**LAB INHERITANCE**

**1.Animal Hierarchy:**

**Problem Statement: Create a hierarchy of animal classes.**

**Start with a base class Animal and then create derived classes like Mammal, Bird, and Fish.**

**Each of these derived classes should have specific properties and methods related to their respective categories of animals.**

#include<iostream>

using namespace std;

class ANIMAL

{

protected:

string name,habitat;

public:

ANIMAL(string name,string habitat )

{

this->name=name;

this->habitat=habitat;

}

virtual void speak()

{

cout<<"Animals make different sounds."<<endl;

}

virtual void move()

{

cout<<"Animals move in different ways."<<endl;

}

};

class MAMMAL:public ANIMAL

{

private:

string furcolour;

public:

MAMMAL(string name,string habitat,string furcolour):ANIMAL(name,habitat)

{

this->name=name;

this->habitat=habitat;

this->furcolour=furcolour;

}

void speak()override

{

cout<<"\nMammals make various sound."<<endl;

}

void move()override

{

cout<<"\nMammals walk or run."<<endl;

}

void showdetails()

{

cout<<"\n"<<name<<" have "<<furcolour<<" furcolour and lives in "<<habitat<<endl;

}

};

class BIRD:public ANIMAL

{

private:

string wingspan;

public:

BIRD(string name,string habitat,string wingspan):ANIMAL(name,habitat)

{

this->name=name;

this->habitat=habitat;

this->wingspan=wingspan;

}

void speak()override

{

cout<<"\nBirds chirp or sing."<<endl;

}

void move()override

{

cout<<"\nBirds fly."<<endl;

}

void showdetails()

{

cout<<"\n"<<name<<" have "<<wingspan<<" wingspan and lives in "<<habitat<<endl;

}

};

class FISH:public ANIMAL

{ private:

string bloodtype;

public:

FISH(string name,string habitat,string bloodtype):ANIMAL(name,habitat)

{

this->name=name;

this->habitat=habitat;

this->bloodtype=bloodtype;

}

void speak()override

{

cout<<"\nFish do not make sound."<<endl;

}

void move()override

{

cout<<"\nFish swims."<<endl;

}

void showdetails()

{

cout<<"\n"<<name<<" is "<<bloodtype<<" and lives in "<<habitat<<endl;

}

};

int main()

{

MAMMAL bear("Bear","land","Black");

FISH shark("Shark","water","cold blooded");

BIRD eagle("Eagle","air","large");

bear.move();

shark.move();

eagle.move();

bear.speak();

shark.speak();

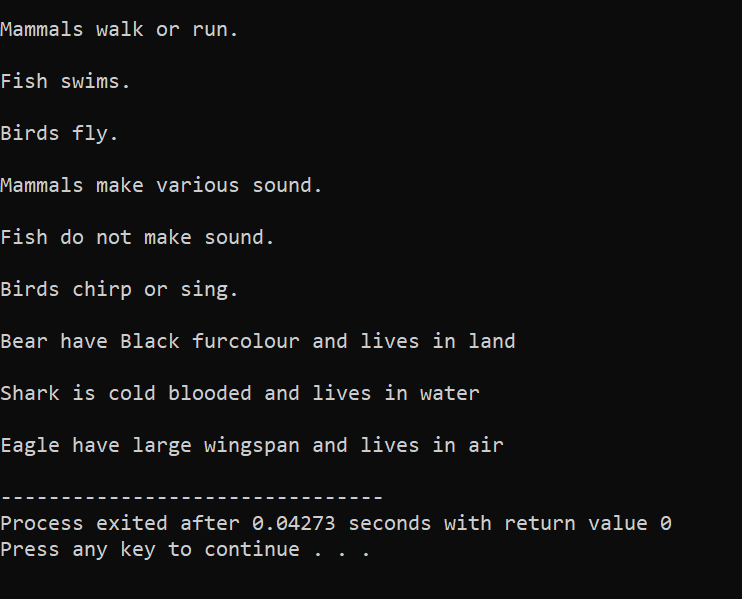
eagle.speak();

bear.showdetails();

shark.showdetails();

eagle.showdetails();

}



**2.Shape Hierarchy:**

**Problem Statement: Design a hierarchy of shape classes.**

**Begin with a base class Shape and then create derived classes like Circle, Rectangle, and Triangle.**

**Each shape should have methods for calculating area and perimeter specific to its geometry.**

#include<iostream>

#include<math.h>

using namespace std;

class SHAPE

{ public:

double area()

