Report ISW2 Modulo SW Testing

Valentina Falaschi - 0295947

Introduzione

L'obiettivo di questo report è quello di mostrare l'attività di software testing svolta sui progetti Apache open source: Bookkeeper e Avro.

Bookkeeper è un servizio di storage scalabile, tollerante ai guasti e a bassa latenza, ottimizzato per carichi di lavoro real-time.

Avro è un sistema di serializzazione di dati che fornisce:

- Numerose strutture dati;
- Un formato dati binario compatto e veloce;
- File per archiviare dati persistenti;
- Chiamate di procedura remota;
- Semplice integrazione con linguaggi dinamici.

Configurazione preliminare

L'attività di testing è stata eseguita sul sistema operativo Xubuntu, sul quale è stato necessario installare Open-jdk11 e Maven per permettere la build dei progetti.
Come ambiente di sviluppo si è utilizzato Intellij IDEA. Entrambi i progetti sono stati inizialmente clonati in una directory locale attraverso il comando *git clone*.
Successivamente sono stati cancellati tutti i test già presenti, facendo attenzione a non eliminare quelli necessari alla corretta configurazione del progetto. Dopo aver verificato, attraverso il comando *mvn clean install*, che la build in locale fosse andata a buon fine, si è caricato il progetto sull'account personale di Github. Infine sono stati introdotti i file travis.yml e sonar-projects.properties, che hanno permesso l'integrazione con Travis CI e SonarCloud.

Nel seguito verrà illustrata in dettaglio l'attività di software testing svolta su entrambi i progetti.

1. Scelta delle classi

Per Bookkeeper si sono testate 2 classi differenti: FileInfo.Java e Bookie.Java, entrambe contenute nel package *org.apache.bookkeeper.bookie*. La prima classe è stata scelta tenendo conto dei risultati ottenuti dal progetto del professor Falessi, dai quali si è visto che sebbene non risulti quasi mai essere una classe difettosa, presenta numerose

modifiche, a livello di codice, nelle prime release del progetto. La seconda classe invece è stata scelta poiché più comprensibile dal punto di vista del suo comportamento.

Per Avro si è scelto di testare metodi provenienti da 3 classi differenti: RecordBuilderBase.java contenuta nel package *org.apache.avro.data*, BinaryData.java contenuta nel package *org.apache.avro.io* ed infine SpecificData.java contenuta nel package *org.apache.avro.specific*.
Le prime due classi sono state scelte poiché più comprensibili dal punto di vista del loro comportamento, grazie anche alla documentazione presente. La terza classe invece è stata scelta in quanto, tenendo conto dei risultati ottenuti dal progetto del professor Falessi, si è notato che essa presenta svariate modifiche in diverse release quali ad esempio il numero medio di righe modificate. Di seguito verranno analizzate più in dettaglio le classi.

Bookkeeper

-FileInfo

In questa classe si introduce il termine di "ledger", che rappresenta l'unità minima di storage in Bookkeeper. Inoltre si parla di "file di indice", che viene creato per ciascun ledger, e comprende un'intestazione e diverse pagine di indice, le quali registrano gli offset dei dati registrati. La classe FileInfo.java gestisce il file di indice di un ledger. I metodi analizzati per questa classe sono:

- Private int readAbsuolute(ByteBuffer bb, long start, boolean bestEffort)
- Public long write(ByteBuffer[] buffs, long position)

Il primo metodo legge i dati, da un file, a partire dalla posizione *start* per poi trasferirli nel ByteBuffer *bb*. *BestEffort* invece è un flag che indica la presenza di una lettura di tipo best-effort. Nel caso in cui i dati letti fossero minori o uguali di zero e *bestEffort=true,* allora ritorna il totale dei dati letti, altrimenti ritorna un'eccezione.

Il secondo metodo scrive dati su un file, contenuti in un array di ByteBuffer *buffs*, a partire da una certa *position.* Il valore di ritorno indica quanti byte sono stati scritti.

-Bookie

Questa classe si occupa dell'implementazione di Bookies, ovvero server di Bookkeeper che gestiscono i Ledger. I metodi testati sono:

- Public static boolean Format(ServerConfiguration conf, boolean isInteractive, boolean force)
- Public static void checkDirectoryStructure(File dir)

Il primo metodo si occupa di formattare dati del bookie server. Si considera la configurazione del server *conf* e si verifica la presenza o meno di vecchi dati. Nel caso in cui questi fossero presenti vengono cancellati, senza chiedere conferma all'utente, se

force=true e *isInteractive=false*. Il valore di ritorno indica se la procedura è andata a buon fine.

Il secondo metodo controlla se la directory *dir* esiste, se non esiste viene creata.

Avro

-BinaryData

La classe BinaryData.java fornisce gli strumenti per confrontare e codificare dati.
I metodi testati sono:

- Public static int CompareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)
- Public static int EncodeInt(int n, byte[] buf, int pos)

Il primo metodo consiste nel confronto lessicografico dell'array di byte *b1* con l'array di byte *b2*. I valori *s1* e *s2* forniscono la posizione di start dell'array dalla quale cominciare il confronto, mentre i valori *s1+l1* e *s2+l2* la posizione di end. In particolare si procede confrontando *b1[s1]* con *b2[s2],* fino ad arrivare al confronto tra *b1[s1+l1]* con *b2[s2+l2]* .
Il metodo ritorna un intero: positivo se *b1* è maggiore di *b2*, uguale a zero se *b1* è uguale a *b2* e negativo se *b1* è minore di *b2*.

Il secondo metodo codifica un intero *n* in un array di byte *buf* ad una data posizione *pos.*
Il metodo ritorna un intero, il quale indica il numero di byte scritti nel buffer.

-SpecificData

Avro si basa su schemi. Quando i dati vengono letti lo schema utilizzato durante la scrittura è sempre presente. Questo consente di scrivere ogni dato senza overhead per-value e rende la serializzazione dei dati più veloce.
La classe SpecificData.java fornisce gli strumenti per la creazione e la gestione di schemi.
Il metodo testato è:

- Protected String GetSchemaName(Object datum)

Questo metodo ritorna il nome dello schema, dato in input un tipo di oggetto (*datum*) .

-RecordBuilderBase

La classe astratta RecordBuilderBase.java fornisce gli strumenti per la creazione e gestione di record di un dato tipo. Il metodo testato per questa classe è:

- Protected static boolean isValidValue(Field f, Object value)

Questo metodo verifica se il valore *value* è valido per un certo campo *field* specificato.
Ritorna vero se il valore è valido per il campo dato in input, falso altrimenti.

2. Category Partition

Successivamente alla scelta delle classi e alla loro comprensione si è proceduto con la category partition. La category partition, consiste nello studiare il dominio dei valori di input e partizionare lo spazio dei valori ammissibili basandosi sulla conoscenza esplicita o implicita di range, stringe, enumerazioni, array e tipi di dato complesso. Il criterio scelto, per identificare i parametri di input dei test, è unidimensionale, ovvero ogni parametro di input al test è considerato in maniera indipendente dagli altri. Si è scelto questo approccio poiché ritenuto più semplice rispetto al criterio multidimensionale, che consiste nel prodotto cartesiano dei parametri di input. Quest'ultimo infatti avrebbe portato a generare un numero elevato di casi di test, avendo metodi con più di un parametro. Inoltre dopo aver partizionato il dominio di riferimento in modo unidimensionale, si è cercato di identificare, dove possibile, i boundary-value, ovvero i valori di confine. Questo perché, come si apprende dalla teoria, l'esperienza comprovata anche da studi empirci suggerisce che i più comuni bug di programmazione sono introdotti su valori "a confine" delle classi di equivalenza.

Bookkeeper

-FileInfo

Il metodo *Private int readAbsuolute(ByteBuffer bb, long start, boolean bestEffort)*, come detto precedentemente, legge dati da un file e li trasferisce in un ByteBuffer. Per il valore booleano si è considerato il caso *true* e *false*. Per il parametro *start* sono state considerate le seguenti partizioni:

- Valori negativi, considerando per l'analisi boundary-value il valore -1;
- Valori nulli, considerando per l'analisi boundary-value il valore 0;
- Valori positivi, considerando per l'analisi boundary-value il valore 1.

Per il parametro *bb* sono state considerate le seguenti partizioni:

- Istanza valida, con ByteBuffer con capacità positiva;
- Istanza valida, con ByteBuffer con capacità nulla;
- Istanza non valida, valore null del ByteBuffer.

Il metodo *Public long write(ByteBuffer[] buffs, long position)*, prende in input un array di ByteBuffer, per il quale si sono considerate le stesse partizioni del parametro *bb* del metodo precedente. Inoltre prende in input un dato di tipo long, per il quale si sono considerate le stesse partizioni del dato *start* del metodo precedente.

