

گزارش آزمایشگاه مهندسی نرم افزار

پاییز ۱۴۰۱



دانشکده مهندسی کامپیوتر

گزارش ۳: تبدیل نیازمندی‌ها به موارد آزمون با استفاده از روش ایجاد مبتنی بر رفتار (BDD)

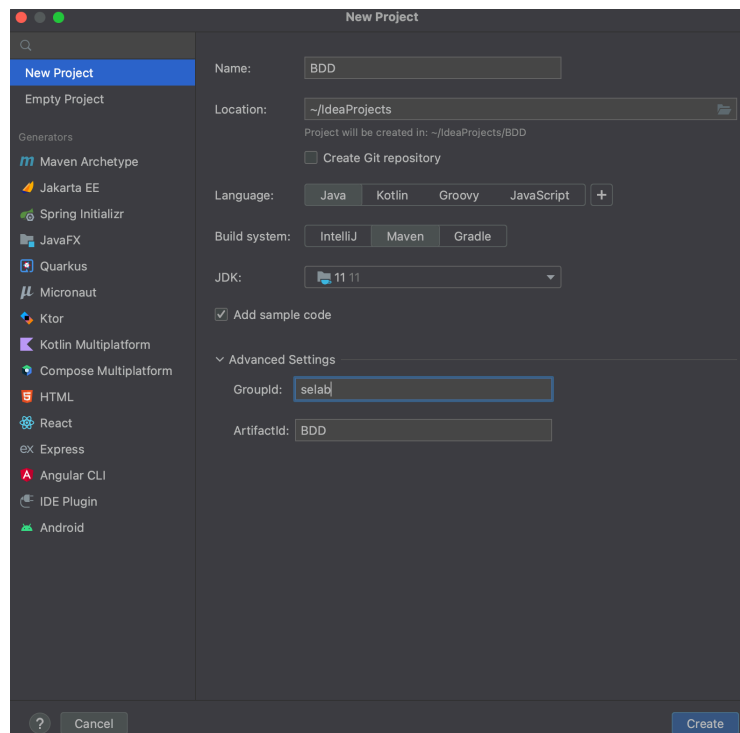
رستا روغنی (۹۷۱۰۵۹۶۳)

مهرانه نجفی (۹۷۱۰۴۷۰۷)

۱ سناریو: جمع دو عدد

۱.۱ راه اندازی پروژه

۱. یک پروژه‌ی Maven ایجاد می‌کنیم.



شکل ۱: ایجاد یک پروژه‌ی جدید Maven

۲. به شکل زیر در فایل pom.xml تغییر ایجاد می‌کنیم و dependency ها را اضافه می‌کنیم. (چون ورژن ما بالاتر بود، بعضی از dependency ها با دستورکار تفاوت دارند)

```

pom.xml (BDD)
6
7 <groupId>selab</groupId>
8 <artifactId>BDD</artifactId>
9 <version>1.0-SNAPSHOT</version>
10
11 <dependencies>
12   <dependency>
13     <groupId>io.cucumber</groupId>
14     <artifactId>cucumber-junit</artifactId>
15     <version>7.8.1</version>
16     <scope>test</scope>
17   </dependency>
18   <dependency>
19     <groupId>junit</groupId>
20     <artifactId>junit</artifactId>
21     <version>4.13.2</version>
22     <scope>test</scope>
23   </dependency>
24   <dependency>
25     <groupId>io.cucumber</groupId>
26     <artifactId>cucumber-java</artifactId>
27     <version>7.8.1</version>
28     <scope>test</scope>
29   </dependency>
30 </dependencies>

```

شکل ۲: اضافه کردن Dependencies به فایل pom.xml

۳. Maven->Lifecycle->test را اجرا می‌کنیم و می‌بینیم که به درستی Build می‌شود.

The screenshot displays an IDE with a Maven project configuration file, `pom.xml`, and its execution output in the console.

pom.xml (BDD) x

```
<groupId>io.cucumber</groupId>
<artifactId>cucumber-java</artifactId>
<version>7.8.1</version>
<scope>test</scope>
</dependency>
</dependencies>

<properties>
<maven.compiler.source>11</maven.compiler.source>
<maven.compiler.target>11</maven.compiler.target>
<project.build.sourceEncoding>UTF-8</project.build.
</properties>
</project>
```

