iPhone SDK For Java Developers

Rakesh Vidyadharan
rakesh@sptci.com

2009-02-17

iPhone SDK

- Uses Objective-C as the base programming language. Can mix C/C++ code as necessary.
- Similar to the Cocoa framework used to build Mac OS X desktop applications.
- Entirely MVC based.
- Requires an Intel based Mac[1].
- Well documented with sample applications.
- Xcode IDE and Interface Builder.
- Simulator for ease of development and testing.

[1] People have got it working on PPC machines.

Language Fundementals

- Smalltalk features added to C.
- Objects follow different syntax.

```
[obj myMethod:param1 paramTwoName:param2];
```

- True dynamic binding.
- Weakly typed language.
- Reference count based memory management.
  - `alloc` message increments refcount.
  - `copy` message increments refcount.
  - `retain` message increments refcount.
  - `release` message decrements refcount.

Memory Management

- Keep `retain` and `release` in the same code block.

```
NSString *str = [[NSString alloc] init];
self.string = str;
[str release];
```

- Use `autorelease` for objects returned from methods.

```
NSString *str = [[NSString alloc] init];
return [str autorelease];
```

- Ensure all fields are released in the `dealloc` method.

```
[field1 release];
[field2 release];
[super dealloc];
```

- Decorators more common than inheritance.
- Class semantics
  - Have to explicitly extend NSObject.
  - All fields are protected by default.
  - Methods declared in header files are always public.
- No garbage collector (on iPhone).
- No method over-loading. Parameter handles are part of method signature.
- Pointer not reference based.
- Methods are invoked via messages.
- Categories allow adding behaviour to objects.
- Accessor methods can be synthesised.
- No namespaces.

Unit Testing

- OCUnit framework included with Xcode.
- Based on SUnit, Kent Beck's Smalltalk unit testing framework from which JUnit is also derived.
- Similar to JUnit 3.x, not as elegant as JUnit 4.x

Unit Testing

- OCUnit framework included with Xcode.
- Based on SUnit, Kent Beck's Smalltalk unit testing framework from which JUnit is also derived.
- Similar to JUnit 3.x, not as elegant as JUnit 4.x

```
@interface MyClassTest : SenTestCase

@implementation MyClassTest
- (void) setUp { [super setUp]; }
- (void testMethod {}
- (void) tearDown { [super tearDown]; }
```

Protocol for encrypter

```
@protocol Encrypter
- (id) initWithKey:(NSString *) secret;
- (NSData *) encrypt:(NSString *) value;
- (NSString *) decrypt:(NSData *) value;
@end
```

Conforms to Encrypter

```
#import "Encrypter.h"

@interface XOREncrypter : NSObject <Encrypter>
{
  NSData *key;
}
@end
```

Implementation Class

```
#import "XOREncrypter.h"

@interface XOREncrypter (PrivateMethods)
- (NSData *) xor:(NSData *) data;
@end

@implementation XOREncrypter
- (id) init {...}
- (id) initWithKey:(NSString *) secret {...}
- (void) dealloc {...}
- (NSData *) encrypt:(NSString *) value {...}
- (NSString *) decrypt:(NSData *) value {...}
- (NSData *) xor:(NSData *) data {...}
@end
```

Auto Release Pools

- Auto release pools are used to manage `autoreleased` objects.
- Auto release pool is automatically added for each event loop for the main UI thread.
- You must maintain your own pool if you perform work in background threads.

Auto Release Pools

- Auto release pools are used to manage `autoreleased` objects.
- Auto release pool is automatically added for each event loop for the main UI thread.
- You must maintain your own pool if you perform work in background threads.

```
NSAutoreleasePool *pool =
    [[NSAutoreleasePool alloc] init];
NSString *str = [NSString
    stringWithFormat:@"%@-%@",
    @"One", @"Two"];
[pool release];
```

Threads

- Primary support through NSOperation and NSThread classes.
- NSTimer presents an elegant means of performing intensive operations in the background and update the UI in a thread-safe manner.
- NSObject `performSelectorInBackground:withObject:` and `performSelectorOnMainThread:withObject: waitUntilDone:` combination also provides higher level access to threads and task queues.

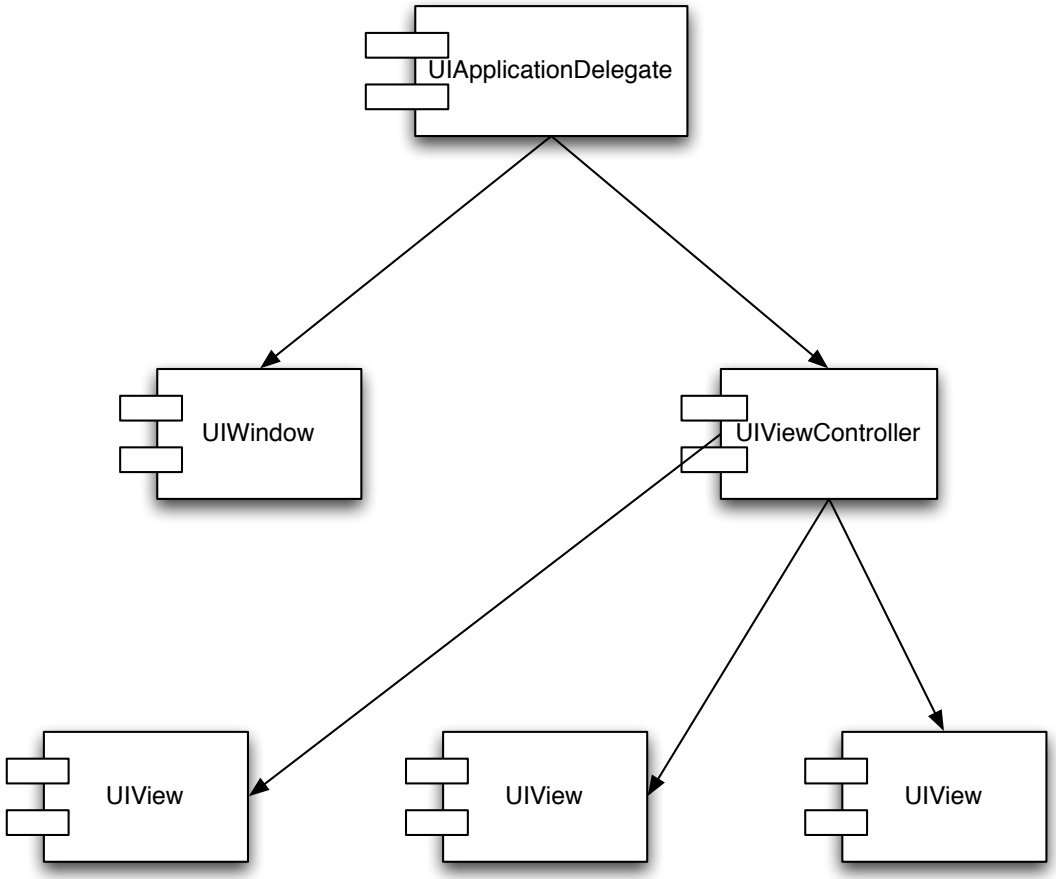
NSTimer Sample

```
@implementation TimerSample : NSObject
- (void) driver
{
   [NSTimer scheduledTimerWithTimeInterval:0.1
        target:self selector:@selector( task: )
        userInfo:nil repeats:NO];

}
- (void) task:(NSTimer *) timer
{
   // Perform background job and update UI
}
@end
```

NSObject performSelector Sample

```
- (void) driver {
    [self performSelectorInBackground:
        @selector( task: ) withObject:@"Test"];
}
- (void) task:(id) str {
    // Set up autorelease pool and work
    [self performSelectorOnMainThread:@selector(one:)
        withObject:myView waitUntilDone:NO];

}
-(void) one:(id) myView {
    // Perform UI updates.
}
```

Anatomy of Application



UIApplicationDelegate

- The delegate that is used to maintain application wide state.
- Used to display the primary view for the application.
- Provides the callback methods for application life-cycle.
  - `applicationDidFinishLaunching:` - Use to build the application UI.
  - `applicationWillResignActive:` - Use to handle application interruption by incoming phone call.
  - `applicationWillTerminate:` - Use to handle application termination (home button, exit button, ...).

