National University of Computer & Emerging Sciences, Karachi

Computer Science Department

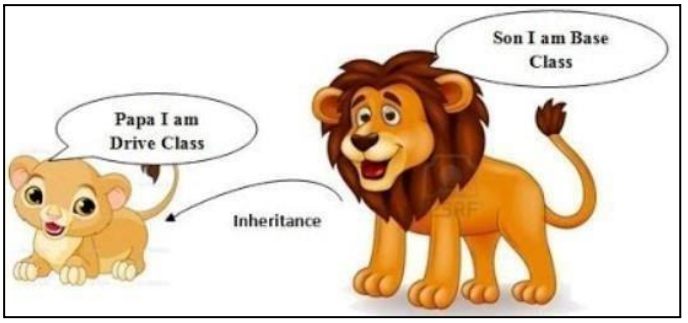
Spring 2023, Lab Manual - 8

Course Code: CL1004	Course: Object Oriented Programming Lab

Contents:

1. INHERITANCE

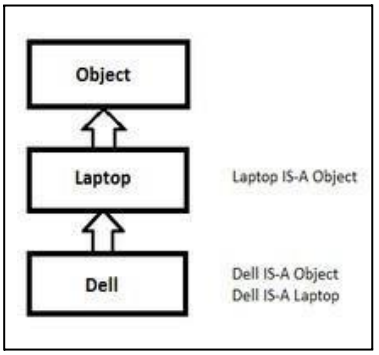
Capability of a class to derive properties and characteristics from another class is known as Inheritance. The existing class is called the base class (or sometimes super class) and the new class is referred to as the derived class (or sometimes subclass). For e.g: The car is a vehicle, so any attributes 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

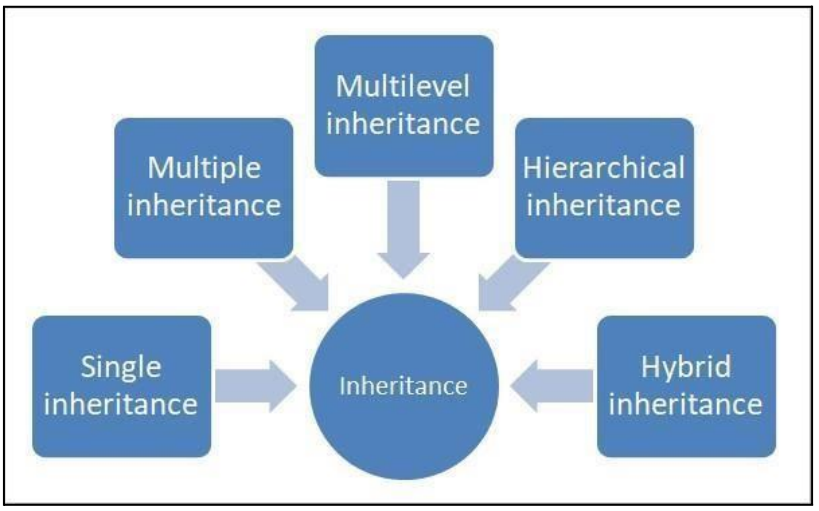
Public mode: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

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.

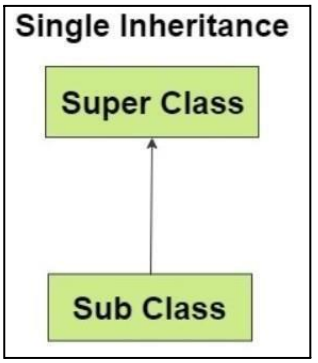
Private mode: If we derive a subclass from a Private base class. Then both public members and protected members of the base class will become Private in the derived class.

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
        };
```

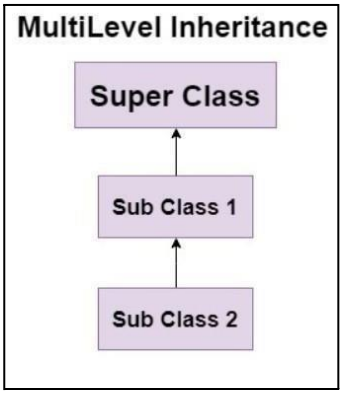
Example Code:

```
#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.



