第４回
「アイデア出しとアプリ作りと」

toru.inoue@kissaki.tv

Agenda

1. プログラミング言語 座学 セミファイナル
2. あなたのアプリケーションの物語

Drive1/50

1.プログラミング言語 座学
セミファイナル

Drive2/50

1-0-0.迷ったら進めは名言だが金言ではない

今日で、今まで作ってきた物の全てがしっくりくる、、手前まで行きます。

- 前回までの授業を受けている事が前提です。
- 迷ったら、今までの資料を見直してみてください。
- もうすぐ作り出せる天国タイムがくるので、後少しの辛抱ですわ。

Drive3/50

1-0-1.びーむ、再び

呼び名の話

A * a とか、

メソッドでびーむを受けていた方の事とか、aって物体のことを、

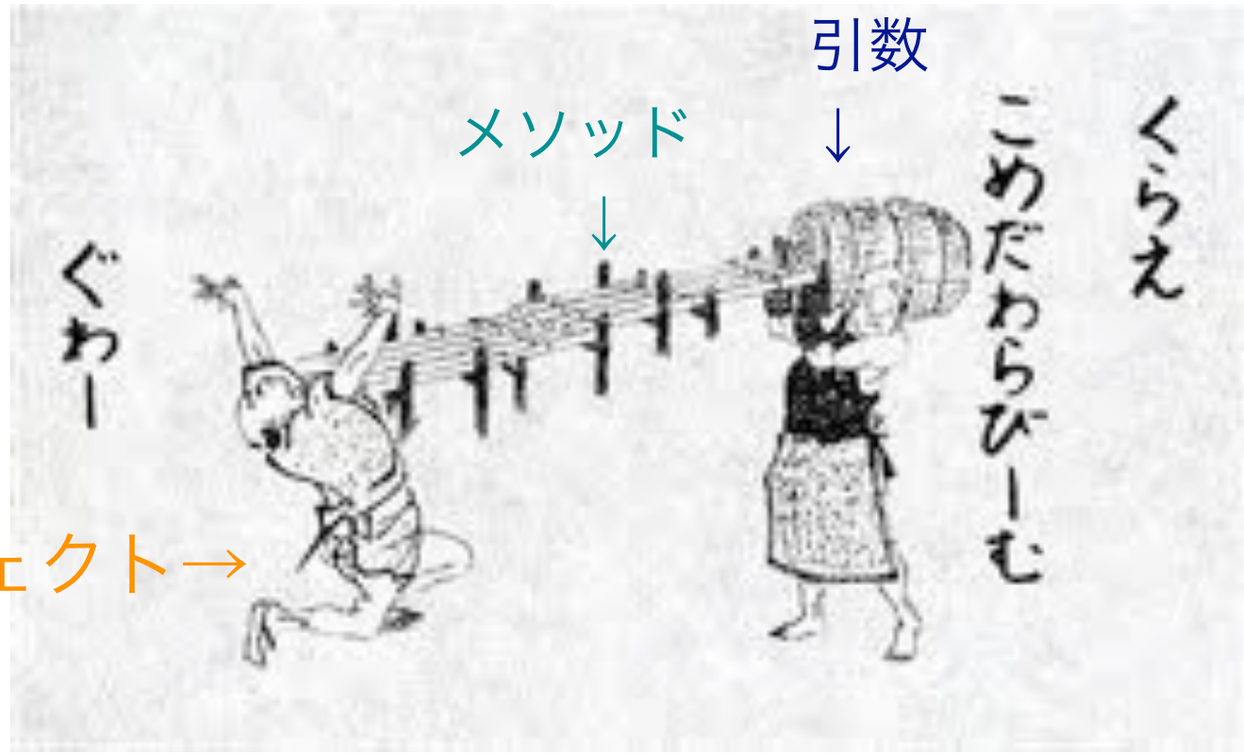
オブジェクト、と呼ぶ。

オブジェクトを作り出すひな形のことを、

型(かた、英語だとクラス class)、とよぶ。

A:型

a:オブジェクト



Drive4/50

1-0-2.つ○だ*○ろ

A * a　は、
　Aで出来てる名前がaのもの、という意味とか前回言ったと思います。

　正しくは、
　Aという型で出来ているaという名前のオブジェクト
　になる訳ですね。

　で、たとえばこんな風に使ってた。

```
PrototypeViewController * pVCont = [[PrototypeViewController alloc] init];
```

Drive5/50

1-0-3.プレゼン資料の書き方が微妙に変わっているのは仕様

ここで、alloc と init　メソッドは、

　PrototypeViewController　という型でできてる、

　pVContっていうオブジェクトを作り出すメソッドです。

　allocでメモリ上に型の大きさ分の面積を確保(allocate)し、

　initで型の内容をメモリ上に展開(initialize)します。

　いきなり「ざわ、、、ざわざわ、、」　みたいな感じですが、C言語習うと

　わかりやすいというか、C知らないとさっぱりです。

~~C言語雑学は、時間のある時に気が向いたら~~

参考

　@iwata さんお薦め　苦しんで覚えるC言語

　http://9cguide.appspot.com/

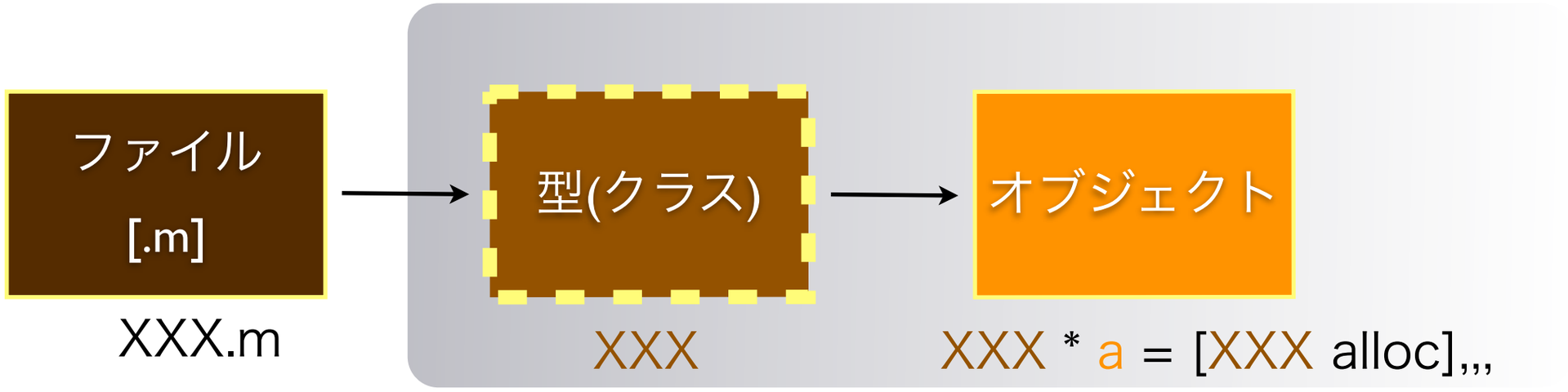
　まあ苦しんで覚える必要は無いと思うが。

Drive6/50

1-0-4.型(クラス)ってなんなんだ!!

