UNIVERSITI TUNKU ABDUL RAHMAN
FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

UCCD2044 Object-Oriented Programming Practices

Title: Card Game

Lecturer: Mr. Ng Hui Fuang

Group Members:

No.	Name	Student ID	Course	Class Group
1	Lee Wen Dick	1401837	CS	L1, P2

Table of Contents

Title	Page
Description	3
Activity Diagram	4
UML Class Diagrams	5
Java Code	
- Card	6
- Player	8
- Eight	9
- Jack	10
- Ace	11
- Queen	12
- IGame	13
- Game	14

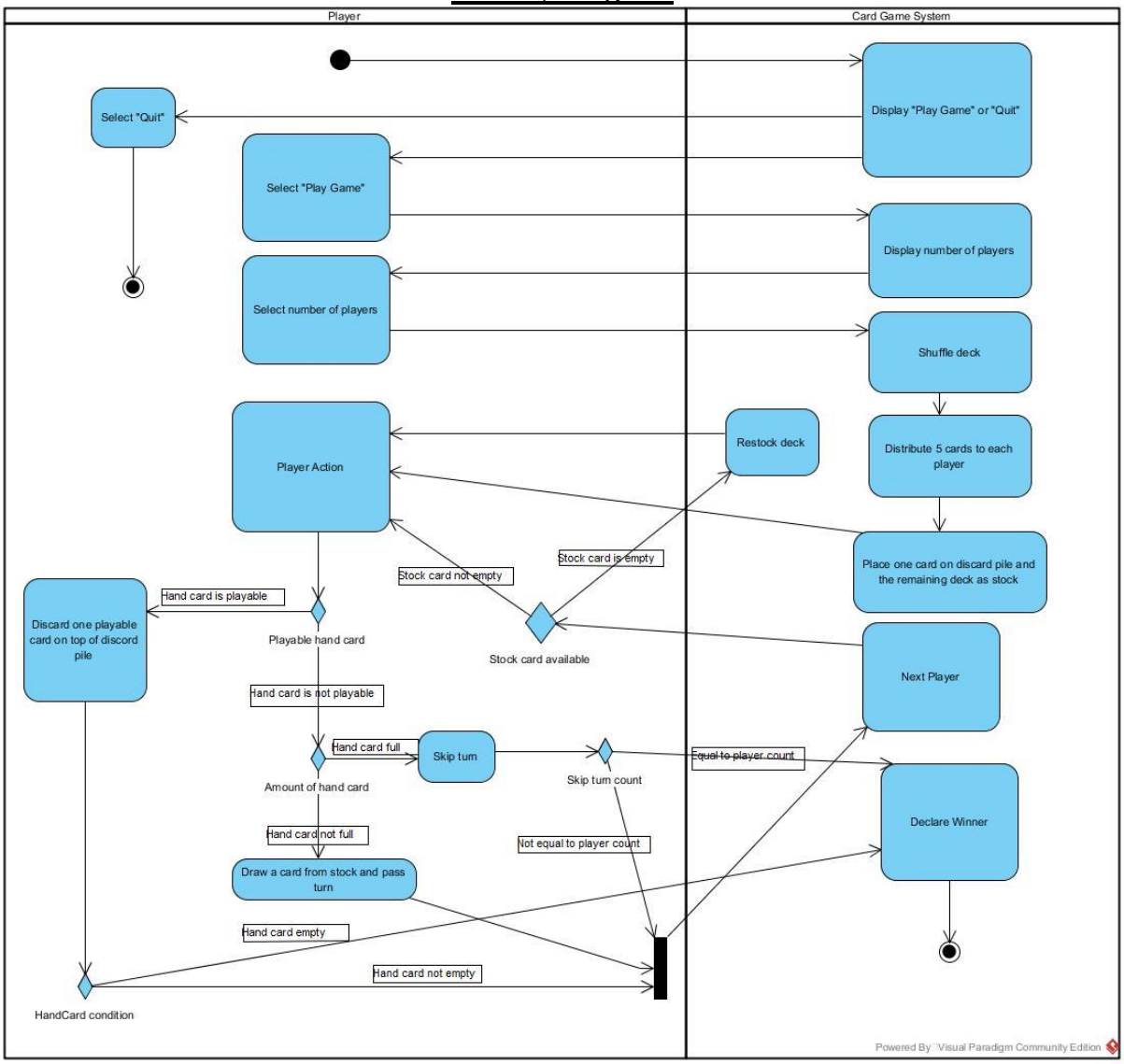
Description

The card game program is developed using Java and object-oriented programming concept with JavaFX GUI design. This game is played using standard 52-card deck. The objective of the game is to get rid of the cards in player's hand onto a discard pile by matching the rank or suit of the previous discard. The game is designed for 2-4 players and each player gets 5 cards. The card game contains special effects to some particular cards. The following ranks provide special actions:

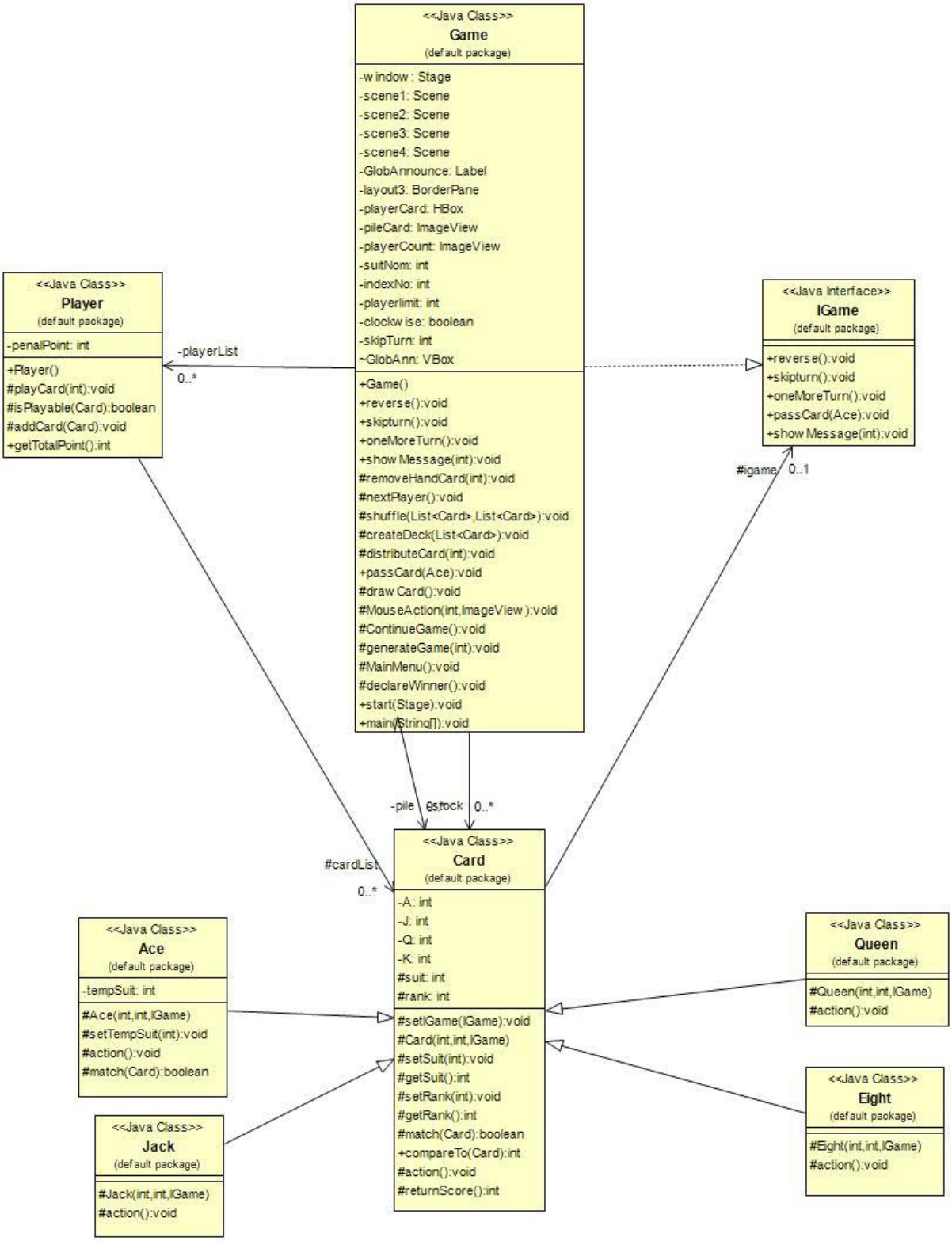
- Ace - An Ace can be played on any card, and the palyer of the Ace can nominate any suit.
- Queen - When a Queen is played, the next player loses a turn and the turn passes to the following player.
- Eight - When an Eight is played, the direction of play reverses.
- Jack - When a jack is played, one more additional turn is offered to the player. If the player do not have playable hand card, he/she has to draw a card and pass the turn.

In this card game program, the user is given selection of two players, three players, and four players. The game will start from player 1 once the number of players is decided. One of the feature in this program is the unplayable hand cards will be shaded to allow player to discard their playable hand cards conviniently. Simplicity is advocated in this game design, but every pictures, font type, font colors, and graphic designs are carefully considered and selected to achieve elegant design.

Activity Diagram



UML Class Diagrams



```java

// TODO: Auto-generated Javadoc
/**
 * The Class Card.
 */
public class Card implements Comparable<Card>{

    /** The igame. */
    protected IGame igame;

    /** The Constant A, J, Q, K. */
    private final static int A = 1, J = 11, Q = 12, K = 13;

    /** The suit. Spade = 0, Heart = 1, Club = 2, Diamond = 3 */
    protected int suit;

    /** The rank. */
    protected int rank;

    /**
     * Sets the i game.
     *
     * @param igame the new i game
     */
    protected void setIGame(IGame igame){
        this.igame = igame;
    }

    /**
     * Instantiates a new card.
     *
     * @param suit the suit
     * @param rank the rank
     * @param igame the igame
     */
    protected Card(int suit, int rank, IGame igame){
        this.suit = suit;
        this.rank = rank;
        this.igame = igame;
    }

    /**
     * Sets the suit.
     *
     * @param suit the new suit
     */
    protected void setSuit(int suit){
        this.suit = suit;
    }

    /**
     * Gets the suit.
     *
     * @return the suit
     */
    protected int getSuit(){
        return suit;
    }