UIView Components

- UIWindow represents the application window on screen.
- UIView components are added to UIWindow to build the interface.
- May be hand-coded or laid out using IB.
  - Easier to layout using IB.
  - Generates XML descriptor for loading pre-compiled and configured UIView instances.
  - No code is generated.
  - Implement `viewDidLoad` method in view controller to perform additional configuration of view.

UIViewController Classes

- Handler for all events generated from UIView.
- Custom UIView components are usually initialised through their associated controller.
- Usually the easiest way to access UIView components.
- IB can be used to assciate fields (properties) with the appropriate UIView components.

Creating a Project

- Launch Xcode
- Choose File− >New Project
- Select iPhone OS− >Application
- Select View-Based Application
- Save project to preferred location.

Files of Interest

- Look under the Classes folder.
- Application delegate as <appname>AppDelegate
- View controller as <appname>ViewController
- IB file <appname>ViewController.xib under Resources folder.

Files of Interest

- Look under the Classes folder.
- Application delegate as <appname>AppDelegate
- View controller as <appname>ViewController
- IB file <appname>ViewController.xib under Resources folder.

Modify ViewController

- Add button and label fields to header file.
- Add a listener method to the header file.
- Implement the listener in the implementation file.

View Controller Header

```
@interface CJUGViewController : UIViewController
{
  UIButton *button;
  UILabel *label;
  int count;
}
@property (nonatomic, retain) IBOutlet
    UIButton *button;
@property (nonatomic, retain) IBOutlet
    UILabel *label;
- (IBAction) listener:(id) sender;
- (IBAction) textListener:(id) sender;
@end
```

View Controller Implementation

```
@implementation CJUGViewController
@synthesize button;
@synthesize label;
- (IBAction) listener:(id) sender
{
   [label setText:[NSString stringWithFormat:
       @"Button clicked: %d", count++]];
}
- (void)dealloc
{
   [label release]; [button release];
   [super dealloc];
}
@end
```

Add Views

- Double-click <appname>ViewController.xib to launch IB.
- Select Objects− >Inputs & Values in Library window.
- DnD Round Rect Button to the View window.
- Dnd Label to the View window.

Add Views

- Double-click <appname>ViewController.xib to launch IB.
- Select Objects− >Inputs & Values in Library window.
- DnD Round Rect Button to the View window.
- Dnd Label to the View window.

Connect views to controller

- CTRL− >Drag from File's Owner to component.
- Select the field corresponding to type of component.
- CTRL− >Drag from button to File's Owner.
- Select the listener method.
- Save IB and click Build and Go in Xcode to launch the application in the simulator.

Create a memory leak

- Introduce a memory leak in the listener method implementation.

```
[label setText:[[NSString alloc]
        initWithFormat:@"Button clicked:%d",
        count++]];
```

- Choose Run− >Start with Performance Tool− >Leaks to launch the application with Instruments running.
- Click the button a few times until you see Instruments registering events.

View Memory Leak

- Click on the Extended Details button to the left of the Leaked Blocks tab.
- Click a row in the Leaked Blocks tab.
- Scroll down the Extended Details area till you see your code segment.
- Double click the stack element to go to the appropriate section in your source file.

Resources

- iPhone Dev Center
- iPhone Developer Forum
- Apple Discussions Developer Forum
- iPhone Developer Tips
- Objective-C 2.0 Language