Due:

April 6th, 2022, **before 11:59 pm**.

Notes

• Remember to follow the "Programming Standards" for all work you submit, in addition to the guidelines provided here. **Marks can be deducted for not following the programming standards and the guidelines described in this document.**

• Any question about the assignment must be posted on the discussion forum on UMLearn, in the "Assignment 3" thread. You are also responsible for reading the answers and clarifications posted in the discussion forum. Please try to avoid duplicate posts by keeping track of what has been posted and answered already (the search tool in the forum can also be useful).

• As usual, hand-in will be via the submission folder on UMLearn. Make sure you leave enough time before the deadline to ensure your hand-in works properly. The assignments are submitted using UMLearn's time (not the time on your computer). Assignments that are late will be marked according to the late policy in the ROASS.

• Please take the time to **read the entire document before you start planning / coding**. Some ideas will make more sense once you have read the entire guidelines.

• Please test your code often as you build it; not just at the very end.

Assignment 3: 5-card draw poker game in Java

General description

We are coming back to Java to practice using Java interfaces! In this assignment, we are building the simplest version of poker that exists: 5-card draw. In 5-card draw, each player receives 5 cards from the deck. During their turn, the players can choose to discard any number of cards. They will then draw the same number of cards from the deck to replenish their 5-card hand. After that, it's showdown time (everybody shows their hand) and the best hand wins! You can read more about 5-card draw here: https://en.wikipedia.org/wiki/Five-card_draw.

To make things simpler, we will follow these rules:

- there is no betting and no folding in our game (only win or lose)
- any number of cards (from 0 to 5) can be discarded by a player during the draw phase
- only 2 players will play: the human player (you) and a cpu player (you will implement the "AI" for it)

A GUI (Graphical User Interface) is given to you in the 'PokerTableDisplay.java' file. This GUI has already been tested on both Windows and MacOS, using different screen sizes, and has shown good stability. It must not be modified in any way.

Your goal is to implement all the game logic and additional supporting objects. In order to accomplish that, you will design a set of classes that are implementing some interfaces that are given to you. This will ensure the safe connection between your classes and the GUI.

Files provided to you

- Images/backCard2150.png: a folder containing one image file (the back side of a card). The 'Images' folder must be present in the current working directory where you have all other files.
- Various Java interface files: Cardable, Deckable, GameLogicable, Handable, TestableHand[1]. These files must not be modified. The classes that you design will implement them.
- PokerGame.java: This is where the main is. You must make only one addition to this file: call the appropriate constructor of your class that implements the "GameLogicable" interface.
- PokerTableDisplay.java: This contains all the classes responsible for the GUI. You should not make ANY changes in this file. The code in this file is using methods defined in the various interfaces, which means that as long as your classes implement the required methods, the connections should be seamless.
- JUnitTests.java: You must test the compareTo method required by the Handable interface using many @Test methods similar to the one provided to you in this file. Markers will be running complex tests (special cases), so you should come up with difficult tests as well.

Details

In this section, we will go over the different interfaces and talk about the classes that you will write to implement them.

1) Cardable: You should define a class implementing this interface to represent one card (naming this class 'Card' is recommended). One card has a value and a suit (heart, diamond, club or spade). Other properties are required to make the GUI work: a card can be selected or unselected (selected cards will be discarded) and can be face up or face down.

You must implement a toString method for this class, as it is used to draw the card. This toString method should return a String representation of the value of the card (could be a number or J/Q/K/A) and the character representing the suit (you should use the Unicode characters: \u2665, \u2666, \u2660 and \u2663).

An enum (Suit) was created for you in this interface to define the 4 possible values of the Suit. Please read the comments in the file for more information and for descriptions of the expected behaviour of each required method.

2) Deckable: You should define a class implementing this interface that will represent a standard 52-card deck. There must be one card of each value (deuce (2) to ace (A) → 13 in total) for each of the 4 possible suits (13 * 4 = 52). The deck must be created only once during the whole game: required methods provide a way to draw cards and return cards back to the deck after a round.

