Ingegneria del software 2

A.A. 2019/2020

0287567

Paolo Melissari

INTRODUZIONE

Il report contiene i risultati dell'attività di testing svolta su due progetti di Apache Software Fondation: Bookkeeper (un servizio di storage scalabile e tollerante ai guasti) ed Openjpa (un'implementazione della Java Persistence API).

Le attività di testing sono state inserite in un ciclo di Continuous Integration, utilizzando strumenti come Travis CI (necessario per effettuare il build dei progetti e lanciare i test) e SonarCloud (necessario a raccogliere delle metriche che indicano la bontà dei test).

Ogni test è stato implementato utilizzando il framework JUnit.

SCELTA DELLE CLASSI

Per la scelta delle classi di Bookkeeper, sono state selezionate:

- *LedgerHandle:*una classe che è risultata sempre buggata e con numero di fix commit superiore alla media.
- *LedgerEntriesImpl:*una classe che al contrario non veniva toccata da tante versioni e per la quale si vuole vedere se, con le versioni più recenti, può essere buggata.

Per la scelta delle classi di OpenJPA, sono state selezionate:

- *ProxyManagerImpl* :una classe che ha subito grandi cambiamenti in versioni più vecchie (nelle quali ha un valore di churn superiore alla media).
- *ImplHelper:* una classe che non veniva toccata da tante versioni e per la quale si vuole vedere se, con le versioni più recenti, può essere buggata.

CATEGORY PARTITION

In questa sezione verrà considerato in che modo è stato effettuato il category partition dei metodi delle classi prese in esame, ovvero in che modo è stato partizionato il dominio di input per avere un insieme di test con minor cardinalità possibile. Per ogni partizione poi, verrà scelto un campione con cui svolgere il test. In particolare, l'esperienza empirica mostra che i valori critici sono quelli ai confini delle classi di equivalenza e quindi è preferibile sceglierli come campione.

CATEGORY PARTITION- BOOKKEEPER

BOOKKEEPER-LedgerEntriesImpl

LedgerEntriesImpl- LedgerEntry getEntry(long entryId)

Metodo che ritorna l'entry specificata da *entryId*.

- Long entryId. Id dell'entry da ritornare. Verrà considerato il caso in cui l' ID è negativo, quello in cui è = 0 e quello in cui è >0 .

Le classi di equivalenza individuate sono: {entryId <0; entryId = 0; entryId>0}
Analisi Boundary-Value: {entryId = -1; entryId = 0; entryId = 1}

Test suite minimale: {entryId = -1}, {entryId = 0}, {entryId = 1}

LedgerEntriesImpl- LedgerEntriesImpl create(List<LedgerEntry> entries)

Metodo che crea una lista di Ledger Entry.

- *List<LedgerEntry> entries*. Lista contenenti le Entry da inserire nella lista. Verrà considerato il caso in cui entries è null, il caso in cui entries è vuoto ed il caso in cui entries contiene almeno un elemento Le classi di equivalenza individuate sono: {entries = null; entries.length = 0; entries.length >0}
- *Analisi Boundary-Value*: {null, entries.length = 0; entries. length = 1}

Test suite minimale: {null}, {entries.length = 0}, {entries. length = 1}

BOOKKEEPER-LedgerHandle

LedgerHandle- void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx)

Metodo che legge le entries appartenenti all'intervallo [firstEntry, lastEntry] in modo asincrono.

- *long firstEntry.* Id della prima entry da leggere. Per il category partition verrà considerato il caso in cui l' ID è negativo, quello in cui è = 0 e quello in cui è >0: {firstEntry<0; firstEntry = 0; firstEntry >0}.
  Analisi boundary value: {firstEntry = -1; firstEntry = 0; firstEntry = 1}
- *long lastEntry*. Id dell'ultima entry da leggere. Per il category partition verrà considerato il caso in cui l' ID è minore di firstEntry, quello in cui è uguale a firstEntry e quello in cui è maggiore di FirstEntry: { lastEntry <firstEntry; lastEntry = firstEntry; lastEntry >firstEntry}
  Analisi boundary value: {LastEntry = firstEntry-1; LastEntry = firstEnty; LastEntry = firstEntry+1}
- *ReadCallback cb*. Oggetto che implementa l'interfaccia ReadCallback. Per il category partition verrà considerato il caso in cui cb è null quello in cui è un'istanza valida (ovvero un'istanza che implementa l'interfaccia ReadCallback): {null, istanza valida}
  Analisi Boundary-Value: {null, istanza che implementa l'interfaccia ReadCallback}
- *Object ctx.* Context object, tiene traccia dello stato di alcune operazioni (ad esempio qual è l'handler del ledger da cui si stanno leggendo le entry). Per il category partition verrà considerato il caso in cui ctx è null e quello in cui è un'istanza valida (ad esempio un'istanza di un oggetto che tiene traccia dell'esito dell'ultima lettura) : {null, istanza valida}
  Analisi Boundary-Value: {null, istanza che tiene traccia dell'handler del ledger da cui si stanno leggendo le entry }

Test-suite minimale: {firstEntry = -1; lastEntry= 0; cb = null; ctx = null}; {firstEntry = 0; LastEntry = -1;cb = istanza che implementa l'interfaccia ReadCallback; ctx = istanza che tiene traccia dell'esito dell'ultima scrittura

};{firstEntry = 1; LastEntry = 1; cb = istanza che implementa l'interfaccia ReadCallback, ctx = istanza che tiene traccia dell'esito dell'ultima lettura }

CATEGORY PARTITION- OPENJPA

OPENJPA-ProxyManagerImpl

Classe che funge da proxy per quegli oggetti che rappresentano lo stato delle classi su cui viene eseguita la persistenza.

Inoltre il proxy effettua una copia di questi oggetti quando viene effettuato un rollback di una transazione, in modo da ripristinarne lo stato. Alcuni dei metodi necessari per farlo sono:

ProxyManagerImpl-*Object copyArray(Object orig)*

Ritorna un array dello stesso tipo di quello dato in input e contenente gli stessi elementi.

Per il category partition, verrà considerata un'istanza nulla, un array vuoto, un array di lunghezza maggiore di 0 ed un oggetto che non sia un array (ad esempio una String).

