Report ISW2 – Software Testing

Fabiano Veglianti 0286057

Sommario

Introduzione

Il presente documento descrive le scelte e le attività effettuate per l'esame pratico del modulo di Software Testing del corso di Ingegneria del Software 2. In accordo alla consegna, quello che è stato fatto è stato progettare e implementare l'attività di testing per due progetti open-source, inserire l'attività di testing in un contesto di *Continuous Integration*, valutare l'adeguatezza dei casi di test sviluppati e, in base ai risultati ottenuti, migliorarli.

Gli strumenti utilizzati per lo svolgimento del progetto sono: *Maven* per la definizione del ciclo di build all'interno del quale si è utilizzato *Jacoco* per la generazione dei report di control-flow coverage dei casi di test sviluppati e *Pit* per l'esecuzione di mutation testing; *Travis CI* per l'esecuzione automatica del build in un ambiente controllato a seguito di ogni modifica sulla repository remota ed infine *SonarCloud* per l'analisi dei risultati dei test. I test sono stati implementati utilizzando i framework JUnit e Mockito.

I progetti open-source analizzati sono Bookkeeper e Syncope.

Questo documento è redatto seguendo una struttura ad albero, per cui si introduce prima il progetto, poi la classe ed infine i metodi della classe per cui sono stati realizzati casi di test.

Bookkeeper

Bookkeeper è un servizio di storage scalabile, tollerante ai guasti e a bassa latenza ottimizzato per carichi di lavoro real-time.

Le classi selezionate per l'attività di testing sono: *WriteCache* e *LedgerMetadataIndex*. Le motivazioni che hanno portato alla selezioni di queste classi non sono legate alla probabilità che queste classi contengano dei difetti, anche se nelle applicazioni reali questo sarebbe uno dei criteri su cui basarsi per scegliere cosa testare, ma sono state scelte queste classi tenendo conto della chiarezza del ruolo della classe all'interno del progetto e della chiarezza della documentazione.

WriteCache

La classe WriteCache implementa una cache di scrittura. La cache di scrittura consiste uno spazio di memoria diviso in segmenti. Le entry vengono aggiunte in un buffer comune e indicizzate tramite una hashmap, finché la cache non viene svuotata.

put

Il metodo *public boolean put(long ledgerId, long entryId, ByteBuf entry)* copia nella cache il contenuto del buffer *entry* identificato da *entryId* da scrivere nel ledger specificato da *ledgerId*; il metodo ritorna un valore booleano che testimonia il successo o il fallimento dell'operazione.

Volendo applicare Category Partition, in un primo momento si può esservare che i parametri *ledgerId* e *enrtyId* sono indici e in quanto tali si presuppone che un partizionamento {<0; ≥0} sia adeguato per entrambi poiché tipicamente gli indici sono valori positivi; analizzando approfonditamente il metodo è possibile osservare che ad un certo punto viene invocato '*checkBiggerEqualZero(ledgerId)*' cosa che ci testimonia che l'intento è non accettare ledgerId minori di 0. Per il parametro *entryId* si può notare che quando si cerca un'entry nell'hashmap, l'entryId=-1 testimonia l'assenza dell'entry, ne consegue che valori negativi dell'entryId sono considerati anomali. Per quanto riguarda il parametro entry, il comportamento del metodo dipende sostanzialmente da quanti bytes devono essere scritti nella cache ed in particolare se i bytes da dover essere scritti nella cache sono meno o pari ai bytes disponibili nella cache si ha un comportamento (la scrittura va a buon fine), se sono di più se ne ha un altro (la scrittura fallisce). Dunque, detta *entry.size* la dimensione del buffer da scrivere nella cache, e detta *freeCacheMemory* la memoria disponibile nella cache le tre classi di equivalenza considerate per questo parametro sono: {null, 0 < entry.size <= freeCacheMemory, entry.size > freeCacheMemory}.

I casi di test individuati sono contenuti nella *Tabella 1*.

La *Figura 1* mostra la statement coverage e la branch coverage per i metodi della classe WriteCache: i valori di statement coverage e branch coverage per il metodo put sono rispettivamente 96% e 62%. Entrando nel dettaglio del metodo put [*Figura 2*] si può osservare che i test non includono il caso in cui l'entry da scrivere ha dimensione superiore a quello di un segmento in cui è suddivisa la cache e non considerano il caso in cui scritture parallele vengano effettuate sullo stesso ledger. Nel miglioramento dell'insieme dei casi di test è stato risolto il primo problema, ma è stato lasciato irrisolto il secondo. I risultati ottenuti, mostrati in *Figura 3* e in *Figura 4*, mostrano una statement coverage del 97% e una branch coverage del 75%.

Applicando mutation testing [*Figura 7*] è possibile vedere che l'insieme dei casi di test prodotto non è in grado di uccidere tutti i mutanti del metodo put(). In particolare, si può notare che i mutanti che non vengono uccisi sono: quelli che cambiano "i confini delle condizioni", ad esempio quelli che presentano ">=" al posto di ">"; quelli che cambiano il segno degli shift logici presenti nel metodo ed infine un mutante che non produce il side-effect di aumentare il contatore del numero di entry presenti nella cache. Si noti che avendo branch coverage pari a 75% è normale aspettarsi che i mutanti che mutano su branch non coperti non vengano uccisi. Per quanto riguarda il mutante che produce side-effect invece, si può modificare l'insieme dei casi di test per includere un test che dopo aver effettuato un certo numero n di inserimenti valuti che il contatore del numero di entry nella cache sia pari ad n.

get

Il metodo *public ByteBuf get(long ledgerId, long entryId)* ritorna un buffer contenente l'entry nella cache di scrittura con id specificato che deve essere scritta sul ledger specificato; il metodo ritorna null in caso in cui l'entry richiesta non sia presente nella cache o è presente ma non deve essere scritta nel ledger specificato.

Le partizioni del dominio di input rispetto ai singoli parametri sono: {<0; ≥0} sia per ledgerId che per entryId. I motivi di ciò sono 2: il primo motivo, più naif, è che per gli indici tipicamente si usano valori maggiori o uguali a 0 e dunque si possono considerare i valori minori di 0 come anomali, mentre i restanti come normali e accettati; il secondo motivo è che i parametri ledgerId e entryId hanno lo stesso significato degli omonimi parametri del metodo put e anche per quelli ci si aspettava valori maggiori o uguali di 0 come casi normali, mentre valori minori di 0 come valori anomali.

I casi di test individuati sono contenuti nella *Tabella 2*.

In *Figura 1* si può vedere che la statement coverage e la branch coverage del metodo get() sono rispettivamente 95% e 50%. In *Figura 5* è possibile notare che si può raggiungere il 100% di statement coverage e di branch coverage includendo il caso in cui l'entry richiesta non è presente nella cache: questo caso non è raggiunto dall'insieme di casi di test sviluppato perché in caso in cui ledgerId=-1 solleva un'eccezione e non permette una copertura totale del metodo. Nel miglioramento dell'insieme dei casi di test è stato risolto questo problema e, come si può vedere dalla *Figura 3* e dalla *Figura 6*, si raggiunge il 100% sia di statement coverage che di branch coverage per questo metodo.

Applicando mutation testing [*Figura 8*] è possibile vedere che l'insieme dei casi di test prodotto non è in grado di uccidere un mutante del metodo get(); in particolare, il mutante che sopravvive è quello che sostituisce l'unsigned shift right con uno shift left nella determinazione dell'indice del segmento da cui leggere i dati.

LedgerMetadataIndex

La classe LedgerMetadataIndex mantiene un indice per i metadati dei ledgers memorizzati nel bookie. La chiave è il ledgerId e il valore è il contenuto della classe LedgerData.

Si noti che in diversi metodi di questa classe sono presenti aspetti di debugging che includono istruzioni di logging. Questi aspetti non sono stati considerati al fine dello sviluppo dei casi di test.

get

Il metodo *public LedgerData get(long ledgerId) throws IOException* restituisce i metadati associati al ledgerId passato come parametro in caso di successo, lancia un eccezione in caso i metadati non vengono trovati.

Il partizionamento del dominio di input individuato è {<0; ≥0}, questo perché l'unico parametro è *ledgerId* e sappiamo, perché lo abbiamo già incontrato, che un id accettato per un ledger deve essere maggiore o uguale a 0.

I casi di test individuati sono contenuti nella *Tabella 3*.

La *Figura 9* mostra le coverage per i metodi della classe LedgerMetadataIndex, in questo caso ci interessano quelle relative al metodo get(). Per il metodo get() si ha statement coverage del 78% e branch coverage del 75%; in *Figura 10* possiamo vedere che c'è un unico branch non percorso, questo branch non è percorso perché tutti i casi di test sono eseguiti tenendo disabilitato il debug, dunque l'istruzione di logging contenuta in quel branch non è mai eseguita.

In *Figura 11* si può vedere che applicando mutation testing tutti i mutanti vengono uccisi.

set

Il metodo *public void set(long ledgerId, LedgerData ledgerData) throws* IOException memorizza i metadati rappresentati dal parametro ledgerData relativi al ledger con id ledgerId.

Nel metodo sono presenti due diramazioni, una annidata all'altra, in particolare la prima valuta se erano già presenti metadati associati allo stesso ledgerId, mentre la seconda valuta se il log è abilitato o meno per stampare (o meno) un messaggio. Data la natura del secondo *if*, i casi di test non sono pensati per valutarne le varie alternative.

