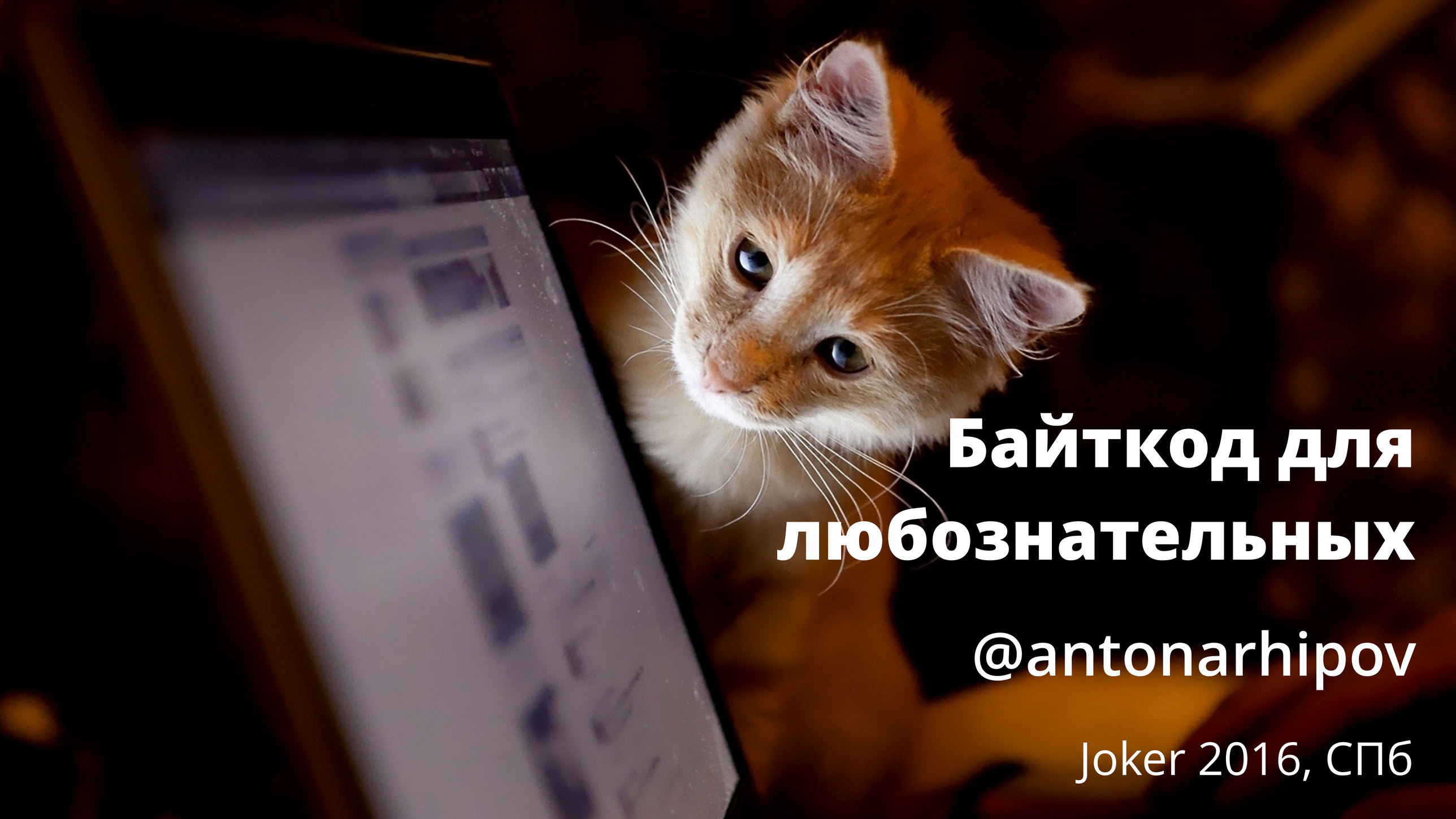


Байткод для любознательных

@antonarhipov

Joker 2016, СПб

Антон Архипов

@antonarhipov

ZEROTURNAROUND

JRebel XRebel

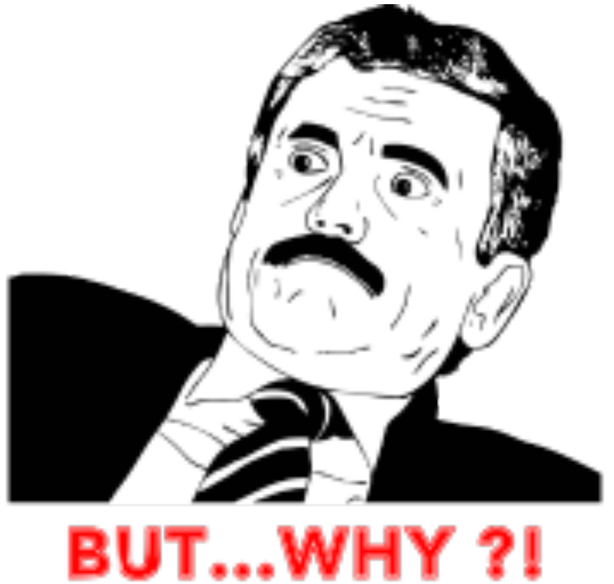
Java Champions

geekOUT.ee

devclub.eu

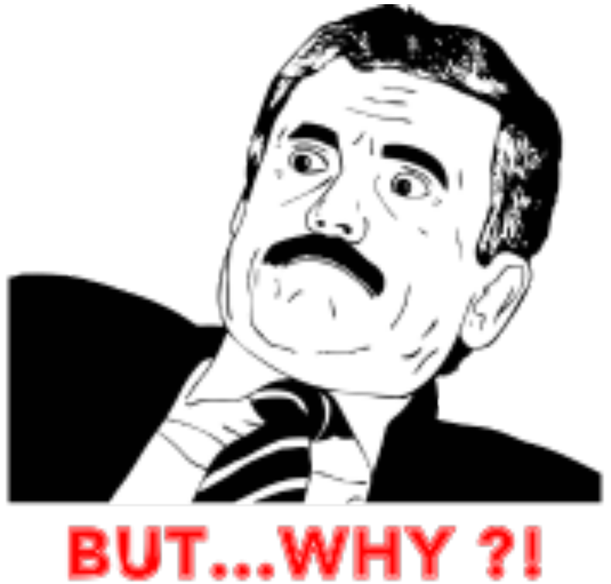
РАЗБОР ПОЛЕТОВ

Зачем?



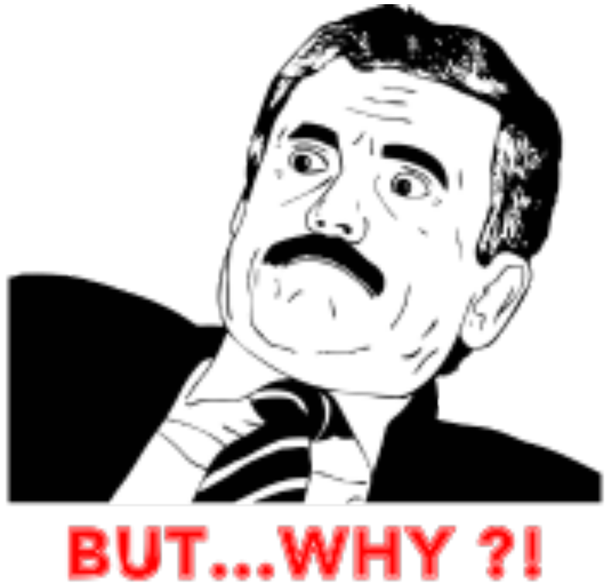
Зачем?

- Знай свою платформу!



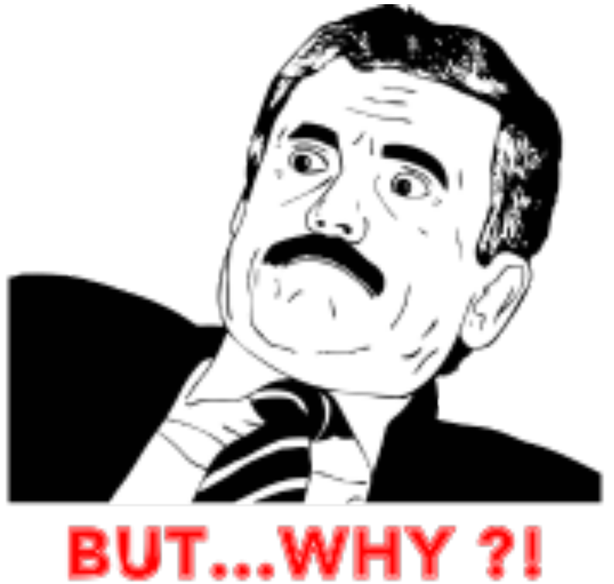
Зачем?

- Знай свою платформу!
- Хотите написать свой компилятор?



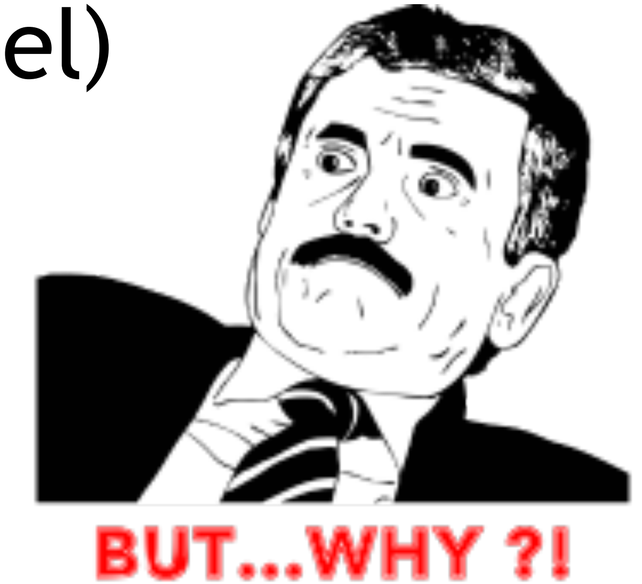
Зачем?

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



Зачем?

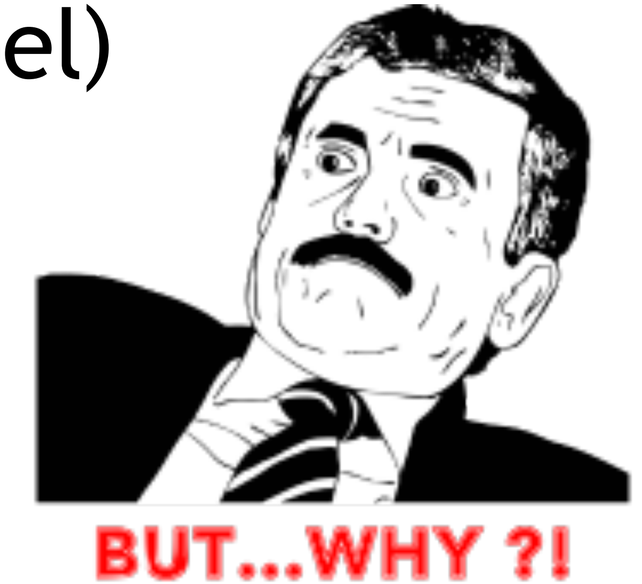
- Знай свою платформу!
- Хотите написать свой компилятор?
- Фреймворки (AOP, ORM)
- Всевозможные инструменты (см. JRebel)



Зачем?

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

- … ну или может просто скучно?



Основы Java-байткода

Инструментарий

ObjectWeb ASM

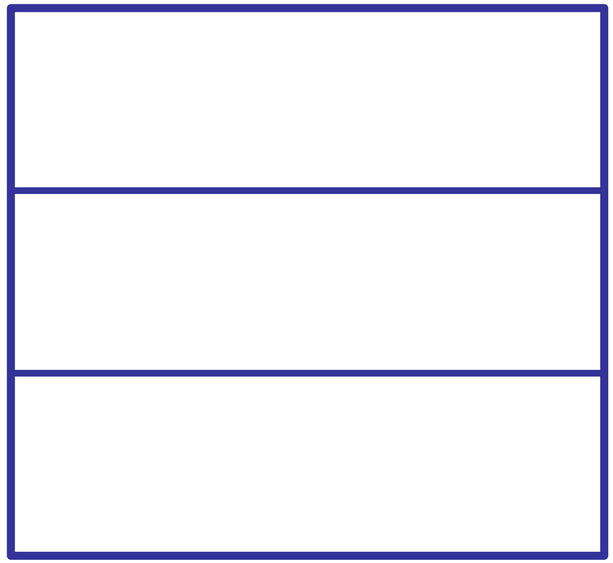
1 + 2

1 + 2

1 2 +

1 + 2

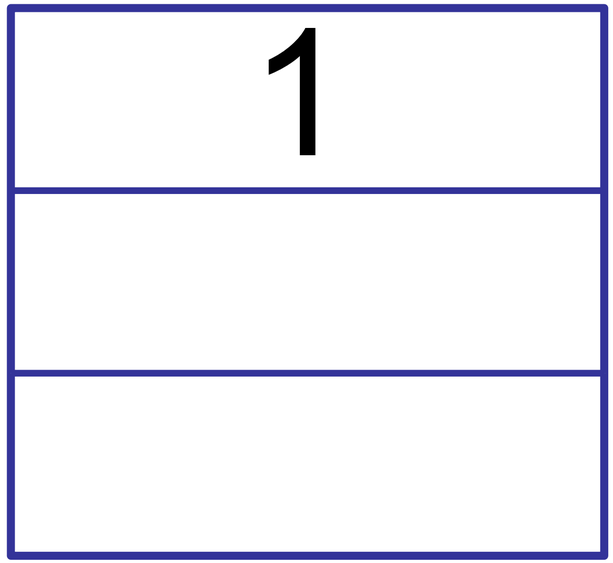
1 2 +



1 + 2

1 2 +

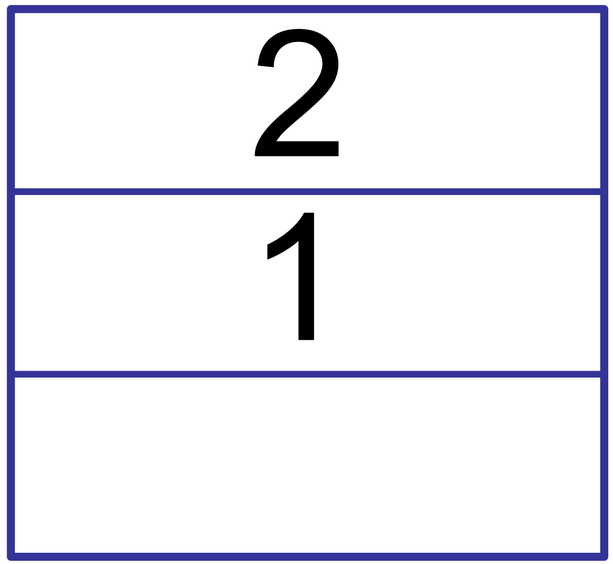
PUSH 1



1 + 2

1 2 +

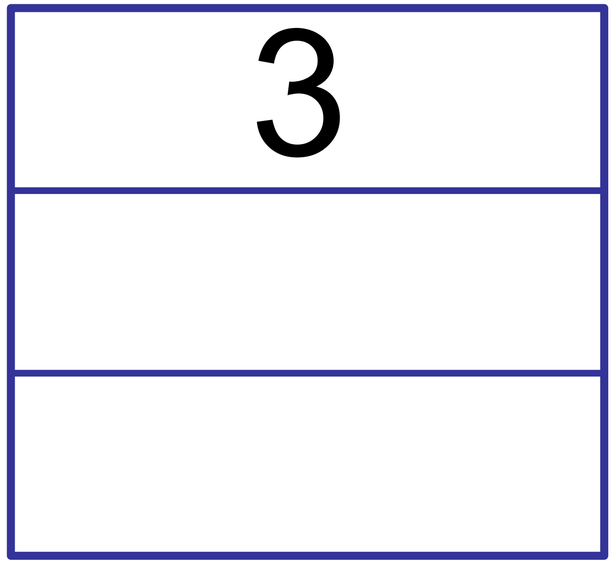
PUSH 1
PUSH 2



1 + 2

1 2 +

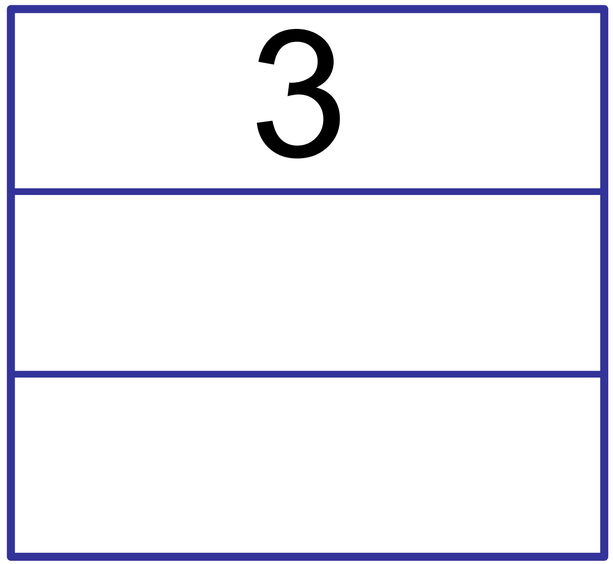
PUSH 1
PUSH 2
ADD



1 + 2

1 2 +

ICONST_1
ICONST_2
IADD



$$? = 1 + 2$$

Таксономия

Таксономия

Таксономия

Таксономия


Работа со стеком
Инструкции управления
Работа с объектами

Таксономия


Работа со стеком
Инструкции управления
Арифметика
Работа с объектами

Таксономия


Работа со стеком
Инструкции управления
monitorenter
monitorexit
Арифметика
Работа с объектами

