- **Selectors**
- **More On Protocols**
- **NSNumber**
- **NSValue**
- **Categories**
- **Class Extensions**

Selectors:

- A name used to refer to a method in order to execute it or pass it around
- Simply identifies a method.
- The selector of a method is just the method name minus return type, and parameters.

```

- (void)myMethodWithData:(NSString *)data;

// notice the colon
SEL mySelector = @selector(myMethodWithData:);

- (NSData *)myMethodWithString:
```

```
(NSString *)string value:(BOOL)value;

SEL myOtherSelector = @selector(myMethodWithString:value:);

```

- 2 common ways to get a selector

```
// compile time

SEL aSelector = @selector(fly);

// run time

SEL aSelector = NSSelectorFromString(@"nameOfMethod");

```

- Many framework methods expect a selector as a parameter.
- For instance, if we want to programmatically setup a target-action on a button we will call the instance method:

```

// definition
- (void)addTarget:(id)target action:(SEL)action forControlEvents:(UIControlEvents)controlEvents;

```

```
// adding it to a button
[aButton addTarget:self action:@selector(buttonTapped:) forControlEvents:UIControlEventTouchUpInside];

// actual method
- (void)buttonTapped:(UIButton *)sender {
    // do stuff
}


```

- A common use of selectors is to test whether an object can handle a message.

```
if ([anObject respondsToSelector:@selector(someMethod)]) {

 [anObject performSelector:@selector(someMethod)];

}
```

- This is a handy way of sorting an array using a selector.

```

NSArray *arr = @[@"Hello", @"Light", @"House", @"Labs"];

arr = [arr sortedArrayUsingSelector:@selector(compare:)];
NSLog(@"%@", arr);
```

Working With Selectors

More Protocols & Delegation

What are protocols?

- In the real world protocols consist of sets of agreed upon procedures, rules or conventions for doing stuff.
- E.g. police follow a legally binding protocol when making an arrest.
- They read you your rights in a specific format, etc.
- Computers communicate on the internet using the *http protocol*.
- The *http protocol* defines the expected request and the expected response data and format.
- There would be no internet without a shared protocol.
- In iOS a protocol usually consists of a group of method signatures (and sometimes properties) that any conforming class agrees to implement.
- Protocol methods can be optional or required.
- Required methods *must* be implemented.

- Optional methods *need not* be implemented. So, we always need to check whether an optional protocol method is implemented before sending the message.

- Protocols are similar to interfaces in other languages.

Why are protocols important?

- Protocols are used everywhere in Cocoa and CocoaTouch especially as part of the *delegate* design pattern.

- If some class agrees to implement a protocol, then other objects can communicate with this object without needing to know any other details about the object. This is a good example of *loose coupling*. Why is "loose coupling" a good OO design principle?

- Identifying objects just by their conformance to a protocol is a big deal in many design patterns.

Protocol Syntax

```

// Protocols can inherit from other protocols
@protocol MyProtocol<NSObject>
- (void)putYourMethodsHere;
@end

```

```

// Optional/required

@protocol AnotherProtocol<MyProtocol>
```

```objc
// @required is default
- (void)putYourMethodsHere;
// optional
@optional
- (void)optionalMethod;
// use @required to switch back
@required
- (NSString*)requiredAgain;
@end


```

```objc
// Conformance syntax

@interface MyClass:NSObject<AnotherProtocol>
// don't put the signatures in the header
@end

@implementation MyClass

// required
- (NSString*)requiredAgain {
    return @"Some result";
}

// required
- (void)putYourMethodsHere {
    // do stuff
}
@end

```

```objc
// Testing protocol conformance
```

```

int main() {
    BOOL result = [MyClass conformsToProtocol:@protocol(AnotherProtocol)];
    MyClass *myClass = [MyClass new];
    BOOL responds = [myClass respondsToSelector:@selector(optionalMethod)];
    if (result && responds) {
        // won't reach here
    }
}
```

