Activity 2 questions:

Deck has a dependency on card. Meaning, we cannot use a deck object without having it full of card objects.

This deck contains 6 cards.

String [] ranks = {“Two”, “Three”, “Four”, “Five”, “Six”, “Seven” , “Eight” , “Nine”, “Ten”, “Jack”, “Queen”, “King”, “Ace”}

String[] suits = {“Diamonds”, “Hearts”, “Spades”, “Clubs”}

Int [] values = {2,3,4,5,6,7,8,9,10,11}

No, because it will create all the cards regardless of the order you put the elements in.

Activity 3:

1. public static String flip()

{

Random randnum = new Random();

int randomInt = randnum.nextInt(4);

if(randomInt <=2)

{

return “Heads”;

}

else

{

return “Tails”;

}

}

2) public static boolean arePermutations(int [] values1, int[] values2)

{

int count = 0;

int total = values1.length;

if( values1.length == values2.length)

{

for( int i =0; i <values1.length; i++)

{

if(values1[i] != values2[i])

{

count ++;

}

}

}

if (count == total)

{

return true;

}

else

{

return false;

}

}

3) 0,1,1

SKIPPED ACTIVITY 5

Activity 6 questions:

6 clubs + 5 spades, 6 clubs + 5 clubs

Yes, because Queen , Jack, King can only be cleared off the board together.

I don’t think this game takes any strategy. In comparison to a game like chess which I have extensive knowledge about. The solutions you chose to play have minimal effect on your chances of winning. Most plays are forced. (Example: Queen King Jack, Ace 10)

Activity 7 questions:

Array of type card, array of point value, array of ranks, array of suits, board size.

Cards will randomly be chosen from the deck to be the initial cards. The game will not start unless there is one combination of queen, jack, king or two cards that add up to eleven. In the case that neither is present, the deck will reshuffle until that condition is met. When the cards that meet those conditions are selected together, those cards are removed from the deck and an equal amount of random cards take their place. If at any point past this, there are no combinations of queen, jack, king or two cards that add up to eleven, you lose. This continues until there are no cards left in play (deck or on board).

No, the starter code does not have completed methods to check for two cards that add up to 11 or to see if a board contains jack, queen, king. Without this, you cannot determine if you have lost or not.

A) dealMyCards is called in the newGame() method.

B) anotherPlayIsPossible(), isLegal()

C) {0, 1, 3, 6, 7}

D) for(int k = 0; k < cIndexes.length; k++)

{

System.out.println(cards[k].toString());

}

E) isLegal() because it needs to know which cards are in play. If for some reason it were to try to use a null spot before calling cardIndexes it would make an illegal move.

Activity 8:

1. The biggest similar between all 3 games is that they inherit from the Board class. Another similarity is the methods they share (ex. isLegal() anotherPlayisPossible()). The differences are the fact that each game has different rules. So while they have the same methods, the code for each method is different based on what game it is.
2. “Super” keyword which calls the constructor of the Board class.
3. Abstract methods: isLegal(), anotherPlayisPossible()

Yes, because the method names are general enough where they can cover all games by having different code for each game in those methods.

Activity 9:

1. Array cards is an instance variable of the abstract class. So you call the .size() method on that.
2. Because the class Deck that the array Cards is stored in is the same across the board. No need for abstract methods.
3. Using an interface instead of the abstract board class would work just as well. It would allow the Elevens GUI to call isLegal() and anotherPlayisPossible() polymorphically. The reason I think it would work just as well is because the functionality of the program would be unchanged.