

        return rent;

    }

    public int getHousesOwned(){
        return housesOwned;
    }

    public int getHousePrice(){
        return housePrice;
    }

    public void setHousesOwned(int amount){
        housesOwned = amount;
    }

}

```

```java
package entity;

import java.awt.Color;

public class Tax extends Field {
    private int taxRate; //in percent. If 0 then no taxRate
    private int taxAmount;

    public Tax(int id, Color color, int taxRate, int taxAmount){
        super(id, color);
        this.taxRate = taxRate;
        this.taxAmount = taxAmount;
    }

    @Override
    public void landOnField(Player player){
        int balance = player.getAccount().getBalance();

        if (taxRate > 0){
            String choice = player.getChoice();
            if (choice.equals(Messages.getGeneralMessages()[4] + taxAmount)){//user chooses taxAmount
                player.getAccount().setBalance(balance - taxAmount);
            }
            else{//user chooses taxRate
                player.getAccount().setBalance(balance - (int)((taxRate/100.0) * player.getAccount().calculateAssets()));
            }
        }
        else{
            player.getAccount().setBalance(balance - taxAmount);
        }
    }

    public int getTaxAmount(){
        return taxAmount;
    }

    public int getTaxRate(){
        return taxRate;
    }
}
```