Lab 09 Tasks

Task 01

You are tasked with developing a software module for a geometry library that calculates the areas and perimeters of various shapes. Your module needs to efficiently handle circles, rectangles, and triangles. You've decided to implement this module in C++, **utilizing function overloading and a global constant for the value of pi.**

Your `Shape` class should have the following functionalities:

Overloaded methods to calculate the `area` of a circle given its `radius`, the `area` of a rectangle given its `length` and `width`, and the `area` of a triangle given its `base` and `height`.

Overloaded methods to calculate the `perimeter` of a circle given its `radius`, the `perimeter` of a rectangle given its `length` and `width`, and the `perimeter` of a triangle given its `three sides`.

Additionally, ensure that your code is well-commented and demonstrates good coding practices such as encapsulation, abstraction, and meaningful variable naming. Provide a `main` function to demonstrate the usage of your `Shape` class by calculating and displaying the `areas` and `perimeters` of various shapes, utilizing the **global constant for the value of pi.**

Task 02

Suppose you are tasked with developing a geometry application that helps users calculate properties of various shapes. You are required to implement classes for different shapes such as `Circle`, `Rectangle`, `Square`, `Triangle`, and `EquilateralTriangle`. Each shape class should inherit from a base class called `Shape`. Additionally, the `EquilateralTriangle` class should inherit from the `Triangle` class. Your implementation should utilize **method overriding** and **pointers to ensure polymorphic behavior.**

Implement the base class `Shape` with the following `virtual` functions:

`virtual double area() const`: This function should calculate and return the area of the shape. **Override this function in each derived class to provide the specific implementation for that shape.**

`virtual double perimeter() const`: This function should calculate and return the perimeter of the shape. **Override this function in each derived class to provide the specific implementation for that shape.**

`virtual void displayProperties() const`: This function should display the properties of the shape. **Override this function in each derived class to provide the specific display for that shape.**

Implement the derived classes `Circle`, `Rectangle`, `Square`, and `Triangle`, each inheriting from the base class `Shape`. For each derived class:

Override the `virtual` functions `area()`, `perimeter()`, and `displayProperties()` to provide the specific implementations for that shape.

Ensure to use pointers to the base class `Shape` when implementing the functions.

Implement the `EquilateralTriangle` class, inheriting from the `Triangle` class. For this class:

Override the `virtual` functions `area()`, `perimeter()`, and `displayProperties()` to provide the specific implementations for an equilateral triangle.

Ensure to use pointers to the base class `Triangle` when implementing the functions.

In the `main` function:

Allow users to select a shape and provide input for the required parameters (e.g., `radius` for `Circle`, `length` and `width` for `Rectangle`, etc.).

Dynamically allocate memory for the selected shape using pointers to the appropriate base class (`Shape` or `Triangle`).

Call the appropriate functions **using the base class pointers** to display the properties of the selected shape.

Ensure proper memory management by deallocating memory when it is no longer needed.

Approach:

1. Start by defining the `Shape` base class with the required `virtual` functions.
2. Implement derived classes (`Circle`, `Rectangle`, `Square`, `Triangle`) inheriting from the Shape class and override their respective `virtual` functions.
3. Implement the `EquilateralTriangle` class, inheriting from the `Triangle` class, and override its `virtual` functions.
4. **Use pointers to the appropriate base class (`Shape` or `Triangle`) to achieve polymorphic behavior.**
5. In the `main` function, prompt the user to select a shape and provide necessary input.
6. Dynamically allocate memory for the selected shape using pointers.
7. Call functions using the appropriate base class pointers to display shape properties.
8. Ensure proper memory deallocation to prevent memory leaks.

Sample Output:

```
Welcome to the Geometry Competition Calculator!

Please select a shape:
1. Circle
2. Rectangle
3. Square
4. Triangle
Enter your choice: 2

Enter the length of the rectangle: 10
```

```
 Enter the width of the rectangle: 15
 Properties of the Rectangle:
- Area: 150
- Perimeter: 50
- Diagonal: 18.0278

Do you want to calculate properties for another shape? (yes/no): yes

Please select a shape:
1. Circle
2. Rectangle
3. Square
4. Triangle
Enter your choice: 1

Enter the radius of the circle: 15
 Properties of the Circle:
- Area: 706.858
- Perimeter: 94.2478
- Diameter: 30

Do you want to calculate properties for another shape? (yes/no): yes

Please select a shape:
1. Circle
2. Rectangle
3. Square
4. Triangle
Enter your choice: 3

Enter the side length of the square: 12
 Properties of the Square:
- Area: 144
- Perimeter: 48
- Diagonal: 16.9706

Do you want to calculate properties for another shape? (yes/no): yes

Please select a shape:
1. Circle
2. Rectangle
3. Square
4. Triangle
Enter your choice: 4

Enter the lengths of the three sides of the triangle: 10
10
5
 Properties of the Triangle:
- Area: 24.2061
- Perimeter: 25
```

```
Do you want to calculate properties for another shape? (yes/no): no

Thank you for using the Geometry Competition Calculator!
```

Task 03

You are tasked with creating a payroll management system for a small company. The system needs to handle two types of employees: full-time employees and part-time employees. Each type of employee has different attributes and methods associated with them. Full-time employees receive a fixed salary per month, while part-time employees are paid based on the number of hours they work.

Define a base class named `Employee` with the following attributes and methods:

Attributes:

`employee ID`, `employee name`.

Methods:

`virtual double calculatePay() const`: This method calculates and returns the pay for the employee. Since this method's implementation depends on the type of employee, it should be declared as `virtual` and have a **default implementation (e.g., returning 0.0).**

`virtual void displayDetails() const`: This method displays the details of the employee (`ID` and `name`).

Implement two derived classes: `FullTimeEmployee` and `PartTimeEmployee`.

`FullTimeEmployee` should inherit from the `Employee` class and have an additional attribute for the monthly salary.

`PartTimeEmployee` should also inherit from the `Employee` class and have attributes for the hourly wage and the number of hours worked.

Override the `calculatePay()` method in both derived classes:

In `FullTimeEmployee`, calculate the pay by simply returning the monthly salary.

In `PartTimeEmployee`, calculate the pay by multiplying the hourly wage with the number of hours worked.

Override the `displayDetails()` method in both derived classes to display the additional attributes along with the base class attributes.

In the `main` function:

- Create instances of `FullTimeEmployee` and `PartTimeEmployee`.
- Call the `displayDetails()` method for each instance to display their details.
- Call the `calculatePay()` method for each instance to calculate and display their pay.

- Demonstrate early or static binding by calling the `calculatePay()` method using a base class pointer.

Approach:

1. Define the `Employee` base class with the required attributes and `virtual` methods.
2. Implement derived classes (`FullTimeEmployee` and `PartTimeEmployee`) inheriting from the `Employee` class and override their `virtual` methods.
3. Use dynamic polymorphism to handle different types of employees by using base class pointers.
4. In the `main` function, create instances of both derived classes and call their methods to demonstrate polymorphic behavior.
5. Use a base class pointer to call the `calculatePay()` method and observe early or static binding in action.

Sample Output:

```
Displaying Details:
Employee ID: 101, Name: AAA
Type: Full-time, Monthly Salary: 5000
Employee ID: 102, Name: BBB
Type: Part-time, Hourly Wage: 15, Hours Worked: 40

Calculating Pay:
Full-time Employee Pay: $5000
Part-time Employee Pay: $600

Using base class pointer to calculate Full-time Employee Pay: $5000
```