Scala для Android. Миф или реальность

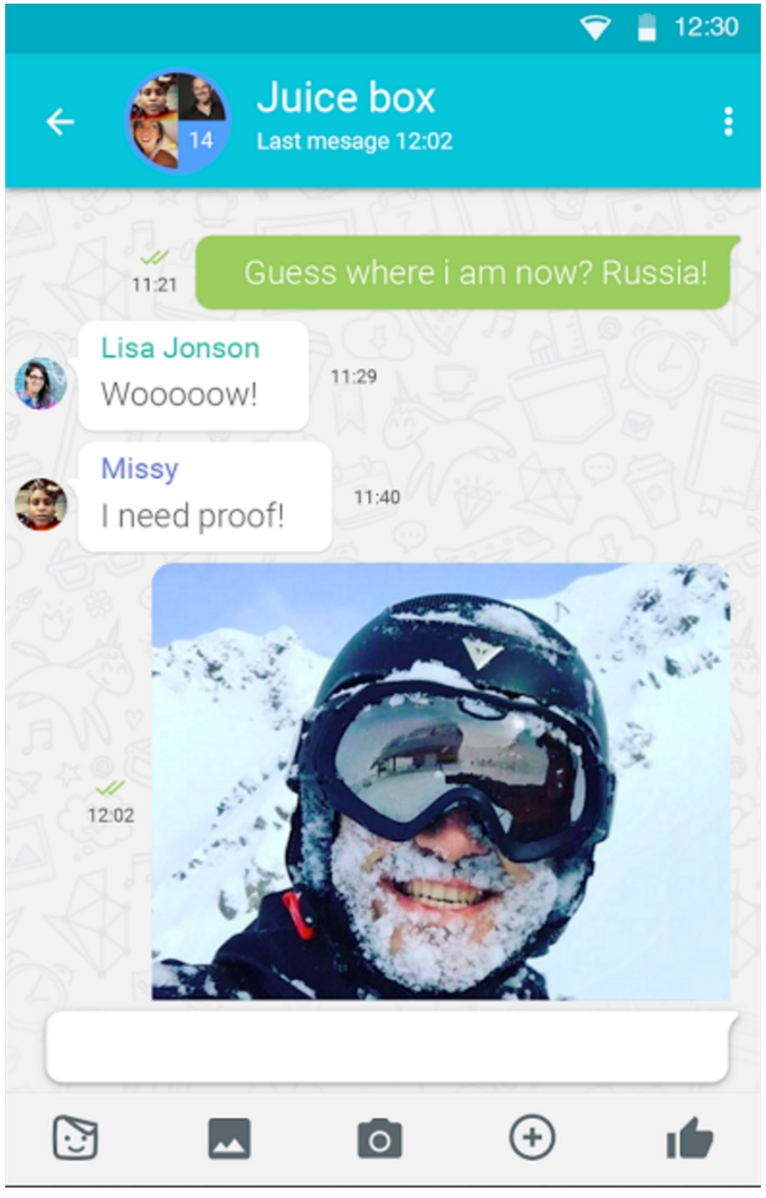
Матвей Мальков

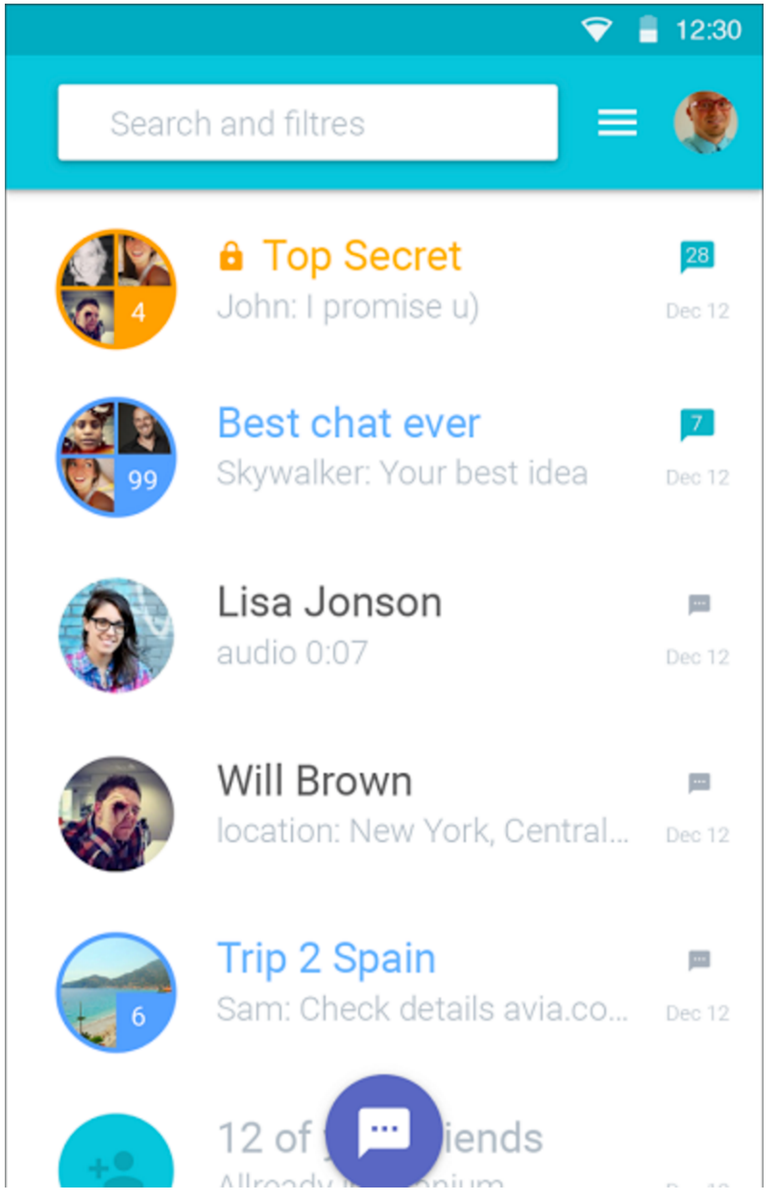
Обо мне

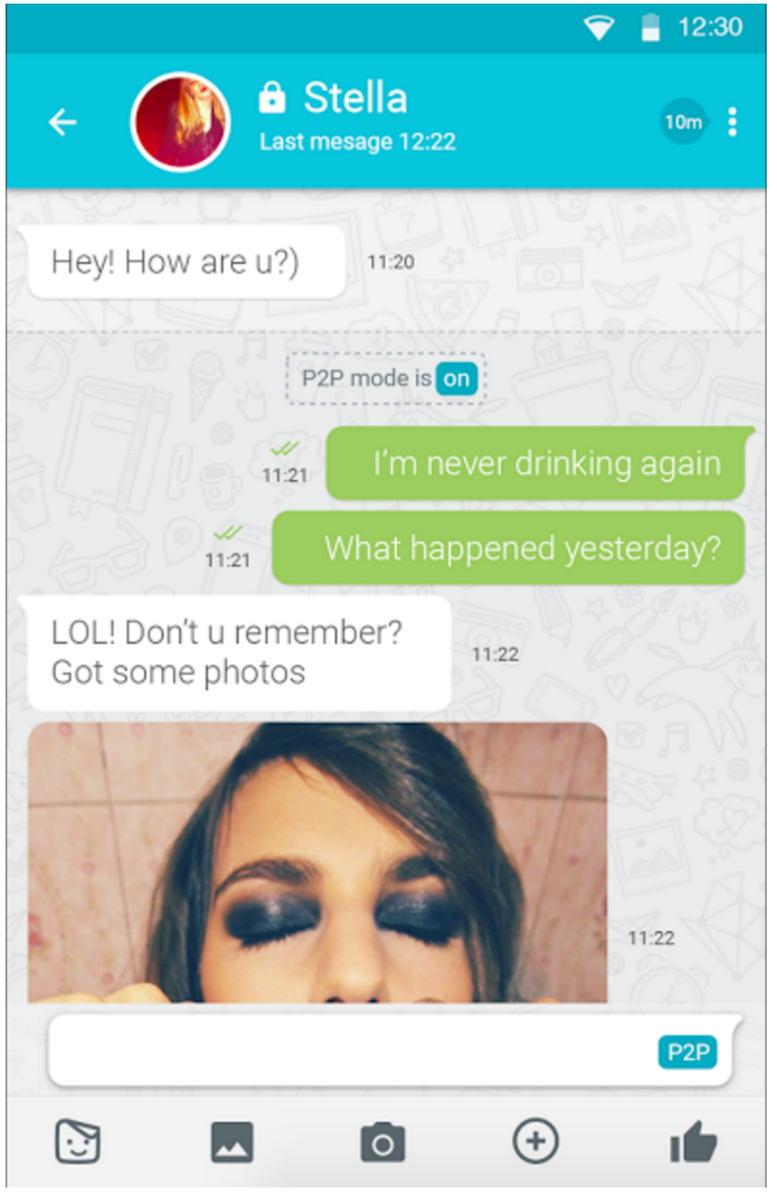
Обо мне

Обо мне

Обо мне


12:30
Juice box
14
Last mesage 12:02
11:21
Guess where i am now? Russia!
Lisa Jonson
Wooooow!
11:29
Missy
I need proof!
11:40
12:02


