1204315 - Wireless Mobile Application Programming

Manasawee Kaenampornpan

มนัสวี แก่นอำพรพันธ์

manasaweek@gmail.com

Mahasarakham University

Review Class and Method

- Note that I have given the parameters of this method and the properties of this class the same names. Because of this, I need to distinguish between the two by putting the self prefix before the property names.

```
import Foundation

class TipCalculatorModel {

  var total: Double
  var taxPct: Double
  var subtotal: Double {
    get {
      return total / (taxPct + 1)
    }
  }

  init(total:Double, taxPct:Double) {
    self.total = total
    self.taxPct = taxPct
  }

  func calcTipWithTipPct(tipPct:Double) -> Double {
    return subtotal * tipPct
  }
```

This creates an initializer for the class that takes two parameters. Initializers are always named init in Swift – you can have more than one if you want, but they need to take different parameters.

Review Class and Method

- Note that I have given the parameters of this method and the properties of this class the same names. Because of this, I need to distinguish between the two by putting the self prefix before the property names.

```
import Foundation

class TipCalculatorModel {

  var total: Double
  var taxPct: Double
  var subtotal: Double {
    get {
      return total / (taxPct + 1)
    }
  }

  init(total:Double, taxPct:Double) {
    self.total = total
    self.taxPct = taxPct
  }

  func calcTipWithTipPct(tipPct:Double) -> Double {
    return subtotal * tipPct
  }
```

Note that I have given the parameters of this method and the properties of this class the same names. Because of this, I need to distinguish between the two by putting the self prefix before the property names.
Note that since there is no name conflict for the subtotal property, you don't need to add the self keyword,

Review Class and Method

- Note that I have given the parameters of this method and the properties of this class the same names. Because of this, I need to distinguish between the two by putting the self prefix before the property names.

```
import Foundation

class TipCalculatorModel {

  var total: Double
  var taxPct: Double
  var subtotal: Double {
    get {
      return total / (taxPct + 1)
    }
  }

  init(total:Double, taxPct:Double) {
    self.total = total
    self.taxPct = taxPct
  }

  func calcTipWithTipPct(tipPct:Double) -> Double {
    return subtotal * tipPct
  }
```

To declare a method, you use the func keyword. You then list the parameters (you must be explicit with the types), add the -> symbol, and finally list the return type.

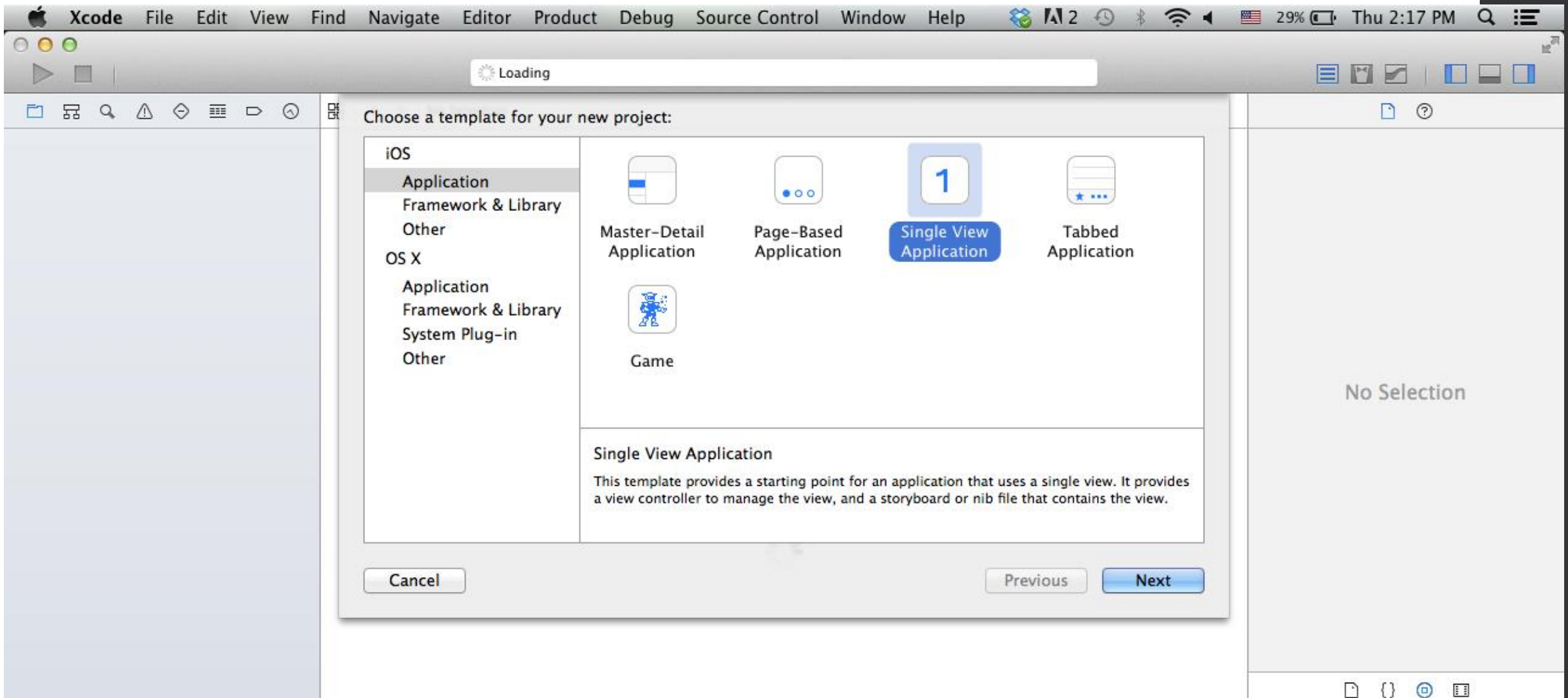
Create new proect

Create a Simple Application Using Xcode

- **Step 1)** Start Xcode. You can either locate it in Finder (Mac's version of Windows Explorer) or use Spotlight. If using Finder, Xcode is located under the *Macintosh HD/Developers/Application* directory. To find Xcode using Spotlight, simply search for "Xcode" using the magnifying glass in the top-right corner of the desktop (⌘+Space).
- **Step 2)** Select *Create a new Xcode project* from the *Welcome to Xcode* dialog box.
- **Step 3)** For your first app, choose the *Single View Application* template.
- **Note:** There are a number of other options here, and you are encouraged to explore the different project templates, but for now use the view-based application template.
- **Step 4)** Give the project a *Product Name* (Lab1) and *Bundle Identifier* (edu.iastate). If you have a device running iOS 5 or greater, you may use Storyboarding and Auto Reference Counting (ARC); otherwise, toggle them off. Click Next.
- **Note:** For this lab, do not use Storyboarding or Auto Reference Counting (ARC). Apps that use Storyboarding will only work with devices running iOS 5 or greater. Auto Reference Counting will work with devices running iOS 5 or greater, and will work for devices running iOS 4.0 or greater but will not set weak references to nil (ARClite).
- **Step 5)** Select a Location for your project (Desktop). Click *Create*.

Create a Project

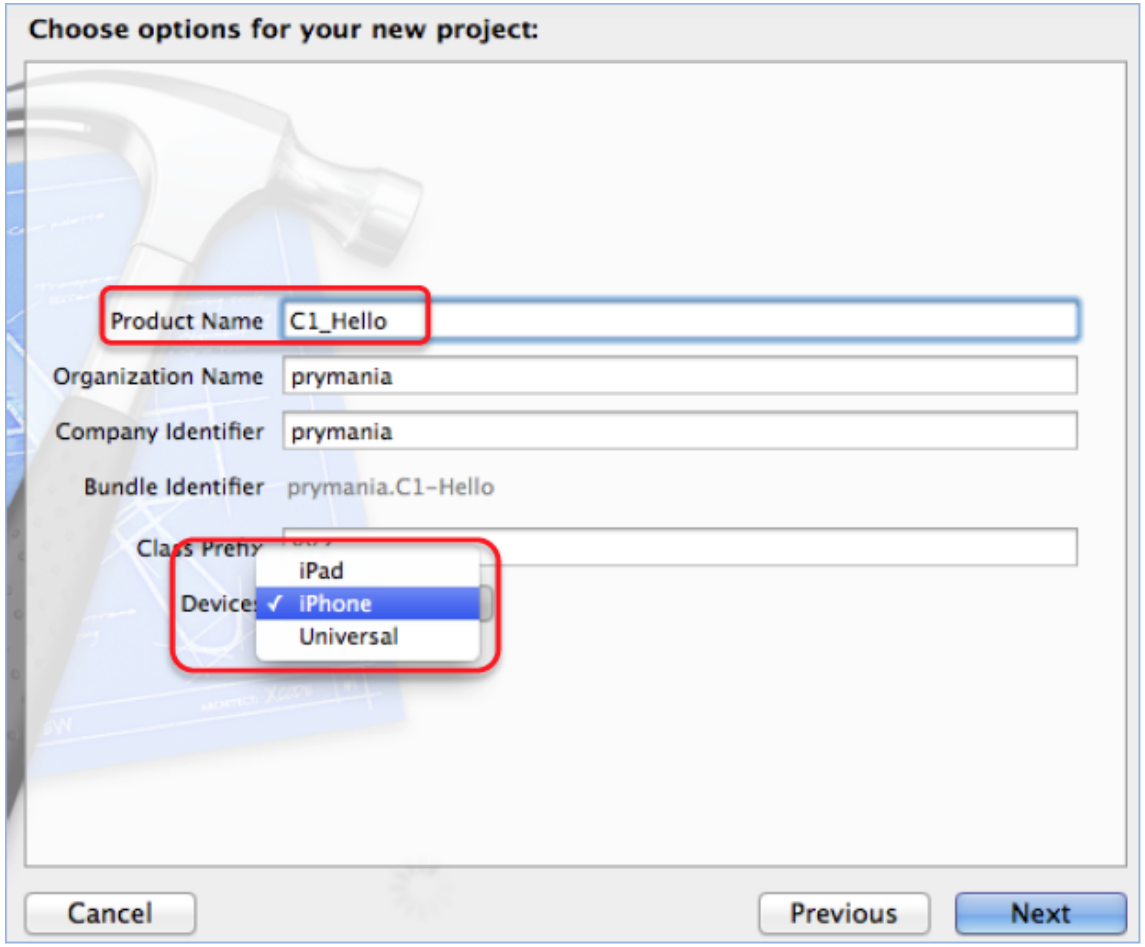
-> *iOS\Application\Single View Application*



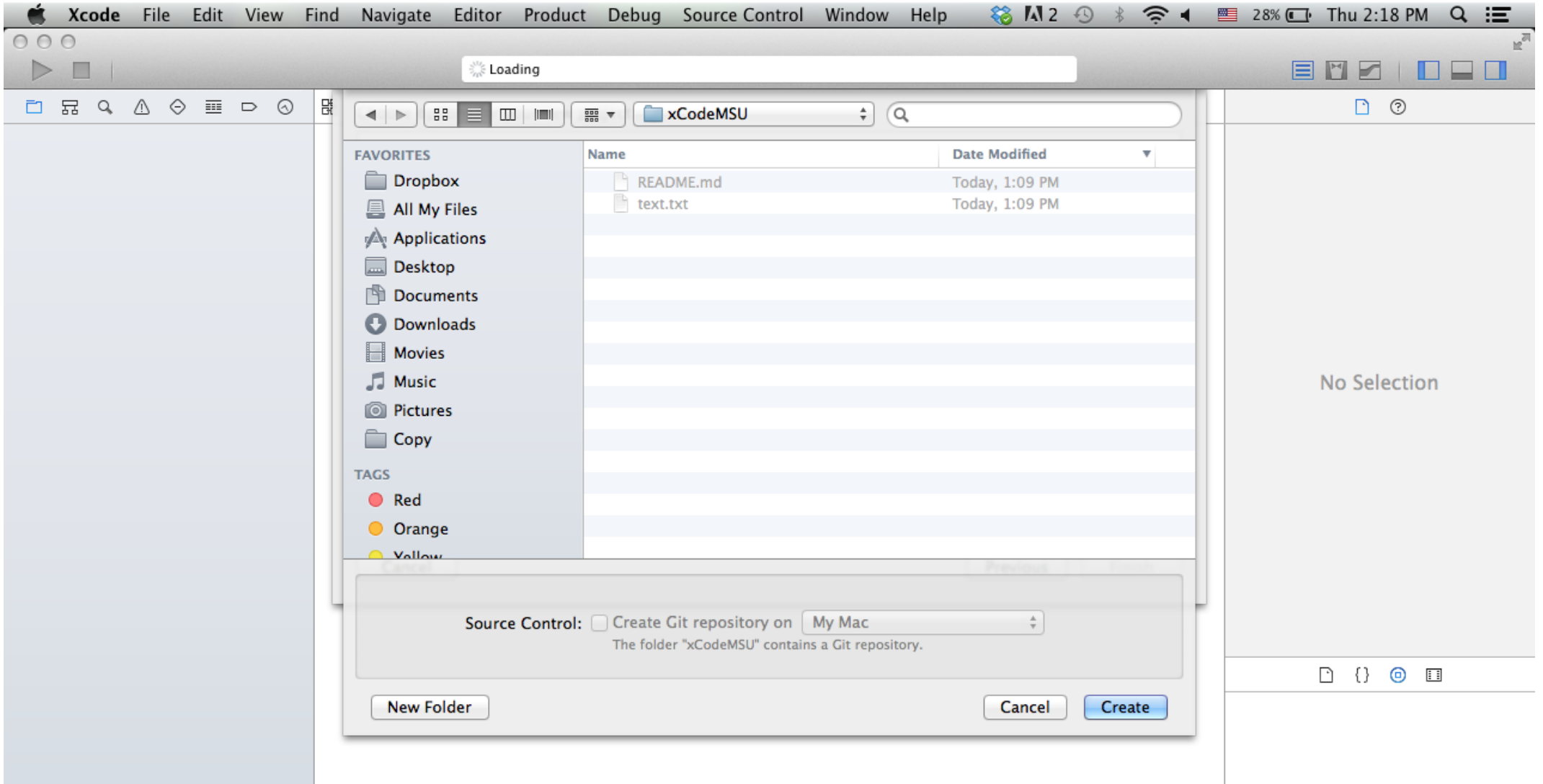
ตั้งค่าชื่อไฟล์

- ชื่อไฟล์ว่า *TipCalculator* ตรง *Product Name*
- ตั้ง *Language* to *Swift*,
- เลือก *Devices* to *iPhone*
- อย่าติ๊กตรง *Use Core Data* is *not checked*
- และกด *Next*.

