UCCN2044 Object-Oriented Programming Practices

Assignment Report

BY

Lim Yun Kai
(1402083)
P5

1. Program Description

The program starts by letting user to select number of player. User can select to start the game with either 2, 3, or 4 players. After user has chosen the number of player, the game begins with a new round.

A round start with 8 pool card on the centre and each player get 4 hand card, all card a randomly shuffled before distributing to the pool card or player card. In each player turn, players had to make a capture or discard a card (known as trailing) to the pool. A capture will consist of exactly one player hand card and several card from the pool. After player click the 'capture' button, the programming will automatically determine the captures of the player (also including trailing where player select only one of the hand card). If it is a valid capture, the current player turns ends and the next player turns begin.

A player cards will be hidden and the capture button for that player is disabled if currently if not that player's turn. Also, capturing status will be shown at the right panel of the windows. All captures that a player made in a round will also be displayed in the right panel of the windows.

After a round had ended (each player had played 4 turns), if none of the player had 21 points or more, a new round begin and the players continue to play the game until a player wins the game (first to reach the score of 21 or more). After a winner is declared, the program will let user to select continue to play a new game or exit the game.

2. New Capture Type

Two new capture type had been added to the game to make the game play more interesting and player can form more capture with his/her current hand cards.

The first capture type added is called 'Triple'. As suggested by the name, user has to captures exactly 2 pools card that has same rank as one of the player hand card forming a triple. A triples capture will get the player 3 points.

Second capture type added is called 'Straight'. The card that form the 'Straight' capture is actually almost same as the 'Run' capture. The capturing card has to form a sequence, but not necessary in a same suit. The capturing card can from any suit as long as they form a sequence with the player hand card. The scoring of this capture is 1.3 x number of card captured.

3. Class Documentation

3.1 Capture

An abstract class that uses to represent a capture and form capture.

Contain following method:

- formCapture()

  An abstract class, subclass had to implement this function. Use to forms capture of its own type. Return the capture object if it is a valid capture, null otherwise.

- getScore()

  An abstract class, subclass had to implement this function. Use the get the total score of this capture.

- getCaptureName()

  An abstract class, subclass had to implement this function. Use to get the name of this capture.

- getCaptureCard()

  Used to get all card that form this capture. Used by Game class to remove pool card and display player's capture.

- formCaptures()

  A static class. Used to form capture automatically. It works by trying to form capture of each type and return the type with highest possible score. Return null if none of the capture can be formed.

3.2 Card

A class representing the cards object. Contain attribute such as rank, suit, card image and card back image.

Contain following method:

- Constructor

  Construct a card object represented by a value from 0 to 51 each number representing 1 card in the playing card. The rank and suit is calculated.

- getSuit()

  Return the suit in string format. Useful for displaying the card and printing to console.

- getSuitValue()

  Return the value representing the suit. Useful for comparison.

- getRank()

Return the rank in string format. Useful for displaying the card and printing to console.

- getRankValue()

  Return the value representing the rank. Useful for comparison and calculation.

- toString()

  Override Java default toString() method to provide better object to string conversion. Useful for printing and displaying the object.

- newDeck()

  Return an array of cards containing new deck of card which include all 52 playing card.

- shuffleDeck()

  Takes in an array of cards and shuffle them randomly.

- compareTo()

  Implement comparable interface to support generic sorting on card object. The sorting order is sorting the suit ascendingly then the rank ascendingly.

- getEmptyCard()

  A static method used to get a card object which represent 'nothing'. It is different to null as null representing invalid or error while empty card representing no card or nothing.

- getImage()

  Return the Image of the card. The Image of the card object is loaded upon requested to increase efficiency as some card are not shown in the game.

- getBackImage()

  A static method to return the backside image of the card. Since all the backside image should be the same, the method is therefore static.

3.3 CardPane

A 'placeholder' of a card image. It is a subclass of StackPane and some extra functionality is implemented to aids the main class for event handling. The card pane has the structure of StackPane -> ImageView -> Image.

Contains following method:

- Constructor

  Create the ImageView, setting the ImageView width and properties. Finally place the ImageView node inside the StackPane.

- setCard()

Set the card object to be associated with this CardPane and bind event handler (select or deselect the card) to this object.

- setCurrentPlayer

  Set the context to current player. All action such as selecting or deselecting of the card is performed by the current player.

- setActive

  Set whether the current card is active or not. An active card can be selected or deselected while an inactive card cannot. An active card will display as the front side of the card while an inactive card will display as the back side of the card.

3.4 Combo

Class that representing capture of combo type.

Contain following method:

- public no-args Constructor

  To construct the object with no capture so that the formCapture() method can be used.

- private Constructor

  Constructor used by formCapture() to construct a valid capture class. formCapture() will perform a validation before constructing the object. Constructor is private so that invalid capture object cannot be constructed.

- formCapture()

  Check if the provided card combination is a valid Combo capture. If it is a valid capture, return the Combo object and null otherwise.

- getScore()

  Return the score of this capture.

- getCaptureName()

  Return a string representing the name of this capture for displaying.

3.5 Game

Main class for the whole Java program. Responsible to perform the game logic and GUI design.

- setNextPlayer()

  A helper function to set the current context to next player.

- initializeRound()

  A helper function to initialize a round to a new round. Deck are shuffle and card are distributed to the pool and players.

- formCaptureImage()

  A helper function to form capture list image at the right capture panel. The returned node can be directly appended to the right capture panel to display player's capture.

- start()

  The main function of the program. First part of the code is about GUI design. The main playing area is a border pane with center is the pool card, bottom, left, up, right are the areas for player 1, 2, 3, 4 respectively. The pool card and hand card for player 2, 4 are in a grid pane while the hand card for player 1, 3 are in a hbox. After the GUI design, event handler are attached to each component. There are mainly 2 type of event handler. First one is the event handler for the number of player selection button. When user select number of player, the event handler will initialize the game with selected number of player. Next is the event handler for the capture button. When player press capture, the event handler will form a capture, check for valid capture, determine current round had finish, set context to next player or start a new round, and finally declare a winner. The event handler also responsible for updating the score and right side capture panel.

