W1005
Intro to CS and Programming in MATLAB

Object Oriented Programming (OOP)

Fall 2014

Instructor: Ilia Vovsha

http://www.cs.columbia.edu/~vovsha/w1005

Outline

- Review of concepts: objects, classes, methods, properties, inheritance
- Classes in MATLAB
  - Self-contained
  - @classname folder
- Initialization, constructor
- Scope & Access

OOP (concepts)

- Program: collection of objects
- Object: data structure with collection of behaviors
- Object: instance of a class
- Class: defines attributes (variables) and methods (functions), i.e. behavior applied to class/instances
- Each instance of class (object) keeps track of its own attribute values
- Instance methods refer to actions of specific object
- Class methods refer to actions of entire class of (many) objects
- To generate a new object we use a constructor method

OOP (more concepts)

- Can share methods between classes: inheritance
- Derived / sub class inherits methods from base / parent / super class
- The derived class can "update" the behavior of the parent class
- Given an (derived) object, how do we select the most appropriate method for it? Polymorphism
- General design principle: when using the object, the object is a black box (we shouldn't manipulate attributes / methods directly)
- Encapsulation: access objects through methods alone

OOP (inheritance)

- We can define a hierarchy of classes to share methods between them
- Derived / sub class inherits methods from
- base / parent / super class
- Single Inheritance: one parent class (search parent for methods)
- Multiple Inheritance: multiple parent classes

Classes in MATLAB

- Two ways to specify classes with respect to folders:
  - Create a single, self-contained class definition file in some folder
  - Distributing a class definition to multiple files in some @classname folder
- Which approach?
  - Simple class, few functions → single file
  - Large class, lots of functionality → class folder
- Useful function:
  - Add directory to search path: addpath('/export/projects/code');

Self-Contained Class (blocks)

- Each block starts with a keyword and closes with 'end'
- The basic components (blocks):
  - *classdef*: contains the class definition within a file
  - *properties:* contains property definitions, optional initial values
  - *methods:* contains function definitions for the class methods
  - ~~*events*: contains the names of events that this class declares.~~
  - *enumeration:* contains the enumeration members defined by the class
- *properties/methods/events* block: one for each unique set of attribute specifications)
- Note: properties, methods, events, and enumeration are keywords only within a classdef block

Class Blocks (example)

```
classdef  circle
    properties
        x = 0.0; y = 0.0;
        r = 1.0;
    end
    methods
        function c = circle(x,y)
            % Constructor:
            c.x = x; c.y = y;
        end
    end
end
```

Properties & Attributes

- We can assign property *attributes* on the same line as the properties keyword
- Common attributes: *Access, GetAccess, SetAccess*
- Typical values: *public, protected, private*

```
classdef  circle
    properties (Attr1  =  val1,  Attr2  =  val2)
        x = 0.0; y = 0.0;
        r = 1.0;
    end
...
```

Access Level

- *public* – unrestricted access (default)
- *protected* – access from class or subclasses
- *private* – access by class members only (not subclasses)

```
classdef  circle
    properties (Access  =  public)
        x = 0.0; y = 0.0;
    end
    properties (Access  =  private)
        r = 1.0;
    end
...
```

Set / Get Access

- *Access* – set both SetAccess and GetAccess to the same value
- Can distinguish between setting and displaying (getting) properties

```
classdef  circle
    properties (SetAccess  =  private, GetAccess = public)
        x = 0.0; y = 0.0;
    end
...
```

Constructor (definition)

- Each class file must contain a *constructor*:
  - Function name is the name of the class
  - Header specifies arguments & return variable

```
classdef  circle
    methods
        function C  = circle(x,y)
            C.x  =  x;
            C.y = y;
        end
    end
…
```

Constructor (purpose)

- Force users to use code provided in the class to initialize (manipulate) objects (user can't mess up)
- Do error checking inside constructor

```
classdef  circle
    methods
        function C  = circle(x,y)
            if nargin == 2
                C.x  =  x; C.y = y;
            end
        end
    end
```