ตั้งชื่อไฟล์ *TipCalculator* และเลือก Devices เป็น iPhone



Create a Project – เลือกโฟล์ดเดอร์ที่สร้าง GitHubไว้



เพิ่ม Swift ไฟล์ที่เขียนไว้แล้วมาใส่ Project

- In this tutorial, your app's model will simply be the TipCalculator class you created in the first Swift tutorial, except you will rename it to TipCalculatorModel.
- go to ***File\New\File***
- กด ***iOS\Source\Swift File***. ตั้งชื่อไฟล์ ว่า ***TipCalculatorModel.swift***
- กด ***Create***.

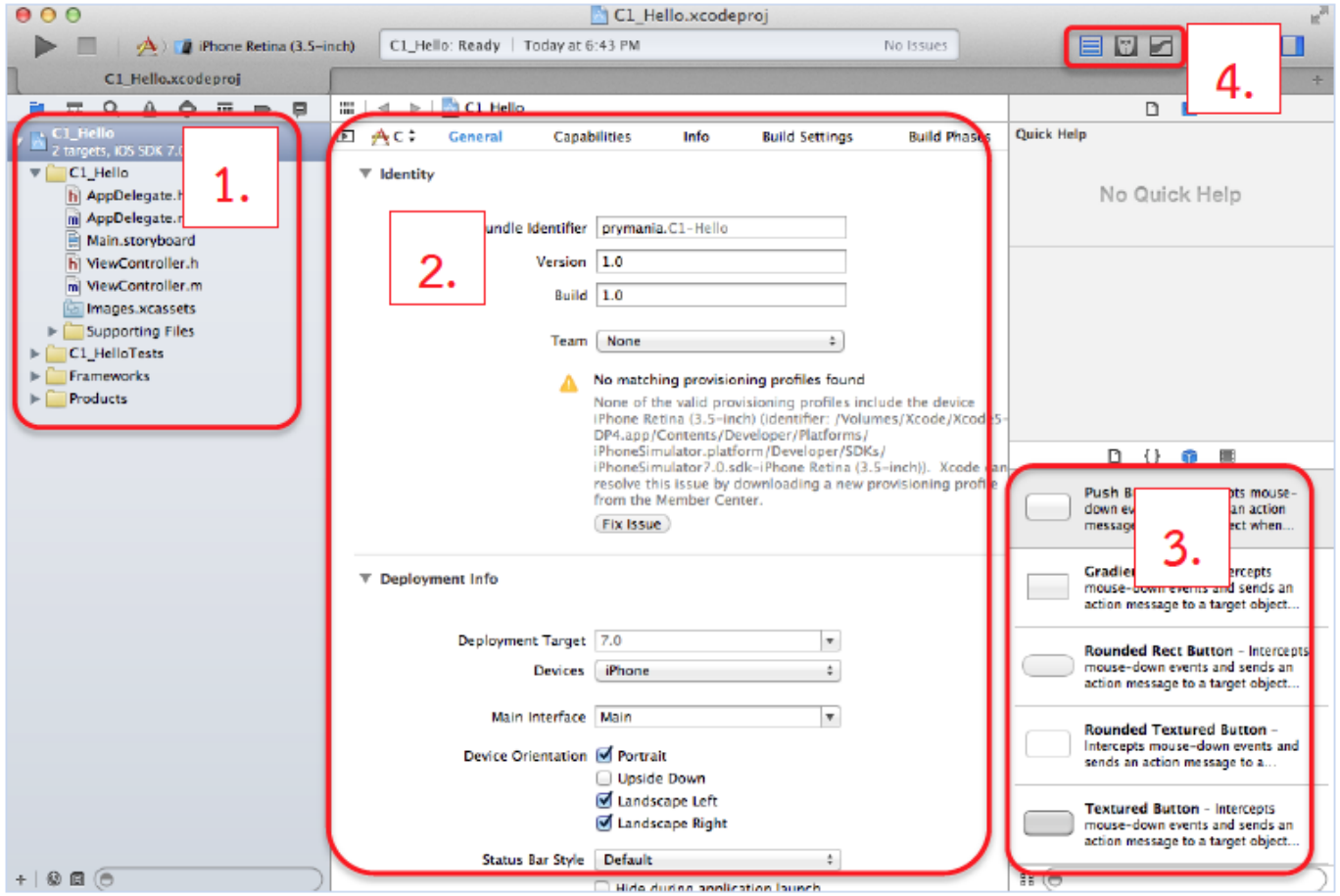
แก้ไขไฟล์ TipCalculatorModel

- เปลี่ยนชื่อ class เป็น TipCalculatorModel
- เปลี่ยน total และ taxPct จาก constants มาเป็น variables
- Because of this, you need to change subtotal to a computed property. Replace the subtotal property with the following:

```
var subtotal: Double { get { return total / (taxPct + 1) } }
```

- Delete the line that sets subtotal in init
- Delete any comments that are in the file

```swift
import Foundation

class TipCalculatorModel {

  var total: Double
  var taxPct: Double
  var subtotal: Double {
    get {
      return total / (taxPct + 1)
    }
  }

  init(total:Double, taxPct:Double) {
    self.total = total
    self.taxPct = taxPct
  }

  func calcTipWithTipPct(tipPct:Double) -> Double {
    return subtotal * tipPct
  }

  func returnPossibleTips() -> [Int: Double] {

    let possibleTipsInferred = [0.15, 0.18, 0.20]
    let possibleTipsExplicit:[Double] = [0.15, 0.18, 0.20]

    var retval = [Int: Double]()
    for possibleTip in possibleTipsInferred {
      let intPct = Int(possibleTip*100)
      retval[intPct] = calcTipWithTipPct(possibleTip)
    }
    return retval

  }
}
```



1. project navigator เพื่อเลือกไฟล์ที่เราต้องการทางาน หลักๆตอนนี้มี 3 ไฟล์คือ

Main.storyboard
ViewController.h และ
ViewController.m

2. file editor แสดงรายละเอียดไฟล์ที่เราเลือก

3. object library เก็บออปเจคต่างๆที่เราจะนามาใช้ในโปรแกรม เช่น button, textfield, label

4. เมนู show editor menu เพื่อเลือกรูปแบบการแสดงไฟล์ มีที่เราสลับใช้ไปมา 2 แบบคือ standard editor และ Assistant

Explore interface

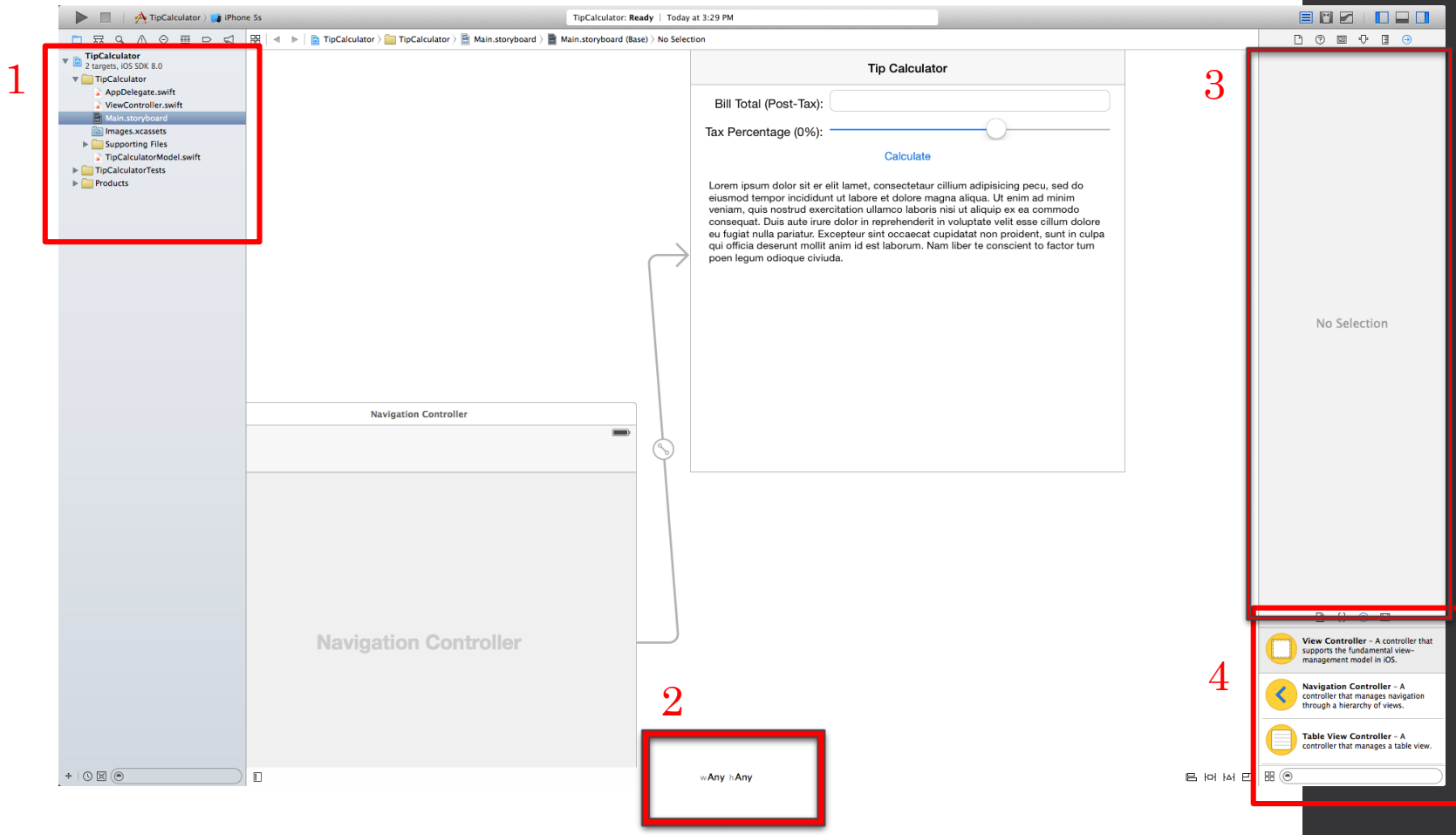
- There's a lot of stuff to cover here, so let's go over each section of the screen one at a time.

1. ***On the far left*** is your ***Project Navigator***, where your can see the files in your project.

