Eclipse

Eclipse

- **Eclipse è una delle più utilizzate IDE (Integrated Development Environment) per programmare in Java e molti altri linguaggi**

- **Ci sono molte altre alternative più potenti (IntelliJ Idea), più leggere (VSCode) oppure specifiche per alcuni linguaggi e ambienti (Android Studio oppure Jbuilder)**
  - Si è scelto Eclipse perché probabilmente è il semplice da utilizzare e installare , anche in assenza di connessione

Materiale da scaricare

- **Eclipse download : https://www.eclipse.org/downloads/**
  - Insieme ad Eclipse c'è anche una versione di Java e Junit, che saranno gli strumenti fondamentali che utilizzeremo
  - Una volta scaricato l'eseguibile è sufficiente avviarlo, rispondere positivamente ad eventuali richieste e lasciarlo installare in una cartella a piacere

Funzioni fondamentali di Eclipse

Importare un progetto

- **Consideriamo uno dei progetti degli esercizi assegnati**
  - Decomprimere lo zip (ad esempio con Estrai tutto … in Windows)
  - Dal menu file scegliere l'opzione Import
  - Scegliere Existing Projects into Workspace (perché stiamo importando dei progetti già sviluppati proprio con Eclipse)
  - In Select Root Directory inserire la cartella nella quale si è decompresso l'esercizio
  - Scegliere il progetto
  - Premere il pulsante Finish

Eseguire un progetto

- **Al termine dell'importazione il progetto viene anche compilato e non dovrebbe avere problemi**

- **Per eseguire un progetto bisogna:**
  - Cercare tra i file del progetto un file con un main di partenza (il codice sorgente è nella sottocartella src)
  - Premere il pulsante destro del mouse
  - Scegliere Run As …
  - Scegliere Java Application
- **L'output del programma dovrebbe essere visibile o in un'interfaccia grafica oppure in una piccola scheda denominata Console**

Eseguire un test

- **In molti dei nostri esempi non ci sono Main, quindi non c'è nulla da eseguire, ma possiamo eseguire i test**

- **Per eseguire un test**
  - Tasto destro su una classe di test (di solito contengono la parola test nel nome oppure si trovano in una sottocartella test)
  - Click su Run as JUnit Test
- **Si aprirà una scheda con i risultati del test**

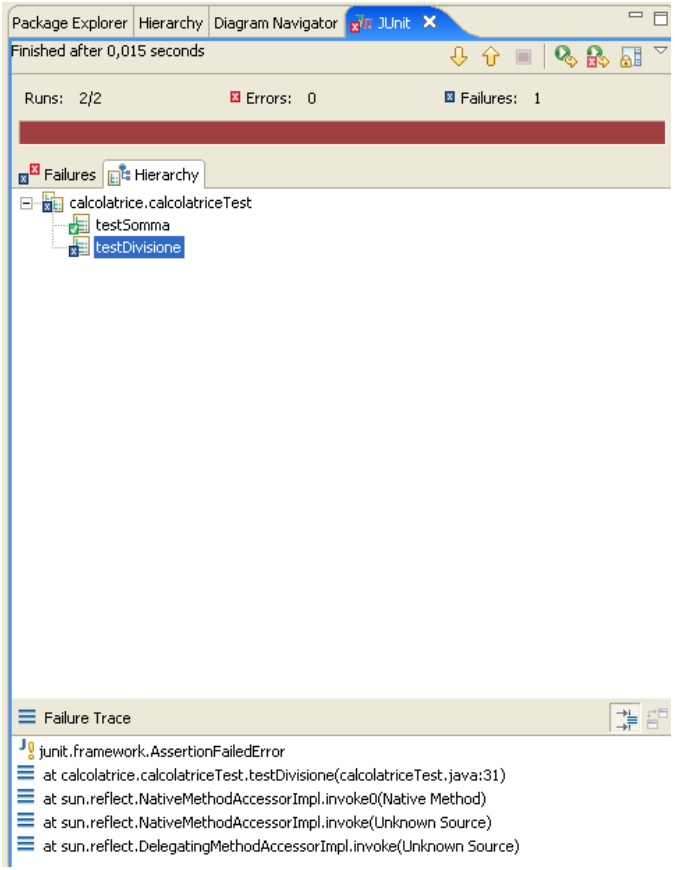
Avvertenza

- **Nei nostri esercizi spesso non andiamo a modificare i file di test ma solo i file csv con i valori dei test**
- **Quando si fa doppio click sul file csv in eclipse si apre un editor esterno (oppure possiamo modificare il csv con il blocco note fuori da Eclipse)**
- **Se andiamo subito a rieseguire i test, Eclipse potrebbe non accorgersi che il file è stato modificato**
  - Per costringerlo ad accorgersene andare sul nome del progetto in Eclipse e premere F5

