

# Report ISW2 – Modulo SWTesting

Francesco Mancini - 0285816

## Introduzione

L'obiettivo di questo report è presentare lo svolgimento e i risultati di attività di testing su classi Java di progetti di Apache Software Foundation. Tali attività sono state integrate in un ciclo di Continuous Integration: i progetti coinvolti nei test, infatti, sono stati modificati affinché l'operazione di building e testing fosse eseguita automaticamente ad ogni modifica del codice sorgente. A supporto di tale processo sono state utilizzate le piattaforme Travis CI, per la build e il test dei progetti, e SonarCloud, per raccogliere e analizzare i risultati dei test. Inoltre, è stato utilizzato il framework JUnit per l'implementazione dei casi di test, JaCoCo per la generazione dei report dei risultati dei test e ba-dua per l'analisi del data-flow coverage. I progetti oggetto di questo studio sono due: Bookkeeper e OpenJPA.

## Bookkeeper

Bookkeeper è un servizio di storage scalabile, tollerante ai guasti e a bassa latenza, ottimizzato per carichi di lavoro real-time. Le classi selezionate per le attività di testing sono: *BufferedChannel* e *LedgerMetadataIndex*. La prima è stata scelta in quanto è stata modificata quasi totalmente nella release 6 [Tabella 1]: il numero di linee di codice eliminate, infatti, sono pressoché identiche al numero di linee aggiunte; tale caratteristica può significare che vi è stata una riprogettazione totale della classe, aumentando la predisposizione della stessa ad avere bug. L'altra, invece, è stata selezionata per la chiarezza delle sue funzionalità, grazie ai commenti nel codice, e per la disponibilità di documentazione per gli aspetti non descritti all'interno della classe stessa.

### BufferedChannel

Questa classe fornisce un livello di buffering davanti a un *FileChannel*, un canale che permette l'accesso e la modifica di un file. *BufferedChannel* utilizza un buffer per registrare i dati che devono essere salvati sul file, ritardando l'operazione di accesso al disco, e utilizza un ulteriore buffer per salvare i dati più recenti, in modo tale da velocizzare le operazioni di lettura. Tale classe è utilizzata per il salvataggio di entries all'interno di un log file, un file sequenziale il cui compito è quello di registrare tutte le transazioni eseguite per conto dei client collegati al servizio.

Il primo metodo analizzato è *public void write(ByteBuf src) throws IOException*. Questo metodo esegue un'operazione di scrittura di una sequenza di bytes, passata come parametro (*src*), sul *FileChannel* associato. L'operazione può essere posticipata e i dati associati vengono salvati momentaneamente in un buffer di scrittura. L'analisi preliminare di tale metodo ha seguito le regole del Category Partition; in particolare, si sono definite le seguenti partizioni: *src = {null, empty, nonEmpty}*. Un'analisi più approfondita ha individuato ulteriori parametri che sono indirettamente collegati con il metodo in questione. Tali parametri sono contenuti nel costruttore della classe:

```
public BufferedChannel(ByteBufAllocator allocator, FileChannel fc, int capacity).
```

I parametri di interesse sono *fc* e *capacity*: il primo parametro rappresenta il *FileChannel* su cui si vuole operare, mentre il secondo la capacità dei buffer di scrittura e di lettura. La funzione *write* deve essere in grado di scrivere correttamente i bytes passati come input, rispettando l'ordine sequenziale del file, indipendentemente dal contenuto già presente e dalla dimensione del buffer. Di conseguenza, l'analisi eseguita tramite Category Partition si amplia con altre due partizioni: *fc = {null, empty, nonEmpty}* e *capacity = {< 0, = 0, > 0}*. Tra le variabili considerate può sussistere una relazione in base alla quale il comportamento del metodo cambia: tale relazione coinvolge i parametri *src* e *capacity*. In base al numero di bytes passati come input e alla capacità del buffer, infatti, il metodo può eseguire le operazioni di scrittura in modo differente. Ad esempio, se la quantità di bytes è superiore alla capacità del buffer, il metodo ne dovrà probabilmente scrivere una porzione direttamente nel *FileChannel*, mentre la rimanente la salverà nel buffer. Le partizioni, di conseguenza, si modificano nel seguente modo: *src = {null, 0 = src.size() < capacity, 0 = src.size() = capacity, 0 = src.size() > capacity, 0 < src.size() < capacity, 0 < src.size() = capacity, src.size() > 0 & src.size() > capacity}*, *fc = {null, empty, nonEmpty}* e *capacity = {< 0, = 0, > 0}*. Lo sviluppo dei casi di test si è basato su un approccio di tipo unidimensionale, ovvero, i parametri sono stati considerati in modo indipendente tra loro e i casi di test cercano di coprire tutte le classi di equivalenza considerate. Il valore *null* associato al parametro *fc* è stato escluso in quanto impediva l'istanziazione della classe *BufferedChannel*. Il comportamento atteso del metodo è stato dedotto in base ai commenti presenti nel codice e alle conoscenze pregresse riguardo alla struttura dati associata. In particolare, le scritture devono essere eseguite con successo indipendentemente dal fatto che la quantità da scrivere sia maggiore della

dimensione del buffer allocato e deve essere mantenuto l'ordinamento sequenziale di quanto scritto. Inoltre, deve essere impedita la creazione di buffer con dimensioni non positive: un buffer di dimensioni negative, infatti, è impossibile da creare, mentre un buffer di dimensioni nulle vanifica le qualità non funzionali che offre la classe, rendendola identica alla classe `FileChannel`. L'esecuzione dei casi di test, sviluppati nel suddetto modo, ha rilevato la presenza di un malfunzionamento: se si richiede la scrittura di una sequenza di bytes non nulla e il buffer ha una capacità pari a zero, l'esecuzione del codice si blocca in un ciclo infinito senza mai concludersi. Un'analisi del codice ha rivelato il difetto che è alla causa del malfunzionamento. Il metodo, prima di eseguire la scrittura, calcola il numero di bytes che possono essere scritti nel buffer; nel caso in cui tale numero sia zero, il metodo svuota il buffer, scrivendo nel `FileChannel` i bytes presenti. Dopo tale operazione, il metodo ricalcola il valore aggiornato ed esegue la scrittura dei bytes richiesti. Nel caso in cui il buffer abbia capacità pari a zero, ad ogni iterazione il metodo otterrà zero come numero di bytes che possono essere scritti nel buffer e, di conseguenza, tenterà di svuotarlo inutilmente, determinando un ciclo infinito. Vi possono essere due errori che stanno alla base di questo difetto: il primo è che non viene impedito l'istanziazione di un `BufferedChannel` con buffer di dimensioni nulle, mentre il secondo è assumere che l'impossibilità di scrivere nel buffer è causata solamente dalla presenza di dati al suo interno da salvare nel `FileChannel`. I risultati dei test hanno riportato un valore di statement coverage pari al 74% e di branch coverage pari al 60% [Figura 1]. Per migliorare i valori di coverage si è analizzato il codice del metodo generato dal framework JaCoCo [Figura 2]; tale codice specifica quali branch e statement non sono stati coperti dai casi di test. Dall'analisi dei risultati si è individuata una funzionalità offerta che non è stata contemplata nei casi di test. La classe `BufferedChannel` offre la possibilità di eseguire regolarmente il salvataggio dei dati nel `FileChannel` al superamento di una soglia `unpersistedBytesBound`. Tale variabile, generalmente minore della dimensione del buffer di scrittura, viene impostata all'atto dell'istanziazione della classe con il seguente costruttore:

```
public BufferedChannel(ByteBufAllocator allocator, FileChannel fc, int writeCapacity, int readCapacity,
    long unpersistedBytesBound) throws IOException
```

I casi di test sono stati estesi in modo tale da poter includere la selezione del valore di `unpersistedBytesBound`. I valori sono stati scelti in relazione alla dimensione dei dati da scrivere; in particolare, si è incluso almeno un caso di test in cui si scrivesse una quantità di bytes minore della soglia `unpersistedBytesBound` e un altro in cui si scrivesse una quantità maggiore. Inoltre, è stato incluso un caso di test in cui tale soglia è impostata a zero; in tal modo la funzionalità si disattiva e si mantiene la copertura dei branch ottenuta nella fase precedente. La modifica dei casi di test ha permesso di raggiungere una copertura massima [Figura 3 e Figura 4]. Il risultato del mutation test sul metodo `write(ByteBuf src)` [Figura 5] rivela la presenza di comportamenti del metodo che non vengono coperti dai casi di test. In particolare, si nota che non viene controllata la correttezza del numero che indica la posizione dell'ultima scrittura; se tale valore non venisse aggiornato correttamente, vi potrebbero essere malfunzionamenti durante le operazioni di lettura. Inoltre, i casi di test non sono in grado di rilevare modifiche dell'operazione di calcolo del numero di bytes da copiare nel buffer. Per rimediare a tali mancanze, si è aggiunto un controllo al valore della posizione dell'ultima scrittura e un caso di test che fosse sensibile alla mutazione del calcolo del numero di bytes da salvare: tale caso è caratterizzato da una un numero di bytes da scrivere maggiore della capacità del buffer. La modifica dei casi di test ha aumentato il valore di mutation coverage associato al metodo [Figura 6].

Il secondo metodo analizzato è:

```
public synchronized int read(ByteBuf dest, long pos, int length) throws IOException
```

Tale metodo permette di leggere `length` bytes dal `FileChannel` a partire dalla posizione `pos`; i bytes letti vengono salvati in `dest` e il numero di tali bytes viene restituito come parametro di ritorno. Il metodo, prima di leggere il file, ricerca i bytes richiesti in un apposito buffer in cui vengono salvate le letture recenti; in tal modo si evitano ritardi dovuti all'accesso al disco. L'analisi delle classi di equivalenza dei parametri in input ha determinato le seguenti partizioni: `dest = {null, notNull}`, `pos = {< 0, = 0, > 0}`, `length = {< 0, = 0, > 0}`. Similmente al metodo precedente, sono stati coinvolti nell'analisi i parametri di input del costruttore: il `FileChannel fc` attraverso il quale la classe interagisce con il file e la capacità del buffer `capacity`. L'analisi delle classi di equivalenza di questi parametri rimane invariata: `fc = {null, empty, nonEmpty}` e `capacity = {< 0, = 0, > 0}`. Si è identificata una possibile relazione che sussiste tra le seguenti variabili: `pos`, `length` e `fc`. Il metodo, infatti deve essere in grado di gestire i casi in cui si richiedono più dati di quanti il file ne possieda oppure che si richieda la lettura di bytes da una posizione non ancora raggiunta. Le relazioni si aggiornano nel seguente modo: `pos = {< fc.size(), = fc.size(), > fc.size()}` e `length = {< fc.size() - pos, = fc.size() - pos, > fc.size() - pos}`. La generazione dei casi di test ha seguito l'approccio unidimensionale. Si considera il comportamento del metodo corretto quando i bytes restituiti corrispondono a quanto richiesto in input (bytes che si trovano nel range `[pos, pos + length]`) e la lunghezza restituita corrisponde al numero di bytes effettivamente letti. Analizzando il codice si è compreso che nel caso in cui il quantitativo di bytes richiesto è maggiore di quanto disponibile, il metodo solleverà un'eccezione. L'esecuzione dei test ha rilevato la presenza

di un malfunzionamento: in un caso di test, il metodo ha letto più bytes di quanti richiesti nella variabile *length*. Dallo studio del codice si nota che il numero di bytes da copiare viene determinato solamente in base al numero di bytes disponibili nel buffer di lettura/scrittura e alla dimensione della struttura dove salvare i bytes (*dest*), senza considerare la variabile *length* passata in input. Nel caso in cui la dimensione di *dest* e il numero di bytes leggibili sono maggiori del valore *length*, allora il metodo continuerà a leggere bytes fintanto che non si saturi *dest* oppure non vi siano ulteriori bytes da leggere. I risultati dei test hanno riportato un valore di statement coverage pari al 60% e di branch coverage pari al 50% [Figura 7]. L'analisi del codice associato al metodo ha rilevato la presenza di percorsi di esecuzione non raggiungibili direttamente con il solo utilizzo della tecnica di Category Partition. In particolare, il metodo controlla che i bytes che deve leggere siano all'interno del buffer di scrittura e, in caso, esegue la lettura da quest'ultimo [Figura 8]. Questa situazione si verifica soltanto quando l'operazione di lettura è preceduta da una di scrittura. I test, quindi, sono stati modificati in modo tale da eseguire delle operazioni di scrittura prima di quelle di lettura. Si può decidere di eseguire o meno tale operazione per ciascun caso di test, in modo tale da non modificare i risultati ottenuti precedentemente. La modifica dei casi di test ha permesso di innalzare i valori di statement e branch coverage rispettivamente al 90% e al 60% [Figura 9 e Figura 10]. Alcuni branch non sono percorribili perché le loro condizioni si basano sul fatto che il buffer di scrittura sia nullo; questo è impossibile in quanto il costruttore alloca sempre un buffer non nullo. Anche se ciò fosse possibile, si avrebbero dei comportamenti inaspettati nel metodo *write()*, in quanto non vi sono controlli sull'esistenza o meno di tale buffer. Applicando la tecnica di mutation testing, si nota che i casi di test possono non coprire porzioni di codice precedentemente coperte [Figura 11]; in particolare, i casi di test non sono in grado di eseguire le righe 247, 248 e 249 a causa della mutazione del controllo del numero di bytes copiabili (riga 243), che rileva se si è oltrepassata la fine del file. Questo può essere risolto aggiungendo un caso di test in cui si richiede la lettura di più bytes di quanti ve ne sono, i bytes da leggere sono stati scritti nel buffer di scrittura e il numero di bytes scritti è minore della capacità del buffer di scrittura. Aggiungendo tale metodo si nota un aumento dello statement coverage e del mutation coverage [Figura 12].

## LedgerMetadataIndex

Questa classe permette di costruire e mantenere un indice dei metadati del Ledger, diminuendo i tempi di ricerca di tali dati. La classe utilizza un database di tipo chiave-valore, per esempio RocksDB, per gestire l'indice. Inoltre, offre dei metodi per creare, cercare, modificare ed eliminare i dati dell'indice. L'indicizzazione dei metadati si basa sull'identificativo del Ledger associato. I casi di test sono stati implementati utilizzando il framework Mockito: tale framework è stato utilizzato per emulare il comportamento del database durante l'interazione con la classe da testare.

Il primo metodo analizzato è:

```
public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException
```

Tale metodo permette di aggiungere una password *masterKey* al ledger con identificativo pari a *ledgerId*. L'analisi dei parametri tramite Category Partition ha individuato le seguenti classi di equivalenza: *ledgerId* = {< 0, = 0, > 0} e *masterKey* = {null, empty, nonEmpty}. Inoltre, si nota che il metodo potrebbe avere dei comportamenti differenti se i metadati identificati da *ledgerId* non fossero nell'indice. Di conseguenza, si aggiunge nei casi di test un altro parametro che identifica la presenza o meno dei dati di *ledgerId* nell'indice: *exist* = {true, false}. Il comportamento del metodo è corretto se imposta correttamente il valore *masterKey* al Ledger con id *ledgerId*. Analizzando il codice, si è notato che la scrittura della *masterKey* in un Ledger che non esiste dovrebbe concludersi comunque senza alcun errore: il metodo si occupa lui stesso dell'istanziazione dei metadati del Ledger richiesto. I casi di test sono stati costruiti utilizzando un approccio unidimensionale. I risultati dei test hanno riportato un valore di statement coverage pari al 81% e di branch coverage pari al 50% [Figura 13Figura 7]. Dall'analisi del codice [Figura 14], si nota che i casi di test non contemplano il caso in cui la *masterKey* sia già stata assegnata al Ledger. In questo caso la modifica avviene con successo solo se la chiave che è stata impostata ha solo bytes nulli, altrimenti solleva un'eccezione. In base a quanto detto, i casi di test sono stati ampliati, offrendo la possibilità di definire una *masterKey* da settare nel Ledger prima di eseguire i test. Modificando i casi di test, sono aumentati i valori di statement e branch coverage rispettivamente al 100% e al 85% [Figura 15]. I branch non coperti corrispondono a sezioni del metodo dedicate ai messaggi di log [Figura 16], pertanto, tali branch non sono direttamente collegati alle funzionalità offerte dal metodo. Si è eseguito un Mutation testing sui casi di test sviluppati finora per il metodo. In base ai risultati ottenuti [Figura 17], si nota che i casi di test sono stati in grado di identificare quasi tutte le mutazioni; vi è solo un'unica eccezione alla riga 197. Questa mutazione non è stata individuata poiché i test non verificano che il numero di Ledger all'interno dell'indice sia corretto.

Un altro metodo testato è:

```
public LedgerData get(long ledgerId) throws IOException
```

Tale metodo permette di ottenere dall'indice i dati del Ledger identificato da *ledgerId*. L'analisi del parametro tramite Category Partition restituisce la seguente classe di equivalenza:  $ledgerId = \{<0,=0,>0\}$ . Il metodo potrebbe avere dei comportamenti alternativi in base all'effettiva presenza o meno del Ledger nell'indice; si è tenuto conto di tale aspetto nella creazione dei casi di test. Inoltre, si è considerato il caso in cui il Ledger fosse presente ma eliminato prima di richiederlo. Il metodo deve restituire il Ledger associato a *ledgerId* se vi è, altrimenti deve sollevare un'eccezione. I casi di test permettono di raggiungere una copertura totale degli statement, mentre una copertura del 75% dei branch [Figura 18]. Vi è un solo branch non coperto, dovuto al fatto che i casi di test non considerano gli aspetti di configurazione dei messaggi di log [Figura 19]. I casi di test implementanti risultano accurati anche in presenza di mutazioni, riuscendo a individuare tutte quelle generate dal framework Pit [Figura 20].