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**, **iconst_1**)
- Локальные переменные и стек (**load**/**store**)

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**)

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**

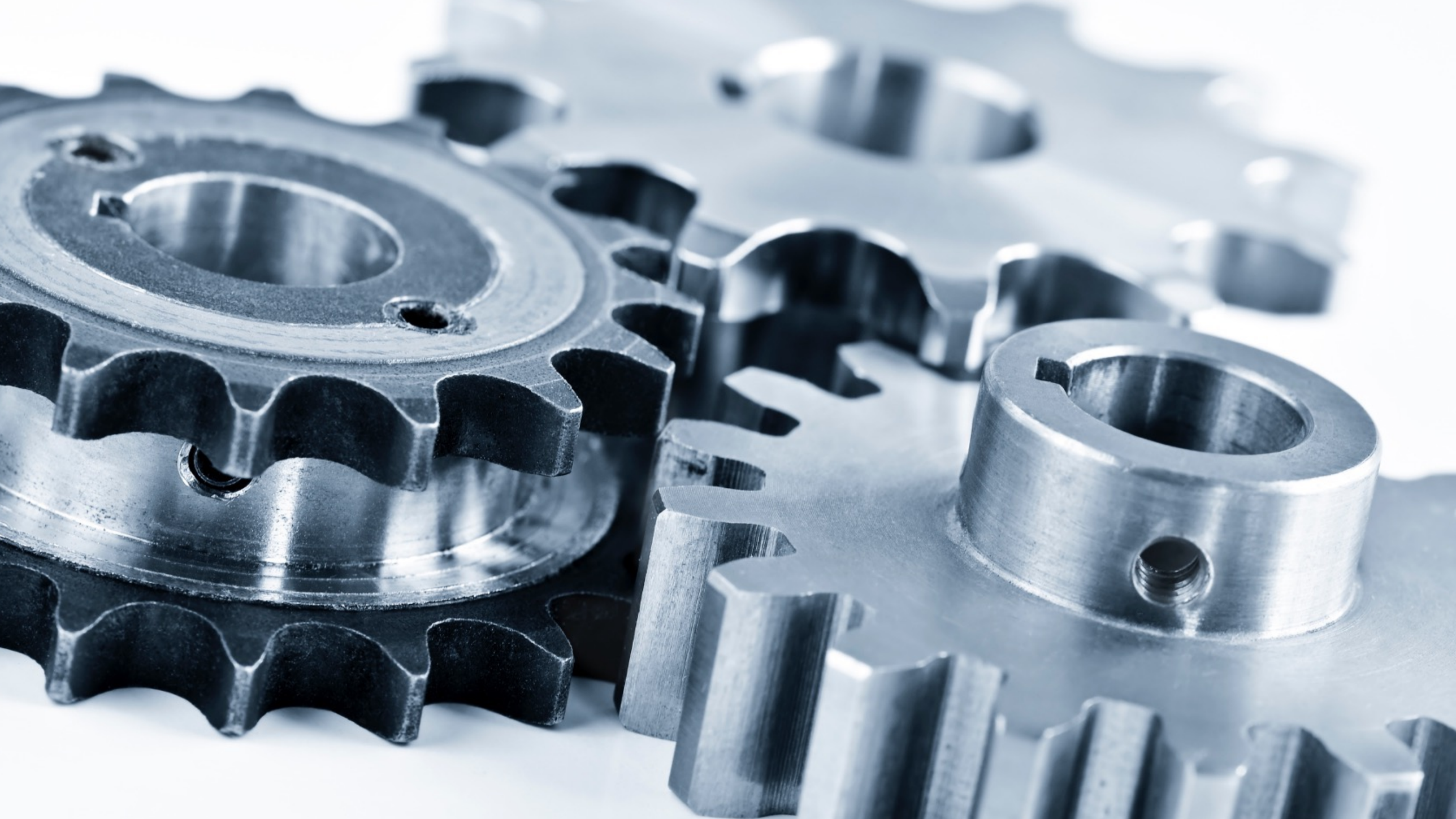
- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc**, **iconst_1**)
- Локальные переменные и стек (**load**/**store**)
- Операции с массивами (**aaload**, **aastore**)
- Арифметика (**add**, **sub**, **mul**, **div**)
- Булевые/битовые операции (**iand**, **ixor**)

ТИП **ОПЕРАЦИЯ**

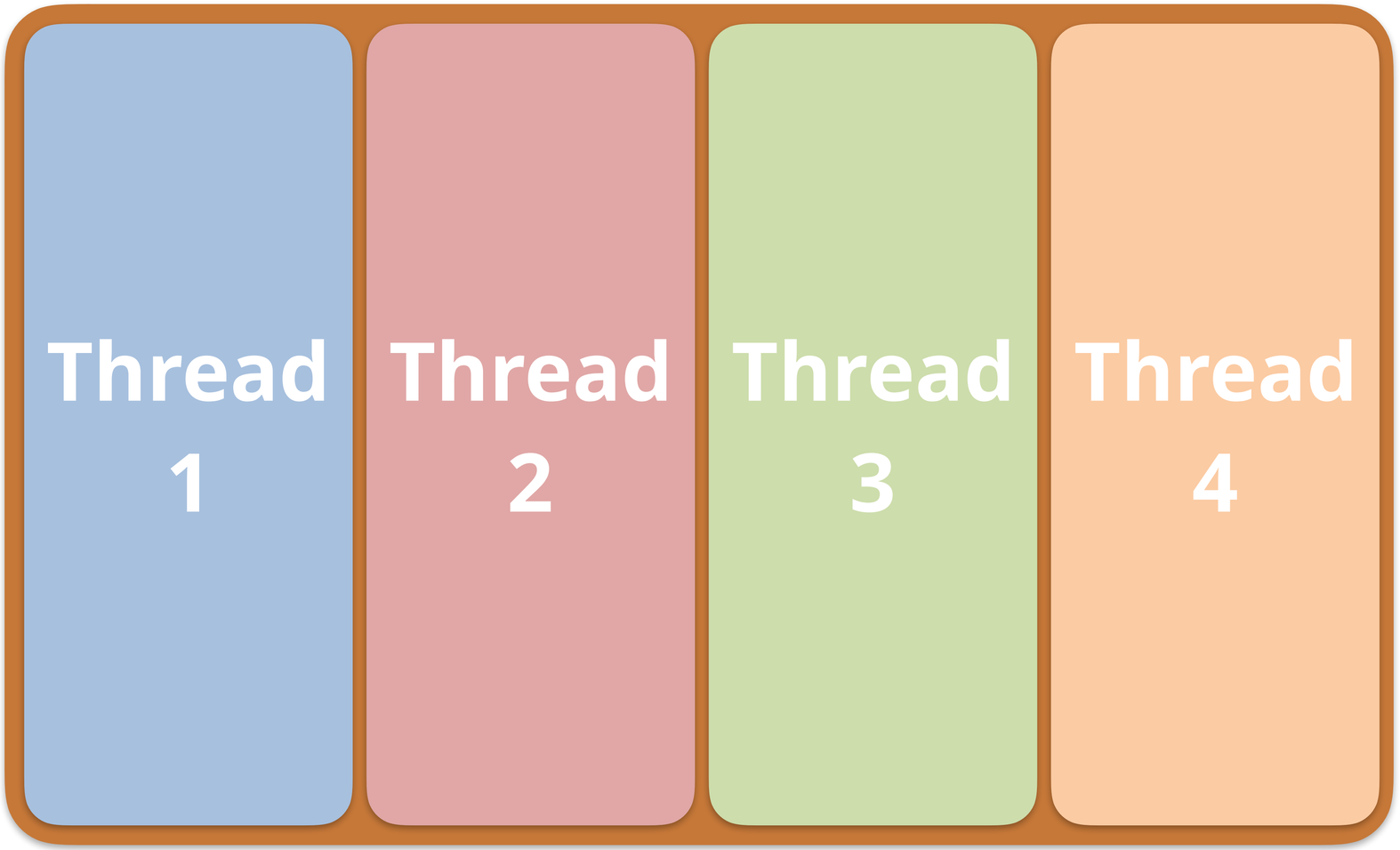
- <тип> ::= b, s, c, i, l, f, d, a
- Константы (**ldc**, **iconst_1**)
- Локальные переменные и стек (**load**/**store**)
- Операции с массивами (**aaload**, **aastore**)
- Арифметика (**add**, **sub**, **mul**, **div**)
- Булевые/битовые операции (**iand**, **ixor**)
- Сравнения (**cmpg**, **cmpl**,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

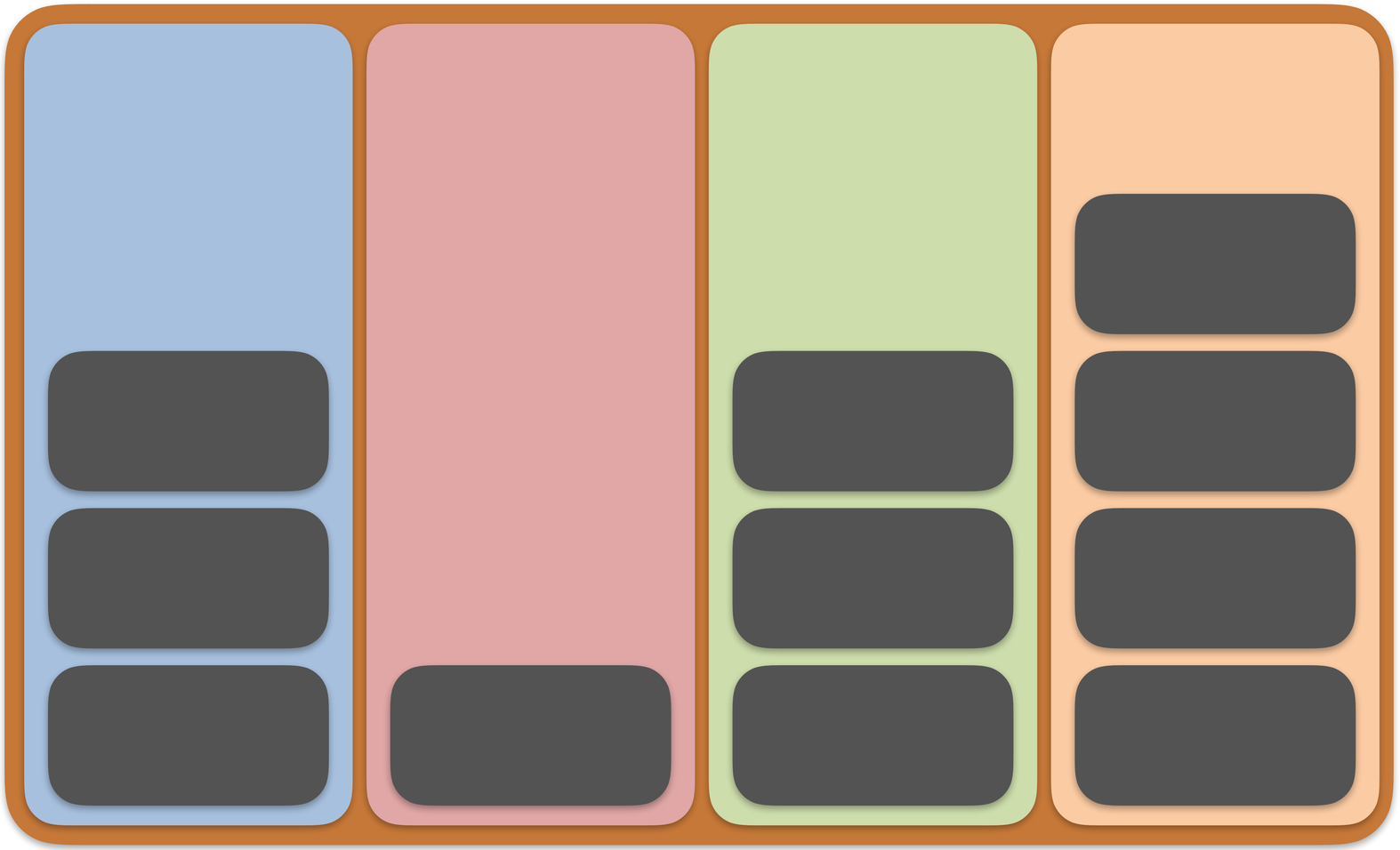
ТИП **ОПЕРАЦИЯ**

- <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 (**cmpg**, **cmpl**, **ifne**, **ifeq**)
- Преобразования (**l2d**, **i2l**)



