

Obiettivi

L'obiettivo di questo progetto consiste nello svolgimento e nella successiva presentazione delle attività di software testing sulle classi java di due progetti OpenSource : Bookkeeper ed OpenJPA. Per entrambi i progetti, nella repository di GitHub, sono stati inseriti file .yml che permettono di effettuare il build e la scansione del progetto su SonarCloud automaticamente al verificarsi di ogni commit nel repository. Successivamente, si ottengono i report (salvati come artifact nella sezione Actions di GitHub) di code coverage con Jacoco, mutation coverage con PIT e data-flow-coverage con Ba-dua (Utilizzando i seguenti profili per entrambi i progetti -Pjacoco, -Ppit-test, -Pba-dua-coverage).

Bookkeeper

Bookkeeper è un progetto open-source della Apache Software Foundation. Si tratta di un sistema di storage distribuito che consente alle applicazioni di scrivere e leggere i dati in modo affidabile in un ambiente distribuito. Bookkeeper è progettato per essere altamente disponibile, scalabile e resistente ai guasti, offrendo quindi garanzie di consistenza dei dati. Si basa su un insieme di nodi Bookie che collaborano per memorizzare e replicare i dati in un nodo distribuito. Ogni nodo Bookie è responsabile della memorizzazione dei dati in un segmento di registro, e più nodi Bookie collaborano per memorizzare e replicare i dati attraverso diverse regioni e zone di disponibilità. Le applicazioni possono utilizzare il client Bookkeeper per scrivere e leggere dati dal sistema, garantendo che i dati siano scritti e letti in modo coerente e affidabile.

Per la campagna di testing, per poter sfruttare le conoscenze apprese con il progetto del Professor Fallessi è stata utilizzata la release 4.6.0. Tuttavia è stato verificato il corretto funzionamento dei framework utilizzati (Jacoco, PIT e ba-dua), definendo un ciclo di CI con le GitHub Actions, anche nella release 4.16.1 (ultima rilasciata) inserendo i test riferiti a classi che non hanno subito modifiche dalla release 4.6.0 alla release 4.16.1 e aggiungendo un semplice test per la classe WriteCache non presente nella release utilizzata. Il lavoro è qui presente: <https://github.com/giuliameni/bookkeeper> ¹

Classi Testate

Sono state testate diverse classi all'interno del progetto Bookkeeper. Le classi testate includono:

0.1 BufferedChannel

Questa classe fornisce uno strato di buffering per un FileChannel, consentendo di ridurre il numero di accessi al file system, in quanto i dati scritti/letti vengono prima memorizzati in un buffer, e solo successivamente scritti/letti dal file system. In questo modo si riduce l'overhead dei singoli accessi al file system, molto costoso rispetto alle operazioni di memoria. In particolare, BufferedChannel estende BufferedReadChannel e aggiunge la capacità di scrivere dati sul canale di output. Il buffering avviene utilizzando un buffer di scrittura e un buffer di lettura. Il buffer di scrittura viene utilizzato per memorizzare i dati da scrivere sul file system, mentre il buffer di lettura viene utilizzato per leggere i dati dal file system.

¹NB: il link è riferito ad un account differente da quello utilizzato per la consegna del progetto, non presenta quindi al suo interno il testing relativo alla release 4.6.0 qui presente <https://github.com/giulia97-w/bookkeeper-1>

0.2 EntryKey

La classe **EntryKey** rappresenta una chiave per identificare un'entry in un ledger. Una chiave è costituita da due campi, **ledgerId** e **entryId**, che identificano rispettivamente l'id del ledger e l'id dell'entry all'interno del ledger. La classe viene utilizzata principalmente all'interno della classe **SortedLedgerStorage** per la gestione e l'ordinamento delle entry nella memtable. La classe fornisce anche un comparatore, **KeyComparator**, che consente di confrontare due oggetti **EntryKey** in base all'ordine dei loro **ledgerID** e **entryID**. Questo comparatore è utilizzato per scopi di ordinamento e confronto all'interno del sistema.

0.3 LedgerEntriesImpl

Tale classe implementa l'interfaccia **LedgerEntries** ed è utilizzata per rappresentare un elenco di voci in un registro. Incapsula una lista di **LedgerEntry** e fornisce metodi per accedere alle singole voci (con l'ID della specifica voce) e per iterare sull'intera lista. La classe utilizza anche il meccanismo di riciclo fornito alla libreria Netty('Recycler') per migliorare le prestazioni e gestione della memoria. Il metodo 'recycle()' viene chiamato per ripristinare l'oggetto 'LedgerEntriesImpl' nel pool di oggetti riutilizzabili dopo averlo utilizzato.

Test Implementati

Sono stati implementati diversi test per le classi sopra descritte. Questi includono:

0.4 BufferedChannelReadTest

Il primo metodo testato è il metodo **read()**. Tale metodo legge i dati dal **FileChannel** sottostante e li restituisce in un **ByteBuffer**. Effettua la lettura dalla posizione specificata all'interno del file. La lettura può essere eseguita in diverse fasi:

- Prima verifica se i dati si trovano nel buffer di scrittura **writeBuffer** e, se presenti, li copia nel **ByteBuffer** di destinazione.
- Successivamente, controlla se i dati sono nel buffer di lettura **readBuffer** e, se presenti, li copia nel **ByteBuffer** di destinazione.
- Se i dati non si trovano in alcuno dei buffer, legge direttamente dal **FileChannel** nel buffer di lettura e quindi copia i dati nel **ByteBuffer** di destinazione. Il metodo restituisce il numero di byte letti durante l'operazione di lettura. E' stata quindi creata la seguente classe di test:

```
public BufferedChannelReadTest(int pos, int destLen, byte[] expected)
```

@Parameterize.Parameters

```
public static Collection<Object> data() {
    return Arrays.asList(new Object[][] {
        { 0, 100, new byte[100] }, // read the first 100 bytes
        { 1024, 0, new byte[0] }, // read after the end of file, expect empty buffer
        { -1, 1, new byte[1] }
        { -1, -1, new byte[0] }
    });
}
```

```
});  
}  
}
```

Il test legge i dati dal fileChannel e confronta i dati attesi. In particolare, il test crea un oggetto ByteBuffer di destinazione con una lunghezza prefissata **destLen** e lo inizializza con una sequenza di byte. Quindi, crea un oggetto BufferedChannel che incapsula un fileChannel e legge i dati da esso nella posizione prefissata **pos** usando il metodo read(dest, pos). Infine i dati letti dal fileChannel sono estratti dall'oggetto ByteBuffer di destinazione e confrontati con i dati attesi memorizzati nell'array di byte **expected**. Per valori di dest e pos negativi si verifica che venga sollevata una eccezione.

0.5 BufferedChannelTest

Per poter aumentare la coverage sono stati aggiunti test ulteriori in questa classe. In particolare i due metodi testati all'interno di questa classe sono il metodo **write()** e il metodo **read()**. Il metodo write() scrive i dati contenuti nel buffer di input **src** sul FileChannel. La scrittura può essere bufferizzata o rordinata a seconda dell'implementazione. Queste scritture verranno effettivamente scritte sul disco solo quando viene invocato il metodo flush(). Il metodo legge i dati dal buffer di input e li scrive nel buffer di scrittura **writeBuffer**. Se lo spazio nel buffer di scrittura è esaurito, viene effettuata una scrittura sul file tramite il FileChannel. La variabile **position** tiene traccia della posizione corrente per la scrittura. Dopo la scrittura, la variabile **position** viene aggiornata in base al numero di byte scritti. Viene scritto quindi un test parametrico con i seguenti parametri che modellano la size del buffer utilizzato:

```
@Parameterized.Parameters  
public void Integer[] data(){  
    return new Integer[] { -1, 0, 1, 100, 1024, 4096 * 100 }  
}
```

Nella classe con annotazione @Before viene definito il metodo **setUp()**, eseguito prima di ogni test, crea un file temporaneo, ne apre un FileChannel ed istanzia un oggetto BufferedChannel. Con annotazione @After viene definito il metodo **tearDown()**, eseguito dopo ogni test, chiude il FileChannel e cancella il file temporaneo.

Con i parametri sopra definiti che identificano la dimensione del buffer si testa il comportamento del metodo **read** quando si tenta la lettura dalla posizione dopo la fine del file. Viene controllato quindi che venga sollevata l'eccezione "ReadPastEOF". Per bufferSize = -1 si verifica che, come atteso, l'istanziazione di un buffer con dimensioni negative sia impossibile. Viene infatti lanciata l'eccezione "IllegalArgumentException" e il test non fallisce. Viene verificato inoltre che i dati scritti su un BufferedChannel possano essere letti correttamente, viene quindi testata la scrittura e la lettura dalla posizione corretta e confrontati i dati letti con quelli scritti. Viene inoltre verificato il comportamento dei metodi **read()** e **write()** quando viene effettuata la chiamata al metodo **flush()**.

