

Introduction to OOP

- **Object-Oriented Programming** (OOP) is a programming paradigm that organizes software design around objects, rather than functions or logic.
- An object is a real-world entity or concept that has two main parts:
 - **Attributes** (data/fields) – the characteristics or properties of the object.
 - **Methods** (functions/behavior) – the actions the object can perform.
- OOP helps in modularizing code, making it more reusable, scalable, and maintainable.

Key Features of OOP:

- **Encapsulation** – bundling data and methods together, restricting direct access to some components.
- **Abstraction** – hiding implementation details and showing only the essential features.
- **Inheritance** – creating new classes based on existing ones, reusing code.
- **Polymorphism** – the ability to take many forms (e.g., the same method name behaving differently in different classes).

Classes and Objects

- **class** is a user-defined datatype, an OOP construct, that can be used to encapsulate data (property) and tasks/functions (methods) which operate on the data.

Example

```
class myPacket;  
    bit [2:0] header;  
    bit      encode;  
    bit [2:0] mode;  
    bit [7:0] data;  
    bit      stop;  
function new (bit [2:0] header = 3'h1, bit [2:0] mode = 5);  
    this.header = header;  
    this.encode = 0;  
    this.mode   = mode;  
    this.stop   = 1;  
endfunction
```

```
function display ();  
$display ("Header = 0x%0h, Encode = %0b, Mode =  
0x%0h, Stop = %0b",this.header, this.encode,  
this.mode, this.stop);  
endfunction  
endclass
```

function new () is called the constructor and is automatically called upon object creation.

this keyword is used to refer to the current class. Normally used within a class to refer to its own properties/methods.

display () is a function, and rightly so, because displaying values does not consume simulation time.

function new () has default values to the arguments, and hence create a packet object with values [3'h1, 0, 2'h3, 1]

Array of classes

```
module tb_top;  
    myPacket pkt0 [3];  
  
    initial begin  
        for(int i = 0; i < $size (pkt0); i++) begin  
            pkt0[i] = new ();  
            pkt0[i].display ();  
        end  
    end  
endmodule
```

Simulation Results

Header = 0x1, Encode = 0, Mode = 0x5, Stop = 1

Header = 0x1, Encode = 0, Mode = 0x5, Stop = 1

Header = 0x1, Encode = 0, Mode = 0x5, Stop = 1

An Array of Objects

```
class transaction;  
    bit [31:0] data;  
    int id;  
endclass  
  
module class_example;  
    transaction tr[5];  
    initial begin  
        foreach(tr[i]) begin  
            tr[i] = new();  
            tr[i].data = i*i;  
            tr[i].id = i+1;  
        end  
  
        foreach(tr[i]) $display("tr[%0d].data = %0d, tr[%0d].id =  
%0d",i, tr[i].data, i, tr[i].id);  
    end  
endmodule
```

Output

```
tr[0].data = 0, tr[0].id = 1  
tr[1].data = 1, tr[1].id = 2  
tr[2].data = 4, tr[2].id = 3  
tr[3].data = 9, tr[3].id = 4  
tr[4].data = 16, tr[4].id = 5
```

Inheritance in SV

- **Inheritance** is a concept in OOP that allows us to extend a class to create another class and have access to all the properties and methods of the original parent class from the handle of a new class object.
- The idea behind this scheme is to allow developers add in new properties and methods into the new class while still maintaining access to the original class members.
- This allows us to make modifications without touching the base class at all.

Example:

```
class parent_trans;
    bit [31:0] data;

    function void disp_p();
        $display("Value of data = %0h", data);
    endfunction
endclass

class child_trans extends parent_trans;
    int id;

    function void disp_c();
        $display("Value of id = %0h", id);
    endfunction
endclass
```

```
module class_example;
    initial begin
        child_trans c_tr;
        c_tr = new();
        c_tr.data = 5; // child class is updating
        //property of its base class
        c_tr.id = 1;

        c_tr.disp_p(); // child class is accessing
        //method of its base class
        c_tr.disp_c();
    end
endmodule
```

Result:

```
Value of data = 5
Value of id = 1
```

'this' keyword in SV

- The '**this**' keyword is used to refer to class properties, parameters and methods of the current instance. It can only be used within non-static methods, constraints and covergroups.
- **this** is basically a pre-defined object handle that refers to the object that was used to invoke the method in which **this** is used.

A very common way of using **this** is within the initialization block.

```
class Packet;
    bit [31:0] addr;

    function new (bit [31:0] addr);
//          addr = addr;          // Which addr should get assigned ?

        this.addr = addr;        // addr variable in Packet class should be
//          assigned with local variable addr in new()

    endfunction
endclass
```

'super' keyword in SV

- The **super** keyword is used from within a sub-class to refer to properties and methods of the base class.
- It is mandatory to use the **super** keyword to access properties and methods if they have been overridden by the sub-class.

```
class Packet;  
    int addr;  
    function display ();  
        $display ("[Base] addr=0x%0h", addr);  
    endfunction  
endclass
```

```
class extPacket extends Packet;    // extPacket is a child class of Packet  
    function new ();  
        super.new ();  
    endfunction  
endclass
```

```
module tb;  
    Packet p;  
    extPacket ep;  
  
    initial begin  
        ep = new();  
        p = new();  
        p.display();  
    end  
endmodule
```

Result
[Base] addr=0x0

Inheritance Vs Composition

Feature	Inheritance	Composition
Relationship	IS-A (child class is a type of parent)	HAS-A (class contains other class objects)
Mechanism	extends keyword	Class members are instances of other classes
Code Reuse	Reuses parent's properties & methods automatically	Reuses functionality by combining objects
Flexibility	Less flexible — tightly coupled to parent	More flexible — objects can be swapped or replaced
Example in SV	<code>class my_driver extends uvm_driver;</code>	<code>class agent; driver d; monitor m; endclass</code>
When to use	To create specialized versions of a base component (e.g., custom transaction, driver)	To build larger components from smaller ones (e.g., agent has driver + monitor + sequencer)
UVM Usage	Customizing UVM base classes (uvm_driver, uvm_monitor, uvm_sequence)	Building environment hierarchy (env has agents, agents have drivers/monitors)

Composition in SystemVerilog

What it is: Building a class by including objects of other classes inside it (HAS-A relationship).

How it works: Instead of extending, you create instances of other classes inside your class.

When used:

To combine multiple independent components.

Example: An environment class that contains driver, monitor, and scoreboard objects.

```
class agent;  
    driver  d; // HAS a driver  
    monitor m; // HAS a monitor  
  
    function new();  
        d = new();  
        m = new();  
    endfunction  
endclass
```