SE 116 CARD GAME PROJECT REPORT

PREPARED BY

Fatih Anamaslı

Ali Berat Akoğlu

Berke Erenç

CONTENT

PREPARED BY
Functional Requirement 1
The program must be able to create a deck of cards:
Functional Requirement 2 :
The program must be able to shuffle and cut the deck
Functional Requirement 3:
he program must be able to read game parameters from the command line
Functional Requirement 4:
The program must be able to move cards from the deck to the players and the boards.
Functional Requirement 5:
The program must be able to calculate each player’s score
Functional Requirement 6:
The program must be able to store a “high score list” on a file
Functional Requirement 7:
The program must include a novice player (level 1)
Functional Requirement 8:
The program must include a regular player.
Functional Requirement 9:
The program must include an expert player.
Functional Requirement 10:
The program must include a human player
DetailRate:
Lvl 0
Lvl1
Lvl2
Top 10:
Card Points List:
Last Winner Collect All Cards:
Git SS:

NOT: Argument should follow this order:

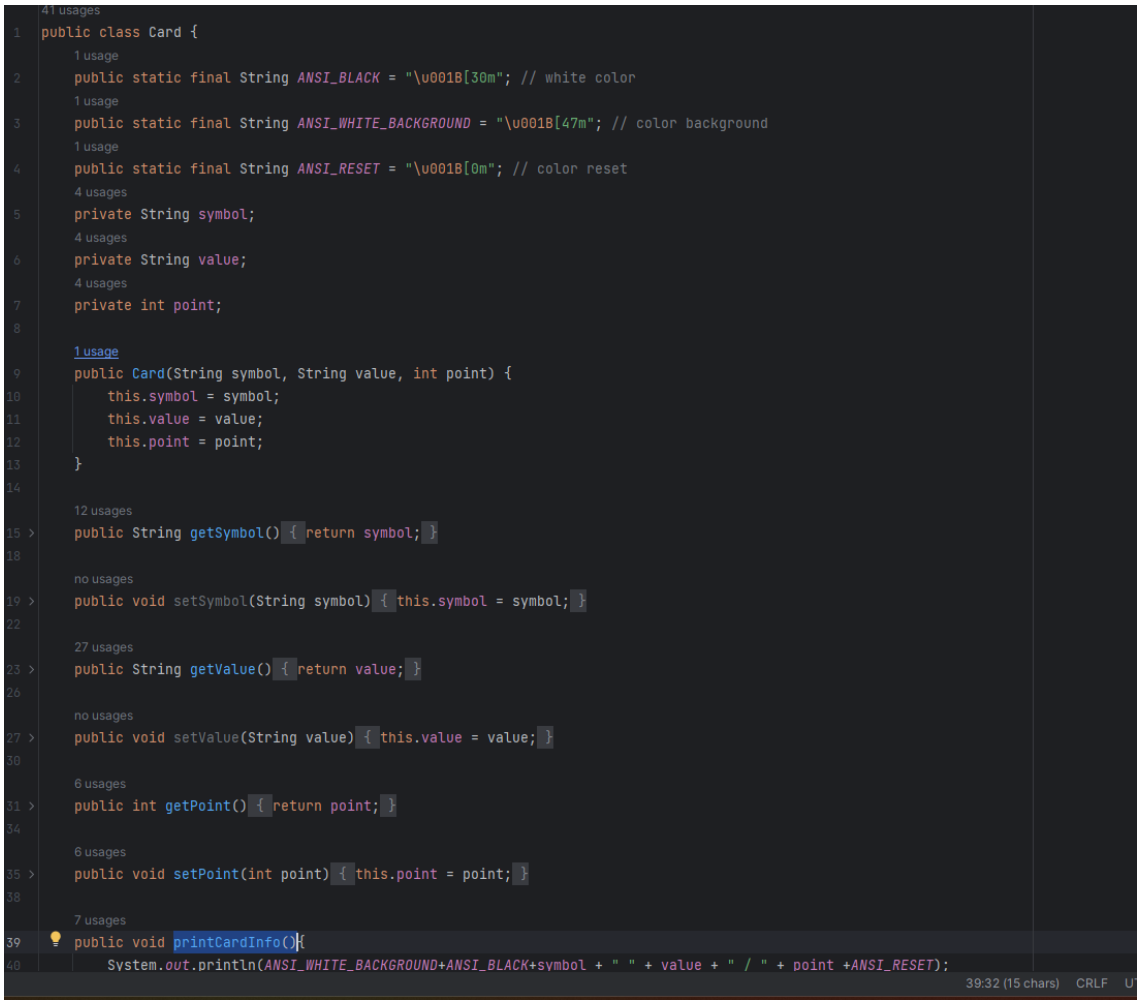
- ***Txt_file_name - player number(3) - is there human player[0,1] - name1 - name2 - name3 - lvl[1,3] - lvl[1,3] - lvl[1,3] - detailRate[0,2] (pls instead of - use space)***

Be careful this order is designed for 3 bot players. If you want to play with less or more than 3 bot you should increase names and lvl information.

Functional Requirement 1

The program must be able to create a deck of cards:

Static Final Variables: These are constant variables, meaning that once they are assigned, they cannot be changed. In this case, ANSI_BLACK, ANSI_WHITE_BACKGROUND, and ANSI_RESET are being assigned ANSI escape codes, which are used to format text output in the console (such as colour and background colour). Here, ANSI_BLACK is misleadingly commented as 'white colour' but the ANSI code suggests it's black colour, and ANSI_WHITE_BACKGROUND is for setting white color as the background.



```
41 usages
public class Card {
    1 usage
    public static final String ANSI_BLACK = "\u001B[30m"; // white color
    1 usage
    public static final String ANSI_WHITE_BACKGROUND = "\u001B[47m"; // color background
    1 usage
    public static final String ANSI_RESET = "\u001B[0m"; // color reset
    4 usages
    private String symbol;
    4 usages
    private String value;
    4 usages
    private int point;

    1 usage
    public Card(String symbol, String value, int point) {
        this.symbol = symbol;
        this.value = value;
        this.point = point;
    }

    12 usages
    public String getSymbol() { return symbol; }

    no usages
    public void setSymbol(String symbol) { this.symbol = symbol; }

    27 usages
    public String getValue() { return value; }

    no usages
    public void setValue(String value) { this.value = value; }

    6 usages
    public int getPoint() { return point; }

    6 usages
    public void setPoint(int point) { this.point = point; }

    7 usages
    public void printCardInfo(){
        System.out.println(ANSI_WHITE_BACKGROUND+ANSI_BLACK+symbol + " " + value + " / " + point +ANSI_RESET);
39:32 (15 chars)  CRLF  U
```

Instance Variables: The Card class has three instance variables: symbol, value, and point. These variables represent the properties of a card:symbol could represent the suit of the card (like hearts, diamonds, clubs, or spades).value might represent the value of the card (like '2', '3', '4', '5', '6', '7', '8', '9', '10', 'J', 'Q', 'K', 'A').point could be the points associated with the card.Constructor: The Card class has a constructor that takes three parameters: symbol, value, and point. When a new Card object is created, this constructor is used to initialise the symbol, value, and point of the card. This keyword refers to the current instance of the class.

Functional Requirement 2 :

The program must be able to shuffle and cut the deck

```
// Card52 shuffling
Collections.shuffle(cards52);
```

The code Collections.shuffle(cards52) shuffles the elements in the cards52 ArrayList, changing their order randomly.

```
int halfSize = cards52.size() / 2;

ArrayList<Card> firstHalf = new ArrayList<Card>(cards52.subList(0, halfSize));
ArrayList<Card> secondHalf = new ArrayList<Card>(cards52.subList(halfSize, cards52.size()));

Collections.reverse(secondHalf);

ArrayList<Card> result = new ArrayList<Card>(secondHalf);
result.addAll(firstHalf);

System.out.println("cut process finished");
System.out.println("cut are ready to play");
```

This code first creates an ArrayList and is populated with those list items. It then splits half of the list into two separate ArrayList objects using the subList() method. It reverses the elements of the second half and then adds them to the first half using the addAll() method.

