Instructions: This homework is intended to test your understanding on OOP concepts and Java Arrays. You are free to discuss the problems with your classmates and instructor. However, it is not allowed to share code. Please indicate the source clearly if you take any help from internet and write sufficient comments to demonstrate your understanding. You will be benefited most if you try to solve these problems by yourself. No late submissions will be accepted. Please submit only the java source files (StackDemo.java, Polygon.java and AirplaneSeatReservation.java).

Problem 1 (45 points):

This problem will ask you to add some more methods to the StackDemo class we studied in lecture. We have already seen methods push(), pop() and printElements(). Your task is to add the following new methods:

a) getNumberOfSlotsAvailable(): This method simply returns the number of available slots in stack. For example, if the capacity of stack is 10 and if there are currently 6 elements in stack, the number of available slots to be returned is 4. This method takes no parameter and returns an integer.

b) pushArrayUsingLoop(): This method takes an integer array arrayToPush as the only parameter. It pushes all the elements of arrayToPush to stack if there are enough available slots. If the size of the array arrayToPush is larger than available slots in stack (i.e. insufficient space to push all elements), the pushArrayUsingLoop() method pushes elements from array arrayToPush until the stack is full. The method then reports “Stack overflow” with the number of elements (of array arrayToPush) that could not be pushed. Check the sample output go get examples. As indicated by the name of this method, you must use a loop for pushing elements from array arrayToPush.

c) pushArrayUsingCopy(): This method works exactly the same as the previous method. However, you are required to use the System.arraycopy() method instead of a loop.

d) pushArrayUsingLoopReverse(): This method is similar to method pushArrayUsingLoop(). The only difference is that it pushes elements from array arrayToPush in the reverse order. Check the sample output for an example.

e) popMultiple(): This function takes an integer k as the only parameter. It pops k elements from the stack. If there are less than k elements in stack, the method simply empties the stack and reports “stack underflow”.

You are given an incomplete program with a sample main function and also the desired output for this program. The output generated by your program should be the same as the desired output. Note that, capacity is initialized to be 10 in the program given to you.

Code template (file StackDemo.java):

```java
public class StackDemo {  
    static final int capacity = 10;  
    int arr[] = new int[capacity];  
    int top = -1;
```
public void push(int pushedElement) {
    if (top < capacity - 1) {
        top++;  
        arr[top] = pushedElement;
        System.out.println("Element " + pushedElement + " is pushed to Stack !");
        printElements();
    } else {
        System.out.println("Stack Overflow !");
    }
}

public int getNumberOfSlotsAvailable(){
    //implement this method
}

public void pushArrayUsingLoop(int[] arrayToPush){
    //implement this method
}

public void pushArrayUsingCopy(int[] arrayToPush){
    //implement this method
}

public void pushArrayUsingLoopReverse(int[] arrayToPush){
    //implement this method
}

public void pop() {
    if (top >= 0) {
        top--;
        System.out.println("Pop operation done !");
    } else {
        System.out.println("Stack Underflow !");
    }
}

public void popMultiple(int k){
    //implement this method
}

public void printElements() {
    if (top >= 0) {
        System.out.print("Elements in stack :");
        for (int i = 0; i <= top; i++) {
            System.out.print(" "+ arr[i]);
        }
        System.out.println();
    }
}

public static void main(String[] args) {
    StackDemo stackDemo = new StackDemo();
    System.out.println(stackDemo.getNumberOfSlotsAvailable());
    stackDemo.pop();
    stackDemo.push(23);
    stackDemo.push(2);
    stackDemo.pushArrayUsingLoop(new int[]{2,5,8,4,6,9});
    stackDemo.printElements();
    stackDemo.pushArrayUsingLoopReverse(new int[]{100,200,300,400,500});
    stackDemo.printElements();
    stackDemo.popMultiple(12);
    stackDemo.printElements();
    stackDemo.pushArrayUsingCopy(new int[]{100,200,300,400,500});
Problem 2 (20 points):

Consider the following incomplete program (Polygon.java):

```java
public class Polygon {
    Point[] points;
    Polygon(Point[] points){
        this.points = points;
    }

    double getLengthOfSides() { 
        //your code goes here
    }

    public static void main(String[] args){
        Point[] arrayOfPoints = {new Point(0,0), new Point(0,10),
                                new Point(10,10), new Point(10,0)};
        Polygon myPolygon = new Polygon(arrayOfPoints);
        System.out.println("Total Length of all sides: "+myPolygon.getLengthOfSides());
    }
}

class Point{
    int x,y;
    Point(int x, int y){
        this.x = x;
        this.y = y;
    }
    int getX(){
        return x;
    }
}
```

Desired output:

10
Stack Underflow!
Element 23 is pushed to Stack!
Elements in stack: 23
Element 2 is pushed to Stack!
Elements in stack: 23 2
Pushing array {2, 5, 8, 4, 6, 9} using method PushArrayUsingLoop
Elements in stack: 23 2 2 5 8 4 6 9
Pushing array {100, 200, 300, 400, 500} using method PushArrayUsingLoop
Stack Overflow! 3 items could not be pushed
Elements in stack: 23 2 2 5 8 4 6 9 100 200
Popping 12 items from stack
Stack Underflow
Pushing array {100, 200, 300, 400, 500} using method pushArrayUsingCopy
Elements in stack: 100 200 300 400 500
5
Pushing array {100, 200, 300, 400, 500, 600, 700} using method pushArrayUsingLoopReverse
Stack Overflow! 12 items could not be pushed
Elements in stack: 100 200 300 400 500 700 600 500 400 300
int getY()
{
    return y;
}

Desired output:
    Total length of all sides: 40.0

The Polygon class uses an array points which is an array of objects of class Point. An object of Point class simply represents a point in 2D space with x and y coordinates. The array is used to store a sequence of endpoints to represent a polygon. The method getLengthOfSides() computes and returns the total length of all sides in the polygon. Your only task for this problem is to implement this method.

Hints: Use a loop. You may need to use Math.sqrt() and Math.pow() methods.

Problem 3 (35 points):

This problem is inspired by the programming project 11 at page 423 of your textbook. You are going to develop a simple airplane seat reservation system in this problem. Assume a small airplane with seat numbers shown in the following pattern:

1 A B C D
2 A B C D
3 A B C D
4 A B C D
5 A B C D
6 A B C D
7 A B C D
8 A B C D

Notice that there are eight rows and four columns in this example. Assume an 'X' marks that the seat is already assigned. Thus, if seats 2C and 5A are already assigned, the representation should look like:

1 A B C D
2 A B X D
3 A B C D
4 A B C D
5 X B C D
6 A B C D
7 A B C D
8 A B C D

You are given an incomplete program (AirplaneSeatReservation.java) that uses a two-dimensional character array seats. Check the constructor method of the given program carefully to understand how the array is initialized. Initially no seat is assigned. Your task is to implement the following methods:

a) reserveSeat (): This method takes an integer parameter row and a character parameter col. Clearly these two parameters represent a specific seat that corresponds to a specific slot in the two-dimensional array. The method simply puts the character 'X' in the correct slot to mark the corresponding seat as taken. For example, if row is 2 and col is 'A', this method replaces the content of seats[1][0] by the character 'X'. Remember that the index of the first element in an array is zero (not one).
b) **freeSeat()**: This method takes exactly the same parameters as the method **reserveSeat()**, but does just the opposite. It puts the character given by parameter **col** at the slot (in 2D array) that represents the seat specified by the parameters **row** and **col**. For example, if **row** is 2 and **col** is 'A', this method replaces the content of **seats[1][0]** by the character 'A'.

c) **getNumberOfSeatsAvailable()**: Counts and returns the number of available (i.e. unassigned) seats. The method returns an integer.

d) **displaySeats()**: Displays seats in the format shown in output.

Note that for the given main method, your output should exactly match with the given output.

**Incomplete program (AirplaneSeatReservation.java):**
```java
public class AirplaneSeatReservation {
    char[][] seats;
    //constructor method
    AirplaneSeatReservation(int numOfRows, int numOfColumns) {
        seats = new char[numOfRows][numOfColumns];
        for (int i = 0; i < numOfRows; i++)
            for (int j = 0; j < numOfColumns; j++)
                seats[i][j] = (char)(65+j);
    }
    void reserveSeat(int row, char col) {
        //implement this method
    }
    void freeSeat(int row, char col) {
        //implement this method
    }
    int getNumberOfSeatsAvailable() {
        //implement this method
    }
    void displaySeats() {
        //implement this method
    }
    public static void main(String[] args) {
        AirplaneSeatReservation airplane1 = new AirplaneSeatReservation(8, 4);
        airplane1.displaySeats();
        System.out.println(airplane1.getNumberOfSeatsAvailable());
        airplane1.reserveSeat(2, 'C');
        airplane1.reserveSeat(2, 'D');
        airplane1.reserveSeat(6, 'B');
        System.out.println(airplane1.getNumberOfSeatsAvailable());
        airplane1.displaySeats();
        airplane1.freeSeat(2, 'C');
        System.out.println(airplane1.getNumberOfSeatsAvailable());
        airplane1.displaySeats();
    }
}
```
Desired output:
1 A B C D
2 A B C D
3 A B C D
4 A B C D
5 A B C D
6 A B C D
7 A B C D
8 A B C D

32
29
1 A B C D
2 A B X X
3 A B C D
4 A B C D
5 A B C D
6 A X C D
7 A B C D
8 A B C D

30
1 A B C D
2 A B C X
3 A B C D
4 A B C D
5 A B C D
6 A X C D
7 A B C D
8 A B C D