ReportISW2-ModuloSWTesting

Marco Marcucci 0286352

Introduzione

L'obiettivo del report è descrivere le attività di testing, presentate durante il corso, effettuate su delle classi java appartenenti a progetti sviluppati da Apache Software Foundation. I progetti presi in considerazione sono **BookKeepper** ed **OpenJPA**.
Entrambi i progetti sono stati inserito sul servizio di hosting GitHub e configurati in modo tale da essere integrati con i servizi Travis-CI e SonarCloud.
Inoltre, per l'implementazione dei casi di test, è stato utilizzato il framework JUnit e Mockito; per lo Statement e Branch Coverage è stata utilizzata la libreria JaCoCo; per la mutation coverage è stata utilizzata la libreria PIT. Questi servizi e librerie sono integrabili facilmente con il sistema di build Maven che, con il supporto di Travis-CI e SonarCloud, permettono di ottenere Continuous Integration e Continuous Testing.

Scelte di Testing

Tutti i test successivamente descritti sono stati identificati secondo un criterio di tipo unidimensionale:

- Ogni parametro di input al test è considerato in modo indipendente;
- I test sono scelti/sintetizzati per coprire tutte le classi di equivalenza considerate.

BookKeeper

BookKeper è un servizio di archiviazione scalabile, tollerante ai guasti e con bassa latenza, ottimizzato per carichi di lavoro in tempo reale.
BookKeeper è il principale client utilizzato per creare, aprire ed eliminare i Ledgers. Un Ledger è un file di registro che contiene una sequenza di entries. Solo l'utente che crea un Ledger può scrivervi. Un LedgerHandle rappresenta il Ledger per l'utente e consente ad esso di leggere e scrivere entries. Quando l'utente ha finito di scrivere, può chiudere il LedgerHandle. Una volta che un Ledger è stato chiuso, a tutti gli utenti che in seguito leggeranno il Ledger, viene garantita la lettura delle entries in modo consistente e nello stesso ordine. Tutti i metodi di BookKeeper e LedgerHandle hanno versioni sincrone e asincrone. Internamente le versioni sincrone vengono implementate utilizzando quella asincrona.
Le classi scelte per eseguire le attività di testing sono *DigestManager e WriteCache*. Come si evince dalla Figura 1, entrambi le classi sono state modificate nella versione 14. In particolare, sono state aggiunte molte più linee di codice di quante non ne siano state cancellate. Questo potrebbe aver portato all'introduzione di bug.

WiteCache

Considerando i commenti nel codice, la classe permette di allocare una parte di memoria, suddivisa in più segmenti. Le entries vengono aggiunte in un buffer comune e indicizzate tramite una hashmap, finché la cache non viene svuotata.

Il primo metodo testato è:

public boolean put(long ledgerId, long entryId, ByteBuf entry)

Considerando il codice, il metodo inserisce la entry specificata nel ledger specificato. Se i parametri forniti superano una serie di controlli interni e se la cache non è piena, il metodo restituisce il valore *true*, altrimenti restituisce il valore *false*.
La classe di equivalenza per il parametro *ledgerId* è la seguente: $\{< 0, \geq 0\}$; La classe di equivalenza per il parametro *entryId* è la seguente: $\{< 0, \geq 0\}$; La classe di equivalenza per il parametro *ByteBuf* è la seguente: {*null*, *ValidIstance, NonValidIstance*}.
Analizzando più nello specifico il codice, il metodo, prima di inserire la entry specificata nel ledger specificato, controlla che questi due valori non siano minori di zero. Però, dato che non è stata trovata nessuna documentazione o commento al riguardo, sono stati considerati anche i valori negativi. Inoltre, non trovando nessuna relazione esplicita tra *ledgerId* e *entryId*, e considerando che i più comuni bug di programmazione sono introdotti sui valori a "confine" delle classi di equivalenza, una possibile selezione dei valori per i primi due parametri, secondo la Boundary-value analysis, è la seguente:

- *ledgerId*: {-1, 0, 1};

- *entryId*: {-1, 0, 1}.

Per quanto riguarda, invece, il parametro *entry*, è stato osservato che non è possibile istanziare un oggetto di tipo *ByteBuf* che sia invalido per il metodo considerato. Quindi, una possibile selezione dei valori, secondo la Boundary-value analysis, è la seguente: *entry*: {*null*, *String "test"*}.
Una possibile selezione minimale dei casi di test è la seguente: {*put(1,1,"test")*, *put(0,-1, null)*, *put(-1,0,"test")*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 94%, di Branch Coverage del 50% e un valore di Mutation Score del : $\frac{|D|}{|M|-|E|} = \frac{7}{14} \cong 50\%$, come si evince dalla Figura 4, Figura 6 e Figura 7.
Per aumentare tali valori, sono stati aggiunti ulteriori casi di test implementati in modo tale da raggiungere più statement e branch e per uccidere ulteriori mutazioni. I nuovi valori ottenuti sono rispettivamente: 98%, 87%, 64%, come si può osservare dalla Figura 5, Figura 8, Figura 9. Le mutazioni a riga 147, 150 e 167 sono equivalenti rispetto a Strong Mutation.

Il secondo metodo testato è:

public ByteBuf get(long ledgerId, long entryId)

Considerando il codice, il metodo restituisce il dato *ByteBuf* corrispondente ai parametri specificati, se il controllo su questi ultimi è corretto, *null* altrimenti.
La classe di equivalenza per il parametro *ledgerId* è la seguente: {< 0,≥ 0 }; La classe di equivalenza per il parametro *entryId* è la seguente: {< 0,≥ 0 }.
Anche in questo caso, analizzando più attentamente il codice, è stato notato che non esiste nessuna relazione esplicita tra i due parametri e che, anche se esiste un controllo sul segno di *ledgerId*, questo non è stato considerato nell'analisi dei Boundary-value, per le stesse motivazioni del metodo *put()*.
Una possibile selezione dei valori per i primi due parametri, secondo la Boundary-value analysis, è la seguente:

- *ledgerId*: {-1, 0, 1};
- *entryId*: {-1, 0, 1}.

Quindi, una possibile selezione minimale dei casi di test è la seguente: {*get(0,0)*, *get(-1,-1)*, *get(1,1)*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 95%, di Branch Coverage del 50% e un valore di Mutation Score del : $\frac{|D|}{|M|-|E|} = \frac{4}{4} \cong 100\%$, come si evince dalla Figura 4, Figura 10Figura 12 e Figura 11.
Per aumentare tali valori, è stato aggiunto un ulteriore caso di test implementato in modo tale da raggiungere più statement e branch. I nuovi valori ottenuti di Statement Coverage e Branch Coverage sono rispettivamente: 100% e 100% come si può osservare dalla Figura 5, Figura 12.

DigestManager

Considerando i commenti nel codice, la classe *DigestManager.java*, da un lato, prima di spedire un pacchetto al bookie, preleva l'entry considerata, allega ad essa un digest e la impacchetta con i dati strutturati nel modo corretto. Dall'altro lato, invece, la classe quando riceve un pacchetto, controlla che il digest corrisponda ed estrae la entrry originale. Attualmente la classe supportat 3 tipi di digest: MAC (basato su SHA-1), CRC32 e CRC32C.

Il primo metodo testato è:

public static byte[] generateMasterKey(byte[] password)

Considerando il codice, il metodo restituisce un oggetto di tipo *byte[]* diverso in base al parametro *password* specificato. In particolare, se la lunghezza del parametro è maggiore di 0, il metodo restituisce un digest costruito in un certo modo in cui interviene sia il parametro che la stringa "ledger"; se invece, il parametro contiene una sequenza di byte vuota, allora il metodo restituisce un digest pre-computato al momento dell'istanziazione della classe.
La classe di equivalenza per il parametro *password* è la seguente: {$byte[].lenght = 0$,byte[].lenght ≥ 0 }.
Una possibile selezione minimale dei casi di test è la seguente: {*generateMasterKey ("".getBytes())*, *generateMasterKey (new byte[])*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 100%, di Branch Coverage del 100% e un valore di Mutation Score del : $\frac{|D|}{|M|-|E|} = \frac{2}{3} \cong 66\%$, come si evince dalla Figura 13, Figura 15 e Figura 16.
L'unica mutazione sopravvissuta è quella in cui l'operatore $>$ è stato sostituito con l'operatore $\geq$. Questa mutazione può essere considerata equivalente rispetto a Strong Mutation, dato che il valore di ritorno del metodo, specificando come parametro un byte di lunghezza 0, con la mutazione attiva è lo stesso anche

senza la mutazione. Invece, i percorsi di esecuzione del metodo sono diversi nel momento in cui viene introdotta la mutazione.

Il secondo metodo testato è:

public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)

Considerando i commenti del codice, il metodo si occupa di verificare che il digest del dato ricevuto coincida con il digest costruito al momento prelevando i dati necessari dal parametro specificato. Nel caso questo confronto abbia successo, il metodo restituisce il tipo di dato *ByteBuf* contenente il dato ricevuto, altrimenti restituisce un'eccezione del tipo *BKDigestMatchException*.
La classe di equivalenza per il parametro *entryId* è la seguente: $\{< 0, \geq 0\}$; La classe di equivalenza per il parametro *dataReceived* è la seguente: {*null*, *ValidIstance*, *NonValidIstance*}.
Per quanto riguarda il parametro *dataReceived*, anche in questo caso, non è stato possibile istanziare un oggetto di tipo *ByteBuf* che sia invalido per il metodo considerato. Considerando inoltre, che i più comuni bug di programmazione sono introdotti sui valori a "confine" delle classi di equivalenza, una possibile selezione dei valori per i due parametri, secondo la Boundary-value analysis, è la seguente:

- *entryId*: {-1, 0, 1};
- *dataReceived*: {*ValidIstance, null*}.

Una possibile selezione minimale dei casi di test è la seguente: {*verifyDigestAndReturnData (*-1, *null)*, *verifyDigestAndReturnData (*1, *ValidIstance)*, *verifyDigestAndReturnData (*0, *ValidIstance)*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 100% e un Mutation Score del : $\frac{|D|}{|M|-|E|} = \frac{2}{3} \cong 66\%$, come si evince dalla Figura 13, Figura 17 e Figura 18. È stato aggiunto un ulteriore caso di test per uccidere l'unica mutazione rimasta in vita, come si può osservare dalla Figura 19.

Il terzo metodo testato è:

public long verifyDigestAndReturnLac(ByteBuf dataReceived)

Considerando il codice, il metodo si occupa di verificare che il digest presente nel dato ricevuto *dataReceived*, coincida con il digest costruito al momento prelevando il LAC (Last Add Confirmed) dal dato specificato come parametro. Nel caso questo confronto abbia successo, io metodo restituisce il LAC ricevuto, altrimenti un'eccezione del tipo *BKDigestMatchException.*
La classe di equivalenza per il parametro *dataReceived* è la seguente: {*null*, *ValidIstance*, *NonValidIstance*}.
Anche in questo caso non è stato possibile istanziare un oggetto di tipo *ByteBuf* che sia invalido per il metodo considerato e, quindi, una possibile selezione minimale dei casi di test è la seguente: {*verifyDigestAndReturnLac (null)*, *verifyDigestAndReturnLac (ValidIstance)*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 49%, di Branch Coverage del 50% e un valore di Mutation Score del : $\frac{|D|}{|M|-|E|} = \frac{7}{8} \cong 88\%$, come si evince dalla Figura 13, Figura 20 e Figura 21.
Per aumentare tali valori, sono stati aggiunti ulteriori casi di test implementati in modo tale da raggiungere più statement e branch e uccidere l'unica mutazione rimasta. I nuovi valori ottenuti sono rispettivamente: 100%, 100%, 100%, come si può osservare dalla Figura 14, Figura 22, Figura 23.

Il quarto metodo testato è:

private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)

Considerando il codice, il metodo si occupa di verificare il digest presente nel dato *dataReceived* specificato sia corretto.
Il codice del metodo presenta più controlli che, se convalidati tutti, certificano l'adeguatezza del dato ricevuto.
La classe di equivalenza per il parametro *entryId* è la seguente: $\{< 0, \geq 0\}$; La classe di equivalenza per il parametro *dataReceived* è la seguente: {*null*, *ValidIstance*, *NonValidIstance*}; La classe di equivalenza per il parametro *skipEntryIdCheck* è la seguente: {*true*, *false*}. Considerando che il valore *skipEntryIdCheck* viene sempre utilizzato come false, una possibile selezione dei valori per i tre parametri, secondo la Boundary-value analysis, è la seguente:

- *entryId*: {-1, 0, 1};
- *dataReceived*: {*ValidIstance, null*};
- *skipEntryIdCheck*: {*false*}.

Una possibile selezione minimale dei casi di test è la seguente: {*verifyDigest(*-1, *null*, *false)*, *verifyDigest(*1*, ValidIstance, false), verifyDigest(*0, *ValidIstance, false)*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 61%, di Branch Coverage del 60% e un valore di Mutation Score del : $\frac{|D|}{|M|-|E|} = \frac{9}{12} \cong 75\%$, come si evince dalla Figura 13, Figura 24 e Figura 25.

Per aumentare tali valori, sono stati aggiunti ulteriori casi di test implementati in modo tale da raggiungere più statement e branch e uccidere più mutazioni possibili. I nuovi valori ottenuti sono rispettivamente: 100%, 90%, 100%, come si può osservare dalla Figura 14, Figura 26, Figura 27.

