**Classes**

1. Design a class named **Triangle** to represent a triangle. The class contains:
   1. Three double data fields named **side1**, **side2** and **side3** that specify the three sides of the triangle. The default values are 1 for all the sides.
   2. A no-args constructor that creates a default triangle.
   3. A constructor that creates a triangle with the specified sides.
   4. The accessor and modifier methods for all the data fields.
   5. A method named **getArea()** that returns the area of this triangle.
   6. A method named **getPerimeter()** that returns the perimeter.
2. Design a class named **Fan** to represent a fan. The class contains:
   1. Three constants named **SLOW, MEDIUM** and **FAST** with values 1,2 and 3 to denote the fan speed
   2. An integer data field named **speed** that specifies the speed of the fan (default is SLOW)
   3. A boolean data field named **on** that specifies whether the fan is on or not (default is false)
   4. A double data field named **radius** that specifies the radius of the fan (default is 5)
   5. A string data field named **color** that specifies the color of the fan (default is blue)
   6. A no-args constructor that creates a default fan
   7. The accessor and modifier methods for all four data fields
   8. A method **toString()** that returns a string description of the fan. If the fan is on, the method returns the fan speed, color and radius in one combined string. If the fan is not on, the method returns the fan color and the radius along with the string “fan is off” in one combined string.
3. Create a class Bicycle with fields – gear, speed and brand. Include constructors, accessor and modifier methods, and methods – *speedup* which takes an integer argument and increments the speed by this amount and method – applybrake which also takes an integer argument and decrements the speed by that amount. Include a toString method to print the fields of the Bicycle. Create another class – MountainBike which inherits from Bicycle with additional field seatHeight. Define constructor, accessor and modifier methods from this class. Finally test your classes using a driver class.

**Interface**

1. Define an interface ‘Comparable’ with one method ‘isEqual’. This method takes as argument a String and returns a boolean. Define a class Car which implements this interface. The Car class has data member ‘license plate no.’ and an implementation of the isEqual method. The method takes as argument the license plate no. and compares it with the license plate no. of the current object. If they are same, then the method returns true else false. Define another class ‘Employee’ which also implements the Comparable interface. The Employee class has data member ‘name’ and an implementation of the isEqual method for comparing names. Finally, test the classes in a tester class.

**Polymorphism**

1. Design a class named Person and its two subclasses named Student and Employee. Make Faculty and Staff subclasses of Employee. A person has a name, address, phone number and email address. A student has a class status (fresher, sophomore, junior, senior). Define the status as a constant. An employee has an office, salary and date-hired. Define a class named MyDate that contains the fields year, month and day. A faculty member has office hours and a rank. A staff member has a title. Override the toString method in each class to display the class name and the person’s name. Make sure that polymorphism is illustrated in this exercise.

**Arrays and Inheritance**

1. Explore the built-in class ArrayList and use it in the following program.   
   Write a program that creates an ArrayList and adds a Person object, Student object, Employee object, Faculty object and Staff object to this list. Use a loop to display all the elements in the list by invoking the toString method.
2. Create a class MyStack that implements a stack data structure. The class inherits from the ArrayList class and contains methods –

* boolean isEmpty() which returns true if the stack is empty; false otherwise
* int getSize() which returns the number of elements in the stack
* Object pop() which returns an Object from the top of the list
* Object push(Object o) which adds an Object to the list
* int search(Object o) which returns the index of the Object from the list
* toString which displays all elements of the list.

Test this class using a tester class.

1. Implement the classes given in this UML diagram (**Next Page**)

GeometricObject

Color : String

Filled: Boolean

dateCreated: Date

GeometricObject()

getColor():String

setColor(color:String) : void

isFilled() : Boolean

setFilled(filled:Boolean): void

getDateCreated():Date

toString():String

**getArea(): double**

**getPerimeter():double**

Abstract methods

Abstract Class

Circle

Radius:double

Circle()

Circle(radius:double)

getradius():double

setRadius(radius:double) : void

getDiameter():double

Rectangle

Width:double

Height:double

Rectangle()

Rectangle(width:double,height:double)

getwidth():double

setWidth(width:double) : void

getHeight():double

setHeight(height:double):String