• **Previous lecture:**
  – Review cell and struct arrays
  – Introduce objects

• **Today’s lecture:**
  – Introduction to Matlab *handle objects*—plots
  – Introduction to objects and classes

• **Announcements:**
  – Discussion in computer lab this week
  – **Project 5** due Friday at 11pm; extra consulting hours Friday 3:45-6:30pm in ACCEL Green Rm (Carpenter Hall)
  – **Prelim 2** on Tues, Apr 26 at 7:30pm
  – Prelim 2 topics: end with Project 5 and Lecture and discussion of previous week; will NOT include OOP
Object-Oriented Programming

• First design and define the classes (of the objects)
  – Identify the properties (data) and actions (methods, i.e., functions) of each class

• Then create the objects (from the classes) that are then used, that interact with one another
Example class: Rectangle

- Properties:
  - $x_{LL}$, $y_{LL}$, width, height

- Methods (actions):
  - Calculate area
  - Calculate perimeter
  - Draw
  - Intersect (the intersection between two rectangles is a rectangle!)
Example class: Time

• Properties:
  – Hour, minute, second

• Methods (actions):
  – Show (e.g., display in hh:mm:ss format)
  – Advance (e.g., advance current time by some amount)
Matlab supports procedural and object-oriented programming

• We have been writing **procedural programs**—focusing on the algorithm, implemented as a set of functions

• We have used objects in Matlab as well, e.g., graphics

• A **plot** is a “*handle graphics*” object
  - Can produce plots without knowing about objects
  - Knowing about objects gives more possibilities
The **plot** handle graphics object in Matlab

\[ x=...; \quad y=...; \]
\[ \text{plot}(x,y) \quad \text{creates a graphics object} \]

- In the past we focused on the visual produced by that command. If we want the visual to look different we make *another* plot.
- We can actually hang on to the graphics object—store its “*handle*”—so that we can later make changes to *that* object.
A “movie” of a ball tracing a curve—3 frames

Without considering objects: draw the three “frames” independently, each time drawing both line and ball.
A “movie” of a ball tracing a curve—3 frames

Without considering objects: draw the three “frames” independently, each time drawing both line and ball.

Make use of objects: make the line and ball two independent objects that we can hold on, i.e., we hold on to their individual handles. Leave the line alone, but at each iteration give the ball new x-, y-values.
Objects of the same class have the same properties

\texttt{x = 1:10;}
\texttt{\% Two separate graphics objects:}
\texttt{plot(x, \sin(x), \ 'k-\')}
\texttt{plot(x(1:5), 2.\textasciitilde{x}, \ 'm-\*')}

- Both objects have some x-data, some y-data, some line style, and some marker style. These are the properties of one kind, or \texttt{class}, of the objects (plots)
- The values of the properties are different for the individual objects

See \texttt{demoPlotObj.m}
Object-Oriented Programming

• First design and define the classes (of the objects)
  – Identify the properties (data) and actions (methods, i.e., functions) of each class

• Then create the objects (from the classes) that are then used, that interact with one another
Class Interval

• An interval has two properties:
  – left, right

• Actions—methods—of an interval include
  – Scale, i.e., expand
  – Shift
  – Add one interval to another
  – Check if one interval is in another
  – Check if one interval overlaps with another

See demoInterval0.m
Class Interval

- An interval has two properties:
  - left, right
- Actions—methods—of an interval include:
  - Scale, i.e., expand
  - Shift
  - Add one interval to another
  - Check if one interval is in another
  - Check if one interval overlaps with another

To specify the properties and actions of an object is to define its class.
Given class Interval (file Interval.m) …

% Create 2 Intervals, call them A and B
A = Interval(2,4.5)
B = Interval(-3,1)

% Assign another right end point to A
A.right = 14

% Half the width of A (scale by 0.5)
A.scale(.5)

% See the result
disp(A.right)  % show value in right property
disp(A)        % show all property values
disp(B)

Observations:
• Each object is referenced by a name.
• Two objects of same class has same properties (and methods).
• To access a property value, you have to specify whose property (which object's property) using the dot notation.
• Changing the property values of one object doesn't affect the property values of another object.
An Interval object

The "handle" or "reference" of the object

left 3
right 7

Interval() scale() shift() overlap() add()

The "constructor" method

An object is also called an "instance" of a class. It contains every property, "instance variable," and every "instance method" defined in the class.
Multiple `Interval` objects

Every object (instance) contains every “instance variable” and every “instance method” defined in the class. Every object has a unique `handle`.
Simplified Interval class

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

```
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
```
The **constructor** method

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

```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
end
```

The **constructor** is a special method whose main jobs are to

- compute the handle of the new object,
- execute the function code (to assign values to properties), and
- return the handle of the object.

Constructor 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's the effect of storing data "by reference"? Let's experiment in Matlab
What is the effect of referencing?

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

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

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

The object still can be accessed through \( s \).