

Software Testing Report: Apache BookKeeper & OpenJPA

Gaia Meola

2 febbraio 2026

Indice

1 Introduzione	5
2 Configurazione dell'Ambiente e Continuous Integration	5
2.1 Repository e Branching Strategy	5
2.2 Pipeline di CI (GitHub Actions)	5
3 Metodologia di Generazione e Analisi della Test Suite	5
4 Progettazione e Architettura dei Test Manuali	5
4.1 Strategia di Identificazione dei Casi di Test	5
4.2 Architettura Tecnica e Implementazione	5
4.3 Classe BufferedChannel (BookKeeper)	5
4.3.1 Collocazione dei sorgenti	5
4.3.2 Descrizione del SUT	5
4.3.3 Descrizione degli attributi	6
4.3.4 Progettazione Manuale	6
4.4 Classe WriteCache (BookKeeper)	6
4.4.1 Collocazione dei sorgenti	6
4.4.2 Descrizione del SUT	6
4.4.3 Descrizione degli attributi	6
4.4.4 Progettazione Manuale	6
4.5 Classe ClassUtil (OpenJPA)	6
4.5.1 Collocazione dei sorgenti	6
4.5.2 Descrizione del SUT	6
4.5.3 Progettazione Manuale	6
4.6 Classe CacheMap (OpenJPA)	7
4.6.1 Collocazione dei sorgenti	7
4.6.2 Descrizione del SUT	7
4.6.3 Descrizione degli attributi	7
4.6.4 Progettazione Manuale	7
4.7 Generazione automatica (Bookkeeper - Buffered Channel)	7
4.7.1 Generazione assistita tramite LLM: strategia di prompting Zero-Shot	7
4.7.2 Generazione automatica guidata da criteri di <i>control-flow coverage</i> : EvoSuite	7
4.7.3 Generazione automatica guidata da criteri stocastici: Randoop	7
5 Analisi Comparativa: Paradigmi di Testing a Confronto	8
6 Discussione dei Risultati	8
7 Analisi Qualitativa: Confronto Metodologico e Filosofie di Verifica	8
8 Definizione e risultati dei test manuali	9
8.1 Definizione e risultati dei test di BufferedChannel	9
8.1.1 Statement Coverage e Branch Coverage (JaCoCo)	9
8.1.2 Mutation Testing (PIT)	10
8.2 Definizione e risultati dei test di WriteCache	11
8.2.1 Statement Coverage e Branch Coverage (JaCoCo)	11
8.2.2 Mutation Testing (PIT)	11
8.3 Definizione e risultati dei test di ClassUtil	12
8.3.1 Statement Coverage e Branch Coverage (JaCoCo)	12
8.3.2 Mutation Testing (PIT)	13

8.4	Definizione e risultati dei test di CacheMap	13
8.4.1	Statement Coverage e Branch Coverage (JaCoCo)	13
8.4.2	Mutation Testing (PIT)	14
9	Integration Testing	15
10	Stima della Reliability	15
10.1	Metodologia di Calcolo	15
10.2	Analisi per Classe e Metodo	15
10.2.1	Classe BufferedChannel	15
10.2.2	Classe WriteCache	16
10.2.3	Classe ClassUtil	16
10.2.4	Classe CacheMap	16
A	Appendice: Listati di Codice e Tabelle	17
A.1	Configurazione Workflow GitHub Actions - BookKeeper	17
A.2	Template (Zero-shot) - Bookkeeper	17
A.3	Dettaglio degli attributi della classe	18
A.4	Materiale Classe BufferedChannel	19
A.4.1	Inizializzazione del componente (Costruttore)	19
A.4.2	Identificazione della classi di equivalenza	19
A.4.3	Boundary Value Analysis	19
A.4.4	Casi di Test - Costruttore	20
A.4.5	Risultati dei test e analisi dei fallimenti	20
A.4.6	Correzioni a seguito dell'analisi dei fallimenti	21
A.4.7	Analisi del metodo write	22
A.4.8	Identificazione della classi di equivalenza	22
A.4.9	Boundary Value Analysis	22
A.4.10	Casi di Test - Metodo Write	22
A.4.11	Risultati dei test e analisi dei fallimenti	23
A.4.12	Correzioni a seguito dell'analisi dei fallimenti	23
A.4.13	Analisi del metodo read	25
A.4.14	Identificazione della classi di equivalenza	25
A.4.15	Boundary Value Analysis	25
A.4.16	Casi di Test - Metodo Read	26
A.4.17	Risultati dei test e analisi dei fallimenti	27
A.4.18	Correzioni a seguito dell'analisi dei fallimenti	27
A.5	Materiale Classe WriteCache	28
A.5.1	Inizializzazione del componente (Costruttore)	28
A.5.2	Identificazione della classi di equivalenza	28
A.5.3	Boundary Value Analysis	28
A.5.4	Casi di Test - Costruttore	28
A.5.5	Risultati dei test e analisi dei fallimenti	28
A.5.6	Correzioni a seguito dell'analisi dei fallimenti	29
A.5.7	Analisi del metodo put	30
A.5.8	Identificazione della classi di equivalenza	30
A.5.9	Boundary Value Analysis	30
A.5.10	Casi di Test - Metodo put	31
A.5.11	Risultati dei test e analisi dei fallimenti	31
A.5.12	Correzioni a seguito dell'analisi dei fallimenti	31
A.5.13	Analisi del metodo get	32
A.5.14	Identificazione della classi di equivalenza	32
A.5.15	Boundary Value Analysis	32
A.5.16	Casi di Test - Metodo get	32
A.5.17	Risultati dei test e analisi dei fallimenti	33
A.5.18	Correzioni a seguito dell'analisi dei fallimenti	33
A.6	Materiale Classe ClassUtil	34
A.6.1	Analisi del metodo getClassName	34
A.6.2	Identificazione della classe di equivalenza	34
A.6.3	Boundary Value Analysis	34
A.6.4	Casi di Test - Metodo getClassName	34
A.6.5	Analisi del metodo getPackageName	34
A.6.6	Identificazione della classe di equivalenza	34
A.6.7	Boundary Value Analysis	34

A.6.8	Casi di Test - Metodo getPackageName	35
A.6.9	Analisi del metodo toClass	35
A.6.10	Identificazione della classe di equivalenza	35
A.6.11	Boundary Value Analysis	35
A.6.12	Casi di Test - Metodo toClass	35
A.6.13	Analisi del metodo getClassName	36
A.6.14	Identificazione della classe di equivalenza	36
A.6.15	Boundary Value Analysis	36
A.6.16	Casi di Test - Metodo getClassName (String)	36
A.6.17	Risultati dei test e analisi dei fallimenti	36
A.6.18	Correzioni a seguito dell'analisi dei fallimenti	36
A.6.19	Analisi del metodo getPackageName	37
A.6.20	Identificazione della classe di equivalenza	37
A.6.21	Analisi boundary	37
A.6.22	Casi di Test - Metodo getPackageName (String)	37
A.6.23	Risultati dei test e analisi dei fallimenti	37
A.6.24	Correzioni a seguito dell'analisi dei fallimenti	38
A.7	Materiale Classe CacheMap	39
A.7.1	Inizializzazione del componente (Costruttore)	39
A.7.2	Identificazione della classi di equivalenza	39
A.7.3	Boundary Value Analysis	39
A.7.4	Casi di Test - Costruttore CacheMap	39
A.7.5	Risultati dei test e analisi dei fallimenti	40
A.7.6	Correzioni a seguito dell'analisi dei fallimenti	40
A.7.7	Analisi del metodo pin	41
A.7.8	Identificazione della classe di equivalenza	41
A.7.9	Boundary Value Analysis	41
A.7.10	Casi di Test - Metodo pin	41
A.7.11	Risultati dei test e analisi dei fallimenti	41
A.7.12	Correzioni a seguito dell'analisi dei fallimenti	41
A.7.13	Analisi del metodo unpin	42
A.7.14	Identificazione della classe di equivalenza	42
A.7.15	Boundary Value Analysis	42
A.7.16	Casi di Test - Metodo unpin	42
A.7.17	Analisi del metodo put	43
A.7.18	Identificazione della classe di equivalenza	43
A.7.19	Boundary Value Analysis	43
A.7.20	Casi di Test - Metodo put	43
A.7.21	Risultati dei test e analisi dei fallimenti	44
A.7.22	Correzioni a seguito dell'analisi dei fallimenti	44
A.7.23	Analisi del metodo get	45
A.7.24	Identificazione della classe di equivalenza	45
A.7.25	Boundary Value Analysis	45
A.7.26	Casi di Test - Metodo get	45
A.7.27	Risultati dei test e analisi dei fallimenti	45
A.7.28	Correzioni a seguito dell'analisi dei fallimenti	45
A.8	Comandi per la Generazione Automatica	46
A.9	Configurazione e Comando Randoop	46
A.10	Risultati dei test di copertura	47
A.10.1	Risultati iniziali della copertura di BufferedChannel	47
A.10.2	Risultati finali della copertura di BufferedChannel	47
A.10.3	Risultati iniziali della copertura di WriteCache	47
A.10.4	Casi di Test Raffinati - Metodo put (WriteCache)	48
A.10.5	Risultati finali della copertura di WriteCache	48
A.10.6	Risultati iniziali della copertura di CacheMap	49
A.10.7	Risultati iniziali della copertura di ClassUtil	49
A.10.8	Identificazione della classe di equivalenza (Raffinata)	49
A.10.9	Boundary Value Analysis (Raffinata)	50
A.10.10	Casi di Test Raffinati - Metodo getClassName (String)	50
A.10.11	Identificazione della classe di equivalenza (Raffinata)	50
A.10.12	Analisi boundary (Raffinata)	51
A.10.13	Casi di Test Raffinati - Metodo getPackageName (String)	51
A.10.14	Identificazione della classe di equivalenza (Raffinata)	51
A.10.15	Analisi boundary (Raffinata)	52

A.10.16	Casi di Test Raffinati - Metodo toClass (ClassUtil)	52
A.10.17	Risultati finali della copertura di ClassUtil	53
B	Report PIT	54
B.1	Analisi BufferedChannel	54
B.2	Analisi WriteCache	54
B.3	Analisi ClassUtil	56
B.4	Analisi CacheMap	58
C	Integration Testing	60

1 Introduzione

Il presente report documenta le attività di verifica e convalida condotte sui progetti open-source **BookKeeper** e **OpenJPA** della *Apache Software Foundation*. L’analisi si è focalizzata sulle classi `BufferedChannel` e `WriteCache` per BookKeeper, e `ClassUtil` e `CacheMap` per OpenJPA. La metodologia adottata segue un approccio incrementale che integra la progettazione manuale black-box, la generazione automatica (coverage-guided e random) e la generazione assistita tramite Large Language Models. La qualità della test suite è stata validata mediante metriche di adeguatezza e robustezza automatizzando l’intero processo in una pipeline di **Continuous Integration** su **GitHub Actions**.

2 Configurazione dell’Ambiente e Continuous Integration

2.1 Repository e Branching Strategy

L’attività è iniziata con il *forking* dei repository ufficiali Apache su GitHub. Per garantire l’isolamento degli artefatti di testing e non interferire con il codice sorgente originale, è stata adottata una strategia di branching dedicata operando sul branch `test_isw2`.

2.2 Pipeline di CI (GitHub Actions)

Il processo di build e test è stato automatizzato tramite **GitHub Actions**. La configurazione completa del workflow YAML è consultabile nell’Appendice A.1.

3 Metodologia di Generazione e Analisi della Test Suite

Per ogni classe target è stata generata una suite di test eterogenea per confrontare diversi paradigmi di testing: test manuali (basati su *Category Partition* e ottimizzati per *mutation score*), test basati su **LLM** (*Zero-Shot Prompting*) e test automatici tramite **EvoSuite** (*coverage-guided*) e **Randoop** (*feedback-directed random*). Al fine di coniugare sintesi e profondità d’analisi, la discussione comparativa dei risultati si focalizza sulla classe `BufferedChannel`. Tale modulo è stato selezionato come caso di studio rappresentativo, in quanto le evidenze sperimentali e le metriche qualitative riscontrate riflettono fedelmente le dinamiche osservate nell’intero set di classi analizzate.

4 Progettazione e Architettura dei Test Manuali

In questa sezione viene descritto l’approccio metodologico e tecnico adottato per la generazione dei test manuali, applicato uniformemente alle quattro classi in esame al fine di garantirne la correttezza funzionale, l’adeguatezza e la manutenibilità.

4.1 Strategia di Identificazione dei Casi di Test

La strategia di testing adotta un approccio metodologico ibrido che coniuga l’analisi funzionale (*black-box*) con quella strutturale (*white-box*), quest’ultima essenziale per sopportare alla scarsità di documentazione e identificare i flussi logici interni. La progettazione dei test si è basata sulla *Category Partition*, articolata in tre livelli: analisi sintattica dei domini di input, analisi semantica del contesto operativo e modellazione dello stato interno del *SUT*. A causa dell’impossibilità di interagire con gli stakeholder, eventuali discrepanze tra specifiche attese e codice sono state risolte tramite una revisione iterativa delle classi di equivalenza basata sull’analisi diretta del software. Tale scelta, sebbene non ortodossa in un workflow standard, si è resa necessaria per ricostruire le specifiche *de facto* e distinguere i reali difetti dalle peculiarità implementative dei componenti.

4.2 Architettura Tecnica e Implementazione

La suite di test è stata implementata seguendo un pattern basato su **JUnit 5**, volto a massimizzare il riuso del codice. Il cuore dell’architettura sfrutta il *Data-driven Testing* tramite `@ParameterizedTest` e `@MethodSource`, separando la logica dai dati per gestire efficacemente ampie partizioni di input. La coerenza del sistema è garantita dal package `customutils`, la cui classe `Utils` centralizza la generazione di input complessi. Per l’ottimizzazione delle risorse e il ripristino dello stato, sono stati utilizzati rispettivamente `@TestInstance(Lifecycle.PER_CLASS)` e `@AfterEach`. Infine, la robustezza del software è stata validata mediante l’uso estensivo di `assertThrows`.

4.3 Classe `BufferedChannel` (BookKeeper)

4.3.1 Collocazione dei sorgenti

La classe di test sviluppata è reperibile al seguente percorso:

https://github.com/GaiaMeola/bookkeeper/tree/test_isw2/bookkeeper-server/src/test/java/manualtest

4.3.2 Descrizione del SUT

Sebbene la documentazione ufficiale¹ si focalizzi sull’interazione ad alto livello tra i nodi, attraverso l’analisi combinata del codice sorgente e della relativa documentazione Javadoc è stato possibile identificare `BufferedChannel` come un componente critico dello *storage layer*. La classe estende le funzionalità di `FileChannel` agendo come *wrapper* specializzato per l’accesso ottimizzato alle *entry logs*.

¹<https://bookkeeper.apache.org/docs/4.10.0/development/protocol/>

4.3.3 Descrizione degli attributi

Gli attributi fondamentali per la definizione dei vincoli di stato e per l'applicazione della *Category Partition* sono descritti nella Tabella 10 in Appendice.

4.3.4 Progettazione Manuale

La strategia di testing per la classe `BufferedChannel` è stata sviluppata per garantire l'integrità del flusso dati tra memoria volatile e supporto fisico, focalizzandosi sulla robustezza del sistema in stati inconsistenti del file system. La progettazione ha isolato la logica di buffering per assicurare trasferimenti privi di corruzione, analizzando in primo luogo il costruttore per verificare la stabilità del legame con il `FileChannel` e la corretta allocazione dei buffer di I/O. Successivamente, l'analisi ha convalidato la gestione dei dati in uscita, ponendo particolare attenzione alla sincronizzazione tra memoria e disco durante le operazioni di scrittura. Parallelamente, per la fase di lettura, è stata verificata l'efficienza nel recupero dei dati. Tutto il materiale relativo alla Category Partition, alla Boundary Value Analysis e la definizione integrale dei casi di test per `BufferedChannel` è consultabile a partire dalla sezione A.4 in Appendice.

4.4 Classe WriteCache (BookKeeper)

4.4.1 Collocazione dei sorgenti

La classe di test sviluppata è reperibile al seguente percorso:

https://github.com/GaiaMeola/bookkeeper/tree/test_isw2/bookkeeper-server/src/test/java/manualtest

4.4.2 Descrizione del SUT

Come indicato nella documentazione tecnica di Apache BookKeeper², la classe `WriteCache` funge da memoria intermedia ad alte prestazioni per le operazioni di scrittura. Attraverso l'impiego di un *buffer condiviso* e di una `lastEntryMap` atomica, la classe ottimizza l'indicizzazione e l'accesso ai dati, riducendo la latenza prima della fase di persistenza fisica.

4.4.3 Descrizione degli attributi

Gli attributi fondamentali per la definizione dei vincoli di stato e per l'applicazione della *Category Partition* sono descritti nella Tabella 11 in Appendice.

4.4.4 Progettazione Manuale

La strategia di testing per la classe `WriteCache` è stata definita per validare la gestione della memoria personalizzata e la persistenza dei dati, focalizzandosi sull'integrità delle *entry* allocate in *direct memory*. Il testing ha esaminato la robustezza del sistema durante la saturazione dei segmenti, accertando la corretta gestione del ciclo di vita della memoria e la stabilità dell'infrastruttura a fronte di parametri di input invalidi. Tutto il materiale relativo alla Category Partition, alla Boundary Value Analysis e la definizione integrale dei casi di test per `WriteCache` è consultabile a partire dalla sezione A.5 in Appendice.

4.5 Classe ClassUtil (OpenJPA)

4.5.1 Collocazione dei sorgenti

La classe di test sviluppata è reperibile al seguente percorso:

https://github.com/GaiaMeola/openjpa/tree/test_isw2/openjpa-lib/src/test/java/manualtest

4.5.2 Descrizione del SUT

Come riportato nella documentazione ufficiale di Apache OpenJPA³, la classe `ClassUtil` funge da componente infrastrutturale critico per la gestione del *runtime* Java, astraendo le complessità legate alla manipolazione dinamica dei tipi. Il suo ruolo principale consiste nel fornire un'interfaccia semplificata per il caricamento dinamico delle risorse e l'estrazione di metadati strutturali. Inoltre, la classe include logiche specializzate per la gestione di classi proxy e *wrapper*, garantendo che il sistema possa distinguere o convertire correttamente tra tipi primitivi e le rispettive controparti a oggetti.

4.5.3 Progettazione Manuale

Data la presenza di un costruttore privato che ne impedisce l'istanziazione, la strategia di testing si è focalizzata esclusivamente sulla validazione dei metodi statici, accertando la correttezza del contratto della classe in scenari critici per il framework. L'analisi è stata orientata alla verifica del caricamento di tipi primitivi e array tramite *Class Loader*, garantendo l'affidabilità delle operazioni di risoluzione dei nomi e di mappatura dei tipi senza la necessità di creare istanze

²<https://bookkeeper.apache.org/docs/4.18.0/development/protocol/>

³<https://openjpa.apache.org/documentation.html>

dell'oggetto. Tutto il materiale relativo alla Category Partition, alla Boundary Value Analysis e la definizione integrale dei casi di test è consultabile a partire dalla sezione A.6 in Appendice.

4.6 Classe CacheMap (OpenJPA)

4.6.1 Collocazione dei sorgenti

La classe di test sviluppata è reperibile al seguente percorso:

https://github.com/GaiaMeola/openjpa/tree/test_isw2/openjpa-kernel/src/test/java/manualtest

4.6.2 Descrizione del SUT

Come indicato nella documentazione ufficiale⁴, la classe CacheMap estende l'interfaccia Map per implementare una struttura dati a dimensione fissa, specializzata nella gestione efficiente di un set limitato di risorse. La sua caratteristica distintiva è il supporto al *pinning* delle entry, un meccanismo di blocco e sblocco che permette di sottrarre determinati elementi alle politiche di rimpiazzamento della cache. Tale funzionalità garantisce che i dati critici rimangano residenti in memoria durante le operazioni ad alta priorità, prevenendone l'evitazione prematura e assicurando la coerenza del sistema in scenari di carico elevato o contesa di risorse.

4.6.3 Descrizione degli attributi

Gli attributi fondamentali per la definizione dei vincoli di stato e per l'applicazione della *Category Partition* sono descritti nella 12 in Appendice.

4.6.4 Progettazione Manuale

La strategia di testing per la classe CacheMap è stata definita per validare il ciclo di vita delle entry e le relative transizioni tra i livelli di memoria interna, operando in uno scenario single-thread per garantire il determinismo dei test ed evitare l'imprevedibilità delle *race condition*. La progettazione ha verificato inizialmente la corretta configurazione del costruttore e l'allocazione delle risorse, focalizzandosi successivamente sulle dinamiche di inserimento e recupero. In questa fase, sono stati validati i meccanismi di *eviction* e promozione, accertando che al riempimento della cache primaria le entry vengano migrate nella softMap e correttamente ripristinate in caso di accesso successivo. Parallelamente, il testing del protocollo di *pinning* ha confermato la capacità della classe di bloccare chiavi specifiche nella memoria primaria, garantendo che gli elementi marcati risultino immuni ai processi di espulsione automatica fino alla loro esplicita rimozione. Tutto il materiale relativo alla Category Partition, alla Boundary Value Analysis e la definizione integrale dei casi di test è consultabile a partire dalla sezione A.7 in Appendice.

4.7 Generazione automatica (Bookkeeper - Buffered Channel)

4.7.1 Generazione assistita tramite LLM: strategia di prompting Zero-Shot

La generazione dei test è stata affidata a **Gemini 1.5 Flash** tramite una strategia di *Zero-Shot Prompting*. La scelta deliberata di non fornire esempi (*few-shot*) risponde alla necessità di valutare le capacità analitiche "pure" del modello, minimizzando i condizionamenti esterni e garantendo l'oggettività nel confronto con la progettazione manuale. Tale approccio ha permesso di osservare l'interpretazione autonoma del codice e delle tabelle di *Equivalence Partitioning*, assicurando che i test riflettano la logica interna dell'LLM anziché un'imitazione di pattern predefiniti. Per garantire la riproducibilità, è stato impiegato un **Template di Prompt standardizzato** (Appendice A.2), adattato ai diversi metodi mediante l'aggiornamento della sola documentazione tecnica di supporto.

4.7.2 Generazione automatica guidata da criteri di control-flow coverage: EvoSuite

L'impiego di **EvoSuite** ha richiesto il superamento di criticità strutturali legate all'ambiente di esecuzione. A causa dell'incompatibilità del plugin Maven con Java 11 (dovuta alla rimozione di tools.jar), si è optato per un *downgrade* controllato a Java 8, utilizzando la versione *standalone* (v1.1.0) per un controllo granulare su classpath e librerie. L'integrazione ha imposto l'inclusione manuale di evosuite-runtime tramite system scope e un *refactoring* dei sorgenti generati per uniformarne la gerarchia dei package. Le fasi di *troubleshooting* hanno riguardato la disabilitazione della *Sandbox* e dello state *resetting* per risolvere conflitti con gli inizializzatori nativi di Netty, oltre alla disattivazione del *separateClassLoader* per consentire la corretta strumentazione da parte di **JaCoCo**⁵.

4.7.3 Generazione automatica guidata da criteri stocastici: Randoop

L'impiego di **Randoop** ha presentato sfide legate alle dipendenze di basso livello della classe *BufferedChannel*, come *FileChannel* e i buffer di *Netty*. L'approccio stocastico iniziale ha prodotto una copertura esigua a causa dell'incapacità dello strumento di sintetizzare autonomamente parametri complessi, esitando in frequenti *NullPointerException*. Per

⁴<https://openjpa.apache.org/documentation.html>

⁵Al contrario, non è stato possibile estendere tale approccio ad **OpenJPA**: i vincoli di compatibilità del progetto verso versioni di Java superiori alla 8 hanno precluso l'ipotesi di un *downgrade* del runtime, rendendo l'ambiente di esecuzione di EvoSuite tecnicamente incompatibile con i requisiti minimi del sistema e limitando la generazione automatica ai soli moduli di BookKeeper.

colmare tale *Mocking Deficiency*, è stata adottata una strategia di *Guided Test Generation* tramite lo sviluppo di un `RandoopHelper`; questa classe *Factory* ha fornito istanze pre-configure di `FileChannel` e `ByteBufAllocator`, permettendo la composizione di sequenze semanticamente corrette. Parallelamente, per superare i limiti dei plugin Maven nella gestione di dipendenze profonde, l'esecuzione è stata migrata su interfaccia a riga di comando (CLI, dettagli in Appendice A.9). Tale configurazione ha garantito un controllo deterministico sul *classpath*, stabilizzando il processo e consentendo il raggiungimento di livelli di copertura significativi.

