1. Scenario:

Design a class hierarchy to represent various entities in a university system.

Base Class (Person):

Data members: name (string), age (int)

Member functions: getDetails(), a virtual function to print basic person details

Derived Class (Student): (Single Inheritance)

Inherits from Person

Data members: studentId (int), major (string)

Member functions:

setMajor(string) to set the student's major

getMajor() to retrieve the major

Override getDetails() to include student-specific information

Derived Class (Faculty): (Single Inheritance)

Inherits from Person

Data members: department (string), employeeId (int)

Member functions:

setDepartment(string) to set the faculty member's department

getDepartment() to retrieve the department

Override getDetails() to include faculty-specific information

Derived Class (TeachingAssistant): (Multilevel Inheritance)

Inherits from Student (inherits indirectly from Person as well)

Data member: coursesTeaching (array/vector of strings)

Member functions:

setCoursesTeaching(string[]) to set the courses the TA is teaching

getCoursesTeaching() to retrieve the list of courses

Override getDetails() to include TA-specific information (e.g., courses)

Derived Class (ResearchAssistant): (Hierarchical Inheritance)

Inherits from Person (separate inheritance from Student)

Data members: researchArea (string), supervisor (string)

Member functions:

setResearchArea(string) to set the research area

getResearchArea() to retrieve the research area

setSupervisor(string) to set the research supervisor

getSupervisor() to retrieve the supervisor

Override getDetails() to include RA-specific information

Derived Class (GraduateStudentTA): (Hybrid Inheritance)

Inherits from both Student and TeachingAssistant (combines functionality)

Might have additional data members or functions specific to graduate student TAs

Code :

#include <iostream>

#include <string>

using namespace std;

class Person { // Base class Person

protected:

string name;

int age;

public:

Person(string name, int age) : name(name), age(age) {} // Virtual function

virtual void getDetails() {

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

}

};

class Student : public Person { // Derived class Student

protected:

int studentId;

string major;

public:

Student(string name, int age, int studentId) : Person(name, age), studentId(studentId) {}

void setMajor(string major) {

this->major = major;

} string getMajor() {

return major; }

void getDetails() override { // Override getDetails to include student-specific information

cout << "Student Details:" << endl;

Person::getDetails();

cout << "Student ID: " << studentId << ", Major: " << major << endl;

}

};

class Faculty : public Person { // Derived class Faculty

protected:

string department;

int employeeId;

public:

Faculty(string name, int age, int employeeId) : Person(name, age), employeeId(employeeId) {}

void setDepartment(string department) {

this->department = department;

} string getDepartment() {

return department;

}

void getDetails() override { // Override getDetails to include faculty-specific information

cout << "Faculty Details:" << endl;

Person::getDetails();

cout << "Employee ID: " << employeeId << ", Department: " << department << endl;

}

};

class TeachingAssistant : public Student { // (Multilevel Inheritance)

protected:

string coursesTeaching;

public:

TeachingAssistant(string name, int age, int studentId) : Student(name, age, studentId) {}

void setCoursesTeaching(string courses) {

coursesTeaching = courses;

}

string getCoursesTeaching() {

return coursesTeaching;

}

void getDetails() override { // Override getDetails to include TA-specific information

cout << "Teaching Assistant Details:" << endl;

Student::getDetails();

cout << "Courses Teaching: " << coursesTeaching << endl;

}

};

int main() {

Student s("hari", 20, 12345);

s.setMajor("Computer Science");

Faculty f("Ram", 35, 1001);

f.setDepartment("Engineering");

TeachingAssistant ta("sai", 25, 23456);

ta.setMajor("Electrical Engineering");

ta.setCoursesTeaching("EE101, EE202");

Person\* people[] = {&s, &f, &ta};

for (auto person : people) {

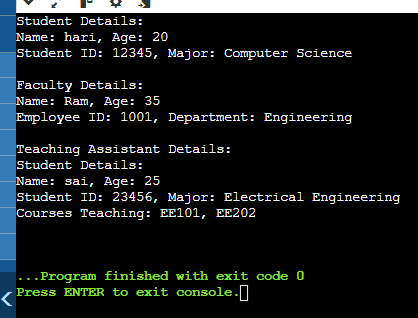
person->getDetails();

cout << endl;

} return 0;

}

OUTPUT:



2. Scenario:

Imagine you're developing a university management system. You have a base class named Person that stores basic information about individuals associated with the university, such as:

name (string)

id (int)

Question: Design a class hierarchy using inheritance to represent different types of people within the university. Consider the following categories:

Student: Inherits from Person and has additional attributes like:

major (string)

gpa (double)

A method calculateSemesterGPA(vector<double> grades) that takes a vector of grades (doubles) and calculates the semester GPA.

Faculty: Inherits from Person and has additional attributes like:

department (string)

title (string) - e.g., "Professor", "Lecturer"

A method teachCourse(string courseName) that simulates assigning a faculty member to teach a specific course.

Additional Considerations:

You can introduce further derived classes if you think of more specific roles within the university (e.g., Staff, Administrator).

Think about access specifiers (public, private, protected) for member variables and methods in the base and derived classes.

Consider virtual functions (especially in the context of polymorphism) if there's common functionality that might have different implementations in derived classes.

Guiding Tips: Focus on code clarity and maintainability.

Use meaningful variable and method names.

Add comments to explain your design choices.

Test your code to ensure it works as expected.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

#include <vector>

#include <numeric>

using namespace std;

class Person {

protected: // Base Class: Person

string name;

int id;

public:

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

string getName() const {

return name; }

int getId() const {

return id; }

void displayInfo() const {

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

};

class Student : public Person { // Derived Class: Student

private:

string major;

double gpa;

public:

Student(const string &name, int id, const string &major, double gpa)

: Person(name, id), major(major), gpa(gpa) {}

string getMajor() const {

return major; }

double getGpa() const {

return gpa;

}

void displayInfo() const {

Person::displayInfo();

cout << "Major: " << major << ", GPA: " << gpa << endl;

}

double calculateSemesterGPA(const vector<double>& grades) const {

if (grades.empty()) {

return 0.0; }

double total = accumulate(grades.begin(), grades.end(), 0.0);

return total / grades.size(); }

};

class Faculty : public Person { // Derived Class: Faculty

private:

string department;

string title;

public:

Faculty(const string &name, int id, const string &department, const string &title)

: Person(name, id), department(department), title(title) {}

string getDepartment() const {

return department; }

string getTitle() const {

return title; }

void displayInfo() const {

Person::displayInfo();

cout << "Department: " << department << ", Title: " << title << endl;

}

void teachCourse(const string& courseName) const {

cout << name << " is now teaching " << courseName << "." << endl;

}

};

int main() { // Main Function

Student student("Harika", 12345, "Computer Science", 3.8);

Faculty faculty("Chaitanya", 67890, "Python", "Professor");

student.displayInfo();

faculty.displayInfo();

vector<double> grades = {4.0, 3.7, 3.3, 3.8};

double semesterGPA = student.calculateSemesterGPA(grades);

cout << "Semester GPA: " << semesterGPA << endl;

faculty.teachCourse("Python");

return 0;

}

OUTPUT:

