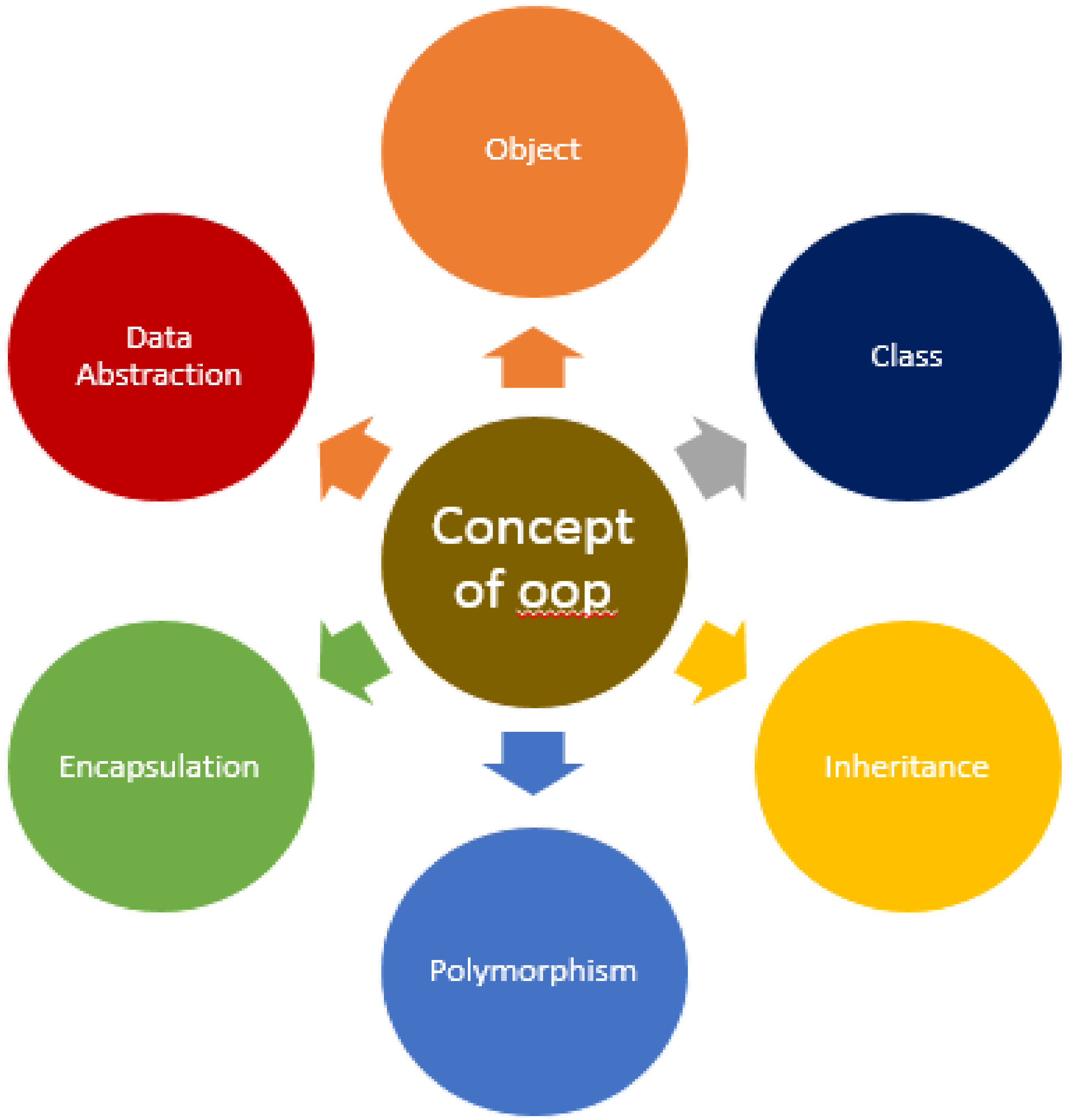
Object Oriented Programming in SystemVerilog

