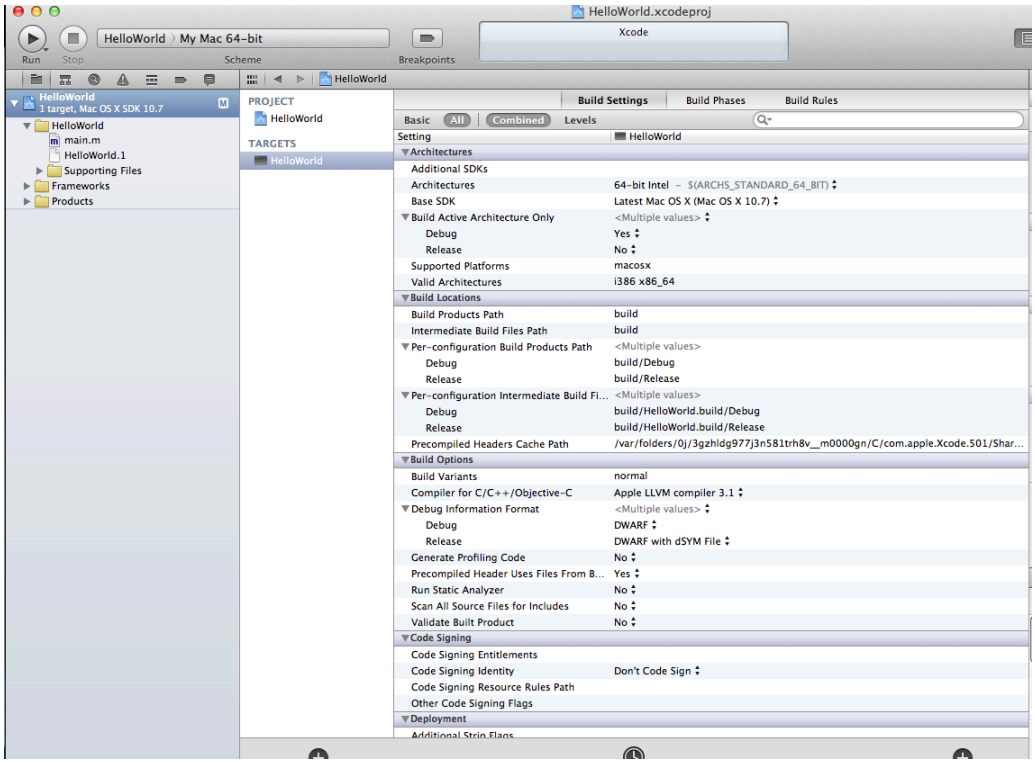
第1回Android開発者のための初めてのiPhoneアプリハンズオン勉強会 #andiphone
口頭分の補足メモ

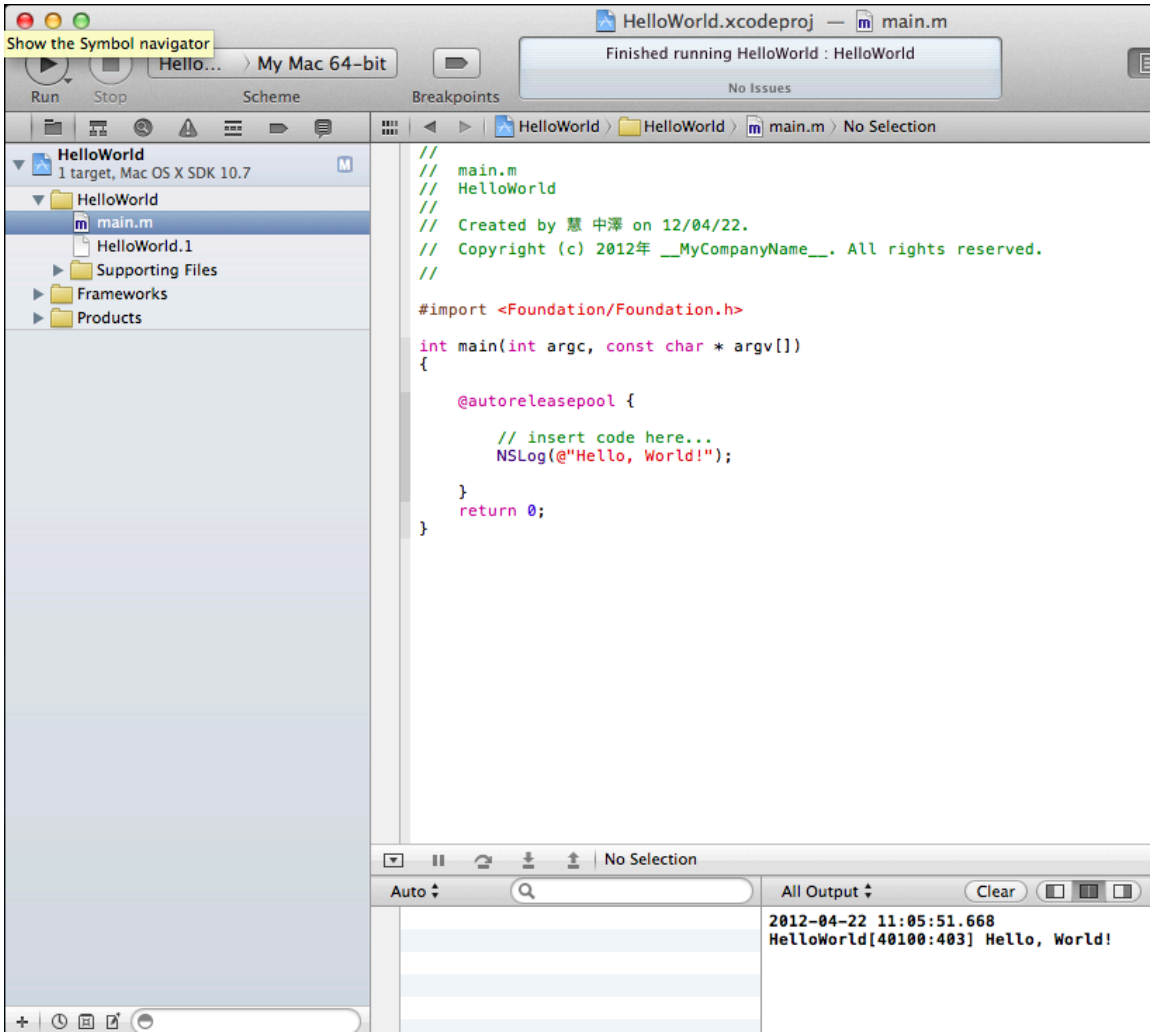
= はろーわーるどするよ =

まずは: Xcode開く→New→New Project
　その後、Mac OS X→Application→Command Line Tool を選ぶ→Nextを押す
　Product Name: HelloWorld
　Company Identifier: なんでもいい(空欄だったらedu.selfとかに)
　Type: Foundation
　Xcode4.3の場合”Use Automatic Reference Counting”のチェックが入った状態になってるので、それを外して下さい
　プロジェクトの保存先を指定して完了。



