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

**a) Employee b) Teacher c) Car**

**Identify the object of the following class and create object.**

**a) Company b) Class c) Vehicle**

#include <iostream>

#include <string>

using namespace std;

// Employee Class

class Employee {

private:

string name;

int id;

public:

Employee(const string& n, int i) : name(n), id(i) {}

void display() const {

cout << "Name: " << name << ", ID: " << id << endl;

}

};

// Teacher Class

class Teacher {

private:

string name;

string subject;

public:

Teacher(const string& n, const string& s) : name(n), subject(s) {}

void display() const {

cout << "Name: " << name << ", Subject: " << subject << endl;

}

};

// Car Class

class Car {

private:

string brand;

string model;

int year;

public:

Car(const string& b, const string& m, int y) : brand(b), model(m), year(y) {}

void display() const {

cout << "Brand: " << brand << ", Model: " << model << ", Year: " << year << endl;

}

};

// Company Class

class Company {

private:

string name;

string location;

public:

Company(const string& n, const string& l) : name(n), location(l) {}

void display() const {

cout << "Company Name: " << name << ", Location: " << location << endl;

}

};

// Class Class

class Class {

private:

string name;

int grade;

public:

Class(const string& n, int g) : name(n), grade(g) {}

void display() const {

cout << "Class Name: " << name << ", Grade: " << grade << endl;

}

};

// Vehicle Class

class Vehicle {

private:

string type;

string color;

public:

Vehicle(const string& t, const string& c) : type(t), color(c) {}

void display() const {

cout << "Vehicle Type: " << type << ", Color: " << color << endl;

}

};

int main() {

// Examples of creating objects using different notations

// Employee objects

Employee emp1("John Doe", 101);

// Teacher objects

Teacher t1("Alice Johnson", "Mathematics");

// Car objects

Car car1("Toyota", "Camry", 2020);

// Company object

Company company("XYZ Corp", "New York");

// Class object

Class schoolClass("Physics", 11);

// Vehicle object

Vehicle vehicle("Sedan", "Blue");

// Displaying object details

emp1.display();

t1.display();

car1.display();

company.display();

schoolClass.display();

vehicle.display();

return 0;

}

OUTPUT:

Name: John Doe, ID: 101

Name: Alice Johnson, Subject: Mathematics

Brand: Toyota, Model: Camry, Year: 2020

Company Name: XYZ Corp, Location: New York

Class Name: Physics, Grade: 11

Vehicle Type: Sedan, Color: Blue

**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**

#include <iostream>

#include <cmath>

class Shape {

public:

virtual double area() const = 0;

virtual double perimeter() const = 0;

};

class Circle : public Shape {

private:

double radius;

public:

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

double area() const override {

return M\_PI \* radius \* radius;

}

double perimeter() const override {

return 2 \* M\_PI \* radius;

}

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

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

double area() const override {

return length \* width;

}

double perimeter() const override {

return 2 \* (length + width);

}

};

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) {}

double area() const override {

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

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

}

double perimeter() const override {

return side1 + side2 + side3;

}

};

int main() {

Circle circle(5.0);

Rectangle rectangle(4.0, 6.0);

Triangle triangle(3.0, 4.0, 5.0);

std::cout << "Circle area: " << circle.area() << ", perimeter: " << circle.perimeter() << std::endl;

std::cout << "Rectangle area: " << rectangle.area() << ", perimeter: " << rectangle.perimeter() << std::endl;

std::cout << "Triangle area: " << triangle.area() << ", perimeter: " << triangle.perimeter() << std::endl;

return 0;

}

OUTPUT:  
Circle area: 78.5398, perimeter: 31.4159

Rectangle area: 24, perimeter: 20

Triangle area: 6, perimeter: 12

**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**

#include <iostream>

#include <string>

#include <vector>

class Person {

protected:

std::string name;

int age;

public:

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

virtual void displayInfo() const {

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

}

};

class Student : public Person {

private:

std::string studentId;

std::vector<std::string> enrolledCourses;

public:

Student(const std::string& \_name, int \_age, const std::string& \_studentId)

: Person(\_name, \_age), studentId(\_studentId) {}

void enrollCourse(const std::string& courseName) {

enrolledCourses.push\_back(courseName);

