Lecture 4-5: Create a draw app template using Auto layout & A1 Tutorials

ADVANCED MOBILE DEVELOPMENT (iOS)

Lecturer: Christine Jiang

Senior Mobile Application Developer at IBM

Lecture 4-5

- Assignment One introduction
- Exercise One:

  Use pan gesture to drag a view.

- Exercise Two:

  Use auto layout to create a draw app template.

  Assignment Tutorials

Assignment One

- Implement a draw application
- Assignment One includes coding, documentation and presentation.
- Suggest to use Swift to do Assignment One because the syntax of Swift is simpler compare to Objective C.

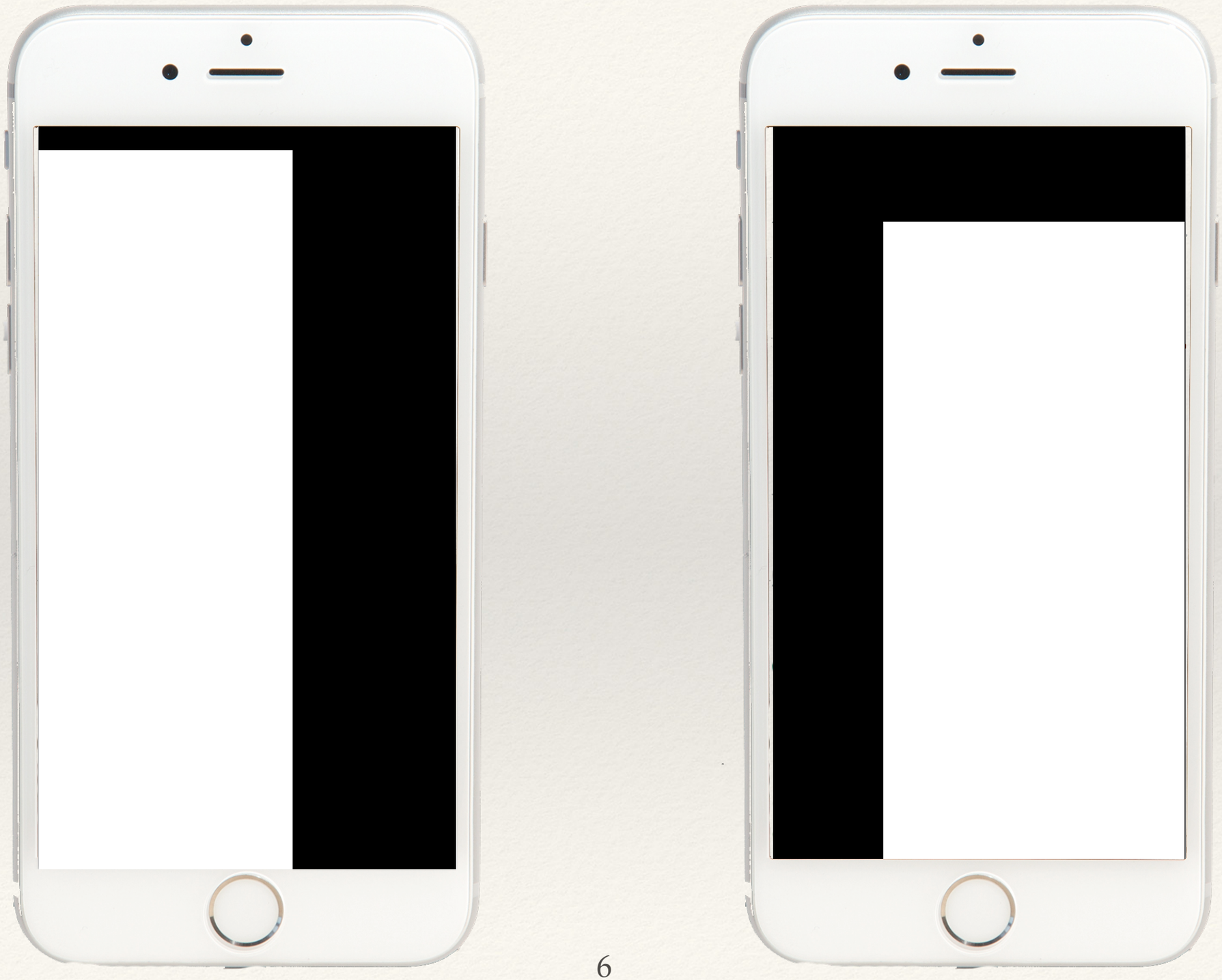
Course Weighting

- Presentation 5%: (Assignment 1)
- Exercises and Assignments: 60%
  - Each class has at least one exercise, this is mandatory.
  - Assignment 1 (35%) & Assignment 2 (25%) show the learning outcome of exercises (coding).
- Group Application: 35%
  - Assignment 3 is group project. 2 people/a team.
  - Assignment 3 shows teamwork and overall leaning outcome.

Exercise One

- Use pan gesture to build the following app

Output



Swift

```
import UIKit

class ViewController: UIViewController {

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

  override func didReceiveMemoryWarning()
  {
    super.didReceiveMemoryWarning()
    // Dispose of any resources that can be recreated.
  }
  @IBAction func handlePan(recognizer: UIPanGestureRecognizer)
  {
    let translation = recognizer.translationInView(self.view)
    if let view = recognizer.view
    {
      view.center = CGPointMake(view.center.x + translation.x, view.center.y +
translation.y)
    }
    recognizer.setTranslation(CGPointZero, inView: self.view)
  }
}
```

Swift Syntax

- Enumeration
- For loop
- Functions

Enumeration

```
enum CompassPoint
{
    case North
    case South
    case East
    case West
}
```

Note:

Unlike C and Objective-C, Swift enumeration cases are not assigned a default integer value when they are created. In the CompassPoint example above, North, South, East and West do not implicitly equal 0, 1, 2 and 3.

Enumeration

```
var directionToHead = CompassPoint.West

directionToHead = .East

directionToHead = CompassPoint.East

directionToHead = .South

switch directionToHead
{
case .North:
  print("Lots of planets have a north")
case .South:
  print("Watch out for penguins")
case .East:
  print("Where the sun rises")
case .West:
  print("Where the skies are blue")
}
```

Enumeration

```
enum Planet
{
  case Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
}
```

=

```
enum Planet
{
  case Mercury
  case Venus
  case Earth
  case Mars
  …
}


var planetName = Planet.Mercury

planetName = .Venus
```

Enumeration - Associated Value

- It is sometimes useful to be able to store *associated values* of other types alongside these case values.
- This enables you to store additional custom information along with the case value, and permits this information to vary each time you use that case in your code.

Enumeration - Associated Value

- For example, suppose an inventory tracking system needs to track products by two different types of barcode. Some products are labeled with 1D barcodes in UPC-A format, which uses the numbers `0` to `9`. Each barcode has a “number system” digit, followed by five “manufacturer code” digits and five “product code” digits.

Enumeration - Associated Value

- Other products are labeled with 2D barcodes in QR code format, which can use any ISO 8859-1 character and can encode a string up to 2,953 characters long:

Enumeration

- It would be convenient for an inventory tracking system to be able to store UPC-A barcodes as a tuple of four integers, and QR code barcodes as a string of any length.

```
enum Barcode
{
    case UPCA(Int, Int, Int, Int)
    case QRCode(String)
}
```

- Define an enumeration type called `Barcode`, which can take either a value of `UPCA` with an associated value of type (`Int`, `Int`, `Int`, `Int`), or a value of `QRCode` with an associated value of type `String`."

```
var productBarcode = Barcode.UPCA(8, 85909, 51226, 3)
```

Enumeration

```
productBarcode = .QRCode("ABCDEFGHIJKLMNOP")

switch productBarcode {
case .UPCA(let numberSystem, let manufacturer, let product, let check):
  print("UPC-A: \(numberSystem), \(manufacturer), \(product), \(check).")
case .QRCode(let productCode):
  print("QR code: \(productCode).")
}
```

Enumeration

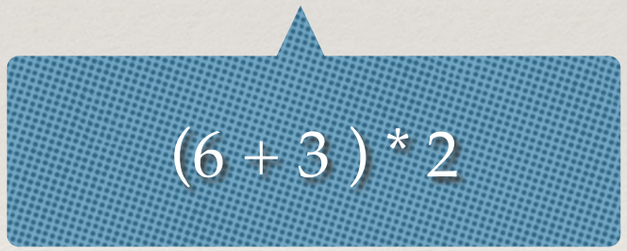
```
enum ASCIIControlCharacter: Character
{
  case Tab = "\t"
  case LineFeed = "\n"
  case CarriageReturn = "\r"
}
```

Enumeration

```
enum Planet: Int
{
  case Mercury = 1, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune
}

let possiblePlanet = Planet(rawValue: 7)
```

Enumeration

```
enum ArithmeticExpression
{
  case Number(Int)
  indirect case Addition(ArithmeticExpression, ArithmeticExpression)
  indirect case Multiplication(ArithmeticExpression, ArithmeticExpression)
}

let six = ArithmeticExpression.Number(6)
let three = ArithmeticExpression.Number(3)
let sum = ArithmeticExpression.Addition(six, three)
let product = ArithmeticExpression.Multiplication(sum, ArithmeticExpression.Number(2))
```



For

```
for index in 1...10
{
  print("\(index) times 10 is \(index * 10)")
}
```

```
 let items = ["Good", "Great", "Better", "Best"]

for item in items
{
  print (item)
}
```

```
 for (index, value) in items.enumerate()
 {
   print("Item \(index + 1): \(value)")
 }
```

Functions

```
func sayHello(personName: String) -> String
{
   let greeting = "Hello, " + personName + "!"
   return greeting
}

print(sayHello("Unitec"))
print(sayHello("Apple"))



func sayHelloWorld() -> String
{
   return "hello, world"
}

print(sayHelloWorld())
```

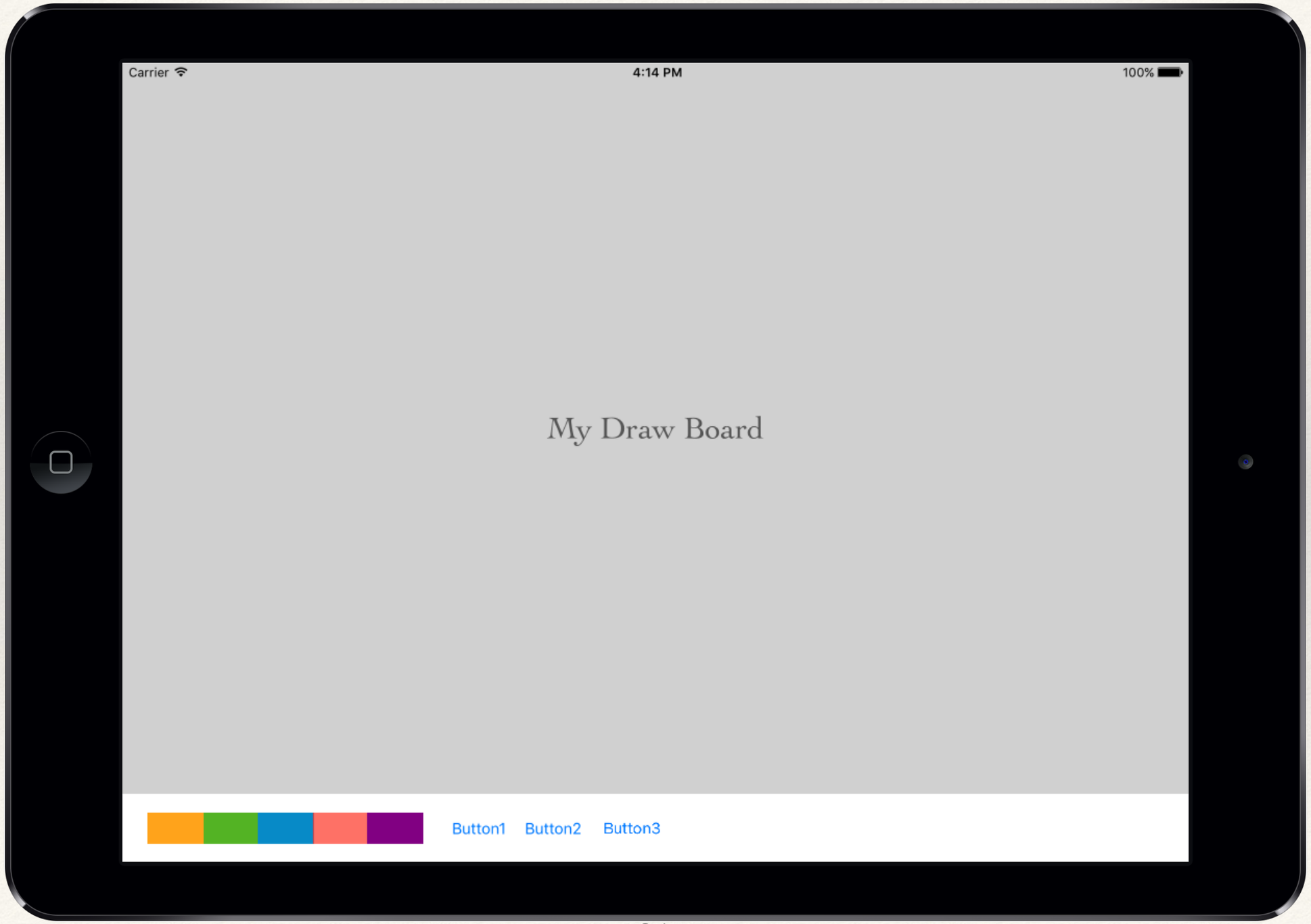
Functions

```
func divideTwoInts(a: Int, _ b: Int) -> Int
{
  return a / b
}

func substractTwoInts(a: Int, _ b: Int) -> Int
{
  return a - b
}

func addTwoInts(a: Int, _ b: Int) -> Int
{
  return a + b
}

func multiplyTwoInts(a: Int, _ b: Int) -> Int
{
  return a * b
}

divideTwoInts(10, 5)
addTwoInts(6, 3)
substractTwoInts(8, 2)
multiplyTwoInts(10, 10)
```