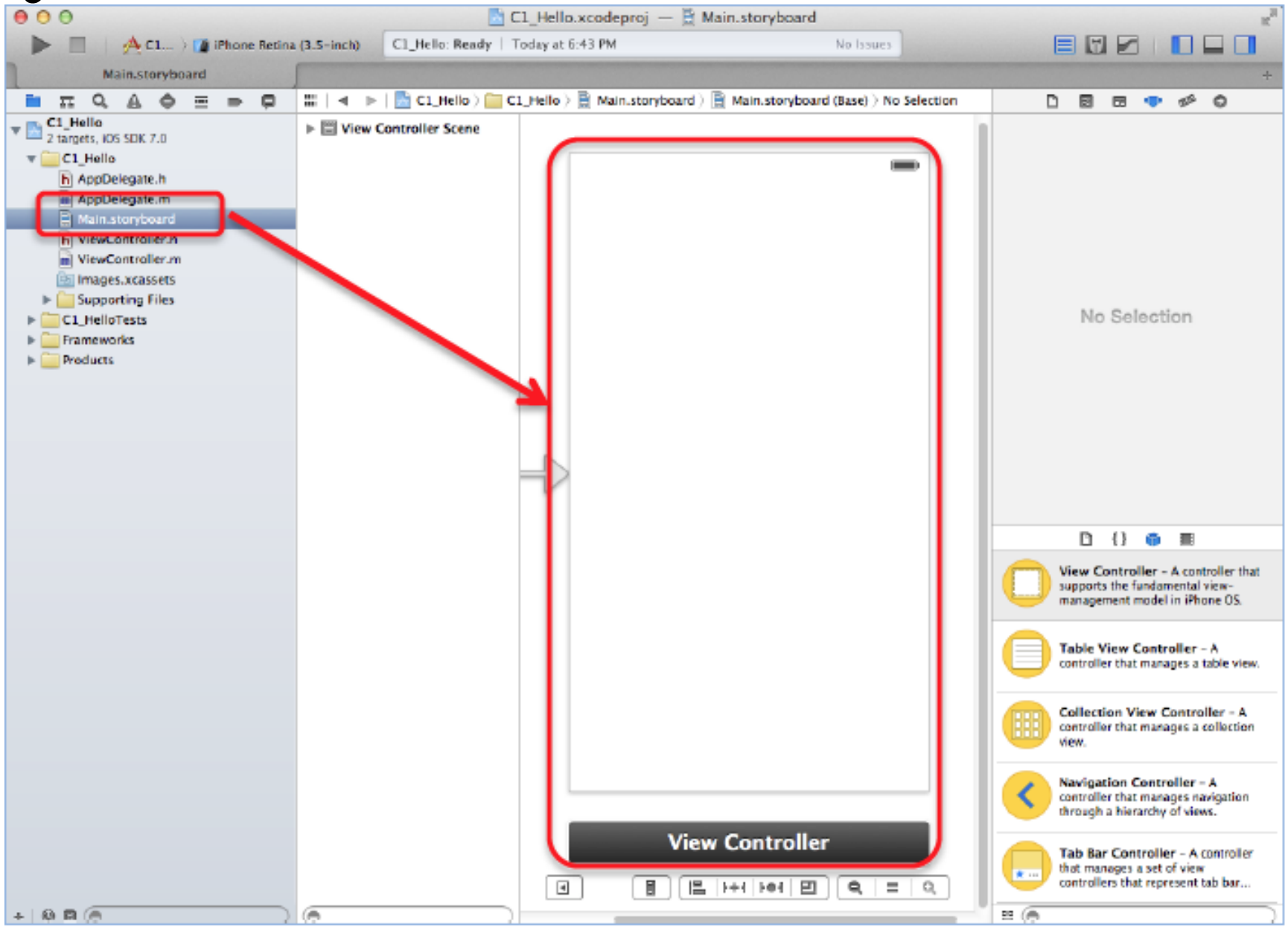
2. ***On the bottom of the Interface Builder*** you'll see something that says "w Any", "h Any". This means that you are editing the layout for your app in a way that should work on *any* sized user interface. You can do this through the power of something called ***Auto Layout***. By clicking this area, you can switch to editing the layout for devices of specific size classes. You'll learn about Adaptive UI and Auto Layout in a future tutorial.

3. ***On the upper right of Interface Builder*** are the ***Inspectors*** for whatever you have selected in the Document Outline. If you do not see the inspectors, go to ***View\Utilities\Show Utilities***.
Note there are several tabs of inspectors. You'll be using these a lot in this tutorial to configure the views you add to this project.

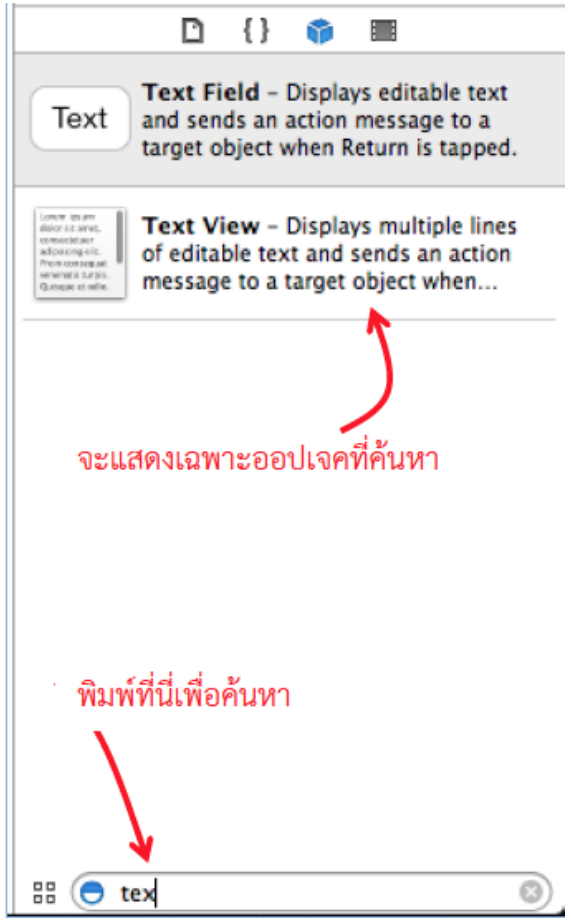
4. ***On the bottom right of Interface Builder*** are the ***Libraries***. This is a list of different types of views or view controllers you can add to your app. Soon you will be dragging items from your library



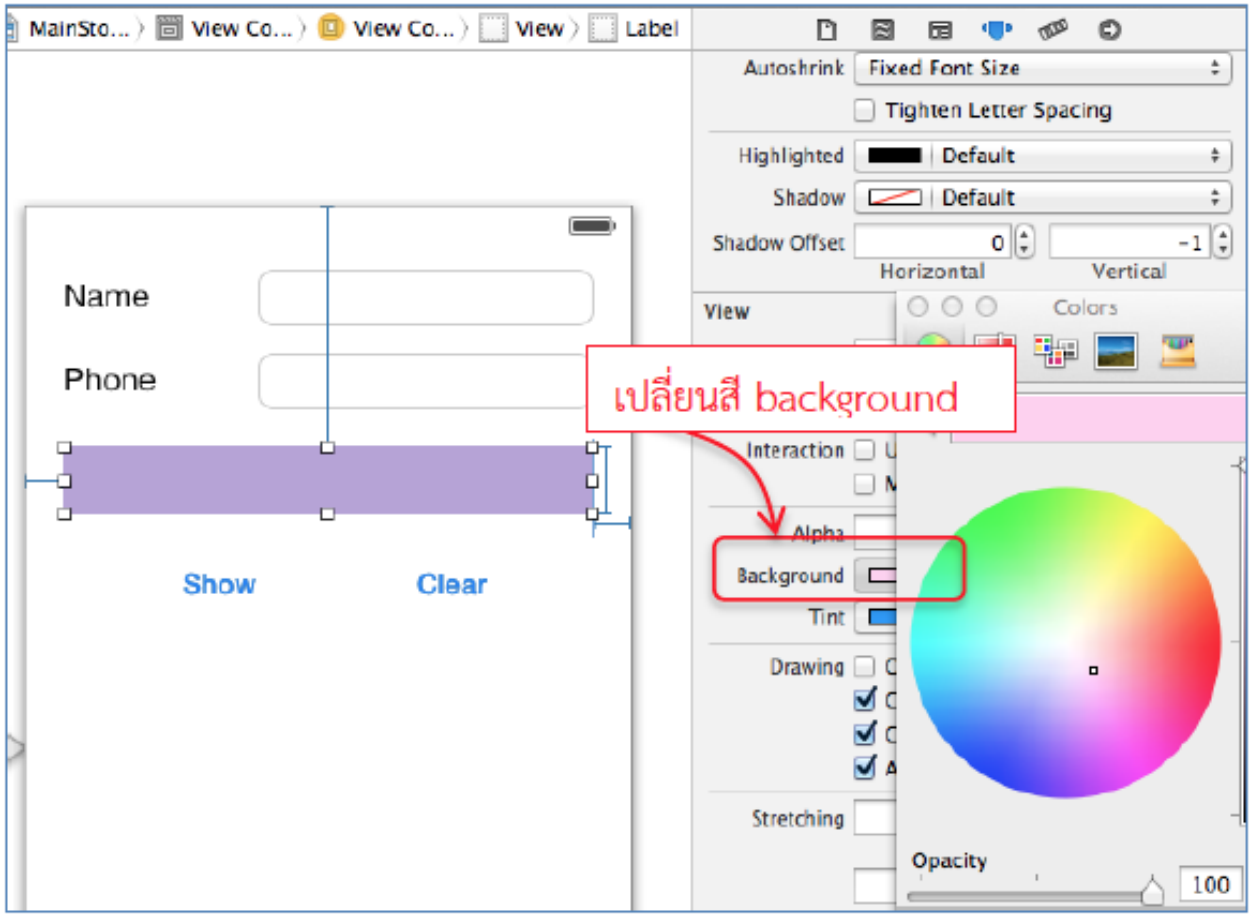
Storyboards and Interface Builder

- 

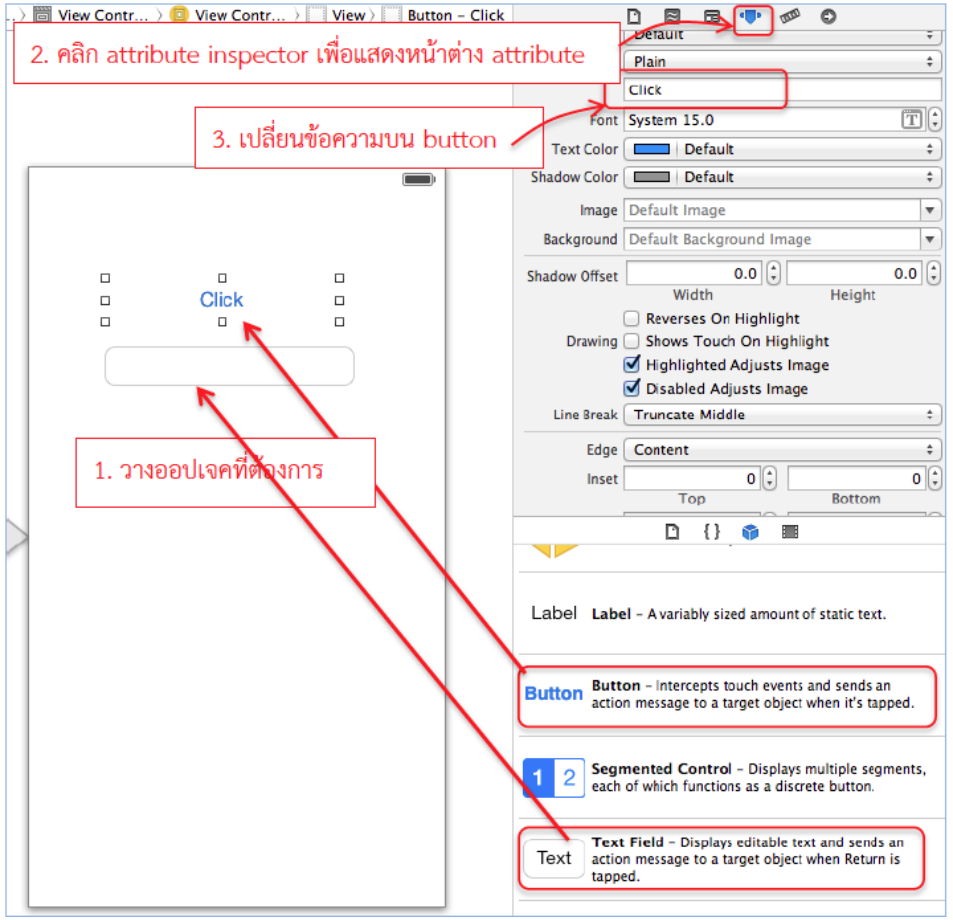
ค้นหา ออปเจคที่ต้องการ



ตั้งค่าเพิ่มเติม


MainSto...
View Co...
View Co...
View
Label
Name
Phone
Show
Clear
เปลี่ยนสี background
Autoshrink
Fixed Font Size
Tighten Letter Spacing
Highlighted
Default
Shadow
Default
Shadow Offset
0
-1
Horizontal
Vertical
View
Colors
Interaction
Alpha
Background
Tint
Drawing
Stretching
Opacity
100

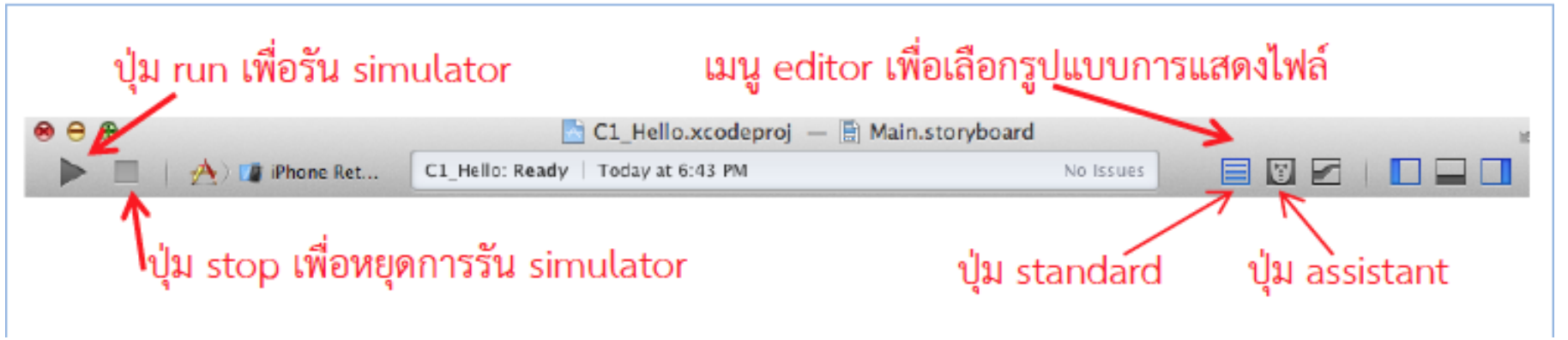
Project Navigator > MainStoryboard



ที่หน้าต่าง object library

- เลื่อนลงมาด้านล่างจนเห็น Label ให้วาง Label บน iPhone
- คลิกเลือก Label บน Phone
- คลิกปุ่ม Attribute inspector เพื่อเปลี่ยนข้อความบน Label เป็นคาว่า ***Bill Total (Post-Tax):***
- เราสามารถเปลี่ยนคุณสมบัติต่างๆของออปเจคได้ที่หน้าต่าง attribute inspector นี้

Tool Bar ด้านบน



Standard Editor จะแสดงหน้าจอเดีย ไฟล์เดียว ง่ายต่อการเขียนโปรแกรม
Assistant Editor จะแสดง 2 หน้าจอ ใช้เลาเราสรางลิงค์เชื่อตัวแปร

Creating your views

Let's build this user interface one piece at a time.

