Challenge 8.2 - Stop Watch: Create a Protocol and a Delegate Object

The StopWatch class and the ViewController class need to create a way to communicate and update the UI. You will need to create a structured way, a protocol, to send messages from the StopWatch to the ViewController object. A common design pattern in Objective-C is the delegate pattern. To implement a delegate pattern you will need to create a protocol that describes the messages (methods) to send, and store a weak reference to another object. When an object implements the methods of a protocol it is called conforming to the protocol.

Create an Objective-C *@protocol* at the top of StopWatch.h. Make a required method to notify the ViewController class that the StopWatch timer did update. A standard practice with delegate methods is to pass the object (i.e. StopWatch instance) as a parameter in the delegate method. Use names that make sense following the format of a protocol below.

```
@protocol MyCustomObjectProtocol <NSObject>
// Required method
- (void)myCustomObjectChanged:(MyObject *)myObject;

// Optional method
@optional
- (void)somethingMinorChanged;
@end
```

Create a delegate property for the StopWatch object, so that it can notify another object (type id) when the StopWatch timer has changed. The delegate property allows you to link two code files together without having a dependency between the two files.

Note: We'll want to use the *weak* property attribute, so that the pointer will auto-zero (*nil*) when the delegate object is cleaned up. If we don't use *weak*, we'll need to set the delegate to *nil* in a dealloc method.

```
@property (nonatomic, weak) id<MyCustomProtocol> delegate;
```

Now that you have a protocol and a delegate object, you need to make the ViewController conform to the protocol. Add the delegate name in angled brackets in ViewController.m.

```
#import "StopWatch.h"
@interface ViewController () <StopWatchDelegate> {
```

Note: You'll have to implement any methods from your protocol to fix the compiler warnings. Conform to the protocol by implementing your StopWatchDelegate method.

```
- (void)myCustomObjectChanged:(MyObject *)myObject {
    // your logic here
}
```

In order to calculate the current time, you'll need to add a new method to StopWatch to calculate the duration (in seconds) using the private ivar _currentDate and _startTime.

```
- (NSTimeInterval)duration {
    return [_currentDate timeIntervalSinceDate:_startDate];
}
```

The last step is necessary or nothing will happen. You'll need to set the delegate object property on the StopWatch object to the current instance of the ViewController. In viewDidLoad:, you'll add the code to set the delegate to self.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    _stopWatch = [[StopWatch alloc] init];
    _stopWatch.delegate = self;
}
```

Bonus 1: In StopWatch.h/m add a @property to set a custom date format NSString. Update the formatTimeInterval: method to use the new property. Hint: You'll want to create an init method to set the default format, if the user of StopWatch doesn't set it.

```
@property (nonatomic, copy) NSString *dateFormatString;
```

Bonus 2: Remove the code to in ViewController.m's resetTimer method to set the label to "00:00:00.000". Instead make the StopWatch resetTimer method invoke the delegate method to redisplay the StopWatch duration.

Bonus 3: The StopWatch duration method has a bug. It doesn't work properly after reseting the StopWatch from Bonus 2. Fix the bug.

Resources

https://developer.apple.com/library/ios/documentation/cocoa/conceptual/ProgrammingWithObjectiveC/EncapsulatingData/EncapsulatingData.html