Functional Requirement 3:

he program must be able to read game parameters from the command line

```
// args = txt_file_name, player number(3), is there human player,  name1, name2, name3, lvl,lvl,lvl, detailRate
```

That is the order of input which is taken from arguments. This order is critical for executing code. ***RULES:***

- before the execution of code, you HAVE TO enter some inputs
- input should follow this order:
- ***Txt_file_name - player number(5) - is there human player - name1 - name2 - name3 - lvl - lvl - lvl - detailRate (pls ınstead of - use space)***
- - playerNumber can not be bigger than 5
- Is there human player part must be 0 or 1 with any other input program will execute exit() function
- For every bot you should declare just 1 name not more. if you enter more than 1 name for a bot that will create problem
- For every bot you should declare harness level these levels are 1 - easy / 2 - normal / 3 - hard with any other input program will execute exit() function
- At the and you should declare a detail rate that number can be 0 = you will see all cards on table 1 = you will see just last card on table 2 = you will see last card of middle when turn comes you

```
public static void main(String[] args) {
    // playerList store all players
    ArrayList<Player> playerList = new ArrayList<>();

    int botNumber=0 ;
    int humanNumber =0;
    int detailRate = 0;
    // information from args
    try {
        botNumber = Integer.parseInt(args[1]);
        humanNumber = Integer.parseInt(args[2]);
        detailRate = Integer.parseInt(args[3+botNumber*2]);
        if (botNumber > 5){
            System.out.println("you can not play with more than 5 bot ");
            System.out.println("Exit function activated ");
            System.exit( status: 0);
        }

        System.out.println("botNumber " + botNumber);
        System.out.println("humanNumber " + humanNumber);
        System.out.println("detailRate " + detailRate);
    }catch (Exception e){
        e.fillInStackTrace();
        System.out.println("invalid input from args pls enter digits");
        System.exit( status: 0);
    }
```

In this part, we are controlling 3 values (botNumber, humanNumber, detailRate), the type of inputs which are taken from the **args series.** If one of these value's type is not valid to use, the catch function will execute and System.Exit(0) code will turn off the program.

```java
// detail rate if it is >2 or <0 exit system
if (detailRate <0 || detailRate >3){
    System.out.println("detailRate can not be negative number or bigger then 2 // 0<detailRate<2");
    System.exit( status: 0);
}

// if botNumber = or < 0 exit
if (botNumber <= 0 ){
    System.out.println("botNumber can not be negative value or 0");
    System.exit( status: 0);
}

// if human player > 0 and < 2 create realPlayer;
if (humanNumber==1) {
    Scanner sc = new Scanner(System.in);
    System.out.println("pls enter the human payer name");
    realPlayer realPlayer = new realPlayer(sc.nextLine());
    playerList.add(realPlayer);
}else if (humanNumber <0){
    System.exit( status: 0);
}
```

In this part we are controlling the variable's values which represent (botNumber, humanNumber, detailRate).botNumber must not equal or smaller than 0, detailRate must be smaller than 3 also must not smaller than 0, also humanPlayer must equal 0 or 1. if one of the variable's value is not valid to use, the system will execute ***exit()*** function.

```java
//declare bot players object
for (int i = 0; i < botNumber; i++) {
    try{
        Player player = new Player(Integer.parseInt(args[3+botNumber+i]),args[3+i]);
        // hardness lvl number control
        if (Integer.parseInt(args[3+botNumber+i]) >3 || Integer.parseInt(args[3+botNumber+i]) <0){
            System.out.println("invalid harness rate input");
            System.exit( status: 1);
        }
        playerList.add(player);

    }catch (Exception e){
        e.fillInStackTrace();
        System.out.println("invalid input");
        System.exit( status: 0);
    }
}
```

If variables (botNumber, humanNumber, detailRate) are usable we can try to create players objects. While creating player objects we should check args inputs which represent the level of players are valid to use. Also we must check these inputs must be in the specific number range [0,2]. If they are not in this range, the system calls the exit() function again. If values are valid we add players to our playerList Arraylist.

Functional Requirement 4:

The program must be able to move cards from the deck to the players and the boards.

```
    // 4 card to middle
    System.out.println("middle cards".toUpperCase());
    for (int i = 0; i < 4; i++) {
        middleCard.add(card52.get(0));
        card52.get(0).printCardInfo();

        //update reminderCards
        playerList[0].updateRemainingCards(card52.get(0));
        card52.remove( index: 0);
    }
    System.out.println("other cards number is "+ card52.size());
}
```

Firstly, the program sends 4 cards to the middle Card arrayList In the gameProcess contractor. We write this code in the constructor because we should just execute 1 time.

Then delete these 4 cards from card52 arrayList with remove function.

Also after sending cards to the middleCard also we called update ***updateRemainingCards*** function that is using for remember which cards are played.

```
        for (int i = 0; i < 48/(playerList.size()*4) ; i++) {
            gameProcess.send4CardToEachPlayer();

            for (int j = 0; j < 4 ; j++) {
                gameProcess.GameReady();
            }
        }
```

```
public ArrayList<Card> giveCard4(){
    ArrayList<Card> card4 = new ArrayList<>();
    tour++; // for count tour

    //hw many card will give to players IF THERE ARE NOT ENOUGH CARD
    int howManyCard = 4;
    if (playerList.length*4 > card52.size()){
        howManyCard = card52.size()/playerList.length;
    }

    for (int j = 0; j < 4; j++) {
        card4.add(card52.get(0));
        card52.remove( index: 0);
    }
        return card4;
}

1 usage
public void send4CardToEachPlayer(){
    // 4 cards to each player
    for (Player player : playerList){
        player.setComputerCard4(giveCard4());
```

In these 2 pictures which are above the first one is written in the main class and the second one is written in gameProcess Class. 2 loops were used to manage the cards. The outer loop executes calculated tour times. We calculate tour number with a basic formula: we divide 48 (52-4 card) to 4 times all player numbers. In this loop we execute the inner loop for 4 times. In this loop we call the gameReady() function. At the end when the inner loop executed 4 times. Players play all their cards then the system executes the outer loop and it executes send4CardToEachPlayer() function. In this function we send 4 cards to each players.

In the ***send4CardToEachPlayer()*** we send 4 cards to each player `computerCard4` Arraylist. We did this with the ***give4Card()*** function. It takes 4 cards from card52 and returns these 4 cards. Before the return 4 cards its clear these 4 cards from card52 Arraylist.

```
Card selectedCard = player.PlayerGameplay(middleCard,detailRate);
middleCard.add(selectedCard);
```

In this 2 line we are taking cards from players by player object's `PlayerGameplay()` and adding them to the middleCard Arraylist.

There are 3 parts of this function. One of these functions parth is executing according to the bots level. In this picture you can see level 1 and 2.

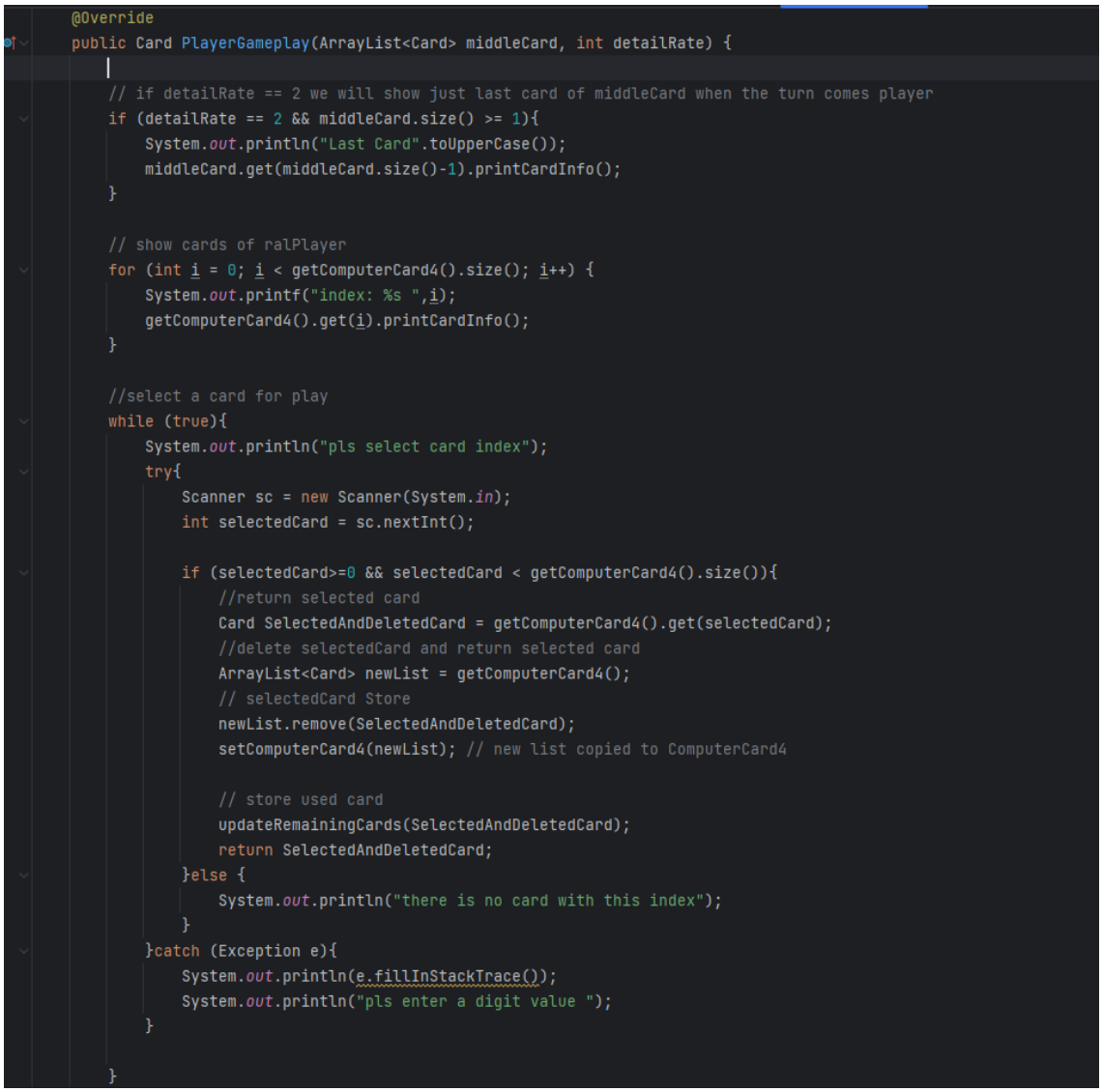
```
public Card PlayerGameplay(ArrayList<Card> middleCard, int detailRate ) {
    if (rateOfHardness == 1){
        Card a1 = computerCard4.get(0);
        computerCard4.remove( index: 0);
        updateRemainingCards(a1);
        return a1;

    } else if (rateOfHardness == 2) {

        for (Card card : computerCard4){
            if (middleCard.size()>=1 && middleCard.get(middleCard.size()-1).getValue().equals(card.getValue())){
                updateRemainingCards(card);
                computerCard4.remove(card);
                return card;
            }
        }
        // joker play
        for (Card card : getComputerCard4()){
            if (card.getValue().equals("J") && middleCard.size() >= 2){
                updateRemainingCards(card);
                computerCard4.remove(card);
                return card;
            }
        }

        Card a1 = computerCard4.get(0);
        computerCard4.remove( index: 0);
        updateRemainingCards(a1);
        return a1;
```

First level is just playing the first card of the `computerCard4` Arraylist.

