Objective

Create a Custom View class that is a sub-class of UIView

Summary

• Create a new iPhone OS Window-based Application

• Create a Controller class that maintains the number of sides you will display. The Controller should also provides actions for incrementing and decrementing the count.

• The controller should have an IBOulet that will be used to establish a connection to your custom view using Interface builder.

• Using XCode, create a new UIView subclass, call it PolygonView.

• Open MainWindow.xib and layout an interface with 2 buttons that increment / decrement the number of sides that you will display in your view.

• Using Interface Builder, place an empty Custom View (UIView from the Library) onto the Window.

• Using Interface Builder’s Class Identity Inspector, set the class of the custom view to PolygonView

• Using Interface Builder, create an instance of your Controller class and wire up the necessary connections to your PolygonView and your buttons.

• Modify PolygonView to have a means to set the number of sides of the polygon by accepting a message from the Controller.

• Derive the geometry and trigonometry of polygons and in your drawRect: method use Core Graphics to draw the polygon.

• Remember, you need to call setNeedsDisplay on your view to get it to redraw itself when you increment / decrement the number of sides.
Stuff You Need to Know

Core Graphics Contexts
To draw using Core Graphics from within UIView custom views, you must first establish a Context.

Here is an example of how you might start a -drawRect: method for your view.

-(void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    /* do some drawing to context */
    ...
}

-(void)addTriangleToContext:(CGContextRef)context
{
    CGRect bounds = [self bounds];
    bounds = CGRectInset(bounds, 50.0, 50.0);

    CGPoint p1 = CGPointMake(CGRectGetMaxX(bounds), CGRectGetMinY(bounds));
    CGContextMoveToPoint(context, p1.x, p1.y); // sets the starting point

    CGPoint p2 = CGPointMake(CGRectGetMaxX(bounds), CGRectGetMaxY(bounds));
    CGContextAddLineToPoint(context, p2.x, p2.y);

    CGPoint p3 = CGPointMake(CGRectGetMinX(bounds), CGRectGetMinY(bounds));
    CGContextAddLineToPoint(context, p3.x, p3.y);

    CGContextClosePath(context); // creates a straight line from the last point to
    // the starting point
}

-(void)drawRect:(CGRect)rect
{
    CGContextRef context = UIGraphicsGetCurrentContext();
    [self addTriangleToContext:context];
    [[UIColor cyanColor] set];
    CGContextFillPath(context);

    [self addTriangleToContext: context];
    [[UIColor magentaColor] set];
    CGContextStrokePath(context);
}
**Trigonometry Review**

Consider a polygon with its enclosing circle. The point $c$ is the center point of the bounds rectangle of your custom view and (for this exercise) the center of the circle that encloses the polygon.

For a regular polygon with $V$ vertices and equal length edges, vertices are spaced at equal angles around the circle. In the picture below, the vertices divide the circle into quarters.

![Diagram of a circle with vertices](image)

From the centre point at $c$ to the vertex point $p_2$ with a radius $r$ of the enclosing circle and angle $\theta$, the coordinates $p_{2.x}$ and $p_{2.y}$ are given by:

$$p_{2.x} = c.x + r \cdot \cos \theta$$
$$p_{2.y} = c.y + r \cdot \sin \theta$$

You can use the UNIX math library functions `cosf` and `sinf` which return their values as floats.

Note that `cosf` and `sinf` require their arguments in radians. The math library also defines the symbol `M_PI` for $\pi$. Review from your trigonometry that there are $2 \pi$ radians in a circle. We do not need to use degrees - we can just do everything in radians. In the case of a polygon with four sides, vertices are at multiples of $2 \pi / 4 = \pi / 2$ around the circle.

For the first point, perform a `CGContextMoveToPoint` call. For subsequent points, use `CGContextLineToPoint`. Finally complete the path with `CGContextClosePath`.

**Hints**

Do not invoke your `-drawRect:` method directly. If you wish to trigger a draw (say, when you change the number of sides), use the `UIView -setNeedsDisplay` method instead.