Циан

Хватит придавать смысл null



Перевалов Данил,
Android-разработчик

Циан – ведущая цифровая платформа операций с недвижимостью

~20 млн
уникальных пользователей в месяц

2,1 млн
актуальных объявлений

99%
чистота базы объявлений

25 лет
экспертизы

15,2 млрд руб.
выручка за 12 месяцев 2025 года выросла на 16,7%

Топ-15
крупнейших сайтов недвижимости в мире по версии Similarweb

1 000+
сотрудников

№1 💗
proptech в инклюзии: **~6 тыс.** новых объявлений в день содержат параметры, важные для людей на колясках

О чём сегодня расскажу

Циан

01 Что такое null и откуда он взялся

02 Проблемы null: придание смысла

03 Функциональная обработка ошибок

04 Принципы проектирования

05 Итоги



Немного истории



Ошибка на миллиард долларов



Источник wikimedia.org

Тони Хоар, придумал null в 1965 году, в ALGOL W.
Позже он назвал это своей «ошибкой на миллиард долларов».



В 1965 году Хоар добавил null прямо в систему типов языка.

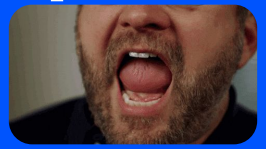
«Так было проще всего реализовать»



В 1965 году Хоар добавил null прямо в систему типов языка.

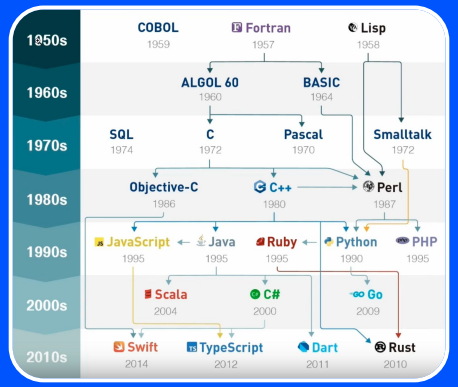
«Так было проще всего реализовать» —

и это переняли все



Источник https://giphy.com/

И это расползлось по всем языкам. За ALGOL null переняли десятки языков



Источник wikimedia.org

NPE — самая частая ошибка на JVM

Годами возглавляет топ исключений на проде. Каждое обращение к ссылке — потенциальное падение.



Что такое null?



> In computing, a null pointer or null reference is a value saved for indicating that the pointer or reference does not refer to a valid object.



Чувак написавший страничку
про null на Wikipedia

1. null — это значение по умолчанию для ссылочных типов данных
2. null — это «отсутствие значения»



Чувак написавший страничку
про null на JavaRush

Что это значит для нас?
Как Java/Kotlin разработчиков

Циан

1

null — это **константа** языка программирования

2

У null **уже есть смысл** определённый языком программирования

3

Во всех местах null должен иметь один и тот же смысл

4

На null нельзя дописать документацию или как-то ещё **придать ему дополнительные смыслы**

Проблемы null: придание смысла



Пример

Исходная модель

Циан

```
data class OriginModel(
    val id: String? = null,
    val text: String? = null,
    // Ещё какой-то менее важный код
)
```

Пример

Нужная модель

Циан

```
data class MappedModel(
    val id: String,
    val text: String? = null,
    // Ещё какой-то менее важный код
)
```

Пример

Интерфейс

Циан

```
interface Mapper {

    fun map(origin: OriginModel): MappedModel?

}
```

Пример

Интерфейс

Циан

```
fun main() {
    val mapper = MapperImpl()

    val origin = OriginModel()
    val mapped = mapper.map(origin)
    println(mapped)
}
```

В консоли написано — null



Источник https://giphy.com/gifs/math-lady-meme-WRQBXSCnEFJIuxktnw

Мемные диалоги
Первый вопрос

Циан

Вопрос:

```
val origin = OriginModel()
val mapped = mapper.map(origin)
```

Ответ: null

Вопрос:

Что-то случилось?

Ответ: Да ничё

Мемные диалоги
Второй вопрос

Циан

Вопрос:

```
val origin = OriginModel(id = null)
val mapped = mapper.map(origin)
```

Ответ: null

Вопрос:

Ну что-то же произошло?

