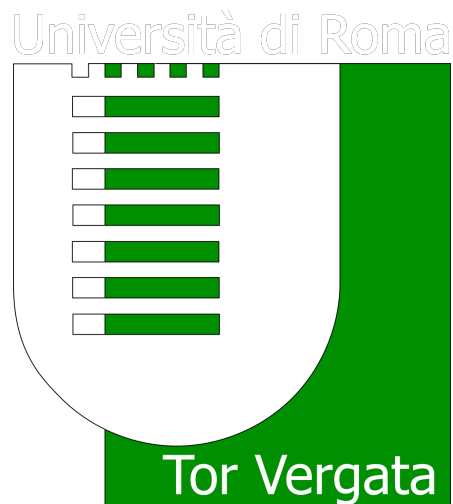


Università di Roma Tor Vergata

Dipartimento di Ingegneria Informatica



Relazione ISW2 sul modulo di testing

Prof.
Guglielmo De Angelis

Roma
7 febbraio 2022

Indice

1	Progetto 1+	2
1.1	Apache JCS 1.3	2
1.1.1	ParametricJCSLightLoadUnitTest	2
1.1.2	ParametricJCSRemovalSimpleConcurrentTest	3
2	Progetto 1+2	4
2.1	Bookkeeper	4
2.1.1	ReadCache	5
2.1.2	WriteCache	6
2.2	OpenJPA	7
2.2.1	QualifiedDBIdentifier	8
2.2.2	ClassUtil	8
3	Utili	11
4	Immagini	12
4.1	Progetto 1+	12
4.2	Progetto 1+2	15
4.2.1	Bookkeeper - ReadCache	15
4.2.2	Bookkeeper - WriteCache	18
4.2.3	Bookkeeper - TravisCI - SonarCloud	21
4.2.4	OpenJPA - QualifiedDBIdentifier	22
4.2.5	OpenJPA - ClassUtil	23
4.2.6	OpenJPA - TravisCI - SonarCloud	25

1 Progetto 1+

1.1 Apache JCS 1.3

L'obiettivo di questa prima parte del progetto della parte di software testing del corso è quello di lavorare sul progetto open source Apache JCS (distributed caching system scritto in Java) nella sua versione 1.3.

Per svolgere il seguente progetto è stata quindi la copia dei sorgenti ed usando l'algoritmo proposto a lezione sono state selezionate le classi di interesse: **JCSLightLoadUnitTest** e **JCSRemovalSimpleConcurrentTest**.

Per ognuna delle due classi di test è stata implementata una suite di test basata su JUnit4 mediante la dichiarazione di test parametrici che usassero almeno un metodo "configure" per legare i parametri passati con il runner Parametrized o il costruttore.

L'approccio utilizzato è stato quello di eliminare i test che non riguardavano la nostra analisi e di lasciare i due test da analizzare da cui poi si è partiti per eseguirne una versione parametrizzata. Le rispettive nuove versioni sono state definite **ParametricJCSLightLoadUnitTest** e **ParametricJCSRemovalSimpleConcurrentTest**.

Per quanto riguarda l'approccio al testing, invece, in questa sezione è stato utilizzato un metodo completamente white-box ovvero ci siamo serviti del codice sorgente stesso per creare criteri di selezione per i test.

Lo stack di framework utilizzati è stato quello di Git per il source control, di Maven per quanto riguarda la build automation per gestione il nostro progetti scritto in Java, di TravisCI che ci mette a disposizione un build server per la continuous integration e per la coverage la libreria di Java JaCoCo i cui dati possono essere analizzati in remoto da SonarCloud.

1.1.1 ParametricJCSLightLoadUnitTest

Questa classi di testing, la JCSLightLoadUnitTest, si eseguire attraverso il metodo testSimpleLoad() degli inserimenti in una cache di test istanziata.

Nel nostro caso il valore degli item che ci permetteva di eseguire correttamente il test è 999, elementi inizializzati in un array nel metodo data() richiamato nel metodo configure() che vengono passati al test mediante il tagging del metodo con la keyword @Parameters.

Il testSimpleLoad() quindi si occupa di fare i 999 inserimenti (la put) di cui uno nullo e viene poi eseguito il controllo della presenza di un elemento con indice nullo ed il relativo, corretto, messaggio di errore in quanto un evento del genere non può accadere.

Nella parte finale poi ci occupiamo della rimozione degli elementi in cache operando quindi l'eliminazione di un item che ha come chiave 300. La corretta eliminazione viene certificata mediante una get dell'elemento appena eliminato che ci restituisce, correttamente, il valore null.

Con questo numero di item, con questa logica di put, remove e get è stato raggiunto il 100% di coverage della classe di test, come viene certificato nelle immagini 1 e 2, migliorando il primo valore che era pari a 89,3%.

1.1.2 ParametricJCSRemovalSimpleConcurrentTest

Nel caso della seconda classe di test del progetto preso in esame, la JCSRemovalSimpleConcurrentTest, si sono istanziati 500 elementi sempre nella funzione data() passati come parametri al test e inizializzati nella classe configure insieme all'istanziamento della cache di test.

Questa serie di oggetti viene poi passata ad un suite di test composta da 4 metodi:

1. testTwoDeepRemoval()
in questo primo metodo di test vengono effettuati 500 inserimenti di secondo livello, di cui uno nullo per controllare la logica riservata a questo caso e soprattutto la logica che si occupa di eseguire le rimozioni (la remove) degli elementi appartenenti al secondo livello con le informazioni relative alle operazioni eseguite.
2. testSingleDepthRemoval()
in questa seconda funzione viene posta l'accezione sulla rimozione dei 500 elementi inseriti in una cache a singola profondità.
3. testClear()
nel terzo metodo di test invece controlliamo che l'operazione di clear elimini correttamente tutti i 500 elementi presenti (che sono stati nuovamente inseriti) attraverso la combinazione di put, clear e get.
4. testClearRepeatedlyWithoutError()
nell'ultimo metodo invece si esegue la clear ciclicamente nel for per testare la corretta esecuzione della clear se eseguita più volte su rispettivi inserimenti.

Anche in questo caso, come nel precedente, si è cercato di ottimizzare il numero degli items e si è modificato il test affinché la coverage fosse la più alta possibile. Con questo procedimento siamo passati dal 85,9% iniziale al 95,5% finale.

2 Progetto 1+2

Per quanto riguarda la seconda parte del progetto di software testing si ha come obiettivo l'analisi di due progetti open source di Apache: Bookkeeper e OpenJPA; per ognuno dei quali sono state individuate due classi da testare.

