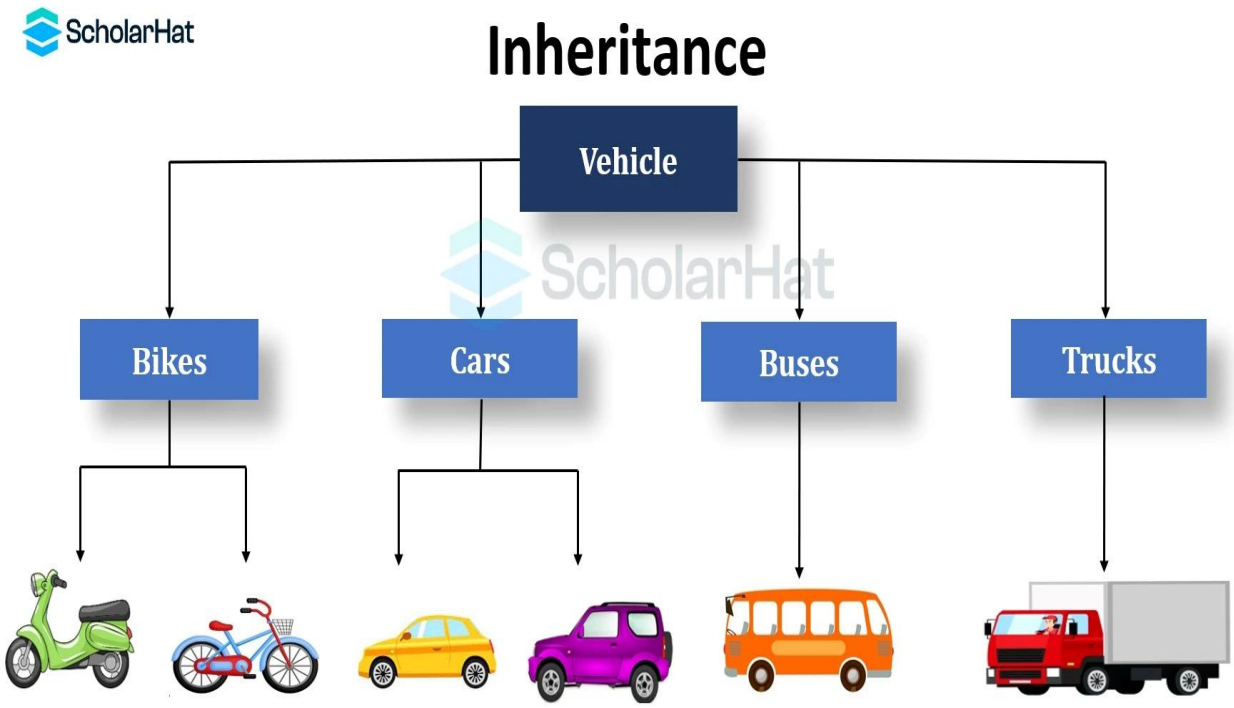
LECTURE-12

Inheritance

Reaccessability is yet another feature of OOP's. C++ strongly supports the concept of reusability. The C++ classes can be used again in several ways. Once a class has been written and tested, it can be adopted by another programmers. This is basically created by defining the new classes, reusing the properties of existing ones. The mechanism of deriving a new class from an old one is called **'INHERITANCE'**. This is often referred to as **'IS-A'** relationship because very object of the class being defined "is" also an object of inherited class. The old class is called **'BASE'** class and the new one is called **'DERIEVED'** class.



