CSC 4104

Information System - BDD - Gherkin and Cucumber II

Dr J Paul Gibson

Dept. INF

Office D311

paul.gibson@telecom-sudparis.eu

http://jpaulgibson.synology.me/~jpaulgibson/TSP/Teaching/CSC4104/CSC4104-BDD-GherkinCucumber.pdf

A complete (pedagogic) example project - a simple calculator

CSC 4104

Context/Motivation/Narrative:

I am not good at mental arithmetic

So I need a terminal/command-line calculator assistant

It will help me add 2 numbers

I should check all combinations of positive, negative and zero.

A complete (pedagogic) example project - a simple calculator

CSC 4104

“calculator.feature”

```
# @domain is the "domain" tag for this subset of scenarios
@domain
Feature: Calculator


Scenario Outline: Add two numbers

Given first number is <a>
And second number is <b>
When two numbers are added
Then result should be <x>

# The examples do not cover the 'zero' cases referred to in the context.
# @TODO - add the zero cases to our table of Examples
Examples:
|  a |  b |  x  |
| 1  |  2 |  3  |
| -2 | -1 | -3  |
| 3  | -2 |  1  |
| -3 | 1  | -2  |
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

“CalculatorSteps.java”

```java
package gibson.jpaul;

import io.cucumber.java.en.Given;
import io.cucumber.java.en.Then;
import io.cucumber.java.en.When;
import static org.junit.Assert.*;

public class CalculatorSteps {

  private final Calculator calc = new Calculator();
  private int actual;

  @Given("first number is {int}")
  public void givenTheFirstNumberIs(int a) {
    calc.setA(a);}

  @Given("second number is {int}")
  public void givenTheSecondNumberIs(int b) {
    calc.setB(b);}

  @When("two numbers are added")
  public void whenTheTwoNumbersAreAdded() {
    actual = calc.getSum();}

  @Then("result should be {int}")
  public void thenTheResultShouldBe(int x) {
    assertEquals(x, actual);}
}
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

“Calculator.java”

```
1  package gibson.jpaul;
2
4⊕  * Binary operator integer calculator with 2 registers (a and b) and
7  public class Calculator {
8
10⊕     * register for operand1
12     private int a;
13
15⊕     * register for operand2
17     private int b;
18
20⊕     * Change the value of register1
23⊖     public void setA(int a) {
24         this.a = a;
25     }
26
28⊕     * Change the value of register2
31⊖     public void setB(int b) {
32         this.b = b;
33     }
34
36⊕     * The Addition
39⊖     public int getSum() {
40         return a + b;
41     }
42 }
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

“RunCucumberTest.java”

```
package gibson.jpaul;

import io.cucumber.junit.Cucumber;
import io.cucumber.junit.CucumberOptions;
import org.junit.runner.RunWith;

@RunWith(Cucumber.class)
@CucumberOptions(plugin = {"pretty"})
public class RunCucumberTest {

}
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

“RunCucumberTest.java” run as JUnit test

```
@domain
Scenario Outline: Add two numbers # gibson/jpaul/calculator.feature:17
  Given first number is 1         # gibson.jpaul.CalculatorSteps.givenTheFirstNumberIs(int)
  And second number is 2          # gibson.jpaul.CalculatorSteps.givenTheSecondNumberIs(int)
  When two numbers are added      # gibson.jpaul.CalculatorSteps.whenTheTwoNumbersAreAdded()
  Then result should be 3         # gibson.jpaul.CalculatorSteps.thenTheResultShouldBe(int)

@domain
Scenario Outline: Add two numbers # gibson/jpaul/calculator.feature:18
  Given first number is -2        # gibson.jpaul.CalculatorSteps.givenTheFirstNumberIs(int)
  And second number is -1         # gibson.jpaul.CalculatorSteps.givenTheSecondNumberIs(int)
  When two numbers are added      # gibson.jpaul.CalculatorSteps.whenTheTwoNumbersAreAdded()
  Then result should be -3        # gibson.jpaul.CalculatorSteps.thenTheResultShouldBe(int)

@domain
Scenario Outline: Add two numbers # gibson/jpaul/calculator.feature:19
  Given first number is 3         # gibson.jpaul.CalculatorSteps.givenTheFirstNumberIs(int)
  And second number is -2         # gibson.jpaul.CalculatorSteps.givenTheSecondNumberIs(int)
  When two numbers are added      # gibson.jpaul.CalculatorSteps.whenTheTwoNumbersAreAdded()
  Then result should be 1         # gibson.jpaul.CalculatorSteps.thenTheResultShouldBe(int)

@domain
Scenario Outline: Add two numbers # gibson/jpaul/calculator.feature:20
  Given first number is -3        # gibson.jpaul.CalculatorSteps.givenTheFirstNumberIs(int)
  And second number is 1          # gibson.jpaul.CalculatorSteps.givenTheSecondNumberIs(int)
  When two numbers are added      # gibson.jpaul.CalculatorSteps.whenTheTwoNumbersAreAdded()
  Then result should be -2        # gibson.jpaul.CalculatorSteps.thenTheResultShouldBe(int)
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

But, have we really tested the interface between the *system* and the user?

“calculatorSystem.feature”

