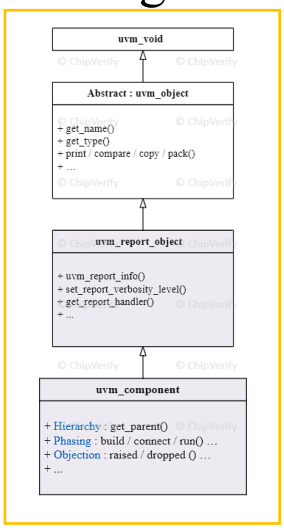
DAY-95
#100DAYSRTL

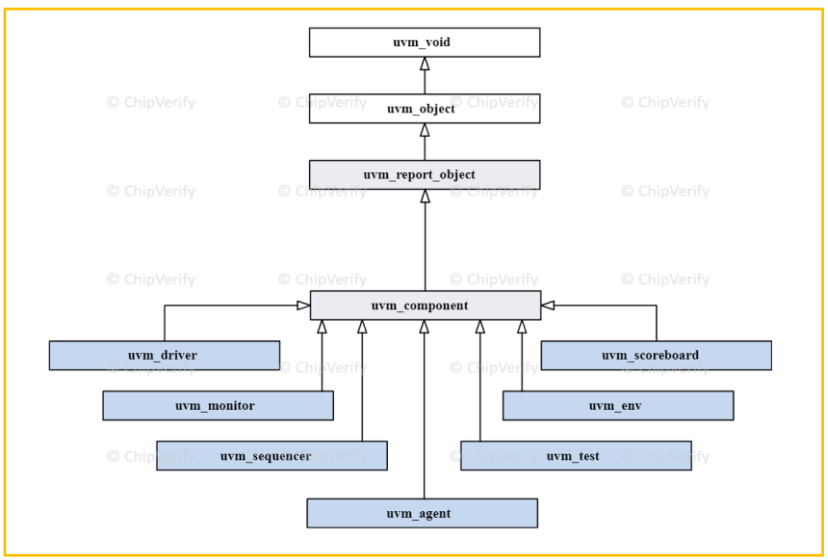
"UVM: UVM Component(Part-1)"

uvm_component is a fundamental base class that serves as the foundation for all UVM components like drivers, monitors and scoreboards in a verification environment. It has the following features:

- **Hierarchy:-**

  Supports a hierarchical structure, where each component can have child components, forming a tree-like structure and provides methods for searching and traversing the tree.





- **Phasing:**
  Components can participate in the UVM phasing mechanism, which organizes the simulation into different phases like build, connect, run, and cleanup. Components can perform specific tasks during each phase.
- **Reporting:-**
  Components can use the UVM messaging infrastructure to report events, warnings, and errors during simulation.
- **Factory:-**
  Components can be registered with the UVM factory mechanism, enabling dynamic object creation and lookup

"Code Practising":-

```
// File: my_component.sv
// Include UVM library
`include "uvm_macros.svh"
`include "uvm_pkg.sv"
// Define your component as a UVM agent
class my_component extends uvm_agent;
  // Declare your component-specific variables and components here
  // For example, you may have interface handles, monitors, drivers, etc.
  // Define the constructor for your component
  function new(string name = "my_component", uvm_component parent = null);
    super.new(name, parent);
  endfunction
  // Define the build_phase to configure and create sub-components
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    // Create and configure your sub-components (e.g., driver, monitor, sequencer)
    // Example:
    // my_driver driver_inst;
    // my_monitor monitor_inst;
    // my_sequencer sequencer_inst;
    // Set the connections between components if needed
  endfunction
  // Define the connect_phase to establish connections between components
  virtual function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    // Connect the components, if not done in build_phase
  endfunction
  // Define the run_phase for the main functionality of your component
  virtual task run_phase(uvm_phase phase);
    // Implement the main functionality of your component here
    // This could include generating stimulus, driving the DUT, monitoring signals, etc.
    // Example:
    // my_sequencer.start(env_config.sequencer_cfg);
    // my_sequencer.wait_for_sequences();
  endtask
  // Optionally, you can implement other UVM phases and tasks as needed
endclass
```