Su ogni classe è stato definito un insieme di test da category partition ed i test sono stati inclusi nel ciclo di build del progetto che è stato integrato con i tool di TravisCI e SonarCloud.

Arrivati a questo punto si considera la bontà del nostro lavoro mediante l'utilizzo di due metriche di accuratezza per i test in questione.

Vengono poi infine applicati a questi le mutazioni per stimare la robustezza dei test sviluppati.

Gli step con i quali ci si è approcciati al lavoro sono:

- **Cloning della repository del progetto in locale.**

Questo step è fondamentale in quanto il progetto Maven doveva prima di tutto eseguire il build ed i test in locale.

Sono stati quindi ripulite tutte le cartelle dei test non di nostro interesse facendo attenzione a non rimuovere risorse o linking utili per la creazione corretta dell'ambiente.

- **Scrittura dei test.**

I test sono stati scritti seguendo la logica di renderli parametrici, utilizzando il metodo configure e sono basati su JUnit.

- **Building remoto.**

Nel momento in cui il progetto gira correttamente in locale, la repository GitHub è stata linkata con TravisCI per il building remoto e SonarCloud per l'analisi dei test e della coverage.

- **Analisi della coverage e creazione delle mutazioni.**

Questo step è stato eseguito sui test utilizzando rispettivamente i framework JaCoCo e PiTest la cui analisi ci ha permesso di arrivare alle nostre conclusioni e soprattutto per tarare bene i test.

2.1 Bookkeeper

Per questo primo progetto da analizzare sono state scelti, utilizzando le informazioni ottenute dalla documentazione ufficiale e dai dati ottenuti in output dalla deliverable 2, i metodi delle rispettive classi:

- **ReadCache**

- count()
- size()
- get(long ledgerId, long entryId)
- put(long ledgerId, long entryId, ByteBuf entry)
- close()

- **WriteCache**

- isEmpty()
- count()
- size()
- put(long ledgerId, long entryId, ByteBuf entry)
- getLastEntry()
- clear()
- close()
- forEach()
- deleteLedger()

Queste due classi sono entrambe presenti al path: BookKeeper/bookkeeper-test/bookkeeper-server/src/main/java/org/apache/bookkeeper/bookie/storage/ldb.

Per quanto riguarda il testing di Bookkeeper si è scelto di utilizzare i tool forniti da Eclipse, diversamente di come si è scelto di fare con OpenJPA che analizzeremo in seguito.

2.1.1 ReadCache

Questo metodo utilizza la quantità di memoria specificata e la accoppia con una hashmap. La memoria è suddivisa in più segmenti che vengono utilizzati secondo il concetto ring-buffer. Quando la cache di lettura è piena, il segmento più vecchio viene cancellato e ruotato per fare spazio alle nuove voci da aggiungere alla cache di lettura.

Per testare il funzionamento della ReadCache sono stati implementati i due metodi di test presenti nella MyReadCacheTest:

- **checkEmptyAndReadingCacheSingleSegmentTest**

in questo test viene eseguito un controllo sull'inizializzazione di una cache che deve essere effettivamente vuota, ovvero la lettura di una cache vuota. In seguito inseriamo un buffer come da parametri e viene eseguito il controllo di lettura di una cache non più vuota. Questo tipo di test viene eseguito un'ulteriore volta ma in questo caso all'interno di un ciclo. Infine eseguiamo il rilascio della risorsa.

- **checkEmptyAndReadingCacheMultiSegmentTest**

per quanto riguarda questo test vengono eseguite controlli di letture, in seguito quindi a scritture di test, sulla cache vuota o piena ma questa volta la risorsa che andiamo a creare è multilivello. Anche in questo caso vengono infine rilasciate tutte le risorse utilizzate.

Nel metodo configure della classe MyReadCacheTest si trovano gli input selezionati per eseguire i test presenti.

UnpooledByteBufAllocator.DEFAULT, 10*1024, 1024, 2*1024

UnpooledByteBufAllocator.DEFAULT, 10*512, 512, 2*512

UnpooledByteBufAllocator.DEFAULT, 10*2048, 2048, 2*2048

Questa è la suite di parametri viene utilizzata per i due test presenti che sono stati fatti girare utilizzando il tool integrato all'interno di Eclipse di JUnit.

La ricerca per la suite migliore è ricaduta dal secondo parametro in poi, attraverso questi 3 input siamo riusciti subito ad arrivare ad una copertura della classe da testare ad una copertura del **100%** finale, valore che è ritenuto ovviamente più che soddisfacente con 357 su 357 istruzioni e statement della ReadCache coperti. Siamo arrivati a questi valori eseguendo i test JUnit con il goal della coverage sempre all'interno dell'software Eclipse.

Per quanto riguarda la parte delle mutazioni, in questo caso, è stato scaricato e provato il plug-in di Eclipse definito PIT Mutations. Questo esegue un'analisi dei test sviluppati e produce i risultati sia a console dei mutanti eseguiti che il file html di summary della coverage dopo le mutazioni di Pit Test. I risultati ottenuti da questa analisi parlano di 5 mutanti sopravvissuti e 34 uccisi

2.1.2 WriteCache

Il metodo in questione allocherà la dimensione richiesta dalla memoria e la suddividerà in più segmenti. Le voci vengono aggiunte in un buffer e indicizzate tramite una hashmap, fino a quando la cache non viene cancellata.

C'è inoltre la possibilità di scorrere le voci memorizzate in modo ordinato, per (ledgerId, entry).

Per testare il funzionamento della WriteCache sono stati implementati i due metodi di test presenti nella MyWriteCacheTest:

- **checkEmptyAndWriteCacheSingleSegmentTest**
in questo metodo viene eseguito il controllo dell'inizializzazione della cache creata per la scrittura e poi vengono eseguite delle scritture su una cache a singolo livello con i parametri definiti nel metodo configure. Vengono infine liberate le risorse impiegate nel test e viene eseguito un controllo anche su questa parte della procedura della WriteCache.
- **fillCacheFullTest**
questo test si occupa di eseguire delle scritture affinché la cache venisse riempita completamente e quindi il testing di questa parte della procedura presente nella WriteCache.

Nel metodo configure della classe MyWriteCacheTest si trovano gli input selezionati per eseguire i test presenti.

