Class static methods

- A static method can access only static properties
  - Through the handle of the class using “dot” .
  - Through scope resolution operator (::)

```
class small_frame;
  static int number = 0;
  //int tag;
  bit [15:0] data;
  bit [15:0] addr;

  function new (bit [15:0] my_data, bit [15:0] my_addr);
    data = my_data;
    addr = my_addr;
    ++number;
  endfunction

  static function int get_count();
    $display("Count=%0d",number);
endclass

  module class_static_methods;
      small_frame f1, f2;
      int count;

    initial begin
      f1 = new(16'habcd,16'hcdab);
      count = f1.get_count();
      f2 = new(16'haaaa,16'hffff);
      count = f2.get_count();
      count = small_frame::get_count();
    end
    endmodule
```

this : current object handler

- Keyword *this* is a handle to the current class object
- Use it to reference class identifiers redeclared within a local scope
- Use it for non-static members
- Unless there is an ambiguity, not needed

```
class small_frame;
  int number;

  function new (int number);
    this.number = number;
    $display("small frame: %d", number);
  endfunction

endclass
```

```
module class_this;
  initial begin
    small_frame f = new(10);
  end

endmodule
```

Shallow copy

- Copies all the variables, handles of the object instance
- Change in c2 does not affect c1 (below example)

```
class small_frame;
  int number1;
  logic [7:0] data1;

  function new ();
    number1 = 10;
    data1   = '1;
  endfunction

endclass
```

```
module shallow_copy;
  small_frame c1, c2;
  initial begin
    c1 = new();
    $display("for c1: %h location number1=%0d, data1=%h",c1,c1.number1,c1.data1);
    c2 = new c1;
    $display("for c2: %h location number1=%0d, data1=%h",c2,c2.number1,c2.data1);
    c2.number1 = 20;
    $display("for c2: number1=%0d, data1=%h",c2.number1,c2.data1);
    $display("for c1: number1=%0d, data1=%h",c1.number1,c1.data1);
  end

endmodule
```

```
for c1: 00000001 location number1=10, data1=ff
for c2: 00000002 location number1=10, data1=ff
for c2: number1=20, data1=ff
for c1: number1=10, data1=ff
```

Assignment/Renaming

- Assigned/renamed handle points to the same memory location
- Change in c2 does affect c1 (below example)

```
initial begin
  c1 = new();
  $display("for c1: %h location number1=%0d, data1=%h",c1,c1.number1,c1.data1);
  //c2 = new c1; Shallow Copy
  c2 = c1; //assigning,renaming
  $display("for c2: %h location number1=%0d, data1=%h",c2,c2.number1,c2.data1);
  c2.number1 = 20;
  $display("for c2: number1=%0d, data1=%h",c2.number1,c2.data1);
  $display("for c1: number1=%0d, data1=%h",c1.number1,c1.data1);
end
```

```
for c1: 00000001 location number1=10, data1=ff
for c2: 00000001 location number1=10, data1=ff
for c2: number1=20, data1=ff
for c1: number1=20, data1=ff
```

Class Inheritance

- Classes can be extended to create a new sub class
- All the methods and properties of parent class are part of sub class

```
class small_frame;
  int number1;
  logic [7:0] data1;

  function new (int number1, logic [7:0] data1);
    this.number1 = number1;
    this.data1   = data1;
  endfunction
endclass

class big_frame extends small_frame;
  int number2;
  function new(int number1,logic [7:0] data1, int number2);
    super.new(number1, data1);
    this.number2 = number2;
  endfunction

endclass
```

```
module class_inheritance;
  small_frame c1;
  big_frame c2;

  initial begin
    c1 = new(100,8'hdd);
    $display("for c1: number1=%0d, data1=%h",c1.number1,c1.data1);
    c2 = new(500,8'haa,1000);
    $display("for c2: number1=%0d, data1=%h, number2=%0d",
                c2.number1,c2.data1,c2.number2);
  end

endmodule
```

```
for c1: number1=100, data1=dd
for c2: number1=500, data1=aa, number2=1000
```

Data hiding and encapsulation

- Properties and methods are accessible anywhere to the class handle, by default
- Can be controlled through – local

```
class small_frame;
  local int number;

  function new (int number);
    this.number = number;
  endfunction

  function disp();
    $display("number=%0d",number);
  endfunction

endclass

module class_encaps;
    small_frame f1;

  initial begin
    $display("try to aceess: number=%0d",f1.number);
  end
  endmodule
```

```
Access to local member 'number' in class 'small_frame' is not allowed here.
```

