EPFL

Unit Testing

Effective Programming in Scala

Intent vs Reality

How to make sure that the program we wrote matches our initial intent?

Intent vs Reality

How to make sure that the program we wrote matches our initial intent?

By *reading* it: works fine for small and concrete things, but requires a lot of effort to reason about a whole program or very abstract code.

Intent vs Reality

How to make sure that the program we wrote matches our initial intent?

By *reading* it: works fine for small and concrete things, but requires a lot of effort to reason about a whole program or very abstract code.

The *type system* ensures that program parts are correctly combined together, but relies on the assumption that we correctly modeled things in the first place. Also, only a small class of errors can be caught by the type system.

Software Is Soft

Code bases evolve over time and by changing small things here and there it is easy to break a piece of code that was previously correctly working.

We need robust ways to prevent these regressions to happen.

Increase Confidence

In the following lessons, we present different testing techniques that we can use to increase our confidence that the code we have written matches our initial intent.

Unfortunately, it is very hard to reach a 100% confidence level. All the techniques have different tradeoffs in terms of costs, coverage, and robustness.

Unit Testing

The simplest way to test a program is to write another program that calls the program under test with parameters for which we know what the expected result should be.

Example: Program Under Test

For the rest of the lesson, we will test the following program:

```scala
// File src/main/scala/testing/Program.scala
package testing

/** @return the sum of ‘x‘ and ‘y‘ */
def add(x: Int, y: Int): Int = ...

/** @return the ‘n‘th Fibonacci number (the first number is 0) */
def fibonacci(n: Int): Int = ...
```

Writing Tests in a Project

In sbt projects, tests go to the `src/test/scala/` directory (as opposed to `src/main/scala` for the sources of the program).

It is common practice to use names that mirror the names of the source files.

For our example, we create a test suite in the file `src/test/scala/testing/ProgramSuite.scala`.

Getting Started With MUnit

There are several testing libraries in Scala. In this lesson, we show how to get started with MUnit, a simple testing library.

Add the following settings to your `build.sbt` file:

```
libraryDependencies += ”org.scalameta” %% ”munit” % ”0.7.19” % Test
testFrameworks += new TestFramework(”munit.Framework”)
```

Defining a Test Suite

A test suite is a class that extends `munit.FunSuite`:

```
// File src/test/scala/testing/ProgramSuite.scala
package testing

class ProgramSuite extends munit.FunSuite:
  test("add") {
    // TODO Write test specification for the ‘add‘ method
  }

  test("fibonacci") {
    // TODO Write test specification for the ‘fibonacci‘ method
  }
end ProgramSuite
```