Maven

- BDD
 - Lifecycle
 - clean
 - validate
 - compile
 - test
 - package
 - verify
 - install
 - site
 - deploy
 - Plugins
 - Dependencies

project

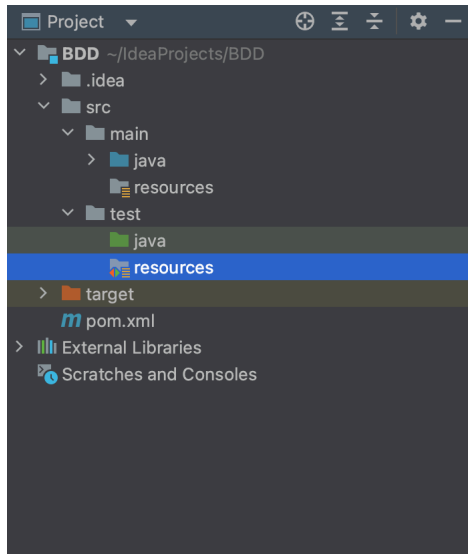
2 sec, 595 ms

```
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ BDD ---
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 1.095 s
[INFO] Finished at: 2022-11-18T19:09:03+03:30
[INFO] -----

Process finished with exit code 0
```

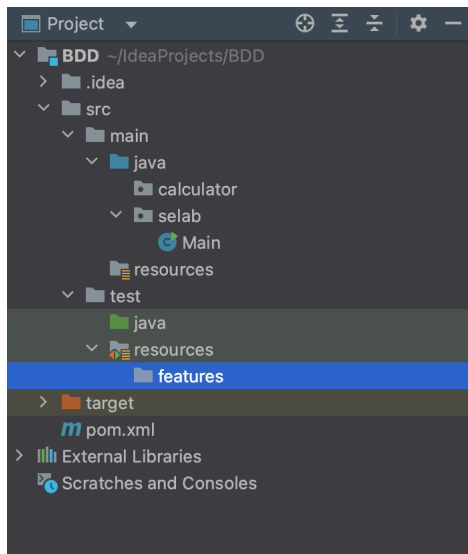
شكل ٣: عمليات build موفق

۴. در پوشه‌ی test یک directory جدید به نام resources درست می‌کنیم و آن را Test Resource Root می‌کنیم.



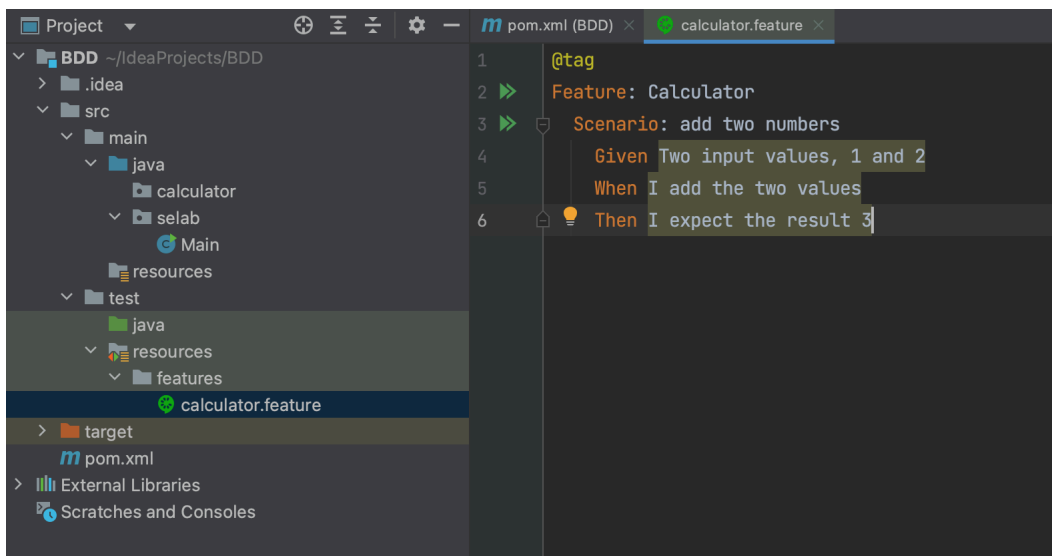
شکل ۴: مشخص کردن Test Resource Root

۵. یک package به نام calculator و یک directory به نام features در آدرس‌های گفته‌شده درست می‌کنیم.



