Classes

Class CAR

Task/Function

-> Start
-> Go Forward
-> Go Reverse
-> Stop

DATA

Accelerator | Brake | Clutch | Engine

HONDA | BMW | Audi | MARUTI SUZUKI

Class Object

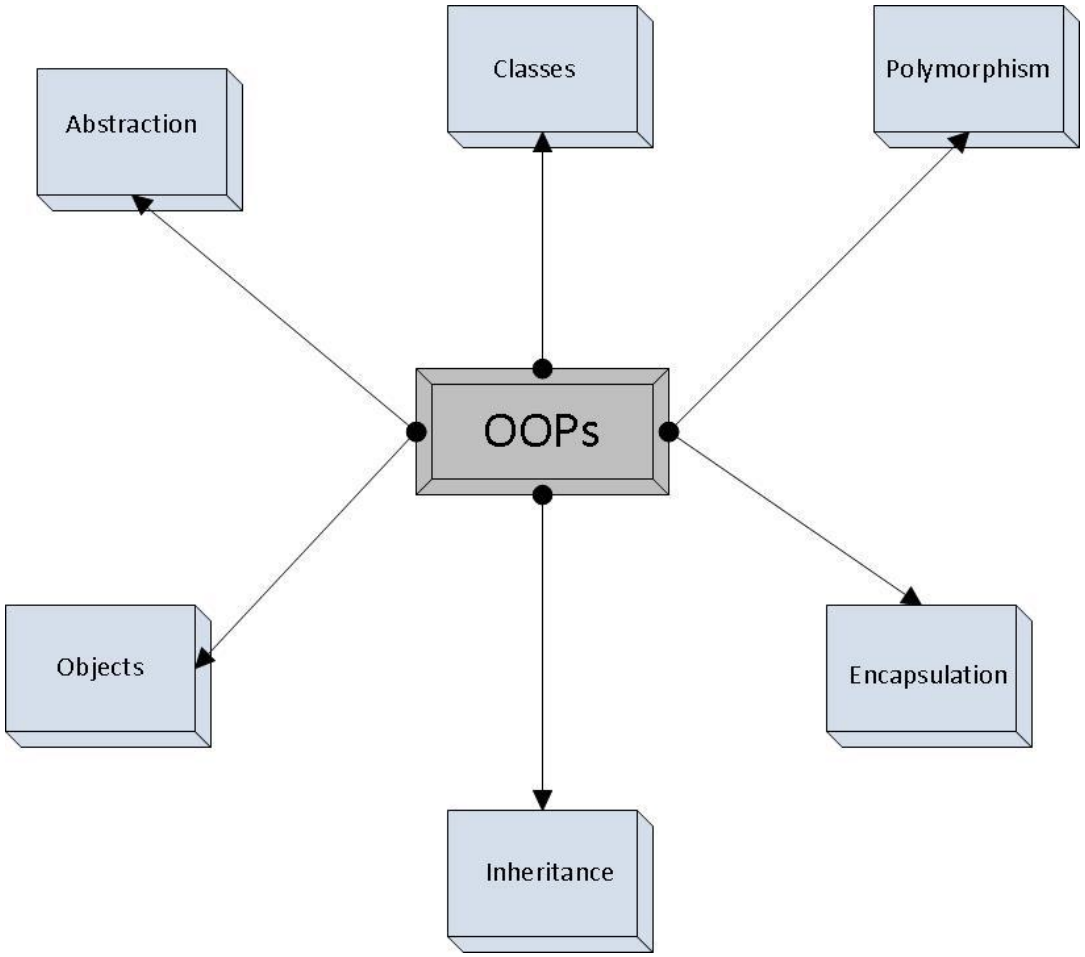
Inheritance =Basic Data + Additional features

Polymorphism= Based on variant engine capacity will vary.

Agenda

- What is OOPs
- Introduction to Classes
- Module Vs Class
- Objects
- Static class properties
- Static methods
- This
- Assignment
- Inheritance
- Super
- Data Hiding
- Protecting class methods
- Class properties and qualifiers
- Polymorphism
- Casting
- Class Scope resolution operator
- Out of block declarations
- Parameterized classes
- Shallow and Deep Copy
- Task and function

What is OOPs?


Classes
Polymorphism
Abstraction
OOPs
Objects
Encapsulation
Inheritance

contd..

- OOP is object-oriented programming
- Classes form the base of OOP programming
- Encapsulation - OOP binds data & subroutine together
- Inheritance –extend the functionality of existing objects
- Polymorphism – wait until runtime to bind data with functions

Introduction to classes

- A class is a type that includes data and subroutines (functions and tasks) that operate on that data.
- A class's data is referred to as class properties, and its subroutines are called methods, both are members of the class.
- Classes allow objects to be dynamically created, deleted, assigned, and accessed via object handles. Object handles provide a safe pointer-like mechanism to the language.
- Code minimization and reuse can be achieved by inheritance, polymorphism and parameterization

Module Vs Class

Class	Module

• Instances of classes are objects
  • A handle points to an object(class instance)
  • An object handle can be pass as arguments
  • Object memory can be copied or compared
	• Instances of modules can't be passed, copied or compared

• Objects are dynamic • Objects are created and destroyed as needed	• Modules are static

• Classes can be inherited
  • Classes can be modified via inheritance without impact existing users
	• Modules can't be inherited
	  • Modification of module will impact to all the existing users

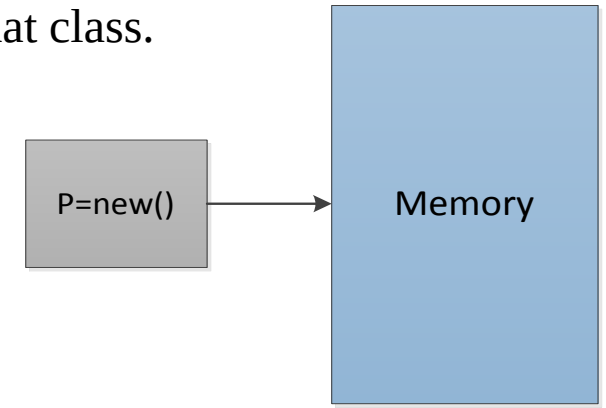
Objects (class instance)

- A class defines a data type. An object is an instance of that class.

  Syntax:

  Packet p; // declare handle or variable of class Packet

  p = **new**(); // object created of class packet



NEW constructor:

- The 'new()' is a method which is part of every class.
- It has default implementation which simply allocates the memory for the object and returns the address to the handle
- Dynamically the memory is allocated by calling the constructor(new) for the class
- The variable p is said to hold an object handle to an object of class Packet.
- Uninitialized object handles are set by default to the special value null.
- An uninitialized object can be detected by comparing its handle with **null**.

Object properties

- The data fields of an object can be used by qualifying class property names with an instance name.

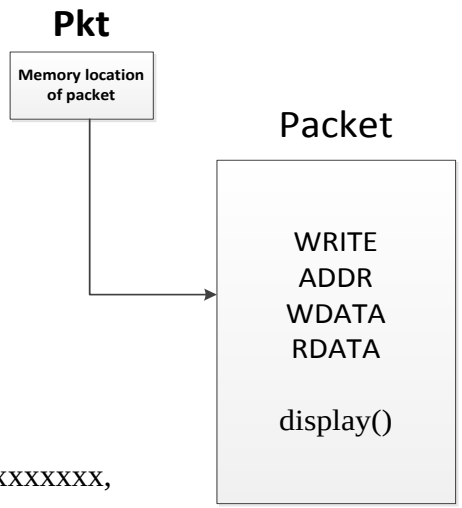
```
Packet p = new;  //refer previous example
if(p==null)
        $display("memory not allocated");
else
        $display("memory allocated");
```

- Memory deallocation can be done by assigning null to object handle, Which is automatically done by the compiler at the end of simulation or end of scope

```
p = null;
```

```
class Packet;
//data or class properties
  reg          WRITE;
  bit     [31:0] ADDR;
  integer       WDATA;
  logic [31:0]   RDATA;

// initialization //constructor
  function void display();
    $display("\nWRITE=%p,\tADDR[31:0]=%h,\tRDATA[31:0]=%h,\tWDATA[31:0]=%h",WRITE,ADDR,RDATA,WDATA);
  endfunction
endclass

  module tb;
    Packet pkt;

      initial begin
       pkt=new();
       pkt.display();
       pkt.WRITE=1;
       pkt.ADDR =32'h8f;
       pkt.RDATA=32'h00;
       pkt.WDATA=32'hf0;
       pkt.display();
      end
  endmodule
```

Note: Any data-type can be declared as a class property, except for net types since they are incompatible with dynamically allocated data



RESULT

WRITE=x, ADDR[31:0]=00000000, RDATA[31:0]=xxxxxxxx, WDATA[31:0]=xxxxxxxx

WRITE=1, ADDR[31:0]=0000008f, RDATA[31:0]=00000000 WDATA[31:0]=000000f0

Constructor inside the class

- We can declare our own new() method.

