1. Activity 1:
   1. No Questions.
2. Activity 2:
   1. A deck is associated with a card.
   2. 6
   3. Ranks:
      1. Ace
      2. Two
      3. Three
      4. Four
      5. Five
      6. Six
      7. Seven
      8. Eight
      9. Nine
      10. Ten
      11. Jack
      12. Queen
      13. King
   4. Suits:
      1. Clubs
      2. Diamonds
      3. Spades
      4. Hearts
   5. Point Values
      1. 1
      2. 2
      3. 3
      4. 4
      5. 5
      6. 6
      7. 7
      8. 8
      9. 9
      10. 10
   6. The order does not matter, as long as all of the elements get added to the deck.
3. Activity 3:
   1. import java.util.Random;

public static String flip()

{

String[] coin = new String[3];

coin[0] = “Heads”;

coin[1] = “Heads”;

coin[2] = “Tails”;

String value;

Random gen = new Random();

int value = gen.nextInt(1);

return coin[value];

}

* 1. public static boolean arePermutations(int[] a1, int[] a2)

{

for (int i = 0; i < a1.length; i++)

{

if (a1[i] != a1[i])

{

return true;

}

}

}

* 1. 3, 2

1. No Questions.
2. Activity 5:
   1. Buggy1: Constructor or Method (write method name):
      1. testEmpty
   2. Describe a Possible Code Error:
      1. The assertion error occurs when what is returned, is false. Because the deck is not empty, it will return false.
   3. Buggy2: Constructor or Method (write method name):
      1. Test1CardDeck
   4. Describe a Possible Code Error:
      1. A card is created, but never added to the deck.
   5. Buggy3: Constructor or Method (write method name):
      1. testShuffle
   6. Describe a Possible Code Error:
      1. The two cards are not equal, however a not symbol (!) was used, making the cards appear to be equal.
   7. Buggy4: Constructor or Method (write method name):
      1. testOneCard
   8. Describe a Possible Code Error:
      1. The card is not null, however the assertion returned false because a not symbol (!) was not used.
3. Activity 6:
   1. 1. A♠ J♥
      2. A♠ K♦
      3. 6♣5♣
      4. 6♣5♠
   2. They do not need to be J, Q, and K. If there was an odd amount of J, Q, and K combinations prior to the last three cards, there is potential for it to be a pair and another card, or no pair at all. There is still chance for the last three cards to be J, Q, and K however.
   3. There is very little strategy involved. The dealt cards should be random, causing the only strategy to be the verification of which cards have already been removed.
4. Activity 7:
   * 1. Cards
     2. Deck
     3. Board Size
     4. Card Ranks
     5. Card Suits
     6. Card Point Values
   1. 1. Deck instantiated
      2. 52 cards created in deck using all the suits, ranks, and point values
      3. Deck size is updated
      4. Deck is shuffled
      5. Deck size is updated
      6. Cards are dealt
      7. Deck size is updated
      8. Dealt card’s size is updated
      9. Two or three cards are selected by a user, if they are a valid pair or triplet, they are removed from the dealt cards
      10. Two or three cards (dependent upon the case) are then dealt in place of the previous cards.
      11. Deck size is updated
      12. Dealt card’s size is updated
      13. If the deck’s size is zero, and the dealt card’s size is zero the player wins
   2. All of these parts are either implemented, or are soon to be implemented.
   3. dealMyCards is called in the newGame method.
      1. anotherPlayIsPossible
      2. isLegal

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| J♥ | 6♣ | 2♠ | A♠ | 4♥ |

* 1. public static printCards(ElevensBoard board)

{

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

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

{

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

}

}

* 1. 1. anotherPlayIsPossible
     2. isLegal

1. Activity 8:

|  |  |
| --- | --- |
| **Similarities:** | **Differences:** |
| * Instance variables: deck of cards, cards, board * Methods: deal, remove and replace selected cards, check for win, check if card follows rules of the game, check for other legal selections available | * The game rules * Valid cards for removal * Board size |

* 1. The mechanism used is called abstraction.
  2. 1. Abstract methods: isLegal, anotherPlayIsPossible
     2. The abstract methods cover almost all the differences between the game types. It adjusts to what the game rules are and also what the different games’ rules are. The methods do not cover the difference in the board sizes. This is because the board size is a private instance variable.

1. Activity 9:
   1. Size is not an abstract method because there is code inside the body.
   2. There are no abstract methods dealing with the selection of the cards to be removed or replaced because those methods have code inside the body.
   3. This would alternated the design work and the class design because those methods would no longer need to be built inside of other classes.
2. Activity 10:
   1. No Questions.