Evolving Kotlin API painlessly

Обо мне

- Software developer at Kotlin Libraries, JetBrains for 4 years
- Main project: `kotlinx.serialization`
- `@sandwwraith`

План на сегодня

1. Совместимость и как её можно сломать

План на сегодня

1. Совместимость и как её можно сломать
2. Какие инструменты мы можем использовать, чтобы предотвратить проблемы

План на сегодня

1. Совместимость и как её можно сломать
2. Какие инструменты мы можем использовать, чтобы предотвратить проблемы
3. Что делать, если всё-таки нужно что-то удалить

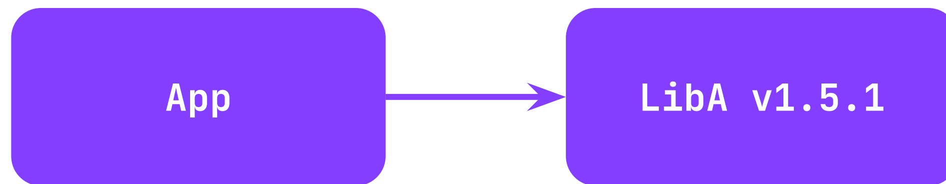
План на сегодня

1. Совместимость и как её можно сломать
2. Какие инструменты мы можем использовать, чтобы предотвратить проблемы
3. Что делать, если всё-таки нужно что-то удалить
4. Как лучше добавлять новый API

Совместимость

Часть I

Зачем мы это рассматриваем



😡 Блин, опять код красный весь, мигрировать надо, деплоить...

App

⬆️LibA v1.6.0

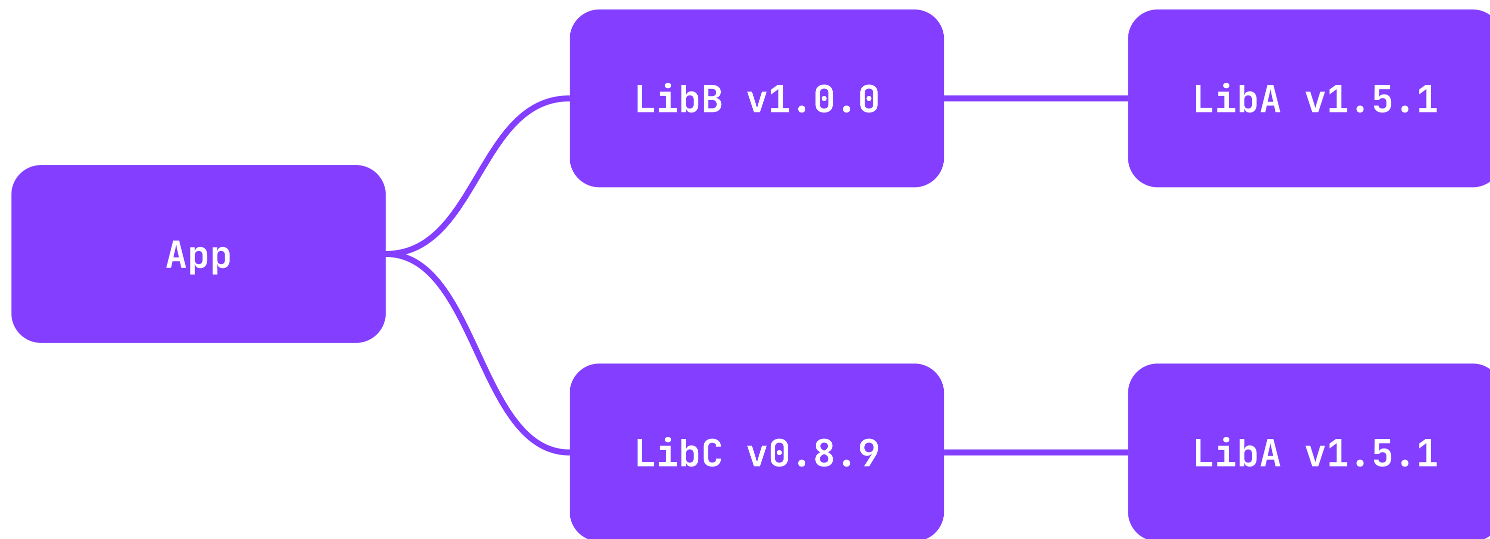
😡 Блин, опять код красный весь, мигрировать надо, деплоить...