JVM процесс


Thread 1
Thread 2
Thread 3
Thread 4

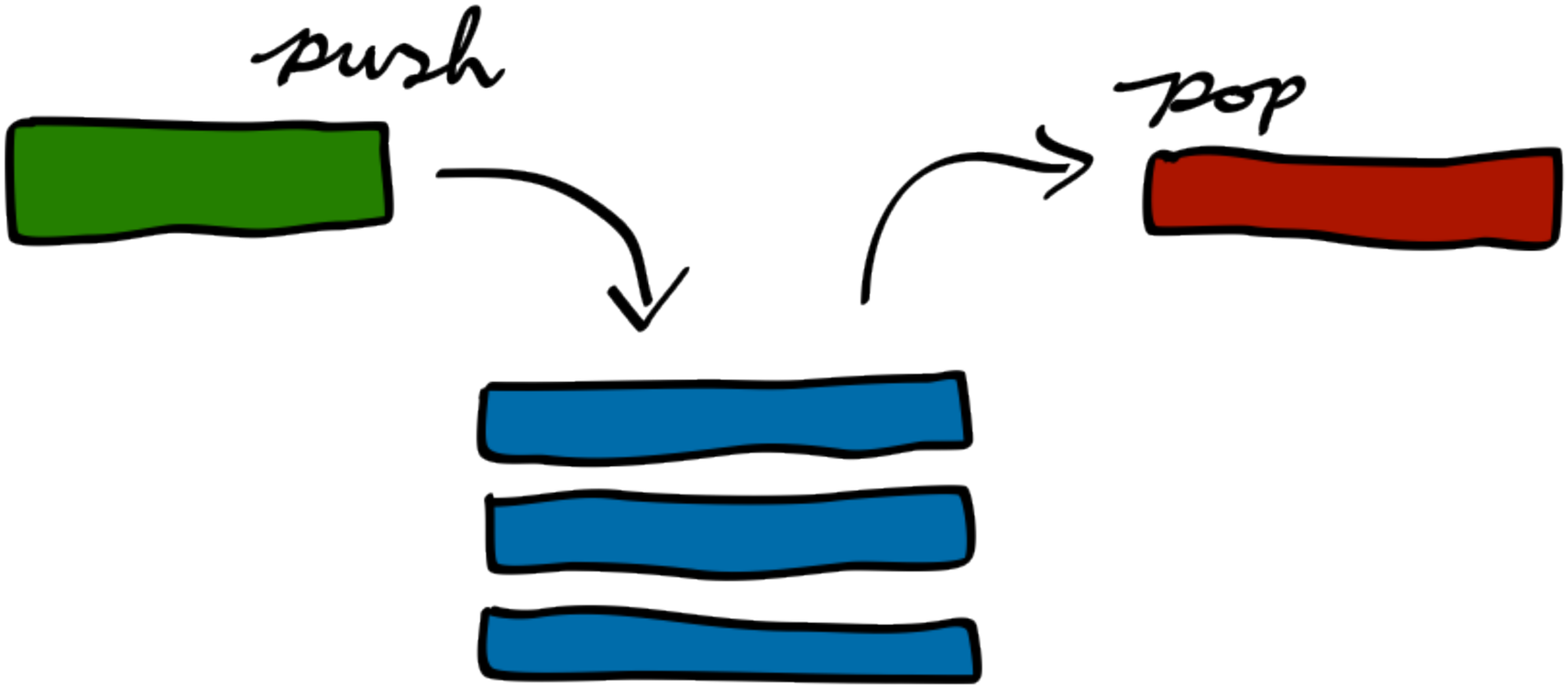


Стековая машина

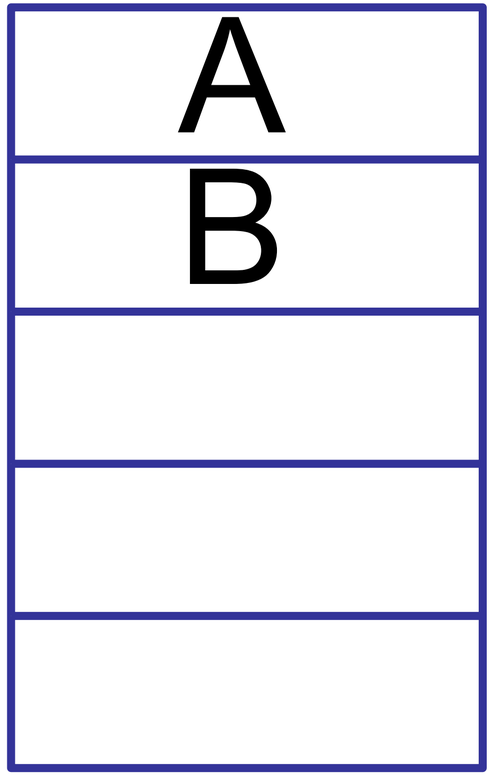
- JVM работает со стеком
- У каждого потока есть стек
- Стек сохраняет “фреймы”
- Новый “фрейм” создаётся при вызове метода
- “Фрейм” состоит из:
  - Стек операций
  - Массив локальных переменных