オブジェクトを作り出すひな形、ってさっくり言ってますが、
具体的には、.mファイルに書いてある内容です。

XXX.mってファイル作る　→　プログラム中で
XXX * a = [XXX alloc],,,　とか、書くと、
プログラムの中にオブジェクトが作れる訳ですね。



ファイルを源にして、オブジェクトの基になってる事から、型(クラス)といいます。
この事から、.mファイルの事を、特に「クラスファイル」とか呼びます。

Drive7/50

1-0-5.人生いろいろ、型もいろいろ

「UIViewとか、UIViewControllerとか、色んな型に出会ったよな、兄者。」

「intとか、doubleとか、floatとかもあるんだぞ。　凄いんだぞ。」

「さーすがー。　兄者は物知りだなー。」

フーン

UIViewは、辞書引くと、描画したりタッチできるビューの型、
とか出てる今まで使ってきたような型ですが、

数値を扱うのに特化した、floatとかintという名前の型もあります。
それぞれ、Integer:整数型(whole number)、Float:浮動小数点型の事です。

```
int a = 100;
```

とか書くと、
aというint型の値に100という数字が入っていることになります。

Drive8/50

1-0-6.オブジェクトと一口に言えない

```
int a = 100;//aは値といい、オブジェクトとは言わない。
```

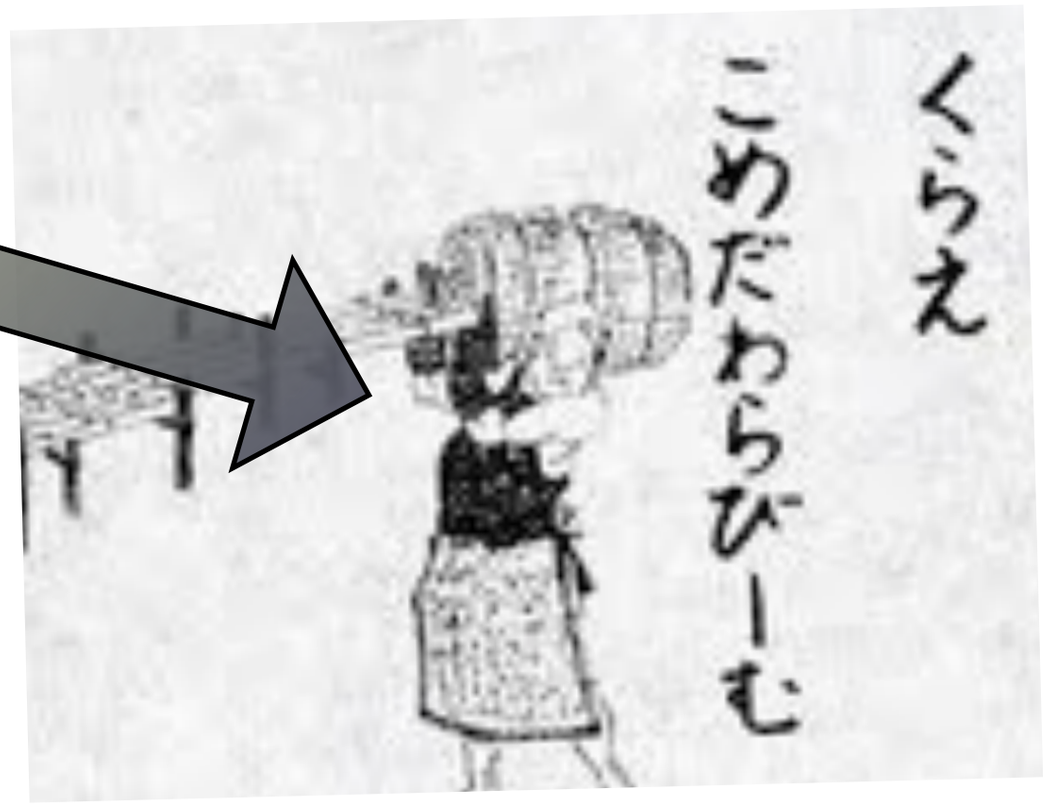
なんで値というか、というと、
今迄の話題のオブジェクトでは無いんですな。
=100(数値)とかダイレクトに書いてて、
allocとか、initとか、無いしな。

違いは、ほかにもいろいろあるんだけど、まあ、追々。

int, float, double, byte, char などなど、値にはいろいろな種類があります。
値の詳細について、個別には取り上げません。~~だって面倒~~
バリエーションについては参考書籍読んでな。
どれか一つでも引けば芋づる式に全部見れます。
これで、型の話は一段落です。

Drive9/50

1-1-0.びーむを撃っていたのは誰だ



こいつな。
メソッドを使い、
オブジェクトに向かって、
引数付きでびーむぶちかましてる
ドSなこいつ。

誰でしょうな、というと、
その時動いているプログラム、つまり自分自身というオブジェクト、です。

プログラムで[object method:arg]; とか書いたとすると、実際には、
このプログラムが書いてあるオブジェクト自身(self)が、objectに向かって、
method:argを実行しています。

で、プログラム中で、特に自分自身のオブジェクトを、
self　と書くことが出来ます。

Drive10/50

1-1-1.自分で自分に、自分の持っているメソッドを使う

自分で自分に何かメソッドを使うには、

[self method:arg];

って、やると、自分で自分に向かって
自分の持ってるmethodが使えるんですわ。

もちろん、自分がmethod:っていうメソッドを
持っている場合にしか、きちんと動きません。

Drive11/50

1-2-0.型の中のオブジェクト

型(.mファイルの内容)の中には、いろんな他の型を書く事が出来ます。

PW>PrototypeAppDelegate.hとか開くと、

```
#import "PrototypeViewController.h"

@interface PrototypeAppDelegate : NSObject <UIApplicationDelegate>
{

    UIWindow *window;

     PrototypeViewController * pVCont;←ここね

}
```

PrototypeViewController を取り込めているのがわかると思います。

こんな風に、他の型のオブジェクトを書く事で、

この型が持っている要素として、取り込むことが出来ます。

Drive12/50

1-2-0.型の中のオブジェクト

型(.mファイルの内容)の中には、いろんな他の型を書く事が出来ます。
ってさっき書きましたけど、今までのはほんとに全部そうなのかな。

こんなのあったよねー。

PW>PrototypeAppDelegate.mを続けて開くと、

```
pVCont = [[PrototypeViewController alloc] init];
[window addSubview:pVCont.view];
```

pVContはPrototypeViewControllerという型のオブジェクトです。
で、viewってなに？

きっとある筈。あるに決まってる。ガイアがそう囁いてる。

探してみましょう。

Drive13/50

1-2-1. ねーじゃん！

PrototypeViewControllerのプログラムの中にも、viewなんてねーじゃん！

確かに、書いてないです。PrototypeViewController.mにも、
PrototypeViewController.hにも、viewなんて出てきません。

って言いつつ、実は、PrototypeViewController.hの上の方に、

```
@interface PrototypeViewContloller : UIViewController {
```

とか書いてあります。　ヒントになりそうです。

Drive14/50

1-2-2. 誰だぅ！

は？　UIViewController？

なにそれ、と思ったら、辞書引いたりしてみましょう。

(右クリック > テキストを製品ドキュメント内で検索)

下の方まで見てみると、、

あ、viewってあった。

view property

→pVCont.viewのviewは、実はコレです。

秘密は @interface PrototypeViewContloller : UIViewController にあり、
これ、：の前後にある記述には、一定の関係があるんです。

PrototypeViewControllerっていう型は、

UIViewControllerっていう型が持ってるものすべて引き継いだものです、
という意味なんです。
Appleのドキュメント資料だと、inheritsとか書いてあります。

Drive15/50

1-2-3. inheritance 1U・C相続［継承］物件, 相続財産；（先祖前代などから）受け継いだもの

UIViewControllerのClass Referenceを読むと、

> UIViewController Class Reference
>
> **Inherits from**
> UIResponder : NSObject

とか書いてあります。
この型自体も、UIResponderとNSObjectをinheritsしてますね。

PrototypeViewControllerはUIViewControllerを全部引き継いでるから、
UIViewControllerのviewっていう要素も、持ってるので、使える、と。

で、PrototypeViewControllerで出来てるpVContも、
UIViewControllerを全部引き継いでるから、viewを持ってて、使える、と。

Drive17/50

1-3-0.ドットシンタックス dot syntax (ぶっちゃけ . の事ね)

pVCont.viewとか見てると出てくるように、
aが持ってるbというオブジェクトを書くとき、a.b とか書ける。

という場合があるだけで、ぶっちゃけどうでもいいので、

「そういうふうに書ける場合がある」という程度の認識でOKです。

Drive18/50

1-4-0.返り値の話

メソッドは、オブジェクトや数字を「出す」ことができる。

オブジェクトに対してメソッドを使ってアレコレさせて、
出てきたもの = 返ってきたもの = 返り値、 と呼んでいる。

前回、こんなのを見た筈。

```
                                ↓このへんとか
    UIView * hoatyaa = [[UIView    alloc] initWithFrame:CGRectMake
(0, 0, 320, 480)];
    [hoatyaa setBackgroundColor:[UIColor blueColor]];
                                                    ↑このへんとか
```

