

- 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?

Lecture 24 2

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

<pre> 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 </pre>	<p style="border: 1px solid blue; border-radius: 10px; padding: 5px; display: inline-block;">"Inherit" the components of class Die</p> <pre> classdef TrickDie < handle properties (Access=private) favoredFace weight=1; end methods function D = TrickDie(...) ... function f = getFavoredFace(...) ... function w = getWeight(...) ... end end </pre>
--	---

Lecture 24 5

Yes! Make TrickDie a subclass of Die

<pre> 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 </pre>	<pre> classdef TrickDie < Die properties (Access=private) favoredFace weight=1; end methods function D = TrickDie(...) ... function f = getFavoredFace(...) ... function w = getWeight(...) ... end end </pre>
--	---

Lecture 24 6

Inheritance

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

```

graph BT
    TrickDie --> Die
    Die --> handle
    
```

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

Lecture 25 7

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**

Lecture 25 8

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`

```

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

Lecture 25

Which components get "inherited"?

- public components get inherited
- private components exist in object of child class, but cannot be directly accessed in child class => we say they are not inherited
- Note the difference between inheritance and existence!

Lecture 25

10

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

Lecture 25

11

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

Lecture 25

12

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

Lecture 25

14

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

Lecture 25

15

Accessing superclass' version of a method

- Subclass can override superclass' methods
- Subclass can access superclass' version of the method

```

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

See method `disp` in `TrickDie.m`

Lecture 25

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

Lecture 25

17

(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

Recursion

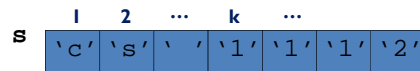
- The Fibonacci sequence is defined **recursively**:
 $F(1)=1, F(2)=1,$
 $F(3)= F(1) + F(2) = 2$
 $F(4)= F(2) + F(3) = 3$ } $F(k) = F(k-2) + F(k-1)$
 It is defined in terms of itself; its **definition invokes itself**.
- Algorithms, and functions, can be recursive as well. I.e., a **function can call itself**.
- Example: remove all occurrences of a character from a string
`'gc aatc gga c ' \rightarrow 'gcaatcggac'`

Lecture 25

20

Example: removing all occurrences of a character

- Can solve using iteration—check one character (one component of the vector) at a time



Subproblem 1:
Keep or discard s(1)

Subproblem 2:
Keep or discard s(2)

Subproblem k:
Keep or discard s(k)

Iteration:
Divide problem into a sequence of equal-sized, identical subproblems

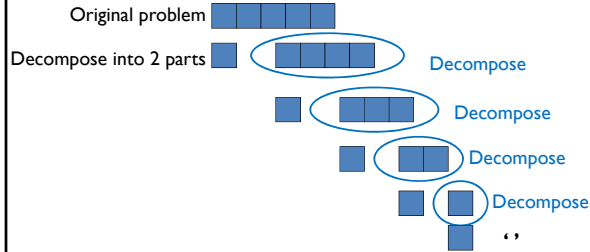
See `RemoveChar_loop.m`

Lecture 25

21

Example: removing all occurrences of a character

- Can solve using **recursion**
 - Original problem: **remove all the blanks** in string s
 - Decompose into two parts: **1. remove blank in s(1)**
2. remove blanks in s(2:length(s))



```
function s = removeChar(c, s)
% Return string s with character c removed

if length(s)==0 % Base case: nothing to do
return
else
if s(1)~=c
% return string is
% s(1) and remaining s with char c removed

else
% return string is just
% the remaining s with char c removed

end
end
```

Lecture 25 26

```
function s = removeChar(c, s)
if length(s)==0
return
else
if s(1)~=c
s = [s(1) removeChar(c, s(2:length(s)))];
else
s = removeChar(c, s(2:length(s)));
end
end
```

s [d _ o _ g]

c []

removeChar - 1st call

```
function s = removeChar(c, s)
if length(s)==0
return
else
if s(1)~=c
s = [s(1) removeChar(c, s(2:length(s)))];
else
s = removeChar(c, s(2:length(s)));
end
end
```

s [d _ o _ g]

c []

removeChar - 1st call removeChar - 2nd call

```
function s = removeChar(c, s)
if length(s)==0
return
else
if s(1)~=c
s = [s(1) removeChar(c, s(2:length(s)))];
else
s = removeChar(c, s(2:length(s)));
end
end
```

s [d _ o _ g]

c []

removeChar - 1st call ~~removeChar - 2nd call~~ ~~removeChar - 3rd call~~ ~~removeChar - 4th call~~

~~removeChar - 6th call~~ ~~removeChar - 5th call~~

Key to recursion

- Must identify (at least) one **base case**, the “trivially simple” case
 - no recursion is done in this case
- The **recursive case(s)** must reflect **progress towards the base case**
 - E.g., give a **shorter vector** as the argument to the recursive call – see **removeChar**