A close-up photograph of a fluffy orange hamster with white markings on its face and chest. The hamster is looking directly at the camera with large, dark eyes. It has long, thin whiskers extending from its muzzle. The background is dark and out of focus.

**Байткод для
любознательных**

@antonarhipov

Joker 2016, СПб

Антон Архипов

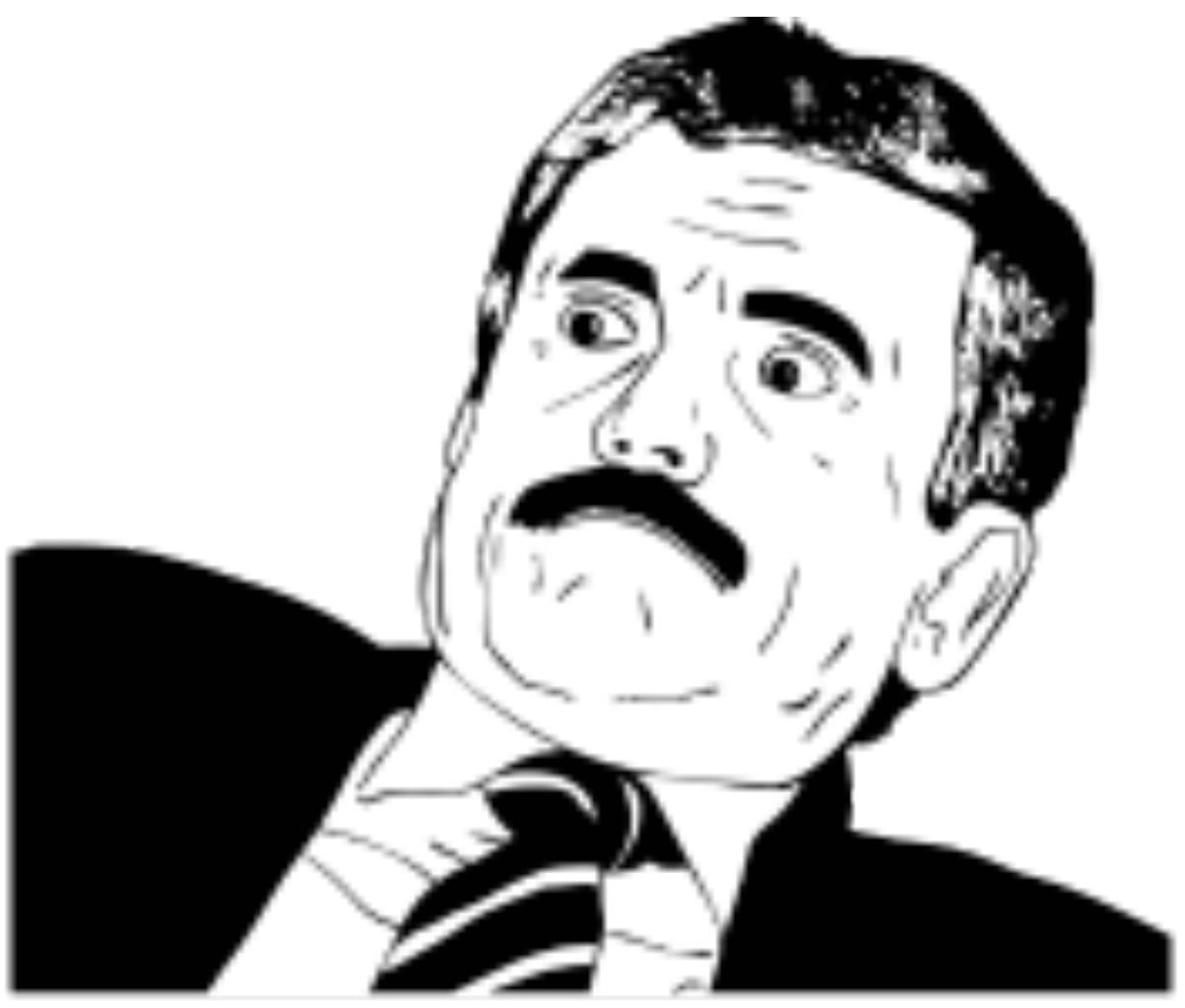
@antonarhipov

ZEROTURNAROUND

JRebel XRebel



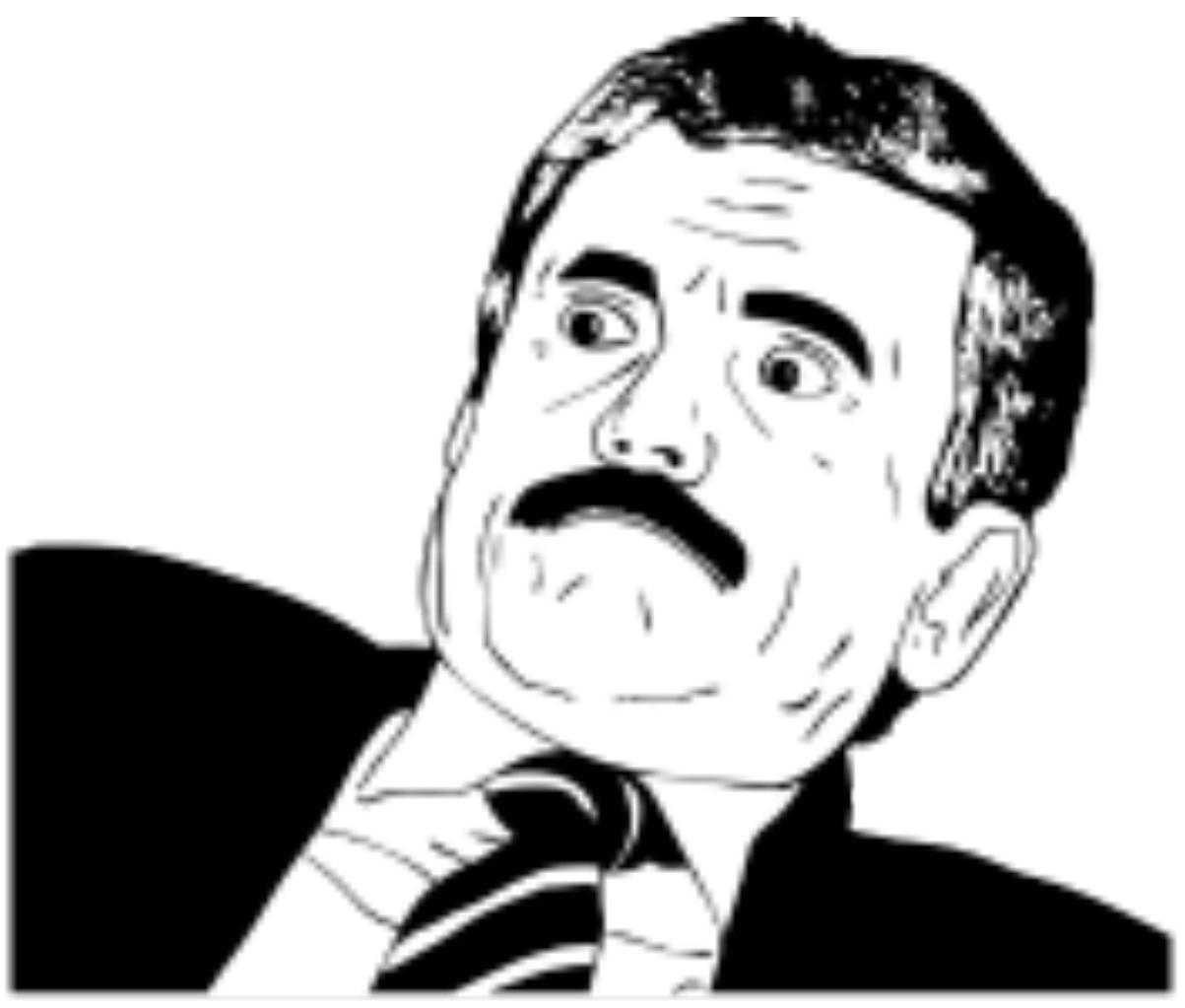
Зачем?



BUT...WHY ?!

Зачем?

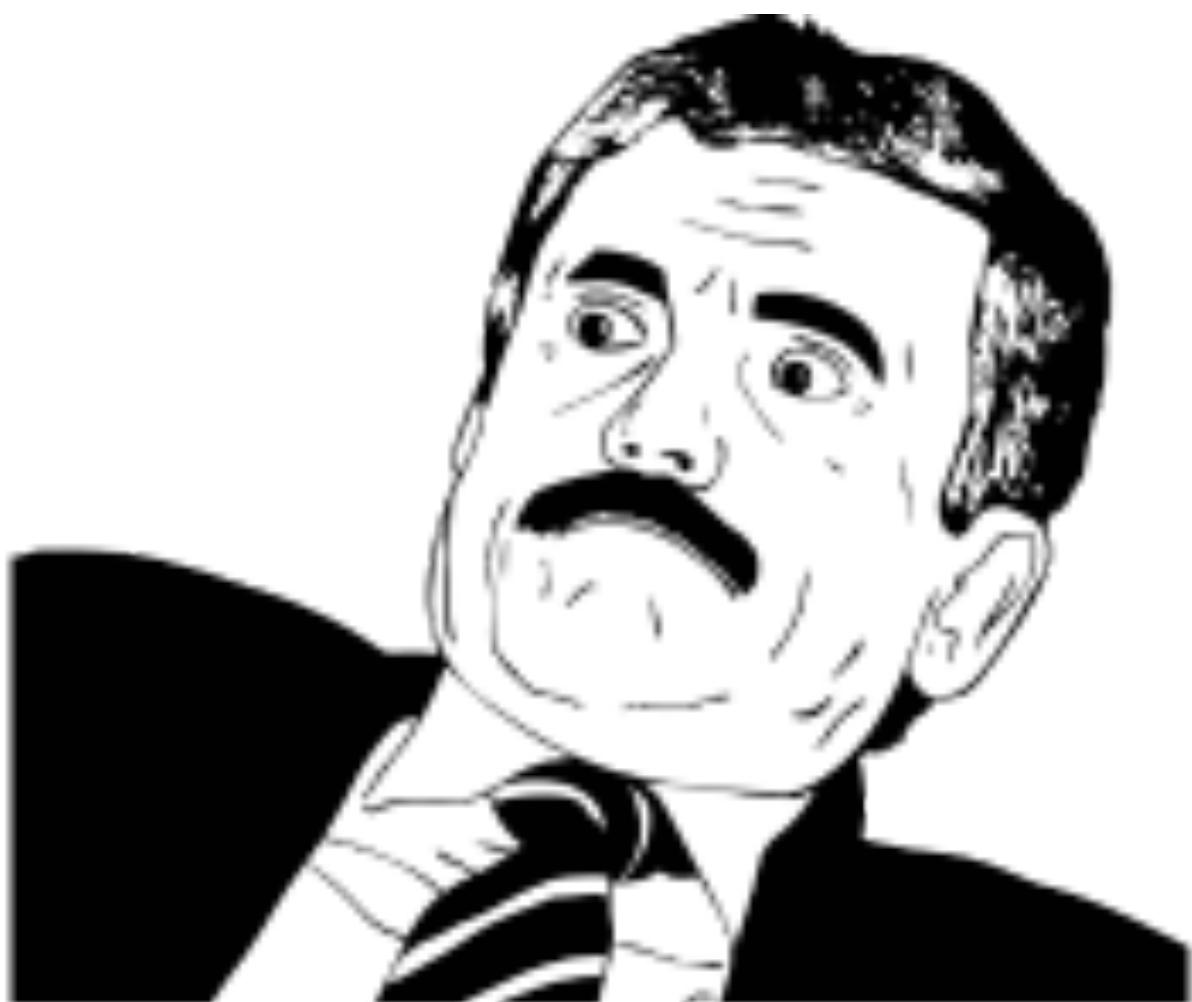
- Знай свою платформу! 



BUT...WHY ?!

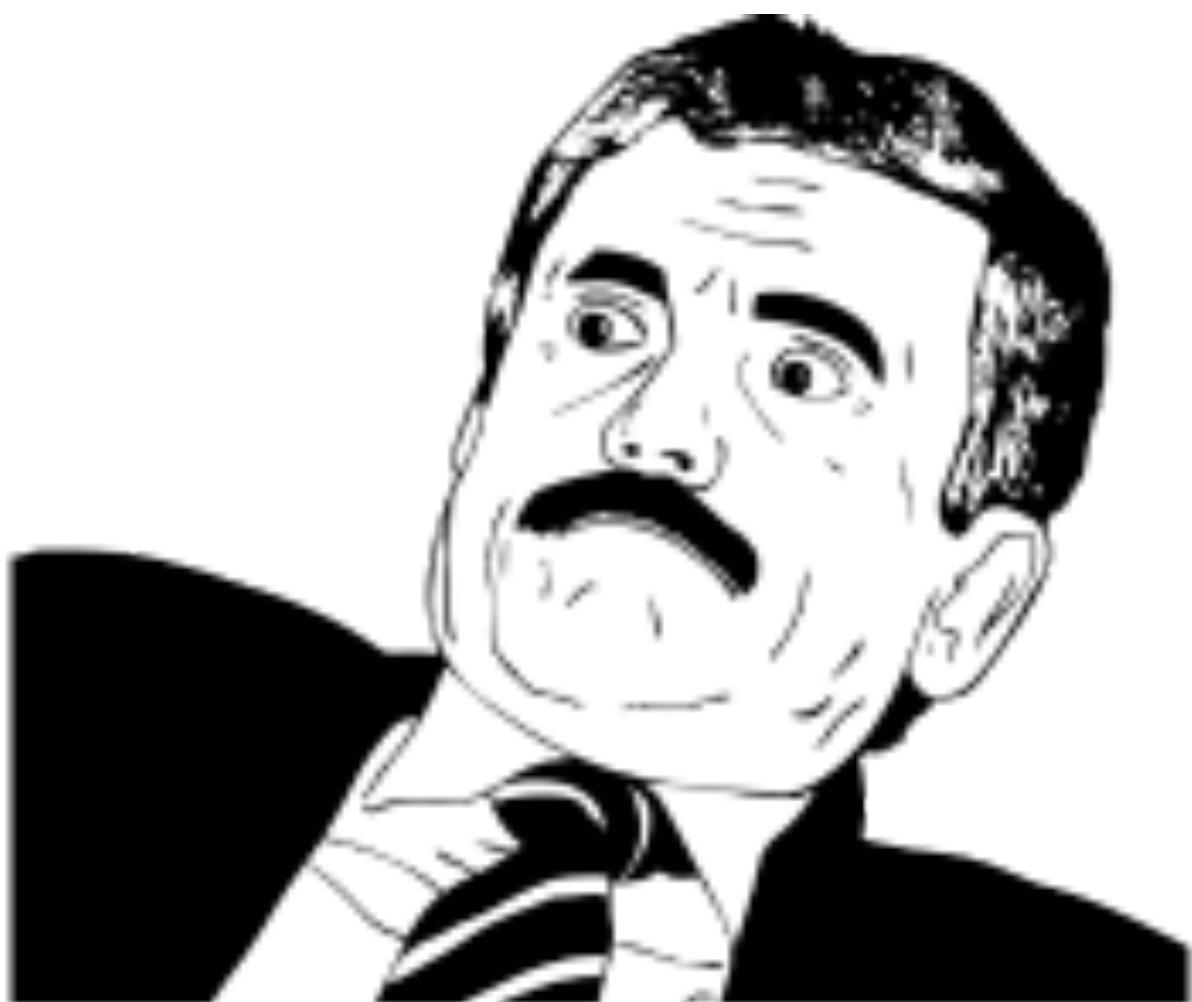
Зачем?

- Знай свою платформу! 
- Хотите написать свой компилятор?



Зачем?

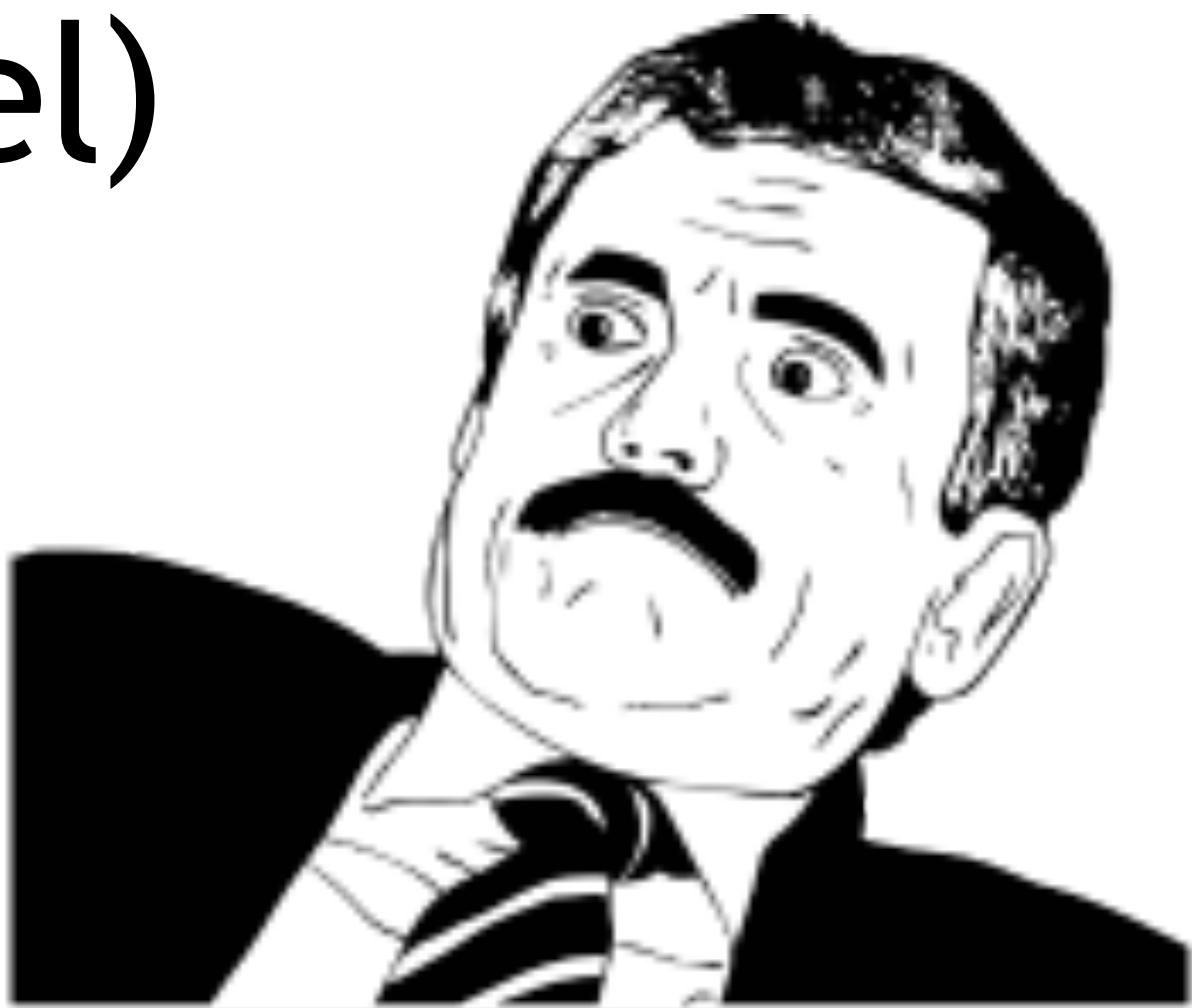
- Знай свою платформу! 
- Хотите написать свой компилятор?
- Фреймворки (AOP, ORM)



BUT...WHY ?!

