Как Котлин разрабатывает фичи

На примере корутин и инлайн классов

О себе

- Ильмир Усманов
- Kotlin Language Research Team @ JetBrains
- Coroutines, Inline Classes, Contracts

План

- Корутины
- Инлайн классы
- Будущее

Корутины

- Поддержка асинхронного программирования в языке

Корутины

- Поддержка асинхронного программирования в языке
- Появилась в C# 5 (2012) как async/await

Корутины

- Поддержка асинхронного программирования в языке
- Появилась в C# 5 (2012) как async/await
- Затем в Haskell (2012)

Корутины

- Поддержка асинхронного программирования в языке
- Появилась в C# 5 (2012) как async/await
- Затем в Haskell (2012)
- Python 3.5 (2015)

Корутины

- Поддержка асинхронного программирования в языке
- Появилась в C# 5 (2012) как async/await
- Затем в Haskell (2012)
- Python 3.5 (2015)
- Kotlin (2017)

Корутины

- Поддержка асинхронного программирования в языке
- Появилась в C# 5 (2012) как async/await
- Затем в Haskell (2012)
- Python 3.5 (2015)
- Kotlin (2017)
- etc.

Корутины до 1.3

- Стандартная библиотека
- Компилятор

Корутины до 1.3

- **Стандартная библиотека**
- Компилятор

Корутины до 1.3: Continuations

```
package kotlin.coroutines.experimental

interface CoroutineContext

interface Continuation<T> {
    val context: CoroutineContext
    fun resume(value: T)
    fun resumeWithException(exception: Throwable)
}
```

Корутины до 1.3

- Стандартная библиотека
- **Компилятор**

Корутины до 1.3: CPS трансформация

```
suspend fun foo(i: Int) {}

fun foo(i: Int, continuation: Continuation<Unit>)
```

Корутины до 1.3: CPS трансформация

```
suspend fun foo(i: Int) {}
```

```
fun foo(i: Int, continuation: Continuation<Unit>)
```

```
fun foo(
    i: Int,
    continuation: kotlin.coroutines.experimental.Continuation<Unit>
)
```

Корутины до 1.3: Продолжения

```
class FooContinuation(
    val completion:
        kotlin.coroutines.experimental.Continuation<Any?>
): kotlin.coroutines.experimental.Continuation<Unit> {

    var data: Any? = null
    var exception: Throwable? = null

    // Rest is omitted
}
```

Корутины до 1.3: Продолжения

```
interface Continuation<T> {
    val context: CoroutineContext
    fun resume(value: T)
    fun resumeWithException(exception: Throwable)
}
```

```
class FooContinuation(
    val completion:
        kotlin.coroutines.experimental.Continuation<Any?>
): kotlin.coroutines.experimental.Continuation<Unit> {
    var data: Any? = null
    var exception: Throwable? = null

    // Rest is omitted
}
```

Корутины до 1.3: CPS трансформация

```
suspend fun foo(i: Int) {
    bar()
}
```

```
fun foo(
    i: Int,
    continuation: kotlin.coroutines.experimental.Continuation<Unit>
) {
    val c = FooContinuation(continuation)
    bar(c)
}
```

Корутины 1.3

- Стандартная библиотека
- Компилятор

Корутины 1.3

- **Стандартная библиотека**
- Компилятор

Корутины 1.3: Продолжения

```
package kotlin.coroutines.experimental

interface CoroutineContext

interface Continuation<T> {
    val context: CoroutineContext
    fun resume(value: T)
    fun resumeWithException(exception: Throwable)
}
```

Корутины 1.3

- Стандартная библиотека
- **Компилятор**

Корутины 1.3

```
suspend fun foo(i: Int) {}
```

```
fun foo(
    i: Int,
    continuation: kotlin.coroutines.experimental.Continuation<Unit>
)
```

Корутины 1.3

EVERYTHING

OK?

Корутины 1.3

Корутины 1.3: Интероп

```
suspend fun foo(i: Int) {
    bar()
}
```

```
fun foo(
    i: Int,
    continuation: kotlin.coroutines.experimental.Continuation<Unit>
) {
    val c = FooContinuation(continuation)
    bar(c)
}
```

Корутины 1.3: Интероп

```
fun foo(
    i: Int,
    continuation: kotlin.coroutines.experimental.Continuation<Unit>
) {
    val c = FooContinuation(continuation)
    bar(c)
}
```

```
fun bar(
    continuation: kotlin.coroutines.experimental.Continuation<Unit>
) {}
```

Корутины 1.3: Интероп

```
fun foo(
    i: Int,
    continuation: kotlin.coroutines.Continuation<Unit>
) {
    val c = FooContinuation(continuation)
    bar(ReleaseToExperimental(c))
}
```

```
fun bar(
    continuation: kotlin.coroutines.experimental.Continuation<Unit>
) {}
```

Корутины 1.3: Интероп

EVERYTHING

OK?

Корутины 1.3: Интероп

Корутины 1.3: Result

```
class FooContinuation(
    val completion:
        kotlin.coroutines.experimental.Continuation<Any?>
): kotlin.coroutines.experimental.Continuation<Unit> {

    var data: Any? = null
    var exception: Throwable? = null

    // Rest is omitted
}
```

Корутины 1.3: Result

```
class FooContinuation(
    val completion:
        kotlin.coroutines.experimental.Continuation<Any?>
): kotlin.coroutines.experimental.Continuation<Unit> {

    var data: Any? = null
    var exception: Throwable? = null
    var result: Result<Any?> = null

    // Rest is omitted
}
```

Корутины 1.3: Result

```
package kotlin.coroutines

interface CoroutineContext

interface Continuation<T> {
    val context: CoroutineContext
    fun resume(value: T)
    fun resumeWithException(exception: Throwable)
    fun resumeWith(result: Result<T>)
}
```

