DAY-05

Question-17 : Hierarchical Inheritance

Scenario: A university system needs to manage different types of users: students and faculty. Both students and faculty share common attributes such as name and ID, but also have their specific attributes like courses and publications, respectively.

Question: How can you use hierarchical inheritance to design a base class UniversityMember and then derive Student and Faculty classes from it, incorporating both shared and unique attributes?

Sol : #include <iostream>

#include <string>

#include <vector>

using namespace std;

class UniversityMember {

protected:

string name;

int id;

public:

UniversityMember(string n, int i) : name(n), id(i) {}

void displayInfo() {

cout << "Name: " << name << endl;

cout << "ID: " << id << endl;

}

};

class Student : public UniversityMember {

private:

vector<string> courses;

public:

Student(string n, int i, vector<string> c) : UniversityMember(n, i), courses(c) {}

void displayInfo() {

UniversityMember::displayInfo();

cout << "Courses: ";

for (const auto& course : courses) {

cout << course << ", ";

}

cout << endl;

}

};

class Faculty : public UniversityMember {

private:

vector<string> publications;

public:

Faculty(string n, int i, vector<string> p) : UniversityMember(n, i), publications(p) {}

void displayInfo() {

UniversityMember::displayInfo();

cout << "Publications: ";

for (const auto& publication : publications) {

cout << publication << ", ";

}

cout << endl;

}

};

int main() {

vector<string> studentCourses = {"Math", "Physics"};

Student student("Rakhi", 12345, studentCourses);

cout << "Student Information:" << endl;

student.displayInfo();

cout << endl;

vector<string> facultyPublications = {"Journal of Physics", "Science"};

Faculty faculty("Dr. Vamshi", 54321, facultyPublications);

cout << "Faculty Information:" << endl;

faculty.displayInfo();

return 0;

}

O/P : Student Information:

Name: vinod

ID: 12345

Courses: Math, Physics,

Faculty Information:

Name: Dr. Vamshi

ID: 54321

Publications: Journal of Physics, Science

Question-18 : 18. Multilevel Inheritance

Scenario: In a gaming application, there's a need to create a specialized type of enemy. Start with a base class GameObject, derive an Enemy class from GameObject, and then derive a BossEnemy class from Enemy. The BossEnemy has unique attributes like special attacks and enhanced health points.

Question: How would you structure the classes GameObject, Enemy, and BossEnemy to demonstrate multilevel inheritance, ensuring that each derived class adds additional details or functionality?

Sol : #include <iostream>

#include <string>

using namespace std;

class GameObject {

protected:

string name;

public:

GameObject(string n) : name(n) {}

virtual void displayInfo() {

cout << "GameObject: " << name << endl;

}

};

class Enemy : public GameObject {

protected:

int health;

public:

Enemy(string n, int h) : GameObject(n), health(h) {}

void displayInfo() override {

GameObject::displayInfo();

cout << "Health: " << health << endl;

}

};

class BossEnemy : public Enemy {

private:

string specialAttack;

public:

BossEnemy(string n, int h, string attack) : Enemy(n, h), specialAttack(attack) {}

void displayInfo() override {

Enemy::displayInfo();

cout << "Special Attack: " << specialAttack << endl;

}

};

int main() {

GameObject gameObject("Basic Enemy");

gameObject.displayInfo();

cout << endl;

Enemy enemy("Normal Enemy", 100);

enemy.displayInfo();

cout << endl;

BossEnemy bossEnemy("Final Boss", 500, "Fire Breath");

bossEnemy.displayInfo();

return 0;

}

O/P:GameObject: Basic Enemy

GameObject: Normal Enemy

Health: 100

GameObject: Final Boss

Health: 500

Special Attack: Fire Breath

Question-19 : 19. Hybrid Inheritance

Scenario: Consider a software system for managing vehicle loans. It includes classes for Vehicle (base class), Car and Truck (derived from Vehicle), and Loan (an independent class). A VehicleLoan class needs to inherit properties from both a specific vehicle type (e.g., Car or Truck) and the Loan class.

Sol : #include <iostream>

#include <string>

using namespace std;

class GameObject {

protected:

string name;

public:

GameObject(string n) : name(n) {}

virtual void displayInfo() {

cout << "GameObject: " << name << endl;

}

};

class Enemy : public GameObject {

protected:

int health;

public:

Enemy(string n, int h) : GameObject(n), health(h) {}

void displayInfo() override {

GameObject::displayInfo();

cout << "Health: " << health << endl;

}

};

class BossEnemy : public Enemy {

private:

string specialAttack;

public:

BossEnemy(string n, int h, string attack) : Enemy(n, h), specialAttack(attack) {}

void displayInfo() override {

Enemy::displayInfo();

cout << "Special Attack: " << specialAttack << endl;

}

};

int main() {

GameObject gameObject("Basic Enemy");

gameObject.displayInfo();

cout << endl;

Enemy enemy("Normal Enemy", 100);

enemy.displayInfo();

cout << endl;

BossEnemy bossEnemy("Final Boss", 500, "Fire Breath");

bossEnemy.displayInfo();

return 0;

}

O/P : GameObject: Basic Enemy

GameObject: Normal Enemy

Health: 100

GameObject: Final Boss

Health: 500

Special Attack: Fire Breath

Question-20 : 20. Multiple Inheritance

Scenario: In an employee management system, there are two base classes: Person, containing personal information, and Job, containing job-related information. A derived class Employee needs to combine information from both Person and Job.

Question: How would you implement the Person, Job, and Employee classes to demonstrate multiple inheritance, ensuring that Employee inherits attributes and methods from both Person and Job?

Sol : #include <iostream>

#include <string>

using namespace std;

class Person {

protected:

string name;

int age;

public:

Person(string n, int a) : name(n), age(a) {}

void displayPersonalInfo() {

cout << "Name: " << name << endl;

cout << "Age: " << age << endl;

}

};

class Job {

protected:

string position;

double salary;

public:

Job(string pos, double sal) : position(pos), salary(sal) {}

void displayJobInfo() {

cout << "Position: " << position << endl;

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

}

};

class Employee : public Person, public Job {

public:

Employee(string n, int a, string pos, double sal) : Person(n, a), Job(pos, sal) {}

void displayEmployeeInfo() {

displayPersonalInfo();

displayJobInfo();

}

};

int main() {

Employee emp("John Doe", 30, "Software Engineer", 75000.0);

cout << "Employee Information:" << endl;

emp.displayEmployeeInfo();

return 0;

}

O/P : Employee Information:

Name: John Doe

Age: 30

Position: Software Engineer

Salary: $75000