Il partizionamento del dominio di ciascun parametro deve tener conto se i metadati siano o non siano già presenti, dunque si sono individuati i casi di test mostrati in *Tabella 4*. Si può notare che l'insieme dei casi di test individuato non è minimale in relazione alla copertura delle classi di equivalenza, cioè eliminando il test t_3 dall'insieme sarebbero comunque coperte tutte le classi di equivalenza di ogni parametro, tuttavia il test t_3 è necessario per coprire un caso di esecuzione corretta della funzione in cui però i dati non erano ancora presenti.

In *Figura 9* si può vedere che per il metodo set() si ha statement coverage dell' 87% e branch coverage del 75%. In *Figura 12* si può notare che l'unico branch non percorso è quello relativo a funzionalità di debugging che si è scelto di non prendere in considerazione nello sviluppo dei casi di test.

Applicando mutation testing [*Figura 13*] è possibile notare che esiste un mutante del metodo set() che non viene ucciso. Questo mutante si differenzia perché nega la condizione di preesistenza dei dati associati allo stesso ledgerId; l'insieme dei casi di test non si accorge della mutazione perché non viene valutato il side-effect prodotto nel caso in cui non c'erano già dati associati allo stesso ledgerId, cioè il fatto che in questo caso viene aumentato il contatore dei ledger presenti. Valutare il cambiamento del contatore in risposta all'inserimento di nuovi dati permette di accorgersi della mutazione e ucciderla.

setMasterKey

Il metodo *public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException* consente di settare come valore del metadato masterKey_ del ledger specificato il valore del parametro di input masterKey.

Notiamo che il metodo si comporta diversamente se il ledger specificato è presente o meno, dunque per il parametro *ledgerId* individuiamo le partizioni {<0; ≥0 ed è presente; ≥0 e non è presente}. Per il parametro *masterKey* invece individuiamo le partizioni {null, empty, notEmpty}.

L'insieme dei casi di test sviluppati è riportato nella *Tabella 5*.

L'insieme dei casi di test sviluppato non è molto buono, infatti in *Figura 9* si può vedere che la statement coverage è pari al 69% e la branch coverage è pari al 50%. In *Figura 14* c'è il dettaglio su quali branch non sono stati percorsi. Oltre ai branch eseguiti quando il debug è attivato, notiamo che ci sono diversi branch relativi allo stato dei metadati non percorsi, in particolare: quando la masterKey è già settata a un valore diverso dalla stringa vuota, se la masterKey passata come parametro è diversa da quella memorizzata e diversa dalla stringa vuota viene sollevata un' eccezione. Per poter comprendere tutti i casi di branch coverage occorre considerare anche i casi intermedi, cioè quando la masterKey passata come parametro è differente da quella memorizzata, ma è una stringa vuota e quello in cui la masterKey passata come parametro è uguale a quella memorizzata, ma è diversa dalla stringa vuota. Con queste modifiche le metriche di coverage per il metodo setMasterKey() salgono a 87% e 85% come osservabile in *Figura 15*; in *Figura 16* possiamo osservare che, dopo il miglioramento dell'insieme dei casi di test, gli unici branch non esplorati sono quelli relativi a funzionalità di debugging.

In *Figura 17* si ha il dettaglio del mutation testing applicato allo studio di questo metodo: in particolare possiamo notare che tutti i mutanti tranne uno vengono uccisi, il mutante che sopravvive è legato ad un side-effect che abbiamo già incontrato in altri metodi e cioè l'incremento di un contatore. L'aggiunta di un caso di test che controlli il corretto incremento del contatore ucciderebbe questo mutante.

setFenced

Il metodo *public boolean setFenced(long ledgerId) throws IOException* imposta lo stato di un ledger a fenced. Il fencing è una tecnica che consiste nell'isolare un elemento di un sistema o delle risorse condivise quando avviene un malfunzionamento ed è utilizzato in Bookkeeper per garantire l'integrità dei dati scritti in un ledger. Il metodo ritorna true se il settaggio dello stato del ledger a fenced ha successo, false altrimenti.

L'unico parametro del metodo è *ledgerId*; per questo metodo non è viene effettuato alcun controllo per verificare che il valore di ledgerId sia maggiore o uguale a 0. Non essendo chiaro se questo comportamento sia voluto o meno e non sapendo, nel secondo caso, cosa dovrebbe succedere in caso non sia voluto, si è ignorato il fatto nell'implementazione dei casi di test, tuttavia lo si fa presente in questa relazione. Nonostante non si possono definire, per quanto detto, partizioni del dominio di input, si possono definire casi di test in base allo stato del ledger associato al parametro di input, in particolare: se non possediamo i dati relativi al ledger specificato ci si aspetta un' eccezione; se il ledger specificato è già settato a *fenced* il metodo termina con *false*; mentre se il ledger specificato non era ancora settato a *fenced* il metodo termina con *true*.

Per quanto appena detto, i casi di test definiti in *Tabella 6* non fanno riferimento alle partizioni in classi di equivalenza del dominio di input, ma a partizioni in classi di equivalenza dei metadati associati al parametro di input, in particolare {metadati assenti; metadati presenti e fenced=true; metadati presenti e fenced=false}.

In *Figura 9* si può vedere che l'insieme dei casi di test sviluppati comporta una statement coverage del 70% e una branch coverage del 50% per il metodo setFenced(). La *Figura 18* mostra quali branch non sono percorsi: oltre ai branch relativi al debugging, non è percorso un branch che gestisce una situazione di concorrenza, cioè quando i metadati associati ad un ledger esistono, ma prima che l'operazione di aggiornamento abbia fine questi vengano eliminati da un altro thread. Per far in modo che questa condizione si verificasse nell'insieme di casi di test prodotto, si è creata una *spy* dell'oggetto LedgerMetadataIndex e si è ridefinito il comportamento del metodo get() in modo che dopo aver trovato un ledger lo eliminasse dalla lista dei ledger memorizzati nel bookie. Con questa modifica, come è osservabile in *Figura 19* gli unici branch non percorsi sono quelli relativi a funzionalità di debugging, i nuovi valori di statement coverage e branch coverage sono rispettivamente 83% e 75% [*Figura 15*].

Applicando mutation coverage, ancora una volta, l'unico mutante non ucciso è quello che altera il side-effect di aumento del contatore dei ledger memorizzati nel bookie[*Figura 20*]. L'aggiunta di un caso di test che controlli il corretto incremento del contatore ucciderebbe questo mutante.

Syncope

Syncope è un sistema Open Source per la gestione delle identità digitali in ambienti aziendali.

Le classi selezionate per l'attività di testing sono: *Encryptor* e *RealmValidator*. Le motivazioni che hanno portato alla scelta di queste classi sono analoghe a quelle specificate per il progetto Bookkeeper.

Encryptor

La classe Encryptor implementa le funzionalità di cifratura utilizzate nel sistema.

encode

Il metodo *public String encode(final String value, final CipherAlgorithm cipherAlgorithm)* prende in input una stringa e un algoritmo di cifratura e restituisce in output il valore del stringa cifrata tramite quell'algoritmo.

I parametri di input per il metodo sono una stringa che rappresenta il testo da cifrare, per la quale possiamo effettuare un partizionamento {null, empty, notEmpty}, e l'elemento di un'enumerazione che rappresenta l'algoritmo di cifratura. Dell'enumerazione sono stati considerati tutti i possibili valori, tuttavia per verificare la correttezza dell'esecuzione dell'algoritmo sono stati utilizzati due approcci differenti: per gli algoritmi AES, SHA, SHA1, SHA256 e SHA512 vale che tutte le esecuzioni del metodo in cui *value* non cambia restituiscono lo stesso risultato e questo risultato è lo stesso che si può ottenere utilizzando dei tools online, dunque questi tools sono stati usati come oracolo. Con gli algoritmi SMD5, SSHA, SSHA1, SSHA256, SSHA512 e BCRYPT, invece, si ha che ogni esecuzione del metodo restituisce una stringa cifrata diversa anche se il valore di *value* non cambia; in questo caso la verifica della correttezza della stringa restituita è stata effettuata utilizzando il metodo *verify* della stessa classe, che tratteremo in seguito. Questo modo di procedere non è ideale perché un errore nel metodo verify potrebbe far fallire il test (e rendere più difficile il degugging) o, peggio ancora, un errore nel metodo encode, replicato nel metodo verify, potrebbe portare i test ad avere successo.

L'insieme dei casi di test sviluppati per il metodo encode() è contenuto nella *Tabella 7*.

In *Figura 21* si può vedere che all'insieme di casi di test sviluppati corrisponde una statement coverage e una branch coverage del 100% ed infatti tutti i branch e gli statement sono raggiunti [*Figura 22*].

In *Figura 23* possiamo vedere che tutti i mutanti del metodo encode() vengono uccisi dall'insieme dei casi di test sviluppati.

decode

Il metodo *public String decode(final String encoded, final CipherAlgorithm cipherAlgorithm)* prende in input la codifica di una stringa e un algoritmo di cifratura e restituisce in output il valore della stringa in chiaro decifrandola tramite quell'algoritmo.

Per il parametro *encoded* si possono individuare tre classi di equivalenza: {null, empty, notEmpty}, per il parametro *cipherAlgorithm*, invece, essendo un'enumerazione si dovrebbero considerare tutti i valori che può assumere, tuttavia non viene fatto ciò. "CipherAlgorithm" fa pensare che si stia parlando di algoritmi di cifratura, tuttavia non è propriamente così: tranne AES, gli altri algoritmi dell'enumerazione sono funzioni hash crittografiche, e dunque per definizione non invertibili, ne consegue che il metodo decode può decodificare una stringa solo se questa è stata "prodotta" utilizzando AES, altrimenti non può farlo. Per questo motivo per il parametro cipherAlgorithm si individuano le seguenti tre classi di equivalenza {null, AES, notAES}.

