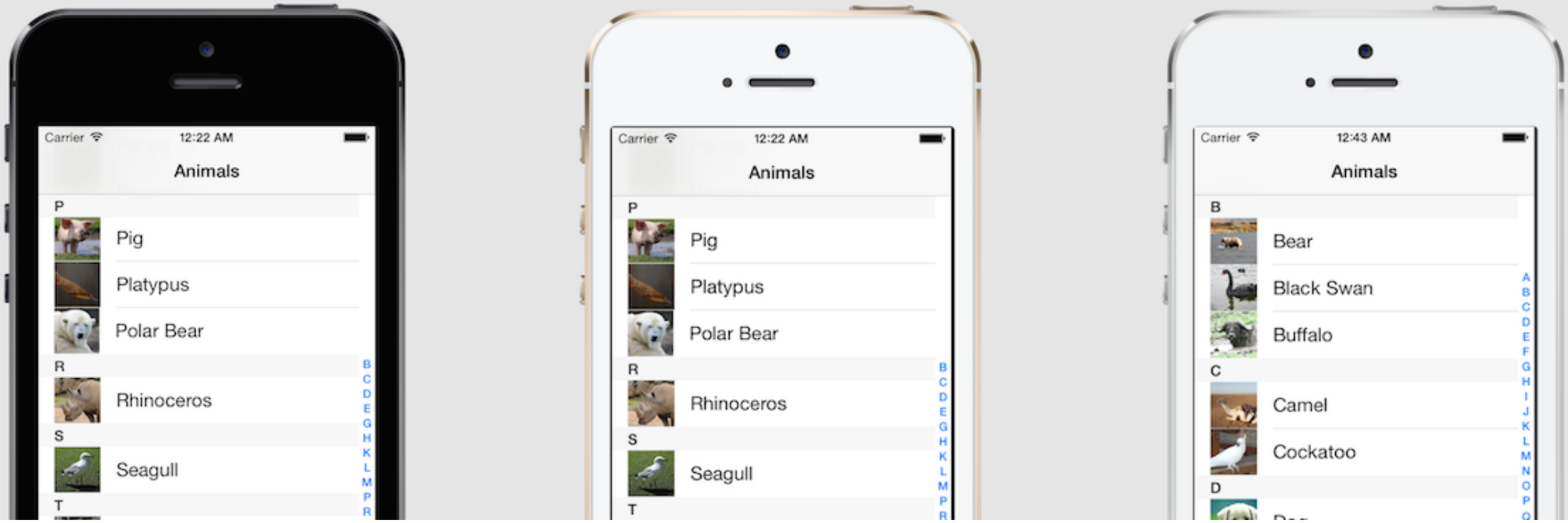
CHAPTER 2

ADDING SECTIONS AND INDEX LIST IN UITABLEVIEW



Grouping data into sections and adding an index for quick access

TABLE SECTION AND INDEX LIST

If you'd like to show a large number of records in UITableView, you'd best rethink the approach of how to display your data. As the number of rows grows, the table view becomes unwieldy. One way to improve the user experience is to organize the data into sections. By grouping related data together, you offer a better way for users to access it.