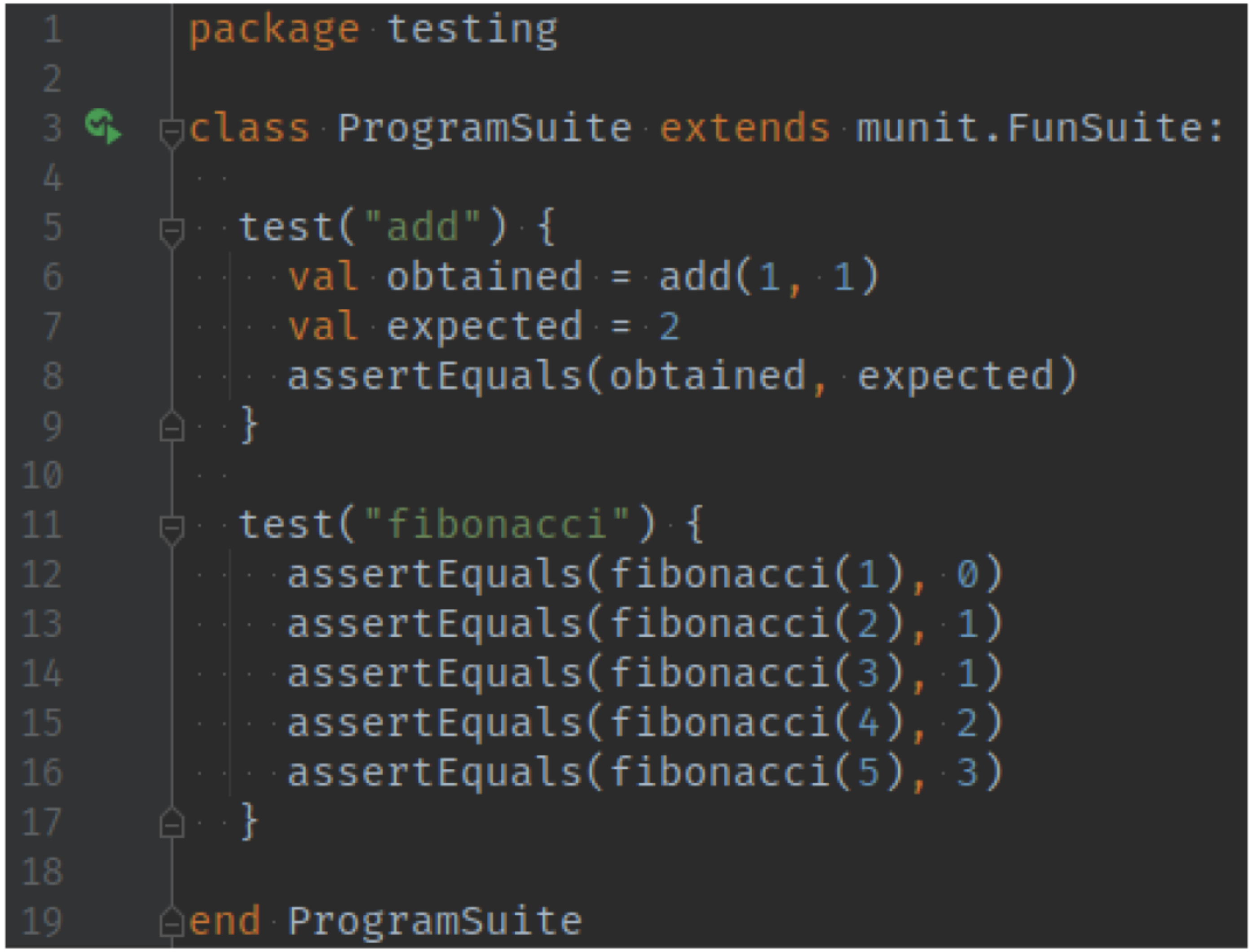
Writing a Test Specification

We use the `test` and `assertEquals` methods, inherited from `munit.FunSuite`, to write our test specification:

```
class ProgramSuite extends munit.FunSuite:
  test(”add”) {
    val obtained = add(1, 1)
    val expected = 2
    assertEquals(obtained, expected)
  }
end ProgramSuite
```

