**Euchre Project Part 4**

**Due, Sunday, July 8, 2018 for maximum 100%**

**Monday, July 9, 2018 for maximum 90%**

**Tuesday, July 10, 2018 for maximum 80%**

**Wednesday, July 11, 2018 for maximum 70%**

**Assignment Scope**

1. Update classes due to changing requirements
2. Adding fields to enumerations
3. Add methods to classes to accomplish specific tasks
4. Begin developing the front end, or User Interface, portion of the Euchre game. Based on my professional experience working in the industry I have ***always*** had to develop a UI for every application, therefore I translate that experience to students so they can have the same opportunity and be prepared professionally.
5. Typically, there is a one-to-one correlation of back end functionality to front end UI component. Depending upon the design of the application it doesn’t always correlate perfectly, however with Euchre, it works well.

|  |  |
| --- | --- |
| **Back-end functionality** | **Front-end UI component** |
| AiPlayer.java | AiPlayerUi.java |
| Card.java | CardUi.java |
| Game.java | GameUi.java |
| HumanPlayer.java | HumanPlayerUi.java |

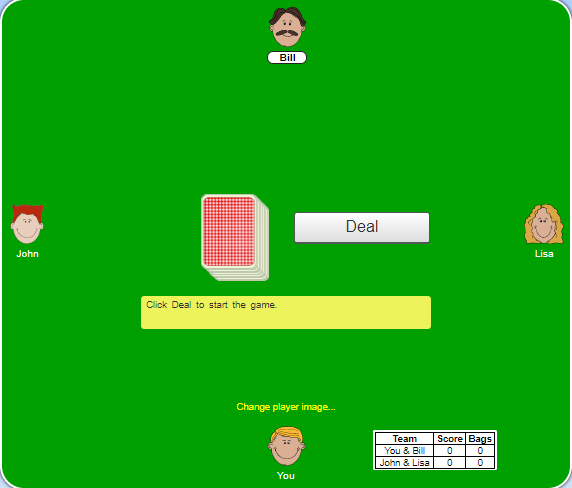
1. The goal is to develop the front-end components of the game Euchre by creating classes:
2. AiPlayerUi.java
3. GameUi.java
4. HumanPlayerUi.java
5. Students will also begin to learn writing simple ActionListeners or Event Handlers.
6. The UI will be developed in multiple assignments, it is ***not*** expected that for Assignment 7 the fully functioning UI is complete.

The image that follows is a prototype of what the UI will look like. It does ***not*** have to be an exact match. The rubric will provide guidance and recommendations on how to accomplish this, however feel free to be creative in developing the look and feel of the UI.

**References**

1. Netbeans.docx
2. Setting up a project in Netbeans.docx
3. Netbeans right click menu help.docx

**Prototype**



**Deliverables**

To complete this assignment you must submit your **compressed Netbeans project** to Webcourses.

Please keep in mind that the tasks are guidance to accomplish the goals of the assignment. At times students will be required to do additional research (e.g. Google That S\*\*T (GTS)!) to find implementation options. In the industry, software engineers are **expected** to be very self-sufficient to find the best solution for the task at hand.

***I have provided multiple code examples on Webcourses that shows how to implement numerous of the tasks below, please reference those code examples prior to asking me for help!***

