Minor HW

@Synthesize

1. Thank to autosynthesis you don't need to explicitly synthesize the property as it will be automatically synthesized by the compiler as
2. @synthesize propertyName = _propertyName
3. However, a few exceptions exist:
    - **readwrite property with custom getter and setter**
    - when providing **both** a getter and setter custom implementation, the property won't be automatically synthesized
    - **readonly property with custom getter**
    - when providing a custom getter implementation for a readonly property, this won't be automatically synthesized
    - **@dynamic**
    - when using @dynamic propertyName, the property won't be automatically synthesized (pretty obvious, since @dynamic and @synthesize are mutually exclusive)
    - **properties declared in a @protocol**
    - when conforming to a protocol, any property the protocol defines won't be automatically synthesized
    - **properties declared in a category**
    - this is a case in which the @synthesize directive is not automatically inserted by the compiler, but this properties cannot be manually synthesized either. While categories can declare properties, they cannot be synthesized at all, since categories cannot create ivars. For the sake of completeness, I'll add that's it's still possible to fake the property synthesis using the Objective-C runtime.
    - **overridden properties** (new since clang-600.0.51, shipping with Xcode 6, thanks Marc Schlüpmann)
    - when you override a property of a superclass, you must explicitly synthesize it

It's worth noting that synthesizing a property automatically synthesize the backing ivar, so if the property synthesis is missing, the ivar will be missing too, unless explicitly declared.

Except for the last three cases, the general philosophy is that whenever you manually specify all the information about a property (by implementing all the accessor methods or using @dynamic) the compiler will assume you want full control over the property and it will disable the autosynthesis on it.

Apart from the cases that are listed above, the only other use of an explicit @synthesize would be to specify a different ivar name. However conventions are important, so my advice is to always use the default naming.

@Synchronize

The @synchronized directive is a convenient way to create mutex locks on the fly in **Objective-C** code.

The @synchronized directive does what any other mutex lock would do—it prevents different threads from acquiring the same lock at the same time.

@Dynamic

The @dynamic keyword tells the compiler that you will provide accessor methods dynamically at runtime. This can be done using the Objective-C runtime functions.

Typically, you would use @dynamic with things like Core Data, where Core Data will provide the accessors based on the Core Data model.

You are correct that in most normal cases you would not use @dynamic. Typically, you would just use @property or @property and @synthesize.

Diff between iVars and external Vars

var is a generic variable, of course. Use this when you don't care to make any further distinction about the variable you are documenting.

ivar is an "instance variable", or a variable that is set on an instance object (an instance of a class). Typically these would be defined (in Python) inside of an __init__ method.

cvar is a "class variable", or a variable that is set on a class object directly. Typically, this would be set inside a class statement, but outside of any particular method in the class.

Getter and Setters in Objc

Getter is a method which gets called every time you access (read value from) a property (declared with @property). Whatever that method returns is considered that property's value:

```
@property int someNumber;

...

- (int)someNumber {
  return 42;
}

...

NSLog("value = %d", anObject.someNumber); // prints "value = 42"
```

Setter is a method which gets called every time property value is changed.

```
- (void)setSomeNumber: (int)newValue { // By naming convention, setter for `someValue` should
                                       // be called `setSomeValue`. This is important!
  NSLog("someValue has been assigned a new value: %d", newValue);
}
```

Method Swizzling

Method swizzling is the process of changing the implementation of an existing selector at runtime. *Simply speaking, we can change the functionality of a method at runtime.*

For example: If you want to track the keys and values added to the UserDefaults and add a prefix string before all the keys, you can switch the implementation of the setValue:forKey method with your own method. So all the calls made to the setValue:forKey method will be routed to the new selector method. You can create the new key string with the prefix added and call the original implementation of setValue:forKey method with the new key.

References

https://stackoverflow.com/questions/19784454/when-should-i-use-synthesize-explicitly

https://stackoverflow.com/questions/6317889/what-does-synchronized-do-as-a-singleton-method-in-objective-

c#:~:text=The%20%40synchronized%20directive%20is%20a,lock%20at%20the%20same%20time.

https://stackoverflow.com/questions/12445463/dynamic-keyword-meaning-in-ios

https://stackoverflow.com/questions/41052221/what-is-the-difference-between-var-cvar-and-ivar-in-pythons-sphinx

https://stackoverflow.com/questions/10425827/please-explain-getter-and-setters-in-objective-c

https://abhimuralidharan.medium.com/method-swizzling-in-ios-swift-1f38edaf984f