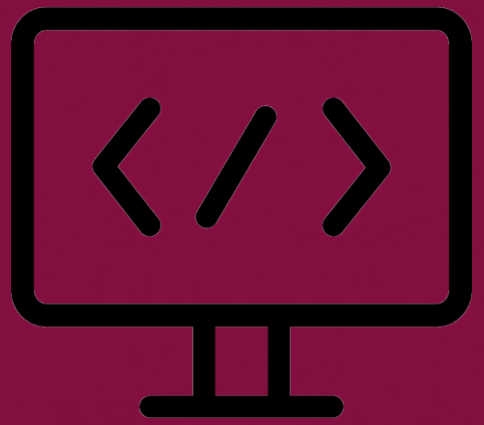
SEEE1022 INTRODUCTION TO SCIENTIFIC PROGRAMMING

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

CH14
Object Oriented Programming



Dr. Mohd Saiful Azimi Mahmud (azimi@utm.my)
P19a-04-03-30, School of Electrical Engineering, UTM

WHAT IS OOP?

Procedural programming is a list of instructions to perform a task

- Has no association between functions and the data they operate on
- Example programming languages: FORTRAN, C

Object Oriented Programming is a programming paradigm organized around objects with data and associated functions.

- Data – state
- Functions – behaviour
- Example programming languages : C++, Java, C#

WHY OOP?

- Modularity
- Abstraction
- Reuse
- Code maintenance/Maintainability
- Code expansion/Extensibility
- Improved Code Understanding
- Data encapsulation
- Logic Encapsulation

OOP FUNDAMENTALS

- **Class**: template for creating objects, defining properties and methods, as well as default values/behavior
- **Object**: instance of a class that has a state (properties) and behaviour (methods)
- **Properties**: data associated with an object: The variables.
- **Methods**: functions (behavior) defined in a class and associated with an object: The operation.
- **Attributes :** modify behavior of classes and class components
- **Inheritance**: object or class (subclass) derived from another object or class (superclass)
- **Polymorphism**: single interface to entities of different types

OOP IN MATLAB

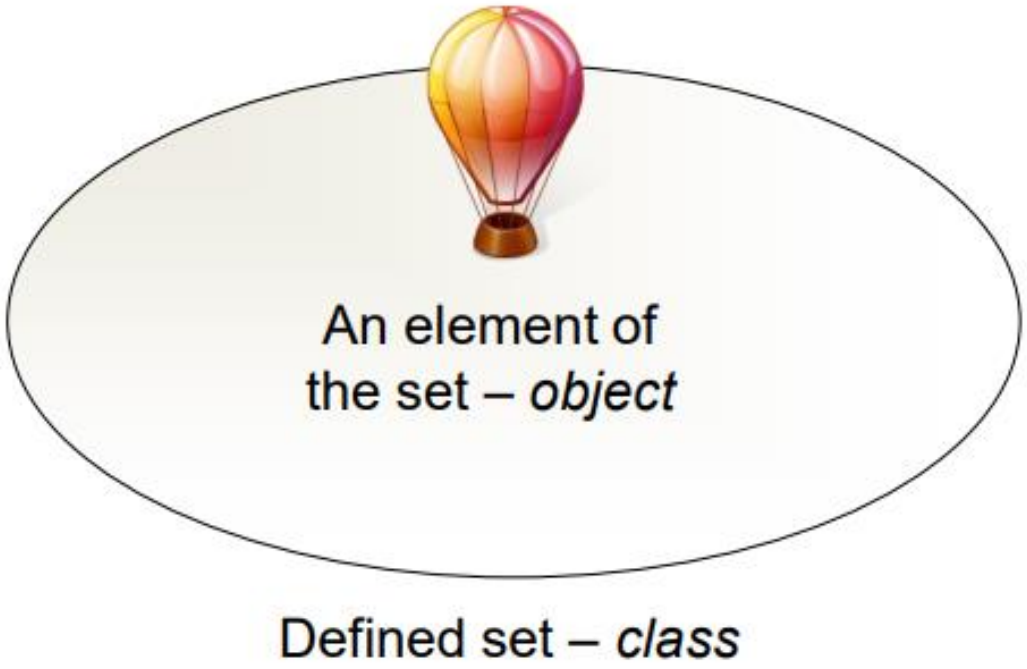
- Full support in MATLAB R2008b and above
- Dot notation works in R2014b and later
- MATLAB R2015a has introduced additional functionalities in term of editing capabilities

WHAT IS A CLASS, OBJECT?