Il fallimento dei test è causato dal valore di bufferSize = 0. Nel codice sorgente non è presente alcun controllo su questo caso, ci si aspetta infatti che venga resa impossibile l'istanziazione di buffer di dimensioni nulle vista in tal caso l'inutilità di utilizzare tale classe poiché la scrittura avverrebbe direttamente senza l'ausilio del buffer con tutto ciò che questo comporta.

Coverage e Mutation Coverage

I test sopra descritti hanno garantito il raggiungimento di una coverage pari al 100% per le Missed Instruction, 100% per Missed Branches, Data Flow Coverage pari al 100% per il metodo **write()** e Coverage pari al 90%, 71% e 67%(Data Flow Coverage) per il metodo **read()**. La classe **BufferedChannel** nel totale raggiunge mutation coverage generale pari al 68%.

0.6 EntryKeyEqualNewTest

La seconda classe testata è EntryKey.java in particolare sono stati scritti test parametrici. Il primo test riguarda il metodo Equal(). Tale metodo confronta se l'oggetto passato come parametro è un'istanza di **EntryKey** e, in caso affermativo, confronta i valori degli attributi **ledgerId** e **entryId** dell'oggetto corrente con quelli dell'oggetto passato come parametro. Se i valori degli attributi **ledgerId** ed **entryId** sono uguali, i due oggetti sono considerati uguali e restituisce **true**, altrimenti restituisce **false**. Sono stati istanziati i seguenti parametri per entryKey ed object.

```
public EntryKeyEqualNewTest(EntryKey entryKey, Object other, boolean expectedResult)
```

```
@Parameterize.Parameters
public static <Object[][]> parameters() {
    return new Object[][] {
        {new EntryKey(0,0), new EntryKey(0,0), true},
        {new EntryKey(0,0), new EntryKey(0,1), false},
        {new EntryKey(0,0), null, false},
        {new EntryKey(0,0), -1, false},
        {new EntryKey(0,0), 0, false},
        {new EntryKey(0,0), Long.MAX_VALUE, false},
    });
}
```

Il test verifica il funzionamento atteso precedentemente descritto. Dei sei casi di test scritti nessuno ha causato il fallimento.

0.7 EntryKeyHashTest

In questa classe si procede con il testing del metodo hashCode() della classe EntryKey.java. Tale metodo restituisce un intero che rappresenta il valore di hash dell'istanza **EntryKey**. Il valore di hash è calcolato utilizzando i valori di **ledgerID** e **entryId** dell'istanza moltiplicati per costanti diverse (13 e 17 rispettivamente). Questo metodo è implementato per essere coerente con il metodo 'equals()', in modo da garantire che due istanze di **EntryKey** uguali restituiscano lo stesso valore di hash.

I parametri definiti sono i seguenti:

```
public EntryKeyHashTest(long ledgerId, long entryId,
```

```
@Parameterize.Parameters
public static <Object[][]> data() {
    return new Object[][] {
        {-1, -1},
        {0L, 0L},
        {1L, 1L},
        {long.MAX_VALUE, long.MAX_VALUE},
    }
}
```

Il test verifica il corretto funzionamento per diversi valori di **ledgerId** ed **entryId**. In particolare viene testato che il valore restituito dal metodo 'hashCode()' per un oggetto 'entryKey' creato con i valori specificati corrisponda al valore atteso. Inoltre, viene verificato che i metodi 'getEntryId' e 'getLedgerId' restituis-

cano rispettivamente **entryId** e **ledgerId** passati al costruttore.

Coverage e Mutation Coverage

Con i test sopra descritti si raggiunge la totale copertura della classe sia per la coverage ottenuta con Jacoco, sia per la mutation coverage ottenuta con PIT con 0 mutanti sopravvissuti sugli 11 istanziati.

0.8 LedgerEntriesImplCreateTest

Il metodo `create()` della classe 'LedgerEntriesImpl' viene utilizzato per creare un'istanza della classe a partire da una lista di 'LedgerEntry'. Viene controllato che la lista non sia vuota, se la lista è vuota viene lanciata una eccezione. Successivamente, viene ottenuto un oggetto 'LedgerEntriesImpl' dal pool di oggetti 'Recycler' utilizzando il metodo 'RECYCLER.get()'. Il pool di oggetti viene utilizzato per riciclare oggetti già presenti invece di crearne sempre nuovi, al fine di ridurre l'impatto sulla memoria.

```
public class LedgerEntriesImplCreateTest { public static Collection<Object> data() {
    return Arrays.asList(new Object[][] {
        {null,true}, //true = expected exception
        {Collections.emptyList(), true}, //non valido
        {Arrays.asList(mock(LedgerEntry.class)), false }, //valido
    })
}
```

Se viene passato un parametro 'null' il test verifica che venga sollevata una eccezione. Se viene passato parametro non 'null' viene creata una istanza di 'LedgerEntriesImpl' con la lista di 'LedgerEntry' passata come parametro e verificato che l'iteratore restituisca esattamente tutti gli elementi della lista passata come parametro e che non ci siano eccezioni sollevate. Se ci si aspetta che venga sollevata un'eccezione (expectedException = true) viene verificato che essa sia effettivamente sollevata. Per i 3 casi di test selezionati non si registrano fallimenti nel test appena descritto.

Coverage e Mutation Coverage

Con i test sopra descritti si raggiunge la totale copertura della classe sia per la coverage ottenuta con Jacoco, sia per la mutation coverage ottenuta con PIT con 0 mutanti sopravvissuti e Data Flow Coverage pari al 75%.

0.9 LedgerEntriesImplGetEntryTest

Il metodo `getEntry()` viene utilizzato per ottenere un singolo 'LedgerEntry' dalla lista di entries in base all'ID specificato. Viene quindi ottenuto l'ID del primo e dell'ultimo 'LedgerEntry' nella lista, se l'ID specificato è inferiore all'ID del primo elemento o superiore all'ID dell'ultimo elemento, viene lanciata una eccezione indicando che l'indice richiesto è fuori dai limiti della lista. Infine viene restituito l'elemento 'LedgerEntry' corrispondente all'ID specificato.

```
public class LedgerEntriesImplGetEntryTest { public static Collection<Object> data() {
```

```
return Arrays.asList(new Object[][] {
    {2L,false}, //validEntry
    {1L,false}, //validEntry
    {3L,false}, //validEntry
    {0L,true}, //invalidEntry
    {4L,true}, //invalidEntry
    {-1L,true}, //invalidEntry
})
```

Se ci si aspetta l'eccezione (`expectException = true`) il test verifica che questa venga lanciata e che sia di tipo 'IndexOutOfBoundsException' quando si cerca di accedere a un'entry con un ID non valido. Viceversa il test verifica se il metodo `getEntry()` restituisce correttamente l'entry corrispondente all'ID richiesto. Nessun caso causa il fallimento del test appena descritto.

Coverage e Mutation Coverage

Fino a questo punto si riesce a raggiungere una Coverage del 100% per le Missed Instruction e Missed Branches, Data Flow Coverage = 100% ma una mutation coverage dell'88% per l'intera classe. I metodi coinvolti nei test raggiungono mutation coverage massima ma questo non garantisce il raggiungimento della copertura massima per l'intera classe. Sono stati quindi inseriti altri due test dopo aver analizzato il report di Pit. In particolare Il primo test "testRecyclerHandleOnClose" verifica se il metodo `close()` richiama il metodo `recycle()` sull'oggetto `mockHandle`, mentre il secondo test verifica se il metodo `clear()` viene chiamato quando il metodo `close()` è invocato.

Con l'aggiunta di questi test si riesce a raggiungere Coverage e Mutation Coverage massime.

1 OpenJPA

Il progetto OpenJPA è un'implementazione di Java Persistence API (JPA), che è una interfaccia di programmazione Java standard per l'accesso ai dati mediante ORM(object-Relational Mapping). In sostanza, OpenJPA consente di mappare oggetti Java su tabelle di database relazionali in modo da semplificare la gestione dei dati tra l'applicazione e il database. L'obiettivo principale di OpenJPA è quello di fornire un'implementazione di JPA che sia facile da usare, scalabile e performante. A tale scopo, fornisce un'ampia gamma di funzionalità, tra cui la gestione della cache dei risultati delle query, la gestione delle transazioni distribuite e la gestione degli eventi del ciclo di vita degli oggetti. Supporta inoltre molte delle funzionalità standard di JPA, come l'annotazione di classi Java per mappare tabelle di database a la definizione di relazioni tra le tabelle. Inoltre fornisce molte estensioni per migliorare le prestazioni e la sclibilità delle applicazioni. Ad esempio, la funzionalità di caching dei risultati delle query consente di ridurre il tempo di esecuzione delle query ripetute, mentre la gestione delle transazioni distribuite consente di gestire le transazioni che coinvolgono più database e applicazioni.

