# iOS Dev Accelerator
# Week2 Day1

- UIAlertController
- Collection Views
- Asset Catalogs
- Custom Delegate Protocol
- UIImagePickerController
- Mobile Monday: Android View Controllers and Table Views

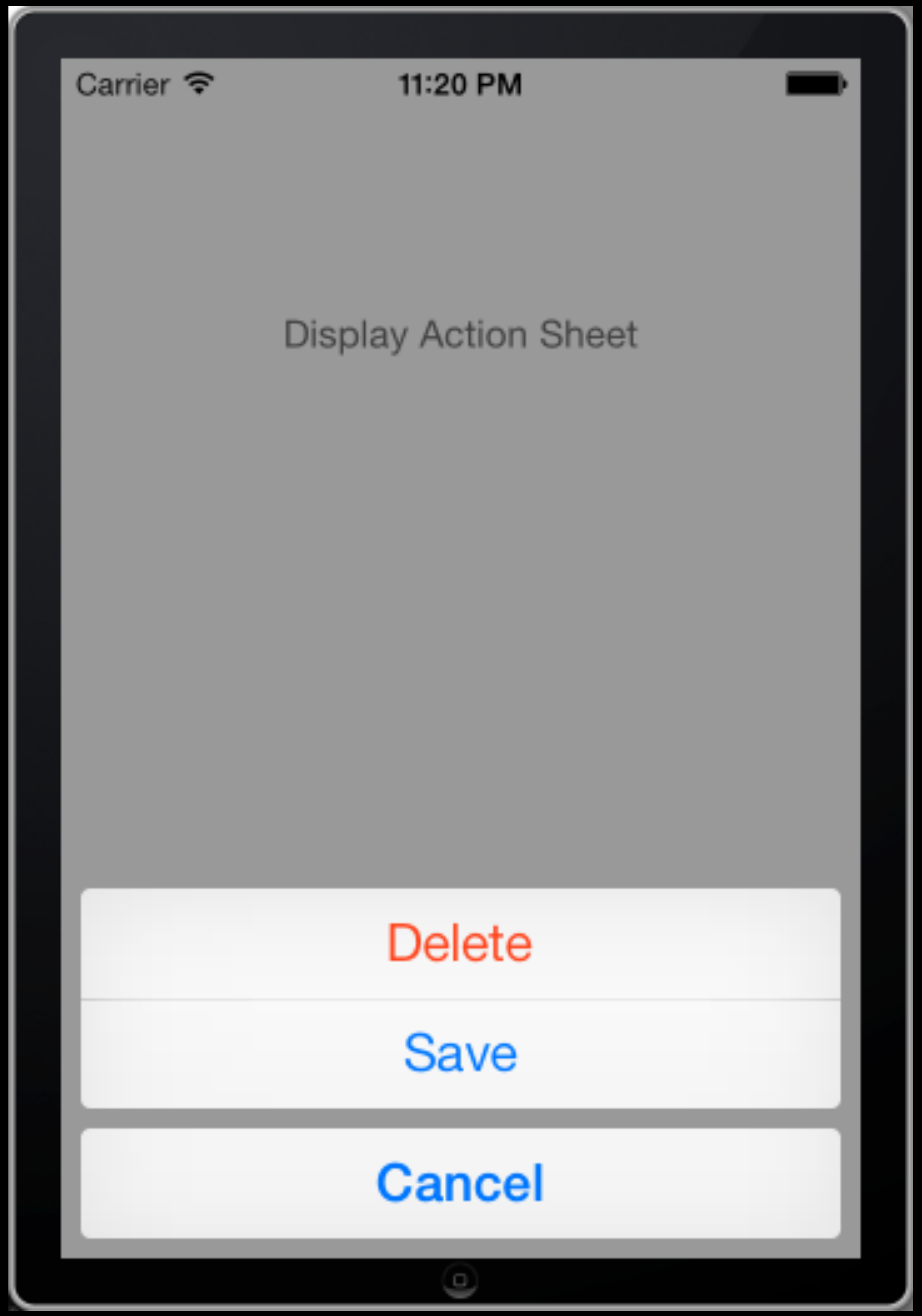
# The MVC layout of our Week 2 App

Controller Layer
Model Layer
View Layer
Filter
CoreData
Filter Thumbnail
Gallery ViewController
PhotosFW ViewController
Home ViewController
UIImage Picker Controller
AVFoundation ViewController
Storyboard
Photo CollectionView Cell
FilterThumbnail CollectionView Cell

