



Scuola Politecnica e delle Scienze di Base  
Corso di Laurea Specialistica in Ingegneria Informatica

Elaborato in **Software Testing**

**Framework Cucumber e linguaggio Gherkin a supporto delle pratiche agile**

**Docente:**

Porfirio Tramontana

**Studenti:**

Anna Lamboglia

M63/1219

Francescantonio Pisano

M63/1069

## Sommario

Introduzione .....	4
Capitolo 1: <i>Behavior-driven development vs Test Driven Development</i> .....	5
1.1 Test Driven Development (TDD) .....	5
1.1.1 Vantaggi del TDD .....	6
1.2 Behavior-Driven-Development (BDD) .....	7
1.2.1 Vantaggi del BDD .....	8
1.3 Confronto tra TDD & BDD .....	9
Capitolo 2: <i>Esempi di integrazione di Cucumber e Gherkin a supporto delle metodologie Agile</i> .....	10
2.1 Esempio 1: Ciclo di sviluppo BDD per una calcolatrice .....	10
2.1.1 Passo 1: Produzione degli Scenari con l'utilizzo di Gherkin .....	10
2.1.2 Passo 2: Implementazione del codice .....	16
2.1.3 Passo 3: Installazione di Cucumber e creazione del progetto .....	17
2.1.4 Passo 4: Generazione degli Steps .....	24
2.1.5 Passo 5: Implementazione degli Steps .....	27
2.1.6 Passo 6: Esecuzione dei test e valutazione .....	32
2.1.7 Passo 7: Esecuzione dei test con il Runner Junit (facoltativo) .....	36
2.2 Esempio 2: Integrazione di Selenium .....	39
2.2.1 Passo 1: Produzione degli Scenari .....	39
2.2.2 Passo 2: Implementazione del codice .....	40
2.2.3 Passo 3: Creazione del progetto .....	41
2.2.4 Passo 4: Generazione degli Steps .....	43
2.2.5 Passo 5: Implementazione degli Steps .....	43
2.2.6 Passo 6: Esecuzione dei test e valutazione .....	49
2.3 Esempio 3: Cucumber in Android Studio .....	49
2.3.1 Passo 1: Produzione degli Scenari .....	49
2.3.2 Passo 2: Implementazione del codice .....	51
2.3.3 Passo 3: Creazione del progetto .....	51
2.3.4 Passo 4: Generazione degli Steps .....	53
2.3.5 Passo 5: Implementazione degli Steps .....	54
2.3.6 Passo 6: Esecuzione dei test e valutazione .....	57
2.4 Tabella Gherkin .....	60
2.5 Cucumber: il Parser Gherkin .....	63

2.5.1 Architettura del Gherkin-parser .....	63
2.5.2 Abstract Syntax Tree (AST) .....	64
2.5.3 Pickles .....	65
Capitolo 3: <i>Cucumber Studio</i> .....	69
3.1 Creazione di un nuovo progetto.....	70
3.2 BDD con Cucumber.....	70
3.3 Test management.....	72
3.4 Creazione di Features in CucumberStudio .....	73
3.4 Esempio Coffee Machine.....	76
Conclusioni .....	80
Bibliografia .....	81

# Introduzione

Nel seguente elaborato, è stato affrontato il problema della descrizione dei casi di test, mostrando come le ambiguità dettate dal linguaggio possano condurre ad errori di incomprensione nelle varie fasi del ciclo di sviluppo del software.

Al fine di introdurre tale tematica, sono state brevemente esposte ed analizzate due tra le più diffuse metodologie di sviluppo in questo settore, ossia il *Test Driven Development* (TDD) ed il *Behavior Driven Development* (BDD).

Concentrando gli sforzi sul BDD, è stato visto come tale metodologia consenta di rendere più semplice la comunicazione tra le varie parti di un team di sviluppo software, andando così a ridurre tempi e costi dovuti ad incomprensioni all'interno del team.

Successivamente, è stato analizzato Cucumber, ossia uno dei principali framework a supporto del BDD. Esso permette di specificare in modo semplice i comportamenti software previsti dal cliente e dal *product owner*, grazie alla descrizione delle *user story* realizzate tramite il linguaggio Gherkin. Inoltre, sono stati illustrati vari esempi applicativi di Cucumber in linguaggio Java, realizzati tramite Eclipse IDE ed Android Studio.

Oltre a ciò, è stata analizzata la piattaforma *CucumberStudio* della *SmartBear*, fornendo un quadro completo delle funzionalità offerte e descrivendo un esempio applicativo.

# Capitolo 1: Behavior-driven development vs Test Driven Development

Nell'ambito dello sviluppo software si parla sempre più spesso di metodologia agile (*agile software development*) e di tecniche di sviluppo agile, indicando in tal modo i metodi di sviluppo software che si concentrano sul consegnare al cliente in tempi brevi un prodotto funzionante, che verrà successivamente ampliato e modificato attraverso un procedimento di integrazione continua.

Tra le tecniche di sviluppo agile si analizzeranno nei paragrafi seguenti il *Test Driven Development* (TDD) ed il *Behavior-Driven-Development* (BDD).

## 1.1 Test Driven Development (TDD)

In ingegneria del software, il Test Driven Development, in italiano sviluppo guidato dai test o sviluppo guidato dalle verifiche, è un modello di sviluppo del software che consiste nella scrittura dei test prima dell'implementazione vera e propria della funzionalità. [1] Tale approccio, come evidenzia Robert C. Martin nella sua documentazione [2][3], è basato su tre punti chiave che danno vita alle tre fasi principali *Red-Green-Refactor*:

1. Scrivere un test che fallisce prima ancora di scrivere qualunque codice applicativo (che andrà in produzione). (Fase: **Red**);
2. Non scrivere più di un test (o più di un'asserzione in esso) di quanto sia sufficiente per farlo fallire, o per far fallire la compilazione. (Fase: **Green**);
3. Non scrivere più codice applicativo di quanto sia necessario per far passare il test costruito al punto precedente. (Fase: **Refactor**).

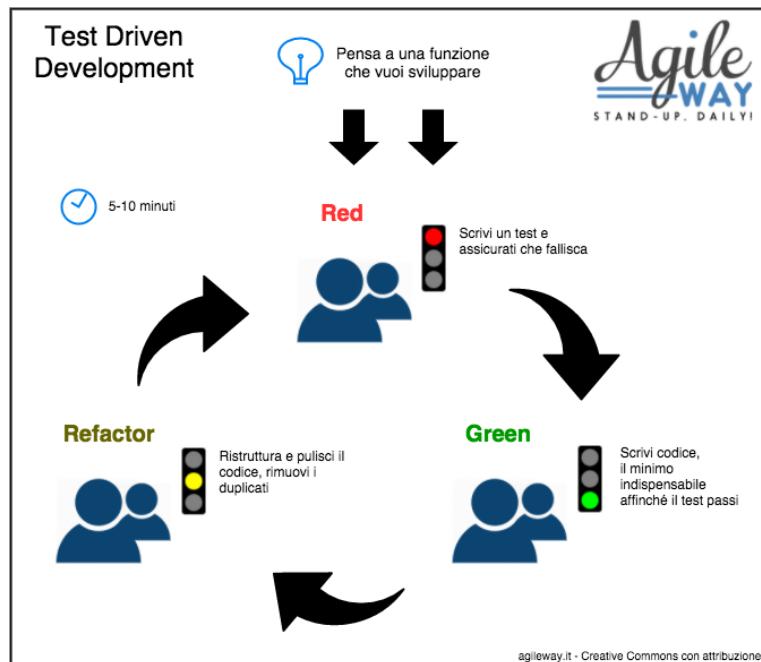


Figura 1: Illustrazione del ciclo di sviluppo TDD [4]

Andando ad esplodere le singole fasi, è possibile fornire un automa a stati finiti che specifica tutti i casi possibili per ogni fase.

Il primo step, *Rewrite the test*, è relativo alla scrittura di un test minimo che fallisce (cerchio blu, in alto a sinistra #1).

Nel caso in cui il test passi (cerchio rosso, in basso a sinistra #2), è necessario che esso venga riscritto in quanto, probabilmente, presenta delle problematiche e va modificato.

Se, invece, il test fallisce, è possibile scrivere il codice applicativo fino a quando quest'ultimo non passi correttamente il test (cerchio blu, a sinistra #3).

Il codice così ottenuto in questa fase potrebbe essere poco elegante e con delle imperfezioni, ma l'obiettivo, per adesso, è far sì che il test risulti superato.

Nella successiva fase di Refactor (cerchio sulla destra in fig.2), avviene la valutazione del test scritto finora, o di tutti i test scritti nel caso iterativo. Se tale test fallisce sono necessarie ulteriori modifiche affinché sia possibile superare il test; in caso contrario, e quindi se il test è superato, si può procedere nell'apportare migliorie al codice applicativo.

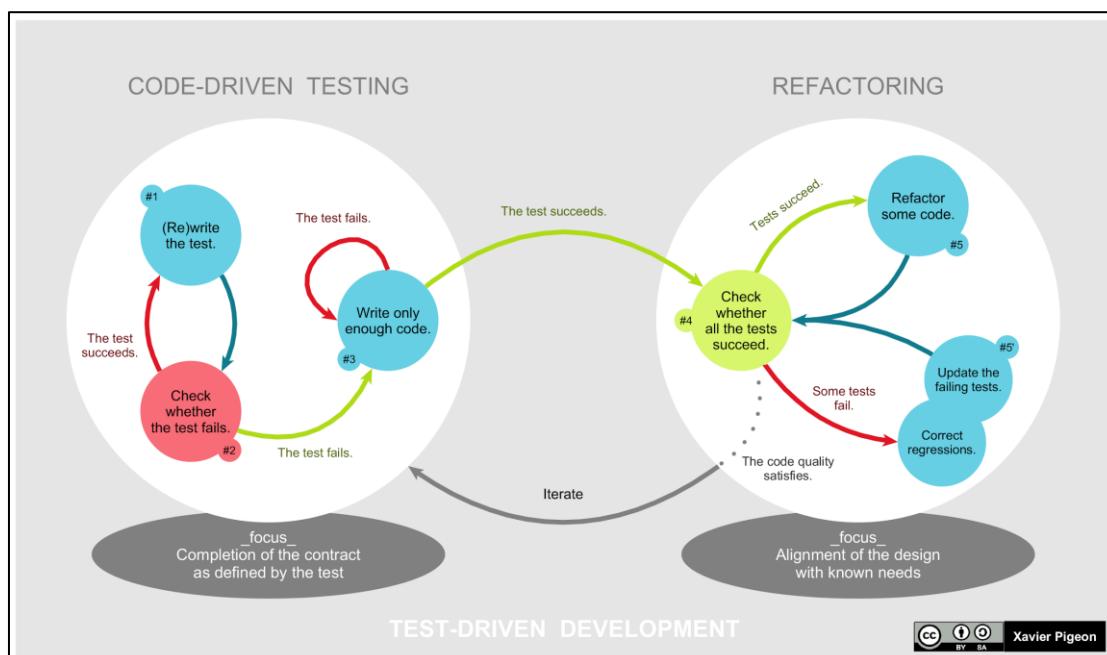


Figura 2: Illustrazione del ciclo di sviluppo BDD [2]

### 1.1.1 Vantaggi del TDD

Il vantaggio principale è che questo approccio porta il programmatore a scrivere “obbligatoriamente” i casi di test per tutte le parti di codice applicativo implementate, garantendo in questo modo maggior copertura del codice del software prodotto.

Si applica il principio della “*line-by-line granularity*” (scrivere il codice step by step), permettendo di avere maggior affidabilità del codice scritto e maggiore facilità nella ricerca dei difetti che provocano fallimenti.

Un ulteriore vantaggio è la possibilità di poter rendere l'esecuzione dei test automatizzata tramite framework della serie *xUnit*.

## 1.2 Behavior-Driven-Development (BDD)

Il *Behavior-Driven-Development* (BDD) in ingegneria del software è un processo di sviluppo che mette al centro i comportamenti delle funzionalità. La paternità di tale approccio si attribuisce a Daniel Terhorst-North (abbreviato in Dan North), che nel 2006 espose nel suo articolo “Introducing BDD” come fosse nata la necessità di un approccio che andasse a supportare i programmati ed i tester che iniziavano ad interfacciarsi con le neonate pratiche di TDD [5].

Egli, infatti, colse nella descrizione del comportamento (*behavior*) di un modulo il punto chiave di tale approccio. Si stima, infatti, che l’85% dei difetti nel software sviluppato sia originato da requisiti software ambigui, incompleti ed illusori, in quanto le specifiche del linguaggio naturale possono essere fonte di confusione e, allo stesso tempo, difficili da comprendere [6].

In particolare, Dan North intuì che pensare al comportamento del software piuttosto che al codice, avrebbe certamente aiutato a suddividere il prodotto che si stava sviluppando in unità semplici da testare.

Per questo, North ridefinì le storie utente in modo da esporre i criteri di accettazione tramite l’uso di scenari che avessero una forma semplice ed immediata. Tale approccio trova riscontro nel fatto che alcuni esperti ritengono che i test di accettazione<sup>1</sup> forniscano informazioni più precise ed accurate rispetto alle descrizioni in linguaggio naturale.

Fu quindi proposto, per la descrizione degli scenari di test, il seguente modello la cui analisi più dettagliata sarà fornita nei paragrafi successivi:

**Given** marca il contesto fornendo le precondizioni per lo scenario,  
**When** descrive le azioni che sono eseguite,  
**Then** descrive i risultati attesi.

Ad oggi, la combinazione tra BDD e TDD è sempre più comune nell’ambito del software development, al fine di ottimizzare la comunicazione tra le parti di un team mediante l’utilizzo delle seguenti strategie di BDD:

1. Applicare il principio “Five Why’s” ad ognuna delle storie utente in modo da rendere l’obiettivo di business più chiaro;
2. Implementare solo quei comportamenti che forniscono risultati utili in termini di business;
3. Descrivere i comportamenti in una notazione che sia diretta e accessibile sia per esperti di dominio che dai tester e sviluppatori, in modo da migliorare la comunicazione;
4. Applicare le tecniche sopra indicate sin dal livello di astrazione del software più basso, ponendo particolare attenzione alla distribuzione delle responsabilità tra le differenti classi.

---

<sup>1</sup> Il test di accettazione è un “*test formale condotto per determinare se un sistema soddisfa o meno i suoi criteri di accettazione e per consentire al cliente di determinare se accettare o meno il sistema*” [7].

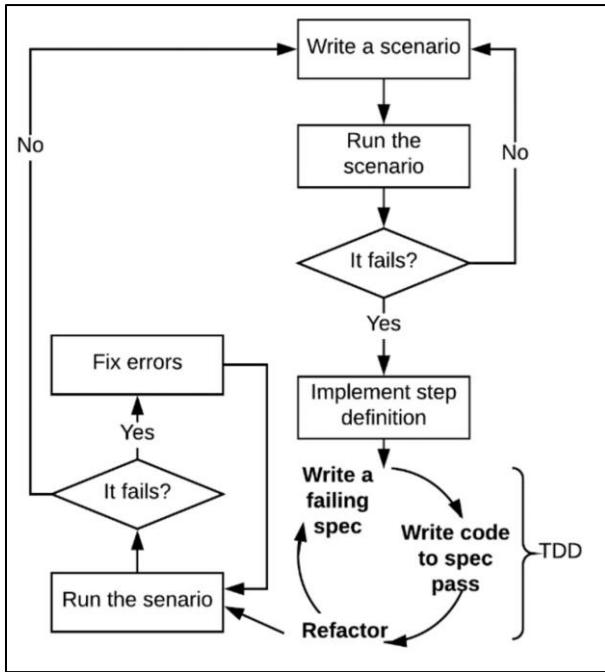


Figura 3: TDD e BDD insieme [8]

L'utilizzo delle precedenti strategie di BDD permette ad un team di sviluppo di fornire una documentazione funzionale sotto forma di storie utente, che possono essere espanso tramite esempi e scenari eseguibili, in cui l'esperto di business descrive i comportamenti del sistema rispondendo alle domande di tester e sviluppatori, attraverso i seguenti step:

1. Selezione della specifica;
2. Scrittura dello scenario relativo alla specifica scelta;
3. Scrittura del codice che testerà lo scenario, e che dovrà superare i test;
4. Superamento della specifica;
5. Refactoring.

Così facendo, viene maturata una conoscenza approfondita delle funzionalità del sistema e si analizzano anche comportamenti aggiuntivi che, inizialmente, non erano stati considerati e che possono essere integrati.

### 1.2.1 Vantaggi del BDD

Come già anticipato nel precedente paragrafo, i vantaggi del BDD possono essere riassunti con i seguenti punti:

- Riduzione dei costi grazie ad un maggior *focus* sulle richieste di business attese dal prodotto finale;
- Riduzione dei costi dovuti ad errori e difetti prodotti ad ogni iterazione del ciclo di sviluppo, in quanto ogni membro del team è allineato in anticipo sui comportamenti attesi dal sistema;

- I requisiti sono esplicati e sviluppati velocemente grazie all'elevata comunicazione tra P.O., testers e developers.

### 1.3 Confronto tra TDD & BDD

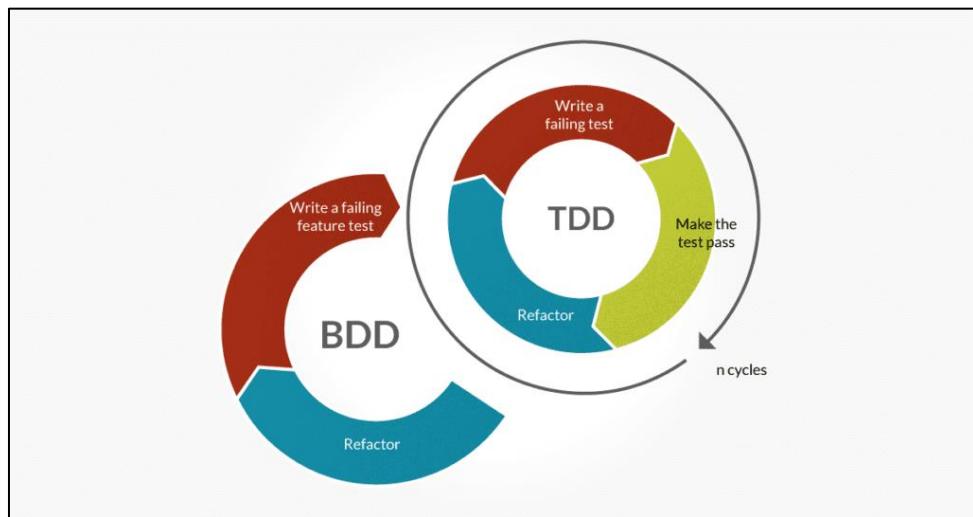
Di seguito, si propone uno schema riassuntivo di sintesi delle principali differenze tra i due approcci:

	<b>TDD</b>	<b>BDD</b>
<b>Obiettivi</b>	Produzione del codice applicativo a partire dai casi di test	Produzione di scenari di test comprensibili per cliente, sviluppatore e tester
<b>Approccio</b>	Tecnico, specializzato	Comune, facilmente comprensibile
<b>Codice applicativo</b>	Specifico, creato per superare i test	Semplice, creato per essere comprensibile

**Tabella 1: Confronto tra TDD e BDD**

Sulla base dello stesso, si evince che nessuna delle due tecniche prevale sull'altra, ma esse sono complementari. Pertanto, si sente sempre più parlare del “binomio” BDD-TDD come pratica di software development.

Si parte dal BDD descrivendo le funzionalità del sistema ed analizzando con il cliente tutti i comportamenti richiesti dallo stesso, evitando così la produzione di codice superfluo. Successivamente, si passa allo sviluppo del codice usando il TDD, in modo da ottenere codice pulito, facilmente mantenibile e di alta qualità.



**Figura 4: Approccio TDD e BDD combinato [9]**

# **Capitolo 2: Esempi di integrazione di Cucumber e Gherkin a supporto delle metodologie Agile**

Nel precedente capitolo sono state mostrate e discusse due metodologie agili quali BDD e TDD, ad oggi sempre più utilizzate nel ciclo di sviluppo di un software.

La dissertazione proseguirà mostrando un esempio di sviluppo che segue la metodologia BDD. Nel mostrare ed applicare tale metodologia si andranno ad utilizzare delle tecniche, dei tool e dei frameworks, che negli anni sono stati sviluppati a supporto di tale tipologia di sviluppo.

Inizialmente, si andranno a definire i problemi da risolvere e da quali requisiti far partire il progetto di sviluppo del software.

## **2.1 Esempio 1: Ciclo di sviluppo BDD per una calcolatrice**

In questa sezione, si andrà a fornire un esempio di ciclo di sviluppo, partendo dal primo step in cui c’è la discussione con il cliente, fino ad arrivare al testing.

In particolare, si mostrerà come applicare la metodologia BDD ad un semplice esempio, dalla definizione degli scenari al testing di una semplice calcolatrice che effettui le quattro operazioni aritmetiche fondamentali.

### **2.1.1 Passo 1: Produzione degli Scenari con l’utilizzo di Gherkin**

Nel primo passo, si andrà ad interagire con il committente/cliente del software, in modo da comprendere nel dettaglio di che prodotto necessita. In particolare, mentre nei cicli di sviluppo tradizionali in questa fase si ricavano i requisiti da rispettare per la verifica della funzionalità del software e vengono scritte le opportune documentazioni, nei cicli di sviluppo agile si pone anche l’obiettivo di stabilire i test di accettazione con il cliente, in modo da renderli meno ambigui e da facilitare la verifica e la futura validazione.

All’interno della metodologia BDD, ciò avviene tramite una riunione denominata “*three amigos*”.

Nel three amigos si analizzano preliminarmente una serie di prospettive per definire l’incremento del lavoro. Questo particolare tipo di meeting si tiene prima, durante e dopo lo sviluppo e prevede la presenza di sole tre prospettive, portate in conto da altrettante persone: *Business, Development, Testing*. La discussione vede come protagonisti:

- **Business Analyst o Product Owner:** rappresenta il Business e porta all’attenzione di sviluppatori e tester quali requisiti devono essere sviluppati e qual è il problema da risolvere. Spesso si occupa della traduzione delle User Stories in una serie di Features [10], sottponendo delle User Stories all’attenzione di developer e tester, portando in gioco il quesito: “*Che problema si sta cercando di risolvere?*” [11]

- **Developer:** rappresentante dell'ottica di development che fornisce la soluzione al problema che deve essere implementato, riuscendo ad introdurre nuovi dettagli da tenere in conto nella realizzazione di una User Story. Egli risponde alla domanda: *“Come si può costruire una soluzione per risolvere un dato problema?”*.
- **Tester:** garantisce una “*Quality Assurance*” sul lavoro svolto, ricercando eventuali User Stories non “soddisfatte” da ciò che è stato prodotto. Il tester porta all’attenzione delle altre due figure eventuali “*edge-cases*” che possono inficiare il funzionamento del prodotto e portare alla manifestazione di difetti. Egli porta in conto nella discussione la domanda: *“Dato questo prodotto, cosa succede se inserisco particolari valori o avviene questo evento?”*.

Dunque, tramite il Three Amigos, si va a definire qual è il lavoro da fare ed in che caso sarà effettuato in modo corretto. Permette, inoltre, di definire per bene e per ogni incremento del lavoro quali devono essere gli obiettivi da realizzare. Tale collaborazione porta anche ad una descrizione chiara e comprensibile del problema, tenendo conto di tre prospettive diverse che devono soddisfare una serie di problematiche e necessità differenti e lontane tra loro.

I benefici nell’utilizzare questa pratica sono:

- Costruzione di una comprensione comune del problema;
- Identificazione preliminare di eventuali incomprensioni;
- Definizione di linee guida chiare per il lavoro che deve essere realizzato, in un dato incremento o nello sviluppo generale.

Ovviamente, al di là dei benefici vi sono anche una serie di accortezze da tenere in considerazione, per non imbattersi in errori comuni:

- Evitare di limitare il tutto a sole tre sole persone, ma includere nella discussione anche gli stakeholders, che sono rilevanti per lo sviluppo di una data funzionalità o di un dato prodotto;
- Espandere questa discussione ad altre prospettive oltre a quella di business, quali il development ed il testing. Ciò garantisce, oltre alla visione da una prospettiva diversa, un prodotto migliore.



Figura 5: I “Three Amigos” nel BDD [12]

Durante la discussione, saranno concordati gli scenari da utilizzare in fase di testing, insieme al cliente ed allo sviluppatore, in uno specifico linguaggio denominato Gherkin. Gherkin [13] è un *Domain Specific Language* che permette di descrivere gli scenari di comportamento, senza specificare dettagli di implementazione. Esso è utilizzato per scrivere le specifiche di Cucumber e framework BDD simili, come ad esempio *SpecFlow*, in modo da condividere la descrizione di scenari di uso all'interno di team composti da anche da professionisti non tecnici informatici [14].

Gli scenari sono definiti in un linguaggio simile a quello naturale e vengono poi inseriti in un file con estensione *.feature*. Essi possono essere scritti anche con l'ausilio di tool quali Online Gherkin Editor [15], offerto da *SpecFlow*, oppure CucumberStudio, offerto dalla *SmartBear*.

Inoltre, nella scrittura gli scenari saranno necessarie alcune keywords [16] del linguaggio Gherkin, tra le quali:

- *Feature*, che descrive in linguaggio naturale la specifica di business (il cosa bisogna fare);
- *Scenario*, che è un esempio rappresentativo di come funzionerà la specifica di business;
- *Given, When, Then, And, But*, che sono utilizzati per descrivere gli step per gli scenari (il come deve essere definito);
- *Background*, che descrive un contesto (*Given*) comune a tutti gli scenari. Si applica alle feature che hanno più di uno scenario ed in cui tutti gli scenari condividono il contesto (passi propedeutici per l'azione) ed evita una ridondante ripetizione della descrizione del medesimo contesto in ogni scenario.

Utilizzando tali keywords, verranno definiti dei file con estensione “.feature”, che saranno redatti in lingua inglese utilizzando il linguaggio Gherkin e dai quali sarà poi possibile realizzare i test comportamentali della calcolatrice una volta sviluppata.

Si analizzerà ora nel dettaglio la struttura di questi file “.feature” partendo dalla keyword *Feature*.

**Feature:** Integer Arithmetic Expressions

This feature provides a range of scenarios corresponding to the intended external behaviour of arithmetic expressions on integers.

