RELAZIONE ISW2
MODULO SOFTWARE TESTING

Software testing & coverage analysis

A cura di
ALESSANDRO AMICI

Matricola : 0280073 Email : a.amici@outlook.it

INTRODUZIONE

Il report del modulo di *Software testing* ha come obiettivo quello di presentare il lavoro svolto nel testing di due progetti *open source* di *Apache*: **BookKeeper** e **OpenJpa**.
Il report è stato stilato in parallelo con la progettazione dei test ed il loro miglioramento, seguendo quindi l'ordine cronologico del lavoro svolto.
L'approccio ai due progetti è stato leggermente differente:per il primo si sono scelte le classi quasi esclusivamente basandosi sui risultati in output dall'applicativo prodotto per l'altro modulo del corso, quindi andando ad analizzare le classi soggette a bugginess che presentassero i valori degli attributi di difettosità più elevati ed inoltre non si è posta troppa enfasi sul corpo dei metodi che si andavano a testare, quanto sulla loro segnatura, al fine di scegliere metodi che avessero i domini di input più vari possibili cosi da comprendere meglio il concetto delle category partition ed acquisirne un po di confidenza. Questo è andato un po a discapito del corpo dei metodi che nei primi esempi in particolare (metodi **initNewCluster** e **readEntries** è risultato un povero).
Quindi già nei test dei metodi successivi ed ancor più per il secondo progetto si è cercato di guidare la scelta della classi da testare non basandosi unicamente sui risultati prodotti dall'applicativo sulla predizione della difettosità ma cercando anche quelle classi e metodi di test che permettessero di esplorare maggiormente le procedure esaminate a lezione (in particolare **Category partition**, **Statement e Branch coverage** e **Mutation coverage**).
L'ambiente di lavoro è stato configurato utilizzando ***Github*** per l'accesso alle repository, ***Maven*** per il build in locale dei progetti ,***Travis CI*** per il build in remoto e **Sonar Cloud** per l'analisi dei test effettuati e le varie **coverages**.
I test precedentemente presenti nelle *repository originali* sono stati eliminati ad eccezione di classi *di supporto* al testing utilizzate per *inizializzare l'ambiente*.
Il setup dei progetti è risultato molto complesso ed ha richiesto molto tempo e tentativi, specialmente nel caso di *OpenJpa : p*er riuscire ad effettuare il setup è stato necessario modificare diversi *pom.xml* ed anche disabilitare *plug-in* non essenziali.
I parametri di input sono stati scelti secondo una procedure **unidimensionale**.
L'ambiente di sviluppo utilizzato è stato Windows per OpenJpa e Ubuntu(Unix like) per Bookkeeper in quanto esplicitamente richiesto nella documentazione di quest'ultimo.
Per entrambi l'ide utilizzata è stata IntellijIdea 2020.
Per entrambi i progetti si è riscontrata una documentazione esigua e la maggior parte delle informazioni utili al testing sono state inferite dal codice stesso.

BOOKKEEPER

Per il progetto Bookkeeper sono state scelte come classi da testare:
- **BookkeperAdmin**
- **ZkUtils**

CATEGORY PARTITION

Per la prima classe sono stati scelti i metodi:
- **public static boolean initNewCluster(ServerConfiguration conf) throws Exception**

- **public Iterable<LedgerEntry> readEntries(long ledgerId, long firstEntry, long lastEntry) throws InterruptedException, BKException**

Relativamente al primo metodo , questo si occupa di inizializzare un nuovo cluster creando gli "znodi" richiesti e da errore nel caso in cui "ledgersrootpath" già esista. Avendo in input un tipo di dato complesso , inizialmente lo si può distinguere in : istanza nulla, istanza valida , istanza non valida. La partizioni del dominio di input risultano quindi essere :

- **ServerConfiguration conf**

{null} , {new ServerConfiguration() } , { new WrongServerConfiguration () }

Un insieme di casi di test minimale è il seguente :

	conf
Caso1	{ null }
Caso2	{ TestBKConfiguration.newServerConfiguration() } //right one
Caso3	

//example of wrong one
{ new ServerConfiguration().setMetadataServiceUri("zk+hierarchical://127.0.0.1/ledgers") }

I **valori attesi** del test sono :

Caso1 >>> sollevamento di un'eccezione

Caso2 >>> successo

Caso3 >>> sollevamento di un'eccezione

Relativamente al secondo metodo, questo si occupa leggere in modo sincrono le entries (sequenze di bytes e meta-dati) che vengono scritte su un ledger (sequenza di entries scritte su bookkeeper con semantica append only) identificato da **ledgerId : firstEntry** indica la prima entrie da cui iniziare a leggere e **lastEntry** indica l'ultima. Nel caso in cui **lastEntry** sia uguale a -1 verranno lette tutte le entries a partire da **firstEntry**.
Dal corpo del metodo si è dedotto che **ledgerId** e **firstEntry** debbano essere valori positivi (>=0) e dalla semantica stessa della segnatura del metodo è ragionevole pensare che **lastEntry** possa essere considerato in funzione di **firstEntry**, si hanno quindi le seguenti partizioni dei domini di input :

- **long ledgerId** {<0} , {>=0 }

- **long firstEntry** {<0} , {>=0 }

- **long lastEntry** { < firstEntry} , { = firstEntry } , { > firstEntry}

Da qui si ricavano i seguenti boundary values:

- **long ledgerId** {-1 , 0 ,1 }

- **long firstEntry** {-1 , 0 ,1 }

- **long lastEntry** {firstEntry -1 , firstEntry , firstEntry +1 }

Un insieme di casi di test minimale è quindi il seguente :

	ledgerId	**firstEntry**	**lastEntry**
Caso1	{ -1	-1	-1 }
Caso2	{ 0	1	2 }

Caso3 { 1 0 -1 }

I **valori attesi** del test sono :

Caso1 >>> sollevamento di un'eccezione

Caso2 >>> successo

Caso3 >>> successo

Per la seconda classe sono stati scelti i metodi:

- **public static void createFullPathOptimistic(ZooKeeper zkc, String path, byte[] data, final List<ACL> acl, final CreateMode createMode)**

- **public static void deleteFullPathOptimistic(ZooKeeper zkc, String path, int znodeVersion) throws KeeperException, InterruptedException**

- **public static List<String> getChildrenInSingleNode(final ZooKeeper zk, final String node, long zkOpTimeoutMs)throws InterruptedException , IOException, KeeperException.NoNodeException**

Il primo dei tre si occupa di creare in modo ricorsivo ed "ottimistico" un path zookeeper e puo lanciare qualunque tipo di eccezione : **NodeExistsException** viene lanciata solo nel caso in cui il full path indichi un path già esistente mentre l'esistenza di nodi padre non è considerata fonte di eccezione. Le partizioni dei domini di input risultano essere:

- **ZooKeeper zkc** {null} , {new ZooKeeper() } , { new WrongZooKeeper() }

- **String path** { valid String} , { non valid String} , {already existing String}

- **byte[] data** {null} , { valid Array} , { empty Array}

- **final List<ACL> acl** { null } , { valid List} , { empty List}

- **CreateMode createMode** { Persistent } , {Persistent_Sequential} , { Ephemeral } ,

{ Ephemeral_sequential} , { Container } , { Persistent_With_TTL} ,

{ Persistent_Sequential_With_TTL }

Da qui sono stati scelti come valori per il testing effettivo i seguenti :

- **ZooKeeper zkc**

{null}

{new ZooKeeper() } = new ZooKeeper(new ZooKeeperUtil().getZooKeeperConnectString(), 10000, null)

{ new WrongZooKeeper() } = new ZooKeeper(" ", 10000, null)

- **String path**

{ valid String} = "/ledgers/000/000/000/001"

{ non valid String} = "/ledgers/000/000/000/00b"

{already existing String} = stringa valida ma che è gia stata utilizzata come path zookeeper Es. "/ledgers/000/000/000/001"

- **byte[] data**

{null}

{ valid Array} = "data".getBytes()

{ empty Array} = new byte[]

- **final List<ACL> acl**

{ null }

{ valid List} = Ids.OPEN_ACL_UNSAFE

{ empty List} = new ArrayList<ACL>()

- **CreateMode createMode**

{ Persistent } (1)

{Persistent_Sequential} (2)

{ Ephemeral } (3)

{ Ephemeral_sequential} (4)

{ Container } (5)

{ Persistent_With_TTL} (6)

{ Persistent_Sequential_With_TTL } (7)

Un insieme di casi di test minimale risulta essere il seguente :

(per motivi di leggibilità, avendo numerosi parametri, si sono indicati i tipi di input ,es. Valid/not valid, piuttosto che le loro inizializzazioni effettive, comunque indicate sopra.Le enumerazioni di **createMode** sono state associate a valori interi per lo stesso motivo.)