Локальные переменные
0
1
2
...
N
Стек операций
#1
Константы

Работа со стеком и локальными переменными



dup
pop
swap
dup_x1
dup2_x1



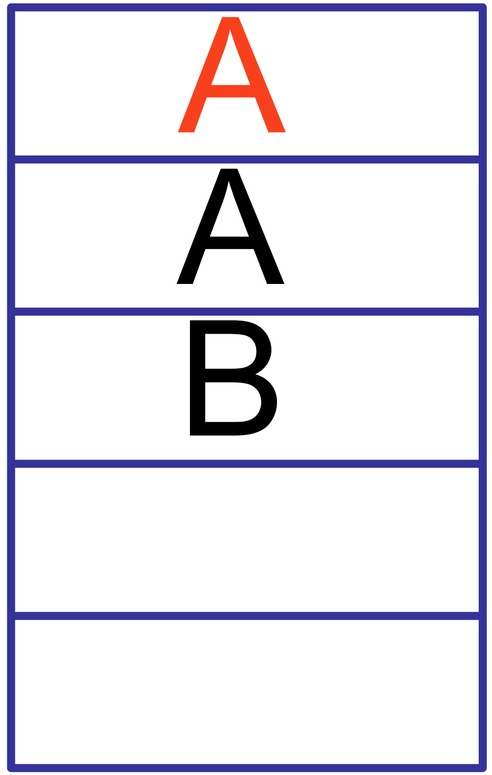
dup

pop

swap

dup_x1

dup2_x1



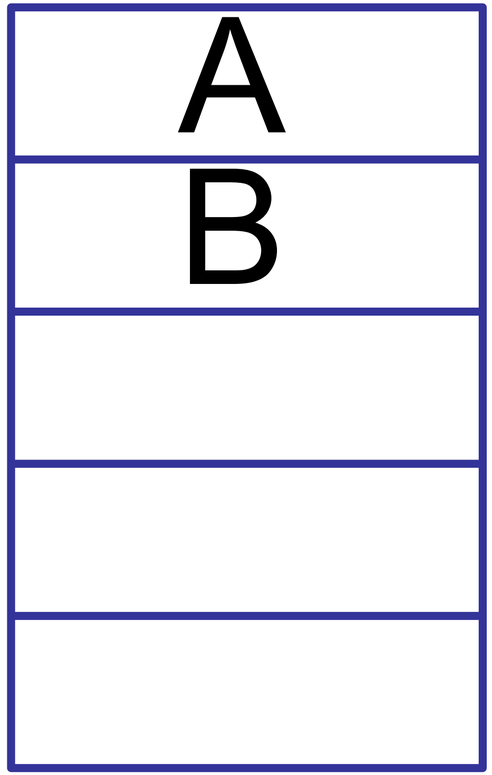
dup

pop

swap

dup_x1

dup2_x1



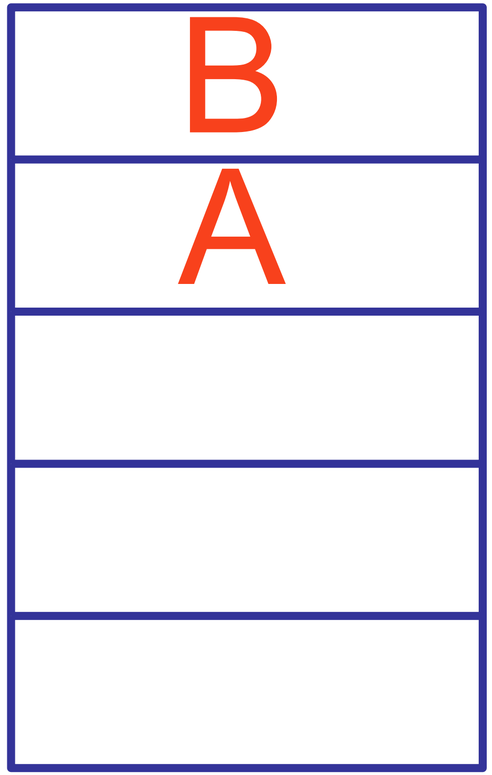
dup

pop

swap

dup_x1

dup2_x1



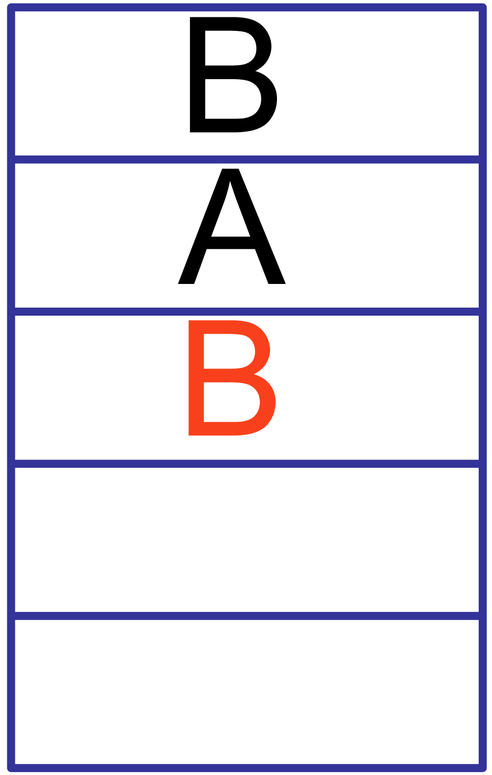
dup

pop

swap

dup_x1

dup2_x1



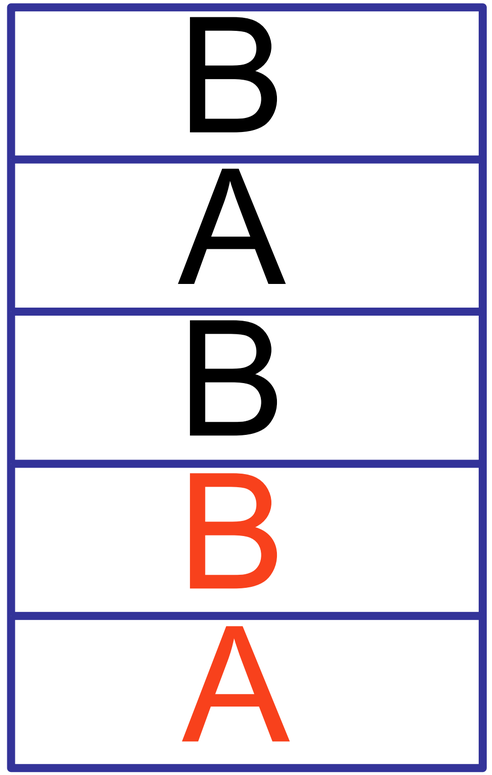
dup

pop

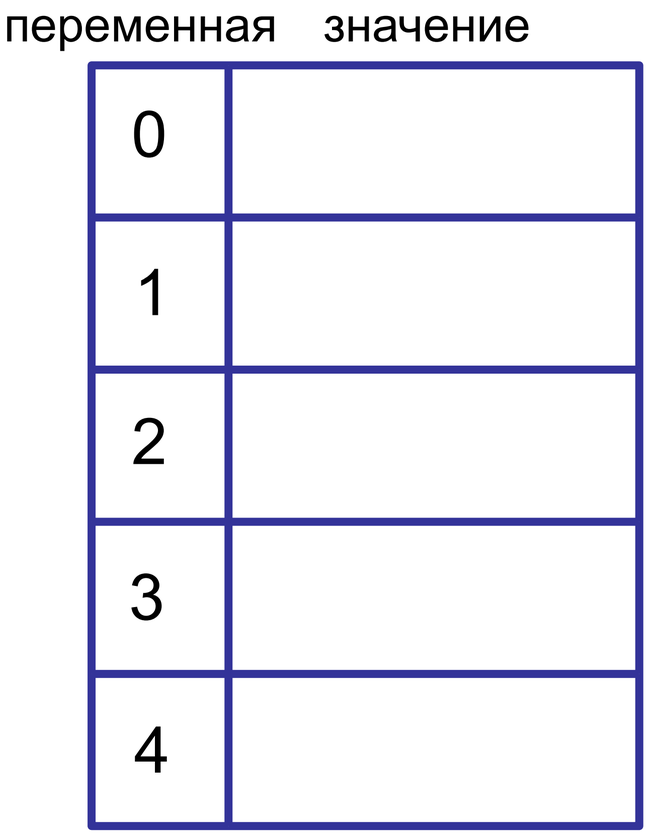
swap

dup_x1

dup2_x1

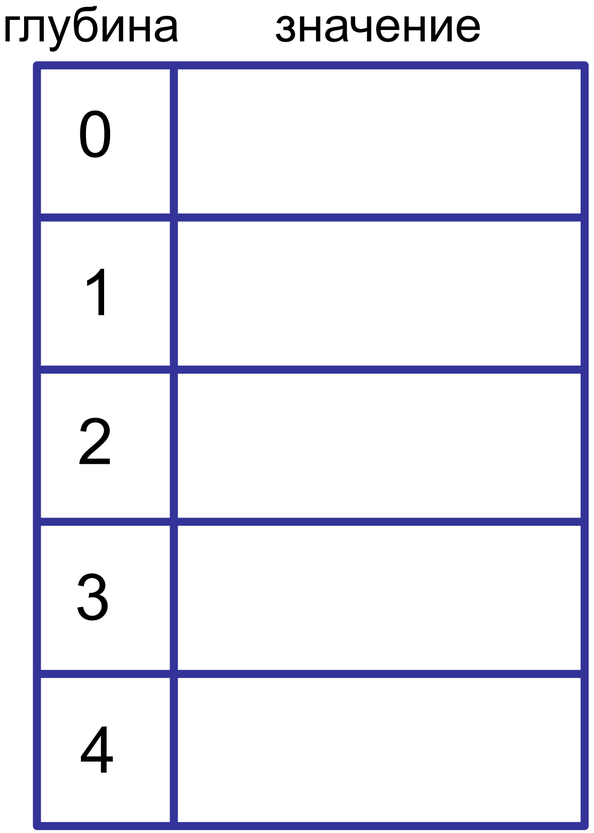


Локальные переменные




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

Стек



Локальные переменные

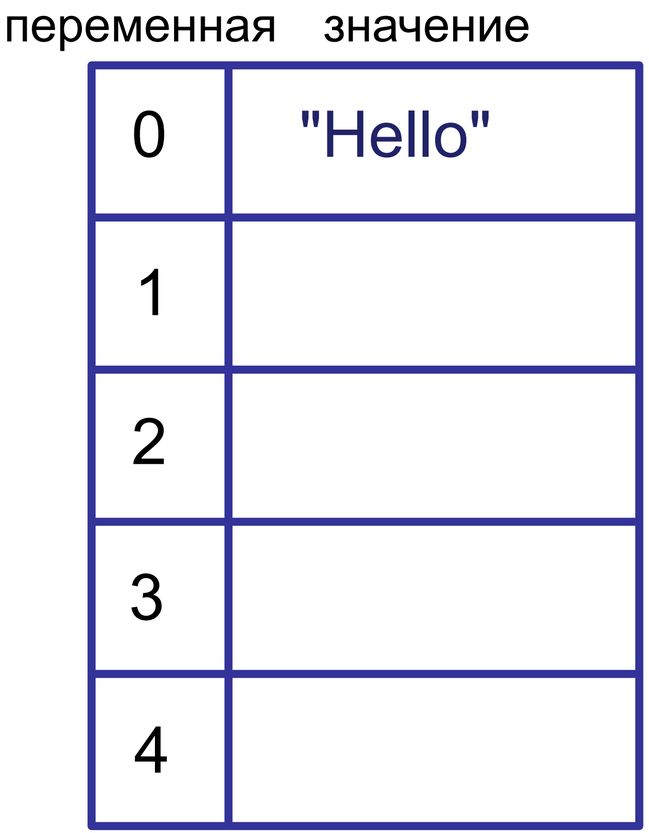
переменная	значение
0	
1	
2	
3	
4	

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

Стек

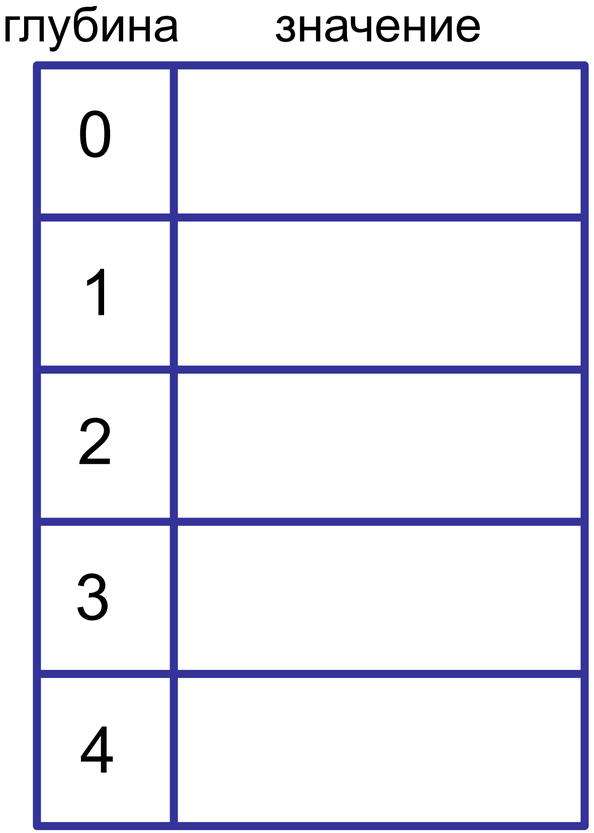
глубина	значение
0	"Hello"
1	
2	
3	
4	

Локальные переменные




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

Стек



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
1	
2	
3	
4	

Локальные переменные

переменная	значение
0	"Hello"
1	1
2	
3	
4	

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

Стек

глубина	значение
0	
1	
2	
3	
4	

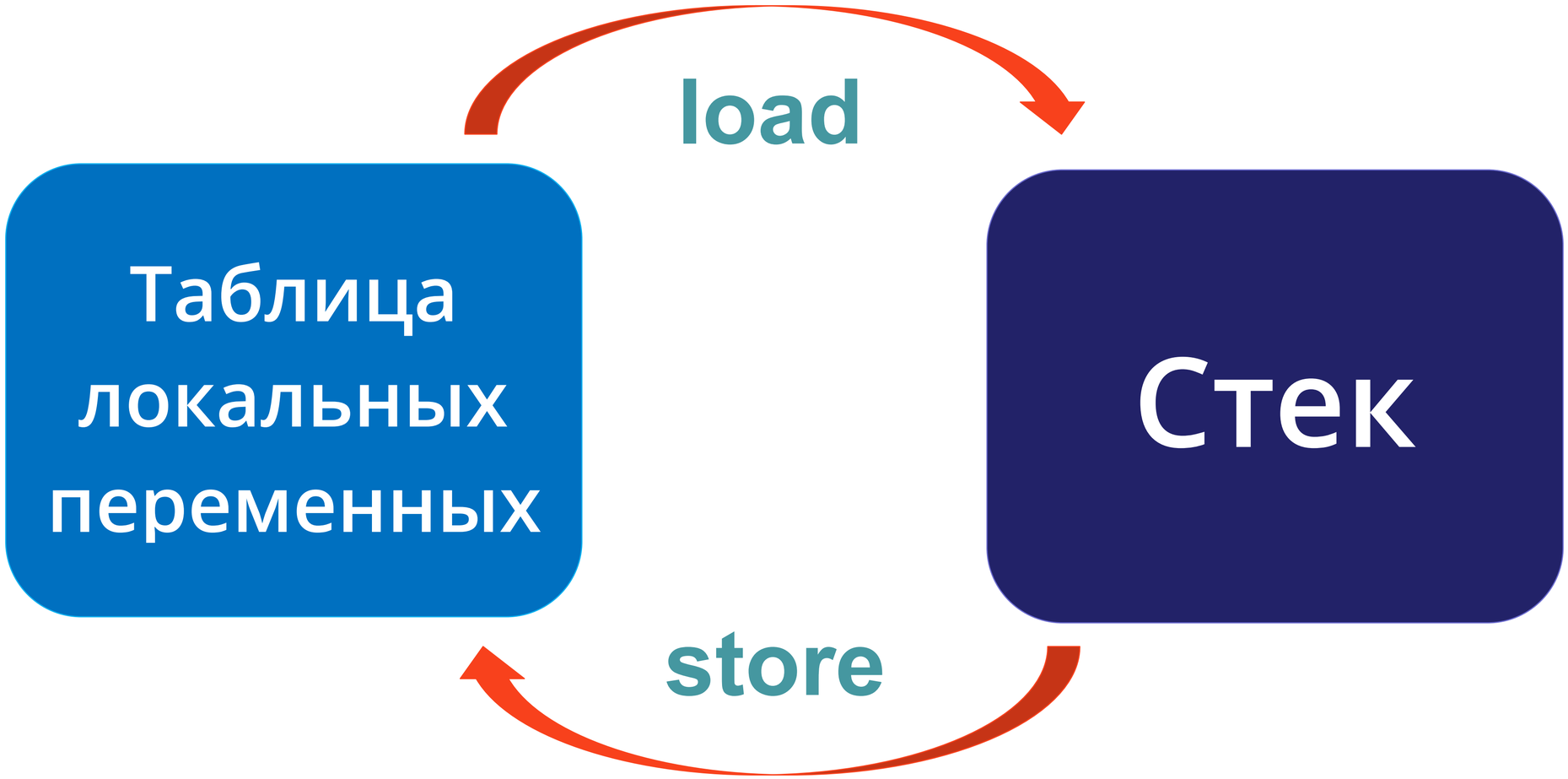
Локальные переменные

переменная	значение
0	"Hello"
1	1
2	
3	
4	

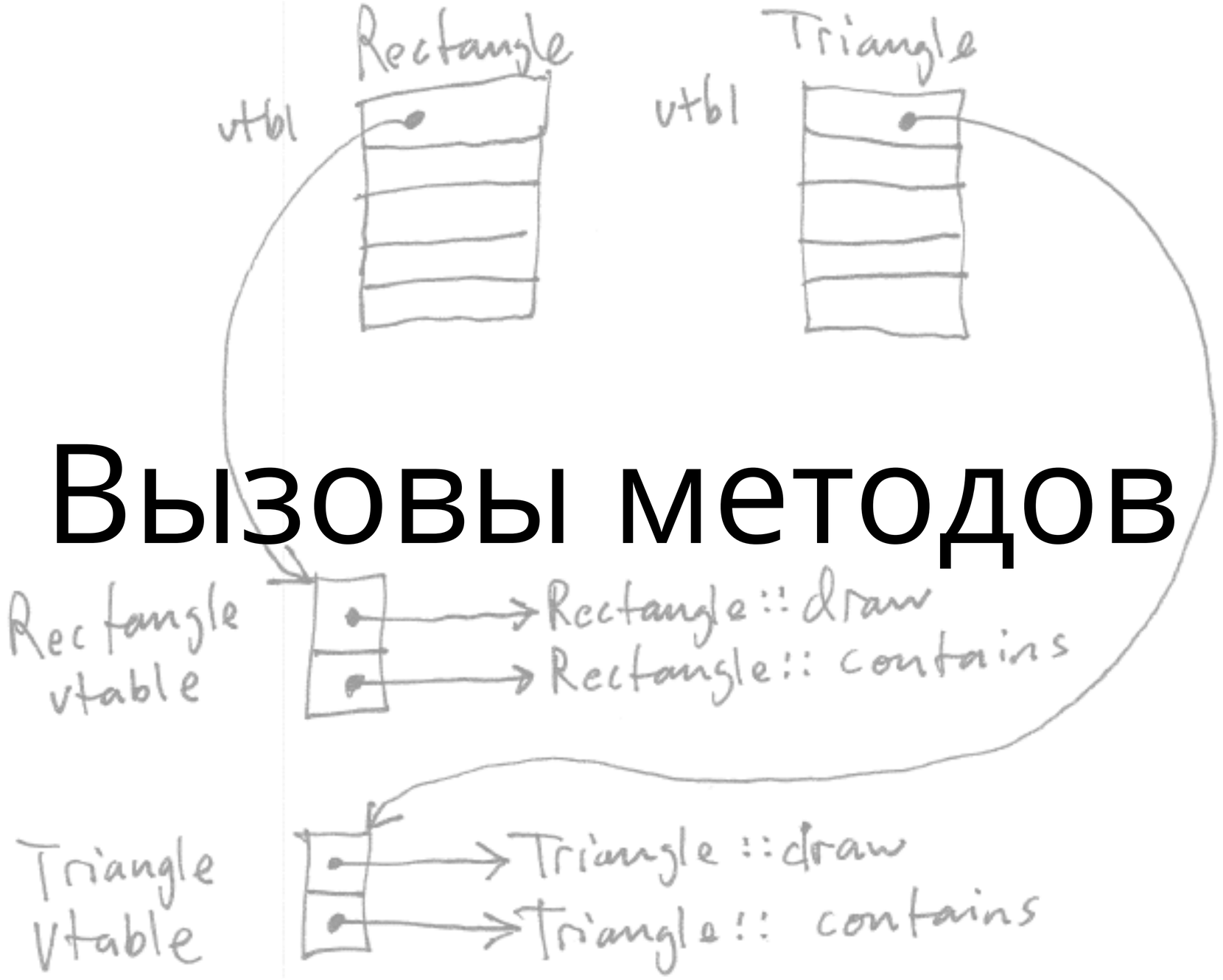
```
ldc "Hello"
astore_0
iconst_1
astore_1
aload_0
```