Reference

SV - LRM

Introduction

- Procedural language v/s Object Oriented
- Data and functions are built around the object



Introduction

- Object
  - Any entity that has state and behavior
  - Physical or logical
- Class
  - Collection of objects
  - logical entity
- Inheritance
  - Acquiring the properties and behavior of parent object
    - code reusability
- Polymorphism
  - More than one form
    - + is used for addition as well as concatenate two strings
- Abstraction
  - Representing only necessary information and hiding remaining details
- Encapsulation
  - Process of binding data and function

Class in SV

```
class frame;
  logic [4:0] addr;
  logic [7:0] payload;
  bit parity;
endclass
```

- A class is a user defined data type
- Defines subroutines (task/functions) to operate on data
- Can be dynamically created and deleted
- Can be used for temporary objects like transaction
- Class variable stores class handles
- Class objects accessed via class handles using “dot” (.)
- Class declaration does not occupy memory, instead creates a new type

Constructors

- The function new is a class constructor
  - Allocates memory and returns the address to the class handle
  - Constructor should be a function not task
  - There can be only one constructor per class

```
class frame;
  logic [4:0] addr;
  logic [7:0] payload;
  bit parity;
endclass

module class_example;

  frame F;

  initial
    F = new();

endmodule
```

```
class frame;
  logic [4:0] addr;
  logic [7:0] payload;
  bit parity;

  function new();
    addr    = 5'b0_0100;
    payload = 8'b1010_0101;
    parity  = 1'b1;
  endfunction
endclass

module class_example;

  frame F;

  initial begin
    F = new();
    $display("The values are, F=%h, addr=%b, payload=%b, parity=%b",
                   F, F.addr,F.payload,F.parity);
  end
endmodule
```