UnpooledByteBufAllocator.DEFAULT, 10*1024,1024
UnpooledByteBufAllocator.DEFAULT, 1000 * 1024,1024
UnpooledByteBufAllocator.DEFAULT, 10*1024,512
UnpooledByteBufAllocator.DEFAULT, 10*1024,2048

Questa suite di parametri è stata ottimizzata per portare la copertura dal 62.2% al **93.5%** valore che è ritenuto quindi soddisfacente considerando quindi soddisfacente la suite di parametri per il test eseguiti utilizzando il tool integrato all'interno di Eclipse di JUnit.

Consideriamo ora le mutazioni, in questo caso come per il precedente è stato utilizzato il plug-in PIT Mutations. I risultati ottenuti da questa analisi parlano di 17 mutanti sopravvissuti e 63 uccisi.

2.2 OpenJPA

Per quanto riguarda il secondo progetto da analizzare le classi che sono ricadute sotto la scelta di:

- **QualifiedDBIdentifier**
classe presente nel path:
OpenJPA/openJPA-test/openjpa-jdbc/src/main/java/org/apache/openjpa/jdbc/identifier

Di cui si è andato a testare il metodo di `splitPath(DBIdentifier sName)` nella **MyQualifiedDBIdentifierTest**.

- **ClassUtil**

classe presente nel path:

OpenJPA/openJPA-test/openjpa-lib/src/main/java/org/apache
/openjpa/lib/util/

Di cui si è andato a testare il metodo di `getClassName(String fullName)` nella **MyClassUtilTest** mentre il `getPackageName(String fullName)` nella **MyClassUtilSecondTest**.

Per questo secondo progetto ho utilizzato come indicato il tool JaCoCo per il reporting della coverage ed quello di PiTest per le mutazioni direttamente all'interno del progetto Maven indicando nei pom i plug-in da utilizzare e nel nello yml i comandi da eseguire nel building. Vedremo in seguito i risultati di ciascuna esecuzione.

2.2.1 QualifiedDBIdentifier

Questa classe estende `DBIdentifier` per fornire supporto per identificatori qualificati come tabelle qualificate schema e colonne qualificate tabella. Fornisce metodi per creare identificatori qualificati da identificatori individuali.

Nella nostra classe di test, la `MyQualifiedDBIdentifier`, infatti si è testata la corretta esecuzione del metodo `splitPath`, che si occupa di prendere un path e di restituire gli identificatori che compongono quest'ultimo, con i seguenti parametri:

```
null
DBIdentifier.newTable("Schema.Table")
QualifiedDBIdentifier.newPath( DBIdentifier.NULL )
```

I rispettivi valori dei casi di test dovevano restituire: **false**, **true** e **false**.

2.2.2 ClassUtil

Questa classe si occupa di restituire la classe per la stringa data, gestendo correttamente i tipi primitivi. Se il caricatore di classi specificato è null, verrà utilizzato il caricatore di contesto del thread corrente.

Si è scelto di eseguire il testing dei due metodi della classe `ClassUtil` in due classi separate per continuare ad utilizzare l'approccio che utilizzava il metodo `configure` per il settaggio dei parametri di esecuzione del test e questi due metodi hanno bisogno di due tipologie di parametri differenti.

- `MyClassUtilTest`

In questa prima classe test i parametri passati per cercare di ottimizzare la nostra analisi sono:

```
MyClassUtilTest.class.toString()  
null  
””
```

Il test quindi si occuperà di eseguire un controllo sul tipo restituito dal metodo `getClassName` se è corretto o meno a cui sono stati passati i precedenti parametri.

- `MyClassUtilSecondTest`

Mentre per la seconda classe di test i parametri utilizzati per eseguire il testing del metodo della `ClassUtil` sono:

```
MyClassUtilSecondTest.class.toString()  
MyInnerClass.class.toString()  
MyInnerClass[].class.toString()  
INSTANCE.getClass().toString()  
long.class.toString()  
long[].class.toString()
```

Questo secondo test invece si occupa di testare, utilizzando degli identificativi per ogni caso di test, se il metodo `getPackageName` restituisce il package atteso nei diversi casi.

Per quanto riguarda l'analisi dei test in questa seconda parte abbiamo per l'insieme dei test utilizzati nella `MyQualifiedDBIdentifierTest` la copertura del **52%** per quanto riguarda la metrica delle missed instruction e quella del **40%** se consideriamo quella dei missed branches sulla classe `QualifiedDBIdentifier` che considerato che abbiamo testato solo un metodo di questa è un risultato più che accettabile.

Per quanto riguarda invece l'analisi con le mutazioni sempre su questa classe abbiamo ottenuto un valore di test strength del **68%**.

Andando invece a sviscerare i dati ottenuti dal testing della classe `ClassUtil` per l'insieme dei test scelti sottoposti nella `MyClassUtilTest` e `MyClassUtilSecondTest` la copertura del **86%** per quanto riguarda la metrica delle missed instruction e quella del **76%** se consideriamo quella dei missed branches sulla classe in questione.

Andando ad analizzare la coverage ottenuta dall'esecuzione delle mutazioni otteniamo anche per questa il valore di test strenght pari al **68%** con il valore 15/22.

3 Utili

Link:

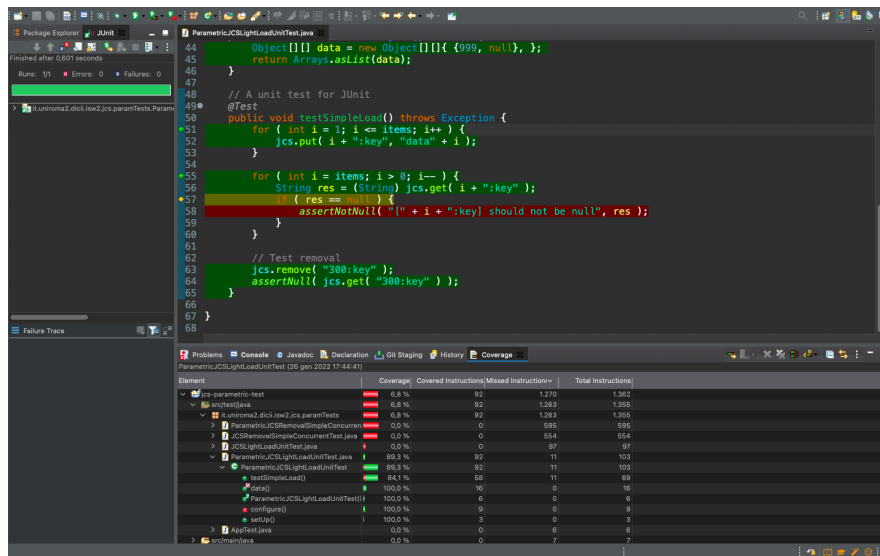
- Github:
 - <https://github.com/danielemariano/mavenTravisJcsParametrizedTests>
 - <https://github.com/danielemariano/bookkeeper-test>
 - <https://github.com/danielemariano/openJPA-test>
- SonarCloud:
 - https://sonarcloud.io/project/overview?id=danielemariano_mavenTravisJcsParametrizedTests
 - https://sonarcloud.io/project/overview?id=danielemariano_bookkeeper-test
 - https://sonarcloud.io/project/overview?id=danielemariano_openJPA-test
- TravisCI:
 - <https://app.travis-ci.com/github/danielemariano/mavenTravisJcsParametrizedTests>
 - <https://app.travis-ci.com/github/danielemariano/bookkeeper-test>
 - <https://app.travis-ci.com/github/danielemariano/openJPA-test>