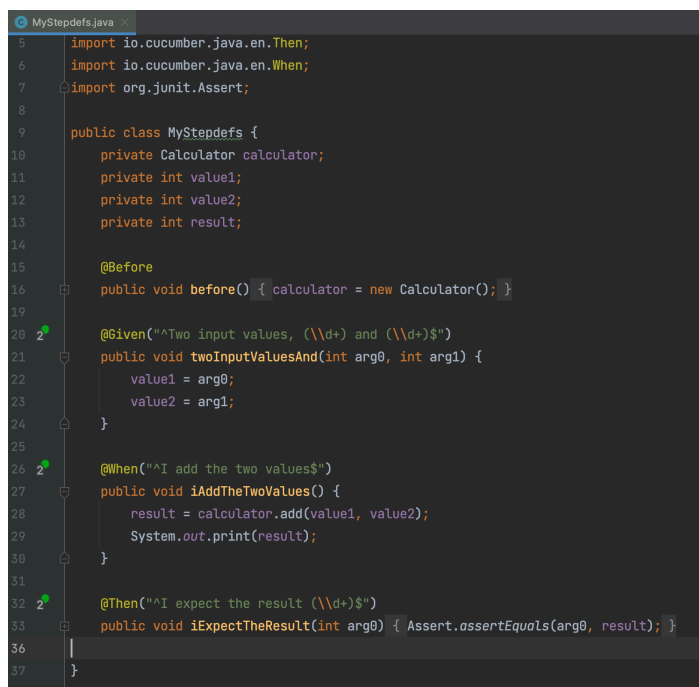
شکل ۵: اضافه کردن پکیج calculator و دایرکتوری features

۶. در بخش features یک فایل به نام calculator.feature اضافه می‌کنیم و در آن سناریوی جمع دو عدد را می‌نویسیم:



شکل ۶: اضافه کردن فایل calculator.feature

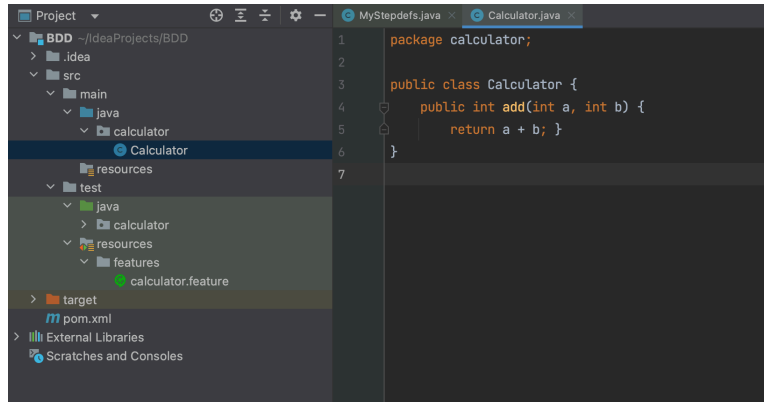
در آدرس test->java یک directory با نام calculator ایجاد می‌کنیم و پس از آن برای هر خط از سناریوی یک step definition درست می‌کنیم. فایل MyStepdefs تولید می‌شود که در آن تغییرات گفته‌شده را اعمال می‌کنیم تا به شکل زیر دربیاید:



شکل ۷: تعریف قدم‌ها در فایل MyStepdefs

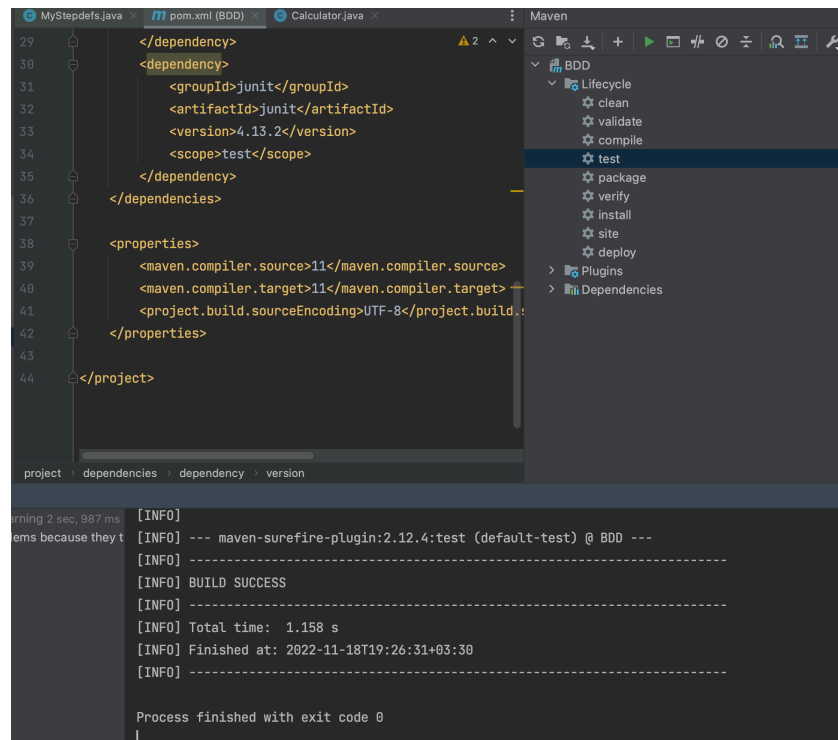
در ابتدا این فایل خطاهایی دارد که برای برطرف کردن‌شان، در مسیر src->main->java->calculator فایل به نام

Calculator را درست می‌کنیم.