Instance Method

- Almost identical to any function, except first parameter in header must be a reference to instance itself

```
classdef  circle
    methods
        function  self = calc_area(self)
            self.area  =  pi*self.radius^2;
        end
    end
...
```

Calling the Methods

- Constructor call syntax: obj_ref = cname(args)
- Instance method syntax: omit the ref to the object itself—since the reference is specified before the method name using the dot notation.

```
CR  = circle(1,2);              % constructor
CR = CR.calc_area();                % instance method
```

Overloading

- Change behavior of built-in function for an object of a class by specifying an instance method with same name
- Example: 'disp' function

```
function  disp(self)
        if  isempty(self)
            % print almost nothing
        else
            % print properties
        end
end
```

Useful Built-in Functions

- All *nonabstract* classes have a static method named *empty* that creates an empty array of the same class
- *isempty:* return true if argument is an empty array
- *class:* return string specifying class of argument

```
C  = circle(1,2);
TF  =  isempty(C);
cname  =  class(C);
disp(C);
```

Inheritance (subclass)

- To inherit from a parent class use “<“
- ‘Left’ class is a subclass of ‘right’ class

```
classdef  circle  <  shape
    properties
        x = 0.0; y = 0.0;
    end
    methods
        …
    end
end
```

Inherited Methods

- A subclass object inherits the superclass' properties and methods that have protected or public accessibility
- Inherited properties and methods can be accessed in the subclass as though they were locally defined in the subclass

Value vs. Handle Classes

- Type of class you need depends on desired behavior
- Value class:
  - represent entities that do not need to be unique
  - Example: polynomial data type
- Handle class:
  - Dynamic properties
  - Create a reference to the data, not copy
  - Example: phone book entry accessed by multiple applications

Value vs. Handle (example)

```
p  =  polynomial( [1 0 -2 -5] );          % poly = value class
p2  =  p;
p.Coefficients  =  [2 3 -1 -2 -3];
p2.Coefficients
          %  ans = 1 0 -2 -5

e = employee('Fred Smith','QE');          % emp = handle class
e2 = e;                                   % Copy handle object
transfer(e,'Engineering')
e2.Department
          %  ans = Engineering
```

Handle Class

- *Handle*: an object that references its data indirectly
- Construct a handle: creates an object with storage for property values, return handle
- Assign/pass the handle: copies the handle, but not the underlying data

```
classdef circle < handle
    properties
    end
    methods
    end
end
```

Outline (part 2)

- Two ways to specify classes with respect to folders:
  - Create a single, self-contained class definition file in some folder
  - Distributing a class definition to multiple files in some @classname folder
- Which approach?
  - Simple class, few functions → single file
  - Large class, lots of functionality → class folder
- Constructor
- Overloading

@folder Class

- Class definition distributed to multiple files
- Creating the class:
  - Create directory @classname
  - Add file classname.m to the directory (the constructor)
  - Add file display.m to the directory (to display variable)
  - Add any other functions (.m files) to implement class

Constructor

- Constructor should handle these input cases:
  - No arguments
  - Variable of the same class
  - Data passed as arguments
- Constructor rules:
  - Check input data for errors
  - To create class variable, define a struct and set field names appropriately. Then, call *class* function to convert struct to class variable
  - Struct fields should be created in the same order inside the constructor

Constructor (example)

```
function  obj = film(varargin)
    if  nargin == 1
        obj = varargin{1};
    else
        obj.name = varargin{1};
        obj.year = varargin{2};
    end
    obj = class(obj, 'film');
end
```

Display Function (example)

```
function display(obj)

disp(obj.name)
disp(obj.year)
%  Can access field names of object inside method
%    functions, but not outside
% 'methods film' to see all class methods
```

Overloading

- Overloading operator: redefine internal rules for an operation
- To support overloading of operators, each operator corresponds to a function name (some examples):

A + B	plus (A,B)
A – B	minus (A,B)
A == B	eq (A,B)
A < B	lt (A,B)
A | B	or (A,B)

Overloading (example)

```
function  TF = eq(F1,F2)
     if  lower(F1.name) == lower(F2.name)
        TF = 1;
     else
        TF = 0;
     end
```