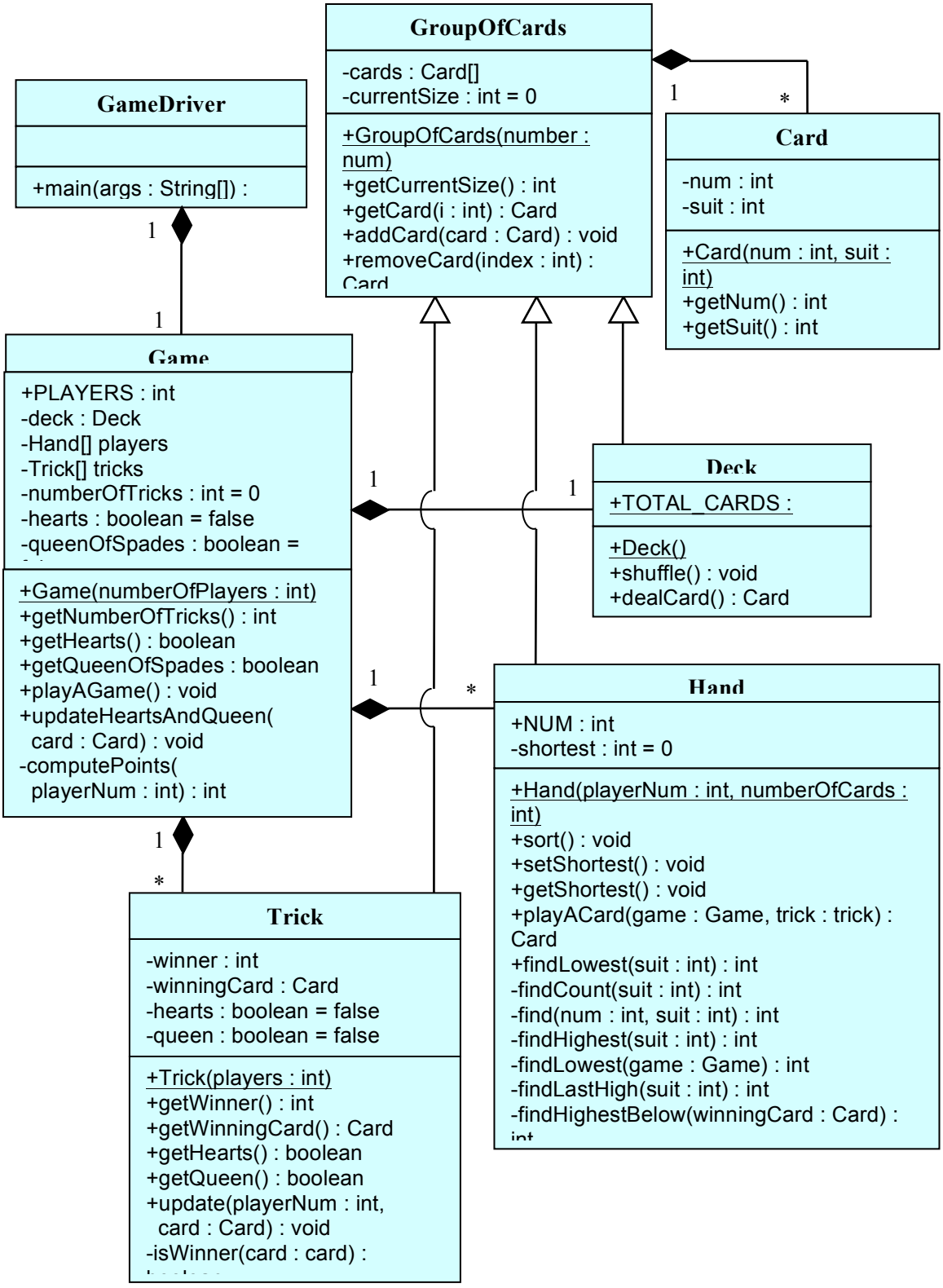
Final Project - Game of Hearts :

This project is a big one, but hopefully the subject will be familiar. Your task is to write a program that plays the game of Hearts. Start with the prototype suggested in the text and enhance it until you get everything in the following UML class diagram.



The `Hand` class is the most difficult one to write, because it includes many of the rules and the strategy for playing the game. Since each player's identification should remain constant after that player has been instantiated, make NUM a final instance variable, and initialize it in the `Hand` constructor with a value equal to the constructor's first parameter value. The second parameter is the maximum number of cards the player will receive. Use it for the base-constructor call argument.

Use a selection sort strategy for the `sort` method. Start with `unsorted` = current size of the array, and step down to `unsorted` = 1. In each step, iterate through the unsorted part of the array, find the card having the greatest value of the expression, (13 * suit + number), and move this card to the high end of the index range. (If you display the result, you should see the cards sorted by suit from Ace of spaces down to 2 of clubs.)

