Sunny kumar
26 june class task
batch-Linux System Programming Track

Inheritance

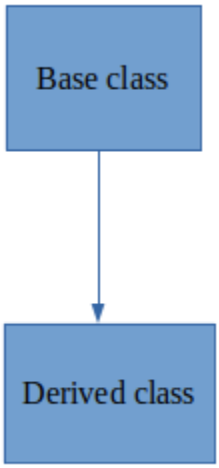
Inheritance in C++ is a feature that allows a class to inherit properties and behaviors (methods) from another class. It promotes code reusability and establishes a relationship between the base class and derived classes.

Types of inheritance:-

1.single inheritance
2.Multiple Inheritance
3.Hierarchical Inheritance
4.Multilevel Inheritance
5.Hybrid Inheritance

single inheritance

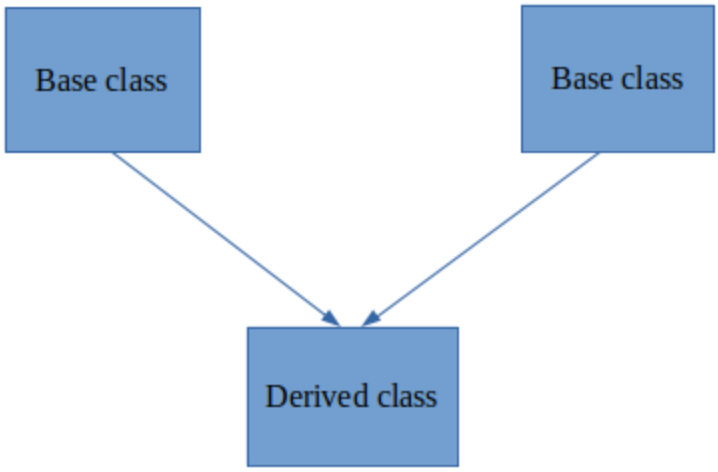
When a single derived class is created from a single base class is called single inheritance.



```cpp
#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;
}
```

Multiple Inheritance

When a derived class is created from more than one base class is called multiple inheritance.

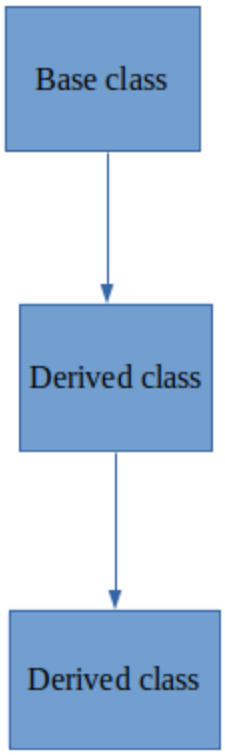



```
#include<iostream>
using namespace std;

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,publicB{
   public:
   void Display()
   {
```

```
        cout<<"the value of a is: "<<a<<endl;
        cout<<"the value of b is: "<<<b<endl;
        cout<<"addition of a and b s: "<<a+b<<endl;
    }
};

int main()
{
    C c1;
    c1.get_a(10);
    c1.get_b(20);
    c1.Display();
    return 0;
}
```

Multilevel Inheritance

When a derived class is created from another derived class is called multilevel inheritance.

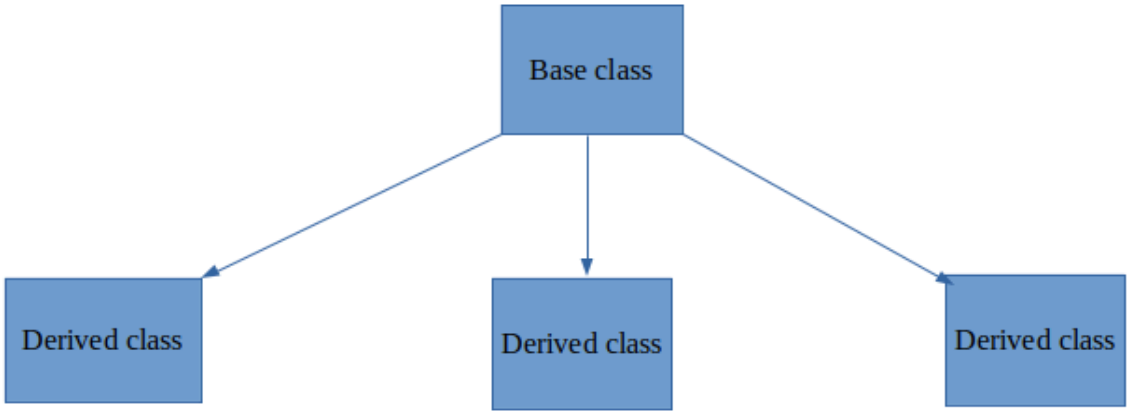



```
#include<iostream>
using namespace std;

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,publicB{
   public:
   void Display()
   {
       cout<<"the value of a is: "<<a<<endl;
       cout<<"the value of b is: "<<<b<endl;
       cout<<"addition of a and b s: "<<a+b<<endl;
   }
};

int main()
{
   C c1;
   c1.get_a(10);
   c1.get_b(20);
   c1.Display();
   return 0;
}
```

