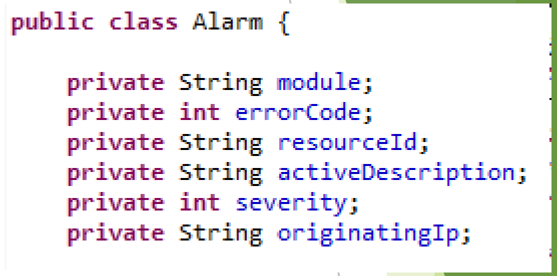
USE OF HAMCREST MATCHERS

Autores: Bárbara Giménez, Carlos Sánchez y Guillermo Gómez

OBJETIVO

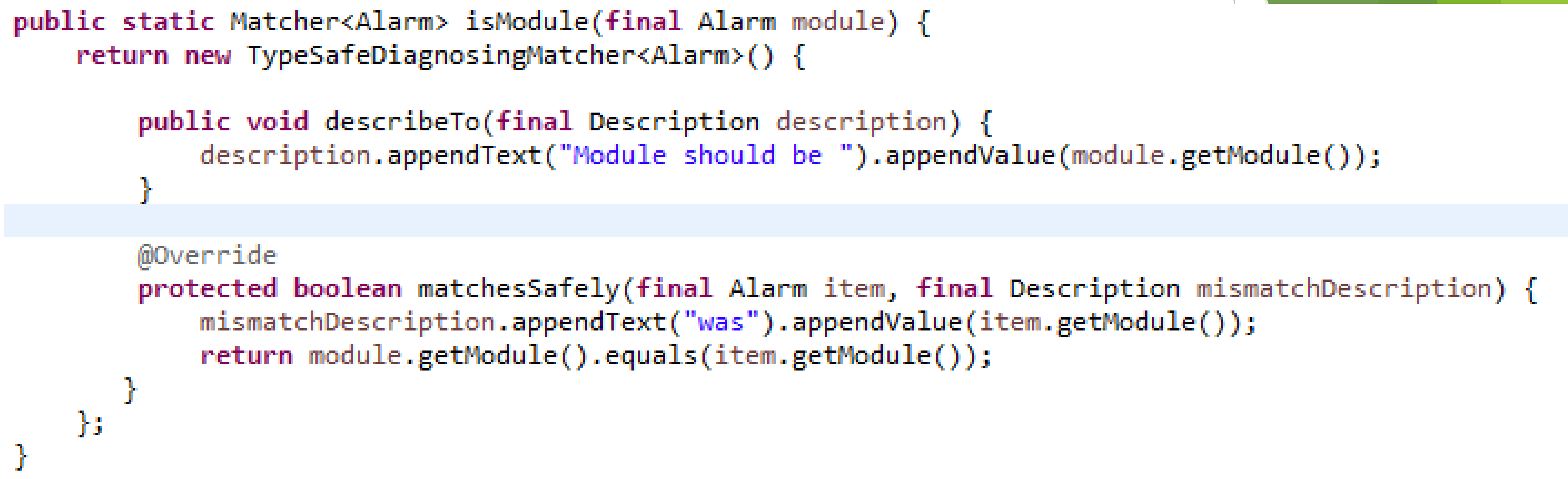
- Partimos de una clase `Alarm.java` que tiene una serie de atributos de los cuáles sólo usaremos tres de ellos.



```
public class Alarm {

    private String module;
    private int errorCode;
    private String resourceId;
    private String activeDescription;
    private int severity;
    private String originatingIp;
```

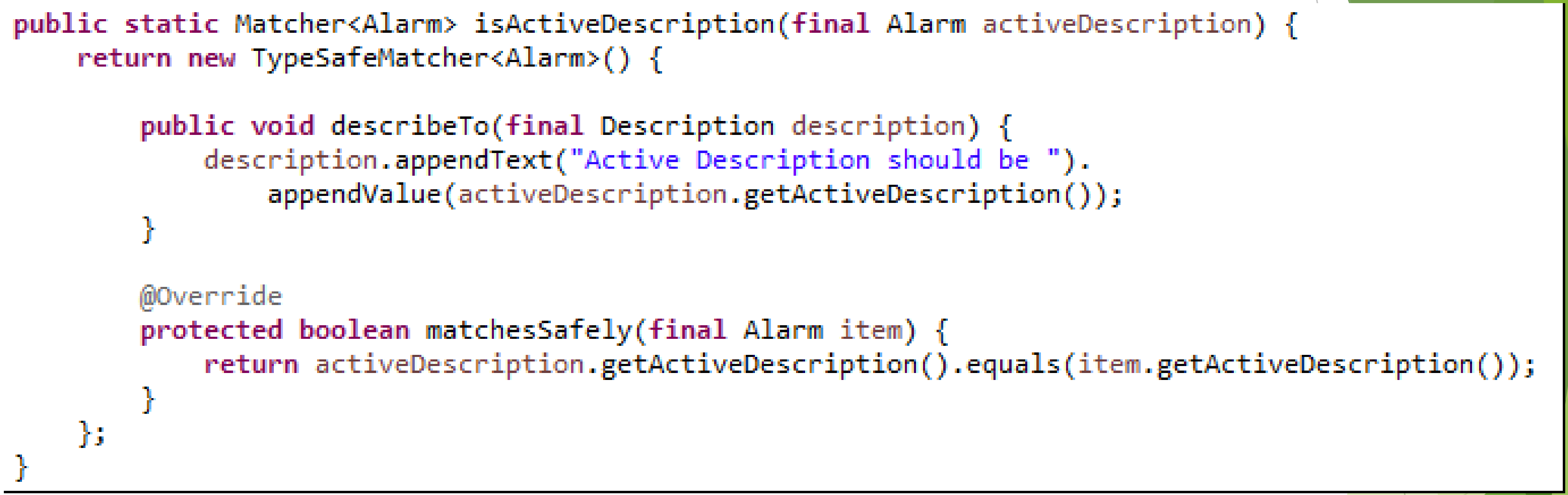
- Idea es usar un matcher que nos permita comparar objetos de tipo `Alarm`.
- Para ello hemos creado un nuevo matcher para cada tipo de atributo que estarán definidos dentro de la clase `MyMatchers.java` y un matcher para el objeto Alarm que hemos llamado `AlarmMatcher`.
- Hemos probado distintos tipo de matcher para ver cómo se comportan.