Un altro metodo testato è:

```
public boolean setFenced(long ledgerId) throws IOException
```

Tale metodo permette di impostare il Ledger avente id *ledgerId* allo stato "fenced". Il fencing è una tecnica utilizzata per garantire le proprietà di persistenza offerte dal servizio. Lo stato "fenced" impedisce a due clients di modificare contemporaneamente un Ledger, aggiungendovi dati oppure chiudendolo. In questo stato è possibile eseguire un recupero dei dati del Ledger senza che si perda la consistenza a causa di altre operazioni che si verificano in concorrenza. L'analisi preliminare del metodo tramite Category Partition ha rilevato il seguente partizionamento del dominio dei casi di test:  $ledgerId = \{<0,=0,>0\}$ . Inoltre, il metodo potrebbe avere comportamenti differenti in caso in cui il Ledger vi sia o meno. Il comportamento atteso del metodo è stato stimato inferendo sul codice associato: se il Ledger non vi è, il metodo solleva un'eccezione, altrimenti restituisce un booleano, che rappresenta il completamento o meno dell'operazione. Lo sviluppo dei casi di test ha seguito un approccio di tipo unidimensionale, considerando le suddette classi di equivalenza. I risultati dei test hanno riportato un valore di statement coverage pari al 75%, mentre un valore di branch coverage del 37% [Figura 21]. Si sono analizzati gli statement e i branch non coperti per poter migliorare i valori di coverage [Figura 22]. In particolare, si nota che il metodo ha un comportamento differente se il Ledger si trova già nello stato "fenced": tale metodo, infatti, non esegue nuovamente l'operazione annessa al cambio di stato e restituisce al chiamante il valore falso per indicargli tale situazione. Inoltre, il metodo ha un comportamento differente in base alla concorrenza delle operazioni eseguite: il metodo, per poter impostare lo stato del Ledger a "fenced", ricerca i metadati del Ledger in questione, ne crea una nuova copia in cui imposta lo stato a "fenced" e li inserisce nella lista dei Ledger del Bookie associato. In caso di operazioni in concorrenza potrebbe accadere che i metadati del Ledger in questione vengano eliminati prima del completamento dell'operazione; il metodo si può accorgere di tale eliminazione quando inserisce i metadati del Ledger aggiornati nella lista dei Ledger del Bookie: la funzione di inserimento (riga 157) restituisce *null* in caso in cui il dato da inserire non era presente. Quando accade questo, il metodo si occupa di inserire nuovamente i metadati eliminati senza interrompere l'esecuzione. Per far verificare questa condizione durante l'esecuzione dei test si è utilizzato il framework Mockito. Grazie a tale framework si è potuto modificare il comportamento delle funzione *LedgerMetadataIndex.get()*, in modo tale da emulare la presenza di concorrenza: quando la funzione *LedgerMetadataIndex.setFenced()* richiede il Ledger da modificare, viene inizialmente eseguita la funzione originale, ma viene eliminato dall'indice il Ledger richiesto prima di restituirlo. Modificando i casi di test in base alle nuove informazioni ottenute si è raggiunta una copertura totale degli statement e una copertura del 75% dei branch [Figura 23 e Figura 24]. I casi di test sviluppati hanno una buona capacità ad individuare mutazioni del codice, ad eccezione della mutazione a riga 157: tale mutazione inverte la condizione di verifica della possibile cancellazione del Ledger a causa di operazioni concorrenti. Ciò comporta all'aumento del contatore che identifica il numero di Ledger nell'indice, discostandolo dal valore effettivo. Tale mutazione può essere individuata controllando che il numero di Ledger sia concorde al relativo contatore, tuttavia, non può essere fatto perché non vi è possibilità di accedere a tale contatore dall'esterno.

Un ulteriore metodo testato è:

```
public void set(long ledgerId, LedgerData ledgerData) throws IOException
```

Questo metodo permette di aggiungere i metadati di un Ledger nell'indice gestito dalla classe *LedgerMetadataIndex*. È stata eseguita un'analisi preliminare dei possibili casi di test tramite la tecnica di Category Partition, con la quale sono state identificate le seguenti classi di equivalenza:  $ledgerId = \{<0,=0,>0\}$  e  $ledgerData = \{null, notNull\}$ . La classe *LedgerData* raccoglie i metadati del Ledger, tra i quali vi è la password (*masterKey*) e se il Ledger sia in stato di fencig (*fenced*); combinazioni di questi dati sono stati utilizzati per creare i *LedgerData* utilizzati nei test. Inoltre, si nota che il metodo potrebbe comportarsi in modo differente se vi è già presente un *LedgerData* associato a *ledgerId* all'interno dell'indice. Le informazioni

raccolte sono state utilizzate per generare i casi di test, utilizzando l'approccio unidimensionale. I risultati dei test mostrano che i casi utilizzati sono sufficienti per avere una copertura elevata: 100% di statement coverage e 75% di branch coverage. L'unico branch non coperto è relativo alla gestione dei messaggi di log, funzionalità non correlata con quella primaria del metodo. L'analisi dei casi di test tramite Mutation Testing ha mostrato che questi test non sono in grado di rilevare l'unica mutazione presente. Come nel metodo precedente, non si riesce ad individuare la mutazione in quanto bisognerebbe controllare il valore della variabile che rappresenta il numero di Ledger presenti nell'indice, inaccessibile dall'esterno della classe.

L'ultimo metodo testato è:

```
public void delete(long ledgerId) throws IOException
```

Tale metodo permette di eliminare i metadati del Ledger associato a *ledgerId*. L'analisi iniziale tramite Category Partition determina una solo possibile partizione:  $ledgerId = \{<0,=0,>0\}$ . Tale partizione può essere ampliata, considerando che il metodo può avere comportamenti differenti se i metadati del Ledger non vi sono prima ancora dell'eliminazione. Le informazioni raccolte sono state utilizzate per costruire l'insieme dei casi di test, utilizzando un approccio unidimensionale. I risultati dei test hanno riportato una statement coverage massima, mentre una branch coverage del 50% [Figura 29]. L'analisi dei branch non coperti ha rilevato comportamenti aggiuntivi del metodo: in particolare, se il Ledger è stato modificato prima dell'eliminazione, il metodo elimina pure i dati relativi alla modifica, che possono essere salvati in un buffer che ritarda le operazioni su disco [Figura 30]. I casi di test sono stati riorganizzati in modo tale da permettere di modificare la classe prima che questa venga eliminata. I nuovi casi di test modificati, hanno aumentato il branch coverage sino al 75% [Figura 31 e Figura 33]. Per quanto riguarda i mutation test, i casi di test sono in grado di rilevare efficientemente le mutazioni ad eccezione di una [Figura 32]. Tale mutazione non può essere rilevata in quanto bisognerebbe eseguire dei controlli sul contatore del numero di Ledger nell'indice, inaccessibile dall'esterno.

## OpenJPA

OpenJPA è l'implementazione di Apache della specifica Java Persistence API (JPA) per la persistenza trasparente di oggetti Java. La maggior parte dei programmi complessi utilizza dati persistenti come, ad esempio, le applicazioni web per tracciare i movimenti e gli ordini degli utenti. Tali dati devono essere salvati in supporti di memorizzazione per persistere anche dopo l'esecuzione del programma. JPA fornisce delle specifiche per un meccanismo di persistenza delle informazioni: tale meccanismo permette di salvare i dati delle classi all'interno di un database relazionale in modo trasparente allo sviluppatore. Grazie a queste API lo sviluppatore non è più direttamente coinvolto nella gestione del salvataggio dei dati, diminuendo, quindi, lo sforzo necessario per lo sviluppo delle applicazioni. Le classi che sono state selezionate per le attività di testing sono *CacheMap* e *ProxyManagerImpl*. Il motivo della scelta di queste classi risiede nell'evoluzione della loro struttura durante le varie versioni del progetto. In particolare, si nota che dalla versione 4 alla versione 10 le classi sono state pesantemente modificate senza che vi siano state aggiunte nuove funzionalità (tante linee di codice aggiunte quante eliminate) [Tabella 2 e Tabella 3]. Queste classi, probabilmente, sono state coinvolte in processi di ristrutturazione, che hanno portato alla modifica del loro comportamento interno senza però variare le funzionalità offerte. Tali ingenti modifiche possono aver introdotto dei difetti all'interno delle classi coinvolte. Un'altra motivazione che ha portato alla scelta di queste classi è la presenza di una documentazione completa sulle funzionalità offerte e sui comportamenti attesi.

## CacheMap

La classe *CacheMap* permette di implementare un meccanismo di caching delle query eseguite per richiedere i dati. Quando viene sottoposta una query, il sistema esegue un'operazione di "parsing" di tale query per individuare le operazioni da svolgere internamente per soddisfare la richiesta. I risultati delle analisi delle query possono essere salvati all'interno della classe *CacheMap* in modo tale che, alle richieste successive, il sistema non debba più eseguire le stesse analisi svolte precedentemente. Tale classe, inoltre, permette di fissare un valore massimo di query che possono essere salvate, in modo tale da occupare una porzione fissa anche in presenza di applicazioni che eseguono un numero elevato di query differenti. La classe offre pure la possibilità di gestire la rimozione delle query in caso di saturazione, utilizzando una politica di tipo Last-Recently-Used (LRU): la classe rimuove la query che non è stata utilizzata da più tempo.

Il primo metodo analizzato è:

```
public Object put(Object key, Object value)
```

Tale metodo permette di aggiungere un oggetto *value* identificato da *key* all'interno della *CacheMap*. L'analisi preliminare del metodo si è basata sulla creazione di partizioni del dominio dei casi di test basandosi sui parametri della classe. Le classi di equivalenza individuate sono le seguenti: *key* = {null, notNull} e *value* = {null, notNull}. Si nota che potrebbero esservi comportamenti differenti nel caso in cui vi sia già presente un oggetto associato a *key* all'interno della *CacheMap*. Il metodo si comporta in modo corretto se inserisce effettivamente l'oggetto *value* all'interno della *CacheMap* e se restituisce l'oggetto precedentemente associato a *key*. Tali comportamenti sono stati individuati tramite la lettura dei commenti all'interno del codice. L'analisi dei risultati dei test ha riportato un valore di Statement coverage pari al 50%, di Branch coverage pari al 50% e di Data-Flow coverage pari al  $\frac{27}{27+32} * 100 \approx 46\%$  [Figura 34 e Figura 35]. Dallo studio dei risultati dei report relativi al Control-flow coverage e al Data-flow coverage [Figura 35 e Figura 36], si è individuata una funzionalità aggiuntiva offerta dalla classe: *CacheMap* mantiene pure una mappa di query bloccate. Tali query vengono fissate tramite l'utilizzo di un'apposita funzione e non possono essere eliminate implicitamente in caso di saturazione dello spazio utilizzato. Nel caso in cui si aggiunge una query la cui chiave è stata segnata come bloccata, il metodo gestisce l'inserimento in modo differente: la query viene inserita in una mappa separata da quella principale e tale query non concorre a determinare lo spazio utilizzato dalla *CacheMap*. Inoltre, il metodo si occupa di eliminare tale query dalla mappa principale, in modo tale da non avere più repliche ed ottimizzare lo spazio. Un'altra funzionalità offerta dalla classe è mantenere una mappa temporanea delle query che sono state rimosse implicitamente dalla mappa principale. Tale mappa permette di ottenere velocemente le query che sono state eliminate recentemente perché non utilizzate da molto tempo. In base a quanto evidenziato, i casi di test sono stati estesi in modo tale da poter bloccare o meno la query prima di eseguire il test e di configurare la capacità della *CacheMap* e il numero dei dati da inserire in modo tale da far verificare la condizione di saturazione. Grazie alla modifica dei casi di test è stato possibile raggiungere una copertura totale sia del Control-Flow coverage che del Data-Flow coverage [Figura 37 e Figura 38]. Eseguendo i casi di test con l'aggiunta delle mutazioni nel codice si è scoperto che tali test non sono in grado di individuare tutte le mutazioni presenti [Figura 39]. Per poter migliorare il grado di copertura bisognerebbe analizzare i valori dei contatori che rappresentano il numero di query salvate per ciascuna mappa (*pinnedMap*, *cachedMap* e *softMap*), però alcuni di essi non possono essere acceduti (*\_pinnedSize*). Inoltre, bisognerebbe eseguire dei controlli aggiuntivi sul corretto utilizzo dei lock per accedere alle risorse: ogni metodo deve acquisire il lock prima di accedere alla risorsa e lo deve rilasciare quando ha concluso la sua operazione, permettendo agli altri processi di accedere alla risorsa. Sono stati estesi i casi di test per controllare la correttezza del procedimento di acquisizione della risorsa. Si è utilizzato il framework Mockito per poter accedere ai metodi dedicati alla gestione del lock all'interno della classe: in particolare, si esegue l'operazione di "spy" dell'istanza testata, prima di invocare il metodo da eseguire. Alla conclusione del metodo si verifica che siano state invocate correttamente le funzioni di acquisizione e rilascio del lock, rispettivamente *writeLock()* e *writeUnlock()*. Grazie a tale intervento si è potuto aumentare il mutation coverage dei casi di test [Figura 40].

Un altro metodo analizzato è:

```
public Object remove(Object key)
```

Tale metodo permette di eliminare esplicitamente query contenute all'interno della *CacheMap*. L'analisi preliminare della classe tramite Category Partition ha definito il seguente partizionamento del dominio dei casi di test: *key* = {null, notNull}. Un'analisi approfondita ha individuato un possibile comportamento differente del metodo nel caso in cui la query non fosse già presente all'atto dell'eliminazione. Si sono definiti i casi di test dalle informazioni ricavate utilizzando l'approccio multidimensionale. Si considera il comportamento della funzione corretto quando la query non viene restituita dopo la sua eliminazione. Il risultato dei casi di test ha riportato una Statement coverage del 57%, una Branch coverage del 62% e una Data-Flow coverage del 51% [Figura 41 e Figura 42]. Analizzando i branch non coperti e le coppie (definizione variabile, utilizzo variabile) non coperte [Figura 42 e Figura 43], si è individuato un comportamento differente nel caso in cui la query da eliminare sia stata bloccata: il metodo, infatti, elimina effettivamente la query ma mantiene le chiavi associate in modo tale da ricordarsi dello stato bloccato della query per gli inserimenti futuri. I casi di test sono stati modificati in modo tale da poter impostare lo stato della query prima di eseguire l'eliminazione. Le modifiche apportate ai casi di test hanno permesso di avere una copertura totale degli Statement e dei Branch e di coprire 36 coppie (definizione variabile, utilizzo variabile) su 37, ottenendo un Data-Flow coverage del 97% [Figura 43 e Figura 44]. Sono state eseguite delle mutazioni nel codice tramite il framework Pit per analizzare la capacità dei casi di test finora sviluppati nell'individuare tali mutazioni. I casi di test sono riusciti ad individuare alcune mutazioni anche se non tutte. Alcune di queste mutazioni possono essere individuate analizzando il contatore associato al numero di query salvate sulla mappa delle query bloccate (*\_pinnedSize*). Come nel caso precedente si è notato che un controllo dell'acquisizione del lock permetterebbe un aumento

del Mutation coverage. Si sono modificati i casi di test allo stesso modo del metodo precedentemente, permettendo di aumentare il mutation coverage [Figura 45].

L'ultimo metodo testato è:

```
public boolean pin(Object key)
```

Tale metodo permette di bloccare la query associata a *key*, in modo tale da impedire una sua eventuale eliminazione in caso di saturazione dello spazio di *CacheMap*. Da un'analisi preliminare si è definito il seguente partizionamento del dominio dei casi di test: *key* = {*null*, *notNull*}. Inoltre, si è considerata la possibilità che il metodo possa avere dei comportamenti differenti in caso in cui non vi sia ancora la query. La creazione dei casi di test ha seguito un approccio multidimensionale. I casi di test hanno riportato una copertura totale sia per il Control-Flow coverage che per il Data-Flow coverage [Figura 46 e Figura 47], pertanto non è stato necessario modificarli. Per quanto riguarda il mutation testing, i casi di test non sono riusciti ad individuare tutte le mutazioni generate [Figura 48]. In particolare, i casi di test non verificano che il metodo gestisca correttamente l'accesso alle risorse tramite lock, pertanto, tali test sono stati modificati per eseguire questo controllo. L'implementazione di tale controllo è stata fatta allo stesso modo dei metodi precedenti, ovvero, è stato utilizzato il framework Mockito per controllare che i metodi di acquisizione e rilascio del lock fossero effettivamente invocati. La modifica dei casi di test ha permesso di aumentare il Mutation coverage [Figura 49].

### ProxyManagerImpl

Questa classe permette di gestire la copia e il proxy di oggetti. Normalmente è utilizzata per quelle classi che compongono lo stato di classi su cui deve essere eseguita la persistenza. La funzionalità di copia viene utilizzata per salvare lo stato di queste classi durante le transazioni, in modo tale da poterlo ripristinare in caso di rollback. La funzionalità di proxy viene utilizzata per tenere traccia delle modifiche di queste classi, che possono verificarsi senza accedere necessariamente alla classe principale su cui avviene la persistenza. Tale funzionalità crea una copia della classe e notifica la classe che la possiede ogni volta viene modificata, in modo tale da poter mantenere lo stato consistente.