Data hiding and encapsulation

- Properties can be made available to derived class only
- Can be controlled through – protected

```
class small_frame;
  protected int number;
  function new (int number);
    this.number = number;
  endfunction

  function disp();
    $display("number=%0d",number);
  endfunction
endclass

class big_frame extends small_frame;
  int number2;
  function new(int number1,int number2);
    super.new(number1);
    this.number2 = number2;
  endfunction
endclass
```

```
module class_encaps;
    small_frame f1 = new(50);
    big_frame f2=new(12,24);
  initial begin
    f1.number1;
    $display("try to aceess: number=%0d",f1.number);
  end
endmodule
```

```
Access to protected member 'number' in class 'small_frame' is not allowed here.
```

Constant class property

- Make the class property – read only
  - Global constants and Instance constants – *const* is the keyword in both the cases

```
class small_frame;
  const int number = 100;
  int data=89;
endclass

module class_constants;
  small_frame f1=new();

  initial begin
    f1.number = 50;
    f1.data   = 50;
  end

endmodule
```

```
class small_frame;
  const int number;
  int data;

  function new();
    data = 55;
  endfunction

endclass

module class_constants;
  small_frame f1=new();

  initial begin
    f1.number = 50;
    f1.data   = 50;
  end

endmodule
```

- Global constant class properties cannot be assigned outside of an initialization
- Instance constants allows the user to assign value in run time only once in the class

Virtual class - properties

- Make the class abstract and be available only to the derived class
  - Abstract class cannot be instantiated – use the keyword *virtual*
  - It includes only the property but not the methods

```
virtual class small_frame;
  int number;

  function new (int number);
    this.number = number;
    $display("number=%0d", number);
  endfunction

endclass

module class_virtual;
  small_frame f1=new();

  initial begin
    f1.number = 50;
  end

endmodule
```

An abstract (virtual) class cannot be instantiated.

```
virtual class small_frame;

  int number;

endclass

class big_frame extends small_frame;

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

endclass

module class_virtual;
  big_frame f1=new;

  initial begin
    f1.number = 10;
    f1.disp();
  end

endmodule
```

number=10

Virtual class - methods

- Virtual methods provides prototypes for derived classes
  - Derived class will implement it in its own way
  - Methods can be
    - Virtual Functions
    - Virtual Tasks

```
virtual class small_frame;

  virtual function void disp;
  endfunction

endclass

class big_frame extends small_frame;

  int number;
  function void disp;
    $display("number=%0d", number);
  endfunction

endclass

module class_virtual_methods;
  big_frame f1=new;

  initial begin
    f1.number = 10;
    f1.disp();
  end

endmodule
```

number=10

Polymorphism

```
class small_frame;

  virtual task disp;
    $display("Base Class");
  endtask

endclass

class big_frame extends small_frame;

  task disp();
    $display("Derived Class");
  endtask

endclass

module class_polymorphism;

  small_frame f1
  big_frame f2;

  initial begin
    f1 = new;
    f1.disp();
    f2 = new;
    f2.disp();
  end

endmodule
```

```
Base Class
Derived Class
```

typedef in class

- Provides forward declaration when two classes dependent on each other
- Situation where a class is instantiated before it is declared

```
typedef class big_frame;

class small_frame;
  big_frame my_var;
  function new();
    $display("Inside small_frame");
  endfunction
endclass

class big_frame;
  small_frame f1;

  function new();
    $display("Inside big_frame");
  endfunction

endclass

module class_typedef;
  small_frame f1;
  big_frame f2;

  initial begin
   f1 = new();
   f2 = new();
   $display("Using typedef class");
  end
endmodule
```

```
Inside small_frame
Inside big_frame
Using typedef  class
```

Parameterized class

- A generic class to provide the template
- Size/Datatype can be set during instantiation
- Similar to parameter in module

```
class small_frame #(depth=32, width=8);

  bit [width-1] data;
  bit [width-1] locations [depth-1];
  int data;

  function new();
    $display("depth=%d, width=%d", depth, width);
  endfunction

endclass
```

```
module class_parameters;
  frame #(32,16) frame_instance();

  initial begin
    small_frame #(32,16) s_frame = new();
  end

endmodule
```

Assignments

1. Create a file in which define a class with constructor having properties: 32-bit address, 16-bit input data, 16-bit output data, 1 bit start, 1 bit done. Create a handle and generate 5 sets of values and display
2. Use the class extension for the above initialize the address and input data with a hexadecimal value of 1111_AAAA and 2222 respectively
3. For the above class, create a virtual method to display the values.
4. Write an example code with static method and static properties