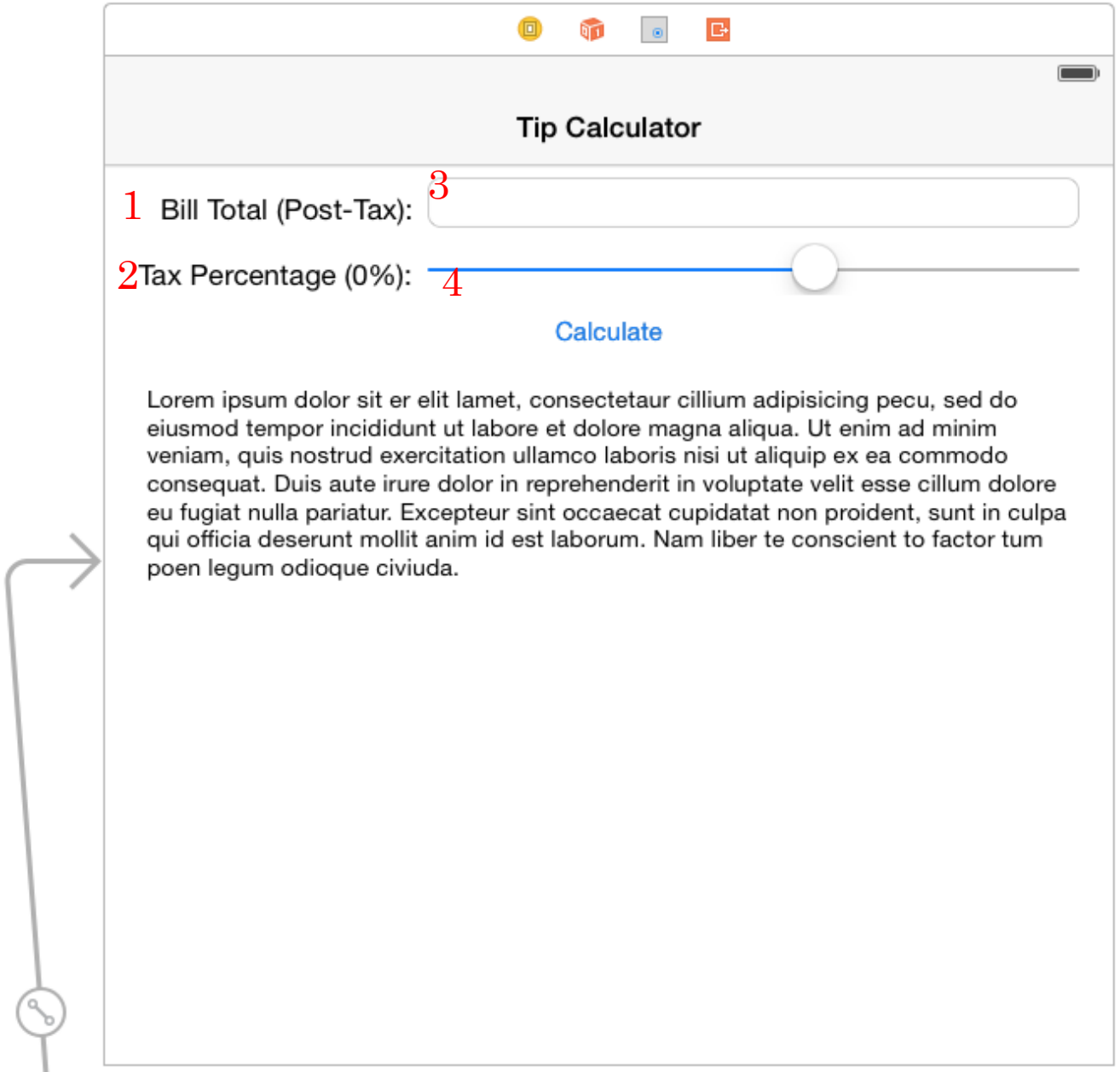
- ***Navigation bar.*** Rather than adding a navigation bar directly, select your view controller and go to ***Editor\Embed In\Navigation Controller***. This will set up a Navigation Bar in your view controller. Double click the Navigation Bar (the one inside your view controller), and set the text to ***Tip Calculator***.

1. ***Labels.*** From the Object Library, drag a ***Label*** into your view controller. Double click the label and set its text to ***Bill Total (Post-Tax):***. Select the label, and in the ***Inspector***'s fifth tab (the ***Size Inspector***), set ***X=33*** and ***Y=81***.

2. Repeat this for another label, but set the text to ***Tax Percentage (0%):***, ***X=20***, and ***Y=120***.

3. ***Text Field.*** From the Object Library, drag a Text Field into your view controller. In the Attributes Inspector, set ***Keyboard Type=Decimal Pad***. In the Size Inspector, set ***X=192***, ***Y=72***, and ***Width=268***.

4. ***Slider.*** From the Object Library, drag a Slider into your view controller. In the Attribute Inspector, set ***Minimum Value=0***, ***Maximum Value=10***, and ***Current Value=6***. In the Size Inspector, set ***X=190***, ***Y=111***, and***Width=272***.



Creating your views

Let's build this user interface one piece at a time.

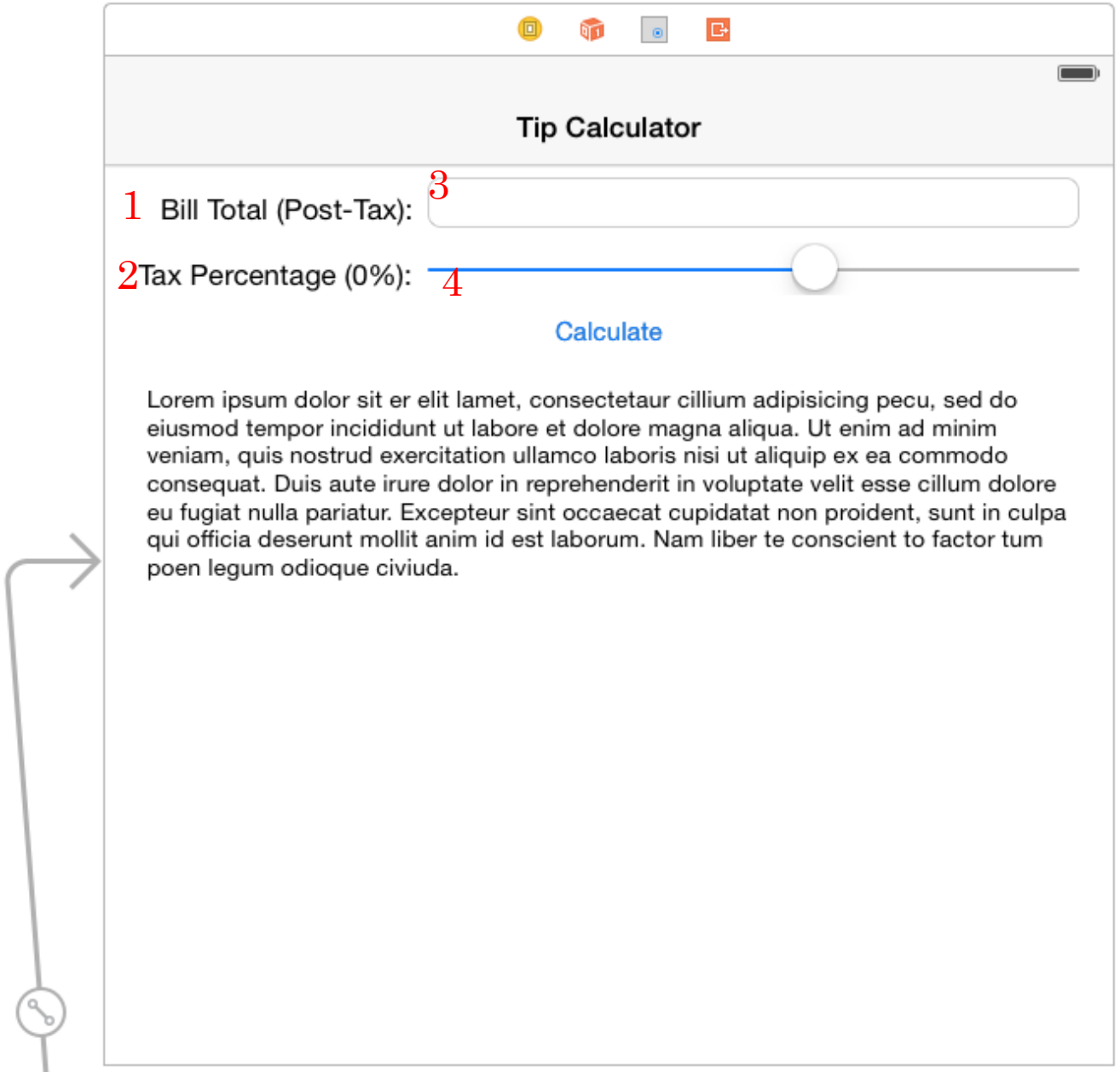
- ***Navigation bar.*** Rather than adding a navigation bar directly, select your view controller and go to ***Editor\Embed In\Navigation Controller***. This will set up a Navigation Bar in your view controller. Double click the Navigation Bar (the one inside your view controller), and set the text to ***Tip Calculator***.

1. ***Labels.*** From the Object Library, drag a ***Label*** into your view controller. Double click the label and set its text to ***Bill Total (Post-Tax):***. Select the label, and in the ***Inspector***'s fifth tab (the ***Size Inspector***), set ***X=33*** and ***Y=81***.

2. Repeat this for another label, but set the text to ***Tax Percentage (0%):***, ***X=20***, and ***Y=120***.

3. ***Text Field.*** From the Object Library, drag a Text Field into your view controller. In the Attributes Inspector, set ***Keyboard Type=Decimal Pad***. In the Size Inspector, set ***X=192***, ***Y=72***, and ***Width=268***.

4. ***Slider.*** From the Object Library, drag a Slider into your view controller. In the Attribute Inspector, set ***Minimum Value=0***, ***Maximum Value=10***, and ***Current Value=6***. In the Size Inspector, set ***X=190***, ***Y=111***, and***Width=272***.



