National University of Computer & Emerging Sciences, Karachi
Computer Science Department
Spring 2024, Lab Manual - 7

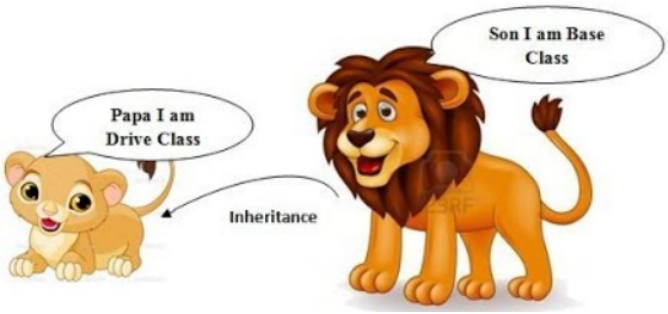
Course Code: CL1004	Course: Object Oriented Programming Lab
Instructor(s):	Muhammad Ali

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).

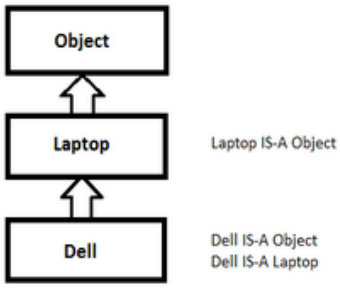
For e.g: The car is a vehicle, so any attributes and behaviors of a vehicle are also attributes and behaviors of a car.



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
```


Object
Laptop
Laptop IS-A Object
Dell
Dell IS-A Object
Dell IS-A Laptop

```
};
```

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

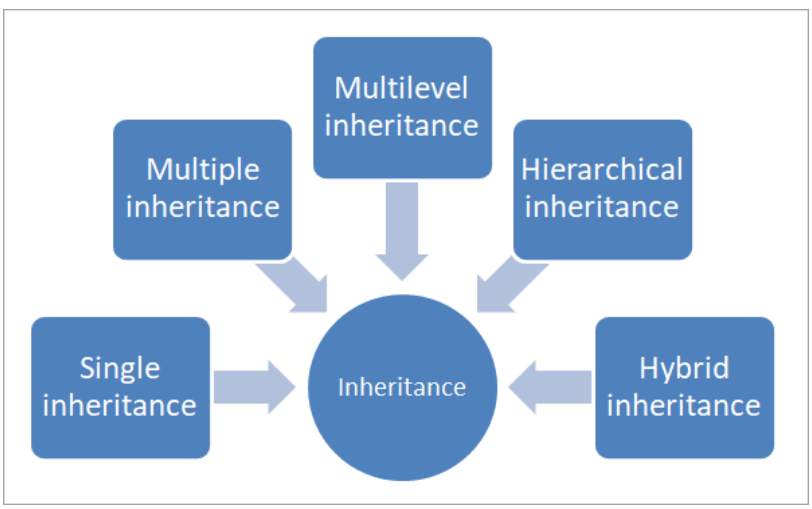
Public mode: If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.

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)

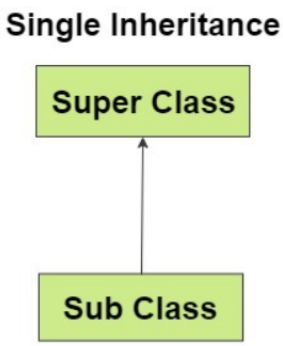
4. TYPES OF INHERITANCE



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
};
```

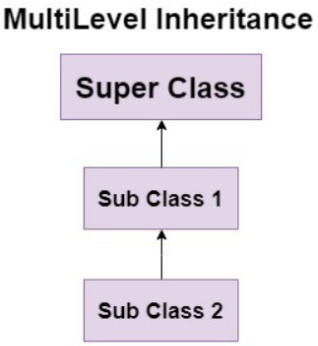
Example Code:

```
#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;
}
```

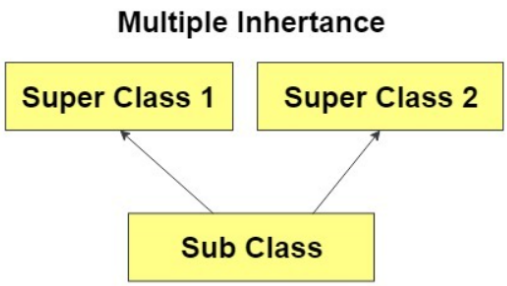
```
void display()
{
Person::display();
cout<<"Name of Company:"<<company<<endl; cout<<"Salary: Rs."<<salary<<endl;
}
};
class Programmer: public Employee
{
int number; public: void getdata()
{
Employee::getdata();
cout<<"Number of programming language known: "; cin>>number;
}
void display()
{
Employee::display();
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
};
```