Il primo metodo testato è:

```
public Object copyCustom(Object orig)
```

Tale metodo permette di eseguire la copia di una istanza che gli viene passata in input. In base a quanto riportato dalla documentazione ufficiale, l'istanza può appartenere ad una qualsiasi classe che rispetta le seguenti condizioni:

- Classi che personalizzano i tipi di dati associati alle date che hanno un costruttore pubblico senza argomenti oppure un costruttore che accetti un *long* rappresentante il tempo corrente.
- Classi generiche che possono essere replicate: classi con il costruttore che accetta un'istanza del suo stesso tipo che si occupa della copia dei dati oppure classi che hanno il costruttore pubblico senza parametri e offre i metodi *get()* e *set()* per ogni attributo (classe Bean).

Si è eseguita un'analisi delle partitioni del dominio dei casi di test utilizzando la tecnica di Category Partition ottenendo le seguenti classi di equivalenza: *orig* = {*null*, *validInstance*, *notValidInstance*}. Un parametro di input è considerato valido nel caso in cui rispetti almeno una delle suddette condizioni, mentre un parametro non è considerato valido nel caso non rispetti tutte le condizioni sopra riportate. In particolare, si è considerata una classe Bean per il primo caso, mentre una classe con un costruttore con parametri e senza metodi *get()* e *set()* (classe NonBean). Il comportamento del metodo è corretto se restituisce una copia identica della classe, in caso questa sia valida, altrimenti il valore null. L'esecuzione dei casi di test ha restituito un valore di Statement Coverage del 55%, un valore di Branch coverage del 62% e un valore di Data-Flow coverage del 48% [Figura 50 e Figura 51]. Un'analisi della struttura del metodo [Figura 52] ha rivelato altre tipologie di classi per cui il metodo si comporta in modo differente; tali classi sono: *Proxy*, *Collection*, *Map*, *Date* e *Calendar*. L'aggiunta di queste tipologie di classi nei casi di test ha permesso di raggiungere dei valori di coverage quasi massimi per lo Statement coverage, il Branch coverage e Data-Flow coverage [Figura 53, Figura 54 e Figura 55]. I casi di test sono stati eseguiti, inoltre, applicando delle mutazioni nel codice tramite il framework Pit. I casi di test sviluppati sono stati in grado di rilevare tutte le mutazioni che sono state inserite [Figura 56], pertanto non vi è la necessità di modificare i test sviluppati.

Un altro metodo testato è:

```
public Proxy newCustomProxy(Object orig, boolean autoOff)
```

Tale metodo permette di creare un proxy per una qualsiasi tipologia di classe che rispetti le condizioni riportate nell'analisi del metodo precedente. Il proxy restituito da questo metodo è una copia identica della classe di partenza e l'interazione con tale istanza rimane invariata rispetto alla classe iniziale; la gestione delle notifiche di modifica dei suoi valori avviene in modo del tutto trasparente a chi vi interagisce. Il parametro *autoOff* è utilizzato per disabilitare automaticamente il tracciamento delle modifiche per le classi di tipo *Collection*: quando una *Collection* viene modificata per un numero di volte superiore al numero di elementi che contiene, OpenJPA disabilita automaticamente il tracciamento delle variazioni della *Collection* in questione. È stata fatta un'analisi preliminare dei possibili casi di test utilizzando la tecnica del Category Partition. Si sono individuate le seguenti partizioni del dominio dei casi di test: *orig* = {*null*, *validInstance*, *notValidInstance*} e *autoOff* = {*true*, *false*}. Si considera corretto il comportamento del metodo se restituisce una classe che è identica a quella in input, nel caso in cui rispetta le suddette condizioni, altrimenti se restituisce *null*. I casi di test sono stati formulati utilizzando un approccio unidimensionale. I risultati dell'esecuzione dei casi di test hanno restituito dei valori di coverage bassi: 27% di Statement coverage, 45% di Branch coverage e 31% di Data-Flow coverage. Analizzando i branch e gli statement non coperti si è individuato che il metodo ha comportamenti differenti in base alla tipologia della classe che ha il parametro *orig*. In particolare, le possibili tipologie di classi che possono essere passate in input sono: *Proxy*, *Collection*, *Map*, *SortedMap*, *Date*, *Timestamp*, *Calendar*. I casi di test sono stati ampliati in modo tale da includere queste tipologie di classi. Grazie ai nuovi casi di test, i valori di coverage hanno raggiunto quasi il valore massimo [Figura 60 e Figura 61]. Inoltre, i test sono in grado di rilevare quasi tutte le mutazioni del metodo in caso di mutation testing. Pertanto, non è necessario attuare ulteriori modifiche ai casi di test implementati.

L'ultimo metodo testato è:

```
public Object copyArray(Object orig)
```

Tale metodo permette di creare la copia di un array, passato in input tramite *orig*. L'analisi dei parametri tramite Category Partition ha restituito la seguente classe di equivalenza: *orig* = {*null*, *validInstance*, *notValidInstance*}. Una istanza valida di *orig* corrisponde ad un'istanza di un array di qualsiasi tipologia, mentre un'istanza non valida corrisponde ad una classe di una qualsiasi altra tipologia. In caso in cui il metodo viene invocato con un'istanza non valida, viene sollevata un'eccezione. I casi di test implementati sono stati sufficienti per poter avere dei valori massimi di copertura per Statement coverage, Branch coverage e Data-Flow coverage. Inoltre, tali test sono in grado di individuare tutte le mutazioni generate da Pit [Figura 65].

| BufferedChannel |     |              |       |          |             |             |            |       |          |          |  |
|-----------------|-----|--------------|-------|----------|-------------|-------------|------------|-------|----------|----------|--|
| Version         | LOC | NumRevisions | NAuth | LOCAdded | AVGLOCAdded | MAXLOCAdded | LOCTouched | Churn | AVGChurn | MAXChurn |  |
| 1               | 170 | 4            | 2     | 195      | 48          | 168         | 210        | 180   | 45       | 168      |  |
| 2               | 170 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |  |
| 3               | 170 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |  |
| 4               | 170 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |  |
| 5               | 170 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |  |
| 6               | 190 | 4            | 2     | 115      | 28          | 69          | 206        | 24    | 6        | 17       |  |

Tabella 1: Risultati delle misure della classe BufferedChannel

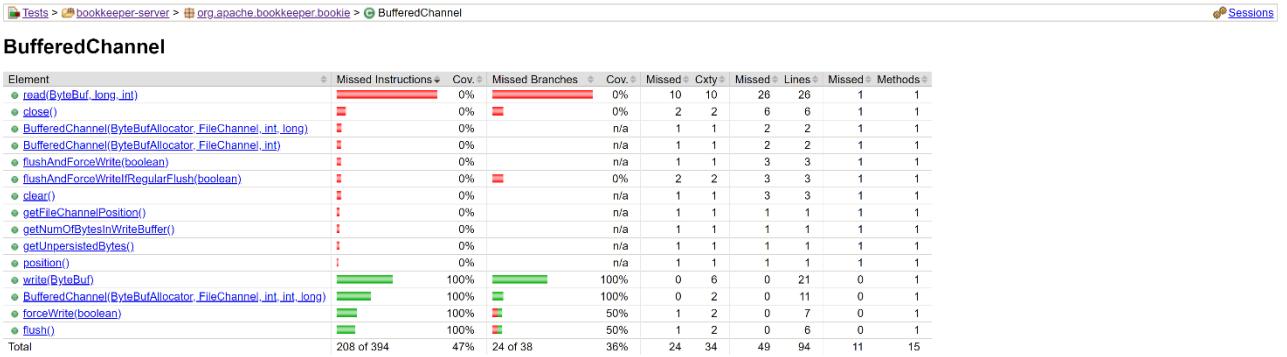


Figura 1: Statement coverage e Branch coverage per il metodo `BufferedChannel.write()`

```

101.
102.    /**
103.     * Write all the data in src to the {@link FileChannel}. Note that this function can
104.     * buffer or re-order writes based on the implementation. These writes will be flushed
105.     * to the disk only when flush() is invoked.
106.     *
107.     * @param src The source ByteBuffer which contains the data to be written.
108.     * @throws IOException if a write operation fails.
109.     */
110.    public void write(ByteBuf src) throws IOException {
111.        int copied = 0;
112.        boolean shouldForceWrite = false;
113.        synchronized (this) {
114.            int len = src.readableBytes();
115.            while (copied < len) {
116.                int bytesToCopy = Math.min(src.readableBytes() - copied, writeBuffer.writableBytes());
117.                writeBuffer.writeBytes(src, src.readerIndex() + copied, bytesToCopy);
118.                copied += bytesToCopy;
119.
120.                // if we have run out of buffer space, we should flush to the
121.                // file
122.                if (!writeBuffer.isWritable()) {
123.                    flush();
124.                }
125.                position += copied;
126.                if (doRegularFlushes) {
127.                    unpersistedBytes.addAndGet(copied);
128.                    if (unpersistedBytes.get() >= unpersistedBytesBound) {
129.                        flush();
130.                        shouldForceWrite = true;
131.                    }
132.                }
133.            }
134.            if (shouldForceWrite) {
135.                forceWrite(false);
136.            }
137.        }
138.    }
139.

```

Figura 2: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `BufferedChannel.write()`

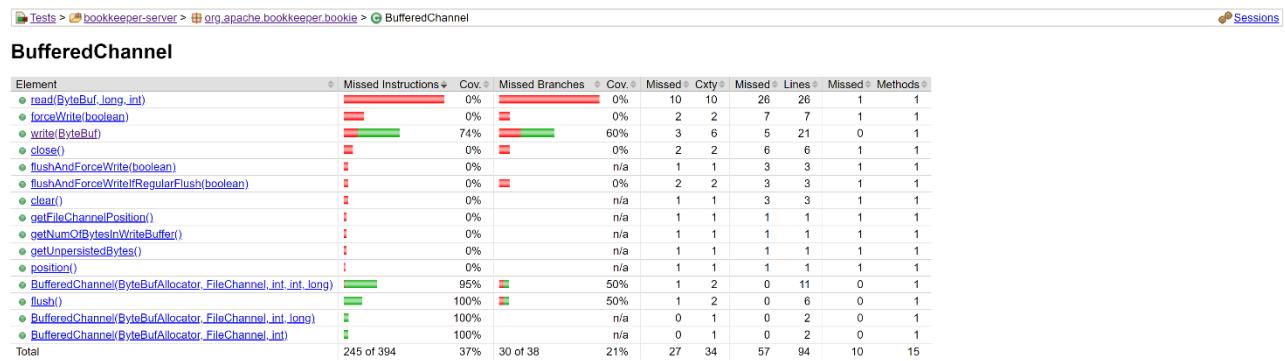


Figura 3: Statement Coverage e Branch Coverage del metodo `BufferedChannel.write()` dopo la modifica dei test

```

101.
102.    /**
103.     * Write all the data in src to the {@link FileChannel}. Note that this function can
104.     * buffer or re-order writes based on the implementation. These writes will be flushed
105.     * to the disk only when flush() is invoked.
106.     *
107.     * @param src The source ByteBuffer which contains the data to be written.
108.     * @throws IOException if a write operation fails.
109.     */
110.    public void write(ByteBuf src) throws IOException {
111.        int copied = 0;
112.        boolean shouldForceWrite = false;
113.        synchronized (this) {
114.            int len = src.readableBytes();
115.            while (copied < len) {
116.                int bytesToCopy = Math.min(src.readableBytes() - copied, writeBuffer.writableBytes());
117.                writeBuffer.writeBytes(src, src.readerIndex() + copied, bytesToCopy);
118.                copied += bytesToCopy;
119.
120.                // if we have run out of buffer space, we should flush to the
121.                // file
122.                if (!writeBuffer.isWritable()) {
123.                    flush();
124.                }
125.            }
126.            position += copied;
127.            if (doRegularFlushes) {
128.                unpersistedBytes.addAndGet(copied);
129.                if (unpersistedBytes.get() >= unpersistedBytesBound) {
130.                    flush();
131.                    shouldForceWrite = true;
132.                }
133.            }
134.        }
135.        if (shouldForceWrite) {
136.            forceWrite(false);
137.        }
138.    }
139.

```

Figura 4: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `BufferedChannel.write()` dopo la modifica dei test

```

101
102 /**
103 * Write all the data in src to the {@link FileChannel}. Note that this function can
104 * buffer or re-order writes based on the implementation. These writes will be flushed
105 * to the disk only when flush() is invoked.
106 *
107 * @param src The source ByteBuffer which contains the data to be written.
108 * @throws IOException if a write operation fails.
109 */
110 public void write(ByteBuf src) throws IOException {
111     int copied = 0;
112     boolean shouldForceWrite = false;
113     synchronized (this) {
114         int len = src.readableBytes();
115     2   while (copied < len) {
116     1     int bytesToCopy = Math.min(src.readableBytes() - copied, writeBuffer.writableBytes());
117     1     writeBuffer.writeBytes(src, src.readerIndex() + copied, bytesToCopy);
118     1     copied += bytesToCopy;
119
120         // if we have run out of buffer space, we should flush to the
121         // file
122     1         if (!writeBuffer.isWritable()) {
123     1             flush();
124         }
125     }
126     1     position += copied;
127     1     if (doRegularFlushes) {
128         unpersistedBytes.addAndGet(copied);
129     2     if (unpersistedBytes.get() >= unpersistedBytesBound) {
130     1         flush();
131         shouldForceWrite = true;
132     }
133     }
134     1     if (shouldForceWrite) {
135         forceWrite(false);
136     }
137     }
138 }
139

```

Figura 5: Risultati del Mutation Testing sul metodo `BufferedChannel.write()`

```

101
102     /**
103      * Write all the data in src to the {@link FileChannel}. Note that this function can
104      * buffer or re-order writes based on the implementation. These writes will be flushed
105      * to the disk only when flush() is invoked.
106      *
107      * @param src The source ByteBuffer which contains the data to be written.
108      * @throws IOException if a write operation fails.
109      */
110     public void write(ByteBuf src) throws IOException {
111         int copied = 0;
112         boolean shouldForceWrite = false;
113         synchronized (this) {
114             int len = src.readableBytes();
115             while (copied < len) {
116                 int bytesToCopy = Math.min(src.readableBytes() - copied, writeBuffer.writableBytes());
117                 writeBuffer.writeBytes(src, src.readerIndex() + copied, bytesToCopy);
118                 copied += bytesToCopy;
119
120                 // if we have run out of buffer space, we should flush to the
121                 // file
122                 if (!writeBuffer.isWritable()) {
123                     flush();
124                 }
125             }
126             position += copied;
127             if (doRegularFlushes) {
128                 unpersistedBytes.addAndGet(copied);
129                 if (unpersistedBytes.get() >= unpersistedBytesBound) {
130                     flush();
131                     shouldForceWrite = true;
132                 }
133             }
134         }
135         if (shouldForceWrite) {
136             forceWrite(false);
137         }
138     }
139

```

Figura 6: Risultati del Mutation Testing sul metodo `BufferedChannel.write()` dopo la modifica dei test

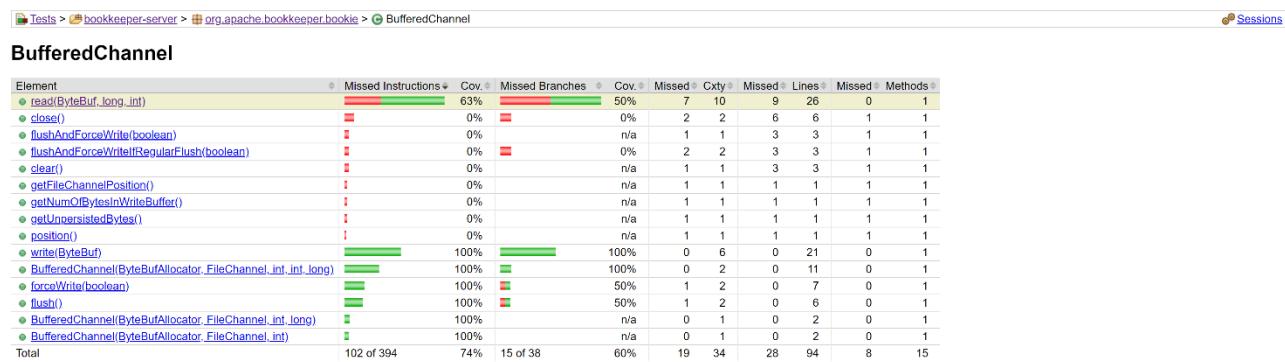


Figura 7: Statement Coverage e Branch Coverage per il metodo `BufferedChanne.read()`

```

233.
234.     @Override
235.     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
236.         long prevPos = pos;
237.         while (length > 0) {
238.             // check if it is in the write buffer
239.             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
240.                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
241.                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
242.
243.                 if (bytesToCopy == 0) {
244.                     throw new IOException("Read past EOF");
245.                 }
246.
247.                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
248.                 pos += bytesToCopy;
249.                 length -= bytesToCopy;
250.             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
251.                 // here we reach the end
252.                 break;
253.             // first check if there is anything we can grab from the readBuffer
254.             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
255.                 int positionInBuffer = (int) (pos - readBufferStartPosition);
256.                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
257.                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
258.                 pos += bytesToCopy;
259.                 length -= bytesToCopy;
260.                 // let's read it
261.             } else {
262.                 readBufferStartPosition = pos;
263.
264.                 int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
265.                                                 readBufferStartPosition);
266.                 if (readBytes <= 0) {
267.                     throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
268.                 }
269.                 readBuffer.writerIndex(readBytes);
270.             }
271.         }
272.         return (int) (pos - prevPos);
273.     }
274.

