



Знакомство с фаззерами

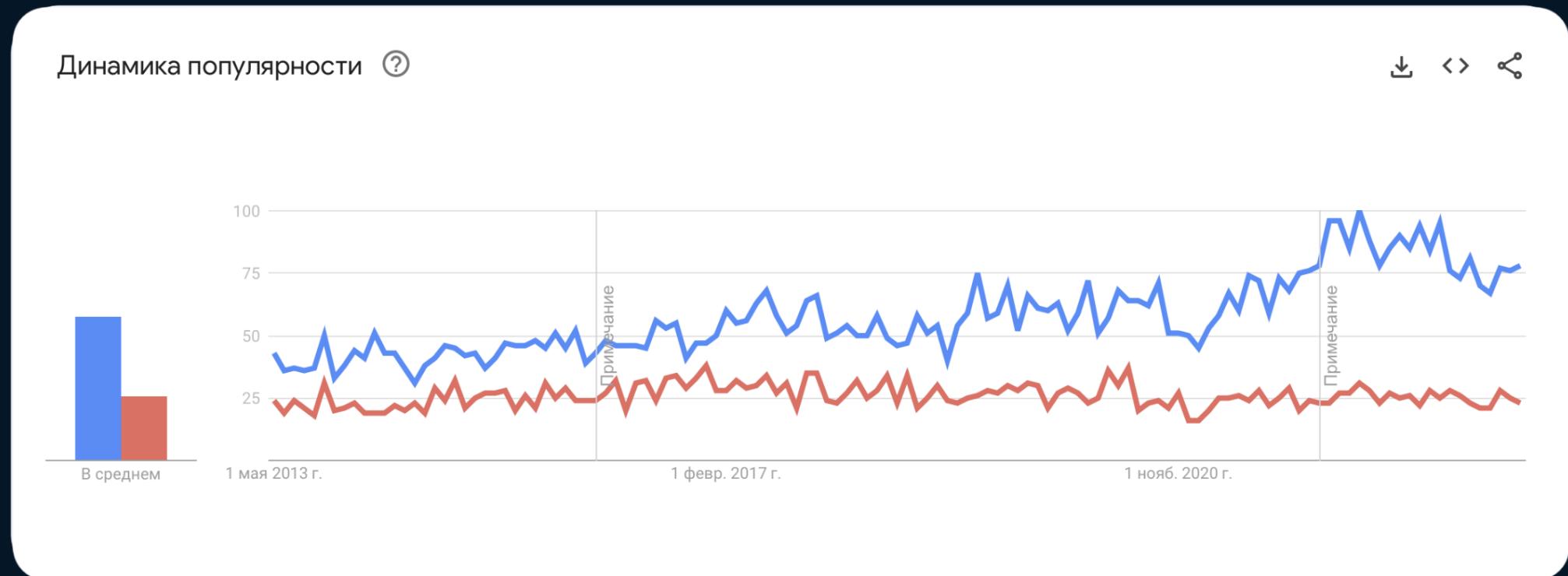


Максим Пелевин

- Разработчик на Java, Kotlin более 10 лет
- **Интересы:** динамический анализ кода
- **Хобби:** преподавание в университете

 markoutte

Популярность запросов Fuzzing и Fuzzer



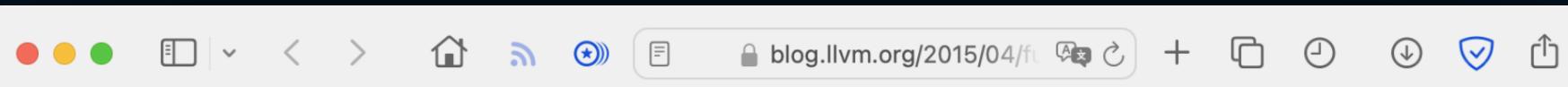
Фаззинг — техника тестирования, когда на вход тестируемой программы подаются неожиданные, неправильные или случайные данные в надежде, что всё сломается

Зачем? Ошибки есть ошибки, а разработчики не так часто тестируют свой код на ошибки несвязанные со спецификацией

Фаззер — программа, которая фаззит

american fuzzy lop 1.86b (test)

process timing		overall results
run time : 0 days, 0 hrs, 0 min, 2 sec		cycles done : 0
last new path : none seen yet		total paths : 1
last uniq crash : 0 days, 0 hrs, 0 min, 2 sec		uniq crashes : 1
last uniq hang : none seen yet		uniq hangs : 0
cycle progress	map coverage	
now processing : 0 (0.00%)	map density : 2 (0.00%)	
paths timed out : 0 (0.00%)	count coverage : 1.00 bits/tuple	
stage progress	findings in depth	
now trying : havoc	favored paths : 1 (100.00%)	
stage execs : 1464/5000 (29.28%)	new edges on : 1 (100.00%)	
total execs : 1697	total crashes : 39 (1 unique)	
exec speed : 626.5/sec	total hangs : 0 (0 unique)	
fuzzing strategy yields	path geometry	
bit flips : 0/16, 1/15, 0/13	levels : 1	
byte flips : 0/2, 0/1, 0/0	pending : 1	
arithmetics : 0/112, 0/25, 0/0	pend fav : 1	
known ints : 0/10, 0/28, 0/0	own finds : 0	
dictionary : 0/0, 0/0, 0/0	imported : n/a	
havoc : 0/0, 0/0	variable : 0	
trim : n/a, 0.00%		
		[cpu: 92%]



The LLVM Project Blog

LLVM Project News and Details from the Trenches



[About](#) [Posts](#) [Tags](#) [llvm.org](#)

Simple guided fuzzing for libraries using LLVM's new libFuzzer

By Kostya Serebryany

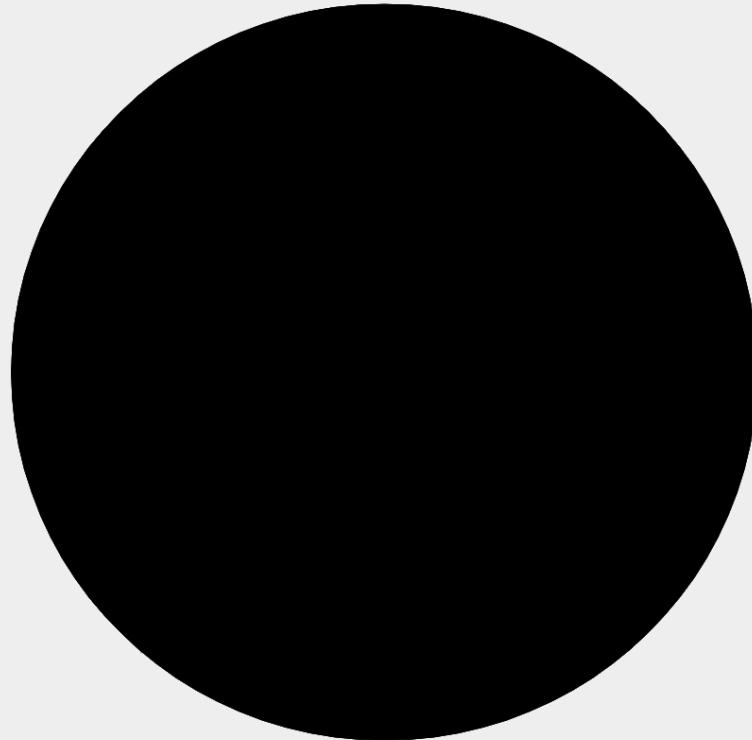
Apr 9, 2015

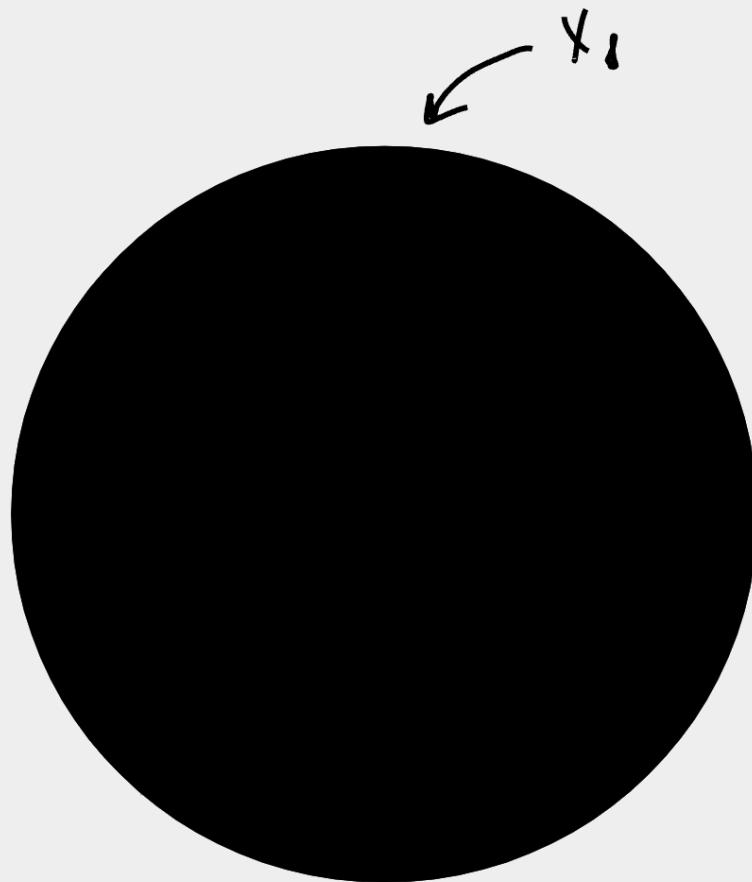
[#sanitizer](#), [#testing](#), [#Clang](#)

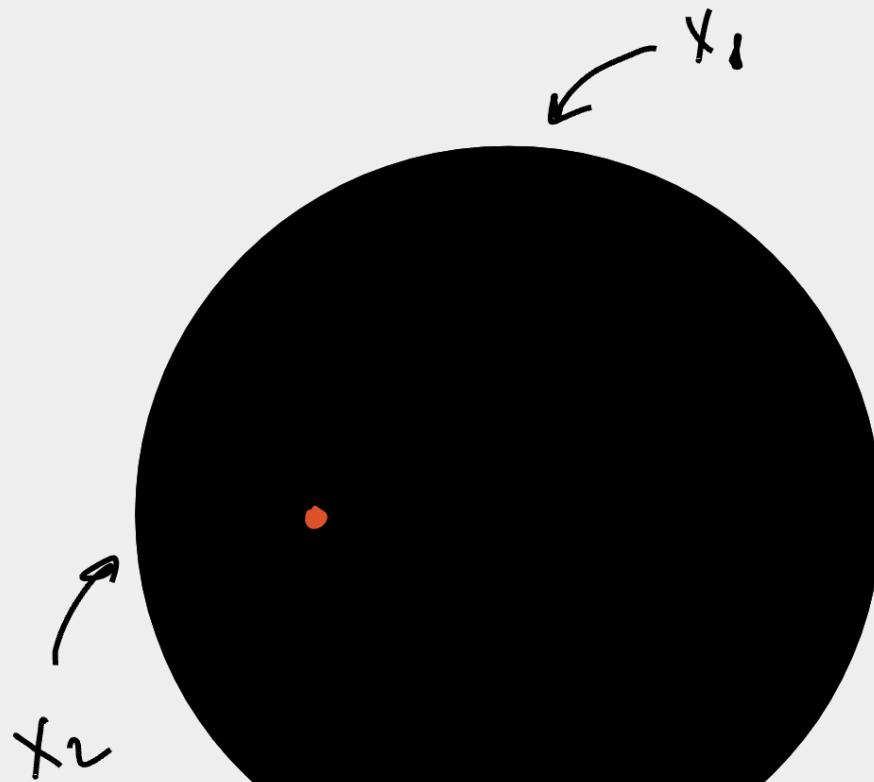
3 minute read

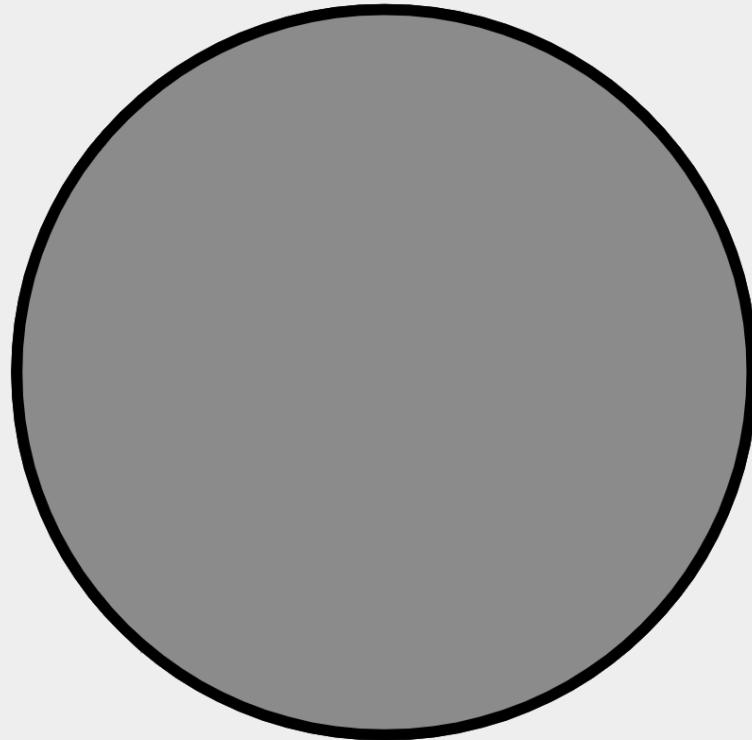
Fuzzing (or [fuzz testing](#)) is becoming increasingly popular. Fuzzing Clang and fuzzing *with* Clang is not new: Clang-based [AddressSanitizer](#) has been used for fuzz-testing the Chrome browser for [several years](#) and Clang itself has been extensively fuzzed using [csmith](#) and, more recently, using [AFL](#). Now we've closed the loop and started to fuzz parts of LLVM (including Clang) using LLVM itself.