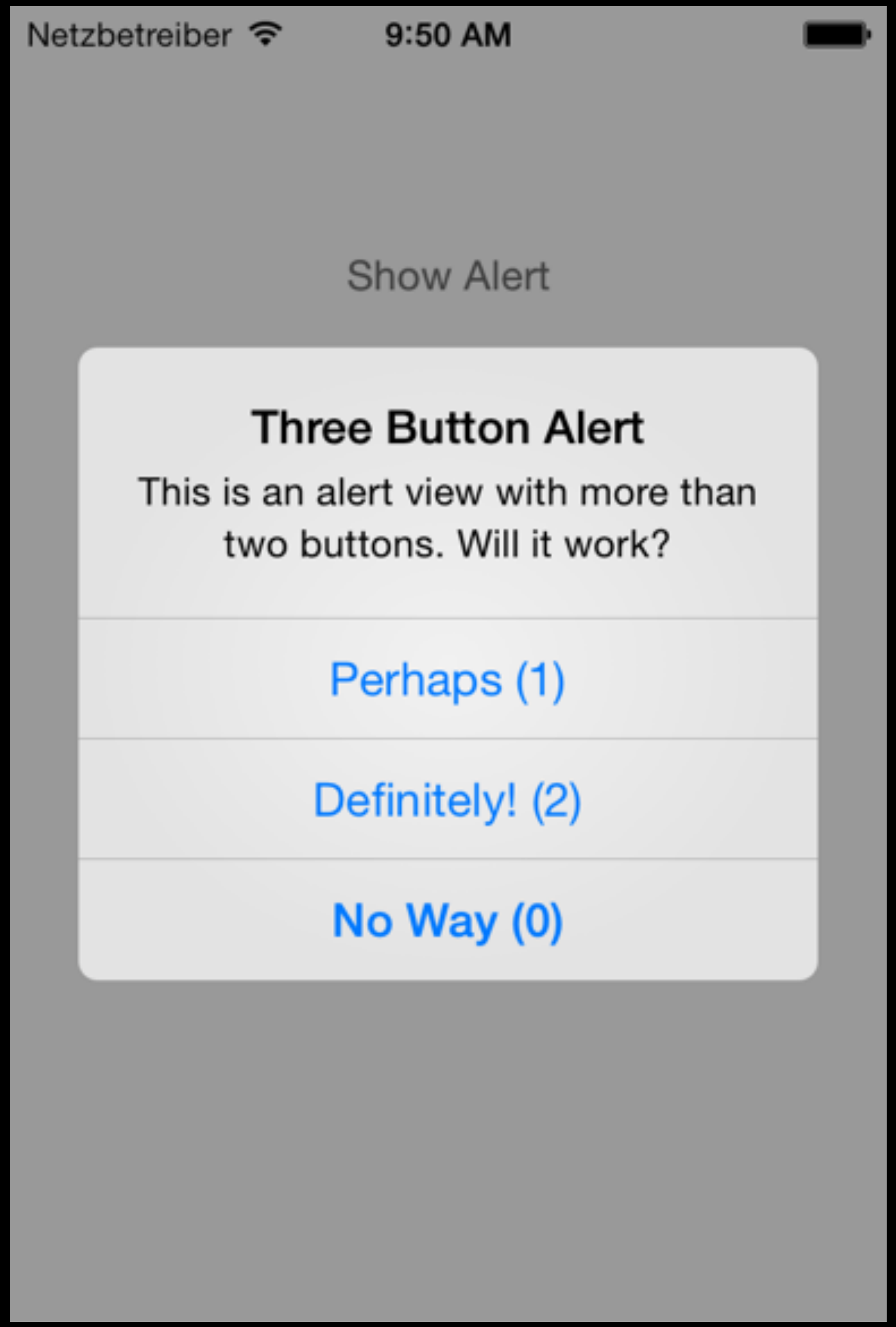
## UIAlertController

# UIAlertController

- "UIAlertController object displays an alert message to the user"
- Replaces both UIActionSheet and UIAlertView in iOS8
- After configuring the Alert Controller present it with presentViewController:animated: Completion:

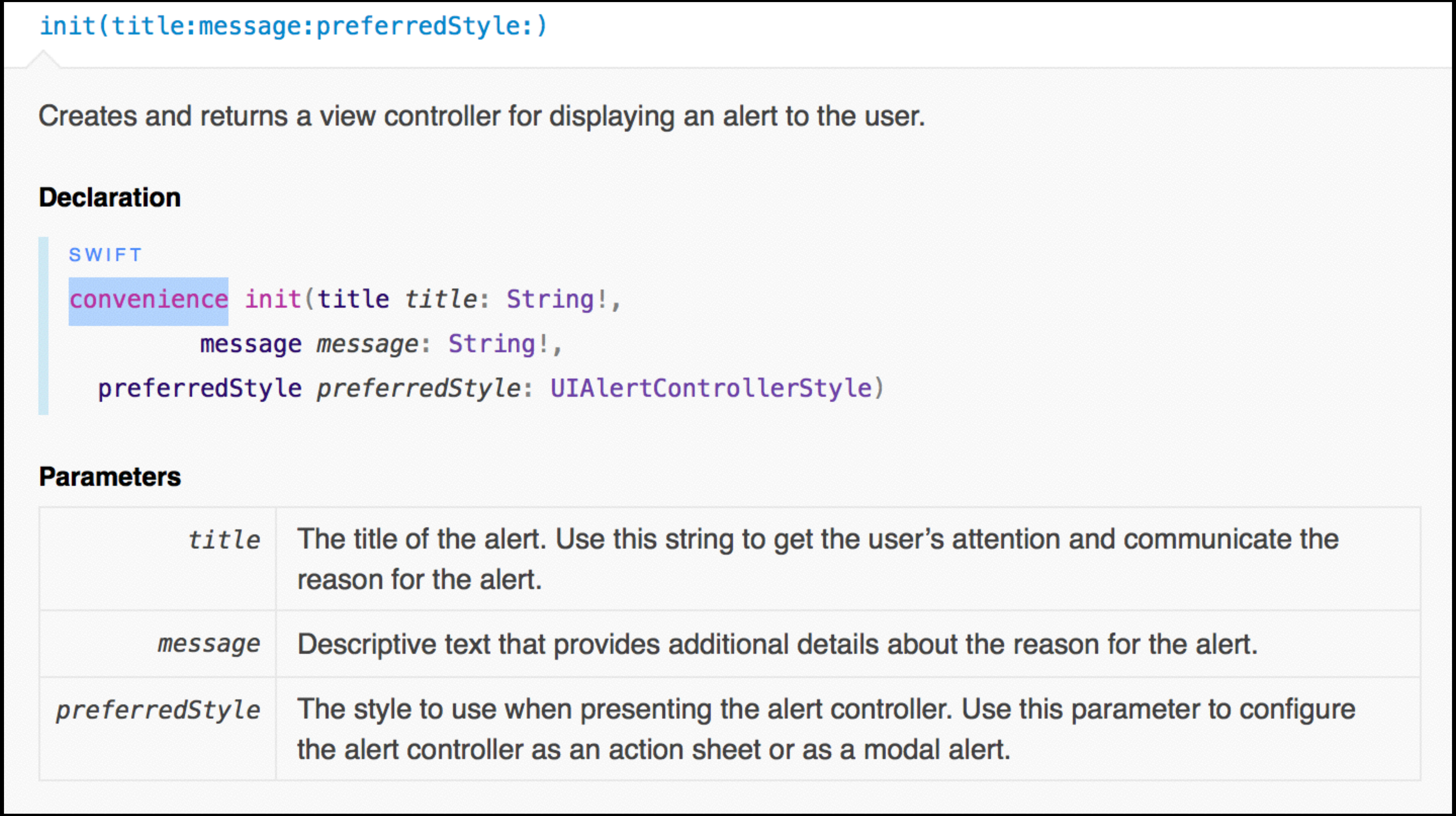


ActionSheet



AlertView

# UIAlertController Setup



`init(title:message:preferredStyle:)`

Creates and returns a view controller for displaying an alert to the user.

**Declaration**

SWIFT

```
convenience init(title title: String!,
          message message: String!,
   preferredStyle preferredStyle: UIAlertControllerStyle)
```

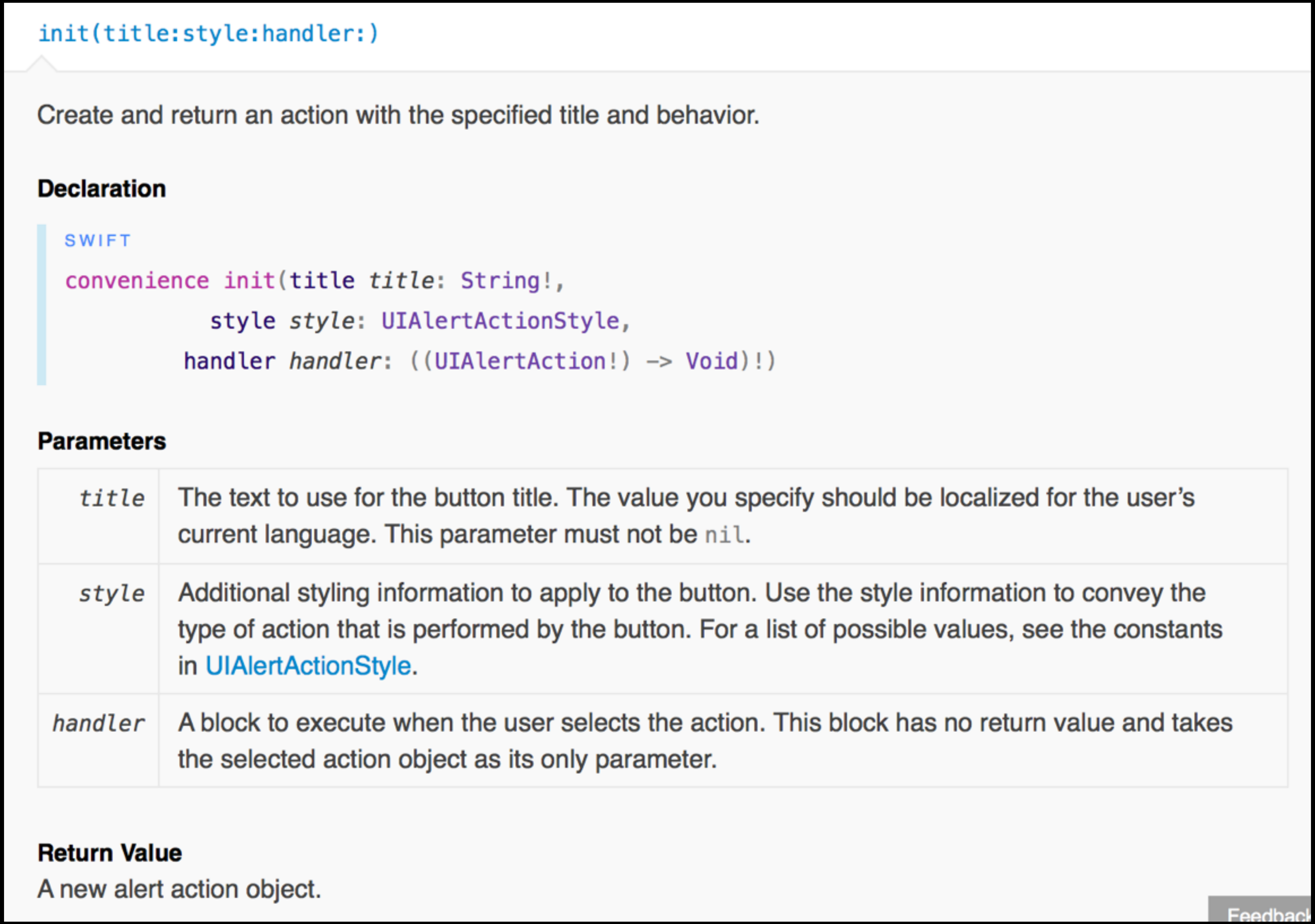
**Parameters**

| | |
|---|---|
| *title* | The title of the alert. Use this string to get the user's attention and communicate the reason for the alert. |
| *message* | Descriptive text that provides additional details about the reason for the alert. |
| *preferredStyle* | The style to use when presenting the alert controller. Use this parameter to configure the alert controller as an action sheet or as a modal alert. |

## UIAlertController Configuration

- In order to add buttons to your alert controller, you need to add actions.
- An action is a instance of the UIAlertAction class.
- “A UIAlertAction object represents an action that can be taken when tapping a button in an alert”
- Uses a closure expression (great!) to define the behavior of when the button is pressed. This is called the handler.

# UIAlertAction Setup



`init(title:style:handler:)`

Create and return an action with the specified title and behavior.