5 Analisi Comparativa: Paradigmi di Testing a Confronto

La valutazione quantitativa della suite di test per la classe `BufferedChannel` si focalizza sulle metriche di copertura ottenute tramite **JaCoCo**. Vengono messi a confronto i risultati della progettazione manuale, della generazione tramite **LLM** e dell'approccio *automated* (**EvoSuite** e **Randoop**), fornendo una panoramica oggettiva sulla capacità di ogni metodologia di esplorare il dominio del *System Under Test*.

Metodologia	Costruttore		Metodo <code>read()</code>		Metodo <code>write()</code>	
	Instr.	Branch	Instr.	Branch	Instr.	Branch
Manuale	100	100	99	90	97	80
LLM (AI-Generated)	100	100	68	60	100	100
EvoSuite	100	100	39	55	100	100
Randoop	100	100	37	45	53	50

Tabella 1: Analisi comparativa della copertura del codice (%)

6 Discussione dei Risultati

L'analisi comparativa dei dati di copertura (Tab. 1) evidenzia come la natura del `BufferedChannel`, caratterizzata dalla stretta interdipendenza tra gestione della memoria (*Netty*) e *file system*, funga da discriminante tra approcci strutturali e semantici. Emerge una rilevante asimmetria nelle performance di **EvoSuite**: a fronte di una saturazione completa del metodo `write()` (100%), lo strumento mostra limiti strutturali nel metodo `read()` (39%). Tale discrepanza è riconducibile alla natura evolutiva dell'algoritmo genetico, il quale incorre in un *Plateau nella Fitness* quando affronta operazioni *state-dependent*. La probabilità di generare spontaneamente una sequenza orchestrata (*Write → Flush → Read*) con parametri coerenti è estremamente bassa; senza tale coordinazione, i test non superano le clausole di guardia del `FileChannel`, impedendo all'algoritmo di evolvere. In questo contesto, **Randoop** ha mostrato le criticità maggiori, con una copertura del 37% nella `read()` e del 53% nella `write()`. Nonostante l'ausilio del `RandoopHelper`, la generazione casuale guidata dal feedback soffre drasticamente la complessità del dominio: senza una comprensione della gerarchia delle chiamate, lo strumento produce sequenze valide sotto il profilo sintattico ma spesso prive di senso operativo, disperpendo lo sforzo computazionale in esplorazioni superficiali che non raggiungono i rami logici più profondi. Al contrario, i test generati tramite **LLM** rappresentano l'elemento di rottura, superando gli strumenti automatici nella `read()` (68%) grazie a una superiore "comprensione contestuale". L'AI, interpretando la finalità del codice anziché esplorare ciecamente il grafo delle decisioni, ha intuito la necessità di pre-condizionare lo stato del canale, superando i blocchi logici che hanno arrestato **EvoSuite** e **Randoop**. Tuttavia, la necessità di interventi manuali per correggere lievi inconsistenze sintattiche suggerisce un'evoluzione del testing verso un modello di co-progettazione assistita. In definitiva, se l'automazione risulta ottimale per componenti *stateless*, la sintesi di protocolli di interazione complessi richiede ancora l'integrazione di modelli semantici o l'apporto euristico del testing manuale.

7 Analisi Qualitativa: Confronto Metodologico e Filosofie di Verifica

L'indagine sulle metodologie adottate evidenzia un dualismo fondamentale tra la produttività degli strumenti automatici e la robustezza del design manuale, tracciando un bilancio che trascende il puro dato numerico a favore della sostenibilità della suite di test. L'impiego di **Gemini 3 Flash** ha rivelato una dicotomia tra l'abbattimento delle barriere tecniche iniziali e i "costi occulti" di raffinamento; sebbene il modello mappi rapidamente le classi di equivalenza in strutture JUnit coerenti e leggibili, caratterizzate da annotazioni descrittive come `@DisplayName`, l'inerzia nella correzione degli errori logici può consumare il tempo risparmiato in complesse sessioni di debugging. Tale approccio, prettamente orientato al *mocking* sistematico tramite **Mockito**, garantisce isolamento ma soffre di una certa rigidità strutturale, producendo test atomici con frequenti duplicazioni del setup che limitano la manutenibilità rispetto alla progettazione manuale parametrizzata. In netto contrasto, l'analisi di **Randoop** evidenzia i limiti della generazione stocastica pura: i test prodotti risultano essere sequenze di regressione che, pur sfruttando il feedback a runtime, appaiono frammentate e difficilmente interpretabili semanticamente. Senza l'intervento del `RandoopHelper` per fornire istanze valide, lo strumento si limita a generare test di esplorazione negativa che saturano esclusivamente i rami legati alle eccezioni, fallendo nel catturare la logica di business. Parallelamente, **EvoSuite** adotta una filosofia intermedia, producendo codice che, pur essendo tecnicamente sofisticato e capace di manipolare lo stato interno, risulta spesso criptico e affetto da *overfitting* sul codice sorgente. Le criticità di integrazione emerse, come i conflitti tra `ClassLoader` e le violazioni di sicurezza della `Sandbox`, sottolineano come l'automazione strutturale possa generare test fragili in ambienti ad alta intensità di risorse come *Netty*. In definitiva, mentre l'approccio manuale garantisce una fedeltà superiore tramite l'uso di istanze reali

(UnpooledByteBufAllocator), l’automazione si conferma un modulo di *augmentation* prezioso ma non autosufficiente. La diversità stilistica tra i test, dalla semantica discorsiva dell’LLM, alla concisione tecnica di EvoSuite, fino alla verbosità meccanica di Randoop, dimostra che la sintesi di protocolli complessi richiede ancora una governance umana centralizzata. La progettazione dello scheletro logico deve rimanere una prerogativa del tester, relegando le macchine al ruolo di acceleratori per la copertura dei casi limite e l’ottimizzazione dei rami logici meno densi di significato funzionale.

8 Definizione e risultati dei test manuali

Al fine di valutare l’adeguatezza della suite di test progettata è stata effettuata un’analisi della Code Coverage utilizzando lo strumento **JaCoCo**. Nello specifico, l’attenzione è stata posta su due indicatori fondamentali: la **statement coverage**, che misura la percentuale di singole istruzioni eseguite almeno una volta, e la **branch coverage**, che analizza tutti i possibili percorsi decisionali all’interno delle strutture di controllo.

8.1 Definizione e risultati dei test di **BufferedChannel**

8.1.1 Statement Coverage e Branch Coverage (JaCoCo)

I risultati riassunti nella Tabella 96 hanno permesso di identificare i rami logici non sollecitati, guidando l’integrazione di nuovi casi di test e il perfezionamento di quelli esistenti. Con riferimento al metodo `write`, le lacune di copertura principali sono state riscontrate alle **righe 131 e 135**: nello specifico, la condizione `if(doRegularFlushes)` risultava incompleta, avendo assunto esclusivamente il valore `true`. Tale mancanza non è derivata da un’errata interpretazione della logica del metodo, i cui meccanismi erano stati correttamente ipotizzati in fase di analisi, quanto piuttosto dai limiti della strategia di *black-box testing*. In tale fase, l’attenzione è stata focalizzata sulla validazione dei parametri di input e sulla gestione delle eccezioni, trascurando le dinamiche di riempimento del buffer interno. L’ausilio di JaCoCo ha reso evidente la necessità di sollecitare l’interazione tra la saturazione fisica della memoria e la variabile `unpersistedBytesBound`, scenario precedentemente ignorato in cui il `flush` viene scatenato esclusivamente dall’esaurimento dello spazio disponibile. L’analisi strutturale ha quindi favorito il passaggio metodologico dalla verifica del contratto del metodo allo stress-test della gestione interna della memoria. Per completare la copertura è stato introdotto il test strutturale **J-W1**: impostando una `writeCapacity` ridotta (6 byte) a fronte di dati sorgente superiori (13 byte), è stata forzata la saturazione fisica del buffer. Disattivando le soglie di persistenza (`unpersistedBytesBound = 0`), il sistema è stato obbligato a invocare il `flush` automatico verso il *FileChannel* per sola necessità di spazio. Il test ha confermato che i primi 12 byte vengono scaricati correttamente in due cicli iterativi, mentre l’ultimo byte permane nel buffer, validando il comportamento del sistema in presenza di *bound* nullo e saturazione della memoria tecnica.

ID	Allocator	FileChannel	wCap	UBB	src	Esito	Motivazione
J-W1	Valid	Valid	6	0	valid(13)	Passato	Forza la saturazione fisica del buffer (WC=6) tramite un input eccedente per verificare il flush automatico dei dati sul FileChannel.

Tabella 2: Integrazione della suite di test per il metodo `write()` (caso J-W1).

Nel caso del metodo `read`, l’analisi strutturale tramite JaCoCo ha evidenziato inizialmente una lacuna relativa allo statement `break` alla **riga 268**. Tale ramo non veniva sollecitato poiché nei test funzionali iniziali il buffer di scrittura risultava sempre istanziato. Per coprire questa evenienza, è stato introdotto il test strutturale **J-R1**, configurato tramite il parametro `wbNullWriteState`. In questo scenario, viene forzata l’allocazione di un buffer di scrittura nullo (`writeBuffer == null`) tentando una lettura da una posizione interna all’area di scrittura. Il test ha confermato la robustezza del sistema in questa situazione limite, permettendo al flusso di eseguire correttamente il salto logico verso il supporto fisico senza sollevare eccezioni impreviste. Una seconda parzialità di *branch coverage* è stata rilevata alla **riga 270**, relativa alla condizione composta che verifica se la posizione di lettura sia contenuta nel `readBuffer`. Nei test standard, ogni esecuzione utilizza una nuova istanza della classe in cui `readBufferStartPosition` è inizializzato al valore minimo di sistema; di conseguenza, la sotto-condizione `readBufferStartPosition <= pos` risulta un’invariante sempre vera, impedendo a JaCoCo di registrare il ramo `false`. Per risolvere questa lacuna, è stato implementato il test **J-R2** che opera in due fasi sullo stesso oggetto. Dopo una prima lettura a `pos=0` che allinea il puntatore interno (`readBufferStartPosition = 0`), viene eseguita una seconda lettura alla medesima posizione. In questa fase, grazie all’invalidazione del buffer tramite `clearReadBuffer`, la seconda parte della condizione composta (`pos < readBufferStartPosition + writerIndex`) fallisce (risultando $0 < 0 + 0$). Questo forza l’intera espressione logica a `false`, obbligando il sistema a ricaricare i dati dal *FileChannel*. Questo scenario di cache miss forzato ha permesso di coprire il ramo decisionale precedentemente irraggiungibile, portando la *branch coverage* del metodo al 100%.

ID	Allocator	FC	pos	len	Expected	Esito	Motivazione
J-R1	Valid	Valid	$fcl + bbc - 1$	1	0	Passato	Verifica l'interruzione del ciclo di lettura (break) in assenza di un writeBuffer inizializzato, prevenendo accessi a memoria nulla.
J-R2	Valid	Valid	$p_2 < p_1$	5	5	Passato	Valida il ricalcolo degli offset in caso di accesso non sequenziale (lettura arretrata), garantendo il corretto riallineamento del readBuffer.

Tabella 3: Integrazione della suite di test strutturali per il metodo `read` (Casi J-R1, J-R2).

8.1.2 Mutation Testing (PIT)

L'analisi della robustezza della suite di test tramite il framework PIT ha evidenziato un limite di rilevazione nel metodo `write` alla **riga 143** (Figura 1). Nello specifico, un mutante sopravvissuto ha sostituito il controllo condizionale `if (shouldForceWrite)` con una costante `true`, rendendo incondizionata l'invocazione del metodo `forceWrite(false)`. Tale mutazione non è stata intercettata dai test funzionali poiché l'esecuzione superflua della sincronizzazione su disco non altera l'output funzionale del sistema, ma determina esclusivamente un degrado delle prestazioni di I/O. Per ovviare a tale lacuna, si è reso necessario verificare il comportamento interno del metodo. È stato quindi introdotto il test strutturale **T7** (Tabella 4), configurato con una soglia di persistenza elevata (`UBB = 1000`) a fronte di un input ridotto (3 byte). Il test verifica che il metodo `forceWrite` non venga invocato quando i criteri di persistenza non sono soddisfatti. Questa strategia garantisce l'aderenza ai vincoli di efficienza del sistema e permette la corretta soppressione del mutante precedentemente sopravvissuto.

ID	Allocator	FileChannel	WCap	RC	UBB	src	Motivazione
W15	Valid	Valid	1000	100	1000	valid(3)	Valida l'inibizione di <code>forceWrite</code> in assenza del raggiungimento della soglia UBB.

Tabella 4: Integrazione della suite di test per il metodo `write` (W15).

L'analisi condotta tramite il framework di mutation testing PIT ha rivelato la sopravvivenza di un mutante critico alla **riga 272** del metodo `read`, vedi Figura 2. Il codice originale esegue il calcolo dell'offset di copia come segue:

$$\text{bytesToCopy} = \min(\text{readBuffer.writerIndex()} - \text{positionInBuffer}, \text{dest.writableBytes}()) \quad (1)$$

Il mutante generato sostituiva l'operatore di sottrazione con un'addizione. Nelle classi di equivalenza definite nella fase di progetto iniziale, i casi di test strutturati per coprire il `readBuffer` (Classe *C1*) utilizzavano tipicamente un parametro `pos = 0`. In tale scenario, assumendo `readBufferStartPosition = 0` a seguito del reset del buffer, l'offset `positionInBuffer` risultava pari a 0, rendendo l'operazione mutata (`writerIndex + 0`) matematicamente identica all'originale (`writerIndex - 0`). Questa coincidenza numerica rendeva il mutante *equivalente* rispetto all'output per i casi di test selezionati, impedendo la rilevazione dell'errore algoritmico sottostante. Per uccidere definitivamente la mutazione, è stato introdotto il test R24 con l'obiettivo di rompere la simmetria tra addizione e sottrazione. Attraverso l'impostazione di `pos = 3`, si forza `positionInBuffer` a un valore positivo, creando una divergenza netta tra il comportamento atteso ($10 - 3 = 7$) e quello mutato ($10 + 3 = 13$). Questa asimmetria degli offset, combinata con un isolamento delle capacità ottenuto tramite un `readCapacity` ridotto (10 byte) e un buffer di destinazione sovradianimensionato (20 byte), garantisce che il valore restituito dalla funzione `Math.min` sia determinato esclusivamente dal calcolo oggetto di mutazione e non dalla saturazione del buffer di destinazione. Infine, l'asserzione puntuale sul numero di byte letti (7) permette di intercettare il tentativo del codice mutato di copiare un volume errato di dati, portando all'uccisione effettiva del mutante.

ID	Allocator	FileChannel	wCap	UBB	pos	len	Esito	Motivazione
R-24	Valid	Valid	100	1	3	7	Passato	Valida l'accuratezza del calcolo degli offset in scenari di lettura non allineati ($pos > 0$), garantendo che la correttezza non dipenda da coincidenze numeriche ($pos = 0$).

Tabella 5: Integrazione della suite di test per il metodo `read` (Caso R-24).

L'analisi dei mutanti alla **riga 266** ha evidenziato la necessità di rifinire i test di frontiera relativi alla fine del flusso di dati. Sebbene la maggior parte dei mutanti condizionali sia stata eliminata, il mutante relativo al confine (*changed conditional boundary*) richiedeva una verifica puntuale al termine del `FileChannel`. L'integrazione del caso di test P-R0 ha permesso di verificare il comportamento del sistema impostando `pos = 13` (corrispondente alla fine del contenuto su file) in uno stato di `writeBuffer` nullo. In questo scenario, il codice originale deve innescare correttamente l'istruzione di `break`, restituendo 0 byte letti. La capacità del test di convalidare questo limite esatto ha garantito l'uccisione del mutante di boundary, assicurando che la logica di interruzione non venga saltata per errori di off-by-one. In conclusione, l'analisi ha evidenziato la sopravvivenza di ulteriori due mutanti residuali alle **righe 266 e 270**, i quali sono stati classificati come *osservazionalmente equivalenti*. Per quanto concerne la riga 266, il mutante che forza a `true` la condizione di assenza del buffer di scrittura (*removed conditional*) induce un'istruzione di `break` che interrompe prematuramente il ciclo `while`. La sopravvivenza del mutante è dovuta al fatto che, negli scenari in cui i dati richiesti sono

già stati interamente prelevati o il buffer di scrittura non contiene dati validi per la posizione corrente, l'uscita forzata tramite `break` produce un valore di ritorno identico al termine naturale del ciclo per esaurimento delle sorgenti. Di conseguenza, il mutante risulta indistinguibile dal comportamento nominale poiché l'interruzione avviene quando non vi sono ulteriori dati da leggere. Relativamente alla riga 270, invece, il mutante di boundary (\leq sostituito con $<$) è risultato algoritmicamente neutro rispetto all'output. Qualora la posizione di lettura coincida esattamente con l'inizio del buffer (`pos == readBufferStartPosition`), il fallimento della guardia mutata causa il bypass della cache di lettura, forzando il sistema a recuperare i dati direttamente dal `FileChannel`. Poiché il contenuto del file e quello del buffer di lettura sono sincronizzati, il dato finale restituito rimane identico. Pur degradando marginalmente l'efficienza algoritmica a causa dell'accesso al supporto fisico invece che alla memoria RAM, la mutazione non altera la correttezza del sistema, confermandosi come equivalente ai fini della validazione funzionale.

8.2 Definizione e risultati dei test di WriteCache

8.2.1 Statement Coverage e Branch Coverage (JaCoCo)

L'analisi della copertura del codice sulla classe `WriteCache` (si veda la Tabella 99) ha inizialmente evidenziato un'importante lacuna metodologica nel metodo `put`. Sebbene la progettazione teorica avesse correttamente identificato la classe di equivalenza *E1* ($0 \leq entryId < lastEntryId$) per gli inserimenti fuori ordine, l'implementazione della prima suite di test è risultata deficitaria a causa di un approccio eccessivamente semplificato. L'errore è consistito nel trascurare l'impatto dello stato interno della cache: operando esclusivamente su una struttura vuota (`NON_WRITTEN`), i test ignoravano sistematicamente i valori di frontiera e le condizioni critiche di aggiornamento. Per superare tale limite, la suite è stata attivamente potenziata con l'integrazione di nuovi casi di test mirati, atti a stabilire uno stato non vuoto della cache come precondizione necessaria. Solo attraverso laggiunta di queste procedure è stato possibile sollecitare i valori di frontiera e verificare il comportamento del sistema a fronte di inserimenti *out-of-order* (P20), duplicati (P21) e sequenziali (P22), garantendo infine la copertura del ramo decisionale alla **riga 171**. Per quanto riguarda la **riga 176** (`compareAndSet`), si è deciso consapevolmente di non forzarne la copertura. La documentazione Javadoc interna al codice sorgente chiarisce esplicitamente che tale logica è implementata per gestire scritture dello stesso ledger provenienti da thread differenti, sottolineando tuttavia che "*in practice it should not happen and the compareAndSet should be always uncontended*". Coerentemente con queste indicazioni tecniche, il mantenimento della copertura all'87% risulta giustificato dalla natura puramente protettiva del ramo, volto alla sincronizzazione atomica in scenari di contesa multi-thread che non rientrano nel perimetro funzionale della presente suite di test.

8.2.2 Mutation Testing (PIT)

L'analisi del costruttore condotta tramite il framework PIT ha evidenziato la sopravvivenza di mutanti alla **riga 95** (*changed conditional boundary e removed conditional*), come illustrato nella Figura 3. Sebbene i casi di test sollecitino tali rami decisionali, i mutanti sopravvivono poiché la suite originaria si limita a verificare il fallimento della creazione dell'oggetto, senza vincolarsi alla classe specifica dell'eccezione o al contenuto del messaggio d'errore. Dall'analisi del codice sorgente, tali mutazioni sono classificabili come mutanti equivalenti. Infatti, la robustezza strutturale del sistema garantisce l'interruzione dell'esecuzione anche in assenza della clausola di salvaguardia alla riga 95: un valore di `maxSegmentSize` nullo causerebbe un'eccezione aritmetica alla riga 106 durante il calcolo dei segmenti, mentre un valore negativo verrebbe intercettato alla riga 112 dal metodo `Unpooled.directBuffer`. Si è scelto pertanto di non sovrastipiccare i test, privilegiando l'indipendenza della suite dall'implementazione interna; la sopravvivenza di tali mutanti non rappresenta dunque una lacuna del testing, bensì una proprietà di ridondanza del *System Under Test* (SUT). A differenza del caso precedente, il mutante alla riga 104 (*replaced integer subtraction with addition*) permette la corretta istanziazione di `WriteCache` corrompendone però silenziosamente lo stato interno. L'errore nel calcolo di `segmentOffsetBits` rimarrebbe latente, manifestandosi solo durante le successive operazioni di scrittura. Per garantire la coerenza dello stato sin dalla costruzione, si è deciso di estendere la suite di test. Data l'assenza di getter pubblici, è stato necessario ricorrere alla *reflection* per ispezionare il campo privato e validare la configurazione della memoria. Questa verifica ha permesso di sopprimere il mutante, assicurando l'integrità della logica di partizionamento. Infine, il mutante alla **riga 110** (*condition boundary* da $<$ a \leq) è stato classificato come equivalente. Sebbene la mutazione induca l'allocazione dell'ultimo elemento (`cacheSegments[segmentsCount-1]`) all'interno del ciclo `for`, la medesima operazione viene reiterata immediatamente all'uscita del ciclo (riga 115). In tale sede, al segmento viene assegnato il valore corretto previsto dalla logica originale, sovrascrivendo l'allocazione mutata e rendendo l'effetto del mutante nullo sullo stato finale dell'oggetto.

L'analisi del mutante alla **riga 198** nel metodo `get()` ha rivelato una lacuna strutturale nella suite di test originale, illustrata in Figura 4). La mutazione *Replaced Unsigned Shift Right with Shift Left* è inizialmente sopravvissuta poiché i casi di test operavano su volumi di dati ridotti, mantenendo l'offset all'interno del primo segmento di memoria (`segmentIdx = 0`). In tale scenario, l'operazione di *shift* produce il medesimo indice nullo indipendentemente dalla direzione dello scorrimento dei bit. Per risolvere questa criticità, la suite è stata integrata con il caso di test **G12**, i cui dettagli sono riportati nella Tabella 6. Configurando la cache con una granularità di segmentazione ridotta (`maxSegmentSize = 256`) e forzando la scrittura di un'entry con un *offset* superiore alla dimensione del singolo segmento, il calcolo dell'indice tramite *shift* a sinistra genera un valore errato o una violazione dei limiti dell'array (`ArrayIndexOutOfBoundsException`). L'introduzione di questo scenario ha permesso di validare rigorosamente la logica di indirizzamento bitwise, garantendo la soppressione del mutante e assicurando che il recupero dei dati avvenga dal segmento di memoria corretto.

ID	Allocator	maxCS	maxSS	LID	EID	writeCacheState	Output Atteso	Esito	Motivazione
G12	Valid	1024	256	10	22	WRITTEN	Valid Entry	Passato	Validazione dell'indirizzamento di memoria per offset superiori al segmento primario ($segmentIdx > 0$).

Tabella 6: Integrazione della suite di test per il metodo get (Caso G12).