Корутины 1.3: Result

Корутины 1.3: Result

Корутины 1.3: Зависимости

```
inline class Result<T>(
    val value: Any?
)
```

Корутины 1.3: Зависимости

```
inline class Result<T>(
    val value: Any?
)
```

Корутины 1.3: Зависимости

```
inline fun <T> Result<T>.onSuccess(
    action: (value: T) -> Unit
): Result<T> {
    contract {
        callsInPlace(action, InvocationKind.AT_MOST_ONCE)
    }
    if (isSuccess) action(value as T)
    return this
}
```

Корутины 1.3: Зависимости

```
inline fun <T> Result<T>.onSuccess(
    action: (value: T) -> Unit
): Result<T> {
    contract {
        callsInPlace(action, InvocationKind.AT_MOST_ONCE)
    }
    if (isSuccess) action(value as T)
    return this
}
```

Корутины 1.3: Зависимости

- Стабильная фича зависит от двух экспериментальных

Корутины 1.3: Уроки

- Стабильная фича зависит от двух экспериментальных
  - Не делать так больше!

Корутины 1.3: Уроки

- Стабильная фича зависит от двух экспериментальных
  - Не делать так больше!
- Переименование пакетов требует конверсии

Корутины 1.3: Уроки

- Стабильная фича зависит от двух экспериментальных
  - Не делать так больше!
- Переименование пакетов требует конверсии
  - Аннотация @Experimental (@RequiresOptIn)

Корутины 1.3: Уроки

Корутины 1.3: Уроки

Инлайнинг корутин

kotlinx.coroutines

kotlinx-coroutines-core / kotlinx.coroutines.flow / Flow

Flow

```
interface Flow<out T> (source)
```

An asynchronous data stream that sequentially emits values and completes normally or with an exception.

Intermediate operators on the flow such as map, filter, take, zip, etc are functions that are applied to the *upstream* flow or flows and return a *downstream* flow where further operators can be applied to. Intermediate operations do not execute any code in the flow and are not suspending functions themselves. They only set up a chain of operations for future execution and quickly return. This is known as a *cold flow* property.

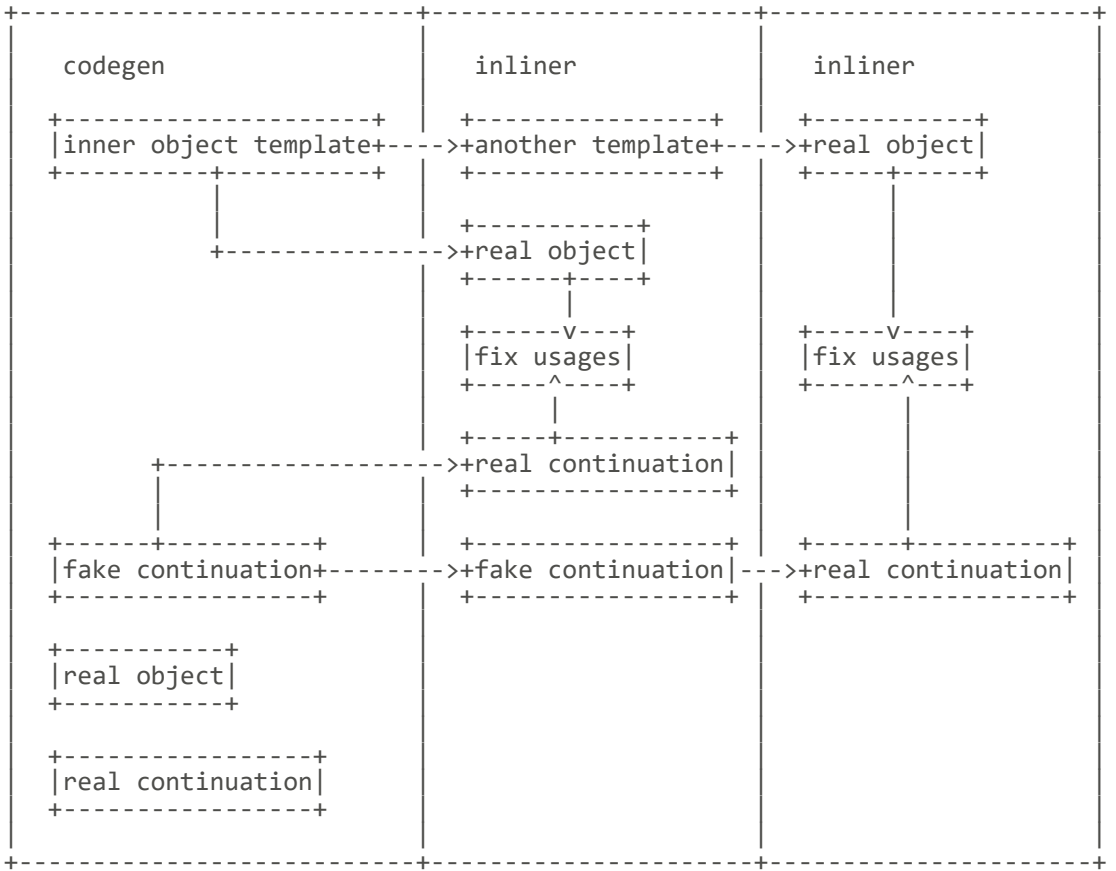
Инлайнинг корутин

```
inline fun <T, R> Flow<T>.map(
    crossinline transform: suspend (value: T) -> R
): Flow<R>
```

Инлайнинг корутин

```
Kotlin: [Internal Error] java.lang.IllegalStateException: Backend Internal error: Exception during code
generation
Cause: Back-end (JVM) Internal error: wrong code generated
org.jetbrains.kotlin.codegen.CompilationException Back-end (JVM) Internal error: Couldn't transform method
node:
doResume (Ljava/lang/Object;Ljava/lang/Throwable;)Ljava/lang/Object;:
@Lorg/jetbrains/annotations/Nullable;() // invisible
@Lorg/jetbrains/annotations/Nullable;() // invisible, parameter 0
@Lorg/jetbrains/annotations/Nullable;() // invisible, parameter 1
L0
L1
L2
LINENUMBER 13 L2
ALOAD 0
GETFIELD InlineSuspendFail$showFail$decoratedWork$1.this$0 : LInlineSuspendFail;
ASTORE 3
NOP
L3
LINENUMBER 48 L3
GETSTATIC java/lang/System.out : Ljava/io/PrintStream;
LDC "Starting suspend"
INVOKEVIRTUAL java/io/PrintStream.println (Ljava/lang/String;)V
L4
LINENUMBER 49 L4
NOP
L5
```

Инлайнинг корутин



Инлайнинг корутин

EVERYTHING

OK?

