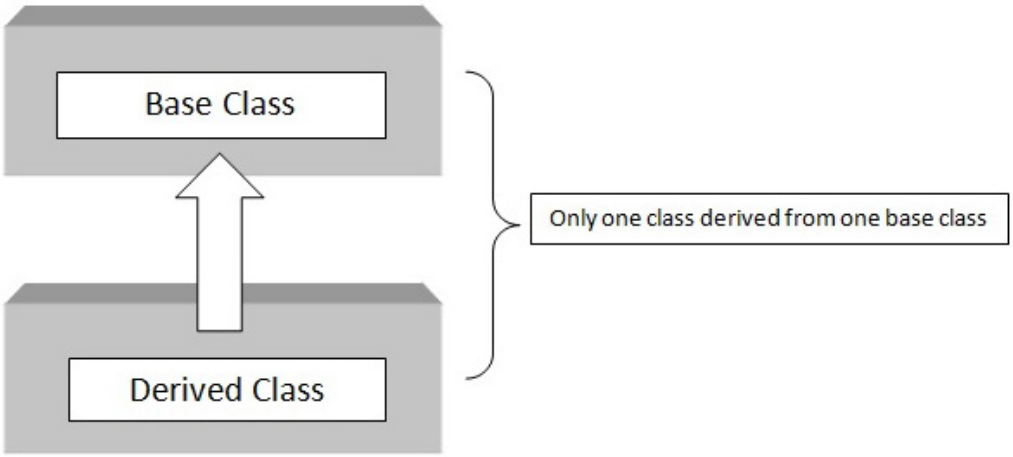
Data Abstraction: Data abstraction allows a program to ignore the details of how a data type is represented.
refers to the act of representing essential features without including the background details or explanations
C++ classes use the technique of abstraction and are defined as a list of abstract attributes such as width, cost, size etc and functions to operate on these attributes.

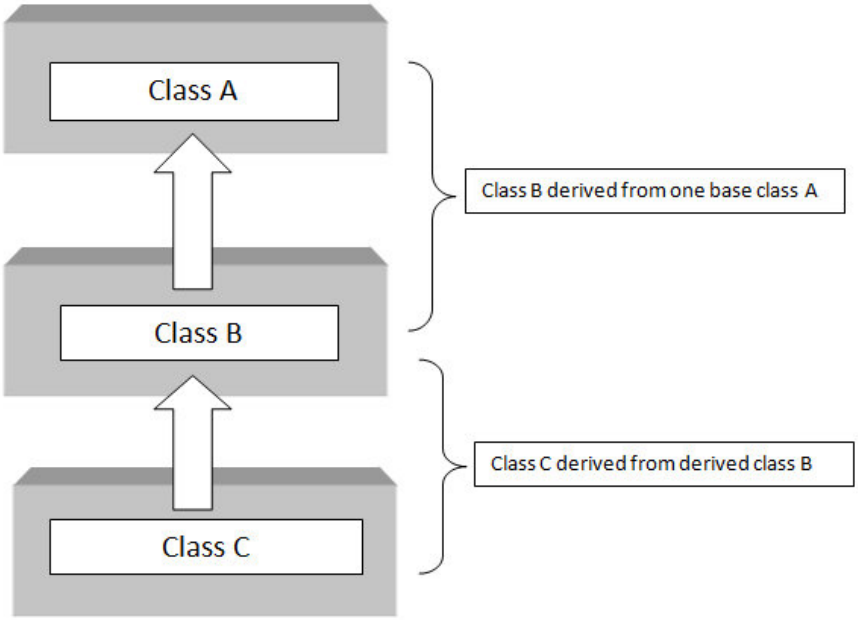
```
#include <iostream>
using namespace std;

class sample
{
     public:  // can be made private:
    int v1, v2;

     public:
    void val()
    {
        cout << "Enter Two values : "; cin >> v1 >> v2;
    }
     // private: this hides display() hidden
    void display()
    {
        cout << v1 << " " << v2;
        cout << endl;
    }
};

int main()
{
    sample S;
    S.val();
     S.v1=10'
    S.display();
}
```

```
Enter Two values : 20

50

20 50
```

