CS 1114:
Programming clarity via Objects – part 2

Prof. Graeme Bailey

http://cs1114.cs.cornell.edu

(notes modified from Matt Dunham, http://www.advancedmcode.org/object-oriented-programming-in-matlab.html)

Cornell University
Computer Science

Pointers to Efficiency

- MATLAB's default behaviour is very wasteful
  - It likes to copy everything!!
  - Called 'passing by value'
  - x = y is done by copying all of y onto x
  - This is ok if y is a number, but what if it's a 10000x10000 matrix?
  - This happens also when passing values into a function! ……..S……L……O……W……..

- Far better to pass the *address* of the data!!!
  - Called 'passing by reference'
  - MATLAB has a handle class we can 'steal from':

*Handl*ing Operator Overloading

- We can redefine + (and *, <, &, ||, etc)!

```
classdef point < handle
  % write a description of the class here.
    properties
      % ....
    end
    methods
      % methods, including the constructor are defined here
        function obj = point(x, y, colour)
          % ....
        end
        function total = plus(a, b)
          temp_x = a.x_coord + b.x_coord;
          temp_y = a.y_coord + b.y_coord;
          temp_c = max(a.colour, b.colour);
          total = point(temp_x, temp_y, temp_c);
        end
        % .... more functions here
    end
end
```

Really elegant – allows passing of addresses instead of full copying of all values; vastly faster and less memory-intensive.

This overides the usual meaning of `+` *and so would be used as follows:*

```
a = point(200, 300, 125);
b = point(159, 203, 224);
c = a + b;
```

giving `c` *the values: 259, 503, and 349 for* x_coord, y_coord, *and* colour

Also makes updating values look cleaner:

```
a = pointy(200, 300, 125);
a.set_colour(98); % changes a's colour to 98
```

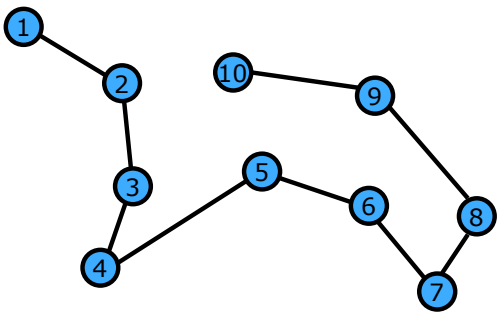

Object behaviours

- Passing by reference:
  - Since `a = pointy(1,2,3)` gives `a` the *address* of the data content, writing `b = a` ends up with both `a` and `b` pointing to the *same* data, so `a.set_x(9)` will mean that `a.get_x()` and `b.get_x()` will both return 9 … it's exactly the same data being changed by `set_x` and being accessed by `get_x` !
  - If you really want to *copy* `a` in order to end up with 2 *distinct* copies, then you'll have to write a `copy` method inside the class to return a new object cloning `a` , then changing one copy will leave the other copy unchanged.
  - Technically, `handle` is a pre-existing class in MATLAB, and writing `pointy < handle` is telling `pointy` to *inherit* everything from `handle` (such as the ability to pass by reference). It can of course have extra abilities all of its own if we define those inside `pointy`.

Linked Lists Revisited

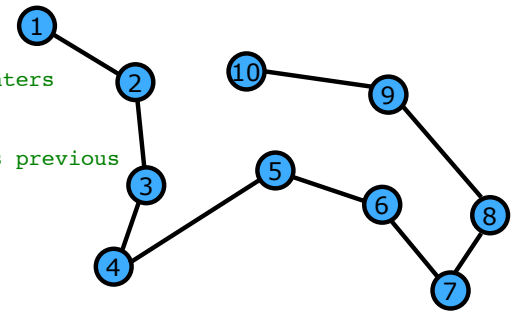
```
classdef node < handle
  % creates nodes for a doubly linked list.
    properties % better to make access private and have gettor and settor methods
        data;
        next;
        previous;
    end

    methods
       function obj = node(d, n, p)
          % class constructor
          if (nargin == 3) % if all input values were given
            obj.data = d;
            obj.next = n;
            obj.previous = p;
          end
          if (nargin == 1) % if only data given
            obj.data = d;
          end
       end % end constructor
    end
end
```



Linked Lists Revisited

```
Classdef linky < handle % creates a doubly linked list.
    properties % better to make access private and have gettor and settor methods
        header; length; % header points to the first real node when there's content
    end
    methods
       function obj = linky() % class constructor
          obj.header = node(); % empty node
          obj.length = 0;
       end % end constructor
       function vide = listempty(obj) % returns true if empty
          vide = (obj.length == 0);
       end % end isempty function
       function preface(obj, data) % insert at front
          temp = node(data) % creates a non-empty node
          temp.next = obj.header.next; % adjust temp's pointers
          temp.previous = obj.header;
          obj.header.next = temp; % adjust header's next
          temp.next.previous = temp; % adjust temp's next's previous
          obj.length = obj.length + 1;
       end % end insertion at front preface method
       function current_list = display(obj)
          current_list = ' HEADER ';
          temp = obj.header;
          for counter = 1 : obj.length
                temp = temp.next;
                current_list = [ current_list , num2str(temp.data) , ' <--> ' ] ;
          end % ending loop through all the list nodes
       end % overwriting the default display method
    end
end
```



Class Inheritance

- So we *inherited* `linky` from the `handle` class … what does this mean?
  - Every property `handle` has, `linky` gets
  - Every method `handle` has, `linky` gets
  - Any values that `handle` might have, `linky` doesn't get! Sorry, you don't get your parent's bank balance!

- We can use inheritance with our own classes, but why and how?
  - The notation is `A < B` for `A` inheriting from `B`
  - If you find yourself writing the same properties and methods for two different classes, consider pooling those that are common to both into a fresh class `C` and then have `A < C` and `B < C`
  - This saves effort and makes maintaining and debugging the code easier