Il mutante sopravvissuto alla **riga 176** (*removed conditional - replaced equality check with true*), illustrato in Figura 6, costituisce un caso emblematico di non rilevabilità in ambiente sequenziale. L'istruzione originale impiega un'operazione `compareAndSet` per prevenire la corruzione dei dati durante inserimenti concorrenti sul medesimo `ledgerId`. Poiché l'attuale suite di test opera in un singolo thread di controllo, la condizione di *race condition* necessaria a invalidare il CAS non può manifestarsi. Pertanto, la mutazione del ramo decisionale a `true` non altera il flusso deterministico dei test unitari. Si conclude che tale sopravvivenza non indichi una lacuna funzionale, bensì evidenzi la necessità di **stress testing concorrente** per validare i meccanismi di sincronizzazione, attività che esula dal perimetro della presente suite unitaria. L'analisi del mutante alla **riga 154** (sostituzione dell'operatore `- con +`) ha, invece, permesso di individuare una criticità nella gestione della memoria frammentata. Per indagare tale lacuna, è stato introdotto il caso di test **P-P3** (Tabella 7), configurando una cache con `maxCacheSize=256` e `maxSegmentSize=128`. Lo scenario prevede un primo segmento parzialmente occupato (64 byte), lasciando il secondo segmento disponibile per un'entry da 128 byte. Nonostante l'output atteso sia `true`, l'esecuzione ha rivelato che il *System Under Test* (SUT) restituisce `false`. L'anomalia è riconducibile a un **bug di design** nella gestione dell'offset globale: l'avanzamento tramite `getAndAdd` (riga 147), essendo un'operazione non transazionale, incrementa irreversibilmente l'indice senza rialineararlo in caso di salto del segmento (*segment skip*). In questo contesto, il mutante ha agito da oracolo negativo: producendo un calcolo palesemente errato ($128 + 64 = 192$ invece di $128 - 64 = 64$), ha tentato una scrittura illegale sollevando un'eccezione di memoria. Sebbene la mutazione sia tecnicamente rilevabile per la divergenza di comportamento (eccezione contro valore booleano errato), si è scelto di non forzarne l'uccisione alterando i test, preferendo documentare il comportamento anomalo del SUT come risultato critico.

ID	Allocator	maxCS	maxSS	writeCacheState	Entry	Output Atteso	Esito
P-P3	Valid	256	128	HALF_FILLED	128	TRUE	FALLITO

Tabella 7: Integrazione della suite di test per il metodo put (Caso P-P3).

L'analisi del fallimento ha rivelato un importante **bug di design** nel SUT. L'uso del metodo atomico `getAndAdd` alla LOC 147 introduce una criticità strutturale, rendendo l'incremento dell'offset globale non atomico rispetto alla logica di partizionamento della memoria. In scenari di frammentazione, il sistema riserva porzioni di indirizzi virtuali prima di validarne l'effettiva appartenenza ai limiti del segmento corrente; tuttavia, qualora l'allocazione fallisca richiedendo un salto al segmento successivo (*segment skip*), l'offset non viene ripristinato. Questo comportamento determina un consumo irreversibile della capacità teorica della cache: ogni iterazione del ciclo `while` sottrae spazio alla `maxCacheSize` senza che avvenga una scrittura reale, esaurendo prematuramente lo spazio virtuale e impedendo l'allocazione persino in segmenti completamente integri. Tale anomalia, evidenziata dal fallimento del test **P-P3**, dimostra come la gestione della memoria sia compromessa da una mancanza di coordinamento tra l'indice globale atomico e i confini fisici dei segmenti.

8.3 Definizione e risultati dei test di ClassUtil

8.3.1 Statement Coverage e Branch Coverage (JaCoCo)

L'analisi della copertura effettuata con JaCoCo (si veda la tabella 101) ha evidenziato i limiti di una generalizzazione eccessiva delle classi di equivalenza. Nella fase preliminare, la definizione delle categorie basata esclusivamente sull'interfaccia pubblica ha portato all'accorpamento di scenari di input che, all'ispezione dei rami esecutivi, si sono rivelati logicamente divergenti. Il superamento di tale astrazione ha richiesto una scomposizione granulare della categoria `Classe array`. Originariamente trattata come entità singola, essa è stata distinta tra descrittori di **tipo primitivo** e di **tipo riferimento**. Questa distinzione è fondamentale poiché i primitivi (es. "[I") agiscono come costanti di terminazione, mentre i riferimenti (es. "[Ljava/lang/Object;") introducono metacaratteri strutturali che attivano logiche di *parsing* ricorsive o differenziate. Parallelamente, l'inclusione di varianti basate sulla *slash-notation* (es. "java/lang/String") ha evidenziato una vulnerabilità funzionale dovuta alla dipendenza esclusiva dal metodo `lastIndexOf('.'`). Tale limite deriva dall'origine della logica di `toClass`, mutuata dal progetto *Serp* (specializzato in manipolazione di bytecode), che privilegia la velocità di elaborazione dei descrittori rispetto alla normalizzazione dell'input. Di conseguenza, il *parser* non riconosce correttamente gli *Internal Names* definiti dalle specifiche della JVM (dove lo *slash* funge da separatore), portando il metodo a restituire l'intero percorso come `simpleName` e compromettendo la robustezza della libreria in contesti di integrazione eterogenei. Al termine del raffinamento, la suite di test ha permesso di raggiungere il 100% di *Statement* e *Branch Coverage* in quasi tutti i metodi della classe, risolvendo anche la copertura inizialmente nulla in `toClass(String, ClassLoader)` tramite l'integrazione di chiamate mirate alla logica di delega interna. Tuttavia, il mancato raggiungimento della copertura totale nel metodo `getClassName` rappresenta una **scelta metodologica deliberata**. Sebbene siano stati identificati i vettori di input necessari per sollecitare i rami decisionali rimanenti, si è scelto di mantenere tali test (**T4.6** e **T4.7**) in uno stato di fallimento documentato per non occultare i difetti di robustezza emersi (Tabella 102):

- **Caso T4.6 ("[X]"):** Il sistema non identifica il carattere 'X' come codice di tipo invalido, restituendo erroneamente "[X]" invece di sollevare un'eccezione.
- **Caso T4.7 ("[Ljava.lang.String"):** La mancanza del terminatore ";" obbligatorio viene ignorata, eseguendo una substring arbitraria che restituisce "String[]".

Forzare il superamento di questi test allineando l'oracolo a un comportamento non conforme alle specifiche JVM avrebbe garantito una metrica di *coverage* ideale, ma avrebbe validato un'implementazione semanticamente errata.

8.3.2 Mutation Testing (PIT)

L'analisi condotta con PIT ha evidenziato la sopravvivenza di una mutazione alla **riga 148** (Figura 7), relativa alla validazione dei descrittori array. Tale sopravvivenza è riconducibile all'esclusione forzata del caso di test **T4.6** (input "[X]"), necessaria per garantire il superamento della suite, requisito vincolante per l'esecuzione del framework di mutation testing. Il test T4.6 punta a un difetto del codice originale che non solleva l'eccezione prevista; finché tale test rimane disabilitato per preservare la stabilità della suite, la mutazione non può essere intercettata. L'uccisione del mutante richiederebbe un intervento correttivo sul codice sorgente o un adeguamento dell'oracolo al comportamento del SUT. La sopravvivenza del mutante alla **riga 158** ha, invece, rivelato una lacuna nella *Boundary Value Analysis* originale, che non considerava la posizione 0 come valore critico per il delimitatore. Per colmare tale zona d'ombra, è stato introdotto il test case **T8** (input: ".SimpleClass"), volto a verificare il comportamento del metodo quando il separatore di package precede immediatamente il nome semplice della classe (vedi 8). Nonostante l'introduzione di T8, alcune mutazioni alla **riga 158**, in particolare (`if (lastDot > -1)`) sono risultate **equivalenti**. Forzando la condizione a `true`, il codice esegue sistematicamente `fullName.substring(lastDot + 1)`. In assenza di punti (`lastDot = -1`), l'istruzione converge in `substring(0)`, restituendo l'intera stringa. Poiché tale risultato coincide con il ramo `else` originale, la mutazione produce una ridondanza logica che la rende indistinguibile dai test funzionali.

Parallelamente, all'interno del metodo `getPackageName` (Figura 8) la mutazione alla **riga 200** (`fullName.length() == dims + 1`) sopravvive a causa di una coincidenza strutturale tra la logica mutata e la sintassi dei descrittori JVM per gli array primitivi. In Java, un descrittore di questo tipo (ad esempio "[I" o "[D") presenta una struttura fissa, in cui la lunghezza della stringa è sempre pari al numero di dimensioni (caratteri '[') più il singolo carattere identificativo del tipo. Per l'intero dominio degli input validi relativi ai primitivi, la condizione originale risulta dunque essere una tautologia. Sostituire tale controllo con la costante `true` non produce deviazioni nel flusso di esecuzione né nell'output finale. Poiché i casi che potrebbero invalidare questa uguaglianza (come i descrittori di oggetti, caratterizzati da lunghezza variabile) sono gestiti da rami logici differenti prima di raggiungere la riga in esame, la mutazione rimane intrinsecamente non rilevabile.

L'analisi dei mutanti sopravvissuti per il metodo `toClass` (Figura 9) permette di distinguere nettamente tra i margini di miglioramento della suite di test e i limiti intrinseci della metodologia funzionale adottata. I mutanti riscontrati alle **righe 76, 77 e 89** evidenziano scenari in cui l'approccio *Black-Box* risulta strutturalmente insufficiente. In questi casi, le mutazioni colpiscono ottimizzazioni del flusso d'esecuzione o parametri di gestione della memoria (come il ridimensionamento dinamico dello `StringBuilder` o cicli di ricerca ridondanti) che non alterano il risultato funzionale restituito. L'uccisione di tali mutanti non dipenderebbe da una migliore definizione delle classi di equivalenza, ma richiederebbe un approccio *White-Box* supportato da strumenti di *memory profiling* o *benchmarking*, in grado di rilevare l'esecuzione di codice superfluo o l'allocazione inefficiente di risorse. La sopravvivenza del mutante alla **riga 103** suggeriva inizialmente la necessità di un raffinamento della *Boundary Value Analysis* riguardo alla logica di fallback del `ClassLoader`. Sebbene la classe di equivalenza relativa al parametro `loader` nullo fosse stata individuata, l'oracolo originario (test **T10** con "`java.lang.String`") mancava di potere discriminante: essendo una classe di sistema, `String` viene risolta correttamente sia dal `ContextClassLoader` che dal `Bootstrap Loader`. Per uccidere il mutante, è stato aggiunto il test **T10.1** (Tabella 9) utilizzando una classe definita internamente al progetto ("`manualtest.ClassUtilToClassTest`"), risolvibile esclusivamente tramite il `ContextClassLoader`. Parallelamente, l'introduzione di **T14** con un `CustomLoader` esplicito ha permesso l'uccisione delle restanti mutazioni (*replaced with true*). Nonostante, però, il raffinamento di **T10**, la mutazione "*replaced equality check with true*" alla **riga 103** (`if (loader == null)`) è sopravvissuta. Tale persistenza è imputabile alla **coincidenza dei ClassLoader** nell'ambiente di esecuzione. Forzando la condizione a `true`, il metodo attiva sistematicamente la logica di fallback (`Thread.currentThread().getContextClassLoader()`), ignorando il `loader` fornito in input. Tuttavia, poiché nei comuni test runner il `ClassLoader` di sistema e il `ContextClassLoader` del thread condividono il medesimo `classpath`, la classe viene risolta in modo identico da entrambi.

ID	Stringa (fullName)	Output Atteso	Esito
T8	".SimpleClass"	"SimpleClass"	Passato

Tabella 8: Integrazione della suite di test per il metodo `getClassName` (Caso T8).

8.4 Definizione e risultati dei test di CacheMap

8.4.1 Statement Coverage e Branch Coverage (JaCoCo)

Il raggiungimento del 100% di *Statement Coverage* e *Branch Coverage* per i metodi della classe `CacheMap` (si veda la Tabella 100) è stato conseguito tramite un raffinamento iterativo delle classi di equivalenza. Tale risultato certifica l'adeguatezza

ID	Stringa (str)	Resolve	Loader	Output Atteso	Esito
T10.1	"manualtest.ClassUtilToClassTest"	true	null	ClassUtilToClassTest.class	Passato
T14	"ClasseCustom"	false	CustomLoader	ClasseCustom.class	Passato

Tabella 9: Integrazione della suite di test per il metodo `toClass` (Caso T10.1 e T14).

della suite di test rispetto alla struttura logica implementata. Tuttavia, è opportuno precisare che la copertura strutturale non riflette necessariamente la qualità intrinseca della suite in termini di *fault detection capability*; pertanto, l'efficacia del testing verrà ulteriormente indagata nel prosieguo del report attraverso l'analisi di *Mutation Testing* mediante lo strumento PIT.

8.4.2 Mutation Testing (PIT)

L'analisi del metodo `pin` ha evidenziato diverse tipologie di mutazioni sopravvissute, vedi Figura 10. In primo luogo, la mutazione alla **riga 306**, riguardante l'incremento di `_pinnedSize`, è risultata inizialmente non rilevata a causa di asserzioni eccessivamente deboli, focalizzate unicamente sull'output booleano del metodo. Tale criticità è stata risolta potenziando la suite di test con verifiche sulla `size()` totale, garantendo così la coerenza dello stato interno e la conseguente estinzione del mutante. In secondo luogo, i mutanti individuati alle **righe 307 e 309** sono da considerarsi equivalenti: lo strumento PIT ha sostituito i valori di ritorno con costanti identiche a quelle originali, producendo un bytecode funzionalmente invariato e, pertanto, non rilevabile. Infine, la sopravvivenza della mutazione relativa al metodo `writeUnlock` alla **riga 311** è riconducibile alla natura *single-threaded* della suite di test. Poiché la mancata gestione dei lock manifesta i propri effetti critici esclusivamente in scenari di contesa multi-thread, tale mutazione risulta non significativa ai fini della validazione in ambiente *single-thread*.

L'analisi del metodo `unpin` delinea uno scenario analogo a quanto riscontrato per il metodo `pin`, come evidenziato nella Figura 11. La mutazione individuata alla **riga 325**, relativa al decremento della variabile `_pinnedSize`, è stata efficacemente soppressa mediante l'integrazione di asserzioni basate sul metodo `size()`; tale accorgimento ha permesso di monitorare con precisione le variazioni del contatore interno, rendendo rilevabile ogni alterazione dello stato. Per quanto concerne le mutazioni alle **righe 326 e 328**, esse sono risultate sopravvissute poiché classificabili come mutanti equivalenti, non producendo variazioni funzionali discriminabili rispetto al comportamento atteso. Infine, la mancata uccisione della mutazione alla **riga 330**, corrispondente all'istruzione `writeUnlock`, ribadisce i limiti intrinseci della test suite nel dominio della sincronizzazione. Coerentemente con quanto osservato in precedenza, l'assenza di scenari di contesa multi-thread rende tale mutazione non rilevante nel contesto operativo attuale, in quanto l'eventuale omissione dei meccanismi di rilascio dei lock non produce effetti osservabili in un ambiente strettamente sequenziale.

L'analisi del metodo `get`, vedi Figura 12, ha rivelato una vulnerabilità nel processo di validazione alla **riga 378**, dove il mutante *negated conditional* associato al flag `putcache` è inizialmente sopravvissuto. Tale flag governa la logica di promozione degli oggetti dalla `softMap` alla `cacheMap` primaria; la sopravvivenza del mutante evidenzia come la suite di test originale fosse limitata alla verifica del valore di ritorno, trascurando gli effetti collaterali per la gestione della memoria. Per ovviare a tale lacuna, la test suite è stata raffinata per monitorare le invocazioni interne al metodo `put`. Questo raffinamento assicura che il reperimento di un elemento nella `softMap` attivi sistematicamente la procedura di riposizionamento nella cache primaria, prevenendo l'espulsione prematura dei dati da parte del *Garbage Collector*. Inoltre, la mutazione relativa alla rimozione della chiamata di rilascio del lock **riga 375** veniva inizialmente rilevata da PIT come *TIMED_OUT*. L'evoluzione della suite di test ha permesso di formalizzare il monitoraggio del protocollo di sincronizzazione, introducendo una verifica esplicita sull'invocazione di `readUnlock`. Tale approccio ha trasformato un evento temporale incerto in un fallimento logico immediato, garantendo l'uccisione del mutante.

L'analisi del metodo `put`, vedi Figura 13, ha inizialmente evidenziato una significativa densità di mutazioni sopravvissute, concentrate in particolare sulla gestione degli *hook* di notifica e sul mantenimento dello stato del contatore `_pinnedSize`. Per quanto concerne la `pinnedMap`, la mutazione alla **riga 392** (`_pinnedSize++`) è stata estinta raffinando i casi di test per includere l'inserimento di un valore in una chiave precedentemente inizializzata tramite `pin()` ma con riferimento `null`. Tale scenario ha permesso di validare l'incremento dello stato interno tramite il monitoraggio del metodo pubblico `size()`, rendendo rilevabile l'omissione dell'operazione di incremento prevista. Le mutazioni sopravvissute alle **righe 393-396 e 411-418** riguardavano la rimozione delle chiamate ai metodi `entryAdded` ed `entryRemoved`. La sopravvivenza di tali mutanti era riconducibile alla natura *protected* di tali metodi nella classe base, che ne rendeva l'eliminazione priva di effetti collaterali osservabili tramite le API pubbliche. La vulnerabilità è stata risolta mediante l'introduzione di una *Spy Class* derivata, denominata `ObservableCacheMap`, che sovrascrive gli *hooks* per incrementare contatori interni. Questo approccio ha trasformato metodi originariamente non osservabili in comportamenti verificabili, garantendo la corretta attivazione delle notifiche del ciclo di vita durante le operazioni di inserimento e rimozione. Infine, la mutazione alla **riga 422** (`writeUnlock`) è risultata sopravvissuta a causa dei vincoli dell'ambiente di esecuzione. Sebbene sia stato adottato un *timeout* rigoroso per prevenire stati di stallo, la natura *single-threaded* della suite non consente di generare una contesa sul lock (*lock contention*). È interessante notare l'asimmetria rispetto alla **riga 385**: mentre la rimozione di `writeLock()` viene rilevata (*Killed*) poiché la successiva invocazione di `writeUnlock()` solleva un'eccezione immediata, la mutazione sulla **riga 422** non produce errori fatali né *deadlock* in un contesto sequenziale. In assenza di thread concorrenti che tentino l'accesso alla risorsa, la mancata liberazione del lock non è rilevabile. In conclusione, gli interventi effettuati confermano come il raggiungimento del 100% di copertura strutturale non sia con-

dizione sufficiente a garantire la qualità della test suite, la quale è stata resa realmente robusta solo a seguito di una sistematica analisi delle mutazioni.

9 Integration Testing

Collocazione dei sorgenti

La classe di test sviluppata è reperibile al seguente percorso:

https://github.com/GaiaMeola/bookkeeper/tree/test_isw2/bookkeeper-server/src/test/java/manualtest

Analisi dell'integrazione: SingleDirectoryDbLedgerStorage e WriteCache

L'attività di test di integrazione si è focalizzata sulla relazione tra la classe `SingleDirectoryDbLedgerStorage` e la sua dipendenza funzionale `WriteCache`. Tale accoppiamento rappresenta un punto critico per il sistema di storage, poiché governa l'efficienza delle operazioni di scrittura attraverso un meccanismo di caching a doppia istanza. Il metodo principale oggetto di analisi, `addEntry`, coordina l'aggiunta di nuove entry nel ledger sfruttando due istanze di cache: una `writeCache` attiva per le operazioni correnti e una `writeCacheBeingFlushed` destinata alla persistenza asincrona su disco. Questa architettura abilita la cosiddetta *rotazione della cache* tramite il metodo `swapWriteCache`, garantendo la continuità delle scritture anche durante le fasi di flushing. Per la gestione della concorrenza, il sistema adotta una strategia di lettura ottimistica mediata da uno `StampedLock`. La validazione del timestamp è essenziale: se un inserimento fallisce a causa di una rotazione concorrente (rilevata tramite *optimistic lock validation*) o della saturazione della memoria, il sistema reagisce eseguendo un nuovo tentativo o invocando il metodo `triggerFlushAndAddEntry` per liberare risorse. La strategia adottata segue un approccio gerarchico **Top-Down**, procedendo all'integrazione delle componenti dall'alto verso il basso lungo il grafo delle dipendenze. In questa metodologia, l'analisi parte dai moduli **radice**, ovvero i nodi privi di archi entranti, per poi scendere iterativamente verso i moduli chiamati. Nel sistema in esame, il ruolo di modulo **Top** è ricoperto da `DbLedgerStorage`, mentre la `WriteCache` rappresenta il modulo **Down**. Coerentemente con i principi del Top-Down, l'integrazione è stata avviata isolando la logica di controllo del Ledger tramite l'impiego di **stub**, tecnica che ha permesso di validare la corretta propagazione dei parametri e i flussi decisionali superiori in totale isolamento dalla dipendenza reale. Per superare i vincoli di incapsulamento del framework e penetrare l'arco di dipendenza, è stata progettata la classe `MockedDbLedgerStorage`: eseguendo l'override del metodo `factory newSingleDirectoryDbLedgerStorage`, questa struttura ha agito come punto di iniezione per inserire uno **Spy** di **Mockito** direttamente nel cuore del sistema. L'evoluzione verso l'uso dello **Spy** è stata determinante per superare i limiti dei test statici; a differenza di uno stub, lo **Spy** ha consentito di preservare la logica operativa originale della `WriteCache`, rendendo possibile la verifica di scenari dinamici complessi. Attraverso questa configurazione, è stato possibile indurre e monitorare analiticamente fenomeni critici come la saturazione deterministica della memoriae la conseguente rotazione automatica delle istanze di cache. Questo percorso ha garantito la validazione dell'intero grafo delle dipendenze, verificando che il coordinamento tra i moduli rispetti rigorosamente i protocolli di resilienza del progetto. Per la definizione degli scenari di test è stata applicata la tecnica del *Category Partition* (i cui dettagli sono consultabili nell'Appendice C). A differenza dello unit testing, l'analisi si è focalizzata sulle combinazioni degli stati interni dei componenti e sulla gestione della concorrenza. In particolare, le categorie individuate sono state incrociate per validare i tre livelli di resilienza previsti nel metodo `addEntry`. Partendo dal *flusso nominale* (lock valido e memoria libera), la suite di test esplora scenari progressivamente più critici, simulando rotazioni concorrenti della cache e condizioni di saturazione estrema. Il dettaglio dei casi di test eseguiti e i relativi parametri di verifica sono riportati nell'Appendice C. In conclusione, tutti i test sono stati superati con successo. I risultati confermano che l'integrazione tra `SingleDirectoryDbLedgerStorage` e `WriteCache` è robusta: il sistema gestisce correttamente la concorrenza ottimistica e garantisce la persistenza dei dati anche in condizioni di saturazione critica della memoria, prevenendo efficacemente la perdita di entry.

10 Stima della Reliability

In questa sezione viene condotta una valutazione della stabilità del sistema sotto il profilo della *Software Reliability*. La metrica è qui declinata come la probabilità di funzionamento esente da guasti (*failure-free operation*) per un numero prefissato di esecuzioni, assumendo un profilo operazionale uniforme in cui gli input sono campionati casualmente dal dominio di test.

10.1 Metodologia di Calcolo

Per ogni classe e per i relativi metodi, la reliability è stata stimata utilizzando la formula:

$$R = 1 - PFD = 1 - \frac{N_{fail}}{N_{tot}} = \frac{N_{pass}}{N_{tot}} \quad (2)$$

dove N_{pass} indica il numero di test superati nella suite finale e N_{tot} il numero totale di test eseguiti.