OpenJPA

OpenJPA è l'implementazione di Apache della specifica API Java Persistence 2.0 per la persistenza trasparente degli oggetti Java.
I dati persistenti sono informazioni che possono sopravvivere al programma che li crea. La maggior parte dei programmi complessi utilizza dati persistenti: le applicazioni GUI devono memorizzare le preferenze dell'utente tra le chiamate dei programmi, le applicazioni web tengono traccia dei movimenti e degli ordini degli utenti per lunghi periodi di tempo, ecc.
Le classi scelte per eseguire le attività di testing sono *ProxyManagerImpl* e *CacheMap*. La prima classe è stata selezionata in quanto dalla versione 2 fino alla versione 12, come si evince dalla Figura 2, sono state inserite numerose righe di codice ed eliminate quasi lo stesso numero di righe. Questi risultati fanno pensare che la classe è stata modificata molte volte durante lo sviluppo del progetto e quindi potrebbero essere presenti eventuali bug. La seconda classe anche è stata selezionata per un discreto numero di righe aggiunte nel corso delle versioni, a partire dalla prima fino alla undicesima. In particolare, nella versione 6 e 11 si riscontra un inserimento ingente di linee di codice, come si evince dalla Figura 3. In questo caso la classe è stata modificata quasi esclusivamente aggiungendo linee di codice che potrebbe aver portato, quindi, all'introduzione di eventuali bug.
Inoltre, entrambe le classi presentano sia una documentazione ufficiale che dei commenti nel codice

ProxyManagerImpl

Per la gestione dei proxies, OpenJPA definisce l'interfaccia *ProxyManager*. Quest'ultima, viene implementata dalla classe scelta per il testing *ProxyManagerImpl*, che rappresenta per il progetto la classe di default per la gestione dei proxies. Quest'ultimo può eseguire il proxy dei metodi standard di qualsiasi classe di tipo Collection, List, Map, Queue, Date o Calendar, comprese le implementazioni personalizzate. Può anche eseguire il proxy di classi personalizzate che abbiano implementato in modo corretto i metodi getter e setter. I tipi personalizzati devono, tuttavia, soddisfare i seguenti criteri:

- I tipi di dati che personalizzano il tipo Container devono avere un costruttore pubblico senza argomenti o che accetti un solo argomento di tipo *Comparator*.
- I tipi di dati che personalizzano il tipo *Date* devono avere un costruttore pubblico senza argomenti o che accetti un singolo argomento di tipo *long* che rappresenta il tempo corrente.
- Gli altri tipi personalizzati devono avere un costruttore pubblico senza argomenti oppure uno che accetti come argomento un'istanza dello stesso tipo in modo tale da copiarla completamente (public copy constructor). Se non è definito il copy constructor, il tipo deve avere la possibilità di copiare completamente un'istanza A creando una nuova istanza B. Per la copia dovranno essere utilizzati, per ogni attributo, i getter dell'istanza B con i valori ottenuti dai corrispondenti getter dell'stanza A.

Il primo metodo testate è:

public Object copyArray(Object orig)

Considerando la documentazione di OpenJPA, il metodo restituisce un nuovo array, dello stesso tipo dell'array passato come parametro, contenente gli stessi elementi di quest'ultimo. Inoltre, attraverso un ulteriore analisi del codice si nota che il metodo effettua alcuni controlli preliminari sull'oggetto passato come parametro. Innanzitutto, si controlla che l'oggetto non sia nullo e, in seguito, si gestisce il fatto che l'array passato possa essere non supportato.
La classe di equivalenza per il parametro *orig* è la seguente: {null, ValidInstance, NonValidIstance}. Per l'istanza valida è stato considerato un array di tipo *Float*; per l'istanza non valida, invece, è stato considerato un oggetto di tipo Array*List*.
Una possibile selezione minimale dei casi di test è la seguente: {copyArray(null), copyArray(new Float[]), copyArray(new ArrayList())}.
Questi tre casi di test evidenziano un valore di Statement Coverage e Branch Coverage massimo, come si può evincere dalla Figura 28 e Figura 31, così come evidenziano un valore di Mutation Score massimo. Infatti, come si può vedere dalla Figura 32, tutte le mutazioni create da PIT sono state uccise dai casi di test. Per questi motivi, non risulta essere necessario aggiungere nuovi casi di test.

Il secondo metodo testato è:

public Object copyCustom(Object orig)

Considerando la documentazione di OpenJPA, il metodo restituisce una copia contenente le stesse informazioni dell'oggetto passato come parametro, oppure il valore null se il gestore non è in grado di copiare l'oggetto. Analizzando il codice, si osserva che tale metodo prevede la copia di tutti gli oggetti descritti nell'introduzione della classe, compresi quelli personalizzati.
La classe di equivalenza per il parametro *orig* è la seguente: {null, ValidInstance, NonValidIstance}. Per quanto riguarda l'istanza valida è stato considerato un oggetto di tipo *Date*; per l'istanza non valida, invece, è stato considerato un oggetto di tipo *NonValid*.
La classe *NonValid.java*, contenuta in *org.apache.openjpa.util.entity*, rappresenta un esempio di classe personalizzata che non rispecchia i requisiti, sopra descritti, del gestore.
Una possibile selezione minimale dei casi di test è la seguente: {copyCustum(null), copyCustum(new Date()), copyCustum(new NonValid())}.
Questi tre casi di test evidenziano un valore di statement coverage del 59% e branch coverage del 62%, come si evince dalla Figura 29 e Figura 33. Il Mutation score, in Figura 34, del singolo metodo è: $\frac{|D|}{|M|-|E|} = \frac{8}{14} \cong 57\%$.
Per aumentare le percentuali sopra elencate, sono stati aggiunti nuovi casi di testi che hanno permesso di ottenere un valore di Statement Coverage, Branch Coverage e Mutation Score del 100%, come si può osservare dalla Figura 30, Figura 35 e Figura 36.
I test aggiunti, sono stati implementati selezionando gli altri tipi di oggetto che il gestore supporta per la copia. È stato anche utilizzato un tipo di dato personalizzato che, al contrario della classe *NonValid.java*, rispecchia tutti i requisiti del gestore. La classe è *Valid.java* ed è contenuta in *org.apache.openjpa.util.entityI.*

Il terzo metodo testato è:

public Proxy newCustomProxy(Object orig, boolean autoOff)

Considerando la documentazione di OpenJPA, il metodo restituisce un oggetto di tipo *Proxy*, a partire dal tipo dell'oggetto passato come parametro, oppure il valore nullo se il gestore non è in grado di istanziare il proxy per il tipo dell'oggetto. Analizzando il codice, si osserva che tale metodo prevede l'istanziazione di un oggetto di tipo *Proxy* per tutti i possibili tipi di oggetto descritti nell'introduzione della classe, compresi quelli personalizzati. Nel caso in cui il tipo dell'oggetto non sia supportato dal gestore oppure venga passato il valore nullo, il metodo restituisce *null*.
Il comportamento predefinito di OpenJPA per il tracciamento del tipo di dato *Collections*, è tale per cui se il numero di modifiche sull'oggetto, supera il numero corrente di elementi di quest'ultimo, allora il tracciamento per tale oggetto verrà disabilitato. Il parametro *autoOff*, se impostato a false, permette di disabilitare tale comportamento.
La classe di equivalenza per il parametro *orig* è la seguente: {null, ValidInstance, NonValidIstance}; La classe di equivalenza per il parametro *autoOff* è la seguente: {true, false}. Per quanto riguarda l'istanza valida è stato considerato un oggetto di tipo *Date*; per l'istanza non valida, invece, è stato considerato un oggetto di tipo *NonValid*, descritto nel metodo precedente.
Una possibile selezione minimale dei casi di test è la seguente: {*newCustomProxy(null, true)*, *newCustomProxy(new NonValid(), false)*, *newCustomProxy(new ArrayList(), true)*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 41%, di Branch Coverage del 50% e un Mutation Score del $\frac{|D|}{|M|-|E|} = \frac{10}{22} \cong 45\%$, come è possibile osservare dalla Figura 29, Figura 37 e Figura 38.
Anche in questo caso sono stati aggiunti nuovi casi di test, utilizzando altri tipi di dato supportati dal gestore. Tali test hanno permesso di ottenere un valore di Statement Coverage, Branch Coverage del e Mutation Score del 100%. I risultati sono mostrati in Figura 30, Figura 39 e Figura 40.
In questo caso, è sopravvissuta una sola mutazione derivante dal fatto che, lo statement rimosso da pit a riga 325, non aveva nessun effetto sul ritorno del metodo. Tale mutazione viene identificata come equivalente rispetto a Strong Mutation.

CacheMap

La classe *CacheMap*, che implementa l'interfaccia Java *Map*, permette di ottenere una mappa di dimensioni fisse che ha la capacità sia di bloccare e sbloccare (pin e unpin) le proprie entry, sia di inserire in una mappa di supporto le entry che non trovano più spazio nella mappa principale.
Analizzando più approfonditamente il codice della classe, infatti, sono stati individuati attributi che ne chiariscono il comportamento:

- *protected final SizedMap cacheMap:* mappa che contiene i riferimenti che non sono già bloccati o scaduti;
- *protected final SizedMap softMap*: mappa utilizzata per i riferimenti scaduti;

- *protected fnal SizedMap pinnedMap*: contiene gli oggetti che sono stati bloccati nella cache.

La classe utilizza anche due attributi di tipo *ReentrantReadWriteLock*, utilizzati per scrivere e leggere, in modo concorrente, la cache. Per questo motivo, in tutti i test presentati successivamente, è stata utilizzata la libreria Mockito per assicurarsi che i metodi presi in considerazione acquisissero e rilasciassero effettivamente i lock descritti. In particolare, è stato utilizzato il metodo *org.mockito.Mockito.spy()* sulla classe *CacheMap*, in modo tale da verificare, attraverso il metodo *org.mockito.Mockito.verify()*, che all'interno dei metodi venissero effettivamente chiamati i metodi *CacheMap.readLock()* e *CacheMap.readUnlock()*.
All'istanziazione della classe, è possibile settare il parametro *boolean lru* che consente di istanziare una mappa che gestisce i riferimenti da rimuovere con una politica di tipo Last Recently Used.

Il primo metodo testato è:

public boolean pin(Object key)

Considerando la documentazione, il metodo blocca la chiave specificata ed il suo valore nella mappa. I riferimenti che sono bloccati, però, non vengono considerati nuovamente per il calcolo della dimensione della cache e non vengono mai eliminati implicitamente. È possibile, inoltre, bloccare chiavi per le quali non è presente alcun valore nella mappa.
Il metodo restituisce il valore *true* se la chiave specificata viene bloccata, *false* se nella *caheMap* non esiste la chiave.
La classe di equivalenza per il parametro *key* è la seguente: {*null*, *ValidInstance*}. Infatti, per il tipo *Object* non è stato possibile considerare un test con un parametro che risultasse non valido per il metodo.
Una possibile selezione minimale dei casi di test è la seguente: {*pin(null)*, *pin(new Object())*}.
La test suite minimale ha evidenziato un valore di Statement Coverage del 96%, di Branch Coverage del 87% e un Mutation Score del $\frac{|D|}{|M|-|E|} = \frac{9}{13} \cong 69\%$, come è possibile osservare dalla Figura 41, Figura 43 e Figura 44.
Per aumentare tali valori, analizzando più approfonditamente il codice del metodo, sono stati aggiunti ulteriori test che hanno permesso di uccidere ulteriori mutazioni e di coprire l'unico Branch non coperto. Infatti, la test suite minimale non prevedeva un test in cui la chiave, passata come parametro, aveva un valore diverso da *null*. La Figura 42, Figura 45 e Figura 46, mostrano le nuove percentuali raggiunte, ovvero rispettivamente: 100%, 100% e 100%. Due mutazioni risultano essere equivalenti rispetto a Strong Mutation e Weak Mutation a riga 308 e 310.

Il secondo metodo testato è:

public boolean unpin(Obect key)

Considerando la documentazione, il metodo sblocca la chiave specificata. Analizzando il codice, si può osservare che il metodo restituisce il valore *true*, se il valore associato alla chiave non è nullo, *false* altrimenti.
La classe di equivalenza per il parametro *key* è la seguente: {*null*, *ValidInstance*}. Infatti, per il tipo *Object* non è stato possibile considerare un test con un parametro che risultasse non valido per il metodo.
Una possibile selezione minimale dei casi di test è la seguente: {*unpin(null)*, *unpin(new Object())*}.
In questo caso, la suite minimale ha evidenziato il massimo valore per Statement Coverage, Branch Coverage e Mutation Score. Non è stato necessario, quindi, aggiungere ulteriori casi di test. In Figura 41, Figura 47 e Figura 48 è possibile osservare tali valori. Si può osservare come ci siano due mutazioni non uccise dalla test suite a riga 327 e 329, che corrispondo a mutazioni equivalenti rispetto a Strong Mutation e Weak Mutation.

Il terzo metodo testato è:

public Object put(Object key, Object value)

Considerando la documentazione, il metodo associa il valore specificato alla chiave specificata nella *cacheMap*. Se la mappa conteneva un valore per questa chiave, il vecchio valore viene sostituito dal valore specificato. Analizzando il codice, si osserva che il valore di ritorno del metodo è *null* se il valore associato alla chiave specificata è *null* oppure la chiave non era mai stata inserita, altrimenti il metodo restituisce il vecchio valore associato alla chiave. Il percorso di esecuzione del metodo prevede che venga fatto un controllo sulla chiave prima nella pinned list e successivamente nella soft map.
La classe di equivalenza per il parametro *key* è la seguente: {*null*, *ValidInstance*}; La classe di equivalenza per il parametro *value* è la seguente: {*null*, *ValidInstance*}. Infatti, per il tipo *Object* non è stato possibile considerare un test con un parametro che risultasse non valido per il metodo.
Una possibile selezione minimale dei casi di test è la seguente: {*put(null, null)*, *put(new Object(), null), put(null, new Obect()), put(new Obect(), new Obect())*}.
La suite minimale ha evidenziato il valore di Statement Coverage e Branch Coverage pari al 90%, mentre il Mutation Score è pari al $\frac{|D|}{|M|-|E|} = \frac{15}{19} \cong 79\%$, come si può osservare alla Figura 41, Figura 49 e Figura 50.

È stato possibile aumentare tali percentuali analizzando le mutazioni non uccise, i branch e statement non raggiunti dalla test suite. In Figura 42, Figura 51 e Figura 52 sono disponibili le nuove percentuali raggiunte aggiungendo ulteriori test alla suite. I nuovi valori di Statement Coverage, Branch Coverage e Mutation Score sono pari al 100%. Anche in questo caso, è presente una mutazione equivalente rispetto a Strong e Weak Mutation a riga 404.

Il quarto metodo testato è:

public void remove(Obect key)

Considerando la documentazione, il metodo rimuove l'oggetto dalla *cacheMap* ed inoltre se la chiave era bloccata, essa viene sbloccata. Anche in questo caso, analizzando il codice, si osserva che il valore di ritorno del metodo è *null* se il valore associato alla chiave specificata è *null* oppure la chiave non era mai stata inserita, altrimenti il metodo restituisce il valore associato alla chiave prima della rimozione.
La classe di equivalenza per il parametro *key* è la seguente: {*null*, *ValidInstance*}. Infatti, per il tipo *Object* non è stato possibile considerare un test con un parametro che risultasse non valido per il metodo.
Una possibile selezione minimale dei casi di test è la seguente: {*remove(null)*, *remove(new Obect())*}.
La suite minimale ha evidenziato un valore di Statement Coverage del 71%, di Branch Coverage del 62% e Mutation score del $\frac{|D|}{|M|-|E|} = \frac{3}{11} \cong 27\%$, come si evince dalla Figura 41, Figura 53 e Figura 54.
Anche in questo caso sono stati aggiunti ulteriori casi di test per aumentare le percentuali sopra descritte fino al 100% per tutti e tre i valori. In Figura 42, Figura 55 e Figura 56 è possibile osservare i nuovi valori ottenuti.