Ответ: Да ничего не случилось

Мемные диалоги

Третий вопрос

Циан

Вопрос:

```
val origin = OriginModel(id = “”)
val mapped = mapper.map(origin)
```

Ответ: null

Вопрос:

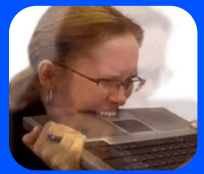
Ну что-то же произошло?

Ответ: Да ничё, я же сказала

Ты видишь что, что-то не так,
но что именно, понять невозможно.
Из ответа дополнительной
информации выудить
не получается

Приходится идти разбираться Потому, что вообще ничего не понятно





Источник https://giphy.com/marvelstudios

Mapper
Разбираемся в коде

Циан

```
override fun map(origin: OriginModel): MappedModel? {
    if (origin.id == null) {
        return null
    }
    if (origin.id == "") {
        return null
    }
    return MappedModel(
        id = origin.id,
        text = origin.text
        // Ещё какой-то менее важный код
    )
}
```

Mapper

Смотрим глубже

Циан

```
override fun map(origin: OriginModel): MappedModel? {
    if (origin.id == null) {
        return null // id не может быть null
    }
    if (origin.id == "") {
        return null // id не может быть пустой строкой
    }
    return MappedModel(
        id = origin.id,
        text = origin.text
        // Ещё какой-то менее важный код
    )
}
```

Таких проверок может быть больше

Циан

```
if (origin.id == null) {
    return null // id не может быть null
}
if (origin.id == "") {
    return null // id не может быть пустой строкой
}
if (origin.text == null) {
    return null // text не может быть null
}
if (origin.text == "") {
    return null // text не может быть пустой строкой
}
```

Разработчик придал смысл null. И этот смысл отличается от устоявшегося



В моём случае, null значит, что id некорректен. Например, он null или пустая строка



Источник https://giphy.com/thesimpsons

Ситуация
Не очень хорошая



Один уверен, что всё хорошо
Ведь в его голове всё было просто, понятно и логично.

Источник https://giphy.com/laff_tv

Ситуация
Не очень хорошая



Один уверен, что всё хорошо
Ведь в его голове всё было просто, понятно и логично.



Второй офигевает
Потому что он ничего не понимает и ему приходится с нуля разбираться.

Источник https://giphy.com/laff_tv

Что можно с этим сделать?

Простые решения
Документация

Циан

```
interface Mapper {

    /**
     * @return  null если исходная модель не валидна
     */
    fun map(origin: OriginModel): MappedModel?
}
```

Цитата



Код должен быть самодокументируемым и по нему должно быть сразу понятно, что он делает.

— Разработчики-теоретики



Цитата

Не согласен!

— Данил

Мы не пишем идеальный код
Никогда

Циан

Из-за того что:

1

Вряд ли у нас есть такие навыки,
их нет ни у кого

Мы не пишем идеальный код
Никогда

Циан

Из-за того что:

1

Вряд ли у нас есть такие навыки,
их нет ни у кого

2

Вам **не выделят**
на это время

Мы не пишем идеальный код
Никогда

Циан

Из-за того что:

1

Вряд ли у нас есть такие навыки,
их нет ни у кого

2

Вам **не выделят**
на это время

3

Вы сами не захотите тратить
время на идеальный код
на каком-то захудалом
экранчике **который увидят**
от силы 5% пользователей

Документация лишней не бывает
Да, даже устаревшая

Циан

Я не хочу разбираться в коде, который:



Писался 5 лет назад
с устаревшим подходом



Писался в 11 ночи,
ведь завтра дедлайн



Писался по-быстрому,
ведь через 20 минут финал
лиги чемпионов

Прямо объявить обязательные параметры

Циан

```
interface Mapper {

    /**
     * @param  id — идентификатор модели, не может быть пустой строкой
     *
     * @return  null если исходная модель не валидна
     */
    fun map(id: String, origin: OriginModel): MappedModel?
}
```

