

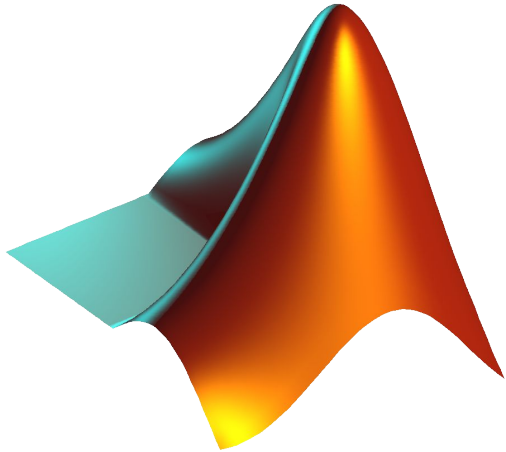
CS 1112 Introduction to Computing Using MATLAB

Instructor: Dominic Diaz

Website:

<https://www.cs.cornell.edu/courses/cs1112/2022fa/>

Today: object-oriented programming -
Inheritance



Agenda and announcements

- Last time
 - Object-oriented programming
 - Private and public properties and methods
 - Intro to inheritance
- Today
 - Object-oriented programming
 - Inheritance
 - Public, private, and protected access
- Announcements
 - Prelim 2 grades released. Review the prelim if you didn't do well!
 - Project 6 released (due Dec 5th)
 - If you need a partner, fill out partner service by Tuesday 11/22
 - Read Insight 14.1 before next lecture
 - Reading for today's lecture is a recap of all OOP that you should know.
 - Course evaluations will happen Monday, November, 28th to Thursday, Dec. 8th.
Fill out for 1 point back on your final exam!

Let's consider a new fair die class

```
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?

Closely related trick die class

```
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 w = getWeight(...) ...
        function f = getFavFace(...) ...
    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 w = getWeight(...) ...
        function f = getFavFace(...) ...
    end

end
```

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

```
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)
```

This class would be the **superclass** or the **parent** class

```
classdef TrickDie < Die
```

< [className] allows us to **inherit** properties and methods from another class!

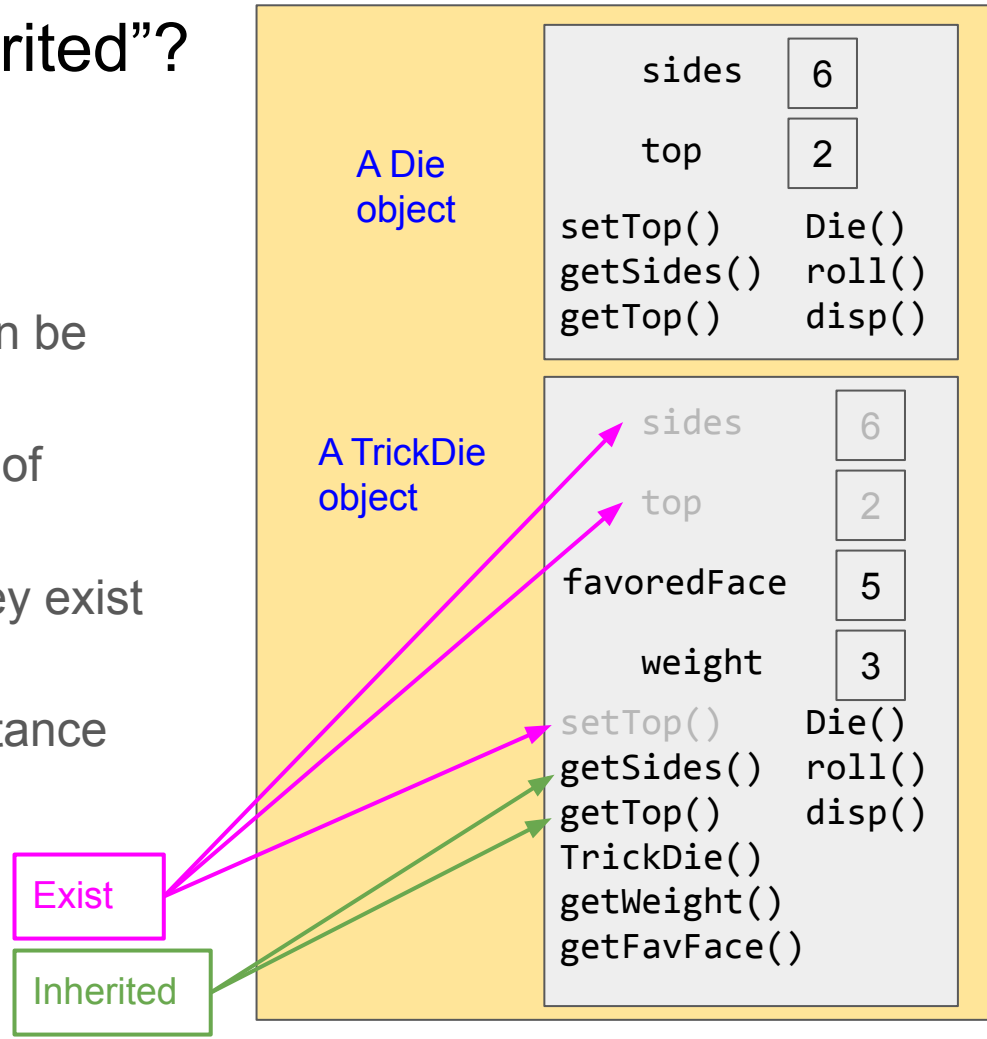
```
properties (Access=private)
    favoredFace
    weight=1;
end
```

