Objective C: Protocols vs. Subclasses

What is a Protocol?

In the real world, a protocol is a set of rigid rules that a person must follow. Now apply that to the universe of programming, especially Objective C. A protocol is a set of rigid “rules” that a class must follow when performing a specific task. In the case, a “rule” is the methods that a class MUST implement when performing a specific task.

Let’s take the example of a server that we might be building. A server is only complete when it has methods to send and receive data. Therefore it must follow the protocol of implementing the send and receive functions.

And, what is a Subclass?

Many times, Objective C requires classes to work with one another. A situation like this demands that a class have certain properties and characteristics that it shares with every other class it interacts with. Inheritance is the age old solution to this problem, and Objective C is no different. All classes are expected to inherit properties from NSObject, and these classes are essentially, subclasses.

A subclass, therefore, is a class that inherits properties from another class.

So why are the two different?

A protocol contains only method declarations that a class using the protocol must implement. A protocol never specifies what the implementation of a method should be, as each class may have a specialized application of the method.

A subclass, on the other hand, is an extension of an already defined class, and therefore may or may not implement the methods already defined in its super class. There is no enforcement of overloading the methods and this is only done on a need basis.

Here are a few examples showing the difference between the two.

```
//Defining a Protocol
#import <Foundation/Foundation.h>
@protocol Server <NSObject>
-(void) send;
-(void) receive;
@end


//Using a Protocol
#import <Foundation/Foundation.h>
@interface MyServer <Server>
-(int) createSocket:(int) port;
@end


//Using the protocol
#import "MyServer.h"
@implementation MyServer
-(int) createSocket:(int)port
{
//Code goes here
}
-(void) send
{
//Code goes here
}
-(void) receive
{
//Code goes here
}
@end
```

```
//Defining a subclass
#import <Foundation/Foundation.h>
@interface MyHttpServer:MyServer
-(void) sendHttpRequest;
-(void) getHttpResponse;
@end


//Defining a subclass
#import "MyHttpServer.h"
@implementation MyHttpServer
-(void) sendHttpRequest
{
//Code goes here
}
-(void) getHttpRequest
{
//Code goes here
}
@end
```

Mithil Arun

1PI10IS054