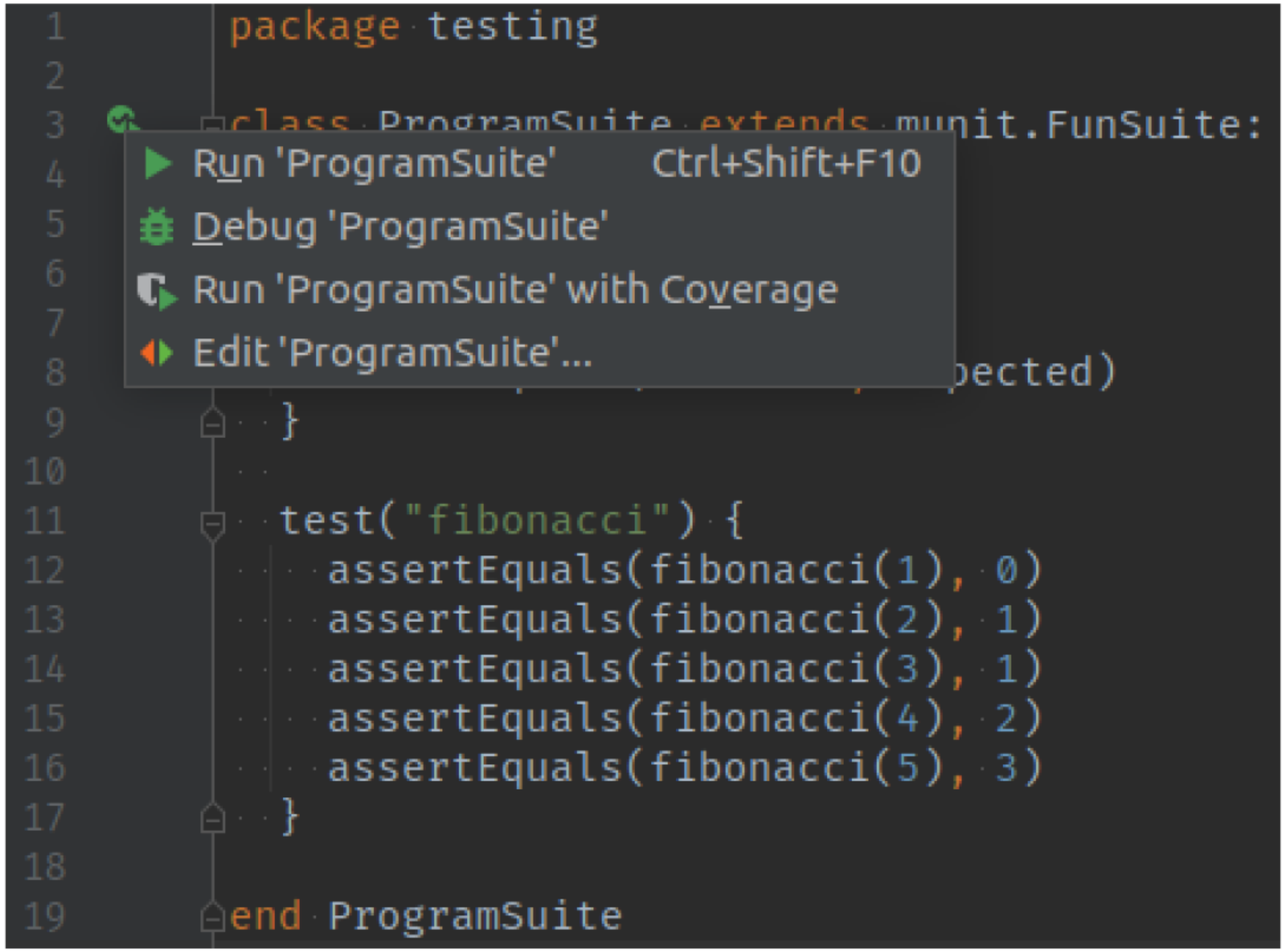
We give a name to the test specification (here, “add”), and we write the implementation in a second argument.

Running the Tests From IntelliJ (1)



```scala
package testing

class ProgramSuite extends munit.FunSuite:

  test("add") {
    val obtained = add(1, 1)
    val expected = 2
    assertEquals(obtained, expected)
  }

  test("fibonacci") {
    assertEquals(fibonacci(1), 0)
    assertEquals(fibonacci(2), 1)
    assertEquals(fibonacci(3), 1)
    assertEquals(fibonacci(4), 2)
    assertEquals(fibonacci(5), 3)
  }

end ProgramSuite
```

Running the Tests From IntelliJ (2)

