Università di Catania

Uni ct MATEMATICA E INFORMATICA

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

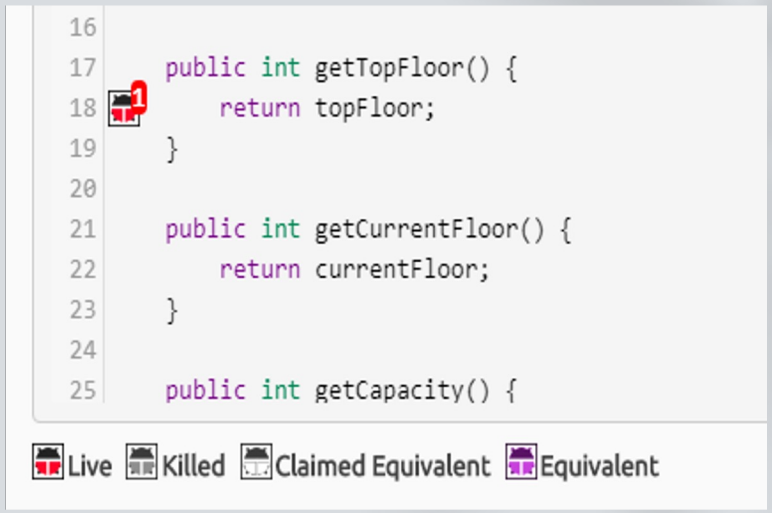
- Gli ***Attaccanti*** dovranno inserire dei ***difetti artificiali*** (mutants) all'interno della classe
- I ***Difensori*** dovranno creare dei ***test*** che siano in grado di scovare i difetti inseriti dagli attaccanti

