

T9-G19: Requisiti sull'esecuzione del robot Randoop



Software Architecture Design

Francesco Panariello - M63001433

Francesca Di Martino - M63001478

Giovanni Riccardi - M63001480

Giuseppe Castaldo - M63001539

Riccardo Romano - M63001489

AA: 2022/2023

Sommario

Capitolo 1: Introduzione al problema.....	3
Traccia.....	3
Primo approccio al problema	3
Copertura del codice nel testing: importanza e misurazione	3
Approfondimento delle tecnologie da utilizzare.....	4
Randoop	4
Emma.....	5
Altri strumenti	5
Glossario dei termini	6
Capitolo 2: Organizzazione del team.....	7
Strategia di Sviluppo	7
Effort table.....	7
Capitolo 3: Documentazione di analisi	8
Requisiti funzionali	8
Requisiti sui dati	8
Requisiti non funzionali	8
Scenario del caso d'uso	9
Diagramma dei casi d'uso.....	10
Diagramma delle Classi di Analisi	11
Sequence Diagram di Analisi	12
Capitolo 4: Documentazione di progetto	13
Approccio alla realizzazione	13
Inizializzazione e definizione variabili.....	15
Component Diagram	16
Architettura a Pipeline.....	17
Sequence Diagram di Progetto.....	19
Package Diagram	20
Soluzioni alternative	20
Capitolo 5: Documentazione di implementazione	21
Deployment Diagram.....	21
Struttura Repository	22
Installazione.....	22
Esempio di utilizzo	23

Capitolo 1: Introduzione al problema

Traccia

L'applicazione deve offrire la funzionalità di generazione dei test su una data classe Java usando il Robot Randoop.

Tale funzionalità riceverà in input un file di testo (classe da testare), dovrà lanciare il generatore ed esecutore di Test Randoop, restituendo in output il codice di casi di test generati ed i risultati dell'esecuzione.

L'esito dell'esecuzione dovrà essere elaborato in maniera da estrarre da essi le informazioni rilevanti ai fini del gioco (ad esempio, la copertura del codice, etc.).

Primo approccio al problema

La traccia prevede la generazione di casi di test in maniera randomica tramite l'utilizzo di Randoop partendo da una classe java ricevuta in input.

Per procedere allo sviluppo del task è stato deciso mediante brainstorming di gruppo di utilizzare in concomitanza con il robot uno strumento di valutazione della copertura per ottenere risultati più efficienti rispetto alla sola generazione randomica dei casi di test; infatti, per evitare la generazione di numerosi casi di test spesso simili tra loro o troppo semplici, la valutazione della coverage aiuta ad effettuare una cernita per creare diversi livelli di difficoltà.

Copertura del codice nel testing: importanza e misurazione

La copertura del codice è un concetto fondamentale nel processo di testing del software. Consiste nell'analizzare il codice sorgente e determinare quanto di esso viene effettivamente eseguito durante l'esecuzione dei test.

La copertura del codice può essere suddivisa in diverse categorie, tra cui:

- Copertura delle linee di codice: Questa metrica indica la percentuale di linee di codice che vengono attraversate durante l'esecuzione dei test. Una linea di codice viene considerata coperta se viene eseguita almeno una volta. Una copertura del 100% delle linee di codice significa che ogni linea è stata eseguita durante i test.
- Copertura dei rami condizionali: I rami condizionali si verificano quando si ha un'istruzione "if" o "switch" nel codice. Questa metrica indica la percentuale di rami condizionali che vengono attraversati dai test. Un ramo condizionale viene considerato coperto se sia la condizione "true" che "false" vengono eseguite almeno una volta. Una copertura del 100% dei rami condizionali indica che ogni possibile ramo è stato attraversato durante i test.
- Copertura delle istruzioni: Questa metrica indica la percentuale di istruzioni che vengono effettivamente eseguite durante i test. Le istruzioni possono includere assegnazioni, chiamate a metodi e altre operazioni. Una copertura del 100% delle istruzioni significa che ogni istruzione è stata eseguita durante i test.

Una copertura del codice elevata è generalmente considerata un segno di buona qualità dei test. Tuttavia, è importante notare che la copertura del codice da sola non garantisce la completa assenza di errori nel software. È possibile avere una copertura del 100% del codice, ma ancora avere difetti o comportamenti non desiderati a causa di errori di logica o di progettazione. La copertura del codice è uno strumento molto utile per gli sviluppatori perché fornisce un feedback

immediato sulla quantità di codice testato. Aiuta a identificare le aree del codice che richiedono ulteriori test e revisioni. Attraverso l'analisi della copertura del codice, gli sviluppatori possono migliorare la qualità del software, individuare e correggere bug potenziali e aumentare la stabilità complessiva dell'applicazione. Strumenti come Emma, semplificano il processo di misurazione della copertura del codice fornendo report chiari e ben strutturati. Questi strumenti consentono agli sviluppatori di identificare rapidamente le aree che richiedono maggiore attenzione e ottimizzazione nei test, migliorando così l'efficienza e l'efficacia del processo di sviluppo del software.

Approfondimento delle tecnologie da utilizzare

Randoop

Randoop è un generatore di test di unità per Java ideato da Microsoft Research. Esso crea automaticamente test per le classi in formato JUnit. Il suo lavoro si basa sulla tecnica del feedback direct random testing (test casuale orientato a retroazione): genera sequenze casuali a partire dalle classi del programma in prova, che poi invia al programma stesso, formulando periodicamente delle asserzioni sul suo comportamento. In questo modo è possibile richiamare l'attenzione su eventuali carenze e/o eccezioni del relativo programma, nonché ottenere test di regressione per evitare di introdurre errori nelle versioni aggiornate. Pertanto, questa tecnica genera in modo pseudo-casuale ma intelligente, chiamate di metodo/costruttore per le classi in fase di test. Randoop esegue le sequenze generate, utilizzando i risultati dell'esecuzione per creare asserzioni che descrivono il comportamento del programma.

Randoop genera tipicamente due tipi di file:

- Error Test: Test che trovano errori/bug nel codice;
- Regression Test: Test di regressione per eventuali modifiche future;

Come descritto dal sito ufficiale di Randoop, la combinazione di generazione ed esecuzione di test di tale strumento, si traduce in una tecnica di generazione di test molto efficace. Randoop ha rivelato errori precedentemente sconosciuti anche in librerie molto utilizzate, tra cui i JDK di Sun e IBM e un componente core di .NET.

Ciò che risulta utile per i fini del progetto è la possibilità di inserire mediante linea di comando, alcuni parametri facoltativi, come:

output-limit : Utile per inserire il numero massimo di test da eseguire;

time-limit: Utile per inserire il tempo massimo durante il quale eseguire i test;

randoom-seed: Seme utilizzato per la generazione di test casuali;

classlist: Permette di inserire un file .txt, con la lista di classi da testare;

Emma

Emma è uno strumento utilizzato per misurare e riportare la copertura del codice Java di un dato progetto, ideato da Vlad Roubtsov per aiutare gli sviluppatori Java fornendo un tool gratuito per il controllo della copertura, anziché spendere cifre esorbitanti tra licenze varie anche per il controllo di un progetto di piccole dimensioni.