	Zkc	**path**	**data**	**acl**	**createMode**
Caso1	{ null	valid String	validArray	validList	7 }
Caso2	{ new ZooKeeper()	non valid String	validArray	validList	6 }
Caso3	{new WrongZooKeeper()	valid String	validArray	validList	5 }
Caso4	{ new ZooKeeper()	valid String	null	validList	2 }
Caso5	{ new ZooKeeper()	already existing String	empty Array	null	3 }
Caso6	{ new ZooKeeper()	valid String	validArray	empty List	4 }
Caso7	{ new ZooKeeper()	valid String	validArray	validList	1 }

I **valori attesi** del test sono :

Caso1 >>> sollevamento di un’eccezione

Caso2 >>> sollevamento di un’eccezione

Caso3 >>> sollevamento di un’eccezione

Caso4 >>> successo (non viene scritto nulla nel nodo)

Caso5 >>> sollevamento di un’eccezione

Caso6 >>> sollevamento di un’eccezione

Caso7 >>> successo

Il metodo **deleteFullPathOptimistic** si occupa invece di rimuovere in modo asincrono e ricorsivo uno zookkeeper path : rimuove i nodi foglia ed i corrispondenti genitori se questi non hanno altri nodi figli. Per cancellare il nodo padre il parametro **znodeVersion** va impostato a -1.Se si fallisce nel rimuovere il nodo foglia l’esecuzione viene ritornata con un codice d’errore, ma se si fallisce nel rimuovere il nodo padre viene ritornato un codice errore ok.

- **ZooKeeper zkc** {null} , {new ZooKeeper() } , { new WrongZooKeeper() }

- **String path** {valid existing String} , {valid non existing String} , {non valid String}

-int znodeVersion {<0} , {>=0}

Da qui sono stati scelti come valori per il testing effettivo i seguenti :

- **ZooKeeper zkc**

{null}

{new ZooKeeper() } = new ZooKeeper(new ZooKeeperUtil().getZooKeeperConnectString(),

10000, null)

{ new WrongZooKeeper() } = new ZooKeeper(” ”, 10000, null)

- **String path**

{ valid existing String} = "/ledgers/000/000/000/001"

{ valid non existing String} = path con sintassi valida ma con associato nessun nodo

Es. "/ledgers/000/000/000/001" dopo che è stato chiamato il metodo delete con questo path

{ non valid String} = "/ledgers/000/000/000/00b"

- **int znodeVersion** { -1 } , { 0 } , { 1 }

Un insieme di casi di test minimale risulta essere il seguente :

Zkc	**path**	**znodeVersion**

Caso1 { null valid non existing String 1 }

Caso2 { new ZooKeeper() valid existing String -1 }

Caso3 {new WrongZooKeeper() non valid String 0 }

I **valori attesi** del test sono :

Caso1 >>> sollevamento di un'eccezione

Caso2 >>> successo

Caso3 >>> sollevamento di un'eccezione

Il metodo **getChildrenInSingleNode** si occupa di ricercare e restituire tutti i nodi figli del nodo zookeeper passato in input. Il metodo può sollevare diverse eccezioni in relazione allo scadere del timeout indicato o se il nodo passato non è conforme a quanto atteso dal metodo.

- **ZooKeeper zkc** {null} , {new ZooKeeper() } , { new WrongZooKeeper() }

- **String node** {valid existing String} , {valid non existing String} , {non valid String}

- long zkOpTimeoutMs {<=0} , {>0}

Da qui sono stati scelti come valori per il testing effettivo i seguenti :

- **ZooKeeper zkc**

{null}

{new ZooKeeper() } = new ZooKeeper(new ZooKeeperUtil().getZooKeeperConnectString(),

10000, null)

{ new WrongZooKeeper() } = new ZooKeeper(" wrong string", 10000, null)

- **String node**

{ valid existing String} = "/ledgers/000/000/000"

{ valid non existing String} = path con sintassi valida ma con associato nessun nodo

Es. "/ledgers/000/000/000/004" dopo che è stato chiamato il metodo delete con questo path

{ non valid String} = "/ledgers/000/000/000/00b"

- **long zkOpTimeoutMs** { -1 } , { 0 } , { 100 }

NB. Si è scelto 100 come valore anche se non un valore di confine con le partizioni del dominio sopradefinite per questo metodo poichè un valore troppo basso, seppur positivo, non avrebbe dato il tempo necessario per completare lo svolgimento del metodo ma avrebbe portato al sollevamento di un'eccezione, come i vedrà in seguito nella fase di analisi dell'adeguatezza.

Un insieme di casi di test minimale risulta essere il seguente :

	Zkc	path	znodeVersion
Caso1	{ null	valid non existing String	0 }
Caso2	{ new ZooKeeper()	valid existing String	100 }
Caso3	{new WrongZooKeeper()	non valid String	-1 }

I **valori attesi** del test sono :

Caso1 >>> sollevamento di un'eccezione

Caso2 >>> successo

Caso3 >>> sollevamento di un'eccezione

Adeguatezza e miglioramento dei casi di test

Statement & Branch coverage

In questa sezione del report sono mostrati per il progetto in esame i risultati ottenuti con il calcolo della **statement e branch coverage** e i miglioramenti apportati ai casi di test per incrementare i suddetti risultati.

Per la classe **BookkeeperAdmin** si ha :

- **initNewCluste** : statement coverage del **100%** e branch coverage **n/a**.
(Per quest'ultima si è provato ad utilizzare mockito e powermock sul metodo chiamato all'interno del metodo under test, cosi da poter triggerare il lancio di un'eccezione ma non si è riusciti a far funzionare i plugin per il mocking ne mockando la classe concreta ne mockando la sua interafccia , in quanto in entrambi gli scenari ad essere chiamato non era il metodo mockato ma quello ufficiale. Si è provato anche a mockare con Powermock il metodo new() sulla classe ZKRegistrationManager cosi da restituire un oggetto mockato alla classe chiamante ed a far sollevare a quest'ultimo l'eccezione, ma non si ha avuto successo neanche in questo modo).

- **readEntries** : statement coverage del **100%** e branch coverage iniziale del 75% che è stata portata poi al **100%** con l'aggiunta del nuovo caso di test :

Caso4 { 1 -1 -2 }

avente come valore atteso : Caso4 >>> sollevamento di un'eccezione

---

Per la classe **ZkUtils** invece :

- **createFullPathOptimistic :** statement coverage del **100%** e branch coverage del **100%**.

- **deleteFullPathOptimistic :** statement coverage del **100%** e branch coverage del **100%**.

- **getChildrenInSingleNode** : statement coverage del 57% e branch coverage del 59% inizialmente.

Con l'aggiunta dei seguenti casi di test le coverage sono state migliorate rispettivamente al **91%** e **100%**.

Caso4 {new ZooKeeper() non valid String 0 }

avente come valore atteso : Caso4 >>> sollevamento di un'eccezione

Caso5 {new ZooKeeper() /ledgers/000/000/000/003" 1 }

avente come valore atteso : Caso5 >>> sollevamento di un'eccezione

Nota1: anche in quest'ultimo metodo si è provato ad utilizzare tools di mocking per mockare l'oggetto ctx instanziato dalla classe under test ma senza successo : si è provato a mockare la creazione dell'oggetto facendo ritornare anzichè l'oggetto reale uno mockato cosi da poter lavorare con quest'ultimo ma non si ha avuto successo. Si è inoltre notato che l'oggetto ctx viene acceduto mediante un lock e si ipotizza che ciò possa aver contribuito alla fallimento dell'utilizzo di mock.

Mutation coverage

Per la classe **BookkeeperAdmin** abbiamo una mutation coverage del **6%** avendo impostato come max numero di mutanti **50**.

- **initNewCluste** : presenta un totale di **4** mutanti creati di cui **2** uccisi , **1** timed_out e **1** sopravvissuto per una coverage del metodo tra il **50%** e **75%** circa.

- **readEntries** : presenta un totale di **6** mutanti creati di cui **1** ucciso e **5** timed_out per una coverage del metodo non ben definibile.

Per la classe **ZkUtils** si ha una mutation coverage del **96%** avendo impostato come max numero di mutanti **50**.

- **createFullPathOptimistic** : presenta un totale di **3** mutanti creati di cui **2** uccisi , **1** timed_out per una coverage del metodo superiore al **66%**.

- **deleteFullPathOptimistic** : presenta un totale di **3** mutanti creati di cui **2** uccisi , **1** timed_out per una coverage del metodo superiore al **66%**.

- **getChildrenInSingleNode** : presenta un totale di **13** mutanti creati di cui **7** uccisi , **5** timed_out e **1** sopravvissuto per una coverage del metodo superiore al **54%**.
.
Nota: si sarebbe voluto migliorare ulteriormente le mutation coverage ma per mancanza di tempo, causata soprattutto dalla difficoltà di configurazione degli ambienti e della gestione dei progetti,purtroppo non è stato possibile.
Tuttavia si è cercato di approfondire anche questo aspetto della trattazione nel secondo progetto, dove, dopo aver preso maggiore confidenza con l'attività di testing nel suo complesso, è stato possibile riportare risultati più soddisfacenti.

