DAY 03

Inheritance

What is inheritance ?

- Inheritance is a mechanism that allows a new class to be based on an existing class, inheriting its members (variables and functions) and adding additional functionality to it.
- The existing class is known as the base class or parent class, while the new class is known as the derived class or child class. The derived class inherits the properties of the base class, such as its data members and member functions, and can also add its own data members and member functions.

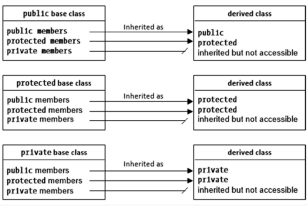
- Inheritance is an important feature in object-oriented programming for several reasons:

1. **Code reuse:** Inheritance allows you to reuse code from an existing class, reducing the amount of code you need to write from scratch. By inheriting from a base class, you can reuse its properties and methods, and only add or modify the parts that are specific to your derived class.
2. **Polymorphism:** Inheritance allows you to create a hierarchy of classes that share common functionality. This makes it possible to write code that can work with objects of different classes, as long as they share a common base class. This is known as polymorphism and it makes your code more flexible and adaptable to different situations.
3. **Abstraction:** Inheritance allows you to create more abstract and general classes that can be used as a basis for more specific classes. For example, you might create a general class for "animals", which can be inherited by more specific classes such as "dogs" or "cats". This helps you to organize your code and make it more modular.
4. **Extensibility:** Inheritance makes it easier to add new functionality to an existing class. By inheriting from a base class, you can add new properties and methods to your derived class without modifying the original code.

Modes of Inheritance :

There are 3 modes of inheritance.

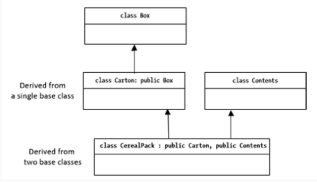
- **Public Mode:** If we derive a subclass from a public base class. Then the public member of the base class will become public in the derived class and protected members of the base class will become protected in the derived class.
- **Protected Mode:** If we derive a subclass from a Protected base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



Multiple inheritance :

- Multiple inheritance is a type of inheritance in which a derived class inherits from multiple base classes. In other words, a single derived class can have multiple parent classes.

- Multiple inheritance can be useful in situations where a single class needs to inherit behavior from multiple unrelated classes. However, it can also make the code more complex and harder to understand, especially if there are name conflicts between the inherited classes. In such cases, it is important to use access specifiers (public, private, and protected) to control the access to the inherited members and avoid name conflicts.



The diamond problem :

- It occurs when a class inherits from two or more base classes, which in turn share a common base class. This creates a diamond-shaped inheritance hierarchy, hence the name.

Problem

```
class A {
public:
   void foo() {
      cout << "A::foo()" << endl;
   }
};

class B : public A {
public:
   void foo() {
      cout << "B::foo()" << endl;
   }
};

class C : public A {
public:
   void foo() {
      cout << "C::foo()" << endl;
   }
};

class D : public B, public C {
public:
   // ...
};
```

Solution

```
class A {
public:
   void foo() {
      cout << "A::foo()" << endl;
   }
};

class B : virtual public A {
public:
   void foo() {
      cout << "B::foo()" << endl;
   }
};

class C : virtual public A {
public:
   void foo() {
      cout << "C::foo()" << endl;
   }
};

class D : public B, public C {
public:
   // ...
};
```

- Now, if we call the foo method on an object of class D, which implementation of the foo method should be used? Should it be B::foo or C::foo? The answer is not clear because both methods inherit A::foo and override it with their own implementation.

```
   A
  / \
 B   C
  \ /
   D
```

- This ambiguity is known as the diamond problem. It can cause errors and make the code difficult to understand and maintain. To solve the diamond problem, C++ uses virtual inheritance. Virtual inheritance ensures that there is only one instance of the common base class, even if it appears multiple times in the inheritance hierarchy. By using virtual inheritance, we can avoid the ambiguity and ensure that the correct version of the inherited member is used.

- By adding the virtual keyword to the base class in both B and C, we tell the compiler that there should be only one instance of A in the inheritance hierarchy. This ensures that there is no ambiguity when calling the foo method on an object of class D, and the correct version of the method is used.

You want: (Achievable with virtual inheritance)

```
   A
  / \
 B   C
  \ /
   D
```

And not: (What happens without virtual inheritance)

```
A   A
|   |
B   C
 \ /
  D
```