import java.awt.event.\*;

import java.util.Random;

import javax.swing.\*;

public class Deck {

// constants

final int deckSize = 52 ;

final int maxSuits = 4 ;

final int maxCards = 13 ;

final int HAND = 5 ;

protected class Card {

public int faceVal ; // contains int values 1 - 13

public int suit ; // contains int values 1 - 4

public boolean exist ;// shows if the card has been displayed

public Card(int face, int suitNum){

faceVal = face ;

suit = suitNum ;

exist = false ;

} // end card constructor

// function: setTrue

// pre: none

// post: sets the existance bool to true; helper function used to prevent duplicate cards

public void setTrue() {

this.exist = true ;

} // end setTrue

// function: setFalse

// pre: none

// post: sets the existance bool to false; helper function used to reset cards

public void setFalse() {

this.exist = false ;

} // end setTrue

} // end Card class

private Card[] Deck ;

// A deck object consists of a Card array

public Deck(){

int cardCount = 0 ;

Deck = new Card[deckSize] ;

for (int i = 0; i < maxSuits; i++){ // sets up 13 cards of each suit

for (int j = 0; j < maxCards; j++){

Deck[cardCount] = new Card(j + 1, i + 1);

++cardCount ;

} // end for

}

} // end Deck constructor

// function: draw

// pre: none

// post: returns an integer of 0 - 51 to represent 1 of 52 cards

public int draw() {

// generates a value 0 - 51 to represent one

// of the 52 cards available

Random generator = new Random() ;

return generator.nextInt(52) ;

} // end draw

// function: getExist

// pre: must pass in Card object

// post: returns a bool from the card to see if card has been displayed

public boolean getExist(Card pick){

return pick.exist ;

}

// function: getSuit

// pre: must pass in Card object

// post: returns a string of containing suit type of card

public String getSuit(Card pick){

switch (pick.suit)

{

case 1: return "Clubs" ;

case 2: return "Hearts" ;

case 3: return "Spades" ;

case 4: return "Diamonds" ;

} // end switch

return "\n\n Suit type does not exist \n\n" ;

} // end getSuit

// function: getFaceVal

// pre: must pass in Card object

// post: returns a string of containing face value of card

public String getFaceVal(Card pick) {

switch (pick.faceVal)

{

case 1: return " Ace" ;

case 2: return " 2" ;

case 3: return " 3" ;

case 4: return " 4" ;

case 5: return " 5" ;

case 6: return " 6" ;

case 7: return " 7" ;

case 8: return " 8" ;

case 9: return " 9" ;

case 10: return " 10";

case 11: return " Jack" ;

case 12: return "Queen" ;

case 13: return " King" ;

} // end switch

return "\n\n Face Value does not exist \n\n" ;

} // end getFaceVal

// function: getCard

// pre: must pass in an integer

// post: returns a a card from the deck object

public Card getCard(int hold) {

return Deck[hold] ;

} // end getCard

// runs through deck to find matching card and returns the index

public int getHold(Card c) {

int count = 0 ;

for (int i = 0 ; Deck[i].faceVal != c.faceVal || Deck[i].suit != c.suit; i++){

count++ ;

} // end for

return count ;

} // end getHold

// function: checkExists

// pre: must pass in Card object

// post: returns an integer for the card number

public int checkExists(int hold) {

while (this.getExist(this.getCard(hold)))

hold = this.draw() ;

this.getCard(hold).setTrue() ;

return hold ;

} // end checkExists

// function: display

// pre: must pass in Card object

// post: prints the card with suit and face value

public void display(Card pick) {

System.out.println(this.getFaceVal(pick) + " of " + this.getSuit(pick));

//pick.exist = true ;

} // end display

// reset

// pre: none

// post: resets all the boolean values of the cards, similar to shuffling

public void reset() {

for (int i = 0, deckSize = 52; i < deckSize ; i++)

Deck[i].setFalse();