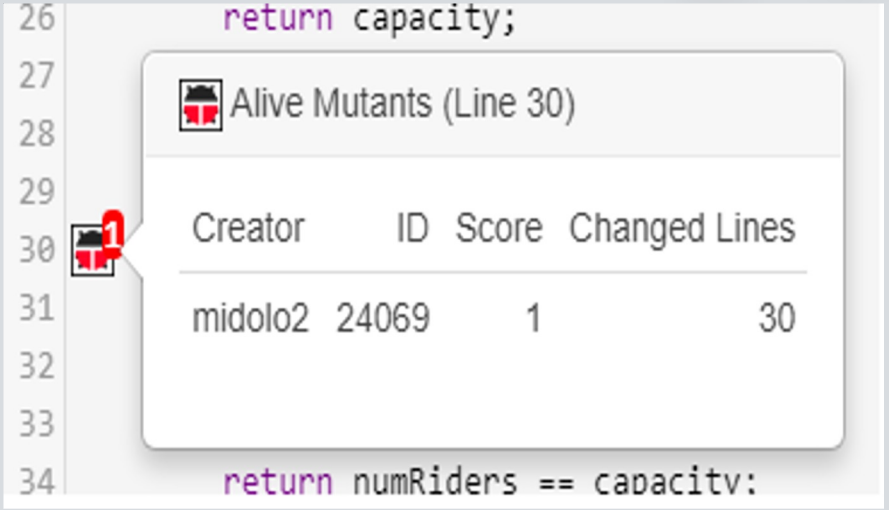
Difensori

- 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

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



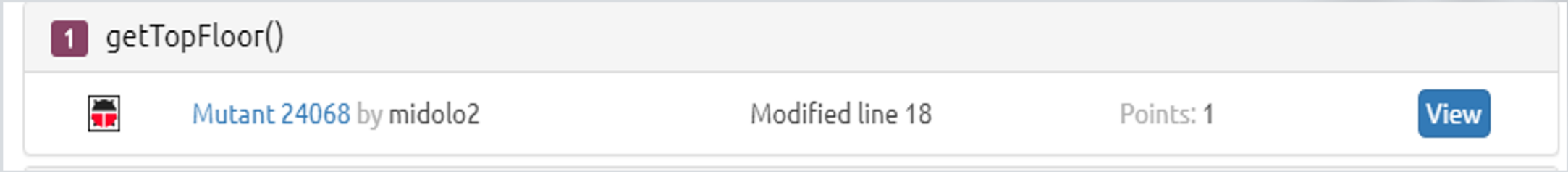


Difensori

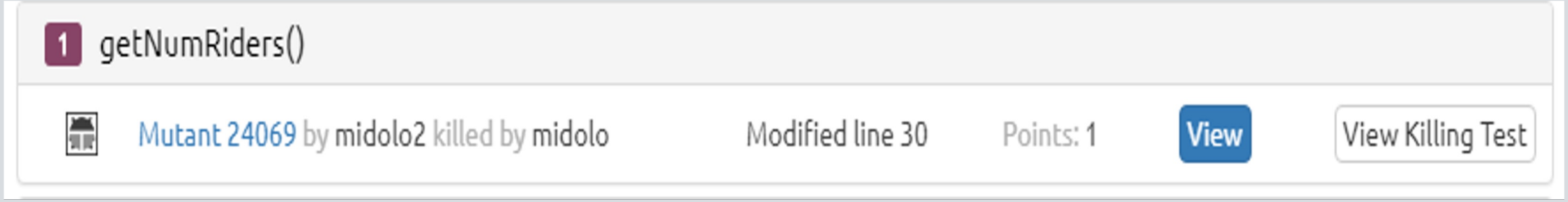
- Il pannello di sinistra mostra i mutants creati dagli attaccanti
- Il pannello di destra mostra i test creati dai difensori

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



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



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.
```

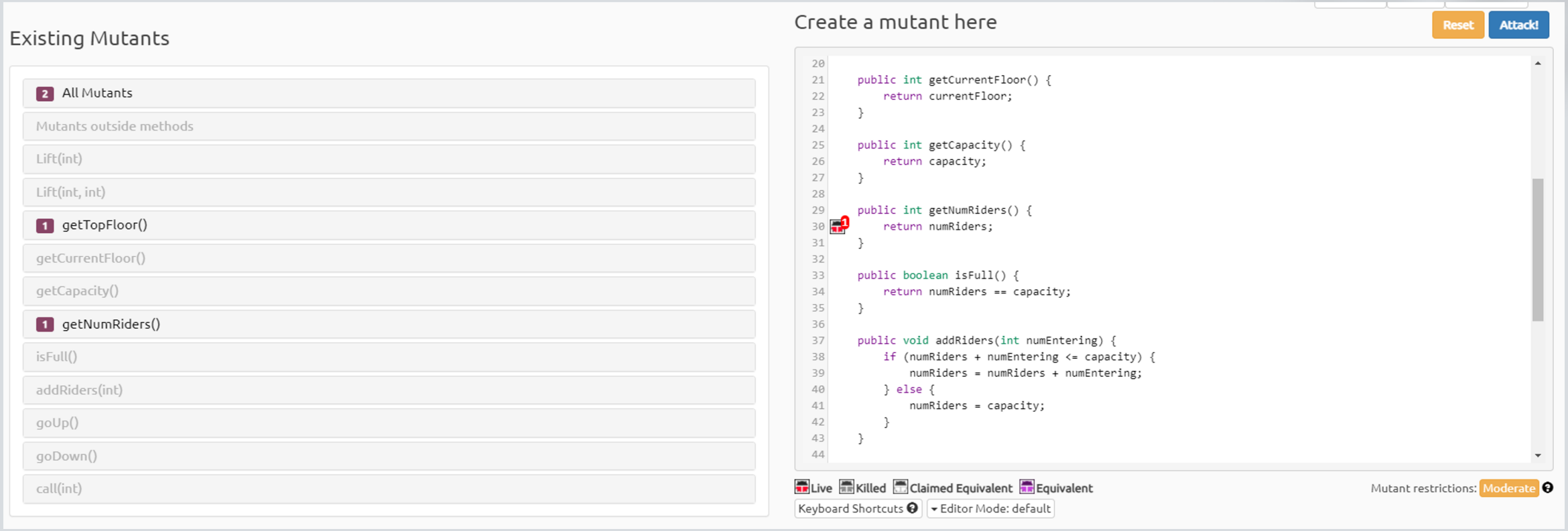
```
[javac] TestLift.java:10: error: no suitable constructor found for Lift(no arguments)
[javac]         Lift lf = new Lift();
[javac]                   ^
[javac]     constructor Lift.Lift(int) is not applicable
[javac]       (actual and formal argument lists differ in length)
[javac]     constructor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