**Declaration**

```
SWIFT
convenience init(title title: String!,
           style style: UIAlertActionStyle,
         handler handler: ((UIAlertAction!) -> Void)!)
```

**Parameters**

| | |
|---|---|
| *title* | The text to use for the button title. The value you specify should be localized for the user's current language. This parameter must not be `nil`. |
| *style* | Additional styling information to apply to the button. Use the style information to convey the type of action that is performed by the button. For a list of possible values, see the constants in UIAlertActionStyle. |
| *handler* | A block to execute when the user selects the action. This block has no return value and takes the selected action object as its only parameter. |

**Return Value**

A new alert action object.

## Adding Actions

- Adding actions to the AlertController is as easy as calling addAction: on your AlertController and passing in the UIAlertAction(s)
- The order in which you add those actions determines their order in the resulting AlertController.

# Demo

# UICollectionView

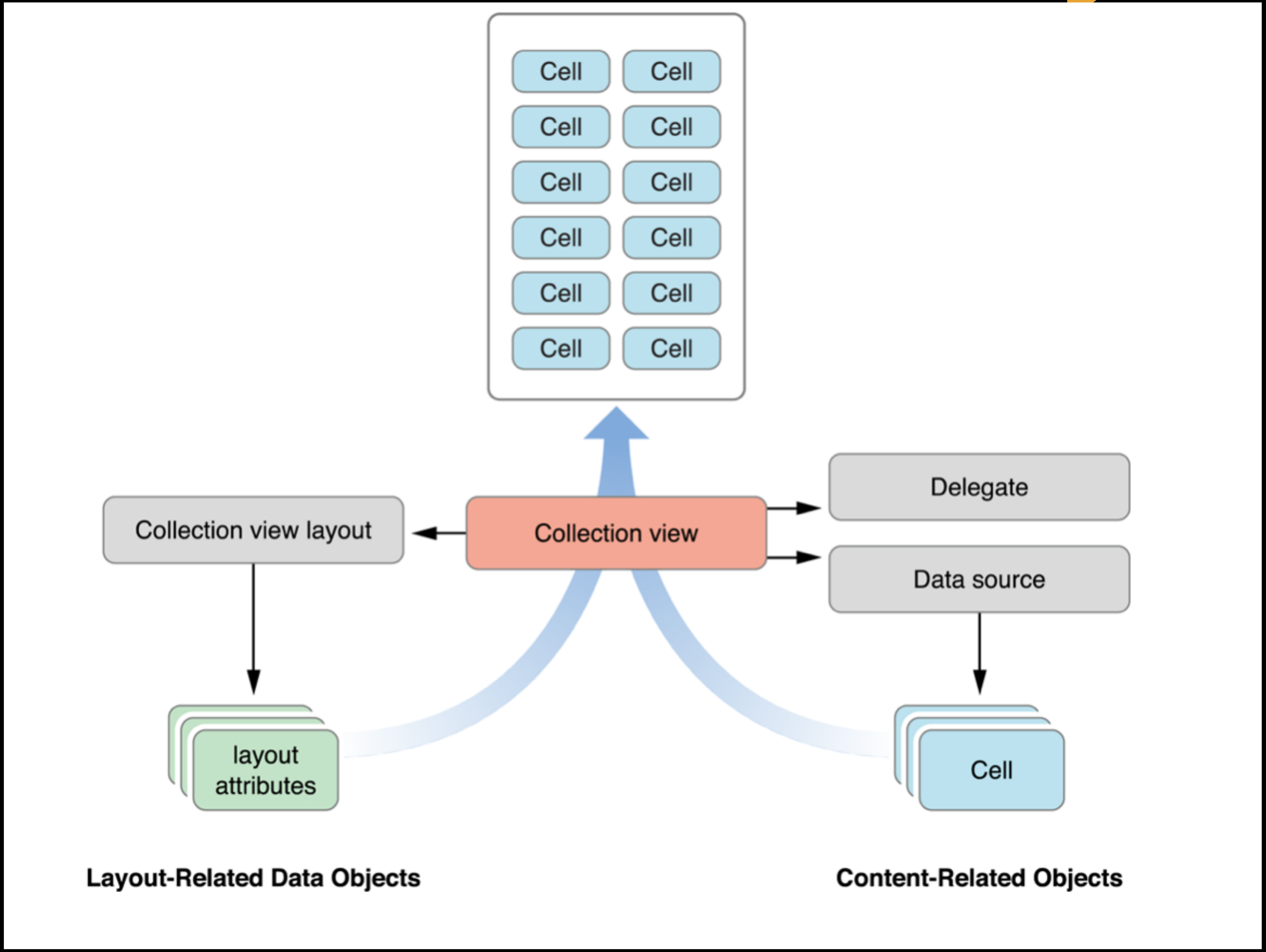
## UICollectionView

- “A collection view is a way to present an ordered set of data items using flexible and changeable layout”
- Most commonly used to present items in a grid-like arrangement. (Items to collection views are as rows to table views)
- Creating custom layouts allow the possibility of many different layouts (grids, circular layouts, stacks, dynamic,etc)

## Internal WorkFlow

- You provide the data (datasource pattern!)
- The layout object provides the placement information
- The collection view merges the two pieces together to achieve the final appearance.

# Collection View Objects



## Reusable Views

- Collection views employ the same recycle program that table views do. (Same Queue data structure)
- 3 reusable views involved with collection views:
    1. Cells : Presents the content of a single item
    2. Supplementary views : headers and footers
    3. Decoration views : wholly owned by the layout object and not tied to any data from your data source. Ex : Custom background appearance.
- Unlike table views, collection view imposes no styles on your reusable views, they are for the most part blank canvases for you to work with.