```

```java
    /**
     * Sets the rank.
     *
     * @param rank the new rank
     */
    protected void setRank(int rank){
        this.rank = rank;
    }

    /**
     * Gets the rank.
     *
     * @return the rank
     */
    protected int getRank(){
        return rank;
    }

    /**
     * Card Matching.
     *
     * @param card the card
     * @return true, if successful
     */
    protected boolean match(Card card){
        if(rank == card.rank || suit == card.suit || card.rank == 1 )
            return true;
        else return false;
    }

    /* (non-Javadoc)
     * @see java.lang.Comparable#compareTo(java.lang.Object)
     */
    @Override
    public int compareTo(Card card){
        if (suit == card.suit) {
            return rank - card.rank;
        } else {
            return suit - card.suit;
        }
    }

    /**
     * Action.
     */
    protected void action(){}

    /**
     * Return score of the card.
     *
     * @return the int
     */
    protected int returnScore(){
        if(rank == A ) return 20;
        else if(rank == J || rank == Q || rank == K) return 10;
        else return rank;
    }
}
```

```java
import java.util.*;

// TODO: Auto-generated Javadoc
/**
 * The Class Player.
 */
public class Player {

    /** The card list. */
    protected List<Card> cardList = new ArrayList<Card>();
    /** The penalty point. */
    private int penalPoint = 0;

    /**
     * Play card.
     *
     * @param index the index
     */
    //Remove a hand card to the pile
    protected void playCard(int index){
        cardList.remove(index);
    }

    /**
     * Checks if is playable.
     *
     * @param card the card
     * @return true, if is playable
     */
    //Check any of the player's hand card is playable
    protected boolean isPlayable(Card card){
        for(int i = 0 ; i < cardList.size(); i++){
            if(card.match(cardList.get(i)))
                return true;
        }
        return false;
    }

    /**
     * Adds a card.
     *
     * @param card the card
     */
    protected void addCard(Card card){
        cardList.add(card);
    }

    /**
     * Gets the total point.
     *
     * @return the total point
     */
    public int getTotalPoint(){
        for(int i = 0 ; i < cardList.size(); i++){
            penalPoint += cardList.get(i).returnScore();
        }
        return penalPoint;
    }
}
```

```java

// TODO: Auto-generated Javadoc
/**
 * The Class Eight.
 */
public class Eight extends Card{

    /**
     * Instantiates a new eight.
     *
     * @param suit the suit
     * @param rank the rank
     * @param igame the igame
     */
    protected Eight(int suit, int rank, IGame igame) {
        super(suit, rank, igame);
    }

    /* (non-Javadoc)
     * @see Card#action()
     */
    @Override
    protected void action(){
        igame.reverse();
        igame.showMessage(1);
    }
}
```

```java

// TODO: Auto-generated Javadoc
/**
 * The Class Jack.
 */
public class Jack extends Card{

    /**
     * Instantiates a new jack.
     *
     * @param suit the suit
     * @param rank the rank
     * @param igame the igame
     */
    protected Jack(int suit, int rank, IGame igame) {
        super(suit, rank, igame);
    }

    /* (non-Javadoc)
     * @see Card#action()
     */
    @Override
    protected void action(){
        igame.showMessage(2);
        igame.oneMoreTurn();
    }
}
```

```java

// TODO: Auto-generated Javadoc
/**
 * The Class Ace.
 */
public class Ace extends Card{

    /** The temporary suit. */
    private int tempSuit = 4;

    /**
     * Instantiates a new ace.
     *
     * @param suit the suit
     * @param rank the rank
     * @param igame the igame
     */
    protected Ace(int suit, int rank, IGame igame) {
        super(suit, rank, igame);
    }

    /**
     * Sets the temporary suit.
     *
     * @param tempSuit the new temporary suit
     */
    protected void setTempSuit(int tempSuit){
        this.tempSuit = tempSuit;
    }

    /* (non-Javadoc)
     * @see Card#action()
     */
    @Override
    protected void action(){
        igame.passCard(this);
    }

    /* (non-Javadoc)
     * @see Card#match(Card)
     */
    @Override
    protected boolean match(Card card){
        if(tempSuit == 4 && ( rank == card.rank || suit == card.suit )){
             return true;
        }
        else if(tempSuit != 4 && ( rank == card.rank || tempSuit == card.suit)){
             return true;
        }
        else return false;
    }
}
```

```java

// TODO: Auto-generated Javadoc
/**
 * The Class Queen.
 */
public class Queen extends Card{

    /**
     * Instantiates a new queen.
     *
     * @param suit the suit
     * @param rank the rank
     * @param igame the igame
     */
    protected Queen(int suit, int rank, IGame igame) {
        super(suit, rank, igame);
    }

    /* (non-Javadoc)
     * @see Card#action()
     */
    @Override
    protected void action(){
        igame.skipturn();
        igame.showMessage(0);
    }
}

```

```java

// TODO: Auto-generated Javadoc
/**
 * The Interface IGame.
 */
public interface IGame {

    /**
     * Reverse.
     */
    public abstract void reverse();

    /**
     * Skip turn.
     */
    public abstract void skipturn();

    /**
     * One more turn.
     */
    public abstract void oneMoreTurn();

    /**
     * Pass card.
     *
     * @param card the card
     */
    public abstract void passCard(Ace card);

    /**
     * Show message.
     *
     * @param choice the choice
     */
    public abstract void showMessage(int choice);
    }