2 Classi Testate

2.1 QualifiedDBIdentifier

La classe QualifiedDBIdentifier estende DBIdentifier e fornisce supporto per la gestione di identificatori qualificati² utilizzati nel contesto dei database. Gli identificatori qualificati possono includere sia un nome di schema che un nome di tabella, consentendo una maggiore precisione nella specifica degli oggetti nel database.

2.2 ClassUtil

La classe 'ClassUtil' fornisce vari metodi di utilità per manipolare classi in Java, viene utilizzata per semplificare le operazioni comuni legate alle classi in Java, come ottenere il nome della classe o del package, convertire una stringa in una classe e gestire correttamente i tipi primitivi e gli array.

Test implementati

2.3 QualifiedIdentifierSplitPathTest

Questa classe modella un test per il metodo splitPath() della classe QualifiedDBIdentifier. Tale metodo accetta un oggetto 'DBIdentifier' come parametro e lo suddivide in un array di 'DBIdentifier', rappresentando il nome dell'oggetto come una serie di identificatori. Se l'oggetto passato come parametro è un'istanza di 'QualifiedDBIdentifier' e il tipo di identificatore non è "SCHEMA", il metodo esegue il parsing del nome dell'oggetto in base al formato '<schema>.<table>.<column>' e restituisce un array di 'DBIdentifier', dove ogni elemento rappresenta uno dei tre elementi del nome dell'oggetto. Se l'oggetto passato come parametro è un'istanza di 'DBIdentifier', il metodo restituisce un array contenente l'oggetto stesso. Se l'oggetto passato come parametro non è un'istanza di 'DBIdentifier' né di 'QualifiedDBIdentifier', il metodo restituisce un array vuoto. Per il test vengono istanziati i seguenti parametri:

```
public QualifiedIdentifierSplitPathTest(boolean output , DBIdentifier identifier )  
  
@Parameterize.Parameters  
public static Collection<Object> data() {  
    return Arrays.asList(new Object[][] {  
        { true , DBIdentifier.newTable("Schema.Table")},  
        { false , DBIdentifier.newColumn( null ) }  
    }  
}
```

Il test verifica il corretto funzionamento del metodo sopra descritto. Viene chiamato splitPath() sul parametro di input e verificato se l'output è coerente con il valore atteso. In particolare il test verifica se il metodo 'splitPath()' restituisce un array di lunghezza 2 per il primo caso di test e un array vuoto altrimenti.

²Un identificatore qualificato è un nome di oggetto che include informazioni aggiuntive per specificare in modo univoco l'oggetto all'interno del database. Solitamente un identificatore qualificato è composto da due o più componenti che vengono separati utilizzando un delimitatore, ad esempio un punto. I componenti di un identificatore possono includere: Nome dello schema a cui l'oggetto appartiene, Nome della tabella a cui l'oggetto appartiene e nome dell'oggetto specifico, ad esempio una colonna.

2.4 Coverage e Mutation Coverage

Il test garantisce, per il metodo esaminato, una Coverage pari al 100% per Missed Instructions, 83% per le Missed Branches, Data Flow Coverage pari all'86.95% e una Mutation Coverage massima con 0 mutanti sopravvisuti.

2.5 QualifiedIdentifierPathEqualTest

Questa classe implementa un test per il metodo PathEqual(). Tale metodo confronta i path di due oggetti 'QualifiedDBIdentifier' per determinare se rappresentano lo stesso identificatore. Restituisce 'true' se i path dei due oggetti sono uguali ovvero se hanno lo stesso schema, gli stessi nomi di table e identifier. Restituisce 'false' se gli oggetti rappresentano identificatori diversi.

```
public QualifiedIdentifierPathEqualTest(QualifieDBIdentifier id1 , QualifieDBIdentifier id2, boolean expectedEquality )
```

@Parameters

```
public static <Object[]> data() {
    return new Object[][] {
        { p1, p2 , true}, // p1 = <schema>.<tabel>.<column> e p1 = p2
        { p1, p3 , false}, // p3 = <schema>.<tabel>.<otherColumn>
        { p1, p5 , false}, // p5 = <otherSchema>.<tabel>.<column>
        { p6, p1 , false}, // p6 = null
        { p1, p6 , false},
        { p7, p1 , false}, // p7 = <schema>
        { p6, p8 , true}, // p8 = null
    }
}
```

Il test verifica quindi il corretto funzionamento del metodo pathEqual confrontando diverse istanze di QualifiedDBIdentifier costituite con percorsi di identificatori di database diversi. Verifica quindi, invocando pathEqual sulle variabili, che il risultato dell'invocazione corrisponda al valore atteso (true o false).

2.6 Coverage e Mutation Coverage

Il test sopra descritto garantisce una Coverage pari al 100% per le Missed Instructions e Missed Branches, Data Flow Coverage pari al 96.42% e una Mutation Coverage pari al 100% con 0 mutanti sopravvisuti.

2.7 QualifiedIdentifierEqualsTest

Questa classe implementa un test per il metodo equals(). Tale metodo controlla se l'oggetto attuale è uguale all'oggetto passato come parametro e in tal caso restituisce true altrimenti false. Il metodo controlla prima se l'oggetto passato non è nullo e quindi controlla se è una istanza della classe QualifieDBIdentifier, DBIdentifier o String. Se l'oggetto passato è istanza di QualifieDBIdentifier, il metodo controlla se gli attributi del path dell'oggetto corrente sono uguali a quello dell'oggetto passato. Se l'oggetto passato è una String, il metodo controlla se la stringa passata come parametro è uguale al

nome dell'oggetto corrente. Infine se l'oggetto non è una istanza delle classi sopra descritte, viene lanciata una eccezione di tipo `IllegalArgumentException`. Per il test sono stati definiti i seguenti parametri:

```
public QualifiedIdentifierEqualsITest(QualifiedDBIdentifier id1 , QualifiedDBIdentifier id2, boolean expectedEquality )
```

@Parameters

```
public static Collection<Object[]> data() {  
    return Array.asList(new Object[][] {  
        { path3, s3 , false},  
        { path3, pathString , true},  
        { path1, path1 , true},  
        { path1, path2 , true},  
        { path1, path3 , true},  
        { path2, path3 , true},  
        { path1, path4 , true},  
        { path1, null , false},  
        { path1, s1 , false},  
    })
```

Il test verifica se il valore di ritorno dell'invocazione del metodo 'equals()' tra due identificatori corrisponde al valore atteso. In particolare, il test verifica se 'QualifiedDBIdentifier' è in grado di effettuare il confronto con una 'String' rappresentante il path. Viene inoltre verificato che l'invocazione del metodo 'equals' su un oggetto 'QualifiedDBIdentifier' passando come parametro un oggetto 'Object' di classe non compatibile (diversa da 'QualifiedDNIdentifier', 'DBIdentifier' o 'String') sollevi correttamente un'eccezione di tipo 'IllegalArgumentException'.

Per maggior chiarezza viene presentato lo snippet dei parametri utilizzati nel capitolo Immagini.

2.8 Coverage e Mutation Coverage

Il test sopra descritto garantisce per il metodo `equals()` una Coverage per le Missed Instructions pari al 100% e Missed Branches pari all'85%, Data Flow Coverage pari all'86.61% e una Mutation coverage massima con 0 mutanti sopravvissuti. L'intera classe dopo il testing effettuato sui 3 metodi descritti ha una Coverage del 75% per le Missed Instructions e 61% per Missed Branches ed una Mutation Coverage pari al 63%.

3 ClassUtilNamePackageTest

In questa classe viene testato il metodo `getPackageName()` della classe `ClassUtil`. Tale metodo prende in input una stringa **fullName** che rappresenta il nome completo di una classe in Java (comprensivo del nome del package e il nome della classe separati da un punto). Il metodo restituisce il nome del package associato a questa classe. In particolare, il metodo controlla se il valore passato come parametro '`fullName`' è nullo o vuoto, e in tal caso restituisce il valore stesso. In caso contrario, il metodo determina il numero di dimensioni di un array contenuto nel nome completo della classe, se presente, utilizzando il metodo '`getArrayDimensions`'. Se il valore delle dimensioni è maggiore di 0, la funzione verifica se '`fullName`' rappresenta un tipo primitivo con il numero di dimensioni specificato. In tal caso restituisce una stringa vuota poiché non ha senso parlare di un package associato a un tipo primitivo. In caso contrario, la funzione rimuove la parte dell'array dal nome completo e cerca l'ultima occorrenza del carattere '.' per

trovare il nome del package. Se trova un punto, restituisce la sottostringa di fullName che va dall'inizio fino all'ultimo punto. In caso contrario restituisce una stringa vuota perché non è stato trovato alcun nome del package.

