ASSIGNMENT 1

GEETHIKA MN

192121108

1) Illustrate the following in two ways of object notations.

1. Employee
2. Teacher
3. Car

Identify the object of the following class and create object.

1. Company
2. Class
3. Vehicle

INPUT:

#include <iostream>

// Object notation 1: Separate object creation

class Employee {};

class Teacher {};

class Car {};

// Object notation 2: Object creation with class definition

class Company {} companyObj;

class Class {} classObj;

class Vehicle {} vehicleObj;

int main() {

// Displaying class names

std::cout << "Object notation 1:" << std::endl;

std::cout << "Employee" << std::endl;

std::cout << "Teacher" << std::endl;

std::cout << "Car" << std::endl;

std::cout << "Object notation 2:" << std::endl;

std::cout << "Company" << std::endl;

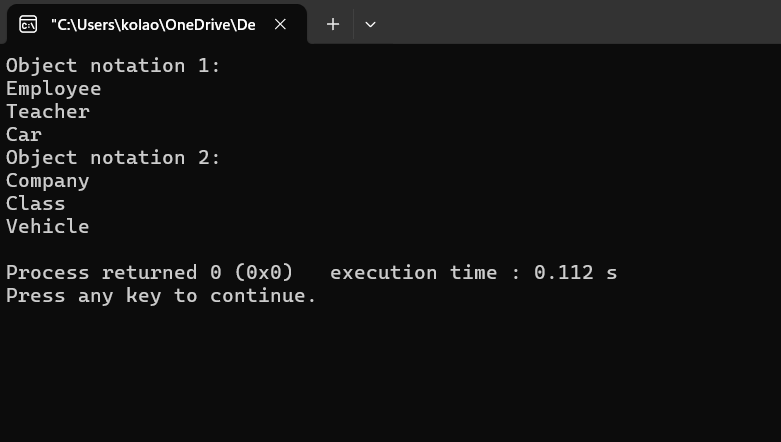
std::cout << "Class" << std::endl;

std::cout << "Vehicle" << std::endl;

return 0;

}

OUTPUT:



2) Design a class hierarchy to represent a geometric shapes system. Include classes for circles, rectangles, and triangles. Implement methods for calculating the area and perimeter of each shape. Demonstrate the use of abstract classes and pure virtual functions. Discuss how this design supports future additions of new shapes.

INPUT:

#include <iostream>

#include <cmath>

// Abstract base class Shape

class Shape {

public:

// Pure virtual functions for calculating area and perimeter

virtual double area() const = 0;

virtual double perimeter() const = 0;

// Virtual destructor

virtual ~Shape() {}

};

// Concrete class representing a circle

class Circle : public Shape {

private:

double radius;

public:

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

// Method to calculate area of circle

double area() const override {

return 3.14159 \* radius \* radius;

}

// Method to calculate perimeter (circumference) of circle

double perimeter() const override {

return 2 \* 3.14159 \* radius;

}

};

// Concrete class representing a rectangle

class Rectangle : public Shape {

private:

double length;

double width;

public:

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

// Method to calculate area of rectangle

double area() const override {

return length \* width;

}

// Method to calculate perimeter of rectangle

double perimeter() const override {

return 2 \* (length + width);

}

};

// Concrete class representing a triangle

class Triangle : public Shape {

private:

double side1;

double side2;

double side3;

public:

Triangle(double s1, double s2, double s3) : side1(s1), side2(s2), side3(s3) {}

// Method to calculate area of triangle using Heron's formula

double area() const override {

double s = (side1 + side2 + side3) / 2;

return sqrt(s \* (s - side1) \* (s - side2) \* (s - side3));

}

// Method to calculate perimeter of triangle

double perimeter() const override {

return side1 + side2 + side3;

}

};

// Function to display area and perimeter of a shape

void displayShapeInfo(const Shape& shape) {

std::cout << "Area: " << shape.area() << std::endl;

std::cout << "Perimeter: " << shape.perimeter() << std::endl;

}

int main() {

// Create instances of shapes

Circle circle(5.0);

Rectangle rectangle(4.0, 6.0);

Triangle triangle(3.0, 4.0, 5.0);

// Display information of each shape

std::cout << "Circle:" << std::endl;

displayShapeInfo(circle);

std::cout << std::endl;

std::cout << "Rectangle:" << std::endl;

displayShapeInfo(rectangle);

std::cout << std::endl;

std::cout << "Triangle:" << std::endl;

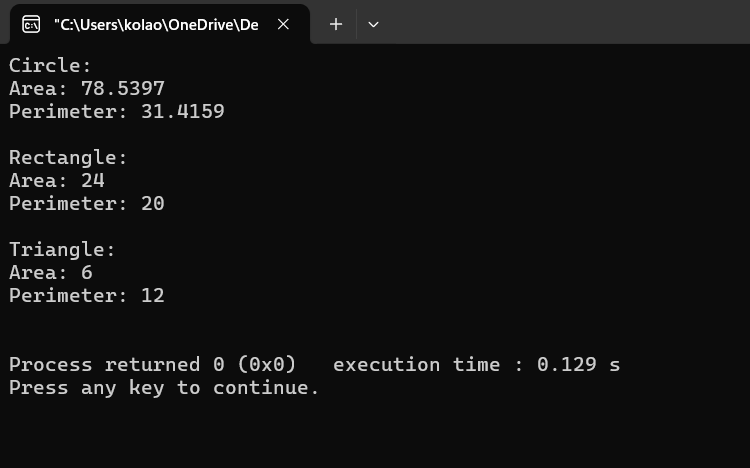
displayShapeInfo(triangle);

std::cout << std::endl;

return 0;

}

OUTPUT:



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.

