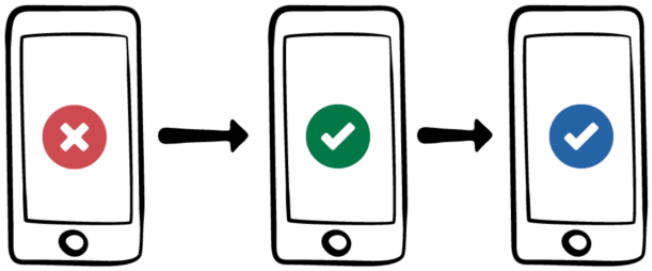
BEGINNING iOS UNIT & UI TESTING



HANDS-ON CHALLENGES

Beginning iOS Unit and UI Testing

Joshua Greene

Notice of Rights

Notice of Liability

Trademarks

5 UI Testing

By Joshua Greene

UI tests can be fun to write and even more fun to run— it’s neat to see your app running itself as it goes through your test cases!

While it can sometimes be difficult to uniquely identify which elements are on screen, there's fortunately an even easier way to write these tests... you'll learn about it in the next video! For now, you'll practice writing one more UI test by hand in this quick challenge. :]

Challenge

Your challenge is to write a new UI test that navigates to the pancake house details screen and then assert that elements *unique* to the this screen are shown.

Hints

- Remember that you may need to tap on the "pancake houses" button if the app is running on a smaller device.
- You can pick any pancake house cell to tap, but whichever you choose will determine the info on the details screen.
- Determining you're actually on the details screen is a bit tricky: while you could look for the name of the pancake house, it's possible this might give a false positive! This is because the pancake house name appears on multiple screens.

  Consequently, you also need to assert another element unique to this screen exists. The map element works great for this, as there's only a map shown on the details screen.

  So, if you look for the name *and* map, this would ensure you're in the right place.

Try to solve this challenge by yourself first. If you get stuck, check out the next page for a sample solution.

Solution

Open `StackReviewUITests.swift` and add the following test method to the class:

```
func testCanNavigateToPancakeHouseScreen() {

  if horizontalSizeClass != .regular {
    let pancakeHouseButton = app.navigationBars.buttons["Pancake Houses"]
    pancakeHouseButton.tap()
  }

  let tableText = app.tables.staticTexts["Stack 'em High"]
  let pancakeText = app.staticTexts["Stack 'em High"]
  let map = app.maps.element

  tableText.tap()

  XCTAssertTrue(pancakeText.exists)
  XCTAssertTrue(map.exists)
}
```

Remember that the `horizontalSizeClass` check is to ensure we're on the master screen that shows all of the pancake houses, just in case the UI tests are running on a smaller device, such as an iPhone in portrait mode.

You might think this looks a little strange: how can you get the `pancakeText` and the `map` right at the start of the test? Those are elements on the details screen, but you haven't even tapped the table view cell yet!

Remember, these are proxy objects. `XCUIElements` represent views, but they aren't actually views themselves.
`app.maps.element` describes a map element, but the test framework doesn't actually go out and link it to a view until you use it. That's why the order in this test works: you can get all your `XCUIElements` set up at the start, but they'll only be queried when you call a method on them or examine a property such as `exists`.

Über challenge

You may have noticed that there's currently duplicate code in these two UI tests. Just like your unit tests, you should strive to follow the "Red-Green-Refactor" cycle when writing your UI tests too!

Create a new helper method called `tapPancakeHousesButtonIfNeeded` and refactor out this duplicate code.