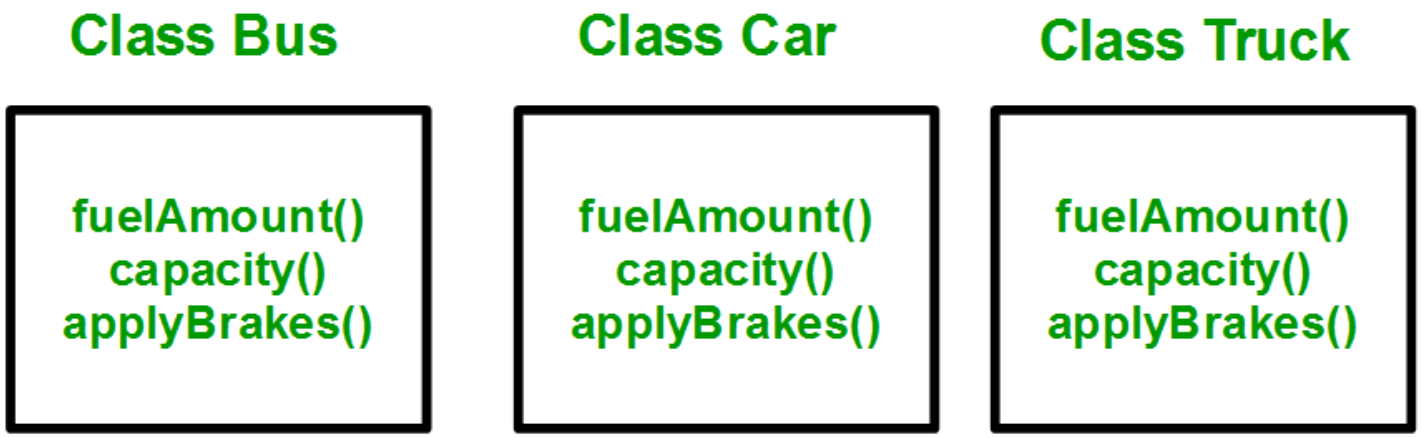
Inheritance in C++ (Part-01)

Inheritance

- The capability of a class to derive properties and characteristics from another class is called **Inheritance**.
- Inheritance is one of the most important feature of Object Oriented Programming.
- **Sub Class:** The class that inherits properties from another class is called Sub class or Derived Class.
- **Super Class:** The class whose properties are inherited by sub class is called Base Class or Super class.

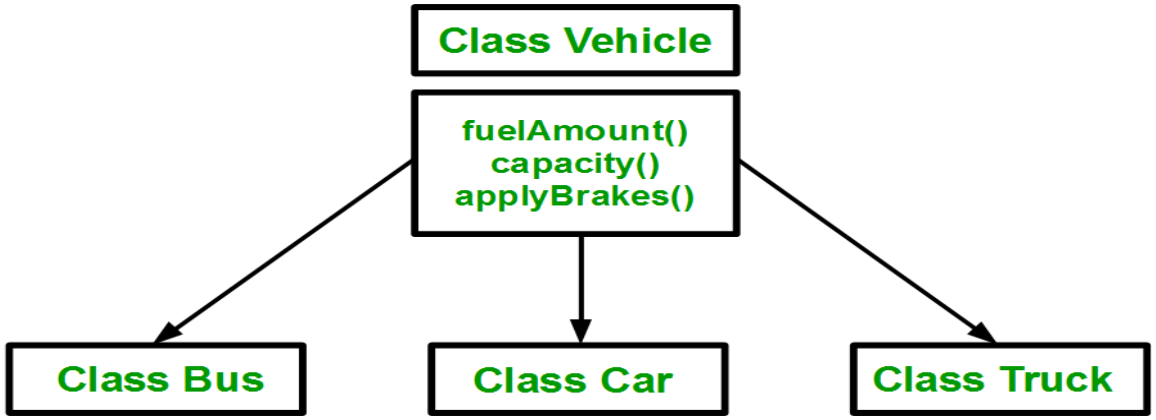
Why and When to Use Inheritance

- Consider a group of vehicles. You need to create classes for Bus, Car and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be same for all of the three classes.
- If we create these classes avoiding inheritance then we have to write all of these functions in each of the three classes as shown in below figure:



Why and When to Use Inheritance

- You can clearly see that above process results in duplication of same code 3 times. This increases the chances of error and data redundancy. To avoid this type of situation, inheritance is used. If we create a class Vehicle and write these three functions in it and inherit the rest of the classes from the vehicle class, then we can simply avoid the duplication of data and increase re-usability. Look at the below diagram in which the three classes are inherited from vehicle class:



Why and When to Use Inheritance?