OPENJPA

Per il progetto OpenJpa sono state scelte come classi da testare:
- **QualifiedDBIdentifier**
- **ClassUtil**

CATEGORY PARTITION

Per la prima classe sono stati scelti i metodi:
- **public void setPath(DBIdentifier...sNames)**

- **public static DBIdentifier[ ] splitPath(DBIdentifier sName)**

Relativamente al primo metodo , questo si occupa di costruire un nuovo path da associare all'oggetto della classe partendo dagli identificatori che vengono forniti in input.
Come si può vedere il numero degli identificatori è variabile.
Nel test si è scelto di utilizzare gli identificatori principali che sono poi quelli che vengono registrati negli attributi dell'oggetto creato.
Quindi sono stati scelti come valori del testing:

- **DBIdentifier sName1**

{null} , {new DBIdentifier() } , { new wrong DBIdentifier() }

- **DBIdentifier sName2**

{null} , {new DBIdentifier() } , { new wrong DBIdentifier() }

Un insieme di casi di test minimale è il seguente :

	sName1	**sName2**
Caso1	{ null }	{ null }
Caso2	{ DBIdentifier.newSchema("Schema_1") }	{ DBIdentifier.newTable("Table_1") }
Caso3	{ DBIdentifier.NULL }	{ DBIdentifier.NULL }

I **valori attesi** del test sono :

Caso1 >>> fallimenti : essendo gli identificatori non validi il path dell'oggetto non viene sovrascritto

Caso2 >>> successo : il path dell'oggetto viene sovrascritto con gli identificatori

Caso3 >>> fallimento : essendo gli identificatori non validi il path dell'oggetto non viene sovrascritto

---

Relativamente al secondo metodo, d'altro canto questo si occupa di prendere un path già formato e ritornarlo sotto forma di lista degli identificatori che lo compongono.
Da notare che il metodo adotta comportamenti diversi a seconda che l'istanza di input sia di tipo DBIdentifier oppure di tipo QualifiedDbIdentifier che è un'estensione della precedente.

- **DBIdentifier sName**

{null} , {new DBIdentifier() } , { new wrong DBIdentifier() }

Un insieme di casi di test minimale è quindi il seguente :

	sName1
Caso1	{ null }
Caso2	{ DBIdentifier.newTable("Schema.Table") }
Caso3	{QualifiedDbIdentifier.newPath(DBIdentifier.NULL) }

I **valori attesi** del test sono :

Caso1 >>> una lsita DBIdentifier [] vuota

Caso2 >>> una lsita DBIdentifier con 2 diversi identificatori, uno per lo schema e l'altro per la tabella

Caso3 >>> una lsita DBIdentifier [] vuota

Per la classe **ClassUtil** invece sono stati scelti i metodi :

- public static Class toClass(String str, boolean resolve, ClassLoader loader)

-public static String getClassName(String fullName)

Il primo metodo si occupa di ritornare la classe associata alla stringa identificativa della classe che viene passata in input.La classe viene anche inizializzata in caso in cui il parametro "resolve" sia impostato a true.Si nota una certa affidabilità dal metodo dal check che si fa sul potenziale valore null dell'oggetto "loader" nel cui caso viene utilizzato il classloader della classe corrente.

- **String str** {existing Class path } , {non existing Class path}, {invalid path}

- **boolean resolve** {true} , {false}

- ClassLoader loader {null} , { new ClassLoader() } , { new wrong ClassLoader() }

Da qui sono stati scelti come valori per il testing effettivo i seguenti :

- **String str**

{"org.apache.openjpa.lib.util.ClassUtilToClassTest" } //valid string

{ "org.apache.openjpa.lib.util.NotAnExistingClass"} /invalid string

{ null }

- **boolean resolve** {true} , {false}

- **ClassLoader loader**

{ null }

{ this.getClass().getClassLoader() } // valid loader

{ ArrayList.class.getClassLoader() } // invalid loader
.
Un insieme di casi di test minimale risulta essere il seguente :

	str	resolve	loader
Caso1	{ null	true	null }
Caso2	{ valid string	false	valid loader }
Caso3	{ invalid string	false	invalid loader }

I **valori attesi** del test sono :

Caso1 >>> sollevamento di un'eccezione

Caso2 >>> successo

Caso3 >>> sollevamento di un'eccezione

Il metodo **getClassName** si occupa di estrapolare dal path completo di una classe soltanto il nome di quest'ultima, ovvero la sottostringa successiva all'ultimo carattere "." .

Essendo l'unico parametro di input una stringa, la category partition per un insieme di casi di test minimale è il seguente:

- **String fullName** {valid String } , {non valid String}

Un insieme di casi di test minimale risulta essere il seguente :

	fullName
Caso1	{ "class org.apache.openjpa.lib.util.ClassUtilGetClassNameTest" }
Caso2	{ null }

I **valori attesi** del test sono :

Caso1 >>> "ClassUtilGetClassNameTest"

Caso2 >>> null

Adeguatezza e miglioramento dei casi di test

Statement & Branch coverage

In questa sezione del report sono mostrati per il progetto in esame i risultati ottenuti con il calcolo della **statement e branch coverage** e i miglioramenti apportati ai casi di test per incrementare i suddetti risultati.

Per la classe **QualifiedDBIdentifier** si ha :

- **setPath**: statement coverage del 80% e branch coverage del 75% inizialmente.

Con l'introduzione del nuovo caso di test con un singolo DBIdentifier abbiamo aumentato la statement coverage al 90%.

Caso4 { DBIdentifier.newSchema("Schema_1") } { \\ }

Avente come valore atteso >>> successo : il path dell'oggetto viene sovrascritto con l'identificatore.

Aggiungendo il caso sotto riportato si è aumentata la statement coverage al **100%** e la branch coverage al 85%.

Caso5 { DBIdentifier.newTable("Table_1") } { DBIdentifier.newColumn("Column_1") }

Avente come valore atteso >>> successo : il path dell'oggetto viene sovrascritto con gli identificatori.

Aggiungendo il caso sotto riportato si è aumentata la branch coverage al 90%.

Caso6 { DBIdentifier.newColumn("Column_1")) } { DBIdentifier.newCatalog("Catalog_1") }

Avente come valore atteso >>> successo : il path dell'oggetto viene sovrascritto con solo l'ultimo identificatore(il secondo parametro di input).

Aggiungendo il caso sotto riportato si è aumentata la branch coverage al **95%**.

Caso7 { null } { \\ }

Avente come valore atteso >>> fallimento : essendo gli identificatori non validi il path dell'oggetto non viene sovrascritto

Nota1: l'ultimo 5% di branch coverage non coperto è dovuto all'istruzione :

if (i == (sNames.length - 1) && sNames.length != 1)

poichè la seconda condizione non assume mai valore false (ovvero la lunghezza della lista uguale ad 1) : tuttavia questo è impossibile che accada poichè nel blocco precedente a questa istruzione che contiene la clausola : if (sNames.length == 1) è presente uno statement di return e quindi non si giunge mai al blocco successivo.

Nota2: la numerosità dei test in questa sezione è dovuta alla doppia natura dei parametri nella segnatura del metodo (DBIdentifier...sNames) secondo cui si possono passare come argomento sia una lista di identificatori (DBIdentifier[]) che i singoli identificatori uno alla volta(DBIdentifier1 , DBIdentifier2).Nei test si è scelto inizialmente un approccio del secondo tipo nella gestione dei parametri ma molti branch effettuavano verifiche che involvevano la prima tipologia.

I valori finali sono quindi : statement coverage **100%** e branch coverage **95%**.

- **splitPath**: statement coverage del **100%** e branch coverage del **100%** sin dall'inizio.

Per la classe **ClassUtil** invece :

- **toClass :** statement coverage del 28% e branch coverage del 25% inizialmente.

Con l'aggiunta del caso di test sotto riportato si sono aumentate entrambe le coverage: la prima all' 85% e la seconda al 75%.

Caso4 { "betterNoExistingClass[]" false null }

Avente come valore atteso >>> fallimento: sollevamento di un’eccezione.

Con l’aggiunta del caso di test sotto riportato si sono aumentate entrambe le coverage: la prima all’ 94% e la seconda al 95%.

Caso5 { "byte[]" false null }

Avente come valore atteso >>> successo.

Con l’aggiunta del caso di test sotto riportato si sono aumentate entrambe le coverage: la prima all’ **100%** e la seconda al **100%**.

Caso6 { "byte" false null }

Avente come valore atteso >>> successo.

- **getClassName** : statement coverage del 24% e branch coverage del 30% inizialmente.

Con l’aggiunta del caso di test sotto riportato si sono aumentate entrambe le coverage: la prima al 26% e la seconda al 35%.

Caso3 { “” }

Avente come valore atteso >>> “”.

