NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES

OBJECT ORIENTED PROGRAMMING - LAB

Instructors: Muhammad Sudais

Email: muhammad.sudais.v@nu.edu.pk

LAB 06

Outline

- Inheritance
- Real World Examples
- Mode of Inheritance
- Types of Inheritance
- Single Inheritance
- Multi Level Inheritance

INHERITANCE

Classes can share, obtain or "inherit" properties and methods that belong to existing classes.
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.

REAL WORLD EXAMPLE

Let's consider the Windows operating system. Windows 98 had certain properties and methods that were used in Windows XP. Windows XP was derived from Windows 98, but it was still different from it. Windows 7 was derived from Windows XP, but they were both different. They both followed a certain basic template and shared properties, however.

SYNTAX (INHERITANCE)

```
class derived-class-name : access base-class-name { // body of class };
```

TYPES OF INHERITANCE WITH RESPECT TO BASE CLASS ACCESS CONTROL

1. Public
2. Private
3. Protected

PUBLIC INHERITANCE

With public inheritance, every object of a derived class is also an object of that derived class's base class. However, base class objects are not objects of their derived classes.
For example, if we have Flower as a base class and Rose as a derived class, then all Roses are Flowers, but not all Flowers are Roses—for example, a Flower could also be Jasmine or a Tulip.

IS-A RELATIONSHIP

Inheritance is represented by an is-a relationship which means that an object of a derived class also can be treated as an object of its base class for example, a Rose is a Flower, so any attributes and behaviors of a Flower are also attributes and behaviors of a Rose.

PUBLIC INHERITANCE SYNTAX

class Rose : **public** Flower

TABLE SHOWING BASE CLASS ACCESS CONTROL (PUBLIC PRIVATE AND PROTECTED)

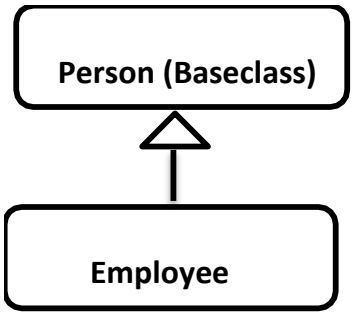
Visibility Of Base Class members	PUBLIC INHERITANCE	PRIVATE INHERITANCE	PROTECTED INHERITANCE
PUBLIC	PUBLIC in derived class	PRIVATE in derived class	PROTECTED in derived class
PRIVATE	HIDDEN in derived class	HIDDEN in derived class	HIDDEN in derived class
PROTECTED	PROTECTED in derived class	PRIVATE in derived class	PROTECTED in derived class

TYPES OF INHERITANCE WITH RESPECT TO DERIVED CLASSES

- SINGLE INHERITANCE
- HIERARICAL
- MULTILEVEL
- MULTIPLE INHERITANCE
- HYBRID INHERITANCE

SINGLE INHERITANCE

It is the type of inheritance in which there is one base class and one derived class.



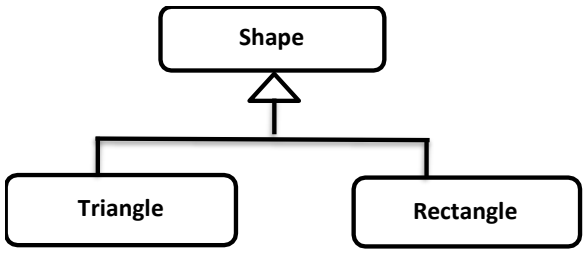
SINGLE INHERITANCE

```
#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;
}
```

HIERARICAL INHERITANCE

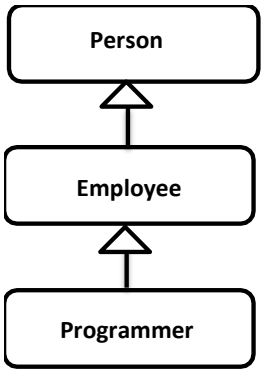
This is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes.



```
#include <iostream> using namespace std;
class Shape
{
        protected:
                float width, height;
        public:
                void set_data (float a, float b)
                {
                        width = a;
                        height = b;
                }
};
class Rectangle: public Shape
{
public:
        float area ()
         {
                return (width * height);
         }
};


class Triangle: public Shape  {
public:
        float area ()
        {
                return (width * height / 2);
        }
};
int main ()
{
        Rectangle rect;
        Triangle tri;
        rect.set_data (5,3);
        tri.set_data (2,5);
        cout << rect.area() << endl;
        cout << tri.area() << endl;
        return 0;
}
```

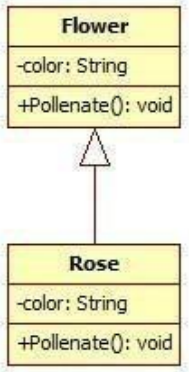
MULTI LEVEL INHERITANCE

When one class is derived from another derived class then this type of inheritance is called multilevel inheritance.




```
#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;
}
```

UML REPRESENTATION FOR INHERITANCE (GENERALIZATION)



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.

BASE CLASS INITIALIZER

C++ requires that a derived-class constructor call its base-class constructor to initialize the base-class data members that are inherited into the derived class. In a derive class, constructor uses a base class initialize list which uses a member initializer to pass arguments to the base-class. When a derived class constructor calls a base-class constructor, the arguments passed to the base-class constructor must be consistent with the number and types of parameters specified in one of the base-class constructors; otherwise, a compilation error occurs.

DESTROYING OBJECTS

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.

ADVANTAGES OF INHERITANCE

Reusability – Inheritance helps the code to be reused in many situations. The base class is defined and once it is compiled, it need not be reworked. Using the concept of inheritance, the programmer can create as many derived classes from the base class as needed while adding specific features to each derived class as needed.

Saves Time and Effort - The above concept of reusability achieved by inheritance saves the programmer's time and effort. Since the main code written can be reused in various situations as needed.

Maintainability - It is easy to debug a program when divided in parts.

EXERCISE

Question 1

Implement a single inheritance. While doing so make sure that your implementation conforms the below points.

1. Select any class from your course project. Create a derived class from the selected class.
2. Each member of your group must have different class selected. No 2 students will have same class(es).
3. Both the classes must have a static data member.
4. Both the classes must have constructor.
5. Both the classes must have a function with same definition but different implementation.
6. In the driver function (main), show the use of static data member and constructor of both the classes.

Question 2

Implement Multi Level Inheritance on the class selected in Question 1 by adding another level to derived class. New class should also conform the points mentioned previously.

Question 3

Implement the inheritance shown with all the members and access specifiers.

Each Function (including Constructor) should print class name and the details it is expected to print.

In the driver function (**main**), create one object of each class using constructor with following attributes.

Assume your id = k21**ABCD**.

- Vehicle v: Id = **A**, PlateNumber = KHI-**AB**
- RegisteredVehicle r: Id = **B**, RegisterationNumber = PK-**CD**
- Bike b: Id = **C**
- MotorBike m: Id = **D**

Call all the functions available for each class.

Hint: for Motorbike m, you will call all the functions but for RegisteredVehicle r you will only call functions of r and its parent(s) (in this case v is the only base class).