



Side Effect Injection

или добродетельные костыли

Владимир Плизга
toparvion@gmx.com

Кто этот чувак?

- Владимир Плизгá
- Backend-разработчик
 - **Позиция:** главный инженер-программист, ЦФТ (Центр Финансовых Технологий)
 - **Опыт** в области: 6 лет
 - **Область:** разработка Интернет-банков для держателей prepaid карт



<https://oplata.kykyryza.ru>



- 20+ партнёров по Интернет-банкам
- 20+ партнёров по сайтам подарочных карт

О чем пойдет речь?

- Удобство
- Безопасность
- Интеграции
- ...

Могут требовать добавления
особых поведений
в тестовом окружении

Как это правильно сделать?

Вместо плана

1

Введение в Side Effect Injection (SEI)

2

Реализация SEI в jMint

3

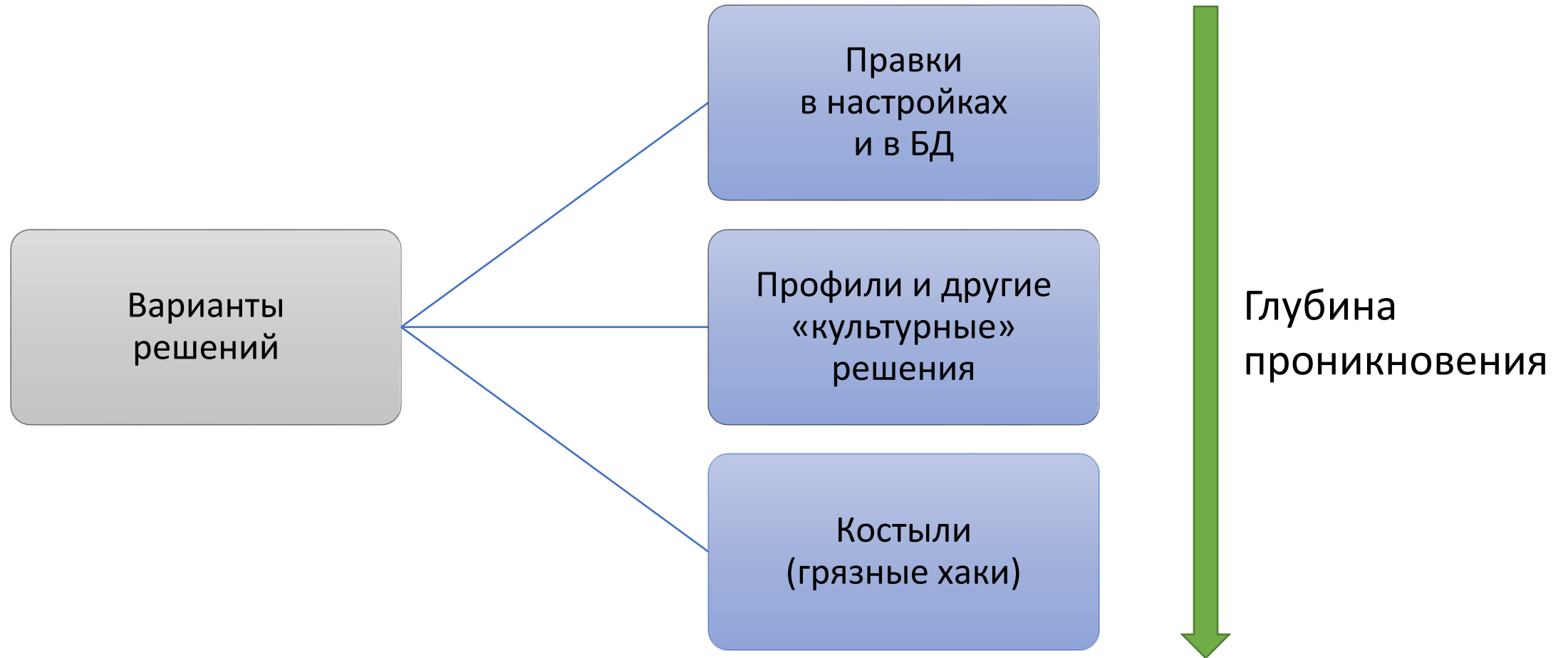
Практический пример

Введение в Side Effect Injection

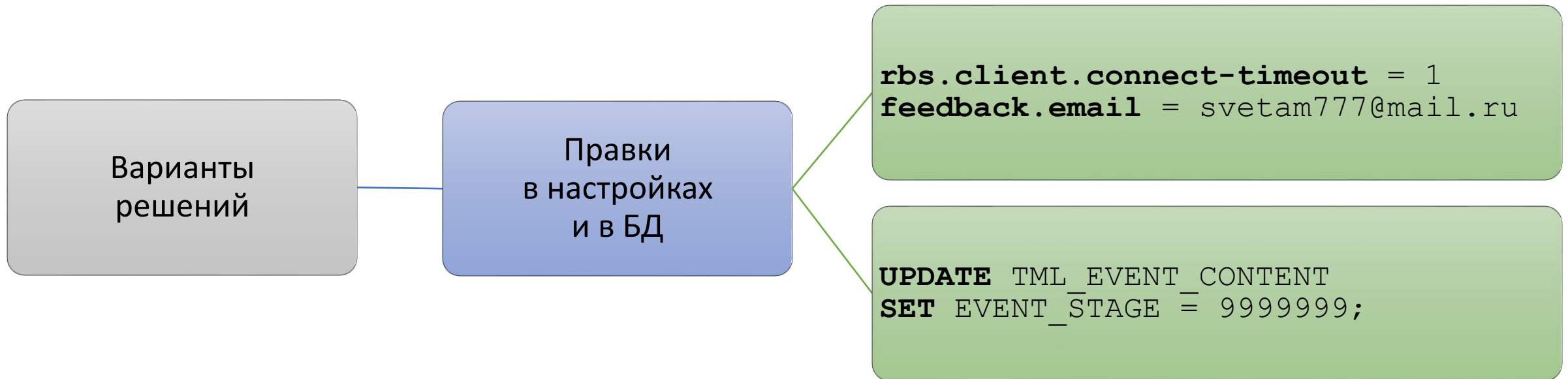
Несвязанные задачи. Почти.



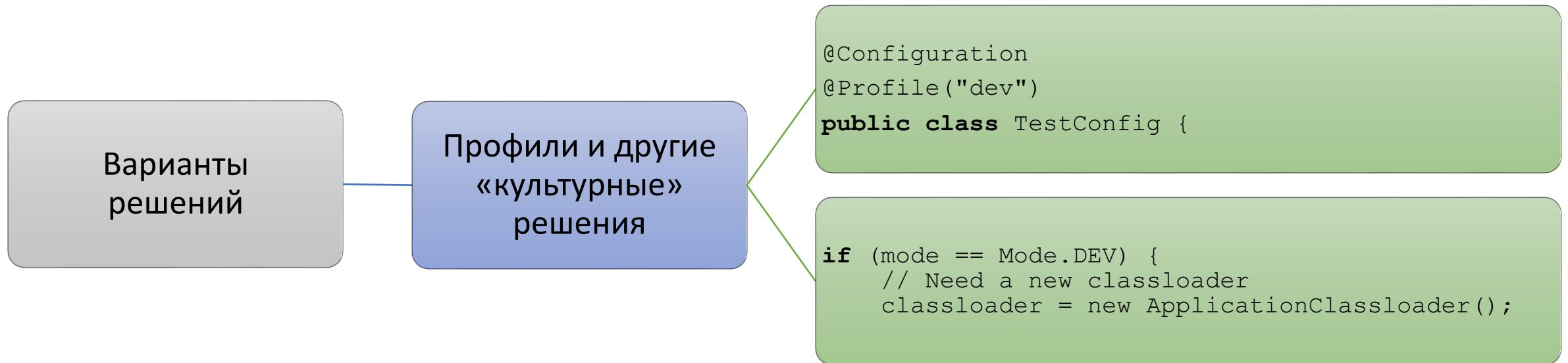
Возможные решения (в общем)