-Bookie

Il metodo *Public static boolean Format(ServerConfiguration conf, boolean isInteractive, boolean force)*, prende in input il dato *conf,* per il quale si sono considerate le seguenti partizioni:

- Istanza valida, con configurazione del server inizializzata con la presenza di directory per memorizzare i journal files;
- Istanza valida, con configurazione del server inizializzata senza directory per memorizzare i journal files;
- Istanza non valida, con valore null di *conf*.

Per i parametri boolean invece si sono considerati valori true e false, facendo attenzione, nel caso di istanza valida di *conf*, a non settare a true il valore del dato *isInteractive*. Questo, infatti, avrebbe portato ad un loop, in quanto si chiede conferma all'utente se poter o meno cancellare i vecchi dati.

Il metodo *Public static void checkDirectoryStructure(File dir)* prende in input il parametro *dir,* per questo dato si è considerata la seguente partizione:

- Istanza valida, la directory non esiste;
- Istanza valida, la directory già esiste;
- Istanza non valida, valore null di *dir*.

Avro

-BinaryData

Il metodo *Public static int CompareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)*, come detto precedentemente, confronta due array di byte e prende in input array e interi. Per quanto riguarda gli array si sono considerati i casi in cui:

- I due array sono uguali ed entrambi vuoti;
- I due array sono diversi.

Per i valori interi *s1* e *s2*, che indicano i valori di start dai quali cominciare il confronto, sono state considerate le seguenti partizioni:

- Valori negativi, considerando per l'analisi boundary-value il valore -1;
- Valori nulli, considerando per l'analisi boundary-value il valore 0;
- Valori positivi, considerando per l'analisi boundary-value il valore 1.

Per i valori interi *l1* e *l2*, che vengono sommati ai valori di start al fine di ottenere la posizione di end nella quale terminare il confronto, si sono nuovamente considerati elementi negativi, positivi e nulli.

Il metodo *Public static int EncodeInt(int n, byte[] buf, int pos)*, come detto precedentemente, codifica interi e prende in input due interi e un array di byte. Per i valori interi *n* e *pos,* che indicano rispettivamente l'intero da codificare e la posizione del buffer che conterrà tale codifica, sono state considerate le seguenti partizioni:

- Valori negativi, considerando per l'analisi boundary-value il valore -1;
- Valori nulli, considerando per l'analisi boundary-value il valore 0;

- Valori positivi, considerando per l'analisi boundary-value il valore 1.

Mentre per quanto riguardo il parametro di input *buf* è stata considerata la seguente partizione:

- Array inizializzati a zero;
- Array inizializzati con valori positivi.

-SpecificData

Il metodo *Protected String GetSchemaName(Object datum)*, come detto precedentemente ritorna il nome dello schema in base al parametro di input dato. Per il dato in input si è considerato un caso di test per ciascuno dei seguenti tipi di schema:

- INT;
- NULL;
- DOUBLE;
- BOOLEAN;
- FLOAT;
- LONG;
- ARRAY;
- FIXED;
- RECORD;
- ENUM;
- MAP;

Oltre a questi si è considerato anche il caso di test in cui viene inizializzato un new Object().

-RecordBuilderBase

Il metodo *Protected static boolean isValidValue(Field f, Object value)*, come detto precedentemente, verifica se il valore *value* è valido per un certo campo *field* specificato. Per l'Object *value* si è considerata la seguente partizione:

- Istanza non valida, valore null;
- Istanza valida, new Object().

Il parametro *field* riguarda il campo di un certo schema, per questo si è considerato un caso di test per ciascuno dei seguenti tipi di schema: NULL, INT, LONG, FLOAT, DOUBLE, BYTES, BOOLEAN, STRING, UNION.

3. Implementazione test

Per l'implementazione dei test si è utilizzato JUnit 4, un framework che supporta l'implementazione di Unit-test in ambienti Java. Per ciascun test si è: etichettata la classe con *@RunWith(value=Parametrized.class)*, così che la classe di test definisse il runner JUnit; Si sono dichiarati i parametri della classe di test come attributi privati; Si è creato il

metodo di configurazione che torna una *Java.util.Collection* ed infine si è generato un costruttore avente tanti argomenti quanti sono i parametri. Ogni caso di test è stato poi etichettato con la notazione *@Test*. In alcuni casi si è ricorso anche all'uso di annotazioni quali *@Before* e *@After* per specificare configurazioni o azioni da eseguire una sola volta, rispettivamente prima e dopo l'esecuzione di tutti i metodi annotati come *@Test*.

Bookkeeper

-FileInfo

1. Private int readAbsuolute(ByteBuffer bb, long start, boolean bestEffort), per questo metodo si è verificato che ritornassero effettivamente il numero di byte letti da un file, nel caso in cui questi fossero un numero >=0 o nel caso in cui fossero un numero <=0 e *bestEffort=True*. Nel caso in cui invece i byte letti fossero <=0 e *bestEffort=false*, come ci si aspetta, il metodo ritorna un'eccezione. Il file, dal quale leggere i byte, è stato appositamente creato nel metodo *setUp()*, annotato con *@Before*, e cancellato nel metodo *delete()*, annotato con *@After.* Il caso di istanza non valida, invece, non è stato gestito nel metodo e ritorna un'eccezione.

2. Public long write(ByteBuffer[] buffs, long position), per questo metodo si è verificato che il valore di ritorno fosse effettivamente il numero di byte scritti nel file. Il caso di istanza non valida non è stato gestito nel metodo e viene ritornata un'eccezione. Anche in questo caso come nel metodo precedente il file su cui scrivere è stato creato prima dell'esecuzione del test e cancellato dopo la fine dell'esecuzione.

-Bookie

1. Public static boolean Format(ServerConfiguration conf, boolean isInteractive, boolean force), per questo metodo il test ha verificato che, nel caso di configurazione del server inizializzata con la presenza di directory per memorizzare i journal files, il metodo ritornasse true nel caso in cui *force=true* e false nel caso in cui *force=false*. Mentre ritorna sempre vero nel caso di configurazione del server inizializzata senza la presenza di directory. Il caso di istanza non valida, invece, non è stato gestito nel metodo e viene ritornata un'eccezione.

2. Public static void checkDirectoryStructure(File dir), per questo metodo con un primo test è stato verificato che, nel caso di directory non esistente questa venisse effettivamente creata. Con un secondo test si è verificato che nel caso di directory già esistente il metodo non ritornasse alcun valore né creasse la directory. Il caso di istanza non valida, invece, non è stato gestito nel metodo e viene ritornata un'eccezione.

Avro

-BinaryData

1. Public static int CompareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2), per questo metodo si è semplicemente testato che in base ai parametri di input, scelti nella fase di category partition, venisse restituito zero nel caso in cui gli array di byte confrontati fossero uguali e -1 nel caso in cui si sono dati in input array diversi.

2. *Public static int EncodeInt(int n, byte[] buf, int pos)*, per i valori di input, scelti nella fase di category partition, si è verificato che il metodo ritornasse effettivamente il numero di byte scritti nel buffer a seguito alla codifica del valore *n*. Il caso in cui *byte[] buf= new byte[0]* non è stato gestito, infatti viene restituita l'eccezione: *Index 1 out of bounds for lenght 0.*

-SpecificData

1. Protected String GetSchemaName(Object datum), i test svolti hanno mostrato che il metodo ritorna correttamente i nomi dei vari tipi di *datum* assegnati, e ritorna *null* nel caso di valori nulli del parametro in input*.*

-RecordBuilderBase

1. Protected static boolean isValidValue(Field f, Object value), per questo metodo si sono implementati due test differenti, uno per tutti i tipi di schema ad eccezione di UNION e uno per il tipo di schema UNION. In particolare per il secondo test si è creato un esempio di schema di tipo UNION, in cui si unisce uno schema di tipo STRING con uno di tipo LONG ed in questo caso i valori di ritorno sono stati:

- True, nel caso in cui *value* è diverso da *null*;
- False, nel caso in cui *value* è uguale a *null*.

Mentre nel caso di uno schema di tipo UNION, in cui si unisce uno schema di qualsiasi tipo con uno di tipo NULL, allora il valore di ritorno è sempre true sia nel caso di *value* uguale a *null* sia nel caso in cui *value* è diverso da *null.* I valori di ritorno rispettano il comportamento del metodo.

4. Adeguatezza dei test e Mutazioni

In questa fase si è utilizzato il tool Jacoco per valutare criteri di adeguatezza control-flow. Per includere tale tool è stato necessario modificare più file pom.xml, aggiungendo un profile nel pom.xml del modulo principale ed introducendo un nuovo modulo in cui confluisce il report generato da Jacoco quando viene attivato il profile relativo.

Infine si è usato PIT per applicare mutazioni alle classi e metodi testati e vedere come i test reagiscano a tali mutazioni, al fine di stimarne la bontà/adeguatezza e di rilevare criticità sfuggite nelle fasi precedenti. Per introdurre PIT è stato necessario modificare i file pom.xml aggiungendo un nuovo profile.

