Name:
Date:

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♠

Five of spades and 6 of cloves.

1. If the deck is empty and the board has three cards left, must they be J, Q, and K? Why or why not?

Yes, because the rules state that the final combinations must either be two cards adding up to 11 or three royal cards.

2. 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, because there can be multiple ways to combine cards to add up to 11.

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.

Space for 9 cards and a standard deck of 52 cards.

1. Write an algorithm that describes the actions necessary to play the Elevens game.
   1. Remove a pair of cards that adds up to 11 or remove three royal cards. Add new cards in place of the ones you have removed.
   2. If no option like this is avaliable, you have lost. Otherwise, repeat until the deck is empty.
2. In the partially-implemented `ElevensBoard.java` file, does the class contain all the state and behavior necessary to play the game? Explain.
   No, because some of the methods are incomplete and would need to be completed for the program to function properly.
2. `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 the class constructor and in the newGame method.

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

anotherPlayIsPossible and isLegal

	0	1	2	3	4	5	6	7	8
cards	J♥	6♣	null	2♠	null	null	A♠	4♥	null
returned list	**0**	1	2	3	4				

c. 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.

d. 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?

IsLegal, as these are necessary to ensure the program is working with a valid card play.

Activity 8 Questions

1. Discuss the similarities and differences between the games *Elevens*, *Thirteens*, and *Tens*.
These are all solitaries and involve pairing cards to add up to a certain number with the exception of certain face cards. The differences involve the specific excluded face cards and the value that the cards must add up to.
1. 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 board instance variables are initialized by a instantiation of a Deck object in the constructor of the ElevensBoard class.

2. 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?

   IsLegal and anotherPlayIsPossible are the two abstract methods implemented in ElevensBoard. They do cover all the differences, as the isLegal method is the most important in making sure the card combinations are correctly aligning to the rule-set.

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 a playing field for the Elevens card game.

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?
   Because it is handled by the pure size of an array of cards.

1. Why are there no abstract methods dealing with the selection of the cards to be removed or replaced in the array `cards`?
   The selection is general and is handled by a integer array that is made by specific games.
2. 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();
}
```

It would allow polymorphism but would not be as useful as the other methods such as size() would need to be implemented in each child class, which would become redundant very fast.