Кстати, раз уж мы тут собрались

Mapper
Странный, странный код

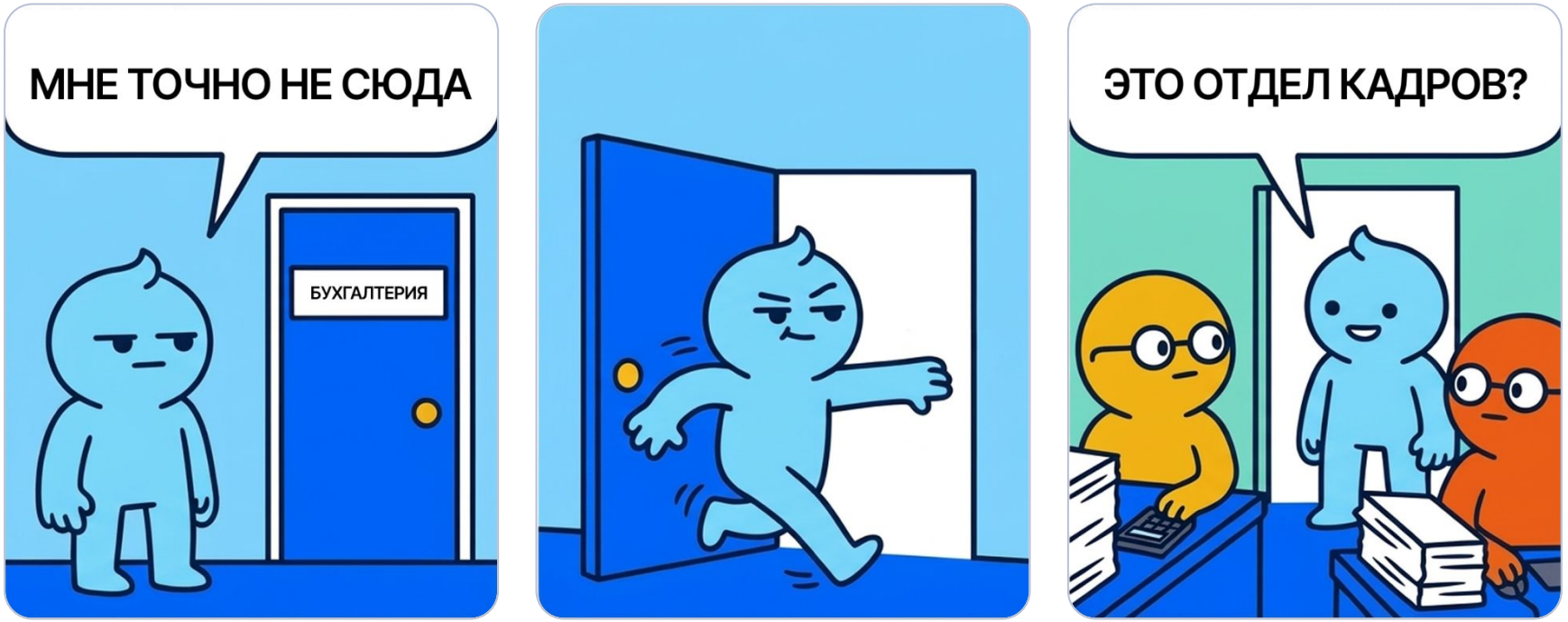
Циан

```
override fun map(origin: OriginModel?): MappedModel? {
    if (origin == null) {
        return null
    }
    // Ещё какой-то менее важный код
}
```

Это лишнее действие

Аналогия

Циан



Проверяем снаружи

Циан

```
fun main() {
    val mapped = origin?.let { mapper.map(origin) }
    println(mapped)
}
```

Разбираемся в коде

Циан

```
override fun map(origin: OriginModel): MappedModel? {
    // Ещё какой-то менее важный код
}
```

Цитата



«А как тогда быть с полиморфизмом?»