Ну сорян чувак, поработай немного, тебе же за это деньги платят

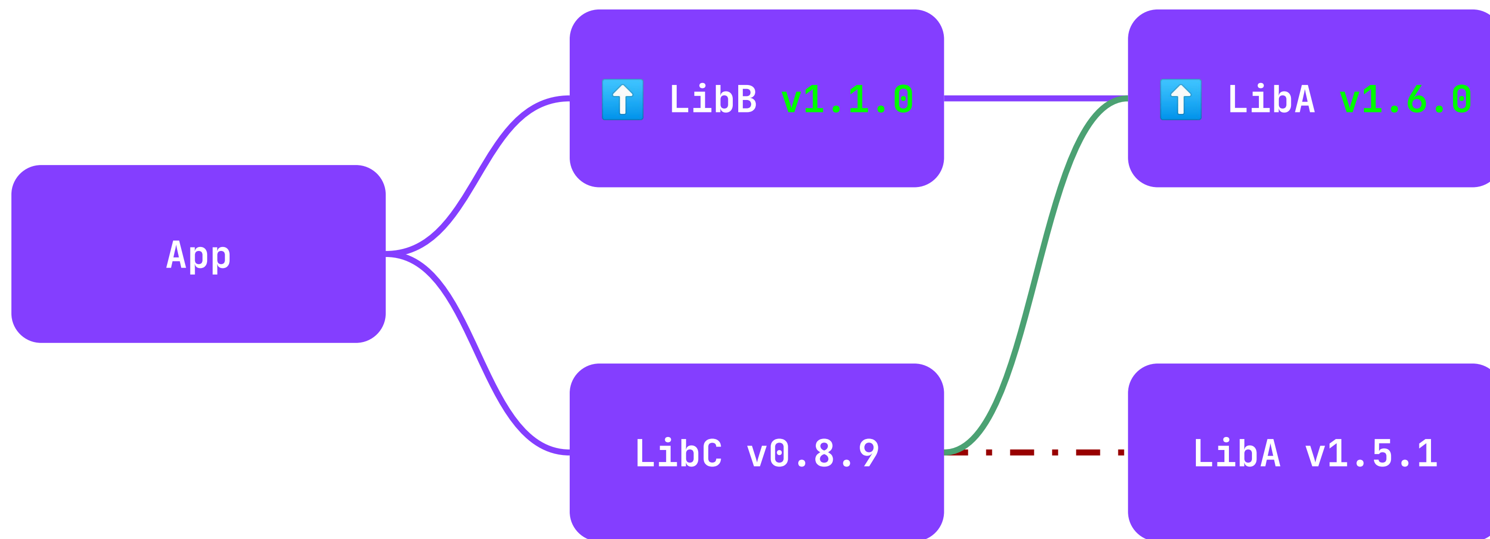
App

⬆️LibA v1.6.0

Как на самом деле происходит в реальности



Как на самом деле происходит в реальности



😡 Блин, какие ещё `NoSuchMethodException` и `IncompatibleClassChangeError`… что это вообще, я же ничего не менял!

App

⬆️ LibA v1.6.0

😡 Блин, какие ещё `NoSuchMethodException` и `IncompatibleClassChangeError`… что это вообще, я же ничего не менял!

ОЙ

кажется, нужно было бы и мне поработать....

App

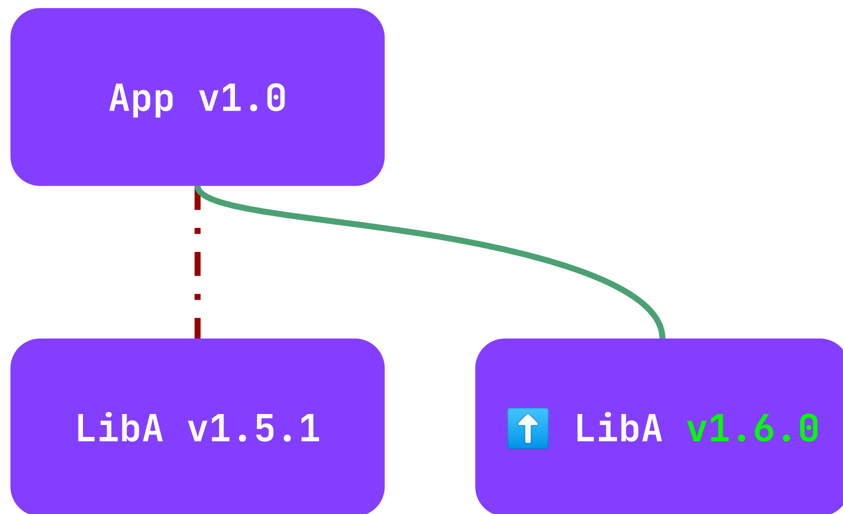
⬆️ LibA v1.6.0

Контекст

- Kotlin/JVM
- Точка зрения разработчика библиотеки
- Или модуля в сложной системе
- Как open-source, так и in-house

Backward compatibility

- Если новая версия библиотеки не ломает клиентов, ожидающих старую
- Ломается, например, удалением деклараций



Виды совместимости

- Functional
- Binary
- Source

Functional compatibility

v1.5.0

v1.5.1

```
fun feed() = "Вы покормили тигра"
```

```
fun feed() = "Тигр откусил вам руку"
```

https://xkcd.com/1172/

Source compatibility

v1.5.0

```
fun foo() {
    Tiger().feed()
}
```

v1.5.1

```
fun foo() {
    Tiger().feed()
}
```

Unresolved reference: feed

Binary compatibility

v1.5.0

```
fun foo() {
    Tiger().feed()
}
```

v1.5.1

```
java.lang.NoSuchMethodError:
Tiger.feed()Ljava/lang/String;
```

- Может произойти, даже если вы напрямую не используете проблемную декларацию и ваш код компилируется нормально
- Играем по правилам Java Virtual Machine

Что хотим

- Full Functional compatibility (по модулю багфиксов)
- Backward Source compatibility — крайне желательно
- Backward Binary compatibility — must have

Source = Binary?

