Assignment 3 Report

Overall I was able to finish the program, excluding the optional task 9. Although I was able to get my interface to work relatively like task 9 asks, I think ultimately it isn't up to par for what our Professor expected. Overall I had a relatively easy time completing this program, I've spent a lot of my time programming doing object oriented programming so once I got used to the syntax of Mathlabs, I found the "language" to be very familiar. The biggest obstacle I had when completing this program was when I had to first figure out mutators / helper functions such as calculate area. I was having a hard time not understanding why the field wasn't being properly updated even though I'm using the syntax obj.Area = CalculateArea()... and I later realized it was because the obj's reference isn't being updated to the new data. Furthermore the biggest struggle outside of coding I found was trying to decipher what the assignment was asking for. Especially towards the end with task 8, I found it initially very easy since I was able to just spit out messages in the terminal asking for the user to give some kind of input. But later decided it was meant to be harder and challenged myself to create "pop-ups" that would take in user input, along with giving them prompts such as "Enter Triangle Base:".

Ultimately I worked alone, I did converse with individuals occasionally to get an idea of how they approached a problem. Most of the outside resources I used were the slides from Canvas, specifically on mutators because I had trouble / still have a problem automatically updating the obj. This can be seen in how when you try to change the color of an obj, instead of obj.SetColor(newColor) working, you have to do obj = obj.SetColor(newColor). Furthermore I accessed mathlab's forums post to get some references / examples on how to do things such as updating fields. One of the examples was:
https://www.mathworks.com/matlabcentral/answers/183246-updating-property-of-an-object-without-creating-new-object

Overall I found math labs to be relatively fun programming assignments. Although personally I would rather stick to more mainstream languages such as Java.

Task 1: Class Creation and Constructors

Code:

Editor - C:\Users\ethan\Documents\MATLAB\Assignment3\Shape.m

Circle.m | Rectangle.m | Shape.m | Triangle.m | MyShapes.m

```
classdef Shape < matlab.mixin.Heterogeneous
    properties
        Name
        Area
    end

    methods
        function obj = Shape(name) %Constructor
            if nargin > 0
                obj.Name = name;
            else
                obj.Name = '';  %default action if name is void
            end
        end
```

Shape1 and Shape2:

Command Window

```
>> shape1 = Shape('Square');
>> shape2 = Shape('Circle');
>> shape1

shape1 =

  Shape with properties:

    Name: 'Square'
    Area: []

>> shape2

shape2 =

  Shape with properties:

    Name: 'Circle'
    Area: []
```

Task 2: Inheritance and Constructor Overloading

Circle Code:

Editor - C:\Users\ethan\Documents\MATLAB\Assignment3\Circle.m

Circle.m | Rectangle.m | Shape.m | Triangle.m | MyShapes.m

```
classdef Circle < Shape & ColorMixin
    properties
        Radius
    end

    methods
        function obj = Circle(radius, color)    %Constructor
            obj@Shape('Circle');    %Calls shape constructor
            obj@ColorMixin(color);  %Calls colormixin constructor
            obj.Radius = radius;
            obj = obj.CalculateArea();  %Calls helper
        end

        function circle = CalculateArea(obj)    %Helper
            obj.Area = pi * obj.Radius^2;
            circle = obj;   %updates obj reference with new obj.
        end
```

Circle Objects:

```
>> circle1 = Circle(5, 'blue');
>> circle2 = Circle(10, 'green');
>> circle1

circle1 =

  Circle with properties:

    Radius: 5
      Name: 'Circle'
      Area: 78.5398
     Color: 'blue'

>> circle2

circle2 =

  Circle with properties:

    Radius: 10
      Name: 'Circle'
      Area: 314.1593
     Color: 'green'
```

Rectangle Code:

```
classdef Rectangle < Shape & ColorMixin
    properties
        Length
        Width
    end

    methods
        function obj = Rectangle(length, width, color) %Constructor
            obj@Shape('Rectangle'); %Calls shape constructor
            obj@ColorMixin(color);  %Calls color constructor
            obj.Length = length;
            obj.Width = width;
            obj = obj.CalculateArea(); %Calls calculateArea helper
        end

        function rectangle = CalculateArea(obj) %Helper
            obj.Area = obj.Length * obj.Width;
            rectangle = obj;    %Updates obj reference
        end
```

Rectangle Obj:

```
>> rectangle1 = Rectangle(4, 8, 'green');
>> rectangle2 = Rectangle(9, 10, 'purple');
>> rectangle1

rectangle1 =

  Rectangle with properties:

    Length: 4
     Width: 8
      Name: 'Rectangle'
      Area: 32
     Color: 'green'

>> rectangle2

rectangle2 =

  Rectangle with properties:

    Length: 9
     Width: 10
      Name: 'Rectangle'
      Area: 90
     Color: 'purple'
```

Triangle Code:

```
classdef Triangle < Shape & ColorMixin
    properties
        Base
        Height
    end

    methods
        function obj = Triangle(base, height, color)
            obj@Shape('Triangle'); %Calls "Shape" constructor passing "Triangle" as name
            obj@ColorMixin(color); %Calls ColorMixin constructor
            obj.Base = base;       %Assigns field vals to args
            obj.Height = height;
            obj = obj.CalculateArea(); %Calls helper function
        end

        function triangle = CalculateArea(obj)  %Helper function to calculateArea
            obj.Area = 0.5 * obj.Base * obj.Height;
            triangle = obj; %Updates the object w/ area
```

