INTERNATIONAL UNIVERSITY
VIETNAM NATIONAL UNIVERSITY
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Object–Oriented Programming Final Report

Semester 1, 2023- 2024

Instructor: Tran Thanh Tung

BlackJack(Xi Dach Mien Tay)

Link Github: Drissdo185/BlackJackV2 (github.com)

1/Member List:

Name	Student ID	Contribution
Do Thanh Dat	ITDSIU21079	30%
Nguyen Ba Duy	ITDSIU21014	30%
Ngo Hoang Thanh	ITDSIU21119	20%
Pham Huynh Thanh Quan	ITDSIU21110	20%

2/BlackJack Data(Code, Relate Information):

- Introduction

This project was designed based on Object – Oriented Programming method in Java language. So this has solved some problems that occurred while building by a common Procedural – Oriented Method:

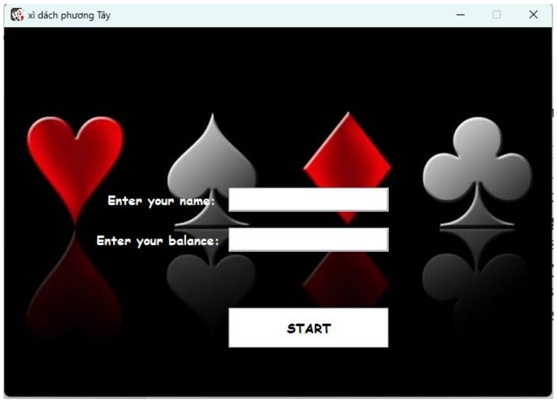
- The code is clear, easy to understand, and concise.
- The project is a unified logic system with many related - Classes combined together.
- Each Class has many Methods which take on different behaviors of their own class. The ability to reuse the resources.

The aim of this undertaking is to create a simple game rooted in object-oriented principles. Our team opted to adapt the game "Blackjack" using this methodology. The game will distinctly exhibit the object-oriented characteristics, showcasing the interplay of classes and objects. In this game, players aim to reach a card total close to or exactly 21 to emerge victorious. It's important to note that our game is a work in progress, and not all functions are currently available for demonstration.

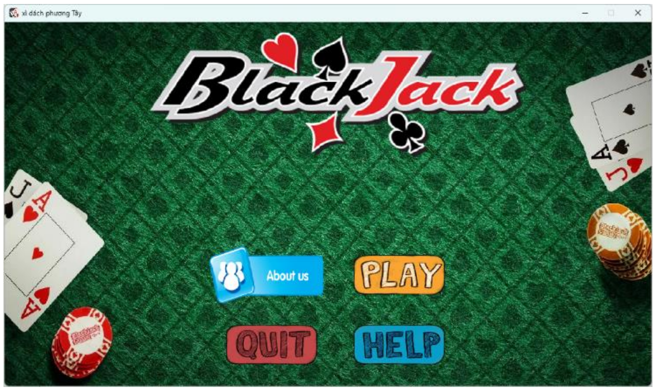
In addition to the previously outlined requirements, our intention in undertaking this project is to apply and reinforce the various techniques we've learned throughout the course. This includes hands-on experience with working with Classes, Objects, and logically implementing problem-solving through Object-Oriented Programming (OOP) methods. Despite acknowledging the potential risks and challenges associated with venturing beyond the confines of our university syllabus, we are motivated to challenge ourselves. This endeavor serves as an opportunity to familiarize ourselves with the real-world pressures we may encounter in future workplaces. In essence, our ultimate goal is to progress further along our path towards becoming proficient and advanced software developers.

- Game BlackJack:

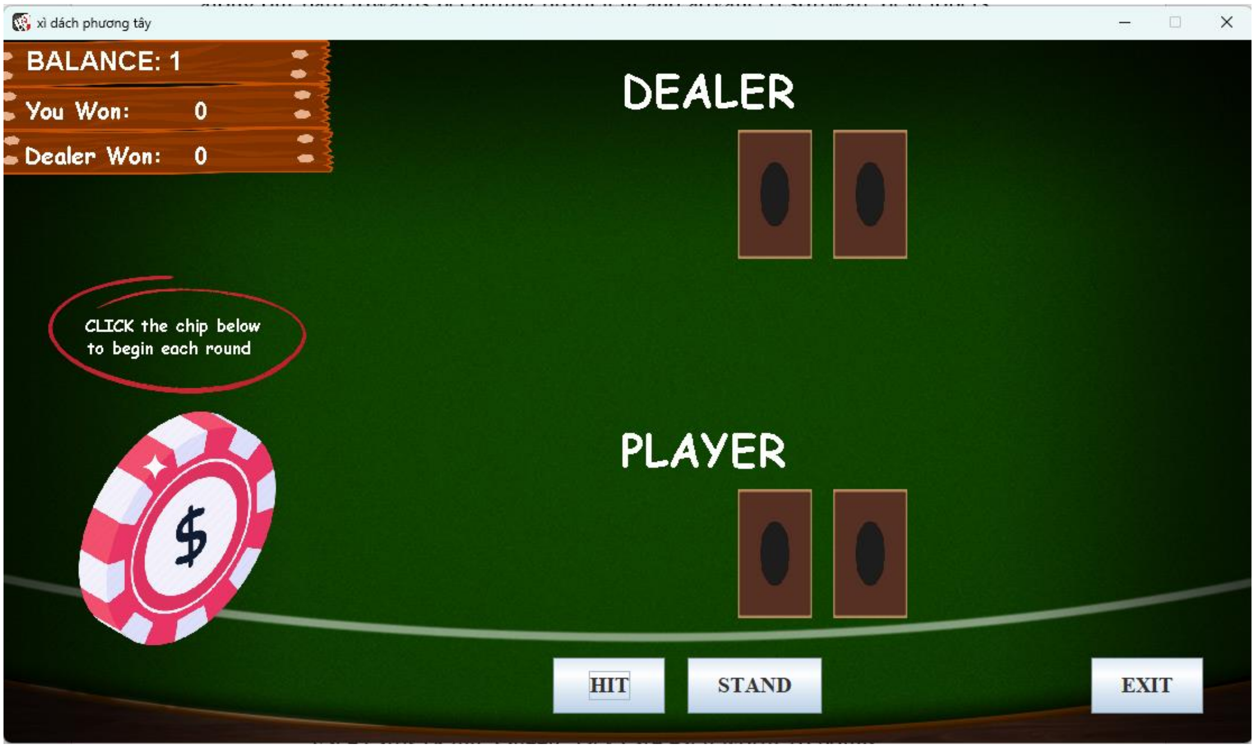
Login



Display



Play Game:



- Introduction to gameplay

1. GOAL: have a hand value closer to 21 than the dealer's hand without exceeding 21.

2. CARD VALUES:

- Face cards (King, Queen, Jack) are each worth 10 points.