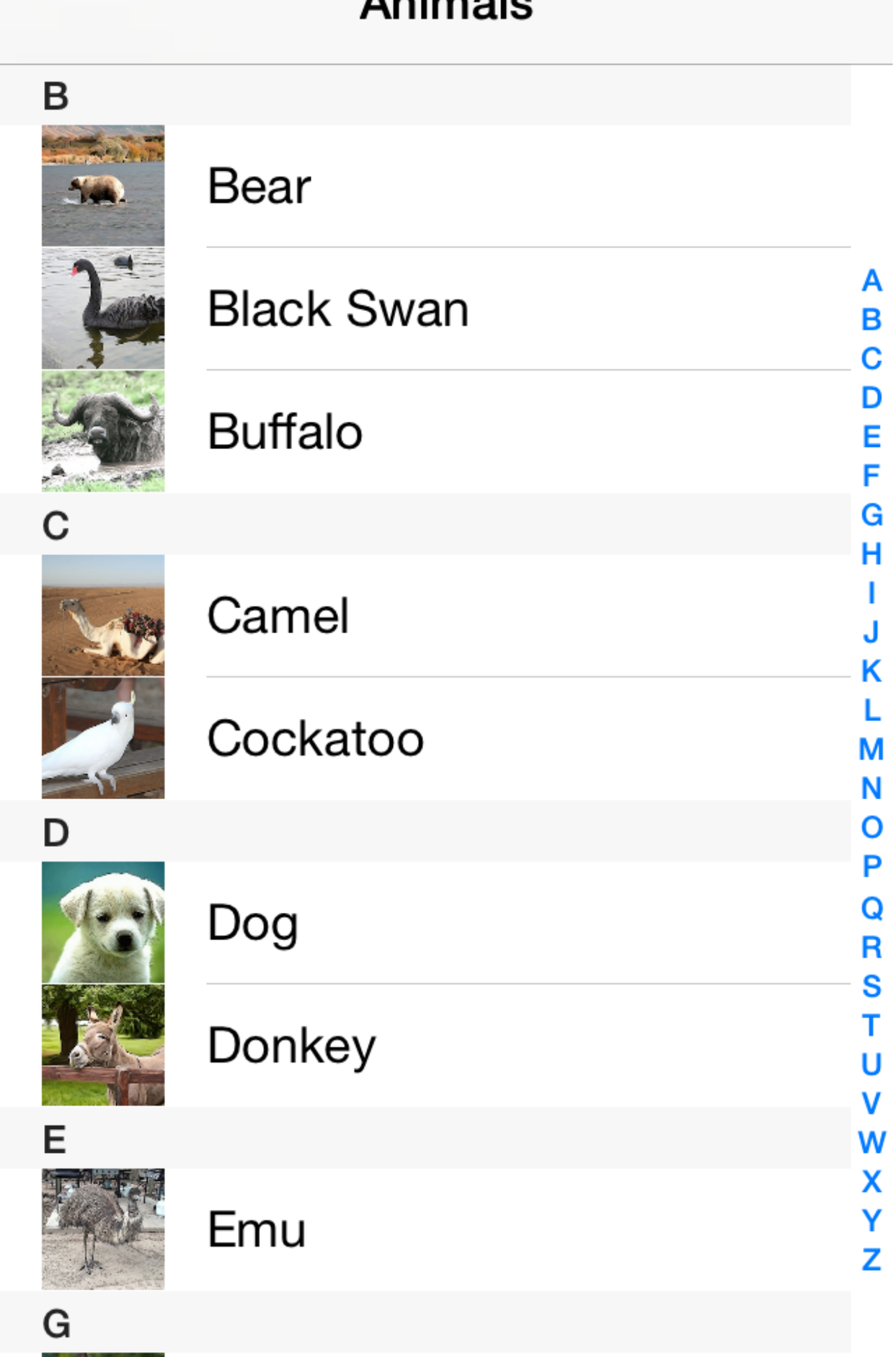
Furthermore, you can implement an index list in the table view. An indexed table view is more or less the same as the plain-styled table view that we covered at the very opening of the book, *Beginning iOS 8 Programming with Swift*. The only difference is that it includes an index on the right side of the table view. An indexed table is very common in iOS apps. The most well-known example is the built-in Contacts app on the iPhone. By offering index scrolling, users have the ability to access a particular section of the table instantly without scrolling through each section.

Let's see how we can add sections and an index list to a simple table app.

If you have a basic understanding of the UITableView implementation, it's not too difficult to add sections and an index list. Basically, to add sections and an index list in the UITableView, you need to deal with these methods as defined in UITableViewDataSource protocol:

- *numberOfSectionsInTableView* method – returns the total number of sections in the table view. Usually we set the number of sections to 1. If you'd prefer to have multiple sections, set this value to other numbers.
- *titleForHeaderInSection* method – returns the header titles for different sections. This method is optional if you do not prefer to assign titles to the section.
- *numberOfRowsInSection* method – returns the total number of rows in a specific section.
- *cellForRowAtIndexPath* method – this method shouldn't be new to you if you know how to display data in UITableView. It returns the table data for a particular section.
- *sectionIndexTitlesForTableView* method – returns the indexed titles that appear in the index list on the right side of the table view. For example, you can return an array of strings containing a value from "A" to "Z".
- *sectionForSectionIndexTitle* method – returns the section index that the table view should jump to when the user taps a particular index.