Sulla base di quanto indicato tramite *Feature*, chiunque legga è in grado di comprendere il generico comportamento che sarà analizzato. In particolare, il tester conosce la tipologia di test in oggetto, che in questo caso mira a verificare la correttezza nello svolgimento delle quattro operazioni elementari utilizzando numeri interi.

Si continuerà a strutturare il feature file con ulteriori informazioni sul singolo scenario, utilizzando la keyword *Scenario*, a cui seguirà la descrizione/nome in modo da chiarire ulteriormente quanto da realizzare. Nel caso in esame, lo scenario si riferisce all’operazione di somma di due numeri interi:

**Scenario:** Adding two integer numbers

Fatto ciò, si è pronti per descrivere concretamente lo scenario utilizzando la sintassi Gherkin, partendo dal *Given* che consente di definire la precondizione:

**Given** I initialise a calculator

Nel caso della somma di due interi, la precondizione corrisponderà all’avvio della calcolatrice e tale operazione, qualora non effettuata, non consentirà di portare a termine la user story, e di conseguenza la continuazione del test, durante il testing.

In seguito, sarà possibile definire nello specifico le azioni che l’utente effettuerà sul programma per realizzare lo scenario di somma. Tale sequenza d’azioni consisterà nell’inserimento del numero e dell’operazione, che in Gherkin sono tradotti e realizzati tramite la parola chiave *When*.

```
When an integer operation '+'
And I provide a first number 4
And I provide a second number 5
```

Si noti come sia stata utilizzata anche la keyword *And*. Essa è una parola chiave che permette di combinare più istruzioni *Given*, *When* e *Then*, evitando così la ripetizione della stessa istruzione più volte. Infatti, dal punto di vista semantico e del funzionamento nulla sarebbe mutato se al posto di *And* fossero stati scritti più *When* e viceversa.

Altra considerazione è che in Gherkin non è possibile definire la OR, in quanto nel testing si è vincolati a seguire una precisa sequenza di azioni senza possibilità di scelta, in modo tale che il tester possa conoscere precisamente l’effettivo risultato atteso. Qualora si

vogliano testare scelte differenti, sarà quindi necessario definire uno scenario per ogni scelta.

Non resta ora che definire l'ultimo passaggio nella descrizione dello scenario. Al fine di terminare la stesura di esso si utilizza la keyword *Then*, che permette di definire la post condizione. In questo modo, si potrà confrontare il risultato del software con quello atteso. Nel caso in oggetto, esso consiste nel risultato dell'operazione di somma:

```
Then the operation evaluates to 9
```

A questo punto, la definizione dello scenario di test è completa e si osserva, inoltre, che sia i numeri che l'operazione da effettuare sono visti come parametri, e che, quindi, è possibile scrivere, utilizzando la stessa forma, anche ulteriori scenari con diverse combinazioni.

Si riportano qui di seguito gli scenari di test delle quattro operazioni:

```
Scenario: Adding two integer numbers
Given I initialise a calculator
When an integer operation '+'
And I provide a first number 4
And I provide a second number 5
Then the operation evaluates to 9
```

```
Scenario: Subtracting two integer numbers
Given I initialise a calculator
When an integer operation '-'
And I provide a first number 7
And I provide a second number 5
Then the operation evaluates to 2
```

```
Scenario: Multiplying two integer numbers
Given I initialise a calculator
When an integer operation '*'
And I provide a first number 7
And I provide a second number 5
Then the operation evaluates to 35
```

```
Scenario: Dividing two integer numbers
Given I initialise a calculator
When an integer operation '/'
And I provide a first number 7
And I provide a second number 5
Then the operation evaluates to 1
```

### **Miglioramenti:**

Una volta descritti tutti gli scenari, risulta opportuno ottimizzare la metodologia presentata attraverso l'utilizzo di ulteriori keywords messe a disposizione dal linguaggio Gherkin. Analizzando il feature file presentato in precedenza, una prima osservazione ricade sulla necessità di ogni scenario di inizializzare la calcolatrice. In Gherkin è possibile definire tale precondizione come il “contesto” iniziale comune ad ogni scenario e per fare ciò si ricorrerà alla keyword *Background*:

**Background:**

```
Given I initialise a calculator
```

Tale operazione sarà effettuata prima di ogni scenario e, una volta definita, sarà possibile attribuirla ad ogni scenario, rimuovendo per ognuno di essi una riga di *When*:

**Scenario:** Adding two integer numbers

```
Given an integer operation '+'
```

```
When I provide a first number 4
```

```
And I provide a second number 5
```

```
Then the operation evaluates to 9
```

**Scenario:** Subtracting two integer numbers

```
Given an integer operation '-'
```

```
When I provide a first number 7
```

```
And I provide a second number 5
```

```
Then the operation evaluates to 2
```

**Scenario:** Multiplying two integer numbers

```
Given an integer operation '*'
```

```
When I provide a first number 7
```

```
And I provide a second number 5
```

```
Then the operation evaluates to 35
```

**Scenario:** Dividing two integer numbers

```
Given an integer operation '/'
```

```
When I provide a first number 7
```

```
And I provide a second number 5
```

```
Then the operation evaluates to 1
```

Un’ulteriore analisi porta a constatare che gli scenari sono definiti da parametri, ossia da numeri ed operazioni che possono seguire diverse combinazioni. Quindi, sembrerebbe logico poter eseguire lo stesso scenario per combinazioni di numeri differenti, senza dover necessariamente riscrivere ogni volta le istruzioni.

Ciò è reso possibile tramite il costrutto *Scenario Outline*, che permette di inserire nello scenario dei parametri definiti in una tabella chiamata *Examples*.

**Scenario Outline:** Evaluating arithmetic operations with two integer parameters

```
Given an integer operation <op>
```

```
When I provide a first number <n1>
```

```
And I provide a second number <n2>
```

```
Then the operation evaluates to <result>
```

**Examples:**

op	n1	n2	result
+"	4	5	9
"-"	8	5	3
"*"	7	2	14
"/"	6	2	3

Osservando il codice riportato sopra, è possibile notare come lo scenario risulti pressoché invariato e come siano state sostituite sia la keyword Scenario con Scenario Outline. Inoltre, all’interno della descrizione dello scenario, che ora si riferisce ad una generica operazione elementare, sono stati inseriti al posto dei numeri dei place holders racchiusi

nelle parentesi angolari <...> e nella tabella "Examples" sono inseriti i valori da sostituire per ogni placeholder.

È bene notare che non è importante l'ordine in cui sono inseriti i placeholders nella tabella, a patto che essi siano coerenti con la propria intestazione.

Ulteriori migliorie possono essere apportate al feature file nel momento in cui sussistono diverse tipologie di test da effettuare. Ad esempio, si potrebbe voler definire uno scenario con una parola chiave specifica e, in questo modo rendere possibile l'esecuzione singola dei i test anziché totale. In Gherkin ciò può avvenire grazie alla possibilità di inserire dei tag, definiti come segue: "@NomeTag", che consentono di identificare un particolare scenario.

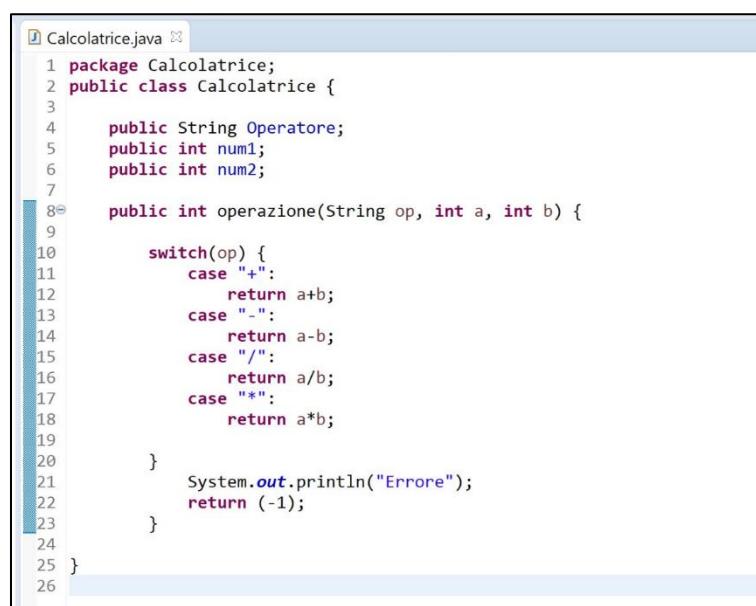
Per approfondire meglio tutte le keywords presenti nel linguaggio Gherkin, è possibile consultare la tabella nel paragrafo 2.4 a fine capitolo.

## 2.1.2 Passo 2: Implementazione del codice

Definito il feature file e discussi gli scenari prodotti a partire dalle user stories, si può pensare al development dei casi di test relativi ai vari scenari. Dunque, superata la fase di accettazione gli scenari, essi vengono implementati mediante la scrittura di codice da parte dello sviluppatore.

Nell'esempio, come già anticipato nei precedenti paragrafi, di realizzazione di una semplice calcolatrice, si utilizzeranno il linguaggio Java e l'ambiente di sviluppo Eclipse IDE.

In figura è mostrato il codice che implementa la predetta calcolatrice:



```
Calcolatrice.java
1 package Calcolatrice;
2 public class Calcolatrice {
3
4     public String Operatore;
5     public int num1;
6     public int num2;
7
8     public int operazione(String op, int a, int b) {
9
10        switch(op) {
11            case "+":
12                return a+b;
13            case "-":
14                return a-b;
15            case "/":
16                return a/b;
17            case "*":
18                return a*b;
19
20        }
21        System.out.println("Errore");
22        return (-1);
23    }
24
25 }
```

Figura 6: Calcolatrice implementata in Java

### 2.1.3 Passo 3: Installazione di Cucumber e creazione del progetto

Sviluppato il codice relativo alla calcolatrice, si può procedere al testing. È importante notare come in questa fase di sviluppo è necessario affidarsi ad un framework che consenta di trasformare gli scenari scritti in Gherkin in test.

Nel presente elaborato sarà utilizzato il framework Cucumber, che consiste in una tecnologia che consente agli sviluppatori di fornire un codice che trasforma le specifiche descritte in linguaggio naturale in test automatizzati.

Esso nasce inizialmente a supporto del linguaggio Ruby, ma, nel corso degli anni, ha iniziato a supportare svariati linguaggi come Java, Javascript, Php ecc. ed è stato integrato in numerosi ambienti di sviluppo, tra cui Eclipse IDE.

Cucumber, essenzialmente, ha la funzione di leggere file di testo (.feature), estraendo gli scenari ed eseguendo ogni scenario nel sistema da testare.

Da un punto di vista tecnico, ciò si ottiene traducendo le istruzioni contenute nel .feature e descritte in linguaggio Gherkin [12], in steps, ossia metodi che riportano le istruzioni in codice adibito al testing.

Per installare Cucumber.io in Eclipse è necessario effettuare i seguenti step:

1. Selezionare l'icona Help e cliccare su Eclipse Marketplace fig[7];
2. Inserire la parola chiave “Cucumber” nell'apposita barra di ricerca fig[8];
3. Selezionare Install ed accettare i termini della licenza fig[9];
4. Riavviare il programma.

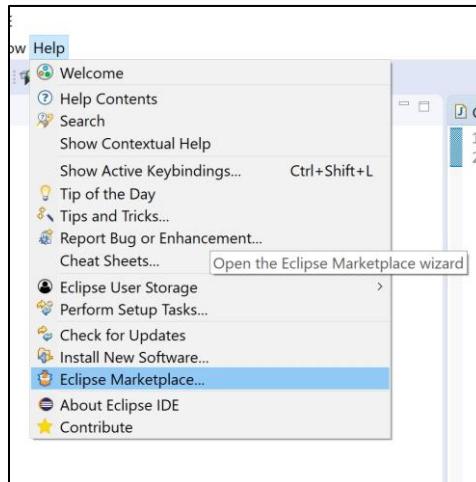
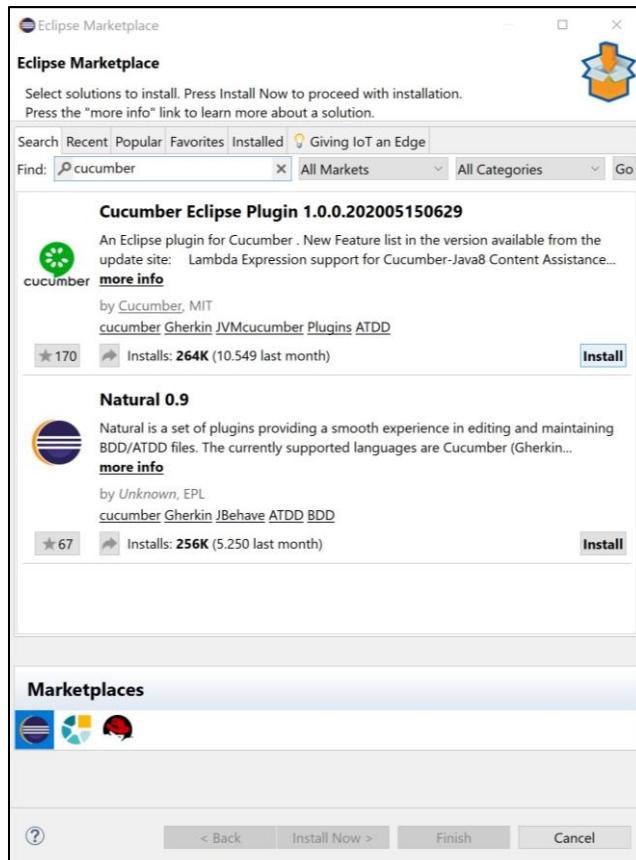
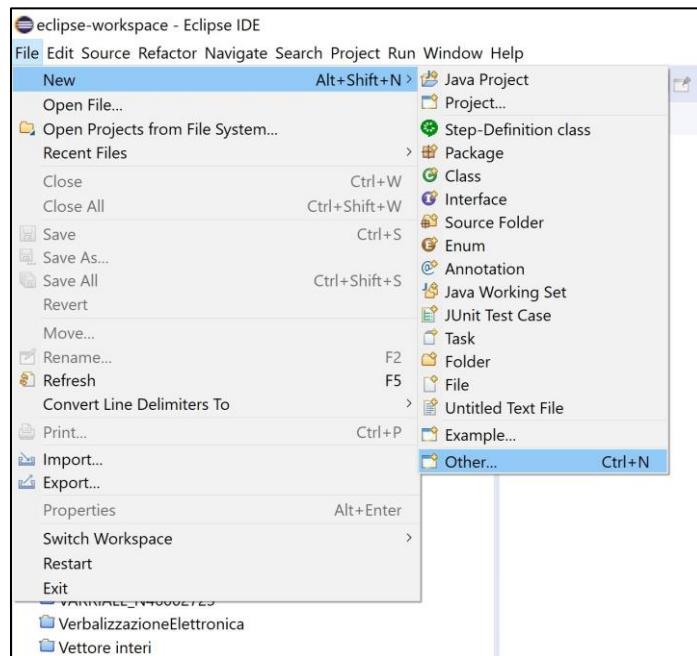


Figura 7: Eclipse Marketplace



**Figura 8: Installazione Plugin Cucumber**



**Figura 9: Creazione di un nuovo progetto**

Una volta installato Cucumber, sarà possibile utilizzarlo nel progetto che si andrà a creare. In particolare, per la creazione di un nuovo progetto si ricorrerà ad un progetto “Maven”

(Figura [10]). Si selezionerà tale tipologia di progetto poiché, tramite il file pom.xml in Maven, sarà possibile includere una serie di dependencies, per permettere l'integrazione di librerie necessarie alla stesura ed esecuzione dei test. Tale processo di creazione del progetto è reso più chiaro dalle figure [11, 12, 13]:

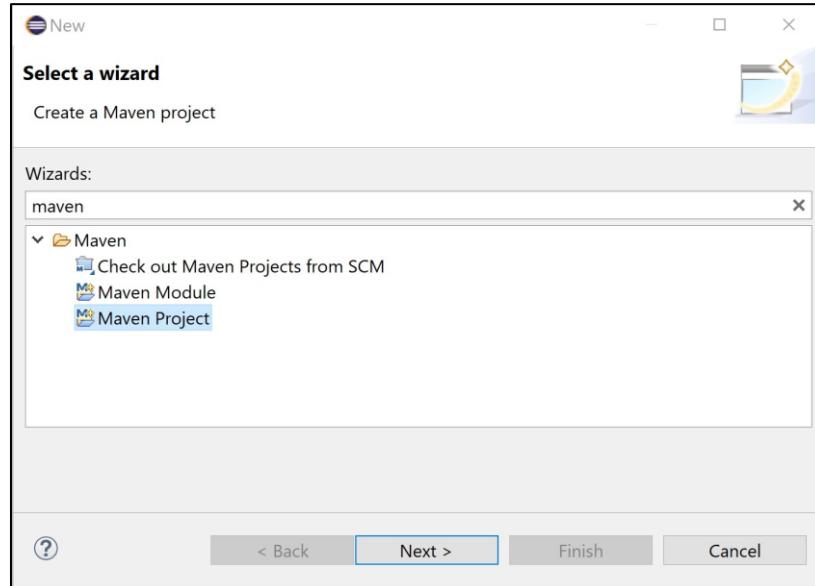


Figura 10: Creazione del progetto Maven

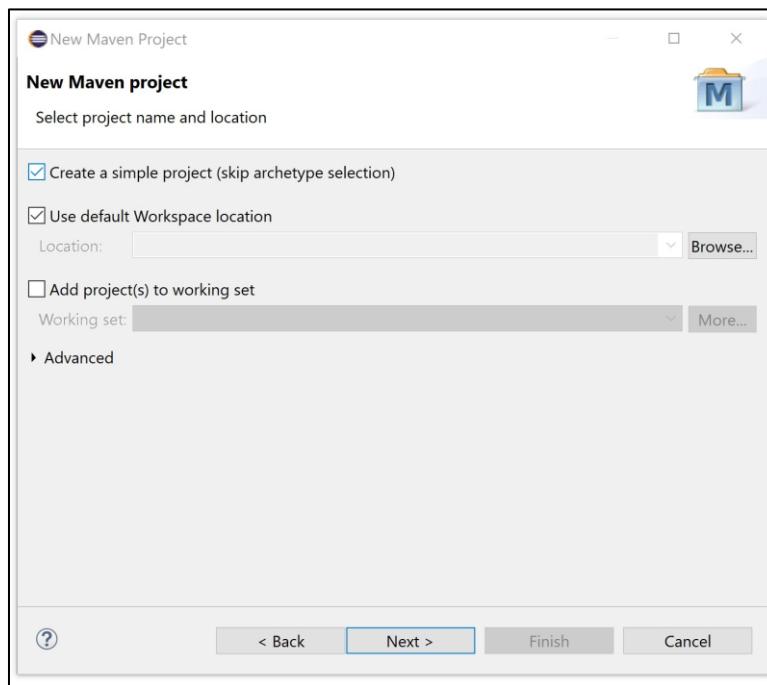


Figura 11: Schermata per la creazione del progetto Maven

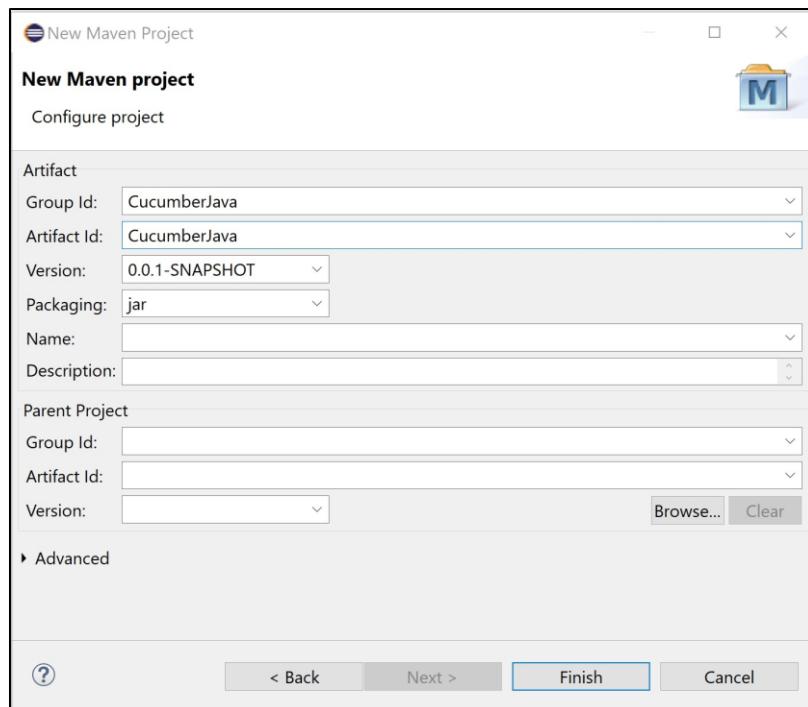


Figura 12: Seconda schermata per la creazione del progetto Maven

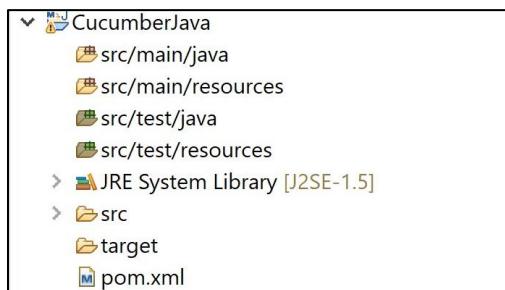


Figura 13: Progetto Maven risultante

Una volta creato il progetto, appariranno i seguenti file (Figura 13).

A questo punto, sarà necessario copiare le dependencies dal seguente link ed incollarle nel file pom.xml:

<https://mvnrepository.com/artifact/io.cucumber/cucumber-java>

The screenshot shows the Maven Repository page for the artifact `io.cucumber:cucumber-java:6.10.0`. Key details include:

- Indexed Artifacts (19.6M)**: A chart showing the growth of indexed artifacts over time.
- Cucumber JVM: Java > 6.10.0**: The main artifact information.
- License**: MIT.
- Date**: Feb 14, 2021.
- Files**: jar (670 KB) - View All.
- Repositories**: Central.
- Used By**: 252 artifacts.
- Code Analyzers**: Maven, Gradle, SBT, Ivy, Gape, Leiningen, Buildr.
- Include comment with link to declaration**: A checked checkbox with a "Copied to clipboard!" message.
- Compile Dependencies (2)**: A section listing dependencies used in compilation.

Figura 14: Dipendenze fornite dal Maven Central

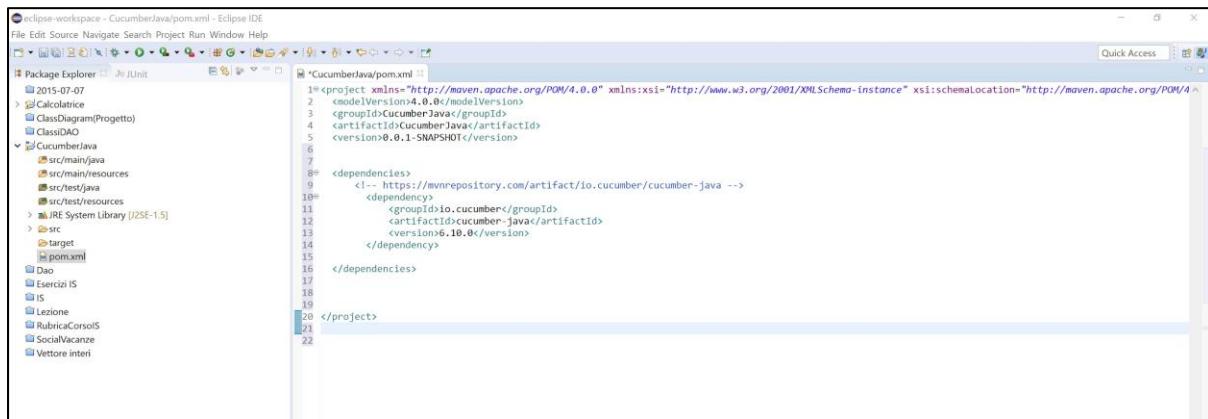


Figura 15: File pom.xml

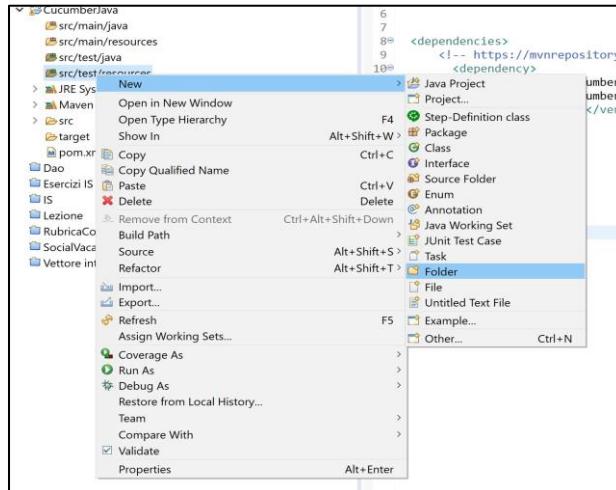
Una volta salvato il file apparirà:



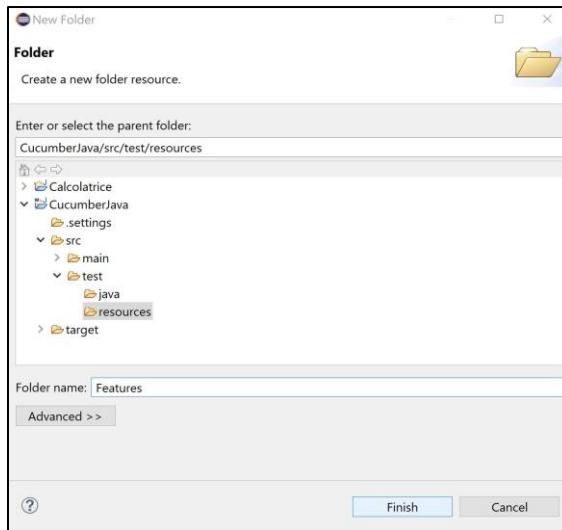
**Figura 16: Cartelle risultanti dopo l'inserimento delle dipendenze**

Effettuate le già menzionate operazioni, è necessario inserire nel file sia il codice che gli scenari.