Emma tiene traccia del codice Java e riporta i valori di copertura in un dato report, che può essere in formato XML, HTML o semplicemente .txt, che contiene i valori di coverage in termini percentuali. Il report risulta chiaro e ben suddiviso, così da fornire allo sviluppatore la possibilità di comprendere velocemente dove ottimizzare i test per la realizzazione di un progetto più completo e stabile.

Inoltre, Emma può essere integrato in molti ambienti di sviluppo integrati (IDE) come Eclipse, IntelliJ IDEA e NetBeans, rendendo facile l'uso da parte degli sviluppatori durante il ciclo di sviluppo del software.

Si noti che attraverso EMMA è possibile esplorare il codice sorgente secondo varie tipologie: per classi, per metodi, per linee e per blocchi, a seconda del dettaglio con cui si vuole misurare la coverage del testing.

Emma permette due tipi di esecuzione:

- **Offline:** La strumentazione è completamente indipendente dall'esecuzione dell'applicazione che si vuole testare; in questa modalità si eseguono i test prima di analizzare il codice e si ottiene un report di copertura dei test successivamente all'esecuzione.
- **On the fly:** Verificando gli eventi a tempo di esecuzione, l'analisi di copertura dunque viene fatta durante l'esecuzione dei test, in tempo reale.

Per lanciare Emma, contestualmente alla generazione dei test con Randoop, viene utilizzata la modalità offline, viene fatta quindi una strumentazione dell'applicazione per valutare la copertura dei test generati.

Emma, inoltre, è fondamentale poiché mette a disposizione la funzione "merge", la quale permette di aggregare facilmente i risultati di sessioni differenti, ottenendo quindi la loro unione.

Altri strumenti

Github	Servizio di hosting per progetti software
Eclipse	IDE utilizzato per lo sviluppo del software. È ampiamente usato per la programmazione in Java, ma supporta anche altri linguaggi come C/C++, Python e PHP.
Visual Paradigm	Strumento di modellazione che supporta UML 2, SysML e la notazione BPMN dell'OMG. È utilizzato per creare diagrammi per la progettazione del software.
Maven	Strumento di build automation utilizzato prevalentemente nella gestione di progetti Java.

Glossario dei termini

Randoop	Tool open-source per la generazione automatica di test di unità in Java
EMMA	Abbreviazione di "Expert system for Multi-objective Markovian Analysis" ed è un framework per l'analisi della copertura del codice Java
Robot	Macchina programmabile in grado di eseguire compiti in modo autonomo, si intende il componente che esegue Randoop
Coverage	Misura della quantità di codice sorgente che è stata coperta da un insieme di test automatizzati
File system	Struttura dati deputata alla gestione ed archiviazione dei file su un computer

Capitolo 2: Organizzazione del team

Strategia di Sviluppo

Il nostro gruppo ha adottato una strategia di sviluppo che ha enfatizzato il gioco di squadra e la collaborazione. Abbiamo riconosciuto l'importanza di sfruttare le diverse competenze e prospettive dei nostri membri, creando così un ambiente in cui ognuno poteva contribuire in modo significativo. Le sessioni di brainstorming hanno alimentato l'innovazione e hanno portato a soluzioni creative, mentre le riunioni regolari ci hanno aiutato a monitorare il progresso e affrontare tempestivamente eventuali sfide. Inoltre, abbiamo adottato un approccio flessibile, consentendoci di adattarci rapidamente ai cambiamenti e alle nuove esigenze che emergevano durante il processo di sviluppo. Questa sinergia di competenze, spirito collaborativo e adattabilità ha reso possibile l'elaborazione di un software per il Robot Randoop che riflette appieno il potenziale del nostro team.

Effort table

Giorno	Tempo	Descrizione
Prima Iterazione		
09/04	2 h 30 m	Brainstorming generale, studio delle tecnologie e possibili sviluppi
11/04	2 h	Studio approfondito del task ed ideazione dei diagrammi e scenari
13/04	2 h	Bozza iniziale del prototipo e completamento dei diagrammi
16/04	3 h 30 m	Ideazione e testing prototipo iniziale
18/04	1 h	Review pre-consegna
Seconda Iterazione		
28/04	1 h 30 min	Brainstorming generale, definizione degli obiettivi
02/05	2 h	Confronto con altri gruppi per la comprensione del prelievo della classe
04/05	1 h	Ideazione dei possibili livelli di difficoltà del Robot Randoop
06/05	2 h 30 min	Confronto con altri gruppi per la coverage mediante Robot EvoSuite
Terza Iterazione		
12/05	1h 30 min	Brainstorming generale, definizione degli obiettivi
18/05	1h 50 min	Estrapolazione dati di coverage
25/05	1h 30 min	Confronto con altri team per l'implementazione di chiamate API
29/05	2h 20 min	Esposizione casi di test attraverso API
31/05	2h	Progettazione livelli di difficoltà del Robot Randoop e primo prototipo
03/06	2h 30 min	Realizzazione diagrammi di sequenza e package
05/06	2h	Presentazione Powerpoint, raffinamento diagrammi, correzione bug
Consegna finale		
15/06	1h	Aggiornamento codice
21/06	2h 30m	Aggiornamento codice
27/06	1h	Revisione e organizzazione attività
02/07	3h	Aggiornamento codice e documentazione
07/07	2h 30m	Aggiornamento codice e documentazione
12/07	5h 30m	Aggiornamento codice e test di funzionamento su altre macchine
13/07	4h	Revisione generale e aggiornamento codice
14/07	7h	Revisione generale e aggiornamento documentazione
28/07	3h	Correzione diagrammi e rivisitazione della documentazione
29/08	1h	Rilettura attenta ed eventuali correzioni
04/09	30min	Revisione finale

Capitolo 3: Documentazione di analisi

Requisiti funzionali

I requisiti funzionali sono descrittivi del comportamento del sistema in termini di funzionalità offerte:

1. La funzionalità deve ricevere in input una classe Java, corrispondente alla classe da testare;
2. La funzionalità deve valutare la coverage;
3. La funzionalità deve restituire in output i casi di test generati;

Requisiti sui dati

I requisiti sui dati sono specifiche o regole che definiscono le caratteristiche, le restrizioni e le condizioni applicabili ai dati utilizzati in un sistema software. Questi requisiti stabiliscono le aspettative riguardo ai dati che il sistema deve elaborare, memorizzare, visualizzare o trasmettere. Essi contribuiscono a garantire che i dati siano corretti, completi, coerenti, sicuri e conformi alle esigenze dell'applicazione.

1. Il nome della classe non deve contenere caratteri speciali;
2. Il nome della classe non deve essere una stringa vuota.

