# Elevens Lab Grading Sheet

Exercises

A1:

1. Complete the implementation of the provided Card class. You will be required to complete:
   1. a constructor that takes two String parameters that represent the card’s rank and suit, and an int parameter that represents the point value of the card;
   2. accessor methods for the card’s rank, suit, and point value;
   3. a method to test equality between two card objects; and
   4. the toString method to create a String that contains the rank, suit, and point value of the card object. The string should be in the following format:

rank of suit (point value = pointValue)

1. Once you have completed the Card class, find the CardTester.java file in the Activity1 Starter Code folder. Create three Card objects and test each method for each Card object.

A2:

1. Complete the implementation of the Deck class by coding each of the following:
   1. *Deck* constructor
   2. *isEmpty* — This method should return true when the size of the deck is 0; false otherwise.
   3. *size* — This method returns the number of cards in the deck that are left to be dealt.
   4. *deal* — This method “deals” a card by removing a card from the deck and returning it, if there are any cards in the deck left to be dealt. It returns null if the deck is empty.
   5. Add code in the *main* method of DeckTester.java to create three *Deck* objects and test each method for each *Deck* object.

A3:

1. Use the file *Shuffler.java*, found in the Activity3 Starter Code, to implement the perfect shuffle and the efficient selection shuffle methods as described in the Exploration section of this activity. You will be shuffling arrays of integers.
2. *Shuffler.java* also provides a *main* method that calls the shuffling methods. Execute the main method and inspect the output to see how well each shuffle method actually randomizes the array elements. You should execute *main* with different values of *SHUFFLE\_COUNT* and *VALUE\_COUNT*.

A4:

1. The file *Deck.java*, found in the Activity4 Starter Code folder, is a correct solution from Activity 2. Complete the *Deck* class by implementing the *shuffle* method. Use the efficient selection shuffle algorithm from Activity 3.
2. Add additional code at the bottom of the *main* method in the *DeckTester.java* file to create a standard deck of 52 cards and test the *shuffle* method. You can use the *Deck* *toString* method to “see” the cards after every shuffle.

A5:

1. For each error that occurs, write down which method or constructor of the *buggy* *Deck* class could contain the bug, and make an educated guess about the cause of the error.
   1. Buggy 1
   2. Buggy 2
   3. Buggy 3
   4. Buggy 4
2. Now, examine the *Buggy5* folder. This folder contains a *Deck.java* file with multiple errors. Use *DeckTester* to help you find the errors. Correct each error until the *Deck* class has passed all of its tests.

A6:

1. Play a few games of Elevens. How many did you win?

**Activity 1 Questions:**

1. Now think about implementing a class to represent a playing card. What instance variables should it have? What methods should it provide? Discuss your ideas for this Card class with classmates.

**Activity 2 Questions:**

1. Explain in your own words the relationship between a deck and a card.
2. Consider the deck initialized with the statements below. How many cards does the deck contain?

*String[] ranks = {"jack", "queen", "king"};*

*String[] suits = {"blue", "red"};*

*int[] pointValues = {11, 12, 13};*

*Deck d = new Deck(ranks, suits, pointValues);*

1. The game of Twenty-One is played with a deck of 52 cards. Ranks run from ace (highest) down to 2 (lowest). Suits are spades, hearts, diamonds, and clubs as in many other games. A face card has point value 10; an ace has point value 11; point values for 2, ..., 10 are 2, ..., 10, respectively. Specify the contents of the *ranks*, *suits*, and *pointValues* arrays so that the statement

*Deck d = new Deck(ranks, suits, pointValues);*

initializes a deck for a Twenty-One game.

1. Does the order of elements of the *ranks*, *suits*, and *pointValues* arrays matter?

**Activity 3 Questions:**

1. Write a static method named *flip* that simulates a flip of a weighted coin by returning either "heads" or "tails" each time it is called. The coin is twice as likely to turn up heads as tails. Thus, *flip* should return "heads" about twice as often as it returns "tails."
2. Write a static method named *arePermutations* that, given two *int* arrays of the same length but with no duplicate elements, returns *true* if one array is a permutation of the other (i.e., the arrays differ only in how their contents are arranged). Otherwise, it should return *false*.
3. Suppose that the initial contents of the *values* array in *Shuffler.java* are *{1, 2, 3,4}.* For what sequence of random integers would the efficient selection shuffle change *values* to contain *{4, 3, 2, 1}*?

**Activity 4 Questions:**

None

**Activity 5 Questions:**

None

**Activity 6 Questions:**

1. List all possible plays for the board 5♠ 4♥ 2♦ 6♣ A♠ J♥ K♦ 5♣ 2♠
2. If the deck is empty and the board has three cards left, must they be J, Q, and K? Why or why not?
3. Does the game involve any strategy? That is, when more than one play is possible, does it matter which one is chosen? Briefly explain your answer.