Please read the comments in the file for more information and for descriptions of the expected behaviour of each required method.

[1] These names are intentionally ridiculous; the idea was to mimic the fact that most built-in Java interfaces end with "able".

3) Handable: You should define a class implementing this interface that will represent one poker hand (naming this class 'Hand' is recommended). A constant is defined in this interface indicating the size of a hand (5 cards).

Please read the comments in the file for more information and for descriptions of the expected behaviour of each required method. In particular, the evaluateHand method should return a String description of the best poker hand that can be made with the cards in the hand (e.g.: "Straight, 9 high"). Your code must consider all possible types of poker hands: straight flush, four of a kind, full house, flush, straight, three of a kind, two pairs, pair, and nothing/no pair/high card. You can read a detailed description of each poker hand type and ranking here: https://www.pagat.com/poker/rules/ranking.html.

Finally, note that the Handable interface extends the Java Comparable interface ('extends Comparable<Handable>'). This means that the class implementing the Handable interface will also need to provide a 'public int compareTo(Handable)' method. This compareTo method will be used to compare the hands of the two players and decide which one wins. As usual with compareTo methods, it should return a negative value (if the hand it was called on is weaker than the parameter), a positive value (if it is stronger) or 0 if both hands are the exact same strength (rare). Each type of hand has a special way to break ties, which are all explained here: https://www.pagat.com/poker/rules/ranking.html. You should use the JUnitTests.java file to produce many tests for your compareTo method (see below).

4) TestableHand: This interface was created specifically for the JUnit tests. It requires a method that allows us to add (an array of) Cardables directly to the hand without drawing them from a deck. This method is for testing only and should never be used outside of the JUnitTests.java file.

Note that this interface is extending the Handable interface, so your 'Hand' class could just implement this interface and all the methods of Handable will be required as well.

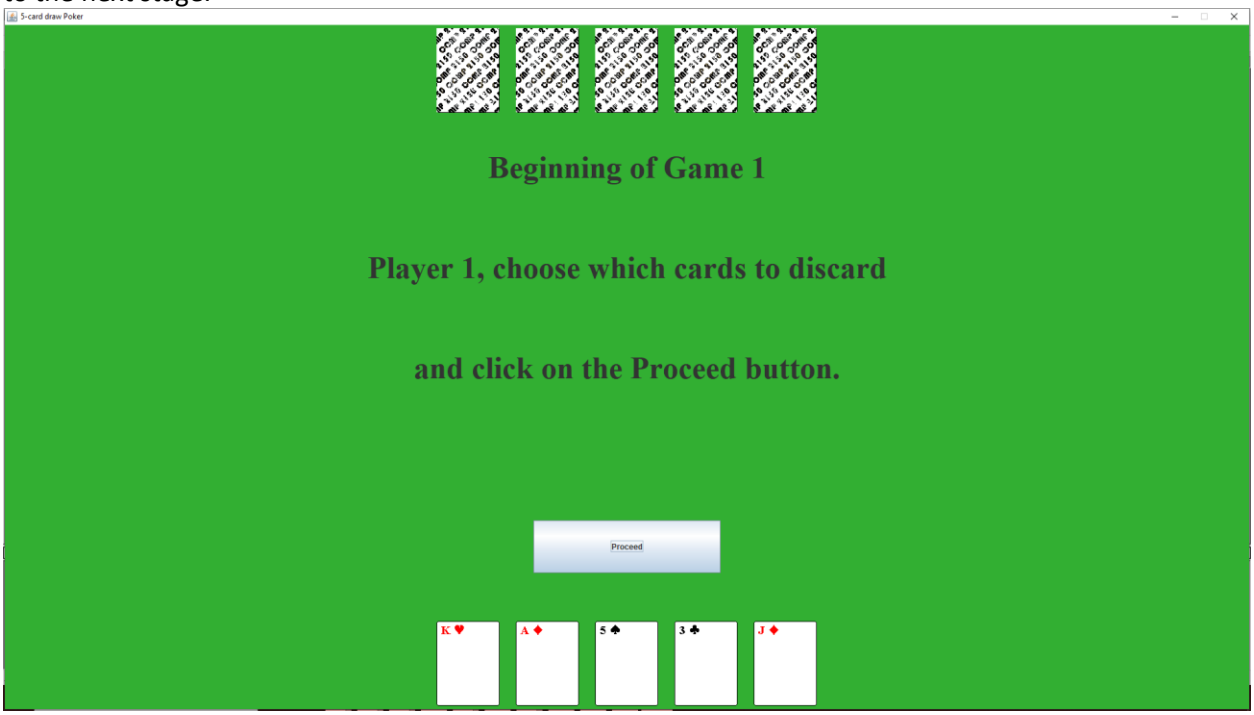
5) GameLogicable: You should define a class implementing this interface that will represent the game logic. The game logic is in charge of keeping track of everything about a game: the players, the deck of cards, the hands of both players, the game number, etc.

