Name: Filip Kin  
Date: 2020-04-24

**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  
   The 5♠ and 6♣ make 11. The 5♣ and 6♣ also make 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?
   1. The deck and the board satisfy three relations before and after each play.
   2. The total number of face cards is divisble by 3.
   3. The number of J,Q, and K are all equal
   4. The total number of nonface cards is even.
   5. If the deck is empty and the board contains three cards, they must all be face cards. There have to be equal numbers of each face card, the three remaining must be JQK.
   6. If there are three face cards left, there must have been three previous plays of face cards. If the remaining cards are not JQK, then each face card rank wasn’t played the same number of times. This can’t happen.
   7. If there are two or one face cards left, we get a contradiction using the same reasoning as in the first case.
   8. If none of the three cards is a face card, there must have been a play of an odd number of nonface cards earlier in the game, as there are 40 nonface cards in all. This play would have been illegal.
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.

This game does not involve any strategy since none of the pairs of cards that add to eleven overlap, winning is pure luck based on which cards are drawn.

**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.  
     
   A deck of cards  
   BOARD\_SIZE, RANKS, SUITS, POINT\_VALUES, cards, deck
2. Write an algorithm that describes the actions necessary to play the Elevens game.  
     
   1. Create new game

2. Draw 9 cards. Or all the left if cards < 9.

3. Look for a set of Jack, Queen, and King. If it exists, replace the three cards.

4. Look for a set of two cards that adds to 11. If it exists, replace the two cards.

5. If nothing was replaced in the last turn, then the game is lost.

6. Repeat steps 2-6 until there are no cards left remaining in the deck or on the board.

7. Game over.

1. In the partially-implemented ElevensBoard.java file, does the class contain all the state and behavior necessary to play the game? Explain.   
     
   Yes
2. 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?  
       
     dealMyCards is called in the newGame() method and in the constructor when a new ElevensBoard is instantiated.
  2. Which public methods should call the containsPairSum11 and containsJQK

methods?

anotherPlayIsPossible(), isLegal()

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **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 |  |  |  |  | |

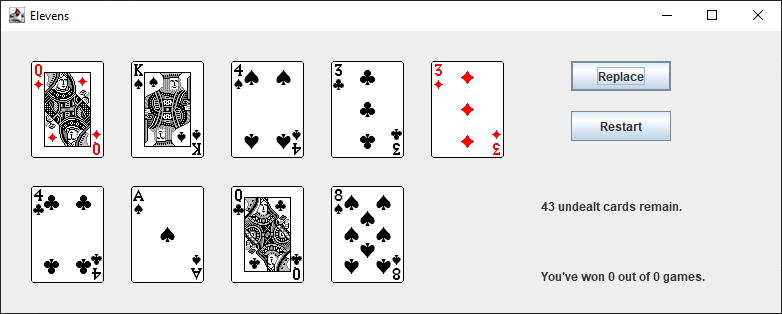
* 1. Suppose that cards contains the elements shown below. Trace the execution of the cardIndexes method to determine what list will be returned. Complete the diagram below by filling in the elements of the returned list, and by showing how those values index cards. Note that the returned list may have less than nine elements.
  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?  
       
     anotherPlayIsPossible()

**Activity 8 Questions**

1. Discuss the similarities and differences between the games *Elevens*, *Thirteens*, and *Tens*.  
     
   All these games involve a full deck of cards and a board. Some functions could be used in all three such as deal, deckSize, isEmpty. A few functions overlap but require different implementations, such as isLegal() and anotherPlayIsPossible(). Some functions are not usable in other games, such as containsJQK().
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?  
     
   The instance variable is initialized in the Board class. Inside constructor of ElevensBoard, the values are passed into the constructor of the superclass.
3. 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?  
     
   They cover all the differences because all of the methods that are exactly shareable between the card games are implemented in the Board class while the overlapping functions that require different implmentations (anotherPlayIsPossible and isLegal) are abstract, and thus implemented in the respective board game subclasses.

**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 and experience a GUI version of the game, and it has very basic graphics.



**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. Don’t need setters or getters in the Board class since the size variable is already defined in the subclass.
2. Why are there no abstract methods dealing with the selection of the cards to be removed or replaced in the array cards?  
     
   Removing and replacing cards is the same regardless of which game it is. It can be implemented in the Board class and doesn’t need to be abstract.
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();

}

isLegal() and anotherPlayIsPossible() would still be called polymorphically. This design can work but all the methods will have to be implemented separately for each card game board class.