Инлайнинг корутин

Инлайнинг корутин

- Корутины – релизная фича

Инлайнинг корутин

- Корутины – релизная фича
  - Нельзя её просто взять и сломать

Инлайнинг корутин

- Корутины – релизная фича
  - Нельзя её просто взять и сломать
    - Спрашиваем комитет

Комитет

Было \ Стало	Красный	Зелёный
Красный	ОК	ОК
Зелёный	Спрашиваем комитет	Спрашиваем комитет

Комитет

Было \ Стало	Красный	Зелёный
Красный	ОК	ОК
Зелёный	Спрашиваем комитет	Спрашиваем комитет

Комитет не отвечает за дизайн языка!

Инлайнинг корутин

- Корутины – релизная фича
  - Нельзя её просто взять и сломать
    - Спрашиваем комитет
      - И только с его одобрения чиним

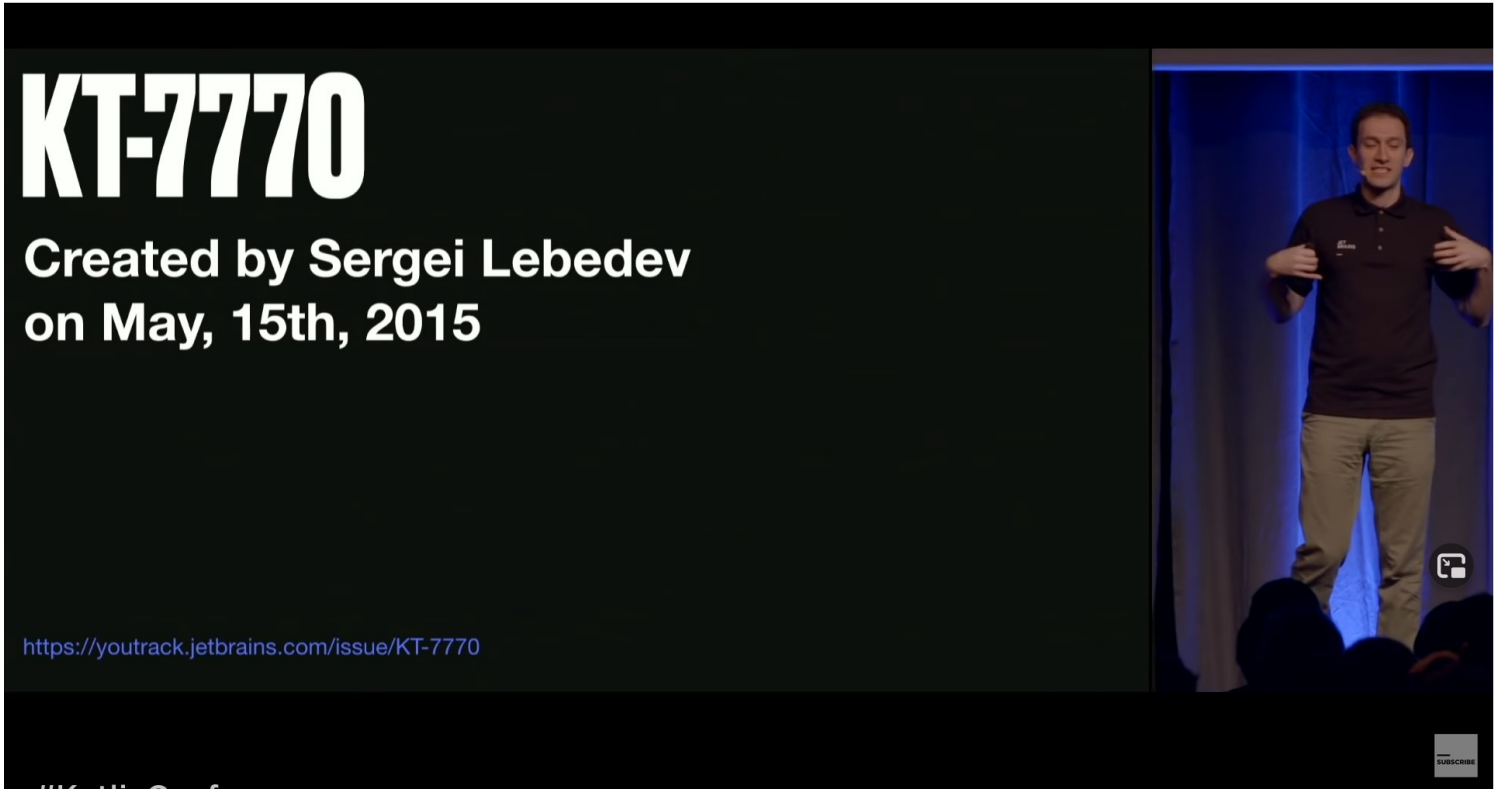
Корутины

EVERYTHING

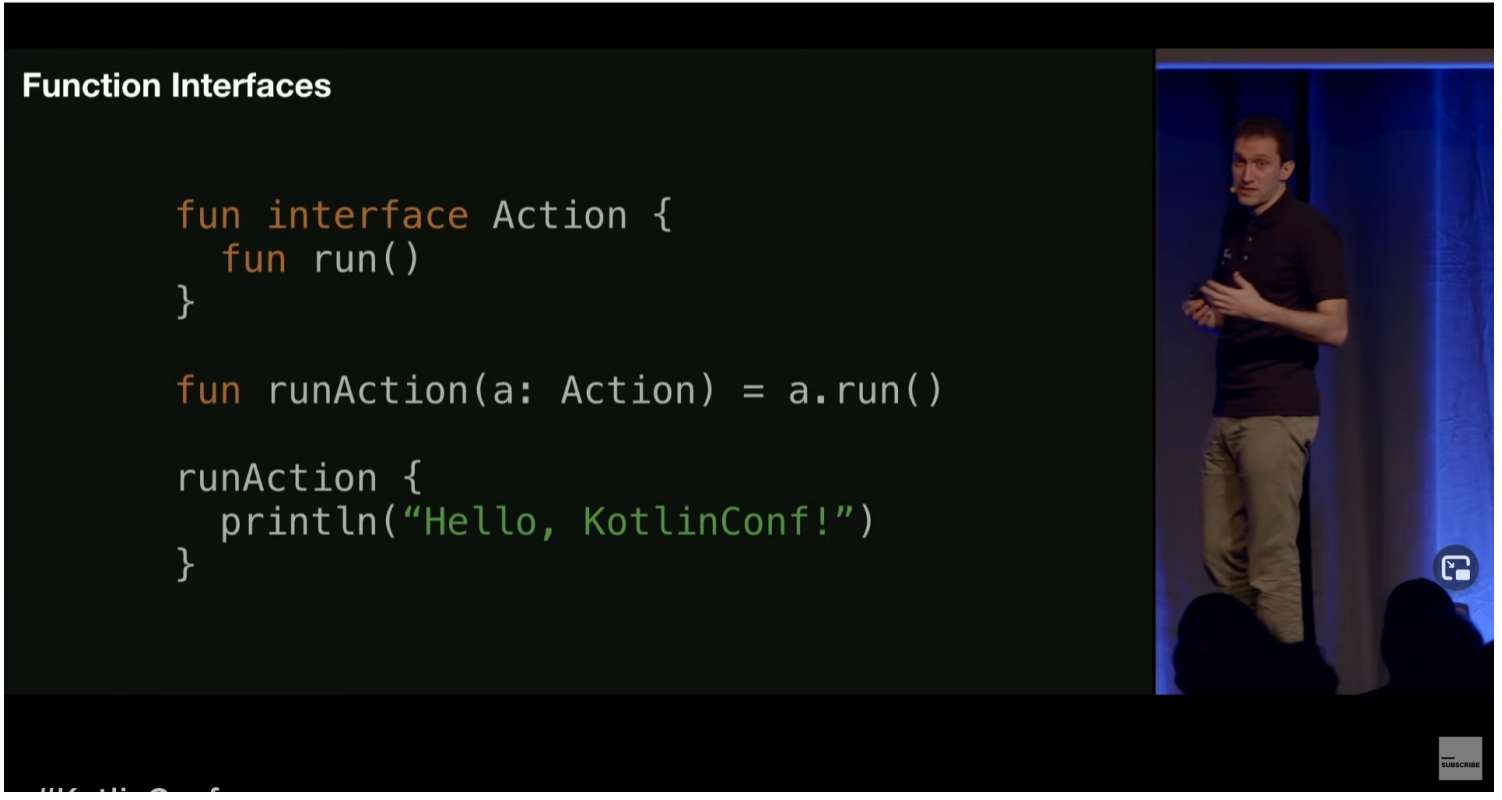
OK?

Корутины

Корутины: Fun Interfaces



Корутины: Fun Interfaces



Корутины: Fun Interfaces

```
fun interface Action {
    suspend fun run()
}
suspend fun runAction(a: Action) {