Il codice sarà inserito nella cartella *src/main/java*, mentre gli scenari dovranno essere inseriti in una cartella denominata Features nel percorso *src/test/resources* con l'estensione .feature (nell'esempio Scenario.feature), così come mostrato nelle seguenti figure:



**Figura 17: Creazione della cartella Features (1/2)**



**Figura 18: Creazione della cartella Features (2/2)**

Effettuati correttamente tutti i passaggi, il progetto e gli scenari appariranno nella finestra “Package Explorer”.

```

eclipse-workspace - CucumberJava/src/test/resources/Features/Scenario.feature - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer JUnit CucumberJava/pom.xml *Scenario.feature
2015-07-07
Calcolatrice JRE System Library [JavaSE-12]
src
  Calcolatrice
  JUnit 3
  ClassDiagram(Progetto)
  ClassDAO
CucumberJava
  src/main/java
  src/main/resources
  src/test/java
  src/test/resources
    Features
      Scenario.feature
JRE System Library [J2SE-1.5]
Maven Dependencies
src
  main
  test
  target
  pom.xml
  Dao
  Esercizi IS
  FizzBuzz
  IS
  Lezione
  RubricaCorsoIS
  SocialVacanze
  Vettore interi
1 #Author: your.email@your.domain.com
2 #Keywords: Summary
3 #Feature: List of scenarios.
4 #Scenario: Business rule through list of steps with arguments.
5 #Given: Some precondition step
6 #When: Some key actions
7 #Then: To observe outcomes or validation
8 #And,But: To enumerate more Given,When,Then steps
9 #Scenario Outline: List of steps for data-driven as an Examples and <placeholder>
10 #Examples: Container for s table
11 #Background: List of steps run before each of the scenarios
12 """ (Doc Strings)
13 #| (Data Tables)
14 #@ (Tags/Labels):To group Scenarios
15 #<> (placeholder)
16 """
17 ## (Comments)
18 #Sample Feature Definition Template
19 @tag
20 Feature: Title of your feature
21 I want to use this template for my feature file
22
23 @tag1
24 Scenario: Title of your scenario
25 Given I want to write a step with precondition
26 And some other precondition
27 When I complete action
28 And some other action
29 And yet another action
30 Then I validate the outcomes
31 And check more outcomes
32
33 @tag2
34 Scenario Outline: Title of your scenario outline
35 Given I want to write a step with <name>
36 When I check for the <value> in step
37 Then I verify the <status> in step
38
39 Examples:
40 | name | value | status |
41 | name1 | 5 | success |
42 | name2 | 7 | Fail |
43

```

**Figura 19: File .feature appena creato**

The screenshot shows the Eclipse IDE interface. In the top right, the code editor displays the `Scenario.feature` file. The content of the file is:

```

1 Feature: Integer Arithmetic Expressions
2 This feature provides a range of scenarios corresponding to the
3 intended external behaviour of arithmetic expressions on integers.
4
5 Background:
6 Given I initialise a calculator
7
8 Scenario Outline: Evaluating arithmetic operations with two integer parameters
9 Given an integer operation <op>
10 When I provide a first number <n1>
11 And I provide a second number <n2>
12 Then the operation evaluates to <result>
13
14 Examples:
15 | op | n1 | n2 | result |
16 | "+" | 4 | 5 | 9 |
17 | "-" | 8 | 5 | 3 |
18 | "*" | 7 | 2 | 14 |
19 | "/" | 6 | 2 | 3 |
20
21
22

```

In the bottom left, the Package Explorer shows the project structure:

- eclipse-workspaceCucumber - CucumberJava/src/test/resources/Features/Scenario.feature - Eclipse IDE
- File Edit Navigate Search Project Run Window Help
- Package Explorer JUnit
- calcolatriceJUnit4
- CucumberFizzBuzz
- CucumberJava
  - src/main/java
    - Calcolatrice
      - Calcolatrice.java
  - src/main/resources
  - src/test/java
  - src/test/resources
    - Features
      - Scenario.feature
      - Scenario2.feature
      - Scenario3.feature
  - JRE System Library [JavaSE-1.7]
  - Maven Dependencies
  - src
  - target
  - pom.xml
- CucumberSelenium

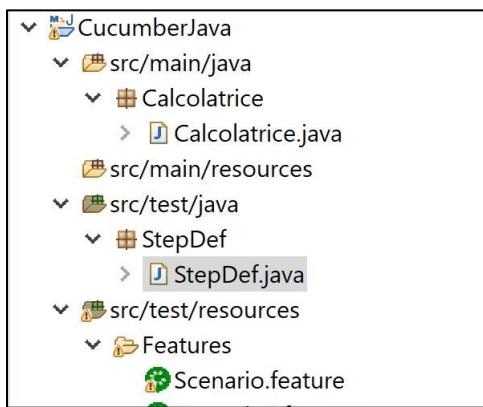
**Figura 20: Scenario.feature**

## 2.1.4 Passo 4: Generazione degli Steps

Nei passi precedenti si è mostrato come strutturare il feature file, implementare il codice da testare, creare il progetto ed installare il plug-in Cucumber. Ora si procederà ad illustrare in che modo è possibile effettuare la trasformazione degli scenari nei test da eseguire.

In particolare, ogni riga dello scenario diverrà uno step e sarà mappata in un blocco di codice eseguibile nel linguaggio adottato che, nel caso in esame, sarà Java. Ciò vuol dire che è presente un “collegamento” tra le righe di codice scritte mediante Given, When, Then ed i nomi dei metodi implementati nel file di test.

È buona pratica, prima di procedere alla creazione di tali metodi, creare un package “StepDef” ed una classe Java al suo interno. Tale package sarà localizzato nella cartella `src/test/java`, come mostrato in figura:



**Figura 21: Percorso del file Stepdef.java**

All’interno della classe Java appena creata, si andranno a definire i metodi che consentiranno di eseguire lo scenario in esame.

La scrittura di tali metodi può essere effettuata a mano, ma grazie al plugin Cucumber, esiste la possibilità di definire la firma dei metodi in modo automatico.

Per ottenere tali metodi si andrà ad eseguire all'interno di Eclipse il file .feature, selezionando per esso “Run As Cucumber Feature” come mostrato nella figura seguente:

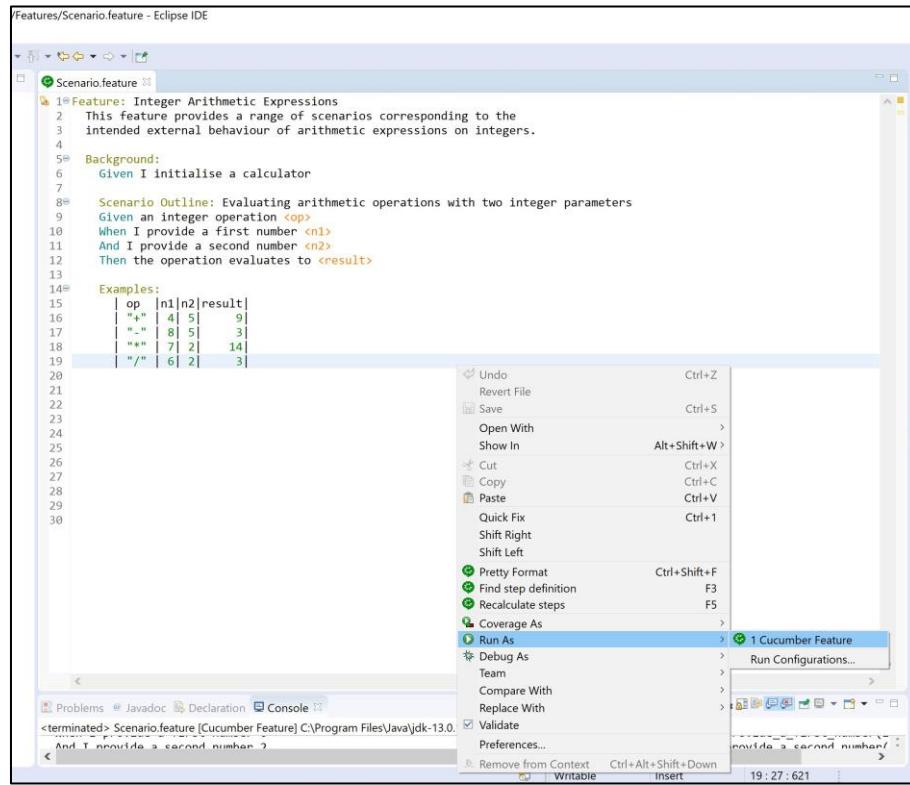


Figura 22: Come effettuare il run dello scenario

Tale Run porterà al seguente risultato:

The screenshot shows the Eclipse IDE interface with the 'Console' tab selected. The console output is as follows:

```
<terminated> Scenario.feature [Cucumber Feature] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (14 mar 2021, 10:33:30)
0m0s, 519s
```

Below the console, a message reads: "You can implement missing steps with the snippets below:". The generated Java code consists of several methods corresponding to the feature steps:

```
@Given("I initialise a calculator")
public void i_initialise_a_calculator() {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}

@Given("an integer operation {string}")
public void an_integer_operation(String string) {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}

@When("I provide a first number {int}")
public void i_provide_a_first_number(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}

@When("I provide a second number {int}")
public void i_provide_a_second_number(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}

@Then("the operation evaluates to {int}")
public void the_operation_evaluates_to(Integer int1) {
    // Write code here that turns the phrase above into concrete actions
    throw new io.cucumber.java.PendingException();
}
```

Figura 23: Step generati dal Run

Gli stessi metodi potranno essere copiati ed inseriti nel file StepDef.java, aggiungendo anche l'import della libreria seguente per consentire la comprensione delle annotazioni @Given, @When, @Then:

```
import io.cucumber.java.en.*;
```

Il file StepDef.java si presenterà in questo modo:

```

1 package StepDef;
2
3 import io.cucumber.java.en.*;
4
5 public class StepDef {
6
7     @Given("I initialise a calculator")
8     public void i_initialise_a_calculator() {
9         // Write code here that turns the phrase above into concrete actions
10        throw new io.cucumber.java.PendingException();
11    }
12
13     @Given("an integer operation {string}")
14     public void an_integer_operation(String string) {
15         // Write code here that turns the phrase above into concrete actions
16         throw new io.cucumber.java.PendingException();
17     }
18
19     @When("I provide a first number {int}")
20     public void i_provide_a_first_number(Integer int1) {
21         // Write code here that turns the phrase above into concrete actions
22         throw new io.cucumber.java.PendingException();
23     }
24
25     @When("I provide a second number {int}")
26     public void i_provide_a_second_number(Integer int1) {
27         // Write code here that turns the phrase above into concrete actions
28         throw new io.cucumber.java.PendingException();
29     }
30
31     @Then("the operation evaluates to {int}")
32     public void the_operation_evaluates_to(Integer int1) {
33         // Write code here that turns the phrase above into concrete actions
34         throw new io.cucumber.java.PendingException();
35     }
36
37
38 }
39

```

Figura 24: StepDef.java

#### Osservazione:

Nella generazione del codice, si può notare come i numeri nello scenario siano stati tradotti come integer, mentre le operazioni +,-,\*/ come stringhe<sup>2</sup>.

Inoltre, tale file non sempre risulta corretto, dato che è possibile effettuare delle modifiche a mano a seconda delle necessità del programma da testare.

Nel caso in esame, per eseguire il programma d'esempio, è necessario sostituire a mano i parametri di ingresso integer con int.

### 2.1.5 Passo 5: Implementazione degli Steps

Fino ad ora, sono stati descritti dei passaggi generali per la corretta impostazione del progetto. In questa sezione si entrerà nello specifico, traducendo gli scenari scritti in Gherkin in codice eseguibile all'interno dei vari step e tale traduzione varierà a seconda della tipologia di progetto.

Nel primo step è necessario verificare la precondizione di inizializzazione della calcolatrice. Per fare ciò, bisogna preliminarmente importare il package e creare nella classe un nuovo oggetto calcolatrice all'interno della classe StepDef:

```

import Calcolatrice.Calcolatrice;
Calcolatrice calc=new Calcolatrice();

```

---

<sup>2</sup> L'annotazione con i doppi apici in Gherkin genera delle stringhe, mentre i numeri vengono codificati come integer.

A questo punto, sarà possibile implementare il primo Given che indica la precondizione, per verificare che l'oggetto calcolatrice non è nullo:

```
@Given("I initialise a calculator")
public void i_initialise_a_calculator() {
    //PRECONDIZIONE
    assertNotNull(calc);
}
```

È da notare che è possibile cancellare la seguente riga di eccezione, che ricorda al programmatore di implementare quel metodo:

```
throw new io.cucumber.java.PendingException();
```

Lo step successivo è l'inserimento dell'operatore, che può essere implementato in questo modo:

```
@Given("an integer operation {string}")
public void an_integer_operation(String operatore) {
    //Do in ingresso l'operatore
    calc.Operatore=operatore;
}
```

Si osserva che nel file sono presenti due @Given nonostante ne sia stato definito uno. Ciò avviene poiché il background viene tradotto con un @Given.

Analogamente, è possibile definire anche i passaggi relativi ai @When, che prevedono l'inserimento dei due numeri interi:

```
@When("I provide a first number {int}")
public void i_provide_a_first_number(int numero1) {
    //Inserisco il primo numero
    calc.num1= numero1;
}

@When("I provide a second number {int}")
public void i_provide_a_second_number(int numero2) {
    //Inserisco il secondo numero
    calc.num2= numero2;
}
```

Particolare attenzione deve essere dedicata all'implementazione dello step relativo al @Then, nel quale vanno inserite l'asserzione e la post condizione, in modo da verificare il corretto funzionamento del programma.

```
@Then("the operation evaluates to {int}")
public void the_operation_evaluates_to(int risultato) {
    //Verifica del risultato atteso dello scenario con il risultato
    //del programma
    assertEquals("Sono
uguali",risultato,calc.operazione(calc.Operatore, calc.num1,
calc.num2));
    calc=null;
```

```

    //POSTCONDIZIONE
    assertNull(calc);
}

```

Nel caso in cui l'asserzione è verificata, si vedrà successivamente che il test è passato, viceversa in caso contrario si avrà un fallimento.

### **Miglioramenti:**

Esistono ulteriori miglioramenti possibili, anche dal punto di vista della codifica degli steps.

In particolare le precondizioni e le postcondizioni, qualora fossero comuni a tutti gli scenari da eseguire, possono essere dichiarate una sola volta tramite gli Hooks.

Gli *Hooks* [17] sono metodi eseguiti prima o dopo ogni scenario e possono essere definiti in un qualsiasi punto del codice degli steps mediante le annotazioni @Before e @After.

```

@Before
public void beforeCallingScenario() {
    System.out.println("***** About to start the scenario.");
}

```

Essi sono globali per default, ossia sono eseguiti per ogni scenario della funzionalità; qualora si vogliano utilizzare per determinati scenari, è necessario utilizzare dei tag. Se sono presenti più Hooks, è possibile definirne l'ordine come nel seguente esempio:

```

@Before( order = 20 )
public void exampleHook() {...}

```

Gli Hooks @Before sono eseguiti in ordine crescente; ad esempio, un Hook con un ordine 10 è eseguito prima di uno con ordine 20, mentre per gli Hooks @After l'ordine è opposto. Anche per gli Hooks dotati di tag è possibile definire un ordine andando ad inserire un parametro value come nell'esempio che segue:

```

@Before( value ="myTag", order = 20)
public void exampleHook() {...}

```

Tornando all'esempio iniziale, si vede come essi possono essere utilizzati per stabilire le precondizioni e le post condizioni, inserendo all'interno della classe i metodi e rimuovendoli dagli step di Given e Then scritti in precedenza:

```

@Before()
public void setup() {
    //PRECONDIZIONE
    assertNotNull(calc);
}

@After()
public void teardown() {
    calc=null;
}

```

```

    //POSTCONDIZIONE
    assertNull(calc);
}

```

Si osserva che non è importante che il Given “I initialise a calculator” non abbia codice all’interno del metodo. In questo caso, però, non sarà possibile rimuovere lo step dalla classe.

Il codice della classe dovrebbe dunque presentarsi in questo modo:

```

Scenario.feature StepDef.java
1 package StepDef;
2
3 import io.cucumber.java.*;
4 import io.cucumber.java.en.*;
5 import static org.junit.Assert.*;
6 import Calcolatrice.Calcolatrice;
7
8 public class StepDef {
9
10     Calcolatrice calc=new Calcolatrice();
11
12     @Before()
13     public void setup() {
14         //PRECONDIZIONE
15         assertNotNULL(calc);
16     }
17
18     @Given("I initialise a calculator")
19     public void i_initialise_a_calculator() {
20     }
21
22     @Given("an integer operation {string}")
23     public void an_integer_operation(String operatore) {
24         //Do in ingresso l'operatore
25         calc.Operatore=operatore;
26     }
27
28     @When("I provide a first number {int}")
29     public void i_provide_a_first_number(int numero1) {
30         //Inserisco il primo numero
31         calc.num1= numero1;
32     }
33
34     @When("I provide a second number {int}")
35     public void i_provide_a_second_number(int numero2) {
36         //Inserisco il secondo numero
37         calc.num2= numero2;
38     }
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
880
881
882
883
884
885
886
887
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
948
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1277
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1286
1287
1288
1288
1289
1290
1291
1292
1293
1294
1295
1296
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1377
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1386
1387
1388
1388
1389
1390
1391
1392
1393
1394
1395
1396
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1466
1467
1468
1468
1469
1470
1471
1472
1473
1474
1475
1476
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1486
1487
1488
1488
1489
1490
1491
1492
1493
1494
1495
1496
1496
1497
1498
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1566
1567
1568
1568
1569
1570
1571
1572
1573
1574
1575
1576
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1586
1587
1588
1588
1589
1590
1591
1592
1593
1594
1595
1596
1596
1597
1598
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1686
1687
1688
1688
1689
1690
1691
1692
1693
1694
1695
1696
1696
1697
1698
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1786
1787
1788
1788
1789
1790
1791
1792
1793
1794
1795
1796
1796
1797
1798
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1877
1878
1879
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1896
1897
1898
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1977
1978
1979
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1996
1997
1998
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2077
2078
2079
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2096
2097
2098
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
21
```

```

59
40@  @Then("the operation evaluates to {int}")
41  public void the_operation_evaluates_to(int risultato) {
42      //Verifica del risultato atteso dello scenario con il risultato del programma
43      assertEquals("Sono uguali",risultato,calc.operazione(calc.Operatore, calc.num1, calc.num2));
44  }
45
46@  @After()
47  public void teardown() {
48      calc=null;
49      //POSTCONDIZIONE
50      assertNull(calc);
51  }
52
53 }
54

```

Figura 26: Stepdef.java (2/2)

Ricapitolando, dal Passo 1 al Passo 5 è stato trasformato il linguaggio naturale nel codice che consente di eseguire i test, così come mostrato in tabella:

Linguaggio naturale	Scenario in Gherkin	Definizione degli steps																				
This feature provides a range of scenarios corresponding to the intended external behaviour of arithmetic expressions on integers.	<p><b>Scenario Outline:</b> Evaluating arithmetic operations with two integer parameters</p> <p><b>Given</b> an integer operation &lt;op&gt;</p> <p><b>When</b> I provide a first number &lt;n1&gt;</p> <p><b>And</b> I provide a second number &lt;n2&gt;</p> <p><b>Then</b> the operation evaluates to &lt;result&gt;</p> <p><b>Examples:</b></p> <table border="1"> <thead> <tr> <th>op</th> <th>n1</th> <th>n2</th> <th>result</th> </tr> </thead> <tbody> <tr> <td>“+”</td> <td>4</td> <td>5</td> <td>9</td> </tr> <tr> <td>“-”</td> <td>8</td> <td>5</td> <td>3</td> </tr> <tr> <td>“*”</td> <td>7</td> <td>2</td> <td>14</td> </tr> <tr> <td>“/”</td> <td>6</td> <td>2</td> <td>3</td> </tr> </tbody> </table>	op	n1	n2	result	“+”	4	5	9	“-”	8	5	3	“*”	7	2	14	“/”	6	2	3	<pre> import Calcolatrice.Calcolatrice; Calcolatrice calc=new Calcolatrice();  @Given("I initialise a calculator") public void i_initialise_a_calculator() {     //PRECONDIZIONE     assertNotNull(calc); }  @Given("an integer operation {string}") public void an_integer_operation(String operatore) {     //Do in ingresso l'operatore     calc.Operatore=operatore; }  @When("I provide a first number {int}") public void i_provide_a_first_number(int numero1) {     //Inserisco il primo numero     calc.num1= numero1; }  @When("I provide a second number {int}") public void i_provide_a_second_number(int numero2) {     //Inserisco il secondo numero     calc.num2= numero2; }  @Then("the operation evaluates to {int}") public void the_operation_evaluates_to(int risultato) {     assertEquals("Sono uguali",risultato,calc.operazio ne(calc.Operatore, calc.num1, calc.num2));     calc=null;     //POSTCONDIZIONE     assertNull(calc); } </pre>
op	n1	n2	result																			
“+”	4	5	9																			
“-”	8	5	3																			
“*”	7	2	14																			
“/”	6	2	3																			

Tabella 2: Dal linguaggio naturale alla definizione degli steps

## 2.1.6 Passo 6: Esecuzione dei test e valutazione

Una volta scritti gli steps, è possibile effettuare il run dello scenario, come già mostrato in Figura 22.

Nella console appariranno gli scenari eseguiti (quattro in questo caso), il numero di steps effettuati (cinque per ogni scenario), il relativo stato (ossia passato o fallito) ed il tempo impiegato. (Figura sotto).

Nell'esempio mostrato tutti i test sono superati, ma qualora lo scenario o il codice del programma o del test fossero sbagliati, il risultato sarebbe differente, come in Figura 28.

Toccherà poi al tester valutare se l'errore sia dovuto al test scritto, oppure demandare alla fase di debug la ricerca e la soluzione del difetto o dei difetti del codice che causano il fallimento.

```
<terminated> Scenario.feature [Cucumber Feature] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (14 mar 2021, 16:40:15)
mar 14, 2021 4:40:15 PM io.cucumber.core.cli.Main run
WARNING: By default Cucumber is running in --non-strict mode.
This default will change to --strict and --non-strict will be removed.
You can use --strict to suppress this warning

Scenario Outline: Evaluating arithmetic operations with two integer parameters # src/test/resources/Features/Scenario.feature
  Given I initialise a calculator
  Given an integer operation "+"
  When I provide a first number 4
  And I provide a second number 5
  Then the operation evaluates to 9

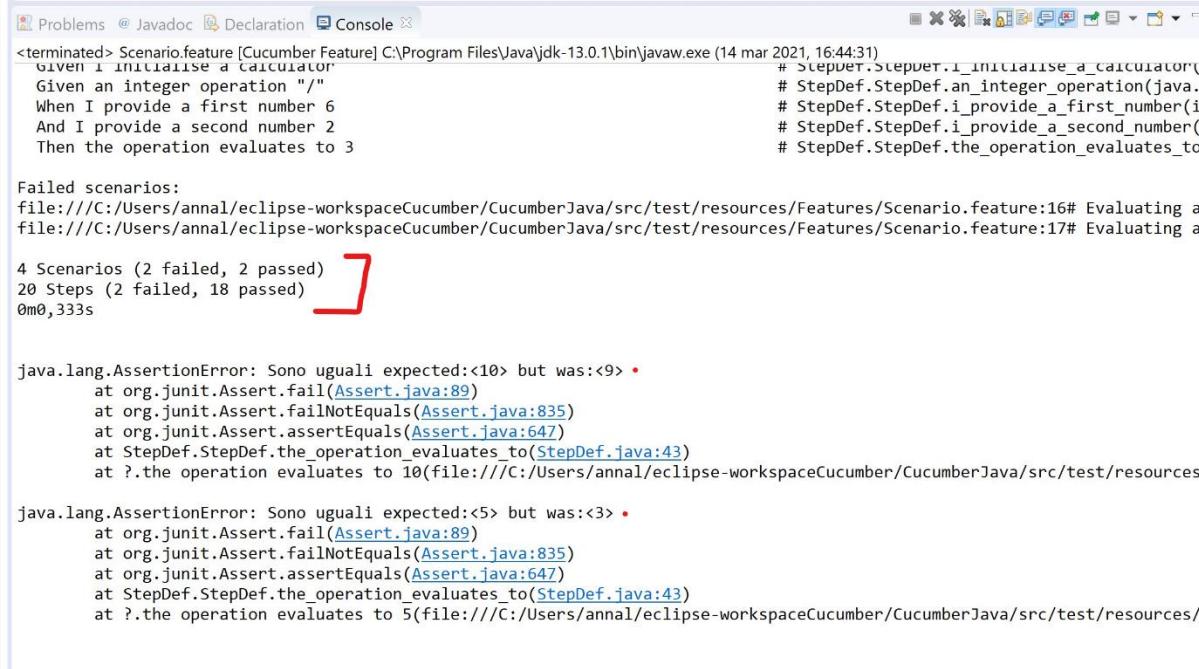
Scenario Outline: Evaluating arithmetic operations with two integer parameters # src/test/resources/Features/Scenario.feature
  Given I initialise a calculator
  Given an integer operation "-"
  When I provide a first number 8
  And I provide a second number 5
  Then the operation evaluates to 3

Scenario Outline: Evaluating arithmetic operations with two integer parameters # src/test/resources/Features/Scenario.feature
  Given I initialise a calculator
  Given an integer operation "*"
  When I provide a first number 7
  And I provide a second number 2
  Then the operation evaluates to 14

Scenario Outline: Evaluating arithmetic operations with two integer parameters # src/test/resources/Features/Scenario.feature
  Given I initialise a calculator
  Given an integer operation "/"
  When I provide a first number 6
  And I provide a second number 2
  Then the operation evaluates to 3

4 Scenarios (4 passed)
20 Steps (20 passed)
0m0,328s
```

