**Activity 7, 8 and 9 Questions**

**Activity 7**

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.

BOARD\_SIZE, RANKS, SUITS, POINT\_VALUES, cards, deck

2. Write an algorithm that describes the actions necessary to play the Elevens game.

Setup board, loop through (player selecting cards, check if valid set, move to foundation, check if no more moves) print win or lose based on remaining cards

3. Now examine the partially implemented ElevensBoard.java file found in the **Activity7 Starter Code** directory. Does the ElevensBoard class contain all the state and behavior necessary to play the game?

No.

4. ElevensBoard.java contains three helper methods. These helper methods are private because they are only called from the ElevensBoard class.

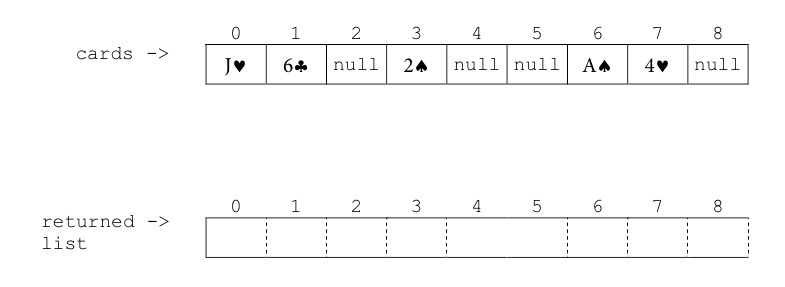
a. Where is the dealMyCards method called in ElevensBoard?

In newGame() and the constructor.

b. Which public methods should call the containsPairSum11 and containsJQK methods?

anotherPlayIsPossible(), isLegal().

c. It’s important to understand how the cardIndexes method works, and how the list that it returns is used. 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 9 elements.



0,1,3,6,7

d. Complete the following printCards method to print all of the elements of cards that are indexed by cIndexes.

public static printCards(ElevensBoard board) { List<Integer> cIndexes = board.cardIndexes();

for (Integer i : cIndexes)

System.out.print(cards[i]+” “);

}

e. 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? anotherMoveIsPossible() should call cardIndexes() because it will need to check all combinations.

**Activity 8**

1. Discuss the similarities and differences between *Elevens*, *Thirteens*, and *Tens*.

They all use similar boards and game styles.

1. As discussed previously, all of the instance variables 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?

ElevensBoard() calls the super class constructor with the correct values.

1. Now examine the files Board.java, and ElevensBoard.java, found in the **Activity8 Starter Code** directory. Identify the abstract methods in Board.java. See how 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?

Yes, all variations are covered.

**Activity 9**

1. The size of the board is one of the differences between *Elevens* and *Thirteens*. Why is size not an abstract method?
2. Why are there no abstract methods dealing with the selection of the cards to be removed or replaced in the array cards?
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();

}