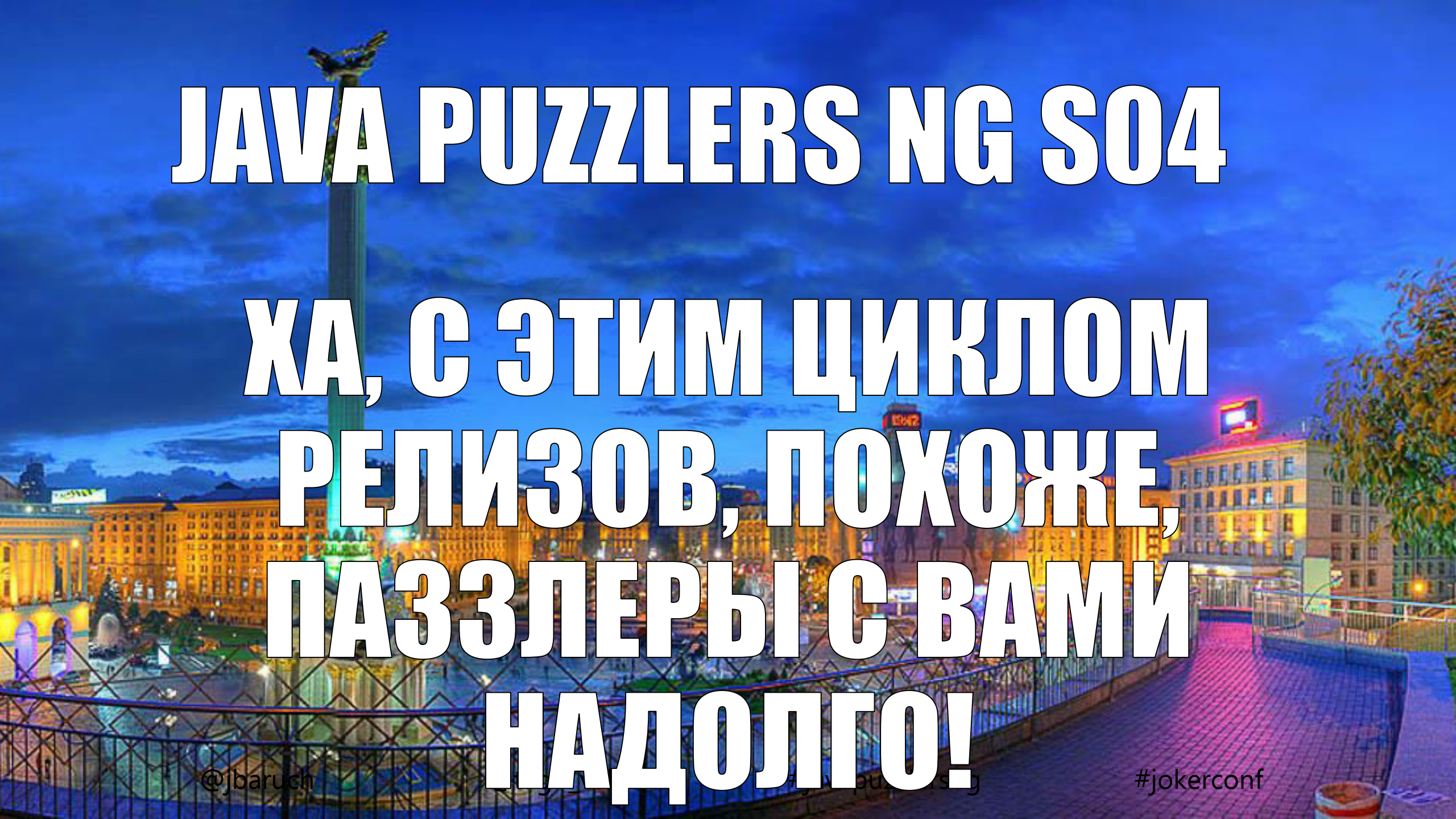

JAVA PUZZLERS NG S04
ХА, С ЭТИМ ЦИКЛОМ РЕЛИЗОВ, ПОХОЖЕ, ПАЗЗЛЕРЫ С ВАМИ НАДОЛГО!
@jbaruch
#jokerconf

CLICK AND HACK THE TYPE-IT BROTHERS

Baruch Sadogursky, JFrog Developer Advocate, @jbaruch

65,460 REPUTATION

10 142 235

Tagir Valeev

Author of StreamEx library.

IntelliJ IDEA Developer.

FindBugs project contributor.

Tech blog in Russian on Habrahabr.

1,263 answers | 12 questions | ~5.9m people reached

Novosibirsk, Russia

amaembo

Member for 3 years, 5 months

5,909 profile views

Last seen 2 days ago

Communities (4)

Stack Overflow	65.5k
Stack Overflow на русском	5k
Code Review	111
English Language & Usage	109

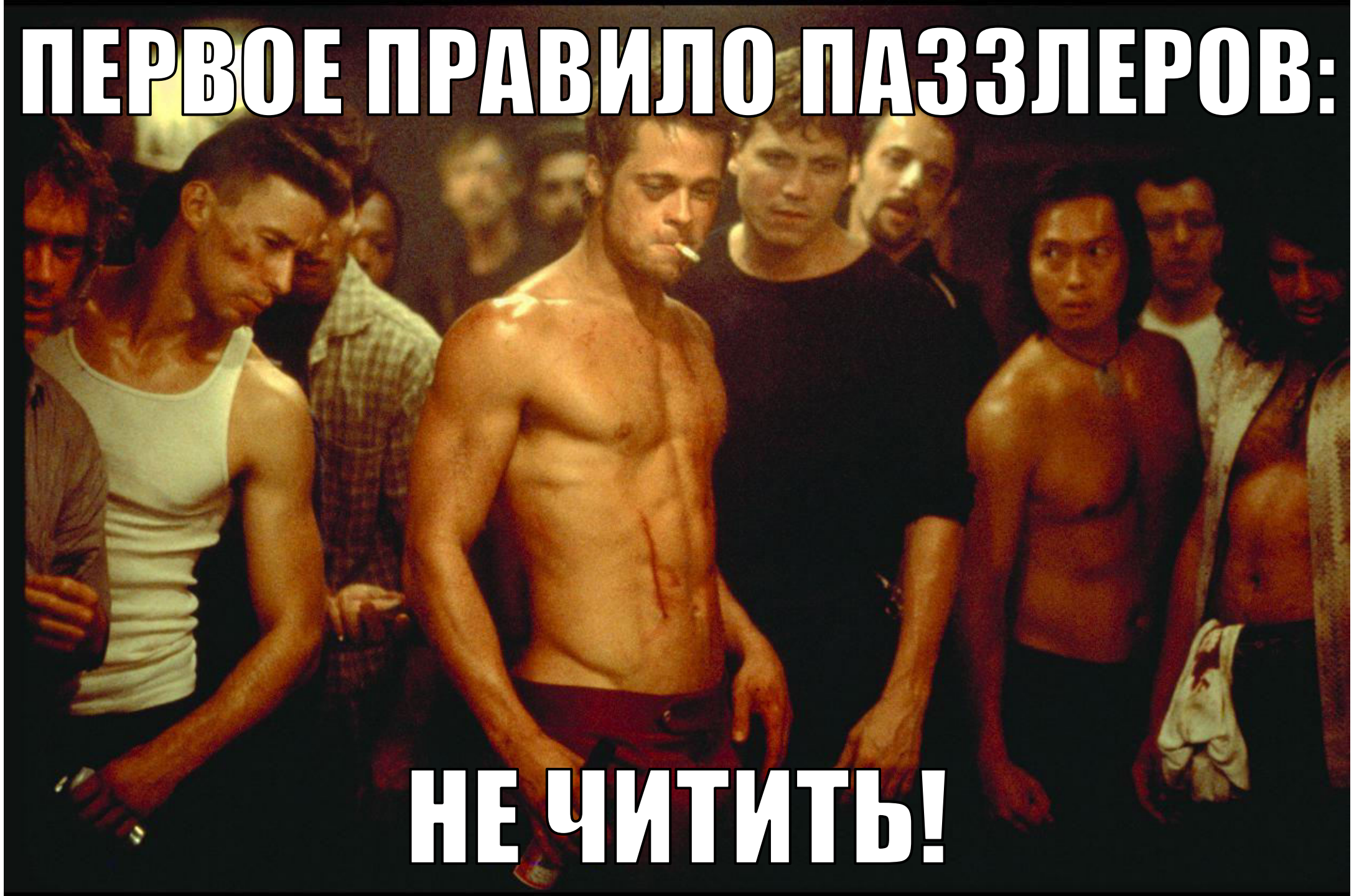
View network profile →

Top Tags (580)

java	SCORE 5,732	POSTS 1,166	POSTS % 91
java-8	SCORE 3,849	POSTS 560	
java-stream	SCORE 2,649	POSTS 425	
lambda	SCORE 778	POSTS 143	
collections	SCORE 292	POSTS 52	
generics	SCORE 252	POSTS 69	

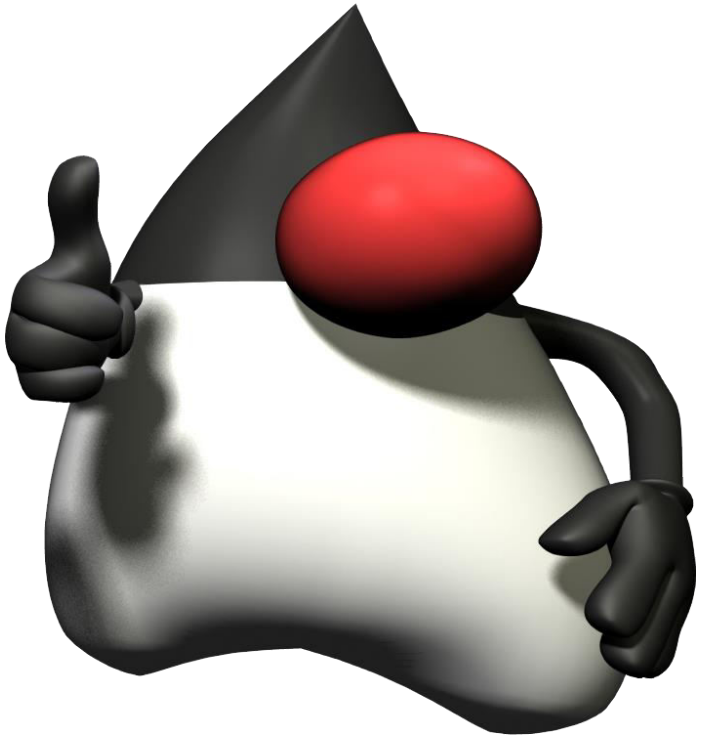
@jbaruch @tagir_valeev #javapuzzlersng #jokerconf

1. Два клевых парня на сцене
2. смешные головоломки
3. Вы голосуете за ответы!
4. Мы кидаемся футболками от JFrog (спасибо им!)
5. Официальные хэштэги: #javapuzzlersng #jokerconf

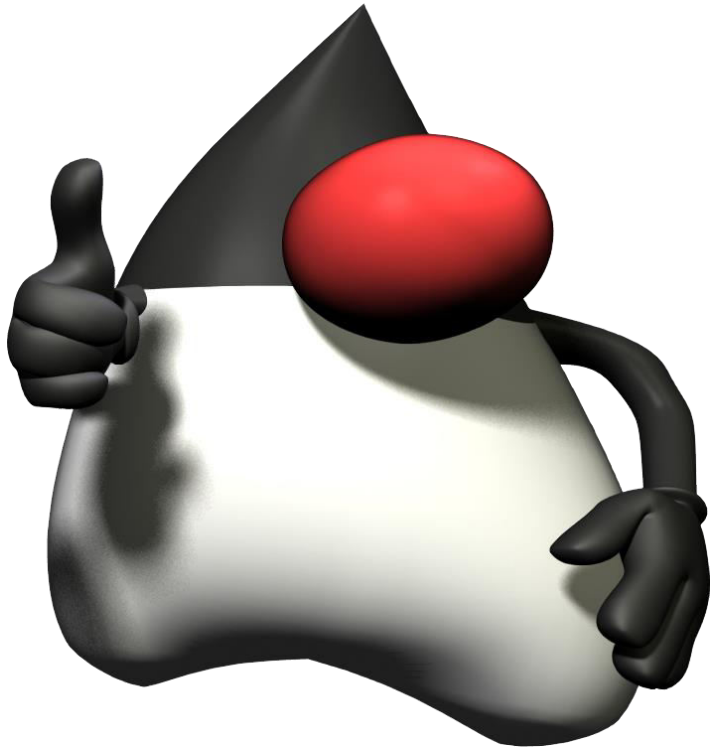

ПЕРВОЕ ПРАВИЛО ПАЗЗЛЕРОВ:
НЕ ЧИТИТЬ!

Проверка системы голосования!
Какая у вас Java?

A. Java 8
B. Java 10
C. Java 11
D. Java 9
E. Java 5
F. Java 2



Готовьтесь, сейчас будет вот так:



Всё работает (ну, или не работает)
на последней версии Java 11!

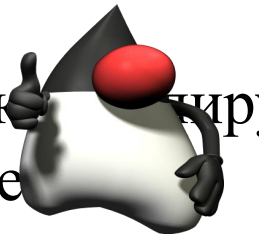
ТРЕТЬИМ БУДЕШЬ?


analyze
this

```
class This {
    final String this;

    This() {
        this.this = This.this.this + "this";
    }

    public static void main(String[] args) {
        System.out.println(new
This().this.length());
    }
}
```

```
class This {
    final String this;

    This() {
        this.this = this.this + "this";
    }

    public static void main(String[] args) {
        System.out.println(new
This().this.length());
    }
}
```

A. Первый компилируется, второй нет
B. Второй компилируется, первый нет
C. Оба компилируются
D. Наркоманству – бой!



```
class This {
    final String this;

    This() {
        this.this = This.this.this + "this";
    }

    public static void main(String[] args) {
        System.out.println(new
This().this.length());
    }
}
```

```
class This {
    final String this;

    This() {
        this.this = this.this + "this";
    }

    public static void main(String[] args) {
        System.out.println(new
This().this.length());
    }
}
```

A. Первый к[illegible]руется, второй нет

B. Второй компилируется, первый нет
C. Оба компилируются
D. Наркоманству – бой!

Aleksey Shipilëv @shipilev · 22 Jul 2014

Instead of "klass", "clazz", "clas" and other atrocities, I'm using "class" next time (notice the Cyrillic "c"). #readability #russiancode

16 80 134

Igal Tabachnik

@hmemcpy

Following

Replying to @shipilev

@shipilev @jbaruch ReSharper team uses 'this' sometimes (lowercase Ukrainian i)

8:33 AM - 22 Jul 2014

9 Retweets **5** Likes

2 9 5

analyze that

```
class This {
    final String this;

    This() {
        this.this = This.this.this + "this";
    }

    public static void main(String[] args) {
        System.out.println(new
This().this.length());
    }
}
```

A. Runtime Exception

B. 4

D. Не скомпилируется (да, я тормоз)


YAHOO!

```
class This {
    final String this;

    This() {
        this.this = This.this.this + "this";
    }

    public static void main(String[] args) {
        System.out.println(new
This().this.length());
    }
}
```

A. Runtime Exception

B. 4

D. Не скомпилируется (да, я тормоз)

15.8.4. Qualified `this`

Any lexically enclosing instance (§8.1.3) can be referred to by explicitly qualifying the keyword `this`.

Let T be the type denoted by *TypeName*. Let *n* be an integer such that T is the *n*'th lexically enclosing type declaration of the class or interface in which the qualified `this` expression appears.

The value of an expression of the form *TypeName*`.this` is the *n*'th lexically enclosing instance of `this`.

The type of the expression is T.

It is a compile-time error if the expression occurs in a class or interface which is not an inner class of class T or T itself.

Chapter 16. Definite Assignment

Each local variable (§14.4) and every blank `final` field (§4.12.4, §8.3.1.2) must have a *definitely assigned* value when any access of its value occurs.

An access to its value consists of the simple name of the variable (or, for a field, the simple name of the field qualified by `this`) occurring anywhere in an expression except as the left-hand operand of the simple assignment operator = (§15.26.1).

For every access of a local variable or blank `final` field `x`, `x` must be definitely assigned before the access, or a compile-time error occurs.

Similarly, every blank `final` variable must be assigned at most once; it must be *definitely unassigned* when an assignment to it occurs.

Such an assignment is defined to occur if and only if either the simple name of the variable (or, for a field, its simple name qualified by `this`) occurs on the left hand side of an assignment operator.

Как я понимаю блокчейн и Биткоин

- Что-то крутится-вертится-майнится
- Проц нагружен
- Комп греется
- Кому-то от этого хорошо
- Больше ничего не понятно

Что из этого не компилируется?

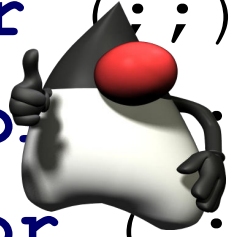
A. `for (;;) {;;}`

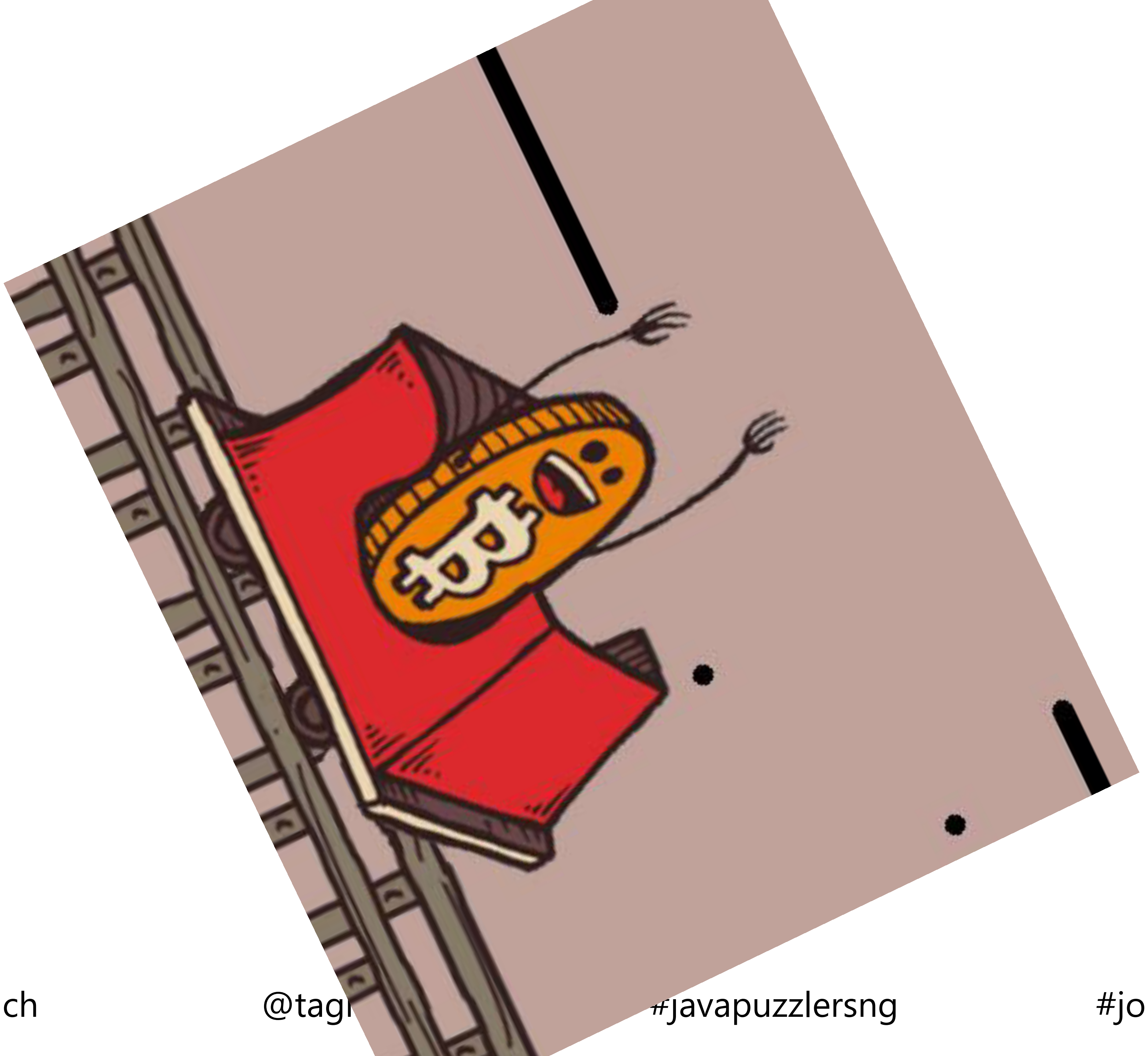
B. `<SEP> for (;;) ;;`

C. `<SEP>{;} for (;;) {;}`

D. `<SEP> ; for (;;) ;`

E. Норм всё, чо!





Что из этого не компилируется?

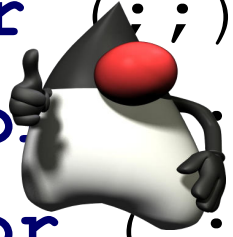
A. `for (;;) {;;}`

B. `<SEP> for (;;) ;;`

C. `<SEP>{;} for (;;) {;}`

D. `<SEP> ; for (;;) ;`

E. Норм всё, чо!



Что из этого не компилируется?

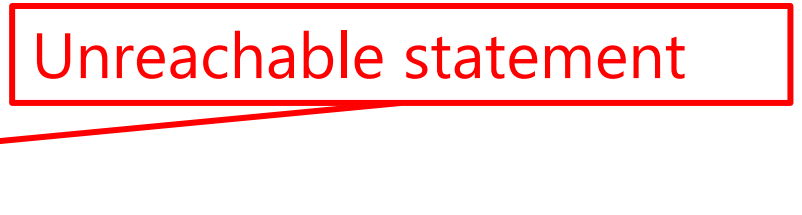
A. `for (;;) {;;}`

B. `<SEP> for (;;) ;;`

C. `<SEP>{;} for (;;) {;}`

D. `<SEP> ; for (;;) ;`

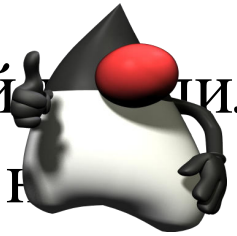
E. Норм всё, чо!



```
public static void main(String[] args) {
    final boolean rick;

    if(true && (rick = true)) {}
    if(false && (rick = false)) {}
    System.out.println(rick);
}
```

```
public static void main(String[] args) {
    final boolean rick;

    if(false && (rick = false)) {}
    if(true && (rick = true)) {}
    System.out.println(rick);
}
```

A. Первый [illegible]илируется, второй н[illegible]

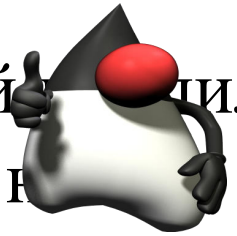


B. Второй компилируется, первый нет
C. Оба компилируются
D. Наркоманству – бой!

```
public static void main(String[] args) {
    final boolean rick;

    if(true && (rick = true)) {}
    if(false && (rick = false)) {}
    System.out.println(rick);
}
```

```
public static void main(String[] args) {
    final boolean rick;

    if(false && (rick = false)) {}
    if(true && (rick = true)) {}
    System.out.println(rick);
}
```

A. Первый [illegible]илируется, второй н[illegible]



B. Второй компилируется, первый нет
C. Оба компилируются
D. Наркоманству – бой!

Table of Contents

Chapter 16. Definite Assignment

Each local variable (§14.4) and every blank `final` field (§4.12.4, §8.3.1.2) must have a *definitely assigned* value when any access of its value occurs.

An access to its value consists of the simple name of the variable (or, for a field, the simple name of the field qualified by `this`) occurring anywhere in an expression except as the left-hand operand of the simple assignment operator = (§15.26.1).

For every access of a local variable or blank `final` field x, x must be definitely assigned before the access, or a compile-time error occurs.

Similarly, every blank `final` variable must be assigned at most once; it must be *definitely unassigned* when an assignment to it occurs.

Such an assignment is defined to occur if and only if either the simple name of the variable (or, for a field, its simple name qualified by `this`) occurs on the left hand side of an assignment operator.

For every assignment to a blank `final` variable, the variable must be definitely unassigned before the assignment, or a compile-time error occurs.

The remainder of this chapter is devoted to a precise explanation of the words "definitely assigned before" and "definitely unassigned before".

The idea behind definite assignment is that an assignment to the local variable or blank `final` field must occur on every possible execution path to the access. Similarly, the idea behind definite unassignment is that no other assignment to the blank `final` variable is permitted to occur on any possible an assignment.

The analysis takes into account the structure of statements and expressions; it also provides a special treatment of the expression operators `!`, `&&`, `||`, and `? :`, and of boolean-valued constant expressions.

Except for the special treatment of the conditional boolean operators `&&`, `||`, and `? :` and of boolean-valued constant expressions, the values of expressions are not taken into account in the flow analysis.

Example 16-1. Definite Assignment Considers Structure of Statements and Expressions

A Java compiler recognizes that k is definitely assigned before its access (as an argument of a method invocation) in the code:

```
{
    int k;
    if (v > 0 && (k = System.in.read()) >= 0)
        System.out.println(k);
}
```

because the access occurs only if the value of the expression:

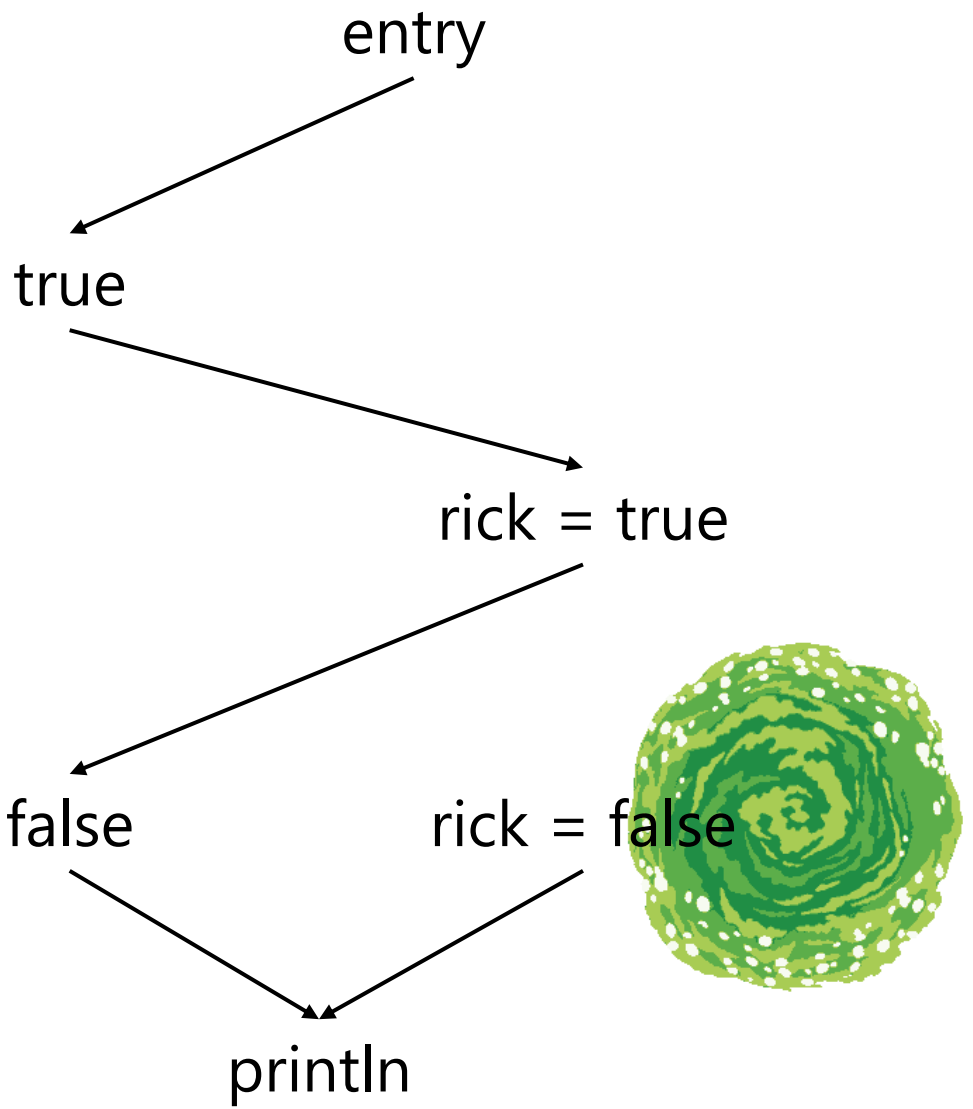
```
v > 0 && (k = System.in.read()) >= 0
```

is `true`, and the value can be `true` only if the assignment to k is executed (more properly, evaluated).

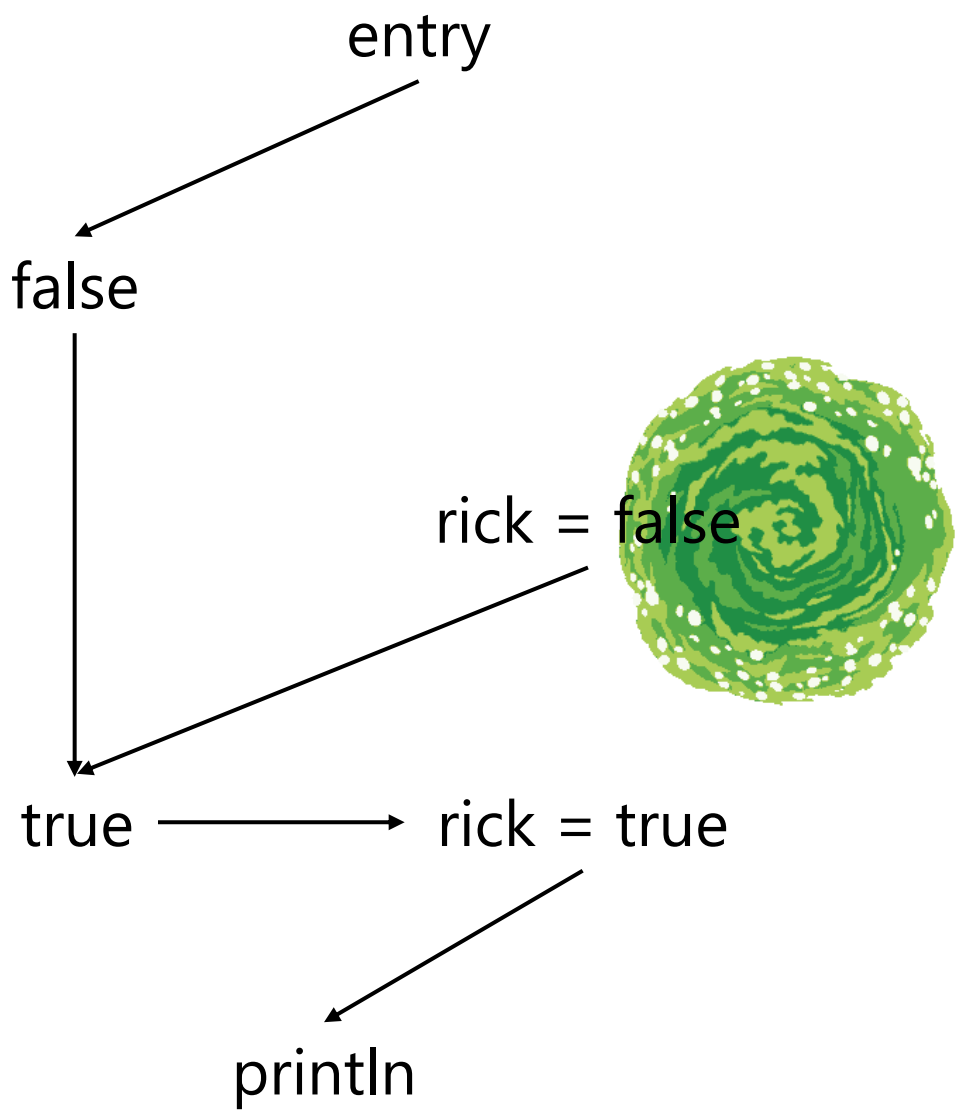
Similarly, a Java compiler will recognize that in the code:

```
{
    int k;
    while (true) {
        k = n;
        if (k >= 5) break;
        n = 6;
    }
    System.out.println(k);
}
```

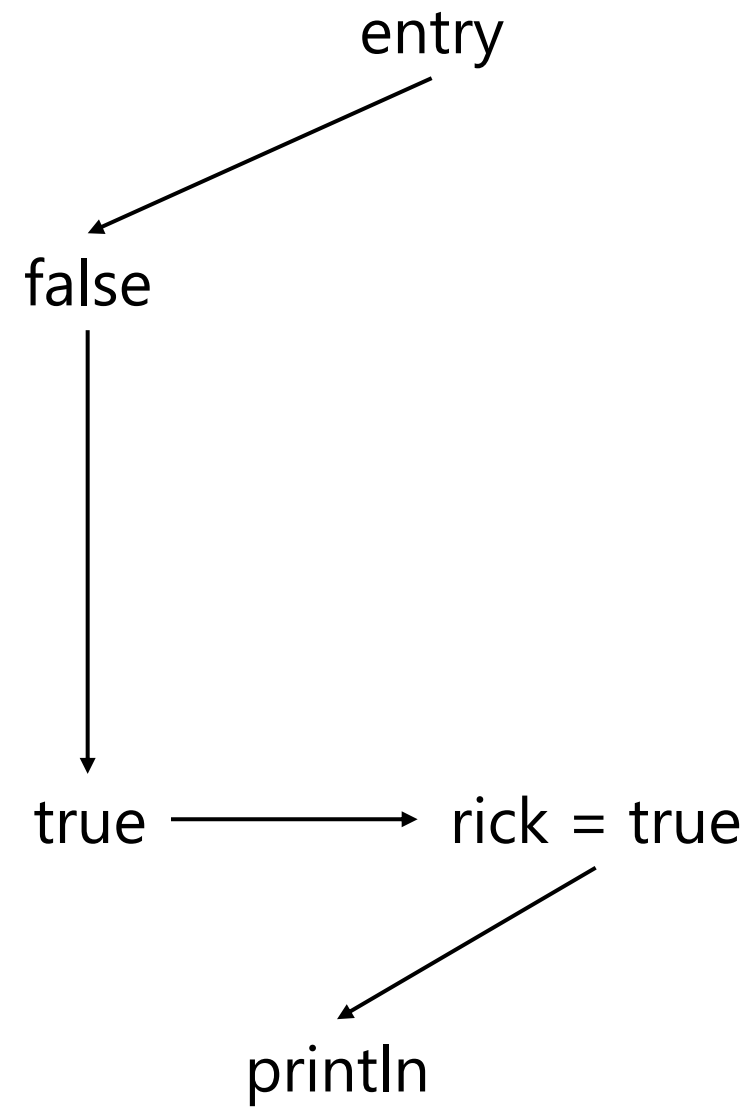
```
public static void main(String[] args) {
    final boolean rick;

    if(true && (rick = true)) {}
    if(false && (rick = false)) {}
    System.out.println(rick);
}
```



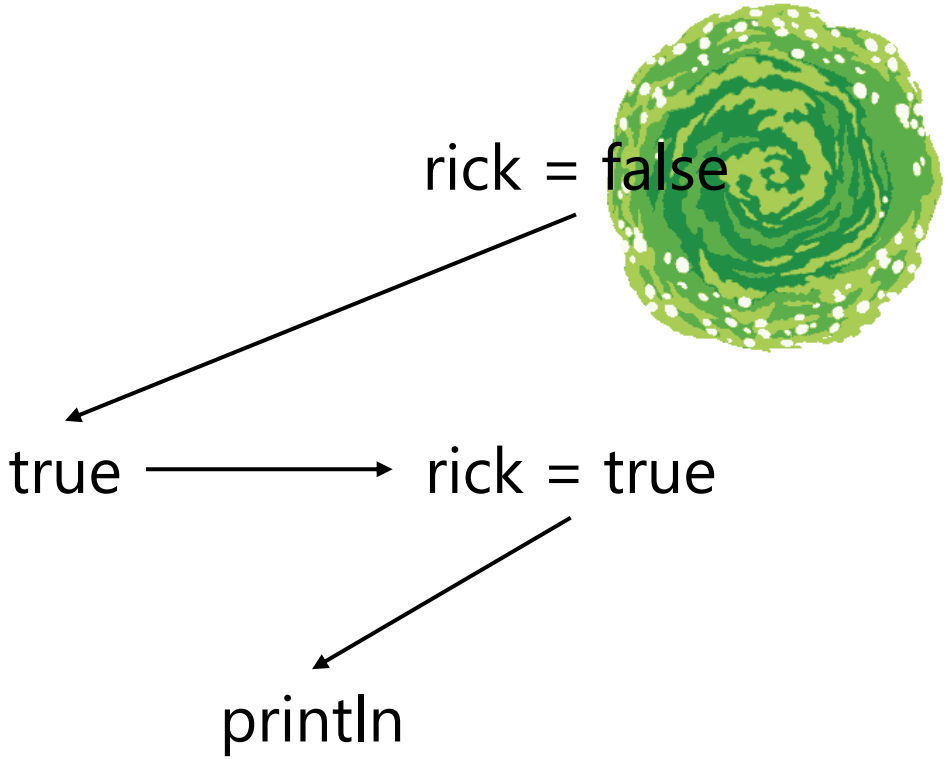
```
public static void main(String[] args) {
    final boolean rick;

    if(false && (rick = false)) {}
    if(true && (rick = true)) {}
    System.out.println(rick);
}
```



```java
public static void main(String[] args) {
    final boolean rick;

    if(false && (rick = false)) {}
    if(true && (rick = true)) {}
    System.out.println(rick);
}
```



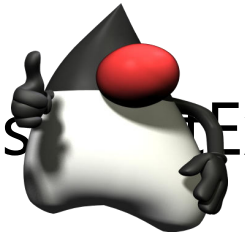
```
public static void main(String[] args) {
    final boolean rick;

    if(false && (rick = false)) {}
    if(true && (rick = true)) {}
    System.out.println(rick);
}
```




ANY COLOUR
YOU LIKE
AS LONG AS IT'S
BLACK

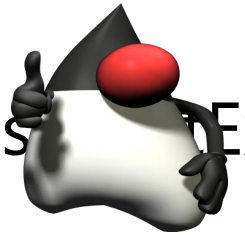
```
public static void main(String[] args) {
    System.out.println(▯▯▯▯▯▯▯▯▯▯▯▯("бирюзовый"));
}

static <Цвет> Цвет заказМашины(Цвет цвет) {
    class ModelT extends RuntimeException {
        Цвет цвет;

        ModelT(Цвет цвет) {
            this.цвет = цвет; // 1
        }
    }
    if (цвет.equals(0x000000)) throw new ModelT(цвет); // 2
    try {
        ▯▯▯▯▯▯▯▯▯▯▯▯(0x000000);
    } catch (ModelT car) {
        цвет = car.цвет; // 3
    }
    return цвет;
}
```

A.Не скомпилируется
B.Выдаст пустую строку
C.Выдаст 0
D.Упадёт с Clas Exception



```
public static void main(String[] args) {
    System.out.println(▯▯▯▯▯▯▯▯▯▯▯▯("бирюзовый"));
}

static <Цвет> Цвет заказМашины(Цвет цвет) {
    class ModelT extends RuntimeException {
        Цвет цвет;

        ModelT(Цвет цвет) {
            this.цвет = цвет; // 1
        }
    }
    if (цвет.equals(0x000000)) throw new ModelT(цвет); // 2
    try {
        ▯▯▯▯▯▯▯▯▯▯▯▯(0x000000);
    } catch (ModelT car) {
        цвет = car.цвет; // 3
    }
    return цвет;
}
```

A.Не скомпилируется
B.Выдаст пустую строку
C.Выдаст 0
D.Упадёт с Clas Exception



```
public static void main(String[] args) {
    System.out.println(□□□□□□□□□□□□("бирюзовый"));
}

static <Цвет> Цвет заказМашины(Цвет цвет) {
    class ModelT extends RuntimeException {
        Цвет цвет;

        ModelT(Цвет цвет) {
            this.цвет = цвет; // 1
        }
    }
    if (цвет.equals(0x000000)) throw new ModelT(цвет); // 2
    try {
        □□□□□□□□□□□□(0x000000);
    } catch (ModelT car) {
        цвет = car.цвет; // 3
    }
    return цвет;
}
```

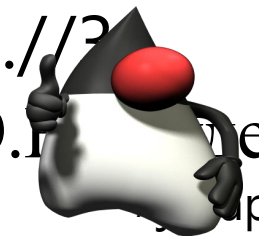
Предупреждение про unchecked:

A.//1

B.//2

C.//3

D.[illegible]ет



```
public static void main(String[] args) {
    System.out.println(□□□□□□□□□□□□("бирюзовый"));
}

static <Цвет> Цвет заказМашины(Цвет цвет) {
    class ModelT extends RuntimeException {
        Цвет цвет;

        ModelT(Цвет цвет) {
            this.цвет = цвет; // 1
        }
    }
    if (цвет.equals(0x000000)) throw new ModelT(цвет); // 2
    try {
        □□□□□□□□□□□□(0x000000);
    } catch (ModelT car) {
        цвет = car.цвет; // 3
    }
    return цвет;
}
```

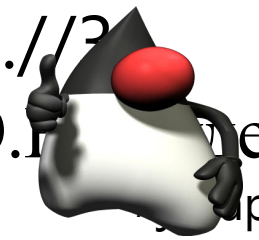
Предупреждение про unchecked:

A.//1

B.//2

C.//3

D.[illegible]ет



Menu

Home
About
Irssi
Security
Puzzles

Recent Posts

The Java type system is broken
More serialization hacks with AnnotationInvocationHandler
Java Puzzle 9: Tweet - Solution
Java Puzzle 9: Tweet
Resurrecting a PhantomReference

Archives

May 2018
November 2015
February 2015
January 2015
August 2012
March 2012
February 2012
September 2011
June 2011
May 2011

May 16 2018

The Java type system is broken

The goal of a programming language's type system is to provide guarantees about what types can be encountered at certain points in your program; to document the values we expect to see in a way that the compiler can verify. But Java's type system has holes, the guarantees can be broken. Some of these holes are deliberate, mostly for backward compatibility. For

ution, leading to a

t when you take its
ot be expressed in
that wildcard; e.g.

a `List<String>` and

restrictions such as
ond `a`. That makes

executed multiple times, e.g in a for-loop. So how can that type represent only a single execution? The trick is that capture types only exist in the scope of a single expression; their scope is so narrow that within that scope the expression can only be executed once. A capture type cannot escape outside the body of the for-loop in which it is created, because that would require

JDK / JDK-8203337

5.1.6.2: Local inner class in generic method should be treated as generic

Export

Details

Type:	Bug	Status:	OPEN
Priority:	P3	Resolution:	Unresolved
Affects Version/s:	8, 9, 10, 10.0.1, 11	Fix Version/s:	tbd_major
Component/s:	specification		
Labels:	dcsfai jls-types reproducer-yes webbug		
Subcomponent:	language		
CPU:	generic		
OS:	generic		

People

Assignee:	Dan Smith
Reporter:	Webbug Group
Votes:	0 Vote for this issue
Watchers:	4 Start watching this issue

Dates

Created:	2018-05-16 22:10
Updated:	2018-05-17 11:25

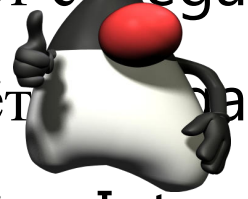
```java
public class Main {
    public static void main(String[] args) throws InterruptedException {
        new Object().wait(9223372036854775807L, 1);
    }
}
```

A. Зависнет на много лет

B. Упадёт с IllegalMonitorStateException

C. Упадёт с IllegalArgumentException: timeout value is negative

D. Упадёт с InterruptedException




THE
CW

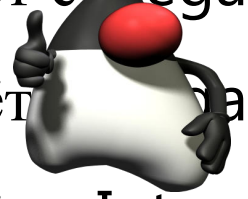
```java
public class Main {
    public static void main(String[] args) throws InterruptedException {
        new Object().wait(9223372036854775807L, 1);
    }
}
```

A. Зависнет на много лет

B. Упадёт с IllegalMonitorStateException

C. Упадёт с IllegalArgumentException: timeout value is negative

D. Упадёт с InterruptedException



JDK / JDK-8210787

Object.wait(long, int) throws inappropriate IllegalArgumentException

Details

Type:	Bug	Status:	RESOLVED
Priority:	P4	Resolution:	Fixed
Affects Version/s:	None	Fix Version/s:	12
Component/s:	core-libs		
Labels:	None		
Subcomponent:	java.lang		
Resolved In Build:	b12		

Description

In an extreme case of obj.wait(Long.MAX_VALUE, nanos) with nanos > 0 it throws "IllegalArgumentException: timeout value is negative" due to the long integer overflow.

People

Assignee:	Ivan Gerasimov
Reporter:	Ivan Gerasimov
Votes:	0 Vote for this issue
Watchers:	2 Start watching this issue

Dates

Created:	2018-09-14 22:27
Updated:	2018-09-19 15:19
Resolved:	2018-09-15 13:52

@jbaruch @tagir_valeev #javapuzzlersng #jokerconf

WTF, Java(script?)

baNaNa

```
"b" + "a" + +"a" + "a";
```

This is an old-school joke in JavaScript, but remastered. Here's the original one:

```
"foo" + +"bar"; // -> 'fooNaN'
```

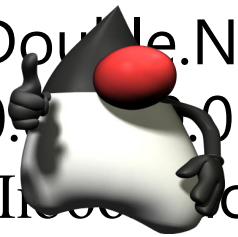
`NaN` is not a `NaN`

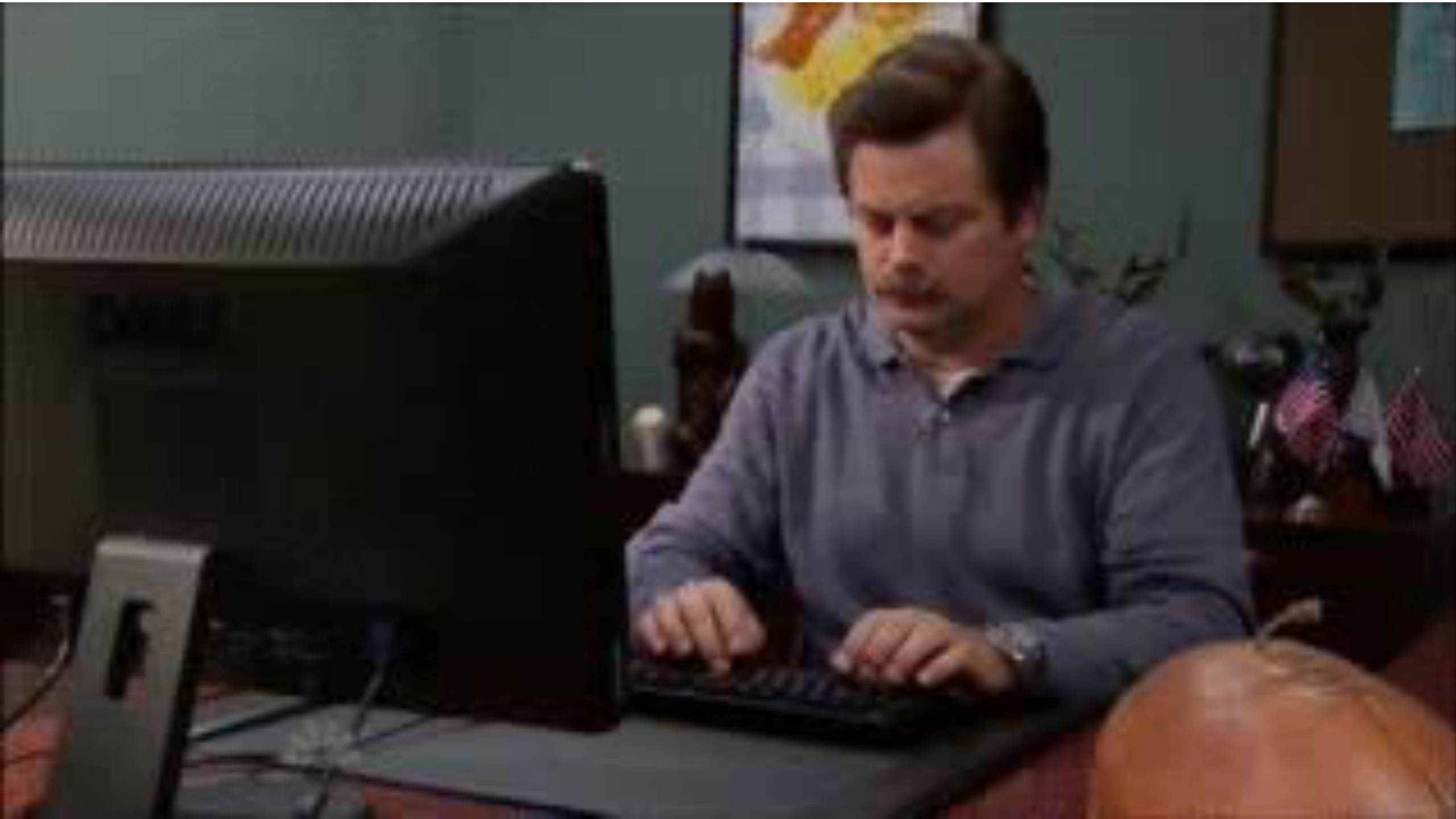
```
NaN === NaN; // -> false
```

Что напечатает true?

```
private static void doubleWtf(Double wtf1, Double wtf2) {
    if (!wtf1.equals(wtf2)) {
        DoubleBuffer dWtf1 = DoubleBuffer.allocate(1).put(wtf1);
        DoubleBuffer dWtf2 = DoubleBuffer.allocate(1).put(wtf2);
       System.out.println(dWtf1.equals(dWtf2));
    }
}
```

A. Double.NaN и Double.NaN
B. Double.NaN и любой double
C. 0.0 и -0.0
D. Любой double и null
E. Ничего

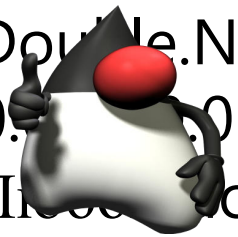




Что напечатает true?

```
private static void doubleWtf(Double wtf1, Double wtf2) {
    if (!wtf1.equals(wtf2)) {
        DoubleBuffer dWtf1 = DoubleBuffer.allocate(1).put(wtf1);
        DoubleBuffer dWtf2 = DoubleBuffer.allocate(1).put(wtf2);
       System.out.println(dWtf1.equals(dWtf2));
    }
}
```

A. Double.NaN и Double.NaN
B. Double.NaN и любой double
C. 0.0 и -0.0
D. Любой double и null
E. Ничего



RTFM

```
java.nio.DoubleBuffer
@Contract(value = "null -> false", pure = true)
public boolean equals(@Nullable Object ob)
```

Tells whether or not this buffer is equal to another object.
Two double buffers are equal if, and only if,

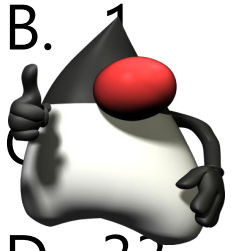
1. They have the same element type,
2. They have the same number of remaining elements, and
3. The two sequences of remaining elements, considered [illegible] equal. This method considers two double elements a and b to be equal if `(a == b) || (Double.isNaN(a) && Double.isNaN(b))`. The values `-0.0` and `+0.0` are considered to be equal, unlike `Double.equals(Object)`.

A double buffer is not equal to any other type of object.

```
Set<Character> set = new HashSet<>();
for(char ch = 'а'; ch <= 'я'; ch++) {
    set.add(ch);
    set.remove(ch - 1);
}
System.out.println(set.size());
```

A. 0

B. 1



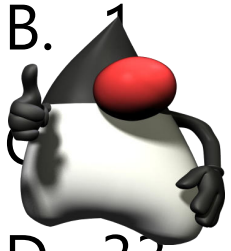
D. 33

E. Упадёт с исключением

```
Set<Character> set = new HashSet<>();
for(char ch = 'а'; ch <= 'я'; ch++) {
    set.add(ch);
    set.remove(ch - 1);
}
System.out.println(set.size());
```

A. 0

B. 1



D. 33

E. Упадёт с исключением

15.18.2. Additive Operators (+ and –) for Numeric Types

The binary + operator performs addition when applied to two operands of numeric type, producing the sum of the operands.

The binary – operator performs subtraction, producing the difference of two numeric operands.

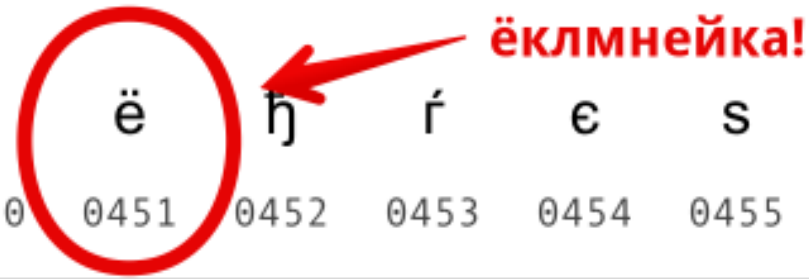
Binary numeric promotion is performed on the operands (§5.6.2).

5.6.2. Binary Numeric Promotion

When an operator applies *binary numeric promotion* to a pair of operands, each of which must denote a value that is convertible to a numeric type, the following rules apply, in order:

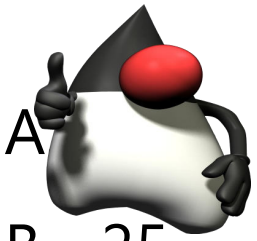
1. If any operand is of a reference type, it is subjected to unboxing conversion (§5.1.8).
2. Widening primitive conversion (§5.1.2) is applied to convert either or both operands as specified by the following rules:
   - If either operand is of type `double`, the other is converted to `double`.
   - Otherwise, if either operand is of type `float`, the other is converted to `float`.
   - Otherwise, if either operand is of type `long`, the other is converted to `long`.
   - Otherwise, both operands are converted to type `int`.

а	б	в	г	д	е	ж	з	и	й	к	л	м	н	о	п
0430	0431	0432	0433	0434	0435	0436	0437	0438	0439	043A	043B	043C	043D	043E	043F
р	с	т	у	ф	х	ц	ч	ш	щ	ъ	ы	ь	э	ю	я
0440	0441	0442	0443	0444	0445	0446	0447	0448	0449	044A	044B	044C	044D	044E	044F
ѐ	ё	ђ	ѓ	є	ѕ	і	ї	ј	љ	њ	ћ	ќ	ѝ	ў	џ
0450	0451	0452	0453	0454	0455	0456	0457	0458	0459	045A	045B	045C	045D	045E	045F



Var of thrones

```java
final var targaryens = 2;
final var lannisters = 25;
final var starks = '1';
final var snow = true ? targaryens * lannisters : starks;
System.out.println(snow);
```

A 

B. 25

C. 1

D. 50

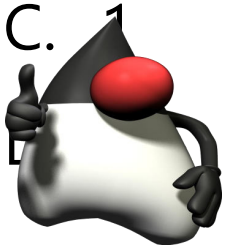
```
final var targaryens = 2;
final var lannisters = 25;
final var starks = '1';
final var snow = true ? targaryens * lannisters : starks;
System.out.println(snow);
```

A. 2

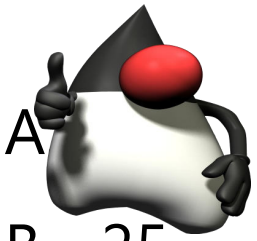
B. 25

C. 1

D.



```java
final var targaryens = 2;
final var lannisters = 25;
final var starks = '1';
final var snow = true ? targaryens * lannisters : starks;
System.out.println(snow);
```

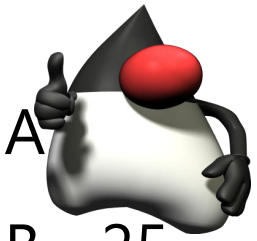
A 

B. 25

C. 1

D. 50

```java
final int targaryens = 2;
final int lannisters = 25;
final char starks = '1';
final char snow = true ? targaryens * lannisters : starks;
System.out.println(snow);
```

A 

B. 25

C. 1

D. 50

15.25.2. Numeric Conditional Expressions

Numeric conditional expressions are standalone expressions (§15.2).

The type of a numeric conditional expression is determined as follows:

- If the second and third operands have the same type, then that is the type of the conditional expression.
- If one of the second and third operands is of primitive type T, and the type of the other is the result of applying boxing conversion (§5.1.7) to T, then the type of the conditional expression is T.
- If one of the operands is of type `byte` or `Byte` and the other is of type `short` or `Short`, then the type of the conditional expression is `short`.
- If one of the operands is of type T where T is `byte`, `short`, or `char`, and the other operand is a constant expression (§15.28) of type `int` whose value is representable in type T, then the type of the conditional expression is T.
- If one of the operands is of type T, where T is `Byte`, `Short`, or `Character`, and the other operand is a constant expression of type `int` whose value is representable in the type U which is the result of applying unboxing conversion to T, then the type of the conditional expression is U.

Выводы


WITH GREAT
POWER
COMES GREAT
RESPONSIBILITY
PETER PARKER is SPIDER MAN

-Пишите читаемый код!
-Комментируйте все трюки
-Иногда даже в джаве
(в этом сезоне аж два
-Статические анализа
рулят! IntelliJ IDEA!
-RTFM!

- Сами понимаете, больше релизов — больше — больше паззлеров!

- Если вы наткнулись на паззлер, давайте его давайте его сюда!

- puzzlers+java@jfrog.com

-Понравилось?
-Хвалите нас скорее в твиттере!

-#javapuzzlersng **#jokerconf**
-@tagir_valeev
-@jbaruch

-Не понравилось?
-/dev/null