



The Objective-C Runtime and You

Contact Info

Email: brandon.alexander@gmail.com

GTalk: brandon.alexander@gmail.com

Twitter: [@whilethis](https://twitter.com/whilethis)

Web: <http://www.whilethis.com>

What We'll Cover

- Brief tour of the Objective-C language
- NSObject
- Message Passing
- Key-Value Coding
- Key-Value Observing

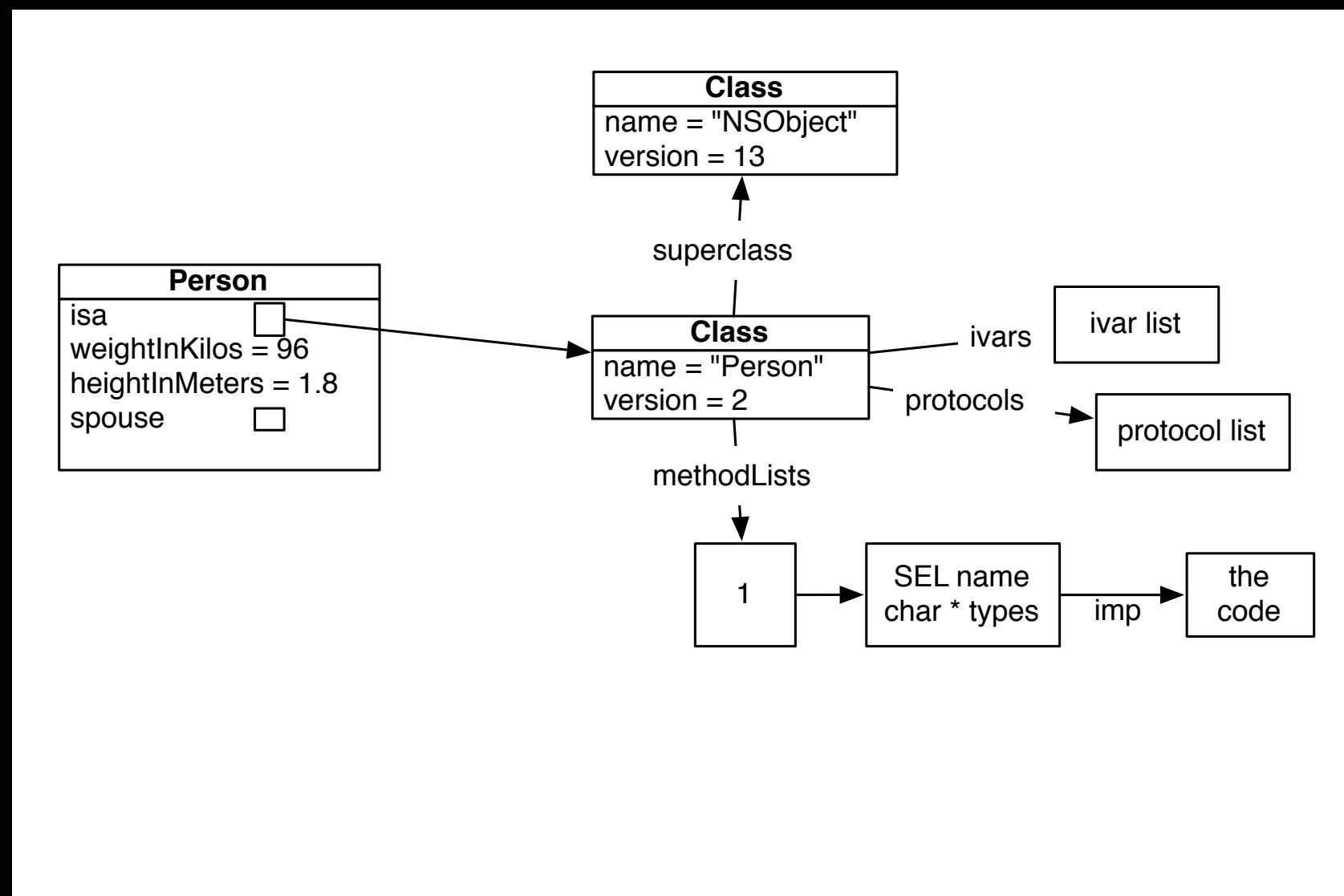
The Language

- Created by Tom Love and Brad Cox
- Strict superset of C
- Original compiler was a preprocessor
- Modern compiler is MUCH smarter