Syntax:

function new([arguments]);

//body of method

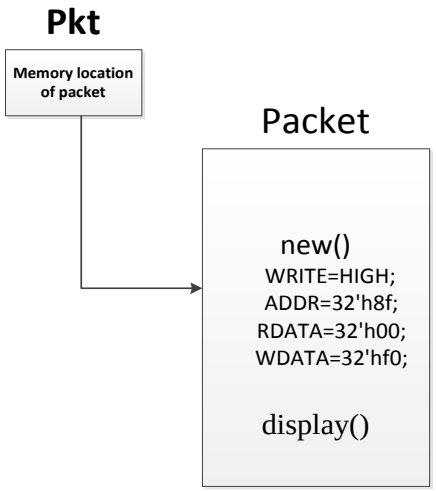
endfunction

Handle_name=**new**([arguments]);

Note:Arguments are optional

```
class Packet;
enum{LOW,HIGH}WRITE;
bit     [31:0] ADDR;
integer        WDATA;
logic [31:0]   RDATA;
function new();
   WRITE=HIGH;
   ADDR=32'h8f;
   RDATA=32'h00;
   WDATA=32'hf0;
endfunction
function void display();
$display("\nWRITE=%p,\tADDR[31:0]=%h,\tRDATA[31:0]=%h,\tWDATA[31:0]=%h",WRITE,ADDR,RDATA,WDATA);
endfunction
  endclass
```

```
module tb;
   Packet Pkt;

   initial
   begin
      pkt=new();
      pkt.display();
      $display("\n pkt=%p",pkt);
   end
endmodule
```



RESULT
WRITE=HIGH, ADDR[31:0]=0000008f, RDATA[31:0]=00000000, WDATA[31:0]=000000f0

pkt='{WRITE:HIGH, ADDR:'h8f, WDATA:240, RDATA:'h0}

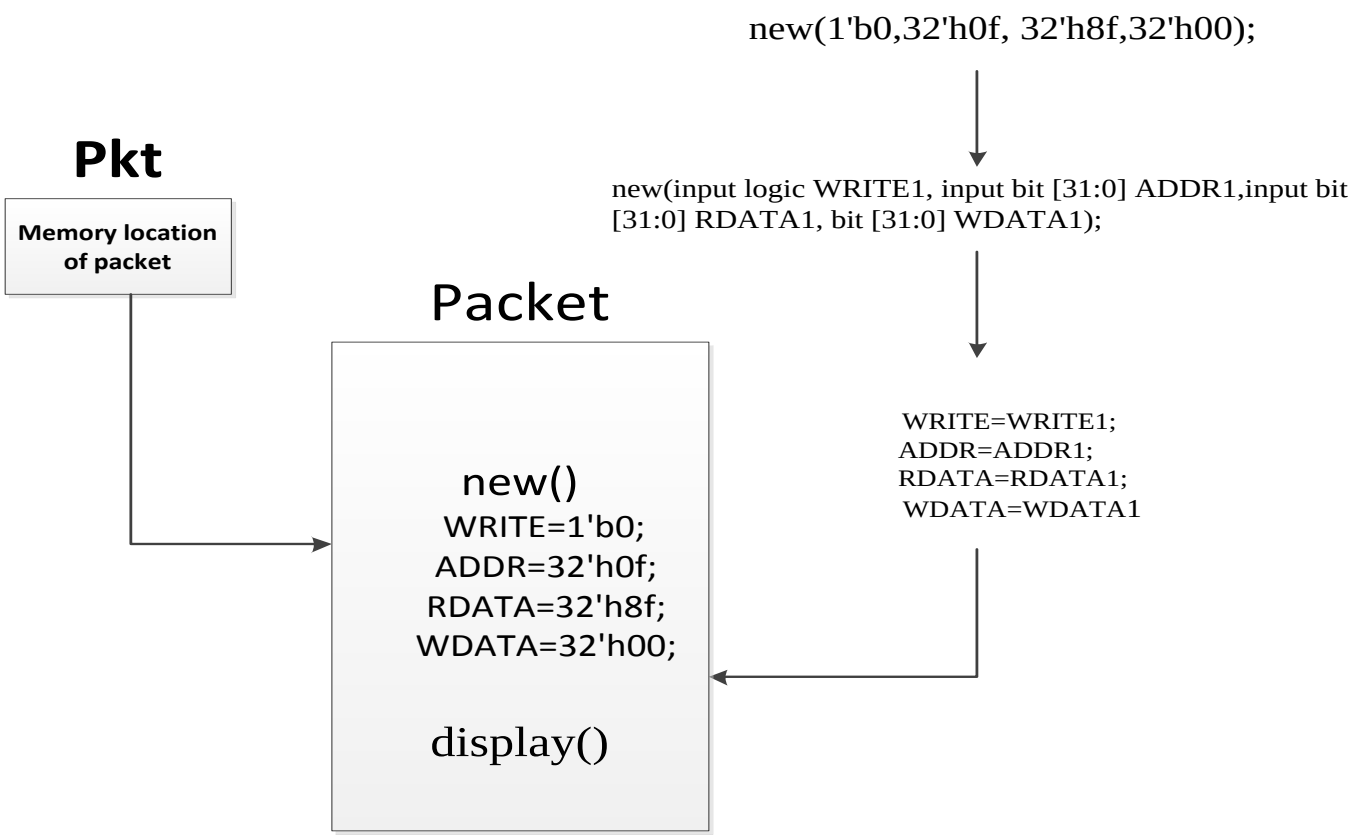
Constructor with arguments

```
class Packet;
  reg              WRITE;
  bit     [31:0]  ADDR;
  integer          WDATA;
  logic [31:0]    RDATA;


  function new(input logic WRITE1, input bit [31:0] ADDR1,input bit [31:0] RDATA1, bit [31:0] WDATA1);

     WRITE=WRITE1;
     ADDR=ADDR1;
     RDATA=RDATA1;
     WDATA=WDATA1;

  endfunction

  function void display();
    $display("\nWRITE=%p,\tADDR[31:0]=%h,\tRDATA[31:0]=%h,\tWDATA[31:0]=%h",WRITE,ADDR,RDATA,WDATA);
  endfunction
    endclass

  module tb;
    Packet pkt;

     initial begin
       pkt=new(1'b0,32'h0f, 32'h8f,32'h00);
      pkt.display();
       $display("\n pkt=%p",pkt);
      end
  endmodule
```

Result:

```
WRITE=0,        ADDR[31:0]=0000000f, RDATA[31:0]=0000008f, WDATA[31:0]=00000000

 pkt='{WRITE:'h0, ADDR:'hf, WDATA:0, RDATA:'h8f}
```

Constructor with arguments



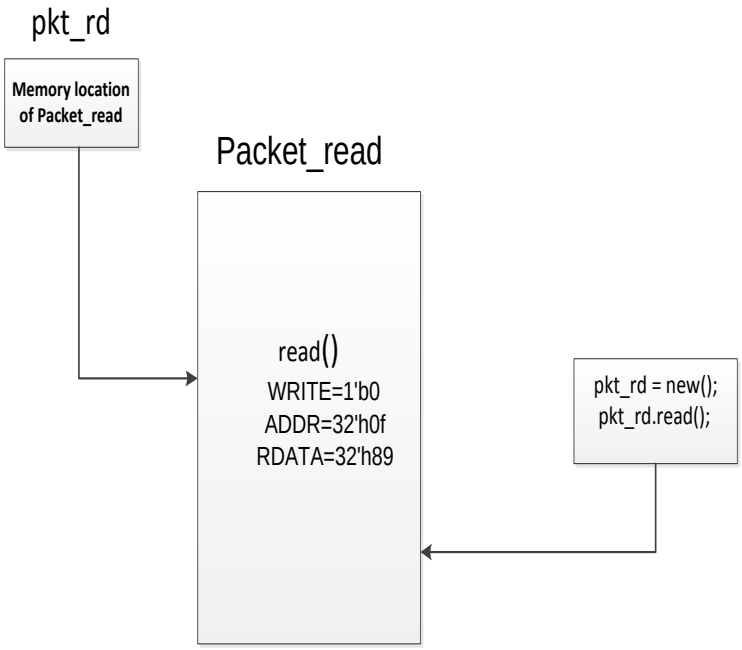
Multiple Objects Creation : Example

```
class Packet_read;
logic       WRITE;
bit [31:0] ADDR;
int          RDATA;

function void read();

    WRITE=1'b0;
    ADDR=32'h0f;
    RDATA=32'h89;
   $display("\nWRITE=%p,\tADDR[31:0]=%h,\tRDATA[31:0]=%h ",WRITE,ADDR,RDATA);
endfunction:read

endclass: Packet_read

class Packet_write;
 logic WRITE;
 bit [31:0] ADDR;
 bit [31:0] WDATA;

  function void write();

    WRITE=1'b1;
    ADDR=32'h0f;
    WDATA=32'h78;
   $display("\nWRITE=%p,\tADDR[31:0]=%h,\tWDATA[31:0]=%h",WRITE,ADDR,WDATA);
endfunction

endclass
```

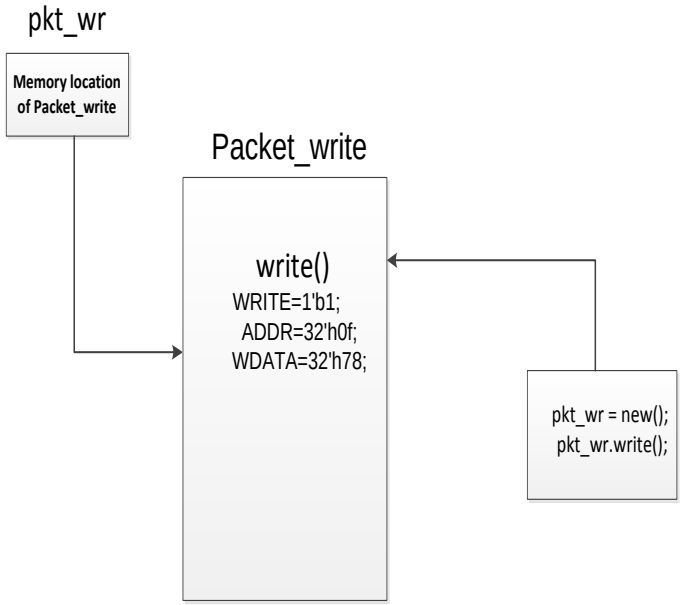
```
module tb;
  Packet_read  pkt_rd;
  Packet_write pkt_wr;

  initial
  begin
    pkt_rd = new();
    pkt_wr = new();
    pkt_rd.read();
    pkt_wr.write();
  end
endmodule
```

Result

```
WRITE=HIGH,  ADDR[31:0]=0000000f,
             RDATA[31:0]=0000008f,
             WDATA[31:0]=00000000
```



