## what are Access Specifiers?

There are 3 access specifiers for a class/struct/Union in C++. These access specifiers define how the members of the class can be accessed. Of course, any member of a class is accessible within that class(Inside any member function of that same class). Moving ahead to type of access specifiers, they are:

**Public** - The members declared as Public are accessible from outside the Class through an object of the class.

**Protected** - The members declared as Protected are accessible from outside the class **BUT** only in a class derived from it.

**Private** - These members are only accessible from within the class. No outside Access is allowed.

An Source Code Example:

class MyClass

{

public:

int a;

protected:

int b;

private:

int c;

};

int main()

{

MyClass obj;

obj.a = 10; //Allowed

obj.b = 20; //Not Allowed, gives compiler error

obj.c = 30; //Not Allowed, gives compiler error

}

## Inheritance and Access Specifiers

Inheritance is C++ can be one of the following types:

* **Private Inheritance**
* **Public Inheritance**
* **Protected inheritance**

Here are the member access rules with respect to each of these:

***First and most important rule Private members of a class are never accessible from anywhere except the members of the same class.***

## Public Inheritance:

All Public members of the Base Class become Public Members of the derived class &  
All Protected members of the Base Class become Protected Members of the Derived Class.

i.e. No change in the Access of the members. The access rules we discussed before are further then applied to these members.

Code Example:

Class Base

{

public:

int a;

protected:

int b;

private:

int c;

};

class Derived:public Base

{

void doSomething()

{

a = 10; //Allowed

b = 20; //Allowed

c = 30; //Not Allowed, Compiler Error

}

};

int main()

{

Derived obj;

obj.a = 10; //Allowed

obj.b = 20; //Not Allowed, Compiler Error

obj.c = 30; //Not Allowed, Compiler Error

}

## Private Inheritance:

All Public members of the Base Class become Private Members of the Derived class &  
All Protected members of the Base Class become Private Members of the Derived Class.

An code Example:

Class Base

{

public:

int a;

protected:

int b;

private:

int c;

};

class Derived:private Base //Not mentioning private is OK because for classes it defaults to private

{

void doSomething()

{

a = 10; //Allowed

b = 20; //Allowed

c = 30; //Not Allowed, Compiler Error

}

};

class Derived2:public Derived

{

void doSomethingMore()

{

a = 10; //Not Allowed, Compiler Error, a is private member of Derived now

b = 20; //Not Allowed, Compiler Error, b is private member of Derived now

c = 30; //Not Allowed, Compiler Error

}

};

int main()

{

Derived obj;

obj.a = 10; //Not Allowed, Compiler Error

obj.b = 20; //Not Allowed, Compiler Error

obj.c = 30; //Not Allowed, Compiler Error

}

## Protected Inheritance:

All Public members of the Base Class become Protected Members of the derived class &  
All Protected members of the Base Class become Protected Members of the Derived Class.

A Code Example:

Class Base

{

public:

int a;

protected:

int b;

private:

int c;

};

class Derived:protected Base

{

void doSomething()

{

a = 10; //Allowed

b = 20; //Allowed

c = 30; //Not Allowed, Compiler Error

}

};

class Derived2:public Derived

{

void doSomethingMore()

{

a = 10; //Allowed, a is protected member inside Derived & Derived2 is public derivation from Derived, a is now protected member of Derived2

b = 20; //Allowed, b is protected member inside Derived & Derived2 is public derivation from Derived, b is now protected member of Derived2

c = 30; //Not Allowed, Compiler Error

}

};

int main()

{

Derived obj;

obj.a = 10; //Not Allowed, Compiler Error

obj.b = 20; //Not Allowed, Compiler Error

obj.c = 30; //Not Allowed, Compiler Error

}

Remember the same access rules apply to the classes and members down the inheritance hierarchy.

## Important points to note:

**- Access Specification is per-Class not per-Object**

Note that the access specification C++ work on per-Class basis and not per-object basis.  
A good example of this is that in a copy constructor or Copy Assignment operator function, all the members of the object being passed can be accessed.

