Student Information:	Surname:	Given name:	Student ID:
1	Limawan	Pandya	2502022433

Course Code : COMP6699001 **Course Name** : Object Oriented Programming

Class : L2CC **Name of Lecturer(s)** : Jude Joseph Lamug Martinez

Major : Computer Science

Title of Assignment : BlackJack using JavaFX

Type of Assignment : Final Project

Submission Pattern

Due Date : 20th June 2022 **Submission Date** : 20th June 2022

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.

5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: (Name of Student)

Pandya Limawan

“BlackJack using JavaFX”

Name : Pandya Limawan

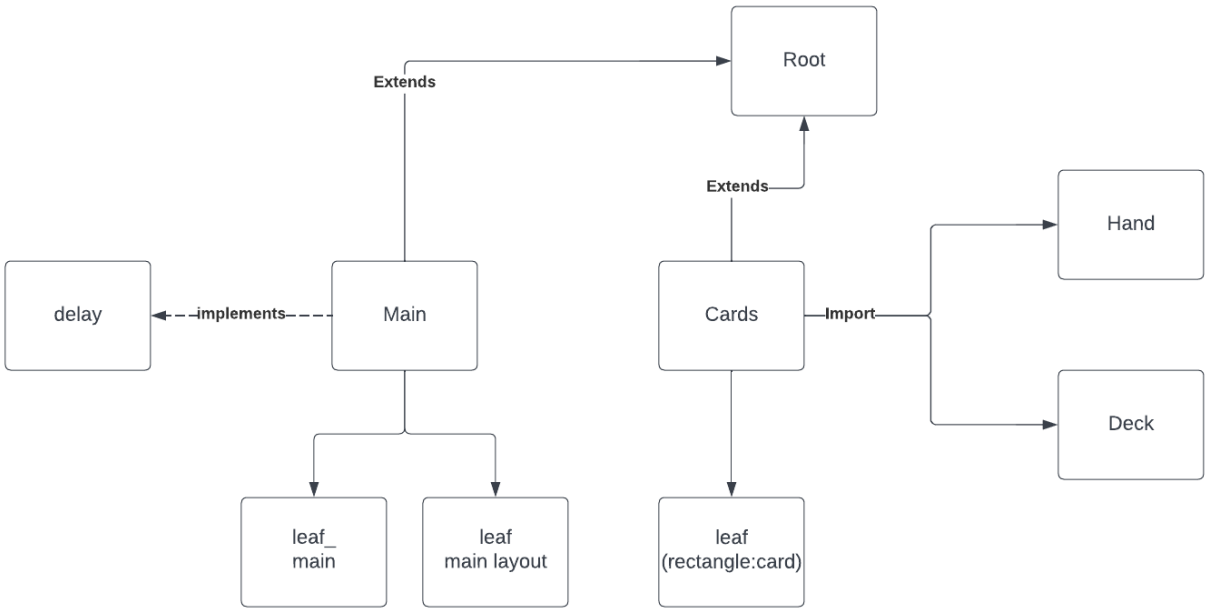
ID : 2502022433

I. Program Description

This is a simple BlackJack game running on Java using the JavaFX library. The program is written using IntelliJ IDE. This program is inspired by YouTube user Almas Baimagambetov and is being re-written and edited by me.

