The University of Hong Kong

Department of Computer Science

COMP2396 Object-oriented Programming and Java

Assignment 4

Deadline: 11:55pm, 18th Apr, 2019.

Overview

This assignment tests your understanding of GUI and event-handling, and their implementations in Java. You are going to design and implement a GUI for the Big Two card game you developed in assignment 3. A number of classes and interfaces will be provided to aid your implementation. These include all the classes provided in assignment 3, a CardGame interface which models a general card game, and a CardGameTable interface which models a GUI for a general card game. You may refer to their Javadoc for details of these classes and interfaces. You should **NOT** modify any of these classes and interfaces in completing your assignment.

You are required to implement a BigTwoTable class which builds a GUI for the game and handles all user actions. This class implements the CardGameTable interface and has a number of inner classes responsible for rendering the user interface and handling user actions. In order to make use of the BigTwoTable class, you will need to modify your BigTwo class accordingly. In particular, you will need to make your BigTwo class implement the CardGame interface. You are free to introduce new instance variables and methods to these classes. Besides, you are also free to design and introduce new classes in the inheritance trees as appropriate. With a proper OO design, it is most likely not necessary for you to touch the rest of the classes you implemented in assignment 3. You should write Javadoc for all public classes and their public class members.

Specifications

Graphical user interface

You are free to design a GUI for your Big Two card game. As a minimum requirement, your GUI should

- Have a panel showing the cards of each player as well as the cards played on the table. The cards should be shown in a partially overlapped manner (see Figure 1).
- For each player, the panel should show his/her name and an avatar for him/her.
- For the active player, the panel should show the faces of his/her cards.
- For other players, the panel should show only the backs of his/her cards.
- For cards played on the table, the panel should show at least (the faces of) the last hand of cards played on the table and the name of the player for this hand.
- Allow the active player to select and deselect his/her cards by mouse clicks. The selected cards should be drawn in a "raised" position with respect to the rest of the cards (see Figure 1).
- Have a "Play" button for the active player to play the selected cards.
- Have a "Pass" button for the active player to pass his/her turn to the next player.
- Have a text area to show the current game status as well as end of game messages.
- Have a "Restart" menu item for restarting the game.
- Have a "Quit" menu item for quitting the game.

The BigTwoTable class

The BigTwoTable class implements the CardGameTable interface. It is used to build a GUI for the Big Two card game and handle all user actions. Below is a detailed description for the BigTwoTable class.

Specification of the BigTwoTable class:

public constructor:

`BigTwoTable(CardGame game)` – a constructor for creating a BigTwoTable. The parameter `game` is a reference to a card game associates with this table.

private instance variables: *

`CardGame game` – a card game associates with this table.

`boolean[] selected` – a boolean array indicating which cards are being selected.

`int activePlayer` – an integer specifying the index of the active player.

`JFrame frame` – the main window of the application.

`JPanel bigTwoPanel` – a panel for showing the cards of each player and the cards played on the table.

`JButton playButton` – a "Play" button for the active player to play the selected cards.

`JButton passButton` – a "Pass" button for the active player to pass his/her turn to the next player.

`JTextArea msgArea` – a text area for showing the current game status as well as end of game messages.

`Image[][] cardImages` – a 2D array storing the images for the faces of the cards.

`Image cardBackImage` – an image for the backs of the cards.

`Image[] avatars` – an array storing the images for the avatars.

* *These are just suggestions to aid your design. It is perfectly fine if your actual implementation deviates from these suggestions.*

CardGameTable interface methods:

`void setActivePlayer(int activePlayer)` – a method for setting the index of the active player (i.e., the current player).

`int[] getSelected()` – a method for getting an array of indices of the cards selected.

`void resetSelected()` – a method for resetting the list of selected cards.

`void repaint()` – a method for repainting the GUI.

`void printMsg(String msg)` – a method for printing the specified string to the message area of the GUI.

`void clearMsgArea()` – a method for clearing the message area of the GUI.

`void reset()` – a method for resetting the GUI. You should (i) reset the list of selected cards using `resetSelected()` method from the CardGameTable interface; (ii) clear the message area using the `clearMsgArea()` method from the CardGameTable interface; and (iii) enable user interactions using the `enable()` method from the CardGameTable interface.

`void enable()` – a method for enabling user interactions with the GUI. You should (i) enable the "Play" button and "Pass" button (i.e., making them clickable); and (ii) enable the BigTwoPanel for selection of cards through mouse clicks.

