

# Report Software Testing

## Progetto “1+”: FastJSON

L'obiettivo del progetto 1+ è quello di sperimentare le tecniche di Software Testing e gli strumenti a supporto di esse, rendendo parametrici i test di due classi di test preesistenti nel progetto *Alibaba FastJSON*. Inoltre, si vuole valutare ed analizzare l'andamento della copertura strutturale al variare dei casi di test eseguiti e dei parametri passati in input agli stessi.

Le due classi di test in esame sono *JSONPatchTest\_0.java* e *MapTest2.java*.

Per quanto riguarda la configurazione del progetto, è stato inserito in una pipeline usando il framework di CI delle GitHub Actions. Per stimare la coverage ottenuta con l'esecuzione dei test e generarne dei report, viene utilizzato JaCoCo, agganciato al processo automatico di build. Tuttavia, non essendo possibile leggerne i risultati su SonarCloud, poiché non sono disponibili i sorgenti testati, ma solo i compilati in bytecode che vengono instrumentati on-the-fly, i report sulla coverage vengono analizzati esclusivamente in seguito ad una build in locale.

Come possibile dedurre dal codice originale della classe *JSONPatchTest\_0* ([Codice 1](#)), lo scopo di questa classe è quello di testare il metodo statico *apply* della classe *JSONPatch*. Tale metodo prende come parametri di input due rappresentazioni di *JSONObject* in formato Stringa. La seconda (*patch*) descrive un pool di operazioni da effettuare sul primo *JSONObject* (*original*). Ognuna di queste operazioni può essere una *add*, *remove*, *replace*, *move*, *copy* o *test*.

La classe di test originale prevedeva un metodo di test separato per ogni passaggio di parametri differente al metodo da testare; inoltre, il risultato atteso è passato in maniera hardcoded. Dunque, in sostanza, il metodo under test risulta essere soltanto uno, ma invocato con parametri differenti. Quindi, la parametrizzazione dei test cases è avvenuta separando la logica di configurazione del test dall'effettiva esecuzione, operando nel seguente modo: il costruttore della classe di test chiama internamente un metodo privato *configureTestClass*, passandogli in input i parametri da utilizzare nel metodo di test; inoltre, viene passato anche il risultato atteso come output dell'invocazione del metodo sotto test.

Tale scelta è stata fatta in quanto FastJSON è basato su un concetto di oggetto JSON differente dallo standard (prevede la possibilità di usare come chiavi anche dei tipi di dato non Stringa) ed implementare un oracolo per il calcolo dell'output atteso avrebbe richiesto una completa re-implementazione del metodo under test e della logica sottostante a livello di dominio. Tutto ciò è stato ritenuto andare oltre gli obiettivi del progetto 1+.

Il metodo *configureTestClass* non fa altro che configurare l'ambiente di test, inizializzando gli attributi privati della classe di test con il valore dei due parametri di input da usare nell'invocazione del metodo sotto test e con il valore dell'output atteso. Non c'è bisogno di istanziare la classe sotto test, in quanto il metodo testato è statico. Nel costruttore, è lasciata de-commentata solo una invocazione per volta al metodo di configure; in questo modo, si seleziona ad ogni build una chiamata al configure con parametri diversi, così da valutare il cambiamento ottenuto in termini di copertura.

In [Codice 2](#) è riportata una visione complessiva del codice della classe *JSONPatchTest\_0* re-implementata per avere dei test parametrici.

Come già anticipato, la coverage viene stimata utilizzando il framework JaCoCo, il quale produce principalmente due metriche di copertura: una statement coverage, basata sul numero di istruzioni in bytecode coperte dall'esecuzione dei casi di test, ed una branch coverage, la quale prende in considerazione i costrutti di switch e gli if-statement.

La prima analisi viene effettuata invocando il metodo under test con una patch comprendente un pool di operazioni; in particolare, si tratta di una operazione di *replace*, una di *add* ed una di *remove*.

Com'è possibile vedere in [Figura 1](#), si ottiene una statement coverage complessiva sul progetto del 14% e, in particolare, del package *com.alibaba.fastjson* (in cui è presente la classe *JSONPatch*) del 7%. La singola classe *JSONPatch* risulta avere una copertura strutturale del 51% ed una branch coverage del 38 % ([Figura 2](#)). La copertura dei singoli metodi della classe in questione è visibile in [Figura 3](#).

Una seconda analisi è effettuata lanciando il test con una patch composta da una singola operazione di *move*. Per quanto riguarda la copertura strutturale del codice sorgente, la [Figura 4](#) ci dimostra che si hanno davvero pochi cambiamenti rispetto al caso precedente; in particolare, la percentuale di statement coverage del package *com.alibaba.fastjson* è esattamente uguale. Invece, la copertura della classe *JSONPatch* risulta essere aumentata in statement coverage (dal 51% al 58%), ma non in branch coverage ([Figura 5](#)). Infatti, guardando la copertura strutturale dei metodi della classe ([Figura 6](#)), si ha un aumento sia per il metodo *apply(Object*,

*String*) che per *isObject(String)*. Per comprendere tali risultati, è stato analizzato il decompilato ottenuto a partire dal file *JSONPatch.class*. Il blocco di codice che si occupa di gestire l'operazione di *move* richiede molteplici istruzioni, più di quante ne richiedano un'operazione di *add*, di *replace* e di *remove* insieme.

L'altra classe di test è *MapTest2.java*. Essa include originariamente un solo metodo di test in cui viene testato il metodo statico *parseObject(String text, TypeReference<T> type)* della classe *JSON*. Tale metodo ha il compito di effettuare il parsing della stringa in input (rappresentante un JSON Object) in un oggetto il cui tipo è indicato dalla classe *TypeReference*. In particolare, per un JSON Object ci si aspetta che venga trasformato in una *Map<Object, Object>*. Il test passa se e solo se ad ogni chiave è associato il valore atteso.

Il codice originale della classe è riportato in [Codice 3](#). Anche in questo caso, per la parametrizzazione si è fatto uso di un metodo *configureTestClass* invocato dal costruttore della classe di test. L'unico parametro passato al metodo di configurazione è la stringa rappresentante il *JSONObject*. La configurazione dell'ambiente di test consiste nel settare un attributo privato della classe di test con la stringa rappresentante il JSON Object con cui si vuole eseguire il test e nell'allocare un'istanza di *TypeReference<Map<Object, Object>>*, associando anch'essa ad un attributo privato.

Questa volta, è stato possibile fare un semplice parsing della stringa rappresentante il *JSONObject* per ottenerne un array di chiavi di tipo *Object* da utilizzare nel metodo di test, ma soltanto perché le chiavi del JSON Object previsto per il test erano unicamente di tipo stringa o intero. Inoltre, è stata utilizzata la libreria **JSON-JAVA (org.json)** per ottenere un oggetto JSON da cui estrarre gli output attesi per il test. Dopodiché, è stata creata una classe *Oracle* interna alla classe di test, la quale agisce da oracolo di test, fornendo gli output attesi. La nuova implementazione parametrica di *MapTest2* è consultabile in [Codice 4](#).

Essendo il test originario eseguito con un'unica rappresentazione in stringa di un *JSONObject*, non è possibile valutare la variazione della copertura al cambio dei valori dei parametri di test (a meno di prevedere ed inserire nuovi valori per il parametro di input). La coverage che si ottiene a seguito dell'esecuzione del solo test in esame è consultabile in [Figura 7](#) e [Figura 8](#): la statement coverage su tutti i package del progetto ammonta al 5%, quella del package *org.alibab.fastjson* (contenente la classe *JSON*) appena all'1%, mentre la statement coverage della classe *JSON* è del 9%. Sono valori molto bassi, ma giustificati dall'esecuzione di un solo test case su un metodo di una singola classe, la quale ha una size molto elevata.

Sono state in seguito valutate le metriche di copertura dell'intera test-suite, composta dai metodi di test di entrambe le classi di test parametrizzate in precedenza. Ci si aspetta, naturalmente, un incremento dei valori di coverage rispetto a quelli analizzati precedentemente.

Relativamente ai risultati ottenuti dal primo run del test di *JSONPatchTest\_0.java* (*patch* composta da più di 1 operazione), la branch coverage è aumentata dal 6% al 7% su tutti i package ([Figura 9](#)), mentre, dal punto di vista percentuale, la statement coverage è rimasta assestata sul 14%. Tuttavia, è possibile notare che il numero di missed instructions è leggermente diminuito. Quindi, come ci si aspettava, si ha un miglioramento generale della coverage.

In particolare, è molto interessante notare come la statement coverage della classe *JSON* sia passata dal 9%, con la sola esecuzione del test di *MapTest2*, al 14%, con l'esecuzione dell'intera test-suite ([Figura 10](#)). Questo perché la classe viene coinvolta anche nel flusso d'esecuzione del test di *JSONPatchTest\_0*.

Repository GitHub: <https://github.com/AndreaPepe/fastjsonTesting>.

## Progetto “2”: Apache Syncope

### Dominio

Apache *Syncope* è un sistema open-source per la gestione di identità digitali in ambienti enterprise. Il suo obiettivo primario è quello di gestire i dati e le informazioni degli utenti e dei loro account su sistemi e applicazioni. Inoltre, svolge il ruolo di access management system, dando la possibilità di definire delle politiche di accesso a risorse, applicabili a singoli utenti, a gruppi o persino ad altri oggetti definibili in modo personalizzato (e.g. stampanti, dispositivi IoT, etc.). Per ulteriori informazioni, consultare la documentazione ufficiale del progetto al seguente [link](#).

### Prima classe: DefaultPasswordGenerator

Una delle due classi scelte per l'attività di Software Testing è stata *DefaultPasswordGenerator*. Tale scelta è stata dettata principalmente dall'importanza della classe, in quanto la gestione e la generazione delle

passwords è un punto focale dell'identity management. Inoltre, è stata scelta anche perché la documentazione forniva qualche indicazione utile a riguardo. Tuttavia, bisogna segnalare che una delle pecche del progetto è sicuramente la mancanza di una documentazione dettagliata.

Come desumibile dal nome della classe e dal seguente commento di documentazione, la classe fornisce la funzionalità di generare automaticamente delle password conformi a determinate policies:

*"Generate random passwords according to given policies."*

*"When no minimum and / or maximum length are specified, default values are set."*

*WARNING: This class only takes DefaultPasswordRuleConf into account."*

Al seguente [link](#) è possibile trovare ciò che la documentazione ufficiale riporta in merito alle password e alle possibili policies cui devono aderire. In particolare, viene specificato che:

*"When defining a password policy, the following information must be provided:*

- allow null password - whether a password is mandatory for Users or not*
- history length - how many values shall be considered in the history*
- rules - set of password rules to evaluate with the current policy"*

Inoltre, per quanto riguarda le rules possibili, abbiamo quanto segue:

*"The default password rule (enforced by DefaultPasswordRule and configurable via DefaultPasswordRuleConf) contains the following controls:*

- maximum length - the maximum length to allow; 0 means no limit set;*
- minimum length - the minimum length to allow; 0 means no limit set;*
- non-alphanumeric required*
- alphanumeric required*
- digit required*
- lowercase required*
- uppercase required*
- must start with digit*
- must not start with digit*
- must end with digit*
- must not end with digit*
- must start with alphanumeric*
- must start with non-alphanumeric*
- must not start with alphanumeric*
- must not start with non-alphanumeric*
- must end with alphanumeric*
- must end with non-alphanumeric*
- must not end with alphanumeric*
- must not end with non-alphanumeric*
- username allowed - whether a username value can be used*
- words not permitted - list of words that cannot be present, even as a substring;*
- schemas not permitted - list of schemas whose values cannot be present, even as a substring;*
- prefixes not permitted - list of strings that cannot be present as a prefix;*
- suffixes not permitted - list of strings that cannot be present as a suffix."*

Non essendo disponibile una Javadoc per la classe under test, è stato necessario assumere un approccio al test maggiormente tendente al white-box piuttosto che al black-box, ma tentando di ragionare sempre sulle funzionalità astratte che la classe deve implementare.

Come prima cosa, si decide di testare il metodo principale della classe: *generate(List<PasswordPolicy> policies)*. Esso genera una password in accordo alla lista di policies specificata. Il codice di tale metodo è consultabile nella tabella [Codice 5](#).

L'unico parametro di input del metodo è una lista di *PasswordPolicy*, la quale è un'interfaccia che è implementata dalla classe *DefaultPasswordPolicy*. Il tipo di ritorno è una stringa, ma può essere lanciata un'eccezione di tipo *InvalidPasswordRuleConf* se la configurazione delle policies è non valida.

Alla luce di queste informazioni, si procede nell'effettuare domain partitioning. Per quanto riguarda l'unico parametro di input, si tratta di una lista; quindi, si considerano in prima battuta i seguenti sottoinsiemi:

- policies: {empty list}, {non empty list}, {null}*

Tuttavia, si tratta di una lista di *PasswordPolicy*, dunque, un tipo di dato complesso che, per quanto descritto in precedenza può avere una configurazione più o meno valida all'interno del dominio di riferimento. Si nota

che lo stato della classe *DefaultPasswordRule*, contenente le regole della policy, è determinato dal valore degli attributi *allowNullPassword*, *historyLength* e *rules*; è ovvio che il fatto che la password sia o meno obbligatoria per l'utente o la history length non sono rilevanti dal punto di vista della generazione automatica della password stessa. L'unico fattore che realmente impatta il risultato ottenuto è l'insieme di regole che la password dovrà rispettare.

Dunque, si distingue subito che tra le *non empty lists* è lecito distinguere tra liste con policies valide e non valide, quindi le classi di equivalenza individuate diventano:

- *policies*: {empty list}, {valid policies}, {invalid policies}, {null}

Per quanto riguarda i possibili output, si hanno tre possibilità rappresentate dal verificarsi di un'eccezione per regole non valide, da password conforme alle policies e da password non conforme alle policies.

Effettuando la boundary analysis, si ottengono i seguenti boundary-values:

- *policies*: empty list, list with 1 valid policy, list with 1 invalid policy, null

Essendo il parametro di input unico, è possibile soltanto avere un approccio unidimensionale. Si procede quindi alla definizione ad alto livello dei casi di test da implementare e del risultato atteso, come riportato in [Tabella 1](#). In occasione di una lista di policies vuota, ci si aspetta che vengano rispettate le regole di default, come confermato anche da una successiva analisi del codice sorgente. Tali regole hanno unicamente constraints sulla size della password: lunghezza minima di 8 caratteri e massima di 64.

La policy non valida è stata ottenuta indicando di dover rispettare due regole in contrasto tra loro: la password deve terminare con una lettera e la password deve terminare con un numero.

Per l'esecuzione dei test, è stato necessario effettuare un mock statico della classe *ImplementationManager*. Tutti i test eseguiti in questa prima iterazione vanno a buon fine e si ottengono risultati per la statement coverage e la branch coverage visibili in [Figura 11](#) e in [Figura 12](#). Le percentuali di coverage sono già molto elevate, poiché il metodo testato invoca internamente quasi tutti gli altri metodi, perlopiù protected, della classe.

Si vuole cercare di migliorare la coverage ottenuta, sia di classe che dei metodi, andando in particolare a stimolare con inputs differenti il metodo protected *check(DefaultPasswordRuleConf)*, il quale controlla che le policies siano valide. Infatti, è questo il punto che con più probabilità può presentare dei difetti, ad esempio ritenendo valida una policy non valida o viceversa.

Per fare ciò, è necessario operare sullo stato della classe *DefaultPasswordRuleConf*, la quale contiene i settaggi di tutte le *rules* elencate nella documentazione, che viene utilizzata per caratterizzare le varie policies.

Tale stato è caratterizzato da ben 24 attributi (uno per ognuna delle *rules* indicate sulla documentazione), di cui due di tipo intero (massima e minima size della password), 16 di tipo booleano e 4 di tipo List<String> per le blacklist di parole da non contenere come prefisso, suffisso o all'interno della password.