Risultati del test

- **Runs : numero di test eseguiti**
- **Errors : numero di test che non sono stati eseguiti (errore nel test)**
- **Failure : numero di test che sono stati eseguiti ma hanno trovato un fallimento**

- **Cliccando su di un test si può leggere in basso l'eventuale messaggio di errore**



Misurare la copertura

- **La copertura viene misurata da un'estensione di Eclipse che si chiama EclEmma che però viene installata insieme ad Eclipse**
- **Per misurare la copertura è sufficiente**
  - Premere il pulsante destro sulla classe di test
  - Scegliere Coverage as …
  - Scegliere JUnit Test
- **Si aprirà la finestra di output e si colorerà il codice sorgente**

Scheda di coverage

- **Se non riuscite a vedere la scheda andate nel menu Window, poi Show View, poi Other …, poi Java, poi Coverage**
- **Attenzione: ci interessa solo la copertura della classe Java e non della classe test**

Element	Covera...	Covered Instructions	Missed Instructions	Total Instructions
PCTO-Calcolatrice	91,5 %	118	11	129
src	91,5 %	118	11	129
calcolatrice	91,5 %	118	11	129
Calcolatrice.java	76,5 %	13	4	17
Calcolatrice	76,5 %	13	4	17
divisione(int, int)	100,0 %	6	0	6
radice(int)	0,0 %	0	4	4
somma(int, int)	100,0 %	4	0	4
calcolatriceTest.java	93,8 %	105	7	112

Codice colorato

- **Verde : eseguito**
- **Rosso : non eseguito**
- **Giallo : eseguito parzialmente**
  - Non sono stati eseguiti test con b==0

```
package calcolatrice;

public class Calcolatrice {

    public int somma(int a, int b) {
        return a+b;
    }

    public float divisione (int a, int b) {
            if (b!=0)
                return (float)(a)/ (float)b;
            else
                return Float.NaN;
    }

    public double radice (int x) {
        return Math.sqrt(x);
    }
}
```

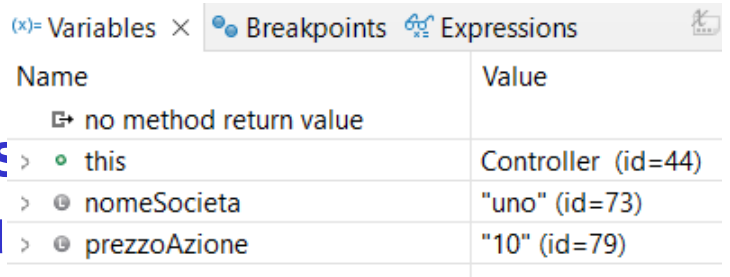
Debugging

Funzionalità di debugging

- **Inserimento break point**
  - Tramite Eclipse è possibile accedere a delle “Breakpoint properties”
    - Breakpoint condizionali:
      - Breakpoint che si attivano solo quando si passa in una certa riga e si verifica una certa condizione
    - Breakpoint dipendenti dallo Hit Count:
      - Breakpoint che si attivano solo dopo un certo numero di passaggi su quella riga di codice
- **Esecuzione passo passo del codice**
  - Entrando o meno all’interno dei metodi chiamati
  - Uscendo da un metodo verso il chiamante