NSObject

- Base class for most objects (NSProxy is the other base class)
- Provides all the functionality that makes the runtime work

Anatomy of NSObject



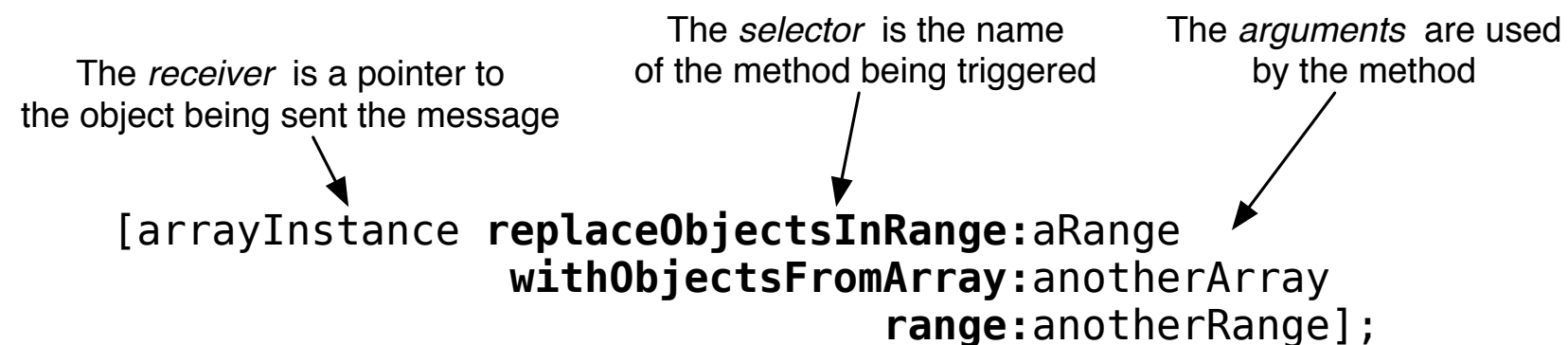
Objective-C Messages

The *receiver* is a pointer to the object being sent the message

The *selector* is the name of the method being triggered

The *arguments* are used by the method

```
[arrayInstance replaceObjectsInRange:aRange  
withObjectsFromArray:anotherArray  
range:anotherRange];
```

The diagram illustrates the structure of an Objective-C message. It features three explanatory labels at the top with arrows pointing to specific parts of a code snippet below. The first label, 'The receiver is a pointer to the object being sent the message', points to '[arrayInstance'. The second label, 'The selector is the name of the method being triggered', points to 'replaceObjectsInRange:aRange'. The third label, 'The arguments are used by the method', points to 'withObjectsFromArray:anotherArray range:anotherRange'. The code snippet is formatted with the selector in bold.

Objective-C Messages

The *receiver* is a pointer to the object being sent the message

The *selector* is the name of the method being triggered

The *arguments* are used by the method

```
[arrayInstance replaceObjectsInRange:aRange  
withObjectsFromArray:anotherArray  
range:anotherRange];
```

```
objc_msgSend(arrayInstance, @selector(aSel:), arg1);
```

Selectors

```
SEL aSel = @selector(foo);  
if ([someObj respondsToSelector:aSel]) {  
    [someObj performSelector:aSel];  
}
```

```
SEL bSel = NSSelectorFromString(@"foo2");  
IMP functionPointer = [someObj methodForSelector:bSel];  
(functionPointer *)(someObj, bSel);
```