- Using inheritance, we have to write the functions only one time instead of three times as we have inherited rest of the three classes from base class(Vehicle).
- ***Inheritance is a term for reusing code by a mechanism of passing down information and behavior from a parent class to a child or subclass.***

Implementing Inheritance in C++

- **Implementing inheritance in C++**: For creating a sub-class which is inherited from the base class we have to follow the below syntax.

```
class SubClass_name :: access_mode BaseClass_name
{
    //Body of the SubClass
};
```

- Here, **Subclass_name** is the name of the sub class, **access_mode** is the mode in which you want to inherit this sub class for example: public, private etc. and **BaseClass_name** is the name of the base class from which you want to inherit the sub class.

Modes of Inheritance

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

Modes of Inheritance

- The below table summarizes the above three modes and shows the access specifier of the members of base class in the sub class when derived in public, protected and private modes:

Base Class member Access Specifier	Type of Inheritance		
	Public	**Protected**	**Private**
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not Accessible (Hidden)	Not Accessible (Hidden)	Not Accessible (Hidden)

Properties of Base Class: Public, Mode: Public

```
#include<iostream>
using namespace std;
class CSE{
public:
    string name = "A"; int id = 12345;
    void PRINT(){
        cout<<"I am from CSE Class"<<endl;
    }
};
class PRIME: public CSE{
public:
    void display(){
        cout<<"Name: "<<name<<"ID: "<<id<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();
    obj.PRINT();
    return 0;
}
```

Properties of Base Class: Public, Mode: Protected

```
#include<iostream>
using namespace std;
class CSE{
public:
    string name = "A"; int id = 12345;
    void PRINT(){
        cout<<"I am from CSE Class"<<endl;
    }
};
class PRIME: protected CSE{
public:
    void display(){
        cout<<"Name: "<<name<<"ID: "<<id<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();
    obj.PRINT();//PRINT() not accessible; Member function must be public
    return 0;
}
```

Properties of Base Class: Public, Mode: Private

```
#include<iostream>
using namespace std;
class CSE{
public:
    string name = "A"; int id = 12345;
    void PRINT(){
        cout<<"I am from CSE Class"<<endl;
    }
};
class PRIME: private CSE{
public:
    void display(){
        cout<<"Name: "<<name<<"ID: "<<id<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();
    obj.PRINT();//PRINT() not accessible; Member function must be public
    return 0;
}
```

Properties of Base Class: Protected, Mode: Public

```
#include<iostream>
using namespace std;
class CSE{
protected:
    string name = "A"; int id = 12345;
};
class PRIME: public CSE{
public:
    void display(){
        cout<<"Name: "<<name<<", ID: "<<id<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();
    return 0;
}
```

Properties of Base Class: Protected, Mode: Protected

```
#include<iostream>
using namespace std;
class CSE{
protected:
    string name = "A"; int id = 12345;
};
class PRIME: protected CSE{
public:
    void display(){
        cout<<"Name: "<<name<<", ID: "<<id<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();
    return 0;
}
```

Properties of Base Class: Protected, Mode: Private

```
#include<iostream>
using namespace std;
class CSE{
protected:
    string name = "A"; int id = 12345;
};
class PRIME: private CSE{
public:
    void display(){
        cout<<"Name: "<<name<<", ID: "<<id<<endl;
    }
};
int main()
{
    PRIME obj;
    obj.display();
    return 0;
}
```

Properties of Base Class: private, Mode: Public

```
#include<iostream>
using namespace std;
class CSE{
private:
    string name = "A"; int id = 12345;
};
class PRIME: public CSE{
public:
    void display(){
        cout<<"Name: "<<name<<", ID: "<<id<<endl;//Not accessible
    }
};
int main()
{
    PRIME obj;
    obj.display();
    return 0;
}
```

Properties of Base Class: private, Mode: Protected

```
#include<iostream>
using namespace std;
class CSE{
private:
    string name = "A"; int id = 12345;
};
class PRIME: protected CSE{
public:
    void display(){
        cout<<"Name: "<<name<<", ID: "<<id<<endl;//Not Accessible
    }
};
int main()
{
    PRIME obj;
    obj.display();
    return 0;
}
```

Properties of Base Class: private, Mode: Private

```
#include<iostream>
using namespace std;
class CSE{
private:
    string name = "A"; int id = 12345;
};
class PRIME: private CSE{
public:
    void display(){
        cout<<"Name: "<<name<<", ID: "<<id<<endl;//Not Accessible
    }
};
int main()
{
    PRIME obj;
    obj.display();
    return 0;
}
```

Again, Modes of Inheritance

- The below table summarizes the above three modes and shows the access specifier of the members of base class in the sub class when derived in public, protected and private modes:

Base Class member Access Specifier	Type of Inheritance		
	Public	**Protected**	**Private**
Public	Public	Protected	Private
Protected	Protected	Protected	Private
Private	Not Accessible (Hidden)	Not Accessible (Hidden)	Not Accessible (Hidden)

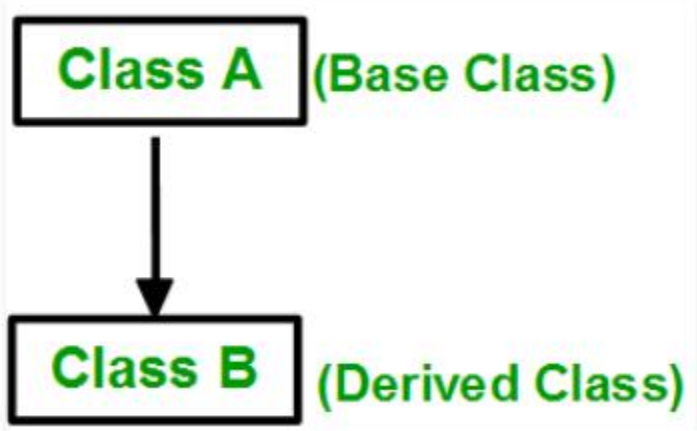
Types of Inheritance

Types of Inheritance in C++

- **Single Inheritance**
- **Multiple Inheritance**
- **Multilevel Inheritance**
- **Hierarchical Inheritance**
- **Hybrid Inheritance (Virtual Inheritance)**

Single Inheritance

- In single inheritance, a class is allowed to inherit from only one class. i.e. one sub class is inherited by one base class only.



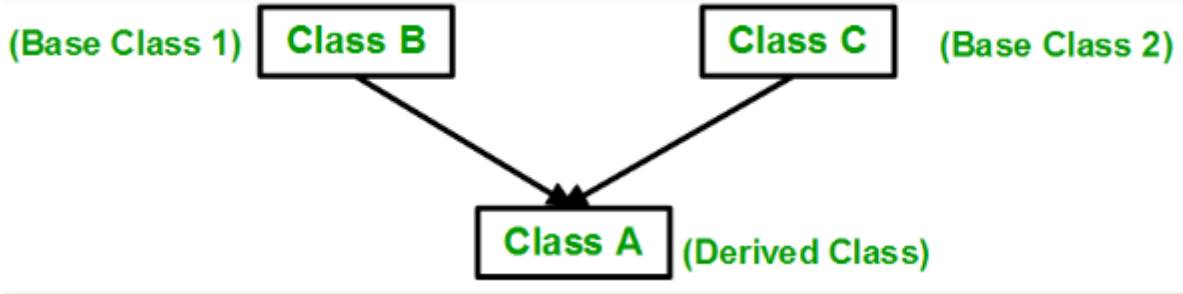

```
Syntax:
class subclass_name : access_mode base_class
{
   ///body of subclass
};
```

Single Inheritance

```
1  #include <iostream>
2  using namespace std;
3  class Prime/// base class
4  {
5      public:
6          Prime()
7          {
8              cout << "This is a Prime University" << endl;
9          }
10 };
11 /// sub class derived from two base classes
12 class Dept: public Prime
13 {
14
15 };
16 int main()
17 {
18     /// creating object of sub class will
19     /// invoke the constructor of base classes
20     Dept obj;
21     return 0;
22 }
```

Multiple Inheritance

- Multiple Inheritance is a feature of C++ where a class can inherit from more than one classes. i.e one **sub class** is inherited from more than one **base classes**.




```
Syntax:
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
  ///body of subclass
};
```