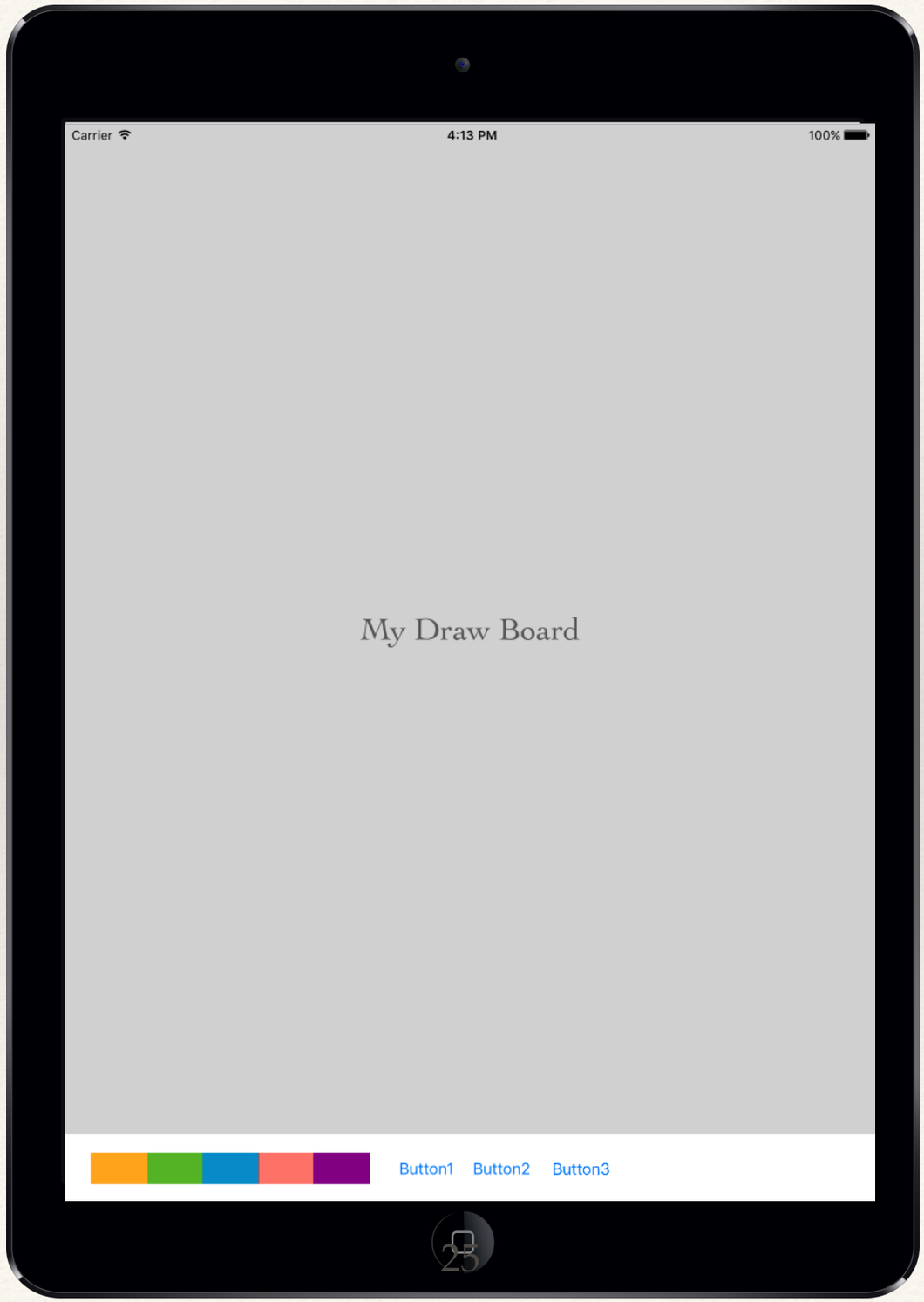
Exercise Two

- Use auto layout to create a draw app template.

