OBJECT PROGRAMMING

- LECTURE 8 -

(1ST YEAR OF STUDY)

Skopje, 2023/24

Contents

8.1. Introduction

- The capability of *a class* to derive properties and characteristics from *another class* is called **Inheritance**. Inheritance is one of the most important features of OOP.
- Inheritance is a feature or a process in which, new classes are created from the existing classes.
- The new class created is called "*derived class*" or "*child class*", and the existing class is known as the "*base class*" or "*parent class*". The derived class now is said to be inherited from the base class.

8.1. Introduction

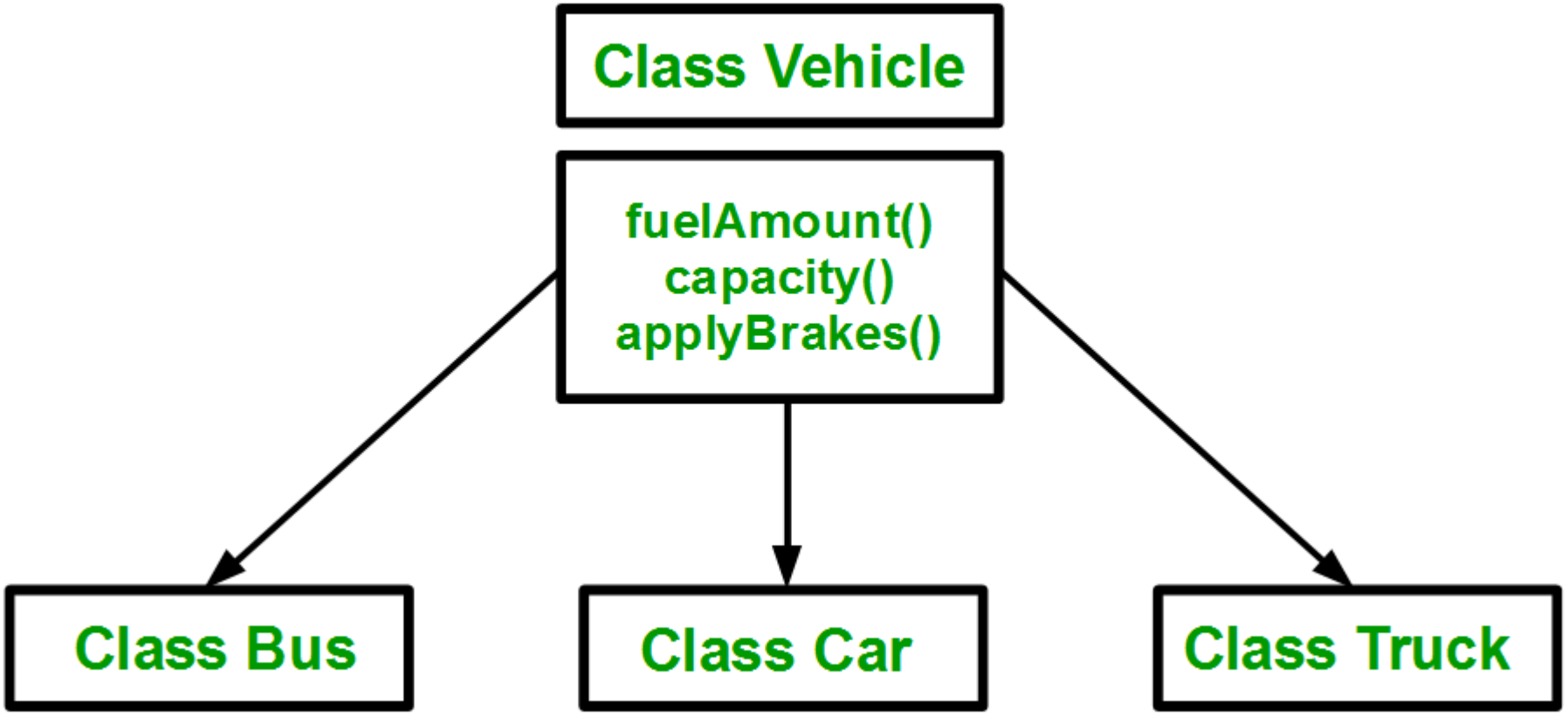
- When we say derived class inherits the base class, it means, the derived class inherits all the properties of the base class, without changing the properties of base class and may add new features to its own.
- These new features in the derived class *will not affect* the base class. The derived class is the specialized class for the base class.
  - **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 a sub class is called ***Super Class*** or ***Base Class***.

