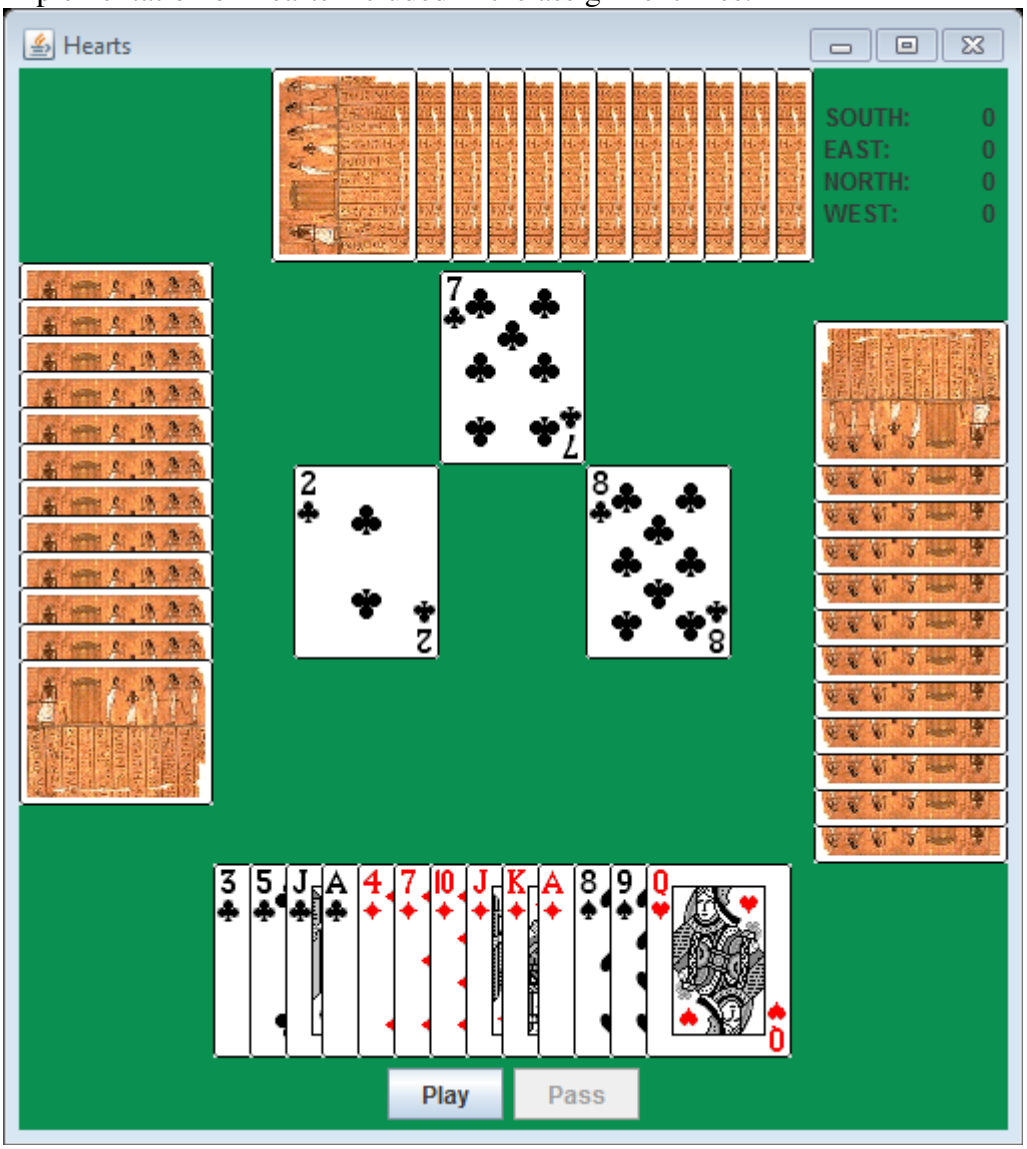
CECS 220: Assignment 5 – Card Game of Choice

By Brent Wenerstrom

Card Game of Choice

In this assignment, you'll be implementing a card game. As an example consider the implementation of Hearts included in the assignment files.