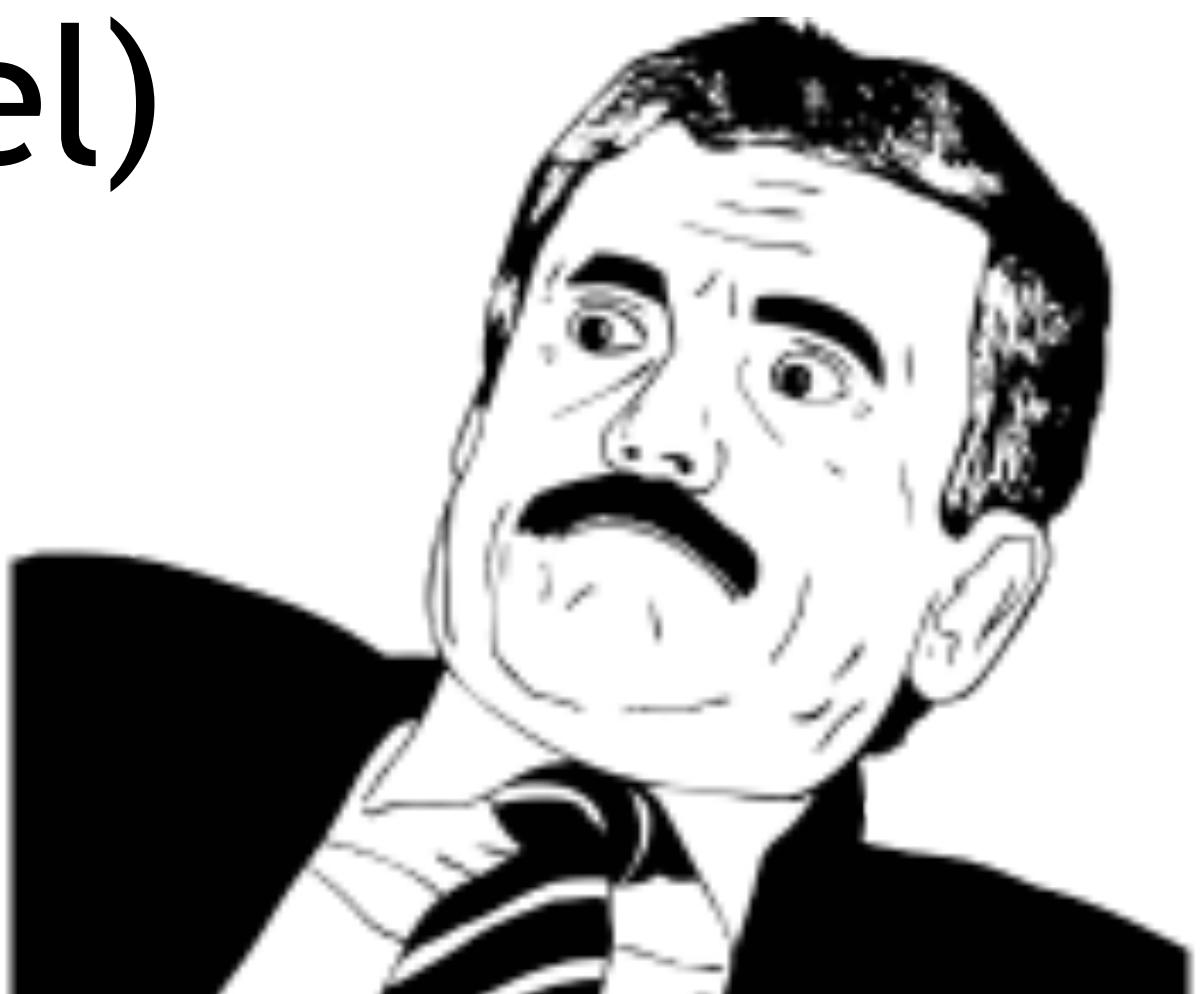
Зачем?

- Знай свою платформу! 
- Хотите написать свой компилятор?
- Фреймворки (AOP, ORM)
- Всевозможные инструменты (см. JRebel)



Зачем?

- Знай свою платформу! 
- Хотите написать свой компилятор?
- Фреймворки (AOP, ORM)
- Всевозможные инструменты (см. JRebel)
- ... ну или может просто скучно?



BUT...WHY ?!



Основы Java-байткода
Инструментарий
ObjectWeb ASM

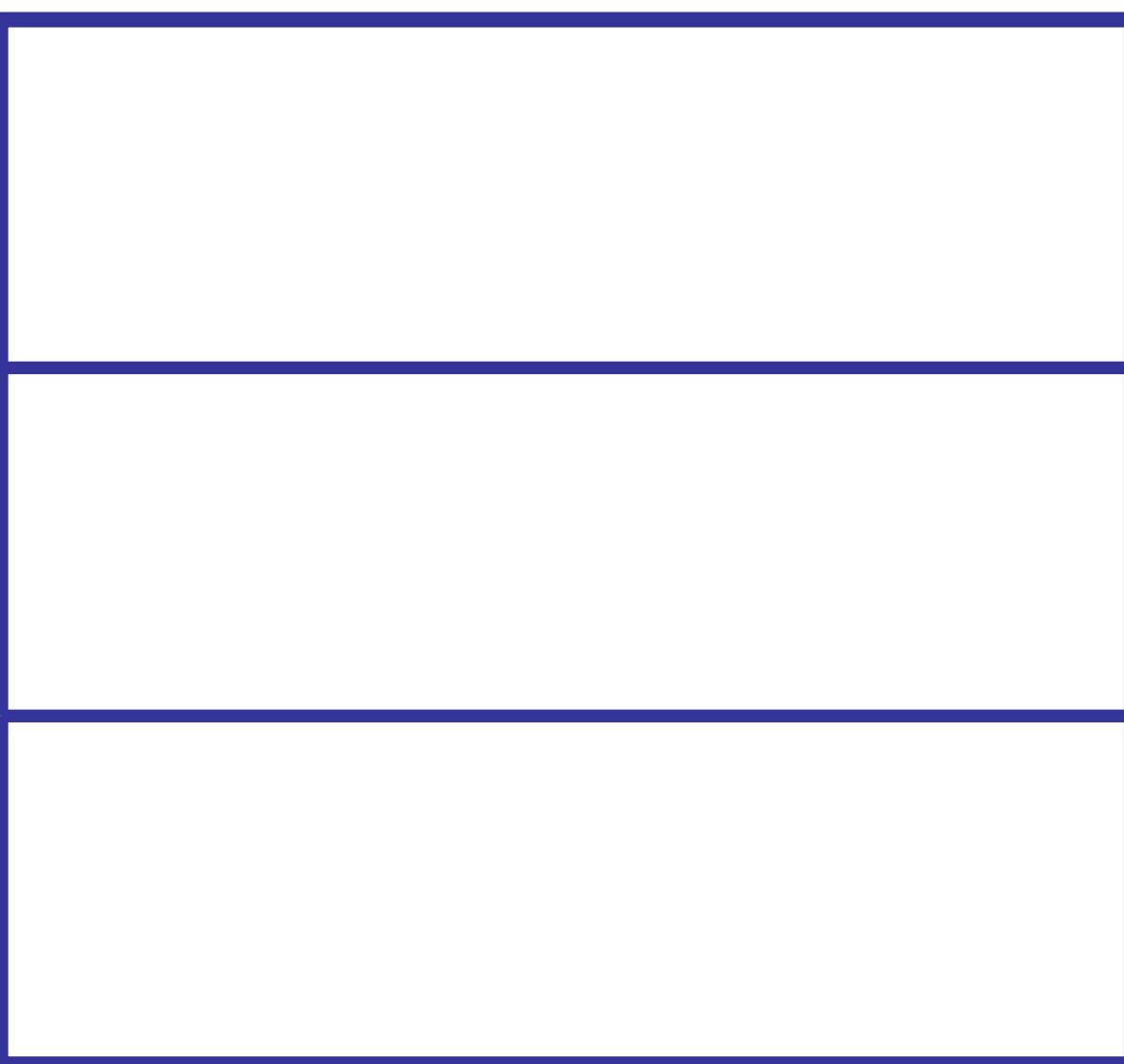
1 + 2

1 + 2

1 2 +

$1 + 2$

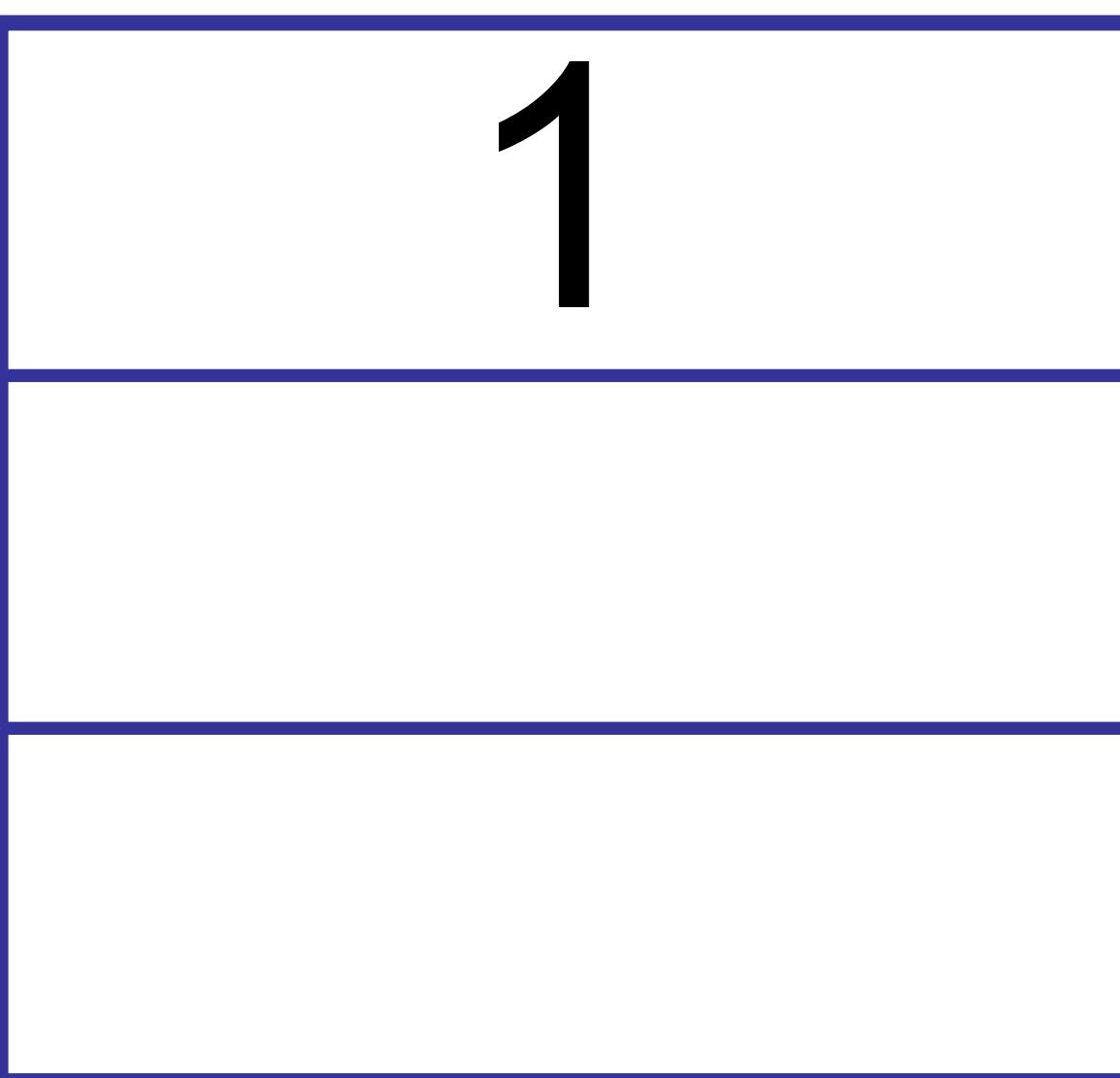
$12 +$



$$1 + 2$$

$$12 +$$

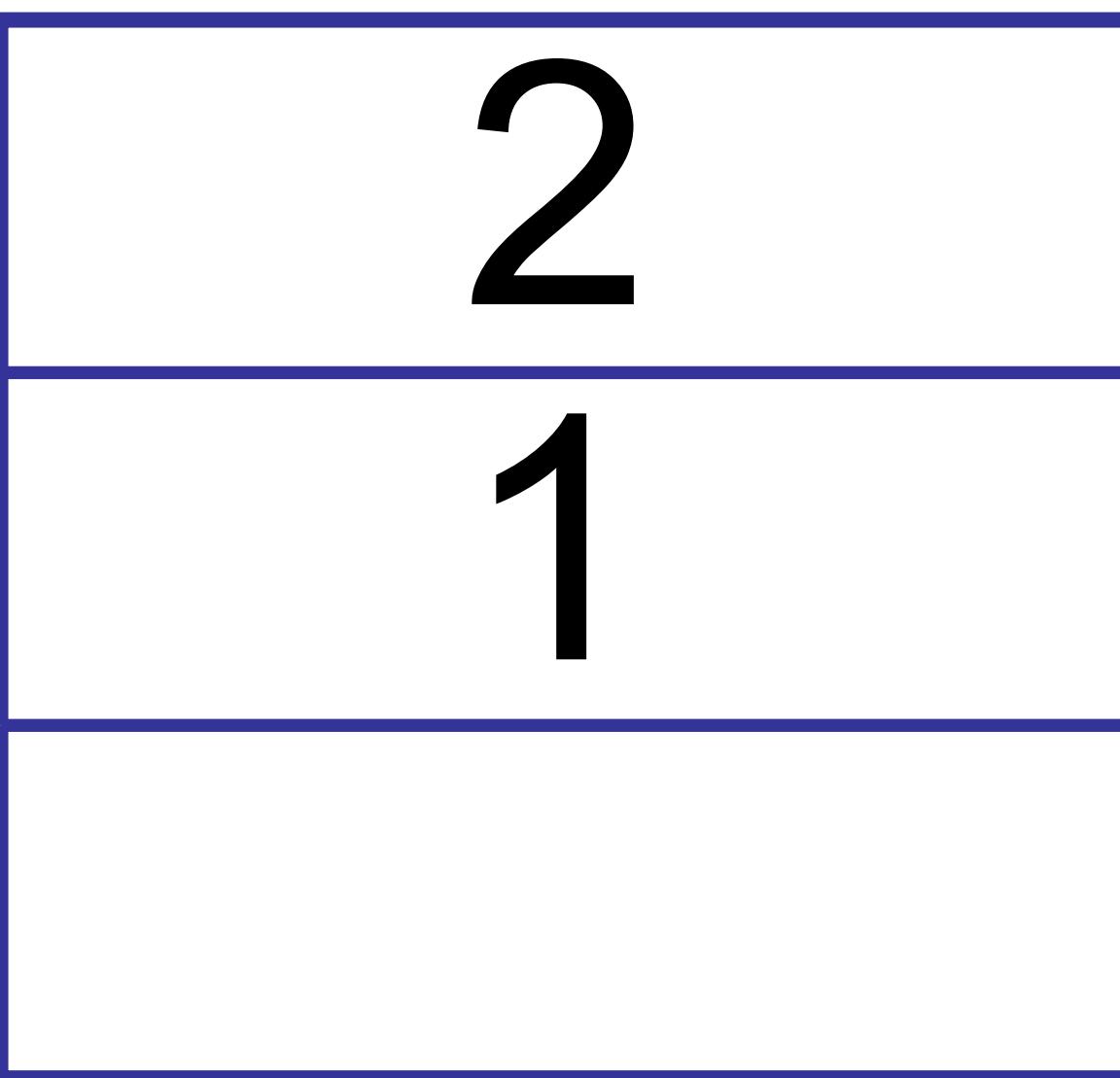
PUSH 1



$$1 + 2$$

$$12 +$$

PUSH 1
PUSH 2



$$1 + 2$$

$$1 \ 2 +$$

PUSH 1
PUSH 2
ADD

3

$$1 + 2$$

$$1\ 2 +$$

ICONST_1
ICONST_2
IADD

3

? = 1 + 2

Таксономия

Таксономия



Работа со
стеком

Таксономия

Работа со
стеком

Инструкции
управления

Таксономия

Работа со
стеком

Инструкции
управления

Работа с
объектами

Таксономия

Работа со
стеком

Инструкции
управления

Арифметика

Работа с
объектами

Таксономия



Байт-код

Байт-код

- Одно-байтные инструкции

Байт-код

- Одно-байтные инструкции
- 256 возможных вариантов

Байт-код

- Одно-байтные инструкции
- 256 возможных вариантов
- Используется 200+

Байт-код

- Одно-байтные инструкции
- 256 возможных вариантов
- Используется 200+
- Google: “Java bytecode instructions listings”

Байт-код

- Одно-байтные инструкции
- 256 возможных вариантов
- Используется 200+
- Google: “Java bytecode instructions listings”
- https://en.wikipedia.org/wiki/Java_bytecode_instruction_listings

ТИП

ОПЕРАЦИЯ

ТИП	ОПЕРАЦИЯ
-----	----------

- <тип> ::= b, s, c, i, l, f, d, a

ТИП	ОПЕРАЦИЯ
-----	----------

- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc**, **iconst_1**)

ТИП	ОПЕРАЦИЯ
-----	----------

- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc, istruct_1**)
- Локальные переменные и стек (**load/store**)

ТИП	ОПЕРАЦИЯ
-----	----------

- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc, istruct_1**)
- Локальные переменные и стек (**load/store**)
- Операции с массивами (**aaload, aastore**)

ТИП	ОПЕРАЦИЯ
-----	----------

- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc**, **iconst_1**)
- Локальные переменные и стек (**load/store**)
- Операции с массивами (**aaload**, **aastore**)
- Арифметика (**add**, **sub**, **mul**, **div**)

ТИП	ОПЕРАЦИЯ
-----	----------

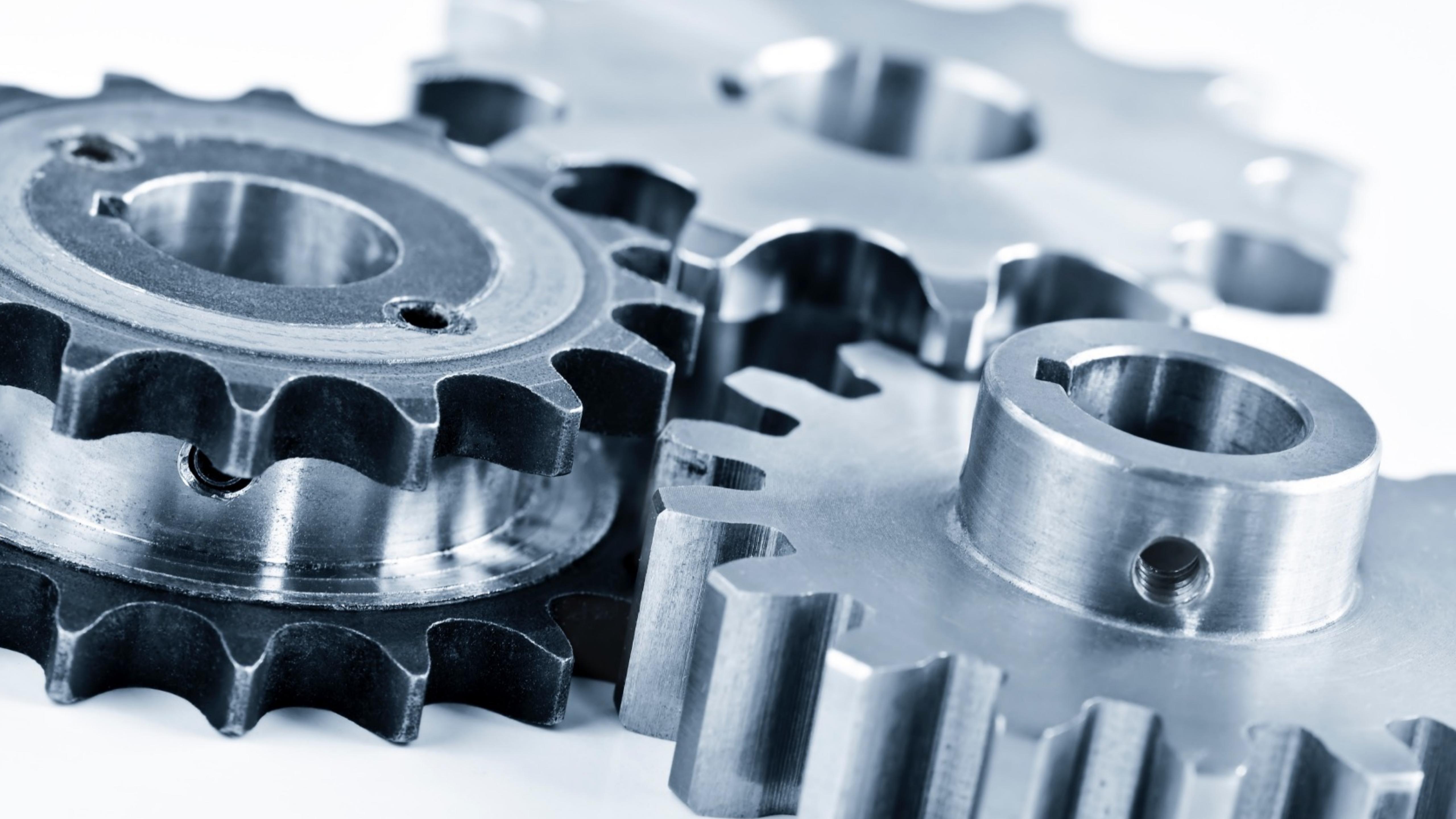
- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc**, **iconst_1**)
- Локальные переменные и стек (**load/store**)
- Операции с массивами (**aaload**, **aastore**)
- Арифметика (**add**, **sub**, **mul**, **div**)
- Булевые/битовые операции (**iand**, **ixor**)