— Вы



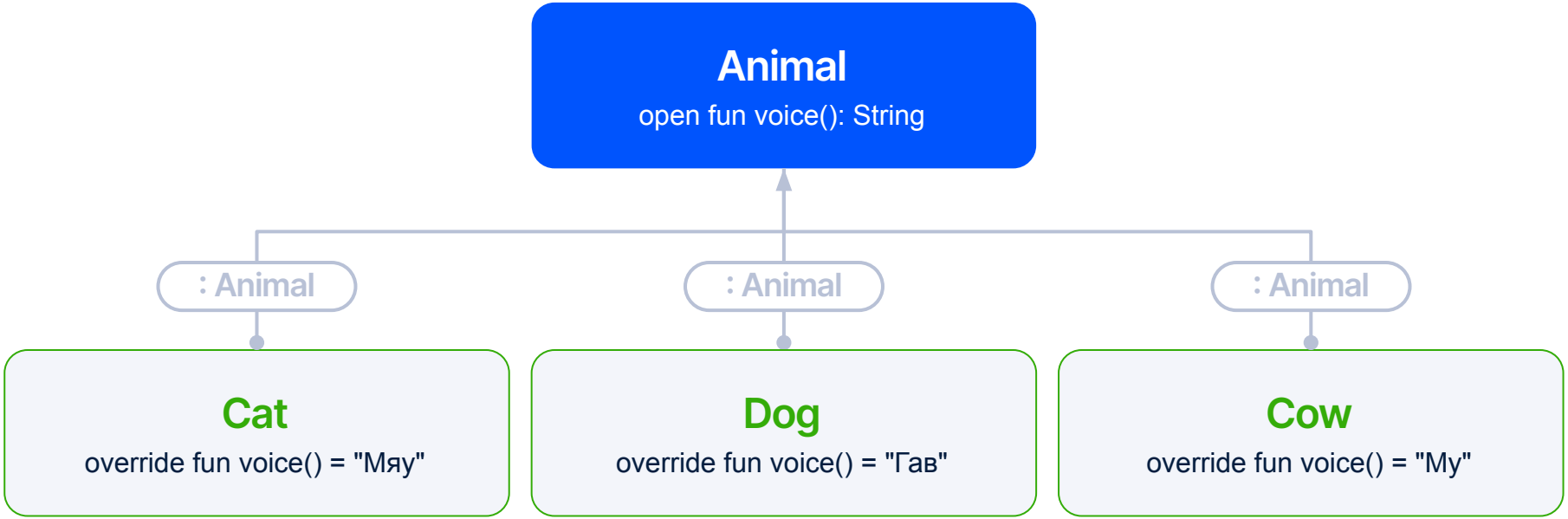
Полиморфизм и null



Полиморфизм

Один интерфейс — разные реализации

Циан



Вызывающий работает с типом Animal и зовёт voice() —
какая реализация сработает, решает конкретный подкласс

Полиморфизм
Интерфейс

Циан

```
interface Mapper {

    fun map(origin: OriginModel): MappedModel?

}
```

История интерфейса
Возможно, грустная

Циан

1

Написан 5 лет назад
Писал его человек который давно уволился

История интерфейса
Возможно, грустная

Циан

1

Написан 5 лет назад
Писал его человек который давно уволился

2

Непонятный мейнтейнер
Кто сейчас ответственен за него непонятно

История интерфейса
Возможно, грустная

Циан

1

Написан 5 лет назад
Писал его человек который давно уволился

2

Непонятный мейнтейнер
Кто сейчас ответственен за него непонятно

3

Много наследников
30+ классов наследующихся от этого интерфейс

Последствия
С которыми приходится жить

Циан

Документацию так просто не добавить, так как она не будет описывать всех наследников

Список параметров не изменить, да и для некоторых наследников id может быть необязательным

Если что-то поменять в интерфейсе, то придётся вносить правки **во всех наследниках**

Что можно с этим сделать?

