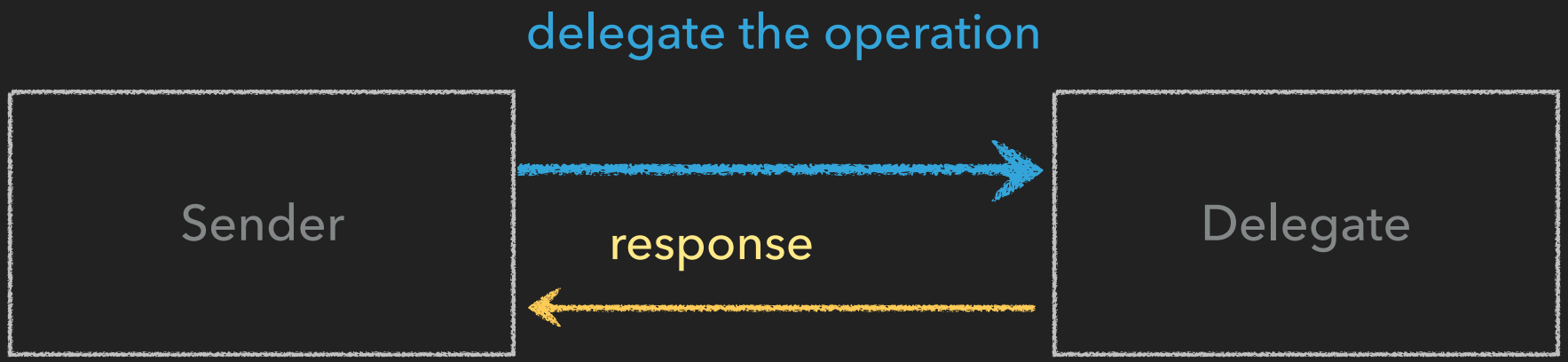
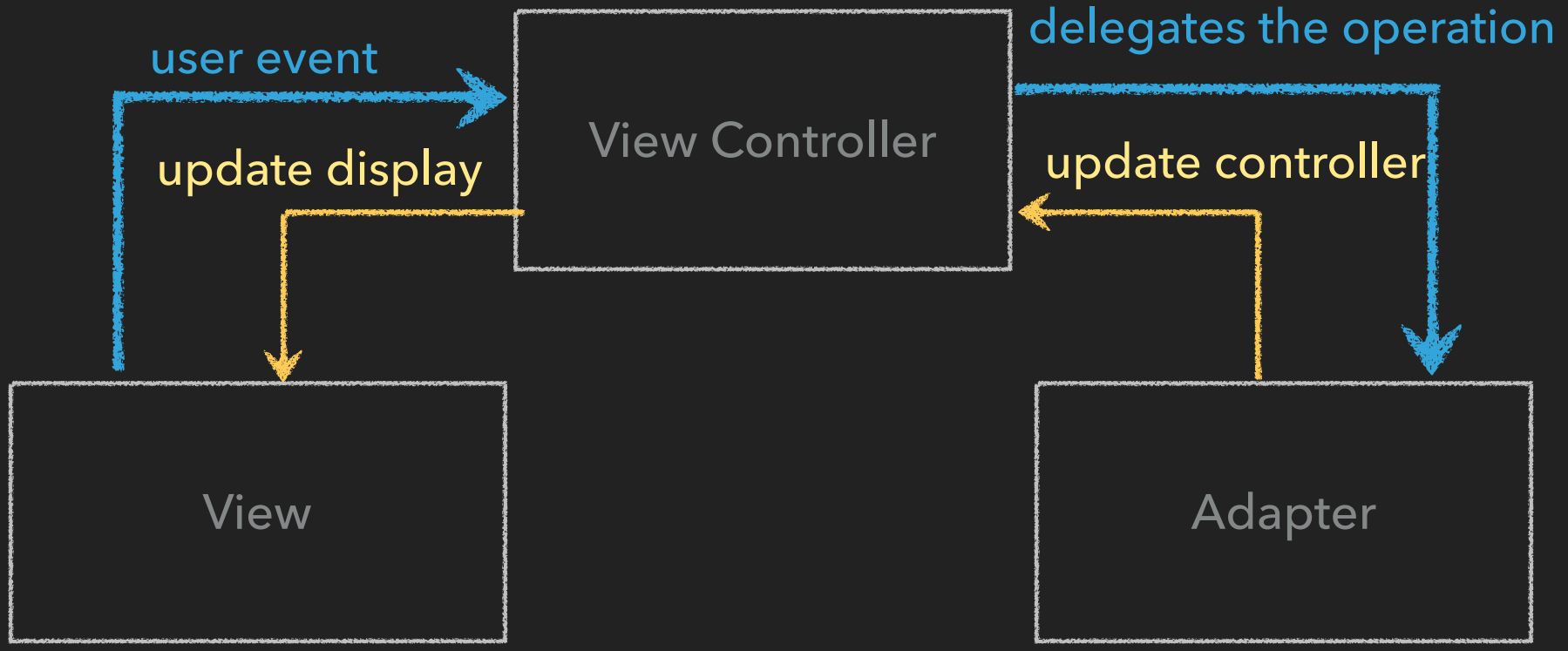
TECH TALK