Esecuzione step by step

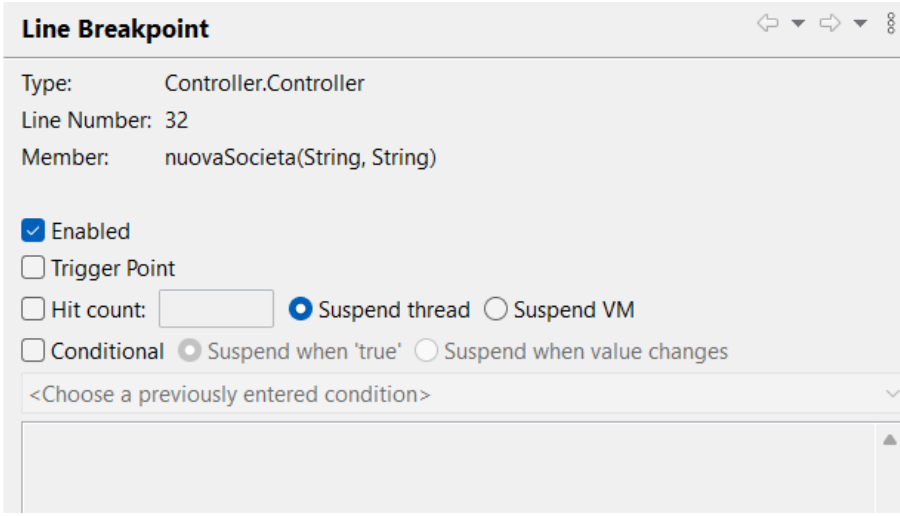
- **Quando il programma si ferma su di un breakpoint è possible:**
  - Farlo continuare (pulsante play)
  - Proseguire all'interno della funzione chiamata nella riga attuale
  - Proseguire con la riga successiva
  - Proseguire fino alla fine dell'esecuzione della funzione corrente



chiedere di calcolare eventuali espressioni

Breakpoint condizionali

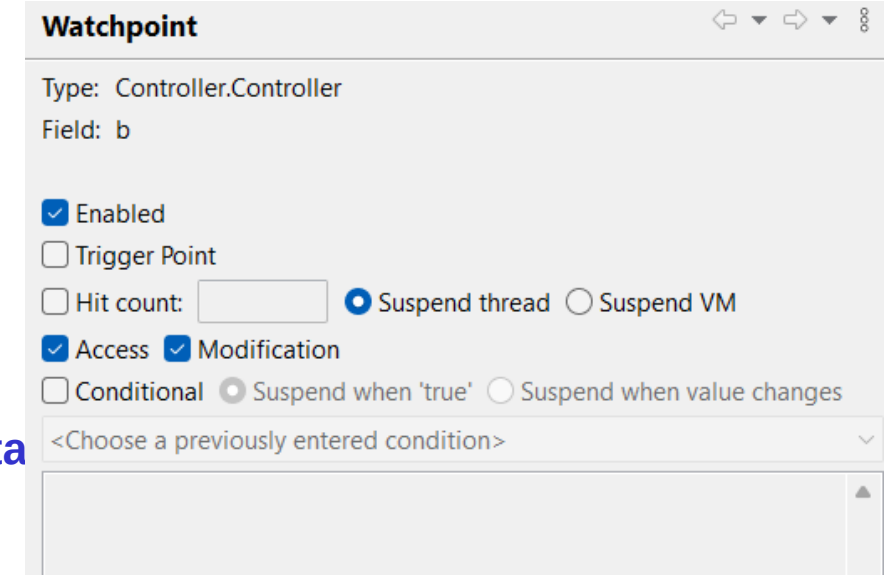
- **Per abbreviare il debugging dovremmo poter sistemare il breakpoint il più vicino possible alla *causa* del difetto**
- **Nel dubbio, c'è il rischio di dover eseguire trope righe prima di arrivare a quella incriminate**
  - Ad esempio se c'è un ciclo ripetuto 1000 volte, come possiamo fare a fermarci dopo 789 esecuzioni senza eseguire altrettante volte il ciclo?
- **Breakpoint condizionali**
  - Si attivano con il tasto destro su di un breakpoint e scegliendo Breakpoint properties



- Hit Count: il breakpoint ferma il programma solo quando si passa per la i-esima volta su di esso
- Conditional: ogni volta che passiamo sul breakpoint si valuta la condition: solo se è vero il programma si sospende

Watchpoint

- **Un watchpoint è un breakpoint collegato non ad una riga ma ad un attributo di una classe**
- **Si inserisce come un breakpoint, puntando sulla riga di dichiarazione dell'attributo**
- **Fa sospendere l'esecuzione ogni volta che l'attributo è letto o scritto**
- **Tramite le Properties è possibile personalizzare quando effettuare la sospensione**