- A **class** is a template for ideas/items. It is composed of a definition of a data structure and methods that can operate on the data structure (if created).
- An **object** is an actual **instance** of a class.
- To use a class one only needs to know the interface to the class (i.e. the methods of the class)

OBJECT ORIENTED TERMINOLOGY

- Class
  - Outline of an idea
  - *Properties* (data)
  - *Methods* (algorithms)
- Object
  - Specific example of a *class*
  - *Instance*



MATLAB OOP

- Class
- Property
- Constructor
- Method
- Static Method
- Constant Property
- Private Property
- Handle vs. Value Classes
- ~~Events~~
- ~~Overloading~~
- ~~Inheritance~~
- ~~Abstract Class~~

DEFINING A CLASS

- A class is instantiated with the `classdef` keyword.
- The source code must go inside a file with the same name (e.g. `myRectangle.m`).

```
classdef myRectangle
    properties
    % Properties (variable) go here
    end

    methods
    % Methods (function) go here
    end
end
```

DEFINING A PROPERTY

- The internal state of the class is saved as properties of the class
- No need of defining the type of each property.
- You can access the property or method using the . operator.

```
classdef myRectangle
    properties
        width; % The width of the rectangle
        height; % The height of the rectangle
    end

    methods
    % Methods go here
    end
end
```

In this example, the object `myRectangle` is defined with two properties, its height and width.

DEFINING A PROPERTY

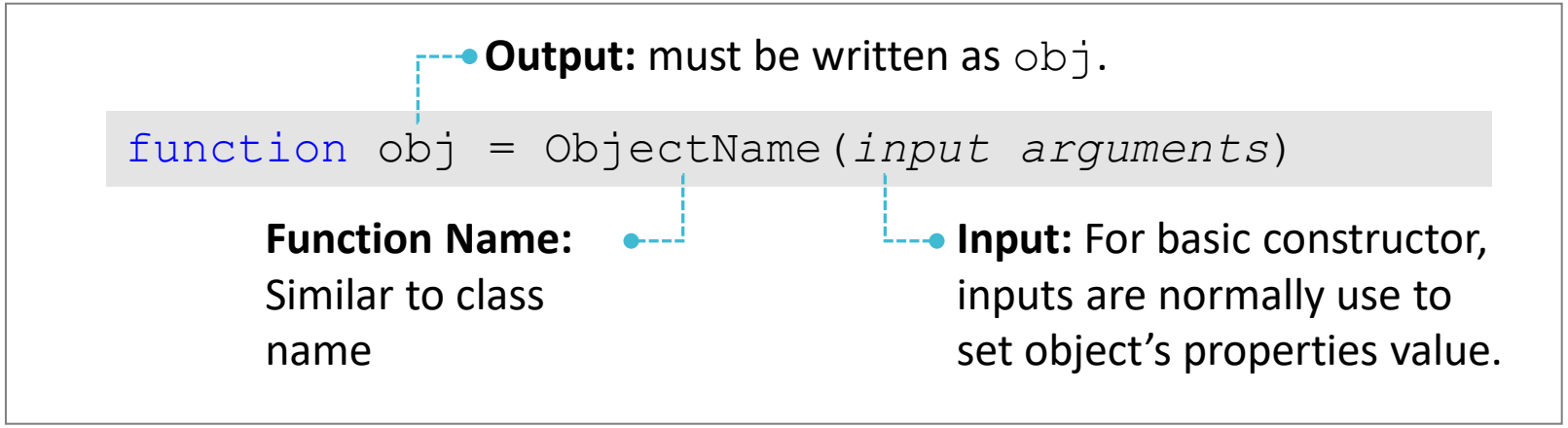
- Create an object `r` of class `Rectangle` as follows.
- Access the property using . operator.

```
>> r = myRectangle
r =
  myRectangle with properties:
     width: []
    height: []

>> r.width = 5
r =
  myRectangle with properties:
     width: 5
    height: []
```

- The properties of the object can be accessed (get or set) using the . operator.
- In this example property `width` is set equals to 5.