- Aces can be worth either 1 or 11 points, depending on which value benefits the hand more.

3. THE DEAL:

- You are dealt two cards each, and the dealer receives one card face up and one face down (hole card).

4. TURN:

- You can choose to "hit" (take another card) or "stand" (keep your current hand).

- You can continue to hit until you decide to stand, or until your hand exceeds 21, resulting in a bust.

5. WINNING & LOSING:

- You win if your hand is closer to 21 than the dealer's hand without busting.

- If you bust (exceeds 21), the dealer wins regardless of the dealer's hand.

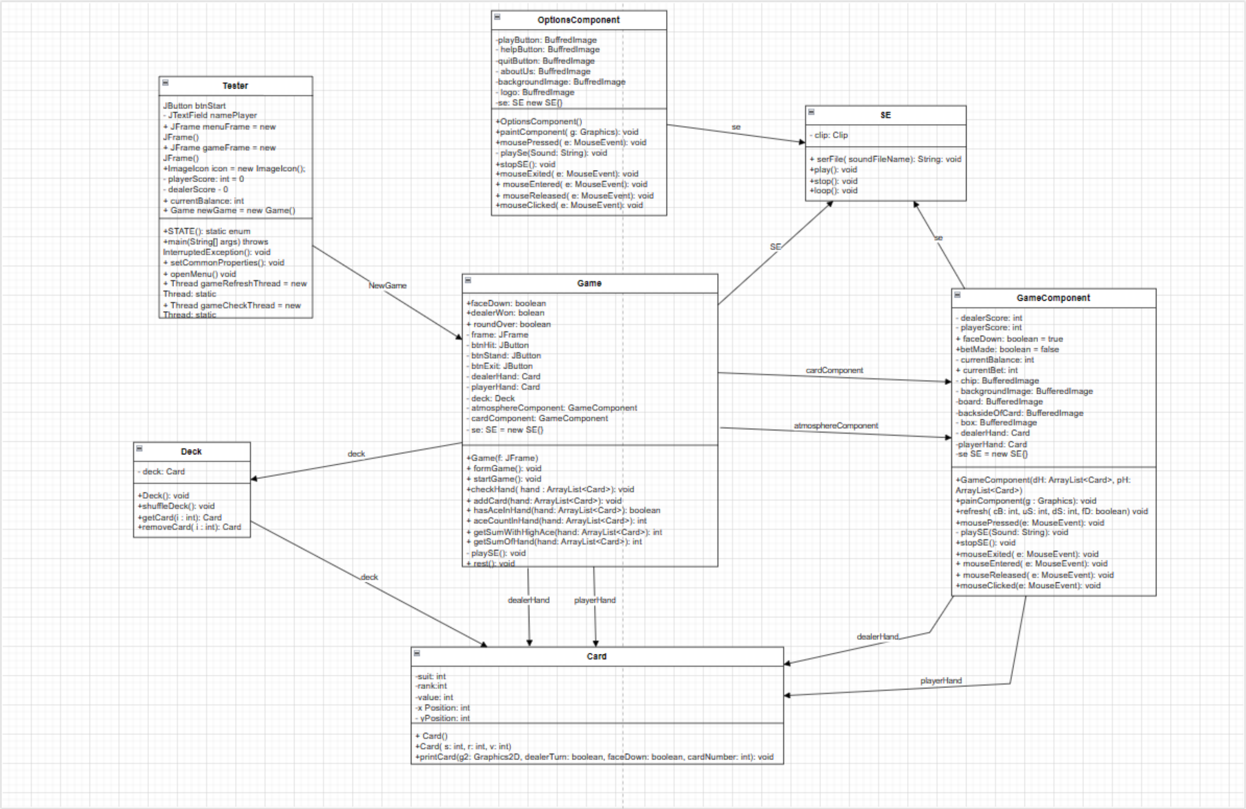
- If the dealer busts, you win.

- If both the player (you) and dealer have the same hand value, it's a push (a tie), and your bet is returned.

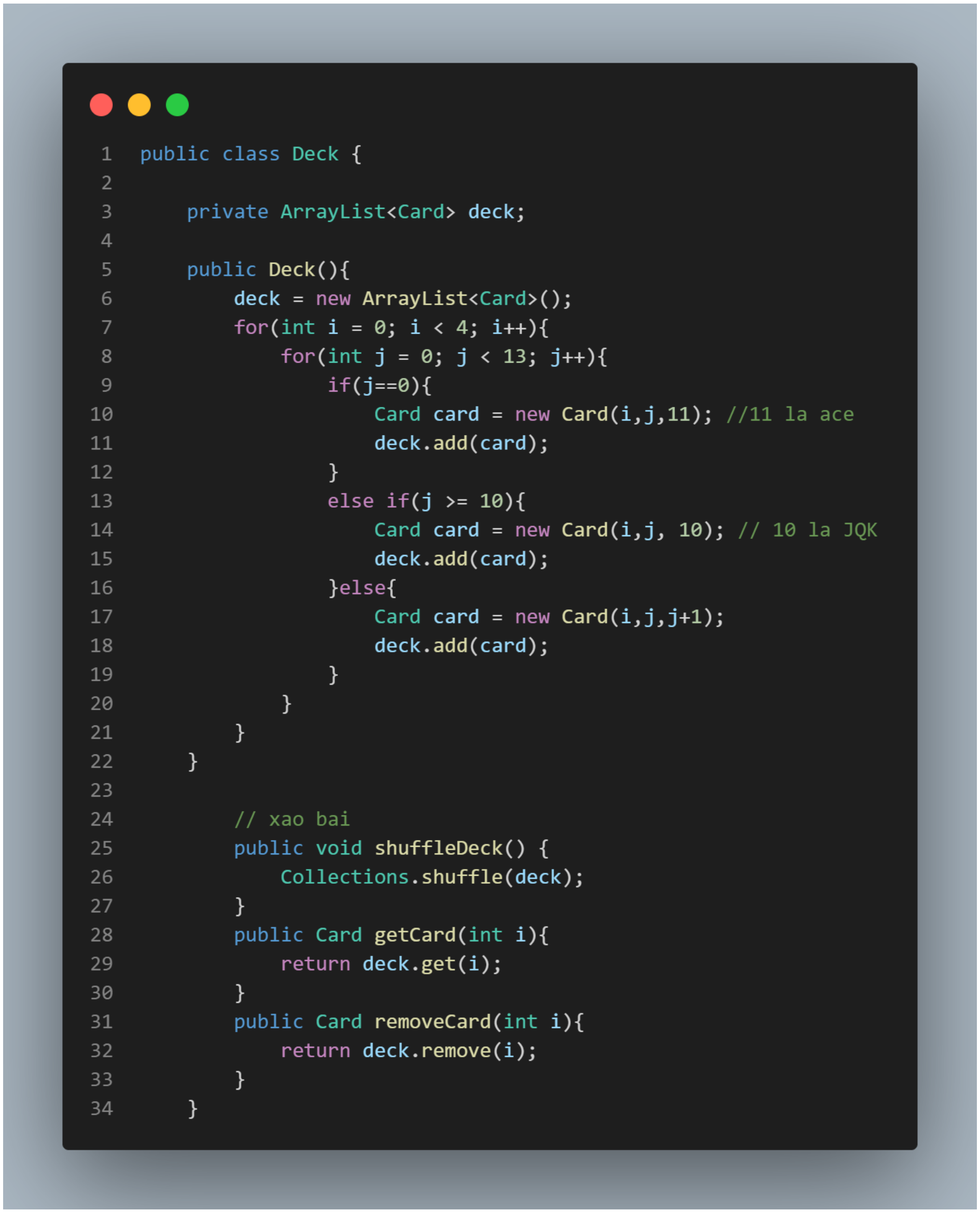
6. BLACKJACK: If the player (you) or dealer is dealt 21 from the start (Ace & a 10-value card), player/dealer got a blackjack.