In virtù della discussione appena fatta, per il metodo decode() vengono definiti i casi di test presentati in *Tabella 8*.

In *Figura 21* si può vedere che l'insieme di casi di test prodotto è sufficiente per raggiungere il 100% di statement coverage e branch coverage, infatti tutti gli statement e tutti i branch sono percorsi [*Figura 24*].

Tutti i mutanti del metodo decode() vengono uccisi dall'insieme dei casi di test [*Figura 25*].

verify

Il metodo *public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded)* prende in input una stringa in chiaro, la cifratura di una stringa e un algoritmo di cifratura e valuta se la cifratura passata come parametro corrisponde alla cifratura della stringa in chiaro tramite l'algoritmo indicato.

Il parametro *value* rappresenta la stringa in chiaro e per questo si sono individuate le tre classi di equivalenza {null, empty, notEmpty}; *cipherAlgorithm* è l'elemento di un' enumerazione, dunque sono stati considerati tutti i possibili valori che può assumere; *encoded* è una stringa, tuttavia non si sono considerate le stesse classi di equivalenza di *value*, bensì dovendo essere la cifratura di *value*, per *encoded* si sono individuate le seguenti classi di equivalenza {null, =cipherAlgorithm(value), ≠cipherAlgorithm(value)}.

L'insieme dei casi di test sviluppato è riportato in *Tabella 9*.

In *Figura 21* si può vedere che l'insieme di casi di test prodotto non è sufficiente per raggiungere il 100% di statement coverage, infatti non è stato testato un caso in cui la stringa cifrata passata come parametro non è ben costruita rispetto all'algoritmo specificato (ad esempio, applicando SHA256 si ottiene una stringa a 256bit, se dunque la stringa cifrata passata come parametro è più corta si assume che la verifica fallisca e si restituisce un messaggio d'errore dicendo che non è stato possibile effettuare il confronto) [*Figura 26*]. Aggiungendo un test che colma questa lacuna si raggiunge il 100% di statement coverage e branch coverage [*Figura 27*][*Figura 28*].

L'applicazione di mutation testing ci garantisce che tutti i mutanti del metodo verify() vengono uccisi [*Figura 29*].

RealmValidator

La classe RealmValidator si occupa di validare i Realm. Un Realm definisce un albero gerarchico nel dominio della sicurezza. La documentazione[1] definisce i requisiti di ogni Realm.

isValid

Il metodo *public boolean isValid(final Realm realm, final ConstraintValidatorContext context)* attesta che un realm soddisfi i requisiti definiti nella documentazione.

I parametri di input del metodo sono *realm* per cui si sono individuate le classi di equivalenza {null, valid, invalid} e *context* per cui si sono individuate le classi di equivalenza {null, new ConstraintValidatorContext()}. Per entrambi i parametri sono stati utilizzati degli stub il cui comportamento è definito ad hoc per il caso di test che si vuole eseguire. Un'analisi più approfondita del metodo fa emergere che la suddivisione del dominio di *realm* nelle classi di equivalenza precedentemente dichiarate non è adeguato e, anzi, può essere raffinato: in particolare, dalla documentazione possiamo osservare che un realm può essere valido o se è root e non ha un parent realm o se non è root, ha un parent realm e il suo nome si adegua ad un certo pattern; segue che per poter testare adeguatamente il metodo isValid() occorre definire diversi casi di test che comprendano sia test in cui queste condizioni sono rispettate in toto, sia test in cui sono rispettate solo parzialmente o non sono per nulla rispettate.

In *Tabella 10* sono definiti i casi di test per il metodo isValid().

Si può osservare che l'insieme di casi di test definito garantisce il 100% sia di statement coverage che di branch coverage [*Figura 30*][*Figura 31*].

[1] http://syncope.apache.org/docs/2.1/reference-guide.html#realms

L'esecuzione di mutation testing fa emergere la presenza di un mutante che riesce a sopravvivere [*Figura 32*], la sopravvivenza di tale mutante è dovuta al fatto che l'istruzione mutata non altera l'esecuzione del metodo, ma disattiva la generazione di oggetti ConstraintViolation di default. L'interfaccia ConstraintValidatorContext specifica che almeno un ConstraintViolation deve essere definito e in particolare che esiste uno di default, ma se questo viene disattivato è necessario dichiararne uno personalizzato. Il metodo isValid() disattiva il ConstraintViolation di default per poi costruirne uno personalizzato a partire dal builder generato tramite l'istruzione *context.buildConstraintViolationWithTemplate()*.

Links

Nota: i link forniti per Travis CI fanno riferimento a travis-ci.com, tuttavia ancora per alcune settimane dovrebbero essere validi i link sostituendo "travis-ci.org" a "travis-ci.com".

Bookkeeper

- GitHub: https://github.com/FabianoVeglianti/bookkeeper
- Travis CI: https://travis-ci.com/github/FabianoVeglianti/bookkeeper
- SonarCloud: https://sonarcloud.io/dashboard?id=FabianoVeglianti_bookkeeper

Syncope

- GitHub: https://github.com/FabianoVeglianti/syncope
- Travis CI: https://travis-ci.com/github/FabianoVeglianti/syncope
- SonarCloud: https://sonarcloud.io/dashboard?id=FabianoVeglianti_syncope

	ledgerId	entryId	entry
t1	0	-1	null
t2	-1	0	0<size<=freeCacheMemory
t3	0	0	size>freeCacheMemory

Tabella 1: Insieme dei casi di test per WriteCache.put()

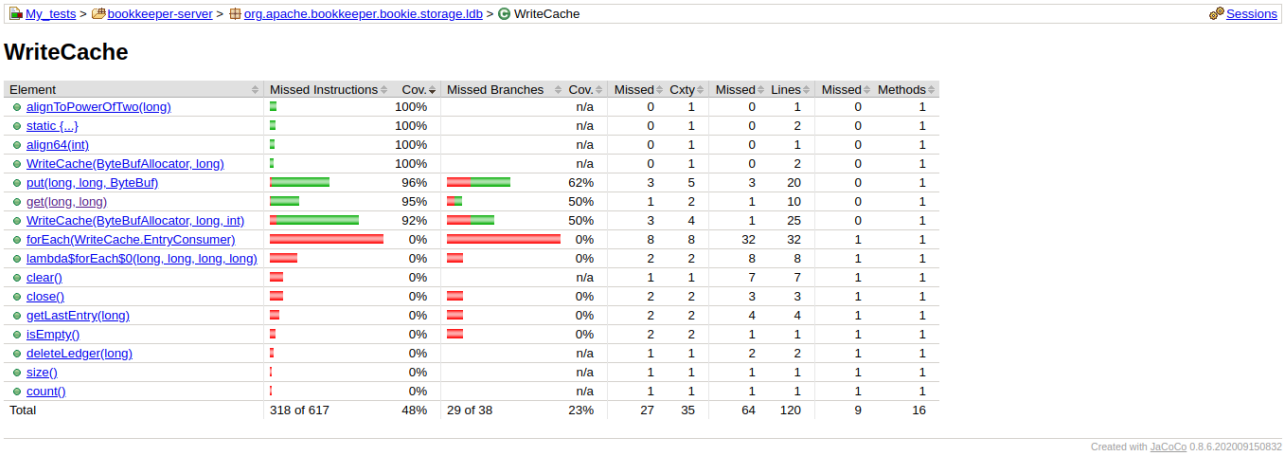


Figura 1: Coverage per WriteCache

```
131.    public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132.        int size = entry.readableBytes();
133.
134.        // Align to 64 bytes so that different threads will not contend the same L1
135.        // cache line
136.        int alignedSize = align64(size);
137.
138.        long offset;
139.        int localOffset;
140.        int segmentIdx;
141.
142.        while (true) {
143.            offset = cacheOffset.getAndAdd(alignedSize);
144.            localOffset = (int) (offset & segmentOffsetMask);
145.            segmentIdx = (int) (offset >>> segmentOffsetBits);
146.
147.            if ((offset + size) > maxCacheSize) {
148.                // Cache is full
149.                return false;
150.            } else if (maxSegmentSize - localOffset < size) {
151.                // If an entry is at the end of a segment, we need to get a new offset and try
152.                // again in next segment
153.                continue;
154.            } else {
155.                // Found a good offset
156.                break;
157.            }
158.        }
159.
160.        cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
161.
162.        // Update last entryId for ledger. This logic is to handle writes for the same
163.        // ledger coming out of order and from different thread, though in practice it
164.        // should not happen and the compareAndSet should be always uncontended.
165.        while (true) {
166.            long currentLastEntryId = lastEntryMap.get(ledgerId);
167.            if (currentLastEntryId > entryId) {
168.                // A newer entry is already there
169.                break;
170.            }
171.
172.            if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173.                break;
174.            }
175.        }
176.
177.        index.put(ledgerId, entryId, offset, size);
178.        cacheCount.increment();
179.        cacheSize.addAndGet(size);
180.        return true;
181.    }
```

Figura 2: Coverage per WriteCache.put()