**Tasks**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Activity | | | | |
| Euchre project | | |  | |
| Euchre class | | | 1. Add to method main()   Instantiate an instance of class GameUi, passing the reference object of class Game as an argument | |
| constants | | |  | |
| Constants class | | |  | |
|  | | | 1. Update enum Face to include the following 2. Nine has the value of 9 3. Ten has the value of 10 4. Jack has the value of 11 5. Queen has the value of 12 6. King has the value of 13 7. Ace has the value of 14 8. Add field **value** of data type int 9. Add a public getter to return the field **value** 10. Add a private constructor for enum Face with one parameter of type int; set the field **value** to the parameter passed in 11. Update enum Suit to include the following 12. Clubs has the rank of 0 13. Diamonds has the rank of 1 14. Hearts has the rank of 2 15. Euchre has the rank of 3 16. Add field **rank** of data type int 17. Add a public getter to return the field **rank** 18. Add a private constructor for enum Suit with one parameter of type int; set the field **rank** to the parameter passed in 19. Example code:   public enum Suit  {  CLUBS (0),  DIAMONDS (1),  HEARTS (2),  EUCHRE (3);    private int rank;    public int getRank()  {  return rank;  }    private Suit(int rank)  {  this.rank = rank;  }  } | |
| core package | | |  | |
| AiPlayer class | | |  | |
| Card class | | |  | |
| Deck class | | | Update class to do the following:   1. Add member variable of interface List, specifically only allowing for instances of class Card to be added named cardList 2. Generate the getter/setter for the member variable above 3. Update method shuffleDeck() so that it instantiates the member variable cardList instead of creating a new instance of interface List 4. Write method displayCardList so that is does the following:    1. Return type void    2. Empty parameter list    3. Using an enhanced for loop, loop through the ArrayList of cards and output to the console the card value, face, and color 5. Update the customer constructor to do the following:    1. Calls method shufflDeck again    2. Calls method displayCardList | |
| Game class | | | Update the class to do the following:   1. Custom constructor Game()    1. Comment out the call to method outputTeams()    2. Add call to method play() 2. Update method setTable() to do the following:    1. Using an enhanced for loop output the names of each Player in the member variable **table** representing the setup of the players seated at the table 3. Update method dealHand() where the Iterator instance is instantiated, change method call getDeck() in class Deck to method call getCardList() 4. Update method dealOne() to comment out any System.out.println() statements that show what card each player is being dealt if they exist 5. Update method dealTwo() to comment out any System.out.println() statements that show what card each player is being dealt if they exist 6. Create method play() to do the following:    1. Return type of void    2. Empty parameter list    3. Using an enhanced for loop, loop through the players at the table; for each player, call method makeTrump() 7. Update method createTeams    1. Replace the use of the console prompt and class Scanner to get the human player’s name with static method call on class JOptionPane using method showInputDialog passing as an argument the explicit text to prompt the human to enter their name    2. Store the data entered in the input dialog in the variable of type String 8. Add a getter ONLY for member variable of data type class ArrayList that represents the table of players | |
| HumanPlayer class | | |  | |
| IPlayer interface | | |  | |
| Player class | | | Update class to include the following:   1. Add method sortBySuit() to do the following:    1. Instantiate an instance of class ArrayList allowing only for instances of class Card to be elements that represents the sorted hand    2. Loop while the size of member variable **hand** is greater than zero (0)       1. Create a local variable of type int representing the current position in the ArrayList of member variable **hand**, initialize it to zero (0)       2. Instantiate and instance of class Card (reference as firstCard) set equal to the first element in the ArrayList of member variable **hand** (hint: use the .get() method of class ArrayList)       3. Using a for loop, starting at index 1, loop through the size of member variable ArrayList **hand** to do the following:          1. Instantiate an instance of class Card (reference as nextCard) set equal to the next element in the ArrayList **hand** (i.e. index 1)          