Comandi:

- Test:
 - `mvn -U clean install`
 - `mvn clean test`
- Coverage:
 - `mvn clean org.jacoco:jacoco-maven-plugin:prepare-agent install`
 - `mvn jacoco:report`
- Mutation:
 - `mvn org.pitest:pitest-maven:mutationCoverage surefire:test`

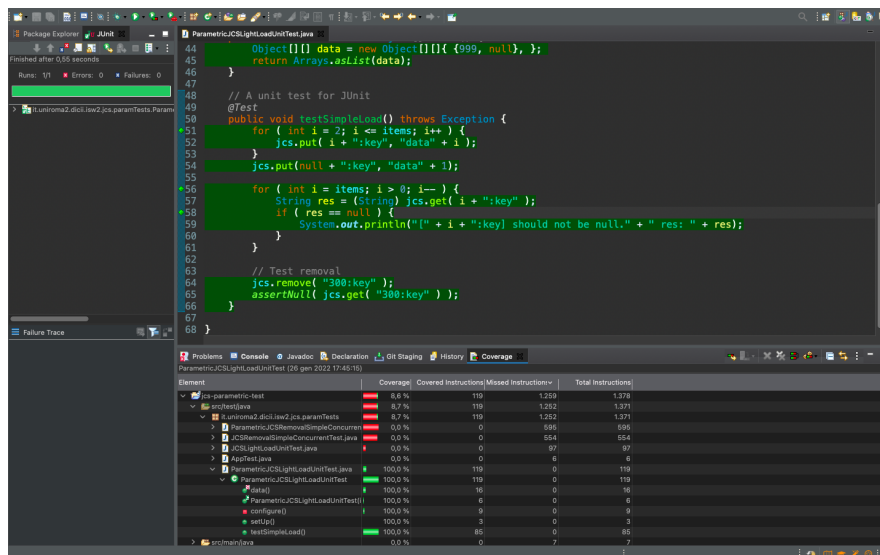
4 Immagini

4.1 Progetto 1+



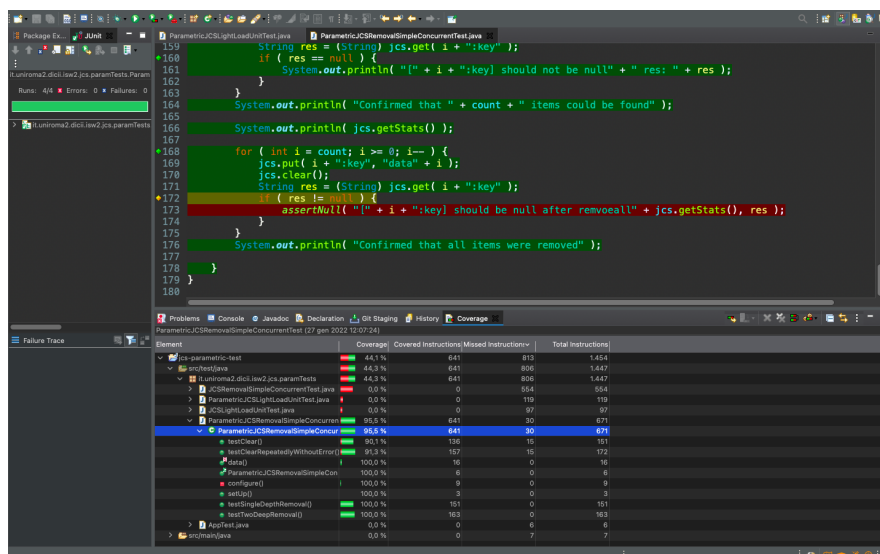
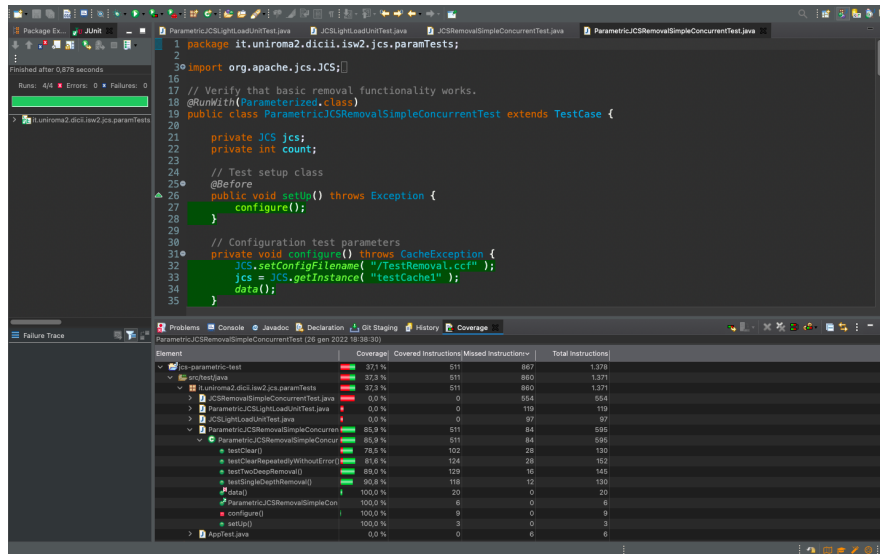
The screenshot shows an IDE with a Java unit test file named `ParametricJCSLightLoadUnitTest.java`. The test includes a `testSimpleLoad` method that uses `JUnit` annotations and assertions. The coverage report at the bottom shows the following data:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jcs-parametric-test	6.8 %	92	1,270	1,362
testTestJava	6.8 %	92	1,263	1,355
ParametricJCSRemovalSimpleConcurrentTest	0.0 %	0	595	595
JCSLightLoadUnitTest	0.0 %	0	554	554
ParametricJCSLightLoadUnitTest	89.3 %	92	11	103
testSimpleLoad	84.1 %	58	11	69
data()	100.0 %	16	0	16
ParametricJCSLightLoadUnitTest()	100.0 %	6	0	6
configure()	100.0 %	9	0	9
setUp()	100.0 %	3	0	3
AppTestJava	0.0 %	0	6	6
srcMainJava	0.0 %	0	7	7