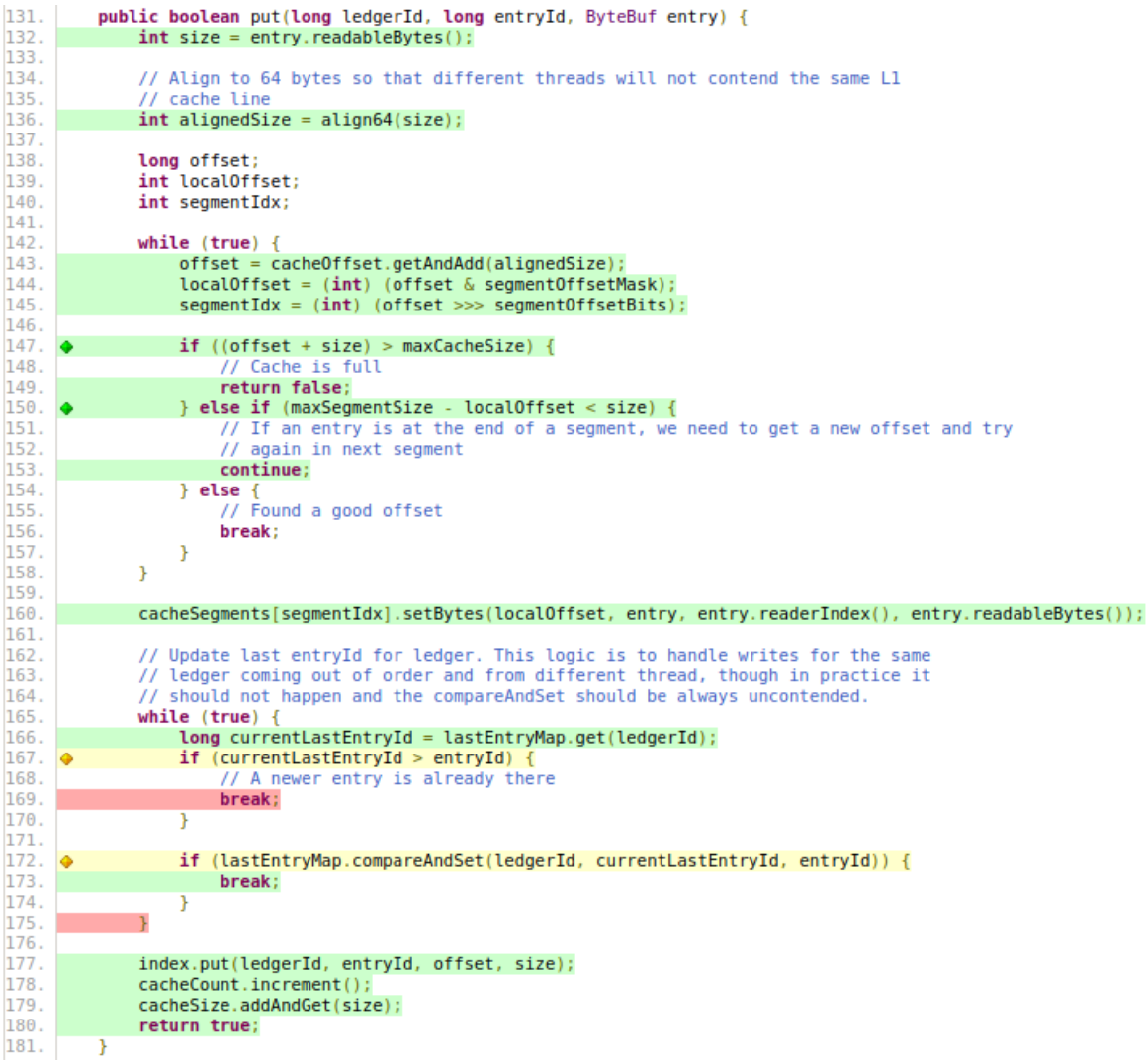
My_tests > bookkeeper-server > org.apache.bookkeeper.bookie.storage.ldb > WriteCache — Sessions

WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
get(long, long)		100%		100%	0	2	0	10	0	1
alignToPowerOfTwo(long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
align64(int)		100%		n/a	0	1	0	1	0	1
WriteCache(ByteBufAllocator, long)		100%		n/a	0	1	0	2	0	1
WriteCache(ByteBufAllocator, long, int)		98%		66%	2	4	0	25	0	1
put(long, long, ByteBuf)		97%		75%	2	5	2	20	0	1
forEach(WriteCache.EntryConsumer)		0%		0%	8	8	32	32	1	1
lambda$forEach$0(long, long, long, long)		0%		0%	2	2	8	8	1	1
clear()		0%		n/a	1	1	7	7	1	1
close()		0%		0%	2	2	3	3	1	1
getLastEntry(long)		0%		0%	2	2	4	4	1	1
isEmpty()		0%		0%	2	2	1	1	1	1
deleteLedger(long)		0%		n/a	1	1	2	2	1	1
size()		0%		n/a	1	1	1	1	1	1
count()		0%		n/a	1	1	1	1	1	1
Total	306 of 617	50%	26 of 38	31%	24	35	61	120	9	16

Created with JaCoCo 0.8.6.202009150832

Figura 3 Coverage per WriteCache dopo il miglioramento dell'insieme dei casi di test



```
131.    public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132.        int size = entry.readableBytes();
133.
134.        // Align to 64 bytes so that different threads will not contend the same L1
135.        // cache line
136.        int alignedSize = align64(size);
137.
138.        long offset;
139.        int localOffset;
140.        int segmentIdx;
141.
142.        while (true) {
143.            offset = cacheOffset.getAndAdd(alignedSize);
144.            localOffset = (int) (offset & segmentOffsetMask);
145.            segmentIdx = (int) (offset >>> segmentOffsetBits);
146.
147.            if ((offset + size) > maxCacheSize) {
148.                // Cache is full
149.                return false;
150.            } else if (maxSegmentSize - localOffset < size) {
151.                // If an entry is at the end of a segment, we need to get a new offset and try
152.                // again in next segment
153.                continue;
154.            } else {
155.                // Found a good offset
156.                break;
157.            }
158.        }
159.
160.        cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
161.
162.        // Update last entryId for ledger. This logic is to handle writes for the same
163.        // ledger coming out of order and from different thread, though in practice it
164.        // should not happen and the compareAndSet should be always uncontended.
165.        while (true) {
166.            long currentLastEntryId = lastEntryMap.get(ledgerId);
167.            if (currentLastEntryId > entryId) {
168.                // A newer entry is already there
169.                break;
170.            }
171.
172.            if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173.                break;
174.            }
175.        }
176.
177.        index.put(ledgerId, entryId, offset, size);
178.        cacheCount.increment();
179.        cacheSize.addAndGet(size);
180.        return true;
181.    }
```

Figura 4: Coverage per WriteCache.put() dopo il miglioramento dell'insieme dei casi di test

	ledgerId	entryId
t1	0	0
t2	-1	-1

Tabella 2: Insieme dei casi di test per WriteCache.get()

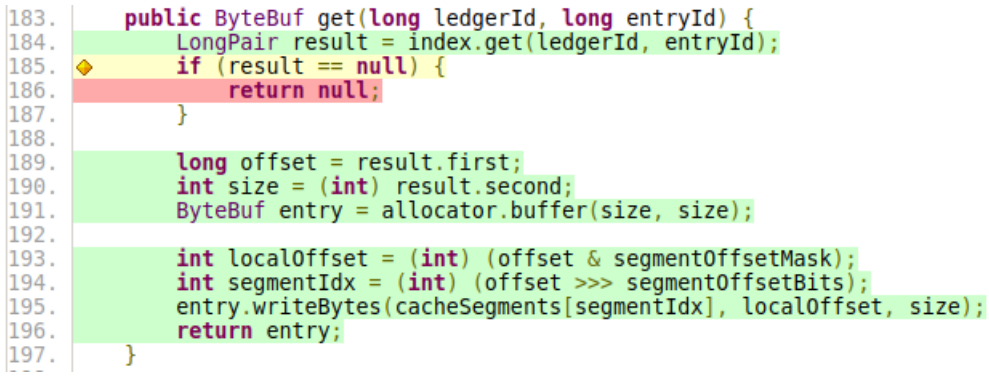


Figura 5: Coverage per WriteCache.get()

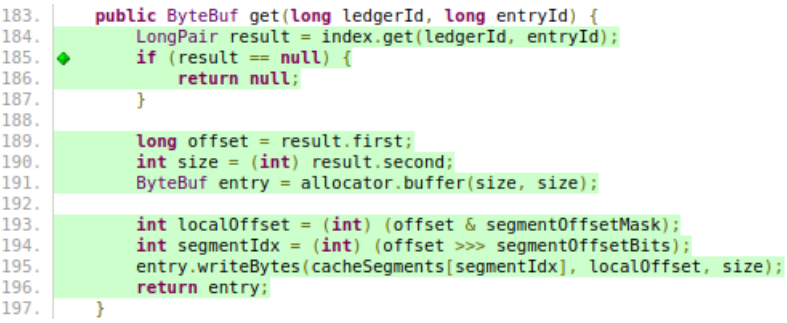


Figura 6: Coverage per WriteCache.get() dopo il miglioramento dell'insieme dei casi di test

```
131     public boolean put(long ledgerId, long entryId, ByteBuf entry) {
132         int size = entry.readableBytes();
133
134         // Align to 64 bytes so that different threads will not contend the same L1
135         // cache line
136         int alignedSize = align64(size);
137
138         long offset;
139         int localOffset;
140         int segmentIdx;
141
142         while (true) {
143             offset = cacheOffset.getAndAdd(alignedSize);
144 1           localOffset = (int) (offset & segmentOffsetMask);
145 1           segmentIdx = (int) (offset >>> segmentOffsetBits);
146
147 3           if ((offset + size) > maxCacheSize) {
148                 // Cache is full
149 1               return false;
150 3           } else if (maxSegmentSize - localOffset < size) {
151                 // If an entry is at the end of a segment, we need to get a new offset and try
152                 // again in next segment
153                 continue;
154             } else {
155                 // Found a good offset
156                 break;
157             }
158         }
159
160         cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
161
162         // Update last entryId for ledger. This logic is to handle writes for the same
163         // ledger coming out of order and from different thread, though in practice it
164         // should not happen and the compareAndSet should be always uncontended.
165         while (true) {
166             long currentLastEntryId = lastEntryMap.get(ledgerId);
167 2           if (currentLastEntryId > entryId) {
168                 // A newer entry is already there
169                 break;
170             }
171
172 1           if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
173                 break;
174             }
175         }
176
177         index.put(ledgerId, entryId, offset, size);
178 1       cacheCount.increment();
179         cacheSize.addAndGet(size);
180 1       return true;
181     }
```