شکل ۸: کلاس Calculator

خطوط گفته‌شده را به فایل pom.xml اضافه می‌کنیم (البته با توجه به اینکه ما از ورژن ۱۱ استفاده می‌کنیم، به جای ۱.۸ عدد ۱۱ را در خطوط قرار می‌دهیم). و در نهایت دوباره تست را اجرا می‌کنیم تا به درستی build شود.



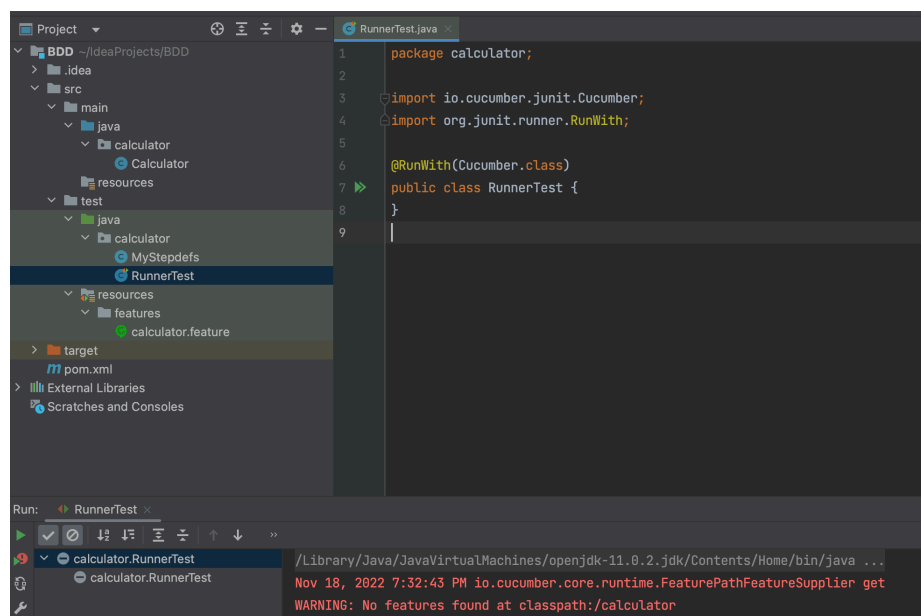
شکل ۹: یک build موفق

۷. گزینه‌ی 'Feature: calculator' Run را می‌زنیم تا سناریو اجرا شود که نتیجه‌ی آن به شکل زیر است:

```
✓ Done: Scenarios 1 of 1 (917 ms) ⚠
/Library/Java/JavaVirtualMachines/openjdk-11.0.2.jdk/Contents/Home/bin/java ...
Testing started at 19:27 ...
3
1 Scenarios (1 passed)
3 Steps (3 passed)
0m0.271s
```

شکل ۱۰: سناریو با موفقیت اجرا می‌شود

۸. برای مشاهده جزئیات اجرا توسط JUnit یک کلاس جاوا به نام RunnerTest در آدرس test->java->calculator درست می‌کنیم و پس از قرار دادن کدها گفته‌شده، آن را اجرا می‌کنیم که به خطا می‌خوریم.



