National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2023, Lab Manual - 9

Course Code: CL1004	Course: Object Oriented Programming Lab
Instructor(s):	Shazia Paras Shaikh

Contents:

1. Inheritance
2. Is -A- Relationship
3. Modes Of Inheritance
4. Types Of Inheritance
    a. Single Inheritance
    b. MultiLevel Inheritance
    c. Multiple Inheritance
    d. Hierarchical Inheritance
    e. Hybrid Inheritance
5. Constructor Calls
6. Destructor Call
7. Lab Tasks

1. INHERITANCE

Capability of a class to derive properties and characteristics from another class is known as Inheritance. The existing class is called the base class (or sometimes super class) and the new class is referred to as the derived class (or sometimes subclass).

For e.g: The car is a vehicle, so any attributes and behaviors of a vehicle are also attributes and behaviors of a car.

Base class: It is the class from which features are to be inherited into another class.

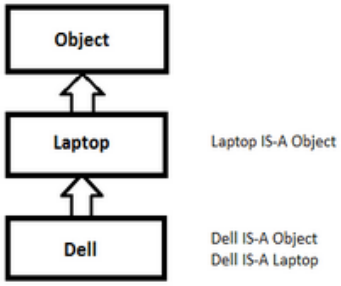
Derived class: It is the class in which the base class features are inherited. A derived class can have additional properties and methods not present in the parent class that distinguishes it and provides additional functionality.

```
class subclass_name : access_mode base_class_name

{
```

```
//body of subclass

};
```

2. IS-A-RELATIONSHIP

An object of a derived class also can be treated as an object of its base class.



3. MODES OF INTHERITANCE

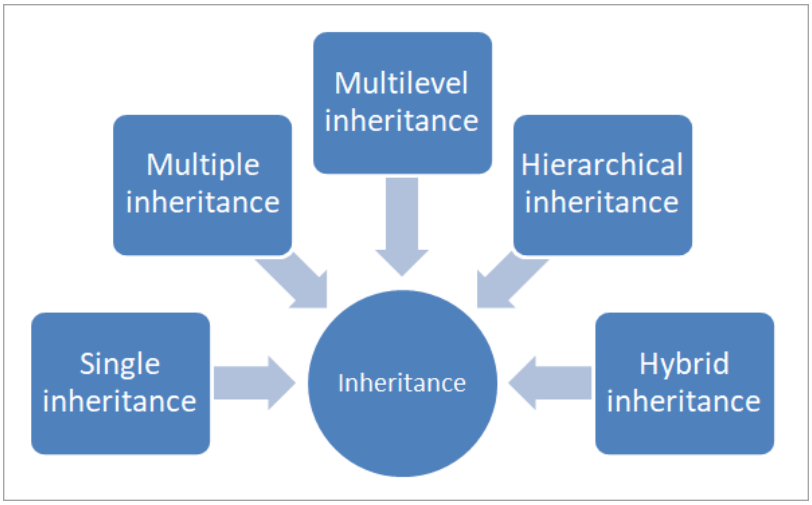
Public mode: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

Protected mode: If we derive a subclass from a Protected base class. Then both public members and protected members of the base class will become protected in the derived class.

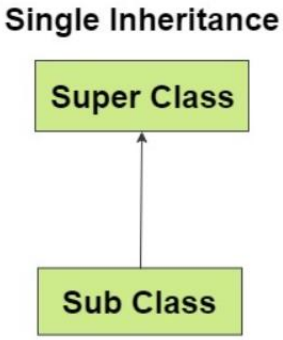
Private mode: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

Base class member access specifier	Type of Inheritence		
	Public	Protected	Private
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not accessible (Hidden)	Not accessible (Hidden)	Not accessible (Hidden)

4. TYPES OF INHERITANCE



4.1. SINGLE INHERITANCE



In single inheritance, a class is allowed to inherit from only one class.

i.e. one subclass is inherited by one base class only.

