APP ARCHITECTURE

OBSERVABLES – DIFFICULTY: HARD

INTRODUCTION

Observables form the core of reactive programming, and allow us to bidirectionally connect our data models directly to our views. This means changing a view updates our model and vice versa, rather than us having to write code to monitor both.

THE PROBLEM

It's common to see this sort of code in view controllers:

```
@IBAction func valueChanged(_ sender: UITextField) {
    server.address = sender.text
}

func addressChanged() {
    addressTextField.text =  server.address
}
```

That code isn't necessarily *bad*, but it does cause problems: if you have lots of values moving back and forth it's easy to forget one, and it clutters up your code with what is effectively plumbing.

THE SOLUTION

We can write code to connect a text field and some model data, so that when either changes the other also updates. No, we're not going to switch to RxSwift: we're going to roll our own **Observable** type that is able to handle the two-way communication for us.

```
class Observable<ObservedType> {
    private var _value: ObservedType?
    var valueChanged: ((ObservedType?) -> ())?

    init(_ value: ObservedType) { _value = value }

    public var value: ObservedType? {
        get { return _value }

        set {
            _value = newValue
            valueChanged?(_value)
```

```
        }
    }

    func bindingChanged(to newValue: ObservedType) {
        _value = newValue
    }
}
```

We can then create subclasses of UIKit controls so they can be bound. For example, we could create a bound UITextField subclass like this:

```
class BoundTextField: UITextField {
    var changedClosure: (() -> ())?

    @objc func valueChanged() {
        changedClosure?()
    }

    func bind(to observable: Observable<String>) {
        addTarget(self, action:
#selector(BoundTextField.valueChanged), for: .editingChanged)

        changedClosure = { [weak self] in
            observable.bindingChanged(to: self?.text ?? "")
        }

        observable.valueChanged = { [weak self] newValue in
            self?.text = newValue
        }

        self.text = observable.value
    }
}
```

That's all the code required, so now all that's left is to use the thing: make sure any bound **UITextFields** have the class **BoundTextField**, and make sure any **String** properties are replaced with **Observable<String>** instead.

You can create observables directly from a string, like this: **yourStruct.someProperty = Observable(str)**. And when you want to make a binding between a text field and an observable, call **bind()** like this: **name.bind(to: user.name)**.

THE CHALLENGE

Can you fix SettingsViewController to use observables? I've given a specific solution for text fields above, but with a little work you can create your own bound replacement for **UISegmentedControl** as well.