第2回
手書きとルールと

Toru.Inoue@KISSAKI.tv

Agenda1/1

- 今日やる事
  - 0.籤
  - 1. Objective-Cについて
  - 2.簡単♡プログラム
  - 3.プロクラマの共通言語
  - 4.VS IB ～IBとMVCの関連性～
  - 5. アプリを作ろう2

Drive1/50

1.Objective-Cについて

Drive2/50

1-0:Objective-Cについて教える事

この授業では、
Objective-Cというプログラミング言語がどんな物なのか、については、
さらっとしか触れません。

主に勉強の仕方を伝えます。

→言語が知りたい人は本買え
詳説Objectiv-C
(とはいえC言語などの知識がいるので、先にCを学んでおく事をおすすめします。)

→私に教わりたい人
KISSAKI私塾に来い。
現在の生徒:あいつ→
今日持ってきてるMacがあれば、そのまま教えられます。通信教育。Skype。

Drive3/50

1-1:法則と内容

プログラムって何?→

人間が作った、「なんか文字を書いて保存してエイヤってやると何かが起こる」
仕組みです。

当然人間が作った物なので、凄くショウモナイです。この程度か、と思うくらいには。

或るところに、プログラムが有りました。
ごく普通に本を読んでごく普通に手でキーを打つだけで、ごく普通に動くのでした。
でも、普通の仕事と違うところが、、、

ただ文字を書くだけで、絶対にお金が儲かるのですw

、、まあ嘘ですけど、スキルとして食いっぱぐれないのは本当です。
そして、物事の考え方の基盤としても、非常に強力です。
だってお金回りのシステムってプログラムでつくられてるんだもん。

Drive4/50

1-2:プログラムという学問

プログラムの勉強の仕方→

辞書を一冊買って、やりたい事を牽きまくる
→前回紹介したものが非常に有用です。

知りたい事しか知れませんが、
知りたい事しか覚えないでしょ。

それ以外にも、Xcodeはいろいろ知っている。
→調べ方
Xcodeから、右クリックで辞書を牽いてみる（90%解決します）

前回作ったPrototype
→shouldAutorotateToInterfaceOrientationなどを選択して、
右クリック
→テキストを製品ドキュメント内で検索

→出てこない人は、ネットに繋げておこう。
Xcodeが辞書を自動でDLしてくれる。

これだけで、いろいろわかります。やってみよう。

Drive5/50

1-3:プログラムが上達するコツ

自分が理解できないコードを一切書かない、と上達しますが、
凄い時間がかかります。

基本はXcodeの辞書とAppleのサンプルと、Google先生です。

ただ、自分の書き加えた/コピペしたコードが一体なにをどうしているのか、
その内容、意図、仕組みの仮定はできるようにしておきましょう。

自分で、「きっとこのコードはこんな事をしてるんだろ？」と推論できるようになっていれば、
余裕が出たとき調べて判るようになり、かなり学習が速くなります。

推論するとき、「アレと似てる」とか、
類推するのがお薦めです。

今まで2年くらいでプログラム極めた感想としては、全体のうち95%くらいは相互に相似なんですけどね。
1を知れば、100個は類似/相似の物事があります。 結局人間の作った物なので。

Drive6/50

1-4:プログラムに向き、不向き

主観からいうと、

- ・文字が読める
- ・文字が意味する内容が単語レベルで判る
- ・他人と何らかの方法でコミュニケーションできる

これらが満たせていれば、プログラマになれます。

他の人にとって、頼り頼られるいいプログラマになるには、

- ・なぜ？という疑問を持ち続ける姿勢
- ・もっと楽に、いままで以上の事が出来る筈、という探究心
- ・人になにかを教える事が出来る程度の論理性

(さすがに、「大宇宙の真理が、、」とか、「ガイアが俺にもっと輝けと、、」とか言い出す人はパスです。)

これらがあれば、十分だと考えています。
さらに有ると嬉しいのは、

- ・未来、自分か、あるいは記憶喪失の自分か、誰かが
  このコードに何か機能を足す時、足しやすく作る心配り

これが、あると、井上は「是非ウチに来てください」、って言います。 あ、Perlなら結構です。

Drive7/50

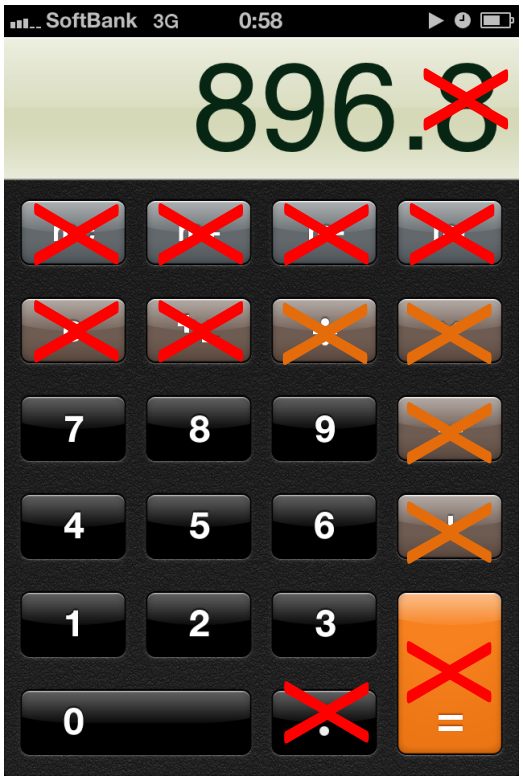
2.簡単♡プログラム

Drive8/50

2-0:先ずは前提を全く有せず、アプリを作ってみる

- 一桁の足し算だけが出来る電卓～（昔のドラえもんの口調で）

数字が振られたボタンを押すと、足された値が表示されるッ!!



ゴゴゴゴゴ

Drive9/50

2-1-1:レッツ写経 or ファイト！

計算機（易）の作り方

1.Windowbasedのプロジェクト(Caluculator)を作って、
ビューコントローラ(CalcViewController)を、
CalcViewController.xib付きで作成。

ビューを画面に表示するとこまでいきましょう。

ヒント→[window addSubView:calcViewController.view];

→前回の授業の復習です。

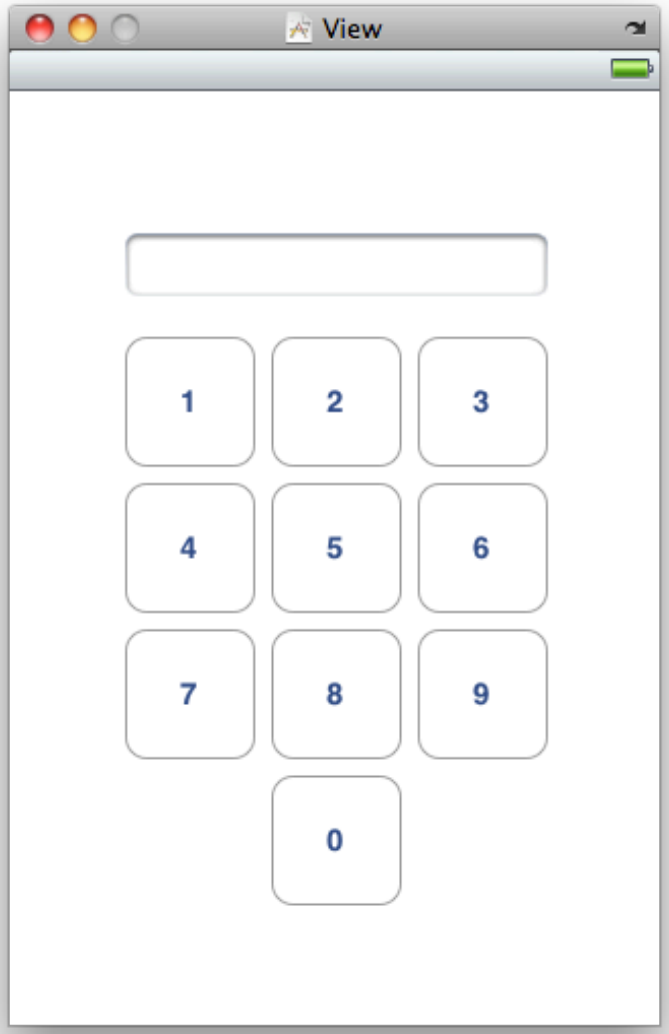
Drive10/50

2-1-2:レッツ写経 or ファイト！

計算機（易）の作り方

2.IBをつかって、
ビューに0~9までのUIButtonと、
計算結果が表示されるUITextFieldを
置きます。

→前回の逆襲みたいな感じ。



Drive11/50

2-1-3:レッツ写経 or ファイト！

計算機（易）の作り方

3. PW>Classes>CalcViewController.h

ビューコントローラに、0～9までのボタンが押された時のIBActionを書きます。

```
@interface CalcViewContloller : UIViewController
{
	IBOutlet UITextField * m_field;int m_sum;
}
- (IBAction) tapped0;
- (IBAction) tapped1;
- (IBAction) tapped2;
- (IBAction) tapped3;

/*・・・こんなのが0~9まで或る筈さ！　ここではめんどくさいから3までしか書かないけど、
	0~9まで書いてね！　あ、この水色はコメントなので書かないでいいYO!　*/
```

Drive12/50

2-1-4:レッツ写経 or ファイト！

計算機（易）の作り方

4. PW>Classes>CalcViewController.m

IBActionをボタン分書こう。

```
@implementation CalculatorViewContloller
- (IBAction) tapped0 {}

- (IBAction) tapped1 {}
//・・・前のページと一緒、0~9まで書いてね。
```