Second player is playing according to the last card of the middle. It is scanning to find the same value with the last card of the middleCard. If it can not find the same value card, bot will play “J” if there is a J card in computer4Card Arraylist, also there is one more condition to play joker MiddleCards size should be bigger than 1.

If everything is okay, both will play the “J” card. If there's a problem about these 2 statement bot will play the first card of its arrayList.

NOT: for every situation before the return selectedCard we delete this card from computer4Card Arraylist and also we called updateRemainingCards(card) function to remember this card played.



```java
@Override
public Card PlayerGameplay(ArrayList<Card> middleCard, int detailRate) {

    // if detailRate == 2 we will show just last card of middleCard when the turn comes player
    if (detailRate == 2 && middleCard.size() >= 1){
        System.out.println("Last Card".toUpperCase());
        middleCard.get(middleCard.size()-1).printCardInfo();
    }

    // show cards of ralPlayer
    for (int i = 0; i < getComputerCard4().size(); i++) {
        System.out.printf("index: %s ",i);
        getComputerCard4().get(i).printCardInfo();
    }

    //select a card for play
    while (true){
        System.out.println("pls select card index");
        try{
            Scanner sc = new Scanner(System.in);
            int selectedCard = sc.nextInt();

            if (selectedCard>=0 && selectedCard < getComputerCard4().size()){
                //return selected card
                Card SelectedAndDeletedCard = getComputerCard4().get(selectedCard);
                //delete selectedCard and return selected card
                ArrayList<Card> newList = getComputerCard4();
                // selectedCard Store
                newList.remove(SelectedAndDeletedCard);
                setComputerCard4(newList); // new list copied to ComputerCard4

                // store used card
                updateRemainingCards(SelectedAndDeletedCard);
                return SelectedAndDeletedCard;
            }else {
                System.out.println("there is no card with this index");
            }
        }catch (Exception e){
            System.out.println(e.fillInStackTrace());
            System.out.println("pls enter a digit value ");
        }

    }
```

This is the humanPlayer `PlayerGameplay()` function. Before the start for design I should mention about realPlayer Class. It is the subclass of Player class. And PlayerGamePlay() function is Override for design humanPalyer's PlayerGamePlay function.

In this code after showing card's of realPlayer we are taking input which represents the index of its cards from humanPlayer. Before using this input we czech it is a digit and there is a card which has the same index number with the input. If there is a card which has a same index number with input this program returns this card.

Before the return it we call updateRemainingCards() Function and remove returned card from players 4 cards arraylist.

With these codes we are controlling card flow and preventing duplicate of lost some cards.

Functional Requirement 5:

The program must be able to calculate each player's score

```
public void pointControl(Player player){
    int lengthOfMiddleCards = middleCard.size();

    if (lengthOfMiddleCards > 1){
        //MISTI
        if (middleCard.get(lengthOfMiddleCards-1).getValue().equals(middleCard.get(lengthOfMiddleCards-2).getValue()) && lengthOfMiddleCards == 2 ){
            System.out.println("MISTI");

            for (Card card : middleCard){
                player.setPoint((card.getPoint()) * 5);
            }

            //clear middle
            middleCard.clear();

            // every player's lastWinner boolean value = false
            for (Player player1 : playerList){
                player1.setLastWinner(false);
            }
            // winner lastWinner = true
            player.setLastWinner(true);

            System.out.println(ANSI_CYAN+player.getPoint()+ " =||= " + player.getName().toUpperCase()+ANSI_RESET);
        }
        //JOKER
        else if (middleCard.get(lengthOfMiddleCards-1).getValue().equals("J")){
            for (Card card : middleCard){
                player.setPoint(player.getPoint() + card.getPoint());
            }

            System.out.println(ANSI_CYAN+player.getPoint()+ " =||= " + player.getName().toUpperCase()+ANSI_RESET);
            //clear middle
            middleCard.clear();

            // every player's lastWinner boolean value = false
            for (Player player1 : playerList){
                player1.setLastWinner(false);
            }
            // winner lastWinner = true
            player.setLastWinner(true);
        }
```

```
    //NORMAL
    else if (middleCard.get(lengthOfMiddleCards-1).getValue().equals(middleCard.get(lengthOfMiddleCards-2).getValue())){

        for (Card card : middleCard){
            player.setPoint(player.getPoint() + card.getPoint());
        }

        System.out.println(ANSI_CYAN+player.getPoint()+ " =||= " + player.getName().toUpperCase()+ANSI_RESET);
        //clear middle
        middleCard.clear();

        // every player's lastWinner boolean value = false
        for (Player player1 : playerList){
            player1.setLastWinner(false);
        }
        // winner lastWinner = true
        player.setLastWinner(true);
    }

}
```

In the `pointControl function` we are controlling, are the last 2 cards of the deck of cards the same?

Before controlling this situation we should add an if statement to control if there are more than 1 card in middleCard arraylist.

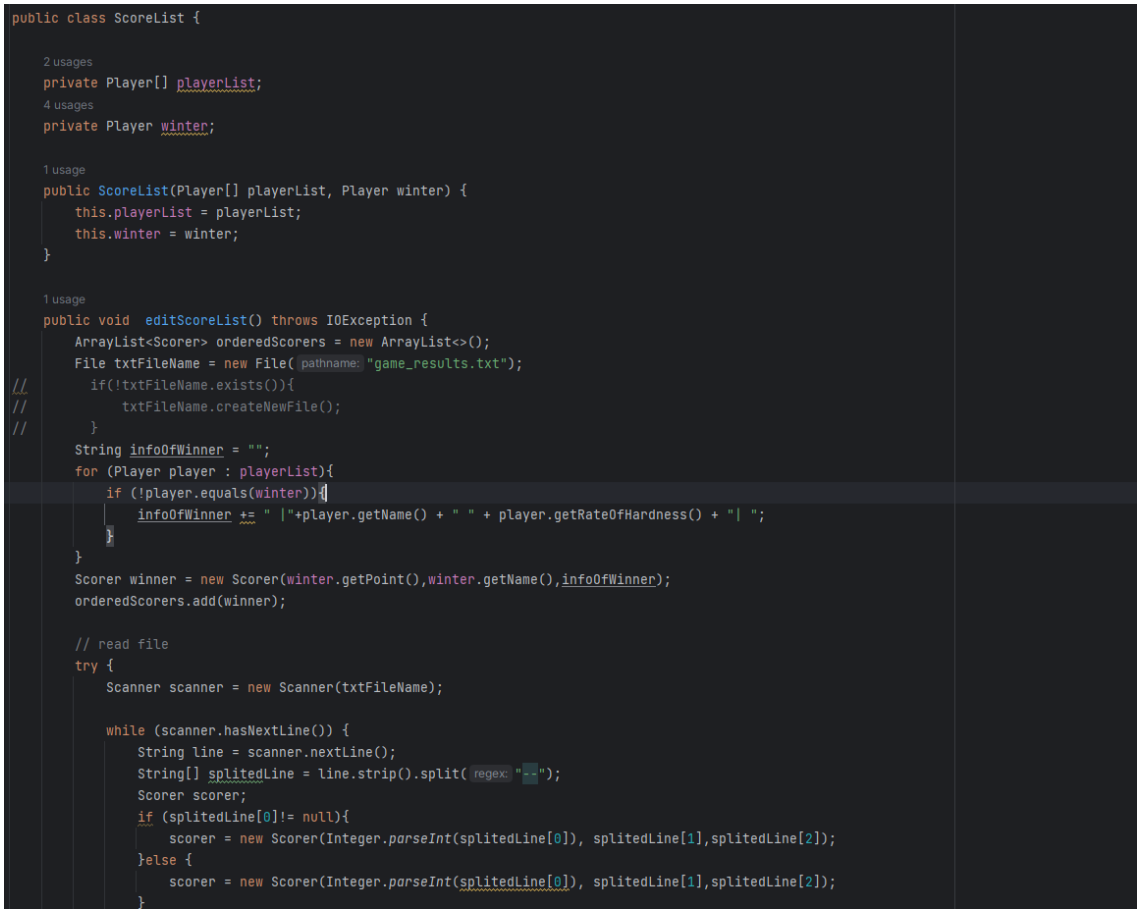
If there is, we are checking the last 2 cards values and size of the middleCard arraylist.

