**Imagine you're designing a smartphone 📱.**

* **You have dozens of phones with different colors, RAM sizes, battery levels, but the same basic structure — screen, camera, processor.**
* **It would be crazy to design each phone separately, right?**

**Instead, you create a blueprint (CLASS),  
and then make individual phones (OBJECTS) based on it.**

**CLASS IN SYSTEMVERILOG**

A class in SystemVerilog is a blueprint for creating objects. Classes are widely used in testbenches for verification, enabling randomized stimulus generation, coverage tracking, and transaction modeling.

class Packet; // Class = Blueprint

bit [7:0] addr;

bit [31:0] data;

endclass

Packet p1; // p1 = Handle (pointer)

p1 = new(); // Now p1 is a real packet in memory!

p1.addr = 8'hA5;

p1.data = 32'h1234\_5678;

🛠️ Class Components

* **Data Members (Variables):** These hold the **state** of the object.

class Packet;

bit [7:0] addr;

bit [31:0] data;

endclass

addr and data are *Data Members*, Each object has its *own copy*.

* **Methods (Functions and Tasks):** These describe the **behavior** (what the object can *do*)

class Packet;

bit [7:0] addr;

bit [31:0] data;

function void display();

$display("Addr = %0h, Data = %0h", addr, data);

endfunction

endclass

display() is a **Function** inside class. Methods can **read/write** data members.

* **Object Creation (new()):** SystemVerilog uses new() keyword to create a real object.

Packet p1; // p1 = Handle (pointer)

p1 = new(); // Now real memory is allocated!

* **Constructor (new() Function):** Used to initialize the class when it's created.

class Packet;

function new(bit [7:0] a, bit [31:0] d); //constructor

addr = a;

data = d;

endfunction

endclass

Packet p1 = new(8'hAB, 32'hDEADBEEF); //initializing the class

www.linkedin.com/in/harshita-harshi-9377891bb

**🎓Properties of a SystemVerilog Class**

* **Encapsulation**

Bundling data + methods together inside a class.

✅ Keeps internal details hidden.  
 ✅ Promotes modular design.  
 ✅ Great for debugging and reuse**.**

Access Modifiers: Control *who* can access the class members.

| **Modifier** | **Meaning** |
| --- | --- |
| public | Can be accessed from anywhere **(default)** |
| local | Private to the class(only access to class) |
| protected | Accessible in the class and child class |

class BankAccount;

local int balance; // Encapsulated data

endclass

* **Inheritance**

One class can extend another and reuse its properties and methods.

class Animal;

function void speak(); $display("Generic sound"); endfunction

endclass

class Dog extends Animal;

function void speak(); $display("Bark"); endfunction

endclass

* **Polymorphism**

Different classes can have the same method name but different behaviors.

Animal a;

a = new Dog(); // Dog method will be called

a.speak(); // Output: Bark

**🧩 Main Properties of a Class**

|  |  |  |
| --- | --- | --- |
| Property | Meaning | Simple Analogy |
| Data Members | Variables inside class | Phone’s color, battery size |
| Methods | Functions/tasks inside class | How the phone makes calls, sends SMS |
| Encapsulation | Bundling data + methods together | A smartphone has apps + settings inside |
| Dynamic Nature | Objects can be created/destroyed anytime | New phones can be made, old ones discarded |

www.linkedin.com/in/harshita-harshi-9377891bb

* **Static Properties and Static Methods**
* Belong to the **class itself**, not individual objects.
* Useful for shared counters, configuration, etc.
* No need to instantiate to access static properties.

class Packet;

static int packet\_count = 0;

function new();

packet\_count++;

endfunction

endclass

* **Virtual Methods and Virtual Classes**

Allow **polymorphism** (runtime method overriding).

* Virtual class = abstract base class (can't be instantiated)
* Virtual function = can be overridden by child
* Pure virtual function = **must** be implemented by child systemverilog

virtual class Animal;

pure virtual function void speak();

endclass

class Dog extends Animal;

function void speak();

$display("Bark");

endfunction

endclass

class Cat extends Animal;

function void speak();

$display("Meow");

endfunction

endclass

// Usage

Animal a;

a = new Dog();

a.speak(); // Bark

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | Static | Virtual | Inheritance | Override | Constructor |
| Belongs to class, not object | ✅ | ❌ | ❌ | ❌ | ❌ |
| Supports polymorphism | ❌ | ✅ | ✅ | ✅ | ❌ |
| Used for initializing object | ❌ | ❌ | ❌ | ❌ | ✅ |

www.linkedin.com/in/harshita-harshi-9377891bb

* **Super keyword**
* To **call parent class functions** inside the child class.
* Useful when you want to **extend** parent behavior, not replace it.

class Parent;

function void greet();

$display("Hello from Parent");

endfunction

endclass

class Child extends Parent;

function void greet();

super.greet(); // Call parent greet first

$display("Hello from Child");

endfunction

endclass

|  |  |
| --- | --- |
| **Shallow Copy**  When you assign one object handle to another, **both point to the same memory**.  Changing one object will **automatically affect the other**.  class Packet;  int id;  endclass  Packet p1 = new();  p1.id = 5;  Packet p2 = p1; // Shallow copy (both p1 and p2 point to same object)  p2.id = 10;  $display("p1.id = %0d", p1.id); // Output: 10 (changed!) | **Deep Copy**  **Create a new object** and **manually copy** the contents.  Now, changes in one **do NOT** affect the other.  class Packet;  int id;  // Deep Copy Method  function Packet copy();  Packet tmp = new();  tmp.id = this.id; // Copy data manually  return tmp;  endfunction  endclass  Packet p1 = new();  p1.id = 5;  Packet p2 = p1.copy(); // Deep copy  p2.id = 10;  $display("p1.id = %0d", p1.id); // Output: 5 (not changed)  $display("p2.id = %0d", p2.id); // Output: 10 |

www.linkedin.com/in/harshita-harshi-9377891bb