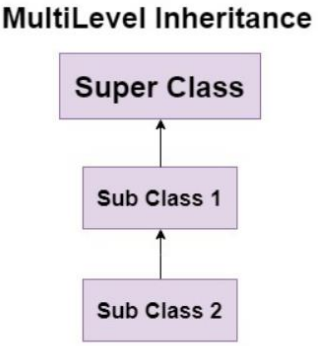
```
class subclass_name : access_mode base_class
{
        //body of subclass
};
```

Example Code:

```
#include <iostream>
using namespace std;
class Person
{
char name[100],gender[10];
int age;
public:
void getdata()
{
cout<<"Name: "; cin>>name; cout<<"Age: "; cin>>age; cout<<"Gender: ";
cin>>gender;
}
void display()
{
cout<<"Name: "<<name<<endl; cout<<"Age: "<<age<<endl; cout<<"Gender: "<<gender<<endl;
} };
class Employee: public Person
{
char company[100]; float salary; public: void getdata()
```

```
{
Person::getdata(); cout<<"Name of Company: "; cin>>company; cout<<" Salary: Rs.";
    cin>>salary;  }
void display()
    {
Person::display(); cout<<"Name of Company:"<<company<<endl; cout<<"Salary:
    Rs."<<salary<<endl;  }
};
int main()
{
Employee emp;
cout<<"Enter data"<<endl; emp.getdata();
cout<<endl<<"Displaying data"<<endl; emp.display();
return 0;  }
```

4.2. MULTILEVEL INHERITANCE



Multilevel inheritance is a process of deriving a class from another derived class.

Example Code:

```
#include <iostream>
using namespace std;
class Person
{
char name[100],gender[10];
int age;
public:
void getdata()
{
cout<<"Name: "; cin>>name; cout<<"Age: "; cin>>age; cout<<"Gender: ";
cin>>gender;
}

void display()
{
cout<<"Name: "<<name<<endl; cout<<"Age: "<<age<<endl;
cout<<"Gender: "<<gender<<endl;
}
```

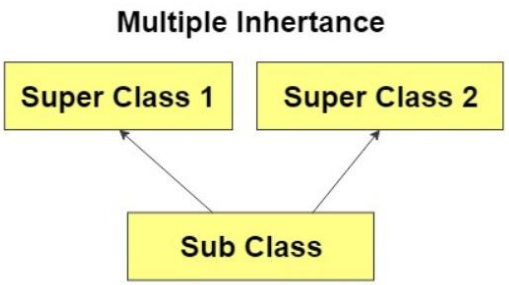
```
};
class Employee: public Person
{
char company[100]; float salary; public: void getdata()
{
Person::getdata(); cout<<"Name of Company: "; cin>>company; cout<<" Salary: Rs.";
    cin>>salary;
}
void display()
{
Person::display();
cout<<"Name of Company:"<<company<<endl; cout<<"Salary: Rs."<<salary<<endl;
}
};
class Programmer: public Employee
{
int number; public: void getdata()
{
Employee::getdata();
cout<<"Number of programming language known: "; cin>>number;
}
void display()
{
Employee::display();
cout<<"Number of programming language known:"<<number;
}
};
int main()
{
Programmer p;
cout<<"Enter data"<<endl; p.getdata(); cout<<endl<<"Displaying data"<<endl; p.display();
return 0;
}
```

4.3. MULTIPLE INHERITANCE

In Multiple Inheritance a class can inherit from more than one class.

i.e one sub class is inherited from more than one base class.

```
class subclass_name : access_mode base_class1,
access_mode base_class2, ....
{
                //body of subclass
};
```

