YTsaurus

Патчим байткод сторонних библиотек в рантайме при помощи javaagent

Александр Токарев,
Яндекс, TeamLead группы разработки
SPYT powered by Apache Spark

16.10.2024

Обо мне

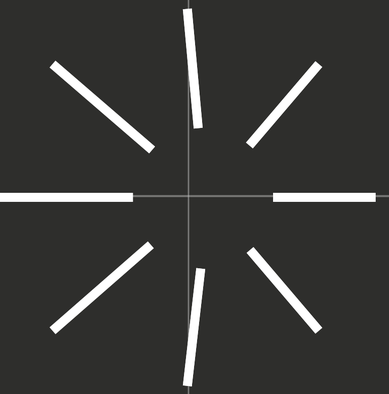
- На Java пишу с 2009 года, на Scala — с 2015
- Более 15 лет работаю бэкенд-разработчиком
- Раньше работал в Qiwi, CleverData, Leroy Merlin
- Сейчас — в Яндексе, подключаю Apache Spark к YTsaurus

Что такое YTsaurus?

- Платформа для распределённого хранения и обработки больших данных
- Ближайший аналог — экосистема Apache Hadoop
- Собственная разработка, C++
- Создана в Яндексе
- Активно разрабатывается и используется с 2010 года
- Open Source с марта 2023
- Лицензия Apache 2.0

Почему именно патчинг?

Задача: иногда при работе со сторонними библиотеками и фреймворками требуется вносить в них доработки



Проблемы

Проблемы

Закрытый исходный код

Проблемы

Закрытый исходный код

- Reverse-Engineering

Проблемы

Закрытый исходный код

- Reverse-Engineering

Открытый исходный код

Проблемы

Закрытый исходный код

- Reverse-Engineering

Открытый исходный код

- Pull-request

Проблемы

Закрытый исходный код

- Reverse-Engineering

Открытый исходный код

- Pull-request
- Code-review у маинтейнеров

Проблемы

Закрытый исходный код

- Reverse-Engineering

Открытый исходный код

- Pull-request
- Code-review у маинтейнеров
- Выпуск релиза

Проблемы

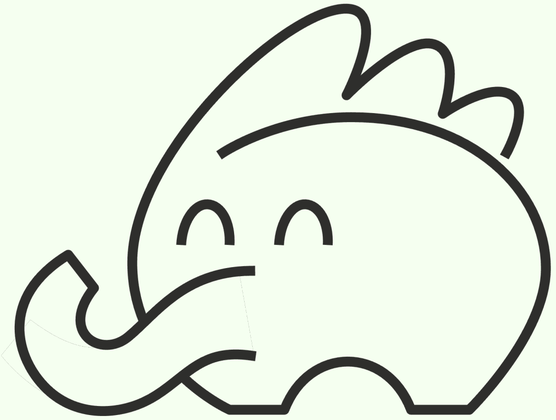
Закрытый исходный код

- Reverse-Engineering

Открытый исходный код

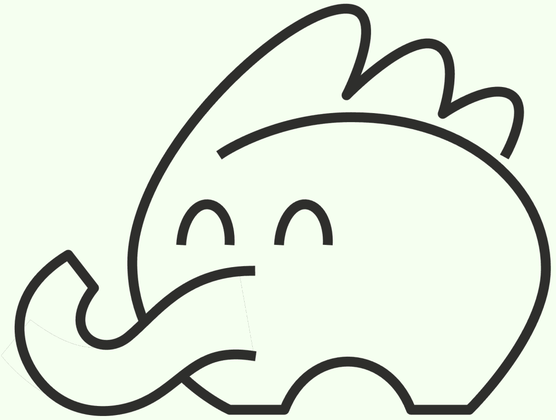
- Pull-request
- Code-review у маинтейнеров
- Выпуск релиза
- Миграция на новую версию

Вариант решения — форк библиотеки



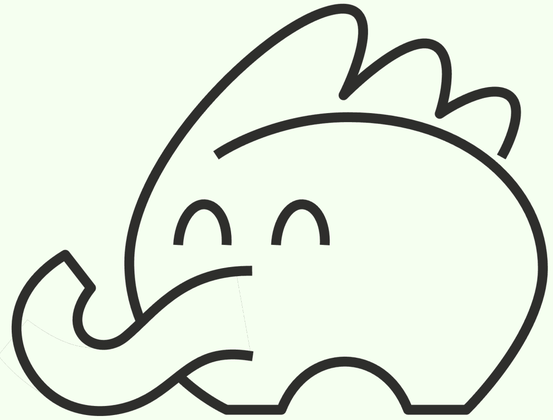
Вариант решения — форк библиотеки

- Привязаны к конкретной версии



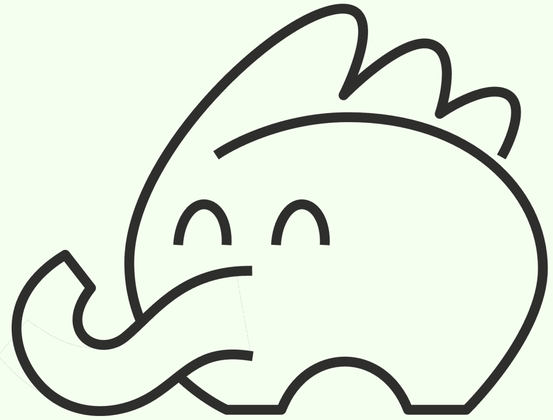
Вариант решения — форк библиотеки

- Привязаны к конкретной версии
- Получаем сильную связность между нашим кодом и кодом библиотеки



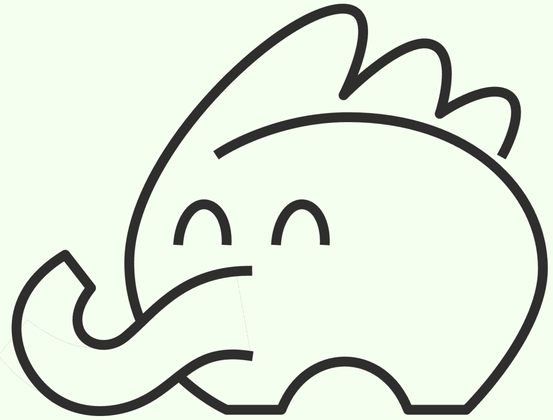
Вариант решения — форк библиотеки

- Привязаны к конкретной версии
- Получаем сильную связность между нашим кодом и кодом библиотеки
- При необходимости обновиться на новую версию нужно как-то синхронизировать наши изменения с апстримом

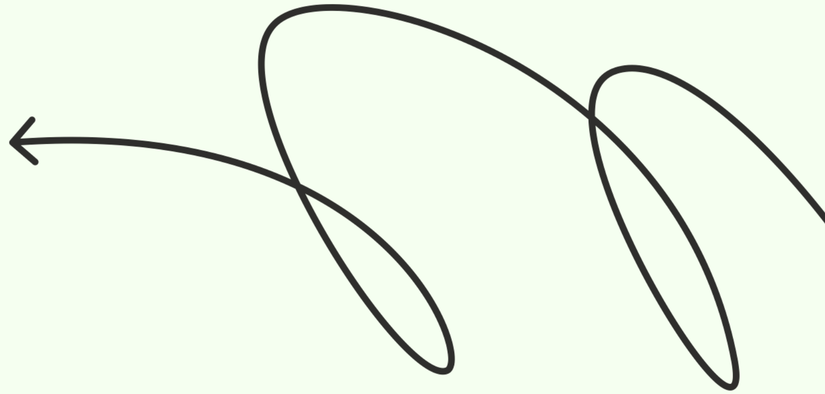


Вариант решения — форк библиотеки

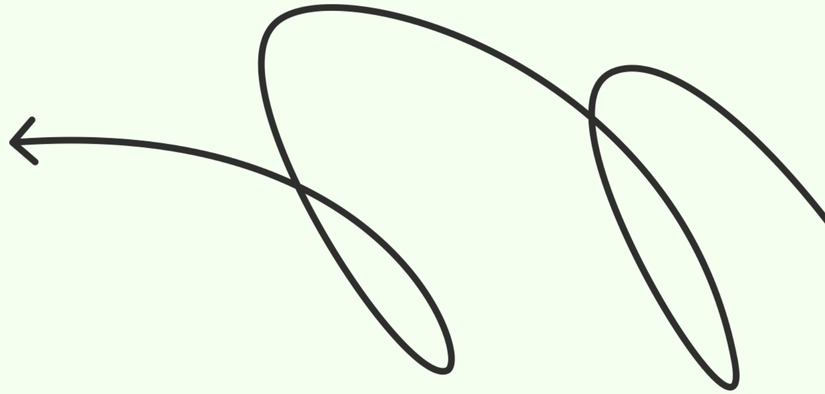
- Привязаны к конкретной версии
- Получаем сильную связность между нашим кодом и кодом библиотеки
- При необходимости обновиться на новую версию нужно как-то синхронизировать наши изменения с апстримом
- Нужно настраивать CI/CD для сборки форка, что увеличивает общее время CI/CD нашего приложения и т. д.



Проблемы с форком при Open Source



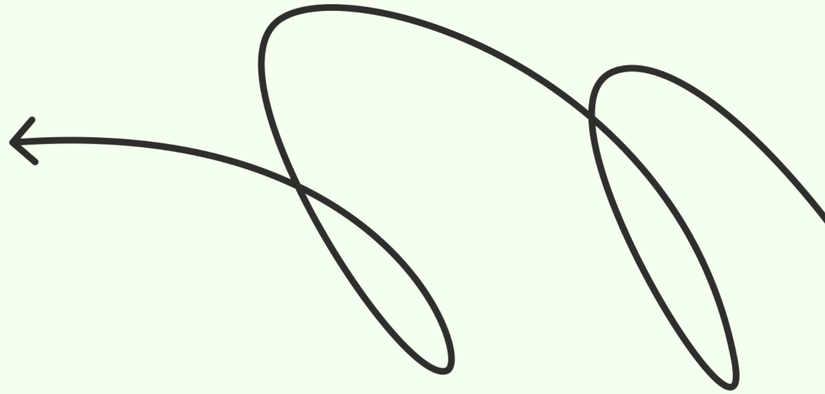
Проблемы с форком при Open Source



1

Пришлось включать
в наш репозиторий
модифицированную
версию библиотеки

Проблемы с форком при Open Source



1

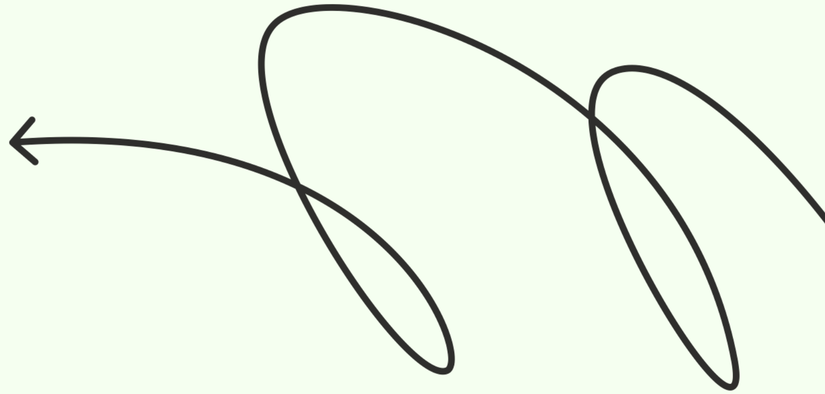
Пришлось включать
в наш репозиторий
модифицированную
версию библиотеки

2

Это увеличивает
кодовую базу проекта

Для спарка —
+2M строк кода

Проблемы с форком при Open Source



1

Пришлось включать в наш репозиторий модифицированную версию библиотеки

2

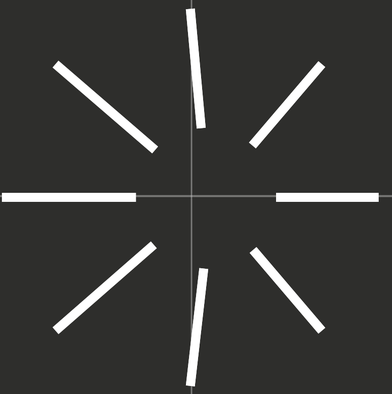
Это увеличивает кодовую базу проекта

Для спарка — +2M строк кода

3

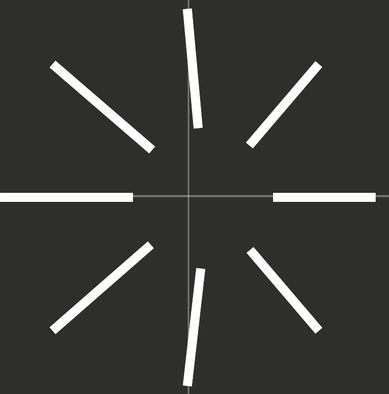
Вносит неудобства для пользователей, привыкших использовать каноническую версию библиотеки

Как бы нам откатиться до использования оригинальной версии библиотеки?



Как бы нам откатиться до использования оригинальной версии библиотеки?

И сохранить при этом все наши модификации?



Но как?

Не всегда есть возможность
использовать стандартные
подходы для расширения
функциональности,
такие как наследование,
композиция и т. д.

Но как?

Не всегда есть возможность использовать стандартные подходы для расширения функциональности, такие как наследование, композиция и т. д.