Example Of Protocols & Polymorphism

```

#import <Foundation/Foundation.h>

// Flyable.h
@protocol Flyable <NSObject>
- (void)fly;
@end

// #import "Flyable.h"
// Duck.h
@interface Duck : NSObject<Flyable>
@end

// #import "Duck.h"
// Duck.m
@implementation Duck
- (void)fly {
```

```objc
    NSLog(@"flyin high!");
}
@end

// #import "Flyable.h"
// RubberDuck.h
@interface RubberDuck : NSObject<Flyable>
@end

// #import "RubberDuck.h"
// RebberDuck.m
@implementation RubberDuck
- (void)fly {
    NSLog(@"can't fly worth beans");
}
@end

// main.m

// C function that takes a Flyable parameter
void executeFlyableObject(id<Flyable>obj) {
    [obj fly];
}

int main(int argc, const char * argv[]) {

    id<Flyable>bird1 = [Duck new];
    id<Flyable>bird2 = [RubberDuck new];

    NSArray *arr = @[bird1, bird2];

    for (id<Flyable>item in arr) {
        if ([item isMemberOfClass:[Duck class]]) {
            [item fly]; // ==> flyin high!
        }
    }
    executeFlyableObject(bird2); // ==> can't fly worth beans
    return 0;
}

```

Simple Delegation Example

```
// Basic Delegation Example Showing How To Get Another Object To Do Work For A Class
// This allows

#import <Foundation/Foundation.h>

// Protocol
@protocol PlayerDelegate <NSObject>
- (void)play;
@end

// Apple Service
@interface AppleMusicService : NSObject<PlayerDelegate>
@end

@implementation AppleMusicService
- (void)play {
    NSLog(@"playing apple music playlist");
}
```

```objc
@end

// Spotify Service
@interface SpotifyService : NSObject<PlayerDelegate>
@end

@implementation SpotifyService
- (void)play {
    NSLog(@"playing spotify playlist");
}
@end

// Player
@interface Player : NSObject
@property id<PlayerDelegate>delegate;
- (instancetype)initWithMusicService:(id<PlayerDelegate>)service;
- (void)play;
- (void)changeServiceTo:(id<PlayerDelegate>)service;
@end

@implementation Player

- (instancetype)initWithMusicService:(id<PlayerDelegate>)service {
    if (self = [super init]) {
        _delegate = service;
    }
    return self;
}

- (instancetype)init {
    NSAssert(NO, @"Use designated initializer instead");
    return nil;
}

- (void)play {
    [self.delegate play];
}

- (void)changeServiceTo:(id<PlayerDelegate>)service {
    if ([service isMemberOfClass:[self.delegate class]]) {
        return;
    }
    self.delegate = service;
}

@end

// Main
int main() {
    AppleMusicService *appleMusic = [AppleMusicService new];
    SpotifyService *spotify = [SpotifyService new];
    Player *player = [[Player alloc] initWithMusicService:appleMusic];
    [player play];
    [player changeServiceTo:spotify];
    [player play];

    return 0;
```

```
}

```

Simple Delegate Callback Example

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

// Master.h
@interface Master : NSObject
- (void)fakeButtonTap;
@end

// #import "Master.h"
// Master.m

// class extension, not really used execept for conforming to DetailDelegate
@interface Master()<DetailDelegate>
@end

@implementation Master
- (void)fakeButtonTap {
    Detail *detail = [Detail new];
    detail.delegate = self;
    [detail saveFakeUserInput:@"some input"];
}

- (void)doStuffForDetailWithData:(NSString *)data {
    NSLog(@"%s data: %@", __PRETTY_FUNCTION__, data);
}
```

```objc
@end

// Detail.h
@protocol DetailDelegate <NSObject>
- (void)doStuffForDetailWithData:(NSString *)data;
@end
@interface Detail : NSObject

@property (nonatomic, weak) id<DetailDelegate>delegate;
- (void)saveFakeUserInput:(NSString *)input;

@end

// #import "Detail.h"
// Detail.m
@implementation Detail
- (void)saveFakeUserInput:(NSString *)input {
    [self.delegate doStuffForDetailWithData:input];
}
@end