We provide the Java code to represent cards, ranks, suits, hands, and GUI elements for hands and playing areas. Your job is to design the GUI interface and implement the rules of the chosen card game.

Don't be fooled by the simplicity of using GUIs. Designing and building interactive, intuitive GUIs is a lot of work. Make sure to start early to understand the code provided and how GUIs are put together.

When it comes to choosing a card game I am fairly flexible, but you will need to find some authoritative reference with a listing of the rules of the game. Most traditional card games have rule listings on pagat.com. For commercial card games please find a reference to the rules from the publishing company.

For the ambitious student, I provide two ways of earning extra credit on this assignment.

1. (+30 points) Turn your card game into an Android app and show me the finished product on an Android phone.
2. (+10 points) Implement my favorite card game Tichu Nanking (http://www.fatamorgana.ch/tichu/Tichu_english.html). Be warned this game has at least twice as many rules as most traditional card games and requires 4 unique cards not included in the card images for this assignment.

You may find this assignment to be fairly challenging in places. Make sure you leave yourself enough time to make your game interesting, as well as functional.

Reference

GUI Classes

```
DisplayHand
static void setMultipleCardsSelectable(boolean b)
DisplayHand(int width, Hand hand)
List<Card> getSelected()
void remove(Card card)
void update()

OpponentHand
OpponentHand(Orientation orientation, int length, Hand hand)
void update()

PlayArea
PlayArea(int width, int height, Color color)
void clear()
void showCard(Card card, int x, int y)
```

Game Representation Classes

```
Card
static void setRankMajorSort()
static void setSuitMajorSort()
Card(Suit suit, Rank rank)
int compareTo(Object otherCardObject)
Rank getRank()
Suit getSuit()
boolean isSameAs(Card card)
String toString()
```

Deck
```
Deck()
void addCard(Card card)
Card dealCard()
int getNumberOfCardsRemaining()
int getSizeOfDeck()
boolean isEmpty()
void restoreDeck()
void shuffle()
```

Hand
```
void addCard(Card card)
int compareTo(Object otherHandObject)
boolean containsCard(Card card)
void discardHand()
int evaluateHand() // not implemented
int findCard(Card card)
Card getCard(int index)
int getNumberOfCards()
boolean isEmpty()
Card removeCard(Card card)
Card removeCard(int index)
boolean replaceCard(Card oldCard, Card replacementCard)
void sort()
String toString()
```

Rank
```
static void setAceHigh()
static void setKingHigh()
int compareTo(Object otherRankObject)
String getName()
String getSymbol()
String toString()
```

Suit
```
int compareTo(Object otherSuitObject)
String getName()
String getSymbol()
String toString()
```

Steps to Completion

Download and unzip CardGame.zip, which contains all the files you will need for this assignment.

1. Become familiar with Deck, Hand, Card, Suit and Rank classes.

   The ***Deck*** class is the object that will contain all cards used in a single game. Before your game begins you will need to create a deck with the particular cards used in your game, the correct proportions of cards, etc. From this deck all hands will be dealt.

   Each card in the deck will need to be created using the ***Card*** class. Each card has a single rank (2-10,J,Q,K,A) and suit (♠,♥,♦,♣). Each possible rank of a card in a standard deck of cards is implemented as a constant in the ***Rank*** class. For example the rank 2 is

implemented as ***Rank.TWO***. The ***Suit*** class contains constant for each of the possible suits. For example, to get a card with a heart suit you would use Suit.HEARTS. Therefore, to create a single card representing the two of clubs you would use the following code:

```
Card c = new Card(Suit.CLUBS, Rank.TWO);
```

The ***Hand*** class is meant to store a number of cards. It may be used to for a player's hand in the game, to represent a trick to which players are playing, a pile of cards won, a nest of cards, etc. It simple is an object containing several cards. Use it as you see fit, or feel free to use any ***Collections*** type object from the standard Java library.

Now practice using the different game representation classes using the interaction pane. Create a deck object and a hand object. Create a couple cards. Add each one to the deck object. Shuffle the cards in the deck using the ***shuffle*** method. Now deal a card to your hand using the ***dealCard*** function.

For an example of creating a standard 52 card deck using loops, see the ***Hearts*** class under the ***initializeDeck*** method.

