Протоколы в языке программирования Kotlin

Сташевский Леонид Евгеньевич
научный руководитель: Д.С. Жарков

СПб АУ НОЦНТ РАН

12 июня 2017 г.

Проблема: отсутствие общего интерфейса

Классы из разных библиотек:

- LazyString
- ImmutableString
- CustomString

Как написать метод

```
1 fun reverse(str: ???):??? {
2 }
```

для всех строк?

Проблема: интерфейс

Интерфейсы используются для указания набора методов при объявлении класса:

- Comparable
- Collection
- MouseListener
- KeyListener

Сложно изменить библиотечные классы, если необходим новый интерфейс.

Основные определения

Структурная типизация - введение отношения подтипизации на основе содержимого типов

Номинальная типизация - способ задания отношения подтипизации, при котором один тип является подтипом другого если это указано явно

Протокол - структурный тип в языке с номинальной типизацией.

Возможное поведение

```
1  protocol interface Indexed<out T> {
2      fun get(i: Int): T
3  }
4
5  class String() {
6      ...
7      fun get(i: Int): Char {
8          ...
9      }
10 }
11
12 val obj: String = String(”Hello”)
13
14 fun <T> even(indexed: Indexed<T>) = ...
15
16 event(obj)
```

Цель и задачи

Цель работы: реализация протоколов в языке программирования Kotlin для платформы JVM.
Задачи:

- Рассмотреть существующие подходы
- Добавить способ задания и проверку типов
- Реализовать механизм вызова метода
- Разработать поиск метода для вызова
- Сравнить результаты с реализациями в других языках

Существующие подходы для JVM

Scala - стирание типа

- Проверка типов реализована в компиляторе
- Тип протокола не генерируется
- Вызов методов с помощью рефлексии:

```
1 a.getClass().getMethod.(”foo”, ...).invoke(a, ...)
```

Whiteoak - генерация обёртки

- протокол - абстрактный класс
- приведение типа - генерация наследника

Подход к решению

Этапы разработки:

- Объявление

```
protocol interface Foo { ... }
```

- Проверка типов

```
val x: Foo = object
```

- Вызов метода

```
x.call(...)
```

Ограничения:

- сохранение идентичности объектов
- отсутствие генерации кода во время выполнения
- сохранение раздельной компиляции

Объявление и проверка типов

Для объявления в язык было добавлено ключевое слово *protocol*.
Тип A является подтипом протокола B если:

- каждое именованное поле A содержится в B
- тип каждого поля в A является подтипом каждого поля B
- типы аргументов методов в точности совпадают
- возвращаемый тип может быть более точным

Вызов метода

Вызов метода

- механизм рефлексии
- библиотека вызовов

Библиотека вызовов

- добавлена в JDK 7
- проверка типов происходит при поиске метода

Для измерения времени вызова функции был написан прототип. Ограничением прототипа - поддержка функций только с одним аргументом.

Поиск метода для вызова

```
1  protocol interface Foo<T> {
2      fun bar(arg: T)
3  }
4
5  class X {
6      fun bar(arg: Int)
7      fun bar(arg: Int?)
8  }
9
10 val first: Foo<Int> = X()
11 val second: Foo<Int?> = X()
```