    a.run()
}
suspend fun main() {
    runAction {
        print("test")
    }
}
```

Корутины: Fun Interfaces

```
Exception in thread "main" java.lang.ClassCastException: TestKt$main$2 cannot be cast to
Action
at TestKt.main(Test.kt:8)
at TestKt$$$main.invoke(Unknown Source)
at kotlin.coroutines.intrinsics.IntrinsicsKt__IntrinsicsJvmKt$createCoroutineUnintercepted$
$inlined$createCoroutineFromSuspendFunction$IntrinsicsKt__IntrinsicsJvmKt$1.invokeSuspend(Intr
insicsJvm.kt:199)
at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33)
at kotlin.coroutines.ContinuationKt.startCoroutine(Continuation.kt:114)
at kotlin.coroutines.jvm.internal.RunSuspendKt.runSuspend(RunSuspend.kt:19)
at TestKt.main(Test.kt)
```

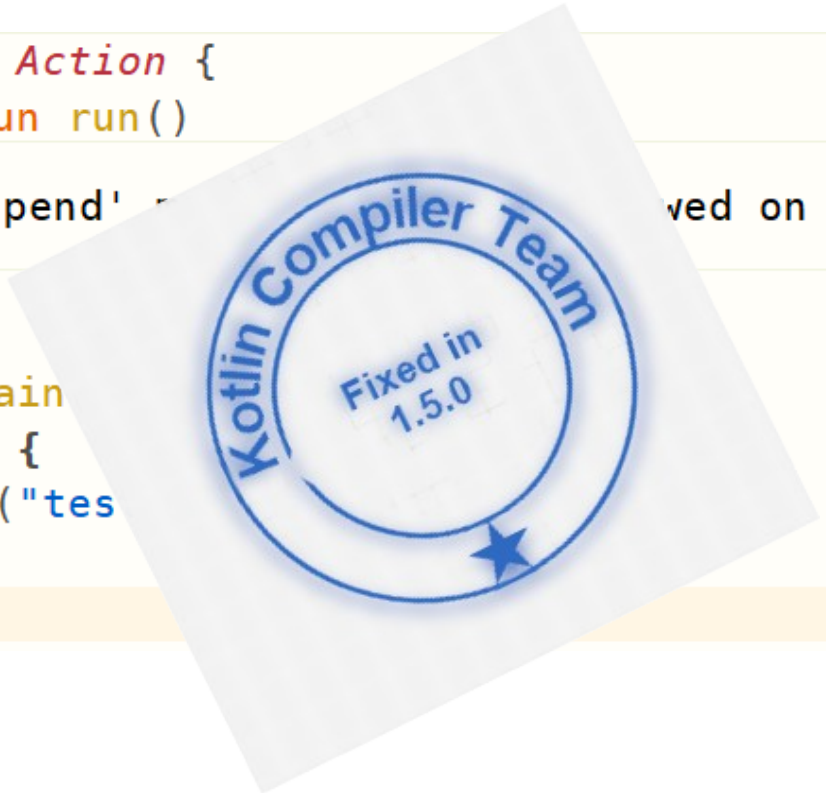
Корутины: Fun Interfaces

```
fun interface Action {
    suspend fun run()
}
suspend
    a.run()
}
suspend fun main() {
    runAction {
        print("test")
    }
}
```