Drive13/50

2-1-5:レッツ写経 or ファイト！

計算機（易）の作り方

5. PW>Classes>CalcViewController.xib
ボタンが押されたら、CalcViewController.mに書いたプログラムの

- (IBAction) tapped0とか

が実行されるように、IBで設定しましょう。

前回こんな感じに作ったのは、「テキストフィールドでEnterを押したらキーボードが引っ込む」ヤツでしたね。

→どんなことしたんでしたっけ。

今度は、ボタン一個一個に対して、「ボタンを押したらtapped0とかが起こる」アクションを設定です。

ぶっちゃけるとIBを使って、0~9のボタンから、
File's Ownerまで合計10個、線を手で繋ぐんだぜ！ めんどくせぇ！

Drive14/50

2-1-6:レッツ写経 or ファイト！

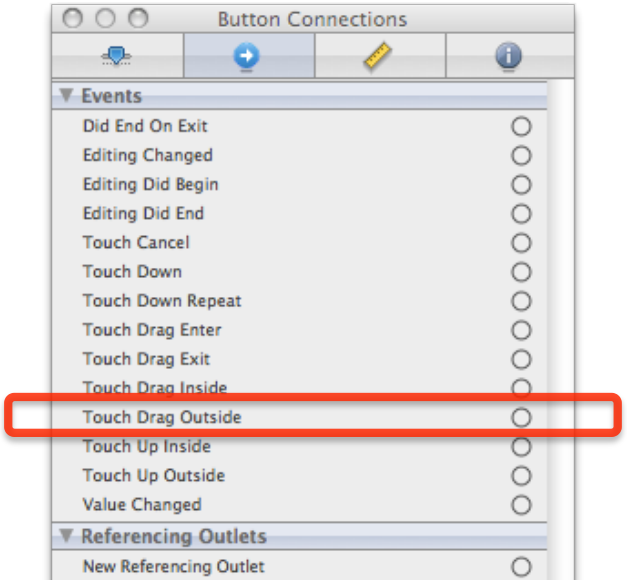
計算機（易）の作り方

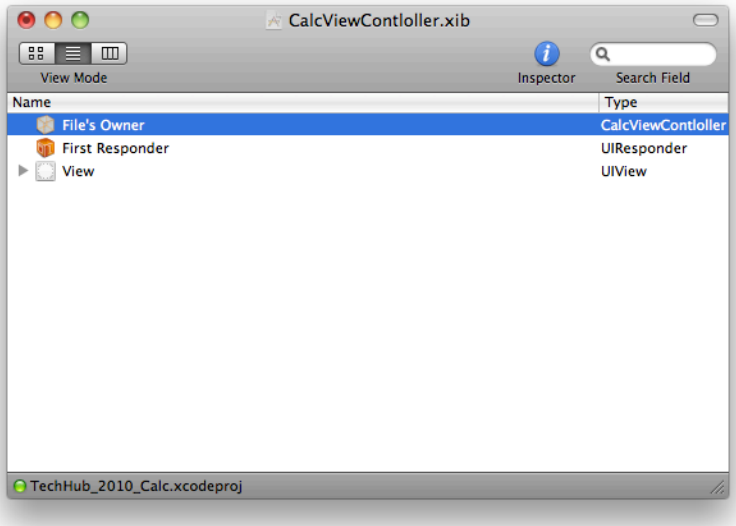
6. PW>Classes>CalcViewController.xibの設定の内訳

IBで、0~9のボタンのアクション「Touch Up Inside」とFile's Ownerを繋ぎます。

「Touch Up Inside」からFile's Ownerまで線を引っ張ったら、

Tapped0~9が出たかな?



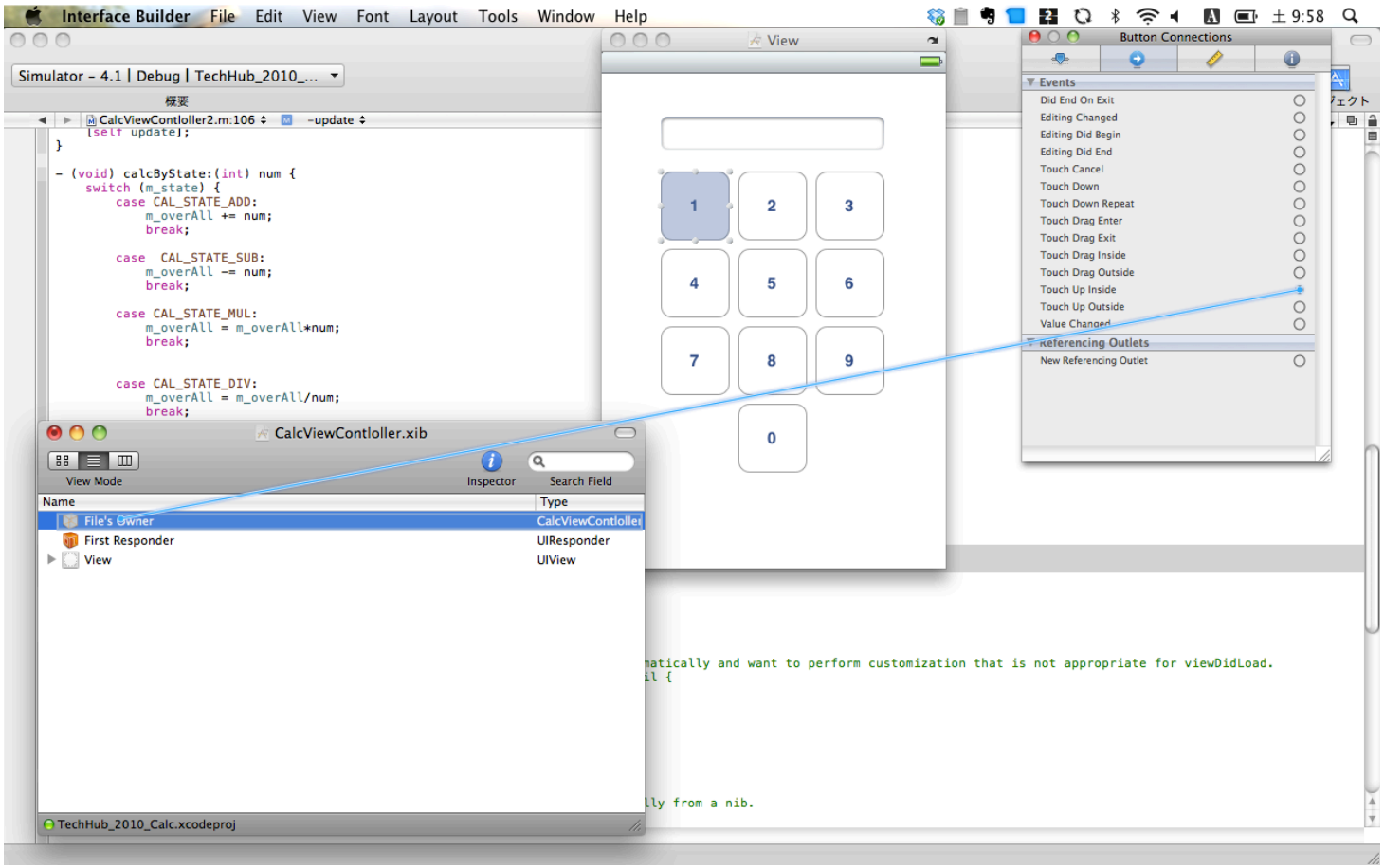


Drive15/50

2-1-7:レッツ写経 or ファイト！

計算機（易）の作り方

7.補足



←この辺からホラ、
アレだよ、アレ、
ソレがこーなって、、

あのアレをほら、
判るだろ？　ナァ？
アレだよ!