- UML Diagram of Blackjack

Link Drive: https://drive.google.com/file/d/1MzeUgUj9GLihWNnLQtkQN1aSIjUvXdpw/view?usp=drive_link


OptionsComponent
-playButton: BuffredImage
- helpButton: BuffredImage
-quitButton: BuffredImage
- aboutUs: BuffredImage
-backgroundImage: BuffredImage
- logo: BuffredImage
-se: SE new SE()
+OptionsComponent()
+paintComponent(g: Graphics): void
+mousePressed(e: MouseEvent): void
- playSe(Sound: String): void
+stopSE(): void
+mouseExited(e: MouseEvent): void
+ mouseEntered(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
+mouseClicked(e: MouseEvent): void
Tester
JButton btnStart
- JTextField namePlayer
+ JFrame menuFrame = new JFrame()
+ JFrame gameFrame = new JFrame()
+ImageIcon icon = new ImageIcon();
- playerScore: int = 0
- dealerScore - 0
+ currentBalance: int
+ Game newGame = new Game()
+STATE(): static enum
+main(String[] args) throws InterruptedException(): void
+ setCommonProperties(): void
+ openMenu() void
+ Thread gameRefreshThread = new Thread: static
+ Thread gameCheckThread = new Thread: static
se
SE
- clip: Clip
+ serFile(soundFileName): String: void
+play(): void
+stop(): void
+loop(): void
SE
se
NewGame
Game
+faceDown: boolean
+dealerWon: bolean
+ roundOver: boolean
- frame: JFrame
- btnHit: JButton
- btnStand: JButton
- btnExit: JButton
- dealerHand: Card
- playerHand: Card
- deck: Deck
- atmosphereComponent: GameComponent
- cardComponent: GameComponent
- se: SE = new SE()
+Game(f: JFrame)
+ formGame(): void
+ startGame(): void
+checkHand(hand : ArrayList<Card>): void
+ addCard(hand: ArrayList<Card>): void
+ hasAceInHand(hand: ArrayList<Card>): boolean
+ aceCountInHand(hand: ArrayList<Card>): int
+ getSumWithHighAce(hand: ArrayList<Card>): int
+ getSumOfHand(hand: ArrayList<Card>): int
- playSE(): void
+ rest(): void
GameComponent
- dealerScore: int
- playerScore: int
+ faceDown: boolean = true
+betMade: boolean = false
- currentBalance: int
+ currentBet: int
- chip: BufferedImage
- backgroundImage: BufferedImage
-board: BufferedImage
-backsideOfCard: BufferedImage
- box: BufferedImage
- dealerHand: Card
-playerHand: Card
-se SE = new SE()
+GameComponent(dH: ArrayList<Card>, pH: ArrayList<Card>)
+painComponent(g : Graphics): void
+refresh(cB: int, uS: int, dS: int, fD: boolean) void
+mousePressed(e: MouseEvent): void
- playSE(Sound: String): void
+stopSE(): void
+mouseExited(e: MouseEvent): void
+ mouseEntered(e: MouseEvent): void
+ mouseReleased(e: MouseEvent): void
+mouseClicked(e: MouseEvent): void
cardComponent
atmosphereComponent
Deck
- deck: Card
+Deck(): void
+shuffleDeck(): void
+getCard(i : int): Card
+removeCard(i : int): Card
deck
deck
dealerHand
playerHand
dealerHand
playerHand
Card
-suit: int
-rank:int
-value: int
-x Position: int
- yPosition: int
+ Card()
+Card(s: int, r: int, v: int)
+printCard(g2: Graphics2D, dealerTurn: boolean, faceDown: boolean, cardNumber: int): void