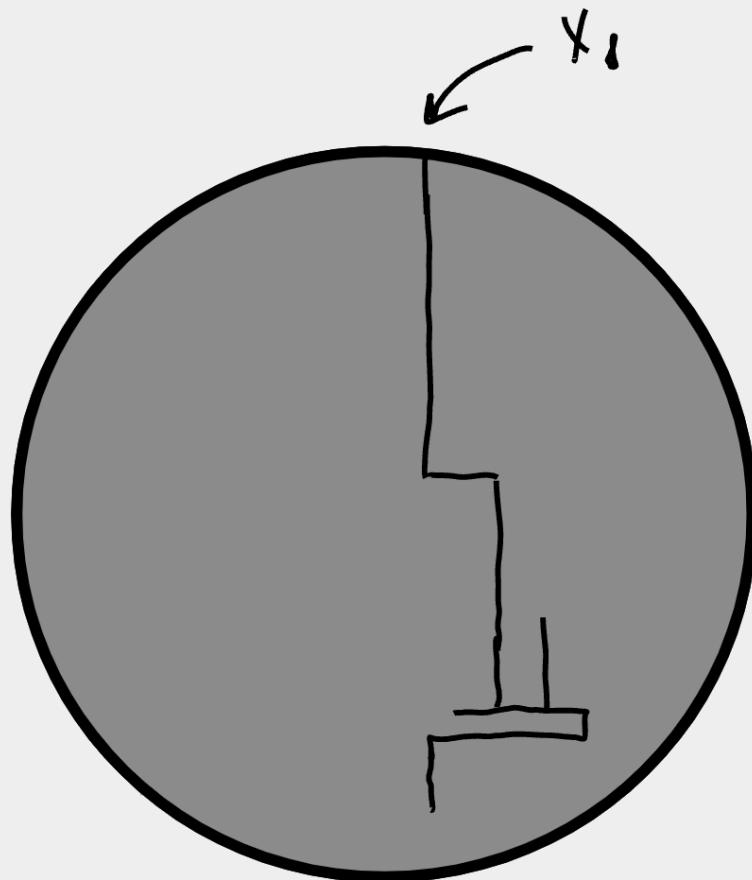
[LibFuzzer](#), recently added to the LLVM tree, is a library for in-process fuzzing that uses [Sanitizer](#).