Method Resolution

- + (BOOL)resolveClassMethod:(SEL)sel
- + (BOOL)resolveInstanceMethod:(SEL)sel

Message Forwarding

- (`id`) forwardingTargetForSelector: (`SEL`) s;
- (`void`) forwardInvocation: (`NSInvocation *`) i;

Message Forwarding

```
- (id)forwardingTargetForSelector:(SEL)aSel {  
    if (aSel == @selector(payBill:)) {  
        return spouse;  
    }  
    return [super forwardingTargetForSelector:aSel];  
}
```

Method Implementations

- Extra parameters
 - (id) self
 - (SEL) _cmd

Invocations

- `NSInvocation`
- Wraps message in a class
- Actual parameters start at index 2
 - `self` and `_cmd`
- Pass parameters by reference

NSInvocation

```
// Look up selector and method signature
SEL someSel = @selector(computePaymentForPrincipal:term:);
NSMethodSignature *ms = [loanCalc
methodSignatureForSelector:someSel];

// Create invocation with target and arguments
NSInvocation *msg = [NSInvocation
invocationWithMethodSignature:ms];
[msg setSelector:someSel];
[msg setTarget:loanCalc];
float principal = 200000;
float term = 30;
[msg setArgument:&principal atIndex:2]; // self is 0, _cmd is 1
[msg setArgument:&rate atIndex:3];

// Trigger it and get results
[msg invoke];
float result;
[msg getResult:&result];
NSLog(@"Your payment is %.2f", result);
```


Method Swizzling

- Replaces implementation of a selector
- Replacement happens at class level
 - Can have unintended consequences
 - If you think you need to swizzle, you probably don't

```
SwizzlerClass *swizzler = [[SwizzlerClass alloc] init];

NSLog(@"%@", [swizzler methodToSwizzle]);

Method methodToSwizzle = class_getInstanceMethod([SwizzlerClass class],
@selector(methodToSwizzle));

method_setImplementation(methodToSwizzle, (IMP) methodSwizzledIMP);

NSLog(@"%@", [swizzler methodToSwizzle]);

SwizzlerClass *anotherSwizzler = [[SwizzlerClass alloc] init];
NSLog(@"Another Swizzler: %@", [anotherSwizzler methodToSwizzle]);
```

More Swizzling

- Previous approach loses original Implementation
- Create a category
- Use `method_exchangeImplementations()`
- Can call original selector (fake inheritance)

To the code!!

Associated Objects

- Built-in object association table
- Great for categories
- objc_setAssociatedObject
- objc_getAssociatedObject
- objc_removeAssociatedObjects

Key Value Coding

- Conventions!!
- setValueForKey: and valueForKey:
 - Tries accessors first
 - Falls back to direct ivar access
- Nib loading on iOS uses KVC
- Key Value Coding Programming Guide

More KVC

- KVC can be performed for one to many relationships
 - Simple array
 - Trees
- Conform to a particular convention

KVC Example

```
ModelObject *model = [[ModelObject alloc] init];

NSMutableArray *array = [model mutableArrayValueForKey:@"superKey"];

//insertObject:inSuperKeyAtIndex:
[array insertObject:[NSNull null] atIndex:0];

//countOfSuperKey
NSLog(@"%i", [array count]);

//objectInSuperKeyAtIndex:
NSLog(@"%@", [array objectAtIndex:0]);

//removeObjectFromSuperKeyAtIndex:
[array removeObjectAtIndex:0];
```


To the code!!