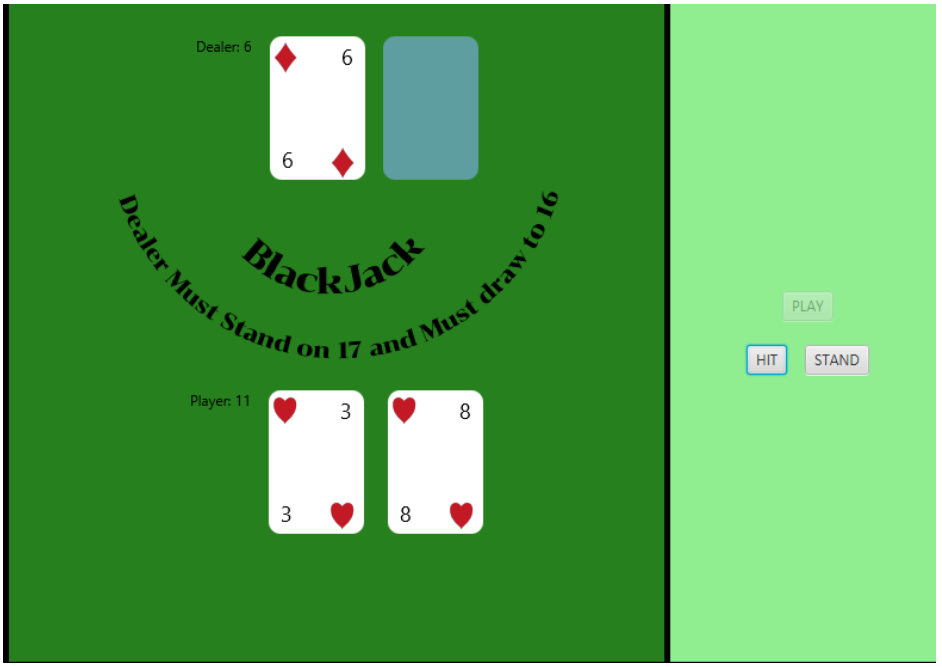
As a game lover, BlackJack is always a very simple yet fun card game. This game is very easy to play, just hit or stand and the closest to 21 wins. However, if you surpass 21 you’ll lose.

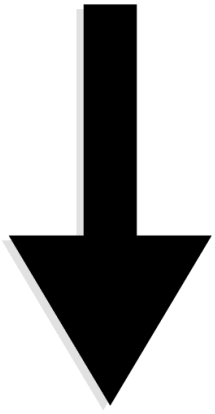
II. Class Diagram



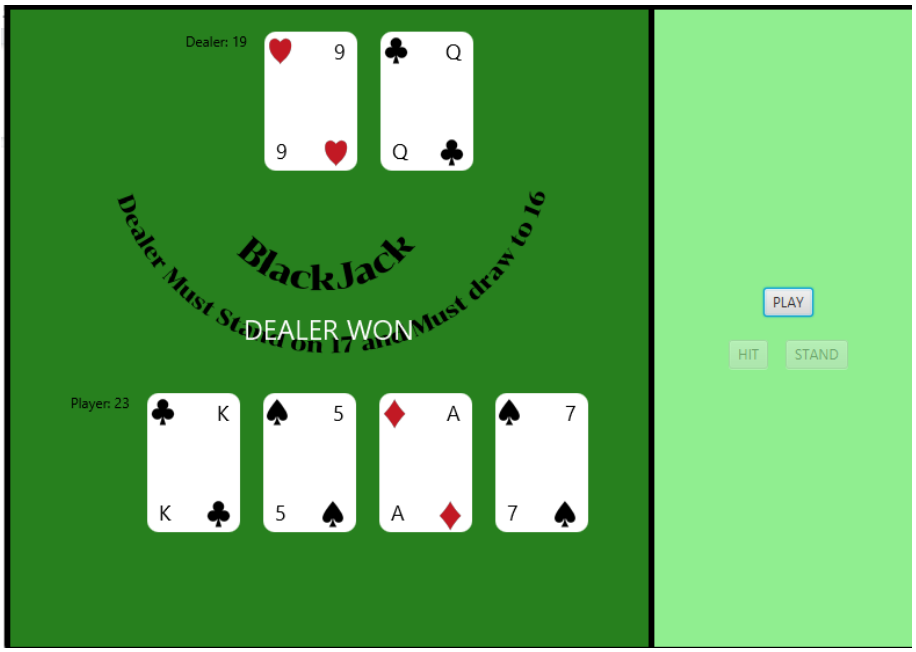
III. Application Flow

Click Play





Hit or Stand



The winner is shown

IV. Lessons that Have Been Learned

In this project, I have learned quite a few things. I got new experience in making a Java program. Moreover, this is my first time using JavaFX. I learned to structure classes and objects properly. I can develop my understanding with OOP. Besides that, I learn event listeners more.

