**Java Laboratory Manual**

**Laboratory Project 1:**

**Problem Statement:**

We know that radio is a very useful device. A radio should have the following capabilities:

1. Turn the power on/off
2. Channel Up
3. Channel Down
4. Volume Up
5. Volume Down

In order to provide these capabilities, radio must retain some information with it, for example current state (on or off), current volume level, current channel. When we have a radio and turning it ON will have its initial values of these state variables. This is the job of your class constructor.

You will need to have an object that simulates the radio, and a main program to actually drive the radio. This should simply be a loop that will allow the user to type in requests.

You need to give a menu that the user can choose from. It should look something like the following:

Press:

P...to press the power button

U...to press the channelUp button

D...to press the channelDown button

+...to press the volumeDown button

-...to press the volumeUp button

Q...to exit simulation

Also, you need to show the state of the radio after every time the user does something.

Current State: Channel: 630 AM Volume: 11 Power: ON

**Note:** Please have an upper and lower limit on the radio station and the volume (ie. volume should be between 0 and 100). Also, it is probably easier to use AM instead of FM, so you don't have to use decimal numbers, but either will do.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Laboratory Project 2:**

**Problem Statement:**

If we go for hiking, we may need a compass to give us the directions. A Compass object will be endowed with two distinct behaviours, one called **rotate**, which will change the state of a Compass object by advancing the value of its state variable in the clockwise direction, and one called **print**, which will announce its orientation by printing a string to standard output, "NORTH" for NORTH, and so on.

Initially, objects of type Compass will have only one data member to record their states, and that member will be allowed to assume only four distinct values, NORTH, EAST, SOUTH and WEST. Since it can take on only an enumerated set of values you will represent it with an *enum*.

In addition you are required to provide for initialization of Compass objects through a constructor which sets the state variable (a.k.a. data member) to the value NORTH.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Laboratory Projects: 3 and 4**

**General Comment:**

We all know that the concepts of vector, complex number, polynomial, binary trees etc have got many applications. However we don’t have any types which directly implement these concepts. Then why not we design them. So the next two assignments are on the design of useful user defined types- also known as ADTs. *Only three examples have been taken up here, you can design many such structures such as rational number, complex number, set etc.*

**Laboratory Assignment 3:**

**Problem statement:**

A vector consists of three components in three space. We will use the coordinates x, y and z to describe a vector. The usual notation for displaying a vector is to put a line over the top or bold face it. We will use the bold face notation for this discussion.

**A** = Ax**i +** Ay**j** + Az**k**

where the **A** is vector, Ax, Ayand Az are the components in the x, y and z directions and finally **i**, **j**, and **k** are vectors that point in the x, y and z directions and have a magnitude of 1.

There are several operations that can be defined for vectors. They are:

1. addition of two vectors displayed as **A** + **B**
2. dot product sometimes shown as **A** dot **B** where the dot would normally be a period in the middle of the line.
3. cross product sometimes shown as **A** X **B.**
4. multiplication by a scalar (a non-vector value that has only magnitude.) **A** \* c where c is the scalar.
5. equality and not equal represented by == and !=.
6. the print function called Print should display the value of a vector in the form Ax i **+** Ayj + Azk.

The following are the definitions for each of these operations:

**Addition: A** + **B** = (Ax + Bx )**i** + (Ay + By )**j** + (Az + Bz )**k**.

**Dot product: A** \* **B** = Ax\*Bx **+** Ay\*By **+** Az\*Bz where we will use the \* for the dot product - notice this is a scalar.