Часть методов и классов, которые мы хотим расширить — приватные, какие-то методы написаны без должной декомпозиции

Python way — MonkeyPatching

```
class Dog:
    def woof(self):
        print("WOOF")
```

Python way — MonkeyPatching

```
class Dog:
    def woof(self):
        print("WOOF")

catdog = Dog()
catdog.woof()
# prints WOOF
```

Python way — MonkeyPatching

```
class Dog:
    def woof(self):
        print("WOOF")

catdog = Dog()
catdog.woof()
# prints WOOF

def meow(self):
    print("MEOW")
```

Python way — MonkeyPatching

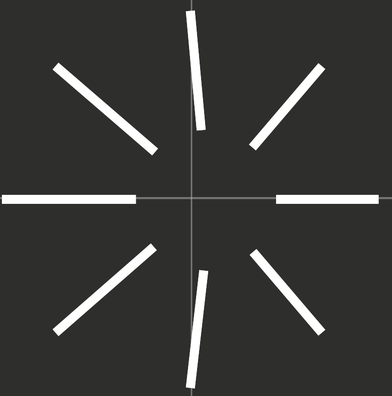
```
class Dog:
    def woof(self):
        print("WOOF")

catdog = Dog()
catdog.woof()
# prints WOOF

def meow(self):
    print("MEOW")

Dog.woof = meow
```

Python way — MonkeyPatching

```
class Dog:
    def woof(self):
        print("WOOF")

catdog = Dog()
catdog.woof()
# prints WOOF

def meow(self):
    print("MEOW")

Dog.woof = meow

catdog.woof()
# prints MEOW
```

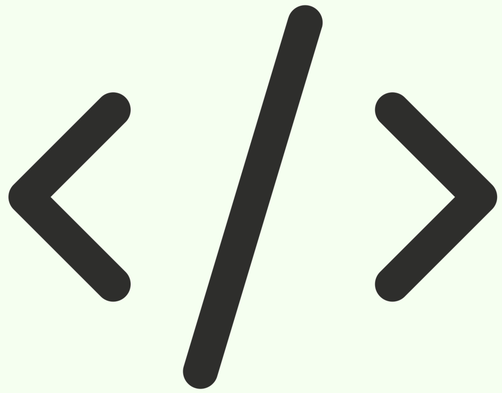
Котопёс

Проблема: Python — интерпретируемый язык, а Java (и Scala) — компилируемый



Как сделать MonkeyPatching скомпилированного кода?

ClassLoader — точка загрузки байткода классов в память процесса jvm



Можно ли повесить хук на ClassLoader?

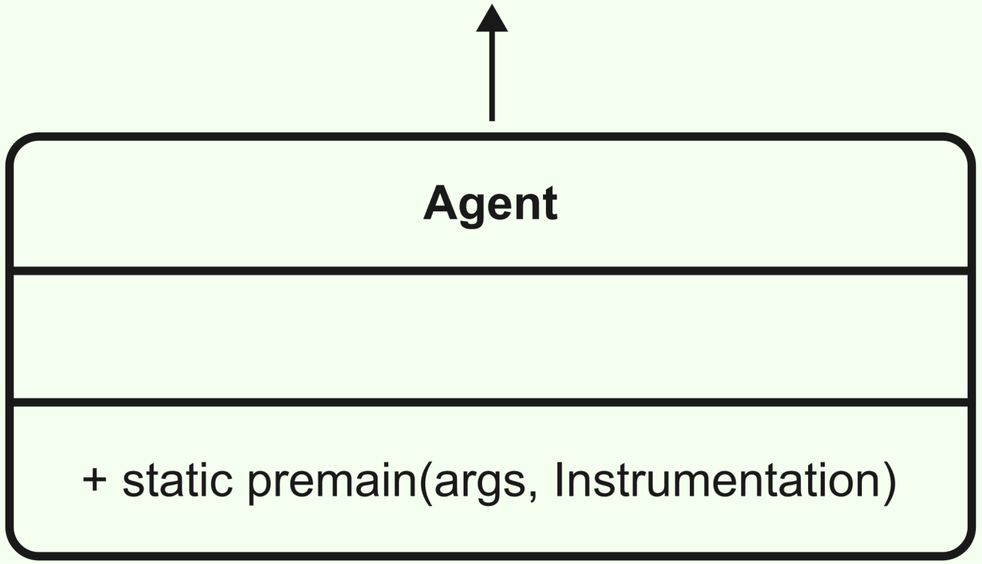
Можно! Для этого есть javaagent!

Можно! Для этого есть javaagent!

```
java -javaagent:agent.jar -jar application.jar
```

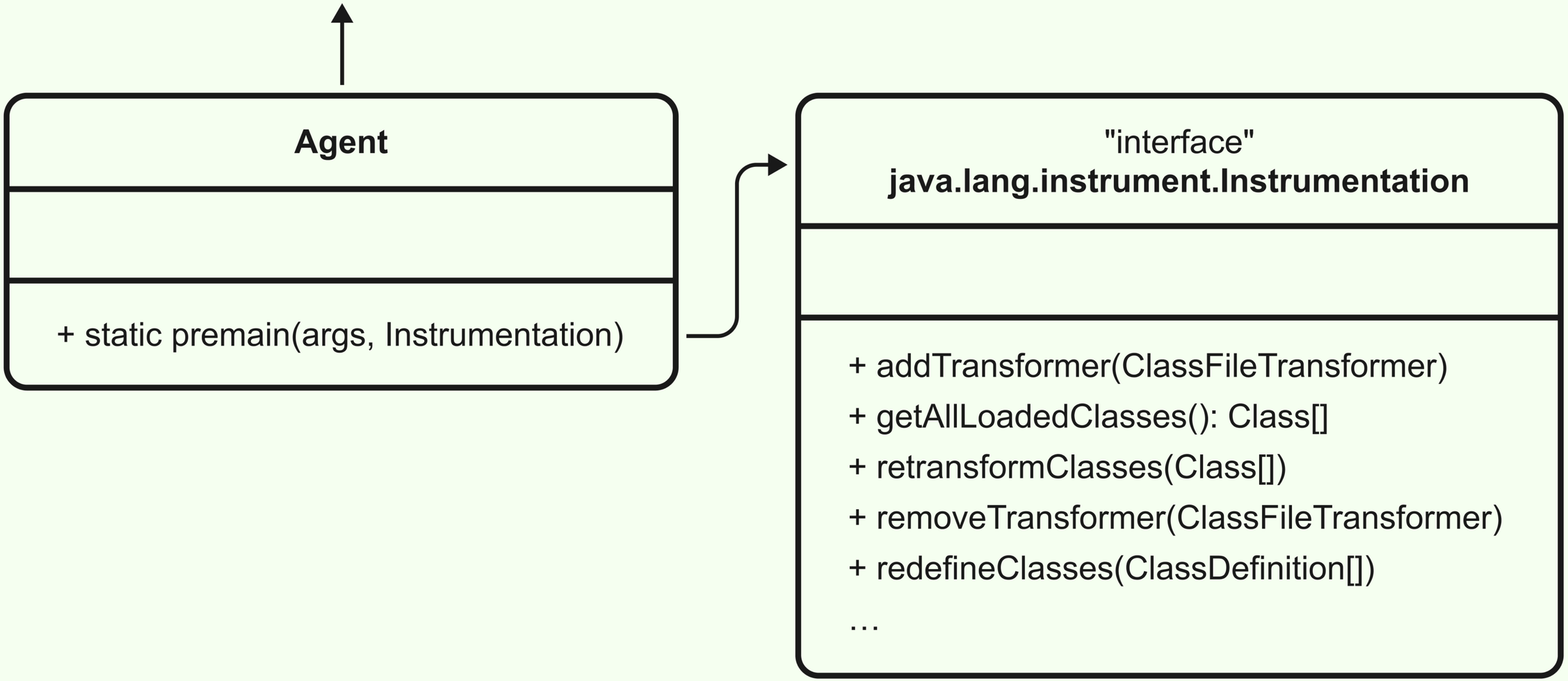
Можно! Для этого есть javaagent!

```
java -javaagent:agent.jar -jar application.jar
```



Можно! Для этого есть javaagent!

```
java –javaagent:agent.jar –jar application.jar
```



ОК, хук повесили,
а как модифицировать
сам класс?

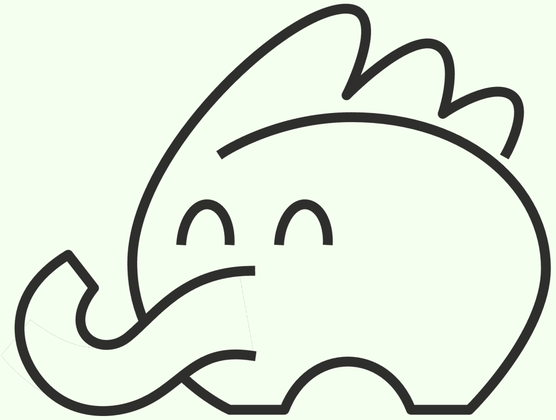
```java
package com.jokerconf.patching.example;


2 usages
public class Dog {
    1 usage
    public void woof() {
        System.out.println("WOOF");
    }
}
```

```
hexdump -C com/jokerconf/patching/example/Dog.class

00000000  ca fe ba be 00 00 00 37  00 1c 0a 00 06 00 0e 09  |.......7........|
00000010  00 0f 00 10 08 00 11 0a  00 12 00 13 07 00 14 07  |................|
00000020  00 15 01 00 06 3c 69 6e  69 74 3e 01 00 03 28 29  |.....<init>...()|
00000030  56 01 00 04 43 6f 64 65  01 00 0f 4c 69 6e 65 4e  |V...Code...LineN|
00000040  75 6d 62 65 72 54 61 62  6c 65 01 00 04 77 6f 6f  |umberTable...woo|
00000050  66 01 00 0a 53 6f 75 72  63 65 46 69 6c 65 01 00  |f...SourceFile..|
00000060  08 44 6f 67 2e 6a 61 76  61 0c 00 07 00 08 07 00  |.Dog.java.......|
00000070  16 0c 00 17 00 18 01 00  04 57 4f 4f 46 07 00 19  |.........WOOF...|
00000080  0c 00 1a 00 1b 01 00 22  63 6f 6d 2f 6a 6f 6b 65  |.......".com/joke|
00000090  72 63 6f 6e 66 2f 70 61  74 63 68 69 6e 67 2f 65  |rconf/patching/e|
000000a0  78 61 6d 70 6c 65 2f 44  6f 67 01 00 10 6a 61 76  |xample/Dog...jav|
000000b0  61 2f 6c 61 6e 67 2f 4f  62 6a 65 63 74 01 00 10  |a/lang/Object...|
000000c0  6a 61 76 61 2f 6c 61 6e  67 2f 53 79 73 74 65 6d  |java/lang/System|
000000d0  01 00 03 6f 75 74 01 00  15 4c 6a 61 76 61 2f 69  |...out...Ljava/i|
000000e0  6f 2f 50 72 69 6e 74 53  74 72 65 61 6d 3b 01 00  |o/PrintStream;..|
000000f0  13 6a 61 76 61 2f 69 6f  2f 50 72 69 6e 74 53 74  |.java/io/PrintSt|
00000100  72 65 61 6d 01 00 07 70  72 69 6e 74 6c 6e 01 00  |ream...println..|
00000110  15 28 4c 6a 61 76 61 2f  6c 61 6e 67 2f 53 74 72  |.(Ljava/lang/Str|
00000120  69 6e 67 3b 29 56 00 21  00 05 00 06 00 00 00 00  |ing;)V.!........|
00000130  00 02 00 01 00 07 00 08  00 01 00 09 00 00 00 1d  |................|
00000140  00 01 00 01 00 00 00 05  2a b7 00 01 b1 00 00 00  |........*.......|
00000150  01 00 0a 00 00 00 06 00  01 00 00 00 03 00 01 00  |................|
00000160  0b 00 08 00 01 00 09 00  00 00 25 00 02 00 01 00  |..........%.....|
00000170  00 00 09 b2 00 02 12 03  b6 00 04 b1 00 00 00 01  |................|
00000180  00 0a 00 00 00 0a 00 02  00 00 00 05 00 08 00 06  |................|
00000190  00 01 00 0c 00 00 00 02  00 0d                    |..........|
```

```
Classfile /src/main/java/com/jokerconf/patching/example/Dog.class
  Last modified 22 Jul 2024; size 410 bytes
  MD5 checksum 67d8ad1df364018384226d5750ee2fd5
  Compiled from "Dog.java"

public class com.jokerconf.patching.example.Dog
  minor version: 0
  major version: 55
  flags: (0x0021) ACC_PUBLIC, ACC_SUPER
  this_class: #5                          // com/jokerconf/patching/example/Dog
  super_class: #6                         // java/lang/Object
  interfaces: 0, fields: 0, methods: 2, attributes: 1
```

```
Constant pool:
   #1 = Methodref    #6.#14
   #2 = Fieldref     #15.#16
   #3 = String       #17
   #4 = Methodref    #18.#19
   #5 = Class        #20
   #6 = Class        #21
   #7 = Utf8         <init>
   #8 = Utf8         ()V
   #9 = Utf8         Code
  #10 = Utf8         LineNumberTable
  #11 = Utf8         woof
  #12 = Utf8         SourceFile
  #13 = Utf8         Dog.java
  #14 = NameAndType  #7:#8
  #15 = Class        #22
  #16 = NameAndType  #23:#24
  #17 = Utf8         WOOF
  #18 = Class        #25
  #19 = NameAndType  #26:#27
```

```
#20 = Utf8  com/jokerconf/patching/example/Dog
#21 = Utf8  java/lang/Object
#22 = Utf8  java/lang/System
#23 = Utf8  out
#24 = Utf8  Ljava/io/PrintStream;
#25 = Utf8  java/io/PrintStream
#26 = Utf8  println
#27 = Utf8  (Ljava/lang/String;)V
```



```
{
  public com.jokerconf.patching.example.Dog();
    descriptor: ()V
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1          // Method java/lang/Object."<init>":()V
         4: return
      LineNumberTable:
        line 3: 0

  public void woof();
    ...
    Code:
      stack=2, locals=1, args_size=1
         0: getstatic     #2          // Field java/lang/System.out:Ljava/io/PrintStream;
         3: ldc           #3          // String WOOF
         5: invokevirtual #4          // Method java/io/PrintStream.println:(Ljava/lang/String;)V
         8: return
      LineNumberTable:
        line 5: 0
        line 6: 8
}
```