V. Project Technical Description

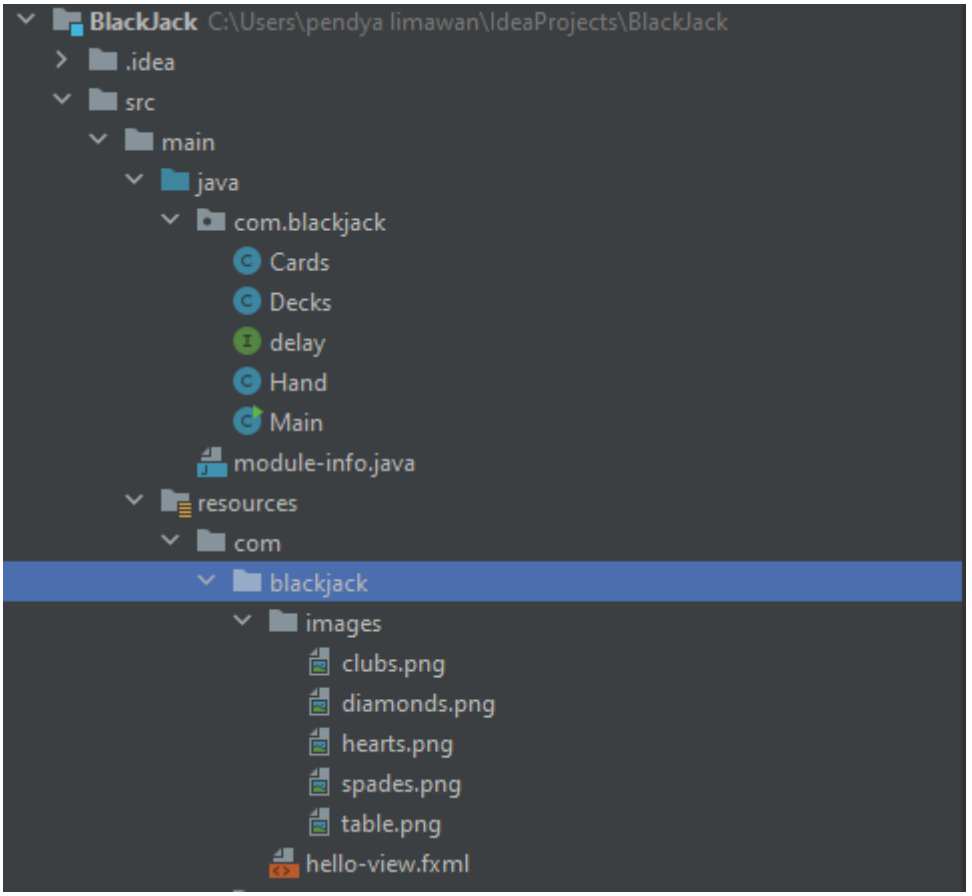
JavaFX

The JavaFX library is a library that has a main root where the scene / GUI will be placed. To draw in the main root, a parent class is needed and you can get the children to draw the contents.

Java

The java classes in this project are separated into 4: main, hand, cards, and decks. Each one has a different function. Main will run the whole application, hand will create a hand for the dealer and the player, cards will create the cards, and decks will create the deck of cards. Other than that, there is also a java interface called delay. Delay will be implemented in the main class to delay some of the animations.

VI. Code Explanation



Project Structure

Here I have main which will run the application
Cards, Decks, Hand, and delay have their own functions. Each one will run the logic behind the application
Under resources/com/blackjack/images/ we have the assets needed for the application.