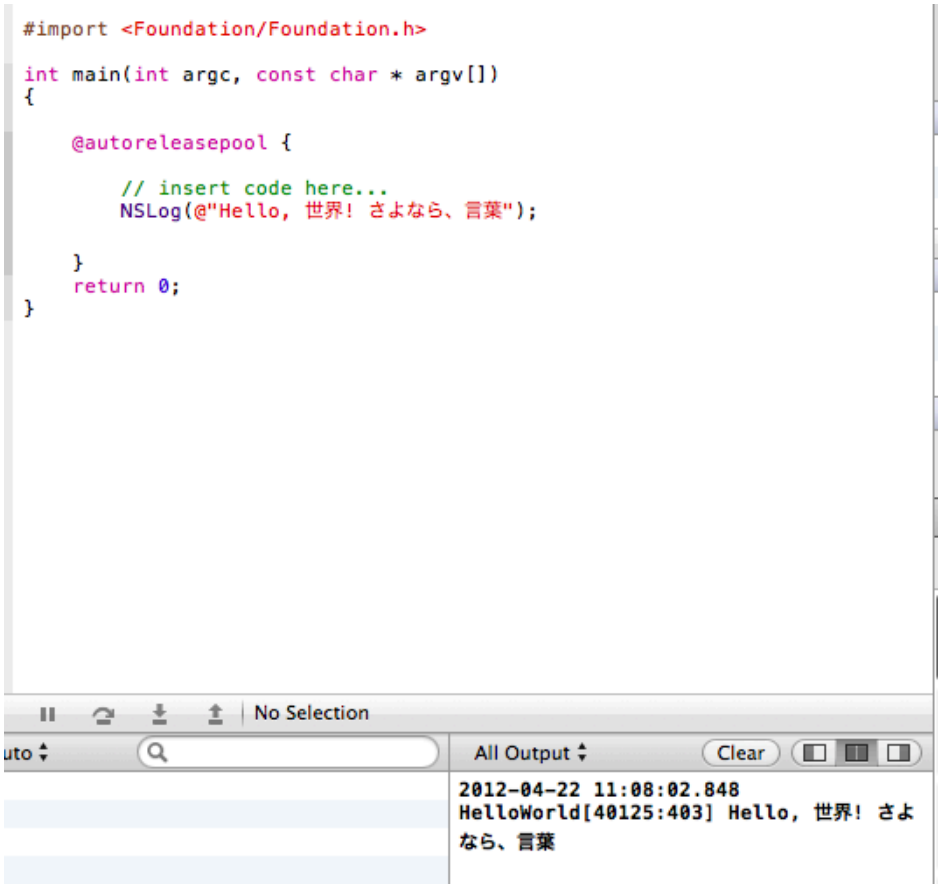
こんな画面になります。
左側のファイル一覧から main.m をクリックして開いて下さい。

おもむろに、左上の▸ Run ボタンをクリックして下さい。



右側がコンソール、左側が変数一覧などです。

実行できたら、”Hello, World”の文字列を適当に書き換えて実行してみてください。



4.2だとNSAutoReleasePoolというのが書かれてるはず。メモリ管理上必要なものなので、後から出てきます。こういうものがある、というのを覚えておいて下さい。

Objective-Cでログを出す際には常にNSLog()を使います。
JavaだとSystem.out.printf()に相当します。

```
// insert code here...
NSString* hello = @"Hello, 世界! さよなら、言葉";
NSLog(@"%@", hello);
```

こういう風に書き換えてみます。%sでも%cでもなく%@です。

マシントレードしていたのでここまで拾えてませんでした。

Objective-Cにおけるメソッド呼び出しという話です。
文字列を連結して出力するサンプル。

```
NSString* hello = @”Hello,”;
NSString* corosuke = @”corosuke”;
NSNumber* number = [NSNumber numberWithInt:1];
NSNumber* boolNumber = [NSNumber numberWithBool:YES];

int i = 0;

NSString* printString = [hello stringByAppendingString:corosuke];
NSLog(@”%@”, printString);
```

こんな感じです。

Objective-Cでは、まずtrue/falseは使わずYES/NOと書きます。定義は(BOOL)1と(BOOL)0だったりします。

Javaだとnullなところは、Objective-Cだとnilです(持つ意味も少々違います)。
NSArrayには、オブジェクトであれば(正確には、idを持つものは。後で説明します)何でも突っ込むことが出来ます。

次は辞書(Dictionary)を作成してみます。

```
NSDictionary* dictionary [NSDictionary dictionaryWithObjectsAndKeys:
                          hello,@"hello",
                          corosuke,@"corosuke",
                          number,"@number",
                          nil];
```

こんな感じです。最後のnilを忘れないようにしてください。
Objective-Cの辞書は、内容を先に、キーを後で指定するところにも注意が必要です。

NSArrayやNSDictionaryの初期化時、途中の要素にnilを指定してみるとどうなるか試してみてください。
→途中までしか表示されません。
arrayWithObjectsやdictionaryWithObjectsAndKeysという可変長の指定を受けるメソッドは指定をnil終わりにすることがあります。NULLを要素として指定したい際にはkCFNullを使います([NSNull null]も使えますが、実際使うことはありません。kCFNullのほうが早いです)。
Objective-Cでは、kで始まる識別子は定数とします。kCFNullを右クリックして定義見に行くと面白いかも。

