For: Christian Oliver

Assignment: Black Jack Final

GitHub URL:

Student: Please answer the questions, then use the Insert, Screenshot option in Word to snip an appropriate sample of your executing program’s output.

Copy the code from your .java file(s) into the code section below. Your code should match the code submitted in GitHub.

Be sure to review your graded assignment for instructor comments!

|  |
| --- |
| **Analysis** |
| *Describe the problem, including input and output, in your own words* |
| The problem was to establish the card and the way the game is played using score and card analysis. |

|  |
| --- |
| **Design** |
| *Describe the major steps for problem solving* |
| I first set up the ui and cards. Then the score, dealer, and stage. Then give the details of the ui. |

|  |
| --- |
| **Testing** |
| *Describe your test plan* |
| I first decided to give the cards their own detail and decided that they needed to correlate with the ranks and suits of the normal cards. The decided score and hand outcome based on score. |

|  |
| --- |
| **Screenshot(s)** |
| *Paste screen shot(s) here, within this table entry* |
|  |

|  |
| --- |
| **Code** |
| *Paste code here, within this table entry. Use the retain formatting option of the Paste* |
| //Christian Oliver  package card;  import javafx.application.Application;  import javafx.beans.property.SimpleBooleanProperty;  import javafx.beans.property.SimpleStringProperty;  import javafx.geometry.Insets;  import javafx.geometry.Pos;  import javafx.scene.Parent;  import javafx.scene.Scene;  import javafx.scene.control.Button;  import javafx.scene.control.TextField;  import javafx.scene.layout.HBox;  import javafx.scene.layout.Pane;  import javafx.scene.layout.Region;  import javafx.scene.layout.StackPane;  import javafx.scene.layout.VBox;  import javafx.scene.paint.Color;  import javafx.scene.shape.Rectangle;  import javafx.scene.text.Text;  import javafx.stage.Stage;  public class blackjack extends Application{    private Deck cards = new Deck();  private Text message = new Text();  private Dealer Rusty, PPlay;    private SimpleBooleanProperty playable = new SimpleBooleanProperty(false);          private HBox aICards = new HBox(20);    private HBox p1Cards = new HBox(20);  private Parent createContent() {  Rusty = new Dealer(aICards.getChildren());  PPlay= new Dealer(p1Cards.getChildren());  Pane root = new Pane();  root.setPrefSize(500, 300);  Region background = new Region();  background.setPrefSize(1000, 800);  background.setStyle("-fx-background-color: WHITE");  HBox rootLayout = new HBox(5);  rootLayout.setPadding(new Insets(5, 5, 5, 5));  Rectangle leftBG = new Rectangle(550, 560);  leftBG.setArcWidth(50);  leftBG.setArcHeight(50);  leftBG.setFill(Color.MAGENTA);  Rectangle rightBG = new Rectangle(230, 560);  rightBG.setArcWidth(50);  rightBG.setArcHeight(50);  rightBG.setFill(Color.YELLOW);    VBox leftVBox = new VBox(50);  leftVBox.setAlignment(Pos.TOP\_CENTER);  Text RustyScore = new Text("Rusty: ");  Text playerScore = new Text("Player: ");  leftVBox.getChildren().addAll(RustyScore, aICards, message, p1Cards, playerScore);    VBox rightVBox = new VBox(20);  rightVBox.setAlignment(Pos.CENTER);  final TextField bet = new TextField("BET");  bet.setDisable(true);  bet.setMaxWidth(50);  Text credits = new Text("Credits");  Button btnPlay = new Button("PLAY");  Button btnHit = new Button("HIT");  Button btnStand = new Button("STAND");  HBox buttonsHBox = new HBox(15, btnHit, btnStand);  buttonsHBox.setAlignment(Pos.CENTER);  rightVBox.getChildren().addAll(bet, btnPlay, credits, buttonsHBox);    rootLayout.getChildren().addAll(new StackPane(leftBG, leftVBox), new StackPane(rightBG, rightVBox));  root.getChildren().addAll(background, rootLayout);    btnPlay.disableProperty().bind(playable);  btnHit.disableProperty().bind(playable.not());  btnStand.disableProperty().bind(playable.not());  playerScore.textProperty().bind(new SimpleStringProperty("Player: ").concat(PPlay.valueProperty().asString()));  RustyScore.textProperty().bind(new SimpleStringProperty("Rusty: ").concat(Rusty.valueProperty().asString()));  PPlay.valueProperty().addListener((obs, old, newValue) -> {  if (newValue.intValue() >= 21) {  endGame();  }  });  Rusty.valueProperty().addListener((obs, old, newValue) -> {  