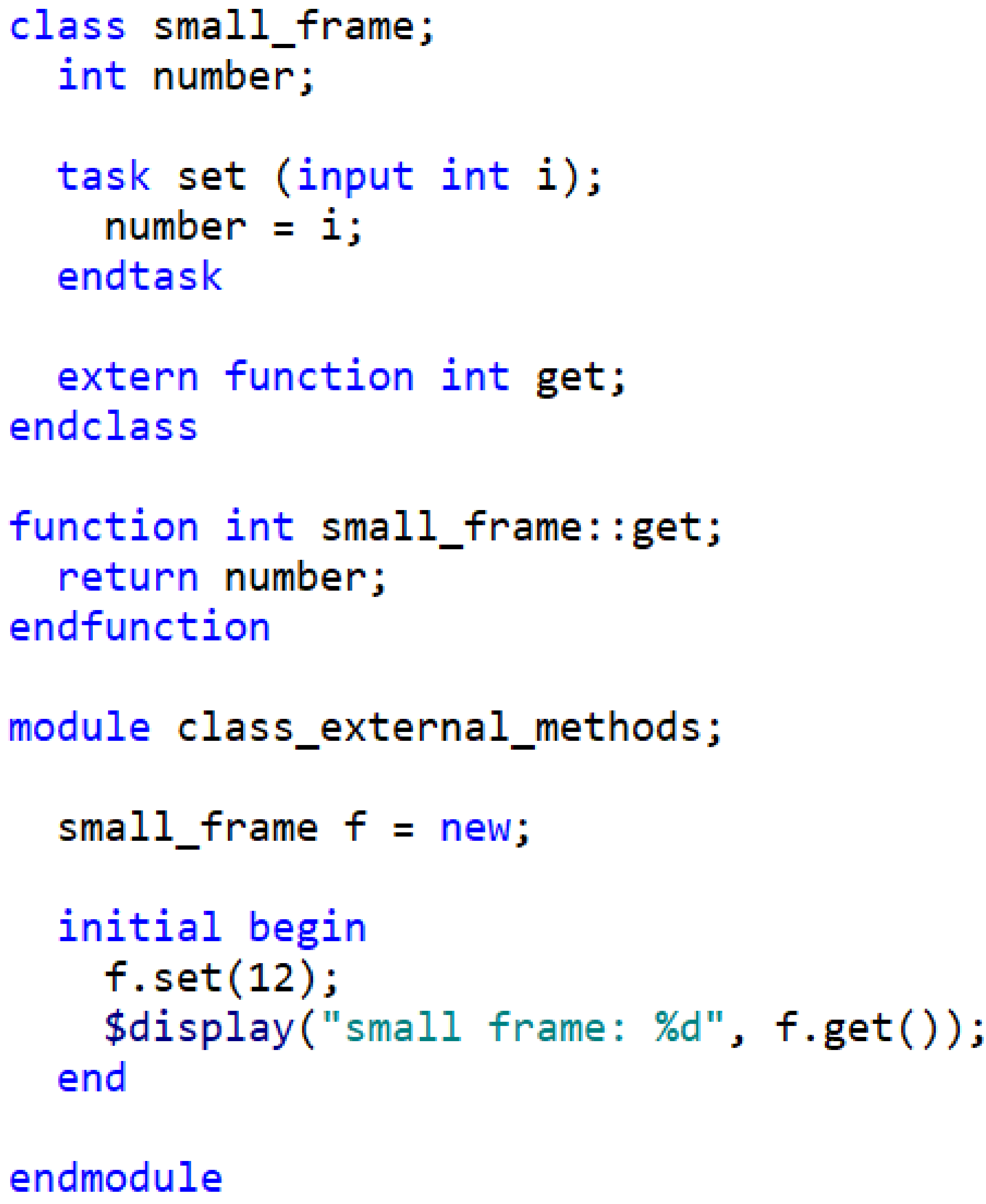
Class with methods

```
class frame;
  bit [3:0] command; bit [3:0] address;
  bit [3:0] master_id; integer   requested_time;
  integer status; integer issue_request;
  integer values;

  function new();
    command   = 0;
    address   = 4'b0;
    master_id = 4'bx;
  endfunction

  task new_values();
    command   = 2;
    address   = 4'b1;
    master_id = 24;
    $display("New values are, command=%0d, address=%0d, master_id=%0d"
                command, address, master_id);
  endtask

  task request_issue (int requested_time);
    $display("Requested time is = %0d", requested_time);
  endtask

  function int current_status (int status);
    $display("Status from function = %0d", status);
    return(status);
  endfunction
endclass
```

```
`include "frame.sv"

module class_methods;

  frame F;
  int current_stat;

  initial begin
    F = new();
    F.command = 1;
    F.address = $random;
    current_stat = F.current_status(20);
    F.new_values();
    F.request_issue(20);
  end

endmodule
```

Class with external methods

- For the sake of convenience, methods can be outside the *class* declaration
- Initially, prototype the method using *extern*
- Extern method_prototype;
  - Type, name, arguments
- Implement the method outside the class but in the same scope
  - Must match prototype
  - Link using ::, scope resolution operator
    - class_name :: method_name



```
class small_frame;
  int number;

  task set (input int i);
    number = i;
  endtask

  extern function int get;
endclass

function int small_frame::get;
  return number;
endfunction

module class_external_methods;

  small_frame f = new;

  initial begin
    f.set(12);
    $display("small frame: %d", f.get());
  end

endmodule
```

Class static properties

- Variable inside a class as static – only copy in all instances
- Any change in the value, will be reflected in other instance
- Useful in cases such as to find the total number of packets generated at a particular time

```
class small_frame;
  static int number = 16;
  int my_number = 10;
  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++;
    $display("static_number=%0d, my_number=%0d, data=%h, addr=%h",
                        number,my_number,data,addr);
  endfunction
endclass

module class_static_prop;
    small_frame f1, f2;

  initial begin
    f1= new(16'habcd,16'hcdab);
    f2= new(16'haaaa,16'hffff);
  end

endmodule
```

```
static_number=17, my_number=10, data=abcd, addr=cdab
static_number=18, my_number=10, data=aaaa, addr=ffff
```