**CPP Lab 5 – Day 5, 6, 7, 8**

Roll no. 248121

PRN: 240841220016

Name: Akshay Chavan

1. Solve this.

Fresh business scenario to apply inheritance , polymorphism to emp based organization scenario.

Create Emp based organization structure --- Emp , Mgr , Worker

1.1 Emp state--- id(int), name, deptId , basicSalary(double)

Accept all of above in constructor arguments.

Methods ---

1.2. compute net salary ---ret 0

(eg : public double computeNetSalary(){return 0;})

1.2 Mgr state ---id,name,basic,deptId , perfBonus

Add suitable constructor

Methods ----

1. compute net salary (formula: basic+perfBonus) -- override computeNetSalary

1.3 Worker state --id,name,basic,deptId,hoursWorked,hourlyRate

Methods :

1. compute net salary (formula: = basic+(hoursWorked\*hourlyRate) --override computeNetSalary

2. get hrlyRate of the worker -- add a new method to return hourly rate of a worker.(getter)

Create suitable array to store organization details.

Provide following options

1. Hire Manager

I/P : all manager details

2. Hire Worker

I/P : all worker details

3. Display information of all employees net salary (by invoking computeNetSal),

4. Exit

**Code:**

**Employee.h – declaration file**

`

// This file conatins declarations

//  1 Solve this.

//  Fresh business scenario to apply inheritance , polymorphism   to emp based organization scenario.

#include <iostream>

using namespace std;

class Emp

{

private:

    static int idGenerator;

    int id, deptId;

    string name;

protected:

    double basicSalary;

    // Constrs

public:

    Emp();

    Emp(string name, int deptId, double basicSalary);

    // Methods

    virtual double computeNetSalary();

    string getName();

};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Mgr : public Emp

{

private:

    double perfBonus;

public:

    // constrs

    Mgr();

    Mgr(string name, int deptId, double basicSalary, double perfBonus);

    // methods

    double computeNetSalary();

};

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

class Worker : public Emp

{

private:

    int hoursWorked;

    float hourlyRate;

public:

    Worker();

    Worker(string name, int deptId, double basicSalary, int houseWorked, float hourlyRate);

    // methods

    double computeNetSalary();

    // Getters

    float getHourlyRate();

};

`

**Emp.cpp – Implementation of Emp class**

`

// conatins implementations of Emp class from Employees.h

#include <iostream>

#include "Employees.h"

using namespace std;

// static var - idGenerator

int Emp::idGenerator = 100;

// constrs

Emp::Emp()

{

    id = ++idGenerator;

    name = "John Doe";

    deptId = 1;

    basicSalary = 20000;

}

Emp::Emp(string name, int deptId, double basicSalary)

{

    this->id = ++idGenerator;

    this->name = name;

    this->deptId = deptId;

    this->basicSalary = basicSalary;

}

// methods

double Emp::computeNetSalary()

{

    return this->basicSalary;

}

string Emp::getName()

{

    return this->name;

}

`

**Mgr.cpp – Implementation of Emp class**

`

// conatins implementations of Mgr class from Employees.h

#include <iostream>

#include "Employees.h"

using namespace std;

// constrs

Mgr::Mgr()

{

    // Default perf bonus

    perfBonus = 5000;

}

Mgr::Mgr(string name, int deptId, double basicSalary, double perfBonus) : Emp(name, deptId, basicSalary)

{

    this->perfBonus = perfBonus;

}

// methods

double Mgr::computeNetSalary()

{

    return basicSalary + perfBonus;

}

**`**

**Worker.cpp – Implementation of Emp class**

`

// conatins implementations of Worker class from Employees.h

#include <iostream>

#include "Employees.h"

using namespace std;

// constrs

Worker ::Worker()

{

    // Default values

    hourlyRate = 50.0;

    hoursWorked = 4;

}

Worker ::Worker(string name, int deptId, double basicSalary, int hoursWorked, float hourlyRate) : Emp(name, deptId, basicSalary)

{

    this->hourlyRate = hourlyRate;

    this->hoursWorked = hoursWorked;

}

// methods

double Worker::computeNetSalary()

{

    double totSalary = (hourlyRate \* hoursWorked) + this->basicSalary;

    return totSalary;

}

`

**Test.cpp – Test file**

`

#include <iostream>

#include "Employees.h"

#include <typeinfo.h>

using namespace std;

int main()

{

    cout << "-------Welcome to Organization-------" << endl;

    Emp \*arr[20];

    int ch, index = 0;

    cout << "1.Hire manager\n2.Hire Worker\n3.Display all employees' details\n4.Exit" << endl;

    do

    {

        cout << "Enter choice ";

        cin >> ch;

        switch (ch)

        {

        case 1:

        {

            cout << "-------Hire manager-------" << endl;

            int deptId;

            string name;

            double basicSalary, perfBonus;

            cout << "Enter name: ";

            cin >> name;

            cout << "Enter deptId of the dept for manager: ";

            cin >> deptId;

            cout << "Enter basic Salary: ";

            cin >> basicSalary;

            cout << "Enter performance bonus: ";

            cin >> perfBonus;

            Mgr \*manager = new Mgr(name, deptId, basicSalary, perfBonus); // ptr of child class Mgr

            arr[index] = manager;                                         // upcasting implicit

            index++;

            break;

        }

        case 2:

        {

            cout << "-------Hire Worker-------" << endl;

            int deptId, hoursWorked;

            string name;

            double basicSalary;

            float hourlyrate;

            cout << "Enter name: ";

            cin >> name;

            cout << "Enter deptId of dept of manager: ";

            cin >> deptId;

            cout << "Enter basic Salary: ";

            cin >> basicSalary;

            cout << "Enter hourlyRate  ";

            cin >> hourlyrate;

            cout << "Enter hours Worked  ";

            cin >> hoursWorked;

            Worker \*worker = new Worker(name, deptId, basicSalary, hoursWorked, hourlyrate); // ptr of child class Worker

            arr[index] = worker;

            index++;

            break;

        }

        case 3:

            // Here we've to do dynamic downcasting

            for (int i = 0; i < index; i++)

            {

                // find type of ref stored at runtime using typeid()

                if (typeid(\*arr[i]) == typeid(Mgr))

                {

                    // downcast basePtr to childPtr-Mgr

                    Mgr \*mgrPtr = dynamic\_cast<Mgr \*>(arr[i]);

                    cout << "Manager Name: " << mgrPtr->getName() << endl;

                    cout << "Salary is: " << mgrPtr->computeNetSalary() << endl;

                }

                if (typeid(\*arr[i]) == typeid(Worker))

                {

                    // downcast basePtr to childPtr-Worker

                    Worker \*workerPtr = dynamic\_cast<Worker \*>(arr[i]);

                    cout << "Worker Name: " << workerPtr->getName() << endl;

                    cout << "Salary is: " << workerPtr->computeNetSalary() << endl;

                }

            }

            break;

        default:

            break;

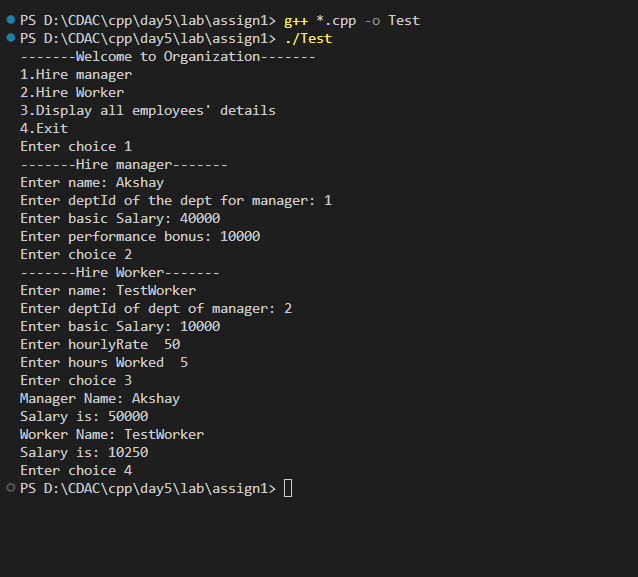
        }

    } while (ch != 4);

}

`

**Output:**



1. Create cpp application for bank account handling.

2.1. Create a class BankAccount -- acct no(int),customer name(string),balance(double)

Add constr. (2 constrs : first to accept all details )

2.2 Add Business logic methods

Methods

public void withdraw(double amt)

public void deposit(double amt)

2.3: Create object of account class and test withdraw and deposit methods.

**code:**

MyBankAccount class

#include <iostream>

using namespace std;

class MyBankAccount

{

private:

    int accountNo;

    string name;

    double balance;

    static int accountCount;

public:

    // default constr

    MyBankAccount()

    {

        accountNo = ++accountCount;

        name = "John Doe";

        balance = 0;

        cout << "Sample account created for " << this->name << " !" << endl;

    }

    // Parameterized

    MyBankAccount(string name)

    {

        this->accountNo = ++accountCount;

        this->name = name;

        this->balance = 0; // initial balance should be 0

        cout << "Account created for " << this->name << " !" << endl;

    }

    // getters and setters

    double getBalance()

    {

        return this->balance;

    }

    // methods

    void withdraw(double amt)

    {

        if (amt > this->balance)

            cout << "Not sufficient balance!!" << endl;

        else

        {

            this->balance -= amt;

            cout << "Amount debited successfully!" << endl;

        }

    }

    void deposit(double amt)

    {

        this->balance += amt;

        cout << "Amount credited successfully!" << endl;

    }

    void display()

    {

        cout << "Acc " << this->accountNo << " Name: " << this->name << " balance: " << this->balance << endl;

        ;

    }

};

int MyBankAccount::accountCount = 1000;

Test.cpp

#include <iostream>

using namespace std;

#include "BankAccount.cpp"

int main()

{

    string name;

    cout << "----Bank Account----" << endl;

    cout << "Enter Name of acoount holder: ";

    cin >> name;

    MyBankAccount acc(name);

    acc.display();

    acc.deposit(20000);

    cout << "Current Balance: Rs." << acc.getBalance() << endl;

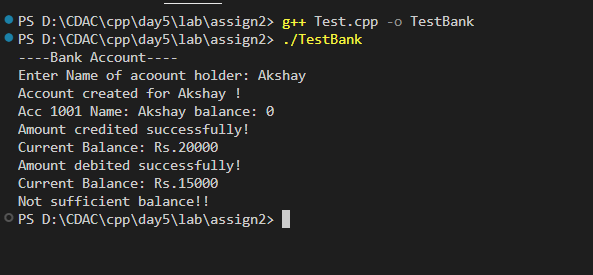
    acc.withdraw(5000);

    cout << "Current Balance: Rs." << acc.getBalance() << endl;

    acc.withdraw(20000);

}

**output:**

****