Nested Class

- A class can contain instance of another class using handle to an object. Such classes are called as Nested Classes.
- Common reasons for using containment are reuse and controlling complexity.

```
Class pack_node;
//properties and methods for
   pack_Node
endclass
```

```
Class node;
pack_node p1,p2;
//properties and methods for
   Node
endclass
```

Example

```
class timestat;
time start_time, end_time;

function void start;
start_time=$time;
endfunction

function void end;
end_time=$time;
endfunction
endclass
```

Example

```
class packet;
int data[7:0];
timestat t;

function new;
 t=new;
endfunction

extern task transmit;

endclass
```

```
task packet :: transmit();
t.start;
//do some operation
t.end;
endtask
```

Typedef Class

- A forward declaration is a declaration of a object which the programmer has not yet given a complete definition.
- System Verilog language supports the typedef class construct for forward referencing of a class declaration.
- This allows for the compiler to read a file from beginning to end without concern for the positioning of the class declaration.

Example

```
module test;
class packet;
timestat t;
//definitions
endclass

class timestat;
//definitions
enclass
endmodule
```

Compilation error class timestat is not defined.

Timestat is referred before it is defined

Example

```
module test;
typedef class timestat;
class packet;
timestat t;
//definitions
endclass

class timestat;
//definitions
enclass
endmodule
```

typedef allows compiler to process packet class before timestat class.

Copy

- User can make a copy of an object to keep a routine from modifying the original.

- There are two ways of copying an object:
  - Using built-in copy with new function (Shallow Copy)
  - Writing your own complex copy function (Deep Copy)

- Using new to copy an object is easy and reliable. A new object is constructed and all variables from the existing object are copied.

Shallow Copy

```
class pkt;
bit addr [15:0];
bit [7:0] data;
int status;

function new();
addr=$randomize;
data=$randomize;
status=0;
endfunction
endclass
```

```
pkt src, dst;
initial begin
src=new;          //create object
dst=new src;      //copy to dst
end
```

src	dst
addr=5 ;	addr=5 ;
data=10;	data=10;
status=0;	status=0;

Shallow Copy

- Shallow copy is similar to photocopy, blindly copying values from source to destination.
- If a class contains handle to another class then only top level objects are copied by new, not the lower one.
- When using new to copy objects, the user define new constructer is not called. New function just copies the value of variables and object handle.

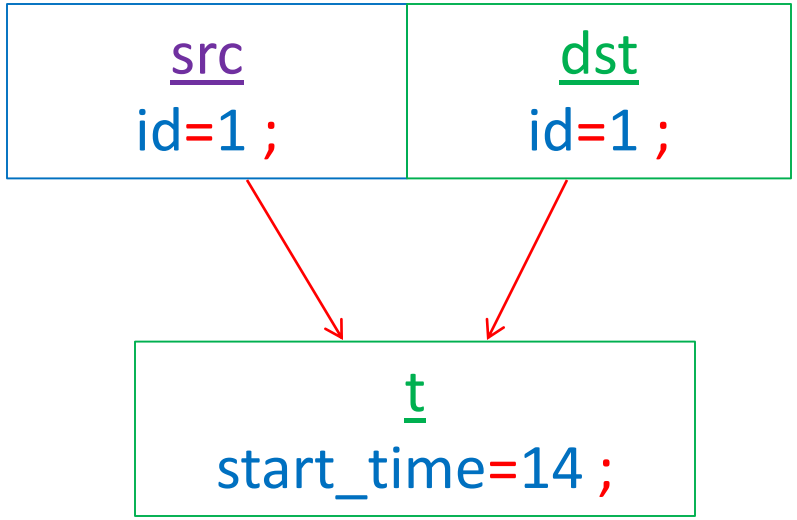
Example

```
class timestat;
time start_time, end_time;

endclass
```

```
class pkt;
bit addr [15:0];
bit [7:0] data;
int id; static int count;

timestat t;

function new();
id=count++;
t=new;
endfunction
endclass
```

Example

```
packet src, dst;
initial begin
src=new;
src.t.start_time=10;

dst=new src;
//handle of t is copied
//id is not incremented

dst.t.start_time=14;
//modifies t since
// handler is common
end
```



Deep Copy

- User can write his own deep copy function.
- This user defined copy function should copy the content of class handle, not handle itself.