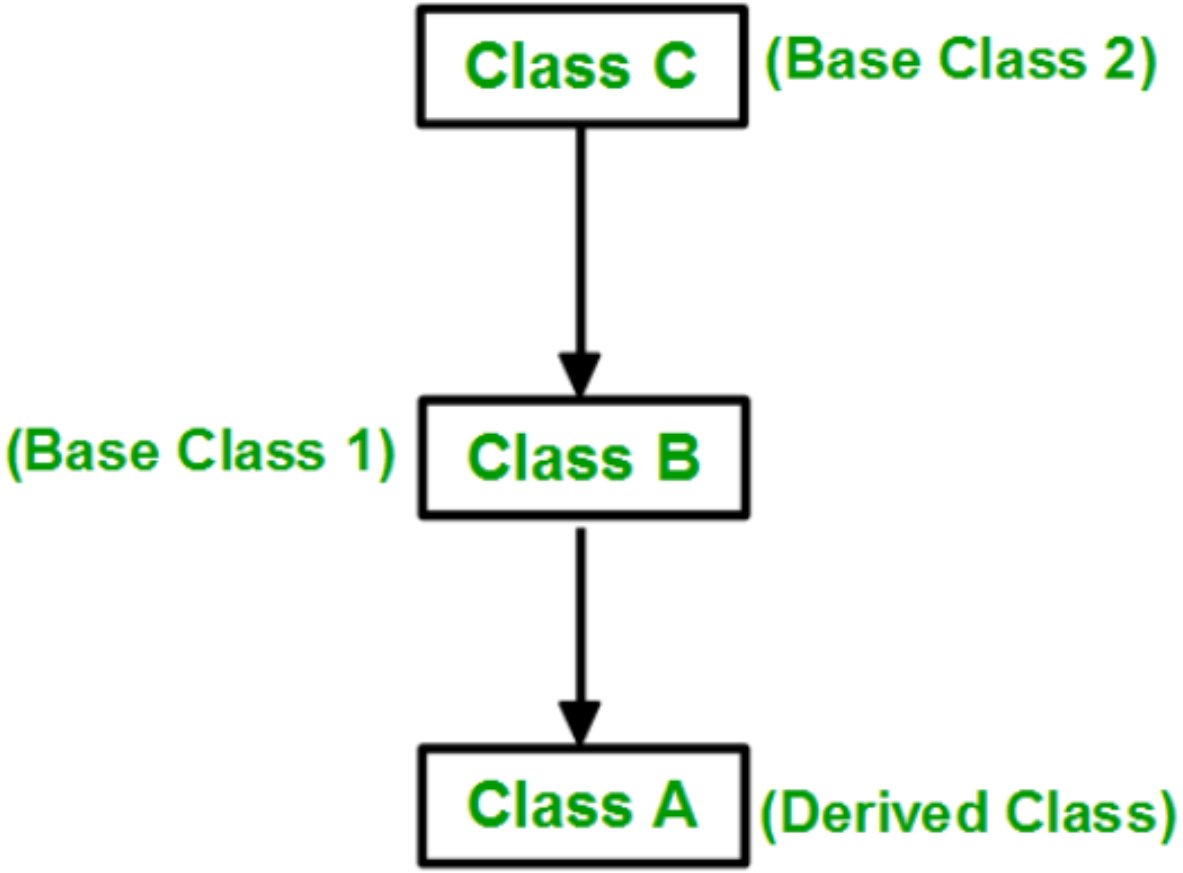
Multiple Inheritance

```
#include <iostream>
using namespace std;
class Vehicle { /// first base class
  public:
    Vehicle()
    {
      cout << "This is a Vehicle" << endl;
    }
};
class FourWheeler { /// second base class
  public:
    FourWheeler()
    {
      cout << "This is a 4 wheeler Vehicle" << endl;
    }
};
/// sub class derived from two base classes
class Car: public Vehicle, public FourWheeler {

};
int main()
{
    /// creating object of sub class will invoke the constructor of base classes
    Car obj;
    return 0;
}
```

Multilevel Inheritance

- In this type of inheritance, a derived class is created from another derived class.