- **Deck class**



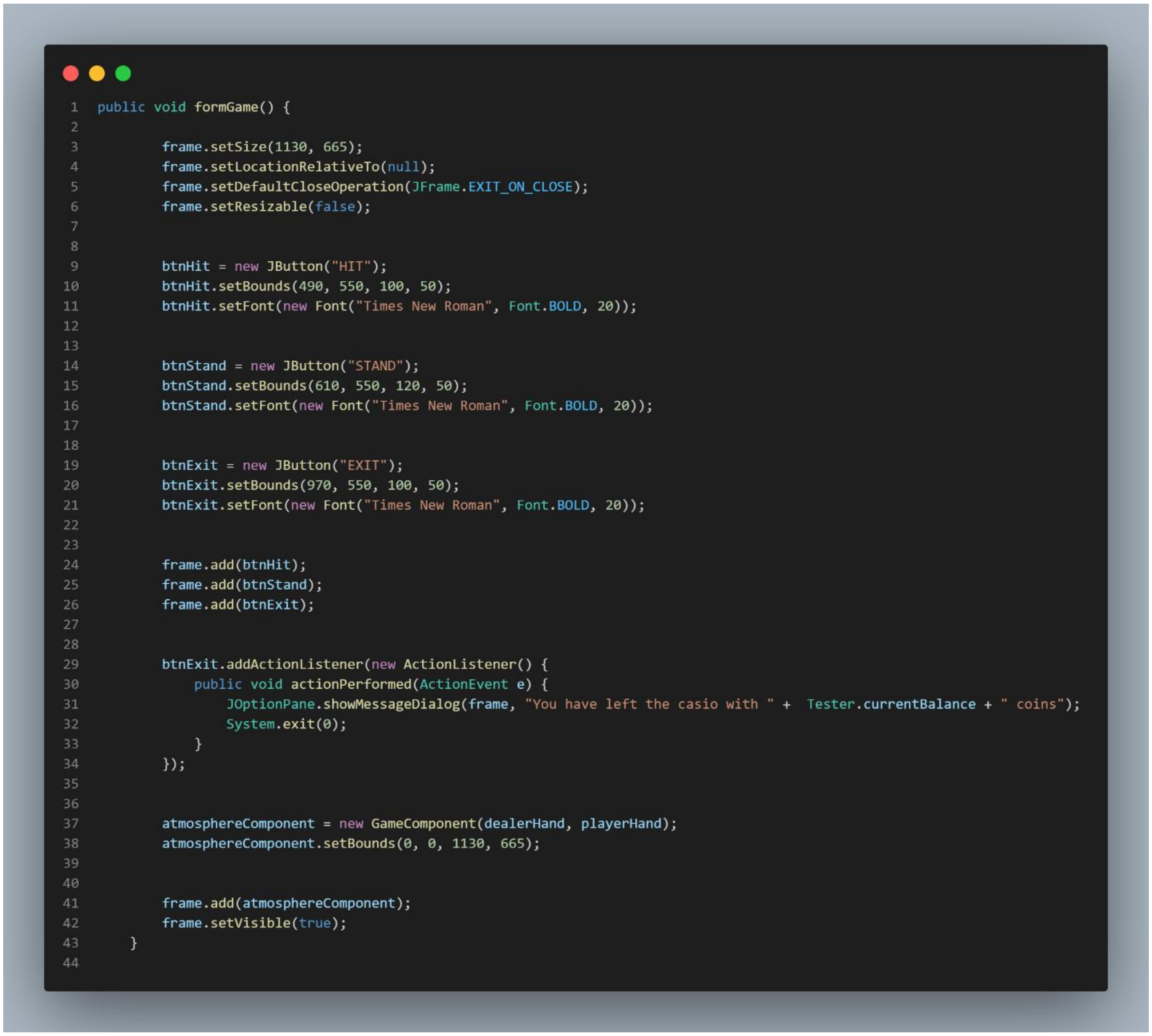
```java
public class Deck {

    private ArrayList<Card> deck;

    public Deck(){
        deck = new ArrayList<Card>();
        for(int i = 0; i < 4; i++){
            for(int j = 0; j < 13; j++){
                if(j==0){
                    Card card = new Card(i,j,11); //11 la ace
                    deck.add(card);
                }
                else if(j >= 10){
                    Card card = new Card(i,j, 10); // 10 la JQK
                    deck.add(card);
                }else{
                    Card card = new Card(i,j,j+1);
                    deck.add(card);
                }
            }
        }
    }

        // xao bai
        public void shuffleDeck() {
            Collections.shuffle(deck);
        }
        public Card getCard(int i){
            return deck.get(i);
        }
        public Card removeCard(int i){
            return deck.remove(i);
        }
    }
```

The Deck class is a Java implementation that represents a standard deck of playing cards. It contains a private ArrayList named deck, which stores instances of a Card class. The constructor initializes the deck by populating it with 52 cards, organized into four suits and thirteen ranks. Special considerations are made for aces, assigned a value of 11, and face cards (Jack, Queen, King), each assigned a value of 10. The class provides methods for shuffling the deck (shuffleDeck), retrieving a specific card at an index (getCard), and removing a card from the deck at a specified index (removeCard). The code exemplifies basic functionalities necessary for card games, making it a foundational component for developers creating applications involving card-based gameplay.

- **Game class**

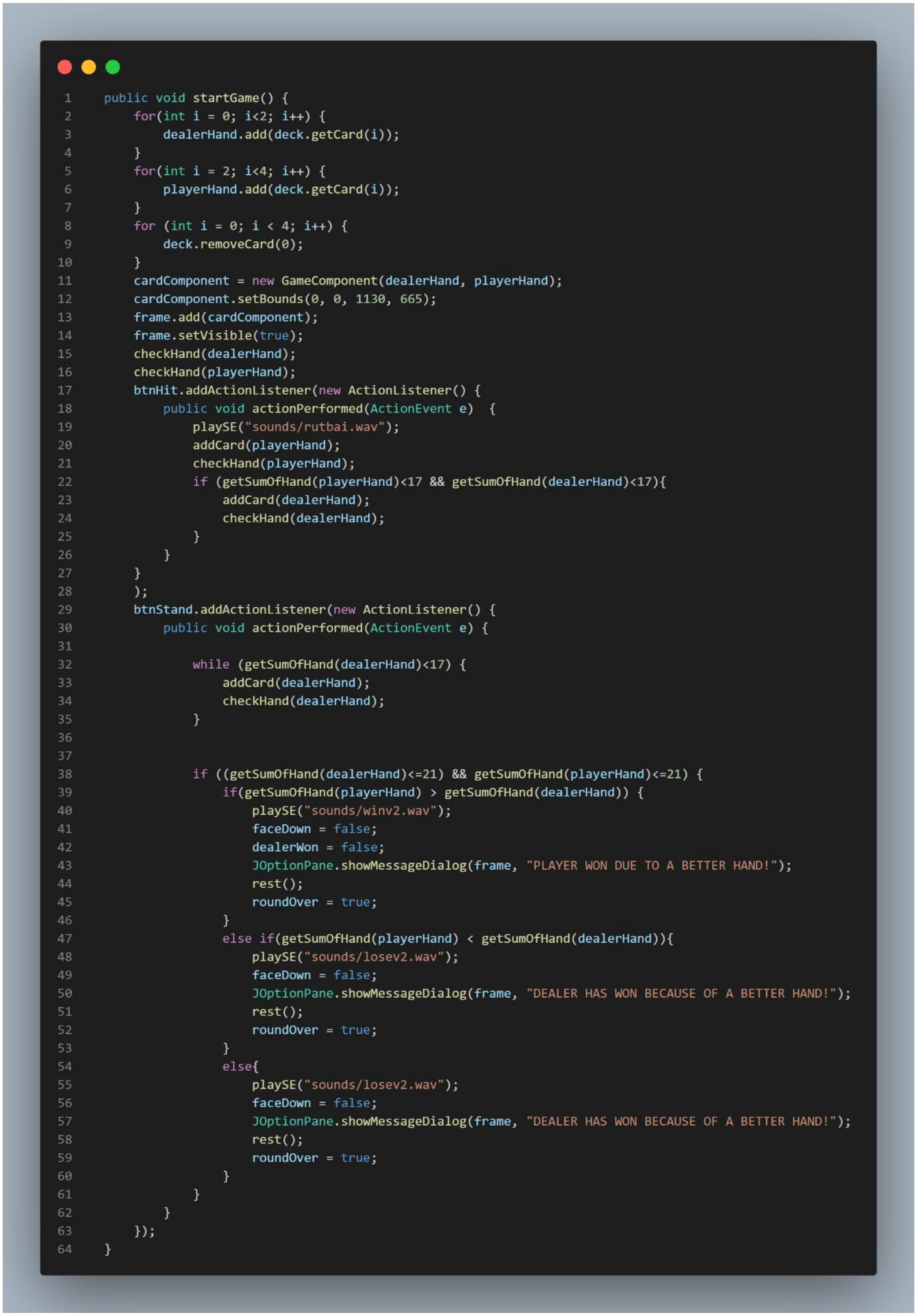
- The constructor initializes key components of the game, including the deck, dealer's hand, player's hand, and a graphical representation (GameComponent) for the game atmosphere. It sets initial game states such as whether cards are face-down, if the dealer has won, and if the round is over.
- The formGame method configures the graphical user interface (GUI) of the game window, adding buttons for "HIT," "STAND," and "EXIT." It also includes event listeners for the "EXIT" button, allowing the player to leave

