For: Evan Musick

Assignment: Final Project: BlackJack

GitHub URL: <https://github.com/musickevan1/CardGame21>

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 is to create a simplified version of a card game, where the user and the house draw cards, and the goal is to get a score as close to 21 as possible without going over. The user can draw additional cards ("hit") or end the game. The house must draw cards until its score is 17 or more. The program's output includes the player's and house's scores, hands won, and hands lost. |

|  |
| --- |
| **Design** |
| *Describe the major steps for problem solving* |
| 1. Create the graphical user interface (GUI) with labels, buttons, and panels for displaying the game state.  2. Initialize a deck of cards and shuffle it.  3. Implement the game logic, such as drawing cards, calculating scores, and determining the winner.  4. Update the GUI with the current game state, including the cards drawn, scores, and the number of hands won and lost. |

|  |
| --- |
| **Testing** |
| *Describe your test plan* |
| 1. Start a new game and verify the initial hands of both the player and the house.  2. Test the "Hit Me" button to draw additional cards and check if the player's score updates correctly.  3. Test the Ace's scoring logic (1/11) to see if the card updates score correctly.  4. Test the "End Game" button to end the game, update the hands won or lost, and display the winner or a draw.  5. Test multiple games to ensure the deck is reshuffled and scores are reset properly. |