DEFINING A CONSTRUCTOR

- An object is created by invoking a special method known as constructor.
- The constructor is a function with similar name to the class's name.
- If no constructor is defined, by default MATLAB generates a constructor with no input arguments.



DEFINING A CONSTRUCTOR

- Below is an example of defining a user-defined constructor for object `myRectangle`.

```
classdef myRectangle
    properties
        width; % The width of the rectangle
        height; % The height of the rectangle
    end

    methods
        %Constructor
        function obj = myRectangle(w,h)
            obj.width = w ;
            obj.height = h ;
        end
    end
end
```

DEFINING A CONSTRUCTOR

- Below is how an object with user-defined constructor is created at command window.

```
>> r = myRectangle(3,2)
r =
  myRectangle with properties:
    width: 3
   height: 2

>> r = myRectangle
Not enough input arguments.

Error in myRectangle (line 15)
            obj.width = w ; obj.height = h ;
```

With the user-defined constructor, setting value for the object's properties can be done straight when creating the object.

DEFINING A METHOD

- The third essential element of a class definition is a set of methods.
- Methods are operations that are common or basic to an object.
- For example, some of the basic operations that can be applied to a rectangle are:
  1) Compute area.
  2) Compute perimeter.
  3) Scale up or down the rectangle.
  4) Rotate the rectangle by $90°$.
  5) Trim the rectangle.

- Every method is define through a function, similar to constructor. The difference is, method name must be different with the class name.

DEFINING A METHOD

- Below is an example of defining methods to an object.

```
classdef myRectangle
    properties
        width; height;
    end

    methods
        %Constructor
        function obj = myRectangle(w,h)
            obj.width = w ; obj.height = h ;
        end
        %Methods
        function a = getArea(obj)
            a = obj.width*obj.height;
        end
        function obj = scale(obj,n)
            obj.width = n*obj.width ;
            obj.height = n*obj.height ;
        end
    end
end
```

DEFINING A METHOD

- Below is how the methods are applied onto the object `myRectangle`.

```
>> a = myRectangle(2,3)
a =
  myRectangle with properties:
     width: 2
    height: 3

>> area = a.getArea()
area =
     6

>> b = a.scale(2).getArea()
b =
    24
```

- First, an object `a` of a class myRectangle is created.
- Second, area of object `a` is computed using method `getArea()`.
- Third, area of scaled version of object `a` by 2 is computed by first applying method `scale()` followed by method `getArea()`.

DEFINING A CONSTANT PROPERTY

- A constant property is a property whose value cannot be modified after the first assignment:

```
classdef myCircle
    properties
        radius;
    end
    properties (Constant)
        PI = 3.14 ;
    end
    % Other definitions
end
```

```
>> a = myCircle;
>> a.PI
ans =
    3.1400

>> a.PI = 2
You cannot set the read-only property 'PI' of myCircle.
```

DEFINING A PRIVATE PROPERTY (ATTRIBUTE)

- Properties that should not be visible (nor modifiable) from the outside.
- You can also have private methods (using an equivalent syntax).

```
classdef myRectangle
    properties (Access = private)
        trimLength = 1;
    end
    % Other definition
End
```

```
>> r = myRectangle(3,2)
r =
  myRectangle with no properties.
```

OBJECT ARRAY

- Next slide shows a class named `myRectangle`. For this class there are:
  1) 4 properties, which their values will be set when an object of this class is created.
  2) 1 constant and also 1 private property.
  3) A user-defined constructor that accept width and height value. This constructor also set value for all of the 4 properties in (1).
  4) There is 1 method that can be used to trim the `myRectangle` object according to the constant and private property value.

