# CS 1112 MATLAB OOP Syntax Summary

**2012 Fall**

## Class File

A class file begins with the keyword classdef and ends with the end keyword. It also contains properties and methods.

```
classdef classname < handle
% The header can be generalized to "classdef classname < superclassname".
% The "< handle" here is to say that this class specifies a handle object,
% and that "classname" is a subclass of class "handle".
    properties
        ...
    end
    methods
        ...
    end
end
```

## Properties and Attributes

Properties of a class in the class file are listed in the properties section:

```
properties
% each property is a separate line
    left
    right
end
```

```
properties
% properties in one line separated by ";"
    left; right
end
```

They can also take on initial values or set to be of specific types:

```
properties
    city=''; % the default value of "city" is an empty string
    temps= Interval.empty(); % "temps" is an empty array of class Interval
    precip
end
```

All properties of a class can have any of the three attributes: public, private, and protected:

```
properties (Access = protected)
% "Access" includes both "GetAccess" and
% "SetAccess"
    left; right
end
```

```
properties (GetAccess=public,SetAccess=private)
% Separate attribute assignments are separated
% by a ","
    left; right
end
```

Within the same class file, you can also define properties that have different attributes:

```
properties (Access=public)
% "left" and "right" are public properties
     left; right
end
properties (Access=private)
% "cost" and "rate" are private properties
     cost; rate
end
```

## Constructor

Each class file must contain a constructor, which is a function whose name is the name of the class. The constructor header specifies a non-zero number of arguments, and a return variable—the object handle:

```
methods
     function Inter = Interval(lt, rt) % arguments are "lt" and "rt"
     % The return variable "Inter" is an Interval object handle; we need it
     % to access the "left" and "right" properties in order to assign them values.
          Inter.left=lt;
          Inter.right=rt;
     end
end
```

Sometimes a constructor is called with zero or fewer arguments. The constructor can handle such a call with the help of nargin, a built-in command that returns the number of function input arguments passed:

```
properties
     left; right
end

methods
     function Inter = Interval(lt, rt) % arguments are "lt" and "rt"
          if nargin == 2
               Inter.left=lt;
               Inter.right=rt;
          end
          % If nargin is not 2, constructor ends without executing the assignment statements.
          % "Inter.left" and "Inter.right" get any default values defined under properties.
          % In this case the default property values are [] (type double)
     end
end
```

## Instance Method

Defining an instance method is very much like defining any function, except the first parameter in the header must be a reference to the instance itself:

```
function scale(self, f) % the "scale" function has two parameters:
%   self is the handle of the object itself--the object that is running this method.
%   f is the factor by which to scale this Interval object.
% Note how "self" is used to call this instance's properties ("left" and "right") and
% method ("getWidth").
     w= self.getWidth();  % Assume instance method getWidth is defined
     self.right= self.left + w*f;
end
```

```
function tf = isIn(self, other) % the "isIn" function has two parameters, both are
% Interval handles:
%   self is the handle of the object itself--the object that is running this method.
%   other is the handle to another Interval object.
% Note how "self" is used to call this instance's properties and "other" is used to access
% the other Interval object's properties.
     tf= self.left>=other.left && self.right<=other.right;
end
```

## Calling the Constructor

A constructor call follows this syntax: *objectreference = classname(arguments)*. Using the Interval class with properties "left" and "right" as an example:

```
r = Interval(4,6); % "r" stores the handle of an Interval object with "left" 4 and "right" 6
```

## Calling an Instance Method

Calling an instance method is very much like calling any function, except we **omit** the argument to the first parameter—the reference to the object itself—since the reference is specified **before** the method name using the dot notation. For example, a valid call to the "scale" function defined above is:

```
r = Interval(4,6);
r.scale(5);  % "5" is the argument for the 2nd parameter in the function header ("f").
             % Argument for the first parameter ("self") is absent because it's the
             % same as "r", the owner of the method as indicated with the dot notation.
```

A valid call to the "isIn" instance method defined above is:

```
r = Interval(4,6);
q = Interval(5,9);
yesno = r.isIn(q); % "q" is the argument to the 2nd parameter in the function header ("other").
                   % Argument for the first parameter ("self") is absent because it's the
                   % same as "r", the owner of the method as indicated with the dot notation.
```

The dot notation explicitly specifies **whose** method to call, i.e., the owner of the instance method. An alternative syntax for calling an instance method is the syntax for calling any function, specifying the arguments for all parameters in the function header: `yesno = isIn(r,q)`
With this syntax, the owner of the method is not explicitly given so Matlab chooses the "isIn" method of one of the parameters. This is NOT recommended.