if (newValue.intValue() >= 21) {  endGame();  }  });  // Buttons to play the game  btnPlay.setOnAction(event -> {  startNewGame();  });  btnHit.setOnAction(event -> {  PPlay.takeCard(cards.drawCard());  });  btnStand.setOnAction(event -> {  while (Rusty.valueProperty().get() < 17) {  Rusty.takeCard(cards.drawCard());  }  endGame();  });  return root;  }  private void startNewGame() {  playable.set(true);  message.setText("");  cards.refill();  Rusty.reset();  PPlay.reset();  Rusty.takeCard(cards.drawCard());  Rusty.takeCard(cards.drawCard());  PPlay.takeCard(cards.drawCard());  PPlay.takeCard(cards.drawCard());  }  private void endGame() {  playable.set(false);  int RustyValue = Rusty.valueProperty().get();  int PPlayValue = PPlay.valueProperty().get();  String winner = "Exceptional case: d: " + RustyValue + " p: " + PPlayValue;  // Checks who one the game  if (RustyValue == 21 || PPlayValue > 21 ||(RustyValue < 21 && RustyValue > PPlayValue)) {  winner = "Rusty";  }  else if (PPlayValue == 21 || RustyValue > 21 || PPlayValue > RustyValue) {  winner = "You";  }  else {  winner = "NO ONE ";  }  message.setText(winner + " WON");  }  @Override  public void start(Stage primaryStage) throws Exception {  primaryStage.setScene(new Scene(createContent()));  primaryStage.setWidth(800);  primaryStage.setHeight(600);  primaryStage.setResizable(false);  primaryStage.setTitle("Black-Jack");  primaryStage.show();  }  public static void main(String[] args) {  launch(args);  }    }//Christian Oliver  package card;  import javafx.scene.Parent;  import javafx.scene.image.Image;  import javafx.scene.layout.StackPane;  import javafx.scene.paint.Color;  import javafx.scene.shape.Rectangle;  import javafx.scene.text.Text;  public class Card extends Parent {      //card suits  enum Suit {  HEARTS, DIAMONDS, SPADES, CLUBS;    };  //card points  enum Rank {  TWO(2), THREE(3), FOUR(4), FIVE(5), SIX(6), SEVEN(7), EIGHT(8),  NINE(9), TEN(10), JACK(10), QUEEN(10), KING(10), ACE(11);    final int value;  private Rank(int value){  this.value = value;  }  };    public final Suit suit;  public final Rank rank;  public final int value;  public Card(Suit suit, Rank rank) {  this.suit = suit;  this.rank = rank;  this.value = rank.value;    Rectangle jack = new Rectangle(90, 150);  // Creates card Background  jack.setArcWidth(20);  jack.setArcHeight(20);  jack.setFill(Color.YELLOW);    //Sets the wrapping of text to smaller than the card it's self  Text t = new Text(toString());  t.setWrappingWidth(80);    //adds stack pane to allow for text to go on top of card  getChildren().add(new StackPane(jack, t));          }  public String toString() {  return rank.toString() + " of " + suit.toString();  }        }//Christian Oliver  package card;  import javafx.beans.property.SimpleIntegerProperty;  import javafx.collections.ObservableList;  import javafx.scene.Node;  import card.Card.Rank;  public class Dealer {  private ObservableList<Node> card;  private SimpleIntegerProperty value = new SimpleIntegerProperty(0);  private int aces = 0;    public Dealer(ObservableList<Node> card) {  this.card = card;  }  public void takeCard(Card card1) {  card.add(card1);  if (card1.rank == Rank.ACE) {  aces++;  }  if (value.get() + card1.value > 21 && aces > 0) {  value.set(value.get() + card1.value - 10); //then count ace as '1' not '11'  aces--;  }  else {  value.set(value.get() + card1.value);  }  }  public void reset() {  card.clear();  value.set(0);  aces = 0;  }  public SimpleIntegerProperty valueProperty() {  return value;  }    }//Christian Oliver  package card;  //Pulls Rank and Suit from Card.java  import card.Card.Rank;  import card.Card.Suit;  public class Deck {  // determines 52 card deck  private Card[] card = new Card[52];    public Deck() {  refill();    }  // Returns all rank values fron Rank and Suit in Card.Java  public final void refill() {  int i= 0;  for (Suit suit : Suit.values()){  for (Rank rank : Rank.values()) {  card[i++] = new Card(suit, rank);  }  }  }  // Calls random card from the deck at random and does not allow to pull same card twice  public Card drawCard() {  Card card1 = null;  while (card1 == null) {  int index = (int)(Math.random()\*card.length);  card1 = card[index];  card[index] = null;  }  return card1;  }  } |