'suspend' modifier is not allowed on a single abstract member

Корутины: Fun Interfaces

```
fun interface Action {
    suspend fun run()
}
suspend 'suspend' ved on a single abstract member
    a.run()
}
suspend fun main
    runAction {
        print("tes
    }
}
```



Бекенд

Версия / Ключ	1.4	1.5
-Xuse-ir	Включает новый бекенд	NOOP
-Xuse-old-backend	NOOP	Включает старый бекенд

Корутины: Fun Interfaces

EVERYTHING

OK?

Корутины: Fun Interfaces

Корутины: Structured Concurrency

kotlinx.coroutines

kotlinx-coroutines-core / kotlinx.coroutines / CancellationException

CancellationException

```
open expect class CancellationException :
    IllegalStateException (source)
```

Constructors

<init>

JS, NATIVE

```
CancellationException(message: String?)
CancellationException(message: String?, cause: Throwable?)
```

Thrown by cancellable suspending functions if the Job of the coroutine is cancelled while it is suspending. It indicates *normal* cancellation of a coroutine. **It is not printed to console/log by default uncaught exception handler**. (see CoroutineExceptionHandler).

Корутины: Structured Concurrency

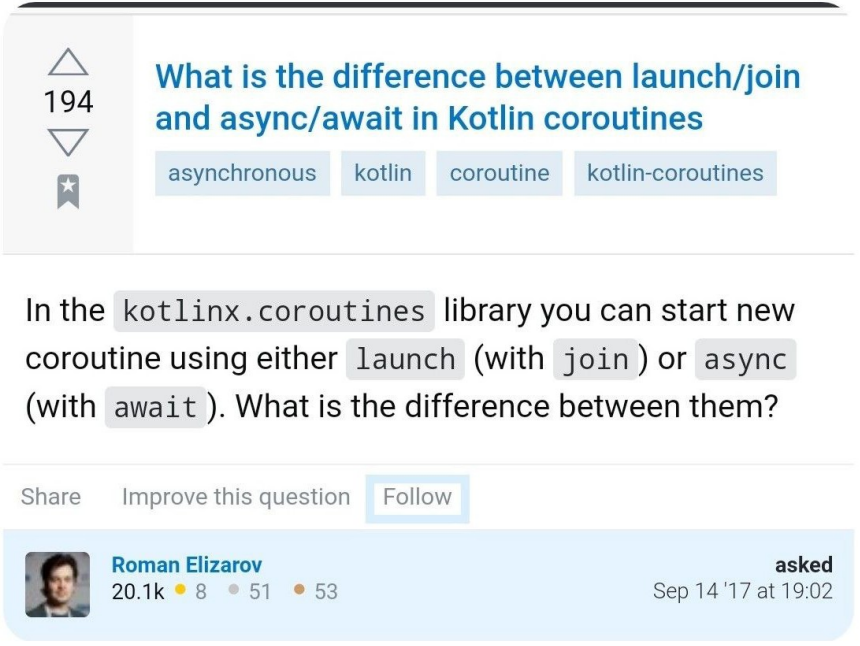
```
runCatching {
    coroutineScope {
        // Execution is cancelled
        throw CancellationException()
    }
}
```

Корутины: Structured Concurrency

- Предложение – встраиваем structured concurrency в язык

Корутины: Конец

Я: отлично знаю этот проект, он написан мной
Также я спустя неделю:



Инлайн Классы

```
inline class Result<T>(
    val value: Any?
)
```

Инлайн Классы: Определение

- Легковесная обёртка

Инлайн Классы: Определение

- Легковесная обёртка
- Аналогия – инлайн функции

Инлайн Классы: Определение

- Легковесная обёртка
- Аналогия – инлайн функции
- Для более типобезопасного API

Инлайн классы: Пример

```
inline class UserName(val s: String) {
    init {
        require(s.none { it.isWhitespace() })
    }
}
```

Инлайн классы: Манглинг

```
inline class IC(val i: Int)

fun foo(i: IC) {}
```

Инлайн классы: Манглинг

```
inline class IC(val i: Int)

fun foo(i: IC) {}


fun foo-K5cTq2M (i: Int) {}
```

Инлайн классы

EVERYTHING

OK?

Инлайн классы

Инлайн Классы

Инлайн классы: Boxing

```
inline class IC(val i: Int)

fun main() {
    val list = arrayListOf<IC>()
    list += IC(0)
}
```

Инлайн классы: Примитивные классы

```
@JvmPrimitive
inline value class IC(val i: Int)
```

Инлайн классы: Примитивные классы

EVERYTHING

OK?