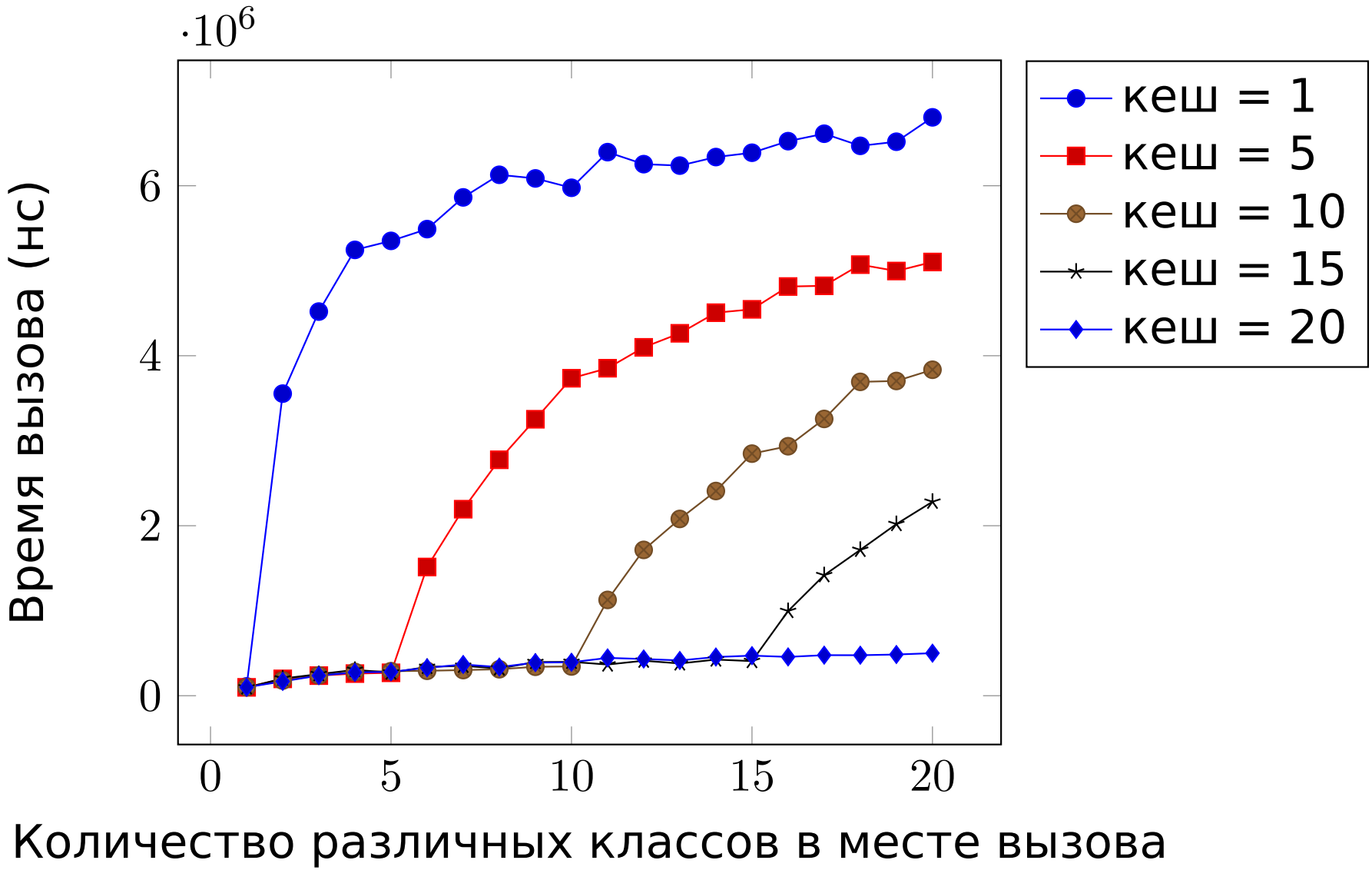
```
1 bar:(I)V
2 bar:(java/lang/Integer)V
```

```
1 (I)V: 0, (Integer)V: 1
2 (I)V: 1, (Integer)V: 0
```