Per l'esecuzione del test sono stati definiti i seguenti parametri:

```
public ClassUtilNamePackageTest(String input, String expectedOutput)
```

```
@Parameters public static Collection<Object> testData() {
    return Arrays.asList(new Object[][] {
        {"", ""},
        {"int", ""},
        {"[I", ""},
        {"MyClass", ""},
        {"com.example.MyClass", "com.example"},
        {"com.example.MyClass$InnerClass", "com.example"},
        {null, null},
        {"[Lcom.example.MyClass;", "com.example"}
    });
}
```

Ogni riga dell'array bidimensionale rappresenta un caso di test specifico che ha due elementi: il primo è una stringa che rappresenta il nome della classe di input, il secondo è una stringa che rappresenta il nome del package atteso per la classe di input.

Il metodo `testGetPackageName()` chiama il metodo `'ClassUtil.getPackageName(input)'` per ogni caso di test specifico e verifica che l'output sia uguale all'output atteso utilizzando il metodo `'assertEquals()'`. In questo modo si verifica che il metodo `getPackageName(String)` restituisca correttamente il nome del package di una classe data in input per diversi casi di test.

3.1 Coverage e mutation coverage

La copertura che si ottiene con questo test per il metodo `getPackageName` è pari al 100% per Missed Instructions, Missed branches e Data Flow Coverage mentre la mutation coverage scende al 93% con 1 mutante sopravvissuto sui 13 istanziati.

4 ClassUtilGetClassNameTest

In questa classe viene testato il metodo `getClassName(Class)` della classe `ClassUtil` il quale restituisce il nome della classe senza il nome del package che la precede accettando in questo caso un oggetto `Class`. Per questo test sono stati istanziati i seguenti parametri:

```
public ClassUtilGetClassNameTest(Class<?> inputClass, String expectedOutput)
```

```
@Parameters public static Collection<Object> testData() {
    return Arrays.asList(new Object[][] {
        {null, null},
        {String[].class, "String[][]"},
        {List.class, "List"},
        {Object.class, "Object"},
```

```
    {Map.Entry.class, "Map$Entry"}  
});
```

In questo test ogni riga dell'array bidimensionale rappresenta un caso di test specifico che ha due elementi: il primo è un oggetto di classe 'Class<? >' che rappresenta la classe di input, il secondo è una stringa che rappresenta il nome della classe atteso senza il nome del package.

Il costruttore della classe 'ClassUtilGetClassNameTest' viene utilizzato per inizializzare le variabili inputClass e expectedOutput per ogni caso di test specifico.

Il metodo di test 'testGetClassNameFromClass()' chiama il metodo 'ClassUtil.getClassName(inputClass)' per ogni caso di test specifico e verifica che l'output sia uguale all'output atteso utilizzando il metodo 'assertEquals()'. In questo modo il test verifica che il metodo ClassUtil.getClassName() restituisca correttamente il nome della classe senza il nome del package a partire dalla classe di input per diversi casi di test.

4.1 Coverage e Mutation Coverage

Con questo test si riesce a raggiungere il 100% di coverage per Missed Instructions, Missed branches, Data Flow Coverage e mutation coverage con 0 mutanti sopravvissuti.

5 ClassUtilGetClassNameStringTest

In questa classe viene testato il metodo getClassName(String). Tale metodo accetta una stringa che rappresenta il nome completo di una classe in Java (inclusi eventuali array), e restituisce il nome semplice della classe (senza il nome del package) in formato stringa. In particolare controlla se la stringa è vuota o nulla, poi analizza se la stringa è rappresentata da un array e infine ne determina il nome.

Per il test sono stati istanziati i seguenti parametri:

```
public ClassUtilGetClassNameNewTest(String fullName, String expectedClassName)
```

```
@Parameters public static Collection<Object> data() {  
    return Arrays.asList(new Object[][] {  
        { null, null },  
        { "", "" },  
        { "int", "int" },  
        { "java.lang.String", "String" },  
        { "[Lorg.example.MyClass;," , "MyClass[]"] },  
        { "[Lorg.example.MyClass[][]," , "MyClass[][][]"] },  
        { "[Ljava.lang.Object;," , "Object[]"] },  
        { "[B", "byte[]"],  
        { "[C", "char[]"},  
        { "[D", "double[]"},  
        { "[F", "float[]"},  
        { "[I", "int[]"},  
        { "[J", "long[]"},  
        { "[S", "short[]"},  
        { "java.util.Map.Entry", "Entry" },  
        { "[[I", "int[][][]"] }  
    });
```

```
{ "[[Ljava.util.Map.Entry;," "Entry[]][]"] },  
});
```

Ogni elemento è un array di due oggetti: una stringa rappresentante il nome completo della classe e una stringa contenente il nome atteso della classe(senza il package). Il test verifica che, per ciascun input, il metodo `getClassName()` restituisca il nome atteso della classe. In altre parole, il test verifica che il metodo `ClassUtil.getClassName()` sia in grado di estrarre il nome della classe a partire dal suo nome completo (`fullName`) restituendo solo il nome della classe (`expectedClassName`) senza il package.

5.1 Coverage e Mutation Coverage

Con questo test viene raggiunto, per il metodo testato, valore di Coverage pari al 100% per le Missed Istruction , 95% per le Missed Branches, Data Flow Coverage pari al 94% e 81% di mutation coverage con 6 mutanti sopravvissuti su 31 istanziati.

6 Link Utili

6.1 Bookkeeper

GitHub: <https://github.com/giulia97-w/bookkeeper-1>

Pit, Jacoco and ba-dua reports: <https://github.com/giulia97-w/bookkeeper-1/actions/runs/5257015128>

SonarCloud: https://sonarcloud.io/project/overview?id=giulia97-w_bookkeeper-1

6.2 OpenJPA

GitHub: https://github.com/giulia97-w/apache_openjpa

SonarCloud: https://sonarcloud.io/project/overview?id=giulia97-w_apache_openjpa

Pit and Jacoco reports: https://github.com/giulia97-w/apache_openjpa/actions/runs/5257088570

Ba-dua report: https://github.com/giulia97-w/apache_openjpa/actions/runs/5257088573

```

@Parameters
public static Collection<Object[]> data() {

    DBIdentifier s1 = DBIdentifier.newTable("table");
    DBIdentifier s2 = DBIdentifier.newSchema("schema");
    DBIdentifier s3 = DBIdentifier.newTable("table");
    QualifiedDBIdentifier path1 = QualifiedDBIdentifier newPath(s2, s1);
    QualifiedDBIdentifier path2 = QualifiedDBIdentifier newPath(s2, s1);
    QualifiedDBIdentifier path3 = QualifiedDBIdentifier newPath(s2, s3);
    QualifiedDBIdentifier path4 = QualifiedDBIdentifier newPath(s1);
    String pathString = s2.toString() + "." + s3.toString();
}

```

Figure 1: Definizione Parametri QualifiedIdentifierEqualTest

BufferedChannel

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Ctry	Missed	Lines	Missed	Methods
read(ByteBuffer, long)	90%	71%	7 15	4 39	0 3	1 1	1 1	1 1	1 1	1 1
clear()	0%	n/a	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1
getFileChannelPosition()	0%	n/a	1 1	1 1	1 1	1 1	1 1	1 1	1 1	1 1
write(ByteBuffer)	100%	100%	0 4	0 14	0 0	0 0	0 0	0 0	0 0	0 0
BufferedChannel(FileChannel, int, int)	100%	n/a	0 1	0 8	0 0	0 0	0 0	0 0	0 0	0 0
flushInternal()	100%	50%	1 2	0 6	0 0	0 0	0 0	0 0	0 0	0 0
flush(boolean)	100%	50%	1 2	0 6	0 0	0 0	0 0	0 0	0 0	0 0
forceWrite(boolean)	100%	n/a	0 1	0 3	0 0	0 0	0 0	0 0	0 0	0 0
BufferedChannel(FileChannel, int)	100%	n/a	0 1	0 2	0 0	0 0	0 0	0 0	0 0	0 0
position()	100%	n/a	0 1	0 1	0 0	0 0	0 0	0 0	0 0	0 0
Total	31 of 381	91%	10 of 38	73%	11 29	8 83	2 2	10 10	2 2	10 10

