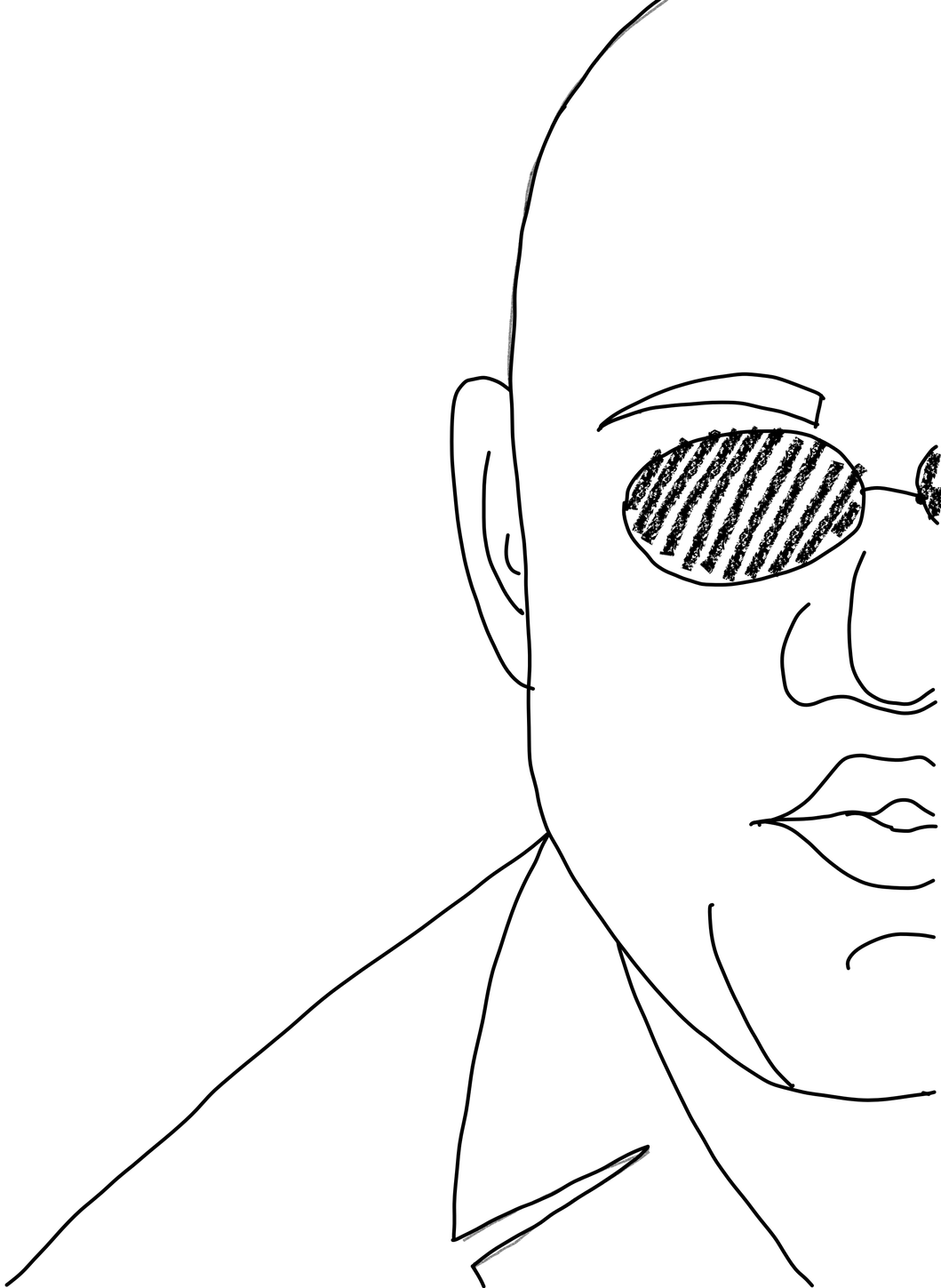
auto.ru

Итак, вы выбрали UDF-архитектуру. Как моделировать стейт?



На какие вопросы будем отвечать

1. Зачем нам Unidirectional Data Flow?
2.
3.
4.
5.

На какие вопросы будем отвечать

1. Зачем нам Unidirectional Data Flow?
2. Почему Object Oriented Programming не подходит для стейта?
3.
4.
5.

На какие вопросы будем отвечать

1. Зачем нам Unidirectional Data Flow?
2. Почему Object Oriented Programming не подходит для стейта?
3. Как Algebraic Data Types заменят нам объекты?
4.
5.

На какие вопросы будем отвечать

1. Зачем нам Unidirectional Data Flow?
2. Почему Object Oriented Programming не подходит для стейта?
3. Как Algebraic Data Types заменят нам объекты?
4. Как выражать доменные правила в структуре стейта?
5.

На какие вопросы будем отвечать

1. Зачем нам Unidirectional Data Flow?
2. Почему Object Oriented Programming не подходит для стейта?
3. Как Algebraic Data Types заменят нам объекты?
4. Как выражать доменные правила в структуре стейта?
5. Зачем думать о стейте как о базе данных?


Образ мышления
для работы со стейтом

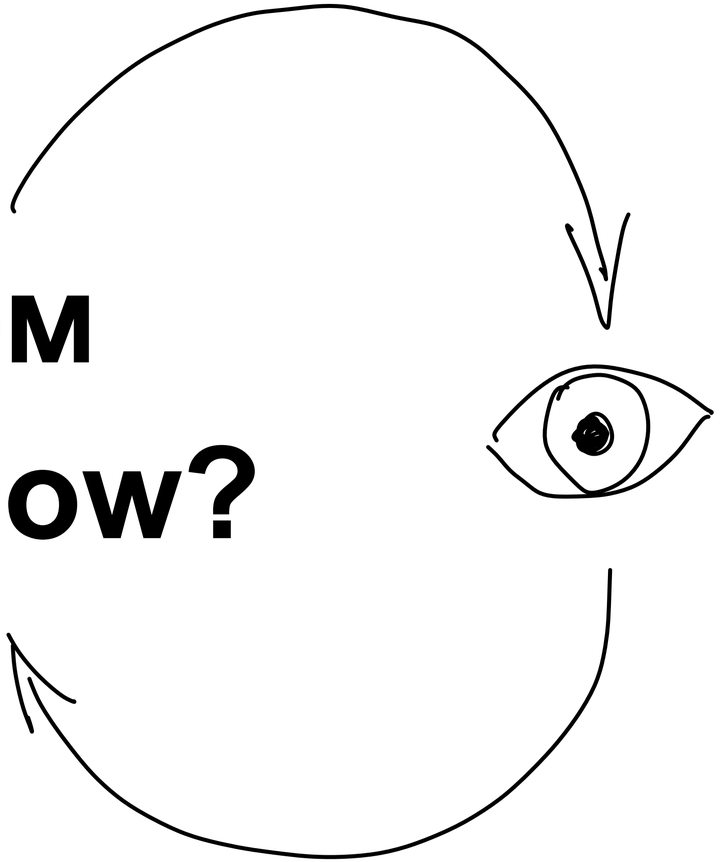
Kotlin? Java? Swift? Dart?

Не проблема!

вело.ru

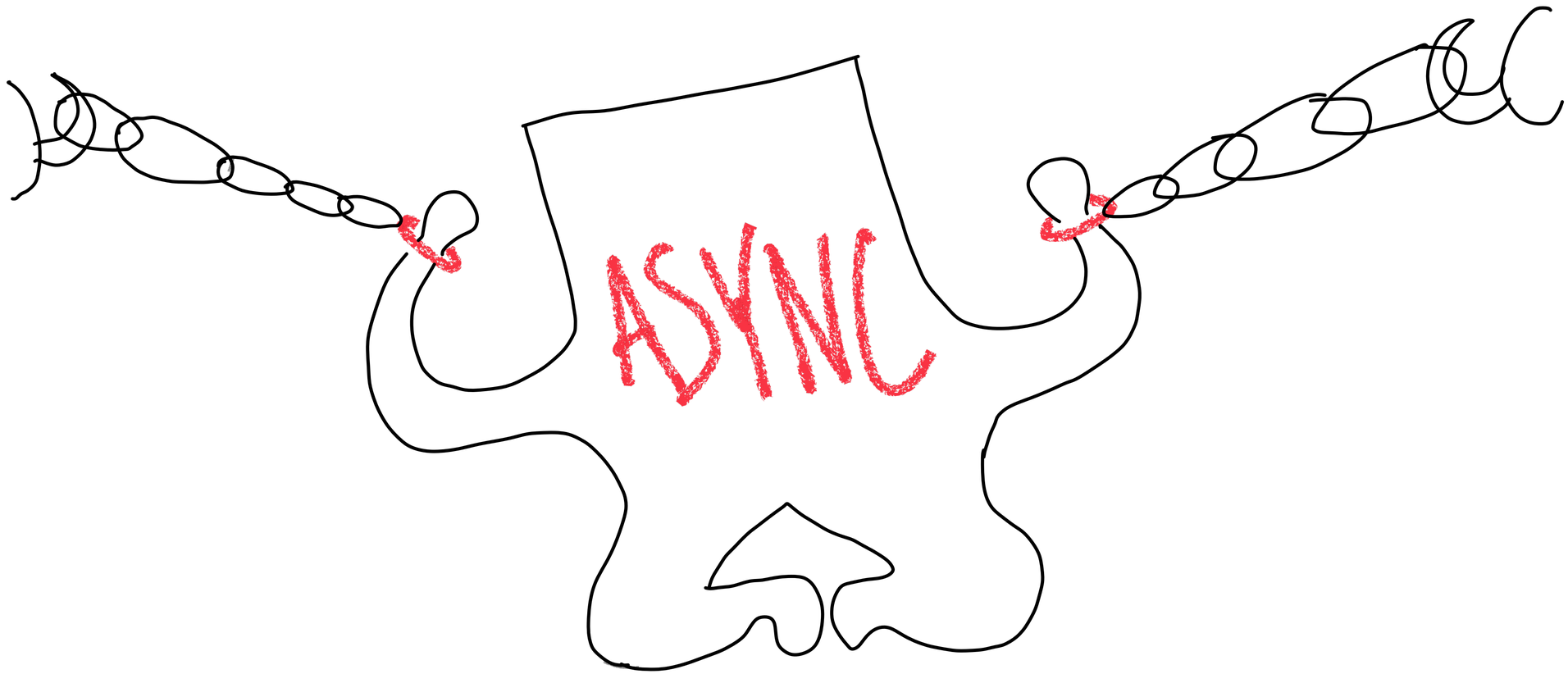
Восходящая звезда на рынке б/у велосипедов

Почему мы выбираем Unidirectional Data Flow?



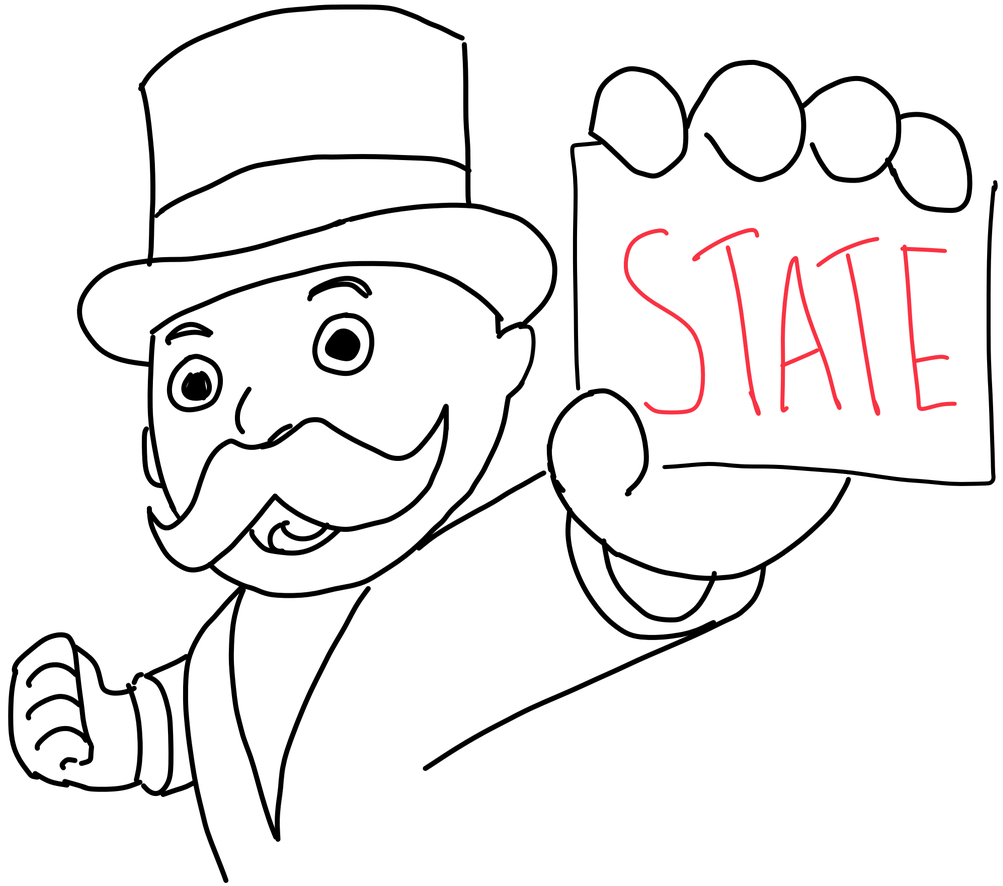
Почему мы выбираем UDF?

Контроль над изменениями состояния



Почему мы любим UDF?