Throwable
Mapper

Циан

```
override fun map(origin: OriginModel): MappedModel? {
    requireNotNull(origin.id) { "Id не может быть null" }
    require(origin.id != "") { "Id не может быть пустой строкой" }
    return MappedModel(
        id = origin.id,
        text = origin.text
        // Ещё какой-то менее важный код
    )
}
```

Throwable
Место вызова

Циан

```
fun main() {
    val mapper = MapperImpl()

    val origin = OriginModel()
    val mapped = try {
        mapper.map(origin)
    } catch (exception: IllegalArgumentException) {
        println("Error mapping origin: ${exception.message}")
        null
    }
    println(mapped)
}
```

Полиморфизм
Интерфейс

Циан

```
interface Mapper {

    /**
    * @throws IllegalArgumentException
    */
    fun map(origin: OriginModel): MappedModel?

}
```



Источник https://giphy.com/sunnyfxx

Проблемы

Циан



Не всё прям ошибка

Не всегда невозможность смаппить объект стоит интерпретировать именно как ошибку с выводом Stacktrace в лог и т.п.



Проблемы

Циан

Не всё прям ошибка

Не всегда невозможность смаппить объект стоит интерпретировать именно как ошибку с выводом Stacktrace в лог и т.п.

Есть шанс пропустить реальный Throwable

Вполне возможны ситуации когда вы реально хотите упасть или показать полноэкранную ошибку в случае неподходящего параметра. **Придётся городить костыли**, например заводить дополнительный тип Exception или определять по message.

Функциональная обработка ошибок



Пример из приложения

NewBuildingFlatStatsMapper

Циан

```
internal class NewBuildingFlatStatsMapper {

    override fun createModel(
        params: NewBuildingFlatStatsParamsHolder
    ): NewBuildingFlatStatsUiModel? {
        val newBuilding = params.newBuilding

        if (newBuilding.flatTypesAgent.isEmpty()
                && newBuilding.flatTypesDeveloper.isEmpty()) {
            return null
        }
        // Ещё какой-то менее важный код
    }
}
```

Result

Ошибка как значение
сбой становится частью типа

Циан

```
internal class NewBuildingFlatStatsMapper {

    override fun createModel(
        params: NewBuildingFlatStatsParamsHolder
    ): Result<NewBuildingFlatStatsUiModel> {
        // Ещё какой-то менее важный код
    }
}
```

runCatching — но внутри зашит Throwable

Циан

```
val result: Result<User> = runCatching { parse(raw) }

result
    .onSuccess { user -> show(user) }
    .onFailure { e -> log(e) }
```

Применяем Result
NewBuildingFlatStatsMapper

Циан

```
internal class NewBuildingFlatStatsMapper {

    override fun createModel(
        params: NewBuildingFlatStatsParamsHolder
    ): Result<NewBuildingFlatStatsUiModel> {
        val newBuilding = params.newBuilding

        if ((newBuilding.flatTypesAgent + newBuilding.flatTypesDeveloper).isEmpty()) {
            return Result.failure(IllegalArgumentException("flatTypesAgent
            и flatTypesDeveloper не должны быть пустыми одновременно"))
        }
        // Ещё какой-то менее важный код
    }
}
```

Sealed Class



Применяем SealedClass
NewBuildingResult

Циан

```
internal sealed interface NewBuildingResult<T> {
    data class Success<T>(val model: T) : NewBuildingResult<T>
    data class Skip<T>(val reason: String) : NewBuildingResult<T>
    data class Failure<T>(val error: Throwable) : NewBuildingResult<T>
}
```

Применяем SealedClass
NewBuildingFlatStatsMapper

Циан

```
internal class NewBuildingFlatStatsMapper {
    override fun createModel(
        params: NewBuildingFlatStatsParamsHolder
    ): NewBuildingResult<NewBuildingFlatStatsUiModel> {
        val newBuilding = params.newBuilding
        if ((newBuilding.flatTypesAgent + newBuilding.flatTypesDeveloper).isEmpty()) {
            return Skip("flatTypesAgent и flatTypesDeveloper
                    не должны быть пустыми одновременно")
        }
        if (newBuilding.id == null) {
            return Failure(IllegalArgumentException("id не должно быть null"))
        }
        // Ещё какой-то менее важный код
}
```

Arrow



Left — ошибка, Right — успех

Циан

```
sealed interface ParseError
object Empty : ParseError
data class BadId(val raw: String) : ParseError

fun parse(raw: String): Either<ParseError, User>

when (val r = parse(raw)) {
    is Right -> show(r.value)
    is Left  -> handle(r.value)    // r.value: ParseError
}
```