Output - Landscape

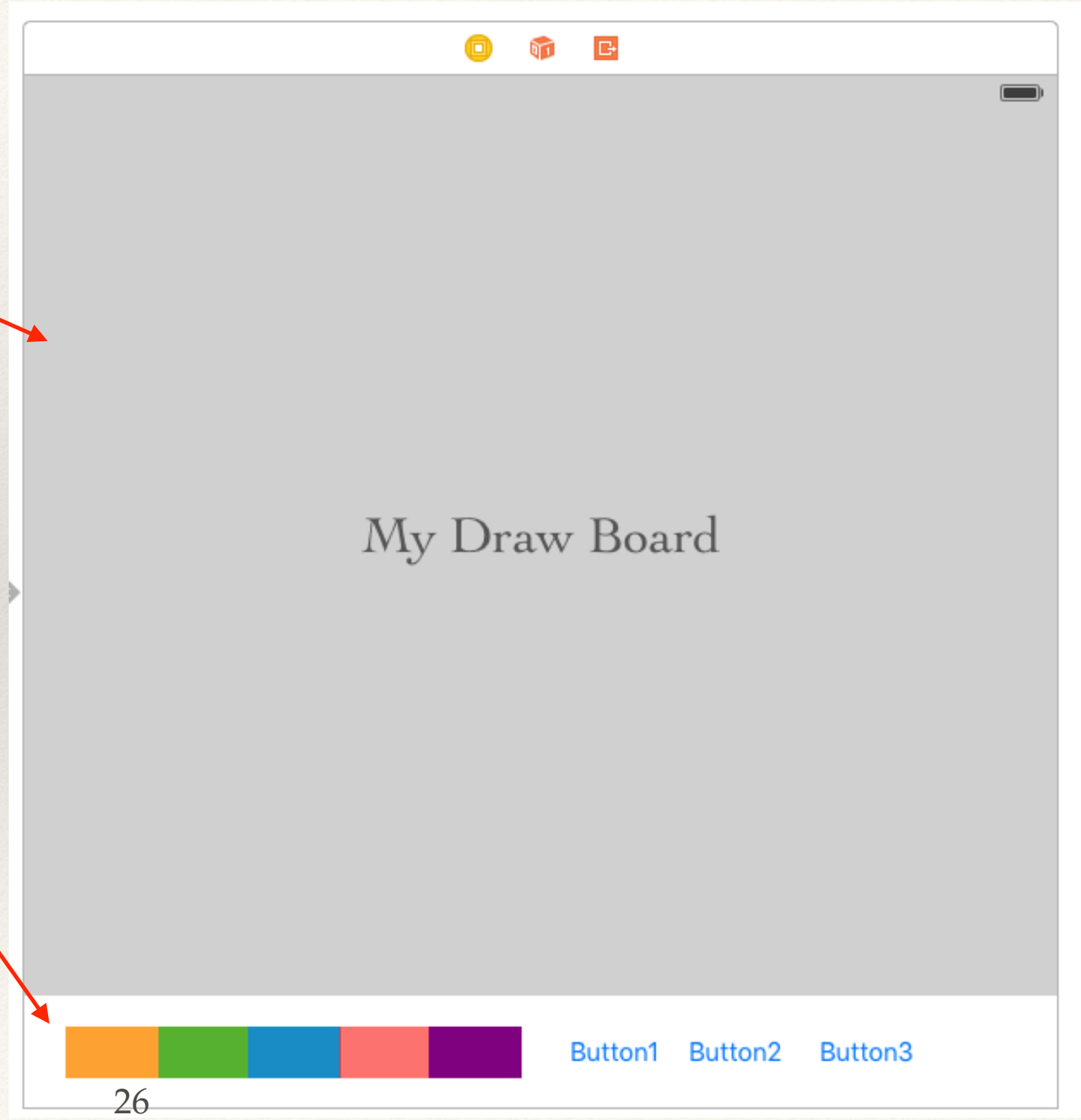


Output - Portrait



Two Sub Views

- Add two views:
- Draw board view
- Tool view



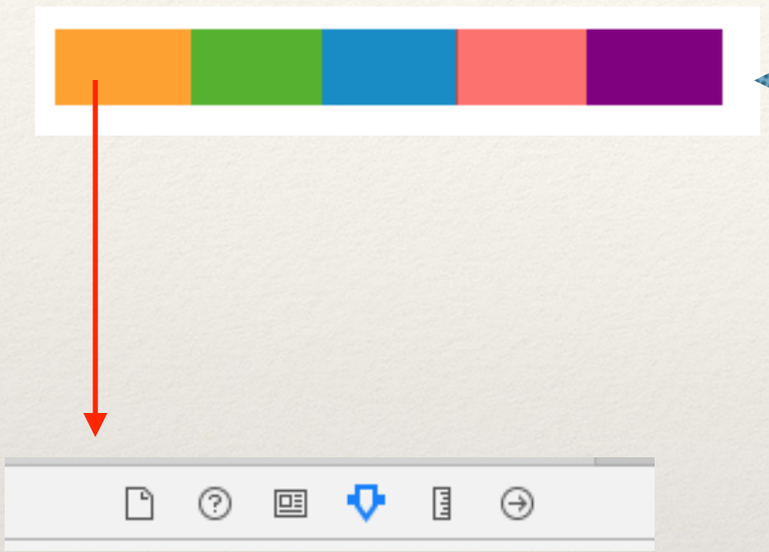
Use Auto Layout

- Add Constraints to Draw board view
  - Trailing Space to Superview
  - Leading Space to SuperView
  - Top Space to SuperView
  - Bottom Space to ToolView

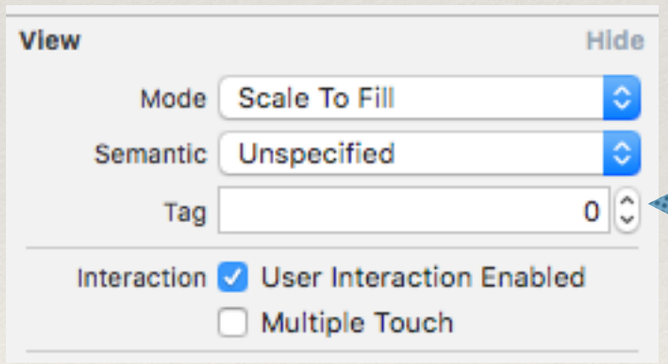
Use Auto Layout

- Add constraints to Tool View
  - Trailing Space to Superview
  - Leading Space to SuperView
  - Height equals: 65
  - Bottom Space to SuperView
  - Top Space to Draw View

Sent SelectColor Event



Set Tag No. from 0 to 1 sequently.



Tag No. 0

Sent Events (1)


```
@IBAction func selectColor(sender: UIButton)
 {
   if sender.tag == 0
   {
     print("yellow")
   }
   else if sender.tag == 1
   {
     print("green")
   }
   else if sender.tag == 2
   {
     print("blue")
   }
   else if sender.tag == 3
   {
     print("pink")
   }
   else if sender.tag == 4
   {
     print("purple")
   }
}
```

Sent Events (2)


```
@IBAction func selectColor(sender: UIButton)
 {
    switch sender.tag
    {
    case 0:
       print("yellow")
      break

    case 1:
      print("green")
      break

    case 2:
      print("blue")
      break

    case 3:
      print("pink")
      break

    case 4:
       print("purple")
      break

    default:
      break
    }
}
```

Sent Events (3)


```
class ViewController: UIViewController
{
  enum Color : Int
  {
    case Yellow = 0
    case Green
    case Blue
    case Pink
    case Purple
  }

  @IBAction func selectColor(sender: UIButton)
  {
    switch sender.tag
    {
    case Color.Yellow.rawValue:
      print("Yellow")
      break

    case Color.Green.rawValue:
      print("Green")
      break

    case Color.Blue.rawValue:
      print("Blue")
      break

    case Color.Pink.rawValue:
      print("Pink")
      break

    case Color.Purple.rawValue:
      print("Purple")
      break

    default:
      print("default")
      break
    }
  }
}
```