Si procede con l'individuazione delle classi di equivalenza:

- *maxLength*: {< 0}, {0}, {> 0}
- *minLength*: {≤ maxLength}, {> maxLength}
- Per i parametri booleani si individuano le classi {true} e {false}; per completezza si elencano i nomi dei parametri: *nonAlphanumericRequired*, *alphanumericRequired*, *digitRequired*, *lowercaseRequired*, *uppercaseRequired*, *mustStartWithDigit*, *mustntStartWithDigit*, *mustEndWithDigit*, *mustntEndWithDigit*, *mustStartWithNonAlpha*, *mustntStartWithNonAlpha*, *mustStartWithAlpha*, *mustntStartWithAlpha*, *mustEndWithNonAlpha*, *mustntEndWithNonAlpha*, *mustEndWithAlpha*, *mustntEndWithAlpha*, *usernameAllowed*;
- *wordsNotPermitted*, *schemasNotPermitted*, *prefixesNotPermitted*, *suffixesNotPermitted*: {emptyList}, {nonEmptyList}

Per questi ultimi parametri, non è stata considerata la classe di equivalenza {null} perché ritenuta non significativa all'interno dello scenario.

Effettuando la boundary-value analysis, si identificano i seguenti valori:

- *maxLength*: - 1, 0, 10
- *minLength*: = *maxLength*, *maxLength* + 1
- Parametri booleani: *true*, *false*
- *wordsNotPermitted*, *schemasNotPermitted*, *prefixesNotPermitted*, *suffixesNotPermitted*: *emptyList*, *List with 1 String*

Come valore >0 per *maxLength* si è scelto un valore sufficientemente grande così che la stringa possa contenere abbastanza caratteri per poter verificare la validità dei requisiti espressi dai parametri booleani.

Per i primi due parametri, si userà un approccio multidimensionale, essendo tra loro strettamente correlati.

Per i parametri di tipo List e per i booleani *nonAlphanumericRequired*, *alphanumericRequired*, *digitRequired*, *lowercaseRequired*, *uppercaseRequired* e *usernameAllowed* si userà un approccio unidimensionale.



Per tutti gli altri parametri booleani, si userà un approccio multidimensionale sulle coppie e triple di parametri che possono potenzialmente essere in contrasto tra loro: ad esempio tra *mustStartWithDigit* e *mustntStartWithDigit* oppure tra *mustStartWithAlpha* e *mustStartWithNonAlpha*.

In generale, per  $X = Digit, Alpha, NonAlpha$ , si avrà un approccio multidimensionale per tutte le coppie di parametri del tipo *mustStartWithX* e *mustntStartWithX*, così come per *mustEndWithX* e *mustntEndWithX*. Inoltre, si avrà un approccio multidimensionale per le triple (*mustntStartWithDigit*, *mustntStartWithAlpha*, *mustntStartWithNonAlpha*) e (*mustntEndWithDigit*, *mustntEndWithAlpha*, *mustntEndWithNonAlpha*).

È importante notare che sono state desunte le seguenti informazioni dal codice sorgente:

- *isAlphanumericRequired* è ritenuto soddisfatto sia dalla presenza di una lettera che di una cifra;
- in tutti gli altri casi, la dicitura *nonAlpha* è soddisfatta da qualsiasi carattere diverso da una lettera (quindi cifra o carattere speciale); invece, *Alpha* è soddisfatto solo da lettere e *Digit* solo da cifre.

È chiaro che i requisiti espressi dai parametri booleani hanno un effetto solo nel caso in cui tali parametri assumano valore *true*. In tutti gli altri casi, è come se quei parametri non esistessero. Alla luce di ciò, tra tutte le possibili combinazioni di parametri del test, ne sono state selezionate alcune ritenute più significative delle altre. Sono stati così definiti ulteriori casi di test, da aggiungere a quelli già implementati nella prima iterazione; i valori dei parametri e i risultati attesi sono consultabili nella [Tabella 2](#) e dalle **risorse esterne**.

L'implementazione di tali test-cases si è concretizzata nell'evoluzione della classe di test precedentemente implementata. I test sono strutturati in maniera parametrica e la classe ha dei metodi privati che hanno il compito di computare il risultato atteso o un eventuale verificarsi di un'eccezione, in accordo a quanto specificato nella documentazione. L'esecuzione dei test rivela alcuni fallimenti:

- con *minLength* = 1, *maxLength* = 0 ci si aspetta di ottenere una password lunga minimo un carattere, in quanto, settando *maxLength* = 0 non si imposta un limite per la lunghezza massima; invece, si ottiene un'eccezione di tipo *ArrayIndexOutOfBoundsException*;
- se si indica che la password deve sia terminare con una cifra che non terminare con una cifra, ci si aspetta un'eccezione, ma si ottiene una password che termina con una cifra;
- accade lo stesso indicando che la password deve iniziare con un *Alpha* e anche con un *NonAlpha*; l'eccezione attesa non viene lanciata e si ottiene una password che inizia con un *NonAlpha*;
- analogamente, specificando una terminazione sia con un *Alpha* che con un *NonAlpha*;
- se si specifica che la password NON deve iniziare né con un *Alpha*, né con un *NonAlpha*, né con un *Digit*, ci si aspetta un'eccezione perché non c'è alcun carattere con cui la password possa iniziare, ma invece viene restituita una password che ha come primo carattere un *Alpha*;
- analogamente se si indica che la password NON deve terminare con nessuna delle tre tipologie di caratteri.

La maggior parte delle failures avvengono per la presenza di bug, causati dalla mancanza di controlli di consistenza, in particolare, nel metodo *protected check()* della classe *DefaultPasswordGenerator*. Inoltre, come è ben visibile dai report di coverage generati da PITest (per la mutation coverage) e da JaCoCo, risulta esserci una linea di codice inaspettatamente non coperta: ciò è dovuto alla mancanza di un operatore logico di NOT nella condizione di un if-statement, come visibile in [Figura 14](#) e in [Figura 15](#).

Con questa seconda iterazione, sono stati ottenuti miglioramenti significativi sia per la statement coverage che, soprattutto, per la branch coverage della classe e dei metodi, come visibile in [Figura 16](#) e in [Figura 18](#).

Attraverso l'uso del framework PITest, è stata valutata la mutation coverage, pari al 92% ([Figura 17](#)). La branch coverage del metodo *protected check()* è giunta ad un valore ritenuto soddisfacente.

Tuttavia, per incrementare ulteriormente la copertura e la mutation coverage, si decide di testare anche l'altro metodo pubblico della classe, ovvero *generate(ExternalResource)*, per il quale risultano esserci mutanti non uccisi ([Figura 19](#)). *ExternalResource* è un'interfaccia che, in particolare, definisce delle operazioni per settare ed ottenere una singola *PasswordPolicy*. Il metodo che si vuole testare non fa altro che ottenere tale policy dall'istanza della classe che implementa *ExternalResource* ed usarla per invocare l'altro metodo pubblico precedentemente testato.

Essendo l'esito del test determinato sostanzialmente dagli stessi parametri di configurazione dei test-cases precedenti, si decide di sfruttare le stesse combinazioni di valori per i parametri in questione per realizzare i casi di test per il nuovo metodo. Dal punto di vista implementativo, ciò è stato ottenuto aggiungendo un ulteriore metodo di test all'interno della classe di test originaria. Ovviamente, anche questi test-cases evidenziano gli stessi fallimenti rilevati in precedenza.

Come atteso, la coverage calcolata con JaCoCo è ulteriormente migliorata ([Figura 20](#)), così come la mutation coverage, salita al 94% ([Figura 21](#)).

Assumendo un profilo operativo con probabilità di utilizzo uniforme rispetto al solo insieme di casi di test progettati ed implementati, visibili ancora una volta in [Tabella 2](#), e ai quali aggiungere i due casi di test con una lista di policy *null* ed una vuota implementati nella prima iterazione (prima e quarta riga di [Tabella 1](#)), si hanno in totale 24 combinazioni testate e, quindi, ognuna di esse ha probabilità di utilizzo pari a  $1/24 = 0.041666$ . Essendo che 6 di esse evidenziano fallimenti, la reliability stimata ammonta al 75% per entrambi i metodi pubblici testati.

## Seconda classe: Encryptor

La successiva classe che si è deciso di testare è la classe *Encryptor*, per la quale non sono presenti tantissime informazioni sulla documentazione ufficiale, ma abbastanza per poterne progettare e implementare dei test, inferendo, ove necessario, dal contesto e dal codice sorgente, adottando una strategia di Software Testing di tipo gray-box.

La classe non è documentata in maniera dedicata, ma è stato possibile reperire alcune informazioni disposte in modo sparso nella documentazione ufficiale, utili ad intuirne le funzionalità e il comportamento previsto; ad esempio, dalla frase “*password (Users only) - hashed or encrypted value, depending on the selected password.cipher.algorithm which can be used for authentication*”, si intuisce che l'uso di un encryptor sia necessario per la cifratura o l'hashing delle password e che possono essere selezionati diversi algoritmi per effettuare tali operazioni. Sugli algoritmi disponibili, la [documentazione](#) è molto chiara ed esaustiva, elencandoli:

- SHA-1
- SHA-256
- SHA-512
- AES
- S-MD5
- S-SHA-1
- S-SHA-256
- S-SHA-512
- BCRYPT

Sono tutti algoritmi di hashing, fatta eccezione per AES (algoritmo di cifratura a blocchi, a chiave simmetrica).

La documentazione fornisce anche altre due importanti informazioni:

- “*The value of the secretKey property in the security.properties file is used for AES-based encryption / decryption. Besides password values, this is also used whenever reversible encryption is needed, throughout the whole system.*”
- “*When the secretKey value has length less than 16, it is right-padded by random characters during startup, to reach such minimum value. It is strongly recommended to provide a value long at least 16 characters, in order to avoid unexpected behaviors at runtime, especially with high-availability.*”

Come prima funzionalità da testare, si sceglie quella ritenuta più importante, ovvero la cifratura. La classe *Encryptor* espone un metodo pubblico *encode()*, il quale prende in input due parametri: il primo parametro *value* è di tipo String e rappresenta la stringa da cifrare, mentre il secondo parametro *cipherAlgorithm* è di tipo *CipherAlgorithm* (una Enum) e rappresenta ovviamente l'algoritmo da utilizzare. Tuttavia, la documentazione fa riferimento anche ad una *secretKey* che è necessaria per l'encryption con AES, ma, se non viene specificata alcuna *secretKey*, ne viene impostata una di default.

Per una prima iterazione di test, si decide di utilizzare esclusivamente la secret key di default, focalizzandosi sulla riuscita o meno dell'encoding. Si procede effettuando **domain partitioning** per i parametri presi in input dal metodo sotto test:

- *value*: {null}, {stringa vuota}, {stringa di dimensione inferiore ad un blocco di cifratura}, {stringa di dimensione superiore ad un blocco di cifratura}
- *cipherAlgorithm*: {SHA - 1}, {SHA - 256}, {SHA - 512}, {AES}, {S - MD5}, {S - SHA - 1}, {S - SHA - 256}, {S - SHA - 512}, {BCRYPT}

Effettuando **boundary-value analysis**, si decide che la stringa di dimensione inferiore ad un blocco di cifratura, deve essere inferiore al blocco più piccolo usato tra tutti gli algoritmi (64 bit per Blowfish, usato da BCRYPT), mentre per la stringa più grande del blocco di cifratura, dovrà essere più grande del blocco di dimensioni maggiori tra tutti gli algoritmi (1024 bit per SHA-512). Dunque, si ha:

- *value*: null, "" (stringa vuota), *stringa* < 64 bit, *stringa* > 1024 bit
- *cipherAlgorithm*: SHA - 1, SHA - 256, SHA - 512, AES, S - MD5, S - SHA - 1, S - SHA - 256, S - SHA - 512, BCRYPT

Si è deciso di avere un approccio multidimensionale, considerando l'insieme di casi di test da category partition riportati in [Tabella 3](#). Il controllo che la stringa codificata ritornata dal metodo sia valida è stato effettuato invocando il metodo *verify()* della stessa classe *Encryptor*, assumendolo essere corretto.

Come visibile in [Figura 22](#) e in [Figura 23](#), si ottiene una class coverage del 77% per quanto riguarda la statement coverage e del 70 % per quanto riguarda la branch coverage; inoltre, per il metodo testato *encode()*, si ha una statement coverage del 100% ed una branch coverage dell' 87%.

La mutation coverage, calcolata usando PITest, è pari al 50% ([Figura 24](#)). Si nota che la branch coverage del metodo *encode()* è ancora migliorabile in quanto non è stato esplorato un ramo di un if-statement che viene considerato quando *cipherAlgorithm* è *null* ([Figura 25](#)), valore inizialmente non previsto nei casi di test progettati ed implementati fin'ora. Si decide dunque di aggiungere altri 4 test-cases in cui tale valore è *null*, combinandolo con gli altri 4 valori del parametro *value* originariamente previsti.

Inoltre, per migliorare la coverage, si decidono di testare anche i metodi *verify(value, cipherAlgorithm, encoded)* e *decode(encoded, cipherAlgorithm)*. In aggiunta, si otterrà un'istanza di *Encryptor* passando una *secretKey* non di default di lunghezza minore di 16 caratteri, per verificare che, come specificato dalla documentazione, ne venga effettuato il padding a 16 caratteri, se non verranno sollevati errori (la chiave deve essere necessariamente lunga 16 bytes).

Per il metodo *verify()*, le classi di equivalenza e i boundary-values individuati per i due parametri *value* e *cipherAlgorithm* sono le stesse del metodo *encode()*, mentre per il terzo parametro *encoded* abbiamo:

- *encoded* = {*encode* corretto di *value*}, {*encode* non corretto di *value*}

I boundary-values scelti sono *encode(value, cipherAlgorithm)* e "*randomCrap*". È stato adottato un approccio multidimensionale tra i tre parametri, definendo i test-cases elencati in [Tabella 4](#).

Il metodo *decode()*, in base a quanto specificato nella documentazione, è utilizzabile in maniera efficace solo con AES, che è l'unico algoritmo di cifratura a chiave simmetrica tra quelli elencati. Dunque, per i due parametri *encoded* e *cipherAlgorithm* si hanno le seguenti classi di equivalenza:

- *encoded*: {*null*}, {stringa vuota}, {stringa codificata correttamente}, {stringa mal codificata}
- *cipherAlgorithm*: {*null*}, {AES}, {≠ AES}

Risultanti nei seguenti boundary-values:

- *encoded*: *null*, "", *encode('myString', CipherAlgorithm.AES), 'randomCrap'*
- *cipherAlgorithm*: *null*, *AES*, *BCRYPT*

A causa della mancanza di documentazione, è stato necessario un approccio white-box per comprendere gli output attesi nelle diverse situazioni. Quando viene passato un *CipherAlgorithm* non consono al decoding, era naturale attendersi il lancio di un'eccezione, ma dal codice sorgente si è appreso che il metodo ritorna *null*. Si entra in logica di errore solo se si verificano errori durante il processo di decoding vero e proprio. I casi di test ottenuti da category partition, applicando un approccio multidimensionale, sono elencati in [Tabella 5](#).

Grazie alla seconda iterazione di test, si è ottenuta una migliore coverage di classe (sia statement coverage che branch coverage) per *Encryptor*, così come per i singoli metodi, come testimoniato in [Figura 26](#) e [Figura 27](#). Inoltre, anche la mutation coverage risulta essere migliorata, arrivando al 64%, rispetto al 50% ottenuto nella prima iterazione ([Figura 28](#)).

Un **profilo operativo** per l'operazione *encode()* è definito in base ai valori assumibili dai due parametri, per i quali è stato utilizzato un approccio multidimensionale, testando tutte le possibili combinazioni:

- *value*: [*null*, "", "ab", stringa lunga 129 caratteri]
- *cipherAlgorithm*: [*null*, SHA-1, SHA-256, SHA-512, AES, S-MD5, S-SHA-1, S-SHA-256, S-SHA-512, BCRYPT].