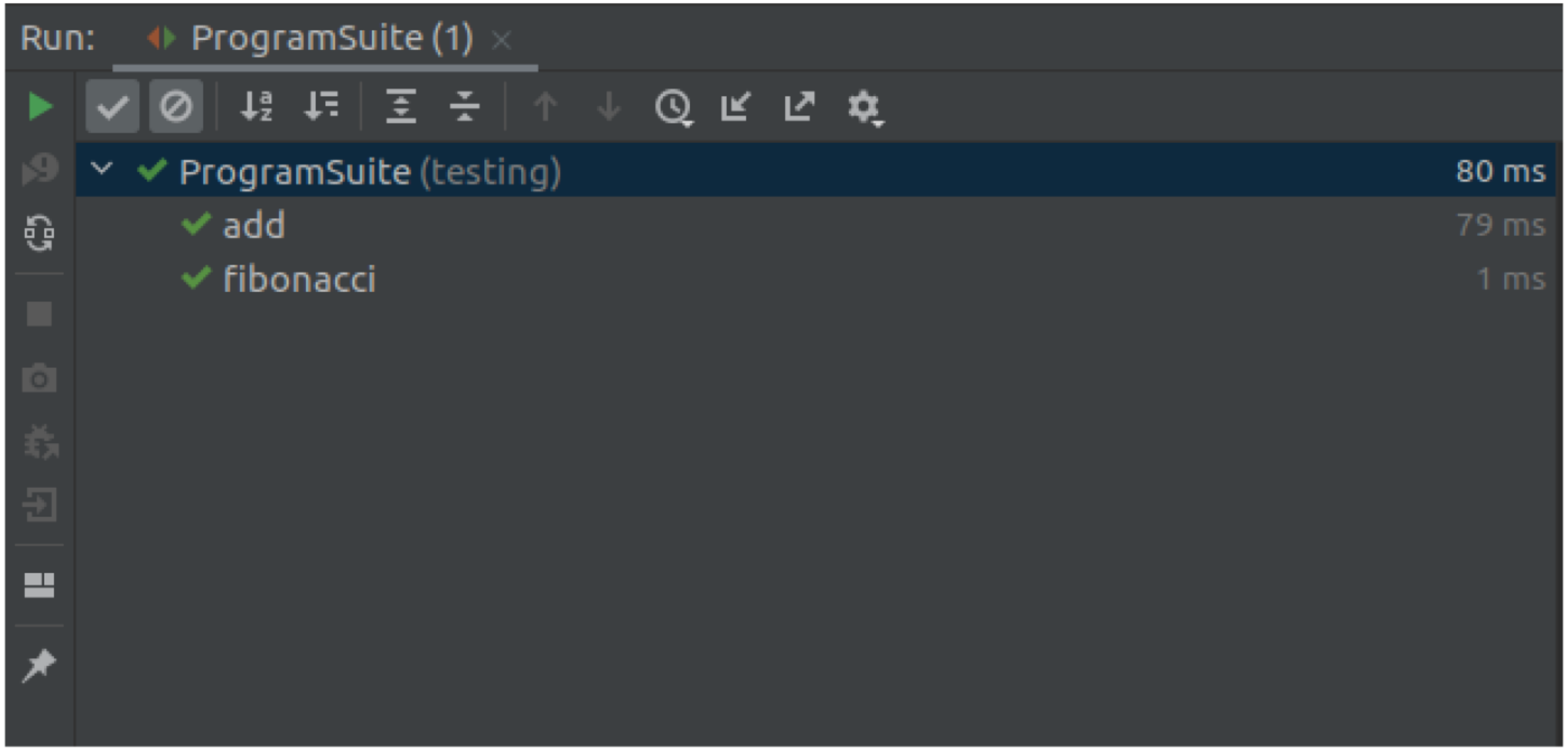


```
package testing

class ProgramSuite extends munit.FunSuite:
```

Run 'ProgramSuite' Ctrl+Shift+F10
Debug 'ProgramSuite'
Run 'ProgramSuite' with Coverage
Edit 'ProgramSuite'...