**- A Derived class can only access members of its own Base class**

Consider the [**following code example**](http://ideone.com/5iAeM):

class Myclass

{

protected:

int x;

};

class derived : public Myclass

{

public:

void f( Myclass& obj )

{

obj.x = 5;

}

};

int main()

{

return 0;

}

It gives an compilation error:

prog.cpp:4: error: ‘int Myclass::x’ is protected

Because the derived class can only access members of its **own Base Class**. Note that the object obj being passed here is no way related to the derived class function in which it is being accessed, it is an altogether different object and hence derived member function cannot access its members.

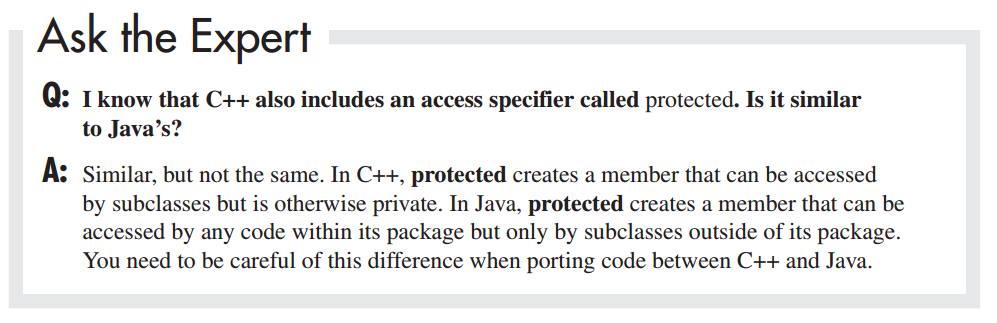
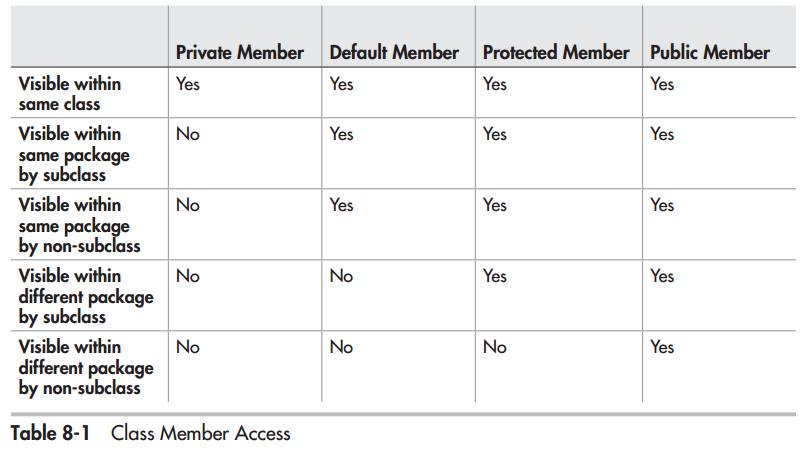
## What is a friend? How does friend affect access specification rules?

You can declare a function or class as friend of another class. When you do so the access specification rules do not apply to the friended class/function. The class or function can access all the members of that particular class.

***So do friends break Encapsulation?***

No they don't, On the contrary they enhance Encapsulation!

friendship is used to indicate a **intentional strong coupling** between two entities.  
If there exists a special relationship between two entities such that one needs access to others private or protected members but You do not want **everyone** to have access by using the public access specifier then you should use friendship.



[file:///D:/c++%20-%20What%20are%20access%20specifiers/c++%20-%20What%20are%20access%20specifiers\_%20Should%20I%20inherit%20with%20private,%20protected%20or%20public\_%20-%20Stack%20Overflow.html](file:///D:\c++%20-%20What%20are%20access%20specifiers\c++%20-%20What%20are%20access%20specifiers_%20Should%20I%20inherit%20with%20private,%20protected%20or%20public_%20-%20Stack%20Overflow.html)