12:30
Search and filtres
Top Secret
John: I promise u)
28
Dec 12
4
Best chat ever
Skywalker: Your best idea
7
Dec 12
99
Lisa Jonson
audio 0:07
Dec 12
Will Brown
location: New York, Central...
Dec 12
Trip 2 Spain
Sam: Check details avia.co...
Dec 12
6


12:30
Stella
Last mesage 12:22
10m
Hey! How are u?)
11:20
P2P mode is on
11:21
I'm never drinking again
11:21
What happened yesterday?
LOL! Don't u remember?
Got some photos
11:22
11:22
P2P

Titanium messenger

Titanium messenger

- end-to-end шифрование

Titanium messenger

- end-to-end шифрование
- качественный продукт, не MVP

Titanium messenger

- end-to-end шифрование
- качественный продукт, не MVP
- повышенная безопасность

Titanium messenger

- end-to-end шифрование
- качественный продукт, не MVP
- повышенная безопасность
- p2p чаты и звонки

Titanium messenger

- end-to-end шифрование
- качественный продукт, не MVP
- повышенная безопасность
- p2p чаты и звонки
- в релизе

Положение дел

Положение дел

Положение дел

- java 7 версии

Положение дел

- java 7 версии
  - лямбды есть

Положение дел

- java 7 версии
  - лямбды есть
  - ничего больше нет

Положение дел

- java 7 версии
  - лямбды есть
  - ничего больше нет
- новые языки

Положение дел

- java 7 версии
  - лямбды есть
  - ничего больше нет
- новые языки
    - лаконичные

Положение дел

- java 7 версии
  - лямбды есть
  - ничего больше нет
- новые языки
    - лаконичные
    - с хорошим API

Положение дел

- java 7 версии
  - лямбды есть
  - ничего больше нет
- новые языки
    - лаконичные
    - с хорошим API
    - JVM-based

Проблемы разработки

Проблемы разработки

- много корнеркейсов

Проблемы разработки

- много корнеркейсов
- уродливость кода

Проблемы разработки

- много корнеркейсов
- уродливость кода
- сложные конструкции

Проблемы разработки

- много корнеркейсов
- уродливость кода
- сложные конструкции
- ***Fragment на 3500 строк кода

Проблемы разработки

- много корнеркейсов
- уродливость кода
- сложные конструкции
- ***Fragment на 3500 строк кода
- многопоточность

JVM-семейство

JVM-семейство

- Kotlin

JVM-семейство

- Kotlin
- Scala

JVM-семейство

- Kotlin
- Scala
- Clojure

JVM-семейство

- Kotlin
- Scala
- Clojure
- Groovy

Scala

Scala

Scala

- ООП + ФП

Scala

- ООП + ФП
- супер синтаксис

Scala

- ООП + ФП
- супер синтаксис
- REPL

Scala

- ООП + ФП
- супер синтаксис
- REPL
- полностью java-совместима

Scala

- ООП + ФП
- супер синтаксис
- REPL
- полностью java-совместима
  - почти

Scala

- ООП + ФП
- супер синтаксис
- REPL
- полностью java-совместима
  - почти
- работа с коллекциями

Scala

- ООП + ФП
- супер синтаксис
- REPL
- полностью java-совместима
  - почти
- работа с коллекциями
- примеси, лямбды, фьючи и т.д.

```scala
case class User(name: String, age: Int)
```

```scala
case class User(name: String, age: Int)

val u = User("Matvey", 16)
```

```scala
case class User(name: String, age: Int)

val u = User("Matvey", 16)

println(u.name)
```

```scala
case class User(name: String, age: Int)

val u = User("Matvey", 16)

println(u.name) //User(Matvey,16)
```

```
def doSmth(u: User): Response = {
  //code here
}
```

```
def doSmth(u: User): Response = {
  //code here
}

var u = User("Matvey", 16)

u = User("Boris", 21)
```

Зачем разрабатывать на Scala под Android?

Зачем мне это?

Зачем мне это?

- безопасность

Зачем мне это?

- безопасность
- разделение и переиспользование

Зачем мне это?