Il quinto metodo testato è:

public Object get(Object key)

Considerando la documentazione, il metodo restituisce il valore della chiave specificata se la chiave è presente nella *cacheMap, pinnedMap* o *softMap*, oppure *null* se la chiave non è presente in una delle tre mappe oppure il valore associato alla chiave è *null*.
La classe di equivalenza per il parametro *key* è la seguente: {*null*, *ValidInstance*}. Infatti, per il tipo *Object* non è stato possibile considerare un test con un parametro che risultasse non valido per il metodo.
Una possibile selezione minimale dei casi di test è la seguente: {*get(null)*, *get(new Obect())*}.
La suite minimale ha evidenziato un valore di Statement Coverage del 82%, di Branch Coverage del 50% e Mutation score del $\frac{|D|}{|M|-|E|} = \frac{2}{6} \cong 33\%$, come si evince dalla Figura 41, Figura 57 e Figura 58.
Analizzando il codice, si è potuto migliorare la test suite, aggiungendo ulteriori test in modo tale da raggiungere delle percentuali più alte: 100%, 100% e 100%, come si evince dalla Figura 42, Figura 59 e Figura 60.

Version	Filename	LOC_added	MAX_LOC_added	AVG_LOC_addded	Churn	MAX_Churn	AVG_Churn	ChgSetSize	MAX_ChgSetSize	AVG_ChgSetSize
14	bookkeeper-server/src/main/java/org/apache/bookkeeper/bookie/storage/ldb/WriteCache.java	7886	303	14.9	306	303	577.358	61880	1650	11.7
14	bookkeeper-server/src/main/java/org/apache/bookkeeper/proto/checksum/DigestManager.java	4390	254	10.7	21	254	0.51219714	45792	1650	1.12

Figura 1 Misure prese in considerazione delle classi WriteCache e DigestManager, del progetto BookKeeper.

	A	B	C	D	E	F	G	H	I	J	K
1	Version	Filename	LOC_adde	MAX_LOC	AVG_LOC_addded	Churn	MAX_Chu	AVG_	ChgSetSiz	MAX_Chg	AVG_ChgSetSize
2	2	openjpa-kernel/src/main/java/org/apache/openjpa/util/ProxyManagerImpl.java	1585	1261	14.0	1339	1116	12.0	1166	1123	106.0
3	3	openjpa-kernel/src/main/java/org/apache/openjpa/util/ProxyManagerImpl.java	80	37	16.0	-6	19	-12	287	97	57.4
4	4	openjpa-kernel/src/main/java/org/apache/openjpa/util/ProxyManagerImpl.java	54	26	18.0	44	26	14.0	8	5	26.0
5	6	openjpa-kernel/src/main/java/org/apache/openjpa/util/ProxyManagerImpl.java	1662	1662	1662.0	-10	0	-10.0	76	76	76.0
6	9	openjpa-kernel/src/main/java/org/apache/openjpa/util/ProxyManagerImpl.java	14	14	14.0	2	2	2.0	72	72	72.0
7	11	openjpa-kernel/src/main/java/org/apache/openjpa/util/ProxyManagerImpl.java	1669	1664	834.5	0	0	0.0	1716	1543	858.0
8	12	openjpa-kernel/src/main/java/org/apache/openjpa/util/ProxyManagerImpl.java	13	13	13.0	2	2	2.0	12	12	12.0

Figura 2 Misure prese in considerazione della classe ProxyManagerImpl, del progetto OpenJPA.

13	1	openjpa-kernel/src/main/java/org/apache/openjpa/util/CacheMap.java	1	1	1.0	0	0	0.0	221	221	221.0
14	2	openjpa-kernel/src/main/java/org/apache/openjpa/util/CacheMap.java	13	13	13.0	3	3	3.0	1123	1123	1123.0
15	6	openjpa-kernel/src/main/java/org/apache/openjpa/util/CacheMap.java	645	645	645.0	6	6	6.0	76	76	76.0
16	7	openjpa-kernel/src/main/java/org/apache/openjpa/util/CacheMap.java	5	3	2.5	-3	0	-1.5	1	1	0.5
17	9	openjpa-kernel/src/main/java/org/apache/openjpa/util/CacheMap.java	8	8	8.0	6	6	6.0	10	10	10.0
18	11	openjpa-kernel/src/main/java/org/apache/openjpa/util/CacheMap.java	648	648	648.0	0	0	0.0	1543	1543	1543.0

Figura 3 Misure prese in considerazione della classe CacheMap, del progetto OpenJPA.

WriteCache

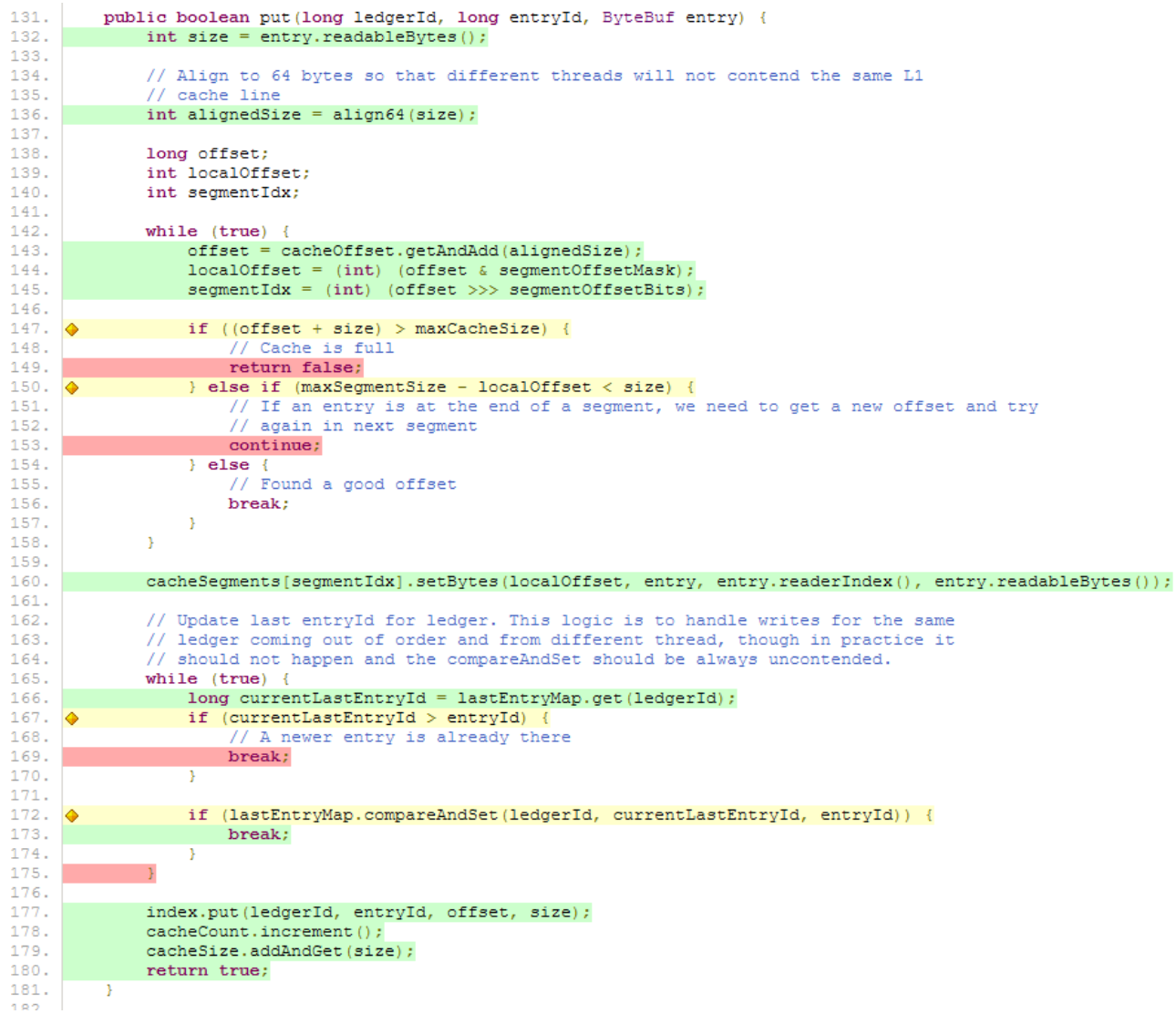
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)		0%		0%	8	8	32	32	1	1
lambda$forEach$0(long, long, long, long)		0%		0%	2	2	8	8	1	1
clear()		0%		n/a	1	1	7	7	1	1
close()		0%		0%	2	2	3	3	1	1
getLastEntry(long)		0%		0%	2	2	4	4	1	1
isEmpty()		0%		0%	2	2	1	1	1	1
WriteCache(ByteBufAllocator, long)		0%		n/a	1	1	2	2	1	1
deleteLedger(long)		0%		n/a	1	1	2	2	1	1
put(long, long, ByteBuf)		94%		50%	4	5	4	20	0	1
size()		0%		n/a	1	1	1	1	1	1
WriteCache(ByteBufAllocator, long, int)		98%		66%	2	4	0	25	0	1
get(long, long)		95%		50%	1	2	1	10	0	1
alignToPowerOfTwo(long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
align64(int)		100%		n/a	0	1	0	1	0	1
count()		100%		n/a	0	1	0	1	0	1
Total	313 of 617	49%	29 of 38	23%	27	35	65	120	9	16

Figura 4 Percentuale Statement e Branch Coverage dei metodi put e get con la test suite minimale.

WriteCache

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
forEach(WriteCache.EntryConsumer)		0%		0%	8	8	32	32	1	1
lambda$forEach$0(long, long, long, long)		0%		0%	2	2	8	8	1	1
clear()		0%		n/a	1	1	7	7	1	1
close()		0%		0%	2	2	3	3	1	1
getLastEntry(long)		0%		0%	2	2	4	4	1	1
isEmpty()		0%		0%	2	2	1	1	1	1
WriteCache(ByteBufAllocator, long)		0%		n/a	1	1	2	2	1	1
deleteLedger(long)		0%		n/a	1	1	2	2	1	1
size()		0%		n/a	1	1	1	1	1	1
WriteCache(ByteBufAllocator, long, int)		98%		66%	2	4	0	25	0	1
put(long, long, ByteBuf)		98%		87%	1	5	1	20	0	1
get(long, long)		100%		100%	0	2	0	10	0	1
alignToPowerOfTwo(long)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
align64(int)		100%		n/a	0	1	0	1	0	1
count()		100%		n/a	0	1	0	1	0	1
Total	307 of 617	50%	25 of 38	34%	23	35	61	120	9	16

Figura 5 Percentuale Statement e Branch Coverage dei metodi put e get dopo l'ampiamento della test suite minimale.



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

Figura 6 Statement e Branch Coverage del metodo put con la test suite minimale.

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

Figura 7 Mutation testing del metodo put con la test suite minimale.

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

Figura 8 Statement e Branch Coverage del metodo put dopo l'ampiamento della test suite minimale.

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

Figura 9 Mutation testing del metodo put dopo l'ampiamento della test suite minimale.

```
183.     public ByteBuf get(long ledgerId, long entryId) {
184.         LongPair result = index.get(ledgerId, entryId);
185.         if (result == null) {
186.             return null;
187.         }
188.
189.         long offset = result.first;
190.         int size = (int) result.second;
191.         ByteBuf entry = allocator.buffer(size, size);
192.
193.         int localOffset = (int) (offset & segmentOffsetMask);
194.         int segmentIdx = (int) (offset >>> segmentOffsetBits);
195.         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196.         return entry;
197.     }
198.
```

Figura 10 Statement e Branch Coverage del metodo get con la test suite minimale.

```
182
183     public ByteBuf get(long ledgerId, long entryId) {
184         LongPair result = index.get(ledgerId, entryId);
185 1       if (result == null) {
186             return null;
187         }
188
189         long offset = result.first;
190         int size = (int) result.second;
191         ByteBuf entry = allocator.buffer(size, size);
192
193 1       int localOffset = (int) (offset & segmentOffsetMask);
194 1       int segmentIdx = (int) (offset >>> segmentOffsetBits);
195         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196 1       return entry;
197     }
198
```

Figura 11 Mutation testing del metodo get con la test suite minimale.

```
183.    public ByteBuf get(long ledgerId, long entryId) {
184.        LongPair result = index.get(ledgerId, entryId);
185.        if (result == null) {
186.            return null;
187.        }
188.
189.        long offset = result.first;
190.        int size = (int) result.second;
191.        ByteBuf entry = allocator.buffer(size, size);
192.
193.        int localOffset = (int) (offset & segmentOffsetMask);
194.        int segmentIdx = (int) (offset >>> segmentOffsetBits);
195.        entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
196.        return entry;
197.    }
198
```

Figura 12 Statement e Branch Coverage del metodo get dopo l'ampiamento della test suite minimale.

DigestManager

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
verifyDigest(long, ByteBuf, boolean)		61%		60%	4	6	7	22	0	1
verifyDigestAndReturnLac(ByteBuf)		49%		50%	3	4	7	17	0	1
instantiate(long, byte[], DataFormats.LedgerMetadataFormat.DigestType, ByteBufAllocator, boolean)		43%		40%	3	5	3	6	0	1
verifyDigestAndReturnLastConfirmed(ByteBuf)		0%		n/a	1	1	6	6	1	1
computeDigestAndPackageForSending(long, long, long, ByteBuf)		81%		50%	1	2	1	11	0	1
update(byte[])		0%		n/a	1	1	2	2	1	1
computeDigestAndPackageForSendingLac(long)		83%		50%	1	2	1	8	0	1
verifyDigest(ByteBuf)		0%		n/a	1	1	2	2	1	1
DigestManager(long, boolean, ByteBufAllocator)		100%		n/a	0	1	0	6	0	1
verifyDigestAndReturnData(long, ByteBuf)		100%		n/a	0	1	0	3	0	1
generateMasterKey(byte[])		100%		100%	0	2	0	1	0	1
verifyDigest(long, ByteBuf)		100%		n/a	0	1	0	2	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	175 of 441	60%	12 of 27	55%	15	28	29	87	3	13

Figura 13 Percentuale Statement e Branch Coverage dei metodi verifyDigest, verifyDigestAndReturnLac, verifyDigestAndReturnData e generateMasterKey con la test suite minimale.