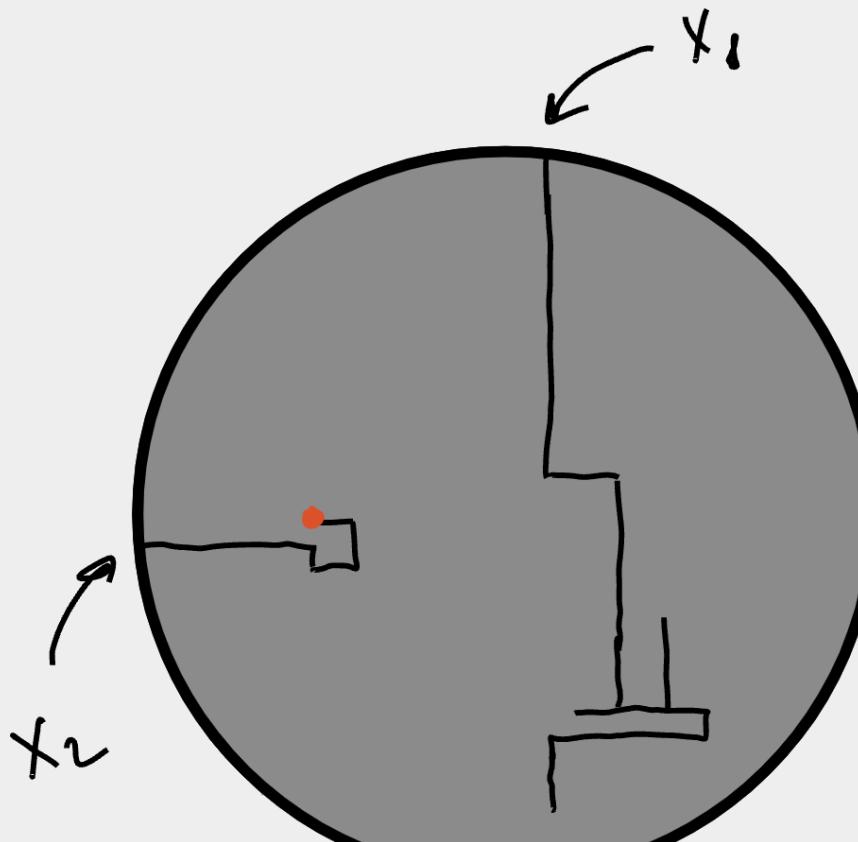


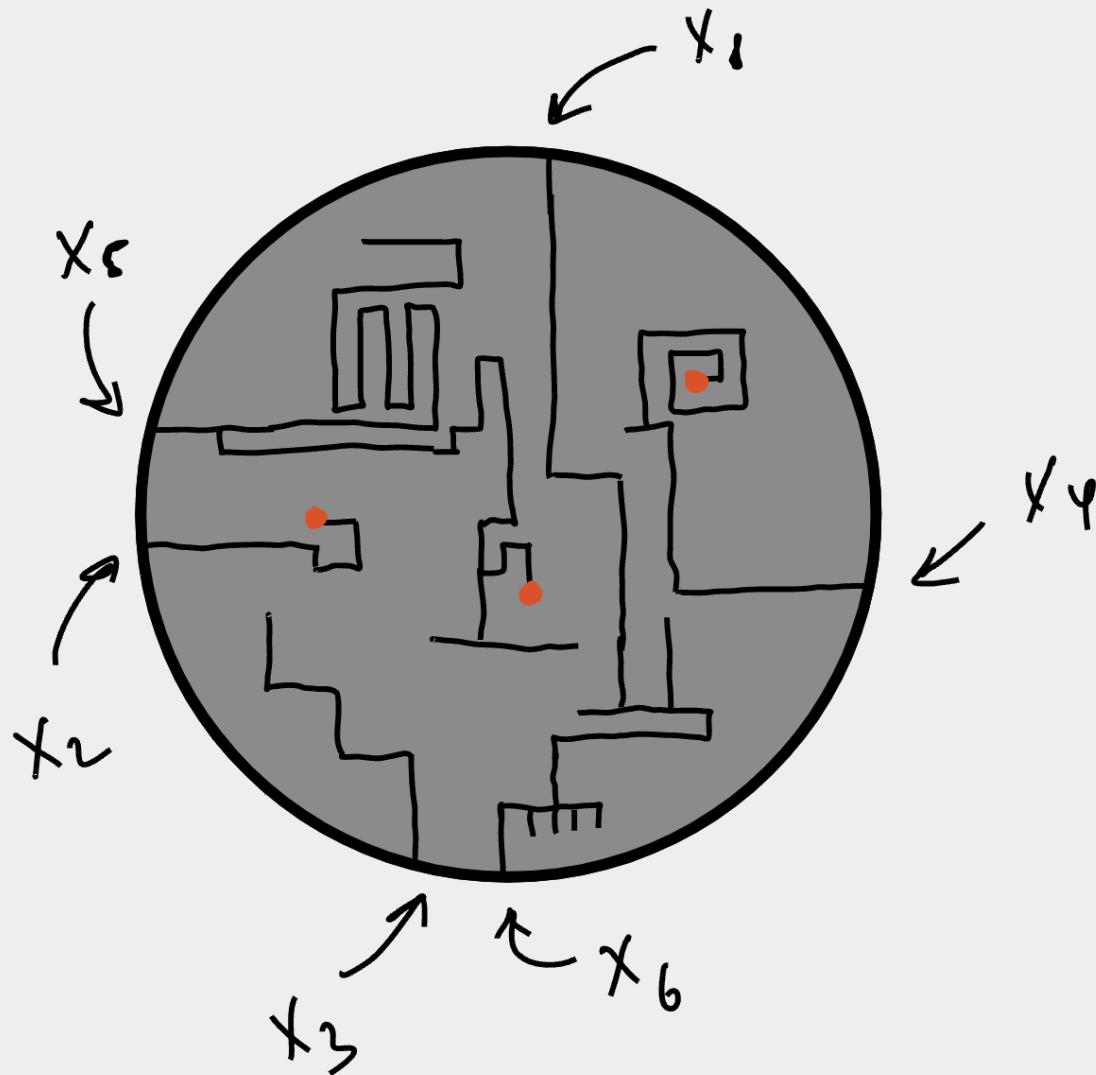


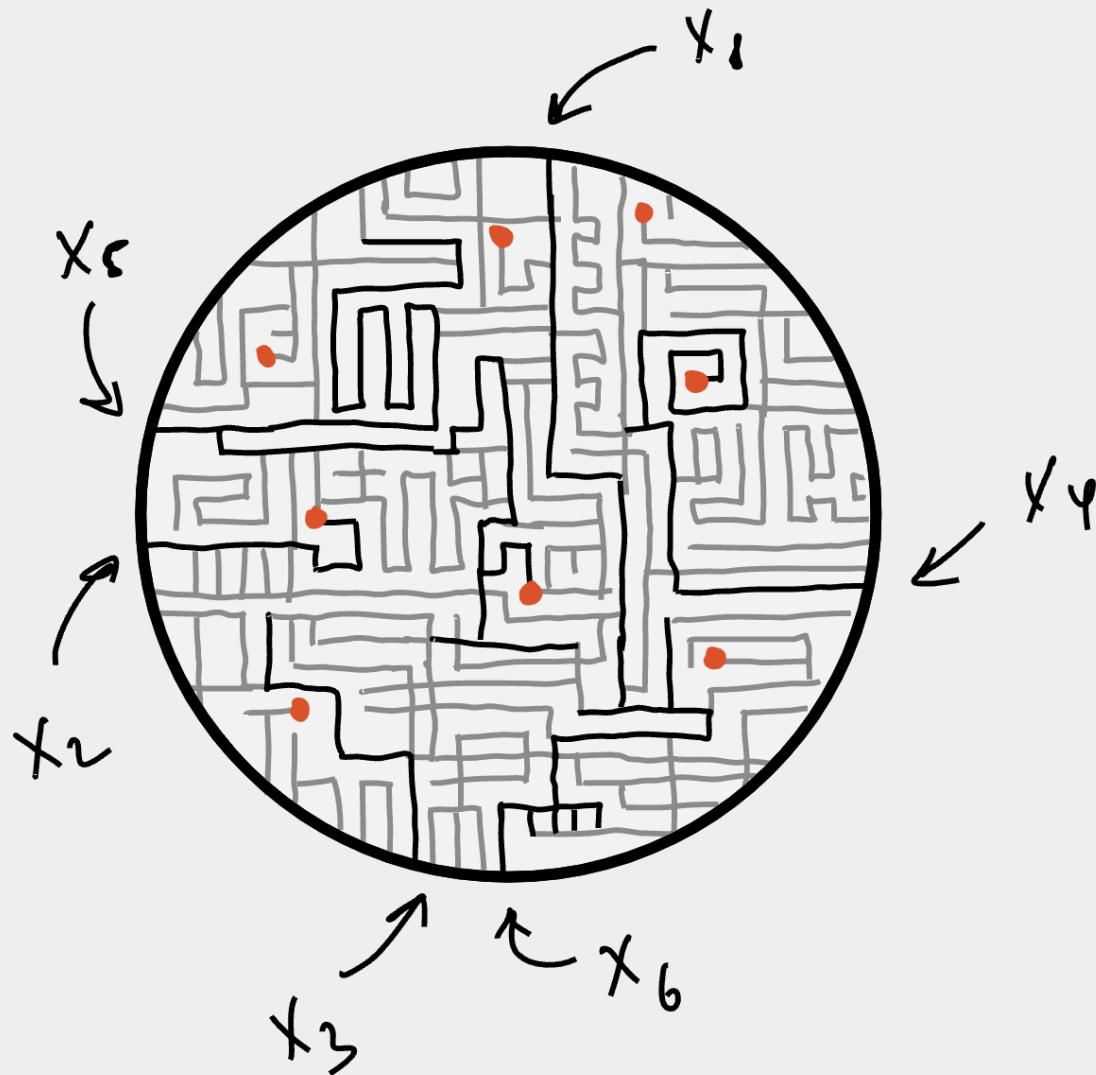


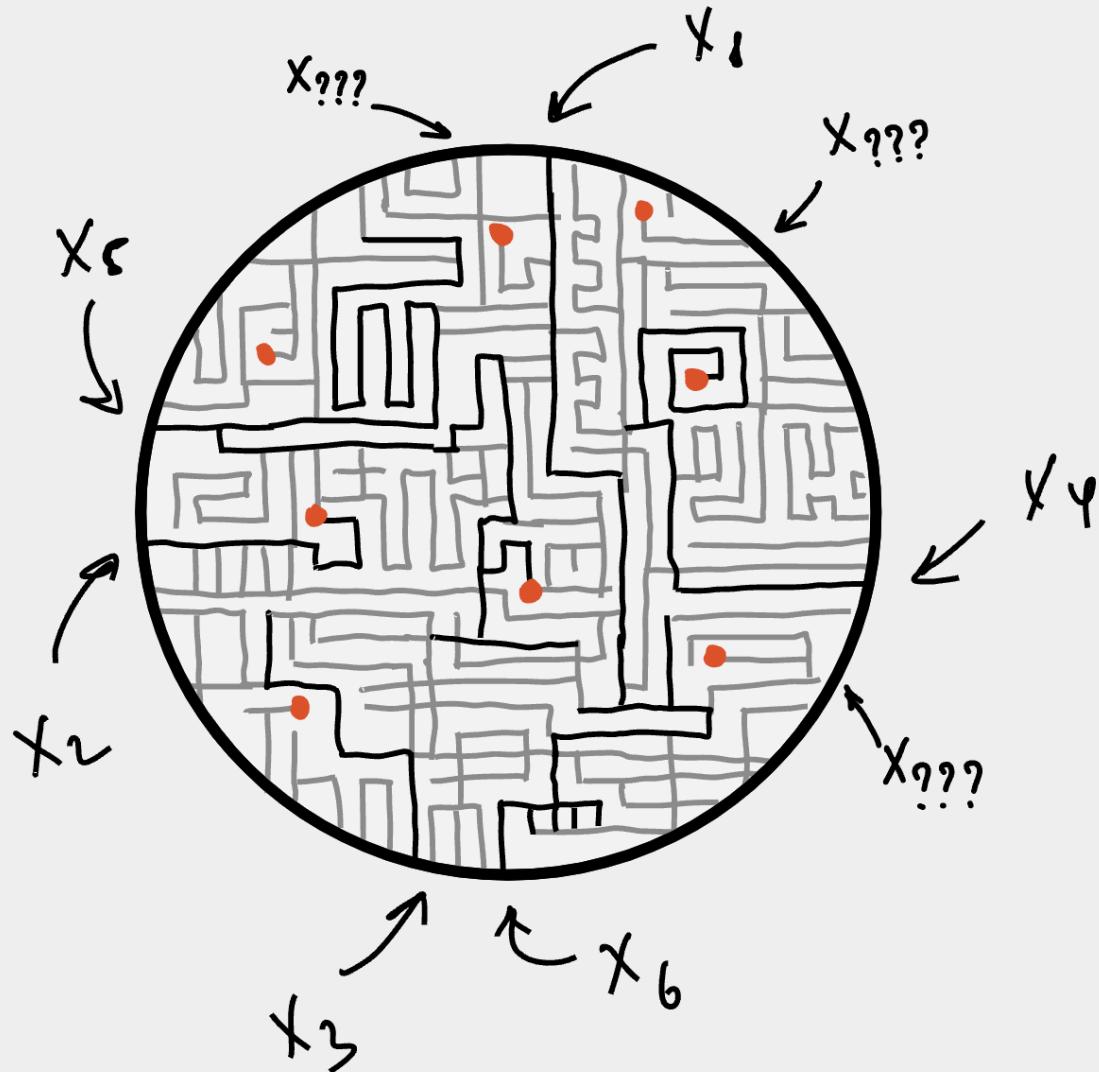


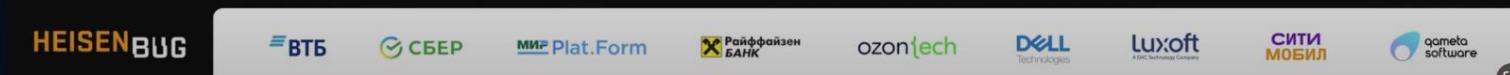
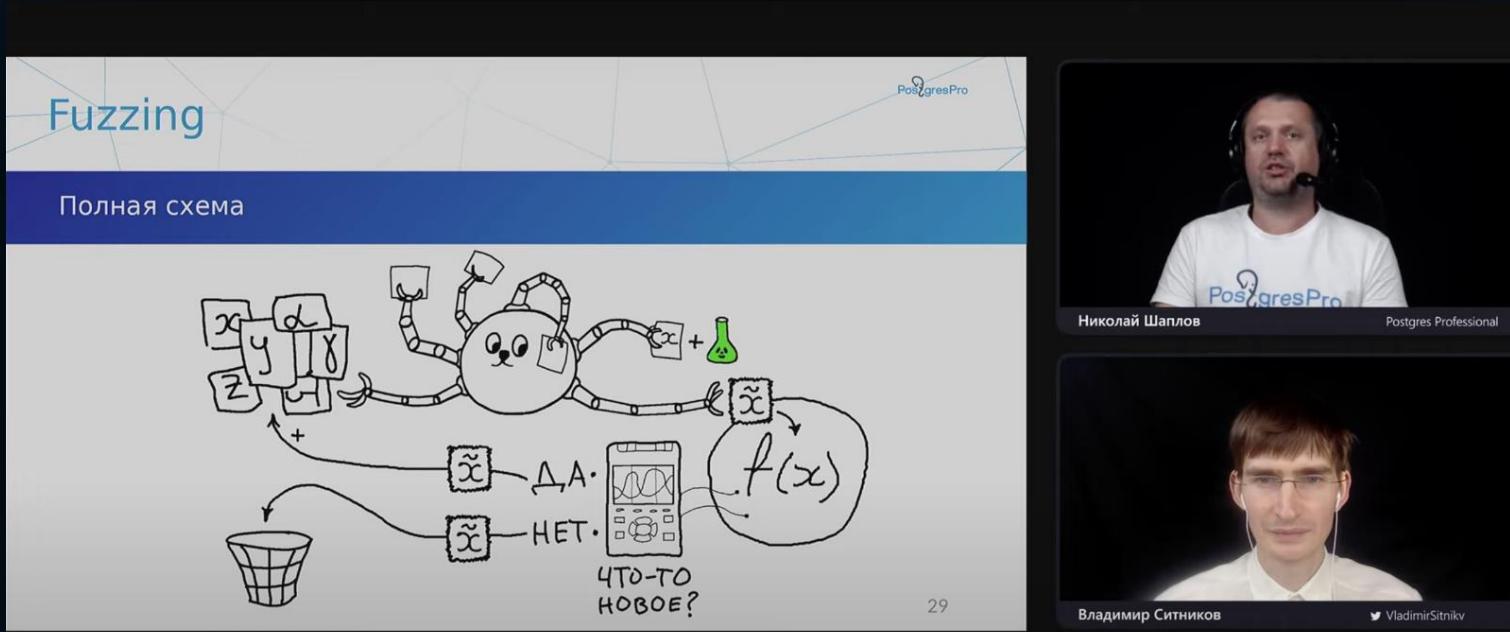














```
3 import java.util.Arrays;
4
5 public final class TimSort {
6
...
7     static int RUN = 32;
8
9     // this function sorts array from left index to
10    // to right index which is of size atmost THRESHOLD
11    1 usage
12    public static void insertionSort(int[] arr, int left, int right) {
13        for (int i = left + 1; i <= right; i++) {
14            int temp = arr[i];
15            int j = i - 1;
16            while (j >= 0 && arr[j] > temp && j >= left) {
17                arr[j + 1] = arr[j];
18                j--;
19            }
20            arr[j + 1] = temp;
21        }
22    }
23}
```

The screenshot shows a Java code editor interface with the following details:

- Title Bar:** Shows the project name "Joker2023", the package "main", and the current file "TimSort.java".
- Status Bar:** Shows the file path "src/main/java/markoutte/examples/TimSort.java", the line number "12", and the total duration "0:00 / 1:13".
- Code Editor:** Displays the source code for the `TimSort` class. The code includes imports for `java.util.Arrays`, a class definition, a static variable `RUN`, and a `insertionSort` method. The `insertionSort` method uses a for loop to iterate from `left + 1` to `right`. Inside the loop, it uses a while loop to shift elements to the right until it finds the correct position for the current element `temp` (at index `i`). The editor highlights the word `temp` in red.
- Syntax Indicators:** On the left margin, there are vertical lines with icons corresponding to the code lines:
 - Line 3: File icon
 - Line 4: None
 - Line 5: Open folder icon (with a green triangle)
 - Line 6: Ellipsis icon
 - Line 7: None
 - Line 8: None
 - Line 9: None
 - Line 10: None
 - Line 11: Bug icon
 - Line 12: None
 - Line 13: None
 - Line 14: None
 - Line 15: None
 - Line 16: None
 - Line 17: None
 - Line 18: None
 - Line 19: None
 - Line 20: None
 - Line 21: None
 - Line 22: None
 - Line 23: None
- Status Area:** At the top right, there are several status icons: a warning icon (yellow triangle), a checkmark icon (green checkmark), a double arrow icon, a bell icon, a mouse icon, and a gear icon.

```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)
}
```

```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)
}
```

```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)
}
```

```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)
}
```

```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)
}
```

```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)
}
```

```
addOption("c", "class", true, "Java class fully qualified name")
addOption("m", "method", true, "Method to be tested")
addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
addOption("s", "seed", true, "The source of randomness")
}

val parser = DefaultParser().parse(options, args)
val className = parser.getOptionValue("class")
val methodName = parser.getOptionValue("method")
val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
val random = Random(seed)

println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
```

```
addOption("m", "method", true, "Method to be tested")
addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
addOption("s", "seed", true, "The source of randomness")
}

val parser = DefaultParser().parse(options, args)
val className = parser.getOptionValue("class")
val methodName = parser.getOptionValue("method")
val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
val random = Random(seed)

println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
```

```
addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
addOption("s", "seed", true, "The source of randomness")
}

val parser = DefaultParser().parse(options, args)
val className = parser.getOptionValue("class")
val methodName = parser.getOptionValue("method")
val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
val random = Random(seed)

println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
}
```

```
val parser = DefaultParser().parse(options, args)
val className = parser.getOptionValue("class")
val methodName = parser.getOptionValue("method")
val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
val random = Random(seed)

println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
}
```

```
println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
}
```



```
val className = parser.getOptionValue("class")
val methodName = parser.getOptionValue("method")
val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
val random = Random(seed)

println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

val javaMethod = try {
    loadJavaMethod(className, methodName)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
}
```

```
println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

val javaMethod = try {
    loadJavaMethod(className, methodName)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
}

fun loadJavaMethod(className: String, methodName: String): Method {
    val classLoader = ClassLoader.getSystemClassLoader()
    val javaClass = classLoader.loadClass(className)
    val javaMethod = javaClass.getDeclaredMethods().first {
        "${it.name}(${it.parameterTypes.joinToString(",")}) {
            it.name → it.typeName
    }
}
```

```
val start = System.nanoTime()

val javaMethod = try {
    loadJavaMethod(className, methodName)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    try {
        javaMethod.invoke(null, *inputValues)
    } catch (e: InvocationTargetException) {

    }
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")

fun loadJavaMethod(className: String, methodName: String): Method {
    val classLoader = ClassLoader.getSystemClassLoader()
    val javaClass = classLoader.loadClass(className)
    val javaMethod = javaClass.getDeclaredMethods first {
```

```
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    try {
        javaMethod.invoke(null, *inputValues)
    } catch (e: InvocationTargetException) {
        if (errors.add(e.targetException::class.qualifiedName!!)) {
            val errorMessage = e.targetException::class.simpleName
            println("New error found: $errorMessage")
            val path = Paths.get("report$errorMessage.txt")
            Files.write(path, listOf(
                "${e.targetException.stackTraceToString()}\n",
                "${javaMethod.name}${inputValues.contentDeepToString()}\n",
                "${buffer.contentToString()}\n",
            ))
            Files.write(path, buffer, StandardOpenOption.APPEND)
            println("Saved to: ${path.fileName}")
        }
    }
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
```

```
loadJavaMethod(className, methodName)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    try {
        javaMethod.invoke(null, *inputValues)
    } catch (e: InvocationTargetException) {
        if (errors.add(e.targetException::class.qualifiedName!!)) {
            val errorMessage = e.targetException::class.simpleName
            println("New error found: $errorMessage")
            val path = Paths.get("report$errorMessage.txt")
            Files.write(path, listOf(
                "${e.targetException.stackTraceToString()}\n",
                "${javaMethod.name}${inputValues.contentDeepToString()}\n",
                "${buffer.contentToString()}\n",
            ))
            Files.write(path, buffer, StandardOpenOption.APPEND)
            println("Saved to: ${path.fileName}")
        }
    }
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
```

```
while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    try {
        javaMethod.invoke(null, *inputValues)
    } catch (e: InvocationTargetException) {
        if (errors.add(e.targetException::class.qualifiedName!!)) {
            val errorMessage = e.targetException::class.simpleName
            println("New error found: $errorMessage")
            val path = Paths.get("report$errorMessage.txt")
            Files.write(path, listOf(
                "${e.targetException.stackTraceToString()}\n",
                "${javaMethod.name}${inputValues.contentDeepToString()}\n",
                "${buffer.contentToString()}\n",
            ))
            Files.write(path, buffer, StandardOpenOption.APPEND)
            println("Saved to: ${path.fileName}")
        }
    }
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
}
```

```
loadJavaMethod(className, methodName)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
    try {
        javaMethod.invoke(null, *inputValues)
    } catch (e: InvocationTargetException) {
        if (errors.add(e.targetException::class.qualifiedName!!)) {
            val errorMessage = e.targetException::class.simpleName
            println("New error found: $errorMessage")
            val path = Paths.get("report$errorMessage.txt")
            Files.write(path, listOf(
                "${e.targetException.stackTraceToString()}\n",
                "$inputValuesString\n",
                "${buffer.contentToString()}\n",
            ))
            Files.write(path, buffer, StandardOpenOption.APPEND)
            println("Saved to: ${path.fileName}")
        }
    }
}

println("Errors found: ${errors.size}")
```

```
val javaMethod = javaClass.declaredMethods.first {
    "${it.name}(${it.parameterTypes.joinToString(",") {
        c → c.typeName
    }})" == methodName
}
return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
    val parameterTypes = method.parameterTypes
    return Array(parameterTypes.size) {
        when (parameterTypes[it]) {
            IntArray::class.java → IntArray(buffer.get().toUByte().toInt()) {
                buffer.get().toInt()
            }
            else → error("Cannot create value of type ${parameterTypes[it]}")
        }
    }
}
```

```
        }

    }

    println("Errors found: ${errors.size}")
    println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
        System.nanoTime() - start
    )} ms")
}

fun loadJavaMethod(className: String, methodName: String): Method {
    val classLoader = ClassLoader.getSystemClassLoader()
    val javaClass = classLoader.loadClass(className)
    val javaMethod = javaClass.declaredMethods.first {
        "${it.name}(${it.parameterTypes.joinToString(",")} {
            c → c.typeName
        })" == methodName
    }
    return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
    val parameterTypes = method.parameterTypes
    return Array(parameterTypes.size) {
        when (parameterTypes[it]) {
            IntArray::class.java → IntArray(buffer.get().toUByte().toInt()) {
                buffer.get().toInt()
            }
            else → error("Cannot create value of type ${parameterTypes[it]}")
        }
    }
}
```

● ● ● ✎*2

-zsh

```
markoutte@Aquarius libs % java -jar fuzzer-timsort.jar -c me.markoutte.examples.  
TimSort -m "timSort(int[])"█
```

▶ 0:29 / 1:34



● ● ● ✎*2

-zsh

```
markoutte@Aquarius libs % java -jar fuzzer-timsort.jar -c me.markoutte.examples.TimSort -m "timSort(int[])"  
Running: me.markoutte.examples.TimSort.timSort(int[]) with seed = -1016803564  
New error found: NegativeArraySizeException  
Saved to: reportNegativeArraySizeException.txt  
Errors found: 1  
Time elapsed: 10000 ms  
markoutte@Aquarius libs % cat reportNegativeArraySizeException.txt
```

▶ 0:52 / 1:34



```
java.lang.NegativeArraySizeException: -22
    at me.markoutte.examples.TimSort.merge(TimSort.java:30)
    at me.markoutte.examples.TimSort.timSort(TimSort.java:84)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at java.base/jdk.internal.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:77)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at me.markoutte.joker.timsort.MainKt.main(Main.kt:45)
```

```
timSort: [[-56, 27, -50, -115, -126, -24, -86, 114, -36, 118, -70, 77, 120, 89,
-64, -118, -82, -29, 96, -113, -59, 105, 65, 22, 23, -34, 76, 17, -19, 109, -26,
85, -50, -96, -7, -51, 97, -73, 114, -97, 54, -66, 2, -18, 95, -58, 97, 100, -9
5, 55, -62, 35, 66, 69, 112, 80, 44, -4, -104, -5, -85, 38, -101, -2, 109, 98, -
7, 107, 134, -109, -107, 85, -16, 60, -75, 111, 13, -102, -48, 28, -2, -62, 107, -10]
```

```
at java.base/java.lang.reflect.Method.invoke(Method.java:568)
at me.markoutte.joker.timsort.MainKt.main(Main.kt:45)
```

```
timSort: [[-56, 27, -50, -115, -126, -24, -86, 114, -36, 118, -70, 77, 120, 89,
-64, -118, -82, -29, 96, -113, -59, 105, 65, 22, 23, -34, 76, 17, -19, 109, -26,
85, -50, -96, -7, -51, 97, -73, 114, -97, 54, -66, 2, -18, 95, -58, 97, 100, -9
5, 55, -62, 35, 66, 69, 112, 80, 44, -4, -104, -5, -85, 38, -101, -2, 109, 98, -
7, -35, -109, -107, 85, -16, 60, -75, 111, 13, -102, -48, 28, -2, -62, -107, -10
1, -2, 54, -11, 63, 89, -98, 71, 109, -127, -57, 12, -88, 73, 16, -14, 120, 34,
-103, 32, -64, 119, 102, -84, -53, -35, 13, 66, 64, -35, -12, 101, 9, 95, 61, -2
, 116, 64, -10, -90, 84, -83, 97, 5, 54, -57, 66, -44, -91, -99, 29, -89, 103, 1
21, -8, 80, 56, -18, -20, 101, -31, -71, 42, 58, 61, -64, 96, 47, -5, -117, -1,
61, -119, -100, 45, -83, 87, 49, -63, -16, -42, -30, -15, 74, 63, 49, 104, 3]]
```



```
[-86, -56, 27, -50, -115, -126, -24, -86, 114, -36, 118, -70, 77, 120, 89, -64,
-118, -82, -29, 96, -113, -59, 105, 65, 22, 23, -34, 76, 17, -19, 109, -26, 85,
-50, -96, -7, -51, 97, -73, 114, -97, 54, -66, 2, -18, 95, -58, 97, 100, -95, 55]
```

