SWIFT BEST PRACTICES

DELEGATION – DIFFICULTY: EASY

INTRODUCTION

A delegate is one object responding to events on behalf of another. A familiar example is **UITableView**: when you assign its **dataSource** and **delegate** properties the table view asks your object what it should show, and tell your object when events happen.

Think about it: when did you last subclass **UITableView**? You probably haven't, because it isn't needed – using delegation your custom object and the table view work together as one, even though the table view has no knowledge of what your object is or how it works.

THE PROBLEM

Delegates allow one object to act on behalf of another, but in practice we often see this:

```
class MegaController: UIViewController, UITableViewDataSource,
UITableViewDelegate, UIPickerViewDataSource, UITextFieldDelegate,
WKNavigationDelegate, URLSessionDownloadDelegate {
```

If someone asked you what the class did you, could you answer in one breath?

THE SOLUTION

Delegation can often be handled by individual objects. In the example of table views, we can create dedicated data source classes like this:

```
class ProjectDataSource: UITableViewDataSource {
   var data = [String]()

      func tableView(_ tableView: UITableView, cellForRowAt indexPath:
      IndexPath) -> UITableViewCell {
         // etc
      }
}
```

We can then store an instance of that as a property in your view controller:

```
let dataSource = ProjectDataSource()
```

Then assign it to your table view in **viewDidLoad()**:

```
tableView.dataSource = dataSource
```

THE CHALLENGE

Our sample app has two problems that better delegation can fix:

1. The **ReadViewController** class conforms to **WKNavigationDelegate** so that it can act as a navigation controller for its web view. This forces the view controller to act in a specific way, and stops us from reusing that delegate in other view controllers.
2. The **ViewController** class acts as data source for its table view, which means it has lots of code for preparing table view cells.

Can you carve both of them off into separate objects?

- Make sure you store instances of your navigation class and data source class as properties in your view controller, because delegates are weak by default and will be freed unless you store them.

- To cleanly separate the data source from ViewController it should also be responsible for loading and storing its data.

- Table view data sources are relatively easy to split into a separate object, but table view delegates are much harder – you should probably leave the delegate alone.