Use the `setShortest` method to determine the best suit to play early in the game, to establish a void as quickly as possible. Start with `shortest` = clubs. If the number of diamonds is less than or equal to the number of clubs, change `shortest` to diamonds. If the number of spades is less than or equal to the shorter of those two, and your spades do not include Ace, King, or Queen, change `shortest` to spades. (Use the `find` method to see if you have an Ace, King, or Queen.)

The `getShortest` method is trivial.

The `playACard` method is the most difficult one, because it contains many of the rules and all of the strategy for winning. We do not know the best way to write this method, because we are not perfect hearts players ourselves, so this is just a suggestion that works reasonably well. All of the methods listed after this one (except for the `count` method) are intended to be used by this method to make this method as simple as possible. Look at them before trying to write this method.

If the current size of the trick is zero (you are the first hand), let a local integer called `index` equal the highest card in your shortest suit, but if this suit is a void let index equal the lowest card in any suit. If the current size of the trick is one less than the total number of players (you are the last hand), and if the trick does not have the queen of spades or any hearts, let `index` equal the value returned by the `findHighest(int suit)` method.

Next, deal with cases where you are the last hand and there are no bad cards in the trick. Since many of the helper methods return −1 if they fail, you can tighten the code in the `playACard` method by using an assignment to `index` inside an empty else-if statement, like this:

```
else if ((trick.getCurrentSize() == game.PLAYERS - 1)
  && !trick.getHearts() && !trick.getQueen()
  && (index = findLastHigh(suit)) >= 0);
```

Then, see what to do if you are a middle hand or the last hand with bad cards already in the trick:

```
else if ((index = findHighestBelow(winningCard)) >= 0);
else if ((index = findMiddleHigh(game, suit)) >= 0);
```

Then, see what to do if you are void, and able to discard a bad card or your own:

```
else if ((index = find(12, 3)) >= 0); // queen of Spades
else if ((index = find(14, 3)) >= 0); // Ace of Spaces
else if ((index = find(13, 3)) >= 0); // King of Spades
else if ((index = findHighest(2)) >= 0); // heart
else
{
  index = findHighest();
}
```

Then, remove the card with the resulting value of `index`, update the trick, update the game, and return the card selected.

Use the `findLowest(int suit)` method to find the lowest club dealt, to start the game. You'll also need it in the rare situation when you have the lead, hearts have not been broken, and hearts are all you have left in your hand. Return the index of the lowest numbered card in the indicated suit. If you have no cards in that suit, return −1.

For the `count` method, return the number of cards in the suit indicated by the value of the parameter.

For the `find` method, return the index of the card having suit and number equal to the parameter values. If you can't find that card, return −1.

Use the `findHighest(int suit)` method to find the highest card in your shortest suit to develop a void as fast as possible early in the game. Also use it to select the highest heart to discard on somebody else's suit. Also use it when you are playing last and there are no bad cards in the trick. Also use as the starting point in the `findHighestBelow` method and the starting point in `findMiddleHigh` method. Return the index of the card having the highest numerical value in the suit indicated by the parameter value. If you have no cards in the suit, return −1.

Use the `findLowest(Game game)` method when leading, after you have developed your void. Return the lowest number in your hand, but not a heart until after hearts have been broken. If hearts have not been broken and all you have left is hearts, return −1.

Use the `findLastHigh` method to return the highest card in the suit led when there are no bad cards in the trick. If this card is the queen of spades, however, and you have another spade, return the highest card you have below your queen.

Use the `findHighestBelow` method when you are neither the first nor last player in a particular trick's play sequence, and you are able to follow suit. Given a reference to the current winning card as the parameter value, search through the cards in your hand whose suit equals the winning card's suit until you find the first one having a number less than the winning card's number, and return the index of that card, but if the next card is a different suit, terminate the search, and return −1.

Use the `findMiddleHigh` method if the `findHighestBelow` method returned −1. Use your highest card in the suit, but if the suit is spades and the queen of spades has not been played yet, try to find a spade that is not higher than the Jack of spades.

Assuming you cannot follow suit and you no longer have the Queen, Ace, or King of spades and no longer have any hearts, use the `findHighest()` method to discard the highest remaining card in your hand, regardless of suit.

`Game` class:

The `Game` class is long but straightforward. Since the number of players should remain constant after a game has been instantiated, make PLAYERS a final instance variable, and initialize it in the `Game` constructor with a value equal to the constructor's parameter value. This constructor should instantiate a `Hand` array with constructor parameter equal to the number of players. It should instantiate individual `Hand` objects for each player, with player identification number and maximum number of cards in a player's hand as constructor arguments. It should also instantiate a `Trick` array with total number of tricks as the constructor argument, but it should not populate this array with any individual tricks.

The `getNumberOfTricks`, `getHearts`, and `getQueenOfSpades` methods are trivial.

