Assignment 1:
Object-Oriented Programming

Trace_it
ACA, IIT Kanpur

Objective

Design and implement a Shape Geometry Calculator using Object-Oriented Programming (OOP) concepts in C++. This system should demonstrate key OOP principles such as abstraction, inheritance, polymorphism, and encapsulation to model and calculate properties of various geometric shapes.

Introduction

In this assignment, you will implement a Shape Geometry Calculator in C++ using Object-Oriented Programming concepts. This system will:

- Model various 2D geometric shapes using an inheritance hierarchy
- Calculate geometric properties (area, perimeter etc.) using polymorphic methods
- Support shape transformations through operator overloading
- Provide a clean, user-friendly interface to interact with shapes
- Demonstrate proper application of OOP principles including inheritance, polymorphism, and encapsulation

The system will utilize an abstract base class to define the common interface for all shapes, with derived classes implementing shape-specific behaviors. This approach will showcase polymorphism as a powerful tool for extending functionality while maintaining a consistent interface.

1 Assignment Requirements

1.1 Class Design & Hierarchy

Design a comprehensive class hierarchy for a Shape Geometry Calculator system. Your design should include:

1. An abstract base `Shape` class
2. At least five derived 2D shape classes (e.g., `Circle`, `Rectangle`, `Triangle`)

For each class, specify:

- Member variables (with appropriate access modifiers)
- Member functions (including constructors, destructors, getters, setters)
- Pure virtual functions and their implementations in derived classes
- Inheritance relationships
- Any friend functions or classes
- Any operator overloads

Provide comments as explanatory notes justifying your design decisions and implementation.

1.2 Complete Implementation

Implement the complete Shape Geometry Calculator according to your design. Your implementation should include:

1.2.1 Core Shape Hierarchy

1. **Shape Base Class**
   - Create an abstract `Shape` base class with:
     - Common attributes shared by all shapes (e.g., color, ID)
     - Pure virtual methods:
       - `display()` - Print detailed information about the shape
       - `area()` - Calculate and return the shape's area
       - `perimeter()` - Calculate and return the shape's perimeter
       - `getType()` - Return a string indicating the shape type
     - Additional virtual methods as needed
     - Appropriate constructor(s) and virtual destructor
2. **2D Shape Classes**
   - Implement at least five derived 2D shape classes (e.g., `Circle`, `Rectangle`, `Square`, `Triangle`, `Polygon`):
     - Appropriate attributes specific to each shape
     - Override all virtual methods from the base class
     - Implement constructors and destructors
     - Implement additional methods specific to each shape (e.g., `getDiameter()` for `Circle`)

1.2.2 Advanced OOP Features

1. **Operator Overloading**

   - Implement at least four of the following operator overloads:
     - `+` or `+=` for scaling shapes (increasing dimensions)
     - `-` or `-=` for reducing shapes (decreasing dimensions)
     - `*` or `*=` for multiplication by scalar (uniform scaling)
     - `==` for comparing shapes
     - `<`, `>`, `<=`, or `>=` for comparing shapes by area or volume

2. **Composition**

   - Demonstrate composition by implementing a `ShapeCollection` class that can:
     - Store multiple shapes in a container (e.g., vector, list)
     - Calculate total area/volume of all shapes
     - Find shapes with specific properties (e.g., largest area, specific color)
     - Sort shapes based on various criteria (area, perimeter, etc.)

3. **Exception Handling and Validation**

   - Create custom exception classes for invalid shapes or operations
     - `InvalidDimensionException` (e.g., negative radius, zero length)
     - `ShapeOperationException` (e.g., invalid transformation)

1.2.3 User Interface

1. **Command-Line Interface (CLI)**

   - Implement a menu-driven interface with the following functions, The User should be able to perform the following operations using your menu, You can map Keys to options, press 1 to Create a shape etc.:
     - Create shapes of different types with user-specified dimensions
     - List all shapes with their properties
     - Search for shapes with specific properties
     - Calculate specific properties (area, perimeter, volume) for selected shapes
     - Perform operations on multiple shapes (e.g., calculate total area)

2 Implementation Guidelines

2.1 Core OOP Requirements

1. **Inheritance:**

   - Use inheritance to model the shape hierarchy
   - Ensure proper use of access specifiers (public, protected, private)
   - Implement constructors that properly initialize base class

2. **Polymorphism:**
    - Use virtual functions for operations that vary by shape type
    - Demonstrate runtime polymorphism by using base class pointers/references
    - Implement at least one pure virtual function in the base class
3. **Encapsulation:**
    - Use private member variables for shape attributes
    - Provide public getters and setters with appropriate validation
    - Protect the internal representation from direct external access
4. **Abstraction:**
    - Design an intuitive interface for shape operations
    - Hide implementation details where appropriate
    - Use abstract classes to define common behavior

2.2 Implementation Details

1. Create separate header (.h) and implementation (.cpp) files for each class
2. Organize your code into logical directories
3. Use consistent naming conventions and code style
4. Comment your code thoroughly, especially for complex algorithms
5. Implement proper error checking and validation

2.3 Sample Class Structure

Here's a sample structure to get you started:

```
// Shape.h
class Shape {
protected:
    string color;
    int id;

public:
    Shape(const string& color, int id);
    virtual ~Shape() = default;

    // Pure virtual functions
    virtual double area() const = 0;
    virtual double perimeter() const = 0;
    virtual string getType() const = 0;
    virtual void display() const = 0;

    // Common methods
    string getColor() const;
    void setColor(const string& color);
```

```
    int getId() const;
};

// Circle.h
class Circle : public Shape {
private:
    double radius;

public:
    Circle(const string& color, int id, double radius);
    ~Circle() override;

    // Override virtual functions
    double area() const override;
    double perimeter() const override;
    string getType() const override;
    void display() const override;

    // Circle-specific methods
    double getRadius() const;
    void setRadius(double radius);
    double getDiameter() const;

    // Operator overloads
    Circle& operator+=(double scalar);
    bool operator==(const Circle& other) const;
    friend ostream& operator<<(ostream& os, const Circle& circle);
};
```

3 Deliverables and Guidelines

1. **Source Code**

   - Make a ZIP file and add the following to it.
   - All `.h` and `.cpp` files properly organized.
   - README with compilation and usage instruction.
   - The ZIP file containing all the code must be submitted on this google form.

     https://forms.gle/ffFAQ92qBmar2Q3p8

   - Comments are very important, ambiguous code without comments and explanation will result in penalty.

Contact

Any queries regarding this assignment should be directed to Rahul Meena preferably on WhatsApp @7410808837 or at email 567rahulm567@gmail.com.