Figure 2: Coverage relativa ai metodi della classe BufferedChannel

```

/**
 * Write all the data in src to the {@link FileChannel}. Note that this function can
 * buffer or re-order writes based on the implementation. These writes will be flushed
 * to the disk only when flush() is invoked.
 *
 * @param src The source ByteBuffer which contains the data to be written.
 * @throws IOException if a write operation fails.
 */
public synchronized void write(ByteBuffer src) throws IOException {
    int copied = 0;
    while (src.remaining() > 0) {
        int truncated = 0;
        if (writeBuffer.remaining() < src.remaining()) {
            truncated = src.remaining() - writeBuffer.remaining();
            src.limit(src.limit() - truncated);
        }
        copied += src.remaining();
        writeBuffer.put(src);
        src.limit(src.limit() + truncated);
        // if we have run out of buffer space, we should flush to the file
        if (writeBuffer.remaining() == 0) {
            flushInternal();
        }
    }
    position += copied;
}

```

Figure 3: Coverage per il metodo write()

```

@Override
public synchronized int read(ByteBuffer dest, long pos) throws IOException {
    long prevPos = pos;
    while (dest.remaining() > 0) {
        // check if it is in the write buffer
        if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
            long positionInBuffer = pos - writeBufferStartPosition.get();
            long bytesToCopy = writeBuffer.position() - positionInBuffer;
            if (bytesToCopy > dest.remaining()) {
                bytesToCopy = dest.remaining();
            }
            if (bytesToCopy == 0) {
                throw new IOException("Read past EOF");
            }
            ByteBuffer src = writeBuffer.duplicate();
            src.position((int) positionInBuffer);
            src.limit((int) (positionInBuffer + bytesToCopy));
            dest.put(src);
            pos += bytesToCopy;
        } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
            // here we reach the end
            break;
        } else if (readBufferStartPosition <= pos < readBufferStartPosition + readBuffer.capacity()) {
            long positionInBuffer = pos - readBufferStartPosition;
            long bytesToCopy = readBuffer.capacity() - positionInBuffer;
            if (bytesToCopy > dest.remaining()) {
                bytesToCopy = dest.remaining();
            }
            ByteBuffer src = readBuffer.duplicate();
            src.position((int) positionInBuffer);
            src.limit((int) (positionInBuffer + bytesToCopy));
            dest.put(src);
            pos += bytesToCopy;
            // let's read it
        } else {
            readBufferStartPosition = pos;
            readBuffer.clear();
            // make sure that we don't overlap with the write buffer
            if (readBufferStartPosition + readBuffer.capacity() >= writeBufferStartPosition.get()) {
                readBufferStartPosition = writeBufferStartPosition.get() - readBuffer.capacity();
            }
            if (readBufferStartPosition < 0) {
                ZeroBuffer.put(readBuffer, (int) -readBufferStartPosition);
            }
            }
            while (readBuffer.remaining() > 0) {
                if (fileChannel.read(readBuffer, readBufferStartPosition + readBuffer.position()) <= 0) {
                    throw new IOException("Short read");
                }
            }
            ZeroBuffer.put(readBuffer);
            readBuffer.clear();
        }
    }
    return (int) (pos - prevPos);
}

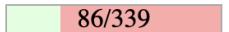
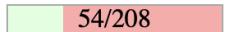
```

Figure 4: Coverage per il metodo read()

Pit Test Coverage Report

Package Summary

org.apache.bookkeeper.bookie

Number of Classes	Line Coverage	Mutation Coverage
3	25%  86/339	26%  54/208

Breakdown by Class

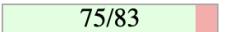
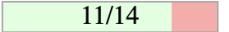
Name	Line Coverage	Mutation Coverage
BufferedChannel.java	90%  75/83	68%  43/63
EntryKey.java	79%  11/14	100%  11/11

Figure 5: Mutation coverage per le classi BufferedChannel ed EntryKey

```

    /*
     * public synchronized void write(ByteBuffer src) throws IOException {
     *     int copied = 0;
     *     while (src.remaining() > 0) {
     *         int truncated = 0;
     *         if (writeBuffer.remaining() < src.remaining()) {
     *             truncated = src.remaining() - writeBuffer.remaining();
     *             src.limit(src.limit() - truncated);
     *         }
     *         copied += src.remaining();
     *         writeBuffer.put(src);
     *         src.limit(src.limit() + truncated);
     *         // if we have run out of buffer space, we should flush to the file
     *         if (writeBuffer.remaining() == 0) {
     *             flushInternal();
     *         }
     *     }
     *     position += copied;
     * }

```

Figure 6: Mutation coverage per il metodo write()

```

} @Override
public synchronized int read(ByteBuffer dest, long pos) throws IOException {
    long prevPos = pos;
    while (dest.remaining() > 0) {
        // check if it is in the write buffer
        if (writeBuffer != null && writeBufferStartPosition.get() <= pos) {
            long positionInBuffer = pos - writeBufferStartPosition.get();
            long bytesToCopy = writeBuffer.position() - positionInBuffer;
            if (bytesToCopy > dest.remaining()) {
                bytesToCopy = dest.remaining();
            }
            if (bytesToCopy == 0) {
                throw new IOException("Read past EOF");
            }
            ByteBuffer src = writeBuffer.duplicate();
            src.position((int) positionInBuffer);
            src.limit((int) (positionInBuffer + bytesToCopy));
            dest.put(src);
            pos += bytesToCopy;
        } else if (writeBuffer == null && writeBufferStartPosition.get() <= pos) {
            // here we reach the end
            break;
        } else check if there is anything we can grab from the readBuffer
        if (readBufferStartPosition <= pos && pos < readBufferStartPosition + readBuffer.capacity()) {
            long positionInBuffer = pos - readBufferStartPosition;
            long bytesToCopy = readBuffer.capacity() - positionInBuffer;
            if (bytesToCopy > dest.remaining()) {
                bytesToCopy = dest.remaining();
            }
            ByteBuffer src = readBuffer.duplicate();
            src.position((int) positionInBuffer);
            src.limit((int) (positionInBuffer + bytesToCopy));
            dest.put(src);
            pos += bytesToCopy;
            // let's read it
        } else {
            readBufferStartPosition = pos;
            readBuffer.clear();
        }
        // make sure that we don't overlap with the write buffer
        if (readBufferStartPosition + readBuffer.capacity() >= writeBufferStartPosition.get()) {
            readBufferStartPosition = writeBufferStartPosition.get() - readBuffer.capacity();
            if (readBufferStartPosition < 0) {
                zeroBuffer.put(readBuffer, (int) -readBufferStartPosition);
            }
        }
        while (readBuffer.remaining() > 0) {
            if (fileChannel.read(readBuffer, readBufferStartPosition + readBuffer.position()) <= 0) {
                throw new IOException("Short read");
            }
        }
        zeroBuffer.put(readBuffer);
        readBuffer.clear();
    }
    return (int) (pos - prevPos);
}

```

Figure 7: Mutation coverage per il metodo read()

EntryKey

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• equals(Object)		100%		100%	0	4	0	4	0	1
• hashCode()		100%		n/a	0	1	0	1	0	1
• EntryKey(long, long)		100%		n/a	0	1	0	4	0	1
• EntryKey()		100%		n/a	0	1	0	2	0	1
• static (...)		100%		n/a	0	1	0	1	0	1
• getLedgerId()		100%		n/a	0	1	0	1	0	1
• getEntryId()		100%		n/a	0	1	0	1	0	1
Total	0 of 60	100%	0 of 6	100%	0	10	0	14	0	7

Figure 8: Coverage per la classe EntryKey di bookkeeper

```

34     public EntryKey() {
35         this(0, 0);
36     }
37
38     public EntryKey(long ledgerId, long entryId) {
39         this.ledgerId = ledgerId;
40         this.entryId = entryId;
41     }
42
43     public long getLedgerId() {
44 1    return ledgerId;
45 }
46
47     public long getEntryId() {
48 1    return entryId;
49 }
50
51     /**
52      * Comparator for the key portion.
53      */
54     public static final KeyComparator COMPARATOR = new KeyComparator();
55
56     // Only compares the key portion
57     @Override
58     public boolean equals(Object other) {
59 1    if (!(other instanceof EntryKey)) {
60 1        return false;
61    }
62    EntryKey key = (EntryKey) other;
63 3    return ledgerId == key.ledgerId && entryId == key.entryId;
64  }
65
66     @Override
67     public int hashCode() {
68 4    return (int) (ledgerId * 13 ^ entryId * 17);
69  }