{

}

double perimeter()

{

}

};

class CIRCLE:public SHAPE

{

private:

int r;

public:

CIRCLE(double r)

{

this->r=r;

}

double area()override

{

return 3.14\*r\*r;

}

double perimeter()override

{

return 2\*3.14\*r;

}

};

class TRIANGLE:public SHAPE

{

private:

double s1,s2,s3,sp,ar;

public:

TRIANGLE(double s1,double s2,double s3)

{

this->s1=s1;

this->s2=s2;

this->s3=s3;

}

double area()

{

sp=(s1+s2+s3)/2;

ar=sqrt(sp\*(sp-s1)\*(sp-s2)\*(sp-s3));

return ar;

}

double perimeter()

{

return s1+s2+s3;

}

};

class RECTANGLE:public SHAPE

{

private:

double l,b;

public:

RECTANGLE(double l,double b)

{

this->l=l;

this->b=b;

}

double area()override

{

return l\*b;

}

double perimeter()override

{

return 2\*(l+b);

}

};

int main()

{

CIRCLE c(3.0);

TRIANGLE t(2.0,3.0,4.0);

RECTANGLE r(2.0,4.0);

cout<<"\nArea of Circle = "<<c.area()<<"\nPerimeter of Circle = "<<c.perimeter()<<endl;

cout<<"\nArea of Triangle = "<<t.area()<<"\nPerimeter of Triangle = "<<t.perimeter()<<endl;

cout<<"\nArea of Reactangle = "<<r.area()<<"\nPerimeter of Rectangle = "<<r.perimeter()<<endl;

}

**3.Employee Inheritance:**

**Problem Statement: Build a system for managing employees.**

**Create a base class Employee with attributes such as name, employee ID, and salary.**

**Then, derive classes like Manager and Developer, each with its own attributes and methods.**

**Implement a common method, like calculate\_salary(), in the base class.**

#include<iostream>

using namespace std;

class EMPLOYEE

{

protected:

string name;

int empid;

double salary;

public:

EMPLOYEE(string name,int empid,double salary)

{

this->empid=empid;

this->salary=salary;

this->name=name;

}

virtual double calsal()=0;

virtual void display\_info()

{

cout<<"\nEmployee Id = "<<empid<<"\nName = "<<name<<"\nSalary = "<<salary<<endl;

}

};

class MANAGER:public EMPLOYEE

{

private:

double bonus;

public:

MANAGER(string name,int empid,double salary,double bonus):EMPLOYEE(name,empid,salary)

{

this->name=name;

this->empid=empid;

this->salary=salary;

this->bonus=bonus;

}

double calsal()

{

return salary+bonus;

}

void display\_info()

{

cout<<"\nEmployee Id = "<<empid<<"\nName = "<<name<<"\nBasic Salary = "<<salary<<"\nNet Salary "<<calsal()<<endl;

}

void monitoring()

{

cout<<"Manager do monitoring."<<endl;

}

};

class DEVELOPER:public EMPLOYEE

{

private:

string language;

public:

DEVELOPER(string name,int empid,double salary,string language):EMPLOYEE(name,empid,salary)

{

this->name=name;

this->empid=empid;

this->salary=salary;

this->language=language;

}

double calsal()

{

return salary;

}

void display\_info()

{

cout<<"\nEmployee Id = "<<empid<<"\nName = "<<name<<"\nSalary = "<<calsal()<<"\nProgramming Language = "<<language<<endl;

}

void development()

{

cout<<"Developer do development"<<endl;

}

};

int main()

{

DEVELOPER d("HARSH",122,5000.0,"C++");

MANAGER m("HK",321,10000.0,1000.0);

d.calsal();

d.display\_info();

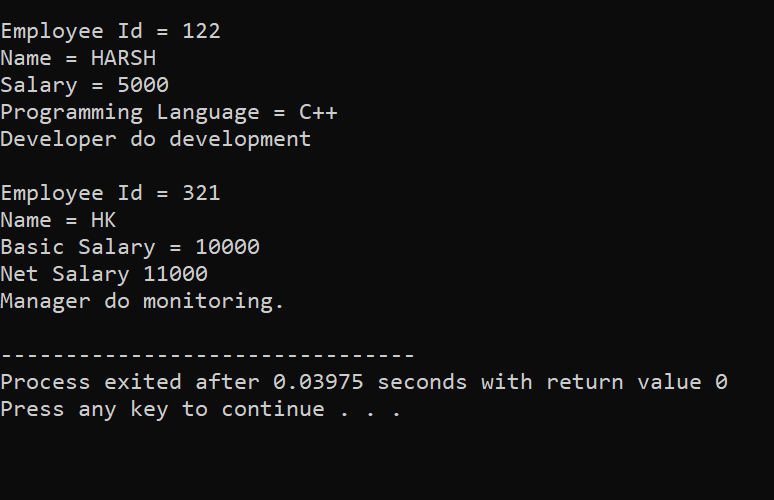
d.development();

m.calsal();

m.display\_info();

m.monitoring();