VIEW PORTS AS CALLBACKS

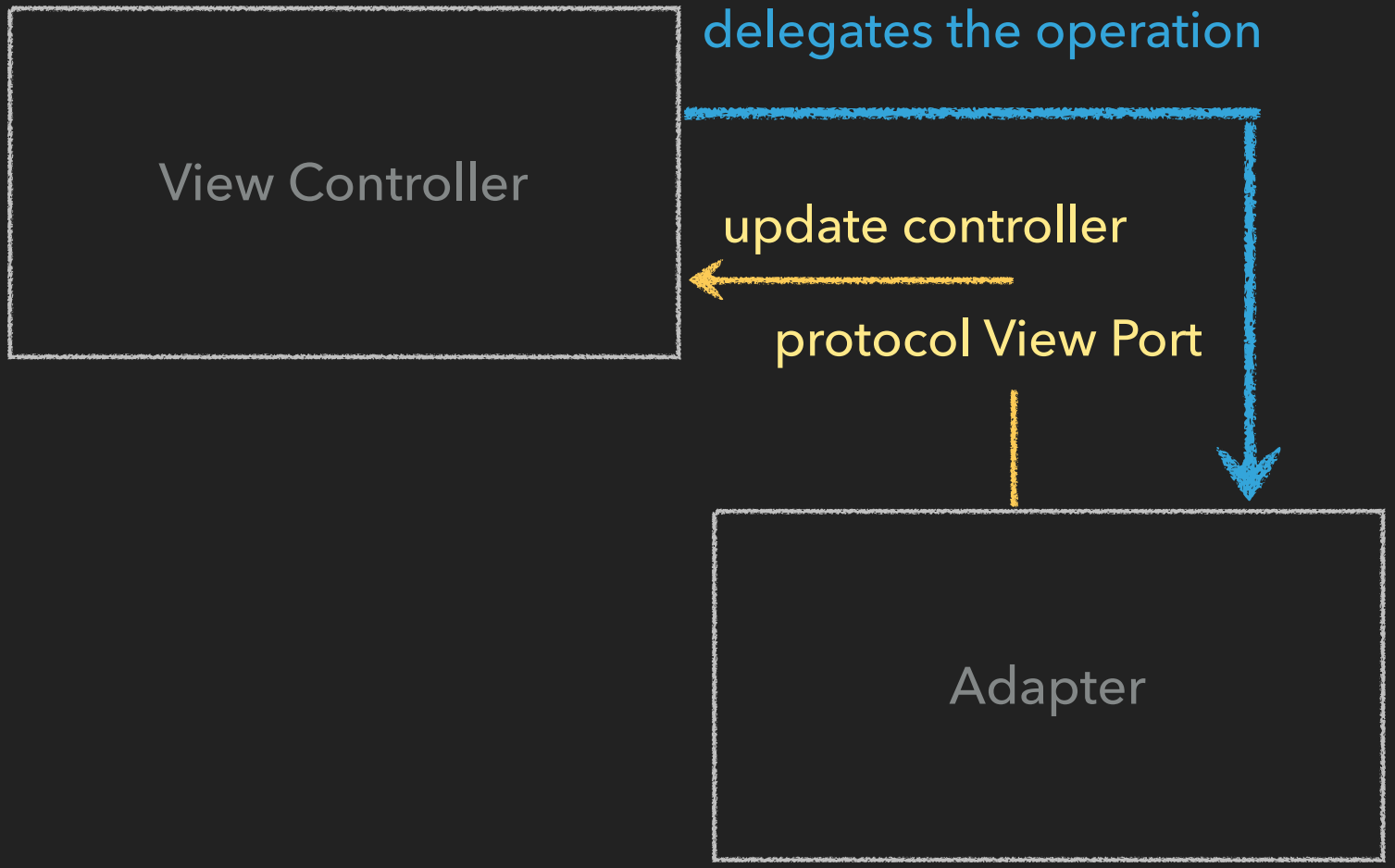
DELEGATION PATTERN



DELEGATION PATTERN W/ A PROTOCOL



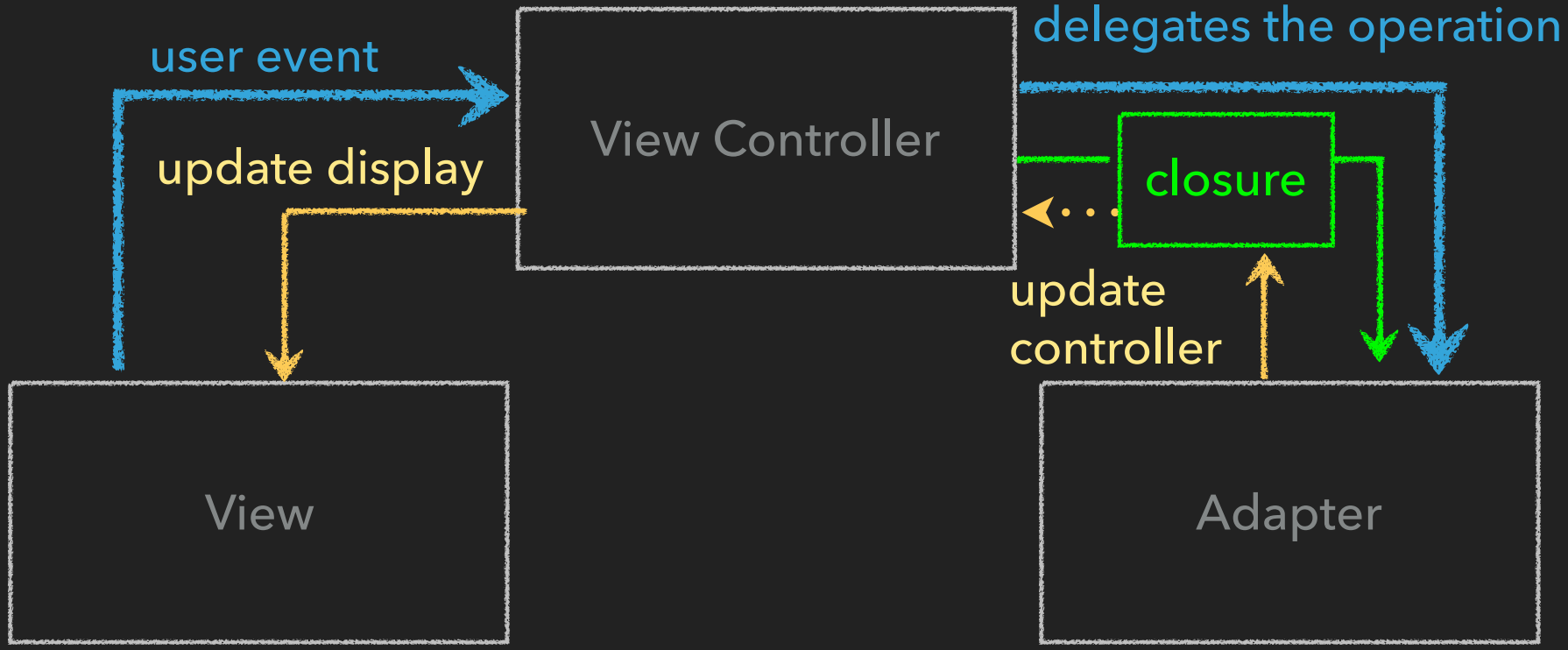
DELEGATION PATTERN W/ A PROTOCOL



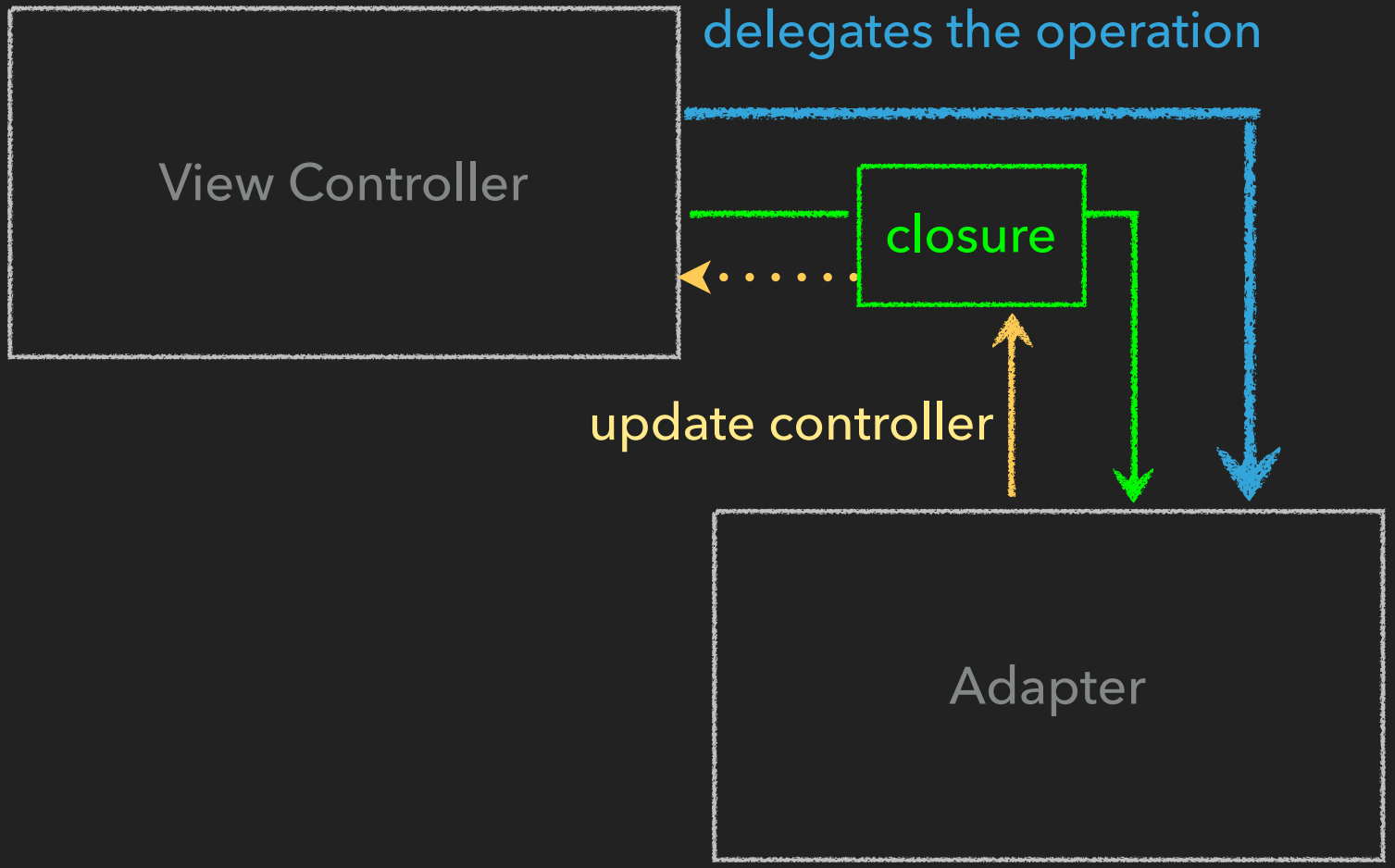
