# Поддержка постепенной типизации в языке Kotlin

Алексей Степанов

Научный руководитель: Андрей Бреслав

САНКТ-ПЕТЕРБУРГСКИЙ АКАДЕМИЧЕСКИЙ УНИВЕРСИТЕТ

27 апреля 2017 г.

# Введение

Типизация в языках программирования

- Статическая типизация
- Динамическая типизация
- Постепенная типизация

# Введение

## Постепенная типизация

Постепенная типизация - система типов, в которой часть переменных и выражений может быть типизированна, и их корректность проверяется в момент компиляции, а часть может быть не типизированна, и об ошибках в них мы узнаем во время исполнения.

## Преимущества

- Функция eval.
- Предметно-ориентированные языки (Gradle).
- Объектная модель документа.

## Удобный доступ к полям и методам

```
dynamic x = dom.html.body.tables.main.tr.td;
```

## Почему не Object?

```
Scriptobj.SetProperty("Cnt",((int)GetProperty("Cnt"))+1);
scriptobj.Cnt += 1;
```

## Варианты постепенной типизации

| От динамической типизации | От статической типизации |
|---|---|
| Python | C# |
| JavaScript | Scala |
| Groovy | Kotlin (JVM)? |

# Предварительная обстановка

## Kotlin

- Поддержка dynamic в Kotlin для JavaScript.
- В JVM dynamic не поддерживается.

# Цель и задачи

## Цель

Для обеспечения поддержки постепенной типизации в языке Kotlin, возникает необходимость реализовать поддержку типа dynamic при компиляции на Java платформе.

## Задачи

- Определить семантику динамических операций.
- Выработать правила разрешения перегрузок.
- Реализовать поддержку динамических вызовов в компиляторе языка Kotlin под JVM.
- Оценить производительность.

## Kotlin

- Присваивание в динамическую переменную.
- Присваивание динамической переменной в типизированную.
- Вызов метода на динамической переменной.
- Вызов метода на типизированной переменной.
- Запрос поля у динамической переменной.
- Запрос динамического поля у не динамической переменной.

## Критерии определения правил выбора перегрузок

- Предсказуемость.
- Производительность.
- Обратная совместимость при стирании типа в статически типизированном коде.
- Схожесть работы нетипизированного кода с типизированным.

# Реализация

## Поддержка в компиляторе

- invokedynamic.
- MethodHandles.
- Отсутствующие во время выполнения методы.
- Обращение к полям или свойствам.

  obj.bar
- Выполнение составного присваивания.

  a += b
- Вызываемые объекты

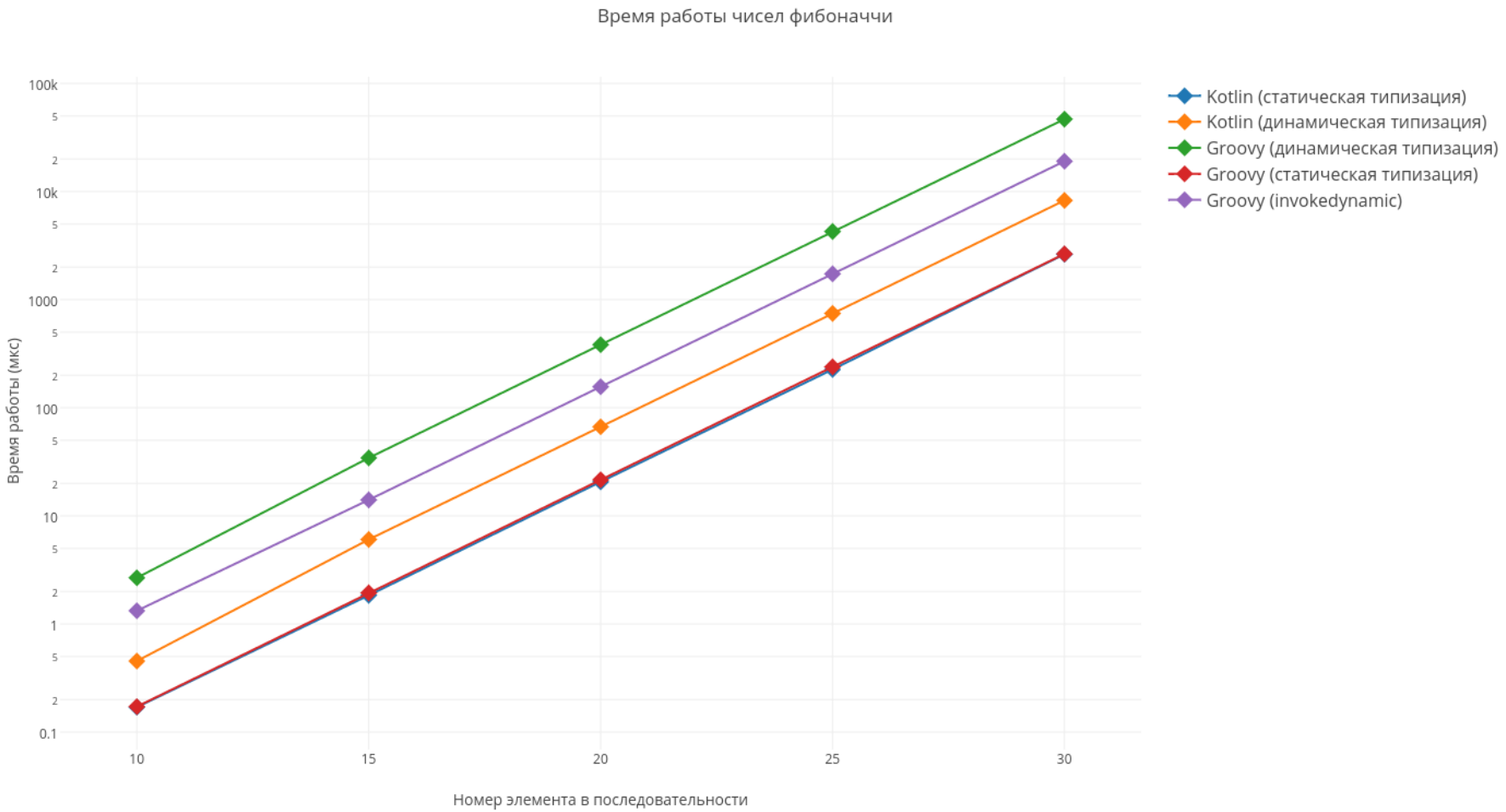
  obj.foo(args)

## Сравнение производительности

- Были написаны тесты с использованием JMH.
- Быстрее чем Groovy (обычный) до 14 раз.
- Быстрее чем Groovy (invokedynamic) до 6 раз.
- В некоторых тестах наблюдается ухудшение производительности обладающее хуже чем линейной зависимостью.

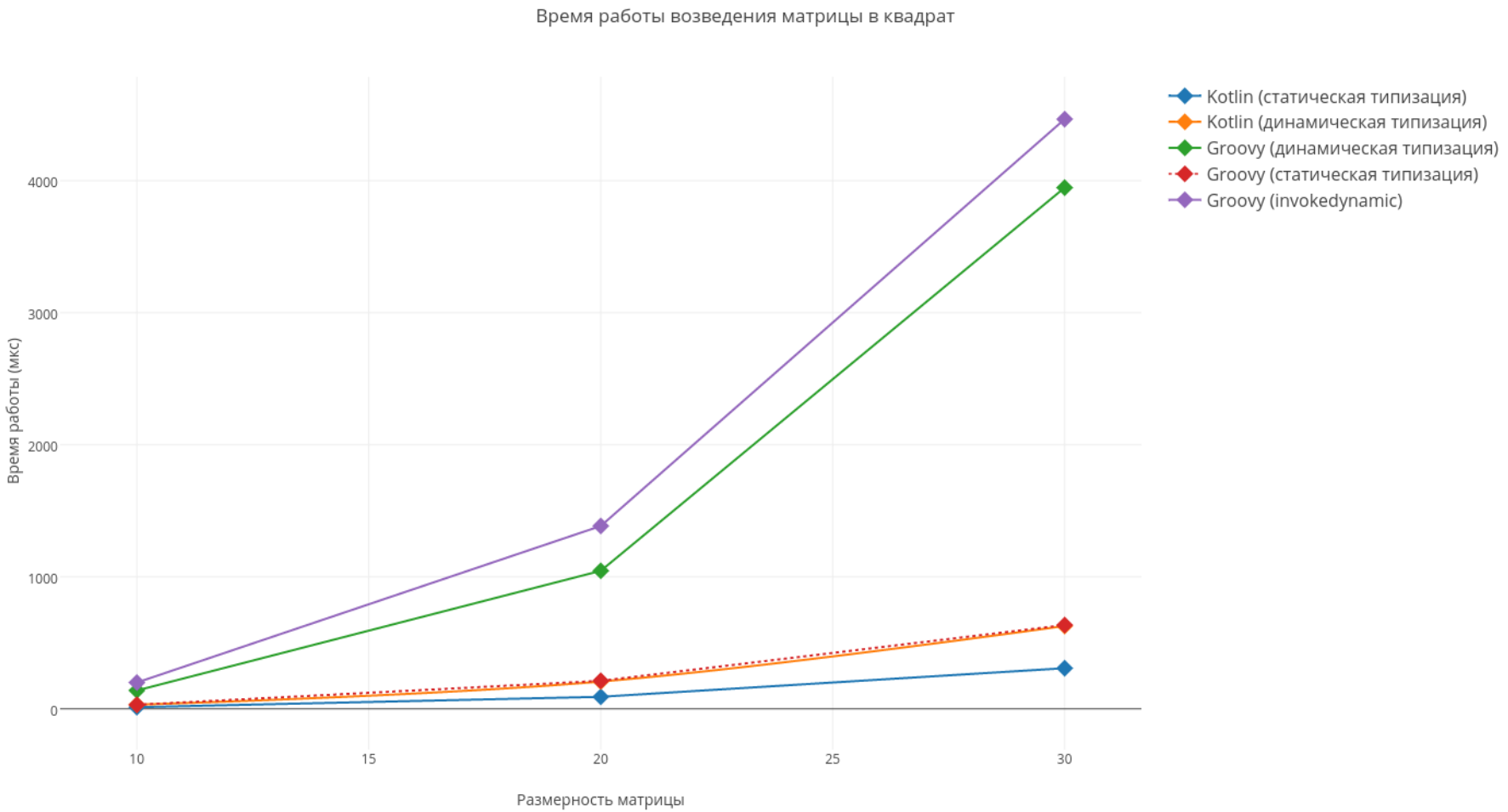
# Реализация

## Сравнение производительности



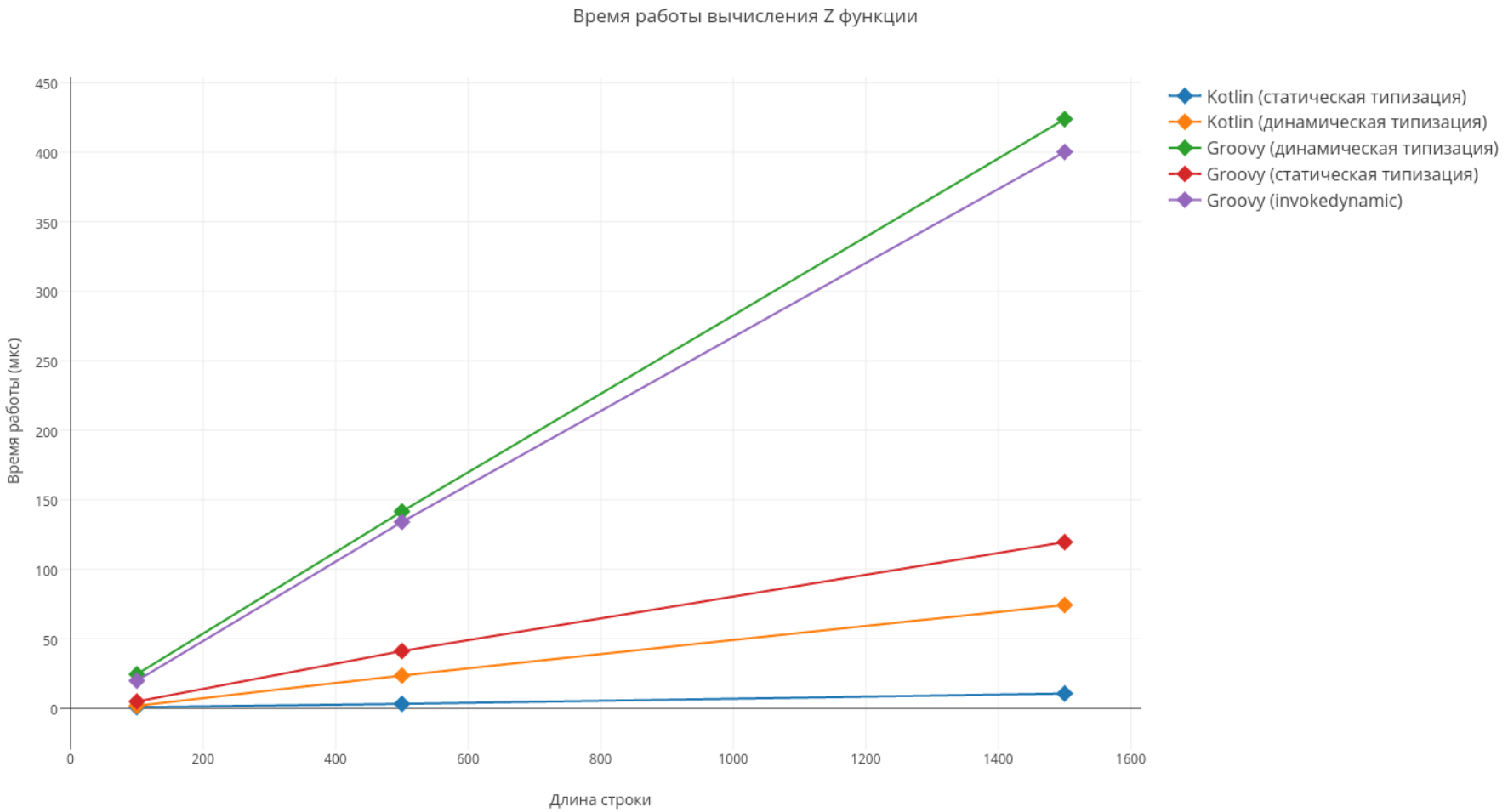
# Реализация

## Сравнение производительности



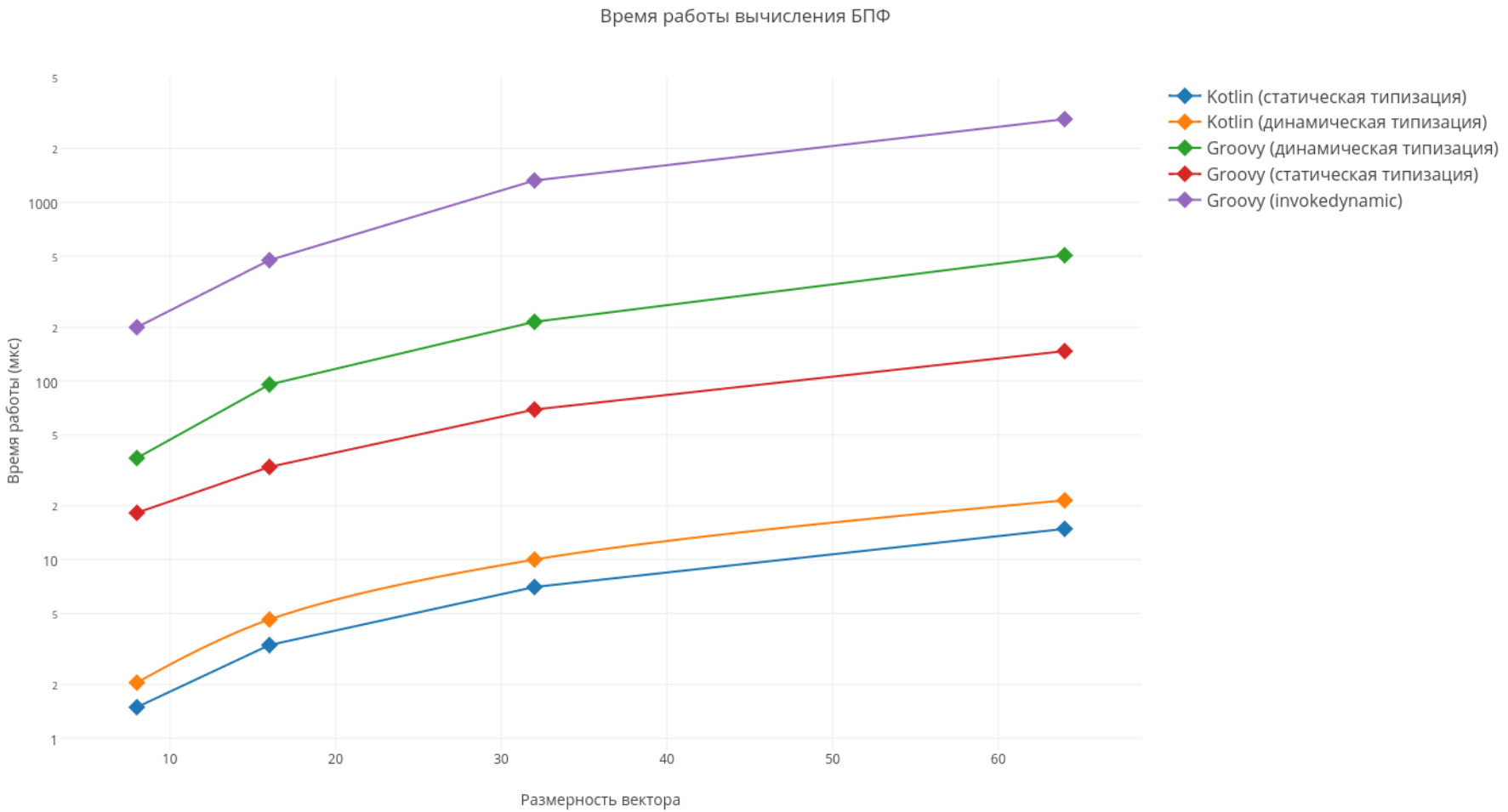
# Реализация

## Сравнение производительности



# Реализация

## Сравнение производительности



# Вопросы?

# Реализация

Про два типа $\phi$ и $\psi$, будем говорить:

- Что $\phi$ и $\psi$ эквивалентные типы, если они в точности совпадают.
- Что $\phi$ и $\psi$ похожие типы, если один из них, является упакованной версией другого.
- Что $\phi$ лучший тип чем $\psi$, если $\phi$ реализует интерфейс $\psi$, или является его потомком.
- Во всех других случаях, будем говорить, что $\phi$ худший тип, чем $\psi$.

# Реализация

## Определение перегрузок

1. Если f совпадает с g, то он более специфичный чем g.
2. Если f является методом-помощником, то он более специфичный чем g.
3. Если g является методом-помощником, то он менее специфичный чем f.
4. Если возвращаемый тип f лучше, чем тип g, то f более специфичный.
5. Если возвращаемый тип f хуже, чем тип g, то f менее специфичный.
6. Если у f существует такой индекс i, что i-ый параметр f хуже чем i-ый параметр g, то f менее специфичный чем g.
7. Если у f существует такой индекс i, что i-ый параметр f лучше чем i-ый параметр g, то f более специфичный чем g.
8. Если g является методом с переменным числом аргументов, а f — нет, то f более специфичный.
9. Во всех остальных случаях, f менее специфичный.

## Алгоритм определения перегрузок методов

- Его имя совпадает с динамически вызванным методом.
- К его аргументам подходят аргументы времени выполнения у динамического метода.
- Он является более специфичный, чем все другие методы, которые удовлетворяют пунктам 1-2.