Di seguito si mostra l'analisi relativa ai report generati da Jacoco e Pit.

Bookkeeper

-FileInfo

1. Private int readAbsuolute(ByteBuffer bb, long start, boolean bestEffort),

Jacoco: Come mostrato in fig. 1 il report di Jacoco mostra alcuni branch non coperti. Si è quindi introdotto un nuovo caso di test in cui i dati letti dal file fossero <=0 e il parametro *bestEffort* settato a *false*. Così da coprire il branch a riga 431. Inoltre è stato aggiunto un ulteriore caso di test in cui il file da cui leggere i dati fosse uguale a *null*. Così da coprire il branch a riga 420. In fig. 2 è riportato il report generato in seguito a tali modifiche.

PIT: Il report generato da PIT è visibile in fig. 3.

2. Public long write(ByteBuffer[] buffs, long position)

Jacoco: Il report relativo a Jcoco è mostrato in fig. 4.

PIT: Il report relativo a PIT, in fig. 5 mostra diverse mutazioni non *killed.* In particolare attraverso l'aggiunta di due casi di test, si è riusciti a killare la mutazione a riga 498, e la mutazione a riga 488, facendo in modo che il valore dei byte letti risultasse pari a 0 e 1. In fig. 6 è mostrato il report a seguito all'aggiunta dei nuovi casi di test.

-Bookie

1. Public static boolean Format(ServerConfiguration conf, boolean isInteractive, boolean force)

I report relativi a Jacoco e PIT sono mostrati in figura fig. 7 e in fig. 8, non si è riusciti a coprire i branch mancanti poiché, ad esempio, sarebbe stato necessario gestire il caso di test in cui *isInteractive=true* ma questo, come spiegato precedentemente, avrebbe portato ad un loop all'interno del metodo.

2. Public static void checkDirectoryStructure(File dir)

Jacoco: il report generato da Jacoco mostrava alcuni branch mancanti che sono stati coperti con l'introduzione di due nuovi casi di test. Uno in cui una vecchia directory conteneva file con estensione *.txn*, *.idx* e *.log*. E uno in cui si simula il fallimento della creazione della directory tramite il framework mockito. I report relativi a Jacoco prima e dopo l'introduzione dei nuovi casi di test sono visibili in fig.9 e in fig. 10. I report Relativi a Pit prima e dopo l'introduzione dei nuovi casi di test sono visibili in fig.11 e in fig. 12.

Avro

-BinaryData

1. Public static int CompareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2)

Jacoco: Come mostrato in fig. 13, il report di Jacoco evidenziano un branch mancante a riga 185. È stato possibile coprire tale branch aggiungendo un ulteriore caso di test in cui *b1>b2*. In seguito a tale aggiunta è stato generato nuovamente il report di Jacoco, visibile in fig. 14.

PIT: Il report relativo alle mutazioni, visibile in fig. 15, mostra diverse mutazione *not killed* tali mutazioni sono poi risultate *killed*, come si vede in fig. 16*,* in seguito all'aggiunta del nuovo caso di test.

2. Public static int EncodeInt(int n, byte[] buf, int pos)

Jacoco: Come mostrato in fig. 17*,* il report di Jacoco evidenzia svariate linee di codice non coperte. Sono quindi stati aggiunti ulteriori casi di test, uno in cui si è dato in input un numero sufficientemente grande in grado di ricoprire tutti gli statement condizionali presenti nel metodo. Mentre per gli altri 3 casi di test sono stati dati in input numeri appositi che andavano a coprire le restanti linee di codice. Il report generato in seguito a tali modifiche è visibile in fig. 18

PIT: I report relativi alle mutazioni, prima e dopo l'aggiunta dei nuovi casi di test sono visibili in fig. 19 e in fig. 20.

-*SpecificData*

1. Protected String GetSchemaName(Object datum)

Jacoco: Come mostrato in fig. 21, il report di Jacoco mostra un "*missed branch"* sulla condizione *if (isStringable).* È stato quindi introdotto un nuovo caso di test dove si sono considerati oggetti di tipo *Stringable,* tra cui ad esempio oggetti di tipo BigDecimal.

In seguito all'aggiunta di questo caso di test, si è raggiunta la totale copertura del metodo, come mostrato in fig. 22. I report relativi a PIT, prima e dopo l'aggiunta dei nuovi casi di test*,* sono mostrati in fig. 23 e fig. 24.

-*RecordBuilderBase*

1. Protected static boolean isValidValue(Field f, Object value)

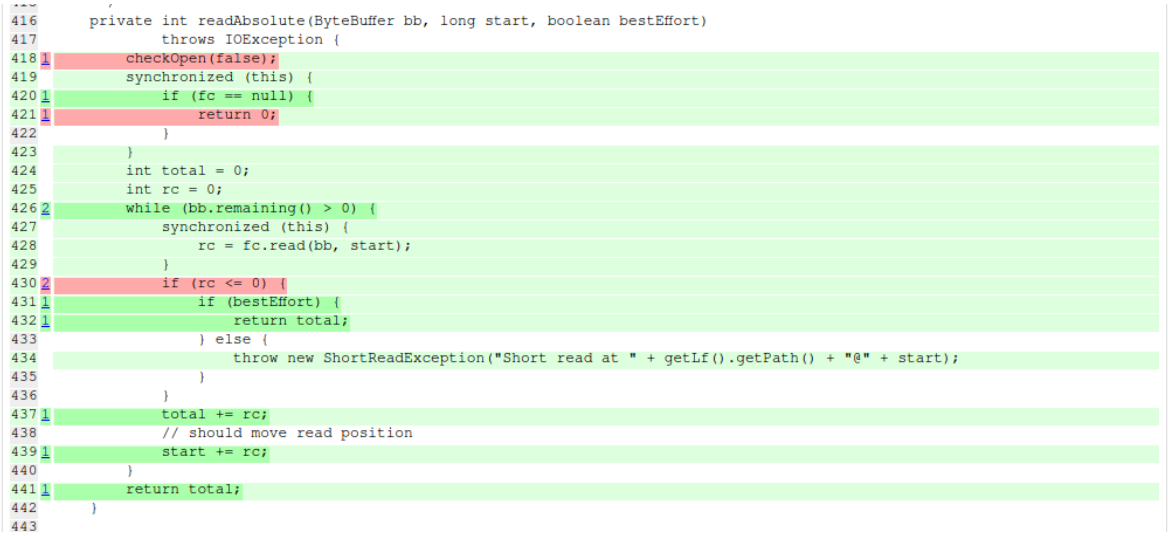
In questo caso fin da subito sia Jacoco che PIT evidenziano una copertura totale del metodo. I report sono visibili in fig. 25 e fig. 26*.*

```
415.     */
416.    private int readAbsolute(ByteBuffer bb, long start, boolean bestEffort)
417.            throws IOException {
418.        checkOpen(false);
419.        synchronized (this) {
420.            if (fc == null) {
421.                return 0;
422.            }
423.        }
424.        int total = 0;
425.        int rc = 0;
426.        while (bb.remaining() > 0) {
427.            synchronized (this) {
428.                rc = fc.read(bb, start);
429.            }
430.            if (rc <= 0) {
431.                if (bestEffort) {
432.                    return total;
433.                } else {
434.                    throw new ShortReadException("Short read at " + getLf().getPath() + "@" + start);
435.                }
436.            }
437.            total += rc;
438.            // should move read position
439.            start += rc;
440.        }
441.        return total;
442.    }
```

Fig. 1

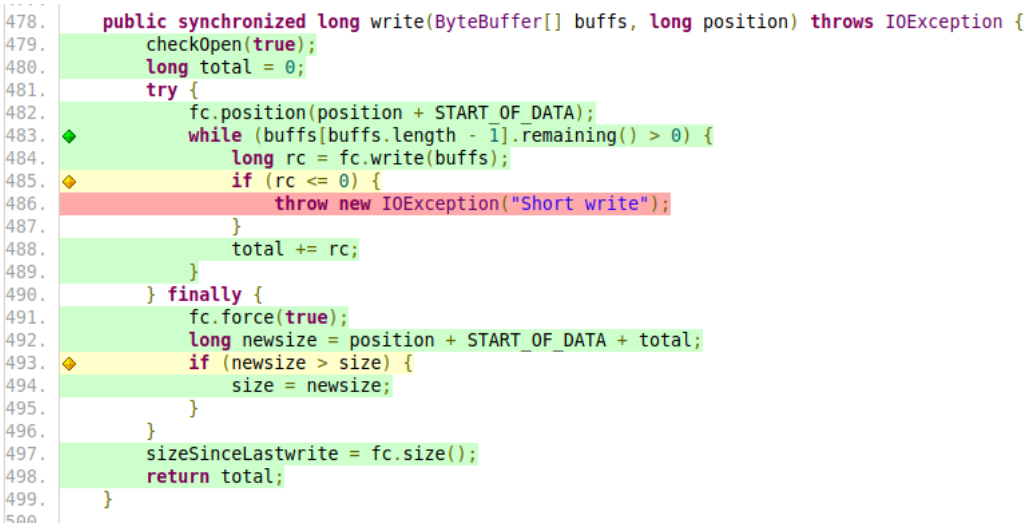
```
415.     */
416.    private int readAbsolute(ByteBuffer bb, long start, boolean bestEffort)
417.            throws IOException {
418.        checkOpen(false);
419.        synchronized (this) {
420.            if (fc == null) {
421.                return 0;
422.            }
423.        }
424.        int total = 0;
425.        int rc = 0;
426.        while (bb.remaining() > 0) {
427.            synchronized (this) {
428.                rc = fc.read(bb, start);
429.            }
430.            if (rc <= 0) {
431.                if (bestEffort) {
432.                    return total;
433.                } else {
434.                    throw new ShortReadException("Short read at " + getLf().getPath() + "@" + start);
435.                }
436.            }
437.            total += rc;
438.            // should move read position
439.            start += rc;
440.        }
441.        return total;
442.    }
```