Hierarchical Inheritance

When more than one derived class created from a single base class is called Hierarchical Inheritance.

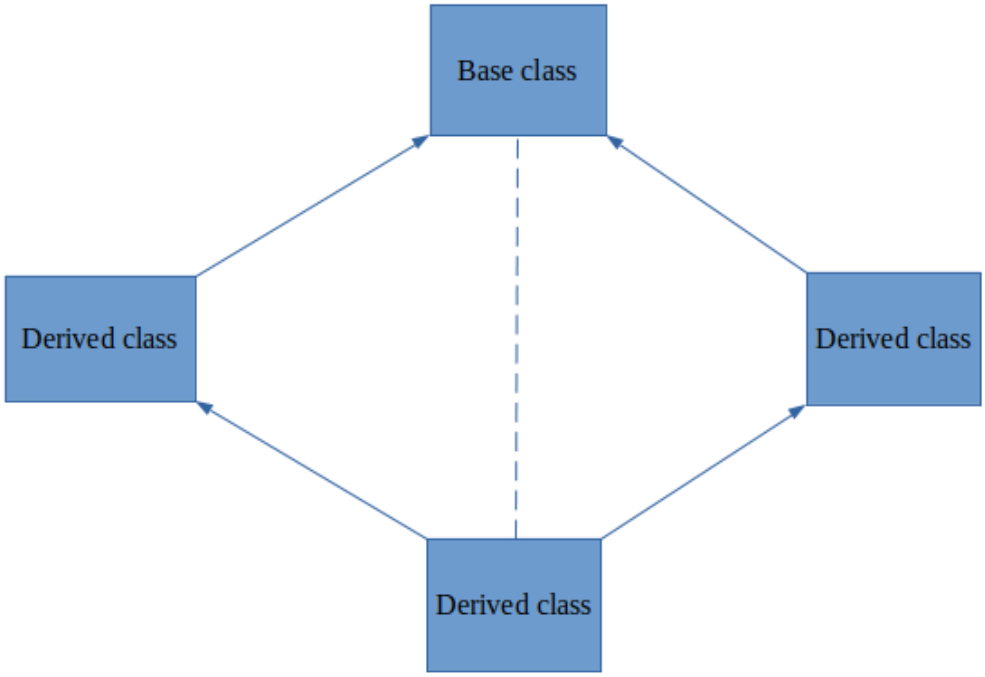



```cpp
#include <iostream>
using namespace std;
class Person {
protected:
   string name;
   int age;
public:
   void getDetails() {
       cout << "Enter name: ";
       cin >> name;
       cout << "Enter age: ";
       cin >> age;
   }
   void displayDetails() {
       cout << "Name: " << name << ", Age: " << age << endl;
   }
};
class Student : public Person {
private:
   int studentID;
public:
   void getStudentID() {
       cout << "Enter student ID: ";
       cin >> studentID;
   }
```

```
    void displayStudentDetails() {
        displayDetails();
        cout << "Student ID: " << studentID << endl;
    }
};
class Teacher : public Person {
private:
    int teacherID;
public:
    void getTeacherID() {
        cout << "Enter teacher ID: ";
        cin >> teacherID;
    }

    void displayTeacherDetails() {
        displayDetails();
        cout << "Teacher ID: " << teacherID << endl;
    }
};

int main() {
    Student student;
    Teacher teacher;

    cout << "Enter details for student:" << endl;
    student.getDetails();
    student.getStudentID();

    cout << "Enter details for teacher:" << endl;
    teacher.getDetails();
    teacher.getTeacherID();

    cout << "\nStudent Details:" << endl;
    student.displayStudentDetails();

    cout << "\nTeacher Details:" << endl;
    teacher.displayTeacherDetails();

    return 0;
}
```

