LCK04-J. Do not synchronize on a collection view if the backing collection is accessible

The Java Tutorials, Wrapper Implementations [Java Tutorials], warns about the consequences of failing to synchronize on an accessible collection object when iterating over its view:

> *It is imperative that the user manually synchronize on the returned `Map` when iterating over any of its `Collection` views rather than synchronizing on the `Collection` view itself.*

Disregarding this advice may result in nondeterministic behavior.

Any class that uses a collection view rather than the backing collection as the lock object may end up with two distinct locking strategies. When the backing collection is accessible to multiple threads, the class that locked on the collection view has violated the thread-safety properties and is unsafe. Consequently, programs that both require synchronization while iterating over collection views and have accessible backing collections must synchronize on the backing collection; synchronization on the view is a violation of this rule.

Unknown macro: 'mc'

Noncompliant Code Example (Collection View)

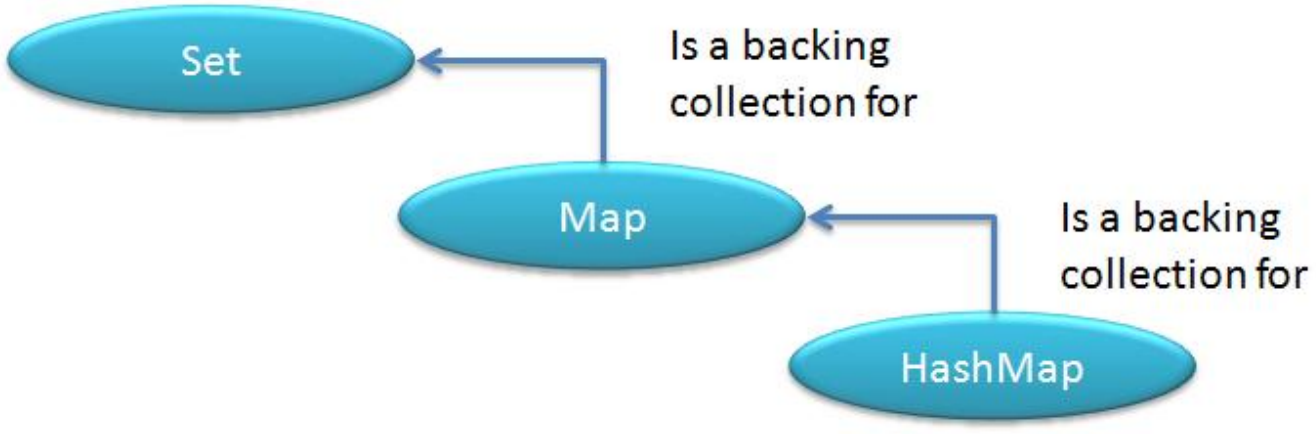
This noncompliant code example creates a `HashMap` object and two view objects: a synchronized view of an empty `HashMap` encapsulated by the `mapVie` field and a set view of the map's keys encapsulated by the `setView` field. This example synchronizes on `setView` [Java Tutorials].

```
private final Map<Integer, String> mapView =
    Collections.synchronizedMap(new HashMap<Integer, String>());
private final Set<Integer> setView = mapView.keySet();

public Map<Integer, String> getMap() {
  return mapView;
}

public void doSomething() {
  synchronized (setView) {  // Incorrectly synchronizes on setView
    for (Integer k : setView) {
      // ...
    }
  }
}
```

In this example, `HashMap` provides the backing collection for the synchronized map represented by `mapView`, which provides the backing collection for `se tView`, as shown in the following figure.



The `HashMap` object is inaccessible, but `mapView` is accessible via the public `getMap()` method. Because the `synchronized` statement uses the intrinsic lock of `setView` rather than of `mapView`, another thread can modify the synchronized map and invalidate the `k` iterator.

Compliant Solution (Collection Lock Object)

This compliant solution synchronizes on the `mapView` field rather than on the `setView` field:

```
private final Map<Integer, String> mapView =
    Collections.synchronizedMap(new HashMap<Integer, String>());
private final Set<Integer> setView = mapView.keySet();

public Map<Integer, String> getMap() {
  return mapView;
}

public void doSomething() {
  synchronized (mapView) {  // Synchronize on map, rather than set
    for (Integer k : setView) {
      // ...
    }
  }
}
```

This code is compliant because the map's underlying structure cannot be changed during the iteration.

Risk Assessment

Synchronizing on a collection view instead of the collection object can cause nondeterministic behavior.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
LCK04-J	Low	Probable	Medium	P4	L3

Automated Detection

Some static analysis tools are capable of detecting violations of this rule.

Tool	Version	Checker	Description
ThreadSafe	1.3	**CCE_CC_SYNC_ON_VIEW** **CCE_CC_ITER_VIEW_NO_LOCK** **CCE_CC_ITER_VIEW_BOTH_LOCKS** **CCE_CC_ITER_VIEW_WRONG_LOCK**	Implemented

Bibliography

[Java Tutorials]	Wrapper Implementations

Issue Tracking

Review List

suggested => "HashMap is not accessible, but the Map view is. Because the set view is synchronized instead of the map view, another thread can modify the contents of map and invalidate the k iterator."