Fig. 2



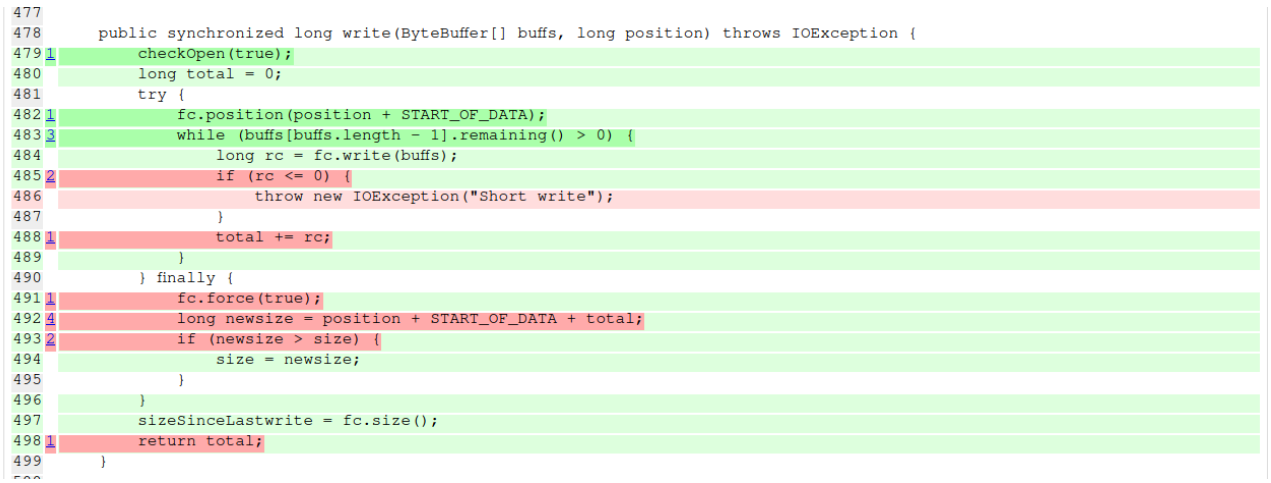
```
416     private int readAbsolute(ByteBuffer bb, long start, boolean bestEffort)
417             throws IOException {
418 1       checkOpen(false);
419         synchronized (this) {
420 1           if (fc == null) {
421 1               return 0;
422             }
423         }
424         int total = 0;
425         int rc = 0;
426 2       while (bb.remaining() > 0) {
427             synchronized (this) {
428                 rc = fc.read(bb, start);
429             }
430 2           if (rc <= 0) {
431 1               if (bestEffort) {
432 1                   return total;
433                 } else {
434                     throw new ShortReadException("Short read at " + getLf().getPath() + "@" + start);
435                 }
436             }
437 1           total += rc;
438             // should move read position
439 1           start += rc;
440         }
441 1       return total;
442     }
443
```

Fig. 3



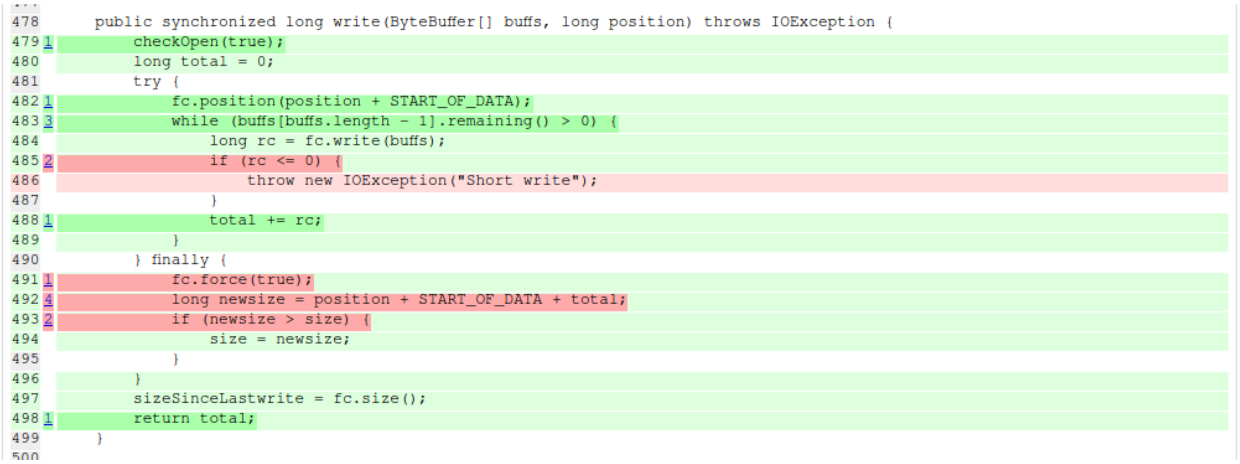
```
478.    public synchronized long write(ByteBuffer[] buffs, long position) throws IOException {
479.        checkOpen(true);
480.        long total = 0;
481.        try {
482.            fc.position(position + START_OF_DATA);
483.            while (buffs[buffs.length - 1].remaining() > 0) {
484.                long rc = fc.write(buffs);
485.                if (rc <= 0) {
486.                    throw new IOException("Short write");
487.                }
488.                total += rc;
489.            }
490.        } finally {
491.            fc.force(true);
492.            long newsize = position + START_OF_DATA + total;
493.            if (newsize > size) {
494.                size = newsize;
495.            }
496.        }
497.        sizeSinceLastwrite = fc.size();
498.        return total;
499.    }
```

Fig. 4



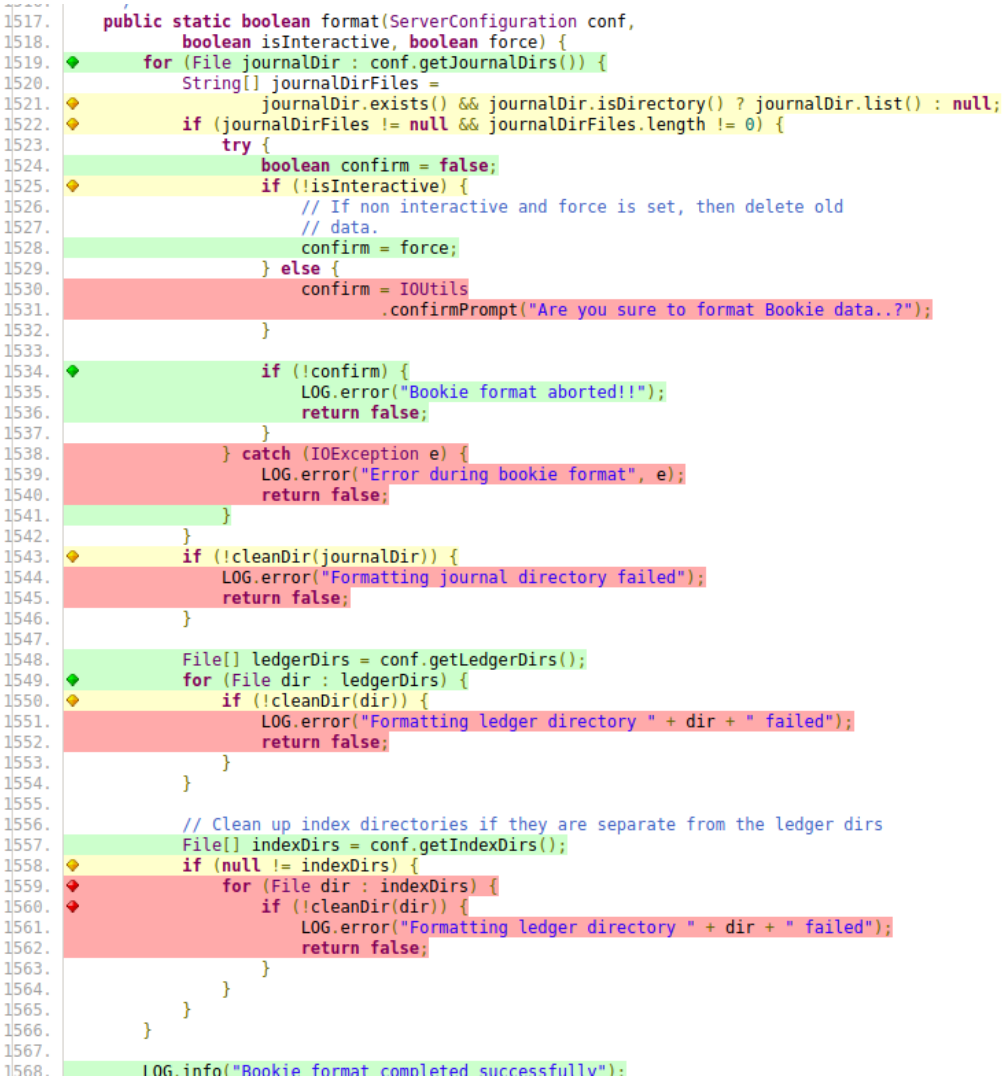
```
477
478     public synchronized long write(ByteBuffer[] buffs, long position) throws IOException {
479 1       checkOpen(true);
480         long total = 0;
481         try {
482 1           fc.position(position + START_OF_DATA);
483 3           while (buffs[buffs.length - 1].remaining() > 0) {
484                 long rc = fc.write(buffs);
485 2               if (rc <= 0) {
486                     throw new IOException("Short write");
487                 }
488 1               total += rc;
489             }
490         } finally {
491 1           fc.force(true);
492 4           long newsize = position + START_OF_DATA + total;
493 2           if (newsize > size) {
494                 size = newsize;
495             }
496         }
497         sizeSinceLastwrite = fc.size();
498 1       return total;
499     }
```