```

Figure 9: Mutation Coverage la classe EntryKey di Bookkeeper

LedgerEntriesImpl

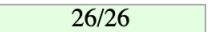
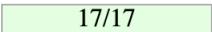
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
• getEntry(long)		100%		100%	0	3	0	6	0	1
• create(List)		100%		100%	0	2	0	4	0	1
• releaseByteBuf()		100%		100%	0	2	0	5	0	1
• iterator()		100%		n/a	0	1	0	2	0	1
• recycle()		100%		n/a	0	1	0	3	0	1
• LedgerEntriesImpl(Recycler.Handle)		100%		n/a	0	1	0	3	0	1
• static (...)		100%		n/a	0	1	0	1	0	1
• close()		100%		n/a	0	1	0	2	0	1
Total	0 of 123	100%	0 of 8	100%	0	12	0	26	0	8

Figure 10: Coverage per la classe LedgerEntriesImpl di bookkeeper

Pit Test Coverage Report

Package Summary

org.apache.bookkeeper.client.impl

Number of Classes	Line Coverage	Mutation Coverage
1	100%  26/26	100%  17/17

Breakdown by Class

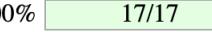
Name	Line Coverage	Mutation Coverage
LedgerEntriesImpl.java	100%  26/26	100%  17/17

Figure 11: Mutation Coverage per la classe LedgerEntriesImpl() del progetto bookkeeper

```
79     @Override
80     public LedgerEntry getEntry(long entryId) {
81         checkNotNull(entries, "entries has been recycled");
82         long firstId = entries.get(0).getEntryId();
83         long lastId = entries.get(entries.size() - 1).getEntryId();
84         if (entryId < firstId || entryId > lastId) {
85             throw new IndexOutOfBoundsException("required index: " + entryId +
86                     " is out of bounds: [ " + firstId + ", " + lastId + " ].");
87         }
88         return entries.get((int) (entryId - firstId));
89     }
```

Figure 12: Mutation Coverage per il metodo getEntry() della classe LedgerEntriesImpl del progetto bookkeeper

```
69     public static LedgerEntriesImpl create(List<LedgerEntry> entries) {
70         checkArgument(!entries.isEmpty(), "entries for create should not be empty.");
71         LedgerEntriesImpl ledgerEntries = RECYCLER.get();
72         ledgerEntries.entries = entries;
73         return ledgerEntries;
74     }
75 }
```

Figure 13: Mutation Coverage per il metodo create() della classe LedgerEntriesImpl del progetto bookkeeper

ClassUtil

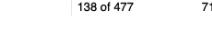
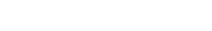
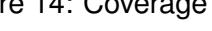
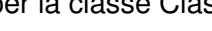
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Oqty	Missed	Lines	Missed	Methods
toClass(String, boolean, ClassLoader)		0%		0%	13	13	29	29	1	1
getPackageName(Class)		0%		0%	2	2	1	1	1	1
toClass(String, ClassLoader)		0%		n/a	1	1	1	1	1	1
static (...)		100%		95%	1	11	0	25	0	1
getClassName(String)		100%		100%	0	6	0	11	0	1
getPackageName(String)		100%		100%	0	2	0	4	0	1
getArrayDimensions(String)		100%		100%	0	2	0	3	0	1
getClassName(Class)		100%		100%	0	2	0	3	0	1
Total	138 of 477	71%	27 of 60	55%	17	38	31	75	3	8

Figure 14: Coverage per la classe ClassUtil di OpenJPA

Pit Test Coverage Report

Package Summary

org.apache.openjpa.lib.util

Number of Classes	Line Coverage	Mutation Coverage
1	57% 	55% 

Breakdown by Class

Name	Line Coverage	Mutation Coverage
ClassUtil.java	57% 	55% 

Figure 15: Mutation Coverage per la classe ClassUtil di OpenJPA

```

--+
118     public static String getClassName(Class cls) {
119 1       if (cls == null) {
120 1         return null;
121     }
122 1       return getClassName(cls.getName());
123   }
124
125 /**
126  * Return only the class name.
127  */
128 public static String getClassName(String fullName) {
129 1       if (fullName == null) {
130 1         return null;
131     }
132 1       if (fullName.isEmpty()) {
133 1         return fullName;
134     }
135
136       int dims = getArrayDimensions(fullName);
137 2       if (dims > 0) {
138 2         if (fullName.length() == dims + 1) {
139
140             String classCode = fullName.substring(dims);
141 3           for (int i = 0; i < _codes.length; i++) {
142 1             if (_codes[i][2].equals(classCode)) {
143                 fullName = (String)_codes[i][1];
144                 break;
145             }
146         }
147     }
148     else {
149 2       if (fullName.charAt(fullName.length()-1) == ';') {
150 2         fullName = fullName.substring(dims + 1, fullName.length() - 1);
151     }
152     else {
153 1         fullName = fullName.substring(dims + 1);
154     }
155   }
156 }
157
158   int lastDot = fullName.lastIndexOf('.');
159 3   String simpleName = lastDot > -1 ? fullName.substring(lastDot + 1) : fullName;
160
161 2   if (dims > 0) {
162 2     StringBuilder sb = new StringBuilder(simpleName.length() + dims * 2);
163     sb.append(simpleName);
164 3     for (int i = 0; i < dims; i++) {
165         sb.append("[]");
166     }
167     simpleName = sb.toString();
168   }
169 1   return simpleName;
170 }
171

```

Figure 16: Mutation Coverage per i metodi `getClassName(String)` e `getClassName(Class)` della classe `ClassUtil` di OpenJPA

```

191     public static String getPackageName(String fullName) {
192     1       if (fullName == null) {
193     1           return null;
194     }
195     1       if (fullName.isEmpty()) {
196     1           return fullName;
197     }
198
199     int dims = getArrayDimensions(fullName);
200 2       if (dims > 0) {
201 2           if (fullName.length() == dims + 1) {
202             // don't care, it's a primitive
203 1           return "";
204         }
205         else {
206 1           fullName = fullName.substring(dims + 1);
207         }
208     }
209
210     int lastDot = fullName.lastIndexOf('.');
211 3       return lastDot > -1 ? fullName.substring(0, lastDot) : "";
212   }
213 }
```

Figure 17: Coverage per il metodo getPackageName() della classe ClassUtil del progetto OpenJPA

```

187     @Override
188     public boolean equals(Object obj) {
189 1       if (obj == null) {
190 1           return false;
191     }
192 1       if (obj instanceof QualifiedDBIdentifier) {
193         QualifiedDBIdentifier sPath = (QualifiedDBIdentifier)obj;
194 2       return DBIdentifier.equal(sPath.getSchemaName(), getSchemaName()) &&
195 1           DBIdentifier.equal(sPath.getObjectTableName(), getObjectTableName()) &&
196 1           DBIdentifier.equal(sPath, this);
197
198     }
199 1       else if (obj instanceof DBIdentifier) {
200       DBIdentifier sName = (DBIdentifier)obj;
201 1       return DBIdentifier.equal(sName, this);
202 1     } else if (obj instanceof String) {
203 1       return obj.equals(this.getName());
204     }
205     throw new IllegalArgumentException("Cannot compare to type: " + obj.getClass().getName());
206   }
```

Figure 18: Mutation Coverage per il metodo equals() della classe QualifiedDBIdentifier del progetto OpenJPA

QualifiedDBIdentifier

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Cov.	Missed Lines	Cov.	Missed Methods
isDelimited()	██████	0%	██████	0%	6	6	9	9	1
compareTo(DBIdentifier)	██████	0%	██████	0%	4	4	5	5	1
getPath(DBIdentifier)	██████	0%	██████	0%	2	2	3	3	1
isUnqualifiedColumn()	██████	0%	██████	0%	3	3	1	1	1
getUnqualifiedName()	██████	0%	n/a	1	1	4	4	1	1
length()	██████	0%	██████	0%	2	2	4	4	1
setName(String)	██████	0%	n/a	1	1	3	3	1	1
isUnqualifiedObject()	██████	0%	n/a	1	1	1	1	1	1
setPath(DBIdentifier[])	██████████	98%	██████	80%	4	11	1	21	0
splitPath(DBIdentifier[])	██████████	100%	██████	83%	2	7	0	13	0
equals(Object)	██████████	100%	██████	85%	2	8	0	13	0
pathEqual(QualifiedDBIdentifier, QualifiedDBIdentifier)	██████████	100%	██████	100%	0	7	0	12	0
clone()	██████████	100%	n/a	0	1	0	7	0	1
getName()	██████████	100%	██████	75%	1	3	0	4	0
QualifiedDBIdentifier(DBIdentifier[])	██████████	100%	n/a	0	1	0	3	0	1
resetNames()	██████████	100%	n/a	0	1	0	1	0	1
newPath(DBIdentifier[])	██████████	100%	n/a	0	1	0	2	0	1
setSchemaName(DBIdentifier)	██████████	100%	n/a	0	1	0	2	0	1
setObjectTableName(DBIdentifier)	██████████	100%	n/a	0	1	0	2	0	1
setBaseName(String)	██████████	100%	n/a	0	1	0	2	0	1
getSchemaName()	██████████	100%	n/a	0	1	0	1	0	1
getObjectTableName()	██████████	100%	n/a	0	1	0	1	0	1
toString()	██████████	100%	n/a	0	1	0	1	0	1
getBaseName()	██████████	100%	n/a	0	1	0	1	0	1
getIdentifier()	██████████	100%	n/a	0	1	0	1	0	1
Total	112 of 460	75%	33 of 86	61%	29	68	31	120	8
									25

