DRY Data Sources

December 15, 2016

Massive View Controllers

UITableViewDataSource

```
func numberOfSections(in: UITableView)

func tableView(UITableView, numberOfRowsInSection: Int)

func tableView(UITableView, cellForRowAt: IndexPath)
```

UICollectionViewDataSource

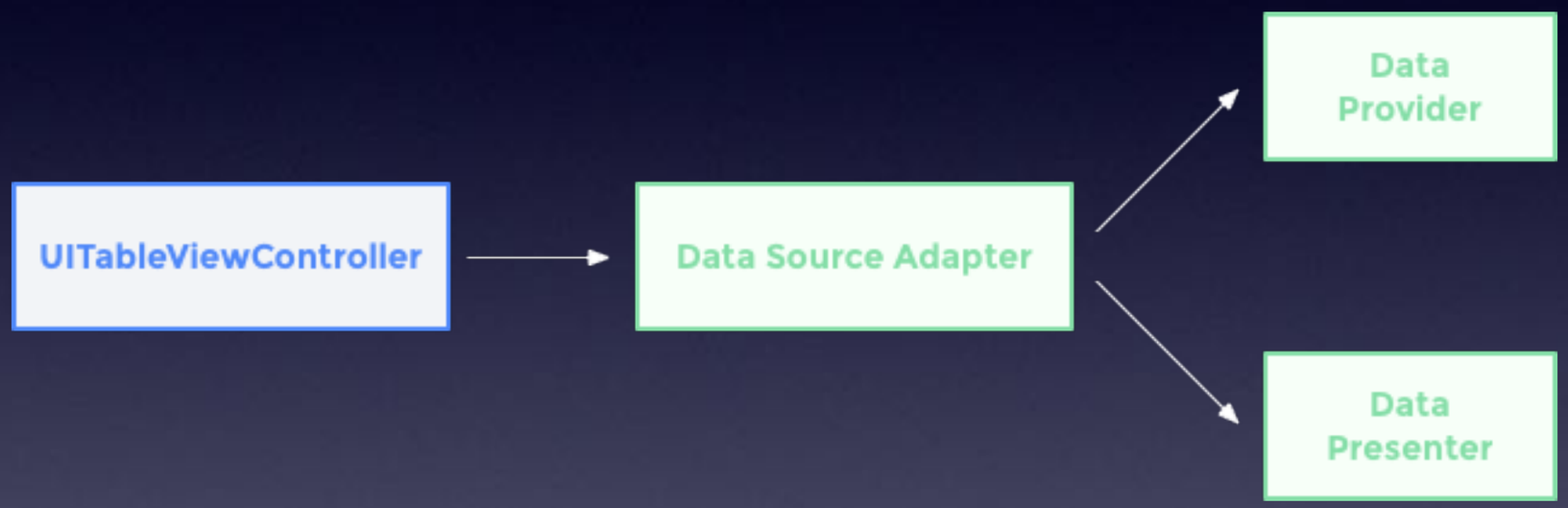
```
func numberOfSections(in: UICollectionView)

func collectionView(UICollectionView, numberOfItemsInSection: Int)

func collectionView(UICollectionView, cellForItemAt: IndexPath)
```

Don’t Repeat Yourself

But how can you keep your data sources DRY?

The Responsibilities of UIDataSource

- Provisioning data
- Preparing data for presentation


UITableViewController
Data Source Adapter
Data Provider
Data Presenter

Part 1

Data Provisioning

- What do we want to provide?
- Where does the data come from?

FRKDataProvider

```
@protocol FRKDataProvider <NSObject>

- (id)itemAtIndexPath:(NSIndexPath *)indexPath;
- (NSIndexPath *)indexPathForItem:(id)item;

- (NSInteger)numberOfSections;
- (NSInteger)numberOfItemsInSection:(NSInteger)section;

- (void)loadDataWithCompletion:(void(^)())completion;

@end
```

FRKArrayDataProvider

```objc
@implementation FRKArrayDataProvider

- (instancetype)initWithItems:(NSArray *)itemsArray {
    self = [super init];
    if (!self) {
        return nil;
    }

    _items = itemsArray;

    return self;
}

- (void)loadDataWithCompletion:(void (^)(NSArray *errors))completion {
    if (completion) {
        completion(nil);
    }
}

@end
```

FRKArrayDataProvider

```objc
@implementation FRKArrayDataProvider

- (id)itemAtIndexPath:(NSIndexPath *)indexPath {
    return self.items[indexPath.item];
}

- (NSIndexPath *)indexForItem:(id)item {
    NSInteger index = [self.items indexOfObject:item];

    if (index != NSNotFound) {
        return [NSIndexPath indexPathForItem:index inSection:0];
    } else {
        return nil;
    }
}

- (NSInteger)numberOfSections {
    if (self.items.count > 0) {
        return 1;
    } else {
        return 0;
    }
}

- (NSInteger)numberOfItemsInSection:(NSInteger)section {
    return self.items.count;
}

@end
```

