**Day 05**

/\*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

----------------------------------------------------\*/

#include <iostream>

using namespace std;

class Employee

{

protected :

int empId;

private:

string name ;

int depId;

static int counter;

protected :

double basicSalary;

public:

Employee(){

empId = 110;

name = "Employee";

depId = 11;

basicSalary = 10000;

counter ++;

}

Employee(int empId, string name, int depId, double basicSalary) : empId(empId), name(name), depId(depId), basicSalary(basicSalary){

counter ++;

}

static void totalEmployees(){

cout<<"Total Number of Employee is "<<counter;

}

virtual void display(){

cout<<"Emp Id : "<<empId<<"\nName : "<<name<<"\nDept Id : "<<depId<<"\nBasic Salary : "<<basicSalary<<endl;

}

virtual double computeNetSalary(){

return basicSalary;

}

int getEmpId(){

return empId;

}

~Employee(){

cout<<"\nI am Destructor"<<endl;

}

};

int Employee :: counter = 0;

class Manager : public Employee {

private :

double performanceBonus;

public :

Manager(){

performanceBonus = 1000;

}

Manager(int empId, string name, int deptId, double basicSalary, double performanceBonus) : Employee(empId, name, deptId, basicSalary) , performanceBonus(performanceBonus){

}

void display(){

Employee :: display();

cout<<"Perfomance Bonus = "<<performanceBonus<<endl;

}

double computeNetSalary(){

return basicSalary + performanceBonus;

}

void mangerTasks(){

cout<<"======Manager Tasks List======"<<endl;

}

void testOfEmployee(){

cout<<"Manager Pass manager related exam "<<endl;

}

};

class Worker : public Employee {

private :

int hoursWorked, hourlyRate;

public :

Worker(){

hoursWorked = 6;

hourlyRate = 5000;

}

Worker(int empId, string name, int deptId, double basicSalary, int hoursWorked,int hourlyRate) : Employee(empId, name, deptId, basicSalary) , hoursWorked(hoursWorked) ,hourlyRate(hourlyRate){

}

void display(){

Employee :: display();

cout<<"Hours Worked = "<<hoursWorked<<endl;

cout<<"Hourly Rate = "<<hourlyRate<<endl;

}

double computeNetSalary(){

return basicSalary + (hoursWorked\*hourlyRate);

}

void workerTasks(){

cout<<"======Worker Tasks List======"<<endl;

}

int getHourlyRate(){

return hourlyRate;

}

void testOfEmployee(){

cout<<"Worker Pass Worker related exam "<<endl;

}

};

void menu(){

cout<<"\n1. Hire Employee"<<endl;

cout<<"2. Hire Manager"<<endl;

cout<<"3. Hire Worker"<<endl;

cout<<"4. Display information of all employees net salary"<<endl;

cout<<"5. Show Employee specific Task"<<endl;

cout<<"6. Search Employee by id"<<endl;

cout<<"7. Net Salary greater than specific amount"<<endl;

cout<<"8. Copy Constructor (Make copy of deatils)"<<endl;

cout<<"9. Exit\n"<<endl;

cout<<"Enter your choice : ";

}

int main(){

int ch, maxSize = 100;

Employee\* Employees[maxSize];

int empId, id, indexOfEmployee;

string name;

int deptId ;

double basicSalary, performanceBonus, specificAmount;

int hoursWorked, hourlyRate;

int empCount = 0;

bool found ;

do

{

menu();

cin>>ch;

switch (ch)

{

case 1 :

cout<<"--------Input all Employee details-------"<<endl;

cout<<"empId name deptId basicSalary "<<endl;

cin>>empId>>name>>deptId>>basicSalary;

Employees [empCount ++] = new Employee(empId, name, deptId, basicSalary);

cout<<"--------------------------------"<<endl;

break;

case 2 :

cout<<"--------Input all manager details-------"<<endl;

cout<<"empId name deptId basicSalary performanceBonus"<<endl;

cin>>empId>>name>>deptId>>basicSalary>>performanceBonus;

Employees [empCount ++] = new Manager(empId, name, deptId, basicSalary, performanceBonus);

cout<<"--------------------------------"<<endl;

break;

case 3 :

cout<<"--------Input all Workers details-------"<<endl;

cout<<"empId name deptId basicSalary hoursWorked hourlyRate "<<endl;

cin>>empId>>name>>deptId>>basicSalary>>hoursWorked>>hourlyRate;