Source = Binary?

Зачастую, но не всегда:

- Удаление декларации
- Изменение списка параметров функции
- Добавление `abstract` метода
- и так далее

Можно ли сломать только source?

```kotlin
fun <T> Iterable<T>.joinToString(
    separator: CharSequence = ", ",
    prefix: CharSequence = "",
    postfix: CharSequence = "",
    limit: Int = -1,
    truncated: CharSequence = "...",
    transform: ((T) -> CharSequence)? = null
)
```

```kotlin
fun <T> Iterable<T>.joinToString(
    separator: CharSequence = ", ",
    prefix: CharSequence = "",
    postfix: CharSequence = "",
    ~~limit~~maxItems: Int = -1,
    truncated: CharSequence = "...",
    transform: ((T) -> CharSequence)? = null
)
```

В байткоде же нет имён параметров!

```kotlin
fun foo(list: List<String>) {
    list.joinToString(limit = 3) {
        it.removePrefix("//")
    }
}
```

Ломаем binary незаметно

```
fun jump(length: Int = 1) = "Jumped $length meters"
```

Ломаем binary незаметно

```
fun jump(length: Int = 1) = "Jumped $length meters"

fun jump(length: Int = 1, height: Int = 1) = "Jumped ${dist(length, height)} meters"
```

Ломаем binary незаметно

```
fun jump(length: Int = 1) = "Jumped $length meters"

fun jump(length: Int = 1, height: Int = 1) = "Jumped ${dist(length, height)} meters"
```

```
-    public final fun jump (I)Ljava/lang/String;
-    public static synthetic fun jump$default
(Ldemo/kotlin/Tiger;IILjava/lang/Object;)Ljava/lang/String;
+    public final fun jump (II)Ljava/lang/String;
+    public static synthetic fun jump$default
(Ldemo/kotlin/Tiger;IIILjava/lang/Object;)Ljava/lang/String;
```

Методы из ниоткуда

```
data class JsonConfiguration(
    val encodeDefaults: Boolean = false,
    val ignoreUnknownKeys: Boolean = false,
)
```

```kotlin
data class JsonConfiguration(
    val encodeDefaults: Boolean = false,
    val ignoreUnknownKeys: Boolean = false,
    val newJsonFlag: Boolean = false
) {
    constructor(
        encodeDefaults: Boolean = false,
        ignoreUnknownKeys: Boolean = false
    ) : this(encodeDefaults, ignoreUnknownKeys, newJsonFlag = false)
}
```

```kotlin
data class JsonConfiguration(
    val encodeDefaults: Boolean = false,
    val ignoreUnknownKeys: Boolean = false,
    val newJsonFlag: Boolean = false
)
```

```
config.co
m copy(encodeDefaults: Boolean = ..., ignoreUnknownKeys: Boolean = ..., newJsonFlag: Boolean = ...)   JsonConfiguration
v encodeDefaults                                                                                      Boolean
```

Чтобы добавить новое свойство, нужно:

- Перегрузить конструктор
- Перегрузить `copy`
- Переписать `equals/hashCode` (если свойство добавлено в тело)

Data классы почти не нужны библиотекам

Чтобы добавить новое свойство, нужно:

- Перегрузить конструктор
- Перегрузить `copy`
- Переписать `equals/hashCode` (если свойство добавлено в тело)

Переставлять свойства почти невозможно из-за `componentN()`

Ещё варианты

```
fun ageAsHuman(age: Int): Number = age * 7
```

Ещё варианты

```
fun ageAsHuman(age: Int): Number = age * 7

fun ageAsHuman(age: Int): Int = age * 7

fun ageAsHuman(age: Number): Int = age * 7
```

Сужение типа результата или расширение типа параметра *скорее всего* не добавит проблем в исходниках, но принесёт головную боль в бинарниках.

Проблемы?

Проблемы?

- Код компилируется
- Проходят тесты

Проблемы?

- Код компилируется
- Проходят тесты
- **Релиз новой версии влечёт много гневных issue**

Проблемы?

- Код компилируется
- Проходят тесты
- **Релиз новой версии влечёт много гневных issue**

Binary-breaking change, сохраняющий source совместимость — самый опасный тип изменений!

Как следить

Часть II

Как следить за бинарной совместимостью?

1. Помнить на зубок, как Kotlin компилируется в байткод
2. Писать интеграционные тесты
3. Сравнивать JAR-ники
4. ????