Creating your views

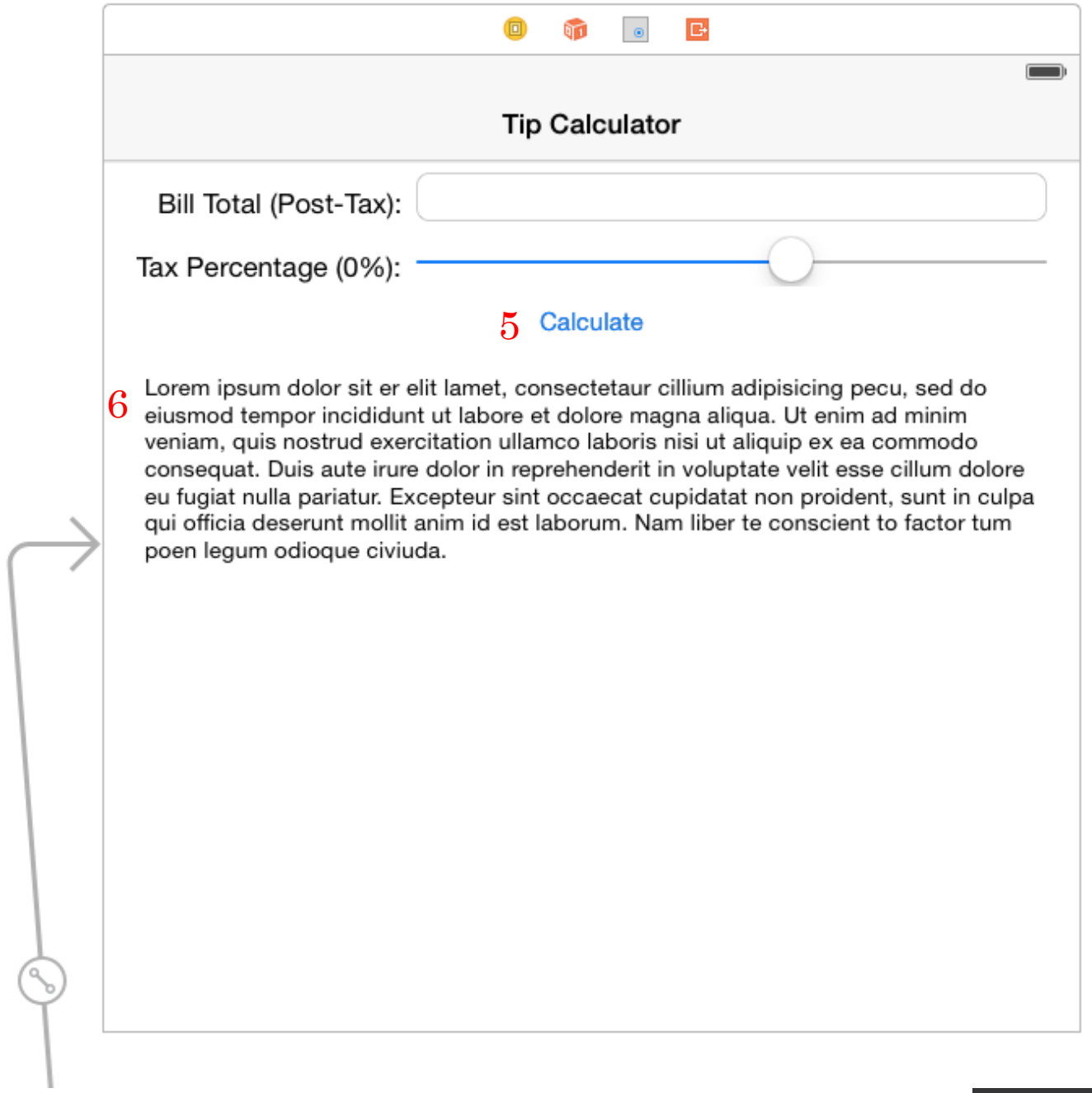
Let's build this user interface one piece at a time.

5. ***Button.*** From the Object Library, drag a Button into your view controller. Double click the Button, and set the text to ***Calculate***. In the Size Inspector, set ***X=208*** and ***Y=149***.

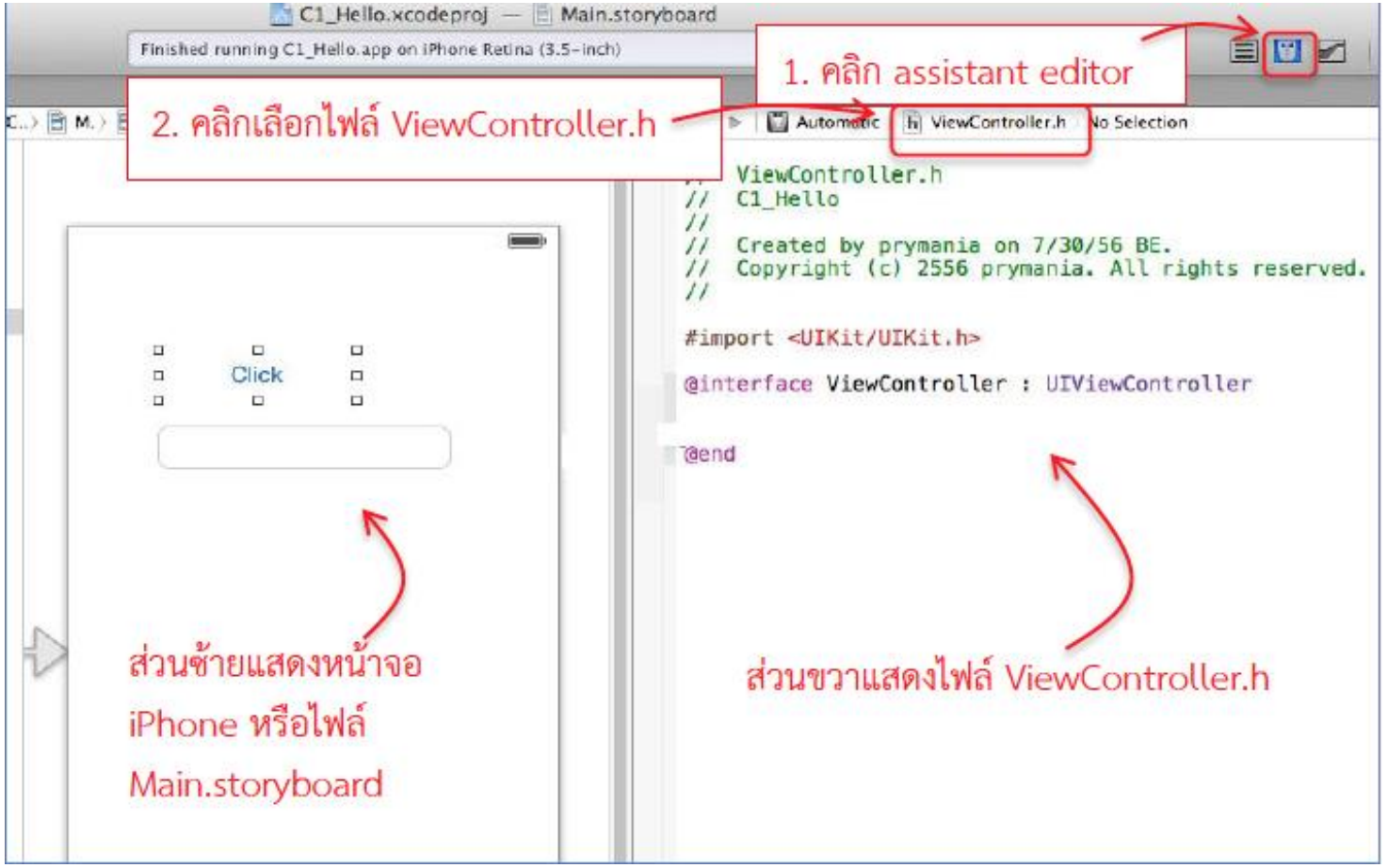
6. ***Text View.*** From the Object Library, drag a Text View into your View Controller. Double click the Text View, and delete the placeholder text. In the Attributes Inspector, make sure ***Editable*** and ***Selectable are not checked***. In the Size Inspector, set ***X=20***, ***Y=187***, ***Width=440***, and ***Height=288***.

7. ***Tap Gesture Recognizer.*** From the Object Library, drag a Tap Gesture Recognizer onto your main view. This will be used to tell when the user taps the view to dismiss the keyboard.

8. ***Auto Layout.*** Interface Builder can often do a great job setting up reasonable Auto Layout constraints for you automatically; and it definitely can in this case. To do this, click on the third button in the lower left of the Interface Builder (which looks like a Tie Fighter) and select ***Add Missing Constraints.***



Assistant editor



A View Controller Tour

- Open ***ViewController.swift***. จะเห็นเป็นไฟล์แบบนี้

This is the Swift code for your single view controller (“screen”) in your app. It is responsible for managing the communication between your views and your model.

```
// 1
import UIKit

// 2
class ViewController: UIViewController {

  // 3
  override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
  }

  // 4
  override func didReceiveMemoryWarning() {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
  }

}
```

Import frameworks & class and subclass

1. iOS is split up into multiple frameworks, each of which contain different sets of code. Before you can use code from a framework in your app, you have to import it like you see here. UIKit is the framework that contains the base class for view controllers, various controls like buttons and text fields, and much more.

2. This is the first example you've seen of a class that subclasses another class. Here, you are declaring a new class ViewController that subclasses Apple's UIViewController.

3. This method is called with the root view of this view controller is first accessed. Whenever you override a method in Swift, you need to mark it with the override keyword. This is to help you avoid a situation where you override a method by mistake.

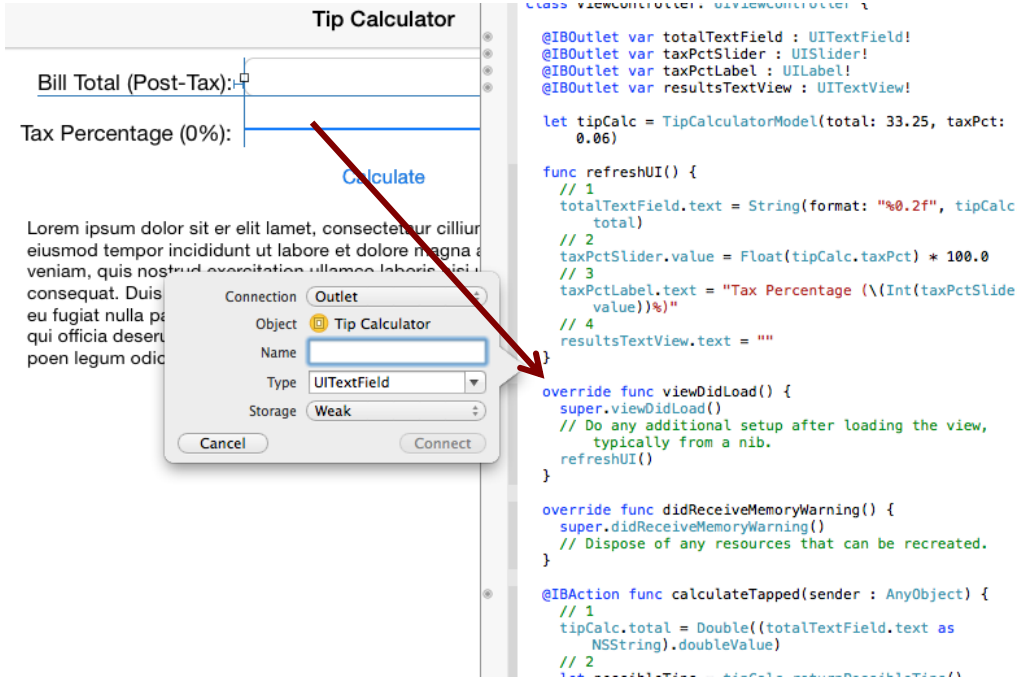
4. This method is called when the device is running low on memory. It's a good place to clean up any resources you can spare.

Connecting your View Controller to your Views

Note: วิธีลัด สามารถทำได้โดย เปิด Main.storyboard Assistant Editor (View\Assistant Editor\Show Assistant Editor) โดยใน assistant editor ให้เปิด view controller's Swift code.

แล้วเราก็กด control ลากจากobject Main.storyboard ไปที่ โค้ดบน Assistant Editor ก่อน viewDidLoad.

แล้วจะมี popup ให้เราใส่ชื่อ property แล้วกด Connect.



ตั้งชื่อ GUI แต่ละตัวดังนี้ ให้ใช้วิธี Control + ลาก มาที่ไฟล์ ViewController.swift

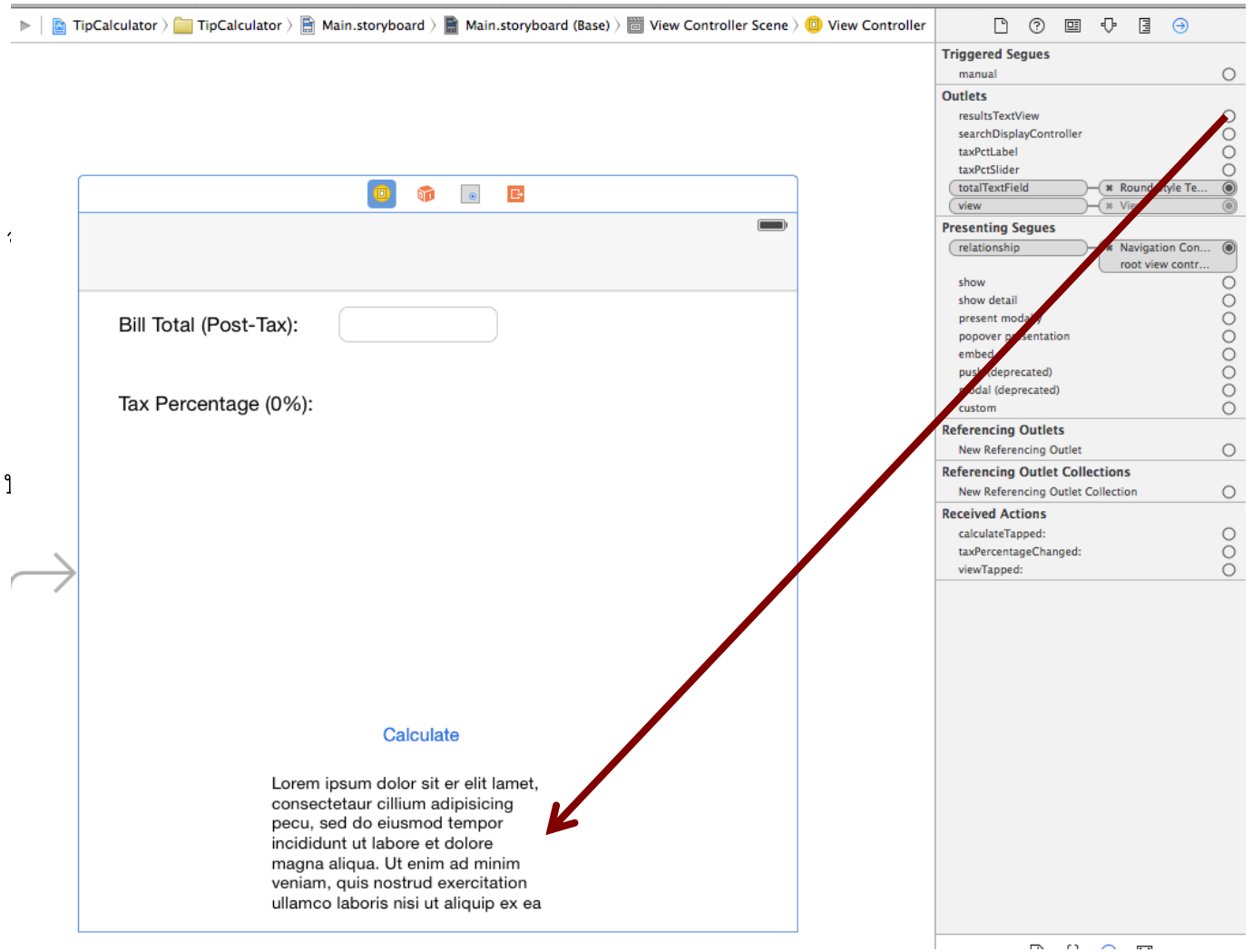
- เพิ่ม properties ลงไป ViewController class (right before viewDidLoad):
- นี่คือการ declaring four variables

```
@IBOutlet var totalTextField : UITextField!
@IBOutlet var taxPctSlider : UISlider!
@IBOutlet var taxPctLabel : UILabel!
@IBOutlet var resultsTextView : UITextView!
```

