Day-3:

**INHERITENCE**

inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically.

 the class which inherits the members of another class is called derived class and the class whose members are inherited is called base class. The derived class is the specialized class for the base class.

**SINGLE INHERITENCE**

Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

A->B

Where 'A' is the base class, and 'B' is the derived class.

When one class inherits another class, it is known as single level inheritance.

#include <iostream>

using namespace std;

class Account {

public:

float salary = 60000;

};

class Programmer: public Account {

public:

float bonus = 5000;

};

int main(void) {

Programmer p1;

cout<<"Salary: "<<p1.salary<<endl;

cout<<"Bonus: "<<p1.bonus<<endl;

return 0;

}

Output:

Salary: 60000

Bonus: 5000

**MULTILEVEL INHERITENCE**

When one class inherits another class which is further inherited

by another class, it is known as multi level inheritance

A->B->C

#include <iostream>

using namespace std;

class Animal {

public:

void eat() {

cout<<"Eating..."<<endl;

}

};

class Dog: public Animal

{

public:

void bark(){

cout<<"Barking..."<<endl;

}

};

class BabyDog: public Dog

{

public:

void weep() {

cout<<"Weeping...";

}

};

int main(void) {

BabyDog d1;

d1.eat();

d1.bark();

d1.weep();

return 0;

}

Output:

Eating...

Barking...

Weeping...

**MULTIPLE INHERITENCE**

Multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.

#include <iostream>

using namespace std;

class A

{

protected:

int a;

public:

void get\_a(int n)

{

a=n;

}

};

class B

{

protected:

int b;

public:

void get\_b(int n)

{

b=n;

}

};

class C :public A,public B

{

public:

void display()

{

std::cout<<"The value of a is:"<<a<<std::endl;

std::cout<<"The value of b is:"<<b<<std::endl;

cout<<"addition of a and b is:"<<a+b;

}

};

int main()

{

C c;

c.get\_a(10);

c.get\_b(20);

c.display();

return 0;

}

Output:

The value of a is : 10

The value of b is : 20

Addition of a and b is : 30

**HYBRID INHERITENCE**

Hybrid inheritance is a combination of more than one type of inheritance.

#include <iostream>

using namespace std;

class A

{

protected:

int a;

public:

void get\_a()

{

std::cout << "Enter the value of 'a' : " << std::endl;

cin>>a;

}

};

class B : public A

{

protected:

int b;

public:

void get\_b()

{

std::cout << "Enter the value of 'b' : " << std::endl;

cin>>b;

}

};

class C

{

protected:

int c;

public:

void get\_c()

{

std::cout << "Enter the value of c is : " << std::endl;

cin>>c;

}

};

class D : public B, public C

{

protected:

int d;

public:

void mul()

{

get\_a();

get\_b();

get\_c();

std::cout << "Multiplication of a,b,c is : " <<a\*b\*c<< std::endl;

}

};

int main()

{

D d;

d.mul();

return 0;

}

Output:

Enter the value of 'a' :

10

Enter the value of 'b' :

20

Enter the value of c is :

30

Multiplication of a,b,c is : 6000

**HIERARCHICAL INHERITANCE**

Hierarchical inheritance is defined as the process of deriving more than one class from a base class.

#include <iostream>

using namespace std;

class Shape // Declaration of base class.

{

public:

int a;

int b;

void get\_data(int n,int m)

{

a= n;

b = m;

}

};

class Rectangle : public Shape // inheriting Shape class

{

public:

int rect\_area()

{

int result = a\*b;

return result;

}

};

class Triangle : public Shape // inheriting Shape class

{

public:

int triangle\_area()

{

float result = 0.5\*a\*b;

return result;

}

};

int main()

{

Rectangle r;

Triangle t;

int length,breadth,base,height;

std::cout << "Enter the length and breadth of a rectangle: " << std::endl;

cin>>length>>breadth;

r.get\_data(length,breadth);

int m = r.rect\_area();

std::cout << "Area of the rectangle is : " <<m<< std::endl;

std::cout << "Enter the base and height of the triangle: " << std::endl;

cin>>base>>height;

t.get\_data(base,height);

float n = t.triangle\_area();

std::cout <<"Area of the triangle is : " << n<<std::endl;

return 0;

