CS1112 Discussion Exercise 13

1. `Die` vs. `TrickDie`

Read the script below for simulating a dice game. Ask if there's anything that you do not understand! Then modify the game:

1. In the part marked `BLOCK 1`, make the third die a `TrickDie`. You can choose which face is favored and by how much. Question to consider: can an array contain both `Die`s and `TrickDie`s? Modify the rest of the code as necessary to make it work.
2. Consider the code in `BLOCK 2`. Do you need to change any method calls now that you have both `Die`s and a `TrickDie`?
3. Come up with a different rule for "winning" (earning a point) and implement it in `BLOCK 2`.

The class definitions of `Die` and `TrickDie` are given at the end of this document.

```
% A game using Dies (dice)

nD= 3;       % number of dice
point= 0;    % number of points earned so far
nTrials= 20; % number of trials

for k= 1:nD   %%%% BLOCK 1 %%%%
   d(k)= Die(6);
end

for t= 1:nTrials

    % Roll dice
    for k= 1:nD
        %Roll kth die
        d(k).roll()
        fprintf('%d ', d(k).getTop())
    end

    % Win 1 point if 3 consecutive numbers in ascending order
    % Winning example: 2,3,4
    % Losing examples: 4,3,2  2,4,3  5,6,1  1,5,6  etc.
    %%%% BLOCK 2 %%%%
    noWin= 0;
    for k= 1:nD-1
       difference(k)= d(k+1).getTop() - ...
                      d(k).getTop();
       if difference(k)~= 1
           noWin= 1;
       end
    end
    if ~noWin
        point= point + 1;
    end
    fprintf('  Points so far: %d\n', point)

end
```

2. Efficient calculation of x^n where n is large

If you cannot use MATLAB's power operator ^ how would you calculate x to the n-th power? One way is to use iteration—a loop that executes $n-1$ times. Another strategy is recursion—repeated squaring in this case. The idea is illustrated with the following schematic that shows how to compute x^{21}:

$$
\begin{aligned}
x^{21} &= (x^{10})^2 \cdot x \\
&\quad\; \llcorner\!\!\rightarrow\; x^{10} = (x^5)^2 \\
&\qquad\quad \llcorner\!\!\rightarrow\; x^5 = (x^2)^2 \cdot x \\
&\qquad\qquad\quad \llcorner\!\!\rightarrow\; x^2 = (x)^2
\end{aligned}
$$

The recursive definition behind the scenes is given by

$$
f(x,n) = \begin{cases} 1 & \text{if } n = 0 \\ f(x, n/2) \cdot f(x, n/2) & \text{if } n > 0 \text{ and } n \text{ is even} \\ f(x, (n-1)/2) \cdot f(x, (n-1)/2) \cdot x & \text{if } n > 0 \text{ and } n \text{ is odd} \end{cases} .
$$

Write the following function based on the *recursive* strategy. Do not use loops.

```
function y = Power(x, n)
% y = x^n where n is an integer >=0
```

```
classdef Die < handle
% A fair die has a certain number of sides
% (default is 6). Any side may be top face.

    properties (Access=private)
       sides=6;
       top
    end

    methods
        function D = Die(s)
        % Constructor:  Create an s-sided Die
            if nargin==1
                D.sides= s;
            end
            D.roll()
        end

        function s = getSides(self)
           s= self.sides;
        end

        function t = getTop(self)
           t= self.top;
        end

        function roll(self)
        % Roll the Die once
            face= ceil(rand*self.getSides());
            self.setTop(face)
        end

        function disp(self)
          if length(self)==1
            fprintf('die showing face %d\n',...
                    self.getTop())
          else
            disp@handle(self)
          end
        end

   end %methods public

   methods (Access=protected)
       function setTop(self, f)
       % Set this Die's top to face f
        self.top= f;
       end
   end %methods protected

end %classdef
```

```
classdef TrickDie < Die
% An unfair die with one face (favoredFace) a
% certain number of times (weight) more likely
% than another face to be on top.

    properties (Access=private)
        favoredFace  % face more likely to be top
        weight= 1;   % favoredFace is WEIGHT times
                     %   more likely to be on top
    end

    methods
        function TD = TrickDie(ff, w, s)
        % Constructor: Favored face ff is w times
        % more likely than another face of this
        % TrickDie to be on top.
            if nargin<3
                s= 6;
            end
            TD= TD@Die(s);
            if nargin>=2
                TD.favoredFace= ff;
                TD.weight= w;
            end
            TD.roll()
        end

        function w = getWeight(self)
            w= self.weight;
        end

        function ff = getFavoredFace(self)
            ff= self.favoredFace;
        end

        function roll(self)
            face= ceil(rand*(self.getSides()+...
                       self.getWeight()-1));
            if face>self.getSides()
                face= self.getFavoredFace();
            end
            self.setTop(face)
        end

        function disp(self)
            fprintf('tricky ')
            disp@Die(self)
        end
    end %methods

end %classdef
```