Employees [empCount ++] = new Worker(empId, name, deptId, basicSalary, hoursWorked,hourlyRate);

cout<<"--------------------------------"<<endl;

break;

case 4 :

cout << "------Employee Details--------" << endl;

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

{

Employees[i]->display();

cout<<"Salary of "<<typeid(\*Employees[i]).name()<<" : "<<Employees[i]->computeNetSalary()<<endl;

cout << "---------------------------" << endl;

}

cout << "------------------------------" << endl;

break;

case 5 :

cout<<"------Show Employee specific Task--------"<<endl;

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

{

if (typeid(\*Employees [i]) == typeid(Manager))

{

cout<<"Employee is "<<typeid(\*Employees[i]).name()<<". "<<endl;

cout<<"Manager type at idex:"<<i<<endl;

Manager \*m = dynamic\_cast<Manager\*>(Employees[i]);

m->mangerTasks();

// m->testOfEmployee();

}

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

{

cout<<"Employee is "<<typeid(\*Employees[i]).name()<<". "<<endl;

cout<<"Worker type at index : "<<i<<endl;

Worker \*w = dynamic\_cast <Worker\*> (Employees[i]);

w->workerTasks();

w->getHourlyRate();

// w->testOfEmployee();

}

cout << "---------------------------" << endl;

}

break;

case 6 :

cout << "Search by Employee Id : " << endl;

cout << "Enter Employee Id : ";

cin >> id;

found = false;

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

{

if (Employees[i]->getEmpId() == id )

{

found = true;

if (typeid(\*Employees[i]) == typeid(Manager))

{

cout<<"Employee is Manager and he is present"<<endl;

}else if(typeid(\*Employees[i]) == typeid(Worker)){

cout<<"Employee is Worker and he is present"<<endl;

}else if (typeid(\*Employees [i]) == typeid(Employee) )

{

cout<<"Employee is present"<<endl;

}

break;

}

}

if (! found)

{

cout<<"Employee with id : "<<id<<" is not present."<<endl;

}

break;

case 7 :

cout<<"Net salary greater than specific amount"<<endl;

cout<<"Enter specific amount : ";

cin>>specificAmount;

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

{

if (Employees[i]->computeNetSalary()>=specificAmount)

{

Employees[i]->display();

cout<<"Net Salay of " <<typeid(\*Employees[i]).name()<<" is " <<Employees[i]->computeNetSalary()<<endl;

}

}

break;

case 8 :

cout<<"Copy Constructor (Make copy of deatils)"<<endl;

if (empCount > 0)

{

cout<<"Enter the index of the employee to copy :\nWhere index star from 0 and ends at "<<empCount - 1<<" : ";

cin >> indexOfEmployee;

if (indexOfEmployee < empCount && indexOfEmployee >= 0)

{

if (typeid(\*Employees[indexOfEmployee]) == typeid(Manager))

{

Employees [empCount ++] = new Manager(\*dynamic\_cast<Manager\*>(Employees[indexOfEmployee]));

cout << "Manager copied successfully." << endl;

}

else if(typeid(\*Employees[indexOfEmployee]) == typeid(Worker)){

Employees [empCount ++ ] = new Worker (\*dynamic\_cast <Worker\*>(Employees[indexOfEmployee]));

cout << "Worker copied successfully." << endl;

}

else{

Employees [empCount ++] = new Employee(\*Employees[indexOfEmployee]);

cout << "Employee copied successfully." << endl;

}

}else{

cout<<"Invalid Index"<<endl;

}

}else{

cout << "No employee to copy."<<endl;

}

break;

case 9 :

cout<<"Exiting...."<<endl;

break;

default:

cout<<"Please enter correct choice"<<endl;

break;

}

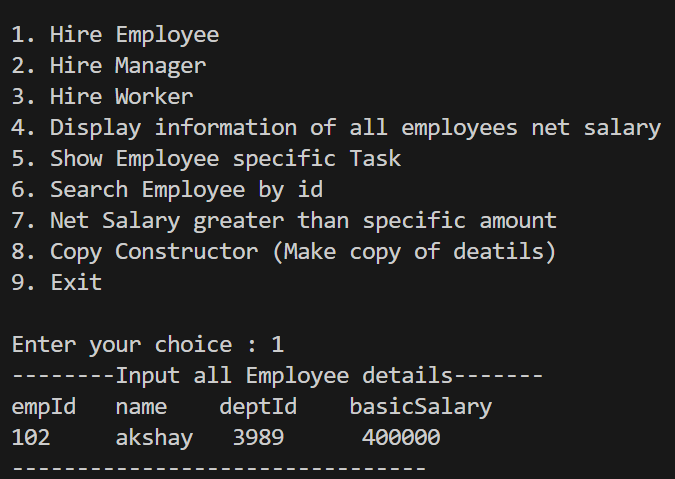
} while (ch != 9);