There's no better way to explain the implementation than by showing an example. As usual, we'll build a simple app, which should give you a better idea of an index list after implementing the app.



A BRIEF LOOK AT THE DEMO APP

First, let's have a quick look at the demo app that we are going to build. It's a very simple app showing a list of animals in a standard table view. Instead of listing all the animals, the app groups the animals into different sections, and displays an index list for quick access. The screenshot on your left displays the final deliverable of the demo app.

DOWNLOAD THE XCODE PROJECT TEMPLATE

The focus of this demo is on the implementation of sections and index list. Therefore, instead of building the Xcode project from scratch, you can download the project template from https://www.dropbox.com/s/2whtwzodzw2xrq3/IndexedTableDemoTemplate.zip?dl=0.

The template already includes everything you need to start with. If you build the template, you'll have an app showing a list of animals in a table view (but without sections and index). Later, we will modify the app, group the data into sections, and add an index list to the table.

DISPLAYING SECTIONS IN UITABLEVIEW

Okay, let's get started. If you open the IndexTableDemo project, the animal data is defined in an array:

```
let animals = ["Bear", "Black Swan", "Buffalo", "Camel", "Cockatoo",
"Dog", "Donkey", "Emu", "Giraffe", "Greater Rhea", "Hippopotamus",
"Horse", "Koala", "Lion", "Llama", "Manatus", "Meerkat", "Panda",
"Peacock", "Pig", "Platypus", "Polar Bear", "Rhinoceros", "Seagull",
"Tasmania Devil", "Whale", "Whale Shark", "Wombat"]
```

Well, we're going to organize the data into sections based on the first letter of the animal name. There are a lot of ways to do that. One way is to manually replace the animals array with a dictionary like I've shown below:

```
let animals: [String: [String]] = ["B" : ["Bear", "Black Swan", "Buffalo"],
    "C" : ["Camel", "Cockatoo"],
    "D" : ["Dog", "Donkey"],
    "E" : ["Emu"],
    "G" : ["Giraffe", "Greater Rhea"],
    "H" : ["Hippopotamus", "Horse"],
    "K" : ["Koala"],
    "L" : ["Lion", "Llama"],
    "M" : ["Manatus", "Meerkat"],
    "P" : ["Panda", "Peacock", "Pig", "Platypus", "Polar Bear"],
    "R" : ["Rhinoceros"],
    "S" : ["Seagull"],
    "T" : ["Tasmania Devil"],
    "W" : ["Whale", "Whale Shark", "Wombat"]]
```

In the above code, we've turned the animals array into a dictionary. The first letter of the animal name is used as a key. The value that is associated with the corresponding key is an array of animal names.

We could manually create the dictionary, but wouldn't it be great if we could create the indexes from the animals array? Let's see how it can be done.

First, declare two instance variables in the AnimalTableViewController class:

```
var animalsDict = [String: [String]]()
var animalSectionTitles = [String]()
```

We initialize an empty dictionary for storing the animals and an empty array for storing the section titles of the table. The section title is the first letter of the animal name (e.g. B).

Because we want to generate a dictionary from the animals array, we need a helper method to handle the generation. Insert the following method in the AnimalTableViewController class:

```
func createAnimalDict() {
    for animal in animals {
        // Get the first letter of the animal name and build the dictionary
        let animalKey = animal.substringToIndex(advance(animal.startIndex,
1))
        if var animalValues = animalsDict[animalKey] {
            animalValues.append(animal)
            animalsDict[animalKey] = animalValues
        } else {
            animalsDict[animalKey] = [animal]
        }
    }
```

```
    // Get the section titles from the dictionary's keys and sort them in
ascending order
    animalSectionTitles = [String](animalsDict.keys)
    animalSectionTitles.sort({ $0 < $1 })
}
```

In this method, we loop through all the items in the *animals* array. For each item, we initially extract the first letter of the animal's name. In Swift, the *subStringToIndex* method of a string can return a new string containing the characters up to a given index. The index should be in **String.Index** type. To construct an index for a specific position, you have to first ask the string for its startIndex and then use the global *advance()* function to iterate over all characters between the beginning of the string and the target position. In this case, the target position is 1.