The screenshot shows the same IDE with a modified version of the `ParametricJCSLightLoadUnitTest.java` file. The `testSimpleLoad` method has been updated to include a `System.out.println` statement. The coverage report at the bottom shows the following data:

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
jcs-parametric-test	8.6 %	119	1,259	1,378
testTestJava	8.7 %	119	1,252	1,371
ParametricJCSRemovalSimpleConcurrentTest	0.0 %	0	595	595
JCSLightLoadUnitTest	0.0 %	0	554	554
ParametricJCSLightLoadUnitTest	100.0 %	119	0	119
testSimpleLoad	100.0 %	119	0	119
data()	100.0 %	16	0	16
ParametricJCSLightLoadUnitTest()	100.0 %	6	0	6
configure()	100.0 %	9	0	9
setUp()	100.0 %	3	0	3
AppTestJava	0.0 %	0	6	6
srcMainJava	0.0 %	0	7	7



[danielemariano](#) / [jcs-parametric-test](#) Not computed

PUBLIC

Last analysis: January 27, 2022, 1:23 PM

0 A

Bugs

0 A

Vulnerabilities

100% A

Hotspots Reviewed

0 A

Code Smells

0.0%

Coverage

0.0%

Duplications

129 XS

XML, Java

✓ [mavenTravisJcsParametrize](#)

LAST BUILD

21

DEFAULT BRANCH

↔ main

COMMIT

62ec964

FINISHED

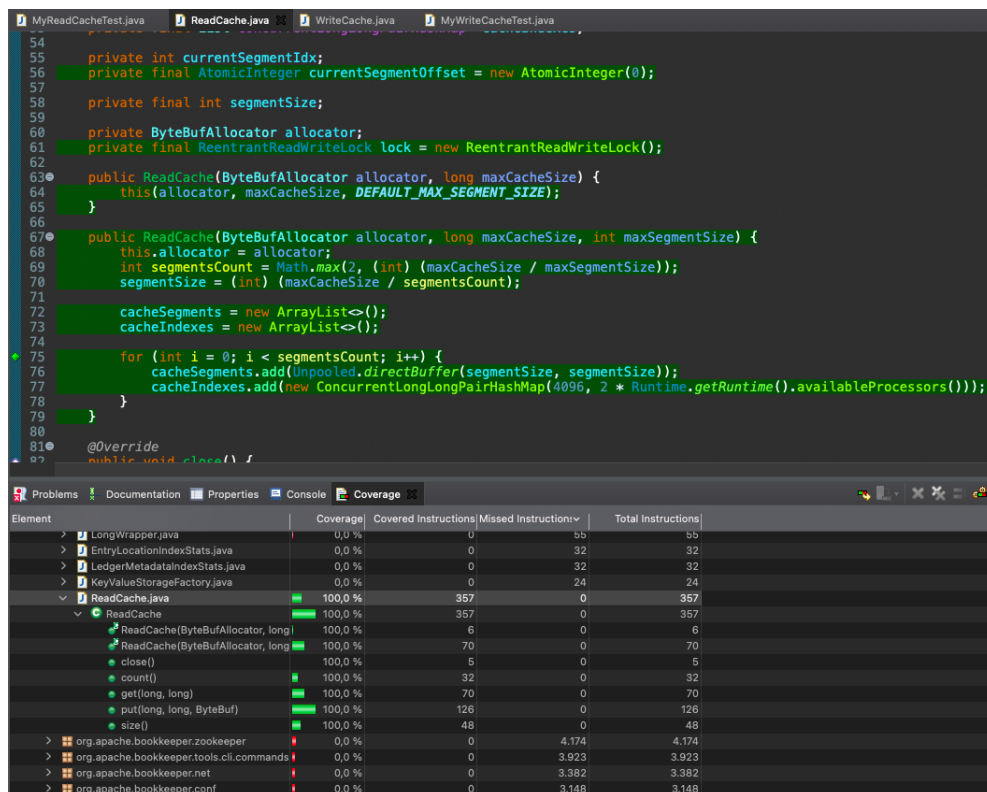
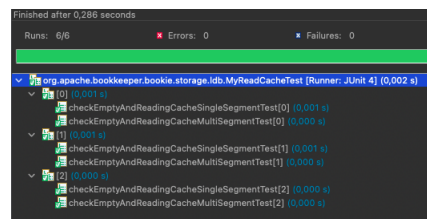
Passed 2 days ago

4.2 Progetto 1+2

In aggiunta alla discussione formale e teorica di quello che si è fatto e di quali sono state le scelte per lo sviluppo del progetto si è aggiunta una sezione di immagini.

Questa sezione va ad aggiungere capitolo per capitolo informazioni visive circa l'analisi appena conclusa, completando il quadro dei valori ottenuti dalle varie esecuzioni di ciascun progetto utilizzando i diversi tool alleggerendo così la discussione precedente.

4.2.1 Bookkeeper - ReadCache

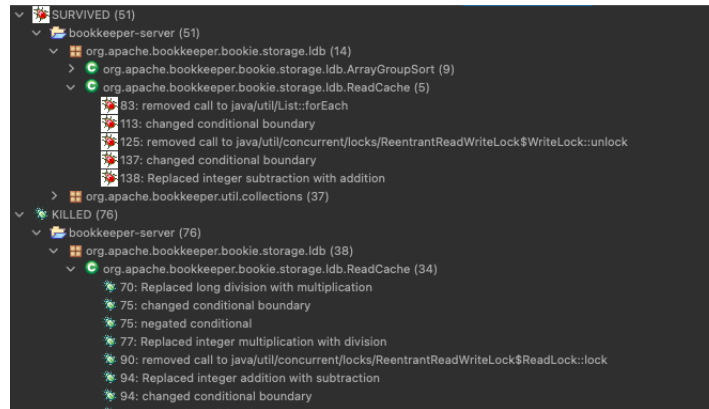


Project Summary

Number of Classes	Line Coverage	Mutation Coverage
358	1% 170/27980	1% 89/12778

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
org.apache.bookkeeper.auth	3	0% 0/49	0% 0/32
org.apache.bookkeeper.bookie	54	0% 0/5502	0% 0/2394
org.apache.bookkeeper.bookie.stats	2	0% 0/23	0% 0/5
org.apache.bookkeeper.bookie.storage ldb	16	7% 77/1140	8% 47/619