- There's only two differences:
- You're prefixing these variables with the @IBOutlet keyword. Interface Builder scans your code looking for any properties in your view controller prefixed with this keyword. It exposes any properties it discovers so you can connect them to views.
- You're marking the variables with an exclamation mark (!). This indicates the variables are optional values, but they are implicitly unwrapped. This is a fancy way of saying you can write code assuming that they are set, and your app will crash if they are not set.

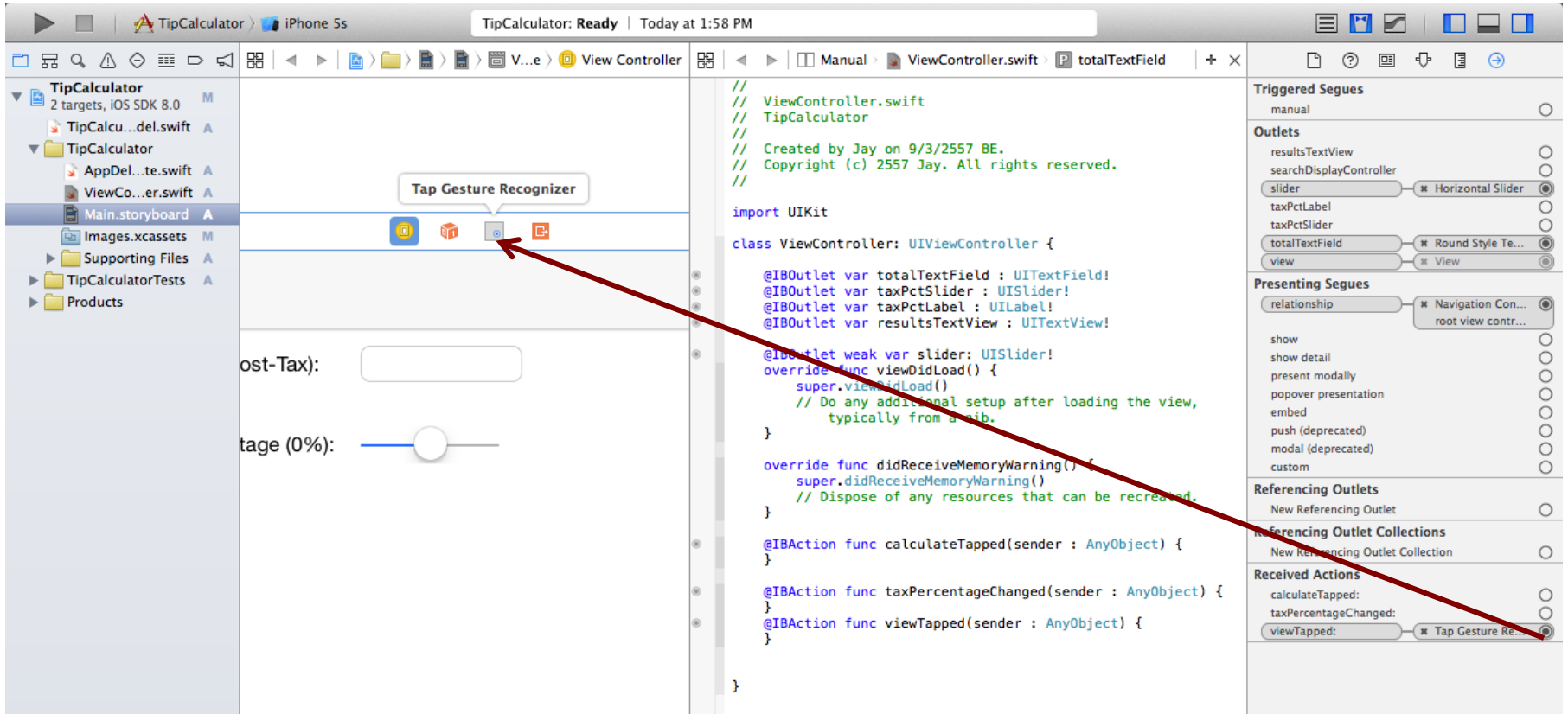
การเช็คเรื่อง Connecting Properties to UI element

- เปิด Main.storyboard เลือก View Controller ใน Document Outline.
- เปิด Connections Inspector (6th tab) เห็น Outlets section.
- ปุ่มกลมเล็กด้านขวาของ resultsTextView
- กดปุ่ม Control แล้วลากจากปุ่มมาที่ text view ใต้ Calculate button แล้วปล่อย เพื่อเชื่อม Swift property มาที่ view



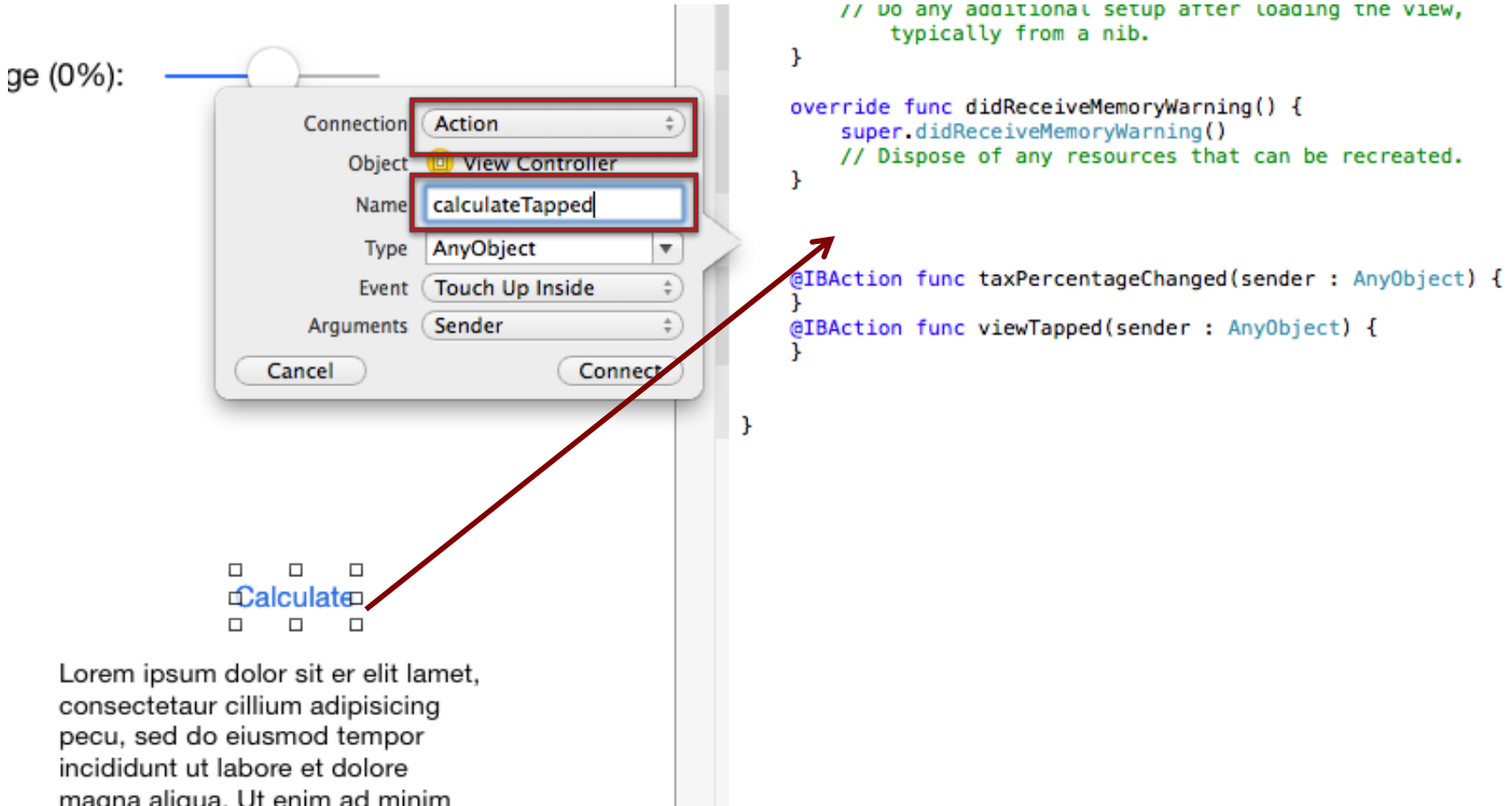
Connecting Actions to your View Controller

- Drag from viewTapped: to the Tap Gesture Recognizer in the document outline. There are no actions to choose from for gesture recognizers; your method will simply be called with the recognizer is triggered.



วิธีการ Connecting Actions to your View Controller ที่ดีและสะดวก

Note - วิธีลัดกด control ลากจาก interface ใน storyboard มาที่ code ก็ได้เหมือนกัน



ให้ตั้งชื่อ Action ตามนี้ในการConnecting Actions to your View Controller

- To do this, open ViewController.swift and add these three new methods anywhere in your class:

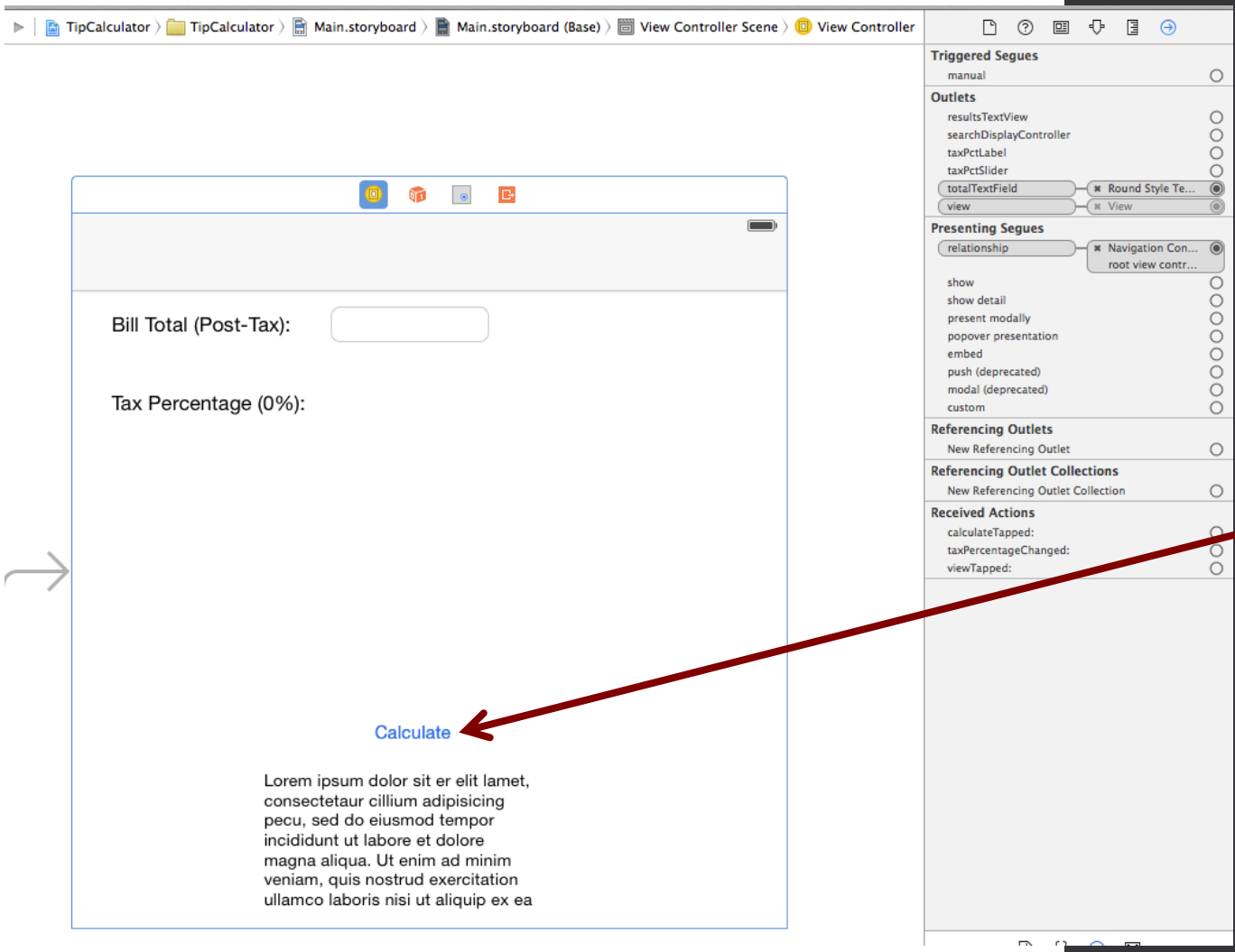
```
@IBAction func calculateTapped(sender : AnyObject) {
}
@IBAction func taxPercentageChanged(sender : AnyObject) {
}
@IBAction func viewTapped(sender : AnyObject) {
}
```

- When you declare callbacks for actions from views, they always need to have this same signature – a function with no return value, that takes a single parameter of type AnyObject as a parameter, which represents a class of any type.