DELEGATION WITH

CALLBACKS

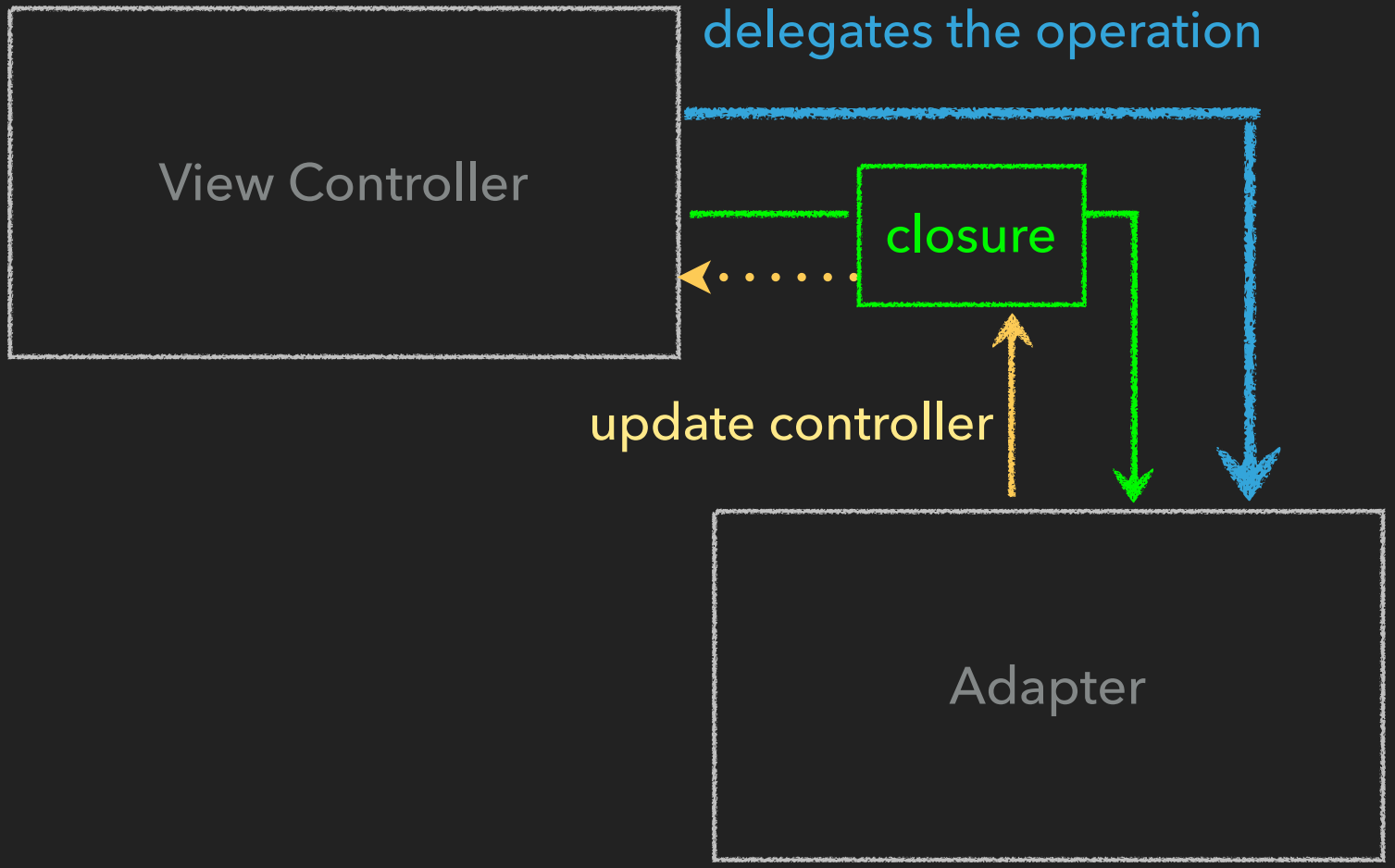
DELEGATION PATTERN USING CALLBACKS/CLOSURE



DELEGATION PATTERN USING CALLBACKS/CLOSURE



DELEGATION PATTERN USING CALLBACKS/CLOSURE



DEMO

Sample App

SAMPLE CODE

DELEGATION W/ PROTOCOLS

ViewController Class

```
36  class ViewController: UIViewController {
37      let button = UIButton()
38      let textlabel = UILabel()
39      lazy var adapter = Adapter(delegate: self as AdapterViewPort)
40      override func viewDidLoad() {
41          super.viewDidLoad()
42          setupView()
43      }
44
45      func buttonTap() {
46          adapter.handleButtonTap(value: textlabel.text ?? "0")
47      }
48  }
49
```