2. Write an if statement to sort the cards based on suit and face value ranking with the following logic:             1. If the suit’s rank of the nextCard in the hand is less than the suit’s rank of the firstCard in the hand             2. OR   The suit of the nextCard is equal to the suit of the firstCard  AND  The face value of the nextCard in the hand is less than the face value of the firstCard in the hand  Update local variable position to equal the for loop’s looping variable  Set the firstCard equal to the nextCard   * + 1. Remove from the member variable hand the card at the current position (hint: on member variable **hand** call method remove() passing the argument defined in step i.)     2. Add the instance represented as firstCard to the local ArrayList that represents the sorted hand created in step a.   1. Set member variable **hand** equal to the local ArrayList that represents the sorted hand. | |
| Team class | | | Update method outputHands() to do the following:   * 1. Call method sortBySuit() for each player   2. Only call method displayHand() if the player is an **instanceof** class HumanPlayer | |
| userinterface package | | |  | |
| GameUi.java | | 1. Add member variables of type    1. Game game;    2. JFrame frame;    3. JPanel aiOnePanel;    4. JPanel tablePanel;    5. JPanel aiTwoPanel;    6. JPanel hpPanel;    7. JPanel aiThreePanel;    8. JPanel northPanel;    9. JPanel scorePanel;    10. JPanel bidPanel;    11. JMenuBar menuBar;    12. JMenu gameMenu;    13. JMenu helpMenu;    14. JMenuItem newGameMenuItem;    15. JMenuItem exitMenuItem;    16. JMenuItem aboutMenuItem;    17. JMenuItem rulesMenuItem; | |
|  | | 1. A custom constructor should be defined that receives a parameter of data type Game class    1. Set member variable of type class Game to the parameter passed in    2. Call method initComponents() | |
|  | | 1. A method initComponents() should initialize all the components for the UI and be called from the constructor    1. Set the size of the JFrame    2. Set the default close operation of the JFrame    3. Use default layout manager BorderLayout    4. Set up the JMenuBar       1. JMenu Game should be added to the JMenuBar       2. JMenuItems New Game and Exit should be added to the JMenu Game       3. JMenu Help should be added to the JMenuBar       4. JMenuItems About and Game Rules should be added to the JMenu Help    5. JMenuBar should be set on the JFrame    6. Instantiate member variable aiOnePanel by calling the constructor for class AiPlayerUi, passing as arguments       1. the instance of class Player stored in class Game, member variable table, at the second position of the ArrayList       2. An integer representing the position of the Player object in the ArrayList       3. Example: aiOnePanel = new AiPlayerUi(game.getTable().get(Constants.POSITION\_2), Constants.POSITION\_2);    7. Instantiate member variable aiTwoPanel by calling the constructor for class AiPlayerUi, passing as arguments       1. the instance of class Player stored in class Game, member variable table, at the second position of the ArrayList       2. An integer representing the position of the Player object in the ArrayList    8. Instantiate member variable aiThreePanel by calling the constructor for class AiPlayerUi, passing as arguments       1. the instance of class Player stored in class Game, member variable table, at the second position of the ArrayList       2. An integer representing the position of the Player object in the ArrayList    9. Instantiate member variable humanPanel by calling the constructor for class HumanPlayerUi, passing as an argument       1. the instance of class Player stored in class Game, member variable table, at the second position of the ArrayList    10. Instantiate member variable northPanel and set the size of the JPanel using the default layout manager FlowLayout    11. Instantiate member variable bidPanel and do the following        1. Set the size of the JPanel        2. Add a border to the JPanel    12. Instantiate member variable scorePanel and do the following        1. Set the size of the JPanel        2. Add a border to the JPanel    13. Resize aiPanelTwo so it will fit in the northPanel with bidPanel and scorePanel    14. Add to northPanel        1. scorePanel        2. aiTwoPanel        3. bidPanel    15. Instantiate member variable tablePanel and do the following        1. Set the size of the JPanel        2. Add a border to the JPanel    16. Add the JPanels to the JFrame in their appropriate locations based on using layout manager BorderLayout    17. Set the visibility of the JFrame (hint: this should ALWAYS be the last step on a UI) | |