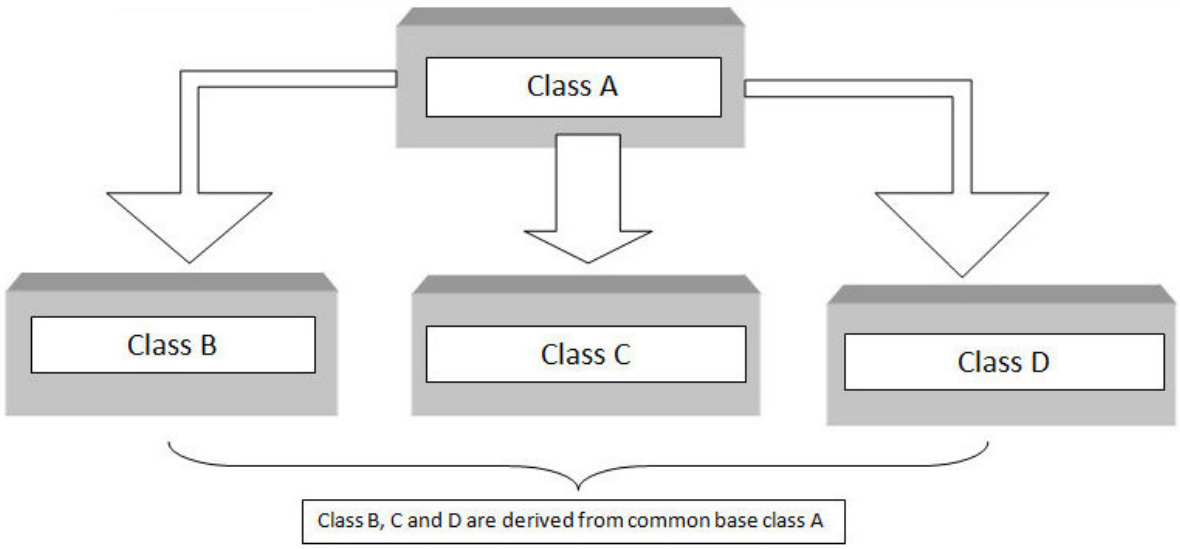
Inheritance: Single



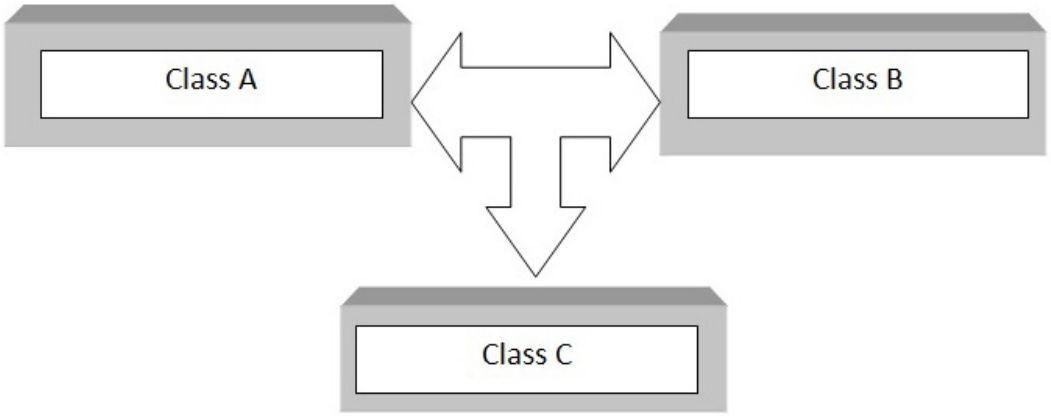
Multilevel Inheritance:



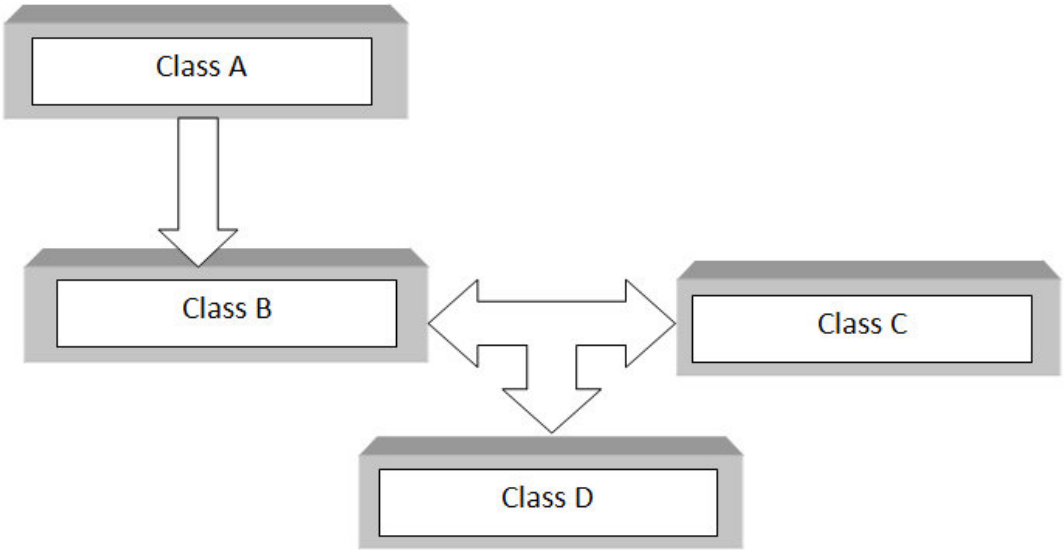
Hierarchical Inheritance



Multiple Inheritance



HYBRID INHERITANCE



CODE:

Single

```
#include<iostream>
using namespace std;
class A
{
        public:
        void disp1()
        {
        cout<<"class A"<<endl;
        }

};

class B: public A
{
```

```
        public:
        void disp2()
        {
        cout<<"class B"<<endl;
        }
};

int main()
{
B ob1;
ob1.disp1();
//ob1.disp2();
return 0;
}
```

Multi level

```
#include<iostream>
using namespace std;
class A
{
        public:
        void disp1()
        {
        cout<<"class A"<<endl;
        }

};

class B: public A
{
        public:
        void disp2()
        {
        cout<<"class B"<<endl;
```

```
        }
};

class C: public B
{
        public:
        void disp3()
        {
        cout<<"class C"<<endl;
        }
};


int main()
{
B ob1;
C ob2;
ob1.disp1();
ob2.disp2();
return 0;
}
```

Hierarchical

```
#include<iostream>
using namespace std;
class A
{
        public:
        void disp1()
        {
        cout<<"class A"<<endl;
        }

};

class B: public A
{
        public:
        void disp2()
        {
        cout<<"class B"<<endl;
        }
};

class C: public A
{
```

```
        public:
        void disp3()
        {
        cout<<"class C"<<endl;
        }
};
class D: public A
{
        public:
        void disp4()
        {
        cout<<"class D"<<endl;
        }
};

int main()
{
B ob1;
C ob2;
D ob3;
ob1.disp1();
ob2.disp1();
ob3.disp1();
return 0;
}
```

Multiple

```
#include<iostream>
using namespace std;
class A
{
        public:
        void disp1()
        {
        cout<<"class A"<<endl;
        }

};

class B
{
        public:
        void disp2()
        {
        cout<<"class B"<<endl;
        }
};

class C
{
```

```
        public:
        void disp3()
        {
        cout<<"class C"<<endl;
        }
};
class D: public A, public B, public C
{
        public:
        void disp4()
        {
        cout<<"class D"<<endl;
        }
};

int main()
{
D ob1;
ob1.disp1();
ob1.disp2();
ob1.disp3();
return 0;
}
```

Hybrid

```
#include<iostream>
using namespace std;
class A
{
        public:
        void disp1()
        {
        cout<<"class A"<<endl;
        }

};

class B:public A
{
        public:
        void disp2()
        {
        cout<<"class B"<<endl;
        }
};

class C
{
        public:
```

```
        void disp3()
        {
        cout<<"class C"<<endl;
        }
};
class D:public B, public C
{
        public:
        void disp4()
        {
        cout<<"class D"<<endl;
        }
};

int main()
{
D ob1;
ob1.disp2();
ob1.disp3();

B ob2;
ob2.disp1();
return 0;
}
```