Con l’aggiunta del caso di test sotto riportato si sono aumentate entrambe le coverage: la prima al 80% e la seconda al 80%.

Caso4 { “[I” }

Avente come valore atteso >>> int[] .

Con l’aggiunta del caso di test sotto riportato si è aumentata la branch coverage all’ 85 %.

Caso5 { “[K” }

Avente come valore atteso >>> [K[] .

Con l’aggiunta del caso di test sotto riportato si sono aumentate entrambe le coverage: la prima al 91% e la seconda al 95%.

Caso6 { “[Byte” }

Avente come valore atteso >>> Byte[] .

Infine abbiamo raggiunto il **100%** su **entrambe** le metriche aggiungendo l’ultimo caso di test:

Caso7 { “[Byte;” }

Avente come valore atteso >>> Byte[] .

Mutation coverage

Per la classe **QualifiedDBIdentifie** si ha una mutation coverage del **39%** avendo impostato come max numero di mutanti **50**.

- **setPath**: presenta un totale di **21** mutanti creati di cui **18** uccisi e **3** sopravvissuti per una coverage del metodo dell'**85,71%**.

- **splitPath**: presenta un totale di **9** mutanti creati di cui tutti e **9** uccisi per una coverage del metodo del **100%**.

Per la classe **ClassUtil** si ha una mutation coverage del **61%** avendo impostato come max numero di mutanti **50**.

- **toClass** : presenta un totale di **23** mutanti creati di cui **17** uccisi , **1** timed_out e **5** sopravvissuti per una coverage del metodo del **78,26%**.

- **getClassName** : presenta un totale di **28** mutanti creati di cui **21** uccisi , **1** memory error e **6** sopravvissuti per una coverage del metodo del **78,57%**.

Nota: in questo secondo progetto si ritiene di aver assunto maggior praticità con le pratiche di testing e di aver prodotto risultati più soddisfacenti.
Anche qui purtroppo per assenza di tempo non è stato possibile incrementare ulteriormente le metriche del mutation coverage poichè si è cercato di mettere in pratica quanto più possibile degli aspetti visti a lezione, ma la configurazione del progetto stesso ha richiesto molto più tempo di quanto preventivato.
Inoltre si ritiene che la bontà dei risultati ottenuti sulla mutation coverage, oltre che al tempo speso su questa coverage stessa, siano da imputare anche al miglioramento apportato alle statement e branch coverage.

CONCLUSIONI

L'attività di testing svolta ha prodotto risultati interessanti .
E' stato istruttivo studiare il codice sorgente di un progetto open source gestito e sviluppato da *Apache* ed è stato molto soddisfacente riuscire ad applicare quanto visto a lezione in modo molto pratico, anche se la complessità dei progetti da testare è stata spesso motivo di rallentamenti nell'attività di testing.

Di seguito si riportano i vari links a quanto prodotto:

- Github : https://github.com/CecBazinga?tab=repositories

-TravisCI : https://travis-ci.org/github/CecBazinga

- SonarCloud : https://sonarcloud.io/organizations/cecbazinga/projects?sort=name

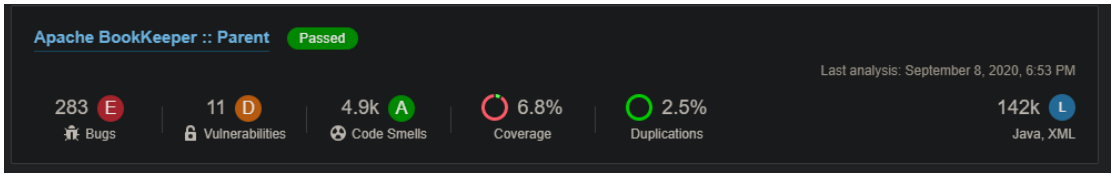


Image 1 Bookkeeper dashboard on **SonarCloud**

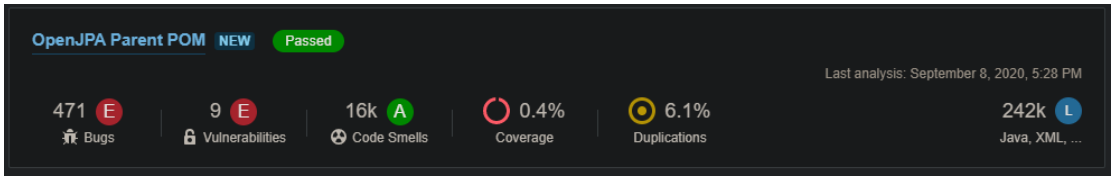


Image 2 OpenJpa dashboard on **SonarCloud**

```
public static boolean initNewCluster(ServerConfiguration conf) throws Exception {
    return runFunctionWithRegistrationManager(conf, rm -> {
        try {
            return rm.initNewCluster();
        } catch (Exception e) {
            throw new UncheckedExecutionException(e.getMessage(), e);
        }
    });
}
```

Image 3 Statement&Branch coverage del metodo **InitNewCluster**

```
public  Iterable<LedgerEntry> readEntries(long ledgerId, long firstEntry, long lastEntry)
        throws InterruptedException, BKException {
    checkArgument(ledgerId >= 0 && firstEntry >= 0);
    return new LedgerEntriesIterable(ledgerId, firstEntry, lastEntry);
}
```

Image 4 Statement&Branch coverage del metodo **ReadEntries**

```
public static void createFullPathOptimistic(ZooKeeper zkc, String path,
        byte[] data, final List<ACL> acl, final CreateMode createMode)
        throws KeeperException, InterruptedException {
    final CountDownLatch latch = new CountDownLatch(1);
    final AtomicInteger rc = new AtomicInteger(Code.OK.intValue());
    asyncCreateFullPathOptimistic(zkc, path, data, acl, createMode,
                                  new StringCallback() {
                                      @Override
                                      public void processResult(int rc2, String path,
                                                                Object ctx, String name) {
                                          rc.set(rc2);
                                          latch.countDown();
                                      }
                                  }, null);
    latch.await();
    if (rc.get() != Code.OK.intValue()) {
        throw KeeperException.create(Code.get(rc.get()));
    }
}
```

Image 5 Statement&Branch coverage del metodo **CreateFullPathOptimistic**

```
public static void deleteFullPathOptimistic(ZooKeeper zkc, String path, int znodeVersion)
        throws KeeperException, InterruptedException {
    final CountDownLatch latch = new CountDownLatch(1);
    final AtomicInteger rc = new AtomicInteger(Code.OK.intValue());
    asyncDeleteFullPathOptimistic(zkc, path, znodeVersion, new VoidCallback() {
        @Override
        public void processResult(int rc2, String path, Object ctx) {
            rc.set(rc2);
            latch.countDown();
        }
    }, path);
    latch.await();
    if (rc.get() != Code.OK.intValue()) {
        throw KeeperException.create(Code.get(rc.get()));
    }
}
```

Image 6 Statement&Branch coverage del metodo **DeleteFullPathOptimistic**

```
public static List<String> getChildrenInSingleNode(final ZooKeeper zk, final String node, long zkOpTimeoutMs)
        throws InterruptedException, IOException, KeeperException.NoNodeException {
    final GetChildrenCtx ctx = new GetChildrenCtx();
    getChildrenInSingleNode(zk, node, new GenericCallback<List<String>>() {
        @Override
        public void operationComplete(int rc, List<String> ledgers) {
            synchronized (ctx) {
                if (Code.OK.intValue() == rc) {
                    ctx.children = ledgers;
                }
                ctx.rc = rc;
                ctx.done = true;
                ctx.notifyAll();
            }
        }
    });

    synchronized (ctx) {
        long startTime = System.currentTimeMillis();
        while (!ctx.done) {
            try {
                ctx.wait(zkOpTimeoutMs > 0 ? zkOpTimeoutMs : 0);
            } catch (InterruptedException e) {
                ctx.rc = Code.OPERATIONTIMEOUT.intValue();
                ctx.done = true;
            }
            // timeout the process if get-children response not received
            // zkOpTimeoutMs.
            if (zkOpTimeoutMs > 0 && (System.currentTimeMillis() - startTime) >= zkOpTimeoutMs) {
                ctx.rc = Code.OPERATIONTIMEOUT.intValue();
                ctx.done = true;
            }
        }
    }
    if (Code.NONODE.intValue() == ctx.rc) {
        throw new KeeperException.NoNodeException("Got NoNode on call to getChildren on path " + node);
    } else if (Code.OK.intValue() != ctx.rc) {
        throw new IOException("Error on getting children from node " + node);
    }
    return ctx.children;
}
```

Image 7 Statement&Branch coverage del metodo **GetChildrenInSingleNode**

```
/**
 * Intializes new cluster by creating required znodes for the cluster. If
 * ledgersrootpath is already existing then it will error out.
 *
 * @param conf
 * @return
 * @throws Exception
 */
public static boolean initNewCluster(ServerConfiguration conf) throws Exception {
    return runFunctionWithRegistrationManager(conf, rm -> {
        try {
            return rm.initNewCluster();
        } catch (Exception e) {
            throw new UncheckedExecutionException(e.getMessage(), e);
        }
    });
}
```