`void disable()` – a method for disabling user interactions with the GUI. You should (i) disable the "Play" button and "Pass" button (i.e., making them not clickable); and (ii) disable the BigTwoPanel for selection of cards through mouse clicks.

inner classes: *

`class BigTwoPanel` – an inner class that extends the JPanel class and implements the MouseListener interface. Overrides the `paintComponent()` method inherited from the JPanel class to draw the card game table. Implements the `mouseClicked()` method from the MouseListener interface to handle mouse click events.

`class PlayButtonListener` – an inner class that implements the ActionListener interface. Implements the `actionPerformed()` method from the ActionListener interface to handle button-click events for the "Play" button. When the "Play" button is clicked, you should call the `makeMove()` method of your CardGame object to make a move.

`class PassButtonListener` – an inner class that implements the ActionListener interface. Implements the `actionPerformed()` method from the ActionListener interface to handle button-click events for the "Pass" button. When the "Pass" button is clicked, you should call the `makeMove()` method of your CardGame object to make a move.

`class RestartMenuItemListener` – an inner class that implements the ActionListener interface. Implements the `actionPerformed()` method from the ActionListener interface to handle menu-item-click events for the "Restart" menu item. When the "Restart" menu item is selected, you should (i) create a new BigTwoDeck object and call its `shuffle()` method; and (ii) call the `start()` method of your CardGame object with the BigTwoDeck object as an argument.

`class QuitMenuItemListener` – an inner class that implements the ActionListener interface. Implements the `actionPerformed()` method from the ActionListener interface to handle menu-item-click events for the "Quit" menu item. When the "Quit" menu item is selected, you should terminate your application. (You may use `System.exit()` to terminate your application.)

* *These are just suggestions to aid your design. It is perfectly fine if your actual implementation deviates from these suggestions.*

The BigTwo class

The BigTwo class implements the CardGame interface. It is used to model a Big Two card game. Below is a detailed description for the BigTwo class.

Specification of the BigTwo class:

public constructor:

`BigTwo()` – a constructor for creating a Big Two card game. You should (i) create 4 players and add them to the list of players; and (ii) create a Big Two table which builds the GUI for the game and handles user actions.

private instance variables:

`Deck deck` – a deck of cards.

`ArrayList<CardGamePLayer> playerList` – a list of players.

`ArrayList<Hand> handsOnTable` – a list of hands played on the table.

`int currentIdx` – an integer specifying the index of the current player.

`BigTwoTable table` – a Big Two table which builds the GUI for the game and handles all user actions.

CardGame interface methods:

`int getNumOfPlayers()` – a method for getting the number of players.

`Deck getDeck()` – a method for getting the deck of cards being used.

`ArrayList<CardGamePlayer> getPlayerList()` – a method for getting the list of players.

`ArrayList<Hand> getHandsOnTable()` – a method for getting the list of hands played on the table.

`int getCurrentIdx()` – a method for getting the index of the current player.

`void start(Deck deck)` – a method for starting/restarting the game with a given shuffled deck of cards. You should (i) remove all the cards from the players as well as from the table; (ii) distribute the cards to the players; (iii) identify the player who holds the 3 of Diamonds; and (iv) set both the `currentIdx` of the BigTwo instance and the `activePlayer` of the BigTwoTable instance to the index of the player who holds the 3 of Diamonds.

`void makeMove(int playerID, int[] cardIdx)` – a method for making a move by a player with the specified playerID using the cards specified by the list of indices. This method should be called from the BigTwoTable when the active player presses either the "Play" or "Pass" button. You should simply call the `checkMove()` method from the CardGame interface with the `playerID` and `cardIdx` as the arguments. *

`void checkMove(int playerID, int[] cardIdx)` – a method for checking a move made by a player. This method should be called from the `makeMove()` method from the CardGame interface. *

`boolean endOfGame()` – a method for checking if the game ends.

** It may not make much sense at this moment why we need to break the playing of a hand of cards by a player into two methods, namely* `makeMove()` *and* `checkMove()`*. It will become clear in assignment 5 when we implement a network version of the game, that* `makeMove()` *is used by the current player to broadcast his/her move to others over the network whereas* `checkMove()` *is used by all users to validate the move made by the current player upon receiving the details of the move over the network.*

public static methods:

`void main(String[] args)` – a method for starting a Big Two card game. It should (i) create a Big Two card game; (ii) create and shuffle a deck of cards; and (iii) start the game with the deck of cards.

`Hand composeHand(CardGamePlayer player, CardList cards)` – a method for returning a valid hand from the specified list of cards of the player. Returns `null` if no valid hand can be composed from the specified list of cards.