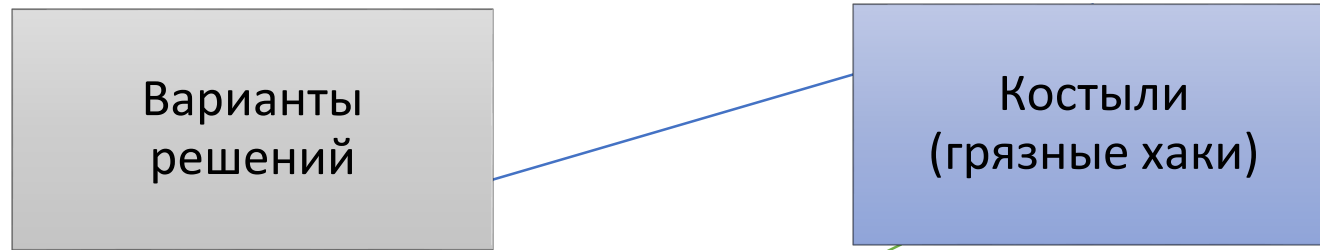
Решение: правки в БД и настройках



Решение: средства фреймворков



Решение: во все тяжкие



```
if (!Boolean.valueOf(System.getenv("GCARD_MODULE_NOSMS"))) {  
    instance.signAndSend(doc);  
} else {  
    log.warn("SMS sending is disabled with environment variable GCARD_MODULE_NOSMS=true.");  
}
```

При чем тут SEI?



Работа с байт-КОДОМ «как есть»

```
package ru.ftc.upc.testing.helloworld;

public class Main {

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

```
// class version 52.0 (52)
// access flags 0x21
public class ru/ftc/upc/testing/helloworld/Main {

    // compiled from: Main.java

    // access flags 0x1
    public <init>()V
    L0
        LINENUMBER 3 L0
        ALOAD 0
        INVOKESPECIAL java/lang/Object.<init> ()V
        RETURN
    L1
        LOCALVARIABLE this Lru/ftc/upc/testing/helloworld/Main; L0 L1 0
        MAXSTACK = 1
        MAXLOCALS = 1

    // access flags 0x9
    public static main([Ljava/lang/String;)V
    L0
        LINENUMBER 6 L0
        GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
        LDC "Hello, World!"
        INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
    L1
        LINENUMBER 7 L1
        RETURN
    L2
        LOCALVARIABLE args [Ljava/lang/String; L0 L2 0
        MAXSTACK = 2
        MAXLOCALS = 1
}
```

Так себе вариант...



Можно всех посмотреть



1. Цель: «Clean modularization of crosscutting concerns»
2. Лишняя зависимость в проекте
(если её не было!)



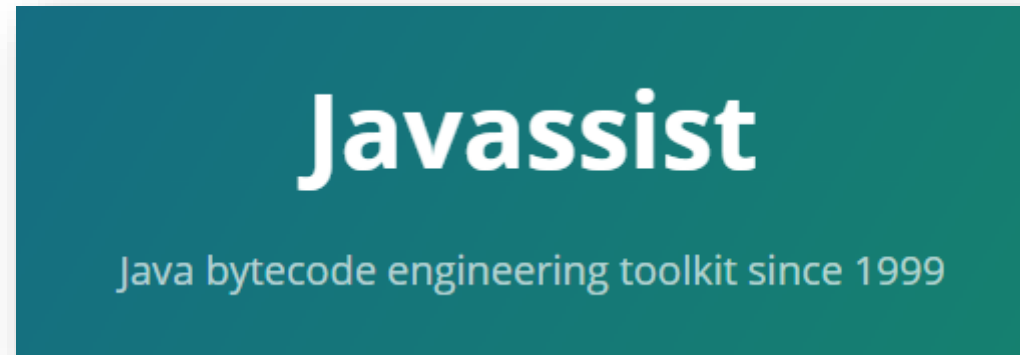
1. Цель: «Simplify Java tracing, monitoring and testing»
2. Нужно изучать и как-то писать на специальном скриптовом языке



Shigeru Chiba's
GluonJ

Сырец,
но...

Основа нашего решения



Используется в



HIBERNATE

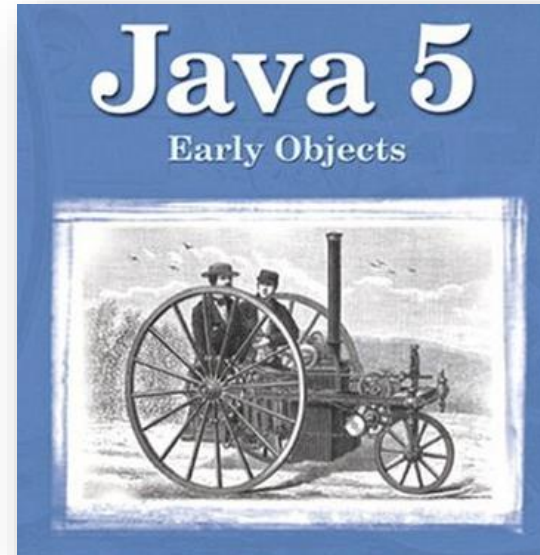


и много где ещё.

Может генерировать байт-код
из фрагментов Java кода
с ограничениями.

Путь внедрения

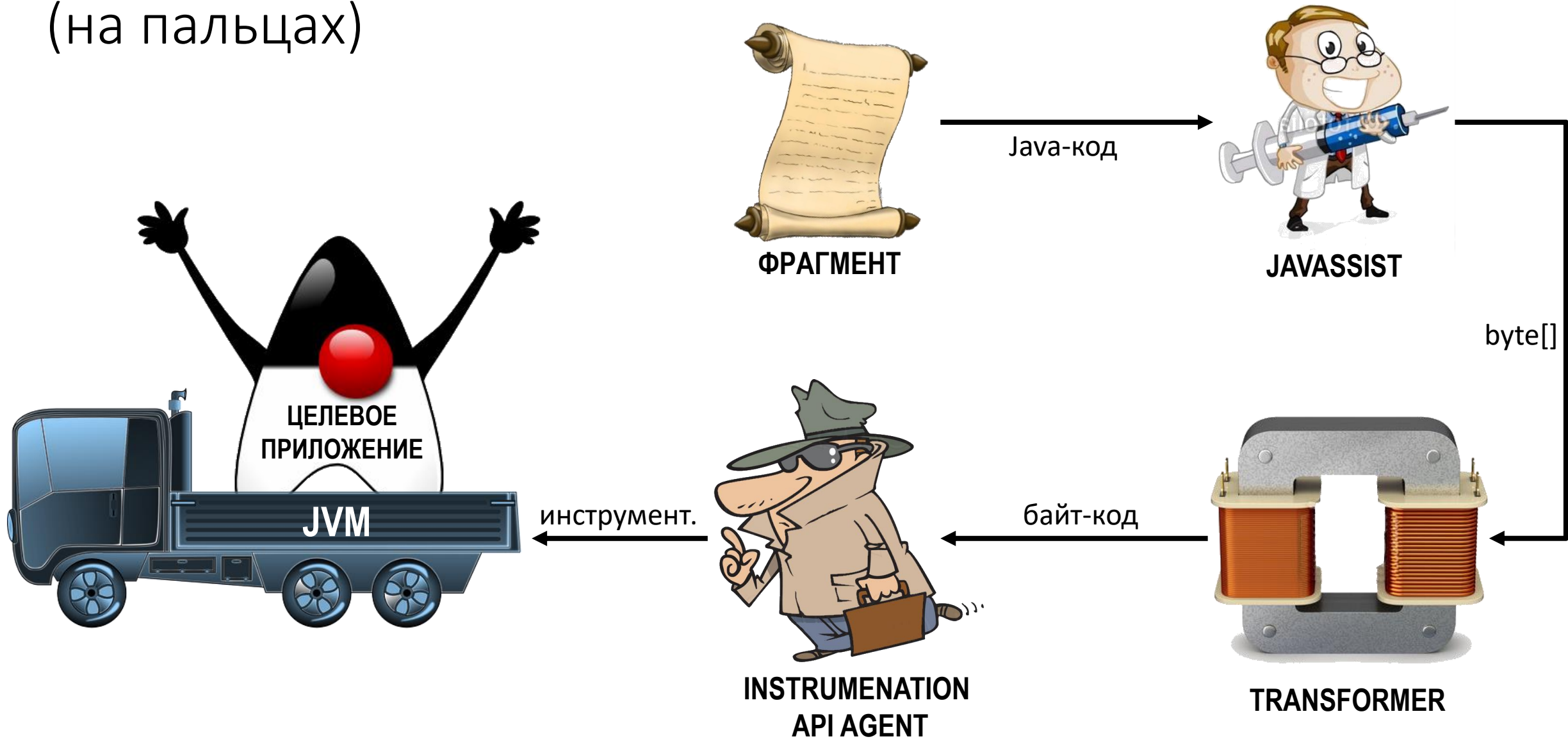
- Для внедрения в процесс загрузки классов нужен отдельный инструмент



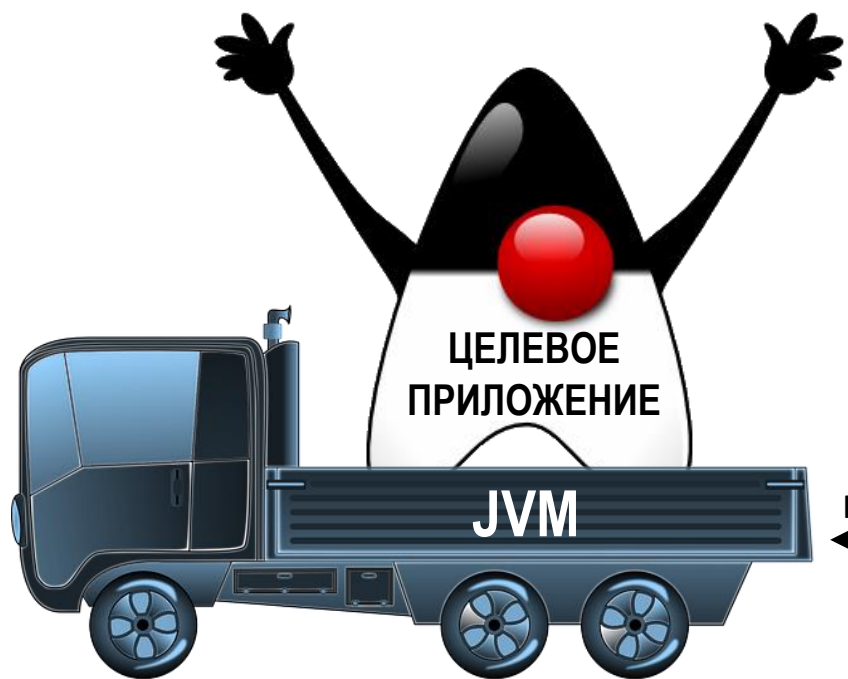
JAVA INSTRUMENTATION API

Реализация SEI

Базовая идея (на пальцах)



Базовая идея (на пальцах)



DROPLETs
(от англ. droplet – капелька, вкрапление)
ФРАГМЕНТЫ

Java-код

JAVASSIST
byte[]

jMint
(от англ. mint – мята)
INSTRUMENTATION API AGENT
TRANSFORMER

инструмент.

байт-код

Что такое jMint?

- Java-агент (приложение, подключаемое к JVM).
Запуск в изоляции от целевой JVM невозможен;
- Поставляется в виде единственного JAR;
- Подключается к целевому приложению при старте с помощью опции JVM `-javaagent` в скрипте запуска:



```
JAVA_OPTS="$JAVA_OPTS -javaagent:jmint.jar=SmsSender.droplet.java"
```

Что такое дроплет?

SmsSenderImpl.java

```
@Override
public void sendMessage(String phone, String message) {
    UpcJob job = new UpcJob<Void>(SimmpJobs.SIMMP_SEND_SMS) {
        ...
    }
}
```



```
//droplet class: sms.SMSSenderImpl
//droplet method: sendMessage
//droplet cutpoint: BEFORE
//droplet text:
log.warn("Текст отправляемой SMS: '{}'", $2);
```



```
@Override
public void sendMessage(String phone, String message) {
    log.warn("Текст отправляемой SMS: '{}'", message);
    UpcJob job = new UpcJob<Void>(SimmpJobs.SIMMP_SEND_SMS) {
        ...
    }
}
```

Пример: «тупая» заглушка

```
// Точка вкрапления (одна из: BEFORE, INSTEAD, AFTER)  
//droplet cutpoint: INSTEAD  
  
// Инструментируемый метод  
//droplet method: isFinite  
  
// Инструментируемый класс (FQ-name)  
//droplet class: models.TransferStatus  
  
// Текст вкрапления (адаптированный Java-код)  
//droplet text:  
return false;
```

Пример: «умная» заглушка

```
//droplet class: dp.DPClientImpl
//droplet method: fetchTransferStatus
//droplet cutpoint: INSTEAD
//droplet text:
{
  String statusStr = null;
  try {
    java.util.Properties stub = new java.util.Properties();
    stub.load(new java.io.FileReader(System.getProperty("user.dir") + java.io.File.separator + "dp-edit-mock.properties"));
    statusStr = stub.getProperty($1);
    if (statusStr != null) log.warn("For oid={} transfer status {} was taken from mock.", $1, statusStr);
  } catch (java.io.IOException e) {
    log.error("Failed to load mocked transfer edit statuses.", e);
  }
  if (statusStr == null) {
    dp.models.QuickPay quickPay = dp.models.QuickPay.infoService();
    dp.models.ReqTransferSearch req = new dp.models.ReqTransferSearch();
    req.setOID($1);
    quickPay.getInfoService().setReqTransferSearch(req);
    quickPay = sendQuickPay(quickPay, false, false);
    statusStr = quickPay.getInfoService().getAnsTransferSearch().getTransferStatus();
  }
  return statusStr;
}
```

Недостатки реализации

- Имена почти всех классов – fully qualified
- Вместо имен формальных параметров – \$1, \$2, ...
- Не поддерживаются перегруженные методы
- Нужно помнить формат написания
- Не годится для работы в IDE

Идея для улучшения

```
package sms;
```

```
import sms.SimmpJobs;  
import job.UpcJob;  
import java.util.Properties;
```

```
public class SMSSenderImpl implements SMSSender {
```

```
    @Override
```

```
    public void sendMessage(String phone, String message) {
```

```
        UpcJob job = new UpcJob<Void>(SimmpJobs.SIMMP_SEND_SMS) {
```

```
            ...
```

```
        }
```

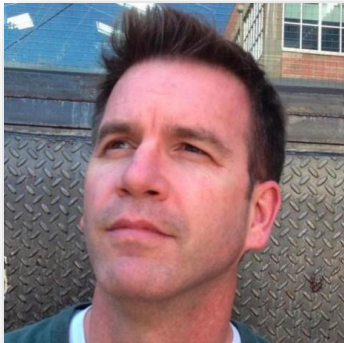
```
    }
```

Формулировка идеи

- Особый формат не нужен
- Пусть выглядят как обычный Java код, а тела методов станут модифицирующими фрагментами
- Профит: простота и поддержка в IDE (на уровне синтаксиса, но не семантики)



Инструмент для разбора



Terence Parr

Samples

```
grammar Expr;
prog:  (expr NEWLINE)* ;
expr:  expr ('*' | '/')
expr
|     expr ('+' | '-')
expr
|     INT
|     '(' expr ')'
;
NEWLINE : [\r\n]+ ;
INT      : [0-9]+ ;
```

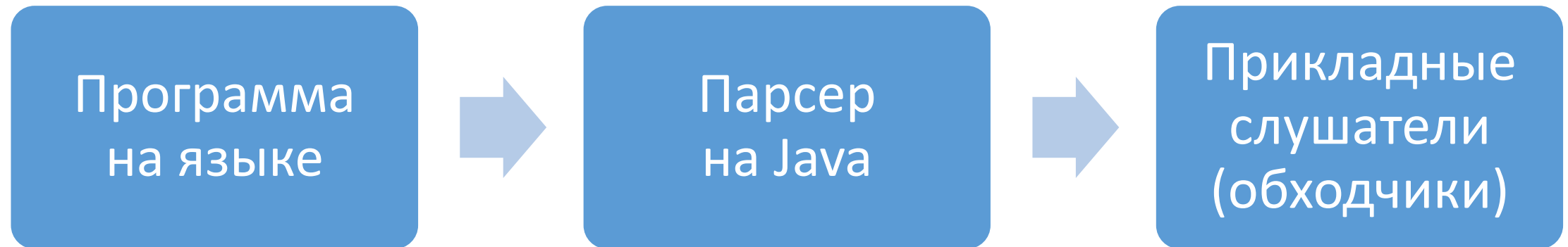
```
$ antlr4 Expr.g4
$ javac Expr*.java
$ grun Expr prog -gui
100+2*34
^D
```

The image shows a screenshot of the ANTLR GUI. On the left, there is a terminal window showing the command sequence: `$ antlr4 Expr.g4`, `$ javac Expr*.java`, and `$ grun Expr prog -gui`. The output shows the expression `100+2*34` and a cursor at the end. On the right, there is a window displaying a parse tree for the expression. The root node is `prog`, which has a child `expr` and a terminal `\n`. The `expr` node has three children: `expr`, `+`, and `expr`. The left `expr` child has a terminal `100`. The middle `expr` child has three children: `expr`, `*`, and `expr`. The left `expr` child of this node has a terminal `2`, and the right `expr` child has a terminal `34`. Below the tree is a slider and an `OK` button.

Как работает ANTLR: подготовка



Как работает ANTLR: применение



Формальная грамматика Java

- ≈ 430 правил
- Максимальное соответствие JLS
- $\approx 13\text{K}$ LoC в сгенерированном парсере
- Редко меняется
- Нужно только ≈ 290 правил



**Ни к чему тащить
всё это в jMint**

Хирургия грамматики (часть 1)

