• **Previous lecture:**
  – Objects are passed by reference to functions
  – Details on class definition (constructor, instance method)

• **Today’s lecture:**
  – Overloading methods
  – Array of objects
  – Class reuse

• **Announcements:**
  – Discussion via Zoom – please attend
  – Test 2A feedback on Gradescope. Learn from the exam by **re-doing** the problems—don’t just read the solutions (to be posted later)
  – Showcase on Piazza – vote for your favorites!
An “array of objects” is really an array of references to objects.

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

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

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

\[
\begin{array}{cccccc}
3 & 1 & 5 & 9 & 0 & 0 & 4
\end{array}
\]

• MATLAB assigns some “default value” to the skipped over components for arrays

• For arrays of objects, you must implement the constructor to handle such a situation
Constructor needs to be able to handle a call with no arguments

```plaintext
>> A = Interval(3,7); % Array of length 1
>> A(2) = Interval(4,6); % Array of length 2
>> A(3) = Interval(1,9); % Array of length 3
>> A(5) = Interval(2,5); % Array of length 5
```

Error!

- Interval constructor we have so far requires two parameters:
  ```plaintext
  function Inter = Interval(lt, rt)
  ```
- User specified two arguments as required for A(5), but...
- Matlab has to assign A(4) “on its own” by calling the constructor, but no arguments get passed → Error!
Function overloading I

Problem: “default construction” passes 0 args, but our constructor has 2 input params

Want a function that performs the same task for different numbers of inputs

- **MATLAB’s solution**: accept all possible arguments, then ask how many we got

Examples

- `rand()`, `rand(2)`, `rand(1, 3)`
- `max(4, 3)`, `max([6 7 5])`
- `plot(x, y)`, `plot(x, y, 'm-*')`, `plot(x, y, v, w)`
- `Interval(4, 6)`, `Interval()`
Constructor that handles variable number of args

- When used inside a function, **nargin** evaluates to the number of arguments that were passed
Constructor that handles variable number of args

- When used inside a function, `nargin` evaluates to the number of arguments that were passed.
- If `nargin ≠ 2`, constructor ends without executing the assignment statements. Then `Inter.left` and `Inter.right` get any default values defined under properties. In this case the default property values are `[]` (type `double`).

```matlab
classdef Interval < handle
    properties
        left
        right
    end
    methods
        function Inter = Interval(lt, rt)
            if nargin == 2
                Inter.left = lt;
                Inter.right = rt;
            end
        end
    end
    ... end
end```

Default values

• Default property value: empty double array `[]`

• Within an array:
  – Default double: 0
  – Default char: null (char(0), but looks like a space in MATLAB)
  – Default cell: empty cell `{}`
  – Default object: call constructor with no arguments
    • Advantage of bundling behavior with data

Later: customizing default property values in objects
Function overloading II (arguably “overriding”)  

Want to customize an existing function for new classes  

- MATLAB’s solution: define a method in the class with the same function name  

Examples:  
- disp  
- plot  
- Operators!
If a class defines an object that may be used in an array…

- **Constructor must be able handle a call that does not specify any arguments**
  - Use built-in command `nargin`, which returns the number of function input arguments passed

- **The overridden `disp` method, if implemented, should check for an input argument that is an array and handle that case explicitly.**
  - Caution: accessing properties of an entire array produces “comma-separated lists” – an advanced topic
Write a function to create an array of random intervals
A function to create an array of *Intervals*

function inters = intervalArray(n)
% Generate n random Intervals. The left and
% right ends of each interval is in (0,1)
function inters = intervalArray(n)
% Generate n random Intervals. The left and
% right ends of each interval is in (0,1)

for k = 1:n
    randVals = rand(1,2);
    if randVals(1) > randVals(2)
        tmp = randVals(1);
        randVals(1) = randVals(2);
        randVals(2) = tmp;
    end
    inters(k) = Interval(randVals(1), randVals(2));
end

An independent function, not an instance method. See intervalArray.m
Write a function to return the widest interval in an array.
A function to find the widest Interval in an array

function inter = widestInterval(A)
% inter is the widest Interval (by width) in
% A, an array of Intervals

An independent function, not an instance method. See widestInterval.m
A function to find the widest Interval in an array

```matlab
function inter = widestInterval(A)
    % inter is the widest Interval (by width) in
    % A, an array of Intervals