// main.m
int main() {
    Master *m = [Master new];
    [m fakeButtonTap];
    return 0;
}

```

Delegation in CocoaTouch

- ApplicationDelegate is the class that the framework sets up in main.m.
- The UIApplication object uses the AppDelegate to call for customization information, or to give your app a chance to respond to system events.

Working with protocols

NSNumber

- Light weight wrapper around primitive integer types.
- Most often used to include number values in collections in Objective-C.
- For instance, to include integers in an NSArray convert to NSNumber

```
// 3 different ways to instantiate
// prefer literal instantiation

NSNumber *num1 = [[NSNumber alloc] initWithInt:22];
NSNumber *num2 = [NSNumber numberWithFloat:12.2];
NSNumber *num3 = @(33);
NSNumber *num4 = @(YES); // BOOL
NSNumber *num5 = @('i'); // Char
NSArray *arr = @[num1, num2, num3, num4, num5];

```

- You may need to unbox NSNumbers. Do it like this:

```

NSInteger unwrappedNum1 = [arr[0] intValue];
NSLog(@"%lu", unwrappedNum1);
float unwrappedNum2 = [arr[1] floatValue];
NSLog(@"%f", unwrappedNum2);
NSInteger unwrappedNum3 = [arr[2] intValue];
NSLog(@"%lu", unwrappedNum3);
BOOL val = [arr[3] boolValue];
NSLog(@"%@", val ? @"YES": @"NO");

// char: What will these logs print?
NSLog(@"char value boxed %@", num[4]);
NSLog@"char value unboxed: %c", [num[4] charValue]);

```

- Some Tricks

```

NSDictionary *dict = @{@1:@"One", @2:@"Two", @3:@"Three"};

for (NSNumber *key in dict.allKeys) {
    NSLog(@"%@", dict[key]);
}

NSInteger num5 = 44;
NSLog(@"logging an NSInteger by wrapping it: %@", @(num5));

```

```
NSString *num5ToString = @(num5).stringValue;

num5ToString = [NSString stringWithFormat:@"%d", 44];

```

- Comparing NSNumbers

```
1
2 // Question: What will the statement at line 7 log out and why?
3
4 NSNumber *num7 = @(22);
5 NSNumber *num8 = [NSNumber numberWithInteger:22];
6 BOOL value2 = num7 == num8;
7 NSLog(@"%@ is equal to %@: %@", num7, num8, value2 ? @"YES" : @"NO");
8
9
```

```
int main() {
    NSNumber *num7 = @(22);
    NSNumber *num8 = [NSNumber numberWithInteger:22];

    if ([num7 intValue] == [num8 intValue]) {
        NSLog(@"they're equal");
    }

    if ([num7 isEqualToNumber:num8]) {
        NSLog(@"they're equal yo");
    }

    NSComparisonResult comparisonResult = [num7 compare:num8];
    if (comparisonResult == NSOrderedAscending) {
        NSLog(@"ascending");
    } else if (comparisonResult == NSOrderedSame) {
        NSLog(@"same");
    } else if (comparisonResult == NSOrderedDescending) {
        NSLog(@"descending");
    }

    return 0;
}
```

NSValue:

```

int main() {

    // Box C struct with NSValue

    /*
     typedef struct {
        int mark;
        char name[10];
        int average;
     } Student;
     */

```

```objc
    struct Student {
        int mark;
        char name[10];
        int average;
    };

    struct Student report1 = { 89, "James", 79 };
    struct Student report2 = { 77, "Sonya", 70 };

    NSValue *reportValue1 = [NSValue value:&report1 withObjCType:@encode(struct Student)];

    NSValue *reportValue2 = [NSValue value:&report2 withObjCType:@encode(struct Student)];

    NSArray *arr = @[reportValue1, reportValue2];

    struct Student result1;
    [arr[0] getValue:&result1];

    NSLog(@"%@", @(result1.average));