}

// images of the cards

static ImageIcon aceClubs = new ImageIcon("AceClubs.png") ;

static ImageIcon aceDiamonds = new ImageIcon("AceDiamonds.png") ;

static ImageIcon aceHearts = new ImageIcon("AceHearts.png") ;

static ImageIcon aceSpades = new ImageIcon("AceSpades.png") ;

static ImageIcon twoClubs = new ImageIcon("2Clubs.png") ;

static ImageIcon twoDiamonds = new ImageIcon("2Diamonds.png") ;

static ImageIcon twoHearts = new ImageIcon("2Hearts.png") ;

static ImageIcon twoSpades = new ImageIcon("2Spades.png") ;

static ImageIcon threeClubs = new ImageIcon("3Clubs.png") ;

static ImageIcon threeDiamonds = new ImageIcon("3Diamonds.png") ;

static ImageIcon threeHearts = new ImageIcon("3Hearts.png") ;

static ImageIcon threeSpades = new ImageIcon("3Spades.png") ;

static ImageIcon fourClubs = new ImageIcon("4Clubs.png") ;

static ImageIcon fourDiamonds = new ImageIcon("4Diamonds.png") ;

static ImageIcon fourHearts = new ImageIcon("4Hearts.png") ;

static ImageIcon fourSpades = new ImageIcon("4Spades.png") ;

static ImageIcon fiveClubs = new ImageIcon("5Clubs.png") ;

static ImageIcon fiveDiamonds = new ImageIcon("5Diamonds.png") ;

static ImageIcon fiveHearts = new ImageIcon("5Hearts.png") ;

static ImageIcon fiveSpades = new ImageIcon("5Spades.png") ;

static ImageIcon sixClubs = new ImageIcon("6Clubs.png") ;

static ImageIcon sixDiamonds = new ImageIcon("6Diamonds.png") ;

static ImageIcon sixHearts = new ImageIcon("6Hearts.png") ;

static ImageIcon sixSpades = new ImageIcon("6Spades.png") ;

static ImageIcon sevenClubs = new ImageIcon("7Clubs.png") ;

static ImageIcon sevenDiamonds = new ImageIcon("7Diamonds.png") ;

static ImageIcon sevenHearts = new ImageIcon("7Hearts.png") ;

static ImageIcon sevenSpades = new ImageIcon("7Spades.png") ;

static ImageIcon eightClubs = new ImageIcon("8Clubs.png") ;

static ImageIcon eightDiamonds = new ImageIcon("8Diamonds.png") ;

static ImageIcon eightHearts = new ImageIcon("8Hearts.png") ;

static ImageIcon eightSpades = new ImageIcon("8Spades.png") ;

static ImageIcon nineClubs = new ImageIcon("9Clubs.png") ;

static ImageIcon nineDiamonds = new ImageIcon("9Diamonds.png") ;

static ImageIcon nineHearts = new ImageIcon("9Hearts.png") ;

static ImageIcon nineSpades = new ImageIcon("9Spades.png") ;

static ImageIcon tenClubs = new ImageIcon("10Clubs.png") ;

static ImageIcon tenDiamonds = new ImageIcon("10Diamonds.png") ;

static ImageIcon tenHearts = new ImageIcon("10Hearts.png") ;

static ImageIcon tenSpades = new ImageIcon("10Spades.png") ;

static ImageIcon jackClubs = new ImageIcon("JackClubs.png") ;

static ImageIcon jackDiamonds = new ImageIcon("JackDiamonds.png") ;

static ImageIcon jackHearts = new ImageIcon("JackHearts.png") ;

static ImageIcon jackSpades = new ImageIcon("JackSpades.png") ;

static ImageIcon queenClubs = new ImageIcon("QueenClubs.png") ;

static ImageIcon queenDiamonds = new ImageIcon("QueenDiamonds.png") ;

static ImageIcon queenHearts = new ImageIcon("QueenHearts.png") ;

static ImageIcon queenSpades = new ImageIcon("QueenSpades.png") ;

static ImageIcon kingClubs = new ImageIcon("KingClubs.png") ;

static ImageIcon kingDiamonds = new ImageIcon("KingDiamonds.png") ;

static ImageIcon kingHearts = new ImageIcon("KingHearts.png") ;

static ImageIcon kingSpades = new ImageIcon("KingSpades.png") ;

