Classes:

Shape: Base class representing a generic shape.

Rectangle: Derived class representing a rectangle with length and width.

Circle: Derived class representing a circle with radius.

Concepts:

Constructors and Destructors:

Define a default constructor for Shape to initialize common properties.

Overload constructors for Rectangle and Circle to take specific dimensions as input during object creation.

Implement destructors for all classes to handle memory cleanup (if applicable).

Overriding:

Override the area() function in Rectangle and Circle to calculate their respective areas using appropriate formulas. The base class Shape can have a pure virtual area() function to enforce implementation in derived classes.

Operator Overloading:

Overload the == operator for Shape to compare shapes based on a chosen criterion (e.g., area for simplicity).

Consider overloading other operators (like +) for specific shapes if applicable (e.g., combining rectangles).

Friend Function:

Define a friend function totalArea outside the class hierarchy that takes an array of Shape pointers and calculates the total area of all shapes. This function needs access to private member variables of Shape and its derived classes.

Template (Optional):

(Optional) Create a template class Point to represent a point in 2D space with x and y coordinates. Use this template class within the Shape hierarchy if needed.

Implementation:

Design the Shape class with appropriate member variables and functions, including a pure virtual area() function.

Implement derived classes Rectangle and Circle with constructors, destructors, overridden area() functions, and potentially overloaded operators.

Define a friend function totalArea that takes an array of Shape pointers and calculates the total area.

(Optional) Implement a template class Point for representing points.

Testing:

Create objects of different shapes (rectangle, circle) and test their constructors, destructors, and overridden area() functions.

Use the overloaded == operator to compare shapes.

Call the totalArea friend function to calculate the total area of an array of shapes.

(Optional) Test the functionality of the Point template class (if implemented).

Answer

Overview:

The goal is to implement a base Shape class and derived classes Rectangle and Circle. Additionally, we will implement functionalities such as area calculation, operator overloading, and a function to calculate the total area of a collection of shapes.

Shape(Base class)

Methods:

Shape() - Default constructor

~Shape() - Virtual destructor

virtual double area() const = 0 - Pure virtual function to calculate area

bool operator==(const Shape& other) const - Overloaded equality operator to compare shapes by their area

friend double totalArea(const vector<Shape\*>& shapes) - Friend function to calculate the total area of a collection of shapes.

Retangle(Derived class)

Methods:

Rectangle(double l, double w) - Constructor

~Rectangle() - Destructor

double area() const override - Overridden function to calculate the area of the rectangle

Rectangle operator+(const Rectangle& other) const - Overloaded addition operator to combine rectangles.

Circle(Derived class)

Methods:

Circle(double r) - Constructor

~Circle() - Destructor

double area() const override - Overridden function to calculate the area of the circle.

Create the base class shape:

Define the class Shape with a pure virtual function area.

Implement the overloaded equality operator to compare shapes by their area.

Implement the friend function totalArea to calculate the total area of a collection of shapes.

Create the derived class rectangle:

Define the class Rectangle inheriting from Shape.

Implement the constructor to initialize length and width.

Override the area function to calculate the area of the rectangle.

Implement the overloaded addition operator to combine rectangles by adding their lengths and widths.

Create the derived class circle:

Define the class Circle inheriting from Shape.

Implement the constructor to initialize radius.

Override the area function to calculate the area of the circle.

The total area function:

Implement the totalArea function that takes a vector of Shape\* and returns the sum of their areas.

Code:

#include <iostream>

#include <cmath>

#include <vector>

using namespace std;

template<typename T>

class Point {

public:

T x, y;

Point(T x = 0, T y = 0) : x(x), y(y) {}

};

class Shape {

public:

Shape() {}

virtual ~Shape() {}

virtual double area() const = 0;

bool operator==(const Shape& other) const {

return this->area() == other.area();

}

friend double totalArea(const vector<Shape\*>& shapes);

};

class Rectangle : public Shape {

private:

double length, width;

public:

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

~Rectangle() {}

double area() const override {

return length \* width;

}

Rectangle operator+(const Rectangle& other) const {

return Rectangle(length + other.length, width + other.width);

}

};

class Circle : public Shape {

private:

double radius;

public:

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

~Circle() {}

double area() const override {

return M\_PI \* radius \* radius;

}

};

double totalArea(const vector<Shape\*>& shapes) {

double total = 0;

for (const Shape\* shape : shapes) {

total += shape->area();

}

return total;

}

int main() {

Rectangle rect1(24, 40);

Rectangle rect2(7, 13);

Circle circ1(19);

cout << "Rectangle 1 Area: " << rect1.area() << endl;

cout << "Rectangle 2 Area: " << rect2.area() << endl;

cout << "Circle 1 Area: " << circ1.area() << endl;

if (rect1 == rect2) {

cout << "Rectangle 1 is equal to Rectangle 2 based on area." << endl;

} else {

cout << "Rectangle 1 is not equal to Rectangle 2 based on area." << endl;

}

Rectangle rect3 = rect1 + rect2;

cout << "Combined Rectangle Area: " << rect3.area() << endl;

vector<Shape\*> shapes = { &rect1, &rect2, &circ1 };

cout << "Total Area of All Shapes: " << totalArea(shapes) << endl;

return 0;

}

Output:

