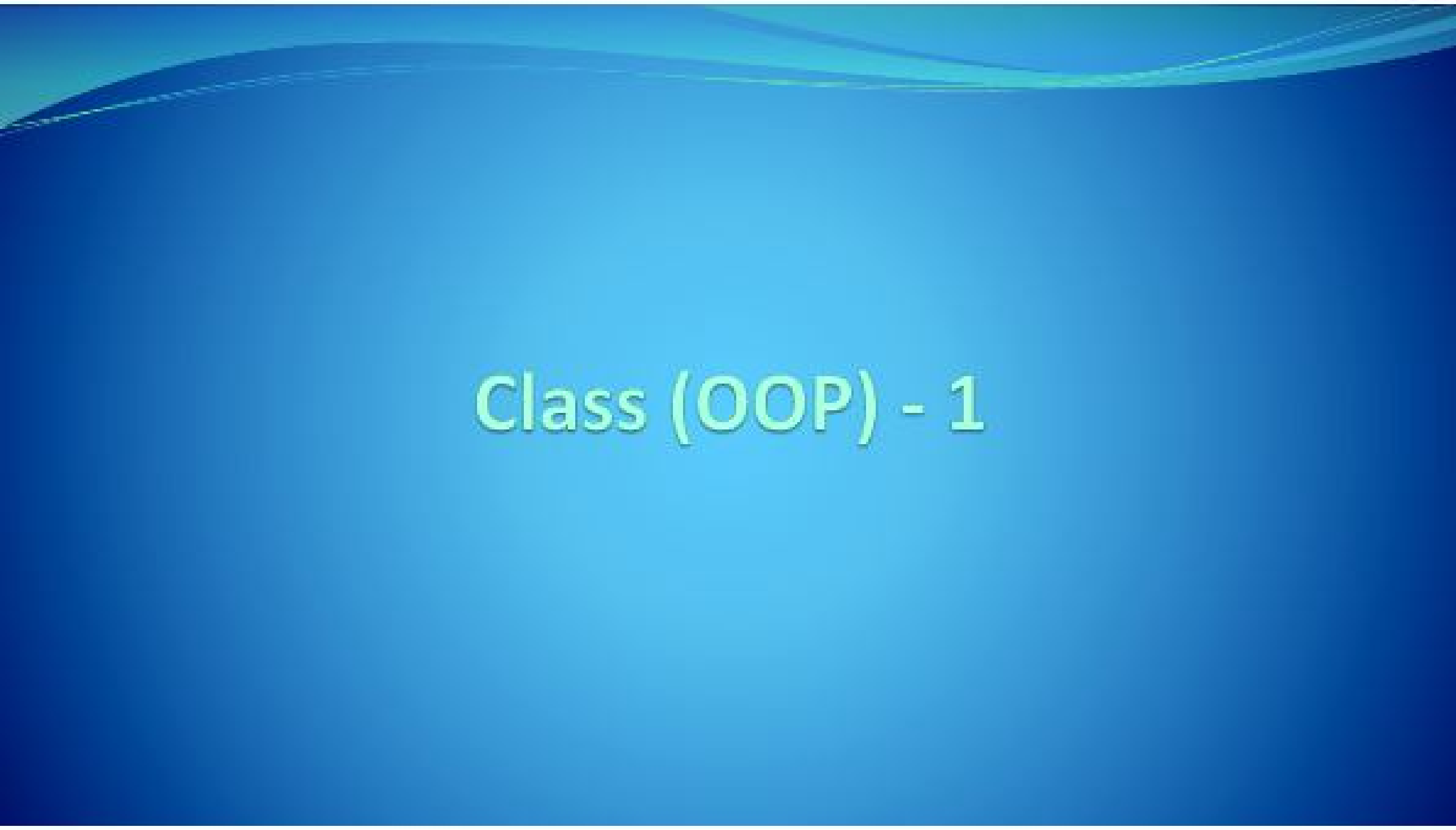
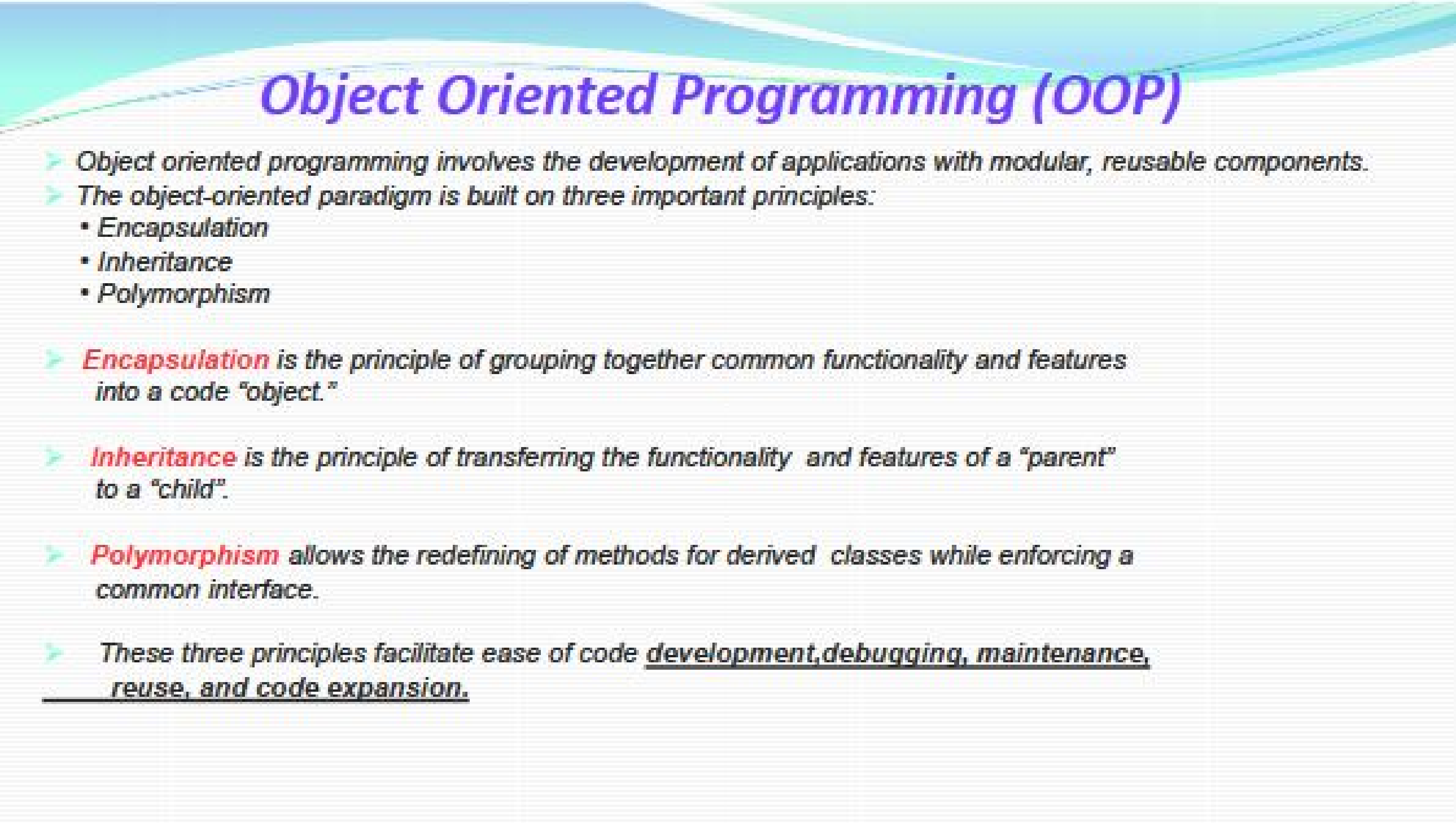


Class (OOP) - 1



Object Oriented Programming (OOP)

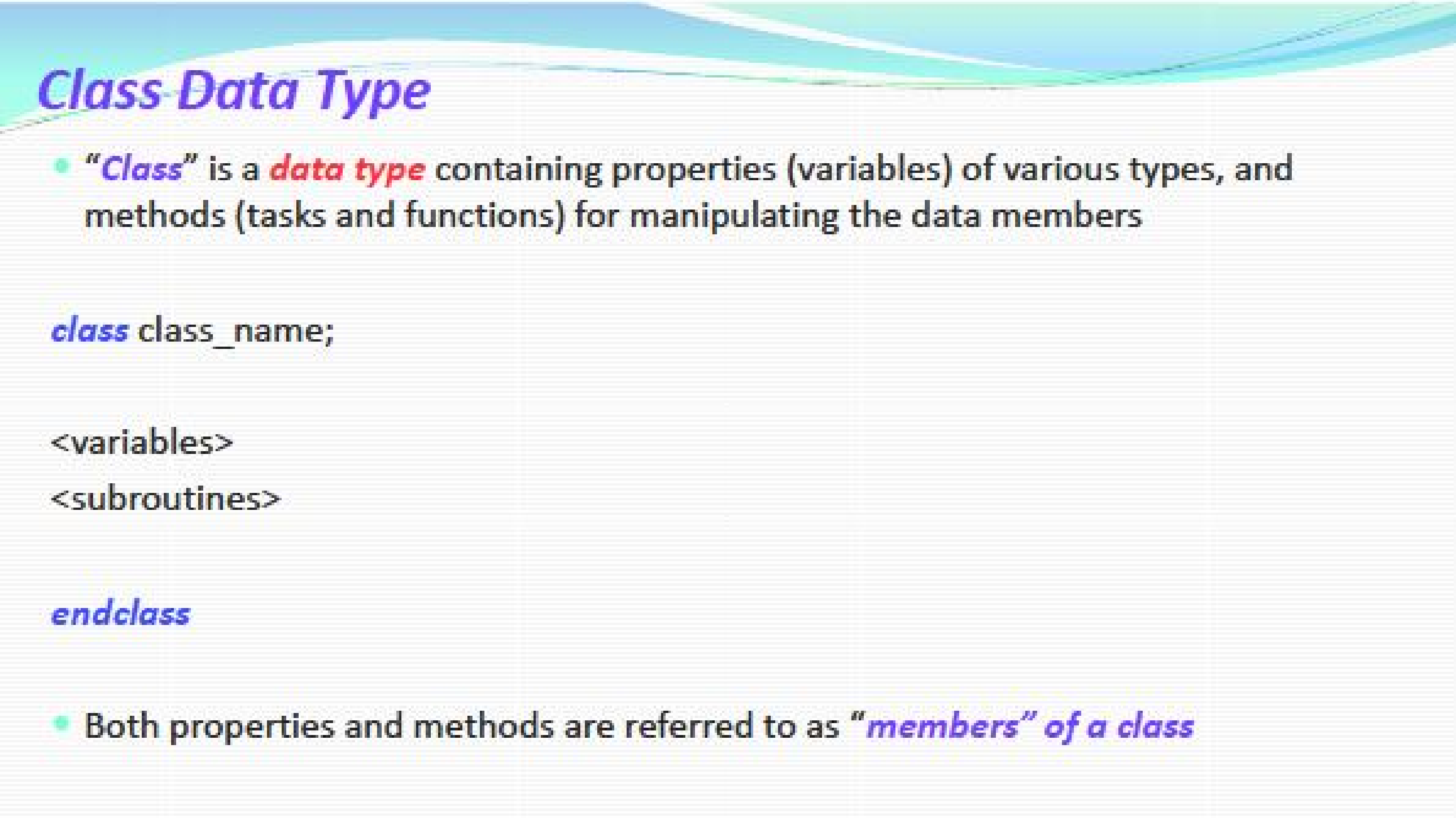
- Object oriented programming involves the development of applications with modular, reusable components.
- The object-oriented paradigm is built on three important principles:
  - Encapsulation
  - Inheritance
  - Polymorphism

- Encapsulation is the principle of grouping together common functionality and features into a code "object."

- Inheritance is the principle of transferring the functionality and features of a "parent" to a "child".

- Polymorphism allows the redefining of methods for derived classes while enforcing a common interface.