<pre>* Productions from \$14 (Blocks and Statements) */ block : '{' blockStatements? '}' ; blockStatements : blockStatement+ ; blockStatement : localVariableDeclarationStatement classDeclaration statement ;</pre>	<pre>713 638 714 639 715 640 716 641 >> 717 642 << 718 643 719 644 720 645 >> 721 646 << 722 647 723 648 724 649 725 650 726 651 727 652 728 653</pre>	<pre>* Productions from \$14 (Blocks and Statements) */ block : '{' blockStatements '}' ; blockStatements : .*? ;</pre>
--	--	---

Упрощён разбор блочных выражений

Хирургия грамматики (часть 2)

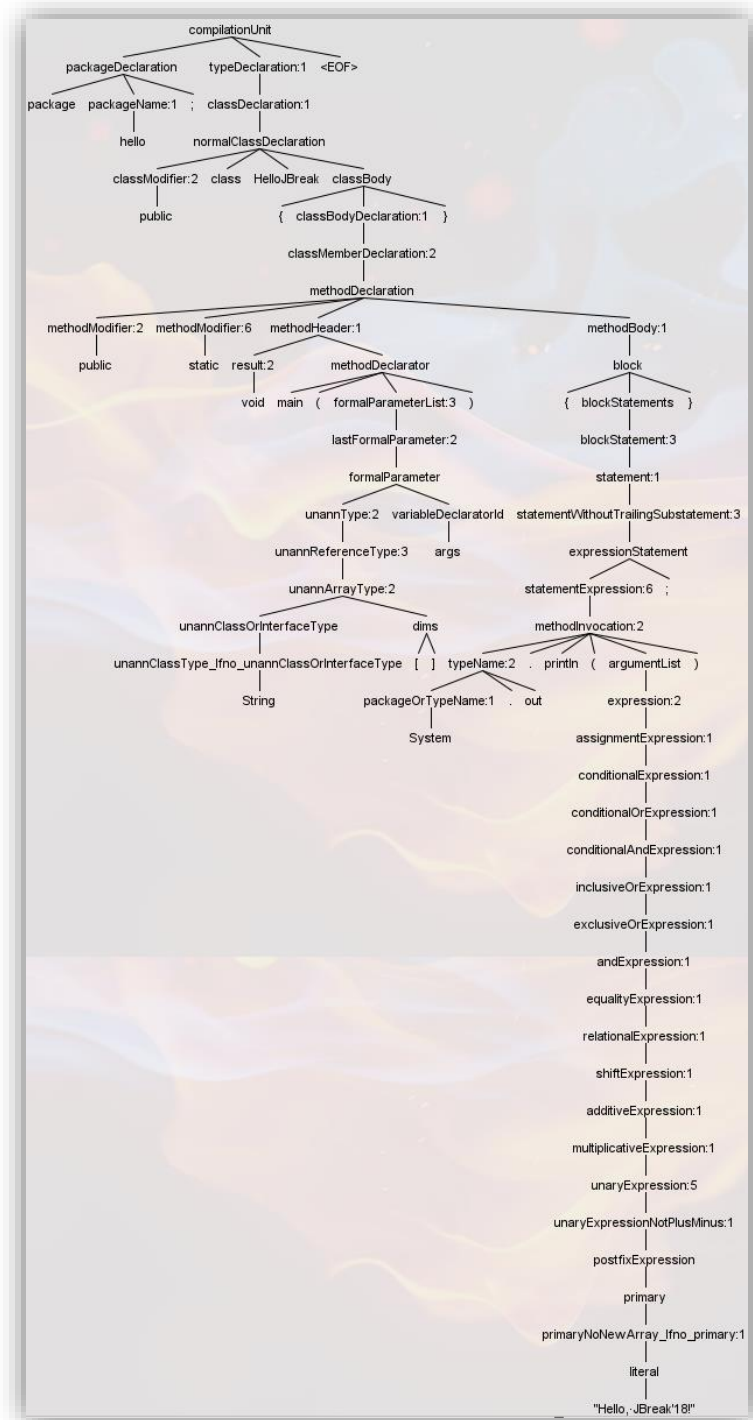
<code>constructorBody</code>	532	400	
<code>: '{' explicitConstructorInvocation? blockStatements? '}'</code>	» 533	401	
<code>;</code>	534	402	
<code>explicitConstructorInvocation</code>	535		
<code>: typeArguments? 'this' '(' argumentList? ')' ';' </code>	536	467	
<code>typeArguments? 'super' '(' argumentList? ')' ';' </code>	537	468	
<code>expressionName '.' typeArguments? 'super' '(' argumentList? ')' ';' </code>	538	469	
<code>primary '.' typeArguments? 'super' '(' argumentList? ')' ';' ;</code>	539	470	<code>constructorBody</code>
	540	471 <<	<code>: '{' blockStatements '}'</code>
	541	472	<code>;</code>

Устранён разбор родительских вызовов в конструкторах

Сопоставление результатов

Дерево разбора на **базовой** грамматике

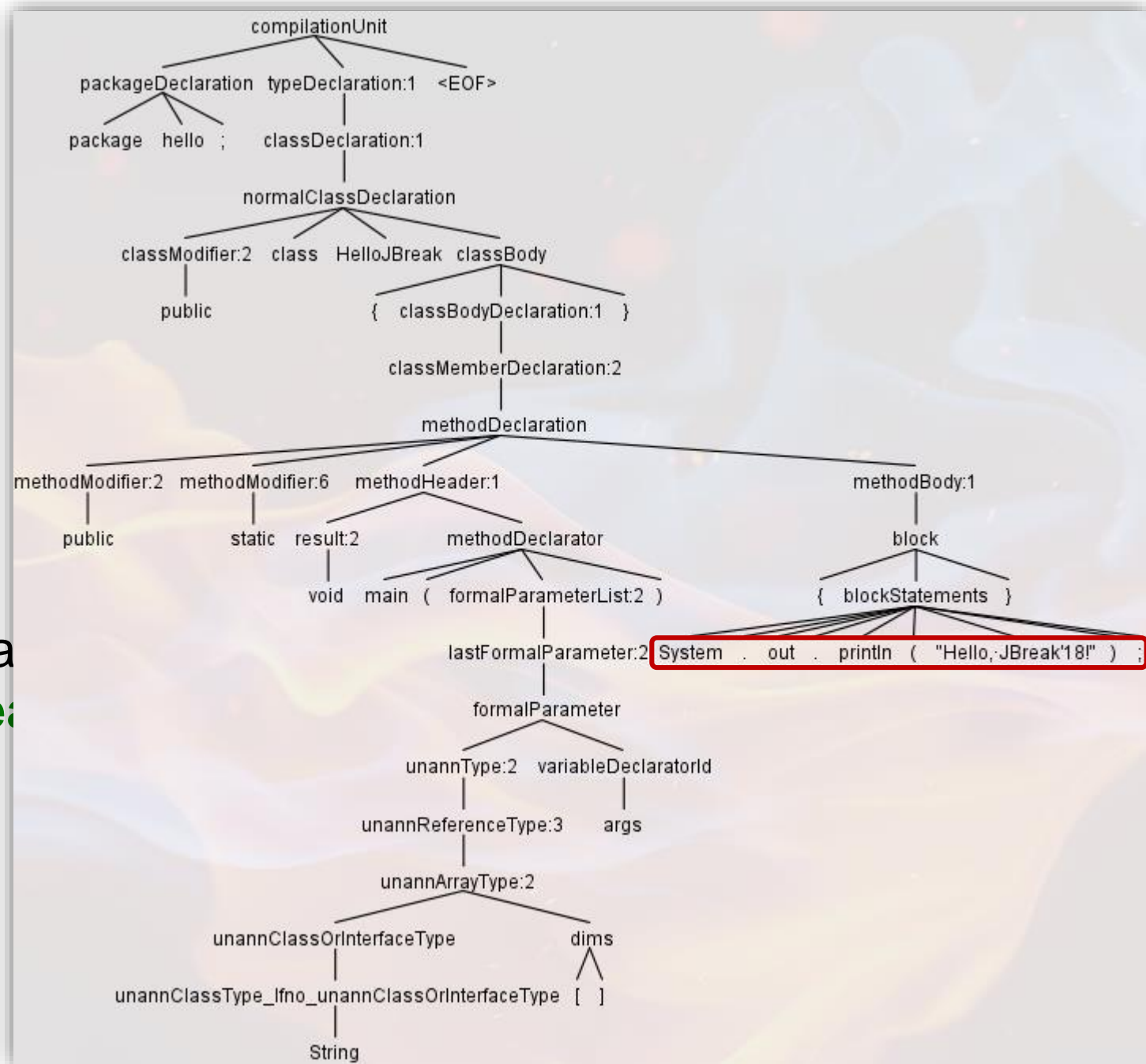
```
package hello;  
public class HelloJBreak {  
    public static void main(String[] args) {  
        System.out.println("Hello, JBreak'18!");  
    }  
}
```



Сопоставление результатов

Дерево разбора на **упрощённой** грамматике