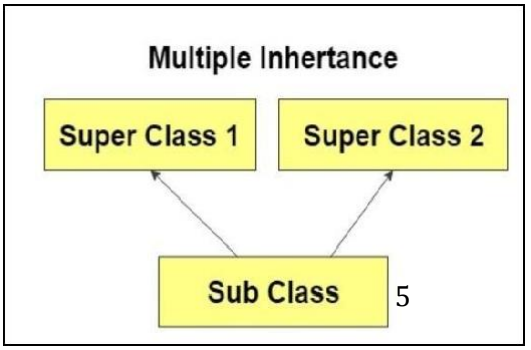
Example Code:

```
#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

};
```

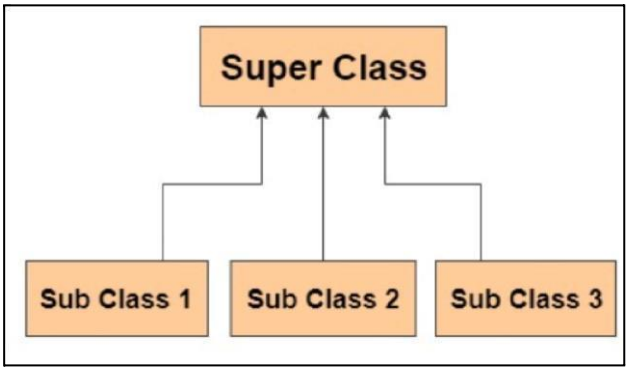
Example Code:

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

return 0; }
```

Account, public

4.4. HIERARCHICAL INHERITANCE

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



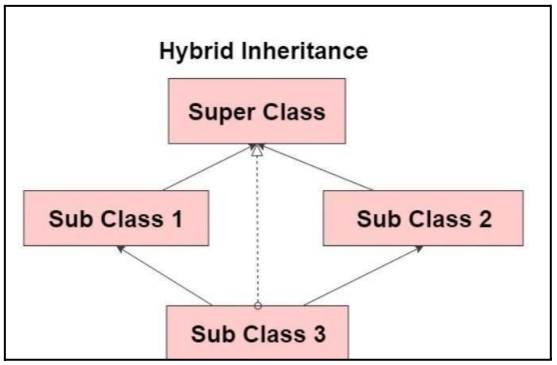
Example Code:

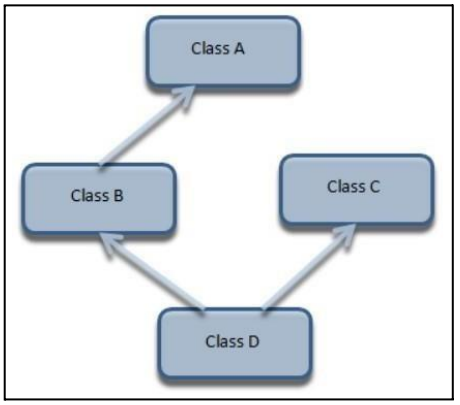
```
#include <iostream>
using namespace std;
class Account {
public: float salary =
60000; };

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

int main(void) {
Programmer p1;
HR h1;
cout<<"Programmer:      "<<p1.salary      +
```

```
float bonus = 5000; };                    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";
```

```
}
}
;
i
n
t
main()
{
Result stu; stu.get_no(123);
stu.get_mark(27.5,33.0); stu.getscore(6.0);
stu.display(); return 0; }
```

---

5. CONSTRUCTORCALLS

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. DESTRUCTORCALLS

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)
{ cout<<"Account's constructor has been called"<<endl<<endl;
}
~ Account() //Destructor
{
cout<<endl<<"Object Destroyed";
}
const long getAccNumber() const      //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)
{ 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();}
```

Exercise:

Q1: In the database only three kinds of employees are represented. Managers manage, scientists perform research to develop better widgets, and laborers operate the dangerous widget-stamping presses.
The database stores a name and an employee identification number for all employees, no matter what their category is. However, for managers, it also stores their titles and golf club dues. For scientists, it stores the number of scholarly articles they have published and their title. Laborers store the title only. You must start with a base class employee. This class handles the employee's last name and employee number. From this class three other classes are derived: manager, scientist, and laborer. All three classes contain additional information about these categories of employee, and member functions to handle this information as shown in figure.