- безопасность
- разделение и переиспользование
- хорошая архитектура

Зачем мне это?

- безопасность
- разделение и переиспользование
- хорошая архитектура
- легкое построение DSL и работа с UI

1. Безопасность

Option

Option

- простой АТД

Option

- простой АТД
- 2 варианта

Option

- простой АТД
- 2 варианта
  - Some(smth)

Option

- простой АТД
- 2 варианта
  - Some(smth)
  - None

Option

- простой АТД
- 2 варианта
  - Some(smth)
  - None
- никаких null

Option

- простой АТД
- 2 варианта
  - Some(smth)
  - None
- никаких null
- куча методов для работы

```
case class User(name: String, age: Option[Int])
```

```
case class User(name: String, age: Option[Int])

val m = User("Matvey", Some(16))
val b = User("Boris", None)
```

```scala
case class User(name: String, age: Option[Int])

val m = User("Matvey", Some(16))
val b = User("Boris", None)

m.age.foreach(println)
```

```scala
case class User(name: String, age: Option[Int])

val m = User("Matvey", Some(16))
val b = User("Boris", None)

m.age.foreach(println)
val borisAge = b.age.getOrElse(27)
```

```
case class User(name: String, age: Option[Int])

val m = User("Matvey", Some(16))
val b = User("Boris", None)

m.age.foreach(println)
val borisAge = b.age.getOrElse(27)

m.age.filter(_ > 18).map(makeDrink).foreach(_.drink)
```

```
val s = Option(getArguments.getString(key))
```

```
val s = Option(getArguments.getString(key))

implicit class AnyToOption[T](x: T) {
  def option: Option[T] = Option(x)
}
```

```
val s = Option(getArguments.getString(key))

implicit class AnyToOption[T](x: T) {
  def option: Option[T] = Option(x)
}


val s = getArguments.getString(key).option
```

```
val s = Option(getArguments.getString(key))

implicit class AnyToOption[T](x: T) {
  def option: Option[T] = Option(x)
}
```

```
val s = getArguments.getString(key).option
s.foreach { arg ⇒
  initUiWithArgument(arg)
}
```

```
val m = User("Matvey", 16.option)
val b = User("Boris", None)
val y = User("Yana", 19.option)

val users = Seq(m, b, y)
```

```
val m = User("Matvey", 16.option)
val b = User("Boris", None)
val y = User("Yana", 19.option)

val users = Seq(m, b, y)
```

```
val m = User("Matvey", 16.option)
val b = User("Boris", None)
val y = User("Yana", 19.option)

val users = Seq(m, b, y)

println(users(0).name)
```

```
val m = User("Matvey", 16.option)
val b = User("Boris", None)
val y = User("Yana", 19.option)

val users = Seq(m, b, y)

println(users(0).name) //плохо
```

```
val m = User("Matvey", 16.option)
val b = User("Boris", None)
val y = User("Yana", 19.option)

val users = Seq(m, b, y)

println(users(0).name) //плохо
users.headOption.foreach(println) //хорошо
```

```
val m = User("Matvey", 16.option)
val b = User("Boris", None)
val y = User("Yana", 19.option)

val users = Seq(m, b, y)

println(users(0).name) //плохо
users.headOption.foreach(println) //хорошо

val f = users
  .filter(_.age.isDefined)
  .find(_.name == "Matvey")
```

Паттерн - матчинг

Паттерн - матчинг

- похоже на switch-case

Паттерн - матчинг

- похоже на switch-case
  - мощнее

Паттерн - матчинг

- похоже на switch-case
  - мощнее
- перебор возможных данных

Паттерн - матчинг

- похоже на switch-case
  - мощнее
- перебор возможных данных
  - всех!