## `CollectionViewDataSource`

- Very similar to tableview’s datasource. It is required.
- Must answer these questions:
    - For a given section, how many items does a section contain?
    - For a given section or item, what views should be used to display the corresponding content? (just like cellForRow)

Demo

## Asset Catalogs

## Asset Catalogs

- “Use asset catalogs to simplify management of images that are used by your app”
- Things you can put in your asset catalogs:
    - Image sets
    - App icons
    - Launch Images

# Image Sets

- Image sets contain all the versions of an image necessary for the different scale factors of different devices' screens. (ie retina vs non-retina)
- When you drag an image into the XCAsset, an image set will be created for it.
- You will see 1x(non retina), 2x (retina), and 3x (retina HD) slots for each image.
- Image sets can also be configured to be device or size class specific.

## 1x, 2x, and 3x

- Lets say we have an image view that is going to be constrained to 300 points height and 300 points width.
- We would need a 300x300 image for 1x, 600x600 image for 2x, and 900x900 image for 3x for the image displayed in the image view to scale properly for each device resolution.

## 1x, 2x, and 3x

## 1x, 2x, and 3x

## Demo

## Custom Delegate Protocol

## Delegation

- “Delegation is a simple and powerful pattern in which one object in a program acts on behalf of, or in coordination with, another object”
- The delegating keeps a reference to the object is it is delegating to, and that reference, aka property, is usually just called delegate.
- Usually the delegating object is a framework object provided by Apple (like tableview) and the delegate is your own custom controller object.
- Lets look at some examples of delegation in Apple’s APIs

## Tableview and CollectionView

- Both of these objects use delegation to handle user interactions, like the user selecting a cell or item, or scrolling.
- They also use the data source pattern for displaying data to the user, and thats just a special form of delegation that is specifically just for retrieving data.

# Textfields and TextViews

- The textfield delegate is notified when the user begins, during, and ends editing the textfield
- Textview is the same set of methods, additionally it notifies of text selection and user interaction with the text/any urls.

## AppDelegate

- “The app delegate is custom object created at app launch time…”
- The primary job of the app delegate is to handle your app’s transitions to and from the background.
- It also gives you a spot for launch-time initialization.
- Great place to manually set the root view controller.

## Being a delegate

- In order to be a delegate of something, you need to conform to the the delegate protocol.
- Think of this as like signing a contract, specifying something you will do at some point in time.

## Protocol

- “A protocol defines a blueprint of methods, properties, and other requirements that suit a particular task or piece of functionality.
- Its important to note that a protocol **does not define the implementation of any of these,** it simply lists things that something that adopts the protocol should have.
- Once an object satisfies all the requirements, it is said to **conform** to the protocol.

# Protocol Syntax

- Protocols are defined just like you define any other type:

```
protocol SomeProtocol {
    // protocol definition goes here
}
```

- To have your custom type conform to the protocol, add a comma after its super class and then list as many protocols as you need:

```
class SomeClass: SomeSuperclass, FirstProtocol,
        AnotherProtocol {
    // class definition goes here
}
```

# Protocol Requirements

- There are two main things your protocols can require: methods and properties.
- You declare properties just like normally do in any custom type you are making.
- Methods look a little different, as you only define the methods name, parameters, and return type. You don't write the implementation of the method:

```
protocol RandomNumberGenerator {
    func random() -> Double
}
```

## Optional Requirements

- By default, everything you list in your protocol is required.
- To make a property or method optional, you simply use the optional keyword before the definition of the property or method:

```
@objc protocol CounterDataSource {
    optional func incrementForCount(count: Int) ->
        Int
    optional var fixedIncrement: Int { get }
}
```

You also need the @objc tag, so only classes can adopt protocols with optional requirements

## Protocol as a type

- You can work with protocols like any other type in your code.
- This comes in handy when you are defining your delegate properties:

```
var delegate: DiceGameDelegate?
```

Here we have a property called delegate, and its type is the DiceGameDelegate protocol. So this property can be type, as long as it conforms to the protocol.

## Best practice delegate method convention

- Delegation methods should begin with the name of object doing the delegating — application, control, controller, etc.
- The name is then followed by a verb of what just occurred — willSelect, didSelect, openFile, etc.

Demo

## Camera Programming

- 2 ways for interfacing with the camera in your app:
  1. UIImagePickerController (easy mode)
  2. AVFoundation Framework (hard mode)

## UIImagePickerController

- The workflow of using UIImagePickerController is 3 steps:
  1. Instantiate and modally present the UIImagePickerController
  2. ImagePicker manages the user’s interaction with the camera or photo library
  3. The system invokes your image picker controller delegate methods to handle the user being done with the picker.

## UIImagePickerController Setup

- The first thing you have to account for is checking if the device has a camera.
- If your app absolutely relies on a camera, add a UIRequiredDeviceCapabilities key in your info.plist
- Use the isSourceTypeAvailable class method on UIImagePickerController to check if camera is available.

