CS121 Tutorial 5

Intro to testing in Xcode!

Purple bubbles give you information you’ll need to know.

Yellow Bubbles tell you what to do.

Orange bubbles tell you what you’re not expected to understand yet. 😊
1. Create a new iOS single view app for the iPad.

2. Call it Tutorial5.

3. Do not check include unit tests.

Xcode will set up the unit testing framework for you. But in this tutorial you’ll learn how to add unit tests to an existing project that isn’t set up for the tests.
Add a new Model class to the object. (It should use NSObject as its base class.)
Define the interface as shown.
Implement the class as shown. Then run to make sure there are no problems.

We do advocate test-driven development but that isn’t the point of this lab. Here we are showing you how to build tests for existing code.
1. Select File then New.

2. Select New Target.

A target is something that is built by the project. More on that in a minute.
1. Select Other.
2. Cocoa Touch Unit Testing Bundle.
3. Click Next.
1. Name the new “product” unitTests.

2. Select Tutorial5.

3. Click Finish.
Open unitTests.h.

Xcode built a new class for us but it is not like other classes we’ve seen. For one thing, it uses the SenTestingKit.

Our unitTests class is a subclass of SenTestCase.
1. Select the tutorial project in the project navigator.

2. Select Tutorial5 in the Target list.

A target is something that is built by the project. When we create an iOS application project, Xcode knows we will want to build the app. That target, Tutorial5, is created for us automatically.
Select the `unitTests` target. `unitTests` is our new target. Its code will compile and run when we direct Xcode to Test our `Tutorial5` app. Well, it will after we hook it up correctly.
Open the product tab.

At the moment Test is not an option.
1. Select Tutorial5 in the project navigator.

2. Select unitTests in the Target list.

3. Select Build settings

4. Select All
Type Bundle Loader in the search field. There should be only one match.
1. Expand Bundle Loader.

2. Select Debug then double click to open its text field.
Type the text shown.
When you hit return \$(TARGET_BUILD_DIR) will be replaced by the directory Xcode uses to store the debug executable.
Repeat for the Release version.
When you hit return, the target directory is again filled in for you. Note the release executable is in a different directory than the debug version.
There is one match.

Now search for Test host.
1. Expand Test Host.

2. Select Debug then double click to open its text field. Enter $(BUNDLE_LOADER)$.
When you hit return it will show the same directory as for debug bundle loader.

Repeat for the Release version.
1. Open the Product tab.

2. Select Edit Scheme...
1. Select unitTests.

2. Select Test.

3. Make sure these boxes are checked.
1. Select Tutorial5.

2. Select Test.

3. Click the +.
1. Select unitTests.

2. Click Add.
unitTests should now appear under Tutorial5’s tests. Make sure these boxes are checked.

Click OK.
1. Open the product tab again.

2. Test is now an option. Choose it.

A build succeeded message will flash briefly. After that a test failed message will flash.
1. Click here to open the log navigator.

2. Select Test unitTests.

This message tells us one of our tests failed.

You might be thinking that you didn’t write any tests. Go to the next slide to see what happened.
1. Open the project navigator.

2. Open the unitTests source file.

Xcode provided us with template code including a test that fails!

Next we’ll write some real tests that should pass.

3. Delete this method.
1. Open the unitTests header file.

2. Import our model header file.

3. We’ll need a model object to test.

4. We’ll create two tests.
1. Open the unitTests source file.

2. Create our Model object to test.

When we choose to Test, Xcode runs our setUp method, then all of the tests we’ve created, and finally the tearDown method. In this example, our setUp method creates the Model object that we will test.
Add the `testInitialValue` code.

`STAssertEquals` fails if its first two arguments are not equal. If it fails, the string `"testInitialValues failed."` is written to the console.
Add the testSetAndGet method.

I want to set the value of the data fields to something other than 0. I also want each data field to be unique.

Note that my test does not check that data[i] is unaffected by a change to data[i+1]. How would you rewrite the test to address that issue?
Run the new tests then open the log. Both tests should pass!
One shortcoming of our model implementation is that we fail to test bounds of the index in the get and set. We'll create a test that checks whether invalid input is handled properly in the get method.

Add this test to our unitTests interface.
Implement our new test.
Run the new tests then open the log. Our new test failed!
1. Open our Model source file.

Add an assertion to catch bad input in the getValue method.
Run the tests again then open the log. Now our new test succeeded!
Update the tests as follows:
1. setUp should fail if testModel is not allocated (i.e. it gets a nil pointer). Check out the link below for STAssert options that will be useful.
2. Add another test to check that the setValue method
   a. Checks that the index is valid
   b. Checks that the value is non-negative
3. Fix our Model implementation as needed so that all tests pass.

Appendix B at the following link gives the STAssert macros that are available.