Read Packet



Write Packet

Array Handles : Example

```
class Packet;
  reg              WRITE;
  bit     [31:0]  ADDR;
  integer          WDATA;
  logic [31:0]   RDATA;


  function new(input logic WRITE1, input bit [31:0] ADDR1,input bit [31:0] RDATA1, bit
[31:0] WDATA1);

      WRITE=WRITE1;
     ADDR=ADDR1;
     RDATA=RDATA1;
     WDATA=WDATA1;

  endfunction

  function void display();

$display("\nWRITE=%p,\tADDR[31:0]=%h,\tRDATA[31:0]=%h,\tWDATA[31:0]=%h",W
RITE,ADDR,RDATA,WDATA);
  endfunction
    endclass
```

```
module tb;
   Packet pkt[4];

    initial begin
      pkt[0]=new(1'b0,32'h0f, 32'h8f,32'h00);
      pkt[0].display();
      pkt[1]=new(1'b1,32'h1a, 32'h00,32'h8e);
      pkt[1].display();
      pkt[2]=new(1'b0,32'hff, 32'h9e,32'h00);
      pkt[2].display();
      pkt[3]=new(1'b1,32'h2e, 32'h8f,32'hae);
      pkt[3].display();

    end
 endmodule
```

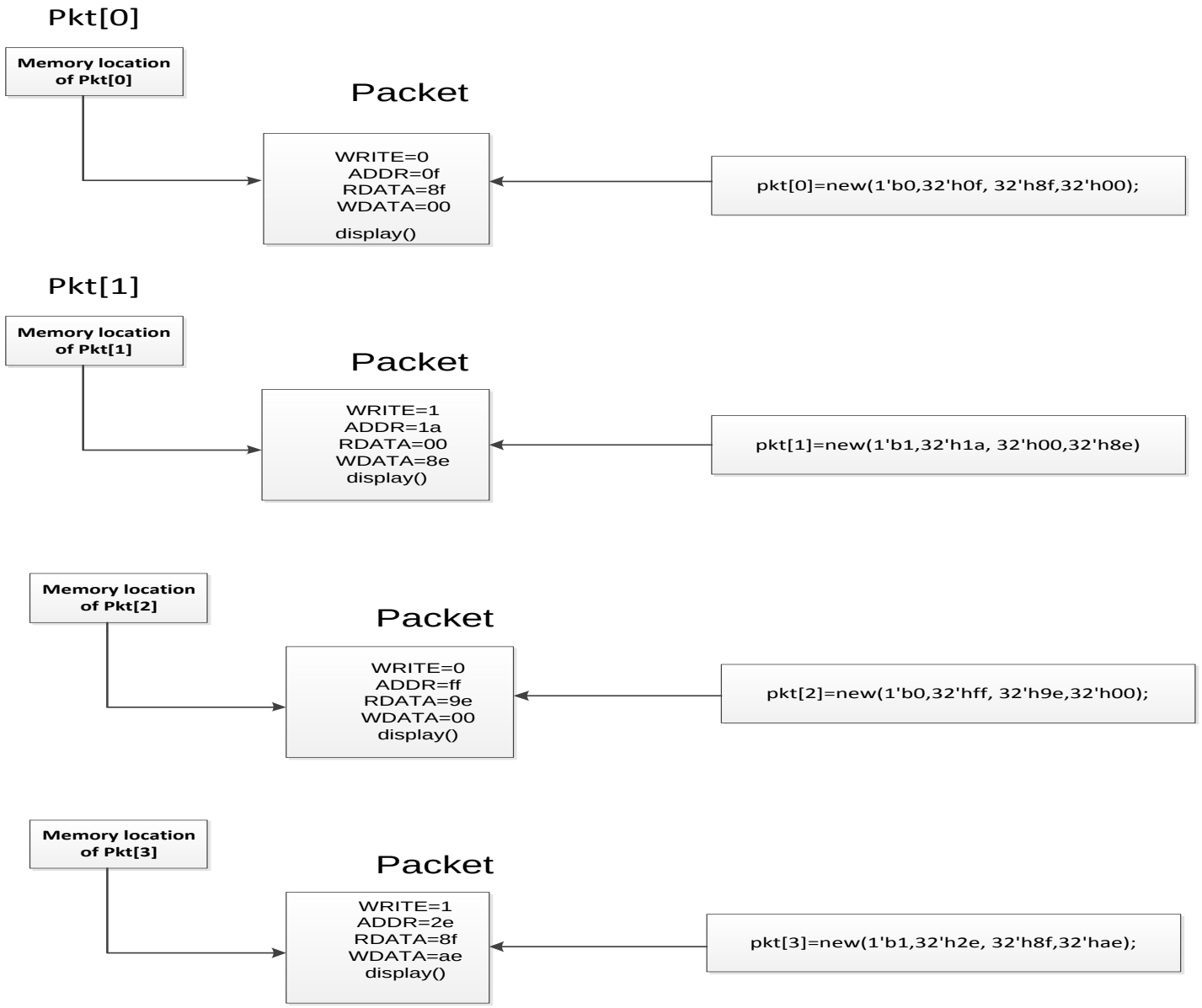
RESULT:

WRITE=0, ADDR[31:0]=0000000f, RDATA[31:0]=0000008f, WDATA[31:0]=00000000

WRITE=1, ADDR[31:0]=0000001a, RDATA[31:0]=00000000, WDATA[31:0]=0000008e

WRITE=0, ADDR[31:0]=000000ff, RDATA[31:0]=0000009e, WDATA[31:0]=00000000

WRITE=1, ADDR[31:0]=0000002e, RDATA[31:0]=0000008f, WDATA[31:0]=000000ae


Pkt[0]
Memory location of Pkt[0]
Packet
WRITE=0
ADDR=0f
RDATA=8f
WDATA=00
display()
pkt[0]=new(1'b0,32'h0f, 32'h8f,32'h00);
Pkt[1]
Memory location of Pkt[1]
Packet
WRITE=1
ADDR=1a
RDATA=00
WDATA=8e
display()
pkt[1]=new(1'b1,32'h1a, 32'h00,32'h8e)
Memory location of Pkt[2]
Packet
WRITE=0
ADDR=ff
RDATA=9e
WDATA=00
display()
pkt[2]=new(1'b0,32'hff, 32'h9e,32'h00);
Memory location of Pkt[3]
Packet
WRITE=1
ADDR=2e
RDATA=8f
WDATA=ae
display()
pkt[3]=new(1'b1,32'h2e, 32'h8f,32'hae);

```
module tb;
typedef enum {WRITE,READ} op_type;

class    Packet ;
 //data or class properties
        reg  [3:0]    command;
        bit  [40:0]   address;
        int master_id;
              string  status;

     op_type rd_wr;

  //constructor is flexible now
  function new(input reg[3:0] cmd=4'b0111,input bit[40:0]
addr='h2f ,input int m_id='h1e,input string st="notokay",input
op_type operation=READ);
    command= cmd;
           address=addr;
           master_id=m_id;
           status = st;
           rd_wr=operation;

  endfunction

 function void display();
   $display("Command=%0d \t Address=%0h \t Master_id=%0h \t
Status=%0s \t rd_wr=%p",command,address,master_id,status,rd_wr);
 endfunction
endclass

  Packet pkt[3];
                  initial begin
        pkt[0]=new(4'b1010,'h8f,'hff,"Good",WRITE);
        pkt[1]=new(4'b1111,'h6f,'he0,"okay");
        pkt[2]=new();
        pkt[0].display();
        pkt[1].display();
        pkt[2].display();
end
endmodule
```

Result:

Command=10 Address=8f Master_id=ff Status=Good rd_wr=WRITE

Command=15 Address=6f Master_id=e0 Status=okay rd_wr=READ

Command=7 Address=2f Master_id=1e Status=notokay rd_wr=READ

Static class properties

- Each instance of the class has its own copy of each of its variables.
- Sometimes only one version of a variable is required to be shared by all instances. These class properties are created using the keyword static.
- Note that static class properties can be used without creating an object of that type.
- The scope of the static variables, till be at the end of the simulation

Static Variable :Example

```
class Accnt;
static int comp_bal;
int extra_bal;

 //---- Method-----//
 function void debit(input int amt);
    comp_bal  = comp_bal - amt;
    extra_bal = 10;
 endfunction
endclass
```

```
module sv_tb;
Accnt emp1,emp2,emp3;

initial begin
Accnt::comp_bal = 100000;
$display("comp_bal=%p", Accnt ::comp_bal);
 emp1  = new();
 emp2  = new();
 emp3  = new();

emp1.debit(20000);
$display("comp_bal=%p", emp1.comp_bal);

emp2.debit(50000);
$display("comp_bal=%p", emp1. comp_bal);

 emp3.debit(10000);
$display("comp_bal=%p", emp1. comp_bal);
end
endmodule
```

o/p:

comp_bal= 100000
comp_bal= 80000
comp_bal= 30000
comp_bal= 20000

Static methods

- A static method is subject to all the class scoping and access rules, but behaves like a regular subroutine that can be called outside the class, even with no class instantiation.
- A static method has no access to non-static members (class properties or methods), but it can directly access static class properties or call static methods of the same class.
- Access to non-static members or to the special this handle within the body of a static method is illegal and results in a compiler error.
- Static methods cannot be virtual.

- A static method refers to the lifetime of the method within the class.
- A method with static lifetime refers to the lifetime of the arguments and variables within the task.

```
class TwoTasks;
  static task foo(); ... endtask // static class method with
 // automatic variable lifetime
   task static bar(); ... endtask // non-static class method
  // static variable lifetime
endclass
```

- By default, class methods have automatic lifetime for their arguments and variables.