TypeSafeDiagnosingMatcher



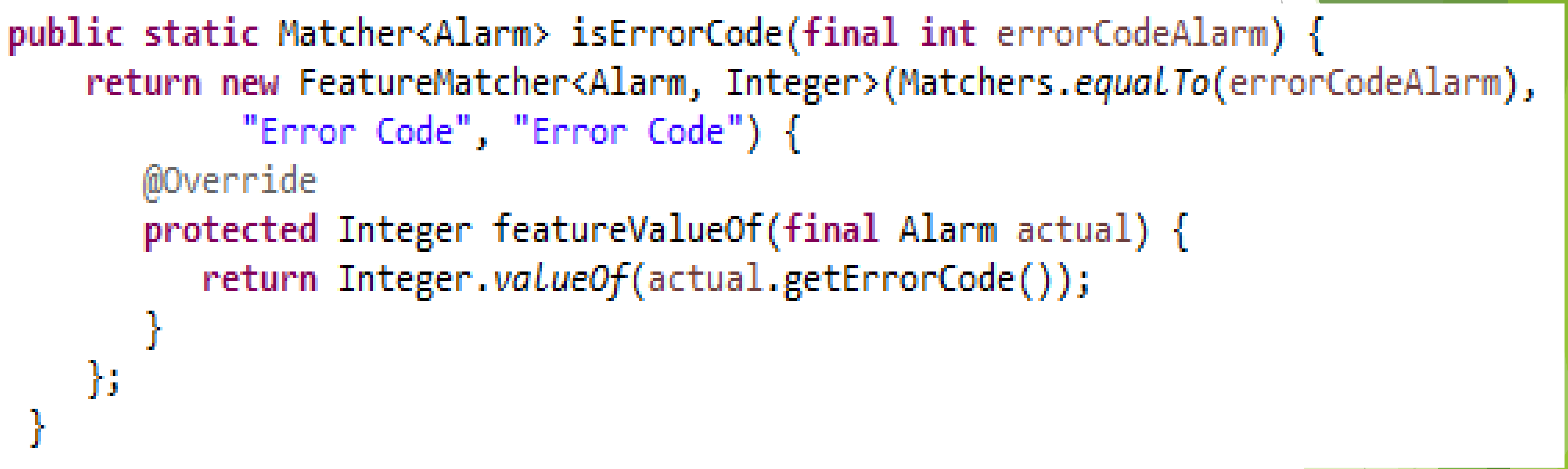
```java
public static Matcher<Alarm> isModule(final Alarm module) {
    return new TypeSafeDiagnosingMatcher<Alarm>() {

        public void describeTo(final Description description) {
            description.appendText("Module should be ").appendValue(module.getModule());
        }

        @Override
        protected boolean matchesSafely(final Alarm item, final Description mismatchDescription) {
            mismatchDescription.appendText("was").appendValue(item.getModule());
            return module.getModule().equals(item.getModule());
        }
    };
}
```

TypeSafeMatcher



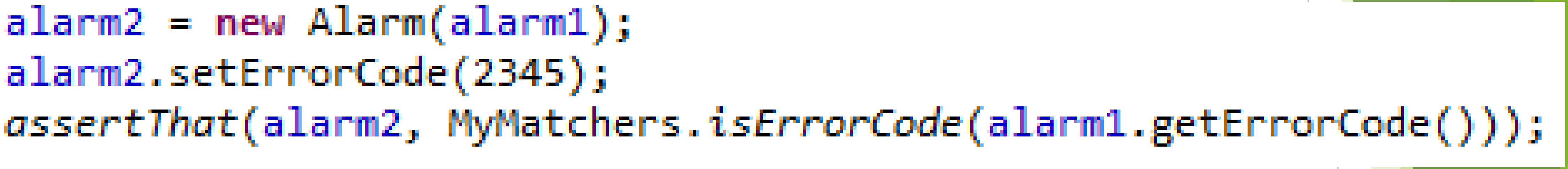
```
public static Matcher<Alarm> isActiveDescription(final Alarm activeDescription) {
    return new TypeSafeMatcher<Alarm>() {

        public void describeTo(final Description description) {
            description.appendText("Active Description should be ").
                appendValue(activeDescription.getActiveDescription());
        }

        @Override
        protected boolean matchesSafely(final Alarm item) {
            return activeDescription.getActiveDescription().equals(item.getActiveDescription());
        }
    };
}
```