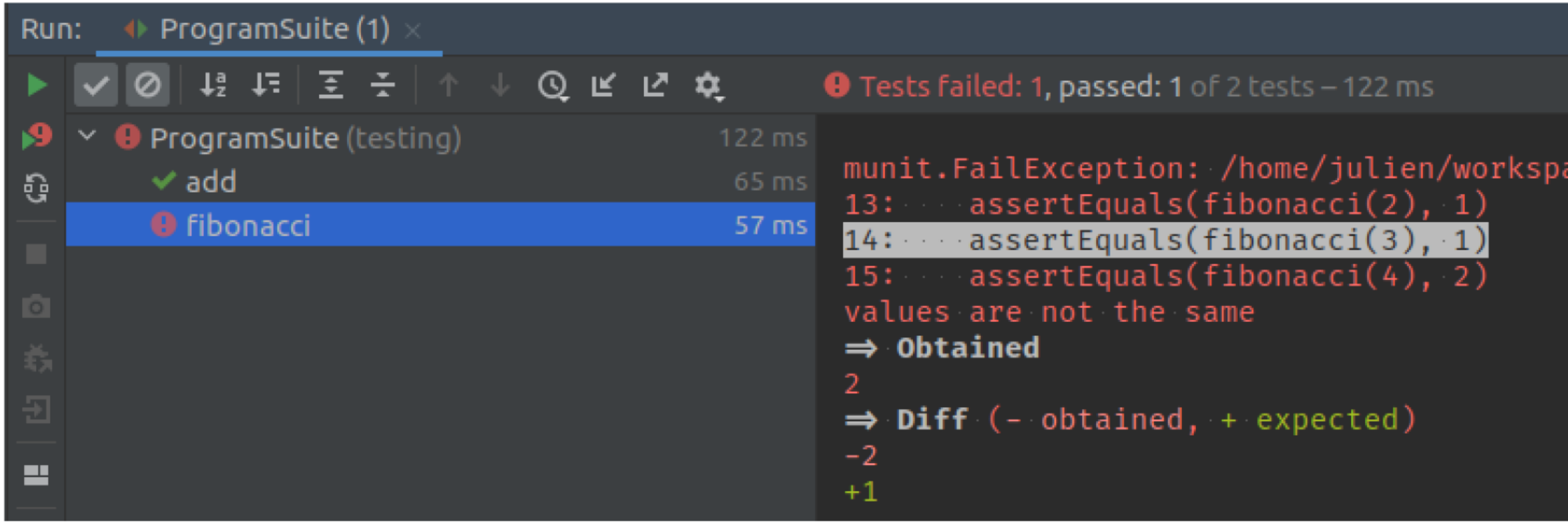
```
                                              pected)
  }

  test("fibonacci") {
    assertEquals(fibonacci(1), 0)
    assertEquals(fibonacci(2), 1)
    assertEquals(fibonacci(3), 1)
    assertEquals(fibonacci(4), 2)
    assertEquals(fibonacci(5), 3)
  }

end ProgramSuite
```

Reading the Test Report (1)



```
Run:  ProgramSuite (1) ×

✓ ProgramSuite (testing)                         80 ms
    ✓ add                                        79 ms
    ✓ fibonacci                                   1 ms
```

Reading the Test Report (2)