(昔、知人→井上で
交わされた会話より抜粋)

Drive16/50

2-1-8:レッツ写経 or ファイト！

計算機（易）の作り方

8.テキストフィールドに計算結果を出したいのだよ！　準備

PW>Classes>CalcViewController.h

テキストフィールドに書かれた文字列を更新できるように、
オブジェクトを2つ書き足す。

```
@interface CalcViewContloller : UIViewController {
    IBOutlet UITextField * m_field;
    int m_sum;
}
```

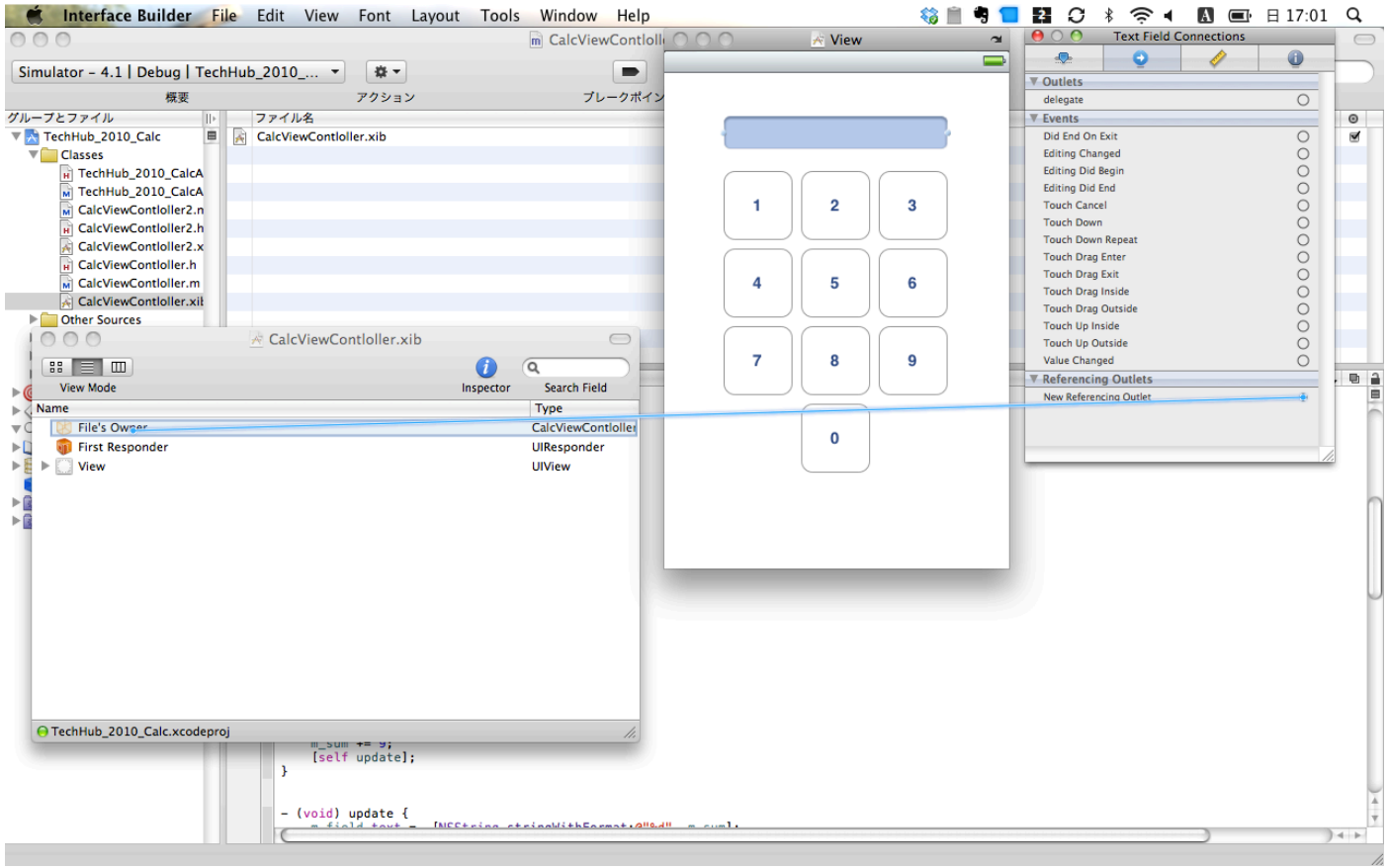
次は、ここで書いたm_fieldを、IでB設定する

Drive17/50

2-1-9:レッツ写経 or ファイト！

計算機（易）の作り方

9. m_fieldを、IBで設定する



Drive18/50

2-1-10:レッツ写経 or ファイト！

計算機（易）の作り方

10.テキストフィールドに計算結果を出したいのだよ！ 仕上げ

PW>Classes>CalcViewController.h

テキストフィールドに書かれた文字列を更新できるように、メソッドを書き足す。

```
-(void) update;
@end
```

PW>Classes>CalcViewController.m

テキストフィールドに書かれた文字列を更新できるように、メソッドを書き足す。

```
-(void) update {
    m_field.text =  [NSString stringWithFormat:@"%d", m_sum];
}
@end
```

Drive19/50

2-1-8:レッツ写経 or ファイト！

計算機（易）の作り方

9.ボタンが押された時の挙動書こうぜ！！！

PW>Classes>CalcViewController.m

数字ボタンが押されたら、数字の分だけ、数字が＋、表示される処理を書こう。

```
- (IBAction) tapped0 {
        m_sum += 0;
        [self update];
}

- (IBAction) tapped1 {
        m_sum += 1;
        [self update];
}
//・・・
```