```

Figura 8: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `BufferedChannel.read()`

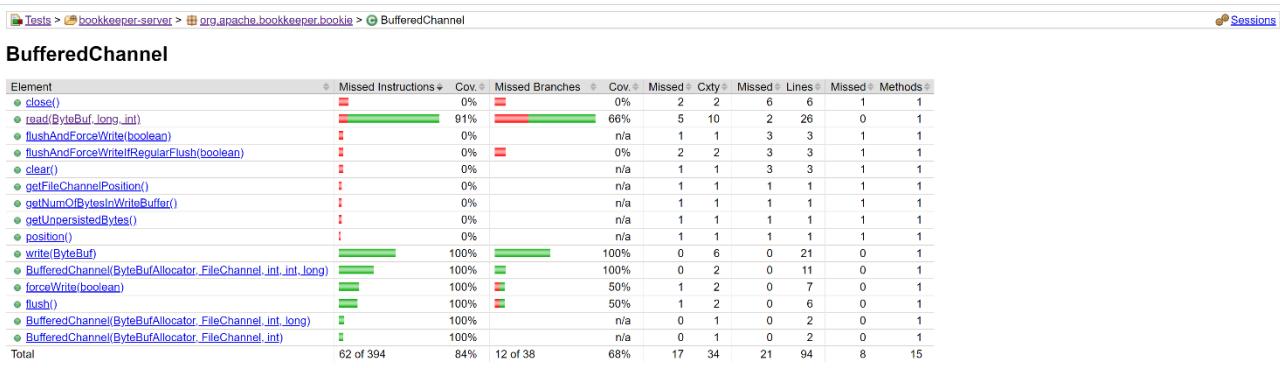


Figura 9: Statement Coverage e Branch Coverage del metodo `BufferedChannel.read()` dopo la modifica dei test

```

233.
234.     @Override
235.     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
236.         long prevPos = pos;
237.         while (length > 0) {
238.             // check if it is in the write buffer
239.             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
240.                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
241.                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
242.
243.                 if (bytesToCopy == 0) {
244.                     throw new IOException("Read past EOF");
245.                 }
246.
247.                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
248.                 pos += bytesToCopy;
249.                 length -= bytesToCopy;
250.             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
251.                 // here we reach the end
252.                 break;
253.             // first check if there is anything we can grab from the readBuffer
254.             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
255.                 int positionInBuffer = (int) (pos - readBufferStartPosition);
256.                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
257.                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
258.                 pos += bytesToCopy;
259.                 length -= bytesToCopy;
260.                 // let's read it
261.             } else {
262.                 readBufferStartPosition = pos;
263.
264.                 int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
265.                                                 readBufferStartPosition);
266.                 if (readBytes <= 0) {
267.                     throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
268.                 }
269.                 readBuffer.writerIndex(readBytes);
270.             }
271.         }
272.         return (int) (pos - prevPos);
273.     }
274.

```

Figura 10: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `BufferedChannel.read()` dopo la modifica dei test

```

233
234     @Override
235     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
236         long prevPos = pos;
237         while (length > 0) {
238             // check if it is in the write buffer
239             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
240                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
241                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
242
243                 if (bytesToCopy == 0) {
244                     throw new IOException("Read past EOF");
245                 }
246
247                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
248                 pos += bytesToCopy;
249                 length -= bytesToCopy;
250             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
251                 // here we reach the end
252                 break;
253                 // first check if there is anything we can grab from the readBuffer
254             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
255                 int positionInBuffer = (int) (pos - readBufferStartPosition);
256                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
257                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
258                 pos += bytesToCopy;
259                 length -= bytesToCopy;
260                 // let's read it
261             } else {
262                 readBufferStartPosition = pos;
263
264                 int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
265                                                 readBufferStartPosition);
266                 if (readBytes <= 0) {
267                     throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
268                 }
269                 readBuffer.writerIndex(readBytes);
270             }
271         }
272         return (int) (pos - prevPos);
273     }
274 }
```

Figura 11: Risultati del Mutation Testing sul metodo `BufferedChannel.read()`

```

233
234     @Override
235     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
236         long prevPos = pos;
237         while (length > 0) {
238             // check if it is in the write buffer
239             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
240                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
241                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
242
243                 if (bytesToCopy == 0) {
244                     throw new IOException("Read past EOF");
245                 }
246
247                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
248                 pos += bytesToCopy;
249                 length -= bytesToCopy;
250             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
251                 // here we reach the end
252                 break;
253                 // first check if there is anything we can grab from the readBuffer
254             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
255                 int positionInBuffer = (int) (pos - readBufferStartPosition);
256                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
257                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
258                 pos += bytesToCopy;
259                 length -= bytesToCopy;
260             } else {
261                 readBufferStartPosition = pos;
262
263                 int readBytes = fileChannel.read(readBuffer.internalNioBuffer(0, readCapacity),
264                                                 readBufferStartPosition);
265                 if (readBytes <= 0) {
266                     throw new IOException("Reading from filechannel returned a non-positive value. Short read.");
267                 }
268                 readBuffer.writerIndex(readBytes);
269             }
270         }
271     }
272     return (int) (pos - prevPos);
273 }
274

```

Figura 12: Risultati del Mutation Testing sul metodo `BufferedChannel.read()` dopo la modifica dei test

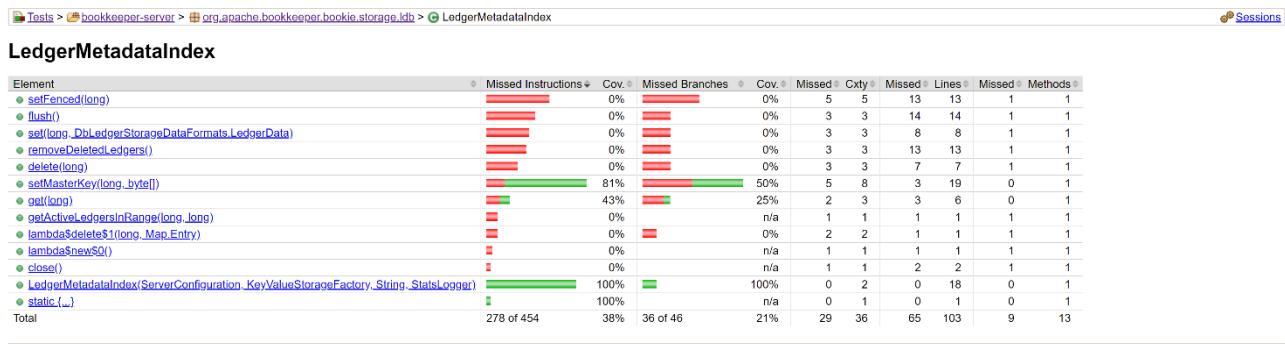


Figura 13: Statement Coverage e Branch Coverage per il metodo `LedgerMetadataIndex.setMasterKey()`

```

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

```

Figura 14: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `LedgerMetadataIndex.setMasterKey()`

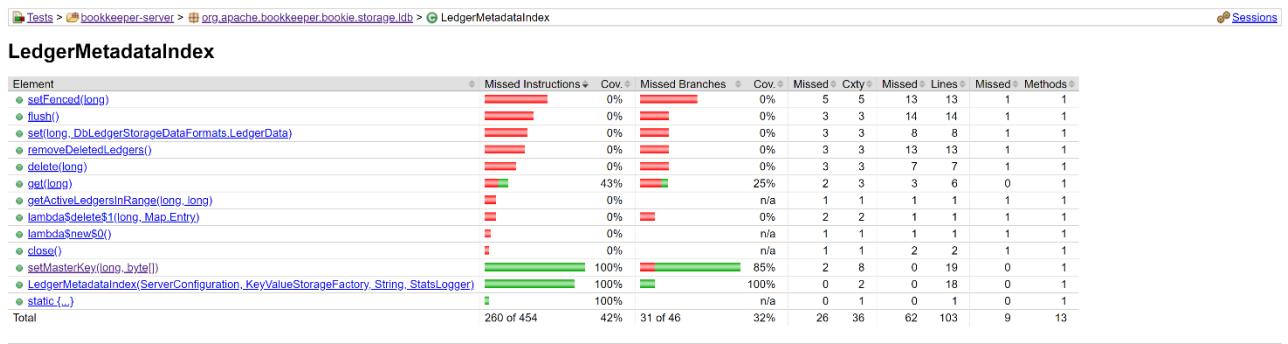


Figura 15: Statement Coverage e Branch Coverage del metodo `LedgerMetadataIndex.setMasterKey()` dopo la modifica dei test

```

173.
174.     public void setMasterKey(long ledgerId, byte[] masterKey) throws IOException {
175.         LedgerData ledgerData = ledgers.get(ledgerId);
176.         if (ledgerData == null) {
177.             // New ledger inserted
178.             ledgerData = LedgerData.newBuilder().setExists(true).setFenced(false)
179.                         .setMasterKey(ByteString.copyFrom(masterKey)).build();
180.             if (log.isDebugEnabled()) {
181.                 log.debug("Inserting new ledger {}", ledgerId);
182.             }
183.         } else {
184.             byte[] storedMasterKey = ledgerData.getMasterKey().toByteArray();
185.             if (ArrayUtil.isArrayAllZeros(storedMasterKey)) {
186.                 // update master key of the ledger
187.                 ledgerData = LedgerData.newBuilder(ledgerData).setMasterKey(ByteString.copyFrom(masterKey)).build();
188.                 if (log.isDebugEnabled()) {
189.                     log.debug("Replace old master key {} with new master key {}", storedMasterKey, masterKey);
190.                 }
191.             } else if (!Arrays.equals(storedMasterKey, masterKey) && !ArrayUtil.isArrayAllZeros(masterKey)) {
192.                 log.warn("Ledger {} masterKey in db can only be set once.", ledgerId);
193.                 throw new IOException(BookieException.create(BookieException.Code.IllegalOpException));
194.             }
195.         }
196.
197.         if (ledgers.put(ledgerId, ledgerData) == null) {
198.             ledgersCount.incrementAndGet();
199.         }
200.
201.         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
202.         pendingDeletedLedgers.remove(ledgerId);
203.     }
204.

```

Figura 16: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `LedgerMetadataIndex.setMasterKey()` dopo la modifica dei test

```

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

```

Figura 17: Risultati del Mutation Testing sul metodo `LedgerMetadataIndex.setMasterKey()`

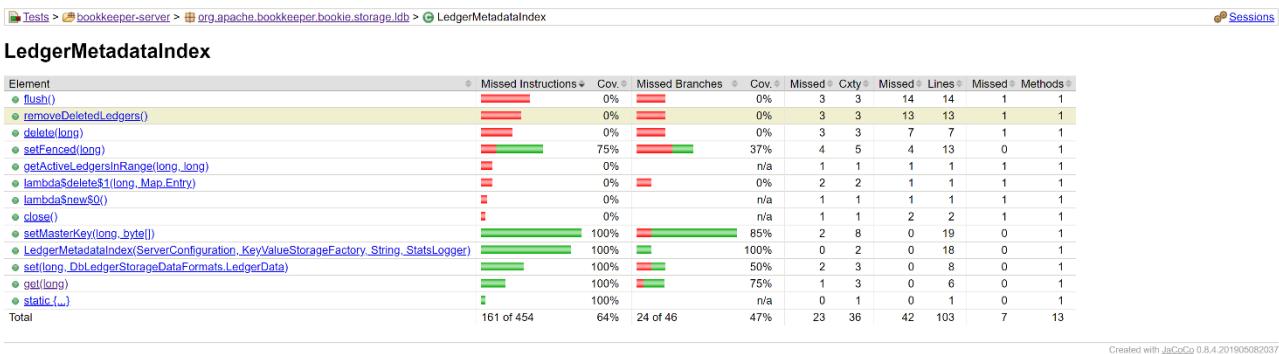


Figura 18: Statement Coverage e Branch Coverage del metodo `LedgerMetadataIndex.get()`

```

100.
101.     public LedgerData get(long ledgerId) throws IOException {
102.         LedgerData ledgerData = ledgers.get(ledgerId);
103.         if (ledgerData == null) {
104.             if (log.isDebugEnabled()) {
105.                 log.debug("Ledger not found {}", ledgerId);
106.             }
107.             throw new Bookie.NoLedgerException(ledgerId);
108.         }
109.
110.         return ledgerData;
111.     }
112.

```

Figura 19: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `LedgerMetadataIndex.get()`

```

100
101     public LedgerData get(long ledgerId) throws IOException {
102         LedgerData ledgerData = ledgers.get(ledgerId);
103.     if (ledgerData == null) {
104.         if (log.isDebugEnabled()) {
105.             log.debug("Ledger not found {}", ledgerId);
106.         }
107.         throw new Bookie.NoLedgerException(ledgerId);
108.     }
109.
110.     return ledgerData;
111. }
112

```

Figura 20: Risultati del Mutation Testing sul metodo `LedgerMetadataIndex()`

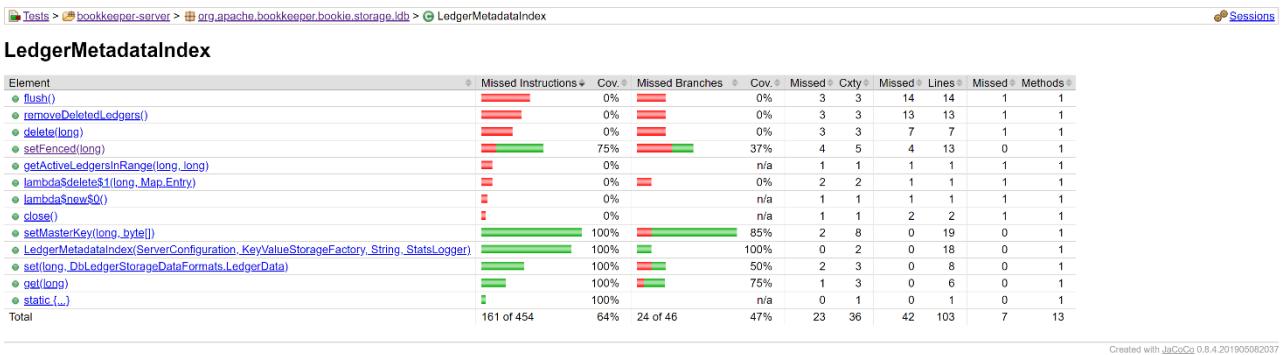


Figura 21: Statement Coverage e Branch Coverage del metodo `LedgerMetadataIndex.setFenced()`

```

148.
149.     public boolean setFenced(long ledgerId) throws IOException {
150.         LedgerData ledgerData = get(ledgerId);
151.         if (ledgerData.getFenced()) {
152.             return false;
153.         }
154.
155.         LedgerData newLedgerData = LedgerData.newBuilder(ledgerData).setFenced(true).build();
156.
157.         if (ledgers.put(ledgerId, newLedgerData) == null) {
158.             // Ledger had been deleted
159.             if (log.isDebugEnabled()) {
160.                 log.debug("Re-inserted fenced ledger {}", ledgerId);
161.             }
162.             ledgersCount.incrementAndGet();
163.         } else {
164.             if (log.isDebugEnabled()) {
165.                 log.debug("Set fenced ledger {}", ledgerId);
166.             }
167.         }
168.
169.         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
170.         pendingDeletedLedgers.remove(ledgerId);
171.         return true;
172.     }
173.

```

Figura 22: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `LedgerMetadataIndex.setFenced()`

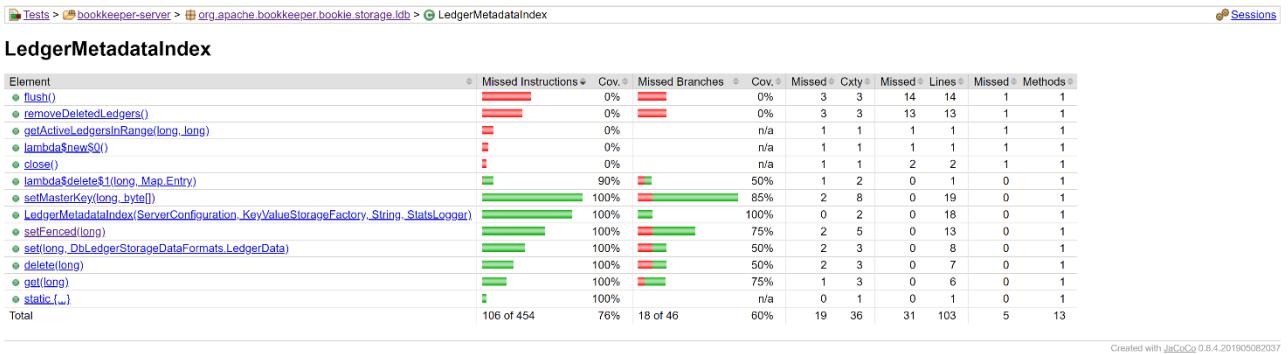


Figura 23: Statement Coverage e Branch Coverage del metodo `LedgerMetadataIndex.setFenced()` dopo la modifica dei test

```

148.
149.     public boolean setFenced(long ledgerId) throws IOException {
150.         LedgerData ledgerData = get(ledgerId);
151.         if (ledgerData.getFenced()) {
152.             return false;
153.         }
154.
155.         LedgerData newLedgerData = LedgerData.newBuilder(ledgerData).setFenced(true).build();
156.
157.         if (ledgers.put(ledgerId, newLedgerData) == null) {
158.             // Ledger had been deleted
159.             if (log.isDebugEnabled()) {
160.                 log.debug("Re-inserted fenced ledger {}", ledgerId);
161.             }
162.             ledgersCount.incrementAndGet();
163.         } else {
164.             if (log.isDebugEnabled()) {
165.                 log.debug("Set fenced ledger {}", ledgerId);
166.             }
167.         }
168.
169.         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, newLedgerData));
170.         pendingDeletedLedgers.remove(ledgerId);
171.         return true;
172.     }
173.