}



**4.Vehicle Inheritance:**

**Problem Statement: Develop a class hierarchy for vehicles.**

**Start with a base class Vehicle and create derived classes like Car, Motorcycle, and Truck.**

**Each derived class should have unique properties like the number of wheels and specific methods like start\_engine().**

#include<iostream>

using namespace std;

class VEHICLE

{

protected:

string fuel\_type,name;

public:

VEHICLE(string name,string fuel\_type)

{

this->fuel\_type=fuel\_type;

this->name=name;

}

virtual void start\_engine()

{

cout<<"Starting vehicles engine"<<endl;

}

virtual void display\_info()=0;

};

class CAR:public VEHICLE

{

protected:

string transmission;

public:

CAR(string name,string fuel\_type,string transmission):VEHICLE(name,fuel\_type)

{

this->name=name;

this->fuel\_type=fuel\_type;

this->transmission=transmission;

}

void start\_engine() override

{

cout<<"Starting car's engine"<<endl;

}

void airbag()

{

cout<<"Car has 4 airbags "<<endl;

}

void display\_info()

{

cout<<name<<" runs on "<<fuel\_type<<" and have "<<transmission<<endl;

}

};

class MOTORCYCLE:public VEHICLE

{

protected:

int num\_wheels;

public:

MOTORCYCLE(string name,string fuel\_type,int num\_wheels):VEHICLE(name,fuel\_type)

{

this->name=name;

this->fuel\_type=fuel\_type;

this->num\_wheels=num\_wheels;

}

void start\_engine

()override

{

cout<<"Starting motorcycle engine"<<endl;

}

void riding\_style()

{

cout<<"Motorcycle has unique riding style "<<endl;

}

void display\_info()

{

cout<<name<<" runs on "<<fuel\_type<<" and have " <<num\_wheels<<" wheels "<<endl;

}

};

class TRUCK: public VEHICLE

{

private:

string cargo\_capacity;

public:

TRUCK(string name,string fuel\_type,string cargo\_capacity):VEHICLE(name,fuel\_type)

{

this->cargo\_capacity=cargo\_capacity;

this->name=name;

this->fuel\_type=fuel\_type;

}

void start\_engine()override

{

cout<<"Starting truck engine"<<endl;

}

void cargo\_\_capacity()

{

cout<<"Trucks has large cargo capacity"<<endl;

}

void display\_info()override

{

cout<<name<<" run on "<<fuel\_type<<" and have "<<cargo\_capacity<<" cargo capacity "<<endl;

}

};

int main()

{

CAR c("Santro","petrol","manual");

MOTORCYCLE m("Bullet","petrol",2);

TRUCK t("Enndeavour","petrol","large");

c.start\_engine();

m.start\_engine();

t.start\_engine();

cout<<endl;

c.airbag();

m.riding\_style();

t.cargo\_\_capacity();

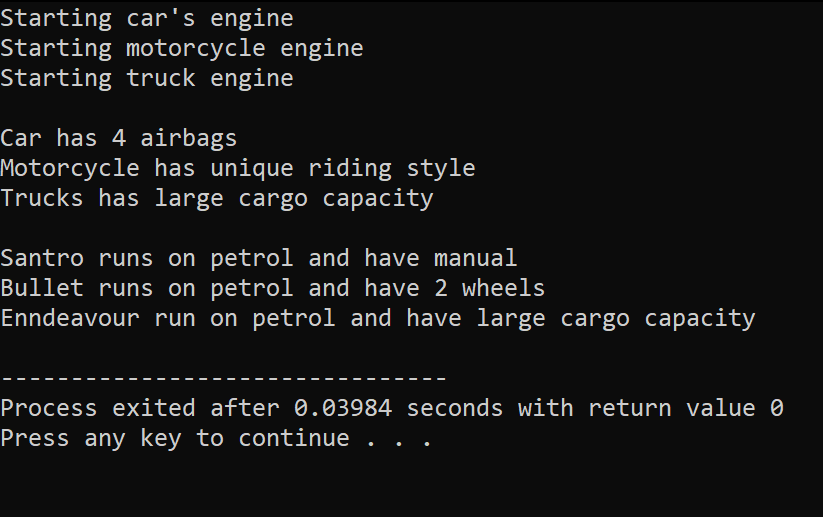
cout<<endl;

c.display\_info();

m.display\_info();

t.display\_info();