    // Box CGRect with NSValue

    CGRect rect1 = CGRectMake(0.0, 0.0, 200.0, 200.0);
    CGRect rect2 = CGRectMake(100.0, 0.0, 200.0, 200.0);
    NSValue *rect1Box = [NSValue valueWithRect:rect1];
    NSValue *rect2Box = [NSValue valueWithRect:rect2];
    NSArray *rectArr = @[rect1Box, rect2Box];

    CGRect rect1Unboxed = [rectArr[0] rectValue];
    NSLog(@"rect1 unboxed: %@", NSStringFromRect(rect1Unboxed));
    CGRect rect2Unboxed = [rectArr[1] rectValue];
    NSLog(@"rect2 unboxed: %@", NSStringFromRect(rect2Unboxed));

    return 0;
}
```

- http://rypress.com/tutorials/objective-c/data-types/nsnumber
- https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSNumber_Class/
- https://developer.apple.com/library/mac/documentation/Cocoa/Reference/Foundation/Classes/NSValue_Class/

Objective-C Categories

What are categories:

- Called *Extensions* in Swift.

- Add functionality to existing classes without modifying original class.

- Can modify private system classes without subclassing.

- Used to break up complex classes into logical components.
- Allows flexibility of adding functionality as needed. For instance, I could add an extension to NSString but choose to only use it in some classes and not others.
- Can be used to expose functionality on otherwise private classes.

File Naming Convention

NameOfExtendedClass+NameOfExtension.h/.m

e.g.
NSString+Utilities.h/.m

- You need to import the category to get the functionality.

Syntax

- interface + implementation just like classes

```
@interface
@end

@implementation
@end
```

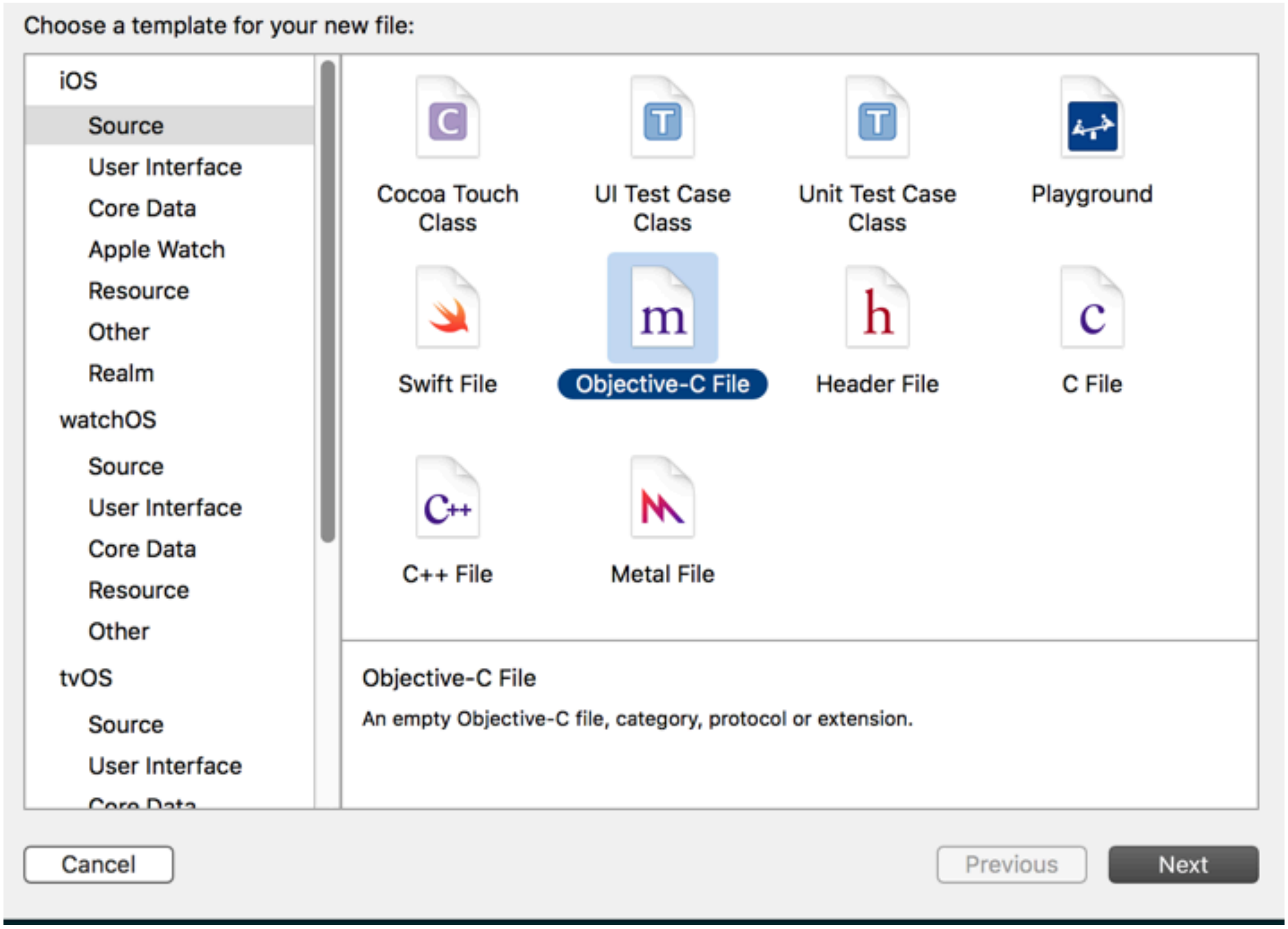
- But the syntax is a bit different than classes.

- Notice the round brackets, no superclass and the name of the category after the class being extended.