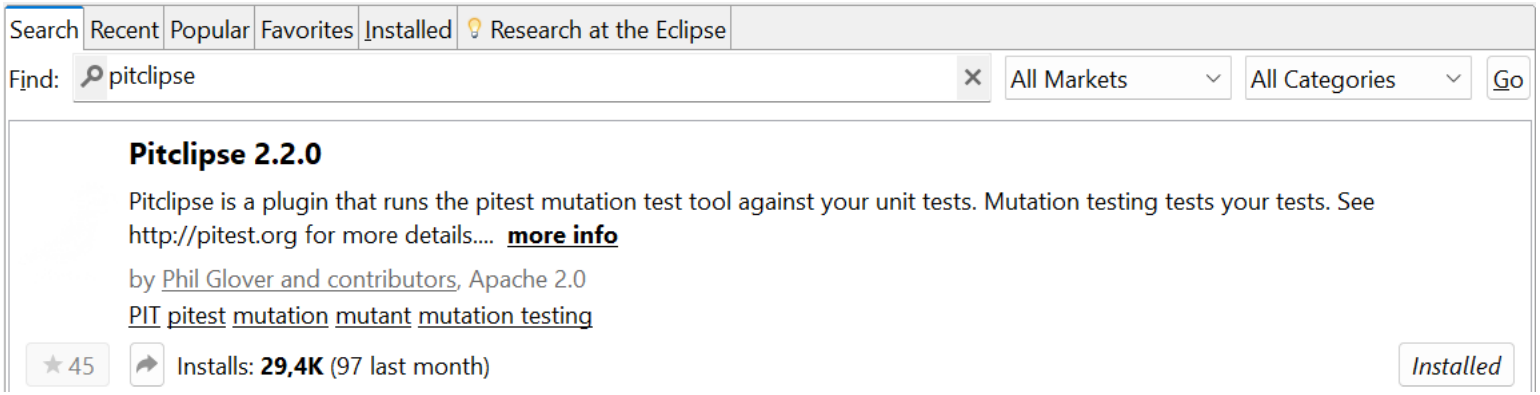


Funzioni avanzate di Eclipse

Mutation Testing con PIT

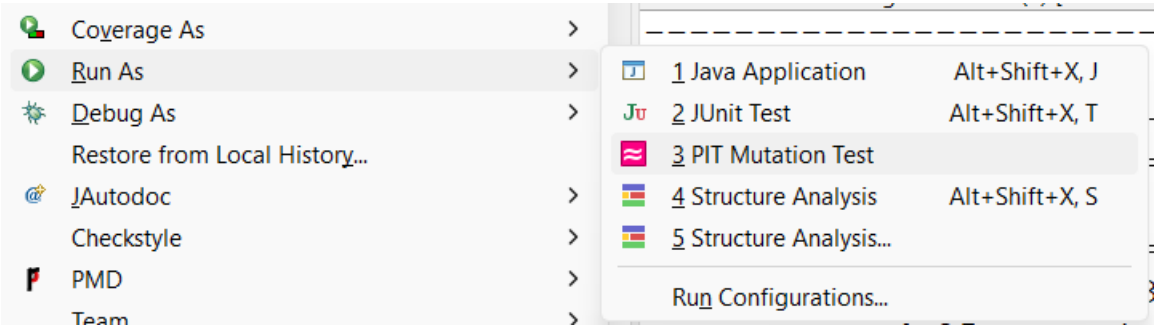
Mutation based testing with pitclipse

- **Pitclipse è un'estensione di Eclipse da installare**
- **Nel menu Help selezionare Eclipse Marketplace**
- **Nella casella Find scrivere pitclipse**
- **Selezionare pitclipse e seguire le istruzioni**



Using Pitclipse

- **PIT can be used in Eclipse by the Run as contextual menu**
- **It can be applied on a JUnit test suite without failing test cases**



Pitclipse Output

- **L’output di Pitclipse si trova nelle schede Console, Pit Summary e Pit Mutations**
- **Se non vedete le schede Pit andate nel menu Window, poi Show View, poi Other …, poi Pit e poi PIT Mutations e/o PIT Summary**

Pitclipse Output

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	90%	9/10	88%	22/25	88%	22/25

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
default	1	90%	9/10	88%	22/25	88%	22/25

Report generated by PIT 1.6.8

Pitclipse Output

- **L'output di Pitclipse si trova nelle schede Console, Pit Summary e Pit Mutations**

