Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

SV - OOP Concepts

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

What is OOP?

OOP is object oriented programming

Organize programs in same way as objects are organized in real world

Break programs in the same way

Several Languages support OOP

C++, Java, (SW) and System Verilog (HDL)

Class Basics

A *Class* is a description of some group of *things* that have something in common.

Objects are individual instances of “classes”.

Example: A class might be “Automobile”. Instances of the “Automobile” class might be “Joe’s car”, “Bob’s car”, “Sally’s truck”, etc.

Classes

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Inheritance: (is-a relationship)

Allows users to extend existing classes

Eg. Extending the “Automobile” class to create subclasses for “car”, “truck”, “van”, etc.

When using inheritance, the sub-class “inherits” all the parents public/protected data properties and methods.

Classes

Composition: (has-a relationship)

Composition is used for the case where one object *HAS-A* instance of another class.

For example, an “Automobile” class might have 4 instances of a “wheel” class

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Classes

Polymorphism:

Most common definition of polymorphism is the ability of the language to process objects differently depending on their *data type* or *class*.

Overloading

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Overloading

multiple methods in the same class with the same name but different signatures

SV Class doesn't support

Overriding

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Overriding

Two methods, one in a parent class and one in a child class, that have the same signature

lets you define a similar operation in different ways for different object types.

System Verilog supports overriding not overloading.

Class Definition

class*name*;
<data_declarations>;
<task/func_decls>;
endclass

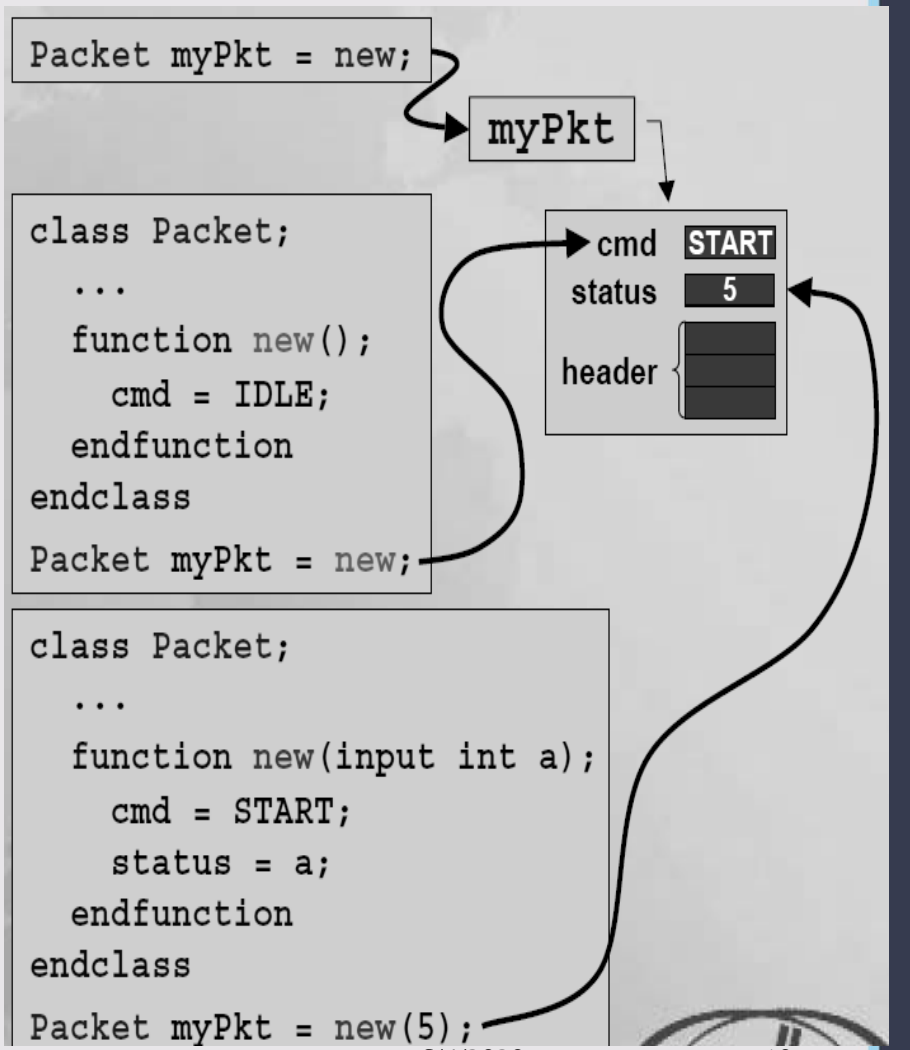
- Extern keyword allows out of body method declaration

```
class Packet;
  bit[3:0] cmd;
  int status;
  myStruct header;
  function int get_status();
    return(status);
  endfunction
  extern task set_cmd(input bit[3:0] a);
endclass

task Packet::set_cmd(input bit[3:0] a);
  cmd = a;
endtask
```

Class Instantiation

- Objects Allocated and Initialized Via Call to the **new** *Constructor* Method
- All objects have built-in **new** method
  - No args
  - Allocates storage for all data properties
- User-defined **new** method can initialize and/or do other things



Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Class Inheritance

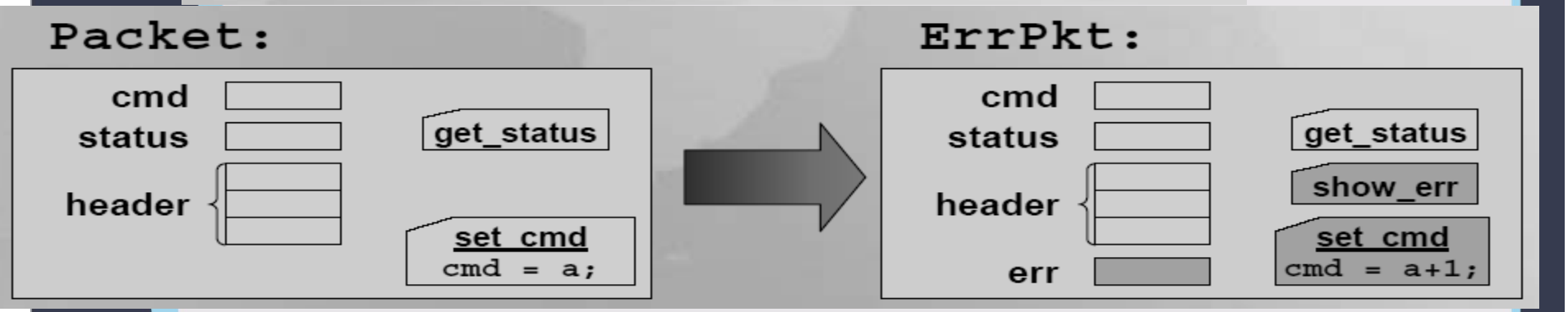
Keyword ***extends*** Denotes Hierarchy of Definitions

- Subclass inherits properties and methods from parent
- Subclass can redefine methods explicitly

Class Inheritance

Centurion University
Shaping Lives...
Empowering Communities...

```
class ErrPkt extends Packet;
  bit[3:0] err;
  function bit[3:0] show_err();
    return(err);
  endfunction
  task set_cmd(input bit[3:0] a);
    cmd = a+1;
  endtask // overrides Packet::set_cmd
endclass
```



Static properties

Static properties/data members are “static” to all instances of the class. This means that all instances share the same value of this variable. If one instance changes the value, it changes the value for all instances

Static methods do not require an instance of the class to operate on. Static methods may only modify static properties. To invoke a static method, use *Classname::methodName*

The special variable ***this*** is a predefined object handle for the current object instance

Polymorphism usage

Centurion UNIVERSITY
Shaping Lives...
Empowering Communities...

Base class pointers can be used to access objects of derived class

```
class BaseClass;
   virtual function in
   myFunc(int b);
        return(b + 10);
   endfunction myFunc
endclass: BaseClass
```

```
class myFirstClass extends
   BaseClass;
   virtual function int
   myFunc(int b);
        return(b – 3);
   endfunction: myFunc
endclass: myFirstClass
```

```
class mySecondClass extends BaseClass;
   virtual function int myFunc(int b);
        return(b + 3);
   endfunction: myFunc
endclass: mySecondClass
```

```
BaseClass bc;

// Returns an instance myFirstClass
bc = getFirstClassInstance();
$display("What do I print?  %d"
                  ,bc.myFunc(6));

// Returns an instance mySecondClass
bc = getSecondClassInstance();
$display("What do I print?  %d"
                  ,bc.myFunc(6));
```

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Data Hiding and Encapsulation

To make data members visible only to the class, use the *local* keyword.

```
class myPacket extends BasePacket;
  local int x;
```

To make data members visible only to the class, or any subclasses, use the *protected* keyword.

```
class myPacket extends BasePacket;
  protected int x;
```

Constant Class Properties

The *const* keyword may be used to make class properties unchangeable.

```
class myPacket extends BasePacket;
  const int size; // Assignment of constant
  value in
            //  declaration makes it constant
            //  to all instances.

  function new(int id);
      size = id * 4096; // Single assignment in
          // constructor OK
  ...
```

Centurion
UNIVERSITY
Shaping Lives...
Empowering Communities...

Abstract Classes

The *virtual* keyword may be used on a class to make the class “abstract”.

An abstract class may not be instantiated. Users must subclass the abstract class to create instances of the class.

```
virtual class BasePacket;
  …
```

Typedef Class and Forward References

Centurion University
Shaping Lives... Empowering Communities...

Sometimes it is necessary to use a class before it has been defined. To do this, you can use a typedef forward reference, then later define the class.

```
typedef class C2; // Forward declaration of
 C2
class C1;
 C2 c2Instance;
 …
endclass: C1
class C2;
 C1 c1Instance;
 …
endclass: C2
```

Parameterized Class

Centurion University
Shaping Lives...
Empowering Communities...

Allows Generic Class to be Instantiated as Objects of Different Types

```
class stack #(parameter type T = int;);
  local T items[];
  task push( T a ); ... endtask
  task pop( ref T a ); ... endtask
endclass

stack i_s; // default: a stack of int's

stack#(bit[1:10]) bs; // a stack of 10-bit vector

stack#(real) rs; // a stack of real numbers

stack#(Vfour) vs; // stack of classes
```