Паттерн - матчинг

- похоже на switch-case
  - мощнее
- перебор возможных данных
  - всех!
- наглядно

```
def printAge(user: User) = user.age match {
  case Some(a) ⇒ println(s"age is $a")
  case None ⇒ println("no age for user")
}
```

```scala
def printAge(user: User) = user.age match {
  case Some(a) ⇒ println(s"age is $a")
  case None ⇒ println("no age for user")
}
```

```
def printAge(user: User) = user.age match {
  case Some(a) ⇒ println(s"age is $a")
  case None ⇒ println("no age for user")
}
```

```
def printAge(user: User) = user.age match {
  case Some(a) ⇒ println(s"age is $a")
  case None ⇒ println("no age for user")
}
```

```
def printAge(user: User) = user.age match {
  case Some(14) ⇒ print("passport required")
  case Some(40) ⇒ print("change your passport")
}
```

```
def printAge(user: User) = user.age match {
  case Some(14) ⇒ print("passport required")
  case Some(40) ⇒ print("change your passport")
}
```

```
warning: match may not be exhaustive.
It would fail on the following inputs:
                        None, Some((x: Int forSome x not in (14, 40)))
    def printAge(user: User) = user.age match {
                               ^
```

```
def printUser(user: User) = user match {
```

```
def printUser(user: User) = user match {
  case User("Matvey", _) ⇒ print("Welcome back")
```

```
def printUser(user: User) = user match {
  case User("Matvey", _) ⇒ print("Welcome back")

  case User(name, Some(age)) ⇒
    print(s"You're at $age, mister $name")
```

```
def printUser(user: User) = user match {
  case User("Matvey", _) ⇒ print("Welcome back")

  case User(name, Some(age)) ⇒
    print(s"You're at $age, mister $name")

  case User(name, None) ⇒
    print(s"You have no age set, mister $name")
}
```

```
def printUser(user: User) = user match {
  case User("Matvey", _) ⇒ print("Welcome back")

  case User(name, Some(age)) ⇒
    print(s"You're at $age, mister $name")

  case User(name, None) ⇒
    print(s"You have no age set, mister $name")
}

//all cases here, no warnings
```

```
val users = Seq(m, b, y)
```

```
val users = Seq(m, b, y)

users.foreach {
  case User(_, Some(14)) ⇒ print("make a passport")

  case _ ⇒ print("don't worry")
}
```

Безопасность

Безопасность

- ошибиться сложно

Безопасность

- ошибиться сложно
- никаких NPE

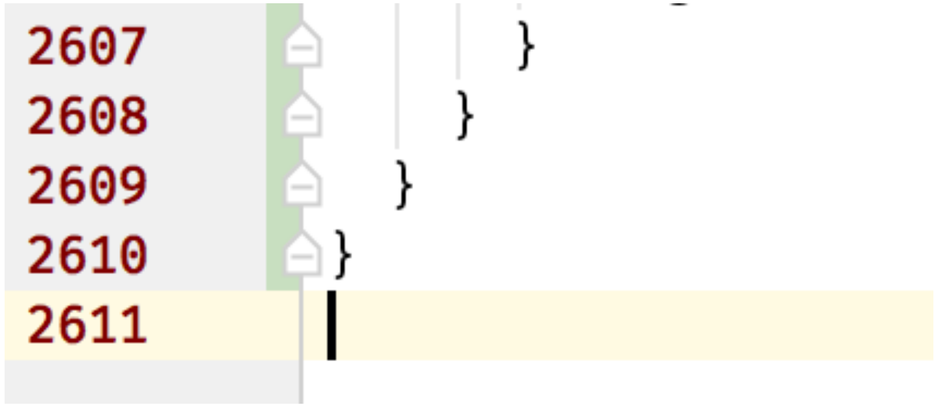
Безопасность

- ошибиться сложно
- никаких NPE
- хорошая читаемость

Безопасность

- ошибиться сложно
- никаких NPE
- хорошая читаемость
- куча операций

2. Разделяй и властвуй


2607
2608
2609
2610
2611
}
}
}
}

```
trait Logger {
  val tag = getClass.getSimpleName

  def info(msg: String) = Log.info(tag, msg)
}
```

```
trait Logger {
  val tag = getClass.getSimpleName

  def info(msg: String) = Log.info(tag, msg)
}
```

```
trait Logger {
  val tag = getClass.getSimpleName

  def info(msg: String) = Log.info(tag, msg)
}
```

```
trait Logger {
  val tag = getClass.getSimpleName

  def info(msg: String) = Log.info(tag, msg)
}

class AwesomeFragment extends Fragment
  with Logger {



}
```

```
trait Logger {
  val tag = getClass.getSimpleName

  def info(msg: String) = Log.info(tag, msg)
}

class AwesomeFragment extends Fragment
  with Logger {

  info("constructor called")

}
```

```
trait Backstack {
  this: Fragment ⇒
```

```
trait Backstack {
  this: Fragment ⇒

  def open(f: Fragment) = getChildFragmentManager
    .beginTransaction()
    .add(R.id.childRoot, f, null)
    .commit()
}
```

```
trait Backstack {
  this: Fragment ⇒

  def open(f: Fragment) = getChildFragmentManager
    .beginTransaction()
    .add(R.id.childRoot, f, null)
    .commit()
}
```

```
trait Backstack {
  this: Fragment ⇒

  def open(f: Fragment) = getChildFragmentManager
    .beginTransaction()
    .add(R.id.childRoot, f, null)
    .commit()
}

class AwesomeFragment extends Fragment
  with Logger
  with Backstack {

  info("constructor called")

  def onConfirmClick = open(new ConfirmFragment())
}
```

```
trait Backstack {
  this: Fragment ⇒

  def open(f: Fragment) = getChildFragmentManager
    .beginTransaction()
    .add(R.id.childRoot, f, null)
    .commit()
}

class AwesomeFragment extends Fragment
  with Logger
  with Backstack {

  info("constructor called")

  def onConfirmClick = open(new ConfirmFragment())
}
```

```
class ChatFragment
  extends Fragment
    with MessagesLoading
    with ChatItemsClicks
    with ChatActionBar
    with ChatMenus {
```

```
class ChatFragment
  extends Fragment
    with MessagesLoading
    with ChatItemsClicks
    with ChatActionBar
    with ChatMenus {
```

```
class RootActivity
  extends BaseActivity
    with Shaker
    with BackStack
    with DrawerHolder {
```