```

Figura 24: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `LedgerMetadataIndex.setFenced()` dopo la modifica dei test

```

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

```

Figura 25: Risultati del Mutation Testing sul metodo `LedgerMetadataIndex.setFenced()`

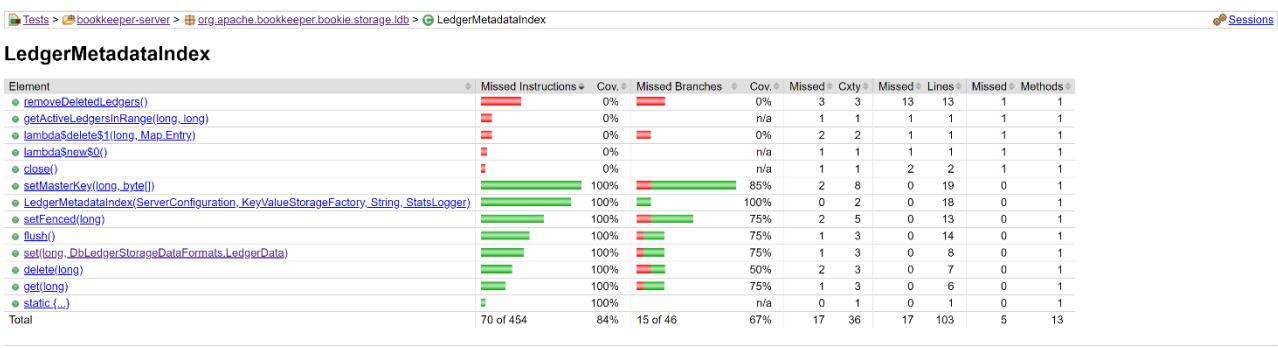


Figura 26: Statement Coverage e Branch Coverage del metodo `LedgerMetadataIndex.set()`

```

112.
113.     public void set(long ledgerId, LedgerData ledgerData) throws IOException {
114.         ledgerData = LedgerData.newBuilder(ledgerData).setExists(true).build();
115.
116.         if (ledgers.put(ledgerId, ledgerData) == null) {
117.             if (log.isDebugEnabled()) {
118.                 log.debug("Added new ledger {}", ledgerId);
119.             }
120.             ledgersCount.incrementAndGet();
121.         }
122.
123.         pendingLedgersUpdates.add(new SimpleEntry<Long, LedgerData>(ledgerId, ledgerData));
124.         pendingDeletedLedgers.remove(ledgerId);
125.     }
126.

```

Figura 27: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `LedgerMetadataIndex.set()`

```

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

```

Figura 28: Risultati del Mutation Testing sul metodo `LedgerMetadataIndex.set()`

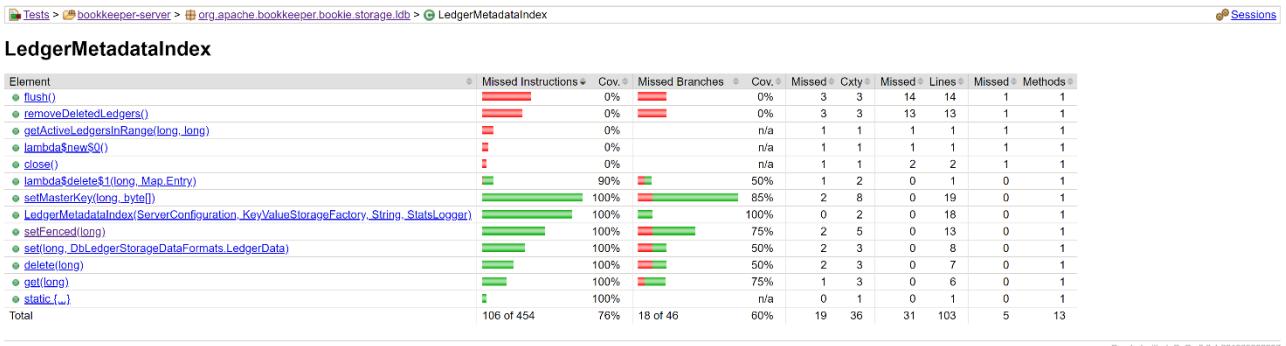


Figura 29: Statement Coverage e Branch Coverage del metodo `LedgerMetadataIndex.delete()`

```

126.
127.     public void delete(long ledgerId) throws IOException {
128.         if (ledgers.remove(ledgerId) != null) {
129.             if (log.isDebugEnabled()) {
130.                 log.debug("Removed ledger {}", ledgerId);
131.             }
132.             ledgersCount.decrementAndGet();
133.         }
134.
135.         pendingDeletedLedgers.add(ledgerId);
136.         pendingLedgersUpdates.removeIf(e -> e.getKey() == ledgerId);
137.     }
138.

```

Figura 30: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `LedgerMetadataIndex.delete()`

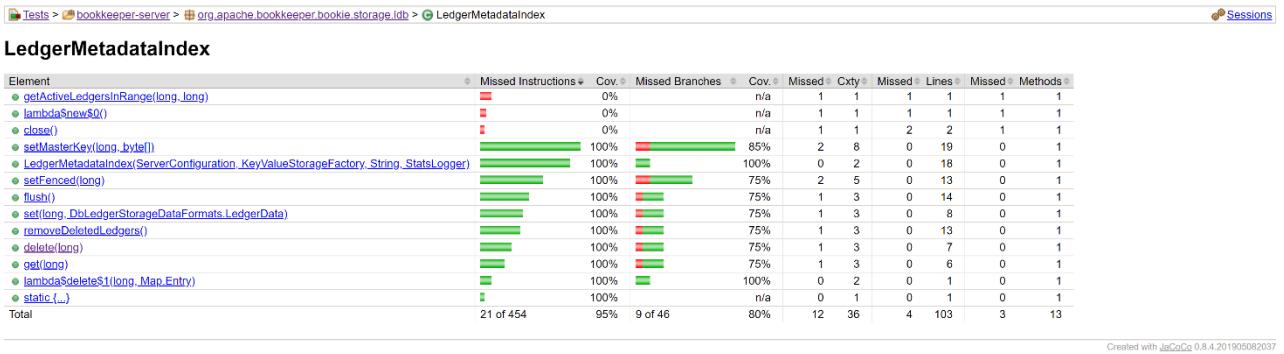


Figura 31: Statement Coverage e Branch Coverage del metodo `LedgerMetadataIndex.delete()` dopo la modifica dei test

```

126.
127.     public void delete(long ledgerId) throws IOException {
128.         if (ledgers.remove(ledgerId) != null) {
129.             if (log.isDebugEnabled()) {
130.                 log.debug("Removed ledger {}", ledgerId);
131.             }
132.             ledgersCount.decrementAndGet();
133.         }
134.
135.         pendingDeletedLedgers.add(ledgerId);
136.         pendingLedgersUpdates.removeIf(e -> e.getKey() == ledgerId);
137.     }
138.

```

Figura 32: Risultati del Mutation Testing sul metodo `LedgerMetadataIndex.delete()` dopo la modifica dei test

```

126
127.    public void delete(long ledgerId) throws IOException {
128.        if (ledgers.remove(ledgerId) != null) {
129.            if (log.isDebugEnabled()) {
130.                log.debug("Removed ledger {}", ledgerId);
131.            }
132.            ledgersCount.decrementAndGet();
133.        }
134.
135.        pendingDeletedLedgers.add(ledgerId);
136.        pendingLedgersUpdates.removeIf(e -> e.getKey() == ledgerId);
137.    }
138.

```

Figura 33: Risultati del Mutation Testing sul metodo `LedgerMetadataIndex.delete()`

| CacheMap |     |              |       |          |             |             |            |       |          |          |
|----------|-----|--------------|-------|----------|-------------|-------------|------------|-------|----------|----------|
| Version  | LOC | NumRevisions | NAuth | LOCAdded | AVGLOCAdded | MAXLOCAdded | LOCTouched | Churn | AVGChurn | MAXChurn |
| 1        | 563 | 7            | 2     | 1312     | 187         | 650         | 1985       | 639   | 91       | 626      |
| 2        | 563 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 3        | 563 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 4        | 566 | 2            | 2     | 647      | 323         | 645         | 1288       | 6     | 3        | 6        |
| 5        | 563 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 6        | 563 | 3            | 2     | 650      | 216         | 645         | 1297       | 3     | 1        | 6        |
| 7        | 563 | 1            | 1     | 642      | 642         | 642         | 642        | 642   | 642      | 642      |
| 8        | 568 | 2            | 1     | 656      | 328         | 648         | 1306       | 6     | 3        | 6        |
| 9        | 563 | 3            | 1     | 1926     | 642         | 642         | 3210       | 642   | 214      | 642      |
| 10       | 568 | 2            | 1     | 656      | 328         | 648         | 1306       | 6     | 3        | 6        |
| 11       | 568 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 12       | 568 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 13       | 568 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 14       | 568 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |

Tabella 2: Risultati delle misure della classe CacheMap

| ProxyManagerImpl |      |              |       |          |             |             |            |       |          |          |
|------------------|------|--------------|-------|----------|-------------|-------------|------------|-------|----------|----------|
| Version          | LOC  | NumRevisions | NAuth | LOCAdded | AVGLOCAdded | MAXLOCAdded | LOCTouched | Churn | AVGChurn | MAXChurn |
| 1                | 1487 | 22           | 5     | 2263     | 102         | 1269        | 2898       | 1628  | 74       | 1116     |
| 2                | 1530 | 2            | 2     | 54       | 27          | 28          | 64         | 44    | 22       | 26       |
| 3                | 1530 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 4                | 1520 | 4            | 3     | 1716     | 429         | 1662        | 3398       | 34    | 8        | 26       |
| 5                | 1530 | 2            | 2     | 54       | 27          | 28          | 64         | 44    | 22       | 26       |
| 6                | 1520 | 4            | 3     | 1716     | 429         | 1662        | 3398       | 34    | 8        | 26       |
| 7                | 1520 | 1            | 1     | 1662     | 1662        | 1662        | 1662       | 1662  | 1662     | 1662     |
| 8                | 1520 | 1            | 1     | 1662     | 1662        | 1662        | 1662       | 1662  | 1662     | 1662     |
| 9                | 1520 | 3            | 1     | 4986     | 1662        | 1662        | 8310       | 1662  | 554      | 1662     |
| 10               | 1524 | 4            | 1     | 1696     | 424         | 1664        | 3388       | 4     | 1        | 2        |
| 11               | 1524 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 12               | 1524 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 13               | 1524 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |
| 14               | 1524 | 0            | 0     | 0        | 0           | 0           | 0          | 0     | 0        | 0        |

Tabella 3: Risultati delle misure della classe ProxyManagerImpl

## CacheMap

| Element   | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxt | Missed | Lines | Missed | Methods |
|---|---------------------|------|-----------------|------|--------|-----|--------|-------|--------|---------|
| • <a href="#">pin(Object)</a>                             |                     | 0%   |                 | 0%   | 5      | 5   | 12     | 12    | 1      | 1       |
| • <a href="#">remove(Object)</a>                          |                     | 0%   |                 | 0%   | 5      | 5   | 14     | 14    | 1      | 1       |
| • <a href="#">put(Object, Object)</a>                     |                     | 50%  |                 | 50%  | 4      | 6   | 10     | 22    | 0      | 1       |
| • <a href="#">clear()</a>                                 |                     | 0%   |                 | n/a  | 1      | 1   | 10     | 10    | 1      | 1       |
| • <a href="#">unpin(Object)</a>                           |                     | 0%   |                 | 0%   | 2      | 2   | 8      | 8     | 1      | 1       |
| • <a href="#">putAll(Map, boolean)</a>                    |                     | 0%   |                 | 0%   | 4      | 4   | 5      | 5     | 1      | 1       |
| • <a href="#">containsKey(Object)</a>                     |                     | 0%   |                 | 0%   | 4      | 4   | 3      | 3     | 1      | 1       |
| • <a href="#">containsValue(Object)</a>                   |                     | 0%   |                 | 0%   | 4      | 4   | 3      | 3     | 1      | 1       |
| • <a href="#">toString()</a>                              |                     | 0%   |                 | n/a  | 1      | 1   | 3      | 3     | 1      | 1       |
| • <a href="#">notifyEntryRemovals(Set)</a>                |                     | 0%   |                 | 0%   | 3      | 3   | 5      | 5     | 1      | 1       |
| • <a href="#">cacheMapOverflowRemoved(Object, Object)</a> |                     | 0%   |                 | 0%   | 2      | 2   | 4      | 4     | 1      | 1       |
| • <a href="#">size()</a>                                  |                     | 0%   |                 | n/a  | 1      | 1   | 3      | 3     | 1      | 1       |
| • <a href="#">CacheMap(boolean, int, int, float, int)</a> |                     | 80%  |                 | 50%  | 3      | 4   | 3      | 16    | 0      | 1       |
| • <a href="#">setCacheSize(int)</a>                       |                     | 0%   |                 | 0%   | 2      | 2   | 4      | 4     | 1      | 1       |
| • <a href="#">setSoftReferenceSize(int)</a>               |                     | 0%   |                 | 0%   | 2      | 2   | 4      | 4     | 1      | 1       |
| • <a href="#">getCacheSize()</a>                          |                     | 0%   |                 | 0%   | 2      | 2   | 2      | 2     | 1      | 1       |
| • <a href="#">getSoftReferenceSize()</a>                  |                     | 0%   |                 | 0%   | 2      | 2   | 2      | 2     | 1      | 1       |
| • <a href="#">getPinnedKeys()</a>                         |                     | 0%   |                 | n/a  | 1      | 1   | 3      | 3     | 1      | 1       |
| • <a href="#">get(Object)</a>                             |                     | 82%  |                 | 66%  | 2      | 4   | 2      | 13    | 0      | 1       |

Figura 34: Statement Coverage e Branch Coverage del metodo CacheMap.put()

```
<method name="put" desc="(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;">
<du var="this" def="385" use="389" target="390" covered="0"/>
<du var="this" def="385" use="389" target="402" covered="1"/>
<du var="this" def="385" use="402" target="403" covered="0"/>
<du var="this" def="385" use="402" target="407" covered="1"/>
<du var="this" def="385" use="407" covered="1"/>
<du var="this" def="385" use="417" covered="1"/>
<du var="this" def="385" use="418" covered="1"/>
<du var="this" def="385" use="422" covered="1"/>
<du var="this" def="385" use="409" covered="1"/>
<du var="this" def="385" use="413" covered="0"/>
<du var="this" def="385" use="414" covered="0"/>
<du var="this" def="385" use="411" covered="1"/>
<du var="this" def="385" use="396" covered="0"/>
<du var="this" def="385" use="396" covered="0"/>
<du var="this" def="385" use="422" covered="0"/>
<du var="this" def="385" use="392" covered="0"/>
<du var="this" def="385" use="393" covered="0"/>
<du var="key" def="385" use="389" target="390" covered="0"/>
<du var="key" def="385" use="389" target="402" covered="1"/>
<du var="key" def="385" use="407" covered="1"/>
<du var="key" def="385" use="417" covered="1"/>
<du var="key" def="385" use="418" covered="1"/>
<du var="key" def="385" use="409" covered="1"/>
<du var="key" def="385" use="413" covered="0"/>
<du var="key" def="385" use="414" covered="0"/>
<du var="key" def="385" use="411" covered="1"/>
<du var="key" def="385" use="390" covered="0"/>
<du var="key" def="385" use="395" covered="0"/>
<du var="key" def="385" use="396" covered="0"/>
<du var="key" def="385" use="393" covered="0"/>
<du var="key" def="385" use="390" covered="0"/>
<du var="key" def="385" use="395" covered="0"/>
<du var="key" def="385" use="396" covered="0"/>
<du var="key" def="385" use="393" covered="0"/>
<du var="value" def="385" use="407" covered="1"/>
<du var="value" def="385" use="418" covered="1"/>
<du var="value" def="385" use="411" covered="0"/>
<du var="value" def="385" use="411" covered="1"/>
<du var="value" def="385" use="396" covered="0"/>
<du var="value" def="385" use="396" covered="0"/>
<du var="value" def="385" use="393" covered="0"/>
<du var="value" def="385" use="390" covered="0"/>
<du var="this.pinnedMap" def="385" use="389" target="390" covered="0"/>
<du var="this.pinnedMap" def="385" use="389" target="402" covered="1"/>
<du var="this.pinnedMap" def="385" use="390" covered="0"/>
<du var="this._pinnedSize" def="385" use="392" covered="0"/>
<du var="this.cacheMap" def="385" use="402" target="403" covered="0"/>
<du var="this.cacheMap" def="385" use="402" target="407" covered="1"/>
<du var="this.cacheMap" def="385" use="407" covered="1"/>
<du var="this.softMap" def="385" use="409" covered="1"/>
<du var="val" def="390" use="391" target="392" covered="0"/>
<du var="val" def="390" use="391" target="395" covered="0"/>
<du var="val" def="390" use="395" covered="0"/>
<du var="val" def="407" use="408" target="409" covered="1"/>
<du var="val" def="407" use="408" target="417" covered="1"/>
<du var="val" def="407" use="417" covered="1"/>
<du var="val" def="407" use="420" covered="1"/>
<du var="val" def="409" use="410" target="411" covered="1"/>
<du var="val" def="409" use="410" target="413" covered="0"/>
<du var="val" def="409" use="413" covered="0"/>
<du var="val" def="409" use="420" covered="1"/>
<counter type="DU" missed="32" covered="27"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figura 35: Data-flow coverage del metodo CacheMap.put()

