##Activity 2

1. A deck and a card are separate, but a deck class creates an ArrayList of card objects.

2. 3

3. Create ranks, suits, and pointValues of length 52. Ranks contains 4 1's, 4 2's, 4 3's, ..., up to 4 Ace's. Suits contains 13 hearts, 13 spades, 13 diamonds, and 13 clubs. pointValues contains 4 1's, 4 2's, 4 3's, ..., 16 10's, and 4 11's.

4. Yes, because the nth index of all the arrays are part of one card.

##Activity 3

1)

```

public static string flip() {

if (Math.random() > 0.33) {

return "head";

}

else {

return "tails";

}

}

```

2)

```

public static boolean arePermutations(int[] a, int[] b){

for(int i = 0; i < a.length; i++){

boolean test1 = false;

for(int k = 0; k < b.length; k++){

if(a[i] == b[k]){

test1 = true;

}

}

if(test1){

return true;

}

}

return false;

}

```

3) 1, 2, 2, 2

##Activity 6

1) - 5♠ 6♣

- 5♣ 6♣

2) Yes because every time two cards are removed unless if it is a J, Q, and K. Thus, the amount of cards remaining on the board without there being a full J, Q, and K set must always be an even number, otherwise there is a full J, Q, and K.

3) This game does not involve any strategy since none of the pairs of cards that add to eleven overlap, thus it is pure luck based on which cards are drawn everytime one replaces a pair.

##Activity 7

1) A deck of cards

2)

```

1. Create new game

2. Check if there are at least 9 cards remaining in the deck

3. If so, draw 9 cards. Otherwise draw the rest of the cards in the deck.

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

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

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

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

8. Game is won.

```

3) Yes

4)

a. dealMyCards is called in the newGame() method and in the constructor when a new ElevensBoard is instantiated.

b. anotherPlayIsPossible(), isLegal()

c. 0, 1, 3, 6, 7

d.

```

for (Integer i : cIndexes) {

System.out.println(board.cards[i].toString());

}

```

e. anotherPlayIsPossible()

##Activity 8

1) These games all involve a full deck of cards and a board. Certain functions are use among the three as deal, deckSize, isEmpty, etc. However, a few functions overlap but require different implementations between the games, such as isLegal() and anotherPlayIsPossible(). Finally, there a few functions that exist purely in one of the games and is not shareable, such as containsJQK().

2) The instance variable is initialized in the Board class. Inside constructor of ElevensBoard, the values are passed into the constructor of the superclass.

3) 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 implementations (anotherPlayIsPossible and isLegal) are abstract, and thus implemented in the respective board game subclasses.

##Activity 9

1) Size is an instance variable. There is no need to create setters or getter methods in the Board class since the size variable is already defined in the subclass.

2) Because removing and replacing cards is uniform regardless of which game is being played. Thus it can be implemented in the Board class and does not need to be an abstract method.

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