8.1. Introduction

- Consider a ***group of vehicles***. We need to create classes for Bus, Car, and Truck. The methods fuelAmount(), capacity(), applyBrakes() will be the same for all 3 classes.
- If we create these classes avoiding Inheritance, then we have to write all of these functions in each of the three classes, resulting in duplication of the same code 3 times.
- 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:

8.1. Introduction

- Using Inheritance, we have to write the functions only *one* time instead of *three* times, as we have inherited the rest of the three classes from the base class (Vehicle).



8.1. Introduction

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

```
class  <derived_class_name> : <access-specifier> <base_class_name>
{
        //body
}
```

  - *class* - keyword to create a new class;
  - *derived_class_name* - name of the new class, which will inherit the base class;
  - *access-specifier* - either of *private*, *public*, or *protected*. If neither is specified, PRIVATE is taken as default;
  - *base_class_name* - name of the base class.

8.1. Introduction

- **Note:** A derived class doesn't inherit ***access*** to private data members. However, it does inherit a full parent object, which contains any private members which that class declares.

Example:

```
1. class ABC : private XYZ          //private derivation
        {          }
2. class ABC : public XYZ           //public derivation
        {          }
3. class ABC : protected XYZ        //protected derivation
        {          }
4. class ABC: XYZ                   //private derivation by default
{       }
```

8.1. Introduction

- When a base class is *privately* inherited by the derived class, public members of the base class becomes the private members of the derived class.
- Therefore, the public members of the base class can only be accessed by the member functions of the derived class. They are inaccessible to the objects of the derived class.

8.1. Introduction

- On the other hand, when the base class is *publicly* inherited by the derived class, public members of the base class also become the public members of the derived class.
- Therefore, the public members of the base class are accessible by the objects of the derived class as well as by the member functions of the derived class.

8.1. Introduction


```
// Example: define member function without argument within
// the class

#include <iostream>
using namespace std;

class Person {
    int id;
    char name[100];

public:
    void set_p()
    {
        cout << "Enter the Id:";
        cin >> id;
        cout << "Enter the Name:";
        cin >> name;
    }

    void display_p()
    {
        cout << endl <<"Id: "<< id << "\nName: " << name <<endl;
    }
};
```

```
class Student : private Person {
    char course[50];
    int fee;

public:
    void set_s()
    {
        set_p();
        cout << "Enter the Course Name:";
        cin >> course;
        cout << "Enter the Course Fee:";
        cin >> fee;
    }

    void display_s()
    {
        display_p();
        cout <<"Course: "<< course << "\nFee: " << fee << endl;
    }
};

int main()
{
    Student s;
    s.set_s();
    s.display_s();
    return 0;
}
```

Output:

```
Enter the Id: 101
Enter the Name: Dev
Enter the Course Name: GCS
Enter the Course Fee:70000

Id: 101
Name: Dev
Course: GCS
Fee: 70000
```

8.1. Introduction


```
// Example: define member function without argument outside the class

#include<iostream>
using namespace std;

class Person
{
    int id;
    char name[100];

    public:
        void set_p();
        void display_p();
};

void Person::set_p()
{
    cout<<"Enter the Id:";
    cin>>id;
    cout<<"Enter the Name:";
    cin>>name;
}

void Person::display_p()
{
    cout<<endl<<"id: "<< id<<"\nName: "<<name;
}

class Student: private Person
{
    char course[50];
    int fee;

    public:
        void set_s();
        void display_s();
};

void Student::set_s()
{
    set_p();
    cout<<"Enter the Course Name:";
    cin>>course;
    cout<<"Enter the Course Fee:";
    cin>>fee;
}

void Student::display_s()
{
    display_p();
    cout<<"\nCourse: "<<course<<"\nFee: "<<fee<<endl;
}

int main()
{
    Student s;
    s.set_s();
    s.display_s();
    return 0;
}
```

Output:

```
Enter the Id: 101
Enter the Name: Dev
Enter the Course Name: GCS
Enter the Course Fee: 70000
Id: 101
Name: Dev
Course: GCS
Fee: 70000
```