Static Method: Example

```
class Accnt;
static int comp_bal;
int extra_bal;

 //-----Non static Method-----//
 function void debit(input int amt);
    comp_bal  = comp_bal - amt;
    extra_bal = 10;
 endfunction

 //----Static Method-----//
 static function void display_bal(input string debited_by="none");

    //$display("comp_bal=%p extra_bal=%p",comp__bal,extra_bal);//Error:Illegal access of non-static member 'extra_bal' from static method
   $display("After ",debited_by," has debited the amount then the remaining","-comp_bal=%p ",comp_bal);
 endfunction:display_bal


endclass
```

```
module sv_tb;
Accnt emp1,emp2,emp3;

initial begin
Accnt::comp_bal = 100000;
Accnt::display_bal();

 emp1  = new();
 emp2  = new();
 emp3  = new();

 emp1.debit(20000);
 emp1.display_bal("emp1");
 emp2.debit(50000);
 emp2.display_bal("emp2");
 emp3.debit(10000);
 emp3.display_bal("emp3");
 Accnt::display_bal();

end
endmodule
```

o/p:

After none has debited the amount then the remaining-comp_bal= 100000

After emp1 has debited the amount then the remaining-comp_bal= 80000

After emp2 has debited the amount then the remaining-comp_bal= 30000

After emp3 has debited the amount then the remaining-comp_bal= 20000

After none has debited the amount then the remaining-comp_bal= 20000

This

- this keyword is used to unambiguously refer to class properties or methods of the current instance.
- this keyword shall only be used within non-static class methods, otherwise an error shall be issued.

Example:

```
class Demo ;
  integer x;
 function new (integer x)
     this.x = x;
  endfunction
endclass
```

Example

```
module tb;
  typedef enum{LOW,HIGH} OP_TYPE;

  class Packet;

    OP_TYPE WRITE;
    bit    [31:0]  ADDR;
    integer        WDATA;
    logic [31:0]   RDATA;


    function new(input OP_TYPE WRITE, input bit [31:0]
ADDR,input bit [31:0] RDATA, bit [31:0] WDATA);

       this.WRITE=WRITE;
       this.ADDR=ADDR;
       this.RDATA=RDATA;
       this.WDATA=WDATA;

    endfunction
```

```
function void display();

$display("\nWRITE=%p,\tADDR[31:0]=%h,\tRDATA[31:0]
=%h,\tWDATA[31:0]=%h",WRITE,ADDR,RDATA,WDATA);
     endfunction
   endclass:Packet


   Packet pkt;

    initial begin
     pkt=new(HIGH,32'h0f, 32'h8f,32'h00);
    pkt.display();

    end
endmodule
```

Output:-

WRITE=HIGH, ADDR[31:0]=0000000f, RDATA[31:0]=0000008f, WDATA[31:0]=00000000

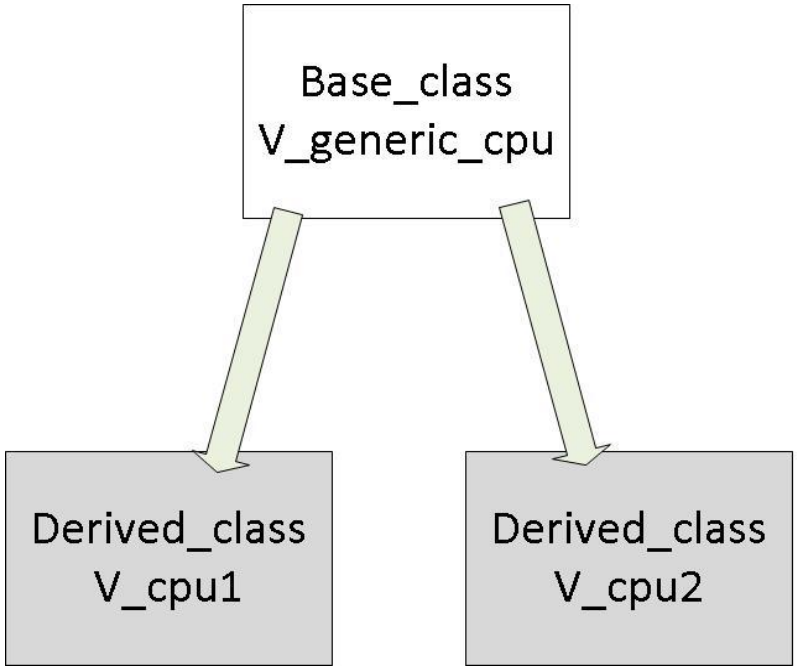
Inheritance and subclasses

- Inheritance is the mechanism which allows a class A to inherit properties of a class B. We say, “A inherits from B”. Objects of class A thus have access to attributes and methods of class B without the need to redefine them.
- Definition of Superclass or Base class: If class A inherits from class B, then B is called superclass of A.
- Definition of Subclass or Derived class: If class A inherits from class B, then A is called subclass of B.
- Derived classes may override the definition of a member inherited from the base class

Inheritance example

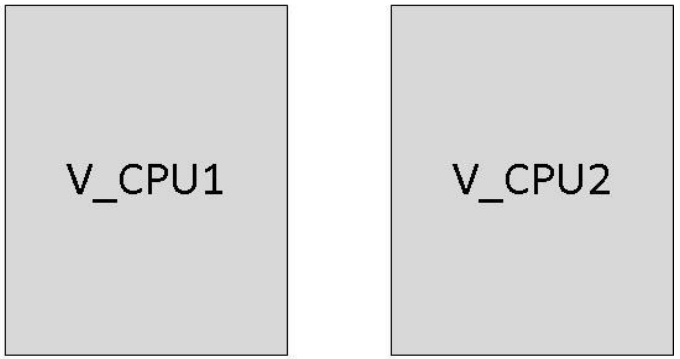
With Inheritance

- Inheritance approach is ideal for verification

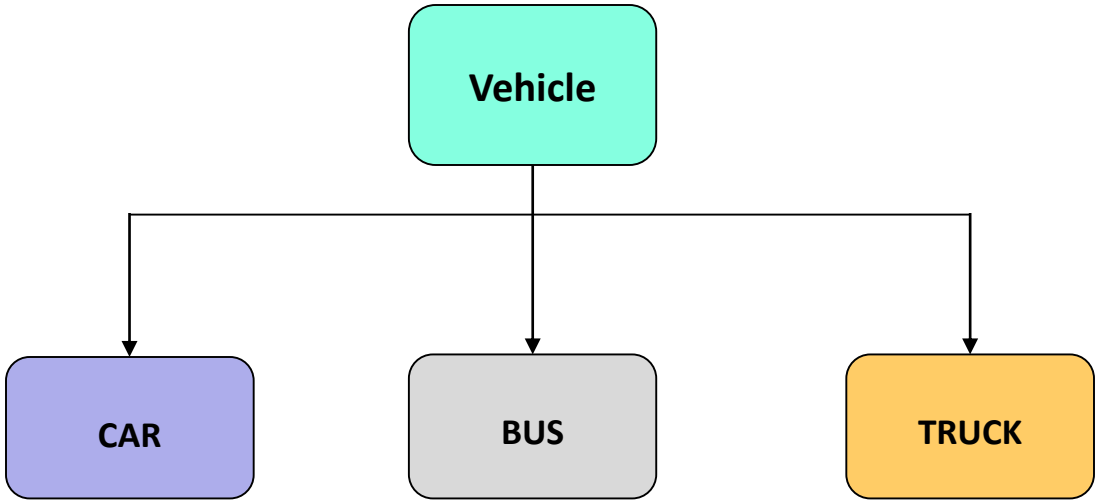


With-out Inheritance

- Non-inheritance approach involves duplicated effort



Inheritance example



Inheritance and subclasses: Example

```
class vehical;
  int start=1;
  int stop=0;
endclass:vehical

class car extends vehical;
  int seat=4;
endclass:car

class bus extends vehical;
  int seat=60;
  int AC=0;
endclass:bus
module tb;
  vehical ve=new();
  car ca=new();
  bus bu=new();
  initial begin
    $display("default value=%0p",ve);
    $display("default value=%0p",ca);
    $display("default value=%0p",bu);
  end
endmodule
```

Result:

default value='{start:1, stop:0}

default value='{start:1, stop:0, seat:4}

default value='{start:1, stop:0, seat:60, AC:0}

Inheritance Packet

```
class Packet;
  logic       WRITE=0;
  bit [31:0] ADDR=32'hff;
  int          RDATA=32'hff;
 endclass: Packet

class P_Packet extends Packet;
  logic PENABLE='h 1;
endclass:P_Packet

class H_Packet extends Packet;
  string HRESP = "OKAY";

endclass:H_Packet
```

```
module tb;
  Packet Pkt;
  P_Packet P_Pkt;
  H_Packet H_Pkt;

   initial
   begin
     Pkt = new();
     P_Pkt = new();
     H_Pkt=new();
     $display("\n Packet=%p",Pkt);
     $display("\n P_Packet=%p",P_Pkt);
     $display("\n H_Packet=%p",H_Pkt);
   end
endmodule
```

Result:

Packet='{WRITE:'h0, ADDR:'hff, RDATA:255}

P_Packet='{WRITE:'h0, ADDR:'hff, RDATA:255, PENABLE:'h1}

H_Packet='{WRITE:'h0, ADDR:'hff, RDATA:255, HRESP:"OKAY"}

Inheritance advantages

- Inheritance is to reuse the existing code
- Common code can be grouped into one class
- No need to modify the existing classes
- Add new features to existing class by means of new derived classes
- Easy debug & easy to maintain the code base

Super

- The super keyword is used ,within a derived class to refer to members of the parent class.
- It is necessary to use super to access members of a parent class when those members are overridden by the derived class.
- Only legal from child classes but not from same parent class

```
class Packet;
  logic        WRITE;
  bit [31:0] ADDR;
  int          RDATA;
  integer    WDATA;

endclass: Packet

 class P_Packet extends Packet;
  logic PENABLE;

endclass:P_Packet
```