Единый источник правды для всей бизнес-логики

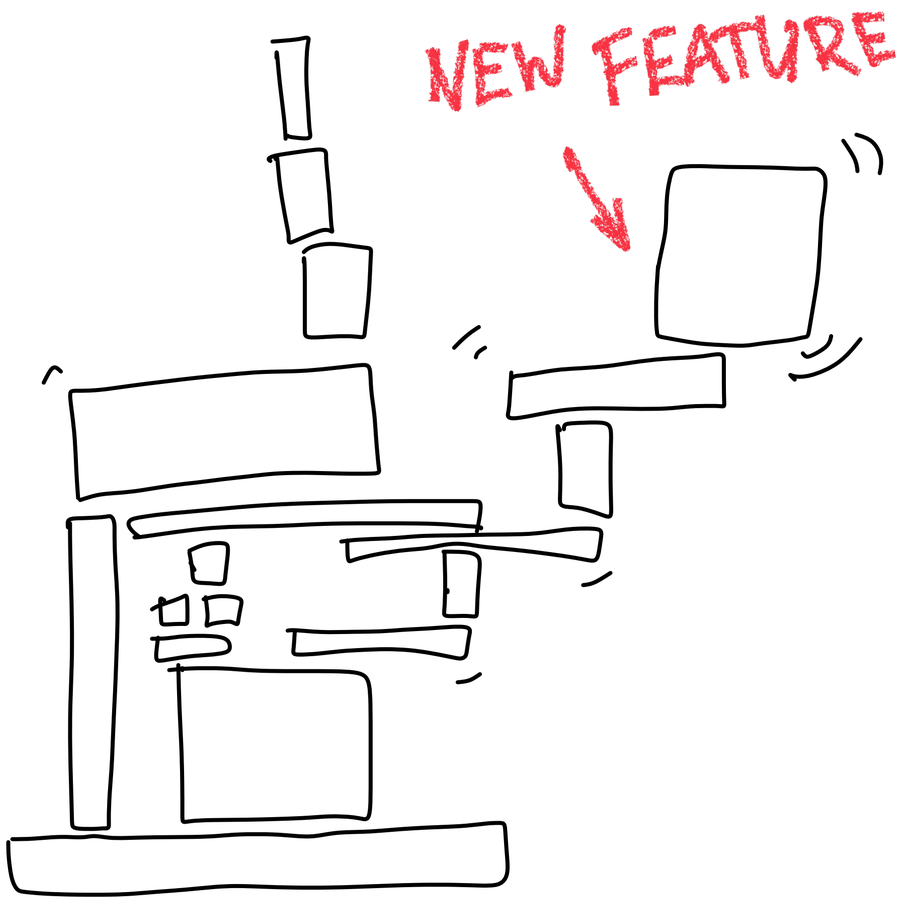


Почему мы выбираем UDF?

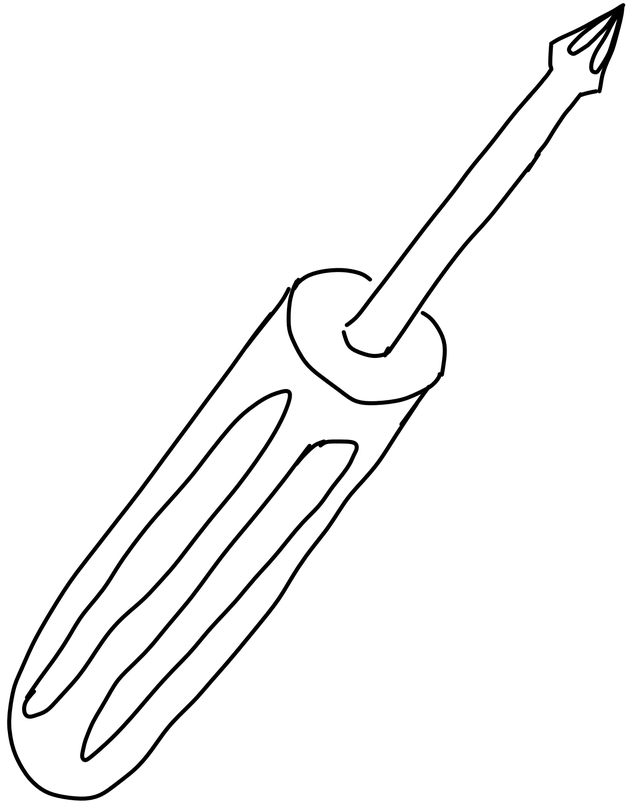
Декларативный UI

Большой Trade-Off

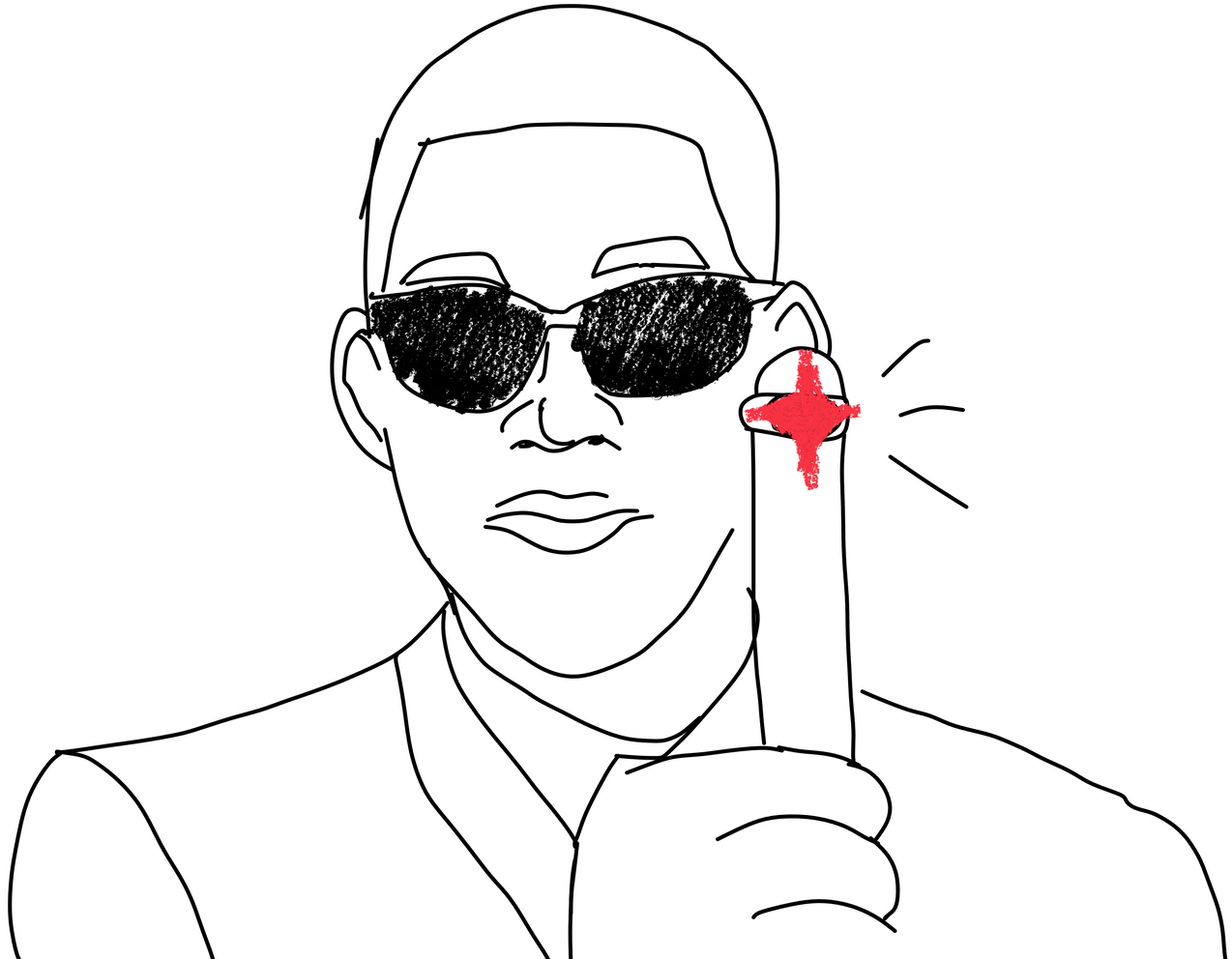
Big Stinky State



Подготовим инструменты



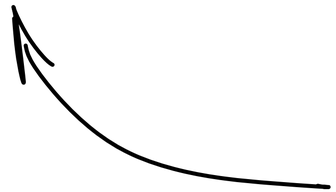
Шаг 1: разучитесь применять ООП



Состояние — не объект!

Не делайте так!

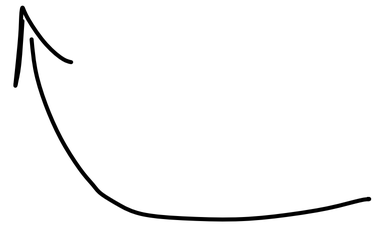
```
interface State {
  fun nextState(msg): State
}
```



Загрузка знает, как создать готовый стейт?!

Не делайте так!

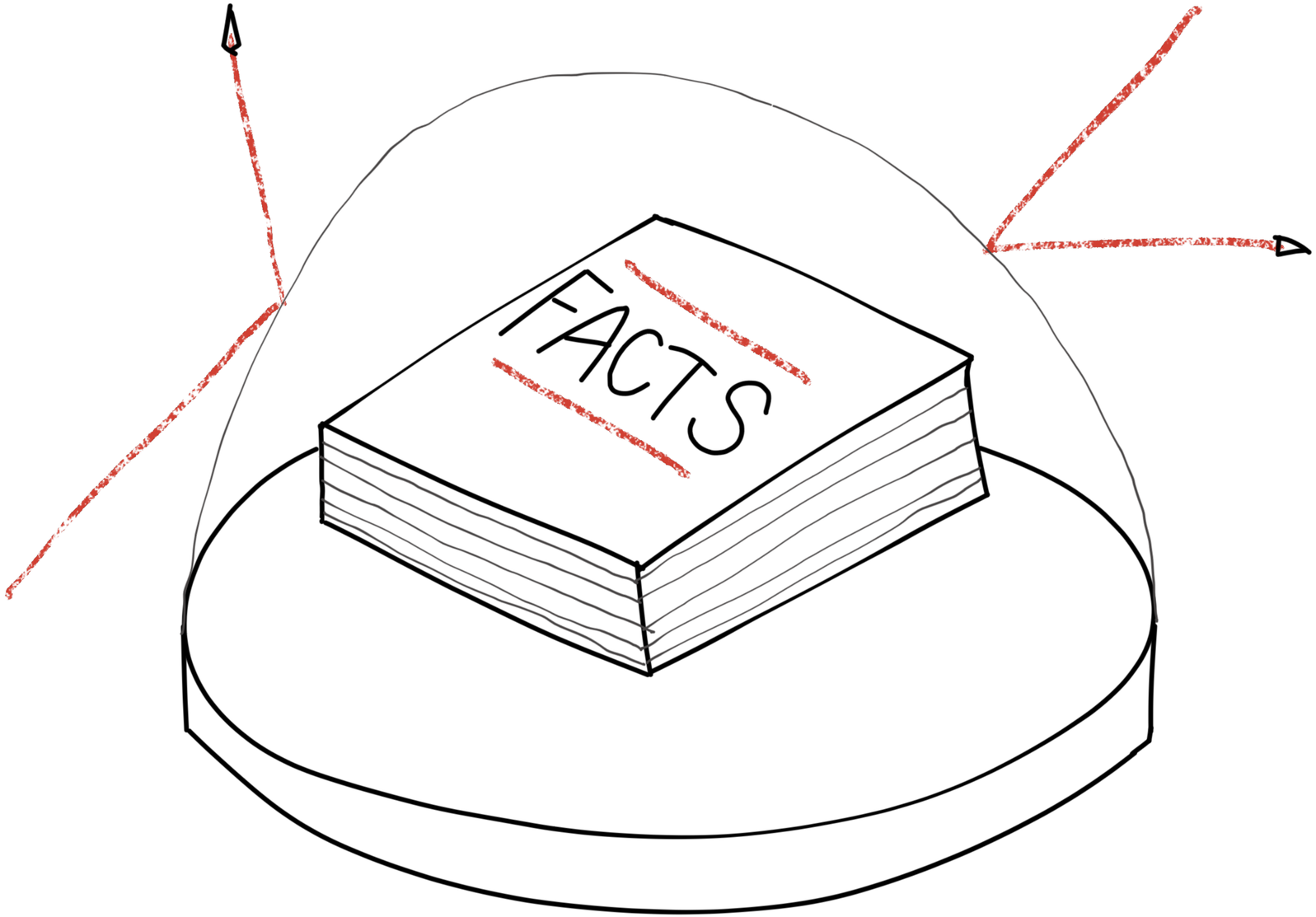
```
interface PaymentState {
  fun checkout(paymentService)
}
```



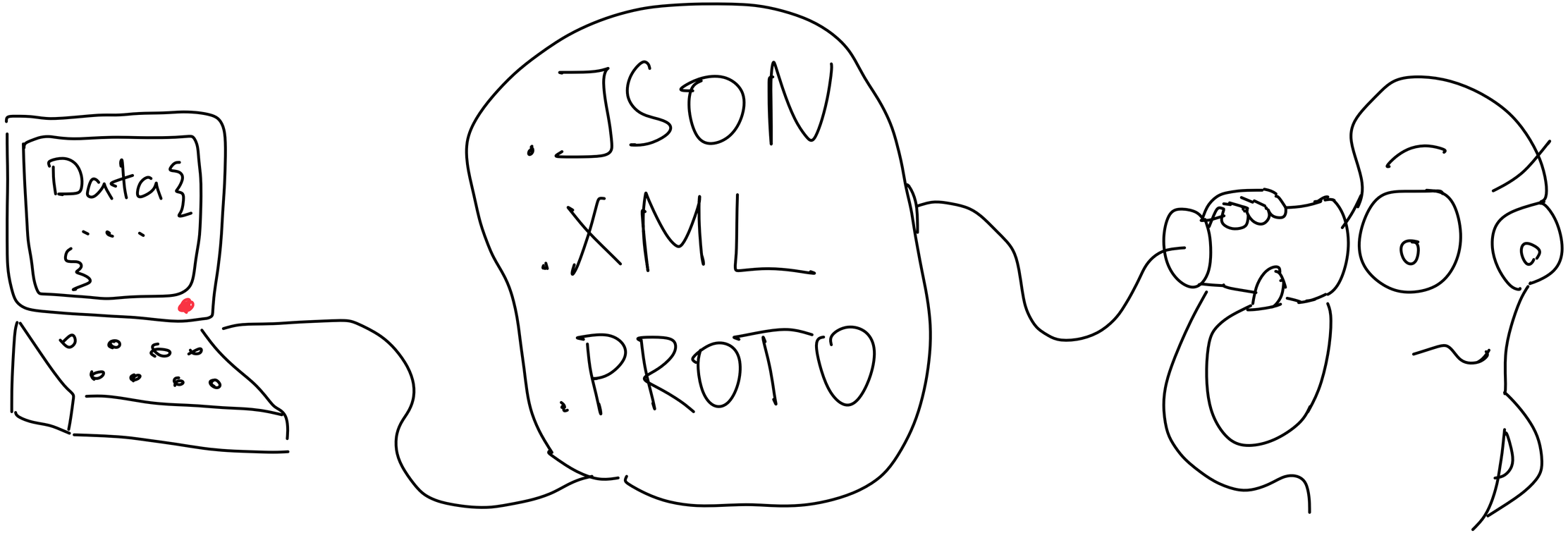
Состояние экрана оплаты знает, как совершить оплату?!

