Class (OOP) - 2

Inheritance

```
class Base;
int b;

function int get ();
return b;
endfunction
endclass

class Derived extends Base;
int c;

function void print();
$display("b=%d c=%d ",b,c);
endfunction

endclass
```

Base:
int b
function get

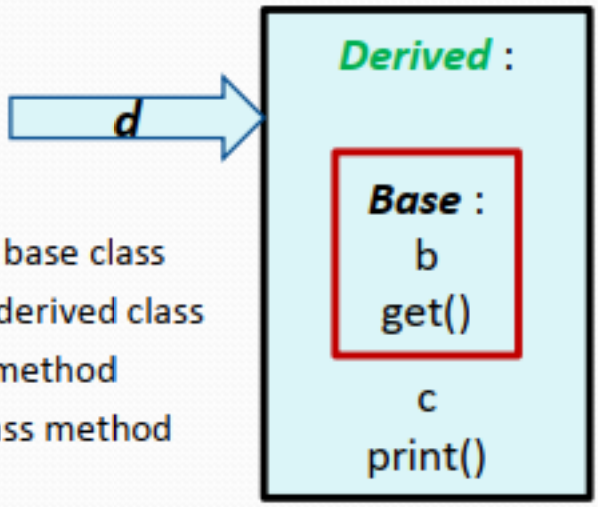
Derived:
int b
function get

int c
function print

```
class Derived;
int b;
int c;

function int get();
return b;
endfunction

function void print();
$display("b=%d c=%d",b,c);
endfunction

endclass
```

Inheritance

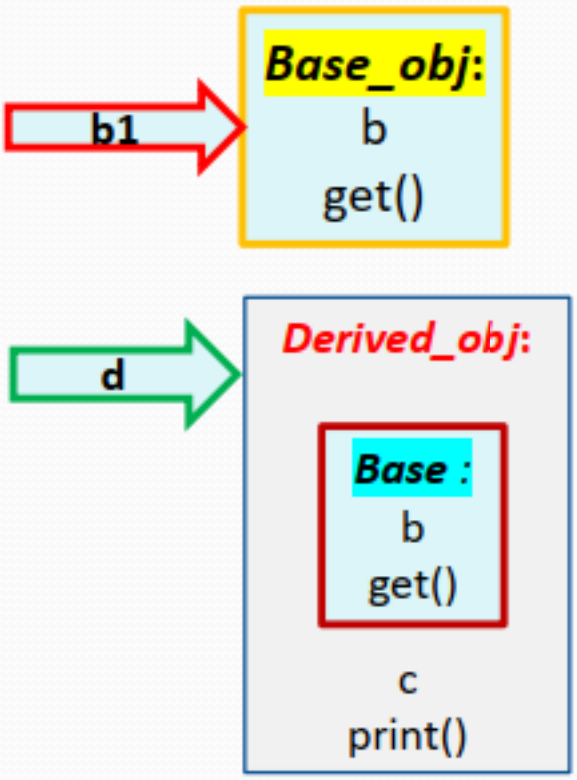
```
program test;
Derived d;

initial begin
d=new;
d.b=10;  //b is part of base class
d.c=20;  //c is part of derived class
d.get;    //base class method
d.print;  //derived class method
end
endprogram
```




```
class Base;
int b;

function int get ();
return b;
endfunction
endclass

class Derived extends Base;
int c;

function void print();
$display("b=%d c=%d ",b,c);
endfunction

endclass
```

Inheritance

```
program test;
Derived d;Base b1;
int ret1,ret2;

initial begin
b1=new;
d=new;
b1.b=40;
d.b=10;
d.c=20;
ret2=b1.get();
ret1=d.get();
d.print();
end
endprogram
```

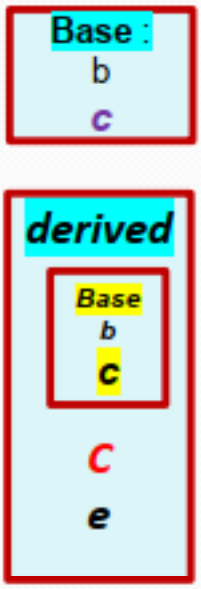



```
class Base;
int b;

function int get ();
return b;
endfunction
endclass

class Derived extends
Base;
int c;

function void print();
$display("b=%d c=%d ",b,c);
endfunction

endclass
```

Super

- The **super** keyword is used from within a derived class to refer to members of the base class.

```
class base;
int b,c;
endclass

class derived extends base;
int c,e;

function void print();
c=10; b=30;e=40;
super.c=20;
endfunction

endclass
```




```
program test;
derived d;
base b;
initial begin
b=new;
b.c=10;
d=new;
d.c=20;
end
endrpgoram
```

Constructors in Extended Classes

```
class base;
int m;
    function new(int f=5);
        m = f;
    endfunction
endclass

class Derived extends base;
int k;
endclass
```

```
d1=new;

base=new();

derived=new();
```

```
program test;
Derived d1;

initial begin
d1=new;
end
```

```
** Error: constructor_derived_ISSUE.sv(14):
Super class constructor has non-default arguments.

Arguments can be passed by calling super.new() explicitly.
```

Constructors in Extended Classes

```
class base;
int m;
        function new(int f);
        m=f;
        endfunction
endclass

class Derived extends base;
int k;
 function new (int d);
 super.new(d);
 endfunction
endclass
```

```
program test;
Derived d1;

initial begin
d1=new(30);
end

endprogram
```

super.new call ***must be the first statement*** executed in the constructor.
This is because the super (base) class shall be initialized before the current (derived) class

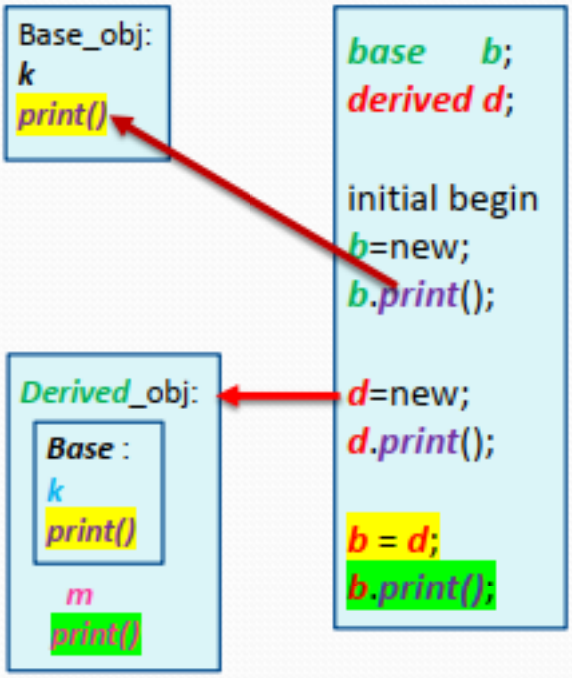
Overridden members

- **What happens when you call a method that exists in both the base and derived classes?**
  - **SystemVerilog calls the method based on handle type.**
  - **If the handle is of type base class, then base class method will be called.**
  - **If the handle is of type derived class, then derived class method will be called.**

```
class base;
int k;

function void print();
$display("[base] k=%d ",k);
endfunction

endclass

class derived extends base;
int m;

function void print();
$display("[derived] m=%d
k=%d",m,k);
endfunction

endclass
```

Overridden members

```
class base;
int k;

function void print();
$display("[base] k=%0d ",k);
endfunction
endclass

class derived extends base;
int m;

function void print();
$display("[derived] m=%0d
k=%0d",m,k);
endfunction
endclass
```



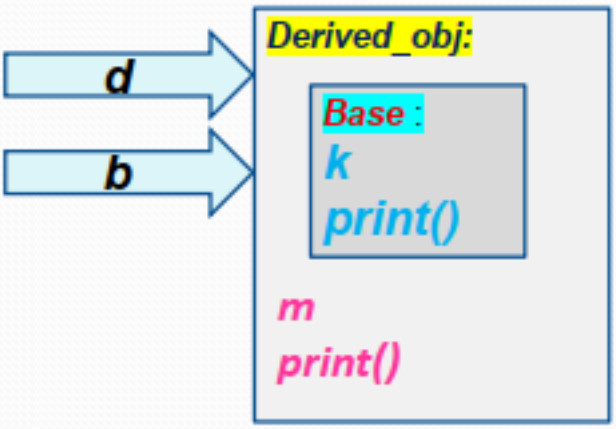
Handle assignments

```
class base;
int k;
function void print();
$display("[base] k=%0d ",k);
endfunction
endclass

class derived extends base;
int m;
function void print();
$display("[derived] m=%0d k=%0d",m,k);
endfunction
endclass
```

Handle assignments

- *Is it legal to assign derived class handle to base handle (b=d)?*

```
base b;
derived  d;

initial begin
d=new;
b = d ; //Is this LEGAL ?
end
```



Ans: Yes , *this is legal .*

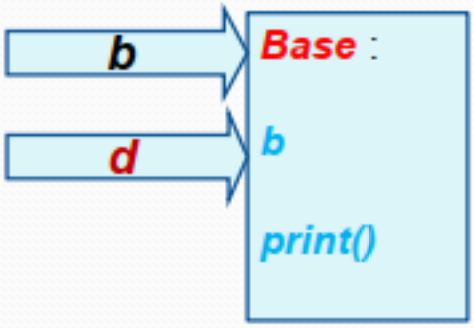
Derived class *handle is* ***assignment compatible*** *with* ***base class*** *handle.*

It is possible because derived class has all the properties of base class

Handle assignments

- Is it legal to assign base class handle to derived handle (d=b)?

```
base b;
derived  d;

initial begin
b=new;
d = b ; //Is this LEGAL ?
end
```



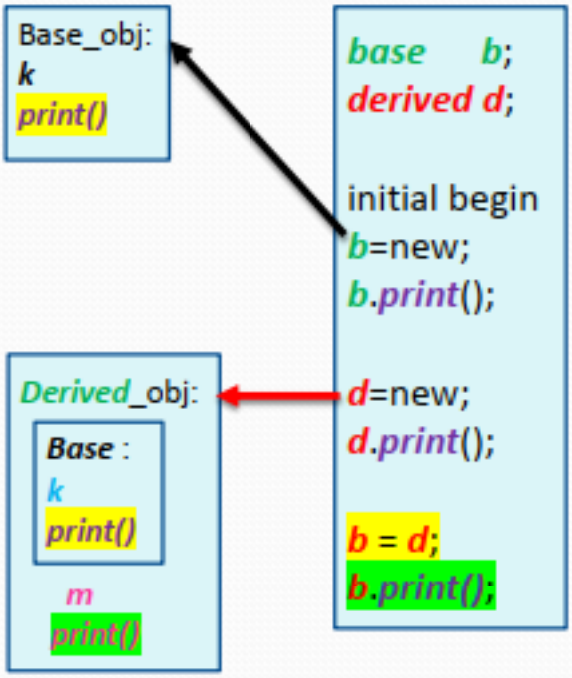
Ans: No .

This is not legal as "*base class* handle is *not assignment compatible* with *derived class handle*".

It is not possible because base class is missing extra members which exists only in derived class .

Polymorphism

```
class base;
int k;

virtual function void print();
$display("[base] k=%0d ",k);
endfunction
endclass

class derived extends base;
int m;

virtual function void print();
$display("[derived] m=%0d
k=%0d",m,k);
endfunction
endclass
```



Polymorphism steps

Step 1: ***Make the method as*** virtual ***in base*** and derived.
(Qualifying the method as virtual in the derived is optional).

Step 2 : ***Assign derived class handle to base class handle***

b = d;

Step 3: ***Call the virtual method with base class handle***.

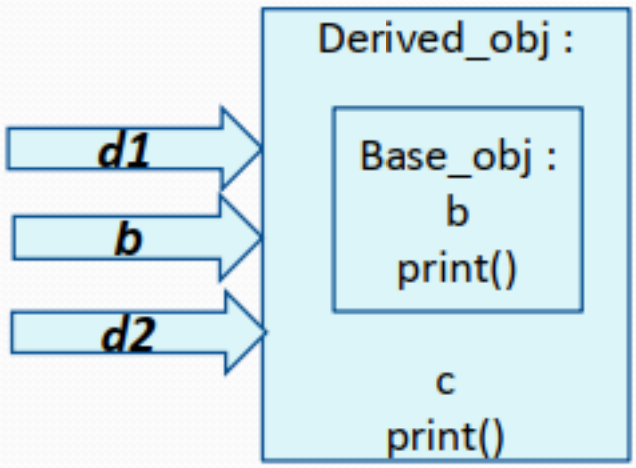
NOTE:

- ***Base class handle has no access to derived class methods*** (***methods which exists only in derived class***) even after handle assignment.
- ***With base class handle we cannot call methods which exists only in derived class.***

$cast(destination,source)

- *Is it legal to assign base class handle to derived handle (d=b)?*

```
base b;
derived  d1,d2;

initial begin
d1=new;
b=d1;   //Is this LEGAL ? Yes
d2=b ; //Is this LEGAL ? No
end
```



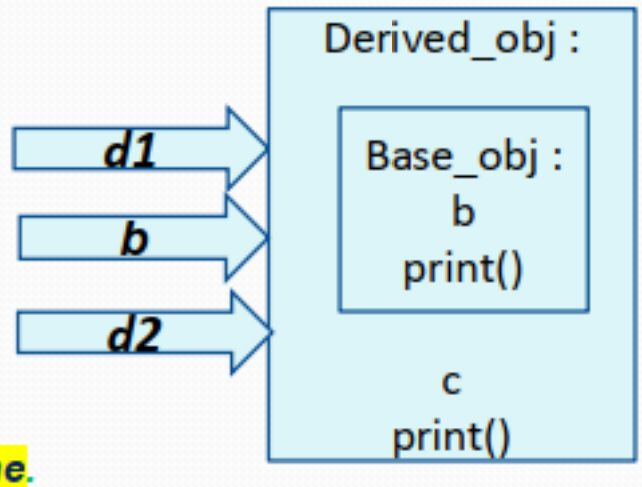
Ans: No . Error at compile time.

This is not legal as "Base class handle is not assignment compatible with derived class handle".

$cast(destination,source)

- *Is it legal to assign base class handle to derived handle (d=b)?*

```
base b;
derived  d1,d2;

initial begin
d1=new;
b=d1;   //Is this LEGAL ? Yes
$cast(d2,b);
end
```



Ans: Yes . No error at compile time.

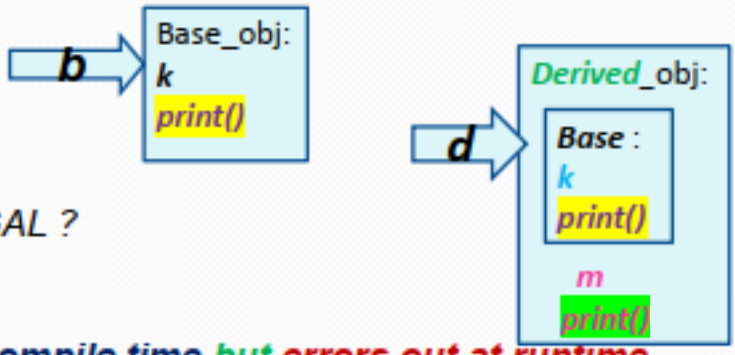
$cast checks the ***assignment compatible check*** **at runtime**.

At runtime handle b is already pointing to derived class hence $cast(d2,b) will succeed.

$cast(destination,source)

- Is it legal to assign base class handle to derived handle (d=b)?

```
base b;
derived  d;

initial begin
d=new;
b=new;
$cast(d,b); //Is this LEGAL ?
end
```



Ans: No , No error at compile time but errors out at runtime.

$cast checks the **assignment compatible check** at runtime.

At runtime handle b is not pointing to derived class hence $cast(d,b) will fail at runtime. We will see runtime error.

```
Error- Dynamic cast failed
cast_example.sv, 8
Casting of source class type 'base' to destination class type 'derived'
failed due to type mismatch.
Please ensure matching types for dynamic cast
```

Parameterized classes

```
class Vector #(int size = 16);
bit [size-1:0]  a;


function void disp();
$display( "size %d", size );
endfunction

function logic [size-1:0] add (logic[size-1:0] in1,in2);
return (in1+in2);
endfunction

endclass
```

```
Vector        v16;
Vector #(8)   v8;
Vector #(.size(2))  v2;

initial begin
v16=new;
v16.a=16'habcd;

v8=new;
v8.a=8'hab;

v2=new;
v2.a=2'h1;

end
```

Parameterized classes

```
class Stack #(type T=int);

local  T   stack[100];
local int top;

function void push(input  T  i);
stack[++top] = i;
endfunction : push

function  T   pop();
return stack[top--];
endfunction

endclass : Stack
```

```
Stack #(real) rstack;
Stack  istack;

initial begin
rstack=new();
rstack.push(13.67);

istack=new();
istack.push(44);
end
```

Data hiding and encapsulation

- In SystemVerilog, ***unqualified class properties and methods are public***, available to anyone who has access to the object's name.

- Often, it is desirable to ***restrict access to class properties and methods from outside the class*** by hiding their names.

Access types:

1. Public (default)
2. Local
3. Protected

Local,protected,public

- local : A member identified as **local** is available only to methods inside the class.
- Further, these local members are not visible within derived classes.

- Protected: A **protected** class property or method has all of the characteristics of a **local** member, except that it can be inherited; it is visible to derived classes.

- Public : Unqualified class properties and methods are public, available to anyone who has access to the object's name

```
class base;
local      int x;   //accessible only within this class
protected  int y;   //can be accessed  in derived class
           int z;  //public can be accessed anywhere

function void run();
x=10; $display("x=%0d ",x,y);
endfunction

endclass

class derived extends base;
function void print();
$display(" y=%0d z=%0d",y,z);
endfunction

endclass
```

```
base b;
derived d;

initial begin
b.z=10;
b.x=10;   <-- ERROR
b.y=10;
d.y=10;
d.x=20;   <-- ERROR
end
```