// handCard

// pre: must pass in an integer (0 - 51) that represents one of the cards form the deck

// post: returns label that contains image

public static JLabel handCard(int hold) {

JLabel label ;

switch (hold)

{

case 0: return label = new JLabel(aceClubs);

case 1: return label = new JLabel(twoClubs);

case 2: return label = new JLabel(threeClubs);

case 3: return label = new JLabel(fourClubs);

case 4: return label = new JLabel(fiveClubs);

case 5: return label = new JLabel(sixClubs);

case 6: return label = new JLabel(sevenClubs);

case 7: return label = new JLabel(eightClubs);

case 8: return label = new JLabel(nineClubs);

case 9: return label = new JLabel(tenClubs);

case 10: return label = new JLabel(jackClubs);

case 11: return label = new JLabel(queenClubs);

case 12: return label = new JLabel(kingClubs);

case 13: return label = new JLabel(aceHearts);

case 14: return label = new JLabel(twoHearts);

case 15: return label = new JLabel(threeHearts);

case 16: return label = new JLabel(fourHearts);

case 17: return label = new JLabel(fiveHearts);

case 18: return label = new JLabel(sixHearts);

case 19: return label = new JLabel(sevenHearts);

case 20: return label = new JLabel(eightHearts);

case 21: return label = new JLabel(nineHearts);

case 22: return label = new JLabel(tenHearts);

case 23: return label = new JLabel(jackHearts);

case 24: return label = new JLabel(queenHearts);

case 25: return label = new JLabel(kingHearts);

case 26: return label = new JLabel(aceSpades);

case 27: return label = new JLabel(twoSpades);

case 28: return label = new JLabel(threeSpades);

case 29: return label = new JLabel(fourSpades);

case 30: return label = new JLabel(fiveSpades);

case 31: return label = new JLabel(sixSpades);

case 32: return label = new JLabel(sevenSpades);

case 33: return label = new JLabel(eightSpades);

case 34: return label = new JLabel(nineSpades);

case 35: return label = new JLabel(tenSpades);

case 36: return label = new JLabel(jackSpades);

case 37: return label = new JLabel(queenSpades);

case 38: return label = new JLabel(kingSpades);

case 39: return label = new JLabel(aceDiamonds);

case 40: return label = new JLabel(twoDiamonds);

case 41: return label = new JLabel(threeDiamonds);

case 42: return label = new JLabel(fourDiamonds);

case 43: return label = new JLabel(fiveDiamonds);

case 44: return label = new JLabel(sixDiamonds);

case 45: return label = new JLabel(sevenDiamonds);

case 46: return label = new JLabel(eightDiamonds);

case 47: return label = new JLabel(nineDiamonds);

case 48: return label = new JLabel(tenDiamonds);

case 49: return label = new JLabel(jackDiamonds);

case 50: return label = new JLabel(queenDiamonds);

case 51: return label = new JLabel(kingDiamonds);

} // end switch

return label = new JLabel("Error - could not determine card.");

} // end handCard

} // end Deck class

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Chris Dang CSCI Java 1125

// Hand Class

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.lang.\* ;

import javax.swing.\* ;

public class Hand extends Deck{

// Payout values for winning hands

final int NO\_PAIR = 0 ;

final int PAIR = 1 ;

final int TWO\_PAIR = 2 ;

final int THREE\_KIND = 3 ;

final int STRAIGHT = 4 ;

final int FLUSH = 5 ;

final int FULL\_HOUSE = 6 ;

final int FOUR\_KIND = 25 ;

final int STRAIGHT\_FLUSH = 50 ;

final int ROYAL\_FLUSH = 250 ;

final int HAND = 5 ;

private Card[] Hand ;

