• Previous lecture:
  – Introduction to objects and classes
  – Value vs. reference
  – Instantiating an object; accessing its properties and methods

• Today’s lecture:
  – Objects are passed by reference to functions
  – Details on class definition
    • Constructor
    • Methods
    • Attributes for properties and methods

• Announcements:
  – Prelim 2 will be returned on Tues
To specify the properties & methods of an object is to define its class

• An interval has two endpoints
• We may want to perform these actions:
  – scale and shift individual intervals
  – Determine whether two intervals overlap
  – Add and subtract two intervals

```matlab
classdef Interval < handle
    properties
        left
        right
    end
    methods
        function Inter = scale(self, f)
            ...
            end
        function Inter = overlap(self, other)
            ...
            end
        function Inter = add(self, other)
            ...
            end
        ...
    end
end
```
A class file begins with keyword `classdef`:

```
classdef classname < handle
```

The class specifies `handle` objects.

Constructor returns a reference to the class object.

Each instance method’s first parameter must be a reference to the instance (object) itself.

Use keyword `end` for keywords `classdef`, `properties`, `methods`, `function`.

---

**Interval**

```
classdef Interval < handle
% An Interval has a left end and a right end

properties
    left
    right
end

methods
    function Inter = Interval(lt, rt)
% Constructor: construct an Interval object
        Inter.left= lt;
        Inter.right= rt;
    end

    function scale(self, f)
% Scale the interval by a factor f
        w= self.right - self.left;
        self.right= self.left + w*f;
    end
end
```
Simplified Interval class

To create an Interval object, use its class name as a function call:  \( p = \text{Interval}(3,7) \)

\begin{align*}
\text{left} & \quad 3 \\
\text{right} & \quad 7 \\
\text{Interval()} & \\
\text{scale()} & \quad 167.32
\end{align*}

classdef Interval < handle
% An Interval has a left end and a right end

properties
  left
  right
end

methods
  function Inter = Interval(lt, rt)
  % Constructor: construct an Interval obj
  Inter.left= lt;
  Inter.right= rt;
  end

  function scale(self, f)
  % Scale the interval by a factor f
  w= self.right - self.left;
  self.right= self.left + w*f;
  end
end
end
The **constructor** method

To create an Interval object, use its class name as a function call:  \( p = \text{Interval}(3, 7) \)

The **constructor** is a special method whose main jobs are to
- compute the handle of the new object,
- execute the function code (to assign values to properties), and
- return the handle of the object.

Constructor has the name of the class
A handle object is referenced by its handle

```cpp
p = Interval(3, 7);
r = Interval(4, 6);
```

A handle, also called a reference, is like an address; it indicates the memory location where the object is stored.
Executing an instance method

```
r = Interval(4, 6);
r.scale(5)
disp(r.right)  %What will it be?
```

Object is passed to a function by reference

```matlab
r = Interval(4,6);
r.scale(5)
disp(r.right) % updated value
```

Objects are passed to functions by reference. Changes to an object’s property values made through the local reference (self) stays in the object even after the local reference disappears when the function ends.
Non-objects are passed to a function by value

Command Window workspace

v 2 4 1

Function space of scale2

v 2 4 1
f 5

v = [2 4 1];
scale2(v,5)
disp(v) %???

function scale2(v,f)
% Scale v by a factor f
v = v*f;
Non-objects are passed to a function by value

\[ v = [2 \ 4 \ 1]; \]
\[
\text{scale2}(v,5) \]
\[
\text{disp}(v) \quad \text{???} \]

\[
\text{function scale2}(v,f) \\
\% \text{ Scale } v \text{ by a factor } f \\
v = v*f; \]
Non-objects are passed to a function by value

\[
v = [2 \ 4 \ 1];
\]

\[
\text{scale2}(v, 5)
\]

\[
\text{disp}(v) \ % \text{NO CHANGE}
\]

\[
\text{function scale2}(v, f) \\
\% \text{Scale v by a factor f}
\]

\[
v = v \times f;
\]
Objects are passed to a function by reference

```matlab
r = Interval(4,6);
r.scale(5)
disp(r.right) % updated value
```

classdef Interval < handle
    
    methods
    
    function scale(self, f)
        % Scale the interval by a factor f
        w= self.right - self.left;
        self.right= self.left + w*f;
    end
    end
end
```

Non-objects are passed to a function by value

```matlab
v= [2 4 1];
function scale2(v,f)
    % Scale v by a factor f
    v= v*f;
end
scale2(v,5)
disp(v) %NO CHANGE
```
Syntax for calling an instance method:

<reference>.<method>(<arguments for 2nd thru last parameters>)

```matlab
p = Interval(3,7);
r = Interval(4,6);

% Explicitly call
% p’s isIn method
yesno = p.isIn(r);

% Matlab chooses the
% isIn method of one
% of the parameters.
yesno = isIn(p,r);
```

classdef Interval < handle
  :
  methods
  :
  function scale(self, f)
  % Scale self by a factor f
  w = self.right - self.left;
  self.right = self.left + w*f;
  end

  function tf = isIn(self, other)
  % tf is true if self is in other interval
  tf = self.left >= other.left && ... 
  self.right <= other.right;
  end

end
```
Method to find overlap between two Intervals

function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% self is empty Interval.
Compare two intervals

1

2 redRight < blueRight

3

4

5 blueRight < redRight

6
The overlap’s left (OLeft) is the rightmost of the two original lefts
The overlap’s left (OLeft) is the rightmost of the two original lefts

The overlap’s right (ORight) is the leftmost of the two original rights
The overlap’s left (OLeft) is the rightmost of the two original lefts.

The overlap’s right (ORight) is the leftmost of the two original rights.

No overlap if OLeft > ORight.
function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% self is empty Interval.

    Inter= Interval.empty();
    left= max(self.left, other.left);
    right= min(self.right, other.right);
    if right-left > 0
       Inter= Interval(left, right);
    end
end

% Example use of overlap function
A= Interval(3,7);
B= Interval(4,4+rand*5);
X= A.overlap(B);
if isempty(X)
    fprintf(’(%f,%f)
’, X.left,X.right)
end
Overloading built-in functions

• You can change the behavior of a built-in function for an object of a class by implementing a function of the same name in the class definition
• Called “overloading”
• A typical built-in function to overload is \texttt{disp}
  – Specify which properties to display, and how, when the argument to \texttt{disp} is an object
  – Matlab calls \texttt{disp} when there’s no semi-colon at the end of an assignment statement

See \texttt{Interval.m}