Null Object/ Empty Object



Null Object

Циан

```
interface Discount {
    fun apply(price: Int): Int
}

object NoDiscount : Discount {
    override fun apply(price: Int) = price
}

val finalPrice = order.discount.apply(price)
```

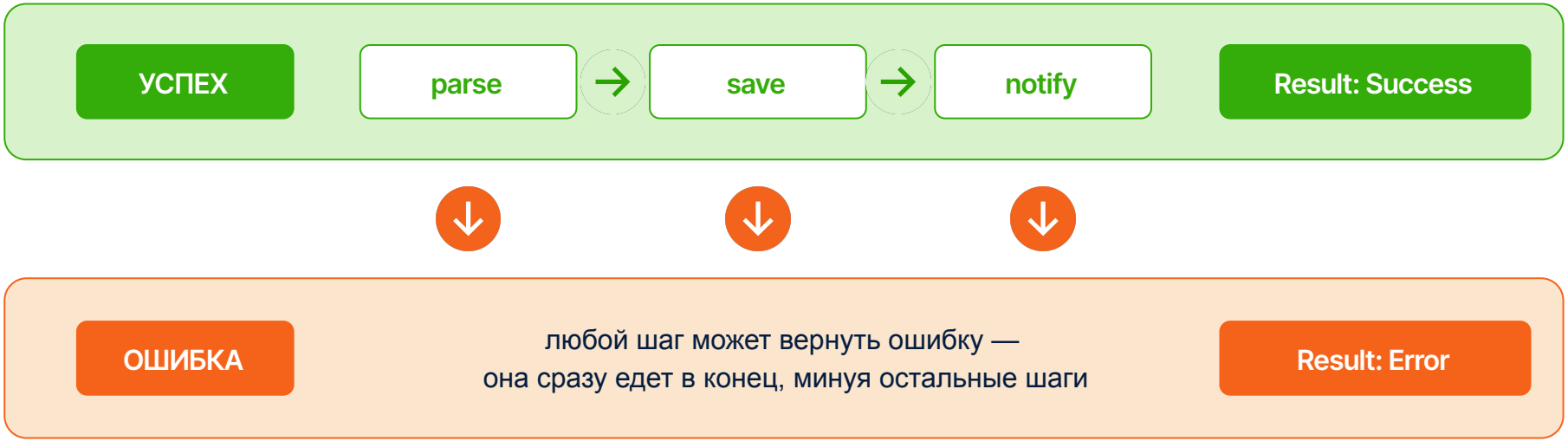
Railway



Railway-oriented programming

ошибки едут по параллельному пути — никаких if на каждом шаге

Циан



Успех течёт по верхнему пути через все шаги; ошибка перескакивает на нижний и минует остальное — **это и есть «две рельсы» railway-oriented programming**

Ошибка едет мимо остальных шагов

Циан

```
fun handle(raw: String): Either<Error, Unit> =
    parse(raw)                          // Either<Error, User>
        .flatMap { save(it) }           // вызовется, только если parse ok
        .flatMap { notify(it) }         // и только если save ok
        .map { }
// любая ошибка сразу «съезжает» в конец цепочки
```

Когда это оверинжиниринг

На Android часто хватает sealed class

Циан

```
sealed interface UiState {
    data class Content(val data: Flats) : UiState
    object Loading : UiState
    data class Error(val message: String) : UiState
}
```

Принципы проектирования

Parse, don't validate

Принцип 1

Циан

```
// validate: проверили — и забыли, тип не изменился
fun validate(input: String)    // бросит, если плохо
val name: String = input       // всё ещё «просто String»

// parse: проверили — и вернули более точный тип
fun parse(input: String): Email?
// Email корректен по построению — проверка не теряется
```

