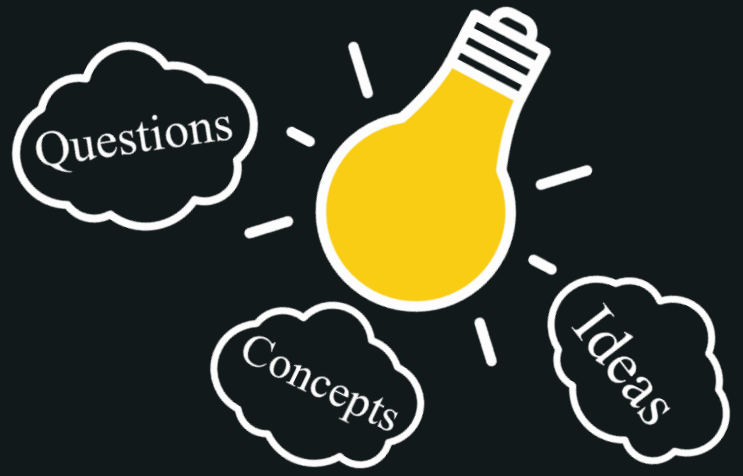


Inheritance

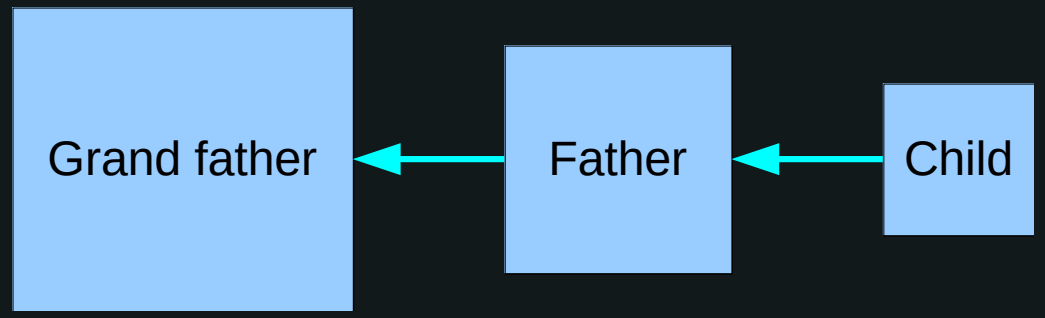
C++ Programming

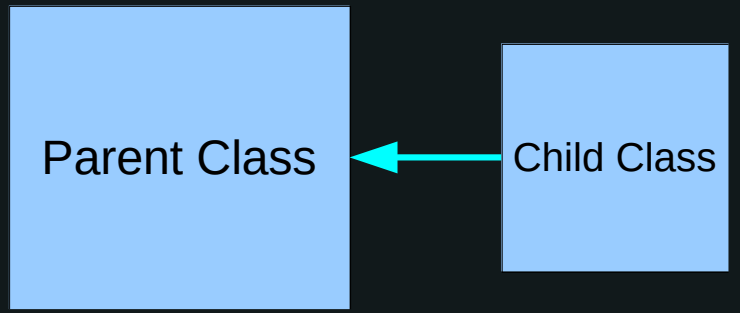
VEDINESH

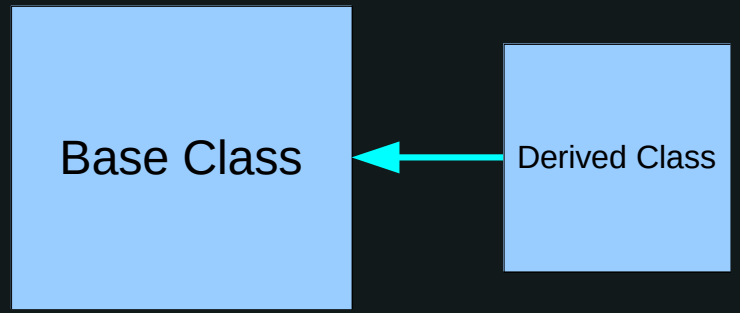
What is Inheritance ?

VEDINESH

So, Inheritance is a mechanism in which one class acqires the property of another class .



