



RuStore

Классы типов —

возможное будущее Java, или как
Брайан Гётц предлагает развивать язык

Андрей Васильев
RuStore





Андрей
Васильев

RuStore

О себе

- Веду команду платформы бэкенда RuStore
- К JVM имею отношение с 2005 года
- Иногда преподаю в ВУЗах

Давайте знакомиться

Кто слышал
про классы
типов в любом
другом языке?

Давайте знакомиться

Кто слышал
(знаком)
про классы
типов в любом
другом языке?

Кто знаком
со Scala
или Haskell?

Давайте знакомиться

Кто слышал
(знаком)
про классы
типов в любом
другом языке?

Кто знаком
со Scala
или Haskell?

Кто считает,
что Java
«тормозит»?



1

Growing
the Java Language
by Brian Goetz

2

Другой инфы
очень мало

3

Мотивацию
лучше никто
не объяснит

1

Growing
the Java Language
by Brian Goetz

2

Другой инфы
очень мало

3

Мотивацию
лучше никто
не объяснит

4

Классы типов —
что за зверь такой
и зачем добавлять?

5

Как это будет
выглядеть (демо)?

6

Какие «дороги»
это открывает?

Развитие языка — дело сложное

01

Добавить
новую фичу
в язык просто,
удалить — **ПОЧТИ
НЕВОЗМОЖНО**

Развитие языка — дело сложное

01

Добавить
новую фичу
в язык просто,
удалить — **почти
невозможно**

02

Новая фича
должна
«дружить» с уже
существующими

Развитие языка — дело сложное

01

Добавить
новую фичу
в язык просто,
удалить — **почти
невозможно**

02

Новая фича
должна
«дружить» с уже
существующими

03

Хорошо когда
новая фича
получается
расширяемой

Развитие языка — дело сложное

01

Добавить
новую фичу
в язык просто,
удалить — **почти
невозможно**

02

Новая фича
должна
«дружить» с уже
существующими

03

Хорошо когда
новая фича
получается
расширяемой

04

Ещё лучше,
если новую фичу
смогут расширять
**обычные
пользователи**

Развитие языка — дело сложное

```
Iterable<String> strings = new BigList<>();  
  
for (var string : strings) {  
    System.out.println(string);  
}
```

Развитие языка — **дело сложное**

Литералы
коллекций

Project Coin,
Java 7

Только для
List, Set, Map

```
List<String> strings = ["a", "b", "c"];
```

Полиморфизм

Полиморфизм

Меньше

повторений
кода

Больше

возможностей
переиспользования

Больше

возможностей
для расширения

Полиморфизм — это хорошо

Меньше

повторений
кода

Больше

возможностей
переиспользования

Больше

возможностей
для расширения

Параметрический полиморфизм

Generics

Один раз для всех типов

```
interface List<E> {  
  
    int size();  
  
    boolean add(E e);  
  
    E get(int index);  
  
}
```

Полиморфизм подтипов

Динамическое
связывание – в runtime

```
class ArrayList<E> implements List<E>

List<String> list = getList("Wonderland");
```

Специальный (ad hoc) полиморфизм

Статическое связывание
– в compile time

Разные реализации для
разных типов

Небольшое кол-во
поддерживаемых типов

```
int plus(int x, int y)
```

```
double plus(double x, double y)
```

```
String plus(String x, String y)
```

Полиморфизм — это хорошо...

Проблемы Java- интерфейсов

Retroactive
modeling

```
class Person {  
    int age;  
    String name;  
}
```

Проблемы Java- интерфейсов

Retroactive
modeling

```
class Person {  
    int age;  
    String name;  
}
```

```
interface Printable<T> {  
    void print(T t, OutputStream out);  
}
```

Проблемы Java- интерфейсов

Retroactive
modeling

```
interface Printable<T> {  
    void print(T t, OutputStream out);  
}
```

```
class Person implements Printable<Person> {  
    int age;  
    String name;  
  
    void print(Person p, OutputStream out)  
}
```

Проблемы Java- интерфейсов

Retroactive
modeling

```
class String implements Printable<String> {  
    void print(String s, OutputStream out)  
}
```

Проблемы Java- интерфейсов

Множественные
реализации

```
interface Convertable<F, T> {  
    T to(F from);  
}
```

```
class MyNumber implements  
    ConvertableTo<MyNumber, Double>,  
    ConvertableTo<MyNumber, Long>  
{  
    public ? to(MyNumber from) {...}  
}
```

Проблемы Java- интерфейсов

Имена методов

```
interface AutoCloseable {  
    void close();  
}
```

```
try (var input = new FileInputStream(...)) {  
    ...  
}
```

Проблемы Java- интерфейсов

Имена методов

```
final var lock = new MySuperLock();  
...  
  
lock.lock();  
try (lock) {  
    ...  
}
```

Свет в конце тоннеля

Comparable – `x.compareTo(y)`

Comparator – `compare(x, y)`

Свет в конце тоннеля

1

Comparator – описывает
поведение «снаружи»

Свет в конце тоннеля

1

Comparator — описывает
поведение «снаружи»

2

Comparator — свидетель
(**witness**) знания о том,
как сравнивать объекты

Классы типов (Type classes)

- ④ 1989, Philip Wadler
- ④ Ad hoc полиморфизм
- ④ Первыми появились в Haskell
- ④ Поведение типа косвенно выражается через объект-свидетель - **witness**

Классы типов (Type classes)

Haskell

```
class Equal a where  
    eq :: a -> a -> Bool
```

Классы типов (Type classes)

Haskell

```
class Equal a where  
    eq :: a -> a -> Bool  
  
data Person = Person {  
    name :: String,  
    age  :: Int  
}
```

Классы типов (Type classes)

Haskell

instance можно объявить
в одном из двух мест:

- Equal.hs
- Person.hs

```
class Equal a where  
  
    eq :: a -> a -> Bool
```

```
data Person = Person {  
  
    name :: String,  
  
    age :: Int  
  
}
```

```
instance Equal Person where  
  
    eq p1 p2 = eq (age p1) (age p2)  
  
            && eq (name p1) (name p2)
```

Классы типов (Type classes)

Haskell

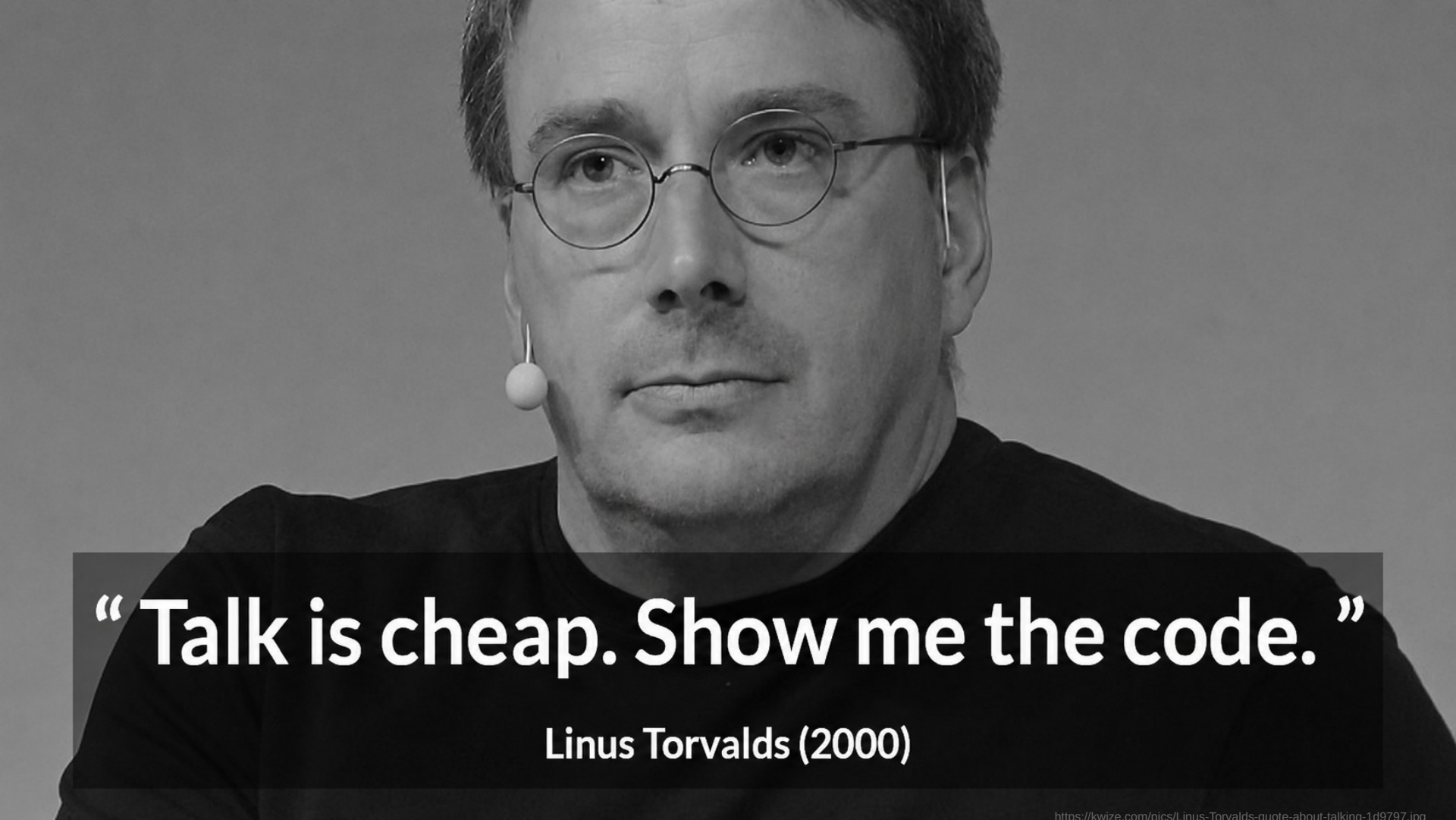
```
elementOf :: Equal a => a -> [a] -> Bool
```

```
elementOf a list =
```

```
    case list of
```

```
        [] -> False
```

```
        (h:t) -> eq a h || elementOf a t
```



“Talk is cheap. Show me the code.”

Linus Torvalds (2000)

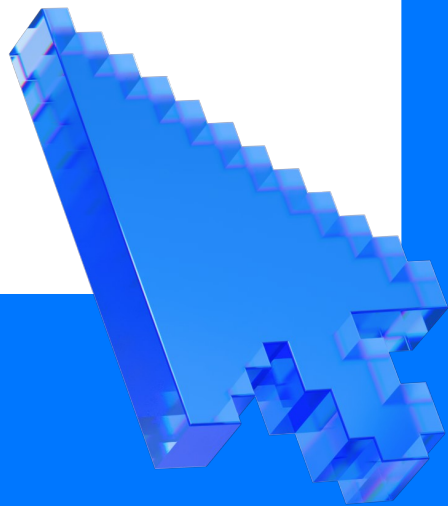


brian_goetz • 2mo ago

It is a well-established convention in JDK development to use double-underscore as a placeholder for "we've picked an obviously deliberately awful syntax for now, in the hope of deterring premature conversations about syntax"...

Классы типов (Type classes)

Набор конкретных типов,
для которых **определён witness**
дженерик-интерфейса, образуют
«класс» — получается «класс типов»



Классы типов (Type classes)

Публикация
и обнаружение

- Witness может быть объявлен в нескольких строго определённых местах
- Поиск подходящего witness происходит в compile time

Классы типов (Type classes)

Что ещё известно?

- Witness может не передаваться явно
- Найденный witness скорее всего будет `CONST_Dynamic` (JEP 309)
- Изменения нужны только в языке, JVM пока не трогают

Какие «дороги»
нам открываются



Поведения «снаружи» типа

- Описываем поведение через интерфейс с параметрами
- Определяем объект `witness` для конкретных типов
- В любом месте программы можем запросить `witness`

```
public interface Printable<T> {  
    void print(T value, OutputStream out);  
  
    static witness Printable<String> STRING_PRINTABLE  
}
```

```
Printable<String>.witness.print("hello", System.out);
```

Множественные реализации

```
interface Convertable<F, T> {  
    T to(F from);  
}
```

```
class MyNumber {  
    witness ConvertableTo<MyNumber, Double> CD  
    witness ConvertableTo<MyNumber, Long> CL  
}
```

Имена методов

```
interface Finalizable<T> {  
    void finalize(T t);  
}
```

```
class MySuperLock implements Lock {  
    static witness Finalizable<MySuperLock> FIN =  
        lock -> lock.unlock();  
}
```

Переопределение операторов

* Addition (+)	Numerical::add
* Subtraction (-)	Numerical:: subtract
* Multiplication (*)	Numerical: multiply
* Division (/)	Numerical:: divide
* Remainder (%)	Numerical: remainder
* Unary plus (+)	Numerical::plus
* Unary minus (-)	Numerical: negate
* Bitwise AND (&)	Integral::and
* Bitwise OR ()	Integral: or
* Bitwise XOR (^)	Integral: xor
* Bitwise complement (~)	Integral::complement
* Left shift (<<)	Integral::shiftLeft
* Right shift (>>)	Integral:: shiftRight
* Unsigned right shift (>>>)	Integral: shiftRightUnsianed
* Less than (<)	Orderable: less Than
* Less than or equal (<=)	Orderable: less ThanEqual
* Greater than (>)	Orderable: greaterThan
* Greater than or equal (>=)	Orderable:: greater ThanEqual

Новые числовые типы в Valhalla

UnsignedInt

Float16, Float8_E5M2

PolynomialDouble

ComplexTextbook,
ComplexPolarTextbook

Литералы коллекций

Возможно, будет так

```
MySuperCoolList<Integer> = #[1, 2, 3]
```

Литералы коллекций

Возможно, будет так

```
MySuperCoolList<Integer> = #[1, 2, 3]
```

```
public interface SequenceBuildable<T> {  
    SequenceBuilder<T> builder(int size);  
}
```

```
public class MySuperCoolList<T> implements List<T> {  
  
    witness SequenceBuildable<MySuperCoolList<T>>...  
}
```

Литералы коллекций

Возможно, будет так

```
MySuperCoolList<Integer> = #[1, 2, 3]
```

```
public interface SequenceBuildable<T> {  
    SequenceBuilder<T> builder(int size);  
}
```

```
public class MySuperCoolList<T> implements List<T> {  
  
    witness SequenceBuildable<MySuperCoolList<T>>...  
}
```

// под капотом превратится

```
SequenceBuildable<MySuperCoolList<Integer>>.witness.builder(3)  
    .add(1)  
    .add(2)  
    .add(3)  
    .build();
```

Creational expressions

Value-типы и значение
по-умолчанию

Creational expressions

Value-типы и значение
по-умолчанию

```
new Person![10]
```

Creational expressions

Value-типы и значение
по-умолчанию

```
new Person![10]
```

ImplicitlyConstructible<T>

Creational expressions

Value-типы и значение
по-умолчанию

`new Person!`[10]

`ImplicitlyConstructible<T>`

Проверка '`new Person!`[10]'
в compile time

Нельзя не упомянуть о...



Andrej Karpathy 

@karpathy



The hottest new programming language is English

11:14 PM · Jan 24, 2023 · **12.8M** Views



2.3K



8.8K



73K



7.1K





Andrej Karpathy 

@karpathy



The hottest new programming language is English

11:14 PM · Jan 24, 2023 · **12.8M** Views



2.3K



8.8K



73K



7.1K



Ryan Dahl 

@rough__sea



This has been said a thousand times before, but allow me to add my own voice: the era of humans writing code is over. Disturbing for those of us who identify as SWEs, but no less true. That's not to say SWEs don't have work to do, but writing syntax directly is not it.

7:02 PM · Jan 19, 2026 · **7.3M** Views



931



2.6K



19K



6K



ИТОГИ

1

Классы типов добавят
больше возможностей
для полиморфизма

Итоги

1

Классы типов добавят
больше возможностей
для полиморфизма

2

Две новые концепции —
публикации
и обнаружения witness

Итоги

1

Классы типов добавят
больше возможностей
для полиморфизма

2

Две новые концепции —
публикации
и обнаружения witness

3

Основание
для будущих
расширений языка

Итоги

1

Классы типов добавят
больше возможностей
для полиморфизма

2

Две новые концепции —
публикации
и обнаружения witness

3

Основание
для будущих
расширений языка

4

Язык Java продолжает
развиваться, несмотря
на БЯМ-хайп



RuStore

Спасибо!

Андрей Васильев
RuStore



GitHub



Growing
the Java Language



Telegram



Reddit