8.1. Introduction


```
// Example: define member function with argument outside the class

#include<iostream>
#include<string.h>
using namespace std;

class Person
{
    int id;
    char name[100];

    public:
        void set_p(int,char[]);
        void display_p();
};

void Person::set_p(int id,char n[])
{
    this->id=id;
    strcpy(this->name,n);
}

void Person::display_p()
{
    cout<<endl<<id<<"\t"<<name;
}

class Student: private Person
{
    char course[50];
    int fee;
    public:
    void set_s(int,char[],char[],int);
    void display_s();
};

void Student::set_s(int id,char n[],char c[],int f)
{
    set_p(id,n);
    strcpy(course,c);
    fee=f;
}


void Student::display_s()
{
    display_p();
    cout<<"t"<<course<<"\t"<<fee;
}

main()
{
    Student s;
    s.set_s(1001,"Ram","B.Tech",2000);
    s.display_s();
    return 0;
}
```

Output:

1001 RamtB.Tech 2000

8.1. Introduction


```cpp
// C++ program to demonstrate implementation
// of Inheritance

#include <bits/stdc++.h>
using namespace std;

// Base class
class Parent {
public:
    int id_p;
};

// Sub class inheriting from Base Class(Parent)
class Child : public Parent {
public:
    int id_c;
};
```

```cpp
// main function
int main()
{
    Child obj1;

    // An object of class child has all data members
    // and member functions of class parent
    obj1.id_c = 7;
    obj1.id_p = 91;
    cout << "Child id is: " << obj1.id_c << '\n';
    cout << "Parent id is: " << obj1.id_p << '\n';

    return 0;
}
```

Output

```
Child id is: 7
Parent id is: 91
```

8.2. Modes and Types 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.

8.2. Modes and Types of Inheritance

- **Note:** The ***private*** members in the base class *cannot* be directly accessed in the derived class, while ***protected*** members *can* be directly accessed.
- For example, Classes B, C, and D all contain the variables x, y, and z in the below example.
- It is just a question of access.

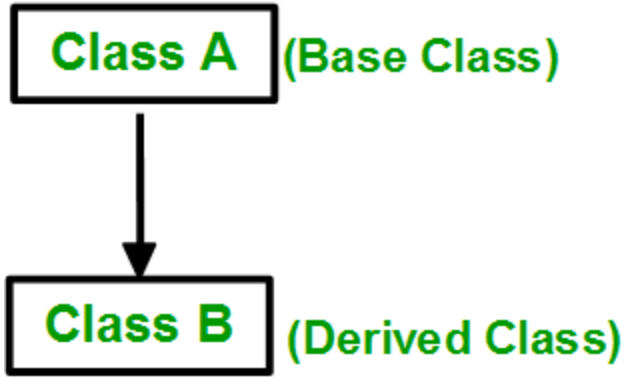
8.2. Modes and Types of Inheritance


```
// C++ Implementation to show that a derived class
// doesn't inherit access to private data members.
// However, it does inherit a full parent object.
class A {
public:
    int x;

protected:
    int y;

private:
    int z;
};

class B : public A {
    // x is public
    // y is protected
    // z is not accessible from B
};

class C : protected A {
    // x is protected
    // y is protected
    // z is not accessible from C
};

class D : private A // 'private' is default for classes
{
    // x is private
    // y is private
    // z is not accessible from D
};
```

8.2. Modes and Types of Inheritance

- Types of Inheritance

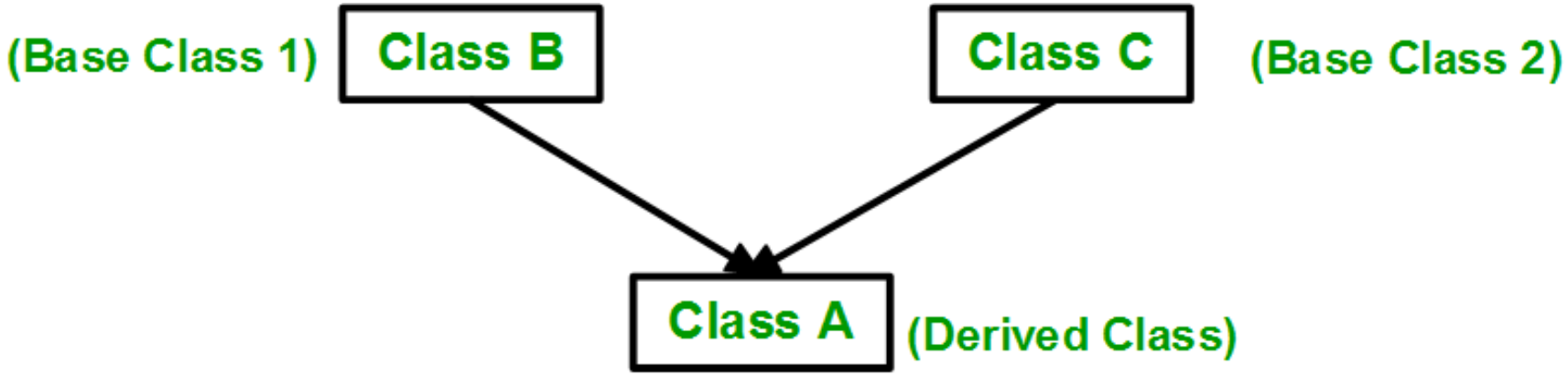
- **1) Single Inheritance:** In single inheritance, a class is allowed to inherit from only one class, i.e., one subclass is inherited by one base class only.




```
class subclass_name : access_mode base_class
{
  // body of subclass
};
OR
class A
{
... .. ...
};
class B: public A
{
... .. ...
};
```