Разделяй и властвуй

- модульность
- читаемость
- переиспользуемость

3. Идеальная архитектура

Чистые функции

Чистые функции

- уверенность в результате

Чистые функции

- уверенность в результате
- легко тестировать

Чистые функции

- уверенность в результате
- легко тестировать
- легко кешировать

Чистые функции

- уверенность в результате
- легко тестировать
- легко кешировать
- легко переиспользовать

Service aka Helper

Service aka Helper

- только бизнеслогика

Service aka Helper

- только бизнеслогика
  - определенная ее часть

Service aka Helper

- только бизнеслогика
  - определенная ее часть
- только чистые функции (>90%)

Service aka Helper

- только бизнеслогика
  - определенная ее часть
- только чистые функции (>90%)
- scala companion objects

```
object MediaHelper {

  def extractFileFromUri(uri: Uri): Future[String] = …

  def makeMedia(id: Long, source: Source): Option[MediaDb] = …

  def saveFileToGallery(name: String, mt: MediaType): Future[String] = …
```

```
object MediaHelper {

  def extractFileFromUri(uri: Uri): Future[String] = …

  def makeMedia(id: Long, source: Source): Option[MediaDb] = …

  def saveFileToGallery(name: String, mt: MediaType): Future[String] = …
```

```
object ChatHelper {

  def updateChatName(chatId: Long, name: String): Unit = …

  def updateChatAvatar(chatId: Long, avatar: String): Unit = …
```

```
object ChatHelper {

  def updateChatName(chatId: Long, name: String): Unit = …

  def updateChatAvatar(chatId: Long, avatar: String): Unit = …
```

```
object ChatHelper {

  def updateChatName(chatId: Long, name: String): Unit = …

  def updateChatAvatar(chatId: Long, avatar: String): Unit = …


Publisher.ChatChanged.publish(
  ChatChangedEvent(chatId, name = newName)
)
```

```
object ChatHelper {

  def updateChatName(chatId: Long, name: String): Unit = …

  def updateChatAvatar(chatId: Long, avatar: String): Unit = …


Publisher.ChatChanged.publish(
  ChatChangedEvent(chatId, name = newName)
)


Publisher.ChatChanged.subscribe(onChatChanged)
Publisher.ChatChanged.unsubscribe(onChatChanged)
```

```
object DbHelper
  extends DatabaseDsl
    with BaseDbHelper
    with KeyDbHelper
    with ChatDbHelper
    with MessageDbHelper
    with NoticeDbHelper
    with Logger {
```

```
object DbHelper
  extends DatabaseDsl
    with BaseDbHelper
    with KeyDbHelper
    with ChatDbHelper
    with MessageDbHelper
    with NoticeDbHelper
    with Logger {
```

```
object DbHelper
  extends DatabaseDsl
    with BaseDbHelper
    with KeyDbHelper
    with ChatDbHelper
    with MessageDbHelper
    with NoticeDbHelper
    with Logger {
```

Идеальная архитектура

Идеальная архитектура

- расширяемость

Идеальная архитектура

- расширяемость
- поддерживаемость

Идеальная архитектура

- расширяемость
- поддерживаемость
- тестируемость

Идеальная архитектура

- расширяемость
- поддерживаемость
- тестируемость
- отсутствие лишних действий

4. Красивые DSL и UI

Красивые DSL

Красивые DSL

- мини-язык для работы с определенное подсистемой

Красивые DSL

- мини-язык для работы с определенное подсистемой
- меньше кода

Красивые DSL

- мини-язык для работы с определенное подсистемой
- меньше кода
- надежнее

Красивые DSL

- мини-язык для работы с определенное подсистемой
- меньше кода
- надежнее
- лямбды лучше методов