- 1. Control+ลาก Button มาที่ ViewController.swift แล้วตั้งเป็น Action ชื่อ calculateTapped In the popup that appears, choose **Touch Up Inside**
- 2 Control+ลาก slider มาที่ ViewController.swift แล้วตั้งเป็น Action ชื่อ taxPercentageChanged In the popup that appears, choose **Touch Up Inside**

เช็คการ Connecting Actions to your View Controller

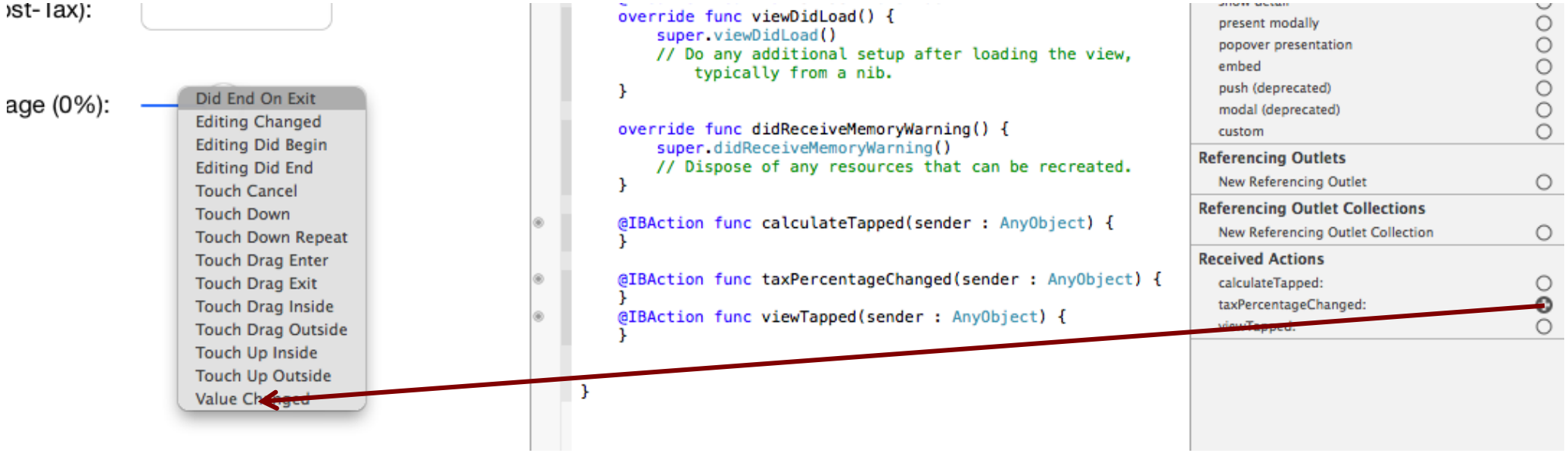
- To make Interface Builder notice your new methods, you need to mark these methods with the @IBAction keyword (just as you marked properties with the @IBOutlet keyword).
- Next, switch back to Main.storyboard and make sure that your view controller is selected in the Document Outline. Make sure the Connections Inspector is open (6th tab) and you will see your new methods listed in a the Received Actions section.
- Find the circle to the right of calculateTapped:, and drag a line from that circle up to the Calculate button.
- In the popup that appears, choose **Touch Up Inside**:
- This is effectively saying “when the user releases their finger from the screen when over the button, call my method calculateTapped:“.



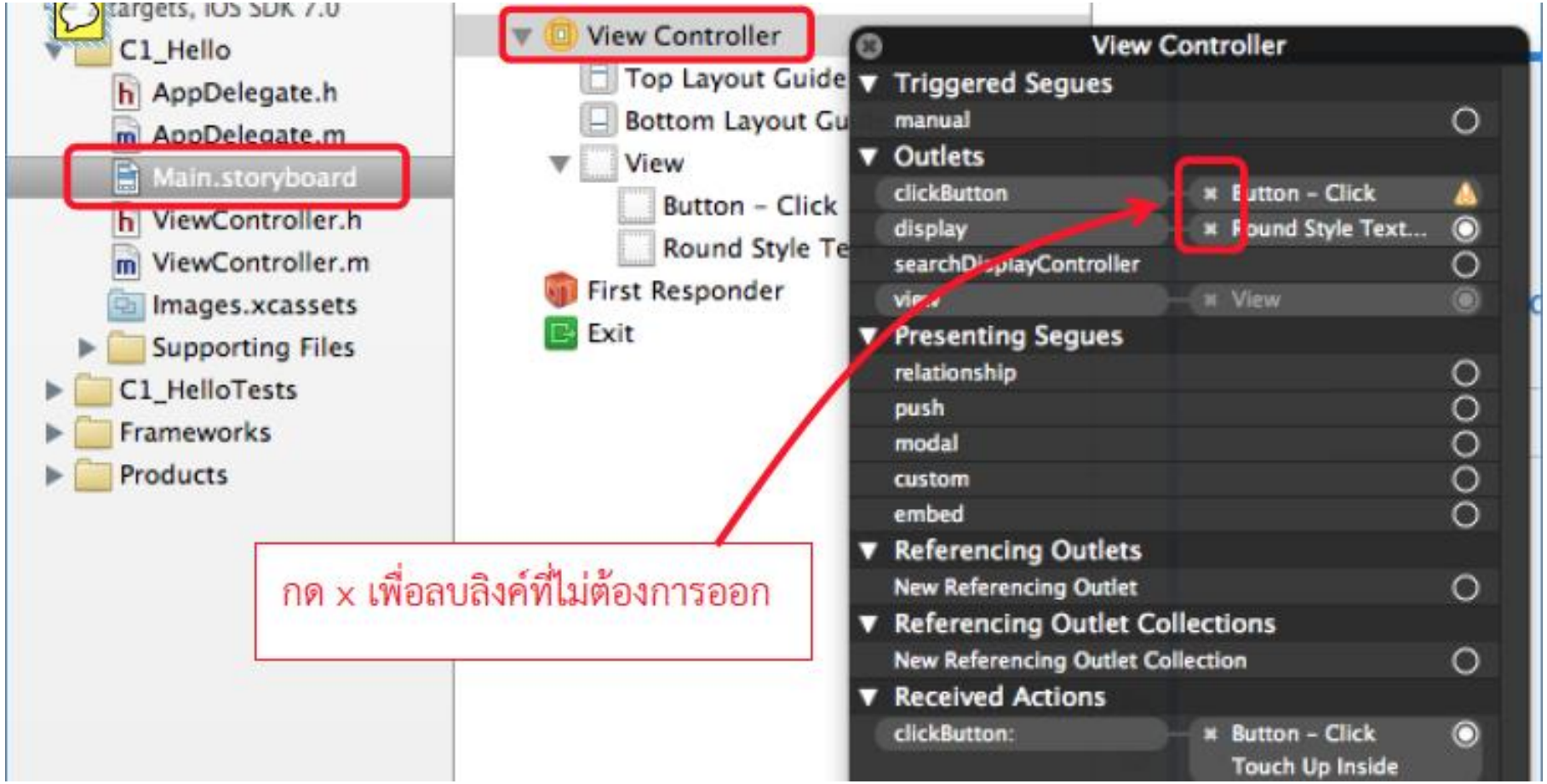
การ Connecting Actions to your View Controller

อีกวิธี ถ้าไม่จำเป็น**ไม่ใช้**วิธีนี้

- Drag from taxPercentageChanged: to your slider, and connect it to the Value Changed action, which is called every time the user moves the slider.



การลบ ลิงค์ที่ไม่ต้องการออก



Connecting Your View Controller to your Model

- เปิด ***ViewController.swift*** and add a property for the model to your class
- เพิ่มmethod to refresh the UI:

```
let tipCalc = TipCalculatorModel(total: 33.25, taxPct: 0.06)

func refreshUI() {
  // 1
  totalTextField.text = String(format: "%0.2f", tipCalc.total)
  // 2
  taxPctSlider.value = Float(tipCalc.taxPct) * 100.0
  // 3
  taxPctLabel.text = "Tax Percentage (\(Int(taxPctSlider.value))%)"
  // 4
  resultsTextView.text = ""
}
```

1. Convert tipCalc.total from double to string
2. เพื่อให้แสดงเป็น 0-10% แทนที่จะเป็นเลขจดทศนิยม เลยคูณ 100
3. เอาค่า String เพื่อมาอัพเดทและแสดงบน Label
4. Clear ค่า จนกว่าจะกดปุ่ม Calculate

เพิ่ม refreshUI

- เพิ่ม refreshUI() ใน viewDidLoad

```
override func viewDidLoad() {
  super.viewDidLoad()
  // Do any additional setup after loading the view,
      typically from a nib.
  refreshUI()
}
```

เพิ่ม Code ใน func taxPercentageChange และ viewTapped

```
@IBAction func taxPercentageChanged(sender : AnyObject) {
  tipCalc.taxPct = Double(taxPctSlider.value) / 100.0
  refreshUI()
}
@IBAction func viewTapped(sender : AnyObject) {
  totalTextField.resignFirstResponder()
}
```

เพื่อ ไม่ให้ใช้ คีบอร์ด

แก้ไข ฟังชั่น calculateTapped

```
@IBAction func calculateTapped(sender : AnyObject) {
  // 1
  tipCalc.total = Double((totalTextField.text as
      NSString).doubleValue)
  // 2
  let possibleTips = tipCalc.returnPossibleTips()
  var results = ""
  // 3
  for (tipPct, tipValue) in possibleTips {
    // 4
    results += "\(tipPct)%: \(tipValue)\n"
  }
  // 5
  resultsTextView.text = results
}
```

แก้ไข ฟังชั่น calculateTapped

```
@IBAction func calculateTapped(sender : AnyObject) {
  // 1
  tipCalc.total = Double((totalTextField.text as
      NSString).doubleValue)
  // 2
  let possibleTips = tipCalc.returnPossibleTips()
  var results = ""
  // 3
  for (tipPct, tipValue) in possibleTips {
    // 4
    results += "\(tipPct)%: \(tipValue)\n"
  }
  // 5
  resultsTextView.text = results
}
```

Convert String มาเป็น Double ตอนนี้ใช้ NSSTring ของ Objective-C

```
func returnPossibleTips() -> [Int: Double] {

  let possibleTipsInferred = [0.15, 0.18, 0.20]
  let possibleTipsExplicit:[Double] = [0.15, 0.18, 0.20]

  var retval = [Int: Double]()
  for possibleTip in possibleTipsInferred {
    let intPct = Int(possibleTip*100)
    retval[intPct] = calcTipWithTipPct(possibleTip)
  }
  return retval
}
```