Библиотеки для работы с байткодом

Библиотеки для работы с байткодом

- Javassist (javassist.org)

Библиотеки для работы с байткодом

- Javassist (javassist.org)
- Asm (asm.ow2.io)

Библиотеки для работы с байткодом

- Javassist (javassist.org)
- Asm (asm.ow2.io)
- BCEL (commons.apache.org/proper/commons-bcel)

Библиотеки для работы с байткодом

- Javassist (javassist.org)
- Asm (asm.ow2.io)
- BCEL (commons.apache.org/proper/commons-bcel)
- Byte buddy (bytebuddy.net)

Библиотеки для работы с байткодом

- Javassist (javassist.org)
- Asm (asm.ow2.io)
- BCEL (commons.apache.org/proper/commons-bcel)
- Byte buddy (bytebuddy.net)
- Janino (janino.net)

Библиотеки для работы с байткодом

- Javassist (javassist.org)
- Asm (asm.ow2.io)
- BCEL (commons.apache.org/proper/commons-bcel)
- Byte buddy (bytebuddy.net)
- Janino (janino.net)
- Soot (сейчас SootUp, soot-oss.github.io/SootUp)

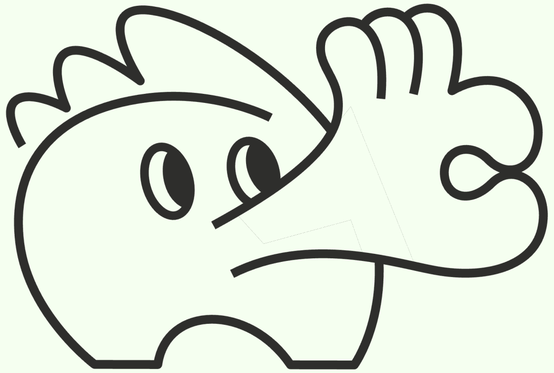
Библиотеки для работы с байткодом

- Javassist (javassist.org)
- Asm (asm.ow2.io)
- BCEL (commons.apache.org/proper/commons-bcel)
- Byte buddy (bytebuddy.net)
- Janino (janino.net)
- Soot (сейчас SootUp, soot-oss.github.io/SootUp)
- JDK Class-File API (since Java 22)

Библиотеки для работы с байткодом

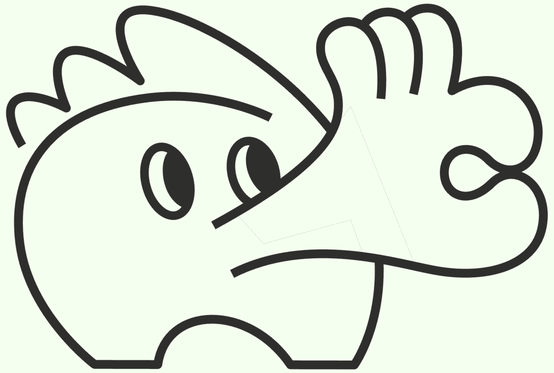
- **Javassist (javassist.org)**
- Asm (asm.ow2.io)
- BCEL (commons.apache.org/proper/commons-bcel)
- Byte buddy (bytebuddy.net)
- Janino (janino.net)
- Soot (сейчас SootUp, soot-oss.github.io/SootUp)
- JDK Class-File API (since Java 22)

MonkeyPatching в JVM



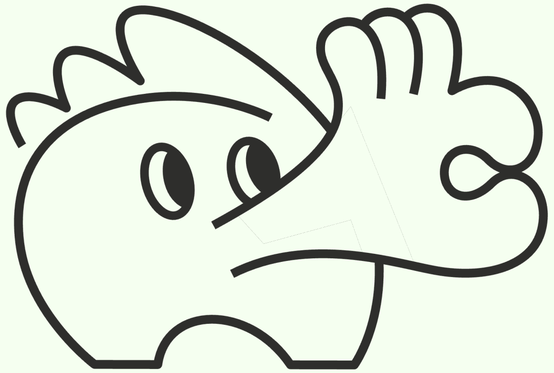
MonkeyPatching в JVM

- Получаем доступ к исходному байткоду класса во время его загрузки в память через javaagent



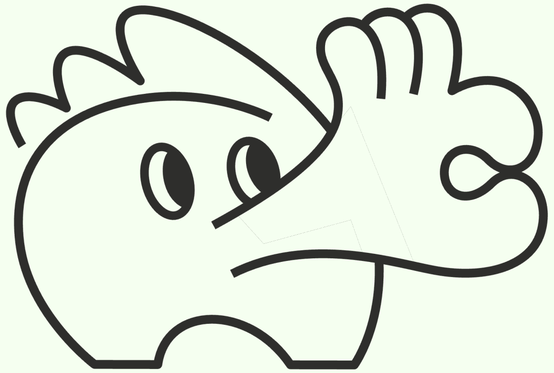
MonkeyPatching в JVM

- Получаем доступ к исходному байткоду класса во время его загрузки в память через javaagent
- Модифицируем его при помощи одной из библиотек для работы с байткодом



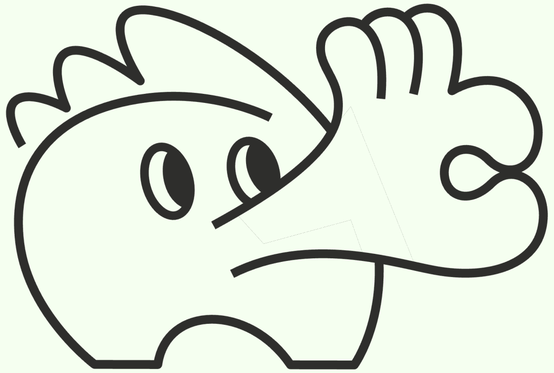
MonkeyPatching в JVM

- Получаем доступ к исходному байткоду класса во время его загрузки в память через javaagent
- Модифицируем его при помощи одной из библиотек для работы с байткодом
- Отдаём ClassLoader-у измененный байткод



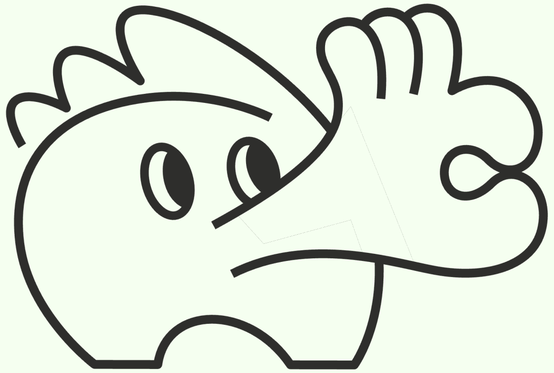
MonkeyPatching в JVM

- Получаем доступ к исходному байткоду класса во время его загрузки в память через javaagent
- Модифицируем его при помощи одной из библиотек для работы с байткодом
- Отдаём ClassLoader-у измененный байткод
- Получаем в runtime измененное поведение



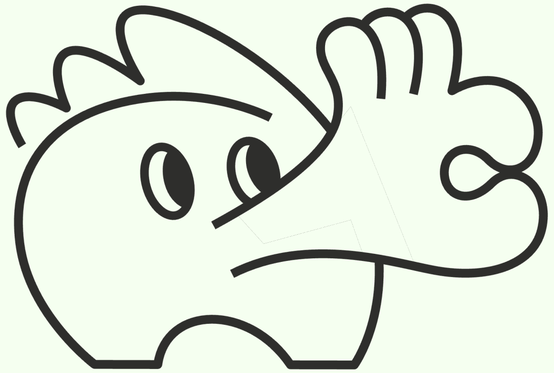
MonkeyPatching в JVM

- Получаем доступ к исходному байткоду класса во время его загрузки в память через javaagent
- Модифицируем его при помощи одной из библиотек для работы с байткодом
- Отдаём ClassLoader-у измененный байткод
- Получаем в runtime измененное поведение
- …



MonkeyPatching в JVM

- Получаем доступ к исходному байткоду класса во время его загрузки в память через javaagent
- Модифицируем его при помощи одной из библиотек для работы с байткодом
- Отдаём ClassLoader-у измененный байткод
- Получаем в runtime измененное поведение
- …
- PROFIT!!!



Байткод класса — это его ДНК

ilksesgazetesi.com/saglik/gen-terapisi-nedir-gen-terapisi-nasil-yapilir-321373

Class runtime patching framework

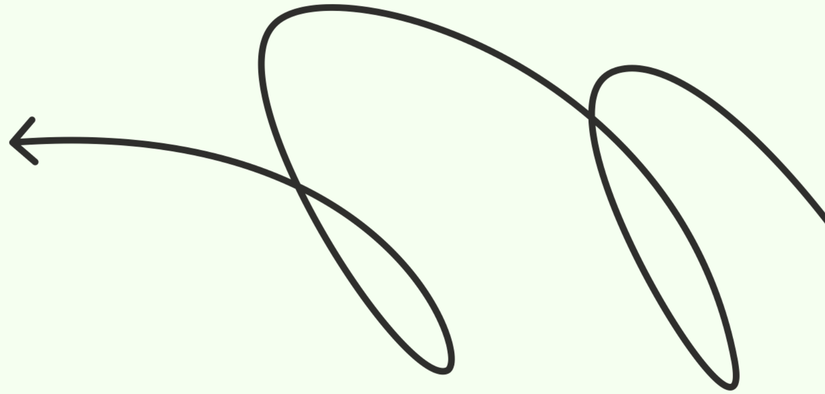
Класс PatchAgent

```
1 usage
public class PatchAgent {
    no usages
    public static void premain(String args, Instrumentation inst) {
        inst.addTransformer(new SampleTransformer());
    }
}
```

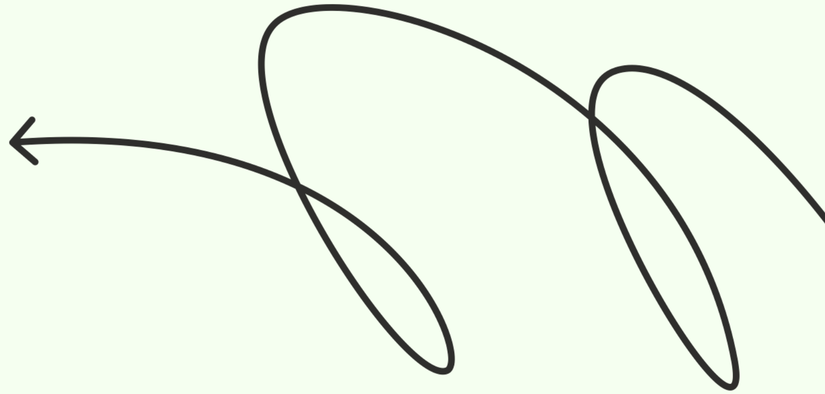
Интерфейс ClassFileTransformer

```
class SampleTransformer implements ClassFileTransformer {

    @Override
    public byte[] transform(
                ClassLoader loader,
                String className,
                Class<?> classBeingRedefined,
                ProtectionDomain protectionDomain,
                byte[] originalClassBytes) throws IllegalClassFormatException {

        byte[] transformedBytes = process(originalClassBytes);

        return transformedBytes;
    }
}
```