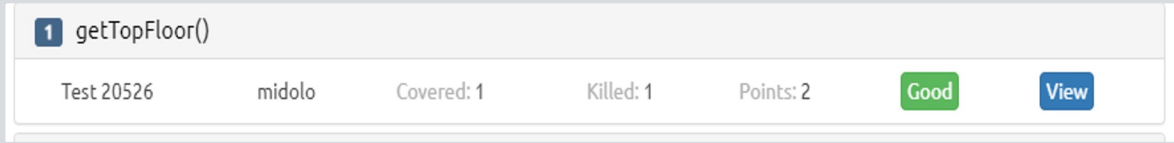
Attaccanti

- 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



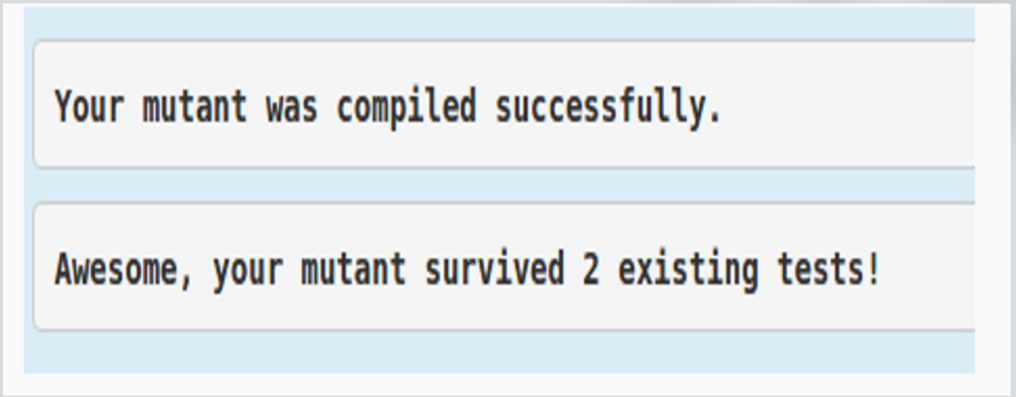
Attaccanti

- 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



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



Mutant Equivalente

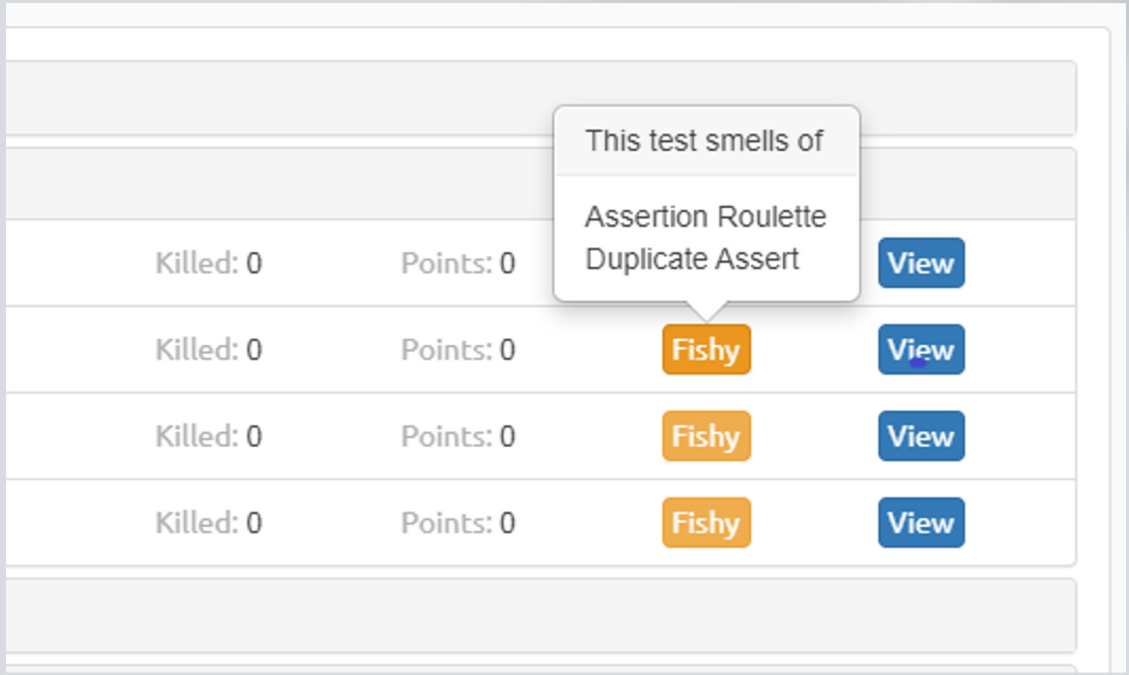
- E' possibile creare un mutant la cui modifica non altera il comportamento della CUT (Class Under Test), di conseguenza nessun test potrà eliminarlo