Adapter Class

```
17  class Adapter {
18      weak var delegate:AdapterViewPort?
19
20      init(delegate: AdapterViewPort) {
21          self.delegate = delegate
22      }
23
24      func handleButtonTap(value: String) {
25          let newValue = incrementByOne(input: value)
26          delegate?.updateView(with: newValue)
27      }
28
29      func incrementByOne(input: String) -> String {
30          var inputValue = Int(input) ?? 0
31          inputValue = inputValue + 1
32          return String(inputValue)
33      }
34  }
35
```

View Port Protocol

```
13  protocol AdapterViewPort: class {
14      func updateView(with model: String)
15  }
```

ViewController Class Extension Impementing View Port

```
50
51  extension ViewController: AdapterViewPort {
52      func updateView(with model: String) {
53          textlabel.text = model
54      }
55  }
56
```

Unit Tests

```
class CallbacksTests: XCTestCase {

    var adapter: Adapter!
    var adapterViewPortSpy: AdapterViewPortSpy!

    override func setUp() {
        super.setUp()
        adapterViewPortSpy = AdapterViewPortSpy()
        adapter = Adapter(delegate: adapterViewPortSpy)
    }

    func testHandleTapEventUpdatesTheDisplay() {
        let _ =  adapter.handleButtonTap(value: "0")
        expect(self.adapterViewPortSpy.updateViewWasCalled).to(beTrue())
    }

    func testIncrementByOneComputesValueCorrectly() {
       let result =  adapter.incrementByOne(input: "0")
        expect(result).to(equal("1"))
    }
}
```

View Port Spy

```
13  class AdapterViewPortSpy: AdapterViewPort {
14      var updateViewWasCalled = false
15      func updateView(with model: String) {
16          updateViewWasCalled = true
17      }
18  }
```

Implementing Delegation w/ Protocols

ViewController class

Adapter Class

View Port Protocol

Unit test

View Port Spy

SAMPLE CODE

DELEGATION W/ CALLBACKS

ViewController Class

```
28  class ViewController: UIViewController {
29      let button = UIButton()
30      let textlabel = UILabel()
31      let adapter = Adapter()
32      override func viewDidLoad() {
33          super.viewDidLoad()
34          setupView()
35          setupAdapterCallbacks()
36      }
37
38      func buttonTap() {
39          adapter.handleButtonTap(value: textlabel.text ?? "0")
40      }
41
42      func setupAdapterCallbacks() {
43          adapter.buttonTapProcessComplete = { [weak self] viewModel in
44              guard let strongSelf = self  else { return }
45              strongSelf.textlabel.text = viewModel
46          }
47      }
48  }
```

Adapter Class

```
13  class Adapter {
14      var buttonTapProcessComplete: ((_ viewModel: String) -> Void)?
15
16      func handleButtonTap(value: String) {
17          let newValue = incrementByOne(input: value)
18          buttonTapProcessComplete?(newValue)
19      }
20
21      func incrementByOne(input: String) -> String {
22          var inputValue = Int(input) ?? 0
23          inputValue = inputValue + 1
24          return String(inputValue)
25      }
26  }
```

Unit Tests

```
14
15      var adapter: Adapter!
16
17      override func setUp() {
18          super.setUp()
19          adapter = Adapter()
20      }
21
23      func testHandleTapEventUpdatesTheDisplay() {
23          let _ =  adapter.handleButtonTap(value: "0")
24          var updateViewWasCalled = true
25          adapter.buttonTapProcessComplete = { viewModel in
26              updateViewWasCalled = true
27          }
28          expect(updateViewWasCalled).to(beTrue())
29      }
30
32      func testIncrementByOneComputesValueCorrectly() {
32         let result =  adapter.incrementByOne(input: "0")
33          expect(result).to(equal("1"))
34      }
35  }
```

Implenting Delegation w/ Callbacks

ViewController class

Adapter Class

~~View Port Protocol~~

Unit test

~~View Port Spy~~

Advantages

Faster compile time with lesser files, classes, and other definitions

Easier to TDD w/o need of making spy and implementing protocol

Lesser modification needed when refactoring w/o having view ports as protocols

Optionality

Disadvantages

Readablity and consistency in code may suffer if there is no standard way of defining callbacks

Closures are not easy to understand in glance

Thank you