// constructs a sample hand to start

// hand consists of a card array of size 5

public Hand(){

Hand = new Card[HAND] ;

for (int i = 0 ; i < HAND; i++) {

Hand[i] = new Card(1, 1);

} // end for

} // end hand constructor

public void display() {

for (int i = 0; i < HAND; i++)

System.out.println(this.getFaceVal(Hand[i]) + " of " + this.getSuit(Hand[i]));

}

// sorts hand

public Hand sortHand() {

Card temp ;

for (int i = 0; i < HAND - 1; i++){

for (int j = 1; j < HAND - i; j++) {

if (Hand[j-1].faceVal > Hand[j].faceVal){

temp = Hand[j-1] ;

Hand[j-1] = Hand[j] ;

Hand[j] = temp ;

}// end if

}// end for

}// end for

return this ;

} // end sortHand

// valueHand

// pre: none

// post: will return an integer value for the payout of the hand based on a chart given

public int valueHand() {

int value = 0 ;

// checks for flush

if(Hand[0].suit == Hand[1].suit &&

Hand[1].suit == Hand[2].suit &&

Hand[2].suit == Hand[3].suit &&

Hand[3].suit == Hand[4].suit) {

// check for straight flush

int count = 0 ;

for (int i = 0; i < HAND ; i++) {

int j = ( i + 1 ) % 5;

if (Hand[i].faceVal % 13 == (Hand[j].faceVal - 1))

count++ ;

}// end for

if (count == 4){ // if count == 4, then check for royal flush

//System.out.println("Count is: " + count) ;

if(Hand[0].faceVal == 1 &&

Hand[1].faceVal == 10 &&

Hand[2].faceVal == 11 &&

Hand[3].faceVal == 12 &&

Hand[4].faceVal == 13)

return value = ROYAL\_FLUSH ;

return value = STRAIGHT\_FLUSH ; // else return straight flush

}

return value = FLUSH ; // else return straight flush

} // end if

else {

// checks for straights

int count = 0 ;

for (int i = 0; i < HAND ; i++) {

int j = ( i + 1 ) % 5;

if (Hand[i].faceVal % 13 == (Hand[j].faceVal - 1))

count++ ;

} // end for

if (count == 4)

return value = STRAIGHT ;

// check for 4 of a kind

if (Hand[0].faceVal == Hand[1].faceVal &&

Hand[1].faceVal == Hand[2].faceVal &&

Hand[2].faceVal == Hand[3].faceVal

&&

Hand[1].faceVal == Hand[2].faceVal &&

Hand[2].faceVal == Hand[2].faceVal &&

Hand[3].faceVal == Hand[4].faceVal)

return FOUR\_KIND ;

// check for 3 the 3 combinations of 3 of a kind

for (int i = 0; i < 3; i++) { // if 3 of a kind is the first 3 cards, middle 3, and last 3

int j = i ; // used to hold start of the three of a kind

if (Hand[i].faceVal == Hand[i+1].faceVal &&

Hand[i+1].faceVal == Hand[i+2].faceVal){

// check for full house

if (j == 2 ) // checks if pair is lower faceVal than 3-of kind

if (Hand[0].faceVal == Hand[1].faceVal)

return FULL\_HOUSE ;

if (j == 0) // checks if pair is higher faceVal than 3-of kind

if (Hand[3].faceVal == Hand[4].faceVal)

// check for pair to match with 3 of kind for full house

return FULL\_HOUSE ;

return THREE\_KIND ;

}// end for

}// end if

// checks for 4 combinations of a pair

int pairs = 0 ;

for (int k = 0; k < 4; k++){

if (Hand[k].faceVal == Hand[k+1].faceVal)

pairs++ ; // keeps track of number of pairs in hand

} // end check-pair for

if (pairs == 1)

return PAIR ;

if (pairs == 2)

return TWO\_PAIR ;

}// end else

// if all checks fail, pay out is 0.

return NO\_PAIR;