- main()

  To start the JavaFX apps.

3.6 Pair

Class that representing capture of Pair type.

- public no-args Constructor

  To construct the object with no capture so that the formCapture() method can be used.

- private Constructor

  Constructor used by formCapture() to construct a valid capture class. formCapture() will perform a validation before constructing the object. Constructor is private so that invalid capture object cannot be constructed.

- formCapture()

  Check if the provided card combination is a valid Pair capture. If it is a valid capture, return the Combo object and null otherwise.
- getScore()

  Return the score of this capture.
- getCaptureName()

  Return a string representing the name of this capture for displaying.

3.7 Player

Class representing player. Contain attribute such as score, hand card, chosen card and captures. Contain following method:

- addHandCard()

  Add a card into the player's hands card
- addHandCard() (overloaded to accept array of card)

  Overloaded method to add multiple card into the player's hands card.
- removeHandCard()

  Remove a card from current player hand card.
- getHandCard()

  Return an array of card that contain player hand card.
- addChoosenCard()

  Add a card to player choosen card. Used when player select a card.
- removeChoosenCard()

  Remove a card from player choosen card. Used when player deselect a card.
- playChoosenCard()

  Play the card that is choosen by the player. If the choosen card form a capture, an empty card is returned. If the only choosen card is player hand card, the card will be discarded to the pool, hence the card will be returned. Null value is return if it is an invalid move. After a valid play, the score will be automatically accumulated to current player score.
- formCaptureChoosenCard()

  Tried to form a capture with the player choosen card. Return the capture object if a capture is successfully formed and null otherwise.
- getLatestCapture()

Return the capture object of latest capture. Used when main function wants to discard the captured card from pool.

- getScore()
  Return the total score accumulated for current player.

3.8 Run

Class that representing capture of Run type.

- public no-args Constructor
  To construct the object with no capture so that the formCapture() method can be used.
- private Constructor
  Constructor used by formCapture() to construct a valid capture class. formCapture() will perform a validation before constructing the object. Constructor is private so that invalid capture object cannot be constructed.
- formCapture()
  Check if the provided card combination is a valid Run capture. If it is a valid capture, return the Combo object and null otherwise.
- getScore()
  Return the score of this capture.
- getCaptureName()
  Return a string representing the name of this capture for displaying.

3.9 Straight

Class that representing capture of Straight type.

- public no-args Constructor
  To construct the object with no capture so that the formCapture() method can be used.
- private Constructor
  Constructor used by formCapture() to construct a valid capture class. formCapture() will perform a validation before constructing the object. Constructor is private so that invalid capture object cannot be constructed.
- formCapture()
  Check if the provided card combination is a valid Straight capture. If it is a valid capture, return the Combo object and null otherwise.
- getScore()

Return the score of this capture.

- getCaptureName()

  Return a string representing the name of this capture for displaying.

3.10 Triple

Class that representing capture of Triple type.

- public no-args Constructor

  To construct the object with no capture so that the formCapture() method can be used.

- private Constructor

  Constructor used by formCapture() to construct a valid capture class. formCapture() will perform a validation before constructing the object. Constructor is private so that invalid capture object cannot be constructed.

- formCapture()

  Check if the provided card combination is a valid Triple capture. If it is a valid capture, return the Combo object and null otherwise.

- getScore()

  Return the score of this capture.

- getCaptureName()

  Return a string representing the name of this capture for displaying.

