Atomicity violation Low

Atomicity violations caused by improper usage of `ConcurrentHashMap` or `ConcurrentLinkedQueue` can result in crashses or incorrect program results. An atomicity violation happens when two instructions are expected and assumed to execute together ("atomically") but another thread executing on the same data structure in the time between them breaks the atomicity expectation.

Detector ID
java/concurrency-atomicity-violation@v1.0

Category
Security
Common Weakness Enumeration (CWE)
-
Tags

availability # concurrency # security-context

Noncompliant example

```
1  public class ConcurrencyAtomicityViolationNoncompliant {
2
3      private ConcurrentHashMap<String, String> concurrentMap = new ConcurrentHashMap
4
5      public void getValue(String key) {
6          // Noncompliant: the key could be removed from the map between the first ca
7          if (concurrentMap.containsKey(key)) {
8              String value = concurrentMap.get(key);
9              System.out.println(value.length());
10         }
11     }
12
13     public void deleteValue(String key) {
14         concurrentMap.remove(key);
```

```
    }
}
```

Compliant example

```
public class ConcurrencyAtomicityViolationCompliant {

    private ConcurrentHashMap<String, String> concurrentMap = new ConcurrentHashMap

    public void getValue(String key) {
        // Compliant: the value is checked for null before being accessed.
        String value = concurrentMap.get(key);
        if (value != null) {
            System.out.println(value.length());
        }
    }

    public void deleteValue(String key) {
        concurrentMap.remove(key);
    }
}
```