```

382.
383.     @Override
384.     public Object put(Object key, Object value) {
385.         writeLock();
386.         try {
387.             // if the key is pinned, just interact directly with the pinned map
388.             Object val;
389.             if (pinnedMap.containsKey(key)) {
390.                 val = put(pinnedMap, key, value);
391.                 if (val == null) {
392.                     _pinnedSize++;
393.                     entryAdded(key, value);
394.                 } else {
395.                     entryRemoved(key, val, false);
396.                     entryAdded(key, value);
397.                 }
398.                 return val;
399.             }
400.
401.             // if no hard refs, don't put anything
402.             if (cacheMap.getMaxSize() == 0)
403.                 return null;
404.
405.             // otherwise, put the value into the map and clear it from the
406.             // soft map
407.             val = put(cacheMap, key, value);
408.             if (val == null) {
409.                 val = remove(softMap, key);
410.                 if (val == null)
411.                     entryAdded(key, value);
412.                 else {
413.                     entryRemoved(key, val, false);
414.                     entryAdded(key, value);
415.                 }
416.             } else {
417.                 entryRemoved(key, val, false);
418.                 entryAdded(key, value);
419.             }
420.             return val;
421.         } finally {
422.             writeUnlock();
423.         }
424.     }
425.

```

Figura 36: Analisi del Branch Coverage e Statement Coverage all'interno del metodo CacheMap.put()

## CacheMap

| Element   | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| <a href="#">remove(Object)</a>                          |                     | 0%   |                 | 0%   | 5      | 5    | 14     | 14    | 1      | 1       |
| <a href="#">clear()</a>                                 |                     | 0%   |                 | n/a  | 1      | 1    | 10     | 10    | 1      | 1       |
| <a href="#">unpin(Object)</a>                           |                     | 0%   |                 | 0%   | 2      | 2    | 8      | 8     | 1      | 1       |
| <a href="#">putAll(Map, boolean)</a>                    |                     | 0%   |                 | 0%   | 4      | 4    | 5      | 5     | 1      | 1       |
| <a href="#">containsKey(Object)</a>                     |                     | 0%   |                 | 0%   | 4      | 4    | 3      | 3     | 1      | 1       |
| <a href="#">containsValue(Object)</a>                   |                     | 0%   |                 | 0%   | 4      | 4    | 3      | 3     | 1      | 1       |
| <a href="#">toString()</a>                              |                     | 0%   |                 | n/a  | 1      | 1    | 3      | 3     | 1      | 1       |
| <a href="#">notifyEntryRemovals(Set)</a>                |                     | 0%   |                 | 0%   | 3      | 3    | 5      | 5     | 1      | 1       |
| <a href="#">size()</a>                                  |                     | 0%   |                 | n/a  | 1      | 1    | 3      | 3     | 1      | 1       |
| <a href="#">CacheMap(boolean, int, int, float, int)</a> |                     | 80%  |                 | 50%  | 3      | 4    | 3      | 16    | 0      | 1       |
| <a href="#">setCacheSize(int)</a>                       |                     | 0%   |                 | 0%   | 2      | 2    | 4      | 4     | 1      | 1       |
| <a href="#">setSoftReferenceSize(int)</a>               |                     | 0%   |                 | 0%   | 2      | 2    | 4      | 4     | 1      | 1       |
| <a href="#">pin(Object)</a>                             |                     | 80%  |                 | 50%  | 3      | 5    | 1      | 12    | 0      | 1       |
| <a href="#">getCacheSize()</a>                          |                     | 0%   |                 | 0%   | 2      | 2    | 2      | 2     | 1      | 1       |
| <a href="#">getSoftReferenceSize()</a>                  |                     | 0%   |                 | 0%   | 2      | 2    | 2      | 2     | 1      | 1       |
| <a href="#">getPinnedKeys()</a>                         |                     | 0%   |                 | n/a  | 1      | 1    | 3      | 3     | 1      | 1       |
| <a href="#">CacheMap(boolean, int)</a>                  |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">CacheMap(boolean, int, int, float)</a>      |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">get(Object)</a>                             |                     | 82%  |                 | 66%  | 2      | 4    | 2      | 13    | 0      | 1       |
| <a href="#">isEmpty()</a>                               |                     | 0%   |                 | 0%   | 2      | 2    | 1      | 1     | 1      | 1       |
| <a href="#">softMapOverflowRemoved(Object, Object)</a>  |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">softMapValueExpired(Object)</a>             |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">keySet()</a>                                |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">values()</a>                                |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">entrySet()</a>                              |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">cacheMapOverflowRemoved(Object, Object)</a> |                     | 76%  |                 | 50%  | 1      | 2    | 1      | 4     | 0      | 1       |
| <a href="#">CacheMap()</a>                              |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">CacheMap(boolean)</a>                       |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">putAll(Map)</a>                             |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">isLRU()</a>                                 |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">put(Object, Object)</a>                     |                     | 100% |                 | 100% | 0      | 6    | 0      | 22    | 0      | 1       |

Figura 37: Statement Coverage e Branch Coverage del metodo CacheMap.put() dopo la modifica dei test

```

▼<method name="put" desc="(Ljava/lang/Object;Ljava/lang/Object;)Ljava/lang/Object;">
<du var="this" def="385" use="389" target="390" covered="1"/>
<du var="this" def="385" use="389" target="402" covered="1"/>
<du var="this" def="385" use="402" target="403" covered="1"/>
<du var="this" def="385" use="402" target="407" covered="1"/>
<du var="this" def="385" use="407" covered="1"/>
<du var="this" def="385" use="417" covered="1"/>
<du var="this" def="385" use="418" covered="1"/>
<du var="this" def="385" use="422" covered="1"/>
<du var="this" def="385" use="489" covered="1"/>
<du var="this" def="385" use="413" covered="1"/>
<du var="this" def="385" use="414" covered="1"/>
<du var="this" def="385" use="411" covered="1"/>
<du var="this" def="385" use="422" covered="1"/>
<du var="this" def="385" use="390" covered="1"/>
<du var="this" def="385" use="395" covered="1"/>
<du var="this" def="385" use="396" covered="1"/>
<du var="this" def="385" use="422" covered="1"/>
<du var="this" def="385" use="392" covered="1"/>
<du var="this" def="385" use="393" covered="1"/>
<du var="key" def="385" use="389" target="390" covered="1"/>
<du var="key" def="385" use="389" target="402" covered="1"/>
<du var="key" def="385" use="407" covered="1"/>
<du var="key" def="385" use="417" covered="1"/>
<du var="key" def="385" use="418" covered="1"/>
<du var="key" def="385" use="409" covered="1"/>
<du var="key" def="385" use="413" covered="1"/>
<du var="key" def="385" use="414" covered="1"/>
<du var="key" def="385" use="411" covered="1"/>
<du var="key" def="385" use="390" covered="1"/>
<du var="key" def="385" use="395" covered="1"/>
<du var="key" def="385" use="396" covered="1"/>
<du var="key" def="385" use="393" covered="1"/>
<du var="value" def="385" use="407" covered="1"/>
<du var="value" def="385" use="418" covered="1"/>
<du var="value" def="385" use="414" covered="1"/>
<du var="value" def="385" use="411" covered="1"/>
<du var="value" def="385" use="390" covered="1"/>
<du var="value" def="385" use="396" covered="1"/>
<du var="value" def="385" use="393" covered="1"/>
<du var="this.pinnedMap" def="385" use="389" target="390" covered="1"/>
<du var="this.pinnedMap" def="385" use="389" target="402" covered="1"/>
<du var="this.pinnedMap" def="385" use="390" covered="1"/>
<du var="this._pinnedsize" def="385" use="392" covered="1"/>
<du var="this.cacheMap" def="385" use="402" target="403" covered="1"/>
<du var="this.cacheMap" def="385" use="402" target="407" covered="1"/>
<du var="this.cacheMap" def="385" use="407" covered="1"/>
<du var="this.softMap" def="385" use="409" covered="1"/>
<du var="val" def="390" use="391" target="392" covered="1"/>
<du var="val" def="390" use="391" target="395" covered="1"/>
<du var="val" def="390" use="395" covered="1"/>
<du var="val" def="390" use="398" covered="1"/>
<du var="val" def="407" use="408" target="409" covered="1"/>
<du var="val" def="407" use="408" target="417" covered="1"/>
<du var="val" def="407" use="417" covered="1"/>
<du var="val" def="407" use="420" covered="1"/>
<du var="val" def="409" use="410" target="411" covered="1"/>
<du var="val" def="409" use="410" target="413" covered="1"/>
<du var="val" def="409" use="413" covered="1"/>
<du var="val" def="409" use="420" covered="1"/>
<counter type="DU" missed="0" covered="59"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 38: Data-flow coverage del metodo CacheMap.put() dopo la modifica dei test

```

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

```

Figura 39: Risultati del Mutation Testing sul metodo CacheMap.put()

```

382
383     @Override
384     public Object put(Object key, Object value) {
385 1       writeLock();
386       try {
387           // if the key is pinned, just interact directly with the pinned map
388           Object val;
389 1           if (pinnedMap.containsKey(key)) {
390               val = put(pinnedMap, key, value);
391 1               if (val == null) {
392 1                   _pinnedSize++;
393 1                   entryAdded(key, value);
394               } else {
395 1                   entryRemoved(key, val, false);
396 1                   entryAdded(key, value);
397               }
398 1           return val;
399       }
400
401           // if no hard refs, don't put anything
402 1           if (cacheMap.getMaxSize() == 0)
403 1               return null;
404
405           // otherwise, put the value into the map and clear it from the
406           // soft map
407           val = put(cacheMap, key, value);
408 1           if (val == null) {
409               val = remove(softMap, key);
410 1               if (val == null)
411 1                   entryAdded(key, value);
412               else {
413 1                   entryRemoved(key, val, false);
414 1                   entryAdded(key, value);
415               }
416           } else {
417 1               entryRemoved(key, val, false);
418 1               entryAdded(key, value);
419           }
420 1           return val;
421       } finally {
422 1           writeUnlock();
423       }
424   }
425 }
```

Figura 40: Risultati del Mutation Testing sul metodo CacheMap.put() dopo la modifica dei test

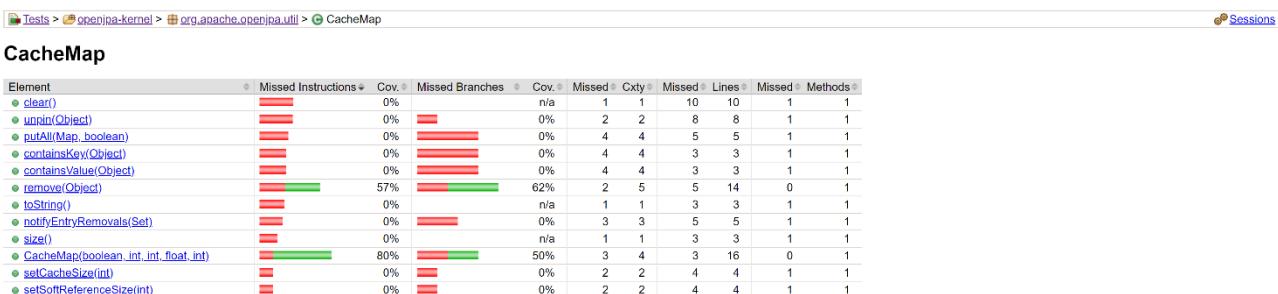


Figura 41: Statement Coverage e Branch Coverage del metodo CacheMap.remove()

```

▼<method name="remove" desc="(Ljava/lang/Object;)Ljava/lang/Object;">
<du var="this" def="447" use="452" target="454" covered="0"/>
<du var="this" def="447" use="452" target="462" covered="1"/>
<du var="this" def="447" use="462" covered="1"/>
<du var="this" def="447" use="470" covered="1"/>
<du var="this" def="447" use="466" covered="1"/>
<du var="this" def="447" use="464" covered="1"/>
<du var="this" def="447" use="454" covered="0"/>
<du var="this" def="447" use="470" covered="0"/>
<du var="this" def="447" use="456" covered="0"/>
<du var="this" def="447" use="457" covered="0"/>
<du var="key" def="447" use="452" target="454" covered="0"/>
<du var="key" def="447" use="452" target="462" covered="1"/>
<du var="key" def="447" use="462" covered="1"/>
<du var="key" def="447" use="466" covered="1"/>
<du var="key" def="447" use="464" covered="1"/>
<du var="key" def="447" use="454" covered="0"/>
<du var="key" def="447" use="457" covered="0"/>
<du var="this.pinnedMap" def="447" use="452" target="454" covered="0"/>
<du var="this.pinnedMap" def="447" use="452" target="462" covered="1"/>
<du var="this.pinnedMap" def="447" use="454" covered="0"/>
<du var="this._pinnedSize" def="447" use="456" covered="0"/>
<du var="this.cacheMap" def="447" use="462" covered="1"/>
<du var="this.softMap" def="447" use="464" covered="1"/>
<du var="val" def="454" use="455" target="456" covered="0"/>
<du var="val" def="454" use="455" target="459" covered="0"/>
<du var="val" def="454" use="459" covered="0"/>
<du var="val" def="454" use="457" covered="0"/>
<du var="val" def="462" use="463" target="464" covered="1"/>
<du var="val" def="462" use="463" target="465" covered="1"/>
<du var="val" def="462" use="465" target="466" covered="1"/>
<du var="val" def="462" use="465" target="468" covered="0"/>
<du var="val" def="462" use="468" covered="1"/>
<du var="val" def="462" use="466" covered="1"/>
<du var="val" def="464" use="465" target="466" covered="0"/>
<du var="val" def="464" use="465" target="468" covered="1"/>
<du var="val" def="464" use="468" covered="1"/>
<du var="val" def="464" use="466" covered="0"/>
<counter type="DU" missed="18" covered="19"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 42: Data-flow coverage del metodo CacheMap.remove()

```
440.  
441.        /**  
442.         * If <code>key</code> is pinned into the cache, the pin is  
443.         * cleared and the object is removed.  
444.         */  
445.        @Override  
446.        public Object remove(Object key) {  
447.            writeLock();  
448.            try {  
449.                // if the key is pinned, just interact directly with the  
450.                // pinned map  
451.                Object val;  
452.                if (pinnedMap.containsKey(key)) {  
453.                    // re-put with null value; we still want key pinned  
454.                    val = put(pinnedMap, key, null);  
455.                    if (val != null) {  
456.                        _pinnedSize--;  
457.                        entryRemoved(key, val, false);  
458.                    }  
459.                return val;  
460.            }  
461.  
462.            val = remove(cacheMap, key);  
463.            if (val == null)  
464.                val = softMap.remove(key);  
465.            if (val != null)  
466.                entryRemoved(key, val, false);  
467.  
468.            return val;  
469.        } finally {  
470.            writeUnlock();  
471.        }  
472.    }  
473.
```

Figura 43: Analisi del Branch Coverage e Statement Coverage all'interno del metodo CacheMap.remove()

```

440
441     /**
442      * If <code>key</code> is pinned into the cache, the pin is
443      * cleared and the object is removed.
444      */
445     @Override
446     public Object remove(Object key) {
447 1       writeLock();
448       try {
449           // if the key is pinned, just interact directly with the
450           // pinned map
451           Object val;
452 1           if (pinnedMap.containsKey(key)) {
453               // re-put with null value; we still want key pinned
454               val = put(pinnedMap, key, null);
455 1               if (val != null) {
456 1                   _pinnedSize--;
457 1                   entryRemoved(key, val, false);
458               }
459 1           return val;
460       }
461
462       val = remove(cacheMap, key);
463 1       if (val == null)
464           val = softMap.remove(key);
465 1       if (val != null)
466 1           entryRemoved(key, val, false);
467
468 1       return val;
469   } finally {
470 1       writeUnlock();
471   }
472 }
473

```

Figura 44: Risultati del Mutation Testing sul metodo CacheMap.remove()

