

Assignment 6

Rasika Mohod

rmohod@gmu.edu

G01044774

Introduction to Software Testing (Edition 2): Book by Jeff Offutt and Paul Amman

Story explaining solution for Exercise 6.4 Problem 1:

Let me narrate you one of my interesting programming experiences where I wrote test cases for testing Iterator interface and its correct implementation using ArrayList.

Iterator interface consists of three methods hasNext(), next() and remove() with different return types and return values for each of these methods. However, all of these methods work with same parameter i.e. state of the iterator.

My programming task was to write a JUnit test case which would fail, i.e. would not throw *ConcurrentModificationException* on calling remove() method even after ArrayList is modified. Also the modification of ArrayList had to be made without using remove() method.

I devised a simple solution to satisfy this scenario. We know that the *ConcurrentModificationException* is thrown by methods that have detected concurrent modification of an object when such modification is not permissible. Thus in case of iterator this exception is thrown when the object i.e. ArrayList for which iterator is defined is modified in the duration of use of iterator.

However, If you are wondering how Iterator checks for the modification, its implementation is present in AbstractList class (super class of ArrayList) where an int variable modCount is defined that provides the number of times ArrayList size has been changed. This value is used in every next() call to check for any modifications in a function checkForComodification().

Since I wanted to modify the ArrayList and yet avoid the *ConcurrentModificationException*, the only way was to modify the list in a way that does not change the size of the list i.e. modCount is not modified. This could be done by changing the existing value of one of the element in the list.

Thus, I devised a solution to modify the list by using set() method and set new value to one of the element in the list. Now, as the list had been modified during the iterator use but this modification was not in terms of change in size of list, the remove() call failed to throw the *ConcurrentModificationException*.

This way I could write a test case that would fail even after list was modified during the use of iterator defined on that list.

JUnit Test case for Excericse 6.4 Problem 1 (highlighted in yellow):

```
package com.gmu.rmohod;

import static org.junit.Assert.*;

import java.util.ArrayList;
import java.util.ConcurrentModificationException;
import java.util.Iterator;
import java.util.List;

import org.junit.Before;
import org.junit.Test;

public class IteratorTest {

    private List<String> list;
    private Iterator<String> itr;

    @Before
    public void setUp() {
        list = new ArrayList<String>();

        list.add("cat");
        list.add("dog");

        itr = list.iterator();
    }



    @Test
    public void testHasNext_BaseCase() {
        assertTrue(itr.hasNext());
    }

    //Test case for Exercise 6.4 Problem 2
    @Test (expected=ConcurrentModificationException.class)
    public void testRemove_C5() {
        itr.next();
        list.set(0, "horse");
        itr.remove();
    }
}
```

JUnit Test case execution results:

Package Explorer

JUnit


 


Finished after 0.016 seconds


Runs: 2/2

Errors: 0



Failures: 1


 com.gmu.rmohod.IteratorTest [Runner: JUnit 4] (0.000 s)

 testHasNext_BaseCase (0.000 s)

 testRemove_C5 (0.000 s)

Failure Trace

 java.lang.AssertionError: Expected exception: java.util.ConcurrentModificationException