|  | | 1. Write an inner class to create an ActionListener that is registered to the JMenuItem with the text Exit; it should 2. Display a JOptionPane message confirming the user wants to exit using method showConfirmDialog() 3. If yes, exit the application by calling method System.exit() passing the value of 0 as an argument 4. If no, do not exit the application | |
|  | | 1. Write an inner class to create an ActionListener that is registered to the JMenuItem with the text About using method showMessageDialog(); it should 2. Display a JOptionPane message informing the user:    1. Application name and version    2. Author    3. Date of development | |
|  | | 1. Write an inner class to create an ActionListener that is registered to the JMenuItem with the text Game Rules using method showMessageDialog(); it should 2. Display a JOptionPane message informing the user:    1. Rules of the game as shown in Figure 6 below | |
| AiPlayerUi.java | 1. Create class AiPlayerUi so it uses class JPanel as the superclass 2. Add member variables of type:    1. AiPlayer ai;    2. int position;    3. ArrayList<JLabel> cards; | | |
|  | 1. A custom constructor should be defined that receives a two parameters; one is data type class Player, the other is data type int    1. Set member variable of type class AiPlayer to the parameter passed in of class Player using an explicit type cast    2. Set member variable position to the parameter of data type int    3. Call method initComponents() | | |
|  | 1. Write method initComponents() to initialize all the components for the UI and be called from the constructor    1. Set the size of the JPanel using two methods to ensure the UI isn’t too small       1. setMinimumSize()       2. setPreferredSize()    2. Instantiate the member variable of data type ArrayList containing elements of class JLabel    3. Using an if statement check if the position is 1 or 3       1. If true, set the layout manager to use BoxLayout so it is aligned on the Y axis       2. Else, set the layout manager to use BoxLayout so it is aligned on the X axis    4. Call method displayCards() | | |
|  | 1. Write method displayCards to do the following    1. Loop through the member variable of data type ArrayList containing elements of class JLabel so there are 5 JLabels       1. Instantiate an instance of class JLabel       2. Set the size of the JLabel       3. Add a border to the JLabel       4. Set the text of the JLabel to explicit text “Card” concatenated with the looping variable       5. Add the JLabel to the ArrayList       6. Add the JLabel to the JPanel | | |
| HumanPlayerUi.java | 1. Create class HumanPlayerUi so it uses class JPanel as the superclass 2. Add member variables of type:    1. HumanPlayer human;    2. ArrayList<JButton> cards; | | |
|  | 1. A custom constructor should be defined that receives a one parameter, data type class Player 2. Set member variable of type class HumanPlayer to the parameter passed in of class Player using an explicit type cast 3. Call method initComponents() | | |
|  | 1. Write method initComponents() to initialize all the components for the UI and be called from the constructor 2. Set the size of the JPanel using two methods to ensure the UI isn’t too small    1. setMinimumSize()    2. setPreferredSize()    3. Instantiate the member variable of data type ArrayList containing elements of class JButton    4. Se the layout manager to use BoxLayout so it is aligned on the X axis    5. Call method displayCards() | | |
|  | 1. Write method displayCards to do the following 2. Loop through the member variable of data type ArrayList containing elements of class JButton so there are 5 JButtons 3. Instantiate an instance of class JButton 4. Set the size of the JButton 5. Add a border to the JButton 6. Set the text of the JButton to explicit text “Card” concatenated with the looping variable 7. Add the JButton to the ArrayList 8. Add the JButton to the JPanel | | |
|  |  | | |
| CardUi.java | Create the class | | |
| Euchre application | | |  | |
|  | | |  | |
| Test Case 1 | | | Test Case 1 passes | |
| Test Case 2 | | | Test Case 2 passes | |
| Test Case 3 | | | Test Case 3 passes | |
| Test Case 4 | | | Test Case 4 passes | |
| Test Case 5 | | | Test Case 5 passes | |
| Test Case 6 | | | Test Case 6 passes | |
| Test Case 7 | | | Test Case 7 passes | |
| Test Case 8 | | | Test Case 8 passes | |
| Test Case 9 | | | Test Case 9 passes | |
| Test Case 10 | | | Test Case 10 passes | |
| Test Case 11 | | | Test Case 11 passes | |
|  | | | Source compiles with no errors | |
|  | | | Source runs with no errors | |
|  | | | Source includes comments | |