От Raw к Valid

Парсим один раз на входе

Циан

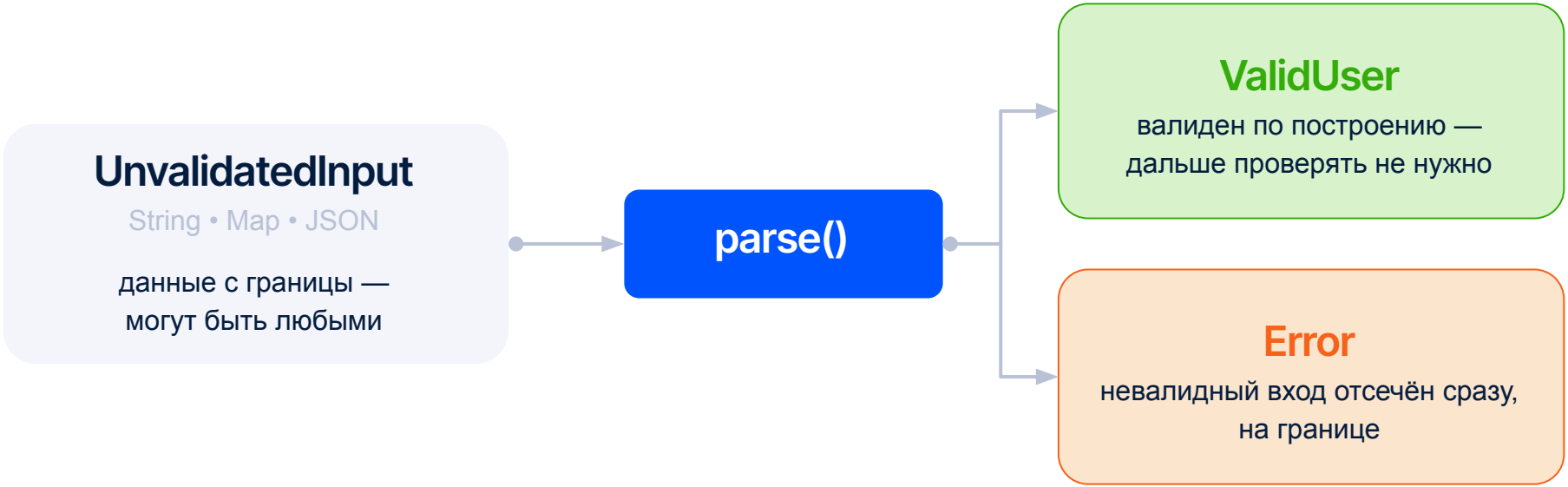
```
data class RawUser(val name: String?, val age: String?)
data class ValidUser(val name: String, val age: Int)

fun parse(u: RawUser): Either<Error, ValidUser> {
    val name = u.name?.takeIf { it.isNotBlank() } ?: return Left(NoName)
    val age = u.age?.toIntOrNull() ?: return Left(BadAge)
    return Right(ValidUser(name, age))
}
// дальше по коду ходит только ValidUser — без проверок
```

Parse, don't validate

Проверяй на входе — и возвращай тип, который уже не может быть невалидным

Циан



Валидация проверяет и забывает.
Парсинг проверяет и сохраняет результат в типе — **и этот факт уже нельзя потерять по пути**

Make illegal states unrepresentable

Принцип 2. Невозможное не компилируется

Циан

```
// ПЛОХО: можно собрать бессмысленное состояние
data class Screen(
    val Loading: Boolean,
    val data: Flats?,
    val error: String?,
)
// loading и error и data одновременно?
```

Make illegal states unrepresentable

Принцип 2. Невозможное не компилируется

Циан

```
// ПЛОХО: можно собрать бессмысленное состояние
data class Screen(
    val Loading: Boolean,
    val data: Flats?,
    val error: String?,
)
// loading и error и data одновременно?
```

```
// ХОРОШО: недопустимое просто не выразить
sealed interface Screen {
    object Loading : Screen
    data class Content(val data: Flats) : Screen
    data class Failure(val error: String) : Screen
}
```

Fail-fast offensive programming

Принцип 3. Падай рано на своих ошибках

Циан

```
fun setAge(age: Int) {
    require(age >= 0) { "возраст не может быть < 0" }
    // ...
}
```

Проектируй типы так, чтобы неверное состояние было невозможно



null — только на границах



Не придавайте null смысл — делайте отсутствие значения и ошибку явными в типах



Вопросы?



Спасибо за внимание

Телеграм: