A subclass inherits methods and instance variables from its superclass.

If we have a class called classname and we would like it to have a subclass called subclassname .It can be declared as:

@interface subclassname : classname

{

// instance variables that subclassname has but classname lacks go here

}

// methods that subclassname has and classname may or may not go here. If both have it, subclassname's implementation override's classname's so long as an instance of subclassname is references.

@end

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. If subclassname overrides a method named methodname it inherits from classname but it'd like to use the classname variation on methodname instead of its own, it can do something like:

[super methodname];

Subclassing is just a subclass of another class and it inherits behavior and members of its parent class.

For example, we may have a "car" class which has properties that are common to all cars such color, tires, engine, etc. It also has behaviors that are common to all cars such as drive forward, drive backwards, stop, etc.

Now for some reason we need to differentiate between different types of cars but don't want to repeat all those properties and behaviors in each new class so we write subclasses of the "car" class.

So we may have subclasses such as "manual cars", "automatic cars", "convertibles", "electric cars", etc. All of these subclasses have some common traits inherited from the "car" class but will also have some unique characteristics that can be included in the subclass.

protocol

A protocol declares a programmatic interface that any class may choose to implement. Protocols make it possible for two classes distantly related by inheritance to communicate with each other to accomplish a certain goal. They thus offer an alternative to subclassing. Any class that can provide behavior useful to other classes may declare a programmatic interface for vending that behavior anonymously. Any other class may choose to adopt the protocol and implement one or more of its methods, thereby making use of the behavior. The class that declares a protocol is expected to call the methods in the protocol if they are implemented by the protocol adopter.

Formal and Informal Protocols

There are two varieties of protocol, formal and informal:

A formal protocol declares a list of methods that client classes are expected to implement. Formal protocols have their own declaration, adoption, and type-checking syntax. You can designate methods whose implementation is required or optional with the @required and @optional keywords. Subclasses inherit formal protocols adopted by their ancestors. A formal protocol can also adopt other protocols.

Formal protocols are an extension to the Objective-C language.

An informal protocol is a category on NSObject, which implicitly makes almost all objects adopters of the protocol. (A category is a language feature that enables you to add methods to a class without subclassing it.) Implementation of the methods in an informal protocol is optional. Before invoking a method, the calling object checks to see whether the target object implements it. Until optional protocol methods were introduced in Objective-C 2.0, informal protocols were essential to the way Foundation and AppKit classes implemented delegation.

Adopting and Conforming to a Formal Protocol

A class can either declare adoption of a formal protocol or inherit adoption from a superclass. The adoption syntax uses angle brackets in the @interface declaration of the class. In the following example, the CAAnimation class declares its superclass to be NSObject and then formally adopts three protocols.

@interface CAAnimation : NSObject <NSCopying, CAMediaTiming, CAAction>

A class—and any instance of that class—are said to conform to a formal protocol if the class adopts the protocol or inherits from another class that adopts it. Conformance to a protocol also means that a class implements all the required methods of the protocol. You can determine at runtime whether an object conforms to a protocol by sending it a conformsToProtocol: message.