



Università
di Catania



Code Defender

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

<https://www.dmi.unict.it/amidolo/>

A.A. 2024/2025

Code Defenders

Code Defenders è un gioco basato sul Mutation Testing applicato su classi Java che fa uso di test in JUnit. Vengono definite due squadre che dovranno competere tra di loro attaccando e difendendo la classe Java testata e la sua test suite

- Gli **Attaccanti** dovranno inserire dei **difetti artificiali** (mutants) all'interno della classe
- I **Difensori** dovranno creare dei **test** che siano in grado di scovare i difetti inseriti dagli attaccanti

Difensori

- Il pannello di sinistra mostra la classe da testare (**CUT**)
- Il pannello di destra permette di scrivere un nuovo test
 - Tramite il tasto "Defend!" è possibile compilare ed eseguire il test

Class Under Test

```
1 public class Lift {
2
3     private int topFloor;
4     private int currentFloor = 0; // default
5     private int capacity = 10;    // default
6     private int numRiders = 0;   // default
7
8     public Lift(int highestFloor) {
9         topFloor = highestFloor;
10    }
11
12    public Lift(int highestFloor, int maxRiders) {
13        this(highestFloor);
14        capacity = maxRiders;
15    }
16
17    public int getTopFloor() {
18        return topFloor;
19    }
20
21    public int getCurrentFloor() {
22        return currentFloor;
23    }
24
25 }
```

Write a new JUnit test here

Defend!

```
1 import org.junit.Test;
2
3 import static org.junit.Assert.*;
4 import static org.hamcrest.MatcherAssert.assertThat;
5 import static org.hamcrest.Matchers.*;
6
7 public class TestLift {
8     @Test(timeout = 4000)
9     public void test() throws Throwable {
10         // test here!
11     }
12 }
```

Difensori

- Il pannello di sinistra permette di vedere i mutant esistenti, classificandoli nelle 4 categorie sottostanti.
- Se si sposta il mouse sul mutant, è possibile vedere le informazioni relative ad esso

```
16
17 public int getTopFloor() {
18  return topFloor;
19 }
20
21 public int getCurrentFloor() {
22     return currentFloor;
23 }
24
25 public int getCapacity() {
```

 Live  Killed  Claimed Equivalent  Equivalent

```
26     return capacity;
27
28
29
30 
31
32
33
34     return numRiders == capacity;
```

 Alive Mutants (Line 30)

Creator	ID	Score	Changed Lines
midolo2	24069	1	30

Difensori

- Il pannello di sinistra mostra i mutants creati dagli attaccanti
- Il pannello di destra mostra i test creati dai difensori

The image shows two side-by-side panels from a software testing tool. The left panel, titled '2 All Mutants', lists mutants created by attackers. It includes a sub-section 'Mutants outside methods' with 'Lift(int)' and 'Lift(int, int)', and a list of methods with their mutant counts: 'getTopFloor()' (1), 'getCurrentFloor()', 'getCapacity()', 'getNumRiders()' (1), 'isFull()', 'addRiders(int)', 'goUp()', 'goDown()', and 'call(int)'. The right panel, titled '1 All Tests', lists tests created by defenders for the same methods: 'Lift(int)', 'Lift(int, int)', 'getTopFloor()', 'getCurrentFloor()', 'getCapacity()', 'getNumRiders()' (1), 'isFull()', 'addRiders(int)', 'goUp()', 'goDown()', and 'call(int)'. Each item is represented by a button with a small icon and the method name.

Panel	Item	Count
Left Panel (Mutants)	All Mutants	2
	Mutants outside methods	-
	Lift(int)	-
	Lift(int, int)	-
	getTopFloor()	1
	getCurrentFloor()	-
	getCapacity()	-
	getNumRiders()	1
	isFull()	-
	addRiders(int)	-
	goUp()	-
	goDown()	-
	call(int)	-
	Right Panel (Tests)	All Tests
Lift(int)		1
Lift(int, int)		-
getTopFloor()		-
getCurrentFloor()		-
getCapacity()		-
getNumRiders()		1
isFull()		-
addRiders(int)		-
goUp()		-
goDown()		-
call(int)		-

Difensori

- Cliccando sul metodo è possibile vedere i mutants inseriti all'interno di esso e cliccando sul tasto **view** è possibile vedere il codice modificato

1 getTopFloor()

 [Mutant 24068](#) by midolo2 Modified line 18 Points: 1 [View](#)

- Per i mutant eliminati è possibile andare a vedere anche il test e l'output della compilazione con l'asserzione fallita

1 getNumRiders()

 [Mutant 24069](#) by midolo2 killed by midolo Modified line 30 Points: 1 [View](#) [View Killing Test](#)