Четыре способа



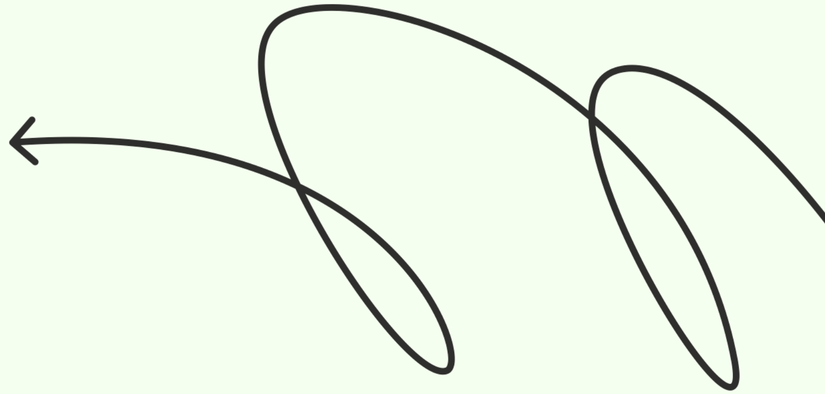
Четыре способа



1

Полная
замена
байткода
класса

Четыре способа



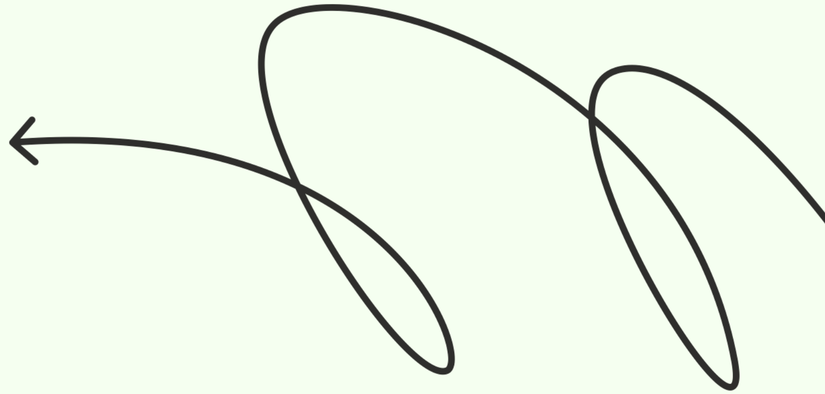
1

Полная замена байткода класса

2

Замена класса подклассом

Четыре способа



1

Полная замена байткода класса

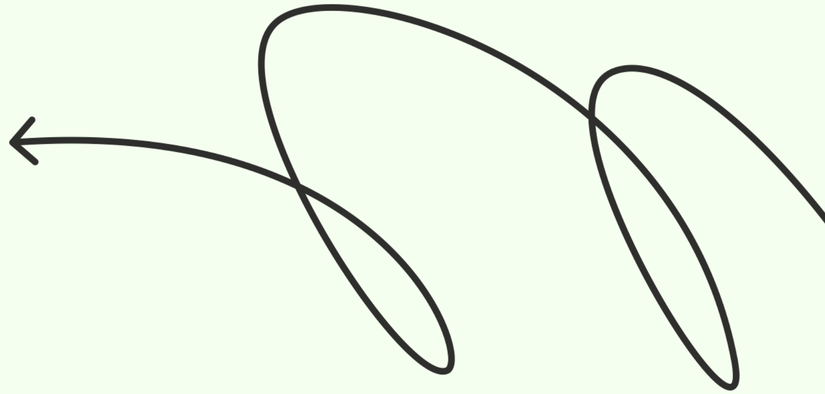
2

Замена класса подклассом

3

Декорирование методов

Четыре способа



1

Полная замена байткода класса

2

Замена класса подклассом

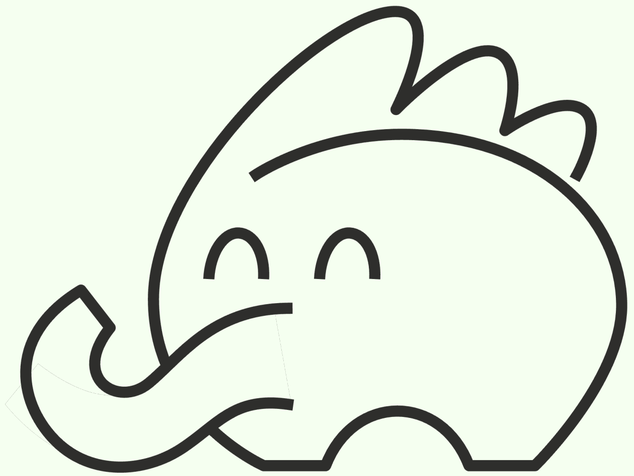
3

Декорирование методов

4

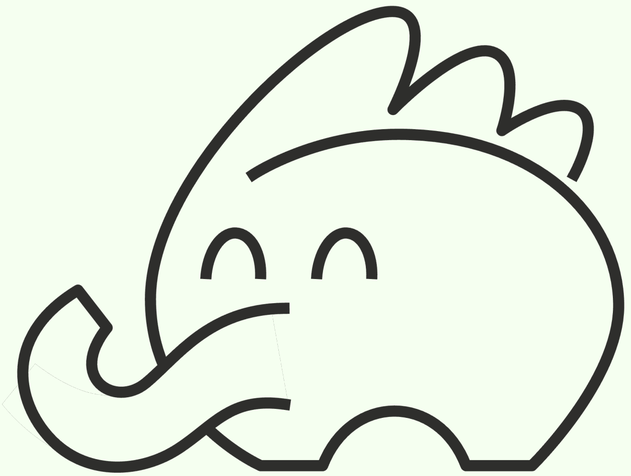
Манипуляции на уровне байткода

Аннотации



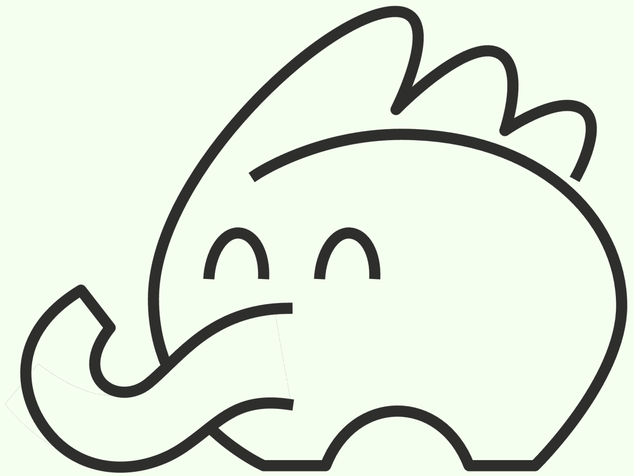
Аннотации

- `@OriginClass` — указываем, какой класс хотим пропатчить (все способы)



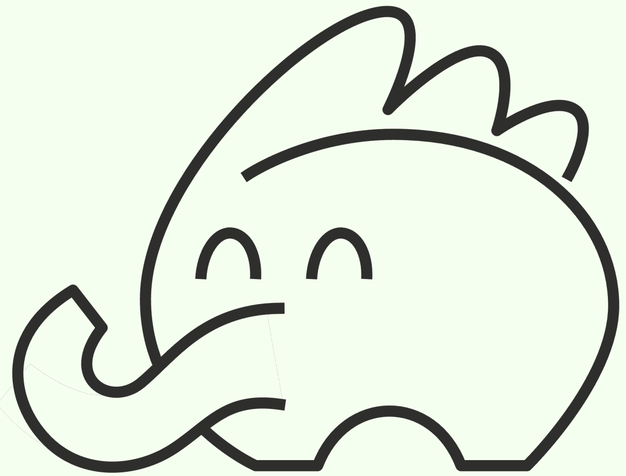
Аннотации

- `@OriginClass` — указываем, какой класс хотим пропатчить (все способы)
- `@Subclass` — заменить оригинальный класс подклассом (способ 2)



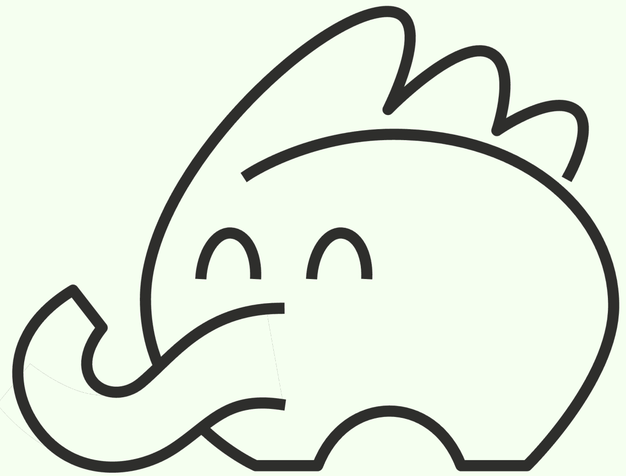
Аннотации

- `@OriginClass` — указываем, какой класс хотим пропатчить (все способы)
- `@Subclass` — заменить оригинальный класс подклассом (способ 2)
- `@Decorate` — декорировать определенные методы оригинального класса (способ 3)



Аннотации

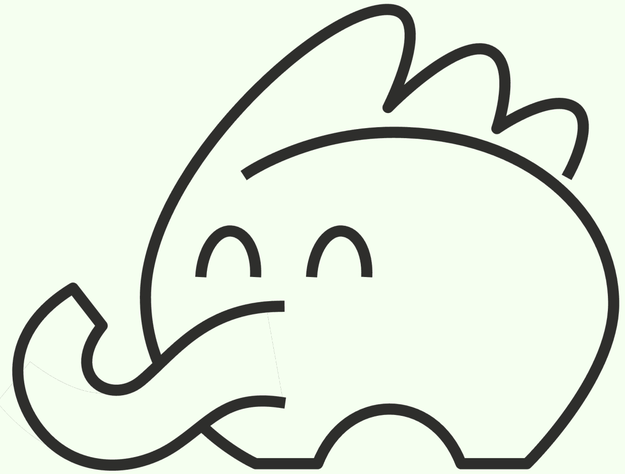
- `@OriginClass` — указываем, какой класс хотим пропатчить (все способы)
- `@Subclass` — заменить оригинальный класс подклассом (способ 2)
- `@Decorate` — декорировать определенные методы оригинального класса (способ 3)
- `@DecoratedMethod` — указывает декорирующий метод и задаёт правила обработки байткода исходного метода (способы 3 и 4)



Аннотации

- `@OriginClass` — указываем, какой класс хотим пропатчить (все способы)
- `@Subclass` — заменить оригинальный класс подклассом (способ 2)
- `@Decorate` — декорировать определенные методы оригинального класса (способ 3)
- `@DecoratedMethod` — указывает декорирующий метод и задаёт правила обработки байткода исходного метода (способы 3 и 4)

github.com/ytsaurus/ytsaurus-spyt/tree/main/sparkpatch/src/main/java/tech/ytsaurus/spyt/patch/annotations



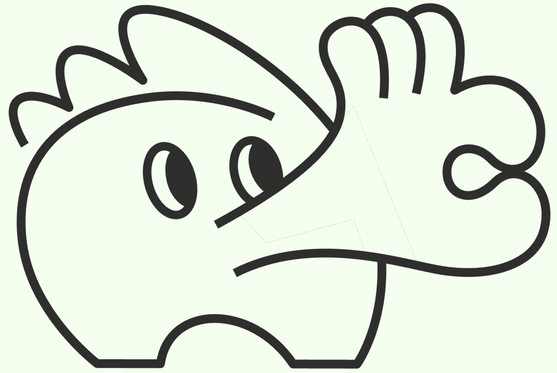
```java
package com.jokerconf.patching.example;


2 usages
public class Dog {
    1 usage
    public void woof() {
        System.out.println("WOOF");
    }
}
```

```java
package com.jokerconf.patching.example;


public class Main {
    public static void main(String[] args) {
        Dog catdog = new Dog();
        catdog.woof();
    }
}
```

Способ 1. Полная замена байткода класса

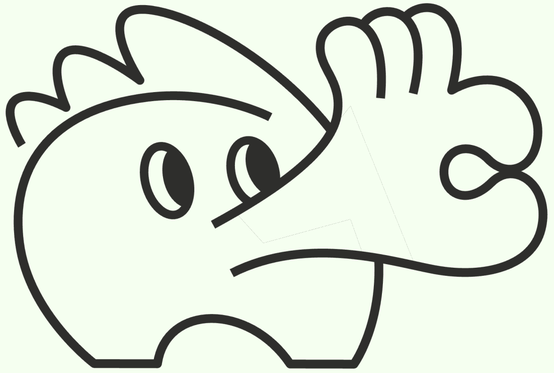
Берём класс из нашего проекта, и подменяем им класс из фреймворка

Способ 1.
Реализация



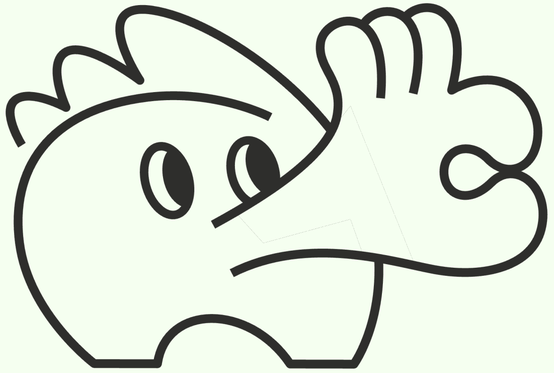
Способ 1. Реализация

- Пишем новый класс



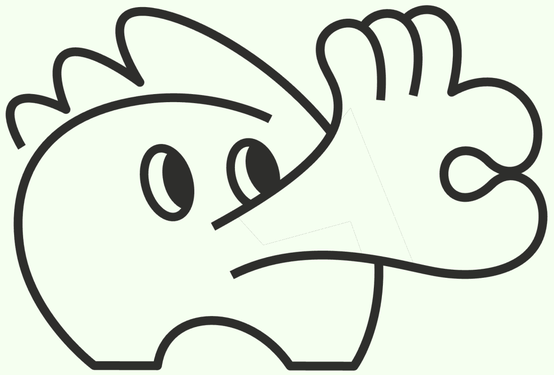
Способ 1.
Реализация

- Пишем новый класс
- Помечаем его аннотацией @OriginClass



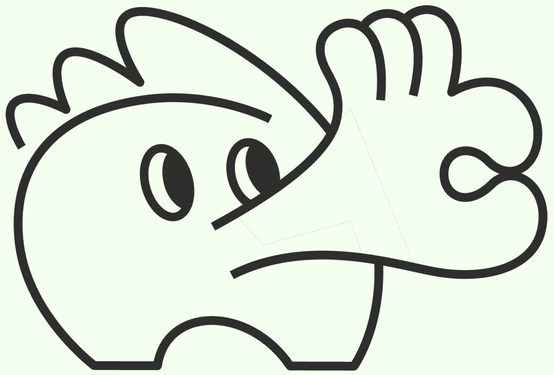
Способ 1.
Реализация

- Пишем новый класс
- Помечаем его аннотацией @OriginClass
- В методе transform при загрузке оригинального класса подменяем его байткод на байткод модифицированного класса