Drive19/50

1-4-1.数学と一緒

これ、一つ一つの括弧[]について、数学と同じ事が部分的に言えたりします。
　実は内側の括弧を先に計算する、という法則があります。
　まあ、数学の基本として習ってますね。

　y = x + (x * (2000 / 16) * (20 + (60 / 20))) とかね。うわぁキタネエ。
　　4　　3　　2　　1、数字の小さい順に計算するよね。

(60 / 20) = 3　とか、括弧のなかの数字が、計算結果を出すように、
ここで入れ子になってる、中のメソッドは、何らかの物体を生成します。

言い方を変えると、メソッドは、いろんな型の物体(オブジェクト)を
"返す"事が出来ます。
　日本語だと、"返す"、英語だと、return。英語の方がしっくりくるかな。

Drive20/50

1-4-2.なんでも発射

intやfloat、double(数字、小数点付き)とかを返すメソッドもあれば、
UIViewとかの型を返すメソッドも書くことがでます。

で、特にvoid型を返すメソッドは、

```
- (void) A;//とか.hに書きます。
- (void) A {//とか、.mに書きます。
    //なんか処理
}
```

いままで何度も見ましたなあ。

- (返す型) メソッド名:(引数の型)引数の仮の名前

これで、メソッドの説明があらかた終わりました。長かった。。。

Drive21/50

1-4-3.その数字、そのままバットで打ち返すよ

intを返すメソッドの例

```
- (int) retInt {
  return 100;
}
```

```
UIView * hoatyaa =[[UIView alloc] initWithFrame:CGRectMake
([self retInt], 0, 320, 480)];
```

```
int i = [self retInt];//こんな風にも使えます
NSLog(@" i をとりもどせ！_%d", i);
```

Drive22/50

1-4-4.オブジェクトを返してみよう

UIViewのオブジェクトを返すメソッドの例

```
- (UIView * ) getUIView {
  UIView * view = [[UIView alloc] initWithFrame:CGRectMake
(0, 0, 320, 480)];
  [view setBackgroundColor:[UIColor blueColor]];

  return view;
}
```

```
hoatyaa = [self getUIView];
[window addSubview:hoatyaa];
[window makeKeyAndVisible];
```

*メモリ的に不味い、って思った人は、Cを知ってるプログラマでしょう。

その通り、メモリ的には不味いです。外部クラスからは怖くて使えない。

Drive23/50

1-5.NSLogの使い方2

さっきさらっと書きましたが、
ログでは、文字が出せる以外にも、数字とかを出せます。

今後必須になるでしょう。

%dで、,の右にある数字を一つ、Logに表示する事が出来ます。

```
int i = [self retInt];
NSLog(@" i をとりもどせ！_%d", i);
```

ちなみに、%dって書いたら数字で、ほかにもいろんなのがあります。

%d　は、digitのdだったと思うのですが、dって書いたのに,の先に
@””とか数字以外の型の内容書くと、エラーで吹っ飛びます。
しかも警告もでません。 わりとよくある事ですw

Drive24/50

1-6-0.IBActionとIBとイベントの話！

メソッド、返り値と来て、思い出してほしい物が一つあります。

```
- (IBAction) なんちゃら;
```

メソッドの、返り値書くとこに IBAction とある。

じゃあ、こいつはIBActionを返すのか？　そういうやつなのか？という話です。

今まで何に使ったかというと、
電卓とかで、ボタンが押されたら動くようにしてましたね。
電卓を思い出しながら調べますかね。

IBActionって何？

結論から言うと、IBActionは返り値ではありません。じゃあ何だ！　> 辞書

Drive25/50

1-6-1.IBActionってば
　ひーてみる。

Constants

`IBAction`

Type qualifier used by Interface Builder to synchronize actions. Use this type as the return type of any action methods defined in your project. For examples of how to use this identifier, see “Xcode Integration”.

Available in iOS 2.0 and later.

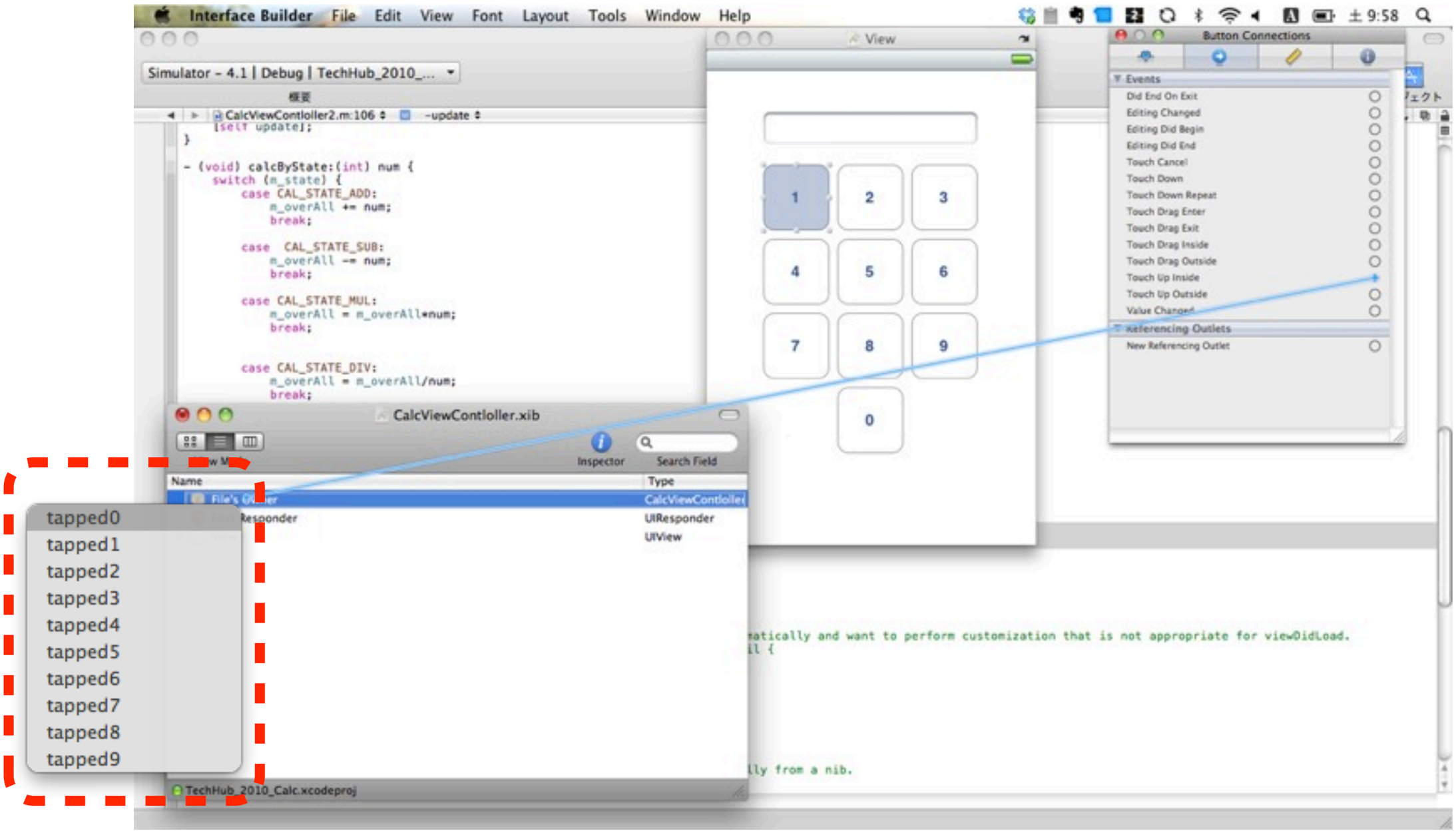
Declared in `UINibDeclarations.h`.