Difensori

- Se il test contiene un errore, questo non viene compilato e l'errore di compilazione viene mostrato nella parte superiore della pagina. Sarà possibile correggere il test successivamente
- Nel caso di un test corretto, questo viene compilato e in caso di esito positivo della compilazione, viene mostrato il numero di mutant eliminati

Your test did not compile. Try again, but with compilable code.

```
[javac] TestLift.java:10: error: no suitable constructor found for Lift(no arguments)
[javac]         Lift lf = new Lift();
[javac]                   ^
[javac]         constructor Lift.Lift(int) is not applicable
[javac]             (actual and formal argument lists differ in length)
[javac]         constructor Lift.Lift(int,int) is not applicable
[javac]             (actual and formal argument lists differ in length)
[javac] 1 error
```

Great! Your test compiled and passed on the original class under test.

Great, your test killed a mutant!

Attaccanti

- Il pannello di sinistra mostra i mutants esistenti
- Il pannello di destra permette di manomettere il codice della CUT per inserire nuovi mutants
 - Il tasto "Attack!" permette di compilare il codice per vedere se il mutant inserito è valido

The screenshot displays the Attaccanti web application interface, divided into two main sections: "Existing Mutants" on the left and "Create a mutant here" on the right.

Existing Mutants: This section lists various mutants. At the top, there is a button labeled "2 All Mutants". Below it, a category "Mutants outside methods" includes "Lift(int)". Under "Methods", there are two mutants marked with a red "1": "getTopFloor()" and "getNumRiders()". Other methods listed include "getCurrentFloor()", "getCapacity()", "isFull()", "addRiders(int)", "goUp()", "goDown()", and "call(int)".

Create a mutant here: This section features a code editor with a "Reset" button (orange) and an "Attack!" button (blue) at the top right. The code editor contains the following Java code:

```
20
21 public int getCurrentFloor() {
22     return currentFloor;
23 }
24
25 public int getCapacity() {
26     return capacity;
27 }
28
29 public int getNumRiders() {
30     return numRiders;
31 }
32
33 public boolean isFull() {
34     return numRiders == capacity;
35 }
36
37 public void addRiders(int numEntering) {
38     if (numRiders + numEntering <= capacity) {
39         numRiders = numRiders + numEntering;
40     } else {
41         numRiders = capacity;
42     }
43 }
44
```

At the bottom of the code editor, there is a legend for mutant status: "Live" (red), "Killed" (green), "Claimed Equivalent" (grey), and "Equivalent" (purple). Below the legend are "Keyboard Shortcuts" and "Editor Mode: default" options. On the far right, the "Mutant restrictions" are set to "Moderate".

Attaccanti

- Nella parte inferiore della pagina, è possibile vedere tutti i test scritti dai difensori
- Cliccando sul metodo è possibile visualizzare tutte le informazioni specifiche di ogni singolo test. Inoltre il tasto **view** permette di vedere il codice del test

JUnit tests

- 1 All Tests
- 1 Lift(int)
- Lift(int, int)
- 1 getTopFloor()
- getCurrentFloor()
- getCapacity()
- getNumRiders()
- isFull()
- addRiders(int)
- goUp()
- goDown()
- call(int)

1 getTopFloor()

Test 20526	midolo	Covered: 1	Killed: 1	Points: 2	Good	View
------------	--------	------------	-----------	-----------	------	------

Attaccanti

- Una volta modificato il codice, questo verrà compilato così da verificare la correttezza della modifica
- Inoltre viene segnalato se il mutant creato è stato già eliminato o meno da un test esistente

Your mutant was compiled successfully.

Awesome, your mutant survived 2 existing tests!

Mutant Equivalente

- E' possibile creare un mutant la cui modifica non altera il comportamento della CUT (Class Under Test), di conseguenza nessun test potrà eliminarlo
- I difensori possono classificare un mutant di questo tipo come equivalente. Gli attaccanti potranno confermare l'equivalenza oppure presentare un test in grado di eliminare il mutant

