• Previous lecture:
  – Introduction to objects and classes

• Today’s lecture:
  – **Defining a class**
    • Properties
    • Constructor and other methods
  – **Objects are passed by reference** to functions

• Announcements:
  – **Project 5** due Friday at 11pm
  – **Optional review sessions** on Saturday 2-3:30pm and Monday 5:30-7pm. Location TBA.
  – **Prelim 2** on Tues at 7:30pm
Simplified Interval class

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

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

**Constructor**, a special method with these jobs:
- compute and return the handle of the new object
- execute the function code (to assign values to properties)

Constructor is the only method that has the name of the class.
A handle object is referenced by its handle

\[
p = \text{Interval}(3, 7); \\
r = \text{Interval}(4, 6);
\]

A **handle**, also called a **reference**, is like an address; it indicates the memory location where the object is stored.
What is the effect of referencing?

```matlab
p = Interval(3,7); % p references an Interval object
s = p; % s stores the same reference as p
s.left = 2; % change value inside object
disp(p.left) % 2 is displayed
```

The object is **not** copied—no new object is created! **s** and **p** both reference the same object.
What is the effect of referencing?

\[ p = \text{Interval}(3,7); \quad \% \text{ } p \text{ references an Interval object} \]
\[ s = p; \quad \% \text{ } s \text{ stores the same reference as } p \]
\[ s.\text{left} = 2; \quad \% \text{ change value inside object} \]
\[ \text{disp}(p.\text{left}) \quad \% \text{ } 2 \text{ is displayed} \]
\[ \text{clear } p \quad \% \text{ get rid of } p \text{ from memory} \]

The object still can be accessed through \( s \).
In contrast, structs are stored by value ...

P.x=5; P.y=0; % A point struct P
Q=P; % Q gets a copy of P--Q is ANOTHER % struct with same field values
Q.y=9; % Changes Q’s copy only, not P’s
disp(P.y) % What is displayed?

A: 0    B: 9    B: Something else
In contrast, structs are stored by value ...

P.x=5; P.y=0;  % A point struct P
Q=P;           % Q gets a copy of P--Q is ANOTHER
               % struct with same field values
Q.y=9;         % Changes Q’s copy only, not P’s
disp(P.y)      % 0 is display

In fact, storing-by-value is true of all non-handle-object variables. You already know this from before ...

a=5;
b=a+1;     % b stores the value 6, not
           % some “definition” a+1
a=8;       % Changing a does not change b
disp(b)    % 6 is displayed
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
Calling an object’s method (instance method)

```
p = Interval(3,7);
r = Interval(4,6);
r.scale(5)
```

The owner of the method to be dispatched

Syntax:

```
<reference>.<method>(<arguments for 2^{nd} thru last parameters>)
```
Executing an instance method

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

1st parameter (self) references itself, i.e., its own handle. It gets what's in `r`
Executing an instance method

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

```
function Inter = Interval(lt, rt)
    Inter.left= lt;
    Inter.right= rt;
end

function scale(self, f)
    w= self.right - self.left;
    self.right= self.left + w*f;
end
```
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 is deleted when the function ends.
Non-objects are passed to a function by value
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];
scale2(v, 5)
disp(v) % NO CHANGE

function scale2(v, f)
% Scale v by a factor f
v = v * f;

Non-objects are passed to a function by value
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];
scale2(v,5)
disp(v)  %NO CHANGE
function scale2(v,f)
    % Scale v by a factor f
    v= v*f;
```
Syntax for calling an instance method:

<reference>.<method>(<arguments for 2<sup>nd</sup> 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);
```

```matlab
classdef Interval < handle
    :
    methods
    :
    function scale(self, f)
        w= self.right - self.left;
        self.right= self.left + w*f;
    end

    function tf = isIn(self, other)
        tf= self.left>=other.left && ...
            self.right<=other.right;
    end
end
end
```
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. redRight < blueRight

2. redRight < blueRight

3.

4. blueRight < redRight

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)\n’, X.left,X.right)
end
A class file has the name of the class and 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 `classdef`, `properties`, `methods`, `function`.

```
function Inter = Interval(lt, rt)
    Inter.left= lt;
    Inter.right= rt;
end
```

% Constructor: construct an Interval object

```
function scale(self, f)
    w= self.right - self.left;
    self.right= self.left + w*f;
end
```

% Scale the interval by a factor f

This file’s name is `Interval.m`