- **Scheda console (parte finale) :**

```
================================================================================
- Statistics
================================================================================
>> Line Coverage: 11/13 (85%)
>> Generated 25 mutations Killed 22 (88%)
>> Mutations with no coverage 0. Test strength 88%
>> Ran 287 tests (11.48 tests per mutation)
```

Pitclipse output : scheda Pit Summary

Pit Test Coverage Report

Project Summary

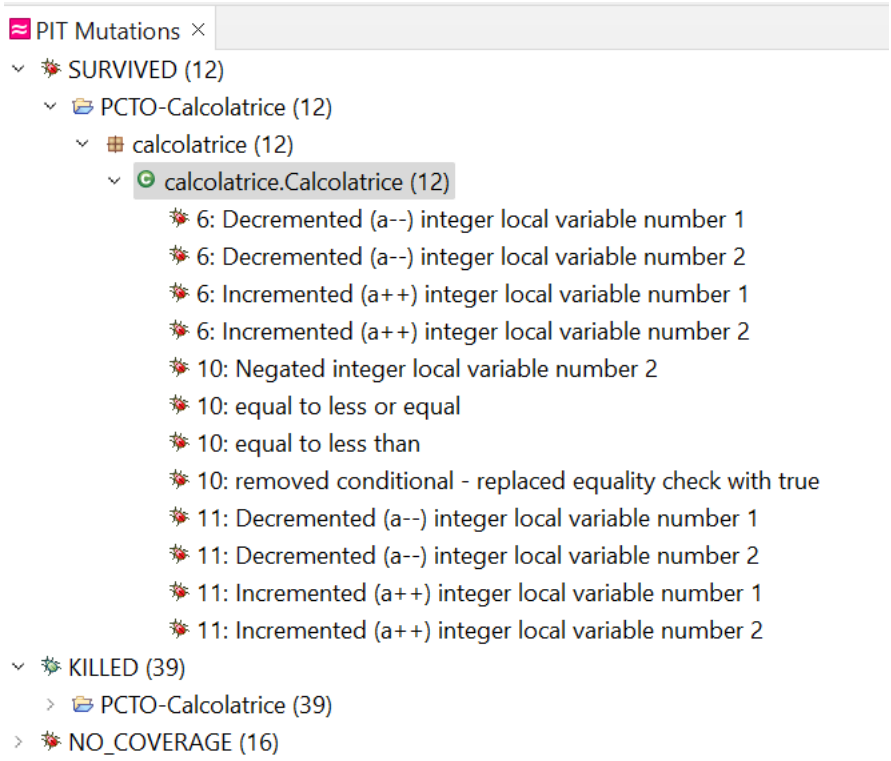
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	90%	9/10	88%	22/25	88%	22/25

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
default	1	90%	9/10	88%	22/25	88%	22/25

Report generated by PIT 1.6.8

Piclipse Output : Scheda Pit Mutations



Pitclipse Output

Active mutators

- BOOLEAN_FALSE_RETURN
- BOOLEAN_TRUE_RETURN
- CONDITIONALS_BOUNDARY_MUTATOR
- EMPTY_RETURN_VALUES
- INCREMENTS_MUTATOR
- INVERT_NEGS_MUTATOR
- MATH_MUTATOR
- NEGATE_CONDITIONALS_MUTATOR
- NULL_RETURN_VALUES
- PRIMITIVE_RETURN_VALS_MUTATOR
- VOID_METHOD_CALL_MUTATOR

Tests examined

- TriangleTypeTest.[engine:junit-jupiter]/[class:TriangleTypeTest]/[test-template:test3WiseACOC(int, int, int, java.lang.String)]/[test-template-invocation:#15] (8 ms)
- TriangleTypeTest.[engine:junit-jupiter]/[class:TriangleTypeTest]/[test-template:testBCCExtended(int, int, int, java.lang.String)]/[test-template-invocation:#6] (6 ms)
- TriangleTypeTest.[engine:junit-jupiter]/[class:TriangleTypeTest]/[test-template:test3WiseACOC(int, int, int, java.lang.String)]/[test-template-invocation:#19] (8 ms)
- TriangleTypeTest.[engine:junit-jupiter]/[class:TriangleTypeTest]/[test-template:testBCCExtended(int, int, int, java.lang.String)]/[test-template-invocation:#2] (9 ms)

Pitclipse Output