## Lab Objective

1. **Write** the definition of the super class and extend it to create multiple subclasses.
2. **Write** codes to implement polymorphism.

## Lab Activities

### A. Abstract Class

- Create an abstract class 'Shape' with one attributed named ‘area’ of double type.
- Write proper setter and getter for the attributes
- Three abstract methods namely 'RectangleArea' taking two parameters, 'SquareArea' and 'CircleArea' taking one parameter each.
- The parameters of 'RectangleArea' are its length and breadth, that of 'SquareArea' is its side and that of 'CircleArea' is its radius.
- Now create another class 'Area' containing all the three methods 'RectangleArea', 'SquareArea' and 'CircleArea' for calculating the area of rectangle, square and circle respectively.
- Create an ArrayList of Shape type. Your main () method must display the following first:

```
Press (1) for calculating Rectangle Area
Press (2) for calculating Square Area
Press (3) for calculating Circle Area
```

- You must create at least 3 shape type reference variable and assign area type object to them in this manner
- Call the respective method for all three objects and display the area.

### B. Comparable Interface

- Comparable Interface is used to compare two objects. In this problem, you'll create a class that implements the comparable interface and use it to sort an array of objects.
- Create a *Player* class with 2 fields: name of String Type and score of integer type.
- Define proper constructor to set the attributes value
- Modify the Player class to implement the Comparable interface
- Given an array of *n Player* objects and sort them in order of decreasing score; if 2 or more players have the same score, sort those players alphabetically by name. To do this, you must override the compareTo ( *Player b )* method of comparable interface in the player class.

**Input Format**

The first line contains an integer, *n*, denoting the number of players.
Each of the *n* subsequent lines contains a player's *name* and *score*, respectively.

**Output Format**

Print each sorted element in the format: *namescore*

**Sample Input**

```
5
amy 100
david 100
heraldo 50
aakansha 75 aleksa
150
```

**Sample Output**

```
aleksa 150
amy 100
david 100
aakansha 75
heraldo 50
```

### C. Java ArrayList

Sometimes it's better to use dynamic size arrays. Java's Arraylist can provide you this feature. Try to solve this problem using Arraylist.

You are given *n* lines. In each line there are zero or more integers. You need to answer a few queries where you need to tell the number located in $y^{th}$ position of $x^{th}$ line. Take your input from System.in.

**Input Format**

The first line has an integer *n*. In each of the next *n* lines there will be an integer *d* denoting number of integers on that line and then there will be *d* space-separated integers. In the next line there will be an integer *q* denoting number of queries. Each query will consist of two integers *x* and *y*.

**Output Format**

In each line, output the number located in $y^{th}$ position of $x^{th\ line}$. If there is no such position, just print "ERROR!"

**Sample Input**

```
5
5 41 77 74 22 44
1 12
4 37 34 36 52
0
3 20 22 33
5
1 3
3 4
3 1
4 3
5 5
```

**Sample Output**

```
74
52
37
ERROR!
ERROR!
```

```
ArrayList<ArrayList> listArray = new ArrayList<ArrayList>();****
ArrayList< Integer > intArrayList =  new ArrayList< Integer >();                    for(int
j=0;j<numOfIntegers;j++){
                intArrayList.add(new Integer(sc.nextInt()));
            }
            listArray.add(intArrayList);
```