4. Screenshot

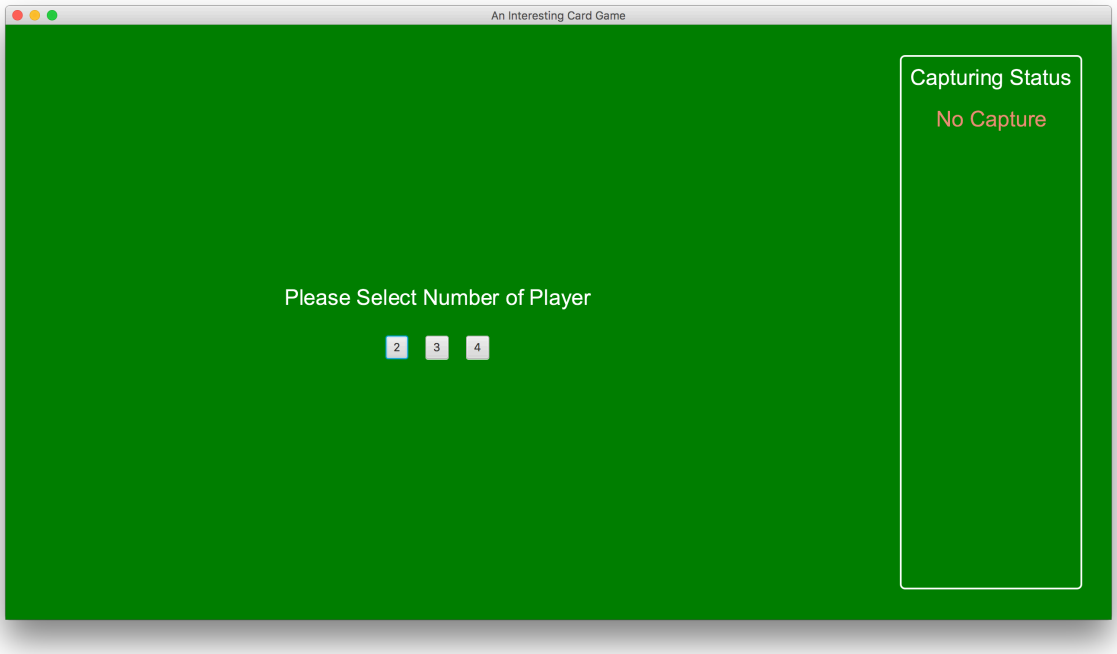

Diagram 4.1 The player selection interface

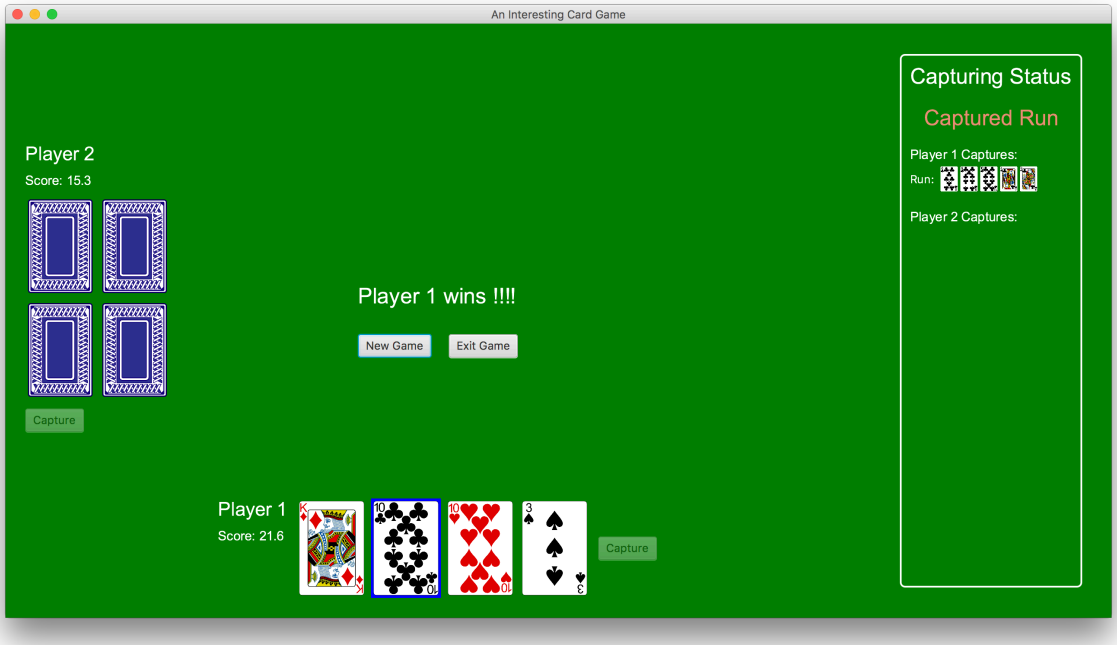

Diagram 4.2 Interface for 2 player game play and winner is declared

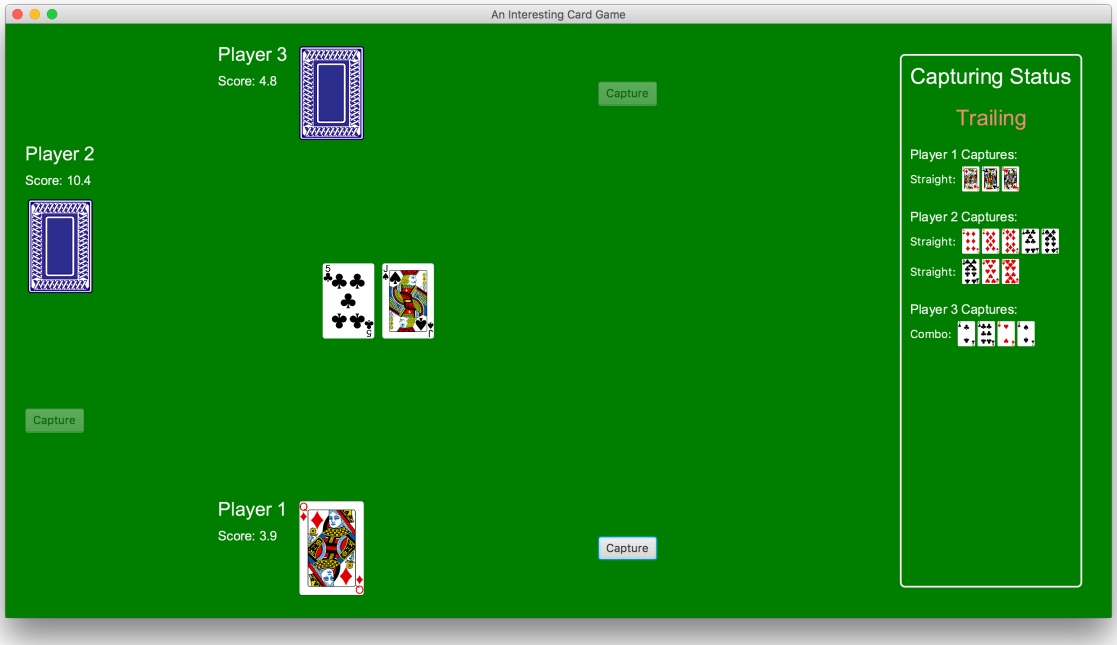

Diagram 4.3 Interface for 3 player game and the right capturing status panel will show the capturing status (text in brown) and each player captures in current round