} // end valueHand

// randomHand

// pre: none

// post: will return an hand of random cards

public Hand randomHand() {

for (int i = 0; i < HAND; i++){

int hold = this.draw() ; // card number is initially drawn

hold = this.checkExists(hold) ;

Hand[i] = this.getCard(hold) ;

}// end for

return this ;

}// end randomHand

// swapCard

// pre: must pass in an integer value for the index in the hand (0 - 4)

// post: Hand[i] will contain a random card.

public void swapCard(int i) {

int hold = this.draw() ; // draw random nunmber

hold = this.checkExists(hold) ; // check to sure number is unique

Hand[i] = this.getCard(hold) ; // sets new random card in hand

} // end swapCard

// showCard

// pre: must pass in an integer value for the index in the hand (0 - 4)

// post: Will return the label containing the image of card

public JLabel showCard(int i) {

int hold = this.getHold(Hand[i]) ; // get index number of card (0 - 51)

return handCard(hold) ; // return label based on index number

} // end showCard

} // end Hand class

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// Chris Dang CSCI Java 1125

// VideoPoker.Java

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

import java.awt.\* ;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.\* ;

public class VideoPoker {

// initial values

private int javaDollar = 10 ;

private int javaEarned = 0 ;

private boolean replaceBool1 = false ;

private boolean replaceBool2 = false ;

private boolean replaceBool3 = false ;

private boolean replaceBool4 = false ;

private boolean replaceBool5 = false ;

// creates deck for game

Deck pokerDeck = new Deck() ;

// creates hand for game

Hand pokerHand = new Hand() ;

// Dimensions for Poker game display

final int WIDTH = 450 ;

final int HEIGHT = 410 ;

final int INFO\_WIDTH = 100 ;

final int INFO\_HEIGHT = 220 ;

final int PAY\_WIDTH = 125 ;

final int PAY\_HEIGHT = 30 ;

final int CARD\_WIDTH = 400 ;

final int CARD\_HEIGHT = 110 ;

// Payout values for winning hands

final int PAIR = 1 ;

final int TWO\_PAIR = 2 ;

final int THREE\_KIND = 3 ;

final int STRAIGHT = 4 ;

final int FLUSH = 5 ;

final int FULL\_HOUSE = 6 ;

final int FOUR\_KIND = 25 ;

final int STRAIGHT\_FLUSH = 50 ;

final int ROYAL\_FLUSH = 250 ;

// Frame of video poker

JFrame frame = new JFrame("Video Poker") ;

// describes payout amounts

JLabel payoutInfo = new JLabel("Payout Info:" ) ;

JLabel pair = new JLabel("Pair: "+ PAIR) ;

JLabel twoPair = new JLabel("Two Pair: " + TWO\_PAIR) ;

JLabel threeKind = new JLabel("Three of a Kind: " + THREE\_KIND) ;

JLabel straight = new JLabel("Straight: " + STRAIGHT) ;

JLabel flush = new JLabel("Flush: " + FLUSH) ;

JLabel fullHouse = new JLabel("Full House: " + FULL\_HOUSE) ;

JLabel fourKind = new JLabel("Four of a Kind: " + FOUR\_KIND) ;

JLabel straightFlush = new JLabel("Straight Flush: " + STRAIGHT\_FLUSH) ;

JLabel royalFlush = new JLabel("Royal Flush: " + ROYAL\_FLUSH) ;

// Declares check boxes for replace

JCheckBox replace1 = new JCheckBox("Replace") ;

JCheckBox replace2 = new JCheckBox("Replace") ;

JCheckBox replace3 = new JCheckBox("Replace") ;

JCheckBox replace4 = new JCheckBox("Replace") ;

JCheckBox replace5 = new JCheckBox("Replace") ;

// Declares buttons to play game

JButton stand = new JButton("Stand") ;

JButton draw = new JButton("Draw") ;

JButton deal = new JButton("Deal") ;

JButton payoutButton = new JButton("Payout") ;