- These three principles facilitate ease of code **development,debugging, maintenance, reuse, and code expansion.**

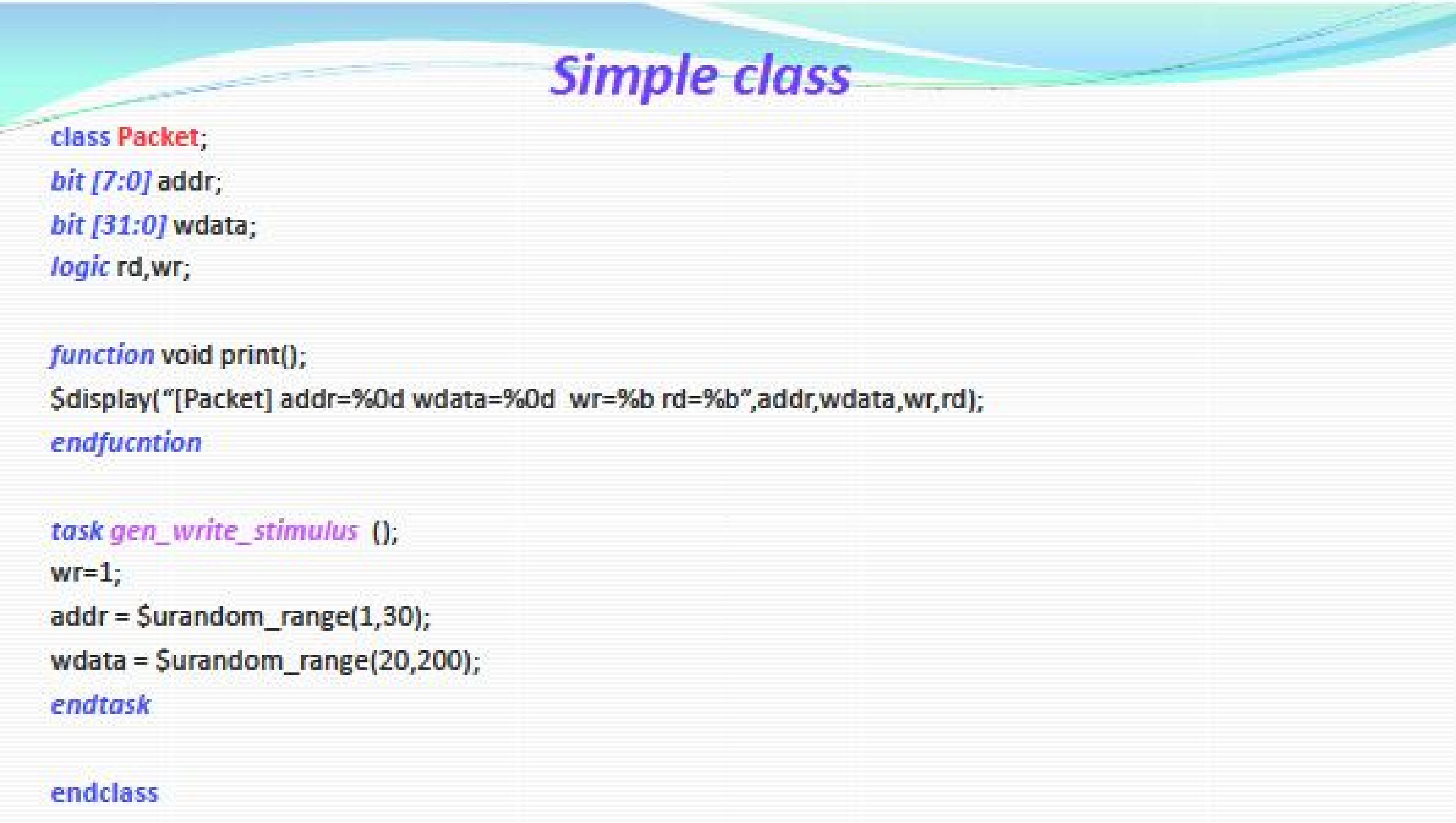


Class Data Type

- "*Class*" is a ***data type*** containing properties (variables) of various types, and methods (tasks and functions) for manipulating the data members

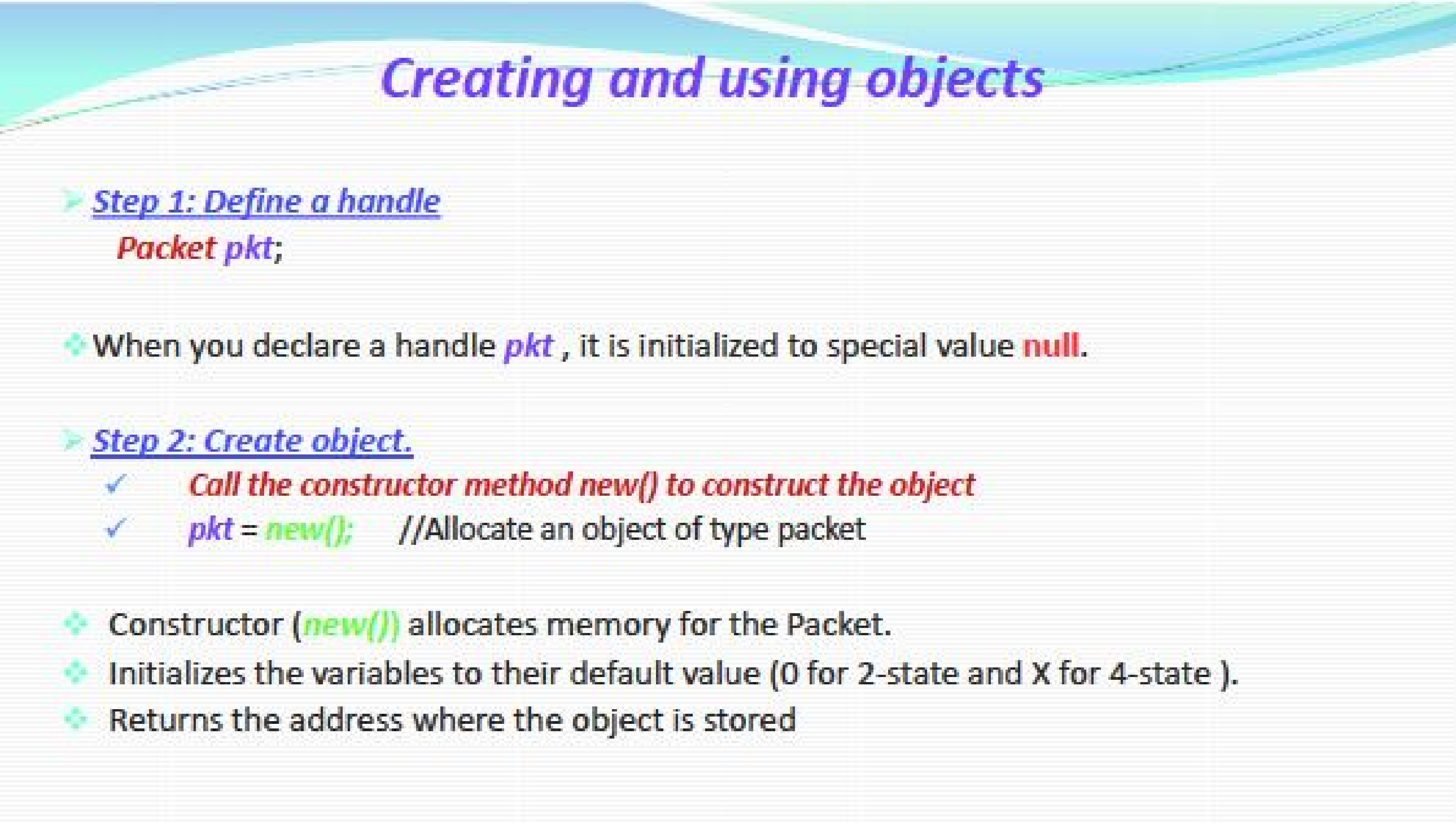
```
class class_name;

<variables>
<subroutines>

endclass
```

- Both properties and methods are referred to as ***"members" of a class***



Simple class

```
class Packet;
bit [7:0] addr;
bit [31:0] wdata;
logic rd,wr;

function void print();
$display(“[Packet] addr=%0d wdata=%0d  wr=%b rd=%b”,addr,wdata,wr,rd);
endfucntion

task gen_write_stimulus  ();
wr=1;
addr = $urandom_range(1,30);
wdata = $urandom_range(20,200);
endtask

endclass
```



Creating and using objects

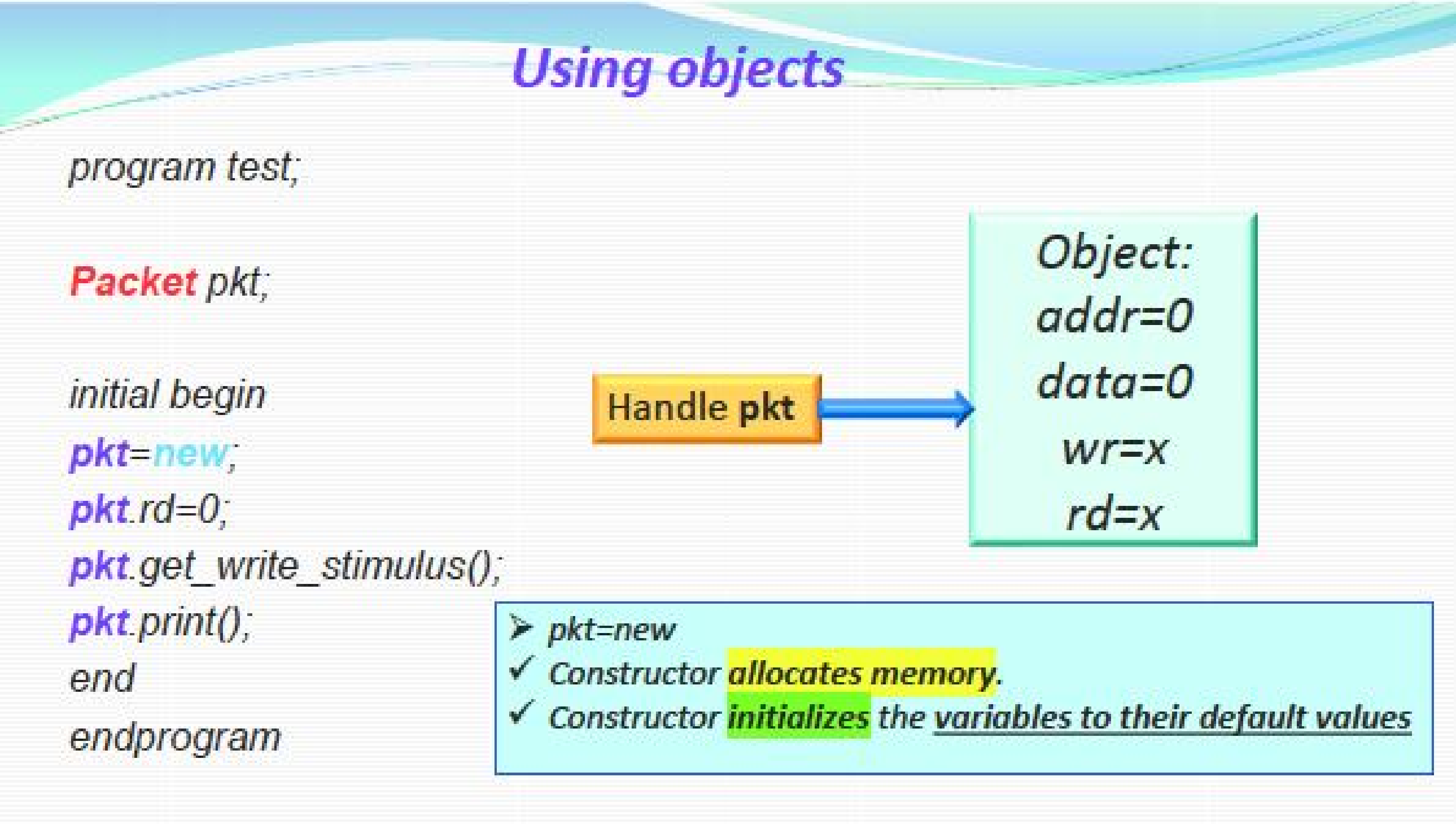
- ***Step 1: Define a handle***

  Packet pkt;

- When you declare a handle ***pkt*** , it is initialized to special value **null**.

