• 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:
  – Prelim 2 tonight, 6:30pm EDT
  – Project 5 due NET next Thurs
    • Check website for updates
Prelim reminders

- **In-person (Barton Hall)**
  - Use West entrance
  - Know your seat number (see CMS); find on map
  - Arrive early (6:15)

- **Online**
  - Charge/plug in your devices
  - Close all background apps
  - Gather blank paper
  - Arrive early
Prelim 2 notes

• Tests problem-solving skills
  – You won’t have seen these problems before

• Take systematic approach:
  – Catalog inputs
  – Draw desired outputs
  – Work example by hand, then find (index) patterns
  – Decompose problem; look for patterns; tackle one req at a time

• Grading policy will be adjusted for this semester’s circumstances
  – Treat raw score as diagnostic – how much more room do you have to grow in context of this material?
  – Weights, “curves,” etc. will be adjusted (later) to compute appropriate final grade
  – Will provide estimator after P5 is returned
Programming patterns for Prelim 2

while loop: while in barrel search
End while search
Advance to next element

Linear Search

Threading Multitasking

Open

neighborhood iteration
check whether found
end while

And iterate, etc.

While it's a foot

Bear it to stop

Four
Quiz: Object-oriented vocabulary

Which of the following is incorrect?

A – **Methods** are functions that define a class’s behavior

B – Variables store **handles** to objects, so two different variables may reference the same object

C – **Classes** are instances of objects, each with their own copy of property values

D – **Constructors** return handles to newly allocated objects
Every object (instance) contains every “instance variable” and every “instance method” defined in the class. Every object has its own handle.
The **constructor** method

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

```matlab
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
```

**Constructor**, a special method with these jobs:
- Automatically compute the handle of the new object; the handle must be returned.
- 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.
Value vs. reference

**Arrays**

c = { [3, 1] };
ap = c{1};
ap(2) = 4;
disp(c{1}(2))

c: ![Array 1]
ap: ![Array 2]

**Object handles**

c = { Pair(3, 1) };
ap = c{1};
ap.x = 4;
disp(c{1}.y)

c: ![Object 1]
ap: ![Object 2]

```plaintext
classdef Pair < handle
    properties
        x
        y
    end
end
```
Syntax for calling an instance method

```matlab
r = Interval(4, 6);
r.scale(5)
```

Reference of the object whose method is to be dispatched

Method name

Argument for the second parameter specified in function header (f). Argument for first parameter (self) is absent because it is the same as r, the owner of the method
Calling an object’s method (instance method)

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

Syntax:
<reference>.<method>(<arguments for 2\textsuperscript{nd} thru last parameters>)
Executing an instance method

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

```matlab
r
```

```
A: 5
B: 6
C: 14
D: 30
```

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```
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

```matlab
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
```

Function space of scale
Executing an instance method

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

```matlab
% Function space of scale

<table>
<thead>
<tr>
<th>self</th>
<th>177.54</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>5</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
</tr>
</tbody>
</table>
```

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

properties
    left
    right
end

definitions
    function Inter = Interval(lt, rt)
        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
```
Object is passed to a function by reference

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 is deleted when the function ends.
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
\[ v = [2 4 1]; \]
\[ \text{scale2}(v, 5) \]
\[ \text{disp}(v) \]  

% ???
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

\[
\begin{align*}
\text{r} &= \text{Interval}(4,6); \\
r.\text{scale}(5) \\
\text{disp}(r.\text{right}) & \quad \% \text{ updated value}
\end{align*}
\]

classdef Interval < handle
  :
  methods
    :
    function scale(self, f)
      % Scale the interval by a factor f
      \text{w} = \text{self.right} - \text{self.left};
      \text{self.right} = \text{self.left} + \text{w} \times f;
    end
  end
end

Non-objects are passed to a function by value

\[
\begin{align*}
\text{v} &= [2 \ 4 \ 1]; \\
\text{scale2}(\text{v},5) \\
\text{disp}(\text{v}) & \quad \% \text{NO CHANGE}
\end{align*}
\]

function scale2(\text{v},f)
  % Scale \text{v} by a factor \text{f}
  \text{v} = \text{v} \times f;
end
Syntax for calling an instance method:

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

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

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

```matlab
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
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
% Inter is empty array of class Interval.
Compare two intervals

1

2
redRight < blueRight

3

4
blueRight < redRight

5

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
Implement overlap method
function Inter = overlap(self, other)
% Inter is overlapped Interval between self
% and the other Interval. If no overlap then
% Inter is empty array of class 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)f\n', X.left,X.right)
end