a) Define the parent "Employee" class with the following attributes.
   - name(string)
   - number(int)
   - age(int)

b) Create the "Manager" class with the following attributes.
   - title(string)
   - club dues

c) Create the "scientist" class with the following attributes.
   - title(string)
   - publication(string)

d) Create the "laborer" class with the following attributes.
   - title(string)

Q2: Account Inheritance Hierarchy Create an inheritance hierarchy that a bank might use to represent customers' bank accounts. All customers at this bank can deposit (i.e., credit) money into their accounts and withdraw (i.e., debit) money from their accounts. 1 More specific types of accounts also exist. Savings accounts, for instance, earn interest on the money they hold. Checking accounts, on the other hand, charge a fee per transaction (i.e., credit or debit).
Create an inheritance hierarchy containing base-class Account and derived classes Savings Account and Checking Account that inherit from class Account.

1) Base-class Account should include one data member of type double to represent the account balance. The

class should provide a constructor that receives an initial balance and uses it to initialize the data member.

- The constructor should validate the initial balance to ensure that it's greater than or equal to 0.0. If not, the balance should be set to 0.0 and the constructor should display an error message, indicating that the initial balance was invalid.
- **The class should provide three member functions.**
- Member function credit should add an amount to the current balance.
- Member function debit should withdraw money from the Account and ensure that the debit amount does not exceed the Account's balance. If it does, the balance should be left unchanged and the function should print the message" Debit amount exceeded account balance."
- Member function getBalance should return the current balance.

2) Derived-class Savings Account should inherit the functionality of an Account, but also include a data member of type double indicating the interest rate (percentage) assigned to the Account.
   - SavingsAccount's constructor should receive the initial balance, as well as an initial value for the SavingsAccount's interest rate.
   - SavingsAccount should provide a public member function calculateInterest that returns a double indicating the amount of interest earned by an account. Member function calculateInterest should determine this amount by multiplying the interest rate by the account balance.
     [Note: SavingsAccount should inherit member functions credit and debit as is without redefining them.]
3) Derived-class CheckingAccount should inherit from base-class Account and include an additional data member of type double that represents the fee charged per transaction.
   - CheckingAccount's constructor should receive the initial balance, as well as a parameter indicating a fee amount.
   - Class Checking Account should redefine member functions credit and debit so that they subtract the fee from the account balance whenever either transaction is performed successfully. CheckingAccount's versions of these functions should invoke the base-class Account version to perform the updates to an account balance.
   - CheckingAccount's debit function should charge a fee only if money is actually withdrawn (i.e., the debit amount does not exceed the account balance). [Hint: Define Account's debit function so that it returns a bool indicating whether money was withdrawn. Then use the return value to determine whether a fee should be charged.] After defining the classes in this hierarchy, write a program that creates objects of each class and tests their member functions. Add interest to the SavingsAccount object by first invoking its calculateInterest function, then passing the returned interest amount to the object's credit function.

Q3: Design a university management system that needs to model different types of employees, including Faculty members and administrative staff Design a class hierarchy using hybrid inheritance to represent these entities.

a) Create base class is "Employee" which contains common attributes.
   - name
   - age
   - aalary
   - empid

b) The derived class "Faculty" inherits from "Employee" and adds attributes specific to faculty members.
   - department
   - teaching_subjects