- If the last card's value is "J" , the player who played the J card collects all middleCards Card's points. Then we clean the middleCard Arraylist.
- If the last 2 cards' values are same and also middleCard arraylist size different from 2 player who played the last card collect all points.
- If there are just 2 cards in middleCard and these card values are same that means ""MISTI" players earn these 2 cards points * 5 points.

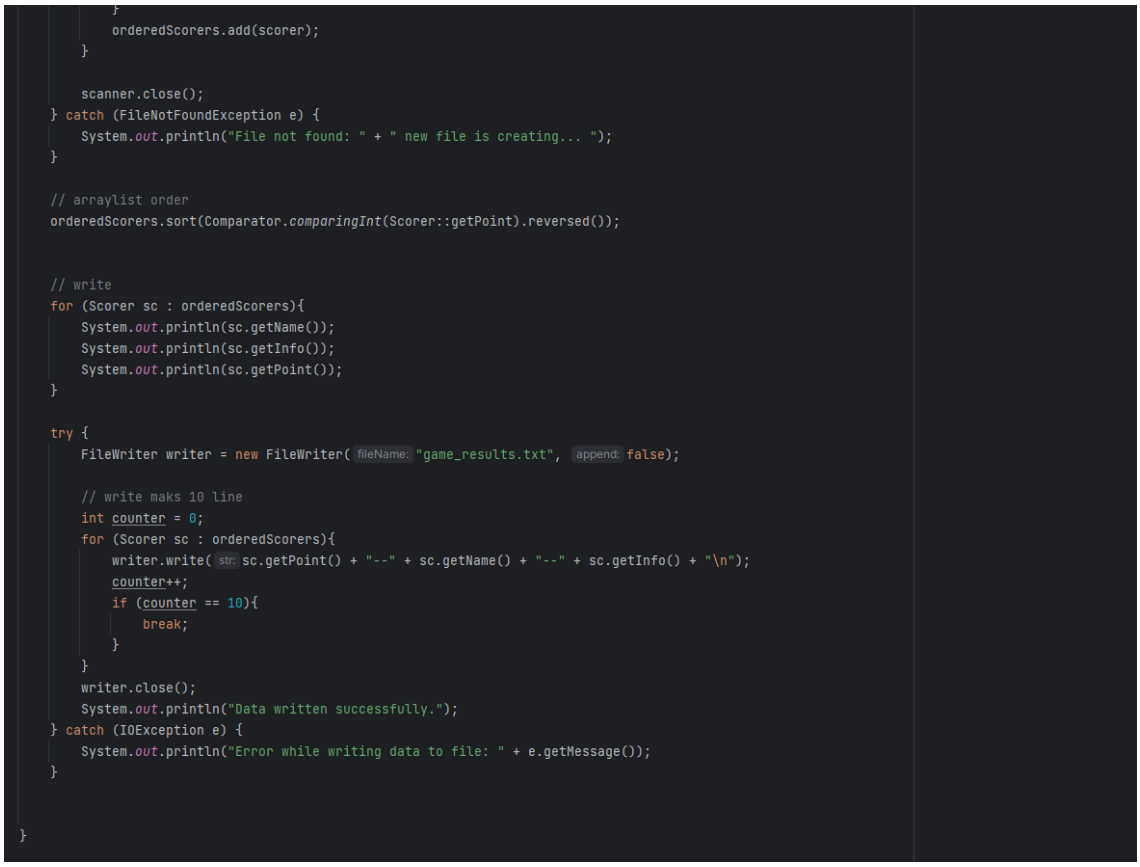
For every situation we clean the middleCard arraylist and also we update each players last winner values.

Functional Requirement 6:

The program must be able to store a “high score list” on a file



```
public class ScoreList {

    2 usages
    private Player[] playerList;
    4 usages
    private Player winter;

    1 usage
    public ScoreList(Player[] playerList, Player winter) {
        this.playerList = playerList;
        this.winter = winter;
    }

    1 usage
    public void  editScoreList() throws IOException {
        ArrayList<Scorer> orderedScorers = new ArrayList<>();
        File txtFileName = new File( pathname: "game_results.txt");
//        if(!txtFileName.exists()){
//            txtFileName.createNewFile();
//        }
        String infoOfWinner = "";
        for (Player player : playerList){
            if (!player.equals(winter)){
                infoOfWinner += " |"+player.getName() + " " + player.getRateOfHardness() + "| ";
            }
        }
        Scorer winner = new Scorer(winter.getPoint(),winter.getName(),infoOfWinner);
        orderedScorers.add(winner);

        // read file
        try {
            Scanner scanner = new Scanner(txtFileName);

            while (scanner.hasNextLine()) {
                String line = scanner.nextLine();
                String[] splitedLine = line.strip().split( regex: "--");
                Scorer scorer;
                if (splitedLine[0]!= null){
                    scorer = new Scorer(Integer.parseInt(splitedLine[0]), splitedLine[1],splitedLine[2]);
                }else {
                    scorer = new Scorer(Integer.parseInt(splitedLine[0]), splitedLine[1],splitedLine[2]);
                }
```



```
            }
            orderedScorers.add(scorer);
        }

        scanner.close();
    } catch (FileNotFoundException e) {
        System.out.println("File not found: " + " new file is creating... ");
    }

    // arraylist order
    orderedScorers.sort(Comparator.comparingInt(Scorer::getPoint).reversed());


    // write
    for (Scorer sc : orderedScorers){
        System.out.println(sc.getName());
        System.out.println(sc.getInfo());
        System.out.println(sc.getPoint());
    }

    try {
        FileWriter writer = new FileWriter( fileName: "game_results.txt",  append: false);

        // write maks 10 line
        int counter = 0;
        for (Scorer sc : orderedScorers){
            writer.write( str: sc.getPoint() + "--" + sc.getName() + "--" + sc.getInfo() + "\n");
            counter++;
            if (counter == 10){
                break;
            }
        }
        writer.close();
        System.out.println("Data written successfully.");
    } catch (IOException e) {
        System.out.println("Error while writing data to file: " + e.getMessage());
    }


}
```

ScoreList which handles a list of Player objects and generates a score list based on those players. The class is also responsible for reading and writing this score list to a text file.

The ScoreList class has three attributes:

playerList: an array of Player objects.

winter: a Player object.

orderedScorers: a list of Scorer objects, which are ordered based on points.

The class has a constructor that takes two parameters: playerList and winter. This constructor initialises the playerList and winter attributes.

The editScoreList() method is where the main functionality happens:

It first creates an instance of Scorer for the "winter" player and adds it to the list of ordered scorers.

It then attempts to read a file named game_results.txt, creating a Scorer object for each line in the file and adding it to the list. Each line in the file is assumed to be in the format

"points--name--info". If the file is not found, it prints out a message saying that a new file is being created.

After reading the file, it sorts the list of Scorer objects in descending order based on the points of each scorer.

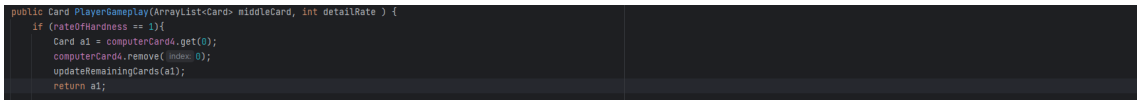
It then prints out the name, info, and points of each scorer in the list.

It attempts to write the Scorer objects to the file game_results.txt. It only writes the top 10 scorers based on the points. Each Scorer is written in the format "points--name--info".

If there's an error while writing to the file, it prints out an error message. If everything goes well, it prints out a success message.

Functional Requirement 7:

The program must include a novice player (level 1)



```
public Card PlayerGameplay(ArrayList<Card> middleCard, int detailRate ) {
    if (rateOfHardness == 1){
        Card a1 = computerCard4.get(0);
        computerCard4.remove( index: 0);
        updateRemainingCards(a1);
        return a1;
```

This bot is just playing the first card of its computerCard4 arraylist. the 0th index is the first index of its cards. When he chooses the 0th index card we remove the 0th card from arraylist then returns this selected card. When the turn came again, it chose the 0th card again but this tour's 0th card was actually last tour's 1st card. It goes on like that.

Functional Requirement 8:

The program must include a regular player.

```
} else if (rateOfHardness == 2) {

    for (Card card : computerCard4){
        if (middleCard.size()>=1 && middleCard.get(middleCard.size()-1).getValue().equals(card.getValue()) && sum>0){
            updateRemainingCards(card);
            computerCard4.remove(card);
            return card;
        }
    }
    // joker play
    for (Card card : getComputerCard4()){
        if (card.getValue().equals("J") && middleCard.size() >= 2 && sum>0){
            updateRemainingCards(card);
            computerCard4.remove(card);
            return card;
        }
    }

    Card a1 = computerCard4.get(0);
    computerCard4.remove( index: 0);
    updateRemainingCards(a1);
    return a1;
```

```
    1 usage
    public int middleSumPoint(ArrayList<Card>middleCard){
        int sum = 0;
        for(Card card : middleCard){
            sum += card.getPoint();
        }
        return sum;
    }
}
```

```
1 usage  1 override
public Card PlayerGameplay(ArrayList<Card> middleCard, int detailRate ) {
    int sum = middleSumPoint(middleCard); // middle Cards sum point
```

With the middleSumPoint() system calculating the middle cards values.