As mentioned before, the first letter of the animal's name is used as a key of the dictionary. The value of the dictionary is an array of animals of that particular key. So once we've got the key, we either create a new array of animals or append the item to an existing array. Here we show the values of animalsDict for the first four iterations:

Iteration #1: animalsDict["B"] = ["Bear"]

Iteration #2: animalsDict["B"] = ["Bear", "Black Swan"]

Iteration #3: animalsDict["B"] = ["Bear", "Black Swan", "Buffalo"]

Iteration #4: animalsDict["C"] = ["Camel"]

After the *animalsDict* is completely generated, we can retrieve the section titles from the keys of the dictionary.

To retrieve the keys of a dictionary, you can simply call the *keys* method. However, the keys returned are unordered. Swift's standard library provides a function called **sort**, which sorts an array of values of a known type, based on the output of a sorting closure that you provide.

The closure takes two arguments of the same type (this is the string) and returns a Bool value to state whether the first value should appear before or after the second value once the values are sorted. If the first value should appear before the second value, it should return true.

One way to write the sort closure is like this:

```
animalSectionTitles.sort( { (s1:String, s2:String) -> Bool in
        return s1 < s2
    })
```

You should be very familiar with the closure expression syntax. In the body of the closure, we compare the two string values. It returns true if the second value is greater than the first value. For instance, the value of s1 is "B" and that of s2 is "E." Because B is smaller than E, the closure returns true, indicating that B should appear before E. In this case, we can sort the values in alphabetical order.

If you read the earlier code snippet carefully, you may wonder why I wrote the sort closure like this:

```
animalSectionTitles.sort({ $0 < $1 })
```

It's a shorthand in Swift for writing inline closures. Here $0 and $1 refer to the first and second String arguments. If you use shorthand argument names, you can omit nearly everything of the closure

including argument list and **in** keyword; you will just need to write the body of the closure.

With the helper method created, update the *viewDidLoad* method to call it up:

```
override func viewDidLoad() {
    super.viewDidLoad()

    // Generate the animal dictionary
    createAnimalDict()
}
```

Next, change the *numberOfSectionsInTableView* method and return the total number of sections:

```
override func numberOfSectionsInTableView(tableView: UITableView) ->
Int {
    // Return the number of sections.
    return animalSectionTitles.count
}
```

To display a header title in each section, we need to implement the *titleForHeaderInSection* method. This method is called every time a new section is displayed. Based on the given section index, we simply return the corresponding section title.

```
override func tableView(tableView: UITableView, titleForHeaderInSection
section: Int) -> String? {
    return animalSectionTitles[section]
}
```

It's very straightforward, right? Next, we have to tell the table view the number of rows in a particular section. Insert the *numberOfRowsInSection* method in the AnimalTableViewController.swift:

```
override func tableView(tableView: UITableView, numberOfRowsInSection
section: Int) -> Int {
    // Return the number of rows in the section.
    let animalKey = animalSectionTitles[section]
    if let animalValues = animalsDict[animalKey] {
        return animalValues.count
    }

    return 0
}
```

When the app starts to render the data in the table view, the *numberOfRowsInSection* method is called every time a new section is displayed. Based on the section index, we get the section title and use it as a key to retrieve the animal names of that section, followed by returning the total number of animal names for that section.