Состояние —это данные

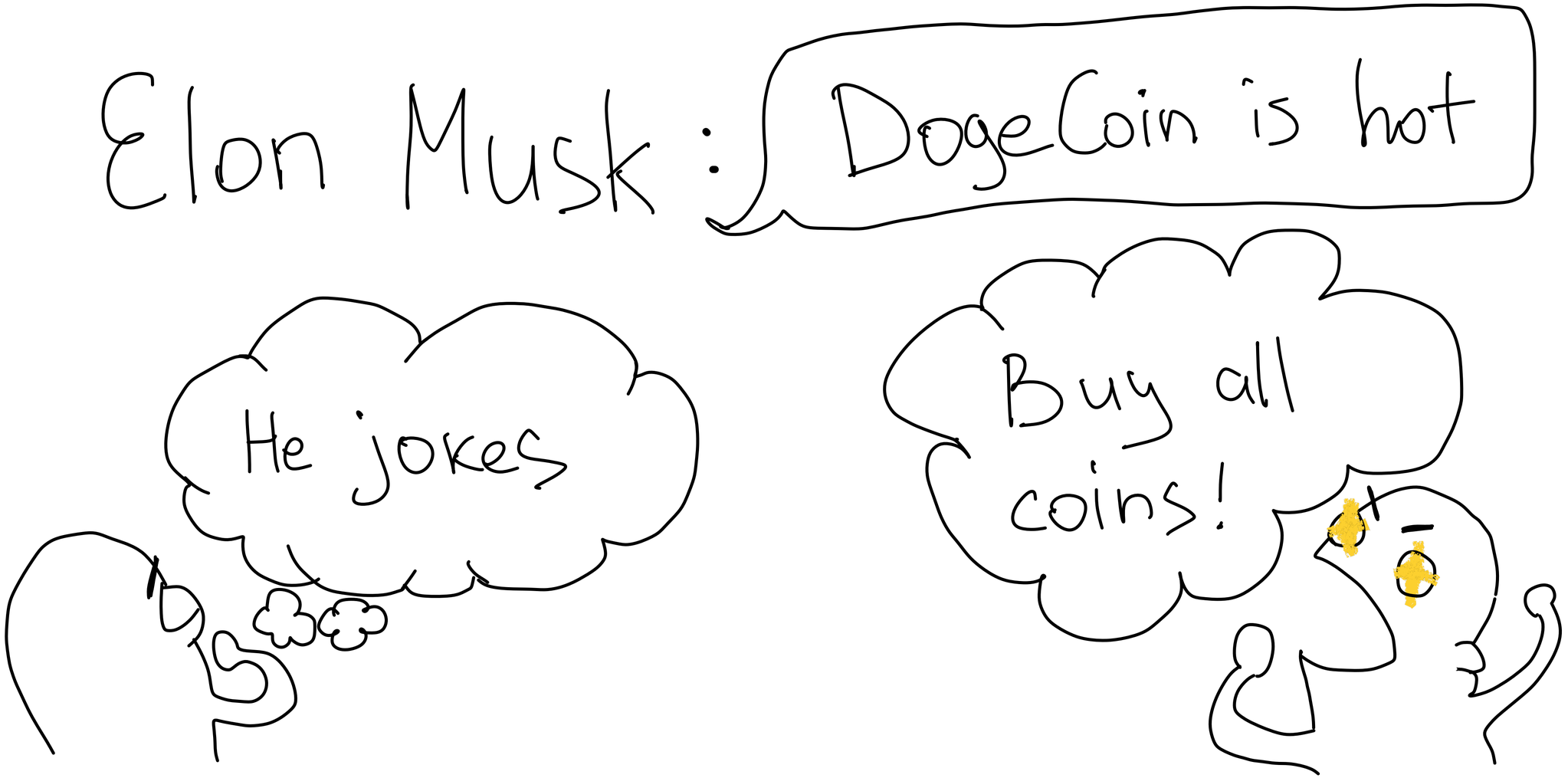
Данные — неизменяемые факты о системе



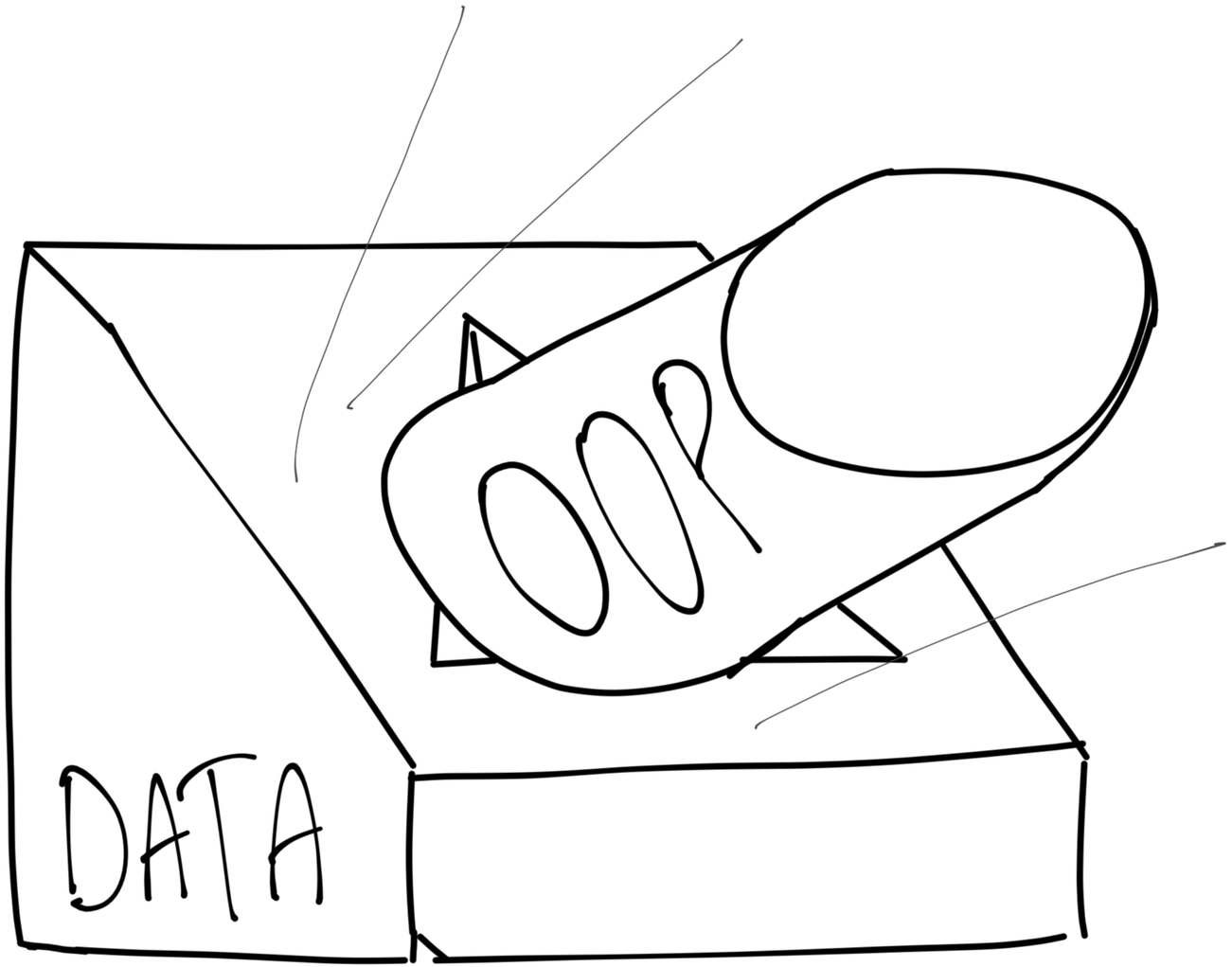
Данные сериализуются



Данные открыты для интерпретации



Так почему ООП не подходит?



Приватные члены класса

не нужны!

Если данные нельзя изменить
— их нечего скрывать

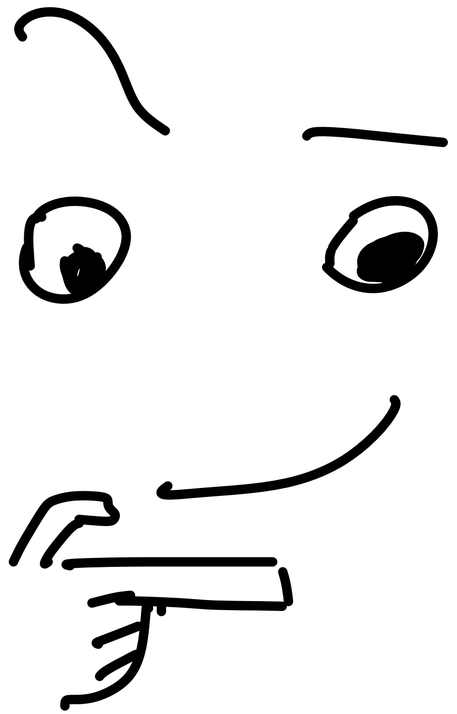
Identity и равенство по ссылке

Факты не различимы
между своими копиями

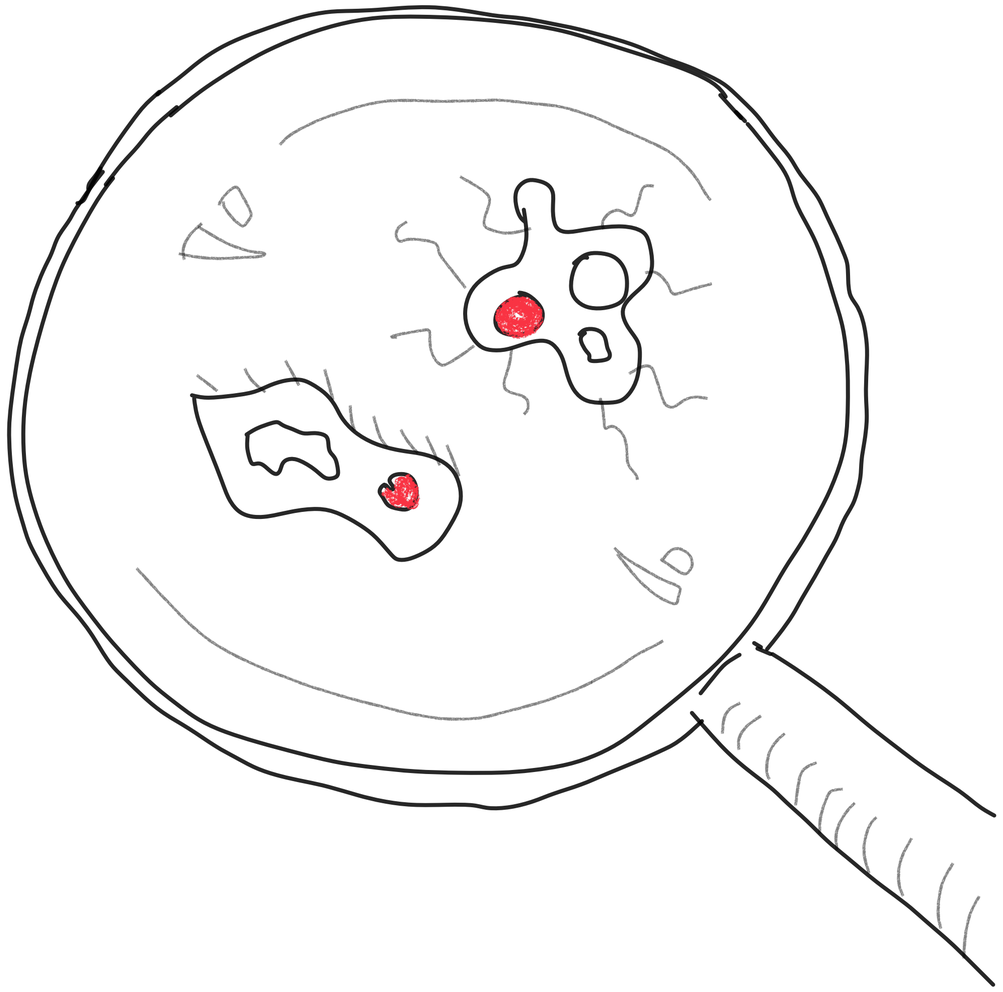
Наследование и полиморфизм

не нужны

Поведение у фактов отсутствует

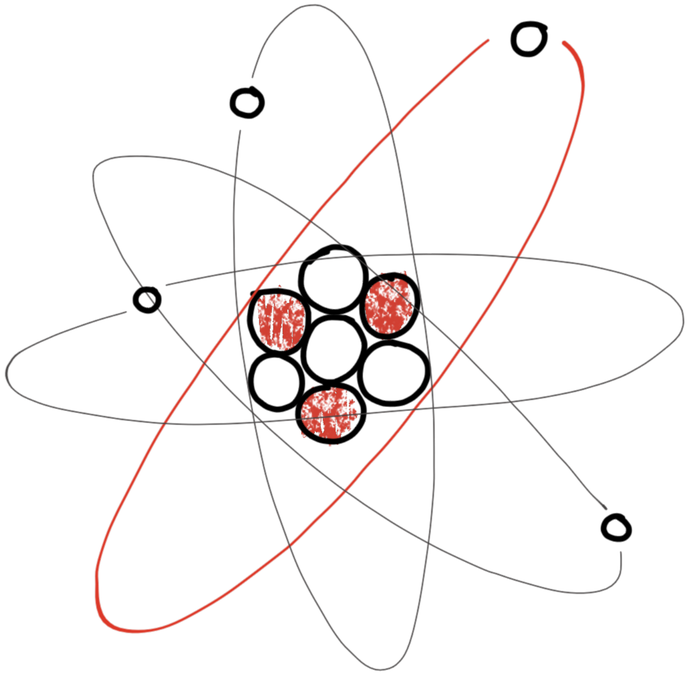


Из чего состоят данные?

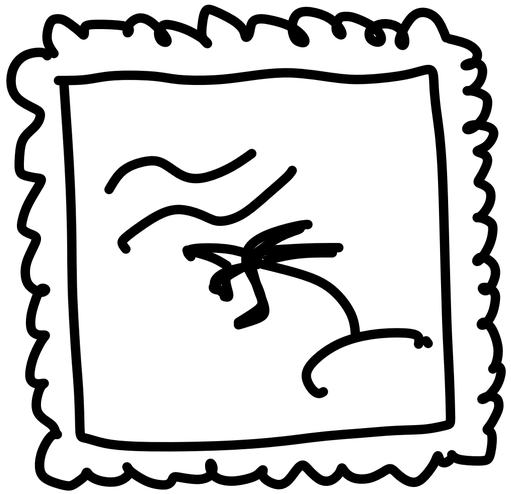


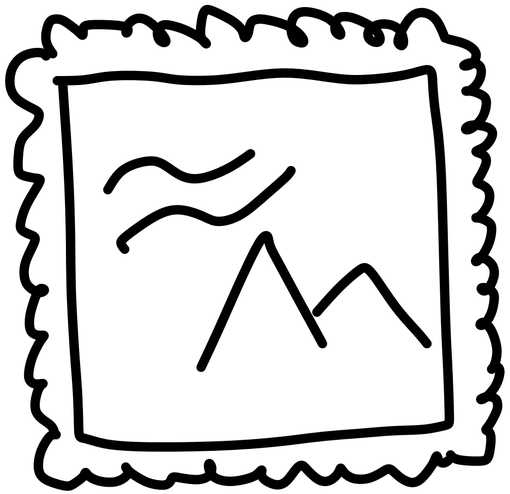
Базовые типы

1) Числа
2) Строки
3) Перечисления



Коллекции





List Map Set

(да, хватит трёх!)