An instance of type GameLogicable is the only object that is sent to the GUI to start the game. The GUI will communicate with the GameLogicable object to display what is happening. Please read the comments in the file for more information and for descriptions of the expected behaviour of each required method.

The game should have 6 stages/states:

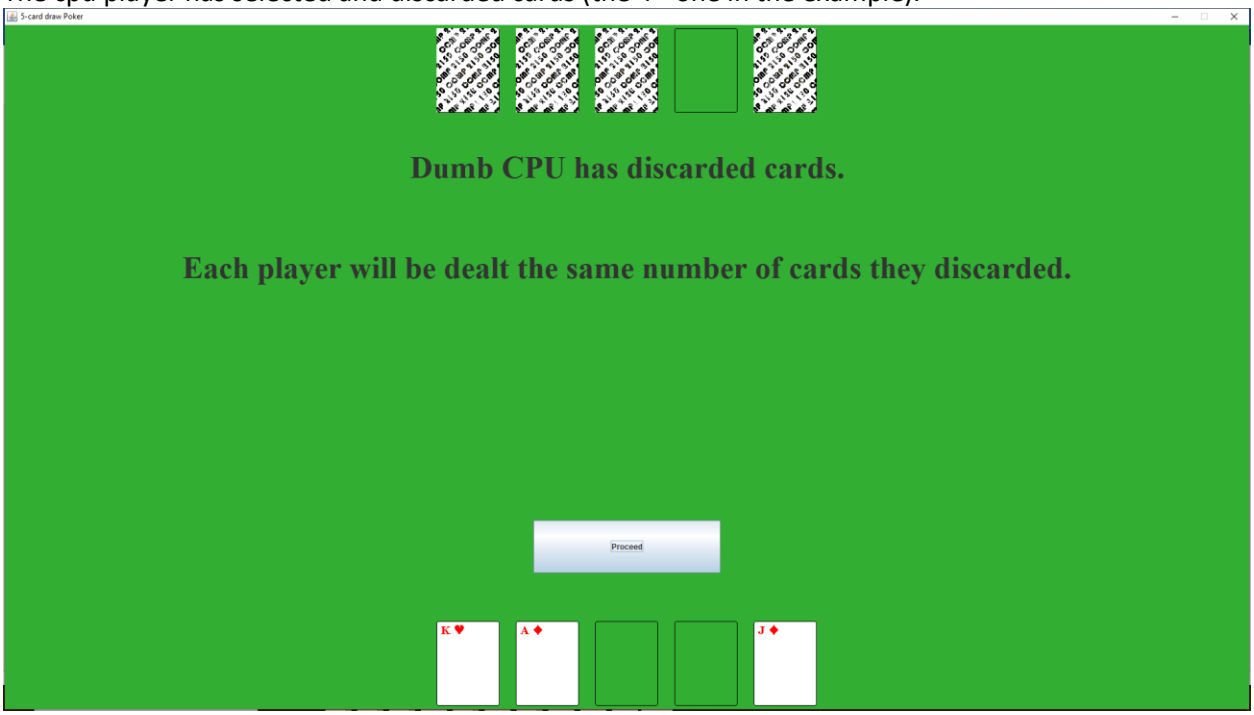
1. The beginning of the game. The GUI displays the game number (Game 1 at first). The human player (called Player 1 in the screenshots) is asked to select cards to discard. The player must select the cards to discard by clicking on them and then click on the Proceed button to proceed

to the next stage.

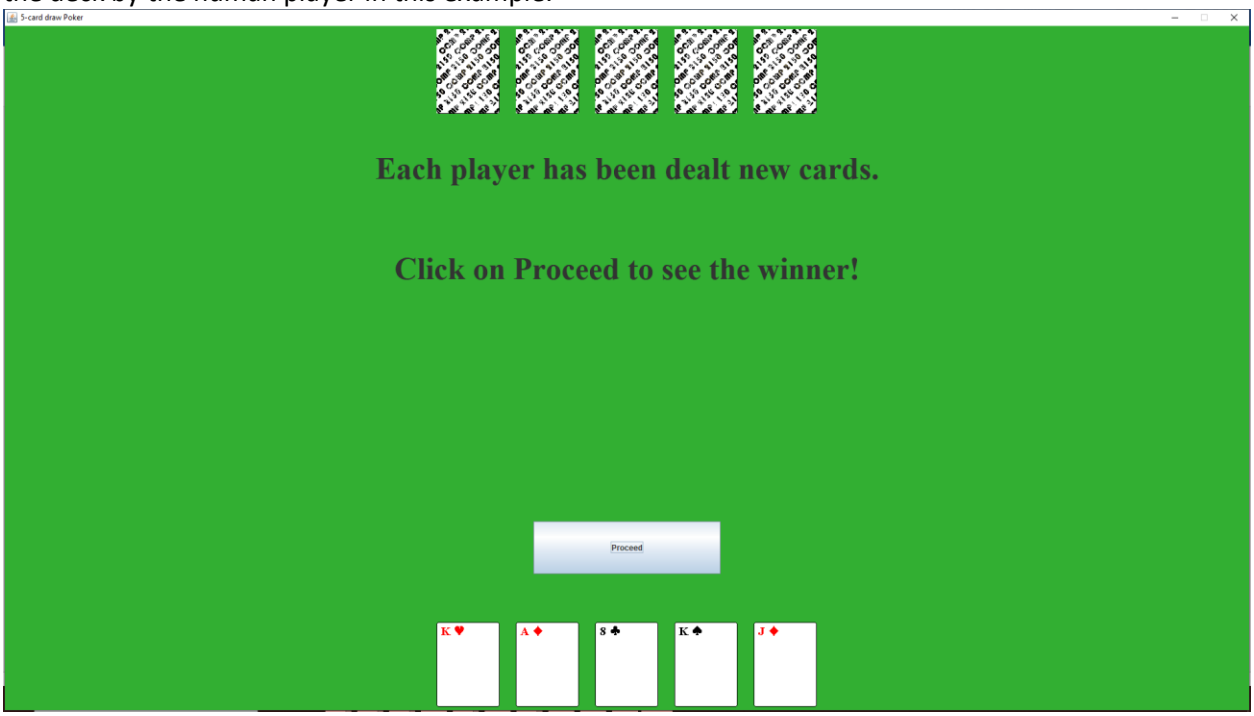


2. The selected cards (5 and 3 in this example) have been discarded, and the cpu player (called Dumb CPU here) is “thinking” about its move. Note that discarded cards need to be set to null in the Handable object to disappear from the GUI.