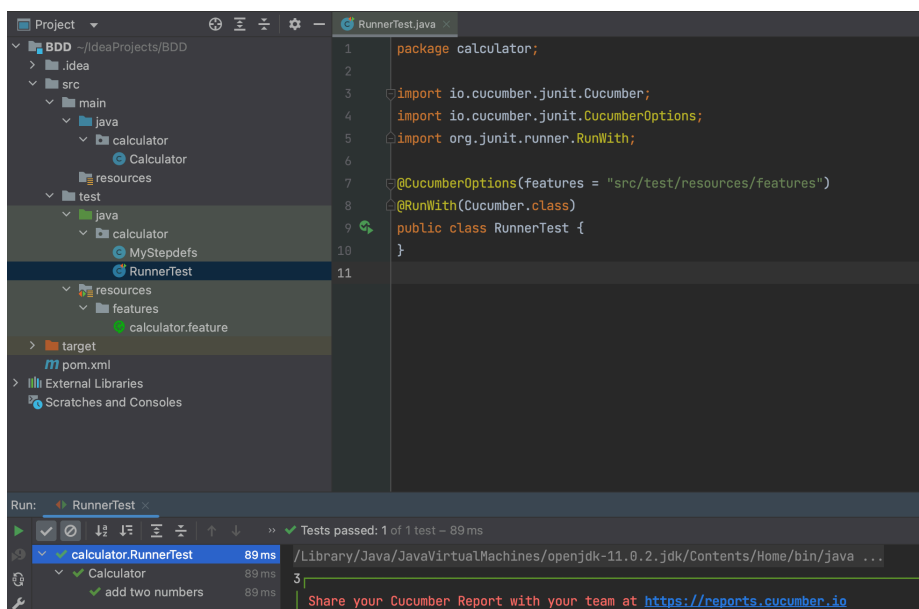
```
Project: BDD -/ideaProjects/BDD
src
  main
    java
      calculator
        Calculator
        resources
        test
          java
            calculator
              MyStepdefs
              RunnerTest
              resources
                features
                  calculator.feature
  target
  pom.xml
  External Libraries
  Scratches and Consoles

RunnerTest.java
1 package calculator;
2
3 import io.cucumber.junit.Cucumber;
4 import org.junit.runner.RunWith;
5
6 @RunWith(Cucumber.class)
7 public class RunnerTest {
8 }
9

Run: RunnerTest
calculator.RunnerTest
/Library/Java/JavaVirtualMachines/openjdk-11.0.2.jdk/Contents/Home/bin/java ...
Nov 18, 2022 7:32:43 PM io.cucumber.core.runtime.FeaturePathFeatureSupplier get
WARNING: No features found at classpath:/calculator
```

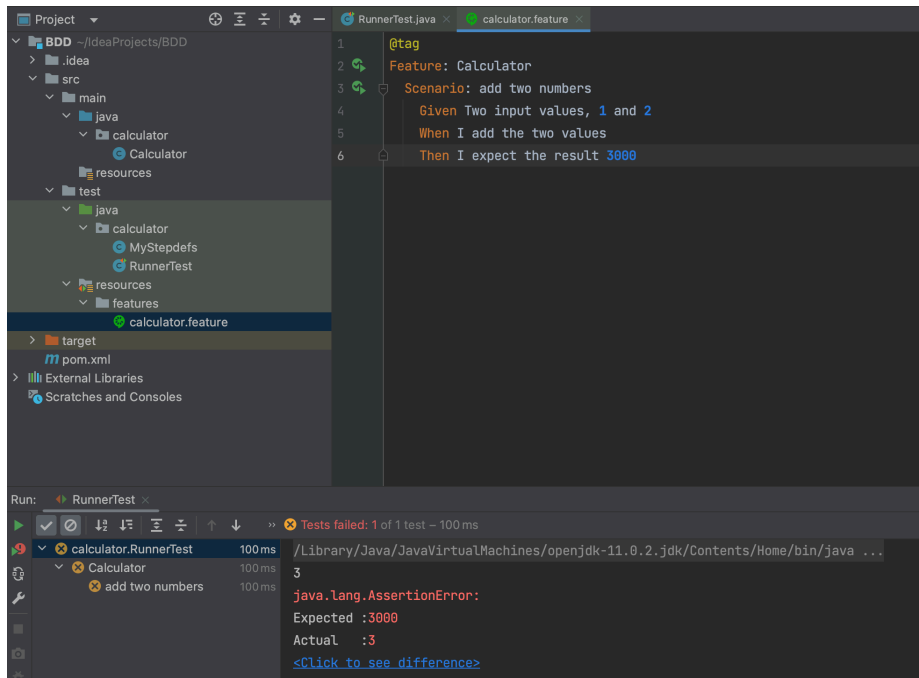
شکل ۱۱: فایل RunnerTest.java

۹. تغییرات گفته شده را می دهیم تا مشکل برطرف شود.