JPanel card1 = new JPanel() ;

JPanel card2 = new JPanel() ;

JPanel card3 = new JPanel() ;

JPanel card4 = new JPanel() ;

JPanel card5 = new JPanel() ;

JPanel javaDollarPan = new JPanel() ;

JPanel javaDollarEarned = new JPanel() ;

JLabel javaDollarDisplay = new JLabel(javaDollar + " Java Dollars") ;

JLabel javaDollarEarnedLbl = new JLabel("You have earned: " + javaEarned + " Java Dollars.") ;

VideoPoker() {

// closes game

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

// sets background

JPanel back = new JPanel() ;

back.setBackground(Color.green) ;

back.setPreferredSize(new Dimension(WIDTH, HEIGHT));

// set payout info box

JPanel payout = new JPanel() ;

payout.setBackground(Color.red) ;

payout.setPreferredSize(new Dimension(INFO\_WIDTH, INFO\_HEIGHT));

// adds panels for payout info, javaDollars, and earned javaDollars

back.add(payout) ;

back.add(javaDollarPan) ;

back.add(javaDollarEarned) ;

frame.getContentPane().add(back) ;

pokerHand.randomHand() ;

// initializes first game

card1.add(pokerHand.showCard(0)) ;

card2.add(pokerHand.showCard(1)) ;

card3.add(pokerHand.showCard(2)) ;

card4.add(pokerHand.showCard(3)) ;

card5.add(pokerHand.showCard(4)) ;

// adds info to panel

payout.add(payoutInfo) ;

payout.add(pair) ;

payout.add(twoPair) ;

payout.add(threeKind) ;

payout.add(straight) ;

payout.add(flush) ;

payout.add(fullHouse) ;

payout.add(fourKind) ;

payout.add(straightFlush) ;

payout.add(royalFlush) ;

// creates panel to show current payout of java dollars

javaDollarPan.setBackground(Color.orange) ;

javaDollarPan.setPreferredSize(new Dimension(PAY\_WIDTH, PAY\_HEIGHT));

javaDollarPan.add(javaDollarDisplay) ;

// creates a panel to show how much player has earned from hand

javaDollarPan.setBackground(Color.orange) ;

javaDollarPan.setPreferredSize(new Dimension(PAY\_WIDTH, PAY\_HEIGHT));

javaDollarEarned.add(javaDollarEarnedLbl) ;

// adds listners to buttons

stand.addActionListener(new standListen()) ;

draw.addActionListener(new drawListen());

deal.addActionListener(new dealListen()) ;

// adds listener for payout button

payoutButton.addActionListener(new payOutListen()) ;

// disables stand, draw, payout buttons until after game has been played

payoutButton.setEnabled(false);

stand.setEnabled(false);

draw.setEnabled(false);

// adds cards to frame

back.add(card1) ;

back.add(card2) ;

back.add(card3) ;

back.add(card4) ;

back.add(card5) ;

// adds buttons to frame

back.add(replace1) ;

back.add(replace2) ;

back.add(replace3) ;

back.add(replace4) ;

back.add(replace5) ;

back.add(stand) ;

back.add(draw) ;

back.add(deal) ;

back.add(payoutButton) ;

frame.pack() ;

frame.setVisible(true) ;

}// end public static void main

// payOutListen

// adds java dollars earned to current amount of java dollars used for play

// earned dollars are reset to 0, and text is updated.

private class payOutListen implements ActionListener

{

public void actionPerformed(ActionEvent event)

{

javaDollar += javaEarned ; // add earned to funds

javaEarned = 0 ; // reset earned

javaDollarDisplay.setText(javaDollar + " Java Dollars"); // update text

javaDollarEarnedLbl.setText("You have earned: " + javaEarned + " Java Dollars.");

// re-enables deal button after adding more funds to java Dollars

if (javaDollar > 0)

deal.setEnabled(true) ;