Triangle Objects:

```
>> triangle1 = Triangle(3, 10, 'yellow');
>> triangle2 = Triangle(10,5,'purple');
>> triangle1

triangle1 =

  Triangle with properties:

      Base: 3
    Height: 10
      Name: 'Triangle'
      Area: 15
     Color: 'yellow'

>> triangle2

triangle2 =

  Triangle with properties:

      Base: 10
    Height: 5
      Name: 'Triangle'
      Area: 25
     Color: 'purple'
```

Task 3: Method Overriding

Circle Code:

```
function Display(obj)   %Overloaded display function
    fprintf('The area of a %s circle with a radius of %.2f units is approximately %.2f square units.\n', obj.Color, obj.Radius, obj.Area);
end
```

Triangle Code:

```
function Display(obj) %Spits out data for Triangle
    fprintf('The area of a %s triangle with a base of %.2f units and a height of %.2f units is %.2f square units.\n', obj.Color, obj.Base, obj.Height, obj.Area);
end
```

Rectangle Code:

```
function Display(obj)   %Overloaded Display functinon
    fprintf('The area of a %s Rectangle with a length of %.2f units and a width of %.2f units is %.2f square units.\n', obj.Color, obj.Length, obj.Width, obj.Area);
end
```

Circle ex. output:

```
>> circle1.Display()
The area of a blue circle with a radius of 5.00 units is approximately 78.54 square units.
>> circle2.Display()
The area of a green circle with a radius of 10.00 units is approximately 314.16 square units.
>>
```

Triangle ex. output:

```
>> triangle1.Display()
The area of a yellow triangle with a base of 3.00 units and a height of 10.00 units is 15.00 square units.
>> triangle2.Display()
The area of a purple triangle with a base of 10.00 units and a height of 5.00 units is 25.00 square units.
>>
```

Rectangle ex. output:

```
>> rectangle1.Display()
The area of a green Rectangle with a length of 4.00 units and a width of 8.00 units is 32.00 square units.
>> rectangle2.Display()
The area of a purple Rectangle with a length of 9.00 units and a width of 10.00 units is 90.00 square units.
>>
```

Task 4: Multi-level Inheritance

EquilateralTriangle Code:

```
classdef EquilateralTriangle < Triangle
    properties
        SideLength
    end

    methods
        function obj = EquilateralTriangle(sideLength, color) %Constructor
            obj@Triangle(sideLength, 0, color); %Creates skeleton of triangle
            obj.SideLength = sideLength;    %Assigns sidelength field
            obj.Name = 'Equilateral Triangle';  %Hard codes name
            obj = obj.CalculateHeight();    %Calls helper function to get height
            obj = obj.CalculateArea();      %Calls parent function CalculateArea
        end

        function eqTriangle = CalculateHeight(obj)  %Helper fucntion to calc. height
            obj.Height = obj.SideLength * sqrt(3) / 2;
            eqTriangle = obj;   %Updates eq. Trig obj.
        end
        function Display(obj)   %Overloaded display
            fprintf('The area of a %s equilateral triangle with a side length of %.2f units is approximately %.2f square units.\n', ...
                obj.Color, obj.SideLength, obj.Area);
        end
    end
end
```

EquilateralTriangle Example SS:

```
>> eqTriangle1 = EquilateralTriangle(5, 'blue');
>> eqTriangle2 = EquilateralTriangle(15, 'green');
>> eqTriangle1

eqTriangle1 =

  EquilateralTriangle with properties:

    SideLength: 5
          Base: 5
        Height: 4.3301
          Name: 'Equilateral Triangle'
          Area: 10.8253
         Color: 'blue'

>> eqTriangle2

eqTriangle2 =

  EquilateralTriangle with properties:

    SideLength: 15
          Base: 15
        Height: 12.9904
          Name: 'Equilateral Triangle'
          Area: 97.4279
         Color: 'green'

>>
```

EquilateralTriangle Display:

```
>> eqTriangle1.Display()
The area of a blue equilateral triangle with a side length of 5.00 units is approximately 10.83 square units.
>> eqTriangle2.Display()
The area of a green equilateral triangle with a side length of 15.00 units is approximately 97.43 square units.
>>
```

Task 5: Multiple Inheritance

ColorMixin Code:

```
classdef ColorMixin
    properties
        Color
    end

    methods
        function obj = ColorMixin(color)%Constructor
            if nargin > 0
                obj.Color = color;
            else
                obj.Color = 'White'; %Sets default to white
            end
        end
        function color = GetColor(obj)  %Simple accessor to get color
            color = obj.Color;
        end
        function obj = SetColor(obj, newColor)  %Changes field to new color
            obj.Color = newColor;
        end
    end
end
```

Circle Constructor Reworked:

```
methods
    function obj = Circle(radius, color)    %Constructor
        obj@Shape('Circle');    %Calls shape constructor
        obj@ColorMixin(color);  %Calls colormixin constructor
        obj.Radius = radius;
        obj = obj.CalculateArea();  %Calls helper
    end
```

Triangle Constructor Reworked:

```
methods
    function obj = Triangle(base, height, color)
        obj@Shape('Triangle'); %Calls "Shape" constructor passing "Triangle" as name
        obj@ColorMixin(color); %Calls ColorMixin constructor
        obj.Base = base;       %Assigns field vals to args
        obj.Height = height;
        obj = obj.CalculateArea(); %Calls helper function
    end
```

Rectangle Constructor Reworked:

```
methods
    function obj = Rectangle(length, width, color) %Constructor
        obj@Shape('Rectangle'); %Calls shape constructor
        obj@ColorMixin(color);  %Calls color constructor
        obj.Length = length;
        obj.Width = width;
        obj = obj.CalculateArea(); %Calls calculateArea helper
    end
```