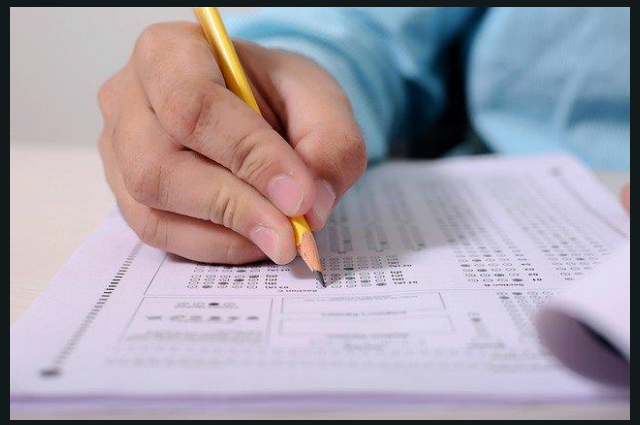




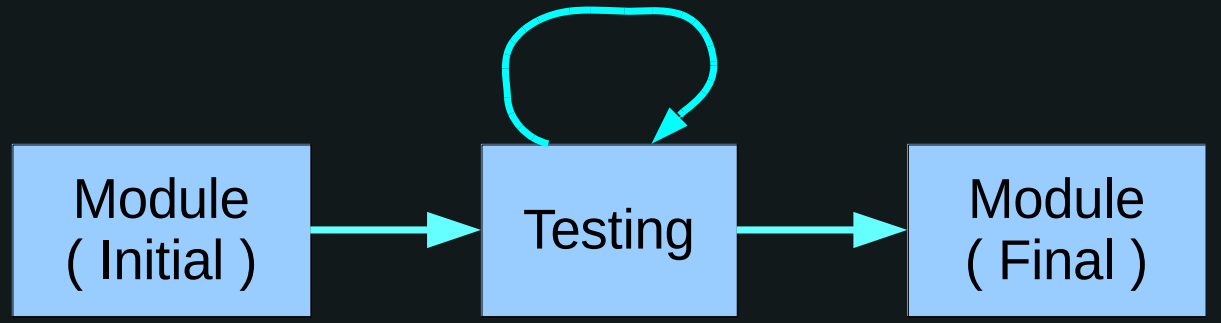
Why Inheritance ?