訳：IBを使うときにアクション達をシンクロする為の識別子です、
あなたのプロジェクト内に定義したどんなメソッドの返り値にでも使えます。

→ぶっちゃけると、IBを使うとき、この返り値のメソッドを検出します。
この返り値のメソッドがあると、IBでそのメソッドが表示されるようになるん。
ということは、IBの、アクションがらみなのだ。

Drive26/50

1-6-2.IBでは何に紐づけたか



このときだ。

つまり、`IBAction` を返り値に持つメソッドを設定すると、IBから見える。

Drive27/50

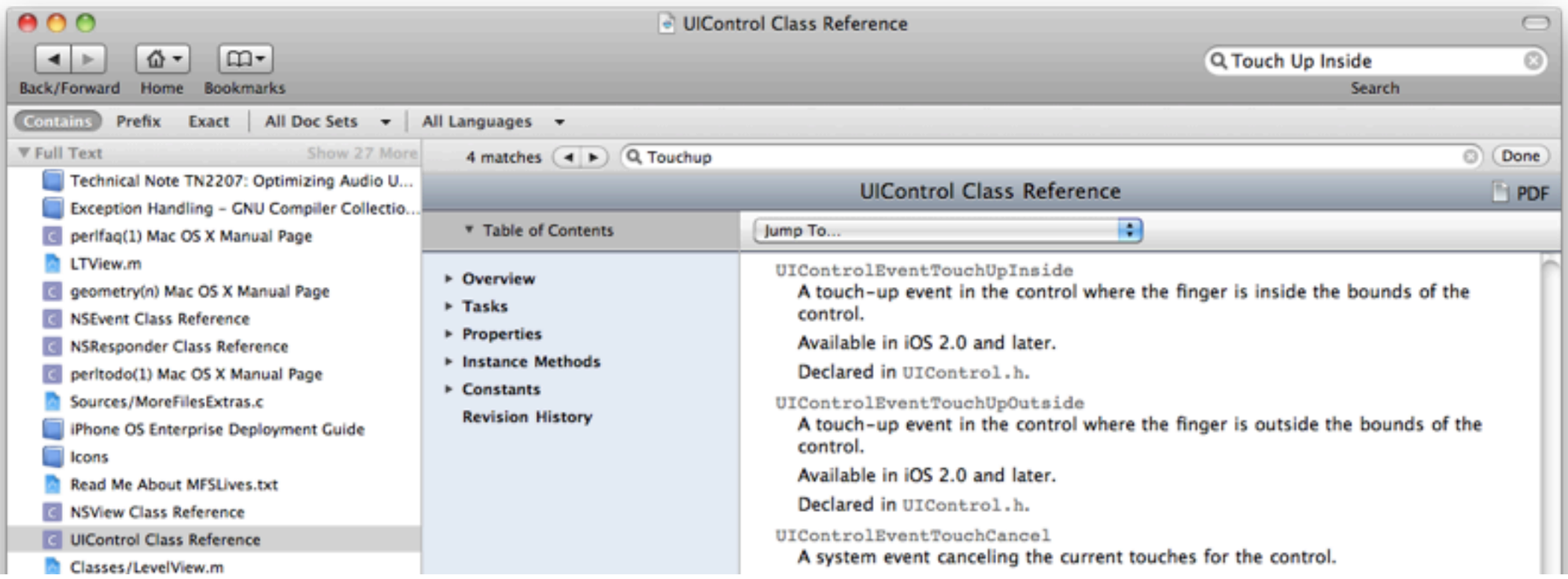
1-6-3.あの日あのときあの場所で

繋いでいるメソッドは、IBActionが返り値に設定してあるメソッド。

で、繋いでる元は、、 `Touch Up Inside` っていう○印。

コレは、、なんだろう。

調べてみよう。



ありましたー。

Drive28/50

1-6-4.調べもの地獄/知れる事が天国
　リファレンスには次のようにある。

　まんまドンピシャでは無いですが、合致する文字の内容を見っけました。
　UIControlねえ。

UIControl Class Reference

`UIControlEventTouchUpInside`
A touch-up event in the control where the finger is inside the bounds of the control.

Available in iOS 2.0 and later.
Declared in `UIControl.h`.

　UIControlEventTouchUpInsideって、
　訳：「コントロールされている領域の内側で指が離れたら発生する
　　touch-up event」なんだってさ。

Drive29/50

1-6-5.終点!!!

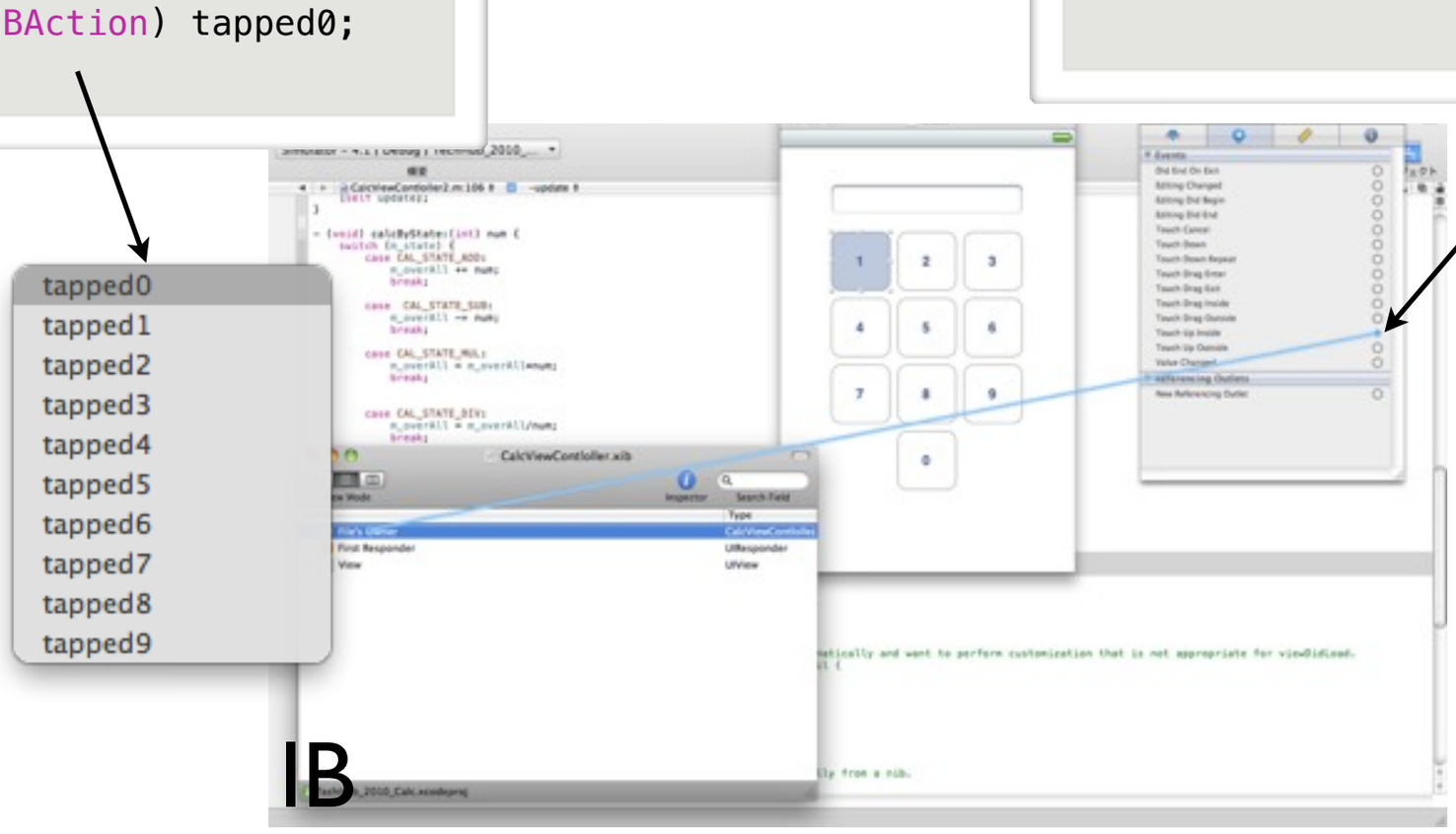
意訳すると、ボタンの範囲の中で指が(タッチしてた状態から)上がると、
このUIControlEventTouchUpInsideってイベントが発生しますよ、と。

コード

```
- (IBAction) tapped0;
```

イベント

```
UIControlEventTouchUpInside
```



これは、`TouchUpInside`が発生したら、`tapped0` メソッドを動かしてね
という意味だったんですね。あーすっきりした。やっと伏線回収した。

Drive30/50

1-6-6.終点!!!、、、ですよ？

この説明をする為には、凄く多くの事の説明、前提が必要でした。

こういう事を書いてくれてる書籍、日本語で無いんだよ。ほんとに。

お陰で理由知らずに暗記だけで乱用する奴らが居て、俺が直すハメになって、

「お前みたいな情弱は大っ嫌いだ！　来世往けバーーカ！！！」ってなる。

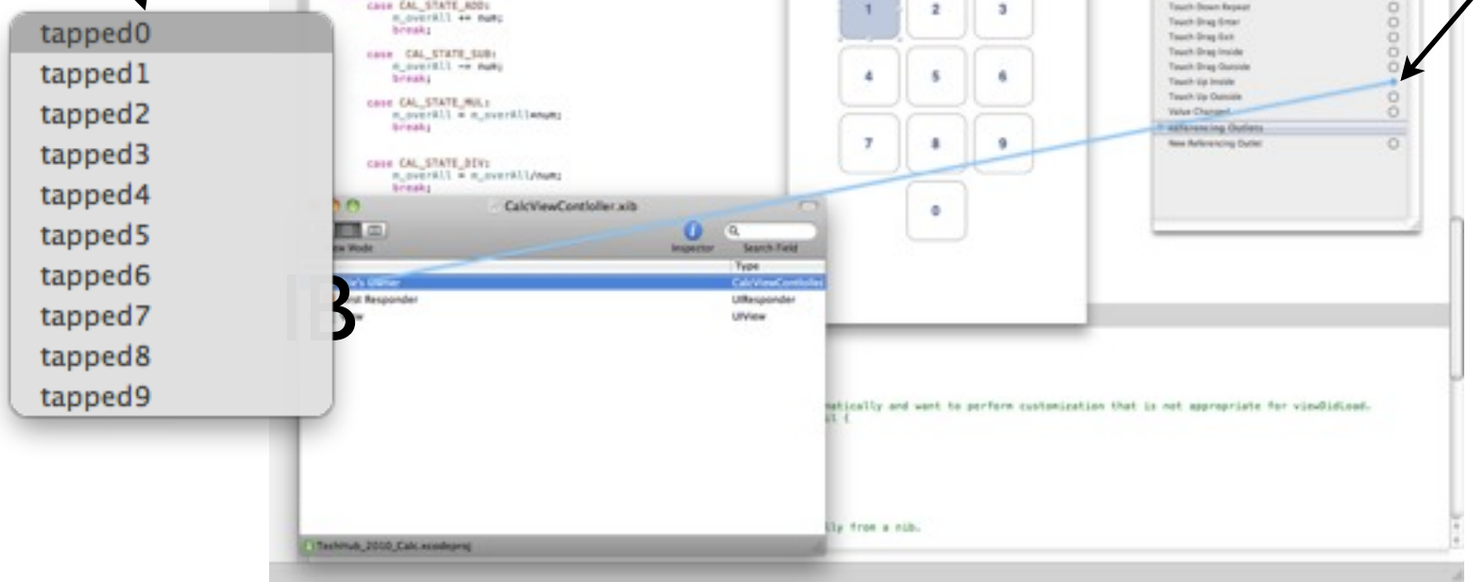
お薦め動画：　　http://bit.ly/cipejW

コード

```
- (IBAction) tapped0;
```

イベント

```
UIControlEventTouchUpInside
```



Drive31/50

1-7-0.命名ルールについて
→名前をつけるのは大変だ

自分で作ったクラスやオブジェクトに、名前を付けるとき、
PrototypeViewController * prototypeViewController;とかやられるとちとウザイっす、という話をしたと思います。まあ正攻法なんだが。

で、過去から今まで、そういうのをまじめに考えた人たちが居まして、

大文字小文字、キャメルケース、他人のソースを見る時の基本に繋がっている命名規則！　なんてものがあるのです。

Drive32/50

1-7-1.詳しくはココを読め

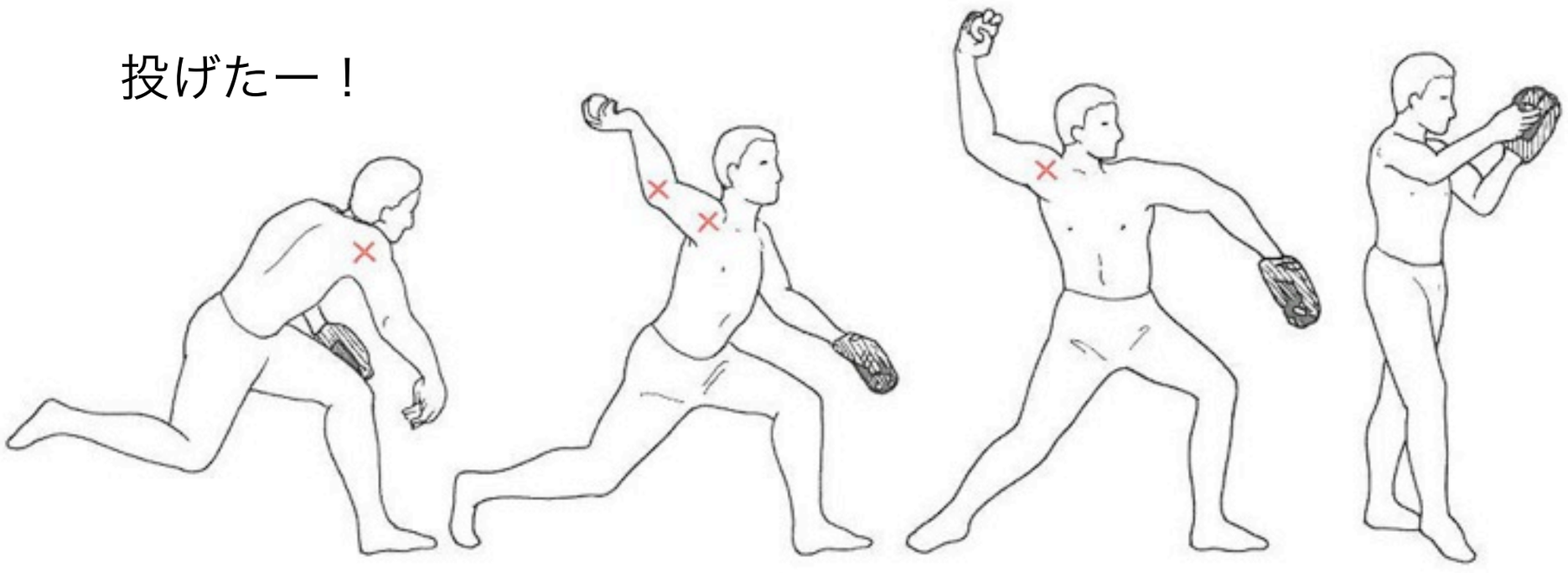
☆Appleのドキュメント

http://developer.apple.com/library/mac/#documentation/Cocoa/Conceptual/CodingGuidelines/Articles/NamingMethods.html

