Max Kellezi AP CSA
Mr. Milonovich

Elevens Lab

Activity 2

1. A deck object has an ArrayList instance variable named cards, which contains card objects.
2. This deck contains 6 cards. It contains two sets of jacks, queens, and kings. One set is red, while the other is blue.
3. The ranks array contain all card types from the ace down to the two, as shown here: {“Ace”, “King”, “Queen”, “Jack”, “10”, “9”, “8”, “7”, “6”, “5”, “4”, “3”, “2”}. The suits array contains the four different usual suits of a typical deck of cards, as shown here: {“Clubs”, “Spades”, “Hearts”, “Diamonds”}. The pointValues array stores the value of each card type in ranks, shown here: {11, 10, 10, 10, 10, 9, 8, 7, 6, 5, 4, 3, 2}.
4. The order of suits does not matter. The order of ranks and pointValues also won’t matter as long as they are shuffled in the same way so that they correspond. Otherwise, the pointValues of cards of a certain rank would be incorrect.

Activity 3

1.

```
public static String flip()
{
        int random = (int)(Math.random * 3);
        if(random <= 1)
        {
                return “heads”;
        } else {
                return “tails”;
        }
}
```

2.

```
public static boolean arePermutations(int[] arr1, int[] arr2)
{
        for(int i = 0; i < arr1.length; i++)
        {
                boolean isFound = false;
                for(int j = 0; j < arr2.length; j++)
                {
```

```
                if(arr1[i] == arr2[j])
                {
                        isFound = true;
                        Break;
                }
        }
        if(!isFound)
        {
                return false;
        }
    }
    return true;
}
```

3. The first random integer would be 0. This would switch the 1 and the 4, giving us the array {4, 2, 3, 1}. The next random integer would be 1. This would switch the 2 and 3 three, giving us the array {4, 3, 2, 1}. The last random integer would also have to be 1, which would switch the 3 with itself, giving us the shuffled array {4, 3, 2, 1}.

Activity 6

1. The user can remove the 6♣ and the 5♣. The user can also remove the 5♠ and the 6♣.
2. Yes, the last remaining cards must be a J, Q, and a K. This occurs because the deck has 52 total cards, and the user can only remove two non-face cards at a time. This removes 40 cards from the game. The remaining 9 cards taken out of the game will also be face cards, so the last 3 cards remaining would also have to be face cards.
3. Although there may be some strategy over the best move to play (based on how many pairings have already been used, how many face cards are on the table, etc.), the game ultimately is not strategic and relies on luck as the next cards drawn will always be randomized.

Activity 7

1. To play a game of Elevens on a desk, you would need: a deck of 52 playing cards, a pile to draw from, 9 initial cards on the table, and a discard pile.

2.
```
    Initialize the draw pile to a normal deck of 52 cards.
    Deal 9 cards and put them on the table
    While the draw pile and table are not empty or no cards on the table add to 11,
        If two cards selected add to 11, remove them to the discard pile, deal 2
        cards to the table.
        If three different face cards are selected, remove them to the discard pile
        and deal 3 cards to the table.
    If the draw pile and the table is empty
        You win
    Otherwise
        You lose
```

3. Yes, the ElevensBoard class contains the necessary states and behaviors needed to play the game. There is additional, although unnecessary functionality that can also be implemented, like a discard pile, though it does not impact how the game actually runs.

4.

a. The dealMyCards method appears in the newGame() method.
b. The containsPairSum11 and containsJQK methods should be used in the isLegal() method and the anotherPlayIsPossible() method.
c. 0, 1, 3, 6, 7. Because the returned array is a list, the indexes 5-8 do not exist.
d.

```
public static void printCards(ElevensBoard board)
{
        List<Integer> cIndexes = board.cardIndexes();
        for(Integer i : cIndexes)
        {
                System.out.println(cards[i].toString())
        }
}
```

e. anotherPlayIsPossible() needs to call cardIndexes before calling the containsPairSum11 and containsJQK methods because this method, unlike isLegal(), searches all card on the board to find another potential move. To do this without comparing cards to null spaces, we need cardIndexes to find where all non-null cards are.

Activity 8

1. Elevens, thirteens, and tens are all very similar games. To start, they are all singular, and use a standard deck of cards. They all have a board onto which the cards are dealt, although their sizes vary. They also remove cards from this board by having pairs add up to a specific sum, where they are replaced by remaining cards in the deck. They all also have alternate ways of removing cards, typically relating to having an amount of face cards or cards of the same rank on the board. None of these alternate methods are the same.

2. The board instance variables get initialized with the ElevensBoard values by calling the super constructor in ElevensBoard's constructor. This would be done by calling super([instance variables of board here]); in the constructor. This accesses the constructor of board from the ElevensBoard constructor.
3. Although all the abstract methods of board are there, they do not cover the differences between the games because some behaviors, like isLegal() and anotherPlayIsPossible() rely on game-specific rules, so their methods will need to be overridden in their own respective classes.

Activity 9

1. Size is not an abstract method because it does not change behavior regardless of the subclass it's being used in; it's behavior is the same regardless of the subclass calling it, so it does not need to be abstract.
2. Selecting and removing cards from the table is the same regardless of the game being played--it does not vary depending on the type of game being played. Because of this, there is no reason for the code to be abstract and can be implemented solely in the Board superclass.
3. Interfaces contain methods that can never be instantiated. However, the subclasses of board can still declare it, which we see in the current ElevensBoard class. Since the isLegal and anotherPlayIsPossible methods are instantiated in the subclasses of board, this new scheme would allow the Elevens GUI to call both of the methods polymorphically. However, this alternate design would not work as well as the abstract Board class because we would need to have separate instantiations for some variables and methods for each subclass, even though they still function the same.