Binary compatibility validator

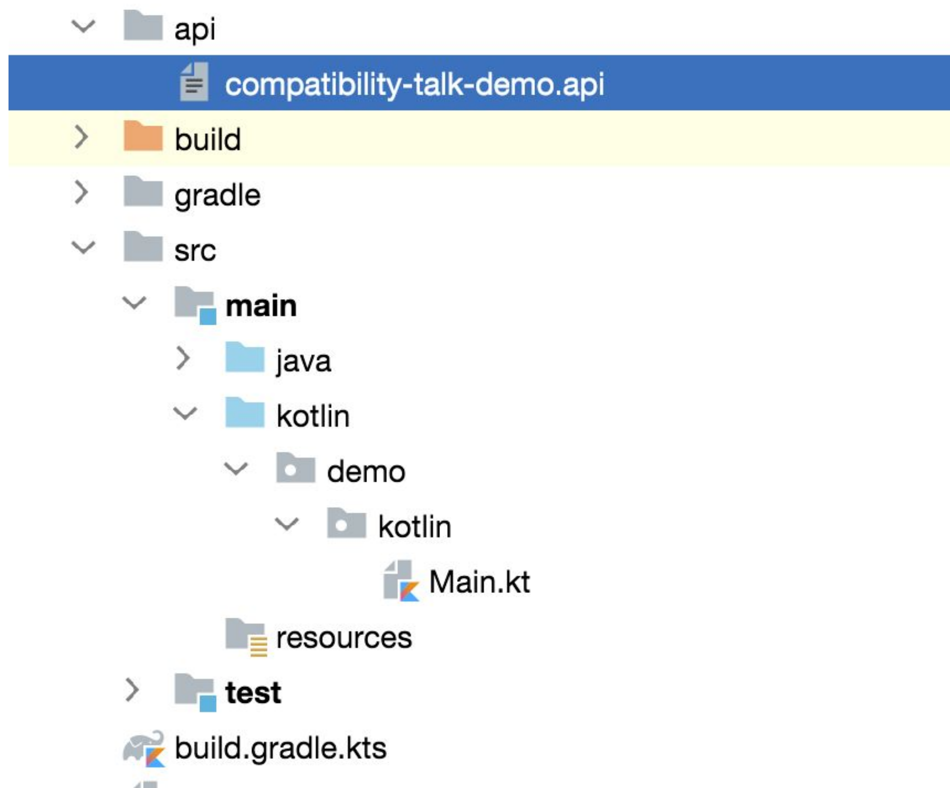
```
plugins {
    kotlin("jvm") version "1.5.10"
    id("org.jetbrains.kotlinx.binary-compatibility-validator") version "0.8.0-RC"
}
```

Binary compatibility validator

```
plugins {
    kotlin("jvm") version "1.5.10"
    id("org.jetbrains.kotlinx.binary-compatibility-validator") version "0.8.0-RC"
}
```

```
./gradlew tasks
apiDump - Syncs API from build dir to provider dir for <project name>
apiCheck - Checks signatures of public API against the golden value in API folder for <project
name>
```

./gradlew apiDump



./gradlew apiDump

```kotlin
class Tiger {
    fun meow() = println("Meow!")
}
```

```
public class demo/java/Tiger {
  public fun <init> ()V
  public fun meow ()V
}
```

Настройка валидатора

1. Подключить плагин
2. Сгенерировать начальный api файл с помощью apiDump
3. Внести его в VCS

./gradlew apiCheck

```kotlin
class Tiger {
    fun meow(loudness: Int = 1) = println("Meow! ".repeat(loudness))
}
```

./gradlew apiCheck

```kotlin
class Tiger {
    fun meow(loudness: Int = 1) = println("Meow! ".repeat(loudness))
}
```

```
API check failed for project compatibility-talk-demo.

 public final class demo/kotlin/Tiger {
     public fun <init> ()V
-    public final fun meow ()V
+    public final fun meow (I)V
+    public static synthetic fun meow$default (Ldemo/kotlin/Tiger;IILjava/lang/Object;)V
 }

 You can run :compatibility-talk-demo:apiDump task to overwrite API declarations
```

Использование валидатора

Если изменения в API внесены случайно, то...

1. `apiCheck` таска (часть `./gradlew check`) провалит вам билд

Если изменения в API внесены случайно, то…

1. `apiCheck` таска (часть `./gradlew check`) провалит вам билд

Если изменения в API внесены специально, то…

1. Обновить api файл

Если изменения в API внесены случайно, то...

1. `apiCheck` таска (часть `./gradlew check`) провалит вам билд

Если изменения в API внесены специально, то...

1. Обновить api файл
2. Отправить как часть ревью
    - Только **добавления** строк — API **совместимо**
    - Есть **modified/deleted** — скорее всего, **нет**

Explicit API mode

По умолчанию, декларации имеют **public** видимость.