```

440
441     /**
442      * If <code>key</code> is pinned into the cache, the pin is
443      * cleared and the object is removed.
444      */
445     @Override
446     public Object remove(Object key) {
447 1       writeLock();
448       try {
449           // if the key is pinned, just interact directly with the
450           // pinned map
451           Object val;
452 1           if (pinnedMap.containsKey(key)) {
453               // re-put with null value; we still want key pinned
454               val = put(pinnedMap, key, null);
455 1               if (val != null) {
456 1                   _pinnedSize--;
457 1                   entryRemoved(key, val, false);
458               }
459 1           return val;
460       }
461
462       val = remove(cacheMap, key);
463 1       if (val == null)
464           val = softMap.remove(key);
465 1       if (val != null)
466 1           entryRemoved(key, val, false);
467
468 1       return val;
469   } finally {
470 1       writeUnlock();
471   }
472 }
473

```

Figura 45: Risultati del Mutation Testing sul metodo CacheMap.remove() dopo la modifica dei test

## CacheMap

| Element   | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| <a href="#">clear()</a>                                 |                     | 0%   |                 | n/a  | 1      | 1    | 10     | 10    | 1      | 1       |
| <a href="#">unpin(Object)</a>                           |                     | 0%   |                 | 0%   | 2      | 2    | 8      | 8     | 1      | 1       |
| <a href="#">putAll(Map, boolean)</a>                    |                     | 0%   |                 | 0%   | 4      | 4    | 5      | 5     | 1      | 1       |
| <a href="#">containsKey(Object)</a>                     |                     | 0%   |                 | 0%   | 4      | 4    | 3      | 3     | 1      | 1       |
| <a href="#">containsValue(Object)</a>                   |                     | 0%   |                 | 0%   | 4      | 4    | 3      | 3     | 1      | 1       |
| <a href="#">toString()</a>                              |                     | 0%   |                 | n/a  | 1      | 1    | 3      | 3     | 1      | 1       |
| <a href="#">notifyEntryRemovals(Set)</a>                |                     | 0%   |                 | 0%   | 3      | 3    | 5      | 5     | 1      | 1       |
| <a href="#">size()</a>                                  |                     | 0%   |                 | n/a  | 1      | 1    | 3      | 3     | 1      | 1       |
| <a href="#">CacheMap(boolean, int, int, float, int)</a> |                     | 80%  |                 | 50%  | 3      | 4    | 3      | 16    | 0      | 1       |
| <a href="#">setCacheSize(int)</a>                       |                     | 0%   |                 | 0%   | 2      | 2    | 4      | 4     | 1      | 1       |
| <a href="#">setSoftReferenceSize(int)</a>               |                     | 0%   |                 | 0%   | 2      | 2    | 4      | 4     | 1      | 1       |
| <a href="#">getCacheSize()</a>                          |                     | 0%   |                 | 0%   | 2      | 2    | 2      | 2     | 1      | 1       |
| <a href="#">getSoftReferenceSize()</a>                  |                     | 0%   |                 | 0%   | 2      | 2    | 2      | 2     | 1      | 1       |
| <a href="#">getPinnedKeys()</a>                         |                     | 0%   |                 | n/a  | 1      | 1    | 3      | 3     | 1      | 1       |
| <a href="#">get(Object)</a>                             |                     | 82%  |                 | 66%  | 2      | 4    | 2      | 13    | 0      | 1       |
| <a href="#">isEmpty()</a>                               |                     | 0%   |                 | 0%   | 2      | 2    | 1      | 1     | 1      | 1       |
| <a href="#">softMapOverflowRemoved(Object, Object)</a>  |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">softMapValueExpired(Object)</a>             |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">keySet()</a>                                |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">values()</a>                                |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">entrySet()</a>                              |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">cacheMapOverflowRemoved(Object, Object)</a> |                     | 76%  |                 | 50%  | 1      | 2    | 1      | 4     | 0      | 1       |
| <a href="#">CacheMap()</a>                              |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">putAll(Map)</a>                             |                     | 0%   |                 | n/a  | 1      | 1    | 2      | 2     | 1      | 1       |
| <a href="#">isLRU()</a>                                 |                     | 0%   |                 | n/a  | 1      | 1    | 1      | 1     | 1      | 1       |
| <a href="#">put(Object, Object)</a>                     |                     | 100% |                 | 100% | 0      | 6    | 0      | 22    | 0      | 1       |
| <a href="#">pin(Object)</a>                             |                     | 100% |                 | 100% | 0      | 5    | 0      | 12    | 0      | 1       |
| <a href="#">remove(Object)</a>                          |                     | 100% |                 | 100% | 0      | 5    | 0      | 14    | 0      | 1       |

Figura 46: Statement Coverage e Branch Coverage del metodo CacheMap.pin()

```

▼<method name="pin" desc="(Ljava/lang/Object;)Z">
<du var="this" def="291" use="295" target="296" covered="1"/>
<du var="this" def="291" use="295" target="299" covered="1"/>
<du var="this" def="291" use="299" covered="1"/>
<du var="this" def="291" use="304" covered="1"/>
<du var="this" def="291" use="311" covered="1"/>
<du var="this" def="291" use="306" covered="1"/>
<du var="this" def="291" use="311" covered="1"/>
<du var="this" def="291" use="301" covered="1"/>
<du var="this" def="291" use="296" target="296" covered="1"/>
<du var="this" def="291" use="296" target="296" covered="1"/>
<du var="this" def="291" use="311" covered="1"/>
<du var="key" def="291" use="295" target="296" covered="1"/>
<du var="key" def="291" use="295" target="299" covered="1"/>
<du var="key" def="291" use="299" covered="1"/>
<du var="key" def="291" use="304" covered="1"/>
<du var="key" def="291" use="301" covered="1"/>
<du var="key" def="291" use="296" target="296" covered="1"/>
<du var="key" def="291" use="296" target="296" covered="1"/>
<du var="this.pinnedMap" def="291" use="295" target="296" covered="1"/>
<du var="this.pinnedMap" def="291" use="295" target="299" covered="1"/>
<du var="this.pinnedMap" def="291" use="304" covered="1"/>
<du var="this.pinnedMap" def="291" use="296" target="296" covered="1"/>
<du var="this.pinnedMap" def="291" use="296" target="296" covered="1"/>
<du var="this.cacheMap" def="291" use="299" covered="1"/>
<du var="this.softMap" def="291" use="301" covered="1"/>
<du var="this._pinnedSize" def="291" use="306" covered="1"/>
<du var="val" def="299" use="300" target="301" covered="1"/>
<du var="val" def="299" use="300" target="304" covered="1"/>
<du var="val" def="299" use="304" covered="1"/>
<du var="val" def="299" use="305" target="306" covered="1"/>
<du var="val" def="299" use="305" target="309" covered="0"/>
<du var="val" def="301" use="304" covered="1"/>
<du var="val" def="301" use="305" target="306" covered="1"/>
<du var="val" def="301" use="305" target="309" covered="1"/>
<counter type="DU" missed="1" covered="33"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 47: Data-flow coverage del metodo CacheMap.pin()

```

281
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 given key's value was pinned; false if no value
288      * for the given key is cached
289      */
290     public boolean pin(Object key) {
291         writeLock();
292         try {
293             // if we don't have a pinned map we need to create one; else if the
294             // pinned map already contains the key, nothing to do
295             if (pinnedMap.containsKey(key))
296                 return pinnedMap.get(key) != null;
297
298             // check other maps for key
299             Object val = remove(cacheMap, key);
300             if (val == null)
301                 val = remove(softMap, key);
302
303             // pin key
304             put(pinnedMap, key, val);
305             if (val != null) {
306                 _pinnedSize++;
307                 return true;
308             }
309             return false;
310         } finally {
311             writeUnlock();
312         }
313     }
314

```

Figura 48: Risultati del Mutation Testing sul metodo CacheMap.pin()

```

281
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 given key's value was pinned; false if no value
288      * for the given key is cached
289      */
290     public boolean pin(Object key) {
291         writeLock();
292         try {
293             // if we don't have a pinned map we need to create one; else if the
294             // pinned map already contains the key, nothing to do
295             if (pinnedMap.containsKey(key))
296                 return pinnedMap.get(key) != null;
297
298             // check other maps for key
299             Object val = remove(cacheMap, key);
300             if (val == null)
301                 val = remove(softMap, key);
302
303             // pin key
304             put(pinnedMap, key, val);
305             if (val != null) {
306                 _pinnedSize++;
307                 return true;
308             }
309             return false;
310         } finally {
311             writeUnlock();
312         }
313     }
314

```

Figura 49: Risultati del Mutation Testing sul metodo CacheMap.pin() dopo la modifica dei test

## ProxyManagerImpl

| Element   | Missed Instructions | Cov.% | Missed Branches | Cov.% | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---------------------|-------|-----------------|-------|--------|------|--------|-------|--------|---------|
| addProxyBeanMethods(BCCClass_Class_Constructor) | ██████              | 80%   | ██████          | 50%   | 4      | 5    | 4      | 30    | 0      | 1       |
| addProxyCalendarMethods(BCCClass_Class)         | ██████              | 0%    | ██████          | 0%    | 4      | 4    | 62     | 62    | 1      | 1       |
| addProxyCollectionMethods(BCCClass_Class)       | ██████              | 0%    | ██████          | 0%    | 10     | 10   | 77     | 77    | 1      | 1       |
| addProxyDateMethods(BCCClass_Class)             | ██████              | 0%    | ██████          | 0%    | 14     | 14   | 59     | 59    | 1      | 1       |
| addProxyMapMethods(BCCClass_Class)              | ██████              | 0%    | ██████          | 0%    | 10     | 10   | 87     | 87    | 1      | 1       |
| addProxyMethods(BCCClass_boolean)               | ██████              | 100%  | ██████          | 50%   | 1      | 2    | 0      | 57    | 0      | 1       |
| addWriteReplaceMethod(BCCClass_boolean)         | ███                 | 96%   | ██████          | 50%   | 1      | 2    | 0      | 11    | 0      | 1       |
| allowsDuplicates(Class)                         | █                   | 0%    | █               | 0%    | 2      | 2    | 1      | 1     | 1      | 1       |
| assertNotFinal(Class)                           | █                   | 0%    | █               | 0%    | 2      | 2    | 3      | 3     | 1      | 1       |
| copyArray(Object)                               | █                   | 0%    | █               | 0%    | 2      | 2    | 10     | 10    | 1      | 1       |
| copyCalendar(Calendar)                          | █                   | 0%    | █               | 0%    | 3      | 3    | 6      | 6     | 1      | 1       |
| copyCollection(Collection)                      | █                   | 0%    | █               | 0%    | 3      | 3    | 6      | 6     | 1      | 1       |
| copyCustom(Object)                              | ███                 | 55%   | ██████          | 62%   | 6      | 9    | 6      | 16    | 0      | 1       |

Figura 50: Statement Coverage e Branch Coverage del metodo ProxyManagerImpl.copyCustom()

```

▼<method name="copyCustom" desc="(Ljava/lang/Object;)Ljava/lang/Object;">
<du var="this" def="277" use="291" covered="1"/>
<du var="this" def="277" use="290" covered="0"/>
<du var="this" def="277" use="288" covered="0"/>
<du var="this" def="277" use="286" covered="0"/>
<du var="this" def="277" use="284" covered="0"/>
<du var="orig" def="277" use="277" target="278" covered="1"/>
<du var="orig" def="277" use="277" target="279" covered="1"/>
<du var="orig" def="277" use="279" target="280" covered="0"/>
<du var="orig" def="277" use="279" target="281" covered="1"/>
<du var="orig" def="277" use="281" target="282" covered="0"/>
<du var="orig" def="277" use="281" target="283" covered="1"/>
<du var="orig" def="277" use="283" target="284" covered="0"/>
<du var="orig" def="277" use="283" target="285" covered="1"/>
<du var="orig" def="277" use="285" target="286" covered="0"/>
<du var="orig" def="277" use="285" target="287" covered="1"/>
<du var="orig" def="277" use="287" target="288" covered="0"/>
<du var="orig" def="277" use="287" target="289" covered="1"/>
<du var="orig" def="277" use="289" target="290" covered="0"/>
<du var="orig" def="277" use="289" target="291" covered="1"/>
<du var="orig" def="277" use="291" covered="1"/>
<du var="orig" def="277" use="292" covered="1"/>
<du var="orig" def="277" use="290" covered="0"/>
<du var="orig" def="277" use="288" covered="0"/>
<du var="orig" def="277" use="286" covered="0"/>
<du var="orig" def="277" use="284" covered="0"/>
<du var="orig" def="277" use="280" covered="0"/>
<du var="proxy" def="291" use="292" target="292" covered="1"/>
<du var="proxy" def="291" use="292" target="292" covered="1"/>
<du var="proxy" def="291" use="292" covered="1"/>
<counter type="DU" missed="15" covered="14"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 51: Data-flow coverage del metodo ProxyManagerImpl.copyCustom()

```

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.     }
294.

```

Figura 52: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `ProxyManagerImpl.copyCustom()`

## ProxyManagerImpl

| Element   | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|---|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| • <code>addProxyBeanMethods(BCClass, Class, Constructor)</code> | ██████              | 80%  | ██████          | 50%  | 4      | 5    | 4      | 30    | 0      | 1       |
| • <code>addProxyCalendarMethods(BCClass, Class)</code>          | ██████              | 0%   | ██████          | 0%   | 4      | 4    | 62     | 62    | 1      | 1       |
| • <code>addProxyCollectionMethods(BCClass, Class)</code>        | ██████              | 0%   | ██████          | 0%   | 10     | 10   | 77     | 77    | 1      | 1       |
| • <code>addProxyDateMethods(BCClass, Class)</code>              | ██████              | 0%   | ██████          | 0%   | 14     | 14   | 59     | 59    | 1      | 1       |
| • <code>addProxyMapMethods(BCClass, Class)</code>               | ██████              | 0%   | ██████          | 0%   | 10     | 10   | 87     | 87    | 1      | 1       |
| • <code>addProxyMethods(BCClass, boolean)</code>                | ██████              | 100% | ██████          | 50%  | 1      | 2    | 0      | 57    | 0      | 1       |
| • <code>addWriteReplaceMethod(BCClass, boolean)</code>          | ██████              | 96%  | ██████          | 50%  | 1      | 2    | 0      | 11    | 0      | 1       |
| • <code>allowsDuplicates(Class)</code>                          | ██████              | 0%   | ██████          | 0%   | 2      | 2    | 1      | 1     | 1      | 1       |
| • <code>assertNotFinal(Class)</code>                            | ██████              | 0%   | ██████          | 0%   | 2      | 2    | 3      | 3     | 1      | 1       |
| • <code>copyArray(Object)</code>                                | ██████              | 0%   | ██████          | 0%   | 2      | 2    | 10     | 10    | 1      | 1       |
| • <code>copyCalendar(Calendar)</code>                           | ██████              | 65%  | ██████          | 50%  | 2      | 3    | 2      | 6     | 0      | 1       |
| • <code>copyCollection(Collection)</code>                       | ██████              | 65%  | ██████          | 50%  | 2      | 3    | 2      | 6     | 0      | 1       |
| • <code>copyCustom(Object)</code>                               | ██████              | 96%  | ██████          | 93%  | 1      | 9    | 1      | 16    | 0      | 1       |

Figura 53: Statement Coverage e Branch Coverage del metodo `ProxyManagerImpl.copyCustom()` dopo la modifica dei casi di test

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

Figura 54: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `ProxyManagerImpl.copyCustom()` dopo la modifica dei casi di test

```

▼<method name="copyCustom" desc="(Ljava/lang/Object;)Ljava/lang/Object;">
<du var="this" def="277" use="291" covered="1"/>
<du var="this" def="277" use="290" covered="1"/>
<du var="this" def="277" use="288" covered="1"/>
<du var="this" def="277" use="286" covered="1"/>
<du var="this" def="277" use="284" covered="1"/>
<du var="orig" def="277" use="277" target="278" covered="1"/>
<du var="orig" def="277" use="277" target="279" covered="1"/>
<du var="orig" def="277" use="279" target="280" covered="1"/>
<du var="orig" def="277" use="279" target="281" covered="1"/>
<du var="orig" def="277" use="281" target="282" covered="0"/>
<du var="orig" def="277" use="281" target="283" covered="1"/>
<du var="orig" def="277" use="283" target="284" covered="1"/>
<du var="orig" def="277" use="283" target="285" covered="1"/>
<du var="orig" def="277" use="285" target="286" covered="1"/>
<du var="orig" def="277" use="285" target="287" covered="1"/>
<du var="orig" def="277" use="287" target="288" covered="1"/>
<du var="orig" def="277" use="287" target="289" covered="1"/>
<du var="orig" def="277" use="289" target="290" covered="1"/>
<du var="orig" def="277" use="289" target="291" covered="1"/>
<du var="orig" def="277" use="291" covered="1"/>
<du var="orig" def="277" use="292" covered="1"/>
<du var="orig" def="277" use="290" covered="1"/>
<du var="orig" def="277" use="288" covered="1"/>
<du var="orig" def="277" use="286" covered="1"/>
<du var="orig" def="277" use="284" covered="1"/>
<du var="orig" def="277" use="280" covered="1"/>
<du var="proxy" def="291" use="292" target="292" covered="1"/>
<du var="proxy" def="291" use="292" target="292" covered="1"/>
<du var="proxy" def="291" use="292" covered="1"/>
<counter type="DU" missed="1" covered="28"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 55: Data-flow coverage del metodo `ProxyManagerImpl.copyCustom()` dopo la modifica dei test

```

274
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);

```

Figura 56: Risultati del Mutation Testing sul metodo `ProxyManagerImpl.copyCustom()`