10.2 Analisi per Classe e Metodo

10.2.1 Classe BufferedChannel

- **Costruttore:** $R = \frac{15}{17} = 88.24\%$
- **Metodo write:** $R = \frac{14}{16} = 87.50\%$

- **Metodo read:** $R = \frac{26}{26} = 100\%$
- **Test totali:** 69
- **Test passati:** 65
- **Reliability stimata:** $R = \frac{65}{69} = 94.20\%$

10.2.2 Classe WriteCache

- **Costruttore:** $R = \frac{11}{13} = 84.62\%$
- **Metodo put:** $R = \frac{22}{23} = 95.65\%$
- **Metodo get:** $R = \frac{12}{12} = 100\%$
- **Test totali:** 48
- **Test passati:** 45
- **Reliability stimata:** $R = \frac{45}{48} = 93.75\%$

10.2.3 Classe ClassUtil

- **Metodo getClassName(Class):** $R = \frac{2}{2} = 100\%$
- **Metodo getPackageName(Class):** $R = \frac{3}{3} = 100\%$
- **Metodo getClassName(String):** $R = \frac{12}{16} = 75,00\%$
- **Metodo getPackageName(String):** $R = \frac{8}{10} = 80,00\%$
- **Metodo toClass:** $R = \frac{17}{18} = 94,44\%$
- **Test totali:** 52
- **Test passati:** 42
- **Reliability stimata:** $R = \frac{42}{52} = 80,77\%$

10.2.4 Classe CacheMap

- **Costruttore:** $R = \frac{11}{13} = 84.62\%$
- **Metodo pin:** $R = \frac{8}{8} = 100\%$
- **Metodo unpin:** $R = \frac{5}{5} = 100\%$
- **Metodo put:** $R = \frac{13}{13} = 100\%$
- **Metodo get:** $R = \frac{6}{6} = 100\%$
- **Test totali:** 45
- **Test passati:** 43
- **Reliability stimata:** $R = \frac{43}{45} = 95.56\%$

A Appendice: Listati di Codice e Tabelle

Per ragioni di leggibilità e in accordo con i requisiti del report, vengono qui riportati i materiali tecnici prodotti riferiti nelle sezioni precedenti.

A.1 Configurazione Workflow GitHub Actions - BookKeeper

In questo paragrafo viene riportato il file di configurazione `.github/workflows/bookkeeper-test.yml` utilizzato per l'automazione del ciclo di build e test dei progetti Apache analizzati. Per il progetto **Apache OpenJPA**, è stata adottata una configurazione speculare, pertanto non si è ritenuto necessario duplicare il listato.

```
1 name: Build and Test BookKeeper
2
3 on:
4   push:
5     branches: [ master, test_lsw2 ]
6   pull_request:
7     branches: [ master, test_lsw2 ]
8
9 env:
10   RUN_TESTS: true
11
12 jobs:
13   build:
14     runs-on: ubuntu-latest
15
16   steps:
17     - name: Checkout repository
18       uses: actions/checkout@v4
19
20     - name: Set up JDK 11
21       uses: actions/setup-java@v4
22       with:
23         distribution: temurin
24         java-version: '11'
25
26     - name: Cache Maven packages
27       uses: actions/cache@v4
28       with:
29         path: ~/.m2/repository
30         key: ${{ runner.os }}-maven-${{ hashFiles('**/pom.xml') }}
31         restore-keys: |
32           ${{ runner.os }}-maven
33
34     - name: Build and Install (No Tests)
35       run: mvn -B clean install -DskipTests
36
37     - name: Run Unit Tests (Surefire)
38       if: env.RUN_TESTS == 'true'
39       run: mvn -B test
40
41     - name: Run Integration Tests (Failsafe)
42       if: env.RUN_TESTS == 'true'
43       run: mvn -B verify -DskipUnitTests
44
45     - name: Tests disabled
46       if: env.RUN_TESTS != 'true'
47       run: echo "Tests are disabled"
```

Listing 1: Workflow YAML per la CI di Apache BookKeeper

A.2 Template (Zero-shot) - Bookkeeper

Agisci come un esperto Software Tester. Il tuo compito è generare una suite di test JUnit 5 e Mockito per il metodo `[METODO]` della classe `BufferedChannel`. L'output deve essere racchiuso tra i tag `###Test START##` e `###Test END##`. [Seguono tabelle EP e Codice Sorgente specifico]

In fase di esecuzione, il segnaposto `[METODO]` è stato sostituito rispettivamente con `Costruttore`, `Read` e `Write`, fornendo di volta in volta le tabelle di partizionamento e il codice sorgente corrispondente.

A.3 Dettaglio degli attributi della classe

Attributo	Descrizione
writeCapacity	Capacità massima del buffer di scrittura in byte.
writeBufferStartPosition	Posizione assoluta nel file dell'inizio del contenuto nel buffer.
writeBuffer	Buffer per lo storage temporaneo dei dati in attesa di scrittura.
position	Offset assoluto del prossimo byte da scrivere nel file.
unpersistedBytesBound	Soglia di byte non persistiti che attiva il <i>flush</i> forzato.
doRegularFlushes	Flag per <i>flush</i> periodici indipendenti dalla soglia.
unpersistedBytes	Conteggio byte in buffer/cache non ancora sincronizzati su disco.
closed	Flag di stato per la chiusura del canale.
fileChannel	Canale di basso livello per l'I/O fisico su disco.
readCapacity	Dimensione massima del buffer di lettura.
readBuffer	Buffer temporaneo per i dati letti dal file.
readBufferStartPosition	Offset nel file corrispondente all'inizio del buffer di lettura.
invocationCount	Numero totale di operazioni di lettura richieste.
cacheHitCount	Numero di letture soddisfatte direttamente dalla cache.

Tabella 10: Descrizione degli attributi principali per l'analisi del testing.

Attributo	Descrizione
index	Indice delle <i>entry</i> (mappa chiave ledgerID/entryID in offset/length).
lastEntryMap	Indice dell'ultima <i>entry</i> scritta per ciascun Ledger.
cacheSegments	Array di segmenti di memoria diretta (<i>off-heap</i>) per lo storage dei dati.
segmentsCount	Numero complessivo di segmenti presenti in cacheSegments.
maxCacheSize	Dimensione massima totale consentita per la cache.
maxSegmentSize	Dimensione massima configurata per ogni singolo segmento.
segmentOffsetMask	Maschera bitwise per calcolare l'offset locale nel segmento.
segmentOffsetBits	Bit di <i>shift</i> per ricavare l'indice del segmento dall'offset globale.
cacheSize	Valore atomico della memoria attualmente occupata.
cacheOffset	Offset globale incrementale aggiornato a ogni scrittura.
cacheCount	Numero totale di <i>entry</i> attualmente residenti nella cache.
deletedLedgers	Set concorrente degli ID dei Ledger contrassegnati per l'eliminazione.
allocator	Componente per l'allocazione dei buffer di memoria (ByteBuf).

Tabella 11: Descrizione degli attributi principali per l'analisi del testing.

Attributo	Descrizione
cacheMap	Mappa contenente i riferimenti ai frammenti non scaduti e non bloccati.
softMap	Mappa dedicata alla gestione dei riferimenti scaduti.
pinnedMap	Mappa contenente i riferimenti ai frammenti bloccati (<i>pinned</i>).
pinnedSize	Contatore degli elementi attualmente bloccati che possiedono un valore.
rwl	<i>Read-Write Lock</i> configurato in modalità <i>fair</i> (FIFO) per gestire l'accesso concorrente.
_readLock	Riferimento rapido al blocco di lettura del rwl.
_writeLock	Riferimento rapido al blocco di scrittura del rwl.

Tabella 12: Descrizione degli attributi principali per l'analisi del testing.

A.4 Materiale Classe BufferedChannel

A.4.1 Inizializzazione del componente (Costruttore)

Si riporta di seguito la firma del costruttore oggetto di analisi:

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

A.4.2 Identificazione della classi di equivalenza

Parametro	Classi di Equivalenza
allocator	{istanza valida}, {istanza non valida}, {istanza nulla}
fc	{istanza valida}, {istanza read-only}, {istanza write-only}, {istanza chiusa}, {istanza con posizione non valida}, {istanza nulla}
writeCapacity	{ ≤ 0 }, { > 0 }
readCapacity	{ ≤ 0 }, { > 0 }
unpersistedBound	{ < 0 }, { $= 0$ }, { > 0 }

Tabella 13: Classi di equivalenza per i parametri del costruttore di BufferedChannel.

A.4.3 Boundary Value Analysis

Parametro	Valori di Frontiera
allocator	<i>istanza valida</i> : un allocatore che genera dei buffer non nulli. <i>istanza non valida</i> : un allocatore che genera dei buffer nulli. <i>istanza nulla</i> : un'istanza nulla che non permette di generare alcun buffer.
fc	<i>istanza valida</i> : un file channel aperto in modalità lettura/scrittura con posizione valida con file già scritto (per un totale di fcContentLen byte). <i>istanza read-only</i> : come valida, ma aperto solo in modalità lettura. <i>istanza write-only</i> : come valida, ma aperto solo in modalità scrittura. <i>istanza chiusa</i> : come valida, ma il file channel è chiuso. <i>istanza con posizione non valida</i> : un file channel il cui indice ha una posizione negativa non valida. <i>istanza nulla</i> : riferimento null.
writeCapacity	-1, 0, 1, 100 (aggiunto per testare il funzionamento in scenari reali)
readCapacity	-1, 0, 1, 100 (aggiunto per testare il funzionamento in scenari reali)
unpersistedBound	-1, 0, 1

Tabella 14: Boundary Value Analysis per il costruttore di BufferedChannel.

A.4.4 Casi di Test - Costruttore

La Tabella 15 descrive i casi di test per l'inizializzazione della classe `BufferedChannel`.

ID	Allocator	FileChannel	wCap	rCap	UBB	Output Atteso	Esito	Motivazione
T1	Valid	Valid	100	100	1	Valid Instance	Passato	Un allocatore valido alloca normalmente un buffer.
T2	Invalid	Valid	100	100	1	Exception	Fallito	Un allocatore non valido che genera dei buffer nulli.
T3	Null	Valid	100	100	1	Exception	Passato	Un allocatore null non permette di allocare buffer.
T4	Valid	Read Only	100	100	1	Valid Instance	Passato	Un file channel read-only non darà problemi in init, ma durante una write.
T5	Valid	Write Only	100	100	1	Valid Instance	Passato	Un file channel write-only non darà problemi in init, ma durante una read.
T6	Valid	Closed	100	100	1	Exception	Passato	Un file channel chiuso non può essere usato per operazioni di I/O.
T7	Valid	Inv. Pos.	100	100	1	Exception	Fallito	Una posizione non valida può generare anomalie durante l'I/O.
T8	Valid	Null	100	100	1	Exception	Passato	Un file channel null non permette di fare operazioni di I/O.
T9	Valid	Valid	1	100	1	Valid Instance	Passato	Un buffer di scrittura di dimensione 1 è utilizzabile.
T10	Valid	Valid	0	100	1	Valid Instance	Passato	Un buffer di scrittura di dimensione 0 non deve dare problemi in init.
T11	Valid	Valid	-1	100	1	Exception	Passato	Un buffer di scrittura non può avere una dimensione negativa.
T12	Valid	Valid	100	1	1	Valid Instance	Passato	Un buffer di lettura di dimensione 1 è utilizzabile.
T13	Valid	Valid	100	0	1	Valid Instance	Passato	Un buffer di lettura di dimensione 0 non deve dare problemi in init.
T14	Valid	Valid	100	-1	1	Exception	Passato	Un buffer di lettura non può avere una dimensione negativa.
T15	Valid	Valid	100	100	1	Valid Instance	Passato	Un valore di UBB positivo è lecito.
T16	Valid	Valid	100	100	0	Valid Instance	Passato	Un valore di UBB nullo è lecito.
T17	Valid	Valid	100	100	-1	Exception	Fallito	Un valore di UBB negativo non è lecito.

Tabella 15: Mappatura completa dei test per il costruttore di `BufferedChannel`.

A.4.5 Risultati dei test e analisi dei fallimenti

In questa sezione vengono analizzati i casi di test che hanno riportato un esito negativo per la classe `BufferedChannel`.

ID	Metodo	Analisi e Motivazione del Fallimento
T2	Costruttore	L'esito negativo del test evidenzia un'omissione nella logica di validazione del costruttore: alla riga 101, il riferimento restituito dall'allocatore viene assegnato al <code>writeBuffer</code> senza una verifica preventiva dello stato dell'istanza. Sebbene il piano di test preveda il sollevamento di un'eccezione in fase di inizializzazione, l'attuale implementazione differisce la gestione dell'errore alla fase operativa. Si è pertanto optato per il mantenimento di classi di equivalenza distinte, al fine di isolare e analizzare la robustezza del componente durante le successive invocazioni dei metodi <code>write</code> e <code>read</code> .
T7	Costruttore	Il fallimento deriva dalla mancanza di una validazione sulla validità dello stato del <code>FileChannel</code> . L'analisi del codice alla riga 100 mostra che il costruttore si limita a leggere la posizione corrente del canale tramite <code>fc.position()</code> per inizializzare il puntatore interno <code>position</code> , senza verificare se tale valore sia coerente (es. non negativo). Il costruttore accetta quindi il canale delegando la gestione di eventuali stati inconsistenti alle operazioni di I/O successive. Si mantiene la distinzione nelle classi di equivalenza per testare la risposta del sistema durante le fasi di lettura e scrittura su canali con posizioni non valide.
T17	Costruttore	Il fallimento deriva dalla mancanza di una validazione sul parametro <code>unpersistedBytesBound</code> . Tuttavia, l'analisi del codice rivela un'incoerenza: mentre il Javadoc suggerisce che ogni valore non-zero attivi il flush, l'implementazione alla riga 102 (<code>unpersistedBytesBound > 0</code>) disabilita la funzione per qualsiasi valore non positivo. Il valore -1 viene, quindi, accettato ma trattato come lo zero, disattivando i flush regolari. A seguito di questa analisi, le classi di equivalenza per questo parametro sono state accorpate in: $\{\leq 0\}$ (Funzione disattivata) e $\{> 0\}$ (Funzione attiva).

Tabella 16: Analisi dei fallimenti riscontrati durante il testing del costruttore di `BufferedChannel`.

A.4.6 Correzioni a seguito dell'analisi dei fallimenti

Parametro	Classi di Equivalenza
allocator	{istanza valida}, {istanza non valida}, {istanza nulla}
fc	{istanza valida}, {istanza read-only}, {istanza write-only}, {istanza chiusa}, {istanza con posizione non valida}, {istanza nulla}
writeCapacity	{ ≤ 0 }, { > 0 }
readCapacity	{ ≤ 0 }, { > 0 }
unpersistedBound	{ ≤ 0 }, { > 0 }

Tabella 17: Classi di equivalenza per i parametri del costruttore di `BufferedChannel`.

La Tabella 18 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Allocator	FileChannel	wCap	rCap	UBB	Output Atteso	Esito	Motivazione
T17	Valid	Valid	100	100	-1	Valid Instance	Passato	L'oracolo è stato riallineato: l'analisi rivelava l'assenza di una logica <i>fail-fast</i> ; il sistema tollera l'input negativo per UBB disabilitando il flush anziché sollevare l'eccezione inizialmente ipotizzata.

Tabella 18: Casi di test corretti a seguito dell'analisi del codice di `BufferedChannel`.

La Tabella 19 identifica i difetti che non possono essere risolti con una semplice modifica dei test, in quanto rappresentano anomalie logiche o di robustezza del codice sorgente nella classe `BufferedChannel`.

ID	Esito	Descrizione del Bug Riscontrato
T2	Anomalia	L'accettazione di un allocatore invalido non viene rilevata in fase di costruzione alla riga 101. Questo causa la generazione latente di buffer nulli che portano a <code>NullPointerException</code> non gestite durante l'uso del canale.
T7	Anomalia	Alla riga 100, la mancata validazione della posizione del <code>FileChannel</code> permette di istanziare un oggetto con puntatori potenzialmente negativi, delegando la gestione dell'inconsistenza alle operazioni di I/O successive.

Tabella 19: Identificazione di anomalie critiche (bug) nella classe `BufferedChannel`.

