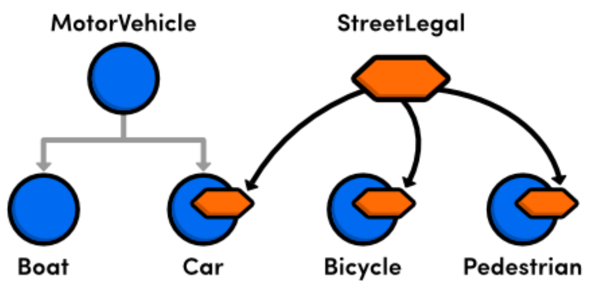
Protocols:

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.



In the figure, StreetLegal is a protocol and all the methods defined in it can be used by an unrelated class.

Any objects that inherit from a protocol are guaranteed to implement all the methods defined in it.

Example:

In the example below since Bicycle is adopting the StreetLegal protocol it is required to implement all the methods of StreetLegal. This is shown in Bicycle.m.

```
// Bicycle.h
#import <Foundation/Foundation.h>
#import "StreetLegal.h"

@interface Bicycle : NSObject <StreetLegal>

- (void)startPedaling;
- (void)removeFrontWheel;
- (void)lockToStructure:(id)theStructure;

@end
```

```objc
// Bicycle.m
#import "Bicycle.h"

@implementation Bicycle

- (void)signalStop {
    NSLog(@"Bending left arm downwards");
}
- (void)signalLeftTurn {
    NSLog(@"Extending left arm outwards");
}
- (void)signalRightTurn {
    NSLog(@"Bending left arm upwards");
}
- (void)startPedaling {
    NSLog(@"Here we go!");
}
- (void)removeFrontWheel {
    NSLog(@"Front wheel is off."
          "Should probably replace that before pedaling...");
}
- (void)lockToStructure:(id)theStructure {
    NSLog(@"Locked to structure. Don't forget the combination!");
}

@end
```

Subclasses:

Every object created is inherited from the 'NSObject' class. This class identifies properties and methods which apply to all objects. The NSObject class is divided into smaller groups of objects, called subclasses. Each subclass will have certain properties and characteristics that it shares with every other class it interacts with.

A subclass can also override a method it inherits from the class it is based on. The class it is based on is called its superclass.

Example:

If we had a class called classname and wanted to a subclass called subclassname then we would do the following:

```
@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
```