Regular bot before the playing it checks middleCard arraylist size if it is bigger than 0 it controls the last card value is it same with any card which is in its card deck, if there is a card which has the same value with last card of middle card. Lastly it checks the all middleCards collected points if also this point is bigger than 0 then it plays card.

If there are any problems, it plays the first card in its hand.

Functional Requirement 9:

The program must include an expert player.

```
        }else if (rateOfHardness == 3){

            for (Card card : computerCard4){

                if(middleCard.size()>1 && middleCard.get(middleCard.size()-1).getValue().equals(card.getValue()) && sum>0){
                    updateRemainingCards(card);
                    computerCard4.remove(card);
                    return card;

                }
//                else if (middleCard.size() >= 2  && sum>0){
//                    // joker play
//                    if (card.getValue().equals("J")){
//                        updateRemainingCards(card);
//                        computerCard4.remove(card);
//                        return card;
//                    }
//
//                }
                else {
                    Card minCard = getComputerCard4().get(0);
                    int index = 0;

                    // what is the card value it gives an index num
                    for (int i = 0; i<13;i++){
                        if (allCardsForRemember[i].equals(minCard.getValue())){
                            index = i;
                        }
                    }
                    int remainingNum = allCardsRemaining[index];

                    int index2 = 0;
                    for (Card card1 : getComputerCard4()){
                        for (int i = 0; i<13;i++){
                            if (allCardsForRemember[i].equals(card1.getValue())){
                                index2 = i;
                                if (remainingNum > allCardsRemaining[index2]){
                                    minCard = card1;
                                }
                            }

                        }
```

```
                    updateRemainingCards(minCard);
                    computerCard4.remove(minCard);
                    return minCard;
                }
            }

            Card a1 = computerCard4.get(0);
            // burada 3. seviye zorluk yaz
            computerCard4.remove( index: 0);
            updateRemainingCards(a1);
            return a1;
        }
        return null;
    }
```

This expert player behave like 2 level bot player but also it remembers all played cards and according to remembered cards it plays the most used card.

System collecting used cards data in 1 array. These are

```
    private static String[] allCardsForRemember = {"A","2","3","4","5","6","7","8","9","K","J","T","Q"};
    7 usages
    private static int[] allCardsRemaining = new int[13];
```

These are key part of remembering feature.

```
2 usages
public Player(int rateOfHardness, String name) {
    this.point = 0;
    this.rateOfHardness = rateOfHardness;
    this.name = name;

    declareArraysForRemaining();
}
```

```
// fill the remember arrays
1 usage
public static void declareArraysForRemaining(){
    // there are 4 cards which have same value
    for (int i = 0; i < 13; i++) {
        allCardsRemaining[i] = 4;
    }
}
```

Fırstly, we called the declareArrayForRemaining() function in constructor of player class.

This function just fills the array with 4. All cards for remember have 13 elements and these are the values of cards. Basically we create a relationship with these 2 arrays. For instance:

“A” is the 0th index of ***AllCardRemember*** and there are 4 cards which have “A” value. We said that ***allCardsRemaining*** array’s 0th index 4 represents 4 cards which have “A” values.

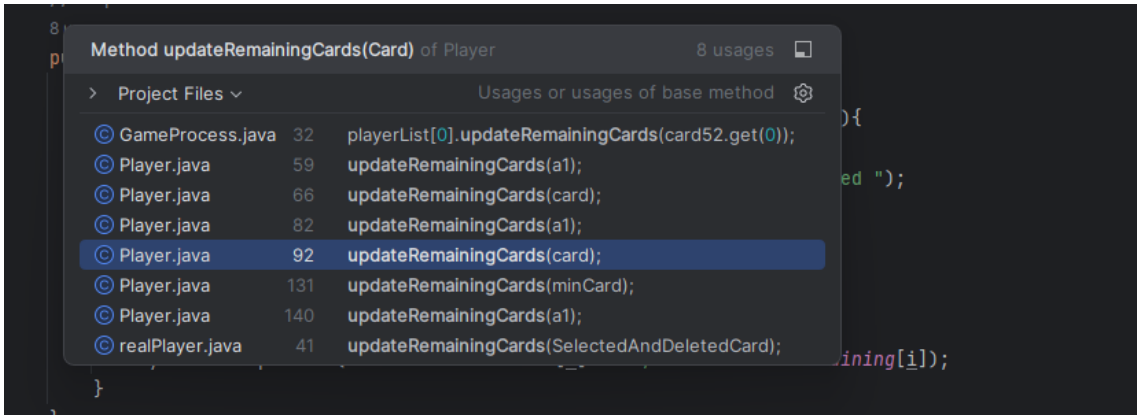
According to this system we just updated allCardsRemaining array's index’s numbers. We did this update with ***updateRemainingCards()*** function;

```
8 usages
public void updateRemainingCards(Card middleCardLastCard){
    for (int i = 0; i<13;i++){
        if (allCardsForRemember[i].equals(middleCardLastCard.getValue())){
            allCardsRemaining[i] -= 1;
            System.out.println(middleCardLastCard.getValue() + " Card used ");
        }
    }

    System.out.println("remembered Cards".toUpperCase());
    for (int i = 0; i < 13; i++) {
        System.out.println(allCardsForRemember[i] + " / " + allCardsRemaining[i]);
    }
}
```

For updating action we should just send a card information to the method. According to this card's value, firstly this method finds the values index in ***AllCardRemember.*** Then it changes the ***allCardsRemaining*** index ,which is the same with ***AllCardRemember,***by decreasing by 1.

We should call this method after every card selection.

And these points are:



before returning In the player class PlayerGameplay() method the selected card we should call this function . Also we should call this function after the HumanPlayer card selection. Lastly, if we want to count all cards, we should call it, at the beginning of the game while we are sending 4 cards to the middleCard Arraylist. These 8 places are these places.

Functional Requirement 10:

The program must include a human player

```java
public realPlayer(String name) { super( rateOfHardness: 0,name); }

1 usage
@Override
public Card PlayerGameplay(ArrayList<Card> middleCard, int detailRate) {

    // if detailRate == 2 we will show just last card of middleCard when the turn comes player
    if (detailRate == 2 && middleCard.size() >= 1){
        System.out.println("Last Card".toUpperCase());
        middleCard.get(middleCard.size()-1).printCardInfo();
    }

    // show cards of ralPlayer
    for (int i = 0; i < getComputerCard4().size(); i++) {
        System.out.printf("index: %s ",i);
        getComputerCard4().get(i).printCardInfo();
    }

    //select a card for play
    while (true){
        System.out.println("pls select card index");
        try{
            Scanner sc = new Scanner(System.in);
            int selectedCard = sc.nextInt();

            if (selectedCard>=0 && selectedCard < getComputerCard4().size()){
                //return selected card
                Card SelectedAndDeletedCard = getComputerCard4().get(selectedCard);
                //delete selectedCard and return selected card
                ArrayList<Card> newList = getComputerCard4();
                // selectedCard Store
                newList.remove(SelectedAndDeletedCard);
                setComputerCard4(newList); // new list copied to ComputerCard4

                // store used card
                updateRemainingCards(SelectedAndDeletedCard);
                return SelectedAndDeletedCard;
            }else {
                System.out.println("there is no card with this index");
            }
        }catch (Exception e){
            System.out.println(e.fillInStackTrace());
            System.out.println("pls enter a digit value ");
```

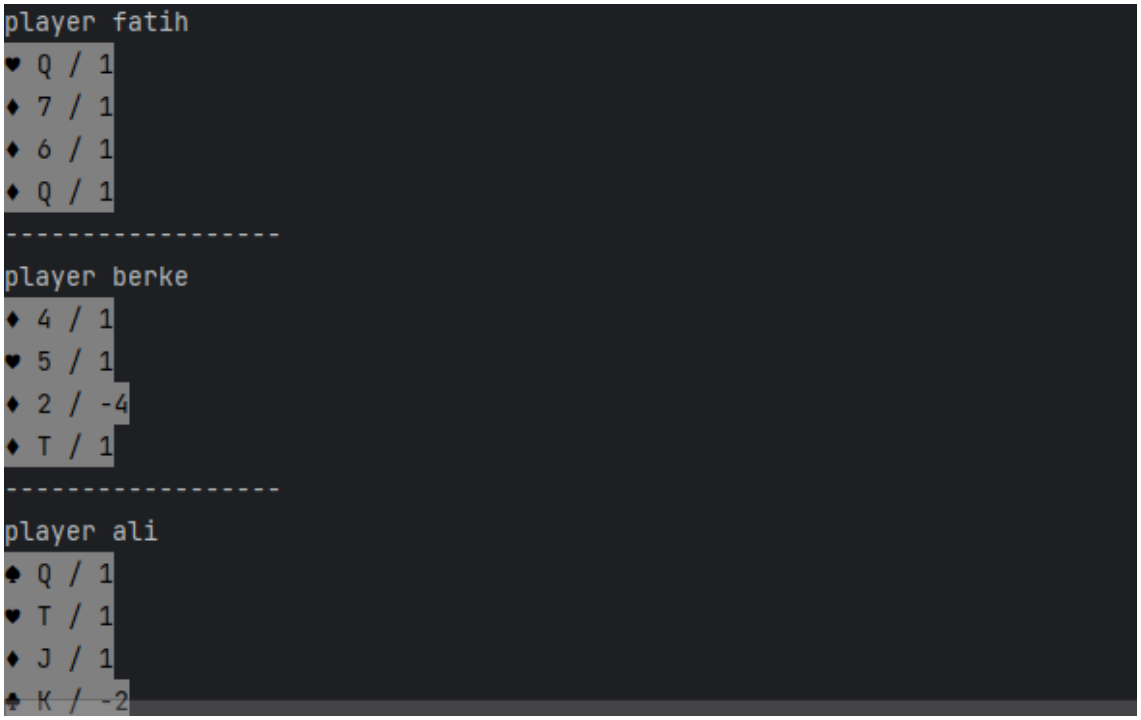
for implement the realPlayer class, we just create a subclass of Player and override the playerGamePlay() method. I mentioned that part while I was explaining card movements.