Requisiti non funzionali

I requisiti non funzionali, invece, sono descrittivi di proprietà relative al sistema. Dunque, definiscono o limitano le proprietà del sistema come prestazioni, scalabilità e sicurezza.

Il nostro team si è posto di soddisfare i seguenti requisiti non-funzionali:

1. Tempi di risposta: Il sistema deve rispondere alle richieste dell'utente entro un tempo accettabile. Ad esempio, il tempo di risposta massimo per l'esecuzione di un'operazione non deve superare i 10 secondi nel 90% dei casi;
2. Per la generazione dei casi di test, il file della classe viene prelevato da un percorso specifico del filesystem in accordo con i team degli altri task;
3. Gestione degli errori: Il sistema deve essere in grado di gestire correttamente eventuali errori durante l'esecuzione, ad esempio, segnalando errori di generazione dei casi di test o errori nel percorso del file della classe;
4. Scalabilità: Randoop deve essere in grado di gestire grandi progetti e classi complesse senza compromettere le prestazioni. Deve essere in grado di gestire un numero significativo di iterazioni e livelli di copertura senza rallentamenti significativi;
5. Usabilità: L'applicazione deve essere intuitiva e facile da usare. Gli utenti devono essere in grado di configurare e avviare la generazione dei casi di test senza difficoltà;

6. Affidabilità: Il sistema deve essere affidabile e stabile durante l'esecuzione. Non dovrebbe verificarsi un'interruzione improvvisa o un comportamento imprevisto che possa compromettere la generazione dei casi di test o causare la perdita di dati.

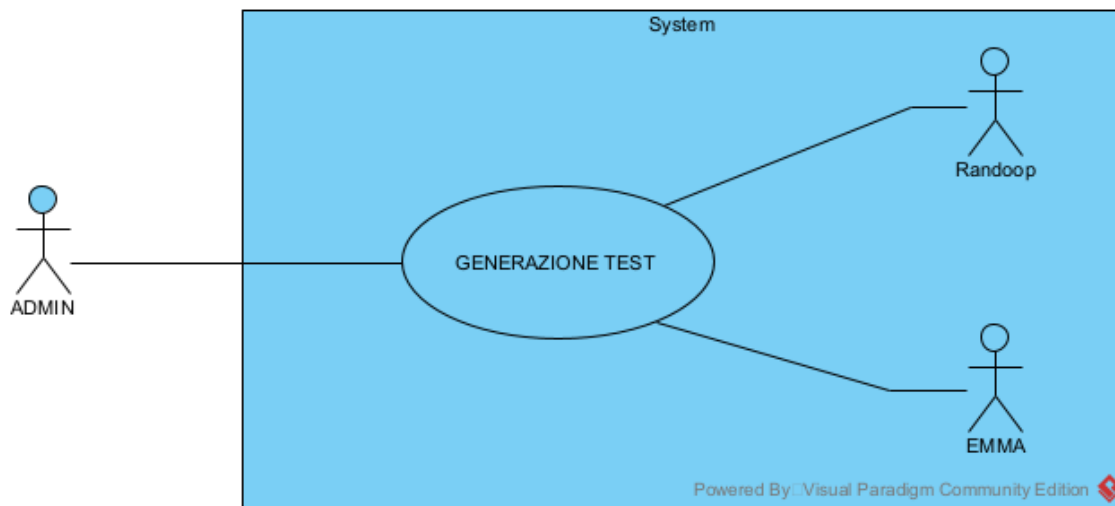
Scenario del caso d'uso

Caso d'uso:	Task T9
Attore primario	Amministratore
Attore secondario	Randoop&Emma
Descrizione	All'occorrenza, quando viene inserita una nuova classe nel gioco, vengono generati i casi di test attraverso Randoop e viene effettuata un'analisi di coverage attraverso Emma.
Pre-Condizioni	L'amministratore ha deciso di generare i casi di test sulla nuova classe.
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il sistema rileva se sono state aggiunte nuove classi nel percorso predefinito. 2. Il sistema lancia Randoop per la generazione dei casi di test ed Emma per l'analisi della coverage. 3. I casi di test relativi alla classe testata vengono memorizzati all'interno del filesystem.
Post-Condizioni	I casi di test vengono generati.
Casi d'uso correlate	<i>nessuno</i>
Sequenza di eventi alternativi	Se la classe non viene trovata: <ol style="list-style-type: none"> 1. Il programma si interrompe

Diagramma dei casi d'uso

Come passo successivo abbiamo realizzato un diagramma dei casi d'uso usando il linguaggio di modellazione UML (Unified Modeling Language) mediante il software Visual Paradigm.

Gli attori che sono coinvolti sono: l'Admin, il robot Randoop e il tool di copertura EMMA.

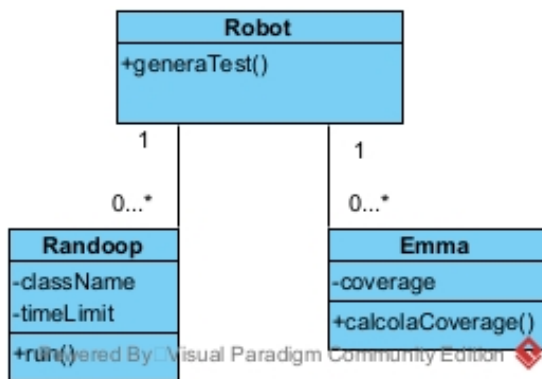


Il diagramma dei casi d'uso rappresenta le interazioni tra gli attori (nel nostro caso l'Admin) e le funzioni o servizi offerti dal sistema. In questo nostro scenario, l'unico caso d'uso identificato è la **generazione dei casi di test**.

"Generazione Test" : rappresenta l'azione o la funzionalità che l'attore "Admin" può eseguire all'interno del sistema. Questo caso d'uso indica che l'Admin è in grado di generare test utilizzando Randoop.

- "Randoop" : componente utilizzato per generare i test.
- "Emma" : componente utilizzato per la copertura.

Diagramma delle Classi di Analisi



Il **diagramma delle classi di analisi** rappresenta le principali classi coinvolte nel programma , quindi , si focalizza sulle loro caratteristiche e funzioni specifiche. Questo tipo di diagramma è utilizzato durante la fase di analisi del sistema, prima della fase di progettazione.