Delay Interface

```java
package com.blackjack;
import javafx.concurrent.Task;

public interface delay {
    //delay method
    static void delay(long millis, Runnable continuation) {
        Task<Void> sleeper = new Task<Void>() {     //stop the task
            @Override
            protected Void call() throws Exception {
                try { Thread.sleep(millis); }
                catch (InterruptedException e) { }
                return null;
            }
        };
        sleeper.setOnSucceeded(event -> continuation.run());
        new Thread(sleeper).start();
    }
}
```

This delay algorithm is not made by me, it's made by user DaveB on StackOverFlow. How this works is by taking 2 parameters; millis (millisecond) and Runnable continuation (the next step after the delay). Task<Void> sleeper will create a new empty task. Then there will be a new method Void call() which will use Thread.sleep which is used to stop a task for a specific millisecond. InterruptedException will be executed next, it's a checked exception, and many blocking operations in Java can throw it. Then sleeper.setOnSucceeded will run the continuation after the sleeper task has succeeded.

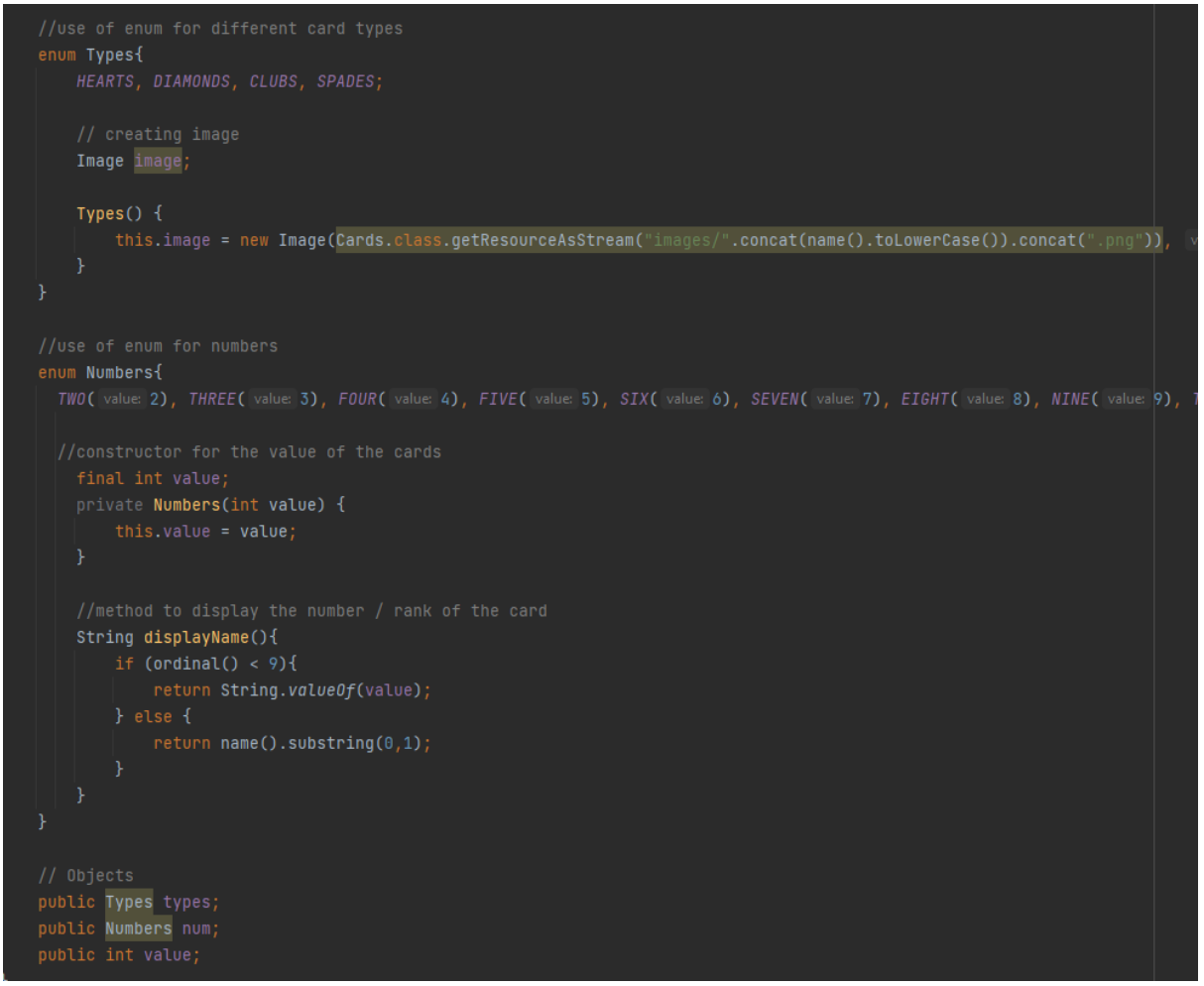
Cards.java

```java
package com.blackjack;

//import
import ...

//extends Parent needed to use getChildren()
public class Cards extends Parent{

    //Constants for card size
    private static final int CARD_WIDTH = 80;
    private static final int CARD_HEIGHT = 120;
```