- I difensori possono classificare un mutant di questo tipo come equivalente. Gli attaccanti potranno confermare l'equivalenza oppure presentare un test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

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

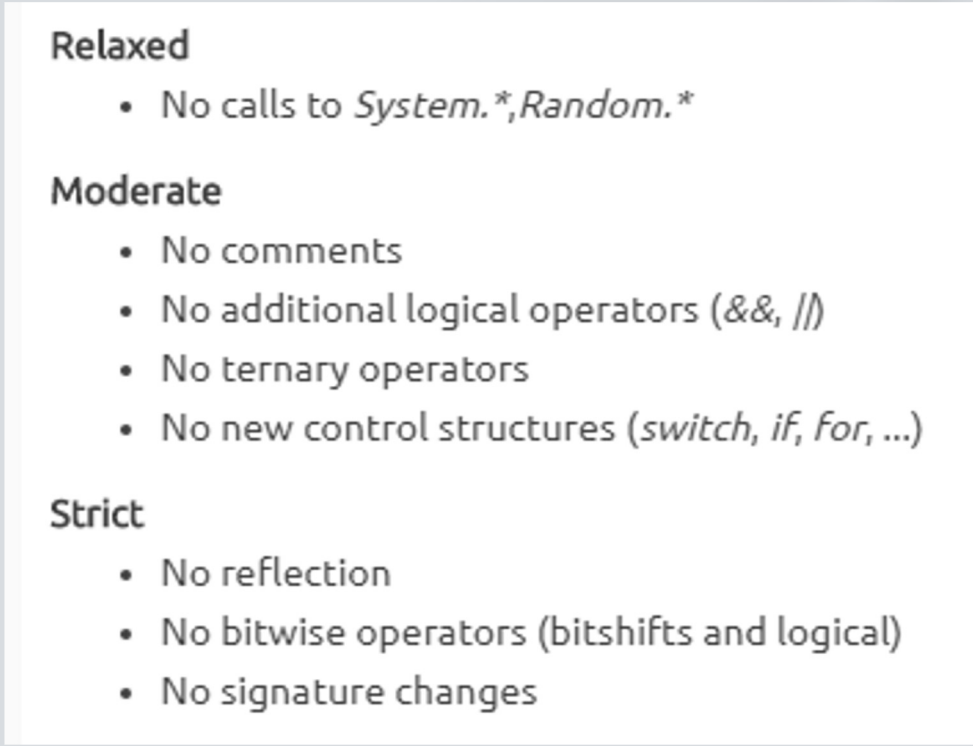
Limiti per i Giocatori

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.*



Università di Catania

Uni ct MATEMATICA E INFORMATICA

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