## Extending a Class

Class inheritance can be specifed in the class file header using this syntax: *classdef* *classname* < *superclassname*
Multiple inheritance can be achieved in a similar fashion: *classdef* *classname* < *superclassname1* & *superclassname2* ...
(Note: multiple inheritance is not studied in CS1112.)

```
% "TrickDie" is a subclass of "Die"
classdef TrickDie < Die
    ...
end
```

```
% "Pattern" inherits from two superclasses
classdef Pattern < Shape & Color
    ...
end
```

## Calling Superclass Constructor

When defining a subclass, the constructor of the subclass must explicitly call the constructor of its superclass (**if the superclass constructor requires arguments**, otherwise Matlab will implicitly call the superclass constructor with zero argument). Call the superclass constructor **before** assigning values to the subclass' properties. Call the superclass constructor with this syntax: *objectreference = objectreference@superclassname(arguments)*

```
% Suppose class "Child" is a subclass of "Parent".
% Here's is the constructor method of "Child":
function obj = Child(argC, argP)
% the "Child" constructor must start with a call to the "Parent" constructor
% and pass it the arguments required by the "Parent" constructor ("argP").
    obj = obj@Parent(argP);
    obj.propC = argC;
end
```

The call to the superclass constructor **cannot be conditional**.

## Calling an Inherited Superclass Method

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.

```
% Suppose class "TrickDie" inherits from class "Die".  Class "Die" includes the instance
% methods "getSides" and "setTop" while class "TrickDie" includes the instance methods
% "getWeight" and "getFavoredFace".
% Here is instance method roll defined in class TrickDie:
function roll(self)
    % This function calls four different instance methods...
    %   Methods "getWeight" and "getFavoredFace" are locally defined in this class ("TrickDie"),
    %     so they can be called in any "TrickDie" instance methods.
    %   Methods "getSides" and "setTop" are defined in superclass "Die", but since their
    %     accessibility are "public" and "protected", respectively, they can be called in
    %     any TrickDie instance method AS THOUGH they are locally defined.
    face= ceil(rand*(self.getSides()+self.getWeight()-1));
    if face>self.getSides()
        face= self.getFavoredFace();
    end
    self.setTop(face)
end
```

## Calling an Overridden Superclass Method

If the subclass overrides a method of its superclass, and the subclass object wishes to call the superclass' version of that method, a function call needs to follow the syntax: *methodname@superclassname(arguments)*.

```
% Suppose both "TrickDie" and "Die" have an instance method defined as "disp(self)"
% and that "TrickDie" is a subclass of "Die".
% Here is the instance method "disp" in "TrickDie" class:
function disp(self)
    fprintf('tricky ')
    disp@Die(self) % "TrickDie" object wants to call the "disp" method of "Die".
end
```

## Determining Which Version of a Method is Used

If a subclass overrides a superclass' method, there are two different versions of the same method: the superclass' version and the subclass' version. The **class (type) of the object** determines which version is used:

```
% Both the "Die" and "TrickDie" classes define an instance method "roll".
a= Die(6);  % "a" is a "Die" handle; its "sides" property has the value 6.
b= TrickDie(5,9,6);  % "b" is a "TrickDie" handle; it's "sides" property has the value 6,
                     % "favoredFace" property has the value 5, and "weight" property has
                     % the value 9.
a.roll()  % Calls the version of "roll" defined in "Die" since "a" has type "Die"
b.roll()  % Calls the version of "roll" defined in "TrickDei" since "b" has type "TrickDie"
```

## Useful Built-in Functions: isempty, class

In addition to the aforementioned built-in function empty, here are some related and useful built-in functions: isempty and class.

isempty returns true if its argument is an empty array of any type:

```
Inter = Interval.empty();
yesno = isempty(Inter); % "yesno" will be 1
```

```
Inter2 = Interval(4,6);
yesno2 = isempty(Inter2); % "yesno2" will be 0
```

```
a = [];  b = 3;
yesno = isempty(a); % "yesno" will be 1
yesno = isempty(b); % "yesno" will be 0
```

```
c = '';  d = 'hi';
yesno = isempty(c); % "yesno" will be 1
yesno = isempty(d); % "yesno" will be 0
```

class returns a string specifying the class of its argument:

```
td = TrickDie(1,2,6);
classname = class(td); % "classname" will be the string 'TrickDie'
```

Refer to the lecture slides and sample files on the course website for more examples.