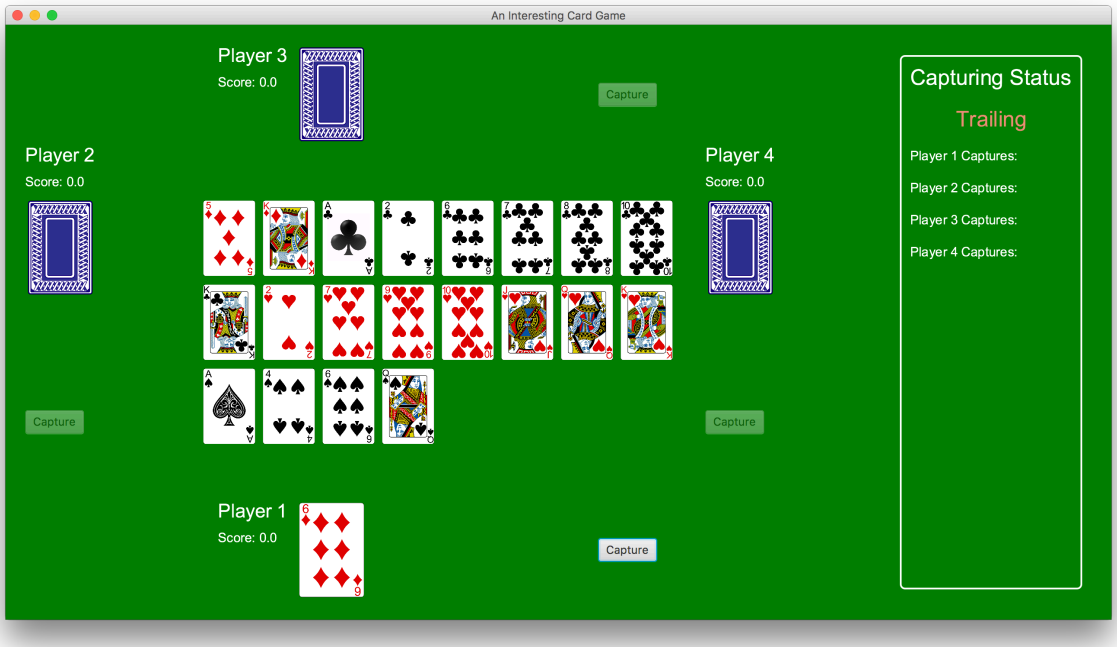

Diagram 4.4 Interface for 4 player game and the dynamic resizing of center pool card feature

5. Code

5.1 Capture

```
import java.util.Arrays;

public abstract class Capture{
    protected double multiplier;
    protected Card[] captureCards;
    private static Capture[] allPossibleCapture = {new Run(), new Straight(), new
Combo(), new Triple(), new Pair()};

    // for subclass to implement
    /* return capture object of its own type for valid capture, null otherwise */f
    public abstract Capture formCapture(Card handCard, Card[] poolCards);
    public abstract double getScore();
    public abstract String getCaptureName();

    // protect the array
    public Card[] getCaptureCards(){
        Card[] returnCards = new Card[captureCards.length];
        for(int i = 0; i < captureCards.length; i++)
            returnCards[i] = captureCards[i];
        Arrays.sort(returnCards);
        return returnCards;
    };

    // return highest possible scored capture, null if cannot form capture
    public static Capture formCaptures(Card handCard, Card[] poolCards){
        // try form higher score capture first
        // allPossibleCapture must sorted from highest possible score to lowest
        for(Capture c: allPossibleCapture)
            if(c.formCapture(handCard, poolCards) != null)
                return c.formCapture(handCard, poolCards);

        return null;
    }
}
```

5.2 Card

```
import java.util.Random;
import java.util.Comparator;
import javafx.scene.image.*;
import java.net.URL;

public class Card implements Comparable<Card>{
    private int suit, rank;
    private final String[] verbose_suit = {"Diamond", "Club", "Heart", "Spade"};
    private final String[] verbose_rank = {"A", "2", "3", "4", "5", "6", "7", "8",
"9", "10", "J", "Q", "K"};
    private final String[] suit_symbol = {"\u2662", "\u2663", "\u2661", "\u2660"};
    private Image cardImage;
    private static Card emptyCard = new Card(-1);
    private static Image backImage = new Image("file:../img/back.png");
    //       private       static       Image       backImage       =       new
Image(Card.class.getResourceAsStream("img/back.png"));

    // value should be 0-51, -1 for empty card
```

```
public Card(int value){
    if(0 <= value && value <= 51){
        suit = value/13;
        rank = value%13;
    }
}

public String getSuit(){
    return verbose_suit[suit];
}

public int getSuitValue(){
    return suit;
}

public String getRank(){
    return verbose_rank[rank];
}

public int getRankValue(){
    return rank+1;
}

public String toString(){
    return suit_symbol[suit]+verbose_rank[rank];
}

public static Card[] newDeck(){
    Card[] deck = new Card[51];
    for(int i = 0; i < deck.length; i++){
        deck[i] = new Card(i);
    }
    return deck;
}

public static Card[] shuffleDeck(Card[] deck){
    Card[] shuffled = new Card[deck.length];

    Random rand = new Random();
    for(int i = 0; i < deck.length; i++){
        int nextCard = rand.nextInt(deck.length-i);
        for(int j = 0; j < deck.length; j++){
            // currect card is the one selected
            if(nextCard == 0 && deck[j] != null){
                shuffled[i] = deck[j];
                deck[j] = null;
                break;
            }

            if(deck[j] != null)
                nextCard--;
        }
    }

    return shuffled;
}

public int compareTo(Card c){
    if(this.suit != c.suit)
```

```
            return this.suit - c.suit;
        return this.rank - c.rank;
    }

    public static Card getEmptyCard(){
        return emptyCard;
    }

    public Image getImage(){
        // load card image only when needed
        if(cardImage == null)
            cardImage = new Image(
                ("file:../img/"   +   verbose_rank[rank]+   verbose_suit[suit]   +
".png").toLowerCase());
                //  Card.class.getResourceAsStream(("img/"  +  verbose_rank[rank]+
verbose_suit[suit] + ".png").toLowerCase()));
        return cardImage;
    }

    public static Image getBackImage(){
        return backImage;
    }

}
```