…In real Java code bases (where public/private decisions are taken explicitly), public occurs a lot more often than private (2.5 to 5 times more often in the code bases that we examined, including Kotlin compiler and IntelliJ IDEA). — https://blog.jetbrains.com/kotlin/2015/09/kotlin-m13-is-out/

```kotlin
class Tiger {
    fun meow(loudness: Int = 1) = println(meowOnce().repeat(loudness))

    fun meowOnce() = "Meow!"
}
```

```kotlin
class Tiger {
    public fun meow(loudness: Int = 1) = println(meowOnce().repeat(loudness))

    public? fun meowOnce() = "Meow!"
}
```

- Легко забыть
- Неочевидно на ревью

Explicit Visibility

```
fun meowOnce(): String = "Meow!"
```

Visibility must be specified in explicit API mode

Make 'meowOnce' public explicitly More actions...

Type Inference

```
public class Tiger {
    public fun meow(loudness: Int = 1): Unit = println(meowOnce().repeat(loudness))

    public fun meowOnce(): String = "Meow!"
}
```

Возможность случайно изменить return type

```
public class Tiger {
    public fun meow(loudness: Int = 1): Unit = println(meowOnce().repeat(loudness))

    public fun meowOnce(): CharSequence = "Meow!" as CharSequence
}
```

```
API check failed for project compatibility-talk-demo.

-    public final fun meowOnce ()Ljava/lang/String;
+    public final fun meowOnce ()Ljava/lang/CharSequence;
```

Explicit return type

```
public class Tiger {
    public fun meow(loudness: Int = 1): Unit = println(meowOnce().repeat(loudness))

    public fun meowOnce() : String  = "Meow!"
}
```

Return type must be specified in explicit API mode

Specify return type explicitly ⌥⇧↵ More actions... ⌥↵

Explicit API mode

- Защищает от “случайных” публичных деклараций
- Защищает от случайной смены return type из-за вывода типов
- Ко всем диагностикам есть quickfixes

Как включить

```
kotlin {
    explicitApi()
    // the same as
    explicitApi = ExplicitApiMode.Strict
}
```

Предупреждения вместо ошибок

```kotlin
kotlin {
    explicitApiWarning()
    // the same as
    explicitApi = ExplicitApiMode.Warning
}
```

All warnings as errors

Позволяет поддерживать проект без предупреждений компиляции, таких как:

- Unused parameter
- Unchecked cast
- Extension shadowed by member
- И другие

```
freeCompilerArgs += “-Werror”
```

Необязательно включать везде

```kotlin
tasks.named<KotlinCompile>("compileKotlin") {
    kotlinOptions.allWarningsAsErrors = true
}

tasks.named<KotlinCompile>("compileTestKotlin") {
    kotlinOptions.allWarningsAsErrors = false
}
```

- Binary compatibility validator
- Explicit API mode
- All warnings as errors

Как ломать

Часть III

@Deprecated в Java

```java
public class Tiger {
    public void meow() {
        System.out.println("Meow!");
    }
}
```

```java
public class Tiger {
    @Deprecated
    public void meow() {
        System.out.println("Meow!");
    }

    public void roar() {
        System.out.println("RRRRRRRRR!");
    }
}
```

```java
public static void main(String[] args) {
    var tiger = new Tiger();
    tiger.meow();
}
```

'meow()' is deprecated

```java
Tiger
@Deprecated
public void meow()
```

compatibility-talk-demo.main

```java
public class Tiger {

    /**
     * Prints 'Meow!' to stdout
     *
     * @deprecated Tigers do not meow, they {@link #roar()}
     */
    @Deprecated
    public void meow() {
        System.out.println("Meow!");
    }
}
```

```java
public static void main(String[] args) {
    var tiger = new Tiger();
    tiger.meow();
}
```

'meow()' is deprecated

Replace method call with Tiger.roar More actions...

```java
Tiger
@Deprecated
public void meow()
```

Prints 'Meow!' to stdout

Deprecated Tigers do not meow, they roar()

compatibility-talk-demo.main

@Deprecated в Kotlin

```
public annotation class Deprecated(
    val message: String,
    val replaceWith: ReplaceWith = ReplaceWith(""),
    val level: DeprecationLevel = DeprecationLevel.WARNING
)
```

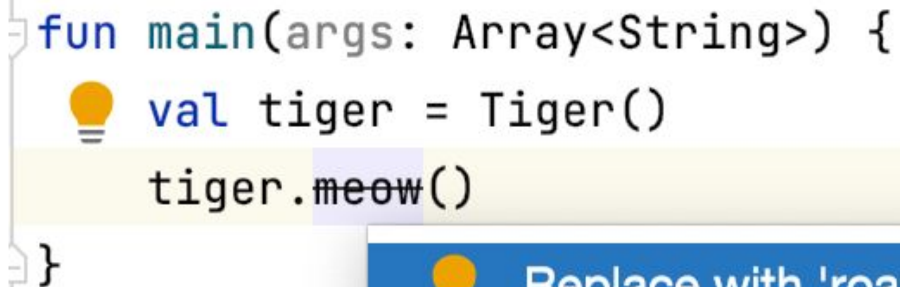
```kotlin
class Tiger {
    /**
     * Prints 'Meow!' to stdout
     */
    @Deprecated("Tigers do not meow, they roar", ReplaceWith("roar()"))
    fun meow() = println("Meow!")

    fun roar() = println("RRRRRRRR!")
}
```

```kotlin
fun main(args: Array<String>) {
    val tiger = Tiger()
    tiger.meow()
}
```

