Table Views in iOS

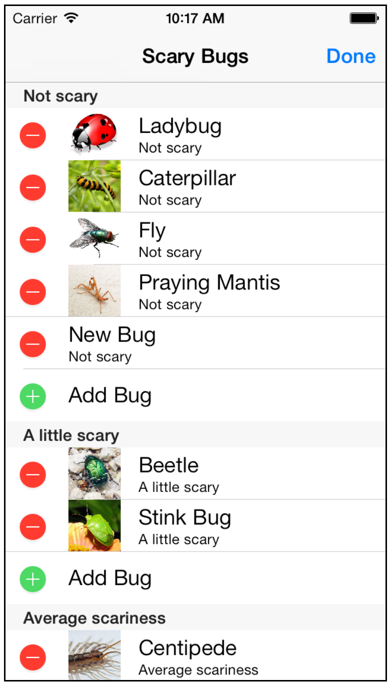
Hands-On Challenges

Table Views: Beginning to Advanced Hands-On Challenges

Challenge #5: Inserting Rows

Although the app so far has an impressive list of bugs, what if you find a new creepy crawly on your floor that you want to document?

In this challenge, you'll solve this conundrum by adding a new row at the end of each section that you can tap to create a new bug:



See if you can do this on your own based on what you learned on the video. If you get stuck, follow along with the full walkthrough below!

Full Walkthrough

Open the Scary Bugs project where you left it off in the last challenge, or use the starter project provided by the instructor.

First, you need to configure your table view to show a new cell to add a new row when the table view is in editing mode.

To do this, open **BugTableViewController.swift** and replace `tableView(_:numberOfRowsInSection:)` with the following:

```
override func tableView(tableView: UITableView,
  numberOfRowsInSection section: Int) -> Int {
    let adjustment = editing ? 1 : 0
    let bugSection = bugSections[section]
    return bugSection.bugs.count + adjustment
  }
```

Here you're running a check to see if the table view is in editing mode or not. If it is, you add an extra "virtual" row into each section.

Next, modify `tableView(_:cellForRowAtIndexPath:)` to the following:

```
  override func tableView(tableView: UITableView,
    cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell
  {

    let cell = tableView.dequeueReusableCellWithIdentifier(
      "BugCell", forIndexPath: indexPath)
    let bugSection = bugSections[indexPath.section]

    if indexPath.row >= bugSection.bugs.count && editing {
      cell.textLabel?.text = "Add Bug"
      cell.detailTextLabel?.text = nil
      cell.imageView?.image = nil
    } else {
      let bug = bugSection.bugs[indexPath.row]
      cell.textLabel?.text = bug.name
      cell.detailTextLabel?.text =
        ScaryBug.scaryFactorToString(bug.howScary)
      guard let imageView = cell.imageView else {
        return cell
      }
      if let bugImage = bug.image {
        imageView.image = bugImage
      } else {
        imageView.image = nil
      }
    }
    return cell
  }
```

Here you've added some extra code to check if this is the "virtual" row, and if so you set the text label to "Add Bug".

Next, you need to tell the table view that there should be some new rows when the user switches back and forth between edit modes. To do this, override `setEditing:animated:` (remember this gets called when the editing mode is toggled on the view controller, which is called automatically by the edit button you added to the navigation bar):

```
override func setEditing(editing: Bool, animated: Bool) {
  super.setEditing(editing, animated: animated)
  if editing {
    tableView.beginUpdates()
    for (index,bugSection) in bugSections.enumerate() {
      let indexPath = NSIndexPath(forRow: bugSection.bugs.count,
        inSection: index)
      tableView.insertRowsAtIndexPaths([indexPath],
        withRowAnimation: .Automatic)
    }
    tableView.endUpdates()
    tableView.setEditing(true, animated: true)
  } else {
    tableView.beginUpdates()
    for (index,bugSection) in bugSections.enumerate() {
      let indexPath = NSIndexPath(forRow: bugSection.bugs.count,
        inSection: index)
      tableView.deleteRowsAtIndexPaths([indexPath],
        withRowAnimation: .Automatic)
    }
    tableView.endUpdates()
    tableView.setEditing(false, animated: true)
  }
}
```

Let's go over this section by section:

1. Calls the superclass's `setEditing(_:animated:)`. This is important because your superclass (`UITableViewController`) implements this to set the editing mode on the table view for you.

2. Checks to see if you are entering edit mode or not.

3. If you are, you need to insert a "virtual" row for each section. Whenever you make a bunch of updates to a table view at once, it is best to wrap them in `beginUpdates/endUpdates` calls so that the animations perform simultaneously.

4. Loop through all of the bug sections. Insert a new row at the end of each section. This will cause the table view to call your tableView(_:cellForRowAtIndexPath:) method for the new section, at which point you'll configure the cell to say "Add Bug".