Mutations

69	1. Replaced long division with multiplication - TIMED_OUT
70	1. Replaced long division with multiplication - KILLED
75	1. changed conditional boundary - KILLED
75	2. Changed increment from 1 to -1 - MEMORY_ERROR
77	3. negated conditional - KILLED
77	1. Replaced integer multiplication with division - KILLED
83	1. removed call to java/util/ArrayList::forEach - SURVIVED
90	1. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::lock - KILLED
94	1. changed conditional boundary - KILLED
94	2. Replaced integer addition with subtraction - KILLED
94	3. negated conditional - KILLED
104	1. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::unlock - TIMED_OUT
104	2. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::unlock - NO_COVERAGE
104	3. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::unlock - TIMED_OUT
109	1. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::lock - KILLED
113	1. changed conditional boundary - SURVIVED
113	2. Replaced integer addition with subtraction - KILLED
113	3. negated conditional - KILLED
115	1. Replaced integer addition with subtraction - KILLED
116	2. Replaced integer modulus with multiplication - KILLED
116	1. removed call to java/util/concurrent/atomic/AtomicInteger::set - KILLED
117	1. removed call to org/apache/bookkeeper/util/collections/ConcurrentLongLongPairHashMap::clear - KILLED
125	1. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::unlock - NO_COVERAGE
125	2. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::unlock - SURVIVED
130	1. removed call to java/util/concurrent/locks/ReentrantReadWriteLock::lock - KILLED
137	1. changed conditional boundary - SURVIVED
137	2. Changed increment from 1 to -1 - TIMED_OUT
137	3. negated conditional - KILLED
138	1. Replaced integer subtraction with addition - SURVIVED
138	2. Replaced integer addition with subtraction - KILLED
141	3. Replaced integer modulus with multiplication - KILLED
141	1. negated conditional - KILLED
147	1. Replaced return value with null for org/apache/bookkeeper/bookie/storage/ldb/ReadCache::get - KILLED

Pit Test Coverage Report

Package Summary

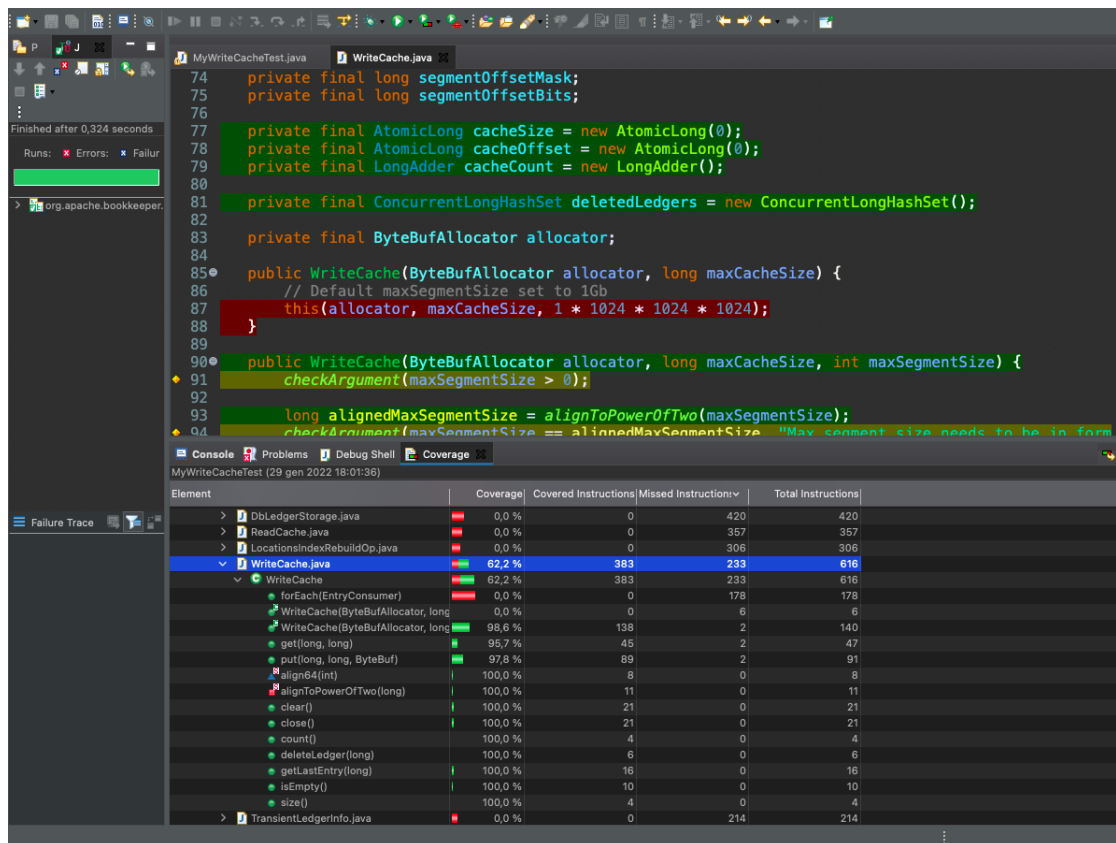
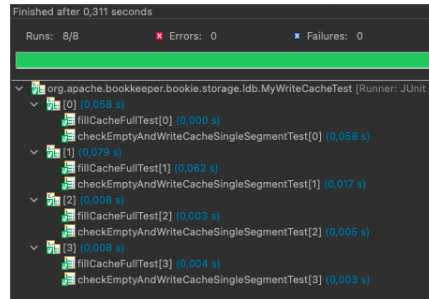
org.apache.bookkeeper.bookie.storage.Idb