```
package hello;  
public class HelloJBreak {  
  public static void main(String[] a  
    System.out.println("Hello, JBreak"  
}  
}
```



Неочевидные следствия

- `replaceAll()` беспощаден

```
public void makeMeDemo(String value) {  
    System.out.println("value passed in: " + value);  
}
```

```
public void makeMeDemo(String value) {  
    System.out.println("$1 passed in: " + $1);  
}
```

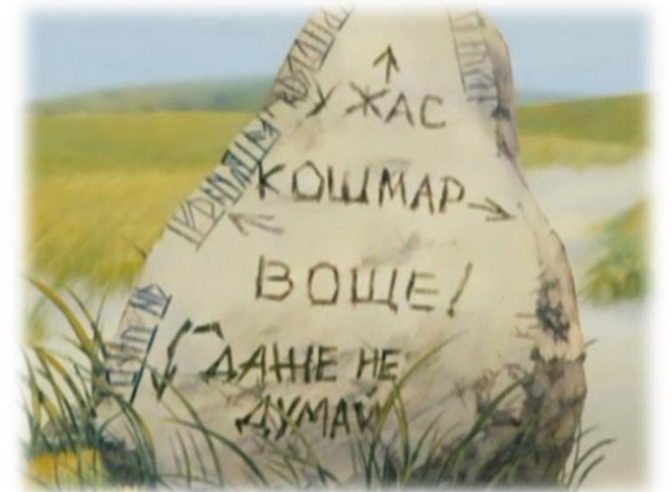
- Зависимость от выходного формата Lexer'a

А как быть с **cutpoint** ?



Варианты передачи cutpoint

- ~~Никак~~ ~~детектировать место внедрения~~ ~~автоматически~~
- ~~Через аннотацию на сигнатуре метода~~
- ~~Добавлением нового элемента в синтаксис~~
- Через тэг в javadoc к методу.

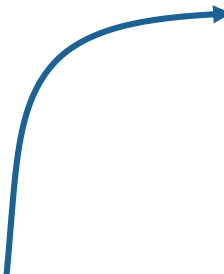


Извлечение cutpoint из javadoc

1. Пишем находим ANTLR-грамматику для javadoc
2. Генерируем по ней lexer и parser
3. Направляем все блочные комментарии основного кода в специальный канал

```
//  
// Whitespace and comments  
//  
COMMENT      : '/*' . *? '*/'      -> channel(BLOCK_COMMENTS);  
WS           : [ \t\r\n\u000C]+ -> skip;  
LINE_COMMENT : '//' ~[\r\n]* -> skip;
```

Канал (в терминах ANTLR) –
именованный накопитель лексем



Извлечение cutpoint из javadoc

4. Проверяем скрытые токены **BLOCK_COMMENTS** слева от **classBodyDeclaration***)
5. Если есть, отправляем их на разбор javadoc-парсером
6. При успешном разборе находим значение тэга **cutpoint**

```
classBodyDeclaration
:   classMemberDeclaration
  |   instanceInitializer
  |   staticInitializer
  |   constructorDeclaration
;
```

*) Обобщенное правило для декларации членов класса, в т. ч. методов

Как это будет выглядеть?

```
/**  
 * Заменяет настоящее поведение метода на тестовое  
 * @cutpoint INSTEAD  
 */  
private void doSomeMagic(Stuff stuff) {  
    // здесь код заменит собою всё тело целевого метода  
}
```

```
/**  
 * Вставляет новое поведение в начало метода  
 * @cutpoint BEFORE  
 */  
private void doSomeMagic(Stuff stuff) {  
    // здесь код встанет в самое начало целевого метода  
}
```


Практическая часть

Постановка задачи

Дано: desktop-приложение JOSM

Требуется:

1. Наглядный вывод **User-Agent**
2. Доступ к system properties
3. Задержка на стартовых задачах

Условие:

- Не делать коммитов в репозиторий
- Не включать тестовый код в дистрибутив на production

Знакомство с пациентом



JOSM

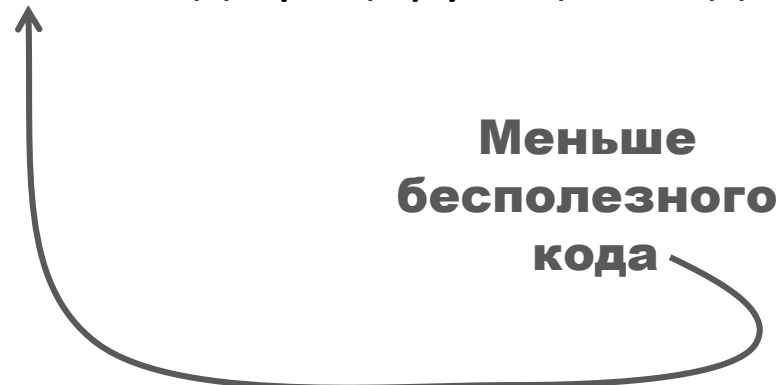


Как подступиться?

С нуля

1. Создаем пустой файл .java
2. **Пишем** в него руками декларацию целевого класса и его метода
3. Вместо тела метода пишем модифицирующий код

Меньше
бесполезного
кода



Меньше ручной
писанины



С исходника

1. Копируем целевой класс (можно с суффиксом _Droplet)
2. **Удаляем** из него всё, кроме целевого метода (и его зависимостей, если надо)
3. Вместо тела метода пишем модифицирующий код



Попробуем этот

Другие варианты применения

(Для обсуждения после доклада)

• Воспроизведение редких поведений

• Подмена инъекций в IoC контейнерах

• Патчинг библиотек без исходного кода

• Снятие ограничений безопасности



Неочевидные выводы

- Хрупкость перед рефакторингом
- Невозможность отладки
Решается отладкой в исходном коде
- непригодность для многоходовых поведений
Признак более глубокой проблемы
- Базовая комплектация
- Средство борьбы с тех. долгом

Вместо заключения

- Всеу свое место:
 - Правки конфигурации
 - Профили
 - ~~Костыли~~
 - **Side Effect Injection**
 - ...
- Чистота репозитория и совести
- Инструмент для тех, кому не по х



Side Effect Injection

или добродетельные костыли

Владимир Плизга
toparvion@gmx.com



<http://toparvion.github.io/jmint>