8.2. Modes and Types of Inheritance

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



8.2. Modes and Types of Inheritance

- Here, the number of base classes will be separated by a comma (', ') and the *access mode* for every base class must be specified.

```
class subclass_name : access_mode base_class1, access_mode base_class2, ....
{
  // body of subclass
};
class B
{
... .. ...
};
class C
{
... .. ...
};
class A: public B, public C
{
... ... ...
};
```

8.2. Modes and Types of Inheritance

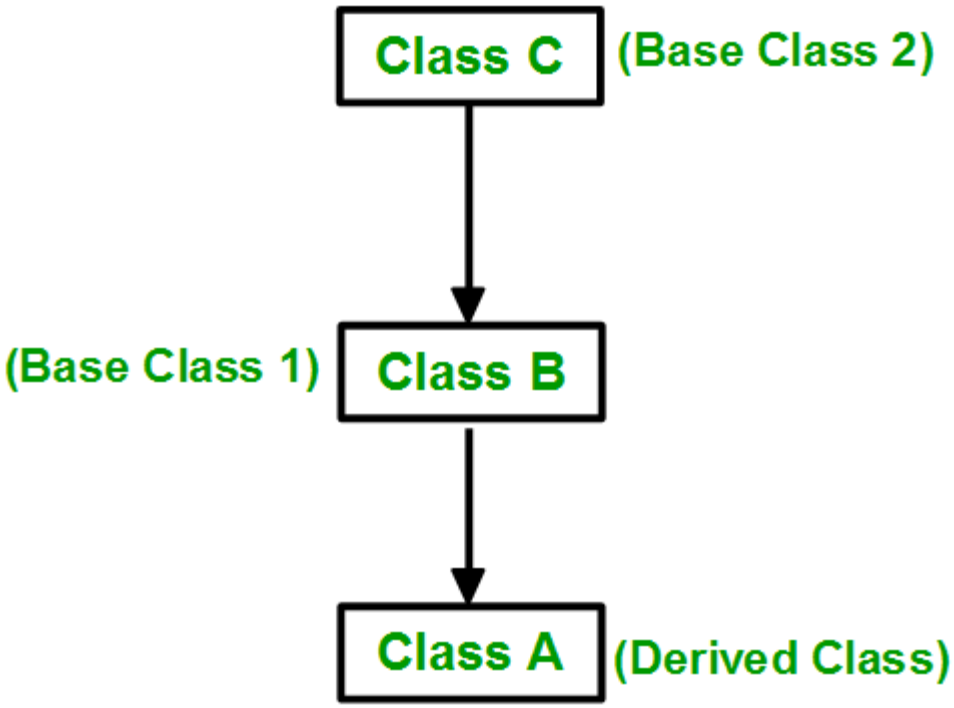

```
// C++ program to explain
// multiple inheritance
#include <iostream>
using namespace std;

// first base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// second base class
class FourWheeler {
public:
    FourWheeler()
    {
        cout << "This is a 4 wheeler Vehicle\n";
    }
};

// sub class derived from two base classes
class Car : public Vehicle, public FourWheeler {
};

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

Output

```
This is a Vehicle
This is a 4 wheeler Vehicle
```

8.2. Modes and Types of Inheritance

- **3) Multilevel Inheritance:** In this type of inheritance, a derived class is created from another derived class.