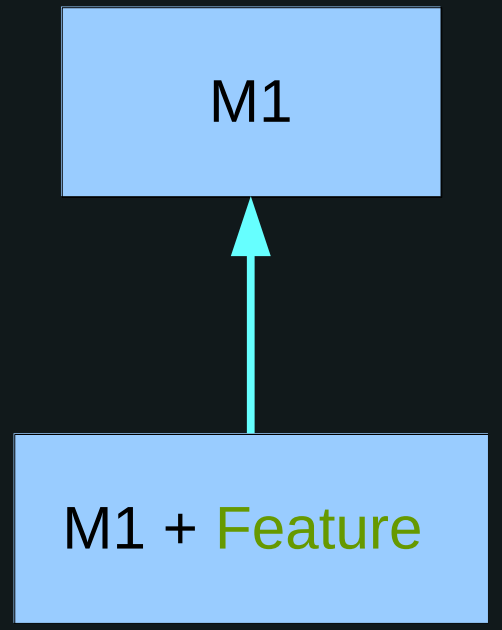
VEDINESH

Test / Exam

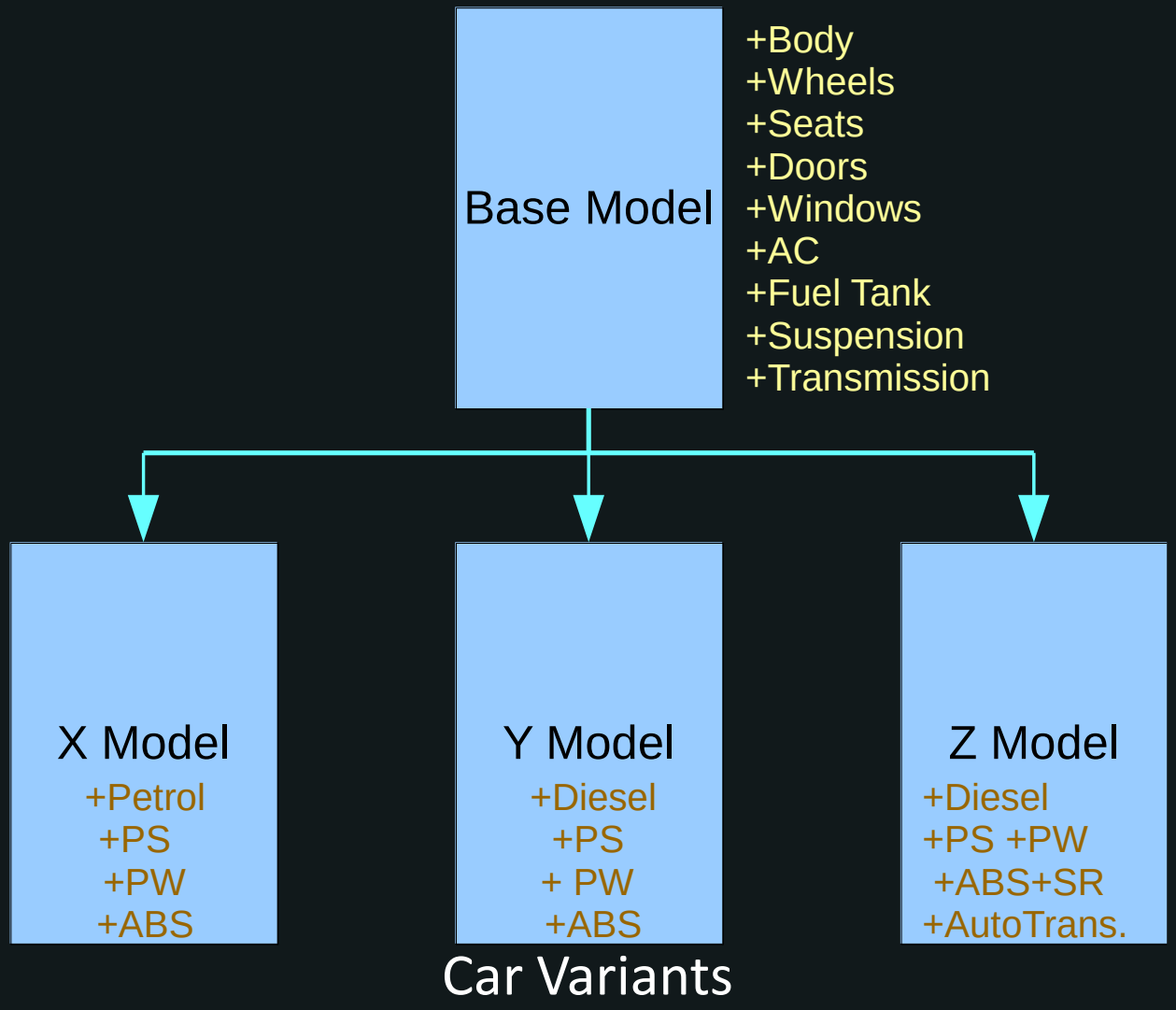


1. Waste of Time
2. Resource , Cost
3. Not feasible





Inheritance Approach



Why Inheritance ?

1. Reduce Duplicate Code
2. Code Reuse
3. Better Organization of Code

Inheritance Example

VEDINESH

```
class rectangle
{
public:
   int length;
   int breadth;

   void show(  )
   {  cout << length;
      cout << breadth;   }
 };
```

```
void main( )
{ rectangle r;
    r.length =10; r.breadth = 20;
    r.show ( );
 };
```

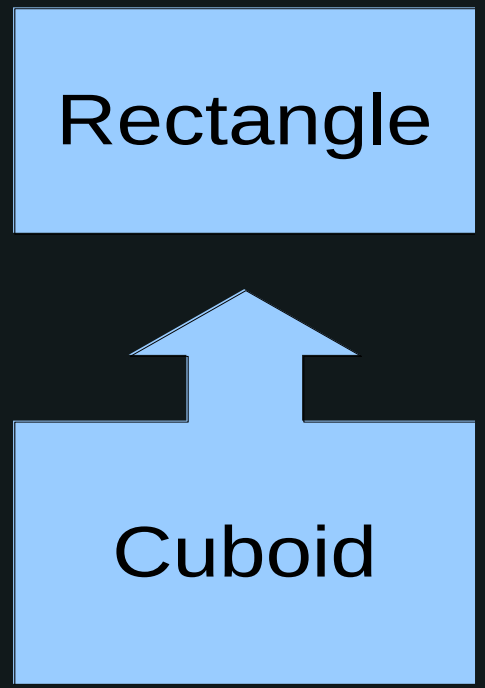
```
class cuboid : public rectangle
{
public:
   int height;

   void display (  )
   {  cout << height; }
 };
```

```
void main( )
{ cuboid c;
   c.length =10; c.breadth = 20; c.height = 30;
   c.show ( );
   c.display ( );
 }
```

Inheritance Example 2

VEDINESH

Write a program in which cuboid class inherit rectangle class and calculate area and volume.