```

```java
import java.util.*;

// TODO: Auto-generated Javadoc
/**
 * The Class Game.
 */
public class Game extends Application implements IGame {

    /** The window. */
    private Stage window;

    /** The scene 1, scene 2, scene 3 and scene 4. */
    private Scene scene1, scene2, scene3, scene4;

    /**  Notify about special card activity and game activity,. */
    private Label GlobAnnounce;

    /**  The third scene. */
    private BorderPane layout3 = new BorderPane();

    /** Store and display player's hand card. */
    private HBox playerCard = new HBox();

    /** Display the current top pile card. */
    private ImageView pileCard = new ImageView();

    /** Display the current player's number.  */
    private ImageView playerCount = new ImageView();

    /**  Default suit(4) or nominated suit(not 4). */
    private int suitNom = 4;

    /** The player's turn's index number. */
    private int indexNo = 0;

    /** The number of player. */
    private static int playerlimit;

    /** The current direction. */
    private boolean clockwise = true;

    /** The skip turn count. */
    private int skipTurn = 0; //

    /** The player list. */
    private List<Player> playerList = new ArrayList<Player>();

    /** The stock. */
    private List<Card> stock = new ArrayList<Card>();

    /** The pile. */
    private List<Card> pile = new ArrayList<Card>();

    /** The Global announcement. */
    VBox GlobAnn = new VBox();
    /**
     * Instantiates a new game.
     */
    public Game(){}
```

```
 75
 76     /* (non-Javadoc)
 77      * @see IGame#reverse()
 78      * Reverse the direction of players' turn
 79      */
 80     public void reverse(){
 81         if(clockwise) clockwise = false;
 82         else clockwise = true;
 83     }
 84
 85     /* (non-Javadoc)
 86      * @see IGame#skipturn()
 87      * Skip the next player
 88      */
 89     public void skipturn(){
 90         nextPlayer();
 91     }
 92
 93     /* (non-Javadoc)
 94      * @see IGame#oneMoreTurn()
 95      */
 96     public void oneMoreTurn() {
 97         if(clockwise){
 98             if(indexNo == 0) indexNo = playerlimit - 1;
 99             else indexNo--;
100         }
101         else{
102             if(indexNo == playerlimit - 1 ) indexNo = 0;
103             else indexNo++;
104         }
105     }
106
107     /* (non-Javadoc)
108      * @see IGame#showMessage(int)
109      * Display activity
110      */
111     public void showMessage(int choice){
112
113         if(choice == 0){ //A queen is drawn
114             GlobAnnounce = new Label("PLAYER " + (indexNo+1) + " SKIPS THE TURN");
115         }
116         else if(choice == 1){ //An Eight is drawn
117             GlobAnnounce = new Label("DIRECTION REVERSED");
118         }
119         else if(choice == 2){
120             GlobAnnounce = new Label("PLAYER " + (indexNo+1) + " HAS ONE MORE TURN");
121         }
122         GlobAnnounce.setFont(Font.font("Arial", FontWeight.BOLD, 30));
123         GlobAnnounce.setTextFill(Color.web("#DAA520"));
124         GlobAnn.getChildren().add(GlobAnnounce);
125         GlobAnn.setMinSize(0, 0);
126         GlobAnn.setAlignment(Pos.CENTER);
127         FadeTransition fader = new FadeTransition(Duration.millis(1500), GlobAnnounce
    );
128         fader.setFromValue(1.0);
129         fader.setToValue(0.5);
130         fader.play();
131         fader.setOnFinished(e-> {
132             GlobAnn.getChildren().clear();
```

```
133             layout3.getChildren().remove(GlobAnn);
134         });
135         layout3.setCenter(GlobAnn);
136     }
137 
138     /**
139      * Removes the hand card and put in the pile.
140      *
141      * @param pos the pos
142      */
143     protected void removeHandCard(int pos){
144         pile.add(0,playerList.get(indexNo).cardList.remove(pos));
145     }
146 
147 
148     /**
149      * Update indexNo for the next player's turn.
150      */
151     protected void nextPlayer(){
152         if(clockwise){
153             indexNo = (indexNo + 1) % playerlimit;
154         }
155         else{
156             if(indexNo == 0 )
157                 indexNo = playerlimit - 1;
158             else indexNo--;
159         }
160     }
161 
162 
163     /**
164      * Shuffle.
165      *
166      * @param stock the stock
167      * @param pile the pile
168      */
169     protected void shuffle(List<Card> stock, List<Card> pile){
170         Card tempPile = null;
171         if(!pile.isEmpty()) {
172             tempPile = (pile.remove(0));
173         }
174         stock.addAll(pile);
175         pile.clear();
176         if(tempPile != null){
177         pile.add(tempPile);
178         }
179         Collections.shuffle(stock);
180         if(pile.isEmpty()){
181             pile.add(stock.remove(0));
182             }
183     }
184 
185     /**
186      * Creates the deck.
187      *
188      * @param stock the stock
189      */
190     protected void createDeck(List<Card> stock){
191         for(int i = 0 ; i < 52 ; i++){
```

```
192             if((i%13)+1 == 1) stock.add(new Ace(i/13, 1, this));
193             else if((i%13)+1 == 12) stock.add(new Queen(i/13, 12, this));
194             else if((i%13)+1 == 8) stock.add(new Eight(i/13, 8, this));
195             else if((i%13)+1 == 11) stock.add(new Jack(i/13, 11, this));
196             else stock.add(new Card(i/13, (i%13)+1, this));
197         }
198     }
199 
200     /**
201      * Create player instances, distribute card and sort card.
202      *
203      * @param noOfPlayer the no of player
204      */
205     protected void distributeCard(int noOfPlayer){
206         for(int i =  0 ; i < noOfPlayer; i++){