    inter= A(1); % widest Interval so far
    for k= 2:length(A)
        if A(k).right - A(k).left > ...
            inter.right - inter.left
            inter= A(k);
        end
    end
end
```

An independent function, not an instance method. See `widestInterval.m`
A function to find the widest Interval in an array

function inter = widestInterval(A)
% inter is the widest Interval (by width) in
% A, an array of Intervals

inter= A(1); % widest Interval so far
for k= 2:length(A)
    if A(k).getWidth() > inter.getWidth()
        inter= A(k);
    end
end

An independent function, not an instance method. See widestInterval.m
Poll: Functions returning objects

v = [Interval(2, 4) Interval(3, 7)];
w = widestInterval(v);
w.scale(2);
disp(v(2).right)

What is displayed?

A: 7  Intervals were copied when passed into function, so original does not change
B: 7  New Interval was created when returned from function, so original does not change
C: 11 Original is modified through returned value

function inter = widestInterval(A)
    % inter is the widest Interval (by width) in A, an array of Intervals
    inter = A(1);  % widest Interval so far
    for k = 2:length(A)
        if A(k).getWidth() > inter.getWidth()
            inter = A(k);
        end
    end
A weather object can make use of **Intervals** …

- Define a class `LocalWeather` to store the weather data of a city, including monthly high and low temperatures and precipitation
  - Temperature: low and high → an **Interval**
    - For a year → length 12 array of **Intervals**
  - Precipitation: a scalar value
    - For a year → length 12 numeric vector
  - Include the city name: a string
## Weather data file

// Syracuse
// Monthly temperature and precipitation
// Lows (cols 4-8), Highs (col 12-16), precip (cols 20-24)
// Units: English

<table>
<thead>
<tr>
<th>Month</th>
<th>Year</th>
<th>Precip</th>
<th>Lows</th>
<th>Highs</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>31</td>
<td>3.07</td>
<td>16</td>
<td>33</td>
</tr>
<tr>
<td>23</td>
<td>42</td>
<td>3.09</td>
<td>34</td>
<td>55</td>
</tr>
<tr>
<td>43</td>
<td>67</td>
<td>3.86</td>
<td>52</td>
<td>76</td>
</tr>
<tr>
<td>58</td>
<td>80</td>
<td>4.03</td>
<td>56</td>
<td>79</td>
</tr>
<tr>
<td>48</td>
<td>70</td>
<td>3.79</td>
<td>42</td>
<td>58</td>
</tr>
<tr>
<td>31</td>
<td>47</td>
<td>3.19</td>
<td>21</td>
<td>36</td>
</tr>
</tbody>
</table>
Weather data file

//Ithaca
//Monthly temperature and precipitation
//Lows (cols 4-8), Highs (col 12-16), precip (cols 20-24)
//Units: English

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>31</td>
<td>2.08</td>
</tr>
<tr>
<td>17</td>
<td>34</td>
<td>2.06</td>
</tr>
<tr>
<td>23</td>
<td>42</td>
<td>2.64</td>
</tr>
<tr>
<td>34</td>
<td>56</td>
<td>3.29</td>
</tr>
<tr>
<td>44</td>
<td>67</td>
<td>3.19</td>
</tr>
<tr>
<td>53</td>
<td>76</td>
<td>3.99</td>
</tr>
<tr>
<td>58</td>
<td>80</td>
<td>3.83</td>
</tr>
<tr>
<td>56</td>
<td>79</td>
<td>3.63</td>
</tr>
<tr>
<td>49</td>
<td>71</td>
<td>3.69</td>
</tr>
<tr>
<td>NaN</td>
<td>59</td>
<td>NaN</td>
</tr>
<tr>
<td>32</td>
<td>48</td>
<td>3.16</td>
</tr>
<tr>
<td>22</td>
<td>36</td>
<td>2.40</td>
</tr>
</tbody>
</table>

See ithacaWeather.txt, LocalWeather.m

Class LocalWeather should be able to construct an object from such data files, given the known file format.
classdef LocalWeather < handle

properties
    city= "";
    temps= Interval.empty();
    precip
end

methods
    function lw = LocalWeather(fname)
        ...
        ...
        end

end
end

Set property variable that will store an array of objects to the correct type, either under properties or in the constructor.
classdef LocalWeather < handle
    properties
        city='';
        temps=Interval.empty();
        precip=0;
    end
    methods
        function lw = LocalWeather(fname)
            fid= fopen(fname,'r');
            s= fgetl(fid);
            lw.city= s(3:length(s));
            for k= 1:3
                s= fgetl(fid);
            end
            for k=1:12
                s= fgetl(fid);
                lw.temps(k)= Interval(str2double(s(4:8)), str2double(s(12:16)));
                lw.precip(k)= str2double(s(20:24));
            end
            fclose(fid);
        end
        ...
    end %methods
end %classdef

See LocalWeather.m for complete code including use of nargin
Function to show data of a month of \textit{LocalWeather}

\begin{verbatim}
function showMonthData(self, m)
% Show data for month m, 1<=m<=12.

end
\end{verbatim}

Should display which month, the high and low temperatures, and precipitation
function showMonthData(self, m)
% Show data for month m, 1<=m<=12.

mo = {'Jan','Feb','Mar','Apr','May','June', ...
     'July','Aug','Sep','Oct','Nov','Dec'};
fprintf('%s Data
', mo{m})
fprintf('Temperature range: ')
disp(self.temps(m))
fprintf('Average precipitation: %.2f
', ... 
     self.precip(m))
end
Observations about our class `Interval`

- We can use it (create `Interval` objects) anywhere
  - Within the `Interval` class, e.g., in method `overlap`
  - “on the fly” in the Command Window
  - In other function/script files – not class definition files
  - In another class definition

- Designing a class well means that it can be used in many different applications and situations
OOP ideas

- Aggregate variables/methods into an abstraction (a class) that makes their relationship to one another explicit.
- Object properties (data) need not be passed to instance methods—only the object handle (reference) is passed. Useful for large data sets!
classdef Interval < handle

properties
    left
    right
end

methods
    function scale(self, f)
        ...
    end

    function shift(self, s)
        ...
    end

    function Inter = overlap(self, other)
        ...
    end

    function Inter = add(self, other)
        ...
    end

end

end

When an instance method executes, the properties—data—are accessible through the handle (reference). No local copy of the data is needed in the method’s memory space.
Poll: Assigning handles

```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
end

r = Interval(4, 6);
s = r;
s.left = 5;
s = Interval(3, 7);
disp(r.left)
```

What will be displayed?

```
A: 3  B: 4  C: 5  D: 7
```