the game with a message displaying their remaining balance.



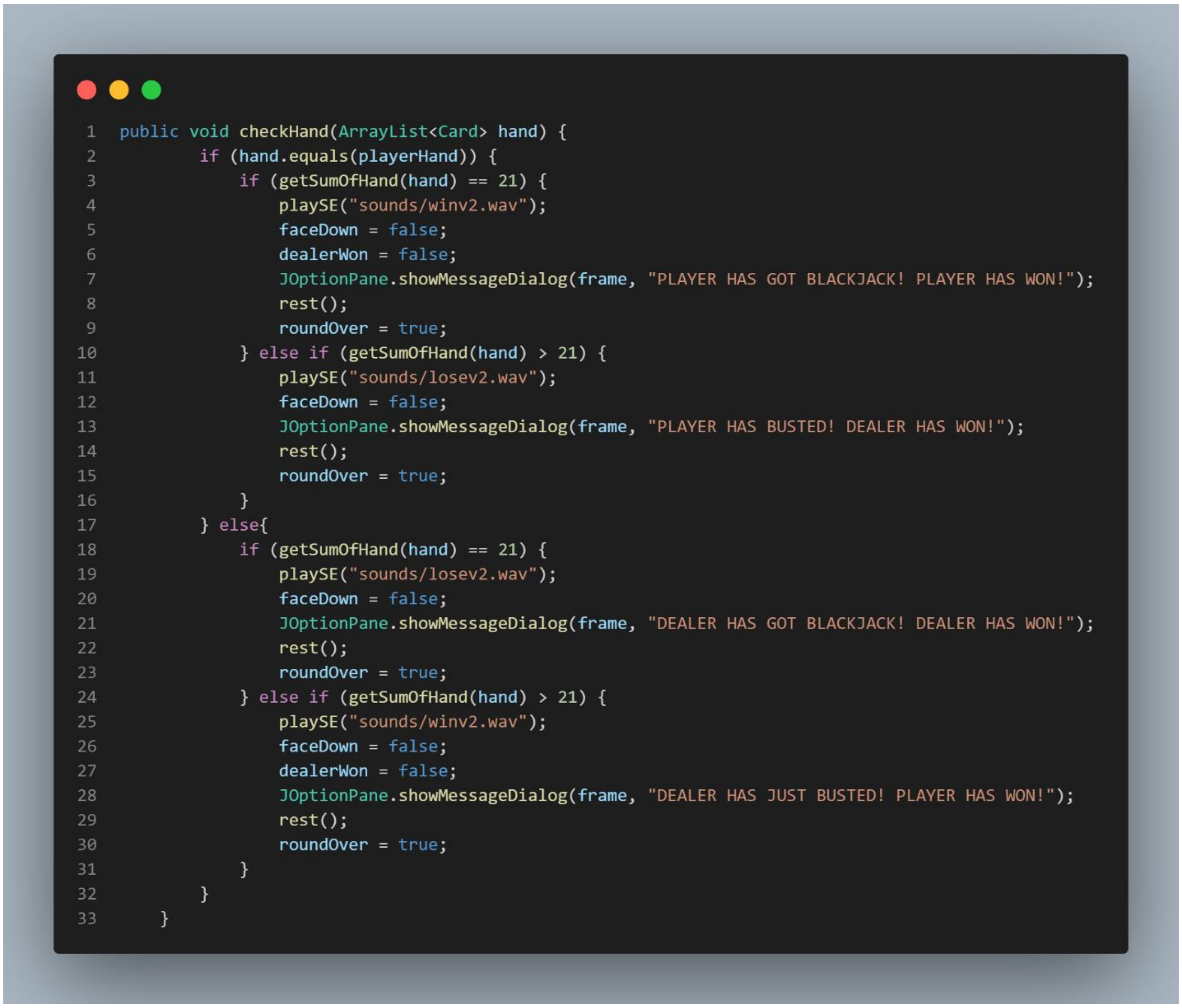
```java
public void formGame() {

        frame.setSize(1130, 665);
        frame.setLocationRelativeTo(null);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setResizable(false);


        btnHit = new JButton("HIT");
        btnHit.setBounds(490, 550, 100, 50);
        btnHit.setFont(new Font("Times New Roman", Font.BOLD, 20));


        btnStand = new JButton("STAND");
        btnStand.setBounds(610, 550, 120, 50);
        btnStand.setFont(new Font("Times New Roman", Font.BOLD, 20));


        btnExit = new JButton("EXIT");
        btnExit.setBounds(970, 550, 100, 50);
        btnExit.setFont(new Font("Times New Roman", Font.BOLD, 20));


        frame.add(btnHit);
        frame.add(btnStand);
        frame.add(btnExit);


        btnExit.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(frame, "You have left the casio with " +  Tester.currentBalance + " coins");
                System.exit(0);
            }
        });


        atmosphereComponent = new GameComponent(dealerHand, playerHand);
        atmosphereComponent.setBounds(0, 0, 1130, 665);


        frame.add(atmosphereComponent);
        frame.setVisible(true);
    }

```

- The startGame method initiates the game by dealing two cards to both the dealer and the player, updating the GUI accordingly. It also sets up event listeners for the "HIT" and "STAND" buttons, allowing the player to draw additional cards or end their turn.