```
methods
    function D = TrickDie(...) ...
    function w = getWeight(...) ...
    function f = getFavFace(...) ...
end
```

This class would be the **subclass** or the **child** class

Which components get “inherited”?

- Public components get inherited
 - Properties and methods of the parent class that are public can be accessed in subclass.
- Private components exist in object of child class, but cannot be directly accessed in child class (we say they exist but are not inherited)
- Note the difference between inheritance and existence!



```

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 < Die
    properties (Access=private)
        favoredFace; weight=1;
    end
    methods
        function D = TrickDie(...) ...
        function w = getWeight(...) ...
        function f = getFavFace(...) ...
        function roll(...) ...
    end
end

```

A Die
object

| | |
|------------|--------|
| sides | 6 |
| top | 2 |
| setTop() | Die() |
| getSides() | roll() |
| getTop() | disp() |

A TrickDie
object

| | |
|--------------|--------|
| sides | 6 |
| top | 2 |
| favoredFace | 5 |
| weight | 3 |
| setTop() | Die() |
| getSides() | roll() |
| getTop() | disp() |
| TrickDie() | |
| getWeight() | |
| getFavFace() | |

Protected attribute

- Attributes dictate which members get inherited
- **Private**
 - Not inherited, can only be directly accessed in the classdef in which it's defined
- **Public**
 - Inherited, can be accessed anywhere (in the classdef, in files outside the classdef, and the command window)
- **Protected**
 - Inherited, can be accessed in the classdef in which it's defined and the classdef for all subclasses.
 - Cannot be directly accessed anywhere else

Note: all members (properties and methods) from a superclass exist in the subclass, but the private ones cannot be accessed directly in the subclass—can be accessed through inherited (public or protected) methods.

Let's play with dice in the command window

```
d = Die(6);  
disp(d.top)      %error: top is private  
d.getTop()      %OK
```

```
t = TrickDie(2, 10, 6);  
disp(t.top)      %error: top is private  
                % to class Die
```

```
t.getTop()      %OK  
d.setTop(5)      %error: setTop is  
                % protected (only  
                % available to clasdef  
                % Die and 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=protected)  
        function setTop(...) ...  
    end  
end  
  
classdef TrickDie < Die  
    properties (Access=private)  
        favoredFace; weight=1;  
    end  
    methods  
        function D = TrickDie(...) ...  
        function f = getWeight(...) ...  
        function f = getFavFace(...) ...  
    end  
end
```

Subclasses must call the superclass' constructor

- In a subclass' constructor, call the superclass' constructor before assigning values to the subclass' properties
 - If you don't, MATLAB implicitly calls parent constructor with no inputs
- Calling the superclass' constructor should not be conditional (should not be inside an if-statement)

```
classdef Child < Parent

    properties
        propC
    end

    methods
        function obj = Child(argC, argP)
            obj = obj@Parent(argP);
            obj.propC = argC;
        end
        ...
    end
end
```

See constructor in TrickDie.m

Overriding methods

- Subclasses can override inherited methods
- To override, the method in the subclass has the same name (but has a different method body)