DigestManager

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
verifyDigestAndReturnLastConfirmed(ByteBuf)		0%		n/a	1	1	6	6	1	1
instantiate(long, byte[], DataFormats.LedgerMetadataFormat.DigestType, ByteBufAllocator, boolean)		73%		80%	1	5	1	6	0	1
computeDigestAndPackageForSending(long, long, long, ByteBuf)		81%		50%	1	2	1	11	0	1
update(byte[])		0%		n/a	1	1	2	2	1	1
verifyDigest(ByteBuf)		0%		n/a	1	1	2	2	1	1
verifyDigest(long, ByteBuf, boolean)		100%		90%	1	6	0	22	0	1
verifyDigestAndReturnLac(ByteBuf)		100%		100%	0	4	0	17	0	1
computeDigestAndPackageForSendingLac(long)		100%		100%	0	2	0	8	0	1
DigestManager(long, boolean, ByteBufAllocator)		100%		n/a	0	1	0	6	0	1
verifyDigestAndReturnData(long, ByteBuf)		100%		n/a	0	1	0	3	0	1
generateMasterKey(byte[])		100%		100%	0	2	0	1	0	1
verifyDigest(long, ByteBuf)		100%		n/a	0	1	0	2	0	1
static {...}		100%		n/a	0	1	0	1	0	1
Total	57 of 441	87%	3 of 27	88%	6	28	12	87	3	13

Figura 14 Percentuale Statement e Branch Coverage dei metodi verifyDigest, verifyDigestAndReturnLac, verifyDigestAndReturnData e generateMasterKey dopo l'ampiamento della test suite minimale.

```
86.
87.     public static byte[] generateMasterKey(byte[] password) throws NoSuchAlgorithmException {
88.         return password.length > 0 ? MacDigestManager.genDigest("ledger", password) : MacDigestManager.EMPTY_LEDGER_KEY;
89.     }
90.
```

Figura 15 Statement e Branch Coverage del metodo generateMasterKey con la test suite minimale.

```
87      public static byte[] generateMasterKey(byte[] password) throws NoSuchAlgorithmException {
88  3       return password.length > 0 ? MacDigestManager.genDigest("ledger", password) : MacDigestManager.EMPTY_LEDGER_KEY;
89      }
```

Figura 16 Mutation testing del metodo generateMasterKey con la test suite minimale.

```
228.     /**
229.      * Verify that the digest matches and returns the data in the entry.
230.      *
231.      * @param entryId
232.      * @param dataReceived
233.      * @return
234.      * @throws BKDigestMatchException
235.      */
236.     public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)
237.             throws BKDigestMatchException {
238.         verifyDigest(entryId, dataReceived);
239.         dataReceived.readerIndex(METADATA_LENGTH + macCodeLength);
240.         return dataReceived;
241.     }
```

Figura 17 Statement e Branch Coverage del metodo verifyDigestAndReturnData con la test suite minimale.

```
236      public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)
237              throws BKDigestMatchException {
238  1       verifyDigest(entryId, dataReceived);
239  1       dataReceived.readerIndex(METADATA_LENGTH + macCodeLength);
240  1       return dataReceived;
241      }
```

Figura 18 Mutation testing del metodo verifyDigestAndReturnData con la test suite minimale.

```
236      public ByteBuf verifyDigestAndReturnData(long entryId, ByteBuf dataReceived)
237              throws BKDigestMatchException {
238 1        verifyDigest(entryId, dataReceived);
239 1        dataReceived.readerIndex(METADATA_LENGTH + macCodeLength);
240 1        return dataReceived;
241      }
```

Figura 19 Mutation testing del metodo verifyDigestAndReturnData dopo l'ampiamento della test suite minimale.

```
195.    public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
196.        if ((LAC_METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
197.            logger.error("Data received is smaller than the minimum for this digest type."
198.                    + " Either the packet it corrupt, or the wrong digest is configured. "
199.                    + " Digest type: {}, Packet Length: {}",
200.                    this.getClass().getName(), dataReceived.readableBytes());
201.            throw new BKDigestMatchException();
202.        }
203.
204.        update(dataReceived.slice(0, LAC_METADATA_LENGTH));
205.
206.        ByteBuf digest = allocator.buffer(macCodeLength);
207.        try {
208.            populateValueAndReset(digest);
209.
210.            if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
211.                logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
212.                throw new BKDigestMatchException();
213.            }
214.        } finally {
215.            digest.release();
216.        }
217.
218.        long actualLedgerId = dataReceived.readLong();
219.        long lac = dataReceived.readLong();
220.        if (actualLedgerId != ledgerId) {
221.            logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
222.                    + actualLedgerId);
223.            throw new BKDigestMatchException();
224.        }
225.        return lac;
226.    }
```

Figura 20 Statement e Branch Coverage del metodo verifyDigestAndReturnLac con la test suite minimale.

```
195      public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
196 3        if ((LAC_METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
197              logger.error("Data received is smaller than the minimum for this digest type."
198                      + " Either the packet it corrupt, or the wrong digest is configured. "
199                      + " Digest type: {}, Packet Length: {}",
200                      this.getClass().getName(), dataReceived.readableBytes());
201              throw new BKDigestMatchException();
202          }
203
204 1        update(dataReceived.slice(0, LAC_METADATA_LENGTH));
205
206          ByteBuf digest = allocator.buffer(macCodeLength);
207          try {
208 1            populateValueAndReset(digest);
209
210 1            if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
211                  logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
212                  throw new BKDigestMatchException();
213              }
214          } finally {
215              digest.release();
216          }
217
218          long actualLedgerId = dataReceived.readLong();
219          long lac = dataReceived.readLong();
220 1        if (actualLedgerId != ledgerId) {
221              logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
222                      + actualLedgerId);
223              throw new BKDigestMatchException();
224          }
225 1        return lac;
226      }
227
```

Figura 21 Mutation testing del metodo verifyDigestAndReturnLac con la test suite minimale.

```
194.
195.    public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
196.        if ((LAC_METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
197.            logger.error("Data received is smaller than the minimum for this digest type."
198.                    + " Either the packet it corrupt, or the wrong digest is configured. "
199.                    + " Digest type: {}, Packet Length: {}",
200.                    this.getClass().getName(), dataReceived.readableBytes());
201.            throw new BKDigestMatchException();
202.        }
203.
204.        update(dataReceived.slice(0, LAC_METADATA_LENGTH));
205.
206.        ByteBuf digest = allocator.buffer(macCodeLength);
207.        try {
208.            populateValueAndReset(digest);
209.
210.            if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
211.                logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
212.                throw new BKDigestMatchException();
213.            }
214.        } finally {
215.            digest.release();
216.        }
217.
218.        long actualLedgerId = dataReceived.readLong();
219.        long lac = dataReceived.readLong();
220.        if (actualLedgerId != ledgerId) {
221.            logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
222.                         + actualLedgerId);
223.            throw new BKDigestMatchException();
224.        }
225.        return lac;
226.    }
227.
```

Figura 22 Statement e Branch Coverage del metodo verifyDigestAndReturnLac dopo l'ampiamento della test suite minimale.

```
195     public long verifyDigestAndReturnLac(ByteBuf dataReceived) throws BKDigestMatchException{
196 3       if ((LAC_METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
197             logger.error("Data received is smaller than the minimum for this digest type."
198                     + " Either the packet it corrupt, or the wrong digest is configured. "
199                     + " Digest type: {}, Packet Length: {}",
200                     this.getClass().getName(), dataReceived.readableBytes());
201             throw new BKDigestMatchException();
202         }
203
204 1       update(dataReceived.slice(0, LAC_METADATA_LENGTH));
205
206         ByteBuf digest = allocator.buffer(macCodeLength);
207         try {
208 1           populateValueAndReset(digest);
209
210 1           if (digest.compareTo(dataReceived.slice(LAC_METADATA_LENGTH, macCodeLength)) != 0) {
211                 logger.error("Mac mismatch for ledger-id LAC: " + ledgerId);
212                 throw new BKDigestMatchException();
213             }
214         } finally {
215             digest.release();
216         }
217
218         long actualLedgerId = dataReceived.readLong();
219         long lac = dataReceived.readLong();
220 1       if (actualLedgerId != ledgerId) {
221             logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
222                          + actualLedgerId);
223             throw new BKDigestMatchException();
224         }
225 1       return lac;
226     }
```

Figura 23 Mutation testing del metodo verifyDigestAndReturnLac dopo l'ampiamento della test suite minimale.

```
151.    private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
152.            throws BKDigestMatchException {
153.
154.        if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
155.            logger.error("Data received is smaller than the minimum for this digest type. "
156.                    + " Either the packet it corrupt, or the wrong digest is configured. "
157.                    + " Digest type: {}, Packet Length: {}",
158.                    this.getClass().getName(), dataReceived.readableBytes());
159.            throw new BKDigestMatchException();
160.        }
161.        update(dataReceived.slice(0, METADATA_LENGTH));
162.
163.        int offset = METADATA_LENGTH + macCodeLength;
164.        update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
165.
166.        ByteBuf digest = allocator.buffer(macCodeLength);
167.        populateValueAndReset(digest);
168.
169.        try {
170.            if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
171.                logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
172.                throw new BKDigestMatchException();
173.            }
174.        } finally {
175.            digest.release();
176.        }
177.
178.        long actualLedgerId = dataReceived.readLong();
179.        long actualEntryId = dataReceived.readLong();
180.
181.        if (actualLedgerId != ledgerId) {
182.            logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
183.                    + actualLedgerId);
184.            throw new BKDigestMatchException();
185.        }
186.
187.        if (!skipEntryIdCheck && actualEntryId != entryId) {
188.            logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
189.                    + actualEntryId);
190.            throw new BKDigestMatchException();
191.        }
192.
193.    }
```

Figura 24 Statement e Branch Coverage del metodo verifyDigest con la test suite minimale.

```
151      private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
152              throws BKDigestMatchException {
153
154 3        if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
155              logger.error("Data received is smaller than the minimum for this digest type. "
156                      + " Either the packet it corrupt, or the wrong digest is configured. "
157                      + " Digest type: {}, Packet Length: {}",
158                      this.getClass().getName(), dataReceived.readableBytes());
159              throw new BKDigestMatchException();
160          }
161 1        update(dataReceived.slice(0, METADATA_LENGTH));
162
163 1        int offset = METADATA_LENGTH + macCodeLength;
164 2        update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
165
166          ByteBuf digest = allocator.buffer(macCodeLength);
167 1        populateValueAndReset(digest);
168
169          try {
170 1            if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
171                  logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
172                  throw new BKDigestMatchException();
173              }
174          } finally {
175              digest.release();
176          }
177
178          long actualLedgerId = dataReceived.readLong();
179          long actualEntryId = dataReceived.readLong();
180
181 1        if (actualLedgerId != ledgerId) {
182              logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
183                          + actualLedgerId);
184              throw new BKDigestMatchException();
185          }
186
187 2        if (!skipEntryIdCheck && actualEntryId != entryId) {
188              logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
189                          + actualEntryId);
190              throw new BKDigestMatchException();
191          }
192
193      }
194
```

Figura 25 Mutation testing del metodo verifyDigest con la test suite minimale.

```
153.
154.         if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
155.             logger.error("Data received is smaller than the minimum for this digest type. "
156.                     + " Either the packet it corrupt, or the wrong digest is configured. "
157.                     + " Digest type: {}, Packet Length: {}",
158.                     this.getClass().getName(), dataReceived.readableBytes());
159.             throw new BKDigestMatchException();
160.         }
161.         update(dataReceived.slice(0, METADATA_LENGTH));
162.
163.         int offset = METADATA_LENGTH + macCodeLength;
164.         update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
165.
166.         ByteBuf digest = allocator.buffer(macCodeLength);
167.         populateValueAndReset(digest);
168.
169.         try {
170.             if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
171.                 logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
172.                 throw new BKDigestMatchException();
173.             }
174.         } finally {
175.             digest.release();
176.         }
177.
178.         long actualLedgerId = dataReceived.readLong();
179.         long actualEntryId = dataReceived.readLong();
180.
181.         if (actualLedgerId != ledgerId) {
182.             logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
183.                     + actualLedgerId);
184.             throw new BKDigestMatchException();
185.         }
186.
187.         if (!skipEntryIdCheck && actualEntryId != entryId) {
188.             logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
189.                     + actualEntryId);
190.             throw new BKDigestMatchException();
191.         }
192.
193.     }
```

Figura 26 Statement e Branch Coverage del metodo verifyDigest dopo l'ampiamento della test suite minimale.

```
147      private void verifyDigest(long entryId, ByteBuf dataReceived) throws BKDigestMatchException {
148 1        verifyDigest(entryId, dataReceived, false);
149      }
150
151      private void verifyDigest(long entryId, ByteBuf dataReceived, boolean skipEntryIdCheck)
152              throws BKDigestMatchException {
153
154 3        if ((METADATA_LENGTH + macCodeLength) > dataReceived.readableBytes()) {
155              logger.error("Data received is smaller than the minimum for this digest type. "
156                      + " Either the packet it corrupt, or the wrong digest is configured. "
157                      + " Digest type: {}, Packet Length: {}",
158                      this.getClass().getName(), dataReceived.readableBytes());
159              throw new BKDigestMatchException();
160          }
161 1        update(dataReceived.slice(0, METADATA_LENGTH));
162
163 1        int offset = METADATA_LENGTH + macCodeLength;
164 2        update(dataReceived.slice(offset, dataReceived.readableBytes() - offset));
165
166          ByteBuf digest = allocator.buffer(macCodeLength);
167 1        populateValueAndReset(digest);
168
169          try {
170 1            if (digest.compareTo(dataReceived.slice(METADATA_LENGTH, macCodeLength)) != 0) {
171                  logger.error("Mac mismatch for ledger-id: " + ledgerId + ", entry-id: " + entryId);
172                  throw new BKDigestMatchException();
173              }
174          } finally {
175              digest.release();
176          }
177
178          long actualLedgerId = dataReceived.readLong();
179          long actualEntryId = dataReceived.readLong();
180
181 1        if (actualLedgerId != ledgerId) {
182              logger.error("Ledger-id mismatch in authenticated message, expected: " + ledgerId + " , actual: "
183                          + actualLedgerId);
184              throw new BKDigestMatchException();
185          }
186
187 2        if (!skipEntryIdCheck && actualEntryId != entryId) {
188              logger.error("Entry-id mismatch in authenticated message, expected: " + entryId + " , actual: "
189                          + actualEntryId);
190              throw new BKDigestMatchException();
191          }
192
193      }
```

Figura 27 Mutation testing del metodo verifyDigest dopo l'ampiamento della test suite minimale.