- *Object orig*: Object che rappresenta l'Array da copiare. Le classi di equivalenza individuate sono: {orig = null; orig. length = 0; orig. length >0; orig = String}
- *Analisi Boundary-Value*: {null, orig. length = 0; orig. length = 1; orig = "testString"}

Test suite minimale: {null}, {orig. length = 0}, {orig. length = 1}, {orig = "testString"}

ProxyManagerImpl- *Map copyMap(Map orig)*

Ritorna un oggetto Map dello stesso tipo e contenente le stesse coppie chiave/valore di quello in input

Per il category partition, verrà considerata un'istanza nulla, un'istanza di Map che non contiene nessuna coppia chiave/valore e un'istanza di Map che contiene almeno una coppia chiave valore.

- Map orig: Object che rappresenta il Map da copiare. Le classi di equivalenza individuate sono: {orig = null; orig.size = 0; orig.size>0}
- *Analisi Boundary-Value*: {null, orig.size = 0; orig.size = 1}

Test suit minimale: {null}, {orig.size = 0}, {orig.size = 1}

ProxyManagerImpl- *Object copyCustom(Object orig)*

Ritorna una copia dell'oggetto passato in input. Viene utilizzato quando serve un proxy per un oggetto che non sia Collection, List, Map, Queue, Date o Calendar. Secondo la documentazione, l'oggetto in input deve avere le seguenti proprietà:

- Oggetti che rappresentano delle date devono avere un costruttore pubblico senza argomenti, oppure un costruttore che abbia come singolo parametro un long che rappresenta il tempo corrente
- Oggetti "container" devono avere un costruttore pubblico senza argomenti, oppure un costruttore che abbia come singolo parametro un Comparator
- Tutti gli altri tipi di oggetto devono avere un costruttore pubblico senza argomenti, oppure copy constructor.

Per il category partition, verrà considerata:

- un'istanza nulla, orig = null.
- un'istanza di un oggetto che rappresenta delle date con costruttore pubblico senza argomenti, origDate().
- un'istanza di un oggetto che rappresenta delle date avente costruttore con argomento long, origDate(long currentTime).
- un'istanza di un oggetto container con costruttore pubblico senza argomenti origContainer()
- un'istanza di un oggetto container avente costruttore con argomento Comparator, origContainer(Comparator comp)
- un'istanza di un oggetto con costruttore pubblico senza argomenti, orig()
- un'istanza d un oggetto avente un costruttore copy, orig(orig)

Le classi di equivalenza individuate sono: {orig = null; origDate(); origDate(long currentTime); origContainer(); origContainer(Comparator comp); orig(); orig(orig)}

Analisi boundary-value: {orig = null; origDate(); origDate(long currentTime); origContainer(); origContainer(Comparator comp); orig(); orig(orig)}

Test suit minimale: {orig = null}, {origDate()}, {origDate(long currentTime)}, {origContainer()}, {origContainer(Comparator comp)}, {orig()}, {orig(orig)}

OPENJPA-ImplHelper

Classe che implementa alcuni metodi di help per il backend di OpenJPA

ImpleHelper- BitSet getUpdateFields(OpenJPAStateManager sm)

Metodo che preso in input un OpenJPAStateManager (oggetto responsabile dello stato di ogni istanza persistence-capable), ne ritorna I campi che devono essere aggiornati (se ce ne sono).

- *OpenJPAStateManager sm*: Rappresenta lo state manager di cui si vogliono controllare i campi non aggiornati. Si considera il caso in cui lo state manager è null, quello in cui è inizializzato con uno state corretto (ovvero uno di quelli definiti nella classe PCState) e non dirty, quello in cui è inizializzato con

uno state corretto (ovvero uno di quelli definiti nella classe PCState) e dirty, quello in cui è inizializzato con uno state = null.
Classi di equivalenza: {null, sm.initialize(PCSTATE not dirty), sm.initialize(PCSTATE dirty), sm.initialize(null)}

Per la test suite, si è deciso di prendere PCNEW come stato non dirty e PCDIRTY come stato dirty,

Test suite minimale: {null, sm.initialize(PCNEW), sm.initialize(PCDIRTY), sm.initialize(null)}

ImpleHelper - boolean isManagedType(OpenJPAConfiguration conf, Class type)

Metodo che preso in input un OPENJPAConfiguration e l'istanza di una classe, ritorna True se OPENJPA può gestire istanze di quella classe (False altrimenti)

- *OpenJPAConfiguration conf*: Definisce la configurazione di OPENJPA. Si considera il caso il cui è null ed il caso in cui la configurazione è correttamente istanziata. Classi di equivalenza: {null, configurazione correttamente istanziata}.
- *Class type*: Classe di cui si vuole sapere se può essere gestita da OPENJPA. Si considera il caso in cui è null, il caso in cui la classe non può essere gestita da OPENJPA e quello in cui può essere gestita. Classi di equivalenza: {null, classe gestibile da OPENJPA, classe non gestibile da OPENJPA}

Test suite minimale: {conf = null, type = null}, {conf = configurazione correttamente istanziata, type = classe che implementa l'interfaccia PersistenceCapable}, {conf = configurazione correttamente istanziata, type = classe che non implementa l'interfaccia PersistenceCapable}

ImpleHelper - boolean isAssignable(Class from, Class to)

Metodo che prende in input una class *from* ed una classe *to* e ritorna True se la classe *to* può essere assegnata alla classe *from.* Il metodo utilizza una cache per non avere troppo overhead.

- *Class from*: Si considera il caso in cui *from* è null, e quello in cui è diverso da null. Classi di equivalenza: {null, not null}.
- *Class to*: Si considera il caso in cui from è null, il caso in cui to non può essere assegnata alla classe *from*, il caso in cui *to* è dello stesso tipo di *from* ed il caso in cui *to* è una specializzazione di *from*. Classi di equivalenza: {null , *to*.type = from.type, *to* extends *from*, !assignableTo(*from*) }.

Per la test suite, si è deciso di definire una classe Foo per i casi in cui *from* appartiene alla partizione {not null}.

Di conseguenza quando *from* è un'istanza di Foo e si vuole testare il caso in cui *to* è una specializzazione di *from*, *to* può essere un'istanza di una classe che specializza Foo. Invece quando si vuole testare il caso in cui *to* non è assignabile a *from*, *to* può essere un'istanza della classe Integer.