```
class C
{
... .. ...
};
class B:public C
{
... .. ...
};
class A: public B
{
... ... ...
};
```

8.2. Modes and Types of Inheritance


```
// C++ program to implement
// Multilevel Inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub_class derived from class vehicle
class fourWheeler : public Vehicle {
public:
    fourWheeler()
    {
        cout << "Objects with 4 wheels are vehicles\n";
    }
};
// sub class derived from the derived base class fourWheeler
class Car : public fourWheeler {
public:
    Car() { cout << "Car has 4 Wheels\n"; }
};
```

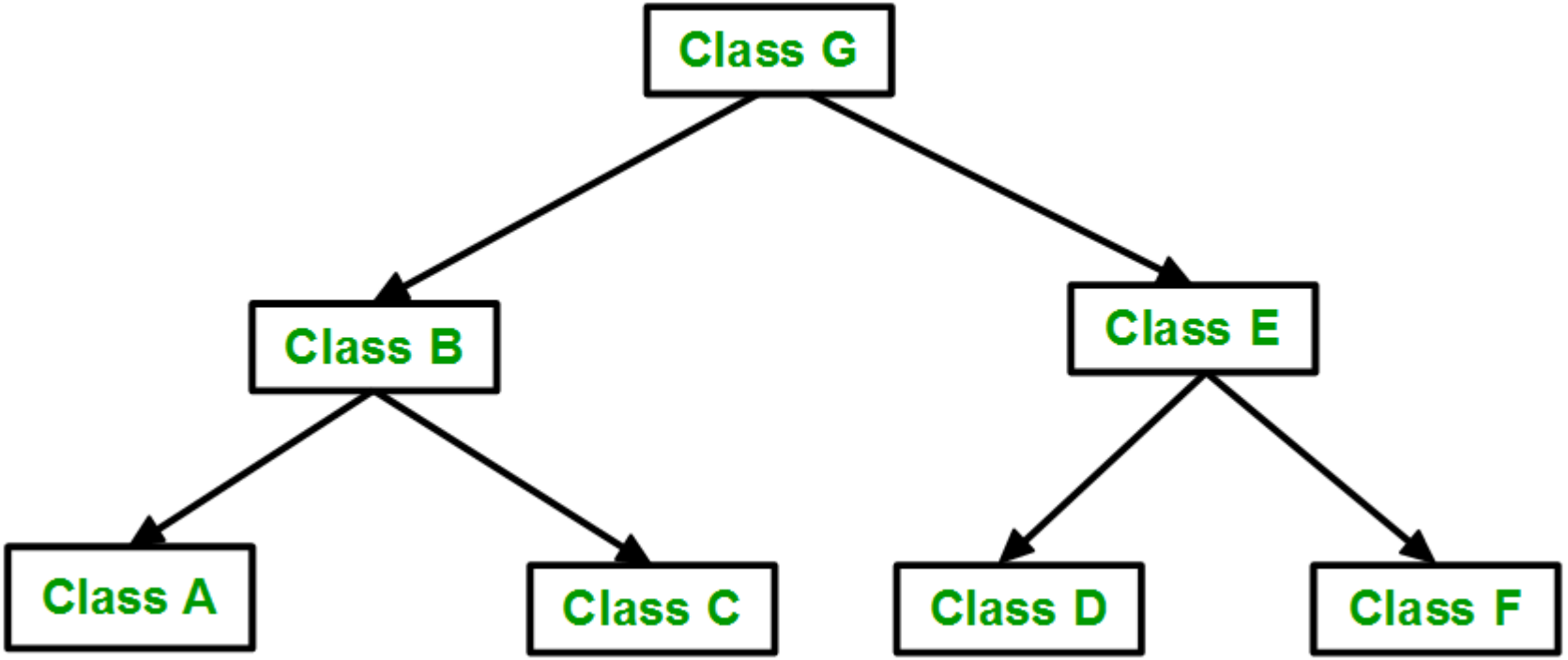
```
// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base classes.
    Car obj;
    return 0;
}
```

Output

```
This is a Vehicle
Objects with 4 wheels are vehicles
Car has 4 Wheels
```

8.2. Modes and Types of Inheritance

- **4) Hierarchical Inheritance:** In this type of inheritance, more than one subclass is inherited from a single base class, i.e., more than one derived class is created from a single base class.



8.2. Modes and Types of Inheritance


