



Apps Incorporated

Category + Swizzling Design Pattern

John Wallace
Apps Incorporated



Apps Incorporated

Overview

- **Category** - ObjC language feature that adds methods into any existing class.
- **Swizzling** - Technique which replaces the implementation of one method with another
- **Category + Swizzling Pattern** - By combining categories and swizzling we can dynamically override the behavior of any method in any class, even if we don't have the source to that class



Apps Incorporated

Why do we care?

- Can override default behaviors of methods in NSObject, UIView, and any other class.
- Behavior inherited by built-in classes and our classes.
- Examples found in Apps Incorporated Toolkit:
 - Add Java-Swing style layouts by overriding UIView's `layoutSubviews`
 - Add object caching by overriding `release`
 - Add dealloc observing by overriding `dealloc`



Apps Incorporated

Why not just subclass?

- Subclasses start a new class subtree; doesn't efficiently solve problem for class trees with more than one level



Apps Incorporated

Why not just subclass?

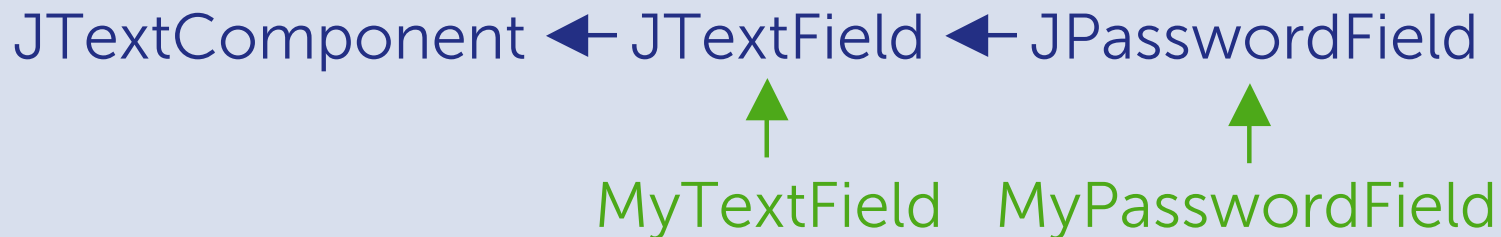
- Subclasses start a new class subtree; doesn't efficiently solve problem for class trees with more than one level

JTextComponent ← JTextField ← JPasswordField



Why not just subclass?

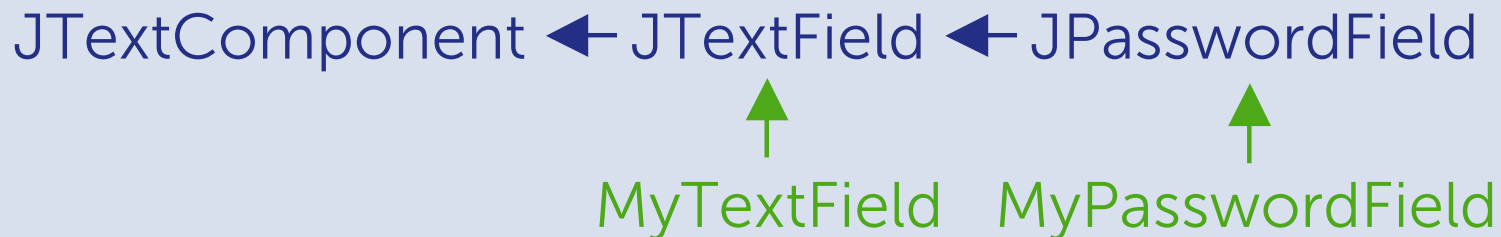
- Subclasses start a new class subtree; doesn't efficiently solve problem for class trees with more than one level





Why not just subclass?

- Subclasses start a new class subtree; doesn't efficiently solve problem for class trees with more than one level



- Pattern in Java is to make subclasses + a common helper class; not as easy as categories+swizzling



Apps Incorporated

Objective C

- Official language of Mac OS X and iOS
- Developed in 1980s by adding SmallTalk method syntax to c
- Objects' instance variables referenced at an offset computed at compile time like a c structure or c++ object or java object
- Objects' methods are resolved at run time by looking up their selector codes from method dispatch tables



Apps Incorporated

ObjC - Hello World

```
Shouter *shouter =  
    [[Shouter alloc] initWithMessage:@"Hello World"];  
[shouter shout];  
[shouter release];
```



Apps Incorporated

ObjC - Hello World

```
Shouter *shouter =  
    [[Shouter alloc]  
     initWithMessage:  
     @"Hello World"];  
[shouter shout];  
[shouter release];
```

```
@interface Shouter : NSObject {  
    NSString *_message;  
}  
- (id)initWithMessage:(NSString *)message;  
- (void)shout;  
@end
```

```
@implementation Shouter  
- (id)initWithMessage:(NSString *)message {  
    if (self = [super init]) {  
        _message = [message retain];  
    }  
    return self;  
}  
- (void)shout {  
    NSLog(@"%@", _message);  
}  
@end
```