Circle Display Reworked:

```
function Display(obj)   %Overloaded display function
    fprintf('The area of a %s circle with a radius of %.2f units is approximately %.2f square units.\n', ...
        obj.Color, obj.Radius, obj.Area);
end
```

Rectangle Display Reworked:

```
function Display(obj)   %Overloaded Display functinon
    fprintf('The area of a %s Rectangle with a length of %.2f units and a width of %.2f units is %.2f square units.\n', ...
        obj.Color, obj.Length, obj.Width, obj.Area);
end
```

Triangle Display Reworked:

```
function Display(obj) %Spits out data for Triangle
    fprintf('The area of a %s triangle with a base of %.2f units and a height of %.2f units is %.2f square units.\n', ...
        obj.Color, obj.Base, obj.Height, obj.Area);
end
```

Equilateral Triangle Display Reworked:

```
function Display(obj)   %Overloaded display
    fprintf('The area of a %s equilateral triangle with a side length of %.2f units is approximately %.2f square units.\n', ...
        obj.Color, obj.SideLength, obj.Area);
end
```

Display (Before Color Change) Output

```
>> circle1.Display()
The area of a blue circle with a radius of 5.00 units is approximately 78.54 square units.
>> eqTriangle1.Display()
The area of a blue equilateral triangle with a side length of 5.00 units is approximately 10.83 square units.
>> rectangle1.Display()
The area of a green Rectangle with a length of 4.00 units and a width of 8.00 units is 32.00 square units.
>> triangle1.Display()
The area of a yellow triangle with a base of 3.00 units and a height of 10.00 units is 15.00 square units.
```

Changing Color:

```
>> circle1 = circle1.SetColor('red');
>> eqTriangle1 = eqTriangle1.SetColor('yellow');
>> rectangle1 = rectangle1.SetColor('purple');
>> triangle1 = triangle1.SetColor('blue');
>>
```

Display (After Color Change) Output

```
>> circle1.Display()
The area of a red circle with a radius of 5.00 units is approximately 78.54 square units.
>> eqTriangle1.Display()
The area of a yellow equilateral triangle with a side length of 5.00 units is approximately 10.83 square units.
>> rectangle1.Display()
The area of a purple Rectangle with a length of 4.00 units and a width of 8.00 units is 32.00 square units.
>> triangle1.Display()
The area of a blue triangle with a base of 3.00 units and a height of 10.00 units is 15.00 square units.
```

Task 6: Static Method

Static Method Code:

```
methods (Static)    %Static functions
    function CalculateStatistics(input)
        n = numel(input);   %Finds how many elements
        list = zeros(1,n);  %Creates vector

        for i = 1:n %For each element...
            list(i) = input(i).Area; %Assign each element to the area of each shape in list
        end
        meanArea = mean(list);  %Mean, median, std
        medianArea = median(list);
        stdDevArea = std(list);

        fprintf('Statistics for the Areas of Shapes:\n');   %Showcase data
        fprintf('Mean Area: %f\n', meanArea);
        fprintf('Median Area: %f\n', medianArea);
        fprintf('Standard Deviation of Areas: %f\n', stdDevArea);
    end
end
```

Creating Array of Shapes / Objects

```
>> shapes = [circle1, circle2, eqTriangle1, eqTriangle2, triangle1, triangle2, rectangle1, rectangle2]

shapes =

  1×8 heterogeneous Shape (Circle, EquilateralTriangle, Triangle, ...) array with properties:

    Name
    Area
```

Testing Calculate Statistics

```
>> Shape.CalculateStatistics(shapes);
Statistics for the Areas of Shapes:
Mean Area: 82.869032
Median Area: 55.269908
Standard Deviation of Areas: 99.649177
```

Task 7: Visualization:

Circle Code:

```
function Draw(obj)
    c = linspace(0, 2 * pi, 100); %Array holding 100 elements equally spaced from 0 - 2pi
    xunit = obj.Radius * cos(c);    %Calculates a vertice for each of the elements above
    yunit = obj.Radius * sin(c);

    fill(xunit, yunit, obj.Color);  %Fills the circle

    axis equal; %Grabs a square view
    axis([-obj.Radius-1, obj.Radius+1, -obj.Radius-1, obj.Radius+1]);   %1 unit + radius

    %Titles / Labels
    title(sprintf('%s Circle', obj.Color));
    xlabel('X-coordinate');
    ylabel('Y-coordinate');

    %Properties / Characteristics of circle
    propertiesText = sprintf('Radius: %.2f\nColor: %s', obj.Radius, obj.Color);
    areaText = sprintf('Area: %.2f square units', obj.Area);
    text(0, -obj.Radius - 0.5, propertiesText, 'HorizontalAlignment', 'center');
    text(0, obj.Radius + 0.2, areaText, 'HorizontalAlignment', 'center');
end
```

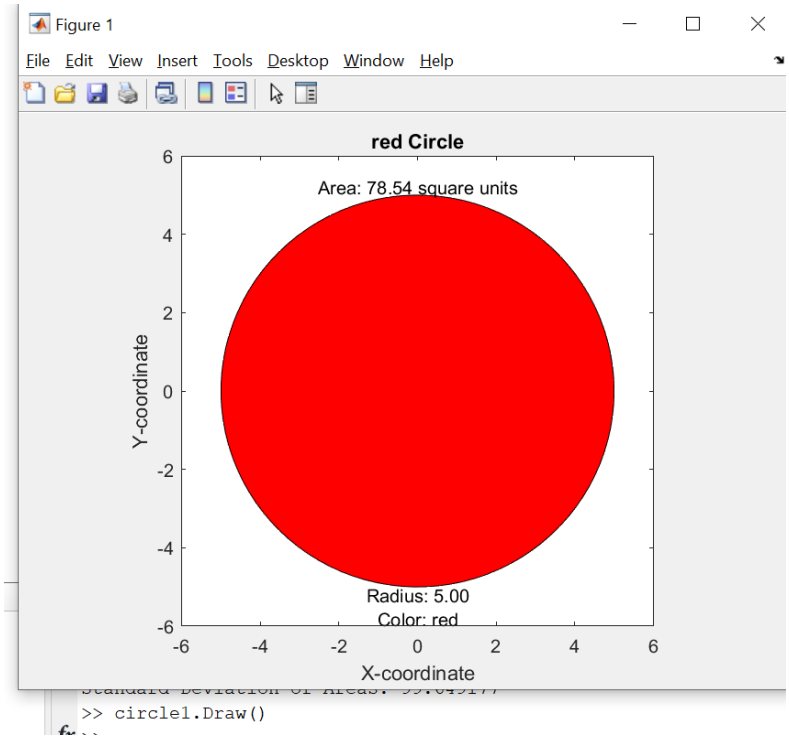
Triangle Code:

```
function Draw(obj)
    x = [0, obj.Base / 2, -obj.Base / 2, 0]; %Sets the vertices for the Triangle
    y = [obj.Height, 0, 0, obj.Height];

    fill(x, y, obj.Color); %Fills Triangle with it's color

    axis equal;     %Sets the "view" to be a square of x range ((-obj.base/2-1) - (obj.base/2+1))
    axis([-obj.Base / 2 - 1, obj.Base / 2 + 1, -1, obj.Height + 1]);    %y range = (-1 - height+1)

    % Add title and labels
    title(sprintf('%s Triangle', obj.Color));
    xlabel('X-coordinate');
    ylabel('Y-coordinate');

    % Display properties and area of the shape
    propertiesText = sprintf('Base: %.2f\nHeight: %.2f\nColor: %s', obj.Base, obj.Height, obj.Color);
    areaText = sprintf('Area: %.2f square units', obj.Area);
    text(0, -1, propertiesText, 'HorizontalAlignment', 'center');
    text(0, obj.Height + 0.2, areaText, 'HorizontalAlignment', 'center');
end
```

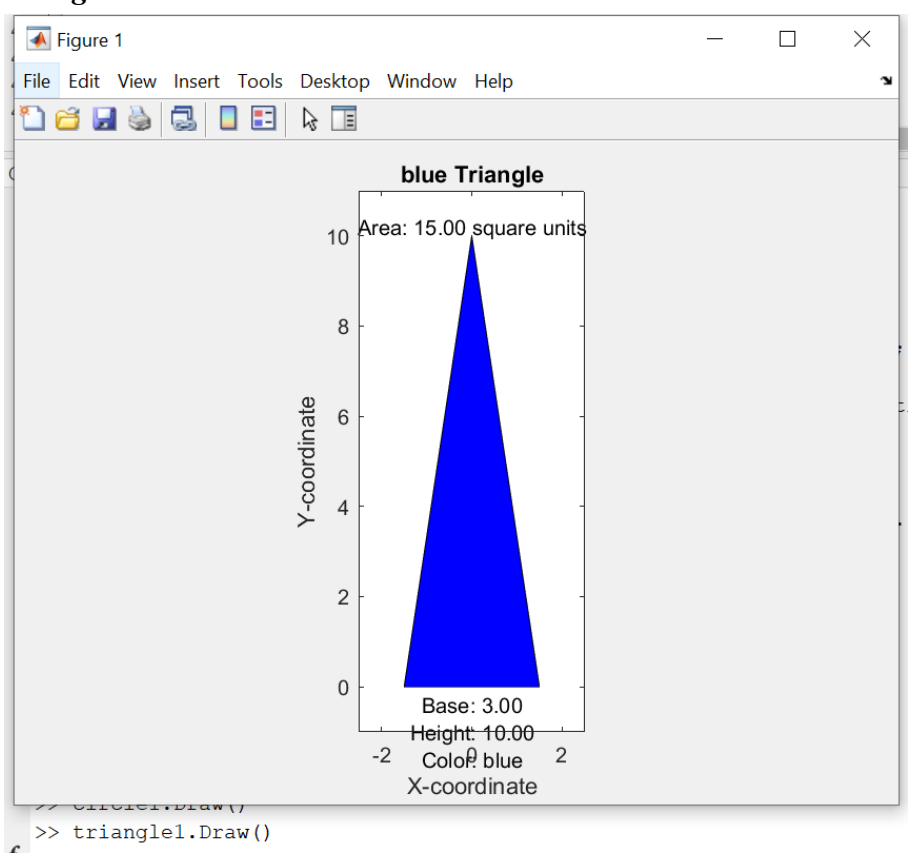
Rectangle Code:

```
function Draw(obj)
    x = [-obj.Length / 2, obj.Length / 2, obj.Length / 2, -obj.Length / 2, -obj.Length / 2];    %Sets the vertices for rectar
    y = [-obj.Width / 2, -obj.Width / 2, obj.Width / 2, obj.Width / 2, -obj.Width / 2];

    fill(x, y, obj.Color);  %Fills rectangle with specified color

    axis equal; %"Square view"
    axis([-obj.Length / 2 - 1, obj.Length / 2 + 1, -obj.Width / 2 - 1, obj.Width / 2 + 1]); %Centers rectangle

    %Titles / Labels
    title(sprintf('%s Rectangle', obj.Color));
    xlabel('X-coordinate');
    ylabel('Y-coordinate');

    %Properties / Characteristics
    propertiesText = sprintf('Length: %.2f\nWidth: %.2f\nColor: %s', obj.Length, obj.Width, obj.Color);
    areaText = sprintf('Area: %.2f square units', obj.Area);
    text(0, -obj.Width / 2 - 0.5, propertiesText, 'HorizontalAlignment', 'center');
    text(0, obj.Width / 2 + 0.2, areaText, 'HorizontalAlignment', 'center');
end
```