Test suite minimale: {null, null}, {from.type = Foo, to.type = Foo},{from.type = Foo, to.type = extendedFoo}, {from.type = Foo, to.type = Integer}.

IMPLEMENTAZIONE TEST

Per l'implementazione è stato utilizzato il framework JUnit. Dove necessario è stato utilizzato il runner Parametrized, un runner che permette di ripetere lo stesso test variando solamente alcuni parametri (come ad esempio i parametri di input del metodo da testare).

BOOKKEEPER-LedgerHandle

LedgerHandle- void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx)

Per poter testare il metodo, era necessario avere un'istanza di Bookkeeper ed una di Zookkeeper up and running. Per fare questo, si è realizzata la classe BookkeeperSetup a partire dalla classe BookKeeperClusterTest e la classe ZookeeperSetup a partire dalla classe ZookeeperUtil, già presenti nel progetto originale.

Queste classi sono state modificate in modo tale da creare un BookieServer su cui verrà scritto il contenuto del ledger, un ZookeeperClient, un ZookeeperServer necessari per salvare i metadati del ledger ed un'instanza BookKeeper (che sarà responsabile della creazione del LedgerHandle su cui verrà effettuato il test).

Per poter testare il metodo sono state anche create le classi CustomReadCallback,(che implementa l'interfaccia ReadCallback e che ha il compito di effettuare la lettura delle entry) e CustomContextObject (che tiene traccia di quale ledgerHandle ha gestito le operazioni su quel ledger).

Il test è stato implementato col metodo *testAsyncReadEntries* che:

1. crea un ledger e aggiunge delle entry
2. esegue la asyncReadEntries dal ledger utilizzando i parametri desiderati
3. attende che il metodo finisca la lettura
4. verifica se il risultato ottenuto è quello aspettato

Il test viene considerato superato se tutte le seguenti condizioni sono soddisfatte:

- il codice di ritorno della lettura è quello aspettato
- il customContextObject ha rilevato correttamente il ledgerHandle
- se il codice di ritorno è Code.OK (cioè se l'operazione di lettura è andata a buon fine), i dati letti da CustomReadCallback sono quelli aspettati

Ad ogni iterazione, i parametri con cui verranno eseguiti i test sono: {firstEntry, lastEntry, readCallback, contextObject, expectedResult}, che rappresentano rispettivamente la prima entry da cui leggere, l'ultima entry da cui leggere, la readCallback con cui effettuare la lettura, il context object ed il risultato atteso.

BOOKKEEPER-LedgerEntriesImpl

LedgerEntriesImpl- LedgerEntry getEntry(long entryId)

Per poter testare il metodo, è stata creata un'istanza di LedgerEntriesImpl contenente alcune Entries che dovranno essere lette.

Il test è stato implementato dal metodo *testGetEntry* che legge la entry specificata dal parametro entryId e verifica se il LedgerId, l'entryId e la lunghezza dell'entry letta sono quelle aspettate.

LedgerEntriesImpl- LedgerEntriesImpl create(List<LedgerEntry> entries)

Il test è stato implementato dal metodo *testCreate*() che prova a creare un'istanza di LedgerEntriesImpl con diversi parametri entries, verificando che la size della lista di entry, la classe di appartenenza di ogni entry ed il loro contenuto siano quelli attesi.

OPENJPA-ProxyManagerImpl

ProxyManagerImpl-*Object copyArray(Object orig)*

Il test è stato implementato dal metodo *testCopyArrays* che prende in input un array di Integer da far copiare al proxyManager, verificando che il risultato del metodo ritorni un array dello stesso tipo, della stessa grandezza e dello stesso contenuto dell'array di input.

ProxyManagerImpl- *Map copyMap(Map orig)*

Il test è stato implementato dal metodo *testCopyMaps* che prende in input una HashMap, verificando che il risultato del metodo ritorni un Map dello stesso tipo, della stessa grandezza e dello stesso contenuto dell'array di input.

ProxyManagerImpl- *Object copyCustom(Object orig)*

Il test è composto da vari metodi:

- *testCopyCustomDate*: effettua una chiamata a copyCustom dando in input un oggetto di tipo Date istanziato con il costruttore di default e verifica che il risultato ritorni un oggetto che sia dello stesso tipo e che rappresenti la stessa data dell'oggetto Date dato input. Successivamente effettua una nuova chiamata a *copyCustom* dando in input un ogetto di tipo CustomDate, definito appositamente per il test, istanziato con un costruttore che prende come input un long (come definito in fase di category partition). Anche per questo input, verifica se il risultato ritorna un oggetto che sia dello stesso tipo e che rappresenti la stessa data dell'oggetto CustomDate.
- *testCopyCustomMap*: serve per testare il caso in cui viene dato in input un oggetto di tipo container. Effettua una chiamata a *copyCustom* dando in input un oggetto di tipo TreeMap istanziato con il costruttore standard e verifica se il risultato ritorna un oggetto che sia dello stesso tipo, della stessa size e con lo stesso contenuto del TreeMap in input. Successivamente viene ripetuto lo stesso test, ma dando in input un oggetto di tipo TreeMap istanziato con un costruttore che prende in input un CustomComparator (un oggetto di tipo Comparator che definisce come effettuare l'ordinamento all'interno del TreeMap) e verificando che il risultato sia un oggetto di tipo TreeMap , della stessa size e con lo stesso contenuto del TreeMap in input.

- *testCopyCustomObject*: serve per testare il caso in cui viene dato in input un oggetto di qualsiasi tipo, purchè abbia un costruttore pubblico senza argomenti oppure un copy constructor. Per questo test è stato definito un oggetto CustomObject che ha un costruttore pubblico senza argomenti ,un copy constructor (un costruttore che crea un oggetto utilizzando un altro oggetto della stessa classe), un attributo int ed uno String con i relativi setters/getters. Il metodo effettua una chiamata a *copyCustom* con input un CustomObject istanziato col costruttore standard e verifica che il risultato sia un oggetto dello stesso tipo e con gli stessi attributi del CustomObject di input. Successivamente fa la stessa verifica, ma dando in input un oggetto CustomObject istanziato col copy constructor e verificando che il risultato sia un oggetto dello stesso tipo e con gli stessi attributi del CustomObject di input.