- ***Step 2: Create object.***
  - ✓ ***Call the constructor method new() to construct the object***
  - ✓ *pkt* = *new();* //Allocate an object of type packet

- Constructor (*new()*) allocates memory for the Packet.
- Initializes the variables to their default value (0 for 2-state and X for 4-state).
- Returns the address where the object is stored



Using objects

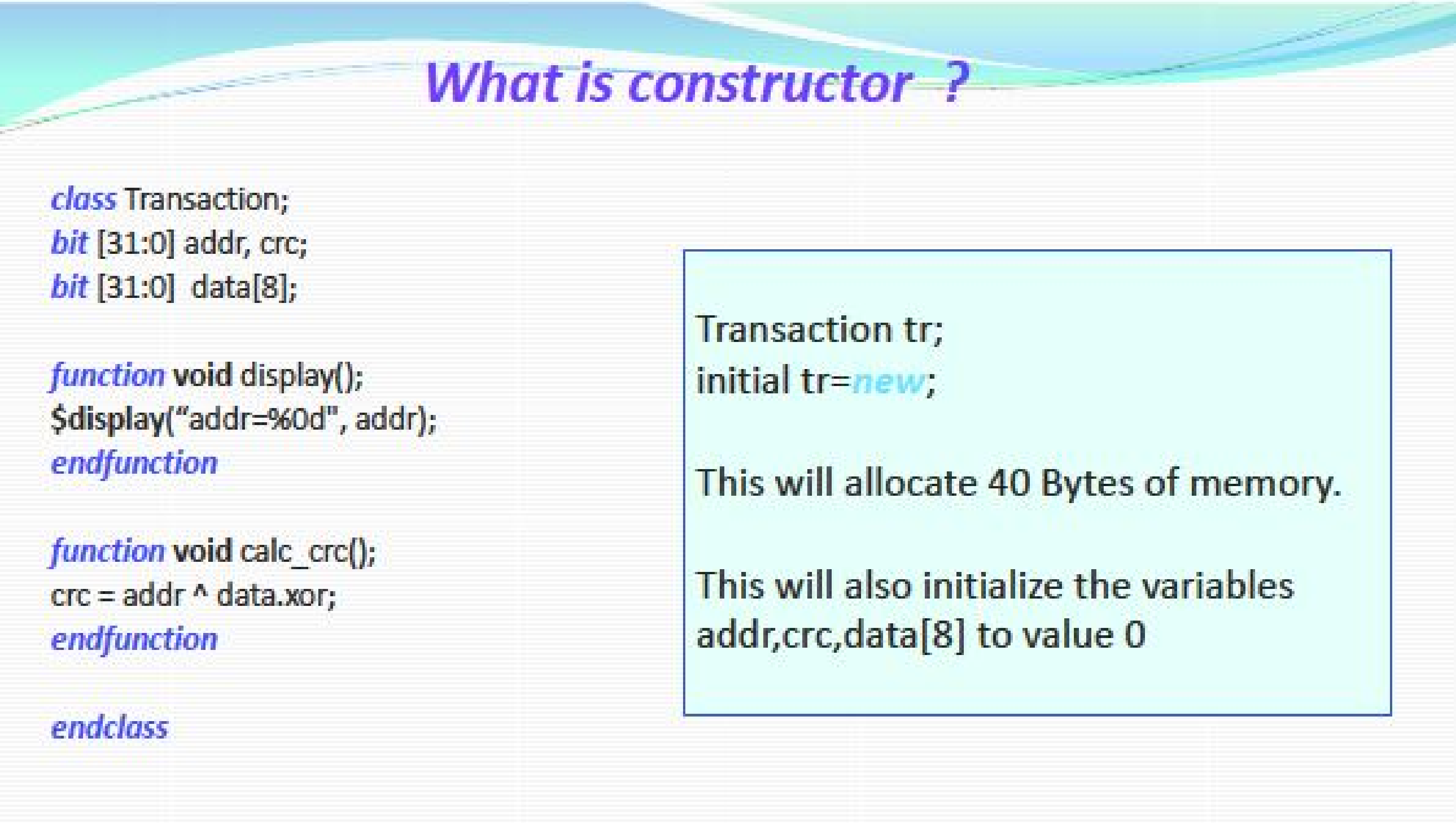
```
program test;

Packet pkt;

initial begin
pkt=new;
pkt.rd=0;
pkt.get_write_stimulus();
pkt.print();
end
endprogram
```

Handle **pkt** → Object:
addr=0
data=0
wr=x
rd=x

➢ pkt=new

✓ Constructor allocates memory.

✓ Constructor initializes the <u>variables to their default values</u>



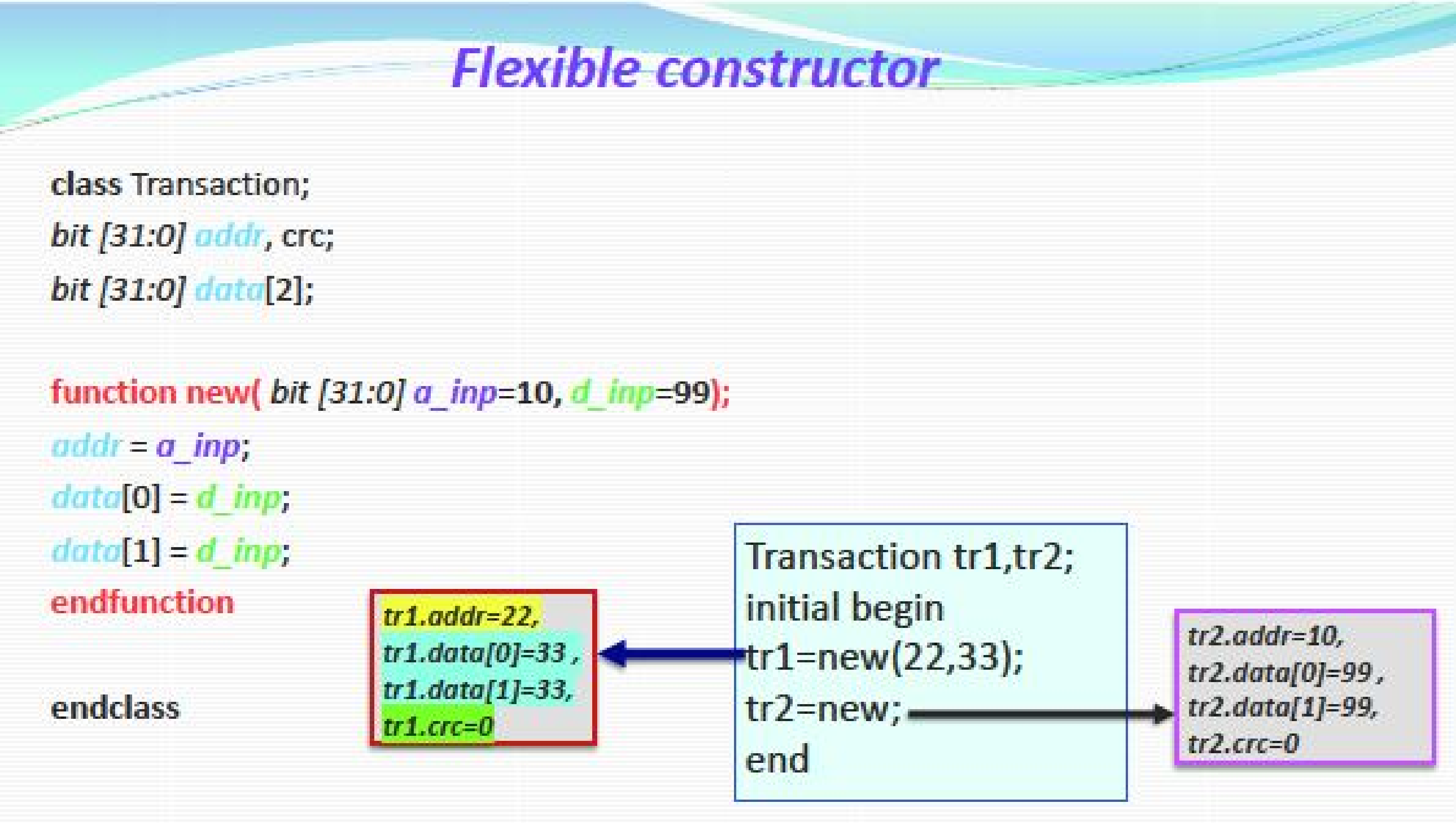
What is constructor ?

```
class Transaction;
bit [31:0] addr, crc;
bit [31:0]  data[8];

function void display();
$display("addr=%0d", addr);
endfunction

function void calc_crc();
crc = addr ^ data.xor;
endfunction

endclass
```

```
Transaction tr;
initial tr=new;
```

This will allocate 40 Bytes of memory.

This will also initialize the variables addr,crc,data[8] to value 0



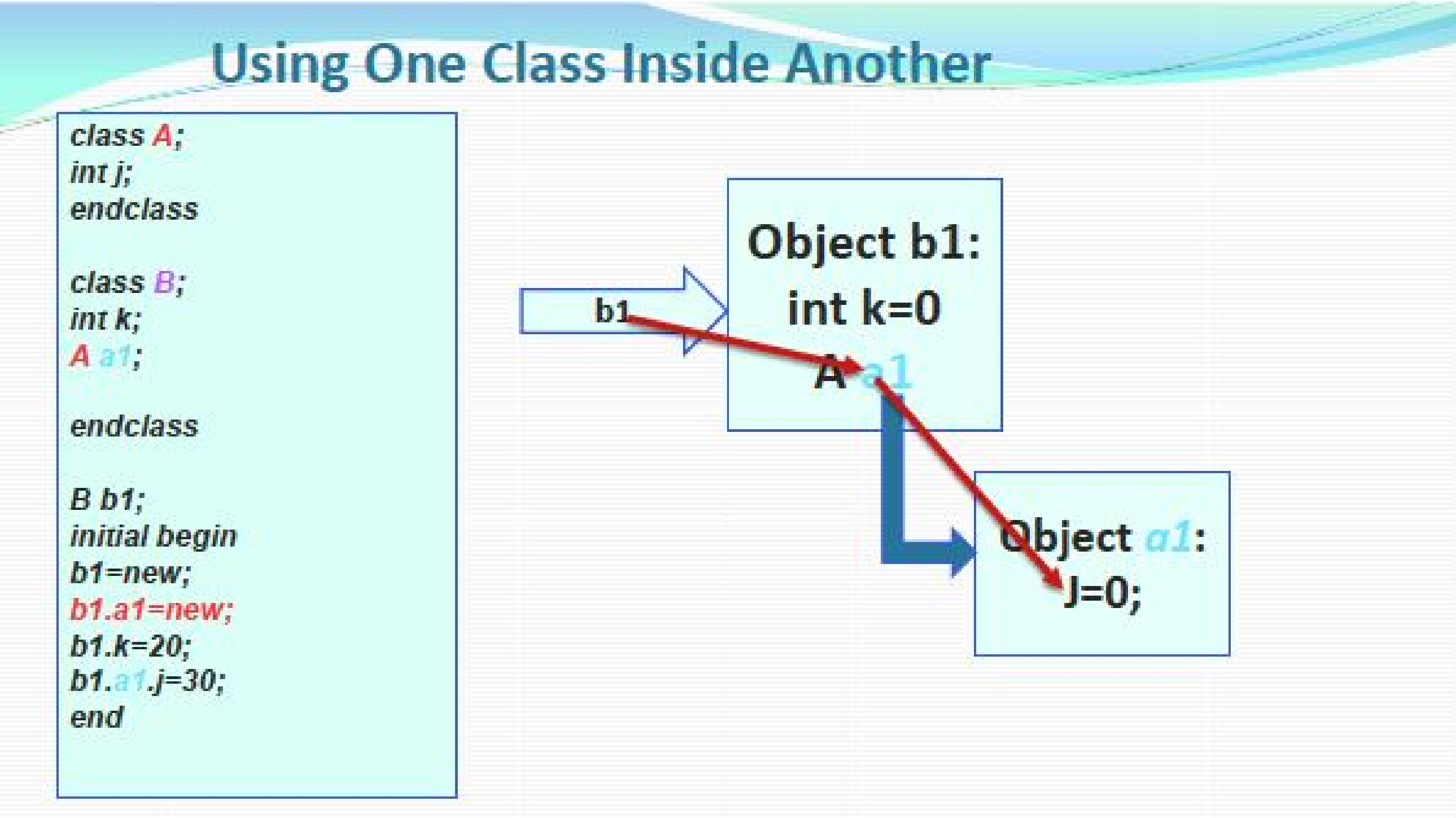
Flexible constructor

```
class Transaction;
bit [31:0] addr, crc;
bit [31:0] data[2];

function new( bit [31:0] a_inp=10, d_inp=99);
addr = a_inp;
data[0] = d_inp;
data[1] = d_inp;
endfunction

endclass
```

```
Transaction tr1,tr2;
initial begin
tr1=new(22,33);
tr2=new;
end
```

