Elena Campell

Mr. Ettlin

APCSA, Period 3

25 February 2019

Questions for Project 6 Elevens

Activity 2

1. Explain in your own words the relationship between a deck and a card.

   The deck object is made out of multiple card objects. Each card object has a rank, suit, and value.

2. Consider the deck initialized with the statements below. How many cards does the deck contain?

   ```
   String[] ranks = {"jack", "queen", "king"};
   String[] suits = {"blue", "red"};
   int[] pointValues = {11, 12, 13};
   Deck d = new Deck(ranks, suits, pointValues);
   ```

   The deck contains six cards.

3. The game of Twenty-One is played with a deck of 52 cards. Ranks run from ace (highest) down to 2(lowest). Suits are spades, hearts, diamonds, and clubs as in many other games. A face card has point value 10; an ace has point value 11; point values for 2, …, 10 are 2, …, 10, respectively. Specify the contents of the ranks, suits, and pointValues arrays so that the statement

   Deck d = new Deck(ranks, suits, pointValues); initializes a deck for a Twenty-One game.

   Ranks would be : {“2”, “3” , “4”, “5”, “6”, “7”, “8”, “9”, “10”, “Jacks”, “Queens”, “Kings”, “Ace”}

   Suit would be: { “Spade”, “club”, “diamond”, “heart”}

   Value would be: {2,3,4,5,6,7,8,9,10,10,10,10,11}

4. Does the order of elements of the ranks, suits, and pointValues arrays matter?

No, the order does not matter because the constructor creates a card with each possible suit of each possible rank no matter the order.

Activity 3

1. Write a static method named flip that simulates a flip of a weighted coin by returning either "heads" or "tails" each time it is called. The coin is twice as likely to turn up heads as tails. Thus, flip should return "heads" about twice as often as it returns "tails."

```
Public static String flip() {
If(Math.random()> 0.33){
Return “head”;
} else{
Return “tails”;
}
}
```

2. Write a static method named arePermutations that, given two int arrays of the same length but with no duplicate elements, returns true if one array is a permutation of the other (i.e., the arrays differ only in how their contents are arranged). Otherwise, it should return false.

```
Public static boolean arePermutaitons (int[] a, int[] b) {
for(int i = 0; i< a.length; i++){
Boolean test = false;
For (int k = 0; k<b.length; k++){
If ( a[i] == b[k]{
```

```
Test = true;
}
}
If (test){
Return true;
}
} return false;
}
```

3. Suppose that the initial contents of the values array in Shuffler.java are {1, 2, 3,4}.For what sequence of random integers would the efficient selection shuffle change values to contain {4, 3, 2, 1}?

1, 2 , 2, 2

Activity 6:

1. List all possible plays for the board 5♠ 4♥ 2♦ 6♣ A♠ J♥ K♦ 5♣ 2♠

   5♠ & 6♣

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

   If the deck is empty, the last three cards must be J, Q, K to win, but it is not guaranteed that they will be those cards.

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.

It doesn't matter which play is chosen. If there are two plays at one time, no matter what order they are chosen, they will be filled in with the same cards.

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.

If you were playing Elevens at your desk, you would need a pile to draw from, 9 cards face up, and a pile to put matched cards in. The private instance varibale needed for the ElevensBoard class would be a deck.

2. Write an algorithm that describes the actions necessary to play the Elevens game
    a. Create new game
    b. Check to make sure there are at least cards remaining in the deck
        i. If yes, draw 9 cards. otherwise , draw the remainder of the deck
    c. Look for a jack, queen, king set. If there is one, select them and replace
    d. Look for a pair of 2 cards that adds up to 11. If there is one, select and replace\
    e. If c or d does not exist, game ends
    f. Repeat b-d until there are no cards left
3. Now examine the partially implemented ElevensBoard.java file found in the Activity 7 Starter Code directory. Does the ElevensBoard class contain all the state and behavior necessary to play the game?

    Yes, the ElevensBoard class contains all the state and behavior necessary to play the game.
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?

dealMyCards is called in the constructor, and in the newGame method.

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

They should be called in the methods anotherPlayIsPossible and 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.

indices	0	1	2	3	4	5	6	7	8
cards	J♥	6♣	null	2♠	null	null	A♠	4♥	null
return list	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 (int i = 0; i < cIndexes.size(); i++) {
System.out.print(board.cars[i].toString());
```

```
}
```

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?

The method anotherPlayIsPossible needs to call the callIdexes method before calling the containsPairSum11 and containsJQK methods.

Activity 8:

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

Both games require a full deck and a board. The games use similar functions, such as deal, deckSize, and isEmpty. Some methods overlaps across the three games, but have different implementations. There are some methods that are specific to each game.

2. 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?

The Board instance variables get initialized with the ELvensBoard values through a super-sub class relationship. The Board class is a super class while the ElevensBoard is a subclass of Board. The exact mechanism is a parent class relationship.

3. 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?

The abstract methods in the Board class are isLegal and anotherPlayisPossible. Yes, they cover all the differences between the three games because the methods that are shared are implemented in the Board class, but the abstract methods declared in Board are implemented differently in each of the sub classes.

Activity 9

1. The size of the board is one of the differences between *Elevens* and *Thirteens*. Why is size not an abstract method?

Size is not an abstract method because it is an instance variable, and it is already defined in the subclasses.

2. Why are there no abstract methods dealing with the selection of the cards to be removed or replaced in the array cards?

There are no abstract methods dealing with the selection of cards to be removed or replaced, because it is the same across each game, so the method can just be implemented in the Board class.

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();
```

```
}
```

If Board was an interface instead of an abstract class, isLegal() and anotherPlayIsPossible() would still be called polymorphically. This design would work, but all of the methods will have to be implemented separately for each card game board class.