3. The cpu player has selected and discarded cards (the 4th one in the example).

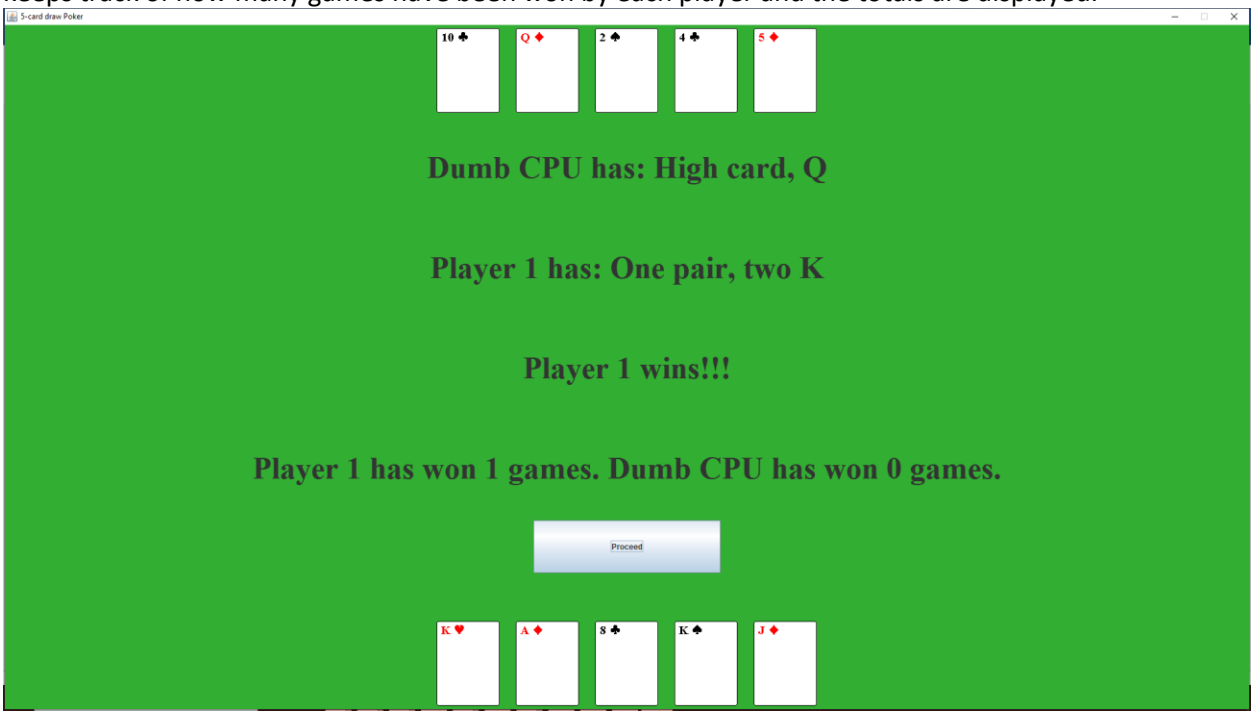


4. Cards have been drawn by both players to replenish their hands (8 and K have been drawn from the deck by the human player in this example.

