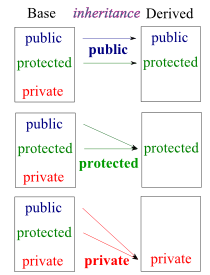
**Inheritance:**

**What ?**

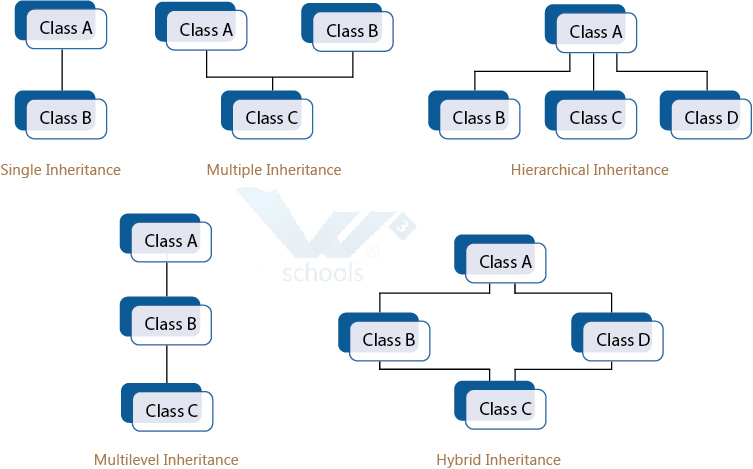
It is used to get the all the feature of the base class and have addition feature of its own in derived class.

A basic outline diagram about the inheritance on using with access specifier,



Through inheritance a user can create a new class from existing class.

Types of inheritance:



**Why ?**

1. Its used to combine the common functionality in a single class (base) and derived class are used to have a additional functionality which differentiates them from the base class.

2. Creating and maintaining the class is easier for inheritance.

**When ?**

When base class features are needed and additional functionality is needed to be added we go for inheritance.

**How ?**

*Program*

#include<iostream>

class Base

{

public:

Base()

{

std::cout << "Base Constructor\n";

}

~Base()

{

std::cout << "Base Destructor\n";

}

int fun()

{

std::cout << "Base::fun\n";

}

};

class Derived: public Base // Inheritance happens at this stage

{

public:

Derived()

{

std::cout << "Derived Constructor\n";

}

~Derived()

{

std::cout << "Derived Destructor\n";

}

int fun1()

{

std::cout << "Derived::fun1\n";

}

};

int main()

{

Derived d;

d.fun();

d.fun1();

return 0;

}

*Output:*

*Base Constructor*

*Derived Constructor*

*Base::fun*

*Derived::fun1*

*Derived Destructor*

*Base Destructor*

**Virtual**:

**What?**

Virtual is always related to inheritance and polymorphism. With virtual method or function the behaviour can be overridden within an inheriting class by function with same signature. Virtual function is generally used for customization of the derived class implementation.

A virtual method implementation is determined at runtime based on the actual type of the invoking object.

**Why?**

There is a necessity to use the single pointer to refer to all the objects of the different classes. So, we create the pointer to the base class that refers to all the derived objects. But, when base class pointer contains the address of the derived class object, always executes the base class function. This issue can only be resolved by using the 'virtual' function.

**When?**

Virtual functions allow us to create a list of base class pointers and call methods of any of the derived classes without even knowing kind of derived class object.

**How?**

Late binding (runtime) is done in accordance to the contents of the pointer.

Early binding (compile time) is done in accordance to the type of the pointer.

Working of virtual function:

If a class contains virtual keyword the compiler does two things,

1. If object of class is created then VPTR (virtual pointer) is created and inserted as data member of the class to point to VTABLE of the class. For each object created, a new VPTR is inserted as data member of the class.
2. Irrespective of the object created or not, a static array of pointer called VTABLE where each cell contain the address of each virtual function contained in the class.

Note

**Vtable:** A table of function pointers. It is maintained per class.  
**Vptr:** A pointer to vtable. It is maintained per object

**Disadvantage:**

* At runtime the machine has to decide which function call needs to be made, this degrades the performance.

**Program:**

**#01\_Pgm**