Algebraic Data Types

(composable)

Algebraic Data Types

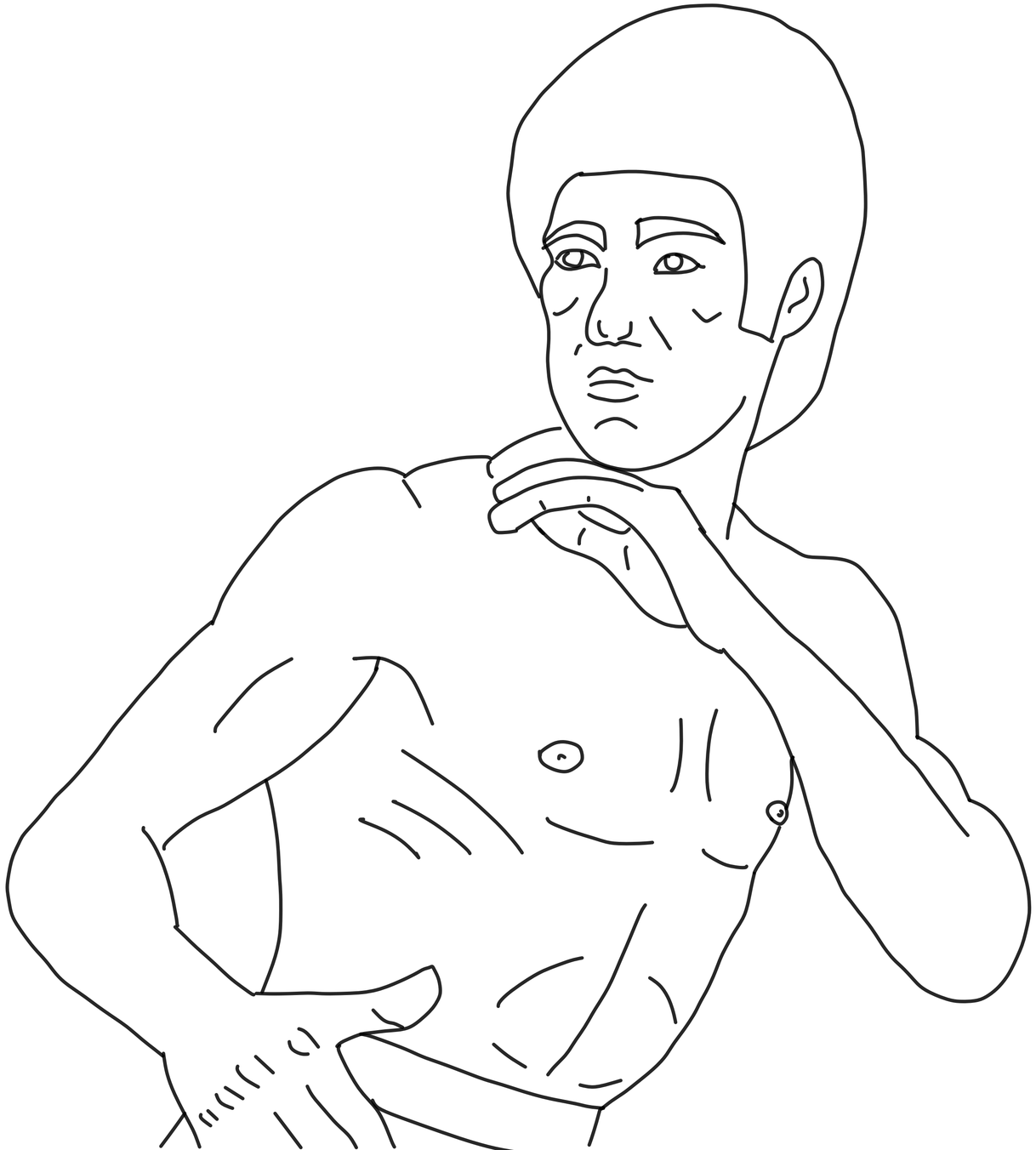
Сложение

(или)

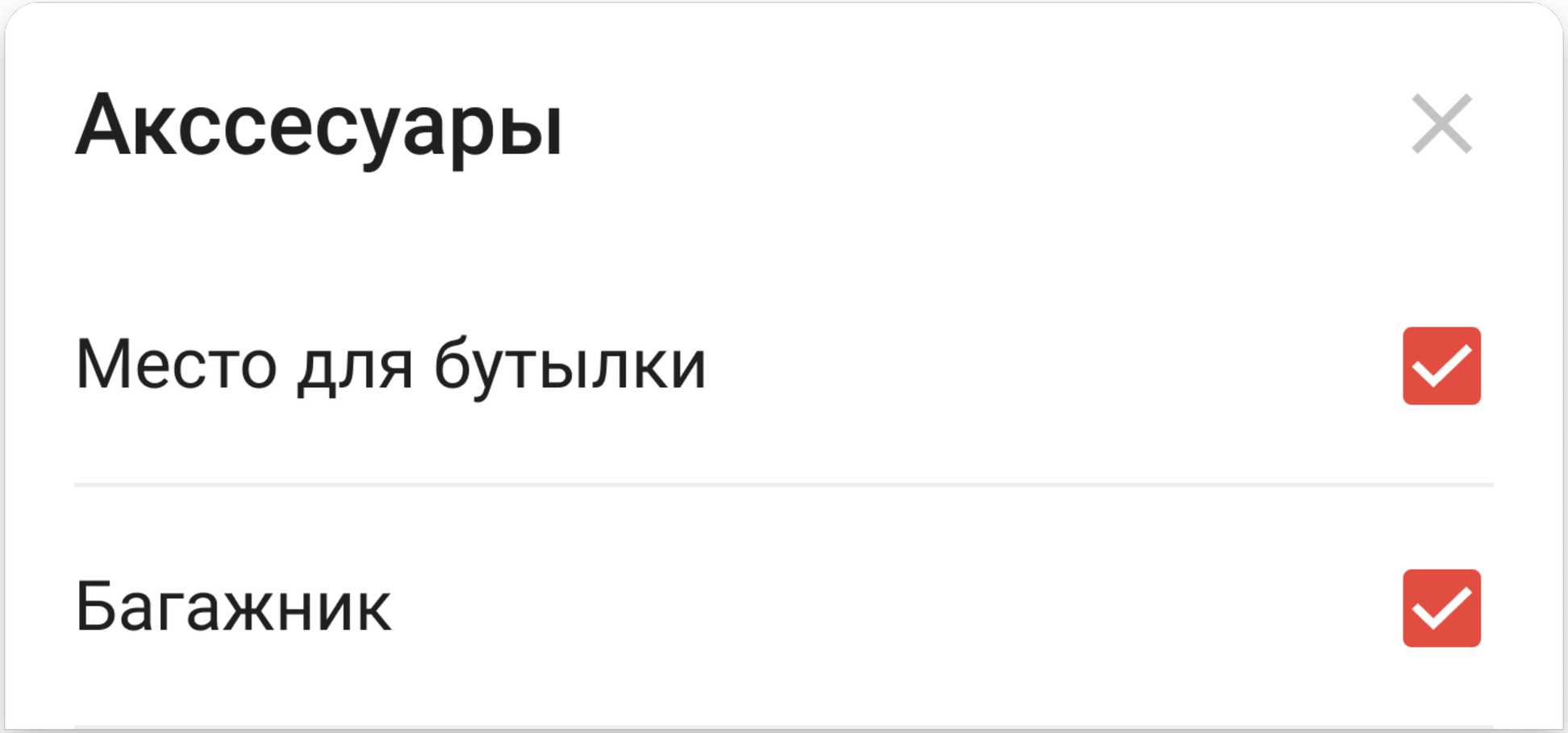
Произведение

(и)

Искусство работы с ADT

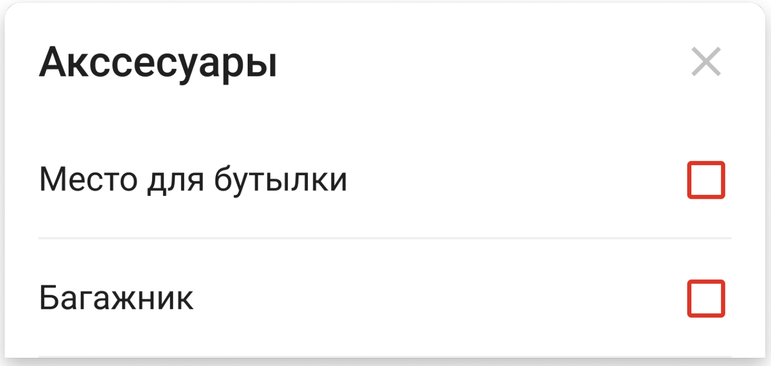


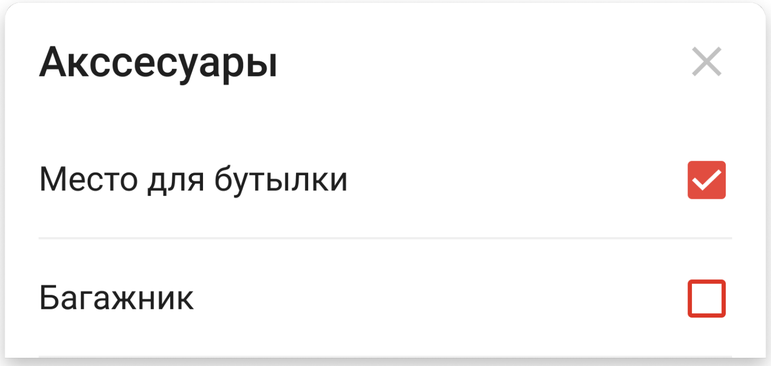
Независимые фильтры

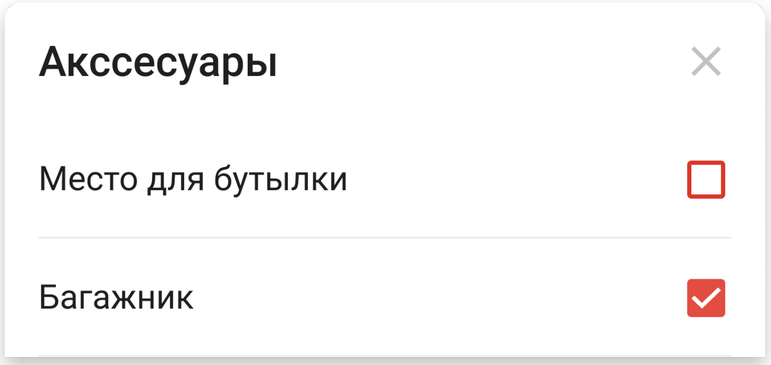


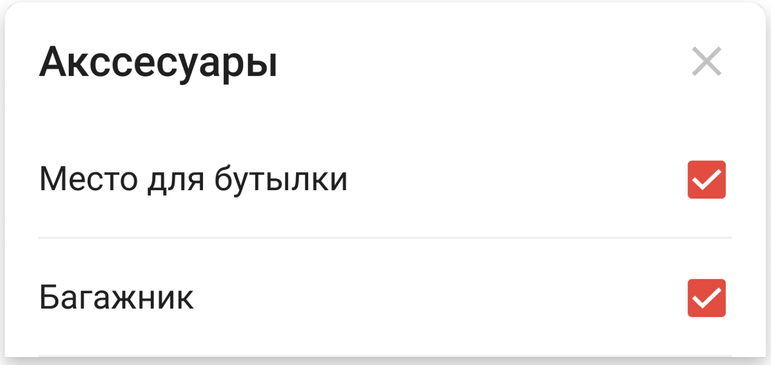
Место для бутылки

Багажник

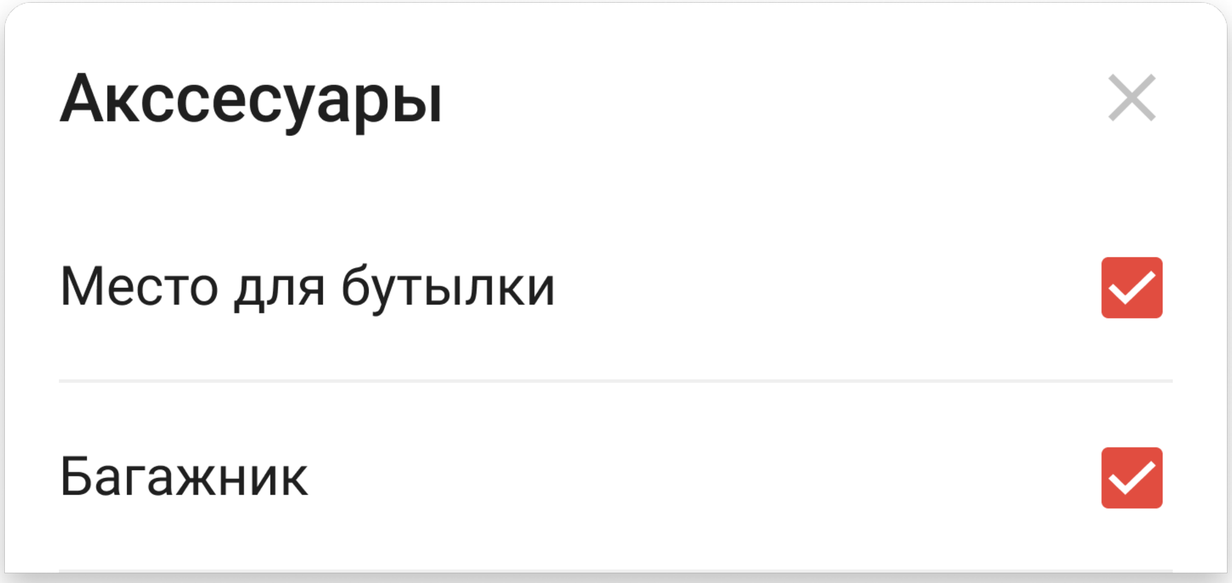








Независимые фильтры – произведение (И)