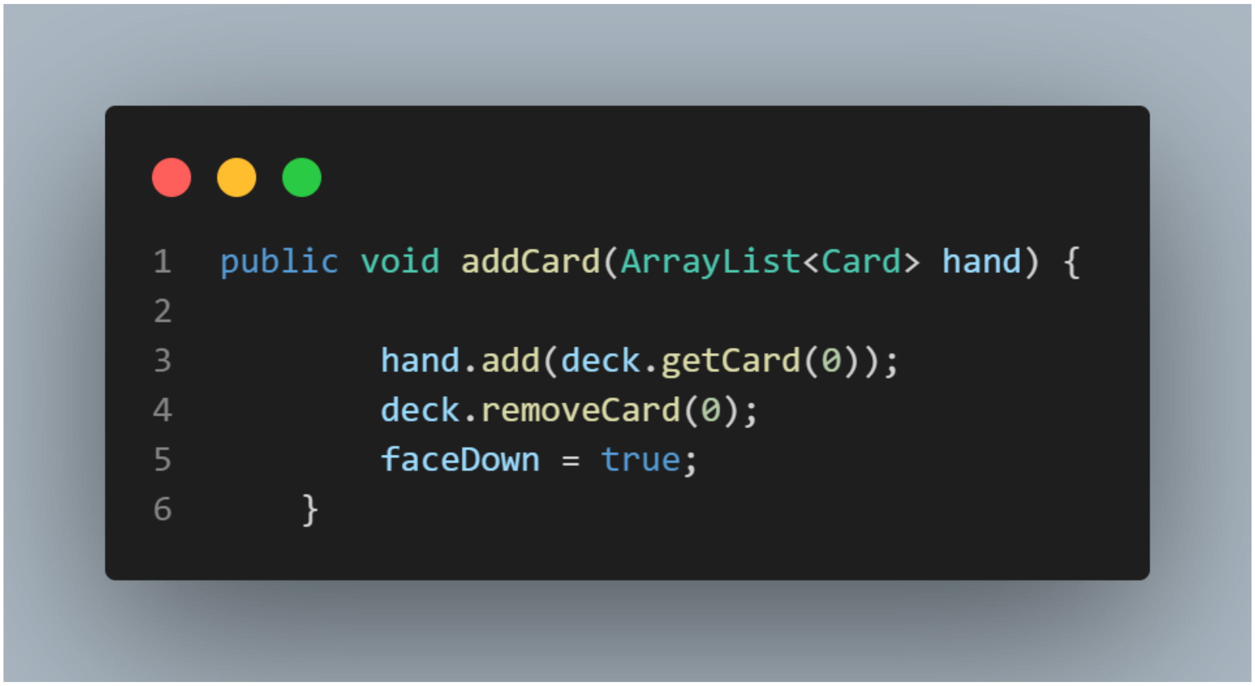
```
public void startGame() {
    for(int i = 0; i<2; i++) {
        dealerHand.add(deck.getCard(i));
    }
    for(int i = 2; i<4; i++) {
        playerHand.add(deck.getCard(i));
    }
    for (int i = 0; i < 4; i++) {
        deck.removeCard(0);
    }
    cardComponent = new GameComponent(dealerHand, playerHand);
    cardComponent.setBounds(0, 0, 1130, 665);
    frame.add(cardComponent);
    frame.setVisible(true);
    checkHand(dealerHand);
    checkHand(playerHand);
    btnHit.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e)  {
            playSE("sounds/rutbai.wav");
            addCard(playerHand);
            checkHand(playerHand);
            if (getSumOfHand(playerHand)<17 && getSumOfHand(dealerHand)<17){
                addCard(dealerHand);
                checkHand(dealerHand);
            }
        }
    }
    );
    btnStand.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {

            while (getSumOfHand(dealerHand)<17) {
                addCard(dealerHand);
                checkHand(dealerHand);
            }


            if ((getSumOfHand(dealerHand)<=21) && getSumOfHand(playerHand)<=21) {
                if(getSumOfHand(playerHand) > getSumOfHand(dealerHand)) {
                    playSE("sounds/winv2.wav");
                    faceDown = false;
                    dealerWon = false;
                    JOptionPane.showMessageDialog(frame, "PLAYER WON DUE TO A BETTER HAND!");
                    rest();
                    roundOver = true;
                }
                else if(getSumOfHand(playerHand) < getSumOfHand(dealerHand)){
                    playSE("sounds/losev2.wav");
                    faceDown = false;
                    JOptionPane.showMessageDialog(frame, "DEALER HAS WON BECAUSE OF A BETTER HAND!");
                    rest();
                    roundOver = true;
                }
                else{
                    playSE("sounds/losev2.wav");
                    faceDown = false;
                    JOptionPane.showMessageDialog(frame, "DEALER HAS WON BECAUSE OF A BETTER HAND!");
                    rest();
                    roundOver = true;
                }
            }
        }
    });
}
```

- The checkHand method evaluates the current state of a hand (either the player's or dealer's), considering conditions such as achieving a blackjack, busting, or winning based on the hand's total value.