5.3 CardPane

```
import javafx.geometry.*;
import javafx.scene.image.*;
import javafx.scene.layout.StackPane;

public class CardPane extends StackPane{
    private Player currentPlayer;
    private Card card;
    private ImageView iv;
    private boolean selected = false;
    private boolean active = true;

    public CardPane(int width){
        iv = new ImageView();
        iv.setFitWidth(width);
        iv.setPreserveRatio(true);
        iv.setSmooth(true);
        iv.setCache(true);

        setMargin(iv, new Insets(3));
        getChildren().add(iv);
    }

    public void setCard(Card card){
        this.card = card;
        selected = false;
        setStyle("-fx-background-color: none");

        if(card == null){
            iv.setImage(null);
            setOnMouseClicked(null);
        }
        else{
```

```
            iv.setImage(card.getImage());

            setOnMouseClicked(e ->{
                // dont do anything if not active
                if(!active)
                    return;

                if(!selected){
                    setStyle("-fx-background-color: blue");
                    currentPlayer.addChoosenCard(card);
                }
                else{
                    setStyle("-fx-background-color: none");
                    currentPlayer.removeChoosenCard(card);
                }

                selected = !selected;
            });
        }
    }

    public void setCurrentPlayer(Player player){
        currentPlayer = player;
    }

    public void setActive(boolean active){
        this.active = active;

        if(card == null)
            return;

        if(!active)
            iv.setImage(card.getBackImage());
        else
            iv.setImage(card.getImage());
    }
}
```

5.4 Combo

```
public class Combo extends Capture{
    public Combo(){};

    private Combo(Card handCard, Card[] poolCards){
        multiplier = 1.2;
        captureCards = new Card[poolCards.length+1];
        captureCards[0] = handCard;
        for(int i = 0; i < poolCards.length; i++){
            captureCards[i+1] = poolCards[i];
        }
    }

    public Capture formCapture(Card handCard, Card[] poolCards){
        if(poolCards.length < 2)
            return null;

        int sum = 0;

        for(Card c: poolCards)
```

```
            if(c.getRankValue() > 10)
                return null;
            else
                sum += c.getRankValue();

        if(handCard.getRankValue() > 10)
            return null;

        if(handCard.getRankValue() != sum)
            return null;

        return new Combo(handCard, poolCards);

    }
    public double getScore(){
        return multiplier*captureCards.length;
    }

    public String getCaptureName(){
        return "Combo";
    }

}
```