```
# @system is the "system" tag for this subset of scenarios
@system
Feature: Calculator System


Narrative:

I am not good at mental arithmetic
So I need a calculator assistant
It will help me add 2 numbers
I should check all combinations of positive, negative and zero.

# The scenarios do not cover the 'zero' cases referred to in the context.
# @TODO - add the zero cases to our table of Examples
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

“calculatorSystem.feature”

```
Scenario: Run program with input 1 and 2 (pos,pos)

Given input is
"""
1
2
"""
When program runs
Then output should be
"""
Sum of 1 and 2 equals 3

"""

Scenario: Run program with input -2 and -1 (neg,neg)

Given input is
"""
-2
-1
"""
When program runs
Then output should be
"""
Sum of -2 and -1 equals -3

"""
```

```
Scenario: Run program with input 3 and -2 (pos,neg)

Given input is
"""
3
-2
"""
When program runs
Then output should be
"""
Sum of 3 and -2 equals 1

"""

Scenario: Run program with input -3 and 1 (neg,pos)

Given input is
"""
-3
1
"""
When program runs
Then output should be
"""
Sum of -3 and 1 equals -2

"""
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

"CalculatorSystemSteps.java"

```
public class CalculatorSystemSteps {

    private String input;
    private String actual;

    @Given("input is")
    public void input_is(String input) {
        this.input = input;
    }

  @When("program runs")
  public void program_runs() throws IOException {

  ...            }


 @Then("output should be")
 public void output_should_be(String expected) {
     assertEquals(expected, actual);
 }
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

“CalculatorSystemSteps.java”

```java
@When("program runs")
public void program_runs() throws IOException {

    InputStream inputStream =
        new ByteArrayInputStream(input.getBytes(StandardCharsets.UTF_8));

    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    PrintStream outputStream = new PrintStream(byteArrayOutputStream);

    PrintStream previousOut = System.out;
    InputStream previousIn = System.in;

    System.setIn(inputStream);
    System.setOut(outputStream);

    TerminalAddition.main(null);

    actual = byteArrayOutputStream.toString();

    inputStream.close();
    outputStream.close();

    System.setOut(previousOut);
    System.setIn(previousIn);
}
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

```
@system
Scenario: Run program with input 1 and 2 (pos,pos) # gibson/jpaul/calculatorSystem.feature:16
  Given input is                                   # gibson.jpaul.CalculatorSystemSteps.input_is(java.lang.String)
  When program runs                                # gibson.jpaul.CalculatorSystemSteps.program_runs()
  Then output should be                            # gibson.jpaul.CalculatorSystemSteps.output_should_be(java.lang.String)

@system
Scenario: Run program with input -2 and -1 (neg,neg) # gibson/jpaul/calculatorSystem.feature:30
  Given input is                                     # gibson.jpaul.CalculatorSystemSteps.input_is(java.lang.String)
  When program runs                                  # gibson.jpaul.CalculatorSystemSteps.program_runs()
  Then output should be                              # gibson.jpaul.CalculatorSystemSteps.output_should_be(java.lang.String)

@system
Scenario: Run program with input 3 and -2 (pos,neg) # gibson/jpaul/calculatorSystem.feature:44
  Given input is                                    # gibson.jpaul.CalculatorSystemSteps.input_is(java.lang.String)
  When program runs                                 # gibson.jpaul.CalculatorSystemSteps.program_runs()
  Then output should be                             # gibson.jpaul.CalculatorSystemSteps.output_should_be(java.lang.String)

@system
Scenario: Run program with input -3 and 1 (neg,pos) # gibson/jpaul/calculatorSystem.feature:58
  Given input is                                    # gibson.jpaul.CalculatorSystemSteps.input_is(java.lang.String)
  When program runs                                 # gibson.jpaul.CalculatorSystemSteps.program_runs()
  Then output should be                             # gibson.jpaul.CalculatorSystemSteps.output_should_be(java.lang.String)
```

A complete (pedagogic) example project - a simple calculator

CSC 4104

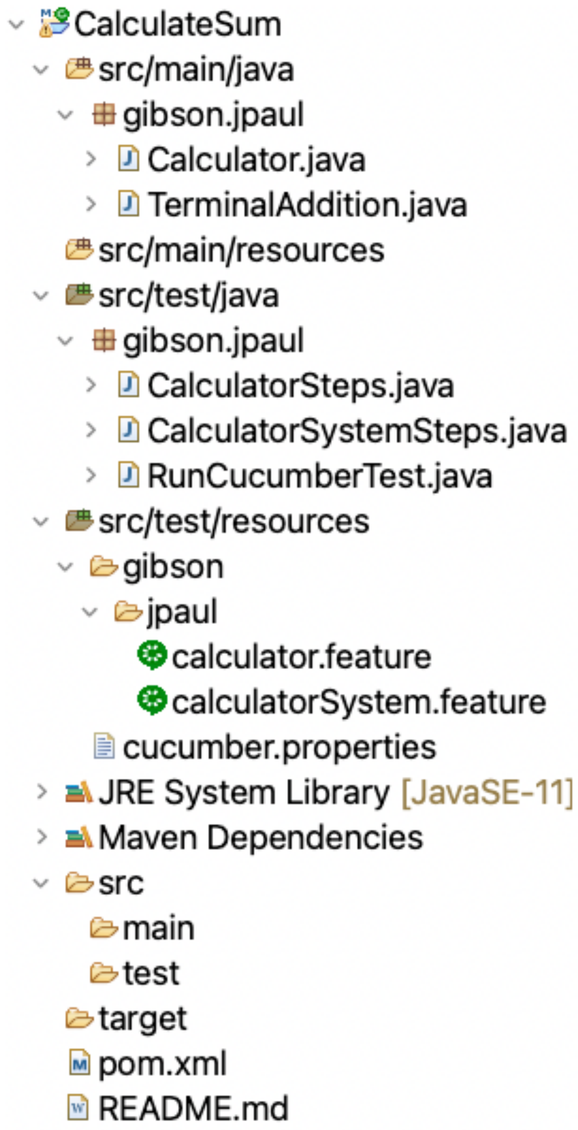
Runs: 8/8 Errors: 0 Failures: 0

- gibson.jpaul.RunCucumberTest [Runner: JUnit 4] (0.113 s)
  - Calculator (0.092 s)
    - Add two numbers #1 (0.084 s)
    - Add two numbers #2 (0.003 s)
    - Add two numbers #3 (0.003 s)
    - Add two numbers #4 (0.002 s)
  - Calculator System (0.020 s)
    - Run program with input 1 and 2 (pos,pos) (0.011 s)
    - Run program with input -2 and -1 (neg,neg) (0.003 s)
    - Run program with input 3 and -2 (pos,neg) (0.003 s)
    - Run program with input -3 and 1 (neg,pos) (0.003 s)

A complete (pedagogic) example project - a simple calculator

CSC 4104

Putting it all together in a **Maven Java project (download from web site)**



Note: you can/should configure the project by **converting it to a cucumber project**

Quiz: what do you think **cucumber.properties** is used for?

A complete (pedagogic) example project - a simple calculator

CSC 4104

TODO

- Complete the feature-tests (domain and system) for zero cases
- Complete the feature-tests (domain and system) for out of bound cases
- Add a new use case for multiplication

A complete (pedagogic) example project - a simple calculator