## UIImagePickerController Setup

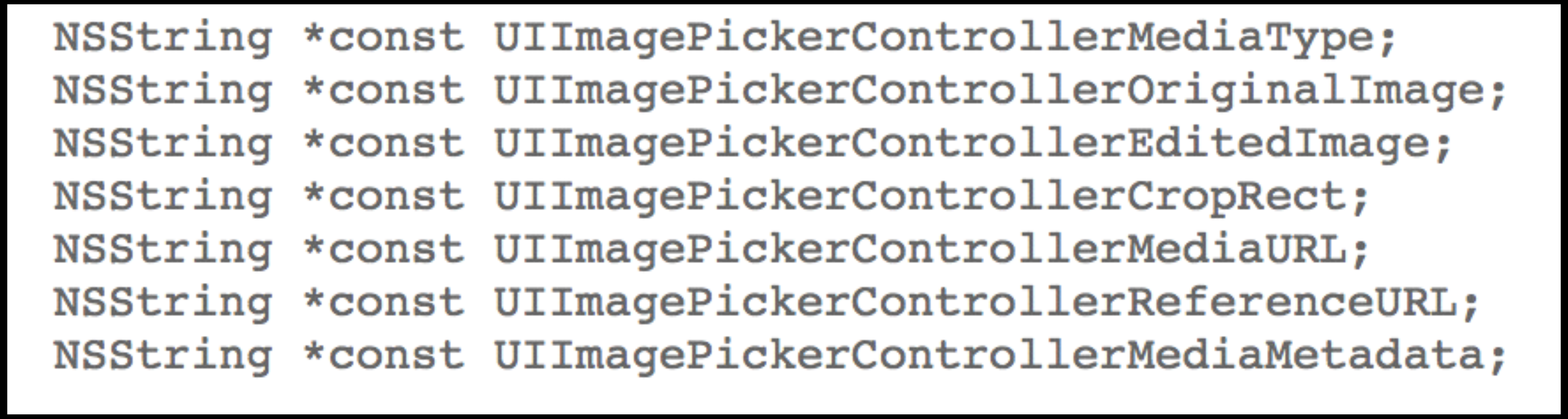
- Next make sure something is setup to be the delegate of the picker. This is usually the view controller that is spawning the picker.
- The final step is to actually create the UIImagePicker with a sourceType of UIImagePickerControllerSourceTypeCamera.
- Media Types: Used to specify if the camera should be locked to photos, videos, or both.
- AllowsEditing property to set if the user is able to modify the photo in the picker after taking the photo.

## `UIImagePickerControllerDelegate`

- The Delegate methods control what happens after the user is done using the picker. 2 big method:
  1. imagePickerControllerDidCancel:
  2. imagePickerController:didFinishPickingMediaWithInfo:

## Info Dictionary

The info dictionary has a number of items related to the image that was taken:



```
NSString *const UIImagePickerControllerMediaType;
NSString *const UIImagePickerControllerOriginalImage;
NSString *const UIImagePickerControllerEditedImage;
NSString *const UIImagePickerControllerCropRect;
NSString *const UIImagePickerControllerMediaURL;
NSString *const UIImagePickerControllerReferenceURL;
NSString *const UIImagePickerControllerMediaMetadata;
```

MediaType is either kUTTypeImage or kUTTypeMovie

Demo

Mobile Monday
Intro to Android

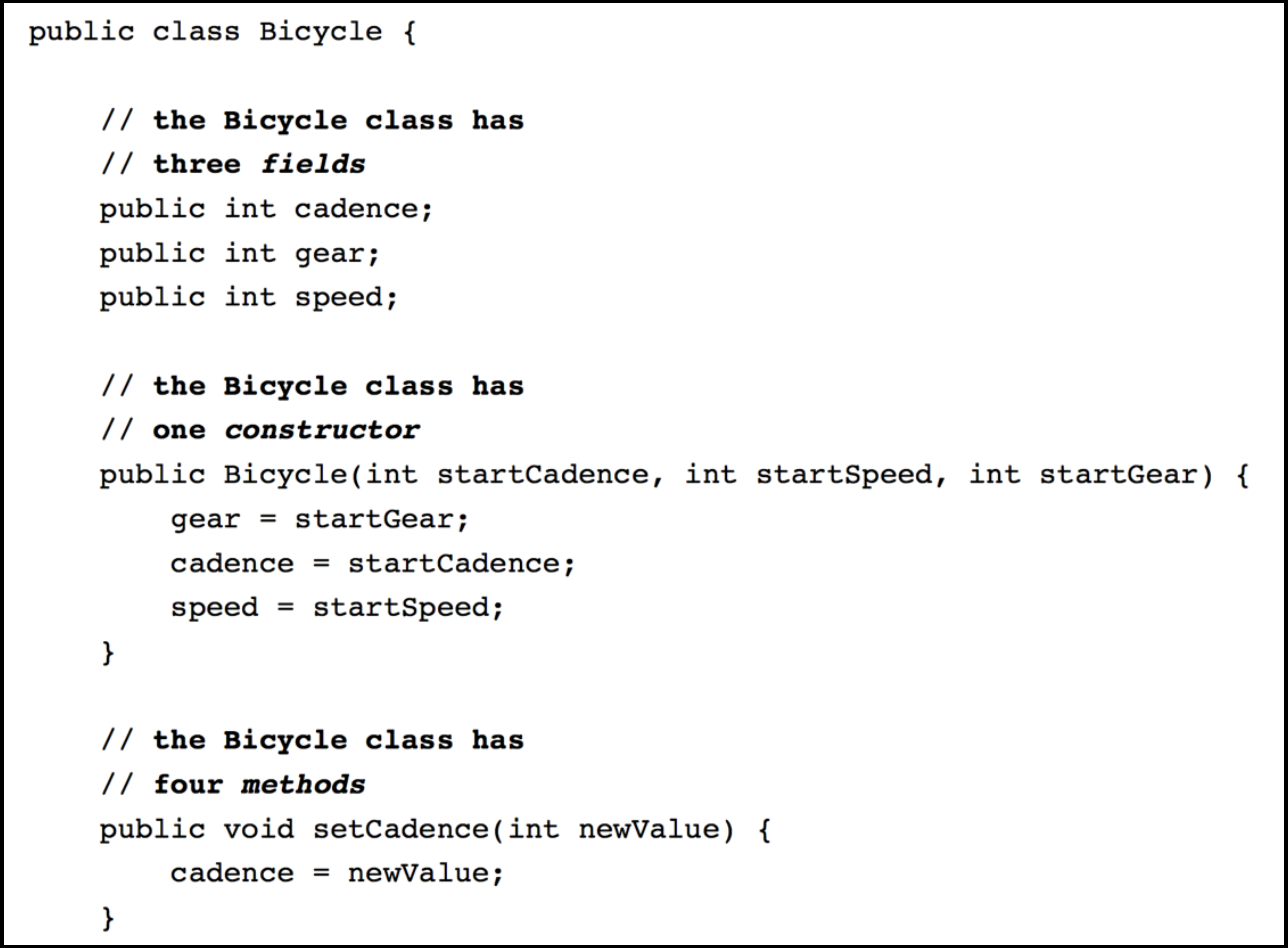
## Android Development

- Android is programmed with Java
- Two primary IDE's for android:
  - Eclipse (older)
  - Android Studio (new replacement of Eclipse; made by Google)
- Both IDE's provide a emulator rather than a simulator, and the emulator takes forever to build so its recommended you build on a device