5.5 Game

```
import java.util.*;
import java.text.DecimalFormat;
import javafx.application.*;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.scene.layout.*;
import javafx.scene.image.*;
import javafx.geometry.*;
import javafx.scene.text.*;
import javafx.event.*;
import javafx.scene.paint.Color;

public class Game extends Application{
    final static int POOLCARD = 8, PLAYERCARD = 4, WINNING_SCORE = 21;
    static int currentPlayerIdx, numberOfPlayer;
    static Player currentPlayer;
    static Player[] players;
    static Card[] deck;
    static ArrayList<Card> poolCards;
    static CardPane[][] playerCardsImage;
    static CardPane[] poolCardsImage;
    static GridPane poolCardsPane;

    public static void setNextPlayer(int nextPlayer){
        for(int i = 0; i < players.length; i++){
            Card[] playerHandCard = players[i].getHandCard();
            for(int j = 0; j < playerCardsImage[i].length; j++){
                if(j < playerHandCard.length){
                    playerCardsImage[i][j].setCard(playerHandCard[j]);
                }
                else{
                    playerCardsImage[i][j].setCard(null);
```

```
            }
            playerCardsImage[i][j].setCurrentPlayer(players[nextPlayer]);
            if(i != nextPlayer || j >= playerHandCard.length)
                playerCardsImage[i][j].setActive(false);
            else
                playerCardsImage[i][j].setActive(true);
        }
    }

    Collections.sort(poolCards);
    for(int i = 0; i < poolCardsImage.length; i++){
        if(i < poolCards.size()){
            poolCardsImage[i].setCard(poolCards.get(i));
            poolCardsImage[i].setCurrentPlayer(players[nextPlayer]);
            poolCardsImage[i].setActive(true);
        }
        else{
            poolCardsImage[i].setCard(null);
            poolCardsImage[i].setActive(false);
        }
    }

    poolCardsPane.getChildren().clear();
    if(poolCards.size() <= 12){
        // 4 col
        for(int i = 0; i < poolCardsImage.length; i++)
            poolCardsPane.add(poolCardsImage[i], i%4, i/4);
    }
    else if(poolCards.size() <= 18){
        // 6 col
        for(int i = 0; i < poolCardsImage.length; i++)
            poolCardsPane.add(poolCardsImage[i], i%6, i/6);
    }
    else{
        // 8 col
        for(int i = 0; i < poolCardsImage.length; i++)
            poolCardsPane.add(poolCardsImage[i], i%8, i/8);
    }
}

public static void initializeRound(int p){
    deck = Card.newDeck();
    deck = Card.shuffleDeck(deck);

    // select pool card
    poolCards = new ArrayList<Card>();
    for(int i = 0; i < POOLCARD; i++)
        poolCards.add(deck[i]);

    // distribute card to player
    for(int i = 0, idx = POOLCARD; i < PLAYERCARD*p; i++, idx++)
        players[i%p].addHandCard(deck[idx]);
}

public HBox formCapturesImage(Capture capture){
    HBox capturePane = new HBox(3);
    Text t = new Text(capture.getCaptureName() + ": ");
    t.setFill(Color.WHITE);
```

```
        capturePane.getChildren().add(t);

        for(Card c: capture.getCaptureCards()){
            ImageView iv = new ImageView();
            iv.setFitWidth(20);
            iv.setPreserveRatio(true);
            iv.setSmooth(true);
            iv.setCache(true);

            iv.setImage(c.getImage());

            capturePane.getChildren().add(iv);
        }

        capturePane.setAlignment(Pos.CENTER_LEFT);
        return capturePane;
    }

    public void start(Stage primaryStage) {
        /* GUI Design begin */
        playerCardsImage = new CardPane[4][PLAYERCARD];
        for(int i = 0; i < playerCardsImage.length; i++){
            for(int j = 0; j < playerCardsImage[i].length; j++){
                playerCardsImage[i][j] = new CardPane(75);
            }
        }

        poolCardsImage = new CardPane[PLAYERCARD*4 + POOLCARD];
        for(int i = 0; i < poolCardsImage.length; i++){
            poolCardsImage[i] = new CardPane(60);
        }

        HBox player1CardsPane = new HBox(5), player3CardsPane = new HBox(5);
        player1CardsPane.setPrefHeight(115); player3CardsPane.setPrefHeight(115);
        for(int i = 0; i < 4; i++){
            player1CardsPane.getChildren().add(playerCardsImage[0][i]);
            player3CardsPane.getChildren().add(playerCardsImage[2][i]);
        }

        GridPane player2CardsPane = new GridPane(), player4CardsPane = new
GridPane();
        player2CardsPane.setVgap(5);    player4CardsPane.setVgap(5);
        player2CardsPane.setHgap(5);    player4CardsPane.setHgap(5);
        player2CardsPane.setPrefHeight(235); player4CardsPane.setPrefHeight(235);
        for(int i = 0; i < 4; i++){
            player2CardsPane.add(playerCardsImage[1][i], i%2, i/2);
            player4CardsPane.add(playerCardsImage[3][i], i%2, i/2);
        }

        poolCardsPane = new GridPane();
        poolCardsPane.setVgap(3);
        poolCardsPane.setHgap(3);

        Text[] playerText = new Text[4];
        Text[] playerScoreText = new Text[4];
        Button[] playerCaptureButton = new Button[4];
        for(int i = 0; i < 4; i++){
            playerText[i] = new Text("Player " + (i+1));
            playerScoreText[i] = new Text("Score: 0.0");
```

```
            playerCaptureButton[i] = new Button("Capture");

            playerText[i].setFill(Color.WHITE);
            playerText[i].setStyle("-fx-font: 22 arial");
            playerScoreText[i].setFill(Color.WHITE);
            playerScoreText[i].setStyle("-fx-font: 15 arial");


        }

        VBox player2Pane = new VBox(10), player4Pane = new VBox(10);
        player2Pane.getChildren().addAll(playerText[1],        playerScoreText[1],
player2CardsPane, playerCaptureButton[1]);
        player4Pane.getChildren().addAll(playerText[3],        playerScoreText[3],
player4CardsPane, playerCaptureButton[3]);

        HBox player1Pane = new HBox(10), player3Pane = new HBox(10);
        VBox player1TextPane = new VBox(10), player3TextPane = new VBox(10);
        player1TextPane.getChildren().addAll(playerText[0], playerScoreText[0]);
        player3TextPane.getChildren().addAll(playerText[2], playerScoreText[2]);
        player1Pane.getChildren().addAll(player1TextPane,        player1CardsPane,
playerCaptureButton[0]);
        player3Pane.getChildren().addAll(player3TextPane,        player3CardsPane,
playerCaptureButton[2]);

        Pane[] playerPane = new Pane[4];
        playerPane[0] = player1Pane;
        playerPane[1] = player2Pane;
        playerPane[2] = player3Pane;
        playerPane[3] = player4Pane;

        BorderPane mainPane = new BorderPane();
        mainPane.setStyle("-fx-background-color: green");
        mainPane.setPadding(new Insets(23));
        mainPane.setBottom(player1Pane);
        mainPane.setLeft(player2Pane);
        mainPane.setTop(player3Pane);
        mainPane.setRight(player4Pane);
        mainPane.setCenter(poolCardsPane);
        mainPane.setPrefSize(1000, 700);
        BorderPane.setAlignment(poolCardsPane, Pos.CENTER);
        player1Pane.setAlignment(Pos.CENTER);
        player3Pane.setAlignment(Pos.CENTER);
        poolCardsPane.setAlignment(Pos.CENTER);
        for(int i = 0; i < 4; i++){
            BorderPane.setAlignment(playerPane[i], Pos.CENTER);
            playerPane[i].setVisible(false);