จาก ***TipCalculatorModel.swift***

To Sort the result

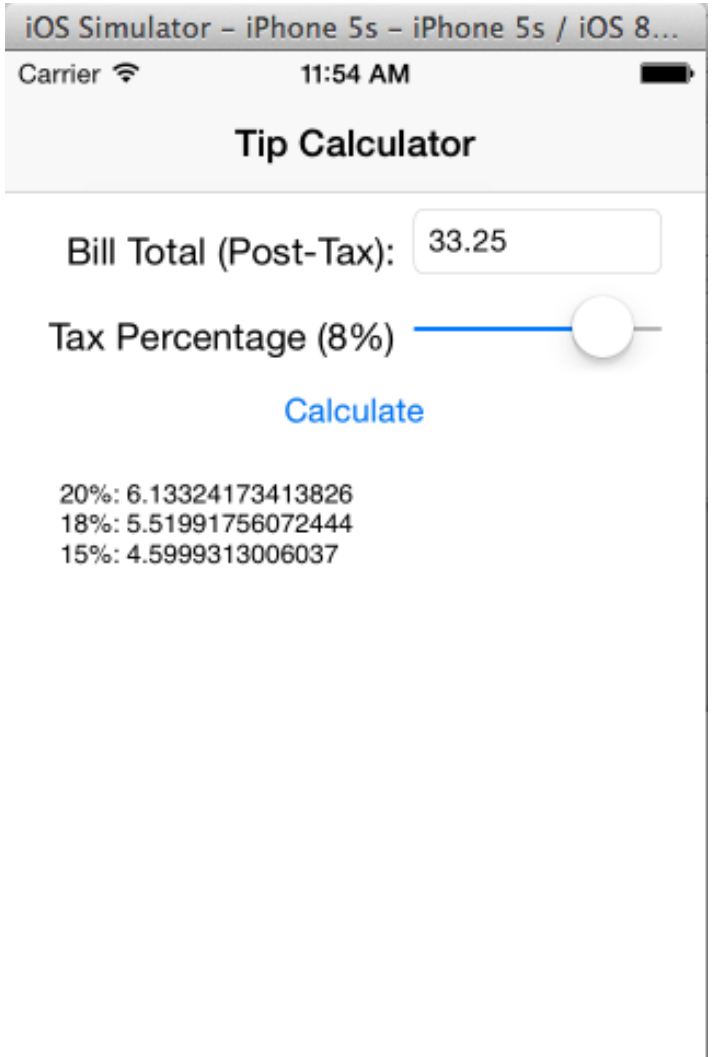
```
var keys = Array(possibleTips.keys)
sort(&keys)
for tipPct in keys {
  let tipValue = possibleTips[tipPct]!
  let prettyTipValue = String(format:"%.2f", tipValue)
  results += "\(tipPct)%: \(prettyTipValue)\n"
}
```

Test Your Application in the iPhone Simulator

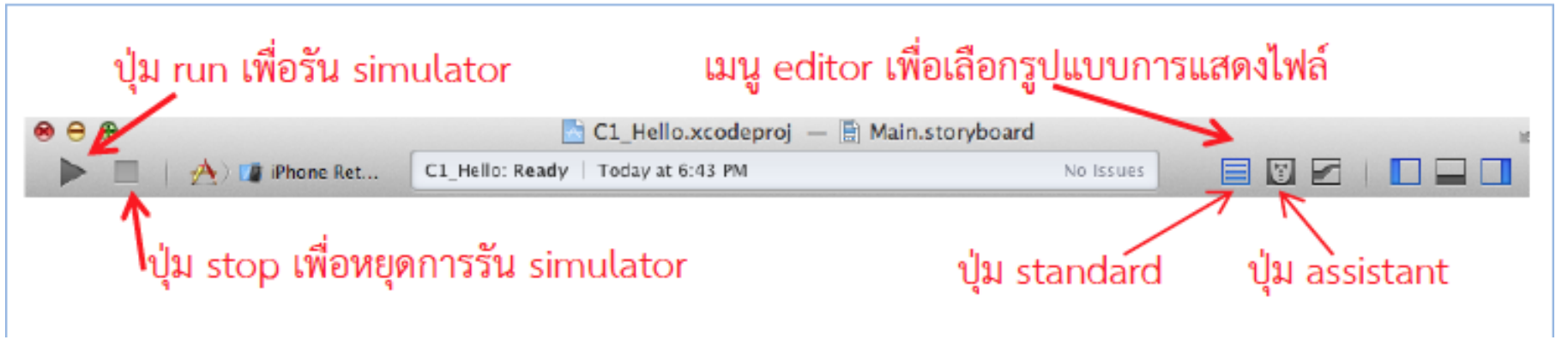
At this point our application can be simulated, though it will not do much since we have not written any code. It will simply be a blank gray application.

- **Step 1)** Select the iPhone Simulator from the drop down at the top of the main window.
- **Step 2)** Click the Run button located at the top left corner of the main window. You should see the simulator appear with a blank application started (image to left).

ต่อจาก TipCalculator

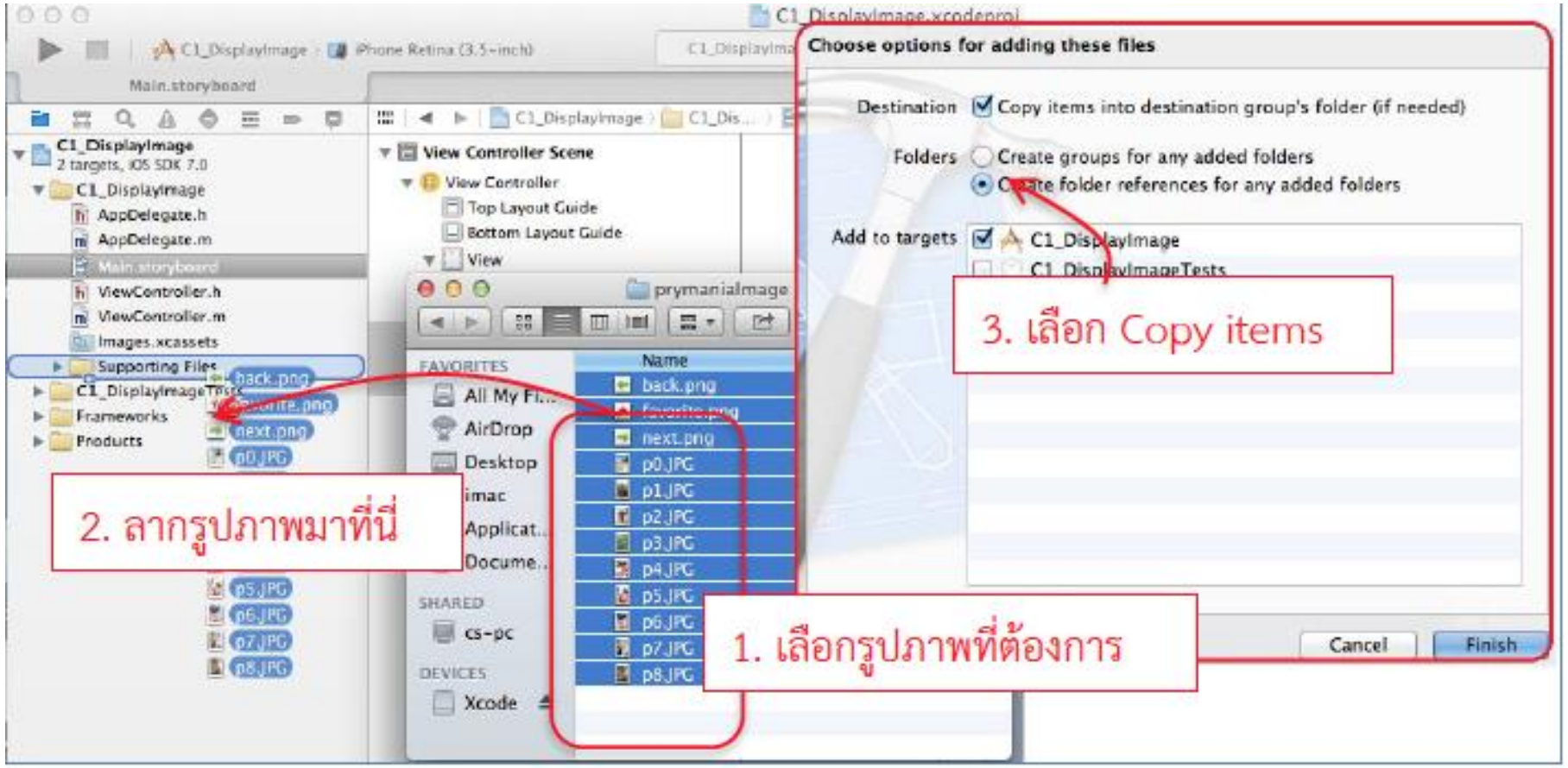


Tool Bar ด้านบน

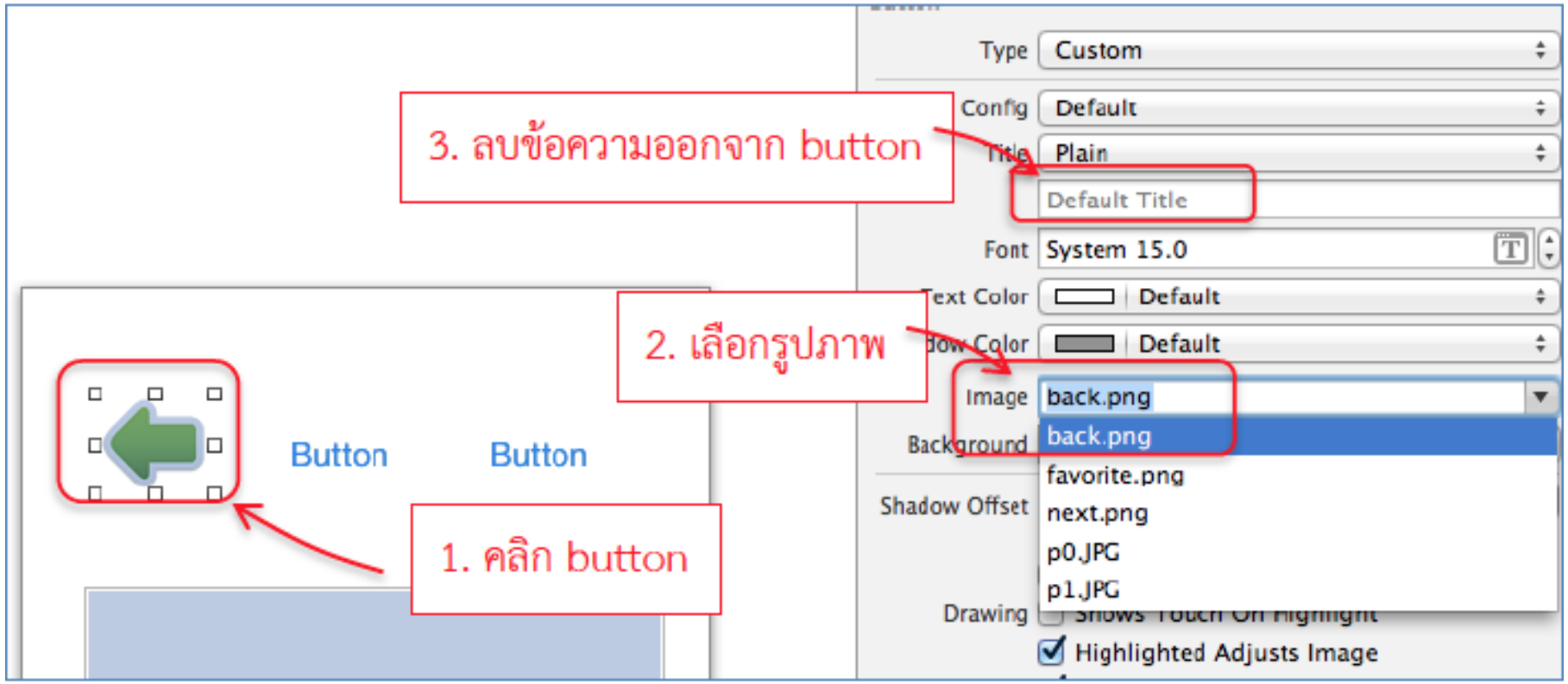


Standard Editor จะแสดงหน้าจอเดีย ไฟล์เดียว ง่ายต่อการเขียนโปรแกรม
Assistant Editor จะแสดง 2 หน้าจอ ใช้เลาเราสรางลิงค์เชื่อตัวแปร

ลองเพิ่ม รูปมาใส่ในโปรเจค



ลองเปลี่ยนชื่อปุ่มเป็นรูปแทน ในปุ่ม calculate ในโปรเจคเรา



แบบฝึกหัด

เขียนโปรแกรมให้มีการทำงานดังนี้

button 3 ตัวเปลี่ยนข้อความเป็น One Two Three
label 3 ตัว เปลี่ยนข้อความเป็น 0 ทั้ง 3 ตัว
button 1 ตัว เปลี่ยนข้อความเป็น Reset All

เมื่อกดปุ่ม One Two หรือ Three ให้นับเลขเพิ่มไปเรื่อยๆ และแสดงเลขบน label ที่ตรงกัน
เมื่อกดปุ่ม Reset All ให้ตัวเลขบน label ทุกตัวกลับเป็น 0
ถ้ากดปุ่ม One Two Three ใหม่ เริ่มนับใหม่ที่เลข 1

Quiz this week

- Single View Application TipCalculator 5
  - GUI Textbox, Label, slider, Button 2
  - Connect Action function to View Controller 2
  - เปลี่ยนปุ่ม button Calculate เป็นรูป Picture 1
- **Extra** แบบฝึกหัด กดปุ่มเปลี่ยนตัวเลข 5

References

- **CprE 388 - Mobile Platforms Department of Electrical and Computer Engineering Copyright © 2013, Iowa State University of Science and Technology. All rights reserved.** http://class.ece.iastate.edu/cpre388/
- เอกสารประกอบการสอน Prymania iPhone โดย อ.พราย