|  |
| --- |
| **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* |
| package com.mycompany.cardgame21;  import java.awt.\*;  import java.awt.image.BufferedImage;  import java.io.IOException;  import java.io.InputStream;  import java.util.ArrayList;  import java.util.Collections;  import javax.imageio.ImageIO;  import javax.swing.\*;  /\*  \* Title: Card Game 21  \* Author: Evan Musick  \* Date: 4/26/2023  \* Instructor: Kirsten Markley  \* Course: 23/SP-CIS-171-W01  \*/  // CardGame21 is a simple card game that implements the core mechanics of a game of Blackjack (21).  public class CardGame21 extends JFrame {  // Initialize instance variables for the game state and UI components.  private ArrayList<String>deck;  private ArrayList<String>playerCards;  private ArrayList<String>houseCards;  private int playerScore, houseScore;  private int handsWon, handsLost;  private JPanel playerPanel, housePanel;  private JLabel lblHandsWon, lblHandsLost, lblPlayerScore, lblHouseScore;  public static void main(String[] args) {  // Launch the game on the Event Dispatch Thread (EDT).  EventQueue.invokeLater(() ->{  try {  CardGame21 frame = new CardGame21();  frame.setVisible(true);  } catch (Exception e) {  e.printStackTrace();  }  });  }  // Constructor for the CardGame21 class.  public CardGame21() {  setTitle("Card Game 21");  setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  setSize(800, 600);  setLayout(new BorderLayout());  initialize();  }    // Method to load a custom font from a resource file.  private Font loadCustomFont(String fontPath, float fontSize) {  try {  InputStream fontInputStream = getClass().getResourceAsStream(fontPath);  Font customFont = Font.createFont(Font.TRUETYPE\_FONT, fontInputStream);  return customFont.deriveFont(fontSize);  } catch (IOException | FontFormatException e) {  System.err.println("Error loading custom font: " + e.getMessage());  return new Font("Arial", Font.PLAIN, (int)fontSize);  }  }  // Initialize the game UI and state.  private void initialize() {  // Set up the main panel with a border layout and a border.  JPanel mainPanel = new JPanel(new BorderLayout());  mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));  mainPanel.setOpaque(false);  getContentPane().add(mainPanel, BorderLayout.CENTER);    // Initialize player and house card arrays.  playerCards = new ArrayList<>();  houseCards = new ArrayList<>();  // Set up the top panel with labels for hands won, hands lost, player score, and house score.  JPanel topPanel = new JPanel(new GridLayout(1, 2));  topPanel.setOpaque(false);  lblHandsWon = new JLabel("Hands Won: " + handsWon);  lblHandsLost = new JLabel("Hands Lost: " + handsLost);  lblPlayerScore = new JLabel("Player Score: 0");  lblHouseScore = new JLabel("House Score: 0");  topPanel.add(lblHandsWon);  topPanel.add(lblPlayerScore);  topPanel.add(lblHouseScore);  topPanel.add(lblHandsLost);  mainPanel.add(topPanel, BorderLayout.NORTH);  // Set up the center panel with player and house card display areas.  JPanel centerPanel = new JPanel(new GridLayout(1, 2, 20, 0));  centerPanel.setOpaque(false);  playerPanel = new JPanel();  playerPanel.setBorder(BorderFactory.createTitledBorder("Player"));  playerPanel.setOpaque(false);  housePanel = new JPanel();  housePanel.setBorder(BorderFactory.createTitledBorder("House"));  housePanel.setOpaque(false);  centerPanel.add(playerPanel);  centerPanel.add(housePanel);  mainPanel.add(centerPanel, BorderLayout.CENTER);  // Set up the bottom panel with buttons for starting a new game, hitting, and ending the game.  JPanel bottomPanel = new JPanel();  bottomPanel.setOpaque(false);  JButton btnNewGame = new JButton("New Game");  btnNewGame.addActionListener(e ->newGame());  JButton btnHitMe = new JButton("Hit Me");  btnHitMe.addActionListener(e ->hitPlayer());  JButton btnEndGame = new JButton("End Game");  btnEndGame.addActionListener(e ->endGame());  bottomPanel.add(btnNewGame);  bottomPanel.add(btnHitMe);  bottomPanel.add(btnEndGame);  mainPanel.add(bottomPanel, BorderLayout.SOUTH);  // Load custom font and apply it to game text and buttons.  Font customFont = loadCustomFont("/Roboto-Regular.ttf", 18);  btnNewGame.setFont(customFont);  btnHitMe.setFont(customFont);  btnEndGame.setFont(customFont);  lblHandsWon.setFont(customFont);  lblPlayerScore.setFont(customFont);  lblHouseScore.setFont(customFont);  lblHandsLost.setFont(customFont);  // Set foreground color for labels and buttons.  Color foregroundColor = new Color(255, 255, 255);  btnNewGame.setForeground(foregroundColor);  btnHitMe.setForeground(foregroundColor);  btnEndGame.setForeground(foregroundColor);  lblHandsWon.setForeground(foregroundColor);  lblPlayerScore.setForeground(foregroundColor);  lblHouseScore.setForeground(foregroundColor);  lblHandsLost.setForeground(foregroundColor);  // Set background color for buttons.  Color buttonColor = new Color(70, 130, 180);  btnNewGame.setBackground(buttonColor);  btnHitMe.setBackground(buttonColor);  btnEndGame.setBackground(buttonColor);  // Set background color for the main frame.  getContentPane().setBackground(new Color(10, 45, 80));  newGame();  }  // Start a new game by creating and shuffling a deck, and dealing two cards to the player and house.  private void newGame() {  deck = createDeck();  Collections.shuffle(deck);  playerPanel.removeAll();  housePanel.removeAll();  playerCards.clear();  houseCards.clear();  // Deal two cards to the player.  for (int i = 0; i <2; i++) {  String card = deck.remove(0);  playerPanel.add(new JLabel(loadImage(card)));  playerCards.add(card);  }  // Deal two cards to the house.  for (int i = 0; i <2; i++) {  String card = deck.remove(0);  housePanel.add(new JLabel(loadImage(card)));  houseCards.add(card);  }  playerScore = calculateScore(playerCards);  houseScore = calculateScore(houseCards);  revalidate();  repaint();  updateScores(lblPlayerScore, lblHouseScore);  }  // Update the player and house scores in the UI.  private void updateScores(JLabel lblPlayerScore, JLabel lblHouseScore) {  lblPlayerScore.setText("Player Score: " + playerScore);  lblHouseScore.setText("House Score: " + houseScore);  }  // Give the player another card and update the game state.  private void hitPlayer() {  String card = deck.remove(0);  playerPanel.add(new JLabel(loadImage(card)));  playerCards.add(card);  playerScore = calculateScore(playerCards);  lblPlayerScore.setText("Player Score: " + playerScore);  revalidate();  repaint();  // End the game if the player busts (score over 21).  if (playerScore >21) {  endGame(); // End the game if the player busts  }  }  // End the game, calculate the final scores, and update the win/loss record.  private void endGame() {  // The house draws cards until its score is 17 or higher.  while (houseScore <17) {  String card = deck.remove(0);  housePanel.add(new JLabel(loadImage(card)));  houseCards.add(card);  houseScore = calculateScore(houseCards);  }  lblHouseScore.setText("House Score: " + houseScore);  revalidate();  repaint();  // Determine the game outcome and update the win/loss record accordingly.  if (playerScore >21 || (houseScore <= 21 &&houseScore >playerScore)) {  handsLost++;  lblHandsLost.setText("Hands lost: " + handsLost);  JOptionPane.showMessageDialog(this, "House wins!");  } else if (houseScore >21 || playerScore >houseScore) {  handsWon++;  lblHandsWon.setText("Hands won: " + handsWon);  JOptionPane.showMessageDialog(this, "Player wins!");  } else {  JOptionPane.showMessageDialog(this, "It's a draw!");  }  newGame();  }  // Create a new deck of 52 playing cards.  private ArrayList<String>createDeck() {  ArrayList<String>deck = new ArrayList<>();  String[] suits = {"c", "d", "h", "s"};  for (String suit : suits) {  for (int i = 1; i <= 10; i++) {  deck.add(suit + i);  }  deck.add(suit + "j");  deck.add(suit + "q");  deck.add(suit + "k");  }  return deck;  }  // Determine the value of a card for scoring purposes.  private int cardValue(String card) {  String value = card.substring(1);  if (value.equals("a")) {  return 1; // Return 1 for Ace  }  try {  return Integer.parseInt(value);  } catch (NumberFormatException e) {  return 10;  }  }  // Calculate the score for a given hand of cards.  private int calculateScore(ArrayList<String>cards) {  int score = 0;  int aces = 0;  for (String card : cards) {  int cardValue = cardValue(card);  score += cardValue;  if (cardValue == 1) {  aces++;  }  }  // If the hand contains an Ace and the score is 11 or less, add 10 to the score.  while (score <= 11 &&aces >0) {  score += 10;  aces--;  }  return score;  }  // Load an image of a playing card from the resource folder.  private ImageIcon loadImage(String cardName) {  try {  String resourcePath = "/cards/" + cardName + ".png";  System.out.println("Loading image: " + resourcePath);  InputStream in = getClass().getResourceAsStream(resourcePath);  BufferedImage img = ImageIO.read(in);  return new ImageIcon(img);  } catch (IOException e) {  System.err.println("Error loading card image: " + cardName);  e.printStackTrace();  return null;  }  }  } |