Image 8 Mutation coverage del metodo **InitNewCluster**

```
    /**
     * Read entries from a ledger synchronously. If the lastEntry is -1, it will read all the entries in the ledger from
     * the firstEntry.
     *
     * @param ledgerId
     * @param firstEntry
     * @param lastEntry
     * @return
     * @throws InterruptedException
     * @throws BKException
     */
    public  Iterable<LedgerEntry> readEntries(long ledgerId, long firstEntry, long lastEntry)
            throws InterruptedException, BKException {
5       checkArgument(ledgerId >= 0 && firstEntry >= 0);
1       return new LedgerEntriesIterable(ledgerId, firstEntry, lastEntry);
    }
```

Image 9 Mutation coverage del metodo **ReadEntries**

```
    public static void createFullPathOptimistic(ZooKeeper zkc, String path,
            byte[] data, final List<ACL> acl, final CreateMode createMode)
            throws KeeperException, InterruptedException {
        final CountDownLatch latch = new CountDownLatch(1);
        final AtomicInteger rc = new AtomicInteger(Code.OK.intValue());
1       asyncCreateFullPathOptimistic(zkc, path, data, acl, createMode,
                                      new StringCallback() {
                                          @Override
                                          public void processResult(int rc2, String path,
                                                                    Object ctx, String name) {
                                              rc.set(rc2);
                                              latch.countDown();
                                          }
                                      }, null);
1       latch.await();
1       if (rc.get() != Code.OK.intValue()) {
            throw KeeperException.create(Code.get(rc.get()));
        }
    }
```

Image 10 Mutation coverage del metodo **CreateFullPathOptimistic**

```
    public static void deleteFullPathOptimistic(ZooKeeper zkc, String path, int znodeVersion)
            throws KeeperException, InterruptedException {
        final CountDownLatch latch = new CountDownLatch(1);
        final AtomicInteger rc = new AtomicInteger(Code.OK.intValue());
1       asyncDeleteFullPathOptimistic(zkc, path, znodeVersion, new VoidCallback() {
            @Override
            public void processResult(int rc2, String path, Object ctx) {
                rc.set(rc2);
                latch.countDown();
            }
        }, path);
1       latch.await();
1       if (rc.get() != Code.OK.intValue()) {
            throw KeeperException.create(Code.get(rc.get()));
        }
    }
```

Image 11 Mutation coverage del metodo **DeleteFullPathOptimistic**

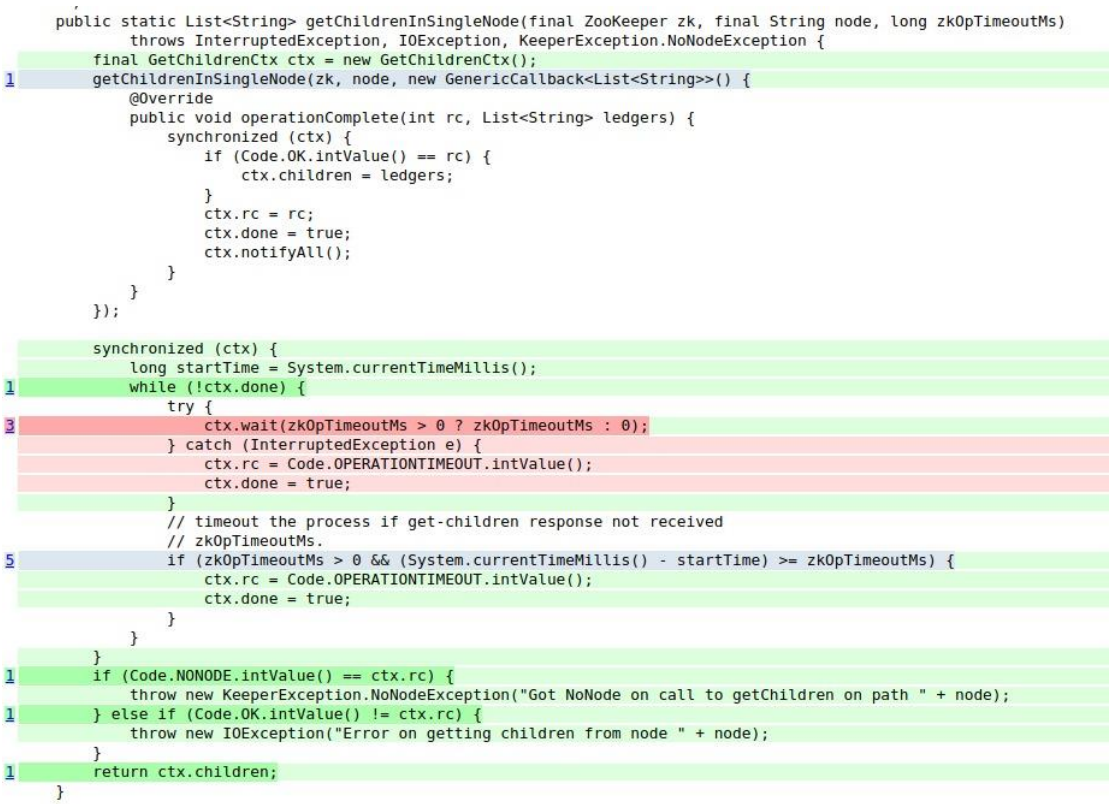
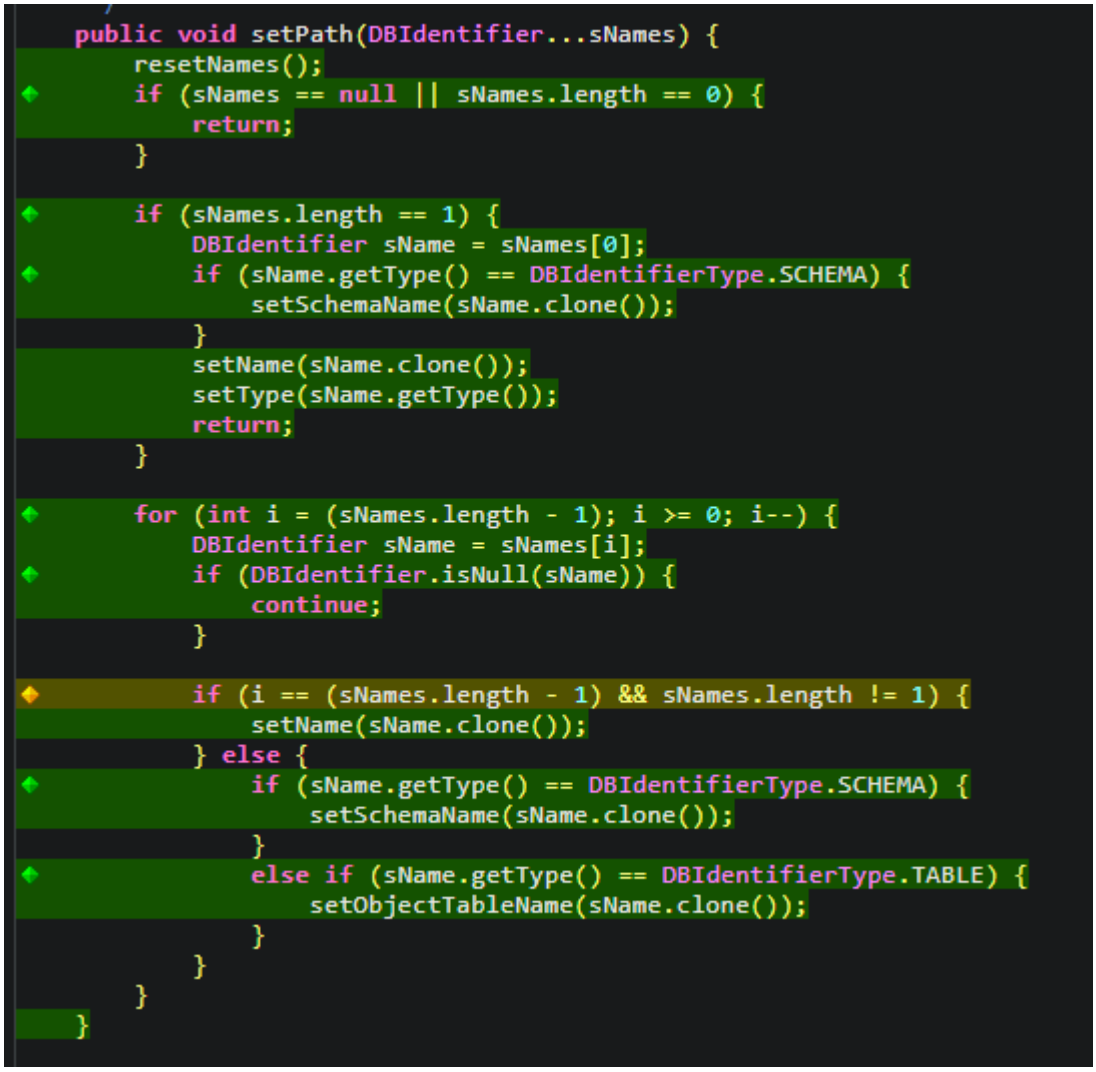
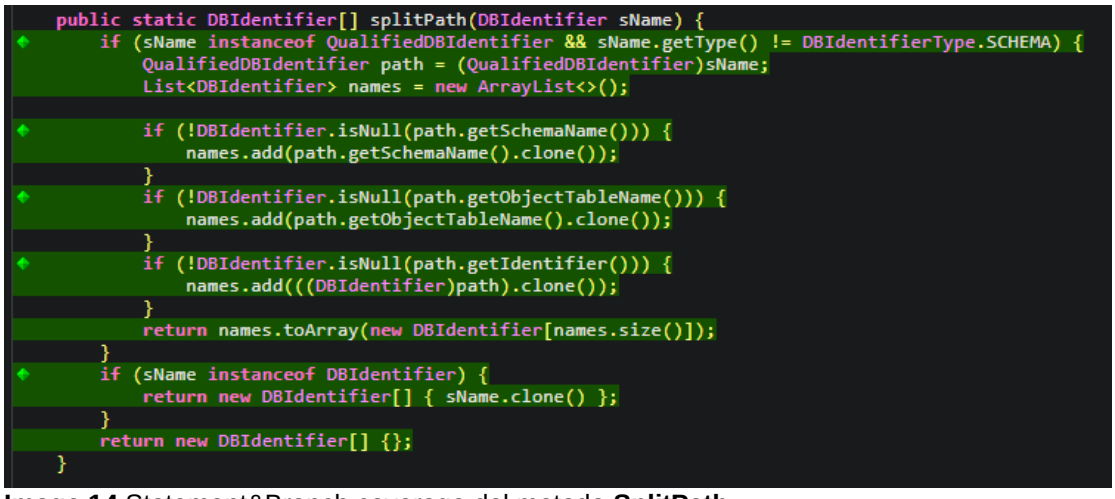


