CS1112 Lab Exercise 12

1 Complete class `Interval`

1.1 Download the file `Interval.m` from the *Exercises* page. Let's play with some `Interval` objects in the Command Window:

```
a= Interval(3,7)       % See in Workspace pane that the class of a is Interval.
                       %   Read Interval.m to see how properties were declared.
disp(a.left)           % Access the left property using dot notation; should be 3
disp(a.right - a.left) % Should be 4, the interval's width
a.shift(10)            % Call a's shift method to shift interval a to the right
                       %   by 10 units.  Method shift doesn't return a value (see
                       %   method definition in Interval.m), so you do not see
                       %   anything displayed in Command Window
disp(a)                % Display interval a now: ________________________
b= Interval(9,15);
g= a.isIn(b)           % Is interval a in interval b? ____________
                       %   Read method isIn.  Ask if you have any questions.
h= b.isIn(a)           % Is interval b in interval a? ____________
```

Observations: To access an *instance variable* (property), the syntax is ReferenceName**.**VariableName . To access an *instance method* (method defined inside a classdef for each object), the syntax is ReferenceName**.**MethodName**(**args for 2nd thru last parameters**)** .

1.2 Complete/revise the three methods `getWidth`, `scale`, and `add`. Then in the Command Window write some code to try out the new class definition, e.g.,

```
clear all          % Must clear objects made using the old class definition; otherwise
                   %  Matlab gives an error message when you use the new class definition
a= Interval(3,7);
w= a.getWidth()    % w should be 4
a.scale(2)
disp(a)            % a should be (3,11)
b= Interval(0,2);
c= a.add(b)        % c should be (3,13)
```

Do you understand everything so far? If not, ask for help!

1.3 Above, we used MATLAB's built-in `disp` function to display the properties of an object. We can *override* the built-in method to display what *we* want to see for an object of class `Interval`! To do so, we simply implement a `disp` method inside the classdef of `Interval`. This was done but commented out. *Un*comment the `disp` method in class `Interval` now, save the file, and type the following code in the Command Window:

```
clear all
x= Interval(3,7)  % What is displayed? ____________________________
                  % In the above statement, since you didn't use a semicolon,
                  % Matlab called the disp method to display x.  Since x is of
                  % type Interval and class Interval has its own disp method, that
                  % specific disp method was used instead of the built-in disp.
```

2 Class `Fraction`

Download the file `Fraction.m` from the *Exercises* page; it is an incomplete class definition. Read it, experiment with it, and implement the incomplete methods. Here're the specific things to note and do:

2.1 Read the class comment carefully. In our simple `Fraction` class we simply assume that the numerator and denominator are integers—we do not check for this. A `Fraction` does not need to be in the reduced form, i.e., 16/6 is fine and does not need to be reduced to 8/3. A negative fraction should have the negative sign associated with the numerator, not denominator. This and other requirements of our `Fraction` are taken care of already in the constructor. Read it carefully.

2.2 Read the given method `isLessThan` in the classdef. Do you understand it? Now experiment!

```
a= Fraction(3,4)
b= Fraction(3,6)
a.isLessThan(b)    % True or false? _______
b.isLessThan(a)    % True or false? _______
```

2.3 Complete method `isEqualTo`. Save the file, clear the Workspace, create some `Fraction`s and call the `isEqualTo` method! For example,

```
a= Fraction(3,4)
b= Fraction(3,6)
c= Fraction(1,2)
a.isEqualTo(b)    % True or false? _______
b.isEqualTo(c)    % True or false? _______
```

2.4 Complete method `add` and then try these statements:

```
a= Fraction(3,4)
b= Fraction(3,6)
c= a.add(b)       % What is fraction c?  Is it correct?
```

2.5 Complete method `toDouble` and then try these statements:

```
a= Fraction(3,4)
x= a.toDouble()       % Call a's toDouble method.  Should be 0.75
```

2.6 Complete method `reduce`. You can use any algorithm you like for calculating the GCD, Greatest Common Divisor, but here's Euclid's algorithm for finding the GCD between two *positive* values a and b where $a \leq b$:

1. Calculate the remainder r from b divided by a.
2. If r is zero than a is the GCD.
3. Otherwise, let b get a and a get r. Repeat from Step 1.

Note that if the numerator is zero or `Inf` then the fraction cannot be reduced (is already in the reduced form). To check whether a variable `x` has the value `Inf`, use the function `isinf`: `isinf(x)` returns true (1) if `x` is `Inf` and false (0) otherwise.

After completing method `reduce`, try these statements (and more) in the Command Window:

```
a= Fraction(8,6)
a.reduce()        % Call a's reduce method. Since this method doesn't return
                  %   anything, nothing is displayed to the command window.
disp(a)           % Is it correct?
a= Fraction(8,2)
a.reduce();  disp(a)  % Is it correct?
a= Fraction(1,3)
a.reduce();  disp(a)  % Numerator and denominator should remain the same
a= Fraction(0,9)
a.reduce();  disp(a)  % Numerator and denominator should remain the same
a= Fraction(9,0)
a.reduce();  disp(a)  % Numerator and denominator should remain the same (Inf and
                      %   1 as originally specified by the constructor)
```

Does your method `reduce` work? It would be nice to call `reduce` whenever we create a `Fraction`! Read the constructor again, and now *un*comment the last statement so that method `reduce` is called whenever a `Fraction` is created.

```
a= Fraction(8,6)  % Fraction has the numerator 4 and denominator 3
```

You can uncomment the `disp` method in order to display a `Fraction` in the format *numerator*/*denominator* if you like. This is not required.

Please delete your files before leaving the lab!