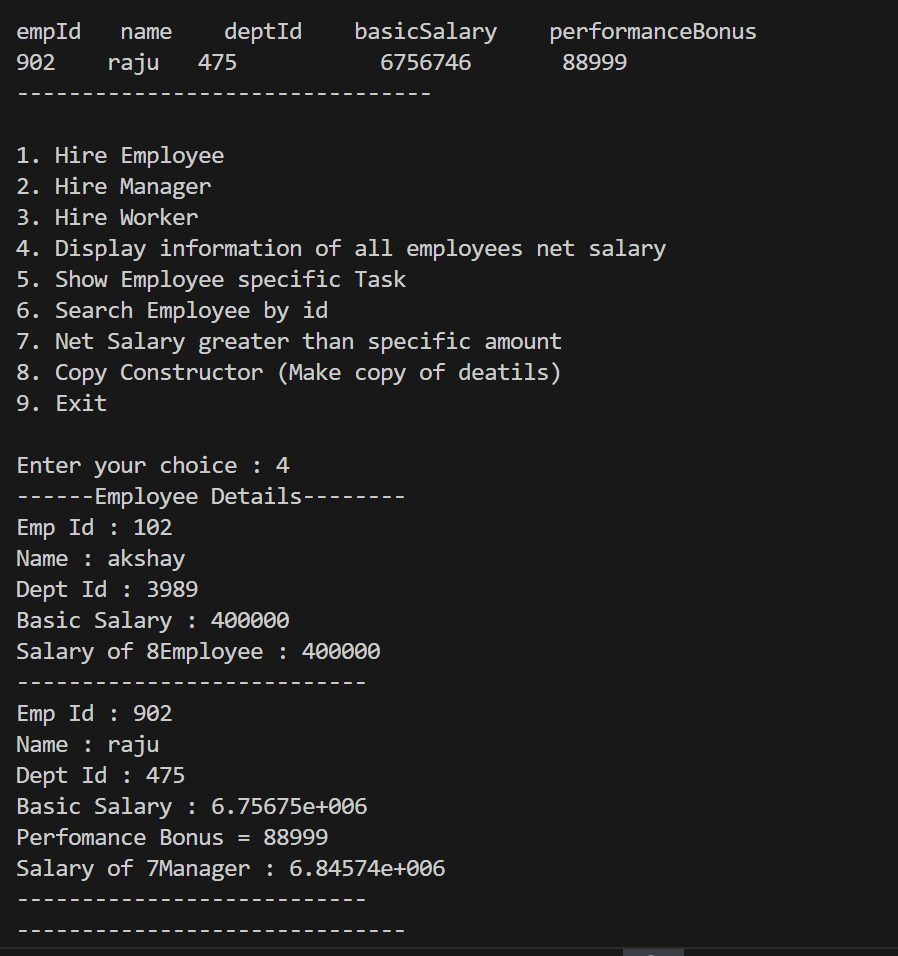
Employee :: totalEmployees();

delete Employees[empCount];

return 0;

}





/\*

2: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.\*/

#include <iostream>

#include <string>

using namespace std;

class BankAccount {

private:

int acctNo;

string customerName;

double balance;

public:

BankAccount(int acctNo, string customerName, double balance) {

this->acctNo = acctNo;

this->customerName = customerName;

this->balance = balance;

}

void withdraw(double amt) {

if (amt > 0 && amt <= balance) {

balance -= amt;

cout << "Withdrawal successful. Current balance: " << balance << endl;

} else {

cout << "Invalid withdrawal amount or insufficient balance." << endl;

}

}

void deposit(double amt) {

if (amt > 0) {

balance += amt;

cout << "Deposit successful. Current balance: " << balance << endl;

} else {

cout << "Invalid deposit amount." << endl;