A.4.7 Analisi del metodo write

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public void write(ByteBuf src) throws IOException
```

A.4.8 Identificazione delle classi di equivalenza

Parametro	Classi di Equivalenza
src	{istanza valida}, {istanza non valida}, {istanza non allocata}, {istanza vuota}, {istanza nulla}, {istanza write-only}

Tabella 20: Classi di equivalenza per il parametro del metodo `write` di `BufferedChannel`.

A.4.9 Boundary Value Analysis

Parametro	Valori di Frontiera
src*	<p><i>istanza valida</i>: buffer correttamente allocato da cui è possibile leggere dati.</p> <p><i>istanza non valida</i>: buffer con indice di lettura non valido (es. negativo).</p> <p><i>istanza non allocata</i>: buffer deallocated.</p> <p><i>istanza vuota</i>: buffer correttamente allocato ma privo di dati leggibili.</p> <p><i>istanza nulla</i>: riferimento <code>null</code>.</p> <p><i>istanza write-only</i>: buffer accessibile in scrittura ma non in lettura.</p>

Tabella 21: Boundary Value Analysis per il metodo `write` di `BufferedChannel`.

A.4.10 Casi di Test - Metodo Write

La Tabella 22 descrive i casi di test per il metodo `write`.

ID	Allocator	FileChannel	wCap	rCap	UBB	src	Output Atteso	Esito	Motivazione
W1	Invalid	Valid	100	100	1	valid	Exception	Passato	Un allocatore non valido che genera dei buffer nulli.
W2	Valid	Inv. Pos.	100	100	1	valid	Exception	Fallito	Un file channel con posizione invalida.
W3	Valid	Valid	100	100	-1	valid	Exception	Fallito	Un valore UBB negativo non è lecito.
W4	Valid	Read-Only	100	100	1	valid	Exception	Passato	Un file channel accessibile solo in lettura.
W5	Valid	Valid	0	100	1	valid	Exception	Fallito	Un buffer di dimensione nulla.
W6	Valid	Valid	100	100	Len()+1	valid	null	Passato	La scrittura avviene solo nel buffer.
W7	Valid	Valid	100	100	Len()	valid	null	Passato	I dati vengono scaricati nel file channel.
W8	Valid	Valid	100	100	Len()-1	valid	null	Passato	I dati vengono scaricati nel file channel.
W9	Valid	Valid	100	100	0	valid	null	Fallito	Con UBB pari a zero i dati dovrebbero essere scaricati immediatamente nel file channel.
W10	Valid	Valid	100	100	1	invalid	Exception	Passato	Sorgente con indice di lettura non valido.
W11	Valid	Valid	100	100	1	deallocated	Exception	Passato	Sorgente deallocate.
W12	Valid	Valid	100	100	1	empty	null	Passato	Sorgente vuota.
W13	Valid	Valid	100	100	1	null	Exception	Passato	Sorgente nulla.
W14	Valid	Valid	100	100	1	write-only	Exception	Fallito	Sorgente accessibile solo in scrittura.

Tabella 22: Mappatura completa dei casi di test per il metodo `write` di `BufferedChannel`.

⁶In aggiunta all'analisi dei parametri, la progettazione dei test ha considerato scenari operativi basati sull'interazione tra `src` e lo stato interno.

A.4.11 Risultati dei test e analisi dei fallimenti

In questa sezione vengono analizzati i casi di test che hanno riportato un esito negativo per la classe `BufferedChannel`.

ID	Metodo	Analisi e Motivazione del Fallimento
W2	write	L'esame del metodo <code>write</code> conferma che l'operazione di scrittura è puramente logica finché il <code>writeBuffer</code> non risulta saturo. Poiché i dati vengono trasferiti nel buffer di memoria tramite <code>writeBuffer.writeBytes</code> , l'invalidità della posizione del <code>FileChannel</code> rimane latente: il metodo <code>flush()</code> , unico punto di interazione fisica con il canale, non viene invocato se la condizione <code>!writeBuffer.isWritable()</code> non è soddisfatta. Inoltre, l'incremento <code>position += copied</code> avviene senza alcuna validazione di integrità, permettendo la propagazione di un valore negativo ereditato dalla fase di inizializzazione.
W3	write	In linea con l'analisi del costruttore, il valore negativo di <code>unpersistedBytesBound</code> non viene validato durante l'operazione di scrittura. Poiché tale valore imposta <code>doRegularFlushes</code> a <code>false</code> , il metodo <code>write</code> salta completamente il blocco logico di controllo della persistenza. Il sistema accetta quindi il parametro e prosegue l'esecuzione in memoria senza sollevare eccezioni. Si conclude che il caso di test debba essere migliorato aggiornando l'output atteso da <code>Exception</code> a <code>null</code> , classificando i valori negativi come una configurazione valida per la disattivazione del <code>flush</code> .
W5	write	Il fallimento riscontrato è riconducibile a un errore logico di tipo <i>infinite loop</i> , originato dall'accettazione passiva di una <code>writeCapacity</code> pari a zero da parte del costruttore. All'interno del metodo <code>write</code> , l'esecuzione entra nel ciclo <code>while (copied < len)</code> , che dipende dal calcolo della variabile <code>bytesToCopy</code> definita come il valore minimo tra i dati rimanenti e lo spazio disponibile nel buffer (<code>writeBuffer.writableBytes()</code>). Poiché un buffer con capacità nulla restituisce costantemente uno spazio scrivibile pari a zero, la variabile <code>bytesToCopy</code> viene valutata come 0 in ogni iterazione, impedendo di fatto l'incremento dell'accumulatore <code>copied</code> . Di conseguenza, la condizione di uscita del ciclo rimane perennemente soddisfatta, intrappolando il thread in un loop di calcolo infinito, causando il <code>timeout</code> rilevato.
W9	write	Il fallimento del test deriva da un'errata interpretazione della logica di <i>buffering</i> . Si ipotizzava che un valore di <code>unpersistedBytesBound</code> pari a 0 indusse un <code>flush</code> immediato ad ogni scrittura; tuttavia, l'analisi del codice mostra che tale valore disabilita la variabile <code>doRegularFlushes</code> . Il sistema opera quindi esclusivamente in memoria, accettando il valore 0 non come una richiesta di persistenza immediata, ma come un comando di disattivazione del controllo di soglia.
W14	write	Il test fallisce poiché l'implementazione del metodo <code>write</code> adotta un approccio permissivo nei confronti di sorgenti dati prive di byte leggibili. Invece di sollevare l'eccezione attesa, il codice calcola una lunghezza di lettura pari a zero, causando il salto immediato dell'intero ciclo di copia. Il metodo termina correttamente l'esecuzione senza aver effettuato alcuna operazione, rendendo l'esito negativo rispetto alle aspettative di validazione rigorosa.

Tabella 23: Analisi dei fallimenti riscontrati durante il testing del metodo `write` di `BufferedChannel`.

A.4.12 Correzioni a seguito dell'analisi dei fallimenti

La Tabella 24 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Allocator	FileChannel	wCap	rCap	UBB	src	Output Atteso	Esito	Motivazione
W3	Valid	Valid	100	100	-1	empty	null	Passato	Il bound negativo disabilita il flush; test corretto accettando l'assenza di eccezioni.
W9	Valid	Valid	100	100	0	empty	null	Passato	Il valore 0 disabilita il flush automatico; test allineato al comportamento rilevato.
W14	Valid	Valid	100	100	1	w-only	null	Passato	Con 0 byte leggibili, il ciclo viene saltato correttamente senza eccezioni.

Tabella 24: Casi di test corretti a seguito dell'analisi del codice di `BufferedChannel`.

La Tabella 25 identifica i difetti che non possono essere risolti con una semplice modifica dei test, in quanto rappresentano anomalie logiche o di robustezza nel codice sorgente della classe `Buffered Channel`.

ID	Esito	Descrizione del Bug Riscontrato
W2	Anomalia	L'invalidità della posizione del <code>FileChannel</code> rimane latente poiché il metodo <code>flush()</code> viene invocato solo a buffer saturo. L'incremento <code>position += copied</code> avviene senza validazione, propagando valori negativi inconsistenti.
W5	Timeout	L'accettazione di una <code>writeCapacity=0</code> porta il metodo <code>write</code> in uno stato di stallo infinito. Il sistema entra in un loop di calcolo che satura la CPU senza mai avanzare nella scrittura, rappresentando un difetto di robustezza critico.

Tabella 25: Identificazione di anomalie critiche (bug) nella classe `BufferedChannel`.

A.4.13 Analisi del metodo read

Si riporta di seguito la firma del metodo oggetto di analisi:

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

A.4.14 Identificazione delle classi di equivalenza

Parametro	Classi di Equivalenza
dest	{istanza scrivibile}, {istanza non scrivibile}, {istanza con indice invalido}, {istanza deallocata}, {istanza nulla}
(pos, length)	$C_0: \{(pos, len) \mid pos < 0 \vee len < 0\}$ $C_1: \{(pos, len) \mid pos \geq 0 \wedge len \geq 0 \wedge pos + len \leq wBSPosition\}$ $C_2: \{(pos, len) \mid pos \geq 0 \wedge len \geq 0 \wedge pos \geq wBSPosition \wedge pos + len \leq wBSPosition + writerIndex\}$ $C_3: \{(pos, len) \mid pos \geq 0 \wedge len \geq 0 \wedge pos < wBSPosition \wedge pos + len \geq wBSPosition \wedge pos + len \leq wBSPosition + writerIndex\}$ $C_4: \{(pos, len) \mid pos \geq 0 \wedge len \geq 0 \wedge pos < wBSPosition \wedge pos + len > wBSPosition + writerIndex\}$

Tabella 26: Classi di equivalenza per i parametri del metodo `read` di `BufferedChannel`.

A.4.15 Boundary Value Analysis

Parametro	Valori di Frontiera
dest	<i>istanza scrivibile</i> : buffer correttamente allocato con capacità residua sufficiente. <i>istanza non scrivibile</i> : buffer allocato ma privo di spazio. <i>istanza con indice invalido</i> : buffer con puntatore di lettura superiore a quello di scrittura. <i>istanza deallocata</i> : riferimento a un buffer precedentemente rilasciato. <i>istanza nulla</i> : riferimento <code>null</code> .

Tabella 27: Boundary Value Analysis per il buffer di destinazione del metodo `read` di `BufferedChannel`.

Classe	Valori di Frontiera*
C_0	$(-1, 1), (1, -1), (0, 0)$: coppie invalide
C_1	$(0, 1), (0, wBSPosition - 1), (0, wBSPosition), (1, wBSPosition - 1)$: lettura valida solo dal file channel
C_2	$(wBSPosition, 1), (wBSPosition + writerIndex - 1, 1)$: lettura valida solo dal buffer
C_3	$(0, wBSPosition + 1), (0, wBSPosition + writerIndex)$: lettura valida sia dal file channel sia dal buffer
C_4	$(0, wBSPosition + writerIndex + 1), (wBSPosition + writerIndex, 1)$: byte non leggibili dal buffer

Tabella 28: Dettaglio valori di frontiera per la coppia $(pos, length)$ nel metodo `read` di `BufferedChannel`.

*L'analisi della coppia $(pos, length)$ considera lo stato interno del canale: `writeBufferStartPosition` ($wBSPosition$) delimita la lettura tra canale fisico e buffer, mentre `writerIndex` definisce il limite superiore dei byte effettivamente leggibili.

A.4.16 Casi di Test - Metodo Read

La Tabella 29 descrive i casi di test per il metodo `read`.⁸

ID	Allocator	FileChannel	wCap	rCap	UBB	wBuf	dest	pos	len	Out	Esito	Motivazione
R1	invalid	valid	100	100	1	null	empty	0	fCL	Exc.	Fallito	Un allocatore non valido che genera dei buffer nulli.
R2	valid	invalid	100	100	1	null	empty	0	fCL	Exc.	Passato	Un file channel con posizione invalida.
R3	valid	w-only	100	100	1	null	empty	0	fCL	Exc.	Passato	Un file channel accessibile solo in scrittura.
R4	valid	valid	100	0	fCL	null	empty	0	1	Exc.	Passato	Un buffer di lettura di dimensione nulla.
R5	valid	valid	100	100	1	null	empty	0	fCL	null	Passato	La destinazione in cui scrivere i dati è vuota.
R6	valid	valid	100	100	1	null	semi-full	0	fCL/2	null	Passato	La destinazione in cui scrivere i dati è parzialmente occupata.
R7	valid	valid	100	100	1	null	full	0	fCL	Exc.	Passato	La destinazione in cui scrivere i dati è satura.
R8	valid	valid	100	100	1	null	inv. idx	0	fCL	Exc.	Passato	La destinazione in cui scrivere ha un indice non valido.
R9	valid	valid	100	100	1	null	dealloc.	0	fcl	Exc.	Passato	La destinazione in cui scrivere è deallocated.
R10	valid	valid	100	100	1	null	null	0	fcl	Exc.	Passato	La destinazione in cui scrivere è nulla.
R11	valid	valid	100	100	1	wbCL	empty	-1	1	Exc.	Passato	Una posizione negativa non è lecita.
R12	valid	valid	100	100	1	wbCL	empty	1	-1	Exc.	Fallito	Una lunghezza negativa non è lecita.
R13	valid	valid	100	100	1	wbCL	empty	0	0	null	Passato	Una lunghezza nulla è lecita.
R14	valid	valid	100	100	1	wbCL	empty	0	1	null	Passato	Lettura del primo byte dal FileChannel.
R15	valid	valid	100	100	1	wbCL	empty	0	fCL-1	null	Passato	Lettura parziale del contenuto del file channel.
R16	valid	valid	100	100	1	wbCL	empty	0	fCL	null	Passato	Lettura completa del contenuto del file channel.
R17	valid	valid	100	100	1	wbCL	empty	1	fCL-1	null	Passato	Lettura a partire dal secondo byte del file channel.
R18	valid	valid	100	100	1	wbCL	empty	fCL	1	null	Passato	Lettura a partire dal primo byte del write buffer.
R19	valid	valid	100	100	1	wbCL	empty	fCL+wb-1	1	null	Passato	Lettura dell'ultimo byte dal write buffer.
R20	valid	valid	100	100	1	wbCL	empty	0	fCL+1	null	Passato	Lettura totale del file channel e del primo byte write buffer.
R21	valid	valid	100	100	1	wbCL	empty	0	fCL+wb	null	Passato	Lettura totale del file channel e del write buffer.
R22	valid	valid	100	100	1	wbCL	empty	0	fCL+wb+1	Exc.	Passato	Lettura oltre il limite massimo consentito.
R23	valid	valid	100	100	1	wbCL	empty	fCL+wb	1	Exc.	Passato	Lettura del primo byte non consentito.

Tabella 29: Mappatura completa dei casi di test per il metodo `read` di `BufferedChannel`.

⁸Per ragioni di sintesi nella documentazione, i risultati attesi sono stati riportati in forma semplificata; tuttavia, la suite di test verifica direttamente che il metodo restituisca il numero corretto di byte letti e non un valore nullo, validando l'intero stato dei buffer coinvolti.

A.4.17 Risultati dei test e analisi dei fallimenti

In questa sezione vengono analizzati i casi di test che hanno riportato un esito negativo per la classe `BufferedChannel`.

ID	Metodo	Analisi e Motivazione del Fallimento
R1	read	Il fallimento evidenzia una discrepanza tra la configurazione del componente e la sua reale gerarchia interna. Sebbene venga passato un allocator invalido, l'analisi del codice rivela che il <code>readBuffer</code> non è gestito direttamente da <code>BufferedChannel</code> , bensì è encapsulato nella superclasse <code>BufferedReadChannel</code> . Poiché l'istanziazione di quest'ultimo avviene in uno strato superiore non influenzato dall'allocatore della componente di scrittura, il buffer di lettura rimane valido. L'assunzione del test non è, quindi, corretta a causa di questa segregazione totale delle risorse.
R12	read	Si riscontra un'assenza di validazione sul parametro <code>length</code> . L'attuale implementazione del ciclo <code>while (length > 0)</code> non gestisce esplicitamente i valori negativi, trattandoli come condizione di terminazione immediata. Di conseguenza, la classe di equivalenza originale è stata scissa per isolare i casi di "lettura nulla" ($len \leq 0$) dagli errori di posizione ($pos < 0$). Il metodo restituisce 0 anziché sollevare la <code>IllegalArgumentException</code> attesa per una lunghezza negativa, portando a un'uscita corretta ma non conforme all'ipotesi di test.

Tabella 30: Analisi dei fallimenti riscontrati durante il testing del metodo `read` di `BufferedChannel`.

A.4.18 Correzioni a seguito dell'analisi dei fallimenti

Parametro	Classi di Equivalenza
dest	{istanza scrivibile}, {istanza non scrivibile}, {istanza con indice invalido}, {istanza deallocata}, {istanza nulla}
(pos, length)	$C0_inv: \{(pos, len) \mid pos < 0\}$ (Posizione invalida). $C0_null: \{(pos, len) \mid pos \geq 0 \wedge len \leq 0\}$ (Lettura nulla - Uscita immediata). $C1: \{(pos, len) \mid pos \geq 0 \wedge len > 0 \wedge pos + len \leq wBSPosition\}$ $C2: \{(pos, len) \mid pos \geq 0 \wedge len > 0 \wedge pos \geq wBSPosition \wedge pos + len \leq wBSPosition + writerIndex\}$ $C3: \{(pos, len) \mid pos \geq 0 \wedge len > 0 \wedge pos < wBSPosition \wedge pos + len \geq wBSPosition \wedge pos + len \leq wBSPosition + writerIndex\}$ $C4: \{(pos, len) \mid pos \geq 0 \wedge len > 0 \wedge pos < wBSPosition \wedge pos + len > wBSPosition + writerIndex\}$

Tabella 31: Classi di equivalenza per i parametri del metodo `read` di `BufferedChannel`.

La Tabella 32 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Allocator	FileChannel	wCap	rCap	UBB	wBuf	dest	pos	len	Out	Esito	Motivazione
R1	invalid	Valid	100	100	1	null	empty	0	fCL	null	Passato	L'allocatore non influenza sul buffer di lettura gestito dalla superclasse.
R12	Valid	Valid	100	100	1	wbCL	empty	1	-1	null	Passato	La lunghezza negativa viene neutralizzata portando a un'uscita senza eccezioni.

Tabella 32: Casi di test corretti a seguito dell'analisi del codice di `BufferedChannel`.

A.5 Materiale Classe WriteCache

A.5.1 Inizializzazione del componente (Costruttore)

Si riporta di seguito la firma del costruttore oggetto di analisi:

```
1 public WriteCache(ByteBufAllocator allocator, long maxCacheSize, int maxSegmentSize)
```

A.5.2 Identificazione della classi di equivalenza

Parametro	Classi di Equivalenza
allocator	{istanza valida}, {istanza non valida}, {istanza nulla}
maxCacheSize	{ ≤ 0 }, { > 0 }
maxSegmentSize	{ $s \mid s \leq 0$ }, { $s \mid s > 0 \text{ e } s = 2^n \text{ e } s \leq maxCacheSize$ }, { $s \mid s > 0 \text{ e } s \neq 2^n \text{ e } s \leq maxCacheSize$ }, { $s \mid s > maxCacheSize$ }

Tabella 33: Classi di equivalenza per i parametri del costruttore di WriteCache.

A.5.3 Boundary Value Analysis

Parametro	Valori di Frontiera
allocator	<i>istanza valida</i> : un allocatore che genera dei buffer non nulli. <i>istanza non valida</i> : un allocatore che genera dei buffer nulli. <i>istanza nulla</i> : riferimento <code>null</code> .
maxCacheSize	-1, 0, 1, 512 (aggiunto per testare il funzionamento in scenari reali).
maxSegmentSize	-1, 0, 1, 100, 128 (potenza di 2), 512 (<code>limite maxCacheSize</code>), 513 ($> maxCacheSize$).

Tabella 34: Boundary Value Analysis per il costruttore di WriteCache.

A.5.4 Casi di Test - Costruttore

La Tabella 35 descrive i casi di test per l'inizializzazione della classe WriteCache.

ID	Allocator	maxCS	maxSS	Output Atteso	Esito	Motivazione
T1	Valid	512	128	Valid Instance	Passato	Un allocatore valido alloca normalmente un buffer.
T2	Invalid	512	128	Exception	Fallito	Un allocatore non valido che genera dei buffer nulli.
T3	Null	512	128	Exception	Fallito	Un allocatore null non permette di allocare alcun buffer.
T4	Valid	1	1	Valid Instance	Passato	La dimensione massima della cache è valida.
T5	Valid	0	1	Exception	Fallito	La dimensione massima della cache non è valida.
T6	Valid	-1	1	Exception	Passato	La dimensione massima della cache non è valida.
T7	Valid	512	-1	Exception	Passato	La dimensione massima del segmento non è valida.
T8	Valid	512	0	Exception	Passato	La dimensione massima del segmento non è valida.
T9	Valid	512	1	Valid Instance	Passato	La dimensione del segmento è valida in quanto potenza di 2 e $\leq maxCacheSize$.
T10	Valid	512	100	Exception	Passato	La dimensione del segmento non è valida in quanto non è una potenza di 2.
T11	Valid	512	128	Valid Instance	Passato	La dimensione del segmento è valida in quanto potenza di 2 e $\leq maxCacheSize$.
T12	Valid	512	512	Valid Instance	Passato	La dimensione del segmento è valida in quanto potenza di 2 e $\leq maxCacheSize$.
T13	Valid	512	513	Exception	Passato	La dimensione del segmento non è valida in quanto $> maxCacheSize$.

Tabella 35: Mappatura completa dei casi di test per il costruttore di WriteCache.

A.5.5 Risultati dei test e analisi dei fallimenti

In questa sezione vengono analizzati i casi di test che hanno riportato un esito negativo per la classe WriteCache.

ID	Metodo	Analisi e Motivazione del Fallimento
T2	Costruttore	Il fallimento conferma che il costruttore non convalida il parametro allocator, limitandosi ad assegnarne il riferimento. Sebbene il test si aspetti un'eccezione immediata per un allocatore invalido, il codice accetta l'istanza rimandando la gestione dell'errore alla fase operativa.
T3	Costruttore	Analogamente al caso precedente, il test fallisce in quanto il sistema non valida esplicitamente la presenza di un allocatore nullo durante la fase di inizializzazione.
T5	Costruttore	Il fallimento è riconducibile all'accettazione del valore 0 per maxCacheSize. L'implementazione non prevede clausole di salvaguardia, procedendo a un'inizializzazione che diverge dal comportamento difensivo ipotizzato. L'analisi rivela che il sistema gestisce lo 0 come un limite operativo che garantisce comunque l'allocazione di un segmento minimo.

Tabella 36: Analisi dei fallimenti riscontrati durante il testing del costruttore di WriteCache.

A.5.6 Correzioni a seguito dell'analisi dei fallimenti

Parametro	Classi di Equivalenza Aggiornate
allocator	{istanza valida}, {istanza non valida}, {istanza nulla}
maxCacheSize	{< 0} (Valore non ammesso) {= 0} (Capacità minima - Normalizzazione a 1 segmento) {> 0} (Range operativo standard)
maxSegmentSize	{ $s \mid s \leq 0$ }, { $s \mid s > 0 \text{ e } s = 2^n \text{ e } s \leq maxCacheSize$ }, { $s \mid s > 0 \text{ e } s \neq 2^n \text{ e } s \leq maxCacheSize$ }, { $s \mid s > maxCacheSize$ }

Tabella 37: Classi di equivalenza per i parametri del costruttore di WriteCache.

La Tabella 38 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Allocator	maxCS	maxSS	Output Atteso	Esito	Motivazione
T5	Valid	0	1	Valid Instance	Passato	L'algoritmo di calcolo garantisce un'allocazione minima anche per capacità nulla; lo 0 è stato riclassificato come valore limite della specifica.

Tabella 38: Casi di test corretti a seguito dell'analisi del codice di WriteCache.

La Tabella 39 identifica i difetti che non possono essere risolti con una semplice modifica dei test, in quanto rappresentano anomalie logiche o di robustezza nel codice sorgente della classe BufferedChannel.

ID	Esito	Descrizione del Bug Riscontrato
T2	Anomalia	Omissione nella validazione alla riga 101: il riferimento dell'allocatore viene assegnato senza verifica. Se l'allocatore è invalido, l'errore rimane latente fino alla fase operativa (I/O).
T3	Anomalia	Mancato controllo sul parametro Allocator nullo. Il sistema permette l'inizializzazione di un'istanza priva di buffer interni, causando fallimenti critici non gestiti durante l'uso del canale.

Tabella 39: Identificazione di anomalie critiche (bug) nella classe di BufferedChannel.

A.5.7 Analisi del metodo put

Si riporta di seguito la firma del metodo oggetto di analisi:

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

A.5.8 Identificazione delle classi di equivalenza

Parametro	Classi di Equivalenza
ledgerId	{< 0}, {≥ 0}
entryId	$E_0: \{entryId < 0\}$ (non valido) $E_1: \{0 \leq entryId < lastEntryId\}$ (fuori ordine) $E_2: \{entryId = lastEntryId\}$ (duplicato) $E_3: \{entryId > lastEntryId\}$ (nuova entry)
entry*	{istanza vuota}, {istanza singolo segmento}, {istanza multi-segmento}, {istanza con indice invalido}, {istanza deallocate}, {istanza nulla}

Tabella 40: Classi di equivalenza per i parametri del metodo put di WriteCache.

A.5.9 Boundary Value Analysis

Parametro	Valori di Frontiera
ledgerId	-1, 0, 1
entryId	-1, 0, 1, 2 (Assumendo lastEntryId = 1 come riferimento di frontiera).
entry	<i>istanza vuota</i> : buffer allocato con nessun byte leggibile. <i>istanza singolo segmento</i> : buffer con dimensione $\leq \text{maxSegmentSize}$. <i>istanza multi-segmento</i> : buffer con dimensione $> \text{maxSegmentSize}$. <i>istanza con indice invalido</i> : buffer con stato inconsistente ($readerIndex > writerIndex$). <i>istanza deallocate</i> : riferimento a un oggetto con <code>refCnt = 0</code> (già rilasciato). <i>istanza nulla</i> : riferimento <code>null</code> .

Tabella 41: Boundary Value Analysis per i parametri del metodo put di WriteCache.

⁹L'analisi di entry integra il vincolo `maxSegmentSize`, discriminante per determinare se il payload richieda un'allocazione contigua o ecceda la capacità del segmento. Tale valutazione condiziona il valore booleano di ritorno e la conseguente mutazione dello stato interno.

A.5.10 Casi di Test - Metodo put

La Tabella 42 descrive i casi di test per il metodo put.

ID	Allocator	maxCS	maxSS	writeCacheType	LID	EID	Entry	Output Atteso	Esito	Motivazione
P1	invalid	512	256	NON_WRITTEN	1	2	full	Exception	Fallito	Un allocatore null non permette di generare buffer.
P2	null	512	256	NON_WRITTEN	1	2	full	Exception	Fallito	L'allocatore null impedisce l'allocazione.
P3	valid	0	1	NON_WRITTEN	1	2	full	FALSE	Passato	Caso in cui la cache ha dimensione nulla.
P4	valid	512	512	NON_WRITTEN	1	2	full	TRUE	Passato	Inserimento in un segmento completamente libero.
P5	valid	512	512	HALF_SEG	1	2	full	TRUE	Passato	Inserimento in un segmento parzialmente libero, ma il cui spazio residuo è sufficiente.
P6	valid	512	512	FULL_SEG	1	2	full	FALSE	Passato	Inserimento in un segmento completamente occupato.
P7	valid	512	256	NON_WRITTEN	-1	2	full	Exception	Passato	Un ledgerId negativo non è lecito.
P8	valid	512	256	NON_WRITTEN	0	2	full	TRUE	Passato	Un ledgerId nullo è lecito.
P9	valid	512	256	NON_WRITTEN	1	2	full	TRUE	Passato	Un ledgerId positivo è lecito.
P10	valid	512	256	NON_WRITTEN	1	-1	full	Exception	Passato	Un entryId negativo non è lecito.
P11	valid	512	256	NON_WRITTEN	1	0	full	TRUE	Passato	Un entryId nullo è lecito.
P12	valid	512	256	NON_WRITTEN	1	1	full	TRUE	Passato	Un entryId positivo è lecito.
P13	valid	512	256	NON_WRITTEN	1	2	full	TRUE	Passato	Un entryId positivo è lecito.
P14	valid	512	256	NON_WRITTEN	1	2	empty	TRUE	Passato	Inserimento di un entry vuota.
P15	valid	512	256	NON_WRITTEN	1	2	size ≤ mSS	TRUE	Passato	Inserimento di un entry che entra esattamente in un segmento.
P16	valid	512	256	NON_WRITTEN	1	2	size > mSS	FALSE	Passato	Inserimento di un entry che ha una dimensione maggiore del segmento.
P17	valid	512	256	NON_WRITTEN	1	2	inv. index	Exception.	Passato	Inserimento di un entry con un indice di lettura non valido.
P18	valid	512	256	NON_WRITTEN	1	2	dealloc.	Exception	Passato	Inserimento di un entry deallocated.
P19	valid	512	256	NON_WRITTEN	1	2	null	Exc.	Passato	Inserimento di un entry null.

Tabella 42: Mappatura completa dei casi di test per il metodo put della classe WriteCache.

A.5.11 Risultati dei test e analisi dei fallimenti

In questa sezione vengono analizzati i casi di test che hanno riportato un esito negativo per la classe WriteCache.

ID	Metodo	Analisi e Motivazione del Fallimento
P1	put	Il fallimento deriva da un'errata assunzione circa il ruolo del ByteBufAllocator nella put. Sebbene si ipotizzasse un'eccezione a fronte di un allocatore invalido, l'analisi rivela che il metodo non interagisce con l'allocatore per la persistenza delle entry. La logica di scrittura si avvale esclusivamente dell'array cacheSegments composto da buffer preallocati in fase di costruzione.
P2	put	Analogamente al caso precedente, il fallimento deriva dall'aspettativa di una validazione sul parametro allocator nullo, che tuttavia non avviene poiché l'architettura del metodo put non prevede nuove allocazioni dinamiche durante l'inserimento.

Tabella 43: Analisi dei fallimenti riscontrati durante il testing del metodo put di WriteCache.

A.5.12 Correzioni a seguito dell'analisi dei fallimenti

La Tabella 44 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Allocator	maxCS	maxSS	writeCacheType	LID	EID	Entry	Out	Esito	Motivazione Corretta
P1	invalid	512	256	NON_W	1	2	full	TRUE	Passato	L'allocatore non è usato; i dati sono scritti nei buffer preallocati (cacheSegments).
P2	null	512	256	NON_W	1	2	full	TRUE	Passato	L'assenza di allocazione dinamica rende l'allocatore ininfluente nella fase di put.

Tabella 44: Casi di test corretti a seguito dell'analisi del codice di WriteCache.

A.5.13 Analisi del metodo get

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public ByteBuf get(long ledgerId, long entryId)
```

A.5.14 Identificazione delle classi di equivalenza

Parametro	Classi di Equivalenza
(<code>ledgerId</code> , <code>entryId</code>)*	$C_{inv} : \{(l, e) \mid l < 0 \vee e < 0\}$ (Almeno un identificativo invalido) $C_{miss} : \{(l, e) \mid l \geq 0 \wedge e \geq 0 \wedge (l, e) \notin cache\}$ (Chiave non presente) $C_{hit} : \{(l, e) \mid l \geq 0 \wedge e \geq 0 \wedge (l, e) \in cache\}$ (Chiave presente)

Tabella 45: Classi di equivalenza per i parametri del metodo `get` di `WriteCache`.

A.5.15 Boundary Value Analysis

Parametro	Valori di Frontiera
(<code>ledgerId</code> , <code>entryId</code>)	$(-1, 1), (1, -1), (0, 1), (1, 0), (0, 0)$ (10, 21) (Esempio di frontiera operativa per chiave non presente - $C1_{miss}$) (10, 20) (Esempio di frontiera operativa per chiave presente - $C2_{hit}$)

Tabella 46: Boundary Value Analysis per il metodo `get` di `WriteCache`.

A.5.16 Casi di Test - Metodo get

La Tabella 47 descrive i casi di test per il metodo `get`.

ID	Allocator	maxCS	maxSS	LID	EID	writeCacheType	Output Atteso	Esito	Motivazione
G1	Invalid	512	128	1	1	NON_W	Exception	Fallito	Un allocatore non valido genera dei buffer nulli.
G2	Null	512	128	1	1	NON_W	Exception	Fallito	Un allocatore nullo non permette di generare alcun buffer.
G3	Valid	0	1	1	1	NON_W	null	Passato	Una cache vuota con una dimensione nulla non permette di recuperare alcun'entry.
G4	Valid	512	128	-1	1	NON_W	Exception	Passato	Un ledgerId negativo non è lecito.
G5	Valid	512	128	0	1	NON_W	null	Passato	Un ledgerId nullo non è lecito.
G6	Valid	512	128	0	0	NON_W	null	Passato	Un ledgerId positivo è lecito.
G7	Valid	512	128	1	-1	NON_W	Exception	Fallito	Un entryId negativo non è lecito.
G8	Valid	512	128	1	0	NON_W	null	Passato	Un entryId nullo è lecito.
G9	Valid	512	128	1	1	NON_W	valid	Passato	Un entryId positivo è lecito.
G10	Valid	512	128	10	20	WRITTEN	valid	Passato	Recupero di un'entry presente nella cache.
G11	Valid	512	128	10	21	WRITTEN	null	Passato	Recupero di un'entry non presente nella cache.