'meow(): Unit' is deprecated. Tigers do not meow, they roar

Replace with 'roar()' More actions...

```
src/main/kotlin/demo/kotlin/Main.kt: (15, 11): 'meow(): Unit'
is deprecated. Tigers do not meow, they roar
```



Replace with 'roar()'

Replace usages of 'meow(): Unit' in whole project

ReplaceWith аргументов

```
public class Tiger {
    fun roar(count: Int) = Unit
}
```

```
public class Tiger {
    fun roar(count: Int) = Unit

    fun roar(count: Long) = Unit
}
```

```kotlin
public class Tiger {
    @Deprecated("Need more", ReplaceWith("roar(count.toLong())"))
    fun roar(count: Int) = Unit

    fun roar(count: Long) = Unit
}
```

```kotlin
public class Tiger {
    @Deprecated("Need more", ReplaceWith("roar(count.toLong())"))
    fun roar(count: Int) = Unit

    fun roar(count: Long) = Unit
}
```

```kotlin
fun foo(t: Tiger, i: Int) {
    t.roar( count: 24)
    t.roar(i)
}
```

Replace with 'roar(count.toLong())'

Replace usages of 'roar(Int): Unit' in whole project

```kotlin
fun foo(t: Tiger, i: Int) {
    t.roar(24.toLong())
    t.roar(i.toLong())
}
```

Можно указать, не только на какую функцию заменить вызов, но и как трансформировать аргументы!

DeprecationLevel

`WARNING`

- Обычное предупреждение в CLI или IDE
- Превращается в ошибку с -Werror

```
tiger.meow()
```

DeprecationLevel

WARNING

- Обычное предупреждение в CLI или IDE
- Превращается в ошибку с -Werror

```
tiger.meow()
```

ERROR

- Использование такой декларации приводит к ошибке компиляции

```
tiger.meow()
```

DeprecationLevel.HIDDEN

- Пропадает из completion-а
- Не видна для компилятора
- Даже для Java
- Остаётся в ABI
- Можно “удалить” функцию, не удаляя её

```
tiger.meow()
```

Unresolved reference: meow

Deprecation cycle

```
WARNING → ERROR → HIDDEN
```

Когда повышать уровень?

Deprecation cycle

```
WARNING → ERROR → HIDDEN
```

Когда повышать уровень?

Тогда, когда это указано в вашей Compatibility Policy, которую вы объявили пользователям

Semantic versioning

`MAJOR.MINOR.PATCH`

- *Major version MUST be incremented if any backwards incompatible changes are introduced to the public API.*
- *Minor version […] MUST be incremented if any public API functionality is marked as deprecated.*
- Тип совместимости (source или binary) не указан

Kotlin versioning

`1.N`	Initial
`1.(N + 1)`	Deprecated with warning
`1.(N + 2)`	Deprecated with error
`1.(N + 3)`	Hidden
`2.0`	We can (but not obliged to) delete declaration

Альтернативная схема

`1.N`	Initial
`1.(N + 1)`	Deprecated with warning
`2.0`	Deprecated with error
`2.1`	Hidden
`3.0`	We can (but not obliged to) delete declaration

Ваша policy может быть любой (в пределах разумного), но о ней должны знать пользователи

Эволюция API

Часть IV

В одной библиотеке может быть разный API

- Stable
- Deprecated
- Internal (with public modifiers for whatever reason)
- **Experimental**
- ...

Почему Experimental?

- Другая compatibility policy
- Более быстрый deprecation cycle
- Возможность семантических изменений

Opt-In аннотации

```
@Target(ANNOTATION_CLASS)
public annotation class RequiresOptIn(
    val message: String = "",
    val level: Level = Level.ERROR
)
```

Создание Opt-In аннотации

```
@RequiresOptIn(level = RequiresOptIn.Level.WARNING)
@Target(...)
public annotation class ExperimentalUnsignedTypes
```

Что видит пользователь

```
fun calculateSeries(time: Duration, data: UByteArray) {
```

Использование: Propagating

```
@ExperimentalTime
@ExperimentalUnsignedTypes
fun calculateSeries(time: Duration, data: UByteArray)
```

Использование: Propagating

```
@ExperimentalTime
@ExperimentalUnsignedTypes
fun calculateSeries(time: Duration, data: UByteArray)
```

- Пользователи увидят такое же предупреждение
- Автоматически, если тип есть в сигнатуре (с 1.5.30)

Использование: Opt-In

```kotlin
@OptIn(ExperimentalTime::class, ExperimentalUnsignedTypes::class)
fun calculateSeries(time: Duration, data: UByteArray) {
```