OPENJPA-ImplHelper

ImpleHelper- BitSet getUpdateFields(OpenJPAStateManager sm)

Il test è stato implementato dal metodo *testgetUpdateFields* che effettua chiamate al metodo *getUpdateFields* dandogli in input un mock dell'interfaccia OpenJPAStateManager e verificando che il BitSet ritornato (che rappresenta quali sono i field che devono essere aggiornati) sia quello atteso. Il mock viene utilizzato per simulare il comportamento di alcuni metodi dell'interfaccia (come ad esempio getPCState, che ritorna lo stato attuale dell'istanza gestita dallo state manager) ed il test viene fatto girare col runner MockitoJunitRunner.

ImpleHelper - boolean isManagedType(OpenJPAConfiguration conf, Class type)

Il test è stato implementato dal metodo *testisManagedType* che effettua una chiamata ad *isManagedType* passando in input una OPENJPAConfiguration ed una Class, verificando che il boolean ritornato sia quello atteso .

ImpleHelper - boolean isAssignable(Class from, Class to)

Il test è stato implementato dal metodo *testisAssignable* che effettua una chiamata ad *isAssignable* passando in input due classi e verificando che il risultato sia False quando la classe *to* non è assegnabile alla classe *from* e che sia True quando invece è assegnabile

ADEGUATEZZA DEI TEST

Per misurare l'adeguatezza dei test, ovvero per stabilire se i test sviluppati permettono di testare a sufficienza il SUT, sono stati raccolti dei dati relativi alle metriche di branch coverage e line coverage, utilizzando il plugin Jacoco.

Per utilizzarlo, è stato necessario creare un modulo jacoco-reports che ha la responsabilità di creare un report riguardante la copertura dei test secondo criteri di control flow. Tale plugin è stato poi integrato con SonarCloud, per permettere l'analisi delle metriche anche in ambiente di produzione.

Dato che su sonarCloud è possibile ottenere le metriche solo relative all'intera classe, portando quindi a risultati non veritieri quando vengono testati solo alcuni metodi della classe stessa, verrà fatto riferimento anche ai report di Jacoco fatti in locale (che permettono invece di avere anche le metriche relative al singolo metodo).

BOOKKEEPER-LedgerHandle

LedgerHandle- void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx)

In Figura 1 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

La classe aveva Condition Coverage del 14.1% e Line coverage del 63% mentre, guardando il report di Jacoco fatto in locale, il metodo aveva Condition coverage del 66% e Line coverge del 63%.

Sono stati aggiunti i seguenti casi di test:

- firstEntry = -1, lastEntry = -2, per testare il caso in cui firstEntry è minore di 0 e maggiore di lastEntry e coprire una delle condizione di riga 680
- firstEntry = 0, lastEntry = 6, per testare il caso in cui lastEntry è maggiore di lastAddConfirmed e coprire la condizione di riga 687

In Figura 2 viene mostrata la situazione dopo l'implementazione dei test. Adesso la classe ha condition coverage del 17% e line coverage del 24%, mentre il metodo ha condition coverage e line coverage del 100%

BOOKKEEPER-LedgerEntriesImpl

LedgerEntriesImpl- LedgerEntry getEntry(long entryId)

In Figura 3 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

La classe aveva Condition Coverage del 75% e Line coverage del 100% e stessa cosa per il metodo.

È stato aggiunto un caso di test in cui viene letta una entry con ID superiore all'ultimo entryID letto, in modo da coprire la condizione di riga 86

In Figura 4 viene mostrata la situazione dopo l'implementazione dei test. Adesso la classe ha condition coverage del 100% e line coverage del 100% e stessa cosa per il metodo.

LedgerEntriesImpl- LedgerEntriesImpl create(List<LedgerEntry> entries)

In Figura 3 viene mostrata la situazione dopo aver implementato i test nella sezione precedente. Dato che il metodo ha già condition coverage e line coverage del 100%, non verranno aggiunti altri test

OPENJPA-ProxyManagerImpl

Prima dell'aggiunta dei nuovi casi di test, la classe aveva Condition Coverage del 28.5% e Line coverage del 36.3%

ProxyManagerImpl-*Object copyArray(Object orig)*

In Figura 5 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva Condition Coverage del 100% e Line coverage del 65% .

È stato aggiunto un caso di test in cui viene data in input una stringa vuota, in modo da poter arrivare alle istruzioni tra riga 190 e 192.

In Figura 6 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha condition coverage del 100% e line coverage del 100%.

ProxyManagerImpl- *Map copyMap(Map orig)*

In Figura 7 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva Condition Coverage del 75% e Line coverage del 73% .

È stato aggiunto un caso di test in cui viene data in input una ProxyMap per coprire la condizione di riga 220.

In Figura 8 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha condition coverage del 100% e line coverage del 100%.

ProxyManagerImpl- *Object copyCustom(Object orig)*

In Figura 9 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva Condition Coverage del 75% e Line coverage del 73% .

Sono stati aggiunti i seguenti casi di test:

- test con input un oggetto di tipo Proxy per coprire la condizione di riga 279
- test con input una oggetto di tipo Collection per coprire la condizione di riga 283
- test con input un oggetto di tipo Calendar per coprire la condizione di riga 289
- test con input una Stringa per coprire la condizione di riga 292 (ovvero passare un oggetto per il quale non è possibile creare un Proxy)

In Figura 10 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha condition coverage del 96% e line coverage del 93%.

Dopo l'aggiunte dei nuovi casi di test, la classe ha Condition Coverage del 37% e Line coverage del 40%

OPENJPA-ImplHelper

Prima dell'aggiunta dei nuovi casi di test, la classe aveva Condition Coverage del 27.8% e Line coverage del 28.3%

ImpleHelper- BitSet getUpdateFields(OpenJPAStateManager sm)

In Figura 11 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva Condition Coverage del 64% e Line coverage del 100% .

Sono stati aggiunti i seguenti casi di test:

- test con isFlushed = true, isFlushedDirty = false, e PCState = Dirty per testare la condizione a riga 184
- test con isFlushed = false, isFlushedDirty = true e PCState = Dirty per testare la condizione a riga 187

In Figura 12 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha condition coverage del 85% e line coverage del 100%.

ImpleHelper - boolean isManagedType(OpenJPAConfiguration conf, Class type)

In Figura 13 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva Condition Coverage del 74% e Line coverage del 100% .

È stato aggiunto un caso di test in cui viene data in input una Integer.class ed una configurazione per la quale RunTimeUnenhancedClasses non è supportato per testare la condizione a riga 225.