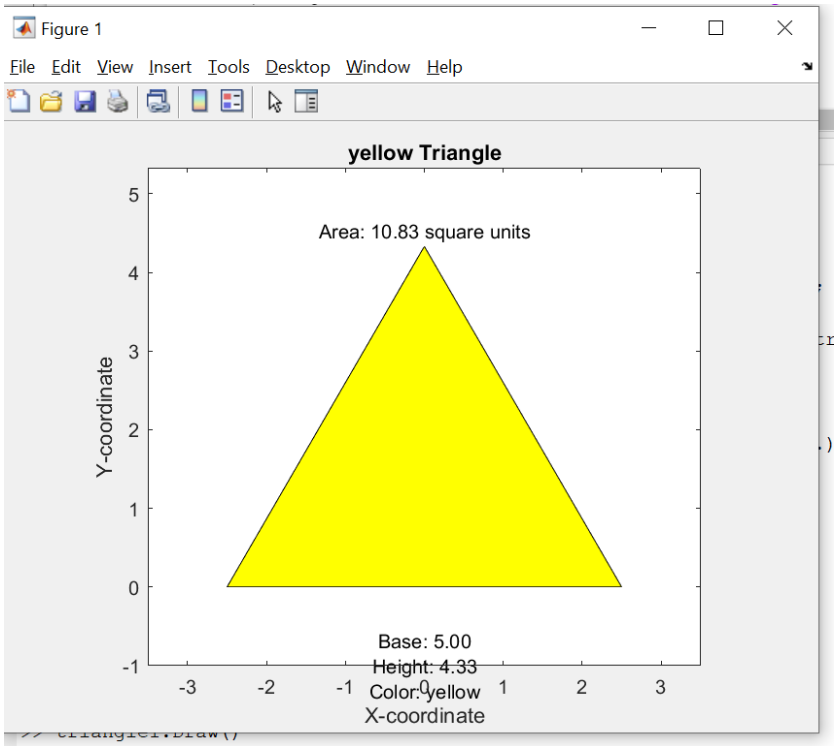
Circle Drawn:



Triangle Drawn:

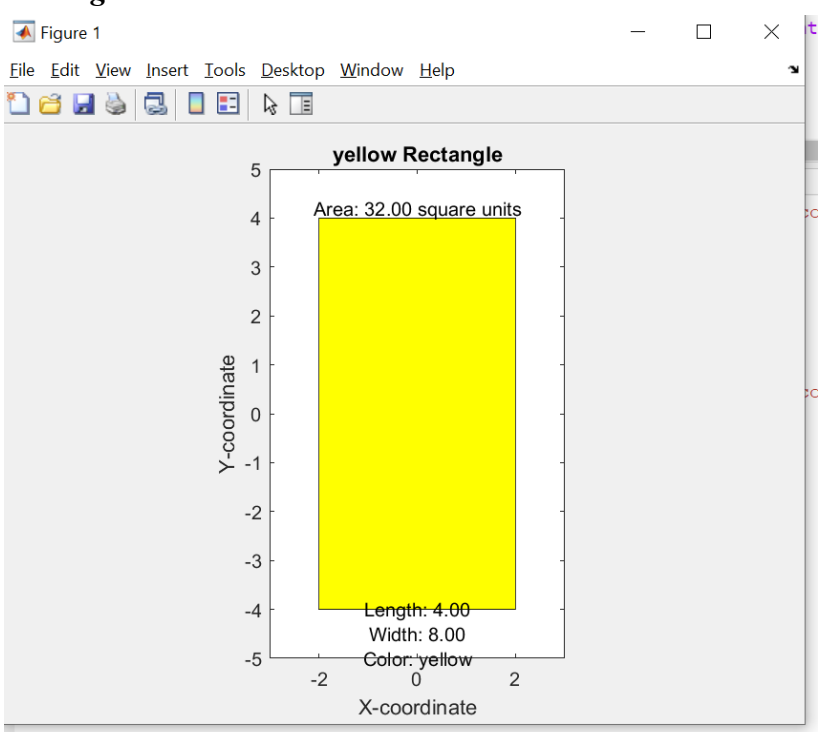


Equilateral Triangle Drawn:




```
>> eqTriangle1.Draw()
```

Rectangle Drawn:




```
>> rectangle1 = rectangle1.SetColor('yellow');
>> rectangle1.Draw()
```

****NOTE COLOR CHANGE***

Task 8: User Interaction

Code Snippet:

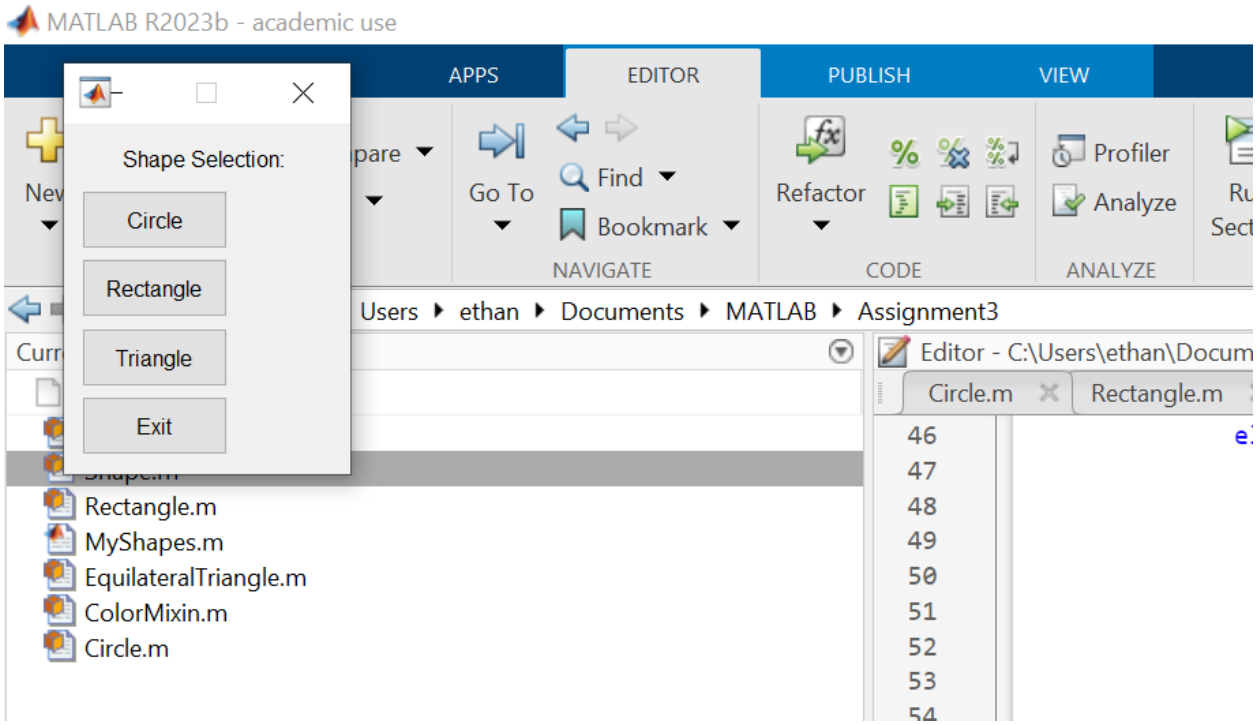
Editor - C:\Users\ethan\Documents\MATLAB\Assignment3\MyShapes.m

Circle.m | Rectangle.m | Shape.m | Triangle.m | MyShapes.m | EquilateralTriangle.m | ColorMixin.m | untitled *

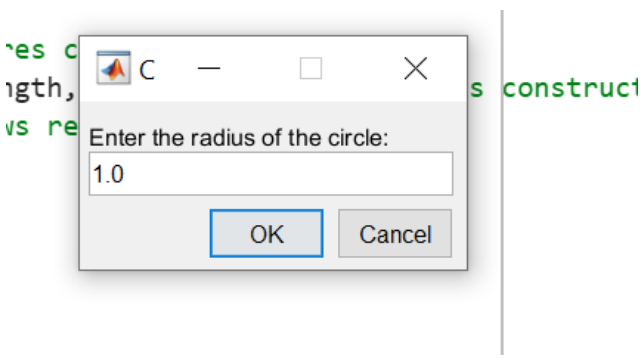
```
choices = {'Circle', 'Rectangle', 'Triangle', 'Exit'};  %Creates array of "choices"
choice = menu('Shape Selection:', choices); %Gives user a menu to choose from, storing answer in choice

while choice ~= 4   %Continues until "exit"
    switch choice
        case 1  %Circle
        prompt = 'Enter the radius of the circle:';
        dlg_title = 'Circle Input';
        num_lines = 1;
        default_answer = {'1.0'};  % Default radius
        user_input = inputdlg(prompt, dlg_title, num_lines, default_answer);    %Gives the user a pop-up, asking for radius

        if isempty(user_input)  %Checks if any input
            disp('User canceled input.');
        else
            radius = str2double(user_input{1}); %Stores radius into a temp val
            color_prompt = 'Enter the color of the circle:';
            color = inputdlg(color_prompt, dlg_title, num_lines, {'White'});    %Now asks user for color (default white)

            if isempty(color)   %Checks if empty
                disp('User canceled input.');
            else
                color = color{1};   %Stores color into temp val
                circle = Circle(radius, color); %Creates circle w/ properties given
                circle.Draw();  %Draws Circle
            end
        end
    break;

    case 2
        prompt = 'Enter the length of the rectangle:';
        dlg_title = 'Rectangle Input';
        num_lines = 1;
        default_answer = {'2.0'};  % Default length
        user_input = inputdlg(prompt, dlg_title, num_lines, default_answer);    %Asks user for length in pop-up

        if isempty(user_input)  %Checks for no input
            disp('User canceled input.');
        else
            length = str2double(user_input{1}); %Converts input to double and store
            width_prompt = 'Enter the width of the rectangle:';
            width = inputdlg(width_prompt, dlg_title, num_lines, {'2.0'});  %Ask user for a width

            if isempty(width)   %Checks if width is empty
                disp('User canceled input.');
            else
                width = str2double(width{1});   %Stores width after conversion
                color_prompt = 'Enter the color of the rectangle:';
                color = inputdlg(color_prompt, dlg_title, num_lines, {'White'});    %asks user for color

                if isempty(color)   %Checks if any input
                    disp('User canceled input.');
                else
```

```
                    color = color{1};   %Stores color
                    rectangle = Rectangle(length, width, color);    %Calls constructor
                    rectangle.Draw();   %Draws rectangle
                end
            end
        end
    break;

    case 3
        prompt = 'Enter the base of the triangle:';
        dlg_title = 'Triangle Input';
        num_lines = 1;
        default_answer = {'3.0'};  % Default base
        user_input = inputdlg(prompt, dlg_title, num_lines, default_answer);    %Asks user for base

        if isempty(user_input)  %Checks for no input
            disp('User canceled input.');
        else
            base = str2double(user_input{1});   %Stores base
            height_prompt = 'Enter the height of the triangle:';
            height = inputdlg(height_prompt, dlg_title, num_lines, {'1.0'});    %Asks for height

            if isempty(height)  %Checks no height
                disp('User canceled input.');
            else
                height = str2double(height{1}); %Stores height
                color_prompt = 'Enter the color of the triangle:';
                color = inputdlg(color_prompt, dlg_title, num_lines, {'White'});    %Asks for color

                if isempty(color)   %Checks no color
                    disp('User canceled input.');
                else
                    color = color{1};   %Stores color
                    triangle = Triangle(base, height, color);   %Creates Triangle
                    triangle.Draw();    %Draws triangle
                end
            end
        end
        break;
        otherwise   %Inputting anything but 1-4 (even closing out XD)
        disp('Invalid choice.');
        break;
    end
end
```

