LCK02-J. Do not synchronize on the class object returned by getClass()

Synchronizing on the return value of the Object.getClass() method can lead to unexpected behavior. Whenever the implementing class is subclassed, the subclass locks on the subclass's type. The Class object of the subclass is entirely distinct from the Class object of the parent class.

According to The Java Language Specification, §4.3.2, "The Class Object" [JLS 2005]:

> A class method that is declared synchronized synchronizes on the lock associated with the class object of the class.

Programmers who interpret this to mean that a subclass using getClass() will synchronize on the Class object of the base class are incorrect. The subclass will actually lock on its own Class object, which may or may not be what the programmer intended. Consequently, programs must not synchronize on the class object returned by getClass().

The programmer's actual intent should be clearly documented or annotated. Note that when a subclass fails to override an accessible noncompliant superclass's method, it inherits the method, which may lead to the false conclusion that the superclass's intrinsic lock is available in the subclass.

When synchronizing on a class literal, the corresponding lock object should be inaccessible to untrusted code. Callers from other packages cannot access class objects that are package-private; consequently, synchronizing on the intrinsic lock object of such classes is permitted (see LCK00-J. Use private final lock objects to synchronize classes that may interact with untrusted code for more information).

Noncompliant Code Example (getClass() Lock Object)

In this noncompliant code example, the parse() method of the Base class parses a date and synchronizes on the class object returned by getClass(). The Derived class also inherits the parse() method. However, this inherited method synchronizes on Derived's class object because the inherited parse method's invocation of getClass() is really an invocation of this.getClass(), and the this argument is a reference to the instance of the Derived class.

The Derived class also adds a doSomethingAndParse() method that locks on the class object of the Base class because the developer misconstrued that the parse() method in Base always obtains a lock on the Base class object, and doSomethingAndParse() must follow the same locking policy. Consequently, the Derived class has two different locking strategies and fails to be thread-safe.

```java
class Base {
    static DateFormat format =
        DateFormat.getDateInstance(DateFormat.MEDIUM);

    public Date parse(String str) throws ParseException {
        synchronized (getClass()) {
            return format.parse(str);
        }
    }
}

class Derived extends Base {
    public Date doSomethingAndParse(String str) throws ParseException {
        synchronized (Base.class) {
            // ...
            return format.parse(str);
        }
    }
}
```

Compliant Solution (Class Name Qualification)

In this compliant solution, the class name providing the lock (Base) is fully qualified:

```java
class Base {
    static DateFormat format =
        DateFormat.getDateInstance(DateFormat.MEDIUM);

    public Date parse(String str) throws ParseException {
        synchronized (Base.class) {
            return format.parse(str);
        }
    }
}
```
```java
class Base {
    static DateFormat format =
        DateFormat.getDateInstance(DateFormat.MEDIUM);

    public Date parse(String str) throws ParseException {
        synchronized (Base.class) {
            return format.parse(str);
        }
    }
}
// ...
```

This code example always synchronizes on the `Base.class` object, even when it is called from a `Derived` object.

**Compliant Solution (Class.forName())**

This compliant solution uses the `Class.forName()` method to synchronize on the `Base` class's `Class` object:

```java
class Base {
    static DateFormat format =
        DateFormat.getDateInstance(DateFormat.MEDIUM);

    public Date parse(String str) throws ParseException {
        try {
            synchronized (Class.forName("Base")) {
                return format.parse(str);
            }
        } catch (ClassNotFoundException x) {
            // "Base" not found; handle error
        }
        return null;
    }
}
// ...
```

Never accept untrusted inputs as arguments while loading classes using `Class.forName()` (see SEC03-J. Do not load trusted classes after allowing untrusted code to load arbitrary classes for more information).

**Noncompliant Code Example (getClass() Lock Object, Inner Class)**

This noncompliant code example synchronizes on the class object returned by `getClass()` in the `parse()` method of class `Base`. The `Base` class also has a nested helper class whose `doSomethingAndParse()` method incorrectly synchronizes on the value returned by `getClass()`.

```java
class Base {
    static DateFormat format =
        DateFormat.getDateInstance(DateFormat.MEDIUM);

    public Date parse(String str) throws ParseException {
        synchronized (Base.class) {
            return format.parse(str);
        }
    }
}
// ...
```
```java
class Base {
    static DateFormat format =
        DateFormat.getDateInstance(DateFormat.MEDIUM);

    public Date parse(String str) throws ParseException {
        synchronized (getClass()) { // Intend to synchronizes on Base.class
            return format.parse(str);
        }
    }

    public Date doSomething(String str) throws ParseException {
        return new Helper().doSomethingAndParse(str);
    }

    private class Helper {
        public Date doSomethingAndParse(String str) throws ParseException {
            synchronized (getClass()) { // Synchronizes on Helper.class
                // ...
                return format.parse(str);
            }
        }
    }
}
```

The call to `getClass()` in the `Helper` class returns a `Helper` class object instead of the `Base` class object. Consequently, a thread that calls `Base.parse()` locks on a different object than a thread that calls `Base.doSomething()`. It is easy to overlook concurrency errors in inner classes because they exist within the body of the containing outer class. A reviewer might incorrectly assume that the two classes have the same locking strategy.

**Compliant Solution (Class Name Qualification)**

This compliant solution synchronizes using a `Base` class literal in the `parse()` and `doSomethingAndParse()` methods:

```java
class Base {
    // ...

    public Date parse(String str) throws ParseException {
        synchronized (Base.class) {
            return format.parse(str);
        }
    }

    private class Helper {
        public Date doSomethingAndParse(String str) throws ParseException {
            synchronized (Base.class) { // Synchronizes on Base class literal
                // ...
                return format.parse(str);
            }
        }
    }
}
```

Consequently, both `Base` and `Helper` lock on `Base`'s intrinsic lock. Similarly, the `Class.forName()` method can be used instead of a class literal.

**Risk Assessment**

Synchronizing on the class object returned by `getClass()` can result in nondeterministic behavior.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCK02-J</td>
<td>Medium</td>
<td>Probable</td>
<td>Medium</td>
<td>P8</td>
<td>L2</td>
</tr>
</tbody>
</table>

**Automated Detection**

Some static analysis tools can detect violations of this rule.
<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SonarQube</td>
<td>6.7</td>
<td>S3067</td>
<td></td>
</tr>
</tbody>
</table>

**Bibliography**

- [API 2014]
- [Findbugs 2008]
- [JLS 2005] §4.3.2, "The Class Object"
- [Pugh 2008] "Synchronization"
- [Miller 2009] "Locking"