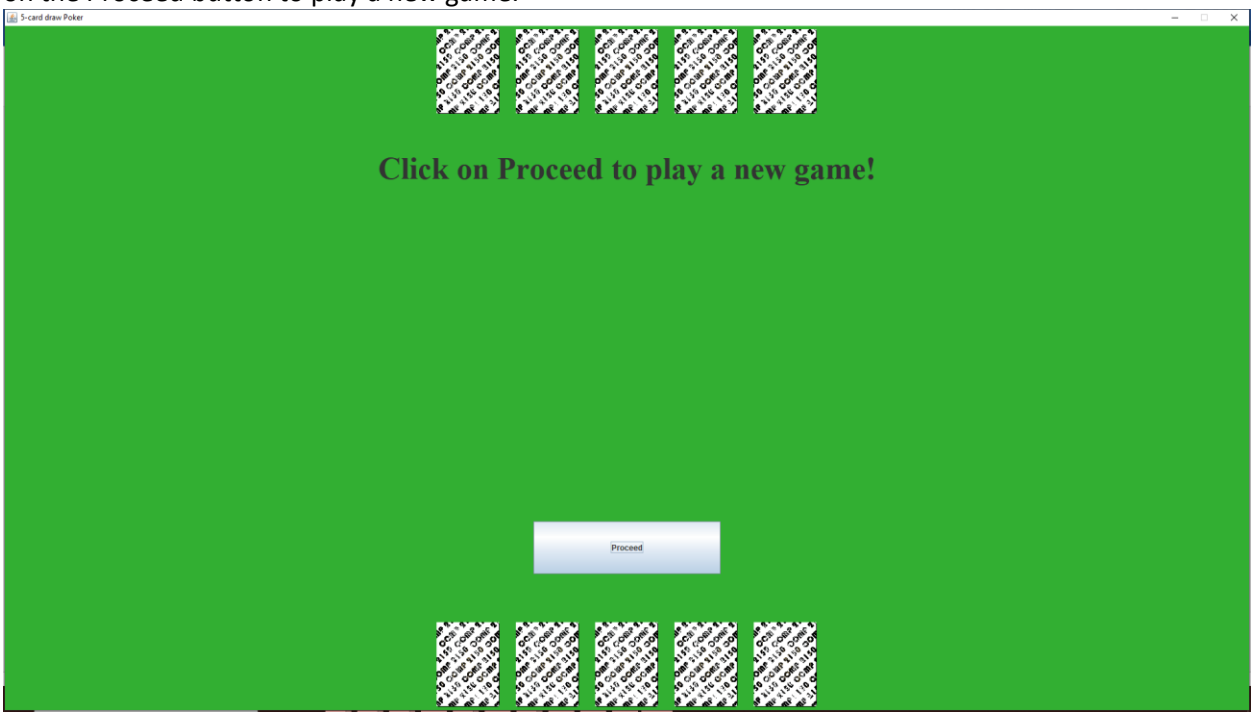


5. The showdown: every player shows their hand (i.e. the cpu player reveals its hand), each hand is evaluated and the winner is decided. In this example, the cpu player has nothing, so the highest card is displayed. The human player has one pair of kings, and wins this match. The game logic

keeps track of how many games have been won by each player and the totals are displayed.