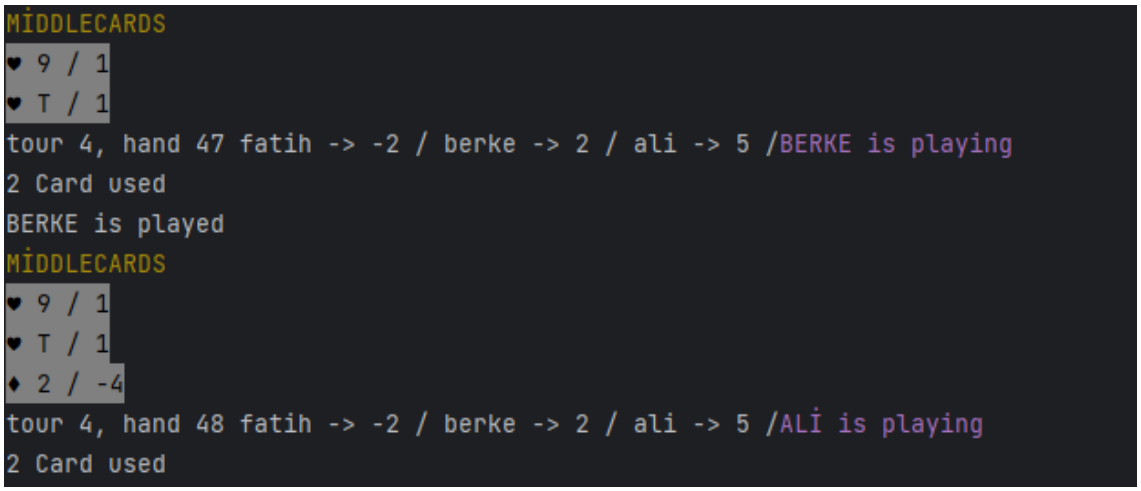
According to args[2] value we create a realPlayer Class and in this class's playerGamePlay() method we take an input which represents its cards index. We control 2 things: is this input type digit and is it usable for selecting a card. This input should not equal or bigger than the size of the computerCard4 arraylist , and it should not be smaller than 0.

If everything is correct the system returns the selected card.

DetailRate:

Lvl 0





You can see all played cards if they aren't collected by others.

Lvl1

```
tour 4, hand 40 fatih -> -10 / berke -> 8 / ali -> 0 /FATİH is playing
3 Card used
FATİH is played
LAST CARD
♠ 3 / -2
tour 4, hand 41 fatih -> -10 / berke -> 8 / ali -> 0 /BERKE is playing
9 Card used
BERKE is played
LAST CARD
♠ 9 / -2
tour 4, hand 42 fatih -> -10 / berke -> 8 / ali -> 0 /ALİ is playing
4 Card used
ALİ is played
```

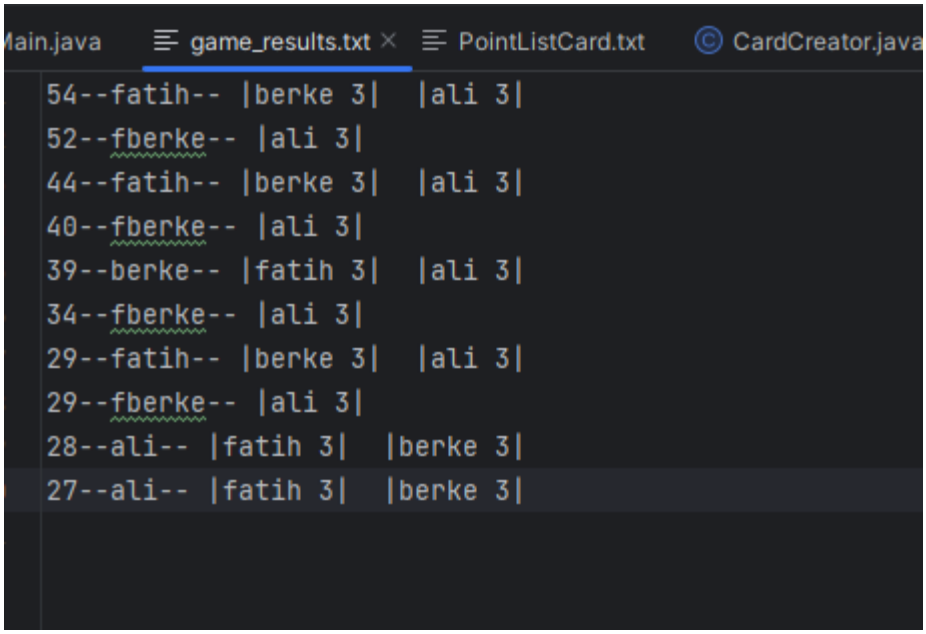
You can see just the last card which is played.

Lvl2

```
tour 1, hand 1 fafa -> 0 / fatih -> 0 / berke -> 0 / ali -> 0 /FAFA is playing
LAST CARD
♠ 4 / -2
index: 0 ♥ 5 / 1
index: 1 ♦ 2 / -4
index: 2 ♥ J / 1
index: 3 ♦ 9 / 1
pls select card index
```

When the turn comes, you can see the last card on the table.

Top 10:

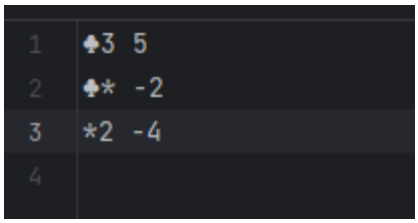


Joker Card Feature:

"J" if a player plays J card he/she can collect all cards on the table.

we put it to avoid congestion caused by negative score cards.

Card Points List:



Last Winner Collect All Cards:

```
2 usages
private Boolean lastWinner = false;
```

```
// every player's lastWinner boolean value = false
for (Player player1 : playerList){
    player1.setLastWinner(false);
}
// winner lastWinner = true
player.setLastWinner(true);
```

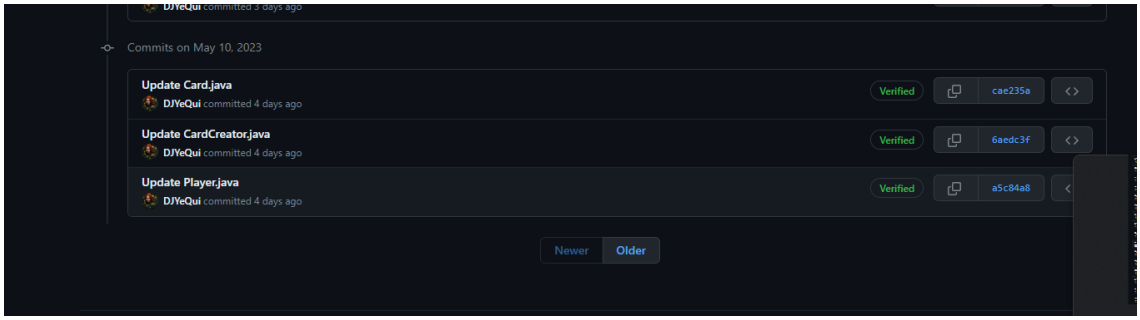
There is a boolean variable that represents its last winner information.

