**SubClasses Vs Protocols**

**Subclasses:** Every object you create in your application descends from the “NSObject” foundation class. The NSObject class includes properties and methods which apply to all objects. The NSObject class is divided into smaller groups of objects, called subclasses.

**Protocol:** The idea is to provide a way for classes to share the same method and property declarations without inheriting them from a common ancestor. This is especially useful in Objective-C when using the [delegation pattern](http://en.wikipedia.org/wiki/Delegation_pattern), on which the iOS and Mac OS SDKs rely heavily.

***-------------------------------------------------------------------------------------------------------------------------------***

***Protocol Syntax:***

***-------------------------------------------------------------------------------------------------------------------------------***

*@protocol prot1 <NSObject> or :<NSProtocol>*

*//methods*

*@end*

*Implementation:*

*//Must import protocol file.*

*@interface class\_name: NSObject <prot1>*

*{ }*

*//methods*

*@end*

***-------------------------------------------------------------------------------------------------------------------------------***

***Subclass Syntax:***

***-------------------------------------------------------------------------------------------------------------------------------***

@interface sub\_class:super\_class

*//methods*

@end

@implementation sub\_class

*//definitions*

@end

Protocols are just the rules or properties to be used in the inherited classes to relate two different type of objects. Hence no concept of over riding. A class must implement all methods in a protocol it declares itself as adopting.

A subclass can override a method it inherits from the class it is based on. The class it is based on is called its superclass. Hence inheriting the variables and methods through which super class can be modified.

For example, you may have a "car" class which has properties that are common to all cars such color, tires, engine, etc. For some reason you need to differentiate between different types of cars but you don't want to repeat all those properties and behaviors in each new class so you write subclasses of the "car" class. So you may have subclasses such as "manual cars", "automatic cars.

But in protocols, A text field class might have a [delegate](http://en.wikipedia.org/wiki/Delegation_(programming)) that implements an informal protocol with an optional method for performing auto-completion of user-typed text. The text field discovers whether the delegate implements that method (via [reflection](http://en.wikipedia.org/wiki/Reflection_(computer_science))) and, if so, calls the delegate's method to support the auto-complete feature.