Fig. 5



```
478     public synchronized long write(ByteBuffer[] buffs, long position) throws IOException {
479 1       checkOpen(true);
480         long total = 0;
481         try {
482 1           fc.position(position + START_OF_DATA);
483 3           while (buffs[buffs.length - 1].remaining() > 0) {
484                 long rc = fc.write(buffs);
485 2               if (rc <= 0) {
486                     throw new IOException("Short write");
487                 }
488 1               total += rc;
489             }
490         } finally {
491 1           fc.force(true);
492 4           long newsize = position + START_OF_DATA + total;
493 2           if (newsize > size) {
494                 size = newsize;
495             }
496         }
497         sizeSinceLastwrite = fc.size();
498 1       return total;
499     }
500
```

Fig. 6



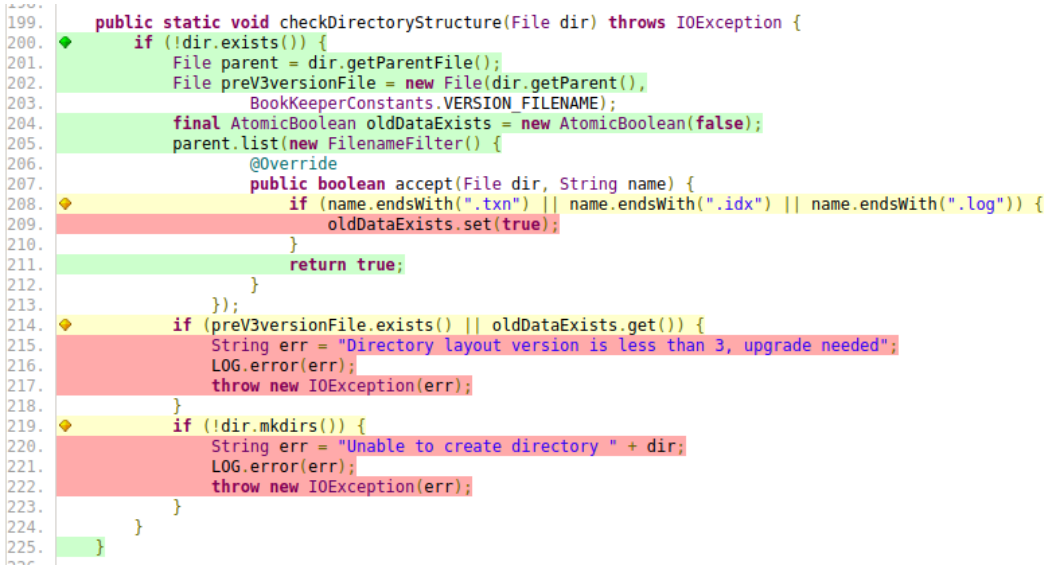
```
1517.   public static boolean format(ServerConfiguration conf,
1518.           boolean isInteractive, boolean force) {
1519.       for (File journalDir : conf.getJournalDirs()) {
1520.           String[] journalDirFiles =
1521.                   journalDir.exists() && journalDir.isDirectory() ? journalDir.list() : null;
1522.           if (journalDirFiles != null && journalDirFiles.length != 0) {
1523.               try {
1524.                   boolean confirm = false;
1525.                   if (!isInteractive) {
1526.                       // If non interactive and force is set, then delete old
1527.                       // data.
1528.                       confirm = force;
1529.                   } else {
1530.                       confirm = IOUtils
1531.                               .confirmPrompt("Are you sure to format Bookie data..?");
1532.                   }
1533.
1534.                   if (!confirm) {
1535.                       LOG.error("Bookie format aborted!!");
1536.                       return false;
1537.                   }
1538.               } catch (IOException e) {
1539.                   LOG.error("Error during bookie format", e);
1540.                   return false;
1541.               }
1542.           }
1543.           if (!cleanDir(journalDir)) {
1544.               LOG.error("Formatting journal directory failed");
1545.               return false;
1546.           }
1547.
1548.           File[] ledgerDirs = conf.getLedgerDirs();
1549.           for (File dir : ledgerDirs) {
1550.               if (!cleanDir(dir)) {
1551.                   LOG.error("Formatting ledger directory " + dir + " failed");
1552.                   return false;
1553.               }
1554.           }
1555.
1556.           // Clean up index directories if they are separate from the ledger dirs
1557.           File[] indexDirs = conf.getIndexDirs();
1558.           if (null != indexDirs) {
1559.               for (File dir : indexDirs) {
1560.                   if (!cleanDir(dir)) {
1561.                       LOG.error("Formatting ledger directory " + dir + " failed");
1562.                       return false;
1563.                   }
1564.               }
1565.           }
1566.       }
1567.
1568.       LOG.info("Bookie format completed successfully");
```

Fig. 7

```
1517     public static boolean format(ServerConfiguration conf,
1518             boolean isInteractive, boolean force) {
1519 1       for (File journalDir : conf.getJournalDirs()) {
1520             String[] journalDirFiles =
1521 2               journalDir.exists() && journalDir.isDirectory() ? journalDir.list() : null;
1522 2           if (journalDirFiles != null && journalDirFiles.length != 0) {
1523                 try {
1524                     boolean confirm = false;
1525 1                   if (!isInteractive) {
1526                         // If non interactive and force is set, then delete old
1527                         // data.
1528                         confirm = force;
1529                     } else {
1530                         confirm = IOUtils
1531                                 .confirmPrompt("Are you sure to format Bookie data..?");
1532                     }
1533
1534 1                   if (!confirm) {
1535                         LOG.error("Bookie format aborted!!");
1536 1                       return false;
1537                     }
1538                 } catch (IOException e) {
1539                     LOG.error("Error during bookie format", e);
1540 1                   return false;
1541                 }
1542             }
1543 1           if (!cleanDir(journalDir)) {
1544                 LOG.error("Formatting journal directory failed");
1545 1               return false;
1546             }
1547
1548             File[] ledgerDirs = conf.getLedgerDirs();
1549             for (File dir : ledgerDirs) {
1550 1               if (!cleanDir(dir)) {
1551                     LOG.error("Formatting ledger directory " + dir + " failed");
1552 1                   return false;
1553                 }
1554             }
1555
1556             // Clean up index directories if they are separate from the ledger dirs
1557             File[] indexDirs = conf.getIndexDirs();
1558 1           if (null != indexDirs) {
1559                 for (File dir : indexDirs) {
1560 1                   if (!cleanDir(dir)) {
1561                         LOG.error("Formatting ledger directory " + dir + " failed");
1562 1                       return false;
1563                     }
1564                 }
1565             }
1566         }
```