FeatureMatcher

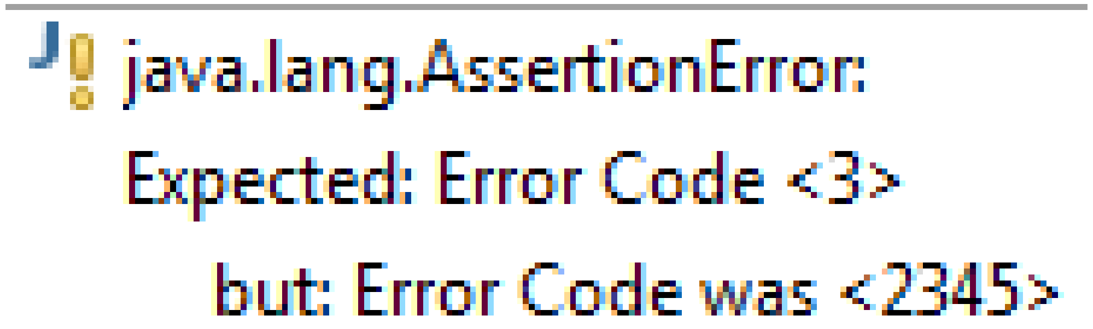


```
public static Matcher<Alarm> isErrorCode(final int errorCodeAlarm) {
    return new FeatureMatcher<Alarm, Integer>(Matchers.equalTo(errorCodeAlarm),
            "Error Code", "Error Code") {
        @Override
        protected Integer featureValueOf(final Alarm actual) {
            return Integer.valueOf(actual.getErrorCode());
        }
    };
}
```

PRUEBA 1: Aplicamos el matcher `FeatureMatcher` de tipo a un atributo

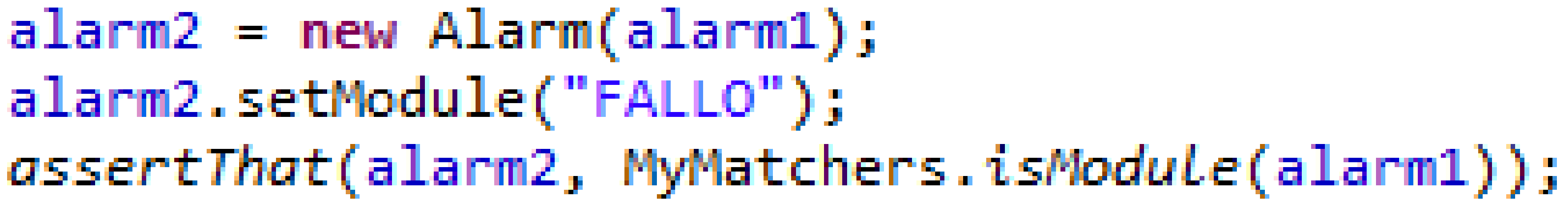


```
alarm2 = new Alarm(alarm1);
alarm2.setErrorCode(2345);
assertThat(alarm2, MyMatchers.isErrorCode(alarm1.getErrorCode()));
```

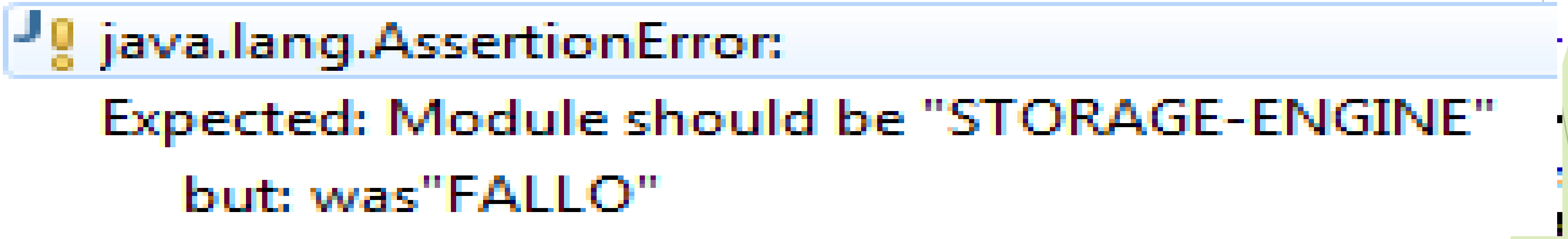


```
java.lang.AssertionError:
Expected: Error Code <3>
     but: Error Code was <2345>
```

PRUEBA 2: Aplicamos un tipo de matcher `TypeSafeDiagnosingMatcher` para un atributo.

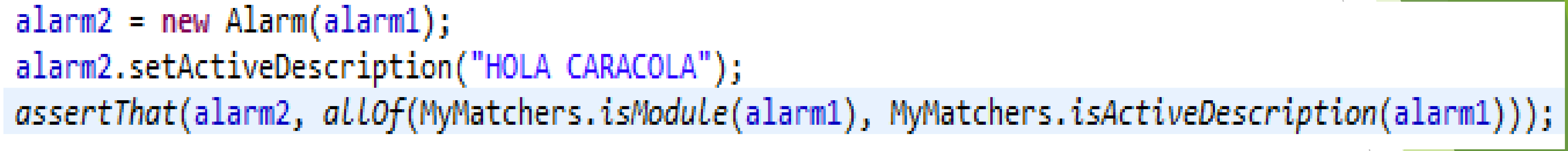


```java
alarm2 = new Alarm(alarm1);
alarm2.setModule("FALLO");
assertThat(alarm2, MyMatchers.isModule(alarm1));
```

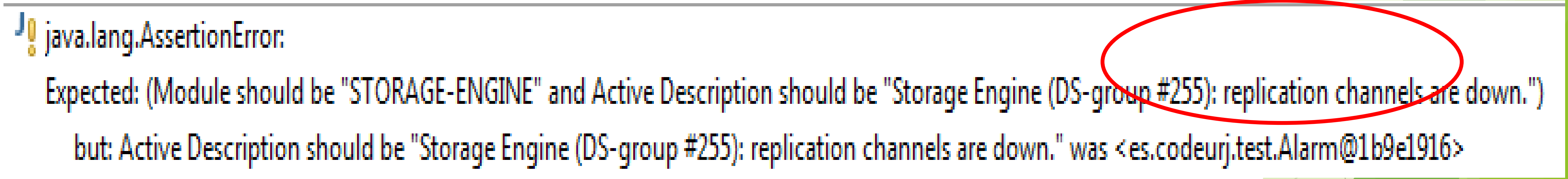


```
java.lang.AssertionError:
Expected: Module should be "STORAGE-ENGINE"
     but: was"FALLO"
```

PRUEBA 3: Aplicamos varios matcher de atributos a la vez con `allOf`. Falla el segundo matcher

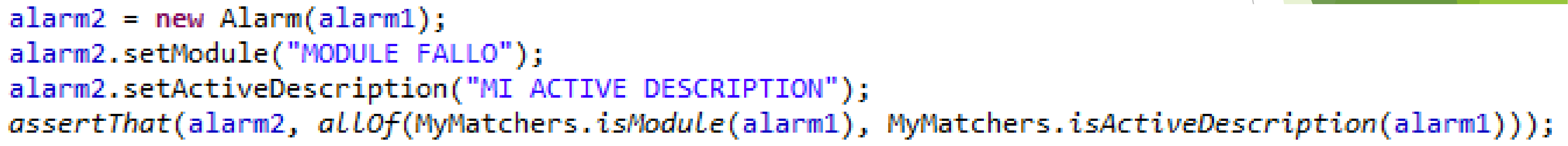


```
alarm2 = new Alarm(alarm1);
alarm2.setActiveDescription("HOLA CARACOLA");
assertThat(alarm2, allOf(MyMatchers.isModule(alarm1), MyMatchers.isActiveDescription(alarm1)));
```

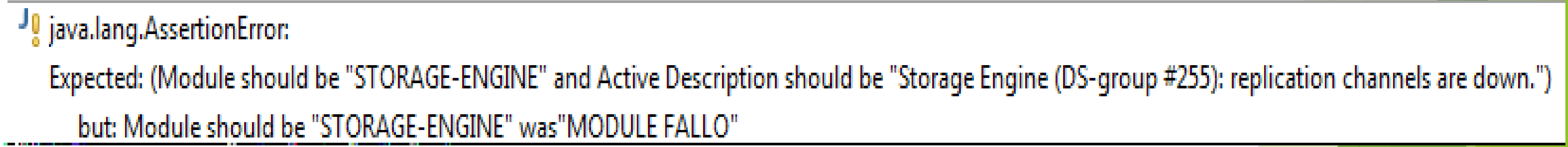


```
java.lang.AssertionError:
Expected: (Module should be "STORAGE-ENGINE" and Active Description should be "Storage Engine (DS-group #255): replication channels are down.")
     but: Active Description should be "Storage Engine (DS-group #255): replication channels are down." was <es.codeurj.test.Alarm@1b9e1916>
```

PRUEBA 4: Aplicamos varios matcher de atributos a la vez con `allOf`. Todos los atributos fallan pero sólo muestra error del primer matcher