Constructors & Inheritance

VEDINESH

```
class base
{
public:
   base(  )
   { cout << “Default Of Base Class”;  }

   base( int b )
   { cout << “Paramaterized Of Base Class”; << b }
 };
```

```
class derived: public base
{
    // Empty
};
```

```
void main( )
{
   derived d1;

   derived d2( 9 );
}
```

NOTE:- If we don't specify a constructor, then derived class will use appropriate constructor from baseclass.
(Applicable only to Default Constructor)

Constructors & Inheritance

VEDINESH

```
class base
{
public:
  base( )
  { cout << "Default Of Base Class";  }

  base( int b )
  { cout << "Paramaterized Of Base Class" << b; }
 };
```

```
class derived: public base
{
public:
  derived( )
  { cout << "Default Of Derived Class";  }

  derived( int d )
  { cout << "Paramaterized Of Derived Class" << d; }
 };
```

```
void main( )
{   derived d1;
    derived d2( 9 );
}
```

NOTE:- 1st Default Constructor Of base class , then Default Constructor of derived class is called.

NOTE:- 2nd Parametrized Constructor of base class is not called when Para. Constructor is present in derived class.

Constructors & Inheritance

VEDINESH

```
class base
{
public:
  base( )
  { cout << “Default of Base Class”; }

  base( int b_arg )
  { cout << “Para of Base Class”; << b_arg }
 };
```

```
class derived: public base
{
public:
 derived( ) : base( )
 { cout << “Default of derived Class”; }

 derived( int d_arg ) : base( d_arg )
 { cout << “Para of derived Class”; }
};
```

```
void main ( )
{
   derived d1;
   derived d2( 9 );
}
```

NOTE:- Derived Class constructor can call base class constructor.

Overriding Member Function

VEDINESH

```
class base
{
public:
  void Msg( )
  {
     cout << “Base Class” ;
  }
};
```

```
class derived: public base
{
public:
   void Msg( )
   {
      cout << “Derived Class”;
   }
};
```

```
void main ( )
{
   base b;
   b.Msg( );

   derived c;
   c.Msg( );
}
```

NOTE:-
Redefining functionality of BASE class into DERIVED class, then if we create OBJECT of DERIVED class

NOTE:-

b.Msg( ) --------- > Base Class

c.Msg( ) --------- > Derived Class

Overriding Member Function

VEDINESH

```
class base
{
public:
  void Msg( )
  {
     cout << "Base Class" ;
  }
};
```

```
class derived: public base
{
public:
   void Msg( )
   {
      cout << "Derived Class";
   }
};
```

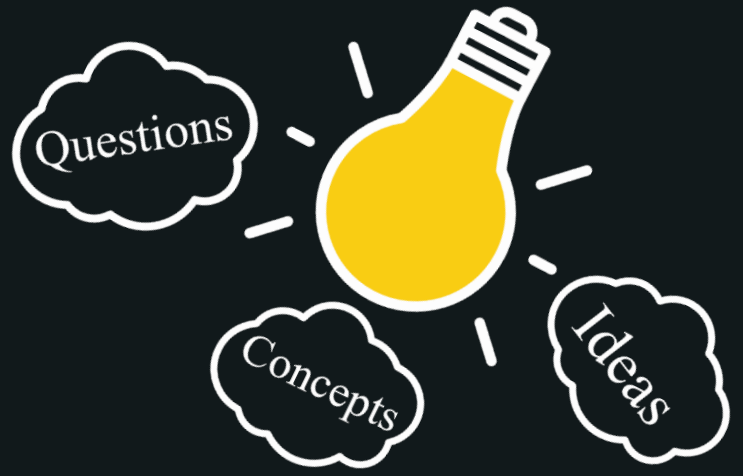
```
void main ( )
{
   derived c;
   c.Msg( );
}
```