In totale si hanno 40 combinazioni, ognuna con probabilità di utilizzo uniforme pari a 0.025. Non si ha nessun fallimento nell'esecuzione dei test cases, quindi la **reliability** per tale metodo risulta essere del 100%.

Analogamente, per i metodi *verify()* e *decode()*, si considerano profili operazionali desumibili dalla [Tabella 4](#) e dalla [Tabella 5](#), in cui ogni combinazione ha rispettivamente probabilità di utilizzo pari a  $1/80 = 0.0125$  e ad  $1/12 = 0.083333$ . Anche in questo caso, non essendoci fallimenti nei test, la reliability ammonta al 100%.

Lo sviluppo dei test è stato incluso nel ciclo di build del progetto utilizzando le GitHub Actions ([link repository GitHub](#)) ed è stato inoltre integrato con [SonarCloud](#), rendendo possibile l'analisi della coverage ottenuta tramite JaCoCo. Purtroppo, non è stato possibile stimare l'adeguatezza dei test secondo dei criteri data-flow oriented con il tool *ba-dua*, a causa di incompatibilità tra le versioni dei JDK richieste dal tool e dalle altre dipendenze del progetto.

# Progetto “2”: Apache BookKeeper

## Dominio

*Apache BookKeeper* è un servizio di storage ottimizzato per carichi di lavoro real-time ed è in grado di garantire scalabilità, resistenza ai guasti, consistenza, replicazione e durabilità.

Dal punto di vista della terminologia, in *BookKeeper*, ogni unità di un log da salvare è chiamata **entry**. Flussi di log entries sono chiamati **ledgers** e vengono salvati su servers chiamati **bookies**.

Per maggiori informazioni, consultare la [documentazione ufficiale](#).

## Prima classe: BookKeeper

Come prima classe da testare, è stata scelta *BookKeeper*, in quanto è la classe che rappresenta il client utilizzabile dall'utente per connettersi ad un server *BookKeeper*, e dunque è di fondamentale importanza.

Come desumibile dalla [Javadoc](#), ci sono 4 operazioni possibili: start di un nuovo ledger, scrittura su un ledger, lettura da un ledger, rimozione di un ledger.

Alla luce di ciò, si decide in una prima iterazione di testare un paio delle funzionalità descritte, in particolare la creazione e la rimozione di un ledger.

Per quanto riguarda la creazione, la classe espone diverse operazioni che la realizzano e ci sono sia creazioni sincrone che asincrone. Si decide di iniziare dal testing del metodo *createLedger()* sincrono e che prende in input il maggior numero di parametri tra tutti i metodi sincroni che offrono la stessa funzionalità.

I parametri risultano essere i seguenti:

- (int) *ensSize* - number of bookies over which to stripe entries
- (int) *writeQuorumSize* - number of bookies each entry will be written to
- (int) *ackQuorumSize* - number of bookies which must acknowledge an entry before the call is completed
- (BookKeeper.DigestType) *digestType* - digest type, either MAC or CRC32
- (byte[]) *passwd* - password
- (Map<String, byte[]>) *customMetadata* - optional customMetadata that holds user specified metadata

Il metodo ritorna un oggetto di tipo *LedgerHandle*, rappresentante un handle al ledger creato; inoltre, l'operazione dichiara di poter lanciare delle eccezioni di tipo *InterruptedException* e *BKException*.

Per quanto riguarda il parametro *digestType*, vengono indicati solo i valori MAC e CRC32. Tuttavia, dalla [documentazione](#), si desume che la enum presenta anche i valori DUMMY e CRC32C, pertanto, anche tali valori verranno presi in considerazione per i test.

Inoltre, sempre dalla documentazione, si apprende che bisogna rispettare il seguente constraint:

*“When creating a ledger, the following invariant must hold:*

***E ≥ Qw ≥ Qa***

*Thus, the ensemble size (E) must be larger than the write quorum size (Qw), which must in turn be larger than the ack quorum size (Qa). If that condition does not hold, then the ledger creation operation will fail.”*

Si procede con l'individuazione delle classi di equivalenza e quindi dei valori derivanti da boundary-value analysis, ottenendo i risultati in [Tabella 6](#). Per i parametri riguardanti le size, è stato scelto un approccio multidimensionale, essendo correlati dal constraint descritto in precedenza. Per gli altri parametri, è stato adottato un approccio unidimensionale. Ciò ha portato alla definizione dei casi di test elencati in [Tabella 7](#).

I test sono stati realizzati lanciando un'istanza di server di *BookKeeper* (e anche una necessaria di *ZooKeeper*) in locale, utilizzando la classe apposita *LocalBookKeeper*, già presente nel progetto, come consigliato nella seguente [pagina web](#). L'istanziatura, sia del server, che della classe client da testare, è stata effettuata prima dell'esecuzione di ogni metodo di test (@Before) e opportunamente chiusa una volta terminato il metodo (@After). Ciò ha garantito che ogni metodo di test usufruisse di un'istanza pulita di server. Essendo questa una procedura comune a tutte le classi di test per la classe *BookKeeper*, si è deciso di creare una classe astratta che implementasse tali operazioni, estesa dalle classi di test concrete.

Per quanto riguarda il metodo *createLedger()*, si è assunto che l'output atteso dovesse essere un'eccezione se la relazione tra le size non fosse stata rispettata oppure se l'*ensSize* fosse negativo.

Per il metodo *deleteLedger()*, invece, l'unico parametro di input è *lId*, di tipo *long*, rappresentante un identificativo del ledger che si vuole eliminare. Non viene ritornato alcun valore, ma ci si aspetta un'eccezione in caso di errore. Per il parametro di input, si individuano le seguenti classi di equivalenza:

- *lId*: {ID di un ledger esistente}, {ID di un ledger non esistente}

Ciò si tramuta nei boundary-values:



- *lId: ledgerHandle.getId(), -1*

assumendo che ID negativi non vengano mai associati ad un ledger al momento della sua creazione. Ci si aspetta che venga lanciata un'eccezione quando la *deleteLedger()* viene invocata con un ID che non corrisponde ad alcun ledger. Di conseguenza, vengono individuati i casi di test descritti in [Tabella 8](#).

La copertura della classe e dei suoi metodi ottenuta con la test-suite costruita nella prima iterazione è consultabile in [Figura 29](#), [Figura 30](#) e [Figura 31](#). Per il metodo *deleteLedger()*, la statement coverage è del 100%, mentre la branch coverage non è calcolabile in quanto non sono identificabili dei branch. La mutation coverage della classe è pari al 33%, come visibile in [Figura 32](#).

Tuttavia, ci sono delle run di test che falliscono, non ottenendo il risultato atteso:

- Metodo *createLedger()*: i casi di test per i quali *ensSize < writeQuorumSize*, in cui il risultato atteso è un'eccezione (non viene rispettata la relazione  $E \geq Qw$ ), provocano invece una situazione di stallo, causando un'esecuzione infinita della run di test. Tali situazioni sono classificate come difettose. Inoltre, nei casi in cui la relazione tra le size viene rispettata, ma *ensSize < 0*, viene lanciata un'eccezione non di tipo *BKException*, bensì di tipo *IllegalArgumentException*. Quest'ultimo caso è ritenuto comunque accettabile, poiché non è specificato nella documentazione quale eccezione attendersi in tale situazione, ma sarebbe consigliabile esplicitarlo.
- Metodo *deleteLedger()*: il tentativo di eliminare un ledger con un ID non esistente, non provoca il lancio di un'eccezione verso il client (classe *BookKeeper*), contrariamente a quanto ci si aspetterebbe in una situazione simile. Infatti, ciò è confermato dal log dell'esecuzione del caso di test, riportato in [Figura 33](#), in cui è esplicitamente indicato che viene ritornato "success". Questo è considerato un difetto da sistemare. Notificare l'utente che l'ID passato non esiste è utile per evitare che egli sia convinto di aver rimosso un ledger, quando in realtà ha usato un ID errato ed il ledger è ancora attivo.

Nella seconda iterazione, con lo scopo di aumentare la copertura della classe, vengono testati gli altri metodi *createLedger()* esposti dalla classe ed il metodo *openLedger()*.

Per quanto riguarda le varianti di *createLedger()*, sono le seguenti:

- *createLedger(ensSize, writeQuorumSize, ackQuorumSize, digestType, passwd)*
- *createLedger(ensSize, qSize, digestType, passwd)*
- *createLedger(digestType, passwd)*

Per il primo dei tre metodi, i parametri sono gli stessi del metodo *createLedger()* testato nella prima iterazione, semplicemente non viene passato il parametro *customMetadata*. Dunque, non considerando tale parametro, le classi di equivalenza, i boundary-values e i casi di test con i risultati attesi, rimangono gli stessi ([Tabella 7](#)).

Per il secondo dei tre metodi, viene introdotto il parametro *qSize*, ovvero quorum size, il quale rappresenta la taglia sia del *writeQuorum* che dell'*ackQuorum*. Di conseguenza, la relazione tra le size viene semplificata in  $ensSize \geq qSize$ . Per le classi di equivalenza e i boundary-values dei parametri individuati per tale metodo, consultare la [Tabella 9](#), mentre per i casi di test la [Tabella 10](#), generata considerando un approccio multidimensionale solo per i parametri *ensSize* e *qSize*, unidimensionale per gli altri.

Per il terzo e ultimo metodo, gli unici parametri sono i già discussi *digestType* e *passwd*. Dalla [documentazione](#), si evince che passando solo tali parametri, verrà restituito un handle ad un ledger che abbia *ensSize* pari a 3 e *writeQuorumSize* ed *ackQuorumSize* pari a 2. Tali informazioni sono state considerate per valutare il successo o meno dei casi di test, andando a verificare i metadati del ledger. I casi di test ottenuti applicando un approccio unidimensionale ai due parametri in questione sono elencati in [Tabella 11](#).

Il metodo *openLedger()* svolge la funzionalità di aprire un ledger in lettura. Prende come parametri in input:

- *lId*: di tipo *long*; ID del ledger da aprire
- *digestType*: enumerazione di tipo *BookKeeper.DigestType*; indica il tipo di digest per la password
- *passwd*: *byte[]* contenente la password per il ledger; deve coincidere con quella indicata al momento della creazione del ledger.

Dunque, sono state individuate le seguenti classi di equivalenza per gli attributi:

- *lId*: {ID di un ledger esistente}, {ID di un ledger inesistente}
- *digestType*: {MAC}, {CRC32}, {CRC32C}, {DUMMY}
- *passwd*: {password corretta vuota}, {password corretta non vuota}, {password errata}

L'analisi dei valori di boundary è risultata in:

- *lId: ledgerHandle.getId(), -1*

- *digestType*: *MAC, CRC32, CRC32C, DUMMY*
- *passwd*: [] *corretta*, "*passwd*".getBytes() *corretta*, "*wrongPassword*".getBytes() *errata*

Quando è indicato che la password deve essere corretta, significa che la password usata come parametro per il metodo *openLedger()* deve essere la stessa specificata al momento della creazione del ledger stesso. Quando, invece, la password deve essere errata, il ledger deve essere creato (solo se si è nel caso di ledger esistente) con una password diversa da quella usata per il test.

Quindi, sono stati selezionati i casi di test in [Tabella 12](#), applicando un approccio multidimensionale per i parametri *lld* e *passwd*, unidimensionale per *digestType*. In caso di ledger inesistente o di password errata ci si aspetta un'eccezione; in tutti gli altri casi un *LedgerHandle* valido.

La coverage della classe ottenuta con questa seconda iterazione di test è leggermente migliorata, arrivando al 42% di statement coverage e al 27% di branch coverage ([Figura 34](#)). Inoltre, i metodi testati nella seconda iterazione risultano essere completamente coperti ([Figura 35](#)). Anche la mutation coverage, calcolata con PITest, è aumentata fino al raggiungimento del 42% ([Figura 36](#)).

Le run di test per il metodo *openLedger()* non hanno evidenziato fallimenti, mentre, per quanto riguarda i metodi *createLedger(ensSize, writeQuorumSize, ackQuorumSize, digestType, passwd)* e *createLedger(ensSize, qSize, digestType, passwd)*, è stato riscontrato lo stesso fallimento discusso in precedenza: il caso in cui *ensSize < writeQuorumSize* oppure *ensSize < qSize* porta l'esecuzione a non terminare mai, anziché lanciare un'eccezione.

Per l'individuazione dei **profili operazionali** e per la stima della **reliability**, si è proceduto nel seguente modo: il calcolo è stato effettuato a livello di metodo e considerando solo le combinazioni dei parametri effettivamente testate. La probabilità di utilizzo di ogni combinazione di parametri facente parte del profilo operazionale è stata assunta uniforme: in altre parole, se un metodo è stato testato con  $X$  diverse combinazioni di parametri, allora, la probabilità di utilizzo del metodo con una particolare combinazione è pari ad  $1/X$ .

La reliability è stata ottenuta partendo dal valore 1 e, per ogni combinazione di parametri del profilo operazionale che ha evidenziato fallimenti dei test, vi è stato sottratto  $1/X$ .

I risultati ottenuti sono visibili in [Tabella 17](#).

**Nota:** i test riguardanti la classe *BookKeeper*, come già specificato, prevedono l'istanziamento e il teardown di un nuovo server prima e dopo ogni run di test. Ciò comporta la necessità di un ingente quantitativo di tempo per l'esecuzione dei test.

## Seconda classe: NetworkTopologyImpl

Come descritto dalla [Javadoc](#), la classe *NetworkTopologyImpl* implementa l'interfaccia *NetworkTopology* e rappresenta un cluster di computer organizzati in una topologia di rete gerarchica ad albero. Ad esempio, un cluster può consistere in diversi data centers in cui sono presenti racks di computer. Le foglie rappresentano **data nodes** (computer), mentre gli **inner nodes** rappresentano switch o router che gestiscono il traffico in ingresso e in uscita dai data centers o dai rack.

Per quanto riguarda un generico nodo, sempre dalla [documentazione](#), abbiamo la seguente descrizione:

"A node may be a leaf representing a data node or an inner node representing a datacenter or rack. Each data has a name and its location in the network is decided by a string with syntax similar to a file name. For example, a data node's **name** is **hostname:port#** and if it's located at rack "orange" in datacenter "dog", the string representation of its network **location** is **/dog/orange**."

Il primo metodo che si decide di testare è il metodo *add()*, in quanto ritenuto essere il metodo fondamentale, per importanza e funzionalità che rappresenta, esposto dalla classe. Esso prende in input un unico parametro, di tipo *Node* (interfaccia) e, come [documentato](#), implementa la funzionalità di aggiungere un nodo foglia all'albero della topologia di rete. Viene specificato che il nodo passato come parametro può essere *null* e che ci si aspetta una *IllegalArgumentException* in due specifici casi:

- Se si tenta di aggiungere un nodo ad un rack che è un nodo foglia (si può aggiungere solo ad inner nodes)
- Se il nodo che si tenta di aggiungere non è una foglia (è esso stesso un inner node)

Dunque, si individuano per il parametro *node* le seguenti classi di equivalenza:

- *node*: {*DataNode*}, {*InnerNode*}, {*null*}, {*DataNode* aggiunto ad un rack non valido}

Assumendo che le classi che implementano *Node* abbiano un costruttore del tipo *Node(name, networkLocation)*, si usa tale notazione per indicare i valori di boundary individuati:

- *node*: *null*, *DataNode*("127.0.0.1:4000", "/root"), *InnerNode*("127.0.0.1:4001", "/inner"), *DataNode*("127.0.0.1:4002", "/rack/127.0.0.1:3999")

Per eseguire il caso di test con l'ultimo boundary-value indicato, è necessario prima inserire nella topologia della rete il **DataNode**("127.0.0.1:3999", "/rack").