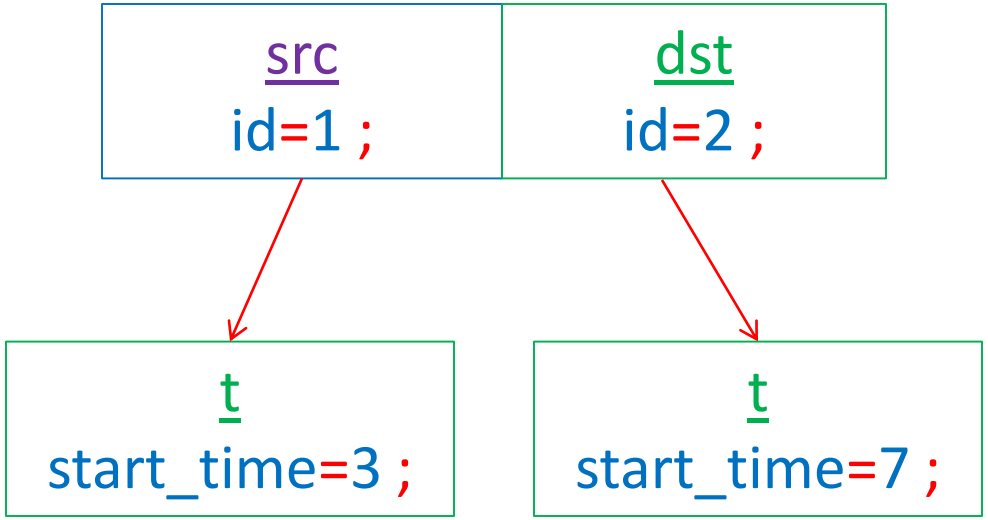
Example

```
class pkt;
bit addr [15:0];
bit [7:0] data;
int id; static int count;

timestat t;

function new();
id=count++;
t=new;
endfunction
extern function pkt copy;
endclass
```

```
function pkt pkt :: copy;
copy=new;
copy.addr=this.addr;
copy.data=this.data;
copy.t.start_time=this.t.start_time;
copy.t.end_time=this.t.end_time;
endfunction
```

Example

```
initial begin
pkt src, dst;
src=new;
src.t.start_time=3;
dst=src.copy;
dst.t.start_time=7;
end
```



Interface Class

- A set of classes can be created that have a common set of behaviors. This set is called Interface class.
- An interface class can only contain pure virtual functions, type declaration and Parameter declarations.
- Pure virtual functions are function that don’t have any implementation.
- implements keyword is used to define a class that implements function defined in interface class.
- When interface class is implemented then nothing is extended, implementation of pure virtual function is defined in class that implements interface class.

Interface Class

```
interface class shape #(type id=int);
int a;
pure virtual function id area(id x=0, y=0);
pure virtual function id perimeter(id x=0, y=0);
endclass
```

Interface Class

```
class int_rectangle implements shape #(int);

virtual function int area(int x=0, y=0);    //virtual keyword
return x*y;                                  //compulsory
endfunction

virtual function int perimeter(int x=0, y=0);
return 2*(x+y);
endfunction
endclass
```

Interface Class

```
class real_rectangle implements shape #(real);

virtual function real area(real x=0, y=0);
return x*y;
endfunction

virtual function real perimeter(real x=0, y=0);
return 2*(x+y);
endfunction

endclass
```

Singleton Class

- These are classes that restricts instantiation of class to just one object.

```
class singleton;
int a;
static singleton obj;

local function new (int a);
this.a=a;
endfunction

//static function

endclass
```

```
static function singleton create(int a);
if (obj==null)
obj=new(a);
return obj;
endfunction

 initial begin
 singleton s1;
 s1=singleton::create();
 end
```

Assignment-11

- **Create a class transaction with following properties:**
  - **Paddr: 4 bit, Pwdata: 32 bit, Psel: 2 bit, Pen:1 bit**
  - **Create another class generator and use the concept of nested class and randomize all the properties of transaction class.**

- **Create a interface class func_lib with following methods:**
  - **Randomize method 4 int type data;**
  - **Print method to display**
  - **Counter method with range**
  - **Implement all the methods in another class chk_intf**

In problem 2, refer generator class and assign all properties to a new class properties named driver with following methods:

Randomize method

Print method to display

Reading Assignments

- **How to transfer data from one class to another class**
- **How to transfer data from class to module**
- **Application of interface classes**