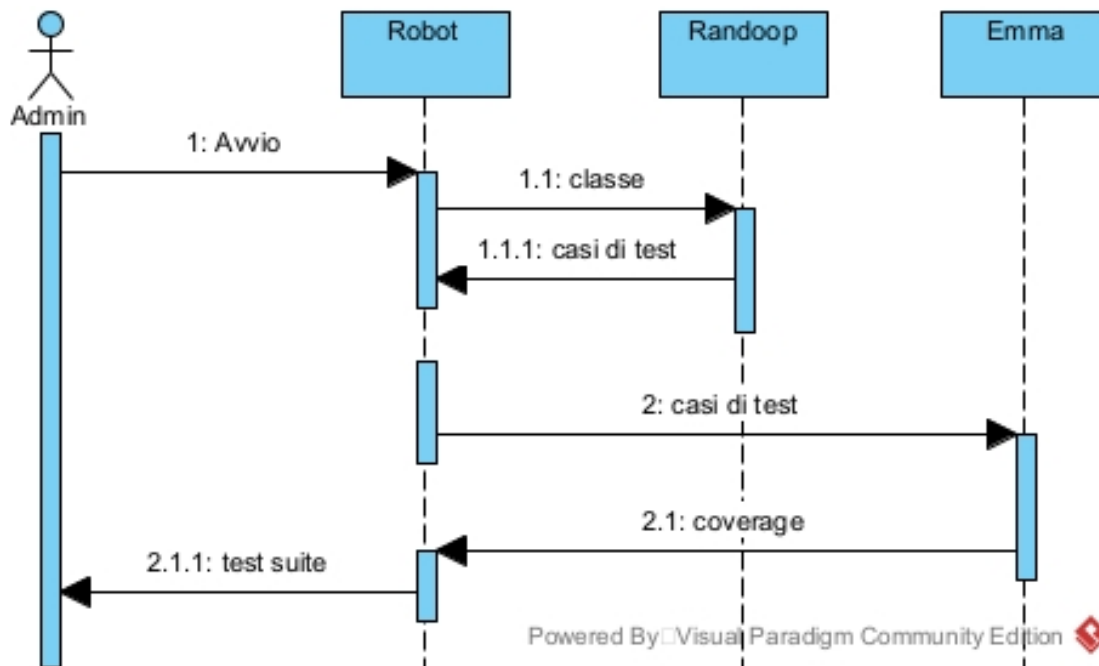
Durante l'analisi, l'obiettivo principale è comprendere i requisiti del sistema e identificare le classi chiave, i loro attributi e le loro relazioni generali. Il diagramma di analisi è quindi più ad alto livello e non include dettagli implementativi o design pattern specifici.

Le classi presenti sono 3, caratterizzate da una relazione di associazione.

Le relazioni tra le classi sono le seguenti:

- *Robot* utilizza la classe *Randoop* per eseguire il test del file specificato.
- *Robot* utilizza anche la classe *Emma* per ottenere la percentuale di copertura delle linee di codice.

Sequence Diagram di Analisi



Il **diagramma di sequenza di analisi** rappresenta la sequenza di azioni e comunicazioni tra gli attori e le classi coinvolte nel sistema durante un determinato scenario. È utile per comprendere il flusso di interazioni tra gli oggetti e le operazioni che avvengono all'interno del sistema.

La sequenza è la seguente:

- (1) L'admin avvia il Robot.
- (2) Il Robot analizza le classi da testare e le comunica a Randoop.
- (3) Randoop genera i casi di test a partire dalle classi da testare.
- (4) Il Robot invoca Emma per un'analisi della coverage sui casi di test generati.
- (5) La test suite contenente i casi di test e l'analisi della coverage viene ritornata all'amministratore.

Capitolo 4: Documentazione di progetto

Approccio alla realizzazione

Si vuole realizzare un'applicazione che implementi l'Educational Game "Man vs automated Testing Tools challenges": si compete contro diversi tipi di strumenti di generazione automatica di casi di test, quali ad esempio Randoop ed Evosuite. La realizzazione della Web Application per il gioco sul Testing è stata divisa in nove task. In particolare, il nostro task, il requisito specifico 9, consiste nello sviluppo di casi di test per classi Java, attraverso il Robot Randoop.

Nello specifico: L'applicazione deve offrire la funzionalità di generazione dei test su una data classe Java usando il Robot Randoop. Tale funzionalità riceverà in input un file di testo (classe da testare), dovrà lanciare il generatore ed esecutore di Test Randoop, restituendo in output il codice di casi di test generati ed i risultati dell'esecuzione. L'esito dell'esecuzione dovrà essere elaborato in maniera da estrarre da essi le informazioni rilevanti ai fini del gioco (ad esempio, la copertura del codice, etc.)

Si eseguono le seguenti operazioni:

1. Scansione del filesystem per trovare classi da testare.

La struttura del filesystem FolderTree è così organizzata:

```
\~~~AUTName
+~~~AUTSourceCode
|   AUTClass.java
|
+~~~RobotTest
|   +~~~EvoSuiteTest
|   |   +~~~01Level
|   |   |   +~~~TestReport
|   |   |   |   \~~~TestSourceCode
|   |   +~~~02Level
|   |   |   +~~~TestReport
|   |   |   |   \~~~TestSourceCode
|   |   \~~~xxLevel
|   |       +~~~TestReport
|   |       |   \~~~TestSourceCode
|   \~~~RandoopTest
|   |   +~~~01Level
|   |   |   +~~~TestReport
|   |   |   |   \~~~TestSourceCode
|   |   +~~~02Level
|   |   |   +~~~TestReport
|   |   |   |   \~~~TestSourceCode
|   |   \~~~xxLevel
|   |       +~~~TestReport
|   |       |   \~~~TestSourceCode
|   \~~~StudentLogin
|   |   \~~~GameId
|   |   |   GameData.csv
|   |   |
|   |   +~~~TestReport
|   |   |   \~~~TestSourceCode
```

2. Inizializzare le variabili necessarie, come il conteggio di copertura, il limite di tempo, il numero massimo di iterazioni, etc...
3. Avviare un ciclo che continua fino a quando non viene soddisfatta una delle seguenti condizioni:
 - Il numero massimo di iterazioni è stato raggiunto;
 - La saturazione massima è stata raggiunta;
 - Il numero massimo di livelli di gioco è stato raggiunto.

All'interno del ciclo:

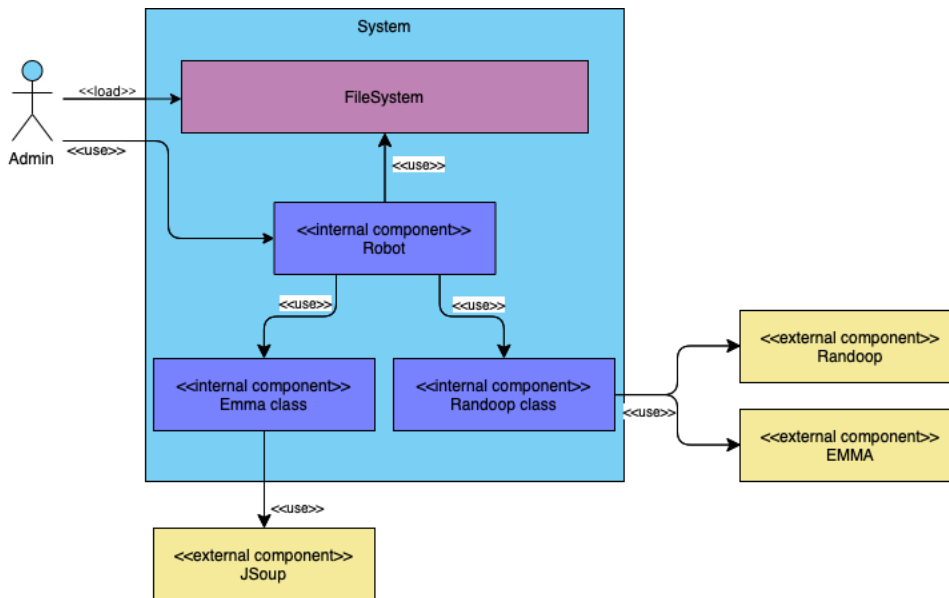
- Eseguire Randoop sulla classe da testare con i parametri appropriati (file di input, limite di tempo, iterazione corrente).
- Calcolare la copertura del codice ottenuta dai risultati dell'esecuzione:
 - Se la copertura del codice è la stessa della precedente iterazione, incrementare la saturazione.
 - Altrimenti, incrementare il livello, quindi, creare una nuova directory per il livello corrente e riempirla con tutti i test creati in precedenza.
- Si incrementa il tempo di esecuzione.

