Bookkepper test

Luca Falasca

7 settembre 2023

Indice

Elenco delle figure

Elenco delle tabelle

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 probabilemente per avere molti test non rilevanti e vanno a coprire scenari già coperti, e quindi sarebbero inutili.
Enumeriano ora i casi di test derivanti da quest prima 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 critedi di adeguatezza.

Criteri di adeguatezza black box

documentazione a disposizione:
javadoc:

```
/**
* List all journal ids by a specified journal id filer.
*
* @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 diagrammma 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) $\rightarrow$ 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) e 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.
$CU_c + PU_c = 18$
$(CU + PU) - (CU_{\text{f}} + PU_{\text{f}}) = 18 + 2 = 20$

$$\frac{CU_{\text{c}} + PU_{\text{c}}}{(CU + PU) - (CU_{\text{f}} + PU_{\text{f}})} = \frac{18}{20} = 0.9$$

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 contentuto 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 6. 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 probabilemente dovuto ad fatto che il metodo listJournalIds è un metodo statico e per il modo in cui pit inserisce le mutazione 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.

Quindi possiamo calcolare la mutation coverage come segue:

$$\frac{Mutazionirilevate}{Mutazionitotali} = \frac{6}{7} = 0.85$$

Per aumentare la coverage vado quindi a modificare la test suite per coprire anche la mutazione a riga 125. il

2.2 Classe 2

2.2.1 Descrizione

2.2.2 Category Partition

2.2.3 Boundary Analysis

2.2.4 Adequacy Control Flow

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

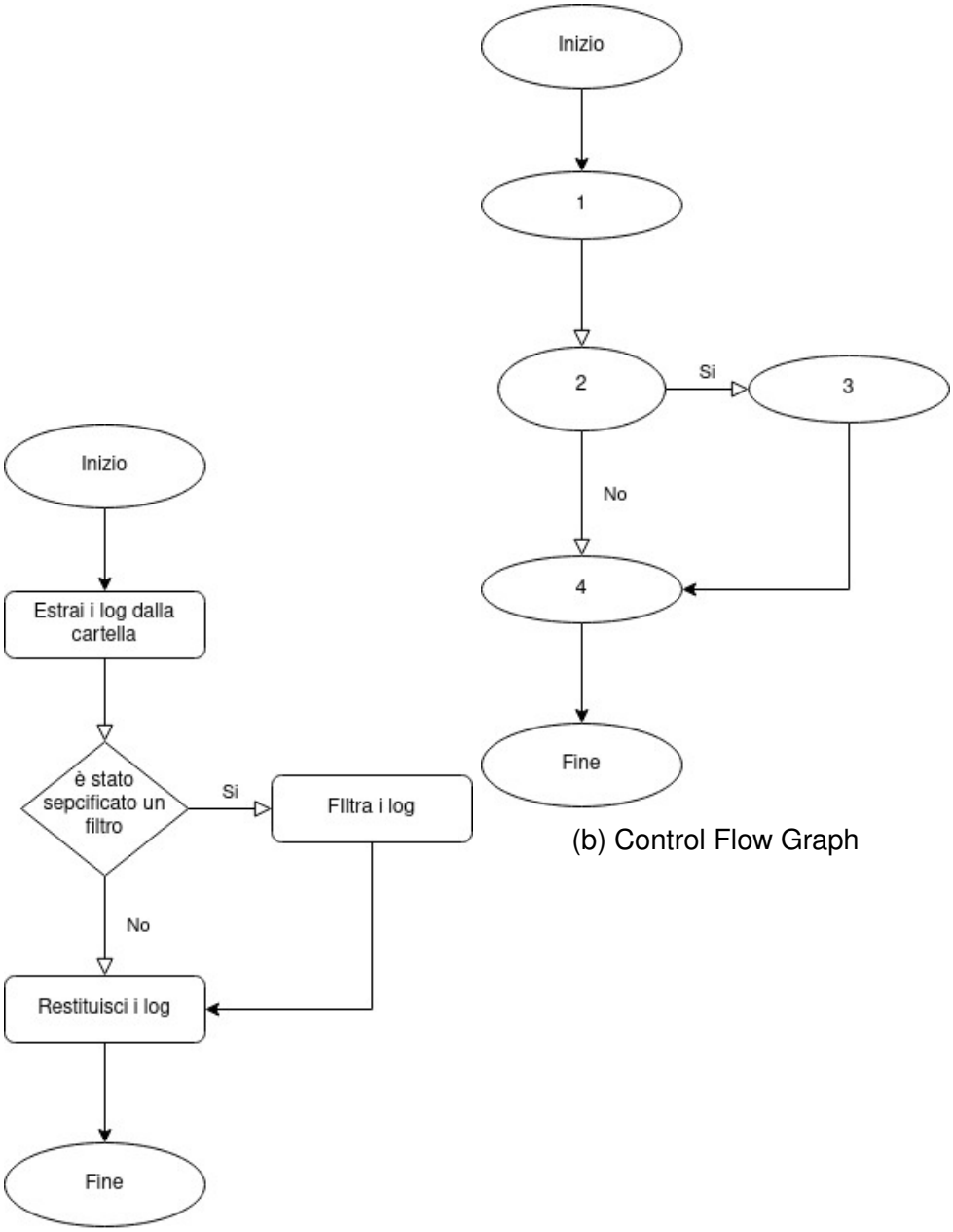
3.2.4 Adequacy Control Flow

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



(a) Functional diagram

(b) Control Flow Graph

Figura 1: Diagrammi di listJournalIds

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

Figura 2: Jacoco coverage dilistJournalIds

Figura 3: Jacoco coverage dilistJournalIds

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

Figura 4: Jacoco coverage dilistJournalIds

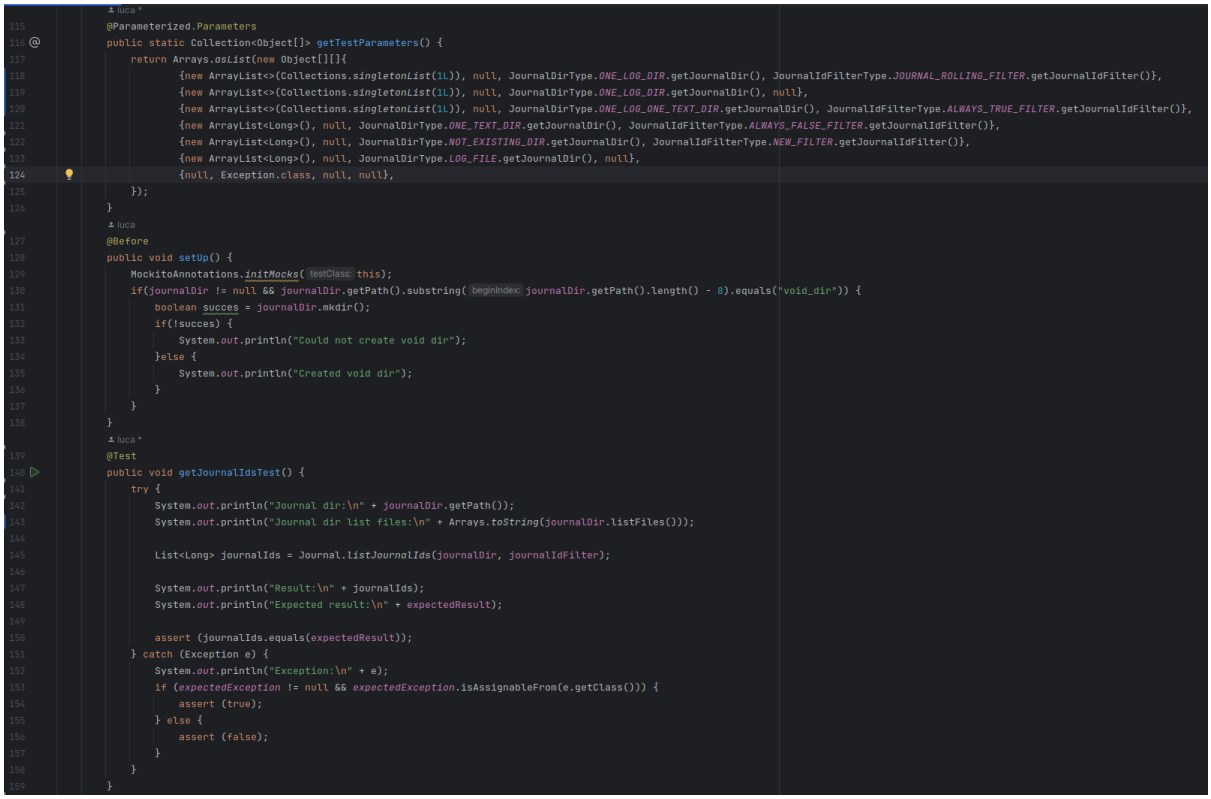


Figura 5: Codice sorgente di JournalTest