5. Call `endUpdates` now that you are done the batch insertion.

6. Similar logic, except when you exit editing mode, you want to remove all of the "virtual" rows.

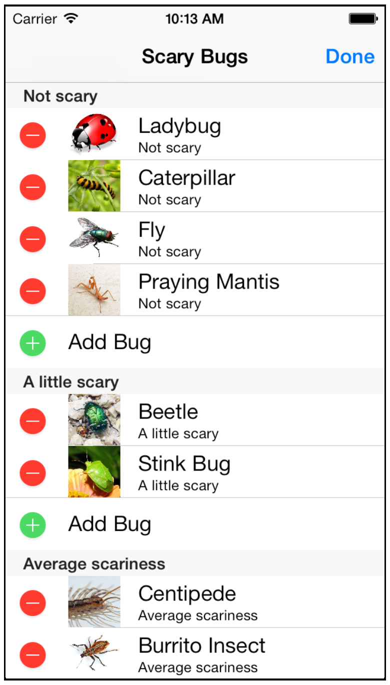
Finally, add this new method:

```
override func tableView(tableView: UITableView,
  editingStyleForRowAtIndexPath indexPath: NSIndexPath) ->
  UITableViewCellEditingStyle {

  let bugSection = bugSections[indexPath.section]
  if indexPath.row >= bugSection.bugs.count {
    return .Insert
  } else {
    return .Delete
  }
}
```

This configures the editing style for the new "virtual" row to be editing rather than delete. This causes a little green + button to appear to the left of the cell (which indicates creating a new row), rather than the red – button (which indicates deleting the row).

Build and run, and tap the Edit button. You should see new "Add Bug" rows animate into place:



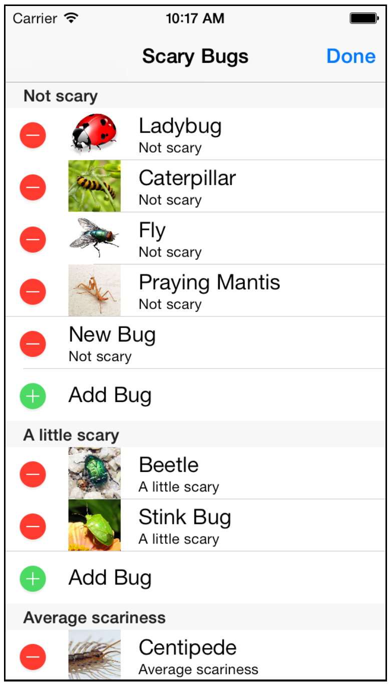
Of course, tapping the + button does nothing yet. When this is tapped, `tableView(_:commitEditingStyle:forRowAtIndexPath:)` is called, with the editing style set to insert. So you need to add code into that method, similar to the way you added code for the delete case in the previous challenge.

So add the following code to the end of `tableView(_:commitEditingStyle:forRowAtIndexPath:)`

```
else if editingStyle == .Insert {
  let bugSection = bugSections[indexPath.section]
  let newBug = ScaryBug(withName: "New Bug", imageName: nil,
    howScary: bugSection.howScary)
  bugSection.bugs.append(newBug)
  tableView.insertRowsAtIndexPaths([indexPath], withRowAnimation:
    .Automatic)
}
```

This creates a new bug in the appropriate section. Note that it does the two steps you always have to do: update the model, and tell the table view about the update (so it animates nicely).

Build and run, and now if you tap the + button it creates a new bug:



One final step for polish. Currently, if you don't tap exactly on the + button, it doesn't create a new row. A user might try to tap the row instead, and have nothing happen, which would be confusing.

By default, cell selection is disabled when a table view is in edit mode. To fix this, add the following line to the bottom of `viewDidLoad`:

```
tableView.allowsSelectionDuringEditing = true
```

However, you don't want the user to be able to select any row in editing mode exept for the "Add Bug" row. To do this, add the following new method at the bottom of the file:

```
override func tableView(tableView: UITableView,
  willSelectRowAtIndexPath indexPath: NSIndexPath)
  -> NSIndexPath? {

  let bugSection = bugSections[indexPath.section]
  if self.editing && indexPath.row < bugSection.bugs.count {
    return nil
  }
  return indexPath
}
```

This method gets called right after a user selects a row. You have an opportunity to either return the index path (and that index path gets selected), or return nil (to cancel the selection). Here you return nil if the table view is in editing mode and it's any row but the "Add Bug" row.

Finally, add this new method to the bottom of the file:

```
override func tableView(tableView: UITableView,
  didSelectRowAtIndexPath indexPath: NSIndexPath) {

  tableView.deselectRowAtIndexPath(indexPath, animated: true)
  let bugSection = bugSections[indexPath.section]
  if indexPath.row >= bugSection.bugs.count && editing {
    self.tableView(tableView, commitEditingStyle: .Insert,
      forRowAtIndexPath: indexPath)
  }
}
```

This makes it so when you tap the "Add Bug" row in editing mode, it calls the `tableView(_:commitEditingStyle:forRowAtIndexPath:)` method you wrote earlier to add a new row.

Build and run, enter edit mode, and tap the Add Bug row anywhere to add a new bug. Go forth, and create some bugs!