Inizializzazione e definizione variabili

Cov	Coverage	0	Rappresenta il valore della copertura delle linee di codice ottenuto dai casi di test generati da Randoop. Viene calcolato utilizzando il metodo <code>LineCoverage</code> della classe <code>Coverage_Emma</code> .
Ex_cov	Previous Coverage	0	Rappresenta il valore della copertura delle linee di codice ottenuto nella precedente iterazione del ciclo. Viene utilizzato per confrontare con la copertura corrente e valutare se c'è stato un miglioramento o una corrispondenza della copertura.
Timelimit	Time Limit	5	Rappresenta il limite di tempo (in secondi) assegnato a Randoop per generare i casi di test, incrementato ad ogni iterazione sulla stessa classe.
Iter	Iteration	0	Rappresenta il numero corrente di iterazioni nel ciclo. Viene incrementato ad ogni iterazione.
Max_iter	Maximum Iterations	1000	Rappresenta il numero massimo di iterazioni consentite nel ciclo do-while. Se il numero di iterazioni raggiunge questo limite, il ciclo si interrompe.
Sat	Saturation	0	Variabile utilizzata per indicare se la generazione dei casi di test ha raggiunto un punto in cui non sta ottenendo miglioramenti significativi nella copertura delle linee di codice. Conta le iterazioni sulla stessa classe consecutive che non producono un miglioramento sulla Coverage.
Max_sat	Maximum Saturation	10	Questa variabile indica il limite di iterazioni consecutive senza miglioramenti accettabili nella copertura.
Max_liv	Maximum Level	10	Rappresenta il numero massimo di livelli di generazione dei casi di test consentiti. Se il valore di livello raggiunge questo limite, il ciclo si interrompe.

Component Diagram

Il diagramma dei componenti offre una visione ad alto livello del sistema, evidenziando le sue interazioni tra i componenti e gli attori esterni.



L'attore admin provvede al caricamento di nuove classi da testare nel filesystem.

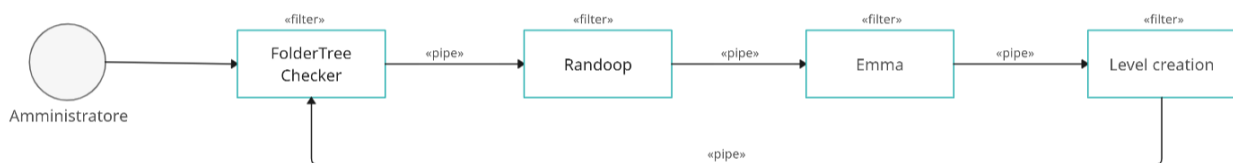
L'admin può avviare *Robot* indipendentemente dal caricamento di nuove classi ma solo nel caso trovi classi non testate nel filesystem. Nel caso in cui ci fossero nuove classi, il robot procederà all'utilizzo di *Randoop* e *Emma*.

La classe *Randoop* fa uso dell'external component Randoop per generare test randomici ed Emma per la valutazione della coverage.

L'internal class *Emma* fa uso di un component esterno JSoup che leggerà il file xml per estrarre il valore di copertura.

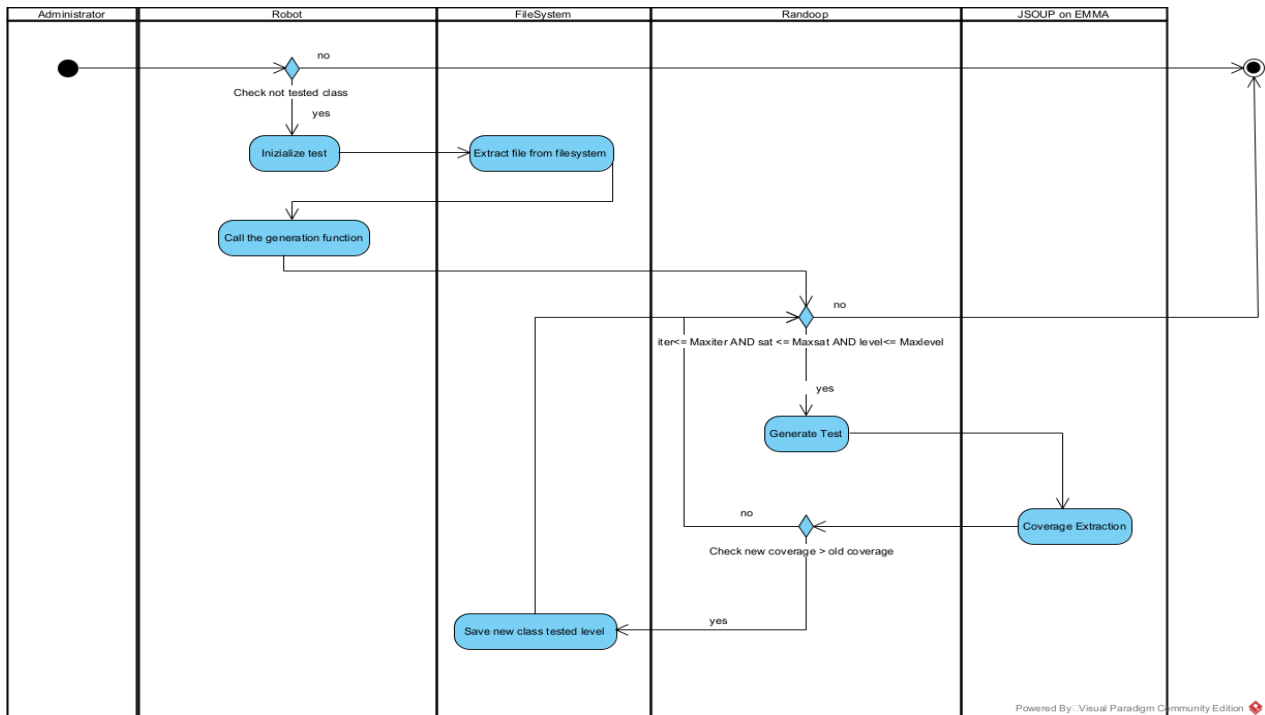
Architettura a Pipeline

Seppur in questo caso abbiamo l'assenza di una vera e propria coda per la gestione delle richieste, (in virtù dell'assenza di uno stato *"always on"* dell'applicativo che invece viene avviato solo quando l'admin ha necessità di generare i casi di test), il passaggio agli stadi successivi della pipe viene implementato a livello applicativo mediante chiamate a funzioni. Data la ciclicità dell'esecuzione si ha un controllo ciclico del Folder Tree in modo da assicurarsi che per tutte le classi inserite vengano creati i casi di test ed i livelli ai fini del gioco.



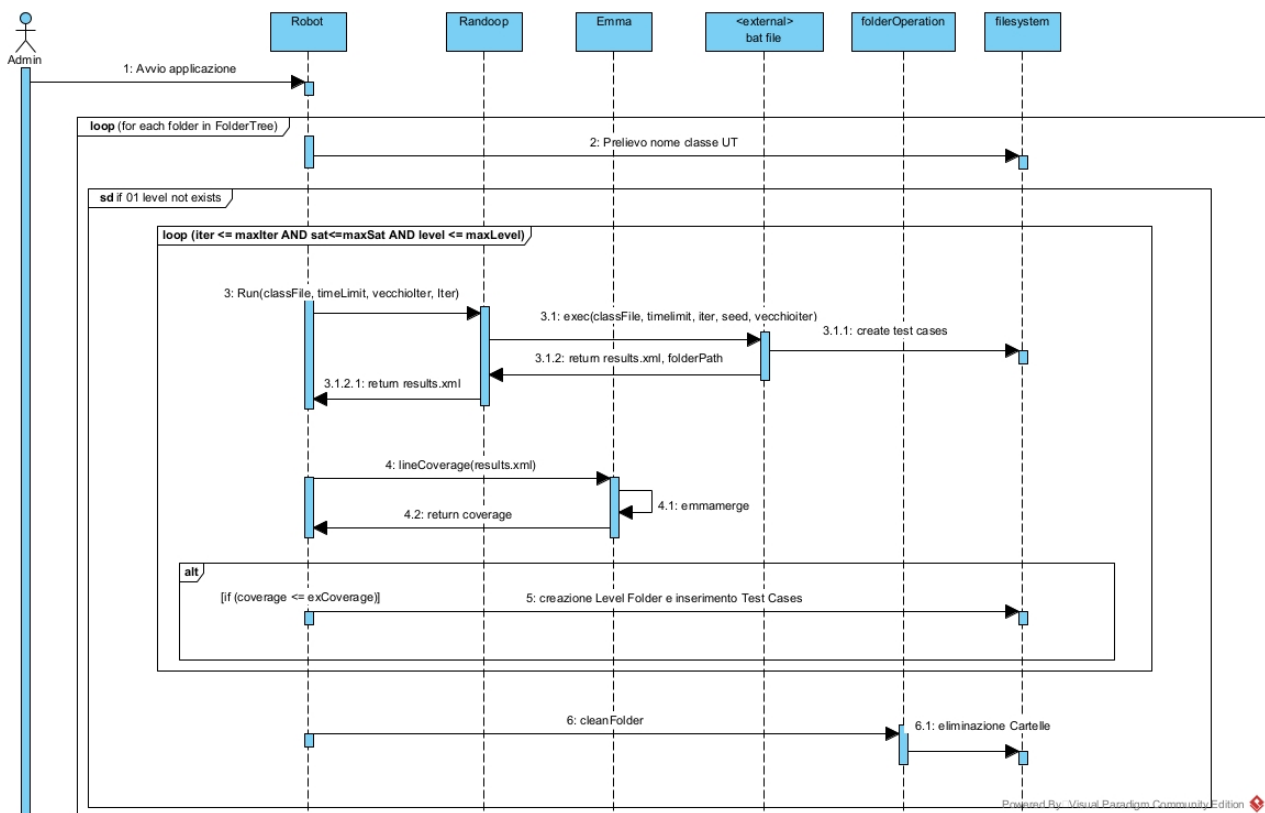
Activity Diagram

Il diagramma delle attività aiuta a comprendere il flusso del processo, l'ordine delle azioni e le decisioni prese. Può essere utilizzato per fornire una panoramica chiara delle attività coinvolte.



- Il processo parte con il caricamento della classe.
- Nel blocco "System", viene eseguito un controllo per verificare se ci sono classi non testate.
 - Se ci sono classi non testate in precedenza, il processo passa direttamente al nodo finale, terminando l'esecuzione del programma.
 - Se la classe è stata testata in precedenza, il blocco "Randoop" inizializza i test.
- Viene estratta la classe dal file system.
- Vengono inizializzati i parametri di valutazione.
- Successivamente, viene controllato se i parametri di valutazione sono verificati
 1. Se tutte le condizioni non sono soddisfatte, vengono generati nuovi test.
 - 1.1. Viene eseguita l'estrazione della copertura (coverage extraction).
 - 1.2. Viene confrontata la nuova copertura con quella precedente. Se la nuova copertura è maggiore della precedente, viene salvato il nuovo livello di classe testato.
 2. Se le condizioni del loop sono soddisfatte, termina l'esecuzione.

Sequence Diagram di Progetto



Nel diagramma di sequenza di progettazione, l'avvio del sistema da parte dell'amministratore è seguito da un ciclo che continua finché ci sono classi non ancora analizzate nel filesystem.

Durante il ciclo:

- Il robot preleva una classe, e se la classe non è stata testata precedentemente, inizia il ciclo di generazione dei casi di test.
 - Viene avviato il ciclo di testing della classe che termina se $iter > max_iter$ o $sat > max_sat$ o $livello > maxlevel$.

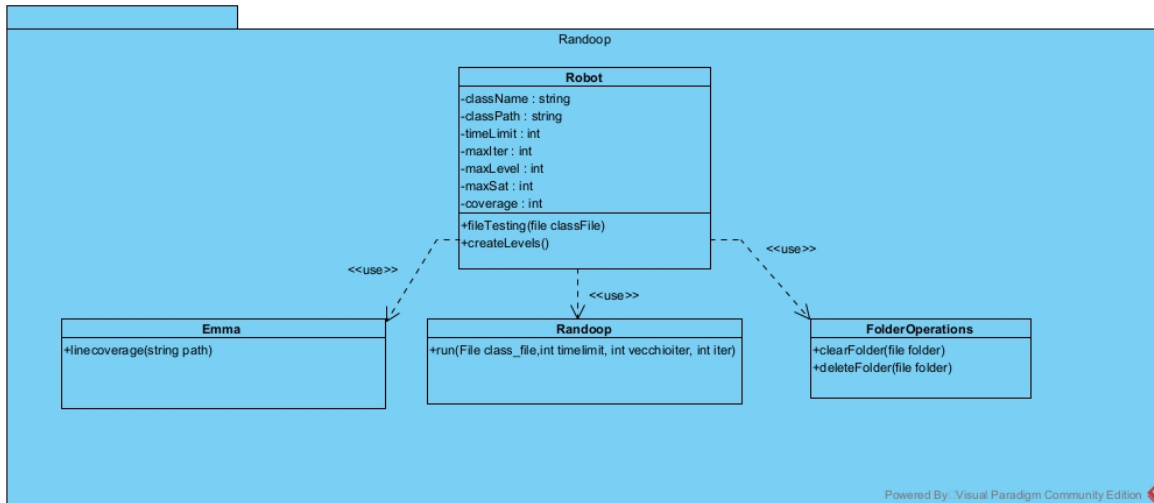
Durante il secondo ciclo:

- Viene chiamato il file batch per la generazione e l'instradamento dei test case della classe tramite Emma e Randoop.
- Il robot chiama `lineCoverage()` della classe EMMA che permette di estrarre il valore di copertura utilizzato per la valutazione della creazione di un nuovo livello. L'estrazione della coverage viene effettuata ricorrendo all'ausilio della libreria JSOUP a partire dal file .xml della precedente operazione. La funzionalità *emmerge* permette di aggregare le coverage.
- Un nuovo livello viene generato nel filesystem se la coverage raggiunta è maggiore rispetto alla coverage precedente.

In conclusione:

- Le cartelle vengono pulite, rimuovendo i file e le directory create durante il processo di testing.

Package Diagram



Il **diagramma dei package** è uno strumento di modellazione che consente di visualizzare la struttura organizzativa del software in base ai package e alle relazioni tra di essi. Aiuta a comprendere l'organizzazione delle classi e dei moduli all'interno di un sistema, consentendo una migliore comprensione dell'architettura complessiva.

Il diagramma dei package è strutturato nelle seguenti classi:

- **Robot**: La classe Robot è una classe che coordina le operazioni e fornisce all'utente il punto d'avvio del processo. Quando l'applicazione viene lanciata, la creazione dei casi di test prende il via attraverso l'invocazione dei metodi di questa classe. Contiene altresì tutti gli attributi necessari all'elaborazione e alla valutazione della coverage ai fini della creazione dei livelli.
- **FolderOperations**: Tale classe contiene metodi utili per gestire le operazioni di cancellazione di file e cartelle.
- **Randoop**: Contiene il metodo statico Run(File class_file, int timelimit, int vecchioiter, int iter) per eseguire il modulo Randoop.
- **Emma**: Contiene il metodo statico LineCoverage(String path) per estrarre il valore della copertura delle linee da un file XML di copertura generato da Emma.

Soluzioni alternative

Una possibile soluzione alternativa è basata sullo sviluppo di API con l'ausilio della tecnologia Spring Boot. In una prima fase del progetto questa soluzione era stata approcciata, con implementazione del pattern MVC, testando i metodi esposti tramite Postman. In particolare, sono state implementate le funzionalità di ricezione di una classe mediante chiamata di tipo POST e generazione dei casi di test sul filesystem.

Tuttavia, confrontandoci durante le iterazioni è emerso che il nostro task è fortemente "locale", sia nelle dipendenze che nell'obiettivo, si è pensato di non percorrere questa strada (anche in ottica di riduzione del

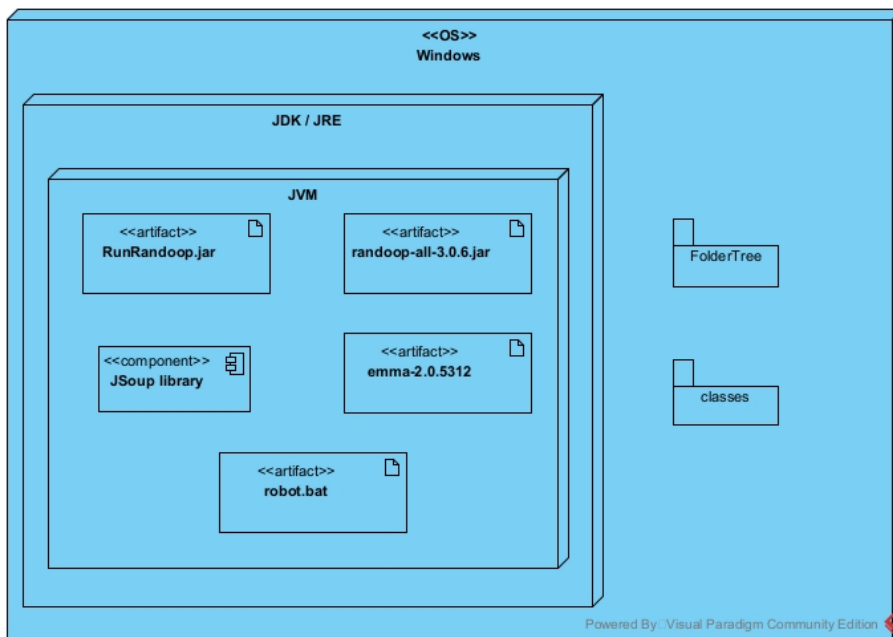
consumo di risorse e banda, in quanto l'applicazione sarebbe dovuta risultare sempre attiva per questioni di raggiungibilità).

Inoltre, si è notato che il pattern MVC non poteva essere applicato al nostro progetto, in quanto, non si disponeva di una view.

Capitolo 5: Documentazione di implementazione

Deployment Diagram

Il diagramma descrive il contesto software di esecuzione.



Il diagramma di deployment illustra la distribuzione fisica dei componenti software e delle risorse hardware per il nostro sistema.

In questo contesto, mostriamo come la nostra applicazione Java sarà eseguita su un sistema Windows tramite la Java Virtual Machine (JVM).

Sistema Windows: Questo componente rappresenta il sistema operativo host su cui verrà eseguita la nostra applicazione. Il Sistema Windows è l'ambiente principale in cui si trova la nostra soluzione locale.

Java Virtual Machine (JVM): La JVM è l'ambiente di esecuzione per la nostra applicazione Java. Rappresenta il substrato su cui viene eseguito il codice Java. La JVM è connessa al Sistema Windows, tramite jdk e jre.

RunRandoop.jar: Artifact eseguibile dall'utente.

Emma-2.0.5312: Tool di Emma eseguito all'interno dell'applicazione.

Randoop-all-3.0.6.jar: Tool di randoop eseguito all'interno dell'applicazione.