Hybrid Inheritance

Combination of single , multiple and hierarchical inheritance is called Hybrid Inheritance.




```
#include <iostream>
using namespace std;

class A {
protected:
   int a;
public:
   void get_a() {
       cout << "Enter the value of 'a': " << endl;
       cin >> a;
   }
};

class B : public A {
protected:
   int b;
public:
```

```
    void get_b() {
        cout << "Enter the value of 'b': " << endl;
        cin >> b;
    }
};

class C {
protected:
    int c;
public:
    void get_c() {
        cout << "Enter the value of 'c': " << endl;
        cin >> c;
    }
};

class D : public B, public C {
public:
    void mul() {
        cout << "Product of a, b, and c is: " << a * b * c << endl;
    }
};

int main() {
    D d;
     d.get_a();
    d.get_b();
    d.get_c();
    d.mul();
    return 0;
}
```

```
#include <iostream>
#include <vector>
using namespace std;

// Base Class
```

```
class Person {
protected:
   string name;
   int age;
public:
   Person() : name(""), age(0) {}
   Person(string n, int a) : name(n), age(a) {}

   void getDetails() const {
       cout << "Name: " << name << ", Age: " << age << endl;
   }

   virtual void printDetails() const {
       cout << "Person Details: " << endl;
       getDetails();
   }
};

// Derived Class Student (Single Inheritance)
class Student : public Person {
protected:
   int studentId;
   string major;
public:
   Student() : studentId(0), major("") {}
   Student(string n, int a, int id, string m) : Person(n, a),
studentId(id), major(m) {}

   void setMajor(string m) {
       major = m;
   }

   string getMajor() const {
       return major;
   }

   void printDetails() const override {
       cout << "Student Details: " << endl;
       getDetails();
```

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

// Derived Class Faculty (Single Inheritance)
class Faculty : public Person {
protected:
    string department;
    int employeeId;
public:
    Faculty() : department(""), employeeId(0) {}
    Faculty(string n, int a, int id, string dept) : Person(n, a),
employeeId(id), department(dept) {}

    void setDepartment(string dept) {
        department = dept;
    }

    string getDepartment() const {
        return department;
    }

    void printDetails() const override {
        cout << "Faculty Details: " << endl;
        getDetails();
        cout << "Employee ID: " << employeeId << ", Department: " <<
department << endl;
    }
};

// Derived Class TeachingAssistant (Multilevel Inheritance)
class TeachingAssistant : public Student {
protected:
    vector<string> coursesTeaching;
public:
    TeachingAssistant() : Student() {}
    TeachingAssistant(string n, int a, int id, string m, vector<string>
courses)
        : Student(n, a, id, m), coursesTeaching(courses) {}
```

```
    void setCoursesTeaching(vector<string> courses) {
        coursesTeaching = courses;
    }

    vector<string> getCoursesTeaching() const {
        return coursesTeaching;
    }

    void printDetails() const override {
        cout << "Teaching Assistant Details: " << endl;
        Student::printDetails();
        cout << "Courses Teaching: ";
        for (const string& course : coursesTeaching) {
            cout << course << " ";
        }
        cout << endl;
    }
};

// Derived Class ResearchAssistant (Hierarchical Inheritance)
class ResearchAssistant : public Person {
protected:
    string researchArea;
    string supervisor;
public:
    ResearchAssistant() : researchArea(""), supervisor("") {}
    ResearchAssistant(string n, int a, string ra, string sup)
        : Person(n, a), researchArea(ra), supervisor(sup) {}

    void setResearchArea(string ra) {
        researchArea = ra;
    }

    string getResearchArea() const {
        return researchArea;
    }

    void setSupervisor(string sup) {
        supervisor = sup;
```

```
    }

    string getSupervisor() const {
        return supervisor;
    }

    void printDetails() const override {
        cout << "Research Assistant Details: " << endl;
        getDetails();
        cout << "Research Area: " << researchArea << ", Supervisor: " <<
supervisor << endl;
    }
};