207             playerList.add(new Player());
208             for(int j = 0 ; j < 5 ; j++){
209                 playerList.get(i).addCard(stock.remove(0));
210             }
211             Collections.sort(playerList.get(i).cardList);
212         }
213     }
214 
215     /* (non-Javadoc)
216      * @see IGame#passCard(Ace)
217      * set temporary suit after player nominate a particular suit
218      */
219 
220     public void passCard(Ace card){
221         playerCard.getChildren().clear();
222         HBox SuitNom = new HBox();
223         Player curplayer = playerList.get(indexNo);
224         for(int i = 0  ; i < curplayer.cardList.size() ; i++){
225             ImageView handCard = new ImageView();
226             ImageView overlay = new ImageView();
227             overlay.setImage(new Image("overlay.png"));
228             handCard.setImage(new
    Image(String.format("%d%d.png",curplayer.cardList.get(i).getSuit(),
    curplayer.cardList.get(i).getRank())));
229             handCard.setFitWidth(120);
230             handCard.setPreserveRatio(true);
231             handCard.setCache(true);
232             overlay.setFitWidth(121);
233             overlay.setPreserveRatio(true);
234             overlay.setCache(true);
235             Group unplayable = new Group();
236             unplayable.setBlendMode(BlendMode.SRC_OVER);
237             unplayable.getChildren().addAll(handCard, overlay);
238             playerCard.getChildren().add(unplayable);
239             playerCard.setAlignment(Pos.BOTTOM_CENTER);
240             layout3.setBottom(playerCard);
241         }
242         for(int i = 0 ; i < 4 ; i++){
243             final int temp = i;
244             ImageView suitCard = new ImageView();
245             suitCard.setImage(new Image(String.format("%d.png", i)));
246             SuitNom.setSpacing(10);
247             SuitNom.getChildren().add(suitCard);
248             SuitNom.setAlignment(Pos.CENTER);
```

```
249             layout3.setCenter(SuitNom);
250             suitCard.setFitWidth(120);
251             suitCard.setPreserveRatio(true);
252             suitCard.setCache(true);
253 
254             //Bind events to suitCard
255             suitCard.setOnMouseClicked(e -> {
256                 suitNom = temp;
257                 card.setTempSuit(suitNom);
258                 pileCard.setImage(new Image(String.format("%d%d.png", suitNom, 1)));
259                 pileCard.setFitWidth(120);
260                 pileCard.setPreserveRatio(true);
261                 pileCard.setCache(true);
262                 playerCount.setImage(new Image(String.format("Player %d.png",
    (indexNo+1))));
263                 playerCount.setFitWidth(160);
264                 playerCount.setPreserveRatio(true);
265                 playerCount.setCache(true);
266                 SuitNom.getChildren().clear();
267                 ContinueGame();
268             });
269         }
270     }
271 
272     /**
273      * Draw card.
274      */
275     protected void drawCard(){
276         //VBox GlobAnn = new VBox(); //A VBox to store and display several events
    sequentially
277         if(playerList.get(indexNo).cardList.size() == 5){
278             GlobAnnounce = new Label("PLAYER " + (indexNo+1) + " SKIPS");
279             skipTurn++;
280         }
281         else{
282             playerList.get(indexNo).addCard(stock.remove(0));
283             Collections.sort(playerList.get(indexNo).cardList);
284             GlobAnnounce = new Label("PLAYER " + (indexNo+1) + " DRAWS A CARD AND
    PASSES THE TURN");
285             skipTurn = 0;
286         }
287         GlobAnnounce.setFont(Font.font("Arial", FontWeight.BOLD, 30));
288         GlobAnnounce.setTextFill(Color.web("#DAA520"));
289         GlobAnn.getChildren().add(GlobAnnounce);
290         GlobAnn.setAlignment(Pos.CENTER);
291         FadeTransition fader = new FadeTransition(Duration.millis(1500), GlobAnnounce
    );
292         fader.setFromValue(1.0);
293         fader.setToValue(0.5);
294         fader.play();
295         fader.setOnFinished(e-> {
296             GlobAnn.getChildren().clear();
297             layout3.getChildren().remove(GlobAnn);
298         });
299         layout3.setCenter(GlobAnn);
300     }
301 
302     /**
303      * Mouse action binding for the ContinueGame method.
```

```
304      *
305      * @param index the index
306      * @param image the image
307      */
308     protected void MouseAction(int index, ImageView image){
309         final int idx = index;
310         Player curplayer = playerList.get(indexNo);
311         image.setOnMouseClicked(e -> {
312             if(pile.get(0).match(curplayer.cardList.get(idx)) &&
    curplayer.cardList.get(idx) instanceof Ace){
313                 removeHandCard(idx);
314                 pile.get(0).action();
315                 skipTurn  = 0;
316                 nextPlayer();
317                 if(curplayer.cardList.isEmpty()){
318                     declareWinner();
319                 }
320             }
321             else if(pile.get(0).match(curplayer.cardList.get(idx)) && !
    (curplayer.cardList.get(idx) instanceof Ace)){
322                 removeHandCard(idx);
323                 pile.get(0).action();
324                 skipTurn  = 0;
325                 nextPlayer();
326                 pileCard.setImage(new Image(String.format("%d%d.png",
    pile.get(0).getSuit(), pile.get(0).getRank())));
327                 pileCard.setFitWidth(120);
328                 pileCard.setPreserveRatio(true);
329                 pileCard.setCache(true);
330                 playerCount.setImage(new Image(String.format("Player %d.png",