NOTE:- Derived class object would call, function in derived class, if same function exists in both classes.

```
class base
{
public:
  void Msg( )
  {
     cout << "Base Class" ;
  }
};
```

```
class derived: public base
{
public:
   void Msg( )
   {  cout << "Derived Class";
      base::Msg( );   // calling
   }
};
```

```
void main ( )
{
   derived c;
   c.Msg( );
}
```



Relations

C++ Programming

VEDINESH

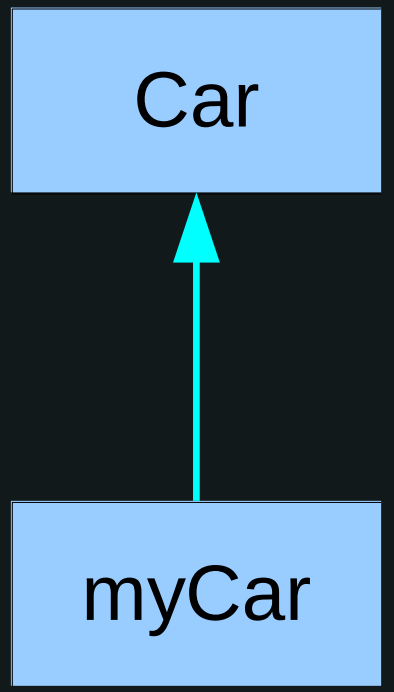
isA Relationship

VEDINESH

```
class Suzuki
{
public:
  void chechis( )
  { . . . . . .. }
  void engine( )
  { . . . . . .. }
  void suspension ( )
  { . . . . . .. }
  void transmission( )
  { . . . . . .. }
  void doors( )
  { . . . . . .. }

};
```

```
class DODO: public Suzuki
{
      void chechis( )
     { . . . . . modify. . . . . .. }

      void doors( )
     { . . . . . modify. . . . . .. }

      void ABS ( )    // added new feature
     { . . . . . modify. . . . . .. }
 };
```

DODO isA Car



isA = Inheritance

hasA Relationship

VEDINESH

```
class Suzuki
{
public:
  void chechis( )
  { . . . . . .. }
  void suspension ( )
  { . . . . . .. }
  void transmission( )
  { . . . . . .. }
  void doors( )
  { . . . . . .. }
};
```

```
class Antoinette
{
public:
  void V8_Engine( )
  {
     . . . . . .
  }
};
```

```
class DODO
{
private:
  Suzuki design_obj;
  Antoinette anto_obj;

public:
  void addChechis()
  { design_obj.chechis( ); }

  void addEngine( )
  { anto_obj.V8_Engine( ); }

 };
```

hasA = Object

isA vs hasA

VEDINESH

isA relationship is based on Inheritance.

hasA relationship is based on Objects.

isA relationship expose all public data of base classes.

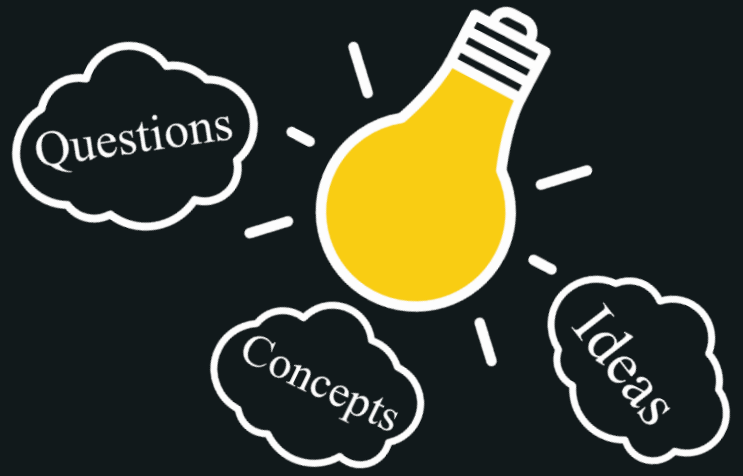
hasA relationship, use public data of derived class.

isA relationship is static binding (compile time).

hasA relationship is dynamic binding (run time).

isA relationship used when can actually inherit .
(person ---------- teacher)
(person ---------- dress)

hasA relationship use when you can't inherit something.



Types Of Inheritance

C++ Programming

VEDINESH

Type of Inheritance

VEDINESH

1. Simple Inheritance

```
class  bicycle
{
   . . . . . .  . . . . . .
 };
```

```
class motorcycle: public bicycle
{
     . . . . . .  . . . . . .
 };
```

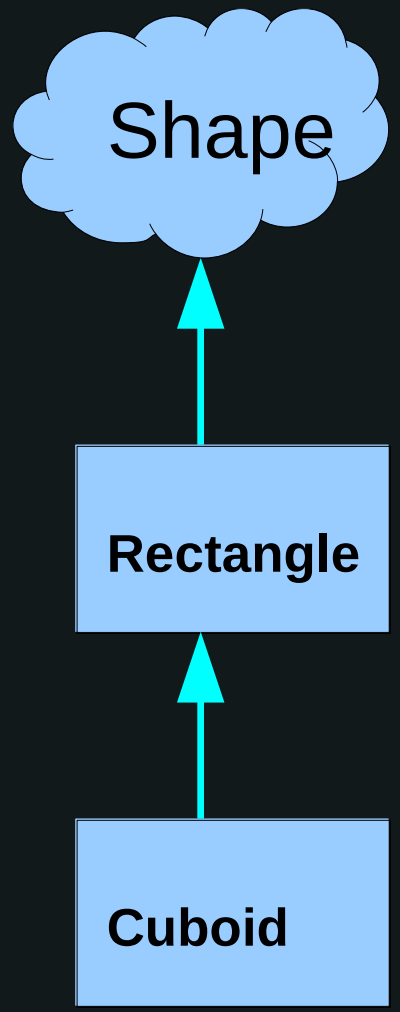
Type of Inheritance

2. Multilevel Inheritance

```
class  shape
{   . . . . . . . . . . . . . . . . . . . .
 };
```

```
class  rectangle : public shape
{   . . . . . . . . . . . . . . . . . . . .
 };
```

```
class cuboid : public rectangle
{      . . . . . . . . . . . . . . . . . . . .
 };
```



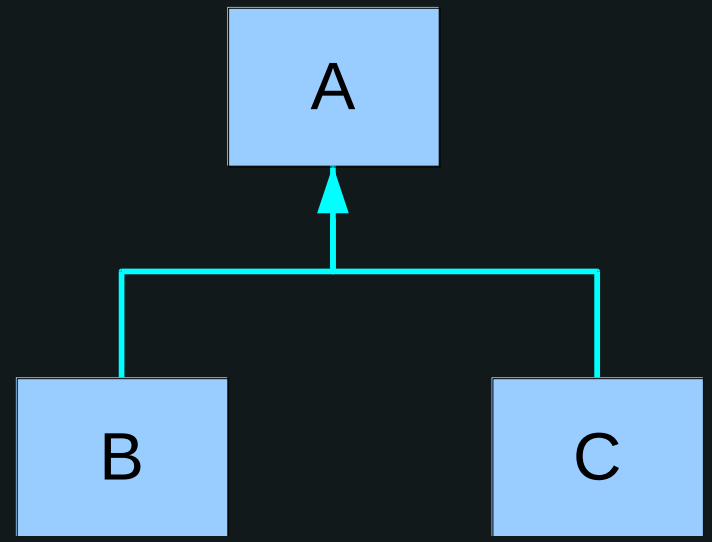
Type of Inheritance

3. Hierarchical Inheritance

```
class  Person
{   . . . . . .  . . . . . .
};
```

```
class Teacher : public Person
{      . . . . . .  . . . . . .
};
```

```
class Student : public Person
{      . . . . . .  . . . . . .
};
```