```
@interface NSString(Utils)
@end
@implementation NSString(Utils)
@end
```

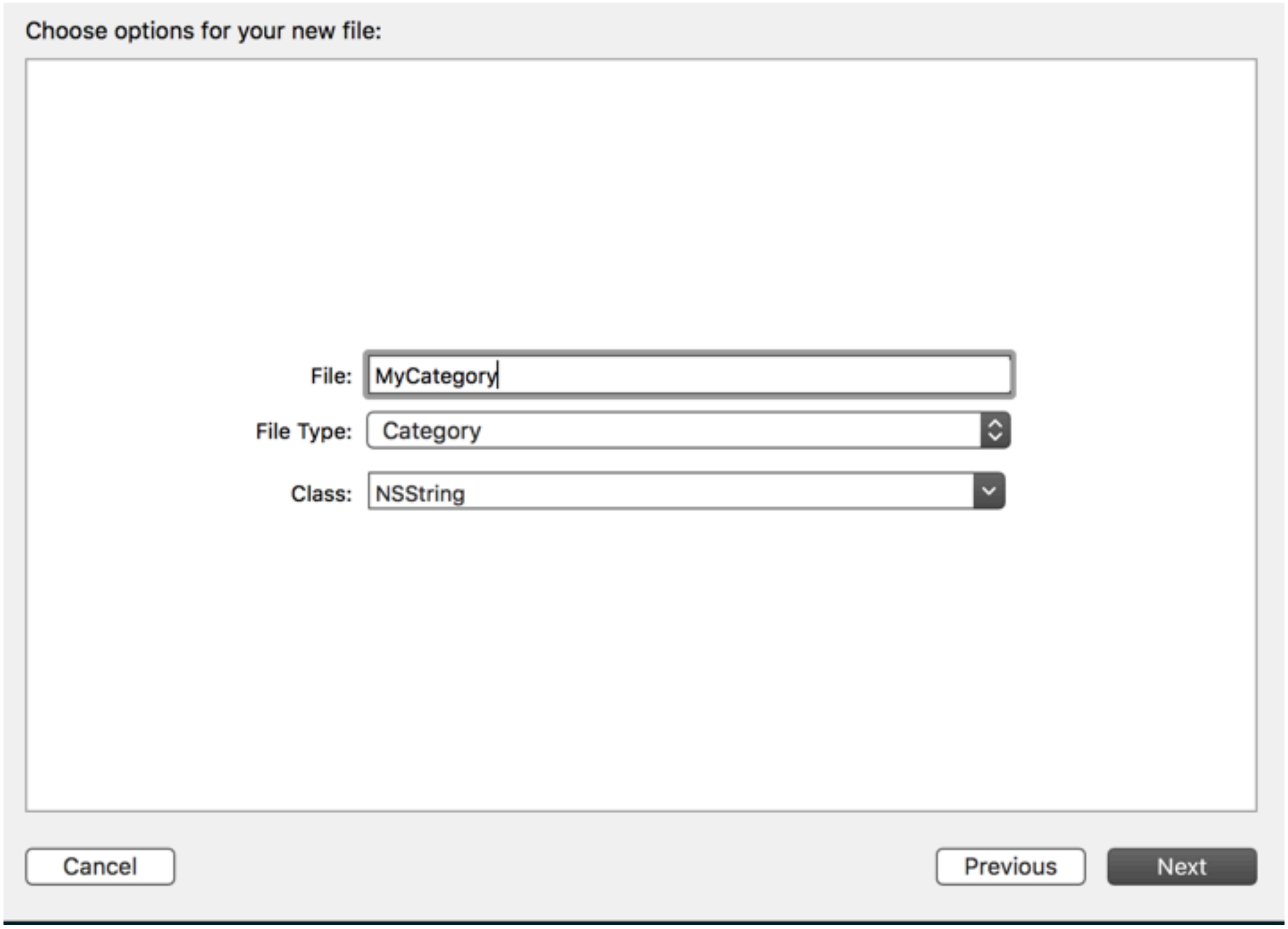
- Xcode will automatically create the files and stubs for you if you do this:

New File >> iOS Source >> Objective-C File >> Category



Objective-C File

An empty Objective-C file, category, protocol or extension.


Choose options for your new file:

File: MyCategory

File Type: Category

Class: NSString

- You refer to the object being acted on when you are inside the implementation as *self*.
- e.g.