ProxyManagerImpl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
nextProxyId()		100%		n/a	0	1	0	1	0	1
startsWith(String, String)		100%		83%	1	4	0	3	0	1
toHelperParameters(Class[], Class)		100%		n/a	0	1	0	4	0	1
ProxyManagerImpl()		100%		n/a	0	1	0	8	0	1
copyArray(Object)		100%		100%	0	2	0	10	0	1
getProxyClassName(Class, boolean)		100%		100%	0	2	0	3	0	1
isSetter(Method)		100%		83%	2	7	0	6	0	1
getFactoryProxyDate(Class)		100%		50%	2	3	0	10	0	1
getFactoryProxyCollection(Class)		100%		50%	2	3	0	10	0	1
generateProxyDateBytecode(Class, boolean)		100%		n/a	0	1	0	12	0	1
static {...}		100%		n/a	0	1	0	13	0	1
generateProxyCollectionBytecode(Class, boolean)		100%		n/a	0	1	0	13	0	1

Figura 28 Percentuale Statement e Branch Coverage del metodo copyArray con la test suite minimale.

copyCollection(Collection)		0%		0%	3	3	6	6	1	1
addProxyBeanMethods(BCClass, Class, Constructor)		80%		50%	4	5	4	30	0	1
copyCustom(Object)		59%		62%	6	9	5	16	0	1
toProxyableCollectionType(Class)		35%		37%	4	5	5	9	0	1
getFactoryProxyBean(Object)		65%		50%	6	7	4	17	0	1
newMapProxy(Class, Class, Class, Comparator, boolean)		0%		0%	3	3	4	4	1	1
toConcreteType(Class, Map)		0%		0%	4	4	9	9	1	1
toProxyableMapType(Class)		0%		0%	5	5	9	9	1	1
getFactoryProxyCalendar(Class)		0%		0%	3	3	10	10	1	1
getFactoryProxyMap(Class)		0%		0%	3	3	10	10	1	1
generateProxyCalendarBytecode(Class, boolean)		0%		n/a	1	1	12	12	1	1
copyProperties(Class, Code)		48%		41%	5	7	11	20	0	1
loadDelayedProxy(Class)		0%		0%	9	9	15	15	1	1
generateProxyMapBytecode(Class, boolean)		0%		n/a	1	1	14	14	1	1
instantiateProxy(Class, Constructor, Object[])		11%		25%	2	3	13	16	0	1
addProxyCollectionMethods(BCClass, Class)		84%		44%	9	10	9	77	0	1
newCustomProxy(Object, boolean)		41%		50%	10	13	17	33	0	1
findGetter(Class, Method)		0%		0%	7	7	20	20	1	1
addProxyDateMethods(BCClass, Class)		52%		46%	13	14	24	59	0	1

Figura 29 Percentuale Statement e Branch Coverage dei metodi copyCustom e newCustomProxy con la test suite minimale.

ProxyManagerImpl

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
nextProxyId()		100%		n/a	0	1	0	1	0	1
newDateProxy(Class)		100%		n/a	0	1	0	2	0	1
allowsDuplicates(Class)		100%		100%	0	2	0	1	0	1
isOrdered(Class)		100%		75%	1	3	0	2	0	1
startsWith(String, String)		100%		83%	1	4	0	3	0	1
toHelperParameters(Class[], Class)		100%		n/a	0	1	0	4	0	1
ProxyManagerImpl()		100%		n/a	0	1	0	8	0	1
copyArray(Object)		100%		100%	0	2	0	10	0	1
getProxyClassName(Class, boolean)		100%		100%	0	2	0	3	0	1
isSetter(Method)		100%		91%	1	7	0	6	0	1
getFactoryProxyCalendar(Class)		100%		50%	2	3	0	10	0	1
getFactoryProxyDate(Class)		100%		50%	2	3	0	10	0	1
getFactoryProxyMap(Class)		100%		50%	2	3	0	10	0	1
getFactoryProxyCollection(Class)		100%		50%	2	3	0	10	0	1
generateProxyCalendarBytecode(Class, boolean)		100%		n/a	0	1	0	12	0	1
generateProxyDateBytecode(Class, boolean)		100%		n/a	0	1	0	12	0	1
static {...}		100%		n/a	0	1	0	13	0	1
generateProxyCollectionBytecode(Class, boolean)		100%		n/a	0	1	0	13	0	1
proxySetters(BCClass, Class)		100%		100%	0	6	0	9	0	1
toHelperAfterParameters(Class[], Class, Class)		100%		75%	3	7	0	15	0	1
generateProxyMapBytecode(Class, boolean)		100%		n/a	0	1	0	14	0	1
proxyOverrideMethod(BCClass, Method, Method, Class[])		100%		100%	0	2	0	12	0	1
copyCustom(Object)		100%		100%	0	9	0	16	0	1
delegateConstructors(BCClass, Class)		100%		100%	0	3	0	14	0	1
proxySetter(BCClass, Class, Method)		100%		100%	0	2	0	16	0	1
newCustomProxy(Object, boolean)		100%		100%	0	13	0	33	0	1
proxyBeforeAfterMethod(BCClass, Class, Method, Method, Class[], Method, Class[])		100%		90%	2	11	0	37	0	1
addProxyMethods(BCClass, boolean)		100%		100%	0	2	0	57	0	1

Figura 30 Percentuale Statement e Branch Coverage dei metodi copyArray, copyCustom e newCustomProxy dopo l'ampiamento della test suite minimale.

```
178.    @Override
179.    public Object copyArray(Object orig) {
180.        if (orig == null)
181.            return null;
182.
183.        try {
184.            int length = Array.getLength(orig);
185.            Object array = Array.newInstance(orig.getClass().
186.                getComponentType(), length);
187.
188.            System.arraycopy(orig, 0, array, 0, length);
189.            return array;
190.        } catch (Exception e) {
191.            throw new UnsupportedException(_loc.get("bad-array",
192.                e.getMessage()), e);
193.        }
194.    }
```

Figura 31 Statement e Branch Coverage del metodo copyArray con la test suite minimale.

```
178       @Override
179       public Object copyArray(Object orig) {
180   1       if (orig == null)
181               return null;
182
183           try {
184               int length = Array.getLength(orig);
185               Object array = Array.newInstance(orig.getClass().
186                   getComponentType(), length);
187
188   1           System.arraycopy(orig, 0, array, 0, length);
189   1           return array;
190           } catch (Exception e) {
191               throw new UnsupportedException(_loc.get("bad-array",
192                   e.getMessage()), e);
193           }
194       }
195
```

Figura 32 Mutation testing del metodo copyArray con la test suite minimale.

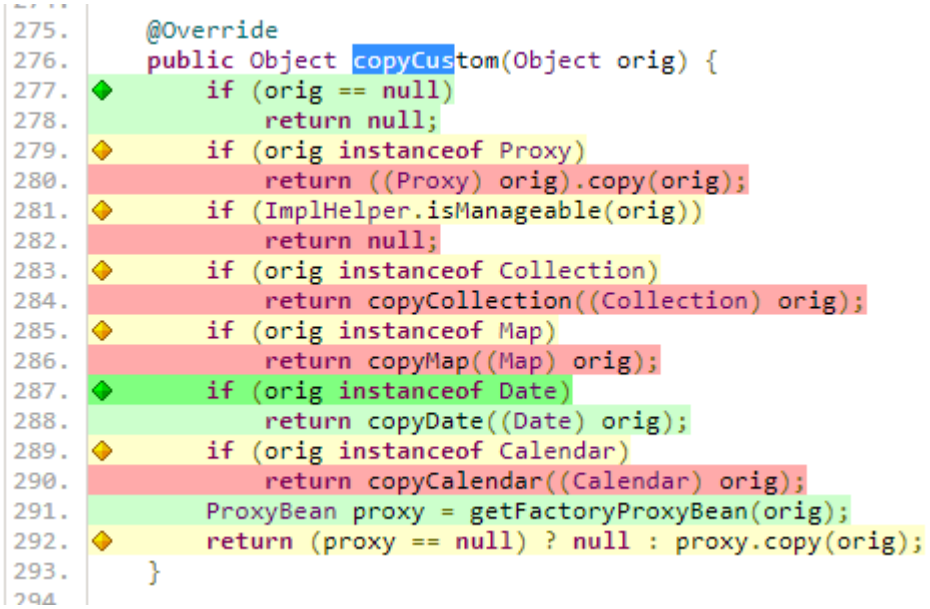


Figura 33 Statement e Branch Coverage del metodo copyCustom con la test suite minimale.

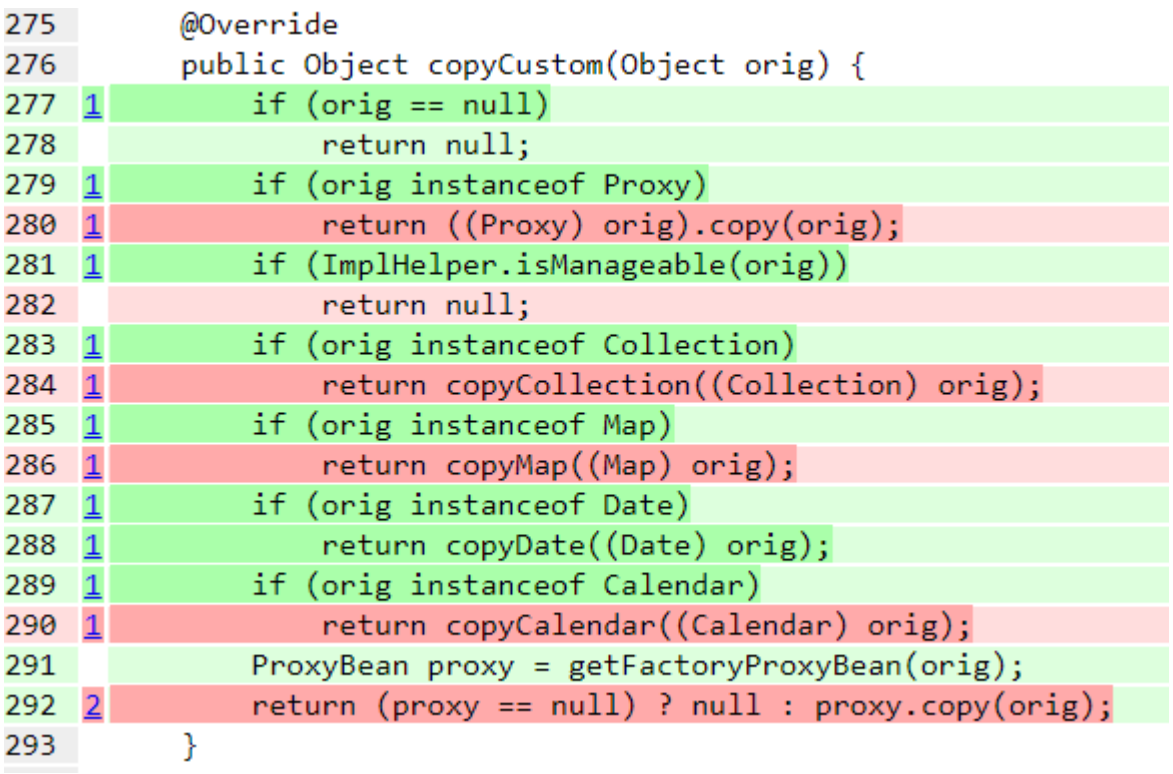
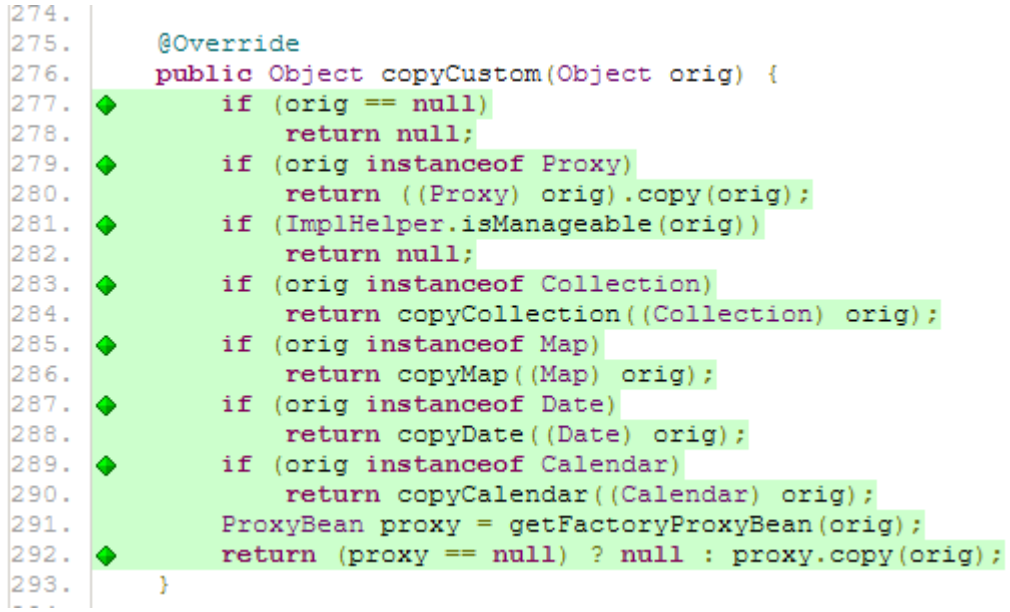


Figura 34 Mutation testing del metodo copyCustom con la test suite minimale



```
274.
275.    @Override
276.    public Object copyCustom(Object orig) {
277.        if (orig == null)
278.            return null;
279.        if (orig instanceof Proxy)
280.            return ((Proxy) orig).copy(orig);
281.        if (ImplHelper.isManageable(orig))
282.            return null;
283.        if (orig instanceof Collection)
284.            return copyCollection((Collection) orig);
285.        if (orig instanceof Map)
286.            return copyMap((Map) orig);
287.        if (orig instanceof Date)
288.            return copyDate((Date) orig);
289.        if (orig instanceof Calendar)
290.            return copyCalendar((Calendar) orig);
291.        ProxyBean proxy = getFactoryProxyBean(orig);
292.        return (proxy == null) ? null : proxy.copy(orig);
293.    }
```