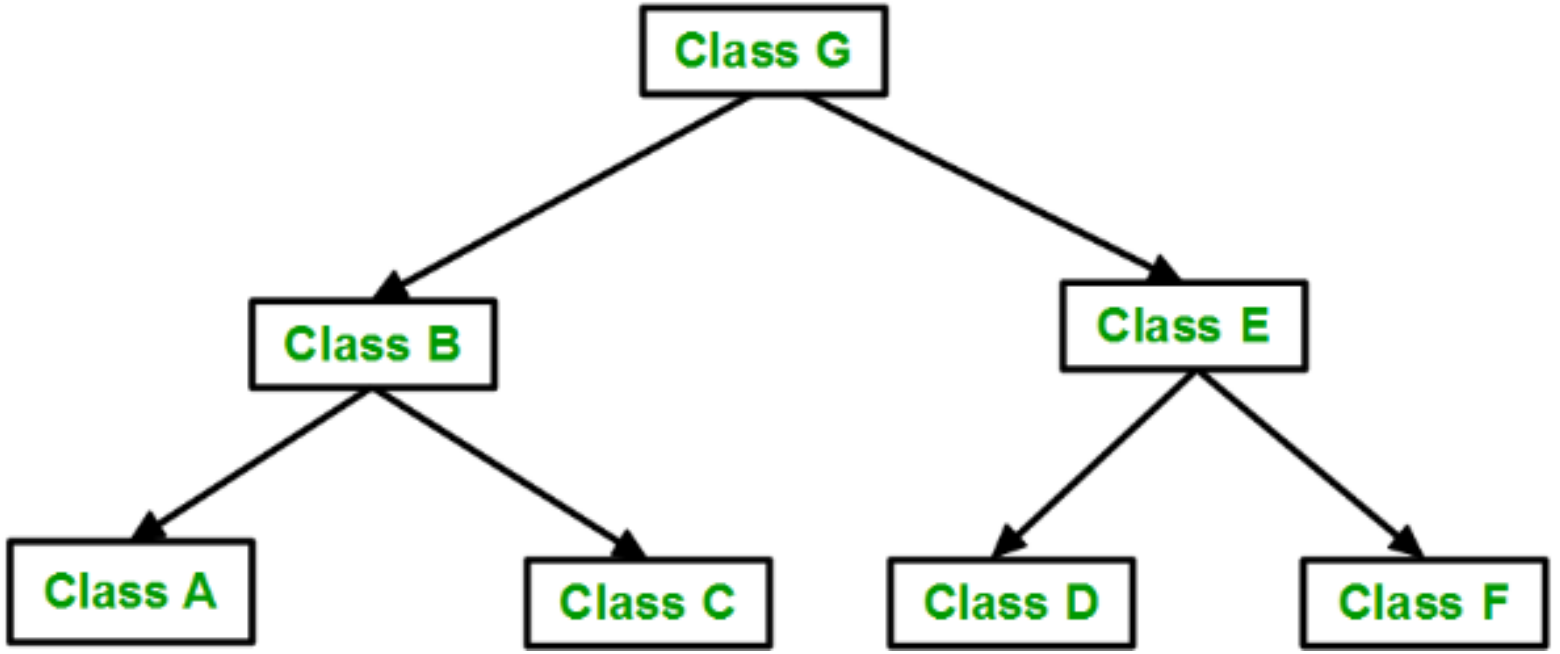


Multilevel Inheritance

```
3   class Vehicle/// base class
4   {
5       public:
6           Vehicle()
7           {
8               cout << "This is from Vehicle class" << endl;
9           }
10  };
11  class fourWheeler: public Vehicle
12  {
13      public:
14          fourWheeler()
15          {
16              cout<<"This is from fourWheeler class"<<endl;
17          }
18  };
19  /// sub class derived from two base classes
20  class Car: public fourWheeler
21  {
22      public:
23          Car()
24          {
25              cout<<"This is from car class"<<endl;
26          }
27  };
28  int main()
29  {
30      ///creating object of sub class will invoke the constructor of base classes
31      Car obj;
32      return 0;
33  }
```

Hierarchical Inheritance

- In this type of inheritance, more than one sub class is inherited from a single base class. i.e. more than one derived class is created from a single base class.