        }

        VBox capturesBoard = new VBox(20);
        Text captureBoardTitle = new Text("Capturing Status"), captureStatus = new
Text("No Capture");
        captureBoardTitle.setFill(Color.WHITE);
        captureBoardTitle.setStyle("-fx-font: 25 arial");
        captureStatus.setStyle("-fx-font: 25 arial; -fx-fill: #ea8f79;");
        capturesBoard.setAlignment(Pos.TOP_CENTER);
        capturesBoard.getChildren().addAll(captureBoardTitle, captureStatus);
        capturesBoard.setPadding(new Insets(20));
        capturesBoard.setPrefWidth(250);
```

```
capturesBoard.setStyle("-fx-padding: 10; -fx-background-color: green;" +
              "-fx-border-style: solid inside;" +
              "-fx-border-width: 2;" +
              "-fx-border-insets: 35;" +
              "-fx-border-radius: 5;" +
              "-fx-border-color: white;");

VBox[] playerCaptures = new VBox[4];
for(int i = 0; i < 4; i++){
    playerCaptures[i] = new VBox(5);
    capturesBoard.getChildren().add(playerCaptures[i]);
}


HBox mainWrapper = new HBox();
mainWrapper.getChildren().addAll(mainPane, capturesBoard);
// main menu design begin
VBox mainMenu = new VBox(30);

Text mainMenuText = new Text("Please Select Number of Player");
mainMenuText.setStyle("-fx-font: 25 arial");
mainMenuText.setFill(Color.WHITE);

HBox mainMenuSelection = new HBox(20);
Button[] playerSelectionButton = new Button[3];
for(int i = 0; i < playerSelectionButton.length; i++){
    playerSelectionButton[i] = new Button(i+2+"");
    mainMenuSelection.getChildren().add(playerSelectionButton[i]);
}

mainMenu.getChildren().addAll(mainMenuText, mainMenuSelection);

mainPane.setCenter(mainMenu);
BorderPane.setAlignment(mainMenu, Pos.CENTER);
mainMenu.setAlignment(Pos.CENTER);
mainMenuSelection.setAlignment(Pos.CENTER);
// main menu design end

// winning menu design begin
VBox winningMenu = new VBox(30);

Text winningText = new Text("");
winningText.setStyle("-fx-font: 25 arial");
winningText.setFill(Color.WHITE);

HBox winningSelection = new HBox(20);
Button winningContinue = new Button("New Game");
Button winningExit = new Button("Exit Game");

winningSelection.getChildren().addAll(winningContinue, winningExit);
winningMenu.getChildren().addAll(winningText, winningSelection);

BorderPane.setAlignment(winningMenu, Pos.CENTER);
winningMenu.setAlignment(Pos.CENTER);
winningSelection.setAlignment(Pos.CENTER);

winningExit.setOnAction(e -> {
    Platform.exit();
});
```

```
winningContinue.setOnAction(e -> {
    mainPane.setCenter(mainMenu);
});

// winning menu design end


Scene scene = new Scene(mainWrapper);
primaryStage.setTitle("An Interesting Card Game");
primaryStage.setScene(scene);
primaryStage.show();
/* GUI Design end */


/* Game Logic begin */
for(int i = 0; i < playerSelectionButton.length; i++){
    int p = i+2;
    playerSelectionButton[i].setOnAction(e -> {
        numberOfPlayer = p;
        players = new Player[numberOfPlayer];
        for(int j = 0; j < players.length; j++)
            players[j] = new Player();

        initializeRound(numberOfPlayer);

        currentPlayerIdx = 0;
        currentPlayer = players[currentPlayerIdx];

        setNextPlayer(currentPlayerIdx);

        mainPane.setCenter(poolCardsPane);

        for(int j = 0; j < 4; j++)
            playerCaptureButton[j].setDisable(true);
        playerCaptureButton[currentPlayerIdx].setDisable(false);

        for(int j = 0; j < 4; j++)
            if(j < numberOfPlayer)
                playerPane[j].setVisible(true);
            else
                playerPane[j].setVisible(false);

        for(int j = 0; j < 4; j++)
            playerScoreText[j].setText("Score: 0.0");

        // reset capture board
        captureStatus.setText("No Capture");
        for(int j = 0; j < 4; j++)
            playerCaptures[j].getChildren().clear();
        for(int j = 0; j < numberOfPlayer; j++){
            Text t = new Text("Player " + (j+1) + " Captures:");
            t.setStyle("-fx-font: 15 arial; -fx-fill: white;");
            playerCaptures[j].getChildren().add(t);
        }

    });
}
```

```
        class CaptureHandler implements EventHandler<ActionEvent> {
            public void handle(ActionEvent e) {
                Button currentControl = (Button)e.getSource();

                Card playedCard = currentPlayer.playChoosenCard();
                if(playedCard != null){
                    // trailing
                    if(playedCard != Card.getEmptyCard()){
                        poolCards.add(playedCard);
                        captureStatus.setText("Trailing");
                    }
                    // capture
                    else{
                        Capture currentCapture = currentPlayer.getLatestCapture();
                        for(Card c: currentCapture.getCaptureCards())
                            poolCards.remove(c);
                        captureStatus.setText("Captured            "            +
currentCapture.getCaptureName());

playerCaptures[currentPlayerIdx].getChildren().add(formCapturesImage(currentCaptur
e));
                    }

                    // successful capture, update player score
                    DecimalFormat df = new DecimalFormat("#0.0");
                    playerScoreText[currentPlayerIdx].setText("Score:      "     +
df.format(currentPlayer.getScore()));

                    // this player won
                    if(currentPlayer.getScore() >= WINNING_SCORE){
                        // game end
                        mainPane.setCenter(winningMenu);
                        winningText.setText("Player  " + (currentPlayerIdx+1) + "
wins !!!!");
                        playerCaptureButton[currentPlayerIdx].setDisable(true);
                        return;
                    }
                    // successful capture, next player
                    playerCaptureButton[currentPlayerIdx].setDisable(true);
                    currentPlayerIdx = (currentPlayerIdx + 1)%numberOfPlayer;
                    currentPlayer = players[currentPlayerIdx];

                    // if current player has no hand card, need new round
                    if(currentPlayer.getHandCard().length == 0){
                        initializeRound(numberOfPlayer);
                        // reset player capture board
                        for(int j = 0; j < numberOfPlayer; j++)
                            playerCaptures[j].getChildren().clear();
                        for(int j = 0; j < numberOfPlayer; j++){
                            Text t = new Text("Player " + (j+1) + " Captures:");
                            t.setStyle("-fx-font: 15 arial; -fx-fill: white;");
                            playerCaptures[j].getChildren().add(t);
                        }
                    }

                    setNextPlayer(currentPlayerIdx);
                    playerCaptureButton[currentPlayerIdx].setDisable(false);

                }
```

```
                else{
                    captureStatus.setText("Invalid Capture");
                }
            }
        }

        for(int i = 0; i < 4; i++){
            playerCaptureButton[i].setOnAction(new CaptureHandler());
        }

        /* Game Logic end */
    }

    public static void main(String[] args){
        launch(args);
    }

}
```