Стек

глубина	значение
0	"Hello"
1	
2	
3	
4	


load
Таблица локальных переменных
Стек
store

Вызовы методов

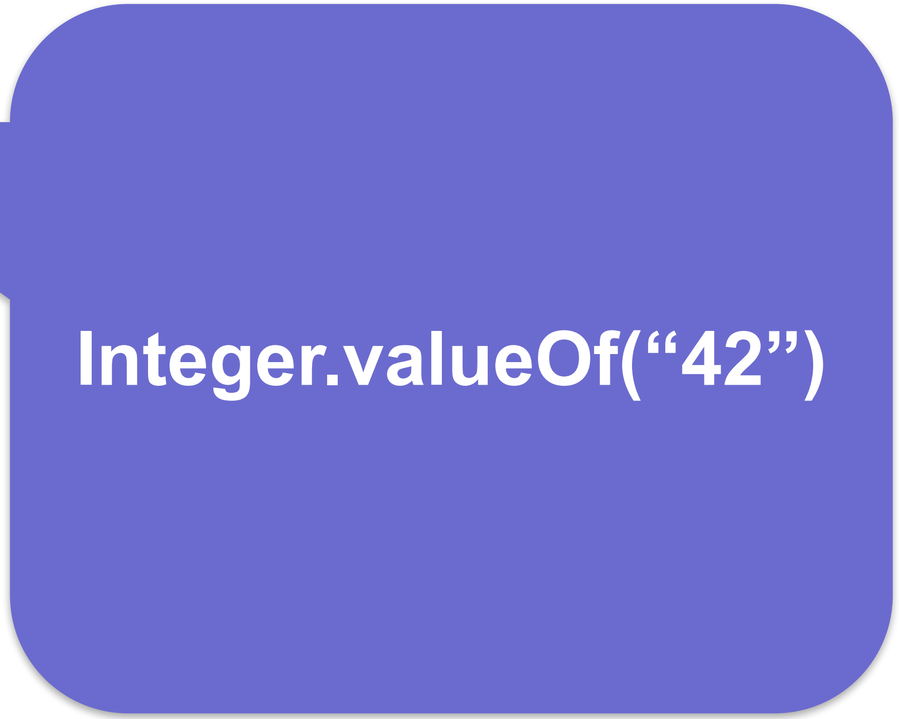


invokeXXX

- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic

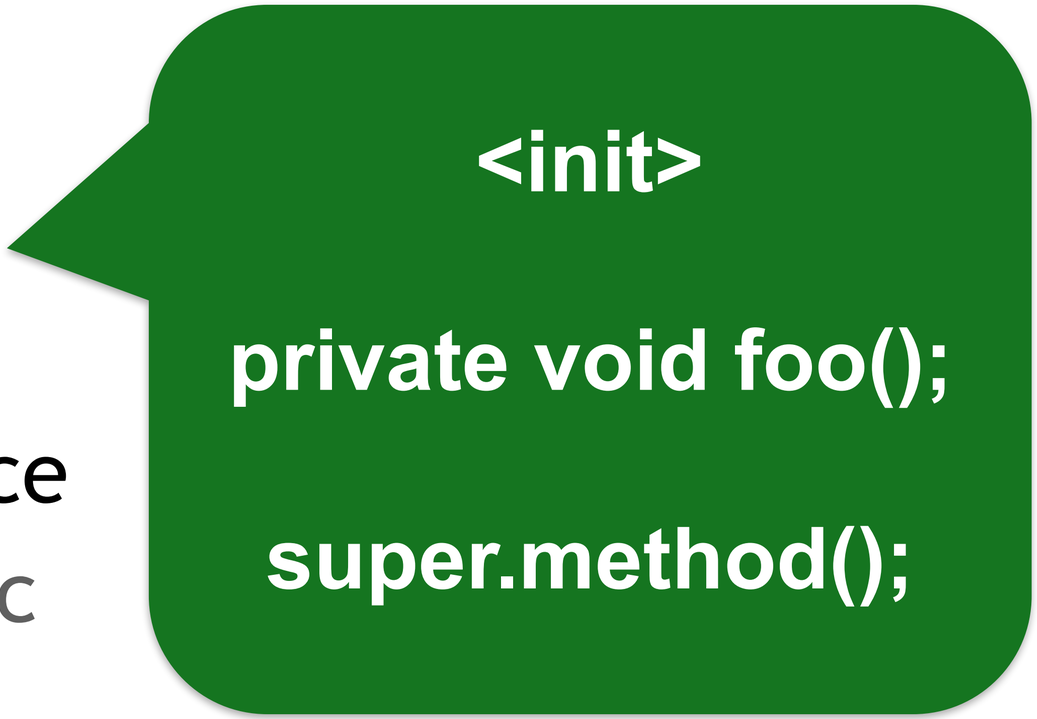
invokestatic

- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic



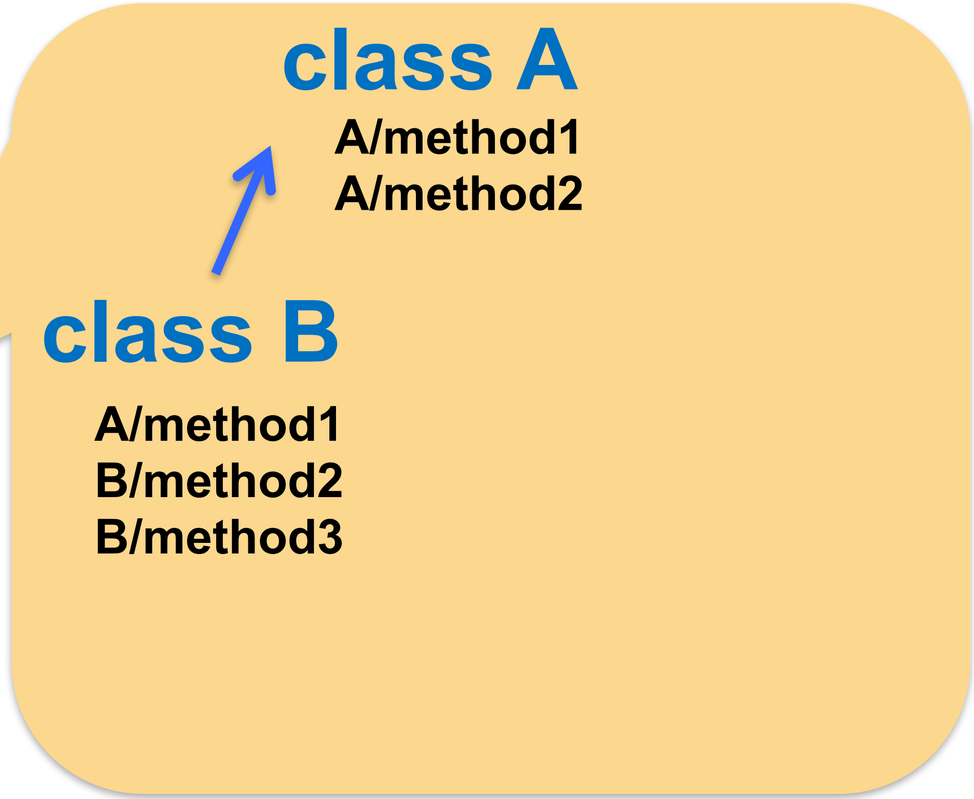
invokespecial

- invokestatic
- invokespecial
- invokevirtual
- invokeinterface
- invokedynamic



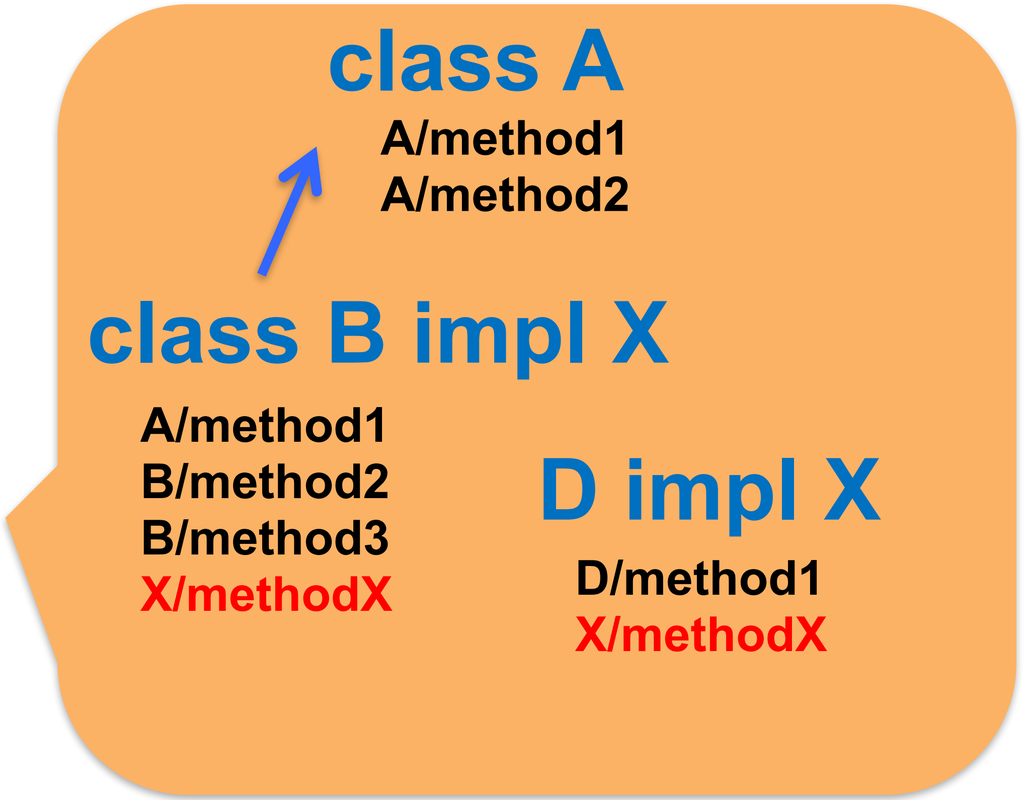
invokevirtual

- 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

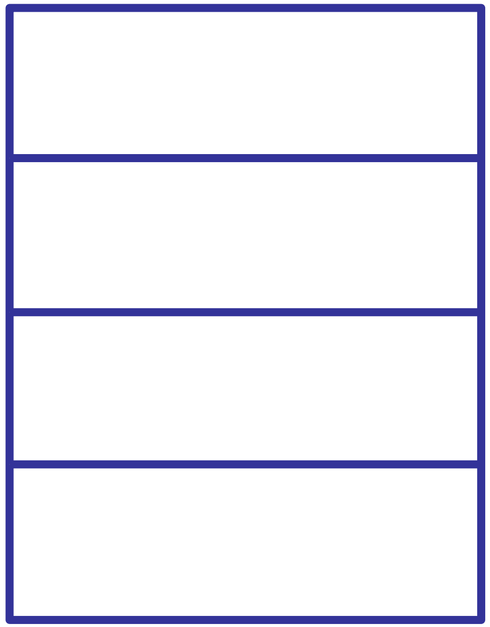
Вызов метода

Вызов метода

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

Вызов метода

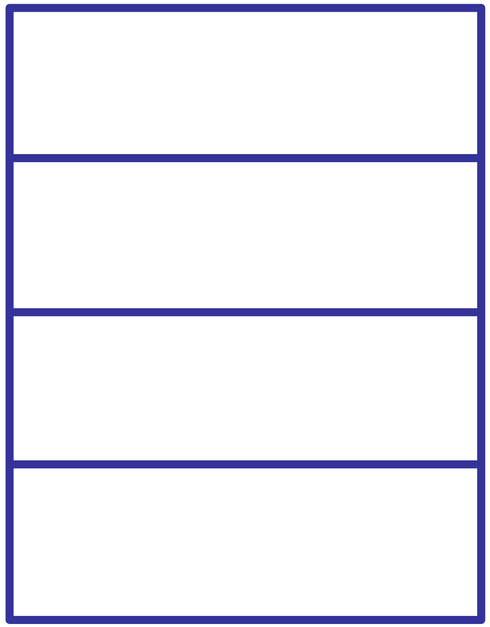
```
obj.method(param1, param2);
```



Вызов метода

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

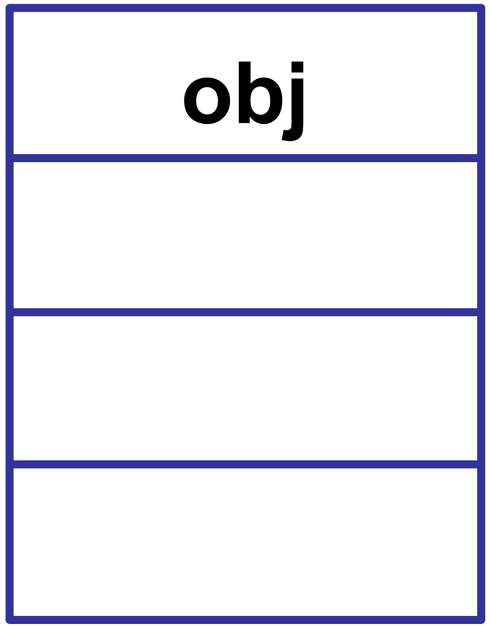
```
push obj
push param1
push param2
call method
```



Вызов метода

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

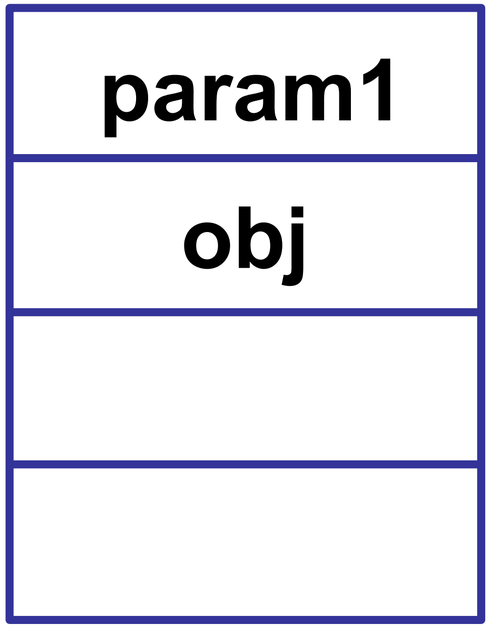
```
push obj
push param1
push param2
call method
```



Вызов метода

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

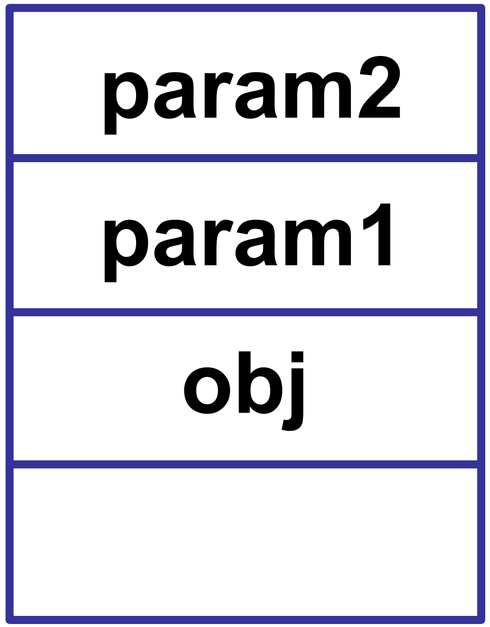
```
push obj
push param1
push param2
call method
```



Вызов метода

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

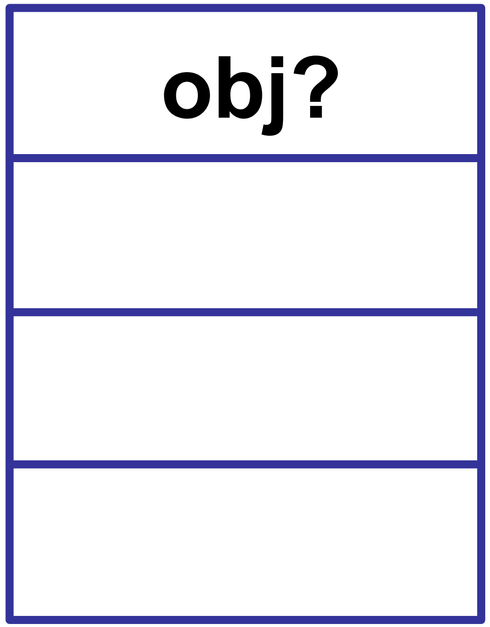
```
push obj
push param1
push param2
call method
```



Вызов метода

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

```
push obj
push param1
push param2
call method
```

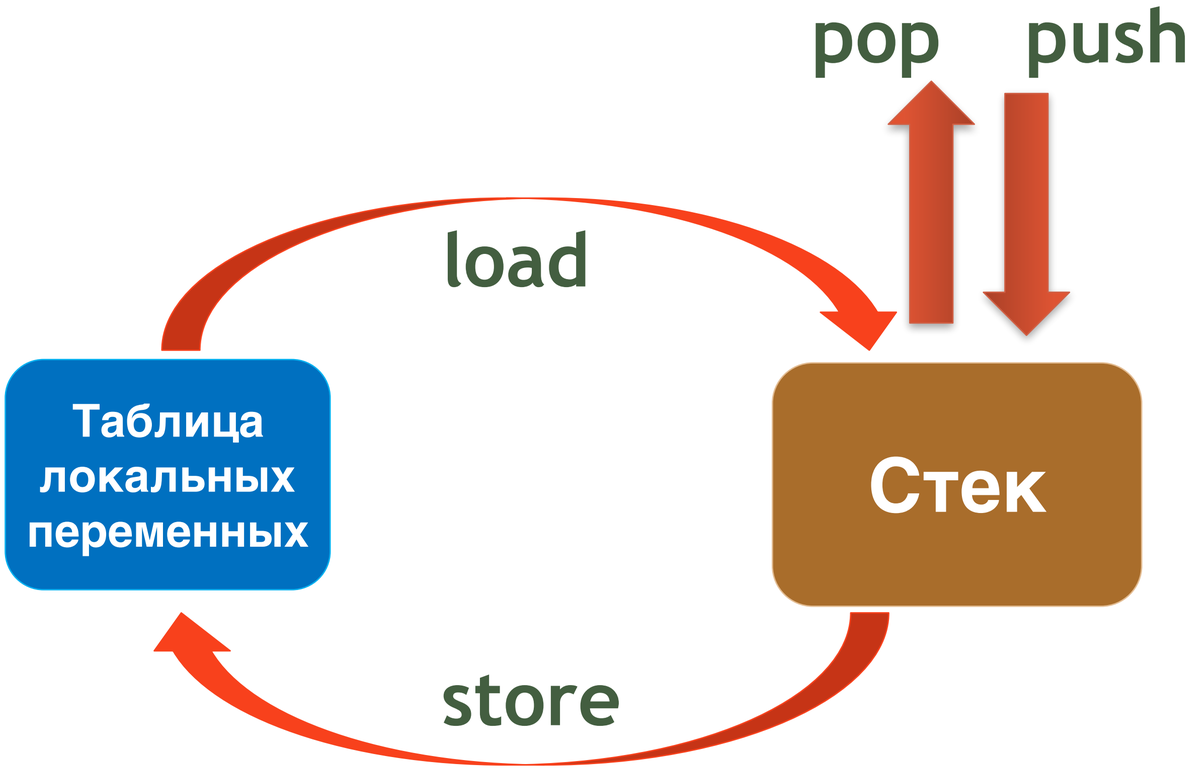


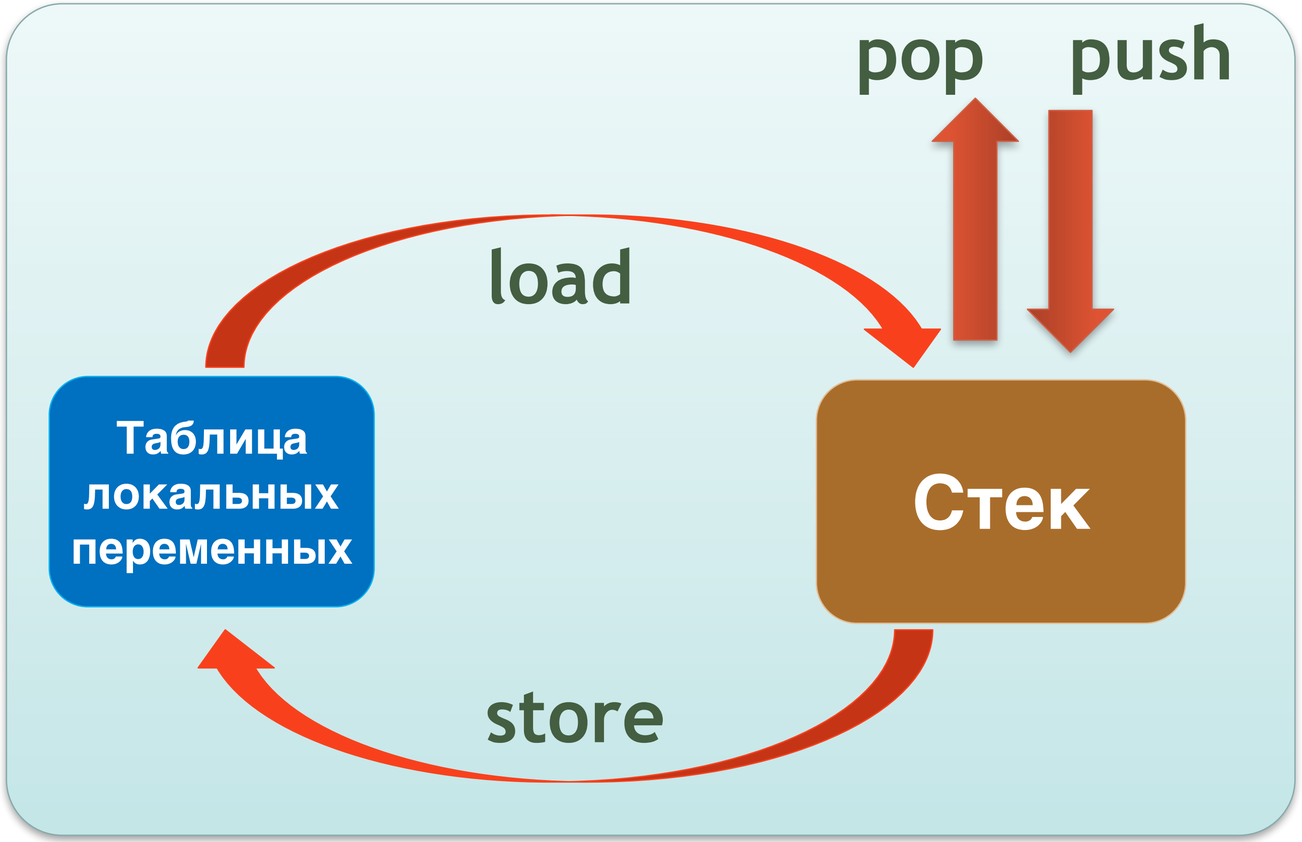
Вызов метода

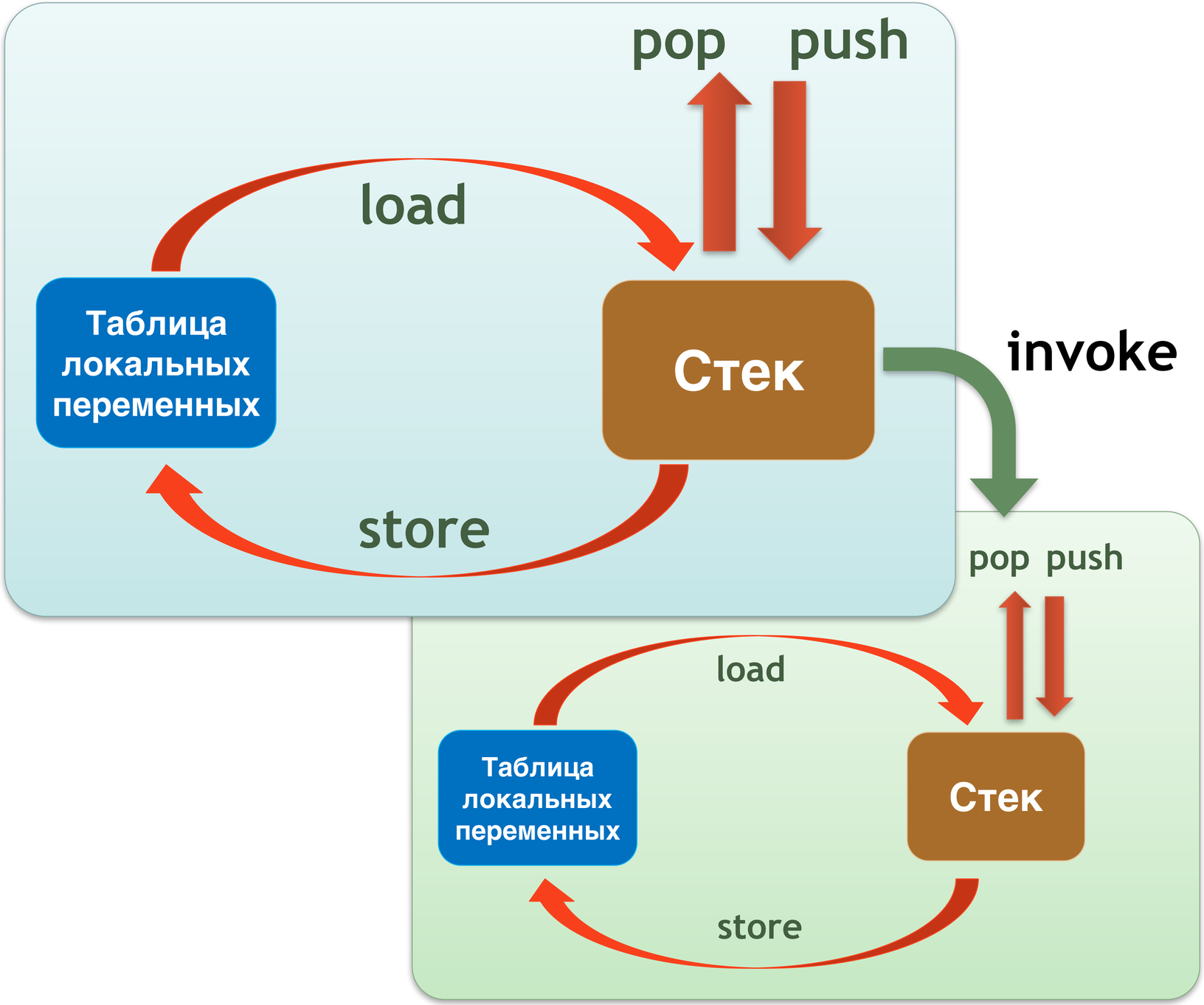
```
this.add(1, 2);
```

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


pop
push
Стек


pop
push
load
Таблица локальных переменных
Стек
store


pop
push
load
Таблица локальных переменных
Стек
store


pop
push
load
Таблица локальных переменных
Стек
invoke
store
pop
push
load
Таблица локальных переменных
Стек
store


>HELLO WORLD
>_

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

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

Дезассемблировать Hello

> javap -c 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

```java
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
```

```java
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

```java
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

```java
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

```java
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
```

```java
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
```

```java
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
```

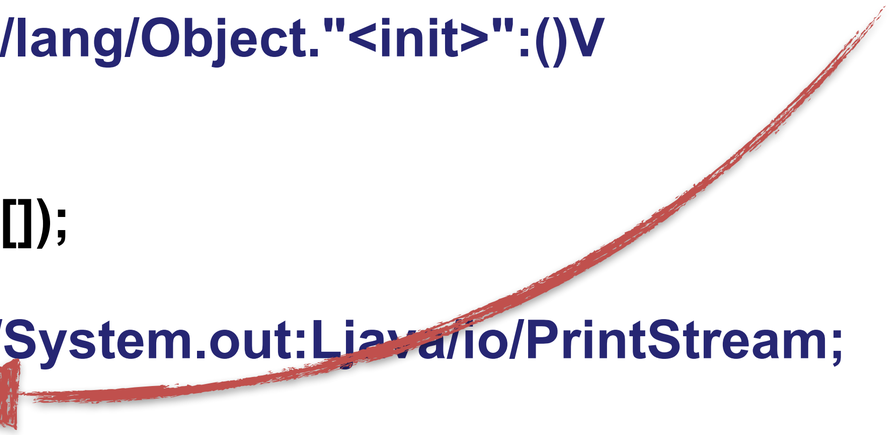
Обратиться к статическому полю

```java
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



```java
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

```java
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
```

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

Что такое #1,#2, итд ?

```
> 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
```

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

```
> javap -c -verbose Hello
```

```java
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>;
```

```java
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>;
```

```java
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
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;
```

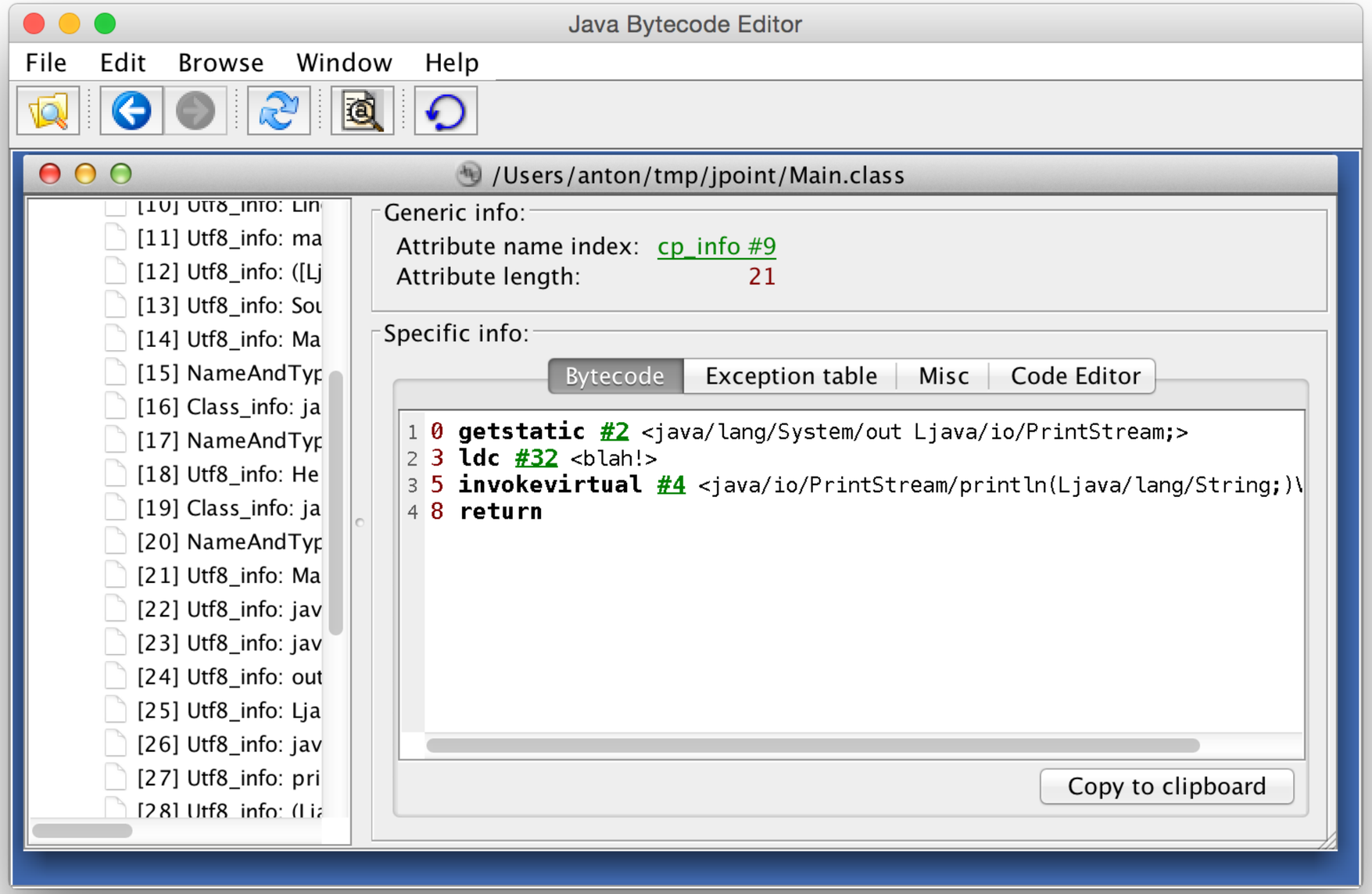
```java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

```
> 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;
```

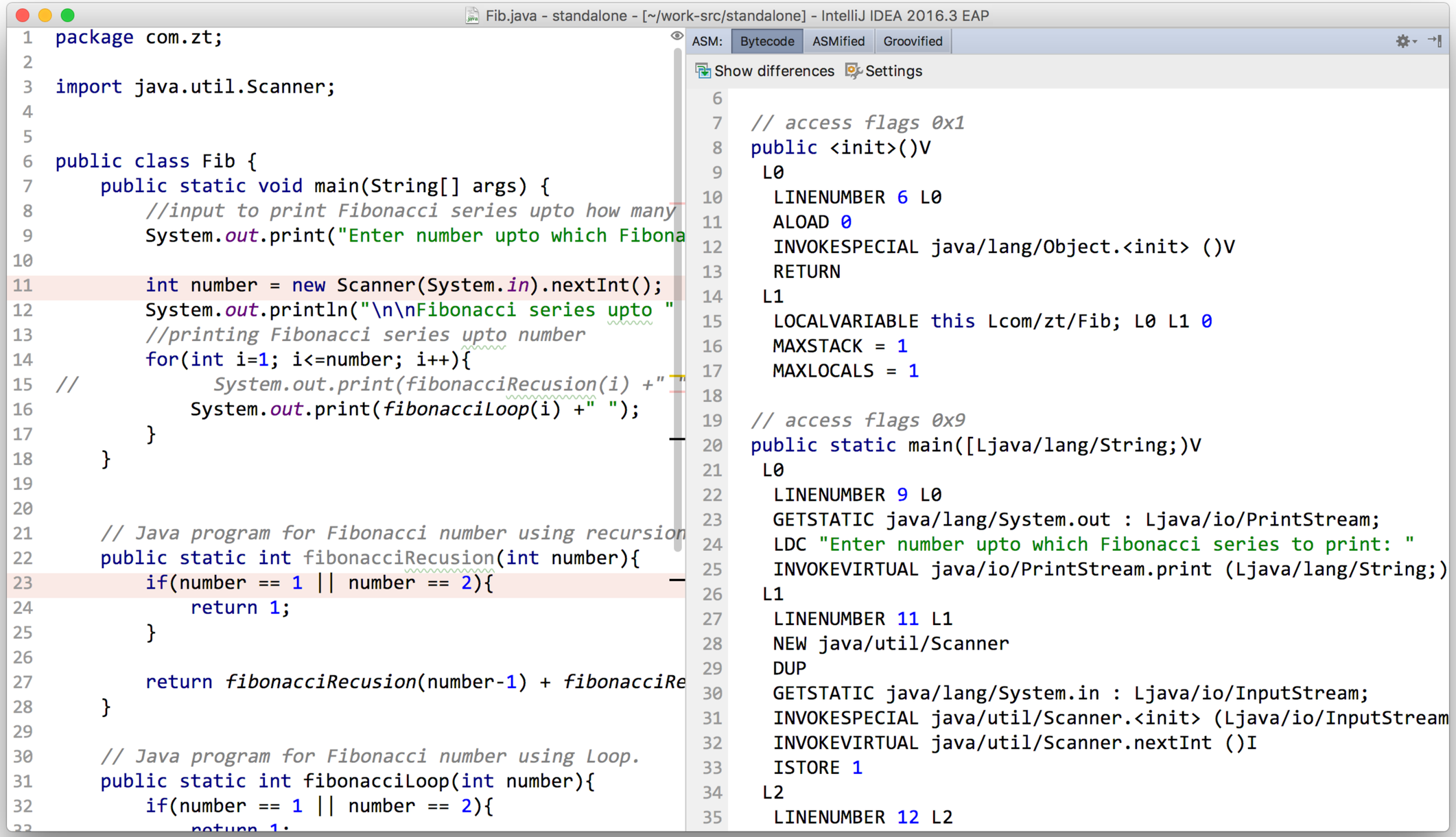
```java
public class Hello {
  public static void main(String[] args) {
    System.out.println("Hello, World!");
  }
}
```

```
> 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/



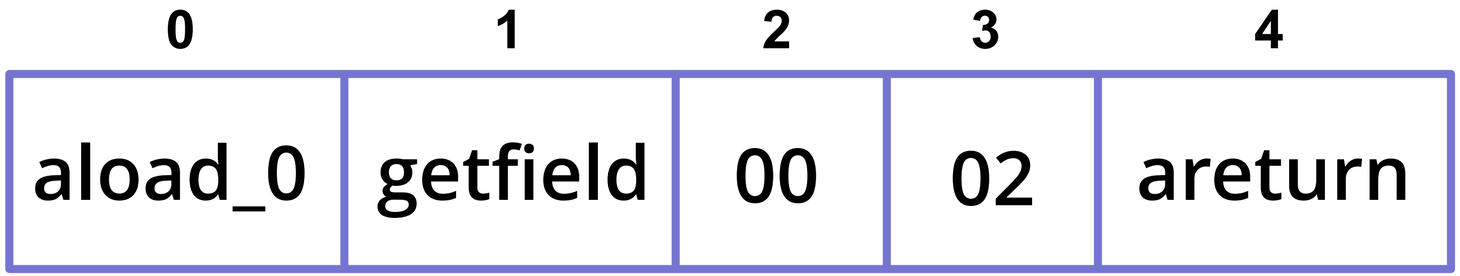
IntelliJ IDEA: ASM Bytecode Outline plugin



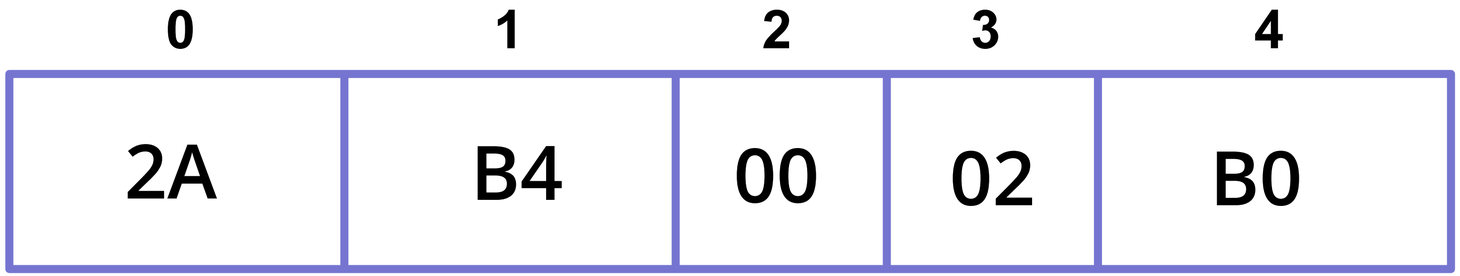
```
public class Get {
  String name;

