- Previous lecture:
  - Why use OOP?
  - Attributes for properties and methods
- Today's lecture:
  - Inheritance: extending a superclass
  - Overriding methods in superclass
  - New topic: Recursion
- Announcement:
  - Project 6 due on Dec 4th (Thurs) at 11pm.
  - Remember academic integrity! We will check all submissions using MOSS.
  - Final exam on Wednesday, Dec 17th, at 7pm. Email Randy Hess (rbh27) now if you have an exam conflict. Specify your entire exam schedule (course numbers/contacts and the exam times). We must have this information by Monday Nov 24th.

A fair die is…

```
classdef Die < handle
 properties (Access=private)
  sides=6;
  top
 end
 methods
  function D = Die(…)   …
  function roll(…)   …
  function disp(…)   …
  function s = getSides(…)   …
  function t = getTop(…)   …
 end
 methods (Access=private)
  function setTop(…)   …
 end
end
```

What about a trick die?

Separate classes—each has its own members

```
classdef Die < handle
  properties (Access=private)
    sides=6;
    top
  end
  methods
    function D = Die(…)   …
    function roll(…)    …
    function disp(…)    …
    function s = getSides(…)   …
    function t = getTop(…)   …
  end
  methods (Access=private)
    function setTop(…)   …
  end
end
```

```
classdef TrickDie < handle
  properties (Access=private)
    sides=6;
    top
    favoredFace
    weight=1;
  end
  methods
    function D = TrickDie(…)   …
    function roll(…)   …
    function disp(…)   …
    function s = getSides(…)   …
    function t = getTop(…)   …
    function f = getFavoredFace(…) …
    function w = getWeight(…)   …
  end
  methods (Access=private)
    function setTop(…)
  end
end
```

Separate classes—each has its own members

```
classdef Die < handle
 properties (Access=private)
  sides=6;
  top
 end
 methods
  function D = Die(…)   …
  function roll(…)   …
  function disp(…)   …
  function s = getSides(…)   …
  function t = getTop(…)   …
 end
 methods (Access=private)
  function setTop(…)   …
 end
end
```

```
classdef TrickDie < handle
 properties (Access=private)
  sides=6;
  top
  favoredFace
  weight=1;
 end
 methods
  function D = TrickDie(…)   …
  function roll(…)   …
  function disp(…)   …
  function s = getSides(…)   …
  function t = getTop(…)   …
  function f = getFavoredFace(…) …
  function w = getWeight(…)   …
 end
 methods (Access=private)
  function setTop(…)
 end
end
```

Can we get all the functionality of Die in TrickDie without re-writing all the Die components in class TrickDie?

```
classdef Die < handle
  properties (Access=private)
    sides=6;
    top
  end
  methods
    function D = Die(…)   …
    function roll(…)   …
    function disp(…)   …
    function s = getSides(…)   …
    function t = getTop(…)   …
  end
  methods (Access=private)
    function setTop(…)   …
  end
end
```

```
classdef TrickDie < handle
```

> **"Inherit" the components of class Die**

```
  properties (Access=private)
    favoredFace
    weight=1;
  end
  methods
    function D = TrickDie(…)   …
    function f =getFavoredFace(…) …
    function w = getWeight(…)   …
  end
end
```
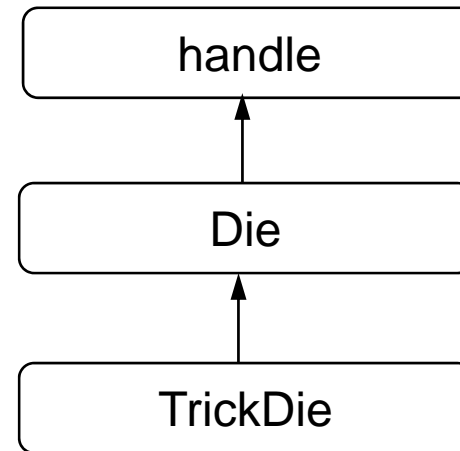
Yes! Make TrickDie a subclass of Die

```matlab
classdef Die < handle
  properties (Access=private)
    sides=6;
    top
  end
  methods
    function D = Die(…)  …
    function roll(…)  …
    function disp(…)  …
    function s = getSides(…)  …
    function t = getTop(…)  …
  end
  methods (Access=protected)
    function setTop(…)  …
  end
end
```

```matlab
classdef TrickDie < Die

  properties (Access=private)
    favoredFace
    weight=1;
  end

  methods
    function D = TrickDie(…)  …
    function f=getFavoredFace(…)…
    function w = getWeight(…)  …
  end

end
```

Inheritance

Inheritance relationships are shown in a *class diagram*, with the arrow pointing to the parent class

```
        ┌──────────────┐
        │    handle    │
        └──────────────┘
               ▲
        ┌──────────────┐
        │     Die      │
        └──────────────┘
               ▲
        ┌──────────────┐
        │   TrickDie   │
        └──────────────┘
```

An ***is-a*** relationship: the child ***is a*** more specific version of the parent. Eg., a trick die *is a* die.

