Advanced Mobile Application Development
Week 6: Collection Views

pictureCollection
File | New Project
Single View App
Universal

Delete ViewController.swift
Delete the view controller in the storyboard
Create a new Cocoa Touch class called CollectionViewController and subclass UICollectionViewController.
We could have also changed the superclass in the file that the template created but this gives us all the stub methods we'll need.
The delegate methods included are very similar to the table view methods we've been using.

Add a new Collection view controller in the storyboard and check Is Initial View Controller.
Change its class to be our new CollectionViewController class.
Note that the UICollectionViewController has also brought with it a collection view and a prototype collection view cell.
Select the collection view and in the connections inspector make sure the delegate and data source are connected to the Collection View Controller.

Create a new Cocoa Touch class called CollectionViewCell and subclass UICollectionViewCell.
In the storyboard click on the collection view cell and change its class to CollectionViewCell.
Also give it a reuse identifier "Cell".(must match the constant in CollectionViewController)
Since our cells are going to hold images, make the prototype cell a bit larger and drag an image view into it.
For the cell size choose custom and make the size 104x104. Make the image view 100x100.
With the Image View selected in the storyboard scene Pin the Spacing to nearest neighbor constraints on all four sides of the view to 2 with the Constrain to margins option unchecked. Create 4 constraints.
This should provide 2 points between the image view and the cell on all sides (this is important for later)
Create a connection from the image view called imageView but make sure you're connecting to the CollectionViewCell class, NOT the view controller class. If you get an error and it's the wrong class delete the connection and the variable and redo to the right class.

Images
Define an array to hold the images
```
var expoImages=[String]()
```

Comment out this line as we set the reuse identifier in the storyboard.
```
self.collectionView!.registerClass(UICollectionViewCell.self,
forCellWithReuseIdentifier: reuseIdentifier)
```

Make sure you define a constant for reuseIdentifier.
```
let reuseIdentifier = "Cell"
```

Update viewDidLoad() to load the image names into the array. I named all my images the same so I could use a for loop.

```
        for i in 0...19{
            expoImages.append("atlas" + String(i+1))
        }
```

Data source methods

Update the data source methods. For now we have one section.

```
    override func numberOfSectionsInCollectionView(collectionView:
UICollectionView) -> Int {
        // #warning Incomplete implementation, return the number of sections
        return 1
    }

    override func collectionView(collectionView: UICollectionView,
numberOfItemsInSection section: Int) -> Int {
        // #warning Incomplete implementation, return the number of items
        return expoImages.count
    }

    override func collectionView(collectionView: UICollectionView,
cellForItemAtIndexPath indexPath: NSIndexPath) -> UICollectionViewCell {
        let cell =
collectionView.dequeueReusableCellWithReuseIdentifier(reuseIdentifier,
forIndexPath: indexPath) as! CollectionViewCell

        // Configure the cell
        let image = UIImage(named: expoImages[indexPath.row])
        cell.imageView.image = image
        return cell
    }
```

Run the app and the images should appear. Because each cell is a fixed size the images are compressed to fit so we'll fix that next.

Cell Item Size

All the images are a different size so we need to set the size of each cell to match the image.
We can use a method in the UICollectionViewDelegateFlowLayout protocol to get the size of each image.
Adopt the UICollectionViewDelegateFlowLayout protocol

```
class CollectionViewController: UICollectionViewController,
UICollectionViewDelegateFlowLayout
```

Implement the method that gets the size of the item's cell.

```
    func collectionView(collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, sizeForItemAtIndexPath
indexPath: NSIndexPath) -> CGSize {
        let image = UIImage(named: expoImages[indexPath.row])
        // code to create resized image
        let newSize:CGSize = CGSize(width: (image?.size.width)!/40, height:
(image?.size.height)!/40)
        let rect = CGRectMake(0, 0, newSize.width, newSize.height)
        UIGraphicsBeginImageContextWithOptions(newSize, false, 1.0)
        image?.drawInRect(rect)
        let image2 = UIGraphicsGetImageFromCurrentImageContext()
        UIGraphicsEndImageContext()
        //end resizing
        return (image2?.size)!
    }
```

Scroll Direction

If you want to change the scroll direction add this to viewDidLoad()

```
        let layout = UICollectionViewFlowLayout()
        layout.scrollDirection = UICollectionViewScrollDirection.Horizontal
        self.collectionView?.setCollectionViewLayout(layout, animated: true)
```

Header

In the storyboard click on the Collection View and in the attributes inspector next to Accessories check Section Header.
This adds a section header that you can configure.
Add a label and use auto layout if needed.
Because the background is black, make the text of the label white.
Now we need a class to control the new header.
File | New File | Cocoa Touch class called CollectionSupplementaryView and subclass UICollectionReusableView.
Then go back into the storyboard, select the collection header view and change its class to the class you just created CollectionSupplementaryView
Also give the header a reuse identifier "Header"
Create an outlet for the label called headerLabel. Make sure you're connecting it to the CollectionSupplementaryView class.