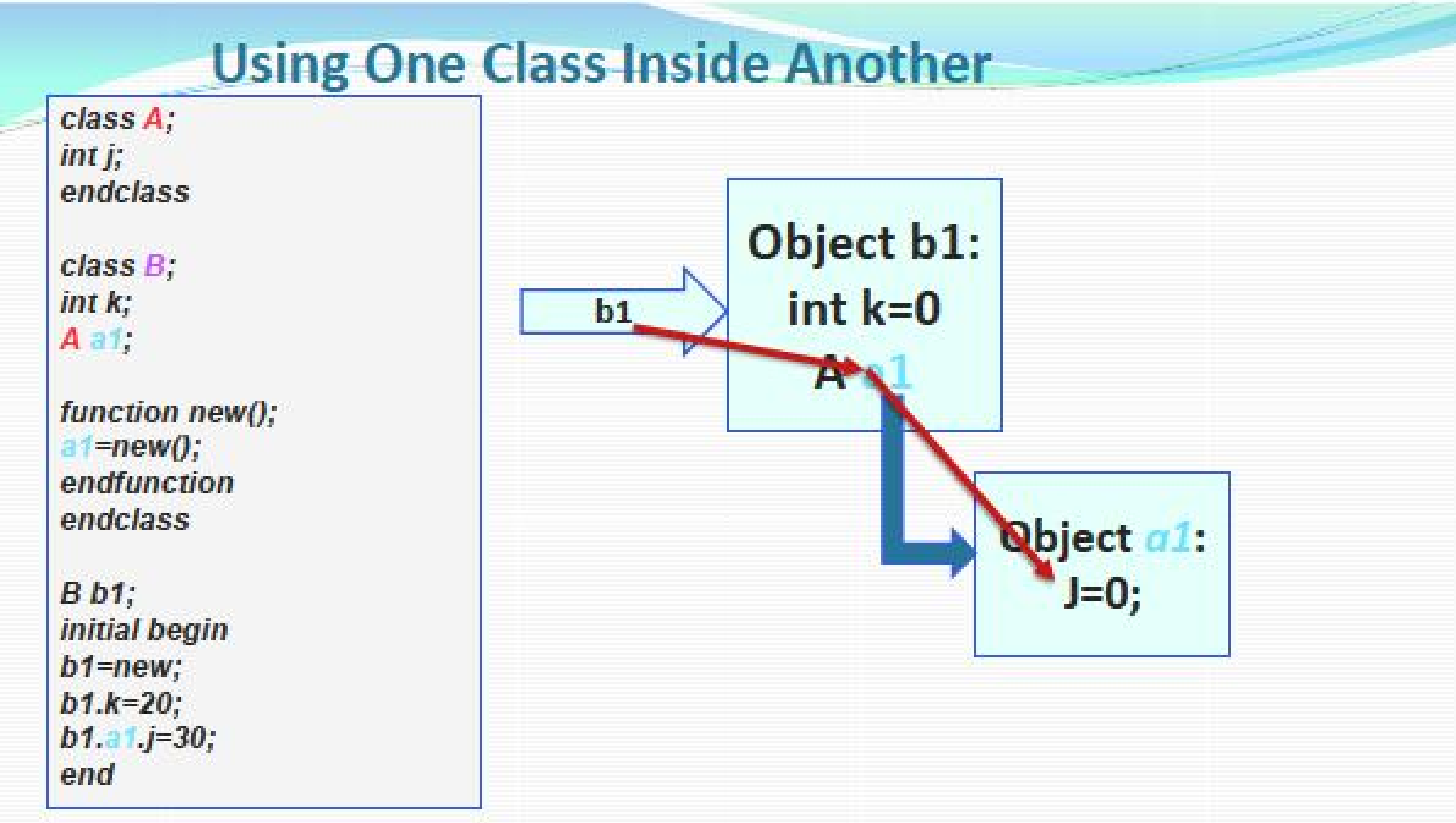
tr1.addr=22,
tr1.data[0]=33 ,
tr1.data[1]=33,
tr1.crc=0

tr2.addr=10,
tr2.data[0]=99 ,
tr2.data[1]=99,
tr2.crc=0



Using One Class Inside Another

```
class A;
int j;
endclass

class B;
int k;
A a1;

endclass

B b1;
initial begin
b1=new;
b1.a1=new;
b1.k=20;
b1.a1.j=30;
end
```



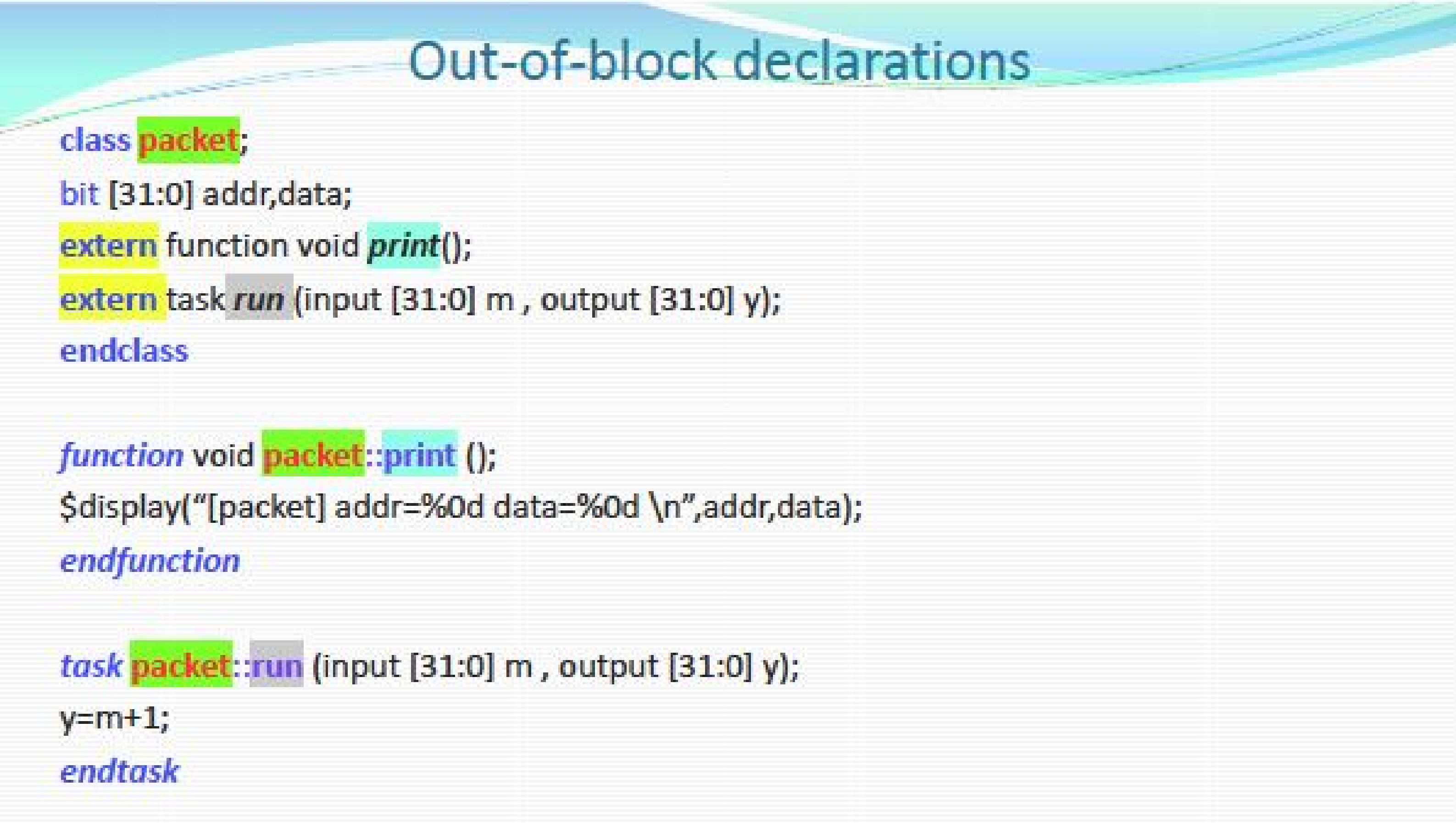
Using One Class Inside Another

```
class A;
int j;
endclass

class B;
int k;
A a1;

function new();
a1=new();
endfunction
endclass

B b1;
initial begin
b1=new;
b1.k=20;
b1.a1.j=30;
end
```

b1

Object b1:
int k=0
A a1

Object a1:
J=0;



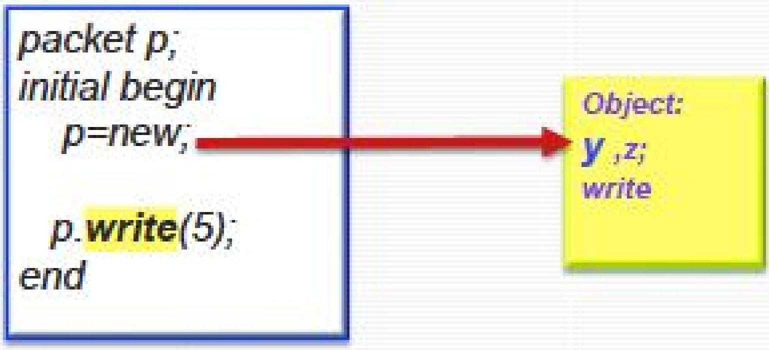
Out-of-block declarations

```
class packet;
bit [31:0] addr,data;
extern function void print();
extern task run (input [31:0] m , output [31:0] y);
endclass

function void packet::print ();
$display("[packet] addr=%0d data=%0d \n",addr,data);
endfunction

task packet::run (input [31:0] m , output [31:0] y);
y=m+1;
endtask
```

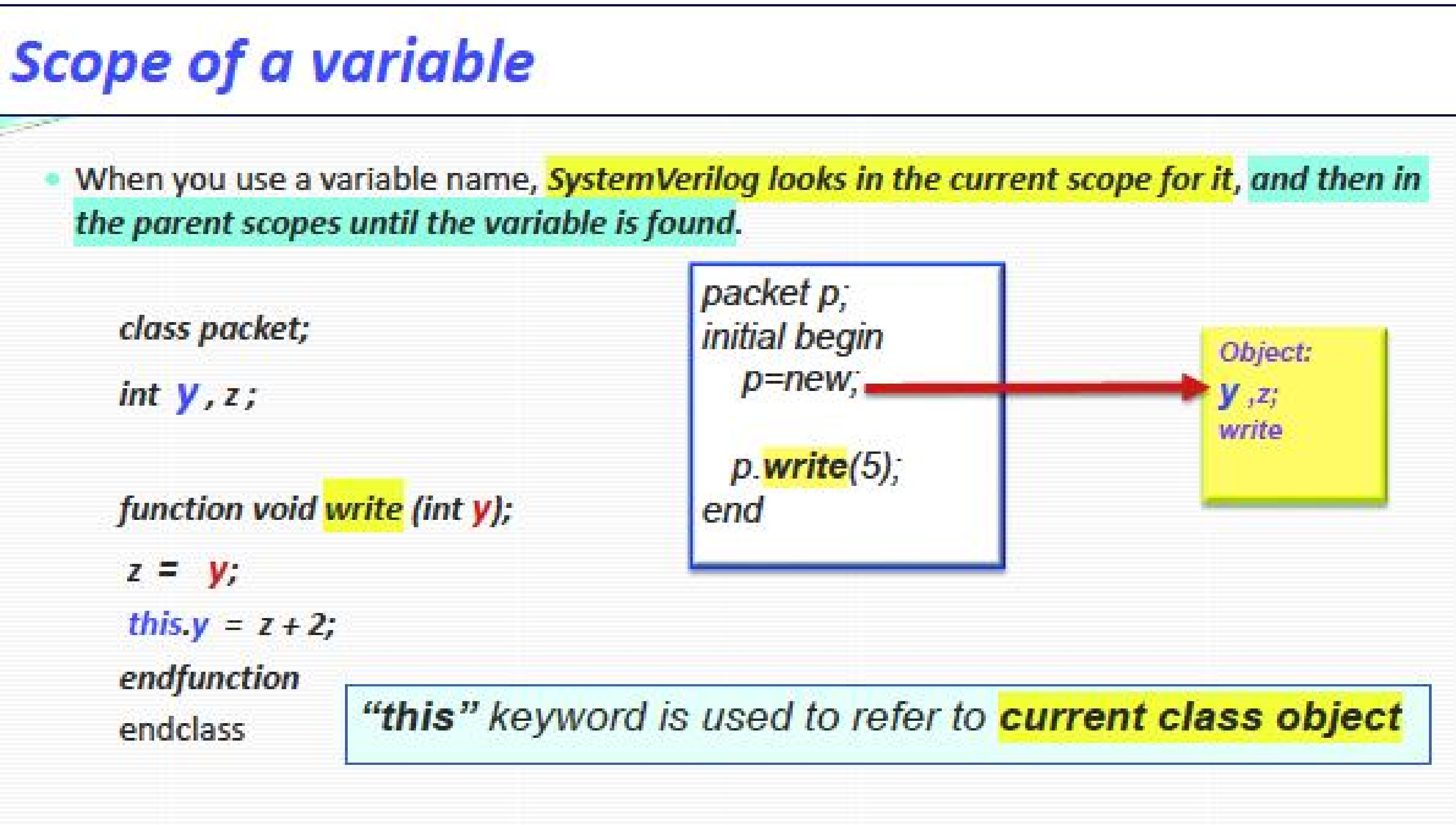
Scope of a variable

- When you use a variable name, ***SystemVerilog looks in the current scope for it***, ***and then in the parent scopes until the variable is found.***

```
class packet;

int y , z ;


function void write (int y);

 z =  y;

 y  =  z + 2;

endfunction

endclass
```



```
packet p;
initial begin
   p=new;

  p.write(5);
end
```