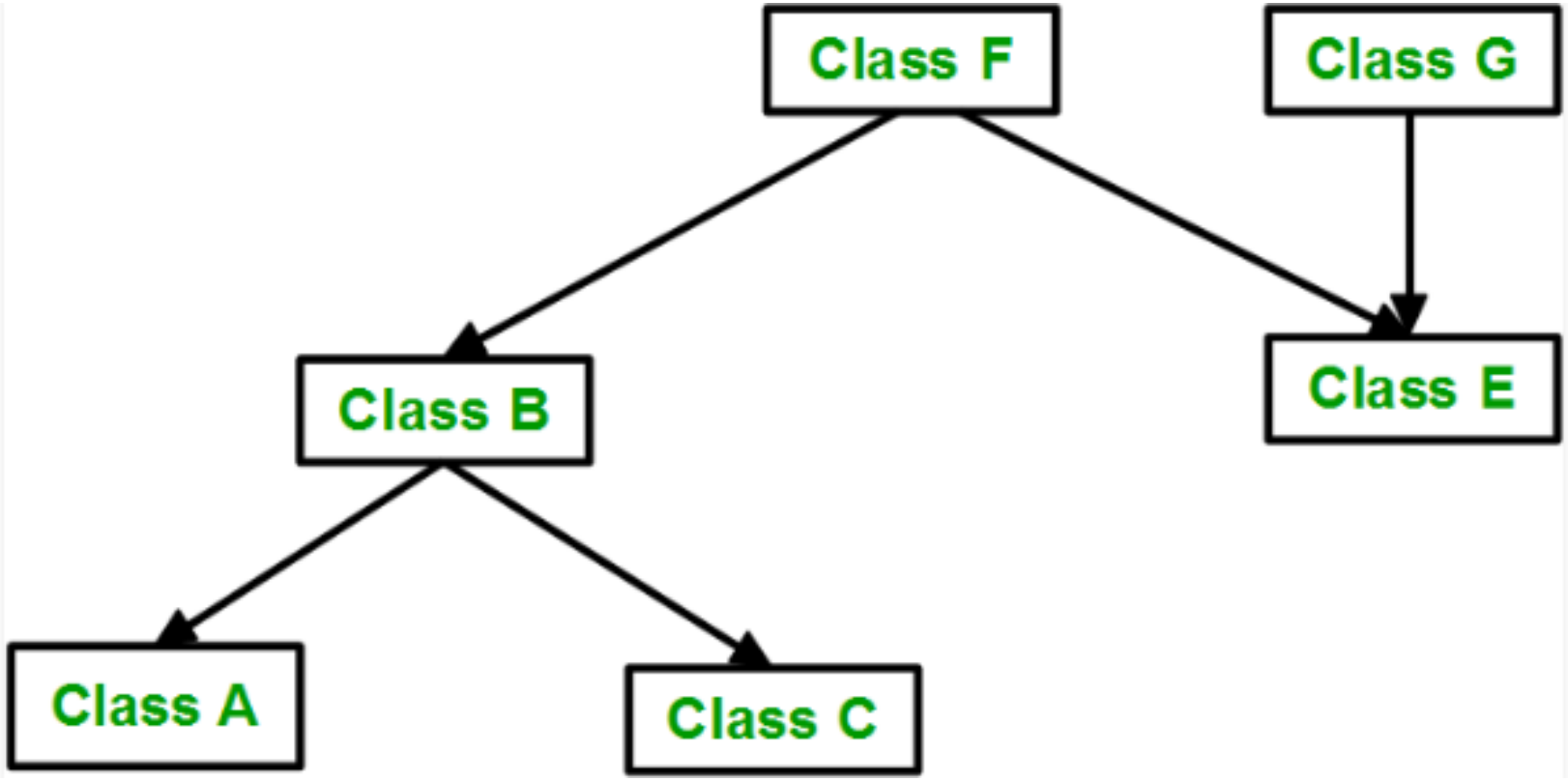


Hierarchical Inheritance

```
1  #include <iostream>
2  using namespace std;
3  class Vehicle/// base class
4  {
5  public:
6      Vehicle()
7      {
8          cout << "This is a Vehicle" << endl;
9      }
10 };
11 class Car: public Vehicle /// first sub class
12 {
13
14 };
15 class Bus: public Vehicle/// second sub class
16 {
17
18 };
19 int main()
20 {
21     /// creating object of sub class will invoke the constructor of base class
22     Car obj1;
23     Bus obj2;
24     return 0;
25 }
```