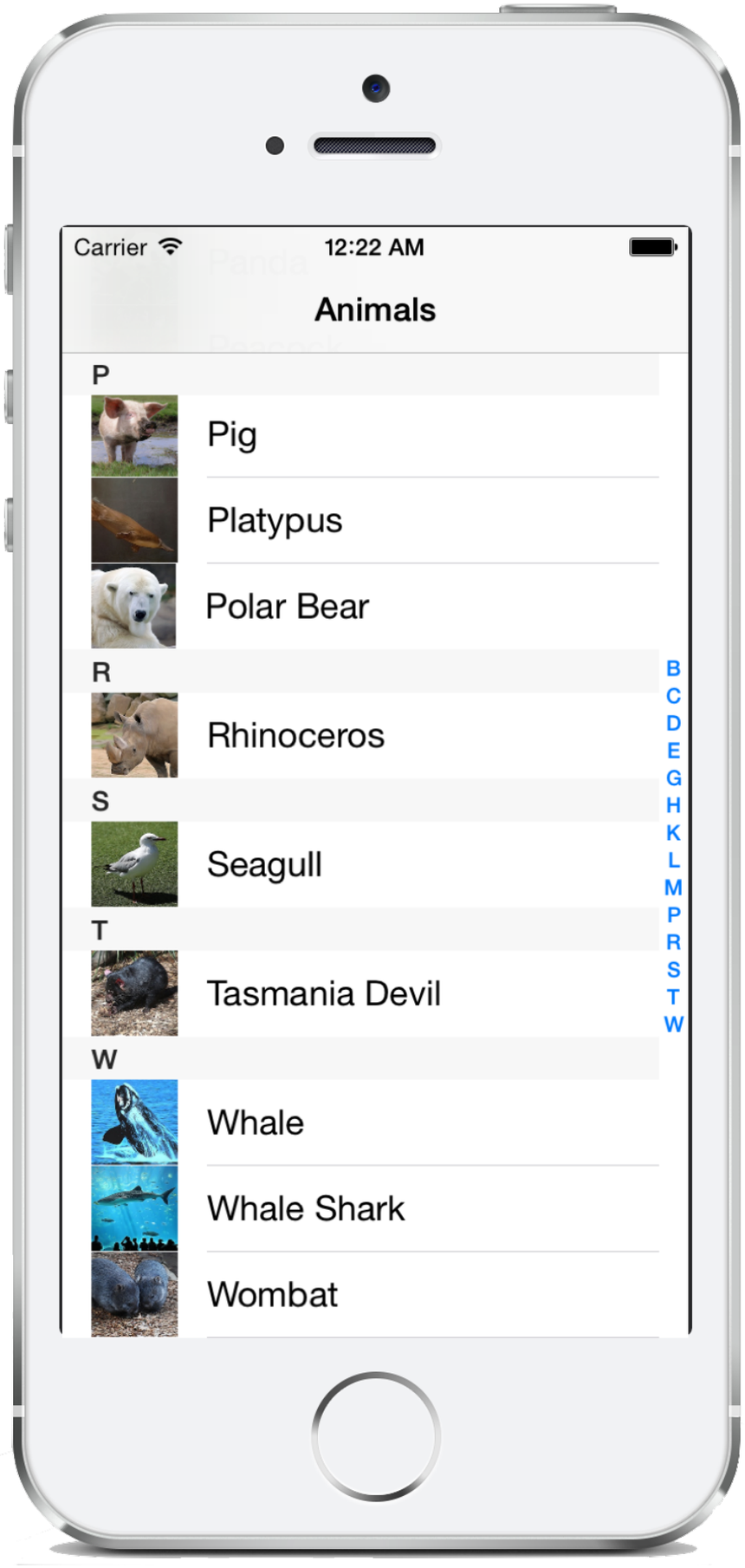
Lastly, modify the *cellForRowAtIndexPath* method as follows:

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath
indexPath: NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("Cell",
forIndexPath: indexPath) as! UITableViewCell

    // Configure the cell...
    let animalKey = animalSectionTitles[indexPath.section]
    if let animalValues = animalsDict[animalKey] {
        cell.textLabel?.text = animalValues[indexPath.row]