- Use the `myRectangle` class to create 2 elements vector of `myRectangle` object that start with certain width and height, followed by the trimmed rectangle.

OBJECT ARRAY

```
classdef myRectangle
    properties
        width; height; area; perimeter;
    end
    properties (Constant)
        trimLength = 1;
    end
    properties (Access = private)
        minimumTrim = 4;
    end

    methods
        % Constructor
        function obj = myRectangle(w,h)
            obj.width = w ; obj.height = h ;
            obj.area = obj.width*obj.height;
            obj.perimeter = 2*obj.width + 2*obj.height;
        end
        % Methods
        function obj = trim(obj)
            if obj.width > obj.minimumTrim
                obj.width = obj.width - 2*obj.trimLength;
            end
        end
    end
end
```

OBJECT ARRAY

- Below is the solution for the Example

```
>> a = myRectangle(5,7);
>> rectarray = [a a.trim()]
rectarray =
  1×2 myRectangle array with properties:

    width
    height
    area
    perimeter
    trimLength

>> rectarray(2)
ans =
  myRectangle with properties:
          width: 3
         height: 7
           area: 35
      perimeter: 24
     trimLength: 1
```

CASE STUDY: GRAPHICS OBJECT

- When plot function is performed, various graphics objects are created to display the graphs. 3 of the objects are as below:
  1) Figure
  2) Axes
  3) Line

- These objects has their own properties and methods, which can be used to customize plotting.
- To customize the plotting, function `figure`, `axes` and `plot` based on the following syntax should be used:

`f = figure(___)`	Get `Figure` object
`ax = axes(f,___)`	Get `Axes` object
`h = plot(ax,___)`	Get `Line` object

GRAPHICS OBJECT

- Below is a user-defined function to create a plot inside another plot where properties of the Axes and Line objects are customized.