Constants for the card size is set here.

```java
//Cards constructor
public Cards(Types types, Numbers num) {
    this.types = types;
    this.num = num;
    this.value = num.value;

    //set the size of the cards
    Rectangle bg = new Rectangle(CARD_WIDTH,CARD_HEIGHT);
    bg.setArcWidth(20);         //add curve to the cards
    bg.setArcHeight(20);        //add curve to the cards
    bg.setFill(Color.WHITE);    //fill the cards with color

    //display the number on the top right
    Text text = new Text(num.displayName());
    text.setFont(Font.font(18));
    text.setX(CARD_WIDTH - text.getLayoutBounds().getWidth() - 10);
    text.setY(text.getLayoutBounds().getHeight());

    //display the number on bottom left
    Text text2 = new Text(text.getText());
    text2.setFont(Font.font(18));
    text2.setX(10);
    text2.setY(CARD_HEIGHT - 10);

    //adding the card suits(types) //top left
    ImageView view = new ImageView(types.image);
    view.setX(CARD_WIDTH - 32);
    view.setY(CARD_HEIGHT - 32);

    //adding the contents to the scene
    getChildren().addAll(bg, new ImageView(types.image), view, text, text2);
}
```

The card constructor. set the size of the card using the constant and add arch and color on the cards. The next bit is where the text will be created for the top right of the card. Text2 is the bottom left part of the card. Both text and text2 use javafx.scene.Font to be set. imageView view is adding the card's suits on the top left. getChildren.addAll is drawing the contents into the root scene. Because it is in order, then the order of writing is important to be background first.



```
//use of enum for different card types
enum Types{
    HEARTS, DIAMONDS, CLUBS, SPADES;

    // creating image
    Image image;

    Types() {
        this.image = new Image(Cards.class.getResourceAsStream("images/".concat(name().toLowerCase()).concat(".png")),
    }
}

//use of enum for numbers
enum Numbers{
  TWO( value: 2), THREE( value: 3), FOUR( value: 4), FIVE( value: 5), SIX( value: 6), SEVEN( value: 7), EIGHT( value: 8), NINE( value: 9),

  //constructor for the value of the cards
    final int value;
    private Numbers(int value) {
        this.value = value;
    }

    //method to display the number / rank of the card
    String displayName(){
        if (ordinal() < 9){
            return String.valueOf(value);
        } else {
            return name().substring(0,1);
        }
    }
}

// Objects
public Types types;
public Numbers num;
public int value;
```

Enum is implemented because we need to access different ordinals. Under enum Types, we can see the 4 different card types. Then, we create the image and construct it in Types().
Under enum Numbers, we can see the different numbers / ranks of the cards from 2 until ace. Then we have the value constructor which will take the value on each enum. Then the String displayName method is to display the rank of each card. On the first condition we have if the ordinal is less than 9 which is the rank 2 until 10 will return the value of the string. The second condition, it will return each substring's first alphabet.