- Использование без предупреждений

Prefer propagating over non-propagating

- Используйте “заразность” там, где экспериментальные типы видны в публичном API

Prefer propagating over non-propagating

- Используйте “заразность” там, где экспериментальные типы видны в публичном API
- Если экспериментальный тип — часть реализации, то:
  - Propagate, если экспериментальная аннотация принадлежит не вам

Prefer propagating over non-propagating

- Используйте “заразность” там, где экспериментальные типы видны в публичном API
- Если экспериментальный тип — часть реализации, то:
  - Propagate, если экспериментальная аннотация принадлежит не вам
  - Non-propagate, если вы контролируете экспериментальность этого API

Prefer propagating over non-propagating

- Используйте “заразность” там, где экспериментальные типы видны в публичном API
- Если экспериментальный тип — часть реализации, то:
  - Propagate, если экспериментальная аннотация принадлежит не вам
  - Non-propagate, если вы контролируете экспериментальность этого API
- Не используйте module opt-in (`-opt-in=org.mylibrary.OptInAnnotation`)

Одна аннотация или несколько?

`kotlinx.serialization:` *`@ExperimentalSerializationApi`*

Одна аннотация или несколько?

`kotlinx.serialization:` *`@ExperimentalSerializationApi`*

`kotlinx.coroutines:` *`@ExperimentalCoroutinesApi, @FlowPreview`*

Одна аннотация или несколько?

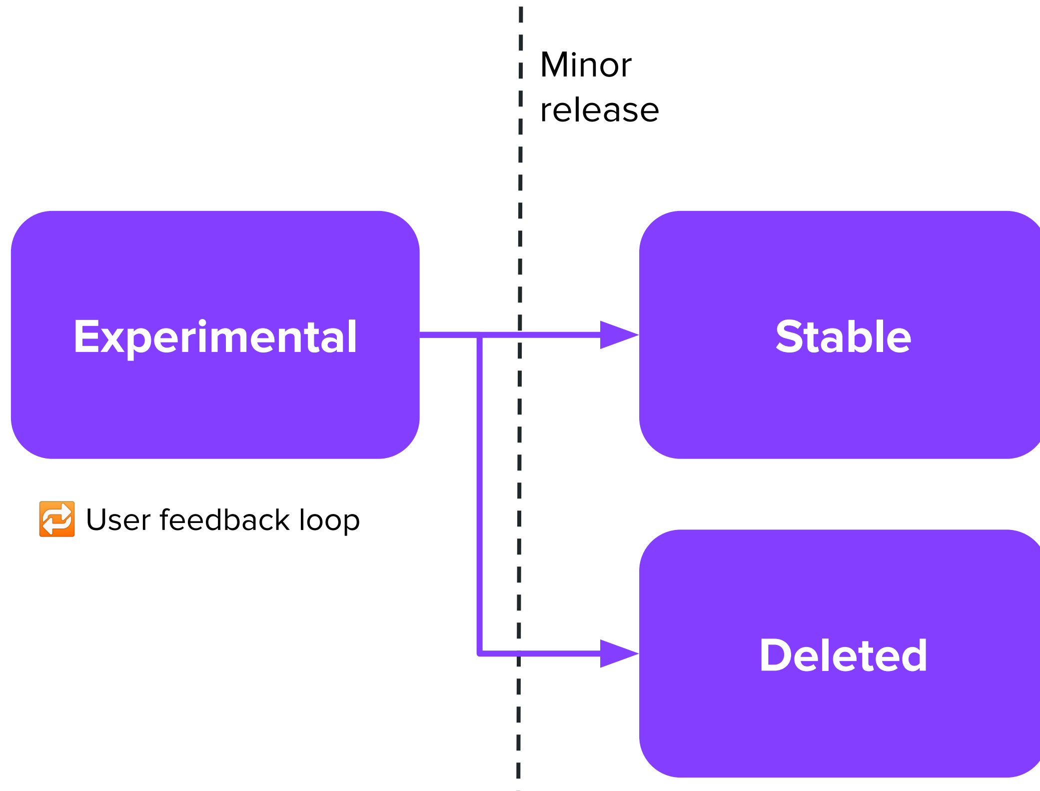
kotlinx.serialization: *@ExperimentalSerializationApi*

