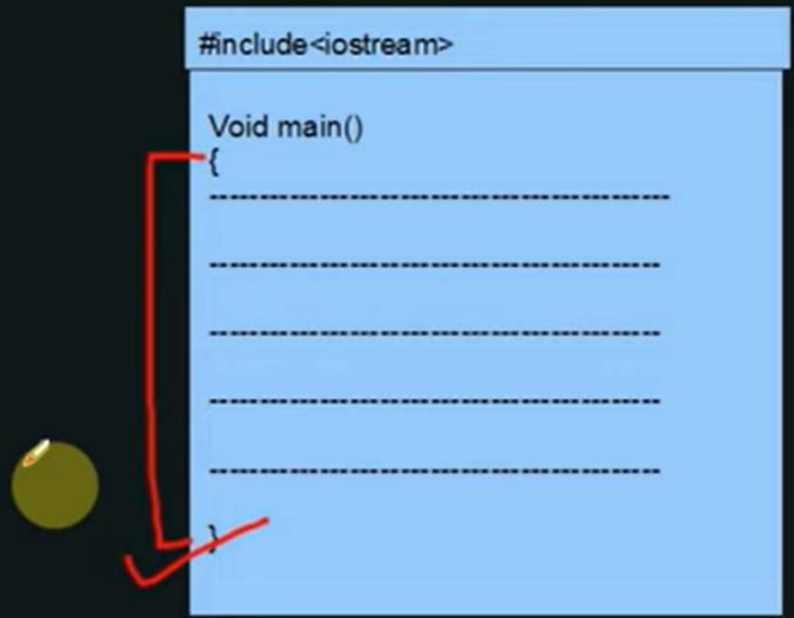
Why OOPs ?

VEDINESH

1. Procedural Programming :- It is a list of instruction in a single block.

Suitable for small program .



Why OOPs ?

VEDINESH

2. Modular Programming :- In this procedural program is divided into functions & each function has a clear purpose.


```
#include<iostream>

Void main()
{

-----------------------------------------

        Function1( );

-----------------------------------------

        Function2( );

-----------------------------------------

}

Void Function1( )
{

 ---------------------------------------

}
Void Function2( )
{
 ---------------------------------------

}
```

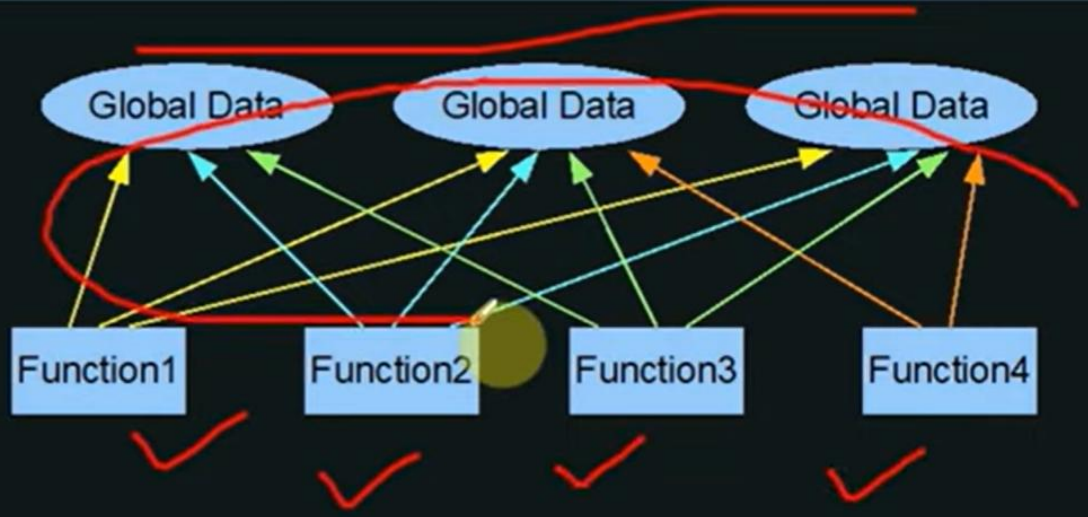
Problems With Modular Programming

Data remains alive within module, so we need some data to global.

Problems With Modular Programming

VEDINESH

Data remains alive within module, so we need some data to global.



ATM application

withdrawal( )

check_balance( )

gen. PIN( )

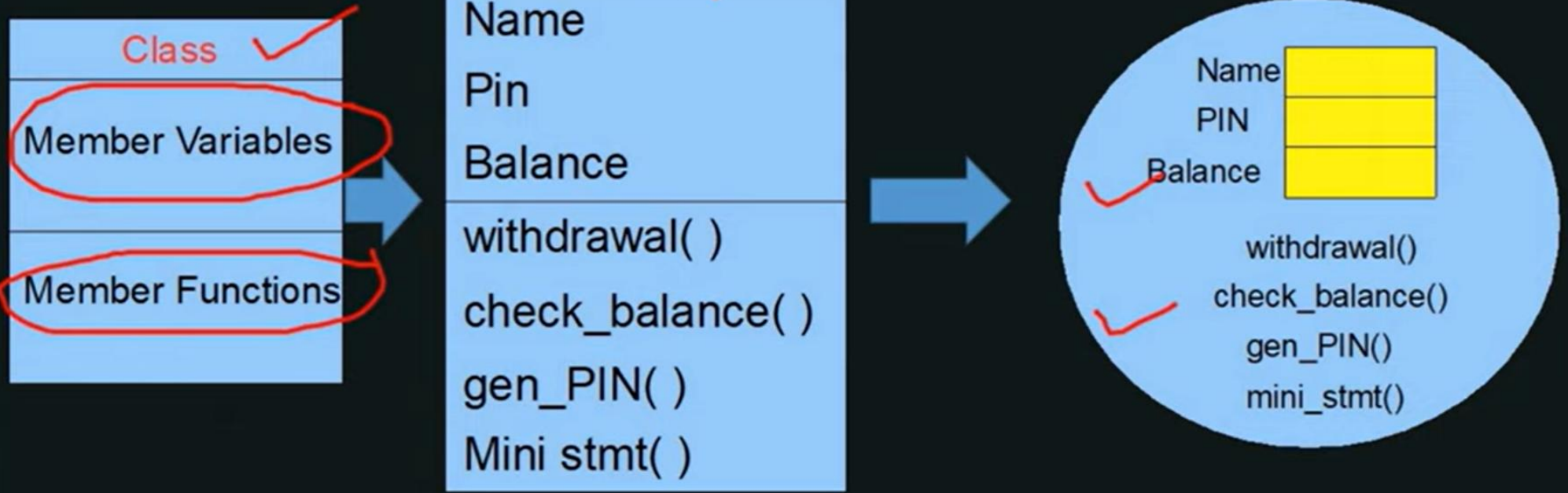
mini stmt( )

Balance

PIN

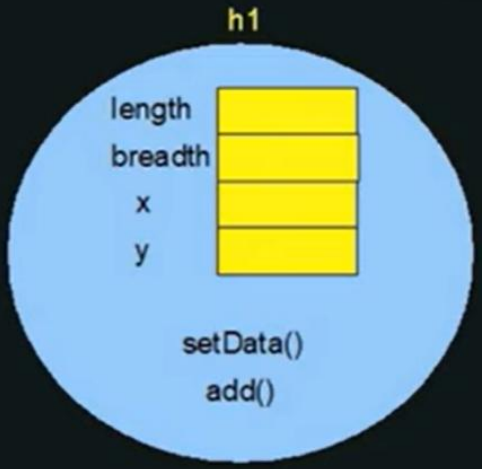
name

Object Oriented Approach



OOP's Example

VEDINESH



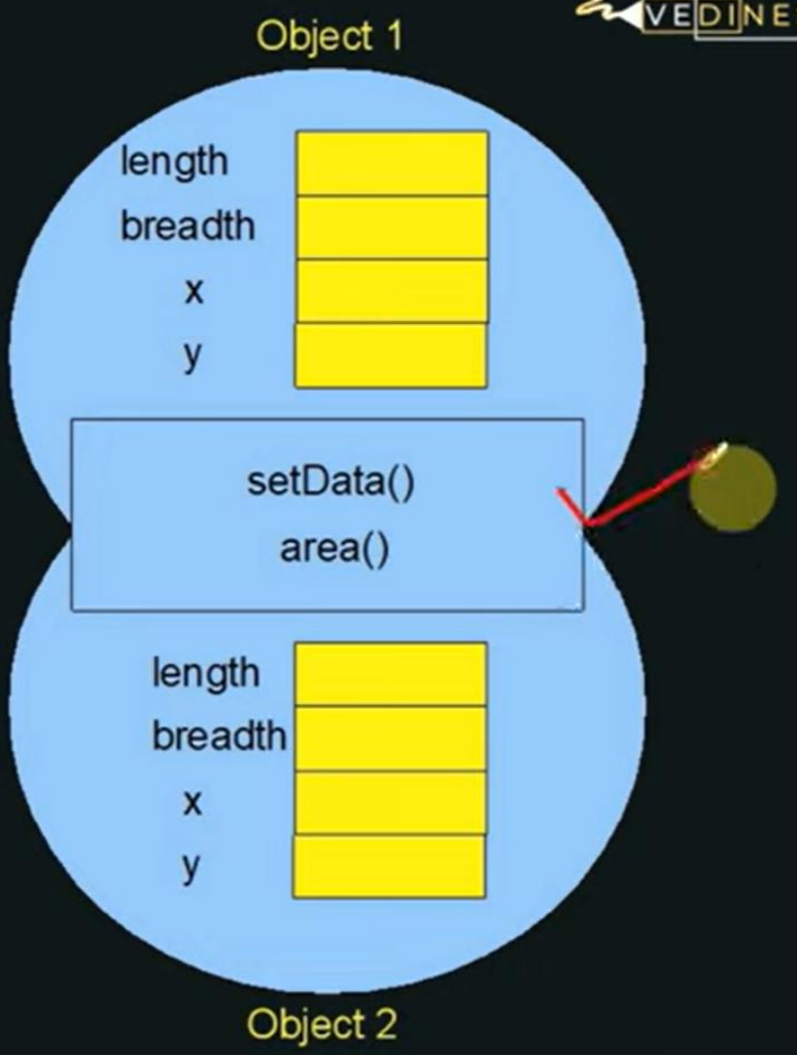

```
class house
{
  int length, breadth;              // member variable

  void setData(int x , int y)       // member function
   { length = x;   breadth = y; }

  void  area ()
   { cout << length * breadth ; }
}
```

```
void main ()
{

  house h1;          // memory allocated

  h1.setData( 500, 600 );
  h1.area();

 }
```

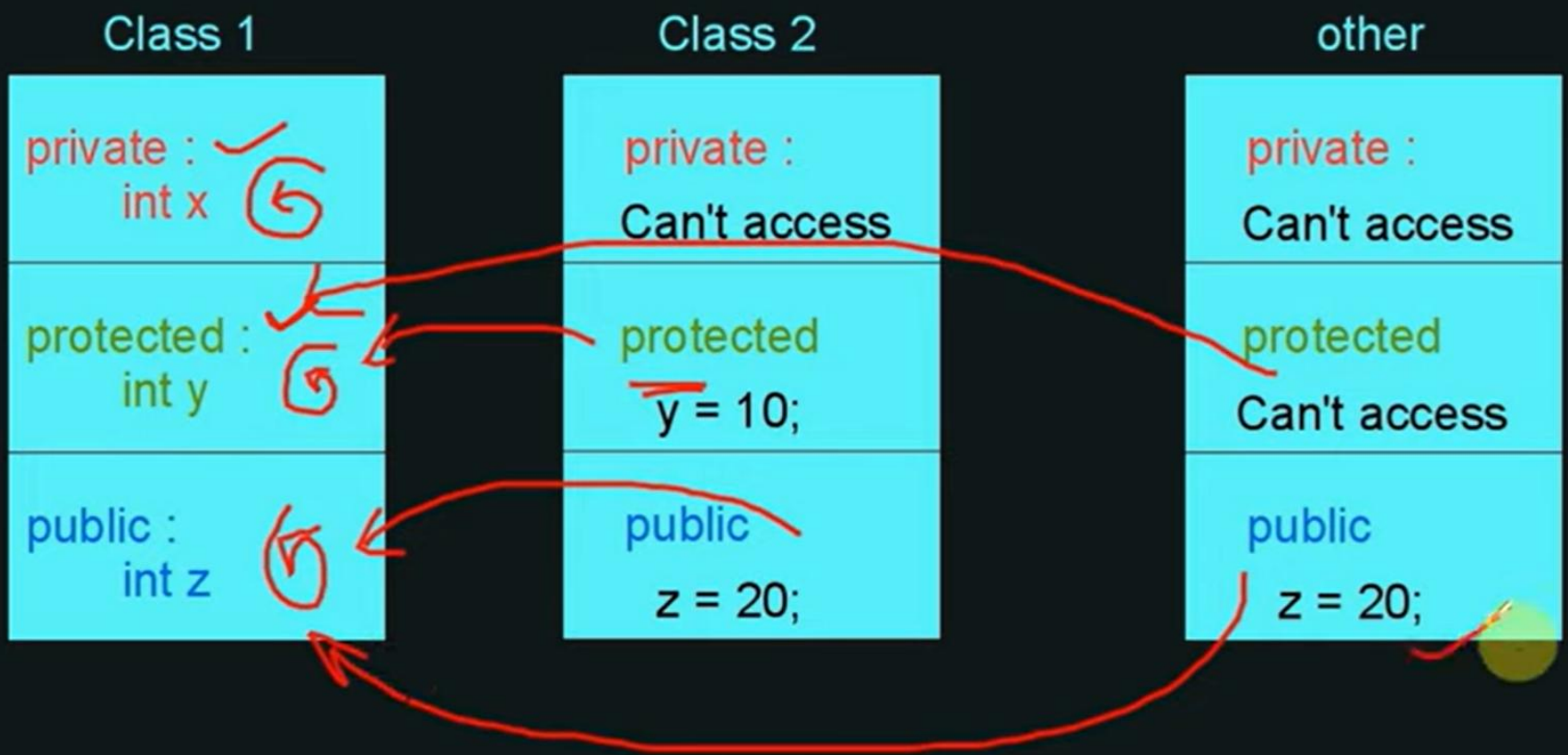
Key Note

```
class house
{
  int length, breadth;                    // member variable

  void setData(int x , int y)             // member function
   { length= x;   breadth = y; }
  void  area()
   { cout << length*breadth; }
}
```

```
void main ()
{
house h1, h2;                  // memory allocated
  h1.setData( 5, 6 );
  h1.area( );
-----------------------------------------
 h2.setData( 7, 1 );
 h2.area( );
 }
```



Access specifiers



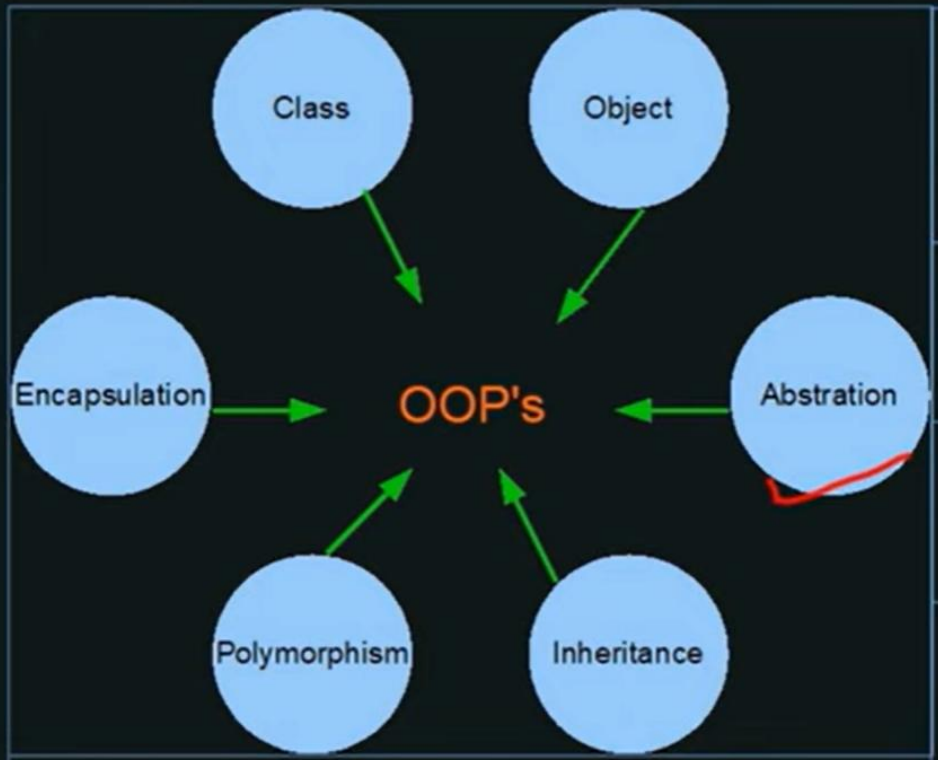
Access specifiers

VEDINESH

```
class house
{
private:
  int length, breadth;          // member variable
 public:
  void setData(int x , int y)    // member function
   { length= x;   breadth= y; }
  void  area ()
   { cout << length*breadth; }
}
```

Characteristics Of OOP's

VEDINESH



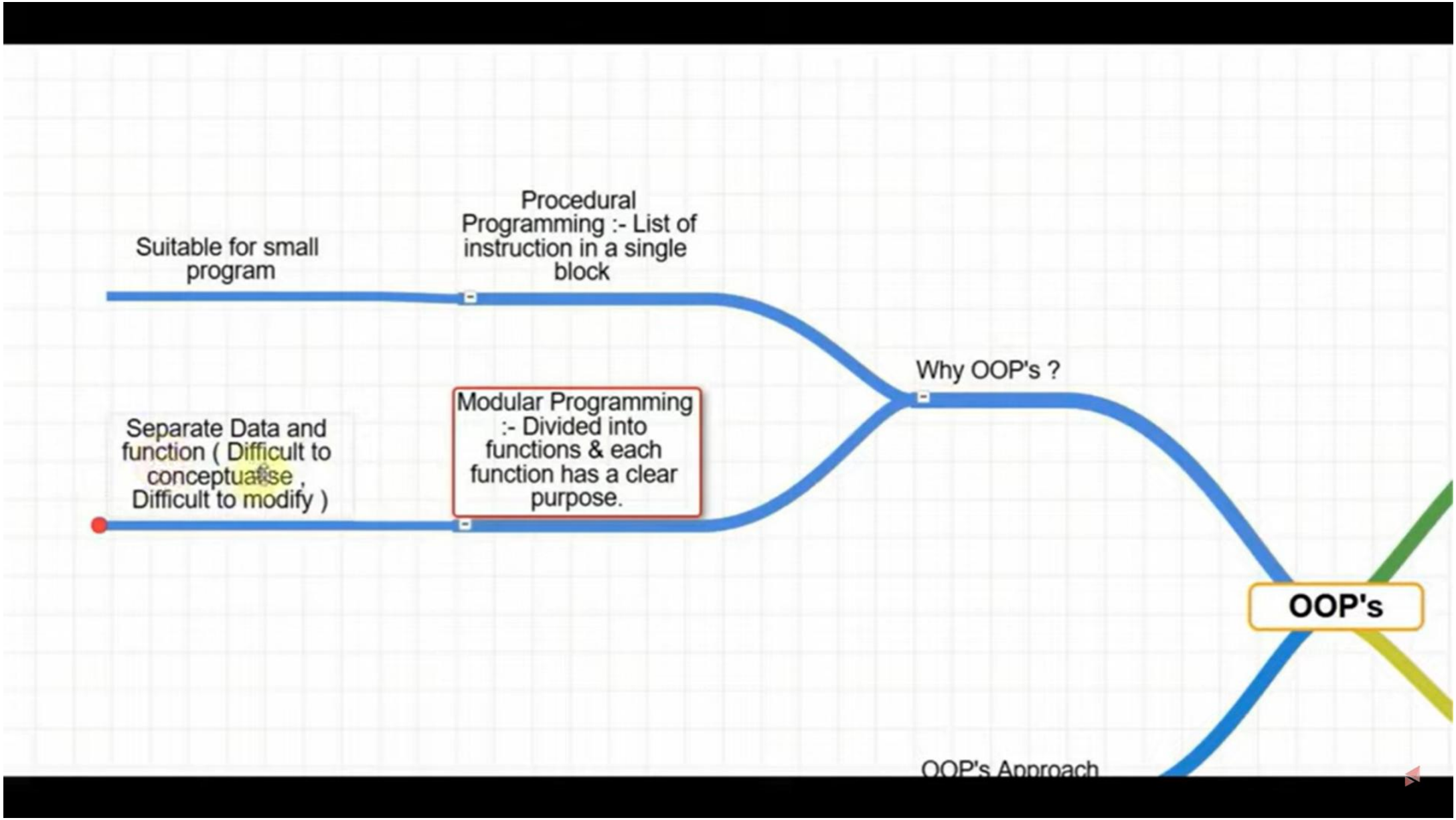
-> Abstraction:- means hiding complicated things from the user.

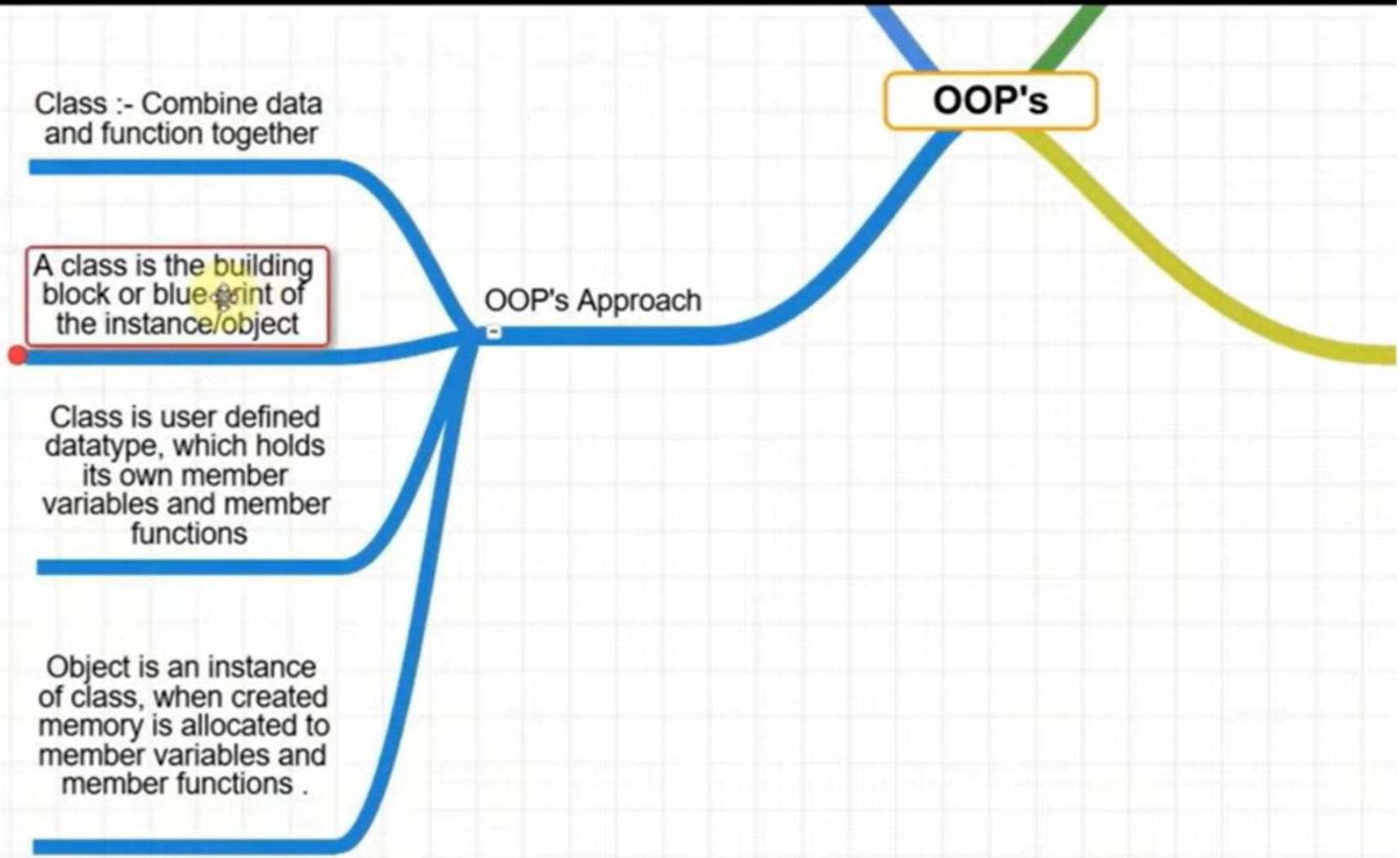
-> Class is a blueprint and Object is instance of class.
-> Class is a user-defined data type, which holds its own data members and member functions.
-> Helps in code reusability.

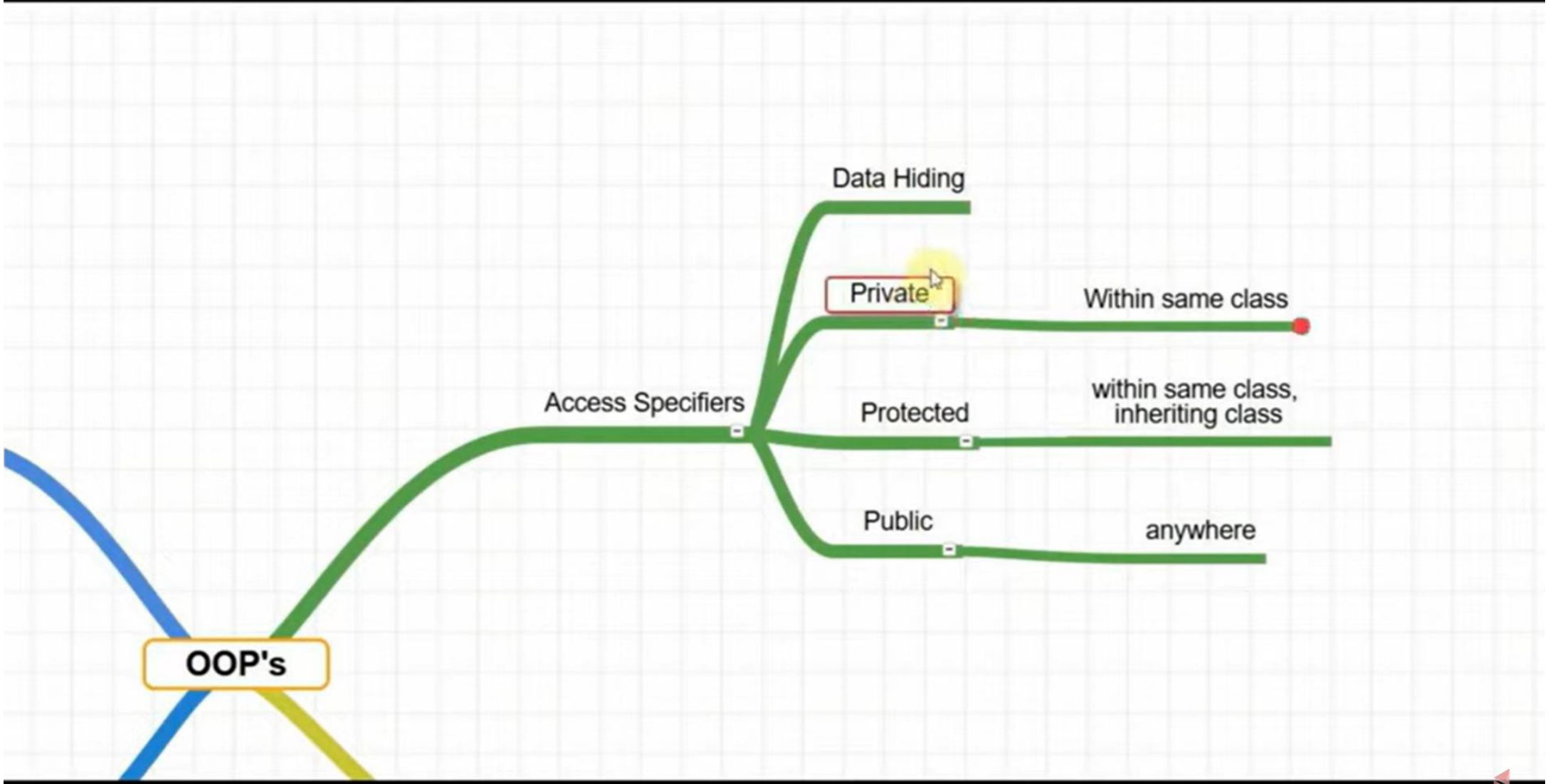
-> Encapsulation wraping up variables and methods in class.
-> It help in data hiding.

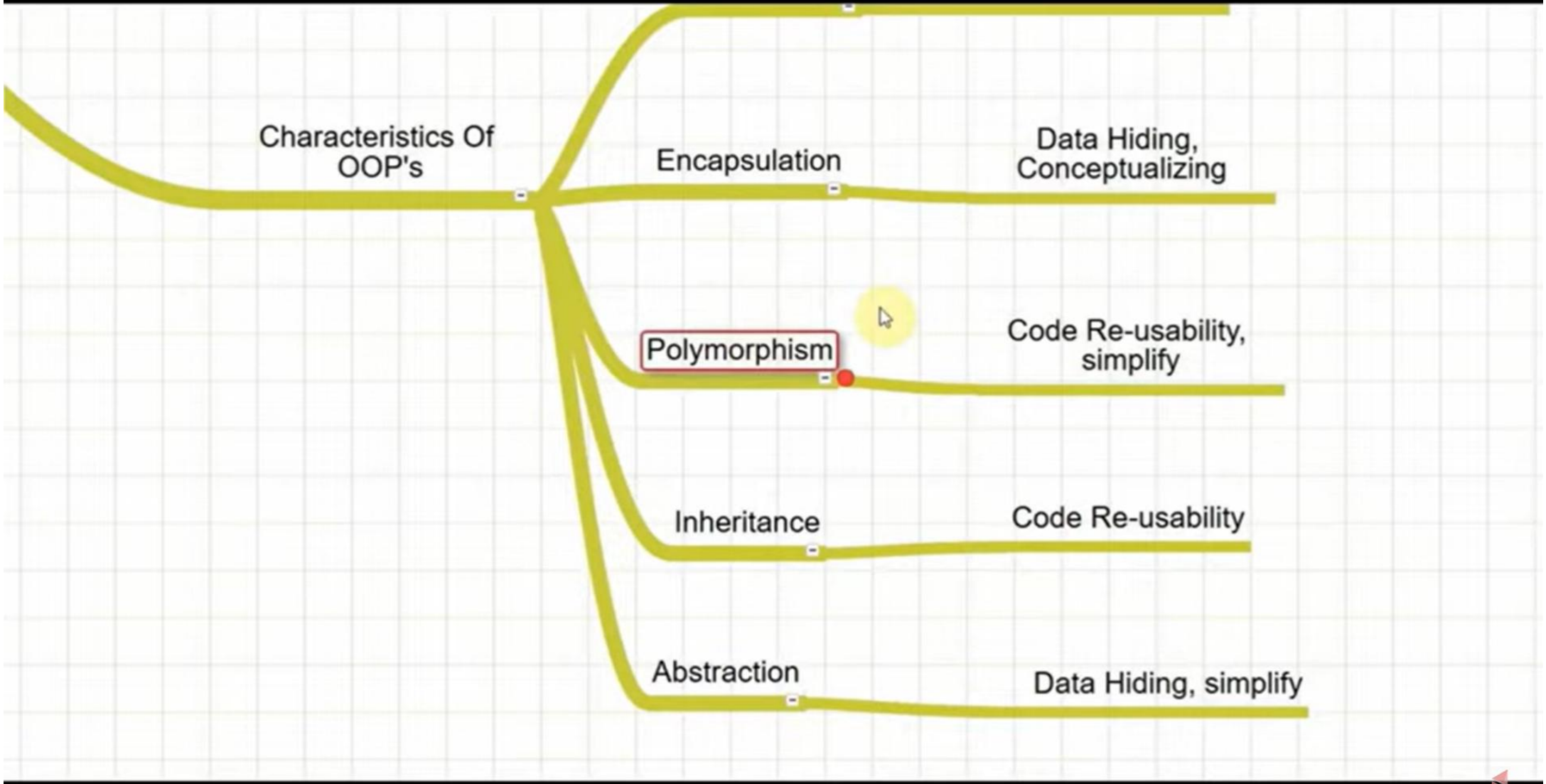
-> Polymorphism means having many forms
-> In class method may behave differently, depending on the inputs. function overloading

-> Inheritance means property of a child class to inherit characteristic of parent class.
like :-
Dog, Cat , Cow Class Inherit from Animal Class.

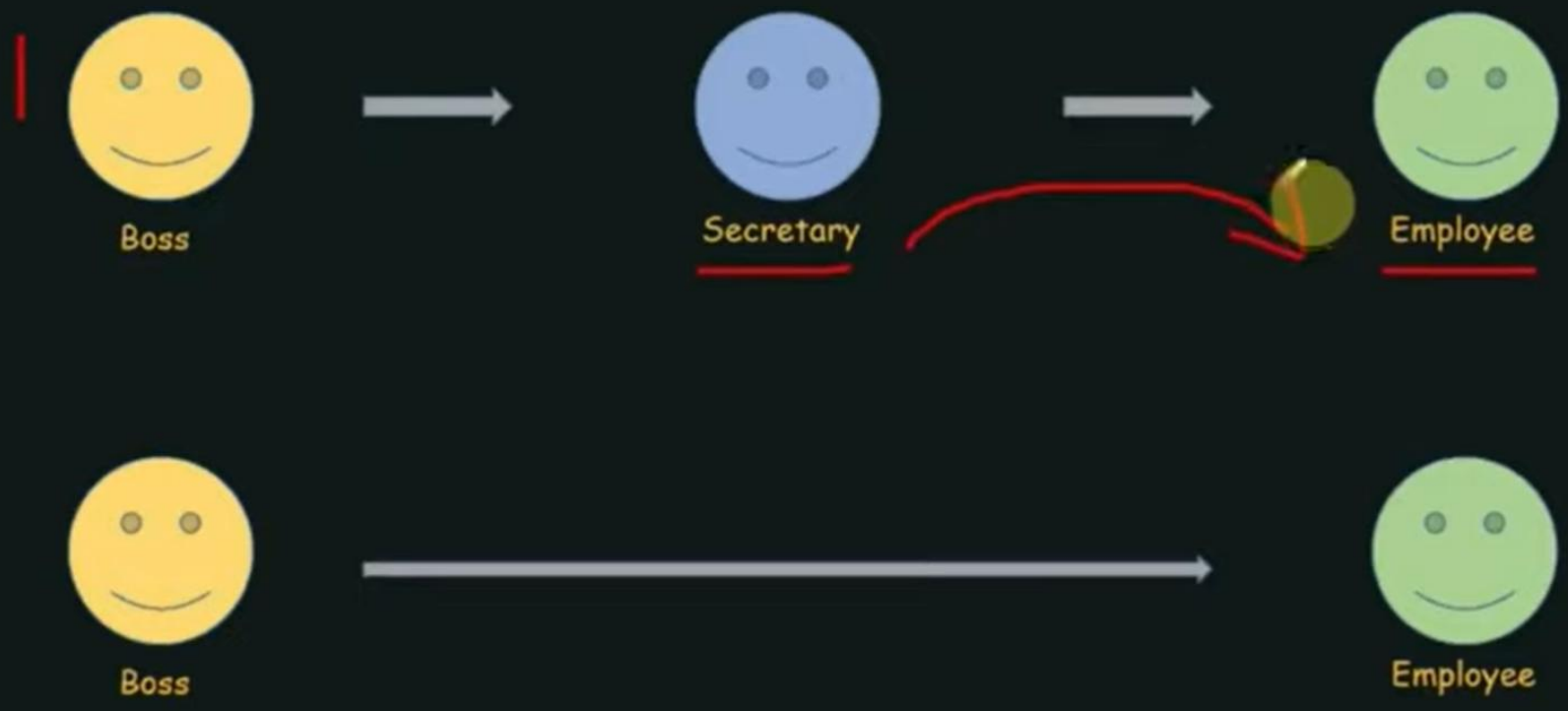

Suitable for small program
Procedural Programming :- List of instruction in a single block
Separate Data and function (Difficult to conceptualise , Difficult to modify)
Modular Programming :- Divided into functions & each function has a clear purpose.
Why OOP's ?
OOP's
OOP's Approach


OOP's
OOP's Approach
Class :- Combine data and function together
A class is the building block or blue-print of the instance/object
Class is user defined datatype, which holds its own member variables and member functions
Object is an instance of class, when created memory is allocated to member variables and member functions .


Data Hiding
Private
Within same class
Access Specifiers
Protected
within same class,
inheriting class
Public
anywhere
OOP's


Characteristics Of OOP's
Encapsulation
Data Hiding, Conceptualizing
Polymorphism
Code Re-usability, simplify
Inheritance
Code Re-usability
Abstraction
Data Hiding, simplify

Constructors



Constructors

VEDINESH

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ------------------------------------------------
   void setData( int x = 0 )
   { age = x; }
   ------------------------------------------------
   int getData( )
   { return age ; }
  ------------------------------------------------
}
```

```
void main ( )
{
  A a_obj ;
  a_obj.setData ( 28 );
  cout << a_obj.getData ( ) ;
}
```

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ------------------------------------------------
   A ( int x )          // constructor
   { age = x;  }       // same name as class & don't return
    ------------------------------------------------
    int getData( )
   { return age ; }
}
```

```
void main ( )
{
  A a_obj ( 28 );
  cout << a_obj.getData ( ) ;
}
```

Constructors

VEDINESH

Why :-

> Programmer may forget to initialize data members in object after creating it.

> When there are many objects, then it would be tedious job.

> Initialize & Allocate memory to Data Members .

Rules :-

> Same Name As Class Name.

> No Return Type.

Constructor Types

VEDINESH

> Non - Parametrized Constructor. or > Default Constructor.

> Parametrized Constructor.

> Copy Constructor.

Non-Parametrized Constructor

VEDINESH

> Constructor that does not take any argument.

```
#include<iostream>
 class A
{ private:
   int age;
   public:
   ------------------------------------------------------
   A ( )                  // Non Parametrized constructor
   { age = 0; }           // same name as class & don't return anything
   ------------------------------------------------------
    int getData( )
   { return age ; }
}
```

Copy Constructor

VEDINESH

```
#include<iostream>
 class A
{ private:
  int age;
  public:
  ---------------------------------------------------
  A ( int x )                  // Parametrized constructor
  { age = x; }
  ---------------------------------------------------
  A ( A &a_obj1 )              // Copy constructor

  {  age = a_obj1.age;  }
 ---------------------------------------------------
   int getData( )
  { return age ; }
}
```

```
void main ( )
{
  A a_obj1 ( 28 );             // Parametrized Constructor

  A a_obj2 ( a_obj1 ) ;        // Copy Constructor

   cout << a_obj2.getData ( ) ;
}
```

Program

VEDINESH

Write a program, take Phone details as input and store them in object & use Constructors.

Phone Details :-

1. Name
2. RAM
3. Processor
4. Batter

Operator Overloading

VEDINESH

When we make operators(+ , - , / , *) work for user-defined types like objects and structures. This is known as operator overloading

Operator Overloading

VEDINESH

```
#include<iostream>
void main( )
{
  int a = 63, b = 74, c = 0;
  c = a + b ;

  A person1( 63 );
  A person2 ( 74 );
  A total;
  total = person1.addWeight(person2);
  total = person2.addWeight( person1 );
```

```
class A
{
 private:
  int weight;
 public:
  A ( int x = 0 )
  { weight = x; }
  A  addWeight ( A  w2 )
  { A temp;
    temp.weight = weight + w2.weight;
    return  temp; }
}
```

Operator Overloading

VEDINESH

```
#include<iostream>
void main( )
{
  int a = 63, b = 74, c = 0;
  c = a + b ;

  A person1( 63 );
  A person2 ( 74 );
  A total;
  total = person1.addWeight(person2);
  total = person2.addWeight( person1 );
  total = person1 + person2;
}
```

```
class A
{
 private:
  int weight;
 public:
  A ( int x = 0 )
  { weight = x; }
  A  operator +  ( A  w2 )
   { A temp;
     temp.weight = weight + w2.weight;
     return  temp; }
}
```

Operator Overloading

VEDINESH

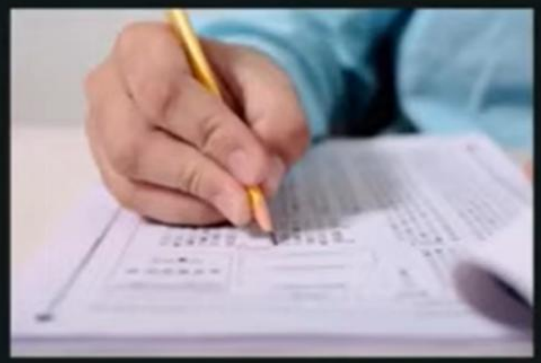
Write a program to overload ++ Pre increment operator

```
#include<iostream>
void main( )
{
  int a = 63;
  ++a;

  Person Jon( 63 );
  ++Jon ;              ( wrong )
}
```

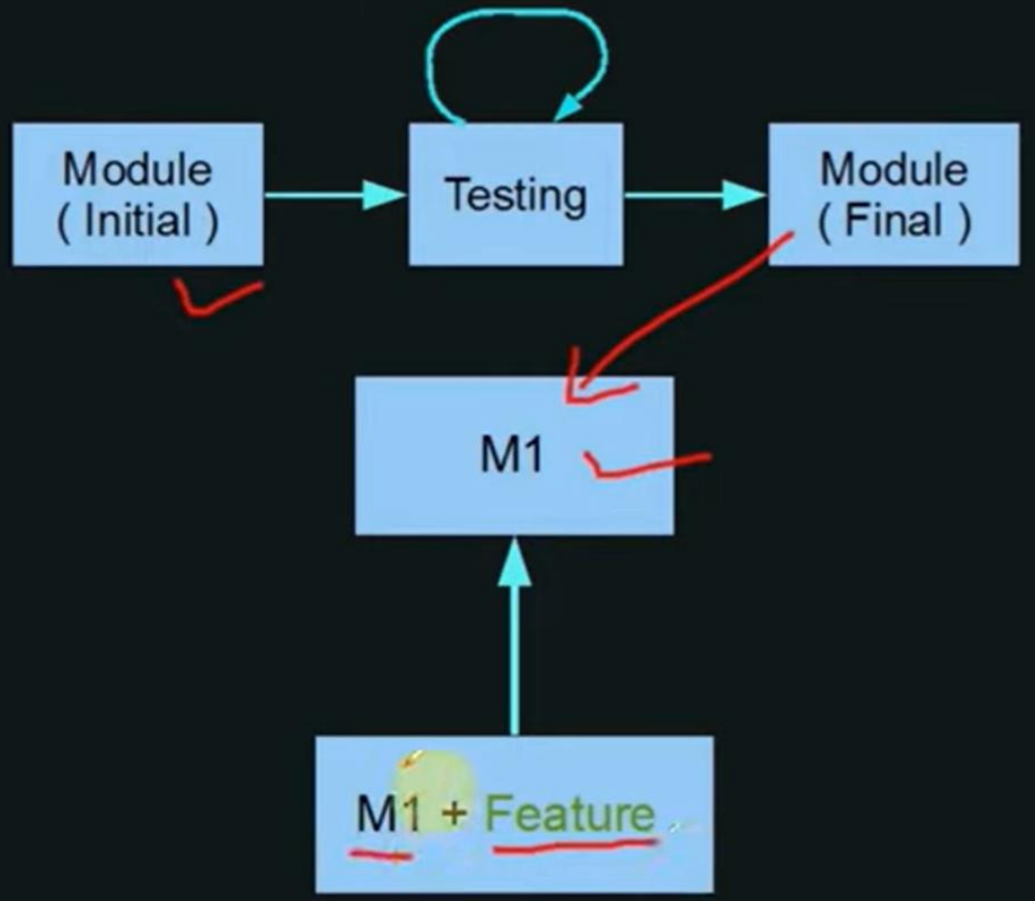
Why Inheritance ?

VEDINESH

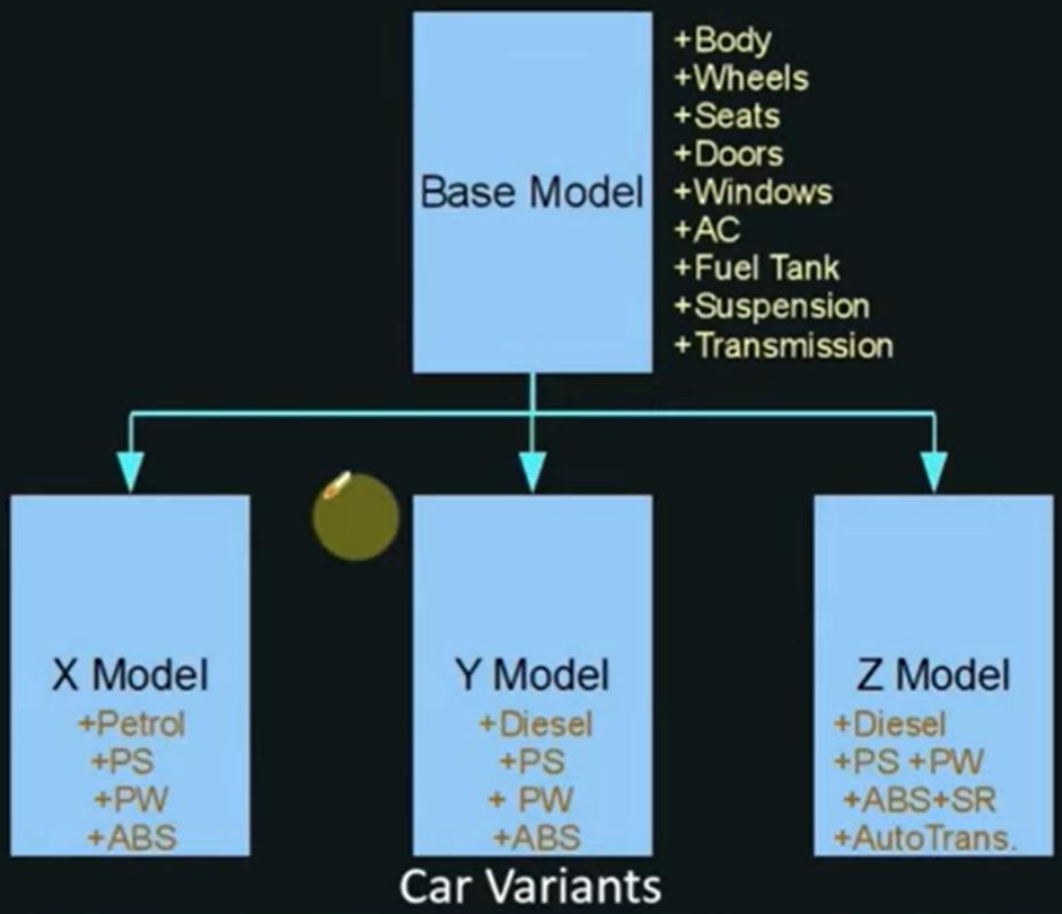
Test / Exam



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



Inheritance Approach



Inheritance Example

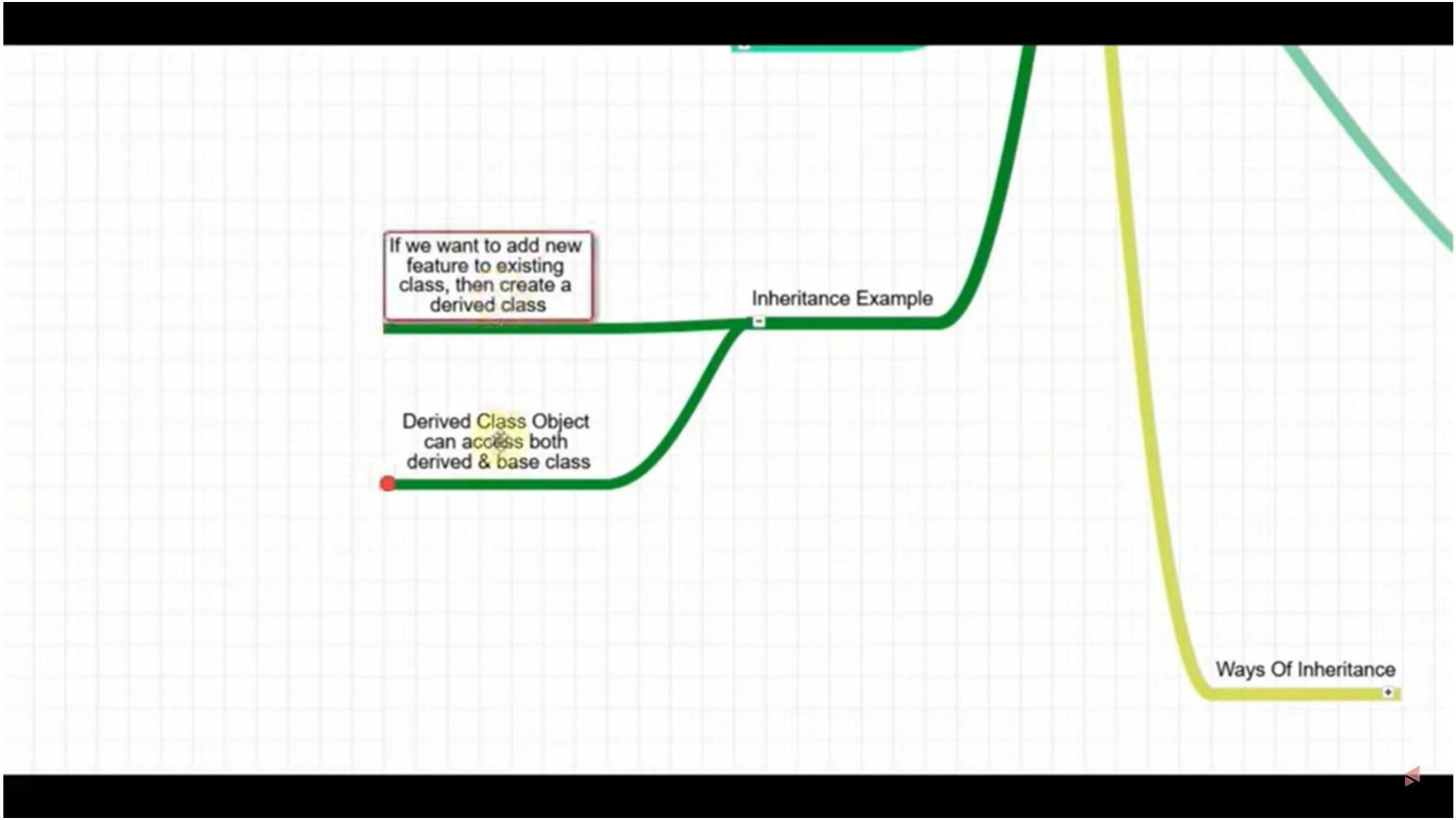
VEDINESH

```
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 ( );
}
```


If we want to add new feature to existing class, then create a derived class
Inheritance Example
Derived Class Object can access both derived & base class
Ways Of Inheritance

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 );
}
```

Constructors & Inheritance

VEDINESH

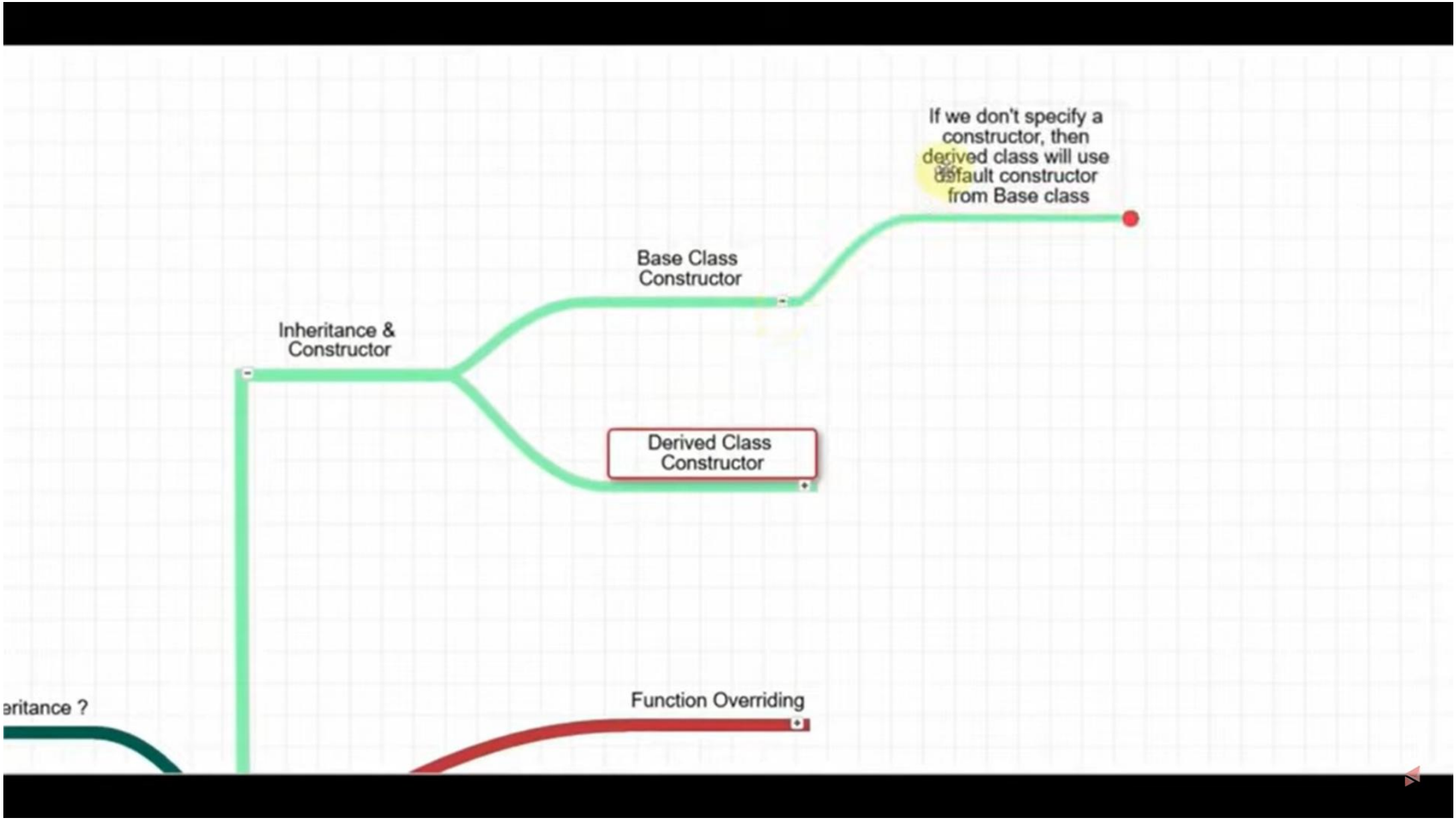
```
class base
{
public:
  base( )
  { cout << "Default Of Base Class"; }

  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.


If we don't specify a constructor, then derived class will use default constructor from Base class
Base Class Constructor
Inheritance & Constructor
Derived Class Constructor
eritance ?
Function Overriding

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:-

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 ( )
{
   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";
  }
};
```

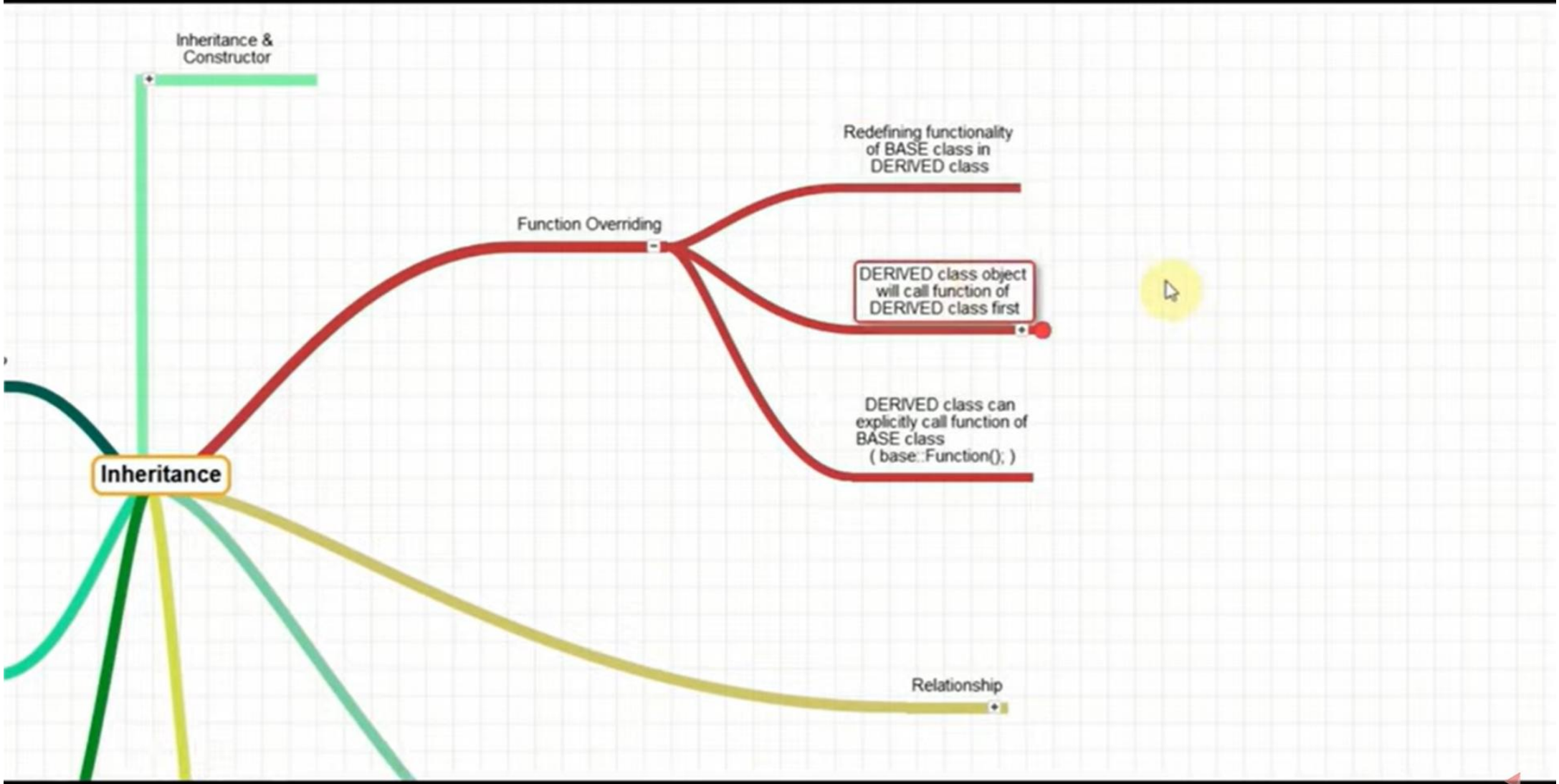
```
void main ( )
{
   derived c;
   c.Msg( );
}
```

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( );
}
```


Inheritance & Constructor
Redefining functionality of BASE class in DERIVED class
Function Overriding
DERIVED class object will call function of DERIVED class first
DERIVED class can explicitly call function of BASE class (base::Function();)
Inheritance
Relationship

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

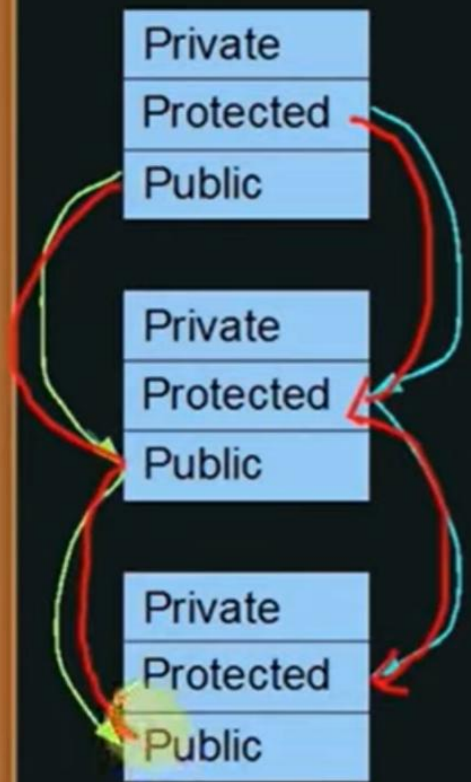
isA vs hasA

VEDINESH

isA	hasA
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.

Ways Of Inheritance

VEDINESH



class Parent

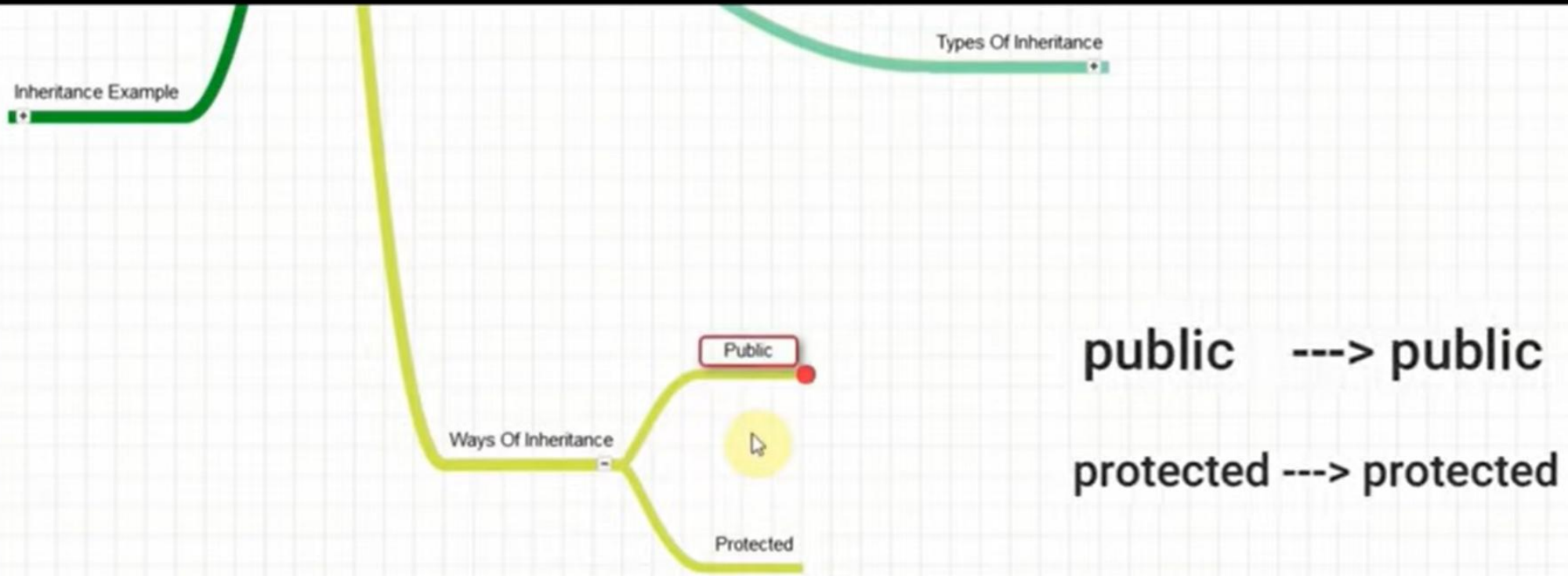
Child AND grandchild would not be able to access both Private.

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

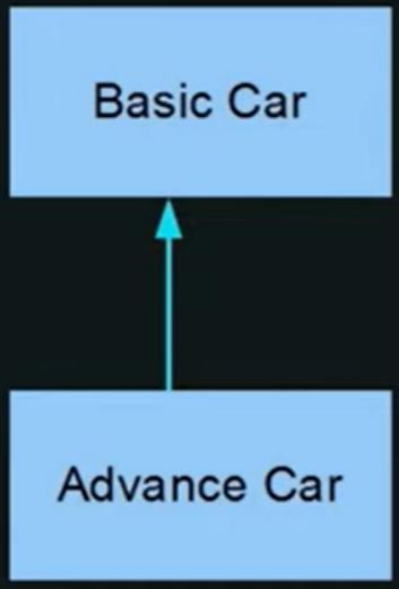


public ---> public

protected ---> protected

Base Class Pointer And Derived Class Object?

VEDINESH



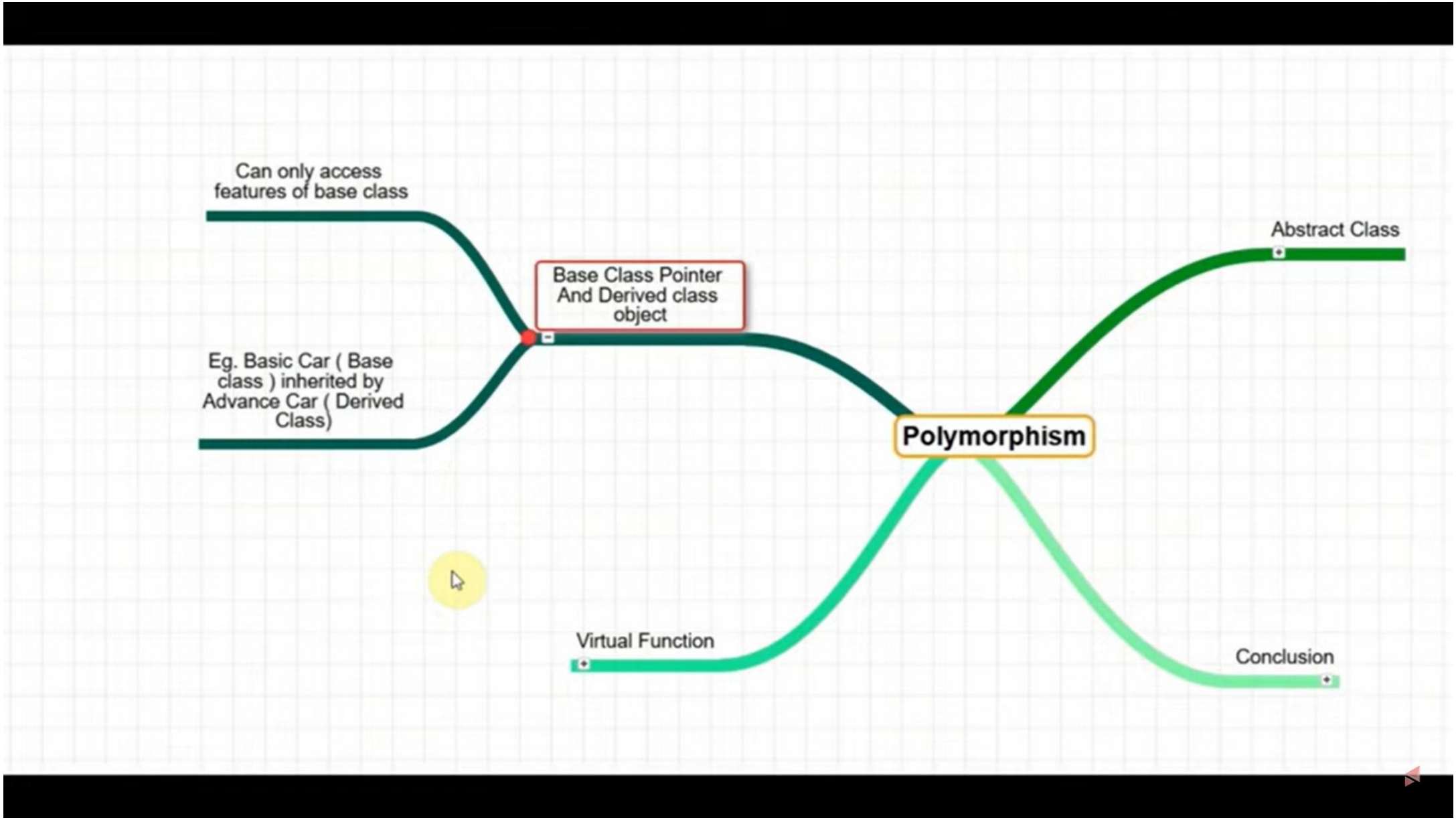

```
class basicCar
{
 public:
        void  body( );
        void  door( );
        void  windows( );
        void  tyres( );
}
```

```
class advCar: public basicCar
{
 public:
        void  ABS( );
        void  PS( );
        void  EngineV8( );
        void  AT( );
}
```

```
void main( )
{
  basicCar *ptr;
  ptr = new advCar( );
```

```
  ptr->body( );
  ptr->doors( );
  ptr->windows( );
  ptr->types( );
```

```
  ptr->ABS( );
  ptr->PS( );
  ptr->EngineV8( );
  ptr->AT( );
```


Can only access features of base class
Base Class Pointer And Derived class object
Eg. Basic Car (Base class) inherited by Advance Car (Derived Class)
Polymorphism
Abstract Class
Virtual Function
Conclusion

Virtual Function

VEDINESH

Virtual means existing in appearance but not in reality

Virtual Function means fun. existing in class but can't be used.

Program that appears to be calling a function of one class may in reality be calling a function of different class .

Virtual Function

VEDINESH

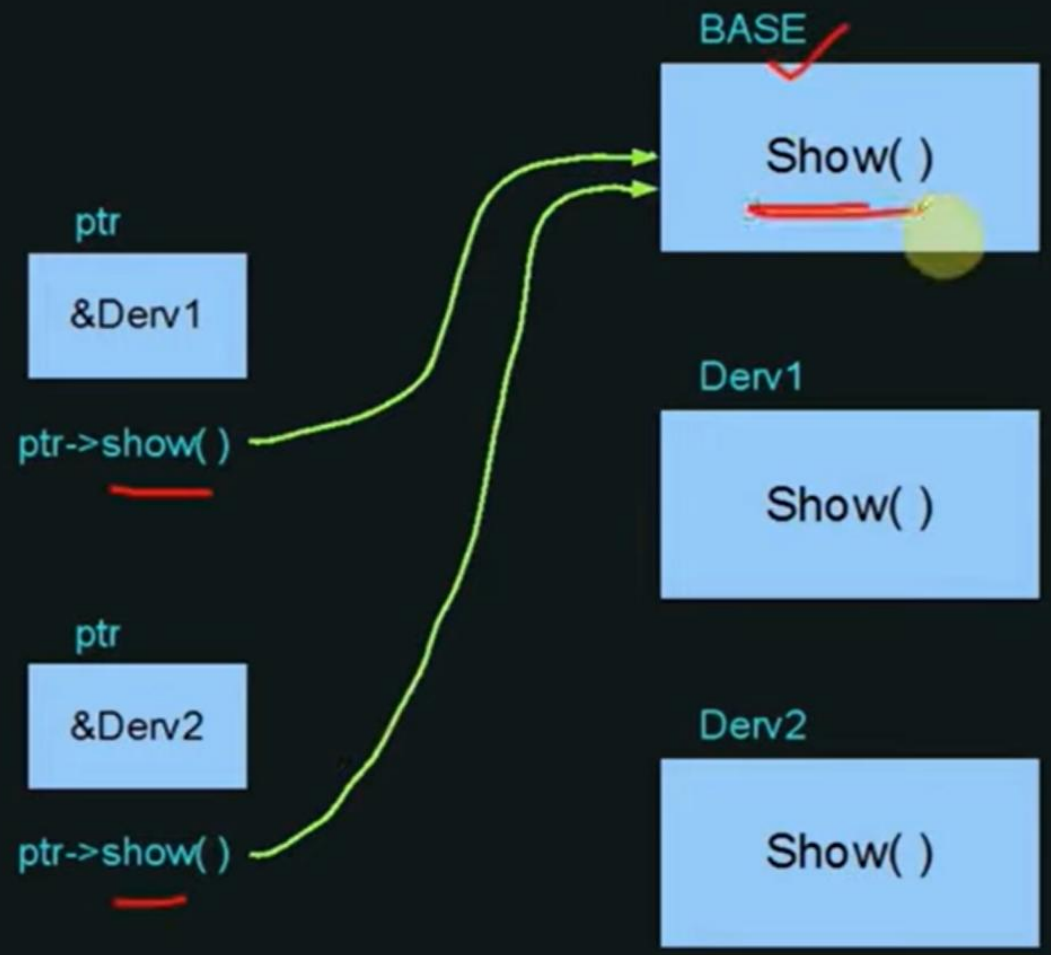
```
class Derv1: public base
{
 public:
        void  show( )
        {
           cout << "Derived1";
        }
}
```

```
class Derv2: public base
{
 public:
        void  show( )
        {
           cout << "Derived2";
        }
}
```