Defining Derived Classes

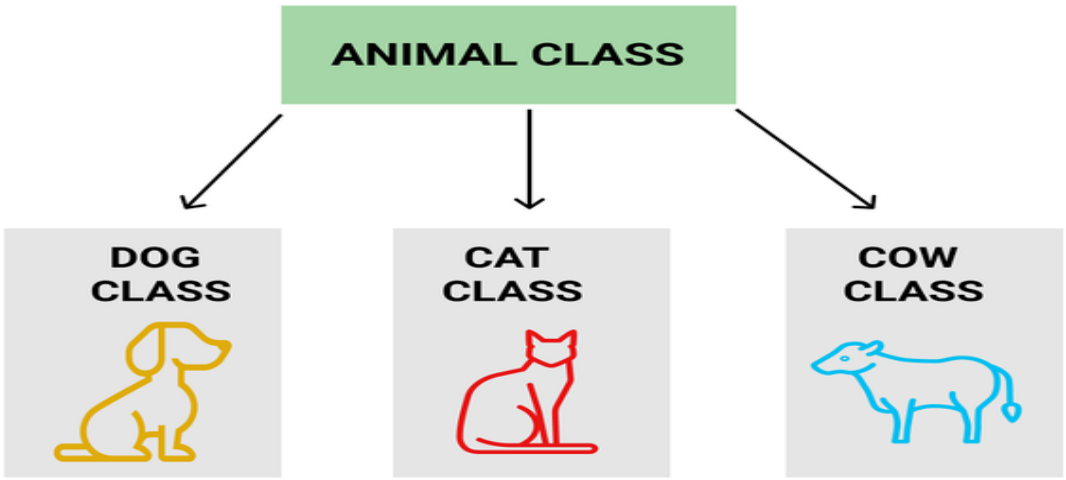
A derived class is specified by defining its relationship with the base class in addition to its own details. The general syntax of defining a derived class is as follows:

```
class  d_classname :  Access  specifier base class name
{
      // members of derived class
};
```

The colon indicates that the a-class name is derived from the base class name. The access specifier or the visibility mode is optional and, if present, may be public, private or protected. By default, it is private. Visibility mode describes the status of derived features e.g.

```
class xyz   //base  class
{
      members of xyz
};
class ABC: public xyz        //public derivation
{
      members of ABC
};
class ABC: XYZ        //private derivation (by default)
{
      members of ABC
};
```

In the inheritance, some of the base class data elements and member functions are inherited into the derived class. We can add our own data and member functions and thus extend the functionality of the base class. Inheritance, when used to modify and extend the capabilities of the existing classes, becomes a very powerful tool for incremental program development.




```cpp
#include <iostream>
// Base class
class Animal {
public:
    void eat() {
        std::cout << "Animal is eating" << std::endl;
    }

    void sleep() {
        std::cout << "Animal is sleeping" << std::endl;
    }
};
// Derived class
class Dog : public Animal {
public:
    void bark() {
        std::cout << "Dog is barking" << std::endl;
    }
};
int main() {
    // Create an object of the derived class
    Dog myDog;
    // Accessing inherited methods from the base class
    myDog.eat();
    myDog.sleep();
    // Accessing methods specific to the derived class
    myDog.bark();
}
```

Types of inheritance

In C++, inheritance can be implemented in following ways:

1. Single Inheritance
2. Multiple Inheritance
3. Multilevel Inheritance
4. Hierarchical Inheritance
5. Hybrid Inheritance

Making a Private Member Inheritable

Basically, we have visibility modes to specify that in which mode you are deriving another class from the already existing base class. They are:

- Private: when a base class is privately inherited by a derived class, 'public members' of the base class become private members of the derived class and therefore the public members of the base class can be accessed by its own objects using the dot operator. The result is that we have no member of base class that is accessible to the objects of the derived class.
- Public: On the other hand, when the base class is publicly inherited, 'public members' of the base class become 'public members' of derived class and therefore they are accessible to the objects of the derived class.
- Protected: C++ provides a third visibility modifier, protected, which serve a little purpose in the inheritance. A member declared as protected is accessible by the member functions within its class and any class immediately derived from it. It cannot be accessed by functions outside these two classes.

The below mentioned table summarizes how the visibility of members undergo modifications when they are inherited.

Base Class Visibility	Derived Class Visibility		
	Public	Private	Protected
Private	X	X	X
Public	Public	Private	Protected
Protected	Protected	Private	Protected

The private and protected members of a class can be accessed by:

- A function i.e. friend of a class.
- A member function of a class that is the friend of the class.
- A member function of a derived class.

Student Activity

1. Define Inheritance. What is the inheritance mechanism in C++?
2. What is the advantage of Inheritance?
3. What should be the structure of a class when it has to be a base for other classes?