Hand.java

```java
package com.blackjack;

/import
import ...

public class Hand {
    private ObservableList<Node> cards;         //observable list to listen to listeners
    private SimpleIntegerProperty value = new SimpleIntegerProperty( i: 0); //simple integer property creates a getter, setter, and
    private int aces = 0;   //count the aces

    //constructor
    public Hand(ObservableList<Node> cards){
        this.cards = cards;
    }

    public void takeCard(Cards card){   //  add values of cards to the hand
        cards.add(card);
        if (card.num == Numbers.ACE){   // ace counter
            aces++;
        }

        if (value.get() + card.value > 21 && aces > 0){
            value.set(value.get() + card.value - 10);
            aces--;
        }
        else {
            value.set(value.get() + card.value);
        }
    }

    public void reset(){    //reset hand
        cards.clear();
        value.set(0);
        aces = 0;
    }

    public SimpleIntegerProperty valueProperty(){   //to access the value, because value is private
        return value;
    }
}
```

Here, we have 3 private variables and objects, the first one is an observable list. ObservableList is implemented because ObservableList can be added and binded with listeners. Next, we have a SimpleIntegerProperty. A SimpleIntegerProperty has a built in getter and setter also it can be added and binded with listeners. Then we have variable aces.

The constructor will construct an observable list for the cards. The takeCard method is implemented to get a value of the hand. It will add a card to the cards list, and it will check a few conditions. If the card is an ace, then the aces variable will be incremented. If the sum of cards in hand and the card itself is more than 21 and the number of aces are more than 0, then the value of the hand will be deducted by 10, because an ace can be 1 or 11. The last condition is everything else, it will just set the value to whatever the sum of the hand and the card is.

Reset method here is to clear the hand and the value of the hand.

Next, the valueProperty method is used because simpleIntegerProperty is private, so we can't access the value. So, we need to return the value of the integer in this method.

Decks.java

```java
package com.blackjack;

//import from Cards.java
import ...

public class Decks{

    //create an array of the deck
    private Cards[] cards = new Cards[52];

    public Decks(){
        refill();
    }

    //refill method to refill the deck
    public final void refill(){
        int i = 0;
        for (Types types : Types.values()){         // for each loop
            for(Numbers num : Numbers.values()){    // for each loop
                cards[i++] = new Cards(types, num); // add the card to the deck
            }
        }
    }

    //draw card from the deck
    public Cards drawCard(){
        Cards card = null;          // a temporary card
        while (card == null){
            int index = (int)(Math.random()*cards.length);     //random index of card
            card = cards[index];
            cards[index] = null;    //empty the index of the deck
        }
        return card;
    }
}
```

In decks.java we are trying to create a deck of cards. The first thing to do is to create an array for the deck containing 52 cards. There is a refill method to refill the deck using for each loop to fill the deck. Then, the drawCard method to draw cards from the deck. We will create a temporary card to fill the random index card from the deck. The math.random is being used here to pick a random index. Afterwards, the temporary card will take that index from the deck and the method will return the temporary card.