ТИП	ОПЕРАЦИЯ
-----	----------

- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc**, **iconst_1**)
- Локальные переменные и стек (**load/store**)
- Операции с массивами (**aaload**, **aastore**)
- Арифметика (**add**, **sub**, **mul**, **div**)
- Булевые/битовые операции (**iand**, **ixor**)
- Сравнения (**cmpg**, **cmpl**, **ifne**, **ifeq**)

ТИП	ОПЕРАЦИЯ
-----	----------

- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc**, **iconst_1**)
- Локальные переменные и стек (**load/store**)
- Операции с массивами (**aaload**, **aastore**)
- Арифметика (**add**, **sub**, **mul**, **div**)
- Булевые/битовые операции (**iand**, **ixor**)
- Сравнения (**cmprg**, **cmpl**, **ifne**, **ifeq**)
- Преобразования (**l2d**, **i2l**)



JVM процесс

Thread

1

Thread

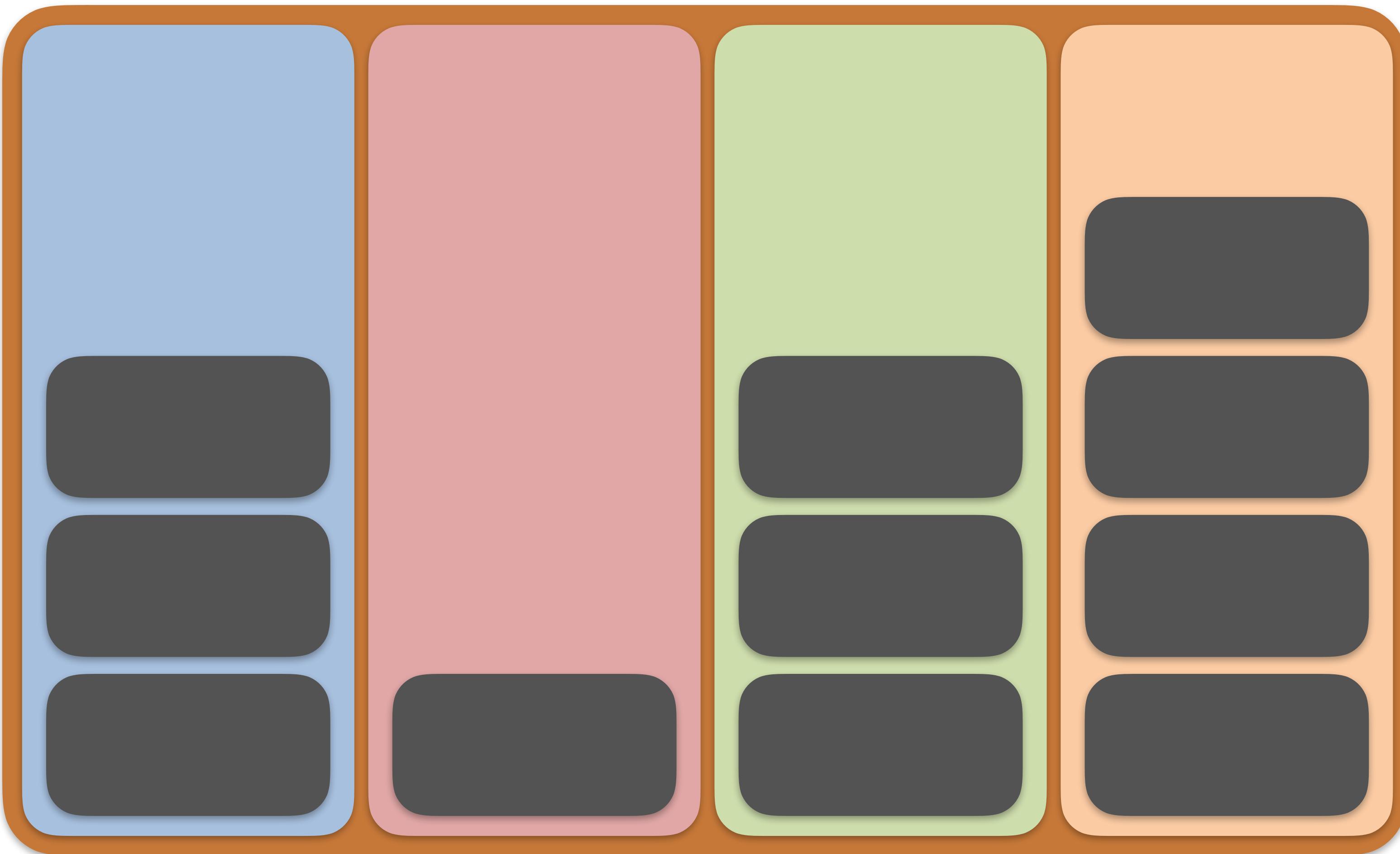
2

Thread

3

Thread

4



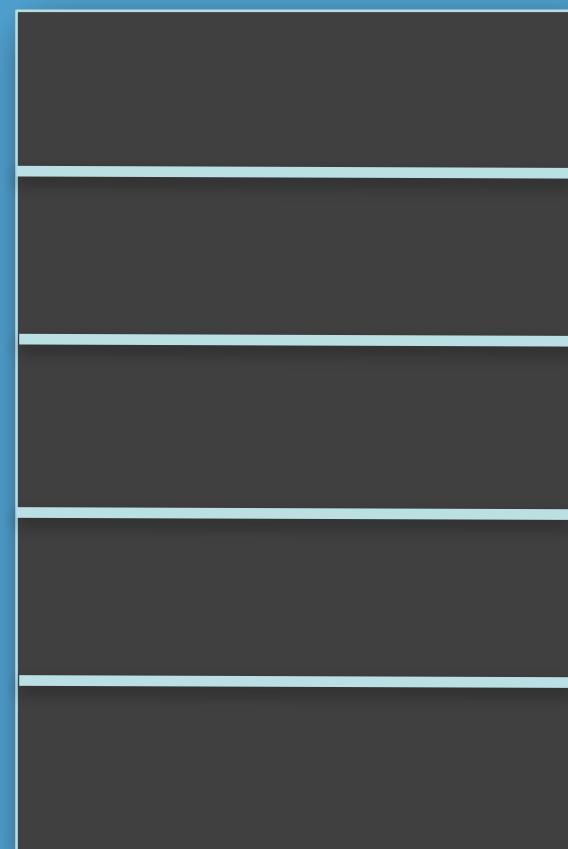
Стековая машина

- JVM работает со стеком
- У каждого потока есть стек
- Стек сохраняет “фреймы”
- Новый “фрейм” создаётся при вызове метода
- “Фрейм” состоит из:
 - Стек операций
 - Массив локальных переменных