INPUT:

#include <iostream>

#include <vector>

#include <string>

// Base class representing a Person

class Person {

protected:

std::string name;

int age;

public:

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

virtual void displayInfo() const {

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

}

};

// Derived class representing a Student, inheriting from Person

class Student : public Person {

private:

std::string studentID;

public:

Student(const std::string& personName, int personAge, const std::string& id) : Person(personName, personAge), studentID(id) {}

void displayInfo() const override {

std::cout << "Student ID: " << studentID << ", ";

Person::displayInfo();

}

};

// Derived class representing a Professor, inheriting from Person

class Professor : public Person {

private:

std::string department;

public:

Professor(const std::string& personName, int personAge, const std::string& dept) : Person(personName, personAge), department(dept) {}

void displayInfo() const override {

std::cout << "Department: " << department << ", ";

Person::displayInfo();

}

};

// Class representing a Course

class Course {

private:

std::string courseName;

std::vector<Student\*> enrolledStudents;

Professor\* courseProfessor;

public:

Course(const std::string& name, Professor\* prof) : courseName(name), courseProfessor(prof) {}

void enrollStudent(Student\* student) {

enrolledStudents.push\_back(student);

}

void displayInfo() const {

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

std::cout << "Professor: ";

courseProfessor->displayInfo();

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

for (const auto& student : enrolledStudents) {

student->displayInfo();

}

}

};

int main() {

// Create instances of students

Student student1("Alice", 20, "S001");

Student student2("Bob", 22, "S002");

// Create instances of professor

Professor professor1("Dr. Smith", 45, "Computer Science");

Professor professor2("Dr. Johnson", 50, "Physics");

// Create instances of courses

Course course1("C++ Programming", &professor1);

Course course2("Physics 101", &professor2);

// Enroll students in courses

course1.enrollStudent(&student1);

course1.enrollStudent(&student2);

course2.enrollStudent(&student1);

// Display information about courses

std::cout << "Course Information:" << std::endl;

course1.displayInfo();

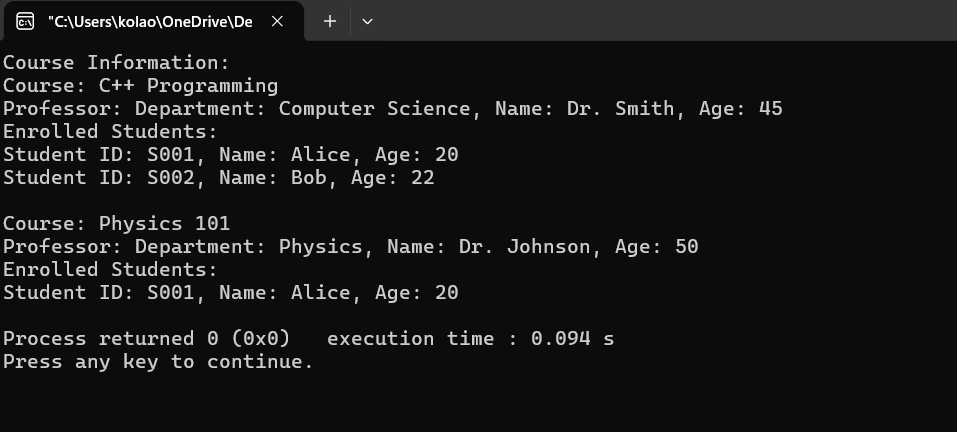
std::cout << std::endl;

course2.displayInfo();

return 0;

}

OUTPUT:



4) In 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

INPUT:

#include<iostream>

using namespace std;

int main(){

int salary;

char grade;

double bonus,orginal;

cout<<"enter the salary of the empolyee:";

cin>>salary;

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

cin>>grade;

if(grade=='A'){

if(salary<10000){

bonus=salary\*0.07;

}

else{

bonus=salary\*0.05;

}

orginal=salary+bonus;

}

if(grade=='B'){

if(salary<10000)

{

bonus=salary\*0.12;

}

else{

bonus=salary\*0.10;

}

orginal=salary+bonus;

}

cout<<" salary is :"<<salary<<endl;

cout<<"bonus is:"<<bonus<<endl;

cout<<"orginal is:"<<orginal;

}

OUTPUT:

