Documentazione modifiche apportate al progetto

1 SOMMARIO

2 METODO RIFLESSIVO IN ANALISI

```
1 private HEntry instanceEntryMapAdapter(Object key, Object value) throws Exception {
2         // codice del metodo
3 }
```

2.1 FUNZIONAMENTO

Il metodo `instanceEntryMapAdapter(Object key, Object value)` è il metodo che andremo ad analizzare nei dettagli e a modificare.

Esso è contenuto nel file sorgente `TestMap.java` e viene utilizzato nella Test Suite `TestMap` per istanziare oggetti della classe `protected` e non statica `EntryMapAdapter`, interna alla classe `MapAdapter`.

A tal scopo, senza andare ad alterare l'implementazione della mappa `MapAdapter` e della entry chiave-valore `EntryMapAdapter`, è necessario ricorrere alla programmazione riflessiva, strumento molto potente che permette di eseguire elaborazioni sul codice sorgente del programma stesso.

3 METODO ORIGINARIO, NON FUNZIONANTE CON LE ULTIME VERSIONI DI JDK E JRE

```
1 private HEntry instanceEntryMapAdapter(Object key, Object value) throws Exception {
2         // ottiene la classe privata EntryMapAdapter
3         Class entryClass = MapAdapter.class.getDeclaredClasses()[2];
4         // ottiene il costruttore privato di EntryMapAdapter
5         Constructor constr = entryClass.getDeclaredConstructors()[0];
6         // abilita l'accesso al costruttore privato
7         constr.setAccessible(true);
8         MapAdapter map = new MapAdapter();
9
10        return (HEntry) constr.newInstance(map, key, value);
11 }
```

3.1 AMBIGUITÀ SCOPERTA

Analizzando attentamente la documentazione Java relativa ai metodi `getDeclaredClasses()` e `getDeclaredConstructors()` della classe `Class`, si può notare un dettaglio molto importante: non ci è dato sapere l'ordinamento degli array, rispettivamente di oggetti della classe `Class` e di oggetti della classe `Constructor`, restituiti.

Quando la documentazione non fornisce informazioni di questo tipo, o in generale quando non fornisce certi dettagli, è rischioso prevedere quale potrebbe essere il comportamento.

In particolare, nel metodo `instanceEntryMapAdapter(Object key, Object value)` l'ambiguità relativa all'ordinamento degli oggetti non crea problemi a riga `5` nell'uso dell'array restituito dal metodo `getDeclaredConstructors()`, bensì li crea a riga `3` nell'uso dell'array restituito dal metodo `getDeclaredClasses()`: la mancata gestione di questa criticità causa la propagazione di effetti negativi alle istruzioni che seguono alle righe successive.

Approfondiamo meglio.

3.2 RIPERCUSSIONI DELL'AMBIGUITÀ NEL PROGRAMMA

L'ambiguità relativa all'ordinamento degli oggetti negli array sopracitati, non sorge a riga `5` perché, se la classe desiderata `EntryMapAdapter` è stata selezionata correttamente a riga `3`, l'array di oggetti della classe `Constructor` restituito dal metodo `getDeclaredConstructors()` è sicuramente unidimensionale per la presenza di un unico costruttore nella classe `EntryMapAdapter`: ne consegue che sicuramento in cella `[0]` vi è il costruttore desiderato.

Tale ambiguità, invece, emerge a riga `3`, essendo che l'array di oggetti della classe `Class` restituito dal metodo `getDeclaredClasses()` non è unidimensionale, bensì è composto da tre oggetti con ordinamento non noto e riferiti alle classi interne alla classe `MapAdapter`.

Affinché la disposizione di questi oggetti nell'array non vada a inficiare la logica di funzionamento delle istruzioni nelle righe successive, bisogna scansionare l'array e cercare mediante un opportuno controllo l'oggetto riferito alla classe di nostro interesse, ossia `EntryMapAdapter`.

Tuttavia in questa prima realizzazione del metodo riflessivo, compilato ed eseguito utilizzando una vecchia versione di JDK e JRE, non è stata considerata questa esigenza ma ho ragionato come segue.

Dopo una semplice sperimentazione per verificare quale fosse l'ordinamento degli oggetti nell'array restituito a riga `3`, ero arrivato a ipotizzare, ingenuamente, che fossero disposti in ordine alfabetico rispetto al nome delle classi interne a `MapAdapter` a cui fanno riferimento.

Secondo questa debole ipotesi, ero dunque giunto a stabilire che la classe interna `EntryMapAdapter` si trovasse in cella `[2]`, e così era: si ricordi che le classi interne a `MapAdapter` sono `BackedCollection`, `BackedSet` e `EntryMapAdapter`.

Sebbene questa ipotesi funzioni con le vecchie versioni di JDK e JRE di un po' di anni fa e porti a compimento tutti i metodi di test con successo, nelle nuove versioni di questi strumenti software non è più vero: questa differenza evidenzia come nel tempo sia stata modificata l'implementazione del metodo `getDeclaredClasses()` e come una qualunque supposizione su funzionamenti non documentati porti prima o poi a incontrare errori di esecuzione.

3.3 Esecuzione del programma con il metodo originario e le ultime versioni di JDK e JRE

Con le nuove versioni di JDK e JRE, l'esecuzione del programma con il metodo originario sopra riportato produce il fallimento di tutti i metodi di test `@Test` della mappa, essendo tale metodo richiamato all'interno del metodo `@Before public void setUp_en()`, utilizzato per inizializzare la entry chiave-valore `en`, variabile membro della Test Suite `TestMap`, prima dell'esecuzione di ciascun test.

L'errore segnalato da ciascun metodo di test della mappa è `wrong number of arguments: 3 expected: 1`, perché la classe interna a `MapAdapter` effettivamente selezionata dal metodo riflessivo non è quella desiderata, ossia `EntryMapAdapter`, ma bensì `BackedCollection` (è il caso del mio interprete Java) oppure `BackedSet`: per ovvie ragioni ne consegue anche un'errata selezione del costruttore.

3.3.1 Output del programma originario

```
_____ myTest.TestCollection IN ESECUZIONE... _____
Ho eseguito 21 tests in 31 millisecondi.
Tutti i test sono stati eseguiti correttamente? true
Sono falliti 0 tests.


_____ myTest.TestSet IN ESECUZIONE... _____
Ho eseguito 21 tests in 15 millisecondi.
Tutti i test sono stati eseguiti correttamente? true
Sono falliti 0 tests.


_____ myTest.TestMap IN ESECUZIONE... _____
Ho eseguito 37 tests in 16 millisecondi.
Tutti i test sono stati eseguiti correttamente? false
Sono falliti 37 tests.
entrySet(myTest.TestMap): wrong number of arguments: 3 expected: 1
put_key_value(myTest.TestMap): wrong number of arguments: 3 expected: 1
[...]
isEmpty(myTest.TestMap): wrong number of arguments: 3 expected: 1
```

Per brevità, si riportano solo alcune delle segnalazioni d'errore da parte dei metodi di test della mappa: le restanti, sempre dello stesso tipo, si suppongano incluse in `[...]`.

3.4 Messaggio da portare a casa

Mai fare ipotesi su comportamenti non dettagliati nella documentazione Java per non introdurre vulnerabilità ed errori di esecuzione in un programma.

I dettagli realizzativi di questi comportamenti possono cambiare con le nuove versioni di JDK e JRE, ma non essendo esposti nella documentazione, non è possibile sapere quando e come cambiano.

Questi comportamenti devono dunque essere gestiti attentamente mediante opportuni controlli al fine di ottenere risultati deterministici.

4 METODO ADATTATO, FUNZIONANTE CON LE ULTIME VERSIONI DI JDK E JRE

```
1  private HEntry instanceEntryMapAdapter(Object key, Object value) throws Exception {
2          // Inizializzo a null l'oggetto entryClass della classe Class.
3          Class entryClass = null;
4
5          // Ottengo un array di oggetti della classe Class corrispondenti alle classi
6          // interne alla classe MapAdapter.
7          Class[] classes = MapAdapter.class.getDeclaredClasses();
8
9          // Itero l'array classes di oggetti Class e assegno all'oggetto entryClass il
10         // riferimento dell'oggetto Class associato alla classe protected EntryMapAdapter.
11         for (Class cl : classes) {
12                 // Condizione in linguaggio naturale: la classe associata all'oggetto cl
13                 // implementa l'interfaccia HEntry?
14                 if (HEntry.class.isAssignableFrom(cl)) {
15                         // In caso affermativo, assegna il riferimento dell'oggetto cl
16                         // all'oggetto entryClass.
17                         entryClass = cl;
18                 }
19         }
20
21         // Inizializzo un array di oggetti della classe Class composto da tre elementi:
22         // il primo parametro MapAdapter.class rappresenta l'oggetto della classe Class
23         // associato alla classe MapAdapter, mentre il secondo e terzo parametro sono
24         // entrambi oggetti Object.class della classe Class associati alla classe
25         // Object.
26         Class[] paramArray = { MapAdapter.class, Object.class, Object.class };
27
28         // Invocando il metodo getDeclaredConstructor(paramArray) sull'oggetto
29         // entryClass della classe Class associato alla classe EntryMapAdapter, si
30         // estrae il costruttore della classe EntryMapAdapter avente come parametri
31         // quelli specificati nell'array paramArray appena definito.
32         Constructor constr = entryClass.getDeclaredConstructor(paramArray);
33
34         // Inizializzo un'istanza map di MapAdapter, necessaria per poter
35         // istanziare entry chiave-valore della classe EntryMapAdapter essendo
36         // quest'ultima non statica.
37         MapAdapter map = new MapAdapter();
38
39         // Il valore di ritorno di questo metodo ausiliario coincide con il
40         // valore ritornato dal metodo newInstance(map, key, value), ossia un'istanza
41         // della classe EntryMapAdapter con chiave key e valore value, chiamato
42         // sull'oggetto constr della classe Constructor definito prima
43         return (HEntry) constr.newInstance(map, key, value);
44 }
```

I commenti qui riportati sono un riassunto di quelli presenti nel codice del progetto: per visionare i commenti completi fare riferimento al codice.

4.1 SOLUZIONE ATTUATA

Affinché il programma funzioni correttamente anche con le nuove versioni di JDK e JRE, è necessario eliminare ogni criticità legata alla disposizione degli oggetti della classe `Class` e degli oggetti della classe `Constructor` negli array restituiti rispettivamente dai metodi `getDeclaredClasses()` e `getDeclaredConstructors()`.

Ciò si ottiene selezionando "ad hoc" la classe `EntryMapAdapter`, interna alla classe `MapAdapter`, e il relativo costruttore `EntryMapAdapter(Object key, Object value)` attuando opportuni controlli e tenendo presente che questa classe interna è `protected` e non statica.

Procediamo con ordine.

4.1.1 Righe `2-19`: selezione della classe interna a `MapAdapter`, `protected` e non statica `EntryMapAdapter`

Per selezionare univocamente la classe `EntryMapAdapter`, si invoca il metodo `getDeclaredClasses()` sull'oggetto `MapAdapter.class` per ottenere l'array di oggetti della classe `Class` associati alle classi interne alla classe `MapAdapter`.

Dopodiché lo si scansiona mediante un `for-each statement` per individuare e salvare nella variabile oggetto `entryClass` l'oggetto associato alla classe desiderata `EntryMapAdapter`: per ciascun oggetto `cl` dell'array, si verifica se la classe associata implementa l'interfaccia `HEntry` invocando il metodo `isAssignableFrom(cl)` sull'oggetto `HEntry.class`.

Terminata la scansione, la variabile oggetto `entryClass` sarà valorizzata dall'oggetto della classe `Class` associato alla classe interna a `MapAdapter`, `protected` e non statica `EntryMapAdapter`.