Type of Inheritance

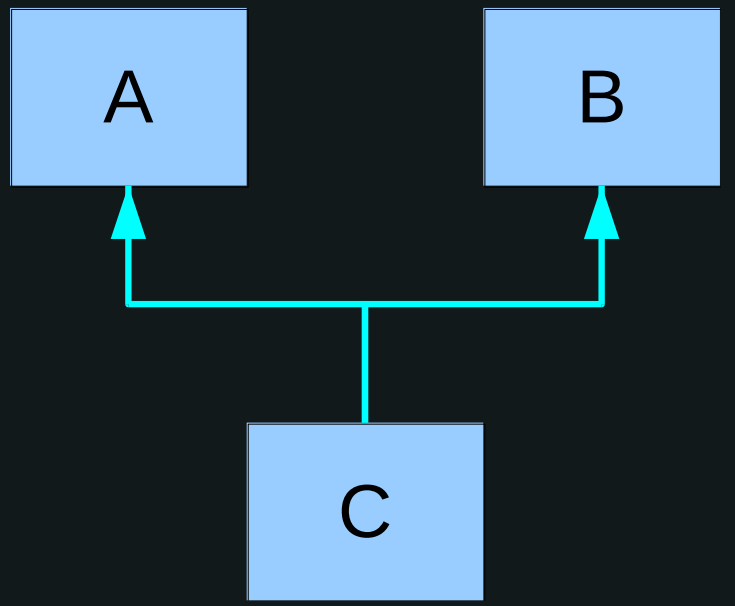
VEDINESH

4. Multiple Inheritance

```
class  Person
{   . . . . . . . . . . . . . . . . . . . .
};
```

```
class  Employee
{   . . . . . . . . . . . . . . . . . . . .
};
```

```
class Teacher : public Person, public Employee
{      . . . . . . . . . . . . . . . . . . . .
};
```



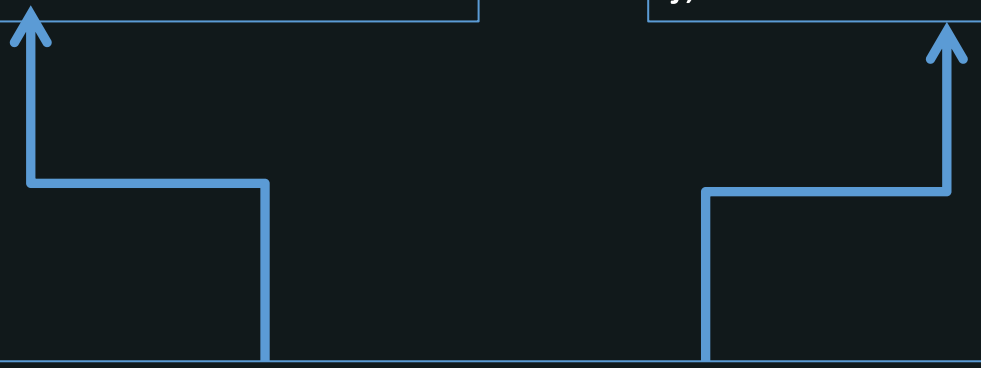
Type of Inheritance

VEDINESH

4. Multiple Inheritance Ambiguity

```
class  Person
{    void show( )
      { cout << "Person "; }
};
```

```
class  Employee
{    void show( )
      { cout << "Employess"; }
};
```



```
class Teacher : public Person, public Employee
{
};
```

```
void main( )
{    Teacher obj;
      obj.Person::show( );
      obj.Employee::show( );
};
```

Type of Inheritance

VEDINESH

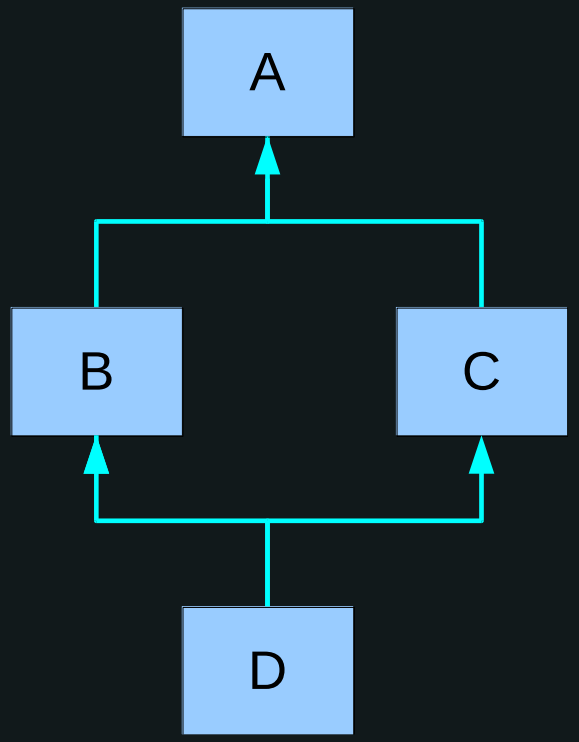
5. Multipath Inheritance

```
class  Person
{   . . . . . . . . . . . . . . . . . . . . . . .
};
```

```
class Teacher : public Person
{      . . . . . . . . . . . . . . . . . . . . .
};
```

```
class Student : public Person
{      . . . . . . . . . . . . . . . . . . . . .
};
```

```
class School : public Teacher, public Student
{      . . . . . . . . . . . . . . . . . . . . .
};
```



