CATEGORIES

* Categories collect method implementations into separate files.
* The programmer can place groups of related methods into a category to make them more reliable.
* For instances one could create a “Reverse String” category “on” String object, while collecting all of the methods related to Reverse String.
* It contains .h and .m files.
* The methods within a category are added to a class at runtime. Thus, categories permit the programmer to add method to an existing class without the need to recompile that or even have access to its source code.
* When we write Object Oriented Programs, we will often want to add some behavior to an existing class. There are always new hoops for objects to jump through.
* For ex: We might have designed a new kind of tire, so we would subclass Tire & add the new behavior. When we want to add behavior to an existing class, we usually create a sub class.
* But sometimes sub classing isn't convenient. For ex: we may want to add new behaviors to NSString, but we realize that NSString is really the front end for a class cluster, and So it's difficult to sub class.
* In other cases, you might be able to make a subclass, but you are using a toolkit or library that won't be able to handle objects of the new class.
* For Ex: SubClass of NSString won't be returned when we create a new string with the string with format class method. The dynamic run-time dispatch mechanism employed by the Objective-C term for those new methods is 'categories'.

**Creating a category**

* A category is a way to add new method to existing classes thus can be done to any class, even classes we don't have the source code for.
* Let us say we are writing a cross word puzzle App that takes a series of strings, determines the length of each string, them puts those length into an NSArray or NSDictionary.

NSNumber \*number;

number = [NSNumber numberWithUnsignedInt:[Stringlength]];

//do something with number

@interface

//The declaration of a category looks a lot like the declaration for a class.

@interface NSString(NumberConvenience)

-(NSNumber \*) lengthAsNumber;

@end//Number Convenience

First, an existing class is mentioned, followed by a new name in parenthesis.

This means that the category is called "NumberConvenience", and it adds methods to NSString. Another way to say this is we are adding a category onto NSString called NumberConvenience. You can add as many categories to a class as you want as long as the category names are unique.

We indicate the class you are putting the category onto (NSString), and the name of the category (Number convince), and list the methods you are adding, following by @end.

Now instance variables cant be added, so there is no instance variable section as there is with a class declaration.

@implementation

There is an @implementation companion to @interface

@implementation NSString (Number convince )

-(NSNumber \* )lengthAs Number

{

unsigned int length =[self length];

return ([NSNumber number withUnsigned int:length]);

}

@end

@implementation has the names of the class the category, along with the bodies of the new methods.

The lenghtAsNumber method gets the length of the string by calling [self length].This well be string to which you send the length AsNumber. then a new NSNumber is added with length.

numberWithUnsignedInt is not ' alloc','copy',or' new'method.

The NSNumber Object we create will get cleaned up when the currently active auto release pool is destroyed.

**Limitations of Categories:**

* New instance variable to a class can't created
* The second limitations concerns name collusion ,in which one the category methods has the name as an existing method. when names collide the category
* Category methods will completely replace the original method, with no way of getting the original back.