}

void displayInfo() const {

Person::displayInfo();

std::cout << "Student ID: " << studentId << std::endl;

std::cout << "Enrolled Courses: ";

for (std::vector<std::string>::const\_iterator it = enrolledCourses.begin(); it != enrolledCourses.end(); ++it) {

std::cout << \*it;

if (it != enrolledCourses.end() - 1) {

std::cout << ", ";

}

}

std::cout << std::endl;

}

};

class Professor : public Person {

private:

std::string employeeId;

std::string department;

public:

Professor(const std::string& \_name, int \_age, const std::string& \_employeeId,

const std::string& \_department)

: Person(\_name, \_age), employeeId(\_employeeId), department(\_department) {}

void displayInfo() const {

Person::displayInfo();

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

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

}

};

class Course {

private:

std::string courseName;

Professor\* instructor;

public:

Course(const std::string& \_courseName, Professor\* \_instructor)

: courseName(\_courseName), instructor(\_instructor) {}

void displayInfo() const {

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

std::cout << "Instructor: ";

if (instructor) {

instructor->displayInfo();

} else {

std::cout << "N/A" << std::endl;

}

}

};

int main() {

Student student("Alice", 20, "S12345");

Professor professor("Dr. Smith", 45, "P98765", "Computer Science");

student.enrollCourse("Introduction to Programming");

student.enrollCourse("Data Structures");

Course programmingCourse("Programming 101", &professor);

student.displayInfo();

professor.displayInfo();

programmingCourse.displayInfo();

return 0;

}

OUTPUT:  
Name: Alice, Age: 20

Student ID: S12345

Enrolled Courses: Introduction to Programming, Data Structures

Name: Dr. Smith, Age: 45

Employee ID: P98765

Department: Computer Science

Course Name: Programming 101

Instructor: Name: Dr. Smith, Age: 45

Employee ID: P98765

Department: Computer Science

**4.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**

#include <iostream>

#include <cmath>

class Shape {

public:

virtual double area() const = 0;

virtual double perimeter() const = 0;

};

class Circle : public Shape {

private:

double radius;

public:

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

double area() const override {

return M\_PI \* radius \* radius;

}

double perimeter() const override {

return 2 \* M\_PI \* radius;

}

};

class Rectangle : public Shape {

private:

double length;

double width;

public:

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

double area() const override {

return length \* width;

}

double perimeter() const override {

return 2 \* (length + width);

}

};

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) {}

double area() const override {

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

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

}

double perimeter() const override {

return side1 + side2 + side3;

}

};

int main() {

Circle circle(5.0);

Rectangle rectangle(4.0, 6.0);

Triangle triangle(3.0, 4.0, 5.0);

std::cout << "Circle area: " << circle.area() << ", perimeter: " << circle.perimeter() << std::endl;

std::cout << "Rectangle area: " << rectangle.area() << ", perimeter: " << rectangle.perimeter() << std::endl;

std::cout << "Triangle area: " << triangle.area() << ", perimeter: " << triangle.perimeter() << std::endl;

return 0;

}

OUTPUT:  
Circle area: 78.5398, perimeter: 31.4159

Rectangle area: 24, perimeter: 20

Triangle area: 6, perimeter: 12

**5. 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**

#include <iostream>

#include <iomanip>

int main() {

char grade;

double salary, bonus = 0.0;

std::cout << "Enter the grade of the employee (A/B): ";

std::cin >> grade;

std::cout << "Enter the employee salary: ";

std::cin >> salary;

if (salary < 10000) {

bonus += 0.02 \* salary;

}

if (grade == 'A') {

bonus += 0.05 \* salary;

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

bonus += 0.10 \* salary;

}

double totalSalary = salary + bonus;

std::cout << std::fixed << std::setprecision(2);

std::cout << "Salary = $" << salary << std::endl;

std::cout << "Bonus = $" << bonus << std::endl;

std::cout << "Total to be paid = $" << totalSalary << std::endl;

return 0;

}

OUTPUT:

Enter the grade of the employee (A/B): B

Enter the employee salary: 50000

Salary = $50000.00

Bonus = $5000.00

Total to be paid = $55000.00