Name: Paul Chafetz  
Date: 5/23/2019

**19.03 Elevens Lab Worksheet**

**Directions**: Make note of your responses to the following questions as you work through the activities and exercise in the lesson.

**Activity 6 Questions**

1. List all possible plays for the board 5 4 2 6 A J K 5 2  
   Both pairs of 5 and 6which equals 11.
2. If the deck is empty and the board has three cards left, must they be J, Q, and K? Why or why not?  
   Yes. Jacks, Queens, and Kings are all greater than 11 points, and thus cannot ever be paired with another card. Every other card would be eliminated in pairs, so the only way three cards can remain is if they are a J, Q, and K combo.
3. Does the game involve any strategy? That is, when more than one play is possible, does it matter which one is chosen? Briefly explain your answer.  
   Yes. It is very strategic in how you play the game. Given multiple options, if you choose poorly, the next dealt cards could not allow you to make any pairs, resulting in a loss. Careful planning (and luck!) is necessary to always have a usable play each turn.

**Activity 7 Questions**

1. What items would be necessary if you were playing a game of Elevens at your desk (not on the computer)? List the private instance variables needed for the ElevensBoard class.  
   It would need private instance variables for a deck of cards (array of all Card objects) and a list of cards on the board (array of Card objects in play).
2. Write an algorithm that describes the actions necessary to play the Elevens game.
3. Shuffle the deck
4. Deal 9 cards from deck to board
5. If a pair of Cards adds together to 11
   1. Remove the pair of Cards
   2. Replace the cards
6. If there is a J, Q, K triplet
   1. Remove the triplet
   2. Replace the cards
7. If there are 0 cards in the deck and board, the game is won
8. In the partially-implemented ElevensBoard.java file, does the class contain all the state and behavior necessary to play the game? Explain.   
   No. There seems to lack a method that chooses the cards that the user clicks on for removal. There is a method to check the selected cards, but not one to select them!
9. ElevensBoard.java contains three helper methods. These helper methods are private

because they are only called from the ElevensBoard class.

* 1. Where is the dealMyCards method called in ElevensBoard?  
     This method is called in the ElevensBoard constructor and in the newGame method.
  2. Which public methods should call the containsPairSum11 and containsJQK

methods?  
You need the isLegal method and anotherPlayIsPossible method to call those methods.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** |
| **cards** | J | 6 | null | 2 | null | null | A | 4 | null |
| **returned**  **list** | 0 | 1 | 3 | 6 | 7 |  |  |  |  |

* 1. Suppose that cards contains the elements shown below. Trace the execution of the cardIndexes method to determine what list will be returned. Complete the diagram below by filling in the elements of the returned list, and by showing how those values index cards. Note that the returned list may have less than nine elements.  
       
     For each non-null element, the returned list is the index of that element. For the sake of neatness, I put the list in each box rather than skipping the null spaces, but each index would be under its corresponding card index.
  2. Which one of the methods that you identified in question 4b above needs to call the cardIndexes method before calling the containsPairSum11 and containsJQK methods? Why?  
     The anotherPlayIsPossible method. This is because the method needs to know the indexes of the non-null cards (by calling cardIndexes) before it can determine whether there are J, Q, K triplets or pairs that sum to 11, as to avoid the null cards.

**Activity 8 Questions**

1. Discuss the similarities and differences between the games *Elevens*, *Thirteens*, and *Tens*.  
   All three games are similar in that they are all played on a board, they all use a deck of 52 cards, and that pairs of cards are added together to target sums in order to proceed with the game. They differ in that the number of cards on the board are different, the target sum is different, and that the specific combination of jacks, queens, and/or kings vary per game.
2. The instance variables for cards and deck are declared in the Board class. But it is the ElevensBoard class that "knows" the board size, and the ranks, suits, and point values of the cards in the deck. How do the Board instance variables get initialized with the ElevensBoard values? What is the exact mechanism?  
   Since the variables are in the superclass Board, the only way to pass them down to the ElevensBoard is by using the super() method for each parameter as follows:

super(BOARD\_SIZE, RANKS, SUITS, POINT\_VALUES);

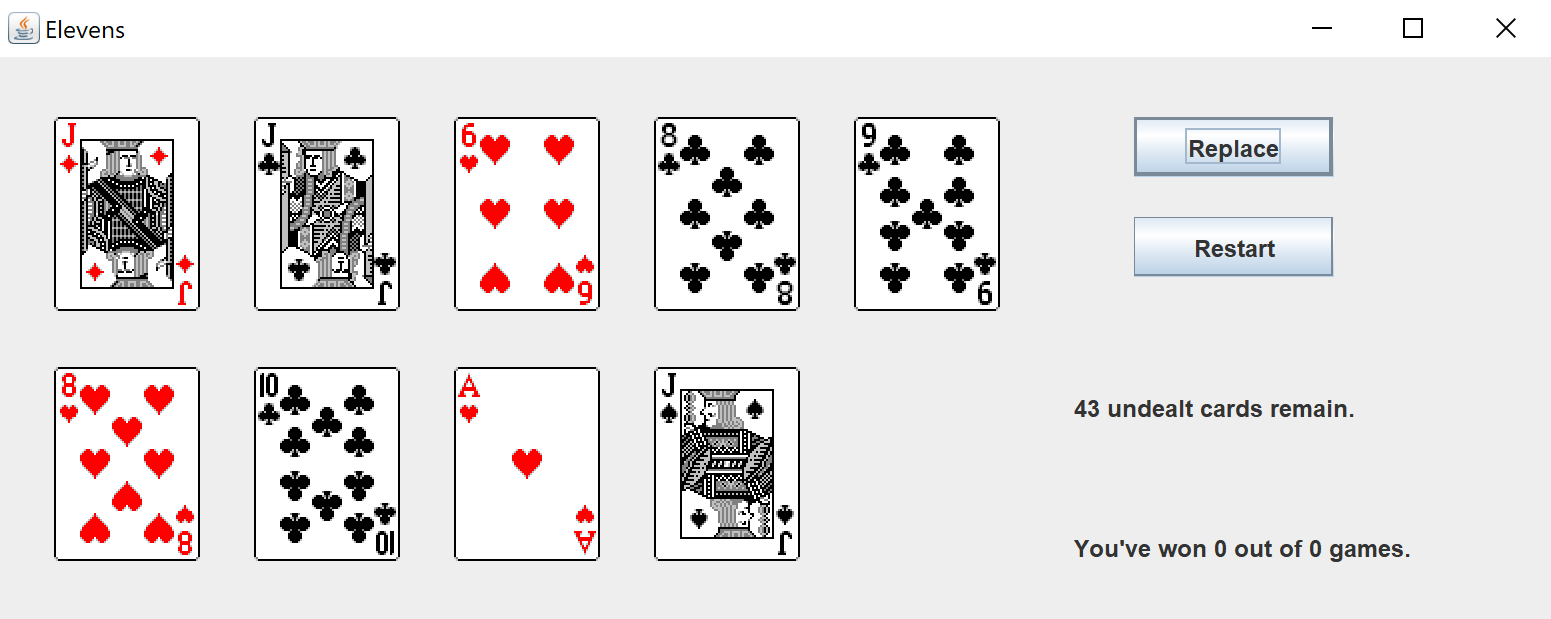
1. List the abstract methods in Board.java. These methods are implemented in ElevensBoard. Do they cover all the differences between *Elevens*, *Thirteens*, and *Tens* as discussed in question 1? Why or why not?  
   public abstract boolean isLegal and public abstract boolean anotherPlayIsPossible. They cover all the differences necessary between the games. These methods must be implemented differently for each game type, but it is understood that all the games share these characteristics, though they may differ somewhat. In each game board class, the methods must be modified for the intended function of each game type.

**Activity 9 Exercise Results**

1. After running the ElevensGUIRunner.java class, describe what you see and experience. Take a picture of the screen and paste it below, if you like, along with the description.

I see the 9 card images, the two buttons to replace and restart, the undealt cards, and my score.

Each card is able to be clicked. The methods all work exactly as designed when it comes down to

checking for triplets or sums of 11 from the possibilities. I won 1 out of 3 games (not shown).  


**Activity 9 Questions**

1. The size of the board is one of the differences between *Elevens* and *Thirteens*. Why is size not an abstract method?  
   Size is an instance variable and varies per board type. There is no point in creating an abstract method in Board if the subclass (ElevensBoard) already defines the size specific to its type.
2. Why are there no abstract methods dealing with the selection of the cards to be removed or replaced in the array cards?  
   All the board sizes/types have the same functionality of removing and replacing cards. Therefore, it can be directly implemented in the Board class without being an undefined abstract method.
3. Another way to create "IS-A" relationships is by implementing interfaces. Suppose that instead of creating an abstract Board class, we created the following Board interface, and had ElevensBoard implement it. Would this new scheme allow the Elevens GUI to call isLegal and anotherPlayIsPossible polymorphically? Would this alternate design work as well as the abstract Board class design? Why or why not?

public interface Board

{

boolean isLegal(List<Integer> selectedCards);

boolean anotherPlayIsPossible();

}

Yes, the abstract methods can still be called in a polymorphic design through an interface. After all, interfaces are only undefined abstract classes with no specific code under the hood. I jokingly call it the abstract class of abstract classes, which isn’t completely wrong! While this design would work, there would be an issue with the other non-abstract methods. All of those would need to be created in either a new abstract class or separately and individually in each game board class, which is a bad and inefficient process.