## ProxyManagerImpl

| Element  | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|--|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| toProxyableMapType(Class)  | 0%                  | 100% | 0%              | 100% | 5      | 5    | 9      | 9     | 1      | 1       |
| toProxyableCollectionType(Class)   | 0%                  | 100% | 0%              | 100% | 5      | 5    | 9      | 9     | 1      | 1       |
| toHelperParameters(Class[], Class)   | 0%                  | 100% | n/a             | 1    | 1      | 4    | 4      | 1     | 1      | 1       |
| toHelperAfterParameters(Class[], Class, Class)                                   | 0%                  | 100% | 0%              | 100% | 7      | 7    | 15     | 15    | 1      | 1       |
| toConcreteType(Class, Map)   | 0%                  | 100% | 0%              | 100% | 4      | 4    | 9      | 9     | 1      | 1       |
| static(...)  | 100%                | n/a  | 0               | 1    | 0      | 13   | 0      | 1     | 0      | 1       |
| startsWith(String, String)   | 100%                | 66%  | 2               | 4    | 0      | 3    | 0      | 1     | 0      | 1       |
| setUpUnproxyable(String)   | 0%                  | 100% | 0%              | 100% | 2      | 2    | 3      | 3     | 1      | 1       |
| setTrackChanges(boolean)   | 0%                  | n/a  | 1               | 1    | 2      | 2    | 1      | 1     | 1      | 1       |
| setDelayCollectionLoading(boolean)   | 0%                  | n/a  | 1               | 1    | 2      | 2    | 1      | 1     | 1      | 1       |
| setAssertAllowedType(boolean)  | 0%                  | n/a  | 1               | 1    | 2      | 2    | 1      | 1     | 1      | 1       |
| proxySetters(BCClass, Class)   | 98%                 | 70%  | 3               | 6    | 0      | 9    | 0      | 1     | 0      | 1       |
| proxySetter(BCClass, Class, Method)  | 100%                | 100% | 0               | 2    | 0      | 16   | 0      | 1     | 0      | 1       |
| proxyRecognizedMethods(BCClass, Class, Class, Class)                             | 0%                  | 100% | 0%              | 100% | 6      | 6    | 32     | 32    | 1      | 1       |
| proxyOverrideMethod(BCClass, Method, Method, Class[])                            | 0%                  | 100% | 0%              | 100% | 2      | 2    | 12     | 12    | 1      | 1       |
| ProxyManagerImpl()   | 100%                | n/a  | 0               | 1    | 0      | 8    | 0      | 1     | 0      | 1       |
| proxyBeforeAfterMethod(BCClass, Class, Method, Method, Class[], Method, Class[]) | 0%                  | 100% | 0%              | 100% | 11     | 11   | 37     | 37    | 1      | 1       |
| nextProxyId()  | 100%                | n/a  | 0               | 1    | 0      | 1    | 0      | 1     | 0      | 1       |
| newMapProxy(Class, Class, Class, Comparator, boolean)                            | 0%                  | 100% | 0%              | 100% | 3      | 3    | 3      | 3     | 1      | 1       |
| newDateProxy(Class)  | 100%                | n/a  | 0               | 1    | 0      | 2    | 0      | 1     | 0      | 1       |
| newCustomProxy(Object, boolean)  | 27%                 | 45%  | 10              | 13   | 22     | 33   | 0      | 1     | 0      | 1       |

Figura 57: Statement Coverage e Branch Coverage del metodo ProxyManagerImpl.newCustomProxy()

```

▼<method name="newCustomProxy" desc="(Ljava/lang/Object;Z)Lorg/apache/openjpa/util/Proxy;">
<du var="this" def="297" use="335" covered="1"/>
<du var="this" def="297" use="329" covered="0"/>
<du var="this" def="297" use="322" covered="0"/>
<du var="this" def="297" use="317" covered="0"/>
<du var="this" def="297" use="309" covered="0"/>
<du var="orig" def="297" use="297" target="298" covered="1"/>
<du var="orig" def="297" use="297" target="299" covered="1"/>
<du var="orig" def="297" use="299" target="300" covered="0"/>
<du var="orig" def="297" use="299" target="301" covered="1"/>
<du var="orig" def="297" use="301" target="302" covered="0"/>
<du var="orig" def="297" use="301" target="303" covered="1"/>
<du var="orig" def="297" use="303" target="304" covered="0"/>
<du var="orig" def="297" use="303" target="306" covered="1"/>
<du var="orig" def="297" use="306" target="307" covered="0"/>
<du var="orig" def="297" use="306" target="314" covered="1"/>
<du var="orig" def="297" use="314" target="315" covered="0"/>
<du var="orig" def="297" use="314" target="321" covered="1"/>
<du var="orig" def="297" use="321" target="322" covered="0"/>
<du var="orig" def="297" use="321" target="328" covered="1"/>
<du var="orig" def="297" use="328" target="329" covered="0"/>
<du var="orig" def="297" use="328" target="335" covered="1"/>
<du var="orig" def="297" use="335" covered="1"/>
<du var="orig" def="297" use="336" covered="1"/>
<du var="orig" def="297" use="329" covered="0"/>
<du var="orig" def="297" use="331" covered="0"/>
<du var="orig" def="297" use="322" covered="0"/>
<du var="orig" def="297" use="323" covered="0"/>
<du var="orig" def="297" use="324" target="325" covered="0"/>
<du var="orig" def="297" use="324" target="326" covered="0"/>
<du var="orig" def="297" use="325" covered="0"/>
<du var="orig" def="297" use="315" target="315" covered="0"/>
<du var="orig" def="297" use="315" target="316" covered="0"/>
<du var="orig" def="297" use="316" covered="0"/>
<du var="orig" def="297" use="317" covered="0"/>
<du var="orig" def="297" use="318" covered="0"/>
<du var="orig" def="297" use="307" target="307" covered="0"/>
<du var="orig" def="297" use="307" target="308" covered="0"/>
<du var="orig" def="297" use="308" covered="0"/>
<du var="orig" def="297" use="309" covered="0"/>
<du var="orig" def="297" use="311" covered="0"/>
<du var="orig" def="297" use="300" covered="0"/>
<du var="autoOff" def="297" use="317" covered="0"/>
<du var="autoOff" def="297" use="309" covered="0"/>
<du var="d" def="322" use="326" covered="0"/>
<du var="d" def="322" use="325" covered="0"/>
<du var="proxy" def="335" use="336" target="336" covered="1"/>
<du var="proxy" def="335" use="336" target="336" covered="1"/>
<du var="proxy" def="335" use="336" covered="1"/>
<counter type="DU" missed="33" covered="15"/>
```

Figura 58: Data-flow coverage del metodo ProxyManagerImpl.newCustomProxy()

```

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 59: Analisi del Branch Coverage e Statement Coverage all'interno del metodo `ProxyManagerImpl.newCustomProxy()`

## ProxyManagerImpl

| Element  | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|--|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| toProxyableMapType(Class)  | 35%                 | 65%  | 37%             | 4    | 5      | 5    | 9      | 0     | 1      | 1       |
| toProxyableCollectionType(Class)   | 35%                 | 65%  | 37%             | 4    | 5      | 5    | 9      | 0     | 1      | 1       |
| toHelperParameters(Class[, Class])   | 0%                  | n/a  | 1               | 1    | 4      | 4    | 1      | 1     | 1      | 1       |
| toHelperAfterParameters(Class[, Class, Class])                                   | 0%                  | 0%   | 0%              | 7    | 7      | 15   | 15     | 1     | 1      | 1       |
| toConcreteType(Class, Map)   | 0%                  | 0%   | 0%              | 4    | 4      | 9    | 9      | 1     | 1      | 1       |
| static(...)  | 100%                | n/a  | 0               | 1    | 0      | 13   | 0      | 1     | 1      | 1       |
| startsWith(String, String)   | 100%                | 66%  | 2               | 4    | 0      | 3    | 0      | 1     | 1      | 1       |
| setUnproxyable(String)   | 100%                | 50%  | 1               | 2    | 0      | 3    | 0      | 1     | 1      | 1       |
| setTrackChanges(boolean)   | 0%                  | n/a  | 1               | 1    | 2      | 2    | 1      | 1     | 1      | 1       |
| setDelayCollectionLoading(boolean)   | 0%                  | n/a  | 1               | 1    | 2      | 2    | 1      | 1     | 1      | 1       |
| setAssetAllowedType(boolean)   | 0%                  | n/a  | 1               | 1    | 2      | 2    | 1      | 1     | 1      | 1       |
| proxySetters(BCClass, Class)   | 98%                 | 70%  | 3               | 6    | 0      | 9    | 0      | 1     | 1      | 1       |
| proxySetter(BCClass, Class, Method)  | 100%                | 100% | 0               | 2    | 0      | 16   | 0      | 1     | 1      | 1       |
| proxyRecognizedMethods(BCClass, Class, Class, Class)                             | 0%                  | 0%   | 6               | 6    | 32     | 32   | 1      | 1     | 1      | 1       |
| proxyOverrideMethod(BCClass, Method, Method, Class[])                            | 0%                  | 0%   | 2               | 2    | 12     | 12   | 1      | 1     | 1      | 1       |
| ProxyManagerImpl()   | 100%                | n/a  | 0               | 1    | 0      | 8    | 0      | 1     | 1      | 1       |
| proxyBeforeAfterMethod(BCClass, Class, Method, Method, Class[], Method, Class[]) | 0%                  | 0%   | 11              | 11   | 37     | 37   | 1      | 1     | 1      | 1       |
| nextProxyId()  | 100%                | n/a  | 0               | 1    | 0      | 1    | 0      | 1     | 0      | 1       |
| newMapProxy(Class, Class, Class, Comparator, boolean)                            | 85%                 | 50%  | 2               | 3    | 0      | 3    | 0      | 1     | 1      | 1       |
| newDateProxy(Class)  | 100%                | n/a  | 0               | 1    | 0      | 2    | 0      | 1     | 1      | 1       |
| newCustomProxy(Object, boolean)  | 98%                 | 95%  | 1               | 13   | 1      | 33   | 0      | 1     | 1      | 1       |

Figura 60: Statement Coverage e Branch Coverage del metodo `ProxyManagerImpl.newCustomProxy()` dopo la modifica dei casi di test

```

▼<method name="newCustomProxy" desc="(Ljava/lang/Object;Z)Lorg/apache/openjpa/util/Proxy;">
<du var="this" def="297" use="335" covered="1"/>
<du var="this" def="297" use="329" covered="1"/>
<du var="this" def="297" use="322" covered="1"/>
<du var="this" def="297" use="317" covered="1"/>
<du var="this" def="297" use="309" covered="1"/>
<du var="orig" def="297" use="297" target="298" covered="1"/>
<du var="orig" def="297" use="297" target="299" covered="1"/>
<du var="orig" def="297" use="299" target="300" covered="1"/>
<du var="orig" def="297" use="299" target="301" covered="1"/>
<du var="orig" def="297" use="301" target="302" covered="0"/>
<du var="orig" def="297" use="301" target="303" covered="1"/>
<du var="orig" def="297" use="303" target="304" covered="1"/>
<du var="orig" def="297" use="303" target="306" covered="1"/>
<du var="orig" def="297" use="306" target="307" covered="1"/>
<du var="orig" def="297" use="306" target="314" covered="1"/>
<du var="orig" def="297" use="314" target="315" covered="1"/>
<du var="orig" def="297" use="314" target="321" covered="1"/>
<du var="orig" def="297" use="321" target="322" covered="1"/>
<du var="orig" def="297" use="321" target="328" covered="1"/>
<du var="orig" def="297" use="328" target="329" covered="1"/>
<du var="orig" def="297" use="328" target="335" covered="1"/>
<du var="orig" def="297" use="335" covered="1"/>
<du var="orig" def="297" use="336" covered="1"/>
<du var="orig" def="297" use="329" covered="1"/>
<du var="orig" def="297" use="331" covered="1"/>
<du var="orig" def="297" use="322" covered="1"/>
<du var="orig" def="297" use="323" covered="1"/>
<du var="orig" def="297" use="324" target="325" covered="1"/>
<du var="orig" def="297" use="324" target="326" covered="1"/>
<du var="orig" def="297" use="325" covered="1"/>
<du var="orig" def="297" use="315" target="315" covered="1"/>
<du var="orig" def="297" use="315" target="316" covered="1"/>
<du var="orig" def="297" use="316" covered="1"/>
<du var="orig" def="297" use="317" covered="1"/>
<du var="orig" def="297" use="318" covered="1"/>
<du var="orig" def="297" use="307" target="307" covered="1"/>
<du var="orig" def="297" use="307" target="308" covered="1"/>
<du var="orig" def="297" use="308" covered="1"/>
<du var="orig" def="297" use="309" covered="1"/>
<du var="orig" def="297" use="311" covered="1"/>
<du var="orig" def="297" use="300" covered="1"/>
<du var="autoOff" def="297" use="317" covered="1"/>
<du var="autoOff" def="297" use="309" covered="1"/>
<du var="d" def="322" use="326" covered="1"/>
<du var="d" def="322" use="325" covered="1"/>
<du var="proxy" def="335" use="336" target="336" covered="1"/>
<du var="proxy" def="335" use="336" target="336" covered="1"/>
<du var="proxy" def="335" use="336" covered="1"/>
<counter type="DU" missed="1" covered="47"/>
```

Figura 61: Data-flow coverage del metodo `ProxyManagerImpl.newCustomProxy()` dopo la modifica dei casi di test

```

294     @Override
295     public Proxy newCustomProxy(Object orig, boolean autoOff) {
296         if (orig == null)
297             return null;
298         if (orig instanceof Proxy)
299             return (Proxy) orig;
300         if (ImplHelper.isManageable(orig))
301             return null;
302         if (!isProxyable(orig.getClass()))
303             return null;
304
305         if (orig instanceof Collection) {
306             Comparator comp = (orig instanceof SortedSet)
307                 ? ((SortedSet) orig).comparator() : null;
308             Collection c = (Collection) newCollectionProxy(orig.getClass(),
309                 null, comp, autoOff);
310             c.addAll((Collection) orig);
311             return (Proxy) c;
312         }
313         if (orig instanceof Map) {
314             Comparator comp = (orig instanceof SortedMap)
315                 ? ((SortedMap) orig).comparator() : null;
316             Map m = (Map) newMapProxy(orig.getClass(), null, null, comp, autoOff);
317             m.putAll((Map) orig);
318             return (Proxy) m;
319         }
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 }
```

Figura 62: Risultati del Mutation Testing sul metodo `ProxyManagerImpl.newCustomProxy()`

## ProxyManagerImpl

| Element  | Missed Instructions | Cov. | Missed Branches | Cov. | Missed | Cxty | Missed | Lines | Missed | Methods |
|--|---------------------|------|-----------------|------|--------|------|--------|-------|--------|---------|
| addProxyBeanMethods(BCClass, Class, Constructor) | ██████              | 80%  | ██████          | 50%  | 4      | 5    | 4      | 30    | 0      | 1       |
| addProxyCalendarMethods(BCClass, Class)          | ██████████          | 0%   | ████            | 0%   | 4      | 4    | 62     | 62    | 1      | 1       |
| addProxyCollectionMethods(BCClass, Class)        | ██████████          | 0%   | ██████          | 0%   | 10     | 10   | 77     | 77    | 1      | 1       |
| addProxyDateMethods(BCClass, Class)              | ██████████          | 0%   | ██████          | 0%   | 14     | 14   | 59     | 59    | 1      | 1       |
| addProxyMapMethods(BCClass, Class)               | ██████████          | 0%   | ██████          | 0%   | 10     | 10   | 87     | 87    | 1      | 1       |
| addProxyMethods(BCClass, boolean)                | ██████              | 100% | ████            | 50%  | 1      | 2    | 0      | 57    | 0      | 1       |
| addWriteReplaceMethod(BCClass, boolean)          | ████                | 96%  | ████            | 50%  | 1      | 2    | 0      | 11    | 0      | 1       |
| allowsDuplicates(Class)                          | ████                | 0%   | ████            | 0%   | 2      | 2    | 1      | 1     | 1      | 1       |
| assertNotFinal(Class)                            | ████                | 0%   | ████            | 0%   | 2      | 2    | 3      | 3     | 1      | 1       |
| copyArray(Object)                                | ████                | 100% | ████            | 100% | 0      | 2    | 0      | 10    | 0      | 1       |

Figura 63: Statement Coverage e Branch Coverage del metodo `ProxyManagerImpl.copyArray()`

```

▼ <method name="copyArray" desc="(Ljava/lang/Object;)Ljava/lang/Object;">
    <du var="orig" def="180" use="180" target="181" covered="1"/>
    <du var="orig" def="180" use="180" target="184" covered="1"/>
    <du var="orig" def="180" use="184" covered="1"/>
    <du var="orig" def="180" use="185" covered="1"/>
    <du var="orig" def="180" use="188" covered="1"/>
    <counter type="DU" missed="0" covered="5"/>
    <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figura 64: Data-flow coverage del metodo `ProxyManagerImpl.copyArray()`

```

168 /**
169 * Provided for auto-configuration. Add the given semicolon-separated
170 * class names to the set of class names we know cannot be proxied correctly
171 * by this manager.
172 */
173 public void setUnproxyable(String clsNames) {
174     1 if (clsNames != null)
175         _unproxyable.addAll(Arrays.asList(StringUtil.split(clsNames, ";", 0)));
176     }
177
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 UnsupportedOperationException(_loc.get("bad-array",
192             e.getMessage()), e);
193     }
194 }

```

*Figura 65: Risultati del Mutation Testing sul metodo ProcyManagerImpl.copyArray()*