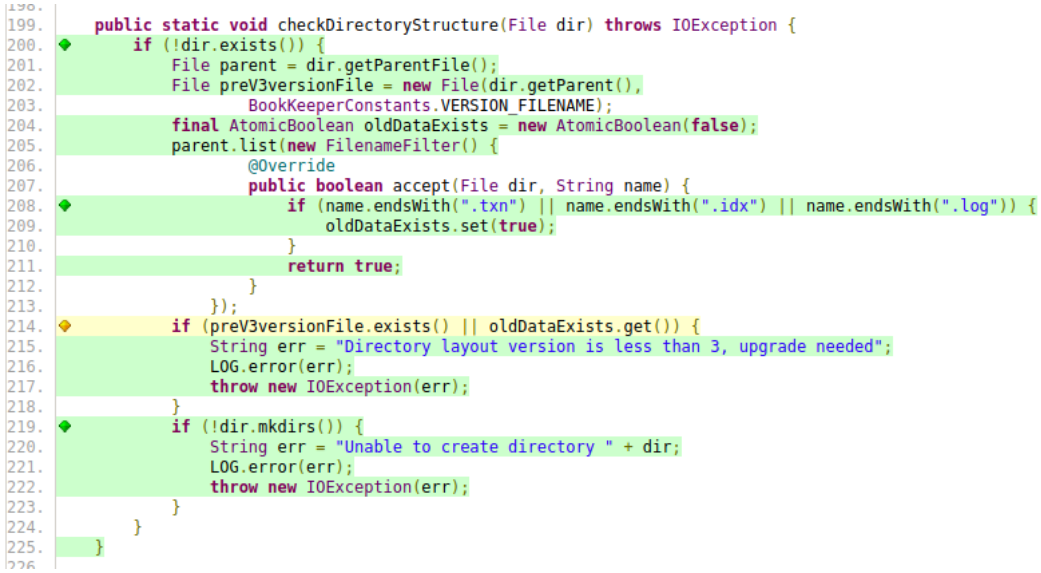
```
1567
1568         LOG.info("Bookie format completed successfully");
1569 1       return true;
1570     }
1571
1572     private static boolean cleanDir(File dir) {
1573 1       if (dir.exists()) {
1574             File[] files = dir.listFiles();
1575 1           if (files != null) {
1576                 for (File child : files) {
1577                     boolean delete = FileUtils.deleteQuietly(child);
1578 1                   if (!delete) {
1579                         LOG.error("Not able to delete " + child);
1580 1                       return false;
1581                     }
1582                 }
1583             }
1584 1       } else if (!dir.mkdirs()) {
1585             LOG.error("Not able to create the directory " + dir);
1586 1           return false;
1587         }
1588 1       return true;
1589     }
1590
```

Fig. 8



```
199.    public static void checkDirectoryStructure(File dir) throws IOException {
200.        if (!dir.exists()) {
201.            File parent = dir.getParentFile();
202.            File preV3versionFile = new File(dir.getParent(),
203.                    BookKeeperConstants.VERSION_FILENAME);
204.            final AtomicBoolean oldDataExists = new AtomicBoolean(false);
205.            parent.list(new FilenameFilter() {
206.                    @Override
207.                    public boolean accept(File dir, String name) {
208.                        if (name.endsWith(".txn") || name.endsWith(".idx") || name.endsWith(".log")) {
209.                            oldDataExists.set(true);
210.                        }
211.                        return true;
212.                    }
213.                });
214.            if (preV3versionFile.exists() || oldDataExists.get()) {
215.                String err = "Directory layout version is less than 3, upgrade needed";
216.                LOG.error(err);
217.                throw new IOException(err);
218.            }
219.            if (!dir.mkdirs()) {
220.                String err = "Unable to create directory " + dir;
221.                LOG.error(err);
222.                throw new IOException(err);
223.            }
224.        }
225.    }
```

Fig. 9

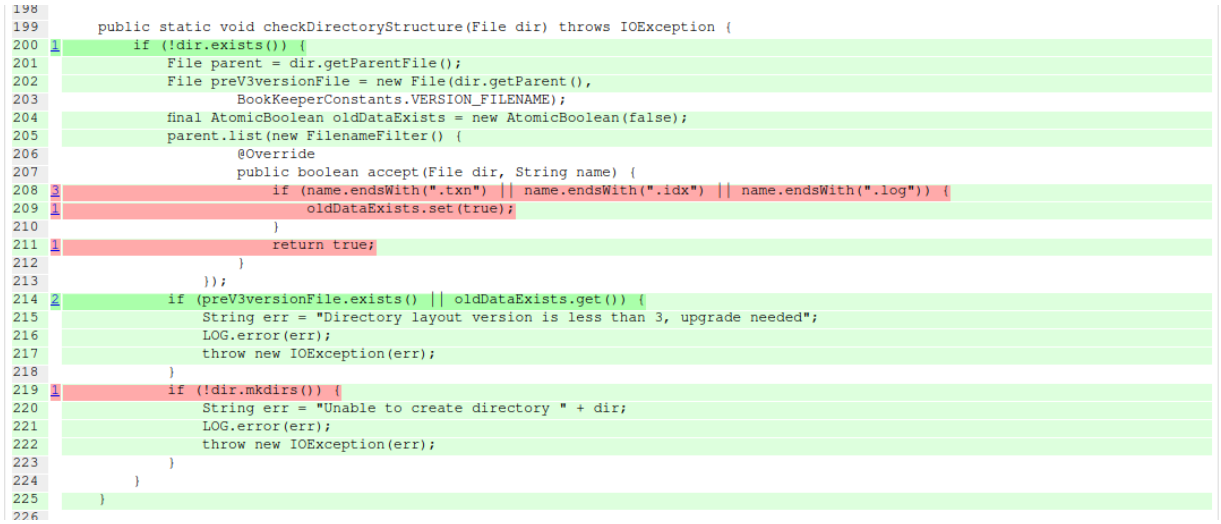


```
199.    public static void checkDirectoryStructure(File dir) throws IOException {
200.        if (!dir.exists()) {
201.            File parent = dir.getParentFile();
202.            File preV3versionFile = new File(dir.getParent(),
203.                    BookKeeperConstants.VERSION_FILENAME);
204.            final AtomicBoolean oldDataExists = new AtomicBoolean(false);
205.            parent.list(new FilenameFilter() {
206.                    @Override
207.                    public boolean accept(File dir, String name) {
208.                        if (name.endsWith(".txn") || name.endsWith(".idx") || name.endsWith(".log")) {
209.                            oldDataExists.set(true);
210.                        }
211.                        return true;
212.                    }
213.                });
214.            if (preV3versionFile.exists() || oldDataExists.get()) {
215.                String err = "Directory layout version is less than 3, upgrade needed";
216.                LOG.error(err);
217.                throw new IOException(err);
218.            }
219.            if (!dir.mkdirs()) {
220.                String err = "Unable to create directory " + dir;
221.                LOG.error(err);
222.                throw new IOException(err);
223.            }
224.        }
225.    }
```

Fig. 10

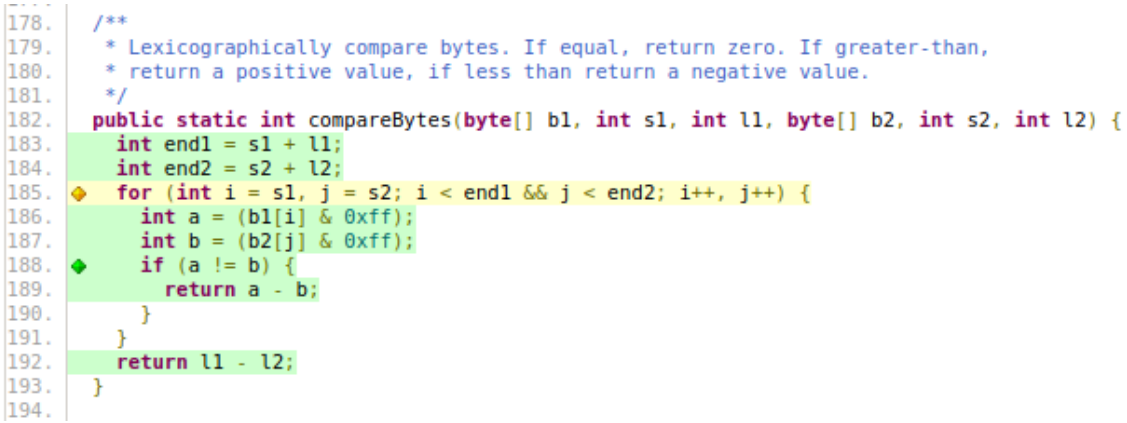
```
199     public static void checkDirectoryStructure(File dir) throws IOException {
200 1       if (!dir.exists()) {
201             File parent = dir.getParentFile();
202             File preV3versionFile = new File(dir.getParent(),
203                     BookKeeperConstants.VERSION_FILENAME);
204             final AtomicBoolean oldDataExists = new AtomicBoolean(false);
205             parent.list(new FilenameFilter() {
206                     @Override
207                     public boolean accept(File dir, String name) {
208 3                       if (name.endsWith(".txn") || name.endsWith(".idx") || name.endsWith(".log")) {
209 1                           oldDataExists.set(true);
210                         }
211 1                       return true;
212                     }
213                 });
214 2           if (preV3versionFile.exists() || oldDataExists.get()) {
215               String err = "Directory layout version is less than 3, upgrade needed";
216               LOG.error(err);
217               throw new IOException(err);
218           }
219 1           if (!dir.mkdirs()) {
220               String err = "Unable to create directory " + dir;
221               LOG.error(err);
222               throw new IOException(err);
223           }
224       }
225   }
226
```