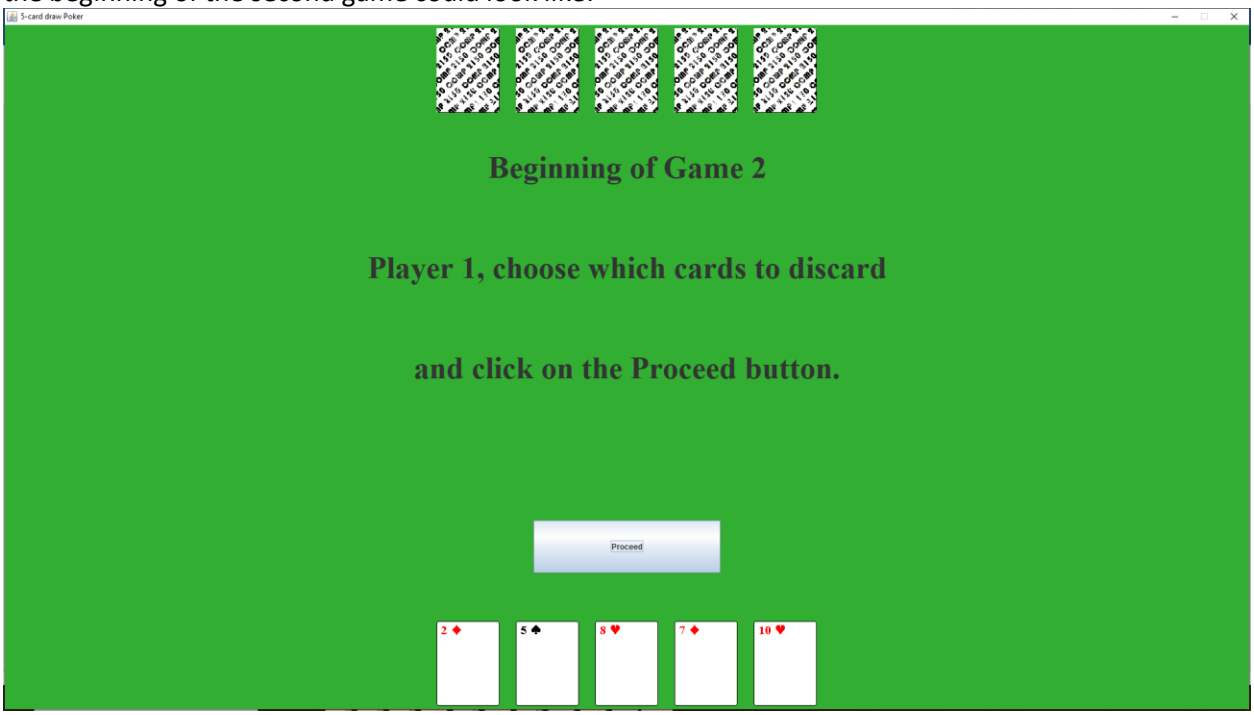


6. Hands and discarded cards are returned to the deck, the deck is shuffled, and new hands are drawn at this stage (but not shown, i.e. cards are facing down). The game asks the player to click on the Proceed button to play a new game.



After this, repeat steps 1-6 if the player clicks on Proceed. The game number should be updated and the counts of how many games have been won by each player will be saved. Here is what

the beginning of the second game could look like:



CPU player AI

You are required to produce 2 cpu player types (AIs): one that makes random choices and another one that is a bit more intelligent (note: it doesn't need to be on the level of a Poker World Champion… just to make smarter decisions than random).

Remember that the only decision that is made by a player is to select which cards to discard, so that is all you have to come up with. You should build (probably a hierarchy of) classes for the players. Your 'game logic' class will be in charge of creating the instance of the cpu version of choice (make sure to indicate this region clearly in your code so the markers can easily switch and try both cpus).

Important note: your cpu players are not allowed to cheat! They cannot peek at the human player's hand, they cannot create new cards, they cannot scan the whole deck to get a card that they need, etc.

JUnit tests

A JUnitTests.java file has been provided to you. One simple test is present in it. As you can see, it constructs instances of 'Card' and 'Hand' (the classes that implement the Cardable and Handable/TestableHand interfaces). That is why we recommend that you call these classes 'Card' and 'Hand': this way the JUnit test will work as is. The 'Card' constructor sets the value of the card and its suit (it is also necessary to create the same constructor for this test to work as is). An empty constructor for the 'Hand' object is used, and then the method required by 'TestableHand' is called to set the hand. The goal here is to test the result of your compareTo method on these hands.

You will be evaluated partly on which tests you provide: how interesting and complex they are. The markers will also add their own tests (with hands of various levels of complexity), and you will be evaluated on how many of these tests pass/fail.

Data Structures

For this assignment, you are allowed to use pre-built data structures. For example, you can use arrays and java.util.LinkedList.

Input and output:

There is no input and output in this assignment. All the results of your programming should be visible in the GUI.

Hand-in

Submit all your source code for all classes, including the updated JUnitTests.java file. Recall that compiled .class files cannot be graded and will not receive any marks. Since the main is in the provided 'PokerGame.java' file, you do not need to submit a readme file (everybody's code will be compiled and executed the same way).

You **MUST** submit all of your source code files in a zip file on UMLearn in the Assignment 3 folder.

Remember: the easier it is to mark your assignment, the more marks you are likely to get. **Do yourself a favour**.