Figura 35 Statement e Branch Coverage del metodo copyCustom dopo l'ampiamento della test suite minimale.

```
275      @Override
276      public Object copyCustom(Object orig) {
277  1       if (orig == null)
278              return null;
279  1       if (orig instanceof Proxy)
280  1           return ((Proxy) orig).copy(orig);
281  1       if (ImplHelper.isManageable(orig))
282              return null;
283  1       if (orig instanceof Collection)
284  1           return copyCollection((Collection) orig);
285  1       if (orig instanceof Map)
286  1           return copyMap((Map) orig);
287  1       if (orig instanceof Date)
288  1           return copyDate((Date) orig);
289  1       if (orig instanceof Calendar)
290  1           return copyCalendar((Calendar) orig);
291          ProxyBean proxy = getFactoryProxyBean(orig);
292  2       return (proxy == null) ? null : proxy.copy(orig);
293      }
```

Figura 36 Mutation testing del metodo copyCustom dopo l'ampiamento della test suite minimale.

```
295.    @Override
296.    public Proxy newCustomProxy(Object orig, boolean autoOff) {
297.        if (orig == null)
298.            return null;
299.        if (orig instanceof Proxy)
300.            return (Proxy) orig;
301.        if (ImplHelper.isManageable(orig))
302.            return null;
303.        if (!isProxyable(orig.getClass()))
304.            return null;
305.
306.        if (orig instanceof Collection) {
307.            Comparator comp = (orig instanceof SortedSet)
308.                ? ((SortedSet) orig).comparator() : null;
309.            Collection c = (Collection) newCollectionProxy(orig.getClass(),
310.                null, comp, autoOff);
311.            c.addAll((Collection) orig);
312.            return (Proxy) c;
313.        }
314.        if (orig instanceof Map) {
315.            Comparator comp = (orig instanceof SortedMap)
316.                ? ((SortedMap) orig).comparator() : null;
317.            Map m = (Map) newMapProxy(orig.getClass(), null, null, comp, autoOff);
318.            m.putAll((Map) orig);
319.            return (Proxy) m;
320.        }
321.        if (orig instanceof Date) {
322.            Date d = (Date) newDateProxy(orig.getClass());
323.            d.setTime(((Date) orig).getTime());
324.            if (orig instanceof Timestamp)
325.                ((Timestamp) d).setNanos(((Timestamp) orig).getNanos());
326.            return (Proxy) d;
327.        }
328.        if (orig instanceof Calendar) {
329.            Calendar c = (Calendar) newCalendarProxy(orig.getClass(),
330.                ((Calendar) orig).getTimeZone());
331.            c.setTimeInMillis(((Calendar) orig).getTimeInMillis());
332.            return (Proxy) c;
333.        }
334.
335.        ProxyBean proxy = getFactoryProxyBean(orig);
336.        return (proxy == null) ? null : proxy.newInstance(orig);
337.    }
338.
```

Figura 37 Statement e Branch Coverage del metodo newCustomProxy con la test suite minimale.

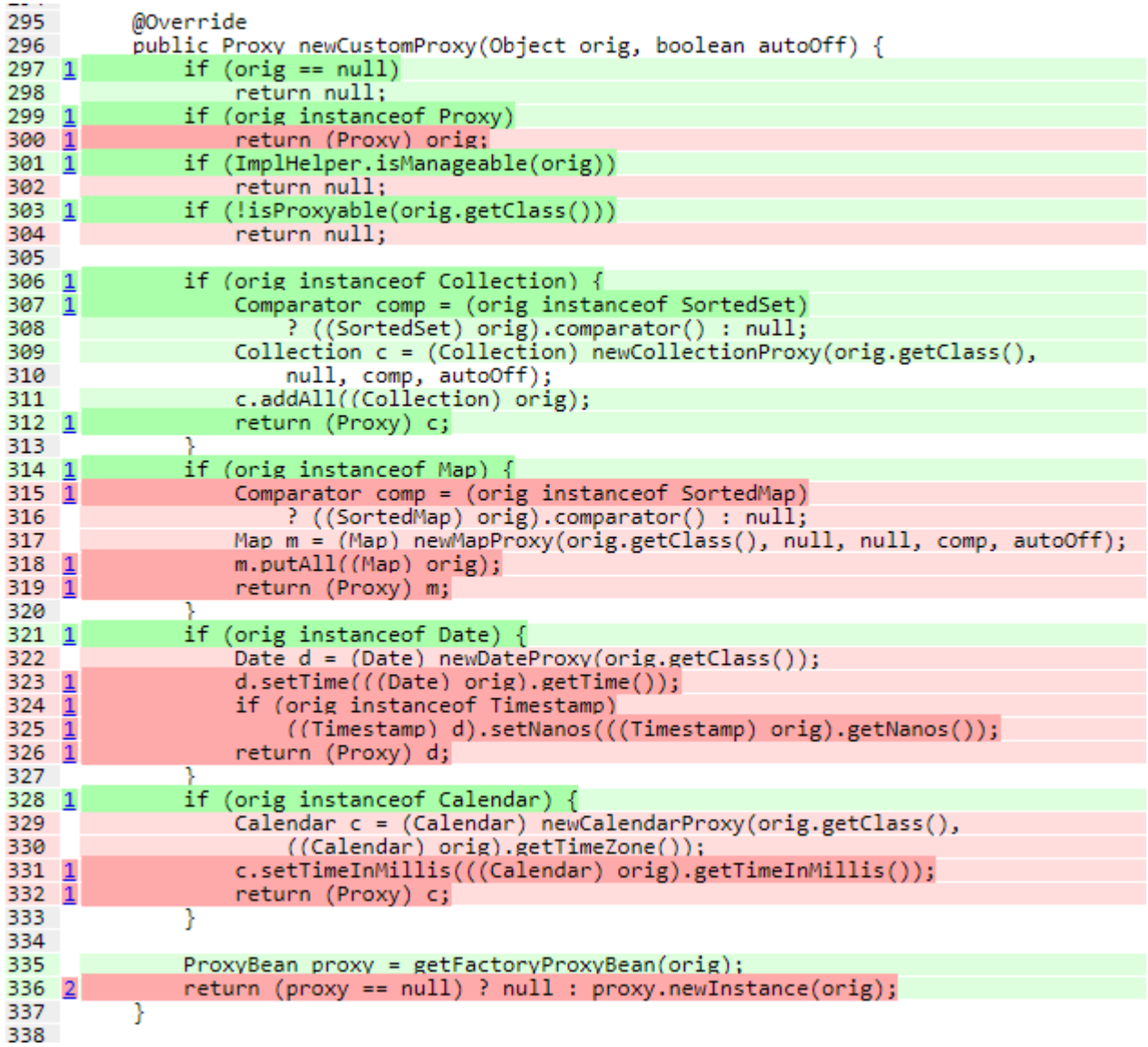
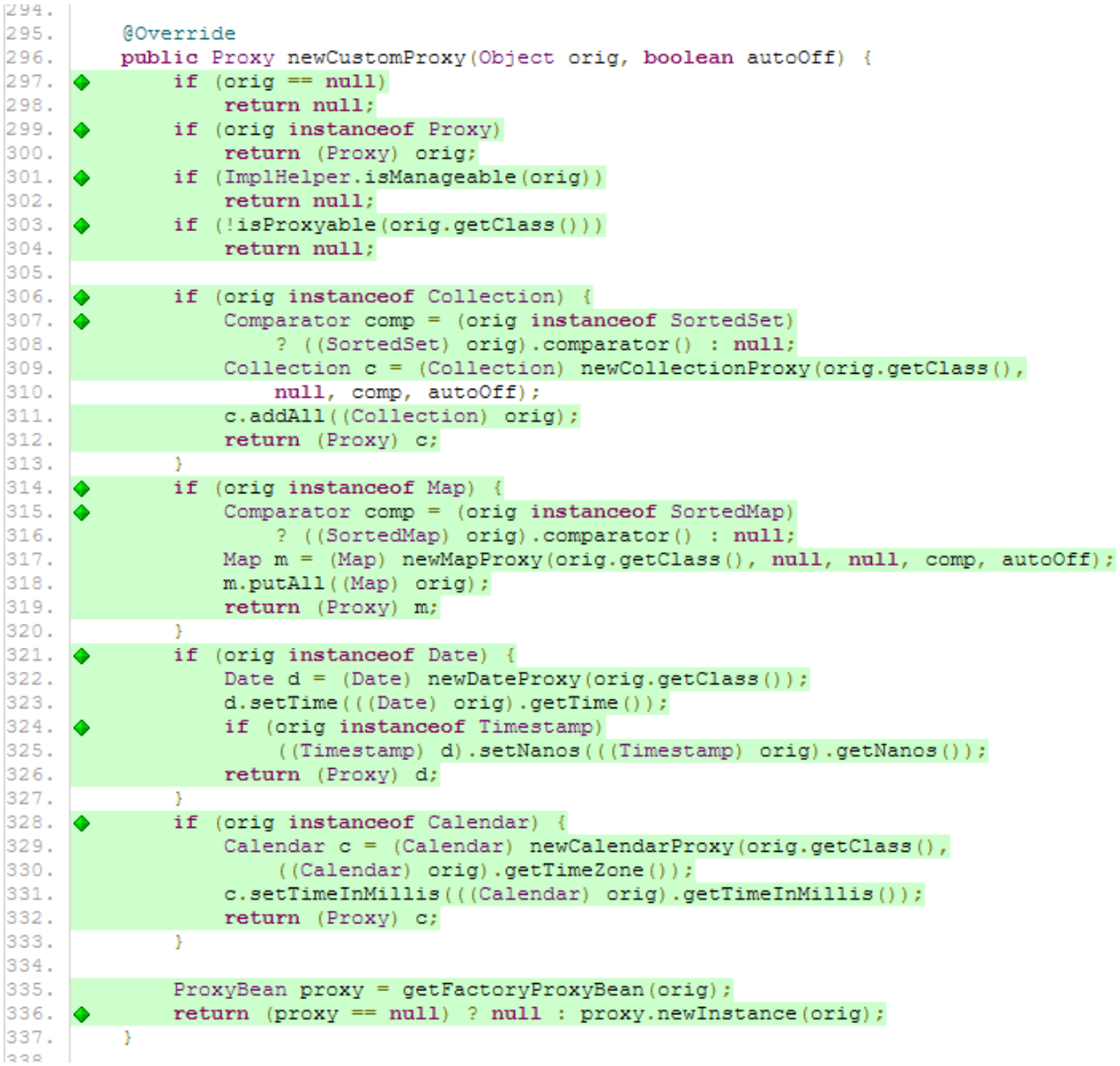


Figura 38 Mutation testing del metodo newCustomProxy con la test suite minimale.



```
294.
295.    @Override
296.    public Proxy newCustomProxy(Object orig, boolean autoOff) {
297.        if (orig == null)
298.            return null;
299.        if (orig instanceof Proxy)
300.            return (Proxy) orig;
301.        if (ImplHelper.isManageable(orig))
302.            return null;
303.        if (!isProxyable(orig.getClass()))
304.            return null;
305.
306.        if (orig instanceof Collection) {
307.            Comparator comp = (orig instanceof SortedSet)
308.                ? ((SortedSet) orig).comparator() : null;
309.            Collection c = (Collection) newCollectionProxy(orig.getClass(),
310.                null, comp, autoOff);
311.            c.addAll((Collection) orig);
312.            return (Proxy) c;
313.        }
314.        if (orig instanceof Map) {
315.            Comparator comp = (orig instanceof SortedMap)
316.                ? ((SortedMap) orig).comparator() : null;
317.            Map m = (Map) newMapProxy(orig.getClass(), null, null, comp, autoOff);
318.            m.putAll((Map) orig);
319.            return (Proxy) m;
320.        }
321.        if (orig instanceof Date) {
322.            Date d = (Date) newDateProxy(orig.getClass());
323.            d.setTime(((Date) orig).getTime());
324.            if (orig instanceof Timestamp)
325.                ((Timestamp) d).setNanos(((Timestamp) orig).getNanos());
326.            return (Proxy) d;
327.        }
328.        if (orig instanceof Calendar) {
329.            Calendar c = (Calendar) newCalendarProxy(orig.getClass(),
330.                ((Calendar) orig).getTimeZone());
331.            c.setTimeInMillis(((Calendar) orig).getTimeInMillis());
332.            return (Proxy) c;
333.        }
334.
335.        ProxyBean proxy = getFactoryProxyBean(orig);
336.        return (proxy == null) ? null : proxy.newInstance(orig);
337.    }
```

Figura 39 Statement e Branch Coverage del metodo newCustomProxy dopo l'ampiamento della test suite minimale.

```
295       @Override
296       public Proxy newCustomProxy(Object orig, boolean autoOff) {
297   1       if (orig == null)
298               return null;
299   1       if (orig instanceof Proxy)
300   1           return (Proxy) orig;
301   1       if (ImplHelper.isManageable(orig))
302               return null;
303   1       if (!isProxyable(orig.getClass()))
304               return null;
305
306   1       if (orig instanceof Collection) {
307   1           Comparator comp = (orig instanceof SortedSet)
308                   ? ((SortedSet) orig).comparator() : null;
309               Collection c = (Collection) newCollectionProxy(orig.getClass(),
310                   null, comp, autoOff);
311               c.addAll((Collection) orig);
312   1           return (Proxy) c;
313           }
314   1       if (orig instanceof Map) {
315   1           Comparator comp = (orig instanceof SortedMap)
316                   ? ((SortedMap) orig).comparator() : null;
317               Map m = (Map) newMapProxy(orig.getClass(), null, null, comp, autoOff);
318   1           m.putAll((Map) orig);
319   1           return (Proxy) m;
320           }
321   1       if (orig instanceof Date) {
322               Date d = (Date) newDateProxy(orig.getClass());
323   1           d.setTime(((Date) orig).getTime());
324   1           if (orig instanceof Timestamp)
325   1               ((Timestamp) d).setNanos(((Timestamp) orig).getNanos());
326   1           return (Proxy) d;
327           }
328   1       if (orig instanceof Calendar) {
329               Calendar c = (Calendar) newCalendarProxy(orig.getClass(),
330                   ((Calendar) orig).getTimeZone());
331   1           c.setTimeInMillis(((Calendar) orig).getTimeInMillis());
332   1           return (Proxy) c;
333           }
334
335       ProxyBean proxy = getFactoryProxyBean(orig);
336   2       return (proxy == null) ? null : proxy.newInstance(orig);
337       }
338
```

Figura 40 Mutation testing del metodo newCustomProxy dopo l'ampiamento della test suite minimale.