Fig. 11



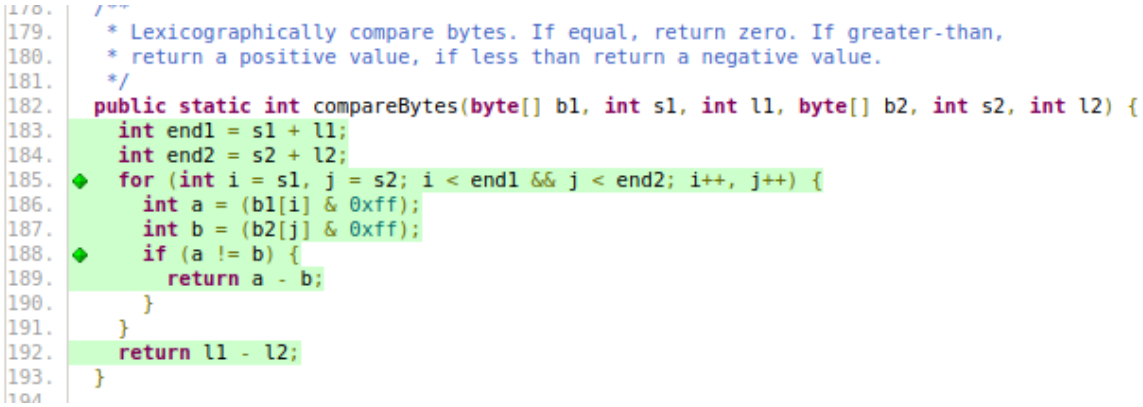
```
198
199     public static void checkDirectoryStructure(File dir) throws IOException {
200 1       if (!dir.exists()) {
201             File parent = dir.getParentFile();
202             File preV3versionFile = new File(dir.getParent(),
203                     BookKeeperConstants.VERSION_FILENAME);
204             final AtomicBoolean oldDataExists = new AtomicBoolean(false);
205             parent.list(new FilenameFilter() {
206                     @Override
207                     public boolean accept(File dir, String name) {
208 3                       if (name.endsWith(".txn") || name.endsWith(".idx") || name.endsWith(".log")) {
209 1                           oldDataExists.set(true);
210                         }
211 1                       return true;
212                     }
213                 });
214 2           if (preV3versionFile.exists() || oldDataExists.get()) {
215                 String err = "Directory layout version is less than 3, upgrade needed";
216                 LOG.error(err);
217                 throw new IOException(err);
218             }
219 1           if (!dir.mkdirs()) {
220                 String err = "Unable to create directory " + dir;
221                 LOG.error(err);
222                 throw new IOException(err);
223             }
224         }
225     }
226
```

Fig. 12



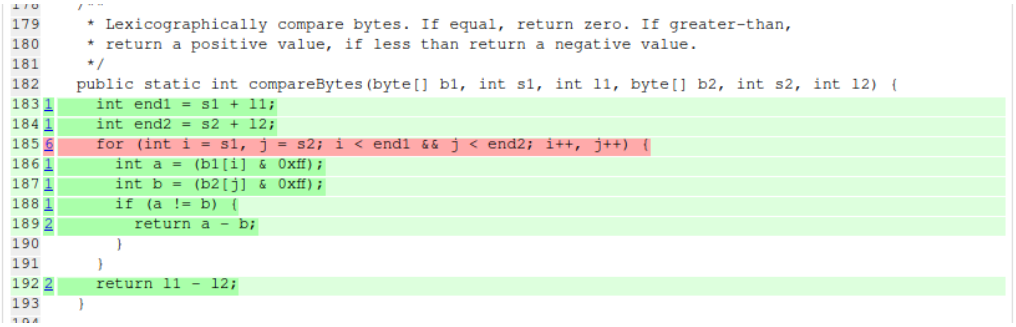
```
178.   /**
179.    * Lexicographically compare bytes. If equal, return zero. If greater-than,
180.    * return a positive value, if less than return a negative value.
181.    */
182.   public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
183.     int end1 = s1 + l1;
184.     int end2 = s2 + l2;
185.     for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
186.       int a = (b1[i] & 0xff);
187.       int b = (b2[j] & 0xff);
188.       if (a != b) {
189.         return a - b;
190.       }
191.     }
192.     return l1 - l2;
193.   }
194.
```

Fig. 13



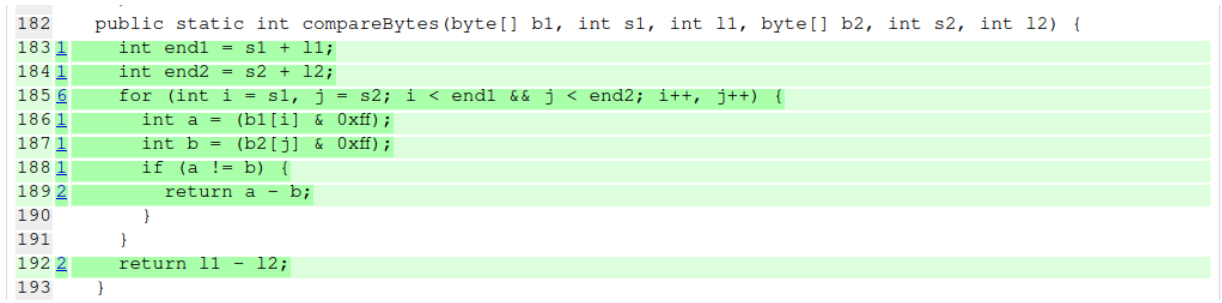
```
179.    * Lexicographically compare bytes. If equal, return zero. If greater-than,
180.    * return a positive value, if less than return a negative value.
181.    */
182.   public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
183.     int end1 = s1 + l1;
184.     int end2 = s2 + l2;
185.     for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
186.       int a = (b1[i] & 0xff);
187.       int b = (b2[j] & 0xff);
188.       if (a != b) {
189.         return a - b;
190.       }
191.     }
192.     return l1 - l2;
193.   }
```

Fig. 14



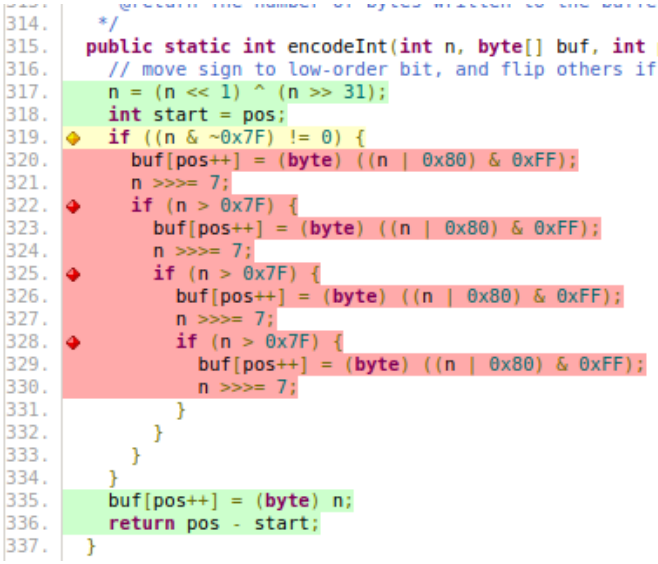
```
179    * Lexicographically compare bytes. If equal, return zero. If greater-than,
180    * return a positive value, if less than return a negative value.
181    */
182   public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
183 1   int end1 = s1 + l1;
184 1   int end2 = s2 + l2;
185 6   for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
186 1     int a = (b1[i] & 0xff);
187 1     int b = (b2[j] & 0xff);
188 1     if (a != b) {
189 2       return a - b;
190       }
191     }
192 2   return l1 - l2;
193   }
```

Fig. 15



```
182   public static int compareBytes(byte[] b1, int s1, int l1, byte[] b2, int s2, int l2) {
183 1   int end1 = s1 + l1;
184 1   int end2 = s2 + l2;
185 6   for (int i = s1, j = s2; i < end1 && j < end2; i++, j++) {
186 1     int a = (b1[i] & 0xff);
187 1     int b = (b2[j] & 0xff);
188 1     if (a != b) {
189 2       return a - b;
190       }
191     }
192 2   return l1 - l2;
193   }
```