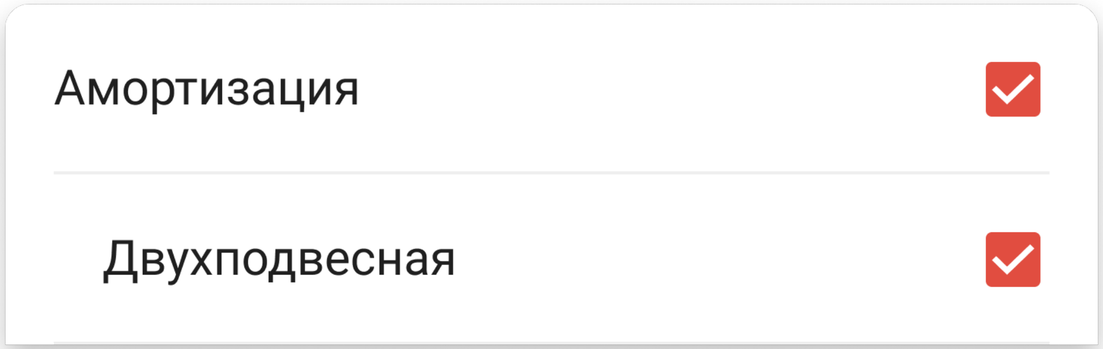


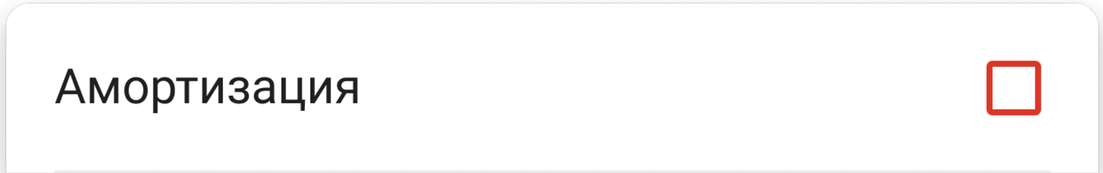
Boolean
×
Boolean

Вот так выглядит в коде

```
data class Filters(
  val bottle: Boolean,
  val electic: Boolean
)
```

Зависимые фильтры





Решение

```
data class Filters(
  val suspension: Boolean,
  val fullSuspension: Boolean
)
```

Амортизация

Двухподвесная

Амортизация ☐

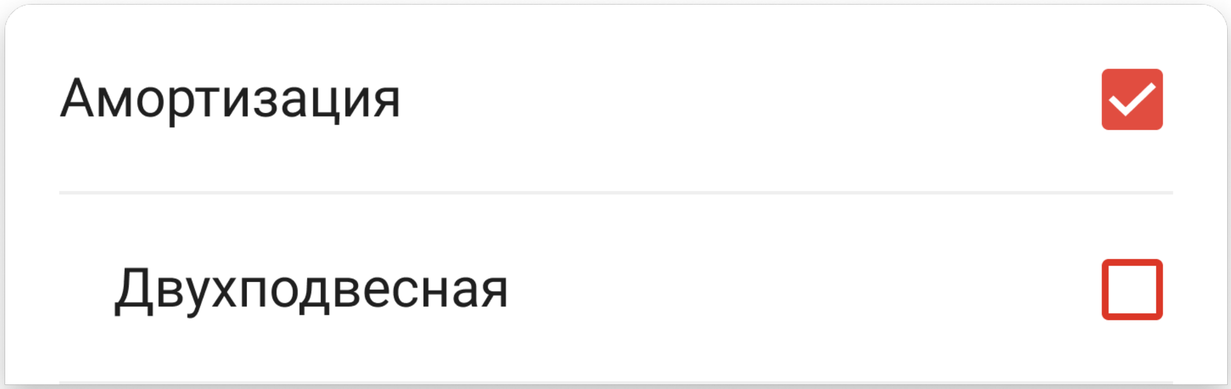
Амортизация ☑

Двухподвесная ☐

Амортизация ☑

Двухподвесная ☑

Зависимые фильтры – сложение (ИЛИ)



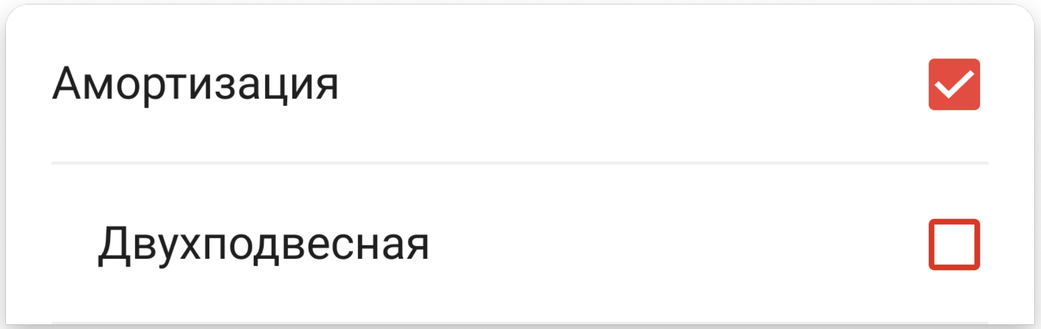
NoSuspension | Boolean

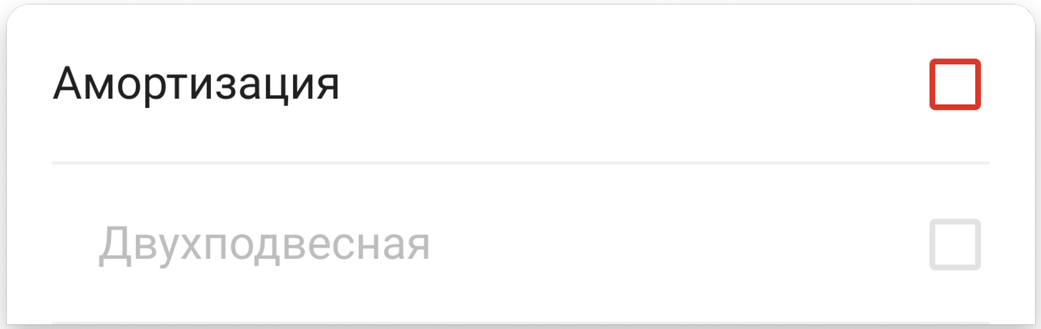
Как это выразить в коде

```
sealed class Filter {
  object NoSuspension
  data class Suspension(val full: Boolean)
}
```

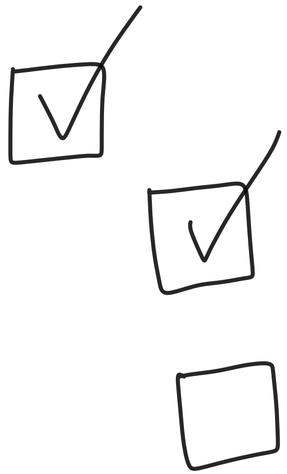
При помощи ИЛИ мы исключаем невалидные состояния

Зависимые фильтры 2

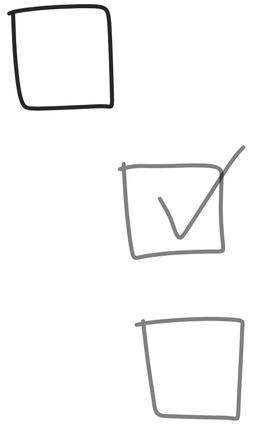




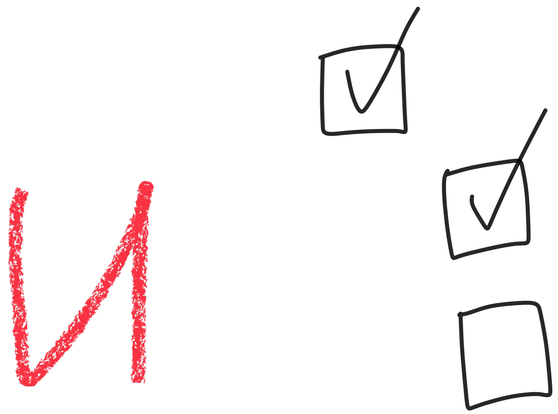
Ловушка ложного ИЛИ



ИЛИ



Ловушка ложного ИЛИ



Как это делают неправильно

```
sealed class Filter {
  data class Suspension(val full: Boolean)
  data class NoSuspension(val full: Boolean)
}
```

Как это делают неправильно

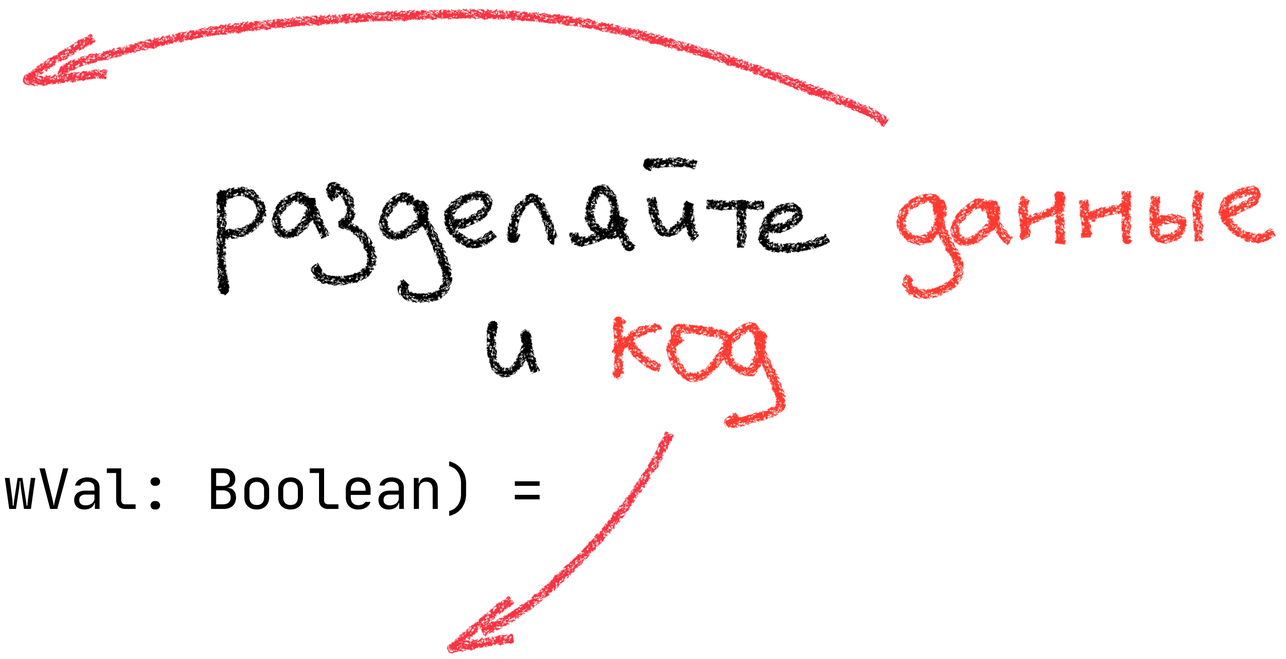
```
sealed class Filter {
  data class Suspension(val full: Boolean)
  data class NoSuspension(val full: Boolean)
}
```

Решение

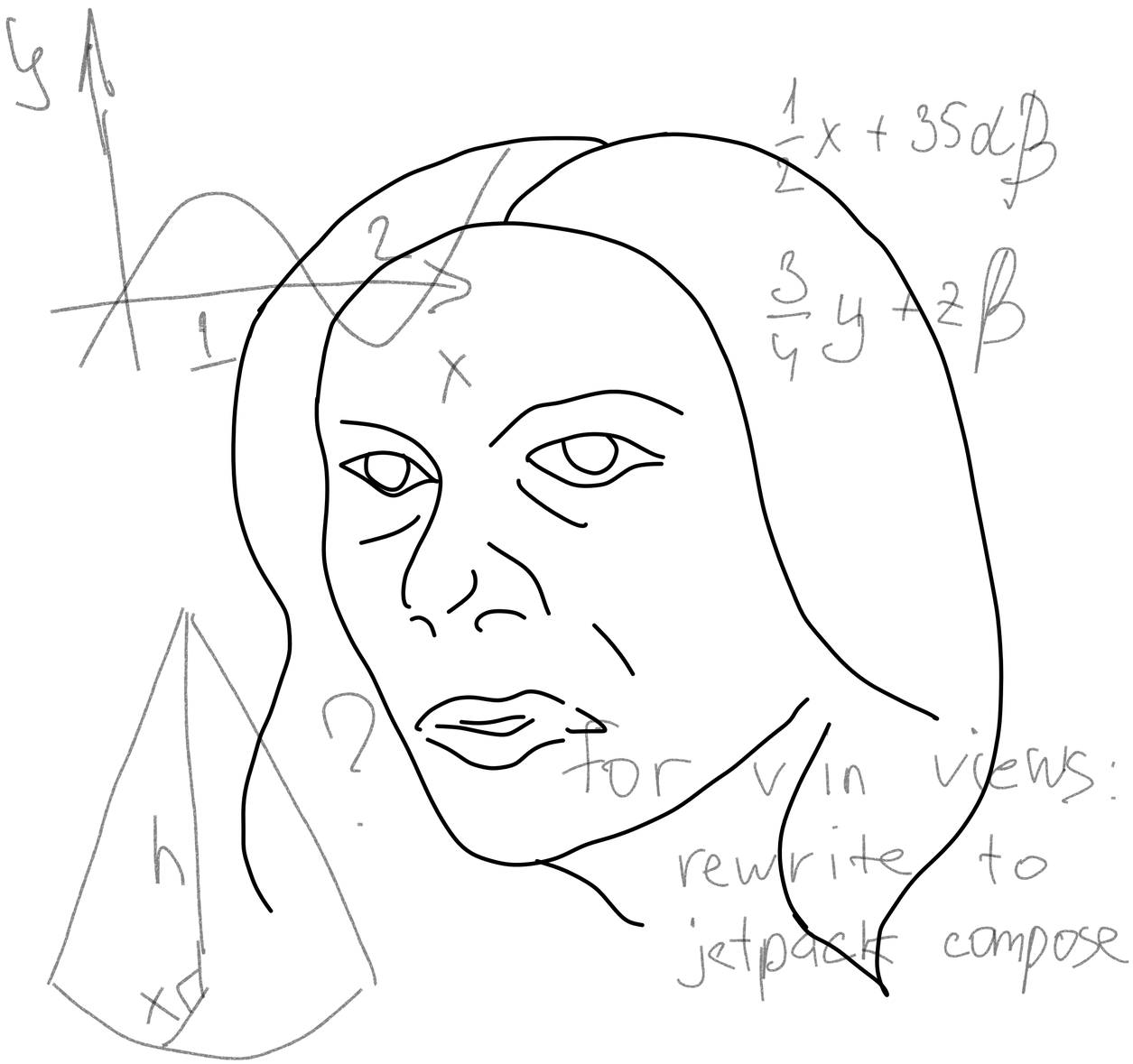
```
data class Filters(
  val hasSuspension: Boolean,
  val fullSuspension: Boolean
)
```

