CS 1112 Final Exam Review

Objects and Classes

- **Class:** A file that specifies *properties* and *methods* (functions) associated with the item that the class represents
  - Contains a *constructor*, a special method that creates new objects
  - A class can have *subclasses*

- **Object:** One *instance* of a class
  - Objects of the same class have the same properties and the same methods
  - The properties of objects of the same class can have *different values*

Objects and Classes example: **Animal**

```
classdef  Animal  < handle
    properties
        name;  species;  age;  hasTail
    end

    methods

        function  a = Animal(n,  s,  a,  hT)
            % set properties  of a
        end

        function  birthday(self)
            self.age  = self.age+1;
        end

        function  c = checkHasTail(self)
            % return  1 if hasTail  = 1,  else  0
        end

        function  c = isOlder(self,  otherAnimal)
            % return  1 if older  than  otherAnimal
        end
    end
end
```

Note that the `end` keyword is used to close the following:

1. The `classdef`
2. The `properties` section
3. The `methods` section
4. Each `function` inside the `methods` section

Objects and Classes: **Constructors**

Constructor: A method (function) that creates a new object

- Must have the same name as the class
- Can take in parameters to set property values
- Use `nargin` to ensure that constructor can be called without any arguments

Objects and Classes example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end

    methods
        function a = Animal(n, s, a, hT)
            % set properties of a
        end

        function birthday(self)
            self.age = self.age+1;
        end

        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end

        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

```
function a = Animal(n, s, a, hT)
    if (nargin == 4)
        a.name = n;
        a.species = s;
        a.age = a;
        a.hasTail = hT;
    end
end
```

If 4 arguments are not provided, the 4 properties will be set to default values.

Objects and Classes: **Create/reference objects**

Create new objects by calling the constructor, which returns a *reference to the new object* that should be stored in a variable.

Example:

```
a = Animal('Bobbert','pig', 2, 1);
% Creates an animal object with the properties:
% name = 'Bobbert', species = 'pig', age = 2, hasTail = 1
```

Create an empty array of Animal objects using .empty()

Example:

```
b = Animal.empty();
```

Check if an object/object array is empty using `isempty(<reference>)`

Example: `isempty(a)` returns 0, `isempty(b)` returns 1

Objects and Classes: **Calling methods**

Each method in a class takes in a minimum of one parameter (named '`self`'), which *is a reference to the object calling the method*

Syntax for calling a method:

<reference>.<methodName>(2nd through last argument)

Equivalently (but better to use above way),

<methodName>(self, rest of arguments)

Objects and Classes example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end

    methods
        function a = Animal(n, s, a, hT)
            % set properties of a
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

How to use this method:

```
% Object reference should be
% created first
a = Animal('Bobbert', 'pig', 2, 1);

% Call method
a.birthday();

% See result of method call
disp(a.age)   % 3 will be displayed
```

Objects and Classes example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end

    methods
        function a = Animal(n, s, a, hT)
            % set properties of a
        end
        function birthday(self)
            self.age = self.age+1;
        end
        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end
        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

Implementation of this method:

```
function c = checkHasTail(self)
    if (self.hasTail == 1)
        c = 1;
    else
        c = 0;
    end
end
```

Objects and Classes example: **Animal**

```
classdef Animal < handle
    properties
        name; species; age; hasTail
    end

    methods

        function a = Animal(n, s, a, hT)
            % set properties of a
        end

        function birthday(self)
            self.age = self.age+1;
        end

        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end

        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end

    end
end
```

Implementation of this method:

```
function c = isOlder(self,otherAnimal)
    if (self.age > otherAnimal.age)
        c = 1;
    else
        c = 0;
    end
end
```

How to use this method:

```
a = Animal('Bobbert', 'pig', 2, 1);
b = Animal('Bob', 'frog', 1, 0);
disp(a.isOlder(b))   % will display 1
disp(b.isOlder(a))   % will display 0
```

Objects and Classes: **Arrays of objects**

Objects of the same class can be stored in a simple vector/array.