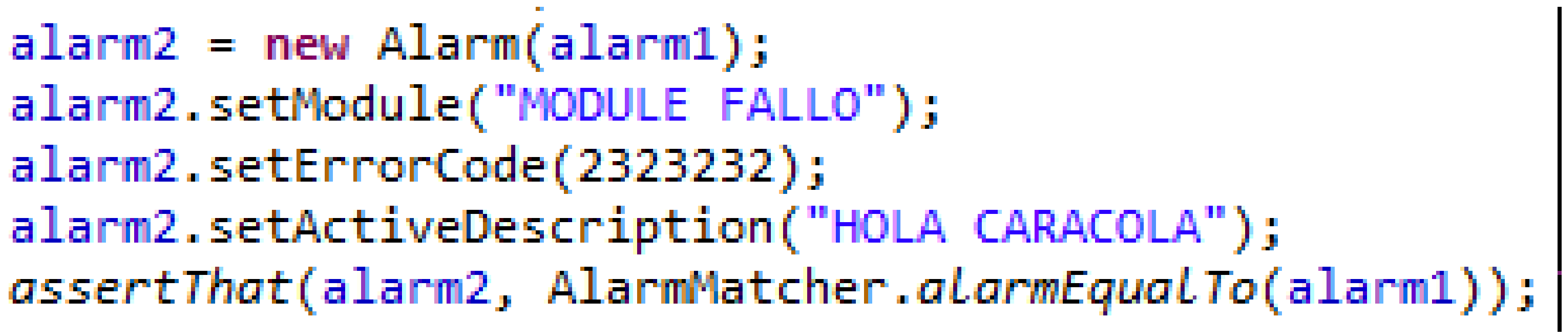


```
alarm2 = new Alarm(alarm1);
alarm2.setModule("MODULE FALLO");
alarm2.setActiveDescription("MI ACTIVE DESCRIPTION");
assertThat(alarm2, allOf(MyMatchers.isModule(alarm1), MyMatchers.isActiveDescription(alarm1)));
```

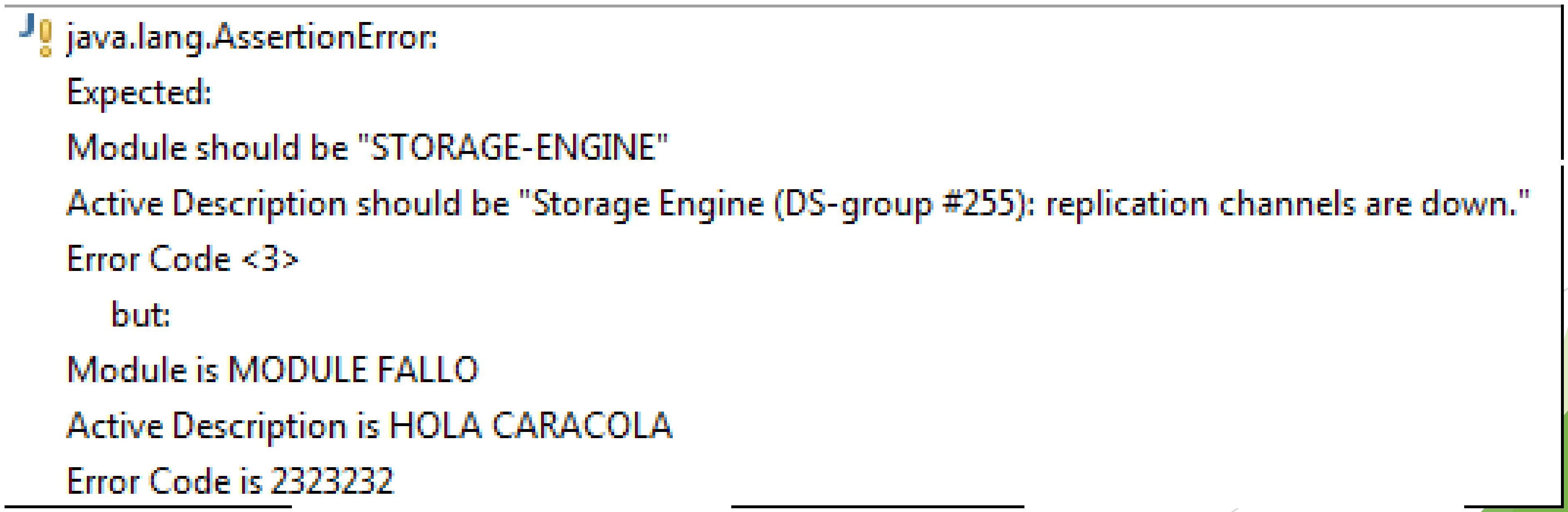


```
java.lang.AssertionError:
Expected: (Module should be "STORAGE-ENGINE" and Active Description should be "Storage Engine (DS-group #255): replication channels are down.")
     but: Module should be "STORAGE-ENGINE" was"MODULE FALLO"
```

PRUEBA 5: Aplicamos un matcher `AlarmMatcher` al objeto `Alarm`.

