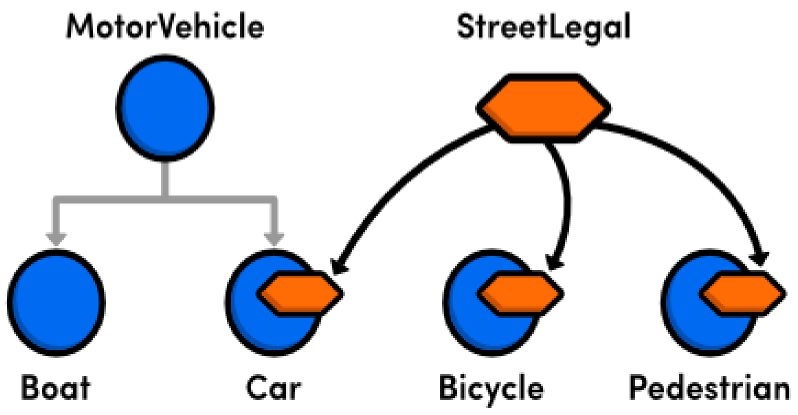
Objective C: Protocols vs. Subclasses

PROTOCOL

Protocols are a way of enforcing certain methods to be utilized, regardless of the actual class the object is part of, thus ensuring that a certain form of method template is implemented. A protocol is a group of related properties and methods that can be implemented by *any* class. They are more flexible than a normal class interface, since they let you reuse a single API declaration in completely unrelated classes. This makes it possible to represent horizontal relationships on top of an existing class hierarchy. Protocols also have an optional property which allows user to implement the method only if they want to.

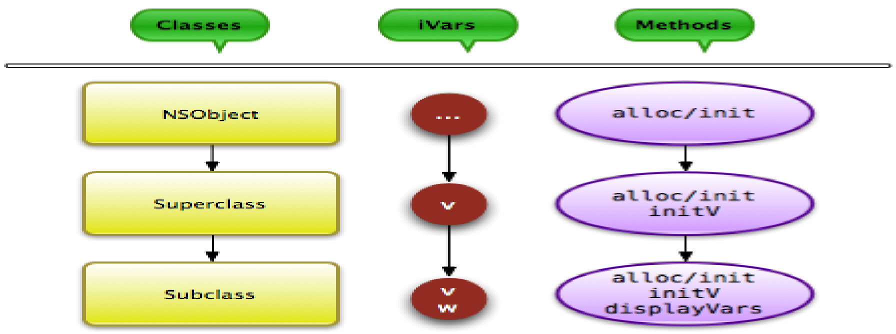


Unrelated classes adopting the `StreetLegal` protocol

SUBCLASS

Objective C requires classes to work with one another. There is a requirement that a class having certain properties and characteristics shares it with every other class it interacts with. All classes inherit properties from NSObject, and these classes are essentially, subclasses.

A subclass is a class definition that inherits the definition of a superclass. All of the methods that are defined in the superclass are automatically defined in the subclass. A real world analogy between a superclass and a subclass is the definition of a smart phone, and an iPhone. Smart phone would be a superclass that might define a method to “place a call”, while iPhone might define more specific methods to “open iTunes” and would not have to re-define the generic smart phone methods like the need to place a call.



PROTOCOL V/S SUBCLASS

A protocol contains only method declarations. A class using the protocol must define these methods. Various classes may use the same protocol with each having its own implementation of the methods declared in the protocol. A protocol never defines the methods on its own.
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.

EXAMPLES

Protocol

```
//Creating a Protocol with method declaration
#import <Foundation/Foundation.h>
@protocol protocol_name <NSObject>
-(void) function_1;
-(void) function_2;
@end

//Adopting a Protocol
#import <Foundation/Foundation.h>
@interface interface_name <protocol_name>
-(void) interface_function;
@end

//Defining the protocol methods and interface methods
#import "headerfile.h"
@implementation interface_name
-(void) interface_function
{
//Code
}
-(void) function_1
{
//Code
}
-(void) function_2
{
//Code
}
@end
```

Subclass

```
//Creating a subclass
#import <Foundation/Foundation.h>
@interface subclass:superclass
-(void) method_1;
-(void) method_2;  //Assume method_2 is also present in superclass. Hence Overriding.
@end

//Defining subclass methods
#import "headerfile.h"
@implementation subclass
-(void) method_1
{
//Code
}
-(void) method_2 //subclass method_2 differs from its superclass implementation
{
//Code
}
@end
```

Akshay Surana
1PI10IS011