CS1112 Exercise 11 Name: _____ NetID: _____

In addition to discussing these questions with your group and TAs during section, please submit scans of your completed worksheet to Gradescope by *Sunday, 4/26, 9pm EDT*. See *Written assessment procedures* on Canvas (Lecture 19) for submission expectations. Thursday's lecture will build on this exercise.

1 Designing Classes

This problem is about generating *ideas*, not writing code. For each design problem below, focus on what *data* (properties) are required and what *actions* may be performed on that data. (*Later* we will turn ideas into code—no code for now.)

1.1 Design a *Fraction* class

(a) What makes up a *Fraction*? (There are at least two properties.)

(b) What actions can you perform on a *Fraction* or multiple *Fraction*s? (Give at least four actions.)

1.2 Design a *Facebook Profile* class

(a) What does a *Facebook Profile* contain? (Give at least four example properties and specify their type.)

(b) What actions can you perform on or perform using one or more *Facebook Profile*s? (Give at least six example actions.)

2 Creating and playing with `Interval` objects

Download the file `Interval.m` from the *Exercises* page and put it in your `Current Directory`. Let's play with some `Interval` objects in the Command Window:

```
a= Interval(3,7)  % Create an Interval object and call it a. See in Workspace pane that the
                  % class of a is Interval.  Output shows the two properties: left, right
disp(a.left)         % Access a's left value; should be 3
w= a.right - a.left  % Should be 4, the interval's width
```

Now look at the file—the *class definition*—`Interval.m`. Do not worry about the syntax shown in the file, which will be explained further in lecture, but simply note that there are the two *properties* as well as four *methods*—functions—defined. Above, you saw that to access an *instance variable* (property), the syntax is

ReferenceName . VariableName

where ReferenceName is the variable name used when you created the object. Next we call the object's methods:

```
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, so you don't see anything displayed in the Command Window.
```

```
disp(a)      % Display interval a now: _________________________
a.scaleRight(2)  % Scale (expand the width) of interval a, maintaining the left end

disp(a)      % Display interval a now: _________________________
```

As shown above, the syntax to access an *instance method* (method defined inside a classdef for each object) is

ReferenceName . MethodName(arguments)

Next, we find out whether interval `a` is in interval `b`. Call `a`'s `isIn` method and pass to it `b`:

```
b= Interval(9,25);      % Create another interval, b
g= a.isIn(b)            % Is interval a in interval b? _______________

h= b.isIn(a)            % Is interval b in interval a? _______________
```

3 Creating and playing with `Fraction` objects

Download the file `Fraction.m` from the *Exercises* page. Our simple `Fraction` class assumes 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 has the negative sign associated with the numerator, not denominator. This and other requirements of our `Fraction` are taken care of already in the method called `Fraction` in the file. Again, don't worry about the syntax shown in the file; we will just play with method calls in the *Command Window*:

```
x= Fraction(3,4)   % Create a fraction and call it x
y= Fraction(3,6)
x.isLessThan(y)    % True or false? _______________
y.isLessThan(x)    % True or false? _______________
```

This exercise shows that we can have multiple objects of the same class as well as objects of different classes in use at the same time. Unless you had cleared the memory space earlier, you should see in the *Workspace* pane different classes of objects—`Interval` and `Fraction`—as well as values of the simple types `double` and `logical`.

Finally, multiply fractions `x` and `y` to create a new fraction `c`:

```
c= x.multiply(y)
d= y.multiply(x)  % You get the same result because multiplication is commutative
```

4 An object is *referenced*

An object is *referenced by* a variable, not *stored in* a variable. What's the difference?

```
u= Interval(3,7)  % u stores the HANDLE, or address, of the object. So u REFERENCES the object.
z= u       % z gets what's in u, which is the HANDLE; so both u and z reference the same object.
z.left= 5  % Change the left property in the object referenced by z to 5.
disp(u)    % What is the left property of the object referenced by u? _______________
           % Both u and z reference the SAME object; z is an alias of u.

p= 10           % Create a scalar variable p storing the VALUE 10.
r= p            % r gets what's in p, which is the value; so r is a separate copy of 10.
r= 12           % Variable r now has a new value.
disp(p)         % What is the value of p? _______________
                % p and r store two DIFFERENT values; changing r does not change p.
```

An object is referenced: in the example above, `u` and `z` reference (refer to) the *same* object. All other variables we've seen before (cell arrays, or `doubles`, `uint8`s, `chars`, etc.), are stored directly in the variable. Look again at the Lecture 19 worksheet; why doesn't 'B' work for question 1?

5 A number guessing game

Download the file `guessingGame.m` from the *Exercises* page and put it in your *Current Directory* along with `Interval.m` which you downloaded earlier. This script is complete and you can run it to play a few games! Next, read the script and you'll see familiar-looking code. Other than the use of an `Interval` object in lines 18, 20, 23, 25, and 34, the script is just *procedural programming* as we have been writing since the beginning of the semester. *Don't be scared about this new thing called an object! You will be building on what you have learned previously.*