Tabella 47: Mappatura completa dei casi di test per il metodo `get` di `WriteCache`.

¹⁰ A seguito dell'analisi strutturale, si è scelto di accorpare i parametri `ledgerId` ed `entryId` in una singola coppia logica identificativa. Tale approccio permette di definire classi di equivalenza basate non solo sulla validità sintattica degli ID, ma anche sullo stato operativo della risorsa all'interno della `WriteCache`.

A.5.17 Risultati dei test e analisi dei fallimenti

ID	Metodo	Analisi e Motivazione del Fallimento
G1	get	Il fallimento evidenzia l'assenza di una validazione preventiva sull'allocatore. Nello stato NON_WRITTEN, il metodo termina con un <code>return null</code> senza mai invocare la risorsa invalida. Per verificare la gestione degli errori, è necessaria una revisione forzando lo stato a WRITTEN. Tale modifica garantisce che il flusso raggiunga l'istruzione <code>allocator.buffer()</code> , permettendo di validare il comportamento quando la dipendenza corrotta diventa operativa.
G2	get	Analogamente al caso precedente, il fallimento è riconducibile alla mancata interazione con l'allocatore nullo nello stato NON_WRITTEN. L'esecuzione non raggiunge l'istruzione critica poiché il metodo restituisce <code>null</code> non appena l'indice non rileva l'entry. Si rende necessario impostare lo stato su WRITTEN per forzare il recupero di un'entry esistente e validare la robustezza del metodo.
G7	get	Il fallimento rivela un'asimmetria nella gestione degli input negativi: mentre per il <code>ledgerId</code> la struttura dati (<code>index</code>) solleva un'eccezione, per l'id della entry il metodo si limita a restituire <code>null</code> , interpretando il valore negativo come chiave non presente anziché come errore sintattico. Essendo un comportamento radicato nella logica dell'indice, si è proceduto alla revisione modificando l'output atteso da <code>Exception a null</code> .

Tabella 48: Analisi dei fallimenti riscontrati durante il testing del metodo `get` di `WriteCache`.

A.5.18 Correzioni a seguito dell'analisi dei fallimenti

Parametro	Classi di Equivalenza
(<code>ledgerId</code> , <code>entryId</code>)	$C_{inv} : \{(l, e) \mid l < 0\}$ (Identificativo <code>ledgerId</code> non valido) $C_{miss} : \{(l, e) \mid l \geq 0 \wedge (l, e) \notin cache\}$ (Chiave non presente) $C_{hit} : \{(l, e) \mid l \geq 0 \wedge (l, e) \in cache\}$ (Chiave presente)

Tabella 49: Classi di equivalenza per i parametri del metodo `get` di `WriteCache`.

La Tabella 50 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Alloc.	maxCS	maxSS	LID	EID	Stato	Output Atteso	Esito	Motivazione Corretta
G1	invalid	512	128	1	1	WRITTEN	Exception	Passato	Forzando lo stato a WRITTEN si garantisce l'invocazione dell'allocatore, validando il fallimento della risorsa.
G2	null	512	128	1	1	WRITTEN	Exception	Passato	Il riallineamento a WRITTEN fa emergere l'eccezione dell'allocatore nullo, prima mascherata dal <code>cache miss</code> .
G7	valid	512	128	1	-1	NON_W	null	Passato	L'indice restituisce <code>null</code> per EID negativi senza sollevare eccezioni; il test è stato riallineato alla specifica reale.

Tabella 50: Casi di test corretti a seguito dell'analisi del codice di `WriteCache`.

A.6 Materiale Classe ClassUtil

A.6.1 Analisi del metodo getClassName

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public static String getClassName(Class cls)
```

A.6.2 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
cls	{istanza valida}, {istanza nulla}

Tabella 51: Classi di equivalenza per i parametri del metodo getClassName di ClassUtil.

A.6.3 Boundary Value Analysis

Parametro	Valori di Frontiera
cls	<i>istanza valida</i> : riferimento ad un oggetto di tipo Class correttamente inizializzato. <i>istanza nulla</i> : riferimento null.

Tabella 52: Boundary Value Analysis per il metodo getClassName di ClassUtil.

A.6.4 Casi di Test - Metodo getClassName

ID	Classe (cls)	Output Atteso	Esito	Motivazione
T1	valid instance	string	Passato	Il parametro fornito è una classe valida; il metodo deve restituire correttamente il nome della classe sotto forma di stringa.
T2	null	null	Passato	Il parametro fornito è nullo; il metodo deve gestire correttamente l'assenza di un'istanza restituendo il valore null.

Tabella 53: Mappatura completa dei casi di test per il metodo getClassName della classe ClassUtil.

A.6.5 Analisi del metodo getPackageName

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public static String getPackageName(Class cls)
```

A.6.6 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
cls	{istanza valida con package}, {istanza valida senza package}, {istanza nulla}

Tabella 54: Classi di equivalenza per i parametri del metodo getPackageName di ClassUtil.

A.6.7 Boundary Value Analysis

Parametro	Valori di Frontiera
cls	<i>istanza valida con package</i> : riferimento a un oggetto di tipo Class appartenente a un package definito (es. java.lang.String). <i>istanza valida senza package</i> : riferimento a una classe definita nel default package o un tipo primitivo (es. int.class). <i>istanza nulla</i> : riferimento null.

Tabella 55: Boundary Value Analysis per il metodo getPackageName di ClassUtil.

A.6.8 Casi di Test - Metodo getPackageName

ID	Classe (cls)	Output Atteso	Esito	Motivazione
T1	valid instance with package	string	Passato	Classe valida associata a un package.
T2	valid instance without package	""	Passato	Classe valida ma non appartente a nessun package.
T3	null	null	Passato	Riferimento null.

Tabella 56: Mappatura completa dei casi di test per il metodo getPackageName della classe ClassUtil.

A.6.9 Analisi del metodo toClass

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public static Class toClass(String str, boolean resolve, ClassLoader loader)
```

A.6.10 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
str	{stringa valida classe interna}, {stringa valida classe con package}, {stringa valida classe senza package}, {stringa valida classe array}, {stringa vuota}, {stringa non valida}, {stringa nulla}
resolve	{true}, {false}
loader	{istanza valida}, {istanza non valida}, {istanza nulla}

Tabella 57: Classi di equivalenza per i parametri del metodo toClass di ClassUtil.

A.6.11 Boundary Value Analysis

Parametro	Valori di Frontiera
str	<p><i>Stringa valida classe interna</i>: stringa valida che rappresenta una classe definita internamente ad un'altra.</p> <p><i>Stringa valida classe con package</i>: stringa valida che rappresenta una classe standard comprensiva di package (es. "java.lang.String").</p> <p><i>Stringa valida classe senza package</i>: stringa valida che rappresenta una classe definita nel default package.</p> <p><i>Stringa valida classe array</i>: stringa valida che rappresenta una classe array, mono o multidimensionale (es. "int[]", "java.lang.Object[][]").</p> <p><i>Stringa vuota</i>: stringa che non contiene alcun carattere.</p> <p><i>Stringa non valida</i>: stringa contenente caratteri non ammessi per nomi binari Java validi.</p> <p><i>Stringa nulla</i>: null.</p>
resolve	<p><i>true</i>: la classe viene inizializzata all'atto del caricamento.</p> <p><i>false</i>: la classe viene caricata ma non inizializzata.</p>
loader	<p><i>Istanza valida</i>: ClassLoader in grado di risolvere e caricare la classe specificata.</p> <p><i>Istanza non valida</i>: ClassLoader che non ha accesso al percorso della classe o non può carregarla.</p> <p><i>Istanza nulla</i>: viene utilizzato il ClassLoader di default (context loader del thread corrente).</p>

Tabella 58: Boundary Value Analysis per il metodo toClass di ClassUtil.

A.6.12 Casi di Test - Metodo toClass

ID	Stringa (str)	Resolve	Loader	Output atteso	Esito	Motivazione
T1	Valid inner class	FALSE	Valid	Class	Passato	Stringa valida che rappresenta una classe interna.
T2	Valid with package	FALSE	Valid	Class	Passato	Stringa valida che rappresenta una classe con package.
T3	Valid w/o package	FALSE	Valid	Class	Passato	Stringa valida che rappresenta una classe senza package.
T4	Valid array class	FALSE	Valid	Class	Passato	Stringa valida che rappresenta una classe array.
T5	Empty string	FALSE	Valid	Exception	Passato	Una stringa vuota deve generare un'eccezione.
T6	Invalid format	FALSE	Valid	Exception	Passato	Stringa non valida.
T7	Null string	FALSE	Valid	Exception	Passato	Stringa nulla.
T8	Valid inner class	TRUE	Valid	Class	Passato	Inizializzazione impostata.
T9	Valid inner class	TRUE	Invalid	Exception	Passato	Utilizzo di un ClassLoader non valido.
T10	Valid inner class	TRUE	Null	Class	Passato	Utilizzo di un ClassLoader nullo.

Tabella 59: Mappatura completa dei casi di test per il metodo toClass della classe ClassUtil.

A.6.13 Analisi del metodo `getClassName`

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public static String getClassName(String fullName)
```

A.6.14 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
fullName	{stringa valida classe interna}, {stringa valida classe con package}, {stringa valida classe senza package}, {stringa valida classe array}, {stringa vuota}, {stringa non valida}, {stringa nulla}

Tabella 60: Classi di equivalenza per i parametri del metodo `getClassName` di `ClassUtil`.

A.6.15 Boundary Value Analysis

Parametro	Valori di Frontiera
fullName	<p><i>Stringa valida classe interna</i>: stringa valida che rappresenta una classe definita dentro un'altra (es. <code>java.util.Map\$Entry</code>).</p> <p><i>Stringa valida classe con package</i>: stringa che rappresenta una classe standard con package (es. <code>java.lang.String</code>).</p> <p><i>Stringa valida classe senza package</i>: stringa che rappresenta una classe nel default package (es. <code>MyClass</code>).</p> <p><i>Stringa valida classe array</i>: stringa che rappresenta una classe array (es. <code>[I</code> per <code>int[]</code> o <code>[Ljava.lang.Object;</code>).</p> <p><i>Stringa vuota</i>: stringa senza alcun carattere.</p> <p><i>Stringa non valida</i>: stringa con caratteri non consentiti per i nomi binari validi.</p> <p><i>Stringa nulla</i>: null.</p>

Tabella 61: Boundary Value Analysis per il metodo `getClassName` di `ClassUtil`.

A.6.16 Casi di Test - Metodo `getClassName (String)`

ID	Stringa (fullName)	Output Atteso	Esito	Motivazione
T1	valid inner class	valid string	Passato	Stringa valida che rappresenta una classe interna.
T2	valid normal class with package	valid string	Passato	Stringa valida che rappresenta una classe con package.
T3	valid normal class without package	valid string	Passato	Stringa valida che rappresenta una classe senza package.
T4	valid array class	valid string	Passato	Stringa valida che rappresenta una classe con array.
T5	empty	empty	Passato	Stringa vuota.
T6	invalid	Exception	Fallito	Stringa non valida.
T7	null	null	Passato	Stringa nulla.

Tabella 62: Mappatura completa dei casi di test per il metodo `getClassName (String)` della classe `ClassUtil`.

A.6.17 Risultati dei test e analisi dei fallimenti

ID	Metodo	Analisi e Motivazione del Fallimento
T6	<code>getClassName</code>	Il fallimento evidenzia un deficit di robustezza: l'implementazione individua l'ultimo punto(.) e restituisce la sottostringa successiva senza sanificazione o validazione sintattica. Di conseguenza, stringhe contenenti metacaratteri non ammessi dalla JVM (es. "binary"+") non innescano eccezioni ma producono output privi di validità semantica. Il componente delega interamente la responsabilità della validazione al chiamante, venendo meno ai criteri di resilienza attesi per i casi di frontiera.

Tabella 63: Analisi dei fallimenti riscontrati durante il testing del metodo `getClassName (String)` di `ClassUtil`.

A.6.18 Correzioni a seguito dell'analisi dei fallimenti

La Tabella 64 identifica i difetti che non possono essere risolti con una semplice modifica dei test, in quanto rappresentano anomalie logiche o di robustezza nel codice sorgente della classe `ClassUtil`.

ID	Esito	Descrizione del Bug Riscontrato
T6	Anomalia	Il fallimento evidenzia un deficit di robustezza: l'implementazione individua l'ultimo punto(.) e restituisce la sottostringa successiva senza sanitizzazione o validazione sintattica. Di conseguenza, stringhe contenenti metacaratteri non ammessi dalla JVM non innescano eccezioni ma producono output privi di validità semantica, delegando interamente la responsabilità della validazione al chiamante.

Tabella 64: Identificazione di anomalie critiche (bug) nel metodo `getClassName(String)` di `ClassUtil`.

A.6.19 Analisi del metodo `getPackageName`

```
1 public static String getPackageName(String fullName)
```

A.6.20 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
fullName	{stringa valida classe interna}, {stringa valida classe con package}, {stringa valida classe senza package}, {stringa valida classe array}, {stringa vuota}, {stringa non valida}, {stringa nulla}

Tabella 65: Classi di equivalenza per i parametri del metodo `getPackageName` di `ClassUtil`.

A.6.21 Analisi boundary

Parametro	Valori di Frontiera
fullName	<p><i>Stringa valida classe interna</i>: stringa valida che rappresenta una classe definita internamente ad un'altra.</p> <p><i>Stringa valida classe con package</i>: stringa valida che rappresenta una classe standard comprensiva di package (es. "java.lang.String").</p> <p><i>Stringa valida classe senza package</i>: stringa valida che rappresenta una classe nel default package; il metodo dovrebbe restituire una stringa vuota.</p> <p><i>Stringa valida classe array</i>: stringa valida che rappresenta una classe array (es. "[Ljava.lang.Object;").</p> <p><i>Stringa vuota</i>: stringa che non contiene alcun carattere.</p> <p><i>Stringa non valida</i>: stringa contenente caratteri non consentiti nei nomi binari validi.</p> <p><i>Stringa nulla</i>: null.</p>

Tabella 66: Boundary Value Analysis per il metodo `getPackageName` di `ClassUtil`.

A.6.22 Casi di Test - Metodo `getPackageName(String)`

ID	Stringa (fullName)	Output Atteso	Esito	Motivazione
T1	valid inner class	valid string	Passato	Stringa valida che rappresenta una classe interna.
T2	valid normal class with package	valid string	Passato	Stringa valida che rappresenta una classe con package.
T3	valid normal class without package	empty	Passato	Stringa valida che rappresenta una classe senza package.
T4	valid array class	valid string	Passato	Stringa valida che rappresenta una classe con array.
T5	empty	empty	Passato	Stringa vuota.
T6	invalid	Exception	Fallito	Stringa non valida.
T7	null	null	Passato	Stringa nulla.

Tabella 67: Mappatura completa dei test per il metodo `getPackageName(String)` della classe `ClassUtil`.

A.6.23 Risultati dei test e analisi dei fallimenti

ID	Metodo	Analisi e Motivazione del Fallimento
T6	getPackageName	Il test fallisce poiché si è riscontrato un comportamento inatteso nella gestione delle stringhe non valide. Il metodo non effettua alcuna validazione sulla correttezza sintattica dell'input e, invece di sollevare l'eccezione prevista, restituisce meccanicamente la parte iniziale della stringa precedente l'ultimo punto. In presenza di input arbitrari come "\$%&gaia..invalid..binary*+-", il sistema non riconosce l'invalidità del nome, limitandosi a un'operazione di ritaglio testuale.

Tabella 68: Analisi dei fallimenti riscontrati durante il testing del metodo `getPackageName(String)` di `ClassUtil`.

A.6.24 Correzioni a seguito dell'analisi dei fallimenti

La Tabella 69 identifica i difetti che non possono essere risolti con una semplice modifica dei test, in quanto rappresentano anomalie logiche o di robustezza nel codice sorgente della classe `ClassUtil`.

ID	Esito	Descrizione del Bug Riscontrato
T6	Anomalia	Il fallimento evidenzia un deficit di robustezza: l'implementazione individua l'ultimo punto(.) e restituisce la sottostringa successiva senza sanificazione o validazione sintattica. Di conseguenza, stringhe contenenti metacaratteri non ammessi dalla JVM (es. "binary*+") non innescano eccezioni ma producono output privi di validità semantica. Il componente delega interamente la responsabilità della validazione al chiamante, venendo meno ai criteri di resilienza attesi per i casi di frontiera.

Tabella 69: Identificazione di anomalie critiche (bug) nel metodo `getClassName(String)` di `ClassUtil`.

A.7 Materiale Classe CacheMap

A.7.1 Inizializzazione del componente (Costruttore)

Si riporta di seguito la firma del costruttore oggetto di analisi:

```
1 public CacheMap(boolean lru, int max, int size, float load, int concurrencyLevel)
```

A.7.2 Identificazione delle classi di equivalenza

Parametro	Classi di Equivalenza
lru	{ true }, { false }
max ¹¹	{ ≤ 0 }, { > 0 }
size	{ ≤ 0 }, { > 0 }
load	{ ≤ 0.0f }, { > 0.0f }
concurrencyLevel ¹²	{ ≤ 0 }, { > 0 }

Tabella 70: Classi di equivalenza per i parametri del costruttore di CacheMap.

A.7.3 Boundary Value Analysis

Parametro	Valori di Frontiera
lru	<i>true</i> : Abilita la politica Least Recently Used per la gestione della cache. <i>false</i> : Disabilita la politica Least Recently Used per la gestione della cache.
max	-1, 0, 1, 100 (aggiunto per testare il funzionamento in scenari reali).
size	-1, 0, 1, 100 (aggiunto per testare il funzionamento in scenari reali).
load	-0.1f, 0.0f, 0.75f (aggiunto per testare il funzionamento in scenari reali).
concurrencyLevel	-1, 0, 1.

Tabella 71: Boundary Value Analysis per il costruttore di CacheMap.

A.7.4 Casi di Test - Costruttore CacheMap

La Tabella 72 descrive i casi di test per l'inizializzazione della classe CacheMap.

ID	lru	maxCap	initSize	loadF	concL	Output Atteso	Esito	Motivazione
T1	true	100	100	0.75	1	Valid Inst.	Passato	Creazione con politica LRU.
T2	false	100	100	0.75	1	Valid Inst.	Passato	Creazione senza politica LRU.
T3	true	-1	100	0.75	1	Exception	Fallito	Capacità massima negativa non ammessa.
T4	true	0	100	0.75	1	Exception	Fallito	Capacità massima nulla non ammessa.
T5	true	1	100	0.75	1	Valid Inst.	Passato	Valore minimo positivo per capacità massima.
T6	true	100	100	0.75	1	Valid Inst.	Passato	Valore positivo standard per capacità massima.
T7	true	100	-1	0.75	1	Exception	Fallito	Dimensione iniziale negativa non ammessa.
T8	true	100	0	0.75	1	Exception	Passato	Dimensione iniziale nulla non ammessa.
T9	true	100	1	0.75	1	Valid Inst.	Passato	Valore minimo positivo per la dimensione.
T10	true	100	100	-1	1	Exception	Passato	Il <code>loadFactor</code> non può essere negativo.
T11	true	100	100	0	1	Exception	Passato	Il <code>loadFactor</code> non può essere nullo.
T12	true	100	100	0.75	-1	Exception	Fallito	Il <code>concurrencyLevel</code> non può essere negativo.
T13	true	100	100	0.75	0	Exception	Fallito	Il <code>concurrencyLevel</code> non può essere nullo.

Tabella 72: Mappatura dei casi di test per il costruttore della classe CacheMap.

¹¹Sebbene sussista una correlazione logica tra `max` e `size` (limite globale vs. capacità segmentale), l'assenza di vincoli sintattici stringenti ha orientato l'analisi verso una valutazione indipendente. Poiché l'eventuale eccedenza della dimensione del segmento rispetto alla soglia massima non determina eccezioni bloccanti, ma unicamente inefficienze prestazionali, i parametri sono stati trattati come variabili disaccoppiate nel dominio di test.

¹²Analogamente, l'analisi del parametro `concurrencyLevel` ha evidenziato un'influenza nulla sulla logica operativa del componente. La persistenza di tale valore nella firma del costruttore, pur in assenza di un impatto funzionale tangibile, rende i test su questa variabile determinanti per rilevare potenziali asimmetrie tra il contratto dell'interfaccia pubblica e l'effettiva implementazione interna.

A.7.5 Risultati dei test e analisi dei fallimenti

ID	Metodo	Analisi e Motivazione del Fallimento
T3	Costruttore	Il fallimento scaturisce dall'assunzione che un valore negativo di <code>max</code> sollevi un'eccezione. L'analisi rivela che il codice interpreta i negativi come capacità illimitata, assegnando <code>Integer.MAX_VALUE</code> . L'oracolo è stato riallineato rimuovendo l'attesa dell'eccezione per riflettere questa logica di autocorrezione.
T4	Costruttore	Il test fallisce poiché il sistema accetta <code>max = 0</code> senza sollevare eccezioni. Sebbene ciò causi l'eviction immediata di ogni entry, l'istanza rimane funzionale per operazioni come il <code>pin()</code> , che opera sulla <code>pinnedMap</code> separata. L'oracolo è stato aggiornato per validare il successo dell'inizializzazione.
T7	Costruttore	Il fallimento deriva dall'aspettativa di un'eccezione per <code>size < 0</code> . Il codice adotta invece una strategia di <i>defaulting</i> : il valore negativo viene corretto internamente assegnando il valore predefinito 500. L'oracolo è stato aggiornato per verificare il corretto completamento del costruttore.
T12, T13	Costruttore	Il fallimento deriva dall'attesa di un'eccezione per <code>concurrencyLevel</code> non positivo. L'analisi mostra che il parametro è totalmente ininfluente: non viene né validato né utilizzato internamente. Data l'irrilevanza funzionale, si è proceduto alla rimozione definitiva dei casi di test associati.

Tabella 73: Analisi dei fallimenti riscontrati durante il testing del costruttore di CacheMap.

A.7.6 Correzioni a seguito dell'analisi dei fallimenti

Parametro	Classi di Equivalenza
<code>lru</code>	{true}, {false}
<code>max</code>	$M_{inf} : \{m < 0\}$ (Capacità illimitata via <code>Integer.MAX_VALUE</code>) $M_{zero} : \{m = 0\}$ (Capacità nulla: storage disabilitato) $M_{lim} : \{m > 0\}$ (Capacità limitata al valore specificato)
<code>size</code>	$S_{def} : \{s < 0\}$ (Autocorrezione al valore predefinito 500) $S_{inv} : \{s = 0\}$ (Dimensione iniziale non valida) $S_{val} : \{s > 0\}$ (Dimensione iniziale valida)
<code>load</code>	$L_{inv} : \{\leq 0.0f\}$ $L_{val} : \{> 0.0f\}$

Tabella 74: Classi di equivalenza per i parametri del costruttore di CacheMap.

La Tabella 75 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Iru	max	size	load	cL	Out Atteso	Esito	Motivazione
T3	true	-1	100	0.75	1	valid instance	Passato	Il sistema corregge i valori negativi di <code>max</code> a <code>Integer.MAX_VALUE</code> senza interrompersi.
T4	true	0	100	0.75	1	valid instance	Passato	La capacità nulla è ammessa; la cache viene creata correttamente pur non potendo ospitare entry.
T7	true	100	-1	0.75	1	valid instance	Passato	L'autocorrezione di <code>size</code> al valore di default (500) evita il lancio dell'eccezione prevista.

Tabella 75: Casi di test corretti a seguito dell'analisi del codice di CacheMap.