終端指定のnilの代わりにkCFNullを指定すると停止できなくなるので注意してください。

HelloWorld関連、かるい初期化系はこれで終わりです。

メモリ管理の話

基本的にガベージコレクションが無いので、自分でメモリ管理をする必要があります。iOS 4.3からARC(Automatic Reference Counting)というのが加わったのですが今日は使いません。メモリ管理できないと困りますので。

初期化の定番句があります。

```
// allocで領域を確保、initで初期化
NSArray* array = [[NSArray alloc] init];

// 使い終わったらreleaseします
[array release];
```

これが基本のパターンですが、alloc/releaseを自動化するための仕組みもあります。

NSAutoreleasePoolを作っておき(例えばNSAutoreelasePool* pool = [[NSAutoreleasePool alloc] init])、
NSArray* autoreleaseArray = [NSArray array];
や
NSString* string = @”hoge”;
NSString* string2 = [NSString string];
というように、自前でallocせずにオブジェクトを生成したもの(またはオートリリースを明示したもの)については最寄りのAutoReleasePoolに格納されます。
これだけで自動解放されるわけではなく、
[pool drain];
とした時点で、格納されたものが解放されます。

さきほどの

```
NSString* hello = @”Hello,”;
NSString* corosuke = @”corosuke”;
NSNumber* number = [NSNumber numberWithInt:1];
NSNumber* boolNumber = [NSNumber numberWithBool:YES];
```

このコードを、自前でalloc/init/releaseする形にかえてみましょう。

```
NSString* hello = [[NSString alloc] initWithString:@”Hello,”];
NSString* corosuke = [[NSString alloc] initWithString:@”corosuke”];
NSNumber* number = [[NSNumber alloc] initNumberWithInt:1];
NSNumber* boolNumber = [[NSNumber alloc] initNumberWithBool:YES];
```

ちなみに4.3の場合、@autoreleasepool { } ブロックの中に書いたオブジェクトについては、drainを行わなくても不要になった時点で解放されます。

初期化した各オブジェクトには、retainCountというものが存在します。
例えば
NSInteger helloRetainCount = [hello retainCount];

NSLog(@"%ld", helloRetainCount);
とすると、1が出力されます。
[hello retain];
としてから再度
NSLog(@"%ld", helloRetainCount);
とすると、今度は2が出力されます。

[hello release];
とすると、カウントが1つ減って1になります。
さらに
[hello release];
とすると、カウントが1つ減って0になり、晴れてメモリが開放されます。

retainはObjective-Cでコードを書いているときによく出てきます。このカウントの対応付けがうまくいっておらず、変数スコープを抜けてもカウントが残った場合にはメモリリークとなります。

...実行するとretainCountが1ではなく-1からスタートするけどなんでだろう?
→多分Mac OSだから。実際のGC(ガベージコレクション)が走るタイミングでメモリ上から消される。
このあたりiOSとは微妙に違う模様。

午後は13:30からです
今しばらく自習などしていて
くださーい

午後のメニュー:
13:30-14:30 - クラスを作る
14:30-15:00 - iOS事始め
15:00-15:30 休憩
15:30-16:30 - カスタムUIを作る(IB使う)

※途中分からないところが出てきたら、
前までくるか声をあげるかtwitterで叫ぶか
してください。初心者殺し多いので
分からないまま放置すると大変。

ここから午後の内容

クラスを作成する

メモリ管理あたり、あんまり理解できてないかもだけど気にせずまずはクラスを作ります。

左側のファイル一覧のmain.mあたりを右クリックし、メニューから"New File..."。
出てくる画面でObjective-C classを選択しNext。
クラス名はひとまずMYClassとして、superclass ofは"NSObject"のままでNext。
(Objective-Cでは、通常クラス名の先頭2文字を大文字にします)

外に見えるものを.hファイルに書き、実際の実装を.mファイルへ書いていきます。
.hファイルの@interfaceから@endまでの間に
- (void)method1;
と書いてください。こういう風になるはずです。

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

@interface MYClass : NSObject
- (void)method1;
@end
```

これに対応する実装は.mファイルに

```
#import "MYClass.h"

@implementation MYClass

- (void)method1 {

}
@end
```

このように書きます。適当に実装しましょう。

```
- (void)method1 {
    NSLog(@"ころすけあざとい");
}
@end
```

main.mから使ってみましょう。ヘッダのインポートは
#import “MYClass.h”
とします。

ここまで書くと、コードはこのようになります→ http://pastebin.com/AbK2nKvN

ここで、最初を - で始めたメソッドは「インスタンスメソッド」です。
+で始めたものは「クラスメソッド」となります(JavaでのstaticメソッドのようなもOの)。
※インスタンスにひもづくかクラスにひもづくか、というところが違い、呼び出し方法も違います。
+ (void)classMethod;
と定義したメソッドは、この場合
[MYClass classMethod];
として呼び出します。

次に、クラス内へ変数を定義してみます。

```
@interface MYClass : NSObject {
    NSString* name;
    int age;
}
```

ヘッダファイルの@interface MyClass : NSObjectという行をこのように変更します。
さらに、これらの変数を扱うメソッドを追加します。

```
- (NSString*)name;
- (void)setName:(NSString*)str;
```

引数も含めると、Objective-Cでの関数定義は +/- (戻り値の型) 関数名:(引数型)引数 という書式で行うことになります。

```
- (void)setName:(NSString*)str {
    if (name) {
        [name release];
        name = nil;
    }
    name = [str retain];
}
```

名前を設定するメソッドはこんな感じになります。ここで、呼び出しの度にretainをひたすら呼んでいくと参照カウントが延々と増えることになるので、nameに中身が入っている場合は一旦releaseしてnilを投入しています。
JavaでのNullPointerExceptionなどと違い、nilなオブジェクトに含まれるメソッドを呼び出してもObjective-Cではエラーになりません(単純に何もしないだけです)。

ここで、Objective-Cではどのクラス/オブジェクトの中で何を参照している、という状態の管理を行っていないため、プログラマは明示的に自オブジェクトに含まれるオブジェクト(この場合はname)を解放してあげる必要があります。このために用意されているのがdeallocです。
MYClass内に以下のメソッドを実装することで、メモリの解放を明示的に行うことが出来ます。

```
- (void)dealloc {
    [name release];