}

}

void display() {

cout << "Account Number: " << acctNo << endl;

cout << "Customer Name: " << customerName << endl;

cout << "Balance: " << balance << endl;

}

};

int main() {

BankAccount account(123456, "John Doe", 5000.0);

cout << "Initial Account Details:" << endl;

account.display();

cout << "\nWithdrawing 2000:" << endl;

account.withdraw(2000);

cout << "\nDepositing 3000:" << endl;

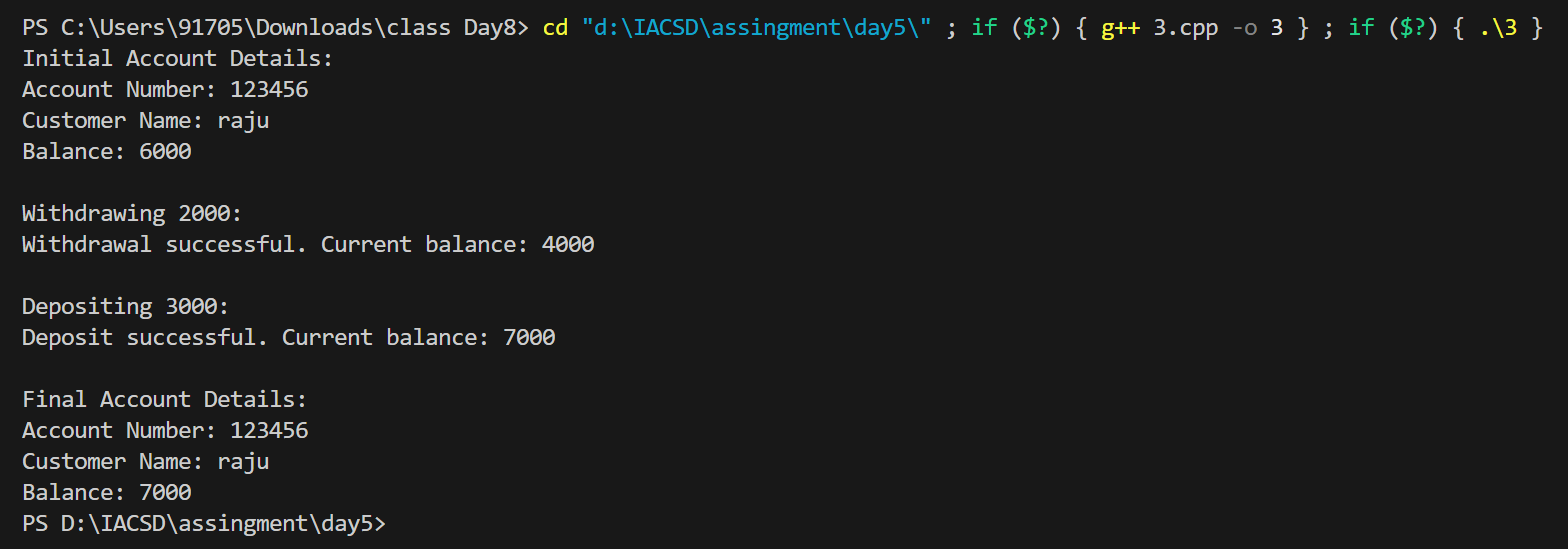
account.deposit(3000);

cout << "\nFinal Account Details:" << endl;

account.display();

return 0;

}



/\*-----------------------------------------------------------------------------------------------------------------------\*/

/\*3:Create a abstract class Shape with pure virtual method area;

Create Rectangle,Circle,Square class..inherit them from Shape class..Override area method.

Test these all classes by creating object of respective class.\*/

#include <iostream>

using namespace std;

class Shape {

public:

virtual double area() = 0;

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

Rectangle(double l, double w) : length(l), width(w) {}

double area(){

return length \* width;

}

};

class Circle : public Shape {

private:

double radius;

public:

Circle(double r) : radius(r) {}

double area() {

return 3.14 \* radius \* radius;

}

};

class Square : public Shape {

private:

double side;

public:

Square(double s) : side(s) {}

double area() {

return side \* side;

}

};

int main() {

Rectangle rectangle(5.0, 3.0);

Circle circle(2.5);

Square square(4.0);

cout << "Area of Rectangle: " << rectangle.area() << endl;

cout << "Area of Circle: " << circle.area() << endl;

cout << "Area of Square: " << square.area() << endl;

return 0;

}