Figura 27: Risultati ottenuti dall'esecuzione dei test



```

<terminated> Scenario.feature [Cucumber Feature] C:\Program Files\Java\jdk-13.0.1\bin\javaw.exe (14 mar 2021, 16:44:31)
  Given I initialise a calculator          # StepDef.StepDef.i_initialise_a_calculator()
  Given an integer operation "/"           # StepDef.StepDef.an_integer_operation(java.
  When I provide a first number 6         # StepDef.StepDef.i_provide_a_first_number(i
  And I provide a second number 2         # StepDef.StepDef.i_provide_a_second_number(
  Then the operation evaluates to 3        # StepDef.StepDef.the_operation_evaluates_to

Failed scenarios:
file:///C:/Users/annal/eclipse-workspaceCucumber/CucumberJava/src/test/resources/Features/Scenario.feature:16# Evaluating a
file:///C:/Users/annal/eclipse-workspaceCucumber/CucumberJava/src/test/resources/Features/Scenario.feature:17# Evaluating a

  4 Scenarios (2 failed, 2 passed)
  20 Steps (2 failed, 18 passed)

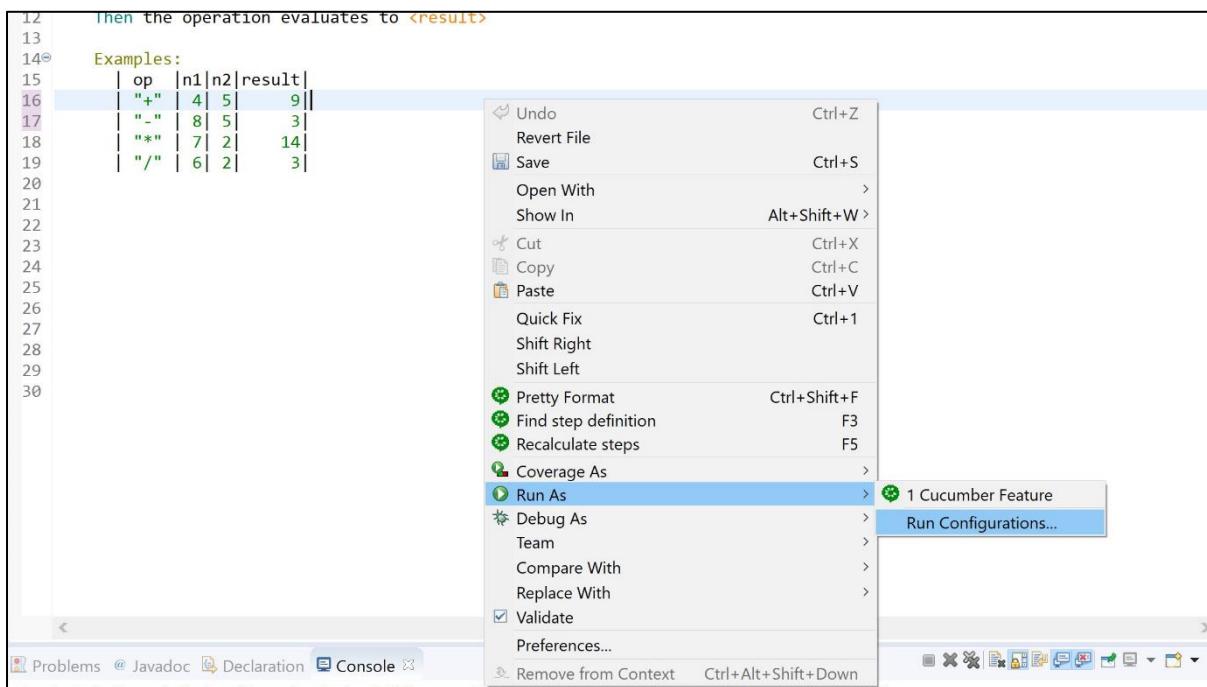
java.lang.AssertionError: Sono uguali expected:<10> but was:<9> *
  at org.junit.Assert.fail(Assert.java:89)
  at org.junit.Assert.failNotEquals(Assert.java:835)
  at org.junit.Assert.assertEquals(Assert.java:647)
  at StepDef.StepDef.the_operation_evaluates_to(StepDef.java:43)
  at ?.the operation evaluates to 10(file:///C:/Users/annal/eclipse-workspaceCucumber/CucumberJava/src/test/resources/Features/Scenario.feature:16)

java.lang.AssertionError: Sono uguali expected:<5> but was:<3> *
  at org.junit.Assert.fail(Assert.java:89)
  at org.junit.Assert.failNotEquals(Assert.java:835)
  at org.junit.Assert.assertEquals(Assert.java:647)
  at StepDef.StepDef.the_operation_evaluates_to(StepDef.java:43)
  at ?.the operation evaluates to 5(file:///C:/Users/annal/eclipse-workspaceCucumber/CucumberJava/src/test/resources/Features/Scenario.feature:17)

```

**Figura 28: Esempio di test non passato a causa di un errore nella scrittura del Then di due scenari.**

Il runner di Cucumber, oltre all'esecuzione degli scenari, consente anche di impostare alcune funzionalità dalla schermata Run Configurations:



**Figura 29: Run Configurations**

Apparirà la seguente schermata:

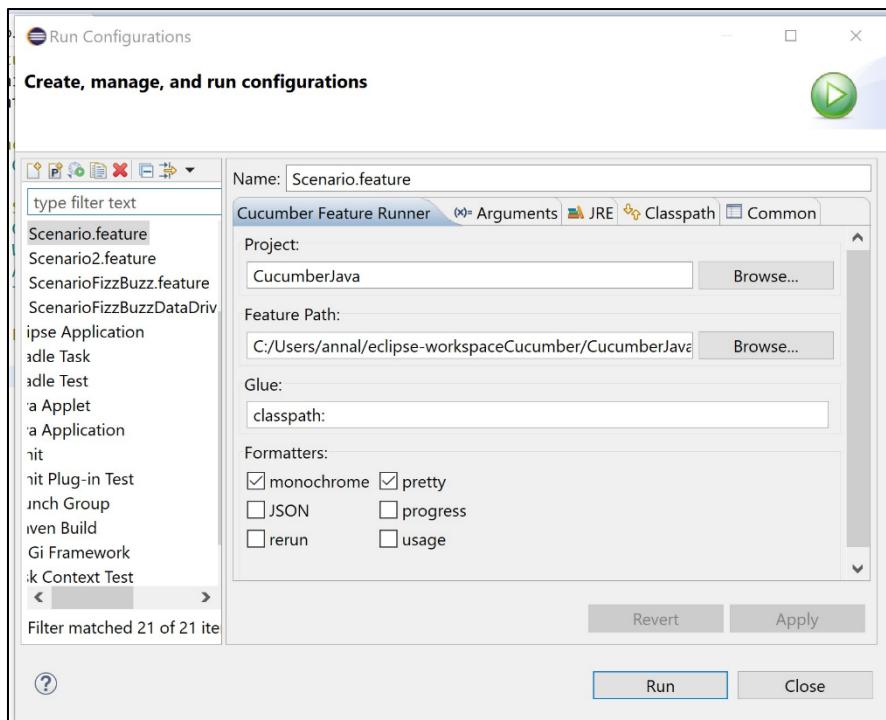


Figura 30: Schermata di configurazione del runner di Cucumber

Nel dettaglio:

- "Feature Path" contiene il percorso dei file delle feature;
- "Glue" contiene il path del file .feature;
- "Formatters" contiene i formattatori predefiniti quando non sono indicati esplicitamente.

I formatters che possono essere impostati sono:

<b>Formatter</b>	<b>Descrizione</b>
monochrome	Consente di vedere gli output di esecuzione in monocromatico. Esso è necessario in quanto di default gli output di Cucumber sono colorati e visibili in Windows soltanto installando "ANSICON".
pretty	Stampa il risultato così com'è.
JSON	Stampa il risultato come un JSON
progress	Stampa un carattere per scenario. In particolare, stampa un punto "." per ogni passaggio superato, una "F" per uno non riuscito, una "U" per un passaggio non definito e un trattino "-" per un passaggio saltato.
rerun	Stampa scenari di errore (test) con i numeri di riga. Tale opzione è utile per rieseguire solo gli scenari falliti, invece dell'intera funzionalità.
usage	Elenca tutte le definizioni dei passaggi, nonché i passaggi utilizzati e mostra le definizioni dei passi inutilizzate e ordina le definizioni dei passi in base al loro tempo medio di esecuzione. Tale output è utile per trovare rapidamente parte lente nel codice, ed è anche un ottimo modo per avere una panoramica delle definizioni dei passaggi.

**Tabella 3: Opzioni del runner Cucumber**

## 2.1.7 Passo 7: Esecuzione dei test con il Runner Junit (facoltativo)

Nel precedente paragrafo sono state descritte la formulazione ed esecuzione dei test a partire dagli scenari scritti in Gherkin, utilizzando il runner interno al plugin Cucumber. Esiste, però, un'altra soluzione che può risultare molto comoda ai tester, ossia l'esecuzione dei test tramite il runner di JUnit.

Ovviamente, i risultati dei test coincideranno, ma può essere utile visualizzare gli esiti allo stesso modo dei test scritti in JUnit (magari come quelli di unità già fatti all'inizio). Inoltre, con questa operazione, è possibile generare dei file di report dei risultati in formato html, xml o JSON a seconda delle esigenze, come sarà mostrato nel seguito dell'elaborato.

Per effettuare ciò, bisogna prima di tutto creare un file .java all'interno della cartella src/test/java, che nell'esempio sarà denominata TestRunner, e scrivere questo codice fuori dalla classe Test Runner:

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

@RunWith(Cucumber.class)
@CucumberOptions(features="src/test/resources/Features",
    glue= {"StepDef"},  
monochrome=true,  
)
```

All'interno del campo features, sarà specificato il path della cartella features che contiene gli scenari di test ed in glue il pattern della classe in cui sono implementati gli steps:

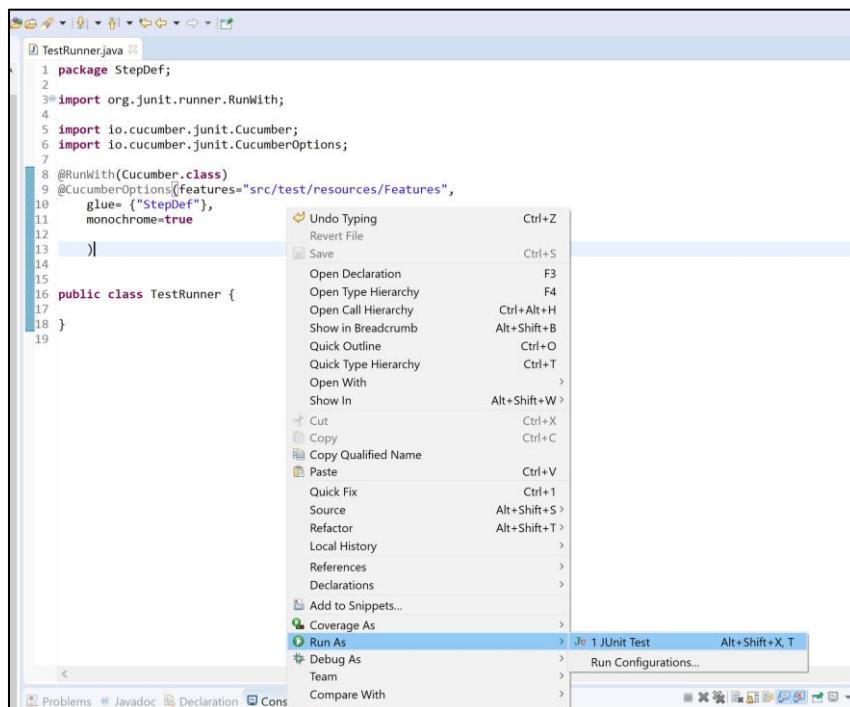


Figura 31: Run con JUnit

Come è possibile vedere dalla schermata, i test sono stati eseguiti con il runner JUnit 4 e nella barra a sinistra sono scritti i test eseguiti ed i vari risultati:

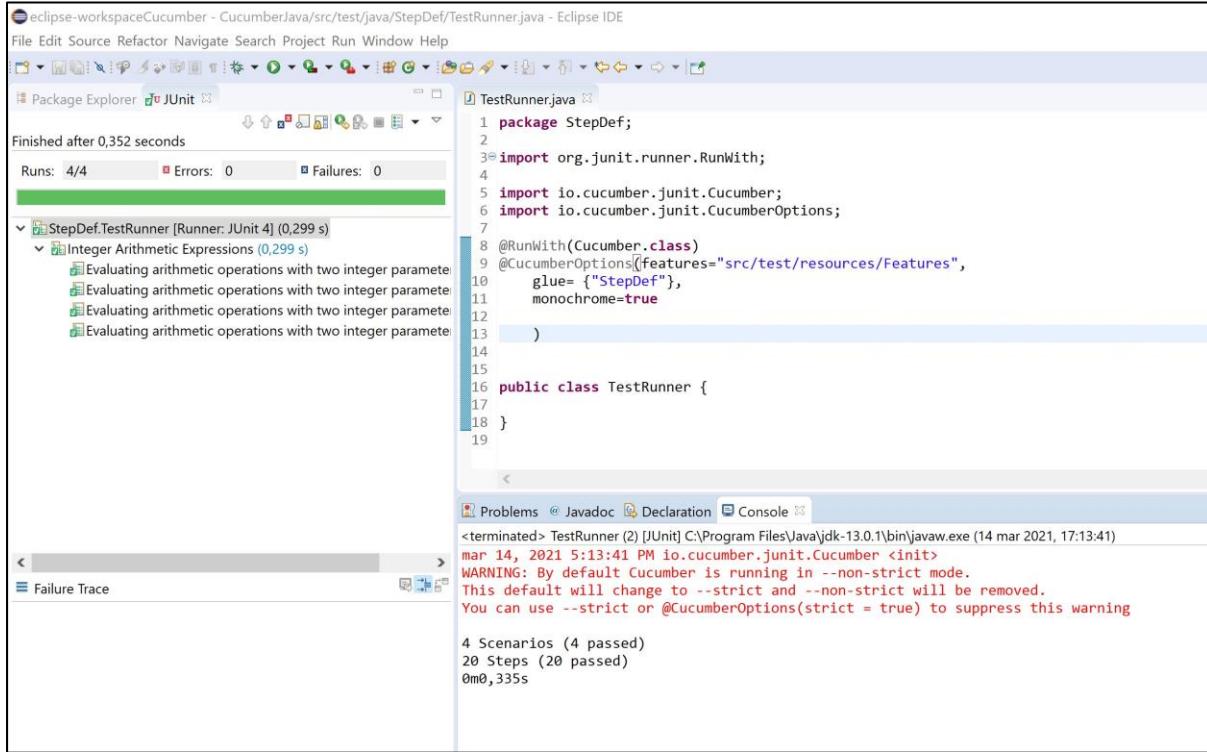


Figura 32: Risultati dopo l'esecuzione tramite il Runner di JUnit

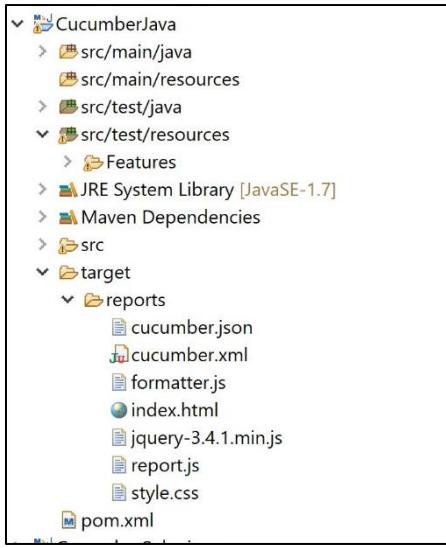
Come già accennato in precedenza, è possibile generare dei file di report in vari formati e, per poterli creare, è necessario aggiungere questo codice nelle CucumberOptions:

```
plugin= {"pretty", "html:target/reports", "json:target/reports/cucumber.json", "junit:target/reports/cucumber.xml"}
```

Per generare i report singolarmente:

```
plugin= {"pretty", "html:target/reports"},  
plugin= {"pretty", "json:target/reports/cucumber.json"},  
plugin= {"pretty", "junit:target/reports/cucumber.xml"}
```

Eseguendo nuovamente i test tramite JUnit ed effettuando un Refresh, i file di report saranno contenuti nella cartella reports:



**Figura 33:** Path della cartella reports

Di seguito, si riporta il file html di output, che permette una semplice lettura dei risultati:

op	n1	n2	result
"+"	4	5	9
"-"	8	5	3
"*"	7	2	14
"/"	6	2	3

**Background:**  
**Given** I initialise a calculator  
**Scenario Outline:** Evaluating arithmetic operations with two integer parameters  
**Given** an integer operation "+"  
**When** I provide a first number 4  
**And** I provide a second number 5  
**Then** the operation evaluates to 9  
**Background:**  
**Given** I initialise a calculator  
**Scenario Outline:** Evaluating arithmetic operations with two integer parameters  
**Given** an integer operation "-"  
**When** I provide a first number 8  
**And** I provide a second number 5  
**Then** the operation evaluates to 3  
**Background:**  
**Given** I initialise a calculator  
**Scenario Outline:** Evaluating arithmetic operations with two integer parameters  
**Given** an integer operation "\*"

**Figura 34:** File di report .html

Come accennato nella prima fase, è possibile anche eseguire test specifici definendo il parametro tag all'interno delle CucumberOption nel file TestRunner.java:

```
tags="@NomeTag"
```

Andando ad effettuare il run con il runner JUnit, si verifica che sono eseguiti solo i test con il tag indicato.

## 2.2 Esempio 2: Integrazione di Selenium

In questa sezione, sarà illustrato come può essere ampliato l'utilizzo di Cucumber, affiancando ad esso un ulteriore framework; in particolare, saranno analizzate le modalità attraverso cui combinare i frameworks Cucumber e Selenium [18] tra di loro, per il testing di una web application.

Al fine di mostrare come attuare la metodologia BDD, in quest' ambito sarà presentato come case-study il testing della navigazione come "guest" all'interno del sito docenti.unina.it. L'obiettivo è il testing di alcune funzionalità di base del sito citato precedentemente, ed in dettaglio si testeranno la funzionalità di ricerca di un docente e di apertura di alcune delle pagine inerenti al profilo cercato.

### 2.2.1 Passo 1: Produzione degli Scenari

Analogamente ai casi di studio precedenti, la prima fase consiste nella discussione con il cliente che porterà alla produzione delle user stories da utilizzare come punto di partenza all'interno del meeting tra PO, Developer e Tester. Al seguito del Three Amigos, si produrranno dei feature file che dalle storie utente sviluppano degli scenari in Gherkin.

In questo caso, partendo dalla descrizione della feature, si ottiene:

```
Feature: Search on Docenti unina
  As guest I want to search for a professor
    so that i can access some sections
```

Si nota come in questo caso la user story semplifichi la sezione di descrizione della Feature. Sarà utilizzato, all'interno di questo esempio, il costrutto Scenario Outline, in modo da poter effettuare i test in modalità data driven:

```
Scenario Outline: Search a prof
```

Dovrà essere innanzitutto implementata una precondizione tramite Given. In questo secondo esempio, la precondizione consiste nell'apertura del browser, quindi:

```
Given Open Browser
```

Successivamente, sarà necessario aprire la pagina di ricerca di docenti unina. Quindi, per definire un'azione da effettuare, sarà utilizzato il When:

```
When Open the "https://www.docenti.unina.it/#!/search"
```

Grazie all'utilizzo dello Scenario Outline, è possibile definire il link con un placeholder, indicato nella tabella Examples:

**When** Open the <link>

Nella funzionalità in esame, la ricerca è possibile grazie all'immissione del nome di un docente. Si andrà quindi a definire un ulteriore When e, anche in questo caso, è necessario un placeholder che consenta di definire il nome del professore da ricercare. Tale nome andrà inserito nella tabella Examples.

Similmente a quanto indicato nel precedente esempio, per rendere lo scenario formalmente più corretto, si utilizzerà un And al posto del When:

**And** insert <name>

Ora sarà necessario scrivere in Gherkin l'operazione successiva, che consiste nel click sul link della pagina professore trovato. Ciò si realizza nel seguente modo:

**And** click on teacher

L'ultima azione da effettuare è la prova dell'apertura di una pagina sul profilo di quel docente. Quindi, si andrà a scrivere quest'ultima azione all'interno del "blocco" When:

**And** click on <pagina>

Definite le precondizioni tramite Given e le azioni da eseguire all'interno del blocco When, non resta che definire le post condizioni. Come previsto dalla sintassi del linguaggio Gherkin, ciò si realizzerà mediante il Then che, per il caso in esame, dovrà fornire come risultato la corretta pagina ricercata. Quindi si avrà:

**Then** You have the <pagina>

Di seguito, è possibile osservare la tabella di Examples in cui sono definiti i dati ed i placeholders per i test. Tale tabella all'interno del .feature sarà collocata al di sotto dello Scenario Outline:

**Examples:**

link	name	pagina
"https://www.docenti.unina.it/#!/search"	"Porfirio Tramontana"	"Avvisi"
"https://www.docenti.unina.it/#!/search"	"Porfirio Tramontana"	"Riferimenti"
"https://www.docenti.unina.it/#!/search"	"Porfirio Tramontana"	"Curriculum"
"https://www.docenti.unina.it/#!/search"	"Porfirio Tramontana"	"Links"

## 2.2.2 Passo 2: Implementazione del codice

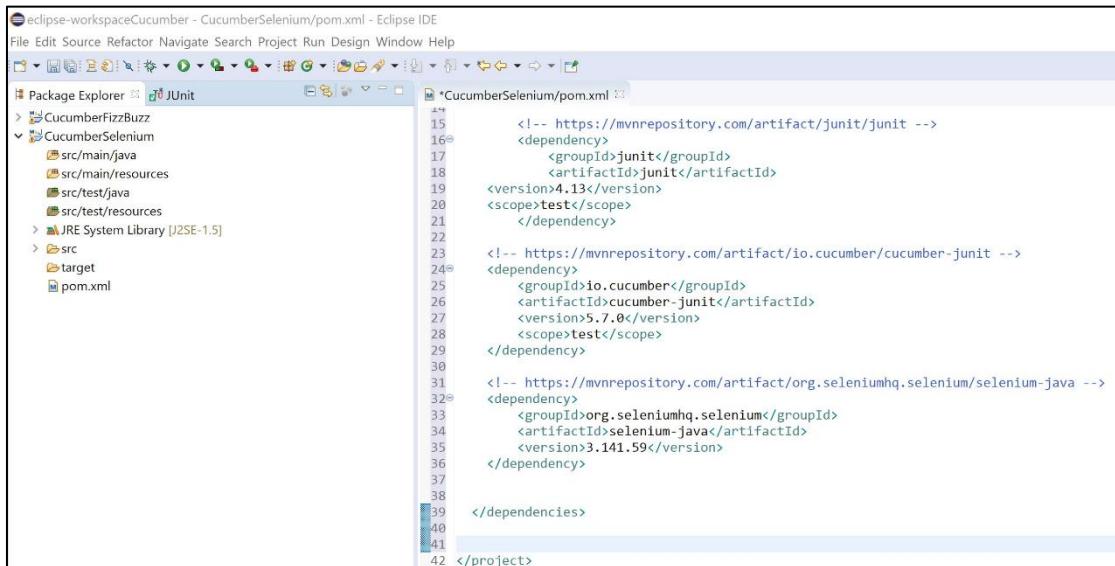
Nell'esempio in questione, il testing riguarda una web application e in particolare il sito [www.docenti.unina.it](http://www.docenti.unina.it), per cui non è necessario implementare codice Java aggiuntivo.

## 2.2.3 Passo 3: Creazione del progetto

La creazione del progetto in Eclipse IDE risulta pressoché identica, a meno di piccole aggiunte, a quella vista nell'esempio precedente. Le aggiunte necessarie in questo caso saranno le seguenti:

- Inserire nel file pom.xml anche la dipendenza da Selenium, trovata dalla repository ufficiale di Maven:

<https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java>



```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13</version>
    <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/io.cucumber/cucumber-junit -->
<dependency>
    <groupId>io.cucumber</groupId>
    <artifactId>cucumber-junit</artifactId>
    <version>5.7.0</version>
    <scope>test</scope>
</dependency>

<!-- https://mvnrepository.com/artifact/org.seleniumhq.selenium/selenium-java -->
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>3.141.59</version>
</dependency>

</dependencies>
</project>
```

Figura 35: File pom.xml

- Scaricare i driver del browser, coerenti con la versione presente sul dispositivo, necessari per poter interagire con la web application:

<https://www.selenium.dev/downloads/>

<https://sites.google.com/a/chromium.org/chromedriver/>

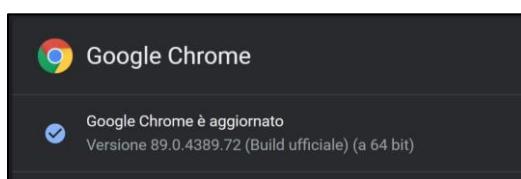


Figura 36: Schermata per valutare la versione di Chrome installata sul dispositivo

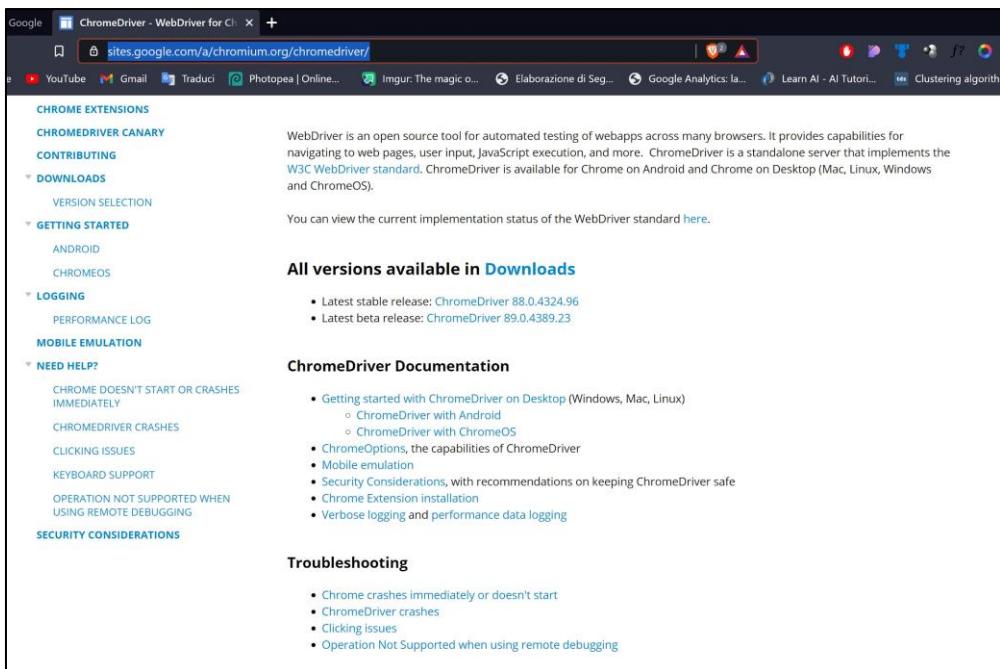


Figura 37: Schermata per il download del browser driver

- Esportare l'eseguibile appena scaricato ed inserirlo in una cartella appositamente creata, denominata drivers, nel seguente path del progetto src/test/resources/:

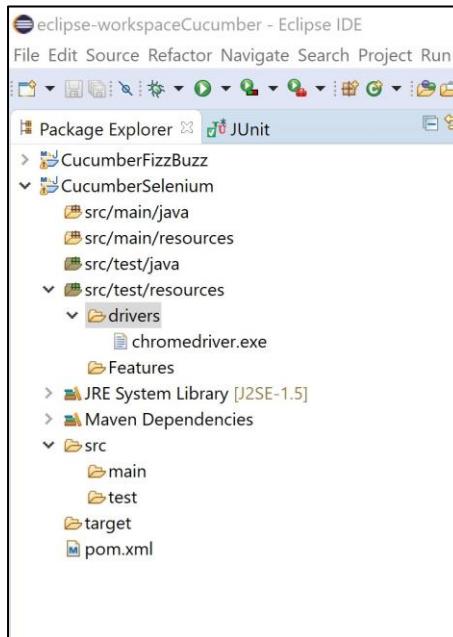


Figura 38: Cartella nella quale inserire il driver.exe

Impostato l'ambiente, sarà possibile lavorare come visto in precedenza. Si procederà, quindi, con la creazione della cartella features nel path src/test/resources, e si inserirà un file .feature in cui verrà scritto lo scenario creato nel Passo 1:

```

1@ Feature: Search on Docenti unina
2   As guest I want to search for a professor
3   so that i can access some sections
4
5
6@ Scenario Outline: Search a prof
7   Given Open Browser
8   When Open the <link>
9   And insert <name>
10  And click on teacher
11  And click on <pagina>
12  Then You have the <pagina>
13
14
15@ Examples:
16  | link | name | pagina |
17  | "https://www.docenti.unina.it#!/search" | "Porfirio Tramontana" | "Avvisi" |
18  | "https://www.docenti.unina.it#!/search" | "Porfirio Tramontana" | "Riferimenti" |
19  | "https://www.docenti.unina.it#!/search" | "Porfirio Tramontana" | "Curriculum" |
20  | "https://www.docenti.unina.it#!/search" | "Porfirio Tramontana" | "Links" |
21
22

```

**Figura 39: Scenario d'esempio**

## 2.2.4 Passo 4: Generazione degli Steps

Il passo di generazione degli steps risulta invariato rispetto a quanto visto nell'esempio precedente.

Infatti, è necessario effettuare il run dello scenario e copiare le firme dei metodi generati in un file .java appositamente creato per l'implementazione degli steps.

## 2.2.5 Passo 5: Implementazione degli Steps

Dato che l'obiettivo è il testing di un'interfaccia web, per implementare gli steps generati a partire dal feature file, saranno necessari i metodi contenuti nelle librerie di Selenium. Tali metodi permetteranno l'apertura del browser e l'interazione con le pagine web:

```
import org.openqa.selenium.*;
```

Operazione fondamentale da dover effettuare in questa tipologia di test, è la creazione di un oggetto Webdriver all'interno della classe. Esso consentirà la chiamata a metodi che permettono di interagire con il browser e, in questo modo, di navigare e testare il caso in esame.