    (indexNo+1))));
331                 playerCount.setFitWidth(160);
332                 playerCount.setPreserveRatio(true);
333                 playerCount.setCache(true);
334                 if(curplayer.cardList.isEmpty()){
335                     declareWinner();
336                 }
337                 else ContinueGame();
338             }
339         });
340     }
341
342     /**
343      * Continue the game.
344      */
345     protected void ContinueGame(){
346         playerCard.setSpacing(10);
347         while(skipTurn != playerlimit){
348             if(stock.size()==0){
349                 shuffle(stock, pile);
350             }
351             if(playerList.get(indexNo).isPlayable(pile.get(0))){
352                 Player curplayer = playerList.get(indexNo);
353                 playerCard.getChildren().clear();
354                 for(int i = 0  ; i < curplayer.cardList.size() ; i++){
355                     ImageView handCard = new ImageView();
356                     ImageView overlay = new ImageView();
357                     overlay.setImage(new Image("overlay.png"));
358                     handCard.setImage(new
```

```
    Image(String.format("%d%d.png",curplayer.cardList.get(i).getSuit(),
    curplayer.cardList.get(i).getRank())));
359                     handCard.setFitWidth(120);
360                     handCard.setPreserveRatio(true);
361                     handCard.setCache(true);
362                     overlay.setFitWidth(121);
363                     overlay.setPreserveRatio(true);
364                     overlay.setCache(true);
365
366                     //Top pile card is Ace and it is not a nominated Ace
367                     if(pile.get(0) instanceof Ace && suitNom==4){
368
369                         //The hand card match the pile normal Ace
370                         if(pile.get(0).match(curplayer.cardList.get(i)) ||
    curplayer.cardList.get(i) instanceof Ace){
371                             playerCard.getChildren().add(handCard);
372                             playerCard.setAlignment(Pos.BOTTOM_CENTER);
373                             layout3.setBottom(playerCard);
374                             MouseAction(i, handCard);
375                         }
376
377                         //Pile's normal Ace do not match the particular hand card
378                         else{
379                             Group unplayable = new Group();
380                             unplayable.setBlendMode(BlendMode.SRC_OVER);
381                             unplayable.getChildren().addAll(handCard, overlay);
382                             playerCard.getChildren().add(unplayable);
383                             playerCard.setAlignment(Pos.BOTTOM_CENTER);
384                             layout3.setBottom(playerCard);
385                             MouseAction(i, overlay);
386                         }
387                     }
388
389                     //Top pile card is Ace and is nominated Ace
390                     else if(pile.get(0) instanceof Ace && suitNom !=4){
391                         //The hand card match the pile nominated Ace
392                         if(pile.get(0).match(curplayer.cardList.get(i)) ||
    curplayer.cardList.get(i) instanceof Ace ){
393
394                             playerCard.getChildren().add(handCard);
395                             playerCard.setAlignment(Pos.BOTTOM_CENTER);
396                             layout3.setBottom(playerCard);
397                             MouseAction(i, handCard);
398                         }
399                         //The hand card does not match the pile nominated Ace
400                         else{
401                             Group unplayable = new Group();
402                             unplayable.setBlendMode(BlendMode.SRC_OVER);
403                             unplayable.getChildren().addAll(handCard, overlay);
404                             playerCard.getChildren().add(unplayable);
405                             playerCard.setAlignment(Pos.BOTTOM_CENTER);
406                             layout3.setBottom(playerCard);
407                             MouseAction(i, overlay);
408                         }
409                     }
410
411                     //Other type of card
412                     else{
413                         //The hand card matches the pile card
```

```
414                         if(pile.get(0).match(curplayer.cardList.get(i))||
    curplayer.cardList.get(i) instanceof Ace){
415
416                             playerCard.getChildren().add(handCard);
417                             playerCard.setAlignment(Pos.BOTTOM_CENTER);
418                             layout3.setBottom(playerCard);
419                             MouseAction(i, handCard);
420                         }
421                         //The hand card do not match the pile card
422                         else{
423                             Group unplayable = new Group();
424                             unplayable.setBlendMode(BlendMode.SRC_OVER);
425                             unplayable.getChildren().addAll(handCard, overlay);
426                             playerCard.getChildren().add(unplayable);
427                             playerCard.setAlignment(Pos.BOTTOM_CENTER);
428                             layout3.setBottom(playerCard);
429                             MouseAction(i, overlay);
430                         }
431                     }
432                 }
433
434                 break;
435             }
436             else{
437                 drawCard();
438                 nextPlayer();