```
class A
{
    // body of the class A.
}
class B : public A
{
    // body of class B.
}
class C : public A
{
    // body of class C.
}
class D : public A
{
    // body of class D.
}
```

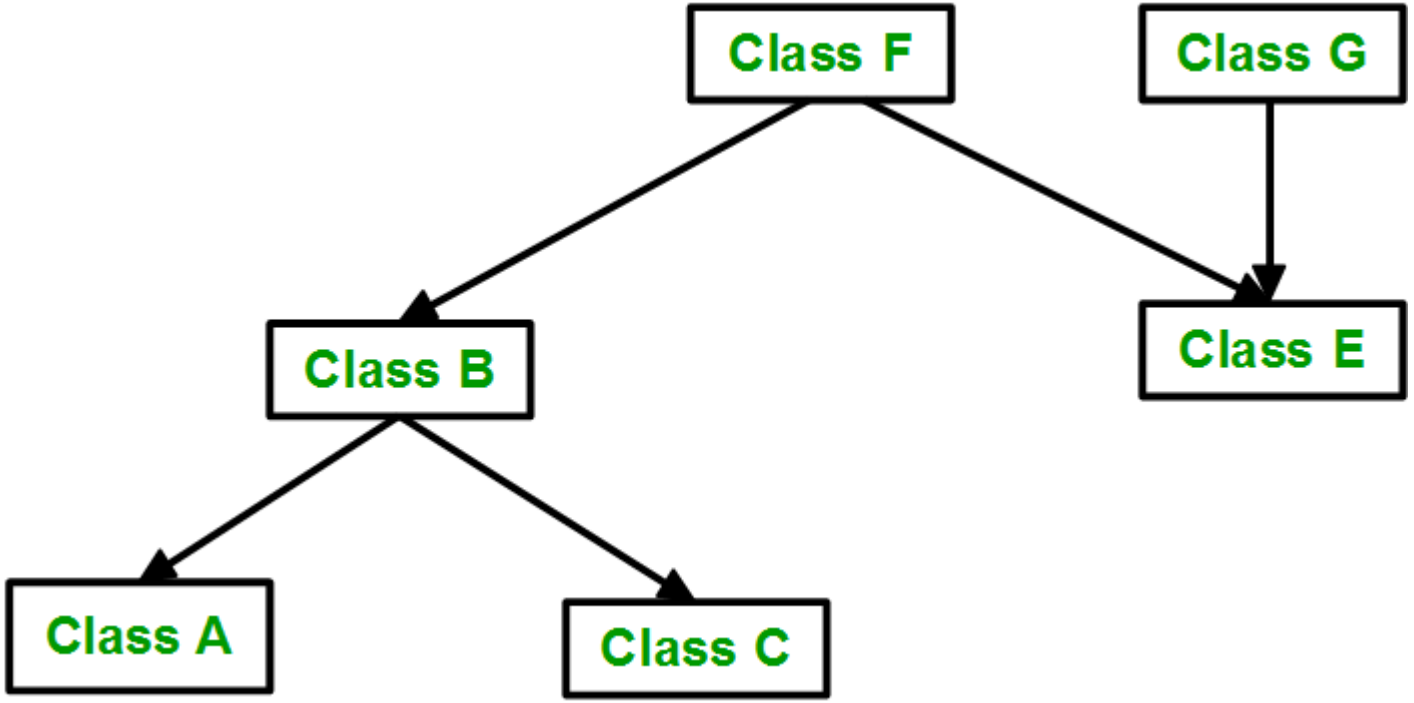
```
// C++ program to implement
// Hierarchical Inheritance
#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// first sub class
class Car : public Vehicle {
};

// second sub class
class Bus : public Vehicle {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
    Car obj1;
    Bus obj2;
    return 0;
}
```

Output

```
This is a Vehicle
This is a Vehicle
```

8.2. Modes and Types of Inheritance

- **5) Hybrid (Virtual) Inheritance:** Hybrid Inheritance is implemented by combining more than one type of inheritance. For example: Combining Hierarchical inheritance and Multiple Inheritance.



8.2. Modes and Types of Inheritance


```
// C++ program for Hybrid Inheritance

#include <iostream>
using namespace std;

// base class
class Vehicle {
public:
    Vehicle() { cout << "This is a Vehicle\n"; }
};

// base class
class Fare {
public:
    Fare() { cout << "Fare of Vehicle\n"; }
};

// first sub class
class Car : public Vehicle {
};

// second sub class
class Bus : public Vehicle, public Fare {
};

// main function
int main()
{
    // Creating object of sub class will
    // invoke the constructor of base class.
    Bus obj2;
    return 0;
}
```

Output

```
This is a Vehicle
Fare of Vehicle
```

8.2. Modes and Types of Inheritance


```
// Example:

#include <iostream>
using namespace std;

class A
{
    protected:
    int a;
    public:
    void get_a()
    {
       cout << "Enter the value of 'a' : ";
       cin>>a;
    }
};
class B : public A
{
    protected:
    int b;
    public:
    void get_b()
    {
       cout << "Enter the value of 'b' : ";
       cin>>b;
    }
};
class C
{
    protected:
    int c;
    public:
    void get_c()
    {
        cout << "Enter the value of c is : ";
        cin>>c;
    }
};
class D : public B, public C
{
    protected:
    int d;
    public:
    void mul()
    {
         get_a();
         get_b();
         get_c();
         cout << "Multiplication of a,b,c is : " <<a*b*c;
    }
};
int main()
{
    D d;
    d.mul();
    return 0;
}
```

Output:

```
Enter the value of 'a' : 1
Enter the value of 'b' : 2
Enter the value of c is : 3
Multiplication of a,b,c is : 6
```

8.3. Inheritance Access

- **1) C++ Public Inheritance**
  - In this example, *public inheritance* is demonstrated.
  - Since private and protected members will not be directly accessed from main(), so we have had to create functions name getPVT() to access the private variable, and getProt() to access the protected variable from the inherited class.

8.3. Inheritance Access


```cpp
// C++ program to demonstrate the working of public
// inheritance
#include <iostream>
using namespace std;

class Base {
private:
    int pvt = 1;

protected:
    int prot = 2;

public:
    int pub = 3;

    // function to access private member
    int getPVT() { return pvt; }
};

class PublicDerived : public Base {
public:
    // function to access protected member from Base
    int getProt() { return prot; }
};

int main()
{
    PublicDerived object1;
    cout << "Private = " << object1.getPVT() << endl;
    cout << "Protected = " << object1.getProt() << endl;
    cout << "Public = " << object1.pub << endl;
    return 0;
}
```

Output

```
Private = 1
Protected = 2
Public = 3
```

8.3. Inheritance Access

- **2) C++ Protected Inheritance**
  - We know that protected members can only be accessed from the Derived class.
  - These members cannot be directly accessed from outside the class.
  - So we cannot use getPVT() from ProtectedDerived. This is also why we need to create getPub() function in the Derived class in order to access the pub variable.

8.3. Inheritance Access


```cpp
// C++ program to demonstrate the working
// of protected inheritance
#include <iostream>
using namespace std;

class Base {
private:
    int pvt = 1;

protected:
    int prot = 2;

public:
    int pub = 3;

    // function to access private member
    int getPVT() { return pvt; }
};

class ProtectedDerived : protected Base {
public:
    // function to access protected member from Base
    int getProt() { return prot; }

    // function to access public member from Base
    int getPub() { return pub; }
};

int main()
{
    ProtectedDerived object1;
    cout << "Private cannot be accessed." << endl;
    cout << "Protected = " << object1.getProt() << endl;
    cout << "Public = " << object1.getPub() << endl;
    return 0;
}
```

Output

```
Private cannot be accessed.
Protected = 2
Public = 3
```

8.3. Inheritance Access

- **3) C++ Private Inheritance**
  - We know that private members cannot be accessed from the Derived class.
  - These members cannot be directly accessed from outside the class.
  - So we cannot use getPVT() from PrivateDerived. So we need to create the getPub() function in PrivateDerived in order to access the pub variable.

8.3. Inheritance Access


```cpp
// C++ program to demonstrate the working
// of private inheritance
#include <iostream>
using namespace std;

class Base {
private:
    int pvt = 1;

protected:
    int prot = 2;

public:
    int pub = 3;

    // function to access private member
    int getPVT() { return pvt; }
};

class PrivateDerived : private Base {
public:
    // function to access protected member from Base
    int getProt() { return prot; }

    // function to access private member
    int getPub() { return pub; }
};

int main()
{
    PrivateDerived object1;
    cout << "Private cannot be accessed." << endl;
    cout << "Protected = " << object1.getProt() << endl;
    cout << "Public = " << object1.getPub() << endl;
    return 0;
}
```

Output

```
Private cannot be accessed.
Protected = 2
Public = 3
```

8.4. Constructor in Multiple Inheritance

- **Constructor** is a class ***member function*** with the same name as the class. The main job of the constructor is to allocate memory for class objects. Constructor is automatically called when the object is created.
- **Multiple Inheritance** is a feature of C++ where a class can derive from several (two, or more) base classes. The constructors of inherited classes are called in the same order in which they are inherited.

8.4. Constructor in Multiple Inheritance

- **Syntax of Multiple Inheritance:**

```
class S: public A1, virtual A2
{
....
};
```

Here,

A2(): virtual base constructor

A1(): base constructor

S(): derived constructor

- **Examples 1 & 2:** Below are C++ programs to show the concept of Constructor in Multiple Inheritance.

8.4. Constructor in Multiple Inheritance