緑色の字の部分は、0～9まであるはず。 なので、それぞれ数字にあわせて書き換えましょう。

Drive20/50

2-2:ファイト完了

計算機（易）の作り方

→できたかな？　すっげーー端折ったんだけど。

まあみんな、勉強のプロだから大丈夫だよね。

→わかんなかったら前回の講義の資料観てね。

余談だけれど、より頭のいい人間が多い大学に入るために必要なのは、

「知識を、如何に短時間で楽して大量かつ正確緻密に理解吸収するか」

だと思ってます。　そういう意味でやはりクーガーの兄貴はエラい。速さが足りないッ！

受験勉強(て何)的な考え方や効率が、

プログラムの勉強にも応用できるので、ファイト。

類推と因果を探る、コレだけですが。

Drive21/50

3.プログラマの共通言語

Drive22/50

3-0:共通言語としてのMVC(エムブイシー)

→プログラマの仕事の大半は、だれかと一緒にやるもの

→プログラマの仕事内容ってなに？

~~2ch~~

~~ネットサーフィン~~

~~無駄飯喰らい~~

プログラミング！　ではなく、

プログラムを通じて生産活動を行う事

Drive23/50

3-0:共通言語としてのMVC(エムブイシー)

→MVCとは、

「アプリケーションを作る時の考え方のレール」みたいなもので、
プログラムを「M」と「V」と「C」という3つの属性に分解して考えます。

判りやすい順に説明すると、

V:View:ビュー。
　　見た目のパーツです。IBで弄ったやつですね。

C:Controller:コントローラ。
　　他の要素を操り、コントロールするパーツです。

M:Model:モデル。
　　主にデータ。数値とかです。ビューに表示したりします。

頭文字を合わせて、MVCとか言います。

Drive24/50

3-1:例題の分解

→さっきの電卓を例に取ってみましょう。

見た目：ビュー：View
→IBでつくった所、ボタンとか画面とか。

計算するところ：コントローラ：Controller
→　~Controller.hと.mで書いたところ

計算結果を格納するところ：Model：モデル
→~Controller.hと.mの中の、m_sum

モデルとコントローラの区別がアレですが、
V:IBでしか触ってない
CとM:プログラムで書いてる

と言えると思います。

Drive25/50

3-1:例題の分解

スゲーざっくり書くとさっきみたいな感じです。

実はもっと明確に、ファイル(.hとか。Mとか)単位とかで分けられるんですが、、
クラスとか説明するのがめんどいのでパス。

・分ける内容がM,V,Cの三つあること
・それぞれ用途や内容で分けられるということ
・プログラムの内容要素は、M,V,Cで分けられる

という三点を、覚えておいてください。

Drive26/50

3-2:あらゆる物には意図が或る、理由が或る

なんでこんなMVCなんていう分け方があるの？

→2つ、説明できる理由があります。（と井上は思ってます。）

一つは、

「算数の公式のように、便利だから」。

当てはまる例が多いんです。

もう一つは、

「プログラマ間の共通のルールに使えるくらい普及しているから」。

Drive27/50

3-3:プログラマの作業は、
（というか人間社会で仕事と言えば）
チームワーク

チームワーク：団結力とかそういう意味のパラメータ(日本語)ではなく、
チームワーク：複数人で一つの事にあたる事(英語)で、

複数人で一つの事をするには、なんらか仕組み(Schema)が必要になります。

共通言語、共通の常識、共通の上司、共通の設計図、共通の連絡手段、etc,,

ここで必要なのが、まあ、決まり事とかお約束とか、そういうもの。
MVCも、そんな話の中の一つ。

Drive28/50

3-3:プログラマの作業は、
（というか人間社会で仕事と言えば）
チームワーク

MとVとCに分けると、次の得が有るんです。

- アプリの設計に共通点が出る
  同じルールで分解したら、そりゃ分解単位が一緒なんだから、
  要素が似てきますわな。

- 共通点があれば、再利用や、応用ができる
  再利用、応用ができれば、プログラムが楽になります。

- 同じ土壌で他のプログラマと話がしやすい
  共通の認識に近い筈、、なので、的外れな議論や合わない前提が減ります。
  もちろん、概念が一致しているかの確認を怠ってはいけませんが。

Drive29/50

3-3:プログラマの作業は、
（というか人間社会で仕事と言えば）
チームワーク

FAQ:何でもかんでも分解できるん？

→分解できるというより、分解できるように考える事で分解する、
という表現かも。

例えば:
目覚まし時計
- 時計の針とか数字:View
- 時間を数える機能:Controller
- 今何時か覚えておくところ:Modelとか。

この辺りは、使うプログラム言語の仕様と密接に関係してくるため、必ずしも、
という答えはありません。
MVCは、あくまで「考え方のテンプレート」と、「一定の常識」として使えます。

さっきの電卓と掛け合わせるのも、MVCで分かれてれば可能かもね。

Drive30/50

4.VS IB　～IBとMVCの関連性～

Drive31/50

4-0:で、MVCとIBとの関係は？

ズバリ、VはIBで作れます。

目に見える部分は、IBでガシガシ組むと、すげえ気持ちいいです。

ただ、それ以外の物事、C、Mにも、かなりIBから干渉できてしまう事が有ります。

→表示はするけど入力を受け付けないようにする、とか、
→コントローラをIBから作れる(!)、とか

ってやられるとね！　もうね、判らんのよ！

時間が余ったら、地獄を見せましょう。
この辺りは、IBで設定するのは避けたほうが良さげです。
C側で設定しましょう。

Drive32/50

4-1:IBの反例、、(時間があったら)