Main.java

```
package com.blackjack;

//import
import ...

public class Main extends Application implements delay{

    private Decks decks = new Decks();  //create a deck
    private Hand dealer, player;        //create a hand for dealer and player
    private Text winner = new Text();   //winner text, will be shown when the game is over

    private SimpleBooleanProperty playable = new SimpleBooleanProperty( b: false); //set playable to false, for buttons

    //create a horizontal box for the dealer and player
    Rectangle flippedCard = new Rectangle( v: 80,  v1: 120);
    private HBox dealerCards = new HBox( v: 20, flippedCard);
    private HBox playerCards = new HBox( v: 20);

    private Parent createContent() {
        dealer = new Hand(dealerCards.getChildren());   //create hand objects
        player = new Hand(playerCards.getChildren());   //

        Pane main = new Pane();             //create the main layout
        main.setPrefSize( v: 800,  v1: 600);

        Region background = new Region();       //create background
        background.setPrefSize( v: 800,  v1: 600);
        background.setStyle("-fx-background-color: rgba(0, 0, 0, 1)");

        HBox mainLayout = new HBox( v: 5);  //horizontal box for the main layout
        mainLayout.setPadding(new Insets( v: 5,  v1: 5,  v2: 5,  v3: 5));
        Image table = new Image(getClass().getResourceAsStream( name: "images/table.png"),  v: 550,  v1: 550,  b: true,  b1: true);
        ImageView view = new ImageView(table);
        view.setX(550);
        view.setY(550);;
        Rectangle right = new Rectangle( v: 230,  v1: 550);
        right.setFill(Color.LIGHTGREEN);
```

In main.java, the application is extended from javafx. Application is needed to start the program. First, we create a deck, hand for dealer and player, and winner message. Next we create a horizontal box containing the two hands. Then the createcontent method is used to store and add the contents to the scene. First, get the children's hands to add the cards to the scene. Next, we create the main pane / root with the size. Then we add a background, add the color and the size. The hbox main layout will split the contents on it horizontally on the scene. The first thing on the main layout is the image, and a right rectangle.

```
VBox leftVBox = new VBox( v: 40);
leftVBox.setAlignment(Pos.CENTER);

flippedCard.setArcHeight(20);
flippedCard.setArcWidth(20);
flippedCard.setFill(Color.CADETBLUE);
flippedCard.setVisible(false);

Text dealerScore = new Text( s: "Dealer: ");
HBox dealerBox = new HBox( v: 15, dealerScore, dealerCards, flippedCard);
dealerBox.setPrefHeight(200);
dealerBox.setAlignment(Pos.TOP_CENTER);
Text playerScore = new Text( s: "Player: ");
HBox playerBox = new HBox( v: 15, playerScore, playerCards);
playerBox.setPrefHeight(200);
playerBox.setAlignment(Pos.TOP_CENTER);

leftVBox.getChildren().addAll(dealerBox, winner, playerBox);

// RIGHT

VBox rightVBox = new VBox( v: 20);
rightVBox.setAlignment(Pos.CENTER);

Button playBtn = new Button( s: "PLAY");
Button hitBtn = new Button( s: "HIT");
Button standBtn = new Button( s: "STAND");

HBox btnBox = new HBox( v: 15, hitBtn, standBtn);
btnBox.setAlignment(Pos.CENTER);

rightVBox.getChildren().addAll(playBtn, btnBox);

// ADD BOTH PART TO MAIN LAYOUT`
mainLayout.getChildren().addAll(new StackPane(view,  leftVBox),  new StackPane(right, rightVBox));
main.getChildren().addAll(background, mainLayout);
```

Here, we have a vertical box on the left to store the cards and the score. Next, we have the right vertical box containing the buttons. Then we add both left and right to the main layout.

```
    //BIND PROPERTIES
    playBtn.disableProperty().bind(playable);
    hitBtn.disableProperty().bind(playable.not());
    standBtn.disableProperty().bind(playable.not());

    playerScore.textProperty().bind(new SimpleStringProperty( s: "Player: ").concat(player.valueProperty().asString()));
    dealerScore.textProperty().bind(new SimpleStringProperty( s: "Dealer: ").concat(dealer.valueProperty().asString()));

    player.valueProperty().addListener((obs, oldVal, newVal) -> {
        if (newVal.intValue() >= 21) {
            endGame();
        }
    });
    dealer.valueProperty().addListener((obs, oldVal, newVal) -> {
        if (newVal.intValue() >= 21) {
            endGame();
        }
    });

    // INIT BUTTONS

    playBtn.setOnAction(event -> startGame());

    hitBtn.setOnAction(event -> player.takeCard(decks.drawCard()));

    standBtn.setOnAction(event -> {
        endGame();
    });
return main;
}
```

