Nic Mills

Activity 2:

1. A deck *has* a card, however a card can exist without a deck.
2. 6 cards
3. See below:
   1. String[] ranks = { “Two”, “Three”, “Four”, “Five”, “Six”, “Seven”, “Eight”, “Nine”,“Ten”,“Jack”,“Queen”,“King” “Ace” }
   2. String[] suits = { “Diamonds”, “Clubs”, “Spades”, “Hearts” }
   3. Int[] pointValues = { 2,3,4,5,6,7,8,9,10,10,10,10,11}
4. If a standard for loop is used and the century variable is used to index the position of the values in ranks, suits, and pointValues, then the positioning of the elements in ranks, and pointValues *do* matter because if they aren’t in order, the ranks will get wrong pointvalues.

Activity 3:

1. Code Below: (Assume random is imported and a random object created)

Public static String flip()

{

If( generator.nextInt(3) == 1 ||

Generator.nextInt(3) == 2 )

{

Return “Heads”;

}

Else

{

Return “Tails”;

}

}

1. Public static Boolean arePermutations( int[] array1, int[] array 2)

{

For( int outer = 0; outer < array1.length; outer++ )

{

Int found = 0;

For( int inner = 0; inner < array2.length; inner++ )

{

If( array1[outer] == array2[inner] )

{

Found++;

}

}

If( found == 0 )

{

{

Return false;

}

}

Return true;

}

1. 3412

Activity 4 N/A

Activity 5 (DO BUGGY ERRORS AT HOME)

Activity 6

1. 5 and 6 (6 and 5)
2. Because if a pair is removed, there would be no play possible becuase no card alone can equal eleven
3. Yes, although the game is highly based on and reliant on “luck”, the strategy/ playstyle that keeps the most possible pairs of cards available at a time is the best, because this allows for multiple turns for random card selection to select viable pairs of cards to play

Activity 7

1. A deck of 52 cards (includes ranks, suits, and pointvalues as they are included in each card)
   1. Shuffle
   2. Deal
   3. Determine if game over
   4. Determine valid moves
   5. Select cards & make move
   6. Repeat steps 2-5 until game over
2. If all the methods were completed, then besides the GUI, ElevensBoard.java has all that it would need to play a game of *Elevens*, however it would not be easily adapted to play sevens, or thirteens for example
3. ElevensBoard.java contains three helper methods. These helper methods are private because they are only called from the ElevensBoard class
   1. In the constructor of ElevensBoard
   2. isLegal(), anotherPlayIsPossible()
   3. {2,4,5,8}
   4. String str = "";

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

{

Str += i + “: “+cards.get(i)+ “\n”;

}

System.out.println();

* 1. isLegal(), needs to call cardIndexes because it needs to get the card’s values before checking whether it is a legal play or not

Activity 8

1. All of the games need for the cards to combine to a number to qualify for a valid move. All those games inherit basic rules from a numbers class, the only thing that differs is the required number.
2. The board instance variables get initialized based on the given daughter class that extend board class. For example, in the ThirteensBoard class, the board size is initialized to thirteen
3. The only methods that are abstract in board are the isLegal() method, and the anotherPlayIsPossible() method. These methods are then defined in the daughter class, specifically *for* the specificities of the daughter class. That way, the code is kept simple and concise.

Activity 9

1. Size is not an abstract method because it fits the requirements of any daughter class- every daughter class has a “size”.
2. The methods that deal with the selection of cards, dealing them aren’t abstract because the selection of cards, and the way to deal them are the same across every daughter class
3. Implementing an interface would not work because every constructor of all the board classes would have to be changed to include redundant instance variables that could be easily defined once in the parent class.