// Derived Class GraduateStudentTA (Hybrid Inheritance)
class GraduateStudentTA : public Student, public TeachingAssistant {
public:
    GraduateStudentTA(string n, int a, int id, string m, vector<string>
courses)
        : Student(n, a, id, m), TeachingAssistant(n, a, id, m, courses) {}

    void printDetails() const override {
        cout << "Graduate Student TA Details: " << endl;
        Student::printDetails();
        cout << "Courses Teaching: ";
        for (const string& course : getCoursesTeaching()) {
            cout << course << " ";
        }
        cout << endl;
    }
};

int main() {
    // Example usage
    Student student("sunny", 23, 12345, "cs");
    student.printDetails();

    Faculty faculty("rohit", 45, 6789, "Math");
    faculty.printDetails();
```

```
    vector<string> courses = {"CS101", "CS102"};
    TeachingAssistant ta("Alice Johnson", 22, 54321, "Electrical
Engineering", courses);
    ta.printDetails();

    ResearchAssistant ra("Bob Brown", 28, "Artificial Intelligence", "Dr.
Green");
    ra.printDetails();

    GraduateStudentTA gta("Charlie Lee", 24, 98765, "Physics", courses);
    gta.printDetails();

    return 0;
}
```

Function overloading

Function overloading is a feature in C++ that allows you to define multiple functions with the same name but with different parameter .

```
#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 << Cal::add(10, 20) << endl;
    cout << Cal::add(12, 20, 23) << endl;
    return 0;
```

```
}
```

```
#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;
   }
   static int subtract(int a, int b) {
       return a - b;
   }
   static int subtract(int a, int b, int c) {
       return a - b - c;
   }
   static int multiply(int a, int b) {
       return a * b;
   }
   static int multiply(int a, int b, int c) {
       return a * b * c;
   }
};
int main(void) {
   Cal C;
   cout << "addition: " << Cal::add(10, 20) << endl;
   cout << "addition: " << Cal::add(12, 20, 23) << endl;
   cout << "subtraction: " << Cal::subtract(20, 10) << endl;
   cout << "subtraction: " << Cal::subtract(50, 20, 10) << endl;
   cout << "multiplication: " << Cal::multiply(10, 20) << endl;
   cout << "multiplication: " << Cal::multiply(2, 3, 4) << endl;

   return 0;
}
```

Question task

```
/*
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>
using namespace std;
class Person {      // Base class Person
protected:
   string name;
   int id;
public:
   Person(const string& name, int id) : name(name), id(id) {}
   void getDetails() {
       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) {}
   void setMajor(const string& m) {
       major = m;
   }
   string getMajor() const {
       return major;
   }
   double calculateSemesterGPA(const vector<double>& grades) const {
       if (grades.empty()) return 0.0;
       double sum = 0.0;
       for(double grade : grades) {
           sum += grade;
       }
       return sum / grades.size();
   }
   void getDetails() const {
```

```
        Person::getDetails();
        cout << "Major: " << major << ", GPA: " << gpa << endl;
    }
};
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) {}

    void setDepartment(const string& d) {
       department = d;
    }
    string getDepartment()  {
        return department;
    }
    void teachCourse(const string& courseName) {
        cout << name << " is teaching " << courseName << " in the " <<
department << " department." << endl;
    }
    void getDetails()  {
        Person::getDetails();
        cout << "department is: " << department << ", title is: " << title
<< endl;
    }
};
int main() {
    Student s1("bal mukund meena", 5884, "cs", 7.6);           // Creating
Student object
    s1.getDetails();
    vector<double> grades = {3.5, 3.7, 3.8, 4.0};
    cout << "Semester GPA is: " << s1.calculateSemesterGPA(grades) << endl;
    Faculty f1("swetank", 12345, "CS", "TRAINER");                 //
Creating Faculty object
    f1.getDetails();
    f1.teachCourse("C++");
    return 0;
```

```
}
```

Access specifier program

```
#include <iostream>
using namespace std;

class Account {
public:
   float salary = 60000;
};

class Programmer: public Account {
public:
   float bonus = 5000;
};

int main() {
   Programmer p1;
   cout << "Salary: " << p1.salary << endl;
   cout << "Bonus: " << p1.bonus << endl;
   return 0;
}
```