Here, we bind the buttons and the score text. The play button is set to playable, but the two buttons are not. Next, the value of the hand will be added to the player and dealer score. Then we add listeners to those two scores and if the score is more than or equal to 21 the game will end. Then we initialize the buttons with event listeners. On the play button, on click the game will start. Next, the player will draw a card if the hit button is clicked. For the stand button, it will end the game.

```
private void startGame(){
    playable.set(true);
    winner.setText("");
    decks.refill();
    dealer.reset();
    player.reset();

    dealer.takeCard(decks.drawCard());
    flippedCard.setVisible(true);
    delay.delay( millis: 500, ()->{
        player.takeCard(decks.drawCard());
        player.takeCard(decks.drawCard());
    });

}
```

The startGame method is used for starting the game, refilling the cards, and resetting the hands. It will draw 2 cards for each dealer and player.

```
//END GAME
private void endGame(){
    playable.set(false);
    while(dealer.valueProperty().get() < 17){
        dealer.takeCard(decks.drawCard());
    }
    flippedCard.setVisible(false);
    int dealerValue = dealer.valueProperty().get();
    int playerValue = player.valueProperty().get();

    if(dealerValue == playerValue){
        delay.delay( millis: 300, ()->{
            winner.setText("DRAW");
            winner.setFont(Font.font(24));
            winner.setFill(Color.WHITE);
        });
    }

    else if (dealerValue == 21 || playerValue > 21 || dealerValue < 21 && playerValue < dealerValue){
        delay.delay( millis: 300, ()->{
            winner.setText("DEALER WON");
            winner.setFont(Font.font(24));
            winner.setFill(Color.WHITE);
        });
    }

    else {
        delay.delay( millis: 300, ()->{
            winner.setText("PLAYER WON");
            winner.setFont(Font.font(24));
            winner.setFill(Color.WHITE);
        });
    }
}
```

The endGame method will stop the game and get the values of the 2 hands. Before that, it will set a while loop to check if the dealer's hand value is more than 17. If it is more than 17, the dealer will stand, but if it's less than 17, the dealer will keep on drawing cards. If the dealer value is equal to the player value, the game will stop. On the other hand, if the condition is in favor of the dealer, the dealer will win. Same thing goes for the player.

```
    @Override
@   public void start(Stage primaryStage) {
        primaryStage.setScene(new Scene(createContent()));  //create the content
        primaryStage.setWidth(800);                         //set size
        primaryStage.setHeight(600);
        primaryStage.setResizable(false);                   //make the window not resizable
        primaryStage.setTitle("BlackJack");                 //name
        primaryStage.show();                                //show the window
    }

    public static void main(String[] args){ //start the game
        launch(args);
    }
}
```

Here, we have a start method that will set the scene. And for the main function, it will launch the program.

VII. GitHub Link

https://github.com/pan-dya/FinalProject

VIII. References

https://stackoverflow.com/questions/37962910/javafx-addlistener-not-working
https://stackoverflow.com/questions/26454149/make-javafx-wait-and-continue-with-code
https://www.programiz.com/java-programming/enhanced-for-loop#:~:text=In%20Java%2C%20the%20for%2Deach,as%20the%20enhanced%20for%20loop.
https://openjfx.io/javadoc/13/javafx.graphics/javafx/scene/class-use/Parent.html
https://docs.oracle.com/javase/8/javafx/api/javafx/beans/property/SimpleIntegerProperty.html
https://docs.oracle.com/javase/8/javafx/api/javafx/collections/ObservableList.html#:~:text=Interface%20ObservableList&text=A%20list%20that%20allows%20listeners%20to%20track%20changes%20when%20they%20occur.
https://docs.oracle.com/javase/8/javafx/api/javafx/beans/property/SimpleBooleanProperty.html

Assets:
https://www.pngitem.com/middle/xwxTx_transparent-card-suits-png-png-download/