```
WebDriver driver=null;
```

Da un'analisi dello Stepdef e considerando i punti cardine del feature file, si andranno a mostrare i principali punti utilizzando come traccia la sequenza Given-When-Then.

Una volta importate le librerie necessarie all'utilizzo dei metodi di Selenium ed aver creato il web driver, si procederà con la realizzazione dello step relativo al Given. La precondizione mostrata in precedenza richiedeva l'apertura del Browser per cui si avrà:

```
@Given("Open Browser")
public void open_Browser() {
    //Implementazione
}
```

Le prime righe di codice all'interno della funzione permettono di localizzare il driver del browser, che nell'esempio mostrato sarà Chrome:

```
String projectPath= System.getProperty("user.dir");
System.out.println("Project path is: "+projectPath);
System.setProperty("webdriver.chrome.driver",
projectPath+"/src/test/resources/drivers/chromedriver.exe");
```

Si assegnerà, poi, all'oggetto driver creato in precedenza una nuova finestra di Chrome:

```
driver= new ChromeDriver();
```

Successivamente, verranno inserite delle wait per far sì che la finestra si apra, mentre con la funzione window().maximize() si renderà la finestra appena aperta a tutto schermo:

```
driver.manage().timeouts().implicitlyWait(8, TimeUnit.SECONDS);
driver.manage().timeouts().pageLoadTimeout(8, TimeUnit.SECONDS);
driver.manage().window().maximize();
```

A questo punto, il test si sposta nell'ambito delle azioni da effettuare descritte tramite il blocco When per cui lo step successivo da effettuare è il primo When. Questa prima azione prevede che sia fornito un primo link da aprire:

```
@When("Open the {string}")
public void open_the(String link) {
    driver.get(link);
}
```

Con la funzione get verrà caricata la pagina web richiesta.

A questo punto, all'interno del webdriver la schermata sarà la seguente:

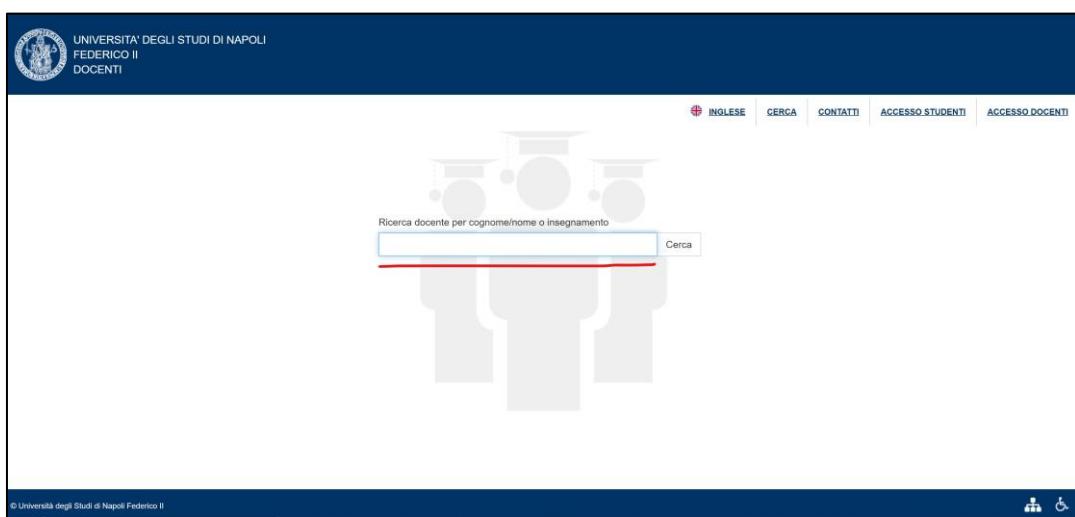
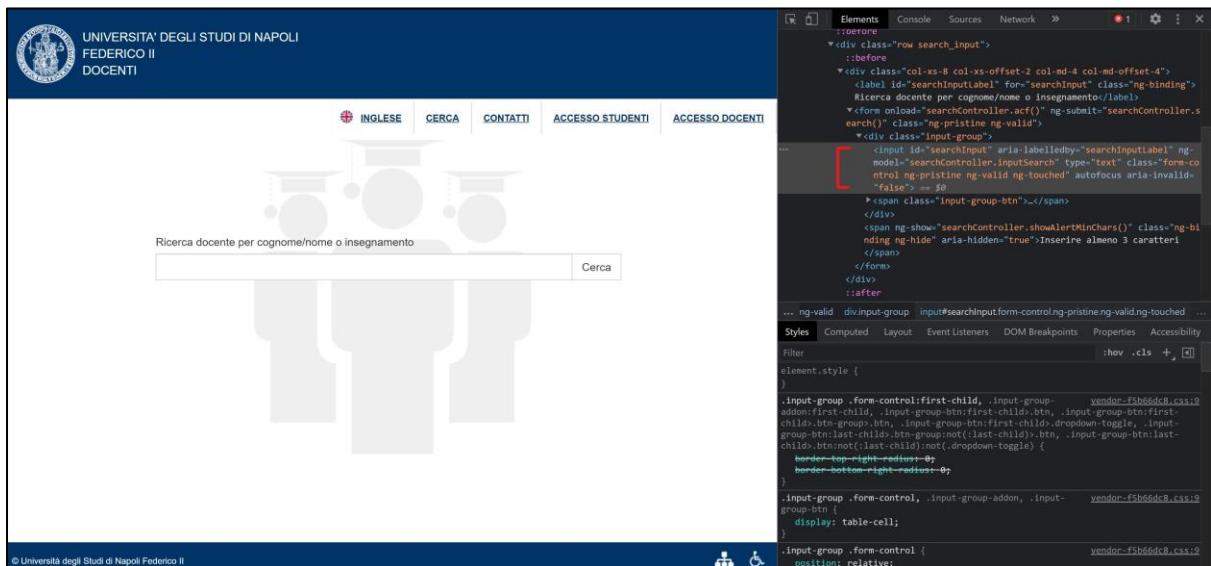


Figura 40: Schermata docenti.unina.it

Continuando ad analizzare la sequenza d'azioni, il passaggio successivo consiste nella ricerca di un docente dato il suo nome.

Per poter interagire con lo schermo, bisogna ispezionare la pagina web e cercare dei parametri "chiave", da poter utilizzare per identificare i vari widget presenti nella schermata:



**Figura 41: Ricerca dell'id del widget**

Nel caso in questione, sarà utilizzato l'id "searchInput" per identificare la barra di ricerca al centro della pagina ed il metodo della libreria di selenium findElement, come riportato nel seguente codice:

```
@When("insert {string}")
public void insert(String docente) {
    driver.findElement(By.id("searchInput")).clear();
    driver.findElement(By.id("searchInput")).sendKeys(docente);
    driver.findElement(By.id("searchInput")).sendKeys(Keys.ENTER);
}
```

Il metodo sendKeys consente di scrivere all'interno della casella di testo il nome del docente, mentre Key.ENTER di cliccare su invio. Il testo da inserire, corrispondente come già detto al nome del docente, sarà coincidente con quello fornito all'interno della tabella Examples nel feature file.

A seguito di ciò, saranno mostrati i risultati della ricerca all'interno del webdriver ed è necessario cliccare sul docente con la corrispondenza richiesta che, come mostrato in figura, presenta un id pari a "ffocus":

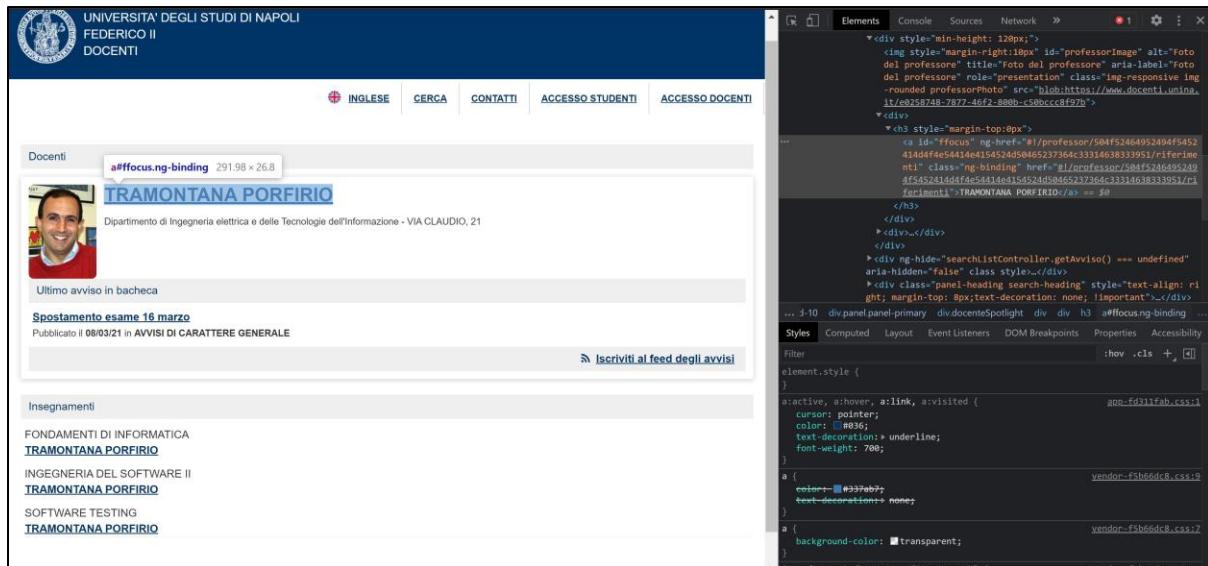


Figura 42: Ricerca id del nome del docente

Questa serie di operazioni all'interno dello step When si andrà a realizzare mediante la funzione `findElement` ed il metodo `click()`:

```
@When("click on teacher")
public void click_on_teacher() {
    driver.findElement(By.id("ffocus")).click();
}
```

L'ultima azione da eseguire sarà l'apertura di una delle sezioni del profilo ricercato. Per quest'ultimo When il ragionamento è il medesimo ma, in modo differente rispetto a quanto visto nei precedenti When, anziché ricercare l'elemento tramite l'id, a causa della variabilità in base alla pagina richiesta si utilizzerà il nome della pagina per aprire un link contenente quel nome specifico:

```
@When("click on {string}")
public void click_on(String pagina) {
    driver.findElement(By.linkText(pagina)).click();
}
```

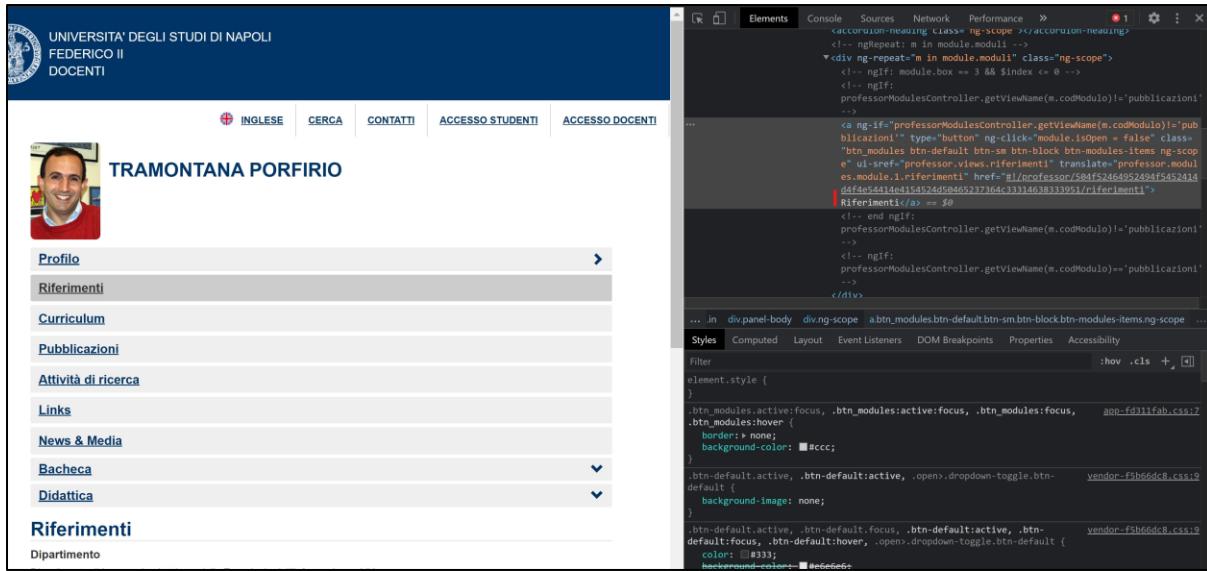


Figura 43: Ricerca parametro per il click sul widget

Ora non resta che verificare le post condizioni e ciò sarà possibile grazie all’ultimo step che è relativo al Then. In esso, si andrà a verificare se la pagina che è stata aperta corrisponde proprio a quella di interesse.

In particolare, viene effettuato un confronto (`equalsIgnoreCase(link)`) tra la stringa fornita in ingresso e quella attualmente vista nel browser, dopodiché è chiusa la schermata all’interno del webdriver di Chrome:

```
@Then("You have the {string}")
public void you_have_the_(String pagina) {
    String
link="https://www.docenti.unina.it#!/professor/504f52464952494f5452414d4f4e5
4414e4154524d50465237364c33314638333951/" +pagina;
    if(driver.getCurrentUrl().equalsIgnoreCase(link)){
        System.out.println("Ho aperto la pagina " + pagina);
    }
    else
        System.out.println("Errore nell'aprire la " + pagina);
    driver.close();
}
```

### Miglioramenti:

Anche in questo caso, così come nel precedente, le operazioni di apertura e chiusura del browser possono essere inserite all’interno degli Hooks Before e After, in quanto comuni a tutti gli scenari.

Di seguito è riportato il codice completo:

```

1 package StepDef;
2
3 import java.util.concurrent.TimeUnit;
4
5 import org.openqa.selenium.By;
6 import org.openqa.selenium.Keys;
7 import org.openqa.selenium.WebDriver;
8 import org.openqa.selenium.chrome.ChromeDriver;
9 import io.cucumber.java.After;
10 import io.cucumber.java.Before;
11 import io.cucumber.java.en.*;
12
13 public class StepDef {
14
15     WebDriver driver=null;
16
17     @Before()
18     public void setup() {
19         String projectPath= System.getProperty("user.dir");
20         System.out.println("Project path is: "+projectPath);
21         System.setProperty("webdriver.chrome.driver", projectPath+"/src/test/resources/drivers/chromedriver.exe");
22
23         driver= new ChromeDriver();
24         driver.manage().timeouts().implicitlyWait(8, TimeUnit.SECONDS);
25         driver.manage().timeouts().pageLoadTimeout(8, TimeUnit.SECONDS);
26         driver.manage().window().maximize();
27     }
28
29     @Given("Open Browser")
30     public void open_Browser() {
31     }
32
33     @When("Open the {string}")
34     public void open_the(String link) {
35         driver.get(link);
36     }

```

Figura 44: Codice completo StepDef.java (1/2)

```

35     driver.get(link);
36 }
37
38     @When("insert {string}")
39     public void insert(String docente) {
40         driver.findElement(By.id("searchInput")).clear();
41         driver.findElement(By.id("searchInput")).sendKeys(docente);
42         driver.findElement(By.id("searchInput")).sendKeys(Keys.ENTER);
43     }
44
45     @When("click on teacher")
46     public void click_on_teacher() {
47         driver.findElement(By.id("ffocus")).click();
48     }
49
50     @When("click on {string}")
51     public void click_on(String pagina) {
52         driver.findElement(By.linkText(pagina)).click();
53     }
54
55     @Then("You have the {string}")
56     public void you_have_the_(String pagina) {
57         String link="https://www.docenti.unina.it/#!/professor/504f52464952494f5452414d4f4e54414e4154524d50465237364c33314638";
58         if(driver.getCurrentUrl().equalsIgnoreCase(link)){
59             System.out.println("Ho aperto la pagina " + pagina);
60         }
61         else
62             System.out.println("Errore nell'aprire la " + pagina);
63     }
64
65
66     @After()
67     public void teardown() {
68         driver.close();
69     }
70 }

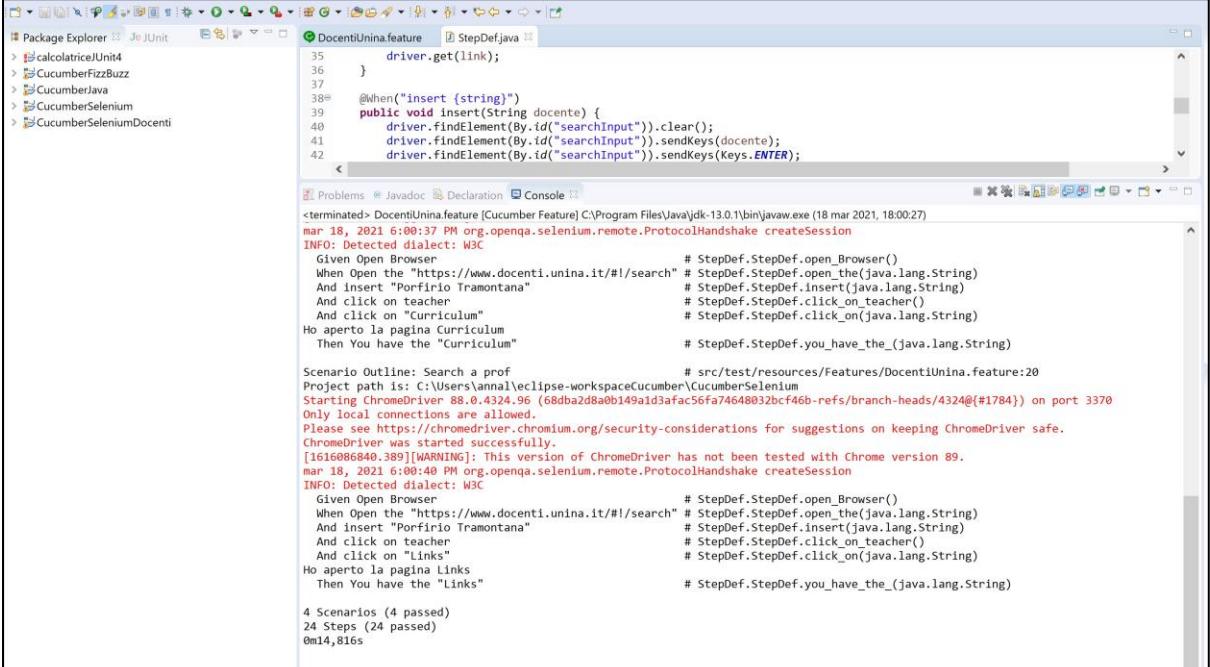
```

Figura 45: Codice completo StepDef.java (2/2)

## 2.2.6 Passo 6: Esecuzione dei test e valutazione

Come già visto in precedenza, l'esecuzione dei test può essere realizzata tramite il Runner di Cucumber, con l'unica differenza che durante l'esecuzione degli scenari verrà aperto ed utilizzato un webdriver.

Nel caso in esame, i test risulteranno tutti passati:



The screenshot shows the Eclipse IDE interface with the 'DocentiUnina.feature' file open in the center. The code defines two scenarios: one for inserting a string and another for searching for 'Porfirio Tramontana'. The execution log at the bottom shows the browser opening the URL, performing the search, and displaying the results. The log also includes Selenium protocol messages and a warning about ChromeDriver version 89.

```
driver.get(link);
}
}

@When("insert {string}")
public void insert(String docente) {
    driver.findElement(By.id("searchInput")).clear();
    driver.findElement(By.id("searchInput")).sendKeys(docente);
    driver.findElement(By.id("searchInput")).sendKeys(Keys.ENTER);
}

Scenario Outline: Search a prof
Given Open Browser
When Open the "https://www.docenti.unina.it/#/search" # StepDef.StepDef.open_the_(java.lang.String)
And insert "Porfirio Tramontana" # StepDef.StepDef.insert_(java.lang.String)
And click on teacher # StepDef.StepDef.click_on_teacher_()
And click on "Curriculum" # StepDef.StepDef.click_on_(java.lang.String)
Ho aperto la pagina Curriculum
Then You have the "Curriculum" # StepDef.StepDef.you_have_the_(java.lang.String)

Scenario Outline: Search a prof
Given Open Browser
When Open the "https://www.docenti.unina.it/#/search" # StepDef.StepDef.open_the_(java.lang.String)
And insert "Porfirio Tramontana" # StepDef.StepDef.insert_(java.lang.String)
And click on teacher # StepDef.StepDef.click_on_teacher_()
And click on "Links" # StepDef.StepDef.click_on_(java.lang.String)
Ho aperto la pagina Links
Then You have the "Links" # StepDef.StepDef.you_have_the_(java.lang.String)

4 Scenarios (4 passed)
24 Steps (24 passed)
0m14,816s
```

Figura 46: Risultati dell'esecuzione degli scenari

## 2.3 Esempio 3: Cucumber in Android Studio

Oltre ad Eclipse IDE, nel prosieguo dell'elaborato sarà mostrato l'utilizzo del framework Cucumber anche all'interno dell'ambiente di sviluppo Android Studio.

Dunque, si andrà a vedere in che modo è possibile testare un'applicazione Android grazie al paradigma BDD e, in modo simile a quanto avvenuto con il case-study dell'esempio 2 in cui era necessario un ulteriore framework a supporto di Cucumber, ossia Selenium, in questo caso il framework di supporto sarà Android Espresso.

Al fine di approfondire la comprensione dell'utilizzo degli strumenti e del paradigma, si testerà anche in ambito Android un'applicazione che realizza una calcolatrice che effettua le quattro operazioni aritmetiche basilari (somma, differenza, moltiplicazione e divisione).

### 2.3.1 Passo 1: Produzione degli Scenari

La metodologia seguita è simile a quanto già indicato nei precedenti paragrafi, per cui la prima fase consiste nello scrivere, a seguito di un confronto con il cliente e di una discussione interna tra PO, developer e tester, degli scenari per il testing dei requisiti richiesti. Nell'esempio considerato, i requisiti richiederanno un test delle quattro operazioni elementari tra due numeri.

Anche in questa sezione, così come nella prima, sarà utilizzato il costrutto Scenario Outline per definire dei test Data Driven.

**Feature:** Arithmetic Expressions

This feature provides a range of scenarios corresponding to the intended external behavior of arithmetic expressions on integers.

**Scenario Outline:** Operation of two numbers

```
Given Open the calculator
When Click the first number button <number>
And Click the first operation button "<op>" 
And Click the second number button <number1>
And Click the second operation button "<op1>" 
Then The result is number <result>
```

**Examples:**

number	op	number1	op1	result
7	+	4	=	"11.0"
5	+	5	=	"10.0"
1	+	2	=	"3.0"
7	-	4	=	"3.0"
5	-	5	=	"0.0"
1	-	2	=	"-1.0"
0	*	2	=	"0.0"
9	*	2	=	"18.0"
4	/	2	=	"2.0"
9	/	3	=	"3.0"

Osservando il codice, si noterà che vi è una differenza rispetto ai casi precedenti, in quanto gli steps relativi ai When non consistono nel fornire in ingresso un numero o l'operazione da effettuare, bensì nel cliccare i tasti corrispondenti. Ciò è dovuto all'interfaccia grafica dell'applicazione ed è proprio tale interfaccia a giustificare l'utilizzo del framework di supporto Android Expresso.

L'uso di tale framework, in abbinamento con Cucumber, porta però alla nascita di alcune limitazioni. Infatti, mentre nello Scenario descritto nell'esempio 1 era possibile scrivere all'interno della tabella Examples ogni numero intero, anche a due o tre cifre, in questo caso sarebbe necessario definire tanti When quanti sono i tasti da cliccare, ossia uno per ogni cifra.

**Miglioramenti:**

Una volta compresa la metodologia dagli esempi precedenti, è noto che gli scenari saranno tradotti in steps. In particolare, il nome di ogni step è collegato alle istruzioni riportate dopo le parole Given, When e Then. Conoscendo tale relazione, è possibile effettuare un accorgimento nella scrittura dello scenario.

In particolare, non sarà più necessario chiamare con un nome differente i due steps When relativi all'inserimento dei due numeri o al click dei due bottoni relativi all'operazione da effettuare, ma possono essere chiamati entrambi con i seguenti nomi generici:

```
When Click the number button <number>
And Click the operation button "<op>"
```

Ciò risulterà molto utile nelle fasi di generazione ed implementazione degli steps, in quanto è necessario scrivere soltanto due steps per tutti i when dello scenario.

### 2.3.2 Passo 2: Implementazione del codice

Nell'esempio mostrato è stato utilizzato il codice di un'applicazione calcolatrice disponibile presso il seguente repository Github:

<https://crunchify.com/how-to-create-simple-calculator-android-app-using-android-studio/>

Si ricorda al lettore che in qualsiasi progetto il codice sorgente della MainActivity è inserito nella cartella che presenta il seguente percorso:

*app\src\main\java\com\NOME\_PROGETTO.*

### 2.3.3 Passo 3: Creazione del progetto

Come per Eclipse IDE, anche in questo caso è necessario installare il plugin di Cucumber per poter successivamente eseguire i test.

All'interno di Android Studio, per installare i plugin di Cucumber, bisogna recarsi in File->Settings- >Plugin ed installare le librerie necessarie mostrate nella seguente figura:

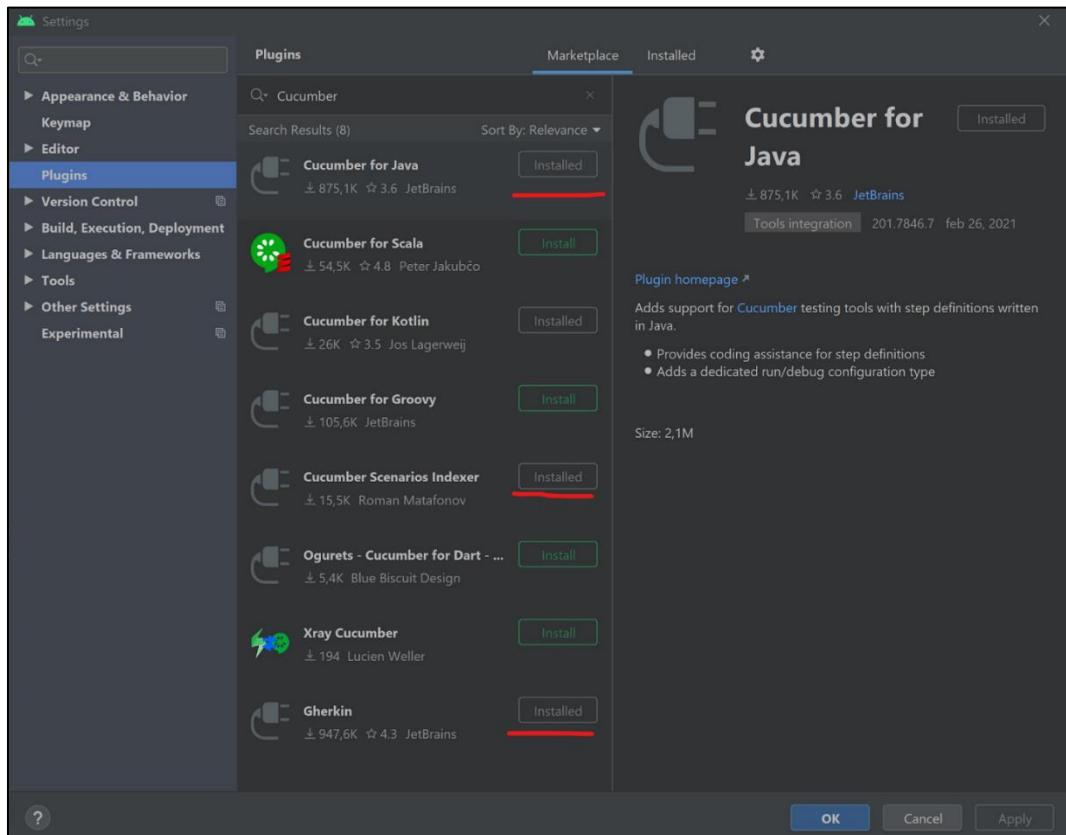


Figura 47: Schermata per installare i plugin necessari

Installato Cucumber, si procede con l'apertura dell'applicazione da testare andando ad inserire le dipendenze, non più in un pom.xml, ma all'interno del file build.gradle. Ciò è dettato dal fatto che AndroidStudio utilizza gradle come builder.

Le dipendenze da aggiungere all'interno del file build.gradle sono fornite dalle seguenti righe di codice:

```
androidTestImplementation 'info.cukes:cucumber-android:1.2.5'  
androidTestImplementation 'info.cukes:cucumber-  
picocontainer:1.2.5'  
androidTestImplementation  
'com.android.support.test.espresso:espresso-core:3.0.2'  
androidTestImplementation'com.android.support.test:runner:1.0.2'  
  
androidTestImplementation 'com.android.support:support-  
annotations:28.0.0'  
androidTestImplementation 'com.android.support.test:rules:1.0.2'
```

Tali righe consentono l'integrazione di entrambi i framework menzionati precedentemente e quindi: Cucumber ed Android Expresso.

Come già visto in Eclipse, è necessaria una cartella Features per inserire gli scenari. Per crearla, è stata cambiata la modalità di visione del progetto da App a Project ed è stata aggiunta nel percorso app/src/androidTest/ una directory assets ed al suo interno un'ulteriore cartella chiamata features:

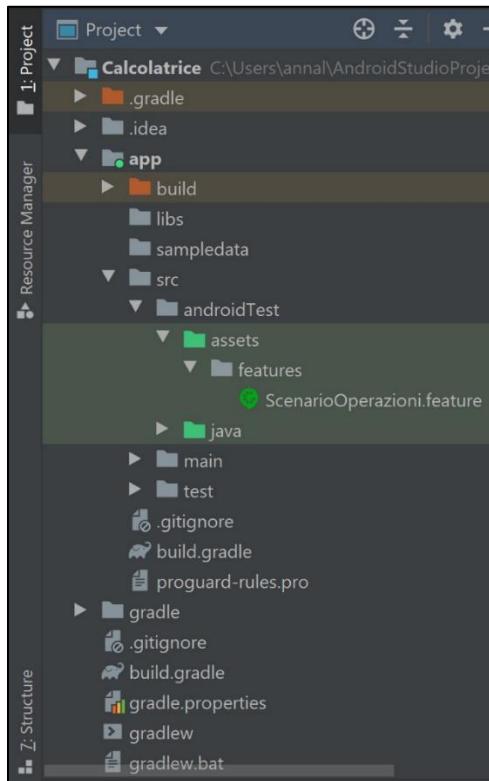
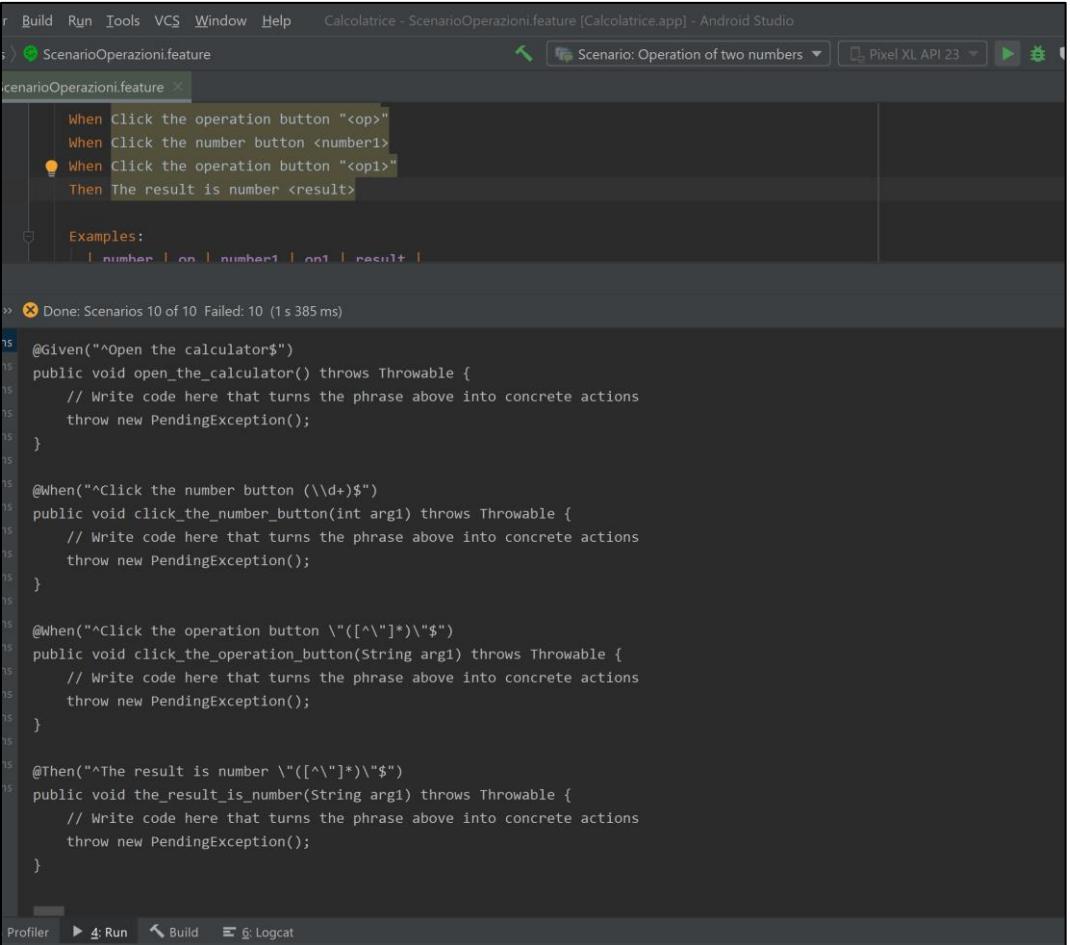


Figura 48: Cartelle del progetto

All'interno della cartella features, è necessario inserire un nuovo file .feature contenente lo scenario di interesse.

### 2.3.4 Passo 4: Generazione degli Steps

Scritto il feature file ed impostato l'ambiente di sviluppo per il progetto, sarà ora possibile procedere alla creazione degli Step partendo dallo scenario. Anche con Android Studio, eseguendo lo scenario, sarà possibile generare automaticamente le firme dei metodi da inserire nel file StepDef.java:



The screenshot shows the Android Studio interface with the feature file "ScenarioOperazioni.feature" open. The file contains a Gherkin scenario with four steps: "When Click the operation button <op>", "When Click the number button <number1>", "When Click the operation button <op1>", and "Then The result is number <result>". Below the scenario, there is an "Examples:" section with placeholder values. The Java code generated for these steps is shown below the feature file. The code includes annotations like @Given, @When, and @Then, and corresponding methods that throw PendingException(). The Java code is as follows:

```
15 @Given("^Open the calculator$")
16 public void open_the_calculator() throws Throwable {
17     // Write code here that turns the phrase above into concrete actions
18     throw new PendingException();
19 }
20
21 @When("^Click the number button (\\d+)$")
22 public void click_the_number_button(int arg1) throws Throwable {
23     // Write code here that turns the phrase above into concrete actions
24     throw new PendingException();
25 }
26
27 @When("^Click the operation button \"([^\"]*)\"$")
28 public void click_the_operation_button(String arg1) throws Throwable {
29     // Write code here that turns the phrase above into concrete actions
30     throw new PendingException();
31 }
32
33 @Then("^The result is number \"([^\"]*)\"$")
34 public void the_result_is_number(String arg1) throws Throwable {
35     // Write code here that turns the phrase above into concrete actions
36     throw new PendingException();
37 }
```

Figura 49: Step generati automaticamente dal run dello scenario

Effettuando il run dello scenario, verranno generati gli steps, i quali possono essere copiati ed incollati all'interno di una classe Java che sarà situata nella cartella com/NOME\_PROGETTO (androidTest).

Nel caso in esempio, è stato creato un package chiamato Test ed è stata inserita la classe StepDef.java:

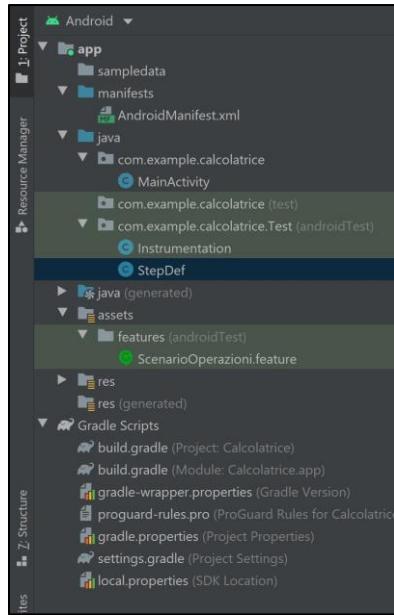


Figura 50: Cartelle del progetto

### 2.3.5 Passo 5: Implementazione degli Steps

A questo punto, bisogna implementare i singoli step. Come per Selenium, anche in questo caso saranno necessari dei metodi messi a disposizione da Android Espresso per poter interagire con l'applicazione.

La prima operazione da effettuare, così come è stato fatto per la calcolatrice e per il driver di Selenium, consiste della creazione di una activitytest rule per poter avviare l'applicazione:

```
@Rule
public ActivityTestRule<MainActivity> activityTestRule = new
ActivityTestRule<>(MainActivity.class);
private Activity activity;
```

Creata tale Activity, è possibile procedere con l'implementazione.

Si inizierà con l'implementare il primo step, il Given, che contiene la precondizione necessaria per eseguire lo scenario di test. Nel caso in esame la precondizione consiste nell'aprire l'applicazione calcolatrice.

```
@Given("^Open the calculator$")
public void open_the_calculator() {
    activityTestRule.launchActivity(new Intent());
    activity = activityTestRule.getActivity();
}
```

Successivamente si può passare al blocco di azioni descritte tramite When. La prima operazione da effettuare consiste nel cliccare sul tasto corrispondente al numero di interesse. Tale azione, resa possibile grazie all'associazione tra numero fornito in ingresso ed il tasto corrispondente, è realizzata mediante il costrutto switch case, mentre, per effettuare il click sull'interfaccia grafica, è possibile utilizzare il comando:

```
-    onView(withId(R.id.button)).perform(click());
```

OnView(withId()) permette di trovare nell'interfaccia il bottone con l'id univoco, mentre il metodo perform consente di definire un'azione da effettuare su quell'elemento che, nel caso in questione, è il click.

Tale comando è disponibile grazie alle seguenti librerie di Android Espresso che devono essere importate nel file:

```
import static androidx.test.espresso.Espresso.onView;
import static androidx.test.espresso.action.ViewActions.click;
```

Qui di seguito è riportato il codice relativo al @When con la soluzione appena descritta

```
@When("^Click the number button (\\d+)$")
public void click_the_number_button(int number) {
    switch (number) {
        case 0:
            onView(withId(R.id.button0)).perform(click());
            break;
        case 1:
            onView(withId(R.id.button1)).perform(click());
            break;
        case 2:
            onView(withId(R.id.button2)).perform(click());
            break;
        case 3:
            onView(withId(R.id.button3)).perform(click());
            break;
        case 4:
            onView(withId(R.id.button4)).perform(click());
            break;
        case 5:
            onView(withId(R.id.button5)).perform(click());
            break;
        case 6:
            onView(withId(R.id.button6)).perform(click());
            break;
        case 7:
            onView(withId(R.id.button7)).perform(click());
            break;
        case 8:
            onView(withId(R.id.button8)).perform(click());
            break;
        case 9:
            onView(withId(R.id.button9)).perform(click());
            break;
    }
}
```

Con lo stesso ragionamento, è possibile implementare il When successivo, in cui si clicca il tasto relativo all'operazione da effettuare.

Si riporta di seguito il codice:

```

@When("^Click the operation button \"([^\"]*)\"$")
public void click_the_operation_button(String op) {
    switch (op) {
        case "+":
            onView(withId(R.id.buttonadd)).perform(click());
            break;
        case "-":
            onView(withId(R.id.buttonsub)).perform(click());
            break;
        case "*":
            onView(withId(R.id.buttonmul)).perform(click());
            break;
        case "/":
            onView(withId(R.id.buttondiv)).perform(click());
            break;
        case "=":
            onView(withId(R.id.buttononeql)).perform(click());
            break;
    }
}

```

Infine, restano da valutare le post condizioni. Come visto negli esempi precedenti ciò è legato al Then dello scenario in esame. Dunque, si potrà ora valutare il risultato dell'operazione e procedere con la chiusura della calcolatrice.

```

@Then("The result is number \"([^\"]*)\"$")
public void theResultIsNumber(String result) {
    EditText display = (EditText) activity.findViewById(R.id.edt1);
    String displayed_result = display.getText().toString();
    assertEquals(result, displayed_result);
    activityTestRule.finishActivity();
}

```

La valutazione del risultato è resa possibile da un assertEquals. In particolare, si verifica se il risultato dell'operazione elementare tra i due numeri coincide con quello specificato nello scenario corrispondente. Per la chiusura della calcolatrice, si fa ricorso invece al metodo finishActivity(), che chiude l'applicazione.

### **Miglioramenti:**

Anche in questo caso, è possibile utilizzare degli Hooks per definire le operazioni da effettuare prima e dopo l'esecuzione di ogni scenario.

In particolare, basterà spostare le operazioni di apertura e chiusura dell'Activity, contenute nei passaggi di Given e Then, all'interno degli Hooks Before ed After, come di seguito riportato:

```

@Before()
public void setup(){
    activityTestRule.launchActivity(new Intent());
    activity = activityTestRule.getActivity();
}

```

```

@After()
public void teardown(){
    activityTestRule.finishActivity();
}

```

### 2.3.6 Passo 6: Esecuzione dei test e valutazione

Per eseguire i test in Android Studio, bisogna effettuare dei passaggi aggiuntivi rispetto ad Eclipse IDE. Ciò accade in quanto frameworks che permettono di interagire con l’interfaccia grafica come Android Espresso, necessitano di un file denominato Instrumentation, che deve essere inserito nel package di Test e che consente di “comandare” l’applicazione sotto test.

Il codice della classe contenuta nel file Instrumentation.java sarà il seguente:

```

package com.example.calcolatrice.Test;
import androidx.test.runner.*;
import cucumber.api.android.CucumberInstrumentationCore;
import android.os.Bundle;

public class Instrumentation extends MonitoringInstrumentation {

    private final CucumberInstrumentationCore mInstrumentationCore = new
CucumberInstrumentationCore(this);

    @Override
    public void onCreate(Bundle arguments) {
        super.onCreate(arguments);

        mInstrumentationCore.create(arguments);
        start();
    }

    @Override
    public void onStart() {
        super.onStart();

        waitForIdleSync();
        mInstrumentationCore.start();
    }
}

```

Inoltre, dato che si utilizza il Runner di Cucumber, è necessario definire le Cucumber Options ed aggiungere i contenuti sotto riportati:

```

import cucumber.api.CucumberOptions;

@CucumberOptions(
    features = "features",
    glue = "com.example.calcolatrice.Test",
    monochrome = true,
    plugin= {"pretty"})
)

```

Successivamente, è necessario inserire le seguenti righe di codice nel file build.gradle all'interno delle parentesi graffe Default Config, a loro volta contenute in android, modificandole a seconda del progetto. Ciò serve per identificare il path in cui si trova il file di Instrumentation.

```
testApplicationId "com.example.calcolatrice.Test"
testInstrumentationRunner "com.example.calcolatrice.Test.Instrumentation"
```

Inoltre, è necessario inserire anche all'interno delle parentesi graffe *android* le seguenti righe di codice, in modo da fornire il path dell'assets contenente i file .feature:

```
sourceSets {
    androidTest {
        assets.srcDirs = [ 'src/androidTest/assets' ]
    }
}
```

L'ultimo passaggio da effettuare è la configurazione del Runner.

Per fare ciò, bisogna cliccare in alto a destra in app->Edit Configurations, selezionare il tasto “+” in alto a sinistra e scegliere l'opzione Android Instrumented Tests.

Dopodiché, è necessario dare un nome allo scenario, inserendo il Module di interesse e cliccando sul tasto ok:

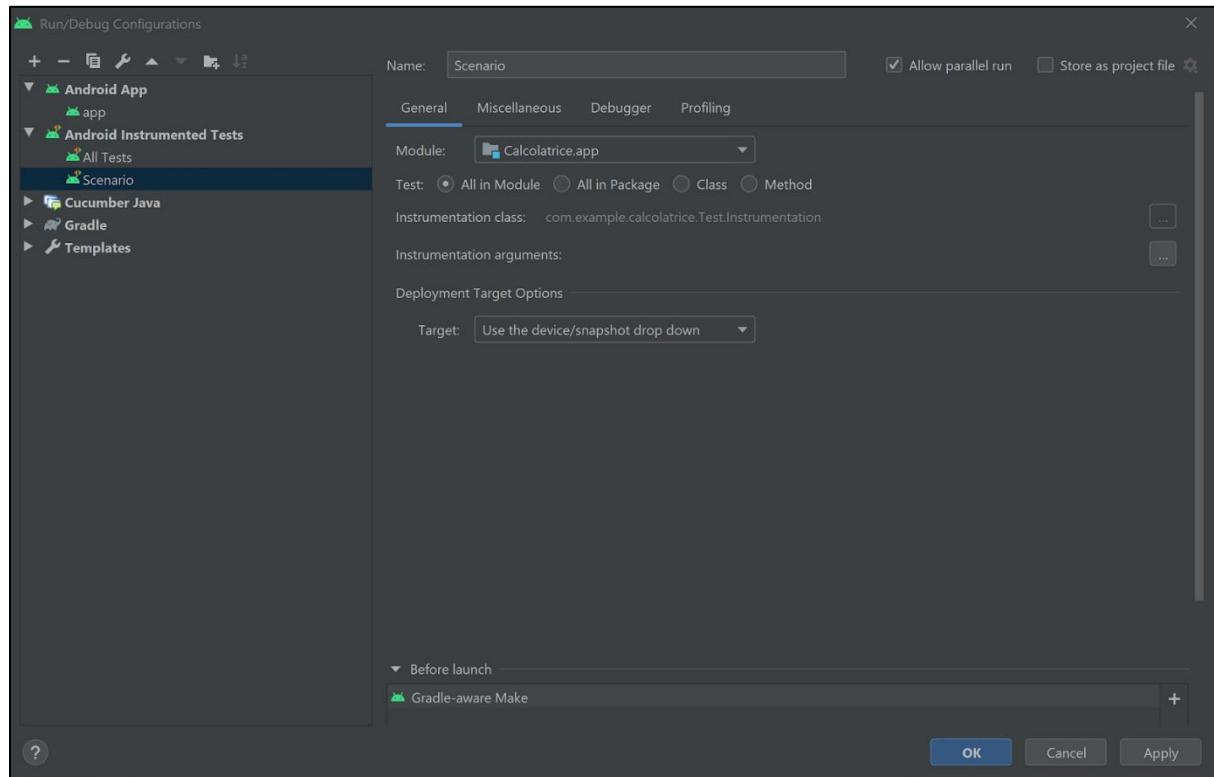
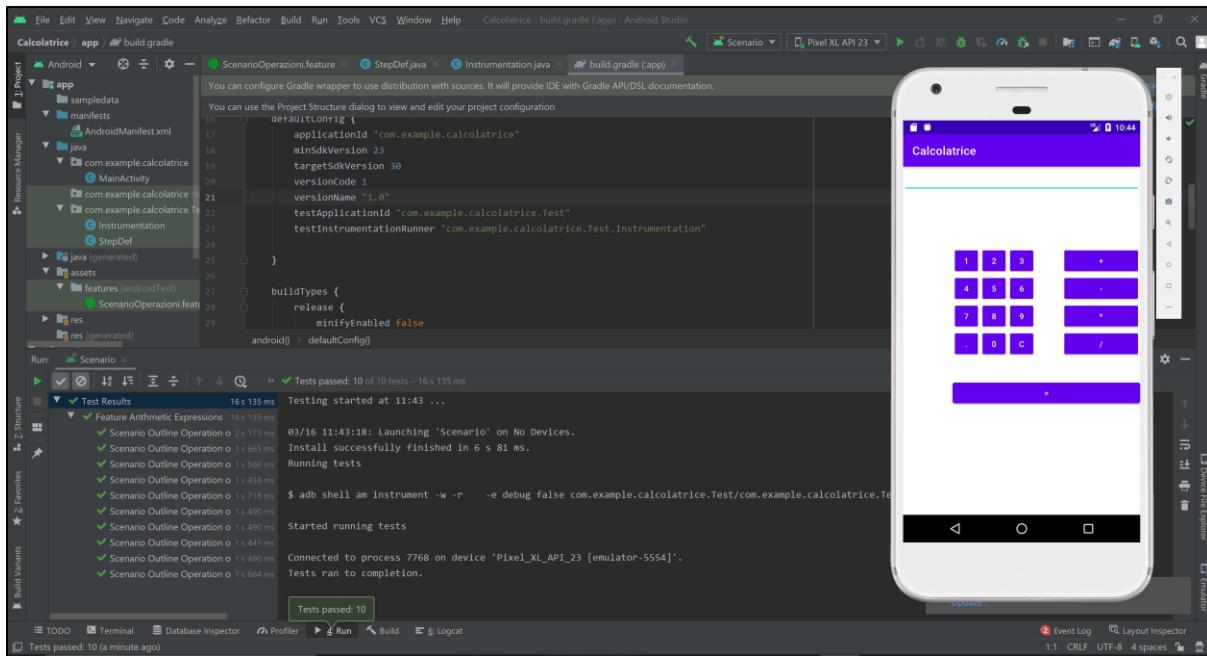


Figura 51: Schemata di Run/Debug Configurations

Una volta effettuati tali passaggi, è possibile cliccare sul tasto Run e, a questo punto, saranno eseguiti i test dell'applicazione sul device o sull'emulatore:



**Figura 52: Risultati ottenuti dall'esecuzione degli scenari**

Nella barra inferiore, selezionando l'opzione Run, è possibile visualizzare i test falliti ed i test passati.

Nel caso mostrato in questo elaborato i test hanno tutti esito positivo, in quanto il risultato fornito dalla applicazione coincide con quello definito nel Then dello scenario. Ciò significa anche che tutte le operazioni di apertura, chiusura ed interazione con l'interfaccia grafica sono andate a buon fine.

Esistono, però, delle casistiche di errore che avrebbero fatto fallire uno o tutti gli scenari. Ad esempio, nel caso in cui non fosse stata inserita la precondizione in Before, non sarebbe stato possibile aprire l'applicazione calcolatrice e tutti gli scenari non avrebbero potuto proseguire, come rappresentato in figura:

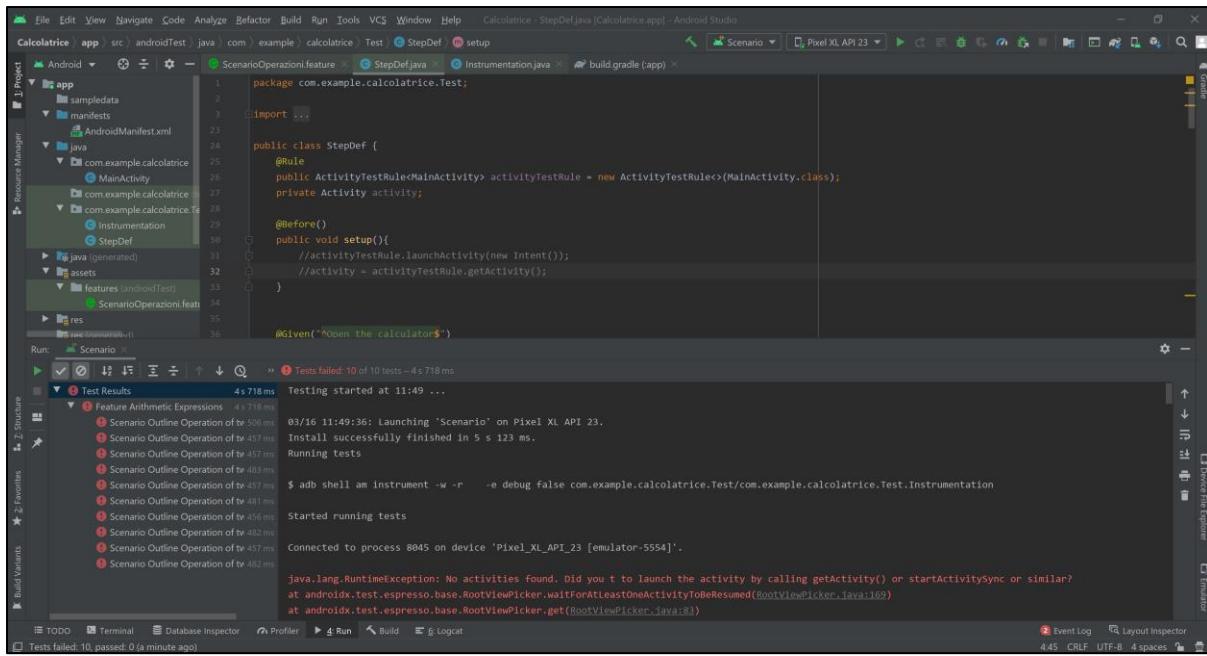


Figura 53: Risultati ottenuti dall'esecuzione degli scenari in caso di fallimento

## 2.4 Tabella Gherkin

Keyword	Scopo
Feature	Ogni file Gherkin inizia con la parola chiave "Feature", che non influenza il comportamento dei test su Cucumber, ma definisce il nome o la funzionalità che si desidera testare. Infatti, esso è subito indicato a seguito della parola chiave "Feature", mentre nelle righe successive ne è riportata una descrizione più dettagliata. In Gherkin, una "Feature" deve essere seguita da una di queste Keywords: <ul style="list-style-type: none"> <li>• Background;</li> <li>• Scenario;</li> <li>• Scenario Outline.</li> </ul>
Background	Consente di definire gli step comuni a tutti i test del file Feature. Il background viene eseguito prima di ciascuno degli scenari, ma dopo un qualsiasi "Before" Hooks.

Scenario	Ogni “Scenario” è un esempio concreto di come dovrebbe comportarsi il sistema in una situazione particolare. Serve ad esprimere effettivamente il comportamento di interesse ed ogni funzionalità ne contiene diversi. Come per “Feature”, la descrizione di “Scenario” può essere definita su più righe.
Scenario Outline	Consente ad uno scenario di utilizzare la propria tabella dati per i parametri, posizionando dei placeholders all'interno della struttura dello scenario racchiusi in parentesi angolari (<..>). Nella tabella "Examples", vengono indicati i valori da sostituire per ogni placeholder. Non è importante l'ordine in cui sono inseriti i placeholders nella tabella, a patto che essi siano coerenti con la propria intestazione.
Given	Permette di definire le precondizioni ed il contesto iniziale. È utile per portare il prodotto sotto test nello stato desiderato.
When	Definisce le interazioni ( <i>Actions</i> ) che saranno effettuate all'interno dello scenario.
Then	Definisce i risultati delle azioni eseguite nello step <i>When</i> .
And (o “But”)	Combina più affermazioni “Given”, “When” o “Then” insieme.

*	Può essere utilizzato come alternativa (meno dettagliata) per "Given", "When", "Then", oppure per parole chiave "And" e "But".
#	Definisce la riga di commento.
"""" ... """"	Definiscono le stringhe di documentazione
@tagname	<p>Uno o più tag possono essere utilizzati prima degli elementi "Scenario", "Feature" o "Scenario Outline".</p> <p>Se un tag viene utilizzato prima della keyword "Feature", esso sarà applicato a tutti gli scenari all'interno della funzionalità.</p> <p>I tag possono essere utilizzati per le seguenti motivazioni:</p> <ol style="list-style-type: none"> <li>1. <i>Documentazione</i>: si utilizzano i tag per allegare un'etichetta a determinati scenari, ad esempio per etichettarli con un determinato ID di progetto ad uno strumento di gestione;</li> <li>2. <i>Filtraggio</i>: Cucumber consente di utilizzare i tag come filtro per selezionare specifici scenari da eseguire o su cui generare report;</li> <li>3. <i>Hook</i>: i tag, inseriti negli Hook, consentono di definire scenari particolari, in modo da eseguire operazioni prima o dopo il blocco di codice.</li> </ol>

Tabella 4: Keywords in Gherkin

## 2.5 Cucumber: il Parser Gherkin

All'interno del plugin Cucumber che, come già visto nei paragrafi precedenti, può essere installato in numerosi ambienti di sviluppo, esistono dei meccanismi che permettono la trasformazione “automatica” degli scenari in Steps. A tale scopo, sono utilizzati il *parser* ed il compilatore, Gherkin, chiamato come il linguaggio.

Ad oggi, gli stessi parser e compilatore sono implementati per le seguenti piattaforme:

- .NET;
- Java;
- JavaScript;
- Ruby;
- Go;
- Python;
- Objective-C;
- Perl.

Inoltre, attualmente, la ricerca sta cercando di espandere il numero delle piattaforme che supportano Gherkin, anche se l'utilizzo di Gherkin su riga di comando (CLI) lo rende comunque supportato da piattaforme per le quali non è ancora disponibile.

Il Gherkin utilizzato su riga di comando legge gli scenari (file .feature) e genera AST (*Abstract Syntax Tree*) (vedi paragrafo [2.5.2]) e Pickles (vedi paragrafo [2.5.3]), ma è possibile anche ricevere file in output di tipo Newline Delimited JSON.

Per adoperare Gherkin su riga di comando [], è necessario installarlo ed eseguire il run dei file all'interno della repository:

```
gherkin testdata/**/*.feature
```

I file NDJSON risultanti non sono di facile lettura, dunque, per renderli più semplici da leggere, è possibile “tradurli” effettuando una pipe ed utilizzando il programma jq [[jq \(stedolan.github.io\)](#)]

```
gherkin testdata/**/*.feature | jq
```

### 2.5.1 Architettura del Gherkin-parser

L'architettura del Gherkin-parser è definita dalle seguenti fasi:

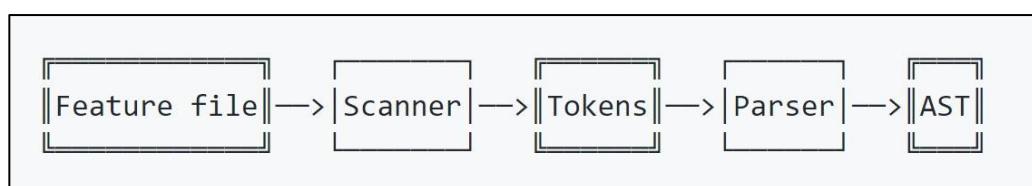


Figura 6: Architettura del Gherkin-parser

Lo scanner legge il file .feature e crea un token per ogni riga e l'insieme di quest'ultimi è passato al parser, che crea un AST [paragrafo 2.5.2].

Inizialmente, lo scanner vede l'intestazione `#language` e cerca dinamicamente le parole del linguaggio Gherkin associate, presenti nel file dati `gherkin-languages.json`.

Lo scanner è scritto a mano, ma il parser è originato dal generatore di parser *Berp*<sup>3</sup> durante il processo di compilazione.

*Berp* prende in input un file grammaticale (`gherkin.berp`) ed un file di modello (`gherkin-X.razor`) per generare il linguaggio X:

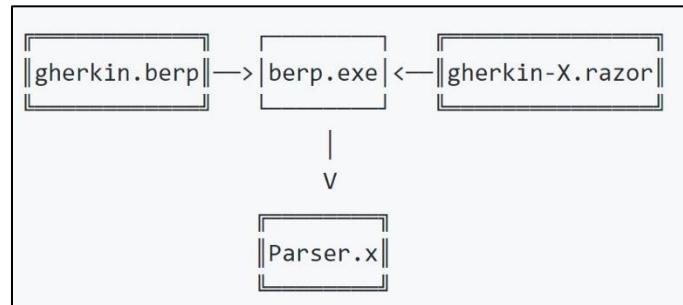


Figura 7: Architettura di Berp

## 2.5.2 Abstract Syntax Tree (AST)

In informatica, un albero di sintassi astratto (AST), o semplicemente albero di sintassi, è una rappresentazione ad albero della struttura sintattica astratta del codice sorgente scritto in un linguaggio di programmazione. [19]

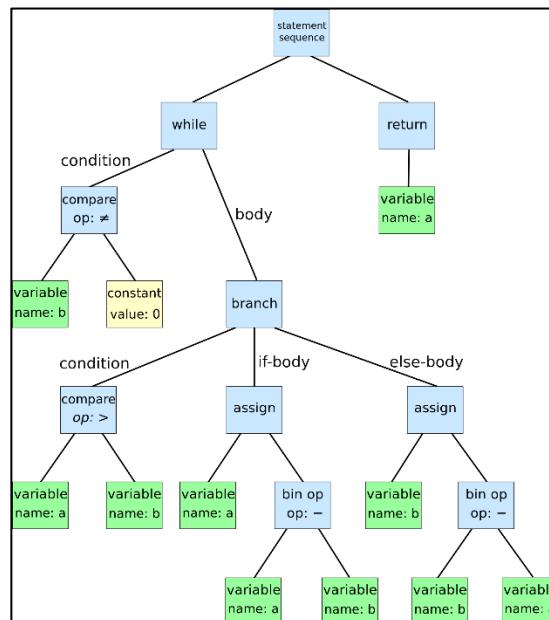
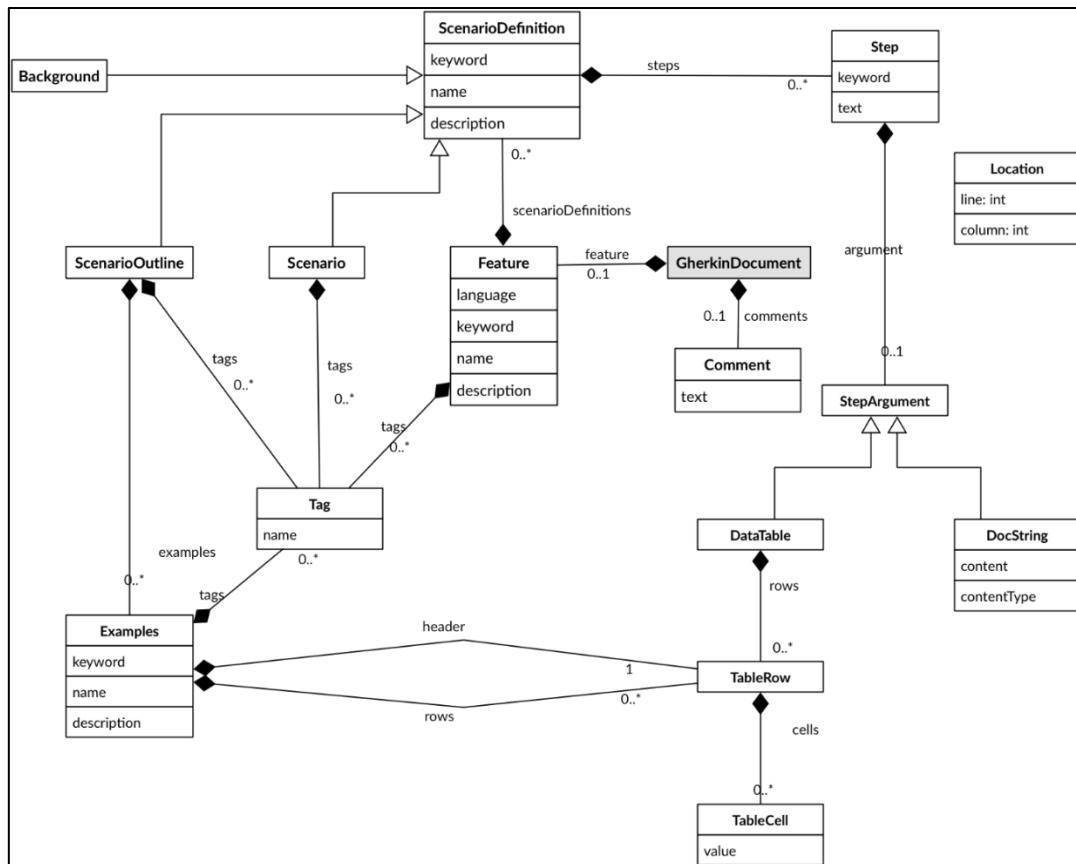


Figura 56: Esempio di AST

<sup>3</sup> Berp è un generatore di parser cross-language, che supporta le lingue senza regole di tokenizzazione esplicite come Gherkin.

In Gherkin, l'AST prodotto dal parser può essere descritto con il seguente diagramma delle classi:



**Figura 57: AST prodotto dal parser**

Ogni classe rappresenta un nodo nell'AST ed ogni nodo ha una posizione che descrive il numero di riga ed il numero di colonna nel file di input.

Tutti i campi sui nodi sono stringhe (ad eccezione di `Location.line` e `Location.column`) e l'implementazione è realizzata in oggetti semplici senza comportamento, ossia comprendenti soltanto dati.

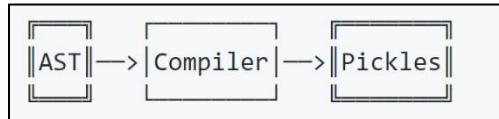
In fase di implementazione bisogna decidere se utilizzare classi o solo raccolte di base, ma, in ogni caso, l'AST deve avere una rappresentazione JSON (utilizzata per i test).

Ogni nodo nella rappresentazione JSON ha anche una proprietà `type`, con il nome del tipo di nodo.

È possibile visualizzare alcuni esempi nel repository GitHub di Cucumber:  
<https://github.com/cucumber/cucumber/tree/master/gherkin/testdata/good>

### 2.5.3 Pickles

L'AST non è utilizzabile direttamente per l'esecuzione da parte di Cucumber, in quanto esso ha bisogno di un'ulteriore elaborazione, dato che il file AST viene compilato in una forma più semplice chiamata *Pickles*:



**Figura 8: Step per passare da Parser Gherkin a Cucumber**

Ciò permette di disaccoppiare Gherkin da Cucumber, in modo sia da semplificare la logica interna di Cucumber, sia da rendere Gherkin disponibile anche in formati alternativi.

Ogni “Scenario” viene compilato in un Pickle, ossia una lista di PickleStep derivanti da quelli dello Scenario.

Ogni “Background” e riga degli “Exemples” associata ad uno “Scenario Outline” diventerà Pickles.

Anche i tag @, sono compilati in Pickle ereditando dagli elementi del file AST.

Esempio:

```

@a
Feature:
@b @c
Scenario Outline:
  Given <x>

  Examples:
  | x |
  | y |

@d @e
Scenario Outline:
  Given <m>

@f
Examples:
| m |
| n |

```

Utilizzando Gherkin da riga di comando, è possibile compilare questo file in oggetti Pickles come segue:

```
gherkin testdata/good/readme_example.feature --no-source --no-ast | jq
```

Output:

```
{
  "type": "pickle",
  "uri": "testdata/good/readme_example.feature",
  "pickle": {
    "name": "",
    "steps": [
      {
        "text": "y",
        "arguments": [],
        "locations": [
          {
            "line": 1,
            "column": 1
          }
        ]
      }
    ]
  }
}
```

```

        "line": 9,
        "column": 7
    },
    {
        "line": 5,
        "column": 11
    }
]
],
"tags": [
    {
        "name": "@a",
        "location": {
            "line": 1,
            "column": 1
        }
    },
    {
        "name": "@b",
        "location": {
            "line": 3,
            "column": 3
        }
    },
    {
        "name": "@c",
        "location": {
            "line": 3,
            "column": 6
        }
    }
],
"locations": [
    {
        "line": 9,
        "column": 7
    },
    {
        "line": 4,
        "column": 3
    }
]
}
{
    "type": "pickle",
    "uri": "testdata/good/readme_example.feature",
    "pickle": {
        "name": "",
        "steps": [
            {
                "text": "n",
                "arguments": [],
                "locations": [
                    {
                        "line": 18,
                        "column": 7
                    },
                    {
                        "line": 19,
                        "column": 7
                    }
                ]
            }
        ]
    }
}

```

```

        {
          "line": 13,
          "column": 11
        }
      ]
    }
  ],
  "tags": [
    {
      "name": "@a",
      "location": {
        "line": 1,
        "column": 1
      }
    },
    {
      "name": "@d",
      "location": {
        "line": 11,
        "column": 3
      }
    },
    {
      "name": "@e",
      "location": {
        "line": 11,
        "column": 6
      }
    },
    {
      "name": "@f",
      "location": {
        "line": 15,
        "column": 5
      }
    }
  ],
  "locations": [
    {
      "line": 18,
      "column": 7
    },
    {
      "line": 12,
      "column": 3
    }
  ]
}

```

Ogni evento Pickle contiene il percorso alla fonte originale e ciò è utile per generare report ed analisi dello stack, quando uno scenario fallisce.

Inoltre, Cucumber trasforma ulteriormente questo elenco di oggetti Pickles in un elenco di oggetti TestCase, i quali si collegano al codice utente come Hooks e Step Definitions.

## **Capitolo 3: Cucumber Studio**

Nelle sezioni precedenti si è mostrato come utilizzare Gherkin e Cucumber all'interno del ciclo di sviluppo e ne sono state analizzate l'applicazione e l'uso in diversi contesti ed ambienti di sviluppo. Al fine di ottimizzare i processi di gestione del ciclo di sviluppo, ad oggi esistono strumenti proprietari che semplificano la scrittura e la successiva implementazione di test nell'ottica BDD guidata dal linguaggio Gherkin. Tali strumenti all'interno di team di sviluppo favoriscono maggiormente la collaborazione nella progettazione di feature files e di test di accettazione. Inoltre, essi offrono anche un ottimo punto di partenza nell'automazione della produzione di StepDefinitions e dell'esecuzione di test tramite servizi come Git e TravisCI.

Tra tali strumenti si è preso in esame CucumberStudio [20], il quale fa parte della famiglia di software realizzata da SmartBear, ed è il risultato dell'integrazione tra il framework Cucumber e la piattaforma proprietaria di SmartBear, Hiptest.

Il prodotto è a pagamento, ma la compagnia mette a disposizione di qualsiasi utente un periodo di prova di quattordici giorni, riscattabile a seguito di un'iscrizione con il proprio account GitHub o Google.

Oltre a supportare pienamente la metodologia BDD e la sintassi Gherkin, tale servizio permette una serie di ulteriori vantaggi all'interno di un team di sviluppo orientato alla filosofia agile ed al DevOps. Infatti, grazie a CucumberStudio, è possibile:

- Gestire la documentazione del progetto a partire dai feature file. Tale documentazione evolverà in maniera dinamica;
- Mantenere un buon supporto per il management della fase di test, secondo le metodologie Agili;
- Mettere a disposizione dell'utente la generazione automatica per script di testing per più di 20 linguaggi e frameworks, sfruttando un publisher open-source;
- Collegare il proprio account GitHub in modo che i repository siano aggiornati in maniera dinamica. Tale funzionalità torna molto utile a fronte di un team ampio, ritrovando le varie modifiche direttamente all'interno del repository;
- Supportare la Continuous Integration, grazie alla cooperazione con strumenti come Travis CI, ossia un servizio di integrazione continua utilizzato per testare progetti che si trovano su repository come GitHub.

La presentazione di tale strumento si focalizzerà su aspetti essenziali, come il supporto alla metodologia BDD ed alla sintassi del linguaggio Gherkin, mostrando anche come sia possibile usufruire di servizi di generazione automatica del codice per un dato linguaggio/framework.

### 3.1 Creazione di un nuovo progetto

Per la creazione di un nuovo progetto in CucumberStudio, si ha la possibilità di selezionare due tipologie differenti, come riportato in figura:

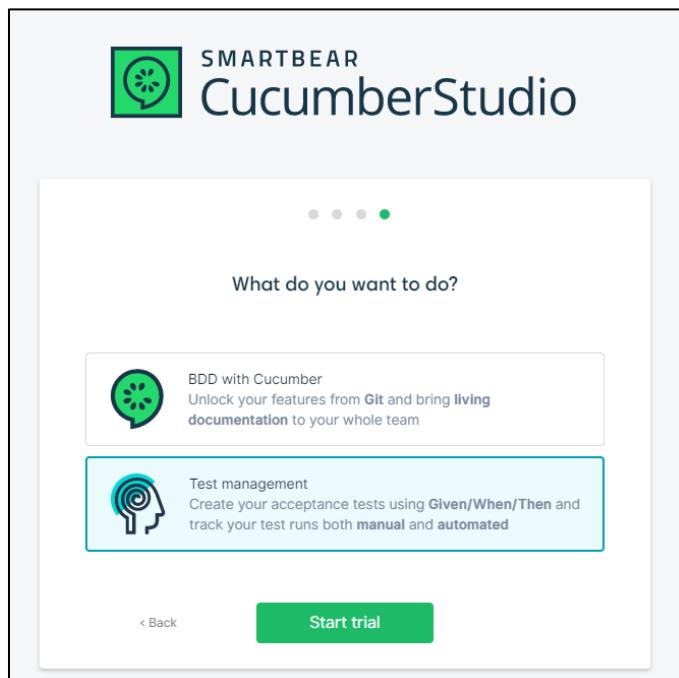


Figura 59: : Schermata finale di creazione progetto

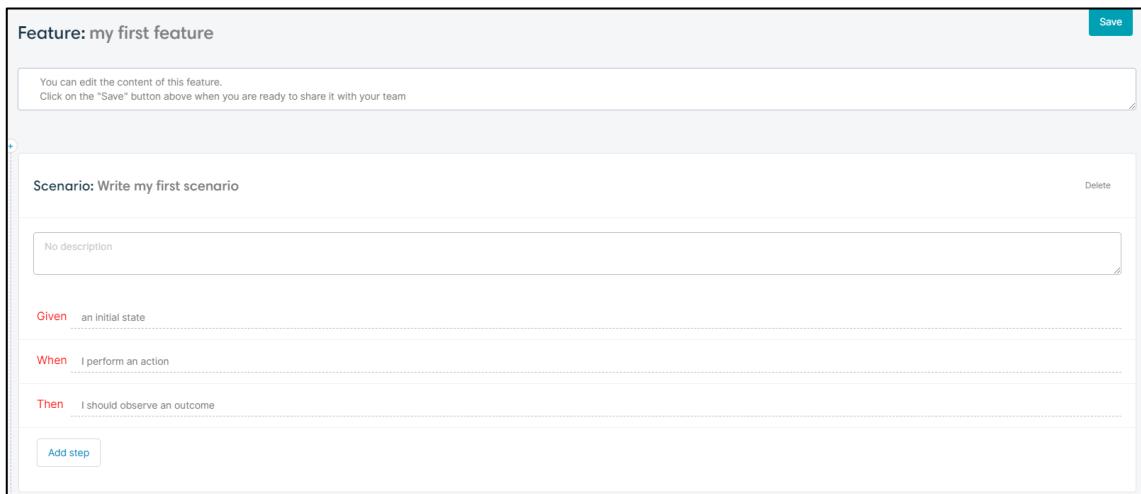
Nell'esaminare tale strumento, si illustreranno entrambe le modalità e si vedrà come creare un Feature File che segue la sintassi Gherkin, e la principale differenza tra le due modalità.

### 3.2 BDD con Cucumber

Selezionando tale modalità per il progetto, si avrà una dashboard molto semplice che vede principalmente la presenza di tre soli sottomenù:

- Uno per i Features files (che sarà analizzato nel dettaglio nel seguito);
- Uno per degli example mapping, che consentono di creare un esempio di utilizzo del software, definendo una storia che può essere arricchita con regole, esempi e quesiti;
- Un pannello da cui gestire il progetto su cui si sta lavorando. Da questa sezione è possibile aggiungere dei collaboratori.

La prima volta in cui si accede alla sezione feature si avrà di fronte una schermata simile a quella mostrata nella seguente figura:



**Figura 60:** Schermata creazione Feature

Una volta creata la propria feature, si può procedere ad effettuare un salvataggio e ciò richiederà l’accesso a GitHub, dove all’interno del repository selezionato verrà salvato il file appena redatto.

A questo punto su CucumberStudio il feature file comparirà come in fig[61], mentre su GitHub sarà salvato all’interno di un Branch “Feature”, come possibile osservare dalla fig[62]:

Scenario:

### Subtracting two integer numbers

- ➊ Given I initialize a calculator
- ➋ When an integer operation '-'
- ➌ And I provide a first number 7
- ➍ And I provide a second number 5
- ➎ Then The operation evaluates to 2
- ➏ And

**Figura 61:**