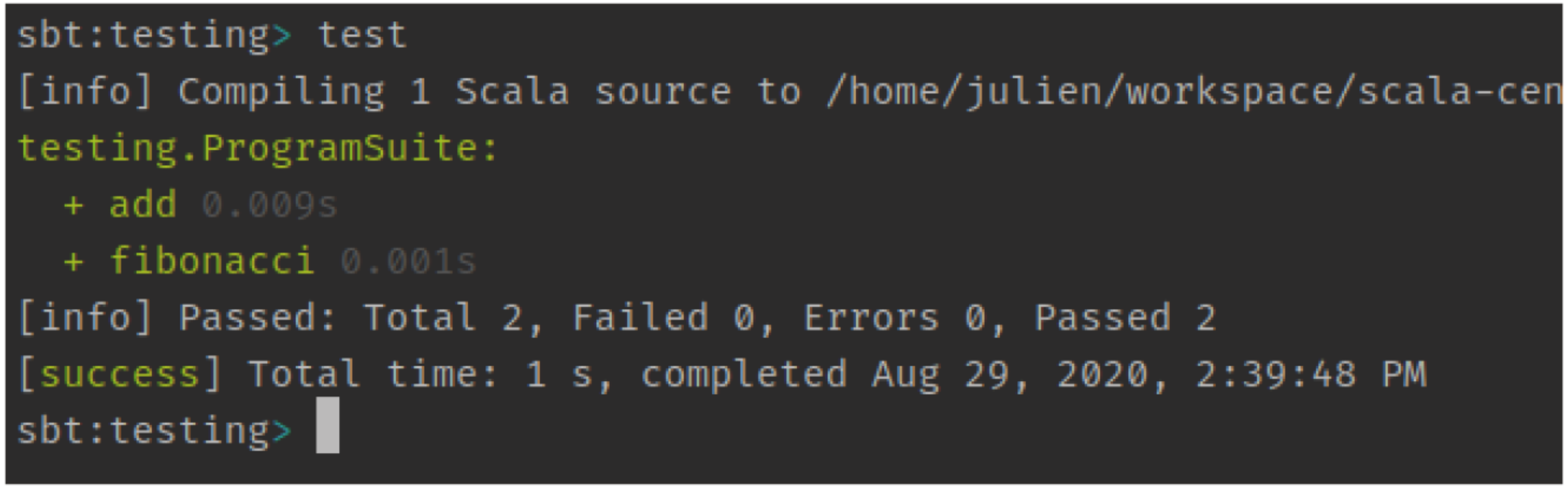


```
Tests failed: 1, passed: 1 of 2 tests – 122 ms

ProgramSuite (testing)        122 ms
    add                        65 ms
    fibonacci                  57 ms
```

```
munit.FailException: /home/julien/workspa
13:     assertEquals(fibonacci(2), 1)
14:     assertEquals(fibonacci(3), 1)
15:     assertEquals(fibonacci(4), 2)
values are not the same
⇒ Obtained
2
⇒ Diff (- obtained, + expected)
-2
+1
```

