

Bookkeeper test

Luca Falasca

11 ottobre 2023

Indice

1	Introduction	1
2	Bookkeeper	1
2.1	Journal	1
2.1.1	Descrizione	1
2.1.2	Category Partition	1
2.1.3	Boundary Analysis	2
2.1.4	Adequacy Control Flow	3
2.1.5	Adequacy Data Flow	4
2.1.6	Mutation Testing	5
2.2	FileInfo	6
2.2.1	Descrizione	6
2.2.2	Category Partition	7
2.2.3	Boundary Analysis	7
2.2.4	Adequacy Control Flow	8
2.2.5	Adequacy Data Flow	9
2.2.6	Mutation Testing	9
3	Avro	9
3.1	Classe 1	9
3.1.1	Descrizione	9
3.1.2	Category Partition	9
3.1.3	Boundary Analysis	9
3.1.4	Adequacy Control Flow	9
3.1.5	Adequacy Data Flow	9
3.1.6	Mutation Testing	9
3.2	Classe 2	9
3.2.1	Descrizione	9
3.2.2	Category Partition	9
3.2.3	Boundary Analysis	9
3.2.4	Adequacy Control Flow	9
3.2.5	Adequacy Data Flow	9
3.2.6	Mutation Testing	9

Elenco delle figure

1	Diagrammi di listJournallds	10
2	Jacoco coverage dilistJournallds	10
3	Jacoco coverage dilistJournallds	11
4	Jacoco coverage dilistJournallds	11
5	Codice sorgente di JournalTest	11
6	Badua coverage dilistJournallds	12
7	Badua coverage dilistJournallds	12
8	Jacoco coverage dilistJournallds	12
9	Pitest coverage dilistJournallds	13
10	Mutazione inserita nel codice sorgente di Bookkeeper	13
11	Mutazione inserita nel codice sorgente di Bookkeeper	14
12	Codice sorgente di JournalTest aggiornato con il mock	14
13	Jacoco coverage readHeader	14

Elenco delle tabelle

1	Journal: Test Suite - Category Partition	9
2	Journal: Test Suite - Adequacy Control Flow 1	9
3	Journal: Test Suite - Adequacy Control Flow 2	15
4	Journal: Test Suite - Adequacy Control Flow 3	15
5	Journal: Test Suite - Adequacy Control Flow 4	15
6	Journal: Test Suite - Adequacy Data Flow 1	15
7	Journal: Test Suite - Adequacy Data Flow 1	15
8	Test Suite - Category Partition	15

1 Introduction

2 Bookkeeper

2.1 Journal

2.1.1 Descrizione

Path: org.apache.bookkeeper.bookie.Journal.java

listJournalIds Lists all journal IDs filtered by a specified journal ID filter.

This method scans the given directory containing journal log files and extracts journal IDs based on the provided filter. If no filter is provided, all journal IDs present in the directory are returned.

Input:

- journalDir journal dir : The directory containing journal log files.
- filter journal id filter

Output: list of filtered ids

2.1.2 Category Partition

journalDir

- {Directory contenente file di log}
- {Directory contenente file di log e altri file}
- {Directory contenente file non di log}
- {path non esistente}
- {Path di un file}
- null

Siccome la variabile journalDir è il path di una directory, ho partizionato il dominio in base al contenuto della directory e al suo effettivo utilizzo, rendendo la scelta delle partizioni una conseguenza del fatto che il metodo lavora su file di log.

JournalIdFilter

- Filtro esistente
- Filtro sempre True
- Filtro sempre False
- Filtro inesistente
- null

2.1.3 Boundary Analysis

Andiamo a definire per ogni partizione i Boundary values

journalDir

- {Directory contenente 1 file di log}
- {Directory contenente 1 file di log e 1 file di testo}
- {Directory contenente 1 file di testo}
- {path non esistente}
- {Path di un file di log}
- null

JournalIdFilter

- JournalRollingFilter – Questo filtro è l'unico filtro esistente utilizzato nell'applicazione, tuttavia siccome viene utilizzato in un contesto molto specifico dell'applicazione e non è un filtro generico, l'ho rimpiazzato con una sua versione semplificata più generale. Altrimenti non sarebbe stato adatto ad un test di unità, ma sarebbe stato più un test di integrazione
- Filtro sempre True
- Filtro sempre False
- MyFilter – Questo filtro è un filtro personalizzato creato appositamente per questo test che va a filtrare i journal in base al loro nome, in particolare se il journalId è > 0 viene accettato, altrimenti no. Questo filtro ha lo scopo di testare come si comporta il sistema in caso di definizione di un nuovo filtro non già esistente nel sistema e quindi non è importante il tipo di filtraggio che fa.
- null

Siccome i due parametri di input sono abbastanza scorrelati tra loro ha più senso adottare un approccio unidimensionale piuttosto che uno multidimensionale che sarebbe più adatto quando ci sono delle interazioni forti e che portano alla necessità di testare tutte le combinazioni tra i parametri. Inoltre avendo un approccio multidimensionale si finirebbe probabilmente per avere molti test non rilevanti e vanno a coprire scenari già coperti, e quindi sarebbero inutili.

Enumerano ora i casi di test derivanti da quest'analisi (Tabella 1)

Codice sorgente di JournalTest (Figura 5)

2.1.4 Adequacy Control Flow

Ora per verificare l'adeguatezza dei casi di test, vado a definire dei criteri di adeguatezza.

Criteri di adeguatezza black box

documentazione a disposizione:
javadoc:

```
/**
 * List all journal ids by a specified journal id filter.
 *
 * @param journalDir journal dir
 * @param filter journal id filter
 * @return list of filtered ids
 */
```

documentazione apache: Journals A journal file contains BookKeeper transaction logs. Before any update to a ledger takes place, the bookie ensures that a transaction describing the update is written to non-volatile storage. A new journal file is created once the bookie starts or the older journal file reaches the journal file size threshold.

Possiamo dedurre da queste documentazioni che questo metodo ricava gli id dei journal in una determinata cartella, filtrandoli in base ad un filtro specificato, se non viene specificato nessun filtro, allora vengono restituiti tutti gli id dei journal presenti nella cartella.

La Figura 1 mostra un diagramma funzionale del metodo listJournalIds. A partire da questo definisco i seguenti criteri di adeguatezza:

- almeno un test con un filtro
- almeno un test senza filtro
- almeno un test senza journal nella cartella
- almeno un test con almeno un journal nella cartella

In questo caso i criteri di adeguatezza sono già stati soddisfatti dai casi di test precedentemente descritti.

Dal diagramma precedente possiamo ricavare un control flow graph (Figura 1), per poi utilizzare dei criteri di copertura basati sul control flow

Dato che ho un approccio black box e quindi non sto utilizzando il codice sorgente per verificare l'adeguatezza dei casi di test, ipotizzo in base agli input se un caso di test andrà a coprire un certo arco del CFG o meno. Dato che in questa fase sto avendo un approccio black box, e quindi basato sulla funzionalità del metodo, eviterò di utilizzare la coverage come parametro di adeguatezza, in quanto non è possibile ricavare dall'output del metodo se un certo arco è stato coperto o meno, ma solo se il risultato è corretto oppure no. Quindi valuterò la copertura del CFG solo in una fase successiva, quando prenderò in considerazione anche il codice sorgente (approccio whitebox), utilizzando la coverage come parametro di adeguatezza. Quello che farò in questa fase è quindi valutare se i casi di test che ho definito in precedenza coprono tutti gli archi del CFG, in base alla mia conoscenza del metodo e dei suoi input, e nel caso aggiungere quelli mancanti.

Arco (1,2) → textgreater Tutti i test che hanno almeno un file di log coprono questo caso, quindi non è necessario aggiungere ulteriori test Arco (2,3) - Tutti i test che hanno almeno un file di log e un filtro non null coprono questo caso, quindi non è necessario aggiungere ulteriori test Arco (2,4) - Questo arco non è coperto perchè manca un test che abbia almeno un file di log e il filtro null. Basta quindi aggiungere il test necessario (Figura 2)

Arco (3,4) - Tutti i test che hanno almeno un file di log e un filtro non null coprono questo caso, quindi non è necessario aggiungere ulteriori test

è stato necessario quindi aggiungere un caso di test e quindi la attuale test suite è descritta nella Tabella 3

Ora andiamo a valutare la coverage ottenuta con jacoco con i casi di test definiti fino ad ora (Figura ??)

Come si può vedere dalla figura ??, la statement coverage è del 100% (colonna missed instruction), e quindi tutte le linee di codice del metodo sono state eseguite. Invece la condition coverage (colonna missed branches) è del 83%, ed è quindi migliorabile. Andando a vedere in dettaglio il report, si può notare che una condizione che non è stata coperta è una delle 4 combinazioni dell'if a riga 106 come si vede dalla figura ??

La condizione presa in esame è la seguente:

```
logFiles == null || logFiles.length == 0
```

il branch mancato è quello dove logFiles == null, questo perchè se il parametro di input journalDir è null non esegue proprio il metodo listFiles(). questo è dovuto al fatto che il path non era esistente e quindi il metodo tornava null. Andiamo quindi a definire un caso di test che copra questo branch Per farlo andiamo semplicemente basta aggiungere un category partition che copre il caso di una cartella esistente ma vuota (Tabella 4)

Ora quindi l'insieme dei test è quello mostrato nella tabella 5, e grazie a questo test la condition coverage è salita al 91% (Figura ??), con un aumento del 8

2.1.5 Adequacy Data Flow

Andiamo ora ad fare un lavoro di adeguatezza del dataflow utilizzando il framework badua per calcolare la all-uses coverage.

Dalla Figura 6 possiamo calcolare la all uses coverage.

$$CU_c + PU_c = 18$$

$$(CU + PU) - (CU_f + PU_f) = 18 + 2 = 20$$

$$\frac{CU_c + PU_c}{(CU + PU) - (CU_f + PU_f)} = \frac{18}{20} = 0.9$$

Ci sono quindi 2 coppie def-use che non sono state coperte, e quindi la all uses coverage è del 90%. Per aumentare la coverage è necessario aggiungere un caso di test che copra almeno una di queste coppie. Prendiamo in considerazione la coppia def-use sulla variabile filter non coperta. Si tratta di una coppia c-use dove il target è il continuo del ciclo for in cui è contenuto lo statement. In effetti non c'è nessun caso di test che va a coprire il caso in cui la coppia def-use è coperta e successivamente continua il ciclo. Per fare ciò vado ad aggiungere un caso di test che continua il ciclo for una volta rifiutata l'espressione booleana in cui è contenuto il comando. Per fare ciò basta aggiungere una cartella in cui controlla un altro id di journal dopo averne rifiutato uno. Effettivamente badua è servito a migliorare la qualità dei test perchè nonostante la coverage coprisse già queste righe di codice non era stata in grado di intercettare la mancanza di test su questo comportamento.

W Il caso di test è descritto nella Tabella 7. Quello che ho fatto è creare una nuova cartella che contiene due file di log e un filtro con condizione ≥ 1 , il primo dei due file viene rifiutato e il secondo invece viene accettato. In questo modo il metodo dopo aver scartato il primo file di log, continua con la sua ricerca e trova il secondo file di log che viene accettato. In questo modo la coppia def-use sulla variabile filter viene coperta e la all uses coverage sale (Figura 7)

Con questo caso di test vado a coprire anche l'altra coppia def-use che non era stata coperta. Questo succede perchè anche a lei mancava il caso in cui una volta che il filtro rifiutava un id il metodo continuava la sua ricerca e trovava un id successivo.

Così la all-use coverage riesce a raggiungere il 100%

Inoltre come effetto secondario si può notare che anche la condition coverage è salita al 100% (Figura 8)

2.1.6 Mutation Testing

Andiamo ora a fare un lavoro di adeguatezza utilizzando il framework pitest per calcolare la mutation coverage.

Come si può notare dalla Figura 9 la mutation coverage è dello 0%, ad una prima occhiata sembrerebbe perchè tutte le mutazioni create da pit non sono state rilevate. Tuttavia se andiamo a verificare ogni mutazione manualmente in realtà i test falliscono e quindi in realtà sembrerebbe che ci sia un bug su pit.

Indagando meglio ho scoperto che questo comportamento è probabilmente dovuto al fatto che il metodo listJournalIds è un metodo statico e per il modo in cui pit inserisce le mutazioni può creare problemi. Purtroppo non sono riuscito a risolvere questo problema, quindi calcolerò la coverage a mano andando a verificare mutazione per mutazione.

Ad esempio prendiamo la mutazione a riga 112, se andiamo nel codice di Bookkeeper e lo modifichiamo per inserire la mutazione (Figura 10) e poi eseguiamo i test, possiamo notare che più di un test fallisce, quindi in realtà la mutazione è stata rilevata dalla test suite. (Figura 11)

Questo succede per tutte le mutazioni, tranne per quella a riga 125, che non viene rilevata dalla test suite. Si tratta di una mutazione che omette la riga che si occupa di ordinare la lista di file di log.

Quindi possiamo calcolare la mutation coverage come segue:

$$\frac{Mutazioni\ rilevate}{Mutazioni\ totali} = \frac{6}{7} = 0.85$$

Per aumentare la coverage vado quindi a modificare la test suite per coprire anche la mutazione a riga 125. Siccome la copertura manca perchè nei test correnti non viene mai ritornato un log con dimensione >1, e quindi non c'è niente da ordinare. In particolare nel test che andrò ad aggiungere andrò a creare una cartella con 2 file di log che hanno id decrescente, in questo modo il metodo ordina i file di log e quindi la mutazione viene rilevata. Tuttavia ciò non è possibile perchè quando si vanno ad inserire dei file in una cartella essi vengono ordinati già in ordine alfabetico e quindi quando vengono presi sono già ordinati. Non è quindi possibile naturalmente creare una situazione in cui i file che vengono presi dalla cartella tramite il metodo listFiles non siano già ordinati.

Quindi ci sono due strade da poter percorrere:

- Andare ad etichettare l'implementazione del metodo senza la riga 112 (la riga che si occupa di ordinare la lista) come equivalente a quella senza la riga, e quindi non considerare la mutazione valida perchè è un codice che non comporta nessun cambiamento.
- Ipotesizzare che in futuro possa esserci un caso in cui la lista non è ordinata, dovuto ad un cambiamento del codice della libreria, oppure ad un sistema operativo diverso. Quindi andare a creare una situazione impossibile tramite mock per verificare questa eventualità.

Ho scelto di intraprendere la seconda strada perchè mi sembra la più manutenibile, vado quindi a mockare il metodo listFiles, in modo tale che una volta aver recuperato i file li vada a ordinare in modo decrescente (Figura 12)

Ora effettivamente il test non passa se commentiamo la linea 125, e quindi la mutazione viene rilevata. Quindi tutte le mutazioni sono state rilevate e quindi la mutation coverage è del 100%

2.2 FileInfo

2.2.1 Descrizione

Path: org.apache.bookkeeper.bookie.EntryLoggerAllocator.java


```

/**
 * This is the file handle for a ledger's index file that maps entry ids to location.
 * It is used by LedgerCache.
 *
 * <p>
 * Ledger index file is made of a header and several fixed-length index pages, which
 * in entry loggers
 * <pre>&lt;header&gt;&lt;index pages&gt;</pre>
 * <b>Header</b> is formatted as below:
 * <pre>&lt;magic bytes&gt;&lt;len of master key&gt;&lt;master key&gt;</pre>
 * <ul>
 * <li>magic bytes: 4 bytes, 'BKLE', version: 4 bytes
 * <li>len of master key: indicates length of master key. -1 means no master key stor
 * <li>master key: master key
 * <li>state: bit map to indicate the state, 32 bits.
 * </ul>
 * <b>Index page</b> is a fixed-length page, which contains serveral entries which po
 * in entry loggers.
 * </p>
 */

```

readHeader Legge l'header di un ledger index file per verificarne la correttezza. Lancia una eccezione se l'header non è corretto, altrimenti non ritorna nulla.

Come input prende ovviamente il file di cui deve leggere l'header. Tuttavia per rendere la category partition più sensata vado a considerare l'input come le componenti di cui è formato l'header, ovvero:

Input:

- magic bytes
- len of master key
- master key

2.2.2 Category Partition

Magic Bytes: valido, non valido, null Master key: null, master key di 0 byte, master key di ≥ 0 byte Len of master key: = len master key | len master key \geq len master key null state: null fenced non fenced Siccome in documentazione non c'è scritto niente su questo stato dell'header, ho lasciato l'approccio black box sulla category partition di questo parametro, e andando a controllare nel codice ho scoperto che nella classe viene solamente controllato se lo stato è fenced o non fenced, quindi ho descritto la category partition in questo modo

2.2.3 Boundary Analysis

Andiamo a definire per ogni partizione i Boundary values

Magic Bytes

- {valido} = BKLE
- {non valido} = BKLU

Master key

- {master key di 0 byte} = new byte[]
- {master key di > 0 byte} = new byte[1]

Len of master key

- {= len effettiva } = len effettiva
- {<len effettiva} = len effettiva - 1
- {>len effettiva} = len effettiva + 1

state

- {fenced} = 1
- {non fenced} = 0

Inoltre nell'header c'è un campo che indica la versione, non specificato nella documentazione, ma analizzando il codice si può notare che ne esistono due tipi, la versione 0 e la versione 1. Per questo motivo vado a specificare una category partition per questo campo, in modo tale da poter testare entrambe le versioni.

Version

- {versione 0} = 0
- {versione 1} = 1

Andrò ad utilizzare un approccio unidimensionale nella stesura dei casi di test, tranne per le coppie di parametri master key e len of master key, che essendo strettamente correlate ho ritenuto opportuno testare tutte le combinazioni tra loro.

I casi di test sono specificati nella Tabella 8

2.2.4 Adequacy Control Flow

Ora per verificare l'adeguatezza dei casi di test, vado ad utilizzare Jacoco per calcolare la coverage.

Come si può vedere della figura 13

2.2.5 Adequacy Data Flow

2.2.6 Mutation Testing

3 Avro

3.1 Classe 1

3.1.1 Descrizione

3.1.2 Category Partition

3.1.3 Boundary Analysis

3.1.4 Adequacy Control Flow

3.1.5 Adequacy Data Flow

3.1.6 Mutation Testing

3.2 Classe 2

3.2.1 Descrizione

3.2.2 Category Partition

3.2.3 Boundary Analysis

3.2.4 Adequacy Control Flow

3.2.5 Adequacy Data Flow

3.2.6 Mutation Testing

Tabella 1: Test Suite derivante dalla category partition e dalla boundary analysis

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	JournalRollingFilter	[1]
Directory contenente 1 file di log e 1 file di testo	Filtro sempre True	[1]
Directory contenente 1 file di testo	Filtro sempre False	[]
path non esistente	MyFilter	[]
Path di un file di log	null	Exception
null	null	Exception

Tabella 2: Test per coprire l'arco (2,4)

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	null	Exception

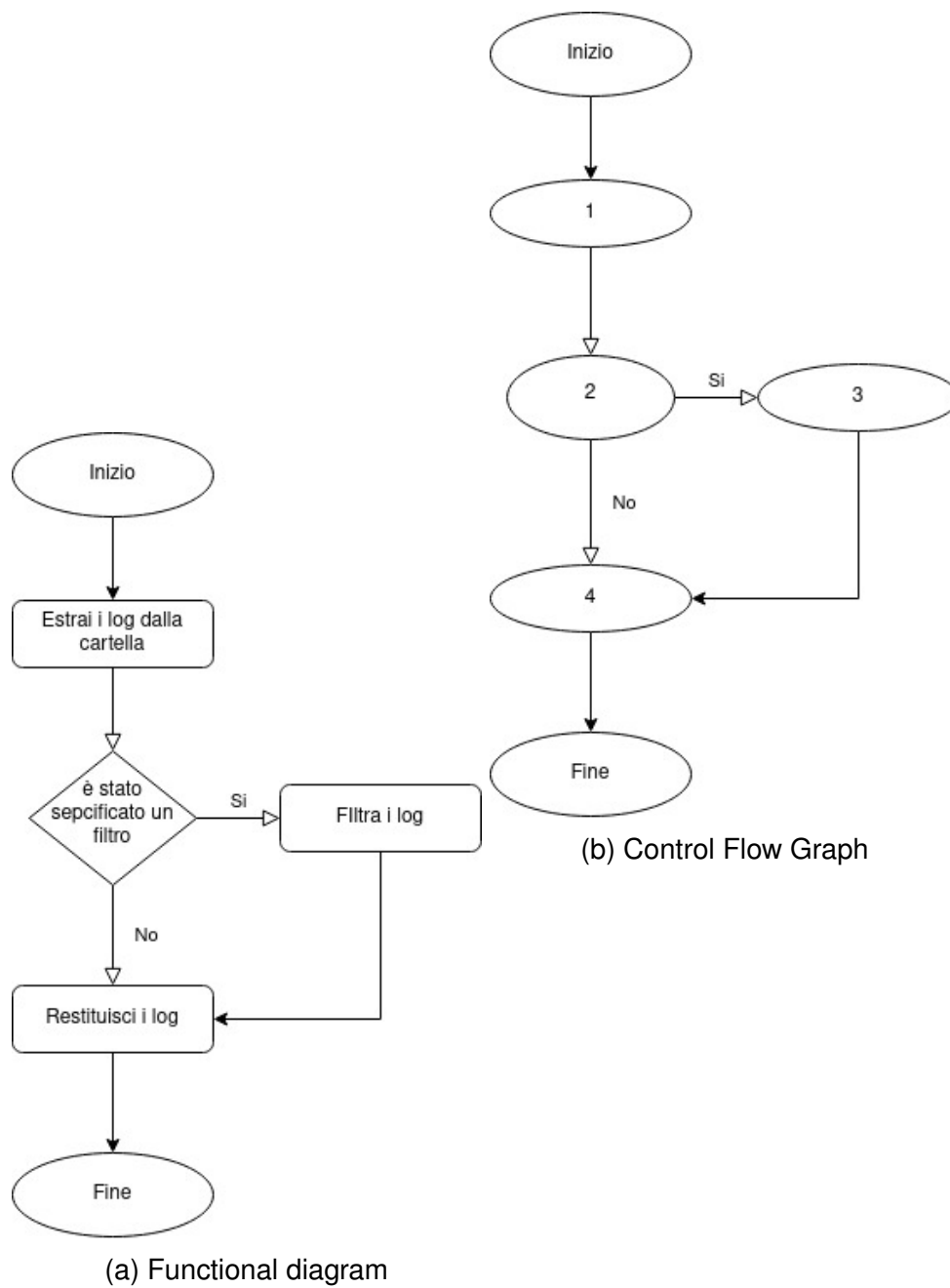


Figura 1: Diagrammi di listJournalIds

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
listJournalIds(File, Journal, JournalIdFilter)	<div></div>	100%	<div></div>	83%	2 7	0 16	0 1

Figura 2: Jacoco coverage di listJournalIds

```

103.    ~/
104.    public static List<Long> listJournalIds(File journalDir, JournalIdFilter filter) {
105.        File[] logFiles = journalDir.listFiles();
106.        if (logFiles == null || logFiles.length == 0) {
107.            return Collections.emptyList();
108.        }
109.        List<Long> logs = new ArrayList<Long>();
110.        for (File f: logFiles) {
111.            String name = f.getName();
112.            if (!name.endsWith(".txn")) {
113.                continue;
114.            }
115.            String idString = name.split("\\.")[0];
116.            long id = Long.parseLong(idString, 16);
117.            if (filter != null) {
118.                if (filter.accept(id)) {
119.                    logs.add(id);
120.                }
121.            } else {
122.                logs.add(id);
123.            }
124.        }
125.        Collections.sort(logs);
126.        return logs;
127.    }

```

Figura 3: Jacoco coverage dilistJournalIds

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
listJournalIds(File, JournalIdFilter)		100%		91%	1	7	0
						16	0
							1

Figura 4: Jacoco coverage dilistJournalIds

```

125.    @Parameterized.Parameters
126.    public static Collection<Object[]> getTestParameters() {
127.        return Arrays.asList(new Object[][]{
128.            {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), JournalIdFilterType.JOURNAL_ROLLING_FILTER.getJournalIdFilter(),
129.             {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), null},
130.             {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_ONE_TEXT_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_TRUE_FILTER.getJournalIdFilter(),
131.             {new ArrayList<>(), null, JournalDirType.ONE_TEXT_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_FALSE_FILTER.getJournalIdFilter(),
132.             {new ArrayList<>(), null, JournalDirType.NOT_EXISTING_DIR.getJournalDir(), JournalIdFilterType.NEW_FILTER.getJournalIdFilter(),
133.             {new ArrayList<>(), null, JournalDirType.LOG_FILE.getJournalDir(), null},
134.             {null, Exception.class, null, null},
135.        });
136.    }
137.
138.    @Before
139.    public void setUp() {
140.        MockitoAnnotations.initMocks(this);
141.        if (journalDir != null && journalDir.getPath().substring(beginIndex, journalDir.getPath().length() - 8).equals("void_dir")) {
142.            boolean succes = journalDir.mkdir();
143.            if (!succes) {
144.                System.out.println("Could not create void dir");
145.            } else {
146.                System.out.println("Created void dir");
147.            }
148.        }
149.    }
150.
151.    @Test
152.    public void getJournalIdsTest() {
153.        try {
154.            System.out.println("Journal dir: " + journalDir.getPath());
155.            System.out.println("Journal dir list files: " + Arrays.toString(journalDir.listFiles()));
156.
157.            List<Long> journalIds = Journal.listJournalIds(journalDir, journalIdFilter);
158.
159.            System.out.println("Result: " + journalIds);
160.            System.out.println("Expected result: " + expectedResult);
161.
162.            assertEquals(journalIds, expectedResult);
163.        } catch (Exception e) {
164.            System.out.println("Exception: " + e);
165.            if (expectedException != null && expectedException.isAssignableFrom(e.getClass())) {
166.                assertTrue();
167.            } else {
168.                assertFalse();
169.            }
170.        }
171.    }

```

Figura 5: Codice sorgente di JournalTest

```

<class name="org/apache/bookkeeper/bookie/Journal">
  <method name="listJournalIds" desc="(Ljava/io/File;Lorg/apache/bookkeeper/bookie/Journal$JournalIdFilter;)Ljava/util/List;">
    <du var="filter" def="105" use="117" target="118" covered="1"/>
    <du var="filter" def="105" use="117" target="122" covered="1"/>
    <du var="filter" def="105" use="118" target="119" covered="1"/>
    <du var="filter" def="105" use="118" target="110" covered="0"/>
    <du var="logFiles" def="105" use="106" target="106" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="109" covered="1"/>
    <du var="logFiles" def="105" use="110" covered="1"/>
    <du var="logs" def="109" use="125" covered="1"/>
    <du var="logs" def="109" use="126" covered="1"/>
    <du var="logs" def="109" use="122" covered="1"/>
    <du var="logs" def="109" use="119" covered="1"/>
    <du var="name" def="111" use="112" target="113" covered="1"/>
    <du var="name" def="111" use="112" target="115" covered="1"/>
    <du var="name" def="111" use="115" covered="1"/>
    <du var="id" def="116" use="122" covered="1"/>
    <du var="id" def="116" use="118" target="119" covered="1"/>
    <du var="id" def="116" use="118" target="110" covered="0"/>
    <du var="id" def="116" use="119" covered="1"/>
    <counter type="DU" missed="3" covered="28"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>

```

Figura 6: Badua coverage dilistJournalIds

```

<class name="org/apache/bookkeeper/bookie/Journal">
  <method name="listJournalIds" desc="(Ljava/io/File;Lorg/apache/bookkeeper/bookie/Journal$JournalIdFilter;)Ljava/util/List;">
    <du var="filter" def="105" use="117" target="118" covered="1"/>
    <du var="filter" def="105" use="117" target="122" covered="1"/>
    <du var="filter" def="105" use="118" target="119" covered="1"/>
    <du var="filter" def="105" use="118" target="110" covered="1"/>
    <du var="logFiles" def="105" use="106" target="106" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="109" covered="1"/>
    <du var="logFiles" def="105" use="110" covered="1"/>
    <du var="logs" def="109" use="125" covered="1"/>
    <du var="logs" def="109" use="126" covered="1"/>
    <du var="logs" def="109" use="122" covered="1"/>
    <du var="logs" def="109" use="119" covered="1"/>
    <du var="name" def="111" use="112" target="113" covered="1"/>
    <du var="name" def="111" use="112" target="115" covered="1"/>
    <du var="name" def="111" use="115" covered="1"/>
    <du var="id" def="116" use="122" covered="1"/>
    <du var="id" def="116" use="118" target="119" covered="1"/>
    <du var="id" def="116" use="118" target="110" covered="1"/>
    <du var="id" def="116" use="119" covered="1"/>
    <counter type="DU" missed="1" covered="30"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>

```

Figura 7: Badua coverage dilistJournalIds

Journal

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
listJournalIds(File, JournalIdFilter)	0	100%	0	100%	0	7	1

Figura 8: Jacoco coverage dilistJournalIds

```

104     public static List<Long> listJournalIds(File journalDir, JournalIdFilter filter) {
105         File[] logFiles = journalDir.listFiles();
106         if (logFiles == null || logFiles.length == 0) {
107             return Collections.emptyList();
108         }
109         List<Long> logs = new ArrayList<Long>();
110         for (File f: logFiles) {
111             String name = f.getName();
112             if (!name.endsWith(".txn")) {
113                 continue;
114             }
115             String idString = name.split("\\.")[0];
116             long id = Long.parseLong(idString, 16);
117             if (filter != null) {
118                 if (filter.accept(id)) {
119                     logs.add(id);
120                 }
121             } else {
122                 logs.add(id);
123             }
124         }
125         Collections.sort(logs);
126         return logs;
127     }
128 }

```

Figura 9: Pitest coverage dilistJournalIds

```

public static List<Long> listJournalIds(File journalDir, JournalIdFilter filter) {
    File[] logFiles = journalDir.listFiles();
    if (logFiles == null || logFiles.length == 0) {
        return Collections.emptyList();
    }
    List<Long> logs = new ArrayList<>();
    for (File f: logFiles) {
        String name = f.getName();
        if (name.endsWith(".txn")) {
            continue;
        }
        String idString = name.split(regex: "\\.[0];
        long id = Long.parseLong(idString, radix: 16);
        if (filter != null) {
            if (filter.accept(id)) {
                logs.add(id);
            }
        } else {
            logs.add(id);
        }
    }
    Collections.sort(logs);
    return logs;
}

```

Figura 10: Mutazione inserita nel codice sorgente di Bookkeeper

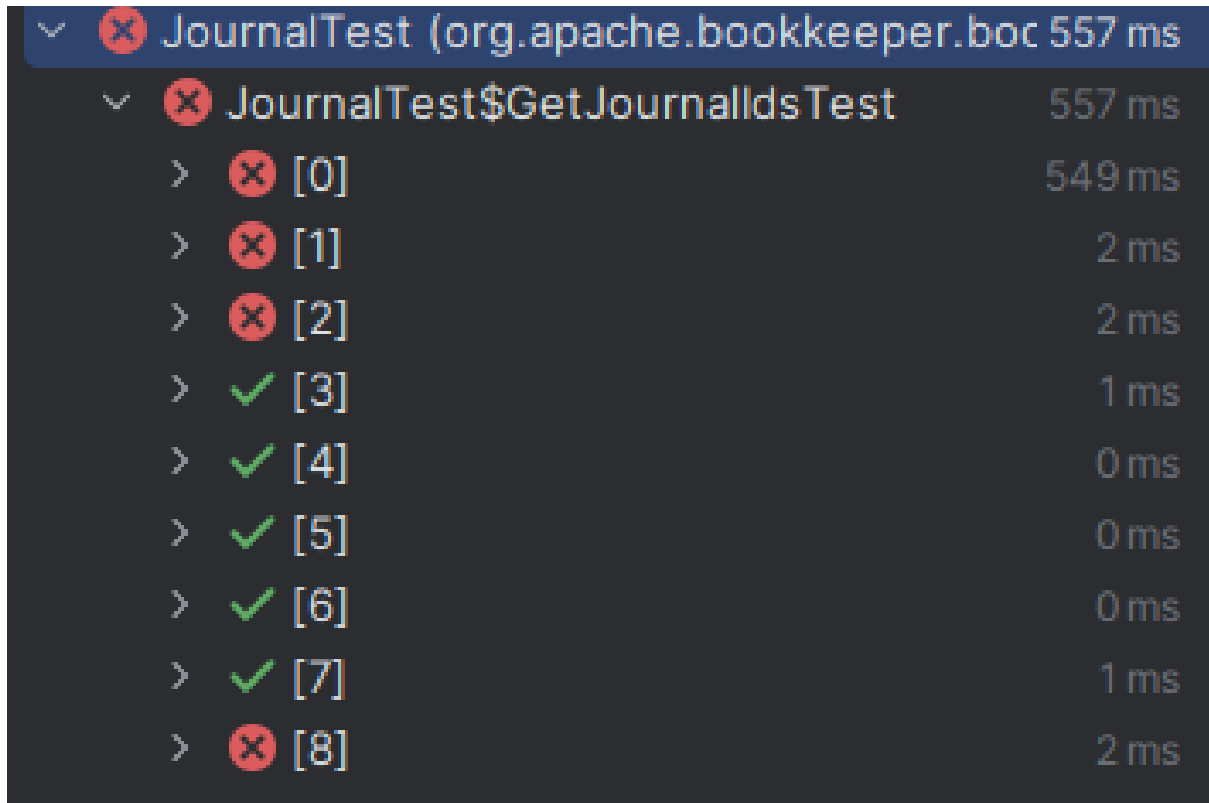


Figura 11: Mutazione inserita nel codice sorgente di Bookkeeper

```

127  * lucia *
128  @Parameterized.Parameters
129  public static Collection<Object[]> getTestParameters() {
130      return Arrays.asList(new Object[][]{
131          {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), JournalIdFilterType.JOURNAL_ROLLING_FILTER.getJournalIdFilter()},
132          {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), null},
133          {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_ONE_TEXT_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_TRUE_FILTER.getJournalIdFilter()},
134          {new ArrayList<>(), null, JournalDirType.ONE_TEXT_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_FALSE_FILTER.getJournalIdFilter()},
135          {new ArrayList<>(), null, JournalDirType.NOT_EXISTING_DIR.getJournalDir(), JournalIdFilterType.NEW_FILTER.getJournalIdFilter()},
136          {new ArrayList<>(), null, JournalDirType.LOG_FILE.getJournalDir(), null},
137          {null, Exception.class, null, null},
138          {new ArrayList<>(), null, JournalDirType.VOID_DIR.getJournalDir(), null},
139          {new ArrayList<>(Collections.singletonList(2L)), null, JournalDirType.ADF_DIR.getJournalDir(), JournalIdFilterType.ADF_FILTER.getJournalIdFilter()},
140          {new ArrayList<>(Arrays.asList(1L, 2L)), null, JournalDirType.ADF_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_TRUE_FILTER.getJournalIdFilter()}
141      });
142  }
143  * lucia *
144  @Before
145  public void setUp() {
146      MockitoAnnotations.initMocks(this);
147      if(journalDir != null && journalDir.getPath().substring(beginIndex, journalDir.getPath().length() - 8).equals("void_dir")) {
148          boolean success = journalDir.mkdir();
149          if(!success) {
150              System.out.println("Could not create void dir");
151          } else {
152              System.out.println("Created void dir");
153          }
154      }
155      if(journalDir != null && journalDir.getPath().equals(JournalDirType.ADF_DIR.getJournalDir().getPath())) {
156          File[] output = journalDir.listFiles();
157          reverse(output);
158          System.out.println("reversed output:\n" + Arrays.toString(output));
159          journalDir = Mockito.spy(journalDir);
160          doReturn(output).when(journalDir).listFiles();
161      }
162  }
163  * lucia *

```

Figura 12: Codice sorgente di JournalTest aggiornato con il mock

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxt	Missed Lines	Missed Methods
readHeader()	<div><div></div></div>	61%	<div><div></div></div>	54%	8 12	13 41	0 1

Figura 13: Jacoco coverage readHeader

Tabella 3: Casi di test aggiornati per coprire l'arco (2,4)

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	JournalRollingFilter	[1]
Directory contenente 1 file di log e 1 file di testo	Filtro sempre True	[1]
Directory contenente 1 file di testo	Filtro sempre False	[]
path non esistente	MyFilter	[]
Path di un file di log	null	Exception
null	null	Exception
Directory contenente 1 file di log	null	Exception

Tabella 4: Test per aumentare la condition coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory vuota	null	[]

Tabella 5: Casi di test aggiornati per aumentare la condition coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	JournalRollingFilter	[1]
Directory contenente 1 file di log e 1 file di testo	Filtro sempre True	[1]
Directory contenente 1 file di testo	Filtro sempre False	[]
path non esistente	MyFilter	[]
Path di un file di log	null	Exception
null	null	Exception
Directory contenente 1 file di log	null	Exception
Directory vuota	null	[]

Tabella 6: Test per aumentare la all-use coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory con un log con id 2 e un file con id 1	Filtro con condizione $\neq 1$	[2]

Tabella 7: Test per aumentare la all-use coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory con un log con id 2 e un log con id 1	Filtro sempre true	[1, 2]

Tabella 8: Test Suite derivante dalla category partition e dalla boundary analysis

Magic Bytes	Master key	Len of master key	Risultato Atteso
BKLE		new byte[]	1