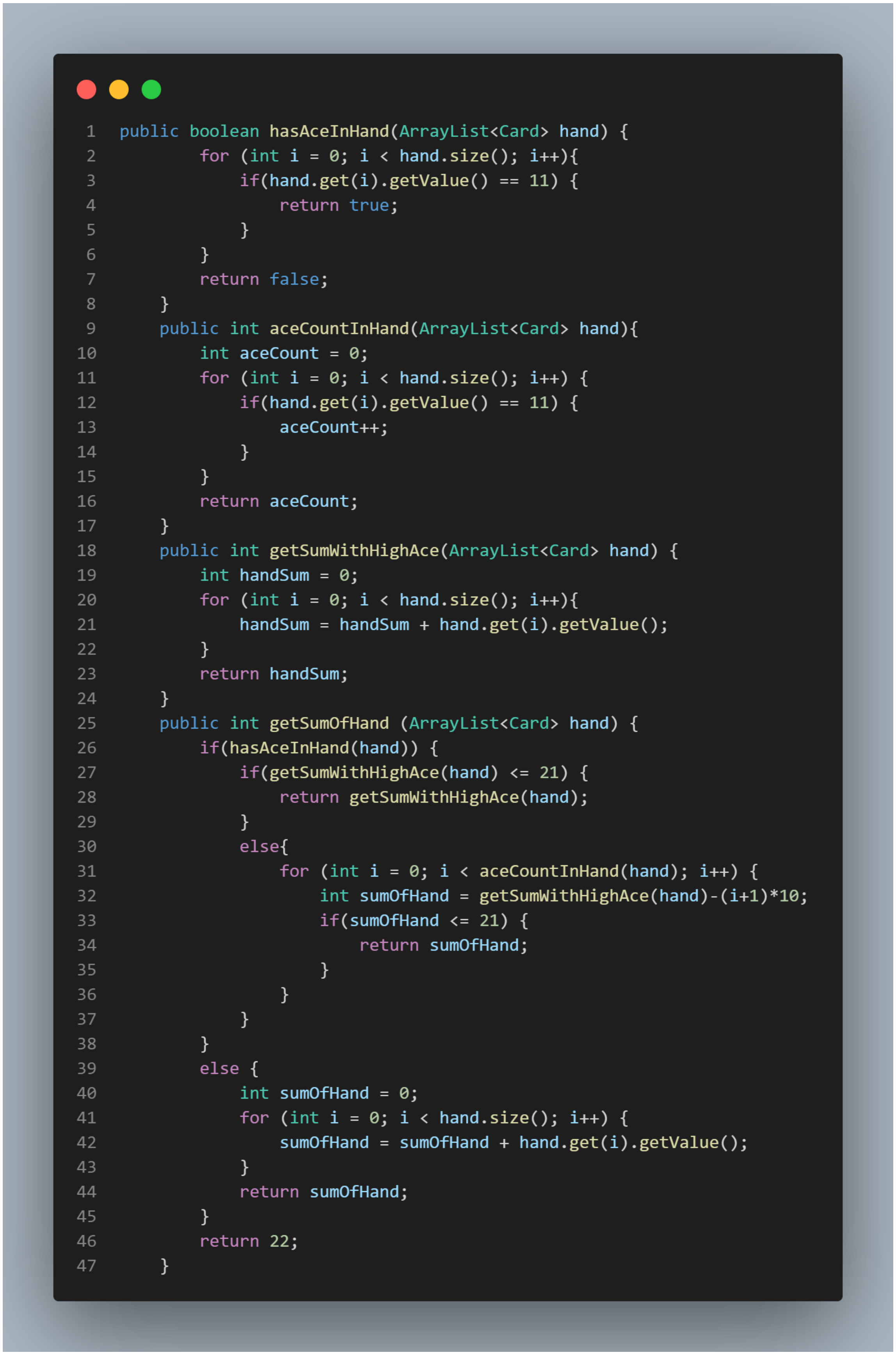
```java
public void checkHand(ArrayList<Card> hand) {
        if (hand.equals(playerHand)) {
            if (getSumOfHand(hand) == 21) {
                playSE("sounds/winv2.wav");
                faceDown = false;
                dealerWon = false;
                JOptionPane.showMessageDialog(frame, "PLAYER HAS GOT BLACKJACK! PLAYER HAS WON!");
                rest();
                roundOver = true;
            } else if (getSumOfHand(hand) > 21) {
                playSE("sounds/losev2.wav");
                faceDown = false;
                JOptionPane.showMessageDialog(frame, "PLAYER HAS BUSTED! DEALER HAS WON!");
                rest();
                roundOver = true;
            }
        } else{
            if (getSumOfHand(hand) == 21) {
                playSE("sounds/losev2.wav");
                faceDown = false;
                JOptionPane.showMessageDialog(frame, "DEALER HAS GOT BLACKJACK! DEALER HAS WON!");
                rest();
                roundOver = true;
            } else if (getSumOfHand(hand) > 21) {
                playSE("sounds/winv2.wav");
                faceDown = false;
                dealerWon = false;
                JOptionPane.showMessageDialog(frame, "DEALER HAS JUST BUSTED! PLAYER HAS WON!");
                rest();
                roundOver = true;
            }
        }
    }
```

- The addCard method adds a card to a specified hand, updating the GUI accordingly.



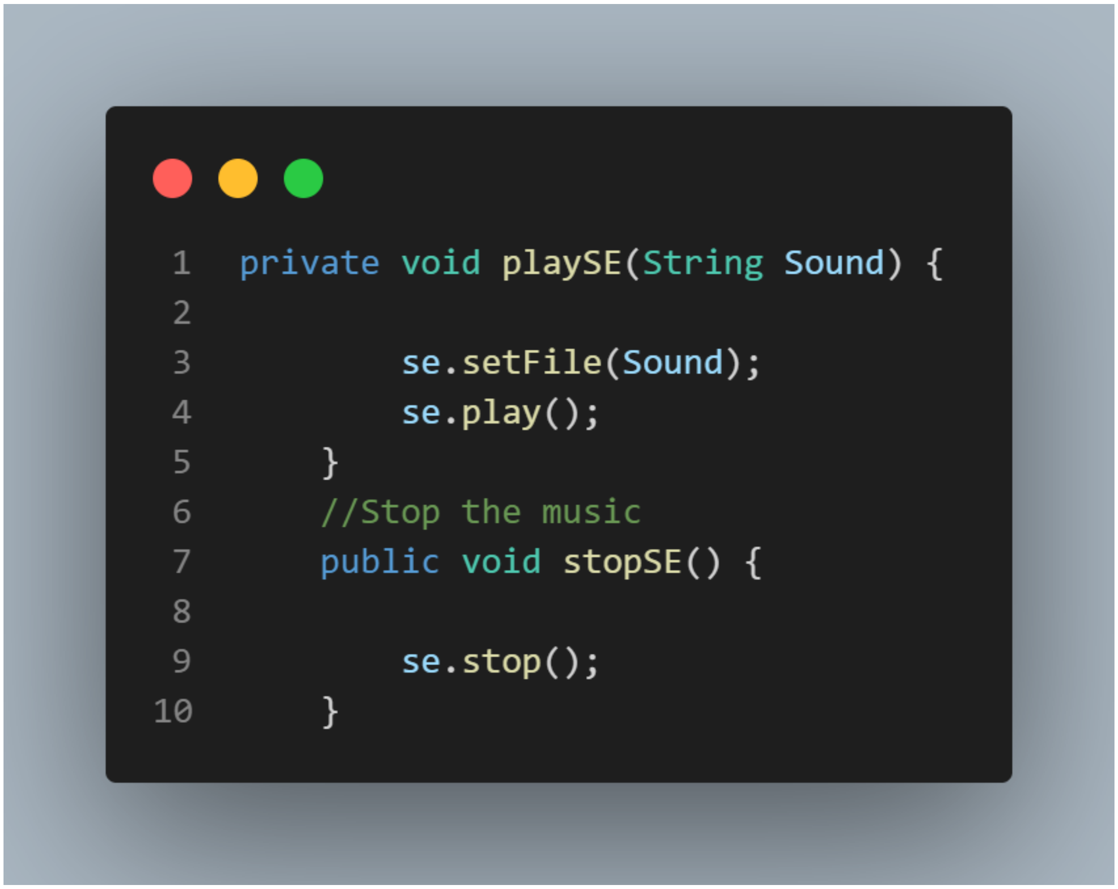
```java
public void addCard(ArrayList<Card> hand) {

        hand.add(deck.getCard(0));
        deck.removeCard(0);
        faceDown = true;
    }
```