}

} // end payOutListen

// dealListen

// deals 5 new random cards

private class dealListen implements ActionListener

{

public void actionPerformed(ActionEvent event){

// if player has credits to play, player bets 1 java dollar to play

if (javaDollar > 0) {

javaDollar-- ;

// sets up new game

card1.removeAll(); // card panels are removed

card2.removeAll();

card3.removeAll();

card4.removeAll();

card5.removeAll();

pokerHand.reset(); // deck is reset

pokerHand.randomHand() ;// hand is randomized with new cards

card1.add(pokerHand.showCard(0)) ; // new card panels are added

card2.add(pokerHand.showCard(1)) ;

card3.add(pokerHand.showCard(2)) ;

card4.add(pokerHand.showCard(3)) ;

card5.add(pokerHand.showCard(4)) ;

// update javaDollars

javaDollarDisplay.setText(javaDollar + " Java Dollars");

frame.pack() ;

}

else { // disables deal button after player has run out of funds

javaDollarDisplay.setText("Thank you for playing."); // displays thank you text

deal.setEnabled(false) ; // buttons are disabled

}

stand.setEnabled(true); // renables stand and draw buttons

draw.setEnabled(true) ;

// resets (unchecks) checkboxes for replace checkboxes

replaceBool1 = false ;

replaceBool2 = false ;

replaceBool3 = false ;

replaceBool4 = false ;

replaceBool5 = false ;

replace1.setSelected(false);

replace2.setSelected(false);

replace3.setSelected(false);

replace4.setSelected(false);

replace5.setSelected(false);

}

} // end deal listen

// standListen

// calculates hand.

private class standListen implements ActionListener

{

public void actionPerformed(ActionEvent event){

// enables payout button

payoutButton.setEnabled(true);

pokerHand.sortHand() ; // sorts internal hand array

javaEarned = pokerHand.valueHand() ; // assigns hand amount to earned funds

javaDollarEarnedLbl.setText("You have earned: " + javaEarned + " Java Dollars."); // updates text

stand.setEnabled(false); // disables stand button

draw.setEnabled(false) ; // disables draw button

// resets checkboxes for replace checkboxes

replaceBool1 = false ;

replaceBool2 = false ;

replaceBool3 = false ;

replaceBool4 = false ;

replaceBool5 = false ;

replace1.setSelected(false);

replace2.setSelected(false);

replace3.setSelected(false);

replace4.setSelected(false);

replace5.setSelected(false);

}

} //end standListen

// draw listen

// swaps out any cards that have been selected and calculates hand.

private class drawListen implements ActionListener

{

public void actionPerformed(ActionEvent event) {

payoutButton.setEnabled(true); // enables payout button

if (replace1.isSelected()) // sets swap booleans to true is check boxes are selected

replaceBool1 = true ;

if (replace2.isSelected())

replaceBool2 = true ;

if (replace3.isSelected())

replaceBool3 = true ;

if (replace4.isSelected())

replaceBool4 = true ;

if (replace5.isSelected())

replaceBool5 = true ;

// swaps card based on check boxes

if (replaceBool1) {

pokerHand.swapCard(0) ; // swaps card 1

card1.removeAll();

card1.add(pokerHand.showCard(0)) ;

} // replaceBool1

if (replaceBool2) {

pokerHand.swapCard(1) ; // swaps card 2

card2.removeAll();

card2.add(pokerHand.showCard(1)) ;

} // replaceBool2

if (replaceBool3) {

pokerHand.swapCard(2) ; // swaps card 3

card3.removeAll();

card3.add(pokerHand.showCard(2)) ;

} // replaceBool3

if (replaceBool4) {

pokerHand.swapCard(3) ; // swaps card 4

card4.removeAll();

card4.add(pokerHand.showCard(3)) ;

} // replaceBool4

if (replaceBool5) {

pokerHand.swapCard(4) ; // swaps card 5

card5.removeAll();

card5.add(pokerHand.showCard(4)) ;