Apps Incorporated

Categories

- Add methods to an existing class
- Cannot add instance variables
- Used extensively by Apple in iOS and Mac OS X
- Allow behavior for a class to be in separate files

```
@interface NSObject(APFoo)
- (void)foo;
@end

@implementation NSObject(APFoo)
-(void)foo {
    NSLog(@"FOO!");
}
@end
```



Apps Incorporated

Swizzling

- Exchanges implementation of one method with the implementation of another method

```
+ (void)swizzleMethod:(SEL)origSel withMethod:(SEL)newSel {  
    Method origMethod = class_getInstanceMethod(self, origSel);  
    Method newMethod = class_getInstanceMethod(self, newSel);  
    method_exchangeImplementations(origMethod, newMethod);  
}
```

- `[+swizzleMethod:withMethod:]` implemented in Apps Incorporated Toolkit
- `class_getInstanceMethod()` and `method_exchangeImplementations()` implemented in ObjC runtime



Apps Incorporated

+load method

- Called once for each class and category as they are loaded; called for classes first
- Entry point we use to swizzle methods of a category with methods of its class

```
@implementation NSObject(DeallocLog)
+ (void)load {
    [[NSObject class] swizzleMethod:@selector(dealloc)
                        withMethod:@selector(dl_dealloc)];
}
... other methods ...
@end
```



Example

- Category that prints out an object's class name before it is deallocated

```
@implementation NSObject(DeallocLog)
+ (void)load {
    [[NSObject class] swizzleMethod:@selector(dealloc)
                          withMethod:@selector(dl_dealloc)];
}
- (void)dl_dealloc {
    NSLog(@"deallocing %@", [self class]);
    [self dealloc]; // calls original [-dealloc] method
}
@end
```



Apps Incorporated

Example (cont.)

- Executing

```
[[[NSObject alloc] init] release];  
[[[Foo alloc] init] release];  
[[[Bar alloc] init] release];
```

- Prints out to console

```
2010-10-09 04:08:46 DeallocLog[6853:a0f] deallocating NSObject  
2010-10-09 04:08:46 DeallocLog[6853:a0f] deallocating Foo  
2010-10-09 04:08:46 DeallocLog[6853:a0f] deallocating Bar
```



Apps Incorporated

Code Path Before Swizzling

NSObject class

```
...  
@selector(dealloc) ●  
...  
@selector(dl_dealloc) ●  
...
```

Whatever code is executed for the dealloc method as implemented by Apple

```
NSLog(@"deallocing %@", [self class]);  
[self dl_dealloc];
```




Apps Incorporated

Code Path Before Swizzling

NSObject class

```
...  
@selector(dealloc) ...  
...  
@selector(dl_dealloc) ...  
...
```

Whatever code is executed for the dealloc method as implemented by Apple

```
NSLog(@"deallocing %@", [self class]);  
[self dl_dealloc];
```

Infinite recursion!



Apps Incorporated

Code Path After Swizzling

NSObject class

```
...  
@selector(dealloc) ...  
...  
@selector(dl_dealloc) ...  
...
```



Whatever code is executed for the dealloc method as implemented by Apple

```
NSLog(@"deallocing %@", [self class]);  
[self dl_dealloc];
```



Apps Incorporated

Code Path After Swizzling

NSObject class

```
...  
@selector(dealloc)  
...  
@selector(dl_dealloc)  
...
```

Whatever code is executed for the
dealloc method as implemented by
Apple

```
NSLog(@"deallocing %@", [self class]);  
[self dl_dealloc];
```



Apps Incorporated

Conclusion

- Categories+Swizzling Pattern implements behavior that isn't possible in some popular languages
- Overrides behavior of existing classes using supported language features (categories, swizzling, `+load` method)
- Overridden behavior is inherited by subclasses
- Reduces complexity for adding/using new behavior
- Used in Apps Incorporated Toolkit to extend the behavior of closed source classes such as `NSObject`, `UIView`, etc.



Apps Incorporated

Apps Incorporated Toolkit

- Toolkit that extends Cocoa and Cocoa touch
- APFoundation, APUIKit, APAppKit, etc.
- Apache license for open source portion
- Proprietary license for closed source portion
- Available from <http://toolkit.appsincorporated.com>
- Will be released in pieces with corresponding blogs at <http://blog.appsincorporated.com>



Apps Incorporated

Downloads

- Slides and code from this talk:
[http://www.appsincorporated.com/talks/categoryS
wizzling.v1.zip](http://www.appsincorporated.com/talks/categoryS
wizzling.v1.zip)



Apps Incorporated

Questions