Dall'analisi effettuata, si procede individuando i casi di test da implementare, descritti in [Tabella 13](#).

Affinché il test abbia successo, è necessario controllare che il nodo sia stato effettivamente inserito nella network topology nei casi in cui non è attesa un'eccezione.

A tal proposito, parallelamente al test di unità per il metodo *add()*, si è deciso di progettare e sviluppare un test d'integrazione per accertarsi che il SUT (la classe *NetworkTopologyImpl*) agisca realmente sull'istanza di nodo passata come parametro al metodo. L'interfaccia *Node* dichiara l'operazione *setParent(Node)*. Ci si aspetta che, durante l'esecuzione della *add()*, il metodo implementante l'operazione dichiarata dall'interfaccia venga invocato esattamente 1 volta sull'istanza di *Node* passata come parametro al metodo *add()*. Ovviamente, ciò è da considerare solo in caso di aggiunta di un nodo che vada a buon fine.

Quindi, è stato realizzato un mock spia sull'istanza di *Node* passata al metodo *add()*, tramite l'utilizzo di *Mockito*, verificando che il metodo *setParent()* venisse invocato esattamente 1 volta ([Codice 6](#)).

Nella prima iterazione, si decide di testare anche il metodo *remove(Node)*. Ovviamente, esso prende come parametro in input il nodo da rimuovere dalla topologia di rete. Come specificato dalla [documentazione](#), il nodo può essere *null*. Anche se non viene indicato, ci si aspetta che non sia possibile la rimozione di un *InnerNode*, in quanto non ne è possibile nemmeno l'aggiunta con il metodo *add()*. In tal caso, l'output atteso è un'eccezione, molto probabilmente di tipo *IllegalArgumentExcepion*, per analogia col metodo *add()*.

Le classi di equivalenza per *node* sono:

- *node*: {*null*}, {*DataNode*}, {*InnerNode*}

I boundary-values diventano: *null*, *DataNode*("127.0.0.1:4000", "/root"), *InnerNode*("127.0.0.1:4001", "/inner").

I casi di test progettati sono elencati in [Tabella 14](#).

La test-suite implementata ha permesso di ottenere una statement coverage di classe pari al 35% ed una branch coverage pari al 25% ([Figura 37](#)). La copertura dei singoli metodi, ottenuta sempre con JaCoCo, è consultabile in [Figura 38](#), mentre, per la mutation coverage, si ha un valore del 15% ([Figura 39](#)). Il test di integrazione risulta essere passato, come visibile dal report generato dal plugin *Failsafe* ([Figura 40](#)).

Come già descritto, per il metodo *add()*, è attesa un'eccezione di tipo *IllegalArgumentExcepion* quando si cerca di aggiungere un nodo foglia ad un nodo che non sia un *InnerNode*. In realtà, l'aggiunta del nodo risulta comunque impossibile, ma l'eccezione lanciata è di tipo *InvalidTopologyException*. In effetti, come visibile dalla coverage generata da JaCoCo ([Figura 41](#)), l'if-statement a riga 422, che dovrebbe triggerare l'eccezione attesa, non viene coperto, poiché il flusso di esecuzione soddisfa la condizione dell'if-statement precedente (riga 416 in figura; risulta comunque non coperta perché il test che portava al fallimento è stato disabilitato).

Valutando la copertura del metodo *add()* anche seguendo un approccio **data-flow oriented** ([Figura 42](#)), con il supporto del tool *ba-dua*, risultano esserci delle coppie def-use non coperte che è possibile coprire. Dunque, si decide di aggiungere un ulteriore caso di test nella seconda iterazione, così da coprire anche il lancio dell'eccezione di tipo *InvalidTopologyException*.

Nella seconda iterazione, vengono testati altri due metodi della classe; in particolare, *isOnSameRack()* e *getDistance()*. Entrambi prendono in input due parametri di tipo *Node*. Da [documentazione](#), per il metodo *isOnSameRack*, i parametri possono essere *null* e viene ritornato *true* se i nodi sono nello stesso rack, falso altrimenti. Ci si aspetta *IllegalArgumentExcepion* se almeno uno dei due nodi è *null* oppure non è appartenente al cluster della network topology. Pertanto, le classi di equivalenza risultano essere:

- *node1*: {*null*}, {aggiunto in rack}, {non aggiunto in rack}
- *node2*: {*null*}, {aggiunto nello stesso rack}, {aggiunto in un rack diverso}, {non aggiunto in rack}

I boundary-values identificati sono:

- *node1*: *null*, *DataNode* aggiunto in rack, *DataNode* non aggiunto in rack
- *node2*: *null*, *DataNode* aggiunto nello stesso rack di *node1*, *DataNode* aggiunto in un rack differente da quello di *node1*, *DataNode* non aggiunto in rack

Considerando un approccio multidimensionale tra i due parametri ed eliminando le combinazioni ritenute non necessarie da testare, i casi di test progettati ed implementati sono descritti in [Tabella 15](#).

Il metodo *getDistance()*, come descritto dalla [Javadoc](#), calcola la distanza tra due nodi nella rete, considerando la distanza tra un nodo ed il suo genitore pari ad 1. Il valore ritornato è la somma delle distanze dei singoli nodi dal loro antenato comune più vicino. Se i due nodi passati come parametro sono lo stesso nodo, viene ritornato 0; invece, se almeno uno dei due nodi non appartiene al cluster, viene ritornato *Integer.MAX\_VALUE*. Questa volta, a differenza di tutti gli altri casi nella documentazione, non viene esplicitamente indicato che i parametri possano essere *null*, quindi, si assume che l'output atteso sia un'eccezione in caso almeno uno dei due parametri sia *null*. Le classi di equivalenza definite sono:

- *node1*: {*DataNode* aggiunto al cluster}, {*DataNode* non aggiunto al cluster}, {*null*}
- *node2*: {=*node1* aggiunto al cluster}, {*DataNode* aggiunto al cluster con location ≠ *node1*}, {*DataNode* non aggiunto al cluster}, {*null*}

I boundary-values sono:

- *node1*: *DataNode* aggiunto, *DataNode* non aggiunto, *null*
- *node2*: *DataNode* aggiunto con stessa location di *node1*, *DataNode* aggiunto con location diversa da *node1*, *DataNode* non aggiunto, *null*

Adottando anche in questo caso un approccio multidimensionale, ma scartando le combinazioni ritenute sovrabbondanti, i test-cases individuati sono elencati in [Tabella 16](#).

Anche per i metodi testati nella nuova iterazione, bisogna registrare delle incongruenze tra comportamento atteso e comportamento effettivo:

- *isOnSameRack()*: se uno dei due parametri (o entrambi) sono *null*, ci si attendeva *IllegalArgumentException*, ma il metodo ritorna *false*. Inoltre, la stessa eccezione era prevista se almeno uno dei due nodi non appartenesse al cluster, ma, in realtà, non si ha mai tale eccezione. Guardando il codice sorgente del metodo ([Codice 7](#)), è evidente come ciò che è scritto nella documentazione, per lo meno per quanto riguarda i valori *null*, non è assolutamente rispettato.
- *getDistance()*: se uno dei nodi non è aggiunto al cluster, era atteso in output il valore *Integer.MAX\_VALUE*; invece, viene in realtà calcolata la distanza tra i nodi come se fossero tutti aggiunti al cluster. Inoltre, l'assunzione sul lancio di un'eccezione quando almeno uno dei due parametri fosse stato *null*, si è rivelata vera solo in parte: se solo uno dei due è *null*, si ha una *NullPointerException*, altrimenti, se entrambi sono *null*, viene ritornato 0, poiché i nodi sono considerati essere lo stesso nodo. Tuttavia, questo approccio è discutibile e si raccomanda comunque di chiarire maggiormente sulla documentazione il risultato da aspettarsi in tali situazioni.

La seconda iterazione di test ha evidenziato una migliore adeguatezza della test-suite in termini di copertura, sia della singola classe che dei metodi. In particolare, come visibile dalla [Figura 43](#) e dalla [Figura 44](#), la statement coverage della classe è salita al 48% e la branch coverage al 39%; inoltre, la copertura del metodo *add()* è aumentata. Infatti, dal report di JaCoCo ([Figura 45](#)), il blocco di codice che lanciava l'eccezione *InvalidTopologyException* è stato coperto e, dal report di *ba-dua* ([Figura 46](#)), si ottiene che sono state coperte 7 coppie def-use in più nel metodo, rispetto all'iterazione precedente. Infine, anche la mutation coverage è aumentata, raggiungendo un valore pari al 32% ([Figura 47](#)).

Per i **profili operazionali** e la stima della **reliability**, è stato adottato lo stesso approccio avuto per la classe *BookKeeper*. I risultati sono mostrati in [Tabella 18](#).

Anche per il progetto *BookKeeper*, lo sviluppo dei test è stato incluso nel ciclo di build del progetto utilizzando le GitHub Actions ([link repository GitHub](#)) ed è stato inoltre integrato con [SonarCloud](#), rendendo possibile l'analisi della coverage ottenuta tramite JaCoCo. I tool *ba-dua* e *PIT* sono stati configurati nel progetto, ma utilizzati solo in locale.



# Risorse

```
public class JSONPatchTest_0 extends TestCase {

    public void test_for_multi_0() throws Exception {
        String original = "{\n" +
            "  \"baz\": \"qux\",\n" +
            "  \"foo\": \"bar\"\n" +
            "}";

        String patch = "[\n" +
            "  { \"op\": \"replace\", \"path\": \"/baz\", \"value\": \"boo\" },\n" +
            "  { \"op\": \"add\", \"path\": \"/hello\", \"value\": [\"world\"] },\n" +
            "  { \"op\": \"remove\", \"path\": \"/foo\" }\n" +
            "];";

        String result = JSONPatch.apply(original, patch);
        assertEquals("{\"baz\":\"boo\",\"hello\":[\"world\"]}", result);
    }

    public void test_for_add_1() throws Exception {
        String original = "{}";

        String patch = "{ \"op\": \"add\", \"path\": \"/a/b/c\", \"value\": [ \"foo\", \"bar\" ] }";

        String result = JSONPatch.apply(original, patch);
        assertEquals("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}", result);
    }

    public void test_for_remove_0() throws Exception {
        String original = "{}";

        String patch = "{ \"op\": \"remove\", \"path\": \"/a/b/c\" }";

        String result = JSONPatch.apply(original, patch);
        assertEquals "{}", result);
    }

    public void test_for_remove_1() throws Exception {
        String original = "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}";

        String patch = "{ \"op\": \"remove\", \"path\": \"/a/b/c\" }";

        String result = JSONPatch.apply(original, patch);
        assertEquals("{\"a\":{\"b\":{\"c\":[]}}}", result);
    }

    public void test_for_replace_1() throws Exception {
        String original = "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}";

        String patch = "{ \"op\": \"replace\", \"path\": \"/a/b/c\", \"value\": 42 }";

        String result = JSONPatch.apply(original, patch);
        assertEquals("{\"a\":{\"b\":{\"c\":42}}}", result);
    }

    public void test_for_move_0() throws Exception {
        String original = "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}";

        String patch = "{ \"op\": \"move\", \"from\": \"/a/b/c\", \"path\": \"/a/b/d\" }";

        String result = JSONPatch.apply(original, patch);
        assertEquals("{\"a\":{\"b\":{\"d\":[\"foo\",\"bar\"]}}}"; result);
    }

    public void test_for_copy_0() throws Exception {
        String original = "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}";

        String patch = "{ \"op\": \"copy\", \"from\": \"/a/b/c\", \"path\": \"/a/b/e\" }";

        String result = JSONPatch.apply(original, patch);
        assertEquals("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"],\"e\":[\"foo\",\"bar\"]}}}"; result);
    }

    public void test_for_test_0() throws Exception {
```

```

String original = "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}";

String patch = "{ \"op\": \"test\", \"path\": \"/a/b/c\", \"value\": \"foo\" }";

String result = JSONPatch.apply(original, patch);
assertEquals("false", result);
}
}

```

*Codice 1 Codice originale della classe JSONPatchTest\_0.java*

```

public class JSONPatchTest_0 {

    /* Instance under test is not needed, because the tested method is static*/
    private String original;
    private String patch;
    private String expected;

    public JSONPatchTest_0() {
        configureTestClass("{\"\n" + "  \"baz\": \"qux\",\n" + "  \"foo\": \"bar\"\n" + "}",
            "[\n" +
                "    { \"op\": \"replace\", \"path\": \"\/baz\", \"value\": \"boo\" },\n" +
                "    { \"op\": \"add\", \"path\": \"\/hello\", \"value\": [\"world\"] },\n" +
                "    { \"op\": \"remove\", \"path\": \"\/foo\" }\n" + "]" ,
            "{\"baz\": \"boo\", \"hello\": [\"world\"]}");

        // configureTestClass("{", "{ \"op\": \"add\", \"path\": \"\/a/b/c\", \"value\": [ \"foo\", \"bar\" ] }",
        //     "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}}");
        //
        // configureTestClass("{", "{ \"op\": \"remove\", \"path\": \"\/a/b/c\" }\n",
        //     "{}");
        //
        // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}} ", "{ \"op\": \"remove\", \"path\": \"\/a/b/c\" }\n",
        //     "{\"a\":{\"b\":{\"c\":[]}}}");
        //
        // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}} ", "{ \"op\": \"replace\", \"path\": \"\/a/b/c\", \"value\": 42 }",
        //     "{\"a\":{\"b\":{\"c\":42}}}");
        //
        // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}} ", "{ \"op\": \"move\", \"from\": \"\/a/b/c\", \"path\": \"\/a/b/d\" }",
        //     "{\"a\":{\"b\":{\"d\":[\"foo\",\"bar\"]}}}");
        //
        // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}} ", "{ \"op\": \"copy\", \"from\": \"\/a/b/c\", \"path\": \"\/a/b/e\" }",
        //     "{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"],\"e\":[\"foo\",\"bar\"]}}}");
        //
        // configureTestClass("{\"a\":{\"b\":{\"c\":[\"foo\",\"bar\"]}}} ", "{ \"op\": \"test\", \"path\": \"\/a/b/c\", \"value\": \"foo\" }",
        //     "false");
        //
    }

    private void configureTestClass(String original, String patch, String expected) {
        this.original = original;
        this.patch = patch;
        this.expected = expected;
    }

    @Test
    public void testJSONPatch() {
        String result = JSONPatch.apply(original, patch);
        Assert.assertEquals(expected, result);
    }
}

```

*Codice 2 Classe JSONPatchTest\_0.java con test parametrico*

JaCoCo Coverage Report				
Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">com.alibaba.fastjson.parser</a>		12%		6%
<a href="#">com.alibaba.fastjson.serializer</a>		7%		3%
<a href="#">com.alibaba.fastjson.util</a>		17%		6%
<a href="#">com.alibaba.fastjson</a>		7%		4%
<a href="#">com.alibaba.fastjson.parser.deserializer</a>		27%		13%
<a href="#">com.alibaba.fastjson.support.spring</a>		0%		0%
<a href="#">com.alibaba.fastjson.asm</a>		54%		35%
<a href="#">com.alibaba.fastjson.support.jaxrs</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.hsf</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.retrofit</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.geo</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.spring.messaging</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.config</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.moneta</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.springfox</a>		0%		n/a
Total	108.577 of 126.842	14%	17.168 of 18.420	6%

Figura 1 Coverage eseguendo solo JSONPatchTest\_0 con una patch da 3 operazioni

JSONPatch	51%	38%
-----------	-----	-----

Figura 2 Coverage della classe JSONPatch con una patch da 3 operazioni

Element	Missed Instructions	Cov.	Missed Branches	Cov.
apply(Object, String)		42%		33%
isObject(String)		68%		50%
JSONPatch()		0%		n/a
apply(String, String)		100%		n/a
Total	85 of 174	51%	16 of 26	38%

Figura 3 Coverage dei metodi della classe JSONPatch con una patch da 3 operazioni