} // replaceBool5

pokerHand.sortHand() ; // sorts hand

javaEarned = pokerHand.valueHand() ; // assigns hand value to earned funds

javaDollarEarnedLbl.setText("You have earned: " + javaEarned + " Java Dollars."); // updates text

stand.setEnabled(false); // disables stand and draw buttons

draw.setEnabled(false) ;

frame.pack() ;

// resets checkboxes for replace checkboxes

replaceBool1 = false ;

replaceBool2 = false ;

replaceBool3 = false ;

replaceBool4 = false ;

replaceBool5 = false ;

replace1.setSelected(false);

replace2.setSelected(false);

replace3.setSelected(false);

replace4.setSelected(false);

replace5.setSelected(false);

} // end actionPerformed

}// end swap1

} // end class VideoPoker

// Chris Dang

// PlayPoker.java

// driver for video poker

// creates a video poker game

public class PlayPoker {

public static void main(String args[]) {

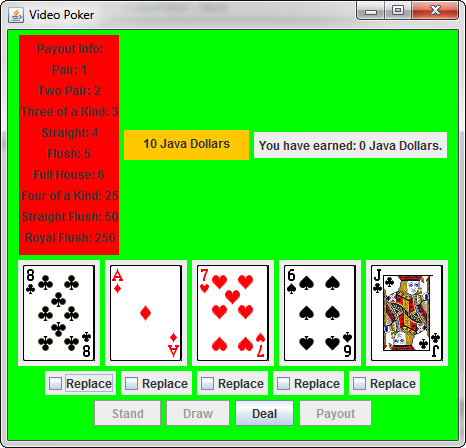
VideoPoker play = new VideoPoker() ;

}

}

**On startup**





**After clicking button Deal**



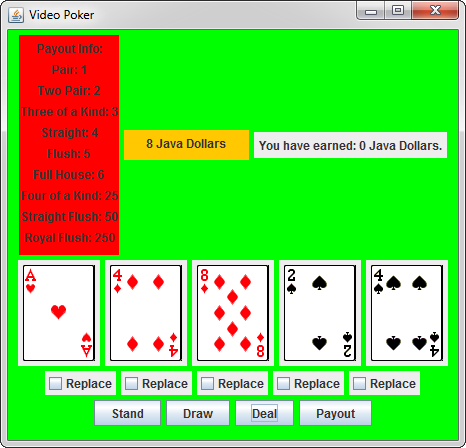
**Selecting cards to be replaced**



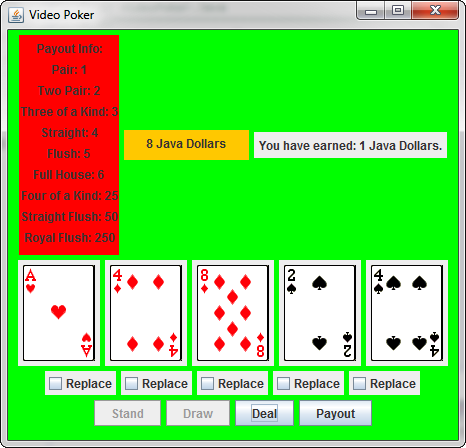
**After pressing draw to replace cards**



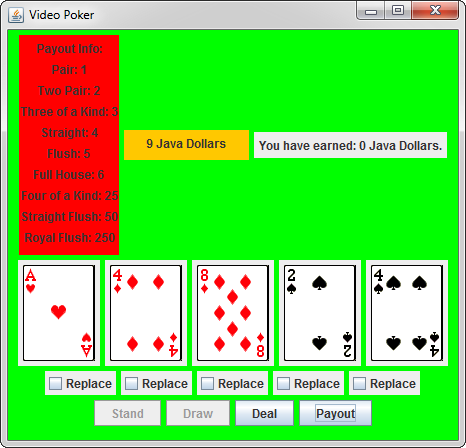
**After pressing deal again to start new hand**



**Click button stand. Pair is evaluated.**



**Payout button has been clicked, returning java dollars earned to javal dollars funds used to play**



**After hitting deal until funds are no longer available.**