Begin the `playAGame` method by shuffling the deck. Evaluate `cardsLeft` after all players get an equal number of cards (example: with five players `cardsLeft` = 2). Loop through all tricks and add one dealt card to each player in succession. Loop through all players and for each player call `sort` the `setShortest` to sort the hand and find that player's best void opportunity. For that player, print the value of `shortest` and display that player's hand to get an output like this (`Math.random` will make the values vary from one execution to the next):

Output – first part:

```
        player 0 shortest= 3
10 of spades
Ace of hearts
King of hearts
Jack of hearts
9 of hearts
2 of diamonds
Jack of clubs
10 of clubs
7 of clubs
6 of clubs
        player 1 shortest= 1
King of spades
6 of spades
Queen of hearts
4 of hearts
3 of hearts
```

```
9 of diamonds
7 of diamonds
Queen of clubs
5 of clubs
3 of clubs
        player 2 shortest= 0
Jack of spades
7 of spades
5 of spades
4 of spades
3 of spades
2 of hearts
Ace of diamonds
8 of diamonds
3 of diamonds
Ace of clubs
        player 3 shortest= 0
Queen of spades
8 of spades
10 of hearts
8 of hearts
6 of hearts
King of diamonds
Queen of diamonds
10 of diamonds
8 of clubs
4 of clubs
        player 4 shortest= 1
Ace of spades
2 of spades
7 of hearts
5 of hearts
Jack of diamonds
6 of diamonds
5 of diamonds
King of clubs
9 of clubs
2 of clubs
```

In this loop, also set `playerNum` equal to the identification number of the player having the lowest club.

Then loop through the total number of tricks. In each iteration, instantiate a new `Trick`, add it to the `tricks` array, and increment `numberOfTricks`. If it's the first trick, set `index` equal to the index of the lowest club in the hand of the player having the lowest club, and set `card` equal to a reference to that card. Have that player remove that card, and update the trick. If it's not the first trick, set `card` equal to the value returned by `player.playACard`. Display the player number and card value for this first play of the trick. Then loop through all remaining players, and for each such player, increment the player number, using % PLAYERS to count in a circle, assign the value returned by `playACard` to `card`, and display the player number and card value. After the loop

through the remaining players is over, set the player number to the winner of the trick. Then, if it's the first trick, deal the rest of the deck's cards to this trick. For each such card, call `updateHeartsAndQueen` to record Hearts broken for the game, print "undelt card" and display that card.

After all the tricks are done, print each player's number and score, using a call to `computePoints`. Recognizing again that `Math.random` will make the details of each execution different, the rest of the output for a game should look like this:

Output – second part:

```
player 4        2 of clubs
player 0        Jack of clubs
player 1        5 of clubs
player 2        Ace of clubs
player 3        8 of clubs
undelt card     9 of spades
undelt card     4 of diamonds

player 2        3 of spades
player 3        8 of spades
player 4        2 of spades
player 0        10 of spades
player 1        King of spades

player 1        9 of diamonds
player 2        8 of diamonds
player 3        King of diamonds
player 4        Jack of diamonds
player 0        2 of diamonds

player 3        4 of clubs
player 4        King of clubs
player 0        10 of clubs
player 1        Queen of clubs
Hearts is now broken
player 2        2 of hearts

player 4        6 of diamonds
player 0        Ace of hearts
player 1        7 of diamonds
player 2        3 of diamonds
player 3        Queen of diamonds

player 3        6 of hearts
player 4        5 of hearts
player 0        King of hearts
player 1        Queen of hearts
player 2        Ace of diamonds

player 0        6 of clubs
```

```
player 1        3 of clubs
player 2        Jack of spades
player 3        Queen of spades
player 4        9 of clubs

player 4        5 of diamonds
player 0        Jack of hearts
player 1        4 of hearts
player 2        7 of spades
player 3        10 of diamonds

player 3        8 of hearts
player 4        7 of hearts
player 0        9 of hearts
player 1        3 of hearts
player 2        5 of spades

player 0        7 of clubs
player 1        6 of spades
player 2        4 of spades
player 3        10 of hearts
player 4        Ace of spades
Player 0 score= 9
Player 1 score= 0
Player 2 score= 0
Player 3 score= 3
Player 4 score= 14
Play another game (y/n)?
```

In the `updateHeartsAndQueen` method, if the parameter card's suit is hearts and `hearts` is still `false`, output "Hearts is now broken" and set `hearts` to `true`. If the parameter card is the queen of spaces set `queenOfSpades` to `true`.

In the `computePoints` method, loop through all tricks. If a particular trick's winner equals this method's parameter value, loop through all cards in that trick. If a card is the queen of spades, add 13 points. For each card that is a heart, add 1 point. Return the total number of points

++

Submit all the (7) .java class files.