Image 12 Mutation coverage del metodo **GetChildrenSingleNode**



```
    public void setPath(DBIdentifier...sNames) {
        resetNames();
        if (sNames == null || sNames.length == 0) {
            return;
        }

        if (sNames.length == 1) {
            DBIdentifier sName = sNames[0];
            if (sName.getType() == DBIdentifierType.SCHEMA) {
                setSchemaName(sName.clone());
            }
            setName(sName.clone());
            setType(sName.getType());
            return;
        }

        for (int i = (sNames.length - 1); i >= 0; i--) {
            DBIdentifier sName = sNames[i];
            if (DBIdentifier.isNull(sName)) {
                continue;
            }

            if (i == (sNames.length - 1) && sNames.length != 1) {
                setName(sName.clone());
            } else {
                if (sName.getType() == DBIdentifierType.SCHEMA) {
                    setSchemaName(sName.clone());
                }
                else if (sName.getType() == DBIdentifierType.TABLE) {
                    setObjectTableName(sName.clone());
                }
            }
        }
    }
```

Image 13 Statement&Branch coverage del metodo **SetPath**

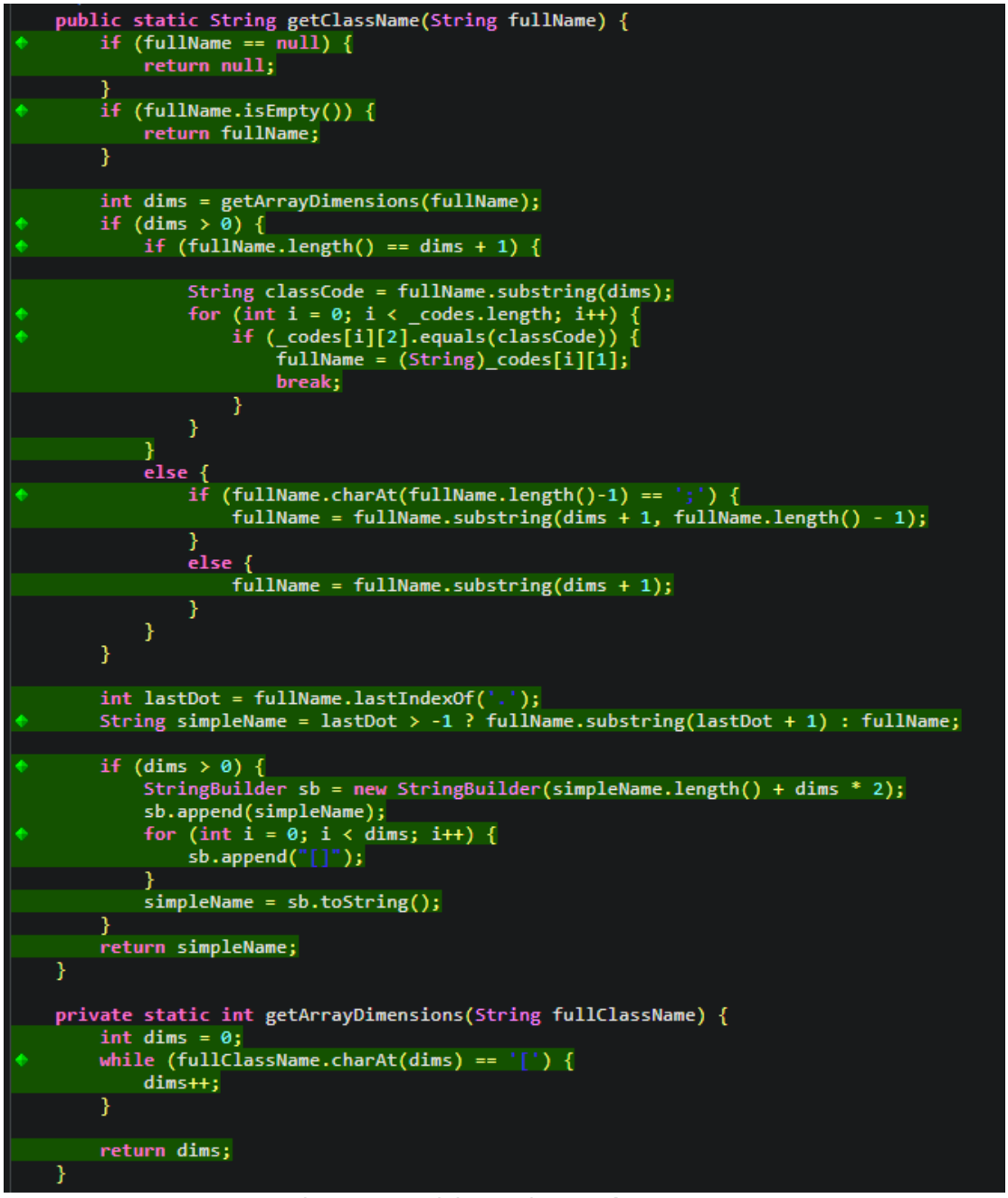


```
    public static DBIdentifier[] splitPath(DBIdentifier sName) {
        if (sName instanceof QualifiedDBIdentifier && sName.getType() != DBIdentifierType.SCHEMA) {
            QualifiedDBIdentifier path = (QualifiedDBIdentifier)sName;
            List<DBIdentifier> names = new ArrayList<>();

            if (!DBIdentifier.isNull(path.getSchemaName())) {
                names.add(path.getSchemaName().clone());
            }
            if (!DBIdentifier.isNull(path.getObjectTableName())) {
                names.add(path.getObjectTableName().clone());
            }
            if (!DBIdentifier.isNull(path.getIdentifier())) {
                names.add(((DBIdentifier)path).clone());
            }
            return names.toArray(new DBIdentifier[names.size()]);
        }
        if (sName instanceof DBIdentifier) {
            return new DBIdentifier[] { sName.clone() };
        }
        return new DBIdentifier[] {};
    }
```