Способ 1.
Реализация

- Пишем новый класс
- Помечаем его аннотацией @OriginClass
- В методе transform при загрузке оригинального класса подменяем его байткод на байткод модифицированного класса
- В байткоде модифицированного класса изменяем название класса на оригинальное



```java
package com.jokerconf.patching.example;


no usages
@OriginClass("com.jokerconf.patching.example.Dog")
public class PatchedDog {
    no usages
    public void woof() {
        System.out.println("MEOW");
    }
}
```

После декомпиляции

```
package com.jokerconf.patching.example;

no usages
@OriginClass("com.jokerconf.patching.example.Dog")
public class Dog {
    no usages
    public Dog() {
    }

    1 usage
    public void woof() {
        System.out.println("MEOW");
    }
}
```

```
public byte[] transform(
        ClassLoader loader,
        String className,
        Class<?> classBeingRedefined,
        ProtectionDomain pd,
        byte[] classfileBuffer) throws IOException {

    if (patchedClasses.contains(className)) {
        byte[] patchedClassBytes = loadClassBytes(toPatchClassName(className));
```

```
public byte[] transform(
        ClassLoader loader,
        String className,
        Class<?> classBeingRedefined,
        ProtectionDomain pd,
        byte[] classfileBuffer) throws IOException {

    if (patchedClasses.contains(className)) {
        byte[] patchedClassBytes = loadClassBytes(toPatchClassName(className));
        ClassFile cf = new ClassFile(new DataInputStream(new ByteArrayInputStream(patchedClassBytes)));
```

```
public byte[] transform(
        ClassLoader loader,
        String className,
        Class<?> classBeingRedefined,
        ProtectionDomain pd,
        byte[] classfileBuffer) throws IOException {

    if (patchedClasses.contains(className)) {
        byte[] patchedClassBytes = loadClassBytes(toPatchClassName(className));
        ClassFile cf = new ClassFile(new DataInputStream(new ByteArrayInputStream(patchedClassBytes)));
        cf.renameClass(classMappings);
```

```
public byte[] transform(
        ClassLoader loader,
        String className,
        Class<?> classBeingRedefined,
        ProtectionDomain pd,
        byte[] classfileBuffer) throws IOException {

    if (patchedClasses.contains(className)) {
        byte[] patchedClassBytes = loadClassBytes(toPatchClassName(className));
        ClassFile cf = new ClassFile(new DataInputStream(new ByteArrayInputStream(patchedClassBytes)));
        cf.renameClass(classMappings);
        ByteArrayOutputStream patchedBytesOutputStream = new ByteArrayOutputStream();
        cf.write(new DataOutputStream(patchedBytesOutputStream));
        patchedBytesOutputStream.flush();
```

```
public byte[] transform(
        ClassLoader loader,
        String className,
        Class<?> classBeingRedefined,
        ProtectionDomain pd,
        byte[] classfileBuffer) throws IOException {

    if (patchedClasses.contains(className)) {
        byte[] patchedClassBytes = loadClassBytes(toPatchClassName(className));
        ClassFile cf = new ClassFile(new DataInputStream(new ByteArrayInputStream(patchedClassBytes)));
        cf.renameClass(classMappings);
        ByteArrayOutputStream patchedBytesOutputStream = new ByteArrayOutputStream();
        cf.write(new DataOutputStream(patchedBytesOutputStream));
        patchedBytesOutputStream.flush();
        return patchedBytesOutputStream.toByteArray();
    }

    return null;
}
```

Способ 1. Преимущества и недостатки

Простота реализации

Способ 1. Преимущества и недостатки

Простота реализации

Необходимо копировать полный код оригинального класса

Способ 1. Применимость

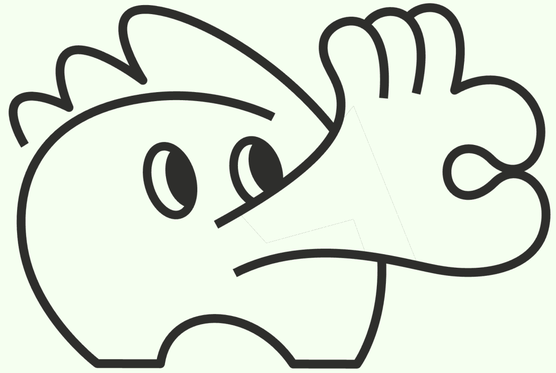
Подходит только
для маленьких и редко
изменяемых классов

```java
package com.jokerconf.patching.example;

3 usages  1 inheritor
public class Dog {
    1 usage
    private void eat() { System.out.println("Eat a bone"); }
    1 usage
    private void sleep() { System.out.println("HRRR!!! HRRRR!!!"); }
    1 usage
    private boolean isAlive() { return true; }
    no usages
    public void live() {
        while (isAlive()) {
            eat();
            sleep();
        }
    }

    1 usage  1 override
    public void woof() {
        System.out.println("WOOF");
    }
}
```

Способ 2. Замена класса подклассом

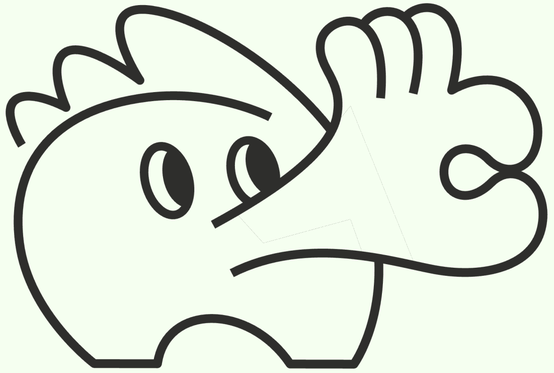
Переименовываем базовый класс
и наследуем от него класс
с именем оригинального класса

Способ 2.
Реализация



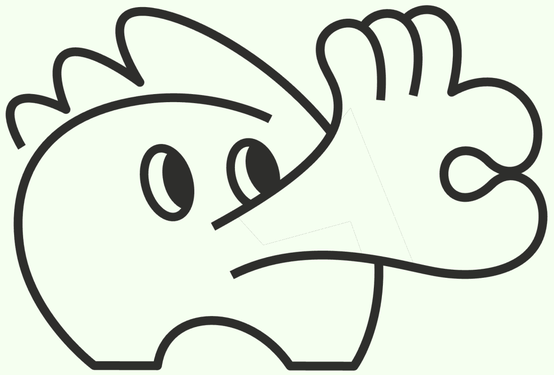
Способ 2.
Реализация

- Пишем свой класс и наследуем его от оригинального класса



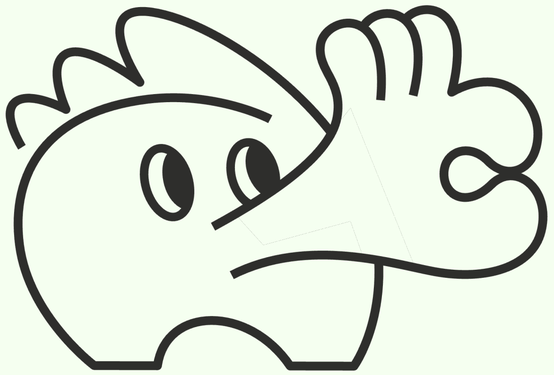
Способ 2.
Реализация

- Пишем свой класс и наследуем его от оригинального класса
- Помечаем аннотациями @Subclass и @OriginClass



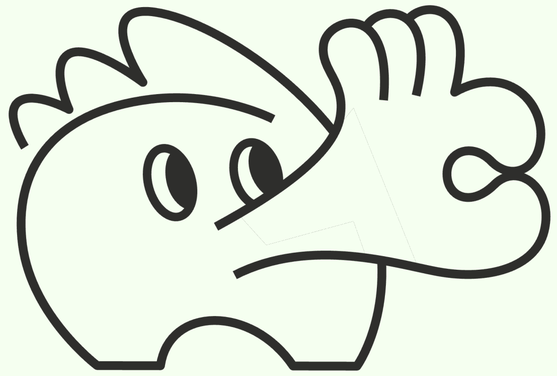
Способ 2.
Реализация

- Пишем свой класс и наследуем его от оригинального класса
- Помечаем аннотациями @Subclass и @OriginClass
- При загрузке переименовываем оригинальный класс в <classname>Base, и также загружаем его в classloader

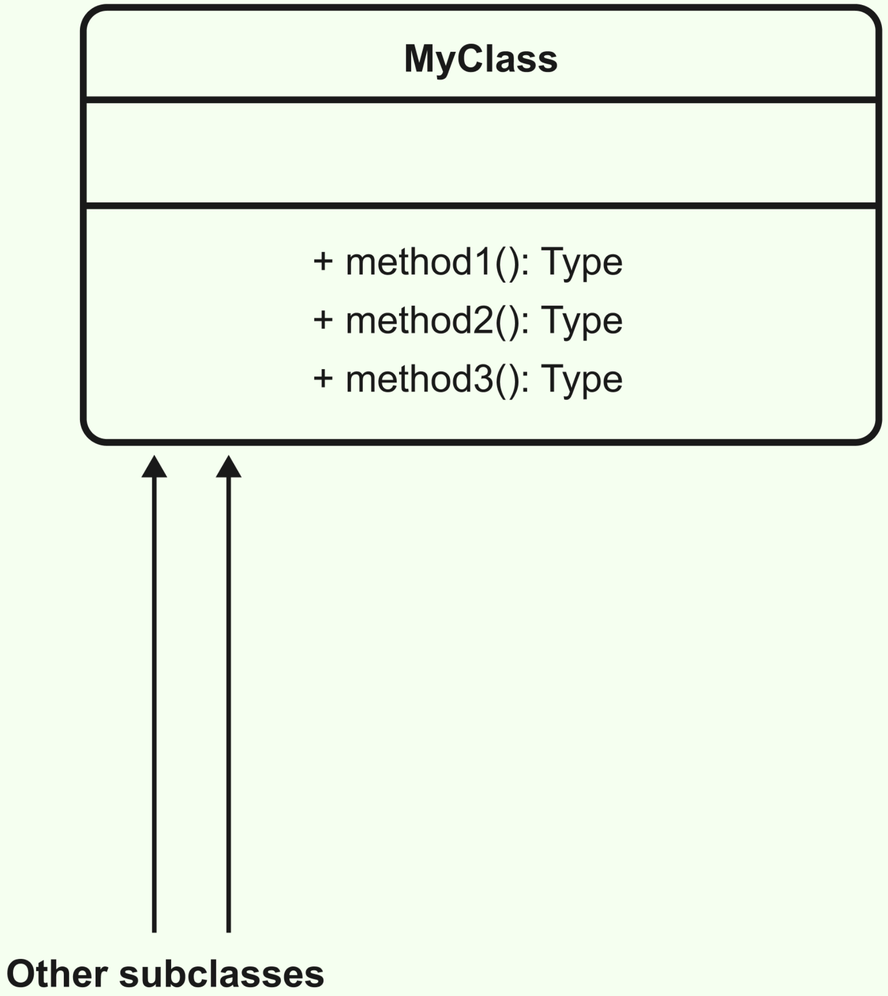


Способ 2.
Реализация

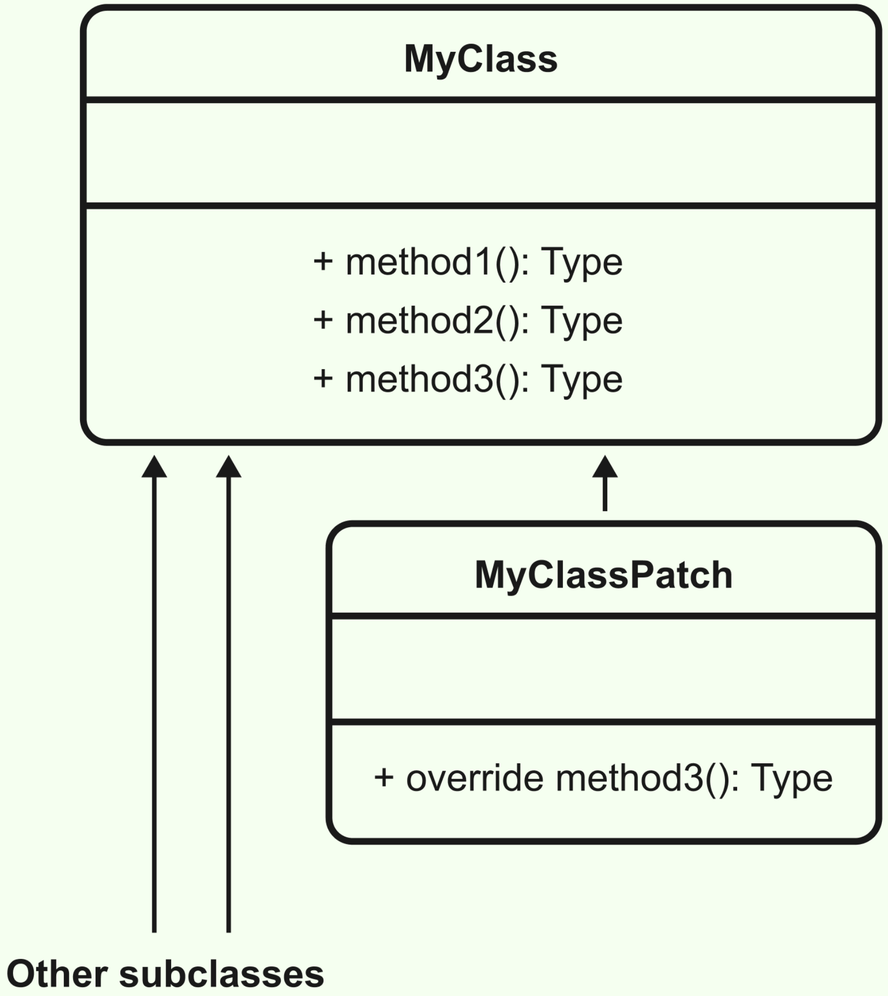
- Пишем свой класс и наследуем его от оригинального класса
- Помечаем аннотациями @Subclass и @OriginClass
- При загрузке переименовываем оригинальный класс в <classname>Base, и также загружаем его в classloader
- Патчу даём имя оригинального класса