```
v 6 ██████████ features/SubtractingTwoIntegerNumbers.feature □
...
...   ... @@ -0,0 +1,6 @@
+ Feature:Subtracting two integer numbers
+ Given I initialize a calculator
+ When an integer operation '-'
+ And I provide a first number 7
+ And I provide a second number 5
+ Then The operation evaluates to 2 ⊖
```

**Figura 62**

Tale funzionalità permette una scrittura rapida degli scenari in Gherkin ed inoltre, prima di poter editare uno scenario, verrà richiesto da CucumberStudio di aggiornare il Feature con l’ultima versione presente sul repository:



**Figura 63:**

### 3.3 Test management

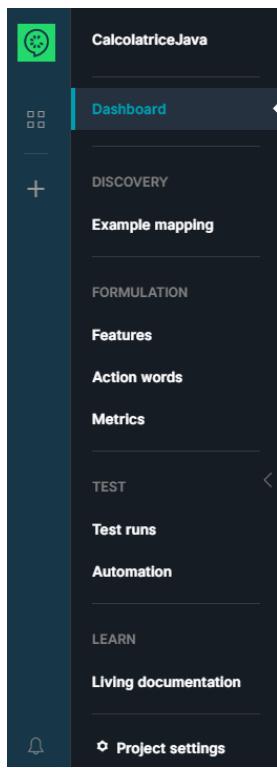


Figura 64: Barra laterale Dashboard

Selezionando invece la seconda tipologia di progetto, si avranno a disposizione una serie di funzionalità aggiuntive che si andranno ad analizzare.

In particolare, osservando l'immagine precedente, si può notare come sia presente la possibilità di utilizzare l'example mapping per accedere ai settings del progetto.

Inoltre, le funzionalità aggiunte riguarderanno anche la sezione di feature, che sarà strutturata in maniera completamente diversa rispetto a prima.

Completamente inedite rispetto alla prima tipologia di progetto analizzata in precedenza sono le voci:

- *Action Words*: identificano un blocco di step all'interno di un test case, permettendo di eseguire manutenzione e refactoring in modo semplice e veloce. Una action word è definita da: Nome, descrizione, tags, parameters, definition e usedby;
- *Metrics*: in tale sezione si può avere una visuale grafica di una serie di metriche, definite in maniera preventiva, per i propri test;
- *Test Runs*: questa sezione consente di gestire le esecuzioni dei test. Un test run è una sorta di istantanea del progetto, che include tutti i test o un loro sottoinsieme, tramite esso è possibile creare un ramo del progetto in cui memorizzare le definizioni correnti dei test. Nel momento in cui si crea un test run, si ha la possibilità di decidere se includere tutti i test o selezionare una parte di essi oppure creare un test run vuoto che potrà essere adibito per integrare i risultati esterni dei test;
- *Automation*: tale sezione permette di ottenere in maniera automatica la definizione degli steps per una coppia framework/linguaggio. È possibile anche integrare tali steps con un repository GitHub per operare in maniera totalmente automatica.
- *Living Documentation*: in questa sezione è possibile condividere all'interno del team le definizioni di features e scenari. Ciò consente di avere un supporto comune durante le riunioni con il lato di business e il lato di sviluppo. Per usufruire di tale funzionalità all'interno del progetto deve essere abilitata la modalità BDD, già attiva di default in Cucumber studio.

### 3.4 Creazione di Features in CucumberStudio

Dalla pagina Feature si accede alla cartella principale di testing del progetto. All'interno di questa sezione, è possibile navigare tra le varie cartelle, che contengono i Feature Files che descrivono gli scenari.

In genere è buona prassi collocare all'interno della stessa sotto cartella gli scenari che descrivono lo stesso comportamento, ma con casistiche differenti.

Per esemplificare l'analisi di CucumberStudio, si è optato per la creazione di un nuovo progetto che vada a simulare il primo esempio visto nella sezione precedente. Quindi, è stato creato un progetto che si prefigge l'obiettivo di strutturare dei feature file con i rispettivi test di una calcolatrice, che realizza le operazioni aritmetiche basilari (somma, prodotto, differenza e divisione). Tale caso sarà analizzato attraverso la realizzazione di due file features differenti.

Nel primo feature file si descriverà uno scenario per ognuna delle operazioni aritmetiche, come mostrato in Figura 60.

The screenshot shows a scenario outline in Gherkin format:

```
Given I Initialize the calculator
When an integer operation '+'
and I provide a first number 4
and I provide a second number 5
Then the operation evaluates to 9
```

Below the outline is a button labeled "Add step".

Figura 65: Scenario in CucumberStudio

In particolare, si è proceduto all'edit della sezione Test Steps. È interessante notare come, man mano che vengano descritti gli scenari, lo strumento supporti l'utente proponendo dei suggerimenti per completare la scrittura dello scenario:

The screenshot shows a partially completed scenario outline:

```
Given I Initialize the calculator
When an integer operation '-'
```

Below the outline, there is a suggestion box containing the text "And I provi".

At the bottom, there are sections for "ACTIONS" and "SUGGESTIONS". The "ACTIONS" section contains a "Create action word" button. The "SUGGESTIONS" section lists two items:

- I provide a second number 5 (used once)
- I provide a first number 4 (used once)

Figura 66: Suggerimenti del linguaggio Gherkin

Nel secondo file feature, si mostrerà come utilizzare la funzione Datatable per costruire uno scenario parametrizzato. In questo caso, si scriverà un unico scenario che però varierà in base ai dati presenti nel datatable.

#	Dataset name	operatore	Primo_numero	Secondo_numero	Risultato	
1	somma	+	4	5	9	...
2	differenza	-	8	5	3	...
3	prodotto	*	7	2	14	...
4	rapporto	/	6	2	3	...

Figura 67: Datatable dello scenario

Test steps	
1	Given an integer operation = operatore
2	When I provide a first number = Primo_numero
3	and I provide a second number = Secondo_numero
4	Then the operation evaluates to = Risultato

Figura 68: Step risultanti dallo scenario

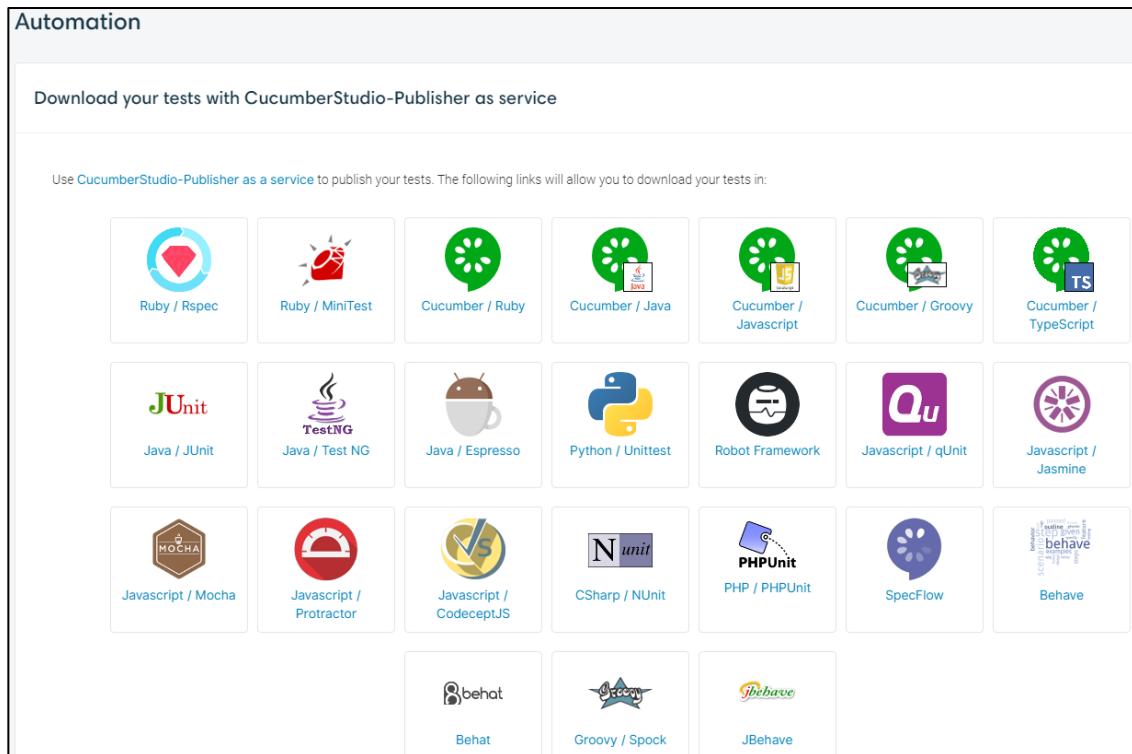
Passando alla modalità “View Test”, si potrà navigare tra i vari test creati. Nell’esempio considerato, tramite l’opzione Datatable, sarà descritto un unico scenario di test che comprende tutte e quattro le operazioni elementari. Si mostreranno di seguito i due casi per somma e prodotto.

1	Given	an integer operation +	1	Given	an integer operation *
2	When	I provide a first number 4	2	When	I provide a first number 7
3	And	I provide a second number 5	3	And	I provide a second number 2
4	Then	the operation evaluates to 9	4	Then	the operation evaluates to 14

Figura 69: Caso di somma e prodotto

Grazie a CucumberStudio, è possibile generare direttamente il codice per lo StepDef in diversi linguaggi e framework tramite la sezione di Automation. Recandosi in tale sezione

sarà possibile generare automaticamente il codice, disponendo la seguente varietà di combinazioni framework/linguaggio:



**Figura 70: Pagina di download del codice**

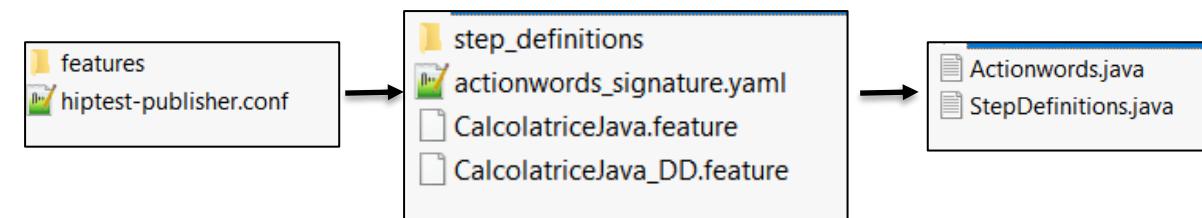
Dall'immagine precedente, con riferimento all'implementazione della calcolatrice elementare, sarà selezionata l'opzione Cucumber/Java.

Una volta selezionato il framework/linguaggio, sarà automaticamente scaricata una cartella contenente:

- Cartella features;
- File di configurazione per Hiptest-Publisher<sup>4</sup>.

Inoltre, aprendo la cartella features, al suo interno si troveranno:

- I file features scritti precedentemente tramite CucumberStudio;
- Un actionwords\_signature.yml che sarà usato insieme al file di configurazione per Hiptest-Publisher;
- Una cartella StepDefinition che conterrà il file StepDef.java che consentirà di eseguire i test come visto nel precedente capitolo.



**Figura 71: Cartelle scaricate**

test.

Dunque, tale strumento software consente una semplificazione in fase di creazione ed organizzazione del lavoro all'interno del team di sviluppo.

Alternativamente al download ed integrazione nel progetto in Java, tramite Hiptest Publisher è possibile integrare il tutto all'interno del proprio progetto su Github ed automatizzare anche l'esecuzione dei casi di test.

Qualora si stia lavorando ad un progetto tramite un servizio di repository e versioning, è possibile eseguire in automatico i test tramite una gem Ruby, messa a disposizione da Hiptest, per generare ed eseguire automaticamente il codice dei test. Per fare ciò bisognerà installare sulla propria macchina Ruby e, successivamente, tramite il comando “**gem install hiptest-publisher**”, anche il servizio hiptest-publisher.

Installato hiptest publisher, aggiungendo il configuration file<sup>5</sup> al proprio progetto o al proprio repository su git, sarà possibile eseguire tutti i test con il comando “**hiptest-publisher -c**” o selezionare un particolare test con il comando “**hiptest-publisher -c –test-run-id=ID\_TEST**”.

Si descriverà il funzionamento di quanto detto attraverso il progetto di esempio messo a disposizione da CucumberStudio.

### 3.4 Esempio Coffee Machine

La prima operazione da effettuare consiste nell'aggiungere il progetto d'esempio alla propria project list. Per farlo, si seleziona l'icona “+” della navbar laterale sinistra e si clicca su Coffee Machine, che verrà così aggiunto alla lista.

A questo punto, è necessario recarsi sulla repository GitHub:

<https://github.com/hiptest/hiptest-publisher-samples>, in cui sono contenuti i collegamenti a repository del progetto per vari linguaggi e frameworks.

---

<sup>5</sup> Il Configuration File segue un particolare template fornito da CucumberStudio a seconda del linguaggio.

È possibile selezionare la coppia linguaggio/framework dai seguenti link:

Ruby					
	Rspec		Minitest		Cucumber
Repository	<a href="#">hps-ruby-rspec</a>		<a href="#">hps-ruby-minitest</a>		<a href="#">hps-cucumber-ruby</a>
Build status	<span>build passing</span>		<span>build passing</span>		<span>build passing</span>
Hiptest	<a href="#">CucumberStudio Ruby / rspec</a>		<a href="#">CucumberStudio Ruby / minitest</a>		<a href="#">CucumberStudio Cucumber / Ruby</a>

Javascript					
	qUnit	Jasmine	Mocha	Protractor	Cucumber-js
Repository	<a href="#">hps-javascript-qunit</a>	<a href="#">hps-javascript-jasmine</a>	<a href="#">hps-javascript-mocha</a>	<a href="#">hps-protractor</a>	<a href="#">hps-cucumber-javascript</a>
Build status	<span>build passing</span>	<span>build passing</span>	<span>build passing</span>	<span>build passing</span>	<span>build passing</span>
Hiptest	<a href="#">CucumberStudio Javascript / qUnit</a>	<a href="#">CucumberStudio Javascript / Jasmine</a>	<a href="#">CucumberStudio Javascript/Mocha</a>	<a href="#">CucumberStudio Javascript/Protractor</a>	<a href="#">CucumberStudio Cucumber/Javascript</a>

Java					
	JUnit	TestNG	Cucumber/Java	JBehave	
Repository	<a href="#">hps-java-junit</a>	<a href="#">hps-java-testng</a>	<a href="#">hps-cucumber-java</a>	<a href="#">hps-jbehave</a>	
Build status	<span>build error</span>	<span>build error</span>	<span>build passing</span>	<span>build error</span>	
Hiptest	<a href="#">CucumberStudio Java / JUnit</a>	<a href="#">CucumberStudio Java / TestNG</a>	<a href="#">CucumberStudio Cucumber/Java</a>	<a href="#">CucumberStudio JBehave</a>	

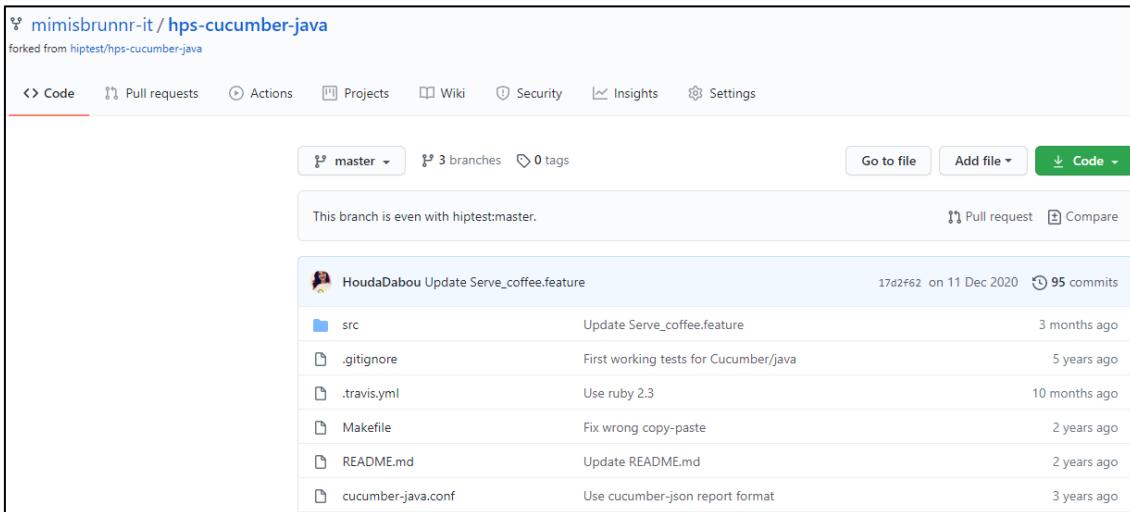
Figura 72: Schermata contenente i link alle repository specifiche

Per portare avanti l'esempio, si selezionerà il link nella colonna Cucumber/Java, anche se sul repository sono presenti collegamenti ad altri linguaggi oltre quelli in figura come: Python, PHP, C#, Groovy ed in ultima istanza una sottocategoria con tutti i repository basati su Gherkin:

All Gherkin-based							
	Cucumber/Ruby	Cucumber/Java	Specflow	Behave	Behat	Cucumber-js	Cucumber/C
Repository	<a href="#">hps-cucumber-ruby</a>	<a href="#">hps-cucumber-java</a>	<a href="#">hps-specflow</a>	<a href="#">hps-behave</a>	<a href="#">hps-behat</a>	<a href="#">hps-cucumber-javascript</a>	<a href="#">hps-cucumber-groovy</a>
Build status	<span>build passing</span>	<span>build passing</span>	<span>build error</span>	<span>build error</span>	<span>build error</span>	<span>build passing</span>	<span>build passing</span>
Hiptest	<a href="#">CucumberStudio Cucumber / Ruby</a>	<a href="#">CucumberStudio Cucumber/Java</a>	<a href="#">CucumberStudio Specflow</a>	<a href="#">CucumberStudio Behave</a>	<a href="#">CucumberStudio Behat</a>	<a href="#">CucumberStudio Cucumber/Javascript</a>	<a href="#">CucumberStudio Cucumber/Groovy</a>

Figure 73: Repository basate su Gherkin

Selezionato Cucumber/Java, si avrà a disposizione un ulteriore repository del progetto, da cui è necessario effettuare una fork, per avere così il repository in maniera privata sul proprio profilo.



**Figura 74: Repository Gitub Cucumber Java**

A questo punto, bisognerà utilizzare il comando “**git clone**” per ottenere in locale una copia del repository.

Una volta clonato il tutto in locale, sarà necessario aggiungere il proprio Test Code univoco, generato all’interno di CucumberStudio. Il Test Code è disponibile nel pannello relativo alle impostazioni del progetto e, in fondo alla pagina, sarà presente la sezione Test Code generation.

Tale codice sarà impostato come token all’interno del file di configurazione contenuto nel progetto appena clonato.

A questo punto, sarà possibile lanciare i test e visualizzare a video i risultati, tramite il comando “mvn test” da lanciare nel prompt dei comandi.

Una volta eseguiti i test, si avrà a video una metrica sugli scenari che sono andati a buon fine e quelli falliti:

```
15 Scenarios (1 failed, 14 passed)
61 Steps (1 failed, 60 passed)
0m0,099s
```

**Figura 75: Risultati del test**

Il comando per il run cambia da linguaggio a linguaggio, ed è riportato all’interno del file ReadMe del progetto di cui si fa la Fork.

Per rendere possibile l’esecuzione di Cucumber/java, sarà necessario aver installato sulla propria macchina Apachee Maven.

A questo punto, grazie al supporto al CI fornito da CucumberStudio, è possibile accedere a Travis CI con il proprio account Github.

In questo modo, andando a configurare il file travs.yml con l’ID del Test Run di CucumberStudio, sarà possibile rendere questi risultati disponibili in automatico su Travis e le metriche relative a Coffe Machine saranno aggiornate:

## Test runs

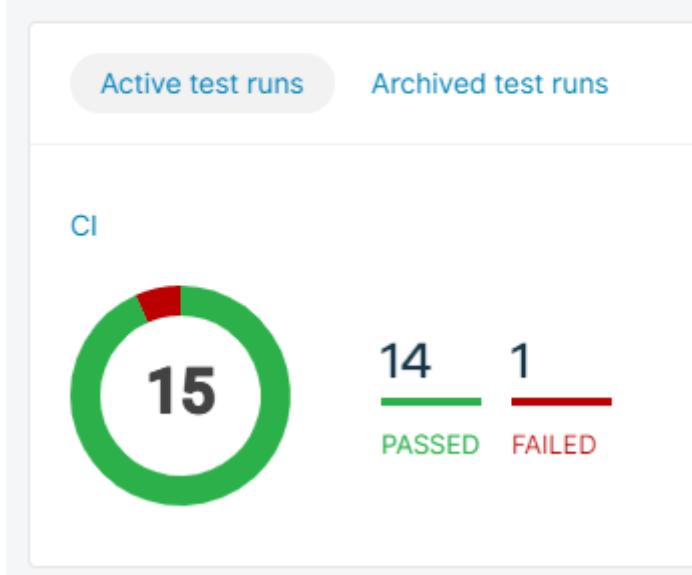


Figura 76: Metriche aggiornate tramite TravisCI

# Conclusioni

Nell'elaborato sono state introdotte le metodologie agile TDD e BDD, evidenziandone l'importanza per la gestione degli attuali cicli di sviluppo.

In particolare, è stata posta particolare attenzione sul BDD, con riferimento agli strumenti di base Gherkin e Cucumber, che sono coinvolti nell'intero ciclo di sviluppo.

Inoltre, ricorrendo a degli esempi pratici, ne è stata illustrata l'applicazione pratica, anche in ambienti di sviluppo differenti, quali Eclipse IDE, Android Studio e Cucumber Studio. Ciò che è emerso, per quanto concerne Cucumber, è che uno dei principali vantaggi di tale framework è la sua versatilità, in quanto esso può essere sia integrato in vari IDE e linguaggi di programmazione, sia può essere utilizzato per testare differenti tipologie di applicazioni, come web-applications ed applicazioni Android. Inoltre, sempre in tema di versatilità, Cucumber può essere facilmente combinato con ulteriori frameworks, come Selenium ed Android Espresso.

Infine, in relazione alla piattaforma CucumberStudio, è stato osservato che essa rappresenta una soluzione per gestire gli scenari di test per organizzazioni numerose, in quanto, nella stessa piattaforma, è presente una struttura che consente in modo immediato di gestire ruoli e condividere i test svolti, oltre ad avere in modo semplice sia la traduzione di Gherkin in step, sia suggerimenti per la scrittura degli scenari di test, facilitando il lavoro alle persone appartenenti al team con un minor grado di competenza informatica.

# Bibliografia

- [1] Wikipedia. (2021, 7 February). *Test Driven Development*. Tratto da Wikipedia, the free encyclopedia: [https://it.wikipedia.org/wiki/Test\\_driven\\_development](https://it.wikipedia.org/wiki/Test_driven_development)
- [2] G. Cerquone. (2018, 25 August). Tratto da ItalianCoders: <https://italiancoders.it/test-driven-development/>
- [3] Robert C. Martin. (2014, 17 December). *The Cycles of TDD*. Tratto da The Clean Code Blog: <https://blog.cleancoder.com/uncle-bob/2014/12/17/TheCyclesOfTDD.html>
- [4] (2020, August). *Test Driven Development (TDD)*. Tratto da Agile Way: <https://www.agileway.it/test-driven-development-tdd/>
- [5] Dan North. (2006, March). *Introducing BDD*. Tratto da Better Software: <https://dannorth.net/introducing-bdd/>
- [6] E. C. dos Santos. P. Vilain. (2018, 17 May). *Automated Acceptance Tests as Software Requirements: An Experiment to Compare the Applicability of Fit Tables and Gherkin Language*. Tratto da Agile Processes in Software Engineering and Extreme Programming, 2018, Volume 314: [https://link.springer.com/chapter/10.1007/978-3-319-91602-6\\_7](https://link.springer.com/chapter/10.1007/978-3-319-91602-6_7)
- [7] IEEE 1012-1986. IEEE: *IEEE Standard for Software Verification and Validation Plans*. IEEE Std 1012-1986. IEEE (1986)
- [8] L. A. Cisneros Gómez. (2018, September). *Analysis of the impact of test based development techniques (tdd, bdd, and atdd) to the software life cycle*. Tratto da Leiria: [https://iconline.ipleiria.pt/bitstream/10400.8/3699/1/Dissertation\\_2160085\\_LuisGomez.pdf](https://iconline.ipleiria.pt/bitstream/10400.8/3699/1/Dissertation_2160085_LuisGomez.pdf)
- [9] J. Nair. (2018, 18 April). *TDD vs BDD – What's the Difference Between TDD and BDD*. Tratto da: <https://blog.testlodge.com/tdd-vs-bdd/>
- [10] The Three Amigos. Tratto da: [Who Does What? - Cucumber Documentation](#)
- [11] What are the Three Amigos in Agile. Tratto da: [What are the Three Amigos in Agile? | Agile Alliance](#)
- [12] Enterprise BDD. Tratto da: <https://embrace-devops.com/2018/11/22/enterprise-bdd-3/amp/>
- [13] Gherkin. Gherkin Wiki. Tratto da:  
<http://github.com/cucumber/cucumber/wiki/Gherkin>
- [14] Gherkin. Tratto da: <https://www.geekandjob.com/wiki/gherkin>
- [15] Online Gherkin Editor. Tratto da SpecFlow: [https://specflow.org/gherkin-editor/?utm\\_source=website&utm\\_campaign=online%20gherkin%20editor%20&utm\\_term=organic&utm\\_medium=blog](https://specflow.org/gherkin-editor/?utm_source=website&utm_campaign=online%20gherkin%20editor%20&utm_term=organic&utm_medium=blog)
- [16] Sergio Scorsoglio. (2019, 28 October). *Introduzione a Gherking, la lingua di BDD per la scrittura dei requisiti di business*. Tratto da:  
<https://www.scaifinance.com/blog/introduzione-a-gherkin-la-lingua-di-bdd-per-la-scrittura-dei-requisiti-di-business/>
- [17] M.P. Korstanje. Cucumber. Tratto da Gherkin:  
<https://github.com/cucumber/cucumber/blob/master/gherkin/README.md>
- [18] Selenium. Tratto da: <https://www.selenium.dev/>

[19] *Abstract Syntax Tree (AST)*. Tratto da Tecnopedia:

<https://www.techopedia.com/definition/22431/abstract-syntax-tree-ast>

[20] CucumberStudio. Tratto da:

<https://support.smartbear.com/cucumberstudio/docs/general-info/about.html>