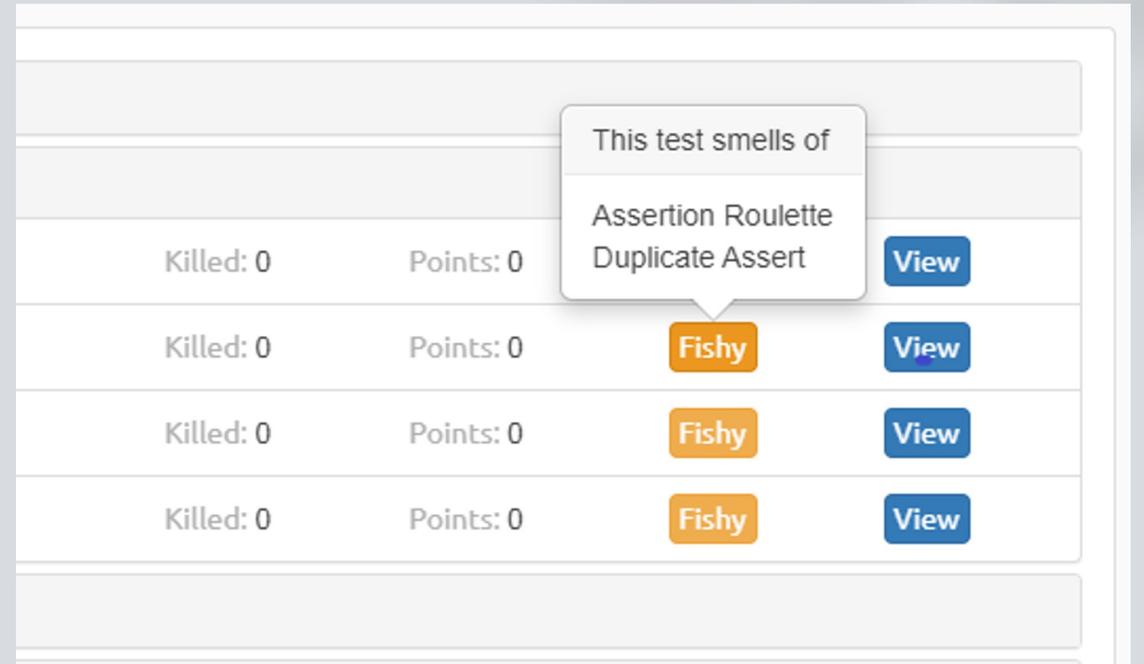
```
39     public void addRiders(int numEntering) {  
40         if (numRiders + numEntering <= capacity) {  
41             numRiders = numRiders + numEntering;  
42         } else {  
43             numRiders = capacity;  
44         }  
45     }
```

```
39     public void addRiders(int numEntering) {  
40         if (numRiders + numEntering > capacity) {  
41             numRiders = capacity;  
42         } else {  
43             numRiders = numRiders + numEntering;  
44         }  
45     }
```

Test Smells

- Gli Unit Test, proprio come il codice sorgente normale, è soggetto a cattive pratiche di programmazione, note anche come anti-pattern, difetti o **"smells"**
 - Assertion Roulette
 - Duplicate Assert
 - Eager Test
 - Unknown Test
 - e altri...

Ref: [Software Unit Test Smells](#)



The screenshot shows a table with four rows of test results. Each row has columns for 'Killed: 0', 'Points: 0', a 'Test Smell' label, and a 'View' button. A tooltip is visible over the first row, indicating the smell is 'Assertion Roulette Duplicate Assert'. The other three rows have the smell 'Fishy'.

Killed: 0	Points: 0	Assertion Roulette Duplicate Assert	View
Killed: 0	Points: 0	Fishy	View
Killed: 0	Points: 0	Fishy	View
Killed: 0	Points: 0	Fishy	View

Punteggi

Mutants

- Il mutante riceve un punto per essere stato creato e per essere sopravvissuto ai test già presenti
- Per ogni test che copre il codice mutato senza eliminare il mutante, quest'ultimo riceve un punto

Tests

- Il test prende un punto per aver ucciso un mutant ancora non eliminato
- Per ogni mutante eliminato, il test riceve tanti punti quanti punti ha il mutante eliminato

Equivalenze

- Se gli attaccanti provano che un mutante non è equivalente, mantengono il punteggio del mutante e ricevono un punto aggiuntivo
- Se gli attaccanti accettano l'equivalenza di un mutante, o il match termina, perdono i punti del mutante e i difensori ricevono un punto

Limiti per i Giocatori

Tests

- Non possono contenere cicli
- Non possono effettuare chiamate a `System.*`
- Non possono contenere nuovi metodi o nuove condizioni
- Possono contenere solo due asserzioni

Mutants

- Ci sono tre livelli di limitazioni ai mutant, noi utilizzeremo la limitazione **Relaxed** con qualche modifica
 - Non sarà quindi possibile effettuare chiamate a `System.*` e `Random.*`

Relaxed

- No calls to `System.*`, `Random.*`

Moderate

- No comments
- No additional logical operators (`&&`, `//`)
- No ternary operators
- No new control structures (`switch`, `if`, `for`, ...)

Strict

- No reflection
- No bitwise operators (bitshifts and logical)
- No signature changes



Università
di Catania



The End 🦙

Alessandro Midolo, PhD

Dipartimento di Matematica e Informatica

Università di Catania

alessandro.midolo@unict.it

<https://www.dmi.unict.it/amidolo/>

A.A. 2024/2025