OBJECT ORIENTED PROGRAMMING LAB

EXRECISE SET 4

METHOD OVERLOADING AND METHOD OVERRIDING

1. Consider the initial design of a class that represents bank accounts:

```
class Account
{
        private long acct_no;
        private double acct_bal;

        Account(){}
        Account (long id, double amount){.......}

        void deposit (double amount){.....}
        bool withdraw (double amount){.....}
        long get_id ( ){.......}
        double get_balance ( ){.......}

}
```

a) Complete the method definitions.

b) Add a copy constructor to the Account class.

c) Include a method for money transfer that transfers a given amount from another account to the host account.

d) Write a driver class (Banking.java) that creates two Account objects -- ravi using default constructor, mala using parameterised constructor, deposit in and withdraw from mala. Create another object leela using copy constructor from mala; display the details of leela. Also, demonstrator transfer method.

e) Add a constructor that randomly generates a 8-digit account number and assigns it to the acct_no data member. Also, it assigns a **default minimum balance** to the acct_bal data member. For that, you need to add a **static** data member for minimum balance to the class definition.

2. Consider the following class for complex numbers

```
class Complex
{
        private float real;
        private float imaginary;
```

```
        Complex() {}
        Complex(float r, float i){....}

        void display(){....}
        bool isReal (){....}
        bool isImag (){....}

        final float getReal(){....}
        final float getImaginary(){....}
}
```

a) Complete all method definitions.

b) Add a method to multiply two complex numbers.

c) Overload the above method to implement multiplication by a real number, and by an imaginary number.

d) Write a method to add/subtract two complex numbers.

e) Overload the above method to add/subtract an integer to/from a complex number.

g) Implement a driver class to demonstrate these methods.

3. Consider a class for fractions given below:

```
class Fraction
{
        private int num,
        private long int denom;

        Fraction() {}
        Fraction(int n, int d){.....}

        void display(){.....}
        bool isZero () {.....}
        bool IsOne () {.....}
        bool isInt() {.....}
        int floor(){.....}
        int ceiling(){.....}

        final int getNum(){....}
        final int getDenom(){....}
}
```

a) Complete all method definions.

b) The parameterised constructor should have a logic to reduce a fraction to its canonical form by removing the gcd between n and d. (It means 1⁄2 and 2/4 are equal; 2/4 must be reduced to 1⁄2)

c) Write a method that compares two fractions and returns 0 if they are equal, and 1 otherwise.

d) Overload the above method to compare a fraction with an integer.

e) Write a method to add two fractions.

f) Overload the above method to add an integer to a fraction.

g) Implement a driver class to demonstrate these methods.

5. Recall the **Vector** class we implemented previously.

```
class Vector{
        private int x;
        private int y;

        Vector(){}
        Vector(int x,int y){....}

        Vector sum(Vector vec){.....}
        Vector difference(Vector vec){.....}
        double norm(){.....}
        double product(Vector vec){......}//computes dot product
        boolean perpendicular(Vector vec){......}
        boolean parallel(Vector vec){......}

        final int getX(){....}
        final int getY(){....}
}
```

a) Add a copy constructor to the Vector class.

b) Overload product method to implement scalar product.

c) Write a driver class to demonstrate the working of methods in Vector class.

d) Extend Vector class to implement 3D Vectors.

6. Consider the following class called **Figure** that stores the dimensions of a two-dimensional object. This class defines a method called area() that computes the area of a two-dimensional object. As the object is not well-defined, its area is undefined and the method returns an error message. The task is to derive two subclasses from Figure. The first is Rectangle and the second is Triangle. Each of these subclasses overrides area( ) so that it

returns the area of a rectangle and a triangle, respectively. Also, write a driver class that demonstrates the overridden methods.

```
class Figure {
        private double dim1;
        private double dim2;

        Figure(double a, double b) {
           dim1 = a;
           dim2 = b;
        }

        double area(){
           System.out.println("Area for Figure is undefined.");
           return 0;
        }

        final double getDim1(){....}
        final double getDim2(){....}
}
```

7. Using the **Account** class as a base class, write two derived classes called **SavingsAccount** and **CurrentAccount**. A SavingsAccount object, in addition to the attributes of an Account object, should have an interest variable and a method which adds interest to the account. A CurrentAccount object, in addition to the attributes of an Account object, should have an overdraft limit variable. Ensure that you have overridden methods of the Account class as necessary in both derived classes.

a) Now create a **Bank** class, an object of which contains an array of Account objects. Accounts in the array could be instances of the Account class, the SavingsAccount class, or the CurrentAccount class. Create some test accounts (some of each type).

b) Write an update method in the Bank class. It iterates through each account, updating it in the following ways: Savings accounts get interest added (via the method you already wrote); CurrentAccounts get a letter sent if they are in overdraft.

c) The Bank class requires methods for opening and closing accounts, and for paying a dividend into each account.

8. Suppose we have the **Circle** class:

```
class Circle
{
   private double radius;
   public Circle(double r) { radius = r; }
   public double getRadius() { return radius; }
```

```
    public double findCircumference() {
        return 2*Math.PI*radius;
    }
    public double findArea() {
        return radius*radius*Math.PI;
    }
}
```

Extend the Circle class to **Cylinder** class with an extra data member length. Overload the findArea() method to compute the surface area of a Cylinder. Add a method findVolume() to compute the volume of the Cylinder.