- Several utility methods, such as hasAceInHand, aceCountInHand, getSumWithHighAce, and getSumOfHand, are implemented to handle scenarios involving aces and calculate the total value of a hand.



```java
public boolean hasAceInHand(ArrayList<Card> hand) {
        for (int i = 0; i < hand.size(); i++){
            if(hand.get(i).getValue() == 11) {
                return true;
            }
        }
        return false;
    }
    public int aceCountInHand(ArrayList<Card> hand){
        int aceCount = 0;
        for (int i = 0; i < hand.size(); i++) {
            if(hand.get(i).getValue() == 11) {
                aceCount++;
            }
        }
        return aceCount;
    }
    public int getSumWithHighAce(ArrayList<Card> hand) {
        int handSum = 0;
        for (int i = 0; i < hand.size(); i++){
            handSum = handSum + hand.get(i).getValue();
        }
        return handSum;
    }
    public int getSumOfHand (ArrayList<Card> hand) {
        if(hasAceInHand(hand)) {
            if(getSumWithHighAce(hand) <= 21) {
                return getSumWithHighAce(hand);
            }
            else{
                for (int i = 0; i < aceCountInHand(hand); i++) {
                    int sumOfHand = getSumWithHighAce(hand)-(i+1)*10;
                    if(sumOfHand <= 21) {
                        return sumOfHand;
                    }
                }
            }
        }
        else {
            int sumOfHand = 0;
            for (int i = 0; i < hand.size(); i++) {
                sumOfHand = sumOfHand + hand.get(i).getValue();
            }
            return sumOfHand;
        }
        return 22;
    }
```

- The class also includes methods for playing sound effects (playSE) and stopping sound (stopSE).



```
private void playSE(String Sound) {

        se.setFile(Sound);
        se.play();
    }
    //Stop the music
    public void stopSE() {

        se.stop();
    }
```

-
- Notably, the Game class appears to be part of a larger Blackjack game implementation, featuring GUI interactions, sound effects, and logic for managing the game flow.

- **GameComponent class**

The GameComponent class extends JComponent and implements the MouseListener interface in Java Swing, serving as a graphical component for rendering and interacting with the user interface of a card game, likely Blackjack. Here's a summary of its key features:

Fields:

- *dealerHand and playerHand*: ArrayLists representing the dealer's and player's hands, respectively.

- *dealerScore and playerScore*: Integers representing the scores of the dealer and player.
- *faceDown*: A boolean indicating whether the dealer's first card should be face-down.
- *betMade*: A static boolean tracking whether a bet has been made.
- *currentBalance* and currentBet: Integers representing the current balance and bet amount.
- *se*: An instance of the SE class for playing sound effects.
- *Static BufferedImages* for various graphical elements like chips, background, cards, and more.
- *Constructor*: Initializes the dealer's and player's hands and sets initial values for scores and other variables. It also adds a MouseListener to handle mouse events.
- *paintComponent Method*: Overrides the paintComponent method to customize the rendering of graphical elements on the component. Draws background images, cards, scores, and other UI elements.
- *refresh Method*: Updates the component with new values for the current balance, player score, dealer score, and the face-down status. Repaints the component to reflect these changes.
- *mousePressed Method*:
    - Implements the MouseListener interface to handle mouse press events.
    - Checks if the player clicks on the chip area to initiate the betting process. Displays a dialog for the player to choose a betting amount, and updates the balance and current bet accordingly. If the bet is valid, it triggers the start of the game.
- *playSE and stopSE Methods*: Utilizes the SE class to play and stop sound effects.

- *mouseExited, mouseEntered, mouseReleased, mouseClicked Methods*: Placeholder methods for the MouseListener interface.

- **OptionsComponents class**

The OptionsComponent class extends JComponent and implements the MouseListener interface in Java Swing. It is responsible for rendering the graphical user interface (GUI) elements of the options menu for a card game, possibly Blackjack. Here's a brief overview:

- *Fields*:

+ Static BufferedImages for various graphical elements such as playButton, helpButton, quitButton, aboutUs, backgroundImage, and logo.

+ An instance of the SE class for playing sound effects.

- *Constructor*:

Initializes the MouseListener interface to handle mouse events.

- *paintComponent Method*:

Overrides the paintComponent method to customize the rendering of graphical elements on the component. Draws background images, buttons, and the game logo.

- *mousePressed Method*:

+ Implements the MouseListener interface to handle mouse press events.

+ Checks for mouse clicks on different buttons (play, help, quit, about us).

+ Performs actions accordingly, such as starting the game, displaying help information, exiting the program, or showing information about the developers.

- *playSE and stopSE Methods*: Utilizes the SE class to play and stop sound effects.
- *mouseExited, mouseEntered, mouseReleased, mouseClicked Methods*: Placeholder methods for the MouseListener interface.

The class encapsulates the visual and interactive elements of the options menu, providing a user-friendly interface for players to navigate through different options such as starting the game, accessing help information, and learning more about the developers.

3/ Conclusion:

The Blackjack game is constructed using object-oriented programming, providing a transparent demonstration of OOP principles such as polymorphism, inheritance, encapsulation, and abstraction. The tight and systematic linkage between classes and objects is evident, showcasing the effective application of OOP concepts. Additionally, undertaking this project offers the valuable opportunity to extend our learning beyond the course boundaries, acquiring knowledge that goes beyond the predefined curriculum. This expansion of knowledge is recognized as a crucial aspect of our project implementation.