```
@implementation NSString(Utils)
- (NSString *)addStar {
    // notice SELF here to represent the NSString instance that receives this message
    return [self stringByAppendingString:@"*"];
}
@end

int main() {
```

```objc
    // Here we call our extension on a string literal George
    NSString *imAStar = @"George";
    NSString *starredString = [imAStar addStar]; // ==> George*
}
```

```objc
// More advanced NSString Extension that returns the vowels on an NSString

#import <Foundation/Foundation.h>

@interface NSString (Vowelize)
- (NSString *)vowelize;
@end

```

```objc

#import "NSString+Vowels.h"

@implementation NSString (Vowelize)
- (NSString *)vowelize {
    NSMutableString *result = [NSString string];
    if (self.length == 0) {
        return result;
    }
     NSString *comparitor = @"aeiou";
    // loop through string
    for (NSInteger i = 0; i < self.length; ++i) {
        NSRange range = NSMakeRange(i, 1);
        NSString *subStr = [self substringWithRange:range];
        if ([comparitor localizedStandardContainsStr
ing:subStr]) {
            [result appendString:subStr];
        }
    }
    return  result;
}
@end

```

Objective C Class Extension

- Way to add another interface to your classes that are *not* visible to outside classes.
- They were more commonly used for methods in early versions of Objc where you had to forward declare all methods.
- Modern Objc uses Class Extensions for properties only.

```
// Simple example of class extension

@import Foundation; // Notice the modern importation syntax

@interface Person: NSObject
// Notice age is readonly
@property (nonatomic, readonly) NSInteger age;
- (instancetype)initWithName:(NSString *)name age:(NSInteger)age;
@end
```

```

```

```objc

#import "Person.h"

// class extension
@interface Person()
@property (nonatomic, readwrite) NSInteger age; // optional way of doing this, because you can write to age using _age privately
@property (nonatomic) NString *name;
@end

@implementation Person

- (instancetype)initWithName:(NSString *)name age:(NSInteger)age {
    if (self = [super init]) {
        _name = name;
        _age = age;
    }
    return self;
}

// calls designated initializer and passes in defaults
- (instancetype)init {
    return [self initWithName:nil age:0];
}

@end


```

General References:

Cocoa Core Competencies