**Perform the following test cases**

|  |  |  |
| --- | --- | --- |
| Test Cases | | |
|  | **Action** | **Expected outcome** |
| Test Case 1 | **Project view** | Completed project view should look like figure 1 |
| Test Case 2 | **Regression testing; Run application** | The JOptionPane.showMessageDialog() method call should look like figure 2 |
| Test Case 3 | **Run application;**  **Updated method setTable()** | Output should be similar to figure 3 |
| Test Case 4 | **Run application;**  **Updated method .displayHands()** | Output should be similar to figure 4 where only the human player’s hand is displayed |
| Test Case 6 | **JOptionPane Prompts User Name** | JOptionPane is similar to figure 5 |
| Test Case 7 | **Euchre Initial UI displays** | Game UI looks similar figure 6 |
| Test Case 8 | **Euchre Game Menu** | Game menu looks similar to figure 7 |
| Test Case 9 | **Euchre Help Menu** | Help menu looks similar to figure 8 |
| Test Case 10 | **About Menu Item Action Listener** | JOptionPane displays similar to figure 9 |
| Test Case 11 | **Exit Menu Item Action Listener** | JOptionPane displays similar to figure 10  if user selects yes, the application should exit;  if user selects no, the application continues to run |

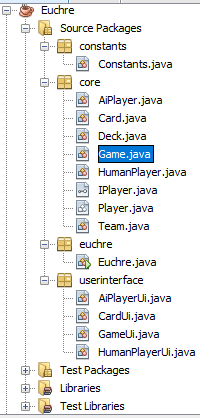
****

Figure 1 Project View

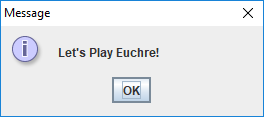


Figure 2 Display from JOptionPane.showMessageDialog() method

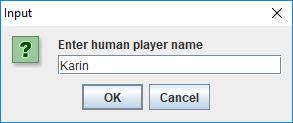


Figure 3 Prompt for User’s Name

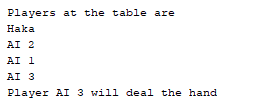


Figure 4 Players at the table

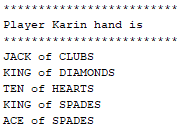


Figure 5 Only human player sorted hand should display

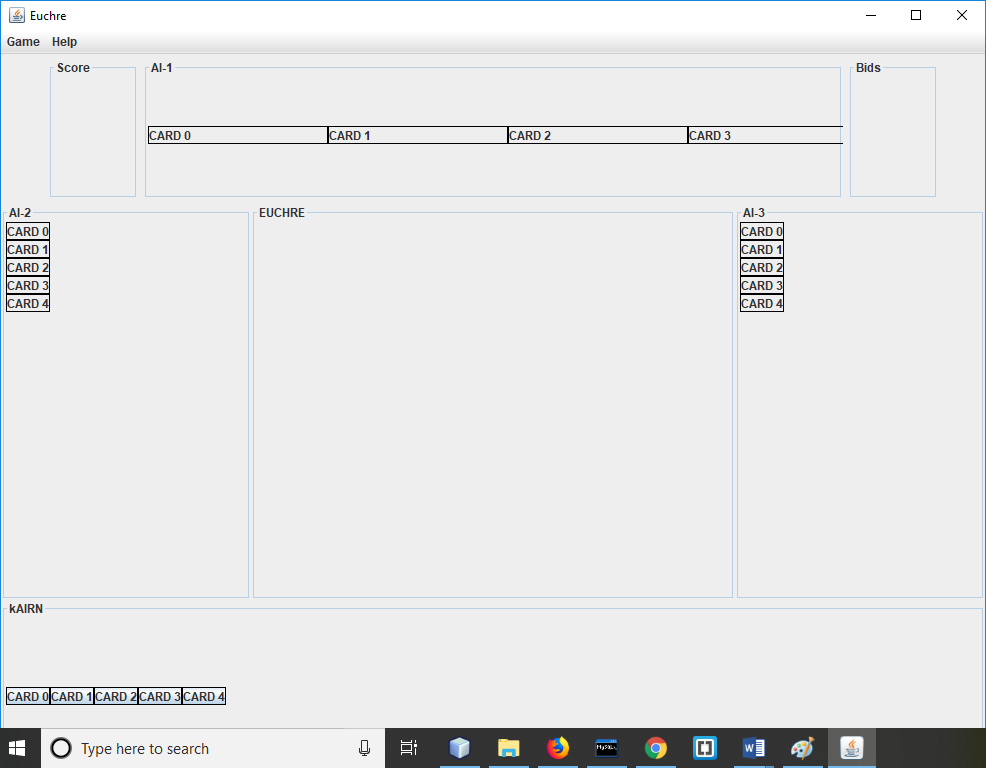


Figure 6 Euchre Initial UI

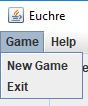


Figure 7 Game Menu

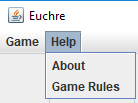


Figure 8 Help Menu

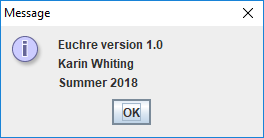


Figure 9 About Menu Item

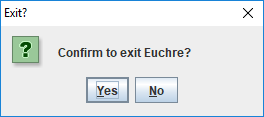


Figure 10 Exit