In Figura 14 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha condition coverage del 80% e line coverage del 100%.

ImpleHelper - boolean isAssignable(Class from, Class to)

In Figura 15 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva Condition Coverage del 75% e Line coverage del 100% .

Sono stati aggiunti i seguenti casi di test:

- test in cui la classe *from* è null e la classe *to* è Foo per testare una delle due condizioni di riga 51.
- test in cui la classe *from* è Foo e la classe *to* è null per testare l'altra condizione di riga 51.

In Figura 16 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha condition coverage del 87% e line coverage del 100%.

Dopo l'aggiunte dei nuovi casi di test, la classe ha Condition Coverage del 40% e Line coverage del 33%

MUTAZIONI DEI TEST

In questa fase, verranno introdotte artificialmente delle modifiche al SUT in modo da alterarne il comportamento. L'obiettivo è fare in modo che i casi di test siano in grado di rilevarle.

Per farlo, verrà utilizzato il framework PIT Mutation Testing, indicando nel pom dei moduli maven contenenti i test quali sono le classi target e quali sono le relative classi di test.

I report sono presenti nelle cartelle "OpenJPA_KernelPitReport" e "BookkeeperServer_pit-report".

BOOKKEEPER-LedgerHandle

LedgerHandle- void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx)

In Figura 17 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva mutation score del 33%

È stato aggiunto il caso di test in cui viene dato in input lastEntry = 4 per fare in modo che lastEntry sia uguale a lastAddConfirmed e uccidere il mutante sopravvissuto a riga 687.

Non è stato possibile uccidere gli altri mutanti per il quale l'esecuzione va in timeout.

In Figura 18 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha mutation score del 66%

BOOKKEEPER-LedgerEntriesImpl

LedgerEntriesImpl- LedgerEntry getEntry(long entryId)

In Figura 19 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva mutation score del 90%

.È stato aggiunto il caso di test in cui viene creato un LedgerEntriesImpl con firstEntry = 1 e viene utilizzato il metodo getEntry con entryId pari a 2. In questo modo è possibile uccidere il mutante sopravvissuto a riga 90 (nel quale veniva sostituita la differenza con una somma. Nei casi implementati, firstId era sempre 0 e dunque non si notava la differenza).

In Figura 20 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha mutation score del 100%

LedgerEntriesImpl- LedgerEntriesImpl create(List<LedgerEntry> entries)

I test considerati in precedenza per questo metodo, hanno ucciso tutti i mutanti generati come visibile dalla figura 19.

Per questo motivo, non sono stati implementati ulteriori test.

OPENJPA-ProxyManagerImpl

I test considerati in precedenza per i metodi di questa classe, hanno ucciso tutti i mutanti generati come visibile dalla figure da 21 a 23.

Per questo motivo, non sono stati implementati ulteriori test.

OPENJPA-ImplHelper

ImpleHelper- BitSet getUpdateFields(OpenJPAStateManager sm)

In Figura 24 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva mutation score del 50%

Sono stati aggiunti i seguenti casi di test:

test in cui sia isFlushed sia isFlushed dirty sono posti a false e PCState = Dirty per uccidere il mutante di riga 184 e 185 (nei quali veniva negata la prima condizione).

test in cui isFlushed è false, isFlushedDirty è true, flusehd è vuoto con PCState = Dirty per uccidere i mutanti alle righe 191 e 192 (per i quali non c'era coverage)

test in cui flushed è true, isFlushedDirty è false e PCState = New per uccidere il mutante di riga 185 (nel quale la seconda condizione era negata)

In Figura 25 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha mutation score del 100%

ImpleHelper - boolean isManagedType(OpenJPAConfiguration conf, Class type)

In Figura 26 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva mutation score del 16%

È stato aggiunto un caso di test con type = null e conf = conf2 (configurazione con RuntimeUnhenhancedClasses non supportato) ed un caso di test con type = notRegisteredPC (istanza che implementa l'interfaccia PersisteCapable, ma non registrata nel PCRegistry) e configurazione con RuntimeUnhenhancedClasses supportato , in modo da uccidere i 2 mutanti presenti a riga 223

Non è stato possibile eliminare i mutanti rimanenti.

In Figura 27 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha mutation score del 57%

ImpleHelper - boolean isAssignable(Class from, Class to)

In Figura 28 viene mostrata la situazione prima dell'aggiunta dei nuovi casi di test .

Il metodo aveva mutation score dell '85%

È stato aggiunto un caso di test con class *from* = Foo e classe *to* = String, in modo da uccidere il mutante presente a riga 256

In Figura 29 viene mostrata la situazione dopo l'implementazione dei test. Adesso il metodo ha mutation score del 100%

```
678.    public void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx) {
679.        // Little sanity check
680.        if (firstEntry < 0 || firstEntry > lastEntry) {
681.            LOG.error("IncorrectParameterException on ledgerId:{} firstEntry:{} lastEntry:{}",
682.                    ledgerId, firstEntry, lastEntry);
683.            cb.readComplete(BKException.Code.IncorrectParameterException, this, null, ctx);
684.            return;
685.        }
686.
687.        if (lastEntry > lastAddConfirmed) {
688.            LOG.error("ReadException on ledgerId:{} firstEntry:{} lastEntry:{}",
689.                    ledgerId, firstEntry, lastEntry);
690.            cb.readComplete(BKException.Code.ReadException, this, null, ctx);
691.            return;
692.        }
693.        asyncReadEntriesInternal(firstEntry, lastEntry, cb, ctx, false);
694.    }
```

Figura 1

```
678.    public void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx) {
679.        // Little sanity check
680.        if (firstEntry < 0 || firstEntry > lastEntry) {
681.            LOG.error("IncorrectParameterException on ledgerId:{} firstEntry:{} lastEntry:{}",
682.                    ledgerId, firstEntry, lastEntry);
683.            cb.readComplete(BKException.Code.IncorrectParameterException, this, null, ctx);
684.            return;
685.        }
686.
687.        if (lastEntry > lastAddConfirmed) {
688.            LOG.error("ReadException on ledgerId:{} firstEntry:{} lastEntry:{}",
689.                    ledgerId, firstEntry, lastEntry);
690.            cb.readComplete(BKException.Code.ReadException, this, null, ctx);
691.            return;
692.        }
693.        asyncReadEntriesInternal(firstEntry, lastEntry, cb, ctx, false);
694.    }
```

Figura 2

```
71.     public static LedgerEntriesImpl create(List<LedgerEntry> entries) {
72.         checkArgument(!entries.isEmpty(), "entries for create should not be empty.");
73.         LedgerEntriesImpl ledgerEntries = RECYCLER.get();
74.         ledgerEntries.entries = entries;
75.         return ledgerEntries;
76.     }
77.
78.     /**
79.      * {@inheritDoc}
80.      */
81.     @Override
82.     public LedgerEntry getEntry(long entryId) {
83.         checkNotNull(entries, "entries has been recycled");
84.         long firstId = entries.get(0).getEntryId();
85.         long lastId = entries.get(entries.size() - 1).getEntryId();
86.         if (entryId < firstId || entryId > lastId) {
87.             throw new IndexOutOfBoundsException("required index: " + entryId
88.                     + " is out of bounds: [ " + firstId + ", " + lastId + " ].");
89.         }
90.         return entries.get((int) (entryId - firstId));
91.     }
```

Figura 3

```
71.     public static LedgerEntriesImpl create(List<LedgerEntry> entries) {
72.         checkArgument(!entries.isEmpty(), "entries for create should not be empty.");
73.         LedgerEntriesImpl ledgerEntries = RECYCLER.get();
74.         ledgerEntries.entries = entries;
75.         return ledgerEntries;
76.     }
77.
78.     /**
79.      * {@inheritDoc}
80.      */
81.     @Override
82.     public LedgerEntry getEntry(long entryId) {
83.         checkNotNull(entries, "entries has been recycled");
84.         long firstId = entries.get(0).getEntryId();
85.         long lastId = entries.get(entries.size() - 1).getEntryId();
86.         if (entryId < firstId || entryId > lastId) {
87.             throw new IndexOutOfBoundsException("required index: " + entryId
88.                 + " is out of bounds: [ " + firstId + ", " + lastId + " ].");
89.         }
90.         return entries.get((int) (entryId - firstId));
91.     }
```

Figura 4

```
178.     @Override
179.     public Object copyArray(Object orig) {
180.         if (orig == null)
181.             return null;
182.
183.         try {
184.             int length = Array.getLength(orig);
185.             Object array = Array.newInstance(orig.getClass().
186.                 getComponentType(), length);
187.
188.             System.arraycopy(orig, 0, array, 0, length);
189.             return array;
190.         } catch (Exception e) {
191.             throw new UnsupportedException(_loc.get("bad-array",
192.                 e.getMessage()), e);
193.         }
194.     }
```

Figura 5

```
178.     @Override
179.     public Object copyArray(Object orig) {
180.         if (orig == null)
181.             return null;
182.
183.         try {
184.             int length = Array.getLength(orig);
185.             Object array = Array.newInstance(orig.getClass().
186.                 getComponentType(), length);
187.
188.             System.arraycopy(orig, 0, array, 0, length);
189.             return array;
190.         } catch (Exception e) {
191.             throw new UnsupportedException(_loc.get("bad-array",
192.                 e.getMessage()), e);
193.         }
194.     }
```

Figura 6

```
216.    @Override
217.    public Map copyMap(Map orig) {
218.        if (orig == null)
219.            return null;
220.        if (orig instanceof Proxy)
221.            return (Map) ((Proxy) orig).copy(orig);
222.
223.        ProxyMap proxy = getFactoryProxyMap(orig.getClass());
224.        return (Map) proxy.copy(orig);
225.    }
```

Figura 7

```
216.    @Override
217.    public Map copyMap(Map orig) {
218.        if (orig == null)
219.            return null;
220.        if (orig instanceof Proxy)
221.            return (Map) ((Proxy) orig).copy(orig);
222.
223.        ProxyMap proxy = getFactoryProxyMap(orig.getClass());
224.        return (Map) proxy.copy(orig);
225.    }
```

Figura 8

```
275.    @Override
276.    public Object copyCustom(Object orig) {
277.        if (orig == null)
278.            return null;
279.        if (orig instanceof Proxy)
280.            return ((Proxy) orig).copy(orig);
281.        if (ImplHelper.isManageable(orig))
282.            return null;
283.        if (orig instanceof Collection)
284.            return copyCollection((Collection) orig);
285.        if (orig instanceof Map)
286.            return copyMap((Map) orig);
287.        if (orig instanceof Date)
288.            return copyDate((Date) orig);
289.        if (orig instanceof Calendar)
290.            return copyCalendar((Calendar) orig);
291.        ProxyBean proxy = getFactoryProxyBean(orig);
292.        return (proxy == null) ? null : proxy.copy(orig);
293.    }
```

Figura 9

```
275.    @Override
276.    public Object copyCustom(Object orig) {
277.        if (orig == null)
278.            return null;
279.        if (orig instanceof Proxy)
280.            return ((Proxy) orig).copy(orig);
281.        if (ImplHelper.isManageable(orig))
282.            return null;
283.        if (orig instanceof Collection)
284.            return copyCollection((Collection) orig);
285.        if (orig instanceof Map)
286.            return copyMap((Map) orig);
287.        if (orig instanceof Date)
288.            return copyDate((Date) orig);
289.        if (orig instanceof Calendar)
290.            return copyCalendar((Calendar) orig);
291.        ProxyBean proxy = getFactoryProxyBean(orig);
292.        return (proxy == null) ? null : proxy.copy(orig);
293.    }
```

Figura 10

```
182.    public static BitSet getUpdateFields(OpenJPAStateManager sm) {
183.        if ((sm.getPCState() == PCState.PDIRTY
184.            && (!sm.isFlushed() || sm.isFlushedDirty()))
185.            || (sm.getPCState() == PCState.PNEW && sm.isFlushedDirty())) {
186.            BitSet dirty = sm.getDirty();
187.            if (sm.isFlushed()) {
188.                dirty = (BitSet) dirty.clone();
189.                dirty.andNot(sm.getFlushed());
190.            }
191.            if (dirty.length() > 0)
192.                return dirty;
193.        }
194.        return null;
195.    }
```

Figura 11

```
182.    public static BitSet getUpdateFields(OpenJPAStateManager sm) {
183.        if ((sm.getPCState() == PCState.PDIRTY
184.            && (!sm.isFlushed() || sm.isFlushedDirty()))
185.            || (sm.getPCState() == PCState.PNEW && sm.isFlushedDirty())) {
186.            BitSet dirty = sm.getDirty();
187.            if (sm.isFlushed()) {
188.                dirty = (BitSet) dirty.clone();
189.                dirty.andNot(sm.getFlushed());
190.            }
191.            if (dirty.length() > 0)
192.                return dirty;
193.        }
194.        return null;
195.    }
196.
```

Figura 12

```
222.    public static boolean isManagedType(OpenJPAConfiguration conf, Class type) {
223.        return (PersistenceCapable.class.isAssignableFrom(type)
224.                || (type != null
225.                    && (conf == null || conf.getRuntimeUnenhancedClassesConstant()
226.                        == RuntimeUnenhancedClassesModes.SUPPORTED)
227.                    && PCRegistry.isRegistered(type)));
228.    }
229.
```

Figura 13

```
222.    public static boolean isManagedType(OpenJPAConfiguration conf, Class type) {
223.        return (PersistenceCapable.class.isAssignableFrom(type)
224.                || (type != null
225.                    && (conf == null || conf.getRuntimeUnenhancedClassesConstant()
226.                        == RuntimeUnenhancedClassesModes.SUPPORTED)
227.                    && PCRegistry.isRegistered(type)));
228.    }
229.
```

Figura 14

```
250.    public static boolean isAssignable(Class from, Class to) {
251.        if (from == null || to == null)
252.            return false;
253.
254.        Boolean isAssignable = null;
255.        Map assignableTo = (Map) _assignableTypes.get(from);
256.        if (assignableTo == null) { // "to" cache doesn't exist, so create it...
257.            assignableTo = new ConcurrentReferenceHashMap(ReferenceStrength.WEAK,
258.                ReferenceStrength.HARD);
259.            _assignableTypes.put(from, assignableTo);
260.        } else { // "to" cache exists...
261.            isAssignable = (Boolean) assignableTo.get(to);
262.        }
263.
264.        if (isAssignable == null) {// we don't have a record of this pair...
265.            isAssignable = Boolean.valueOf(from.isAssignableFrom(to));
266.            assignableTo.put(to, isAssignable);
267.        }
268.
269.        return isAssignable.booleanValue();
270.    }
```

Figura 15

```
250.    public static boolean isAssignable(Class from, Class to) {
251.        if (from == null || to == null)
252.            return false;
253.
254.        Boolean isAssignable = null;
255.        Map assignableTo = (Map) _assignableTypes.get(from);
256.        if (assignableTo == null) { // "to" cache doesn't exist, so create it...
257.            assignableTo = new ConcurrentReferenceHashMap(ReferenceStrength.WEAK,
258.                ReferenceStrength.HARD);
259.            _assignableTypes.put(from, assignableTo);
260.        } else { // "to" cache exists...
261.            isAssignable = (Boolean) assignableTo.get(to);
262.        }
263.
264.        if (isAssignable == null) {// we don't have a record of this pair...
265.            isAssignable = Boolean.valueOf(from.isAssignableFrom(to));
266.            assignableTo.put(to, isAssignable);
267.        }
268.
269.        return isAssignable.booleanValue();
270.    }
```

Figura 16

```
678      public void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx) {
679          // Little sanity check
680  4       if (firstEntry < 0 || firstEntry > lastEntry) {
681              LOG.error("IncorrectParameterException on ledgerId:{} firstEntry:{} lastEntry:{}",
682                      ledgerId, firstEntry, lastEntry);
683  1           cb.readComplete(BKException.Code.IncorrectParameterException, this, null, ctx);
684              return;
685          }
686
687  2       if (lastEntry > lastAddConfirmed) {
688              LOG.error("ReadException on ledgerId:{} firstEntry:{} lastEntry:{}",
689                      ledgerId, firstEntry, lastEntry);
690  1           cb.readComplete(BKException.Code.ReadException, this, null, ctx);
691              return;
692          }
693  1       asyncReadEntriesInternal(firstEntry, lastEntry, cb, ctx, false);
694      }
```

Figura 17

```
678      public void asyncReadEntries(long firstEntry, long lastEntry, ReadCallback cb, Object ctx) {
679          // Little sanity check
680  4       if (firstEntry < 0 || firstEntry > lastEntry) {
681              LOG.error("IncorrectParameterException on ledgerId:{} firstEntry:{} lastEntry:{}",
682                      ledgerId, firstEntry, lastEntry);
683  1           cb.readComplete(BKException.Code.IncorrectParameterException, this, null, ctx);
684              return;
685          }
686
687  2       if (lastEntry > lastAddConfirmed) {
688              LOG.error("ReadException on ledgerId:{} firstEntry:{} lastEntry:{}",
689                      ledgerId, firstEntry, lastEntry);
690  1           cb.readComplete(BKException.Code.ReadException, this, null, ctx);
691              return;
692          }
693  1       asyncReadEntriesInternal(firstEntry, lastEntry, cb, ctx, false);
694      }
695
```

Figura 18

```
71      public static LedgerEntriesImpl create(List<LedgerEntry> entries) {
72  2       checkArgument(!entries.isEmpty(), "entries for create should not be empty.");
73          LedgerEntriesImpl ledgerEntries = RECYCLER.get();
74          ledgerEntries.entries = entries;
75  1       return ledgerEntries;
76      }
77
78      /**
79       * {@inheritDoc}
80       */
81      @Override
82      public LedgerEntry getEntry(long entryId) {
83          checkNotNull(entries, "entries has been recycled");
84          long firstId = entries.get(0).getEntryId();
85  1       long lastId = entries.get(entries.size() - 1).getEntryId();
86  4       if (entryId < firstId || entryId > lastId) {
87              throw new IndexOutOfBoundsException("required index: " + entryId
88                  + " is out of bounds: [ " + firstId + ", " + lastId + " ].");
89          }
90  2       return entries.get((int) (entryId - firstId));
91      }
```