Инлайн классы: Примитивные классы

Инлайн классы: Primitive Classes

Terminology update: primitive objects

Dan Smith daniel.smith at oracle.com
Mon Oct 5 19:18:40 UTC 2020

- Next message (by thread): Terminology update: primitive objects
- **Messages sorted by:** [date] [thread] [subject] [author]

We've been struggling with some uncomfortable rough edges of the "inline class"/"inline type" terminology for awhile. After multiple rounds of bikeshedding, here's an alternative that the Oracle team feels pretty good about:

- A *primitive object* is a new kind of object that lacks identity. It has special behavior with respect to equality, synchronization, and related operations. *Identity object* describes all other objects (including arrays). (The new objects are "primitive" in the sense that they are lighter-weight and represent simple, immutable values.)

- A *primitive class* (formerly *inline class*) is a special class whose instances are primitive objects. Primitive classes are always concrete and final, and their declarations are subject to various restrictions. A class that is not primitive is either an *identity class* or an *abstract class* (or Object, if we don't end up making it abstract).

- A *primitive value type* (formerly *inline type*) is a type whose values are primitive objects—the objects themselves, not *references* to the objects. Each primitive class has a primitive value type, typically denoted by the class name.

- A *primitive reference type* is a type whose values are references to primitive objects, or null. Each primitive class has a primitive reference type, typically denoted as ClassName.ref.

- The general term *primitive type* refers to either a primitive value type or a primitive reference type. The general term *reference type* continues to mean a type whose values are reference to objects (of unspecified kind), or null.

Инлайн классы

```
@JvmPrimitiveInline
inline value class IC(val i: Int)

inline class IC(val i: Int)
```

Инлайн классы

EVERYTHING

OK?

Инлайн классы

Инлайн Классы: Result

```
inline class Result<T>(
    val value: Any?
)
```

Инлайн Классы: Result

```
fun foo(result: Result<Unit>) {}
```

```
public final static foo(Ljava/lang/Object;)V
```

Инлайн Классы: Result

```
Caused by: java.lang.ClassCastException: class kotlin.Result cannot be cast to class
java.lang.Boolean (kotlin.Result is in unnamed module of loader 'app'; java.lang.Boolean is in
module java.base of loader 'bootstrap')
at kotlinx.coroutines.AsyncJvmTest$testResult$1$invokeSuspend$
$inlined$collect$1.emit(Collect.kt:133)
at kotlinx.coroutines.flow.internal.CombineKt$zipImpl$
$inlined$unsafeFlow$1$lambda$1$3$1$1.invokeSuspend(Combine.kt:133)
at kotlin.coroutines.jvm.internal.BaseContinuationImpl.resumeWith(ContinuationImpl.kt:33)
at kotlinx.coroutines.DispatchedTask.run(DispatchedTask.kt:106)
```

Инлайн Классы: Result

Инлайн Классы: Result

Инлайн Классы: Массивы

```
inline class UByte(val v: Byte)
inline class UByteArray(val a: ByteArray)

inline class UShort(val v: Short)
inline class UShortArray(val a: ShortArray)

inline class UInt(val v: Int)
inline class UIntArray(val a: IntArray)

inline class ULong(val v: Long)
inline class ULongArray(val a: LongArray)
```

Инлайн Классы: vararg

```
inline class IC(val a: Any)

fun foo(vararg args: IC) {

}
```

Forbidden vararg parameter type: IC

Инлайн Классы: Массивы

- Нам нужны реифированные массивы!