Image 14 Statement&Branch coverage del metodo **SplitPath**

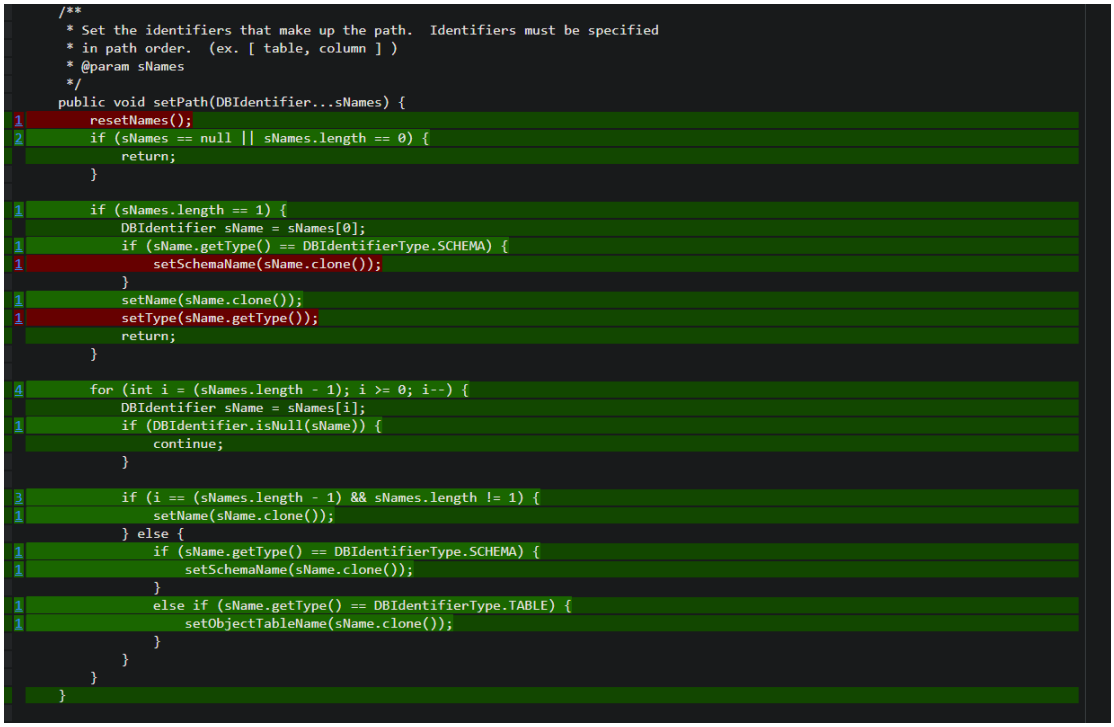
```
    public static Class toClass(String str, boolean resolve,
                                ClassLoader loader) {
        if (str == null) {
            throw new NullPointerException("str == null");
        }

        // array handling
        int dims = 0;
        while (str.endsWith("[]")) {
            dims++;
            str = str.substring(0, str.length() - 2);
        }

        // check against primitive types
        boolean primitive = false;
        if (str.indexOf('.') == -1) {
            for (int i = 0; !primitive && (i < _codes.length); i++) {
                if (_codes[i][1].equals(str)) {
                    if (dims == 0) {
                        return (Class) _codes[i][0];
                    }
                    str = (String) _codes[i][2];
                    primitive = true;
                }
            }
        }

        if (dims > 0) {
            StringBuilder buf = new StringBuilder(str.length() + dims + 2);
            for (int i = 0; i < dims; i++) {
                buf.append('[');
            }
            if (!primitive) {
                buf.append('L');
            }
            buf.append(str);
            if (!primitive) {
                buf.append(';');
            }
            str = buf.toString();
        }

        if (loader == null) {
            loader = Thread.currentThread().getContextClassLoader();
        }

        try {
            return Class.forName(str, resolve, loader);
        }
        catch (ClassNotFoundException | NoClassDefFoundError e) {
            throw new IllegalArgumentException(e.getMessage());
        }
    }
```

Image 15 Statement&Branch coverage del metodo **ToClass**



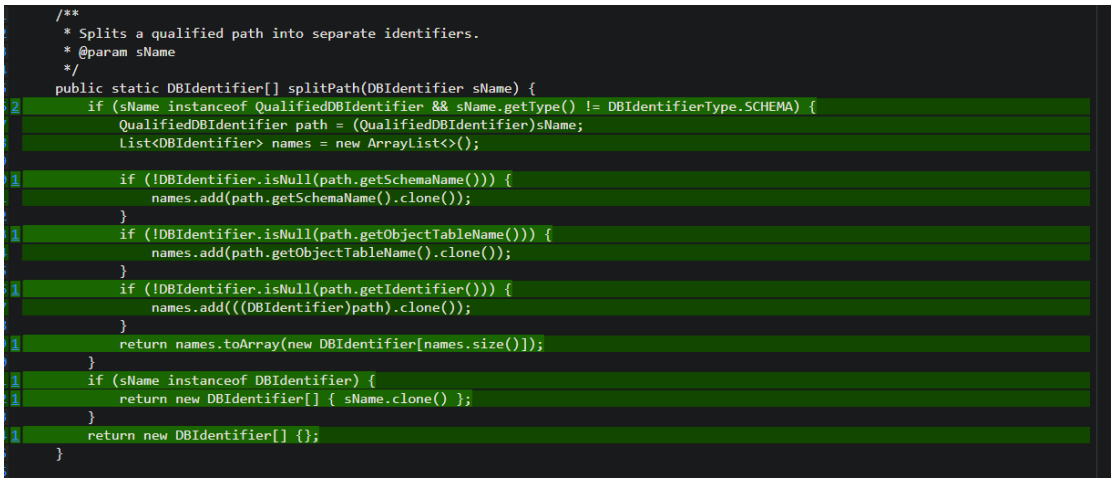
```
    public static String getClassName(String fullName) {
        if (fullName == null) {
            return null;
        }
        if (fullName.isEmpty()) {
            return fullName;
        }

        int dims = getArrayDimensions(fullName);
        if (dims > 0) {
            if (fullName.length() == dims + 1) {

                String classCode = fullName.substring(dims);
                for (int i = 0; i < _codes.length; i++) {
                    if (_codes[i][2].equals(classCode)) {
                        fullName = (String)_codes[i][1];
                        break;
                    }
                }
            }
            else {
                if (fullName.charAt(fullName.length()-1) == ';') {
                    fullName = fullName.substring(dims + 1, fullName.length() - 1);
                }
                else {
                    fullName = fullName.substring(dims + 1);
                }
            }
        }

        int lastDot = fullName.lastIndexOf('.');
        String simpleName = lastDot > -1 ? fullName.substring(lastDot + 1) : fullName;

        if (dims > 0) {
            StringBuilder sb = new StringBuilder(simpleName.length() + dims * 2);
            sb.append(simpleName);
            for (int i = 0; i < dims; i++) {
                sb.append("[]");
            }
            simpleName = sb.toString();
        }
        return simpleName;
    }

    private static int getArrayDimensions(String fullClassName) {
        int dims = 0;
        while (fullClassName.charAt(dims) == '[') {
            dims++;
        }

        return dims;
    }
```

Image 16 Statement&Branch coverage del metodo **GetClassName**



```
    /**
     * Set the identifiers that make up the path.  Identifiers must be specified
     * in path order.  (ex. [ table, column ] )
     * @param sNames
     */
    public void setPath(DBIdentifier...sNames) {
1       resetNames();
2       if (sNames == null || sNames.length == 0) {
            return;
        }

1       if (sNames.length == 1) {
            DBIdentifier sName = sNames[0];
1           if (sName.getType() == DBIdentifierType.SCHEMA) {
1               setSchemaName(sName.clone());
            }
1           setName(sName.clone());
1           setType(sName.getType());
            return;
        }

4       for (int i = (sNames.length - 1); i >= 0; i--) {
            DBIdentifier sName = sNames[i];
1           if (DBIdentifier.isNull(sName)) {
                continue;
            }

3           if (i == (sNames.length - 1) && sNames.length != 1) {
1               setName(sName.clone());
            } else {
1               if (sName.getType() == DBIdentifierType.SCHEMA) {
1                   setSchemaName(sName.clone());
                }
1               else if (sName.getType() == DBIdentifierType.TABLE) {
1                   setObjectTableName(sName.clone());
                }
            }
        }
    }
```

Image 17 Mutation coverage del metodo **SetPath**



```
    /**
     * Splits a qualified path into separate identifiers.
     * @param sName
     */
    public static DBIdentifier[] splitPath(DBIdentifier sName) {
2       if (sName instanceof QualifiedDBIdentifier && sName.getType() != DBIdentifierType.SCHEMA) {
            QualifiedDBIdentifier path = (QualifiedDBIdentifier)sName;
            List<DBIdentifier> names = new ArrayList<>();

1           if (!DBIdentifier.isNull(path.getSchemaName())) {
                names.add(path.getSchemaName().clone());
            }
1           if (!DBIdentifier.isNull(path.getObjectTableName())) {
                names.add(path.getObjectTableName().clone());
            }
1           if (!DBIdentifier.isNull(path.getIdentifier())) {
                names.add(((DBIdentifier)path).clone());
            }
1           return names.toArray(new DBIdentifier[names.size()]);
        }
1       if (sName instanceof DBIdentifier) {
1           return new DBIdentifier[] { sName.clone() };
        }
1       return new DBIdentifier[] {};
    }
```