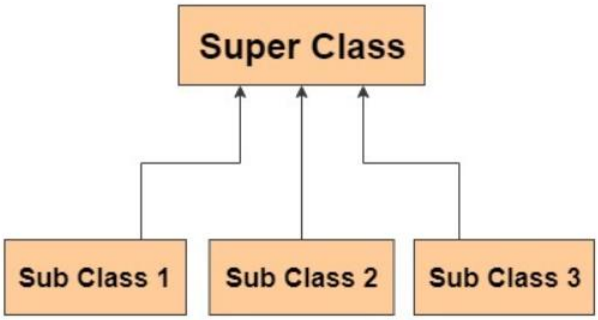


Example Code:

```
#include <iostream>
using namespace std;
class Account {
public:
float salary = 60000;
void display(){
        cout<<salary; }
};

class Languages {
public:
string language1 = "C++";
void display(){
        cout<<language1; }
};

class Programmer: public Account, public Languages {
public:
float bonus = 5000; };
int main(void) {
Programmer p1;
p1.Languages :: display();
p1.Account :: display();
return 0; }
```

4.4. HIERARCHICAL INHERITANCE

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

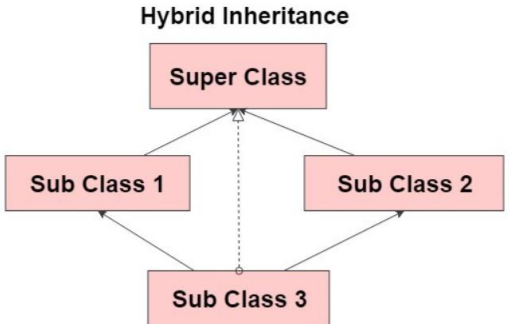


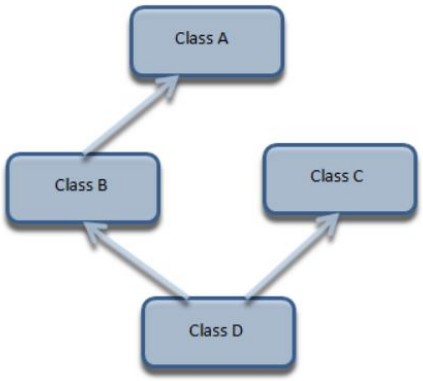
Example Code:

```
#include <iostream>
using namespace std;
class Account {
public:
float salary = 60000; };

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

class HR: public Account {
public:
float bonus = 1000; };

int main(void) {
Programmer p1;
HR h1;
cout<<"Programmer:  "<<p1.salary +
p1.bonus<<endl;
cout<<"HR : "<<h1.salary +h1.bonus<<endl;
return 0; }
```

4.5. HYBRID INHERITANCE

Hybrid inheritance is a combination of more than one type of inheritance.
For example: Combining Hierarchical inheritance and Multiple Inheritance.





Example Code:

```
#include<iostream>
using namespace std;
class Student
{
protected:
int rno; public:
void get_no(int a)
{ rno=a;
}
void put_no(void)
{
cout<<"Roll no"<<rno<<"\n";
}
};
class Test:public Student
{
protected:
float part1,part2; public: void get_mark(float x,float y)
{
part1=x; part2=y;
}
void put_marks()
{
cout<<"Marksobtained:\npart1="<<part1<<"\n"<<"part2="<<part2 <<"\n";}
};
class Sports
{
protected: float score; public: void getscore(float s)
{ score=s;
}
void putscore()
{ cout<<"sports:"<<score<<"\n";
} };
class Result: public Test, public Sports
{ float total; public:
void display()
{
total=part1+part2+score; put_no(); put_marks(); putscore();
cout<<"Total Score="<<total<<"\n";
} };
int main()
{
Result stu; stu.get_no(123); stu.get_mark(27.5,33.0); stu.getscore(6.0);
stu.display();
return 0;
}
```

---

5. CONSTRUCTOR CALLS

The constructor of a derived class is required to create an object of the derived class type. As the derived class contains all the members of the base class, the base sub-object must also be created and initialized. The base class constructor is called to perform this task. Unless otherwise defined, this will be the default constructor.