**Cross product: A** X **B** = **i**\*(Ay\*Bz**-**Az\*By) **- j**\*(Ax\*Bz - Az\*Bx) + **k**\*(Ax\*By - Ay\*Bx) where we will use the X for the cross product

**Scalar multiplication: A** \* c = c \* **A** = (Ax \* c)**i** + (Ay \* c)**j** + (Az \* c)**k**

**Equality: A** == **B** this should be true only if Ax = Bx , Ay = By and Az = Bz

You are to write a class called Vector which contains all these functionalities. Finally write a main program that will exercise all the operations.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

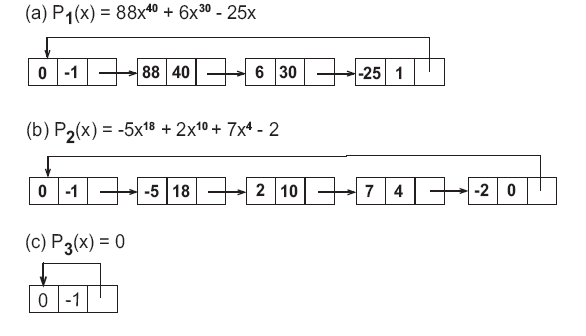
**Laboratory Project 4:**

**Problem statement:**

Design, implement and test an ADT Polynomial that provides some of the basic operations for univariate polynomials.

For example, P*(x) = 3x4 − 7x + 18* is such a polynomial in *x*.

Your ADT **Polynomial** should be implemented as a Circular List of terms with a header node (term consists of coefficient and exponent and a pointer to next term in the chain). The circular list representation of a polynomial has one node for each term that has non-zero coefficient. The terms are in decreasing order of exponent and the head node has its coefficient and exponent field equal to 0 and -1 respectively. The following figure gives some examples.



You must use the two classes Node, and Polynomial (should be implemented as a circular structure) to do the job:

The ADT Polynomial should support the following operations. Note that some operators should be overloaded to make your code more readable. You may add any other public/private member functions that you think are necessary.

Polynomial()

- Create the zero polynomial, that is *P(x) = 0*. Polynomial() is the class default constructor.

friend istream& operator>>(istream&, const Polynomial&)

- Read in a polynomial in the following form:

*c1 e1 c2 e2 · · · cm em* 0 − 1

where *ci* and *ei* are integers denoting the coefficient and exponent of the ith term, respectively. The last pair 0 − 1 denotes the end of polynomial. You can assume that the exponents are in decreasing order; that is *e1 > e2 > . . . > em ≥ 0*, and there is no zero coefficient in the input; that is *ci <> 0* for all i.

**Functions:**

- Output the polynomial to cout. The output format should be the same as the input format. That is, the exponents should be in decreasing order and all coefficients are non-zero. Also it should end with the pair 0 − 1.

* Add the two polynomials p1 and p2 and return the result. P1 and p2 are left unaltered.
* Subtract the first polynomial p1 from the second polynomial p2 and return the result leaving p1 and p2 unchanged.
* Multiply the two polynomials p1 and p2 and return the result. Leave p1 and p2 unaltered.
* Assign Polynomial a to **\*this** ( ie the current polynomial)

~Polynomial()

* The polynomial destructor. It disposes all the nodes to the heap

Input and output. You need to write a main program which obtains an input line from users. The end of input is signalled by the Ctrl-Z character which your program should detect. Each input line is a sequence of integers that are separated by blanks and have one of the following three possible formats:

1 <polynomial> + <polynomial>

2 <polynomial> - <polynomial>

3 <polynomial> \* <polynomial>

where <polynomial> represents one polynomial.

1. means that the two polynomials are to be added;

2 means that the second polynomial should be subtracted from the first polynomial,

3 means that the two polynomials are to be multiplied.

Note that you should make use of the overloaded input and output operator functions to get polynomials and output the result respectively. Similarity, you should use the overloaded arithmetic operators to calculate the results.

**Laboratory Project 5:**

**Problem statement:**

Create an inheritance hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).

Create an inheritance hierarchy containing base class *Account* and derived classes *SavingsAccount* and *CheckingAccount* that inherit from class *Account*.