Красивые DSL

- мини-язык для работы с определенное подсистемой
- меньше кода
- надежнее
- лямбды лучше методов
- особенно крут в UI

TypedResources

TypedResources

- пришел с android-sbt-plugin

TypedResources

- пришел с android-sbt-plugin
- compile-time типизация айдишников

TypedResources

- пришел с android-sbt-plugin
- compile-time типизация айдишников
- жутко удобно

TypedResources

- пришел с android-sbt-plugin
- compile-time типизация айдишников
- жутко удобно
- безопасно

TypedResources

- пришел с android-sbt-plugin
- compile-time типизация айдишников
- жутко удобно
- безопасно
- приходится компилировать чаще

```
TextView tv = (TextView) getView()
   .findViewById(R.id.title)
```

```
TextView tv = (TextView) getView()
  .findViewById(R.id.title)

val tv = getView.findViewById(R.id.title)
  .asInstanceOf[TextView]
```

```
TextView tv = (TextView) getView()
  .findViewById(R.id.title)
```

```
val tv = getView.findViewById(R.id.title)
  .asInstanceOf[TextView]
```

```
val tv = getView.find(TR.title)
```

```
TextView tv = (TextView) getView()
  .findViewById(R.id.title)
```

```
val tv = getView.findViewById(R.id.title)
  .asInstanceOf[TextView]
```

```
val tv = getView.find(TR.title)
```

Методы жизненного цикла

Методы жизненного цикла

- разрастаются

Методы жизненного цикла

- разрастаются
- используются всегда

Методы жизненного цикла

- разрастаются
- используются всегда
- содержат кучу ненужных слов

Методы жизненного цикла

- разрастаются
- используются всегда
- содержат кучу ненужных слов

```
@Override
public void onViewCreated(View view, Bundle
savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    //код здесь
}
```

Методы жизненного цикла

- разрастаются
- используются всегда
- содержат кучу ненужных слов

```
@Override
public void onViewCreated(View view, Bundle
savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    //код здесь
}
```

```
trait HostedFragment extends Fragment with Logger {
```

```
trait HostedFragment extends Fragment with Logger {

  protected val onCreateBodies = new ArrayBuffer[() ⇒ Unit]
```

```
trait HostedFragment extends Fragment with Logger {

  protected val onCreateBodies = new ArrayBuffer[() ⇒ Unit]
```

```
trait HostedFragment extends Fragment with Logger {

  protected val onCreateBodies = new ArrayBuffer[() ⇒ Unit]

  def onCreate(body: ⇒ Any): Unit = {
    onCreateBodies += { () ⇒
      body
      ()
    }
  }
```

```
trait HostedFragment extends Fragment with Logger {

  protected val onCreateBodies = new ArrayBuffer[() ⇒ Unit]

  def onCreate(body: ⇒ Any): Unit = {
    onCreateBodies += { () ⇒
      body
      ()
    }
  }
```

```
trait HostedFragment extends Fragment with Logger {

  protected val onCreateBodies = new ArrayBuffer[() ⇒ Unit]

  def onCreate(body: ⇒ Any): Unit = {
    onCreateBodies += { () ⇒
      body
      ()
    }
  }

  override def onCreate(savedInstanceState: Bundle): Unit = {
    super.onCreate(savedInstanceState)
    onCreateBodies.foreach(_ ())
  }
```

```
class AwesomeFragment extends HostedFragment {

  onCreate {
    //do some stuff
  }
```

```
class AwesomeFragment extends HostedFragment {

  onCreate {
    //do some stuff
  }

  onCreate {
    //do more stuff
  }
```

```
class AwesomeFragment extends HostedFragment {

  onCreate {
    //do some stuff
  }

  onCreate {
    //do more stuff
  }

  onViewCreated {
    getView.find(TR.title).setText("I love Scala")
  }
```

```
trait BackStackApi {

  def clear(): Unit

  def removeLast(): Unit

  def last: Option[HostedFragment]

  def removeUntil[F <: HostedFragment]: Boolean
```

```
trait BackStackApi {

  def clear(): Unit

  def removeLast(): Unit

  def last: Option[HostedFragment]

  def removeUntil[F <: HostedFragment]: Boolean


removeUntil[ChatFragment]
```