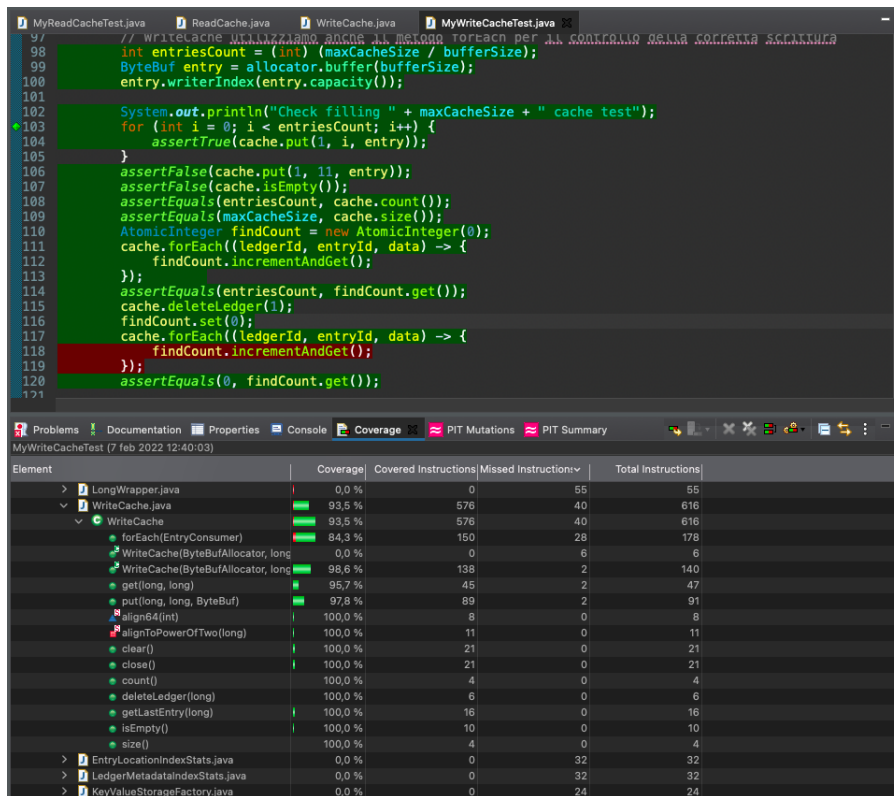
Number of Classes	Line Coverage	Mutation Coverage
16	7% <div><div></div></div> 77/1140	8% <div><div></div></div> 47/619

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ArrayGroupSort.java	18% <div><div></div></div> 7/38	0% <div><div></div></div> 0/49
ArrayUtil.java	0% <div><div></div></div> 0/28	0% <div><div></div></div> 0/54
DbLedgerStorage.java	0% <div><div></div></div> 0/88	0% <div><div></div></div> 0/56
DbLedgerStorageStats.java	0% <div><div></div></div> 0/20	0% <div><div></div></div> 0/4
EntryLocationIndex.java	0% <div><div></div></div> 0/124	0% <div><div></div></div> 0/69
EntryLocationIndexStats.java	0% <div><div></div></div> 0/4	0% <div><div></div></div> 0/1
KeyValueStorageRocksDB.java	0% <div><div></div></div> 0/179	0% <div><div></div></div> 0/59
LedgerMetadataIndex.java	0% <div><div></div></div> 0/115	0% <div><div></div></div> 0/35
LedgerMetadataIndexStats.java	0% <div><div></div></div> 0/4	0% <div><div></div></div> 0/1
LocationsIndexRebuildOp.java	0% <div><div></div></div> 0/39	0% <div><div></div></div> 0/7
LongPairWrapper.java	0% <div><div></div></div> 0/17	0% <div><div></div></div> 0/8
LongWrapper.java	0% <div><div></div></div> 0/15	0% <div><div></div></div> 0/6
ReadCache.java	89% <div><div></div></div> 67/75	81% <div><div></div></div> 43/53

4.2.2 Bookkeeper - WriteCache





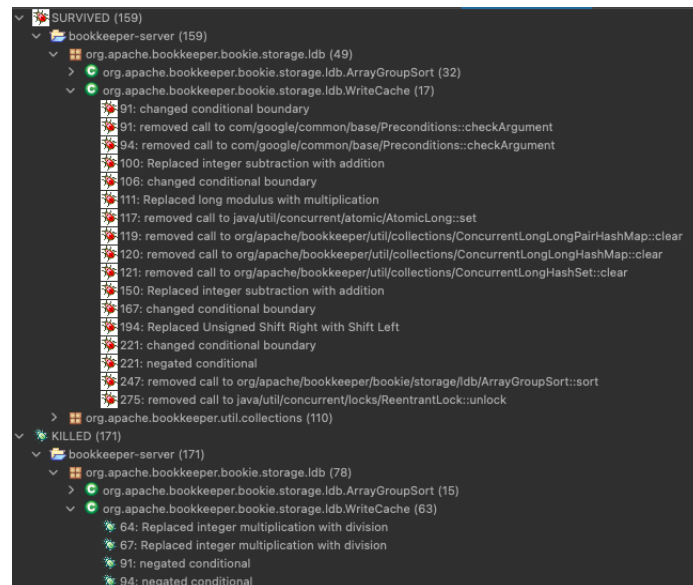
Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage	Mutation Coverage
358	1% 403/27980	2% 193/12778

Breakdown by Package

Name	Number of Classes	Line Coverage	Mutation Coverage
org.apache.bookkeeper.auth	3	0% 0/49	0% 0/32
org.apache.bookkeeper.bookie	54	0% 0/5502	0% 0/2394
org.apache.bookkeeper.bookie.stats	2	0% 0/23	0% 0/5
org.apache.bookkeeper.bookie.storage ldb	16	13% 145/1140	13% 80/619



Pit Test Coverage Report

Package Summary

org.apache.bookkeeper.bookie.storage.ldb

Number of Classes	Line Coverage	Mutation Coverage
16	13% 145/1140	13% 80/619

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ArrayGroupSort.java	92% 35/38	31% 15/49
ArrayUtil.java	0% 0/28	0% 0/54
DbLedgeStorage.java	0% 0/88	0% 0/56
DbLedgeStorageStats.java	0% 0/20	0% 0/4
EntryLocationIndex.java	0% 0/124	0% 0/69
EntryLocationIndexStats.java	0% 0/4	0% 0/1
KeyValueStorageRocksDB.java	0% 0/179	0% 0/59
LedgeMetadataIndex.java	0% 0/115	0% 0/35
LedgeMetadataIndexStats.java	0% 0/4	0% 0/1
LocationsIndexRebuildOp.java	0% 0/39	0% 0/7
LongPairWrapper.java	0% 0/17	0% 0/8
LongWrapper.java	0% 0/15	0% 0/6
ReadCache.java	0% 0/75	0% 0/53
SingleDirectoryDbLedgeStorage.java	0% 0/217	0% 0/109
TransientLedgeInfo.java	0% 0/57	0% 0/25
WriteCache.java	92% 110/120	78% 65/83

4.2.3 Bookkeeper - TravisCI - SonarCloud




✓ bookkeeper-test

LAST BUILD
34

DEFAULT BRANCH
↔ main



COMMIT
553d68a [↗](#)