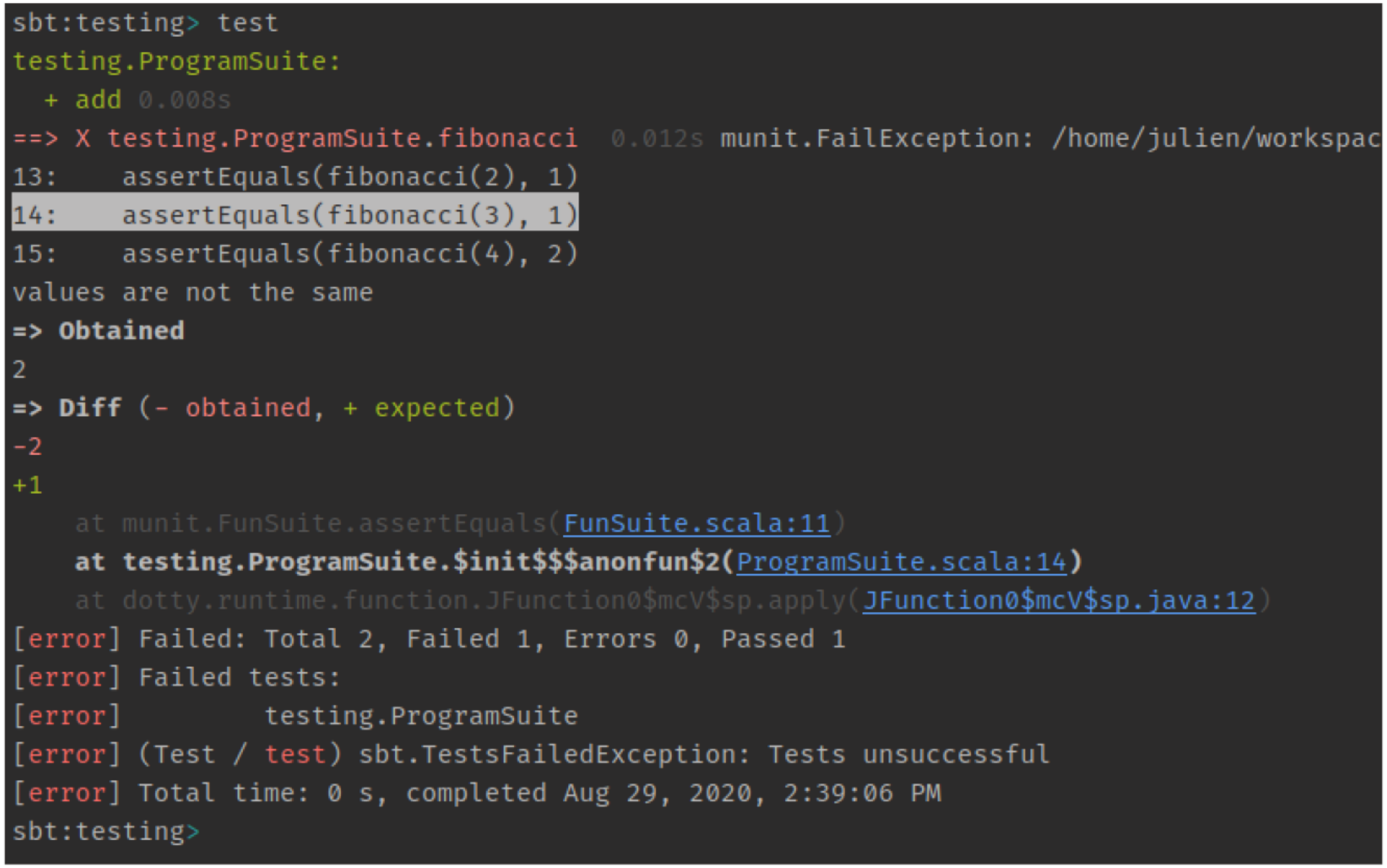
Running the Tests From Sbt (1)

From the sbt prompt, invoke the task `test`:



```
sbt:testing> test
[info] Compiling 1 Scala source to /home/julien/workspace/scala-cen
testing.ProgramSuite:
  + add 0.009s
  + fibonacci 0.001s
[info] Passed: Total 2, Failed 0, Errors 0, Passed 2
[success] Total time: 1 s, completed Aug 29, 2020, 2:39:48 PM
sbt:testing>
```

Running the Tests From Sbt (2)



```
sbt:testing> test
testing.ProgramSuite:
  + add 0.008s
==> X testing.ProgramSuite.fibonacci  0.012s munit.FailException: /home/julien/workspac
13:    assertEquals(fibonacci(2), 1)
14:    assertEquals(fibonacci(3), 1)
15:    assertEquals(fibonacci(4), 2)
values are not the same
=> Obtained
2
=> Diff (- obtained, + expected)
-2
+1
    at munit.FunSuite.assertEquals(FunSuite.scala:11)
    at testing.ProgramSuite.$init$$$anonfun$2(ProgramSuite.scala:14)
    at dotty.runtime.function.JFunction0$mcV$sp.apply(JFunction0$mcV$sp.java:12)
[error] Failed: Total 2, Failed 1, Errors 0, Passed 1
[error] Failed tests:
[error]         testing.ProgramSuite
[error] (Test / test) sbt.TestsFailedException: Tests unsuccessful
[error] Total time: 0 s, completed Aug 29, 2020, 2:39:06 PM
sbt:testing>
```

Summary

Testing your programs can increase your level of confidence in the fact that they behave as expected. Furthermore, tests are a good way to prevent regressions.

Unit tests let you check the result of a program for some specific inputs. However, you have to think about all the corner cases to check.