1. Classes, Objects, and Handles

A class is a user-defined data type that groups variables and methods.
An object is a runtime instance of a class, created dynamically.
A handle is a reference used to access the object in memory.

```
class Packet1;
  int addr;   // data member: address
  int data;   // data member: data value
endclass

module tb_packet;
  Packet1 p;  // handle declaration
  initial begin
    p = new();        // object creation
    p.addr = 10;     // assign address
    p.data = 20;     // assign data
    $finish;
  end
endmodule
```

2. Creating New Object (Constructor)

A constructor is a special function named `new()` used for initialization.
It is called automatically when an object is created.
Constructors help ensure objects start in a known state.

```
class Counter;
  int count;           // instance variable
  function new(int init = 0);
    count = init;      // initialize variable
  endfunction
endclass
```

```
module tb_constructor;
  Counter c;            // handle declaration
  initial begin
    c = new(5);         // constructor call with argument
  end
endmodule
```

3. Object Deallocation

Object deallocation is managed automatically by the simulator.
When no handle references an object, it becomes eligible for garbage collection.
Users release objects by assigning handles to `null`.

```
class Packet2;
  int id;               // packet identifier
endclass

module tb_dealloc;
  Packet2 p;            // handle declaration
  initial begin
    p = new();          // object creation
    p = null;           // release handle (eligible for GC)
    $finish;
  end
endmodule
```

4. Inheritance

Inheritance allows a child class to reuse properties of a base class.
The child class can override base-class methods.
It represents an *is-a* relationship.

```
class Base1;
  function void show();
    $display("Base");  // base class method
  endfunction
endclass

class Child1 extends Base1;
  function void show();
    $display("Child"); // overridden method
  endfunction
endclass

module tb_inheritance;
  Child1 c;              // child-class handle
  initial begin
    c = new();           // create child object
    c.show();            // calls child version
    $finish;
  end
endmodule
```

5. Polymorphism

Polymorphism allows a base-class handle to refer to a child-class object.
Method binding is resolved at runtime using `virtual` functions.
It enables flexible and extensible testbench designs.

```
class PolyBase;
  virtual function void show();
    $display("Base");  // base implementation
  endfunction
endclass

class PolyChild extends PolyBase;
  function void show();
    $display("Child"); // overridden implementation
  endfunction
endclass

module tb_poly;
```

```
  PolyBase b;            // base-class handle
  initial begin
    b = new PolyChild(); // base handle → child object
    b.show();            // runtime binding
    $finish;
  end
endmodule
```

6. Composition

Composition represents a *has-a* relationship between classes.
One class contains an object of another class as a member.
It is used to build complex systems from simpler components.

```
class Engine;
  function void start();
    $display("Engine start"); // engine behavior
  endfunction
endclass

class Car;
  Engine e;                 // has-a relationship
  function new();
    e = new();              // create engine object
  endfunction
endclass

module tb_composition;
  Car c;                    // car object
  initial begin
    c = new();
    c.e.start();            // access composed object
  end
endmodule
```

7. Static Variables vs Global Variables

Static Class Variables

Static variables belong to the class, not to individual objects.
They are shared among all instances of the class.
Commonly used for counters and shared resources.

```
class C;
  static int total;   // shared across all objects
  int id;             // instance-specific variable
  function new();
    total++;          // update shared counter
    id = total;       // assign unique ID
  endfunction
endclass

module tb_static;
  C c1, c2;            // multiple objects
  initial begin
    c1 = new();
    c2 = new();
  end
endmodule
```

Global Variables (Package Scope)

Package variables act as global variables.
They are shared by all modules and classes that import the package.
Their lifetime spans the entire simulation.

```
package globals_pkg;
  int g_count;         // global variable
endpackage

module tb_global;
  import globals_pkg::*;
  initial begin
    g_count = 10;      // access global variable
  end
endmodule
```

Comparison summary:

- Static class variable → one per class, shared by all objects
- Package variable → one per package, shared globally