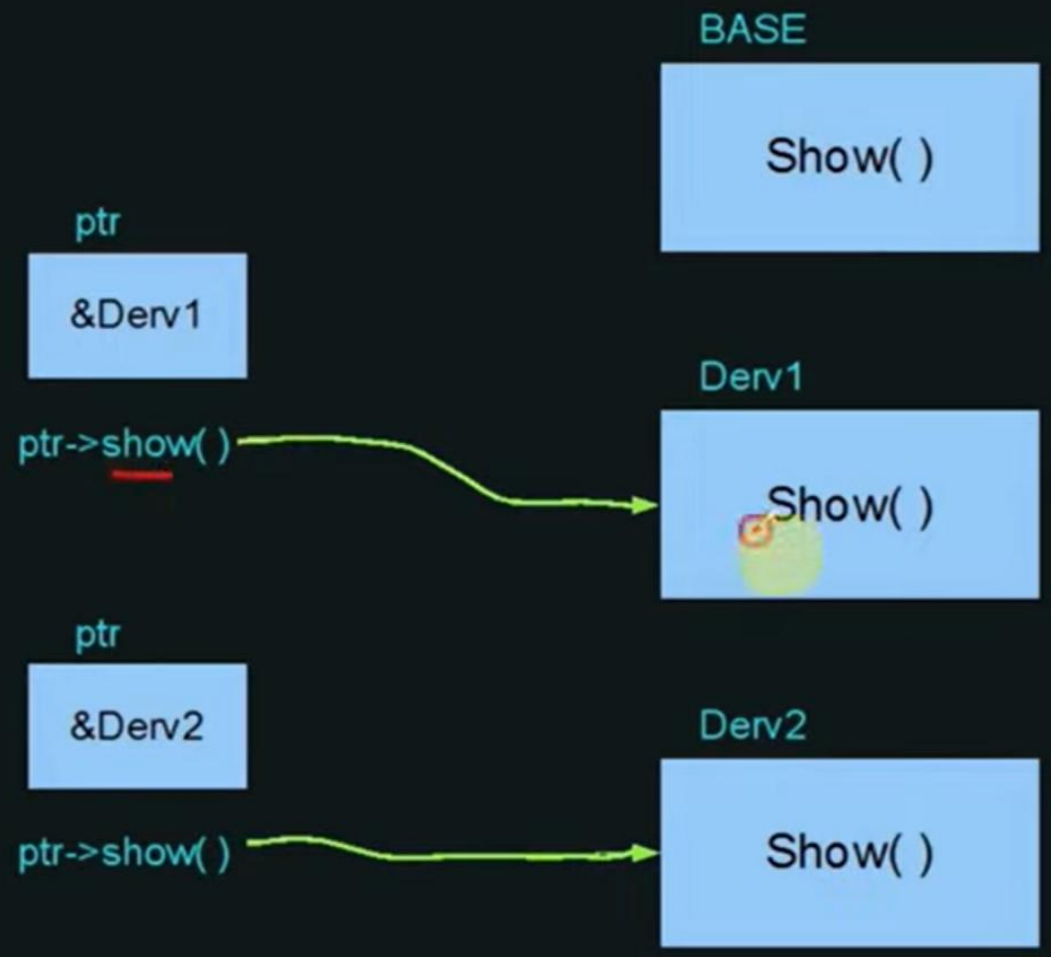
```
class base
{
 public:
        void  show( )
        {
           cout << "Base";
        }
}
```

```
void main( )
{
  Derv1  dv1;
  Derv2  dv2;

  base *ptr;
  ptr = &dv1;
  ptr->show( );

  ptr = &dv2;
  ptr->show( );
}
```

Reason


BASE
Show( )
ptr
&Derv1
ptr->show( )
Derv1
Show( )
ptr
&Derv2
ptr->show( )
Derv2
Show( )

Reason


BASE
Show( )
ptr
&Derv1
Derv1
ptr->show( )
Show( )
ptr
&Derv2
Derv2
ptr->show( )
Show( )

Why Virtual Function ?

Late Binding:-

Compiler deferr the decision untill, the program is running.
And at runtime when it come to know which class is pointed by PTR, then appropriate function would be called.
This is called Dynamic Binding / Late Binding

Virtual Function

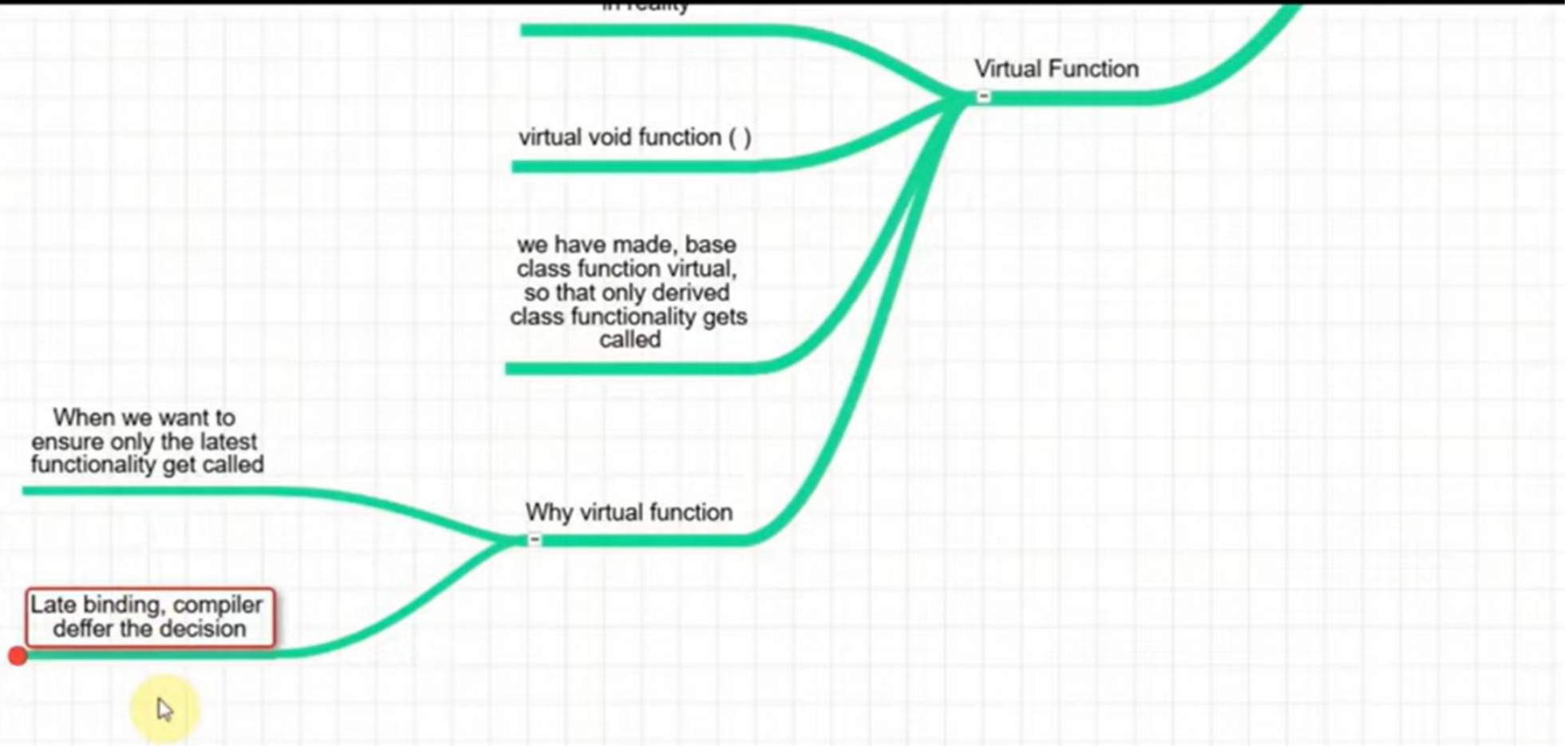
VEDINESH

```
class boy: public person
{
 public:
        void  give( )
        {
           cout << "Brown Bun";
        }
}
```

```
class girl: public person
{
 public:
        void  give( )
        {
           cout << "Pink Bun";
        }
}
```

```
class person
{
 public:
        virtual void  give( )
        {
           cout << "Bun";
        }
}
```

```
void main ( )
{
   boy b1;
   girl g1;

   person *ptr = NULL;

   ptr = &b1;
   ptr->give();

    ptr = &g1;
    ptr->give();
}
```


Virtual Function
virtual void function ( )
we have made, base class function virtual, so that only derived class functionality gets called
Why virtual function
When we want to ensure only the latest functionality get called
Late binding, compiler deffer the decision

Abstract Class

VEDINESH

```
class boy: public person
{
 public:
        void  give( )
        {
           cout << "Brown Bun";
        }
}
```

```
class girl: public person
{
 public:
        void  give( )
        {
           cout << "Pink Bun";
        }
}
```

Define:- Abstract class is used when we never want to instantiate object of BASE class.

```
class person
{
 public:
    virtual void  give( )  = 0;
}
```

Pure Virtual Function

Define:- Abstract class exists only to act as parent of DERIVED CLASS.

```
void main ( )
{
   boy b1;
   girl g1;

   person *ptr = NULL;

   ptr = &b1;
   ptr->give();

   ptr = &g1;
   ptr->give();
}
```

Conclusion

VEDINESH

give( )

give( )

ptr = &girl1

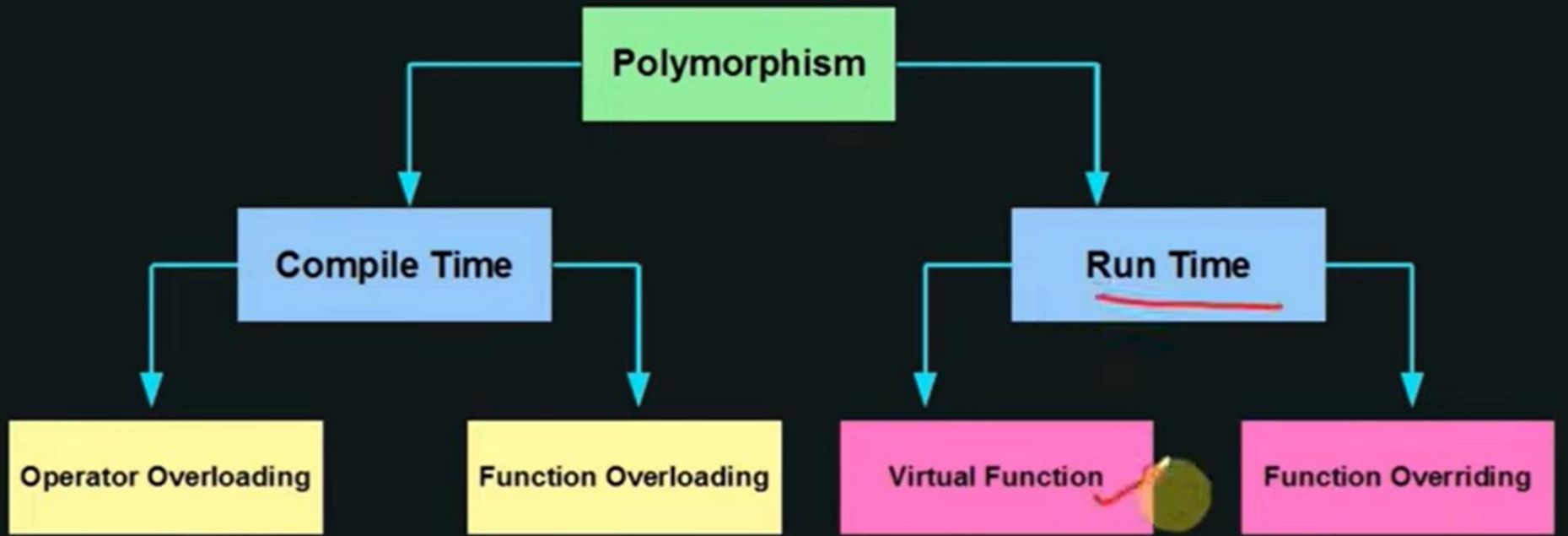
ptr->give( )