The order in which the constructors are called is important. The base class constructor is called first, then the derived class constructor. The object is thus constructed from its core outwards.

6. DESTRUCTOR CALLS

When an object is destroyed, the destructor of the derived class is first called, followed by the destructor of the base class. The reverse order of the constructor calls applies. You need to define a destructor for a derived class if actions performed by the constructor need to be reversed. The base class destructor need not be called explicitly as it is executed implicitly.

BASE CLASS INITIALIZER WITH SINGLE INHERITANCE

```
#include<iostream>
#include<string>
using namespace std;
class Account
{ private:
long accountNumber;     // Account number
protected:
   string name;     // Account holder
public:     //Public interface:
const string accountType;     // Account Type
Account(long accNumber, string accHolder, const string& accType)
: accountNumber(accNumber), name(accHolder), accountType(accType)    // use of initializer list to
// initialize the data members
{ cout<<"Account's constructor has been called"<<endl<<endl;
}
~ Account()   //Destructor
{
cout<<endl<<"Object Destroyed";
}
const long getAccNumber() const   // read-only accessor for privately defined data menber;
accountNumber
{ return accountNumber;
}
void DisplayDetails()
{
cout<<"Account Holder: "<<name<<endl; cout<<"Account Number: "<<accountNumber<<endl;
cout<<"Account Type: "<<accountType<<endl;
}
};
```

```
class CurrentAccount : public  Account  //Single Inheritance
{ private:
double balance; public:
CurrentAccount(long accNumber, const string& accHolder, string accountType, double accBalance)
: Account(accNumber, accHolder, accountType), balance(accBalance)   // inheritance with
//constructor
{ cout<<"CurrentAccount's constructor has been called"<<endl<<endl;
}
void deposit_currbal()
{
float deposit;
cout<<"Enter amount to Deposit : "; cin>>deposit; cout<<endl; balance = balance + deposit;
}
void Display()
{
name = "Dummy"; //can change protected data member of Base class DisplayDetails();
cout<<"Account Balance: "<<balance<<endl<<endl;
}
};
int main()
{
CurrentAccount currAcc(7654321,"Dummy1", "Current Account", 1000);
currAcc.deposit_currbal();
currAcc.Display();
return 0;
}
```

BASE CLASS INITIALIZER WITH MULTIPLE INHERITANCE

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

class FirstBase
{
protected: int a; public:
FirstBase(int x)
{
cout<<"Constructor of FirstBase is called: "<<endl; a=x;
}
};

class SecondBase
{
protected: string b; public:
SecondBase(string x)
{
cout<<"Constructor of SecondBase is called: "<<endl; b=x;
} };
class Derived : public FirstBase, public SecondBase
{ public:
Derived(int a,string b):
FirstBase(a),SecondBase(b)
{
```

```
cout<<"Child Constructor is called: "<<endl;
}

void display()
{ cout<<a<<" "<<b<<endl;
}
};