CacheMap

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
clear()		0%		n/a	1	1	10	10	1	1
putAll(Map, boolean)		0%		0%	4	4	5	5	1	1
containsKey(Object)		0%		0%	4	4	3	3	1	1
containsValue(Object)		0%		0%	4	4	3	3	1	1
toString()		0%		n/a	1	1	3	3	1	1
notifyEntryRemovals(Set)		0%		0%	3	3	5	5	1	1
remove(Object)		71%		62%	3	5	3	14	0	1
setCacheSize(int)		0%		0%	2	2	4	4	1	1
setSoftReferenceSize(int)		0%		0%	2	2	4	4	1	1
getCacheSize()		0%		0%	2	2	2	2	1	1
getSoftReferenceSize()		0%		0%	2	2	2	2	1	1
getPinnedKeys()		0%		n/a	1	1	3	3	1	1
put(Object, Object)		90%		90%	1	6	2	22	0	1
get(Object)		82%		50%	3	4	2	13	0	1
isEmpty()		0%		0%	2	2	1	1	1	1
softMapOverflowRemoved(Object, Object)		0%		n/a	1	1	2	2	1	1
softMapValueExpired(Object)		0%		n/a	1	1	2	2	1	1
keySet()		0%		n/a	1	1	1	1	1	1
values()		0%		n/a	1	1	1	1	1	1
entrySet()		0%		n/a	1	1	1	1	1	1
cacheMapOverflowRemoved(Object, Object)		76%		50%	1	2	1	4	0	1
putAll(Map)		0%		n/a	1	1	2	2	1	1
CacheMap(boolean, int, int, float, int)		94%		66%	2	4	2	16	0	1
isLRU()		0%		n/a	1	1	1	1	1	1
pin(Object)		96%		87%	1	5	0	12	0	1
unpin(Object)		100%		100%	0	2	0	8	0	1
size()		100%		n/a	0	1	0	3	0	1

Figura 41 Percentuale Statement e Branch Coverage dei metodi pin, unpin, put, remove e get con la test suite minimale.

CacheMap

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
entryAdded(Object, Object)		100%		n/a	0	1	0	1	0	1
entryRemoved(Object, Object, boolean)		100%		n/a	0	1	0	1	0	1
writeUnlock()		100%		n/a	0	1	0	2	0	1
writeLock()		100%		n/a	0	1	0	2	0	1
readUnlock()		100%		n/a	0	1	0	2	0	1
readLock()		100%		n/a	0	1	0	2	0	1
remove(Map, Object)		100%		n/a	0	1	0	1	0	1
put(Map, Object, Object)		100%		n/a	0	1	0	1	0	1
CacheMap(boolean)		100%		n/a	0	1	0	2	0	1
CacheMap()		100%		n/a	0	1	0	2	0	1
CacheMap(boolean, int, int, float)		100%		n/a	0	1	0	2	0	1
CacheMap(boolean, int)		100%		n/a	0	1	0	2	0	1
size()		100%		n/a	0	1	0	3	0	1
unpin(Object)		100%		100%	0	2	0	8	0	1
get(Object)		100%		100%	0	4	0	13	0	1
remove(Object)		100%		100%	0	5	0	14	0	1
pin(Object)		100%		100%	0	5	0	12	0	1
put(Object, Object)		100%		100%	0	6	0	22	0	1
isLRU()		0%		n/a	1	1	1	1	1	1

Figura 42 Percentuale Statement e Branch Coverage dei metodi unpin, pin, get, put e remove dopo l'ampiamento della test suite minimale.

```
291.    public boolean pin(Object key) {
292.        writeLock();
293.        try {
294.            // if we don't have a pinned map we need to create one; else if the
295.            // pinned map already contains the key, nothing to do
296.            if (pinnedMap.containsKey(key))
297.                return pinnedMap.get(key) != null;
298.
299.            // check other maps for key
300.            Object val = remove(cacheMap, key);
301.            if (val == null)
302.                val = remove(softMap, key);
303.
304.            // pin key
305.            put(pinnedMap, key, val);
306.            if (val != null) {
307.                _pinnedSize++;
308.                return true;
309.            }
310.            return false;
311.        } finally {
312.            writeUnlock();
313.        }
314.    }
```

Figura 43 Statement e Branch Coverage del metodo pin con la test suite minimale.

```
291      public boolean pin(Object key) {
292 1        writeLock();
293          try {
294              // if we don't have a pinned map we need to create one; else if the
295              // pinned map already contains the key, nothing to do
296 1            if (pinnedMap.containsKey(key))
297 3                return pinnedMap.get(key) != null;
298
299              // check other maps for key
300              Object val = remove(cacheMap, key);
301 1            if (val == null)
302                  val = remove(softMap, key);
303
304              // pin key
305              put(pinnedMap, key, val);
306 1            if (val != null) {
307 1                _pinnedSize++;
308 2                return true;
309              }
310 2            return false;
311          } finally {
312 1            writeUnlock();
313          }
314      }
315
```

Figura 44 Mutation testing del metodo pin con la test suite minimale.

```
291.     public boolean pin(Object key) {
292.         writeLock();
293.         try {
294.             // if we don't have a pinned map we need to create one; else if the
295.             // pinned map already contains the key, nothing to do
296.             if (pinnedMap.containsKey(key))
297.                 return pinnedMap.get(key) != null;
298.
299.             // check other maps for key
300.             Object val = remove(cacheMap, key);
301.             if (val == null)
302.                 val = remove(softMap, key);
303.
304.             // pin key
305.             put(pinnedMap, key, val);
306.             if (val != null) {
307.                 _pinnedSize++;
308.                 return true;
309.             }
310.             return false;
311.         } finally {
312.             writeUnlock();
313.         }
314.     }
```

Figura 45 Statement e Branch Coverage del metodo pin dopo l'ampiamento della test suite minimale.

```
282     /**
283      * Locks the given key and its value into the map. Objects pinned into
284      * the map are not counted towards the maximum cache size, and are never
285      * evicted implicitly. You may pin keys for which no value is in the map.
286      *
287      * @return true if the givne key's value was pinned; false if no value
288      * for the given key is cached
289      */
290
291     public boolean pin(Object key) {
292 1       writeLock();
293         try {
294             // if we don't have a pinned map we need to create one; else if the
295             // pinned map already contains the key, nothing to do
296 1           if (pinnedMap.containsKey(key))
297 3               return pinnedMap.get(key) != null;
298
299             // check other maps for key
300             Object val = remove(cacheMap, key);
301 1           if (val == null)
302                 val = remove(softMap, key);
303
304             // pin key
305             put(pinnedMap, key, val);
306 1           if (val != null) {
307 1               _pinnedSize++;
308 2               return true;
309             }
310 2           return false;
311         } finally {
312 1           writeUnlock();
313         }
314     }
```

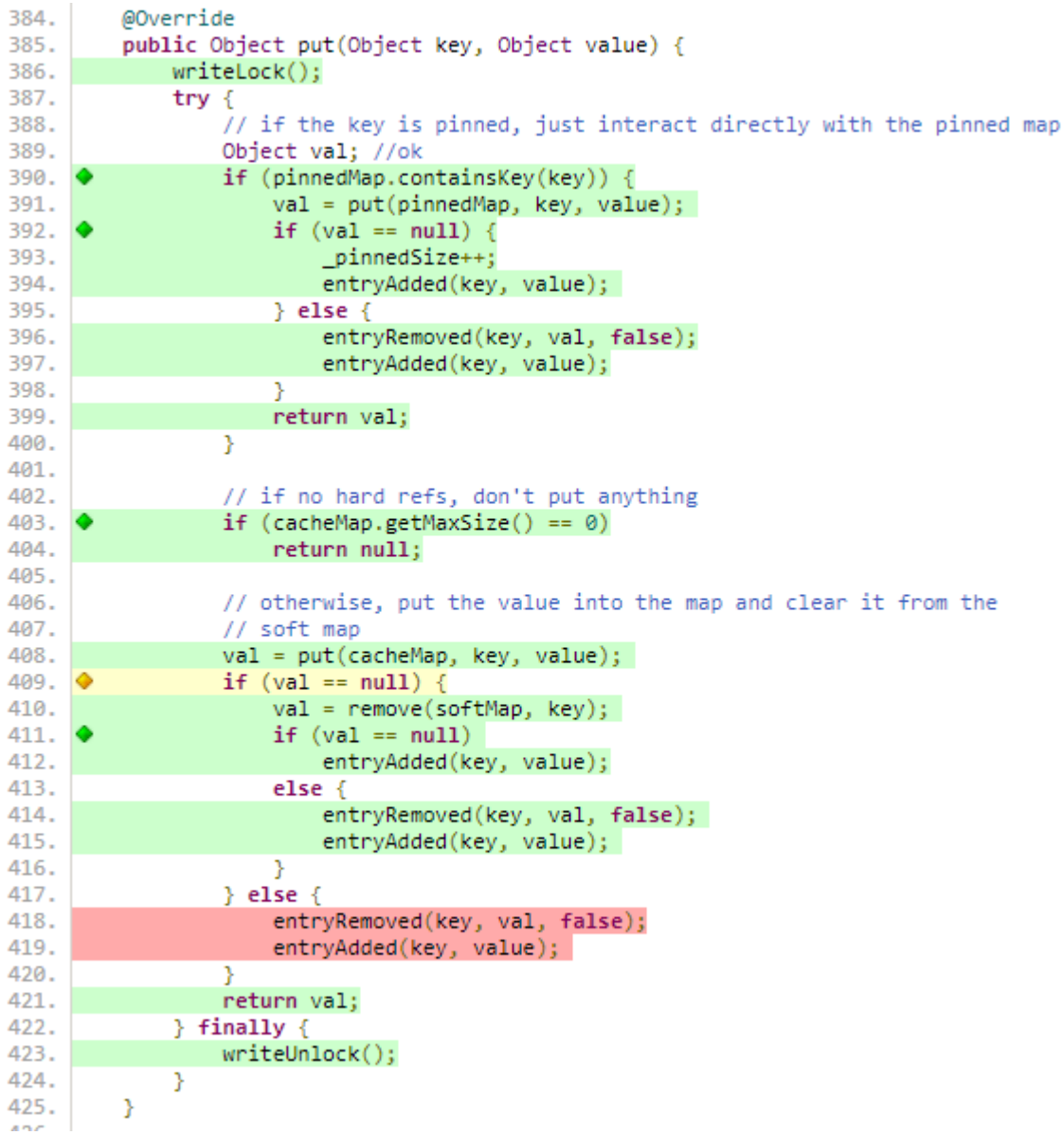
Figura 46 Mutation testing del metodo pin dopo l'ampiamento della test suite minimale.

```
319.    public boolean unpin(Object key) {
320.        writeLock();
321.        try {
322.            Object val = remove(pinnedMap, key);
323.            if (val != null) {
324.                // put back into unpinned cache
325.                put(key, val);
326.                _pinnedSize--;
327.                return true;
328.            }
329.            return false;
330.        } finally {
331.            writeUnlock();
332.        }
333.    }
334.
```

Figura 47 Statement e Branch Coverage del metodo unpin con la test suite minimale.

```
319      public boolean unpin(Object key) {
320 1        writeLock();
321          try {
322              Object val = remove(pinnedMap, key);
323 1            if (val != null) {
324                  // put back into unpinned cache
325                  put(key, val);
326 1                _pinnedSize--;
327 2                return true;
328              }
329 2            return false;
330          } finally {
331 1            writeUnlock();
332          }
333      }
334
```

Figura 48 Mutation testing del metodo unpin con la test suite minimale.



```
384.    @Override
385.    public Object put(Object key, Object value) {
386.        writeLock();
387.        try {
388.            // if the key is pinned, just interact directly with the pinned map
389.            Object val; //ok
390.            if (pinnedMap.containsKey(key)) {
391.                val = put(pinnedMap, key, value);
392.                if (val == null) {
393.                    _pinnedSize++;
394.                    entryAdded(key, value);
395.                } else {
396.                    entryRemoved(key, val, false);
397.                    entryAdded(key, value);
398.                }
399.                return val;
400.            }
401.
402.            // if no hard refs, don't put anything
403.            if (cacheMap.getMaxSize() == 0)
404.                return null;
405.
406.            // otherwise, put the value into the map and clear it from the
407.            // soft map
408.            val = put(cacheMap, key, value);
409.            if (val == null) {
410.                val = remove(softMap, key);
411.                if (val == null)
412.                    entryAdded(key, value);
413.                else {
414.                    entryRemoved(key, val, false);
415.                    entryAdded(key, value);
416.                }
417.            } else {
418.                entryRemoved(key, val, false);
419.                entryAdded(key, value);
420.            }
421.            return val;
422.        } finally {
423.            writeUnlock();
424.        }
425.    }
```

Figura 49 Statement e Branch Coverage del metodo put con la test suite minimale.

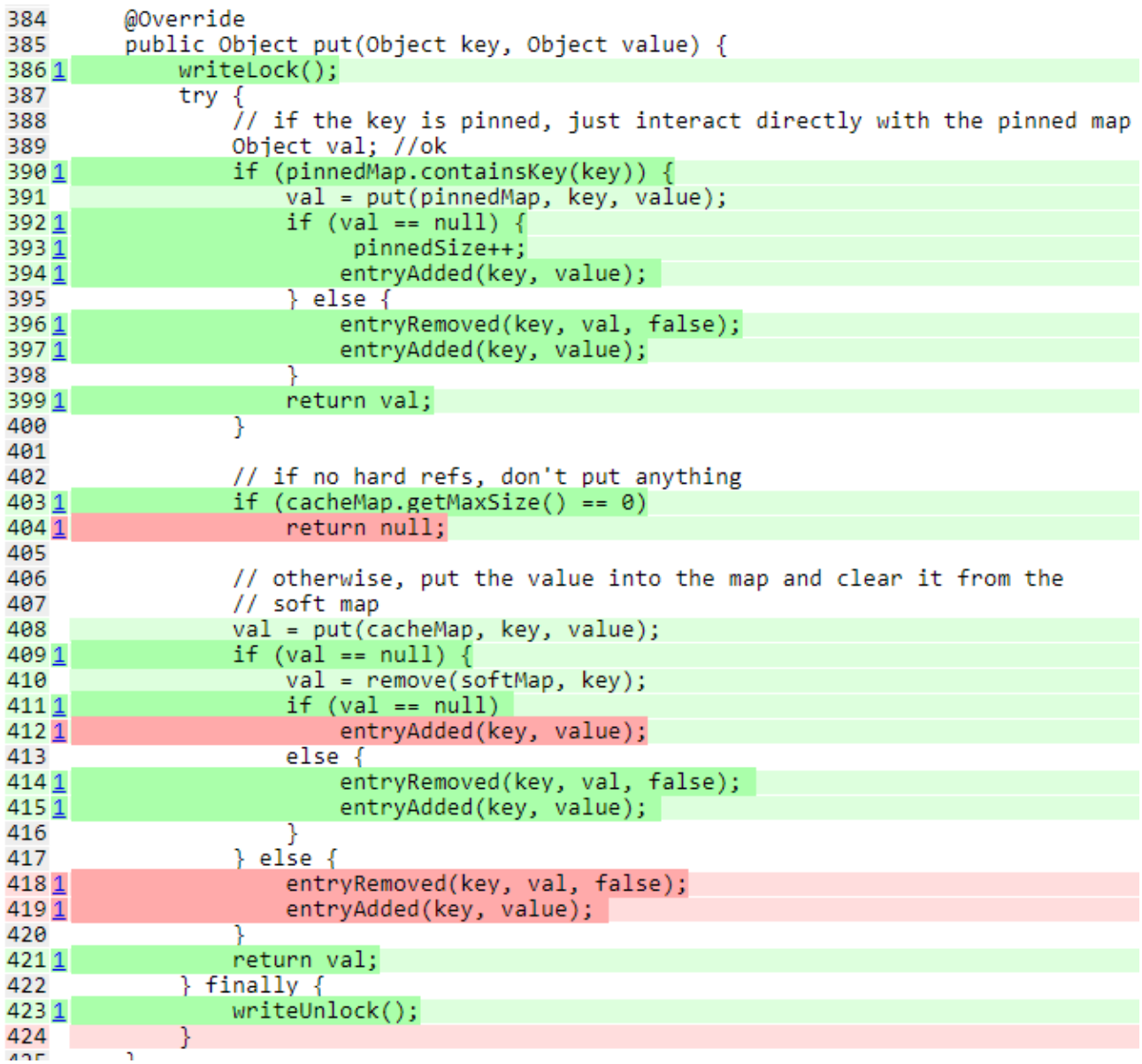


Figura 50 Mutation testing del metodo put con la test suite minimale.

```
384.     @Override
385.     public Object put(Object key, Object value) {
386.         writeLock();
387.         try {
388.             // if the key is pinned, just interact directly with the pinned map
389.             Object val; //ok
390.             if (pinnedMap.containsKey(key)) {
391.                 val = put(pinnedMap, key, value);
392.                 if (val == null) {
393.                     _pinnedSize++;
394.                     entryAdded(key, value);
395.                 } else {
396.                     entryRemoved(key, val, false);
397.                     entryAdded(key, value);
398.                 }
399.                 return val;
400.             }
401. 
402.             // if no hard refs, don't put anything
403.             if (cacheMap.getMaxSize() == 0)
404.                 return null;
405. 
406.             // otherwise, put the value into the map and clear it from the
407.             // soft map
408.             val = put(cacheMap, key, value);
409.             if (val == null) {
410.                 val = remove(softMap, key);
411.                 if (val == null)
412.                     entryAdded(key, value);
413.                 else {
414.                     entryRemoved(key, val, false);
415.                     entryAdded(key, value);
416.                 }
417.             } else {
418.                 entryRemoved(key, val, false);
419.                 entryAdded(key, value);
420.             }
421.             return val;
422.         } finally {
423.             writeUnlock();
424.         }
425.     }
```

Figura 51 Statement e Branch Coverage del metodo put dopo l'ampiamento della test suite minimale.

```
384      @Override
385      public Object put(Object key, Object value) {
386 1        writeLock();
387          try {
388              // if the key is pinned, just interact directly with the pinned map
389              Object val; //ok
390 1            if (pinnedMap.containsKey(key)) {
391                  val = put(pinnedMap, key, value);
392 1                if (val == null) {
393 1                    _pinnedSize++;
394 1                    entryAdded(key, value);
395                  } else {
396 1                    entryRemoved(key, val, false);
397 1                    entryAdded(key, value);
398                  }
399 1                return val;
400              }
401
402              // if no hard refs, don't put anything
403 1            if (cacheMap.getMaxSize() == 0)
404 1                return null;
405
406              // otherwise, put the value into the map and clear it from the
407              // soft map
408              val = put(cacheMap, key, value);
409 1            if (val == null) {
410                  val = remove(softMap, key);
411 1                if (val == null)
412 1                    entryAdded(key, value);
413                  else {
414 1                    entryRemoved(key, val, false);
415 1                    entryAdded(key, value);
416                  }
417              } else {
418 1                entryRemoved(key, val, false);
419 1                entryAdded(key, value);
420              }
421 1            return val;
422          } finally {
423 1            writeUnlock();
424          }
425      }
```

Figura 52 Mutation testing del metodo put dopo l'ampiamento della test suite minimale.

```
442.     /**
443.      * If <code>key</code> is pinned into the cache, the pin is
444.      * cleared and the object is removed.
445.      */
446.     @Override
447.     public Object remove(Object key) {
448.         writeLock();
449.         try {
450.             // if the key is pinned, just interact directly with the
451.             // pinned map
452.             Object val;
453.             if (pinnedMap.containsKey(key)) {
454.                 // re-put with null value; we still want key pinned
455.                 val = put(pinnedMap, key, null);
456.                 if (val != null) {
457.                     _pinnedSize--;
458.                     entryRemoved(key, val, false);
459.                 }
460.                 return val;
461.             }
462. 
463.             val = remove(cacheMap, key);
464.             if (val == null)
465.                 val = softMap.remove(key);
466.             if (val != null)
467.                 entryRemoved(key, val, false);
468. 
469.             return val;
470.         } finally {
471.             writeUnlock();
472.         }
473.     }
```

Figura 53 Statement e Branch Coverage del metodo remove con la test suite minimale.

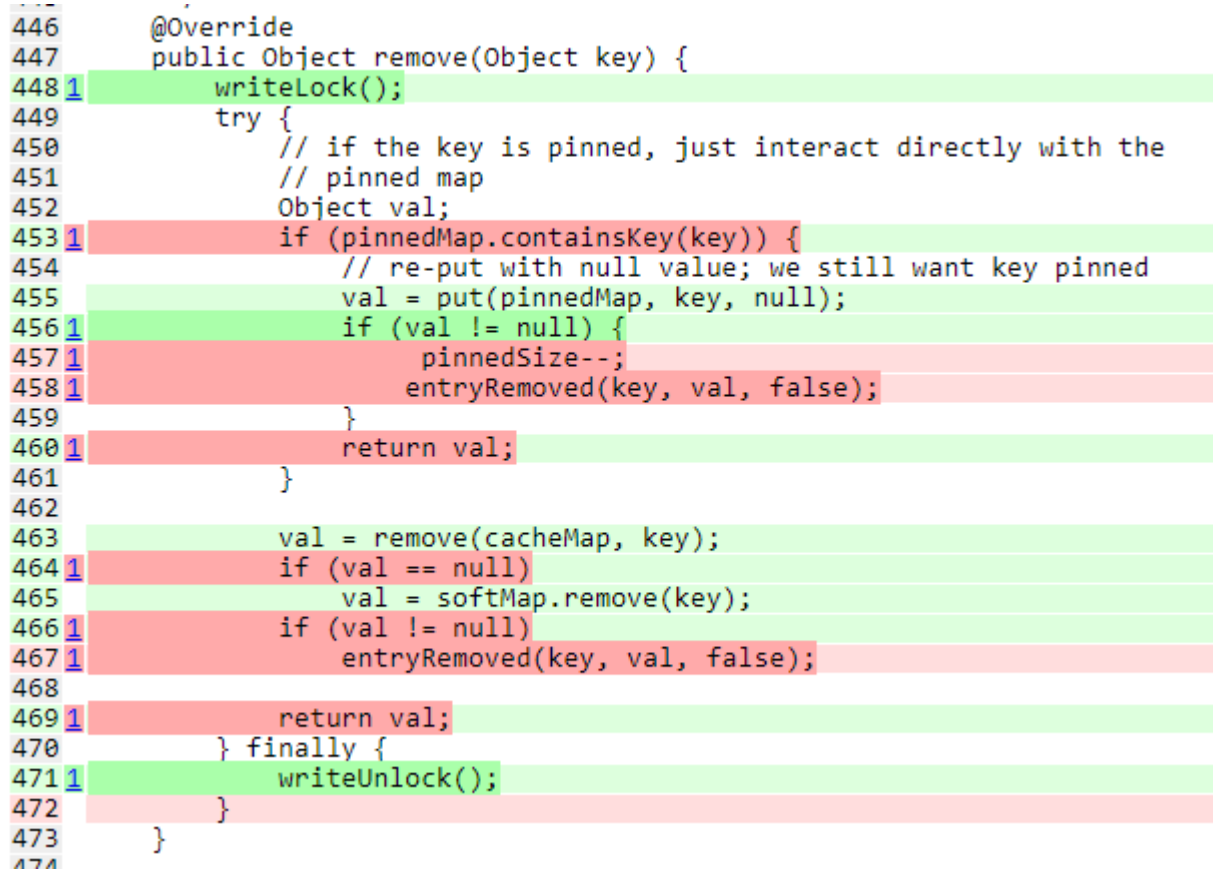


Figura 54 Mutation testing del metodo remove con la test suite minimale.

```
442.     /**
443.      * If <code>key</code> is pinned into the cache, the pin is
444.      * cleared and the object is removed.
445.      */
446.     @Override
447.     public Object remove(Object key) {
448.         writeLock();
449.         try {
450.             // if the key is pinned, just interact directly with the
451.             // pinned map
452.             Object val;
453.             if (pinnedMap.containsKey(key)) {
454.                 // re-put with null value; we still want key pinned
455.                 val = put(pinnedMap, key, null);
456.                 if (val != null) {
457.                     _pinnedSize--;
458.                     entryRemoved(key, val, false);
459.                 }
460.                 return val;
461.             }
462. 
463.             val = remove(cacheMap, key);
464.             if (val == null)
465.                 val = softMap.remove(key);
466.             if (val != null)
467.                 entryRemoved(key, val, false);
468. 
469.             return val;
470.         } finally {
471.             writeUnlock();
472.         }
473.     }
```

Figura 55 Statement e Branch Coverage del metodo remove dopo l'ampiamento della test suite minimale.

```
442       /**
443        * If <code>key</code> is pinned into the cache, the pin is
444        * cleared and the object is removed.
445        */
446       @Override
447       public Object remove(Object key) {
448 1         writeLock();
449           try {
450               // if the key is pinned, just interact directly with the
451               // pinned map
452               Object val;
453 1             if (pinnedMap.containsKey(key)) {
454                   // re-put with null value; we still want key pinned
455                   val = put(pinnedMap, key, null);
456 1                 if (val != null) {
457 1                     _pinnedSize--;
458 1                     entryRemoved(key, val, false);
459                   }
460 1                 return val;
461               }
462
463               val = remove(cacheMap, key);
464 1             if (val == null)
465                   val = softMap.remove(key);
466 1             if (val != null)
467 1                 entryRemoved(key, val, false);
468
469 1             return val;
470           } finally {
471 1             writeUnlock();
472           }
473       }
```

Figura 56 Mutation testing del metodo remove dopo l'ampiamento della test suite minimale.

```
352.     /**
353.      * Invoked when an entry is added to the cache. This may be invoked
354.      * more than once for an entry.
355.      */
356.     protected void entryAdded(Object key, Object value) {
357.     }
358.
359.     @Override
360.     public Object get(Object key) {
361.         boolean putcache = false;
362.         Object val = null;
363.         readLock();
364.         try {
365.             val = softMap.get(key);
366.             if (val == null) {
367.                 val = cacheMap.get(key);
368.                 if (val == null) {
369.                     val = pinnedMap.get(key);
370.                 }
371.             } else {
372.                 putcache = true;
373.             }
374.             return val;
375.         } finally {
376.             readUnlock();
377.             //cannot obtain a write lock while holding a read lock
378.             //doing it this way prevents a deadlock
379.             if (putcache)
380.                 put(key, val);
381.         }
382.     }
383
```

Figura 57Statement e Branch Coverage del metodo get con la test suite minimale.

```
359       @Override
360       public Object get(Object key) {
361           boolean putcache = false;
362           Object val = null;
363 1         readLock();
364           try {
365               val = softMap.get(key);
366 1             if (val == null) {
367                   val = cacheMap.get(key);
368 1                 if (val == null) {
369                       val = pinnedMap.get(key);
370                   }
371               } else {
372                   putcache = true;
373               }
374 1             return val;
375           } finally {
376 1             readUnlock();
377               //cannot obtain a write lock while holding a read lock
378               //doing it this way prevents a deadlock
379 1             if (putcache)
380                   put(key, val);
381           }
382       }
```

Figura 58 Mutation testing del metodo get con la test suite minimale.

```
59.     @Override
60.     public Object get(Object key) {
61.         boolean putcache = false;
62.         Object val = null;
63.         readLock();
64.         try {
65.             val = softMap.get(key);
66.             if (val == null) {
67.                 val = cacheMap.get(key);
68.                 if (val == null) {
69.                     val = pinnedMap.get(key);
70.                 }
71.             } else {
72.                 putcache = true;
73.             }
74.             return val;
75.         } finally {
76.             readUnlock();
77.             //cannot obtain a write lock while holding a read lock
78.             //doing it this way prevents a deadlock
79.             if (putcache)
80.                 put(key, val);
81.         }
82.     }
```

Figura 59 Statement e Branch Coverage del metodo get dopo l'ampiamento della test suite minimale.

```
359      @Override
360      public Object get(Object key) {
361          boolean putcache = false;
362          Object val = null;
363 1        readLock();
364          try {
365              val = softMap.get(key);
366 1            if (val == null) {
367                  val = cacheMap.get(key);
368 1                if (val == null) {
369                      val = pinnedMap.get(key);
370                  }
371              } else {
372                  putcache = true;
373              }
374 1            return val;
375          } finally {
376 1            readUnlock();
377              //cannot obtain a write lock while holding a read lock
378              //doing it this way prevents a deadlock
379 1            if (putcache)
380                  put(key, val);
381          }
382      }
```

Figura 60 Mutation testing del metodo get dopo l'ampiamento della test suite minimale.