    [super dealloc];
}
```

プリミティブ (この場合はint age)については明示的な解放の必要はありません。

ここまでやると、MYClass.mの実装はこんな感じになります。
http://pastebin.com/DAuFssk7

インスタンス変数については、
Apple式: NSString* _name;
Google式: NSString* name_;
と、アンダーバーを付けるのは同じですが流儀があります。今回はApple式です。

これにプロパティを実装すると、以下のようになります。
http://pastebin.com/j1KQrh3C
説明はこれからします。

MYClass.hから、さきほど書いた
- (NSString*)name
- (void)setName
をごっそり消します。もう要りません。MYClass.mからも消しましょう。
代わりにMYClass.hの@interface から @endまでの間に

```
@property (nonatomic, retain) NSString* name;
```

と書き、また MYClass.m でも
@implementation MYClass の次行あたりに
@synthesize name = _name;
と書きます。
今のコードはこんな感じ。
MYClass.h:

```
@interface MYClass : NSObject {
    NSString* _name;
    int _age;
}
- (void)method1;
+ (void)classMethod;
```

```
@property (nonatomic, retain) NSString* name;
@end
```

MYClass.m:

```
#import "MYClass.h"
@implementation MYClass
@synthesize name = _name;
- (void)method1 {
    NSLog(@"ころすけあざとい");
}
+ (void)classMethod {
    NSLog(@"クラスメソッドなんだよ");
}
- (void)dealloc {
    [_name release];
    [super dealloc];
}
@end
```

main.m:

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

int main(int argc, const char * argv[]) {
    NSAutoreleasePool* pool = [[NSAutoreleasePool alloc] init];

    MYClass* instance = [[MYClass alloc] init];
    [instance method1];
    [MYClass classMethod];
    [instance release];

    [pool release];
    return 0;
}
```

しばしQ&A。
ヘッダで定義した変数は、そもそも全部プライベートになってる(けど、名前が分かっていれば呼べる)。
メソッドについてヘッダに定義せず実装だけ書くとXcode上では警告が出るけど、呼ぼうと思えば外部から普通に呼べる。

さて、上では出てこなかったけどよく使うのが「引数を複数持つメソッド」。Objective-Cではこう書きます。

```
- (NSString*)stringWithName:(NSString*)name Age:(NSString*)age;
```

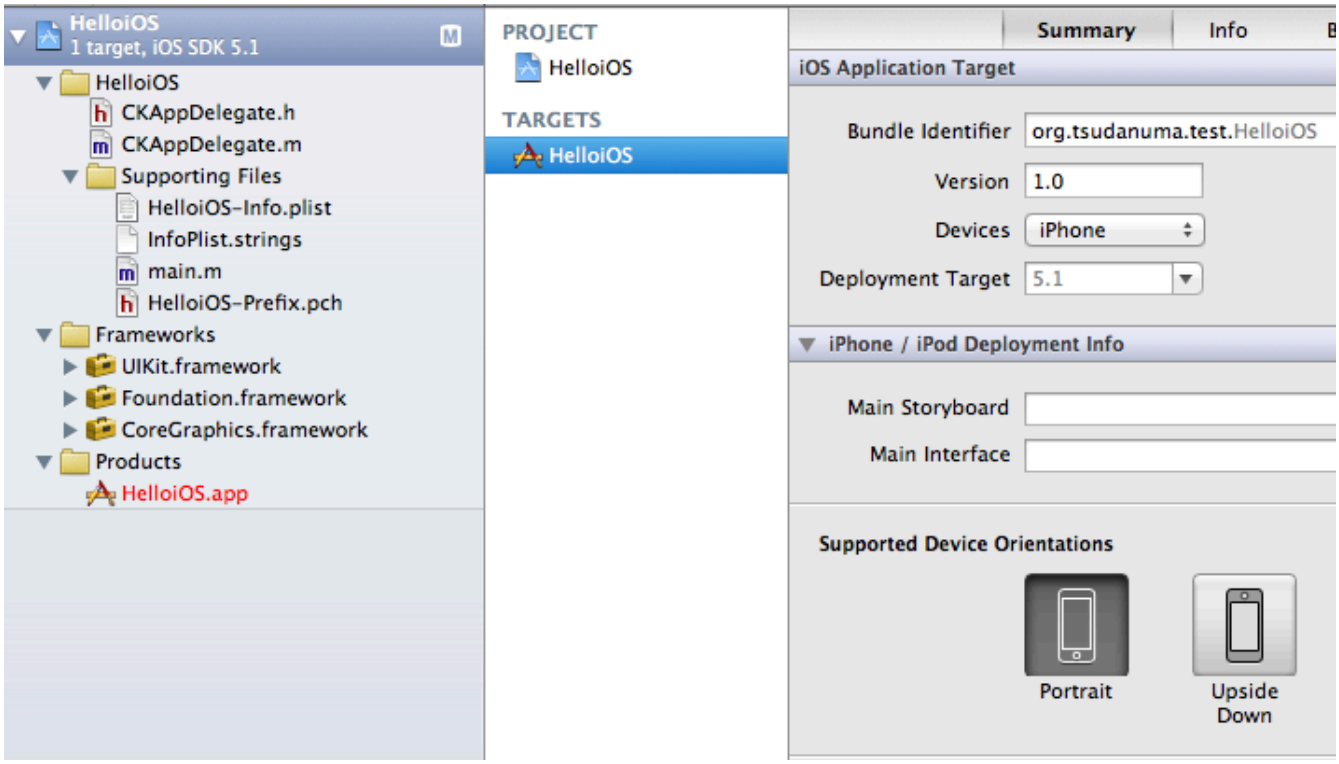
ここでAgeはメッセージ名(表記ゆれがあってタグと呼んだりもする。メッセージタグとか)と呼ぶ(※実はメッセージ名省略できるけど推奨されないのでここでは扱わない)。

Objective-Cのポリシーは「メソッド名を自己説明型にすること。名前が長くなってもいいから、外部からメソッドを叩く際に迷わないように」というのを重視する。

クラスの話はここまででおしまい。

はじめての**iOS**アプリ

ここまで開いていたプロジェクトは閉じて、Xcodeのメニューから再度
File → New → Projectと行き、出てきた画面でiOS Applicationを選択してください。
様々なテンプレートが出てくるので Empty Application を選択し、Next。
Product Name: HelloiOS
Company Name: 適当な名前か、よく分からなければself.edu
Class Prefix: CK
Device Family: iPhone
下の3つのチェックはいずれも 外して おいてください。
Nextを押して保存先を指定し、完了すると