Инлайн Классы: Конец

Будущее: Прошлое

- KEEP

Будущее: Прошлое

- KEEP
  - Работает только для членов команды

Будущее: Прошлое

- KEEP
  - Работает только для членов команды
    - Не работает

Будущее

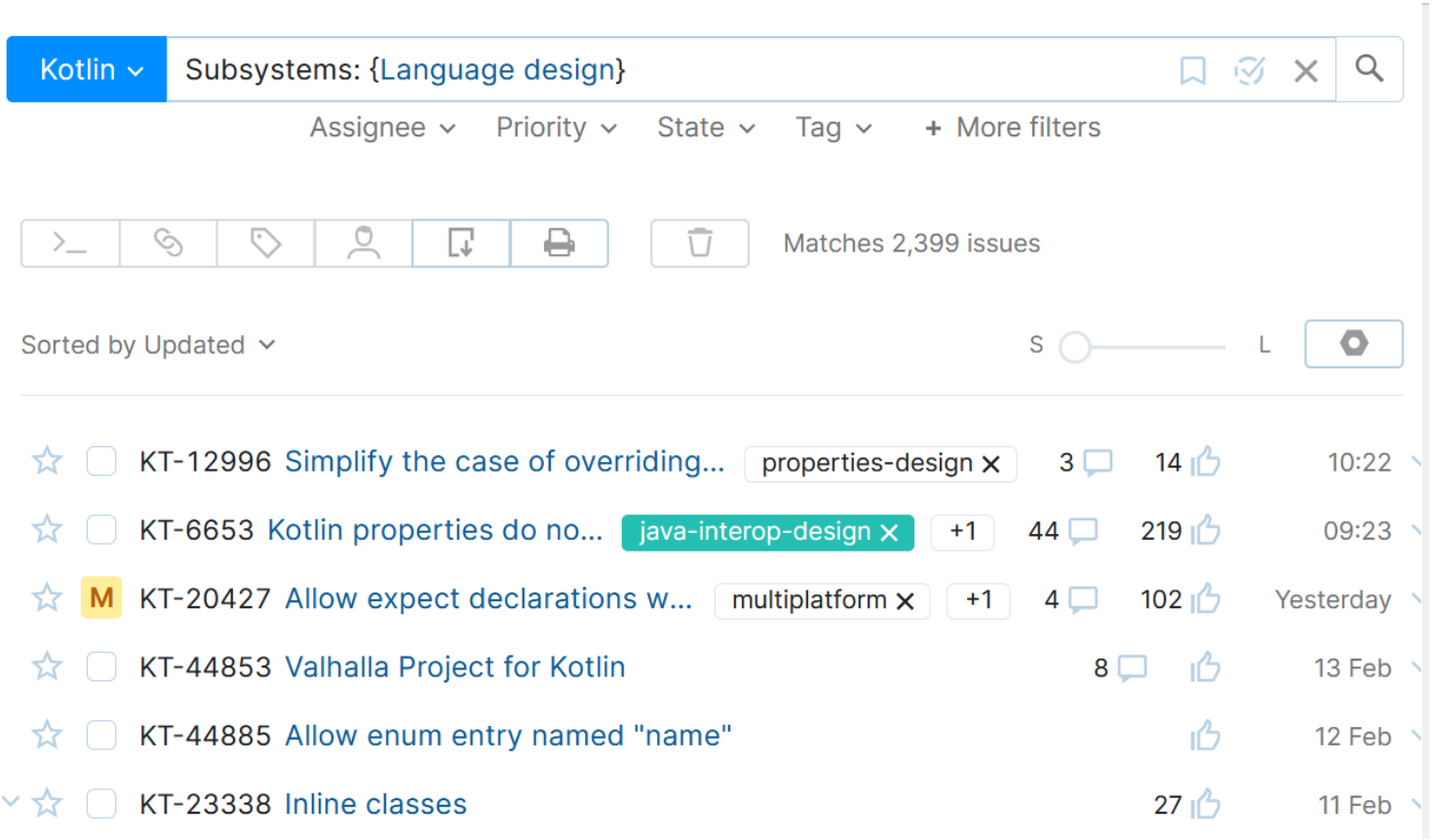
- YouTrack

Будущее

Пользователи

YouTrack

Будущее


Kotlin
Subsystems: {Language design}
Assignee
Priority
State
Tag
+ More filters
Matches 2,399 issues
Sorted by Updated
S
L
KT-12996 Simplify the case of overriding... properties-design 3 14 10:22
KT-6653 Kotlin properties do no... java-interop-design +1 44 219 09:23
M KT-20427 Allow expect declarations w... multiplatform +1 4 102 Yesterday
KT-44853 Valhalla Project for Kotlin 8 13 Feb
KT-44885 Allow enum entry named "name" 12 Feb
KT-23338 Inline classes 27 11 Feb