Scope of a variable

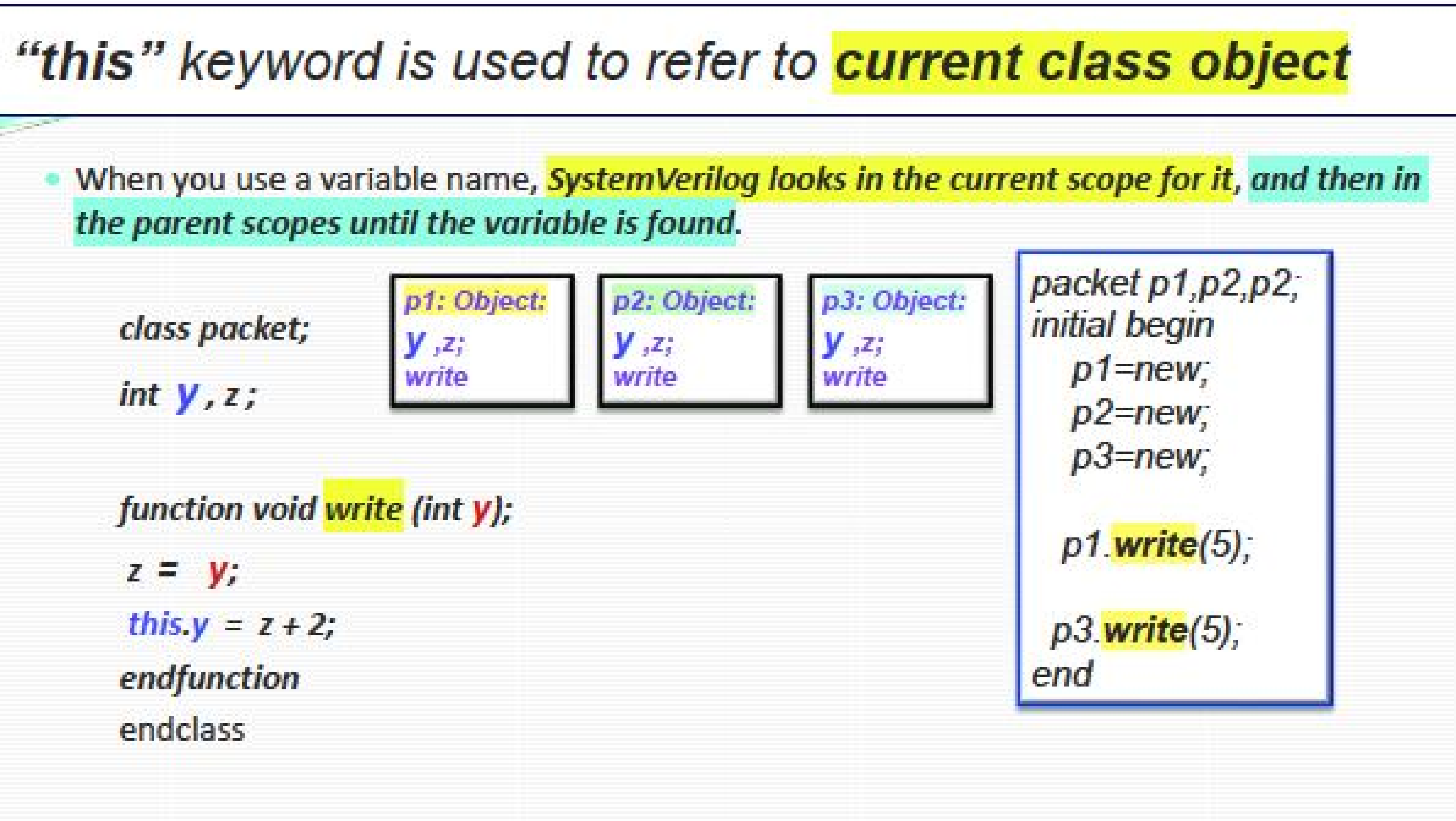
- When you use a variable name, *SystemVerilog looks in the current scope for it*, *and then in the parent scopes until the variable is found.*

```
class packet;

int  y , z ;


function void write (int y);

 z =   y;
 this.y  =  z + 2;
endfunction
endclass
```

```
packet p;
initial begin
   p=new;

  p.write(5);
end
```

Object:
y ,z;
write

“this” keyword is used to refer to **current class object**



“this” keyword is used to refer to **current class object**

- When you use a variable name, ***SystemVerilog looks in the current scope for it***, ***and then in the parent scopes until the variable is found.***

```
class packet;

int y , z ;


function void write (int y);
 z =  y;
 this.y = z + 2;
endfunction
endclass
```

p1: Object:
y ,z;
write

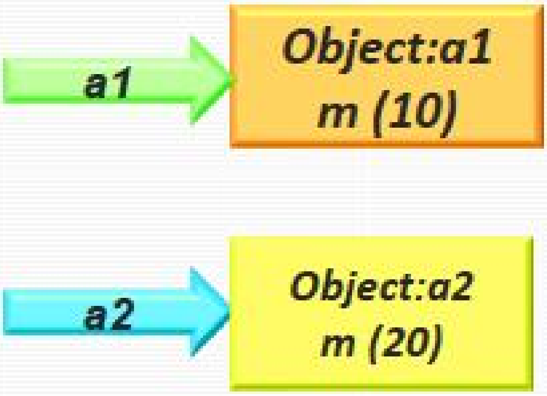
p2: Object:
y ,z;
write

p3: Object:
y ,z;
write

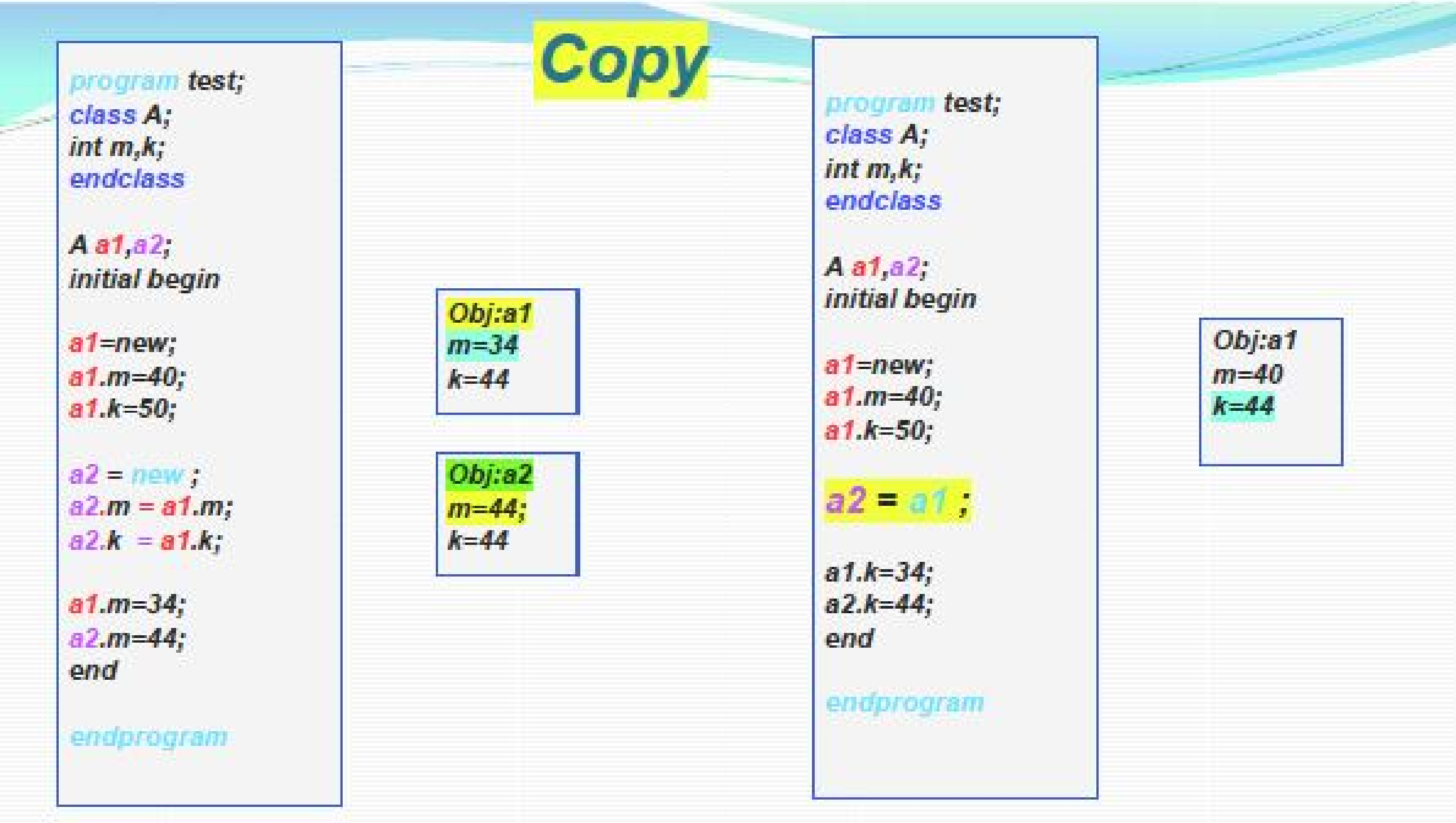
```
packet p1,p2,p2;
initial begin
   p1=new;
   p2=new;
   p3=new;

  p1.write(5);

 p3.write(5);
end
```

Handle assignments

```
class A;
int m;
enclass

A a1,a2;
initial begin
a1=new; a1.m=10 ;
a2=new; a2.m=20;
a1=a2;
a1.m=30;
a2.m=40;
end
```



```
module test;
reg [3:0] a1,a2;

initial begin
a2=10;
a1=a2;
$display(a1);
a2=20;
$display(a1);
end
endmodule
```



Copy

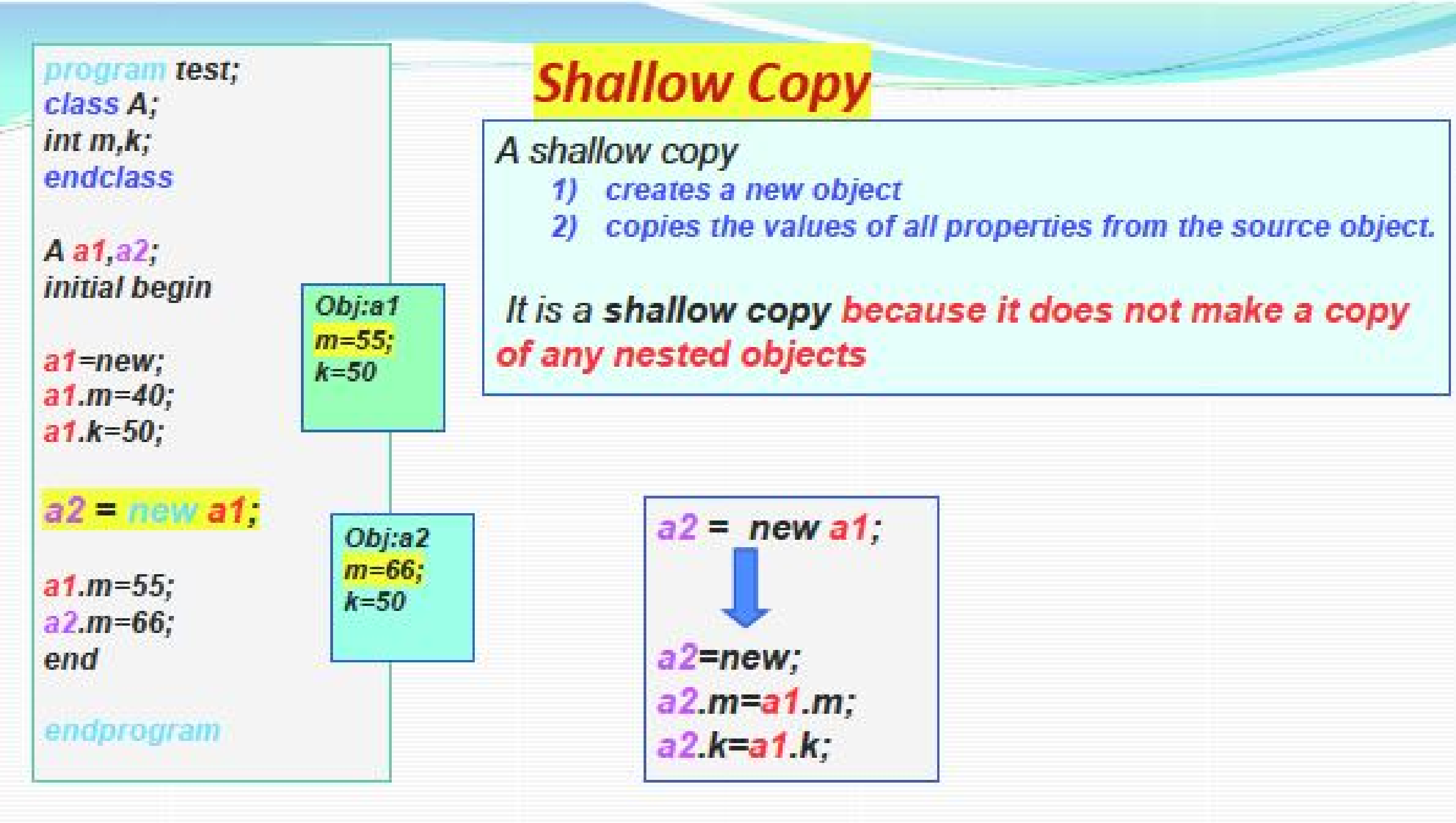
```
program test;
class A;
int m,k;
endclass

A a1,a2;
initial begin

a1=new;
a1.m=40;
a1.k=50;

a2 = new ;
a2.m = a1.m;
a2.k  = a1.k;

a1.m=34;
a2.m=44;
end

endprogram
```

```
Obj:a1
m=34
k=44
```

```
Obj:a2
m=44;
k=44
```

```
program test;
class A;
int m,k;
endclass

A a1,a2;
initial begin

a1=new;
a1.m=40;
a1.k=50;

a2 = a1 ;

a1.k=34;
a2.k=44;
end

endprogram
```

```
Obj:a1
m=40
k=44
```