Локальные переменные

0	1	2	...	N
---	---	---	-----	---

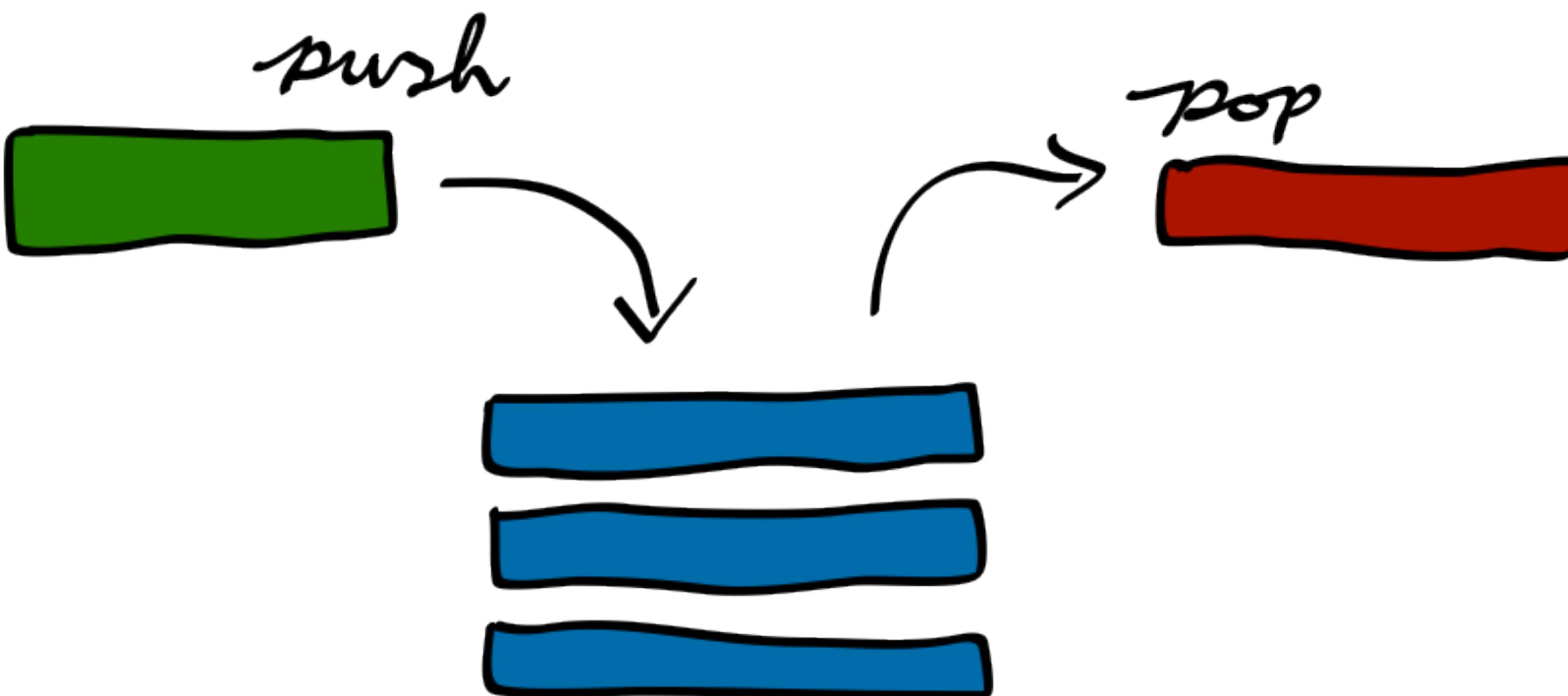
Стек операций



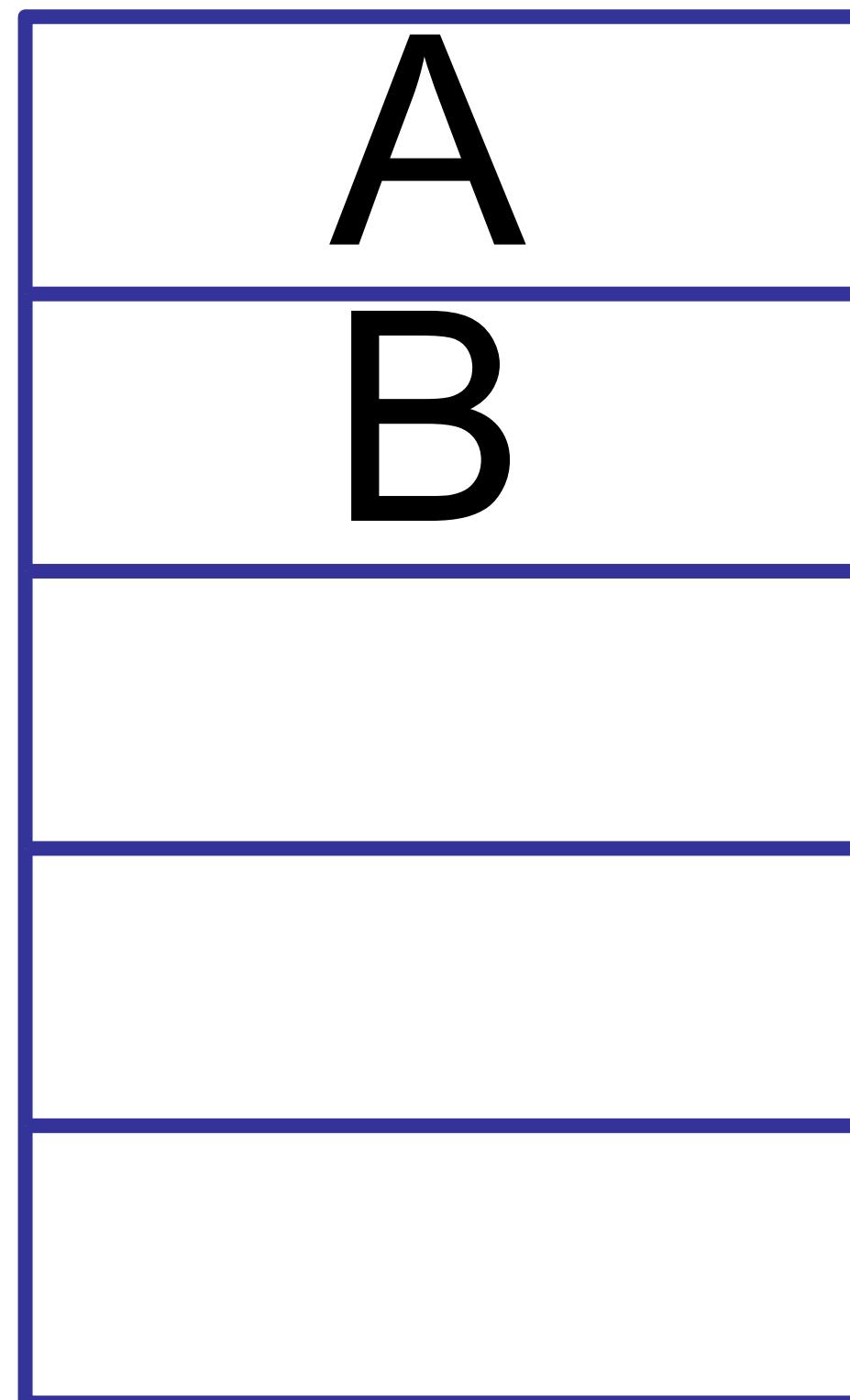
#1

Константы

Работа со стеком и локальными переменными



dup
pop
swap
dup_x1
dup2_x1



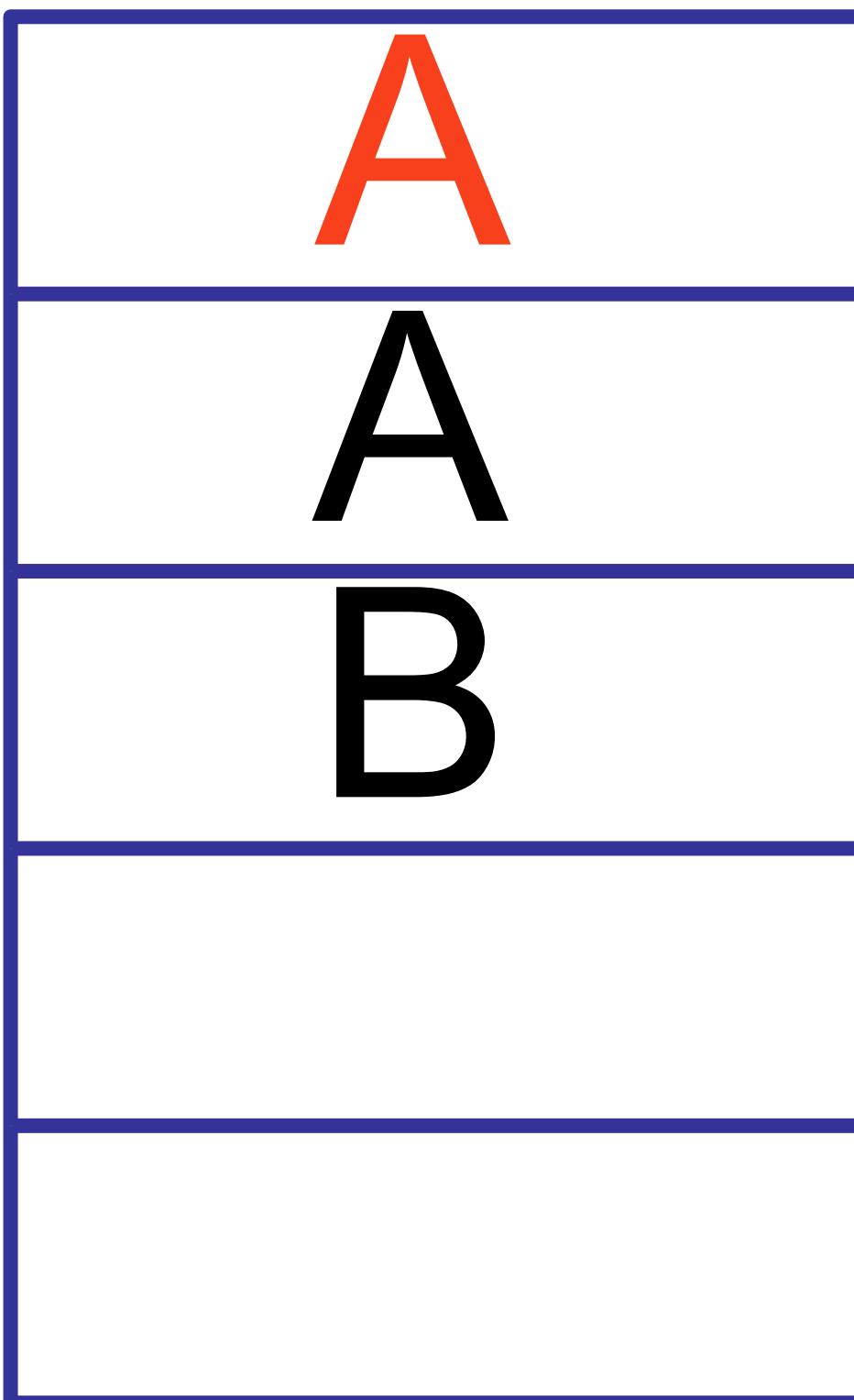
dup

pop

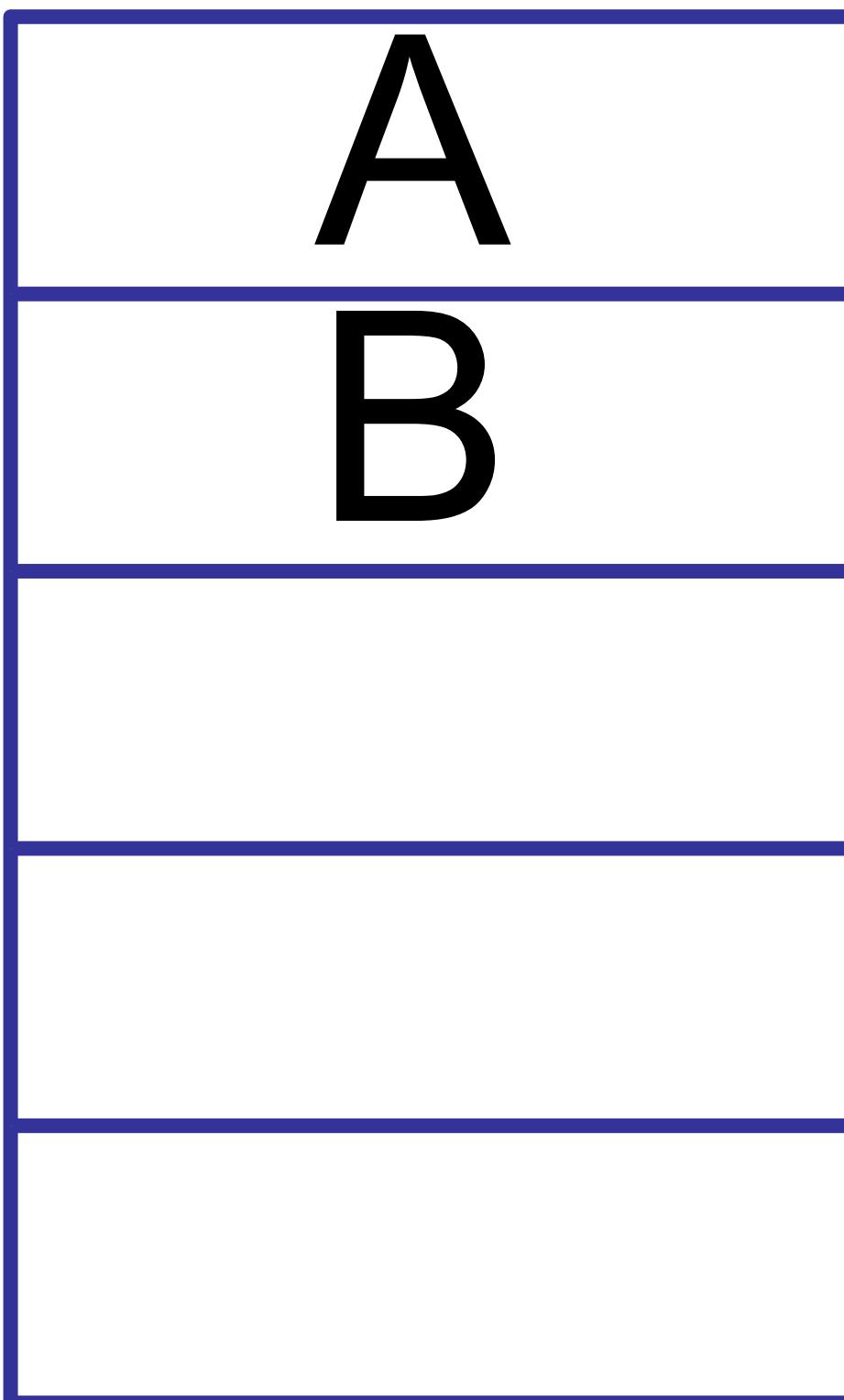
swap

dup_x1

dup2_x1



dup
pop
swap
dup_x1
dup2_x1



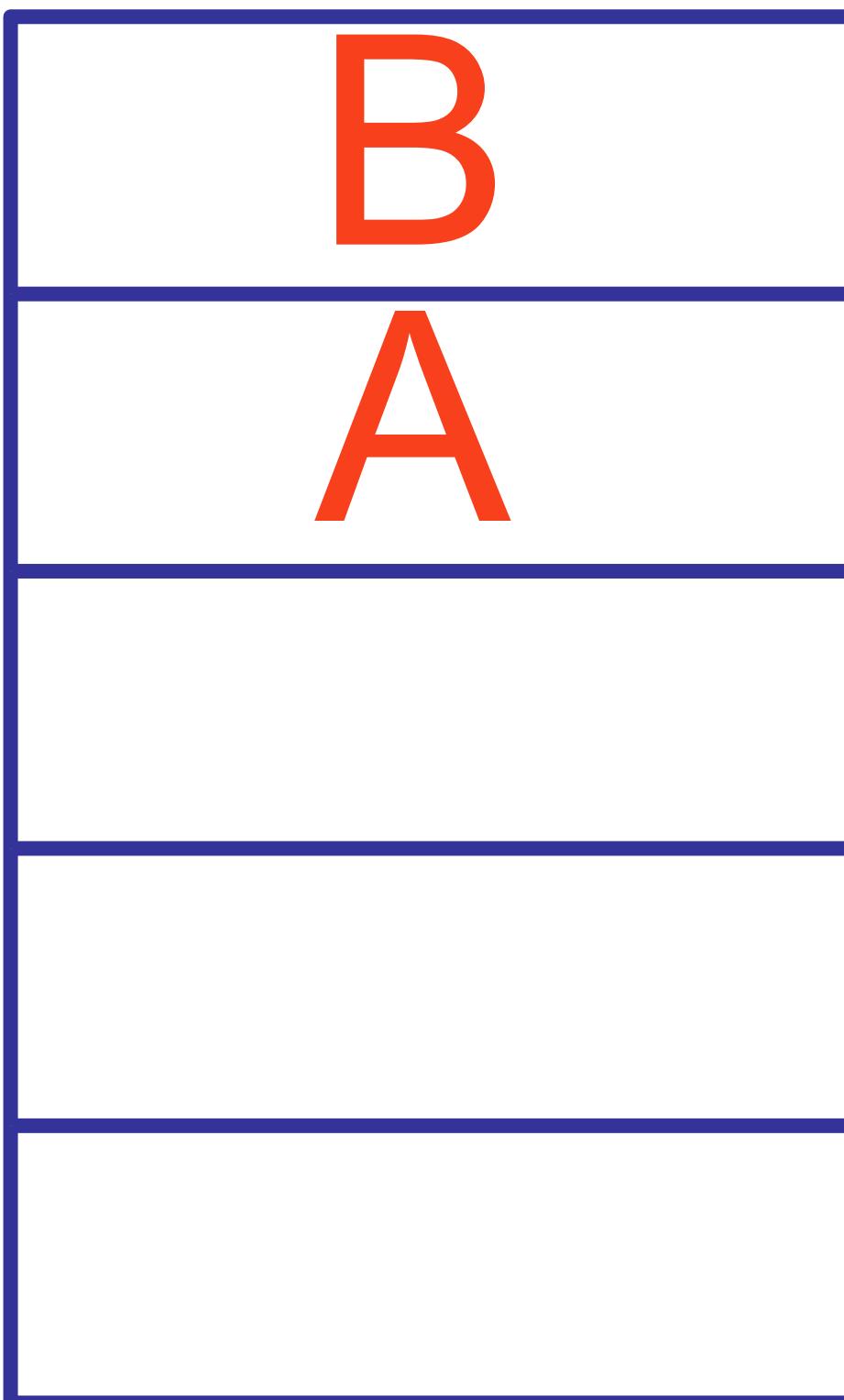
dup

pop

swap

dup_x1

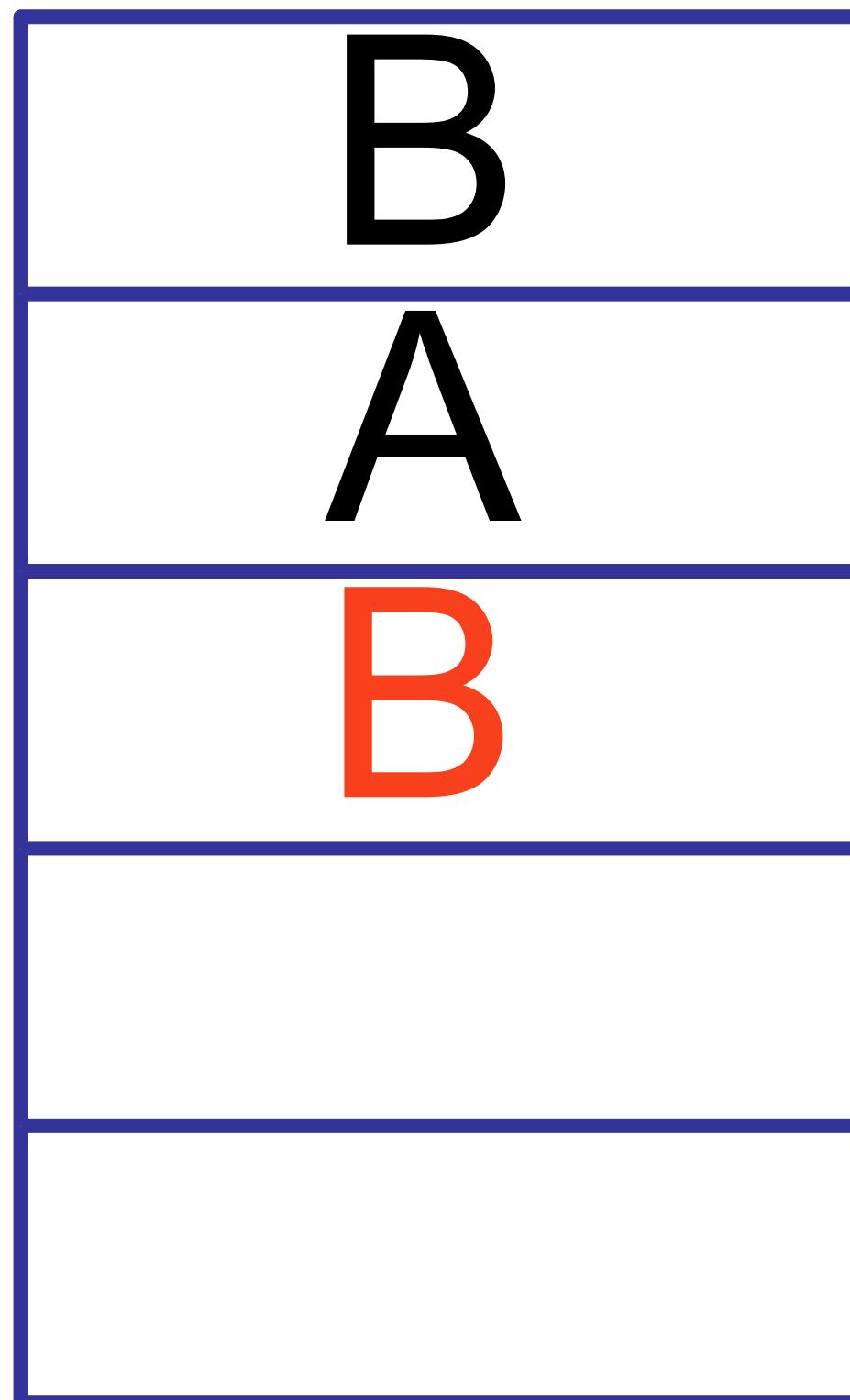
dup2_x1



dup
pop
swap

dup_x1

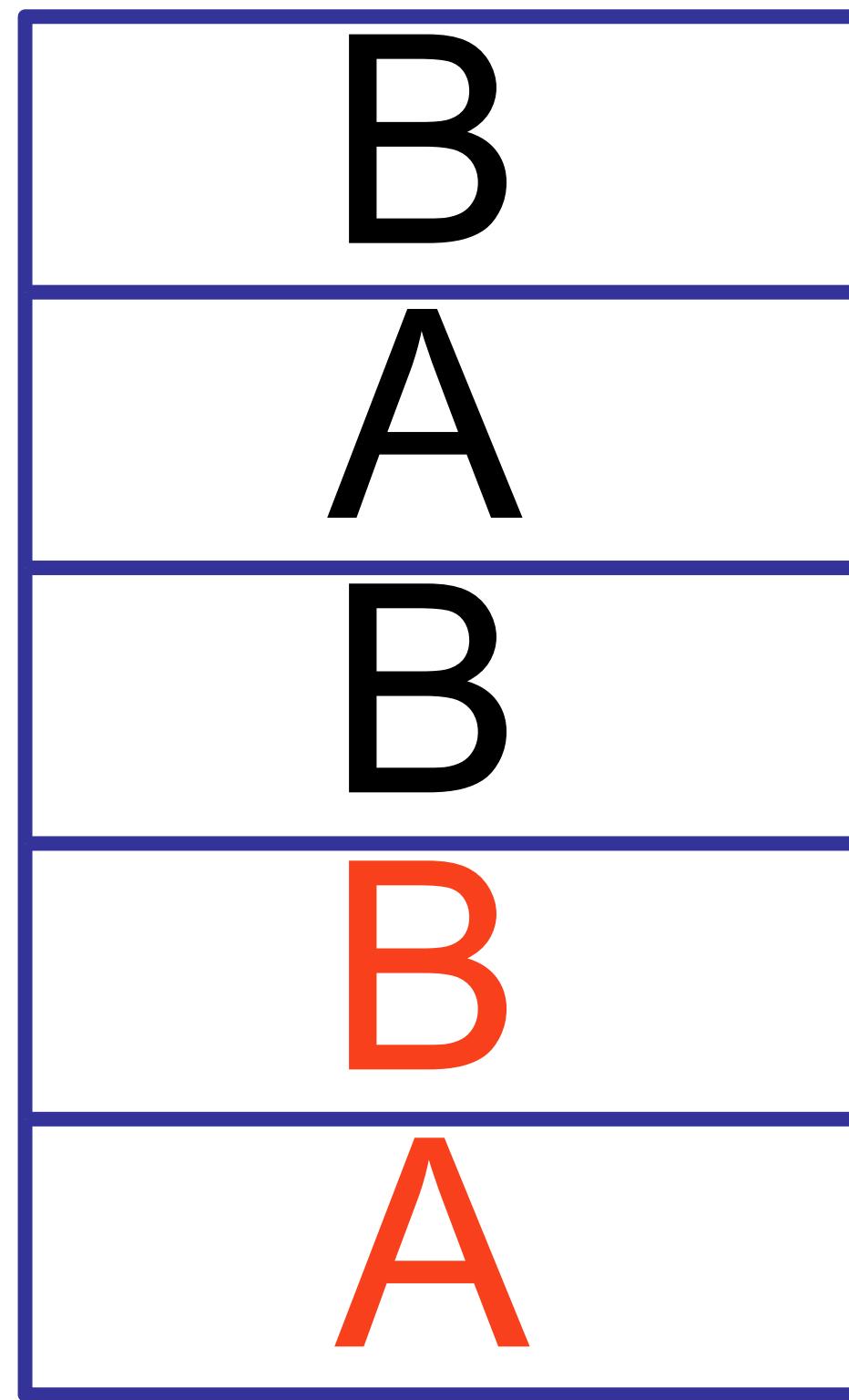
dup2_x1



dup
pop
swap

dup_x1

dup2_x1



Локальные переменные

переменная	значение
------------	----------

переменная	значение
0	
1	
2	
3	
4	

ldc "Hello"
astore_0
iconst_1
astore_1
aload_0

Стек

глубина	значение
---------	----------

глубина	значение
0	
1	
2	
3	
4	

Локальные переменные

переменная значение

0	
1	
2	
3	
4	

ldc "Hello"
astore_0
iconst_1
astore_1
aload_0

Стек

глубина значение

0	"Hello"
1	
2	
3	
4	

Локальные переменные

переменная значение

0	"Hello"
1	
2	
3	
4	

ldc "Hello"
astore_0
iconst_1
astore_1
aload_0

Стек

глубина значение

0	
1	
2	
3	
4	

Локальные переменные

переменная значение

переменная	значение
0	"Hello"
1	
2	
3	
4	

```
ldc "Hello"  
astore_0  
iconst_1  
astore_1  
aload_0
```

Стек

глубина значение

глубина	значение
0	1
1	
2	
3	
4	

Локальные переменные

переменная значение

переменная	значение
0	"Hello"
1	1
2	
3	
4	

ldc "Hello"
astore_0
iconst_1
astore_1
aload_0

Стек

глубина значение

глубина	значение
0	
1	
2	
3	
4	

Локальные переменные

переменная значение

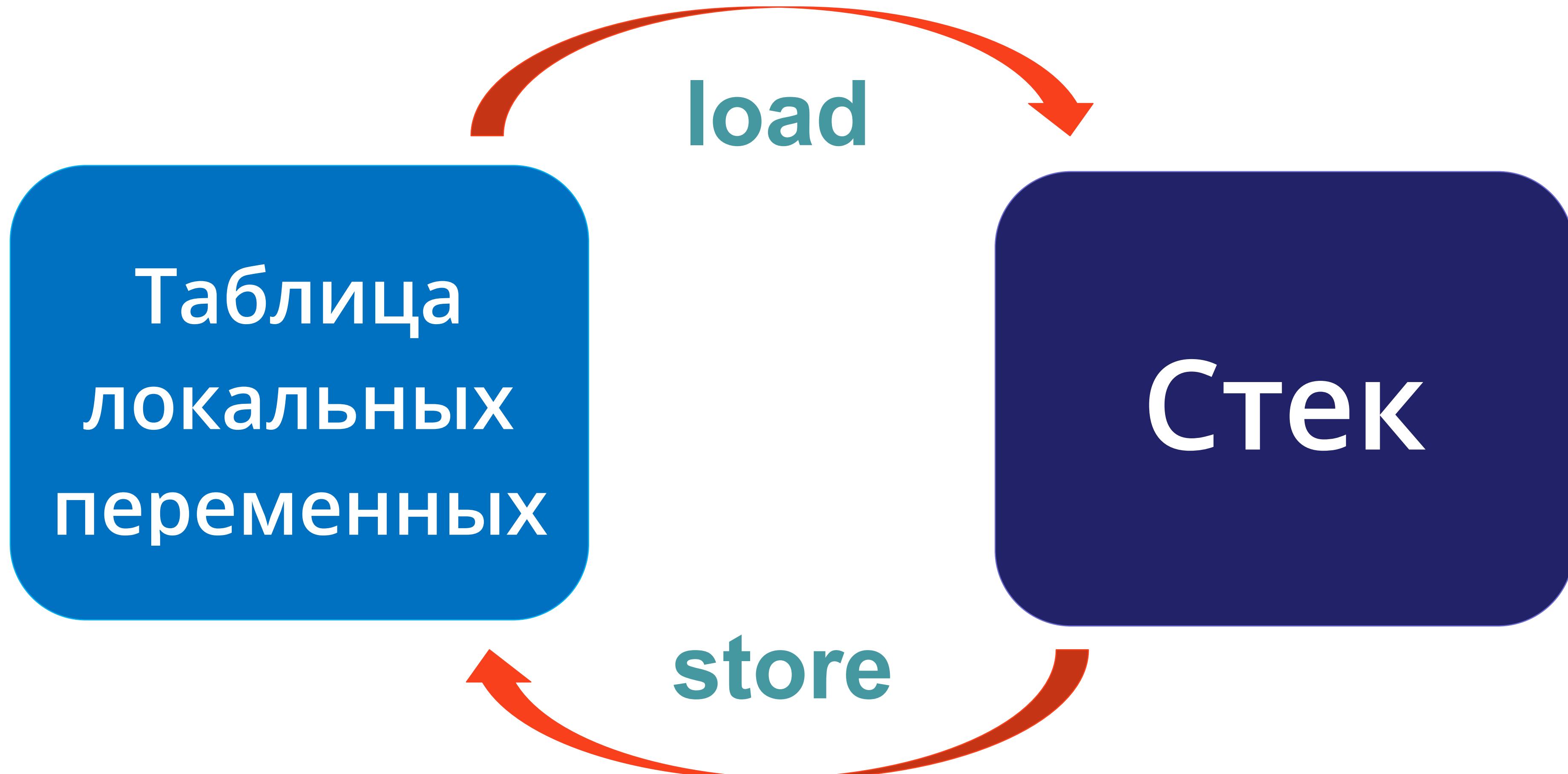
переменная	значение
0	"Hello"
1	1
2	
3	
4	

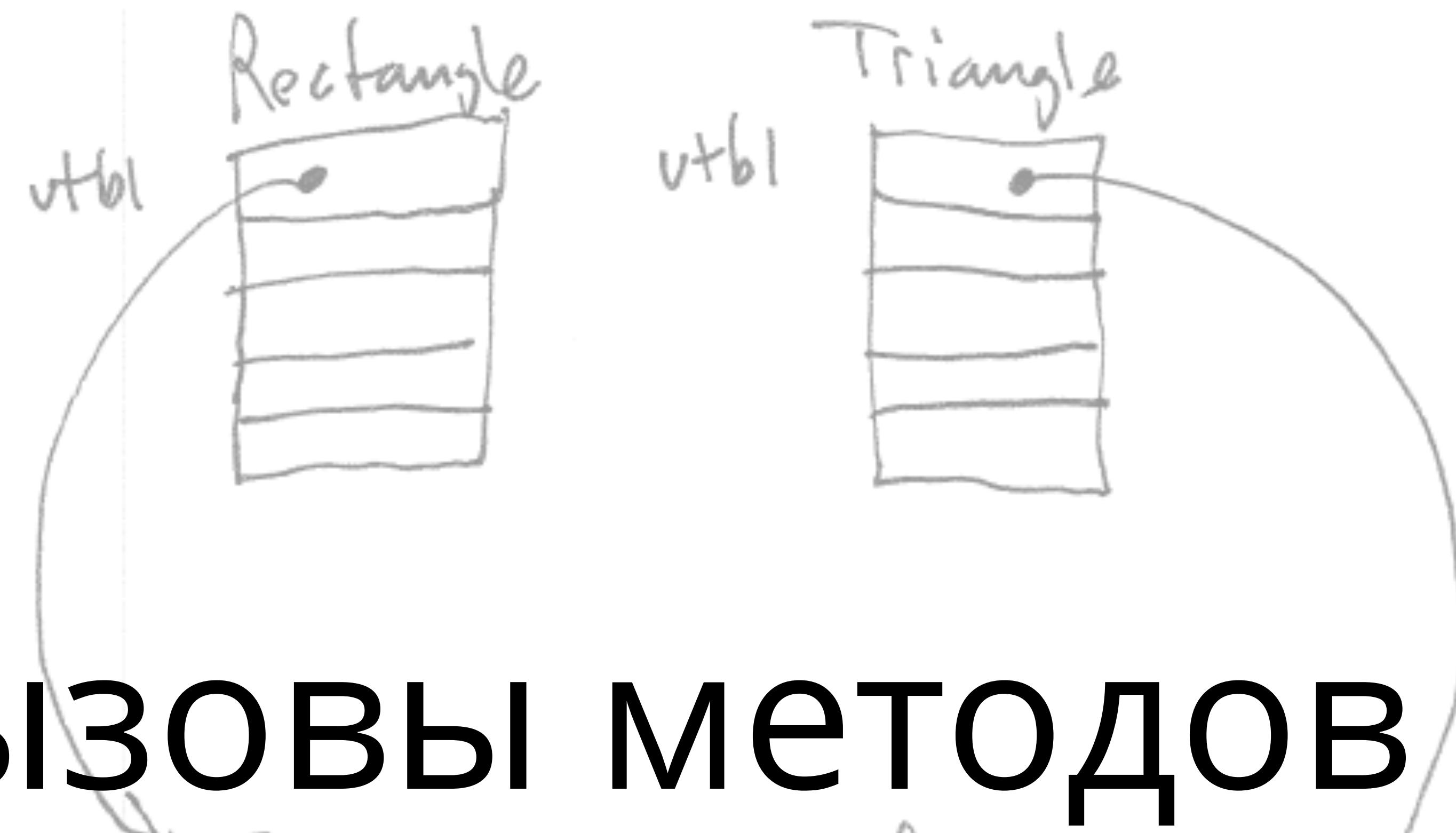
ldc "Hello"
astore_0
iconst_1
astore_1
aload_0

Стек

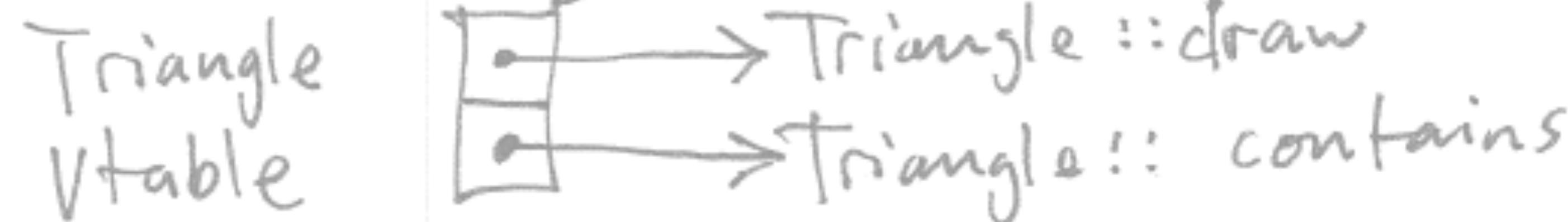
глубина значение

глубина	значение
0	"Hello"
1	
2	
3	
4	





Вызовы методов

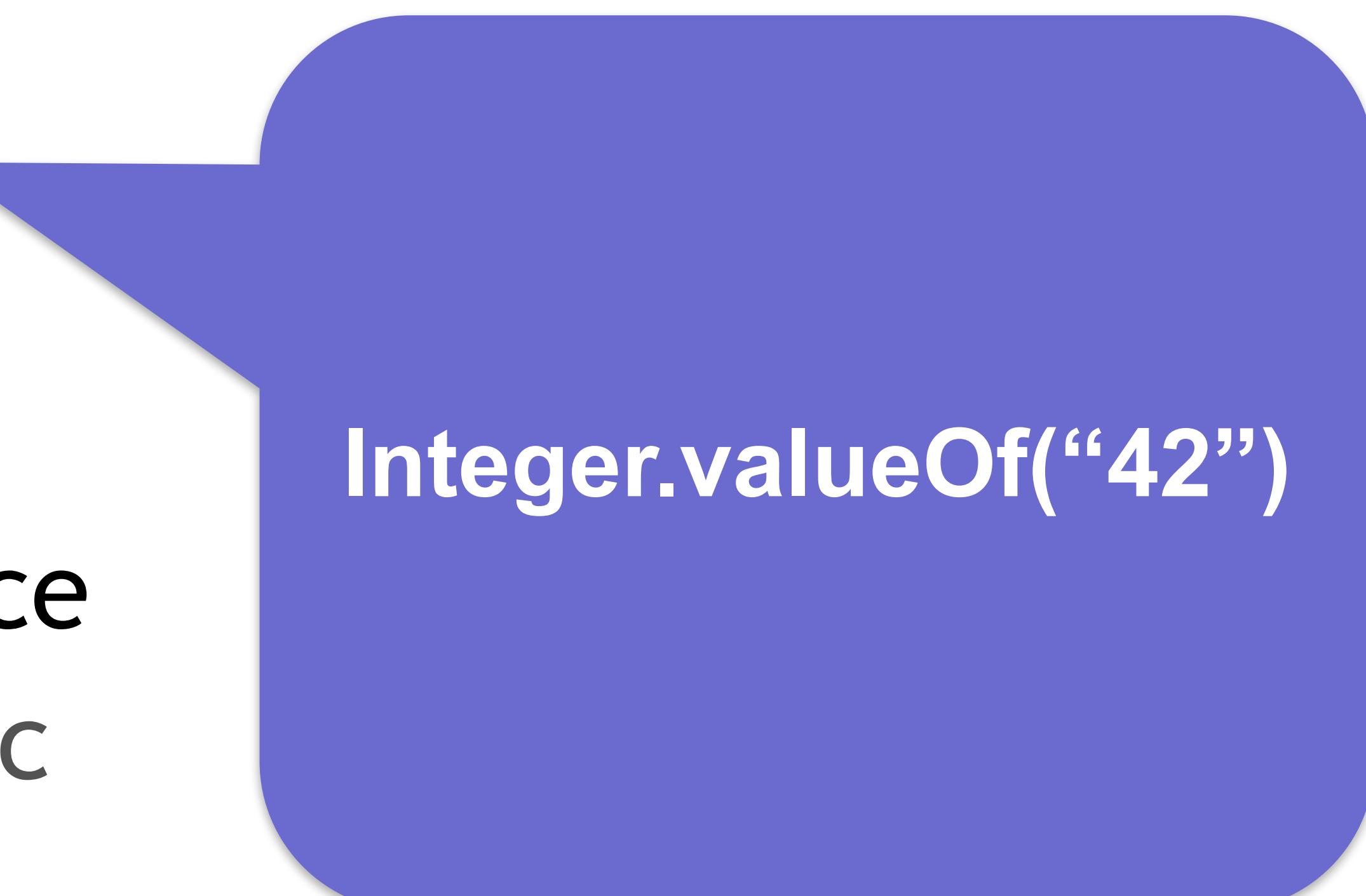


invokeXXX

- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic

invokestatic

- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic



Integer.valueOf("42")

invokespecial

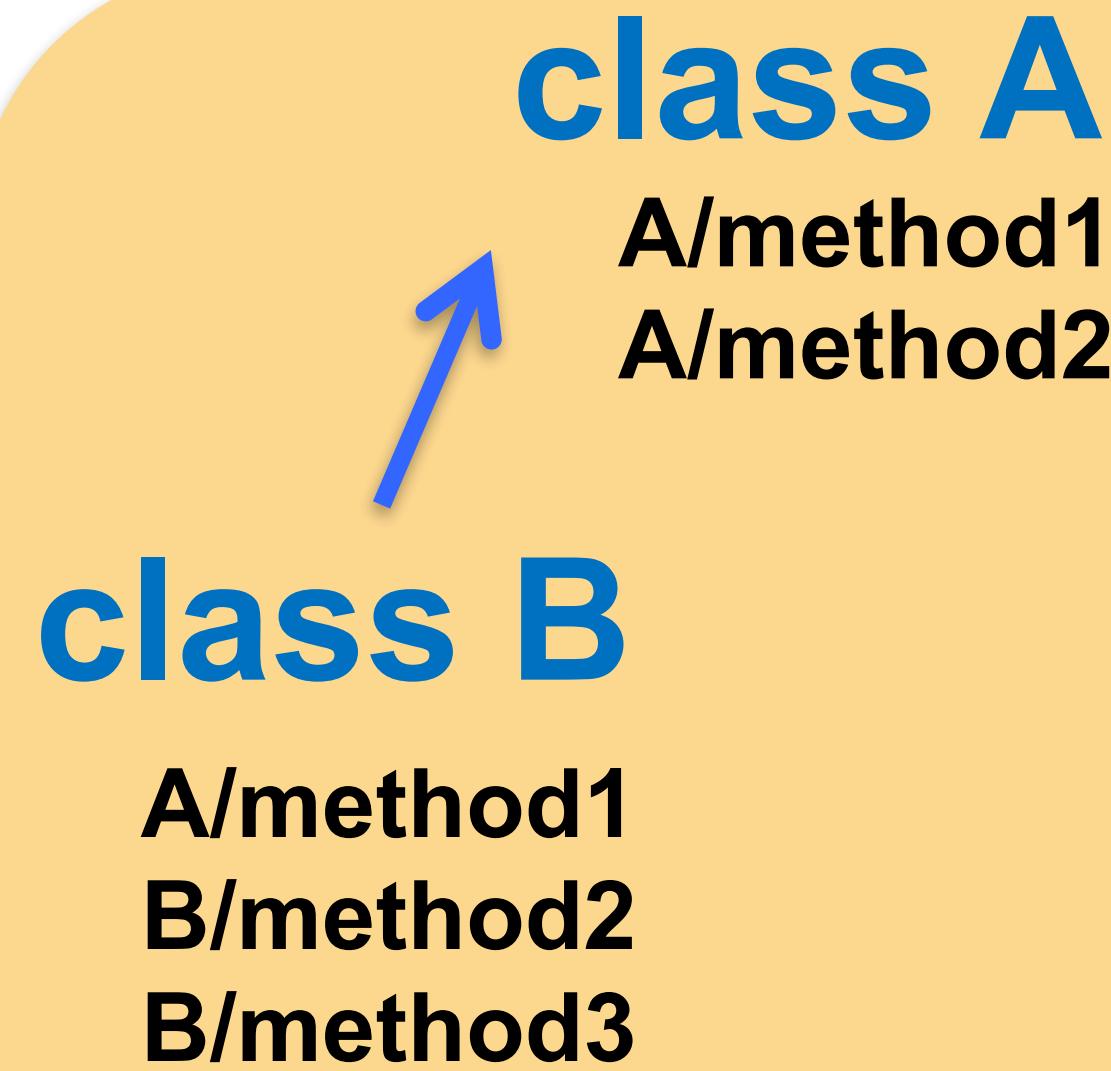
- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic

<init>

```
private void foo();  
super.method();
```

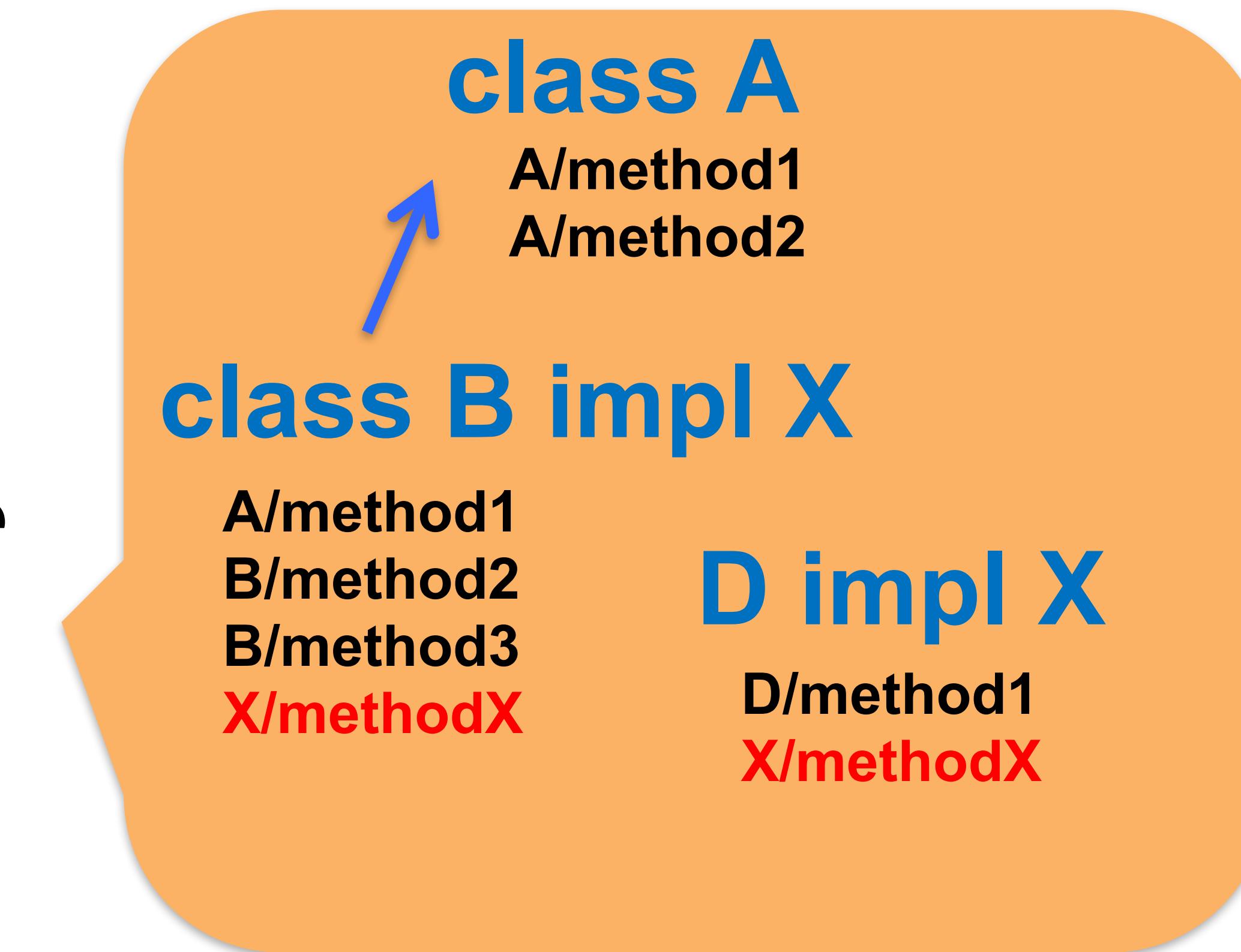
invokevirtual

- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic



invokeinterface

- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic



Efficient Implementation of Java Interfaces: *Invokeinterface Considered Harmless*, Bowen Alpern, Anthony Cocchi, Stephen Fink, David Grove, and Derek Lieber, OOPSLA'01

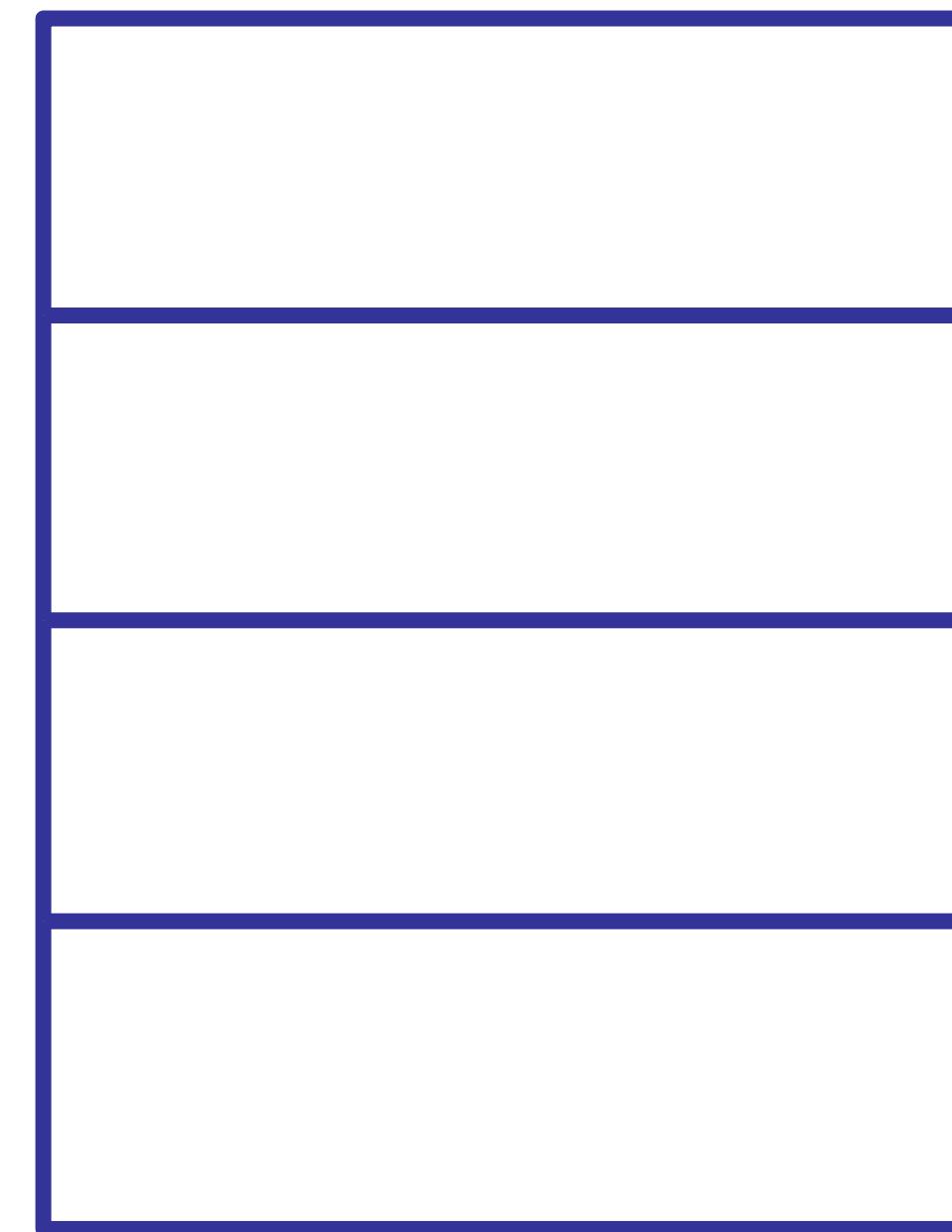
Вызов метода

Вызов метода

```
obj.method(param1, param2);
```

Вызов метода

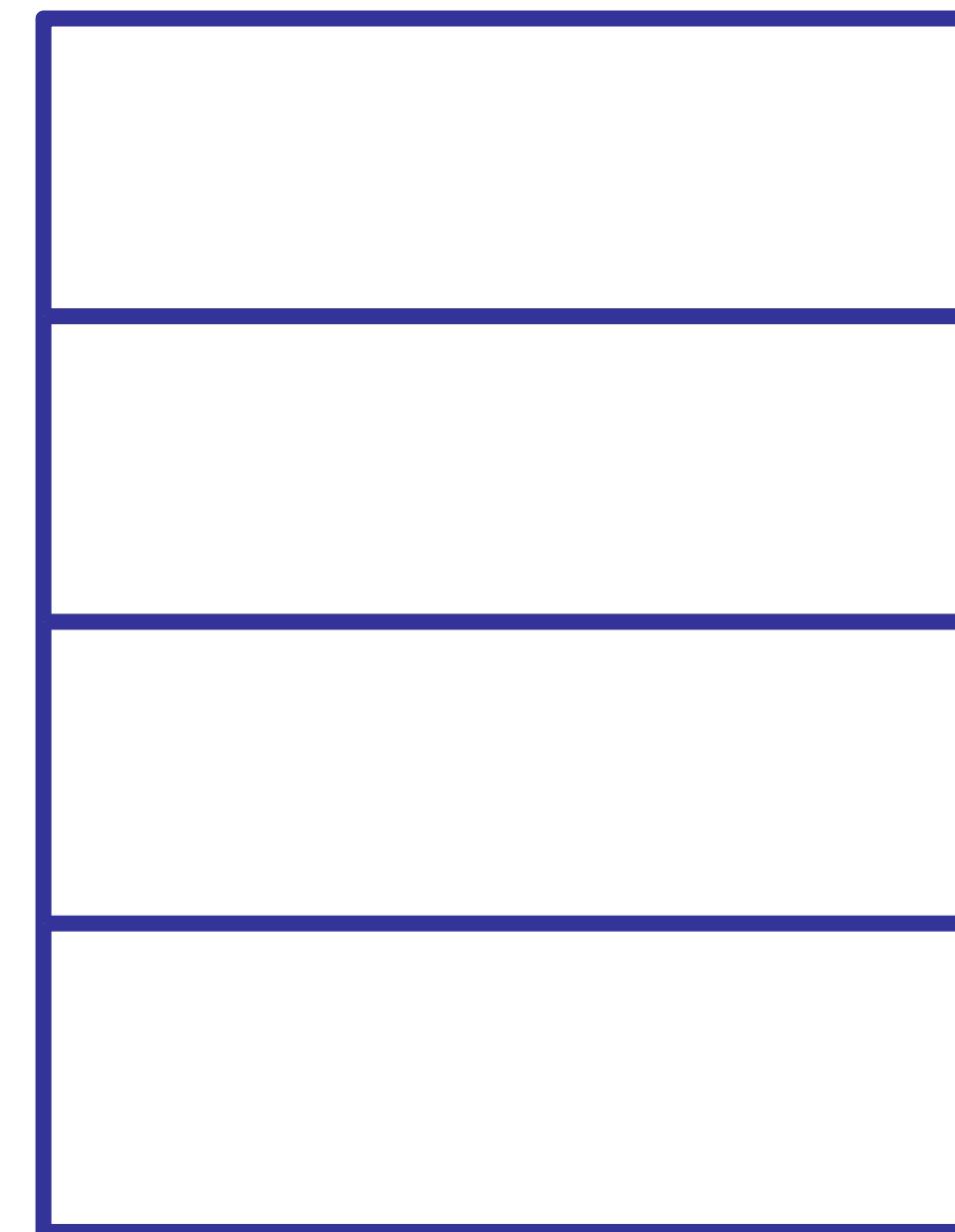
obj.method(param1, param2);



Вызов метода

obj.method(param1, param2);

**push obj
push param1
push param2
call method**



Вызов метода

obj.method(param1, param2);

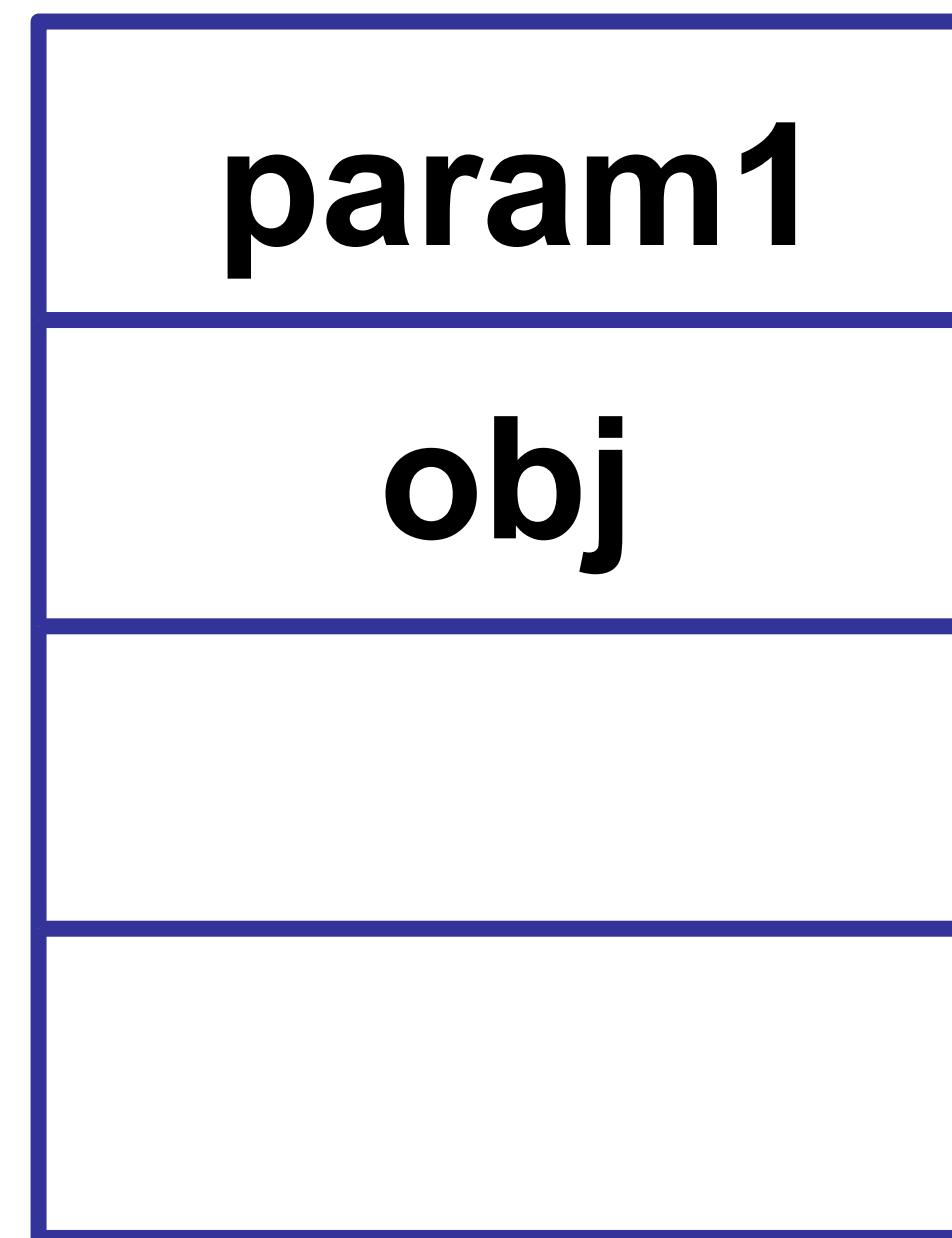
push obj
push param1
push param2
call method



Вызов метода

obj.method(param1, param2);

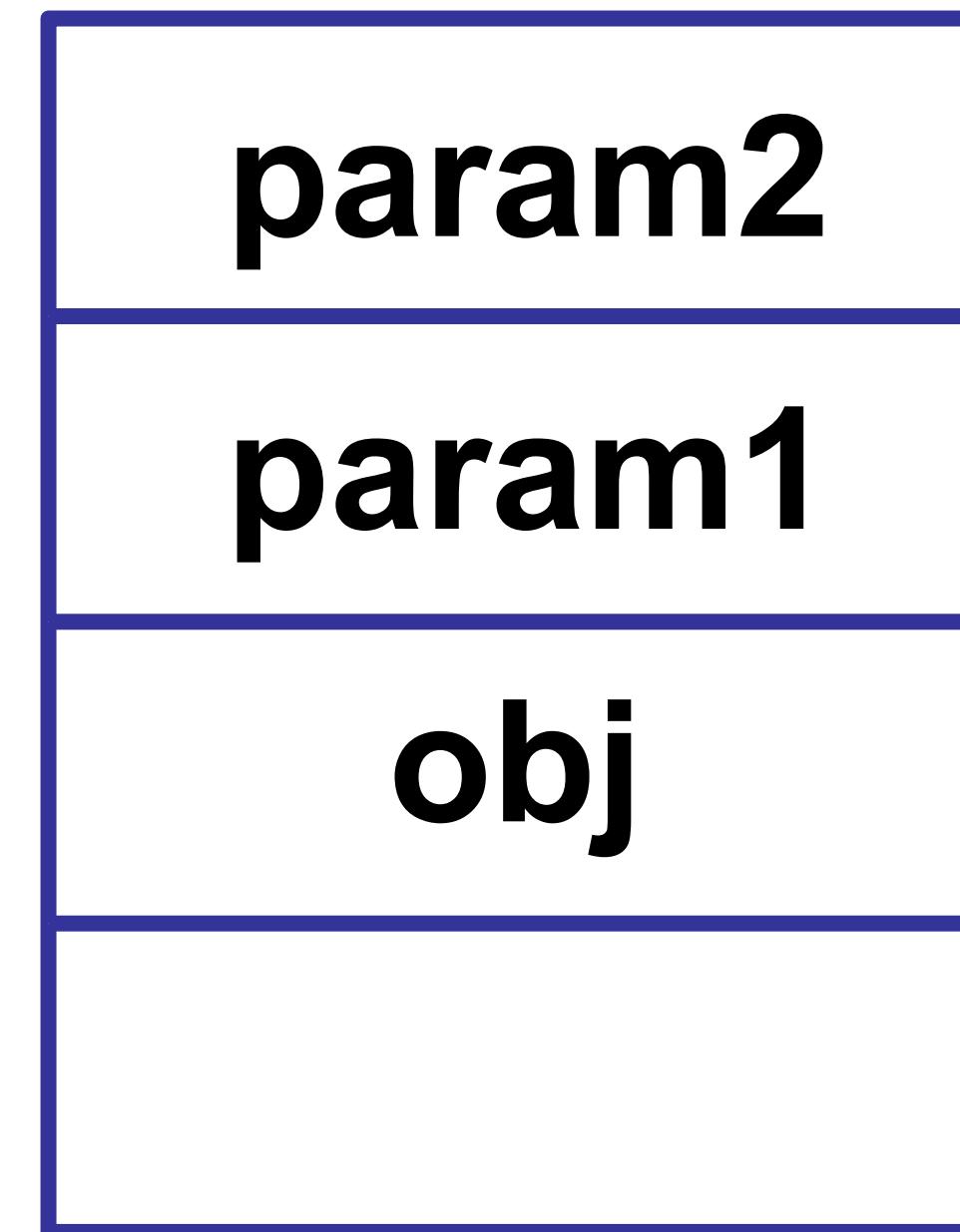
push obj
push param1
push param2
call method



Вызов метода

obj.method(param1, param2);

push obj
push param1
push param2
call method



Вызов метода

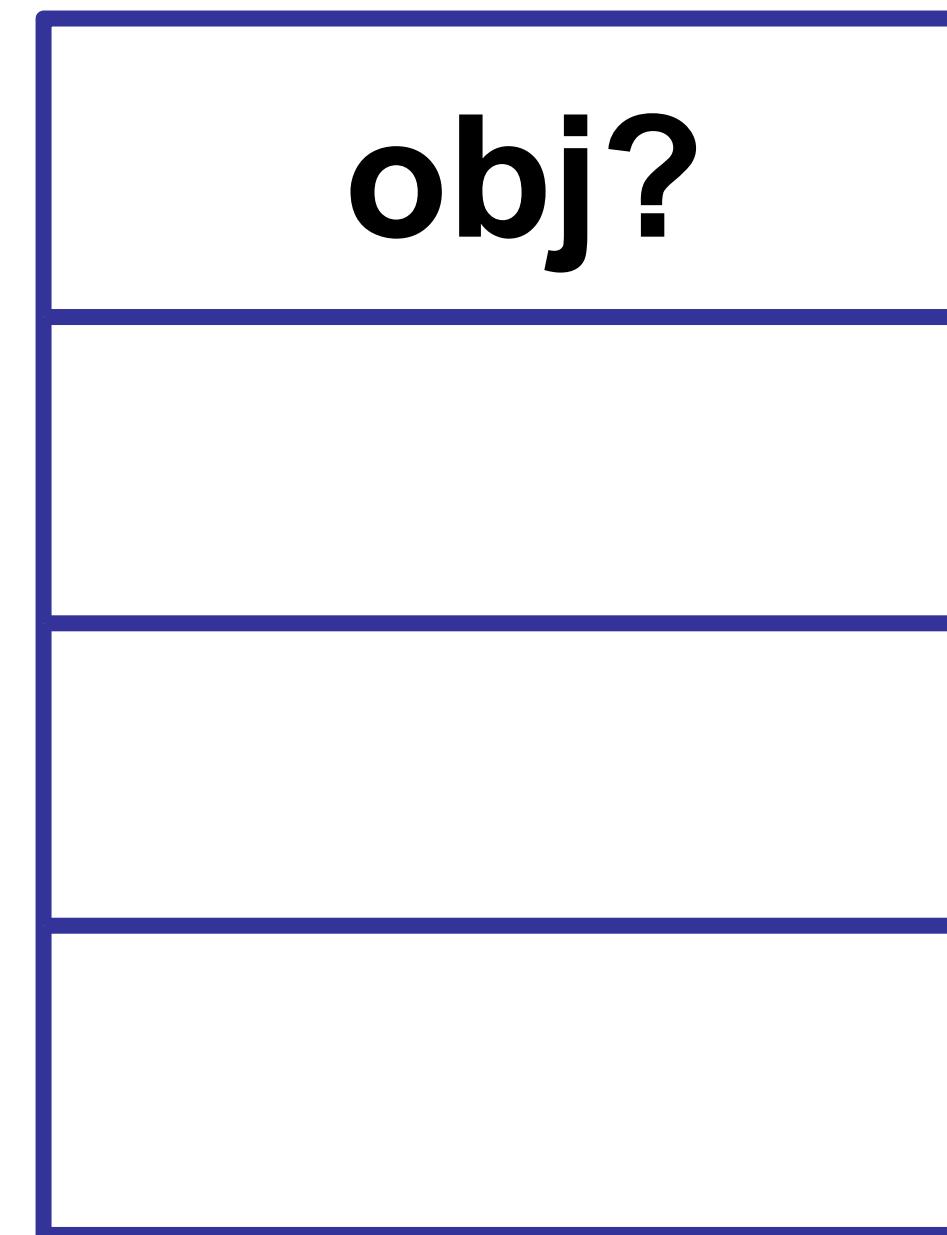
obj.method(param1, param2);

push obj

push param1

push param2

call method

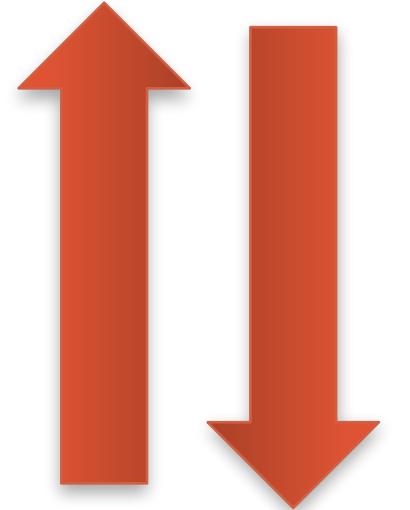


Вызов метода

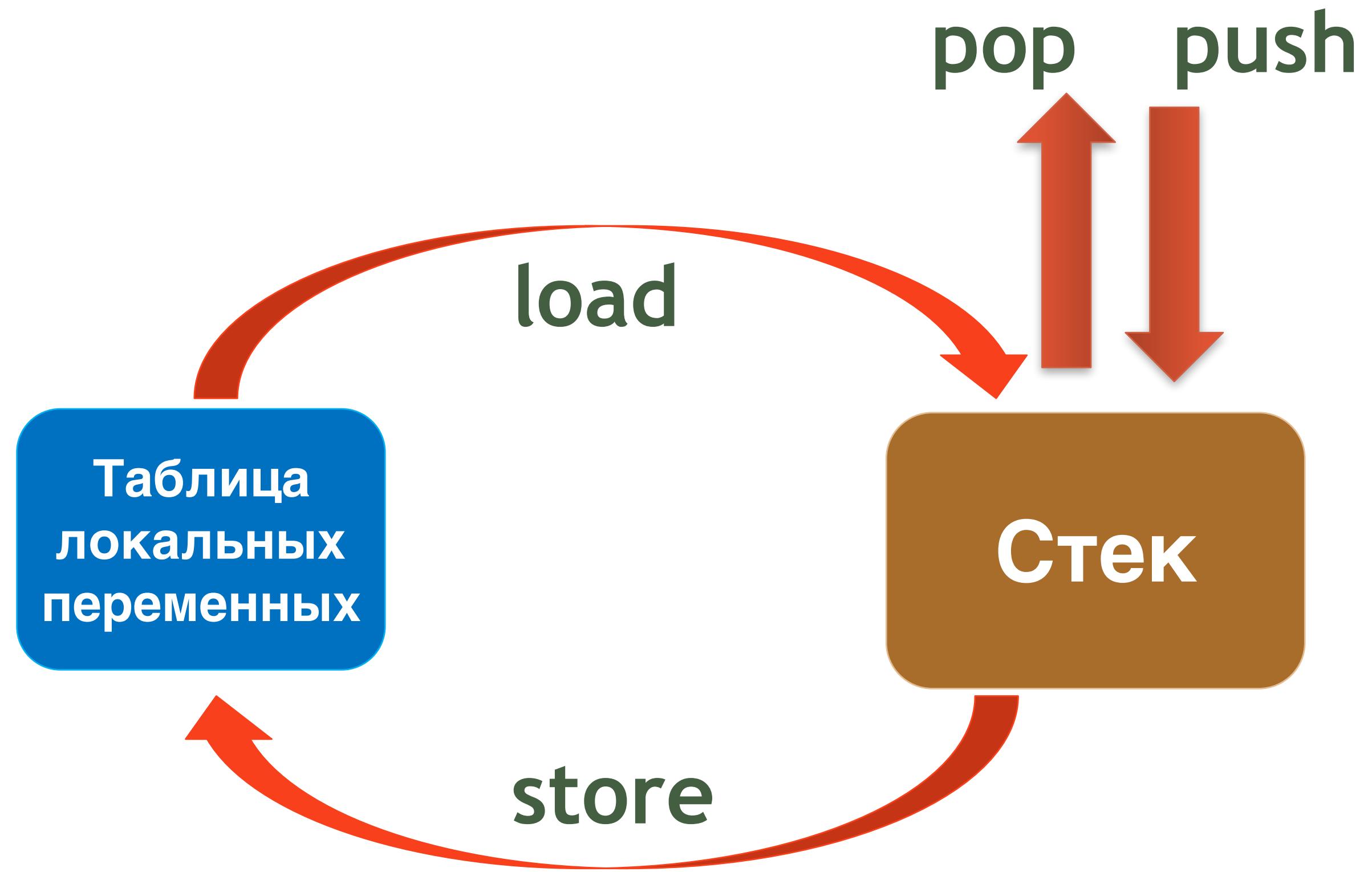
this.add(1, 2);

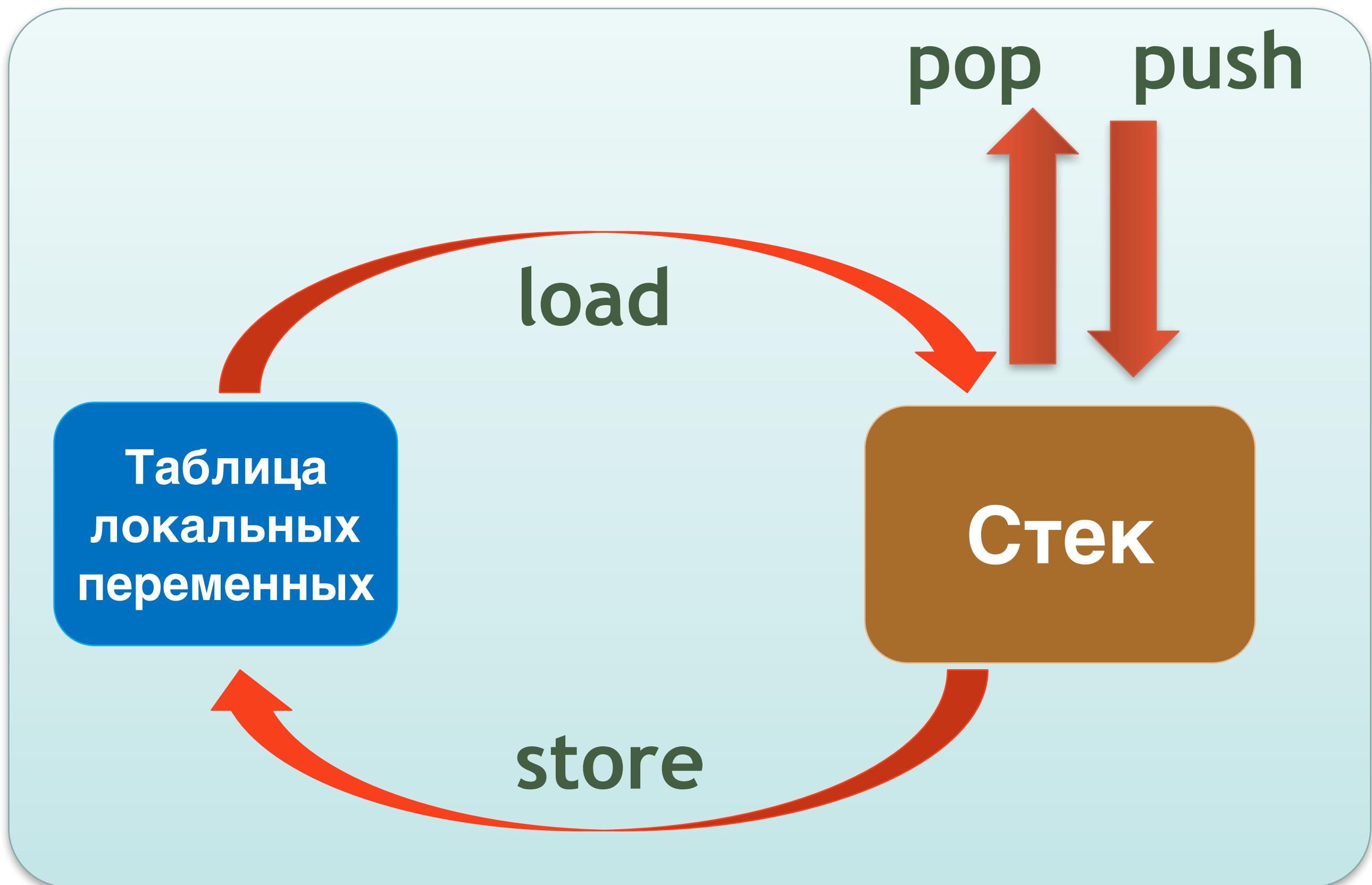
```
0:  aload_0
1:  iconst_1
2:  iconst_2
3:  invokevirtual #2; //Method add:(II)I
```

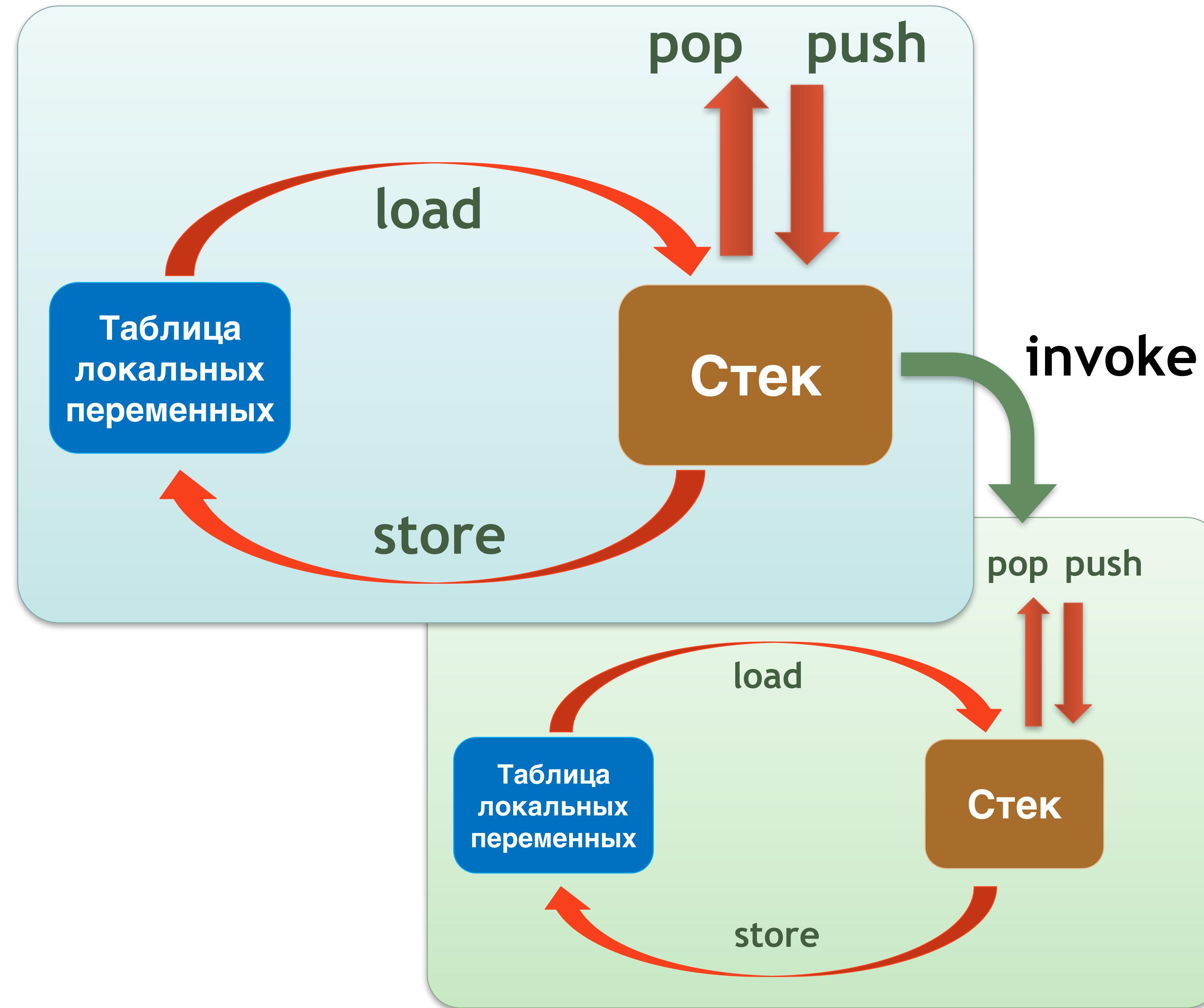
pop push



Стек







>HELLO WORLD

>_

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}  
> javap -c Hello
```

Дезассемблировать Hello



```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

javap

- Дизассемблер Java класс-файлов
- По-умолчанию показывает только структуру класса
 - Методы, супер-класс, интерфейсы, итд
- **-c** покажет байткод методов
- **-private** покажет все приватные поля и методы
- **-s** покажет сигнатуры
- **-l** покажет номера строк и таблицу локальных переменных

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
> javap -c Hello
```

```
Compiled from "Hello.java"
```

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

Конструктор
по-умолчанию

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

Выложить this на стек



```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

Вызвать <init> для Object



```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

public static void main(java.lang.String[]);

Code:

```
0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
3:  ldc #3; //String Hello, World!  
5:  invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

public static void main(java.lang.String[]);

Code:

```
0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
3:  ldc #3; //String Hello, World!  
5:  invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

Обратиться к
статическому полю



```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

```
public static void main(java.lang.String[]);
```

Code:

```
0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
3:  ldc #3; //String Hello, World! ←  
5:  invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

Загрузить строковую
константу в стек

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

```
public static void main(java.lang.String[]);
```

Code:

```
0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
3:  ldc #3; //String Hello, World!  
5:  invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

Вызвать метод
с параметром



```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c Hello**

Compiled from "Hello.java"

```
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

public static void main(java.lang.String[]);

Code:

```
0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
3:  ldc #3; //String Hello, World!  
5:  invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
> javap -c Hello  
Compiled from "Hello.java"  
public class Hello extends java.lang.Object{  
    public Hello();
```

Code:

```
0:  aload_0  
1:  invokespecial #1; //Method java/lang/Object."<init>":()V  
4:  return
```

```
public static void main(java.lang.String[]);
```

Code:

```
0:  getstatic #2; //Field java/lang/System.out:Ljava/io/PrintStream;  
3:  ldc #3; //String Hello, World!  
5:  invokevirtual #4; //Method java/io/PrintStream.println:(Ljava/lang/String;)V
```

Что такое #1, #2, и тд?

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c -verbose Hello**

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
> javap -c -verbose Hello  
Compiled from "Hello.java"  
public class Hello extends java.lang.Object  
  SourceFile: "Hello.java"  
  minor version: 0  
  major version: 50  
  Constant pool:  
const #1 = Method    #6.#20; // java/lang/Object."<init>":()V  
const #2 = Field     #21.#22;   // java/lang/System.out:Ljava/io/PrintStream;  
const #3 = String    #23; // Hello, World!  
const #4 = Method    #24.#25;   // java/io/PrintStream.println:(Ljava/lang/String;)V  
const #5 = class     #26; // Hello  
const #6 = class     #27; // java/lang/Object  
const #7 = Asciz    <init>;
```

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c -verbose Hello**

Compiled from "Hello.java"

public class Hello extends java.lang.Object

 SourceFile: "Hello.java"

 minor version: 0

 major version: 50

Constant pool:

const #1 = Method	#6.#20; // java/lang/Object."<init>":()V
const #2 = Field	#21.#22; // java/lang/System.out:Ljava/io/PrintStream;
const #3 = String	#23; // Hello, World!
const #4 = Method	#24.#25; // java/io/PrintStream.println:(Ljava/lang/String;)V
const #5 = class	#26; // Hello
const #6 = class	#27; // java/lang/Object
const #7 = Asciz	<init>;

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c -verbose Hello**

```
public class Hello extends java.lang.Object
```

...

```
public Hello();
```

Code:

```
Stack=1, Locals=1, Args_size=1
```

```
0: aload_0
```

```
1: invokespecial #1; //Method java/lang/Object."<init>":()V
```

```
4: return
```

LineNumberTable:

```
line 1: 0
```

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LHello;

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c -verbose Hello**

```
public class Hello extends java.lang.Object
```

...

```
public Hello();
```

Code:

Stack=1, Locals=1, Args_size=1

```
0: aload_0  
1: invokespecial #1; //Method java/lang/Object."<init>":()V  
4: return
```

LineNumberTable:

line 1: 0

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LHello;

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

```
> javap -c -verbose Hello  
public class Hello extends java.lang.Object
```

...

```
public Hello();
```

Code:

```
Stack=1, Locals=1, Args_size=1
```

```
0:  aload_0
```

```
1:  invokespecial #1; //Method java/lang/Object."<init>":()V
```

```
4:  return
```

LineNumberTable:

```
line 1: 0
```

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LHello;

```
public class Hello {  
    public static void main(String[] args) {  
        System.out.println("Hello, World!");  
    }  
}
```

> **javap -c -verbose Hello**

```
public class Hello extends java.lang.Object
```

...

```
public Hello();
```

Code:

```
Stack=1, Locals=1, Args_size=1
```

```
0: aload_0
```

```
1: invokespecial #1; //Method java/lang/Object."<init>":()V
```

```
4: return
```

LineNumberTable:

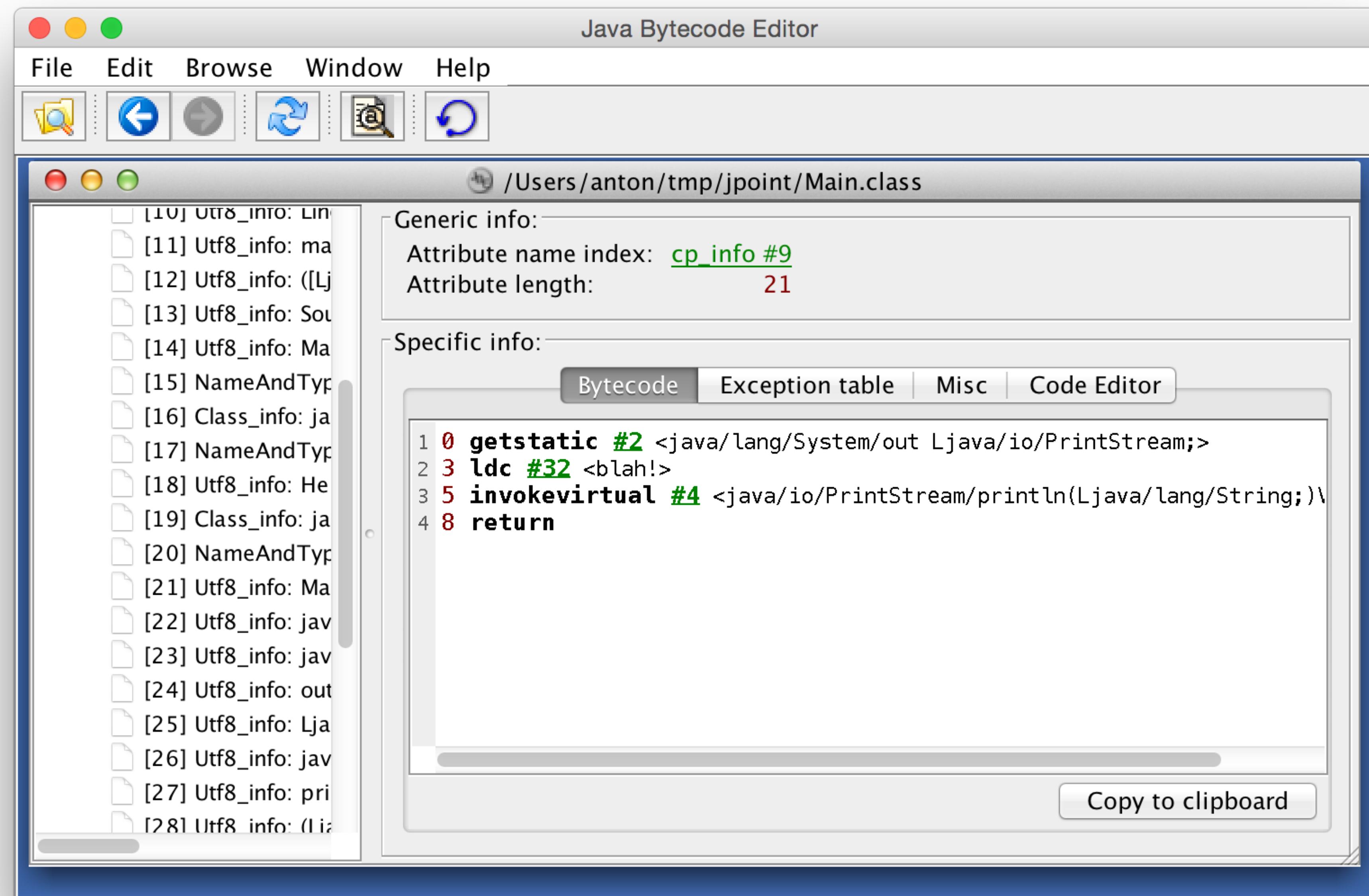
```
line 1: 0
```

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LHello;



Java Bytecode Editor – <http://set.ee/jbe/>



IntelliJ IDEA: ASM Bytecode Outline plugin

The screenshot shows the IntelliJ IDEA interface with a Java file named `Fib.java` open. The code implements two methods for generating Fibonacci series: one using recursion and one using a loop. The right side of the screen displays the ASM bytecode outline for the `main` method, showing the assembly instructions generated by the Java compiler.

```
1 package com.zt;
2
3 import java.util.Scanner;
4
5
6 public class Fib {
7     public static void main(String[] args) {
8         //input to print Fibonacci series upto how many
9         System.out.print("Enter number upto which Fibona");
10
11         int number = new Scanner(System.in).nextInt();
12         System.out.println("\n\nFibonacci series upto ");
13         //printing Fibonacci series upto number
14         for(int i=1; i<=number; i++){
15             System.out.print(fibonacciRecusion(i) + " ");
16             System.out.print(fibonacciLoop(i) + " ");
17         }
18     }
19
20
21     // Java program for Fibonacci number using recursion
22     public static int fibonacciRecusion(int number){
23         if(number == 1 || number == 2){
24             return 1;
25         }
26
27         return fibonacciRecusion(number-1) + fibonacciRe;
28     }
29
30     // Java program for Fibonacci number using Loop.
31     public static int fibonacciLoop(int number){
32         if(number == 1 || number == 2){
33             return 1;
34         }
35     }
36 }
```

ASM: Bytecode ASMified Groovified

Show differences Settings

```
6
7     // access flags 0x1
8     public <init>()V
9
L0
10    LINENUMBER 6 L0
11    ALOAD 0
12    INVOKESPECIAL java/lang/Object.<init> ()V
13    RETURN
L1
14    LOCALVARIABLE this Lcom/zt/Fib; L0 L1 0
15    MAXSTACK = 1
16    MAXLOCALS = 1
17
18
19     // access flags 0x9
20     public static main([Ljava/lang/String;)V
21
L0
22    LINENUMBER 9 L0
23    GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
24    LDC "Enter number upto which Fibonacci series to print: "
25    INVOKEVIRTUAL java/io/PrintStream.print (Ljava/lang/String;)
L1
26
27    LINENUMBER 11 L1
28    NEW java/util/Scanner
29    DUP
30    GETSTATIC java/lang/System.in : Ljava/io/InputStream;
31    INVOKESPECIAL java/util/Scanner.<init> (Ljava/io/InputStream;
32    INVOKEVIRTUAL java/util/Scanner.nextInt ()I
33    ISTORE 1
L2
34    LINENUMBER 12 L2
```

```
public class Get {  
    String name;  
  
    public String getName() {  
        return name;  
    }  
}
```

public java.lang.String getName();

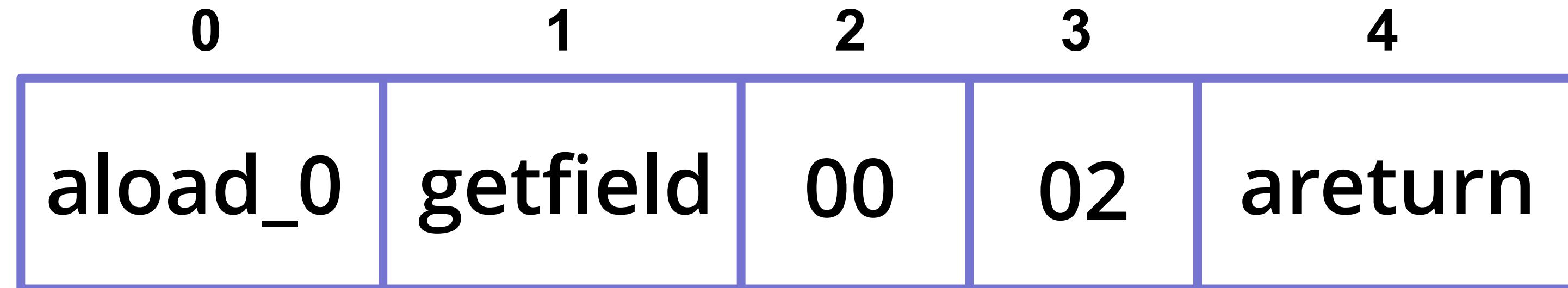
Code:

Stack=1, Locals=1, Args_size=1

0: **aload_0**
1: **getfield #2; //Field name:Ljava/lang/String;**
4: **areturn**

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LGet;



public java.lang.String getName();

Code:

Stack=1, Locals=1, Args_size=1

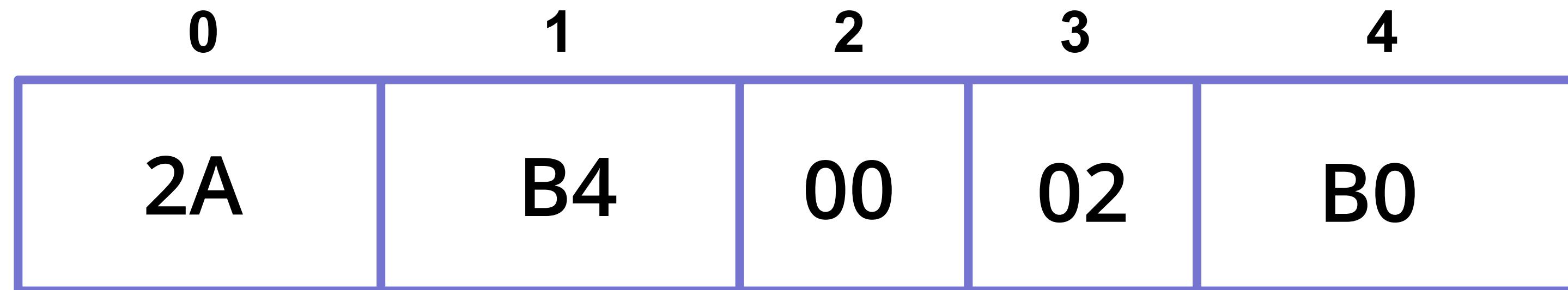
0: **aload_0**

1: **getfield** #2; //Field name:Ljava/lang/String;

4: **areturn**

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LGet;



public java.lang.String getName();

Code:

Stack=1, Locals=1, Args_size=1

```

0:  aload_0
1:  getfield    #2; //Field name:Ljava/lang/String;
4:  areturn

```

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LGet;

29	4C	6A	61	76	61	2F	6C	61	6E	67	3B	01	00	0A	53	6F	75	72	63	65	2F	53	74	72	69					
6E	67	3B	01	00	0A	53	6F	61	76	61	0C	00	08	47	65	74	61	76	61	0C	00	07	00	08	6C	65				
01	00	08	47	65	74	2E	6A	61	76	61	0C	00	05	00	06	01	00	03	47	65	74	01	00	10	6A	61				
0C	00	05	00	06	01	00	03	76	61	2F	6C	61	6E	67	2F	4F	62	6A	65	63	74	00	21	00	21	6A	61			
00	03	00	04	00	00	00	01	00	03	00	01	00	07	00	08	00	00	00	05	00	00	06	00	00	00	00	00	00		
00	02	00	01	00	07	00	08	00	01	00	01	00	01	00	05	00	01	00	09	00	00	00	00	00	00	00	00	2F		
00	01	00	01	00	00	00	05	00	01	00	01	00	01	00	05	00	01	B7	00	01	B1	00	00	00	00	00	00	00		
02	00	0A	00	00	00	06	00	00	00	0C	00	01	00	00	06	00	01	00	00	00	01	00	00	0B	00	00	0B	00		
00	00	0C	00	01	00	00	00	00	01	00	00	01	00	00	00	00	05	00	0C	00	00	0D	00	00	00	00	00	00		
01	00	0E	00	0F	00	01	00	00	01	00	00	00	00	00	00	00	09	00	00	00	00	00	00	00	00	2F	00	01	00	
01	00	00	00	05	2A	B4	00	00	02	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0A	
00	00	00	06	00	01	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	0C	
00	01	00	00	00	05	00	0C	00	00	01	00	00	00	00	00	00	00	00	00	00	00	00	01	00	00	10	00	00	00	
00	00	00	02	00	11																									

public java.lang.String getName();

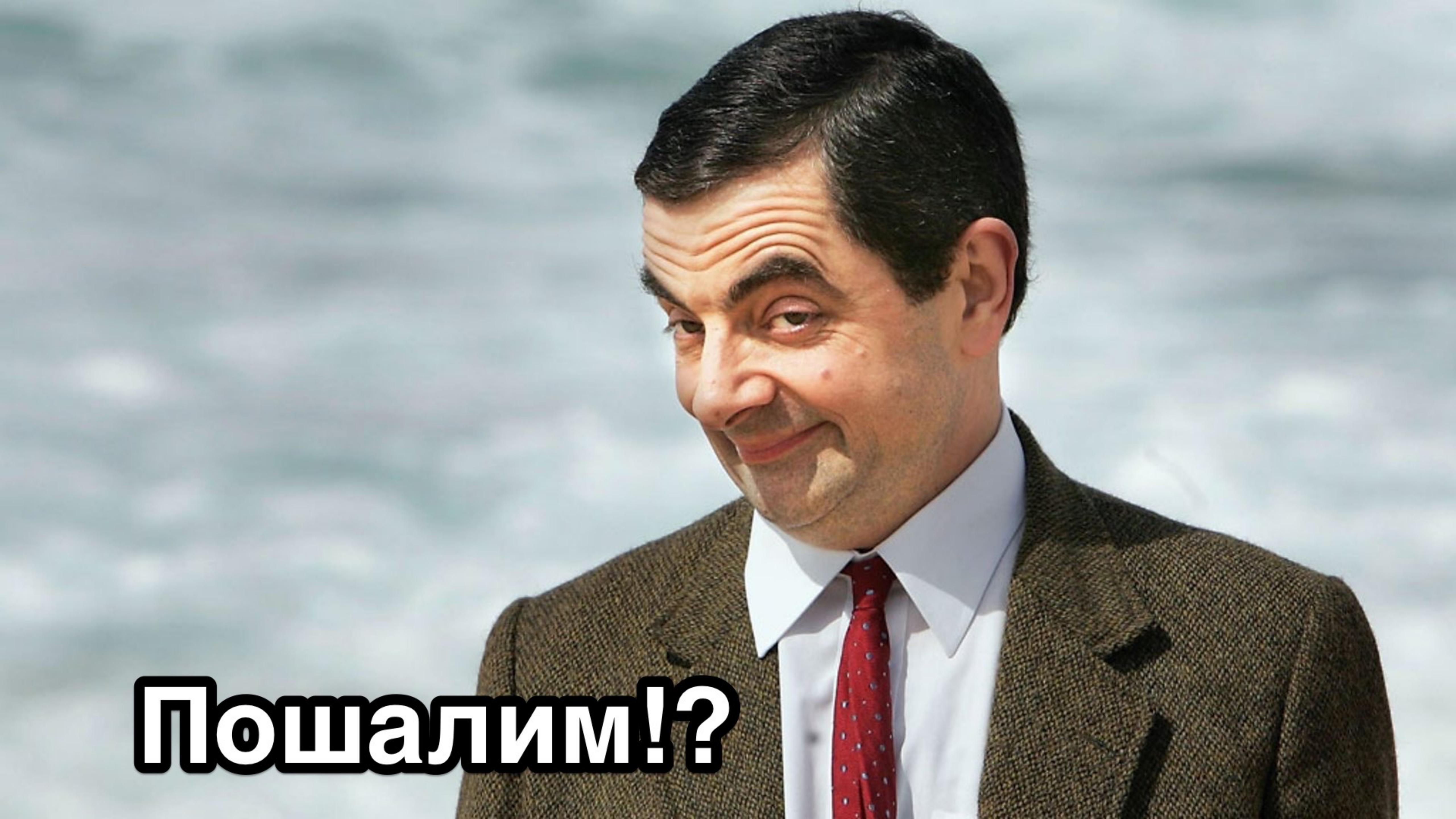
Code:

Stack=1, Locals=1, Args_size=1

0: **aload_0**
 1: **getfield #2; //Field name:Ljava/lang/String;**
 4: **areturn**

LocalVariableTable:

Start	Length	Slot	Name	Signature
0	5	0	this	LGet;



Пошалим!?

ObjectWeb ASM

- Низкоуровневый фреймворк для манипуляций и анализа Java-байткода
- Стандарт *de facto*
- <http://asm.ow2.org>



**Сначала байткод, пуш, поп...
теперь ASM..**

Зачем?!?!

Зачем манипулировать байткодом (ещё раз)

- Профилировщики
- Агенты для мониторинга
- Дебаггеры
- Фреймворки
- JRebel

Зачем манипулировать байткодом (ещё раз)

- Профилировщики
- Агенты для мониторинга
- Дебаггеры
- Фреймворки
- JRebel



Требуется внедрение
функциональности в
работающий код



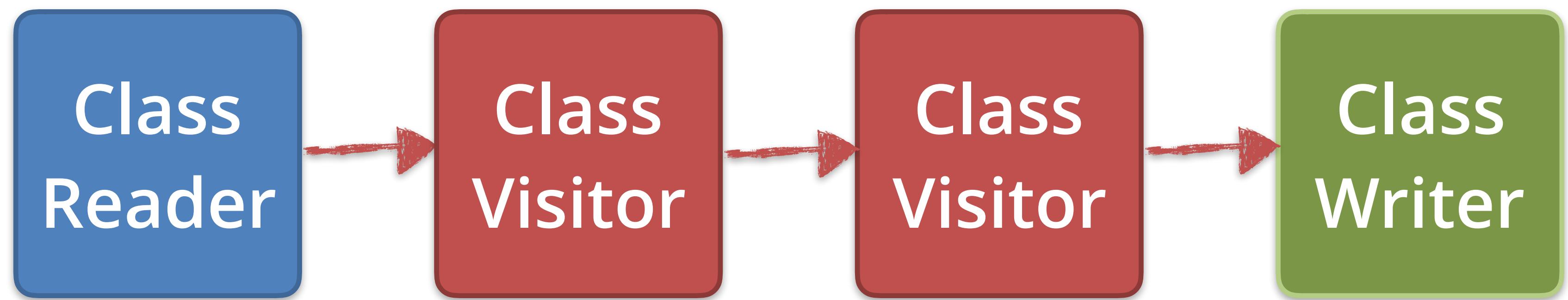
ASM: общий сценарий

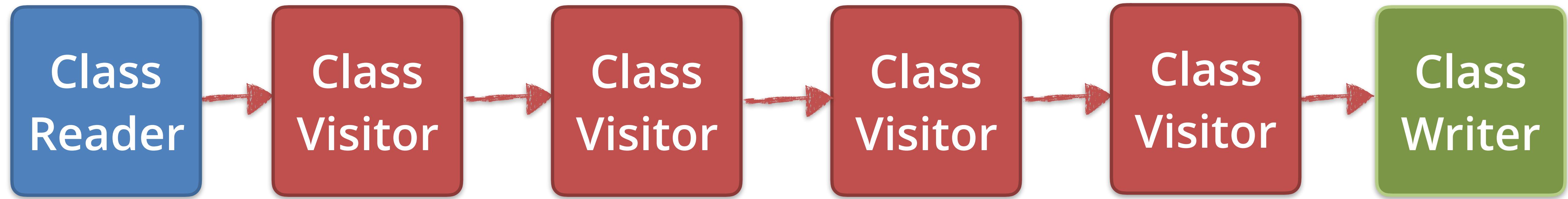
- 1 Создать ClassWriter
- 2 Скомпоновать visitor-ы:
аннотации, поля, методы, итд
- 3 Генерировать новый байткод

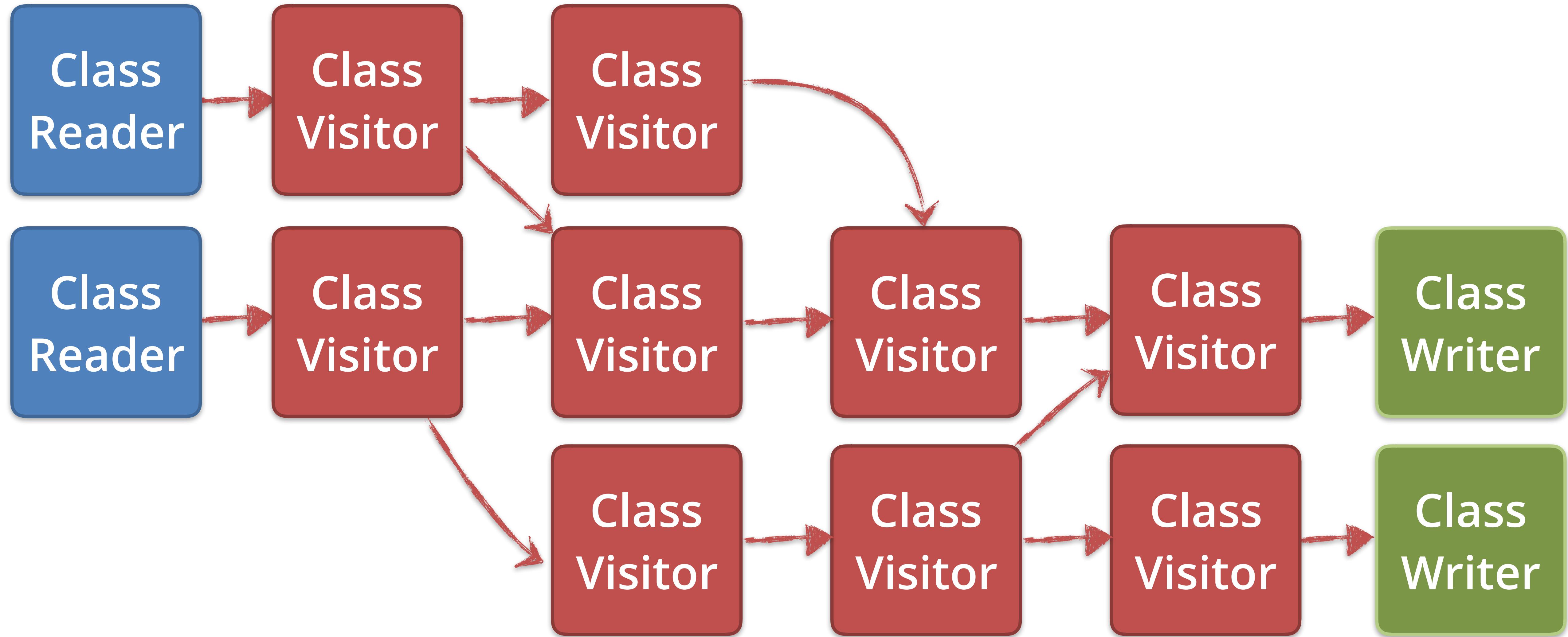
Class
Reader

Class
Visitor









ClassReader

ClassVisitor

visit

visitSource

visitOuterClass

visitAnnotation

visitAttribute

visitInnerClass

visitField

visitMethod

visitEnd

ClassReader

ClassVisitor

visit

visitSource

visitOuterClass

visitAnnotation

visitAttribute

visitInnerClass

visitField

visitMethod

visitEnd

ClassVisitor

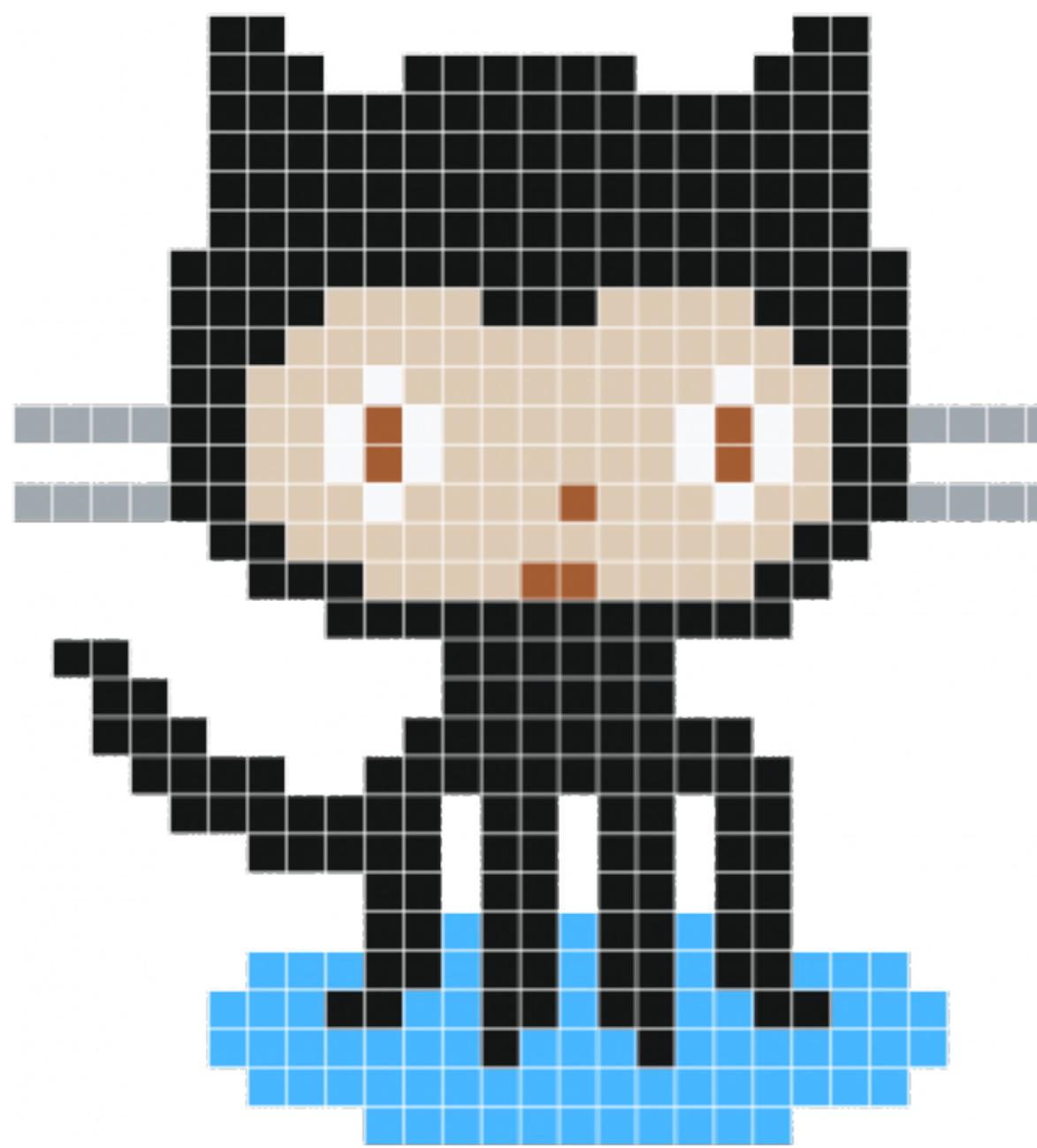
AnnotationVisitor

FieldVisitor

MethodVisitor

**Ты код
покажешь
когда нибудь, а?**





<https://github.com/antonarhipov/asmdemo>

THANKS FOR LISTENING



ANY QUESTIONS

memegenerator.net