Будущее

Пользователи	Language Research Team

YouTrack  Proposal

Будущее

Пользователи	Language Research Team

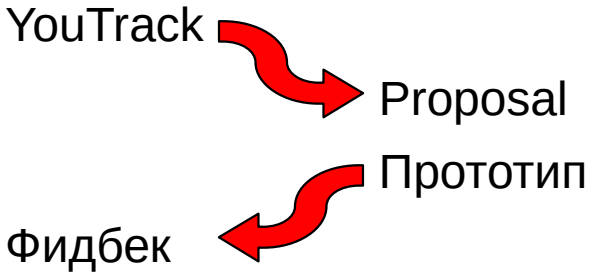
YouTrack

Proposal

Прототип

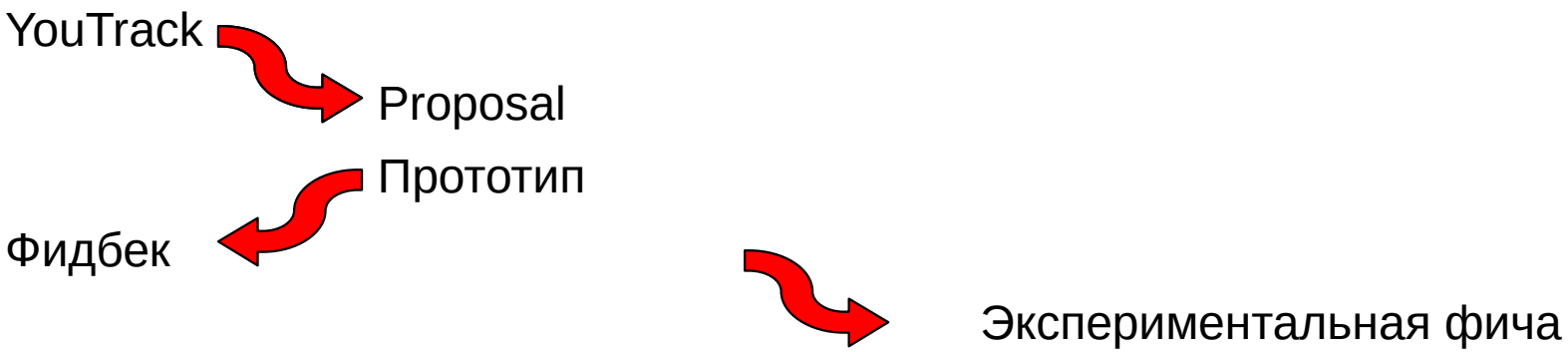
Будущее

Пользователи	Language Research Team



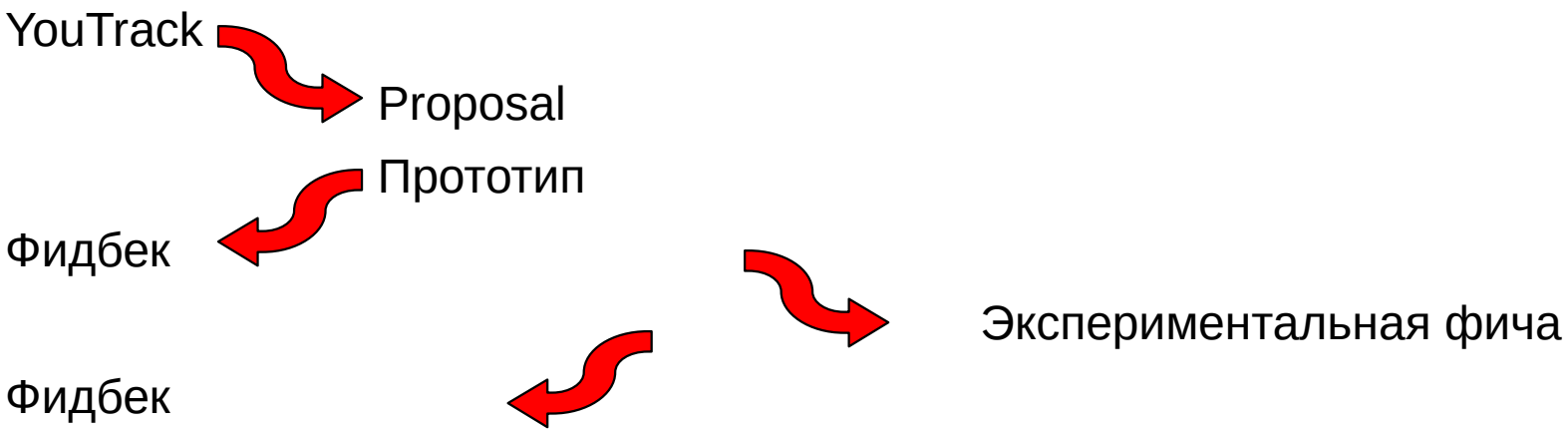
Будущее

Пользователи	Language Research Team	Команда компилятора



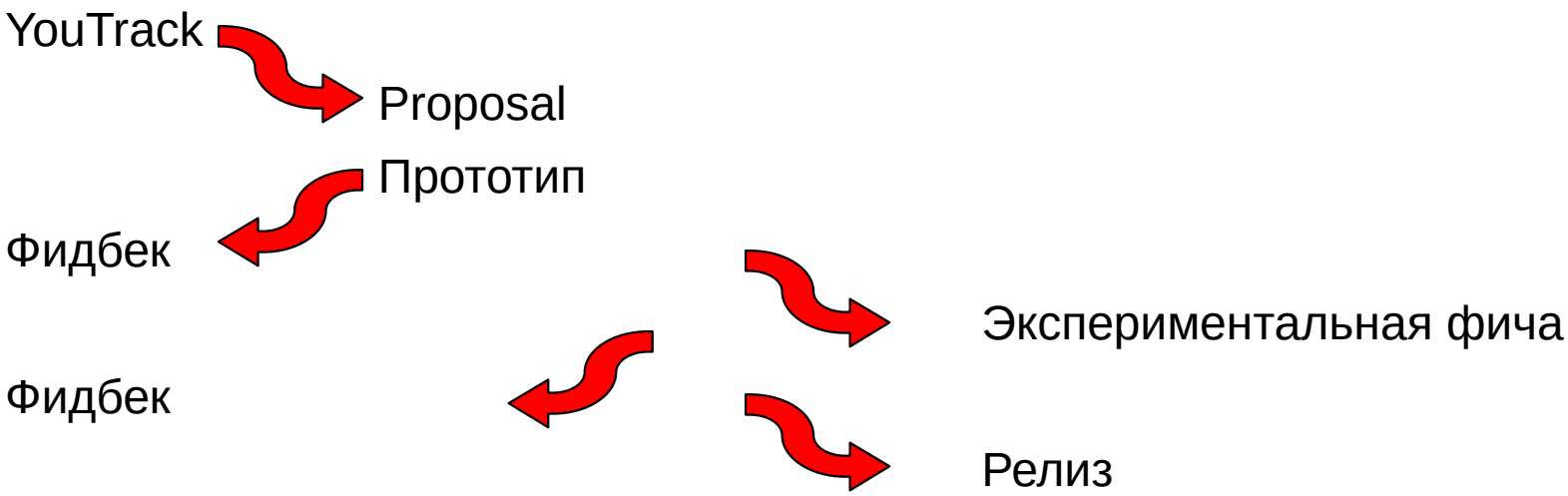
Будущее

Пользователи	Language Research Team	Команда компилятора



Будущее

Пользователи	Language Research Team	Команда компилятора



Будущее

- YouTrack
- Design Notes

Будущее

Design Notes on Kotlin Value Classes

- **Type**: Design notes
- **Author**: Roman Elizarov
- **Contributors**: Alexander Udalov, Andrey Breslav, Dmitry Petrov, Ilmir Usmanov, Ilya Gorbunov, Marat Akhin, Maxim Shafirov, Mikhail Zarechenskiy, Stanislav Erokhin
- **Status**: Under consideration
- **Discussion and feedback**: KEEP-237

This is not a design document, but an exploratory description of the current state of value classes in Kotlin and potential ways for their future evolution. The purpose of this document is to define a common terminology, provide some insight into potential avenues for features that are based on value classes, and thus establish a channel for discussion with the Kotlin community on use-cases for these features, their potential syntax, impact on existing Kotlin code, etc.

Будущее

Design Notes: Code Coloring

- **Type**: Design notes
- **Author**: Ilmir Usmanov
- **Contributors**: Roman Elizarov, Anton Banykh, Mikhail Zarechenskii, Sergey Rostov, Andrey Breslav, Zalim Bashorov, Mikhail Belyaev
- **Status**: Under consideration
- **Discussion and feedback**: TBD

While discussing new features, we, as Kotlin Language Research Team, are constantly running into cases when the context of one block of code is different from the context of another. Each this discussion ends with two words: "code coloring".

So, it is time to sit down and write about what code coloring is, which cases it covers and which issues it brings.

Будущее

- YouTrack
- Design Notes
- KEEP

Будущее

Inline classes

- **Type**: Design proposal
- **Author**: Mikhail Zarechenskiy
- **Contributors**: Andrey Breslav, Denis Zharkov, Dmitry Petrov, Ilya Gorbunov, Roman Elizarov, Stanislav Erokhin, Ilmir Usmanov
- **Status**: Beta since 1.4.30
- **Prototype**: Implemented in Kotlin 1.2.30

Discussion of this proposal is held in this issue.

Summary

Currently, there is no performant way to create wrapper for a value of a corresponding type. The only way is to create a usual class, but the use of such classes would require additional heap allocations, which can be critical for many use cases.

Контакты

- TG: @ilmirus
- koltinlang.slack.com: Ilmir Usmanov [JB]

Q&A