Key Value Observing

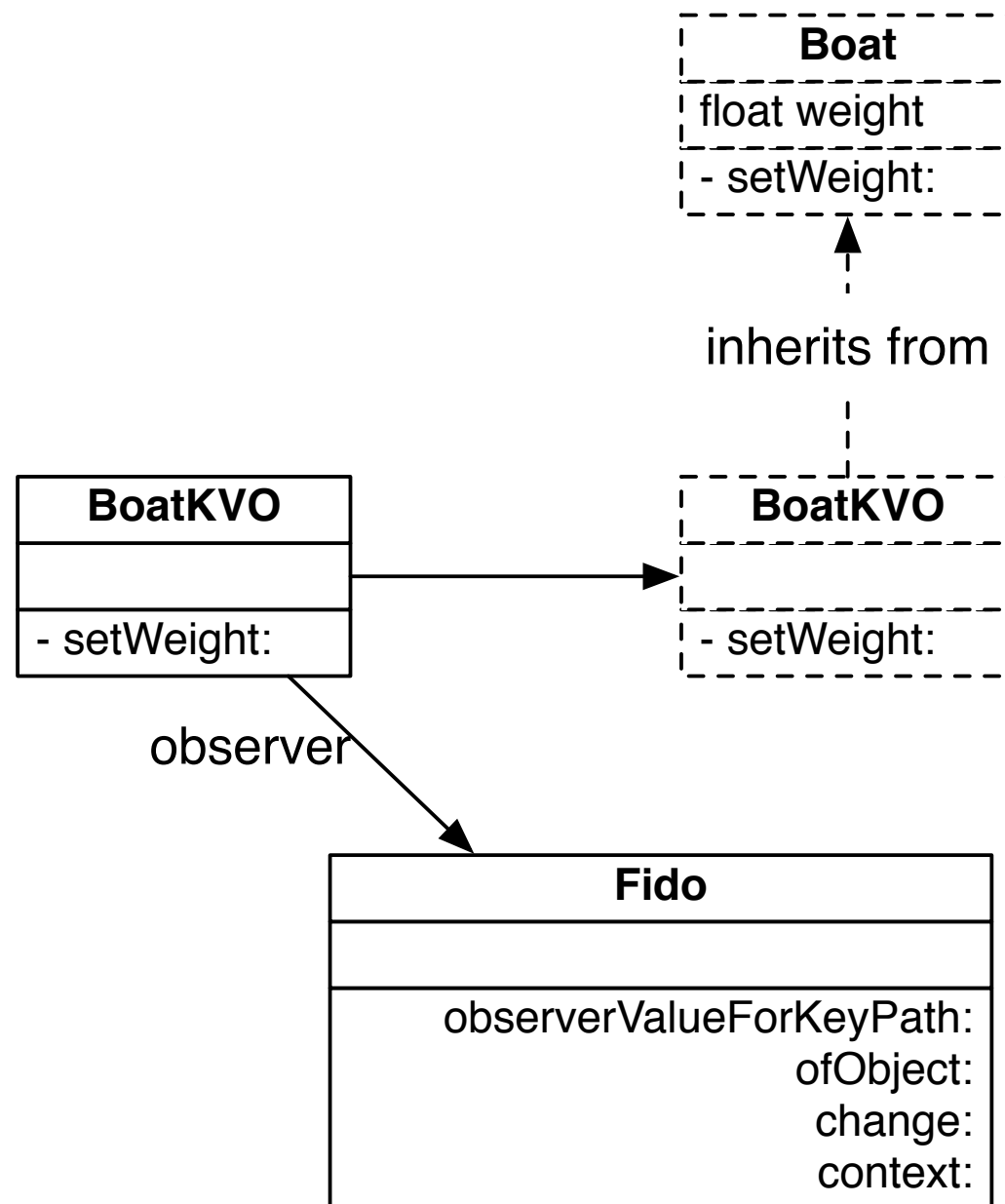
- Observer pattern built into NSObject
- Based off keys and keyPaths (remember KVC?)

KVO Implementation

```
[boat addObserver:self  
    forKeyPath:@"weight"  
    options:options  
    context:c];
```

```
- (void) observeValueForKeyPath:(NSString *)keyPath  
    ofObject:(id)object  
    change:(NSDictionary *)change  
    context:(void *)context {  
    //Code here...  
}
```

KVO Secrets Revealed



Questions

(or more code)

Contact Info

Email: brandon.alexander@gmail.com

GTalk: brandon.alexander@gmail.com

Twitter: [@whilethis](https://twitter.com/whilethis)

Web: <http://www.whilethis.com>