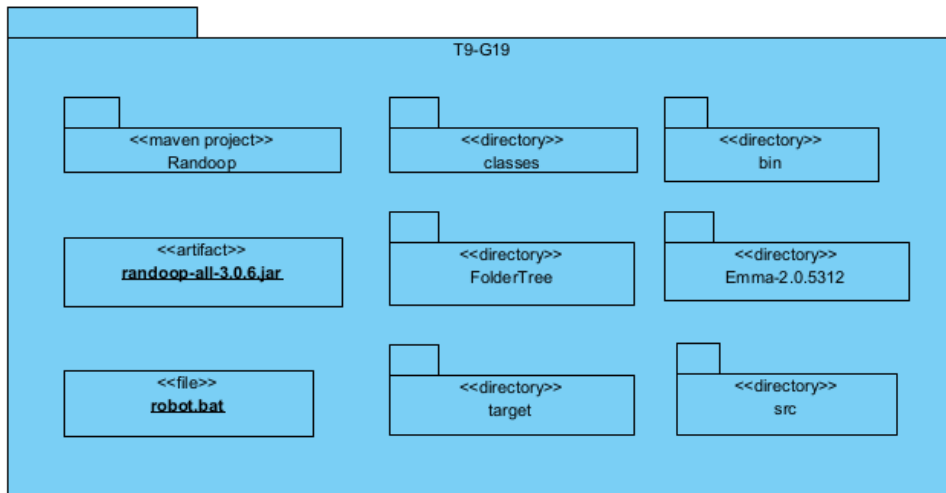
Robot.bat: File eseguibile batch che richiama i tool necessari per la generazione dei test.

FolderTree: Filesystem in cui sono inserite le classi da testare ed i test generati suddivisi in livelli.

Classes: Folder di appoggio utilizzata durante la generazione dei test.

Struttura Repository

Di seguito è possibile visualizzare la struttura complessiva del nostro repository per singoli elementi.



Installazione

L'installazione dell'applicativo **Robot Randoop** è stata effettuata sulla macchina locale relativa al progetto, questo poiché il robot dovrà generare file di test in maniera *offline*, senza dunque la necessità di utilizzare un Docker. Proprio per questo sono state eliminate le chiamate API realizzate nelle scorse iterazioni, e sostituite da un "input" dell'admin.

Prerequisiti: sulla macchina deve essere stata installata la versione 1.8 della JRE e della JDK. Questo è necessario perché Emma funziona solamente con versioni di Java minori o pari alla 8. Per procedere con l'installazione è necessario scaricare il repository sulla propria macchina e spaccettarlo. In seguito, è necessario settare due variabili d'ambiente all'interno del file *batch* "**robot.bat**":

- `set java1.8_path`
- `set javac`

```
REM Percorso java1.8 da modificare per ogni computer
set java1.8_path= "C:\Program Files (x86)\Java\jre1.8.0_351\bin\java.exe"

REM Percorso java1.8 da modificare per ogni computer
set javac= "C:\Program Files\Java\jdk1.8.0_202\bin\javac.exe"
```

Successivamente bisognerà soltanto utilizzare l'applicativo.

Esempio di utilizzo

Dopo aver installato correttamente l'ambiente, l'admin potrà iniziare con la generazione dei test; infatti, una volta che apriremo la cartella, potremo utilizzare l'artefatto **RunRandoop.jar** per iniziare l'esecuzione dei test.

Tale file verrà avviato solo nel caso in cui ci fossero classi, appartenenti alla cartella FolderTree, che non siano ancora state testate. Dunque, nel caso in cui si volessero generare test su di una nuova classe, basterà inserirla all'interno di FolderTree, nell'apposita cartella (secondo lo schema dell'albero).


classes	14/07/2023 15:56	Cartella di file	
emma-2.0.5312	14/07/2023 11:45	Cartella di file	
FolderTree	14/07/2023 11:45	Cartella di file	
src	14/07/2023 11:45	Cartella di file	
target	14/07/2023 11:45	Cartella di file	
pom.xml	14/07/2023 11:44	File XML	2 KB
randoop-all-3.0.6.jar	14/07/2023 11:44	File JAR	12.105 KB
robot.bat	14/07/2023 15:49	File batch Windows	4 KB
RunRandoop.jar	14/07/2023 11:44	File JAR	421 KB

Una volta inserita una nuova classe, per far partire l'esecuzione bisognerà utilizzare dunque RunRandoop.jar. Il programma avvierà uno script da linea di comando che inizierà a generare test case per *ogni* classe non ancora testata, producendo vari livelli di difficoltà in base alla coverage generata da **EMMA**.

```
Robot randoop script
---CREAZIONE CARTELLA INSTRADAMENTO SE NON ESISTENTE---
Sottodirectory o file .\classes già esistente.
---COMPILAZIONE---
Note: classes\XMLParser.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
---INSTRUMENTAZIONE---
EMMA: processing instrumentation path ...
EMMA: instrumentation path processed in 24 ms
EMMA: [1 class(es) instrumented, 1 resource(s) copied]
EMMA: metadata merged into [C:\Users\Ricky\Desktop\Cartella_Esami_Vari\MAGISTRALE\Software Architecture Design\Progetto-
SAD-G19-main\Progetto-SAD-G19-master\FolderTree\XMLParser\RobotTest\RandoopTest\XMLParser-2-dati_di_copertura\coverage.e
m] {in 0 ms}
18603
---ESEGUI LA SESSIONE---
policy = sun.security.provider.PolicyFile@20c684
EMMA: collecting runtime coverage data ...
PUBLIC MEMBERS=9
Explorer = randoop.generation.ForwardGenerator@a8f0b4
```

```
Directory --> VCardBean
Directory --> XmlElement
Directory --> XMLParser
Generazione test per la classe --> XMLParser
Notifica: Il file batch è stato eseguito con successo.
new level--> coverage: 37
File copiato: RegressionTest_it0_livello1.java
File copiato: RegressionTest0_it0_livello1.java
Notifica: Il file batch è stato eseguito con successo.
Notifica: Il file batch è stato eseguito con successo.
Notifica: Il file batch è stato eseguito con successo.
```

Al termine dell'esecuzione, potremo recarci all'interno di FolderTree/className/RobotTest/RandoopTest e visualizzare il risultato di tutti i livelli che sono stati generati da Randoop.



- 01Level
- 02Level
- 03Level

Ogni cartella, ordinata per livello di difficoltà crescente, conterrà RegressionTest diversi tra di loro grazie all'operazione di merge (contenuta in Emma) e con una copertura man mano sempre maggiore, proprio per rendere più difficoltosa la sfida contro il Robot.



- RegressionTest_it0_livello1_it1_livello2_it3_livello...
- RegressionTest_it0_livello1_it1_livello2_it4_livello...
- RegressionTest_it1_livello2_it3_livello3.java
- RegressionTest_it1_livello2_it4_livello3.java
- RegressionTest_it3_livello3.java
- RegressionTest_it4_livello3.java
- RegressionTest0_it0_livello1_it1_livello2_it3_livell...
- RegressionTest0_it0_livello1_it1_livello2_it4_livell...
- RegressionTest0_it1_livello2_it3_livello3.java
- RegressionTest0_it1_livello2_it4_livello3.java
- RegressionTest0_it3_livello3.java
- RegressionTest0_it4_livello3.java