Shallow Copy

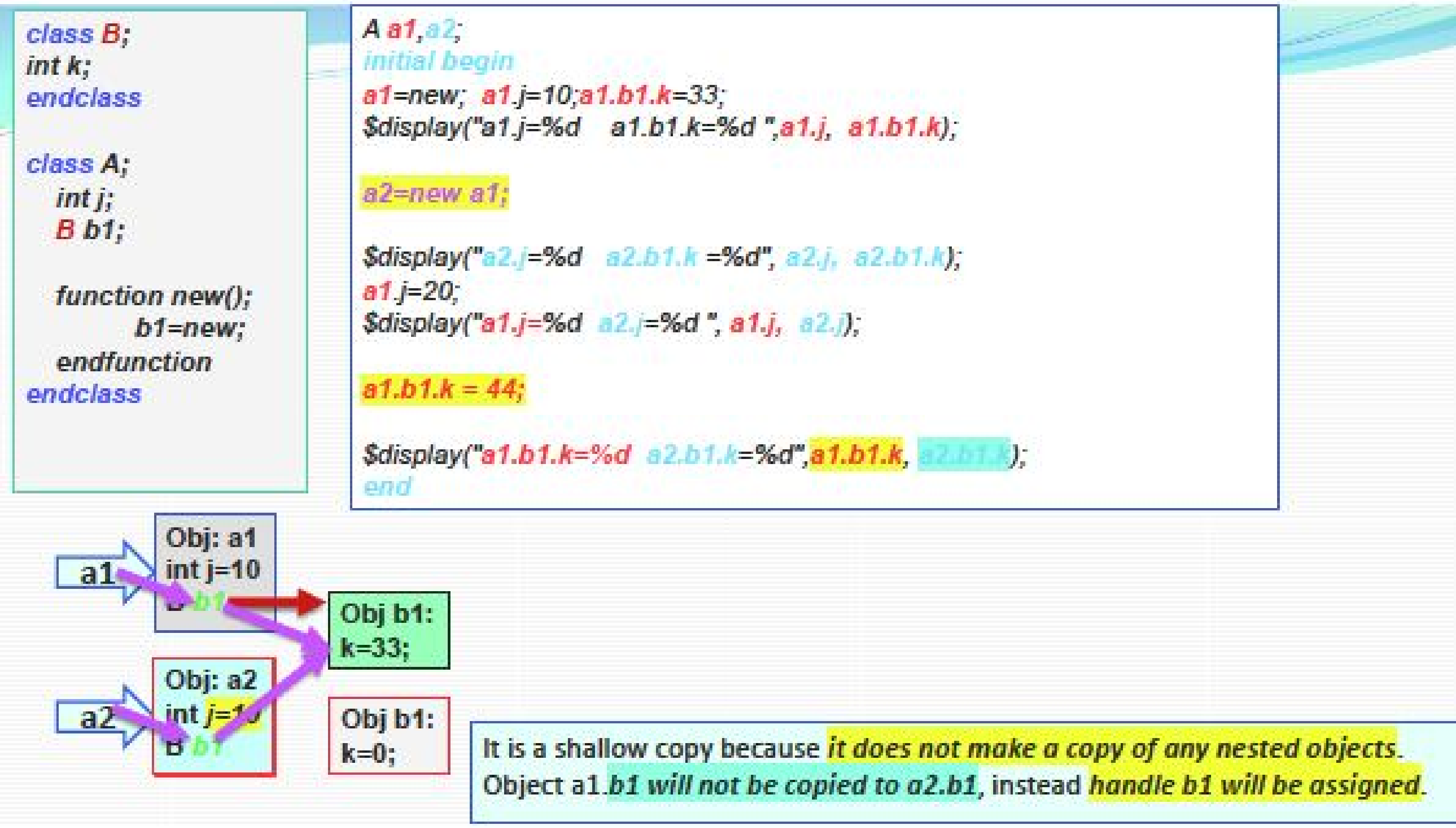
```
program test;
class A;
int m,k;
endclass

A a1,a2;
initial begin

a1=new;
a1.m=40;
a1.k=50;

a2 = new a1;

a1.m=55;
a2.m=66;
end

endprogram
```

A shallow copy

1) creates a new object
2) copies the values of all properties from the source object.

It is a **shallow copy** because it does not make a copy of any nested objects

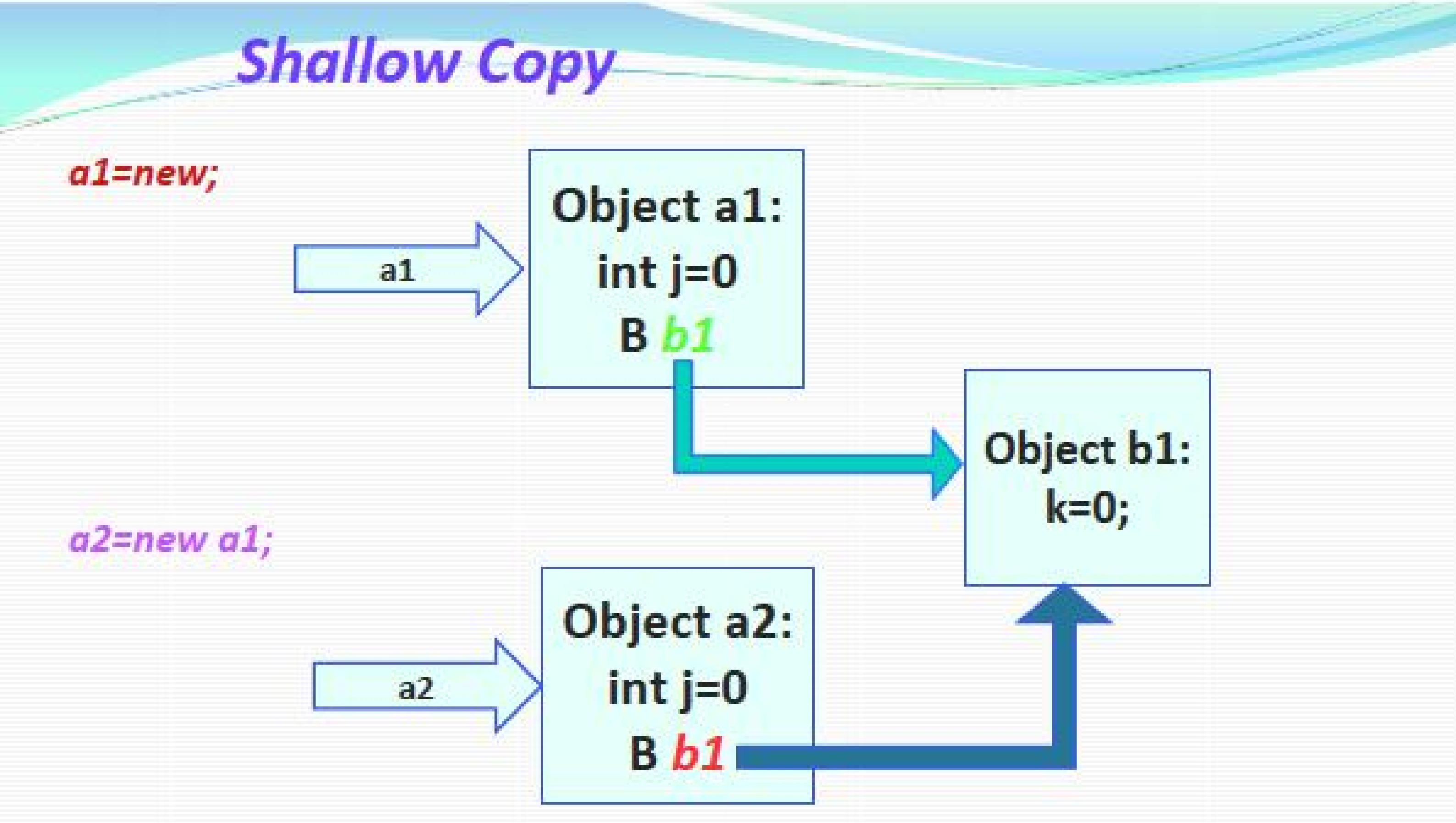
```
a2 =  new a1;

a2=new;
a2.m=a1.m;
a2.k=a1.k;
```



```
class B;
int k;
endclass

class A;
   int j;
   B b1;

   function new();
            b1=new;
   endfunction
endclass
```

```
A a1,a2;
initial begin
a1=new;  a1.j=10;a1.b1.k=33;
$display("a1.j=%d    a1.b1.k=%d ",a1.j,  a1.b1.k);

a2=new a1;

$display("a2.j=%d   a2.b1.k =%d", a2.j,  a2.b1.k);
a1.j=20;
$display("a1.j=%d  a2.j=%d ", a1.j,  a2.j);

a1.b1.k = 44;

$display("a1.b1.k=%d  a2.b1.k=%d",a1.b1.k, a2.b1.k);
end
```

It is a shallow copy because *it does not make a copy of any nested objects*. Object a1.*b1 will not be copied to a2.b1*, instead *handle b1 will be assigned*.



Shallow Copy

a1=new;

a1

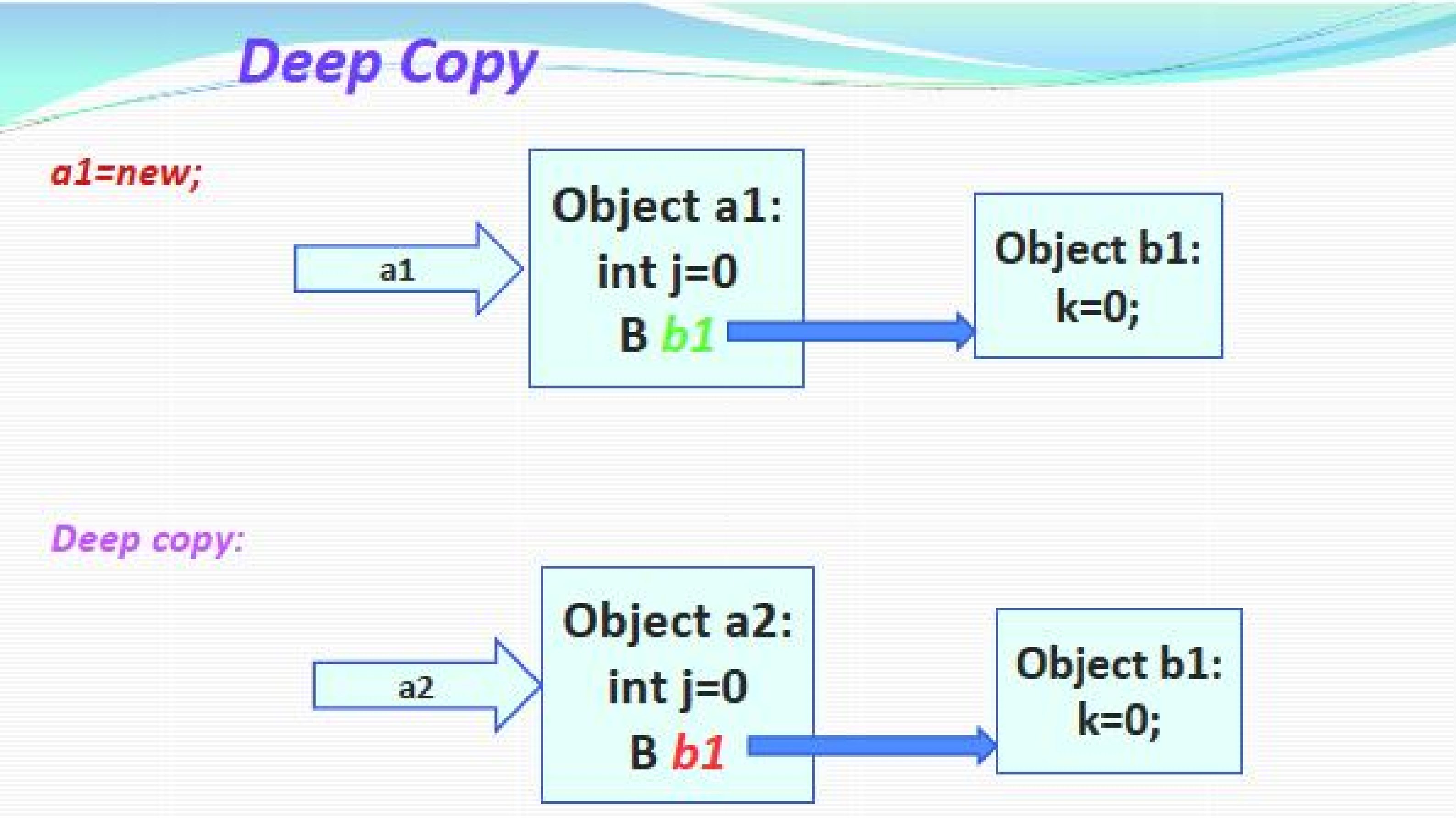
Object a1:
int j=0
B *b1*

Object b1:
k=0;

a2=new a1;

a2

Object a2:
int j=0
B *b1*



Deep Copy

a1=new;

a1

Object a1:
int j=0
B *b1*

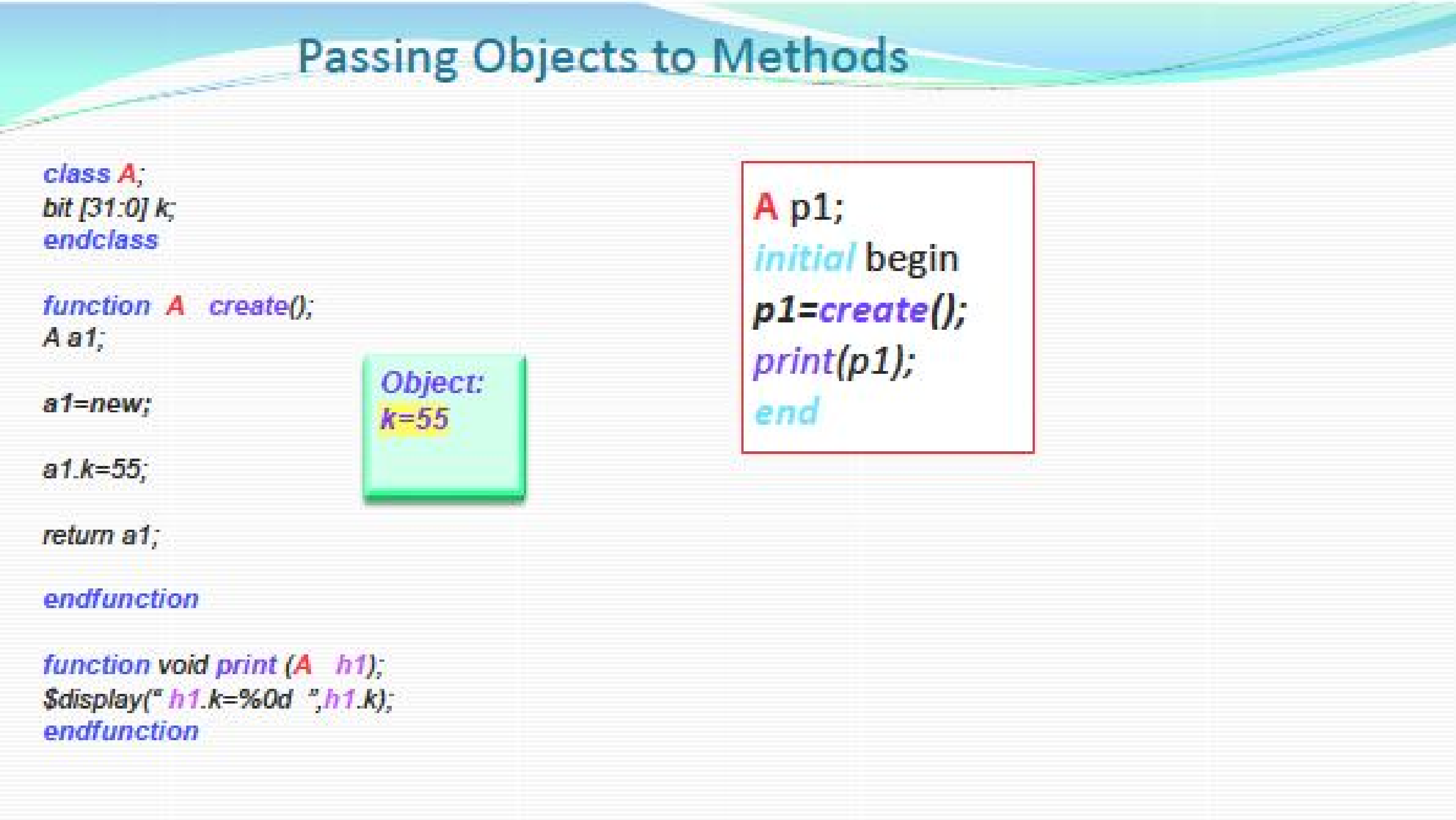
Object b1:
k=0;

Deep copy:

a2

Object a2:
int j=0
B *b1*

Object b1:
k=0;

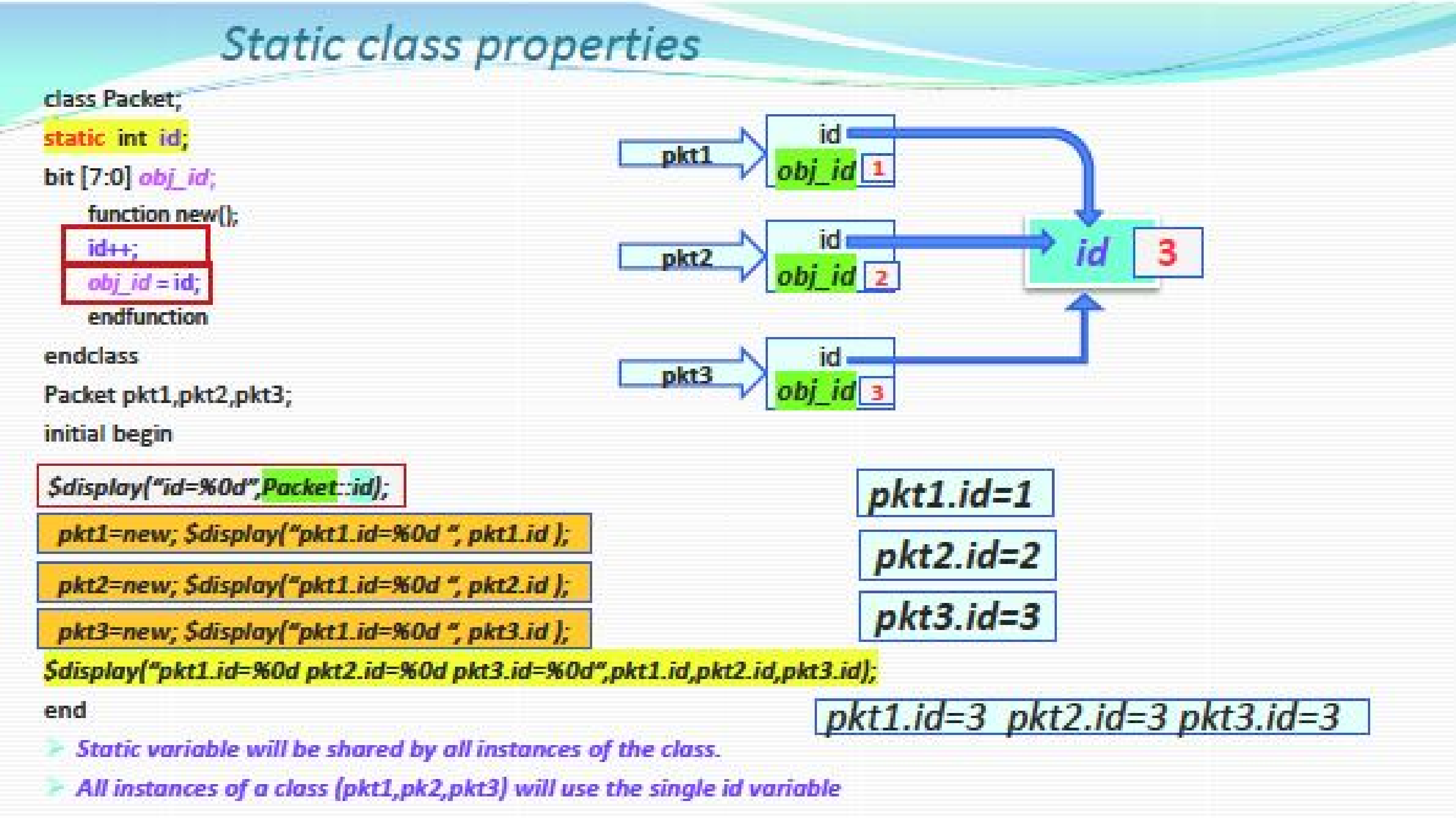


Passing Objects to Methods

```
class A;
bit [31:0] k;
endclass

function  A   create();
A a1;

a1=new;

a1.k=55;

return a1;

endfunction

function void print (A   h1);
$display(" h1.k=%0d  ",h1.k);
endfunction
```

Object:
k=55

```
A p1;
initial begin
p1=create();
print(p1);
end
```

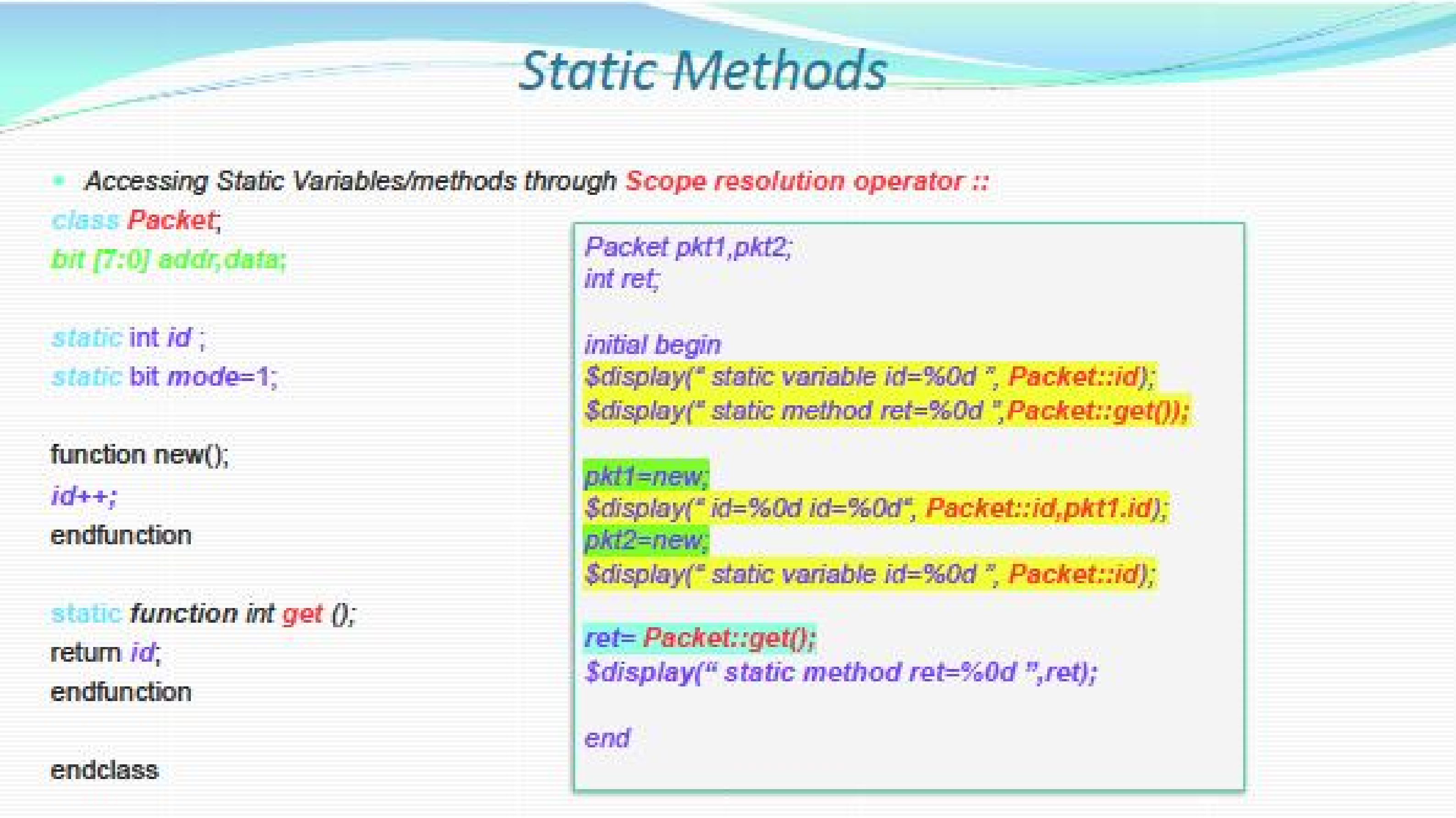


Static class properties

```
class Packet;
static  int  id;
bit [7:0] obj_id;
        function new();
        id++;
        obj_id = id;
        endfunction
endclass
Packet pkt1,pkt2,pkt3;
initial begin
$display(“id=%0d”,Packet::id);
 pkt1=new; $display(“pkt1.id=%0d “, pkt1.id );
 pkt2=new; $display(“pkt1.id=%0d “, pkt2.id );
 pkt3=new; $display(“pkt1.id=%0d “, pkt3.id );
$display(“pkt1.id=%0d pkt2.id=%0d pkt3.id=%0d”,pkt1.id,pkt2.id,pkt3.id);
end
```

```
pkt1.id=1
pkt2.id=2
pkt3.id=3
pkt1.id=3  pkt2.id=3 pkt3.id=3
```

- *Static variable will be shared by all instances of the class.*
- *All instances of a class (pkt1,pk2,pkt3) will use the single id variable*