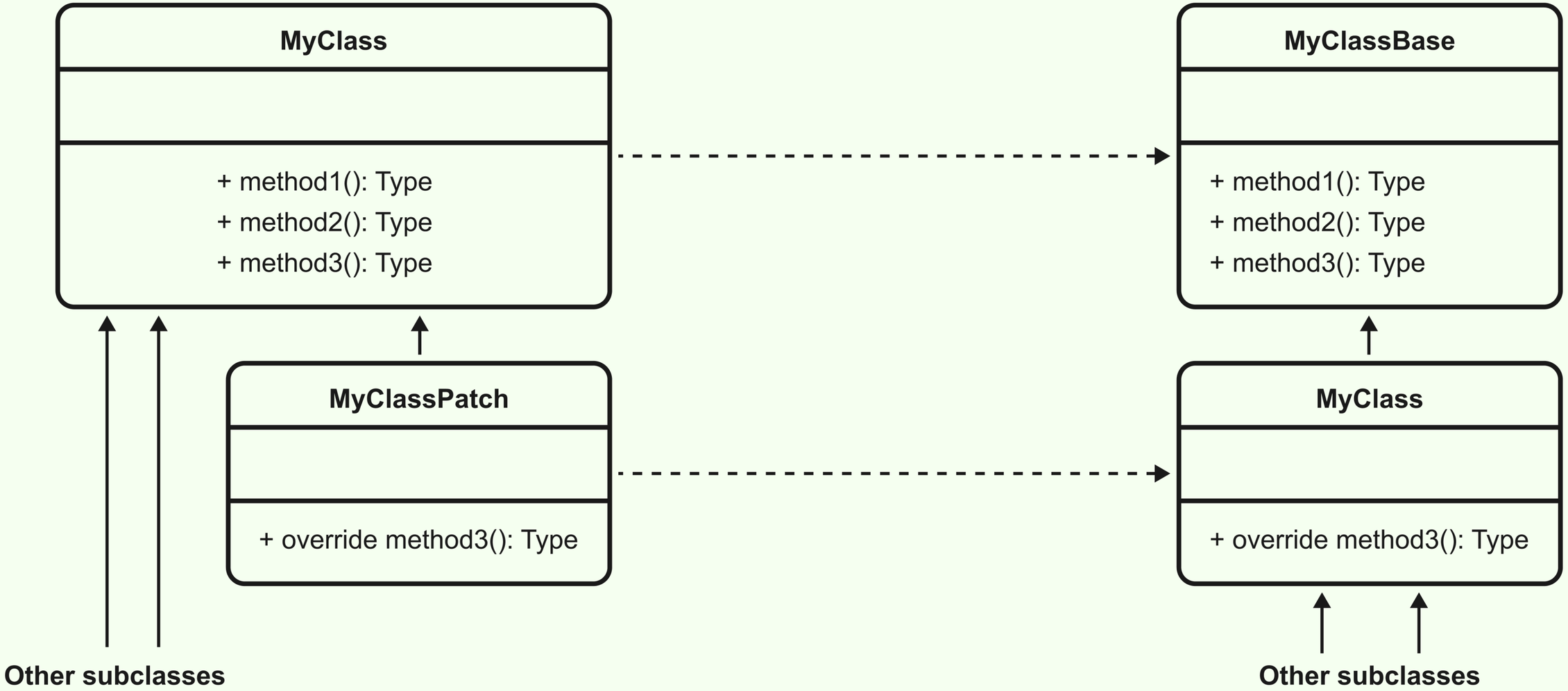
Способ 2. Реализация



Способ 2. Реализация



Способ 2. Реализация



```java
package com.jokerconf.patching.example;

no usages
@OriginClass("com.jokerconf.patching.example.Dog")
@Subclass
public class PatchedDog extends Dog {
    1 usage
    @Override
    public void woof() {
        System.out.println("MEOW");
    }
}
```

```java
package com.jokerconf.patching.example;

no usages
public class DogBase {

    no usages
    public DogBase() {}

    1 usage
    private void eat() { System.out.println("Eat a bone"); }

    1 usage
    private void sleep() { System.out.println("HRRR!!! HRRRR!!!"); }

    1 usage
    private boolean isAlive() { return true; }

    public void live() {...}

    no usages
    public void woof() { System.out.println("WOOF"); }
}
```

```java
package com.jokerconf.patching.example;

no usages
@OriginClass("com.jokerconf.patching.example.Dog")
@Subclass
public class Dog extends DogBase {
    no usages
    public Dog() {
    }

    1 usage
    public void woof() { System.out.println("MEOW"); }
}
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);

for (Object annotation : patchCtClass.getAnnotations()) {
    if (annotation instanceof Subclass) {
        String baseClassName = originClassName + "Base";
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);

for (Object annotation : patchCtClass.getAnnotations()) {
    if (annotation instanceof Subclass) {
        String baseClassName = originClassName + "Base";
        patchCf.setSuperclass(baseClassName);
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);

for (Object annotation : patchCtClass.getAnnotations()) {
    if (annotation instanceof Subclass) {
        String baseClassName = originClassName + "Base";
        patchCf.setSuperclass(baseClassName);
        patchCf.renameClass(originClassName, baseClassName);
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);

for (Object annotation : patchCtClass.getAnnotations()) {
    if (annotation instanceof Subclass) {
        String baseClassName = originClassName + "Base";
        patchCf.setSuperclass(baseClassName);
        patchCf.renameClass(originClassName, baseClassName);

        ClassFile baseCf = loadClassFile(baseClassBytes);
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);

for (Object annotation : patchCtClass.getAnnotations()) {
    if (annotation instanceof Subclass) {
        String baseClassName = originClassName + "Base";
        patchCf.setSuperclass(baseClassName);
        patchCf.renameClass(originClassName, baseClassName);

        ClassFile baseCf = loadClassFile(baseClassBytes);
        baseCf.renameClass(originClassName, baseClassName);
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);

for (Object annotation : patchCtClass.getAnnotations()) {
    if (annotation instanceof Subclass) {
        String baseClassName = originClassName + "Base";
        patchCf.setSuperclass(baseClassName);
        patchCf.renameClass(originClassName, baseClassName);

        ClassFile baseCf = loadClassFile(baseClassBytes);
        baseCf.renameClass(originClassName, baseClassName);
        ClassPool.getDefault().makeClass(baseCf).toClass(loader, domain: null);
    }
}
```

```
ClassFile patchCf = loadClassFile(toPatchClassName(className));
CtClass patchCtClass = ClassPool.getDefault().makeClass(patchCf);
String originClassName = getOriginClass(patchCtClass);

for (Object annotation : patchCtClass.getAnnotations()) {
    if (annotation instanceof Subclass) {
        String baseClassName = originClassName + "Base";
        patchCf.setSuperclass(baseClassName);
        patchCf.renameClass(originClassName, baseClassName);

        ClassFile baseCf = loadClassFile(baseClassBytes);
        baseCf.renameClass(originClassName, baseClassName);
        ClassPool.getDefault().makeClass(baseCf).toClass(loader, domain: null);
    }
}
patchCf.renameClass(classMappings);
byte[] patchedBytes = serializeClass(patchCf);
```

Способ 2. Преимущества и недостатки

Относительная простота
реализации

Способ 2. Преимущества и недостатки

Относительная простота реализации

Изменяем только методы, в которые нужно внести правки

Способ 2. Преимущества и недостатки

Относительная простота реализации

Изменяем только методы, в которые нужно внести правки

Подходит только для публичных нефинальных методов

Способ 2. Преимущества и недостатки

Относительная простота реализации

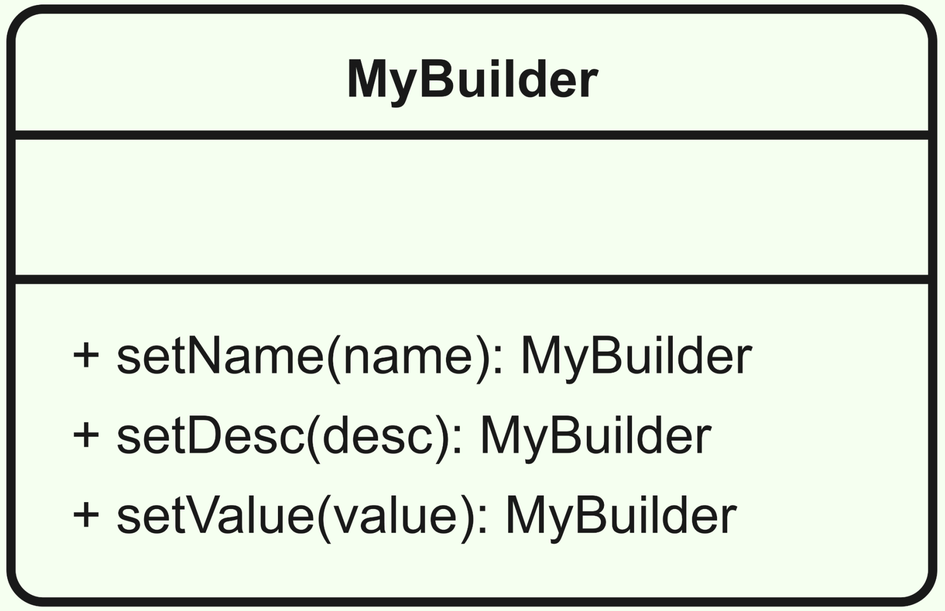
Изменяем только методы, в которые нужно внести правки

Подходит только для публичных нефинальных методов

Не подходит для паттерна builder и случаев, когда методы возвращают экземпляр изменяемого класса

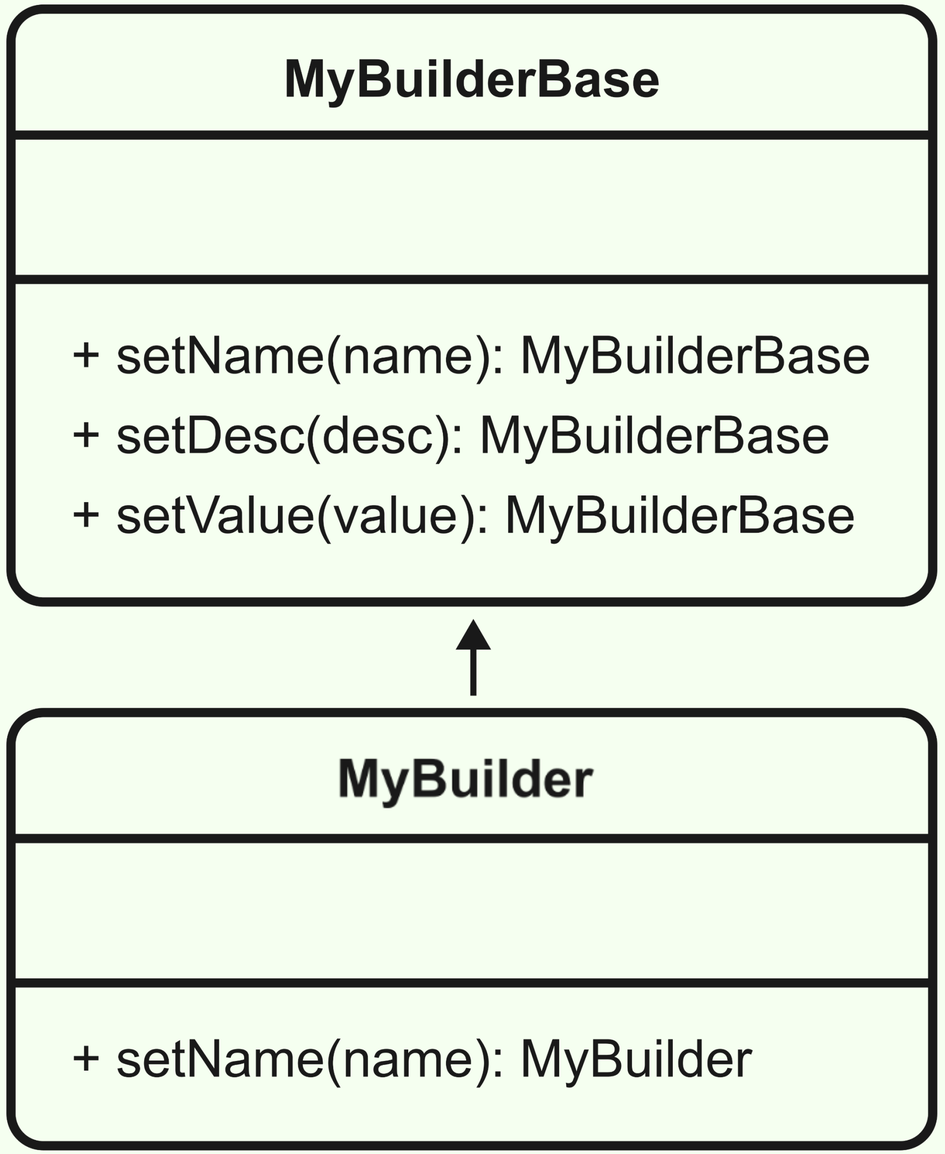
Способ 2: Проблемы

```
MyBuilder builder = new MyBuilder();
builder = builder.setName (name);
builder = builder.setValue (value);
…
```

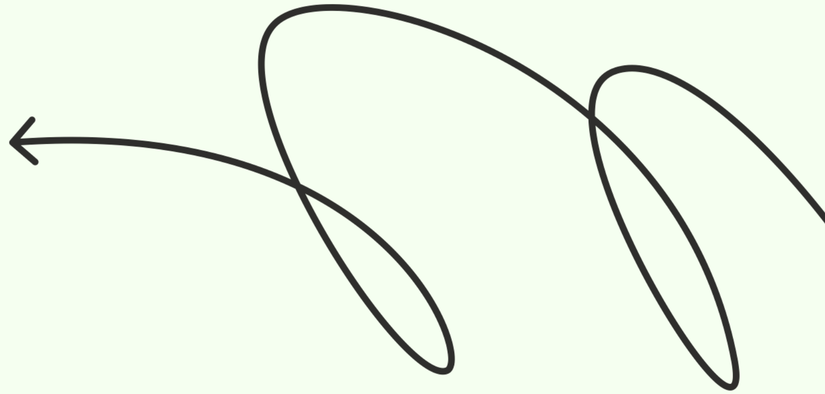


Способ 2: Проблемы

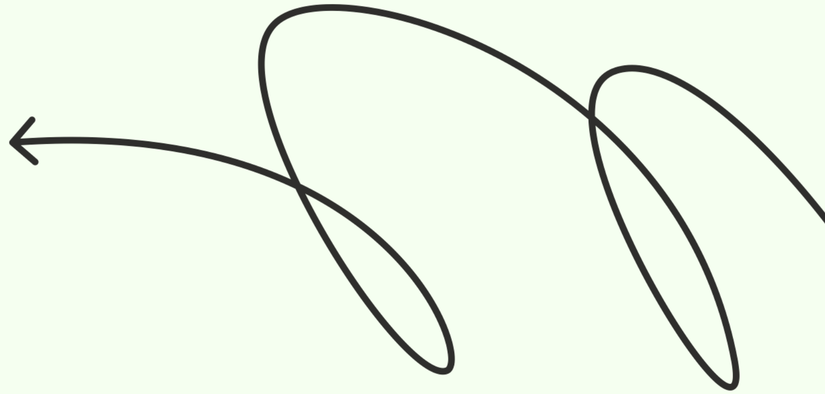
```
MyBuilder builder = new MyBuilder();
builder = builder.setName (name);
builder = builder.setValue (value);

java.lang.NoSuchMethodError:
MyBuilder MyBuilder.setValue (value)
…
```



Способ 2.
Применимость



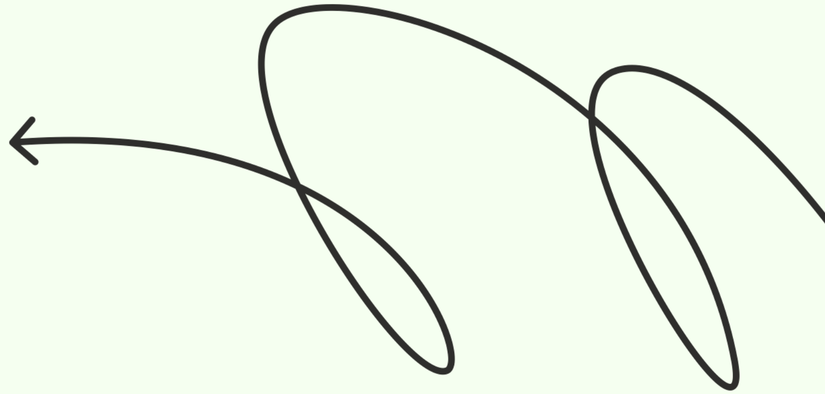
Способ 2. Применимость



1

Подходит для классов с публичными методами

Способ 2. Применимость



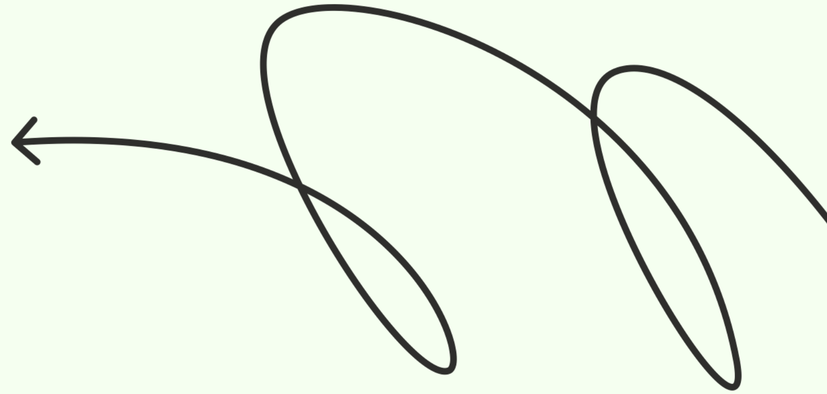
1

Подходит для классов с публичными методами

2

Ни один из методов не должен возвращать экземпляр модифи-цируемого класса

Способ 2. Применимость



1

Подходит для классов с публичными методами

2

Ни один из методов не должен возвращать экземпляр модифи-цируемого класса

3

Может встраиваться в иерархию наследования

```java
package com.jokerconf.patching.example;

3 usages  1 inheritor
public class Dog {
    1 usage
    private void eat() { System.out.println("Eat a bone"); }
    1 usage
    private void sleep() { System.out.println("HRRR!!! HRRRR!!!"); }
    1 usage
    private boolean isAlive() { return true; }
    no usages
    public void live() {
        while (isAlive()) {
            eat();
            sleep();
        }
    }

    1 usage  1 override
    public void woof() {
        System.out.println("WOOF");
    }
}
```

Способ 3. Декорирование методов

Переименовываем метод
в оригинальном классе,
и добавляем свою реализацию
метода

Способ 3. Реализация

Способ 3. Реализация

- Создаём в патче декораторы
  для нужных методов

Способ 3. Реализация

- Создаём в патче декораторы для нужных методов
- Помечаем класс патча и методы-декораторы аннотациями

Способ 3. Реализация

- Создаём в патче декораторы для нужных методов
- Помечаем класс патча и методы-декораторы аннотациями
- Переименовываем соответствующие методы в оригинальном классе, добавляя им префикс

Способ 3. Реализация

- Создаём в патче декораторы для нужных методов
- Помечаем класс патча и методы-декораторы аннотациями
- Переименовываем соответствующие методы в оригинальном классе, добавляя им префикс
- Для вызова оригинальных методов в патче необходимо создать заглушки с названием префикс+имя метода

Способ 3. Реализация

- Создаём в патче декораторы для нужных методов
- Помечаем класс патча и методы-декораторы аннотациями
- Переименовываем соответствующие методы в оригинальном классе, добавляя им префикс
- Для вызова оригинальных методов в патче необходимо создать заглушки с названием префикс+имя метода
- Переносим методы из патча в оригинальный класс

```java
package com.jokerconf.patching.example;

no usages
@OriginClass("com.jokerconf.patching.example.Dog")
@Decorate
public class DogDecorators {
    no usages
    @DecoratedMethod
    private void eat() {
        System.out.println("Wash paws");
        __eat();
        System.out.print("Clean the dishes");
    }
    no usages
    @DecoratedMethod
    private void sleep() {
        System.out.println("sssss....");
    }

    1 usage
    private void __eat() {}
}
```

```java
package com.jokerconf.patching.example;

3 usages  1 inheritor
public class Dog {
    2 usages
    public Dog() {
    }

    1 usage
    private void __eat() { System.out.println("Eat a bone"); }

    no usages
    private void __sleep() { System.out.println
    ("HRRR!!! HRRRR!!!"); }

    1 usage
    private boolean isAlive() { return true; }

    no usages  1 override
    public void live() {
        while (this.isAlive()) {
            this.eat();
            this.sleep();
        }
    }
```

```java
    1 usage  1 override
    public void woof() {
        System.out.println("WOOF");
    }

    1 usage
    private void eat() {
        System.out.println("Wash paws");
        this.__eat();
        System.out.print("Clean the dishes");
    }

    1 usage
    private void sleep() {
        System.out.println("sssss....");
    }
}
```

SparkPatchAgent

clck.ru/3Cy5DB

Способ 3. Преимущества и недостатки

Позволяет декорировать практически любой метод оригинального класса

Способ 3. Преимущества и недостатки

Позволяет декорировать
практически любой метод
оригинального класса

В том числе и статические,
как публичные,
так и приватные

Способ 3. Преимущества и недостатки

Позволяет декорировать практически любой метод оригинального класса

В том числе и статические, как публичные, так и приватные

Можем добавить новый функционал либо в начало, либо в конец, но не в середину метода

Способ 3. Преимущества и недостатки

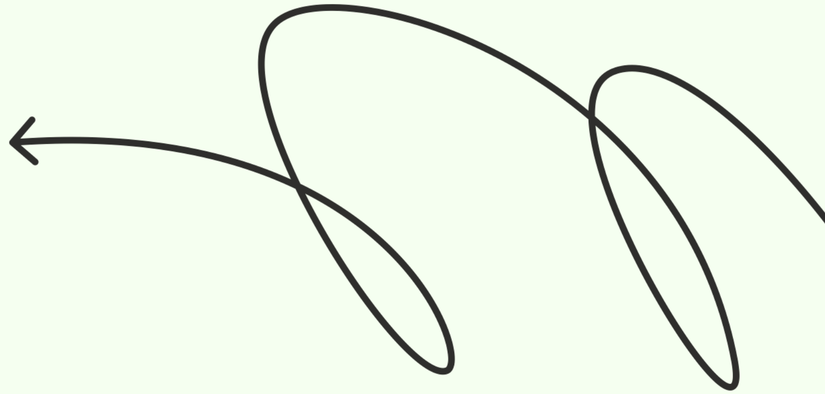
Позволяет декорировать практически любой метод оригинального класса

В том числе и статические, как публичные, так и приватные

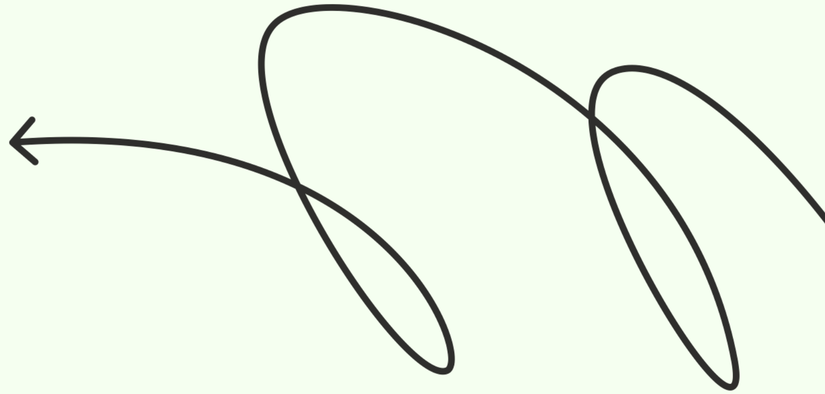
Можем добавить новый функционал либо в начало, либо в конец, но не в середину метода

Нельзя использовать функциональные конструкции внутри декоратора

Способ 3. Применимость



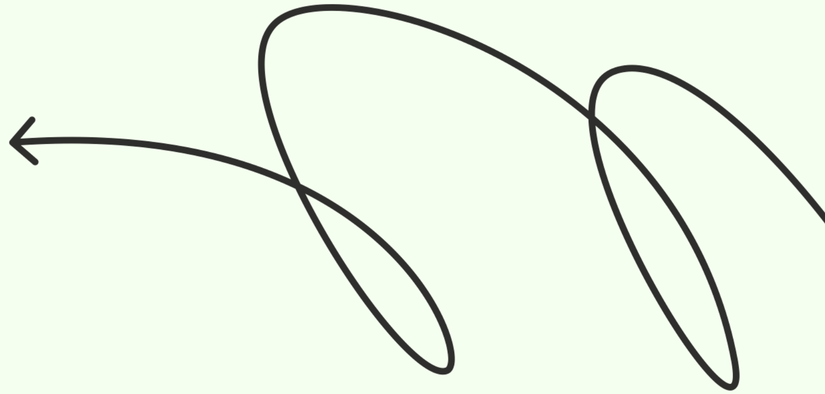
Способ 3. Применимость



1

Для модификации приватных методов

Способ 3. Применимость



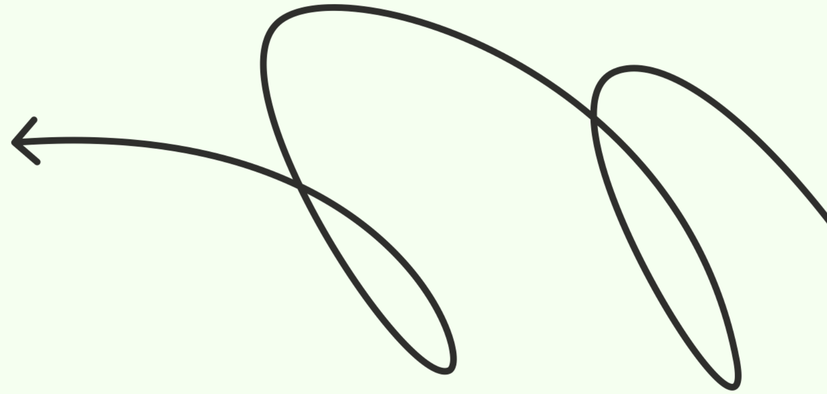
1

Для модификации приватных методов

2

Для модификации статических методов

Способ 3. Применимость



1

Для модификации приватных методов

2

Для модификации статических методов

3

Когда предыдущие два способа не подошли

Например, паттерн builder

Способ 4. Манипуляции на уровне байткода

Использование низкоуровнего API для манипуляции непосредственно инструкциями байткода

Способ 4. Реализация

</>

Индивидуальный код
для каждого патча

```
// Set the cluster manager
val clusterManager: Int = args.master match {
  case "yarn" => YARN
  case m if m.startsWith("spark") => STANDALONE
  case m if m.startsWith("mesos") => MESOS
  case m if m.startsWith("k8s") => KUBERNETES
  case m if m.startsWith("local") => LOCAL
  case _ =>
    error("Master must either be yarn or start with spark, mesos, k8s, or local")
    -1
}
```

```
// Set the cluster manager
val clusterManager: Int = args.master match {
  case "yarn" => YARN
  case m if m.startsWith("spark") => STANDALONE
  case m if m.startsWith("mesos") => MESOS
  case m if m.startsWith("k8s") => KUBERNETES
  case m if m.startsWith("local") => LOCAL
  case _ =>
    error("Master must either be yarn or start with spark, mesos, k8s, or local")
    -1
}
```

```
case m if m.startsWith("ytsaurus") => YTSAURUS
```

```
String var13 = args.master();
int var7;
if ("yarn".equals(var13)) {
    var7 = .MODULE$.org$apache$spark$deploy$SparkSubmit$$YARN();
} else if (var13.startsWith("spark")) {
            ...
} else if (var13.startsWith("local")) {
    var7 = .MODULE$.org$apache$spark$deploy$SparkSubmit$$LOCAL();
} else {
    this.error("Master must either be yarn or start with spark, mesos, k8s, or local");
    var7 = -1;
}
```

javap -p -c SparkSubmit.class

```
139: aload          13
141: ldc_w          #430  // String local
144: invokevirtual #415
  // Method java.lang.String.startsWith(String)
147: ifeq           161
150: getstatic      #43
  // Field org.apache.spark.deploy.SparkSubmit$.MODULE$
153: invokevirtual #433
  // Method org$apache$spark$deploy$SparkSubmit$$LOCAL()
156: istore         7
158: goto           177
161: goto           164
```

SparkSubmitSpyt

clck.ru/3Cy525

Добавленный байткод

```
164: aload          13
166: ldc_w          #2462     // String ytsaurus
169: invokevirtual #415
// Method java/lang/String.startsWith:(Ljava/lang/String;)Z
172: ifeq           186
175: ldc_w          #2463     // int 32
178: nop
179: nop
180: nop
181: istore         7
183: goto           202
186: goto           189
```

```
String var13 = args.master();
int var7;
if ("yarn".equals(var13)) {
    var7 = .MODULE$.org$apache$spark$deploy$SparkSubmit$$YARN();
} else if (var13.startsWith("spark")) {
    var7 = .MODULE$.org$apache$spark$deploy$SparkSubmit$$STANDALONE();
} else if (var13.startsWith("mesos")) {
    var7 = .MODULE$.org$apache$spark$deploy$SparkSubmit$$MESOS();
} else if (var13.startsWith("k8s")) {
    var7 = .MODULE$.org$apache$spark$deploy$SparkSubmit$$KUBERNETES();
} else if (var13.startsWith("local")) {
    var7 = .MODULE$.org$apache$spark$deploy$SparkSubmit$$LOCAL();
} else if (var13.startsWith("ytsaurus")) {
    var7 = 32;
} else {
    this.error("Master must either be yarn or start with spark, mesos, k8s, ytsaurus or local");
    var7 = -1;
}
```

Способ 4. Преимущества и недостатки

Нет ограничений
в вариантах модификации
исходного класса

Способ 4. Преимущества и недостатки

Нет ограничений
в вариантах модификации
исходного класса

Очень высокая
сложность реализации
и отладки

Способ 4.
Применимость

Когда предыдущие три способа
нельзя применить по тем
или иным причинам

Runtime patching при сборке и тестировании

Сборка javaagent (sbt)

```
lazy val `spark-patch` = (project in file("spark-patch"))
  .settings(
    libraryDependencies ++= spark ++ livy,
    Compile / packageBin / packageOptions +=
      Package.ManifestAttributes(
        "PreMain-Class" -> "tech.ytsaurus.spyt.patch.SparkPatchAgent"
      )
  )
```

Конфигурация javaagent для тестов

```
lazy val javaAgents = Def.task {
  Seq(ResolvedAgent(
    AgentModule(
      "spark-patch", null,
      AgentScope(test = true, dist = false),
      ""
    ),
    (`spark-patch` / Compile / packageBin).value
  ))
}
```

Добавление javaagent в конфигурацию проекта

```
lazy val `data-source` = (project in file("data-source"))
  .dependsOn(`spark-patch` % Provided)
  .enablePlugins(JavaAgent)
  .settings(
    resolvedJavaAgents := javaAgents.value
  )
```

Запуск автотестов

sbt clean data-source/test

```
$ ps auxww | grep javaagent


...


java -javaagent:spark-patch.jar
-classpath data-source/target/scala-2.12/test-classes:
data-source/target/scala-2.12/classes:
...
sbt.ForkMain 46875
```

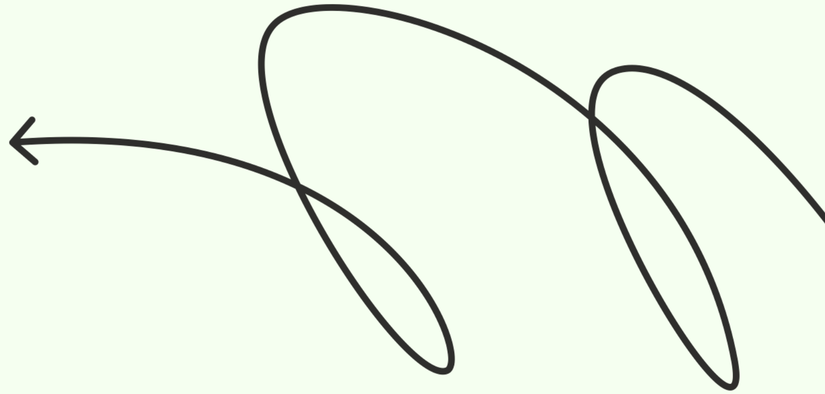
Подведение итогов

Надёжно ли это?

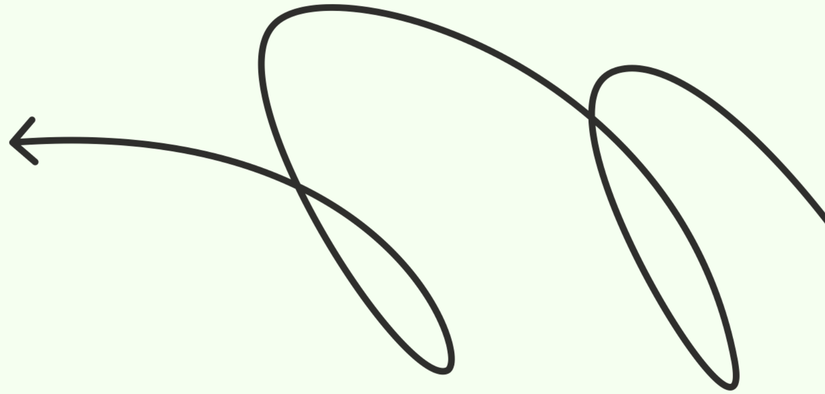
Надёжно ли это?

На первый взгляд кажется что нет,
но при соответствующем покрытии
тестами сомнения исчезают

Удобно ли это?



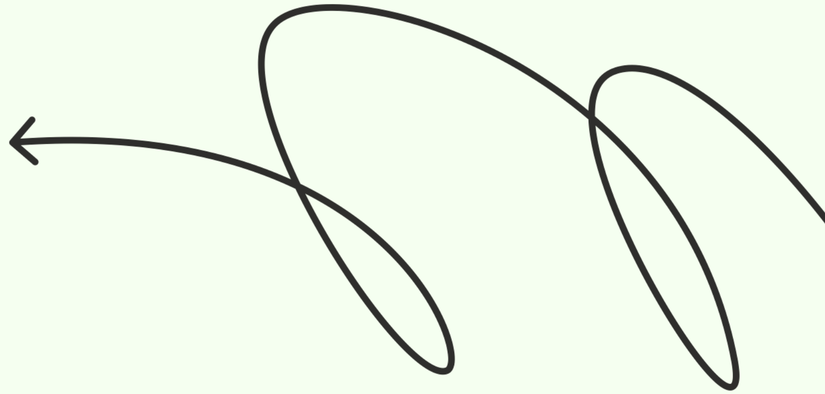
Удобно ли это?



1

Первые три способа
относительно простые
в интерфейсе
и реализации

Удобно ли это?



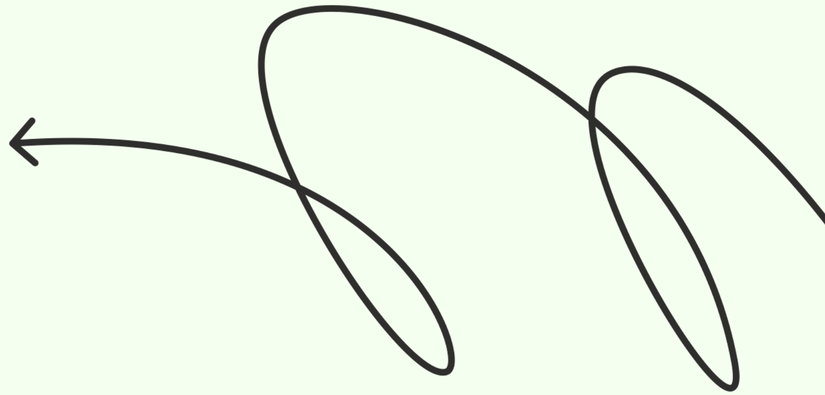
1

Первые три способа относительно простые в интерфейсе и реализации

2

Позволяют делать модификации без непосредственно манипулирования байткодом на низком уровне

Удобно ли это?



1

Первые три способа относительно простые в интерфейсе и реализации

2

Позволяют делать модификации без непосредственно манипулирования байткодом на низком уровне

3

Байткод — это несложно)

Насколько вообще легитимно так делать?

Насколько вообще легитимно так делать?

- В идеальном мире в этом нет необходимости

Насколько вообще легитимно так делать?

- В идеальном мире в этом нет необходимости
- Но мы живём в реальном мире, с неидеальным кодом, спроектированном по не всегда идеальной архитектуре, поэтому приходится)

Насколько вообще легитимно так делать?

- В идеальном мире в этом нет необходимости
- Но мы живём в реальном мире, с неидеальным кодом, спроектированном по не всегда идеальной архитектуре, поэтому приходится)
- Наличие большого количества библиотек для манипуляции байткодом говорит о высокой потребности в этом

Ссылки

Мы на GitHub:
clck.ru/3Cy9yz

Реализация runtime patching framework:
clck.ru/3CyA4y

Опенсорс-чат:
t.me/ytsaurus_ru

YTsaurus

Вопросы?

Александр Токарев, Яндекс,
TeamLead группы разработки SPYT
powered by Apache Spark