Решение

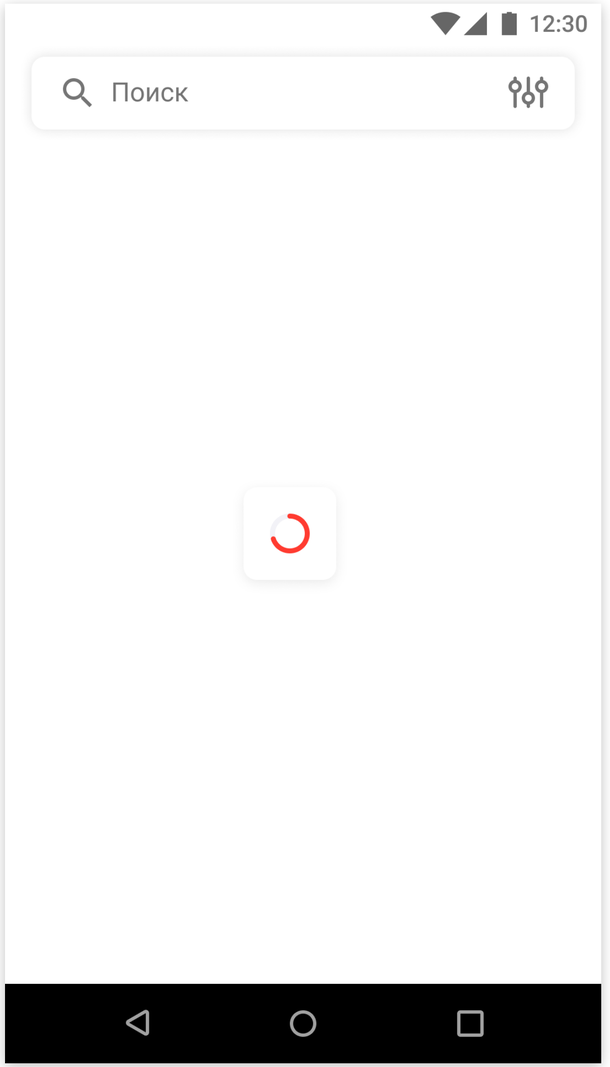
```
data class Filters(
  val hasSuspension: Boolean,
  val fullSuspension: Boolean
)

fun Filters.changeSuspension(newVal: Boolean) =
  if (hasSuspension) {
    copy(fullSuspension = newVal)
  }
```



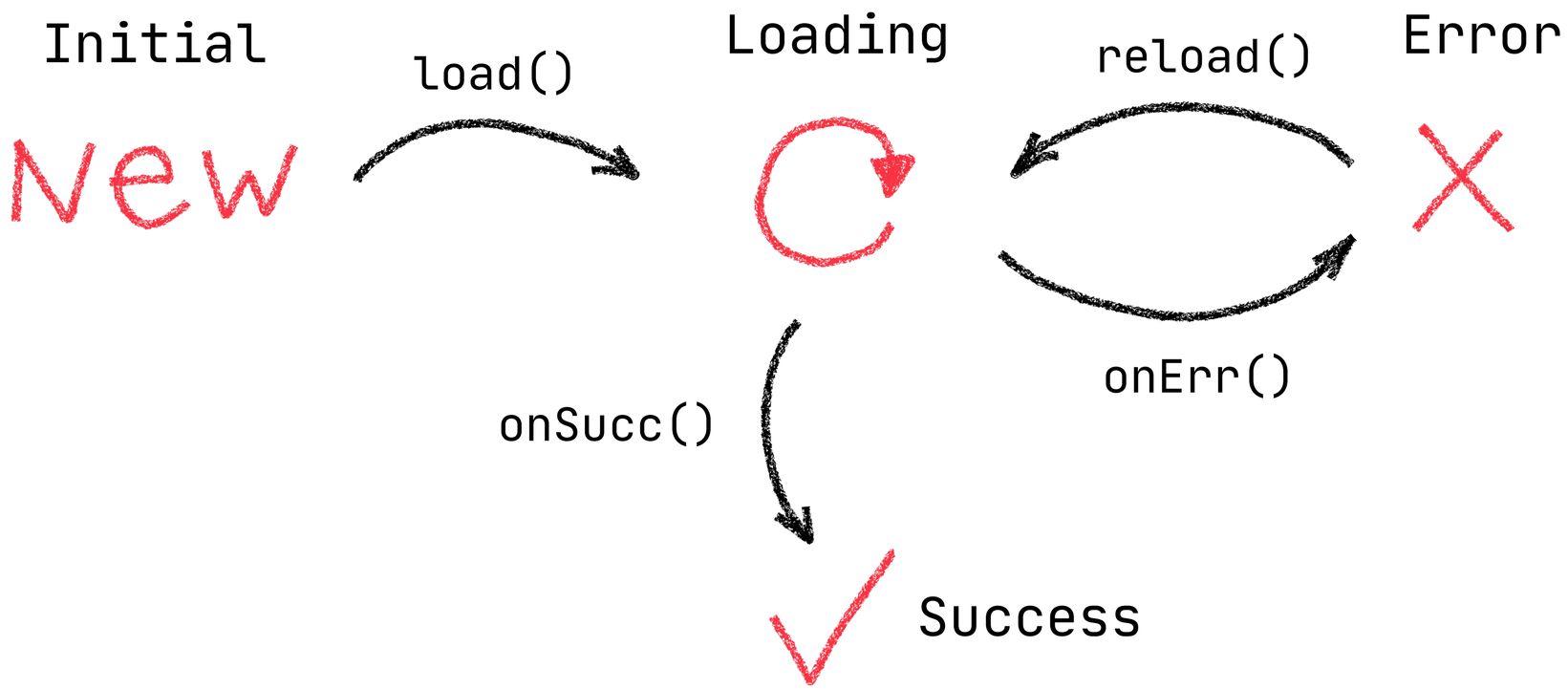
Как считать количество вариантов?



Листинг объявлений



LoadableData (a.k.a. RemoteData)

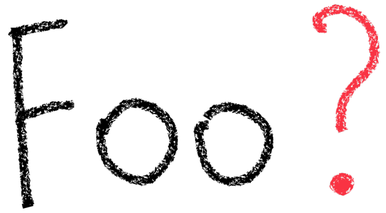


Как это выглядит в коде

```
sealed class LoadableData<T>
object Initial
object Loading
data class Success(value: T)
data class Error(err: Exception)
```

Три метатипа данных

Есть всегда

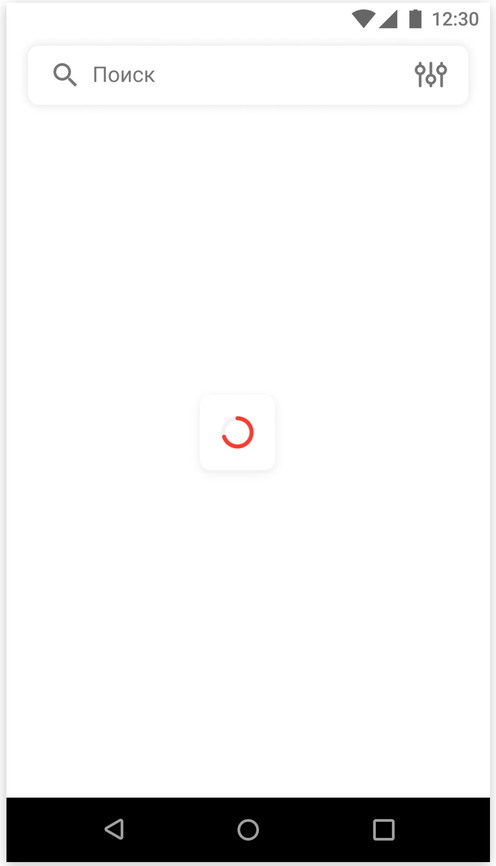


Может отсутствовать

Нужно загрузить

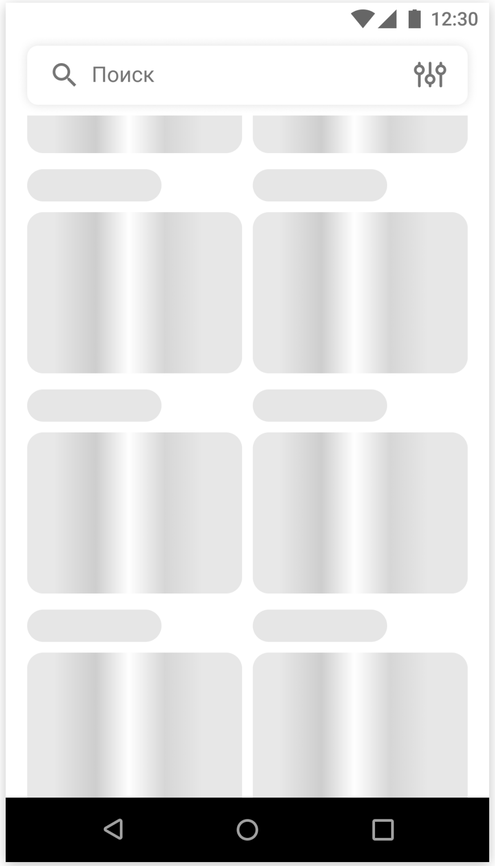
Пример — ленивая загрузка

`LoadableData<List<Offer>>`



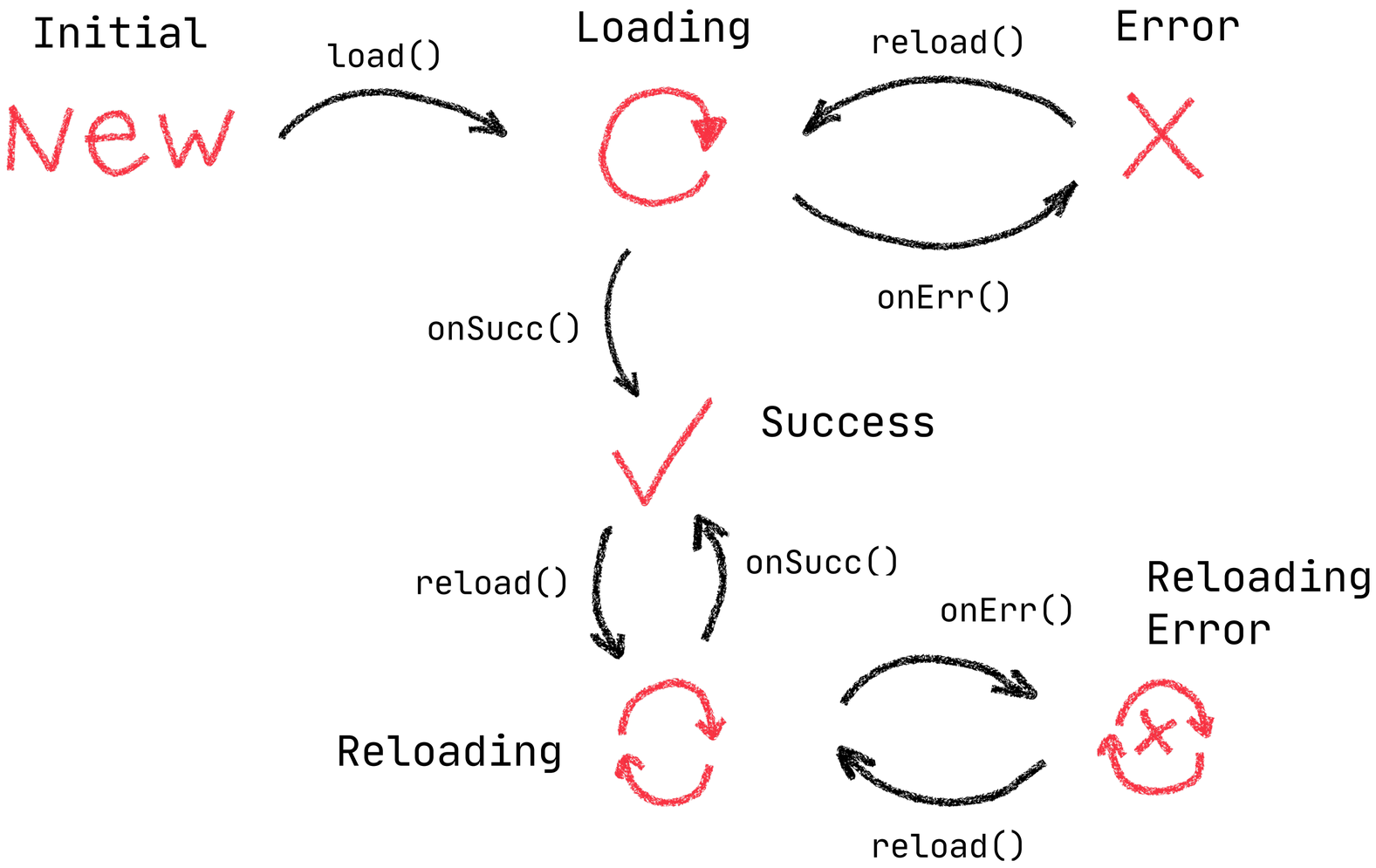
Нужно загрузить весь список сразу

`List<LoadableData<Offer>>`



Можно загружать элементы по отдельности

ReloadableData



Как это выглядит в коде

```
sealed class ReloadableData<T>
object Initial
object Loading
data class Error(err: Exception)
data class Success(value: T)
data class Reloading(value: T)
data class ReloadingError(value: T, err: Exception)
```

Избавляемся от вариантов

```
object Loading
data class Reloading(value: T)

data class Error(err: Exception)
data class ReloadingError(value: T, err: Exception)
```

Избавляемся от вариантов

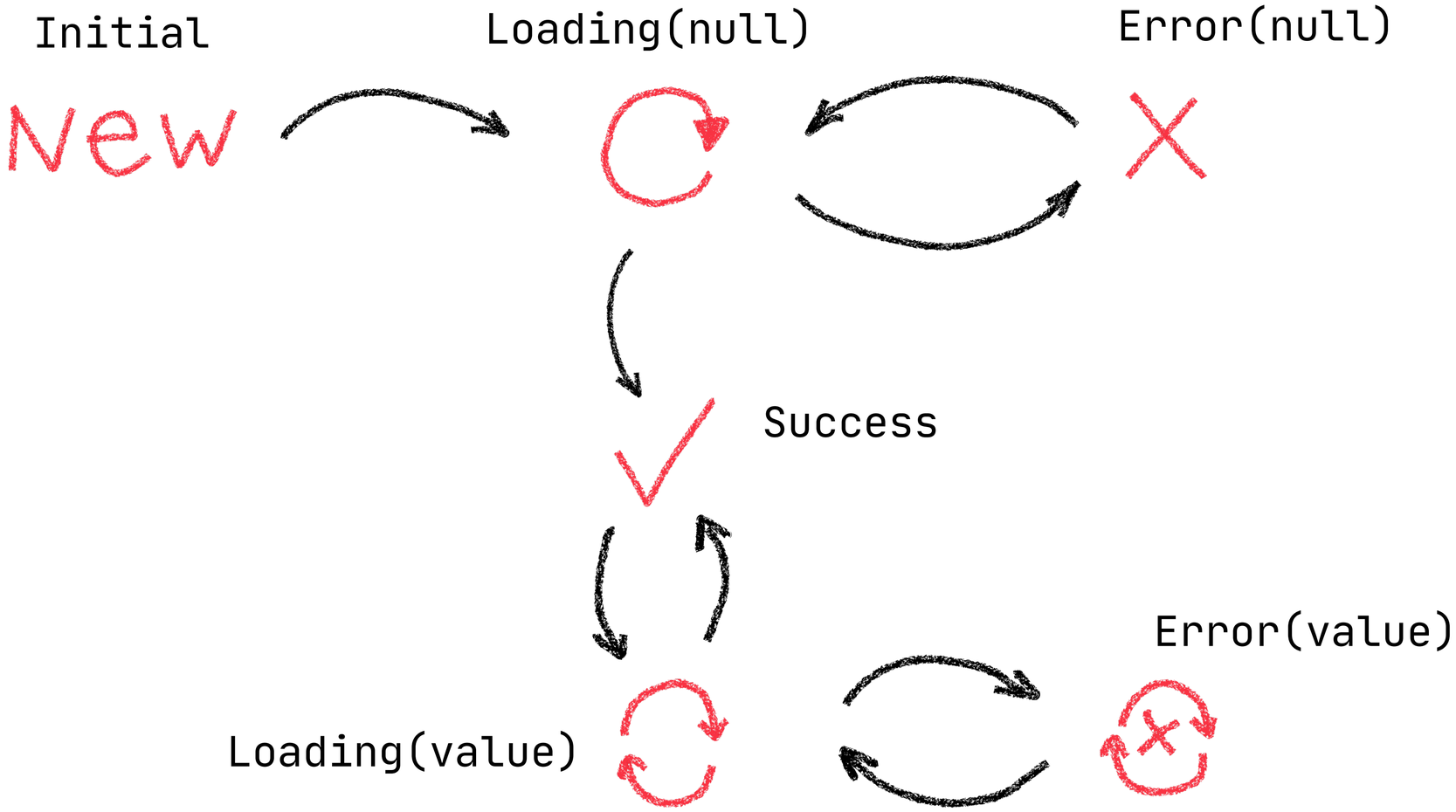
```
object Loading(value: Nothing?)
data class Reloading(value: T)

data class Error(value: Nothing?, err: Exception)
data class ReloadingError(value: T, err: Exception)
```


T | Nothing? = T?