4.1.2 Righe `21-32`: selezione del costruttore `EntryMapAdapter(Object key, Object value)`

Selezionata la classe interna a `MapAdapter`, `protected` e non statica `EntryMapAdapter`, per selezionare univocamente il costruttore `EntryMapAdapter(Object key, Object value)`, si inizializza un array `paramArray` di oggetti della classe `Class` con quelli associati alle classi `MapAdapter`, `Object` e `Object`, ossia `MapAdapter.class`, `Object.class` e `Object.class`.

Dopodiché si invoca il metodo `getDeclaredConstructor(paramArray)` sull'oggetto `entryClass` per estrarre il costruttore `EntryMapAdapter(Object key, Object value)` che viene salvato nella variabile oggetto `constr` della classe `Constructor`.

Per capire questo passaggio, bisogna porre attenzione a quali oggetti riempiono l'array `paramArray` e alla loro disposizione: procedendo in ordine, il primo oggetto `MapAdapter.class` è richiesto dal metodo `getDeclaredConstructor(Class… parameterTypes)` come primo parametro perché l'oggetto `entryClass` su cui viene invocato rappresenta una classe interna a `MapAdapter` e dichiarata in un contesto non statico; a seguire, invece, vi sono gli oggetti della classe `Class` associati al tipo dei parametri effettivi del costruttore della classe `EntryMapAdapter` che si vuole invocare, ossia due oggetti `Object.class`, uno per la chiave `key` e uno per il valore `value` della entry chiave-valore.

Al termine di queste istruzioni, la variabile oggetto `constr` sarà valorizzata dall'oggetto della classe `Constructor` associato al costruttore `EntryMapAdapter(Object key, Object value)` della classe `EntryMapAdapter`.

4.1.3 Righe `34-43`: creazione di una nuova istanza della classe `EntryMapAdapter`

Selezionato il costruttore `EntryMapAdapter(Object key, Object value)` della classe `EntryMapAdapter`, possiamo ora procedere alla creazione di una nuova istanza invocando il metodo `newInstance(map, key, value)` sul costruttore `constr`.

Notare che il primo parametro `map` è un oggetto della classe `MapAdapter`, istanziato appena prima dell'invocazione: esso è richiesto come primo parametro perché la classe dichiarante del costruttore, ossia `EntryMapAdapter`, è una classe interna a `MapAdapter` e dichiarata in un contesto non statico.

Invece, i successivi due parametri, `key` e `value`, sono i valori effettivi passati al costruttore e che vanno a inizializzare la entry chiave-valore della classe `EntryMapAdapter` che si sta istanziando: in particolare, `key` e `value` sono i parametri accolti dal metodo che stiamo adattando.

A questo punto, dopo aver invocato il metodo `newInstance(map, key, value)` per creare una nuova istanza della classe `EntryMapAdapter`, per essere restituita all'esterno è necessario prima convertirla, mediante cast esplicito, ad una variabile dell'interfaccia `HEntry`, e il gioco è fatto.

4.2 OUTPUT DEL PROGRAMMA ADATTATO

```
_____________ myTest.TestCollection IN ESECUZIONE... _____________
Ho eseguito 21 tests in 31 millisecondi.
Tutti i test sono stati eseguiti correttamente? true
Sono falliti 0 tests.


_____________ myTest.TestSet IN ESECUZIONE... _____________
Ho eseguito 21 tests in 16 millisecondi.
Tutti i test sono stati eseguiti correttamente? true
Sono falliti 0 tests.


_____________ myTest.TestMap IN ESECUZIONE... _____________
Ho eseguito 37 tests in 58 millisecondi.
Tutti i test sono stati eseguiti correttamente? true
Sono falliti 0 tests.
```

5 DETTAGLI TECNICI

5.1 VERSIONE DI JDK E JRE UTILIZZATA

```
C:\Windows\System32>javac -version
javac 18.0.1.1

C:\Windows\System32>java -version
java version "18.0.1.1" 2022-04-22
Java(TM) SE Runtime Environment (build 18.0.1.1+2-6)
Java HotSpot(TM) 64-Bit Server VM (build 18.0.1.1+2-6, mixed mode, sharing)
```

5.2 DOCUMENTAZIONE DI RIFERIMENTO

https://docs.oracle.com/en/java/javase/18/docs/api/index.html