こんな感じになるはずです。
ディレクトリ構成:
HelloiOS以下: 今回のメイン部分です
Frameworks: この場合は、ライブラリの参照が書かれています。UIを扱うもの、NSLogなどのユーティリティなどがデフォルトで含まれています。

まずはぽちっとRunを押して(またはCtrl+Rで)シミュレータ実行しましょう。

少々いじってみます。
CKAppDelegate.m からdidFinishLaunchingWithOptionsメソッドを探して

```
self.window.backgroundColor = [UIColor whiteColor];
```

となっている部分を

```
self.window.backgroundColor = [UIColor blueColor];
```

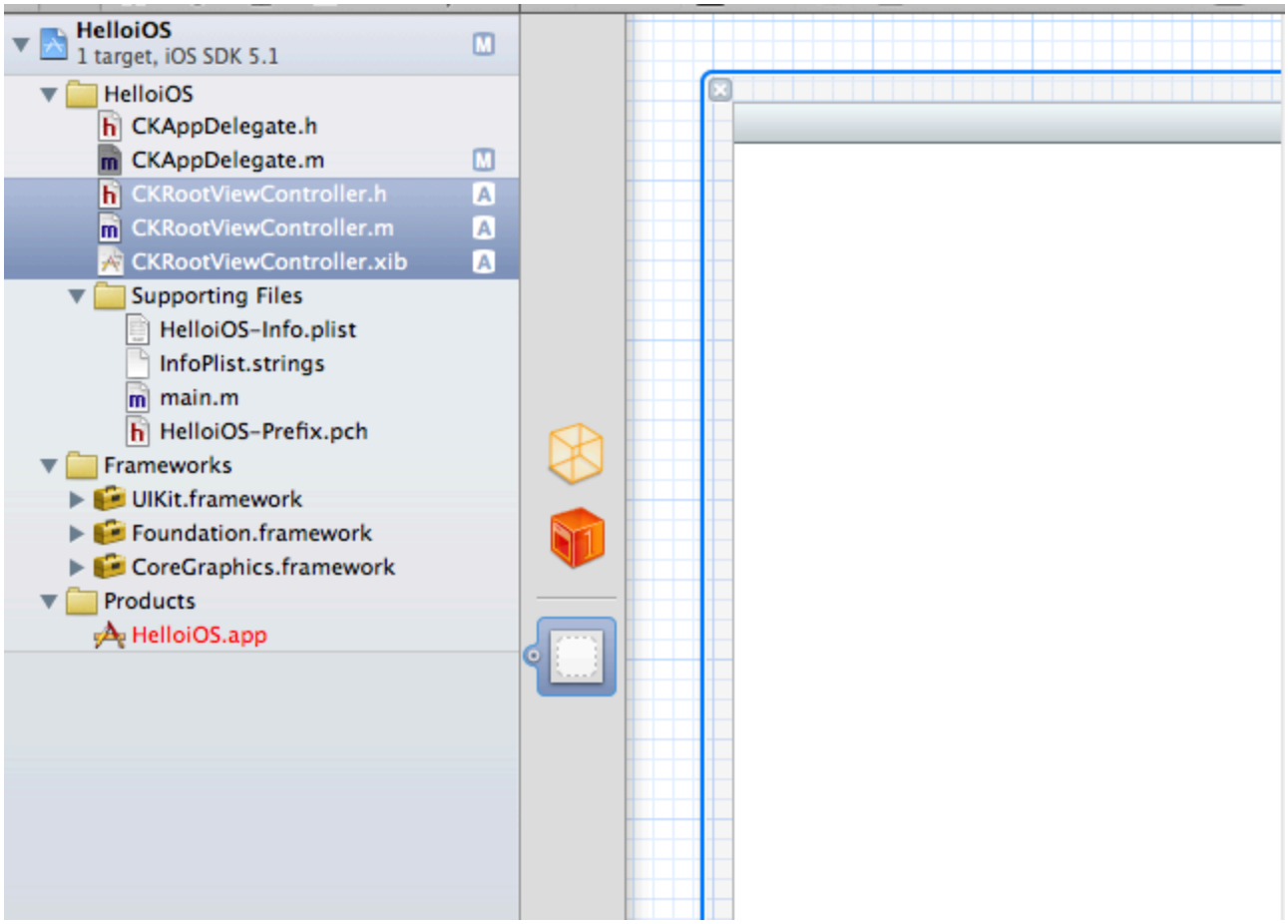
に変えてもう一回実行してみましょう。「既にシミュレータ上でアプリが動いてるけどどうする?」というように聞かれた場合は、終了を選択するかXcodeの左上にある ■Stop を押して停止してください。

CKAppDelegate.h に書かれてるのがウィンドウの宣言です。

```
@property (strong, nonatomic) UIWindow *window;
```

今回は1ウィンドウしか使わないのでこれだけです。
もう少し実装をしましょう。画面を作っていきます。

File → New → File で左側のセレクタが iOS Cocoa Touchに合っていることを確認のうえ、
Objective-C class(4.2の場合は多少違う)を選択してNext。
クラス名: CKRootViewController
Subclass of: UIViewController
として、”With XIB for user interface”にチェックを入れてNext。
保存先を確認してファイルを作成しましょう。



このように、ファイルが3つ増えて画面編集エディタが開くことを確認してください。
次の準備のために、無心で以下のコードをCKAppDelegate.mに書きます(既存のメソッドを編集してください)。

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{
    self.window = [[[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]]
autorelease];
    // Override point for customization after application launch.
    self.window.backgroundColor = [UIColor blueColor];
    CKRootViewController* vc = [[CKRootViewController alloc] init];
    CGRect viewControllerFrame = CGRectMake(0, 20, 320, 460);
    [vc.view setFrame:viewControllerFrame];