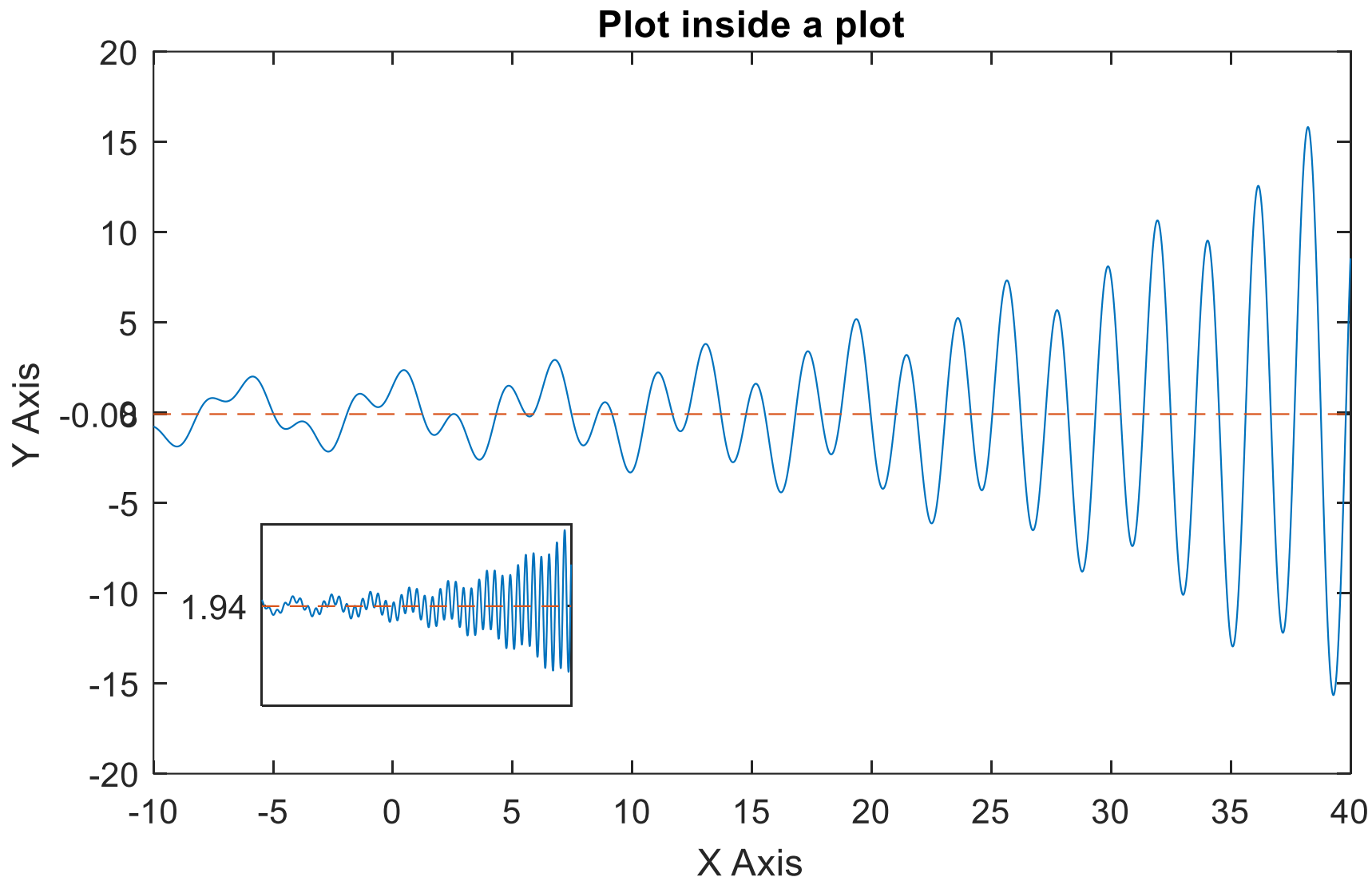
```
function plotinplot(x,y,xin,yin)
% This function will plot y vs x as the main plot and yin vs xin as the
% smaller plot inside the main plot
avgy = mean(y)*ones(1,length(x));
avgyin = mean(yin)*ones(1,length(xin));

hfig = figure('Name','Plot in Plot');  % Create Figure object
hax = axes('Parent',hfig);             % Create Axes object on hfig
haxin = axes('Parent',hfig,'Position',[0.2 0.2 0.2 0.2]); % Create Axes object by
% specifying Position property.
p = plot(hax,x,y,x,avgy);              % Create Line object array on hax object
pin = plot(haxin,xin,yin,xin,avgyin);  % Cretae Line object array on haxin object

xlabel(hax,'X Axis'), ylabel(hax,'Y Axis'), title(hax,'Plot inside a plot')
p(2).LineStyle = '--';                      % Set p(2) object property
pin(2).LineStyle = '--';                    % Set pin(2) object property
haxytick = sort([hax.YTick avgy(1)]);
hax.YTick = haxytick;                       % Set Ytick property of hax object
haxin.XTick = [];                           % Set several haxin propeties
haxin.YTick = avgyin(1);                    % .
haxxtick = hax.XTick;                       % .
haxin.XLim = [haxxtick(1) haxxtick(end)];   % .
haxin.YLim = [haxytick(1) haxytick(end)];   % .
```

GRAPHICS OBJECT

- Below is how two signals are plotted using the `plotinplot()` function.
- Note that the `plotinplot()` function is also displaying a line indicating the average value of the plotted signal.

```
>> x = -10:.005:40;
>> y1 = 1.5*cos(x) + exp(.07*x).*sin(3*x);
>> y2 = 1.5*sin(x) + exp(.07*x).*cos(5*x) + 2;
>>
>> plotinplot(x,y1,x,y2)
```

GRAPHICS OBJECT



REFERENCES

- Matthew J. Zahr, Advanced MATLAB for Scientific Programming, Stanford University, 21st April 2015
- Simone Scardapane, Adaptive Algorithms and Parallel Programming - OOP in MATLAB, 2014-2015

UTM
UNIVERSITI TEKNOLOGI MALAYSIA

univteknologimalaysia utm_my utmofficial

Thank You

www.utm.my

innovative • entrepreneurial • global