```
classdef Die
    ...

    function D = Die(...)
        ...
        D.roll()
    end

    function roll(self)
        ...
    end

    ...
end

classdef TrickDie < Die
    ...

    function TD = TrickDie(...)
        ...
        TD = TD@Die(...);
        ...
        TD.roll();
    end

    function roll(self)
        ...
    end

    ...
end
```

Overriding methods

- Subclasses can override inherited methods
- To override, the method in the subclass has the same name (but has a different method body)
- How do we determine which method will be used?
 - The object that is used to invoke a method determines which version is used

`% In the command window`

```
d = Die(6);
```

```
d.roll();           % calls the roll method from Die
```

```
t = TrickDie(6, 2, 6);
```

```
t.roll();           % calls the roll method from TrickDie
```

```
t2 = TrickDie(4, 3, 6) % calls disp method from TrickDie  
                        % (if it exists).
```

- The method most specific to the class of the object is used!

Accessing superclass' version of a method

- We've seen that subclasses can override superclass' methods
- Subclasses can still access superclass' version of the method

For this code to work, `methoda` in the parent class must be:

```
public  
protected  
private
```













```
classdef Child < Parent  
  
    properties  
        propC  
    end  
  
    methods  
        ...  
        function x = methoda(arg)  
            y = methoda@Parent(arg);  
        ...  
    end  
end
```

See `method disp` in `TrickDie.m`

Important ideas in inheritance

- Keep common features as high in the hierarchy as reasonably possible
- Use the superclass' features (methods and properties) as much as possible
- “Inherited” from parent class to child class → can be accessed as though defined in the child class itself.
 - Private members in a superclass exist in subclasses but cannot be accessed directly
- Inherited features are continually passed down the line

Where can properties and method be directly accessed?

| | In command window | In external script or function | In class methods | In subclass methods |
|-----------|---|---|--|--|
| public |  |  |  |  |
| private |  |  |  |  |
| protected |  |  |  |  |

SetAccess and GetAccess

```
classdef Schedule < handle
```

```
    properties (SetAccess = private, GetAccess = public)
```

```
        sname = '';
```

```
        window = Interval.empty();
```

```
        eventArray = {};
```

```
    end
```

```
    methods
```

```
        ...
```

```
    end
```

```
end
```

Attribute **SetAccess** allows us to restrict how we are able to **set** properties and methods

```
% in command window  
s = Schedule(a, b, c);  
s.window = b2; % error
```

Attribute **GetAccess** allows us to restrict how we are able to **get** properties and methods

```
% in command window  
s = Schedule(a, b, c);  
disp(s.window) % OK
```

Arrays of objects

- An array of objects can reference objects of a single class

```
B(1) = Die();           % vec of die objects (length 1)
B(2) = Die(6);         % vec of die objects (length 2)
B(3) = TrickDie(1, 2, 6); % Error!
```

- A cell array can reference objects of different classes

```
A{1} = Die();          % cell array of length 1
A{2} = TrickDie(2,10); % cell array of length 2
```

Vocab you should know

Try to fill in on your own before the next slides

- _____ : The template that specifies a custom MATLAB type.
 - Defines _____ and _____ for that class.
- _____ : Specific instance of a class.
- _____ : special method that returns the handle to a newly allocated object
- _____ : unique identifier of an object generated by MATLAB; output of the constructor
- _____ : change the behavior of a built-in function for an object of a class
- _____ : writing functions that take variable number of input arguments
 - _____ : returns the number of input arguments given in the call to the currently executing function
- A subclass _____ from a superclass. A child class _____ from a parent class

Vocab you should know

- **Class** : The template that specifies a custom MATLAB type.
 - Defines **properties** and **methods** for that class.
- **Object** : Specific instance of a class.
- **Constructor** : special method that returns the handle to a newly allocated object
- **Handle** : unique identifier of an object generated by MATLAB; output of the constructor
- **Function overriding** : change the behavior of a built-in function for an object of a class
- **Function overloading** : writing functions that take variable number of input arguments
 - **nargin** : returns the number of input arguments given in the call to the currently executing function
- A subclass **inherits** from a superclass. A child class **inherits** from a parent class