Super keyword with variable

```
class Packet;
  logic      WRITE;
  bit [31:0] ADDR;
  int        RDATA;
  integer    WDATA;
 endclass: Packet


class P_Packet extends Packet;
  logic PENABLE;
  bit   WRITE;

    function void initialize();
                    super.WRITE='bz;
                    ADDR='d4;
                    RDATA='d48;
                    WDATA='d55;
                    WRITE=1'b1;
    endfunction

endclass:P_Packet
```

```
module tb;
  Packet Pkt;
  P_Packet P_Pkt;
   initial
   begin
     Pkt = new();
     P_Pkt = new();
     P_Pkt.initialize();
     $display("\n------> P_Packet=%p",P_Pkt);
     $display("\n------> Packet=%p",Pkt);
    end
endmodule
```

Result

```
------>   P_Packet='{WRITE:'hz, ADDR:'h4, RDATA:48, WDATA:55, PENABLE:'hx, WRITE:'h1}

------>  Packet='{WRITE:'hx, ADDR:'h0, RDATA:0, WDATA:x}
```

Super keyword in constructor

```
class Packet;
  logic      WRITE;
  bit [31:0] ADDR;
  int        RDATA;
  integer    WDATA;

  function new(input logic wr,input bit[31:0] address,input int readdata,input integer writedata);
     WRITE = wr;
     ADDR  = address;
     RDATA = readdata;
     WDATA = writedata;
  endfunction:new

 endclass: Packet

class P_Packet extends Packet;
  logic PENABLE;
  function new(input logic p_wr,input bit[31:0] p_address,input int p_readdata,input integer p_writedata,input logic pen);

    super.new(p_wr,p_address,p_readdata,p_writedata);

    PENABLE = pen;
  endfunction:new

endclass:P_Packet
```

```
class H_Packet extends Packet;
  string HRESP;
  function new(input logic h_wr,input bit[31:0] h_address,input int h_readdata,input integer h_writedata,input string hrsp);
    super.new(h_wr,h_address,h_readdata,h_writedata);
    HRESP = hrsp;
  endfunction:new
endclass:H_Packet


module tb;
  Packet Pkt;
  P_Packet P_Pkt;
  H_Packet H_Pkt;

   initial
   begin
     Pkt = new(1'b1,32'hFF,'d30,'d1234);
     P_Pkt = new(.p_address(32'h35),.p_wr(0),.p_readdata('d48),.pen(1),.p_writedata('d7456));
     H_Pkt=new(0,32'hB2,'d508,32'd2205,"RETRY");
     $display("\n Packet=%p",Pkt);
     $display("\n P_Packet=%p",P_Pkt);
     $display("\n H_Packet=%p",H_Pkt);
   end
endmodule
```

Result:

Packet='{WRITE:'h1, ADDR:'hff, RDATA:30, WDATA:1234}

P_Packet='{WRITE:'h0, ADDR:'h35, RDATA:48, WDATA:7456, PENABLE:'h1}

H_Packet='{WRITE:'h0, ADDR:'hb2, RDATA:508, WDATA:2205, HRESP:"RETRY"}

Super keyword with constructor and Method

```
class Packet;
  logic      WRITE;
  bit [31:0] ADDR;
  int        RDATA;
  integer    WDATA;

  function new(input logic wr,input bit[31:0] address,input int readdata,input integer writedata);
     WRITE = wr;
     ADDR  = address;
     RDATA = readdata;
     WDATA = writedata;
  endfunction:new

  task Incrval_members(input int val);
     WRITE = WRITE+val;
     ADDR  = ADDR+val;
     RDATA = RDATA+val;
     WDATA = WDATA+val;
  endtask:Incrval_members

 endclass: Packet

class P_Packet extends Packet;
  logic PENABLE;

  function new(input logic p_wr,input bit[31:0] p_address,input int p_readdata,input integer p_writedata,input logic pen);
    super.new(p_wr,p_address,p_readdata,p_writedata);
    PENABLE = pen;
  endfunction:new

  task Incrval_members(input int p_val);
    super.Incrval_members(p_val);
    PENABLE = PENABLE+p_val;
  endtask:Incrval_members

endclass:P_Packet
```

```
class H_Packet extends Packet;
  string HRESP;
  function new (input logic h_wr,input bit[31:0] h_address,input int h_readdata,input
integer h_writedata,input string hrsp);
    super.new(h_wr,h_address,h_readdata,h_writedata);
    HRESP = hrsp;
  endfunction:new

  task Incrval_members(input int h_val,input string resp);
    super.Incrval_members(h_val);
  endtask:Incrval_members

endclass:H_Packet
```

Result:

Packet='{WRITE:'h1, ADDR:'h4, RDATA:32, WDATA:1236}

P_Packet='{WRITE:'h0, ADDR:'h8, RDATA:52, WDATA:7460, PENABLE:'h1}

H_Packet='{WRITE:'h1, ADDR:'h7, RDATA:509, WDATA:2206, HRESP:"RETRY"}

```
module tb;
  Packet Pkt;
  P_Packet P_Pkt;
  H_Packet H_Pkt;

   initial
   begin
     Pkt = new(1'b1,'h2,'d30,'d1234);
     P_Pkt =
new(.p_address('h4),.p_wr(0),.p_readdata('d48),.pen(1),.p_writedata('d7456));
     H_Pkt=new(0,'h6,'d508,32'd2205,"RETRY");
     Pkt.Incrval_members(2);
     P_Pkt.Incrval_members(4);
     H_Pkt.Incrval_members(1,"OKAY");
     $display("\n Packet=%p",Pkt);
     $display("\n P_Packet=%p",P_Pkt);
     $display("\n H_Packet=%p",H_Pkt);

   end
endmodule
```

Base class handle is pointing to the child object:

```
class Packet;
  logic      WRITE;
  bit [31:0] ADDR;
  int        RDATA;
  integer    WDATA;

  function new(input logic wr,input bit[31:0] address,input int readdata,input integer writedata);
     WRITE = wr;
     ADDR  = address;
     RDATA = readdata;
     WDATA = writedata;
  endfunction:new

  task Display();
    $display("------BASECLASS:WRITE=%0d ADDR=%0d RDATA=%0d  WDATA=%0d-------",WRITE,ADDR,RDATA,WDATA);
  endtask:Display


 endclass: Packet
```

```
class P_Packet extends Packet;
  logic PENABLE;
  function new(input logic p_wr,input bit[31:0] p_address,input int p_readdata,input integer p_writedata,input logic pen);
    super.new(p_wr,p_address,p_readdata,p_writedata);
    PENABLE = pen;
  endfunction:new

  task Display();
    $display("-------DERIVEDCLASS:WRITE=%0d ADDR=%0d RDATA=%0d  WDATA=%0d PENABLE=%0d-------",WRITE,ADDR,RDATA,WDATA,PENABLE);
  endtask:Display

endclass:P_Packet

module tb;
  Packet Pkt;
  P_Packet P_Pkt;

   initial
   begin
     Pkt = new(1'b1,'h2,'d30,'d1234);
     P_Pkt =new(.p_address('h4),.p_wr(0),.p_readdata('d48),.pen(1),.p_writedata('d7456));
     Pkt.Display();
     P_Pkt.Display();

     //pointing a base class handle to child(derived) object
     Pkt = P_Pkt;
     $display("**************pointing a base class handle to child(derived) object************");
     Pkt.Display();
     P_Pkt.Display();

   end
endmodule
```

Cond ..

------BASECLASS:WRITE=1 ADDR=2 RDATA=30 WDATA=1234-------
-------DERIVEDCLASS:WRITE=0 ADDR=4 RDATA=48 WDATA=7456 PENABLE=1-------

**********pointing a base class handle to child(derived) object**********

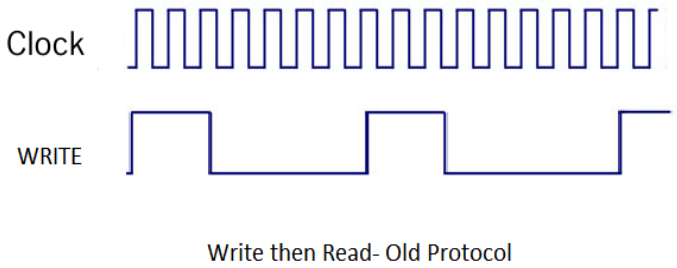
------BASECLASS:WRITE=0 ADDR=4 RDATA=48 WDATA=7456-------
-------DERIVEDCLASS:WRITE=0 ADDR=4 RDATA=48 WDATA=7456 PENABLE=1-------

Can we achieve the below Requirement:

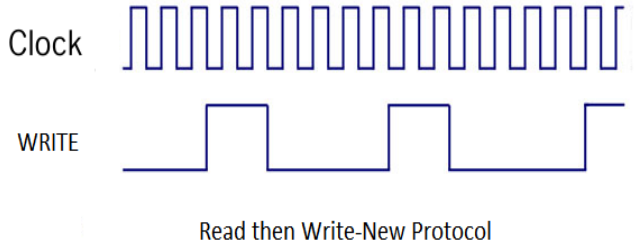
In Base the data driving mechanism is Write then Read //**Old rule of the protocol**

In new version we want the Read to be first then write //**No more old rule , new rule has to be taken place**

Handle can be Base or Child, we want to see only the New rule of Protocol, means making no existence to the old rule



Write then Read- Old Protocol



Read then Write-New Protocol

Polymorphism

- Polymorphism allows the use of a variable in the super class to hold subclass objects, and to reference the methods of those subclasses directly from the super class variable.
- This means that you can ask many different objects to perform the same action.
- Polymorphism allows the redefining of methods for derived classes.
- To achieve polymorphism the 'virtual' identifier must be used when defining the method(s) within the base class.
- The methods which are added in the subclasses which are not in the parent class can't be accessed using the parent class handle. This will result in a compilation error.