439                 playerCount.setImage(new Image(String.format("Player %d.png",
    (indexNo+1))));
440                 playerCount.setFitWidth(160);
441                 playerCount.setPreserveRatio(true);
442                 playerCount.setCache(true);
443
444             }
445         }
446         if(skipTurn == playerlimit){
447             declareWinner();
448         }
449     }
450
451     /**
452      * Generate game.
453      *
454      * @param totalP the total player number
455      */
456     protected void generateGame(int totalP){
457         createDeck(stock);
458         shuffle(stock, pile);
459         distributeCard(playerlimit);
460
461         layout3.setId("ThirdScene");
462         scene3 = new Scene(layout3,  1400, 800);
463         scene3.getStylesheets().addAll(this.getClass().getResource("style.css").toExte
    rnalForm());
464
465         //Hold stockCard and pileCard
466         HBox gameCard = new HBox();
467         gameCard.setSpacing(10);
468
469         playerCount.setImage(new Image("Player 1.png"));
```

```
470         playerCount.setFitWidth(160);
471         playerCount.setPreserveRatio(true);
472         playerCount.setCache(true);
473 
474         //Decorated stock card
475         ImageView stockCard = new ImageView();
476         stockCard.setImage(new Image("Cover.png"));
477         stockCard.setFitWidth(120);
478         stockCard.setPreserveRatio(true);
479         stockCard.setCache(true);
480 
481         pileCard.setImage(new Image(String.format("%d%d.png", pile.get(0).getSuit(),
    pile.get(0).getRank())));
482         pileCard.setFitWidth(120);
483         pileCard.setPreserveRatio(true);
484         pileCard.setCache(true);
485 
486         gameCard.getChildren().addAll(playerCount, stockCard, pileCard);
487         gameCard.setAlignment(Pos.CENTER);
488         layout3.setTop(gameCard);
489         ContinueGame();
490     }
491 
492     /**
493      * Main menu when the program is launched.
494      */
495     protected void MainMenu(){
496         //Start button
497         ImageView startButton = new ImageView();
498         startButton.setImage(new Image("PLAY GAME.png"));
499         startButton.setOnMouseClicked(e -> window.setScene(scene2));
500         startButton.setOnMouseEntered(e->{
501             startButton.setImage(new Image("PLAY GAME HOVER.png"));
502         });
503         startButton.setOnMouseExited(e->{
504             startButton.setImage(new Image("PLAY GAME.png"));
505         });
506 
507         //Exit button
508         ImageView exitButton = new ImageView();
509         exitButton.setImage(new Image("QUIT GAME.png"));
510         exitButton.setOnMouseClicked(e->System.exit(0));
511         exitButton.setOnMouseEntered(e->{
512             exitButton.setImage(new Image("QUIT GAME HOVER.png"));
513         });
514         exitButton.setOnMouseExited(e->{
515             exitButton.setImage(new Image("QUIT GAME.png"));
516         });
517 
518 
519         //Back button
520         ImageView backButton = new ImageView();
521         backButton.setImage(new Image("BACK.png"));
522         backButton.setOnMouseClicked(e-> window.setScene(scene1));
523         backButton.setOnMouseEntered(e->{
524             backButton.setImage(new Image("BACK HOVER.png"));
525         });
526         backButton.setOnMouseExited(e->{
527             backButton.setImage(new Image("BACK.png"));
```

```
528         });
529 
530         BorderPane layout1 = new BorderPane();
531         layout1.setId("FirstScene");
532         VBox startexitBox = new VBox();
533         startexitBox.setSpacing(10);
534         startexitBox.getChildren().addAll(startButton, exitButton);
535         startexitBox.setAlignment(Pos.CENTER);
536         layout1.setCenter(startexitBox);
537 
538         scene1 = new Scene(layout1, 1400, 800);
539         scene1.getStylesheets().addAll(this.getClass().getResource("style.css").toExte
    rnalForm());
540 
541         //Two player button
542         ImageView player2 = new ImageView();
543         player2.setImage(new Image("TWO.png"));
544         //Binding events
545         player2.setOnMouseClicked(e-> {
546             playerlimit = 2;
547             generateGame(playerlimit);
548             window.setScene(scene3);
549             });
550         //Binding events
551         player2.setOnMouseEntered(e->{
552             player2.setImage(new Image("TWO HOVER.png"));
553         });
554         //Binding events
555         player2.setOnMouseExited(e->{
556             player2.setImage(new Image("TWO.png"));
557         });
558 
559         //Three player button
560         ImageView player3 = new ImageView();
561         player3.setImage(new Image("THREE.png"));
562         //Binding events
563         player3.setOnMouseClicked(e-> {
564             playerlimit = 3;
565             generateGame(playerlimit);
566             window.setScene(scene3);
567             });
568         //Binding events
569         player3.setOnMouseEntered(e->{
570             player3.setImage(new Image("THREE HOVER.png"));
571         });
572         player3.setOnMouseExited(e->{
573             player3.setImage(new Image("THREE.png"));
574         });
575 
576         //Four player button
577         ImageView player4 = new ImageView();
578         player4.setImage(new Image("FOUR.png"));
579         player4.setOnMouseClicked(e-> {
580             playerlimit = 4;
581             generateGame(playerlimit);
582             window.setScene(scene3);
583             });
584         player4.setOnMouseEntered(e->{
585             player4.setImage(new Image("FOUR HOVER.png"));
```

```
586         });
587         player4.setOnMouseExited(e->{