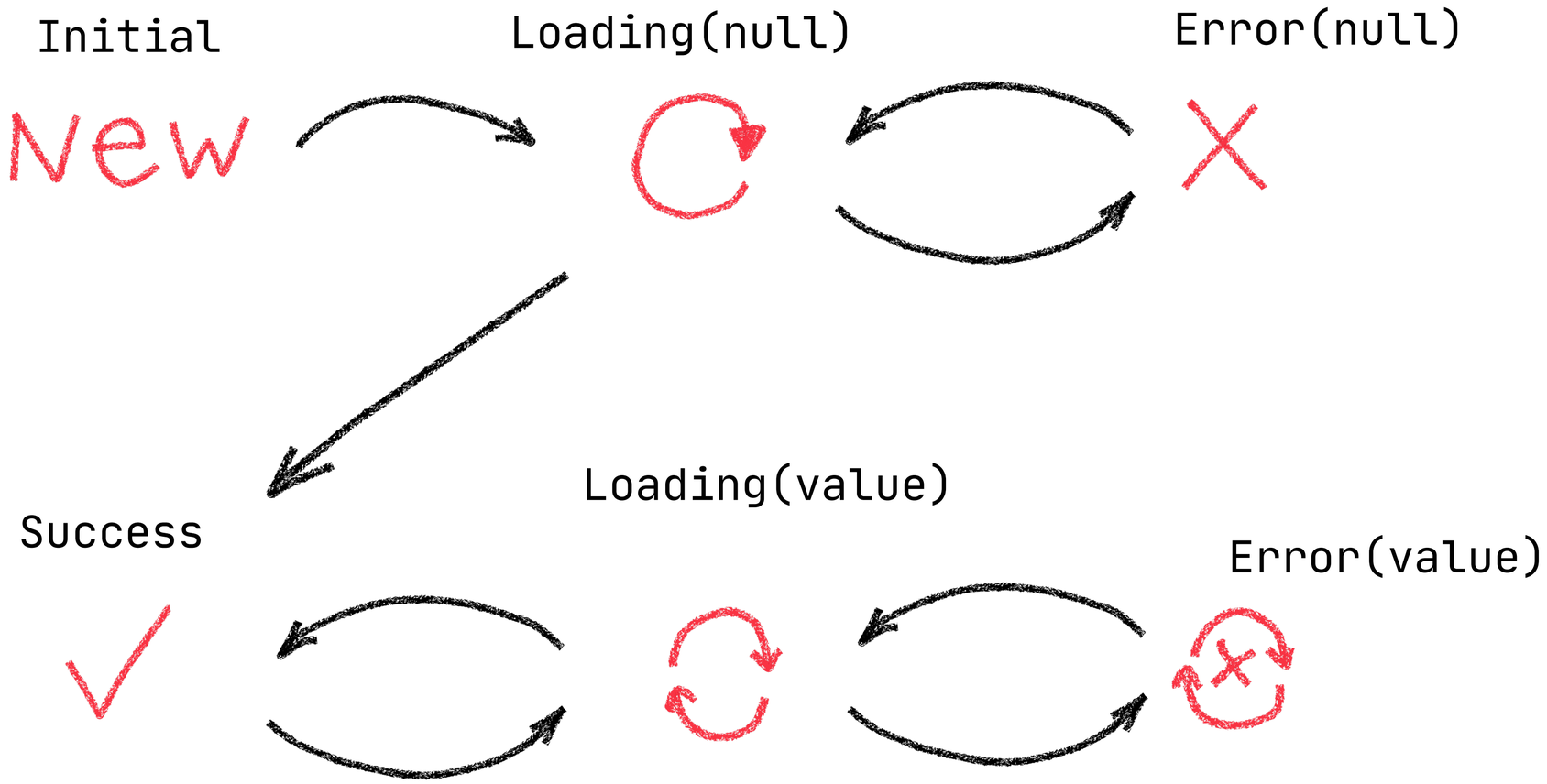
Избавляемся от вариантов

```
data class Loading(value: T?)

data class Error(value: T?, err: Exception)
```

Всё ещё 6 вариантов!



Если присмотреться



Ещё меньше классов

```
object Initial
data class Success(value: T)
```

Еще меньше классов

```
data class Idle(value: T?)
```

Было: 6 классов, 6 вариантов

```
sealed class ReloadableData<T>
object Initial
object Loading
data class Error(err: Exception)
data class Success(value: T)
data class Reloading(value: T)
data class ReloadingError(value: T, err: Exception)
```

Стало: 3 класса, 6 вариантов

```
  sealed class ReloadableData<T>
2 data class Idle(value: T?)
2 data class Loading(value: T?)
2 data class Error(value: T?, err: Exception)
```

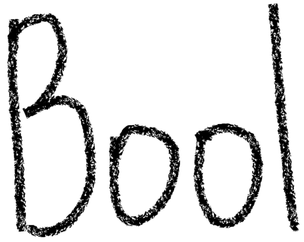
Выносим общую часть

```
sealed class ReloadableData<T>
data class Idle(value: T?)
data class Loading(value: T?)
data class Error(value: T?, err: Exception)
```

Состояний всё ещё шесть

```
data class ReloadableData<T>(value: T?, status: Status)
sealed class Status
object Idle
object Loading
data class Error(err: Exception)
```

Как считать количество вариантов?



2 варианта:

〉 true

〉 false

2+1 варианта:

〉 true

〉 false

〉 null

2*3 варианта:

〉 true true

〉 true false

〉 true null

〉 false true

〉 false false

〉 false null

2+3 варианта

〉 true

〉 false

ИЛИ

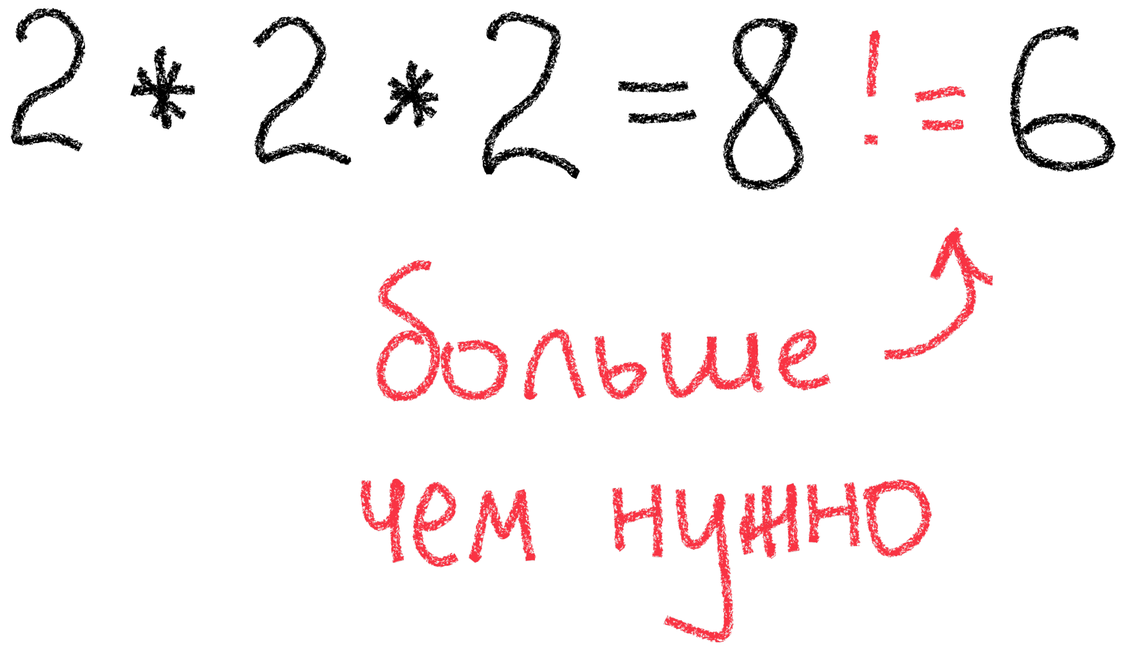
〉 true

〉 false

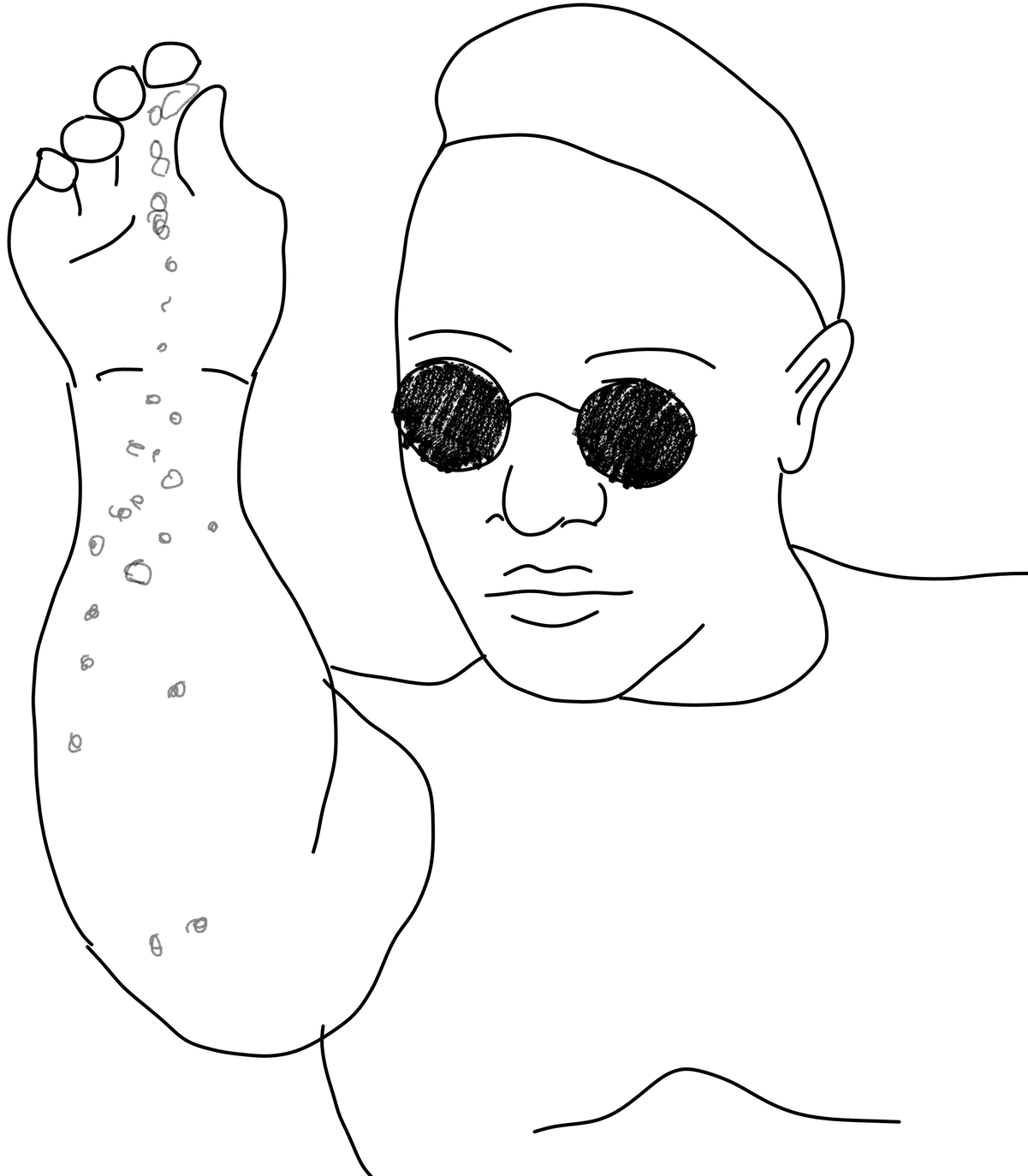
〉 null

Неправильно

```
data class ReloadableData<T>(
  value: T?,
  err: Exception?,
  loadable: Boolean
)
```