→コントローラをIBから無尽蔵に作れちゃう！

実は、素でxibからオブジェクトを作れるんだぜ。ただし、、

→どうなっているのか、よく見ないとわからん
オブジェクトにカーソル合わせて、オブジェクトがどんなものを使っているか、
手でいちいち調べないと判らない！

→いっぱいあったらもう判らないw

→IBなんだから、インターフェースしか作っちゃ駄目だろw
ということで、V以外を作ってはいけません。
かつ、Cから複数のVならいいんですが、Vから複数のCとか、作ってはいけません。

Drive33/50

4-2:IBの使いどころ

→考え方の指標としては、

見た目にしか関わらないところ:

V確定なのでIBで作成

見た目以外に、なんか算数とか表示したりしなかったりが必要なところ:

Cで頑張る

データとか、アプリが動いてる間保存しとかなければいけない部分:

Mで固める

Cを基準にVを考えると、楽になります。
見た目から入れる、【IBから作成】の方が楽だけど、
ぐっちゃぐちゃになって飽きて終わるのがオチです。

→生兵法は怪我の元、IBはVの元。

Drive34/50

4-3:残念な話(時間があったら)

→xibファイルの使い方をここまで説明するのは、ココと海外サイトだけだと思います。

日本の書籍にして30冊、サイトにして200程度をずら一っと観た井上が言うので、
まあ、適当に信じたり信じなかったりしていただいて結構ですが、

はっきり言うと日本語のモノは6割以上が痛い内容です。
応用の利かない一時しのぎばっかり。拡張しにくいのばっかり。
バグでたら直しにくいのばっかり。

というのも、根本を説明しないままIBとか使ってるからなんですが。

IBだって、まともに再現したり利用したり出来ないんだぜあいつらw
という残念な感じです。

もうね！　みんな　英語読めよ！　日本語とか要らないyone!

Drive35/50

5.アプリを作ろう2

If (30 < time)

Drive36/50

5-0:どれにしようかな

電卓(笑)
ストップウォッチ
お絵描きアプリ
目覚まし時計
メモ帳
マインドマップ

の中から、どれか一つ！、、、、

電卓(笑)っすかね、、さっきのサンプルに書き加えて、四則演算やてみよう！

Drive37/50

5-1:とりあえず考えようか。

電卓(笑)

カッコ笑いの名に恥じないように、
「とりあえず四則演算は出来るんだけど、、、」
というところを目指そう。

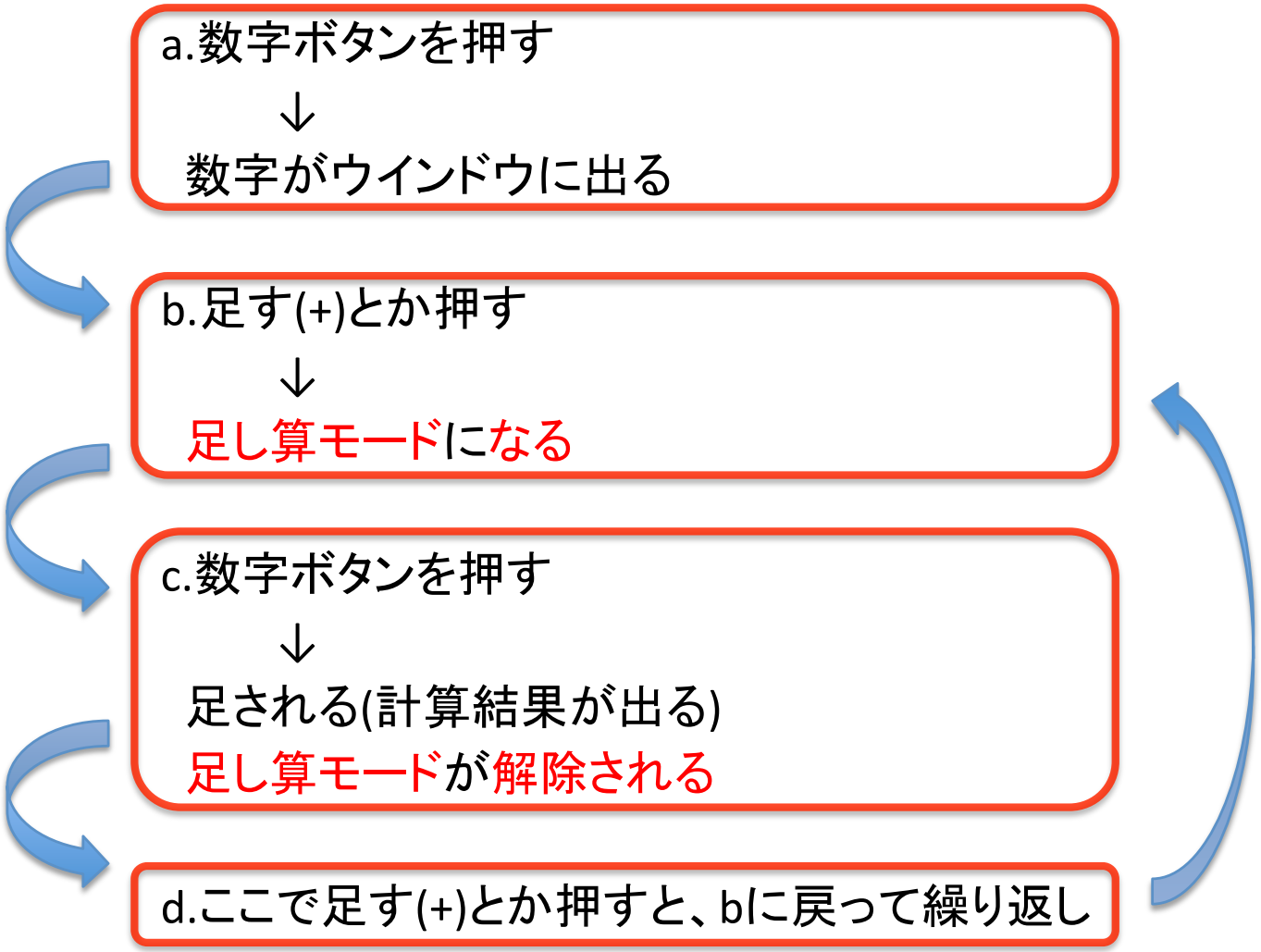
足す、引く、割る、掛ける
　→さっきは数字ボタンを押したら、
押した数字が足されるだけ、だった。

