

iOS Development Workshop @ Cohab

Brent Nycum
brentnycum@gmail.com

- Preferences
- Memory Management
- Tab Bar Controller
- Split View Controller

Preferences

Preferences

- Will be using the NSUserDefaults class
- Allows an app to customize its behavior to match a user's preference
 - Store your users unit of measure (°F or °C)

NSUserDefaults

- Provides convenience method for storing and accessing common types
 - floats
 - doubles
 - integers
 - Booleans
 - Strings
 - URLs

NSUserDefaults

- Do not alloc / init
- Use the static standardUserDefaults method
 - `NSUserDefaults *defaults = [NSUserDefaults standardUserDefaults];`

NSUserDefaults

- Class behaves much like an NSMutableDictionary would
 - `[defaults setBool:YES forKey:@"makeMoney"];`
 - `[defaults boolForKey:@"makeMoney"];`

NSUserDefaults

- Lastly to save changes use synchronize
 - `[defaults synchronize];`

Memory Management

Memory Management

- Fundamental Rule
- You only release or autorelease objects that you own

Memory Management

- You take ownership of an object if you create it using a method that begins with:
 - `alloc`
 - `new`
 - `copy`
 - `mutableCopy`
- Or if you send it a retain message

Memory Management

- An object may have one or more owners
- When an object has no owners, it is destroyed
- To make sure an object you're interested in is not destroyed, you must become an owner
- To allow an object in which you're no longer interested to be destroyed, you relinquish ownership

Memory Management

- Use release or autorelease to relinquish ownership of an object

Memory Management

- autorelease just means send a release message sometime in the future
- Normally when the autorelease pool receives a drain message
- By sending an object an autorelease message, you declare that you don't want to own the object beyond the scope in which you sent the message

Memory Management

- The autorelease method makes it easy for you to return an object from a method and still abide by the ownership policy

Memory Management

- As a corollary of the fundamental rule, if you need to store a received object as a property in an instance variable, you must retain or copy it.

Memory Management

- A received object is normally guaranteed to remain valid within the method it was received in, and that method may also safely return the object to its invoker

Memory Management

- If it's a local variable, release it as soon as you're done using it
- If it's a local variable that you are returning, autorelease it

Memory Management

- Watch out when you set a variable multiple times in code
- Assigning a new value to an instance variable without releasing old value

Memory Management

- Use immutable versions
- Simpler and less error-prone than allocating a mutable object, modifying it, then releasing it

Memory Management

```
– (NSArray *)sprockets {  
    NSArray *array = [[NSArray alloc] initWithObjects: mainSprocket,  
    auxiliarySprocket, nil];  
  
    return array;  
  
}
```

Wrong! Results in a memory leak

Memory Management

- The object's reference to the new array object is limited to the sprockets method
- After the method returns, the object loses its reference to the new object so cannot relinquish ownership
- The caller would therefore not relinquish ownership of the returned object, leading to a memory leak

Memory Management

```
– (NSArray *)sprockets {  
    NSArray *array = [[NSArray alloc] initWithObjects: mainSprocket,  
    auxiliarySprocket, nil];  
  
    [array release];  
  
    return array; // array is invalid here  
}
```

Wrong! Object released too early

Memory Management

```
– (NSArray *)sprockets {  
    NSArray *array = [NSArray arrayWithObjects: mainSprocket,  
    auxiliarySprocket, nil];  
  
    return array;  
}
```

Correct!

Memory Management

- You don't own the array returned from `arrayWithObjects:`, you are therefore not responsible for relinquishing ownership.
- You can, though, safely return it from the `sprockets` method.

