Homework #1
Complexity Analysis and Abstract Data Types

Submission guideline:

Submit the homework through blackboard. Before submission make sure your codes do not have any error, unexecutable code and/or late submission will not receive any credit. Submit the complexity calculations of problem one in a .pdf file along with your Java code (.java files only) as a single .zip archive. Include a README text file to give the TA's instructions on how to run your code. The .zip file name should be in the following format:

< firstname >_< lastname >_< id > _hw< num >.zip

For example, if John Doe with student ID 123456789 is submitting the third homework, the submitted file should be named john_doe_123456789_hw3.zip

Number Distribution:

(1) 5 + 5 + 5 = 15
(2) 10
(3) 10
(4) 15

Total Marks: 50

Homework posted on: September 10, 2017

Submission Date: September, 18, 2017

1. Complexity Analysis:
    a. Write a short Java program that takes an array of int values as input and find if there is a pair of numbers with a given sum. What is the total number of operations that occur in terms of input size *n* in closed form for worst case scenario. What is the time complexity in Big O-notation?

    Example:

    ```
    Input: arr[] = {11, 15, 6, 8, 9, 10}, x = 16
    Output: 6 + 10 = 16 , true

    Input: arr[] = {11, 15, 6, 8, 9, 10}, x = 27
    Output: false
    ```

    b. Write a short Java program that takes an array of int values as input and find if there is a triplet of numbers with a given sum. What is the total number of operations that occur in terms of input size *n* in closed form for worst case scenario. What is the time complexity in Big O-notation?

    Example:

    ```
    Input: arr[] = {11, 15, 6, 8, 9, 10}, x = 25
    Output: 11 + 6 + 8 = 25 , true

    Input: arr[] = {11, 15, 6, 8, 9, 10}, x = 39
    Output: false
    ```

c. Given the following matrix multiplication code. What is the total number of operations that occur in terms of input size *n* in closed form. What is the time complexity in Big O-notation? What is the space complexity of this code?

```
for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
            double sum = 0;
            for (int k = 0; k < n; k++) {
                  sum += a[i][k] * b[k][j];
            }
            c[i][j] = sum;
      }
}
```

2. Consider following CreditCard class:

```java
package hw.cse214.cc;

public class CreditCard {
    private String creditCardNumber;
    private String cardHolderName;
    private String bank;
    private int limit;
    private double balance;

    /*
     * Constructor
     */
    public CreditCard(String creditCardNumber, String cardHolderName,
            String bank, int limit, double balance) {
        this.creditCardNumber = creditCardNumber;
        this.cardHolderName = cardHolderName;
        this.bank = bank;
        this.limit = limit;
        this.balance = balance;
    }
    /*
     * Accessor Methods
     */
    public String getCreditCardNumber() { return creditCardNumber; }
    public String getCardHolderName() { return cardHolderName; }
    public String getBank() { return bank; }
    public int getLimit() { return limit; }
    public double getBalance() { return balance;}

    @Override
    public String toString() {
        return "CreditCard [creditCardNumber=" + creditCardNumber
                + ", cardHolderName=" + cardHolderName + ", bank=" + bank
                + ", limit=" + limit + ", balance=" + balance + "]";
    }
}
```

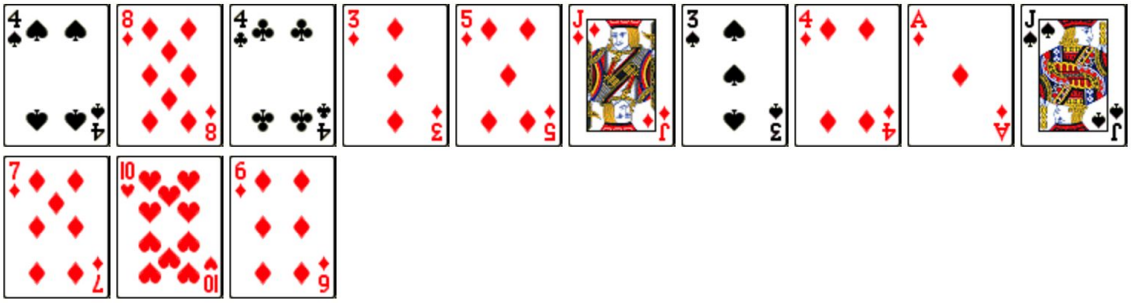
   a. Add an action method **chargeIt(price)**, which is called on a new transaction. This method takes price as argument and returns whether transaction was successful or not. If the purchase makes the balance exceed the limit, transaction should fail.
   b. Add an action method **payment(amount)**, which is called when cardholder makes a payment.
   c. Make necessary changes to add a late fee if payment is done after due date (15th of the month).

3. Suppose, 4 Players are playing a game of cards. Players are distributed cards from a standard 52-cards deck randomly. Before starting the game each player sorts the cards in their hand for convenience during the play. While sorting, a player first looks at the suits and groups same suits together. For this certain version of that game, the order of the value of suits is as follows:
    a. spades (♠)
    b. hearts (♥)
    c. diamonds (♦)
    d. clubs (♣)

    Among the same suit of cards the order of cards: A, K, Q, J, 10, 9, 8, 7, 6, 5, 4, 3, 2

    1. Write a class Card
    2. Write a class Player that has
        a. An array of 13 cards
        b. A method to sort the array of cards (you can use the Insertion Sort shown in the class)

    A sample test program is given here:



```java
package hw.cse214.cards;

public class CardSortingTest {

    public static void main(String[] args) {

        String[] cardsForPlayer1 = {
                            "S4", "D8", "C4", "D3", "D5", "DJ", "S3", "D4", "DA", "SJ",
                            "D7", "H10", "D6"
                        };

        Card[] cards = new Card[13];
        for (int i = 0 ; i < cardsForPlayer1.length; i++){
            Card mCard = new Card(cardsForPlayer1[i]);
            cards[i] = mCard;
        }

        int id = 1;
        Player player = new Player(id);
        player.setCards(cards);
        player.printCards();
        // should print: S4 D8 C4 D3 D5 DJ S3 D4 DA SJ D7 H10 D6

        /*
         * Sort and show output
         */
        player.sortCards();
        player.printCards();
        // should print: SJ S4 S3 H10 DA DJ D8 D7 D6 D5 D4 D3 C4
    }
}
```

4. Suppose your Car/**Vehicle** has a navigation system/**GPS**. You can set your destination **Location** (x, y) and current **Location** (a, b) in your **GPS**. The **GPS** then calculates approximate time to reach the destination and shows in the screen along with turn by turn direction. For simplicity let us consider a 2d plane, where instead of (latitude, longitude), we consider (x,y) coordinate and since we do not have turn by turn navigation data, let us consider that the **GPS** calculates straight line shortest distance between current and destination **Location**s.
    a. Design and implement a class for Location
    b. Design and implement a class for GPS
    c. Design and implement a class for Vehicle/Car that has a GPS
    d. Create a public testing class with main method where:
        i. You can create a Car and set its current speed
        ii. You can set the current Location of the GPS
        iii. You can set the destination Location of the GPS
        iv. Once the above values are set, the vehicle and it's GPS should be able to calculate
            1. the distance between current and destination
            2. Approximate time required to reach the destination

Here is a sample test program:

```
package hw.cse214.navigation;

public class NavigationTest {

    public static void main(String[] args) {

        Location myCurrentLocation = new Location(354, 538);
        Location myDestination = new Location(108, 25);

        Car myCar = new Car();

        /*
         * Initializing GPS object and setting current and destination locations
         */
        GPS myGPS = new GPS();

        myGPS.setCurrentLocation(myCurrentLocation);
        myGPS.setDestination(myDestination);
        myCar.setGPS(myGPS);

        /*
         * Start the car and set current speed
         */
        myCar.setCurrentSpeed(35.0);


        // see output
        System.out.println("Distance: " + ((GPS) myCar.getGPS()).getCalculatedDistance());
        System.out.println("Arrival : " + ((GPS) myCar.getGPS()).getArrivalTime());

    }
}
```