int main()
{
Derived obj(24,"Multiple Inheritance"); obj.display();
}
```

/*The Derived class has a constructor that takes an integer and a string argument. The constructor calls the constructors of the base classes FirstBase and SecondBase using the initialization list, passing the corresponding arguments to them. Then, it prints a message to indicate that the child constructor is called. The display() function simply prints the values of the a and b data members.

In terms of inheritance, the Derived class is inheriting from FirstBase and SecondBase using multiple inheritance. This means that the Derived class has access to the protected members of both base classes. The Derived constructor is calling the constructors of both base classes using the initialization list. This ensures that the base class objects are properly initialized before the derived class constructor runs.

*/

Lab Tasks:

1 Create a class hierarchy for a car dealership. Start with a base class called Car and create subclasses for specific types of cars, such as Sedan and SUV.

Data members of classes:

- Car
  - brand (string): the brand of the car
  - model (string): the model of the car
  - price (float): the price of the car in USD
  - color (string): the color of the car
  - features (list): a list of Feature objects representing the features of the car
- Sedan
  - fuel_type (string): the type of fuel the sedan uses
  - engine_capacity (float): the engine capacity of the sedan in liters
- SUV
  - seating_capacity (int): the maximum number of passengers the SUV can seat
  - cargo_capacity (float): the maximum cargo capacity of the SUV in cubic feet

Methods:

- Car
  - Default and parameterized constructor
  - add_feature(self, feature): adds a Feature object to the car's list of features
  - get_features(self): returns a list of Feature objects representing the car's features
- Sedan

○ Default and parameterized constructor
○ start_engine(self): starts the engine of the sedan
○ stop_engine(self): stops the engine of the sedan
● SUV
○ Default and parameterized constructor
○ open_trunk(self): opens the trunk of the SUV
○ close_trunk(self): closes the trunk of the SUV

2- A company called TechCo needs a software system to manage its employees. The system should include classes to represent employees, departments, and managers. Each employee belongs to a specific department, and each department has one or more managers.

● Create a base class called Employee that includes the following properties:

○ employee_id (int): the unique ID assigned to the employee

○ name (string): the name of the employee

○ salary (float): the salary of the employee

○ position (string): the position of the employee

○ department (string): the name of the department the employee belongs to

○ manager (string): the name of the employee's manager

● Create a subclass called Department that inherits using protected mode from the Employee class and includes the following properties:

○ department_id (int): the unique ID assigned to the department

○ department_name (string): the name of the department

○ employees (list): a list of employees in the department

○ manager (string): the name of the department manager

● Create a subclass called Manager that inherits from Department and includes the following properties:

○ manager_id (int): the unique ID assigned to the manager

○ manager_name (string): the name of the manager

○ subordinates (list): a list of employees managed by the manager

● Create methods for each class to manage the employees:

○ Employee:

■ Default and parameterized constructor

■ assign_department(self, department): assigns the employee to the specified department

○ Department:

■ Default and parameterized constructor

■ add_employee(self, employee): adds an employee to the department

■ get_employees(self): returns a list of employees in the department

○ Manager:

■ Default and parameterized constructor

■ add_subordinate(self, employee): adds an employee to the manager's list of subordinates

■ get_subordinates(self): returns a list of subordinates managed by the manager.

● Create a program that demonstrates how the class hierarchy works. The program should allow the user to create new employees, departments, and managers, add and remove employees from departments, and add and remove subordinates from managers. The program should also allow the user to view the employees in a specific department, view the subordinates of a specific manager, and search for employees by various criteria (such as name, position, and salary).

3. Create four classes: Vehicle, Car, Truck, and Motorcycle with the following specifications:

● The class Vehicle should have data members make, model, and year.

● The classes Car, Truck, and Motorcycle should inherit from the class Vehicle.

● The Car class should have an attribute num_doors indicating the number of doors on the car. The function getdata should get the input from the user of Car Class. The function putdata should print the data of Car Class.

● The Truck class should have an attribute payload_capacity indicating the maximum weight the truck can carry. The function getdata should get the input from the user of Truck Class. The function putdata should print the data of Truck Class.

● The Motorcycle class should have an attribute engine_size indicating the size of the motorcycle's engine in cc. The function getdata should get the input from the user of Motorcycle Class. The function putdata should print the data of Motorcycle Class.

● Sequential id's should be assigned to each object being created of the four classes starting from 1.

4. Suppose there are two types of students in a school: regular students and honors students. Both types of students have a name, age, and student ID, but honors students have additional attributes such as GPA and a list of honors classes taken.

In this scenario, we can use inheritance to create a derived class "HonorsStudent" that inherits from the "Student" base class. The "HonorsStudent" class will have additional attributes and behaviors specific to honors students, while still inheriting the common attributes and behaviors from the "Student" base class.

What type of inheritance should be used in this scenario, and how can constructors, accessor functions, and mutator functions be implemented for the derived class?