}

Output:

Enter the length and breadth of a rectangle:

23

20

Area of the rectangle is : 460

Enter the base and height of the triangle:

2

5

Area of the triangle is : 5

Base Class (Person):

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

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

Derived Class (Student): (Single Inheritance)

#include <iostream>

#include <string>

// Base class

class Person {

protected:

std::string name;

int age;

public:

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

virtual void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

virtual ~Person() {} // Virtual destructor for proper cleanup

};

// Derived class

class Student : public Person {

public:

Student(std::string n, int a) : Person(n, a) {}

void getDetails() override {

std::cout << "Student Details -\n";

Person::getDetails();

}

};

int main() {

Student s("Ravi teja", 25);

s.getDetails();

return 0;

}

Output:

Student Details -

Name: Ravi teja

Age: 25

2).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)

#include <iostream>

#include <string>

class Person {

protected:

std::string name;

int age;

public:

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

virtual void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

virtual ~Person() {}

};

class Student : public Person {

private:

int studentId;

std::string major;

public:

Student(std::string n, int a, int id, std::string m) : Person(n, a), studentId(id), major(m) {}

void setMajor(std::string m) {

major = m;

}

std::string getMajor() {

return major;

}

void getDetails() override {

std::cout << "Student Details -\n";

Person::getDetails();

std::cout << "Student ID: " << studentId << "\nMajor: " << major << std::endl;

}

};

class Faculty : public Person {

private:

std::string department;

public:

Faculty(std::string n, int a, std::string dept) : Person(n, a), department(dept) {}

void setDepartment(std::string dept) {

department = dept;

}

std::string getDepartment() {

return department;

}

void getDetails() override {

std::cout << "Faculty Details -\n";

Person::getDetails();

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

}

};

int main() {

Student s("vasu", 24, 123, "Computer Science");

s.getDetails();

Faculty f("Dr. siva", 40, "English");

f.getDetails();

return 0;

}

Output:

Student Details -

Name: vasu

Age: 24

Student ID: 123

Major: Computer Science

Faculty Details -

Name: Dr. siva

Age: 40

Department: English

3). 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)

#include <iostream>

#include <string

class Person {

protected:

std::string name;

int age;

public:

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

virtual void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

virtual ~Person() {}

};

class Faculty : public Person {

protected:

std::string department;

int employeeId;

public:

Faculty(std::string n, int a, std::string dept, int id) : Person(n, a), department(dept), employeeId(id) {}

void setDepartment(std::string dept) {

department = dept;

}

std::string getDepartment() {

return department;

}

void getDetails() override {

std::cout << "Faculty Details -\n";

Person::getDetails();

std::cout << "Department: " << department << "\nEmployee ID: " << employeeId << std::endl;

}

};

class TeachingAssistant : public Faculty {

private:

std::string course;

public:

TeachingAssistant(std::string n, int a, std::string dept, int id, std::string course)

: Faculty(n, a, dept, id), course(course) {}

void setCourse(std::string c) {

course = c;

}

std::string getCourse() {

return course;

}

void getDetails() override {

std::cout << "Teaching Assistant Details -\n";

Faculty::getDetails();

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

}

};

int main() {

TeachingAssistant ta("Emily Brown", 28, "Computer Science", 12345, "Introduction to Programming");

ta.getDetails();

return 0;

}

Output:

Teaching Assistant Details -

Faculty Details -

Name: Emily Brown

Age: 28

Department: Computer Science

Employee ID: 12345

Course: Introduction to Programming

4). 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)

#include <iostream>

#include <string>

#include <vector>

class Person {

protected:

std::string name;

int age;

public:

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

void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

};

class Student : public Person {

protected:

int studentId;

std::vector<std::string> courses;

public:

Student(std::string n, int a, int id) : Person(n, a), studentId(id) {}

void addCourse(std::string course) {

courses.push\_back(course);

}

void setCourses(std::vector<std::string> coursesList) {

courses = coursesList;

}

std::vector<std::string> getCourses() {

return courses;