        // Convert the animal name to lower case and
```

```
        // then replace all occurrences of a space with an underscore
        let imageFilename = animalValues[indexPath.row].lowercaseString.stringByReplacingOccurrencesOfString(" ", withString: "_", options: nil, range:
nil)
        cell.imageView?.image = UIImage(named: imageFilename)
    }

    return cell
}
```

The indexPath argument contains the current row number, as well as, the current section index. So, based on the section index, we retrieve the section title (e.g. “B”) and use it as the key to retrieve the animal names for that section. The rest of the code is very straightforward. We simply get the animal name and set it as the cell label. The *imageFilename* is computed by converting the animal name to lowercase letters, followed by replacing all occurrences of a space with an underscore.

Okay, you’re ready to go! Hit the Run button and you should end up with an app with sections but without the index list.



ADDING AN INDEX LIST TO UITABLEVIEW

Cool, right? But how can you add an index list to the table view? Again it's easier than you thought and can be achieved with just a few lines of code. Simply add the *sectionIndexTitlesForTableView* method and return an array of section indexes. Here we will use the section titles as the indexes.

```
override func sectionIndexTitlesForTableView(tableView: UITableView) -> [AnyObject]! {
    return animalSectionTitles
}
```

That's it! Compile and run the app again. You should find the index on the right side of the table.

Interestingly, you do not need any implementation and the indexing already works! Try to tap any of the indexes and you'll be brought to a particular section of the table.

ADDING AN A-Z INDEX LIST

Looks like we've done everything. So why did we mention the *sectionForSectionIndexTitle* method at the very beginning?

Currently, the index list doesn't contain the entire alphabet. It just shows those letters that are defined as the keys of the animals dictionary. Sometimes, you may want to display A-Z in the index list. Let's declare a new variable named *animalIndexTitles* in the AnimalTableViewController.swift:

```
let animalIndexTitles = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]
```

Next, change the *sectionIndexTitlesForTableView* method and return the *animalIndexTitles* array instead of the *animalSectionTitles* array.

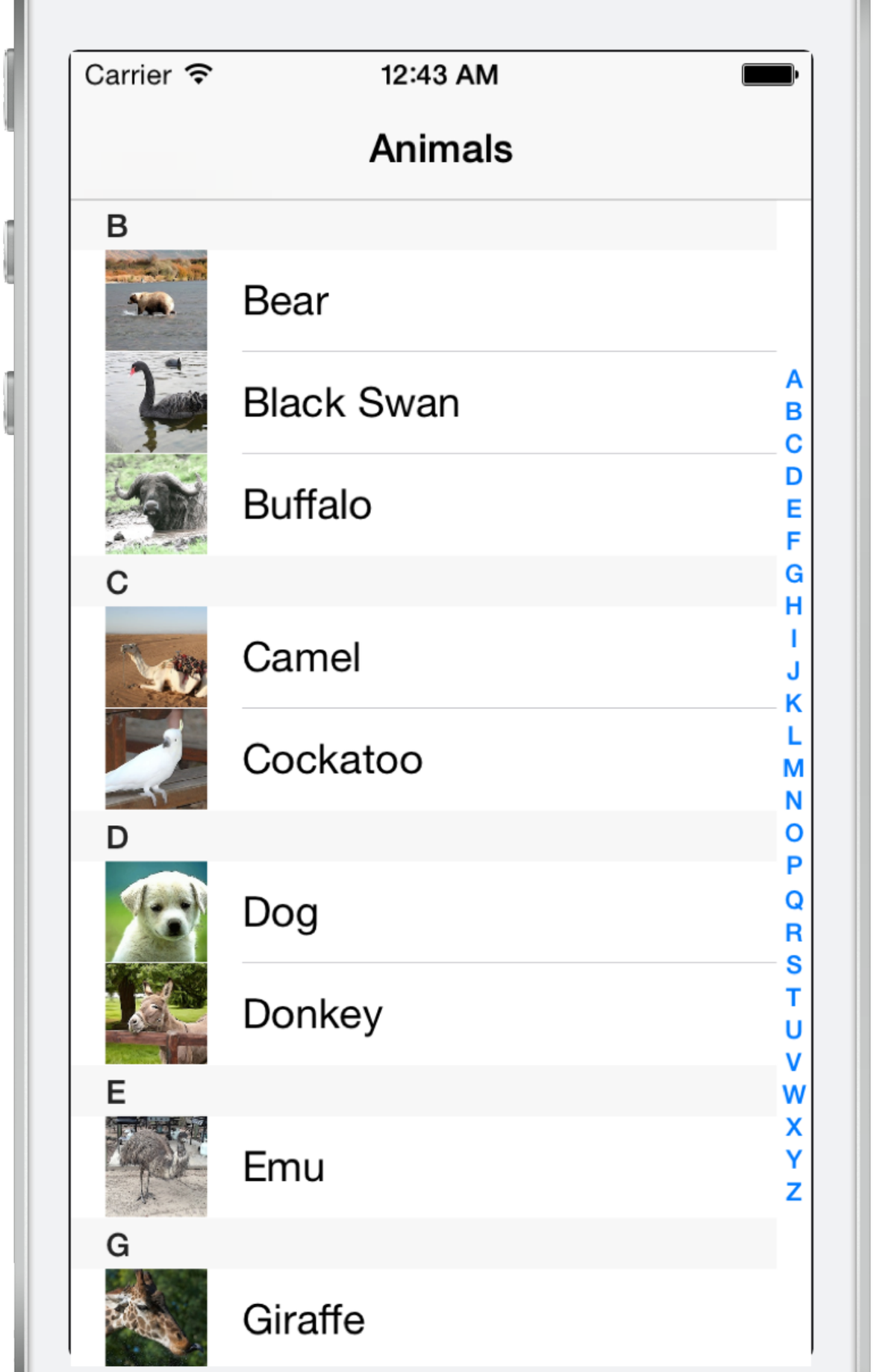
```
override func sectionIndexTitlesForTableView(tableView: UITableView) -> [AnyObject]! {
    return animalIndexTitles
}
```

Now, compile and run the app again. Cool! The app displays the index from A to Z.

But wait a minute… It doesn't work properly! If you try tapping the index "C," the app jumps to the "D" section. And if you tap the index "G," it directs you to the "K" section. Below shows the mapping between the old and new indexes.

Well, as you may notice, the number of indexes is greater than the number of sections, and the UITableView object doesn't know how to handle the indexing. It's your responsibility to implement the *sectionForSectionIndexTitle* method and explicitly tell the table view the section number when a particular index is tapped. Add the following new method:

```
override func tableView(tableView: UITableView, sectionForSectionIndexTitle title: String, atIndex index: Int) -> Int {
```




```
    if let index = find(animalSectionTitles, title) {
        return index
    }

    return -1
}
```

Based on the selected index name (i.e. title), we locate the correct section index from the animalSectionTitles. In Swift, you can use a global function called **find** to find the index of a particular item in the array.

Compile and run the app again. The index list should now work!

CUSTOMIZING SECTION HEADERS

You can easily customize the section headers by overriding some of the methods defined in the UITableView class and the UITableViewDelegate protocol. In this demo, we'll make two simple changes:

- Alter the height of the section header

- Change the font of the section header

To alter the height of the section header, you can simply override the *heightForHeaderInSection* method and return the preferred height:

```
override func tableView(tableView: UITableView, heightForHeaderInSection section: Int) -> CGFloat {
    return 50
}
```

Before the section header view is displayed, the *willDisplayHeaderView* method will be called. The method includes an argument named **view**. This view object can be a custom header view or a standard one. In our demo, we just use the standard header view, which is the UITableViewHeaderFooterView object. Once you have the header view, you can alter the text color and font accordingly.

```
override func tableView(tableView: UITableView, willDisplayHeaderView view: UIView, forSection section: Int) {
    let headerView = view as! UITableViewHeaderFooterView
    headerView.textLabel.textColor = UIColor.orangeColor()
    headerView.textLabel.font = UIFont(name: "Avenir", size: 25.0)
}
```

Run the app again. The header view should be updated with your preferred font and color.

SUMMARY

When you need to display a large number of records, it is simple and effective to organize the data into sections and provide an index list for easy access. In this chapter, we've walked you through the implementation of an indexed table. By now, I believe you should know how to add sections and an index list to your table view.

For your reference, you can download the complete Xcode project from https://www.dropbox.com/s/kpd0q1m5ccsaup7/IndexedTableDemo.zip?dl=0.