## Java

- Java is object oriented just like Objective-C and Swift
- Java has garbage collection similar to ARC
- Java is statically typed, like Swift
- Considered platform independent. Java code is compiled and converted to bytecode, and then executed by the Java Virtual Machine (JVM). So any platform with a JVM can run Java code.

# Java Class

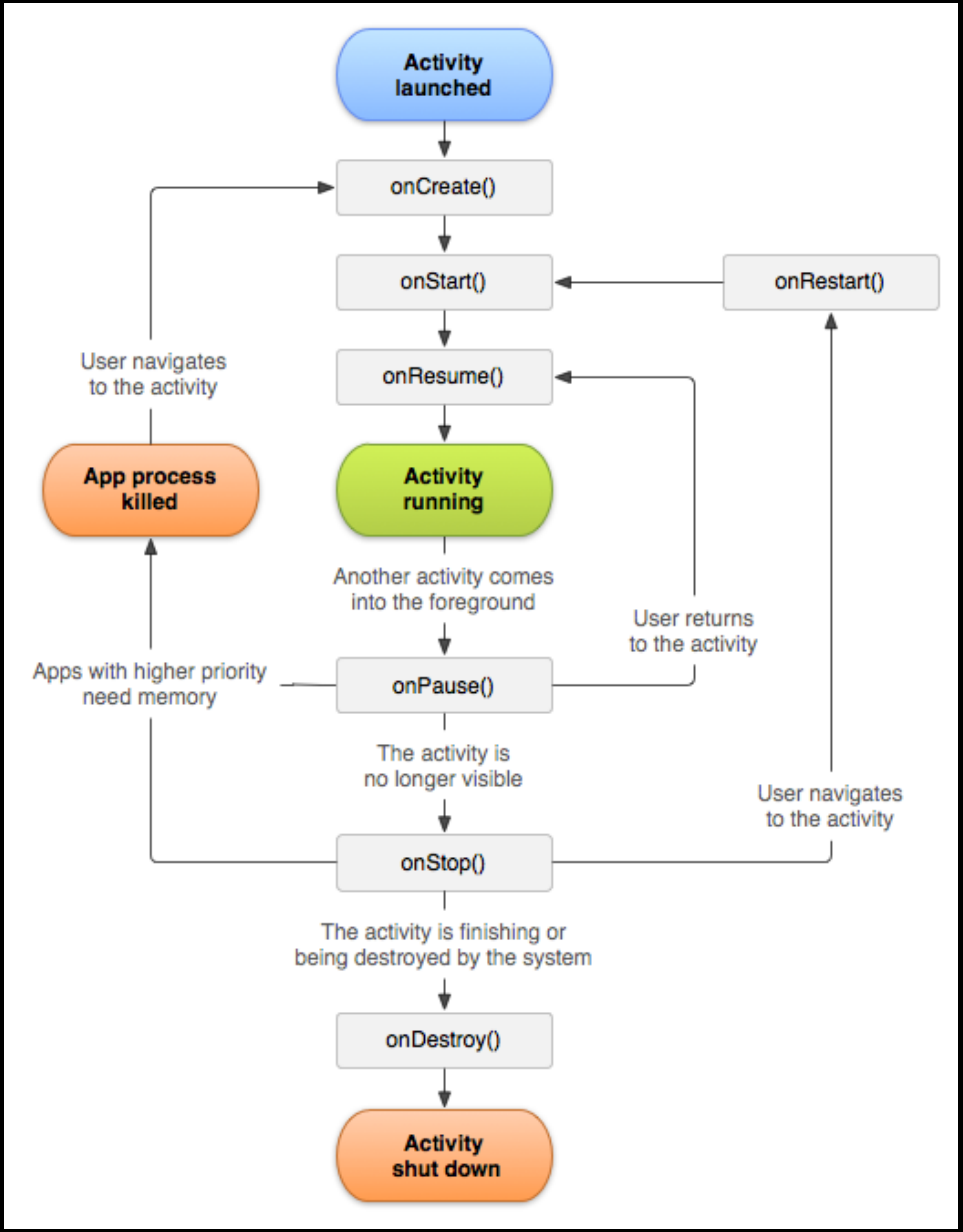


```
public class Bicycle {

    // the Bicycle class has
    // three fields
    public int cadence;
    public int gear;
    public int speed;

    // the Bicycle class has
    // one constructor
    public Bicycle(int startCadence, int startSpeed, int startGear) {
        gear = startGear;
        cadence = startCadence;
        speed = startSpeed;
    }

    // the Bicycle class has
    // four methods
    public void setCadence(int newValue) {
        cadence = newValue;
    }
```

## Android Activities

- Activities are Android's version of View Controllers.
- Instead of Navigation Controllers, Android keeps a global navigation stack for all apps to use.
- When an app is launched, the system pushes the app's first activity onto the stack.
- Pressing the back button on an Android device pops the top activity off from the global stack.

## Activity LifeCycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

*Rotation kills your activity!

## Android Intents

- An intent is a messaging object that you can use to request an action from another app component.
- Intents have 3 major uses:
  - To start an activity - Similar to passing an object in prepareForSegue:
  - To start a service - A service is a component that performs operations in the background without a user interface.
  - To deliver a broadcast - A broadcast is a message that any app can receive. So your app can directly pass messages to other apps. Seems legit.

# Android ListView

- List View is Android's version of a table view.
- Similar to how UIKit provides a UITableViewController, Android provides a pre-built ListActivity.
- Instead of datasource/delegate, List View's have adapters.
- Adapters take data and adapt it to populate a list view.

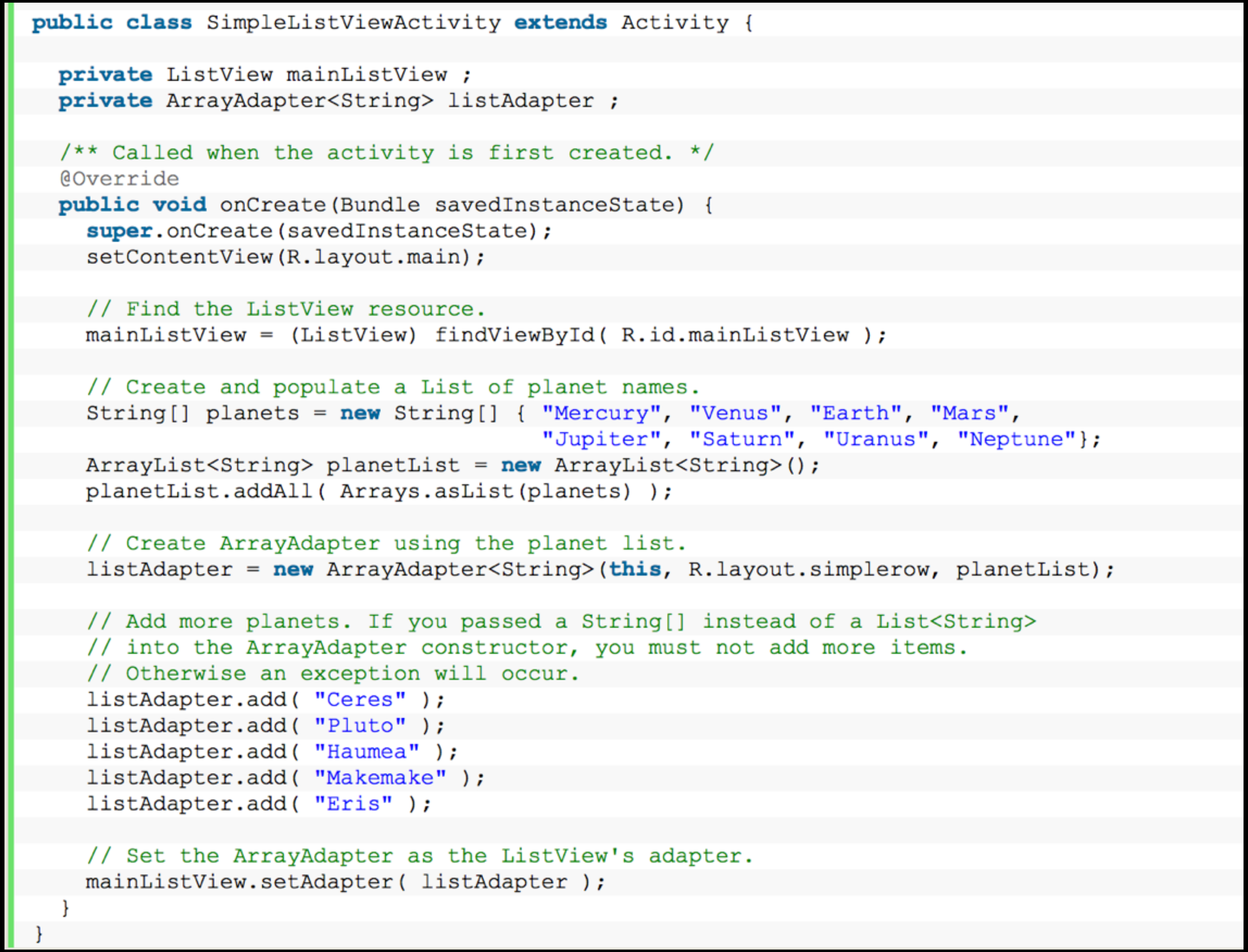
## Array Adapter

- The ArrayAdapter class can take in a list or array of objects as input.
- The ArrayAdapter will then map a row in the list view for each object in the array.
- The ArrayAdapter will call toString() on each object and will take the return value and plug it in to each row.

## Custom Adapter

- Creating a custom Adapter is like having one of your custom objects be the data source of a table view.
- You can implement the method getView() and return a view for each row. This is the equivalent for cellForRowAtIndexPath:

# Activity with a ListView



```java
public class SimpleListViewActivity extends Activity {

  private ListView mainListView ;
  private ArrayAdapter<String> listAdapter ;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Find the ListView resource.
    mainListView = (ListView) findViewById( R.id.mainListView );

    // Create and populate a List of planet names.
    String[] planets = new String[] { "Mercury", "Venus", "Earth", "Mars",
                                      "Jupiter", "Saturn", "Uranus", "Neptune"};
    ArrayList<String> planetList = new ArrayList<String>();
    planetList.addAll( Arrays.asList(planets) );

    // Create ArrayAdapter using the planet list.
    listAdapter = new ArrayAdapter<String>(this, R.layout.simplerow, planetList);

    // Add more planets. If you passed a String[] instead of a List<String>
    // into the ArrayAdapter constructor, you must not add more items.
    // Otherwise an exception will occur.
    listAdapter.add( "Ceres" );
    listAdapter.add( "Pluto" );
    listAdapter.add( "Haumea" );
    listAdapter.add( "Makemake" );
    listAdapter.add( "Eris" );

    // Set the ArrayAdapter as the ListView's adapter.
    mainListView.setAdapter( listAdapter );
  }
}
```