```
-<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"/>
```

Figura 7: Badua coverage dilistJournalIds

Journal

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

Figura 8: Jacoco coverage dilistJournalIds

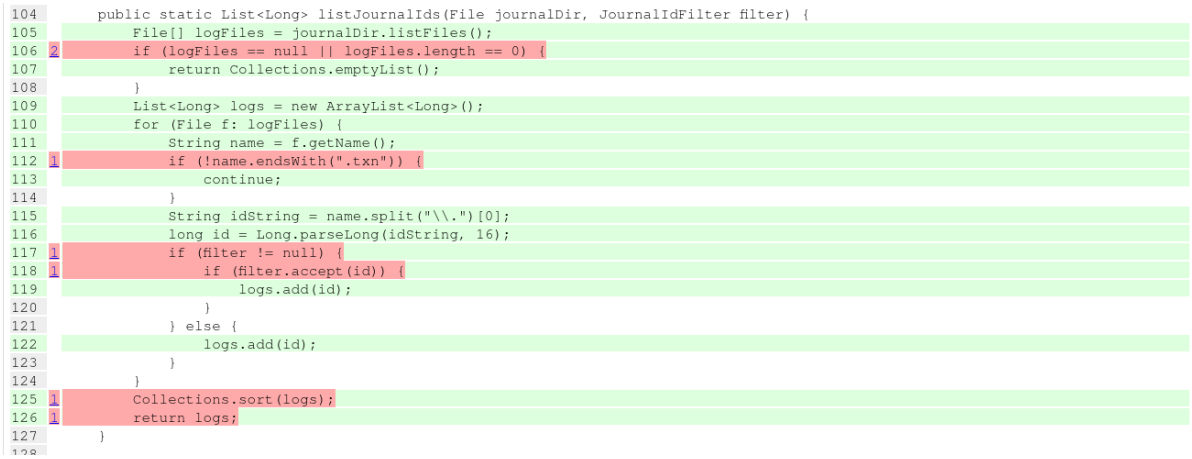


Figura 9: Pitest coverage dilistJournalIds

Figura 10: Mutazione inserita nel codice sorgente di Bookkeeper

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

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

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 1 file con id 1 e un file con id 2	Filtro con condizione ¿1	[ ]

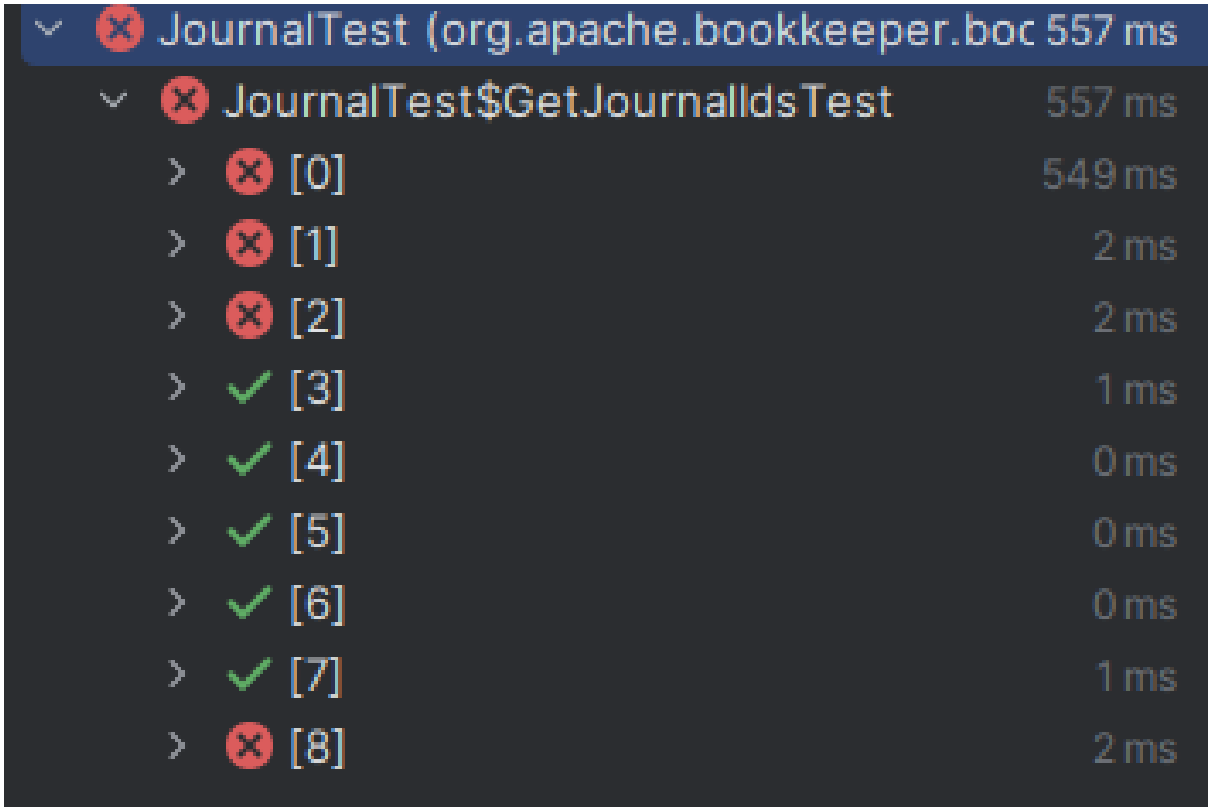


Figura 11: Mutazione inserita nel codice sorgente di Bookkeeper