Memory Management

```
-(NSString *)tenThousandGs {  
    NSString *s = @"";  
  
    for (int i = 0; i < 10000; i++) {  
        s = [s stringByAppendingString:@"G"];  
    }  
  
    return s;  
}
```

Bad

Memory Management

```
-(NSString *)tenThousandGs {  
    NSMutableString *s = [[NSMutableString alloc] init];  
  
    for (int i = 0; i < 10000; i++) {  
        [s appendString:@"G"];  
    }  
  
    return [s autorelease];  
}
```

Better

Memory Management

```
-(NSString *)tenThousandGs {  
    NSMutableString *s = [NSMutableString string];  
  
    for (int i = 0; i < 10000; i++) {  
        [s appendString:@"G"];  
    }  
  
    return s;  
}
```

Best

Tab Bar Controller

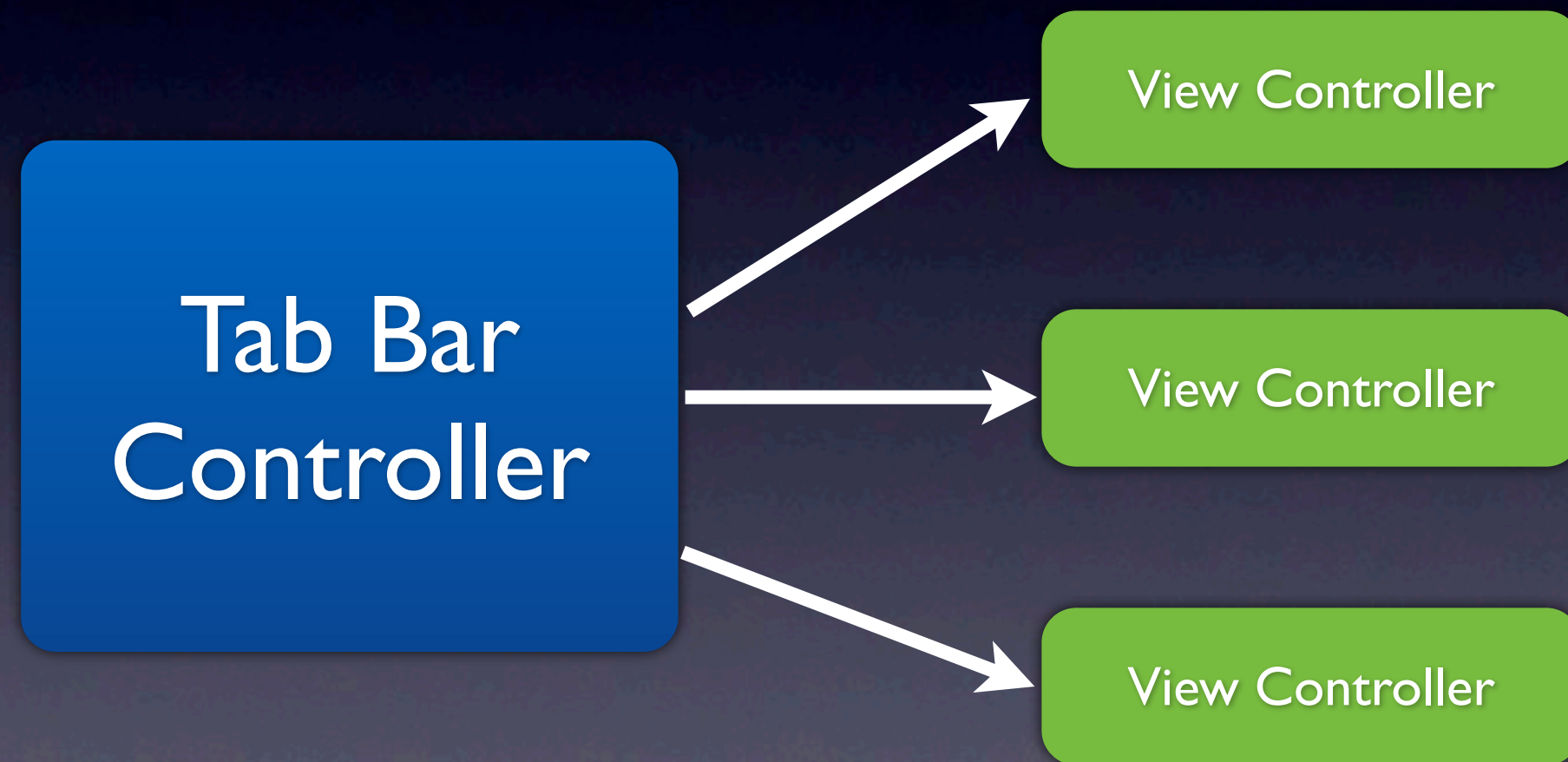
Tab Bar Controller



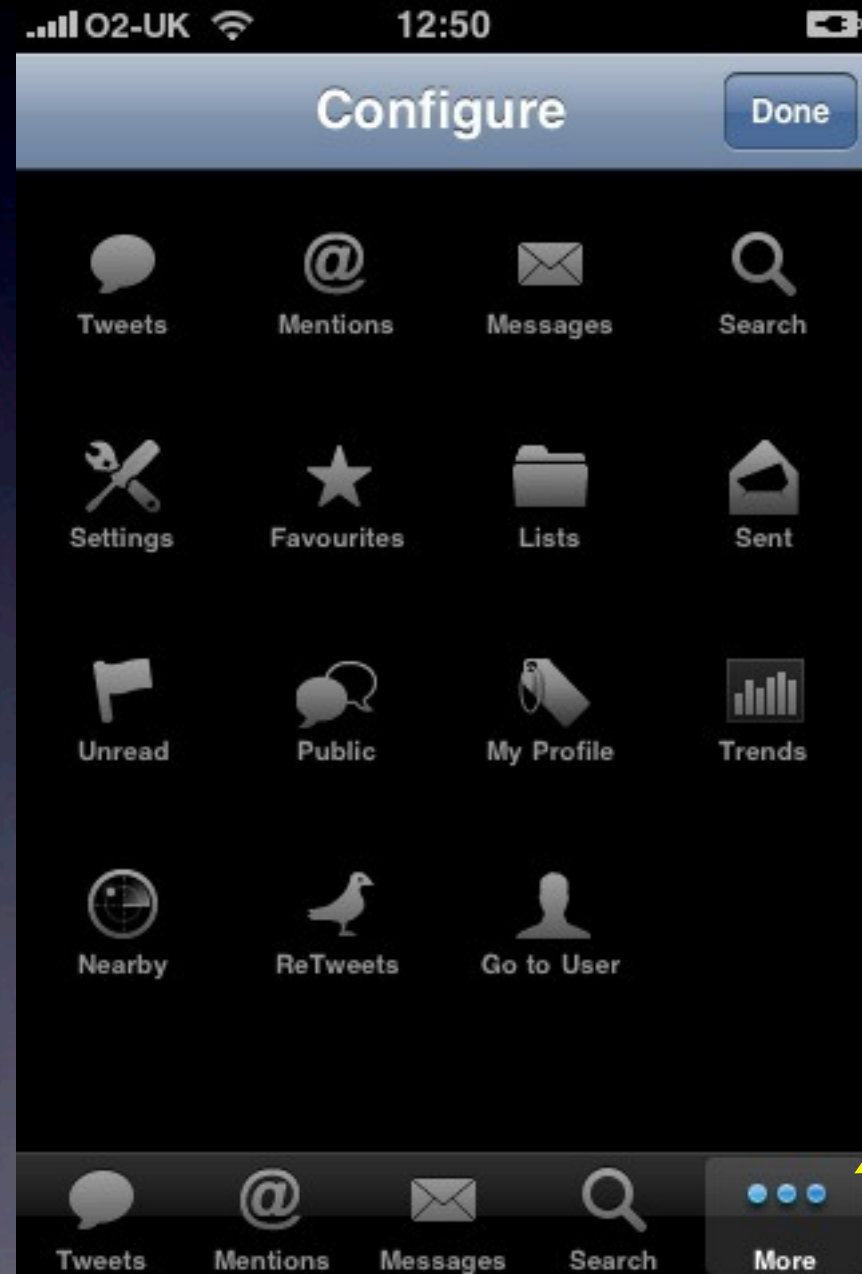
Tab Bar Controller



Tab Bar Controller

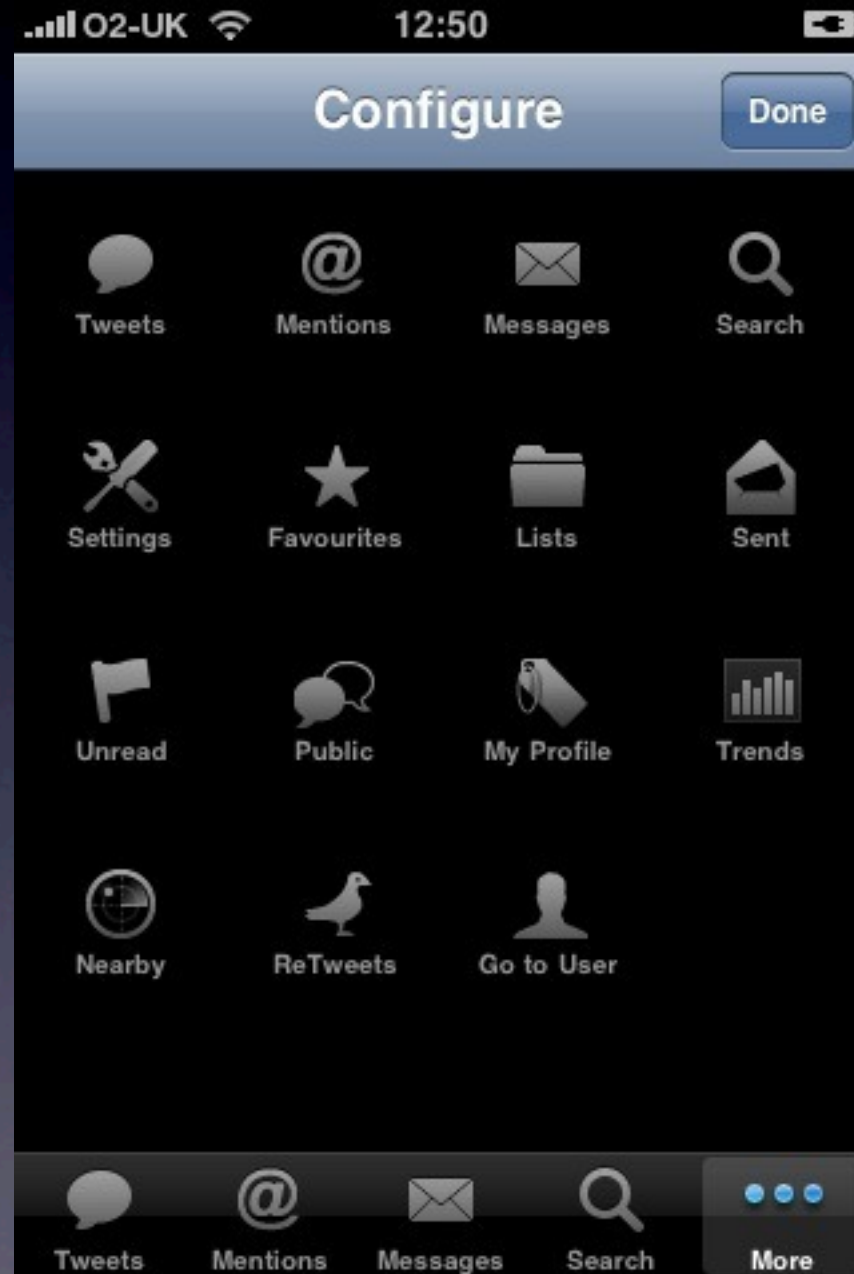


Tab Bar Controller



If more than 5 View Controllers

Tab Bar Controller



UI to allow the user
to edit which
buttons appear

More Button

Tab Bar Controller



↑ ↑ ↑ ↑
Tab Bar Items

Tab Bar Controller

- Tab Bar Items
 - `tabBarItem`
 - Image
 - Title



Tab Bar Controller

- Tab Bar Items can be badged
- `tabBarItem.badgeValue = @"5";` 

Split View Controller

Split View Controller



Split View Controller

- Instead of talking about it, lets dig into some code

Questions?
brentnycum@gmail.com