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

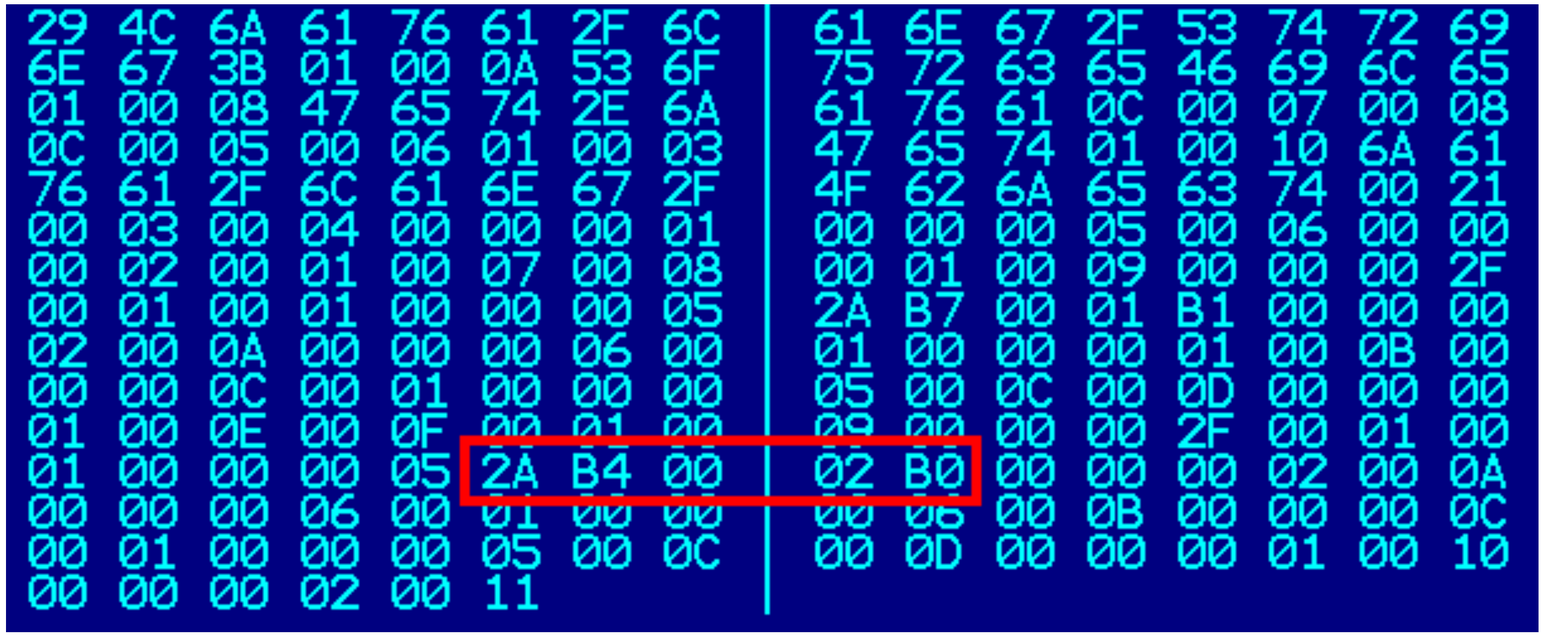
```
public java.lang.String getName();
  Code:
    Stack=1, Locals=1, Args_size=1
    0:   aload_0
    1:   getfield      #2; //Field name:Ljava/lang/String;
    4:   areturn
LocalVariableTable:
    Start  Length  Slot  Name   Signature
    0      5      0    this      LGet;
```




```
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;
```



public java.lang.String getName();

Code:

Stack=1, Locals=1, Args_size=1

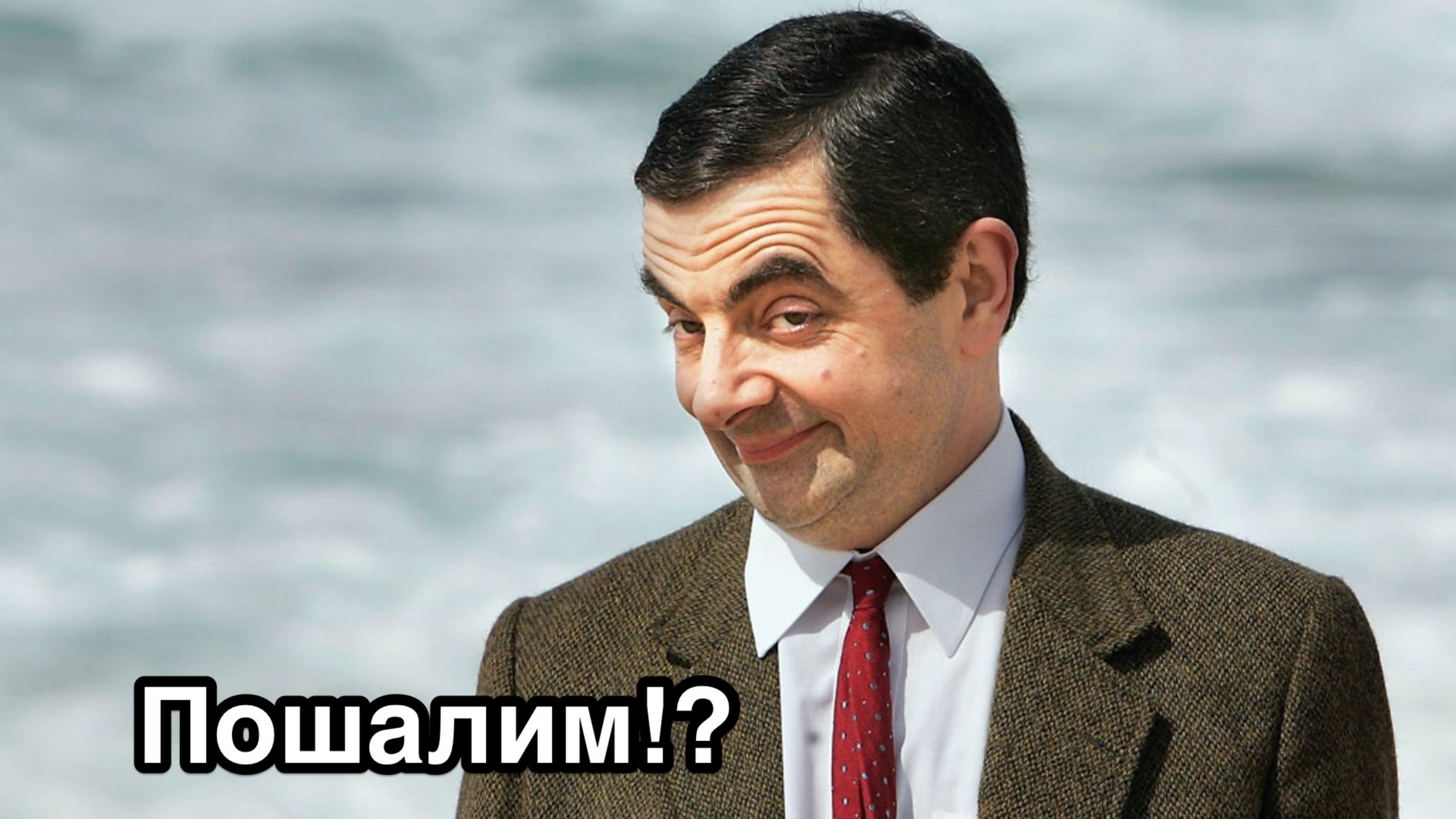
0: aload_0

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

4: areturn

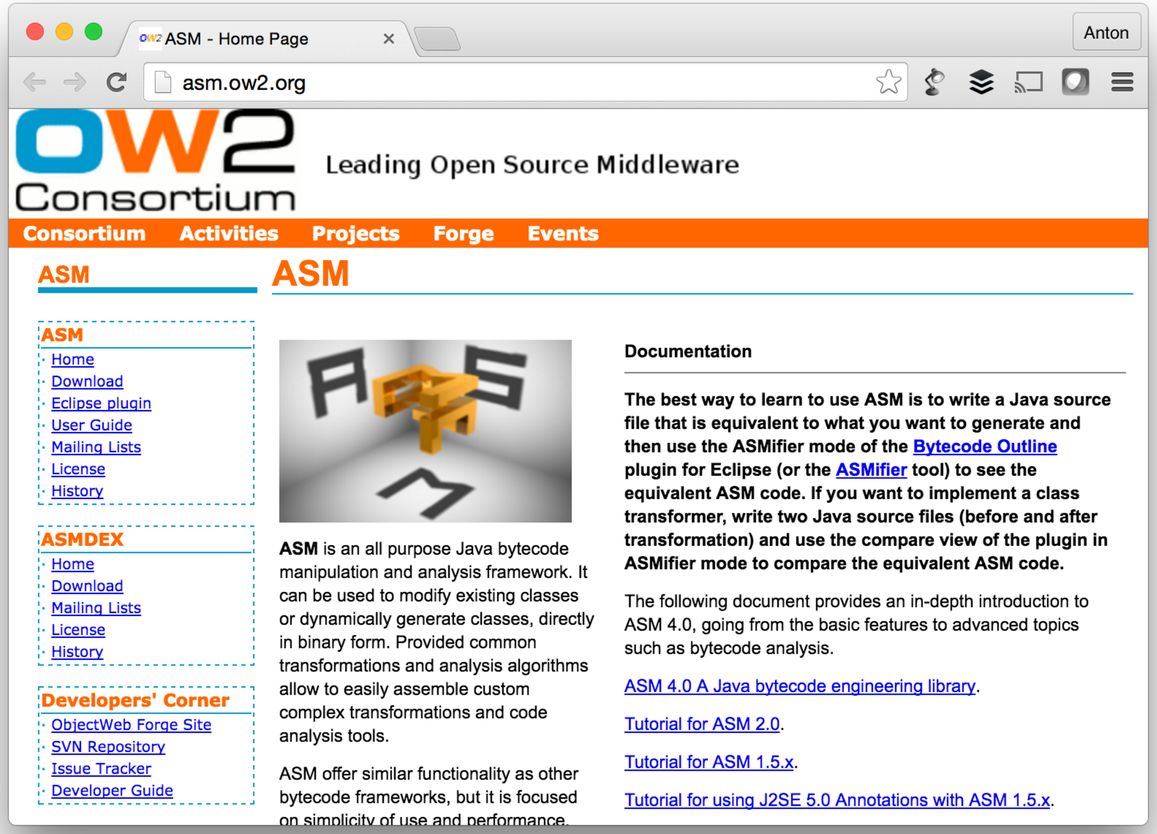
LocalVariableTable:

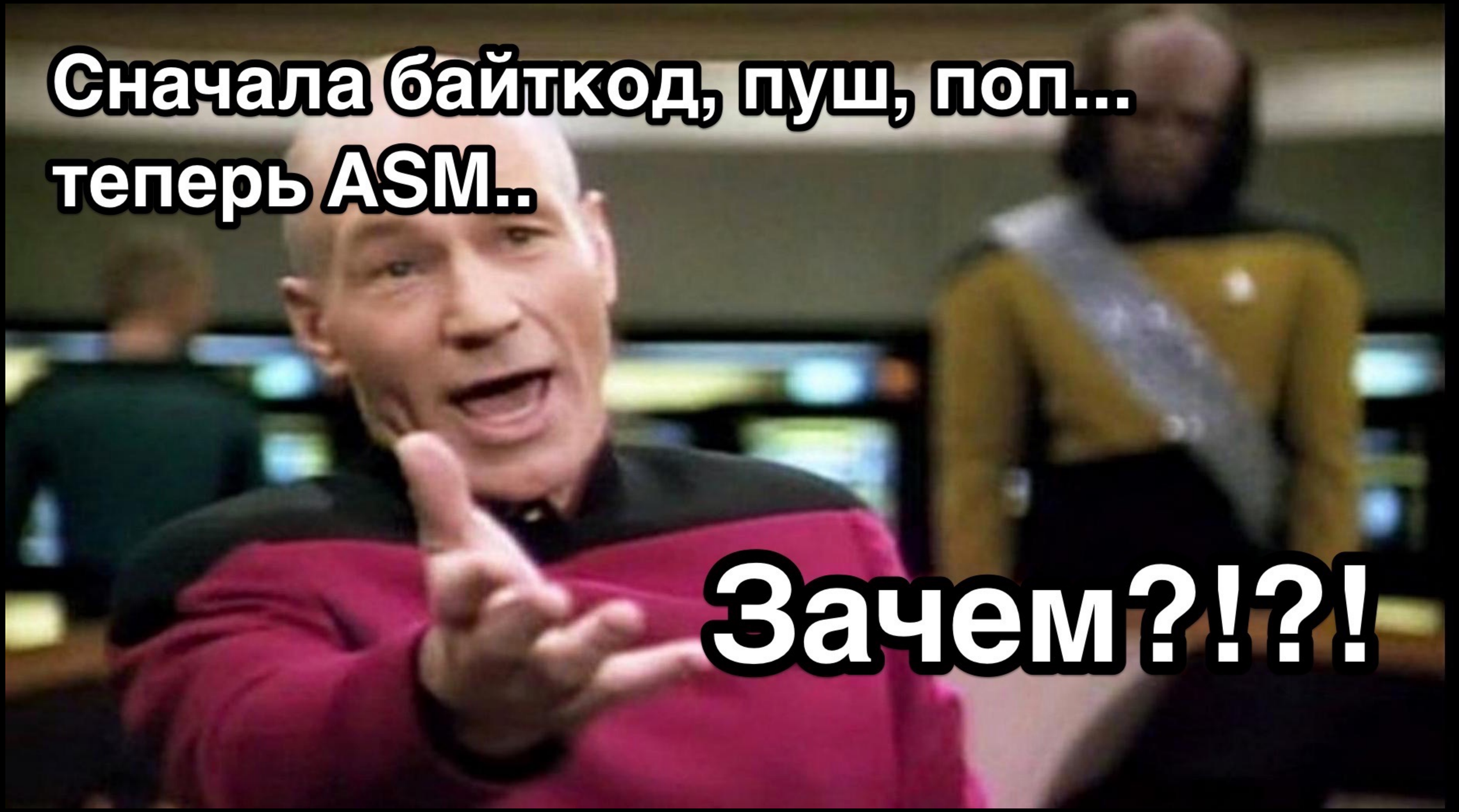
Start	Length	Slot	Name	Signature
0	5	0	this	LGet;


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)

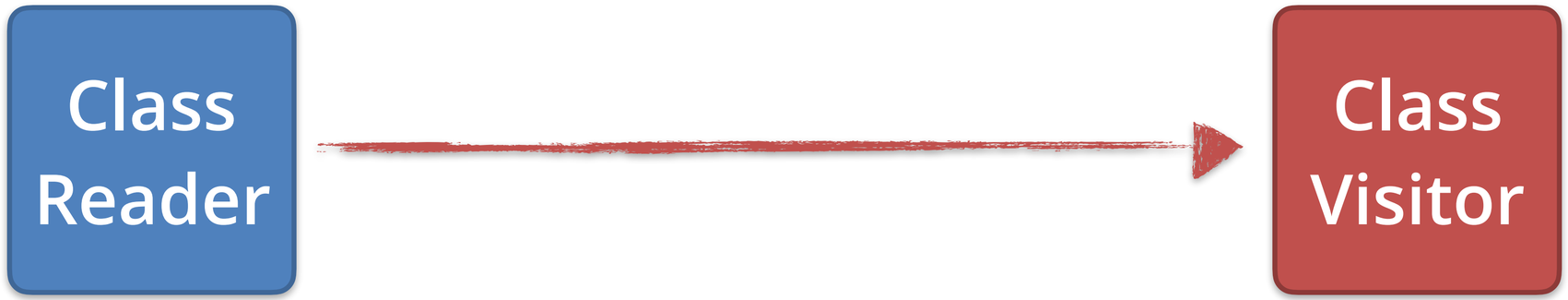
- Профилировщики
- Агенты для мониторинга
- Дебаггеры
- Фреймворки
- **JRebel**

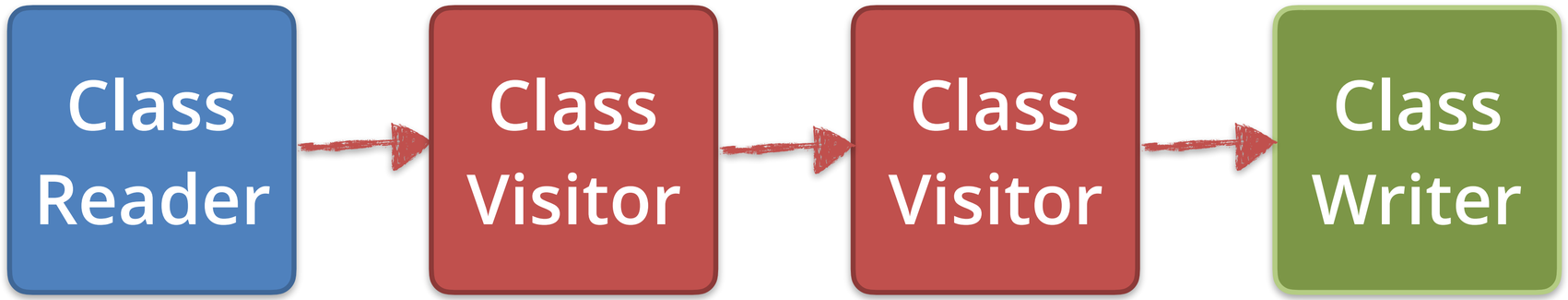
Требуется внедрение функциональности в работающий код

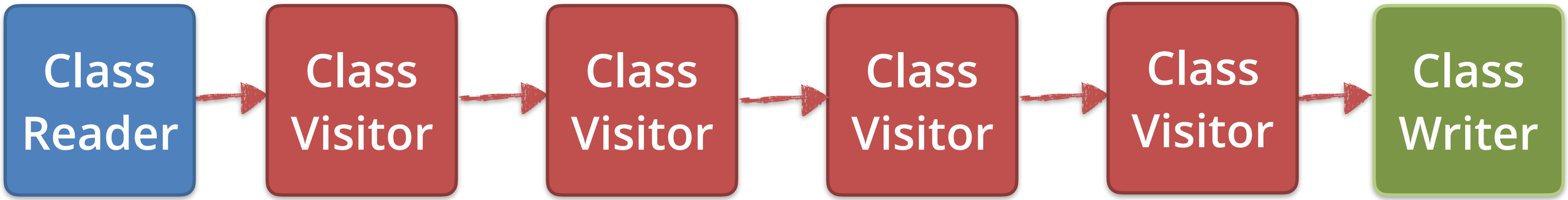
ASM: общий сценарий

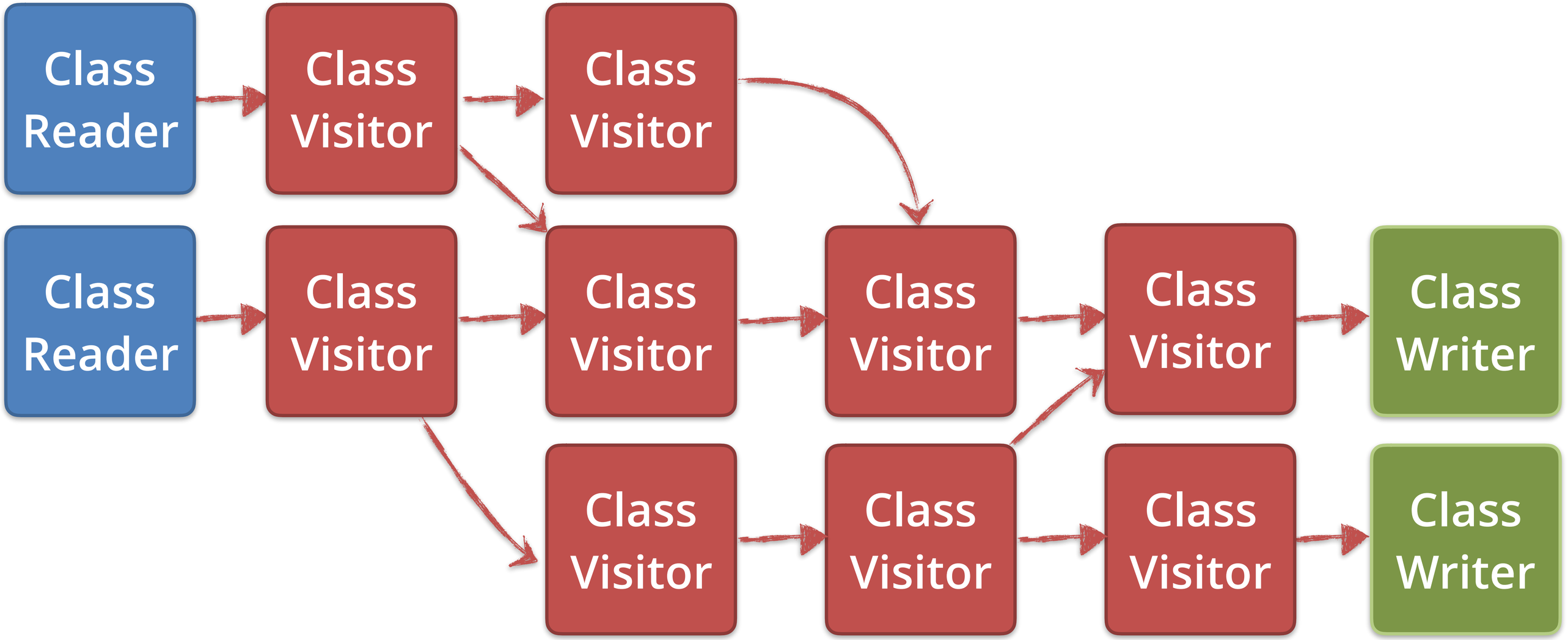
1. Создать ClassWriter
2. Скомпановать visitor-ы:

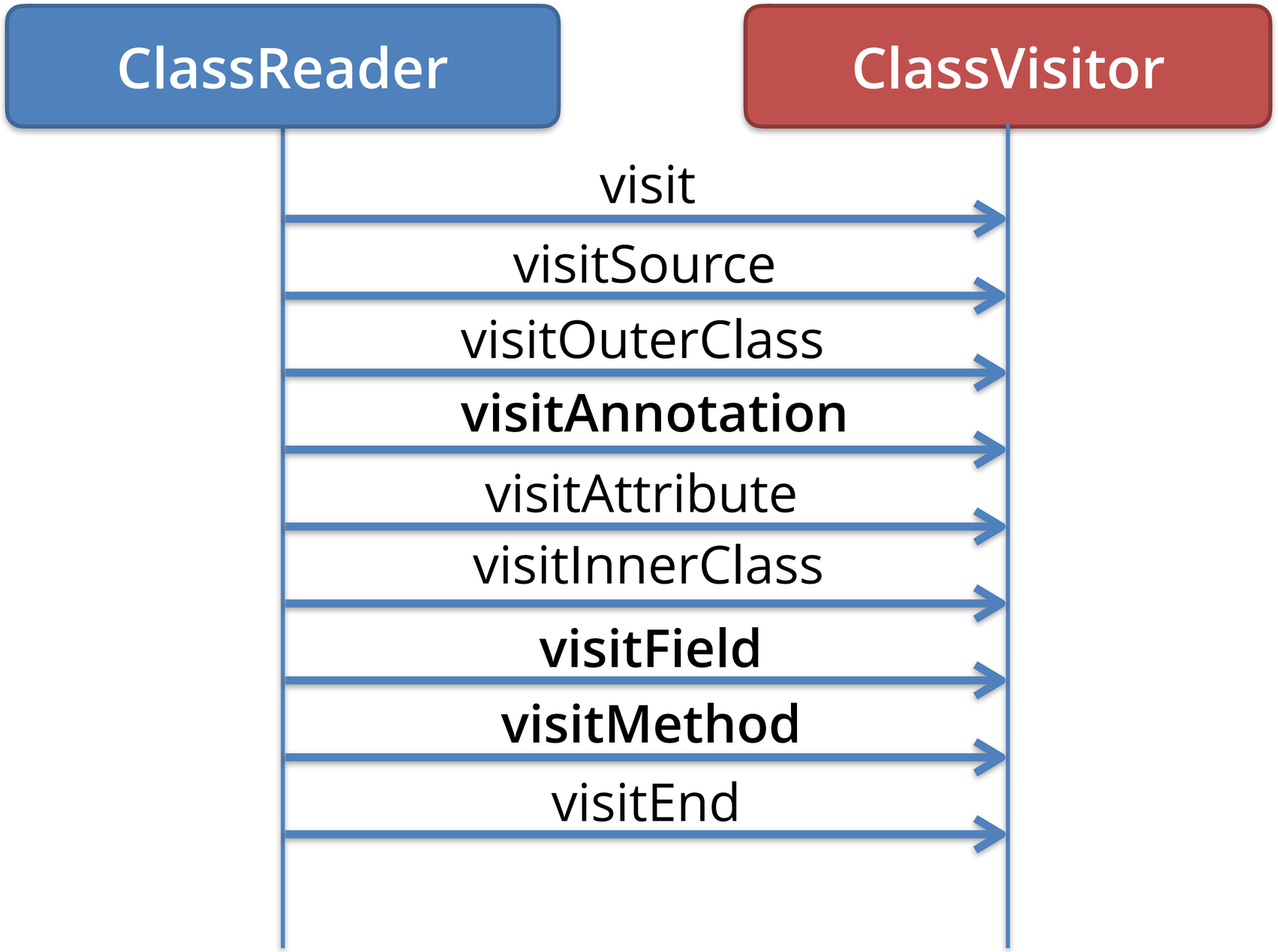
   аннотации, поля, методы, итд
3. Генерировать новый байткод

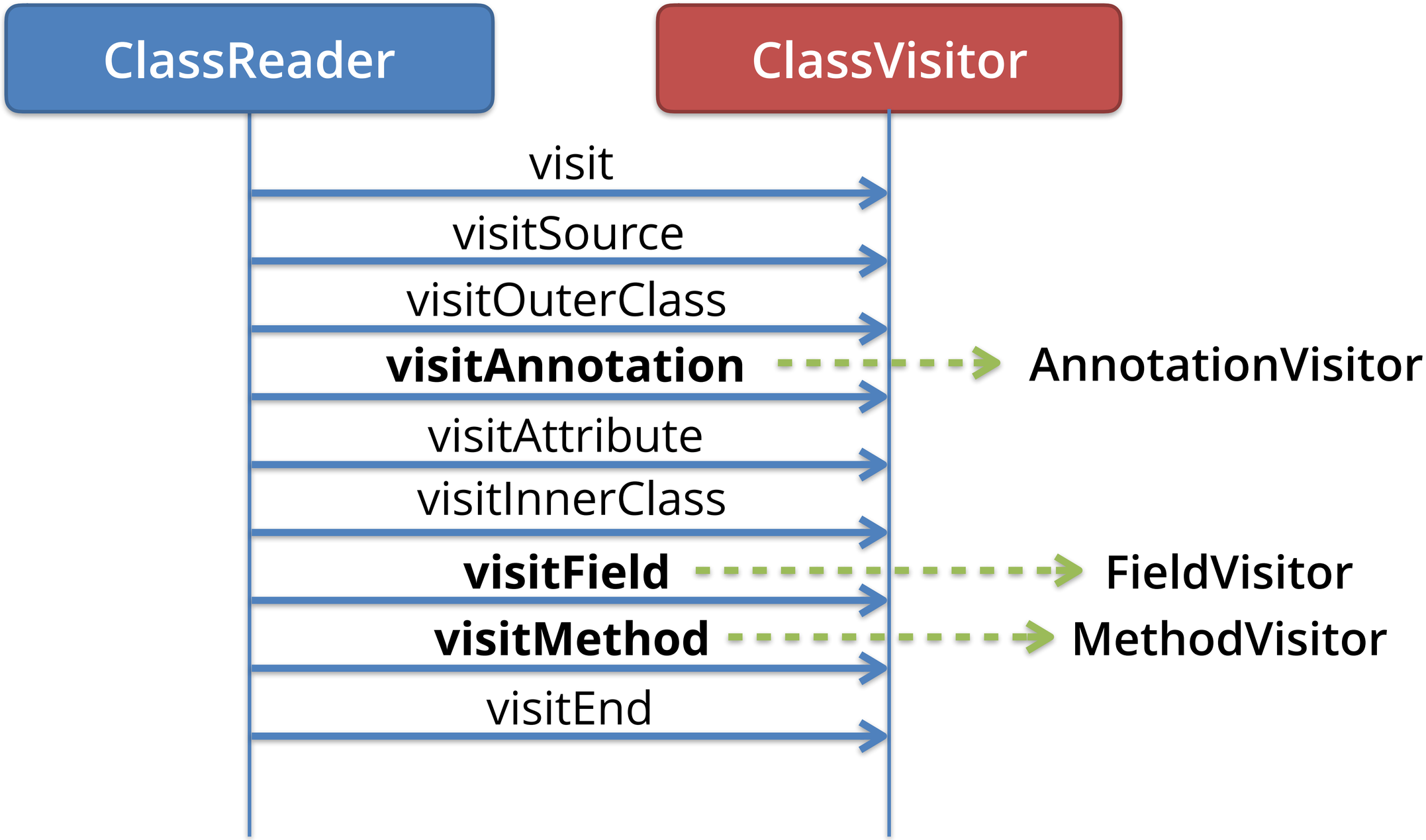

Class
Reader
Class
Visitor


Class Reader
Class Visitor
Class Visitor
Class Writer

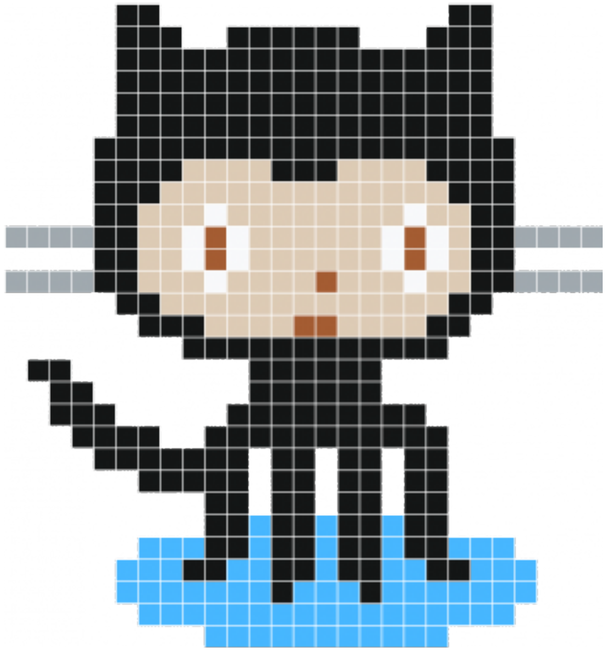

Class Reader
Class Visitor
Class Visitor
Class Visitor
Class Visitor
Class Writer


Class Reader
Class Visitor
Class Visitor
Class Reader
Class Visitor
Class Visitor
Class Visitor
Class Visitor
Class Writer
Class Visitor
Class Visitor
Class Visitor
Class Writer


ClassReader
ClassVisitor
visit
visitSource
visitOuterClass
visitAnnotation
visitAttribute
visitInnerClass
visitField
visitMethod
visitEnd


ClassReader
ClassVisitor
visit
visitSource
visitOuterClass
visitAnnotation
AnnotationVisitor
visitAttribute
visitInnerClass
visitField
FieldVisitor
visitMethod
MethodVisitor
visitEnd

Ты код
покажешь
когда нибудь, а?



https://github.com/antonarhipov/asmdemo


THANKS FOR LISTENING
ANY QUESTIONS
memegenerator.net