CS447 Project

GROUP 6
Yunfei Cui 20455776
Junnan Chen 20595284

April 3, 2015

Part 1 c

FOR CINDY!!!!

Part 2

0.1 Resolving Bugs in Apache Commons

0.1.1 CID:10022

Classification False Positive.
Explain The function here returns a KeySetView object and doesn't need to call super.keyset() to get the keyset of map.

0.1.2 CID:10023

Classification False Positive.
Explain In this case, the function map.keyset() is the same function as super.keyset().

0.1.3 CID:10024

Classification False Positive.
Explain Same as the previous one.

0.1.4 CID:10025

Classification False Positive.
Explain The currentIterator has to be checked before because in different branches, the program contains other if-statements and should execute different code.

0.1.5 CID:10026

Classification Bug
Explain The get function reads the list l, integer first and last without holding lock. Some of the usages didn't require a lock before calling this function. That might cause a conflict or fault.
Suggested Fix

```
private List get(List l) {
             if (list != expected) {
                 throw new ConcurrentModificationException();
             }
              synchronized(FastArrayList.this)
              {
             List tore = l.subList(first, last);
              return l.subList(first, last);
              }
         }
```

0.1.6 CID:10027

Classification False Positive.
Explain nextGreater function will not return null because we have checked that (deletedNode.getRight(index) != null) and it will go to the first else if (line 622, TreeBidiMap.java) and return leastNode.

0.1.7 CID:10028

Classification Bug
Explain To avoid synchronous modification, the parameter last should acquire a lock when used.
Suggested Fix

```
public void add(Object o) {
           checkMod();
           synchronized(FastArrayList.this){
             int i = nextIndex();
             get().add(i, o);
           last++;
                expected = list;
                iter = get().listIterator(i + 1);
                lastReturnedIndex = -1;
            }
        }
```

0.1.8 CID:10029

Classification Bug
Explain If two threads execute the function synchronously, both of them will pass the first

check(lastReturned == null) and try to acquire the lock. After the first thread succeeds removing the list, parameter lastRetured is set to null. In this case, the second thread should throw IllegalStateException, but it already passed that check.
Suggested Fix

```
public void remove() {
  synchronized (FastHashMap.this) {
                if (lastReturned == null) {
                    throw new IllegalStateException();
                }
                if (fast) {
                        if (expected != map) {
                            throw new ConcurrentModificationExcept
                        }
                        FastHashMap.this.remove(lastReturned.getKe
                        lastReturned = null;
                        expected = map;
                } else {
                    iterator.remove();
                    lastReturned = null;
                }
      }
}
```

0.1.9 CID:10030

Classification False Positive.
Explain Normally, this is not a problem because the parameter lock is protected, only the member functions can use lock. And all the other member functions won't require to synchronized(map) outside the synchronized(lock). So there won't be a deadlock.

0.1.10 CID:10031

Classification False Positive.
Explain the function rotateLeft didn't check if the current node has a rightsubtree. But the function is private and is called only when the current node has a rightsubtree that is 2 levels deeper than its leftsubtree because of the definition of AVL tree.

0.1.11 CID:10032

Classification Bug
Explain If two threads execute the function synchronously, both of them will pass the first check(bucket < buckets.length) and try to acquire the lock. After the first thread succeeds removing the list, parameter bucket adds 1. If after the adding ,bucket equals to buckets.length, in this case, the second thread shouldn't pass the check, but it already passed that check.
Suggested Fix

```
public boolean hasNext() {
            if (current.size() > 0) return true;
```

```
            synchronized(bucket){
            while (bucket < buckets.length) {
                synchronized (locks[bucket]) {
                    Node n = buckets[bucket];
                    while (n != null) {
                        current.add(n);
                        n = n.next;
                    }
                    bucket++;
                    if (current.size() > 0) return true;
                }
            }
            }
            return false;
        }
```

0.1.12 CID:10033

Classification False Positive.
Explain This is not a bug because the function is in "INVERSE MAP ENTRY", what we need here is to get the reversed entry. The value should be passed as a key and the key should be passed as a value.

0.1.13 CID:10034

Classification Bug
Explain Same as CID 10032.
Suggested Fix

```
public boolean hasNext() {
            if (current.size() > 0) return true;
            synchronized(m_bucket){
            while (bucket < m_buckets.length) {
                synchronized (m_locks[bucket]) {
                    Node n = m_buckets[bucket];
                    while (n != null) {
                        current.add(n);
                        n = n.next;
                    }
                    bucket++;
                    if (current.size() > 0) return true;
                }
            }
            }
            return false;
        }
```

0.1.14 CID:10035

Classification Bug
Explain To avoid synchronous modification, the parameter last should acquire a lock when used.
Suggested Fix

```
public void remove() {
            checkMod();
            if (lastReturnedIndex < 0) {
                throw new IllegalStateException();
            }
            synchronized(FastArrayList.this){
            get().remove(lastReturnedIndex);
            last--;
            expected = list;
            iter = get().listIterator(lastReturnedIndex);
            lastReturnedIndex = -1;
            }
        }
```

0.1.15 CID:10036

Classification Bug
Explain Same as CID 10029.
Suggested Fix

```
public void remove() {
  synchronized (FastHashMap.this) {
                if (lastReturned == null) {
                    throw new IllegalStateException();
                }
                if (fast) {
                        if (expected != map) {
                            throw new ConcurrentModificationExcept
                        }
                        FastHashMap.this.remove(lastReturned.getKe
                        lastReturned = null;
                        expected = map;
                } else {
                    iterator.remove();
                    lastReturned = null;
                }
      }
}
```

0.1.16 CID:10037

Classification False Positive.
Explain Similar with CID:10031

0.1.17 CID:10038

Classification False Positive.
Explain Normally, this is not a problem because the parameter lock is protected, only the member functions can use lock. And all the other member functions won't require to synchronized(list) outside the synchronized(lock). So there won't be a deadlock.

0.1.18 CID:10039

Classification False Positive.
Explain same as CID 10027.

0.1.19 CID:10040

Classification False Positive.
Explain Normally, this is not a problem because the parameter lock is protected, only the member functions can use lock. And all the other member functions won't require to synchronized(map) outside the synchronized(lock). So there won't be a deadlock.

0.1.20 CID:10041

Classification Intentional
Explain In this case, all the member functions are not locked in ReferenceMap. The developer intentionally wrote it this way because for most single threads, locks are not necessary, and for multi-threads users, the developer wrote comments to remind them add a lock when calling the corresponding functions.
Bad Practice thread1 reads the value of original modCount as v1. Control switches to thread2. Thread2 reads value of original modCount as v1. Control switches to thread1. Thread1 changes the value of modCount to v1+1. Control switches to thread2. Thread2 changes the value of modCount to v1+1. This is a fault and the correct value should be v1+2.

0.1.21 CID:10042

Classification Intentional **Explain & Bad Practice** The same reason as the previous one(CID:10041).

Analyze Own Code

Total Bug: 1.
Classification Bug.
Explain Argument fixed will change the output format of cerr. It should be reset after use.
Fix

```
ios::fmtflags f(cerr.flags());
//code
cerr.flags(f);
```