JaCoCo Coverage Report				
Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">com.alibaba.fastjson.parser</a>		11%		5%
<a href="#">com.alibaba.fastjson.serializer</a>		6%		2%
<a href="#">com.alibaba.fastjson.util</a>		17%		6%
<a href="#">com.alibaba.fastjson</a>		7%		4%
<a href="#">com.alibaba.fastjson.parser.deserializer</a>		26%		12%
<a href="#">com.alibaba.fastjson.support.spring</a>		0%		0%
<a href="#">com.alibaba.fastjson.asm</a>		54%		35%
<a href="#">com.alibaba.fastjson.support.jaxrs</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.hsf</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.retrofit</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.geo</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.spring.messaging</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.config</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.moneta</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.springfox</a>		0%		n/a
Total	108.830 of 126.842	14%	17.217 of 18.420	6%

Figura 4 Coverage ottenuta eseguendo solo JSONPatchTest\_0 con una patch da una singola operazione




 <b>JSONPatch</b>	 <b>58%</b>	 <b>38%</b>
--	--	--

Figura 5 Copertura della classe JSONPatch a seguito dell'esecuzione di un test con una patch da una singola operazione







Element	Missed Instructions	Cov.	Missed Branches	Cov.
● apply(Object, String)		51%		33%
● isObject(String)		72%		50%
● JSONPatch()		0%		n/a
● apply(String, String)		100%		n/a
Total	73 of 174	58%	16 of 26	38%

Figura 6 Coverage dei metodi della classe JSONPatch in seguito al test con patch con singola operazione

```
public class MapTest2 extends TestCase {

    public void test_map () throws Exception {
        Map<Object, Object> map = JSON.parseObject("{1:\\\"2\\\",\\\"3\\\":4,'5':6}", new TypeRefer-
ence<Map<Object, Object>>() {});
        Assert.assertEquals("2", map.get(1));
        Assert.assertEquals(4, map.get("3"));
        Assert.assertEquals(6, map.get("5"));
    }
}
```

Codice 3 Codice originale della classe MapTest2

```
public class MapTest2 {

    /* The instance to be tested is not needed because the tested method is static*/

    private String jsonObject;
    private TypeReference<Map<Object, Object>> typeReference;

    public MapTest2 () {
        configureTestClass("{1:\\\"2\\\",\\\"3\\\":4,'5':6}");
    }

    private void configureTestClass(String jsonObject) {
        this.jsonObject = jsonObject;
        this.typeReference = new TypeReference<>() {};
    }

    /**
     * This test method invokes the <i>parseObject</i> static method of the JSON class.
     * In particular, it checks that the conversion from the string representation
     * of the jsonObject to a Map between keys and values is well done.
     * The test is passed only if each value is associated to the correct key.
     */
    @Test
    public void test_map () {
        Map<Object, Object> map = JSON.parseObject(this.jsonObject, this.typeReference);

        // Use an oracle to compute the keys and the expected values bounded with those keys
        Oracle oracle = new Oracle();
        Object[] keys = oracle.getKeys(jsonObject);
        Object[] expected = oracle.getExpectedResults(jsonObject, keys);

        boolean passed = true;
        for (int i = 0; i < keys.length; i++) {
            if (map.get(keys[i]) == null) {
                if (! (map.get(keys[i]) == expected[i])) {
                    passed = false;
                    break;
                }
            }
            else if (!map.get(keys[i]).equals(expected[i])) {
                passed = false;
                break;
            }
        }
        Assert.assertTrue(passed);
    }
}
```



```

private static class Oracle{
    /**
     * This private method is used as a Software Testing oracle to know the
     * expected values of the JSON Object referred by the key in the input array of keys.
     * It's an oracle because it uses the library <b>org.json</b> as a trusted entity.
     *
     * @param jsonObject The string representing the JSONObject of the FastJSON project
     * @param keys The array of the objects representing the keys of the JSON Object
     * @return an array of Object, containing the values expected to be contained in the
jsonObject,
     * in the same order of their keys
     */
    private Object[] getExpectedResults(String jsonObject, Object[] keys){
        JSONObject obj = new JSONObject(jsonObject);
        List<Object> expected = new LinkedList<>();
        for (int i = 0; i < keys.length; i++){
            expected.add(i, obj.get(String.valueOf(keys[i])));
        }
        return expected.toArray();
    }

    /**
     * This private method is used to retrieve the array of keys
     * of the json object string representation in input.
     * @param jsonObject The string representing the JSONObject of the FastJSON project
     * @return an array of Objects, containing the keys of the JSONObject.
     */
    private Object[] getKeys(String jsonObject){
        String noBrackets = jsonObject.replace("{", "");
        noBrackets = noBrackets.replace("}", "");
        noBrackets = noBrackets.replace(" ", "");
        noBrackets = noBrackets.replace("\n", "");
        noBrackets = noBrackets.replace("\r", "");
        noBrackets = noBrackets.replace("\t", "");

        String[] pairs = noBrackets.split(",");

        List<Object> ret = new LinkedList<>();
        for (int i = 0; i < pairs.length; i++){
            String key = pairs[i].split(":")[0];
            if (key.contains("\"") || key.contains("'')){
                ret.add(i, key.replace("\"", "").replace("'", ""));
            }
            else if (key.contains("."))
                ret.add(i, Float.valueOf(key));
            else
                ret.add(i, Integer.valueOf(key));
        }
        return ret.toArray();
    }
}
}

```

*Codice 4 Classe MapTest2.java con test parametrico e inner class Oracle*

Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">com.alibaba.fastjson.serializer</a>		1%		0%
<a href="#">com.alibaba.fastjson.parser</a>		8%		2%
<a href="#">com.alibaba.fastjson.util</a>		12%		2%
<a href="#">com.alibaba.fastjson.parser.deserializer</a>		2%		1%
<a href="#">com.alibaba.fastjson</a>		1%		0%
<a href="#">com.alibaba.fastjson.asm</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.spring</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.jaxrs</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.hsf</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.retrofit</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.geo</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.spring.messaging</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.config</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.moneta</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.springfox</a>		0%		n/a
Total	119.924 of 126.842	5%	18.158 of 18.420	1%

Figura 7 Coverage ottenuta eseguendo il solo test della classe MapTest2.java

<a href="#">JSON</a>		9%		4%
----------------------	--	----	--	----

Figura 8 Class coverage della classe JSON in seguito al test di MapTest2.java

Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">com.alibaba.fastjson.parser</a>		13%		7%
<a href="#">com.alibaba.fastjson.serializer</a>		7%		3%
<a href="#">com.alibaba.fastjson.util</a>		17%		6%
<a href="#">com.alibaba.fastjson</a>		7%		4%
<a href="#">com.alibaba.fastjson.parser.deserializer</a>		28%		15%
<a href="#">com.alibaba.fastjson.support.spring</a>		0%		0%
<a href="#">com.alibaba.fastjson.asm</a>		54%		35%
<a href="#">com.alibaba.fastjson.support.jaxrs</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.hsf</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.retrofit</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.geo</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.spring.messaging</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.config</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.moneta</a>		0%		0%
<a href="#">com.alibaba.fastjson.support.springfox</a>		0%		n/a
Total	107.981 of 126.842	14%	17.081 of 18.420	7%

Figura 9 Coverage in seguito all'esecuzione dell'intera test-suite

Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">JSONPath</a>		7%		5%
<a href="#">JSONPath.JSONPathParser</a>		7%		5%
<a href="#">JSON</a>		14%		7%
<a href="#">JSONObject</a>		6%		5%

Figura 10 Coverage della classe JSON aumentata con l'esecuzione di tutti i test della test-suite

```

public String generate(final List<PasswordPolicy> policies) throws InvalidPasswordRuleConf {
    List<DefaultPasswordRuleConf> defaultRuleConfs = new ArrayList<>();

    policies.stream().forEach(policy -> policy.getRules().forEach(impl -> {
        try {
            ImplementationManager.buildPasswordRule(impl).ifPresent(rule -> {
                if (rule.getConf() instanceof DefaultPasswordRuleConf) {
                    defaultRuleConfs.add((DefaultPasswordRuleConf) rule.getConf());
                }
            });
        } catch (Exception e) {
            LOG.error("Invalid {}, ignoring...", impl, e);
        }
    }));

    DefaultPasswordRuleConf ruleConf = merge(defaultRuleConfs);
    check(ruleConf);
    return generate(ruleConf);
}

```

Codice 5 metodo generate(List<PasswordPolicy>) di DefaultPasswordGenerator

policies	Expected output
Empty list	Password conforme alle policies di default
List with 1 valid policy	Password conforme alla policy indicata
List with 1 invalid policy	Eccezione
null	Eccezione

Tabella 1 Test cases iterazione 1 metodo generate(List<PasswordPolicy>) di DefaultPasswordGenerator

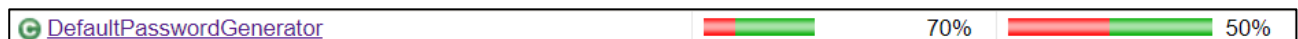


Figura 11 Class coverage DefaultPasswordGenerator (iterazione 1)

DefaultPasswordGenerator				
Element	Missed Instructions	Cov.	Missed Branches	Cov.
check(DefaultPasswordRuleConf)		42%		44%
checkRequired(String[], DefaultPasswordRuleConf)		51%		50%
checkEndChar(String[], DefaultPasswordRuleConf)		60%		64%
checkStartChar(String[], DefaultPasswordRuleConf)		59%		50%
generate(ExternalResource)		0%		0%
lambda\$checkPrefixAndSuffix\$5(String[], DefaultPasswordRuleConf, String)		0%		0%
lambda\$checkPrefixAndSuffix\$4(String[], DefaultPasswordRuleConf, String)		0%		0%
lambda\$generate\$1(List, Implementation)		53%		n/a
lambda\$merge\$3(DefaultPasswordRuleConf, DefaultPasswordRuleConf)		97%		50%
merge(List)		89%		75%
generate(DefaultPasswordRuleConf)		100%		100%
generate(List)		100%		n/a
checkPrefixAndSuffix(String[], DefaultPasswordRuleConf)		100%		n/a
firstEmptyChar(String[])		100%		100%
lambda\$generate\$0(List, PasswordRule)		100%		50%
lambda\$generate\$2(List, PasswordPolicy)		100%		n/a
static {...}		100%		n/a
DefaultPasswordGenerator()		100%		n/a
Total	183 of 629	70%	68 of 138	50%

Figura 12 Method coverage DefaultPasswordGenerator (iterazione 1)





```

83     protected static DefaultPasswordRuleConf merge(final List<DefaultPasswordRuleConf> defaultRuleConfs) {
84         DefaultPasswordRuleConf result = new DefaultPasswordRuleConf();
85         result.setMinLength(VERY_MIN_LENGTH);
86         result.setMaxLength(VERY_MAX_LENGTH);
87
88         defaultRuleConfs.forEach(ruleConf -> {
89             if (ruleConf.getMinLength() > result.getMinLength()) {
90                 result.setMinLength(ruleConf.getMinLength());
91             }
92
93             if ((ruleConf.getMaxLength() != 0) && ((ruleConf.getMaxLength() < result.getMaxLength())) {
94                 result.setMaxLength(ruleConf.getMaxLength());
95             }
96             result.getPrefixesNotPermitted().addAll(ruleConf.getPrefixesNotPermitted());
97             result.getSuffixesNotPermitted().addAll(ruleConf.getSuffixesNotPermitted());
98
99             if (!result.isNonAlphanumericRequired()) {
100                 result.setNonAlphanumericRequired(ruleConf.isNonAlphanumericRequired());
101             }
102
103             if (!result.isAlphanumericRequired()) {
104                 result.setAlphanumericRequired(ruleConf.isAlphanumericRequired());
105             }
106             if (!result.isDigitRequired()) {
107                 result.setDigitRequired(ruleConf.isDigitRequired());
108             }
109
110             if (!result.isLowercaseRequired()) {
111                 result.setLowercaseRequired(ruleConf.isLowercaseRequired());
112             }
113             if (!result.isUppercaseRequired()) {
114                 result.setUppercaseRequired(ruleConf.isUppercaseRequired());
115             }
116             if (!result.isMustStartWithDigit()) {
117                 result.setMustStartWithDigit(ruleConf.isMustStartWithDigit());
118             }
119             if (!result.isMustntStartWithDigit()) {
120                 result.setMustntStartWithDigit(ruleConf.isMustntStartWithDigit());
121             }
122             if (!result.isMustEndWithDigit()) {
123                 result.setMustEndWithDigit(ruleConf.isMustEndWithDigit());
124             }
125             if (result.isMustntEndWithDigit()) {
126                 result.setMustntEndWithDigit(ruleConf.isMustntEndWithDigit());
127             }
128             if (!result.isMustStartWithAlpha()) {
129                 result.setMustStartWithAlpha(ruleConf.isMustStartWithAlpha());

```

Figura 15 Bug nella funzione merge() visto dal report di PIT. Evidenziato l'if-statement incriminato



Figura 16 JaCoCo class coverage DefaultPasswordGenerator (iterazione 2)

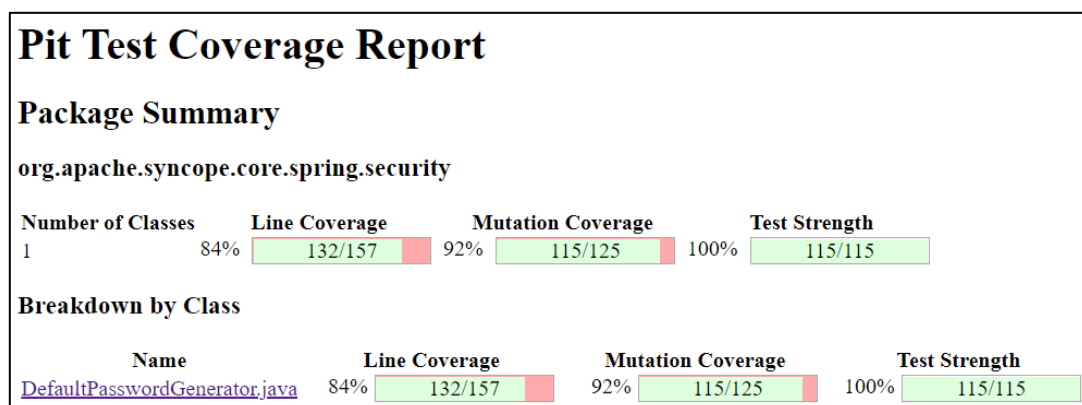


Figura 17 PIT mutation coverage su DefaultPasswordGenerator (iterazione 2)

DefaultPasswordGenerator					
Element	Missed Instructions	Cov.	Missed Branches	Cov.	
<a href="#">generate(ExternalResource)</a>		0%		0%	
<a href="#">check(DefaultPasswordRuleConf)</a>		86%		94%	
<a href="#">lambda\$checkPrefixAndSuffix\$5(String[], DefaultPasswordRuleConf, String)</a>		0%		0%	
<a href="#">checkEndChar(String[], DefaultPasswordRuleConf)</a>		90%		92%	
<a href="#">checkRequired(String[], DefaultPasswordRuleConf)</a>		90%		75%	
<a href="#">lambda\$generate\$1(List, Implementation)</a>		53%		n/a	
<a href="#">lambda\$merge\$3(DefaultPasswordRuleConf, DefaultPasswordRuleConf)</a>		97%		54%	
<a href="#">lambda\$checkPrefixAndSuffix\$4(String[], DefaultPasswordRuleConf, String)</a>		66%		50%	
<a href="#">checkStartChar(String[], DefaultPasswordRuleConf)</a>		100%		92%	
<a href="#">generate(DefaultPasswordRuleConf)</a>		100%		100%	
<a href="#">merge(List)</a>		100%		100%	
<a href="#">generate(List)</a>		100%		n/a	
<a href="#">checkPrefixAndSuffix(String[], DefaultPasswordRuleConf)</a>		100%		n/a	
<a href="#">firstEmptyChar(String[])</a>		100%		100%	
<a href="#">lambda\$generate\$0(List, PasswordRule)</a>		100%		50%	
<a href="#">lambda\$generate\$2(List, PasswordPolicy)</a>		100%		n/a	
<a href="#">static {...}</a>		100%		n/a	
<a href="#">DefaultPasswordGenerator()</a>		100%		n/a	
Total	65 of 629	89%	33 of 138	76%	

Figura 18 JaCoCo method coverage su DefaultPasswordGenerator (iterazione 2)

```

50     @Transactional(readonly = true)
51     @Override
52     public String generate(final ExternalResource resource) throws InvalidPasswordRuleConf {
53         List<PasswordPolicy> policies = new ArrayList<>();
54
55         if (resource.getPasswordPolicy() != null) {
56             policies.add(resource.getPasswordPolicy());
57         }
58
59         return generate(policies);
60     }

```

Figura 19 PIT: mutanti non uccisi nel metodo generate(ExternalResource)

<a href="#">DefaultPasswordGenerator</a>		92%		77%
--	--	-----	--	-----

Figura 20 JaCoCo class coverage DefaultPasswordGenerator (iterazione 3)

## Pit Test Coverage Report

### Package Summary

org.apache.syncope.core.spring.security

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	87%	94%	100%

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">DefaultPasswordGenerator.java</a>	87%	94%	100%

Figura 21 PIT: mutation coverage incrementata sulla classe DefaultPasswordGenerator (iterazione 3)

value	cipherAlgorithm	expected output
null	SHA-1	Stringa valida
""	SHA-1	Stringa valida
"ab"	SHA-1	Stringa valida
Stringa da 129 char (8 bit per char)	SHA-1	Stringa valida
null	SHA-256	Stringa valida
""	SHA-256	Stringa valida
"ab"	SHA-256	Stringa valida
Stringa da 129 char (8 bit per char)	SHA-256	Stringa valida
null	SHA-512	Stringa valida
""	SHA-512	Stringa valida
"ab"	SHA-512	Stringa valida
Stringa da 129 char (8 bit per char)	SHA-512	Stringa valida
null	AES	Stringa valida
""	AES	Stringa valida
"ab"	AES	Stringa valida
Stringa da 129 char (8 bit per char)	AES	Stringa valida
null	S-MD5	Stringa valida
""	S-MD5	Stringa valida
"ab"	S-MD5	Stringa valida
Stringa da 129 char (8 bit per char)	S-MD5	Stringa valida
null	S-SHA-1	Stringa valida
""	S-SHA-1	Stringa valida
"ab"	S-SHA-1	Stringa valida
Stringa da 129 char (8 bit per char)	S-SHA-1	Stringa valida
null	S-SHA-256	Stringa valida
""	S-SHA-256	Stringa valida
"ab"	S-SHA-256	Stringa valida
Stringa da 129 char (8 bit per char)	S-SHA-256	Stringa valida
null	S-SHA-512	Stringa valida
""	S-SHA-512	Stringa valida
"ab"	S-SHA-512	Stringa valida
Stringa da 129 char (8 bit per char)	S-SHA-512	Stringa valida
null	BCRYPT	Stringa valida
""	BCRYPT	Stringa valida
"ab"	BCRYPT	Stringa valida
Stringa da 129 char (8 bit per char)	BCRYPT	Stringa valida

Tabella 3 Test cases per il metodo encode() della classe Encryptor

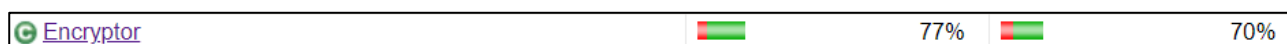


Figura 22 JaCoCo class coverage della classe Encryptor (iterazione 1)

## Encryptor

Element	Missed Instructions	Cov.	Missed Branches	Cov.
● <a href="#">Encryptor(String)</a>	<div><div></div></div>	50%	<div><div></div></div>	50%
● <a href="#">decode(String, CipherAlgorithm)</a>	<div><div></div></div>	0%	<div><div></div></div>	0%
● <a href="#">verify(String, CipherAlgorithm, String)</a>	<div><div></div></div>	87%	<div><div></div></div>	75%
● <a href="#">getInstance(String)</a>	<div><div></div></div>	96%	<div><div></div></div>	75%
● <a href="#">getDigester(CipherAlgorithm)</a>	<div><div></div></div>	100%	<div><div></div></div>	100%
● <a href="#">encode(String, CipherAlgorithm)</a>	<div><div></div></div>	100%	<div><div></div></div>	87%
● <a href="#">static {...}</a>	<div><div></div></div>	100%		n/a
● <a href="#">getInstance()</a>	<div><div></div></div>	100%		n/a
Total	62 of 280	77%	9 of 30	70%

Figura 23 JaCoCo method coverage, classe Encryptor (iterazione 1)

## Pit Test Coverage Report

### Package Summary

org.apache.syncope.core.spring.security

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	84% <div><div></div></div> 194/231	84% <div><div></div></div> 135/161	91% <div><div></div></div> 135/148

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">DefaultPasswordGenerator.java</a>	87% <div><div></div></div> 136/157	94% <div><div></div></div> 117/125	100% <div><div></div></div> 117/117
<a href="#">Encryptor.java</a>	78% <div><div></div></div> 58/74	50% <div><div></div></div> 18/36	58% <div><div></div></div> 18/31

Figura 24 PIT: mutation coverage classe Encryptor (iterazione 1)

```

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. }

```

Figura 25 Missing branch per cipherAlgorithm == null

value	cipherAlgorithm	encoded	expected output
null	SHA-1	encode(value, cipherAlgorithm)	false
""	SHA-1	encode(value, cipherAlgorithm)	true
"ab"	SHA-1	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	SHA-1	encode(value, cipherAlgorithm)	true
null	SHA-256	encode(value, cipherAlgorithm)	false
""	SHA-256	encode(value, cipherAlgorithm)	true
"ab"	SHA-256	encode(value, cipherAlgorithm)	true



Stringa da 129 char (8 bit per char)	SHA-256	encode(value, cipherAlgorithm)	true
null	SHA-512	encode(value, cipherAlgorithm)	false
"ab"	SHA-512	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	SHA-512	encode(value, cipherAlgorithm)	true
null	AES	encode(value, cipherAlgorithm)	false
"ab"	AES	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	AES	encode(value, cipherAlgorithm)	true
null	S-MD5	encode(value, cipherAlgorithm)	false
"ab"	S-MD5	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	S-MD5	encode(value, cipherAlgorithm)	true
null	S-SHA-1	encode(value, cipherAlgorithm)	false
"ab"	S-SHA-1	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	S-SHA-1	encode(value, cipherAlgorithm)	true
null	S-SHA-256	encode(value, cipherAlgorithm)	false
"ab"	S-SHA-256	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	S-SHA-256	encode(value, cipherAlgorithm)	true
null	S-SHA-512	encode(value, cipherAlgorithm)	false
"ab"	S-SHA-512	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	S-SHA-512	encode(value, cipherAlgorithm)	true
null	BCRYPT	encode(value, cipherAlgorithm)	false
"ab"	BCRYPT	encode(value, cipherAlgorithm)	true
Stringa da 129 char (8 bit per char)	BCRYPT	encode(value, cipherAlgorithm)	true
null	null	encode(value, cipherAlgorithm)	false
"ab"	null	encode(value, cipherAlgorithm)	false
Stringa da 129 char (8 bit per char)	null	encode(value, cipherAlgorithm)	false
Stringa da 129 char (8 bit per char)	SHA-1	"randomCrap"	false
null	SHA-256	"randomCrap"	false
"ab"	SHA-256	"randomCrap"	false
Stringa da 129 char (8 bit per char)	SHA-256	"randomCrap"	false
null	SHA-512	"randomCrap"	false
"ab"	SHA-512	"randomCrap"	false
Stringa da 129 char (8 bit per char)	SHA-512	"randomCrap"	false
null	AES	"randomCrap"	false
"ab"	AES	"randomCrap"	false
Stringa da 129 char (8 bit per char)	AES	"randomCrap"	false

null	S-MD5	"randomCrap"	false
""	S-MD5	"randomCrap"	false
"ab"	S-MD5	"randomCrap"	false
Stringa da 129 char (8 bit per char)	S-MD5	"randomCrap"	false
null	S-SHA-1	"randomCrap"	false
""	S-SHA-1	"randomCrap"	false
"ab"	S-SHA-1	"randomCrap"	false
Stringa da 129 char (8 bit per char)	S-SHA-1	"randomCrap"	false
null	S-SHA-256	"randomCrap"	false
""	S-SHA-256	"randomCrap"	false
"ab"	S-SHA-256	"randomCrap"	false
Stringa da 129 char (8 bit per char)	S-SHA-256	"randomCrap"	false
null	S-SHA-512	"randomCrap"	false
""	S-SHA-512	"randomCrap"	false
"ab"	S-SHA-512	"randomCrap"	false
Stringa da 129 char (8 bit per char)	S-SHA-512	"randomCrap"	false
null	BCRYPT	"randomCrap"	false
""	BCRYPT	"randomCrap"	false
"ab"	BCRYPT	"randomCrap"	false
Stringa da 129 char (8 bit per char)	BCRYPT	"randomCrap"	false
null	null	"randomCrap"	false
""	null	"randomCrap"	false
"ab"	null	"randomCrap"	false
Stringa da 129 char (8 bit per char)	null	"randomCrap"	false

Tabella 4 Test-cases per il metodo verify() della classe Encryptor

encoded	cipherAlgorithm	expected output
null	null	null
""	null	null
encode("myString", AES)	null	null
"randomCrap"	null	null
null	AES	null
""	AES	""
encode("myString", AES)	AES	"myString"
"randomCrap"	AES	exception
null	BCRYPT	null
""	BCRYPT	null
encode("myString", AES)	BCRYPT	null
"randomCrap"	BCRYPT	null

Tabella 5 Test cases per il metodo decode() della classe Encryptor

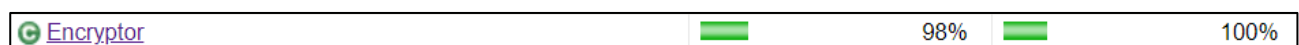


Figura 26 JaCoCo class coverage per Encryptor (iterazione 2)

Encryptor				
Element	Missed Instructions	Cov.	Missed Branches	Cov.
Encryptor(String)		91%		100%
getDigester(CipherAlgorithm)		100%		100%
encode(String, CipherAlgorithm)		100%		100%
verify(String, CipherAlgorithm, String)		100%		100%
decode(String, CipherAlgorithm)		100%		100%
getInstance(String)		100%		100%
static {...}		100%		n/a
getInstance()		100%		n/a
Total	5 of 280	98%	0 of 30	100%

Figura 27 JaCoCo method coverage per la classe Encryptor (iterazione 2)

# Pit Test Coverage Report

## Package Summary

org.apache.syncope.core.spring.security

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
2	90% <div><div></div></div> 208/231	87% <div><div></div></div> 140/161	92% <div><div></div></div> 140/153

## Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">DefaultPasswordGenerator.java</a>	87% <div><div></div></div> 136/157	94% <div><div></div></div> 117/125	100% <div><div></div></div> 117/117
<a href="#">Encryptor.java</a>	97% <div><div></div></div> 72/74	64% <div><div></div></div> 23/36	64% <div><div></div></div> 23/36

Figura 28 PIT: mutation coverage della classe Encryptor incrementata al 64% (iterazione 2)

Tipo	Parametro	Classi di equivalenza	Boundary-values
int	ensSize (E)	{<Qw}, {=Qw}, {>Qw}	Qw-1, =Qw, Qw+1
int	writeQuorumSize (Qw)	{<Qa}, {=Qa}, {>Qa}	Qa-1, =Qa, Qa+1
int	ackQuorumSize (Qa)	{<0}, {=0}, {>0}	-1, 0, 1
DigestType	digestType	{MAC}, {CRC32}, {CRC32C}, {DUMMY}	MAC, CRC32, CRC32C, DUMMY
byte[]	passwd	{array vuota}, {array non vuota}	[], "passwd".getBytes()
Map<String, byte[]>	customMetadata	{null}, {mappa vuota}, {mappa non vuota}	null, emptyMap(), Map(("myMetadata", "myCustomMetdata"))

Tabella 6 Individuazione delle classi di equivalenza e boundary-value analysis per il metodo createLedger() della classe BookKeeper

ensSize	writeQuorumSize	ackQuorumSize	digestType	passwd	customMetadata	Output atteso
3	2	1	MAC	"passwd".getBytes()	Map(("myMetadata", "myCustomMetdata"))	Handle valido
2	2	1	CRC32	"passwd".getBytes()	null	Handle valido
1	2	1	MAC	[]	null	BKException

2	1	1	MAC	"passwd".getBytes()	null	Handle valido
1	1	1	CRC32	[]	emptyMap()	Handle valido
0	1	1	MAC	"passwd".getBytes()	null	BKException
1	0	1	MAC	"passwd".getBytes()	null	BKException
0	0	1	CRC32C	"passwd".getBytes()	null	BKException
-1	0	1	MAC	"passwd".getBytes()	null	BKException
2	1	0	DUMMY	[]	emptyMap()	Handle valido
1	1	0	MAC	"passwd".getBytes()	null	Handle valido
0	1	0	MAC	"passwd".getBytes()	null	BKException
1	0	0	CRC32	"passwd".getBytes()	null	Handle valido
0	0	0	CRC32C	"passwd".getBytes()	Map(("myMeta-data", "myCustomMetdata"))	Handle valido
-1	0	0	MAC	[]	null	BKException
0	-1	0	MAC	"passwd".getBytes()	null	BKException
-1	-1	0	MAC	"passwd".getBytes()	null	BKException
-2	-1	0	MAC	[]	null	BKException
1	0	-1	MAC	"passwd".getBytes()	null	Handle valido
0	0	-1	DUMMY	"passwd".getBytes()	null	Handle valido
-1	0	-1	MAC	"passwd".getBytes()	null	BKException
0	-1	-1	MAC	"passwd".getBytes()	null	Handle valido
-1	-1	-1	MAC	[]	null	BKException
-2	-1	-1	MAC	"passwd".getBytes()	null	BKException
-1	-2	-1	MAC	"passwd".getBytes()	null	BKException
-2	-2	-1	MAC	"passwd".getBytes()	Map(("myMeta-data", "myCustomMetdata"))	BKException
-3	-2	-1	MAC	"passwd".getBytes()	null	BKException

Tabella 7 Test-cases da category partition per createLedger() di BookKeeper

lId	expected output
ledgerHandle.getId()	No exception
-1 (assunto non esistente)	Exception

Tabella 8 Test cases per deleteLedger() della classe BookKeeper

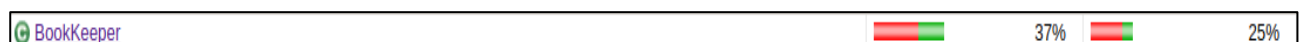


Figura 29 JaCoCo coverage: classe BookKeeper, iterazione 1









● <a href="#">asyncCreateLedger(int, int, int, BookKeeper.DigestType, byte[], AsyncCallback.CreateCallback, Object, Map)</a>	 85%	 75%
● <a href="#">createLedger(int, int, int, BookKeeper.DigestType, byte[], Map)</a>	 81%	 50%
● <a href="#">validateEventLoopGroup(EventLoopGroup)</a>	 0%	 n/a
● <a href="#">asyncDeleteLedger(long, AsyncCallback.DeleteCallback, Object)</a>	 81%	 50%

Figura 30 Jacoco coverage, method coverage per la classe BookKeeper, iterazione 1

```

941.  /**
942.   * Synchronous call to create ledger. Parameters match those of asyncCreateLedger
943.   *
944.   * @param ensSize
945.   * @param writeQuorumSize
946.   * @param ackQuorumSize
947.   * @param digestType
948.   * @param passwd
949.   * @param customMetadata
950.   * @return a handle to the newly created ledger
951.   * @throws InterruptedException
952.   * @throws BKException
953.   */
954.  public LedgerHandle createLedger(int ensSize, int writeQuorumSize, int ackQuorumSize,
955.                                  DigestType digestType, byte[] passwd, final Map<String, byte[]> customMetadata)
956.      throws InterruptedException, BKException {
957.      CompletableFuture<LedgerHandle> future = new CompletableFuture<>();
958.      SyncCreateCallback result = new SyncCreateCallback(future);
959.
960.      /*
961.       * Calls asynchronous version
962.       */
963.      asyncCreateLedger(ensSize, writeQuorumSize, ackQuorumSize, digestType, passwd,
964.                      result, null, customMetadata);
965.
966.      LedgerHandle lh = SyncCallbackUtils.waitForResult(future);
967.      if (lh == null) {
968.          LOG.error("Unexpected condition : no ledger handle returned for a success ledger creation");
969.          throw BKException.create(BKException.Code.UnexpectedConditionException);
970.      }
971.      return lh;
972.  }

```

Figura 31 Coverage del metodo createLedger() testato nella prima iterazione di test per la classe BookKeeper

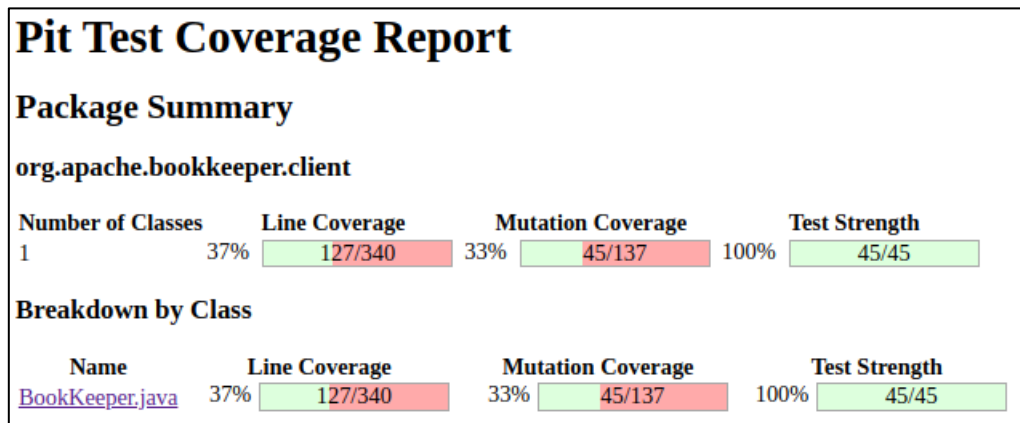


Figura 32 Mutation coverage classe BookKeeper, iterazione 1

```

2022-06-29T12:18:32,397 - WARN - [main-EventThread:AbstractZkLedgerManager$2@350] - Ledger node does not exist in ZooKeeper: ledgerId=-1. Returning success.
Expected exception, but not thrown

```

Figura 33 L'invocazione del metodo deleteLedger() su un ledger non esistente ritorna con successo anziché lanciare un'eccezione

Parametro	Classi di equivalenza	Boundary-values
ensSize	{<qSize}, {=qSize}, {>qSize}	qSize -1, =qSize, qSize +1
qSize	{<0}, {=0}, {>0}	-1, 0, 1
digestType	{MAC}, {CRC32}, {CRC32C}, {DUMMY}	MAC, CRC32, CRC32C, DUMMY
passwd	{empty array}, {non empty array}	[], "passwd".getBytes()

Tabella 9 Classi di equivalenza e boundary-values per il metodo createLedger(ensSize, qSize, digestType, passwd)

ensSize	qSize	digestType	passwd	Output atteso
-2	-1	MAC	[]	exception
-1	-1	CRC32	"passwd".getBytes()	Exception
0	-1	CRC32C	"passwd".getBytes()	Handle valido
-1	0	DUMMY	"passwd".getBytes()	Exception
0	0	MAC	[]	Handle valido



1	0	DUMMY	"passwd".getBytes()	Handle valido
0	1	MAC	"passwd".getBytes()	Exception
1	1	MAC	"passwd".getBytes()	Handle valido
2	1	MAC	"passwd".getBytes()	Handle valido

Tabella 10 Casi di test per il metodo createLedger(ensSize, qSize, digestType, passwd)

digestType	passwd	Output atteso
MAC	[]	Handle valido
CRC32	"passwd".getBytes()	Handle valido
CRC32C	"passwd".getBytes()	Handle valido
DUMMY	"passwd".getBytes()	Handle valido

Tabella 11 Casi di test per il metodo createLedger(digestType, passwd)

lId	digestType	passwd	Output atteso
ledgerHandle.getId()	MAC	"passwd".getBytes() corretta	Handle valido
ledgerHandle.getId()	CRC32	[] corretta	Handle valido
ledgerHandle.getId()	CRC32C	"wrongPassword".getBytes()	BKException
-1	DUMMY	"passwd".getBytes() corretta	BKException
-1	MAC	[] corretta	BKException
-1	CRC32	"wrongPassword".getBytes()	BKException

Tabella 12 Test cases per il metodo openLedger() della classe BookKeeper

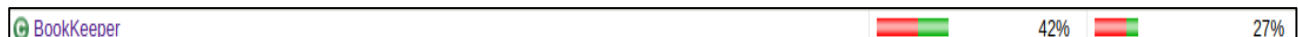


Figura 34 Class coverage JaCoCo per la classe BookKeeper, iterazione 2

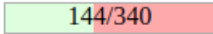
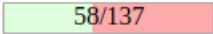
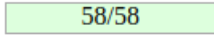
initializeEnsemblePlacementPolicy(ClientConfiguration, DNSToSwitchMapping, HashedWheelTimer, FeatureProvider, StatsLogger, BookieAddressResolver)	68%	n/a
asyncCreateLedger(int, int, BookKeeper.DigestType, byte[], AsyncCallback.CreateCallback, Object, Map)	85%	75%
createLedger(int, int, BookKeeper.DigestType, byte[], Map)	81%	50%
asyncOpenLedger(long, BookKeeper.DigestType, byte[], AsyncCallback.OpenCallback, Object)	80%	50%
validateEventLoopGroup(EventLoopGroup)	0%	n/a
asyncDeleteLedger(long, AsyncCallback.DeleteCallback, Object)	81%	50%
forConfig(ClientConfiguration)	0%	n/a
getReturnRc(int)	0%	n/a
getUnderlyingLedgerManager()	0%	n/a
newCreateLedgerOp()	0%	n/a
newOpenLedgerOp()	0%	n/a
newDeleteLedgerOp()	0%	n/a
getBookieAddressResolver()	0%	n/a
getBookieInfo()	0%	n/a
lambda\$getLedgerMetadata\$2(Versions)	0%	n/a
getLedgerManagerFactory()	0%	n/a
getCloseLock()	0%	n/a
getPlacementPolicy()	0%	n/a
getMetadataClientDriver()	0%	n/a
getStatsLogger()	0%	n/a
newListLedgersOp()	0%	n/a
openLedger(long, BookKeeper.DigestType, byte[])	100%	n/a
BookKeeper(String)	100%	n/a
deleteLedger(long)	100%	n/a
BookKeeper(ClientConfiguration)	100%	n/a
createLedger(int, int, BookKeeper.DigestType, byte[])	100%	n/a
createLedger(int, int, int, BookKeeper.DigestType, byte[])	100%	n/a
createLedger(BookKeeper.DigestType, byte[])	100%	n/a
static {...}	100%	n/a
getLedgerManager()	100%	n/a
getLedgerIdGenerator()	100%	n/a
isClosed()	100%	n/a
getBookieWatcher()	100%	n/a
getMainWorkerPool()	100%	n/a
getScheduler()	100%	n/a
getConf()	100%	n/a
getBookieClient()	100%	n/a
getClientCtx()	100%	n/a
Total	762 of 1.335	42% 54 of 74 27%

Figura 35 JaCoCo, method coverage classe BookKeeper, iterazione 2. Sono evidenziati i metodi complessivamente testati nelle due iterazioni

# Pit Test Coverage Report

## Package Summary

org.apache.bookkeeper.client

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	42% 	42% 	100% 

## Breakdown by Class

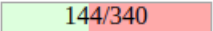
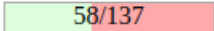
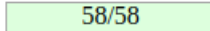
Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">BookKeeper.java</a>	42% 	42% 	100% 

Figura 36 Mutation coverage classe BookKeeper, iterazione 2

node	Output atteso
null	Nodo non aggiunto, ma nessuna eccezione
DataNode("127.0.0.1:4000", "/root")	Nodo correttamente aggiunto
InnerNode("127.0.0.1:4001", "/inner")	IllegalArgumentException
DataNode("127.0.0.1:4002", "/rack/127.0.0.1:3999")	IllegalArgumentException

Tabella 13 Test-cases per il metodo add() della classe NetworkTopologyImpl

```
@RunWith(Parameterized.class)
public class NetworkTopologyImplIT {
    // SUT
    private NetworkTopologyImpl sut;

    private Node node;

    public NetworkTopologyImplIT(Node node) {
        configure(node);
    }

    public void configure(Node node) {
        this.sut = new NetworkTopologyImpl();
        this.node = node;
    }

    @Parameterized.Parameters
    public static Collection<Object[]> parameters() {
        return Arrays.asList(new Object[][]{
            {new NodeBase("127.0.0.1:9999", "/rack")}
        });
    }

    @Test
    public void testAddIT() {
        node = Mockito.spy(node);
        sut.add(node);

        Mockito.verify(node, Mockito.times(1)).setParent(any());
    }
}
```

Codice 6 Integration test tra Node e NetworkTopologyImpl, riguardante la correttezza del metodo add()

node	Output atteso
null	Nessun valore di ritorno e nessun cambiamento della topologia
DataNode("127.0.0.1:4000", "/root")	Nodo rimosso correttamente
InnerNode("127.0.0.1:4001", "/inner")	IllegalArgumentException

Tabella 14 Casi di test per il metodo remove() di NetworkTopologyImpl



Figura 37 JaCoCo, class coverage per NetworkTopologyImpl, iterazione 1

NetworkTopologyImpl				
Element	Missed Instructions	Cov.	Missed Branches	Cov.
<a href="#">countNumOfAvailableNodes(String, Collection)</a>		0%		0%
<a href="#">getDistance(Node, Node)</a>		0%		0%
<a href="#">pseudoSortByDistance(Node, Node[])</a>		0%		0%
<a href="#">chooseRandom(String, String)</a>		0%		0%
<a href="#">add(Node)</a>		64%		77%
<a href="#">getLeaves(String)</a>		0%		0%
<a href="#">doGetLeaves(String)</a>		0%		0%
<a href="#">getDatanodesInRack(String)</a>		0%		0%
<a href="#">chooseRandom(String)</a>		0%		0%
<a href="#">isOnSameRack(Node, Node)</a>		0%		0%
<a href="#">swap(Node[], int, int)</a>		0%		n/a
<a href="#">remove(Node)</a>		85%		70%
<a href="#">isSameParents(Node, Node)</a>		0%		0%
<a href="#">getFirstHalf(String)</a>		0%		n/a
<a href="#">getLastHalf(String)</a>		0%		n/a
<a href="#">lambda\$getLeaves\$0(Set, String)</a>		0%		n/a
<a href="#">getRack(String)</a>		0%		n/a
<a href="#">isNodeGroupAware()</a>		0%		n/a
<a href="#">isOnSameNodeGroup(Node, Node)</a>		0%		n/a
<a href="#">toString()</a>		100%		100%
<a href="#">contains(Node)</a>		100%		87%
<a href="#">getNode(String)</a>		100%		50%
<a href="#">NetworkTopologyImpl()</a>		100%		n/a
<a href="#">getNumOfLeaves()</a>		100%		n/a
<a href="#">getNumOfRacks()</a>		100%		n/a
<a href="#">static {...}</a>		100%		n/a
<a href="#">getNodeForNetworkLocation(Node)</a>		100%		n/a
Total	602 of 933	35%	102 of 136	25%

Figura 38 JaCoCo method coverage, classe NetworkTopologyImpl, iterazione 1

## Pit Test Coverage Report

### Package Summary

org.apache.bookkeeper.net

Number of Classes	Line Coverage	Mutation Coverage	Test Strength
1	32%  73/230	15%  22/150	52%  22/42

### Breakdown by Class

Name	Line Coverage	Mutation Coverage	Test Strength
<a href="#">NetworkTopologyImpl.java</a>	32%  73/230	15%  22/150	52%  22/42

Figura 39 Mutation coverage classe NetworkTopologyImpl, iterazione 1

```
<failsafe-summary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="ht
<completed>1</completed>
<errors>0</errors>
<failures>0</failures>
<skipped>0</skipped>
<failureMessage xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:nil="true"/>
</failsafe-summary>
```

Figura 40 Risultato esecuzione del test di integrazione con failsafe, andato a buon fine

```
397. /**
398.  * Add a leaf node.
399.  * Update node counter and rack counter if necessary
400.  * @param node node to be added; can be null
401.  * @exception IllegalArgumentException if add a node to a leave
402.  *                                     or node to be added is not a leaf
403.  */
404. @Override
405. public void add(Node node) {
406.     if (node == null) {
407.         return;
408.     }
409.     String oldTopoStr = this.toString();
410.     if (node instanceof InnerNode) {
411.         throw new IllegalArgumentException("Not allow to add an inner node: " + NodeBase.getPath(node));
412.     }
413.     int newDepth = NodeBase.locationToDepth(node.getNetworkLocation()) + 1;
414.     netlock.writeLock().lock();
415.     try {
416.         if ((depthOfAllLeaves != -1) && (depthOfAllLeaves != newDepth)) {
417.             LOG.error("Error: can't add leaf node {} at depth {} to topology:\n{}", node, newDepth, oldTopoStr);
418.             throw new InvalidTopologyException("Invalid network topology. "
419.                 + "You cannot have a rack and a non-rack node at the same level of the network topology.");
420.         }
421.         Node rack = getNodeForNetworkLocation(node);
422.         if (rack != null && !(rack instanceof InnerNode)) {
423.             LOG.error("Unexpected data node {} at an illegal network location", node);
424.             throw new IllegalArgumentException("Unexpected data node " + node
425.                 + " at an illegal network location");
426.         }
427.         if (clusterMap.add(node)) {
428.             LOG.info("Adding a new node: " + NodeBase.getPath(node));
429.             if (rack == null) {
430.                 numOfRacks++;
431.             }
432.             if (!(node instanceof InnerNode)) {
433.                 if (depthOfAllLeaves == -1) {
434.                     depthOfAllLeaves = node.getLevel();
435.                 }
436.             }
437.         }
438.         if (LOG.isDebugEnabled()) {
439.             LOG.debug("NetworkTopology became:\n" + this);
440.         }
441.     } finally {
442.         netlock.writeLock().unlock();
443.     }
444. }
```

Figura 41 JaCoCo coverage del metodo add() della classe NetworkTopologyImpl, iterazione 1

```

<class name="org/apache/bookkeeper/net/NetworkTopologyImpl">
  <method name="add" desc="(Lorg/apache/bookkeeper/net/Node;)V">
    <du var="this" def="406" use="409" covered="1"/>
    <du var="this" def="406" use="414" covered="1"/>
    <du var="this" def="406" use="416" target="416" covered="1"/>
    <du var="this" def="406" use="416" target="421" covered="1"/>
    <du var="this" def="406" use="421" covered="1"/>
    <du var="this" def="406" use="427" target="428" covered="1"/>
    <du var="this" def="406" use="427" target="438" covered="0"/>
    <du var="this" def="406" use="442" covered="1"/>
    <du var="this" def="406" use="439" covered="0"/>
    <du var="this" def="406" use="433" target="434" covered="1"/>
    <du var="this" def="406" use="433" target="438" covered="1"/>
    <du var="this" def="406" use="434" covered="1"/>
    <du var="this" def="406" use="430" covered="1"/>
    <du var="this" def="406" use="416" target="417" covered="0"/>
    <du var="this" def="406" use="416" target="421" covered="1"/>
    <du var="node" def="406" use="406" target="407" covered="1"/>
    <du var="node" def="406" use="406" target="409" covered="1"/>
    <du var="node" def="406" use="410" target="411" covered="1"/>
    <du var="node" def="406" use="410" target="413" covered="1"/>
    <du var="node" def="406" use="413" covered="1"/>
    <du var="node" def="406" use="421" covered="1"/>
    <du var="node" def="406" use="427" target="428" covered="1"/>
    <du var="node" def="406" use="427" target="438" covered="0"/>
    <du var="node" def="406" use="428" covered="1"/>
    <du var="node" def="406" use="432" target="433" covered="1"/>
    <du var="node" def="406" use="432" target="438" covered="0"/>
    <du var="node" def="406" use="434" covered="1"/>
    <du var="node" def="406" use="423" covered="0"/>
    <du var="node" def="406" use="424" covered="0"/>
    <du var="node" def="406" use="417" covered="0"/>
    <du var="node" def="406" use="411" covered="1"/>
    <du var="this.netlock" def="406" use="414" covered="1"/>
    <du var="this.netlock" def="406" use="442" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="416" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="421" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="433" target="434" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="433" target="438" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="417" covered="0"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="421" covered="1"/>
    <du var="LOG" def="406" use="438" target="439" covered="0"/>
    <du var="LOG" def="406" use="438" target="442" covered="1"/>
    <du var="LOG" def="406" use="439" covered="0"/>
    <du var="LOG" def="406" use="428" covered="1"/>
    <du var="LOG" def="406" use="423" covered="0"/>
    <du var="LOG" def="406" use="417" covered="0"/>
    <du var="this.clusterMap" def="406" use="427" target="428" covered="1"/>
    <du var="this.clusterMap" def="406" use="427" target="438" covered="0"/>
    <du var="this.numOfRacks" def="406" use="430" covered="1"/>
    <du var="oldTopoStr" def="409" use="417" covered="0"/>
    <du var="newDepth" def="413" use="416" target="417" covered="0"/>
    <du var="newDepth" def="413" use="416" target="421" covered="1"/>
    <du var="newDepth" def="413" use="417" covered="0"/>
    <du var="rack" def="421" use="422" target="422" covered="1"/>
    <du var="rack" def="421" use="422" target="427" covered="1"/>
    <du var="rack" def="421" use="429" target="430" covered="1"/>
    <du var="rack" def="421" use="429" target="432" covered="1"/>
    <du var="rack" def="421" use="422" target="423" covered="0"/>
    <du var="rack" def="421" use="422" target="427" covered="1"/>
    <counter type="DU" missed="18" covered="40"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>

```

Figura 42 Ba-dua coverage: metodo add(), classe NetworkTopologyImpl, iterazione 1

node1	node2	Output atteso
DataNode("127.0.0.1:4000", "/sameRack") aggiunto	DataNode("127.0.0.1:4001", "/sameRack") aggiunto	true
DataNode("127.0.0.1:4000", "/rackA") aggiunto	DataNode("127.0.0.1:4001", "/rackB") aggiunto	false
null	DataNode("127.0.0.1:4002", "/rackC") aggiunto	IllegalArgumentException



DataNode("127.0.0.1:4003", "/rackD") aggiunto	null	IllegalArgumentException
null	null	IllegalArgumentException
DataNode("127.0.0.1:4000", "/rackA") non aggiunto	DataNode("127.0.0.1:4001", "/rackB") ag- giunto	IllegalArgumentException
DataNode("127.0.0.1:4000", "/rackA") aggiunto	DataNode("127.0.0.1:4001", "/rackB") non aggiunto	IllegalArgumentException
DataNode("127.0.0.1:4000", "/sameRack") non aggiunto	DataNode("127.0.0.1:4001", "/sameRack") non aggiunto	IllegalArgumentException

Tabella 15 Casi di test per il metodo `isOnSameRack()` della classe `NetworkTopologyImpl`

node1	node2	Output atteso
DataNode("127.0.0.1:4000", "/rack") aggiunto	DataNode("127.0.0.1:4000", "/rack") aggiunto	0
DataNode("127.0.0.1:4001", "/root/rack1") aggiunto	DataNode("127.0.0.1:4002", "/root/rack2") aggiunto	4
DataNode("127.0.0.1:4003", "/a") non aggiunto	DataNode("127.0.0.1:4004", "/b") aggiunto	Integer.MAX_VA- LUE
DataNode("127.0.0.1:4005", "/c") aggiunto	DataNode("127.0.0.1:4006", "/d") non aggiunto	Integer.MAX_VA- LUE
null	DataNode("127.0.0.1:4007", "/e") aggiunto	Exception
DataNode("127.0.0.1:4008", "/f") aggiunto	null	Exception
null	null	Exception

Tabella 16 Test-cases per il metodo `getDistance()` della classe `NetworkTopologyImpl`

```

/**
 * Check if two nodes are on the same rack.
 * @param node1 one node (can be null)
 * @param node2 another node (can be null)
 * @return true if node1 and node2 are on the same rack; false otherwise
 * @exception IllegalArgumentException when either node1 or node2 is null, or
 * node1 or node2 do not belong to the cluster
 */
public boolean isOnSameRack(Node node1, Node node2) {
    if (node1 == null || node2 == null) {
        return false;
    }

    netlock.readLock().lock();
    try {
        return isSameParents(node1, node2);
    } finally {
        netlock.readLock().unlock();
    }
}

```

Codice 7 Codice sorgente del metodo `isOnSameRack()`; viene ritornato `false` quando uno dei due parametri è `null`, anziché `IllegalArgumentException`



Figura 43 JaCoCo class coverage, classe `NetworkTopologyImpl`, iterazione 2

NetworkTopologyImpl					
Element	Missed Instructions	Cov.	Missed Branches	Cov.	
<a href="#">countNumOfAvailableNodes(String, Collection)</a>		0%		0%	
<a href="#">pseudoSortByDistance(Node, Node[])</a>		0%		0%	
<a href="#">chooseRandom(String, String)</a>		0%		0%	
<a href="#">getLeaves(String)</a>		0%		0%	
<a href="#">doGetLeaves(String)</a>		0%		0%	
<a href="#">getDatanodesInRack(String)</a>		0%		0%	
<a href="#">add(Node)</a>		80%		86%	
<a href="#">chooseRandom(String)</a>		0%		0%	
<a href="#">swap(Node[], int, int)</a>		0%		n/a	
<a href="#">getDistance(Node, Node)</a>		83%		70%	
<a href="#">remove(Node)</a>		85%		70%	
<a href="#">getFirstHalf(String)</a>		0%		n/a	
<a href="#">getLastHalf(String)</a>		0%		n/a	
<a href="#">lambda\$getLeaves\$0(Set, String)</a>		0%		n/a	
<a href="#">isOnSameRack(Node, Node)</a>		90%		50%	
<a href="#">getRack(String)</a>		0%		n/a	
<a href="#">isNodeGroupAware()</a>		0%		n/a	
<a href="#">isOnSameNodeGroup(Node, Node)</a>		0%		n/a	
<a href="#">toString()</a>		100%		100%	
<a href="#">contains(Node)</a>		100%		87%	
<a href="#">getNode(String)</a>		100%		50%	
<a href="#">NetworkTopologyImpl()</a>		100%		n/a	
<a href="#">getNumOfLeaves()</a>		100%		n/a	
<a href="#">getNumOfRacks()</a>		100%		n/a	
<a href="#">isSameParents(Node, Node)</a>		100%		100%	
<a href="#">static {...}</a>		100%		n/a	
<a href="#">getNodeForNetworkLocation(Node)</a>		100%		n/a	
Total	479 of 933	48%	82 of 136	39%	

Figura 44 JaCoCo method coverage, classe NetworkTopologyImpl, iterazione 2

```

404.  @Override
405.  public void add(Node node) {
406.      if (node == null) {
407.          return;
408.      }
409.      String oldTopoStr = this.toString();
410.      if (node instanceof InnerNode) {
411.          throw new IllegalArgumentException("Not allow to add an inner node: " + NodeBase.getPath(node));
412.      }
413.      int newDepth = NodeBase.locationToDepth(node.getNetworkLocation()) + 1;
414.      netlock.writeLock().lock();
415.      try {
416.          if ((depthOfAllLeaves != -1) && (depthOfAllLeaves != newDepth)) {
417.              LOG.error("Error: can't add leaf node {} at depth {} to topology:\n{}", node, newDepth, oldTopoStr);
418.              throw new InvalidTopologyException("Invalid network topology. "
419.                  + "You cannot have a rack and a non-rack node at the same level of the network topology.");
420.          }
421.          Node rack = getNodeForNetworkLocation(node);
422.          if (rack != null && !(rack instanceof InnerNode)) {
423.              LOG.error("Unexpected data node {} at an illegal network location", node);
424.              throw new IllegalArgumentException("Unexpected data node " + node
425.                  + " at an illegal network location");
426.          }
427.          if (clusterMap.add(node)) {
428.              LOG.info("Adding a new node: " + NodeBase.getPath(node));
429.              if (rack == null) {
430.                  numOfRacks++;
431.              }
432.              if (!(node instanceof InnerNode)) {
433.                  if (depthOfAllLeaves == -1) {
434.                      depthOfAllLeaves = node.getLevel();
435.                  }
436.              }
437.          }
438.          if (LOG.isDebugEnabled()) {
439.              LOG.debug("NetworkTopology became:\n" + this);
440.          }
441.      } finally {
442.          netlock.writeLock().unlock();
443.      }
444.  }

```

Figura 45 Coverage metodo add(), classe NetworkTopologyImpl, iterazione 2; coperto anche il lancio dell'eccezione InvalidTopologyException

```

▼<class name="org/apache/bookkeeper/net/NetworkTopologyImpl">
  ▼<method name="add" desc="(Lorg/apache/bookkeeper/net/Node;)V">
    <du var="this" def="406" use="409" covered="1"/>
    <du var="this" def="406" use="414" covered="1"/>
    <du var="this" def="406" use="416" target="416" covered="1"/>
    <du var="this" def="406" use="416" target="421" covered="1"/>
    <du var="this" def="406" use="421" covered="1"/>
    <du var="this" def="406" use="427" target="428" covered="1"/>
    <du var="this" def="406" use="427" target="438" covered="0"/>
    <du var="this" def="406" use="442" covered="1"/>
    <du var="this" def="406" use="439" covered="0"/>
    <du var="this" def="406" use="433" target="434" covered="1"/>
    <du var="this" def="406" use="433" target="438" covered="1"/>
    <du var="this" def="406" use="434" covered="1"/>
    <du var="this" def="406" use="430" covered="1"/>
    <du var="this" def="406" use="416" target="417" covered="1"/>
    <du var="this" def="406" use="416" target="421" covered="1"/>
    <du var="node" def="406" use="406" target="407" covered="1"/>
    <du var="node" def="406" use="406" target="409" covered="1"/>
    <du var="node" def="406" use="410" target="411" covered="1"/>
    <du var="node" def="406" use="410" target="413" covered="1"/>
    <du var="node" def="406" use="413" covered="1"/>
    <du var="node" def="406" use="421" covered="1"/>
    <du var="node" def="406" use="427" target="428" covered="1"/>
    <du var="node" def="406" use="427" target="438" covered="0"/>
    <du var="node" def="406" use="428" covered="1"/>
    <du var="node" def="406" use="432" target="433" covered="1"/>
    <du var="node" def="406" use="432" target="438" covered="0"/>
    <du var="node" def="406" use="434" covered="1"/>
    <du var="node" def="406" use="423" covered="0"/>
    <du var="node" def="406" use="424" covered="0"/>
    <du var="node" def="406" use="417" covered="1"/>
    <du var="node" def="406" use="411" covered="1"/>
    <du var="this.netlock" def="406" use="414" covered="1"/>
    <du var="this.netlock" def="406" use="442" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="416" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="421" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="433" target="434" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="433" target="438" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="417" covered="1"/>
    <du var="this.depthOfAllLeaves" def="406" use="416" target="421" covered="1"/>
    <du var="LOG" def="406" use="438" target="439" covered="0"/>
    <du var="LOG" def="406" use="438" target="442" covered="1"/>
    <du var="LOG" def="406" use="439" covered="0"/>
    <du var="LOG" def="406" use="428" covered="1"/>
    <du var="LOG" def="406" use="423" covered="0"/>
    <du var="LOG" def="406" use="417" covered="1"/>
    <du var="this.clusterMap" def="406" use="427" target="428" covered="1"/>
    <du var="this.clusterMap" def="406" use="427" target="438" covered="0"/>
    <du var="this.numOfRacks" def="406" use="430" covered="1"/>
    <du var="oldTopoStr" def="409" use="417" covered="1"/>
    <du var="newDepth" def="413" use="416" target="417" covered="1"/>
    <du var="newDepth" def="413" use="416" target="421" covered="1"/>
    <du var="newDepth" def="413" use="417" covered="1"/>
    <du var="rack" def="421" use="422" target="422" covered="1"/>
    <du var="rack" def="421" use="422" target="427" covered="1"/>
    <du var="rack" def="421" use="429" target="430" covered="1"/>
    <du var="rack" def="421" use="429" target="432" covered="1"/>
    <du var="rack" def="421" use="422" target="423" covered="0"/>
    <du var="rack" def="421" use="422" target="427" covered="1"/>
    <counter type="DU" missed="11" covered="47"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>

```

Figura 46 Ba-dua coverage, metodo add(), classe NetworkTopologyImpl, iterazione 2; sono evidenziate le coppie def-use coperte rispetto all'iterazione precedente

Pit Test Coverage Report					
Package Summary					
org.apache.bookkeeper.net					
Number of Classes	Line Coverage		Mutation Coverage		Test Strength
1	43%	99/230	32%	48/150	70% 48/69
Breakdown by Class					
Name	Line Coverage		Mutation Coverage		Test Strength
<a href="#">NetworkTopologyImpl.java</a>	43%	99/230	32%	48/150	70% 48/69

Figura 47 Mutation coverage, classe NetworkTopologyImpl, iterazione 2

metodo	parametri	profilo operativo	# test	# test passati	reliability
createLedger	ensSize, writeQuorumSize, ackQuorumSize, digestType, passwd, customMetadata	vedi <a href="#">Tabella 7</a>	27	18	0.667
createLedger	ensSize, writeQuorumSize, ackQuorumSize, digestType, passwd	vedi <a href="#">Tabella 7</a>	27	18	0.667
createLedger	ensSize, qSize, digestType, passwd	vedi <a href="#">Tabella 10</a>	9	6	0.667
createLedger	digestType, passwd	vedi <a href="#">Tabella 11</a>	4	4	1
deleteLedger	lld	vedi <a href="#">Tabella 8</a>	2	1	0.5
openLedger	lld, digestType, passwd	vedi <a href="#">Tabella 12</a>	6	6	1

*Tabella 17 Riferimenti ai profili operazionali e stima della reliability per i metodi testati della classe BookKeeper*

metodo	parametri	profilo operativo	# test	# test passati	reliability
add	node	vedi <a href="#">Tabella 13</a> + ulteriore caso di test per la copertura di InvalidTopologyException	5	4	0.8
remove	node	vedi <a href="#">Tabella 14</a>	3	3	1
isOnSameRack	node1, node2	vedi <a href="#">Tabella 15</a>	8	2	0.25
getDistance	node1, node2	vedi <a href="#">Tabella 16</a>	7	4	0.571

*Tabella 18 Riferimenti a profili operazionali e stima della reliability per i metodi testati della classe NetworkTopologyImpl*