}

void getStudentDetails() {

std::cout << "Student ID: " << studentId << "\nCourses Enrolled:\n";

for (const auto& course : courses) {

std::cout << "- " << course << std::endl;

}

}

};

class ResearchAssistant : public Student {

private:

std::vector<std::string> coursesTeaching;

public:

ResearchAssistant(std::string n, int a, int id) : Student(n, a, id) {}

void setCoursesTeaching(std::vector<std::string> courses) {

coursesTeaching = courses;

}

std::vector<std::string> getCoursesTeaching() {

return coursesTeaching;

}

void getDetails() {

std::cout << "Research Assistant Details -\n";

Person::getDetails();

getStudentDetails();

std::cout << "Courses Teaching:\n";

for (const auto& course : coursesTeaching) {

std::cout << "- " << course << std::endl;

}

}

};

int main() {

ResearchAssistant ra("Emily Brown", 25, 12345);

ra.addCourse("Computer Science");

ra.addCourse("Data Structures");

ra.setCoursesTeaching({"Introduction to Programming", "Algorithms"});

ra.getDetails();

return 0;

}

Output:

Research Assistant Details -

Name: Emily Brown

Age: 25

Student ID: 12345

Courses Enrolled:

- Computer Science

- Data Structures

Courses Teaching:

- Introduction to Programming

- Algorithms

5). 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)

#include <iostream>

#include <string>

#include <vector>

class Person {

protected:

std::string name;

int age;

public:

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

void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

};

class Student {

protected:

int studentId;

std::vector<std::string> courses;

public:

Student(int id) : studentId(id) {}

void addCourse(std::string course) {

courses.push\_back(course);

}

void setCourses(std::vector<std::string> coursesList) {

courses = coursesList;

}

std::vector<std::string> getCourses() {

return courses;

}

void getStudentDetails() {

std::cout << "Student ID: " << studentId << "\nCourses Enrolled:\n";

for (const auto& course : courses) {

std::cout << "- " << course << std::endl;

}

}

};

class GraduateStudentTA : public Person, public Student {

private:

std::string researchArea;

std::string supervisor;

public:

GraduateStudentTA(std::string n, int a, int id) : Person(n, a), Student(id) {}

void setResearchArea(std::string area) {

researchArea = area;

}

std::string getResearchArea() {

return researchArea;

}

void setSupervisor(std::string sup) {

supervisor = sup;

}

std::string getSupervisor() {

return supervisor;

}

void getDetails() {

std::cout << "Graduate Student TA Details -\n";

Person::getDetails();

getStudentDetails();

std::cout << "Research Area: " << researchArea << "\nSupervisor: " << supervisor << std::endl;

}

};

int main() {

GraduateStudentTA gst("Alice Johnson", 27, 12345);

gst.addCourse("Computer Science");

gst.addCourse("Data Structures");

gst.setResearchArea("Machine Learning");

gst.setSupervisor("Dr. Smith");

gst.getDetails();

return 0;

}

Output:

Graduate Student TA Details -

Name: Alice Johnson

Age: 27

Student ID: 12345

Courses Enrolled:

- Computer Science

- Data Structures

Research Area: Machine Learning

Supervisor: Dr. Smith

6). Inherits from both Student and TeachingAssistant (combines functionality)

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

#include <iostream>

#include <string>

#include <vector>

class Person {

protected:

std::string name;

int age;

public:

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

void getDetails() {

std::cout << "Name: " << name << "\nAge: " << age << std::endl;

}

};

class Student : public Person {

protected:

int studentId;

std::vector<std::string> courses;

public:

Student(std::string n, int a, int id) : Person(n, a), studentId(id) {}

void addCourse(std::string course) {

courses.push\_back(course);

}

void setCourses(std::vector<std::string> coursesList) {

courses = coursesList;

}

std::vector<std::string> getCourses() {

return courses;

}

void getStudentDetails() {

std::cout << "Student ID: " << studentId << "\nCourses Enrolled:\n";

for (const auto& course : courses) {

std::cout << "- " << course << std::endl;

}

}

};

class TeachingAssistant : public Person {

protected:

std::vector<std::string> coursesTeaching;

public:

TeachingAssistant(std::string n, int a) : Person(n, a) {}

void setCoursesTeaching(std::vector<std::string> courses) {

coursesTeaching = courses;

}

std::vector<std::string> getCoursesTeaching() {

return coursesTeaching;

}

void getTeachingDetails() {

std::cout << "Courses Teaching:\n";

for (const auto& course : coursesTeaching) {

std::cout << "- " << course << std::endl;

}

}

}

class GraduateStudentTA : public Student, public TeachingAssistant {

private:

std::string researchArea;

std::string supervisor;

public:

GraduateStudentTA(std::string n, int a, int id)

: Person(n, a), Student(n, a, id), TeachingAssistant(n, a) {}

void setResearchArea(std::string area) {

researchArea = area;

}

std::string getResearchArea() {

return researchArea;

}

void setSupervisor(std::string sup) {

supervisor = sup;

}

std::string getSupervisor() {

return supervisor;

}

void getDetails() {

std::cout << "Graduate Student TA Details -\n";

Person::getDetails();

getStudentDetails();

getTeachingDetails();

std::cout << "Research Area: " << researchArea << "\nSupervisor: " << supervisor << std::endl;

}

};

int main() {

GraduateStudentTA gst(“ Johnson", 23, 9876);

gst.addCourse("Computer Science");

gst.addCourse("Data Structures");

gst.setCoursesTeaching({"Introduction to Programming", "Algorithms"});

gst.setResearchArea("Machine Learning");

gst.setSupervisor("Dr. siva");

gst.getDetails();

return 0;

}

Output:

Graduate Student TA Details -

Name: Johnson

Age: 23

Student ID: 9876

Courses Enrolled:

- Computer Science

- Data Structures

Courses Teaching:

- Introduction to Programming

- Algorithms

Research Area: Machine Learning

Supervisor: Dr. Siva

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 <vector>

#include <string>

using namespace std;

// Base class Person

class Person {

protected: // Protected access specifier allows derived classes to access these members

string name;

int id;

public:

// Constructor to initialize name and id

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

// Virtual destructor for proper cleanup of derived class objects

virtual ~Person() {}

// Method to display basic information

void displayInfo() {

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

}

};

// Derived class Student from Person

class Student : public Person {

private:

string major;

double gpa;

public:

// Constructor to initialize name, id, major, and gpa

Student(string n, int i, string m, double g) : Person(n, i), major(m), gpa(g) {}

// Method to calculate the semester GPA

double calculateSemesterGPA(vector<double> grades) {

double total = 0.0;

for(double grade : grades) {

total += grade;

}

gpa = total / grades.size();

return gpa;

}

// Override displayInfo to include major and gpa

void displayInfo() override {

Person::displayInfo(); // Call base class displayInfo

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

}

};

// Derived class Faculty from Person

class Faculty : public Person {

private:

string department;

string title;

public:

// Constructor to initialize name, id, department, and title

Faculty(string n, int i, string d, string t) : Person(n, i), department(d), title(t) {}

// Method to simulate teaching a course

void teachCourse(string courseName) {

cout << title << " " << name << " is teaching " << courseName << " in the " << department << " department." << endl;

}

// Override displayInfo to include department and title

void displayInfo() override {

Person::displayInfo(); // Call base class displayInfo

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

}

};

// Main function to test the classes

int main() {

// Create a Student object

Student student("Alice", 123, "Computer Science", 3.5);

student.displayInfo();

vector<double> grades = {3.7, 3.8, 3.6};

cout << "New Semester GPA: " << student.calculateSemesterGPA(grades) << endl;

// Create a Faculty object

Faculty faculty("Dr. Smith", 456, "Engineering", "Professor");

faculty.displayInfo();

faculty.teachCourse("Physics 101");

return 0;

}

Function overloading:

#include <iostream>

using namespace std;

class cal{

public:

static int add(int a,int b)

{

return a+b;

}

static int add(int a,int b,int c)

{

return a+b+c;

}

};

int main(void) {

cal C;

cout<<C.add(10,20)<<endl;

cout<<C.add(12,20,23);

return 0;

}