☆Googleのドキュメント

http://www.textdrop.net/google-styleguide-ja/objcguide.xml#命名規則

投げたー！



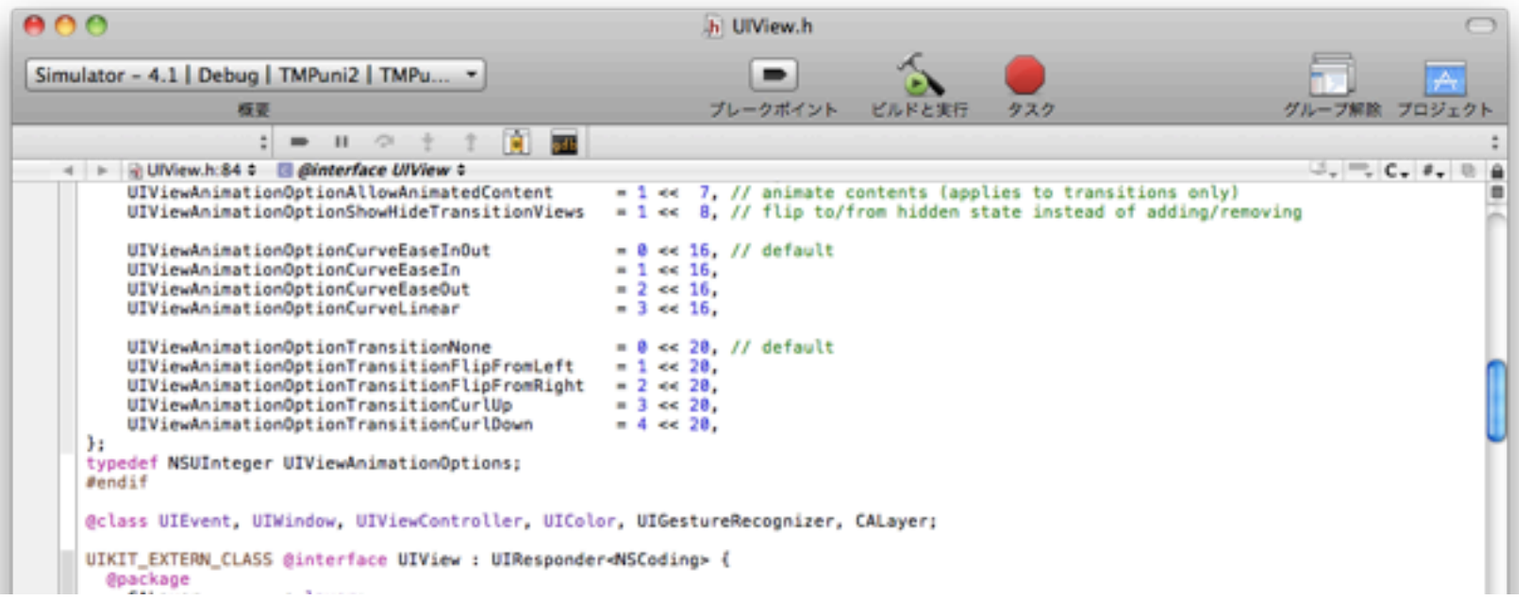
べ、ベストサンプルだもん!

