• 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:
  – **Test 2A** due today 4:30pm EDT
    • Consulting hours, Piazza resume afterwards
  – **Project 5** released, due next Thurs
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
Multiple **Interval** 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 = Interval(3,7)`

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] };
a = c{1};
a(2) = 4;
disp(c{1}(2))

c: 3 1
a: 3 1

Object handles

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

c: 167.32
a: 167.32

classdef Pair < handle
  properties
    x
    y
  end
end

167.32
x 3
y 1
Syntax for calling an instance method

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

```plaintext
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
```

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)

\[
p = \text{Interval}(3,7); \\
\text{r} = \text{Interval}(4,6); \\
\text{r} \text{.scale}(5)
\]

Syntax:

\[\text{<reference>}.\text{<method>}(\text{<arguments for 2}\text{nd thru last parameters>})\]
Executing an instance method

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

```
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
```

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

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

```
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
```

```
177.54
```

```
Function space of scale

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

<table>
<thead>
<tr>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>left</th>
<th>right</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

```

```
177.54
```

```
177.54
```
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 is deleted when the function ends.
v = [2 4 1];
scale2(v,5)
disp(v)  %???

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

\begin{verbatim}
function scale2(v, f)
\%
Scale v by a factor f
v = v * f;
\end{verbatim}

Non-objects are passed to a function by value
Objects are passed to a function by reference

Non-objects are passed to a function by value
Syntax for calling an instance method:

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

defInterval(3,7);
\begin{align*}
\text{r} &= \text{Interval}(4,6); \\
\text{yesno} &= \text{p}.\text{isIn}(\text{r}); \\
&\quad \text{% Explicitly call} \\
&\quad \text{% p’s isIn method} \\
\text{yesno} &= \text{isin}(\text{p},\text{r}); \\
&\quad \text{% Matlab chooses the} \\
&\quad \text{% isIn method of one} \\
&\quad \text{% of the parameters.}
\end{align*}

\textit{Better!}

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

\begin{align*}
function\ &\text{tf} = \text{isin}(\text{self}, \text{other}) \\
&\quad \% \text{tf is true if self is in other interval} \\
&\quad \quad \text{tf} = \text{self.left} \geq \text{other.left} \&\& \ldots \\
&\quad \quad \quad \text{self.right} \leq \text{other.right}; \\
end
\end{align*}

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)
', X.left,X.right)
end
A class file has the name of the class and begins with keyword `classdef`:

```
classdef classname < handle
```

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`.

This file’s name is `Interval.m`
Overriding 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 “overriding” (called “overloading” in Matlab documentation)

• A typical built-in function to override is `disp`
  – Specify which properties to display, and how, when the argument to `disp` is (a reference to) an object
  – Matlab calls `disp` when there’s no semi-colon at the end of an assignment statement

See Interval.m
An “array of objects” is really an …

array of references to objects

```
>> A = Interval(3,7);
>> A(2) = Interval(4,6);
>> A(3) = Interval(1,9);
```
MATLAB allows an array to be appended

\[ v = [3 \ 1 \ 5 \ 9] \]
\[ v(7) = 4 \]

• What happens to \( v(5) \) and \( v(6) \)?

```
3 1 5 9 0 0 4
```

• MATLAB assigns some “default value” to the skipped over components for simple, cell, and struct arrays

• For arrays of objects, you must implement the constructor to handle such a situation