• Previous lecture:
  – Array of objects
  – Methods that handle a variable number of arguments
  – Using objects of a class in other functions
• Today’s lecture:
  – Using objects of a class in other functions
  – Why use OOP?
  – Attributes for properties and methods (private vs. public)
• Announcements:
  – Discussion this week in classrooms, not the lab
  – Pick up prelim 2 at ACCEL Green Rm during consulting hrs
  – Final exam for both lectures on Dec 7th at 2pm. There will not be a separate exam for lec2 on Dec 15th.
  – Check your final exam schedule now and notify us by emailing Randy Hess (rbh27) if you have an exam conflict.

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 \( \rightarrow \) an Interval
    • For a year \( \rightarrow \) length 12 array of Intervals
  – Precipitation: a scalar value
    • For a year \( \rightarrow \) length 12 numeric vector
    – Include the city name: a string

```matlab
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)));
            end
            lw.precip(k) = str2double(s(20:24));
            fclose(fid);
        end
```

```matlab
//Syracuse
//Monthly temperature and precipitation
//Lows (cols 4-8), Highs (col 12-16), precip (cols 20-24)
//Units:   English
14  31  3.07
16  33  2.96
23  42  3.09
34  55  3.91
43  67  3.86
52  76  4.27
58  80  4.03
56  79  3.95
48  70  3.79
42  58  3.44
31  47  3.19
21  36  2.82
```

```matlab
//Ithaca
//Monthly temperature and precipitation
//Lows (cols 4-8), Highs (col 12-16), precip (cols 20-24)
//Units:   English
15  31  2.08
17  34  2.06
23  42  2.64
34  56  3.29
44  67  3.19
53  76  3.99
58  80  3.83
56  79  3.63
49  71  3.69
NaN 59  NaN
32  48  3.16
22  36  2.40
```

```matlab
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);
                lw.temps(k) = Interval(str2double(s(4:8)), str2double(s(12:16)));
            end
            lw.precip(k) = str2double(s(20:24));
            fclose(fid);
        end
```

```matlab
//Ithaca
//Monthly temperature and precipitation
//Lows (cols 4-8), Highs (col 12-16), precip (cols 20-24)
//Units:   English
15  31  2.08
17  34  2.06
23  42  2.64
34  56  3.29
44  67  3.19
53  76  3.99
58  80  3.83
56  79  3.63
49  71  3.69
NaN 59  NaN
32  48  3.16
22  36  2.40
```

```matlab
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);
                lw.temps(k) = Interval(str2double(s(4:8)), str2double(s(12:16)));
            end
            lw.precip(k) = str2double(s(20:24));
            fclose(fid);
        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. Important for large data sets!

OOP ideas ➔ Great for managing large projects

- 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. Important for large data sets!
- Objects (instances of a class) are self-governing (protect and manage themselves)
  - Hide details from client, and restrict client’s use of the services
  - Provide clients with the services they need so that they can create/manipulate as many objects as they need
Restricting access to properties and methods

- Hide data from “outside parties” who do not need to access that data—need-to-know basis
- E.g., we decide that users of Interval class cannot directly change left and right once the object has been created. Force users to use the provided methods—scale, shift, etc.—to cause changes in the object data
- Protect data from unanticipated user action
- Information hiding is very important in large projects

Constructor can be written to do error checking

```matlab
classdef Interval < handle
    properties
        left
        right
    end
    methods
        function Inter = Interval(lt, rt)
            % Constructor: construct an Interval obj
            % Inter.left = lt; Inter.right = rt;
        end
    end
end
```

Should force users (clients) to use code provided in the class to create an Interval or to change its property values once the Interval has been created.

E.g., if users cannot directly set the properties left and right, then they cannot accidentally “mess up” an Interval.

Attributes for properties and methods

- public
  - Client has access
  - Default
- private
  - Client cannot access

```matlab
classdef Interval < handle
    % An Interval has a left end and a right end
    properties (SetAccess-private, GetAccess-private)
        left
        right
    end
    methods
        function Inter = Interval(lt, rt)
            % Constructor: construct an Interval obj
            % Inter.left = lt; Inter.right = rt;
        end
    end
end
```

Within the class, there is always access to the object's properties, even if private.
Attributes for properties and methods

• **public**
  - Client has access
  - Default
• **private**
  - Client cannot access

% Client code
r = Interval(4, 6);
r.right = 9; % error
r = Interval(4, 14); % OK
r.scale(5); % OK
r = Interval(4, 6);

% Client code
r = Interval(4, 6);
disp(r.right); % error
disp(r.getRight()); % OK

classdef Interval < handle

% An Interval has a left end and a right end
properties (Access=private)
  left; right
end

methods
  function Inter = Interval(lt, rt)
  \( \text{lt} \) = self.left;
  \( \text{rt} \) = self.right;
  end
  
  function setLeft(self, lt)
  \( \text{lt} \) is the interval's left end gets \( \text{lt} \) = self.left;
  end
  
  function setRight(self, rt)
  \( \text{rt} \) is the interval's right end gets \( \text{rt} \) = self.right;
  end

end

end

Public "getter" method

• Provides client the ability to get a property value
• Don't do it unless really necessary! If you implement public setters, include error checking (not shown here).

% Client code
r = Interval(4, 6);
r.right = 9; % error
r.setRight(9); % OK

classdef Interval < handle

% An Interval has a left end and a right end
properties (Access=private)
  left; right
end

methods
  function Inter = Interval(lt, rt)
  \( \text{lt} \) = self.left;
  \( \text{rt} \) = self.right;
  end
  
  function getLeft(self)
  \( \text{lt} \) is the interval's left end gets \( \text{lt} \) = self.left;
  end
  
  function getRight(self)
  \( \text{rt} \) is the interval's right end gets \( \text{rt} \) = self.right;
  end

end

end

Always use available methods, even when within same class

% Client code
A = Interval(4, 7);
disp(A.getRight());

% ... lots of client code that uses
% class Interval, always using the
% provided public getters and
% other public methods ...

OOP ideas → Great for managing large projects

• 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. Important for large data sets!
• Objects (instances of a class) are self-governing (protect and manage themselves)
  – Hide details from client, and restrict client’s use of the services
  – Provide clients with the services they need so that they can create/manipulate as many objects as they need to