Drive33/50

1-8.メソッドの宣言を見る

Xcodeのプログラムに対して右クリック>定義へジャンプから見れる。

誰が作ったどんなメソッドなのか、実際のコード見れてわかりやすくていい感じになる。



Drive34/50

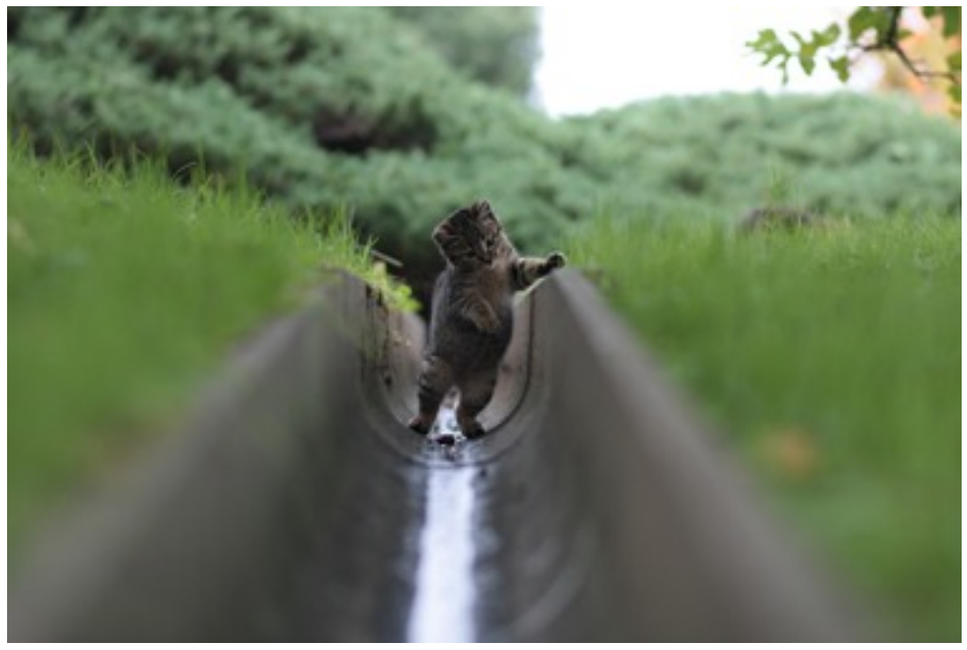
1-9.クラスメソッド

オブジェクトを alloc + init しないでも、
直接型に対して働きかけて(オブジェクトを作らずに)使えるメソッド、
なんてものがあります。

[UIColor purpleColor] とか、その辺ですね。

ま、今のところはそういう名前ですよという所です。
問題が出てくるのが、メモリ周りで困った時とかなので、
放置プレイします。

困ってから考えよう、という罠を仕掛けます。



Drive35/50

2.あなたのアプリケーションの物語

Drive36/50

2-0-0.あなたの人生の物語ていうSFがあって、読んでない。SF(すこしふしぎ)。

皆さんのアプリケーション、作り出しますかね。

Xcodeで、新規プロジェクト→学生番号_アプリ名を書き込んでください。

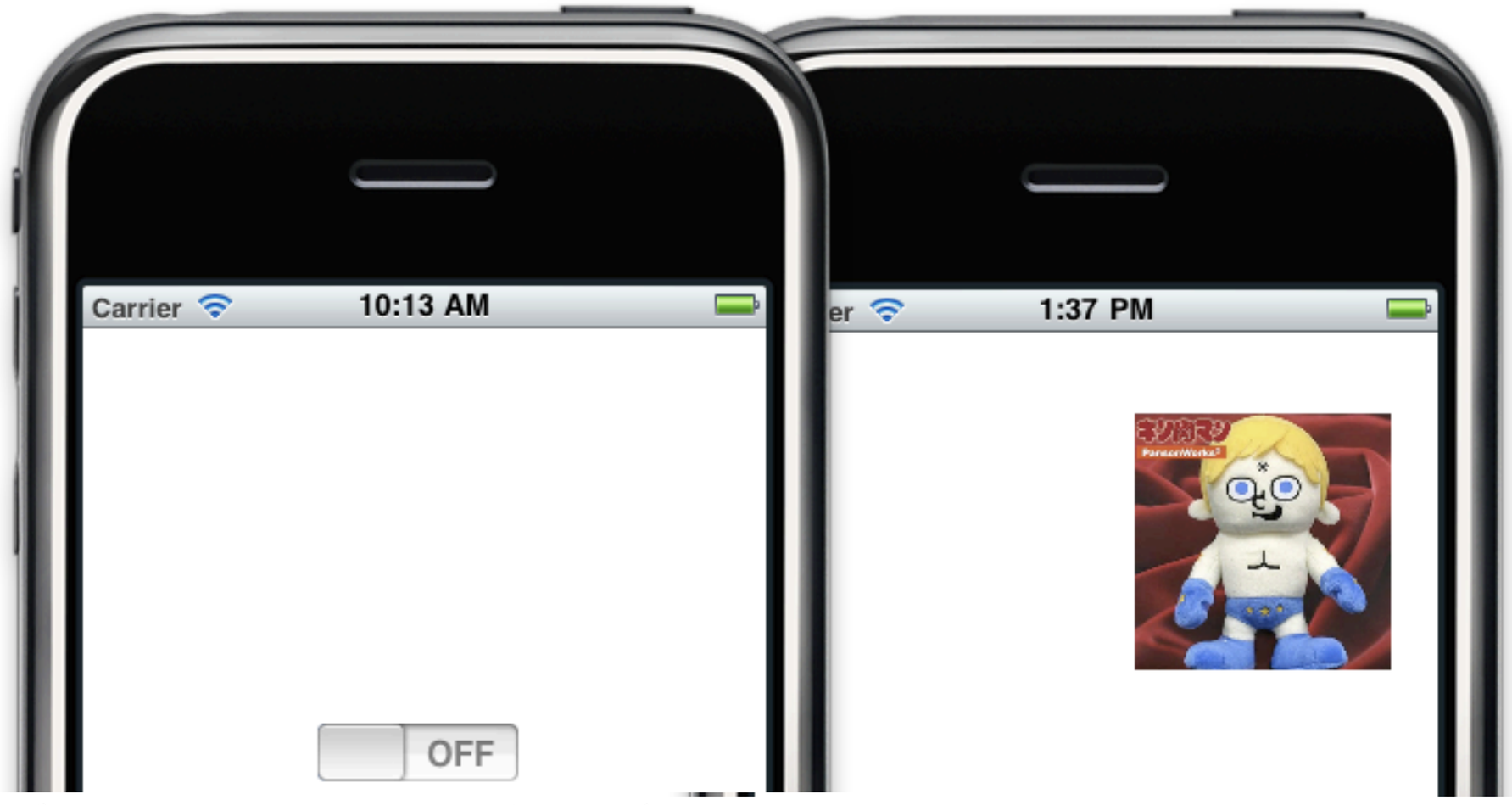
相変わらずWindowBaseで行きますかね。

最初に、アプリケーションが起動したら、まっしろな筈です。

Drive37/50

2-1-0.遷移

ユーザーから見て、遷移、画面の内容が切る変わる事を、
特に画面遷移といいます。
たとえば、画面１、画面２などがあるとき、
これらの切り替えを行う事を、「画面遷移させる」と言います。



まず、一画面でて、その後にボタン押したら次の画面に行く、ってくらい作ってみますかね。

Drive38/50

2-1-1.まずは２つ

スイッチとかボタンとか、押すと交互に切り替わるようにしましょうかね。

・スイッチ/ボタンを押したら次の画面へ

これ、作ってみましょう。

UIViewControllerを２つ、自分のアプリに使うつもりの名前で、
作ってみてください。

タイトル画面と、サブ画面みたいなイメージでも、なんでもOKです。

Drive39/50

2-1-2.AppDelegateに加えます

AppDelegateにさっきの２つのコントローラを加えます。

PW> XXXAppDelegate.h　に#import ~ と、型 * オブジェクトを書く。

```
#import "MyTitleViewController.h"
#import "MyMainViewController.h"
```

```
@interface XXXAppDelegate : NSObject <UIApplicationDelegate>
{
    UIWindow *window;
    XXXTitleViewController * myTitleVCont;
    XXXMainViewController * myMainVCont;
}
```

#importと.hの説明については、次回なんでまあ、わからず書いてください。許せ。

Drive40/50

2-1-3.AppDelegateに加えます

PW>XXXAppDelegate.m　にオブジェクトのalloc init と描画セット書く。

```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

    myTitleVCont = [[XXXTitleViewController alloc] init];
    myMainVCont = [[XXXMainViewController alloc] init];

    [window addSubview:myTitleVCont.view];//タイトルをまずセット！
```

これで、画面に XXXTitleViewController の内容が表示される筈。

次は、ボタンが押されたら動くように、メソッドを作ってスイッチ/ボタンと連動させよう。電卓でやったな。

Drive41/50

2-1-4.スイッチ/ボタンが押されたら

PW>XXXTitleViewController.hとmにメソッド追加、
返り値の型はIBAction。

```
- (IBAction) gotoNext;
```

```
- (IBAction) gotoNext {
}
```

で、スイッチ/ボタンのアクションに紐付けよう。
特にどのアクションだったらどう、とか、つなぎ方とか、説明しない。

辞書で調べりゃ。過去の資料みりゃ。

Drive42/50

2-1-5.スイッチ/ボタンでgotoNextが動くかな？
　で、動く前提で話を続けよう。

　gotoNextが動いたとき、画面が切り替わってほしい。
　ので、gotoNextに、画面が切り替わる処理を書く。

　準備として、まず、
　PW>XXXAppDelegate.hとmに、画面を切り替えるメソッドを書く。

```
- (void) drive;
```

```
- (void) drive {
  [window addSubview:myMainVCont.view];//Mainを表示する
}
```

Drive43/50

2-1-6.けんかをやめて！ ２つの画面を切り替えて！

gotoNextからメソッドdriveが呼べれば、画面が切り替わる。
多分。

ただし、XXXAppDelegateを XXXTitleViewControllerから呼ぶには、一筋縄ではいかない。

なんでかというと、XXXTitleViewControllerがXXXAppDelegateで呼ばれているからだ。　下記ね。

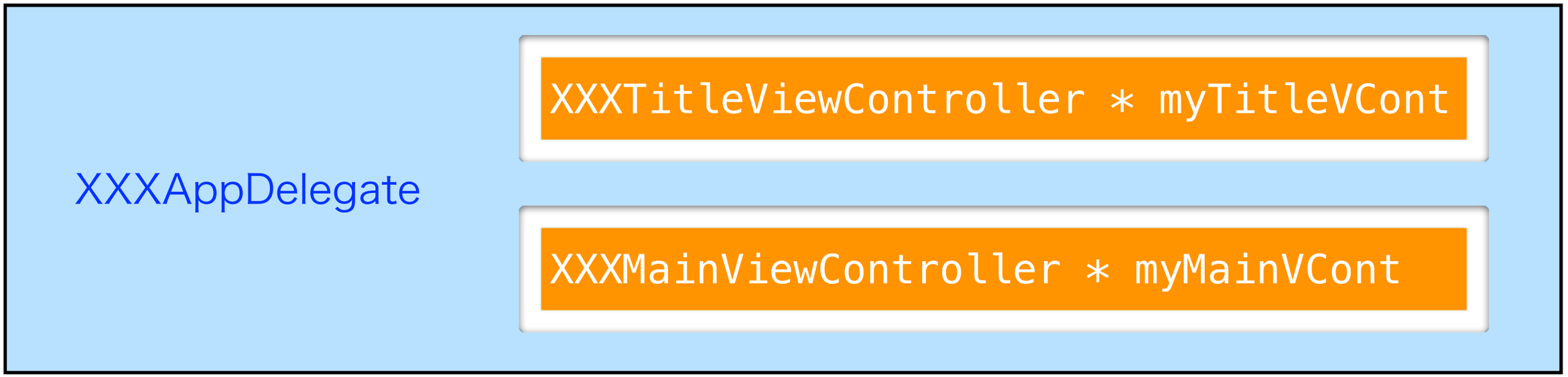
```
- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
{

  myTitleVCont = [[XXXTitleViewController alloc] init];
  myMainVCont = [[XXXMainViewController alloc] init];
```

Drive44/50

2-1-7.自分の髪を持ち上げて自分が浮くかっていう話(画像求む)

こいつは、実はヤベェ。

すでに、myTitleVContとかは、XXXAppDelegateの持ち物なんだわ。



で、 持ち主がもちものをどう扱おうが楽なんだけど、

もちものから持ち主を呼ぶには、ちょっと普通はめんどい手段が必要になる。

なんで？っつうと、今度。

今回のアプリ作りでは、とある手段で狡(ずる)をする。

Drive45/50

2-1-8.AppDelegateは罪な奴

AppDelegate型は、実はもちもの側から呼べる特殊なメソッドを持ってる。
それを使って、もちもの側からdelegateのオブジェクトを呼ぼう。
メソッドgotoNextの中身を書き足す。

```
- (IBAction) gotoNext {
  XXXAppDelegate * app = [[UIApplication sharedApplication]
delegate];
  [app drive];
}
```

合わせて、gotoNextを持ってるクラスの.mに、
XXXAppDelegate.hをimportする。

```
#import "XXXAppDelegate.h"
```