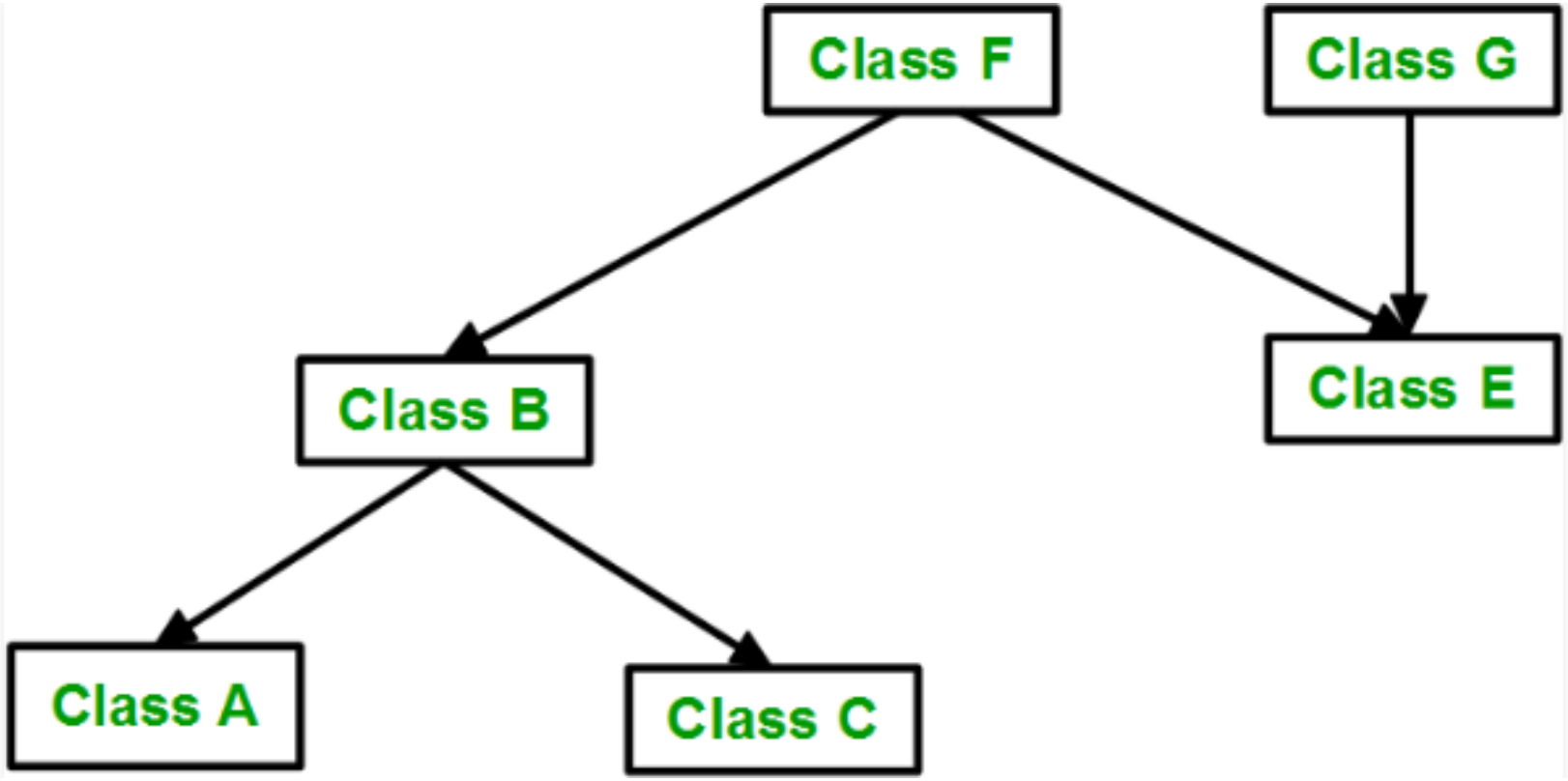
Hybrid(Virtual) Inheritance

- Hybrid Inheritance is implemented by combining more than one type of inheritance.



Hybrid(Virtual) Inheritance

- Hybrid Inheritance is implemented by combining more than one type of inheritance.



Hybrid(Virtual) Inheritance

```
3   class Vehicle  /// base class
4   {
5         public:
6           Vehicle()
7           {
8             cout << "This is a Vehicle" << endl;
9           }
10  };
11  class Fare ///base class
12  {
13      public:
14          Fare()
15          {
16              cout<<"Fare of Vehicle\n";
17          }
18  };
19  /// first sub class
20  class Car: public Vehicle
21  {
22
23  };
24  /// second sub class
25  class Bus: public Vehicle, public Fare
26  {
27
28  };
29  int main()
30  {
31      /// creating object of sub class will invoke the constructor of base class
32      Bus obj2;
33      return 0;
34  }
```

Thank You