Fig. 16



```
314.   */
315.  public static int encodeInt(int n, byte[] buf, int pos) {
316.    // move sign to low-order bit, and flip others if negative
317.    n = (n << 1) ^ (n >> 31);
318.    int start = pos;
319.    if ((n & ~0x7F) != 0) {
320.      buf[pos++] = (byte) ((n | 0x80) & 0xFF);
321.      n >>>= 7;
322.      if (n > 0x7F) {
323.        buf[pos++] = (byte) ((n | 0x80) & 0xFF);
324.        n >>>= 7;
325.        if (n > 0x7F) {
326.          buf[pos++] = (byte) ((n | 0x80) & 0xFF);
327.          n >>>= 7;
328.          if (n > 0x7F) {
329.            buf[pos++] = (byte) ((n | 0x80) & 0xFF);
330.            n >>>= 7;
331.          }
332.        }
333.      }
334.    }
335.    buf[pos++] = (byte) n;
336.    return pos - start;
337.  }
```

Fig. 17

```
315.  public static int encodeInt(int n, byte[] buf, int pos) {
316.    // move sign to low-order bit, and flip others if negative
317.    n = (n << 1) ^ (n >> 31);
318.    int start = pos;
319.    if ((n & ~0x7F) != 0) {
320.      buf[pos++] = (byte) ((n | 0x80) & 0xFF);
321.      n >>>= 7;
322.      if (n > 0x7F) {
323.        buf[pos++] = (byte) ((n | 0x80) & 0xFF);
324.        n >>>= 7;
325.        if (n > 0x7F) {
326.          buf[pos++] = (byte) ((n | 0x80) & 0xFF);
327.          n >>>= 7;
328.          if (n > 0x7F) {
329.            buf[pos++] = (byte) ((n | 0x80) & 0xFF);
330.            n >>>= 7;
331.          }
332.        }
333.      }
334.    }
335.    buf[pos++] = (byte) n;
336.    return pos - start;
337.  }
338.
```

Fig. 18

```
315    public static int encodeInt(int n, byte[] buf, int pos) {
316      // move sign to low-order bit, and flip others if negative
317 3    n = (n << 1) ^ (n >> 31);
318      int start = pos;
319 2    if ((n & ~0x7F) != 0) {
320 3      buf[pos++] = (byte) ((n | 0x80) & 0xFF);
321 1      n >>>= 7;
322 2      if (n > 0x7F) {
323 3        buf[pos++] = (byte) ((n | 0x80) & 0xFF);
324 1        n >>>= 7;
325 2        if (n > 0x7F) {
326 3          buf[pos++] = (byte) ((n | 0x80) & 0xFF);
327 1          n >>>= 7;
328 2          if (n > 0x7F) {
329 3            buf[pos++] = (byte) ((n | 0x80) & 0xFF);
330 1            n >>>= 7;
331            }
332          }
333        }
334      }
335 1    buf[pos++] = (byte) n;
336 2    return pos - start;
337    }
```

Fig. 19

```
314     */
315    public static int encodeInt(int n, byte[] buf, int pos) {
316      // move sign to low-order bit, and flip others if negative
317 3    n = (n << 1) ^ (n >> 31);
318      int start = pos;
319 2    if ((n & ~0x7F) != 0) {
320 3      buf[pos++] = (byte) ((n | 0x80) & 0xFF);
321 1      n >>>= 7;
322 2      if (n > 0x7F) {
323 3        buf[pos++] = (byte) ((n | 0x80) & 0xFF);
324 1        n >>>= 7;
325 2        if (n > 0x7F) {
326 3          buf[pos++] = (byte) ((n | 0x80) & 0xFF);
327 1          n >>>= 7;
328 2          if (n > 0x7F) {
329 3            buf[pos++] = (byte) ((n | 0x80) & 0xFF);
330 1            n >>>= 7;
331            }
332          }
333        }
334      }
335 1    buf[pos++] = (byte) n;
336 2    return pos - start;
337    }
```

Fig. 20

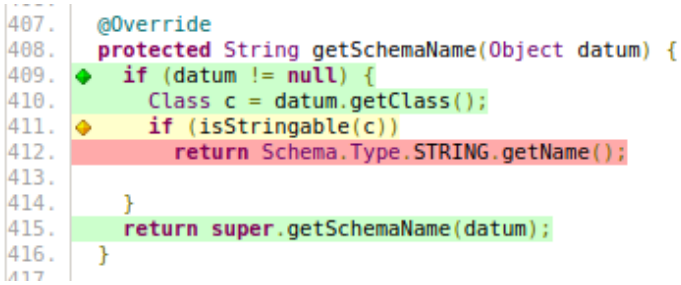


Fig. 21

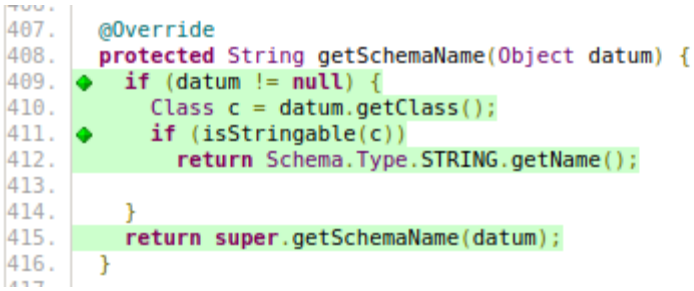


Fig. 22

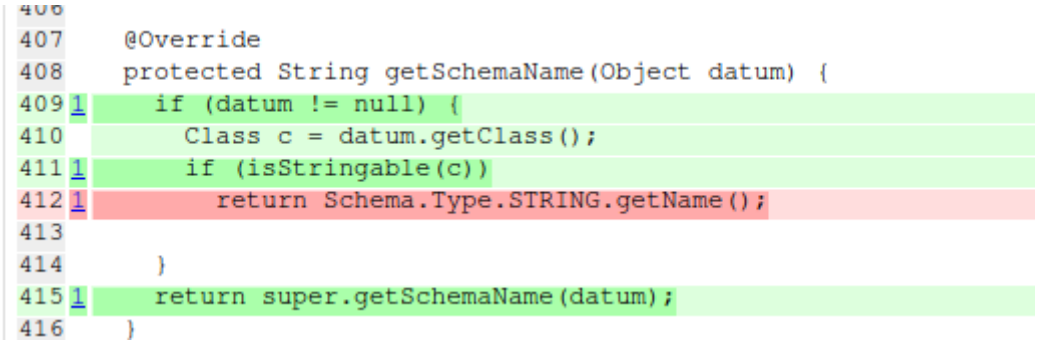


Fig. 23

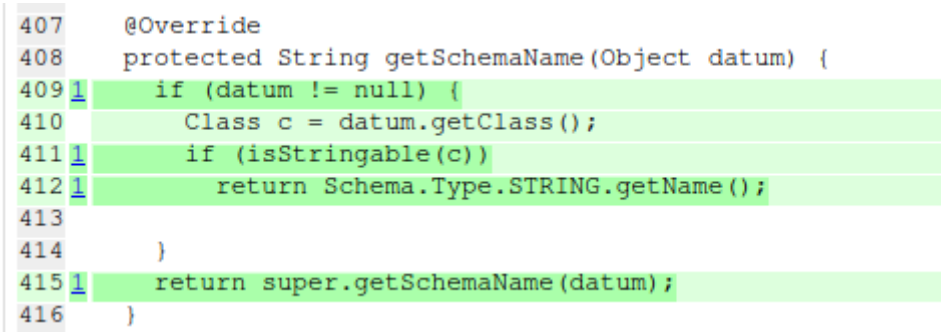


Fig. 24

```
101.   */
102.  protected static boolean isValidValue(Field f, Object value) {
103.    if (value != null) {
104.      return true;
105.    }
106.
107.    Schema schema = f.schema();
108.    Type type = schema.getType();
109.
110.    // If the type is null, any value is valid
111.    if (type == Type.NULL) {
112.      return true;
113.    }
114.
115.    // If the type is a union that allows nulls, any value is valid
116.    if (type == Type.UNION) {
117.      for (Schema s : schema.getTypes()) {
118.        if (s.getType() == Type.NULL) {
119.          return true;
120.        }
121.      }
122.    }
123.
124.    // The value is null but the type does not allow nulls
125.    return false;
126.  }
```

Fig. 25

```
102    protected static boolean isValidValue(Field f, Object value) {
103 1    if (value != null) {
104 1      return true;
105      }
106
107      Schema schema = f.schema();
108      Type type = schema.getType();
109
110      // If the type is null, any value is valid
111 1    if (type == Type.NULL) {
112 1      return true;
113      }
114
115      // If the type is a union that allows nulls, any value is valid
116 1    if (type == Type.UNION) {
117        for (Schema s : schema.getTypes()) {
118 1        if (s.getType() == Type.NULL) {
119 1          return true;
120          }
121        }
122      }
123
124      // The value is null but the type does not allow nulls
125 1    return false;
126    }
```

Fig. 26