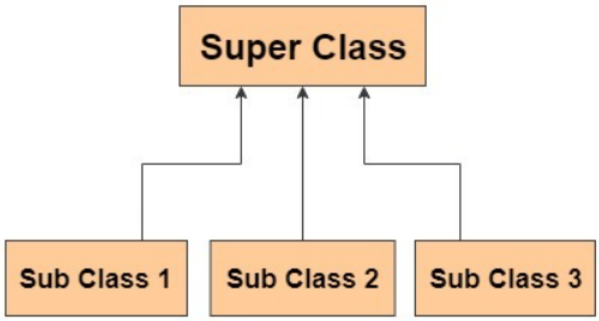
Example Code:

```
#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.

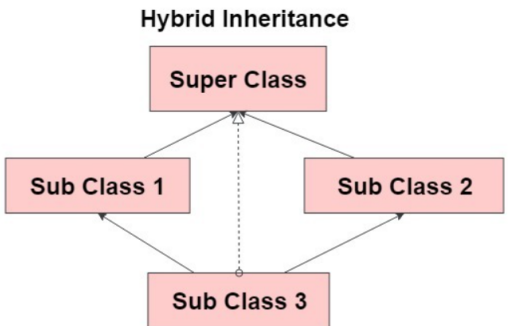


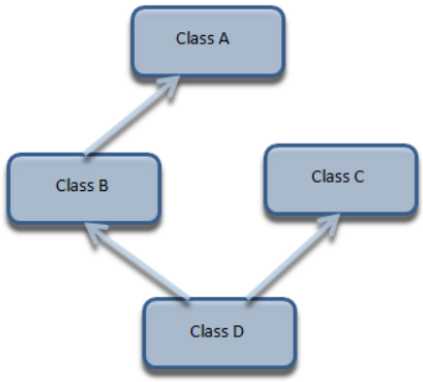
Example Code:

```
#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)
{ 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();
}
```

Lab Tasks:

1 Create a class hierarchy for a restaurant. Start with a base class called Restaurant and create subclasses for specific types of restaurants, such as Italian and Mexican.

Data members of classes:

- Restaurant
  - name (string): the name of the restaurant
  - address (string): the address of the restaurant
  - phone_number (string): the phone number of the restaurant
  - rating (float): the rating of the restaurant (out of 5)
  - menu (list): a list of MenuItem objects representing the menu of the restaurant
- ItalianRestaurant
  - pasta_type (string): the type of pasta the restaurant specializes in
  - pizza_type (string): the type of pizza the restaurant specializes in
- MexicanRestaurant
  - spice_level (int): the level of spiciness in the restaurant's dishes (out of 10)
  - salsa_type (string): the type of salsa the restaurant specializes in

Methods:

- Restaurant
  - Default and parameterized constructor
  - add_menu_item(self, item): adds a MenuItem object to the restaurant's menu
  - get_menu(self): returns a list of MenuItem objects representing the restaurant's menu
- ItalianRestaurant
  - Default and parameterized constructor
  - make_pasta(self, type): prepares a pasta dish with the specified type of pasta
  - make_pizza(self, type): prepares a pizza with the specified type of pizza
- MexicanRestaurant
  - Default and parameterized constructor
  - make_taco(self, filling): prepares a taco with the specified filling
  - make_burrito(self, filling): prepares a burrito with the specified filling

2- Daraz has hired you to create a software system to manage its inventory. The system should include classes to represent products, categories, and suppliers. Each product belongs to a specific category, and one or more suppliers supply each category.

- Create a base class called Product that includes the following properties
    - product_id (int): the unique ID assigned to the product
    - product_name (string): the name of the product
    - price (float): the price of the product
    - quantity (int): the number of units of the product in stock
    - category (string): the category the product belongs to
    - supplier (string): the name of the supplier that provides the product
- Create a subclass called Category that inherits using protected mode from the Product class and includes the following properties:
    - category_id (int): the unique ID assigned to the category
    - category_name (string): the name of the category
- Create a subclass called Supplier that inherits from Category and includes the following properties:
    - supplier_id (int): the unique ID assigned to the supplier
    - supplier_name (string): the name of the supplier
    - products (list): a list of products supplied by the supplier
- Create methods for each class to manage the inventory:
    - Product:
        - Default and parameterized constructor
        - add_stock(self, quantity): adds a specified number of units to the product's stock
    - Category:
        - Default and parameterized constructor
        - add_product(self, product): adds a product to the category
        - get_products(self): returns a list of products in the category
    - Supplier:
        - Default and parameterized constructor
        - add_product(self, product): adds a product to the supplier's inventory
        - get_products(self): returns a list of products supplied by the supplier
- Create a program that demonstrates how the class hierarchy works. The program should allow the user to create new products, categories, and suppliers, add and remove products from categories, and add and remove products from suppliers. The program should also allow the user to view the inventory for a specific category or supplier, and search for products by various criteria (such as name, price, and quantity).

3. Create four classes animal, griffin, lion, and eagle with the following specifications:

- The class Animal should have data members' names and ages.
- The classes Lion and Eagle should inherit from the class Animal.
- The Eagle class should have two integer members, wings and a head. The function getdata should get the input from the user of Eagle Class. The function putdata should print the data of Eagle Class.

- Lion Class should have teeth attribute. The function getdata should get the input from the user of Lion Class. The function putdata should print the data of Lion Class.
- A griffin is known to be a mythical beast having the body of a lion and the wings and head of an eagle.
- Sequential id's should be assigned to each object being created of the four classes starting from 1 .

4. PSL franchise has players who are affiliated with it throughout the season. Some are batsmen and some are bowlers. All of the affiliated players are collectively known as players. An all-rounder is both a batsman and a bowler. All the players have a name, age, and the amount for which he has been sold. Moreover, A batsman has attributes; role (opener, middle order, or hard hitter), and A bowler has an attribute; bowling type (Fast, Spin or other). An all-rounder is capable of displaying all the data with respect to roles. Identify the type of inheritance that needs to be implemented in the above scenario. Also, implement the scenario and create an object of an all-rounder (only) in the main by adding appropriate constructors, accessor functions, and mutator functions.