じゃあ、それぞれのボタンが押されたら、
どうしようか。

Drive38/50

5-2:で、どうなるとどうなるんだか考えよう

とりあえず考えてみた。こんな感じでどうだろう。



Drive39/

5-3-0:実際にどこに何を書くか、考えよう

が押されたら足すモードになって、
次に押された数字を足す、とかでどうだろう。

ということは、
四則演算のボタンを押すと、
+-x÷のどれかのモードになる、というわけだ。

→四則演算ボタンを押すと、モードが変化する
　→モードが4つと、あと起動した時のモード
　（まだどのモードでもない！）の、計5つのモードが必要。
　1.→モードを5つ作ろう！

→ボタンが押されたら、モード変化が起こるように、
　ボタンが押されたら発生する処理が必要。
　2.→四則演算各ボタン用のIBActionを書こう！
　3.→アクションが起動したら、モードが変わるようにしよう！

Drive40/50

5-3-1:モードを5つ作ろう

1. PW>Classes>CalcViewController.m

5つ、モードを書き足そう。こんな風に書ける。

```
#import "CalcViewContloller.h”

#define CAL_STATE_NULL      (-1)
#define CAL_STATE_ADD       (0)
#define CAL_STATE_SUB       (1)
#define CAL_STATE_MUL       (2)
#define CAL_STATE_DIV       (3)

@implementation CalcViewContloller
```

Drive41/50

5-3-2:四則演算各ボタン用のIBActionを書こう

PW>Classes>CalcViewController.h

四則演算用に、メソッドを書き足す。

```
@interface CalcViewContloller : UIViewController {
        IBOutlet UITextField * m_field;
        int m_sum;//消します！
}

- (IBAction) plusTapped ;
- (IBAction) minusTapped ;
- (IBAction) mulTapped;
- (IBAction) divTapped;
```

Drive42/50

5-3-3:四則演算各ボタン用のIBActionを書こう

PW>Classes>CalcViewController.m

四則演算用に、メソッドを書き足す。

```
@implementation CalcViewContloller
- (IBAction) plusTapped {
        m_state = CAL_STATE_ADD;
}
(IBAction) minusTapped {
        m_state = CAL_STATE_SUB;
}
//あと2つ、想像してみよう。
```

Drive43/50

5-3-4:四則演算各ボタン用のIBActionを書こう

PW>Classes>CalcViewController.h

四則演算用に、メソッドを書き足す。

```
@interface CalcViewContloller : UIViewController {
        IBOutlet UITextField * m_field;
        int m_state;
        int m_overAll;
}

- (void) calcByState:(int) num;
```

Drive44/50

5-3-5:四則演算各ボタン用のIBActionを書こう

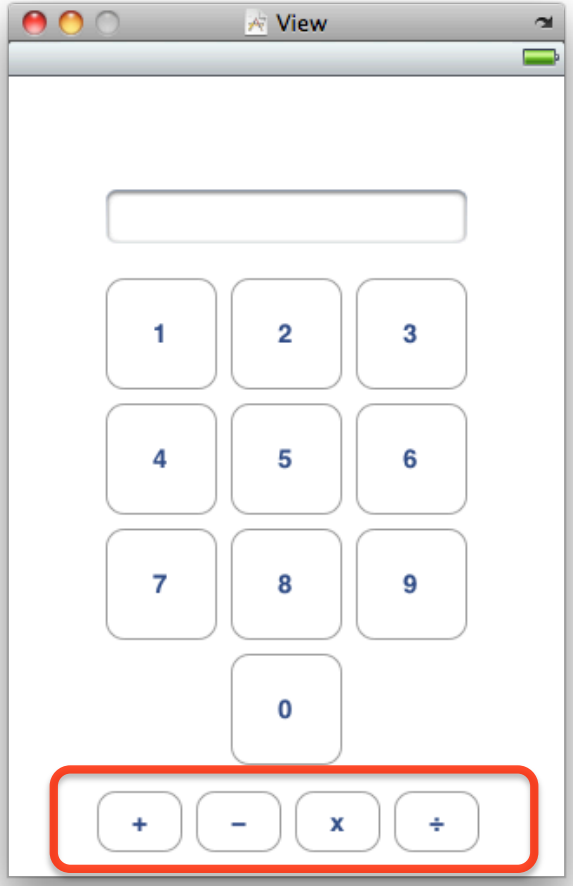
PW>Classes>CalcViewController.m　　四則演算用に、メソッドを書き足す。

```
-(void) calcByState:(int) num {
switch (m_state) {
        case CAL_STATE_ADD:
                m_overAll += num;
                break;
        case  CAL_STATE_SUB:
                m_overAll -= num;
                break;
        case CAL_STATE_MUL:
                m_overAll = m_overAll*num;
                break;
        case CAL_STATE_DIV:
                m_overAll = m_overAll/num;
                break;
        default:
        break;
        }
        m_state = CAL_STATE_NULL;
}
```

