**Activity 2**

1. Composition; a deck “owns” cards and cannot exist independently without them
2. 6
3. String[] ranks = {“two”, “three”, “four”, “five”, “six”, “seven”, “eight”, “nine”, “ten”, “jack”, “queen”, “king”, “ace};

String[] suits = {“spades”, “hearts”, “diamonds”, “clubs”};

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

1. The order of the elements in ranks must match the order of elements in pointValues, but other than that it can be in any order

**Activity 3**

1. import java.util.Random;

public static String flip()

{

int[] side = {1, 1, 2}

random rand = new Random();

for (int i = 0; i < 3; i++)

{

int j = rand.nextInt(3);

if (side[j] == 1)

{

return “heads”;

}

else

{

return “tails”;

}

}

}

1. public static arePermutations(int[] array1, int[] array2)

{

for (int i = 0; i < array1.length; i++)

{

for (int j = 0; j < array2.length; j++)

{

if (array1[i] != array2[j];

{

return false;

}

}

}

return true;

}

1. 1, 3

**Activity 5**

1. Buggy 1

Method: public Boolean isEmpty()

Possible Code Error: method could return if size!= 0 instead of size == 0, which would return false if the deck was not empty

Buggy 2

Method: public void shuffle() / public int size()

Possible Code Error: method could have forgotten to reset size of deck to represent entire new deck after shuffling, so that public int size() returns wrong size

Buggy 3

Method: public void shuffle()

Possible Code Error: the for loop could have had k <= 0 which would never be true, so then the loop would never fun and the deck would never be shuffled and the 2 decks would have the same cards

Buggy 4

Method: public void shuffle()

Possible Code Error: shuffle() method doesn’t reset size of deck, so deck size remains as 0 and every time the player tries to deal a card the deal() method returns null

**Activity 6**

1. 5 spades, 6 clubs

5 clubs, 6 clubs

1. Yes, because every card numbered ace – 10 can be paired and removed from the board, so if there is an odd number of cards left it cannot be any of those cards. The rest of the deck can only be removed in trios of jacks, queens, and kings, so if there are three cards left it can only be one jack, one queen, and one king.
2. There’s no strategy, because the cards that replace the ones you remove are random, so you cannot know which card pair you choose is best.

**Activity 7**

1. Board, undealt deck, cards
2. While number of cards in undealt deck < 0

If board doesn’t have 9 cards

Deal necessary number of cards from undealt deck to get 9 cards on board

Check if any cards can be removed

If no, print “You Lose”, end game

If board has 9 cards

If sum of point values of cards == 9

Remove cards from boards, adjust size of undealt deck

Else if there is one jack, one king, one queen

Remove cards from boards, adjust size of undealt deck

1. ElevensBoard needs to depend on cards and deck
   1. It is called at the beginning of the game to deal cards to the board to start the game
   2. Public boolean isLegal(List<Integer> selectedCards) and public boolean anotherPlayIsPossible()
   3. 0 1 2 3 4 5 6 7 8

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 3 | 6 | 7 | Null | Null | Null | null |

* 1. public static printCards(ElevensBoard board)

{

List<Integer> cIndexes = board.cardIndexes();

For (int i = 0; i < cIndexes.size(); i++)

{

While (cIndexes.get(i) != null)

{

System.out.println(cIndexes.get(i).toString());

}

}

}

* 1. public Boolean anotherPlayIsPossible() because this method needs to know all the valid cards on the board in order to determine if another play is possible by using containsPairSum11 or containsJQK

**Activity 8**

1. Similarities: Elevens, Tens, Thirteens all have similar states – they need a board, a deck, and cards on the board. They all have similar methods – they need to deal cards, select and remove cards, replace remove cards, determine if another play is possible, check if the selected cards are legal, check if a player has won, etc.

Differences: They have different forms of the same board – Elevens needs 9 cards on the board, Ten needs 13, Thirteen needs 10. They have different rules for making a play and winning, so the methods for checking for another play, checking if the selected cards are legal, etc. are specific to their own game.

1. Yes, because Board contains all of the states and methods that the three games have in common and can be shared while ElevensBoard has all the specific states and methods that are only found in that game. However, Board also needs to depend on a Card and Deck class.

**Activity 9**

1. Because it returns cards.length() which can be used in all the games, and instance variable cards will be different for each game
2. Because removing/replacing cards at certain positions works the same way for all the games, regardless of different methods to determine if the cards can be removed or not, so these methods can be implemented in the Board class
3. Yes, the two methods would still be called polymorphically because with the interface the method implementation in each specific board object would be used. The interface would not be as efficient as the abstract Board superclass because similar states and methods that could be handled by one class would have to be repeated for each specific board. It makes more sense to use inheritance because all the different board objects *are* boards rather than just implementing board features.