Multiple inheritance: can have multiple parents ← e.g., Matlab
Single inheritance: can have one parent only ← e.g., Java

Inheritance

- Allows programmer to *derive* a class from an existing one

- Existing class is called the *parent class*, or *superclass*

- Derived class is called the *child class* or *subclass*

- The child class *inherits* the (public and protected) members defined for the parent class

- Inherited trait can be *accessed as though it was **locally** defined*

Must call the superclass' constructor

- In a subclass' constructor, call the superclass' constructor before assigning values to the subclass' properties.

- Calling the superclass' constructor cannot be conditional: explicitly make one call to superclass' constructor

See constructor in `TrickDie.m`

```
classdef  Child < Parent

  properties
   propC
  end


  methods

   function  obj = Child(argC, argP)

      obj = obj@Parent(argP)

      obj.propC = argC;
   end

   …
  end

end
```

Which components get "inherited"?

- public components get inherited

- private components exist in object of child class, but cannot be directly accessed in child class $\Rightarrow$ we say they are not inherited

- Note the difference between <u>inheritance</u> and <u>existence</u>!

`protected` attribute

- Attributes dictate which members get inherited

- **`private`**
  - Not inherited, can be *accessed* by local class only
- **`public`**
  - Inherited, can be *accessed* by all classes
- **`protected`**
  - Inherited, can be *accessed* by subclasses

- *Access:* access as though defined locally
- All members from a superclass *exist* in the subclass, but the **`private`** ones cannot be *accessed* directly—can be accessed through inherited (public or protected) methods

td = TrickDie(2, 10, 6);
disp(td.sides)
% disp statement is incorrect because

A Property sides is private.

B Property sides does not
exist in the TrickDie object.

C Both a, b apply

Overriding methods

- Subclass can *override* definition of inherited method
- New method in subclass has the same name (but has different method body)

See method **roll** in **TrickDie.m**

Overridden methods: which version gets invoked?
To create a TrickDie: call the TrickDie constructor, which
calls the Die constructor, which calls the roll method. Which
roll method gets invoked?

```
classdef Die

  …
  function D=Die(…)

    …
    D.roll()
  end


  function roll(self)


  end

  …
end
```

```
classdef TrickDie < Die

  …
  funciton TD=TrickDie(…)

    …
    TD@Die(…);

    …
  end


  function roll(self)


  end

  …
end
```

Overriding methods

- Subclass can *override* definition of inherited method

- New method in subclass has the same name (but has different method body)

- Which method gets used??

  *The **object** that is used to invoke a method determines which version is used*

- Since a TrickDie object is calling method `roll`, the TrickDie's version of `roll` is executed

- In other words, the method most specific to the type (class) of the object is used

Accessing superclass' version of a method

- ## Subclass can override superclass' methods

- ## Subclass can access superclass' version of the method

See method `disp` in `TrickDie.m`

```
classdef  Child < Parent

 properties
  propC
 end

 methods
  ...

  function x= method(arg)

    y= method@Parent(arg);

    x = ... y ... ;
  end

  ...
 end
end
```

Important ideas in inheritance

- Keep common features as high in the hierarchy as reasonably possible

- Use the superclass' features as much as possible

- "Inherited" $\Rightarrow$ "can be accessed as though declared locally"

  (`private` member in superclass *exists* in subclasses; they just cannot be accessed directly)

- Inherited features are continually passed down the line

(Cell) array of objects

- A cell array can reference objects of different classes

```
A{1}= Die();
A{2}= TrickDie(2,10);   % OK
```

- A simple array can reference objects of only one single class

```
B(1)= Die();
B(2)= TrickDie(2,10);   % ERROR
```

- (Assignment to B(2) above would work if we define a "convert method" in class TrickDie for converting a TrickDie object to a Die. We won't do this in CS1112.)

End of Matlab OOP in CS1112

OOP is a concept; in different languages it is expressed differently.

In CS (ENGRD) 2110 you will see Java OOP