}



**5.Bank Account Inheritance:**

**Problem Statement: Design a system for managing bank accounts.**

**Create a base class BankAccount with attributes like account number and balance.**

**Derive classes like SavingsAccount and CheckingAccount, each with specialized methods like withdraw() and calculate\_interest().**

#include<iostream>

using namespace std;

class BANK\_ACCOUNT

{

protected:

string ac\_no,name;

double balance=10000;

public:

BANK\_ACCOUNT(string ac\_no,string name)

{

this->ac\_no=ac\_no;

this->name=name;

}

virtual void deposit(double amount)

{

if (amount>0)

{

balance=balance+amount;

cout<<"\nDeposit Succesfully!!!"<<endl;

}

else

{

cout<<"\nEnter valid amount!!!"<<endl;

}

}

virtual void withdraw(double amount)

{

if(amount>0&&amount<=balance)

{

balance=balance-amount;

cout<<"\nWithdraw Successfully!!!"<<endl;

}

else

cout<<"\nEnter valid amount!!!"<<endl;

}

virtual void display()

{

cout<<"Account Number : "<<ac\_no<<"\nBalance : "<<balance<<endl;

}

virtual void cal\_int() =0;

};

class SAVING\_ACCOUNT:public BANK\_ACCOUNT

{

private:

double interest\_rate=5.0,interest,time;

public:

SAVING\_ACCOUNT(string ac\_no,string name,double time):BANK\_ACCOUNT(ac\_no,name)

{

this->ac\_no=ac\_no;

this->name=name;

this->time=time;

}

void cal\_int()

{

interest=(balance\*interest\_rate\*time)/100;

balance=balance+interest;

cout<<"\nInterest of "<<interest<<" is added to account number "<<ac\_no<<endl;

}

void display()

{

cout<<"\nAccount Number : "<<ac\_no<<"\nBalance : "<<balance<<endl;

}

};

class CHECKING\_ACCOUNT:public BANK\_ACCOUNT

{

private:

double overdraft\_limit;

public:

CHECKING\_ACCOUNT(string ac\_no,string name,double time):BANK\_ACCOUNT(ac\_no,name)

{

this->ac\_no=ac\_no;

this->name=name;

}

virtual void withdraw(double amount)

{

if(amount>0&&amount<=balance+overdraft\_limit)

{

balance=balance-amount;

cout<<"\nWithdraw Successfully!!!"<<endl;

}

else

cout<<"\nEnter valid amount!!!"<<endl;

}

void display()

{

cout<<"\nAccount Number : "<<ac\_no<<"\nBalance : "<<balance<<endl;

}

void cal\_int()

{

}

};

int main()

{

string ac\_no,name;

double overdraft\_limit,time;

SAVING\_ACCOUNT s1("HKT2303","HARSH",3.0);

CHECKING\_ACCOUNT c1("MKT1090","MARK",3000.0);

s1.display();

s1.deposit(5000.0);

s1.cal\_int();

s1.withdraw(2000);

s1.display();

cout<<"\n";

c1.display();

c1.deposit(5000);

c1.withdraw(2000);

c1.withdraw(20000); //when overdraft limit exist

c1.display();

}

**6.Geometric Shapes with Polymorphism:**

**Problem Statement: Extend the shape hierarchy example by implementing polymorphism.**

**Define a base class Shape with methods to calculate area and perimeter.**

**Then, create derived classes like Circle, Rectangle, and Triangle, each with its own implementation of these methods.**

#include<iostream>

#include<math.h>

using namespace std;

class SHAPE

{

virtual void area()=0;

virtual void perimeter()=0;

};

class CIRCLE:public SHAPE

{

private:

double rad;

public:

CIRCLE(double rad);

void area()

{

cout<<"\nArea of circle = "<<(22/7)\*rad\*rad;

}

void perimeter ()

{

cout<<"\nPerimeter of circle = "<<2\*(22/7)\*rad;

}

};

class RECTANGLE:public SHAPE

{

private:

double length,width;

public:

RECTANGLE(int length,int width);

void area()