Part 2

Data Presenting

- Separate interface for UICollectionView and UITableView

- Register cells

- Prepare cells for display

FRKTableViewDataPresenter

```
@protocol FRKTableViewDataPresenter <NSObject>

- (void)registerCellsForTableView:(UITableView *)tableView;

- (UITableViewCell *)tableView:(UITableView *)tableView
                 cellForItem:(id)item
                 atIndexPath:(NSIndexPath *)indexPath;

@end
```

FRKTableViewDataPresenter

```
@implementation ExpansionDataPresenter

- (void)registerCellsForTableView:(UITableView *)tableView {
    [tableView registerClass:[NDKExpansionTableViewCell class] forCellReuseIdentifier:@"expansionCell"];
}

- (UITableViewCell *)tableView:(UITableView *)tableView
                  cellForItem:(id)item
                  atIndexPath:(NSIndexPath *)indexPath {
    NDKExpansionTableViewCell *cell = [tableView dequeueReusableCellWithIdentifier:@"expansionCell"
                                                                      forIndexPath:indexPath];
    [cell updateWithExpansion:item];

    return cell;
}

@end
```

Part 3

Data Adapters

- Adapter for UICollectionViewDataSource and UITableViewDataSource
- Thin layer between what we have and what UIKit wants
- Reusable

FRKTableViewAdapter

```
@interface FRKTableViewAdapter : NSObject <UITableViewDataSource, FRKDataProvider>

- (instancetype)initWithProvider:(id<FRKDataProvider, FRKSectionInfoProvider>)provider
                       presenter:(id<FRKTableViewDataPresenter>)presenter;

- (void)registerCellsForTableView:(UITableView *)tableView;

@end
```

FRKTableViewAdapter

```
- (id)itemAtIndexPath:(NSIndexPath *)indexPath {
    return self.provider;
}

- (NSIndexPath *)indexPathForItem:(id)item {
    return [self.provider indexPathForItem:item];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
    return [self.provider numberOfSections];
}

- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {
    return [self.provider numberOfItemsInSection:section];
}
```

FRKTableViewAdapter

```
-(UITableViewCell *)tableView:(UITableView *)tableView
        cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    id item = [self.provider itemAtIndexPath:indexPath];

    return [self.presenter tableView:tableView
                            cellForItem:item
                            atIndexPath:indexPath];
}
```

So what does this look like?

Example 1: Array

```
NSArray *decks = //Create array or something
FRKArrayDataProvider *provider = [[FRKArrayDataProvider alloc] initWithItems:decks];
DecksDataPresenter *presenter = [[DecksDataPresenter alloc] init];
FRKTableViewAdapter *adapter = [[FRKTableViewAdapter alloc] initWithDataProvider:provider
                                                                    dataPresenter:presenter];

DecksTableViewController *controller = [[DecksTableViewController alloc] initWithDataSource:adapter];
```

Example 2: Core Data

```
NSFetchRequest *fetchRequest = //Create fetch request
FRKCoreDataProvider *provider = [[FRKCoreDataProvider alloc] initWithFetchRequest:fetchRequest
                                                                               moc:self.childMOC
                                                                   sectionKeypath:@"type"];
ExpansionDataPresenter *presenter = [[ExpansionDataPresenter alloc] init];
FRKTableViewAdapter *adapter = [[FRKTableViewAdapter alloc] initWithDataProvider:provider
                                                                  dataPresenter:presenter];

ExpansionsController *expansionsController = [[ExpansionsController alloc] initWithDataSource:adapter];
```

Example 3: Network

```
FetchDecksOperation *fetchDecksOp = // NSOperations!
FRKArrayDataProvider *provider = [[FRKAPIDataProvider alloc] initWithOperation:fetchDecksOp];
DecksDataPresenter *presenter = [[DecksDataPresenter alloc] init];
FRKTableViewAdapter *adapter = [[FRKTableViewAdapter alloc] initWithDataProvider:provider
                                                                      dataPresenter:presenter];

DecksTableViewController *controller = [[DecksTableViewController alloc] initWithDataSource:adapter];
```

Inside our Controller

```
- (instancetype)initWithDataSource:(FRKTableViewAdapter *)dataSource {
    self = [super initWithStyle:UITableViewStyleGrouped];
    if (!self) {
        return nil;
    }

    _dataSource = dataSource;

    return self;
}

- (void)viewDidLoad {
    [super viewDidLoad];

    self.tableView.dataSource = self.dataSource;

    [self.dataSource registerCellsForTableView:self.tableView];

    [self.dataSource loadDataWithCompletion:^(NSArray *errors) {
        [self.tableView reloadData];
    }];
}
```

What we've gained

- We can build data sources with the capabilities we need without repeating ourselves
- No magic 🙅‍♀️🙅
- Every piece is easy to test in isolation
- Shallow inheritance tree