Щепотка
Clean Code

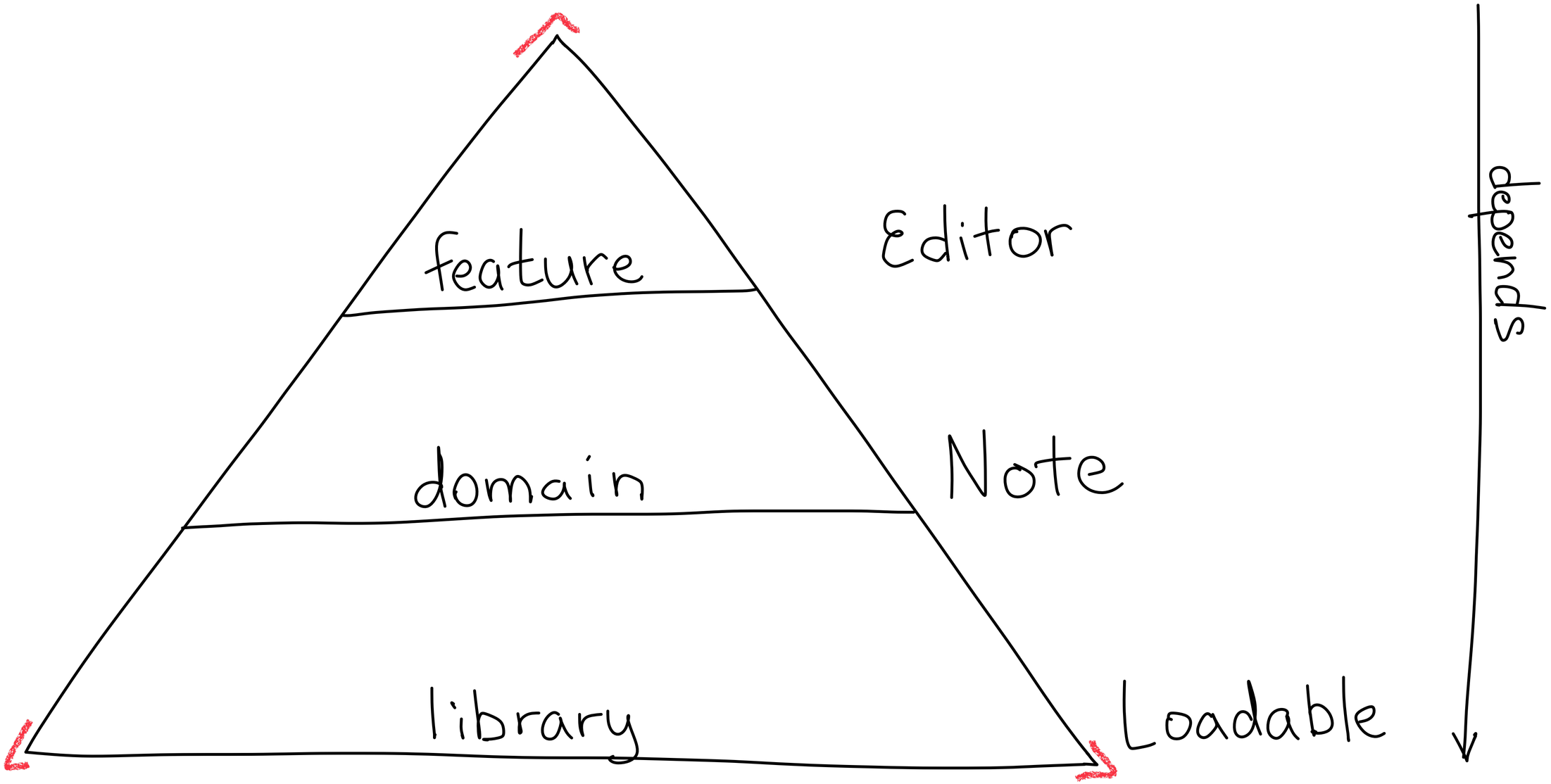


Просмотренность объявлений

Решение

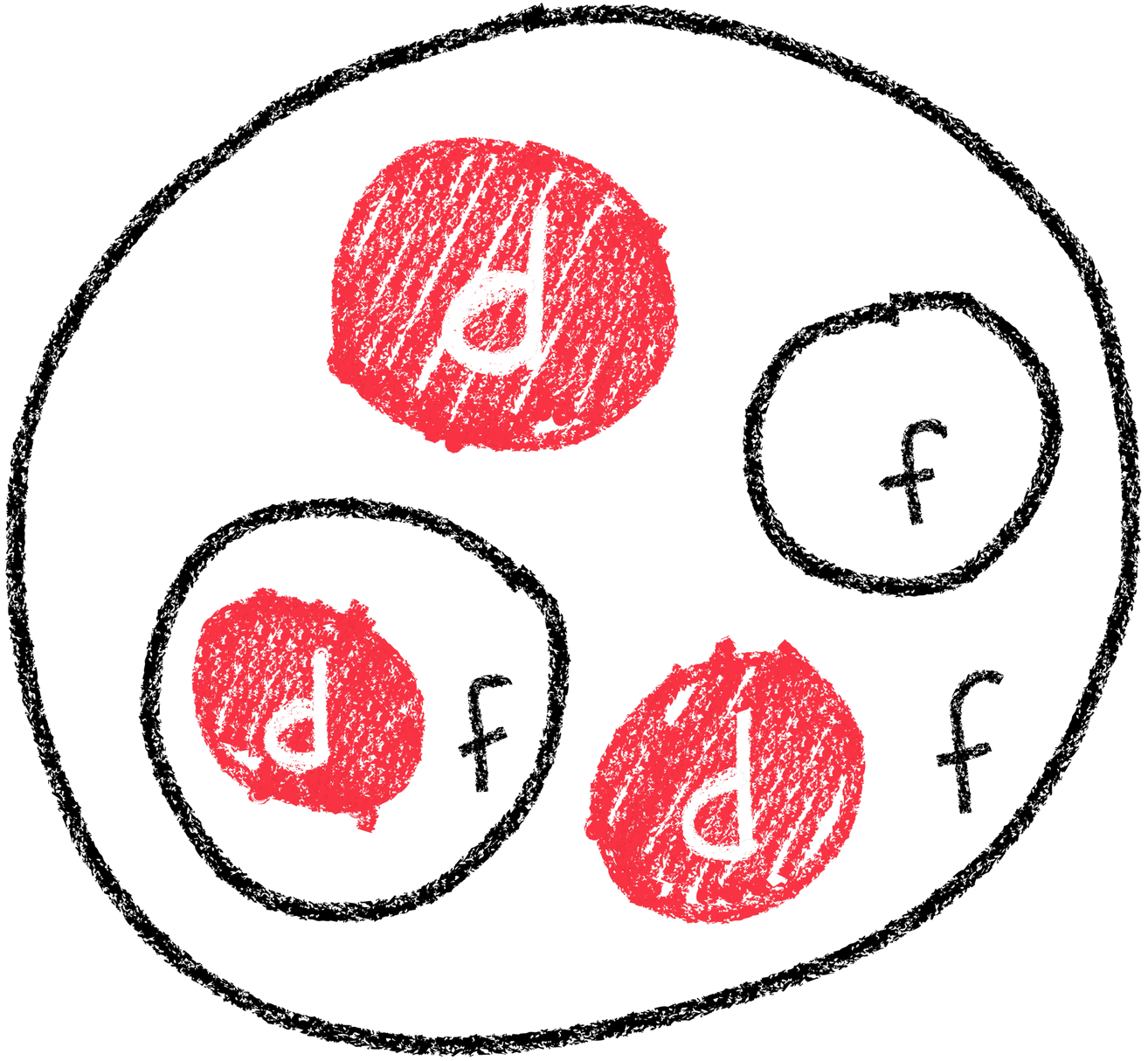
```
data class Offer(
   ...
   val viewed: Boolean
)
```

Dependency Rule



Решение: композиция

```
data class OfferItem(
  val offer: Offer
  val viewed: Boolean
)
```



Стейт — твоя база данных

Просмотренность объявлений 2

```
data class SearchScreen(
  val offers: List<Offer>,
  val viewedOffers: List<Offer>
)
```

Трюки DB Modelling

Дедупликация

```
data class SearchScreen(
  val offers: List<Offer>,
  val viewed: Map<ID, Boolean>
)
```

Дедупликация

```
Map<ID, Boolean>  ID → { true, false, null }

Set<ID>           ID → { true, false }
```

Дедупликация

```
data class SearchScreen(
  val offers: List<Offer>,
  val viewed: Set<ID>
)
```

Вьюшки

```
data class SearchScreen(
  val offers: List<Offer>,
  val viewed: Set<ID>
)

val SearchScreen.viewedOffers =
  offers.filter { it.id in viewed }
```

Дополнительные услуги

Выделить цветом	200 ₽	☑
VIP	1 000 ₽	☐

Итого: 1 200 ₽

Купить

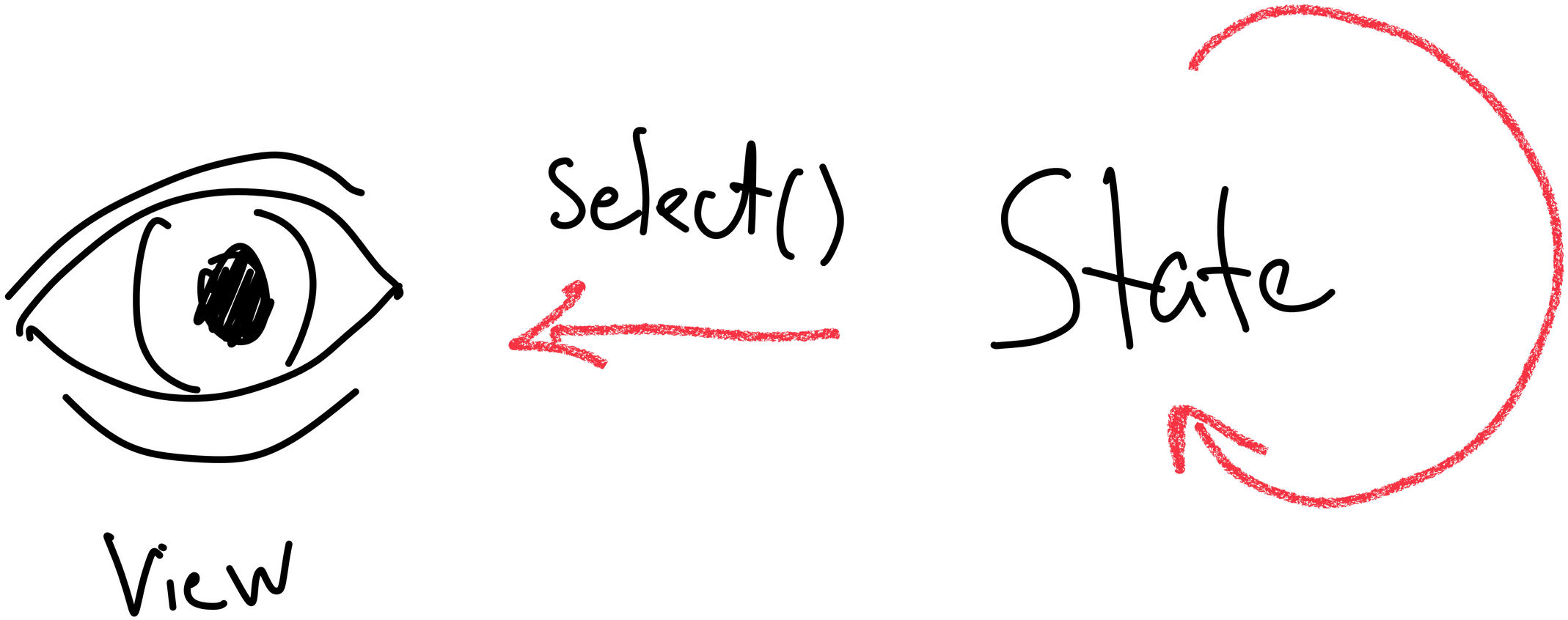
Неправильное решение

```
data class Services(
  val services: List<Service>,
  val selected: Set<String>,
  val total: Long
)
```

Используйте селекторы

```
fun selectTotal() = services
   .filter { it.id in selected }
   .sumBy { it.price }
```

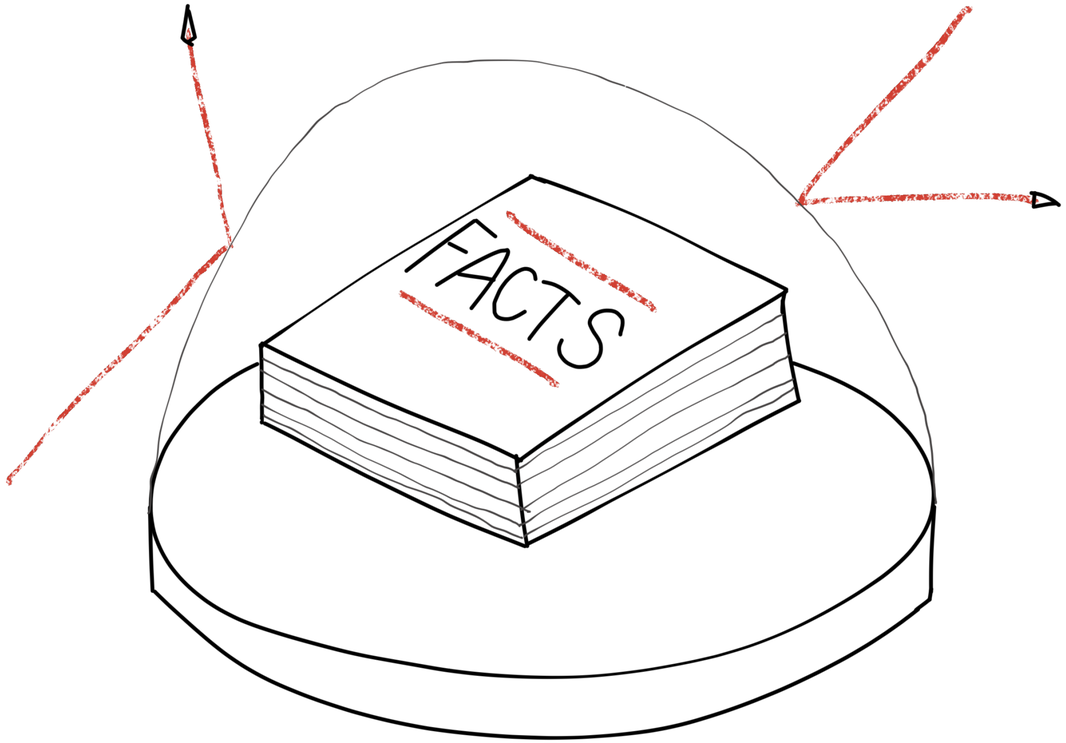
Почему UDF не просто очередной MV*



Перфоманс?

Кешируйте селекторы

Данные неизменяемы – обычного LruCache по ссылке хватит всем



Храните редактируемые поля отдельно

```
data class Note(
  val offerId: OfferId,
  val text: String
)
```

Храните редактируемые поля отдельно

```
data class NoteScreen(
  val note: Note,
  val noteText: String
)
```

Бонус: простейший откат назад

Стройте индексы

```
/offers → List<Offer>
/notes → List<Note>

Note { offerId: OfferId, ... }

offersToNotes: Map<OfferId, NoteId>
```

Как моделировать стейт?

1. Стейт – данные, а не объекты
2. Для данных нужны другие инструменты – ADT и коллекции
3. С увеличением размера стейт должен превращаться в БД
4. Как только стейт стал БД — применяйте все практики оттуда

auto.ru ← приходи к нам :)

Стейт – упрощённая БД на ADT и коллекциях

Личные контакты:

@themishkun

@themishkun

Мои проекты:

@podvedro – канал с мемами

@izpodshtorki – блог о программировании