Figura 7: Mutation Testing su WriteCache.put()

```
183     public ByteBuf get(long ledgerId, long entryId) {
184         LongPair result = index.get(ledgerId, entryId);
185 1       if (result == null) {
186             return null;
187         }
188
189         long offset = result.first;
190         int size = (int) result.second;
191         ByteBuf entry = allocator.buffer(size, size);
192
193 1       int localOffset = (int) (offset & segmentOffsetMask);
194 1       int segmentIdx = (int) (offset >>> segmentOffsetBits);
195         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196 1       return entry;
197     }
```

Figura 8: Mutation Testing su WriteCache.get()

	ledgerId
t1	-1
t2	0

Tabella 3: Insieme di casi di test per LedgerMetadataIndex.get()

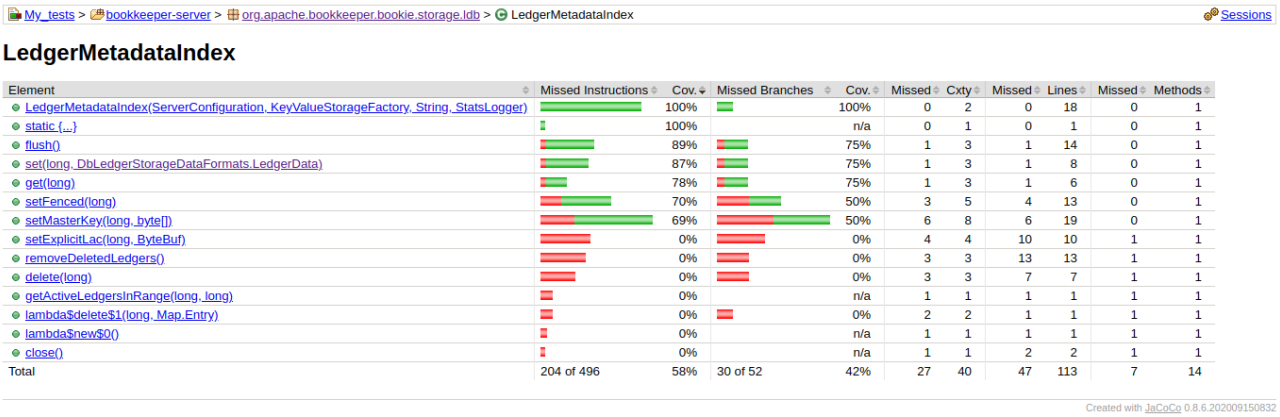


Figura 9: Coverage per LedgerMetadataIndex

```
102.    public LedgerData get(long ledgerId) throws IOException {
103.        LedgerData ledgerData = ledgers.get(ledgerId);
104.        if (ledgerData == null) {
105.            if (log.isDebugEnabled()) {
106.                log.debug("Ledger not found {}", ledgerId);
107.            }
108.            throw new Bookie.NoLedgerException(ledgerId);
109.        }
110.
111.        return ledgerData;
112.    }
```

Figura 10: Coverage per LedgerMetadataIndex.get()

```
102      public LedgerData get(long ledgerId) throws IOException {
103          LedgerData ledgerData = ledgers.get(ledgerId);
104 1        if (ledgerData == null) {
105              if (log.isDebugEnabled()) {
106                  log.debug("Ledger not found {}", ledgerId);
107              }
108              throw new Bookie.NoLedgerException(ledgerId);
109          }
110
111 1        return ledgerData;
112      }
```

Figura 11: Mutation Testing su LedgerMetadataIndex.get()

	ledgerId	ledgerData	datiPresenti
t1	-1	null	false
t2	0	notNull	true
t3	0	notNull	false

Tabella 4: Insieme di casi di test per LedgerMetadataIndex.set()

```
114.    public void set(long ledgerId, LedgerData ledgerData) throws IOException {
115.        ledgerData = LedgerData.newBuilder(ledgerData).setExists(true).build();
116.
117.        if (ledgers.put(ledgerId, ledgerData) == null) {
118.            if (log.isDebugEnabled()) {
119.                log.debug("Added new ledger {}", ledgerId);
120.            }
121.            ledgersCount.incrementAndGet();
122.        }
123.
124.        pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
125.        pendingDeletedLedgers.remove(ledgerId);
126.    }
```

Figura 12: Coverage per LedgerMetadataIndex.set()

```
114     public void set(long ledgerId, LedgerData ledgerData) throws IOException {
115         ledgerData = LedgerData.newBuilder(ledgerData).setExists(true).build();
116
117 1       if (ledgers.put(ledgerId, ledgerData) == null) {
118             if (log.isDebugEnabled()) {
119                 log.debug("Added new ledger {}", ledgerId);
120             }
121             ledgersCount.incrementAndGet();
122         }
123
124         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
125         pendingDeletedLedgers.remove(ledgerId);
126     }
```

Figura 13: Mutation Testing per LedgerMetadataIndex.set()

	ledgerId	masterKey
t1	<0	null
t2	>=0 ed è presente	empty
t3	>=0 e non è presente	Non empty

Tabella 5: Insieme dei casi di test per LedgerMetadataIndex.setMasterKey()

```
175.     public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException {
176.         LedgerData ledgerData = ledgers.get(ledgerId);
177.         if (ledgerData == null) {
178.             // New ledger inserted
179.             ledgerData = LedgerData.newBuilder().setExists(true).setFenced(false)
180.                     .setMasterKey(ByteString.copyFrom(masterKey)).build();
181.             if (log.isDebugEnabled()) {
182.                 log.debug("Inserting new ledger {}", ledgerId);
183.             }
184.         } else {
185.             byte[] storedMasterKey = ledgerData.getMasterKey().toByteArray();
186.             if (ArrayUtil.isArrayAllZeros(storedMasterKey)) {
187.                 // update master key of the ledger
188.                 ledgerData = LedgerData.newBuilder(ledgerData).setMasterKey(ByteString.copyFrom(masterKey)).build();
189.                 if (log.isDebugEnabled()) {
190.                     log.debug("Replace old master key {} with new master key {}", storedMasterKey, masterKey);
191.                 }
192.             } else if (!Arrays.equals(storedMasterKey, masterKey) && !ArrayUtil.isArrayAllZeros(masterKey)) {
193.                 log.warn("Ledger {} masterKey in db can only be set once.", ledgerId);
194.                 throw new IOException(BookieException.create(BookieException.Code.IllegalOpException));
195.             }
196.         }
197.
198.         if (ledgers.put(ledgerId, ledgerData) == null) {
199.             ledgersCount.incrementAndGet();
200.         }
201.
202.         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
203.         pendingDeletedLedgers.remove(ledgerId);
204.     }
```

Figura 14: Coverage per LedgerMetadataIndex.setMasterKey()

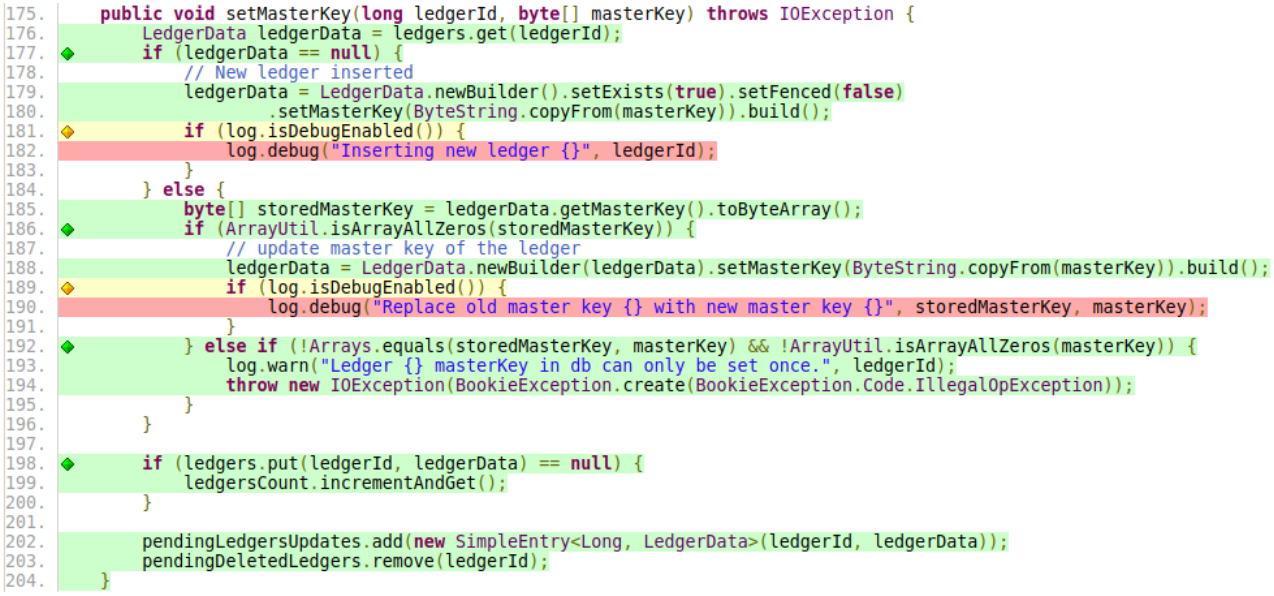
My_tests > bookkeeper-server > org.apache.bookkeeper.bookie.storage.ldb > LedgerMetadataIndex — Sessions