package object

package object

- глобальная видимость

package object

- глобальная видимость
  - внутри пакета

package object

- глобальная видимость
  - внутри пакета
- содержит переменные и функции

```
implicit class RichView[V <: View](view: V) {
```

```
implicit class RichView[V <: View](view: V) {
```

```
implicit class RichView[V <: View](view: V) {
```

```
implicit class RichView[V <: View](view: V) {
```

```
implicit class RichView[V <: View](view: V) {

  def onClick[U](f: ⇒ U): Unit = {
    view.setOnClickListener(new View.OnClickListener {
      def onClick(p: View): Unit = f
    })
  }
```

```
implicit class RichView[V <: View](view: V) {

  def onClick[U](f: ⇒ U): Unit = {
    view.setOnClickListener(new View.OnClickListener {
      def onClick(p: View): Unit = f
    })
  }
```

```
val tv = getView.find(TR.title)
```

```
val tv = getView.find(TR.title)
new RichView(tv).onClick(findLastUnread)
```

```
val tv = getView.find(TR.title)
new RichView(tv).onClick(findLastUnread) //не очень
```

```
val tv = getView.find(TR.title)
new RichView(tv).onClick(findLastUnread) //не очень

tv.onClick(findLastUnread) // очень даже
```

```
val tv = getView.find(TR.title)
new RichView(tv).onClick(findLastUnread) //не очень

tv.onClick(findLastUnread) // очень даже


editText.afterTextChanged { text: String ⇒
  info(text)
}
```

```
@inline def dp(value: Float): Int =
  math.round(ctx.getResources.getDisplayMetrics.density * value)
```

```
@inline def dp(value: Float): Int =
  math.round(ctx.getResources.getDisplayMetrics.density * value)

@inline def color(resId: Int): Int =
  ctx.getResources.getColor(resId)
```

```
@inline def dp(value: Float): Int =
  math.round(ctx.getResources.getDisplayMetrics.density * value)

@inline def color(resId: Int): Int =
  ctx.getResources.getColor(resId)

@inline def string(resId: Int): String =
  ctx.getString(resId)
```

```
val str = string(R.string.APP_VERSION)
```

```
val str = string(R.string.APP_VERSION)

val title = getView.find(TR.title)
```

```
val str = string(R.string.APP_VERSION)

val title = getView.find(TR.title)

title.setCompoundDrawablePadding(dp(60))
```

```
val str = string(R.string.APP_VERSION)

val title = getView.find(TR.title)

title.setCompoundDrawablePadding(dp(60))

title.setTextColor(color(R.color.red))
```

Красота

5. Неприятности

Неприятности

Неприятности

- количество методов

Неприятности

- количество методов
    - сразу +15к

Неприятности

- количество методов
    - сразу +15к
- долгая сборка

Неприятности

- количество методов
    - сразу +15к
- долгая сборка
    - 120 секунд

Неприятности

- количество методов
    - сразу +15к
- долгая сборка
    - 120 секунд
- dalvik и ART не любят Scala

Неприятности

- количество методов
    - сразу +15к
- долгая сборка
    - 120 секунд
- dalvik и ART не любят Scala
- возможность выстрелить в ногу из-за незнания Scala

Неприятности

- количество методов
    - сразу +15к
- долгая сборка
    - 120 секунд
- dalvik и ART не любят Scala
- возможность выстрелить в ногу из-за незнания Scala
- поиск сотрудников

Что в итоге?

Итоги

Итоги

- мощнейший инструмент

Итоги

- мощнейший инструмент
- требует знаний и умений

Итоги

- мощнейший инструмент
- требует знаний и умений
- позволяет писать восхитительный DSL

Итоги

- мощнейший инструмент
- требует знаний и умений
- позволяет писать восхитительный DSL
- надежный

Итоги

- мощнейший инструмент
- требует знаний и умений
- позволяет писать восхитительный DSL
- надежный
- очень много плюшек при правильном использовании

Итоги

- мощнейший инструмент
- требует знаний и умений
- позволяет писать восхитительный DSL
- надежный
- очень много плюшек при правильном использовании
- долгое время компиляции

Итоги

- мощнейший инструмент
- требует знаний и умений
- позволяет писать восхитительный DSL
- надежный
- очень много плюшек при правильном использовании
- долгое время компиляции
- сложный поиск команды

Android разработка может быть приятной

Попробуйте Scala и вам не захочется обратно

Спасибо

Мальков Матвей

matveyka_jj

H102RPT4

Q & A