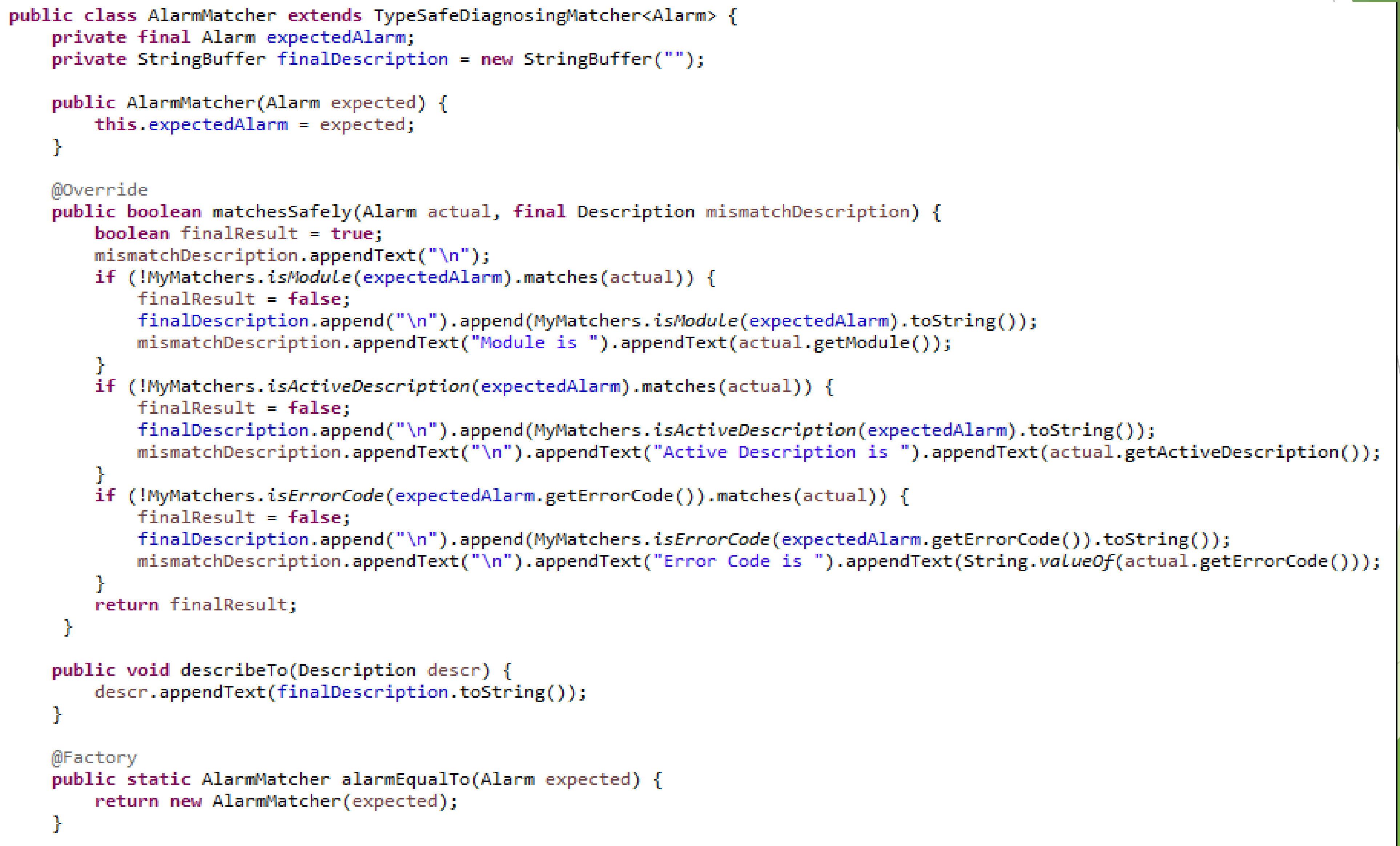


```java
alarm2 = new Alarm(alarm1);
alarm2.setModule("MODULE FALLO");
alarm2.setErrorCode(2323232);
alarm2.setActiveDescription("HOLA CARACOLA");
assertThat(alarm2, AlarmMatcher.alarmEqualTo(alarm1));
```



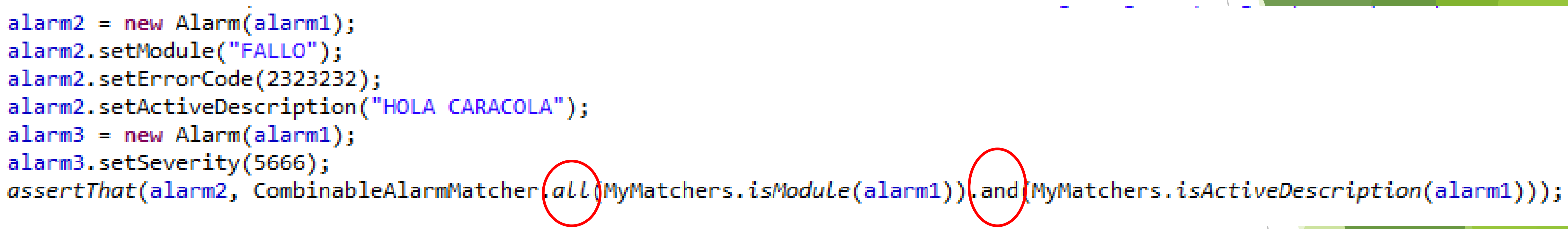
```
java.lang.AssertionError:
Expected:
Module should be "STORAGE-ENGINE"
Active Description should be "Storage Engine (DS-group #255): replication channels are down."
Error Code <3>
     but:
Module is MODULE FALLO
Active Description is HOLA CARACOLA
Error Code is 2323232
```