شکل ۱۲: ایجاد تغییرات در RunnerTest.java

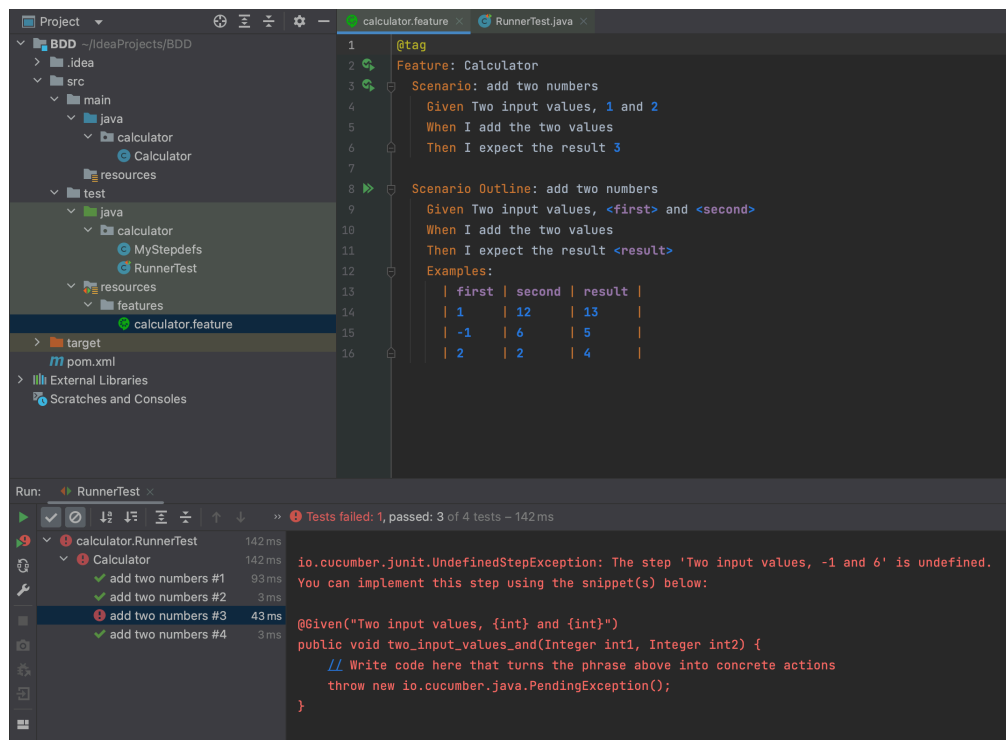
همچنین می بینیم که اگر در سناریو، حاصل جمع را به جای ۳ عدد ۳۰۰۰ قرار بدهیم، تست پاس نمی شود.



شکل ۱۳: یک نمونه از سناریو که شکست می خورد

۲.۱ اجرای scenario outline

سناریو را به شکل زیر در ادامه‌ی فایل calculator.feature تعریف می‌کنیم و با اجرای RunnerTest می‌بینیم که تست سوم (که ورودی‌های آن ۱ - ۶ هستند) پاس نمی‌شوند و به مشکل undefined می‌خورند.



شکل ۱۴: تست سوم پاس نمی‌شود

علت این مشکل این است که برای بخش Given به شکل

@Given("Two input values, (\\d+)and(\\d+)")

نوشته شده است. بنابراین عددهای منفی یا اعدادی که با علامت‌های + و - شروع می‌شوند، با این رجکس میچ نمی‌شوند (که در این تست عدد ۱ - ۶ چهار این مشکل می‌شود و چون علامت - را نمی‌توان با هیچ بخشی از نوشته‌ی بالا میچ کرد، ارور می‌گیریم). پس در همین زمینه باید تغییر ایجاد کنیم. کد جدید به شکل زیر است:


```

9      public class MyStepdefs {
10         private Calculator calculator;
11         private int value1;
12         private int value2;
13         private int result;
14
15         @Before
16         public void before() {
17             calculator = new Calculator();
18         }
19
20         @Given("Two input values, {int} and {int}")
21         public void twoInputValuesAnd(int arg0, int arg1) {
22             value1 = arg0;
23             value2 = arg1;
24         }
25
26         @When("I add the two values")
27         public void iAddTheTwoValues() {
28             result = calculator.add(value1, value2);
29             System.out.print(result);
30         }
31
32         @Then("I expect the result {int}")
33         public void iExpectTheResult(int arg0) {
34             Assert.assertEquals(arg0, result);
35         }
36     }

```

شکل ۱۵: تغییرات لازم در فایل MyStepdefs داده شد

این بار به جای فرمت d+ که فقط تعداد یک یا بیشتر رقم را دربر می‌گیرد، از int استفاده می‌کنیم که تمامی اعداد صحیح را (مثبت و منفی) می‌کند. پس از اجرای دوباره‌ی RunnerTest می‌بینیم که این بار تمامی تست‌ها پاس می‌شوند.