    [self.window addSubview:vc.view];
    [self.window makeKeyAndVisible];

    self.rootViewController = vc;
    [vc release];

    return YES;
}
```

対応する形で

```
@property (nonatomic, retain) CKRootViewController *rootViewController;
```

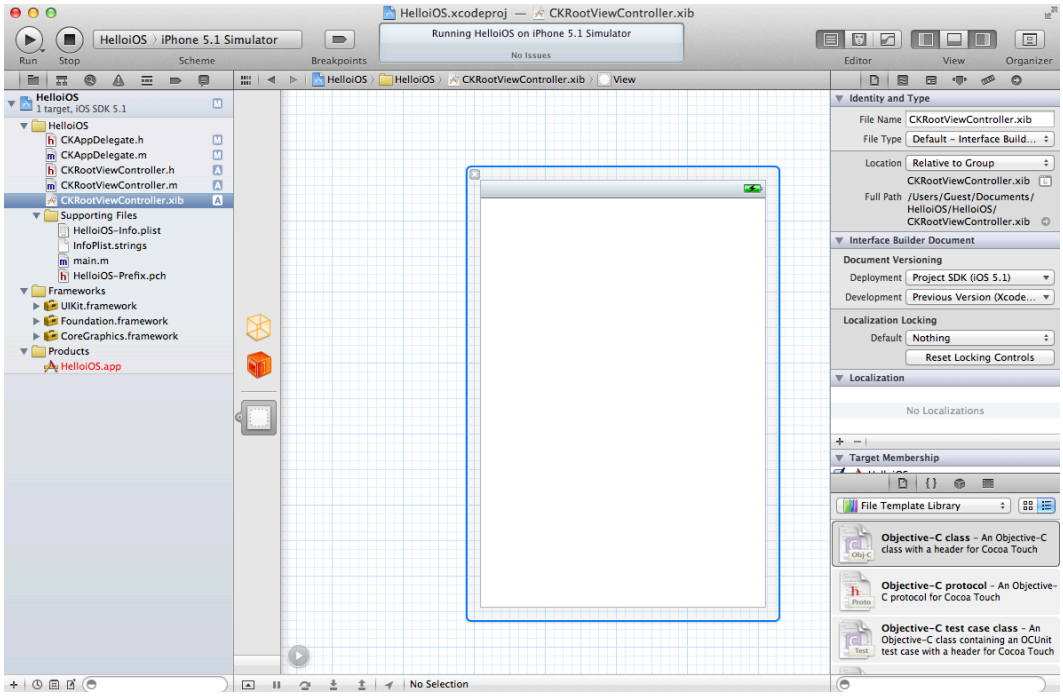
をCKAppDelegate.hに記載してください。

http://pastebin.com/SqJQrBC0
CKAppDelegate.hとCKAppDelegate.mのコードこんな感じです。
これを実行(Ctrl+R)してみて、白い画面が出てくればOKです。

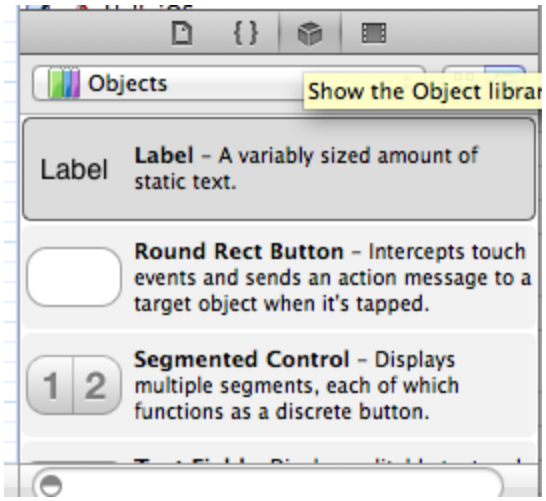
次は、ついに画面のお絵描きタイムです。
左のファイル一覧からCKRootViewController.xibを開いてください。

Xcode右上のビュー部分で、右端のもの(Utilitiesといいます)が選択されていない場合は選択してください。

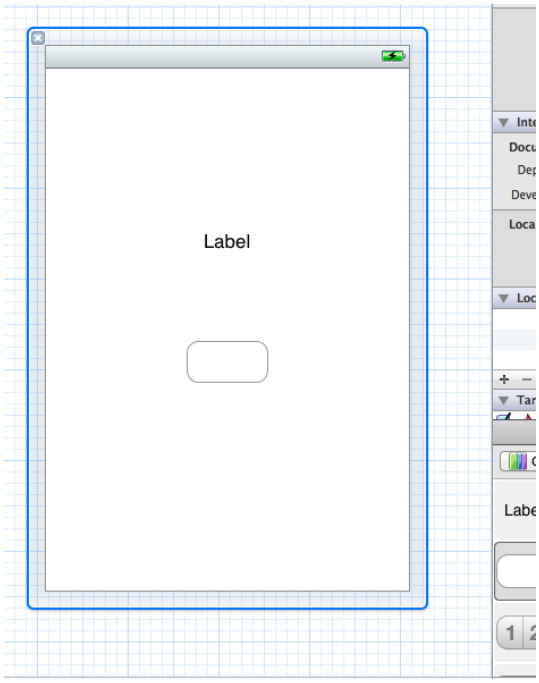
こういうふうに、右ペインが表示されていればOKです。



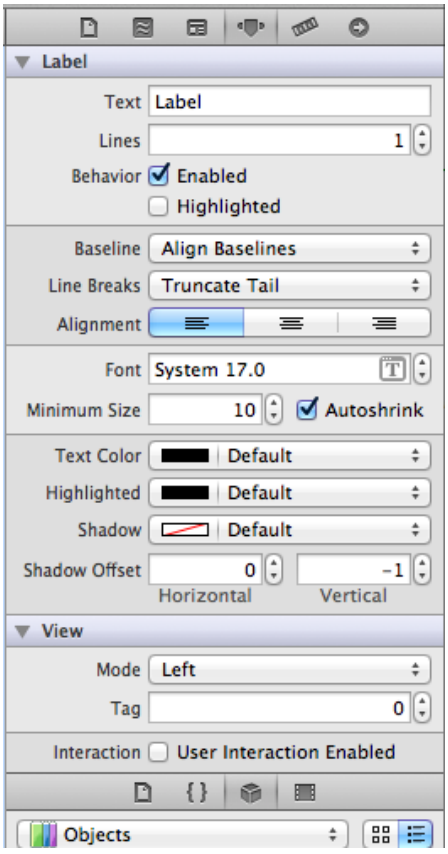
まずは画面上にラベルを配置します。画面右下のパネルで



立方体マークをクリックし、表示を切り替えておきましょう。このエリアから”Label” “Round Rect Button”をドラッグし、画面イメージ上にドロップしてください。

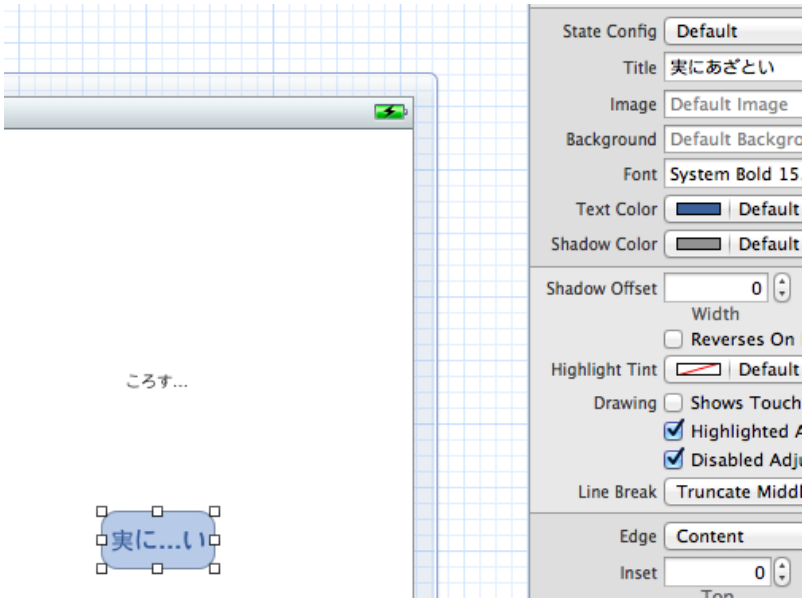


こんな感じになります。いくつか、オブジェクトのプロパティを変更してみましょう。