Polymorphism : Example

- In order to achieve polymorphism, three steps must be followed
  a) Method name of both parent and child should be same
  b) Prior to the method name of parent, 'virtual' keyword must be included
  c) Parent handle should be pointed to child handle
- When 'virtual' keyword is used , the simulator checks whether the parent class handle is pointed to child class handle only during run time

  Syntax: parent_handle = child_handle;
- This helps in overriding parent class methods with child class methods

  Syntax: virtual **function void** display();

Example 1

```
class BasePacket;
logic   WRITE='b1;
  bit [31:0] ADDR='h88;
  int       RDATA='hff;
  integer    WDATA='d00;


   virtual function void print();
$display("\nBasePacket::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction : print
endclass : BasePacket

Class P_Packet extends BasePacket;

 logic   WRITE='b0;
  bit [31:0] ADDR='hE8;
  int       RDATA='h00;
  integer    WDATA='d12;

function void print();
    $display("\nDerived Class::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction : print
endclass : P_Packet
```

```
module tb;
BasePacket Pkt = new;
P_Packet P_Pkt = new;
initial begin
        Pkt.print;

        Pkt = P_Pkt ;
        Pkt.print;
        P_Pkt.print;

end
endmodule
```

Without Virtual:

BasePacket::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

BasePacket::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

With Virtual:

BasePacket::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

Example 2

```
class BasePacket;
logic   WRITE='b1;
  bit [31:0] ADDR='h88;
  int       RDATA='hff;
  integer    WDATA='d00;


   function void printA();
     $display("\n printA:::BasePacket ::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction : printA

  function void printB();
    $display("\n printB:::BasePacket::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction : printB
endclass : BasePacket

class P_Packet extends BasePacket;

 logic   WRITE='b0;
  bit [31:0] ADDR='hE8;
  int       RDATA='h00;
  integer    WDATA='d12;

  function void printA();
    $display("\n printA:::Derived Class::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction: printA
```

```
function void printB();
  $display("\n printB:::Derived
Class::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction : printB
endclass : P_Packet

module tb;
BasePacket Pkt = new;
P_Packet P_Pkt = new;
initial begin
Pkt.printA;
Pkt.printB;

Pkt = P_Pkt ;
  $display("--------Base pointing to Derived-------");
Pkt.printA;
Pkt.printB;
P_Pkt.printA;
P_Pkt.printB;

end
endmodule
```

1)Without virtual:

printA:::BasePacket ::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

printB:::BasePacket::WRITE=1,ADDR=88,RDATA=ff,WDATA=0
--------Base pointing to Derived-------

printA:::BasePacket ::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

printB:::BasePacket::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

printA:::Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

printB:::Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

2) **With virtual to PrintB**
printA:::BasePacket ::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

printB:::BasePacket::WRITE=1,ADDR=88,RDATA=ff,WDATA=0
--------Base pointing to Derived-------

printA:::BasePacket ::WRITE=1,ADDR=88,RDATA=ff,WDATA=0

printB:::Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

printA:::Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

printB:::Derived Class::WRITE=0,ADDR=e8,RDATA=0,WDATA=12

Abstract classes and Pure virtual methods

- class ‘A’ is called abstract class if it is only used as a super class for other classes. Class ‘A’ only specifies properties. It is not used to create objects.
- Derived classes must define the properties of ‘A’.
- Virtual classes or Abstract class can not be instantiated, it can only be derived.
- Methods of normal classes can also be declared virtual, but the method must have a body.
- A virtual method overrides a method in all the base classes, whereas a normal method only overrides a method in that class and its descendant.
- Pure virtual methods are recommended
    a) These pure virtual methods are only declared inside virtual class and can’t be defined
    b) The definition can only be given in child class, provided it has to follow with same prototype methods

Abstract classes :Example1

```
module tb;
  Packet Pkt;
    P_Packet P_Pkt;
   initial
   begin
     //Pkt=new();
    P_Pkt=new();
    P_Pkt.Print();
    P_Pkt.display();

    end
endmodule
```

```
virtual class Packet; //Virtual classes infront of class;
  logic      WRITE='b1;
  bit [31:0] ADDR='h88;
  int        RDATA='hff;
  integer    WDATA='d00;

  function void Print();
    $display("------>Inside Base Method:WRITE=%0d,ADDR=%0h,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction

endclass: Packet


class P_Packet extends Packet;


  function void display();
    $display("------>Inside Derived Method:WRITE=%0d,ADDR=%0h,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction


endclass:P_Packet
```

Result:

```
------>Inside Base Method:WRITE=1,ADDR=88,RDATA=255,WDATA=0
------>Inside Derived Method:WRITE=1,ADDR=88,RDATA=255,WDATA=0
```

Abstract class with Pure virtual Methods

```
module tb;
  Packet Pkt;
   P_Packet P_Pkt;
  initial
  begin
    P_Pkt=new();
    P_Pkt.Print();
    P_Pkt.display();

   end
endmodule
```

```
virtual class Packet;
  logic      WRITE='b1;
  bit [31:0] ADDR='h88;
  int        RDATA='hff;
  integer    WDATA='d00;

  pure virtual function void Print();
   /* $display("Inside Base Method:WRITE= %0d, ADDR= %0h,RDATA= %0d,WDATA= %d",WRITE,ADDR ,RDATA ,WDATA);*/
 // endfunction

endclass: Packet


class P_Packet extends Packet;

  function void Print();
    $display("------>Print:::Inside Derived Method:WRITE=%0d,ADDR=%0h,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction

  function void display();
    $display("------>display:::Inside Derived Method:WRITE=%0d,ADDR=%0h,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction

endclass:P_Packet
```

Note:

1)Pure virtual with definition // gets an Error

2)Without definition and not implemented in Derived

3)Definition in Derived

Instantiation of the object 'Pkt' can not be done because its type 'Packet' is an abstract base class.

Perhaps there is a derived class that should be used

```
------>Print::: Inside Derived
Method:WRITE=1,ADDR=88,RDATA=255,WDATA=0
------>display:::Inside Derived
Method:WRITE=1,ADDR=88,RDATA=255,WDATA=0
```

Data hiding

- **It is desirable to restrict access to class properties and methods from outside the class by hiding their names.**
- This keeps other programmers from relying on a specific implementation, and it also protects against accidental modifications to class properties that are internal to the class.
- When all data becomes hidden being accessed only by public methods testing and maintenance of the code becomes much easier.
- We have following access specifiers to achieve data hiding:
    a) Local
    b) protected
- By default every class members are '**public**'. Which can access from anywhere
- '**local**' keyword is for availability of members within the same class but not from extended class or outside of the class
- '**protected**' members are like '**local**' members but it's accessible to their derived classes

Protecting class methods

Local:

- When the method is declared with the keyword ‘local’, that method is valid within the class and not accessible outside the class or from the extended class(child class).

Protected:

- When the method is declared with the keyword ‘protected’, that method is valid within the class and extended classes but not accessible outside the class.

Data hiding - public: Example 1

```
class Packet;
  logic       WRITE='b1;
  bit [31:0] ADDR='h88;
  int          RDATA='hff;
  integer    WDATA='d00;

function new();
  $display("Inside Base Constructor:::::WRITE=%0d,ADDR=%0d,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction
endclass: Packet

class P_Packet extends Packet;
  function display();
    $display("Inside Derived Method:::::WRITE=%0d,ADDR=%0d,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction

endclass:P_Packet

 module tb;
  Packet Pkt;
  P_Packet P_Pkt;

   initial
   begin
    Pkt      =new();
    P_Pkt =new();
     P_Pkt.ADDR='h55;
     P_Pkt.display();
     $display("\n At Module level Packet=%p",Pkt);
     $display("\n At Module level P_Packet=%p",P_Pkt);
    end
endmodule
```

Result:

```
Inside Base Constructor:::::WRITE=1,ADDR=136,RDATA=255,WDATA=0
Inside Base Constructor:::::WRITE=1,ADDR=136,RDATA=255,WDATA=0
Inside Derived Method:::::WRITE=1,ADDR=85,RDATA=255,WDATA=0

At Module level Packet='{WRITE:'h1, ADDR:'h88, RDATA:255, WDATA:0}

At Module level P_Packet='{WRITE:'h1, ADDR:'h55, RDATA:255, WDATA:0}
```

Data hiding - local: Example 2

```
class Packet;
  logic   WRITE='b1;
  local bit [31:0] ADDR='h88;
  int        RDATA='hff;
  integer    WDATA='d00;
endclass: Packet

class P_Packet extends Packet;
  function new();
    $display("Inside Derived Constructor
WRITE=%0d,ADDR=%0d,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction


endclass:P_Packet


module tb;
  Packet Pkt;
  P_Packet P_Pkt;


   initial
   begin
    Pkt=new;

    P_Pkt=new;
    // P_Pkt.ADDR='h55;
     $display("\n Packet=%p",Pkt);
     $display("\n P_Packet=%p",P_Pkt);

   end
endmodule
```

Result:

1)Local member 'ADDR' of class 'Packet' is not visible to scope 'P_Packet'.
Please make sure that the above member is accessed only from its own class properties as it is declared as local.

2)Local member 'ADDR' of class 'Packet' is not visible to scope 'tb'.
Please make sure that the above member is accessed only from its own class properties as it is declared as local. // P_Pkt.ADDR='h55;

3)Packet='{WRITE:'h1, ADDR:'h88, RDATA:255, WDATA:0}

P_Packet='{WRITE:'h1, ADDR:'h88, RDATA:255, WDATA:0}

Data hiding - Protected: Example 1

```
class Packet;
  logic   WRITE='b1;
  protected bit [31:0] ADDR='h88;
  int        RDATA='hff;
  integer    WDATA='d00;
  function new();
   $display("Inside Base Constructor:::WRITE=%0d,ADDR=%0d,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction
endclass: Packet

class P_Packet extends Packet;
    task Display();
    $display("Inside Derived Method:::WRITE=%0d,ADDR=%0d,RDATA=%0d,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endtask:Display

  endclass:P_Packet
```

```
module tb;
  Packet Pkt;
  P_Packet P_Pkt;
   initial
  begin
   Pkt=new;
   P_Pkt=new;
   //P_Pkt.ADDR='h55;
    $display("\n At module level Packet=%p",Pkt);
    $display("\n At module level P_Packet=%p",P_Pkt);


   end
endmodule
```

```
1)Error-[SV-ICVA] Illegal class variable access
testbench.sv, 27
  Protected member 'ADDR' of class 'Packet' is not visible to scope 'tb'.
  Please make sure that the above member is accessed only from its own class
  or inherited class properties as it is declared as protected.
```

2)Inside Base Constructor:::WRITE=1,ADDR=136,RDATA=255,WDATA=0
Inside Base Constructor:::WRITE=1,ADDR=136,RDATA=255,WDATA=0

At module level Packet='{WRITE:'h1, ADDR:'h88, RDATA:255, WDATA:0}

At module level P_Packet='{WRITE:'h1, ADDR:'h88, RDATA:255, WDATA:0}

Constant class properties

- Class properties can be made read-only by a const declaration like any other System Verilog variable.
- Because class objects are dynamic objects, class properties allow two forms of read-only variables: **global constants** and **instance constants**.
- Typically, global constants are also declared static since they are the same for all instances of the class.
- However, an instance constant cannot be declared static, since that would disallow all assignments in the constructor
- Global constant class properties include an initial value as part of their declaration. They are similar to other const variables in that they cannot be assigned a value anywhere other than in the declaration
- Instance constants do not include an initial value in their declaration, only the const qualifier. This type of constant can be assigned a value at run time, but the assignment can only be done once in the corresponding class constructor.

Global constant

```
class Packet;
 const int i=5;
 logic      WRITE='b1;
 bit [31:0] ADDR='h88;
 int        RDATA='hff;
 integer    WDATA='d00;

  function void Print();

    $display("Inside Base Method:WRITE= %0d, ADDR= %0h,RDATA= %0d,WDATA= %0d  I= %0d",WRITE,ADDR ,RDATA ,WDATA, i);
 endfunction

endclass: Packet



module tb;
  Packet Pkt;

   initial
   begin

      Pkt=new();
     //Pkt.i=10;
      Pkt.Print();
    end
endmodule
```

Inside Base Method:WRITE= 1, ADDR= 88,RDATA= 255,WDATA= 0 I= 5

Instance constant

```
class Packet;
const int i;
 logic      WRITE='b1;
 bit [31:0] ADDR='h88;
 int        RDATA='hff;
 integer    WDATA='d00;

  function new();
    i++;
    $display("Inside Base Method:WRITE= %0d, ADDR= %0h,RDATA= %0d,WDATA= %d  I= %0d",WRITE,ADDR ,RDATA ,WDATA, i);
 endfunction

endclass: Packet



module tb;
  Packet Pkt;

   initial
   begin

      Pkt=new();

    end
endmodule
```

Inside Base Method:WRITE= 1, ADDR= 88,RDATA= 255,WDATA= 0 I= 5

Parameterized classes

- It is often useful to define a generic class whose objects can be instantiated to have different array sizes or data types. This avoids writing similar code for each size or type and allows a single specification to be used for objects that are fundamentally different.
- The normal Verilog parameter mechanism is used to parameterize a class:

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

- Instances of this class can then be instantiated like modules or interfaces:

```
vector #(10) vten;          // object with vector of size 10
vector #(.size(2)) vtwo;    // object with vector of size 2
typedef vector#(4) Vfour;   // Class with vector of size 4
```

- Parameterized classes are two types

a) Value parameter

b) Type parameter

Type as parameter

- This feature is particularly useful when using types as parameters:

```
class stack #(type T = int);
        local T items[];
    task push( T a ); ... endtask
    task pop( ref T a ); ... endtask
endclass
```

- The above class defines a generic stack class that can be instantiated with any arbitrary type:

```
stack is;                    // default: a stack of int's
stack #(bit[1:10]) bs;  // a stack of 10-bit vector
stack #(real) rs;          // a stack of real numbers
```

Value Parameterized classes : Example

```
class BasePacket#(int size= 2 );
logic   WRITE='b1;
  bit [size-1:0] ADDR='h84;
  int        RDATA='hff;
  integer    WDATA='d00;


   function void printA;
     $display("\nBasePacket::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction : printA

endclass : BasePacket


module tb;
  BasePacket #(12) P1;
  BasePacket #(4)  P2;
  BasePacket         P3;


initial begin
   P1 = new();
   P2 = new();
   P3 = new();

P1.printA();
P2.printA();
P3.printA();

end
endmodule
```

```
Result:

BasePacket::WRITE=1,ADDR=84,RDATA=ff,WDATA=0

BasePacket::WRITE=1,ADDR=4,RDATA=ff,WDATA=0

BasePacket::WRITE=1,ADDR=0,RDATA=ff,WDATA=0
```

Extending Parameterized classes

- A parameterized class can extend another parameterized class.

Ex:-

```
class C #(type T = bit);
   ...
endclass      // base class
```

- **class** D1 # (**type** P = real) **extends** C;

/* T is bit (the default) class D1 extends the base class C using the base class's default type (bit) parameter */

- **class** D2 #(**type** P = **real**) **extends** C #(**integer**);

/*T is integer class D2 extends the base class C using an integer parameter */

- **class** D3 #(**type** P = **real**) **extends** C #(P);

/*T is P class D3 extends the base class C using the parameterized type (P) with which the extended class is parameterized */

Extending Parameterized classes :Example

```
class Packet #(type I=int);
  I out;

  function I add(I a);
  return out+a;
  endfunction
endclass

module tb;

  // Override default value of 8 with the given values in #()
  Packet            Pkt;          // pass 16 as "size" to this class object
  Packet  #(integer)  Pkt1;       // pass 8 as "size" to this class object
  typedef Packet #(int) HPkt;     // create an alias for a class with "size" = 4
as "nibble"
  HPkt nibble;

initial begin
  // 1. Instantiate class objects
  Pkt = new();
  Pkt1 = new();
  nibble = new();
   Pkt.out='h2;
  // 2. Print size of "out" variable. $bits() system task will return
  // the number of bits in a given variable
  $display ("Pkt.out   = %0d bits", Pkt.add(10));
        Pkt1.out='h2;
  $display ("Pkt1.out   = %0d bits", Pkt1.add('hF));
        nibble.out='h2;
  $display ("nibble.out = %0d bits", nibble.add(4));
 end
endmodule
```

```
Pkt.out    = 12 bits
Pkt1.out    = 17 bits
nibble.out = 6 bits
```

```
class BasePacket#(type Datatype=int);
  logic             WRITE='b1;
  bit [31:0]   ADDR='h84;
 Datatype         RDATA;
 Datatype         WDATA;


   function void printA;

$display("\nBasePacket::WRITE=%0d,ADDR=%0h,RDATA=%0h,WDATA=%0d"
,WRITE,ADDR,RDATA,WDATA);
  endfunction : printA

    function void printB;

class My_Packet#(int Max =20) extends BasePacket;
logic   WRITE='b1;
  bit [Max-1:0] ADDR='hA3;
  int       RDATA='h0f;
  integer    WDATA='d00;
```

```
 function void printA;

$display("\nMy_Packet::WRITE=%0d,ADDR=%0h,RDATA=%0
h,WDATA=%0d",WRITE,ADDR,RDATA,WDATA);
  endfunction : printA

endclass : My_Packet
module tb;
  BasePacket #(integer) P_integer;
  BasePacket #(int) P_int;

  My_Packet #(4) P2;
initial begin
 P_integer   = new();
 P_int          = new();
  P2             = new();

 P_integer .printA();
 P_int.printA();

end
endmodule
```

Result:

BasePacket::WRITE=1,ADDR=84,RDATA=xxxxxxxx,WDATA=x

BasePacket::WRITE=1,ADDR=84,RDATA=0,WDATA=0

My_Packet::WRITE=1,ADDR=3,RDATA=f,WDATA=0

Object Deallocation

- Garbage collection is the process of automatically freeing objects that are no longer referenced. One-way System Verilog can tell if an object is no longer being used is by keeping track of the number of handles that point to it. When the last handle no longer references an object, System Verilog releases the memory for it.

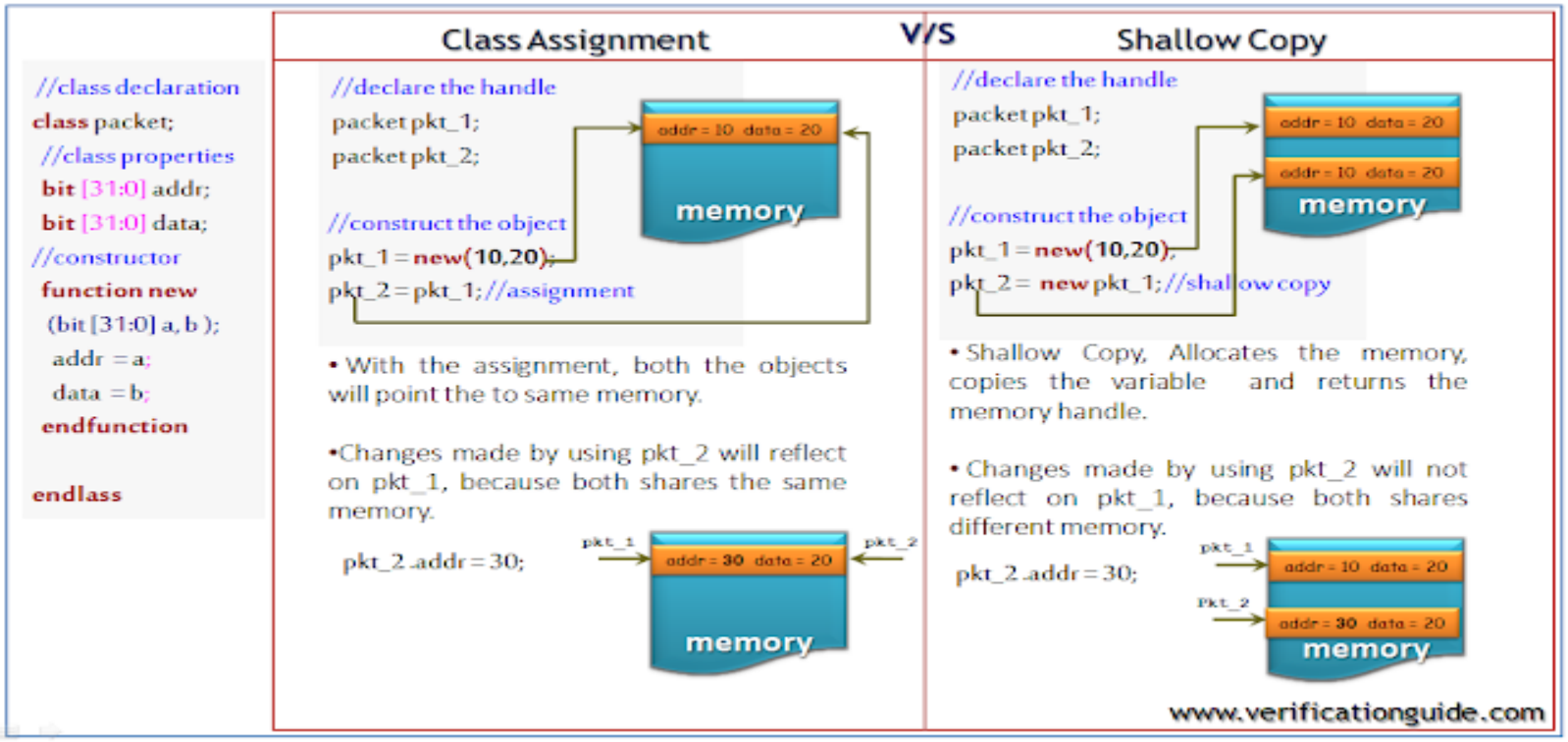
Ex :-

```
Transaction t;   // Create a handle
t = new();       // Allocate a new Transaction
t = new();       // Allocate a second one, free the first
t = null;        // Deallocate the second
```

- The second line in code calls new() to construct an object and store the address in the handle t. The next call to new() constructs a second object and stores its address in t, overwriting the previous value. Since there are no handles pointing to the first object, System Verilog can deallocate it. The object may be deleted immediately or wait for short time. The last line explicitly clears the handle so that now the second object can be deallocated.

Shallow Copy

- In shallow copy, both instances have different addresses.
- So after copy, it doesn't impact on other copy
- All the properties of the class is duplicated and stored in new memory except the objects of sub-class



Class inside the another class with shallow copy

```
class Packet;
        reg write=1;

        function void display();
                $display ("write=0x%0d", write);
        endfunction
endclass

class M_Packet;
        int     addr=30;
        int     data=40;
  Packet        Pkt=new();


        function void display (string name);
     $display ("\n[%s] addr=0x%0h data=0x%0h write=%0d", name, addr, data, Pkt.write);
        endfunction
endclass

module tb;
         M_Packet M_P1,M_P2;
        initial begin
                // Create a new pkt object called p1
                M_P1 = new ();
                M_P1.display ("M_P1");

// Shallow copy M_P1 into M_P2 ; M_P2 is a new object with contents in M_P1
                 M_P2 = new M_P1;
                 M_P2.display (" M_P2 ");
                end
endmodule
```

Result:

[M_P1] addr=0x1e data=0x28 write=x

[M_P2] addr=0x1e data=0x28 write=x

Deep Copy

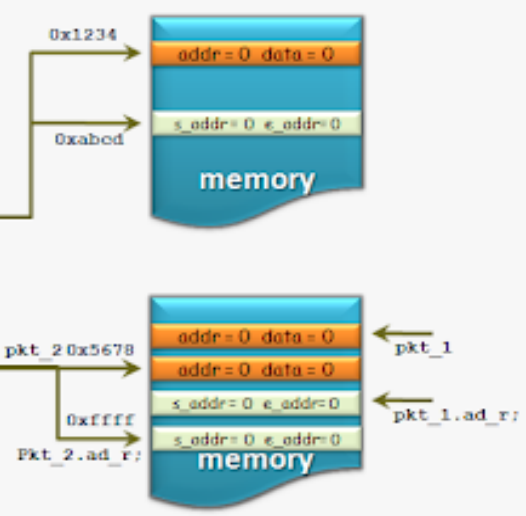
- Unlike shallow copy, it copies all the properties including the object of the sub-class
- There is no keyword to perform deep copy directly. The user must write own methods to achieve it

Deep Copy:

```
class address_range;
  bit [31:0] s_addr;
  bit [31:0] e_addr ;

//copy method
  function address_range copy;
    copy = new();
    copy.s_addr = this.s_addr;
    copy.e_addr = this.e_addr;
    return copy;
  endfunction
endclass
```

```
class packet;
  //class properties
  bit [31:0] addr;
  bit [31:0] data;
  address_range  ad_r;

//constructor
  function new();
    //creating object
    ad_r = new();
  endfunction

//copy method
  function packet copy();
    copy = new();
    copy.addr = this.addr;
    copy.data = this.data;
    copy.ad_r  = ad_r.copy;
    return copy;
  endfunction
endlass
```

```
//declare the handle's
packet pkt_1;
packet pkt_2;

//construct the object
pkt_1 = new();

//Deep copy,
pkt_2 = pkt_1. copy();
```



User has to write the copy method for deep copy.

On calling the copy method, new memory will be allocated and the variable values will be copied to new object and also it call's the copy method of handle inside the object and returns the new object handle.

www.verificationguide.com

```
class Packet;
  reg write=1;

  function Packet P_copy();
    P_copy=new();
  P_copy.write=write;
  endfunction

endclass

class M_Packet;
                    int             addr=30;
                    int             data=40;
  Packet            Pkt=new();

  function M_Packet M_copy();
      M_copy=new();
                    M_copy.addr=addr;
                    M_copy.data=data;
    M_copy.Pkt=Pkt.P_copy();
    endfunction
endclass
```

Result

```
----------Constructing the M_P1 ---------
M_P1::::addr=30 data=40 write=1
----------Copying the M_P1 content to M_P2 using deep copy
approach---------
M_P2::::addr=30 data=40 write=1
----------Modifing the M_P2 Content---------
M_P1::::addr=30 data=40 write=1
M_P2::::addr=55 data=12 write=0
```

```
module tb;
        M_Packet M_P1,M_P2;
       initial begin

    //----------Constructing the M_P1 ---------//
         M_P1 = new ();
    $display("----------Constructing the M_P1 ---------");
    $display("M_P1::::addr=%0d data=%0d write=%b",M_P1.addr,M_P1.data,M_P1.Pkt.write);

    //----------Copying the M_P1 content to M_P2 using deep copy approach---------//
    M_P2 = M_P1.M_copy();
    $display("----------Copying the M_P1 content to M_P2 using deep copy approach---------");
    $display("M_P2::::addr=%0d data=%0d write=%b",M_P2.addr,M_P2.data,M_P2.Pkt.write);

    //----------Modifing the M_P2 Content---------//
    M_P2.addr=55;
     M_P2.data=12;
    M_P2.Pkt.write=0;
    $display("----------Modifing the M_P2 Content---------");
    $display("M_P1::::addr=%0d data=%0d write=%b",M_P1.addr,M_P1.data,M_P1.Pkt.write);
         $display("M_P2::::addr=%0d data=%0d write=%b",M_P2.addr,M_P2.data,M_P2.Pkt.write);

       end
endmodule
```

KNOWING IS NOT ENOUGH;
WE MUST APPLY.

WILLING IS NOT ENOUGH;
WE MUST DO

by ***Bruce lee***

THANK YOU

Class scope resolution operator

The class scope operator '::' is used to specify an identifier defined within the scope of a class.

Ex :-

```
class Base;
    typedef enum {bin,oct,dec,hex} radix;
    static task print( radix r, integer n ); ... endtask
endclass
Base b = new;
int bin = 123;
b.print( Base::bin, bin );          // Base::bin and bin are different
Base::print( Base::hex, 66 );
```

The scope resolution operator enables

- Access to static public members (methods and class properties) from outside the class hierarchy.
- Access to public or protected class members of a super class from within the derived classes.
- Access to type declarations and enumeration named constants declared inside the class from outside the class hierarchy or from within derived classes.

Ex : - From the previous example it can be

```
Base::print( Base::bin, bin );        // Base::bin and bin are different
```

Contd..

```
class base;
    typedef enum {bin,hex} radix;
endclass
class ext extends base;
    typedef enum {hex,bin} radix;
    task print();
     $display(" Ext  classs :: %d %d ",hex,bin);
     $display(" Base classs :: %d %d ",base::hex,base::bin);
    endtask
endclass
```

```
program main ;
  ext e;
  initial
    begin
      e = new();
      e.print();
    end
endprogram
```

Output:- Ext classs :: 0 1

Base classs :: 1 0

Extern with scope resolution

- Definition of method can be written outside the body of class.
- To do this, need to declare the method (Function/Task) with *extern* qualifier along with
  a) any qualifiers (local, protected or virtual)
  b) full argument list.
- The *extern* qualifier indicates that the body of the method (its implementation) is to be found outside the class declaration.

```
class packet;
  bit [31:0] addr;
  bit [31:0] data;
  //function declaration - extern
  extern virtual function void display();
endclass
//function implementation outside class
function void packet::display();
  $display("Addr = %0d Data =
%0d",addr,data);
endfunction
initial begin
    packet p = new();
    p.addr = 10; p.data = 20;
    p.display();
  end

Output:Addr = 10 Data = 20
```