```cpp
// C++ program to implement
// constructor in multiple
// inheritance
#include<iostream>
using namespace std;
class A1
{
  public:
  A1()
  {
    cout << "Constructor of the base class A1 \n";
  }

};

class A2
{
  public:
  A2()
  {
    cout << "Constructor of the base class A2 \n";
  }

};

class S: public A1, virtual A2
{
  public:
  S(): A1(), A2()
  {
    cout << "Constructor of the derived class S \n";
  }
};

// Driver code
int main()
{
  S obj;
  return 0;
}
```

Output

```
Constructor of the base class A2
Constructor of the base class A1
Constructor of the derived class S
```

Time complexity: O(1)

Auxiliary Space: O(1)

8.4. Constructor in Multiple Inheritance


```
// C++ program to implement
// constructors in multiple
// inheritance
#include<iostream>
using namespace std;
class A1
{
    public:
        A1()
        {
            int a = 20, b = 35, c;
            c = a + b;
            cout << "Sum is:" <<
                    c << endl;
        }
};

class A2
{
    public:
        A2()
        {
            int x = 50, y = 42, z;
            z = x - y;
            cout << "Difference is:" <<
                    z << endl;
        }
};

class S: public A1,virtual A2
{
    public:
        S(): A1(), A2()
        {
            int r = 40, s = 8, t;
            t = r * s;
            cout << "Product is:" <<
                    t << endl;
        }
};

// Driver code
int main()
{
    S obj;
    return 0;
}
```

Output

```
Difference is:8
Sum is:55
Product is:320
```

Time complexity: O(1)

Auxiliary Space: O(1)

8.5. Inheritance and Friendship

- **Inheritance in C++:** This is an OOP concept. It allows creating classes that are *derived* from other classes, so that they automatically include some of the functionality of its base class and some functionality of its own.
- **Friendship in C++:** Usually, *private* and *protected* members of a class cannot be accessed from outside the same class in which they are declared. However, a friend class has the access to the protected and private members of the first one.

8.5. Inheritance and Friendship

- Classes that are 'friends' can access not just the public members, but the private and protected members too.
- **Difference between Inheritance and Friendship:** In C++, friendship is not inherited. If a base class has a friend function, then the function doesn't become a friend of the derived class(es).
- For example, the following program prints an ***error*** because the ***show()*** which is a friend of base class *A* tries to access private data of derived class *B*.

8.5. Inheritance and Friendship


```cpp
// CPP Program to demonstrate the relation between
// Inheritance and Friendship
#include <iostream>
using namespace std;

// Parent Class
class A {
protected:
    int x;

public:
    A() { x = 0; }
    friend void show();
};

// Child Class
class B : public A {
private:
    int y;

public:
    B() { y = 0; }
};

void show()
{
    B b;
    cout << "The default value of A::x = " << b.x;

    // Can't access private member declared in class 'B'
    cout << "The default value of B::y = " << b.y;
}

int main()
{
    show();
    getchar();
    return 0;
}
```

Output

```
prog.cpp: In function ‘void show()’:
prog.cpp:19:9: error: ‘int B::y’ is private
    int y;
        ^
prog.cpp:31:49: error: within this context
    cout << "The default value of B::y = " << b.y;
```

8.6. Overloading and Inheritance

- If we have a function in base class, and another function with the same name in derived class, can the base class function be called from derived class object?
- This is an interesting question and as an experiment, predict the output of the following C++ program:

8.6. Overloading and Inheritance


```
#include <iostream>
using namespace std;
class Base
{
public:
    int f(int i)
    {
        cout << "f(int): ";
        return i+3;
    }
};
class Derived : public Base
{
public:
    double f(double d)
    {
        cout << "f(double): ";
        return d+3.3;
    }
};
```

```
int main()
{
    Derived* dp = new Derived;
    cout << dp->f(3) << '\n';
    cout << dp->f(3.3) << '\n';
    delete dp;
    return 0;
}
```

The output of this program is:

```
f(double): 6.3
f(double): 6.6
```

Instead of the supposed output:

```
f(int): 6
f(double): 6.6
```

8.6. Overloading and Inheritance

- Overloading does not work for derived class in the C++ programming language.
- There is no overload resolution between Base and Derived. The compiler looks into the scope of Derived, finds the single function “double f(double)” and calls it. It never disturbs the (enclosing) scope of Base.
- In C++, *there is no overloading across scopes* and derived class scopes are not an exception to this general rule.
- In Java, overloading works across scopes, contrary to C++.