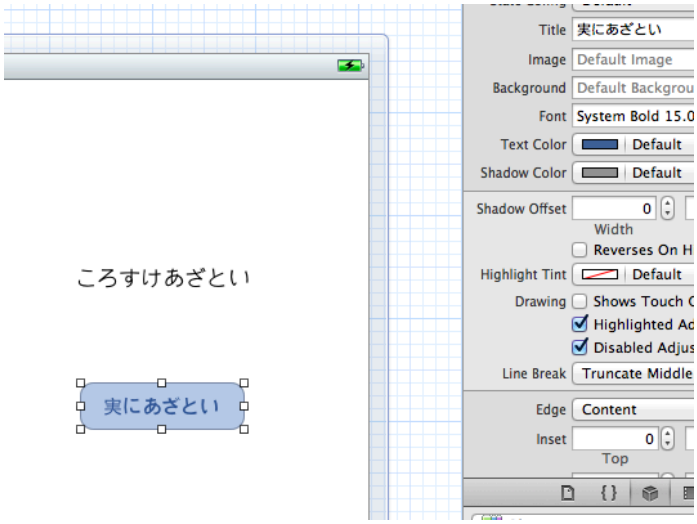


右上のパネルで、左から4番目(4.2だと違うかも)の"Attributes"を選択し、Textを変更します。

ここでは、ラベルのテキストに”ころすけあざとい"、ボタンのタイトルに”実にあざとい”と書いてみます。



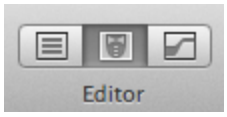
このように文字列が見切れてしまった場合は、オブジェクトの端をドラッグして大きくしましょう。



こんな感じになります。ここでは、ついでにラベルのalignmentを変更してセンタリングしています。

次に、コード内からオブジェクトを利用できるようにプロパティとしてバインドします。ここでは、Interface Builderを使ってプロパティのひも付けを行います。

まず、画面右上の



Editors部分で中央のAssistant editorを開きます。次にInterface Builder内のラベルをクリックして選択し、Ctrlを押しながらCKRootViewController.hの@interface から @endまでの間にドラッグします。
ドロップすると名前をつけるように求められるので、ここでは contentsLabel とします。

同様に、ボタンを選択後Ctrlを押しながら右側のヘッダへドラッグ &ドロップすると、こちらも名前をつけるように求められます。ここでは changeAction とします。

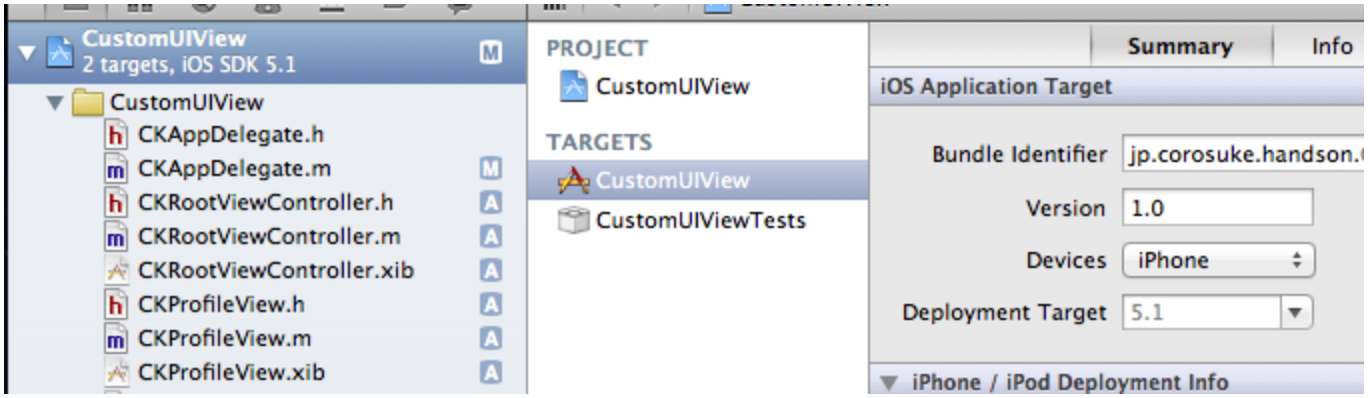
こんな感じになります。

右上のEditorメニューから左端の Standard editor へ戻し、CKRootViewController.mを開くと@synthesize追加されてるし、deallocもちゃんと追加されてます。えらい。

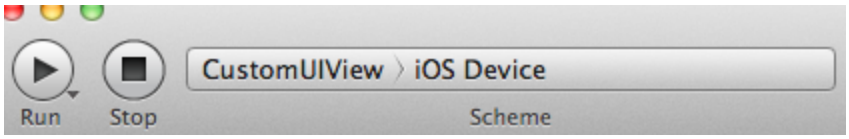
```objc
#import "CKRootViewController.h"