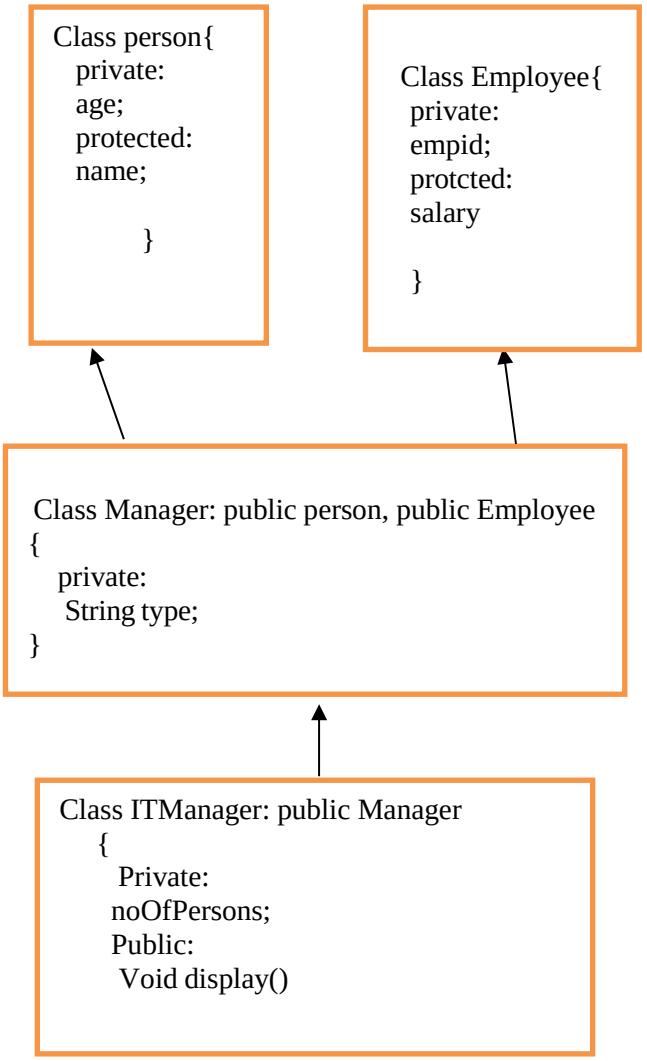
c) Another derived class, "AdministrativeStaff," also inherits from "Employee"
   - position

- office_location.

d) Now, create a third class called "TeachingAssistant" that inherits from both "Faculty" and "AdministrativeStaff." This class represents individuals who work as teaching assistants while pursuing their own studies.

Design the class hierarchy and provide the necessary attributes and methods for each class. Also, briefly explain the concept of hybrid inheritance and how it is utilized in this scenario

Q4: Implement the following scenario in C++:



1) The Display() function in “ITManager” should be capable of displaying the values of all the data members declared in the scenario (age,name,empId,salary,type,noOfPersons) without being able to alter the values.
2) The “int main()” function should contain only three program statements which are asfollows: In
a) the first statement, create object of “ITManager” and pass the values for all the datamembers: ITManager obj(age,name,empId,salary,type,noOfPersons);
b) In the second statement, call the Display() function.
c) In the third statement, return 0.

Q5: Inheritance hierarchy, we will be creating animals that consists of a smaller hierarchy than the previous problem. These classes include an Animal, Dog, and Cat. The Dog and Cat classes are both children of the Animal Class. The following requirements for each class.

1. **Class Animal:** Create a class named **Animal** that has the following instance properties and public interface.
    - **a.** Instance Properties:
        - **I.** Int age – the age of the animal
        - **II.** string size- the size of animal
    - **b.** Public Interface:
        - **I.** Constructor: Two constructors need to provide, namely a default constructor and a workhorse constructor. The default constructor will initialize the instance properties t their default values (number to 0 and string to “”)
        - **II.** Getters and setters for each instance property.
2. **Class Dog:** Create a class named **Dog** that is child of **Animal** has the following instance properties and public interface.
    - **a.** Instance Properties:
        - **I.** string breed – the breed of the dog.
    - **b.** Public Interface:
        - **II.** Constructor: Two constructors need to provide, namely a default constructor and a workhorse constructor. The default constructor will initialize the instance properties t their default values (number to 0 and string to “”). Note you must utilize the super() constructor call appropriately to set the parent’s instance property.
        - **III.** Getters and setters for each instance property.
3. **Class Cat:** Create a class named **Cat** that is child of **Animal** has the following instance properties and public interface.
    - **a.** Instance Properties:
        - **I.** int numberOfLives—the number of lives the cat left
    - **b.** Public Interface:
        - **II.** Constructor: Two constructors need to provide, namely a default constructor and a workhorse constructor. The default constructor will initialize the instance properties t their default values (number to 0 and string to “”). Note you must utilize the super() constructor call appropriately to set the parent’s instance property.
        - **III.** Getters and setters for each instance property.