Objects of different classes could be stored in a cell array.

Example: Write a function that takes in a vector `z` of Animal objects and returns a vector of the indices from `z` which contain objects whose species is 'pig':

```
function idx = findPigs(z)
idx = [];  k = 1;
for i=1:length(z)
   if(strcmp(z(i).species, 'pig'))
       idx(k) = i;
       k = k+1;
   end
end
```

Objects and Classes: **Accessibility**

Keywords *public, private, protected* can be used to restrict access to properties.

- **Public** properties: can be directly in any subclasses any other files that create objects of this class
- **Private** properties: cannot be directly accessed outside the class
- **Protected** properties: can only be directly accessed by subclasses

Direct access means being able to access a property via statements such as:

```
<reference>.propertyName
```

Indirect access could be in the form of:

```
<reference>.getPropertyValue()
% Need ‘‘getter’’ methods to access private or protected properties
```

Objects and Classes: **Inheritance**

- A class can have subclasses that share properties and methods.
  - **Private** properties are not inherited, but can be accessed through methods
  - **Protected** properties are inherited; all subclasses can access them
  - **Public** properties are inherited; all classes can access them

- **In the constructor of a subclass**, there must be a call to the superclass constructor (using "@" notation)

Objects and Classes example: **Animal and Bird**

```
classdef Animal < handle
    properties (Access = protected)
        name; species; age; hasTail
    end

    methods

        function a = Animal(n, s, a, hT)
            % set properties of a
        end

        function birthday(self)
            self.age = self.age+1;
        end

        function c = checkHasTail(self)
            % return 1 if hasTail = 1, else 0
        end

        function c = isOlder(self, otherAnimal)
            % return 1 if older than otherAnimal
        end
    end
end
```

```
classdef Bird < Animal
    properties (Access = private)
        color
    end
    methods
        function b = Bird(c, n, s, a)
            b = b@Animal(n, s, a, 1);
            b.color = c;
        end
        function c = getColor(self)
            c = self.color;
        end
    end
end
```

Bird is a subclass of Animal and inherits all of its protected properties.
Bird has one additional property, color.

Structs

Structs provide a way to group related data together into one variable. Each piece of data is labeled by a field name.

A struct can be created using the struct function. Example:

```
point1 = struct('x', 1, 'y', 2);
```

Spring 2015 Prelim: **Question 4**

```
r=0; % number of rooms processed so far
for i= 1:length(D)
    pos= strfind(D{i}, ' ');
    for k= 1:2:length(pos)
        id= D{i}(pos(k)+1:pos(k+1)-1);
        seats= str2double(D{i}(pos(k+1):pos(k+2)));
        r= r+1;
        Q(r)= MakeRoom(id, seats);
    end
end
```

Review question 3b: **Sorting structs**

```
n= length(Pts);
dis= zeros(1,n);  % stores the distance of each point from the origin
dis(1)= sqrt(Pts(1).x^2 + Pts(1).y^2);
for i= 1:n-1
    % Sort dis(1:i+1) given that dis(1:i) is sorted
    dis(i+1)= sqrt(Pts(i+1).x^2 + Pts(i+1).y^2);
    j= i;
    need2swap= dis(j+1) < dis(j);
    while  need2swap
        % swap elements in dis array
        tempd= dis(j);
        dis(j)= dis(j+1);
        dis(j+1)= tempd;
        % swap elements in Pts array
        tempp= Pts(j);
        Pts(j)= Pts(j+1);
        Pts(j+1)= tempp;
        j= j-1;
        need2swap= j>0 && dis(j+1)<dis(j);
    end
end
```

Open coding questions 1: **Pascal's triangle**

```
function p = pascalVector(lev)

 p= 1;   % values in level 0 of Pascal's triangle
 for currentLevel= 1:lev
     % Temporary vec (q) for current level
     q= ones(1,currentLevel+1);

     % Use the general formula for INTERNAL entries
     for i= 2:length(p)
         q(i)= p(i-1) + p(i);
     end

     p= q;  % Update p
 end
```