FINISHED
Passed 25 minutes ago



 **Apache BookKeeper :: Parent** NEW Not computed



PUBLIC



Last analysis: February 7, 2022, 6:55 PM



325   Bugs



7   Vulnerabilities

0.0%   Hotspots Reviewed

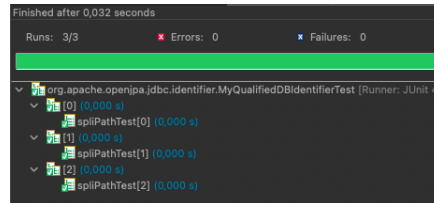
5k   Code Smells

 0.0%  Coverage

 2.2%  Duplications

147k   Java, XML

4.2.4 OpenJPA - QualifiedDBIdentifier



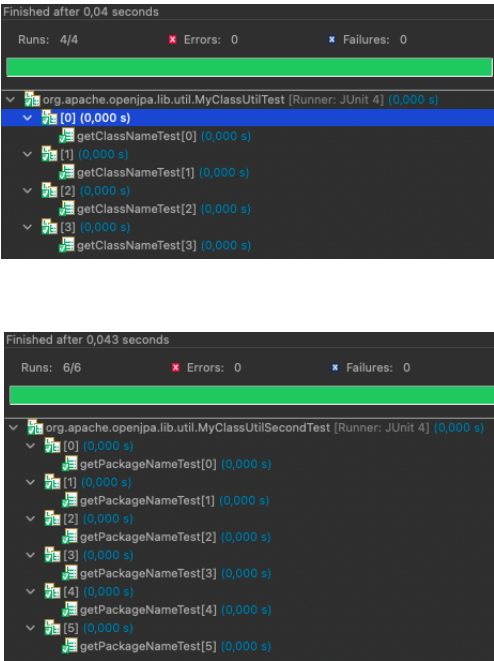
Tests > openjpa-jdbc > org.apache.openjpa.jdbc.identifier > QualifiedDBIdentifier

QualifiedDBIdentifier

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
equals(Object)		0%		0%	8	8	13	13	1	1
pathEqual(QualifiedDBIdentifier, QualifiedDBIdentifier)		0%		0%	7	7	12	12	1	1
isDelimited()		0%		0%	6	6	9	9	1	1
compareTo(Identifier)		0%		0%	4	4	5	5	1	1
getPath(DBIdentifier)		0%		0%	2	2	3	3	1	1
isUnqualifiedColumn()		0%		0%	3	3	1	1	1	1
getUnqualifiedName()		0%		n/a	1	1	4	4	1	1
length()		0%		0%	2	2	4	4	1	1
setName(String)		0%		n/a	1	1	3	3	1	1
isUnqualifiedObject()		0%		n/a	1	1	1	1	1	1
toString()		0%		n/a	1	1	1	1	1	1
setPath(DBIdentifier[])		100%		95%	1	11	0	21	0	1
splitPath(DBIdentifier)		100%		100%	0	7	0	13	0	1
clone()		100%		n/a	0	1	0	7	0	1
getName()		100%		100%	0	3	0	4	0	1
QualifiedDBIdentifier(DBIdentifier[])		100%		n/a	0	1	0	5	0	1
resetNames()		100%		n/a	0	1	0	3	0	1
newPath(DBIdentifier[])		100%		n/a	0	1	0	1	0	1
setSchemaName(DBIdentifier)		100%		n/a	0	1	0	2	0	1
setObjectTableName(DBIdentifier)		100%		n/a	0	1	0	2	0	1
setBaseName(String)		100%		n/a	0	1	0	2	0	1
getSchemaName()		100%		n/a	0	1	0	1	0	1
getObjectTableName()		100%		n/a	0	1	0	1	0	1
getBaseName()		100%		n/a	0	1	0	1	0	1
getIdentfier()		100%		n/a	0	1	0	1	0	1
Total	219 of 460	52%	51 of 86	40%	37	68	56	120	11	25

```
> Total : 18 seconds
=====
- Statistics
=====
>> Line Coverage: 22/11469 (0%)
>> Generated 7303 mutations Killed 15 (0%)
>> Mutations with no coverage 7281. Test strength 68%
>> Ran 55 tests (0.01 tests per mutation)
```

4.2.5 OpenJPA - ClassUtil



OpenJPA Utilities Library > org.apache.openjpa.lib.util > ClassUtil

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toClass(String, boolean, ClassLoader)	<div></div>	0%	<div></div>	0%	13	13	29	29	1	1
getClassName(String)	<div></div>	27%	<div></div>	40%	7	11	15	25	0	1
getPackageName(String)	<div></div>	57%	<div></div>	50%	4	6	5	11	0	1
getPackageName(Class)	<div></div>	0%	<div></div>	0%	2	2	1	1	1	1
toClass(String, ClassLoader)	<div></div>	0%	<div></div>	n/a	1	1	1	1	1	1
getArrayDimensions(String)	<div></div>	81%	<div></div>	50%	1	2	1	4	0	1
getClassName(Class)	<div></div>	75%	<div></div>	50%	1	2	1	3	0	1
static {...}	<div></div>	100%	<div></div>	n/a	0	1	0	1	0	1
Total	248 of 477	48%	45 of 60	25%	29	38	53	75	3	8

Pit Test Coverage Report

Package Summary


org.apache.openjpa.lib.util

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
28	1% <div><div></div><div>22/2194</div></div>	1% <div><div></div><div>15/1565</div></div>	68% <div><div></div><div>15/22</div></div>

Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
Base16Encoder.java	0% <div><div></div><div>0/21</div></div>	0% <div><div></div><div>0/41</div></div>	0% <div><div></div><div>0/0</div></div>
Bytes.java	0% <div><div></div><div>0/46</div></div>	0% <div><div></div><div>0/66</div></div>	0% <div><div></div><div>0/0</div></div>
ClassUtil.java	29% <div><div></div><div>22/77</div></div>	21% <div><div></div><div>15/71</div></div>	68% <div><div></div><div>15/22</div></div>

4.2.6 OpenJPA - TravisCI - SonarCloud



✓ openJPA-test

LAST BUILD
31

DEFAULT BRANCH
main

COMMIT
4a977d5

FINISHED
Passed less than a minute ago

★ OpenJPA Parent POM NEW Not computed PUBLIC

Last analysis: February 7, 2022, 7:20 PM

162 E Bugs

0 A Vulnerabilities

0.0% E Hotspots Reviewed

5.4k A Code Smells

0.7% Coverage

6.8% Duplications

95k M Java, XML, ...