Drive45/50

5-3-6:とりあえずボタンを足そう。

電卓(笑)、まず、インターフェースに、四則演算の+-X÷を足してみるかな。



うはwww　おk wwww

Drive46/50

5-3-7:とりあえずボタンを足そう。

インターフェースと繋いでしまおう！　数字の時と一緒

IBが何をやっているのか、などは、、実は、ほとんどやってる事に
種類がないのが判ってきたと思います。

さて、なにしてると思います？

Drive47/50

5-3-8:四則演算各ボタン用のIBActionを書こう

PW>Classes>CalcViewController.m

四則演算用に、メソッドを書き足す。

```
-(IBAction) tapped0 {
        [self calcByState:0];
        [self update];
}

/** 0~9まで、全部に書こう！　緑の部分は、、どうなると思う？　*/
```

メソッドを繋ぎ終わったら、完成！のはず。

Drive48/50

5-4:【人生に深刻なエラーが発生しますた】

・オーブストラクショーン。これだと、一桁の計算しか出来ない。
　→２、３、４、、桁の計算をするには、どうすればいいでしょう。

・「OOPS!」『どうしたんだGreg！』
　「聞いてくれよBobby、1を2で割ったら0になっちまった！
　どうかしてるのはこの電卓か、俺の頭なのか？」
　『HAHAHA、両方、ってのは、どうだ？(DOYAッ)』

　→そうですね。
　　→正直、プログラマのための勉強が、必要になってきます。

・今、どの四則計算が入ってるのか判りづらいっす！
　→そうですね。
　　→アイデア求む

Drive49/50

5-5:、、、で、果て

、、、で、これらの変更点を加えると、、、、まあ、その、いろいろした結果、

電卓（普）とかいう事になる訳です。

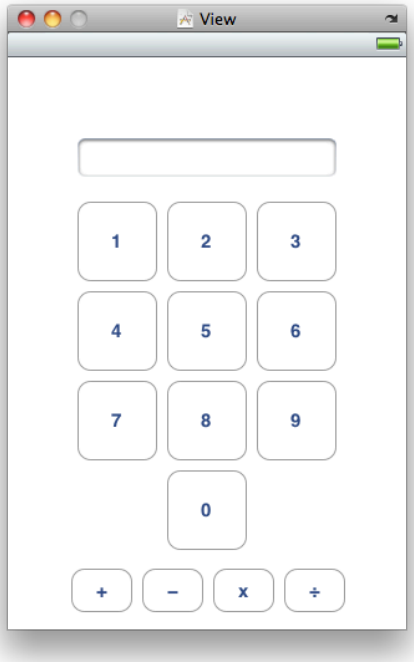
、、、なるんだよ！　果てまでいけば！

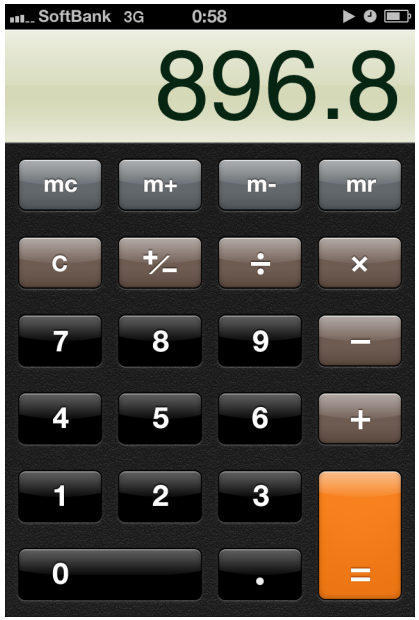
Drive50/50

5-6:とりあえずアプリはこんな感じ。

四則計算以外にも、対数とかSin/Cosとかいろいろ有るけど、俺使った事ないや。
入れたかったら入れてみてください。次回までに関数電卓機能とか入れて、
井上宛にソースコード送ったら、ウチでインターンさせてあげます。

むしろ案件何個かあげます。小遣い稼げば？





イスカンダルよりは、近い筈。

LookBack

0.籤引いた
1. Objective-Cについて
2.簡単♡プログラム
3.プログラムがしやすくなるルール
4.VS IB　～IBとルールの関連性～
5. アプリを作ろう2
6.コレ

ハイ、おつかれさまです。

メーリングリストについての共有。
メーリスって言ってますが、実質講師(井上とか)にしか届きません。
なので、プライバシーの流出に関しては、「任せとけ！」

techhubteach@googlegroups.com

Continue?

次回予告:そのまえに

次回までのあらすじ！

始めに籤を引いてもらったと思います。

なんと驚きですが、書いてある番号は、そのままこの授業での生徒番号なので、覚えておいてください。

あと、番号を techhubteach@googlegroups.com まで、
こないだ紹介したメール形式で、送ってください。

自分の名前と宛先を書け、ってあれね！

Continue?

次回予告:

次回までのあらすじ！

さっき作った電卓みたいな変更点、あらかじめ予見できるんすかね。
→できます。なにをどのように作るのか書いたものを、【仕様】といいます。
→次回はそんな話。

さあ、次回からはいよいよ、作り上げるアプリ決めだぜよ！ ばってん！

Continue?

次回予告：

超大事な事を、最後の最後の最後に。

次回までに、

各人が「欲しい!」、「作りたい」アプリのアイデアを考えておけ！

で、

紙切れでもPDFでもテキストファイルでもパワーポイントでもいいから書いて、

で、

当日授業に持ってくるがいいよ！(ぶっちゃけその場ででっちあげてもいいよ、、、)

そしたら、何事もなかったかのように、
「はい、君が今日作るのはコレです。 アイデア？　なんのことですか？」って勝手に言い出すから。

次回、

「ルールとアイデア出しと」