From: Stack: http://stackoverflow.com/questions/5431413/difference-between-protocol-and-delegates

A protocol, declared with the (@protocol syntax in Objective-C) is used the declare a set of methods that a class that "adopts" (declares that it will use this protocol) will implement. This means that you can specify in your code that, "you don't care which class is used so long as it implements a particular protocol". This can be done in Objective-C as follows:

id<MyProtocol> instanceOfClassThatImplementsMyProtocol;

If you state this in your code, then any class that "conforms" to the protocol MyProtocol can be used in the variable instanceOfClassThatImplementsMyProtocol. This means that the code that uses this variable knows that it can use whichever methods are defined in MyProtocol with this particular variable, regardless of what class it is. This is a great way of avoiding the inheritance design pattern, and avoids tight coupling.

Delegates are a use of the language feature of protocols. The [delegation design pattern](http://en.wikipedia.org/wiki/Delegation_pattern) is a way of designing your code to use protocols where necessary. In the Cocoa frameworks, the delegate design pattern is used to specify an instance of a class which conforms to a particular protocol. This particular protocol specifies methods that the delegate class should implement to perform specific actions at given events. The class that uses the delegate knows that its delegate coforms to the protocol, so it knows that it can call the implemented methods at given times. This design pattern is a great way of decoupling the classes, because it makes it really easy to exchange one delegate instance for another - all the programmer has to do is ensure that the replacement instance or class conforms to the necessary protocol (i.e. it implements the methods specified in the protocol)!

Protocols and delegates are not restricted only to Objective-C and Mac/iOS development, but the Objective-C language and the Apple frameworks make heavy use of this awesome language feature and design pattern.

**Edit:**

Here's an example. In the UIKit framework of Cocoa Touch, there is a UITextFieldDelegate protocol. This protocol defines a series of methods that classes which are delegates of a UITextField instance should implement. In other words, if you want to assign a delegate to a UITextField (using thedelegate property), you'd better make sure that this class conforms to UITextFieldDelegate. In fact, because the delegate property of UITextField is defined as:

@property(nonatomic, assign) id<UITextFieldDelegate> delegate

Then the compiler will give warnings if you assign a class to it that doesn't implement the protocol. This is really useful. You have to state that a class implements a protocol, and in saying that it does, you're letting other classes know that they can interact in a particular way with your class. So, if you assign an instance of MyTextFieldDelegateClass to the delegate property of UITextField, the UITextField**knows** that it can call some particular methods (related to text entry, selection etc.) of yourMyTextFieldDelegateClass. It knows this because MyTextFieldDelegateClass has said that it will implement the UITextFieldDelegate protocol.