Activity 2 questions:

1. Relationship between deck and card: aggregation. A deck “has” cards stored in the instance variable cards.
2. 6 cards
3. Ranks: [Ace, King, Queen, Jack, 10, 9,8,7,6,5,4,3,2]

Suits: [spades, hearts, diamonds, clubs]

Point values: [11, 10,10,10,9,8,7,6,5,4,3,2]

1. Yes. Cards objects are created using index values, so each card rank’s point value is from the same index value in the array pointValues.

Activity 3 questions:

1. Method for flip:

public static String flip() {

double r = Math.random();

if (r<=0.66) {return “heads”;}

else {return “false”;}

1. Method for arePermutations:

public static Boolean arePermutations(int[] first, int[] second) {

int size = 0;

for (int element: first) {

for (int element2:second) { if (element2==element) {count++;} }

}

If (size == first.length)

{return true;}

else {return false;}

1. 3,2,1,0

Activity 5 Questions:

1. Possible error: might have used the != instead of the == Boolean comparison operator for isEmpty()
2. Possible error: didn’t use the .add() method to add created card to the ArrayList deck
3. Possible error: shuffle doesn’t reassign the “shuffled” deck to the instance variable deck
4. Possible error: off-by-one: the programmer made an arrayList one space too large and the first card that was dealt was the null space at the end of the arraylist

Activity 6 Questions:

1. 5 and 6, 6 and 5
2. Yes. Since they only disappear in threes, it’s impossible for them to ever be present in only pairs if they are the only 3 remaining.
3. No. It all depends on which cards are dealt next, and since you don’t know what they are, you can’t do anything to change it.

Activity 7 Questions:

1. What items are necessary?

Table, MatchChecker, Deck

1. Actions necessary:

Deal 9 cards, remove from deck

While cards can be matched:

If cards selected add to 11 or are face cards, remove from table (delete)

Replace with 2 or 3 cards from deck

1. Yes, it contains all the state and behavior necessary
2. Helper methods:
   1. Dealmycards() is called in the constructor and in newGame()
   2. anotherPlayIsPossible() and isLegal() should call containsJQK() and pairSum11()
   3. Values: 0,1,3,6,7
   4. For (Card card : cIndexes) {System.out.println(card.toString());
   5. isLegal() needs to call cardIndexes to determine if the cards on the board can be replaced

Activity 8

1. Similarities: want to get rid of all the cards, add up point values to certain number

Differences: how to get rid of cards: elevens can remove JQK triples, thirteens-kings are removed singly, tens-quartets of KQJ10

1. Call the superclass Board constructor and pass in variables to initialize them
2. Yes. They all share the commonality of having a deck and a board that needs to hold current cards. The only differences in their implementations are in determining the legality of the next move, which is specified by the fact that anotherPlayIsPossible() and isLegal() are both abstract methods.

Activity 9

1. Size doesn’t need to be an abstract method. Though the size is different, thirteens and elevens both have an array that stores the size. It is an accessor method that doesn’t require different implementation.
2. Again, the implementation isn’t required to be different. Their only function is to select and remove cards from the Array, which can be implemented the exact same way regardless of the rules of the game.
3. Using an interface would allow us to do the same thing as an abstract class. Yes, it could be used polymorphically, but the problem here lies in its lack of efficiency. Making Board an interface would require both the ElevensBoard and the ThirteensBoard to implement all of its methods, even though some of the methods are implemented the exact same way and would only need to be written once had an abstract class been used.