Base class *Account* should include one data member of type double to represent the account balance. The class should provide the following member functions:

* a constructor that receives an initial balance and uses it to initialize the data member. The constructor should validate the initial balance to ensure that it is greater than or equal to 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid.
* Member function *credit* should add an amount to the current balance.
* Member function *debit* should withdraw money from the Account and ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the function should print the message "Debit amount exceeded account balance."
* Member function *getBalance* should return the current balance.

Derived class *SavingsAccount* should inherit the functionality of an Account, but also include a data member of type double indicating the interest rate (percentage) assigned to the *Account*. The members functions of *SavingsAccount* are:

* *SavingsAccount*'s constructor should receive the initial balance, as well as an initial value for the *SavingsAccount*'s interest rate.
* Member function *calculateInterest* that returns a double indicating the amount of interest earned by an account. Member function *calculateInterest* should determine this amount by multiplying the interest rate by the account balance. [Note: SavingsAccount should inherit member functions *credit* and *debit* as is without redefining them.]

Derived class *CheckingAccount* should inherit from base class *Account* and include an additional data member of type double that represents the fee charged per transaction. The member functions are:

* *CheckingAccount*'s constructor should receive the initial balance, as well as a parameter indicating a fee amount.
* Class *CheckingAccount* should redefine member functions *credit* and *debit* so that they subtract the fee from the account balance whenever either transaction is performed successfully. *CheckingAccount's* versions of these functions should invoke the base-class *Account* version to perform the updates to an account balance. *CheckingAccount's* debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [Hint: Define Account's debit function so that it returns a *bool* indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.]

After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the *SavingsAccount* object by first invoking its *calculateInterest* function, then passing the returned interest amount to the object's credit function.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Laboratory Project 6:**

**Problem Statement:**

For this assignment, you are to pretend that you are creating a drawing program. Such a program, in reality, must be quite complicated. However in this assignment we will implement or rather mimic one important part of it. What is required in such a program is to maintain information about all objects that are currently drawn on the screen. So we design a container for this purpose.

Drawing objects can be any of the following:

1. Point
2. Line
3. Rectangle
4. Square
5. Circle

Also, Rectangle, Square, and Circle must have filled versions that are filled with a certain color.

This is a description of all the objects in the program:

1. All objects will have a certain color.
2. The filled objects will have a color **and** a fill color, which is not necessarily the same.
3. A **Point** will simply be an x and y coordinate that represents the location of the point.
4. A **Line** will be represented by two sets of x and y coordinates, which are the endpoints.
5. A **Rectangle** has a top left, and a bottom right coordinate.
6. A **Square** is a rectangle that must have all the sides the same size.
7. A **Circle** has a center coordinate, and a radius.

The list of Objects will be a *heterogeneous container*, which means that it is a list that can store more than one type. The only restriction is that the types must all be related. What we can do here is that an abstract class *Object* can be used and all the objects in the drawing program will be subtypes of *Object* class.

There are two "good" options as to how to get the information from the user. Obviously, if this was a real drawing program, the user input would be retrieved via the mouse, and clicking around on the screen. Since we don't want to do that, we are going to get the information about the objects from the keyboard. You can do this either by putting the requests to the user in the constructors for the different objects, or by creating a function like *get\_info\_from\_user()* that asks for the appropriate data. This would need to be a pure virtual function in the *Object* class and that would be redefined for the different subclasses. The advantage of using a constructor is that it is called automatically. Thus, when you create a triangle, for example, the constructor for any parent classes of triangle will get called, and then the constructor for triangle will get called. This will automatically retrieve all the info that is necessary for the triangle object.

You will need to provide two functions for the list of objects. You need to be able to add an object and draw all the objects. Drawing the objects just means saying something like

Drawing Rectangle: Color 5 Top Left (3,5) Bottom Right (6,7)

Drawing Point: Color 9 (6,7)

Part of the grading of this program will reflect how well you designed the class hierarchy to represent the different types of drawing objects. There should be as little code repetition as possible, as that indicates a poorly designed inheritance structure.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*