AlarmMatcher

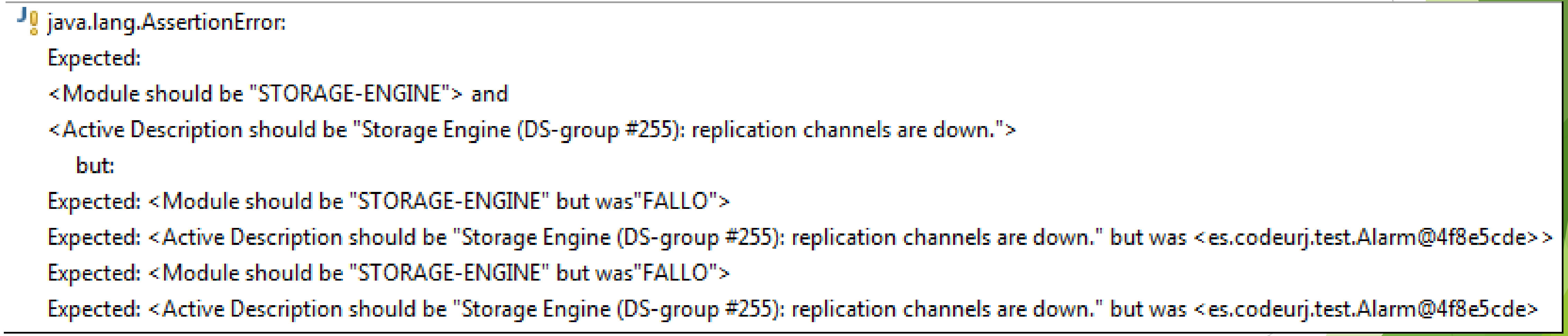


```java
public class AlarmMatcher extends TypeSafeDiagnosingMatcher<Alarm> {
    private final Alarm expectedAlarm;
    private StringBuffer finalDescription = new StringBuffer("");

    public AlarmMatcher(Alarm expected) {
        this.expectedAlarm = expected;
    }

    @Override
    public boolean matchesSafely(Alarm actual, final Description mismatchDescription) {
        boolean finalResult = true;
        mismatchDescription.appendText("\n");
        if (!MyMatchers.isModule(expectedAlarm).matches(actual)) {
            finalResult = false;
            finalDescription.append("\n").append(MyMatchers.isModule(expectedAlarm).toString());
            mismatchDescription.appendText("Module is ").appendText(actual.getModule());
        }
        if (!MyMatchers.isActiveDescription(expectedAlarm).matches(actual)) {
            finalResult = false;
            finalDescription.append("\n").append(MyMatchers.isActiveDescription(expectedAlarm).toString());
            mismatchDescription.appendText("\n").appendText("Active Description is ").appendText(actual.getActiveDescription());
        }
        if (!MyMatchers.isErrorCode(expectedAlarm.getErrorCode()).matches(actual)) {
            finalResult = false;
            finalDescription.append("\n").append(MyMatchers.isErrorCode(expectedAlarm.getErrorCode()).toString());
            mismatchDescription.appendText("\n").appendText("Error Code is ").appendText(String.valueOf(actual.getErrorCode()));
        }
        return finalResult;
     }

    public void describeTo(Description descr) {
        descr.appendText(finalDescription.toString());
    }

    @Factory
    public static AlarmMatcher alarmEqualTo(Alarm expected) {
        return new AlarmMatcher(expected);
    }
```

PRUEBA 6: Usar un clase donde combinar varios matchers, `CombinableAlarmMatcher`

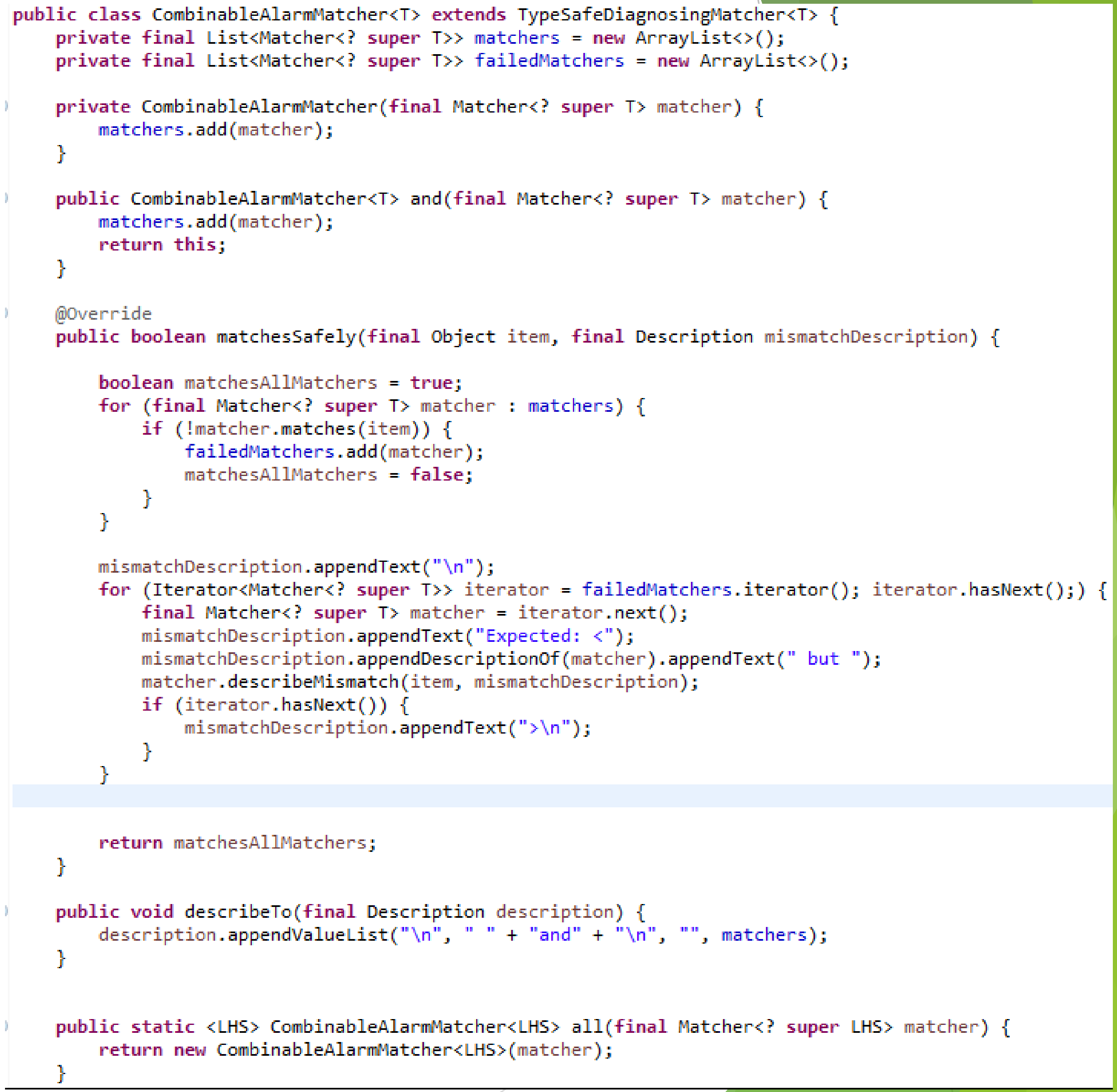


```
alarm2 = new Alarm(alarm1);
alarm2.setModule("FALLO");
alarm2.setErrorCode(2323232);
alarm2.setActiveDescription("HOLA CARACOLA");
alarm3 = new Alarm(alarm1);
alarm3.setSeverity(5666);
assertThat(alarm2, CombinableAlarmMatcher.all(MyMatchers.isModule(alarm1)).and(MyMatchers.isActiveDescription(alarm1)));
```



```
java.lang.AssertionError:
Expected:
<Module should be "STORAGE-ENGINE"> and
<Active Description should be "Storage Engine (DS-group #255): replication channels are down.">
     but:
Expected: <Module should be "STORAGE-ENGINE" but was"FALLO">
Expected: <Active Description should be "Storage Engine (DS-group #255): replication channels are down." but was <es.codeurj.test.Alarm@4f8e5cde>>
Expected: <Module should be "STORAGE-ENGINE" but was"FALLO">
Expected: <Active Description should be "Storage Engine (DS-group #255): replication channels are down." but was <es.codeurj.test.Alarm@4f8e5cde>
```

Combinable AlarmMatcher



```java
public class CombinableAlarmMatcher<T> extends TypeSafeDiagnosingMatcher<T> {
    private final List<Matcher<? super T>> matchers = new ArrayList<>();
    private final List<Matcher<? super T>> failedMatchers = new ArrayList<>();

    private CombinableAlarmMatcher(final Matcher<? super T> matcher) {
        matchers.add(matcher);
    }

    public CombinableAlarmMatcher<T> and(final Matcher<? super T> matcher) {
        matchers.add(matcher);
        return this;
    }

    @Override
    public boolean matchesSafely(final Object item, final Description mismatchDescription) {

        boolean matchesAllMatchers = true;
        for (final Matcher<? super T> matcher : matchers) {
            if (!matcher.matches(item)) {
                failedMatchers.add(matcher);
                matchesAllMatchers = false;
            }
        }

        mismatchDescription.appendText("\n");
        for (Iterator<Matcher<? super T>> iterator = failedMatchers.iterator(); iterator.hasNext();) {
            final Matcher<? super T> matcher = iterator.next();
            mismatchDescription.appendText("Expected: <");
            mismatchDescription.appendDescriptionOf(matcher).appendText(" but ");
            matcher.describeMismatch(item, mismatchDescription);
            if (iterator.hasNext()) {
                mismatchDescription.appendText(">\n");
            }
        }


        return matchesAllMatchers;
    }

    public void describeTo(final Description description) {
        description.appendValueList("\n", " " + "and" + "\n", "", matchers);
    }


    public static <LHS> CombinableAlarmMatcher<LHS> all(final Matcher<? super LHS> matcher) {
        return new CombinableAlarmMatcher<LHS>(matcher);
    }
```