**2 Area of circle rectangle triangle**

#include <iostream>

#include <cmath> // For M\_PI constant

class Shape {

public:

// Pure virtual function for area calculation

virtual double getArea() const = 0;

// Pure virtual function for perimeter calculation

virtual double getPerimeter() const = 0;

};

// Abstract class for shapes with sides (can be extended for Polygons)

class ShapeWithSides : public Shape {

public:

virtual int getNumSides() const = 0;

};

class Circle : public Shape {

private:

double radius;

public:

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

double getArea() const override { return M\_PI \* radius \* radius; }

double getPerimeter() const override { return 2 \* M\_PI \* radius; }

};

class Rectangle : public ShapeWithSides {

private:

double width;

double height;

public:

Rectangle(double width, double height) : width(width), height(height) {}

double getArea() const override { return width \* height; }

double getPerimeter() const override { return 2 \* (width + height); }

int getNumSides() const override { return 4; }

};

class Triangle : public ShapeWithSides {

private:

// Triangle can have different types (equilateral, isosceles, etc.)

// For simplicity, assume basic triangle here

double base;

double height;

public:

Triangle(double base, double height) : base(base), height(height) {}

double getArea() const override { return 0.5 \* base \* height; }

double getPerimeter() const override {

// Needs implementation based on specific triangle type

return 0.0; // Placeholder for incomplete implementation

}

int getNumSides() const override { return 3; }

};

int main() {

Circle circle(5);

Rectangle rect(4, 6);

Triangle triangle(3, 4);

std::cout << "Circle Area: " << circle.getArea() << std::endl;

std::cout << "Rectangle Perimeter: " << rect.getPerimeter() << std::endl;

// Triangle perimeter needs specific type implementation

std::cout << "Triangle Area: " << triangle.getArea() << std::endl;

// Placeholder value for incomplete perimeter

std::cout << "Triangle Perimeter (incomplete): " << triangle.getPerimeter() << std::endl;

return 0;

}

**3.** **Design a class hierarchy to represent a university system. Include classes for students, professors, and courses. Use appropriate OOP principles like encapsulation, inheritance, and polymorphism. Provide a brief explanation of how your design promotes code reusability and flexibility in c++ program**

#include <iostream>

#include <vector>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

Person(const std::string& name, int age) : name(name), age(age) {}

virtual void display() const = 0; // Pure virtual function

};

class Student : public Person {

private:

int studentID;

public:

Student(const std::string& name, int age, int studentID) : Person(name, age), studentID(studentID) {}

void display() const override {

std::cout << "Student Name: " << name << ", Age: " << age << ", Student ID: " << studentID << std::endl;

}

};

class Professor : public Person {

private:

std::string department;

public:

Professor(const std::string& name, int age, const std::string& department) : Person(name, age), department(department) {}

void display() const override {

std::cout << "Professor Name: " << name << ", Age: " << age << ", Department: " << department << std::endl;

}

};

class Course {

private:

std::string courseName;

Professor\* professor;

std::vector<Student\*> students;

public:

Course(const std::string& courseName, Professor\* professor) : courseName(courseName), professor(professor) {}

void addStudent(Student\* student) {

students.push\_back(student);

}

void display() const {

std::cout << "Course Name: " << courseName << std::endl;

professor->display();

std::cout << "Students enrolled:" << std::endl;

for (const auto& student : students) {

student->display();

}

}

};

int main() {

Professor prof("John Doe", 40, "Computer Science");

Student student1("Alice", 20, 12345);

Student student2("Bob", 21, 54321);

Course cs101("Introduction to Programming", &prof);

cs101.addStudent(&student1);

cs101.addStudent(&student2);

cs101.display();

return 0;

}

5 .  **An organization they decide to give bonus to all the employees on New Year. A 5% bonus on salary is given to the grade A workers and 10% bonus on salary to the grade B workers. Write a program to enter the salary and grade of the employee. If the salary of the employee is less than $10,000 then the employee gets an extra 2% bonus on salary Calculate the bonus that has to be given to the employee and print the salary that the employee will get. Sample Input & Output: Enter the grade of the employee: B Enter the employee salary: 50000 Salary=50000 Bonus=5000.0 Total to be paid: 55000.0**

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

double calculateBonus(double salary, char grade) {

double bonusPercentage = 0.0;

if (salary < 10000) {

bonusPercentage += 0.02;

}

if (grade == 'A') {

bonusPercentage += 0.05;

} else if (grade == 'B') {

bonusPercentage += 0.10;

}

double bonus = salary \* bonusPercentage;

return bonus;

}

int main() {

char grade;

double salary;

cout << "Enter the grade of the employee: ";

cin >> grade;

grade = toupper(grade); // Convert to uppercase for uniformity

cout << "Enter the employee salary: ";

cin >> salary;

double bonus = calculateBonus(salary, grade);

double totalSalary = salary + bonus;

cout << fixed << setprecision(1);

cout << "Salary = " << salary << endl;

cout << "Bonus = " << bonus << endl;

cout << "Total to be paid: " << totalSalary << endl;

return 0;

}