Nummer 4 - 2018

JAVA MAGAZINE

.nl. jug

Onafhankelijk tijdschrift voor de Java-professional

Features

Front-end development zonder runtime errors

Microservices bouwen met Eclipse

Functioneel programmeren met Vavr

Lees dit niet! ☠ 💣 💥

Gegarandeerde manieren om ruzie te krijgen met je collega's.

Ben je dit nu toch aan het lezen? Is de titel niet duidelijk genoeg? Nou ja. Het zou toch ook jammer zijn als de redactie deze pagina's opvulde met een artikel over blockchain. Vooruit dan maar. Hier heb je wat manieren om je collega's te ergeren. Vooral Martijn[0], die vraagt er gewoon om.

Laten we beginnen met iets kleins. Iedereen weet dat we sinds Java 8 lambda's hebben. Maar een lambda is natuurlijk geen échte lambda als je hem niet λ noemt: zie **Listing 0**. Veel Unicode-symbolen zijn namelijk toegestaan in Java identifiers. z̷a̷l̷g̷o̷ kun je gewoon gebruiken als variabelenaam[1]. Het is lastig te typen maar Java vindt het prima.

Unicode escape sequences

Wat de meeste mensen wel weten, is dat je in strings de \uXXXX notatie kunt gebruiken om Unicode-symbolen in te voegen. Wat veel mensen echter hebben verdrongen na hun Certified Programmer examen, is dat dit niet alleen in strings werkt, maar in alle code. De test in **Listing 1** zal slagen.

\u000A is namelijk de Unicode-notatie van een newline en de Java-compiler behandelt hem ook als zodanig. Een eenvoudige manier om flaky tests te fixen dus! Maar als je wilt, dan kun je hele programma's verstoppen in dit soort Unicode escape sequences! De compiler en Maven vinden dat allemaal prima. Zie bijvoorbeeld **Listing 2**. Wie mij tweet met de hashtag #javamagazine en een screenshot met daarin deze snippet én het resultaat, krijgt van mij een gratis retweet!

Reflection

Met reflection kun je trouwens ook leuke trucs uithalen. Maar voordat ik die laat zien even een snelle disclaimer: alle code in dit artikel is getest met Java 11 en is dus helemaal toekomst-vast, maar in oude versies van Java werken de trucs natuurlijk ook gewoon.

Oh en let op: als je illegal reflective access waarschuwingen krijgt,dan kun je die onderdrukken door

```
--add-opens java.base/java.lang=ALL-UNNAMED
```

toe te voegen aan de opstartparameters van de JVM. Het zou jammer zijn als Martijn je betrapt door zoiets opvallends in het log.

Loopy

Listing 3 is een voorbeeld van een mooie reflection-truc. Dit programma print 0, 1, 2, 3, en dan een oneindig aantal 4'en. Dat komt, omdat de meeste JVM's de eerste pakweg 128 boxed Integers cachen. Het stukje reflection-code pakt de 5 uit de cache en verandert het value veld van die Integer in 4. Dus wanneer de loop 4 bereikt en daar 1 bij optelt, pakt de JVM de 5 uit de cache. Maar die is nu 4! Dus i wordt 4. Vervolgens tellen we daar 1 bij op en komen we uit op 5. De JVM haalt dat uit de cache, ziet een 4, en print die. Vervolgens tellen we daar 1 bij op... ad nauseam.

Jan Ouwens is senior Scala-ontwikkelaar bij Codestar. Hij houdt zich graag bezig met Java, Scala, softwarekwaliteit en zijn open source-tool EqualsVerifier.

```
var ints = List.of(1, 2, 3);
Consumer<Integer> λ = i -> System.out.println(i);
ints.forEach(λ);
```

Listing 0: λ

```
@Test
public void falseIsTrue() {
    // Flaky test! \u000A if (true) return;
    assertEquals("false", "true");
}
```

Listing 1: flaky test

```
\u0070\u0075\u0062\u006c\u0069\u0063\u0020\u0063\u006c\u0061\u0073\u0073\u0020\u0058\u007b
\u0070\u0075\u0062\u006c\u0069\u0063\u0020\u0073\u0074\u0061\u0074\u0069\u0063
\u0020\u0076\u006f\u0069\u0064\u0020\u006d\u0061\u0069\u006e\u0028
\u0053\u0074\u0072\u0069\u006e\u0067\u005b\u005d\u0061\u0029\u0074\u0068\u0072\u006f\u0077\u0073
\u0020\u0045\u0078\u0063\u0065\u0070\u0074\u0069\u006f\u006e\u007b
\u006a\u0061\u0076\u0061\u002e\u0061\u0077\u0074\u002e\u0044\u0065\u0073\u006b\u0074\u006f\u0070
\u002e\u0067\u0065\u0074\u0044\u0065\u0073\u006b\u0074\u006f\u0070\u0028\u0029
\u002e\u0062\u0072\u006f\u0077\u0073\u0065\u0028\u006e\u0065\u0077\u0020
\u006a\u0061\u0076\u0061\u002e\u006e\u0065\u0074\u002e\u0055\u0052\u0049\u0028
\u0022\u0068\u0074\u0074\u0070\u003a\u002f\u002f\u0079\u006f\u0075\u0074\u0075\u002e\u0062\u0065\u002f\u0022\u002b
\u0022\u0064\u0051\u0077\u0034\u0077\u0039\u0057\u0067\u0058\u0063\u0051\u0022\u0029\u0029\u003b\u007d\u007d
```

Listing 2: Unicode escapes. Let op: het maakt uit waar de regeleindes staan! Noem het bestand X.java en run het met javac X.java && java X.

Als je zo'n vervelende collega hebt die het nodig vindt om altijd boxed primitives te gebruiken in plaats van gewone (ja, zulke mensen bestaan écht), dan kun je heel makkelijk zijn unit tests om zeep helpen door ergens in een ongerelateerde test de cache op deze manier te vervuilen. 💣!

Constructors zijn lastig

Als je toevallig Objenesis[2] op je classpath (of module path!) hebt staan, dan kun je nog wat verder gaan. Objenesis is een library die objecten kan instantiëren zonder de constructor aan te roepen. Dit kan handig zijn als de constructor private is, of heel veel lastige parameters vereist, of gewoon netjes z'n precondities checkt. Bah. Objenesis wordt daarom veel gebruikt in mocking frameworks. De kans dat je het al hebt, is dus best groot! Heb je altijd al Void willen instantiëren? Met Objenesis kan het: zie **Listing 4**.

Enums

Objecten instantiëren kan Objenesis zó goed dat het zelfs werkt met enums! Met enums? Maar zei Joshua Bloch in Effective Java niet dat de JVM een "ironclad guarantee" geeft tegen "sophisticated [...] reflection attacks" om stiekem enums te instantiëren[3], en dat ze daarom zo geschikt zijn voor het schrijven van singletons? Ja, dat zei hij inderdaad.

```
public class Loopy {
    public static void main(String... args) throws Exception {
        var field = Integer.class.getDeclaredField("value");
        field.setAccessible(true);
        field.set(5, 4);

        for (Integer i = 0; i < 10; i++) {
            System.out.println(i);
        }
    }
}
```

Listing 3: loopy

```
var objenesis = new ObjenesisStd();
Void avoid = objenesis.newInstance(Void.class);
System.out.println(avoid);
```

Listing 4: void

```
public static <T> void setPrivateFieldValue(Class<T> type, String
fieldName, T receiver, Object newValue)
        throws NoSuchFieldException, IllegalAccessException {
    var field = type.getDeclaredField(fieldName);
    field.setAccessible(true);
    field.set(receiver, newValue);
}

enum Singleton { INSTANCE }

Singleton copy = objenesis.newInstance(Singleton.class);
setPrivateFieldValue(Enum.class, "ordinal", copy, 0);
setPrivateFieldValue(Enum.class, "name", copy, "INSTANCE");
```

Listing 5: singleton

Sterker nog: de Java Language Specification zegt het ook[4].
Toch werkt het. Kijk maar naar **Listing 5**: kinderspel. Daar gaat je singleton! En we kunnen nog verder gaan. Vanaf hier is het namelijk een kleine stap om nieuwe elementen toe te voegen aan je enum; zie **Listing 6.** Nadat we de ordinal en name hebben ingevuld, voegen we ook ons nieuwe element toe aan Kleuren. values(), die zijn waarde ontleent aan het private veld $VALUES in de enum. Daar moeten we even de final modifier van wegvegen, en presto: als we nu itereren over Kleuren, dan krijgen we alle kleuren. Dus ook de jokers. 🃏

▸▸

Agents

Met Java Agents kun je Martijn op een heel andere manier in verwarring brengen. Agents zijn gewone Java-applicaties die zich kunnen bevestigen aan een draaiende JVM, om vervolgens code te injecteren. Ze zijn een standaardfeature in Java. Op het moment dat een agent bevestigd wordt aan een Java-proces, wordt de methode agentmain (vergelijkbaar met de welbekende public static void main) aangeroepen, en die heeft vervolgens vrij spel.

Ho eens even, zal je nu denken. Dat is best wel nasty! Hoezo zit dat standaard in Java? Nou, er zijn heel veel Java Agents met legitieme use cases. Profilers, debuggers, monitoring tools: zomaar drie toepassingen die onmogelijk zouden zijn zonder agents. Maar daar zijn we nu niet in geïnteresseerd. Wij willen gewoon Martijn pesten. Dat doen we door System.currentTimeMillis() onklaar te maken in een draaiend proces.

Stel dat je werkt aan een high-speed trading system of een planner voor dienstregelingen van treinen. Dan is het belangrijk om goed te weten wat de tijd is. We vereenvoudigen die applicatie tot een programma dat simpelweg elke seconde print hoe laat het is. Zie **Listing 7.**

Met een agent kunnen we ervoor zorgen dat dit programma elke seconde het getal 1337 print. Daarvoor gebruiken we de open source library Byte Buddy[5], die het gemakkelijk maakt om bytecode te genereren. Ook voor Byte Buddy is de kans groot dat het al op je classpath staat, omdat het een transitieve dependency is van veelgebruikte tools, zoals Hibernate en Jackson.

Onze agent kun je zien in **Listing 8**. Hij print een korte tekst en gaat dan aan de slag. De transformer (die een lambda is en daarom uiteraard λ heet) zoekt naar een methode met de naam currentTimeMillis, onderschept die, en laat hem 1337 teruggeven.

De AgentBuilder neemt die transformer en koppelt hem aan java.lang.System. Vervolgens installeert hij de agent op instrumentation, een variabele die verwijst naar het draaiende proces dat we willen slopen.

Compileer deze klasse en stop hem in een jar file, bijvoorbeeld attack.jar. Zorg dat het MANIFEST.MF-bestand in die jar op zijn minst dezelfde regels bevat als **Listing 9**.

Het laatste stukje van de puzzel is het daadwerkelijk koppelen van de agent aan het proces. Zorg dat je de PID bij de hand hebt. Victim print die netjes uit, maar op een POSIX-systeem kun je die bijvoorbeeld ook gewoon opvragen met ps op de commandline. Met Byte Buddy is het nu een oneliner die je eventueel in JShell kunt uitvoeren:

MET REFLECTION KUN JE TROUWENS OOK LEUKE TRUCS UITHALEN

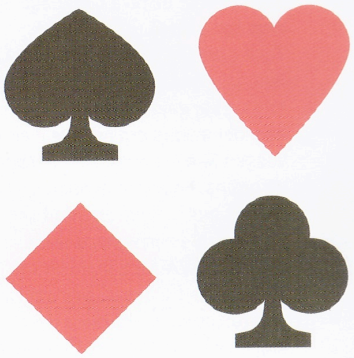
```
ByteBuddyAgent.attach("/pad/naar/attack.jar", pid);
```

```
enum Kleuren { SCHOPPEN, HARTEN, KLAVEREN, RUITEN };

Kleuren jokers = objenesis.newInstance(Kleuren.class);
setPrivateFieldValue(Enum.class, "ordinal", jokers, 4);
setPrivateFieldValue(Enum.class, "name", jokers, "JOKERS");

Field valuesField = Kleuren.class.getDeclaredField("$VALUES");
valuesField.setAccessible(true);

setPrivateFieldValue(Field.class, "modifiers", valuesField, values-
Field.getModifiers() & ~Modifier.FINAL);
Kleuren[] alleKleuren = { SCHOPPEN, HARTEN, KLAVEREN, RUITEN, jokers
};
valuesField.set(null, alleKleuren);

for (Kleuren kleur : Kleuren.values()) {
    System.out.println(kleur);
}
```

Listing 6: enum

```
public class Victim {
    public static void main(String... args) throws Exception {
        System.out.println("Current PID: " + ProcessHandle.current().
pid());
        while (true) {
            // Superbelangrijk dat dit ↓ klopt!
            System.out.println(System.currentTimeMillis());
            Thread.sleep(1000);
        }
    }
}
```

Listing 7: victim



```
public class Attack {

    public static void agentmain(String arguments, Instrumentation instrumentation) {
        System.out.println("Let's install some malicious code...  ¯\( ಠ )/¯ ");

        AgentBuilder.Transformer λ =
                (builder, typeDescription, classLoader, javaModule) -> builder
                                .method(named("currentTimeMillis"))
                                .intercept(value(1337));

        new AgentBuilder.Default()
                .ignore(none())
                .disableClassFormatChanges()
                .with(AgentBuilder.RedefinitionStrategy.RETRANSFORMATION)
                .type(named("java.lang.System"))
                .transform(λ)
                .installOn(instrumentation);
    }
}
```

Listing 8: attack

ALS JE TOEVALLIG OBJENESIS OP JE CLASSPATH (OF MODULE PATH) HEBT STAAN, DAN KUN JE NOG WAT VERDER GAAN

```
Agent-Class: de.package.naam.die.je.gebruikt.Attack
Can-Retransform-Classes: true
```

Listing 9: manifest

De output van Victim ziet er dan ongeveer uit als **Listing 10**. En het leuke is: als je toegang hebt tot de productieserver, kun je dit trucje ook daar toepassen. Martijn weet niet wat hem overkomt!👆

```
Current PID: 63157
1528464432723
1528464433725
1528464434726
Let's install some malicious code...  ¯\( ಠ )/¯
1528464435728
1337
1337
1337
1337
1337
1337
```

Listing 10: output van victim

Awareness

Je kunt ook voor de double bluf gaan: maak ruzie met collega's die ook begonnen zijn dit artikel te lezen, maar halverwege zijn afgeleid om een mooie hack in dit artikel te proberen en daarna nooit meer de rest van dit artikel hebben gelezen. Deze mensen kun je irriteren door ervoor te zorgen dat die hack niet werkt. Zo kun je de meeste Unicode-trucs gewoon tegengaan met een paar strategisch geplaatste SonarQube-regels.

De reflection-hacks zijn lastiger, maar ook die kun je eenvoudig tegengaan. Daarvoor kun je een security manager gebruiken. Voeg de volgende regel code ergens toe aan je programma:

```
System.setSecurityManager(new
SecurityManager());
```

Je hoeft hiervoor niet eens iets te importeren. De JVM zal nu een SecurityException gooien bij elke reflection-hack. En trouwens ook bij het gebruik van Spring, want Spring is in feite ook niet veel meer dan een grote brij aan vieze, vieze reflection-hacks.😈

Wat je ook kan doen, is je codebase modulariseren en de JVM-parameter --illegal-access=deny gebruiken. Dit zorgt ervoor dat je alleen nog reflection kan toepassen op klassen binnen dezelfde module. Aangezien alle Java API klassen in een andere modulen zitten (namelijk java.base), kun je daar niks meer mee doen. En zeg nou zelf: een gedwongen upgrade naar Java 11 is ook een mooie manier om je collega's te ergeren.

Conclusie

Voor degenen die vroeger op school hebben geleerd om eerst de inleiding en daarna de conclusie van een artikel te lezen en pas daarna te beslissen of ze de rest willen lezen: move along now, nothing to see here! Je bent te laat, het artikel is al afgelopen. ■

VOETNOTEN

[0] Als je nu Martijn heet, kijk je wel even raar op hè? Maar in de praktijk heeft bijna iedereen wel een collega die zo heet, da's algemeen bekend.
[1] **https://www.zalgotextgenerator.com/**
[2] **http://objenesis.org**
[3] Joshua Bloch, Effective Java, 2nd én 3rd Edition, Item 3. Ik heb speciaal voor dit artikel de 3e editie aangeschaft om te zien of de quote er nog in stond. De quote stond er nog in.
[4] Section 8.9: **https://docs.oracle.com/javase/specs/jls/se10/html/jls-8.html#jls-8.9**
[5] **http://bytebuddy.net**