Figure 19: Coverage per la classe QualifiedDBIdentifier di OpenJPA

Pit Test Coverage Report

Package Summary

org.apache.openjpa.jdbc.identifier

Number of Classes	Line Coverage	Mutation Coverage
1	74% 89/120	63% 63/100

Breakdown by Class

Name	Line Coverage	Mutation Coverage
QualifiedDBIdentifier.java	74% 89/120	63% 63/100

Figure 20: Mutation Coverage per la classe QualifiedDBIdentifier del progetto openJPA

```
/*
 * public static DBIdentifier[] splitPath(DBIdentifier sName) {
2   if (sName instanceof QualifiedDBIdentifier && sName.getType() != DBIdentifierType.SCHEMA) {
      QualifiedDBIdentifier path = (QualifiedDBIdentifier)sName;
      List<DBIdentifier> names = new ArrayList<>();
1     if (!DBIdentifier.isNull(path.getSchemaName())) {
        names.add(path.getSchemaName().clone());
      }
1     if (!DBIdentifier.isNull(path.getObjectTableName())) {
        names.add(path.getObjectTableName().clone());
      }
1     if (!DBIdentifier.isNull(path.getIdentifier())) {
        names.add(((DBIdentifier)path).clone());
      }
1     return names.toArray(new DBIdentifier[names.size()]);
    }
1   if (sName instanceof DBIdentifier) {
1     return new DBIdentifier[] { sName.clone() };
    }
1   return new DBIdentifier[] {};
}
```

Figure 21: Mutation Coverage per il metodo splitPath() della classe QualifiedDBIdentifier del progetto OpenJPA

```
213   public static boolean pathEqual(QualifiedDBIdentifier path1, QualifiedDBIdentifier path2) {
214     if (path1 == null && path2 == null) {
215       return true;
216     }
217     if (path1 == null) {
218       return false;
219     }
220     DBIdentifier[] names1 = QualifiedDBIdentifier.splitPath(path1);
221     DBIdentifier[] names2 = QualifiedDBIdentifier.splitPath(path2);
222     if (names1.length != names2.length) {
223       return false;
224     }
225     for (int i = 0; i < names1.length; i++) {
226       if (!DBIdentifier.equal(names1[i], names2[i])) {
227         return false;
228       }
229     }
230     return true;
231   }
---
```

Figure 22: Mutation Coverage per il metodo pathEqual() della classe QualifiedDBIdentifier del progetto OpenJPA

6.3 Ba-dua-report

```
<method name="write" desc="(Ljava/nio/ByteBuffer;)V">
<du var="this" def="68" use="83" covered="1"/>
<du var="this" def="68" use="71" target="72" covered="1"/>
<du var="this" def="68" use="71" target="75" covered="1"/>
<du var="this" def="68" use="76" covered="1"/>
<du var="this" def="68" use="79" target="80" covered="1"/>
<du var="this" def="68" use="79" target="82" covered="1"/>
<du var="this" def="68" use="80" covered="1"/>
<du var="this" def="68" use="72" covered="1"/>
<du var="src" def="68" use="69" target="70" covered="1"/>
<du var="src" def="68" use="69" target="83" covered="1"/>
<du var="src" def="68" use="71" target="72" covered="1"/>
<du var="src" def="68" use="71" target="75" covered="1"/>
<du var="src" def="68" use="75" covered="1"/>
<du var="src" def="68" use="76" covered="1"/>
<du var="src" def="68" use="77" covered="1"/>
<du var="src" def="68" use="72" covered="1"/>
<du var="src" def="68" use="73" covered="1"/>
<du var="this.writeBuffer" def="68" use="71" target="72" covered="1"/>
<du var="this.writeBuffer" def="68" use="71" target="75" covered="1"/>
<du var="this.writeBuffer" def="68" use="76" covered="1"/>
<du var="this.writeBuffer" def="68" use="79" target="80" covered="1"/>
<du var="this.writeBuffer" def="68" use="79" target="82" covered="1"/>
<du var="this.writeBuffer" def="68" use="72" covered="1"/>
<du var="this.position" def="68" use="83" covered="1"/>
<du var="copied" def="68" use="83" covered="1"/>
<du var="copied" def="68" use="75" covered="1"/>
<du var="truncated" def="70" use="77" covered="1"/>
<du var="truncated" def="72" use="77" covered="1"/>
<du var="copied" def="75" use="83" covered="1"/>
<du var="copied" def="75" use="75" covered="1"/>
<counter type="DU" missed="0" covered="30"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figure 23: ba-dua report per il metodo write() della classe BufferedChannel con DFCoverage = 100%

Il metodo read() della classe BufferedChannel ha un report molto lungo per questa ragione si rimanda il lettore al repository di GitHub (sezione Actions di ogni progetto) dove è presente il report salvato come artifact.

```

<method name="equals" desc="(Ljava/lang/Object;)Z">
    <du var="this" def="59" use="63" target="63" covered="1"/>
    <du var="other" def="59" use="59" target="60" covered="1"/>
    <du var="other" def="59" use="59" target="62" covered="1"/>
    <du var="other" def="59" use="62" covered="1"/>
    <du var="this.ledgerId" def="59" use="63" target="63" covered="1"/>
    <du var="this.ledgerId" def="59" use="63" target="63" covered="1"/>
    <du var="this.entryId" def="59" use="63" target="63" covered="1"/>
    <du var="this.entryId" def="59" use="63" target="63" covered="1"/>
    <du var="key" def="62" use="63" target="63" covered="1"/>
    <du var="key.ledgerId" def="62" use="63" target="63" covered="1"/>
    <du var="key.ledgerId" def="62" use="63" target="63" covered="1"/>
    <du var="key.entryId" def="62" use="63" target="63" covered="1"/>
    <du var="key.entryId" def="62" use="63" target="63" covered="1"/>
    <counter type="DU" missed="0" covered="19"/>
    <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figure 24: ba-dua report per il metodo equals() della classe EntryKey con DFCoverage = 100%

```

<method name="create" desc="(Ljava/util/List;)Lorg/apache/bookkeeper/client/impl/LedgerEntriesImpl;">
    <du var="entries" def="70" use="70" target="70" covered="1"/>
    <du var="entries" def="70" use="70" target="70" covered="0"/>
    <du var="entries" def="70" use="72" covered="1"/>
    <du var="RECYCLER" def="70" use="71" covered="1"/>
    <counter type="DU" missed="1" covered="3"/>
    <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figure 25: ba-dua report per il metodo create() della classe LedgerEntriesImpl con DFCoverage = 75%

```

<method name="getEntry" desc="(J)Lorg/apache/bookkeeper/client/api/LedgerEntry;">
    <du var="this" def="81" use="88" covered="1"/>
    <du var="entryId" def="81" use="84" target="84" covered="1"/>
    <du var="entryId" def="81" use="84" target="85" covered="1"/>
    <du var="entryId" def="81" use="85" covered="1"/>
    <du var="entryId" def="81" use="84" target="85" covered="1"/>
    <du var="entryId" def="81" use="84" target="88" covered="1"/>
    <du var="entryId" def="81" use="88" covered="1"/>
    <du var="this.entries" def="81" use="88" covered="1"/>
    <du var="firstId" def="82" use="84" target="84" covered="1"/>
    <du var="firstId" def="82" use="84" target="85" covered="1"/>
    <du var="firstId" def="82" use="85" covered="1"/>
    <du var="firstId" def="82" use="88" covered="1"/>
    <du var="lastId" def="83" use="85" covered="1"/>
    <du var="lastId" def="83" use="84" target="85" covered="1"/>
    <du var="lastId" def="83" use="84" target="88" covered="1"/>
    <counter type="DU" missed="0" covered="15"/>
    <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figure 26: ba-dua report per il metodo getEntry() della classe LedgerEntriesImpl con DFCoverage = 100%

```

<method name="splitPath" desc="(Lorg/apache/openjpa/jdbc/identifier/DBIdentifier;)Lorg/apache/openjpa/jdbc/identifier/DBIdentifier;">
    <du var="sName" def="105" use="105" target="105" covered="1"/>
    <du var="sName" def="105" use="105" target="120" covered="1"/>
    <du var="sName" def="105" use="120" target="121" covered="1"/>
    <du var="sName" def="105" use="120" target="123" covered="1"/>
    <du var="sName" def="105" use="121" covered="1"/>
    <du var="sName" def="105" use="105" target="106" covered="1"/>
    <du var="sName" def="105" use="105" target="120" covered="0"/>
    <du var="sName" def="105" use="106" covered="1"/>
    <du var="SCHEMA" def="105" use="105" target="106" covered="1"/>
    <du var="SCHEMA" def="105" use="105" target="120" covered="0"/>
    <du var="path" def="106" use="109" target="110" covered="1"/>
    <du var="path" def="106" use="109" target="112" covered="1"/>
    <du var="path" def="106" use="112" target="113" covered="1"/>
    <du var="path" def="106" use="112" target="115" covered="1"/>
    <du var="path" def="106" use="115" target="116" covered="1"/>
    <du var="path" def="106" use="115" target="118" covered="0"/>
    <du var="path" def="106" use="116" covered="1"/>
    <du var="path" def="106" use="113" covered="1"/>
    <du var="path" def="106" use="110" covered="1"/>
    <du var="names" def="107" use="118" covered="1"/>
    <du var="names" def="107" use="116" covered="1"/>
    <du var="names" def="107" use="113" covered="1"/>
    <du var="names" def="107" use="110" covered="1"/>
    <counter type="DU" missed="3" covered="20"/>
    <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figure 27: ba-dua report per il metodo splitPath() della classe QualifiedDBIdentifier con DFCoverage = 86,95%

```
<method name="equals" desc="(Ljava/lang/Object;)Z">
    <du var="this" def="188" use="202" covered="1"/>
    <du var="this" def="188" use="200" covered="1"/>
    <du var="this" def="188" use="193" target="193" covered="1"/>
    <du var="this" def="188" use="193" target="195" covered="1"/>
    <du var="this" def="188" use="194" target="194" covered="1"/>
    <du var="this" def="188" use="194" target="195" covered="0"/>
    <du var="this" def="188" use="195" target="195" covered="1"/>
    <du var="this" def="188" use="195" target="195" covered="0"/>
    <du var="obj" def="188" use="188" target="189" covered="1"/>
    <du var="obj" def="188" use="188" target="191" covered="1"/>
    <du var="obj" def="188" use="191" target="192" covered="1"/>
    <du var="obj" def="188" use="191" target="198" covered="1"/>
    <du var="obj" def="188" use="198" target="199" covered="1"/>
    <du var="obj" def="188" use="198" target="201" covered="1"/>
    <du var="obj" def="188" use="201" target="202" covered="1"/>
    <du var="obj" def="188" use="201" target="204" covered="1"/>
    <du var="obj" def="188" use="204" covered="1"/>
    <du var="obj" def="188" use="202" covered="1"/>
    <du var="obj" def="188" use="199" covered="1"/>
    <du var="obj" def="188" use="192" covered="1"/>
    <du var="sPath" def="192" use="193" target="193" covered="1"/>
    <du var="sPath" def="192" use="193" target="195" covered="1"/>
    <du var="sPath" def="192" use="194" target="194" covered="1"/>
    <du var="sPath" def="192" use="194" target="195" covered="0"/>
    <du var="sPath" def="192" use="195" target="195" covered="1"/>
    <du var="sPath" def="192" use="195" target="195" covered="0"/>
    <counter type="DU" missed="4" covered="22"/>
    <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figure 28: ba-dua report per il metodo equals() della classe QualifiedDBIdentifier con DFCoverage = 84,61%

```

<method name="pathEqual" desc="(Lorg/apache/openjpa/jdbc/identifier/QualifiedDBIdentifier;Lorg/apache/openjpa/jdbc/identifier/QualifiedDBIdentifier;)Z">
<du var="path1" def="213" use="213" target="213" covered="1"/>
<du var="path1" def="213" use="213" target="216" covered="1"/>
<du var="path1" def="213" use="216" target="217" covered="1"/>
<du var="path1" def="213" use="216" target="219" covered="1"/>
<du var="path1" def="213" use="219" covered="1"/>
<du var="path2" def="213" use="220" covered="1"/>
<du var="path2" def="213" use="213" target="214" covered="1"/>
<du var="path2" def="213" use="213" target="216" covered="1"/>
<du var="names1" def="219" use="221" target="222" covered="1"/>
<du var="names1" def="219" use="221" target="224" covered="1"/>
<du var="names1" def="219" use="224" target="225" covered="1"/>
<du var="names1" def="219" use="224" target="229" covered="1"/>
<du var="names1" def="219" use="225" target="226" covered="1"/>
<du var="names1" def="219" use="225" target="224" covered="1"/>
<du var="names2" def="220" use="221" target="222" covered="1"/>
<du var="names2" def="220" use="221" target="224" covered="1"/>
<du var="names2" def="220" use="225" target="226" covered="1"/>
<du var="names2" def="220" use="225" target="224" covered="1"/>
<du var="names2" def="220" use="224" target="225" covered="1"/>
<du var="names2" def="220" use="224" target="229" covered="1"/>
<du var="names2" def="220" use="224" target="226" covered="1"/>
<du var="names2" def="220" use="224" target="225" covered="1"/>
<du var="1" def="224" use="224" target="225" covered="1"/>
<du var="1" def="224" use="224" target="229" covered="0"/>
<du var="1" def="224" use="225" target="226" covered="1"/>
<du var="1" def="224" use="225" target="224" covered="1"/>
<du var="1" def="224" use="224" target="225" covered="1"/>
<du var="1" def="224" use="224" target="229" covered="1"/>
<du var="1" def="224" use="225" target="226" covered="1"/>
<du var="1" def="224" use="225" target="224" covered="1"/>
<du var="1" def="224" use="224" target="229" covered="1"/>
<du var="1" def="224" use="224" target="226" covered="1"/>
<du var="1" def="224" use="224" target="225" covered="1"/>
<du var="1" def="224" use="224" target="229" covered="1"/>
<du var="1" def="224" use="224" covered="1"/>
<du var="1" def="224" use="224" target="229" covered="1"/>
<counter type="DU" missed="1" covered="27"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figure 29: ba-dua report per il metodo pathEqual() della classe QualifiedDBIdentifier con DFCoverage = 96,42%

```

<method name="getPackageName" desc="(Ljava/lang/String;)Ljava/lang/String;">
<du var="fullName" def="192" use="192" target="193" covered="1"/>
<du var="fullName" def="192" use="192" target="195" covered="1"/>
<du var="fullName" def="192" use="195" target="196" covered="1"/>
<du var="fullName" def="192" use="195" target="199" covered="1"/>
<du var="fullName" def="192" use="199" covered="1"/>
<du var="fullName" def="192" use="210" covered="1"/>
<du var="fullName" def="192" use="211" covered="1"/>
<du var="fullName" def="192" use="201" target="203" covered="1"/>
<du var="fullName" def="192" use="201" target="206" covered="1"/>
<du var="fullName" def="192" use="206" covered="1"/>
<du var="fullName" def="192" use="196" covered="1"/>
<du var="dims" def="199" use="200" target="201" covered="1"/>
<du var="dims" def="199" use="200" target="210" covered="1"/>
<du var="dims" def="199" use="201" target="203" covered="1"/>
<du var="dims" def="199" use="201" target="206" covered="1"/>
<du var="dims" def="199" use="206" covered="1"/>
<du var="fullName" def="206" use="210" covered="1"/>
<du var="fullName" def="206" use="211" covered="1"/>
<du var="lastDot" def="210" use="211" target="211" covered="1"/>
<du var="lastDot" def="210" use="211" target="211" covered="1"/>
<du var="lastDot" def="210" use="211" covered="1"/>
<counter type="DU" missed="0" covered="21"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figure 30: ba-dua report per il metodo getPackageName() della classe ClassUtil con DFCoverage=100%

```
<method name="getClassName" desc="(Ljava/lang/Class;)Ljava/lang/String;">
    <du var="cls" def="119" use="119" target="120" covered="1"/>
    <du var="cls" def="119" use="119" target="122" covered="1"/>
    <du var="cls" def="119" use="122" covered="1"/>
    <counter type="DU" missed="0" covered="3"/>
    <counter type="METHOD" missed="0" covered="1"/>
</method>
```

Figure 31: ba-dua report per il metodo `getClassName(Class)` della classe `ClassUtil` con DFCoverage=100%

Il metodo `getClassName(String)` presenta un report molto lungo si rimanda quindi il lettore al repository di GitHub.