La Tabella 76 identifica i difetti riscontrati che non possono essere risolti con una semplice modifica dei test, in quanto rappresentano anomalie logiche o di robustezza nel codice sorgente della classe WriteCache.

ID	Esito	Descrizione del Bug Riscontrato
T12	Anomalia	Il fallimento deriva dall'attesa di un'eccezione per <code>concurrencyLevel</code> non positivo. L'analisi mostra che il parametro è totalmente ininfluente: non viene né validato né utilizzato internamente.
T13	Anomalia	Coerentemente con il caso precedente, l'analisi mostra che il parametro è totalmente ininfluente: non viene né validato né utilizzato internamente.

Tabella 76: Identificazione di anomalie critiche (bug) nella classe di WriteCache.

A.7.7 Analisi del metodo pin

Si riporta di seguito la firma del metodo oggetto di analisi:

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

A.7.8 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
key	{istanza valida in cacheMap}, {istanza valida in softMap}, {istanza valida in pinnedMap con valore non nullo}, {istanza valida in pinnedMap con valore nullo}, {istanza valida non presente in alcuna mappa}, {istanza non valida}, {istanza nulla}

Tabella 77: Classi di equivalenza per il parametri del metodo pin di CacheMap.

A.7.9 Boundary Value Analysis

Parametro	Valori di Frontiera
key	<p><i>istanza valida in cacheMap</i>: la chiave esiste nella mappa principale con un valore associato.</p> <p><i>istanza valida in softMap</i>: la chiave esiste nella softMap.</p> <p><i>istanza valida in pinnedMap con valore non nullo</i>: la chiave è bloccata con un valore valido.</p> <p><i>istanza valida in pinnedMap con valore nullo</i>: la chiave è bloccata con un valore nullo.</p> <p><i>istanza valida non presente</i>: oggetto correttamente istanziato assente da ogni struttura della cache.</p> <p><i>istanza non valida</i>: oggetto che viola il contratto di equals() o hashCode().</p> <p><i>istanza nulla</i>: riferimento null.</p>

Tabella 78: Boundary Value Analysis per il metodo pin di CacheMap.

A.7.10 Casi di Test - Metodo pin

ID	Iru	maxCap	initS	loadF	concl	Stato Iniziale	Key	Out	Esito	Motivazione
P1	true	0	100	0.75	1	INVALID	Valid	false	Passato	Una cacheMap con dimensione nulla.
P2	true	100	100	0.75	1	WITH_KEY	Valid In Cache	true	Passato	Chiave valida presente in cacheMap.
P3	true	100	100	0.75	1	IN_SOFT	Valid In SoftMap	true	Passato	Chiave valida in softMap.
P4	true	100	100	0.75	1	PINNED	Valid Pinned Not Null	true	Passato	Chiave valida con valore non nullo in pinnedMap.
P5	true	100	100	0.75	1	PINNED	Valid Pinned Nul	false	Passato	Chiave valida con valore nullo in pinnedMap.
P6	true	100	100	0.75	1	EMPTY	Valid Not Present	false	Passato	Chiave valida ma assente da ogni struttura di memoria.
P7	true	100	100	0.75	1	EMPTY	Invalid	Exc.	Passato	Chiave non valida.
P8	true	100	100	0.75	1	EMPTY	Null	Exc.	Fallito	Chiave nulla.

Tabella 79: Mappatura completa dei casi di test per il metodo pin della classe CacheMap.

A.7.11 Risultati dei test e analisi dei fallimenti

ID	Metodo	Analisi e Motivazione del Fallimento
P8	pin	Il fallimento è imputabile alla gestione permissiva dei riferimenti null. Invece di sollevare l'eccezione prevista, il metodo propaga il valore nullo alle mappe interne, che restituiscono false. L'oracolo è stato allineato per validare l'esito booleano, confermando la mancata intercettazione preventiva di input non validi.

Tabella 80: Analisi dei fallimenti riscontrati durante il testing del metodo pin di CacheMap.

A.7.12 Correzioni a seguito dell'analisi dei fallimenti

La Tabella 81 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Iru	maxCap	initS	loadF	concl	Key	Out	Esito	Motivazione
P8	true	100	100	0.75	1	null	false	Passato	Il test è stato riallineato accettando il valore false anziché l'eccezione, riflettendo la reale gestione degli input nulli.

Tabella 81: Casi di test corretti a seguito dell'analisi del codice di CacheMap.

A.7.13 Analisi del metodo unpin

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public boolean unpin(Object key)
```

A.7.14 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
key	{istanza valida in pinnedMap con valore non nullo}, {istanza valida in pinnedMap con valore nullo}, {istanza valida non presente in alcuna mappa}, {istanza non valida}, {istanza nulla}

Tabella 82: Classi di equivalenza per i parametri del metodo `unpin` di CacheMap.

A.7.15 Boundary Value Analysis

Parametro	Valori di Frontiera
key	<i>istanza valida in pinnedMap con valore non nullo</i> : la chiave è bloccata con un valore associato. <i>istanza valida in pinnedMap con valore nullo</i> : la chiave è bloccata con un valore nullo. <i>istanza valida non presente</i> : oggetto correttamente definito ma assente da ogni struttura della cache. <i>istanza non valida</i> : oggetto che viola il contratto di <code>equals()</code> o <code>hashCode()</code> . <i>istanza nulla</i> : riferimento <code>null</code> .

Tabella 83: Boundary Value Analysis per il metodo `unpin` di CacheMap.

A.7.16 Casi di Test - Metodo unpin

ID	Key	Out	Esito	Motivazione
T1	Valid Not Null	true	Passato	Chiave valida con valore non nullo, presente nella pinnedMap.
T2	Valid Null	false	Passato	Chiave valida con valore nullo presente nella pinnedMap.
T3	Valid Not Present	false	Passato	Chiave valida non presente nella pinnedMap.
T4	Invalid	false	Passato	Chiave non valida.
T5	Null	false	Passato	Chiave nulla.

Tabella 84: Mappatura completa dei casi di test per il metodo `unpin` della classe CacheMap.

A.7.17 Analisi del metodo put

Si riporta di seguito la firma del metodo oggetto di analisi:

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

A.7.18 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
key	{istanza valida in pinnedMap con valore non nullo}, {istanza valida in pinnedMap con valore nullo}, {istanza valida in softMap}, {istanza valida in cacheMap}, {istanza valida non presente in alcuna mappa}, {istanza non valida}, {istanza nulla}
value	{istanza valida}, {istanza nulla}

Tabella 85: Classi di equivalenza per i parametri del metodo `put` di CacheMap.

A.7.19 Boundary Value Analysis

Parametro	Valori di Frontiera
key	<i>istanza valida in pinnedMap con un valore non nullo</i> : la chiave è bloccata con un valore associato. <i>istanza valida in pinnedMap con un valore nullo</i> : la chiave è bloccata con un valore nullo. <i>istanza valida in softMap</i> : la chiave è in softMap. <i>istanza valida in cacheMap</i> : la chiave è in cacheMap. <i>istanza valida non presente</i> : oggetto correttamente definito ma assente da ogni struttura della mappa. <i>istanza non valida</i> : oggetto che viola il contratto di <code>equals()</code> o <code>hashCode()</code> . <i>istanza nulla</i> : riferimento <code>null</code> .
value	<i>istanza valida</i> : qualsiasi riferimento a oggetto non nullo che può essere memorizzato. <i>istanza nulla</i> : riferimento <code>null</code> .

Tabella 86: Boundary Value Analysis per il metodo `put` di CacheMap.

A.7.20 Casi di Test - Metodo put

ID	Iru	maxC	initS	loadF	concl	Key State	Value	Out	Esito	Motivazione
P1	true	0	100	0.75	1	valid	Valid	null	Passato	La cache ha una dimensione nulla.
P2	true	100	100	0.75	1	in PINNED (val)	valid	pin val	Passato	Chiave in pinnedMap con un valore non nullo: restituzione del vecchio valore.
P3	true	100	100	0.75	1	in PINNED (val)	null	pin val	Passato	Chiave in pinnedMap: inserimento di un valore null ammesso.
P4	true	100	100	0.75	1	in PINNED (null)	valid	null	Passato	Chiave in pinnedMap con valore null: inserimento di valore valido.
P5	true	100	100	0.75	1	in PINNED (null)	null	null	Passato	Chiave in pinnedMap co valore null: inserimento di un valore nullo.
P6	true	100	100	0.75	1	in SOFT	valid	soft val	Passato	Chiave in softMap: aggiornamento nella cache principale.
P7	true	100	100	0.75	1	in SOFT	null	soft val	Passato	Chiave in softMap: inserimento di un valore null.
P8	true	100	100	0.75	1	in CACHE	valid	cache val	Passato	Chiave in cacheMap: sovrascrittura standard.
P9	true	100	100	0.75	1	in CACHE	null	cache val	Passato	Chiave in cacheMap: inserimento di un valore nullo.
P10	true	100	100	0.75	1	not mapped	valid	null	Passato	Inserimento di una nuova coppia chiave-valore.
P11	true	100	100	0.75	1	not mapped	null	null	Passato	Inserimento di una nuova chiave con valore nullo.
P12	true	100	100	0.75	1	invalid	null	Exception	Passato	Inserimento di una chiave non valida.
P13	true	100	100	0.75	1	null	null	Exception	Fallito	Inserimento di una chiave nulla.

Tabella 87: Mappatura completa dei casi di test per il metodo `put` della classe CacheMap.

A.7.21 Risultati dei test e analisi dei fallimenti

ID	Metodo	Analisi e Motivazione del Fallimento
P13	get	Il fallimento conferma una carenza strutturale nella validazione degli input. Nonostante la specifica prevedesse un'eccezione per chiavi null, il sistema non implementa controlli preventivi. Il riferimento nullo viene propagato alle strutture interne senza generare errori fatali, rendendo necessario l'allineamento dell'oracolo alla reale tolleranza del codice.

Tabella 88: Analisi dei fallimenti riscontrati durante il testing del metodo get di CacheMap.

A.7.22 Correzioni a seguito dell'analisi dei fallimenti

La Tabella 89 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziale, è coerente con l'implementazione attuale.

ID	Key	Out Atteso	Esito	Motivazione
P13	null	null	Passato	Il sistema non implementa controlli <i>fail-fast</i> , ma restituisce un valore vuoto per input nulli.

Tabella 89: Casi di test corretti a seguito dell'analisi del codice di CacheMap.

A.7.23 Analisi del metodo get

Si riporta di seguito la firma del metodo oggetto di analisi:

```
1 public Object get(Object key)
```

A.7.24 Identificazione della classe di equivalenza

Parametro	Classi di Equivalenza
key	{istanza valida in softMap}, {istanza valida in cacheMap}, {istanza valida in pinnedMap}, {istanza non presente in alcuna mappa}, {istanza non valida}, {istanza nulla}

Tabella 90: Classi di equivalenza per i parametri del metodo get di CacheMap.

A.7.25 Boundary Value Analysis

Parametro	Valori di Frontiera
key	<p><i>istanza valida in softMap</i>: la chiave è presente nella memoria secondaria (soft reference) e il suo accesso potrebbe causarne il rientro nella cache principale.</p> <p><i>istanza valida in cacheMap</i>: la chiave è presente nella mappa principale con un valore associato.</p> <p><i>istanza valida in pinnedMap</i>: la chiave è presente tra gli elementi bloccati.</p> <p><i>istanza valida non presente</i>: oggetto ben definito ma assente da ogni struttura della mappa.</p> <p><i>istanza non valida</i>: oggetto che non rispetta il contratto di equals() o hashCode().</p> <p><i>istanza nulla</i>: riferimento null.</p>

Tabella 91: Boundary Value Analysis per il metodo get di CacheMap.

A.7.26 Casi di Test - Metodo get

ID	Key	Out	Esito	Motivazione
T1	In softMap	softValue	Passato	Recupero corretto di una chiave presente nella softMap.
T2	In cacheMap	cacheValue	Passato	Recupero corretto di una chiave presente nella cacheMap.
T3	In pinnedMap	pinnedValue	Passato	Recupero corretto di una chiave presente nella pinnedMap.
T4	Not Mapped	null	Passato	La chiave è valida ma non è censita in alcuna struttura di memoria.
T5	Invalid	Exception	Passato	Chiave non valida.
T6	Null	Exception	Fallito	Chiave nulla.

Tabella 92: Mappatura completa dei casi di test per il metodo get della classe CacheMap.

A.7.27 Risultati dei test e analisi dei fallimenti

ID	Metodo	Analisi e Motivazione del Fallimento
T6	get	Il fallimento del test conferma una carenza strutturale nella validazione degli input già emersa in altre operazioni della classe CacheMap. Nonostante la specifica preveda il sollevamento di un'eccezione a fronte di una chiave null, il sistema non implementa controlli di integrità preventivi. L'analisi rivela che il riferimento nullo viene propagato direttamente alle strutture dati interne, rendendo necessario l'allineamento dell'oracolo alla reale tolleranza del codice verso input non validi.

Tabella 93: Analisi dei fallimenti riscontrati durante il testing del metodo get di CacheMap.

A.7.28 Correzioni a seguito dell'analisi dei fallimenti

La Tabella 94 riporta i casi di test i cui risultati attesi sono stati aggiornati dopo aver verificato che il comportamento del codice, seppur diverso dalle ipotesi iniziali, è coerente con l'implementazione attuale.

ID	Key	Out Atteso	Esito	Motivazione
T6	null	null	Passato	L'analisi rivela l'assenza di una logica fail-fast; il sistema tollera l'input nullo restituendo un valore vuoto anziché sollevare l'eccezione inizialmente ipotizzata.

Tabella 94: Casi di test corretti a seguito dell'analisi del codice di CacheMap.

A.8 Comandi per la Generazione Automatica

Per la generazione dei test con EvoSuite 1.1.0 standalone, è stato utilizzato il seguente script Bash per la corretta gestione del classpath su ambiente macOS:

```
1 # Copia delle dipendenze del progetto
2 mvn dependency:copy-dependencies
3
4 # Costruzione del classpath dinamico
5 export MY_CP=$(find target/dependency -name "*.jar" | tr '\n' ':')target/classes
6
7 # Esecuzione della generazione
8 java -jar evosuite-1.1.0.jar \
9 -class org.apache.bookkeeper.bookie.BufferedChannel \
10 -projectCP "$MY_CP" \
11 -Dsearch_budget=120 \
12 -Dstopping_condition=MaxTime \
13 -Dshow_progress=true \
14 -Dsandbox=false
```

Listing 2: Comando di generazione EvoSuite

A.9 Configurazione e Comando Randoop

Il comando eseguito all'interno della directory bookkeeper-server è riportato di seguito:

```
1 # Definizione del percorso assoluto al JAR di Randoop
2 RANDOOP_JAR="/Users/gaiameola/Dev/ISW2/randoop-all-4.3.3.jar"
3
4 # Generazione con risoluzione dinamica delle dipendenze Maven
5 java -Xmx4g -cp "target/classes:target/test-classes:$(mvn dependency:build-classpath | grep
6   -v '[INFO\']'):$RANDOOP_JAR" \
7   randoop.main.Main gentests \
8   --testclass=org.apache.bookkeeper.bookie.BufferedChannel \
9   --testclass=org.apache.bookkeeper.bookie.randoop.RandoopHelper \
10  --npe-on-null-input=EXPECTED \
11  --time-limit=90 \
12  --junit-package-name=org.apache.bookkeeper.bookie.randoop \
13  --junit-output-dir=src/test/java \
14  --regression-test-basename=RandoopRegression \
--generated-limit=300
```

Listing 3: Comando CLI per la generazione dei test guidati

Analisi dei parametri utilizzati:

- mvn dependency:build-classpath: Comando fondamentale per risolvere e concatenare automaticamente tutte le dipendenze esterne (Netty, Guava, etc.) necessarie all'esecuzione.
- -testclass (RandoopHelper): Istruisce il generatore a utilizzare la classe helper come fonte per la creazione di oggetti validi, permettendo di superare le fasi di inizializzazione della classe sotto test.
- -npe-on-null-input=EXPECTED: Specifica che le eccezioni di tipo *NullPointerException* generate da input nulli devono essere considerate comportamenti attesi, evitando lo scarto di sequenze di test potenzialmente utili per l'esplorazione di rami secondari.

A.10 Risultati dei test di copertura

A.10.1 Risultati iniziali della copertura di BufferedChannel

In questa sezione sono riportati i dati estratti dai report JaCoCo relativi alla classe BufferedChannel.

Metodo	Statement Coverage	Branch Coverage
BufferedChannel(ByteBufAllocator, FileChannel, int, int, long)	100%	100%
read(ByteBuf, long, int)	99%	90%
write(ByteBuf)	97%	80%

Tabella 95: Risultati di Statement e Branch Coverage per i metodi principali di BufferedChannel.

A.10.2 Risultati finali della copertura di BufferedChannel

In questa sezione vengono riportati i risultati ottenuti a seguito dell'attività di raffinamento basata sulla metodologia coverage-driven.

Metodo	Statement Coverage	Branch Coverage
BufferedChannel(ByteBufAllocator, FileChannel, int, int, long)	100%	100%
read(ByteBuf, long, int)	100%	100%
write(ByteBuf)	100%	100%

Tabella 96: Risultati finali di Statement e Branch Coverage per i metodi principali di BufferedChannel.

A.10.3 Risultati iniziali della copertura di WriteCache

In questa sezione sono riportati i dati estratti dai report JaCoCo relativi alla classe WriteCache.

Metodo	Statement Coverage	Branch Coverage
WriteCache(ByteBufAllocator, long, int)	100%	100%
put(long, long, ByteBuf)	97%	75%
get(long, long)	100%	100%

Tabella 97: Risultati di Statement e Branch Coverage per i metodi principali di WriteCache.

A.10.4 Casi di Test Raffinati - Metodo put (WriteCache)

ID	Allocator	maxCS	maxSS	writeCacheType	LID	EID	Entry	Output Atteso	Esito	Motivazione
P1	invalid	512	256	NON_WRITTEN	1	2	full	TRUE	Passato	Un allocatore invalido non permette di generare buffer nulli, tuttavia l'allocator è ininfluente dal momento che i dati sono scritti nei buffer pre-allocati.
P2	null	512	256	NON_WRITTEN	1	2	full	TRUE	Passato	Un allocatore null non permette di generare buffer nulli, tuttavia l'allocator è ininfluente dal momento che i dati sono scritti nei buffer preallocati.
P3	valid	0	1	NON_WRITTEN	1	2	full	FALSE	Passato	Caso in cui la cache ha dimensione nulla.
P4	valid	512	512	NON_WRITTEN	1	2	full	TRUE	Passato	Inserimento in un segmento completamente libero.
P5	valid	512	512	HALF_SEGMENT_WRITTEN	1	2	full	TRUE	Passato	Inserimento in un segmento parzialmente libero, ma il cui spazio residuo è sufficiente.
P6	valid	512	512	ONE_SEGMENT_WRITTEN	1	2	full	FALSE	Passato	Inserimento in un segmento completamente occupato.
P7	valid	512	256	NON_WRITTEN	-1	2	full	Exception	Passato	Un ledgerId negativo non è lecito.
P8	valid	512	256	NON_WRITTEN	0	2	full	TRUE	Passato	Un ledgerId nullo non è lecito..
P9	valid	512	256	NON_WRITTEN	1	2	full	TRUE	Passato	Un ledgerId positivo è lecito.
P10	valid	512	256	NON_WRITTEN	1	-1	full	Exception	Passato	Un entryId negativo non è lecito.
P11	valid	512	256	NON_WRITTEN	1	0	full	TRUE	Passato	Un entryId nullo è lecito.
P12	valid	512	256	NON_WRITTEN	1	1	full	TRUE	Passato	Un entryId positivo è lecito.
P13	valid	512	256	NON_WRITTEN	1	2	full	TRUE	Passato	Un entryId positivo è lecito.
P14	valid	512	256	NON_WRITTEN	1	2	empty	TRUE	Passato	Inserimento di un'entry vuota.
P15	valid	512	256	NON_WRITTEN	1	2	size ≤ mSS	TRUE	Passato	Inserimento di un'entry che entra esattamente in un segmento.
P16	valid	512	256	NON_WRITTEN	1	2	size > mSS	FALSE	Passato	Inserimento di un'entry che ha una dimensione maggiore del segmento.
P17	valid	512	256	NON_WRITTEN	1	2	index invalid	Exception	Passato	Inserimento di un'entry che ha un indice di lettura non valido.
P18	valid	512	256	NON_WRITTEN	1	2	deallocated	Exception	Passato	Inserimento di un'entry deallocated.
P19	valid	512	2566	NON_WRITTEN	1	2	null	Exception	Passato	Inserimento di un'entry nulla.
P20	valid	512	256	PRE_WRITTEN (ID: 1)	1	0	full	TRUE	Passato	Inserimento di un'entry con entryId (0) < lastEntryId (1).
P21	valid	512	256	PRE_WRITTEN (ID: 1)	1	1	full	TRUE	Passato	Inserimento di un'entry con entryId (1) = lastEntryId (1).
P22	valid	512	256	PRE_WRITTEN (ID: 1)	1	2	full	TRUE	Passato	Inserimento di un'entry con entryId (2) > lastEntryId (1).

Tabella 98: Mappatura completa dei casi di test per il metodo put di WriteCache.

A.10.5 Risultati finali della copertura di WriteCache

In questa sezione vengono riportati i risultati ottenuti a seguito dell'attività di raffinamento basata sulla metodologia coverage-driven.

Metodo	Statement Coverage	Branch Coverage
WriteCache(ByteBufAllocator, long, int)	100%	100%
put(long, long, ByteBuf)	98%	87%
get(long, long)	100%	100%

Tabella 99: Risultati finali di Statement e Branch Coverage per i metodi principali di WriteCache.

A.10.6 Risultati iniziali della copertura di CacheMap

In questa sezione sono riportati i dati estratti dai report JaCoCo relativi alla classe CacheMap.

Metodo	Statement Coverage	Branch Coverage
put(Object, Object)	100%	100%
pin(Object)	100%	100%
CacheMap(boolean, int, int, float, int)	100%	100%
get(Object)	100%	100%
unpin(Object)	100%	100%

Tabella 100: Risultati di Statement e Branch Coverage per i metodi principali di CacheMap

A.10.7 Risultati iniziali della copertura di ClassUtil

In questa sezione sono riportati i dati estratti dai report JaCoCo relativi alla classe ClassUtil.

Metodo	Statement Coverage	Branch Coverage
getClassName(String)	26%	40%
getPackageName(String)	66%	70%
toClass(String, boolean, ClassLoader)	92%	83%
getClassName(Class)	100%	100%
getPackageName(Class)	100%	100%

Tabella 101: Risultati di Statement e Branch coverage per i metodi principali di ClassUtil

A.10.8 Identificazione della classe di equivalenza (Raffinata)

A seguito dell'analisi della copertura, le classi di equivalenza per il parametro `fullName` sono state raffinate per distinguere formalmente i diversi domini di input che influenzano la logica di decodifica:

Parametro	Classi di Equivalenza
<code>fullName</code>	{stringa valida classe interna}, {stringa valida classe con package (dot-notation)}, {stringa valida classe con package (slash-notation)}, {stringa valida classe senza package}, {stringa valida classe array di primitivi}, {stringa valida classe array di riferimenti}, {stringa valida classe array multidimensionale}, {stringa valida classe array con descrittore non mappato}, {stringa valida classe array di riferimenti senza terminatore}, {stringa vuota}, {stringa non valida}, {stringa nulla}

Tabella 102: Classi di equivalenza per i parametri del metodo `getClassName(String)`.

A.10.9 Boundary Value Analysis (Raffinata)

Parametro	Valori di Frontiera
fullName	<p><i>Stringa valida classe interna:</i> nomi con separatore binario (es. <code>java.util.Map\$Entry</code>).</p> <p><i>Stringa valida classe con package (dot):</i> notazione standard (es. <code>java.lang.String</code>).</p> <p><i>Stringa valida classe con package (slash):</i> notazione interna (es. <code>java/lang/String</code>).</p> <p><i>Stringa valida classe senza package:</i> classe nel default package (es. <code>MyClass</code>).</p> <p><i>Stringa valida classe array (descrittori):</i> descrittori tipo <code>[I</code> o <code>[Z</code>.</p> <p><i>Stringa valida classe array (sintassi Java):</i> descrittori tipo <code>[Ljava.lang.Object;</code>.</p> <p><i>Stringa valida classe array multidimensionale:</i> livelli multipli di profondità (es. <code>[[[I</code>).</p> <p><i>Stringa valida classe con tipo ignoto:</i> descrittore array non mappato (es. <code>"[X"</code>).</p> <p><i>Array L-type malformato:</i> oggetto senza punto e virgola finale (es. <code>"[Ljava.lang.String"</code>).</p> <p><i>Stringa vuota:</i> <code>" "</code>.</p> <p><i>Stringa non valida:</i> caratteri illegali.</p> <p><i>Stringa nulla:</i> <code>null</code>.</p>

Tabella 103: Boundary Value Analysis per il metodo `getClassName(String)`.

A.10.10 Casi di Test Raffinati - Metodo `getClassName(String)`

ID	Stringa (fullName)	Output Atteso	Esito	Motivazione
T1	<code>"java.util.Map\$Entry"</code>	<code>"Map\$Entry"</code>	Passato	Stringa valida che rappresenta una classe interna.
T2.1	<code>"java.lang.String"</code>	<code>"String"</code>	Passato	Stringa valida che rappresenta una classe con package (dot-notation).
T2.2	<code>"java/lang/Object"</code>	<code>"Object"</code>	Fallito	Stringa valida che rappresenta una classe con package (slash-notation).
T3.1	<code>"MyClass"</code>	<code>"MyClass"</code>	Passato	Stringa valida che rappresenta una classe senza package.
T3.2	<code>"int"</code>	<code>"int"</code>	Passato	Stringa valida che rappresenta un tipo primitivo senza package.
T4.1	<code>"[I"</code>	<code>"int[]"</code>	Passato	Stringa valida che rappresenta un array di primitivi (ad esempio I).
T4.2	<code>"[Z"</code>	<code>"boolean[]"</code>	Passato	Stringa valida che rappresenta un array di primitivi (ad esempio Z).
T4.3	<code>"[Ljava.lang.String;"</code>	<code>"String[]"</code>	Passato	Stringa valida che rappresenta un array di primitivi di riferimenti.
T4.4	<code>"[[[I"</code>	<code>"int[][][]"</code>	Passato	Stringa valida che rappresenta un array multidimensionale.
T4.5	<code>"[[[Ljava.awt.Point;"</code>	<code>"Point[][]"</code>	Passato	Stringa valida che rappresenta un array multidimensionale di oggetti.
T4.6	<code>"[X"</code>	<code>Exception</code>	Fallito	Stringa che rappresenta un array con codice tipo 'X' (inesistente).
T4.7	<code>"[Ljava.lang.String"</code>	<code>Exception</code>	Fallito	Stringa valida che rappresenta un descrittore di tipo 'L' senza il terminatore ';' sintatticamente invalido per la JVM.
T5	<code>" "</code>	<code>" "</code>	Passato	Stringa vuota.
T6	<code>"java..lang.String"</code>	<code>Exception</code>	Fallito	Stringa non valida.
T7	<code>null</code>	<code>null</code>	Passato	Stringa nulla.

Tabella 104: Mappatura completa e raffinata dei test per il metodo `getClassName(String)`.

A.10.11 Identificazione della classe di equivalenza (Raffinata)

A seguito dell'analisi della copertura, le classi di equivalenza per il parametro `fullName` sono state raffinate per distinguere formalmente i diversi domini di input che influenzano la logica di decodifica:

Parametro	Classi di Equivalenza
fullName	{stringa valida classe interna}, {stringa valida classe con package (dot-notation)}, {stringa valida classe con package (slash-notation)}, {stringa valida classe senza package}, {stringa valida classe array di primitivi}, {stringa valida classe array di riferimenti}, {stringa valida classe array multidimensionale}, {stringa vuota}, {stringa non valida}, {stringa nulla}

Tabella 105: Classi di equivalenza raffinate per il parametro del metodo `getPackageName(String)`.

A.10.12 Analisi boundary (Raffinata)

Parametro	Valori di Frontiera
fullName	<p><i>Classe interna:</i> nomi con separatore binario (es. <code>java.util.Map\$Entry</code>). <i>Classe con package (dot):</i> notazione standard (es. <code>java.lang.String</code>). <i>Classe con package (slash):</i> notazione interna (es. <code>java/lang/String</code>). <i>Classe senza package:</i> classe nel default package (es. <code>MyClass</code>). <i>Classe array (descrittori):</i> descrittori tipo <code>[I</code> o <code>[Z</code>. <i>Classe array (sintassi Java):</i> descrittori tipo <code>[Ljava.lang.Object;</code>. <i>Classe array multidimensionale:</i> livelli multipli di profondità (es. <code>[[[I</code>). <i>Stringa vuota:</i> "". <i>Stringa non valida:</i> caratteri illegali. <i>Stringa nulla:</i> null.</p>

Tabella 106: Boundary Value Analysis raffinata per il metodo `getPackageName(String)`.

A.10.13 Casi di Test Raffinati - Metodo `getPackageName(String)`

ID	Stringa (fullName)	Output Atteso	Esito	Motivazione
T1	" <code>java.util.Map\$Entry</code> "	" <code>java.util</code> "	Passato	Stringa valida classe interna.
T2.1	" <code>java.lang.String</code> "	" <code>java.lang</code> "	Passato	Stringa valida classe con package (dot-notation).
T2.2	" <code>java/lang/Object</code> "	" <code>java/lang</code> "	Fallito	Stringa valida classe con package (slash-notation).
T3.1	" <code>MyClass</code> "	""	Passato	Stringa valida classe nel default package.
T3.2	" <code>int</code> "	""	Passato	Stringa valida tipo primitivo.
T4.1	" <code>[I</code> "	""	Passato	Array primitivo.
T4.2	" <code>[[Ljava.lang.String;</code> "	" <code>java.lang</code> "	Passato	Stringa valida classe array di oggetti.
T4.3	" <code>[[[Ljava.awt.Point;</code> "	" <code>java.awt</code> "	Passato	Stringa valida classe array multidimensionale.
T5	""	""	Passato	Stringa vuota.
T6	" <code>java..lang.String</code> "	Exception	Fallito	Stringa malformata.
T7	null	null	Passato	Stringa nulla.

Tabella 107: Mappatura completa e raffinata dei test per il metodo `getPackageName(String)`.

A.10.14 Identificazione della classe di equivalenza (Raffinata)

Parametro	Classi di Equivalenza
str	{stringa valida classe interna}, {stringa valida classe con package (dot-notation)}, {stringa valida classe con package (slash-notation)}, {stringa valida classe senza package}, {stringa valida classe array di primitivi}, {stringa valida classe array di riferimenti}, {stringa valida classe array multidimensionale} {stringa vuota}, {stringa non valida}, {stringa nulla}
resolve	{true}, {false}
loader	{istanza valida}, {istanza non valida}, {istanza nulla}

Tabella 108: Classi di equivalenza raffinate per i parametri del metodo `toClass`.

A.10.15 Analisi boundary (Raffinata)

Parametro	Valori di Frontiera
str	<p><i>Classe interna:</i> nomi con separatore binario (es. "java.util.Map\$Entry").</p> <p><i>Classe con package (dot):</i> notazione standard Java (es. "java.lang.String").</p> <p><i>Classe con package (slash):</i> notazione interna/binaria (es. "java/lang/String").</p> <p><i>Classe senza package:</i> nomi nel default package o tipi primitivi letterali (es. "int").</p> <p><i>Classe array (descrittori):</i> nomi in formato JVM (es. "[I", "[Ljava.lang.String;").</p> <p><i>Classe array (sintassi Java):</i> nomi letterali (es. "int[]" per primitivi e "String[]" per riferimenti).</p> <p><i>Classe array multidimensionale:</i> livelli multipli di profondità sia in formato descrittore ("[[I") che letterale ("double[][]").</p> <p><i>Stringa vuota:</i> "".</p> <p><i>Stringa non valida:</i> caratteri illegali o nomi malformati (es. "java..String").</p> <p><i>Stringa nulla:</i> null.</p>
resolve	<p><i>true:</i> inizializzazione forzata della classe.</p> <p><i>false:</i> caricamento senza inizializzazione statica.</p>
loader	<p><i>Istanza valida:</i> ClassLoader specifico per il caricamento di classi utente.</p> <p><i>Istanza non valida:</i> ClassLoader con restrizioni o permessi insufficienti.</p> <p><i>Istanza nulla:</i> attivazione della logica di fallback tramite il context loader corrente.</p>

Tabella 109: Boundary Value Analysis per il metodo `toClass`.

A.10.16 Casi di Test Raffinati - Metodo `toClass` (ClassUtil)

ID	Stringa (str)	Resolve	Loader	Output Atteso	Esito	Motivazione
T1	"java.util.Map\$Entry"	F	Valid	Map\$Entry.class	Passato	Stringa valida classe interna standard (dot-notation).
T2.1	"java.lang.String"	F	Valid	String.class	Passato	Stringa valida classe con package standard (dot-notation).
T2.2	"java/lang/Object"	F	Valid	Object.class	Fallito	Stringa valida classe con package standard (slash-notation).
T3	"int"	F	Valid	int.class	Passato	Stringa valida tipo primitivo letterale.
T4.3	"[I"	F	Valid	int[].class	Passato	Stringa valida classe array di primitivi.
T4	"[[I"	F	Valid	int[] [].class	Passato	Stringa valida classe array multidimensionale.
T5	"[Ljava.lang.String;"	F	Valid	String[].class	Passato	Stringa valida classe array di riferimenti (descrittore binario).
T6	""	F	Valid	Exception	Passato	Stringa vuota.
T7	"java..lang.String"	F	Valid	Exception	Passato	Stringa malformata.
T8	null	F	Valid	Exception	Passato	Stringa nulla.
T8	"java.lang.String"	T	Valid	String.class	Passato	Inizializzazione impostata.
T9	"java.lang.String"	F	Invalid	Exception	Passato	Utilizzo di un ClassLoader non valido.
T10	"java.lang.String"	T	null	String.class	Passato	Utilizzo di un ClassLoader nullo.
T11	"int[]"	F	Valid	int[].class	Passato	Stringa valida classe array primitivo (sintassi letterale).
T12	"java.lang.String[]"	F	Valid	String[].class	Passato	Stringa valida classe array di oggetti (sintassi letterale).
T13	"double[][]"	F	Valid	double[][] .class	Passato	Stringa valida classe array multidimensionale letterale.

Tabella 110: Mappatura dei casi di test per il metodo `toClass`.

A.10.17 Risultati finali della copertura di ClassUtil

In questa sezione vengono riportati i risultati ottenuti a seguito dell'attività di raffinamento basata sulla metodologia *coverage-driven*.

Metodo	Statement Coverage	Branch Coverage
getClassName(String)	95%	90%
getPackageName(String)	100%	100%
toClass(String, boolean, ClassLoader)	100%	100%
getClassName(Class)	100%	100%
getPackageName(Class)	100%	100%

Tabella 111: Risultati finali di Statement e Branch coverage per i metodi selezionati di ClassUtil

B Report PIT

In questa sezione vengono riportati gli esiti dell'analisi di *Mutation Testing* effettuata tramite lo strumento PIT sulle classi in esame. L'analisi evidenzia la capacità della suite di test di identificare mutazioni introdotte nel codice sorgente; le mutazioni sopravvissute (evidenziate nelle figure seguenti) indicano aree in cui i test potrebbero essere ulteriormente raffinati.

B.1 Analisi BufferedChannel

Nelle Figure 1 e 2 sono dettagliati i mutanti che non sono stati uccisi dai test per i metodi della classe `BufferedChannel`.

```
118     public void write(ByteBuf src) throws IOException {
119         int copied = 0;
120         boolean shouldForceWrite = false;
121         synchronized (this) {
122             int len = src.readableBytes();
123             while (copied < len) {
124                 int bytesToCopy = Math.min(src.readableBytes() - copied, writeBuffer.writableBytes());
125                 writeBuffer.writeBytes(src, src.readerIndex() + copied, bytesToCopy);
126                 copied += bytesToCopy;
127
128                 // if we have run out of buffer space, we should flush to the
129                 // file
130                 if (!writeBuffer.isWritable()) {
131                     flush();
132                 }
133             }
134             position += copied;
135             if (doRegularFlushes) {
136                 unpersistedBytes.addAndGet(copied);
137                 if (unpersistedBytes.get() >= unpersistedBytesBound) {
138                     flush();
139                     shouldForceWrite = true;
140                 }
141             }
142         }
143         if (shouldForceWrite) {
144             forceWrite(false);
145         }
146     }
```

Figura 1: Dettaglio report PIT: mutazioni sopravvissute del metodo `write`

```
245     @Override
246     public synchronized int read(ByteBuf dest, long pos, int length) throws IOException {
247         if (dest.writableBytes() < length) {
248             throw new IllegalArgumentException("dest buffer remaining capacity is not enough"
249                     + "(must be at least as \"length\"]=" + length + ")");
250         }
251
252         long prevPos = pos;
253         while (length > 0) {
254             // check if it is in the write buffer
255             if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
256                 int positionInBuffer = (int) (pos - writeBufferStartPosition.get());
257                 int bytesToCopy = Math.min(writeBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
258
259                 if (bytesToCopy == 0) {
260                     throw new IOException("Read past EOF");
261                 }
262
263                 dest.writeBytes(writeBuffer, positionInBuffer, bytesToCopy);
264                 pos += bytesToCopy;
265                 length -= bytesToCopy;
266             } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
267                 // here we reach the end
268                 break;
269                 // first check if there is anything we can grab from the readBuffer
270             } else if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.writerIndex()) {
271                 int positionInBuffer = (int) (pos - readBufferStartPosition);
272                 int bytesToCopy = Math.min(readBuffer.writerIndex() - positionInBuffer, dest.writableBytes());
273                 dest.writeBytes(readBuffer, positionInBuffer, bytesToCopy);
274                 pos += bytesToCopy;
275                 length -= bytesToCopy;
276                 // let's read it
277             } else {
278                 readBufferStartPosition = pos;
```

Figura 2: Dettaglio report PIT: mutazioni sopravvissute del metodo `read`

B.2 Analisi WriteCache

Nelle Figure 3, 4, 5 e 6 sono dettagliati i mutanti che non sono stati uccisi dai test per i metodi della classe `WriteCache`.

```

94     public WriteCache(ByteBufAllocator allocator, long maxCacheSize, int maxSegmentSize) {
95         checkArgument(maxSegmentSize > 0);
96
97         long alignedMaxSegmentSize = alignToPowerOfTwo(maxSegmentSize);
98         checkArgument(maxSegmentSize == alignedMaxSegmentSize, "Max segment size needs to be in form of 2^n");
99
100        this.allocator = allocator;
101        this.maxCacheSize = maxCacheSize;
102        this.maxSegmentSize = (int) maxSegmentSize;
103        this.segmentOffsetMask = maxSegmentSize - 1;
104        this.segmentOffsetBits = 63 - Long.numberOfLeadingZeros(maxSegmentSize);
105
106        this.segmentsCount = 1 + (int) (maxCacheSize / maxSegmentSize);
107
108        this.cacheSegments = new ByteBuf[segmentsCount];
109
110        for (int i = 0; i < segmentsCount - 1; i++) {
111            // All intermediate segments will be full-size
112            cacheSegments[i] = Unpooled.directBuffer(maxSegmentSize, maxSegmentSize);
113        }
114
115        int lastSegmentSize = (int) (maxCacheSize % maxSegmentSize);
116        cacheSegments[segmentsCount - 1] = Unpooled.directBuffer(lastSegmentSize, lastSegmentSize);
117    }

```

Figura 3: Dettaglio report PIT: mutazioni sopravvissute del costruttore

```

187     public ByteBuf get(long ledgerId, long entryId) {
188         LongPair result = index.get(ledgerId, entryId);
189         if (result == null) {
190             return null;
191         }
192
193         long offset = result.first;
194         int size = (int) result.second;
195         ByteBuf entry = allocator.buffer(size, size);
196
197         int localOffset = (int) (offset & segmentOffsetMask);
198         int segmentIdx = (int) (offset >>> segmentOffsetBits);
199         entry.writeBytes(cacheSegments[segmentIdx], localOffset, size);
200         return entry;
201     }
202

```

Figura 4: Dettaglio report PIT: mutazioni sopravvissute del metodo get

```

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

```

Figura 5: Dettaglio report PIT: mutazioni sopravvissute del metodo put (parte 1)

```

158         } else {
159             // Found a good offset
160             break;
161         }
162     }
163
164     cacheSegments[segmentIdx].setBytes(localOffset, entry, entry.readerIndex(), entry.readableBytes());
165
166     // Update last entryId for ledger. This logic is to handle writes for the same
167     // ledger coming out of order and from different thread, though in practice it
168     // should not happen and the compareAndSet should be always uncontended.
169     while (true) {
170         long currentLastEntryId = lastEntryMap.get(ledgerId);
171         if (currentLastEntryId > entryId) {
172             // A newer entry is already there
173             break;
174         }
175
176         if (lastEntryMap.compareAndSet(ledgerId, currentLastEntryId, entryId)) {
177             break;
178         }
179     }
180
181     index.put(ledgerId, entryId, offset, size);
182     cacheCount.increment();
183     cacheSize.addAndGet(size);
184     return true;
185 }

```

Figura 6: Dettaglio report PIT: mutazioni sopravvissute del metodo put (parte 2)

B.3 Analisi ClassUtil

Nelle Figure 7, 8 e 9 sono dettagliati i mutanti che non sono stati uccisi dai test per i metodi di utilità della classe ClassUtil.

```

146     }
147     else {
148         if (fullName.charAt(fullName.length()-1) == ';') {
149             fullName = fullName.substring(dims + 1, fullName.length() - 1);
150         }
151         else {
152             fullName = fullName.substring(dims + 1);
153         }
154     }
155 }
156
157 int lastDot = fullName.lastIndexOf('.');
158 String simpleName = lastDot > -1 ? fullName.substring(lastDot + 1) : fullName;
159
160 if (dims > 0) {
161     StringBuilder sb = new StringBuilder(simpleName.length() + dims * 2);
162     sb.append(simpleName);
163     for (int i = 0; i < dims; i++) {
164         sb.append("[ ]");
165     }
166     simpleName = sb.toString();
167 }
168 return simpleName;

```

Figura 7: Dettaglio report PIT: mutazioni sopravvissute del metodo getClassName()

```

190     public static String getPackageName(String fullName) {
191 2       if (fullName == null) {
192         return null;
193     }
194 2       if (fullName.isEmpty()) {
195         return fullName;
196     }
197
198     int dims = getArrayDimensions(fullName);
199 3       if (dims > 0) {
200 3         if (fullName.length() == dims + 1) {
201           // don't care, it's a primitive
202           return "";
203         }
204       else {
205 1         fullName = fullName.substring(dims + 1);
206       }
207     }
208
209     int lastDot = fullName.lastIndexOf('.');
210 3       return lastDot > -1 ? fullName.substring(0, lastDot) : "";
211   }
212 }
```

Figura 8: Dettaglio report PIT: mutazioni sopravvissute del metodo getPackageName()

```

76 2       if (str.indexOf('.') == -1) {
77 6         for (int i = 0; !primitive && (i < _codes.length); i++) {
78 2           if (_codes[i][1].equals(str)) {
79 2             if (dims == 0) {
80               return (Class) _codes[i][0];
81             }
82             str = (String) _codes[i][2];
83             primitive = true;
84           }
85         }
86     }
87
88 3       if (dims > 0) {
89 2         StringBuilder buf = new StringBuilder(str.length() + dims + 2);
90 3         for (int i = 0; i < dims; i++) {
91           buf.append('[');
92         }
93 2         if (!primitive) {
94           buf.append('L');
95         }
96         buf.append(str);
97 2         if (!primitive) {
98           buf.append(';');
99         }
100        str = buf.toString();
101      }
102
103 2       if (loader == null) {
104         loader = Thread.currentThread().getContextClassLoader();
105     }
```

Figura 9: Dettaglio report PIT: mutazioni sopravvissute del metodo toClass()

B.4 Analisi CacheMap

Nelle Figure 10, 11, 12 e 13 sono dettagliati i mutanti che non sono stati uccisi dai test per i metodi della classe CacheMap.

```
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     }
```

Figura 10: Dettaglio report PIT: mutazioni sopravvissute del metodo pin()

```
315     /**
316      * Undo a pinning.
317      */
318     public boolean unpin(Object key) {
319         writeLock();
320         try {
321             Object val = remove(pinnedMap, key);
322             if (val != null) {
323                 // put back into unpinned cache
324                 put(key, val);
325                 _pinnedSize--;
326                 return true;
327             }
328             return false;
329         } finally {
330             writeUnlock();
331         }
332     }
```

Figura 11: Dettaglio report PIT: mutazioni sopravvissute del metodo unpin()

```

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

Figura 12: Dettaglio report PIT: mutazioni sopravvissute del metodo get()

```

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     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 }
```

Figura 13: Dettaglio report PIT: mutazioni sopravvissute del metodo put()

C Integration Testing

In questa sezione vengono riportati i risultati analitici dei test di integrazione condotti tra SingleDirectoryDbLedgerStorage e WriteCache. In questa sezione vengono riportati i criteri di partizionamento e i risultati analitici dei test di integrazione condotti tra SingleDirectoryDbLedgerStorage e WriteCache¹³.

Piano di Category Partition

La seguente tabella illustra le categorie e le partizioni individuate per coprire gli stati interni del sistema di storage.

Categoria	Partizioni (Classi di Equivalenza)
Disponibilità Memoria	<i>Capacità Residua</i> : La cache ha spazio sufficiente. <i>Saturazione</i> : La cache è piena e richiede flush.
Stato Concorrenza	<i>Timestamp Coerente</i> : Nessuna rotazione durante l'operazione. <i>Timestamp Invalido</i> : Rotazione rilevata tramite StampedLock.
Esito Primitiva put	<i>Successo</i> : Completato al primo tentativo. <i>Fallimento Transitorio</i> : Richiede retry logico. <i>Fallimento Critico</i> : Richiede rotazione forzata (Backpressure).
Stato del Flushing	<i>Inattivo</i> : Cache di backup libera. <i>In corso</i> : Thread di persistenza occupato a liberare la cache secondaria.

Tabella 112: Piano di Category Partition per l'integrazione Ledger-Cache.

Riepilogo Esecuzione Test

In questa sezione vengono riportati i risultati analitici dei test di integrazione condotti tra SingleDirectoryDbLedgerStorage e WriteCache¹⁴.

ID	Nome Test	Motivazione Tecnica e Logica di Verifica	Esito
IT_01	testAddEntry_nominal	Validazione del <i>Happy Path</i> : verifica che in condizioni di memoria libera e lock valido l'entry venga inserita con una singola operazione atomica.	Passato
IT_02	retryOnInvalidLock	Verifica della <i>Retry Logic</i> ottimistica: simula un'invalidazione dello StampedLock forzando il fallimento della prima put. Si accerta che il sistema esegua il secondo tentativo previsto dal protocollo di recupero.	Passato
IT_03	testWriteCacheRotation	Verifica della gestione della saturazione: riempie la cache fino al limite per forzare l'attivazione di triggerFlushAndAddEntry e conferma che l'inserimento avvenga correttamente sulla nuova istanza di cache post-rotazione.	Passato
IT_04	triplePutOnExtremeSaturation	Test dello scenario critico (<i>Worst Case</i>): simula il fallimento sequenziale sia del tentativo ottimistico che di quello post-rotazione, validando la resilienza del sistema fino al terzo tentativo di scrittura forzata.	Passato

Tabella 113: Riepilogo analitico dell'esecuzione dei test di integrazione.

¹³Per indurre la rotazione deterministica è stata forzata una configurazione con dbStorage_writeCacheMaxSizeMb = 1.

¹⁴Per tutti i test è stata forzata una configurazione con dbStorage_writeCacheMaxSizeMb = 1 per indurre la rotazione deterministica. Lo stato della cache è stato monitorato tramite uno Spy di Mockito iniettato a runtime per tracciare le interazioni senza alterare la persistenza reale.