OBJECTIVE-C PROTOCOLS

Objective-C allows you to define protocols, which declare the methods expected to be used for a particular situation. Protocols are implemented in the classes conforming to the protocol.

A simple example would be a network URL handling class, it will have a protocol with methods like processCompleted delegate method that intimates the calling class once the network URL fetching operation is over.

A syntax of protocol is shown below.

```
@protocol ProtocolName
@required
// list of required methods
@optional
// list of optional methods
@end
```

The methods under keyword **@required** must be implemented in the classes that conforms to the protocol and the methods under **@optional** keyword are optional to implement.

Here is the syntax for class conforming to protocol

```
@interface MyClass : NSObject <MyProtocol>
...
@end
```

This means that any instance of MyClass will respond not only to the methods declared specifically in the interface, but that MyClass also provides implementations for the required methods in MyProtocol. There's no need to redeclare the protocol methods in the class interface - the adoption of the protocol is sufficient.

If you need a class to adopt multiple protocols, you can specify them as a comma-separated list. We have a delegate object that holds the reference of the calling object that implements the protocol.

An example is shown below.

```
#import <Foundation/Foundation.h>

@protocol PrintProtocolDelegate

- (void)processCompleted;

@end

@interface PrintClass :NSObject
{
    id delegate;
}

- (void) printDetails;
- (void) setDelegate:(id)newDelegate;
@end

@implementation PrintClass

- (void)printDetails{
    NSLog(@"Printing Details");
    [delegate processCompleted];
}

- (void) setDelegate:(id)newDelegate{
```

```
    delegate = newDelegate;
}

@end


@interface SampleClass:NSObject<PrintProtocolDelegate>

- (void)startAction;

@end

@implementation SampleClass

- (void)startAction{
    PrintClass *printClass = [[PrintClass alloc]init];
    [printClass setDelegate:self];
    [printClass printDetails];
}

-(void)processCompleted{
    NSLog(@"Printing Process Completed");
}

@end


int main(int argc, const char * argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    SampleClass *sampleClass = [[SampleClass alloc]init];
    [sampleClass startAction];
    [pool drain];
    return 0;
}
```

Now when we compile and run the program, we will get the following result.

```
2013-09-22 21:15:50.362 Protocols[275:303] Printing Details
2013-09-22 21:15:50.364 Protocols[275:303] Printing Process Completed
```

In the above example we have seen how the delgate methods are called and executed. Its starts with startAction, once the process is completed, the delegate method processCompleted is called to intimate the operation is completed.

In any iOS or Mac app, we will never have a program implemented without a delegate. So its important we understand the usage of delegates. Delegates objects should use unsafe_unretained property type to avoid memory leaks.