LedgerMetadataIndex

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
LedgerMetadataIndex(ServerConfiguration, KeyValueStorageFactory, String, StatsLogger)		100%		100%	0	2	0	18	0	1
static {...}		100%		n/a	0	1	0	1	0	1
flush()		89%		75%	1	3	1	14	0	1
set(long, DbLedgerStorageDataFormats.LedgerData)		87%		75%	1	3	1	8	0	1
setMasterKey(long, byte[])		87%		85%	2	8	2	19	0	1
removeDeletedLedgers()		86%		75%	1	3	1	13	0	1
setFenced(long)		83%		75%	2	5	2	13	0	1
delete(long)		83%		50%	2	3	1	7	0	1
get(long)		78%		75%	1	3	1	6	0	1
setExplicitLac(long, ByteBuf)		0%		0%	4	4	10	10	1	1
getActiveLedgersInRange(long, long)		0%		n/a	1	1	1	1	1	1
lambda$delete$1(long, Map.Entry)		0%		0%	2	2	1	1	1	1
lambdanew0()		0%		n/a	1	1	1	1	1	1
close()		0%		n/a	1	1	2	2	1	1
Total	121 of 496	75%	18 of 52	65%	19	40	23	113	5	14

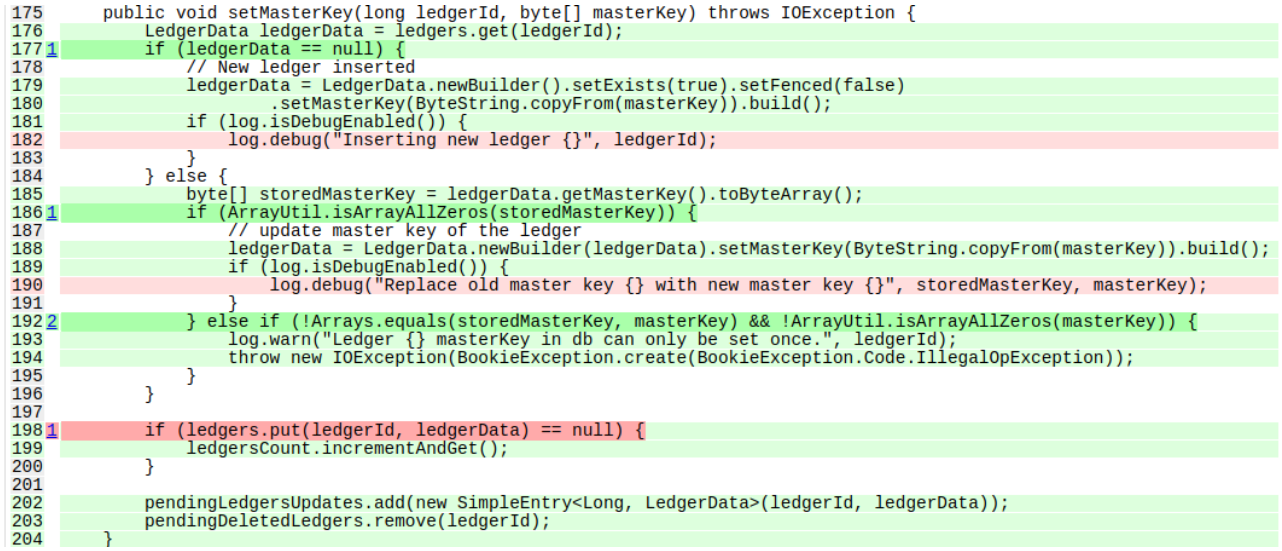
Created with JaCoCo 0.8.6.202009150832

Figura 15: Coverage per LedgerMetadataIndex dopo il miglioramento dell'insieme dei casi di test



```
175.    public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException {
176.        LedgerData ledgerData = ledgers.get(ledgerId);
177.        if (ledgerData == null) {
178.            // New ledger inserted
179.            ledgerData = LedgerData.newBuilder().setExists(true).setFenced(false)
180.                    .setMasterKey(ByteString.copyFrom(masterKey)).build();
181.            if (log.isDebugEnabled()) {
182.                log.debug("Inserting new ledger {}", ledgerId);
183.            }
184.        } else {
185.            byte[] storedMasterKey = ledgerData.getMasterKey().toByteArray();
186.            if (ArrayUtil.isArrayAllZeros(storedMasterKey)) {
187.                // update master key of the ledger
188.                ledgerData = LedgerData.newBuilder(ledgerData).setMasterKey(ByteString.copyFrom(masterKey)).build();
189.                if (log.isDebugEnabled()) {
190.                    log.debug("Replace old master key {} with new master key {}", storedMasterKey, masterKey);
191.                }
192.            } else if (!Arrays.equals(storedMasterKey, masterKey) && !ArrayUtil.isArrayAllZeros(masterKey)) {
193.                log.warn("Ledger {} masterKey in db can only be set once.", ledgerId);
194.                throw new IOException(BookieException.create(BookieException.Code.IllegalOpException));
195.            }
196.        }
197.
198.        if (ledgers.put(ledgerId, ledgerData) == null) {
199.            ledgersCount.incrementAndGet();
200.        }
201.
202.        pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
203.        pendingDeletedLedgers.remove(ledgerId);
204.    }
```

Figura 16: Coverage per LedgerMetadataIndex.setMasterKey() dopo il miglioramento dell'insieme dei casi di test



```
175     public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException {
176         LedgerData ledgerData = ledgers.get(ledgerId);
177 1       if (ledgerData == null) {
178             // New ledger inserted
179             ledgerData = LedgerData.newBuilder().setExists(true).setFenced(false)
180                     .setMasterKey(ByteString.copyFrom(masterKey)).build();
181             if (log.isDebugEnabled()) {
182                 log.debug("Inserting new ledger {}", ledgerId);
183             }
184         } else {
185             byte[] storedMasterKey = ledgerData.getMasterKey().toByteArray();
186 1           if (ArrayUtil.isArrayAllZeros(storedMasterKey)) {
187                 // update master key of the ledger
188                 ledgerData = LedgerData.newBuilder(ledgerData).setMasterKey(ByteString.copyFrom(masterKey)).build();
189                 if (log.isDebugEnabled()) {
190                     log.debug("Replace old master key {} with new master key {}", storedMasterKey, masterKey);
191                 }
192 2           } else if (!Arrays.equals(storedMasterKey, masterKey) && !ArrayUtil.isArrayAllZeros(masterKey)) {
193                 log.warn("Ledger {} masterKey in db can only be set once.", ledgerId);
194                 throw new IOException(BookieException.create(BookieException.Code.IllegalOpException));
195             }
196         }
197
198 1       if (ledgers.put(ledgerId, ledgerData) == null) {
199             ledgersCount.incrementAndGet();
200         }
201
202         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
203         pendingDeletedLedgers.remove(ledgerId);
204     }
```

Figura 17: Mutation Testing per LedgerMetadataIndex.setMasterKey()

	Stato metadati
t1	Metadati assenti
t2	Metadati presenti e fenced=true
t3	Metadati presenti e fenced=false

Tabella 6: Insieme dei casi di test per LedgerMetadataIndex.setFenced()

```
150.    public boolean setFenced(long ledgerId) throws IOException {
151.        LedgerData ledgerData = get(ledgerId);
152.        if (ledgerData.getFenced()) {
153.            return false;
154.        }
155.
156.        LedgerData newLedgerData = LedgerData.newBuilder(ledgerData).setFenced(true).build();
157.
158.        if (ledgers.put(ledgerId, newLedgerData) == null) {
159.            // Ledger had been deleted
160.            if (log.isDebugEnabled()) {
161.                log.debug("Re-inserted fenced ledger {}", ledgerId);
162.            }
163.            ledgersCount.incrementAndGet();
164.        } else {
165.            if (log.isDebugEnabled()) {
166.                log.debug("Set fenced ledger {}", ledgerId);
167.            }
168.        }
169.
170.        pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
171.        pendingDeletedLedgers.remove(ledgerId);
172.        return true;
173.    }
```

Figura 18: Coverage per LedgerMetadataIndex.setFenced()

```
150.    public boolean setFenced(long ledgerId) throws IOException {
151.        LedgerData ledgerData = get(ledgerId);
152.        if (ledgerData.getFenced()) {
153.            return false;
154.        }
155.
156.        LedgerData newLedgerData = LedgerData.newBuilder(ledgerData).setFenced(true).build();
157.
158.        if (ledgers.put(ledgerId, newLedgerData) == null) {
159.            // Ledger had been deleted
160.            if (log.isDebugEnabled()) {
161.                log.debug("Re-inserted fenced ledger {}", ledgerId);
162.            }
163.            ledgersCount.incrementAndGet();
164.        } else {
165.            if (log.isDebugEnabled()) {
166.                log.debug("Set fenced ledger {}", ledgerId);
167.            }
168.        }
169.
170.        pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
171.        pendingDeletedLedgers.remove(ledgerId);
172.        return true;
173.    }
```

Figura 19: Coverage per LedgetMetadataIndex.setFenced() dopo il miglioramento dell'insieme dei casi di test

```
150      public boolean setFenced(long ledgerId) throws IOException {
151          LedgerData ledgerData = get(ledgerId);
152 1        if (ledgerData.getFenced()) {
153 1            return false;
154          }
155
156          LedgerData newLedgerData = LedgerData.newBuilder(ledgerData).setFenced(true).build();
157
158 1        if (ledgers.put(ledgerId, newLedgerData) == null) {
159              // Ledger had been deleted
160              if (log.isDebugEnabled()) {
161                  log.debug("Re-inserted fenced ledger {}", ledgerId);
162              }
163              ledgersCount.incrementAndGet();
164          } else {
165              if (log.isDebugEnabled()) {
166                  log.debug("Set fenced ledger {}", ledgerId);
167              }
168          }
169
170          pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
171          pendingDeletedLedgers.remove(ledgerId);
172 1        return true;
173      }
```

Figura 20: Mutation Testing per LedgerMetadataIndex.setFenced()

	value	cipherAlgorithm
t1	null	SHA
t2	nonEmptyString	SHA1
t3	nonEmptyString	null
t4	nonEmptyString	AES
t5	nonEmptyString	SHA256
t6	nonEmptyString	SHA512
t7	nonEmptyString	BCRYPT
t8	nonEmptyString	SMD5
t9	emptyString	SSHA
t10	nonEmptyString	SSHA1
t11	nonEmptyString	SSHA256
t12	nonEmptyString	SSHA512

Tabella 7: Insieme dei casi di test per Encryptor.encode()

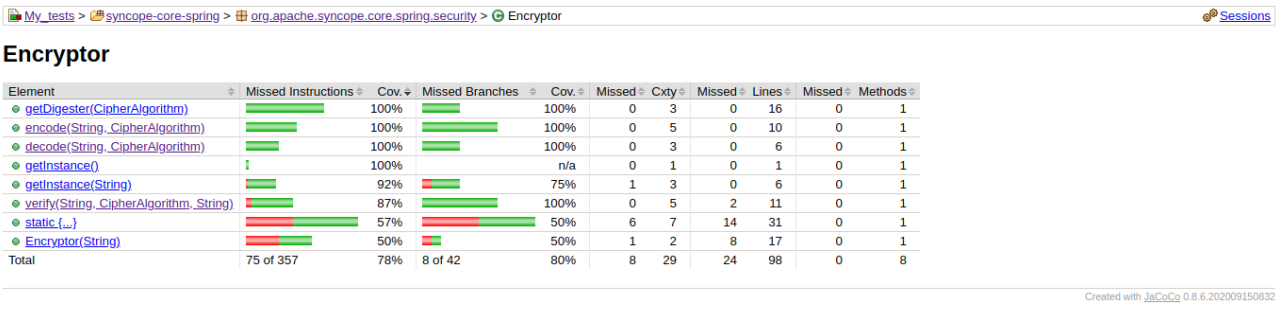


Figura 21: Coverage per Encryptor

```
172.    public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
173.            throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyExcepti
174.            IllegalBlockSizeException, BadPaddingException {
175.
176.        String encoded = null;
177.
178.        if (value != null) {
179.            if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
180.                Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
181.                cipher.init(Cipher.ENCRYPT_MODE, keySpec);
182.
183.                encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)))
184.            } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
185.                encoded = BCrypt.hashpw(value, BCrypt.gensalt());
186.            } else {
187.                encoded = getDigester(cipherAlgorithm).digest(value);
188.            }
189.        }
190.
191.        return encoded;
192.    }
```

Figura 22: Coverage per Encryptor.encode()

```
172     public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
173             throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
174             IllegalBlockSizeException, BadPaddingException {
175
176         String encoded = null;
177
178 1       if (value != null) {
179 2           if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
180                 Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
181 1               cipher.init(Cipher.ENCRYPT_MODE, keySpec);
182
183                 encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)));
184 1           } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
185                 encoded = BCrypt.hashpw(value, BCrypt.gensalt());
186             } else {
187                 encoded = getDigester(cipherAlgorithm).digest(value);
188             }
189         }
190
191 1       return encoded;
192     }
```

Figura 23: Mutation Testing per Encryptor.encode()

	encoded	cipherAlgorithm
t1	null	SHA
t2	emptyString	AES
t3	nonEmptyString	null

Tabella 8: Insieme di casi di test per il metodo decode()

```
214.    public String decode(final String encoded, final CipherAlgorithm cipherAlgorithm)
215.            throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
216.            IllegalBlockSizeException, BadPaddingException {
217.
218.        String decoded = null;
219.
220.        if (encoded != null && cipherAlgorithm == CipherAlgorithm.AES) {
221.            Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
222.            cipher.init(Cipher.DECRYPT_MODE, keySpec);
223.
224.            decoded = new String(cipher.doFinal(Base64.getDecoder().decode(encoded)), StandardCharsets.UTF_8);
225.        }
226.
227.        return decoded;
228.    }
```

Figura 24: Coverage per Encryptor.decode()

```
214     public String decode(final String encoded, final CipherAlgorithm cipherAlgorithm)
215             throws UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
216             IllegalBlockSizeException, BadPaddingException {
217
218         String decoded = null;
219
220 2       if (encoded != null && cipherAlgorithm == CipherAlgorithm.AES) {
221             Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
222 1           cipher.init(Cipher.DECRYPT_MODE, keySpec);
223
224             decoded = new String(cipher.doFinal(Base64.getDecoder().decode(encoded)), StandardCharsets.UTF_8);
225         }
226
227 1       return decoded;
228     }
```

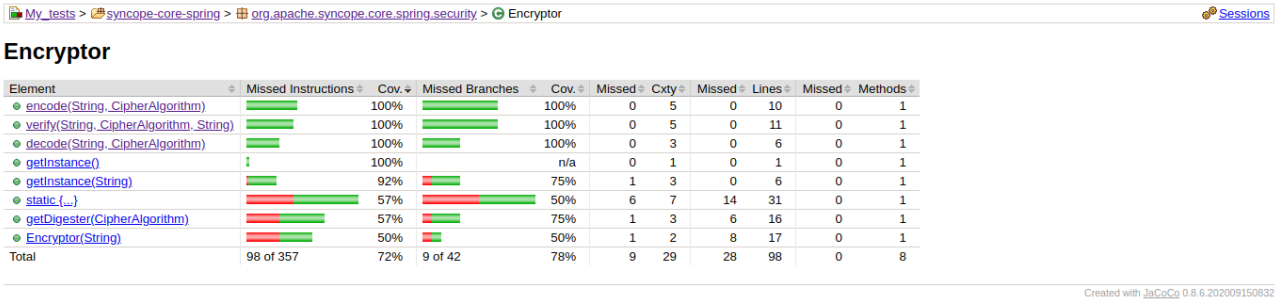
Figura 25: Mutation Testing per Encryptor.decode()

	value	cipherAlgorithm	encoded
t1	null	SHA	=cipherAlgorithm(value)
t2	emptyString	SHA1	null
t3	nonEmptyString	null	≠cipherAlgorithm(value)
t4	nonEmptyString	SHA256	≠cipherAlgorithm(value)
t5	nonEmptyString	SHA512	=cipherAlgorithm(value)
t6	nonEmptyString	BCRYPT	≠cipherAlgorithm(value)
t7	nonEmptyString	AES	=cipherAlgorithm(value)
t8	nonEmptyString	SMD5	≠cipherAlgorithm(value)
t9	nonEmptyString	SSHA	=cipherAlgorithm(value)
t10	nonEmptyString	SSHA1	≠cipherAlgorithm(value)
t11	nonEmptyString	SSHA256	=cipherAlgorithm(value)
t12	nonEmptyString	SSHA512	≠cipherAlgorithm(value)

Tabella 9: Insieme dei casi di test per Encryptor.verify()

```
194.    public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
195.        boolean verified = false;
196.
197.        try {
198.            if (value != null) {
199.                if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
200.                    verified = encode(value, cipherAlgorithm).equals(encoded);
201.                } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
202.                    verified = BCrypt.checkpw(value, encoded);
203.                } else {
204.                    verified = getDigester(cipherAlgorithm).matches(value, encoded);
205.                }
206.            }
207.        } catch (Exception e) {
208.            LOG.error("Could not verify encoded value", e);
209.        }
210.
211.        return verified;
212.    }
```

Figura 26: Coverage per Encryptor.verify()