5.6 Pair

```
public class Pair extends Capture{
    public Pair(){};

    private Pair(Card handCard, Card[] poolCards){
        multiplier = 1;
        captureCards = new Card[poolCards.length+1];
        captureCards[0] = handCard;
        for(int i = 0; i < poolCards.length; i++){
            captureCards[i+1] = poolCards[i];
        }
    }

    public Capture formCapture(Card handCard, Card[] poolCards){
        if(poolCards.length != 1)
            return null;
        if(handCard.getRankValue() != poolCards[0].getRankValue())
            return null;

        return new Pair(handCard, poolCards);

    }
    public double getScore(){
        return multiplier*captureCards.length;
    }

    public String getCaptureName(){
        return "Pair";
    }

}
```

5.7 Player

```
import java.util.Arrays;
import java.util.ArrayList;

public class Player{
    double score = 0;
    ArrayList<Card> handCards = new ArrayList<Card>();
```

```
ArrayList<Card> choosenCards = new ArrayList<Card>();
ArrayList<Card> choosenHandCards = new ArrayList<Card>();
ArrayList<Capture> captures = new ArrayList<Capture>();

public void addHandCard(Card[] cards){
    for(int i = 0; i < cards.length; i++){
        handCards.add(cards[i]);
    }
}
public void removeHandCard(Card card){
    handCards.remove(card);
}
public void addHandCard(Card card){
    handCards.add(card);
}
public Card[] getHandCard(){
    Card[] returnCard = new Card[handCards.size()];
    for(int i = 0; i < returnCard.length; i++)
        returnCard[i] = handCards.get(i);
    Arrays.sort(returnCard);
    return returnCard;
}
public void addChoosenCard(Card card){
    if(handCards.contains(card))
        choosenHandCards.add(card);
    else
        choosenCards.add(card);
}

public boolean removeChoosenCard(Card card){
    if(handCards.contains(card)){
        return choosenHandCards.remove(card);
    }
    return choosenCards.remove(card);
}

// return card if trailling, empty card if capture, null otherwise
public Card playChoosenCard(){
    if(choosenHandCards.size() != 1)
        return null;

    if(choosenCards.size() == 0){
        // play trailling
        removeHandCard(choosenHandCards.get(0));
        Card returnCard = choosenHandCards.get(0);
        choosenHandCards.clear();
        return returnCard;
    }

    if(formCaptureChoosenCard() != null){
        Capture currentCapture = formCaptureChoosenCard();
        score += currentCapture.getScore();
        captures.add(currentCapture);

        removeHandCard(choosenHandCards.get(0));
        choosenCards.clear();
        choosenHandCards.clear();

        return Card.getEmptyCard();
```

```
        }

        return null;
    }
    private Capture formCaptureChoosenCard(){
        if(choosenHandCards.size() != 1 || choosenCards.size() == 0)
            return null;

        Card[] capturingCard = new Card[choosenCards.size()];
        for(int i = 0; i < choosenCards.size(); i++)
            capturingCard[i] = choosenCards.get(i);

        return Capture.formCaptures(choosenHandCards.get(0), capturingCard);
    }

    // to help main remove card from pool
    public Capture getLatestCapture(){
        if(captures.size() == 0)
            return null;
        return captures.get(captures.size()-1);
    }
    public double getScore(){
        return score;
    }

}
```

5.8 Run

```
import java.util.Arrays;

public class Run extends Capture{
    public Run(){};

    private Run(Card handCard, Card[] poolCards){
        multiplier = 1.5;
        captureCards = new Card[poolCards.length+1];
        captureCards[0] = handCard;
        for(int i = 0; i < poolCards.length; i++){
            captureCards[i+1] = poolCards[i];
        }
    }

    public Capture formCapture(Card handCard, Card[] poolCards){
        if(poolCards.length < 2)
            return null;

        int suit = handCard.getSuitValue();

        for(Card c: poolCards)
            if(c.getSuitValue() != suit)
                return null;

        int[] allCardRank = new int[poolCards.length+1];
        allCardRank[0] = handCard.getRankValue();
        for(int i = 0; i < poolCards.length; i++)
            allCardRank[i+1] = poolCards[i].getRankValue();

        Arrays.sort(allCardRank);
```

```
        for(int i = 1; i < allCardRank.length; i++)
            if(allCardRank[i] != allCardRank[i-1]+1)
                return null;

        return new Run(handCard, poolCards);
    }
    public double getScore(){
        return multiplier*captureCards.length;
    }

    public String getCaptureName(){
        return "Run";
    }
}
```

5.9 Straight

```
import java.util.Arrays;

public class Straight extends Capture{
    public Straight(){};

    private Straight(Card handCard, Card[] poolCards){
        multiplier = 1.3;
        captureCards = new Card[poolCards.length+1];
        captureCards[0] = handCard;
        for(int i = 0; i < poolCards.length; i++){
            captureCards[i+1] = poolCards[i];
        }
    }

    public Capture formCapture(Card handCard, Card[] poolCards){
        if(poolCards.length < 2)
            return null;

        int[] allCardRank = new int[poolCards.length+1];
        allCardRank[0] = handCard.getRankValue();
        for(int i = 0; i < poolCards.length; i++)
            allCardRank[i+1] = poolCards[i].getRankValue();

        Arrays.sort(allCardRank);
        for(int i = 1; i < allCardRank.length; i++)
            if(allCardRank[i] != allCardRank[i-1]+1)
                return null;

        return new Straight(handCard, poolCards);
    }
    public double getScore(){
        return multiplier*captureCards.length;
    }

    public String getCaptureName(){
        return "Straight";
    }
}
```

5.10 Triple

```
public class Triple extends Capture{
    public Triple(){};
```

```
    private Triple(Card handCard, Card[] poolCards){
        multiplier = 1;
        captureCards = new Card[poolCards.length+1];
        captureCards[0] = handCard;
        for(int i = 0; i < poolCards.length; i++){
            captureCards[i+1] = poolCards[i];
        }
    }

    public Capture formCapture(Card handCard, Card[] poolCards){
        if(poolCards.length != 2)
            return null;
        if(handCard.getRankValue() != poolCards[0].getRankValue()
            || handCard.getRankValue() != poolCards[1].getRankValue() )
            return null;

        return new Triple(handCard, poolCards);

    }
    public double getScore(){
        return multiplier*captureCards.length;
    }

    public String getCaptureName(){
        return "Triple";
    }

}
```