Sample output

Figure 1 shows an example of GUI for the Big Two card game that satisfies the minimum requirement. You are not required to reproduce the same GUI exactly.

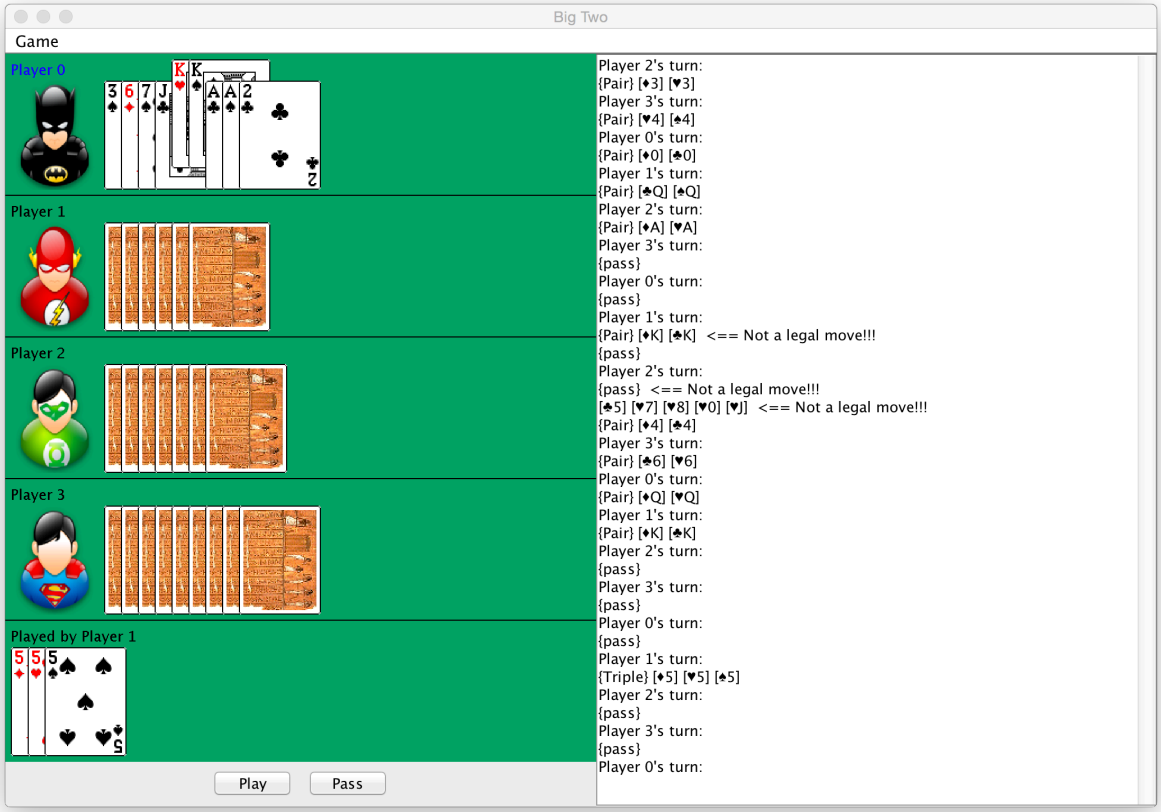

Figure 1. An example of GUI for the Big Two card game.

You may download the card images used in the above example from *

https://www.waste.org/~oxymoron/cards/

* *You are free to use any other card images.*

Marking Scheme

Marks are distributed as follows:

- Implementation of the BigTwo class (10%)
- Implementation of the BigTwoTable class
  - Rendering of players' cards and cards on the table (20%)
  - Implementation of card selection using mouse clicks (20%)
  - Implementation of the message area (10%)
  - Implementation of the "Play" button (5%)
  - Implementation of the "Pass" button (5%)
  - Implementation of the "Restart" menu item (5%)
  - Implementation of the "Quit" menu item (5%)
  - Overall design of the GUI (10%)
- Javadoc and comments (10%)

Submission

Please pack the source code (`*.java`) and images of your application into a single zip file, and submit it to the course Moodle page.

A few points to note:

- Always remember to write Javadoc for all public classes and their public class members.
- Always remember to submit the source code files (`*.java`) but **NOT** the bytecode files (`*.class`).
- Always double check after your submission to ensure that you have submitted the most up-to-date source code files.
- Your assignment will not be marked if you have only submitted the bytecode files (`*.class`). You will get zero mark for the assignment.
- Please submit your assignment on time. Late submission will not be accepted.

~ End ~