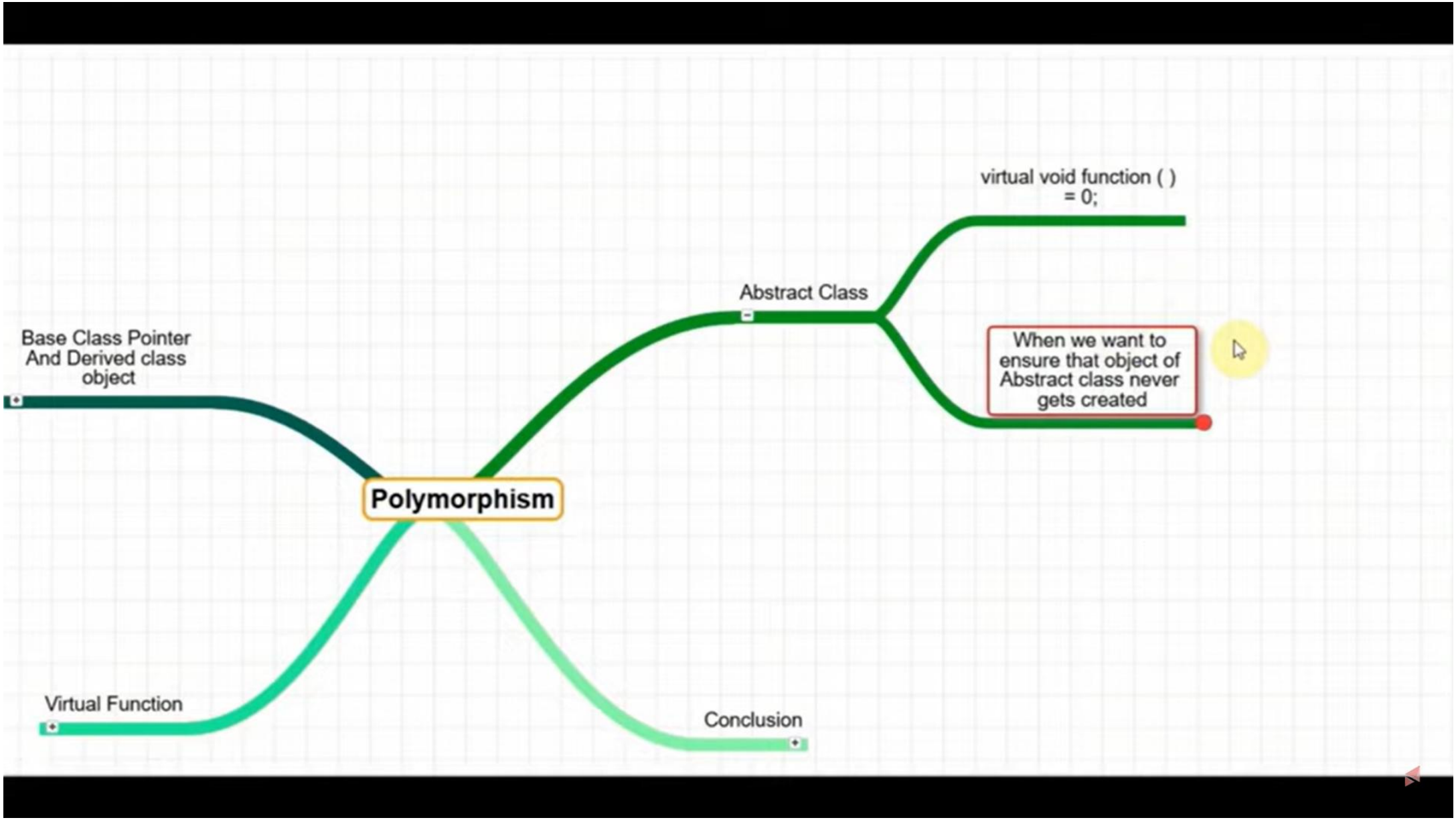
ptr = &boy1

Polymorphism

Poly means many, something existing in more than one form.

Polymorphism




virtual void function ( )
= 0;
Abstract Class
When we want to ensure that object of Abstract class never gets created
Base Class Pointer And Derived class object
Polymorphism
Virtual Function
Conclusion

Virtual Destructor

VEDINESH

```
class base
{
 public:
        ~base( )
        {
           cout << "Base Class Destroyed";
        }
}
```

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

```
void main ( )
{
   base *b1 = new derived;

   delete b1;
}
```

Note:- In this case only base class destructor is called.

Virtual Destructor

VEDINESH

```
class base
{
 public:
        virtual ~base( )
        {
           cout << "Base Class Destroyed";
        }
}
```

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

```
void main ( )
{
   base *b1 =  new derived;

   delete b1;
}
```

Now

Static Members

VEDINESH

```
class Alpha
{
 private:
            int a;
            int b;
 public:
            Alpha( )
            {
              a = 5;
              b = 5;
            }
};
```

```
void main( )
{
   Alpha a1;
   Alpha a2;
}
```

NOTE:- Each Object will create separate copy of itself in memory .

a1

5
5

a2

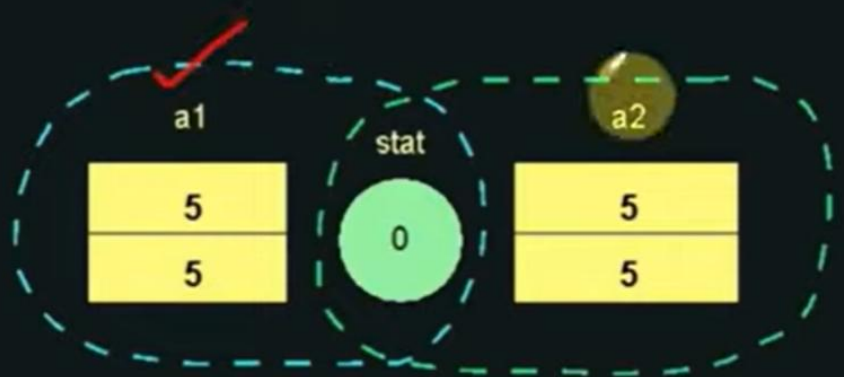
5
5

Static Member

VEDINESH

```
class Alpha
{
 private:
            int a;
            int b;
 public:
            Alpha( )
            {
              a = 5;
              b = 5;
              stat ++;
            }

            static int stat;
};
```

```
void main( )
{
   Alpha a1;
   Alpha a2;
```



Static Member

VEDINESH

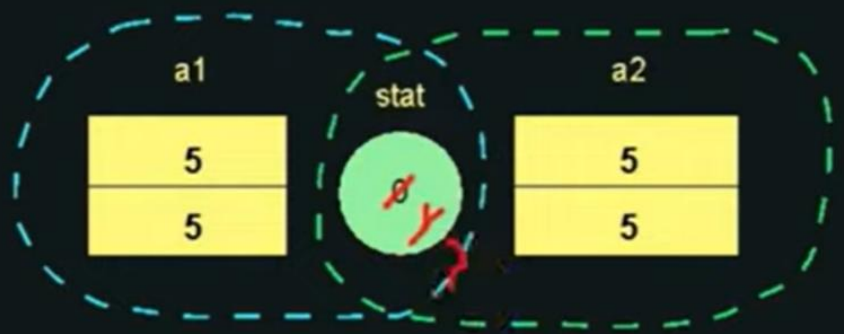
```
class Alpha
{
 private:
          int a;
          int b;
 public:
          Alpha( )
          {
            a = 5;
            b = 5;
            stat ++;
          }

          static int stat;
};

int  Alpha ::  stat = 0;
```

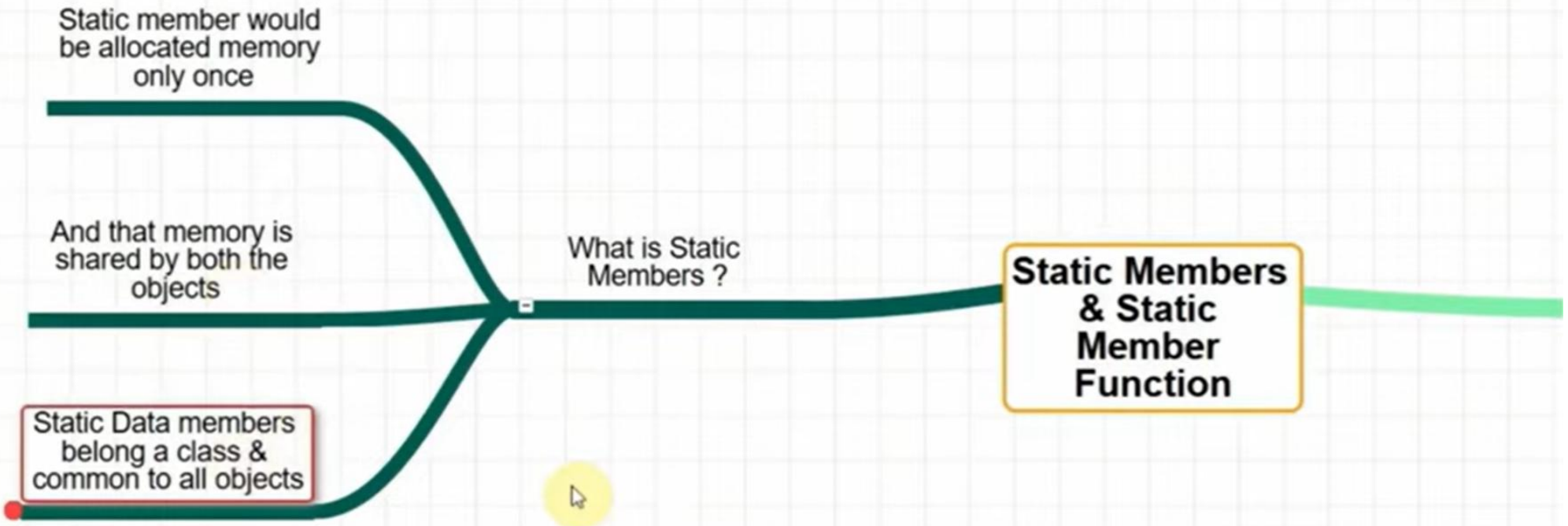
```
void main( )
{
   Alpha a1;
   Alpha a2;

   cout <<  a1.stat;
```

NOTE:- Static member would be allocated memory only once .

NOTE:- And that memory is shared by both the objects.

NOTE:- Static Data members belong a class & common to all objects.



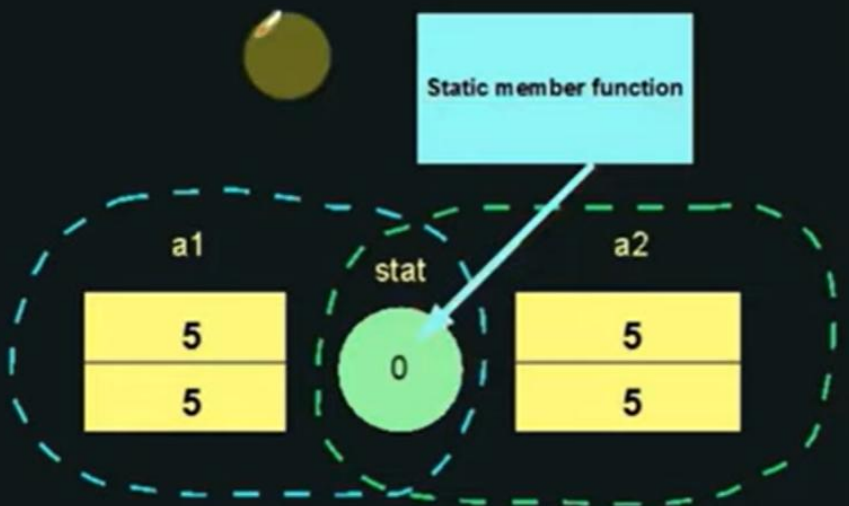

Static member would be allocated memory only once
And that memory is shared by both the objects
Static Data members belong a class & common to all objects
What is Static Members ?
Static Members & Static Member Function

Static Member Function

VEDINESH

```
class Alpha
{ private:
        int a;
        int b;
  public:
        Alpha( )
        {  a = 5;
           b = 5;
        }

        static int stat;

        static int getStat( )
        {  stat ++ ;
           return stat ; }
};

int  Alpha ::  stat = 0;
```

NOTE:- Static member can only access static members .



Static Member Function

VEDINESH

```
class Alpha
{ private:
        int a;
        int b;
  public:
        Alpha( )
        { a = 5;
          b = 5;
        }

        static int stat;

        static int getStat( )
        { stat ++ ;
          return stat ; }
};

int Alpha :: stat = 0;
```

```
void main( )
{
   cout << Alpha :: getStat( );
   Alpha a1;
   Alpha a2;
   cout << Alpha :: getStat( );
   cout << a1.getStat( );
   cout << a2.getStat( );
}
```

NOTE:- Static member can only access static members .