Figura 19

```
71      public static LedgerEntriesImpl create(List<LedgerEntry> entries) {
72  2       checkArgument(!entries.isEmpty(), "entries for create should not be empty.");
73          LedgerEntriesImpl ledgerEntries = RECYCLER.get();
74          ledgerEntries.entries = entries;
75  1       return ledgerEntries;
76      }
77
78      /**
79       * {@inheritDoc}
80       */
81      @Override
82      public LedgerEntry getEntry(long entryId) {
83          checkNotNull(entries, "entries has been recycled");
84          long firstId = entries.get(0).getEntryId();
85  1       long lastId = entries.get(entries.size() - 1).getEntryId();
86  4       if (entryId < firstId || entryId > lastId) {
87              throw new IndexOutOfBoundsException("required index: " + entryId
88                  + " is out of bounds: [ " + firstId + ", " + lastId + " ].");
89          }
90  2       return entries.get((int) (entryId - firstId));
91      }
92
```

Figura 20

```
217       public Map copyMap(Map orig) {
218   1       if (orig == null)
219               return null;
220   1       if (orig instanceof Proxy)
221   1           return (Map) ((Proxy) orig).copy(orig);
222
223           ProxyMap proxy = getFactoryProxyMap(orig.getClass());
224   1       return (Map) proxy.copy(orig);
225       }
```

Figura 21

```
178     @Override
179     public Object copyArray(Object orig) {
180 1       if (orig == null)
181             return null;
182
183         try {
184             int length = Array.getLength(orig);
185             Object array = Array.newInstance(orig.getClass().
186                 getComponentType(), length);
187
188 1           System.arraycopy(orig, 0, array, 0, length);
189 1           return array;
190         } catch (Exception e) {
191             throw new UnsupportedException(_loc.get("bad-array",
192                 e.getMessage()), e);
193         }
194     }
```

Figura 22

```
275     @Override
276     public Object copyCustom(Object orig) {
277 1       if (orig == null)
278             return null;
279 1       if (orig instanceof Proxy)
280 1           return ((Proxy) orig).copy(orig);
281 1       if (ImplHelper.isManageable(orig))
282             return null;
283 1       if (orig instanceof Collection)
284 1           return copyCollection((Collection) orig);
285 1       if (orig instanceof Map)
286 1           return copyMap((Map) orig);
287 1       if (orig instanceof Date)
288 1           return copyDate((Date) orig);
289 1       if (orig instanceof Calendar)
290 1           return copyCalendar((Calendar) orig);
291         ProxyBean proxy = getFactoryProxyBean(orig);
292 2       return (proxy == null) ? null : proxy.copy(orig);
293     }
```

Figura 23

```
182     public static BitSet getUpdateFields(OpenJPAStateManager sm) {
183 1       if ((sm.getPCState() == PCState.PDIRTY
184 2           && (!sm.isFlushed() || sm.isFlushedDirty()))
185 2           || (sm.getPCState() == PCState.PNEW && sm.isFlushedDirty())) {
186             BitSet dirty = sm.getDirty();
187 1           if (sm.isFlushed()) {
188                 dirty = (BitSet) dirty.clone();
189 1               dirty.andNot(sm.getFlushed());
190             }
191 2           if (dirty.length() > 0)
192 1               return dirty;
193         }
194         return null;
195     }
```

Figura 24

```
182      public static BitSet getUpdateFields(OpenJPAStateManager sm) {
183 1        if ((sm.getPCState() == PCState.PDIRTY
184 2            && (!sm.isFlushed() || sm.isFlushedDirty()))
185 2            || (sm.getPCState() == PCState.PNEW && sm.isFlushedDirty())) {
186              BitSet dirty = sm.getDirty();
187 1            if (sm.isFlushed()) {
188                  dirty = (BitSet) dirty.clone();
189 1                dirty.andNot(sm.getFlushed());
190              }
191 2            if (dirty.length() > 0)
192 1                return dirty;
193          }
194          return null;
195      }
```

Figura 25

```
222      public static boolean isManagedType(OpenJPAConfiguration conf, Class type) {
223 4        return (PersistenceCapable.class.isAssignableFrom(type)
224                || (type != null
225 1                && (conf == null || conf.getRuntimeUnenhancedClassesConstant()
226                        == RuntimeUnenhancedClassesModes.SUPPORTED)
227 1                && PCRegistry.isRegistered(type)));
228      }
```

Figura 26

```
222      public static boolean isManagedType(OpenJPAConfiguration conf, Class type) {
223 3        return (PersistenceCapable.class.isAssignableFrom(type)
224 1            || (type != null
225 2                && (conf == null || conf.getRuntimeUnenhancedClassesConstant()
226                        == RuntimeUnenhancedClassesModes.SUPPORTED)
227 1                && PCRegistry.isRegistered(type)));
228      }
```

Figura 27

```
250     public static boolean isAssignable(Class from, Class to) {
251 2       if (from == null || to == null)
252 1           return false;
253
254         Boolean isAssignable = null;
255         Map assignableTo = (Map) _assignableTypes.get(from);
256 1       if (assignableTo == null) { // "to" cache doesn't exist, so create it...
257             assignableTo = new ConcurrentReferenceHashMap(ReferenceStrength.WEAK,
258                 ReferenceStrength.HARD);
259             _assignableTypes.put(from, assignableTo);
260         } else { // "to" cache exists...
261             isAssignable = (Boolean) assignableTo.get(to);
262         }
263
264 1       if (isAssignable == null) {// we don't have a record of this pair...
265             isAssignable = Boolean.valueOf(from.isAssignableFrom(to));
266             assignableTo.put(to, isAssignable);
267         }
268
269 2       return isAssignable.booleanValue();
270     }
```

Figura 28

```
250     public static boolean isAssignable(Class from, Class to) {
251 2       if (from == null || to == null)
252 1           return false;
253
254         Boolean isAssignable = null;
255         Map assignableTo = (Map) _assignableTypes.get(from);
256 1       if (assignableTo == null) { // "to" cache doesn't exist, so create it...
257             assignableTo = new ConcurrentReferenceHashMap(ReferenceStrength.WEAK,
258                 ReferenceStrength.HARD);
259             _assignableTypes.put(from, assignableTo);
260         } else { // "to" cache exists...
261             isAssignable = (Boolean) assignableTo.get(to);
262         }
263
264 1       if (isAssignable == null) {// we don't have a record of this pair...
265             isAssignable = Boolean.valueOf(from.isAssignableFrom(to));
266             assignableTo.put(to, isAssignable);
267         }
268
269 2       return isAssignable.booleanValue();
270     }
```

Figura 29