Diamond Problem

Type of Inheritance

VEDINESH

5. Multipath Inheritance

```
class  Person
{   . . . . . . . . . . . . . . . . . . . . . .
};
```

```
class Teacher : Virtual public Person
{      . . . . . . . . . . . . . . . . . . . .
};
```

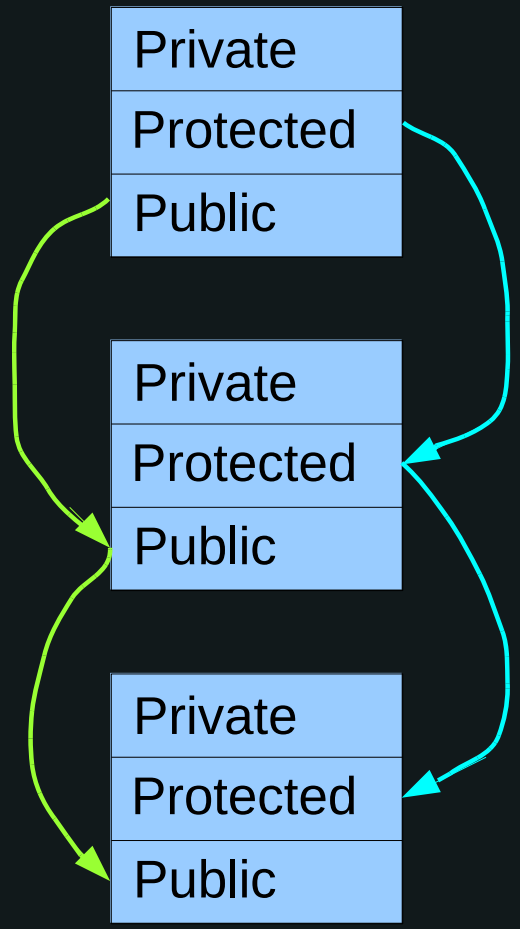
```
class Student : Virtual  public Person
{      . . . . . . . . . . . . . . . . . . . .
};
```

```
class School : public Teacher, public Student
{      . . . . . . . . . . . . . . . . . . . .
};
```

Ways Of Inheritance

VEDINESH

class Parent



Child AND grandchild would not be able to access both Private.

class child : public Parent

Child AND grandchild would be able to access both Protected AND public

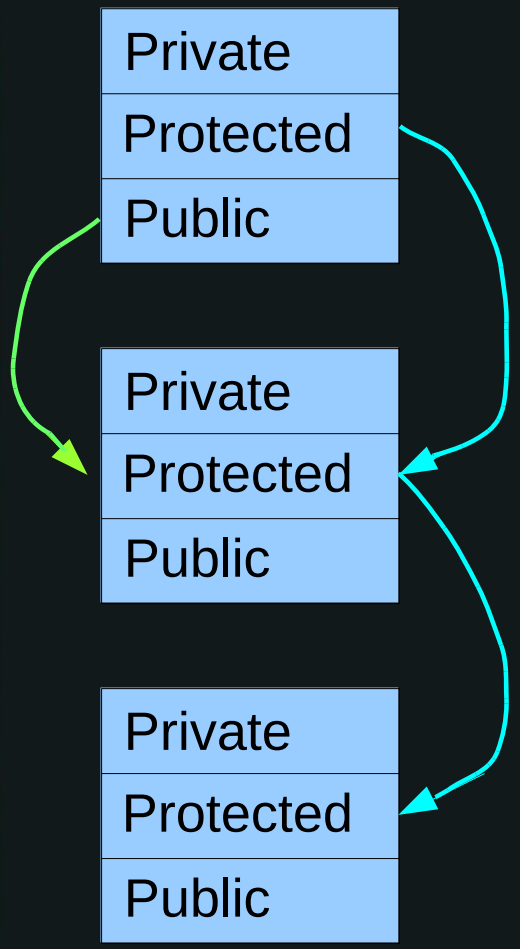
class grandchild : public child

Protected would be inherited as Protected AND
Public would be inherited as Public in Child AND Grandchild

Ways Of Inheritance

VEDINESH

class Parent



class child : protected Parent

class grandchild : protected child

Protected Inheritance:-
Protected AND Public in Parent class would be inherited as Protected in Child And GrandChild classes.