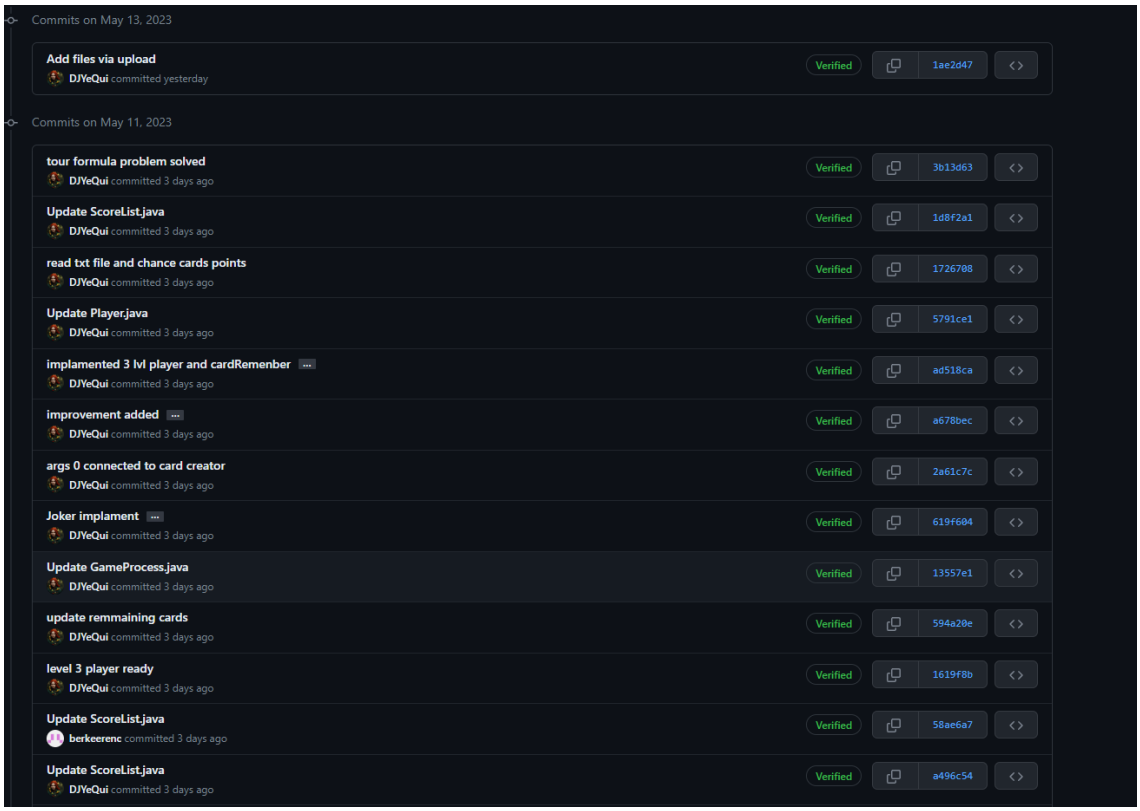
In the pointControl method if a player gains a point, we change all players' LastWinner variable value to false then , the winner player's variable set true.

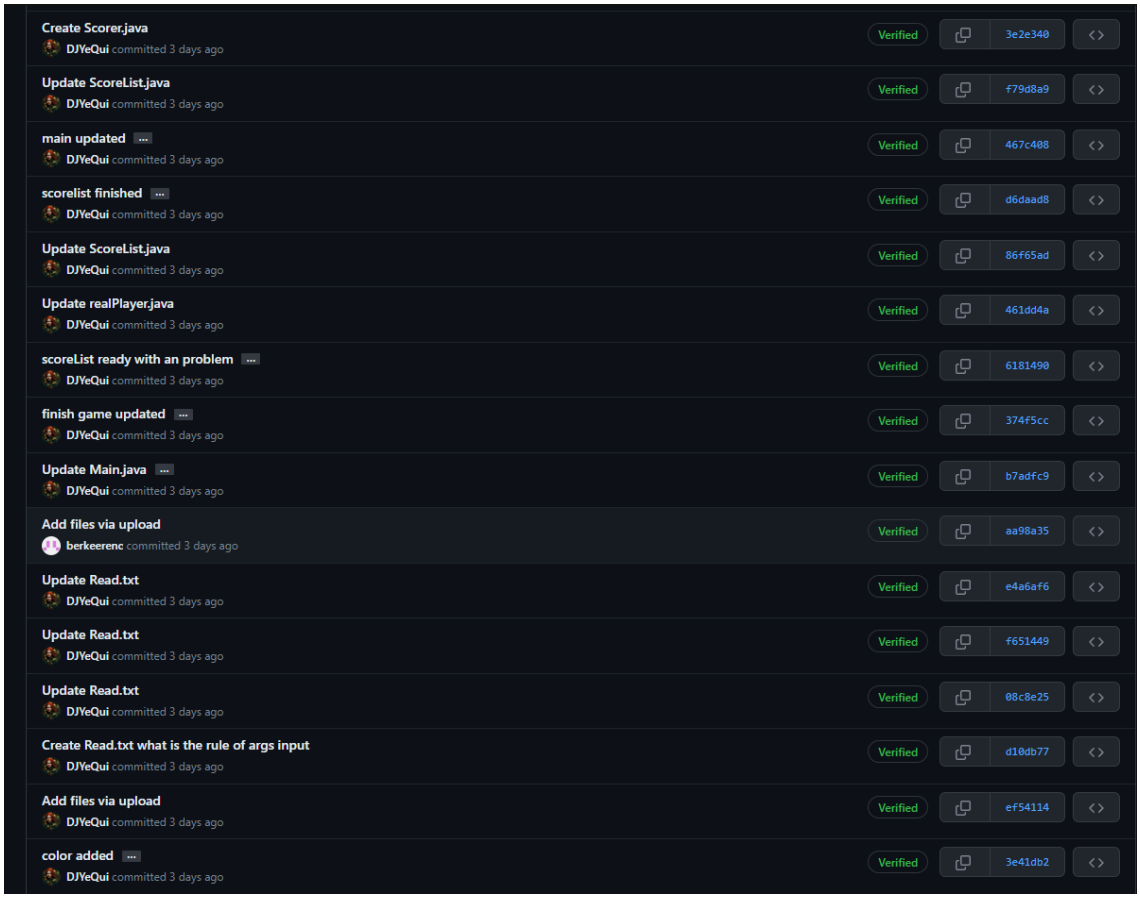
```
public Player finishGame(){
    // every player's lastWinner boolean value = false
    for (Player player1 : playerList){
        if (player1.getLastWinner() == true){
            System.out.println( player1.getName() + " is last winner and collected all cards on table");

            // set points to last winner
            for (Card card : middleCard){
                player1.setPoint(player1.getPoint() + card.getPoint());
            }

            System.out.println(ANSI_CYAN+player1.getPoint()+ " =||= " + player1.getName().toUpperCase()+ANSI_RESET);
            //clear middle
            middleCard.clear();
        }
    }
```

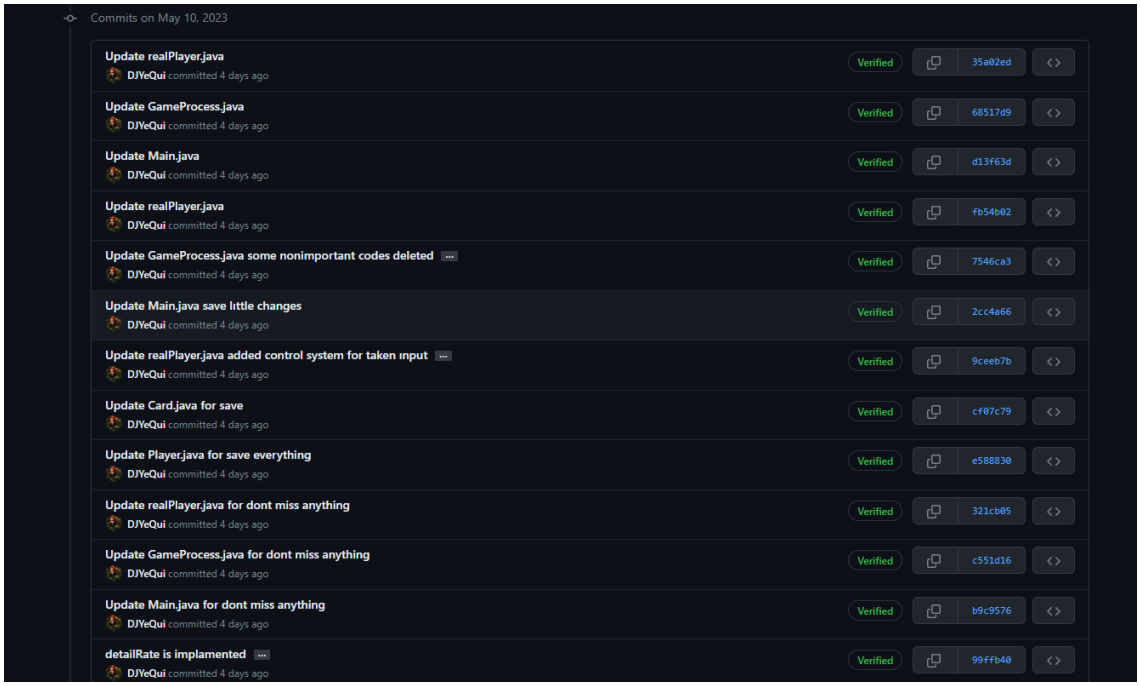
Then, in the finishGame() method, the system finds the player whose boolean variable is true. After that, all remaining cards points are calculated and players point variable updated.