Go into CollectionViewController.swift and implement the following data source method.

```
    override func collectionView(collectionView: UICollectionView,
viewForSupplementaryElementOfKind kind: String, atIndexPath indexPath:
NSIndexPath) -> UICollectionReusableView {
        var header: CollectionSupplementaryView?
        if kind == UICollectionElementKindSectionHeader{
            header =
collectionView.dequeueReusableSupplementaryViewOfKind(kind,
withReuseIdentifier: "Header", forIndexPath: indexPath) as?
CollectionSupplementaryView
            header?.headerLabel.text = "Fall 2015"
        }
        return header!
    }
```

Now when you run it you should see your header.
You can do the same thing for a footer and use the same method but check for UICollectionElementKindSectionFooter to configure it.

Spacing

You can change the spacing between cells, headers, and footers using UIEdgeInsets.

In your class definition add

```
let sectionInsets = UIEdgeInsets(top: 25.0, left: 10.0, bottom: 25.0, right:
10.0)
```

Then add the following UICollectionViewDelegateFlowLayout method to return margins for the cells.

```
    func collectionView(collectionView: UICollectionView, layout
collectionViewLayout: UICollectionViewLayout, insetForSectionAtIndex
section: Int) -> UIEdgeInsets {
        return sectionInsets
    }
```

<u>Detail</u>
Now let's add a detail view to show the image larger when the user taps on a cell.
Embed the collection view controller in a navigation controller.
In the navigation bar make the title ATLAS Expo.
Drag a view controller into the storyboard.
Add an image view to take up the full view, all the way up to the navigation bar.
Change its mode to Aspect Fit (or fill)
Add missing constraints for the view controller.
Create a show segue from the collection view cell to the new view controller. Make sure the connection is from the cell, use the document outline to be sure.
Give the segue an identifier called showDetail.
Add a new Cocoa touch class called DetailViewController and subclass UIViewController.
Back in the storyboard make this the class for the new view controller.
Go into the assistant editor and connect the image view as an outlet called imageView. Make sure you make the connection to DetailViewController.swift

In DetailViewController.swift add a string to hold the name of the image passed to this view.
```
var imageName : String?
```

In viewDidLoad() add the following to populate the image view with the image the user selected

```
if let name = imageName {
    imageView.image = UIImage(named: name)
}
```

In ViewController.swift we need to figure out which cell the user selected and then pass that data to DetailViewController when the segue occurs.

```
    override func prepareForSegue(segue: UIStoryboardSegue, sender:
AnyObject?) {
        if segue.identifier == "showDetail"{
            let indexPath = collectionView?.indexPathForCell(sender as!
CollectionViewCell)
            let detailVC = segue.destinationViewController as!
DetailViewController
           detailVC.imageName = expoImages[(indexPath?.row)!]
        }
    }
```

Sharing

Now we want to be able to select multiple images and share them.
The trick here is going to be knowing if we should segue to our detail view controller or let the user select, and deselect, multiple images to share.
In the storyboard drag a bar button item onto the right side of the navigation bar of the collection view.
Set the identifier to action to get the sharing icon.
Create an action for the button called share(). Make sure you are connecting the button to the CollectionViewController class.

The rest of the work for this will be in CollectionViewController.swift
In the class add the following to track if it's in sharing mode and an array for the selected images

```
    var sharing = false
    var selectedImages = [String]()

    @IBAction func share(sender: AnyObject) {
        sharing = !sharing
        collectionView?.allowsMultipleSelection = sharing
        collectionView?.selectItemAtIndexPath(nil, animated: true, 
scrollPosition: .None)

        if !sharing {
            for cell in (collectionView?.visibleCells())!{
                cell.backgroundColor = UIColor.blackColor()
            }
            selectedImages.removeAll(keepCapacity: false)
        }
    }
```

Uncomment and implement the following delegate method.
This handles the issue of adding the image name to our array if in sharing mode and calling our segue if we're not in sharing mode.
We return false because otherwise it lets you select the cell which calls the segue automatically.
We're also setting the background color to yellow so the user can tell which images they've selected.
You'll only see this if you left a little room around your image to the cell size.

```
    override func collectionView(collectionView: UICollectionView, 
shouldSelectItemAtIndexPath indexPath: NSIndexPath) -> Bool {
        if sharing {
            let image = expoImages[indexPath.row]
            if let foundIndex = 
selectedImages.indexOf(expoImages[indexPath.row]) {
                //deselect and remove
                selectedImages.removeAtIndex(foundIndex)

collectionView.cellForItemAtIndexPath(indexPath)?.backgroundColor = 
UIColor.blackColor()
            } else {
                //add and select
                selectedImages.append(image)
collectionView.cellForItemAtIndexPath(indexPath)?.backgroundColor =
UIColor.yellowColor()
           }
     }
        else {
           //segue to detail view controller
            self.performSegueWithIdentifier("showDetail", sender:
collectionView.cellForItemAtIndexPath(indexPath))
        }
        return false
    }
```

If you run at this point you should be able to still go to the detail view, or go into sharing mode and select multiple images.

Now as the last step we're ready to implement sharing. Add this at the top of share()

```
        var imageArray = [UIImage]()
        if !selectedImages.isEmpty {
            //share images
            print("share images")
            for imageName in selectedImages{
                imageArray.append(UIImage(named: imageName)!)
            }
            let shareScreen = UIActivityViewController(activityItems:
imageArray, applicationActivities: nil)
            shareScreen.modalPresentationStyle = .Popover
            shareScreen.popoverPresentationController?.barButtonItem =
sender as? UIBarButtonItem
            presentViewController(shareScreen, animated: true, completion:
nil)
```