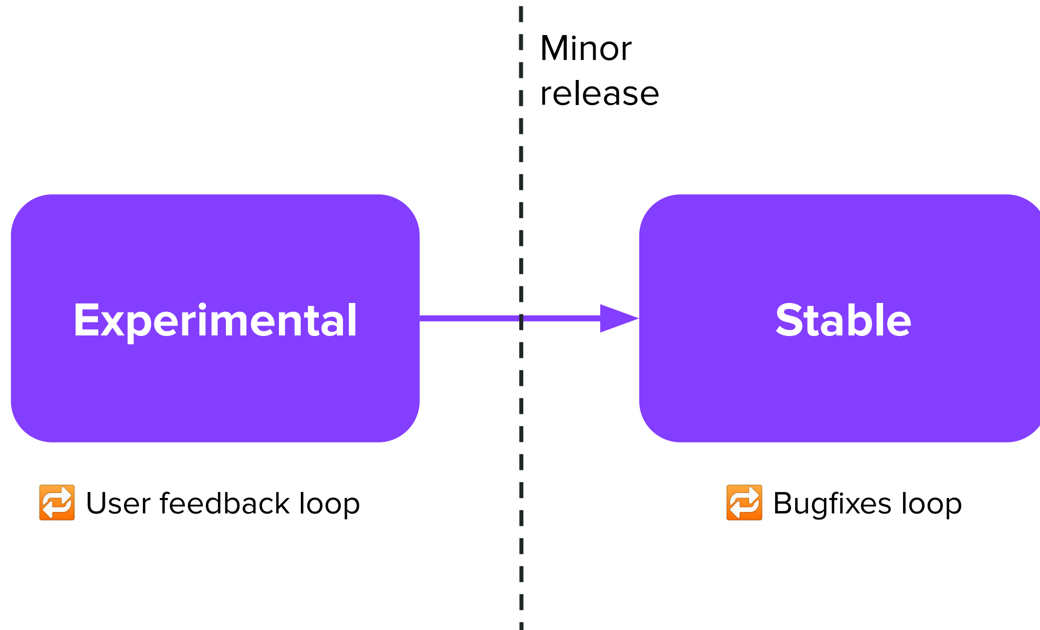
kotlinx.coroutines: *@ExperimentalCoroutinesApi*, *@FlowPreview*

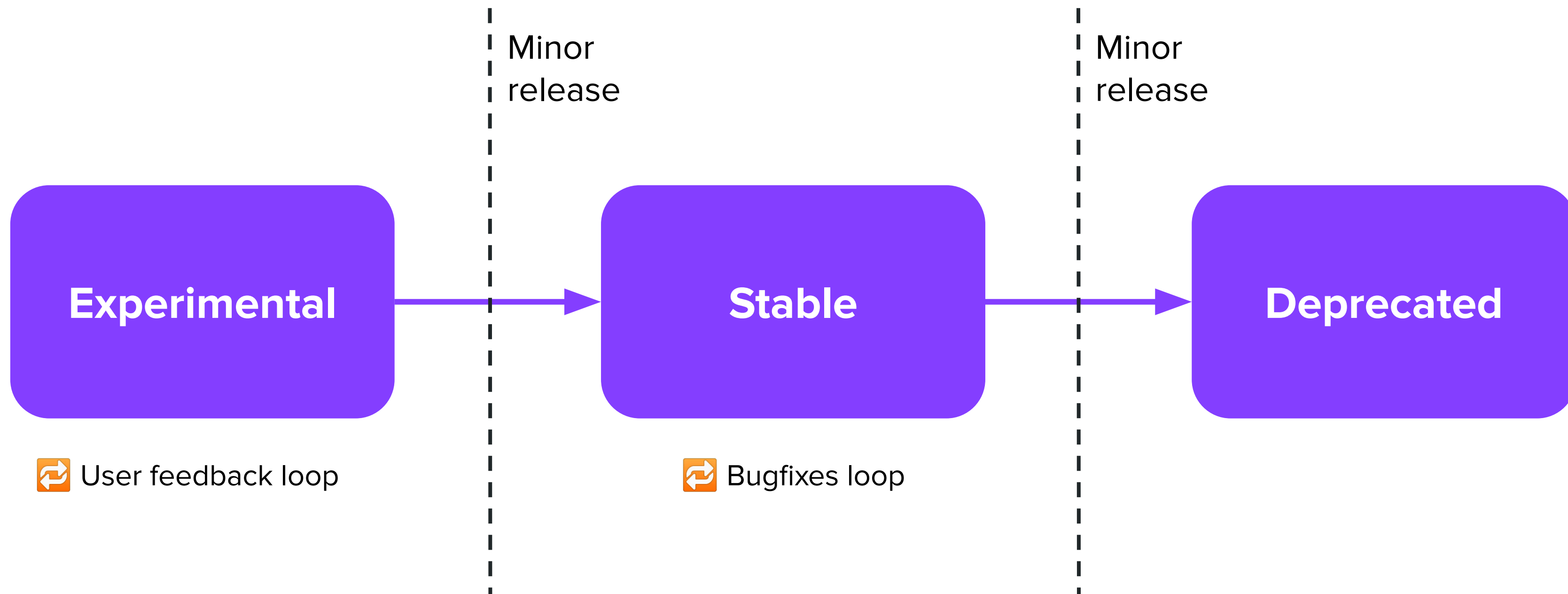
kotlin.stdlib: *@ExperimentalTime*,

@ExperimentalUnsignedTypes, *@ExperimentalStdlibApi*

Experimental


Minor
release
Experimental
Stable
Deleted
User feedback loop


Minor
release
Experimental
Stable
🔁 User feedback loop
🔁 Bugfixes loop


Minor
release
Minor
release
Experimental
Stable
Deprecated
🔁 User feedback loop
🔁 Bugfixes loop

Thanks for joining

and have a nice Kotlin!