● ● ● ↵⌘2

less

```
5, -51, -50, 123, -95, -33, 58, 11, -37, -74, 123, -51, -98, -10, 29, -47, -60,
-12, -88, 81, 46, 34, -124, 15, -70, 124, -74, 17, -26, -78, 26, 109, 8, -50, -2
7, 88, -70, 55, -73, -97, -1, 36, -70, 29, 35, -111, 90, -11, 46, 11, 43, -106,
2, -63, -119, -117, 19, 93, -9, -62, 90, 67, -75, -2, 78, 24, -113, -92, 61, -28
, 112, 89, -46, 65, -28, 122, -125, -83, 88, 16, 122, -111, -108, 104, -52, 72,
93, 76, -65, 68, 26, -66, 85, 40, -91, -121, -95, -47, 50, -113, -16, -95, 119,
-79, 27, 103, -88, 23, -88, -52, 125]
```

```
<AA><C8>ESC[<82><E8><AA>r<DC>v<BA>MxY<C0><8A><AE><E3>`<8F><C5>iA^V^W<DE>L^Q<ED>m
<E6>U\P<F9><CD>a<B7>r<9F>6<BE>^B<EE>_<C6>ad<A1>7<C2>#BEpP,<FC><98><FB><AB>&<9B><F
<FE>mb<F9>\<95>U<F0><<B5>o^M<9A><D0>^\<FE><U+0095><9B><FE>6<F5>?Y<9E>Gm<81><C7>^
^L<A8>I^P<F2>x"
```



(END)

The screenshot shows an IDE interface with the following details:

Top Bar: Shows project name "Joker2023", file "main", and "TimSort.java". Status icons include a clock, copyright symbol, green square, red square, and more.

Code Editor: Displays Java code for the `merge` method of the `TimSort` class. The cursor is at line 30, which contains a line of code that has triggered a warning (indicated by a lightning bolt icon). The code is as follows:

```
1 usage  Maksim Pelevin
26
27     public static void merge(int[] arr, int left, int mid, int right) {    arr: [-127, -
28         int leftArrLen = mid - left + 1, rightArrLen = right - mid;    left: 128      mid
29         int[] leftArr = new int[leftArrLen];    leftArrLen: 64      leftArr: [0, 0, 0, 0
30     int[] rightArr = new int[rightArrLen];    rightArrLen: -22
31
```

Call Stack: Shows the current stack frame and its callers:

- "main...NNING" (checked)
- merge:30, TimSort (me.main)
- timSort:84, TimSort (me.main)
- main:92, TimSort (me.main)

Variables: Inspected variables from the current frame (merge:30):

- Exception = {NegativeArraySizeException@492}
- arr = {int[170]@494} [-127, -126, -118, -115, -113, -109, -107, -107, -104, -103, ... View]
- left = 128
- mid = 191
- right = 169
- leftArrLen = 64
- rightArrLen = -22

Bottom Bar: Shows the file path: "src > main > java > me > markoutte > examples > TimSort > merge". Status includes "30:47", "LF", "UTF-8", and file status indicators for "main".

The screenshot shows a Java development environment with the following details:

- Project:** Joker2023
- File:** TimSort.java
- Code Snippet:** A portion of the TimSort.java code is shown, specifically the merge sort loop. The code includes a bug fix for the right boundary of the merge operation.

```
77     . . . . .
78
79     . . . . for (int size = RUN; size < length; size = 2 * size) {
80     . . . .     for (int left = 0; left < length; left += 2 * size) {
81     . . . .         int mid = left + size - 1;
82     . . . .         int right = Math.min((left + 2 * size - 1), (length - 1));
83     . . . .         // perform merge sort
84     . . . .         merge(arr, left, mid, right);
85     . . . .     }
86     . . . . }
87
88 }
```

- Debug View:** The "Threads & Variables" tab is selected in the debugger interface.
- Variables:** The current frame is "timSort:84, TimSort". The variables listed are:
 - size = 64
 - left = 128
 - mid = 191
 - right = 169
- Status Bar:** The status bar at the bottom shows the path markoutte > examples > TimSort > timSort, the file name TimSort.java, the line count 82:17 (58 chars), the encoding LF, the character set UTF-8, the TypeScript version 5.1.3, and the code style setting 4 spaces.

Find x .





← →



guava



master



© guava-tests/.../IntsTest.java



IntsTest.testRotateIndexed

```
414 public void testRotateIndexed() {  
415     testRotate(new int[] {}, distance: 0, fromIndex: 0, toIndex: 0, new int[] {});  
416  
417     testRotate(new int[] {1}, distance: 0, fromIndex: 0, toIndex: 1, new int[] {1});  
418     testRotate(new int[] {1}, distance: 1, fromIndex: 0, toIndex: 1, new int[] {1});  
419     testRotate(new int[] {1}, distance: 1, fromIndex: 1, toIndex: 1, new int[] {1});  
420  
421     // Rotate the central 5 elements, leaving the ends as-is  
422     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: -6, fromIndex: 1, toIndex: 6,  
423     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: -1, fromIndex: 1, toIndex: 6,  
424     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: 0, fromIndex: 1, toIndex: 6,  
425     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: 5, fromIndex: 1, toIndex: 6,  
426     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: 14, fromIndex: 1, toIndex: 6  
427  
428     // Rotate the first three elements  
429     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: -2, fromIndex: 0, toIndex: 3,  
430     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: -1, fromIndex: 0, toIndex: 3,  
431     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: 0, fromIndex: 0, toIndex: 3,  
432     testRotate(new int[] {0, 1, 2, 3, 4, 5, 6}, distance: 1, fromIndex: 0, toIndex: 3}
```

guava master guava/.../Ints.java IntsTest.testRotateIndexed

```
525
526 /**
527 * Performs a right rotation of {@code array} between {@code fromIndex} inclusi
528 * toIndex} exclusive. This is equivalent to {@code
529 * Collections.rotate(Ints.asList(array).subList(fromIndex, toIndex), distance)}
530 * considerably faster and avoids allocations and garbage collection.
531 *
532 * <p>The provided "distance" may be negative, which will rotate left.
533 *
534 * @throws IndexOutOfBoundsException if {@code fromIndex < 0}, {@code toIndex >
535 *         {@code toIndex > fromIndex}}
536 * @since 32.0.0
537 */
538 public static void rotate(int[] array, int distance, int fromIndex, int toIndex
539     // There are several well-known algorithms for rotating part of an array (or,
540     // exchanging two blocks of memory). This classic text by Gries and Mills men
541     // https://ecommons.cornell.edu/bitstream/handle/1813/6292/81-452.pdf.
542     // (1) "Reversal", the one we have here.
543     // (2) "Dolphin". If we're rotating an array a of size n by a distance of d,
```

guava > src > com > google > common > primitives > Ints > rotate 538:22 LF UTF-8 2 spaces* ⌂

```
/**  
 * Performs a right rotation of {@code array} between {@code fromIndex} inclusive and {@code  
 * toIndex} exclusive. This is equivalent to {@code  
 * Collections.rotate(Ints.asList(array).subList(fromIndex, toIndex), distance)}, but is  
 * considerably faster and avoids allocations and garbage collection.  
 *  
 * The provided "distance" may be negative, which will rotate left.  
 *  
 * @throws IndexOutOfBoundsException if {@code fromIndex < 0}, {@code toIndex > array.length}, or  
 *      {@code toIndex > fromIndex}  
 * @since 32.0.0  
 */
```



```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)

    println("Running: $className.$methodName) with seed = $seed")
    val errors = mutableSetOf<String>()
    val b = ByteArray(300)
    val start = System.nanoTime()

    val javaMethod = try {
        loadJavaMethod(className, methodName)
    } catch (e: Exception) {
        errors.add("Failed to load Java method: ${e.message}")
    }
```

```
fun main(args: Array<String>) {
    val options = Options().apply {
        addOption("c", "class", true, "Java class fully qualified name")
        addOption("m", "method", true, "Method to be tested")
        addOption("cp", "classpath", true, "Classpath with libraries")
        addOption("t", "timeout", true, "Maximum time for fuzzing in seconds")
        addOption("s", "seed", true, "The source of randomness")
    }
    val parser = DefaultParser().parse(options, args)
    val className = parser.getOptionValue("class")
    val methodName = parser.getOptionValue("method")
    val classPath = parser.getOptionValue("classpath")
    val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
    val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
    val random = Random(seed)

    println("Running: $className.$methodName) with seed = $seed")
    val errors = mutableSetOf<String>()
    val b = ByteArray(300)
    val start = System.nanoTime()

    val javaMethod = try {
```

```
val parser = DefaultParser().parse(options, args)
val className = parser.getOptionValue("class")
val methodName = parser.getOptionValue("method")
val classPath = parser.getOptionValue("classpath")
val timeout = parser.getOptionValue("timeout")?.toLong() ?: 10L
val seed = parser.getOptionValue("seed")?.toInt() ?: Random.nextInt()
val random = Random(seed)

println("Running: $className.$methodName with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

val javaMethod = try {
    loadJavaMethod(className, methodName, classPath)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
    try {
        javaMethod.invoke(null, *inputValues)
    } catch (e: InvocationTargetException) {
        if (errors.add(e.targetException::class.qualifiedName!!)) {
            val errorMessage = e.targetException::class.simpleName
            errors.add(errorMessage)
        }
    }
}
```

```
        }

        println("Errors found: ${errors.size}")
        println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
            System.nanoTime() - start
        )} ms")
    }

fun loadJavaMethod(className: String, methodName: String): Method {
    val classLoader = ClassLoader.getSystemClassLoader()
    val javaClass = classLoader.loadClass(className)
    val javaMethod = javaClass.declaredMethods.first {
        "${it.name}(${it.parameterTypes.joinToString(",")} {
            c → c.typeName
        })" == methodName
    }
    return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
    val parameterTypes = method.parameterTypes
    return Array(parameterTypes.size) {
        when (parameterTypes[it]) {
            IntArray::class.java → IntArray(buffer.get().toUByte().toInt()) {
                buffer.get().toInt()
            }
            else → error("Cannot create value of type ${parameterTypes[it]}")
        }
    }
}
```

```
        println("Saved to: ${path.fileName}")
    }
}

println("Errors found: ${errors.size}")
println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
    System.nanoTime() - start
)} ms")
}

fun loadJavaMethod(className: String, methodName: String, classPath: String): Method {
    val libraries = classPath
        .split(File.pathSeparatorChar)
        .map { File(it).toURI().toURL() }
        .toTypedArray()
    val classLoader = URLClassLoader(libraries)
    val javaClass = classLoader.loadClass(className)
    val javaMethod = javaClass.getDeclaredMethods().first {
        "${it.name}(${it.parameterTypes.joinToString(",")} {
            c → c.typeName
        })" == methodName
    }
    return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
```

```
val javaMethod = javaClass.declaredMethods.first {
    "${it.name}(${it.parameterTypes.joinToString(",")} {
        c → c.typeName
    })" == methodName
}
return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
    val parameterTypes = method.parameterTypes
    return Array(parameterTypes.size) {
        when (parameterTypes[it]) {
            IntArray::class.java → IntArray(buffer.get().toUByte().toInt()) {
                buffer.get().toInt()
            }
            else → error("Cannot create value of type ${parameterTypes[it]}")
        }
    }
}
```

```
"${it.name}(${it.parameterTypes.joinToString(",")} {  
    c → c.typeName  
})" == methodName  
}  
  
return javaMethod  
}  
  
fun generateInputValues(method: Method, data: ByteArray): Array<Any> {  
    val buffer = ByteBuffer.wrap(data)  
    val parameterTypes = method.parameterTypes  
    return Array(parameterTypes.size) {  
        when (parameterTypes[it]) {  
            Int::class.java → buffer.get().toInt()  
            IntArray::class.java → IntArray(buffer.get().toUByte().toInt()) {  
                buffer.get().toInt()  
            }  
            else → error("Cannot create value of type ${parameterTypes[it]}")  
        }  
    }  
}
```

● ● ● ✎ 2

-zsh

```
markoutte@Aquarius libs % java -jar fuzzer-rotate.jar -c com.google.common.primitives.Ints -m "rotate(int[],int,int,int)" -cp guava-32.1.1-jre.jar
```

▶ 0:45 / 1:21



● ● ● ✎⌘2

-zsh

```
markoutte@Aquarius libs % java -jar fuzzer-rotate.jar -c com.google.common.primitives.Ints -m "rotate(int[],int,int,int)" -cp guava-32.1.1-jre.jar
Running: com.google.common.primitives.Ints.rotate(int[],int,int,int)) with seed
= 179047264
New error found: IndexOutOfBoundsException
Saved to: reportIndexOutOfBoundsException.txt
New error found: ArithmeticException
Saved to: reportArithmeticException.txt
Errors found: 2
Time elapsed: 10000 ms
markoutte@Aquarius libs %
```

▶ 0:56 / 1:21

🔇 🔍 ⏱

```
java.lang.ArithmetricException: / by zero
    at com.google.common.primitives.Ints.rotate(Ints.java:575)
    at jdk.internal.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at me.markoutte.joker.rotate.MainKt.main(Main.kt:49)
```

```
rotate: [[99, 113, -97, -64, -97, -3, 59, 126, 34, -54, 61, -37, 41, 51, -126, -102, 107, 93, 65, -12, 51, 101, -37, 42, -116, -8, 106, -121, 94, -46, -38, -82, -51, 51, 108, -24, -70, -78, 19, 38, 37, 95, 116, -19, -29, -5, -62, -58, -22, 18, 86, 120, -63, -33, 58, 110, -90, -79, 8, 43, -107, -66, -12, 101, 32, 115, -72, 114, -3, -94, 106, 101, -13, -83, -17, 19, 109, 30, 0, 125, -81, 95, 63, 18, -99, -62, -31, -115, 28, -125, 127, -81, -58, -93, -88, 76, 65, -11, 43, 34, 16, 54, -48, 26, -79, -115, 114, -73, 85, -71, 97, -102, 12, -69, -42, 111, 40, 45, -43, 112, 64, 93, -18, 48, -2, -125, 108, -68, -36, 5, 58, 8, 37, -118, -66, 98▶ 1:0971:2139, -57, 35, -1, 66, -24, -72, -73, 123, 17, 19, -105, 4, 117, ↵59, ↵-62,
```

```
legatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at me.markoutte.joker.rotate.MainKt.main(Main.kt:49)
```

```
rotate: [[99, 113, -97, -64, -97, -3, 59, 126, 34, -54, 61, -37, 41, 51, -126, -102, 107, 93, 65, -12, 51, 101, -37, 42, -116, -8, 106, -121, 94, -46, -38, -82, -51, 51, 108, -24, -70, -78, 19, 38, 37, 95, 116, -19, -29, -5, -62, -58, -22, 18, 86, 120, -63, -33, 58, 110, -90, -79, 8, 43, -107, -66, -12, 101, 32, 115, -72, 114, -3, -94, 106, 101, -13, -83, -17, 19, 109, 30, 0, 125, -81, 95, 63, 18, -99, -62, -31, -115, 28, -125, 127, -81, -58, -93, -88, 76, 65, -11, 43, 34, 16, 54, -48, 26, -79, -115, 114, -73, 85, -71, 97, -102, 12, -69, -42, 111, 40, 45, -43, 112, 64, 93, -18, 48, -2, -125, 108, -68, -36, 5, 58, 8, 37, -118, -66, 98, -73, 39, -57, 35, -1, 66, -24, -72, -73, 123, 17, 19, -105, 4, 117, -59, -62, -50, 42, -105, -11, -85, -6, -44, 18, 67, 11], 45, 120, 120]
```

```
[-93, 99, 113, -97, -64, -97, -3, 59, 126, 34, -54, 61, -37, 41, 51, -126, -102, 107, 65, -12, 51, 101, -37, 42, -116, -8, 106, -121, 94, -46, -38, -82, -51]
```

The screenshot shows a dark-themed IDE interface with the following details:

- Top Bar:** Shows project names "sandbox" and "master", file name "Ints.java", and tabs for "TestGuavaRotate".
- Code Editor:** Displays Java code for rotating an array. A specific line is highlighted with a blue background: `int m = -distance % length; distance: 45 length: 0`. The code involves calculating a distance and adjusting indices.
- Debug Tool Window:** Titled "Debug", it contains a "Threads & Variab..." tab. It lists variables and their values:
 - "main:8, TestGuavaRotate" has an "Exception" value of `{ArithmaticException@1111}`.
 - "array" is an int[163] array with values [99, 113, -97, -64, -97, -3, 59, 126, ...].
 - "distance" is 45.
 - "fromIndex" is 120.
 - "toIndex" is 120.
- Bottom Status Bar:** Shows the path "guava-32.1.1-jre-sources.jar > com > google > common > primitives > Ints", line number 575:1, encoding LF, character set UTF-8, and a 4 spaces setting.





main ▾

fastjson2 / core / src / main / java / com / alibaba / fastjson2 / JSON.java

↑ Top

Code

Blame

3950 lines (3593 loc) · 152 KB

Raw



```
36     public interface JSON {  
37  
38         /**  
39          * Parses the json string as a {@link JSONArray} or {@link JSONObject}.  
40          * Returns {@code null} if received {@link String} is {@code null} or empty.  
41          *  
42          * @param text the specified text to be parsed  
43          * @return either {@link JSONArray} or {@link JSONObject} or null  
44          * @throws JSONException If a parsing error occurs  
45          */  
46  
47         static Object parse(String text) {  
48             if (text == null || text.isEmpty()) {  
49                 return null;  
50             }  
51  
52             ObjectReaderProvider provider = JSONFactory.getDefaultObjectReaderProvider();  
53             final JSONReader.Context context = new JSONReader.Context(provider);  
54             try (JSONReader reader = JSONReader.of(text, context)) {  
55                 Object object;  
56                 char ch = reader.current();  
57  
58                 if (context.objectSupplier == null  
59                     && (context.features & UseNativeObject.mask) == 0  
60                     && (ch == '{' || ch == '[')  
61                 ) {  
62                     if (ch == '{') {  
63                         JSONObject jsonObject = new JSONObject();  
64                         reader.read(jsonObject, 0);  
65                         object = jsonObject;  
66                     } else {  
67                         JSONArray jsonArray = new JSONArray();  
68                         reader.read(jsonArray, 0);  
69                         object = jsonArray;  
70                     }  
71                 } else {  
72                     if (ch == '[') {  
73                         JSONArray jsonArray = new JSONArray();  
74                         reader.read(jsonArray, 0);  
75                         object = jsonArray;  
76                     } else {  
77                         JSONObject jsonObject = new JSONObject();  
78                         reader.read(jsonObject, 0);  
79                         object = jsonObject;  
80                     }  
81                 }  
82             } catch (JSONException e) {  
83                 throw new JSONException("Failed to parse JSON: " + text, e);  
84             }  
85         }  
86     }  
87 }
```

```
val javaClass = classLoader.loadClass(className)
val javaMethod = javaClass.getDeclaredMethods().first {
    "${it.name}(${it.parameterTypes.joinToString(",")} {
        c → c.typeName
    })" == methodName
}
return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
    val parameterTypes = method.parameterTypes
    return Array(parameterTypes.size) {
        when (parameterTypes[it]) {
            Int::class.java → buffer.get().toInt()
            IntArray::class.java → IntArray(buffer.get().toUByte().toInt()) {
                buffer.get().toInt()
            }
            else → error("Cannot create value of type ${parameterTypes[it]}")
        }
    }
}
```

```
        c → c.typeName
    })" = methodName
}
return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
    val parameterTypes = method.parameterTypes
    return Array(parameterTypes.size) {
        when (parameterTypes[it]) {
            Int::class.java → buffer.get().toInt()
            IntArray::class.java → IntArray(buffer.get().toUByte().toInt()) {
                buffer.get().toInt()
            }
            String::class.java → String(ByteArray(
                buffer.get().toUByte().toInt() + 1
            ) {
                buffer.get()
            }, Charset.forName("koi8"))
            else → error("Cannot create value of type ${parameterTypes[it]}")
        }
    }
}
```

● ● ●

-zsh

```
markoutte@Aquarius libs % java -jar fuzzer-parse-step1.jar -c com.alibaba.fastjson2.JSON -m "parse(java.lang.String)" -cp fastjson2-2.0.38.jar
```

▶ 0:41 / 1:13



● ● ●

java

```
markoutte@Aquarius libs % java -jar fuzzer-parse-step1.jar -c com.alibaba.fastjson2.JSON -m "parse(java.lang.String)" -cp fastjson2-2.0.38.jar  
Running: com.alibaba.fastjson2.JSON.parse(java.lang.String) with seed = 1581094789  
New error found: JSONException  
Saved to: reportJSONException.txt  
New error found: ArrayIndexOutOfBoundsException  
Saved to: reportArrayIndexOutOfBoundsException.txt  
Errors found: 2  
Time elapsed: 10000 ms  
markoutte@Aquarius libs %
```

▶ 0:52 / 1:13

🔇 🔍 ⏱

● ● ● ✎ 2

less

```
java.lang.ArrayIndexOutOfBoundsException: Index 1 out of bounds for length 1
    at com.alibaba.fastjson2.JSONReaderUTF8.readNumber0(JSONReaderUTF8.java:6295)
    at com.alibaba.fastjson2.JSONReader.readNumber(JSONReader.java:1057)
    at com.alibaba.fastjson2.reader.ObjectReaderImplObject.readObject(ObjectReaderImplObject.java:273)
    at com.alibaba.fastjson2.JSON.parse(JSON.java:79)
    at jdk.internal.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at me.markoutte.joker.parse.step1.MainKt.main(Main.kt:50)
```

parse: [.]

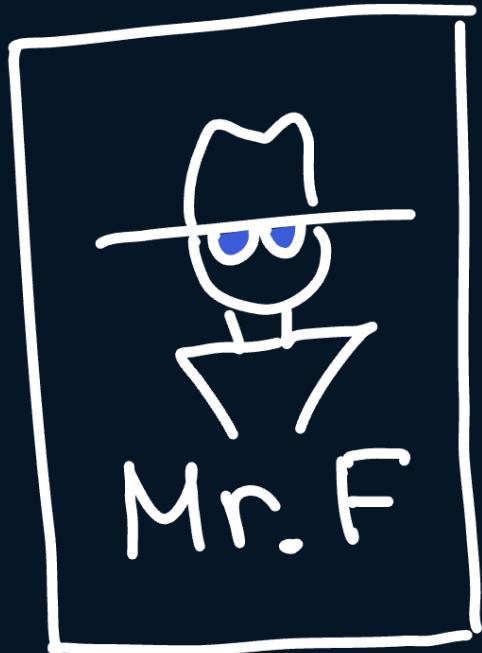
```
[0, 46, -6, -79, 86, 98, -103, 103, -23, -67, 99, -77, 97, -41, 93, 37, 42, 113,
53, 103, 71, 13, -39, -98, 8, 55, 12, -24, -86, -12, -64, 87, -115, 63, 102, -39, 99, 80]
```

● ○ ● ← → S sandbox v ⌂ master v ⌂ JSONReaderUTF8.java TestFastjsonParse v D Q

6278 }
6279 ch = (char) bytes[offset++];
6280 }
6281
... 6282 if (ch == '.') {
6283 valueType = JSON_TYPE_DEC;
6284 ch = (char) bytes[offset++]; bytes: [46]
6285 while (ch >= '0' && ch <= '9') {
6286 if (!intOverflow) {
6287 int digit = ch - '0';

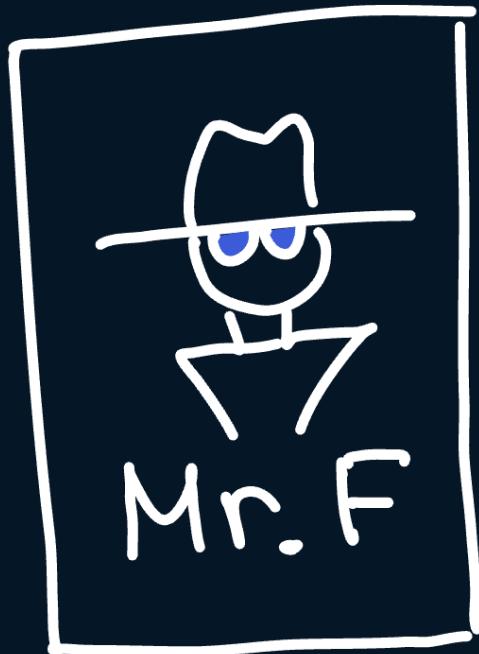
Debug Threads & Variab... Evaluate expression (⌚) or add a watch (⌚⌚)
✓ "main"@1 ...: RUNNING ↴
↳ readNumber0:6284, JSONReaderL ⑩ limit = -2147483647
readNumber:1057, JSONReader (c ⑩ multmin = -214748364
readObject:277, ObjectReaderImp ⑩ intOverflow = false
parse:79, JSON (com.alibaba.fastjson2 ⑩ offset = 2
main:10, TestFastjsonParse (me.m ⑩ ch = '.' 46
Switch frames from anywhere in the l... x > ⑩ bytes = {byte[1]@1351} [46]
fastjson2-2.0.40-sources.jar > com > alibaba > fastjson2 > ⌂ JSONReaderUTF8 6285:45 LF UTF-8 4 spaces





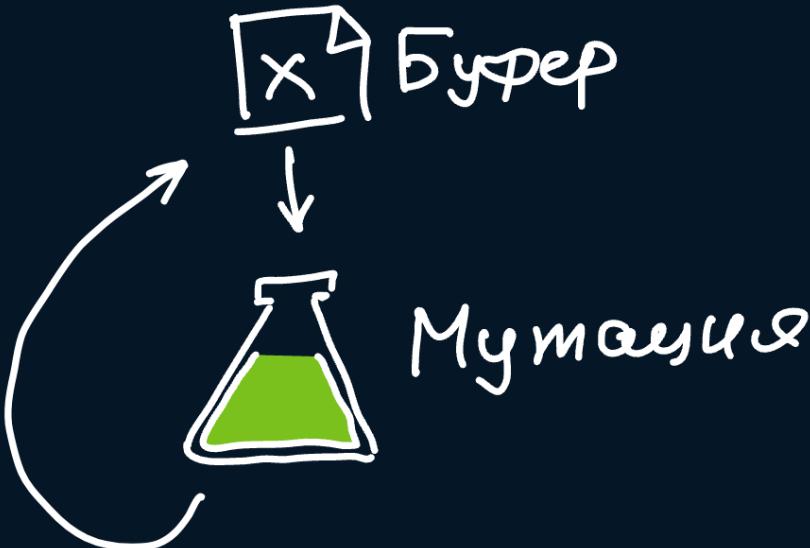
Mr. F

Буфер





Mr. F

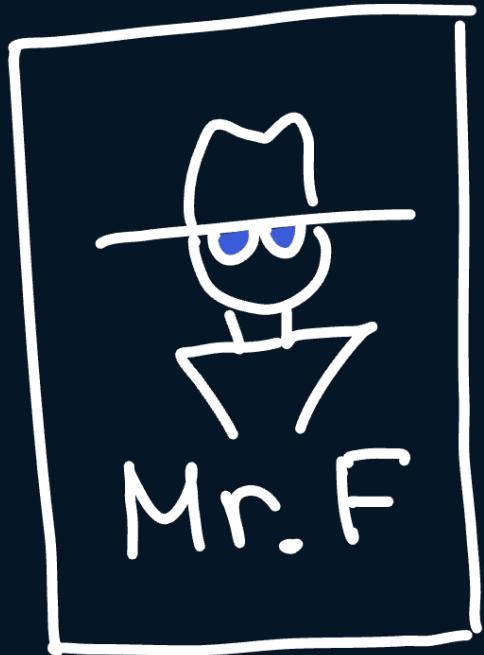


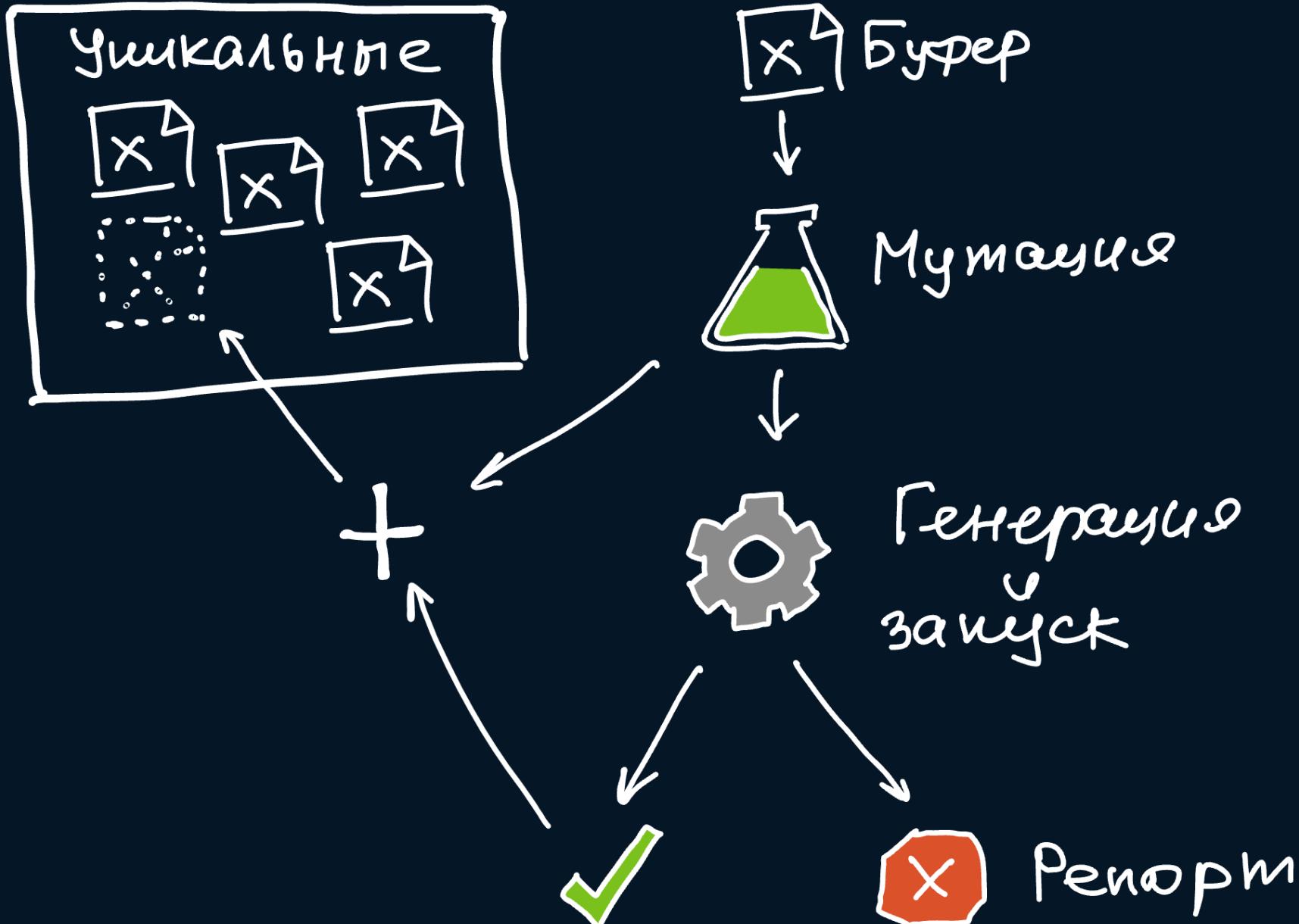
Буфер

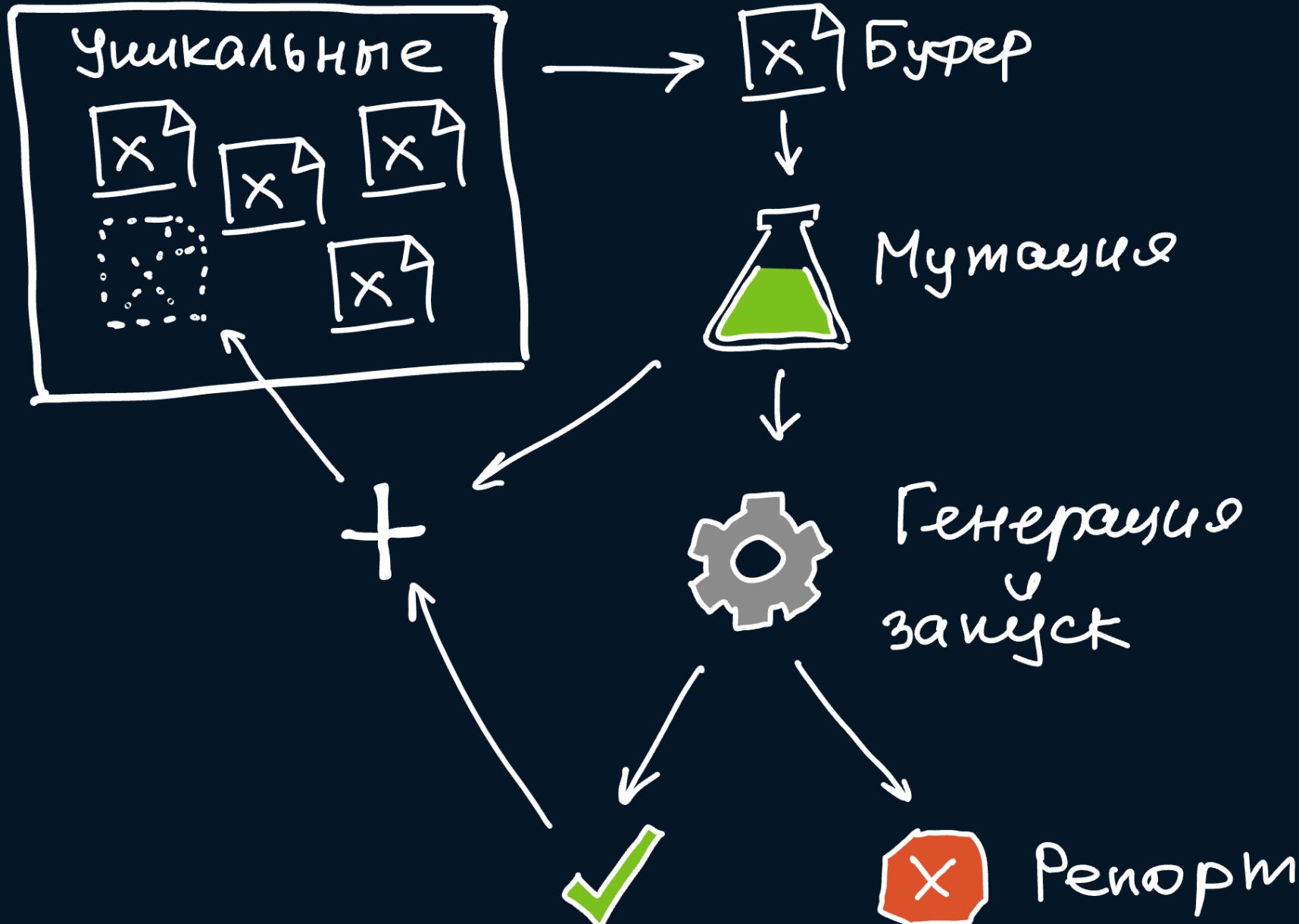
Мумасик











```
println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

val javaMethod = try {
    loadJavaMethod(className, methodName, classPath)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
    try {
        javaMethod.invoke(null, *inputValues)
    } catch (e: InvocationTargetException) {
        if (errors.add(e.targetException::class.qualifiedName!!)) {
            val errorMessage = e.targetException::class.simpleName
            println("New error found: $errorMessage")
            val path = Paths.get("report$errorMessage.txt")
            Files.write(path, listOf(
                "${e.targetException.stackTraceToString()}\n",
                "$inputValuesString\n",
                "${buffer.contentToString()}\n",
            ))
        }
    }
}
```

```
val javaMethod = try {
    loadJavaMethod(className, methodName, classPath)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

val seeds = mutableMapOf<Int, ByteArray>()

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = seeds.values.randomOrNull(random)?.let(Random::mutate)
        ?: b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
    try {
        javaMethod.invoke(null, *inputValues).apply {
            val seedId = buffer.contentHashCode()
            if (seeds.putIfAbsent(seedId, buffer) == null) {
                println("New seed added: ${seedId.toHexString()}")
            }
        }
    } catch (e: InvocationTargetException) {
        if (errors.add(e.targetException::class.qualifiedName!!)) {
            val errorMessage = e.targetException::class.simpleName
            println("New error found: $errorMessage")
            val path = Paths.get("report$errorMessage.txt")
            Files.write(path, listOf(
                "${e.targetException.stackTraceToString()}\n"
            ))
        }
    }
}
```

```
String..class.java > String(B, ByteArray,
    buffer.get().toUByte().toInt() + 1
) {
    buffer.get()
}, Charset.forName("koi8"))
else → error("Cannot create value of type ${parameterTypes[it]}")
}
}

fun Random.mutate(buffer: ByteArray): ByteArray = buffer.clone().apply {
    val position = nextInt(0, size)
    val repeat = nextInt((size - position))
    val from = nextInt(-128, 127)
    val until = nextInt(from + 1, 128)
    repeat(repeat) { i →
        set(position + i, nextInt(from, until).toByte())
    }
}
```

● ● ● ✎ 2

-zsh

```
markoutte@Aquarius libs % ls | grep 'report*'  
markoutte@Aquarius libs % java -jar fuzzer-parse-step2.jar -c com.alibaba.fastjson2.JSON -m "parse(java.lang.String)" -cp fastjson2-2.0.38.jar  
bck-i-search: ste_
```

▶ 0:14 / 0:46



● ● ● ✎⌘2

-zsh

```
New seed added: e56fbe2f
New seed added: cadcd04e
New seed added: f9a7890d
New seed added: a64ffb8b
New seed added: c704f73f
New seed added: 357bf1ae
New seed added: 5c63d53f
New seed added: 5d2cdfb2
New seed added: 0ad42c50
New seed added: 2cd21dfe
New seed added: a81aeb24
New seed added: 38cd242e
Traces found: 64815
Errors found: 2
Time elapsed: 10000 ms
markoutte@Aquarius libs % ls | grep 'report*'
reportArrayIndexOutOfBoundsException.txt
reportJSONException.txt
markoutte@Aquarius libs %
```



```
void choose( int x ) {  
    if ( x > 0 ) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

```
void choose( int x ) {  
    if ( x > 0 ) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

choose(1)

```
void choose( int x ) {  
    if ( x > 0 ) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

choose(1)
choose(2)

```
void choose( int x ) {  
    if ( x > 0 ) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

choose(1)
choose(2)
choose(3)

```
void choose( int x ) {  
    if (x > 0) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

choose(1)
choose(2)
choose(3)
choose(-10)

```
void choose( int x ) {  
    if (x > 0) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

choose(1)
choose(2)
choose(3)
choose(-10)
choose(0)

```
void choose( int x ) {  
    if ( x > 0 ) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

```
void choose (int x) {  
    if (x > 0) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

The diagram illustrates the execution flow of the `choose` function. The code is annotated with green highlights and arrows pointing to green boxes on the right, which are labeled with values from +1 to +6.

- The first green bar under `choose` points to a box labeled `+1`.
- The opening brace of the `if` statement points to a box labeled `+2`.
- The comment `// Hello` points to a box labeled `+3`.
- The opening brace of the `else` statement points to a box labeled `+4`.
- The comment `// GoodBye` points to a box labeled `+5`.
- The closing brace of the `choose` function points to a box labeled `+6`.

```
void choose (int x) {  
    if (x > 0) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

The diagram illustrates the execution flow of the `choose` function. The code is annotated with green highlights for function names (`choose`, `if`, `else`) and conditionals (`(x > 0)`). Gray highlights are used for comments (`// Hello 1`, `// GoodBye 2`). Arrows point from each annotated section to specific memory locations. A brace on the right groups the first four locations (green boxes) as they all point to the same memory address. The final result is shown as `+ 6 = 82`.

Annotations:

- `choose` (highlighted in green)
- `if` (highlighted in green)
- `(x > 0)` (highlighted in green)
- `// Hello 1` (highlighted in gray)
- `} else` (highlighted in green)
- `// GoodBye 2` (highlighted in gray)
- `}` (highlighted in green)

Memory Locations:

- `+ 1` (green box)
- `+ 2` (green box)
- `+ 3` (green box)
- `+ 4` (gray box)
- `+ 5` (gray box)
- `+ 6` (green box)

Result:

`+ 6 = 82`

```
void choose (int x) {  
    if (x > 0) {  
        // Hello 1  
    } else {  
        // GoodBye 2  
    }  
}
```

The diagram illustrates the execution flow of the `choose` function. The stack starts with `+ 1` at the top. If `x > 0`, it goes to `+ 2`. If `x ≤ 0`, it goes to `+ 3`. From `+ 3`, it branches to `+ 4` (if `x < 0`) or `+ 5` (if `x = 0`). Finally, it reaches `+ 6` at the bottom, which is annotated with `= 16`.

```
        } catch (e: Throwable) {
            println("Method $className#$methodName is not found")
            return
        }

    val seeds = mutableMapOf<Int, ByteArray>()

    while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
        val buffer = seeds.values.randomOrNull(random)?.let(Random::mutate)
            ?: b.apply(random::nextBytes)
        val inputValues = generateInputValues(javaMethod, buffer)
        val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
        try {
            javaMethod.invoke(null, *inputValues).apply {
                val seedId = buffer.contentHashCode()
                if (seeds.putIfAbsent(seedId, buffer) == null) {
                    println("New seed added: ${seedId.toHexString()}")
                }
            }
        } catch (e: InvocationTargetException) {
            if (errors.add(e.targetException::class.qualifiedName!!)) {
                val errorMessage = e.targetException::class.simpleName
                println("New error found: $errorMessage")
                val path = Paths.get("report$errorMessage.txt")
                Files.write(path, listOf(
                    "${e.targetException.stackTraceToString()}\n",
                    "$inputValuesString\n",
                    "${buffer.contentToString()}\n",
                ))
            }
        }
    }
}
```

```
    } catch (e: Throwable) {
        println("Method $className#$methodName is not found")
        return
    }

    val seeds = mutableMapOf<Int, ByteArray>()

    while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
        val buffer = seeds.values.randomOrNull(random)?.let(Random::mutate)
            ?: b.apply(random::nextBytes)
        val inputValues = generateInputValues(javaMethod, buffer)
        val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
        try {
            ExecutionPath.id = 0
            javaMethod.invoke(null, *inputValues).apply {
                val seedId = ExecutionPath.id
                if (seeds.putIfAbsent(seedId, buffer) == null) {
                    println("New seed added: ${seedId.toHexString()}")
                }
            }
        } catch (e: InvocationTargetException) {
            if (errors.add(e.targetException::class.qualifiedName!!)) {
                val errorMessage = e.targetException::class.simpleName
                println("New error found: $errorMessage")
                val path = Paths.get("report$errorMessage.txt")
                Files.write(path, listOf(
                    "${e.targetException.stackTraceToString()}\n",
                    "$inputValuesString\n",
                    "$buffer.contentToString()\n"
                ))
            }
        }
    }
}
```

```
        buffer.get().toInt()
    }
    String::class.java → String(ByteArray(
        buffer.get().toUByte().toInt() + 1
    ) {
        buffer.get()
    }, Charset.forName("koi8"))
else → error("Cannot create value of type ${parameterTypes[it]}")
}
}
}

object ExecutionPath {
    @JvmField
    var id: Int = 0
}

fun Random.mutate(buffer: ByteArray): ByteArray = buffer.clone().apply {
    val position = nextInt(0, 300)
    val repeat = nextInt((300 - position))
    val from = nextInt(-128, 127)
    val until = nextInt(from + 1, 128)
    repeat(repeat) { i →
        set(position + i, nextInt(from, until).toByte())
    }
}
```

```
        println("Seeds found: ${seeds.size}")
        println("Errors found: ${errors.size}")
        println("Time elapsed: ${TimeUnit.NANOSECONDS.toMillis(
            System.nanoTime() - start
        )} ms")
    }

fun loadJavaMethod(className: String, methodName: String, classPath: String): Method {
    val libraries = classPath
        .split(File.pathSeparatorChar)
        .map { File(it).toURI().toURL() }
        .toTypedArray()
    val classLoader = URLClassLoader(libraries)
    val javaClass = classLoader.loadClass(className)
    val javaMethod = javaClass.getDeclaredMethods().first {
        "${it.name}(${it.parameterTypes.joinToString(",")} {
            c → c.typeName
        })" == methodName
    }
    return javaMethod
}

fun generateInputValues(method: Method, data: ByteArray): Array<Any> {
    val buffer = ByteBuffer.wrap(data)
    val parameterTypes = method.parameterTypes
    return Array(parameterTypes.size) {
        when (parameterTypes[it]) {
```

```
fun loadJavaMethod(className: String, methodName: String, classPath: String): Method {
    val libraries = classPath
        .split(File.pathSeparatorChar)
        .map { File(it).toURI().toURL() }
        .toTypedArray()
    val classLoader = object : URLClassLoader(libraries) {
        override fun loadClass(name: String, resolve: Boolean): Class<*> {
            return if (name.startsWith(className.substringBeforeLast('.'))) {
                transformAndGetClass(name).apply {
                    if (resolve) resolveClass(this)
                }
            } else {
                super.loadClass(name, resolve)
            }
        }
    }

    fun transformAndGetClass(name: String): Class<*> {
        val owner = name.replace('.', '/')
        var bytes =
            getResourceAsStream("$owner.class")!!.use { it.readBytes() }
        val reader = ClassReader(bytes)
        val cl = this
        val writer = ComputeClassWriter(
            reader,
            ClassWriter.COMPUTE_MAXS or ClassWriter.COMPUTE_FRAMES,
            cl
        )
        val transformer = object : ClassVisitor(Opcodes.ASM9, writer) {
            override fun visitMethod(
                access: Int,
                name: String?,
                descriptor: String?,
                signature: String?,
                exceptions: Array<out String>?
            ): MethodVisitor {
                return object : MethodVisitor(
                    Opcodes.ASM9,
                    super.visitMethod(
                        access, name, descriptor, signature, exceptions
                    )
                ) {
                    val主人名 =
                        ExecutionPath.javaClass.canonicalName.replace('.', '/')
                    val fieldName = "id"

                    override fun visitLineNumber(line: Int, start: Label?) {
                        visitFieldInsn(
                            Opcodes.GETSTATIC, 主人名, fieldName, "I"
                        )
                        visitLdcInsn(line)
                        visitInsn(Opcodes.IADD)
                        visitFieldInsn(
                            Opcodes.PUTSTATIC, 主人名, fieldName, "I"
                        )
                        super.visitLineNumber(line, start)
                    }
                }
            }
        }
        reader.accept(transformer, ClassReader.SKIP_FRAMES)
        bytes = writer.toByteArray()
        return defineClass(name, bytes, 0, bytes.size)
    }
}

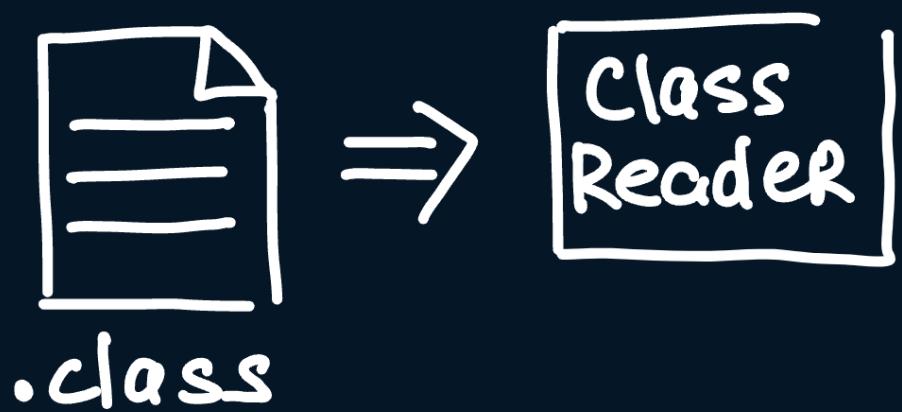
val javaClass = classLoader.loadClass(className)
val javaMethod = javaClass.getDeclaredMethods().first {
    "${it.name}(${it.parameterTypes.joinToString(",") {
        c → c.typeName
    }})" == methodName
}
```



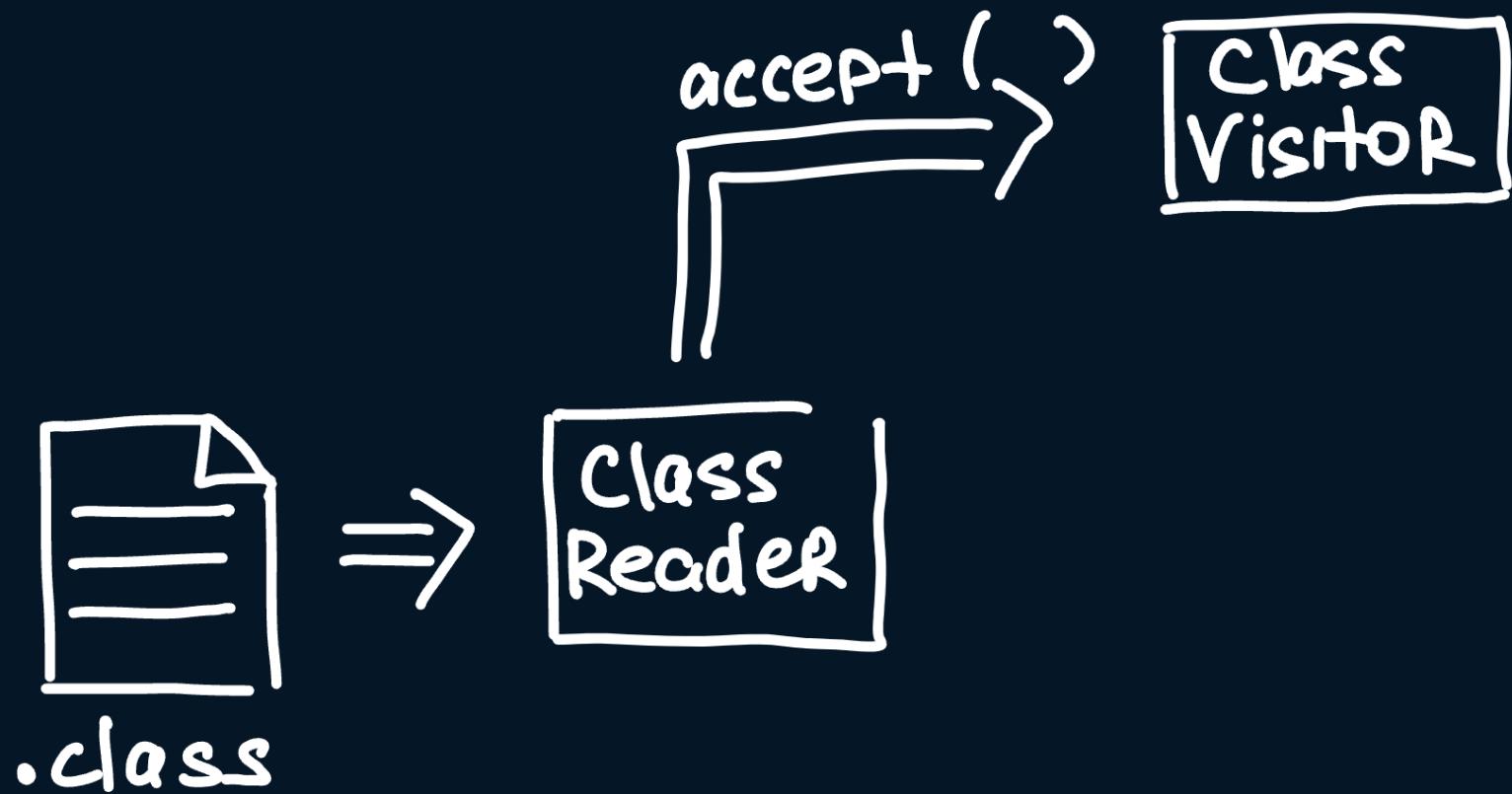
asm.ow2.io

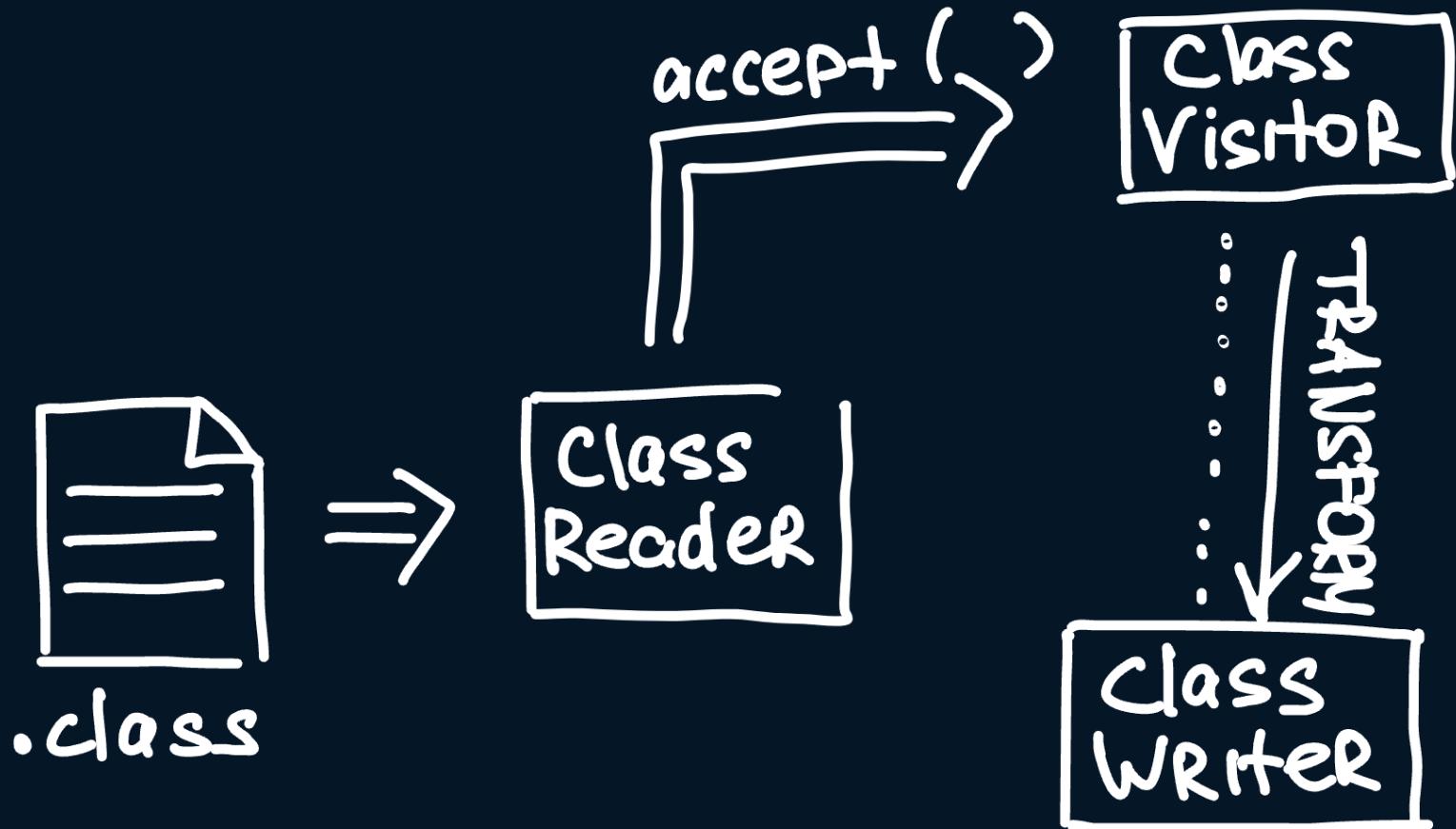


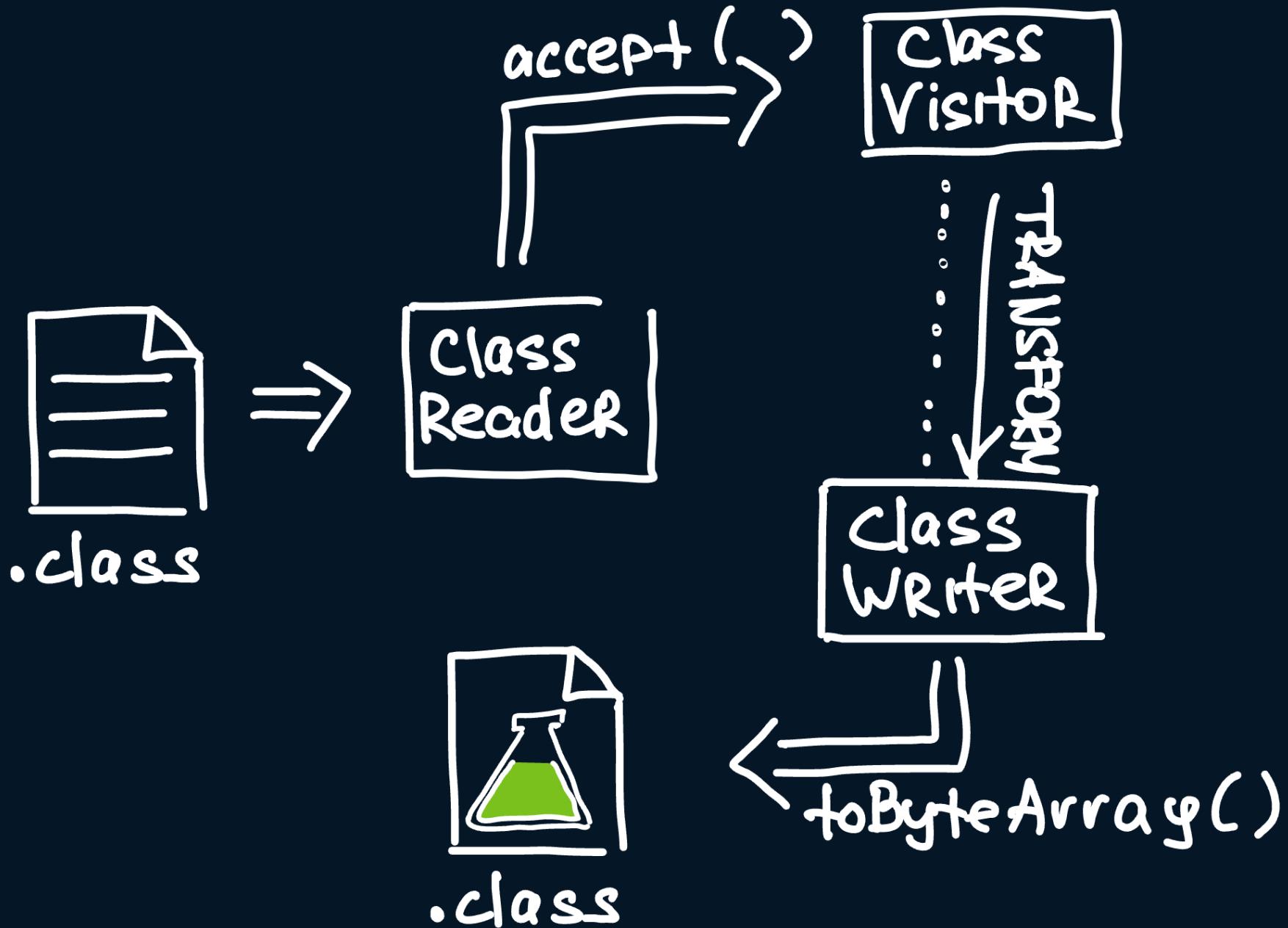
.class



.class







```
        System.nanoTime() - start
    )} ms")
}

fun loadJavaMethod(className: String, methodName: String, classPath: String): Method {
    val libraries = classPath
        .split(File.separatorChar)
        .map { File(it).toURI().toURL() }
        .toTypedArray()
    val classLoader = object : URLClassLoader(libraries) {
        override fun loadClass(name: String, resolve: Boolean): Class<*> {
            return if (name.startsWith(className.substringBeforeLast('.'))) {
                transformAndGetClass(name).apply {
                    if (resolve) resolveClass(this)
                }
            } else {
                super.loadClass(name, resolve)
            }
        }
    }

    fun transformAndGetClass(name: String): Class<*> {
        val owner = name.replace('.', '/')
        var bytes =
            getResourceAsStream("$owner.class")!!.use { it.readBytes() }
        val reader = ClassReader(bytes)
        val cl = this
        val writer = ComputeClassWriter(
            reader,
            ClassWriter.COMPUTE_MAXS or ClassWriter.COMPUTE_FRAMES,
            object : ComputeClassWriter.Visitor {
                override fun visitEnd() {
                    writer.visitEnd()
                }
            })
        writer.visitEnd()
        return writer.`class`!!
    }
}
```

```
.copy()
```

```
val classLoader = object : URLClassLoader(libraries) {
    override fun loadClass(name: String, resolve: Boolean): Class<*> {
        return if (name.startsWith(className.substringBeforeLast('.'))) {
            transformAndGetClass(name).apply {
                if (resolve) resolveClass(this)
            }
        } else {
            super.loadClass(name, resolve)
        }
    }

    fun transformAndGetClass(name: String): Class<*> {
        val owner = name.replace('.', '/')
        var bytes =
            getResourceAsStream("$owner.class")!!.use { it.readBytes() }
        val reader = ClassReader(bytes)
        val cl = this
        val writer = ComputeClassWriter(
            reader,
            ClassWriter.COMPUTE_MAXS or ClassWriter.COMPUTE_FRAMES,
            cl
        )
        val transformer = object : ClassVisitor(0pcodes.ASM9, writer) {
            override fun visitMethod(
                access: Int,
                name: String?,
                descriptor: String?,
                signature: String?
            ) {
                if (name == "main" && descriptor == "([Ljava/lang/String;)V") {
                    writer.visitCode()
                    writer.visitVarInsn(ALOAD, 0)
                    writer.visitMethodInsn(INVOKESTATIC, "java/lang/System", "arraycopy", "[Ljava/lang/Object; I [Ljava/lang/Object; I I)V", false)
                    writer.visitInsn(RETURN)
                    writer.visitMaxs(1, 1)
                } else {
                    visitor.visitMethod(access, name, descriptor, signature)
                }
            }
        }
        transformer.visitEnd()
        val classWriter = ClassWriter.COMPUTE_MAXS or ClassWriter.COMPUTE_FRAMES
        val classFileWriter = ClassFileWriter(owner, classWriter)
        classWriter.visit(cl)
        classWriter.visitEnd(classFileWriter)
        classFileWriter.write()
        bytes.close()
    }
}
```

```
        }
    } else {
        super.loadClass(name, resolve)
    }
}

fun transformAndGetClass(name: String): Class<*> {
    val owner = name.replace('.', '/')
    var bytes =
        getResourceAsStream("$owner.class")!!.use { it.readBytes() }
    val reader = ClassReader(bytes)
    val cl = this
    val writer = ComputeClassWriter(
        reader,
        ClassWriter.COMPUTE_MAXS or ClassWriter.COMPUTE_FRAMES,
        cl
    )
    val transformer = object : ClassVisitor(Opcodes.ASM9, writer) {
        override fun visitMethod(
            access: Int,
            name: String?,
            descriptor: String?,
            signature: String?,
            exceptions: Array<out String>?
        ): MethodVisitor {
            return object : MethodVisitor(
                Opcodes.ASM9,
                super.visitMethod(
                    name,
                    descriptor,
                    signature,
                    exceptions,
                    access
                )
            ) {
                override fun visitVarInsn(opcode: Int) {
                    if (opcode == ICONST_M1 || opcode == ICONST_1 || opcode == ICONST_2 || opcode == ICONST_3 || opcode == ICONST_4 || opcode == ICONST_5) {
                        writer.visitIntInsn(opcode, 0)
                    } else {
                        super.visitVarInsn(opcode)
                    }
                }
            }
        }
    }
    transformer.visit(cl)
    return writer.toByteArray()
}
```

```
val owner = name.replace('.', '/')
var bytes =
    getResourceAsStream("$owner.class")!!.use { it.readBytes() }
val reader = ClassReader(bytes)
val cl = this
val writer = ComputeClassWriter(
    reader,
    ClassWriter.COMPUTE_MAXS or ClassWriter.COMPUTE_FRAMES,
    cl
)
val transformer = object : ClassVisitor(Opcodes.ASM9, writer) {
    override fun visitMethod(
        access: Int,
        name: String?,
        descriptor: String?,
        signature: String?,
        exceptions: Array<out String>?
    ): MethodVisitor {
        return object : MethodVisitor(
            Opcodes.ASM9,
            super.visitMethod(
                access, name, descriptor, signature, exceptions
            )
        ) {
            val ownerName =
                ExecutionPath.javaClass.canonicalName.replace('.', '/')
            val fieldName = "id"

```

```
signature: String?,
exceptions: Array<out String>?
): MethodVisitor {
    return object : MethodVisitor(
        Opcodes.ASM9,
        super.visitMethod(
            access, name, descriptor, signature, exceptions
        )
    ) {
        val ownerName =
            ExecutionPath.javaClass.canonicalName.replace('.', '/')
        val fieldName = "id"

        override fun visitLineNumber(line: Int, start: Label?) {
            visitFieldInsn(
                Opcodes.GETSTATIC, ownerName, fieldName, "I"
            )
            visitLdcInsn(line)
            visitInsn(Opcodes.IADD)
            visitFieldInsn(
                Opcodes.PUTSTATIC, ownerName, fieldName, "I"
            )
            super.visitLineNumber(line, start)
        }
    }
}
reader.accept(transformer, ClassReader.SKIP_FRAMES)
```

```
        return object : MethodVisitor(
            Opcodes.ASM9,
            super.visitMethod(
                access, name, descriptor, signature, exceptions
            )
        ) {
            val ownerName =
                ExecutionPath.javaClass.canonicalName.replace('.', '/')
            val fieldName = "id"

            override fun visitLineNumber(line: Int, start: Label?) {
                visitFieldInsn(
                    Opcodes.GETSTATIC, ownerName, fieldName, "I"
                )
                visitLdcInsn(line)
                visitInsn(Opcodes.IADD)
                visitFieldInsn(
                    Opcodes.PUTSTATIC, ownerName, fieldName, "I"
                )
                super.visitLineNumber(line, start)
            }
        }
    }
    reader.accept(transformer, ClassReader.SKIP_FRAMES)
    bytes = writer.toByteArray()
    return defineClass(name, bytes, 0, bytes.size)
}
```

● ● ●

-zsh

```
markoutte@Aquarius libs % java -jar fuzzer-parse-step3.jar -c com.alibaba.fastjson2.JSON -m "parse(java.lang.String)" -cp fastjson2-2.0.38.jar  
bck-i-search: st_
```

▶ 0:00 / 0:49



● ● ● 7%2

-zsh

```
New seed added: 001b6a5b
New seed added: 001d4e94
New seed added: 0024678e
New seed added: 0009fa1e
New seed added: 0003b01b
New seed added: 00220cab
New seed added: 00131410
New seed added: 001b496b
New seed added: 0009a187
New seed added: 00052527
New seed added: 001df754
Traces found: 1455
Errors found: 3
Time elapsed: 10000 ms
markoutte@Aquarius libs % ls | grep 'report*'
reportArrayIndexOutOfBoundsException.txt
reportJSONException.txt
reportNumberFormatException.txt
markoutte@Aquarius libs %
```



less

```
java.lang.NumberFormatException: For input string: "88888889."
    at java.base/java.lang.NumberFormatException.forInputString(NumberFormatException.java:67)
    at java.base/java.lang.Integer.parseInt(Integer.java:668)
    at java.base/java.math.BigInteger.<init>(BigInteger.java:547)
    at java.base/java.math.BigInteger.<init>(BigInteger.java:676)
    at com.alibaba.fastjson2.JSONReader.getNumber(JSONReader.java:2977)
    at com.alibaba.fastjson2.JSONReader.readNumber(JSONReader.java:1058)
    at com.alibaba.fastjson2.reader.ObjectReaderImplObject.readObject(ObjectReaderImplObject.java:273)
    at com.alibaba.fastjson2.JSON.parse(JSON.java:79)
    at jdk.internal.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
    at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    at java.base/java.lang.reflect.Method.invoke(Method.java:568)
    at me.markoutte.joker.parse.step3.MainKt.main(Main.kt:58)
```



```
val random = Random(seed)

println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

val javaMethod = try {
    loadJavaMethod(className, methodName, classPath)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

val seeds = mutableMapOf<Int, ByteArray>()

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = seeds.values.randomOrNull(random)?.let(Random::mutate)
        ?: b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
    try {
        ExecutionPath.id = 0
        javaMethod.invoke(null, *inputValues).apply {
            val seedId = ExecutionPath.id
            if (seeds.putIfAbsent(seedId, buffer) == null) {
                println("New seed added: ${seedId.toHexString()}")
            }
        }
    }
}
```

```
println("Running: $className.$methodName) with seed = $seed")
val errors = mutableSetOf<String>()
val b = ByteArray(300)
val start = System.nanoTime()

val javaMethod = try {
    loadJavaMethod(className, methodName, classPath)
} catch (t: Throwable) {
    println("Method $className#$methodName is not found")
    return
}

val seeds = mutableMapOf<Int, ByteArray>(
    -1 to """{"name": { "arr": [1, 2, 3] }}""".asByteArray(b.size)!!
)

while(System.nanoTime() - start < TimeUnit.SECONDS.toNanos(timeout)) {
    val buffer = seeds.values.randomOrNull(random)?.let(Random::mutate)
        ?: b.apply(random::nextBytes)
    val inputValues = generateInputValues(javaMethod, buffer)
    val inputValuesString = "${javaMethod.name}: ${inputValues.contentDeepToString()}"
    try {
        ExecutionPath.id = 0
        javaMethod.invoke(null, *inputValues).apply {
            val seedId = ExecutionPath.id
            if (seeds.putIfAbsent(seedId, buffer) == null) {
                println("New seed added: ${seedId.toHexString()}")
            }
        }
    } catch (e: Exception) {
        errors.add(e.message)
    }
}
```

```
    val until = nextInt(from + 1, 128)
    repeat(repeat) { i →
        set(position + i, nextInt(from, until).toByte())
    }
}

fun Any.asByteArray(length: Int): ByteArray? = when (this) {
    is String → {
        val bytes = toByteArray(Charset.forName("koi8"))
        ByteArray(length) {
            if (it == 0) {
                (bytes.size - 1).toUByte().toByte()
            } else if (it - 1 < bytes.size) {
                bytes[it - 1]
            } else {
                0
            }
        }
    }
    else → null
}
```



-zsh

```
markoutte@Aquarius libs % java -jar fuzzer-parse.jar -c com.alibaba.fastjson2.JS  
ON -m "parse(java.lang.String)" -cp fastjson2-2.0.38.jar
```

0:00 / 0:54



● ● ● ✎⌘2

-zsh

```
New seed added: 0013ea83
New seed added: 004f4513
New seed added: 001535f2
New seed added: 002716fc
New seed added: 0014df79
New seed added: 00210bff
New seed added: 005f2ce3
New seed added: 001dd803
New seed added: 00152ab3
New seed added: 0014b6b1
Traces found: 1981
Errors found: 4
Time elapsed: 10000 ms
markoutte@Aquarius libs % ls | grep 'report*'
reportArrayIndexOutOfBoundsException.txt
reportJSONException.txt
reportNullPointerException.txt
reportNumberFormatException.txt
markoutte@Aquarius libs %
```

● ● ● ↵⌘2

less

java.lang.NullPointerException: Cannot invoke "java.util.List.add(Object)" because "list" is null

```
at com.alibaba.fastjson2.JSONReader.readArray(JSONReader.java:2603)
at com.alibaba.fastjson2.JSONReader.readObject(JSONReader.java:2301)
at com.alibaba.fastjson2.JSONReader.read(JSONReader.java:2122)
at com.alibaba.fastjson2.JSON.parse(JSON.java:67)
at jdk.internal.reflect.GeneratedMethodAccessor1.invoke(Unknown Source)
at java.base/jdk.internal.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.base/java.lang.reflect.Method.invoke(Method.java:568)
at me.markoutte.joker.parse.MainKt.main(Main.kt:60)
```

```
parse: [{"name": {ws4B5@B [1, 23/58>31//A:@?B:3=<7564?B042219@22;14 -^T^R&^Y
) ^S&^_.'&^W^Q^H^_ ^Z^Q+^^^\"%^V^Q+^U, ^M#^X! ^O^L^L ^\^S
-ESC' ^M' "^\^Y^]#^]^\^Y^0-^\^L)('^^S%^\^P-
}]
```

:

Что в итоге?

Coverage guided mutation-based grey-box fuzzer

Что в итоге?

5 ошибок в опенсурс проектах

Как попробовать?



[Markoutte/Joker2023](#)

Генерация тестов для Spring: из чего же, из чего же сделаны тесты



<https://utbot.org>

Денис Фокин, Егор Куликов

Joker, 14 октября, 12:00 – 12:45

Вывести типы из Python: проблемы анализа Python- кода

Екатерина Точилина, Вячеслав Тамарин

PiterPy, 13 ноября, 11:30 – 12:15





Максим Пелевин



markoutte