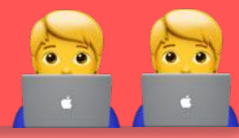
Mobile Dev Diary



Swift Testing parametrized tests

scroll to find out

mobiledevdiary.com mdd

Parametrized tests 🧪

What is parametrized test? 🤔

- special type of test with arguments,
- parameterized test is defined once, but is executed as many times as the number of input arguments.
- Swift Testing framework feature available in Xcode 16.

mdd

Test case 

Let's take a property "containsEmoji" in String extension as an example ⤵️

```
extension String {
  var containsEmoji: Bool {
    ...
  }
}
```

It returns true when a text contains at least one emoji and false when there is no emoji at all.

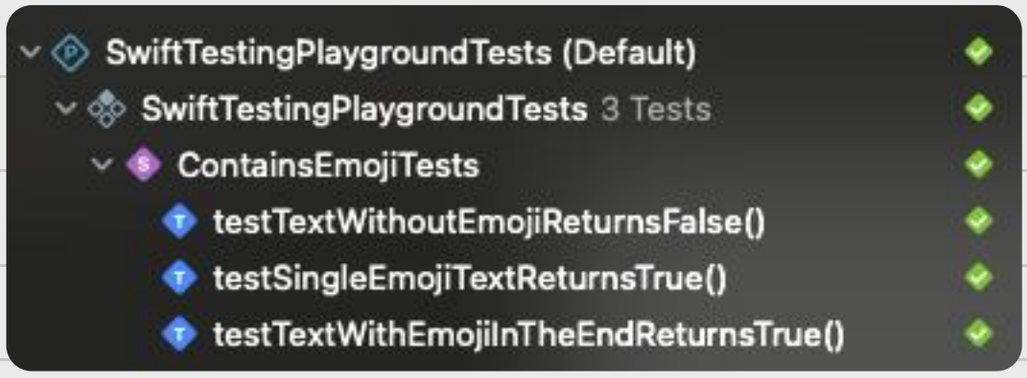
mdd

XCTest - good 🙂

In a traditional XCTest approach, three separate tests can be added to validate three different Strings ⤵️

```
class ContainsEmojiTests: XCTestCase {
  func testTextWithoutEmojiReturnsFalse() {
    XCTAssertFalse("ABC".containsEmoji)
  }

  func testSingleEmojiTextReturnsTrue() {
    XCTAssertTrue("🚀".containsEmoji)
  }

  func testTextWithEmojiInTheEndReturnsTrue() {
    XCTAssertTrue("ABC 👾".containsEmoji)
  }
}
```

Then each case is listed in the output ⤵️



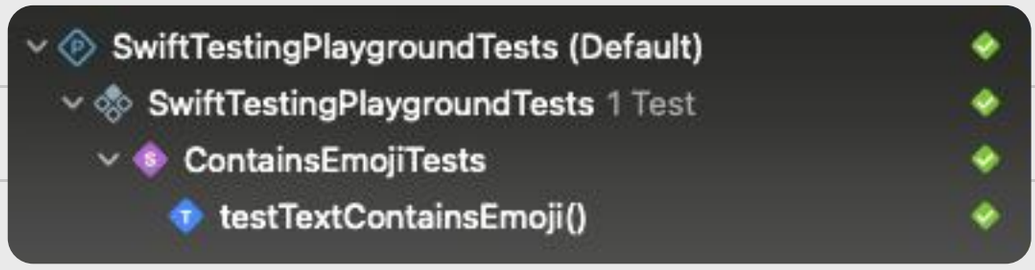
mdd

XCTest - better 😀

Tests suite can be refactored to a one test that iterates through test cases avoiding the code duplications ⤵️

```
class ContainsEmojiTests: XCTestCase {

  struct TestCase {
    let text: String
    let result: Bool
  }

  static let testCases: [TestCase] = [
    .init(text: "ABC", result: false),
    .init(text: "🚀", result: true),
    .init(text: "ABC 👾", result: true)
  ]

  func testTextContainsEmoji() {
    Self.testCases.forEach { test in
      XCTAssertTrue(test.text.containsEmoji == test.result)
    }
  }

}
```

the downside of the approach is loss of separate descriptions ⤵️



mdd

Swift Testing - the best 🤩

There are a few steps to transform the test into a parametrized test:

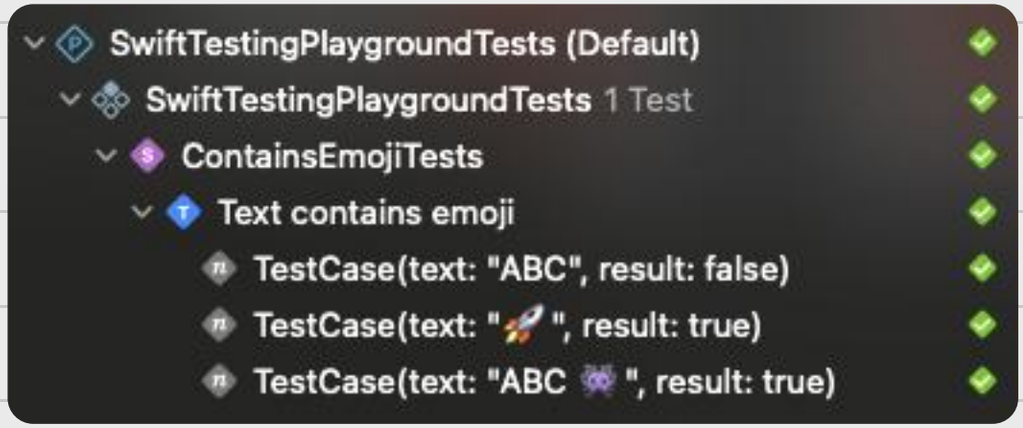
1. remove the “XCTestCase” inheritance, and replace keyword class ➡️ struct,
2. modify the test function signature to accept the TestCase,
3. use the @Test macro and pass testCases as an argument ⤵️

```
struct ContainsEmojiTests {

  struct TestCase {
    let text: String
    let result: Bool
  }

  static let testCases: [TestCase] = [
    .init(text: "ABC", result: false),
    .init(text: "🚀", result: true),
    .init(text: "ABC 👾", result: true)
  ]

  @Test("Text contains emoji", arguments: testCases)
  func textContainsEmoji(test: TestCase) {
    #expect(test.text.containsEmoji == test.result)
  }

}
```

mdd

Swift Testing - the best 🤩

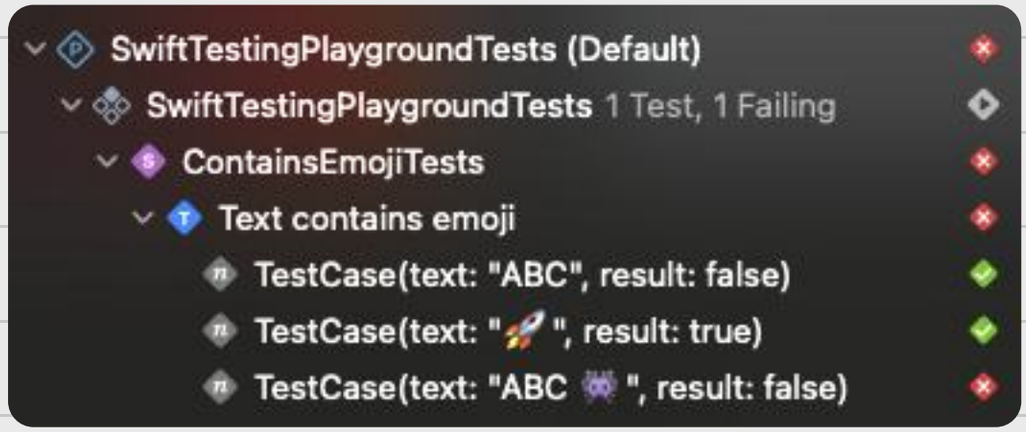
The huge advantage of the Swift Testing parametrized tests is the capability to check the output from the execution in the test navigator in a way that all test inputs (TestCase for us) are listed together with execution results ⤵️



mdd

Swift Testing - the best 🤩

In case of an error, it's possible to check the test navigator and see for which input test fails ⤵️




```
@Test("Text contains emoji", arguments: testCases)
Arguments
  test : TestCase(text: "ABC 👾", result: false)
  TestCase
func textContainsEmoji(test: TestCase) {
  #expect(test.text.containsEmoji == test.result)
Results
  test.text.containsEmoji : true
  Bool
  test.result : false
  Bool
}
```

Find this Interesting?

Follow me for more!

mobiledevdiary.com mdd