

REPORT SOFTWARE TESTING

Projects: BookKeeper, Syncope

De Angelis Alessandro

0317176

Github repositories:

BookKeeper: <https://github.com/AlessDea/bookkeeper>

Syncope: <https://github.com/AlessDea/syncope>

BookKeeper

BookKeeperAdmin.java

La scelta di questa classe è dovuta al fatto che dall'analisi delle classi condotta risulta essere una delle più anziane e più 'buggy'. In particolare, dalla release 1 (4.0.0) alla 7 (4.2.3), è stata coinvolta in fix di bug per ben 5 volte.

```
nukeExistingCluster(ServerConfiguration conf, String  
ledgersRootPath, String instanceId, boolean force)
```

Questo metodo 'distrugge' i metadati di un cluster esistente. I parametri indicano rispettivamente: la configurazione del server Zookeeper, il path della root del ledger, l'id dell'istanza del cluster e un booleano che serve ad indicare se l' `instanceId` debba essere validata o meno.

In particolare, la distruzione dei metadati viene effettuata solamente se `ledgersRootPath` corrisponde con `zkLedgersRootPath` che è mantenuto nella configurazione del server di Zookeeper (`conf`). Inoltre `instanceId` deve corrispondere a quello contenuto nei metadati del cluster ma se il parametro `force` è pari a `true` allora la 'distruzione' dei metadati avviene senza controllare se `instanceId` corrisponde a quello nei metadati del cluster.

Category Partition

Per `ledgersRootPath` e `instanceId` bisogna considerare come invalid dei valori che non esistono nella `conf` e non dei valori che potrebbero non essere validi per quel tipo di dato. Quindi valori diversi da quelli con cui è stato configurato il cluster

- `ServerConfiguration conf: {invalid, valid, null}`

- `invalid`: si può creare un configuration per poi settare un attributo in modo che la configurazione non sia più valida, per esempio settando a `-1` la porta del bookie e settando a null il `metadataServiceUri`
- `String ledgersRootPath : {invalid, valid, null}`
 - `invalid/invalid` rispetto al root path presente nella `conf`
- `String instanceId : {invalid, valid, null}`
 - `invalid/invalid` rispetto all'instance id presente nella `conf`
- `boolean force : {true, false}`

N.B.

Dalla documentazione, un instance id deve essere un oggetto di tipo `immutable universally unique identifier (UUID)` che rappresenta un valore di 128 bit. In particolare è una stringa di 36 caratteri divisa in 5 gruppi separati da trattini, per esempio:

"7940c93b-5da8-4fa7-941e-d254d678fb1c"

Boundary values

Per la configurazione valida si consideri `ledgersRootPath = "/ledgers"`.

L' `instanceId` viene generato automaticamente alla creazione della configurazione, quindi quando si considera un `instanceId` non valido può considerarsi qualsiasi UUID, come indicato sopra, che sia diverso da quello effettivo della configurazione.

```
ServerConfiguration conf: {invalid_c, valid_c, null}
String ledgersRootPath: {"/rootpath", "/ledgers", null}
String instanceId: {invalid_Id, valid_Id, null}
boolean force: {true, false}
```

I parametri che sono legati fra loro sono:

- `conf, ledgersRootPath`: poichè viene fatto il controllo se il path della root del ledger corrisponde con quello nella configurazione
- `instanceId, force`: poichè in caso di `force` valido allora `instanceId` non viene considerato

Per i primi due si può optare per un approccio unidimensionale ed identificare i seguenti casi limite:

- `{invalid_c, "/ledgers"}` -> permette di verificare che venga effettivamente fatto il controllo con il path specificato nella configurazione e che non ci si basi solo da quello fornito con `ledgerRootPath`
- `{valid_c, "/rootpath"}` -> permette di verificare che un path errato non venga preso per buono o non causi errori non gestiti

- {null, "/ledgers"} -> come il primo caso ed inoltre garantisce la buona gestione in caso di configurazione nulla
- {valid_c, null}

Quindi si hanno per `conf` e `ledgersRootPath`: ***{invalid_c, "/ledgers"}, {valid_c, "/rootpath"}, {null, "/ledgers"}, {valid_c, null}***

Per `instanceId` e `force` invece, seguendo sempre un approccio unidimensionale:

- {invalid_id, true} -> permette di verificare che con `force` settato a true, anche con un id non valido, la distruzione viene comunque effettuata
- {valid_id, false} -> verificare la corretta gestione del controllo sull'instance id
- {null, false} -> verificare che un id nullo non causi problemi nel controllo

Ricapitolando si hanno: ***{invalid_id, true}, {valid_id, false}, {null, false}***

A questo punto si può effettuare un approccio multidimensionale fra i due insiemi creati.

```
{expected, conf, ledgersRootPath, instanceId, force}:
{false, INVALID, "/ledgers", VALID, false},
{false, INVALID, "/ledgers", INVALID, true},
{false, INVALID, "/ledgers", NULL, false},
{false, VALID, "/rootpath", VALID, false},
{false, VALID, "/rootpath", INVALID, true},
{false, VALID, "/rootpath", NULL, false},
{false, VALID, null, VALID, false},
{false, VALID, null, INVALID, true},
{false, VALID, null, NULL, false},
{false, NULL, "/ledgers", VALID, false},
{false, NULL, "/ledgers", INVALID, true},
{false, NULL, "/ledgers", NULL, false},
```

Tutti i test hanno esito corretto.

Coverage

Analizzando la coverage con **Jacoco** (Fig. 1 e Fig. 2) si può verificare che i casi di test proposti ottengono una statement coverage del 63% e una branch coverage del 50%. Questo è dovuto al fatto che i casi di test considerati portano il test a fermarsi subito dopo aver verificato la corrispondenza tra la configurazione e il root path (riga 1311) che non ha mai esito positivo in quanto o la configuraione non è valida oppure il rootPath non è quello esatto.

Lo stesso si può vedere anche tramite la **mutation coverage** fatta con **Pit** (Fig. 3): la maggior parte dei mutanti non vengono uccisi proprio per una mancanza di copertura in quei punti dove sono stati introdotti.

Con la **dcf** ottenuta con **ba-dua** (Fig. 4) si ottengono informazioni rilevanti le variabili che

non vengono mai toccate, in particolare `instanceId` ha coverage pari a 0. Questo è in linea ovviamente con quanto già detto e di conseguenza bisogna considerare dei casi in cui il root path corrisponda con quello della configurazione in modo che il resto del metodo possa essere testato.

Si aggiungono quindi i seguenti casi di test e ci si aspetta così che le tre coverage aumentino significativamente:

```
{true, VALID, "/ledgers", VALID, false},  
{false, VALID, "/ledgers", INVALID, true},  
{false, VALID, "/ledgesr", NULL, false},
```

Con l'aggiunta di tali test la statement e branch coverage diventano quelle indicate nelle seguenti figure Fig. 5 e Fig. 6. Si nota quindi che entrambe sono aumentate significativamente.

La stessa cosa può essere detta riguardo la mutation coverage e la data flow, entrambe di molto migliorate (Fig. 7, Fig. 8).

`format(ServerConfiguration conf, boolean isInteractive, boolean force)`

Tale metodo è utilizzato per formattare o inizializzare un insieme di journal e ledger per l'istanza di BookKeeper specificata. Può essere utilizzato per avviare un'istanza di BookKeeper da zero, rimuovendo tutti i dati precedenti e creando una nuova configurazione iniziale. Prende come parametri una configurazione iniziale del server (`conf`), un booleano che indica se la formattazione deve avvenire in modo interattivo (`isInteractive`) e un booleano che indica se bisogna chiedere conferma o meno prima di rimuovere i vecchi dati (`force`). In particolare se `force` è `true`, e lo è anche `isInteractive`, allora non viene chiesto all'utente di confermare, altrimenti è richiesta la conferma.

