Create an abstract class PaymentMethod with a pure virtual function processPayment().

Derive classes CreditCardPayment, PayPalPayment, and CashPayment, implementing the processPayment() function differently for each payment method.

#include <iostream>

using namespace std;

// Abstract class

class PaymentMethod {

public:

// Pure virtual function

virtual void processPayment() = 0;

};

// Derived class for Credit Card Payment

class CreditCardPayment : public PaymentMethod {

public:

int p;

void processPayment() {

cout<<"enter credit card payment";

cin>>p;

cout << "Processing credit card payment." << p<<endl;

}

};

// Derived class for PayPal Payment

class PayPalPayment : public PaymentMethod {

public:

int pay;

void processPayment(){

cout<<"enter paypalpayment";

cin>>pay;

cout << "Processing PayPal payment."<<pay << endl;

}

};

// Derived class for Cash Payment

class CashPayment : public PaymentMethod {

public:

int cash;

void processPayment() {

cout<<"enter cashpayment";

cin>>cash;

cout << "Processing cash payment."<<cash << endl;

}

};

// Main function

int main() {

PaymentMethod\* payment;

// Credit card payment

CreditCardPayment creditCard;

payment = &creditCard;

payment->processPayment();

// PayPal payment

PayPalPayment payPal;

payment = &payPal;

payment->processPayment();

// Cash payment

CashPayment cash;

payment = &cash;

payment->processPayment();

return 0;

}

 Create an abstract base class Account that has a **pure virtual function** display\_balance(). The class should have a protected member variable to store the account balance.

 Implement two derived classes:

* SavingsAccount: This class should override the display\_balance() function to display the balance with a bonus amount added.
* CurrentAccount: This class should override the display\_balance() function to display the balance after deducting a maintenance fee.

 Use pointers of the base class Account to demonstrate **runtime polymorphism** and call the appropriate display\_balance() function for each account type.

#include <iostream>

using namespace std;

// Base class

class Account {

protected:

double balance;

public:

void bal() { // Default constructor to initialize balance

balance = 1000; // Default balance

}

// Pure virtual function making Account an abstract class

virtual void display\_balance() = 0;

};

// Derived class for Savings Account

class SavingsAccount : public Account {

private:

double bonus = 200.0; // Bonus for savings account

public:

void display\_balance() {

bal();// Override pure virtual function

cout << "Savings Account Balance: " << (balance + bonus) << endl;

}

};

// Derived class for Current Account

class CurrentAccount : public Account {

private:

double maintenance\_fee = 50.0; // Maintenance fee for current account

public:

void display\_balance() {

bal();// Override pure virtual function

cout << "Current Account Balance: " << (balance - maintenance\_fee) << endl;

}

};

int main() {

Account \*accountPtr; // Pointer of type Account

// Create a SavingsAccount object and use pointer to call display\_balance

SavingsAccount savings;

accountPtr = &savings;

accountPtr->display\_balance(); // Calls SavingsAccount's display\_balance()

// Create a CurrentAccount object and use pointer to call display\_balance

CurrentAccount current;

accountPtr = &current;

accountPtr->display\_balance(); // Calls CurrentAccount's display\_balance()

return 0;

}

Create a base class named **Shape** that includes a pure virtual function called **calculateArea()**. This function will be responsible for calculating the area of the shapes.

You will derive two classes from the **Shape** class:

1. **Square**: Implement the **calculateArea()** function to compute the area of a square using the formula:

Area=dimension\*dimension

**Circle**: Implement the **calculateArea()** function to compute the area of a circle using the formula:

Area=π\*dimension\*dimension

#include<iostream>

using namespace std;

class Shape

{

protected:

float dimension;

public:

void getDimension()

{

cin >> dimension;

}

// pure virtual Function

virtual float calculateArea()=0;

};

class Square : public Shape

{

public:

float calculateArea()

{

return dimension \* dimension;

}

};

class Circle : public Shape

{

public:

float calculateArea()

{

return 3.14\*dimension\*dimension;

}

};

int main()

{

Shape \*s1;

Square s;

Circle c;

cout<<"enter dimension";

s.getDimension();

s1=&s;

cout<<"area of square is"<<s1->calculateArea();

s1=&c;

cout<<"area of cirle";

cout<<s1->calculateArea()<<endl;

return 0;

}

Write a C++ program that demonstrates **function overloading** by creating a class named Adder. The class should have the following overloaded functions named add:

1. **add(int a, int b)**: This function takes two integers as parameters and returns their sum.
2. **add(float a, float b)**: This function takes two floating-point numbers as parameters and returns their sum.
3. **add(int a, int b, int c)**: This function takes three integers as parameters and returns their sum.

In the main() function, create an instance of the Adder class and call each of the overloaded add() functions to demonstrate the concept of function overloading.

#include <iostream>

using namespace std;

class adder {

public:

// Function to add and display the sum of two integers

void add(int x, int y) {

cout << "Sum of x and y is " << x + y << endl;

}

// Function to add and display the sum of two doubles

void add(double x, double y) {

cout << "Sum of x and y is " << x + y << endl;

}

void add(int x, int y, int z) {

cout << "Sum of x and y is " << x + y + z << endl;

}

};

int main() {

adder obj1;

obj1.add(7,7);

// func() is called with double value

obj1.add(9.132,10.890);

// func() is called with 3 int values and adds them

obj1.add(10,10,10);

return 0;

}

Explain polymorphism with its types.

Explain Compile time and run time polymorphism with example.

Explain Function overloading with example

Explain virtual function with example

Explain abstract class with example