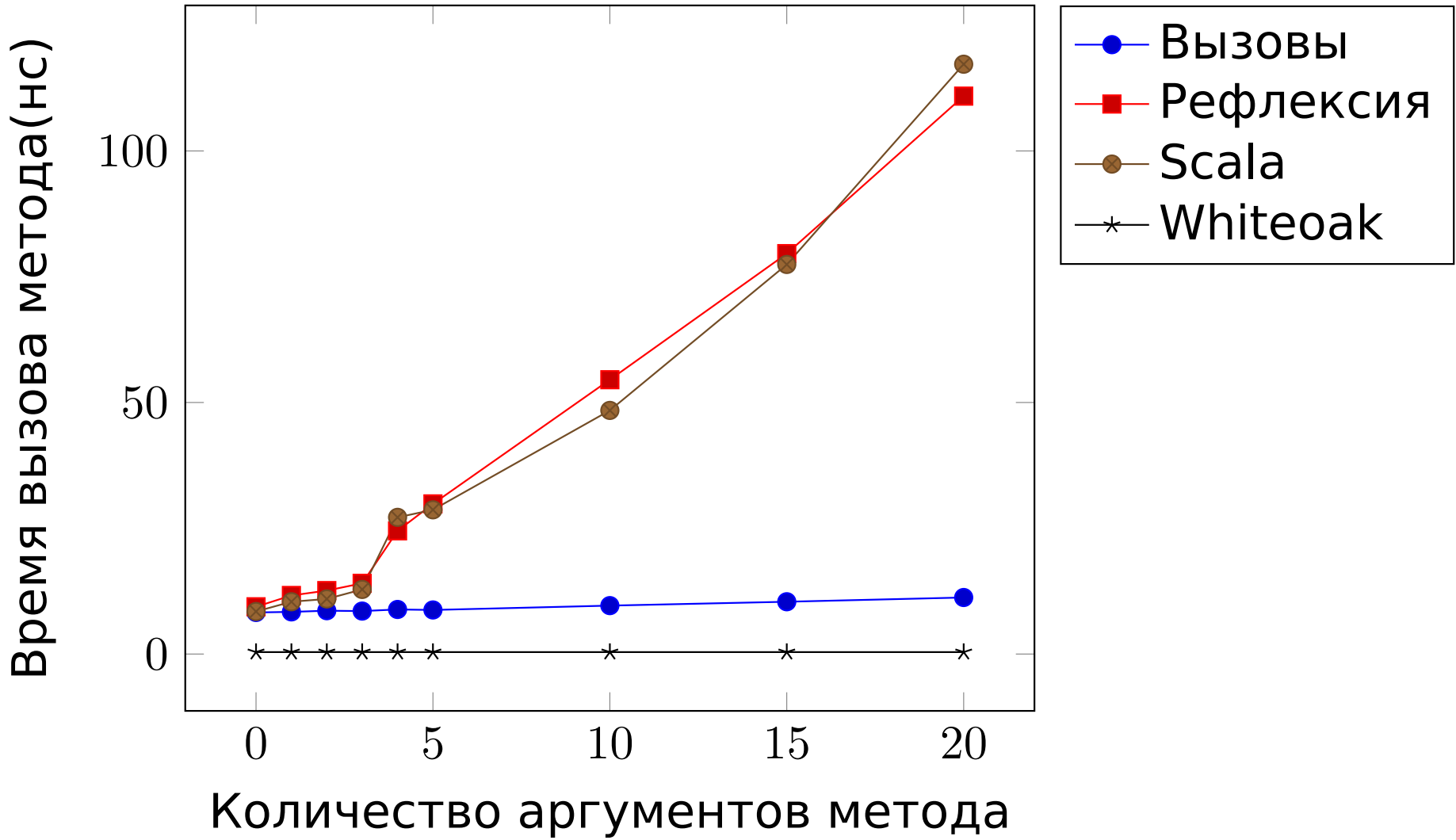
Метрика для аргументов:

- 0 - совпадение типов
- 1 - присваиваемый тип можно присвоить в родительский
- +1 за отличие ссылочный/примитивный тип

Кеширование



Сравнение реализаций



Результаты

В работе была получена реализация протоколов в языке Kotlin:

- введено ключевое слово protocol
- реализовано 2 способа для вызова метода: с помощью рефлексии и библиотеки вызовов
- использование алгоритма поиска метода во время выполнения позволило расширить возможности использования протоколов с параметрическими типами
- выявлена зависимость времени вызова от числа аргументов при вызове с помощью рефлексии
- для уменьшения времени вызова было добавлено кеширование

Результаты работы доступны в репозитории по ссылке `https://github.com/e5l/ProtocolsGenerator`