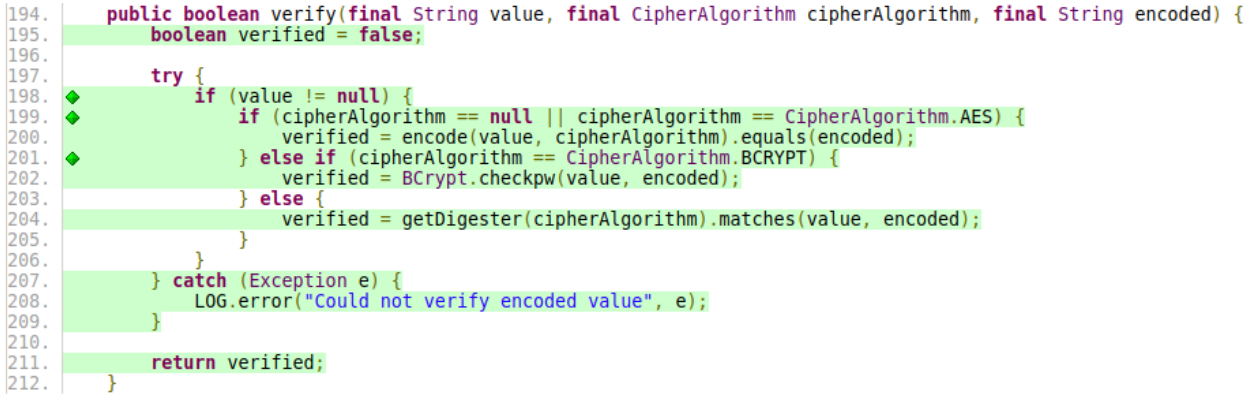
My_tests > syncope-core-spring > org.apache.syncope.core.spring.security > Encryptor — Sessions

Encryptor

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
encode(String, CipherAlgorithm)		100%		100%	0	5	0	10	0	1
verify(String, CipherAlgorithm, String)		100%		100%	0	5	0	11	0	1
decode(String, CipherAlgorithm)		100%		100%	0	3	0	6	0	1
getInstance()		100%		n/a	0	1	0	1	0	1
getInstance(String)		92%		75%	1	3	0	6	0	1
static {...}		57%		50%	6	7	14	31	0	1
getDigester(CipherAlgorithm)		57%		75%	1	3	6	16	0	1
Encryptor(String)		50%		50%	1	2	8	17	0	1
Total	98 of 357	72%	9 of 42	78%	9	29	28	98	0	8

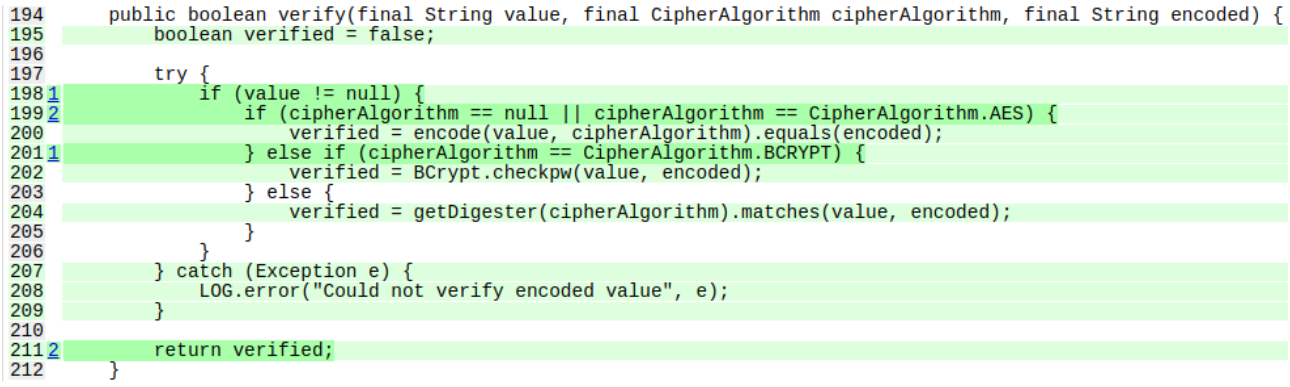
Created with JaCoCo 0.8.6.202009150832

Figura 27: Coverage per Encryptor dopo il miglioramento dell'insieme dei casi di test



```
194.    public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
195.        boolean verified = false;
196.
197.        try {
198.            if (value != null) {
199.                if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
200.                    verified = encode(value, cipherAlgorithm).equals(encoded);
201.                } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
202.                    verified = BCrypt.checkpw(value, encoded);
203.                } else {
204.                    verified = getDigester(cipherAlgorithm).matches(value, encoded);
205.                }
206.            }
207.        } catch (Exception e) {
208.            LOG.error("Could not verify encoded value", e);
209.        }
210.
211.        return verified;
212.    }
```

Figura 28: Coverage per Encryptor.verify() dopo il miglioramento dell'insieme dei casi di test

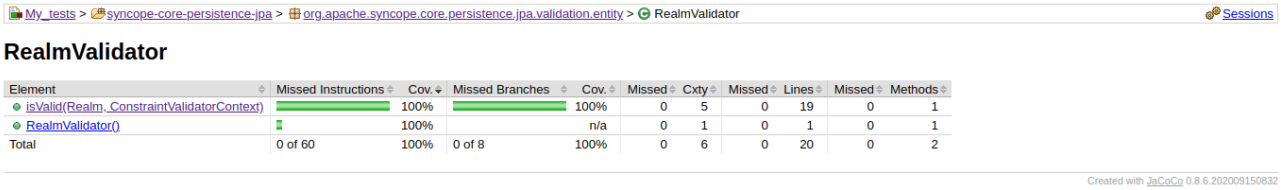


```
194     public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
195         boolean verified = false;
196
197         try {
198 1           if (value != null) {
199 2               if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
200                     verified = encode(value, cipherAlgorithm).equals(encoded);
201 1               } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
202                     verified = BCrypt.checkpw(value, encoded);
203                 } else {
204                     verified = getDigester(cipherAlgorithm).matches(value, encoded);
205                 }
206             }
207         } catch (Exception e) {
208             LOG.error("Could not verify encoded value", e);
209         }
210
211 2       return verified;
212     }
```

Figura 29: Mutation Testing per Encryptor.verify()

	realm	context
t1	null	null
t2	valid_root	New obj
t3	invalid_root_with_parent	New obj
t4	invalid_no_root_with_no_parent	New obj
t5	invalid_no_root_out_pattern	New obj

Tabella 10: Insieme di casi di test per RealmValidator.isValid()



My_tests > syncope-core-persistence-jpa > org.apache.syncope.core.persistence.jpa.validation.entity > RealmValidator — Sessions

RealmValidator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
isValid(Realm, ConstraintValidatorContext)		100%		100%	0	5	0	19	0	1
RealmValidator()		100%		n/a	0	1	0	1	0	1
Total	0 of 60	100%	0 of 8	100%	0	6	0	20	0	2

Created with JaCoCo 0.8.6.202009150832

Figura 30: Coverage per RealmValidator

```
30.     public boolean isValid(final Realm realm, final ConstraintValidatorContext context) {
31.         context.disableDefaultConstraintViolation();
32.
33.         boolean isValid = true;
34.
35.         if (SyncopeConstants.ROOT_REALM.equals(realm.getName())) {
36.             if (realm.getParent() != null) {
37.                 isValid = false;
38.
39.                 context.buildConstraintViolationWithTemplate(
40.                         getTemplate(EntityViolationType.InvalidRealm, "Root realm cannot have a parent realm")).
41.                         addPropertyNode("parent").addConstraintViolation();
42.             }
43.         } else {
44.             if (realm.getParent() == null) {
45.                 isValid = false;
46.
47.                 context.buildConstraintViolationWithTemplate(
48.                         getTemplate(EntityViolationType.InvalidRealm, "A realm needs to reference a parent realm")).
49.                         addPropertyNode("parent").addConstraintViolation();
50.             }
51.
52.             if (!RealmDAO.NAME_PATTERN.matcher(realm.getName()).matches()) {
53.                 isValid = false;
54.
55.                 context.buildConstraintViolationWithTemplate(
56.                         getTemplate(EntityViolationType.InvalidRealm, "Only alphanumeric chars allowed in realm name")).
57.                         addPropertyNode("name").addConstraintViolation();
58.             }
59.         }
60.
61.         return isValid;
62.     }
63. }
```

Figura 31: Coverage per RealmValidator.isValid()

```
27  public class RealmValidator extends AbstractValidator<RealmCheck, Realm> {
28
29      @Override
30      public boolean isValid(final Realm realm, final ConstraintValidatorContext context) {
31 1        context.disableDefaultConstraintViolation();
32
33          boolean isValid = true;
34
35 1        if (SyncopeConstants.ROOT_REALM.equals(realm.getName())) {
36 1            if (realm.getParent() != null) {
37                  isValid = false;
38
39                  context.buildConstraintViolationWithTemplate(
40                          getTemplate(EntityViolationType.InvalidRealm, "Root realm cannot have a parent realm")).
41                          addPropertyNode("parent").addConstraintViolation();
42              }
43          } else {
44 1            if (realm.getParent() == null) {
45                  isValid = false;
46
47                  context.buildConstraintViolationWithTemplate(
48                          getTemplate(EntityViolationType.InvalidRealm, "A realm needs to reference a parent realm")).
49                          addPropertyNode("parent").addConstraintViolation();
50              }
51
52 1            if (!RealmDAO.NAME_PATTERN.matcher(realm.getName()).matches()) {
53                  isValid = false;
54
55                  context.buildConstraintViolationWithTemplate(
56                          getTemplate(EntityViolationType.InvalidRealm, "Only alphanumeric chars allowed in realm name")).
57                          addPropertyNode("name").addConstraintViolation();
58              }
59          }
60
61 2        return isValid;
62      }
63  }
```

Figura 32: Mutation Testing per RealmValidator.isValid()