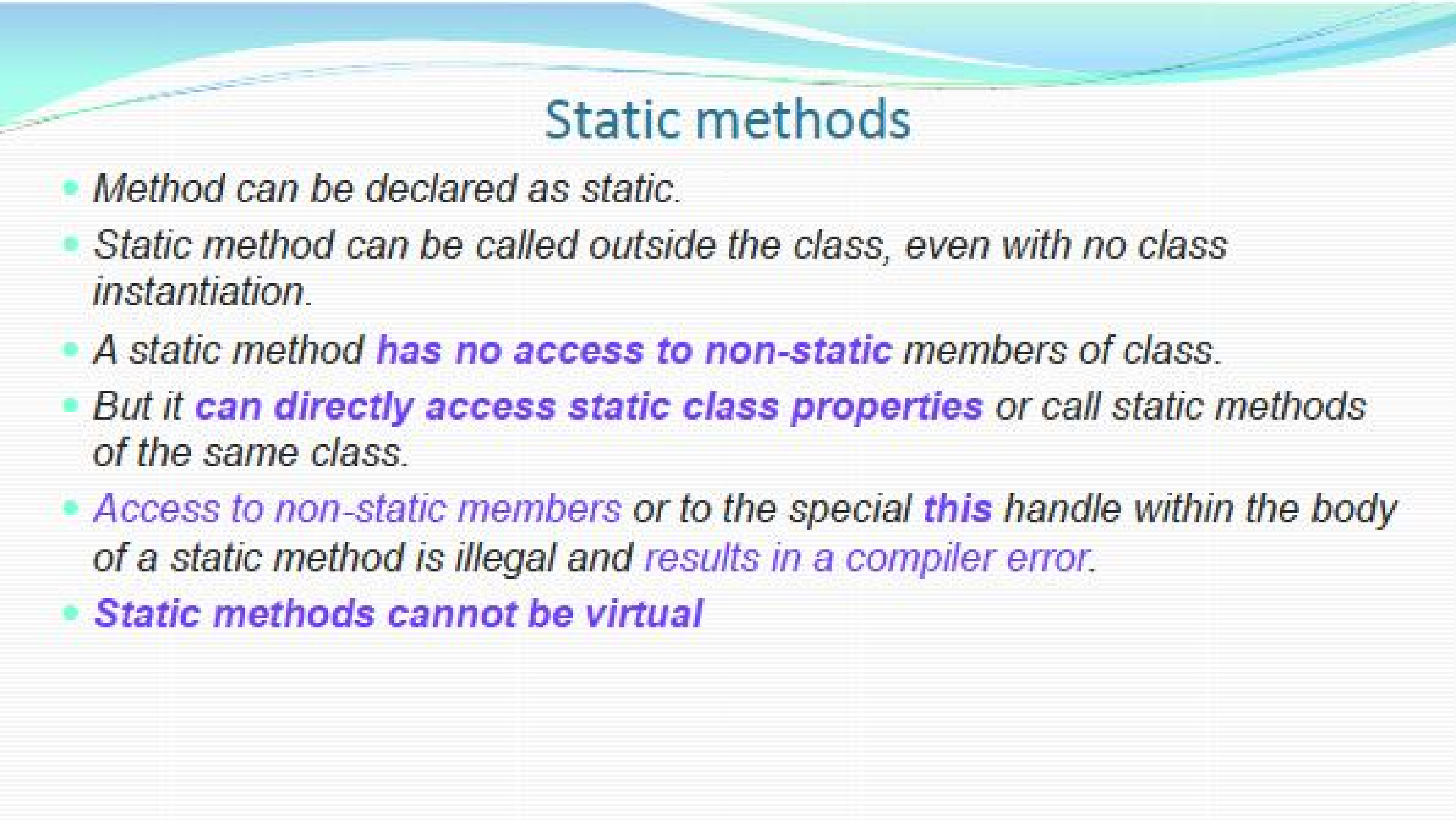


Static Methods

- *Accessing Static Variables/methods through* **Scope resolution operator ::**

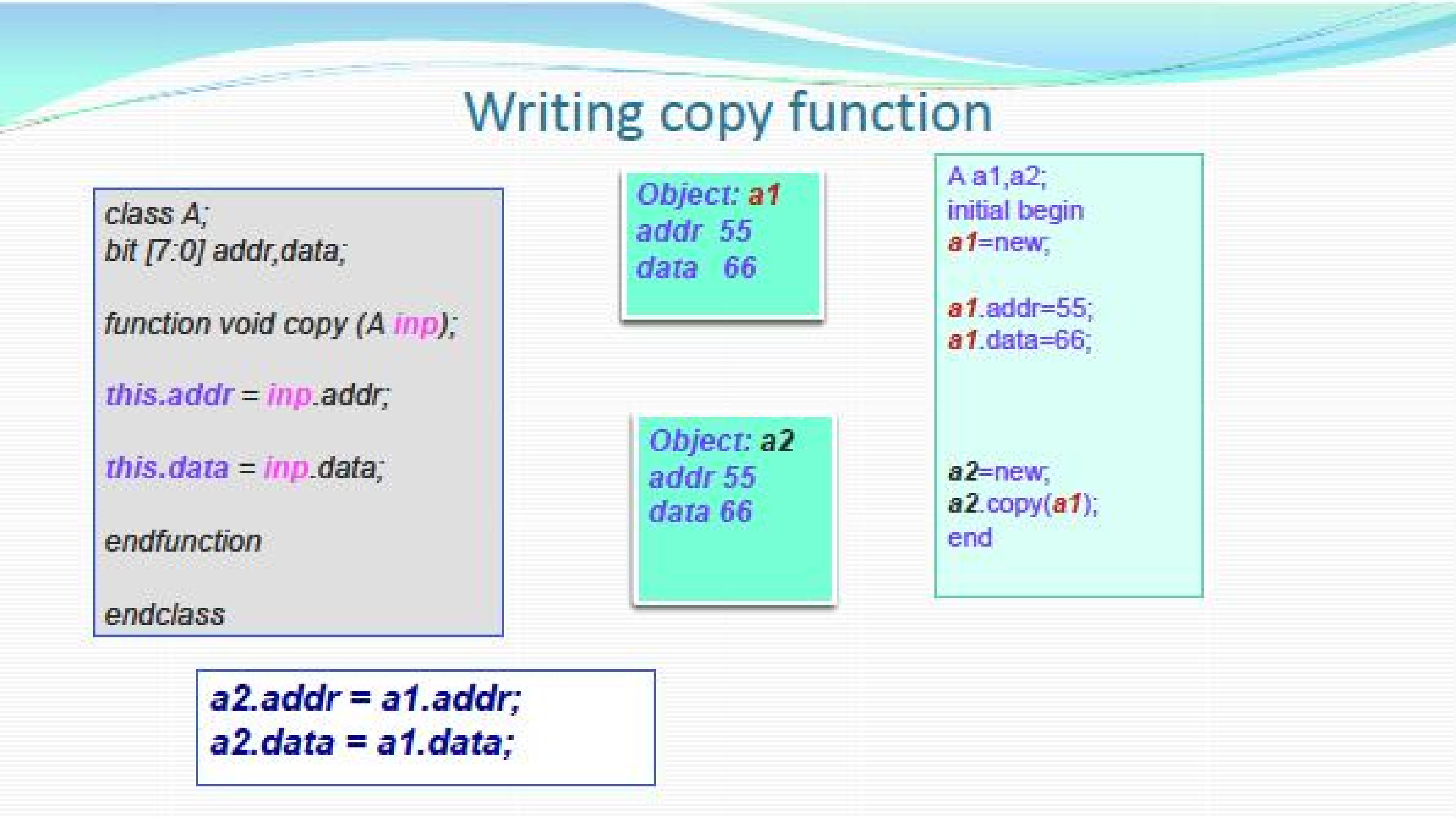
```
class Packet;
bit [7:0] addr,data;

static int id ;
static bit mode=1;

function new();
id++;
endfunction

static function int get ();
return id;
endfunction

endclass
```

```
Packet pkt1,pkt2;
int ret;

initial begin
$display(" static variable id=%0d ", Packet::id);
$display(" static method ret=%0d ",Packet::get());

pkt1=new;
$display(" id=%0d id=%0d", Packet::id,pkt1.id);
pkt2=new;
$display(" static variable id=%0d ", Packet::id);

ret= Packet::get();
$display(" static method ret=%0d ",ret);

end
```



Static methods

- *Method can be declared as static.*
- *Static method can be called outside the class, even with no class instantiation.*
- *A static method* ***has no access to non-static*** *members of class.*
- *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***



Writing copy function

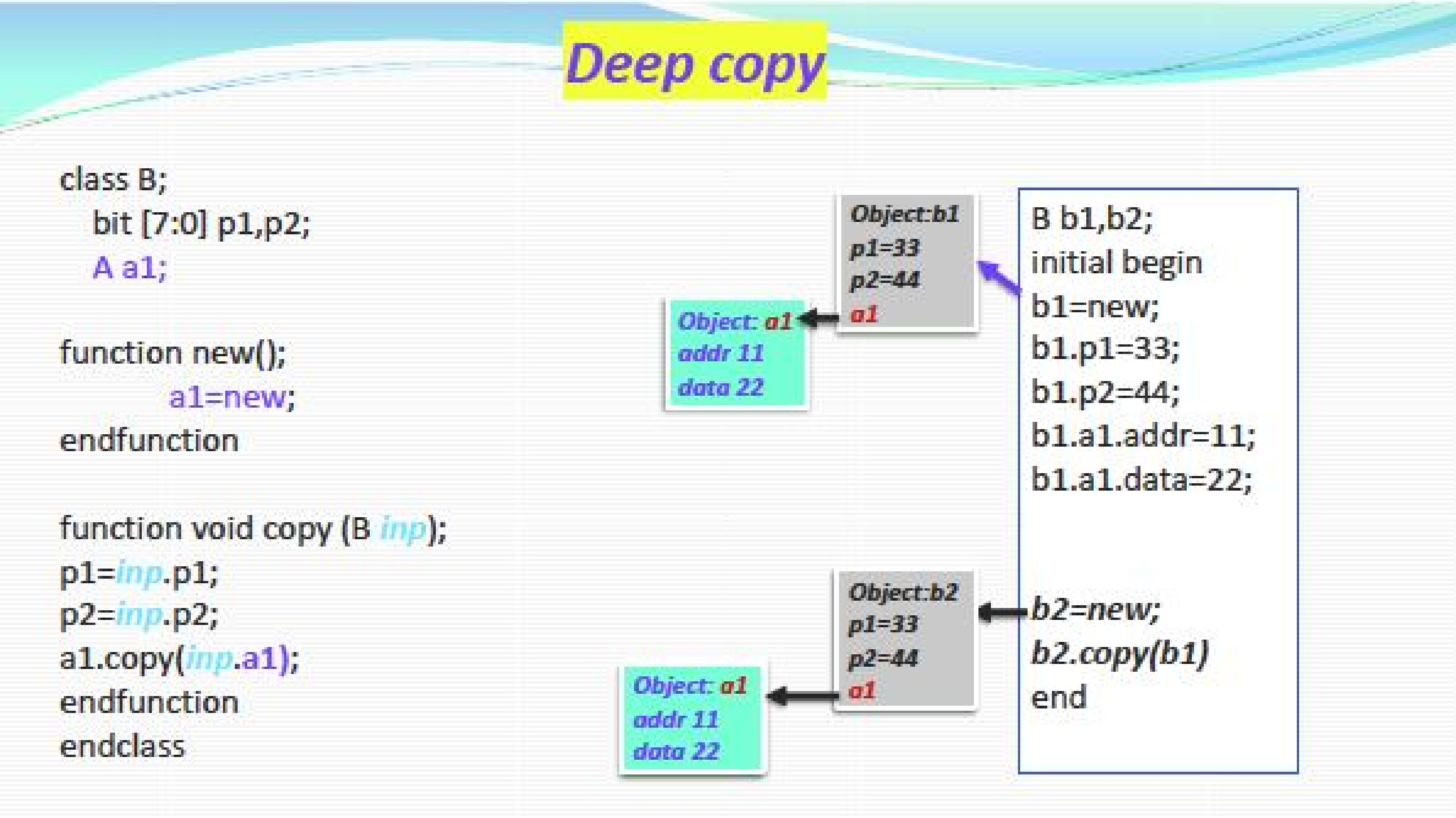
```
class A;
bit [7:0] addr,data;

function void copy (A inp);

this.addr = inp.addr;

this.data = inp.data;

endfunction

endclass
```

Object: a1
addr 55
data 66

Object: a2
addr 55
data 66

```
A a1,a2;
initial begin
a1=new;

a1.addr=55;
a1.data=66;

a2=new;
a2.copy(a1);
end
```

```
a2.addr = a1.addr;
a2.data = a1.data;
```



Deep copy

```
class B;
   bit [7:0] p1,p2;
   A a1;

function new();
          a1=new;
endfunction

function void copy (B inp);
p1=inp.p1;
p2=inp.p2;
a1.copy(inp.a1);
endfunction
endclass
```

```
B b1,b2;
initial begin
b1=new;
b1.p1=33;
b1.p2=44;
b1.a1.addr=11;
b1.a1.data=22;


b2=new;
b2.copy(b1)
end
```