588             player4.setImage(new Image("FOUR.png"));
589         });
590 
591         //layout2
592         BorderPane layout2 = new BorderPane();
593         layout2.setId("SecondScene");
594 
595         //A VBox to store and display the 2-4 player selection button for user to make
    selection
596         VBox amtPlayerSelectionBox = new VBox();
597         amtPlayerSelectionBox.setSpacing(10);
598         amtPlayerSelectionBox.getChildren().addAll(player2, player3, player4,
    backButton);
599         amtPlayerSelectionBox.setAlignment(Pos.CENTER);
600         layout2.setCenter(amtPlayerSelectionBox);
601         scene2 = new Scene(layout2, 1400, 800);
602         scene2.getStylesheets().addAll(this.getClass().getResource("style.css").toExte
    rnalForm());
603 
604         //Display scene 1 at first
605         window.setScene(scene1);
606     }
607 
608     /**
609      * Declare winner and display the player score list.
610      */
611     protected void declareWinner(){
612         //VBox to hold the playerStatus's HBoxes
613         VBox mainPlayerBox = new VBox();
614         mainPlayerBox.setSpacing(10);
615         mainPlayerBox.setAlignment(Pos.CENTER);
616         BorderPane layout4 = new BorderPane();
617         layout4.setId("FourthScene");
618 
619         //Identify winner
620         int WinnerIndex = 0, tempMin = 999999999, minScore;
621 
622         for(int i = 0 ; i < playerlimit; i++){
623             minScore = tempMin;
624             tempMin = Math.min(playerList.get(i).getTotalPoint(), minScore);
625             if(minScore != tempMin){
626                 WinnerIndex = i;
627             }
628         }
629 
630         for(int i = 0 ; i < playerlimit; i++){
631             //HBox to store and display the winner ImageView, player index and the
    player's score
632             HBox playerStatus = new HBox();
633             playerStatus.setMinWidth(350);
634             playerStatus.setSpacing(10);
635             playerStatus.setAlignment(Pos.CENTER_LEFT);
636             Label playerindex = new Label("Player " + (i+1) + "           ");
637             playerindex.setFont(Font.font("Cambria", FontPosture.ITALIC, 40));
638             Label playerScore = new Label("Penalty Points: " +
    playerList.get(i).getTotalPoint());
639             playerScore.setFont(Font.font("Cambria", FontPosture.ITALIC, 40));
```

```
640             if(WinnerIndex == i){
641                 //Store and display image of winner stamp
642                 ImageView winner = new ImageView();
643                 winner.setImage(new Image("Winner Stamp.png"));
644                 playerStatus.getChildren().addAll(winner, playerindex,playerScore);
645             }
646             else{
647                 //a dummy image for consistency alignment for the GUI
648                 ImageView dummy = new ImageView();
649                 dummy.setImage(new Image("dummy.png"));
650                 playerStatus.getChildren().addAll(dummy, playerindex,playerScore);
651             }
652 
653             //a dummy VBox for consistency alignment for the GUI
654             VBox dummmy2 = new VBox();
655             dummmy2.setPrefSize(370,200);
656             mainPlayerBox.getChildren().addAll(playerStatus);
657             layout4.setLeft(dummmy2);
658             layout4.setCenter(mainPlayerBox);
659         }
660 
661         //Buttons
662         HBox buttons = new HBox();
663         buttons.setSpacing(10);
664 
665         //Main menu button
666         ImageView MainMenu = new ImageView();
667         MainMenu.setImage(new Image("MAIN MENU.png"));
668         //Binding events
669         MainMenu.setOnMouseClicked(e-> {
670             window.close();
671             Game game = new Game();
672             game.start(new Stage());
673             });
674         //Binding events
675         MainMenu.setOnMouseEntered(e->{
676             MainMenu.setImage(new Image("MAIN MENU HOVER.png"));
677         });
678         //Binding events
679         MainMenu.setOnMouseExited(e->{
680             MainMenu.setImage(new Image("MAIN MENU.png"));
681         });
682 
683         //Exit button
684         ImageView exitButton = new ImageView();
685         exitButton.setImage(new Image("QUIT GAME.png"));
686         exitButton.setOnMouseClicked(e->System.exit(0));
687         //Binding events
688         exitButton.setOnMouseEntered(e->{
689             exitButton.setImage(new Image("QUIT GAME HOVER.png"));
690         });
691         //Binding events
692         exitButton.setOnMouseExited(e->{
693             exitButton.setImage(new Image("QUIT GAME.png"));
694         });
695 
696         //Dummy image
697         ImageView dummy3 = new ImageView();
698         dummy3.setImage(new Image("dummy3.png"));
```

```
699         buttons.getChildren().addAll(dummy3 ,MainMenu, exitButton);
700         mainPlayerBox.getChildren().addAll(buttons);
701 
702         scene4 = new Scene(layout4, 1400, 800);
703         scene4.getStylesheets().addAll(this.getClass().getResource("style.css").toExte
    rnalForm());
704         window.setScene(scene4);
705     }
706 
707     /* (non-Javadoc)
708      * @see javafx.application.Application#start(javafx.stage.Stage)
709      */
710     @Override
711     public void start(Stage primaryStage){
712         window = primaryStage;
713         MainMenu();
714         window.setTitle("Poker Game");
715         window.show();
716     }
717 
718     /**
719      * The main method.
720      *
721      * @param args the arguments
722      */
723     public static void main (String[] args) {
724         launch(args);
725     }
726 }
```