2. Design and create a GUI.

   The GUI you create in this step should not be functional. At this point you are simply deciding what pieces you need in your GUI and where to place them. You will likely need to represent the players hand of card(s), perhaps one or more opponent hands, some playing area, scores of the players (depending on the game), and GUI objects that allow for interaction. The ***HeartsGUI*** class does most of this for a four a player game. Feel free to use the code in this class and change it as you see fit.

   Design first on paper what you want your GUI to look like. Then create GUI objects placing them one at a time where you would like them to go. Test out your GUI often to see how well it matches your design. Feel free to change your design as often as you like.

   The ***DisplayHand*** class is a class that I have implemented for your convenience. It allows you to enter the number of pixels wide the hand is and to pass in the cards contained in the DisplayHand. With this information a hand is created showing the cards passed in the Hand object. The cards can be selected by the user at this point. If the getSelected() method is called on the DisplayHand object, then all selected cards will be returned. When a card is removed during the game for the hand represented, updated the DisplayHand using the remove method. The remove method preserves the original positions of all cards displayed. If one or more cards are added, then use the update method.

   You may want to use an ***OpponentHand*** object to display the backs of cards for a hand. This object will display the number of cards visually for some hand object. The OpponenentHand may be oriented in any one of four directions. When the hand object containing cards for the OpponentHand is changed, call the update function, and the number of cards displayed will be updated appropriately.

   Be sure to include GUI objects for interacting with game. The most simple approach is to include one or more buttons. An ***ActionListener*** will need to be implemented to

react to clicking of the button. It will be in this ActionListener that the game will account for user decisions.

The HeartsGUI shows example code using a GridBagLayout. You may use whatever layout you wish. The GridBagLayout is extremely flexible and creates a grid that adjusts to the sizes of the components in each cell of the grid. The HeartsGUI class shows how to display four hands and the playing area where those cards may be placed.

The ***`PlayArea`*** is a very flexible object which will draw cards where ever you may want them in a custom grid area. Each card will be drawn over the top of previously displayed cards. Simply choose a card and an x,y-coordinate for that card. It will then be displayed. There is an example of a single card placement in the HeartsGUI example.

When you are done with this step, you will have all of the visual components of your game present.

3. Create two or more classes (neither can be the GUI class) to implement the logic of your game.

   The first class (or classes) will implement the logic of the game. It should contain the logic for legal plays. This class will contain the objects and variables that track the progress of the game. For example this is where all of the hand objects will be contained. If this game is for two or more players, you will need to implement the AI for all of the players in the game here.

   The second class will inherit from hand and will implement the function ***`evaluateHand`***. This may be used to score taken tricks, evaluate how good a hand is for bidding, etc.

   It is recommended that you not implement all rules of the game all at once. Consider breaking up the rules into logical units that you can implement and test one at a time. For example, with the hearts game, I did not implement passing until trick tacking was implemented. I did not implement “shooting the moon” until the more common cases were accounted for. Also, determining a legal play would all for messages to be shown to the user. Once this legal plays were established it could be used to determine AI player plays.

   Lastly, feel free to alter any of the code provided in any way necessary to complete the game you intend to implement.

4. Determine if you have completed the assignment.

   You will need to have the following components in your game to be considered complete:

   - A deck is created and shuffled.
   - Hand(s) are dealt randomly.
   - The GUI shows the user their cards.

- The GUI shows the presence of all other computer players in the game.
- Cards played are shown in the GUI.
- The player is shown a message and prevented from making illegal plays.
- The AI players do not make illegal plays.
- If the game is scored, scores are shown to the user.
- All the rules of your game or variant are implemented as explained on pagat.com or whatever authority source is used.
- A message is displayed at the conclusion of the game to the user on their placement, winnings, or something else related to winning conditions.
- An entire game may be played without Exception messages appearing.
- All messages to the user needed for play are present in the GUI.

What to turn in

Submit all of the source Java files and images necessary to compile and run your game. Include as a text file instructions for playing your game. Also provide one document that answers the following questions:

- Provide an authoritative source of rules for the game.
- How much time did you spend on this project?
- How would you change the project for future classmates?
- What have you learned through this project?
- Why did you select this game?