Image 18 Mutation coverage del metodo **SplitPath**

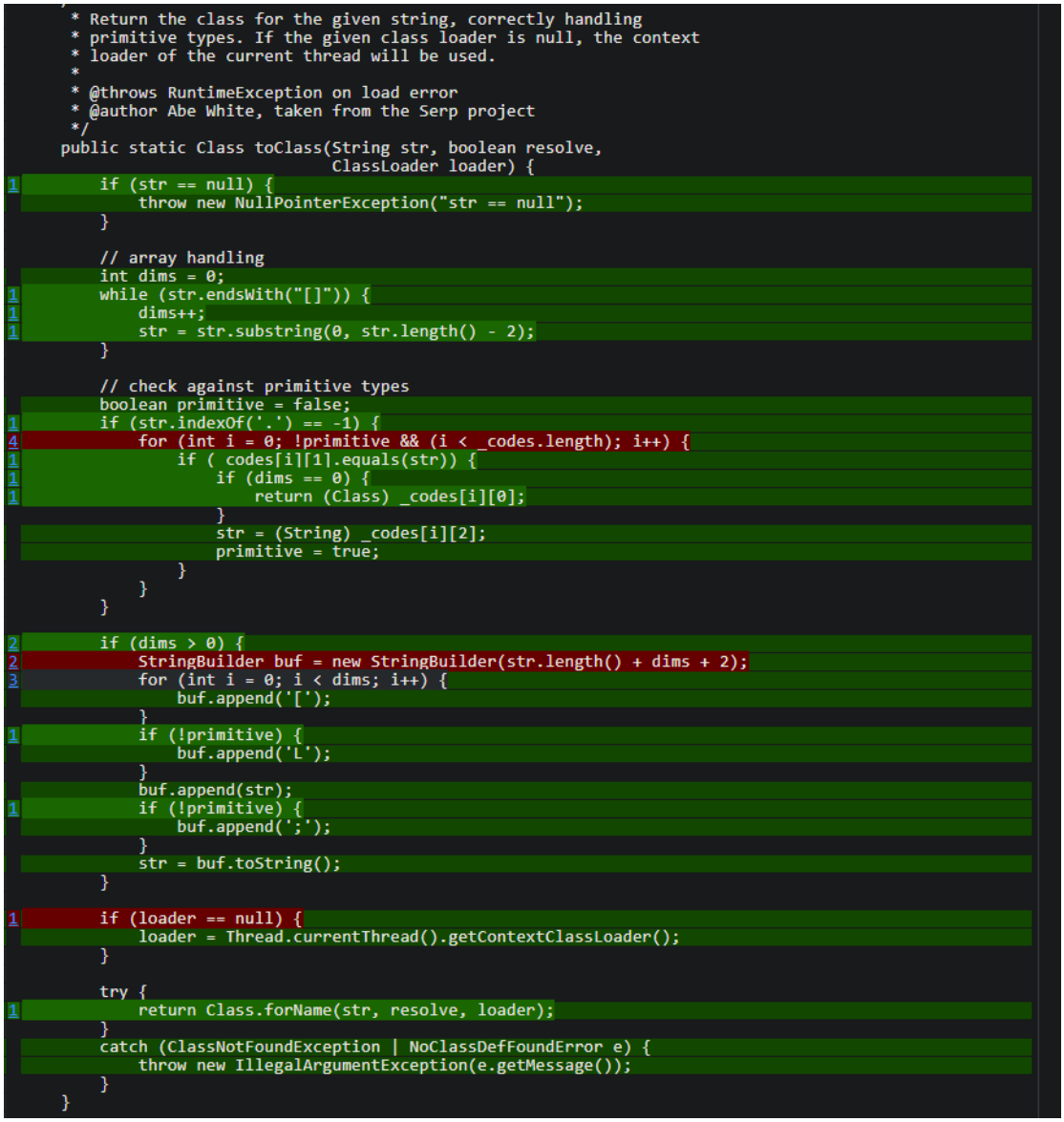
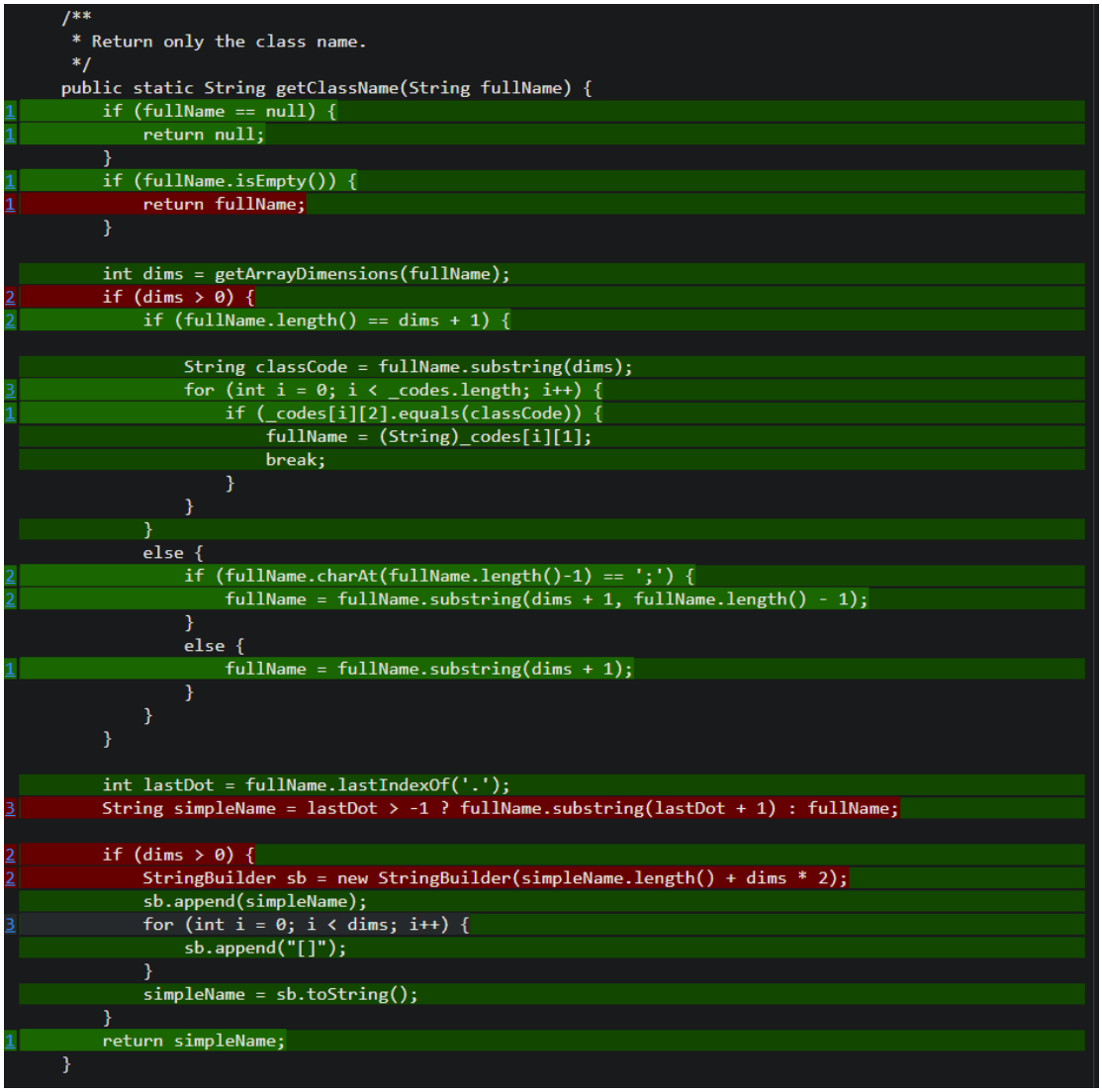


Image 19 Mutation coverage del metodo **ToClass**



```
    /**
     * Return only the class name.
     */
    public static String getClassName(String fullName) {
1       if (fullName == null) {
1           return null;
        }
1       if (fullName.isEmpty()) {
1           return fullName;
        }

        int dims = getArrayDimensions(fullName);
2       if (dims > 0) {
2           if (fullName.length() == dims + 1) {

                String classCode = fullName.substring(dims);
3               for (int i = 0; i < _codes.length; i++) {
1                   if (_codes[i][2].equals(classCode)) {
                        fullName = (String)_codes[i][1];
                        break;
                    }
                }
            }
            else {
2               if (fullName.charAt(fullName.length()-1) == ';') {
2                   fullName = fullName.substring(dims + 1, fullName.length() - 1);
                }
                else {
1                   fullName = fullName.substring(dims + 1);
                }
            }
        }

        int lastDot = fullName.lastIndexOf('.');
3       String simpleName = lastDot > -1 ? fullName.substring(lastDot + 1) : fullName;

2       if (dims > 0) {
2           StringBuilder sb = new StringBuilder(simpleName.length() + dims * 2);
            sb.append(simpleName);
3           for (int i = 0; i < dims; i++) {
                sb.append("[]");
            }
            simpleName = sb.toString();
        }
1       return simpleName;
    }
```

Image 20 Mutation coverage del metodo **GetClassName**