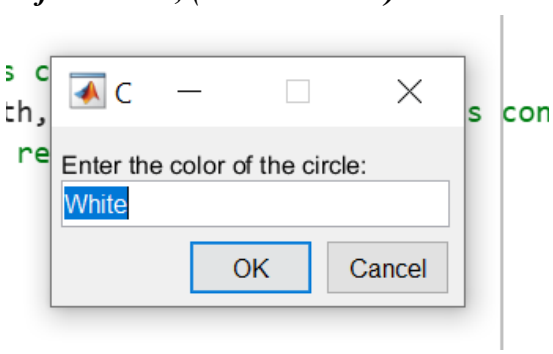
Running the Script:



Test 1: Circle

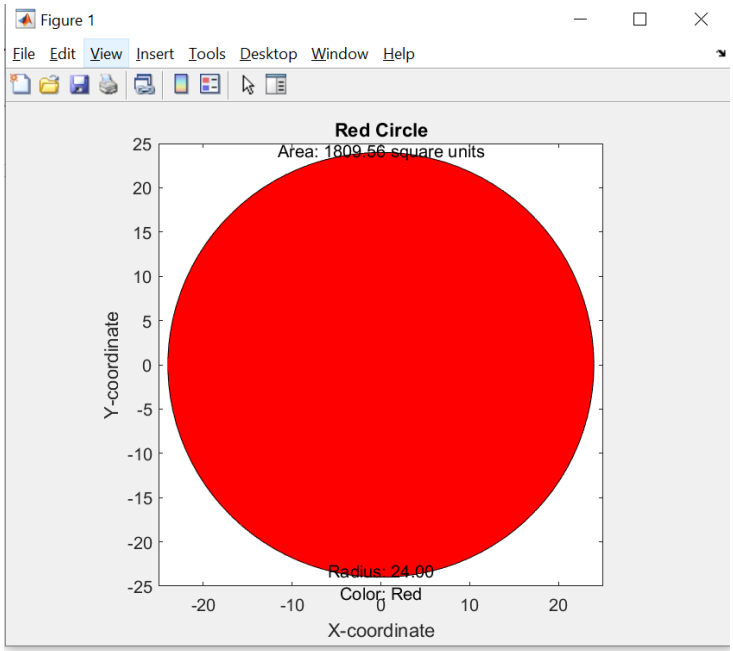


Default = 1.0, (I will enter 24).

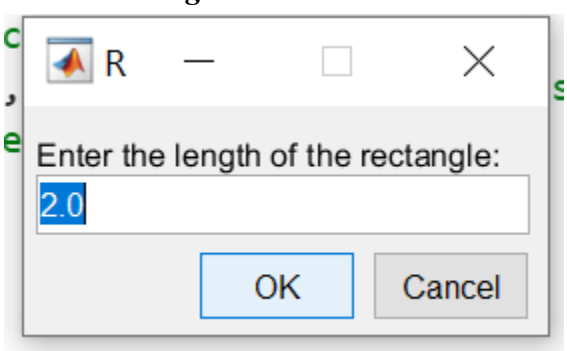


Default = White, (I will enter Red)

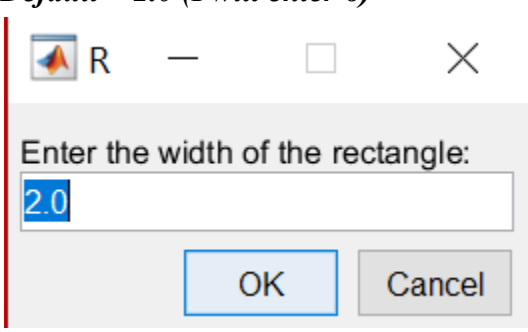
Test 1 (Circle) Output:



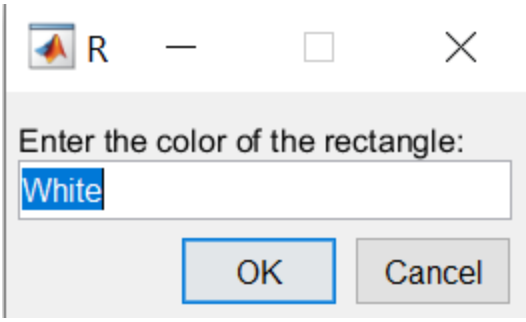
Test 2: Rectangle



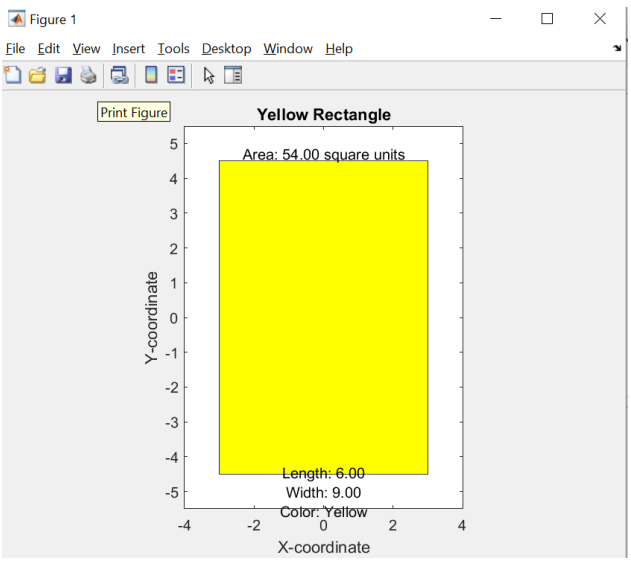
Default = 2.0 (I will enter 6)



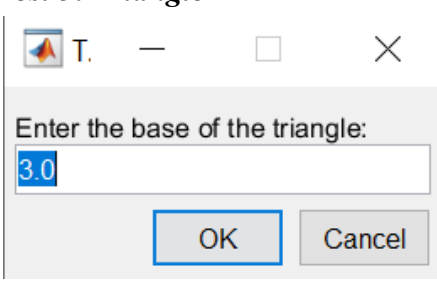
Default = 2.0 (I will enter 9)



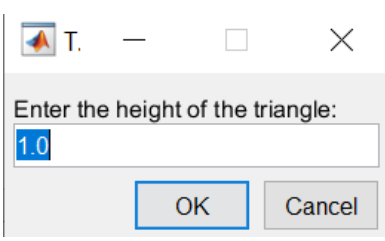
Default = White (I will enter Yellow)



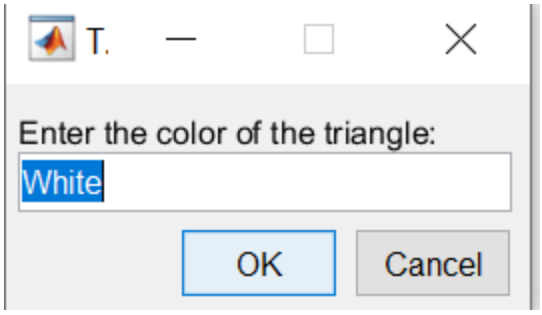
Test 3: Triangle



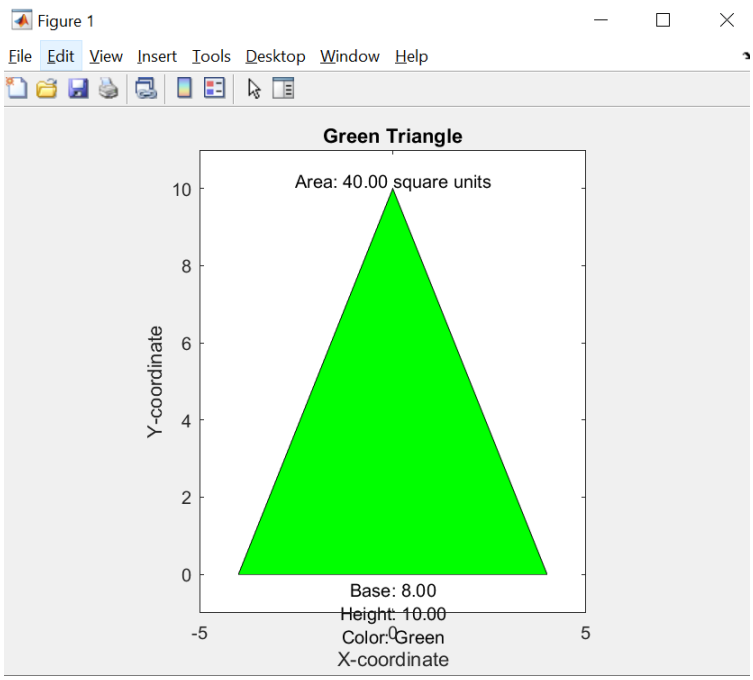
Default = 3 (I will enter 8)



Default = 1.0 (I will enter 10)



Default = white (I will enter Green)



Test 4: Exit

Closes Window / Program (Nothing to show since silent exit)

Function Interactions

These are interactions that I thought would be important but weren't specifically specified

SetColor / GetColor (With a Shape)

```
>> rectangle1 = Rectangle(3,5,'white');
>> rectangle1

rectangle1 =

  Rectangle with properties:

    Length: 3
     Width: 5
      Name: 'Rectangle'
      Area: 15
     Color: 'white'

>> rectangle1 = rectangle1.SetColor('green');
>> rectangle1

rectangle1 =

  Rectangle with properties:

    Length: 3
     Width: 5
      Name: 'Rectangle'
      Area: 15
     Color: 'green'

>> rectangle1.GetColor()

ans =

    'green'
```

CalculateArea() Function Properly updates Area…

```
rectangle1 =

  Rectangle with properties:

    Length: 3
     Width: 5
      Name: 'Rectangle'
      Area: 15
     Color: 'green'

>> rectangle1.Length = 15

rectangle1 =

  Rectangle with properties:

    Length: 15
     Width: 5
      Name: 'Rectangle'
      Area: 15
     Color: 'green'

>> rectangle1.CalculateArea()

ans =

  Rectangle with properties:

    Length: 15
     Width: 5
      Name: 'Rectangle'
      Area: 75
     Color: 'green'
```

Although with proper data encapsulation, you shouldn't be able to modify fields like this, Calculate Area properly updates the Area given a property's length / width.