Test Case	Duration
calculator.RunnerTest	196 ms
Calculator	196 ms
add two numbers #1	171 ms
add two numbers #2	12 ms
add two numbers #3	6 ms
add two numbers #4	7 ms

```

/Library/Java/JavaVirtualMachines/openjdk-11.0.2.jdk/Contents/Home/bin/java ...
31354
Share your Cucumber Report with your team at https://reports.cucumber.io
Activate publishing with one of the following:

src/test/resources/cucumber.properties:    cucumber.publish.enabled=true
src/test/resources/junit-platform.properties:  cucumber.publish.enabled=true
Environment variable:      CUCUMBER_PUBLISH_ENABLED=true
JUnit:                      @CucumberOptions(publish = true)

```

شکل ۱۶: تمامی تست‌ها پاس می‌شوند

کدهای این قسمت در پوشه‌ای با نام BDD پیوست شده است.

۲ سناریوی دوم

برای راه‌اندازی اولیه‌ی پروژه مانند بخش قبلی عمل می‌کنیم. سناریوهای گفته‌شده را در فایل calculator.feature می‌نویسیم. هنگام تقسیم دو عدد باید از چند مورد غیر مجاز اطمینان حاصل کنیم:

- تقسیم بر صفر
- تقسیم صفر بر صفر
- جذر گرفتن از عدد منفی

در سناریوهای تست‌سازی علاوه بر تست‌هایی که باید با موفقیت حاصلشان به دست می‌آید تست‌هایی با ویژگی‌های بالا طراحی و آزمایش شدند. در صورتی که عمل غیرمجازی رخ دهد، خروجی برابر با ۱- خواهد بود. سناریوهای ما شامل سه تست عادی، سه تست مربوط به چگونگی برخورد با ورودی برابر با صفر و سه تست در رابطه با برخورد با اعداد منفی ورودی است.

```
@tag
Feature: Calculator

Scenario: square root of division of two numbers
  Given Two input values, 36 and 4
  When I take the square root of the division of the first by the second
  Then I expect the result 3

Scenario Outline: square root of division of two numbers
  Given Two input values, <first> and <second>
  When I take the square root of the division of the first by the second
  Then I expect the result <result>
  Examples:
  | first | second | result |
  | 36    | 4      | 3       |
  | 1.44  | 1      | 1.2     |
  | 144   | 100    | 1.2     |
  | 12    | 0      | -1      |
  | 0     | 0      | -1      |
  | 12    | -1     | -1      |
  | -15   | 1      | -1      |
  | -9    | -1     | 3       |
  | 0     | 1      | 0       |
```

شکل ۱۷: Scenarios in a .feature file

همچنین در stepdefs برای هر دستوری در فایل feature یک تابع تعریف می‌کنیم که پس از خواندن آن دستور اجرا شود و شامل گرفتن اعداد ورودی، انجام عملیات تقسیم و جذرگیری و مقایسه‌ی حاصل با آن نتیجه‌ای که در تست مشخص شده بود، می‌شود. تصاویر توابع مختلف این فایل در ذیل قرار دارد و همچنین می‌بینیم برخی قسمت‌های توابع به آزمایش ورودی‌ها برای تشخیص عملیات‌های غیرمجاز می‌پردازد.

```
@Before
public void before() { calculator = new Calculator();}
```

شکل ۱۸: انتصاب یک محاسبه‌گر در ابتدای کار برای انجام محاسبات دستورات بعدی

```
@Given("Two input values, {float} and {float}")
public void inputValues(float arg0, float arg1) {

    value1 = arg0;
    value2 = arg1;
}
```

شکل ۱۹: گرفتن اعداد ورودی

```
@When("I take the square root of the division of the first by the second")
public void divideAndSquareRoot() {
    if(value1 == 0 && value2 == 0){
        result = -1;
    }
    else if(value2 == 0){
        result = -1;
    }else if(value1*value2 < 0){
        result = -1;
    }else{
        result = calculator.squareRoot(calculator.divide(value1, value2));
    }
}
```

شکل ۲۰: بررسی برای عملیات‌های غیرمجاز و انجام عملیات‌های ریاضی مدنظر فرمول

```
@Then("I expect the result {double}")
public void expectResult(double arg0) { Assert.assertEquals((long)arg0, result);}
```

شکل ۲۱: بررسی نتیجه‌ی بدست آمده با خروجی تست

همچنین کلاس calculator شامل دو تابع divide برای محاسبه‌ی حاصل تقسیم دو عدد و تابع squareRoot برای محاسبه‌ی رادیکال دو عدد است:

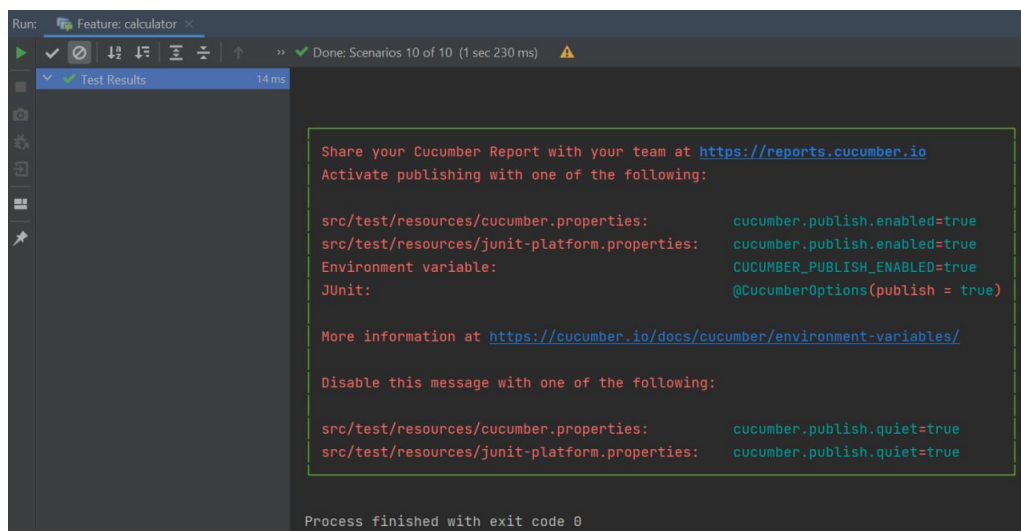
```
package calculator;
import java.lang.Math;

public class Calculator {
    public float divide(float a, float b){
        return a / b;
    }

    public long squareRoot(float a){
        return (long) Math.sqrt(a);
    }
}
```

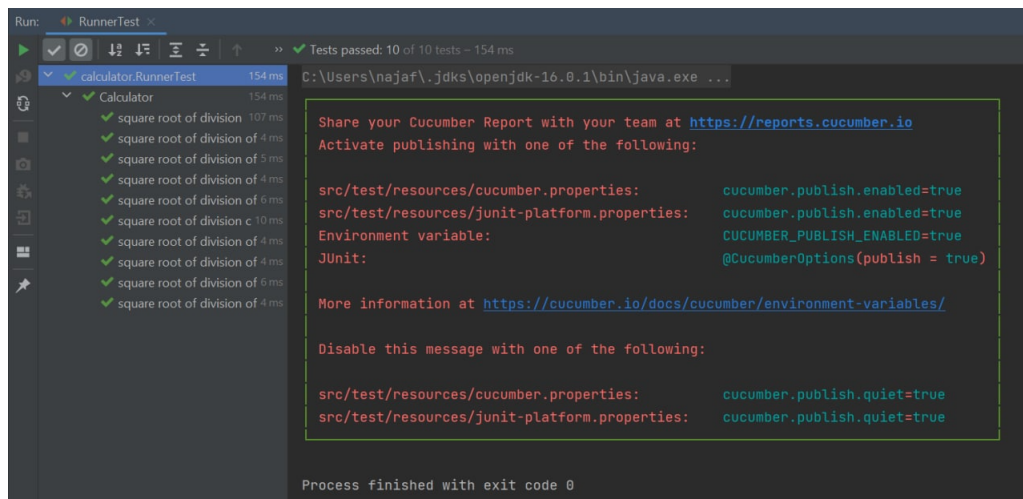
شکل ۲۲: کلاس calculator

پس از ران گرفتن مستقیم از calculator.feature نتیجه‌ی زیر بدست آمد:



شکل ۲۳: نتیجه‌ی اجرای تست‌ها (مستقیم)

برای دیدن نتیجه‌ی تست‌ها RunnerTest را اجرا کردیم که همه‌ی تست‌ها پاس شدند.



شکل ۲۴: نتیجه‌ی اجرای تست‌ها با استفاده از RunnerTest

فایل کد مربوط به این بخش در پوشه‌ی BDD_3_3 قرار گرفته‌است.
در ریپازیتوری زیر هم تمامی کدها و گزارش کار آزمایش قرار داده‌شده‌است.
<https://github.com/Miraneh/SoftwareLab>