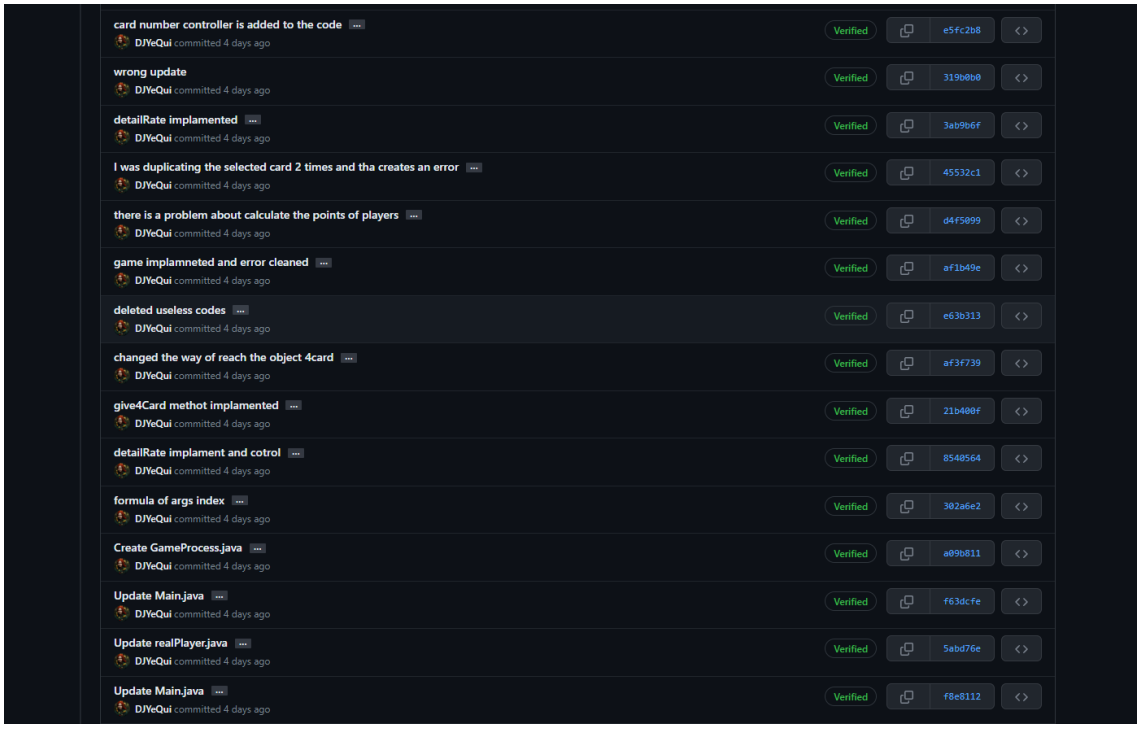
Git SS:

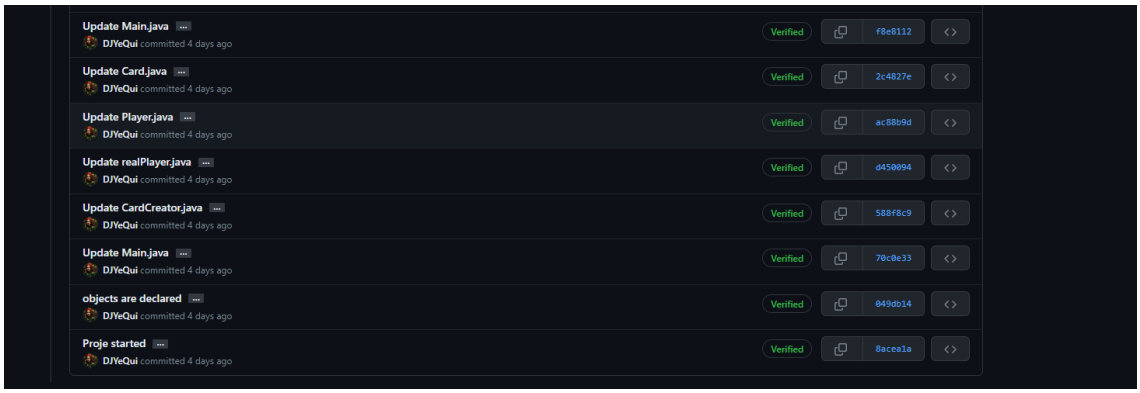

Commits on May 10, 2023
Update Card.java
DJYeQui committed 4 days ago
Verified
cae235a
Update CardCreator.java
DJYeQui committed 4 days ago
Verified
6aedc3f
Update Player.java
DJYeQui committed 4 days ago
Verified
a5c84a8
Newer
Older


Commits on May 13, 2023
Add files via upload
DJYeQui committed yesterday
Verified
1ae2d47
Commits on May 11, 2023
tour formula problem solved
DJYeQui committed 3 days ago
Verified
3b13d63
Update ScoreList.java
DJYeQui committed 3 days ago
Verified
1d8f2a1
read txt file and chance cards points
DJYeQui committed 3 days ago
Verified
1726708
Update Player.java
DJYeQui committed 3 days ago
Verified
5791ce1
implamented 3 lvl player and cardRemenber
DJYeQui committed 3 days ago
Verified
ad518ca
improvement added
DJYeQui committed 3 days ago
Verified
a678bec
args 0 connected to card creator
DJYeQui committed 3 days ago
Verified
2a61c7c
Joker implament
DJYeQui committed 3 days ago
Verified
619f604
Update GameProcess.java
DJYeQui committed 3 days ago
Verified
13557e1
update remmaining cards
DJYeQui committed 3 days ago
Verified
594a20e
level 3 player ready
DJYeQui committed 3 days ago
Verified
1619f8b
Update ScoreList.java
berkeerenc committed 3 days ago
Verified
58ae6a7
Update ScoreList.java
DJYeQui committed 3 days ago
Verified
a496c54


Create Scorer.java
DJYeQui committed 3 days ago
Verified
3e2e340
Update ScoreList.java
DJYeQui committed 3 days ago
Verified
f79d8a9
main updated ...
DJYeQui committed 3 days ago
Verified
467c408
scorelist finished ...
DJYeQui committed 3 days ago
Verified
d6daad8
Update ScoreList.java
DJYeQui committed 3 days ago
Verified
86f65ad
Update realPlayer.java
DJYeQui committed 3 days ago
Verified
461dd4a
scoreList ready with an problem ...
DJYeQui committed 3 days ago
Verified
6181490
finish game updated ...
DJYeQui committed 3 days ago
Verified
374f5cc
Update Main.java ...
DJYeQui committed 3 days ago
Verified
b7adfc9
Add files via upload
berkeerenc committed 3 days ago
Verified
aa98a35
Update Read.txt
DJYeQui committed 3 days ago
Verified
e4a6af6
Update Read.txt
DJYeQui committed 3 days ago
Verified
f651449
Update Read.txt
DJYeQui committed 3 days ago
Verified
08c8e25
Create Read.txt what is the rule of args input
DJYeQui committed 3 days ago
Verified
d10db77
Add files via upload
DJYeQui committed 3 days ago
Verified
ef54114
color added ...
DJYeQui committed 3 days ago
Verified
3e41db2


Commits on May 10, 2023
Update realPlayer.java
DJYeQui committed 4 days ago
Verified
35a02ed
Update GameProcess.java
DJYeQui committed 4 days ago
Verified
68517d9
Update Main.java
DJYeQui committed 4 days ago
Verified
d13f63d
Update realPlayer.java
DJYeQui committed 4 days ago
Verified
fb54b02
Update GameProcess.java some nonimportant codes deleted ...
DJYeQui committed 4 days ago
Verified
7546ca3
Update Main.java save little changes
DJYeQui committed 4 days ago
Verified
2cc4a66
Update realPlayer.java added control system for taken input ...
DJYeQui committed 4 days ago
Verified
9ceeb7b
Update Card.java for save
DJYeQui committed 4 days ago
Verified
cf07c79
Update Player.java for save everything
DJYeQui committed 4 days ago
Verified
e588830
Update realPlayer.java for dont miss anything
DJYeQui committed 4 days ago
Verified
321cb05
Update GameProcess.java for dont miss anything
DJYeQui committed 4 days ago
Verified
c551d16
Update Main.java for dont miss anything
DJYeQui committed 4 days ago
Verified
b9c9576
detailRate is implamented ...
DJYeQui committed 4 days ago
Verified
99ffb40


card number controller is added to the code
DJYeQui committed 4 days ago
Verified
e5fc2b8
wrong update
DJYeQui committed 4 days ago
Verified
319b0b0
detailRate implamented
DJYeQui committed 4 days ago
Verified
3ab9b6f
I was duplicating the selected card 2 times and tha creates an error
DJYeQui committed 4 days ago
Verified
45532c1
there is a problem about calculate the points of players
DJYeQui committed 4 days ago
Verified
d4f5099
game implamneted and error cleaned
DJYeQui committed 4 days ago
Verified
af1b49e
deleted useless codes
DJYeQui committed 4 days ago
Verified
e63b313
changed the way of reach the object 4card
DJYeQui committed 4 days ago
Verified
af3f739
give4Card methot implamented
DJYeQui committed 4 days ago
Verified
21b400f
detailRate implament and cotrol
DJYeQui committed 4 days ago
Verified
8540564
formula of args index
DJYeQui committed 4 days ago
Verified
302a6e2
Create GameProcess.java
DJYeQui committed 4 days ago
Verified
a09b811
Update Main.java
DJYeQui committed 4 days ago
Verified
f63dcfe
Update realPlayer.java
DJYeQui committed 4 days ago
Verified
5abd76e
Update Main.java
DJYeQui committed 4 days ago
Verified
f8e8112


Update Main.java
DJYeQui committed 4 days ago
Verified
f8e8112
Update Card.java
DJYeQui committed 4 days ago
Verified
2c4827e
Update Player.java
DJYeQui committed 4 days ago
Verified
ac88b9d
Update realPlayer.java
DJYeQui committed 4 days ago
Verified
d450094
Update CardCreator.java
DJYeQui committed 4 days ago
Verified
588f8c9
Update Main.java
DJYeQui committed 4 days ago
Verified
70c0e33
objects are declared
DJYeQui committed 4 days ago
Verified
049db14
Proje started
DJYeQui committed 4 days ago
Verified
8acea1a