{

cout<<"\nArea of rectangle = "<<length\*width;

}

void perimeter ()

{

cout<<"\nPerimeter of rectangle = "<<2\*(length+width);

}

};

class TRIANGLE:public SHAPE

{

private:

double side\_1,side\_2,side\_3,semi\_perimeter,area;

public:

TRIANGLE(int side\_1,int side\_2,int side\_3);

void area()

{

semi\_perimeter=side\_1+side\_2+side\_3;

area=sqrt(semi\_perimeter\*(semi\_perimeter-side\_1)\*(semi\_perimeter-side\_2)\*(semi\_perimeter-side\_3));

cout<<"\nArea of triangle = "<<area;

}

void perimeter ()

{

cout<<"\nPerimeter of triangle = "<<side\_1+side\_2+side\_3;

}

};

**7.Person and Student Inheritance:**

**Problem Statement: Model a system for handling individuals and students within an educational institution.**

**Create a base class Person with attributes like name and age.**

**Derive a Student class with additional attributes like student ID and GPA, inheriting the common attributes from the Person class.**

class PERSON

{

protected:

string name;

int age;

public:

PERSON(string name,int age)

{

this->name=name;

this->age=age;

}

void display()

{

cout<<"\nName - "<<name<<"Age - "<<age<<endl;

}

}:

class STUDENT:public PERSON

{

private:

string name; int age;

double gpa;

string std\_id;

public:

STUDENT(string name,int age,double gpa,string std\_id):(name,age)

{

this->gpa=gpa;

this->std\_id=std\_id;

}

void display()

{

cout<<"\nName - "<<name<<"Age - "<<age<<"Student Id - "<<std\_id<<"GPA - "<<grade<<endl;

}

};

int main()

{

string name,std\_id;

double gpa;

int age;

STUDENT s(name,age,sd\_id,gpa);

}

**8.Library Catalog with Books and Journals:**

**Problem Statement: Build a library catalog system.**

**Create a base class LibraryItem with properties like title and author.**

**Then, derive classes like Book and Journal, each with their unique properties.**

**Implement methods to check out and return items in the derived classes.**

**9.Shape Sorting with Interfaces:**

**Problem Statement: Implement a shape sorting program.**

**Define a base class Shape with properties like area and perimeter.**

**Create derived classes like Circle, Rectangle, and Triangle.**

**Implement an interface Sortable with a method to compare shapes by area.**

**Use this interface to sort a list of shapes.**

**10.Employee Payroll System with Abstract Classes:**

**Problem Statement: Design an employee payroll system.**

**Create an abstract class Employee with attributes like name and employee ID.**

**Derive concrete classes like HourlyEmployee and SalariedEmployee.**

**Define abstract methods for calculating pay in the base class and implement them in the derived classes.**

#include <iostream>

#include <string>

// Abstract base class Employee

class Employee {

public:

Employee(const std::string& name, int employeeId)

: name(name), employeeId(employeeId) {}

virtual double calculatePay() const = 0; // Abstract method for calculating pay

void displayDetails() const {

std::cout << "Name: " << name << std::endl;

std::cout << "Employee ID: " << employeeId << std::endl;

std::cout << "Pay: $" << calculatePay() << std::endl;

}

protected:

std::string name;

int employeeId;

};

// Derived class HourlyEmployee

class HourlyEmployee : public Employee {

public:

HourlyEmployee(const std::string& name, int employeeId, double hourlyRate, double hoursWorked)

: Employee(name, employeeId), hourlyRate(hourlyRate), hoursWorked(hoursWorked) {}

double calculatePay() const override {

return hourlyRate \* hoursWorked;

}

private:

double hourlyRate;

double hoursWorked;

};

// Derived class SalariedEmployee

class SalariedEmployee : public Employee {

public:

SalariedEmployee(const std::string& name, int employeeId, double salary)

: Employee(name, employeeId), salary(salary) {}

double calculatePay() const override {

return salary;

}

private:

double salary;

};

int main() {

// Create instances of HourlyEmployee and SalariedEmployee

HourlyEmployee hourlyEmployee("John Doe", 101, 15.0, 40.0);

SalariedEmployee salariedEmployee("Alice Smith", 102, 50000.0);

// Display details and calculate pay

std::cout << "Hourly Employee Details:" << std::endl;

hourlyEmployee.displayDetails();

std::cout << "\nSalaried Employee Details:" << std::endl;

salariedEmployee.displayDetails();

return 0;

}