Drive46/50

2-1-9.それだ！

これで起動したら、巧く切り替わるんじゃないかしら。

切り替わ、、った

かな？

Drive47/50

2-2-0.今後の授業の物語

画面遷移が出来たので、まずは自分の作りたいアプリケーションの全体像を作ってみましょう。

今後、
あと一回、座学込みの時間がありますが、
それ以降に明確な座学は、失敗学という学問のやつ以外予定していません。

ほぼ実習メインです。
→この場の意味が変わってきます。

次回~~コアラ~~からこの場は、
「この時間を使ってアプリを作る場」
「この時間を使ってアプリ作りに関して質問する場」
「この時間を使ってアプリ作り後式を共有する場」になっていきます。
あ、別に作ってきちゃって質問でもOKです。

Drive48/50

2-2-1.授業側で把握してるアプリケーションそれぞれの機能、情報

()の中は、全員とある場合全員が必要な機能なので面倒見ます。数字は学生番号。該当番号の人はがんばるのですよフフフ。

アクション系

写真を撮る(5,

BTでの相互通信(1,3,

データ保存(全員)

データロード(全員)

データ入力(全員)

センシング系

加速度センサー(20

画像解析(5

iPhone描画系

iPhone的オブジェクトの描画(全員),,,実は、もう出来るんですよね、、いままで勉強しちゃった。

グラフィックの描画(1,20

インターネットインターフェース系

Twitterへの投稿(12

Twitterへの写真投稿(12

メール送付(11

ネットワークダウンロード(15,17,

ネットワーク向け入力(15,17

ジオロケーション(スタッフ原田

ブラウザ(?

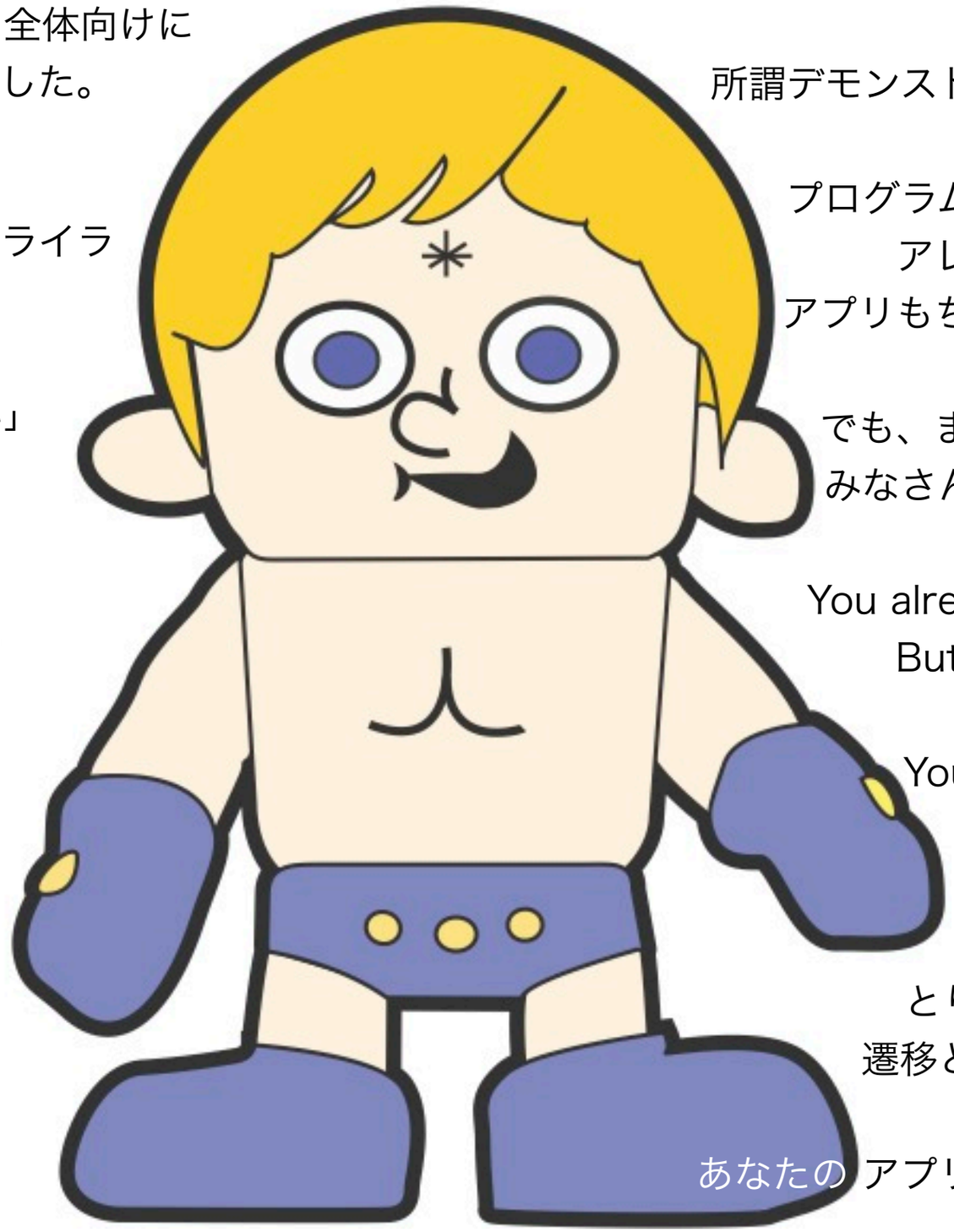
Drive49/50

2-3-0.TMP2 テリーマンプニプニ、みんなの手に負える筈です

ダウンロード場所は、全体向けに
メールで流しておきました。

なんか、、すっげーイライラ
する絵柄なんですが、
ちいさくなるとまあ
普通に「あ、かわいい」
とか思えるのが
またむかつきます。

こいつは、一応既存の
教えた技術+
辞書で調べてわかる
内容で出来てます。
えらく機敏に
動きますが、
OpenGLなど
ハイソな事は全く
していません。



こいつは、
所謂デモンストレーションアプリです。

プログラムの勉強はしてきました。
アレなのやいらない電卓とか
アプリもちょこちょこ作りました。

でも、まだ何をどう出来るのか、
みなさんさっぱりだと思います。

You already know what can do.
But you can't do anything.

You should do something.
Make it! just do it!

とりあえず、やりたい画面を
遷移として作ってみましょう。
ココから先は、
あなたのアプリケーションの物語です。

Drive50/50

2-3-1.遷移をつくったり画面を作ったり、さて、自由な時間ですよ

参考までに、私は今からテリーマンプニプニのコードに手を加えて、
タイトル画面を作ります。

実演、、っつーか、今やんないと時間なさそうなんで。

見たけりゃ見てて、私の操作で~~不自然~~質問とかあったらそのタイミングで
聞いてくれてOKです。

ま、最初のアプリ作りになるので、設計とか気にしすぎずに、
やりたい事をどうすれば出来るか、ズカズカ書いてしまいましょう。

迷ったら、進め、です(お

LookBack

1.プログラミング言語 座学 セミファイナル

2.あなたのアプリケーションの物語

プログラムの質問、提出はこっちな！！
自分の名前と宛名忘れんなよ！！
↓

techhubteach@googlegroups.com

Continue?

次回、正直進むペースはお任せですが、

全てのアプリで必要な機能として、セーブとロードを行う部分を作ります。
あと、第一回にやったgitを使って、アプリケーション自体のセーブも行います。

遷移は各人で好きに作っちゃってー。

次回、

第5回

「アプリ作りと永久回廊(1)と」

DriveEX/EX

EX.　　、、、で、、、、終わると思ったんだよ！

ただ、この会場を提供してくださっているECナビさんのご厚意により、
この場所、使用時間の期限が16時までじゃ無いんだわ。

で、18時からイベントとかあるんだってさ。
俺はよく知らないけど。

という訳で、テリーマンぷにぷにの解説とか、
gitでセーブする為の資料作りとか、
ライブコーディング(いろいろつくるのの実演)とかします！

うわーーい。　仕事ーw