@interface CKRootViewController ()

@end

@implementation CKRootViewController
@synthesize contentsLabel;
@synthesize changeAction;

- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)nibBundleOrNil
{
    self = [super initWithNibName:nibNameOrNil bundle:nibBundleOrNil];
    if (self) {
        // Custom initialization
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];
    // Do any additional setup after loading the view from its nib.
}

- (void)viewDidUnload
{
    [self setChangeAction:nil];
    [self setContentsLabel:nil];
    [self setChangeAction:nil];
    [super viewDidUnload];
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
    interfaceOrientation
{
    return (interfaceOrientation == UIInterfaceOrientationPortrait);
}

- (void)dealloc {
    [changeAction release];
    [contentsLabel release];
    [changeAction release];
    [super dealloc];
}
@end
```

このあたりまでの主要ファイルのソースは以下のようになっているはずです。
http://pastebin.com/pU3KULCS
zipでくれ→ zipです http://t.co/1t5f4L5l

まずはzip展開してプロジェクト開いて、ビルドが通るか試しましょう。



シミュレータで起動出来ず、



スキーム一覧にiOS Device以外が表示されない場合、CustomUIViewのプロジェクトをクリックして表示されるプロジェクト設定からSummary → iOS Application Target → Deployment Target で、5.1または5.0を選択してください。iPhone 5.1 Simulatorが選択肢に追加され、選択してRunすると ころすけの名刺 が表示されるはずです。

CKProfileView.xibをクリックして開き、内容をいじって自分の情報に変更してみましょう。

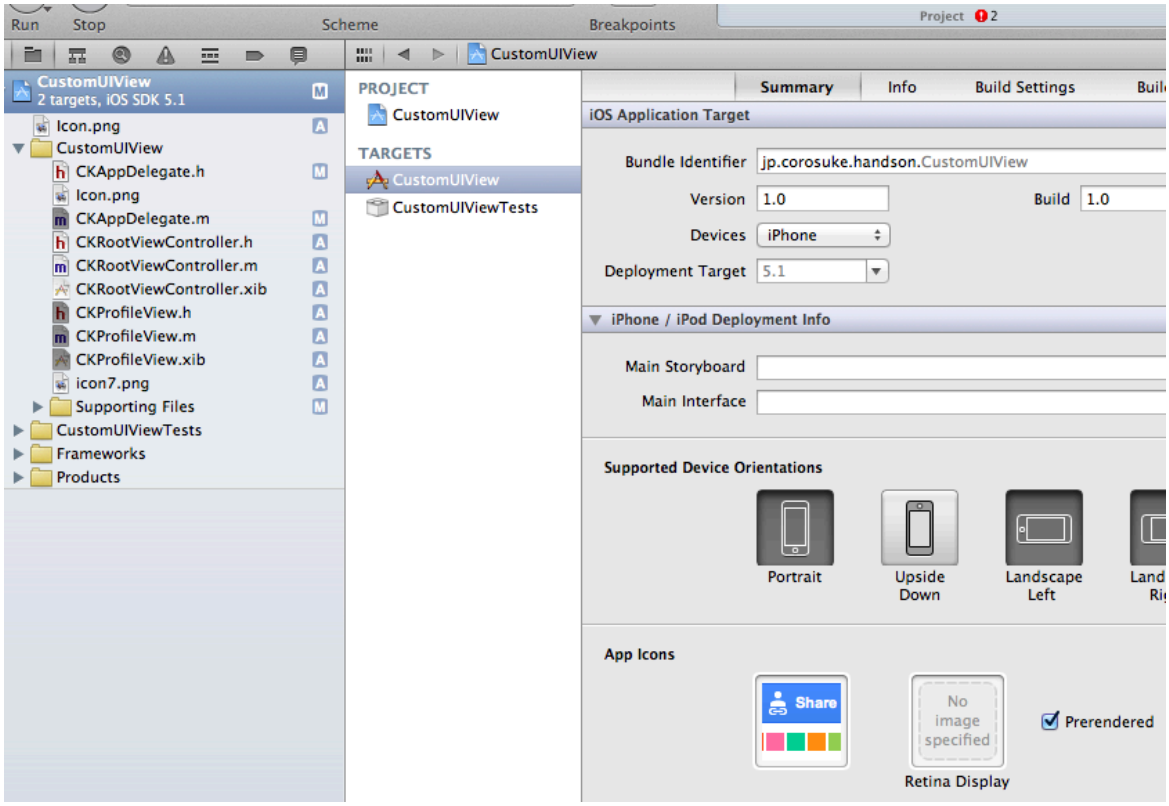
いじったらビルドしてみて正しく表示されるのを確認してください。

シミュレータのホームを見ると真っ白な画像がアイコンとして表示されていて寂しいので、適当にアイコンを作成しましょう。

小さいアイコンは57x57にする必要があります。適当にpngで作成し、Xcodeのプロジェクト内ファイルー覧ビューへドラッグ &ドロップします。

こんな感じになります。

次に、ファイルー覧ビューで"CustomUIView"をクリックし、表示される一覧からTARGETS → CustomUIView が開かれていることを確認、Summaryタブを開いてください。



ここでIcon.pngをApp Iconsの左側にドラッグ &ドロップすると、このスクリーンショトのようにアイコンが指定出来ます。このとき、Prerenderedにチェックを入れてください。
Run(Ctrl+R)し、シミュレータのホームボタンを押した際にアイコンが反映されていればOKです。

参考書籍:
- 詳解Objective-C 2.0 第3版
- iPhoneプログラミング UIKit詳解リファレンス

- エキスパートObjective-Cプログラミング

よくわかるiPhoneアプリ開発の教科書
ドM向け: iOSプログラミング第2版(どちらかというと原著の第3版)

Apple公式ドキュメントまとめ: http://golog.plus.vc/iphone/879/

以上、おつかれさまでした!