• Previous lecture:
  – Structure & structure array
• Today's lecture:
  – Introduction to objects and classes
• Announcements:
  – Discussion this week in classrooms, not UP B7
  – P5 due Thursday at 11 pm; late submission accepted until Friday 11 pm with a late penalty of 1 point
  – Prelim 2 on Tues, Apr 16, at 7:30 pm

How do you assign to \( g \) the green-color component of disk \( D \)?

\[
D = \text{struct}('xc',3.5, 'yc',2, ... \\
'x',1.0, 'c',[.4 .1 .5])
\]

**A:** \( g = D.g \);

**B:** \( g = D.c.g \);

**C:** \( g = D.c.2 \);

**D:** \( g = D.c(2) \);

**E:** other

A structure's field can hold a structure

\[
A = \text{MakePoint}(2,3) \\
B = \text{MakePoint}(4,5) \\
L = \text{struct}('P',A, 'Q',B)
\]

• This could be used to represent a line segment with endpoints \( P \) and \( Q \), for instance
• Given the MakePoint function to create a point structure, what is \( x \) below?

\[
x = L.P.y;
\]

**A:** 2  
**B:** 3  
**C:** 4  
**D:** 5  
**E:** error

Different kinds of abstraction

• Packaging procedures (program instructions) into a function
  – A program is a set of functions executed in the specified order
  – Data is passed to (and from) each function
• Packaging data into a structure
  – Elevates thinking
  – Reduces the number of variables being passed to and from functions
• Packaging data, and the instructions that work on those data, into an object
  – A program is the interaction among objects
  – Object-oriented programming (OOP) focuses on the design of data-instructions groupings

A card game, developed in two ways

• Develop the algorithm—the logic—of the card game:
  – Set up a deck as an array of cards. (First, choose representation of cards.)
  – Shuffle the cards
  – Deal cards to players
  – Evaluate each player's hand to determine winner
• Identify “objects” in the game and define each:
  – Card
    • Properties: suit, rank
    • Actions: compare, show
  – Deck
    • Property: array of Cards
    • Actions: shuffle, deal, get #cards left
  – Hand …
  – Player …
• Then write the game—the algorithm—using objects of the above “classes”

Procedural programming: focus on the algorithm, i.e., the procedures, necessary for solving a problem

Object-oriented programming: focus on the design of the objects (data + actions) necessary for solving a problem

A card game, developed in two ways

• Develop the algorithm—the logic—of the card game:
  – Set up a deck as an array of cards. (First, choose representation of cards.)
  – Shuffle the cards
  – Deal cards to players
  – Evaluate each player's hand to determine winner
• Identify “objects” in the game and define each:
  – Card
    • Properties: suit, rank
    • Actions: compare, show
  – Deck
    • Property: array of Cards
    • Actions: shuffle, deal, get #cards left
  – Hand …
  – Player …
Notice the two steps involved in OOP?

- Define the classes (of the objects)
  - Identify the properties (data) and actions (methods, i.e., functions) of each class
- Create the objects (from the classes) that are then used—that interact with one another

Defining a class ≠ creating an object

- A class is a specification
  - E.g., a cookie cutter specifies the shape of a cookie
- An object is a concrete instance of the class
  - Need to apply the cookie cutter to get a cookie (an instance, the object)
  - Many instances (cookies) can be made using the class (cookie cutter)
  - Instances do not interfere with one another. E.g., biting the head off one cookie doesn't remove the heads of the other cookies

Example class: Rectangle

- Properties:
  - xLL, yLL, width, height
- Methods (actions):
  - Area
  - 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)

Example class: Window (e.g., dialog box)

- Properties:
  - Title, option buttons, input dialog ...
- Methods (actions):
  - show
  - ...

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

```matlab
x=...; y=...; plot(x,y) 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 “hold on” to the graphics object—store its “handle”—so that we can later make changes to that object.

See `demoPlotObj.m`

---

Objects of the same class have the same properties

```matlab
x= 1:10; % Two separate graphics objects: plot(x, sin(x), 'k-') plot(x(1:5), 2.^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 class, of the objects (plots)
- The values of the properties are different for the individual objects

---

To specify the properties & methods of an object is to define its **class**

- An interval has two endpoints
- We may want to perform these actions:
  - scale and shift individual intervals
  - Determine whether two intervals overlap
  - Add and subtract two intervals

---

A handle object is **referenced** by its handle

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

A handle, also called a reference, is like an address; it indicates the memory location where the object is stored.

---

Simplified Interval class

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

---

What is the effect of referencing?

```matlab
p = Interval(3,7); % p references an Interval object
s = p; % s stores the same reference as p
s.left = 2; % change value inside object
disp(p.left) % 2 is displayed
```

The object is not copied—no new object is created! `s` and `p` both reference the same object.
By contrast, structs are stored by value ...

```
P.x=5; P.y=0;  % A point struct P
Q=P;        % Q gets a copy of P--copy
   % all the values in the fields
Q.y=9;       % Changes Q's copy only, not P's
disp(P.y)   % 0 is display
```

In fact, storing-by-value is true of all non-handle-object variables. You already know this from before ...

```
a=5;
b=a+1;  % b stores the value 6, not
   % the "definition" a+1
a=8;   % Changing a does not change b
disp(b) % 6 is displayed
```