Category Partition

- `ServerConfiguration conf`: {invalid, valid, null}
- `boolean isInteractive`: {true, false}
- `boolean force`: {true, false}

Boundary values

Per un `ServerConfiguration` invalido: come prima.

```
ServerConfiguration conf: {invalid_c, valid_c, null}  
boolean isInteractive: {true, false}  
boolean force: {true, false}
```

I parametri `isInteractive` e `force` sono strettamente legati in quanto se `isInteractive` è pari a `true` allora è richiesta l'interazione con l'utente ma se anche `force` fosse pari a `true` allora questo implicherebbe che bisognerebbe saltare l'interazione

con l'utente.

Di conseguenza è stato utilizzato un approccio unidimensionale:

- {true, true} -> in questo modo si può verificare se anche se abilitata l'interattività, non venga comunque utilizzata in caso di `force` pari a `true`
- {true, false}
- {false, false}

I casi in cui `force` è uguale a `true` non è stato testato in realtà poichè non si è riusciti a simulare l'interazione con l'utente. Siccome in Mockito è stata introdotta la possibilità di fare un Mock di un metodo statico si è provato a farlo nel seguente modo (seguendo la documentazione):

```
MockedStatic mocked = mockStatic(IOUTils.class)
```

```
mocked.when(() ->
```

```
IOUTils.confirmPrompt(Mockito.anyString())).thenReturn(true);
```

Il mock però non ha provocato alcun effetto, infatti il test si è concluso con un eccezione lanciata dal runner: `org.junit.runners.model.TestTimedOutException: test timed out after 120 seconds`

Viene applicato poi approccio multidimensionale tra i due insiemi per ottenere i casi di test.

```
{expected, conf, isInteractive, force}:
```

```
{true, VALID, false, true},
```

```
{false, VALID, false, false},
```

```
{false, INVALID, false, true},
```

```
{false, INVALID, false, false},
```

Coverage

Con **jacoco** (Fig. 9 , Fig. 10) possiamo verificare che la coverage di tale metodo è pari al 100% per quanto riguarda la statement coverage. Stessa cosa non si può invece dire per la branch coverage ma questo è giustificato dal fatto che i branch che non hanno raggiunto una coverage completa sono quelli che verificato se l'operazione deve essere effettuata in modo interattivo con l'utente e, come detto prima, questo meccanismo non è stato possibile imitarlo.

La mutation coverage non ha portato alcun risultato se non limitato, a quanto pare PIT non segue le funzioni nidificate. (Fig. 11)

Per quanto riguarda l'analisi con ba-dua invece (Fig. 12) si ottiene una dfc di 13 su 19, ovvero che sono stati 19 def-use e ne sono stati saltati 13. Questo, ancora una volta, è causato dal caso non testabile dell'interazione con l'utente.

ReadCache.java

Questa classe è stata scelta perchè molto più recente rispetto a quelle analizzate e quindi potrebbe contenere dei bug.

`get(long ledgerId, long entryId)`

Il metodo `get` di `ReadCache` prende come parametri due `long` che identificano l'id del ledger e l'id dell'entry all'interno del **ledger**. Tale metodo restituisce un oggetto di tipo `ByteBuf` nel caso l'entry identificata da `entryId` esista all'interno del ledger specificato da `ledgerId`. In caso negativo ci si aspetta una certa eccezione o il valore `null`.

Category Partition

- `long ledgerId` : {valid, invalid}
 - valid: id di un ledger esistente
 - invalid: un id diverso da quello del ledger esistente
- `long entryId` : {valid, invalid}
 - valid: id di una entry presente nel ledger
 - invalid: id di una entry non presente nel ledger

Boundary values

Con **valid_lid** si identifica l'id di un ledger mentre con **invalid_lid** ci si riferisce ad un id differente da quello del ledger.

Con **valid_eid** ci si riferisce ad una entry effettivamente presente all'interno del ledger mentre con **invalid_eid** ci si riferisce al caso opposto.

```
long ledgerId : {valid_lid, invalid_lid}
```

```
long entryId : {valid_eid, invalid_eid}
```

Con un approccio multidimensionale si ottengono quindi i seguenti casi di test:

{valid_lid, valid_eid}, {valid_lid, invalid_eid}, {invalid_lid, valid_eid}, {invalid_lid, invalid_eid},

Si deve quindi effettuare una put preventiva sulla cache inserendo una certa sequenza di byte in un ledger (ad esempio quello identificato dall'ID 123) ad una certa entry (ad esempio 0), i casi di test saranno:

```
{ expected, ledgerId, entryId }:
```

```
{true, 123, 0},
```

```
{false, 123, 1},
```

```
{false, 10000, 0},
```

```
{false, 10000, 1}
```

Coverage

Con i test sopra elencati si ottiene una coverage con **Jacoco** dell'100% sia per la statement che per la branch. Ciò permette di dire che la test suite è buona per quanto riguarda la flow coverage. (Fig. 13, Fig. 14)

Per quanto riguarda la mutation coverage (Fig. 15), effettuata con **PIT**, tutti i mutanti sono

stati uccisi tranne uno. Tramite un approccio white-box si è effettuata un'analisi su come venga creata una cache, in particolare si evince che se la `maxCacheSize` è minore di un certo valore allora la cache viene creata con due segmenti che è proprio il caso in questione di come sono stati implementati i test. Le due mutazioni introdotte da PIT che non sono state uccise non fanno altro che restituire l'indice di uno dei due segmenti, infatti `int segmentIdx = (currentSegmentIdx - (size - i)) % size` e `int segmentIdx = (currentSegmentIdx + (size + i)) % size` (con `size = 2`) fanno sì che `segmentIdx` sia sempre, in ordine, prima 0 e poi 1:

- No mutazione: `int segmentIdx = (currentSegmentIdx + (size - i)) % size`
 - `i = 0 -> (1 + (2 - 0)) % 2 = 1`
 - `i = 1 -> (1 + (2 - 1)) % 2 = 0`
- Mutazione 1: `int segmentIdx = (currentSegmentIdx - (size - i)) % size`
 - `i = 0 -> (1 - (2 - 0)) % 2 = 1`
 - `i = 1 -> (1 - (2 - 1)) % 2 = 0`
- Mutazione 2: `int segmentIdx = (currentSegmentIdx + (size + i)) % size`
 - `i = 0 -> (1 + (2 + 0)) % 2 = 1`
 - `i = 1 -> (1 + (2 + 1)) % 2 = 0`

Inoltre, prima di restituire la entry viene controllato che essa non sia nulla. Da questi risultati quindi si evince che le mutazioni apportate da PIT rendono il SUT equivalente alle mutazioni sia per strong che weak mutation. Le mutazioni non possono essere uccise poiché M e il SUT non produrranno mai risultati diversi per ognuno dei casi di test proposti ed inoltre non è possibile creare dei nuovi casi che vadano a smascherare tali mutazioni perché tali mutazioni sono indipendenti dai parametri del metodo.

Per quanto riguarda la **mutation coverage** si ottiene un'ottima copertura quindi questo rafforza la bontà dei test. (Fig. 16)

hasEntry(long ledgerId, long entryId)

Il metodo `hasEntry` di `ReadCache` prende come parametri due `long` che identificano l'id del ledger e l'id dell'entry all'interno del **ledger**. Tale metodo restituisce `true` nel caso l'entry identificata da `entryId` esista all'interno del ledger specificato da `ledgerId`. In caso negativo ritorna `false`.

Category Partition

- `long ledgerId : {valid, invalid}`
 - `valid`: id di un ledger esistente
 - `invalid`: un id diverso da quello del ledger esistente
- `long entryId : {valid, invalid}`
 - `valid`: id di una entry presente nel ledger
 - `invalid`: id di una entry non presente nel ledger

Boundary values

```
long ledgerId : {valid_lid, invalid_lid}
long entryId : {valid_eid, invalid_eid}
```

Con un approccio multidimensionale si ottengono quindi i seguenti casi di test:

{valid_lid, valid_eid}, {valid_lid, invalid_eid}, {invalid_lid, valid_eid}, {invalid_lid, invalid_eid},

```
{expected, ledgerId, entryId}:
{true, 123, 0},
{false, 123, 1},
{false, 10000, 0},
{false, 10000, 1}
cd
```

Coverage

Dai risultati di **Jacoco**, per quanto riguarda la statement coverage e la branch coverage i test producono dei risultati pari al 100% per entrambe (Fig. 17 , Fig. 18).

Per quanto riguarda la **mutation coverage** invece le cose vanno peggio. In particolare 4 mutanti su 11 non sono stati uccisi. Analizzando in dettaglio le mutazioni si può vedere che le mutazioni non uccise a righe 173 e 174 potrebbero derivare tutte dallo stesso motivo del metodo get. Infatti i cambiamenti apportati fanno comunque sì che `segmentIdx` sia un valore valido per effettuare la lettura in cache.

La mutazione a riga 178 invece indica che è stato modificato il ritorno da `true` a `true` il che fa sì che ovviamente la mutazione non possa essere trovata, questo è evidentemente un bug di pit (<https://github.com/hcoles/pitest/issues/497>)

Per quanto riguarda la **data flow coverage** questa risulta essere buona, con un solo def-use mancato (Fig. 19).

Syncope

RealmUtils.java

Questa classe è relativamente nuova (fino alla venticinquesima release, ultima tra quelle su cui è stata calcolata la buggyness, non è presente), quindi potrebbe contenere dei bug.

```
normalizingAddTo(final Set<String> realms, final String
newRealm)
```

Tale metodo si occupa dell'aggiunta di un nuovo Realm. Prende come parametro il set dei Realm già esistenti ed il nome del nuovo.

Dalla documentazione si può evincere che un real è una sorta di albero che definisce un dominio di sicurezza ed è rappresentato come un path:

Realms define a hierarchical security domain tree, primarily meant for containing Users, Groups and Any Objects.

Each realm:

1. has a unique name and a parent realm except for the pre-defined root realm, which is named ``/``;
2. is either a leaf or root of a sub-tree of realms;
3. is uniquely identified by the path from the root realm, e.g. ``/a/b/c`` identifies the sub-realm ``c`` in the sub-tree rooted at ``b``, having in turn ``a`` as parent realm, directly under the root realm;

ref. <https://syncope.apache.org/docs/reference-guide.html#realms>

Category Partition

- `final Set<String> realms: {fullSet, emptySet, null}`
- `final String newRealm: {notEmptyString, emptyString, null}`

Boundary values

```
final Set<String> realms: {["/a/b", "/c", "/d/e"], [], null}
final String newRealm: {"/a/c", "", null}
```

Con un approccio multidimensionale si ottengono i seguenti casi di test:

```
exp set newRealm
{true, ["/a/b", "/c", "/d/e"], "/a/c"}
{false, ["/a/b", "/c", "/d/e"], ""}
{false, ["/a/b", "/c", "/d/e"], null}
{true, [], "/a/c"}
{false, [], ""}
{false, [], null}
{false, null, "/a/c"}
{false, null, ""}
{false, null, null}
```

POSSIBILE BUG 1 (P.Bug1)

Tutti i test, tranne quelli evidenziati, producono un risultato corretto. I test che falliscono sono quelli in cui il `newRealm` è una stringa vuota, in particolare tali test provocano l'eliminazione di tutto il set di realm e l'inserimento proprio di tale realm 'vuoto'. Questo è dovuto al fatto che nell'implementazione del metodo viene utilizzato il metodo `startsWith(String prefix)` che restituisce `true` in caso l'argomento sia una stringa vuota. Di conseguenza, restituendo `true` per ogni realm presente nel test, questi vengono tutti eliminati, provocando quindi lo svuotamento del set.

POSSIBILE BUG 2 (P.Bug2)

Un altro problema è stato riscontrato nel caso di test in cui il nuovo realm da inserire è null. In particolare, se il set di realm è vuoto e si prova ad inserire un nuovo realm `null` allora l'inserimento viene effettuato (non ci si aspetta questo) mentre se si prova ad inserire un oggetto `null` in un set non vuoto allora l'operazione non viene effettuata poichè viene correttamente lanciata un'eccezione da `startsWith(String prefix)` poichè l'oggetto su cui viene chiamata è `null`. Nell'altro caso invece non viene lanciata eccezione poichè, essendo la lista vuota, non si entra nel ciclo e quindi la chiamata a `startsWith(String prefix)` non viene effettuata ma si procede direttamente con l'inserimento.

A seguito di tale incongruenze ho dovuto procedere con un approccio white box e verificare l'effettiva implementazione, dalla quale si evince che ogni volta che il nuovo realm è presente nel set allora non viene aggiunto. Se nel set c'è un realm il cui path inizia con lo stesso nome del nuovo realm allora questo viene rimosso dal set. Se nessuno dei due casi è vero allora il nuovo realm viene aggiunto nel set. Esempi:

1. immaginiamo che il set di realm esistenti sia `[/a/b, /c, /d/e]` e che il nuovo realm sia `/a`
 - allora in questo caso il realm `/a/b` viene eliminato e `/a` non viene aggiunto
2. in questo caso sia `[/c, /d/e, /f]` il set di realm esistenti e `/a` il nuovo realm
 - in tal caso il nuovo realm viene aggiunto
3. nel caso in cui il set fosse `[/a, /c, /d/e]` e si volesse ancora aggiungere `/a/b` allora:
 - in questo caso il nuovo realm non viene aggiunto poichè ne esiste già uno con path che inizia in modo uguale
4. Se il set fosse `[/a/b, /c, /d/e]` e si volesse aggiungere `/a/c`:
 - in questo caso `/a/c` viene aggiunto

Da questi esempi si evince quindi che vengono aggiunti nuovi realm solo se non esistono, nel set, dei realm di livello superiore a quello da inserire; inoltre due realm identici non faranno parte del set.

`/a` è un realm di livello superiore rispetto a `/a/b`.

Sono stati creati nuovi casi di test per verificare che tali regole vengano rispettate. I nuovi casi di test sono:

```
{true, ["/a/b", "/c", "/d/e"], "/b"},  
{false, ["/a/b", "/c", "/d/e"], "/d/e"},  
{true, ["/a/b", "/c", "/d/e"], "/a"},
```

Sono stati effettuati altri test per verificare la correttezza del metodo ed andare a fondo ai casi precedenti. In particolare è stato implementato un test (non parametrizzato) che invece di verificare il valore di ritorno del metodo verifica i cambiamenti effettuati sul set (in caso ci

fossero stati), in questo modo si è verificato in maniera più puntuale la correttezza del metodo testato e i problemi riscontrati.

Le due classi di test ulteriori sono: `RealmUtilsSecondTest` e `RealmUtilsEmptySetTest`.

Coverage

Con **PIT** si può verificare che la coverage, sia statement sia branch, è del 100% anche prima dell'aggiunta dei nuovi casi di test (`Fig. 21`, `Fig. 22`).

Per quanto riguarda la **mutation coverage** (`Fig. 23`) invece alcuni mutanti non vengono uccisi. Entrambe le mutazioni (righe 49 e 55) non vengono uccise poichè non viene controllato il set dopo la chiamata del metodo ma solo il valore di ritorno.

Dopo l'introduzione dei nuovi test (che controllano il set atteso dopo la chiamata al metodo) tutti i mutanti vengono uccisi (`Fig. 24`).

```
public static Pair<Set<String>, Set<String>>
normalize(final Collection<String> realms)
```

Tale metodo prende una collection di realms e divide il contenuto in due set: realm normali e realm assegnati ad un gruppo. I realm assegnati ad un gruppo sono caratterizzati dalla presenza del carattere '@'.

Category Partition

- `Collection<String> realms: {fullCollection, emptyCollection, null}`

Boundary values

```
Collection<String> realms: {"g1@/a/b", "/a/b", "/c", "g2@/c", "/d/e", "g3@/d/e", null}, [],
null}
```

```
exp realms
{["/a/b", "/c", "/d/e"] ["g1@/a/b", "g2@/c", "g3@/d/e"], "g1@/a/b", "/a/b", "/c", "g2@/c", "/d/e",
"g3@/d/e"]}
{[], []}
{[], null}
```

Tutti e tre i test producono un risultato atteso. Per andare più in dettaglio e cercare di produrre dei casi di test migliori si è analizzato il metodo con un approccio white-box. Da questa analisi è venuto fuori che all'interno di tale metodo, per la creazione del Set contenente i realm 'normali', viene invocato il metodo `normalizeAddTo`. Di conseguenza anche `normalize` dovrebbe soffrire degli stessi problemi emersi per `normalizeAddTo` e per verificarlo sono stati testati i seguenti casi:

```
{["/a/b", "/c", "/d/e"] ["g1@/a/b",], ["g1@/a/b", "/a/b", "/c", "/d/e", ""]}
{[] ["g1@/a/b",], ["g1@/a/b", "g2@/c", "g3@/d/e", ""]}
```

```
{[], ["g1@/a/b",], ["g1@/a/b", "g2@/c", "g3@/d/e", null]}
```

Tali casi hanno effettivamente provato, fallendo, che a causa di `normalizeAddTo` anche tale metodo soffre degli stessi problemi tranne per l'ultimo caso, in cui si prova ad inserire una stringa nulla. Quest'ultimo non provoca i problemi discussi in precedenza poichè viene lanciata l'eccezione `NullPointerException` da `realm.indexOf('@')`.

Coverage

Per tale metodo la statement coverage e la branch coverage, calcolate tramite jacoco, risultano entrambe del 100% (Fig. 25 , Fig. 26). Anche la mutation coverage risulta essere del 100% (Fig. 27).

I casi di test sviluppati quindi sembrano essere buoni.

Encryptor.java

Su tale classe non è stato mai effettuato un fix di un bug e non è stata etichettata come buggy dall'analisi della buggyness. Tale classe però ha subito delle modifiche in ognuna delle 25 release analizzate quindi potrebbe ospitare dei bug ancora dormienti.

```
verify(final String value, final CipherAlgorithm  
cipherAlgorithm, final String encoded)
```

Questo metodo prende una stringa, un algoritmo di cifratura ed una stringa che rappresenta la password cifrata. La stringa `value` viene cifrata tramite `cipherAlgorithm` e confrontata poi con la stringa `encoded`. Restituisce un booleano.

In Syncope, con `CipherAlgorithm`, si identificano indistintamente sia algoritmi di hashing che ciphers. Per semplicità e facilità di comprensione del report verrà utilizzata la parola cipher tutti.

I cipher che possono essere utilizzati sono:

- producono digest di lunghezza fissa:
 - SHA/SHA1: 20 byte
 - SHA256: 32 byte
 - SHA512 : 64 byte
 - MD5: 16 byte
 - BCrypt: 23 byte
 - salted: SSHA/SSHA1, SSHA256, SSHA512 (uguali ai base)
- Cifrario a blocchi quindi lavora con blocchi da 16 byte, può cifrare qualsiasi lunghezza:
 - AES

Per la classe `Encryptor` la documentazione non riporta molte informazioni, di conseguenza tutte le assunzioni iniziali fatte derivano dall'analisi della segnatura del metodo.

Category Partition

- `String value`: {notEmptyString, emptyString, null}
- `CipherAlgorithm cipherAlgorithm`: {each element of the enum, null}
- `String encoded`: {notEmptyString, emptyString, null}

Boundary values

`String value`: {"password123", "", null}

`CipherAlgorithm cipherAlgorithm`: {SHA, SHA1, SHA256, SHA512, AES, SMD5, SSHA, SSHA1, SSHA256, SSHA512, BCrypt, null}

`String encoded`: {encode("password") for each cipher, "", null}

Si utilizza un approccio multidimensionale.

Durante l'esecuzione di alcuni di questi vengono lanciate delle eccezioni, in particolare riguardanti l'ApplicationContext che viene restituito null quando syncopce cerca di fare il retrieve delle security properties, per questo si è deciso di fare il Mock di tale invocazione.

Altri casi di test che falliscono sono quelli in cui il cipher utilizzato è SHA oppure SSHA ma questi due non sono altro che SHA1 e SSHA1, quindi il loro utilizzo dovrebbe portare lo stesso risultato degli ultimi.

Inoltre i test in cui il cipher è null non hanno esito positivo. Per trovare risposta a questo problema si è effettuato un approccio white-box con il quale si è scoperto che quando il cipher passato è null, viene utilizzato di default AES.

Coverage

Tramite Jacoco si può vedere che la coverage, sia statement che branch, è del 100% (Fig. 28, Fig. 29).

Per quanto riguarda la mutation coverage effettuata utilizzando PIT, tutti i mutanti sono stati uccisi (Fig. 30).

Da ciò si può asserire che i casi di test sono buoni.

`encode(final String value, final CipherAlgorithm cipherAlgorithm)`

Tale metodo prende come parametro una stringa ed un algoritmo di cifratura. Tramite l'algoritmo la stringa (password) viene cifrata e restituita.

Category Partition

- `String value`: {notEmptyString, emptyString, null}
- `CipherAlgorithm cipherAlgorithm`: {each element of the enum, null}

Boundary values

```
String value: {"password123", "", null}
CipherAlgorithm cipherAlgorithm: {SHA, SHA1, SHA256, SHA512, AES, SMD5, SSHA, SSHA1, SSHA256, SSHA512, BCrypt, null}
```

I test case vengono ottenuti tramite un approccio multidimensionale. Per i casi in cui viene utilizzato un cipher salted bisogna usare il metodo `Encryptor.verify()` per verificare che l'encode abbia avuto successo poichè i risultati di due encode su uno stesso valore produrranno sempre due risultati differenti a causa del sale.

A questo punto però è utile verificare se tutti i Cipher producano un risultato corretto, di conseguenza devo fare un test per ogni cipher quando questo è valido.

Coverage

Tramite l'analisi con Jacoco si può vedere che la statement coverage e la branch coverage sono entrambe del 100% (Fig. 31, Fig. 32).

Anche la mutation coverage è del 100% in quanto tutti i mutanti sono stati uccisi (Fig. 33). Si può concludere quindi che i casi di test sono buoni.

Integration Test

AuthDataAccessor

Il test di integrazione è stato fatto sulla classe `AuthDataAccessor` ed in particolare sul metodo `getDelegatedAuthorities(final Delegation delegation)`. Questa scelta è dovuta al fatto che tale metodo invoca `protected Set<SyncopeGrantedAuthority> buildAuthorities(final Map<String, Set<String>> entForRealms)` il quale a sua volta invoca più volte il metodo `normalize` della classe `RealmUtils`.

Il metodo `GetDelegatedAuthorities` prende come parametro un oggetto di tipo `Delegation` che rappresenta una delega di autorità o responsabilità da un utente a un altro utente. In particolare in un oggetto `Delegation` vengono settati una serie di `Role` (ruoli) che si vogliono delegare.

Tale metodo restituisce un set di `SyncopeGrantedAuthority` che contiene le informazioni sull'autorità concessa, come ad esempio il nome dell'autorità o del ruolo. In particolare rappresenta un `Entitlement` con i rispettivi `Realm` associati.

Un `Role` è un oggetto composto principalmente da `Entitlements` e `Realms` tra i quali c'è la seguente relazione: ad un entitlement possono essere associati più realm. In particolare un `Entitlement` è una stringa sostanzialmente che rappresenta un 'diritto' di poter effettuare una certa operazione sul sistema. Alcuni tipo di entitlements sono:

```
REALM_CREATE, REALM_DELETE, ecc..
```

Category Partition

- `Delegation delegation: {valid, invalid, null}`
 - `valid`: un `Delegation` in cui ci sono effettivamente settati dei ruoli da delegare
 - `invalid`: un `Delegation` in cui non ci sono ruoli delegati

Boundary values

`Delegation delegation: {withRoles, emptyRoles, null}`

Tutti i casi di test sono stati pensati in modo da far emergere i problemi rilevati per i metodi `normalize` e `normalizeAddTo`.

`withRoles`: sono stati creati differenti casi:

```
| delegation |
| expected entitlement realms |
| {"/a", "/b", "/d"}, ENT1, {"/a/b", "/a", "/b", "/a/b/c", "/d"} |
| {"/a/b"}, ENT2, {"/a/b", "/a/b/c"} |
| {"/a/b", "/c/d"}, ENT3, {"/a/b", "/c/d", ""} |
| {"/a/b", "/c/d"}, ENT4, {"/a/b", "/c/d", null} |
| {null, ENT5, {} |
| {}, NULL, {} |
```

Dall'analisi dei risultati di tali casi di test si evince che i problemi rilevati per i metodi `normalize` e `normalizeAddTo` si presentano anche in tale scenario. In particolare il caso della stringa vuota che produce l'eliminazione di tutti gli elementi del set dei realm. Anche qui, con nel caso di `normalize` l'inserimento di un realm null provoca l'eccezione `NullPointerException` quindi non si verifica il problema discusso per `normalizeAddTo`.

Coverage

Jacoco non ha prodotto un report sul test d'integrazione, quindi non è stato possibile eseguire un'analisi sulle statement e branch coverage.

Tramite l'utilizzo di pit però è stato possibile eseguit la mutation coverage. Dal report ottenuto (Fig. 34, Fig. 35) si può vedere che due delle tre mutazioni non uccise riguardano la rimozione del `forEach` ma tale rimozione comporta, fondamentalmente, l'eliminazione da riga 404 a riga 415 e questo provoca che la chiamata a `buildAuthorities` venga effettuata su un oggetto di tipo `Map<String, Set<String>>` **vuoto**. Di conseguenza è impossibile che l'oggetto ritornato dal metodo in tale scenario sia uguale a quello atteso.

Note

L'applicazione di ba-dua per il progetto Syncope non è stata possibile. Questo è dovuto al fatto che Syncope ha bisogno della jdk-20 per essere compilato ma ba-dua necessita della jdk-8.


```

1308. public static boolean nukeExistingCluster(ServerConfiguration conf, String ledgersRootPath, String instanceId
1309. boolean force) throws Exception {
1310.     String confLedgersRootPath = ZKMetadataDriverBase.resolveZkLedgersRootPath(conf);
1311.     if (!confLedgersRootPath.equals(ledgersRootPath)) {
1312.         LOG.error("Provided ledgerRootPath : {} is not matching with config's ledgerRootPath: {}, "
1313.             + "so exiting nuke operation", ledgersRootPath, confLedgersRootPath);
1314.         return false;
1315.     }
1316.
1317.     return runFunctionWithRegistrationManager(conf, rm -> {
1318.         try {
1319.             if (!force) {
1320.                 String readInstanceId = rm.getClusterInstanceId();
1321.                 if ((instanceId == null) || !instanceId.equals(readInstanceId)) {
1322.                     LOG.error("Provided InstanceId : {} is not matching with cluster InstanceId in ZK: {}",
1323.                         instanceId, readInstanceId);
1324.                     return false;
1325.                 }
1326.             }
1327.             return rm.nukeExistingCluster();
1328.         } catch (Exception e) {
1329.             throw new UncheckedExecutionException(e.getMessage(), e);
1330.         }
1331.     });
1332. }

```

Fig. 1

• nukeExistingCluster(ServerConfiguration, String, String, boolean)	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	63%	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	50%
• lambda\$nukeExistingCluster\$4(boolean, String, RegistrationManager)	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	32%	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	33%

Fig. 2

```

1308 public static boolean nukeExistingCluster(ServerConfiguration conf, String ledgersRootPath, String instanceId,
1309 boolean force) throws Exception {
1310     String confLedgersRootPath = ZKMetadataDriverBase.resolveZkLedgersRootPath(conf);
1311     if (!confLedgersRootPath.equals(ledgersRootPath)) {
1312         LOG.error("Provided ledgerRootPath : {} is not matching with config's ledgerRootPath: {}, "
1313             + "so exiting nuke operation", ledgersRootPath, confLedgersRootPath);
1314     }
1315     return false;
1316 }
1317 return runFunctionWithRegistrationManager(conf, rm -> {
1318     try {
1319         if (!force) {
1320             String readInstanceId = rm.getClusterInstanceId();
1321             if ((instanceId == null) || !instanceId.equals(readInstanceId)) {
1322                 LOG.error("Provided InstanceId : {} is not matching with cluster InstanceId in ZK: {}",
1323                     instanceId, readInstanceId);
1324             }
1325             return false;
1326         }
1327         return rm.nukeExistingCluster();
1328     } catch (Exception e) {
1329         throw new UncheckedExecutionException(e.getMessage(), e);
1330     }
1331 });
1332 }
1333 }

```

Fig. 3

```

<method name="lambda$nukeExistingCluster$4" desc="(Ljava/lang/String;Lorg/apache/bookkeeper/discover/RegistrationManager;)Ljava
/lang/Boolean;">
  <du var="force" def="1319" use="1319" target="1320" covered="1"/>
  <du var="force" def="1319" use="1319" target="1327" covered="1"/>
  <du var="instanceId" def="1319" use="1321" target="1321" covered="0"/>
  <du var="instanceId" def="1319" use="1321" target="1322" covered="0"/>
  <du var="instanceId" def="1319" use="1322" covered="0"/>
  <du var="instanceId" def="1319" use="1321" target="1322" covered="0"/>
  <du var="instanceId" def="1319" use="1321" target="1327" covered="0"/>
  <du var="rm" def="1319" use="1327" covered="1"/>
  <du var="rm" def="1319" use="1320" covered="1"/>
  <du var="LOG" def="1319" use="1322" covered="0"/>
  <du var="readInstanceId" def="1320" use="1322" covered="0"/>
  <du var="readInstanceId" def="1320" use="1321" target="1322" covered="0"/>
  <du var="readInstanceId" def="1320" use="1321" target="1327" covered="0"/>
  <counter type="DU" missed="9" covered="4"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Fig. 4

```

1308. public static boolean nukeExistingCluster(ServerConfiguration conf, String ledgersRootPath, String instanceId,
1309. boolean force) throws Exception {
1310.     String confLedgersRootPath = ZKMetadataDriverBase.resolveZkLedgersRootPath(conf);
1311.     if (!confLedgersRootPath.equals(ledgersRootPath)) {
1312.         LOG.error("Provided ledgerRootPath : {} is not matching with config's ledgerRootPath: {}, "
1313.             + "so exiting nuke operation", ledgersRootPath, confLedgersRootPath);
1314.         return false;
1315.     }
1316.
1317.     return runFunctionWithRegistrationManager(conf, rm -> {
1318.         try {
1319.             if (!force) {
1320.                 String readInstanceId = rm.getClusterInstanceId();
1321.                 if ((instanceId == null) || !instanceId.equals(readInstanceId)) {
1322.                     LOG.error("Provided InstanceId : {} is not matching with cluster InstanceId in ZK: {}",
1323.                         instanceId, readInstanceId);
1324.                     return false;
1325.                 }
1326.             }
1327.             return rm.nukeExistingCluster();
1328.         } catch (Exception e) {
1329.             throw new UncheckedExecutionException(e.getMessage(), e);
1330.         }
1331.     });
1332. }

```

Fig. 5

• <code>lambda\$nukeExistingCluster\$4(boolean, String, RegistrationManager)</code>	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	74%	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	83%
• <code>nukeExistingCluster(ServerConfiguration, String, String, boolean)</code>	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	100%	<div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div><div></div></div>	100%

Fig. 6

```

1308     public static boolean nukeExistingCluster(ServerConfiguration conf, String ledgersRootPath, String instanceId,
1309         boolean force) throws Exception {
1310         String confLedgersRootPath = ZKMetadataDriverBase.resolveZkLedgersRootPath(conf);
1311         if (!confLedgersRootPath.equals(ledgersRootPath)) {
1312             LOG.error("Provided ledgerRootPath : {} is not matching with config's ledgerRootPath: {}, "
1313                 + "so exiting nuke operation", ledgersRootPath, confLedgersRootPath);
1314             return false;
1315         }
1316
1317         return runFunctionWithRegistrationManager(conf, rm -> {
1318             try {
1319                 if (!force) {
1320                     String readInstanceId = rm.getClusterInstanceId();
1321                     if ((instanceId == null) || !instanceId.equals(readInstanceId)) {
1322                         LOG.error("Provided InstanceId : {} is not matching with cluster InstanceId in ZK: {}",
1323                             instanceId, readInstanceId);
1324                         return false;
1325                     }
1326                 }
1327                 return rm.nukeExistingCluster();
1328             } catch (Exception e) {
1329                 throw new UncheckedExecutionException(e.getMessage(), e);
1330             }
1331         });
1332     }

```

Fig. 7

```

<method name="lambda$nukeExistingCluster$4" desc="(ZLjava/lang/String;Lorg/apache
/lang/Boolean;)">
  <du var="force" def="1319" use="1319" target="1320" covered="1"/>
  <du var="force" def="1319" use="1319" target="1327" covered="1"/>
  <du var="instanceId" def="1319" use="1321" target="1321" covered="1"/>
  <du var="instanceId" def="1319" use="1321" target="1322" covered="1"/>
  <du var="instanceId" def="1319" use="1322" covered="1"/>
  <du var="instanceId" def="1319" use="1321" target="1322" covered="0"/>
  <du var="instanceId" def="1319" use="1321" target="1327" covered="1"/>
  <du var="rm" def="1319" use="1327" covered="1"/>
  <du var="rm" def="1319" use="1320" covered="1"/>
  <du var="LOG" def="1319" use="1322" covered="1"/>
  <du var="readInstanceId" def="1320" use="1322" covered="1"/>
  <du var="readInstanceId" def="1320" use="1321" target="1322" covered="0"/>
  <du var="readInstanceId" def="1320" use="1321" target="1327" covered="1"/>
  <counter type="DU" missed="2" covered="11"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Fig. 8

```

1238. public static boolean format(ServerConfiguration conf,
1239. boolean isInteractive, boolean force) throws Exception {
1240. return runFunctionWithMetadataBookieDriver(conf, new Function<MetadataBookieDriver, Boolean>() {
1241. @Override
1242. @SuppressWarnings("RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE")
1243. public Boolean apply(MetadataBookieDriver driver) {
1244. try {
1245. try (RegistrationManager regManager = driver.createRegistrationManager()) {
1246. boolean ledgerRootExists = regManager.prepareFormat();
1247. // If old data was there then confirm with admin.
1248. boolean doFormat = true;
1249. if (ledgerRootExists) {
1250. if (!isInteractive) {
1251. // If non interactive and force is set, then delete old data.
1252. doFormat = force;
1253. } else {
1254. // Confirm with the admin.
1255. doFormat = IOUtils
1256. .confirmPrompt("Ledger root already exists. "
1257. + "Are you sure to format bookkeeper metadata? "
1258. + "This may cause data loss.");
1259. }
1260. }
1261. if (!doFormat) {
1262. return false;
1263. }
1264. driver.getLedgerManagerFactory().format(
1265. conf,
1266. driver.getLayoutManager());
1267. return regManager.format();
1268. }
1269. } catch (Exception e) {
1270. System.out.println("EXC:" + Arrays.toString(e.getStackTrace()));
1271. throw new UncheckedExecutionException(e.getMessage(), e);
1272. }
1273. }
1274. });
1275. }

```

Fig. 9

format(ServerConfiguration, boolean, boolean)



100%

Fig. 10

```

1238     public static boolean format(ServerConfiguration conf,
1239                                boolean isInteractive, boolean force) throws Exception {
1240         return runFunctionWithMetadataBookieDriver(conf, new Function<MetadataBookieDriver, Boolean>() {
1241             @Override
1242             @SuppressWarnings("RCN_REDUNDANT_NULLCHECK_WOULD_HAVE_BEEN_A_NPE")
1243             public Boolean apply(MetadataBookieDriver driver) {
1244                 try {
1245                     try (RegistrationManager regManager = driver.createRegistrationManager()) {
1246                         boolean ledgerRootExists = regManager.prepareFormat();
1247                         // If old data was there then confirm with admin.
1248                         boolean doFormat = true;
1249                         if (ledgerRootExists) {
1250                             if (!isInteractive) {
1251                                 // If non interactive and force is set, then delete old data.
1252                                 doFormat = force;
1253                             } else {
1254                                 // Confirm with the admin.
1255                                 doFormat = IOUtils
1256                                     .confirmPrompt("Ledger root already exists. "
1257                                         + "Are you sure to format bookkeeper metadata? "
1258                                         + "This may cause data loss.");
1259                             }
1260                         }
1261                         if (!doFormat) {
1262                             return false;
1263                         }
1264                         driver.getLedgerManagerFactory().format(
1265                             conf,
1266                             driver.getLayoutManager());
1267                         return regManager.format();
1268                     }
1269                 } catch (Exception e) {
1270                     System.out.println("EXC:" + Arrays.toString(e.getStackTrace()));
1271                     throw new UncheckedExecutionException(e.getMessage(), e);
1272                 }
1273             }
1274         });
1275     }
1276 }

```

Fig. 11

```

<class name="org/apache/bookkeeper/client/BookKeeperAdmin$7">
  <method name="apply" desc="(Lorg/apache/bookkeeper/meta/MetadataBookieDriver;)Ljava/lang/Boolean;">
    <du var="this" def="1245" use="1264" covered="1"/>
    <du var="this" def="1245" use="1250" target="1252" covered="1"/>
    <du var="this" def="1245" use="1250" target="1255" covered="0"/>
    <du var="this" def="1245" use="1252" covered="1"/>
    <du var="driver" def="1245" use="1264" covered="1"/>
    <du var="this.val$isInteractive" def="1245" use="1250" target="1252" covered="1"/>
    <du var="this.val$isInteractive" def="1245" use="1250" target="1255" covered="0"/>
    <du var="this.val$force" def="1245" use="1252" covered="1"/>
    <du var="this.val$conf" def="1245" use="1264" covered="1"/>
    <du var="regManager" def="1245" use="1267" covered="1"/>
    <du var="regManager" def="1245" use="1268" target="1268" covered="1"/>
    <du var="regManager" def="1245" use="1268" target="1267" covered="0"/>
    <du var="regManager" def="1245" use="1268" covered="1"/>
    <du var="regManager" def="1245" use="1268" covered="0"/>
    <du var="regManager" def="1245" use="1268" target="1268" covered="1"/>
    <du var="regManager" def="1245" use="1268" target="1262" covered="0"/>
    <du var="regManager" def="1245" use="1268" covered="1"/>
    <du var="regManager" def="1245" use="1268" covered="0"/>
    <du var="ledgerRootExists" def="1246" use="1249" target="1250" covered="1"/>
    <du var="ledgerRootExists" def="1246" use="1249" target="1261" covered="0"/>
    <du var="doFormat" def="1248" use="1261" target="1262" covered="0"/>
    <du var="doFormat" def="1248" use="1261" target="1264" covered="0"/>
    <du var="doFormat" def="1252" use="1261" target="1262" covered="1"/>
    <du var="doFormat" def="1252" use="1261" target="1264" covered="1"/>
    <du var="doFormat" def="1256" use="1261" target="1262" covered="0"/>
    <du var="doFormat" def="1256" use="1261" target="1264" covered="0"/>
    <counter type="DU" missed="13" covered="19"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>

```

Fig. 12

```

139.     public ByteBuf get(long ledgerId, long entryId) {
140.         lock.readLock().lock();
141.
142.         try {
143.             // We need to check all the segments, starting from the current one and looking
144.             // backward to minimize the
145.             // checks for recently inserted entries
146.             int size = cacheSegments.size();
147.             for (int i = 0; i < size; i++) {
148.                 int segmentIdx = (currentSegmentIdx + (size - i)) % size;
149.
150.                 LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
151.                 if (res != null) {
152.                     int entryOffset = (int) res.first;
153.                     int entryLen = (int) res.second;
154.
155.                     ByteBuf entry = allocator.buffer(entryLen, entryLen);
156.                     entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
157.                     return entry;
158.                 }
159.             }
160.         } finally {
161.             lock.readLock().unlock();
162.         }
163.
164.         // Entry not found in any segment
165.         return null;
166.     }

```

Fig. 13

● <u>get(long, long)</u>	<div><div></div></div>	100%	<div><div></div></div>	100%
--------------------------	------------------------	------	------------------------	------

Fig. 14

```

139     public ByteBuf get(long ledgerId, long entryId) {
140 1     lock.readLock().lock();
141
142     try {
143         // We need to check all the segments, starting from the current one and looking
144         // backward to minimize the
145         // checks for recently inserted entries
146         int size = cacheSegments.size();
147 2     for (int i = 0; i < size; i++) {
148 3         int segmentIdx = (currentSegmentIdx + (size - i)) % size;
149
150         LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
151 1     if (res != null) {
152         int entryOffset = (int) res.first;
153         int entryLen = (int) res.second;
154
155         ByteBuf entry = allocator.buffer(entryLen, entryLen);
156         entry.writeBytes(cacheSegments.get(segmentIdx), entryOffset, entryLen);
157 1     return entry;
158     }
159 }
160 } finally {
161 1     lock.readLock().unlock();
162 }
163
164     // Entry not found in any segment
165     return null;
166 }

```

1. Replaced integer addition with subtraction → SURVIVED
- 148 2. Replaced integer modulus with multiplication → KILLED
3. Replaced integer subtraction with addition → SURVIVED

Fig. 15

```

<method name="get" desc="(J)Lio/netty/buffer/ByteBuf;">
  <du var="this" def="140" use="161" covered="1"/>
  <du var="this" def="140" use="148" covered="1"/>
  <du var="this" def="140" use="150" covered="1"/>
  <du var="this" def="140" use="155" covered="1"/>
  <du var="this" def="140" use="156" covered="1"/>
  <du var="this" def="140" use="161" covered="1"/>
  <du var="ledgerId" def="140" use="150" covered="1"/>
  <du var="entryId" def="140" use="150" covered="1"/>
  <du var="this.lock" def="140" use="161" covered="1"/>
  <du var="this.lock" def="140" use="161" covered="1"/>
  <du var="this.cacheSegments" def="140" use="156" covered="1"/>
  <du var="this.currentSegmentIdx" def="140" use="148" covered="1"/>
  <du var="this.cacheIndexes" def="140" use="150" covered="1"/>
  <du var="this allocator" def="140" use="155" covered="1"/>
  <du var="size" def="146" use="147" target="148" covered="1"/>
  <du var="size" def="146" use="147" target="161" covered="1"/>
  <du var="size" def="146" use="148" covered="1"/>
  <du var="i" def="147" use="147" target="148" covered="1"/>
  <du var="i" def="147" use="147" target="161" covered="0"/>
  <du var="i" def="147" use="148" covered="1"/>
  <du var="i" def="147" use="147" covered="1"/>
  <du var="segmentIdx" def="148" use="156" covered="1"/>
  <du var="res" def="150" use="151" target="152" covered="1"/>
  <du var="res" def="150" use="151" target="147" covered="1"/>
  <du var="res" def="150" use="152" covered="1"/>
  <du var="res" def="150" use="153" covered="1"/>
  <du var="res.first" def="150" use="152" covered="1"/>
  <du var="res.second" def="150" use="153" covered="1"/>
  <du var="i" def="147" use="147" target="148" covered="1"/>
  <du var="i" def="147" use="147" target="161" covered="1"/>
  <du var="i" def="147" use="148" covered="1"/>
  <du var="i" def="147" use="147" covered="1"/>
  <counter type="DU" missed="1" covered="31"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Fig. 16

```

168.     public boolean hasEntry(long ledgerId, long entryId) {
169.         lock.readLock().lock();
170.
171.         try {
172.             int size = cacheSegments.size();
173.             for (int i = 0; i < size; i++) {
174.                 int segmentIdx = (currentSegmentIdx + (size - i)) % size;
175.
176.                 LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
177.                 if (res != null) {
178.                     return true;
179.                 }
180.             }
181.         } finally {
182.             lock.readLock().unlock();
183.         }
184.
185.         // Entry not found in any segment
186.         return false;
187.     }

```

Fig. 17

hasEntry(long, long)	<div><div></div></div>	100%	<div><div></div></div>	100%
----------------------	------------------------	------	------------------------	------

Fig. 18


```

168     public boolean hasEntry(long ledgerId, long entryId) {
169 1. lock.readLock().lock();
170
171     try {
172         int size = cacheSegments.size();
173 2. for (int i = 0; i < size; i++) {
174 3.         int segmentIdx = (currentSegmentIdx + (size - i)) % size;
175
176         LongPair res = cacheIndexes.get(segmentIdx).get(ledgerId, entryId);
177 1.         if (res != null) {
178 2.             return true;
179         }
180     }
181     } finally {
182 1.         lock.readLock().unlock();
183     }
184
185     // Entry not found in any segment
186 1.     return false;
187 }
188
173 1. negated conditional → KILLED
173 2. changed conditional boundary → SURVIVED
174 1. Replaced integer addition with subtraction → KILLED
174 2. Replaced integer subtraction with addition → SURVIVED
174 3. Replaced integer modulus with multiplication → KILLED
177 1. negated conditional → KILLED
178 1. replaced boolean return with false for org/apache/bookkeeper/bookie/storage/ldb/ReadCache::hasEntry → KILLED
178 2. replaced boolean return with true for org/apache/bookkeeper/bookie/storage/ldb/ReadCache::hasEntry → SURVIVED
182 1. removed call to java/util/concurrent/locks/ReentrantReadWriteLock$ReadLock::unlock → SURVIVED
186 1. replaced boolean return with true for org/apache/bookkeeper/bookie/storage/ldb/ReadCache::hasEntry → KILLED

```

Fig. 19

```

<method name="hasEntry" desc="(J)Z">
  <du var="this" def="169" use="182" covered="1"/>
  <du var="this" def="169" use="174" covered="1"/>
  <du var="this" def="169" use="176" covered="1"/>
  <du var="this" def="169" use="182" covered="1"/>
  <du var="ledgerId" def="169" use="176" covered="1"/>
  <du var="entryId" def="169" use="176" covered="1"/>
  <du var="this.lock" def="169" use="182" covered="1"/>
  <du var="this.lock" def="169" use="182" covered="1"/>
  <du var="this.currentSegmentIdx" def="169" use="174" covered="1"/>
  <du var="this.cacheIndexes" def="169" use="176" covered="1"/>
  <du var="size" def="172" use="173" target="174" covered="1"/>
  <du var="size" def="172" use="173" target="182" covered="1"/>
  <du var="size" def="172" use="174" covered="1"/>
  <du var="i" def="173" use="173" target="174" covered="1"/>
  <du var="i" def="173" use="173" target="182" covered="0"/>
  <du var="i" def="173" use="174" covered="1"/>
  <du var="i" def="173" use="173" covered="1"/>
  <du var="res" def="176" use="177" target="178" covered="1"/>
  <du var="res" def="176" use="177" target="173" covered="1"/>
  <du var="i" def="173" use="173" target="174" covered="1"/>
  <du var="i" def="173" use="173" target="182" covered="1"/>
  <du var="i" def="173" use="174" covered="1"/>
  <du var="i" def="173" use="173" covered="1"/>
  <counter type="DU" missed="1" covered="22"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Fig. 20

```

43.     public static boolean normalizingAddTo(final Set<String> realms, final String newRealm)
44.     {
45.         boolean dontAdd = false;
46.         Set<String> toRemove = new HashSet<>();
47.         for (String realm : realms) {
48.             if (newRealm.startsWith(realm)) {
49.                 dontAdd = true;
50.             } else if (realm.startsWith(newRealm)) {
51.                 toRemove.add(realm);
52.             }
53.         }
54.         realms.removeAll(toRemove);
55.         if (!dontAdd) {
56.             realms.add(newRealm);
57.         }
58.         return !dontAdd;
59.     }

```

Fig. 21

● normalizingAddTo(Set, String)	<div><div></div></div>	100%	<div><div></div></div>	100%
---	------------------------	------	------------------------	------

Fig. 22

```

43     public static boolean normalizingAddTo(final Set<String> realms, final String newRealm) {
44         boolean dontAdd = false;
45         Set<String> toRemove = new HashSet<>();
46         for (String realm : realms) {
47             1 if (newRealm.startsWith(realm)) {
48                 dontAdd = true;
49             } 1 else if (realm.startsWith(newRealm)) {
50                 toRemove.add(realm);
51             }
52         }
53
54         realms.removeAll(toRemove);
55         1 if (!dontAdd) {
56             realms.add(newRealm);
57         }
58         2 return !dontAdd;
59     }

```

Fig. 23

```

43     public static boolean normalizingAddTo(final Set<String> realms, final String newRealm) {
44         boolean dontAdd = false;
45         Set<String> toRemove = new HashSet<>();
46         for (String realm : realms) {
47             1 if (newRealm.startsWith(realm)) {
48                 dontAdd = true;
49             } 1 else if (realm.startsWith(newRealm)) {
50                 toRemove.add(realm);
51             }
52         }
53
54         realms.removeAll(toRemove);
55         1 if (!dontAdd) {
56             realms.add(newRealm);
57         }
58         2 return !dontAdd;
59     }
60

```

Fig. 24


```

61.     public static Pair<Set<String>, Set<String>> normalize(final Collection<String> realms) {
62.         Set<String> normalized = new HashSet<>();
63.         Set<String> groupOwnership = new HashSet<>();
64.         if (realms != null) {
65.             realms.forEach(realm -> {
66.                 if (realm.indexOf('@') == -1) {
67.                     normalizingAddTo(normalized, realm);
68.                 } else {
69.                     groupOwnership.add(realm);
70.                 }
71.             });
72.         }
73.
74.         return Pair.of(normalized, groupOwnership);
75.     }
76.

```

Fig. 25

 normalize(Collection)		100%		100%
---	---	------	---	------

Fig. 26

```

61     public static Pair<Set<String>, Set<String>> normalize(final Collection<String> realms) {
62         Set<String> normalized = new HashSet<>();
63         Set<String> groupOwnership = new HashSet<>();
64 1     if (realms != null) {
65 1         realms.forEach(realm -> {
66 1             if (realm.indexOf('@') == -1) {
67                 normalizingAddTo(normalized, realm);
68             } else {
69                 groupOwnership.add(realm);
70             }
71         });
72     }
73
74 1     return Pair.of(normalized, groupOwnership);
75     }
76

```

Fig. 27

```

116.     public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
117.         boolean verified = false;
118.
119.         try {
120.             if (value != null) {
121.                 if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
122.                     verified = encode(value, cipherAlgorithm).equals(encoded);
123.                 } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
124.                     verified = BCrypt.checkpw(value, encoded);
125.                 } else {
126.                     verified = getDigester(cipherAlgorithm).matches(value, encoded);
127.                 }
128.             }
129.         } catch (Exception e) {
130.             LOG.error("Could not verify encoded value", e);
131.         }
132.
133.         return verified;
134.     }

```

Fig. 28




 verify(String, CipherAlgorithm, String)		100%		100%
---	---	------	---	------

Fig. 29

```

116     public boolean verify(final String value, final CipherAlgorithm cipherAlgorithm, final String encoded) {
117         boolean verified = false;
118
119         try {
120             if (value != null) {
121                 if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
122                     verified = encode(value, cipherAlgorithm).equals(encoded);
123                 } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
124                     verified = BCrypt.checkpw(value, encoded);
125                 } else {
126                     verified = getDigester(cipherAlgorithm).matches(value, encoded);
127                 }
128             }
129         } catch (Exception e) {
130             LOG.error("Could not verify encoded value", e);
131         }
132
133         return verified;
134     }

```

Fig. 30

```

94.     public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
95.         throws UnsupportedOperationException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
96.             IllegalBlockSizeException, BadPaddingException {
97.
98.         String encoded = null;
99.
100.         if (value != null) {
101             if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
102                 Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
103                 cipher.init(Cipher.ENCRYPT_MODE, keySpec);
104
105                 encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)));
106             } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
107                 encoded = BCrypt.hashpw(value, BCrypt.gensalt());
108             } else {
109                 encoded = getDigester(cipherAlgorithm).digest(value);
110             }
111         }
112
113         return encoded;
114     }

```

Fig. 31

● encode(String, CipherAlgorithm)	<div><div></div></div>	100%	<div><div></div></div> 100%
---	------------------------	------	-----------------------------

Fig. 32

```

94     public String encode(final String value, final CipherAlgorithm cipherAlgorithm)
95         throws UnsupportedOperationException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException,
96             IllegalBlockSizeException, BadPaddingException {
97.
98         String encoded = null;
99.
100     if (value != null) {
101         if (cipherAlgorithm == null || cipherAlgorithm == CipherAlgorithm.AES) {
102             Cipher cipher = Cipher.getInstance(CipherAlgorithm.AES.getAlgorithm());
103             cipher.init(Cipher.ENCRYPT_MODE, keySpec);
104
105             encoded = Base64.getEncoder().encodeToString(cipher.doFinal(value.getBytes(StandardCharsets.UTF_8)));
106         } else if (cipherAlgorithm == CipherAlgorithm.BCRYPT) {
107             encoded = BCrypt.hashpw(value, BCrypt.gensalt());
108         } else {
109             encoded = getDigester(cipherAlgorithm).digest(value);
110         }
111     }
112
113     return encoded;
114 }

```

Fig. 33

```

400     protected Set<SyncopeGrantedAuthority> getDelegatedAuthorities(final Delegation delegation) {
401         Map<String, Set<String>> entForRealms = new HashMap<>();
402
403         delegation.getRoles().stream().filter(role -> !GROUP_OWNER_ROLE.equals(role.getKey())).
404         forEach(role -> role.getEntitlements().forEach(entitlement -> {
405             Set<String> realms = Optional.ofNullable(entForRealms.get(entitlement)).orElseGet(() -> {
406                 HashSet<String> r = new HashSet<>();
407                 entForRealms.put(entitlement, r);
408             });
409             return r;
410         }));
411         realms.addAll(role.getRealms().stream().map(Realm::getFullPath).collect(Collectors.toSet()));
412         if (!entitlement.endsWith("_CREATE") && !entitlement.endsWith("_DELETE")) {
413             realms.addAll(role.getDynRealms().stream().map(DynRealm::getKey).collect(Collectors.toList()));
414         }
415     });
416
417     return buildAuthorities(entForRealms);
418 }

```

Fig. 34

```

403 1. replaced boolean return with true for org/apache/syncope/core/spring/security/AuthDataAccessor::lambda$getDelegatedAuthorities$14 -> KILLED
403 2. negated conditional -> SURVIVED
404 1. removed call to java/util/Set::forEach -> SURVIVED
404 2. removed call to java/util/stream/Stream::forEach -> SURVIVED
408 1. replaced return value with Collections.emptySet for org/apache/syncope/core/spring/security/AuthDataAccessor::lambda$getDelegatedAuthorities$15 -> KILLED
412 1. negated conditional -> KILLED
412 2. negated conditional -> KILLED
417 1. replaced return value with Collections.emptySet for org/apache/syncope/core/spring/security/AuthDataAccessor::getDelegatedAuthorities -> KILLED

```

Fig. 35