Кинжал в ножны или как написать свой DI-контейнер на Kotlin

Tinkoff.ru

Исхаков Марат
Android-разработчик в Тинькофф Бизнес

Приложения Тинькофф

Тинькофф

Тинькофф
Джуниор

Тинькофф
Инвестиции

Тинькофф
Мобайл

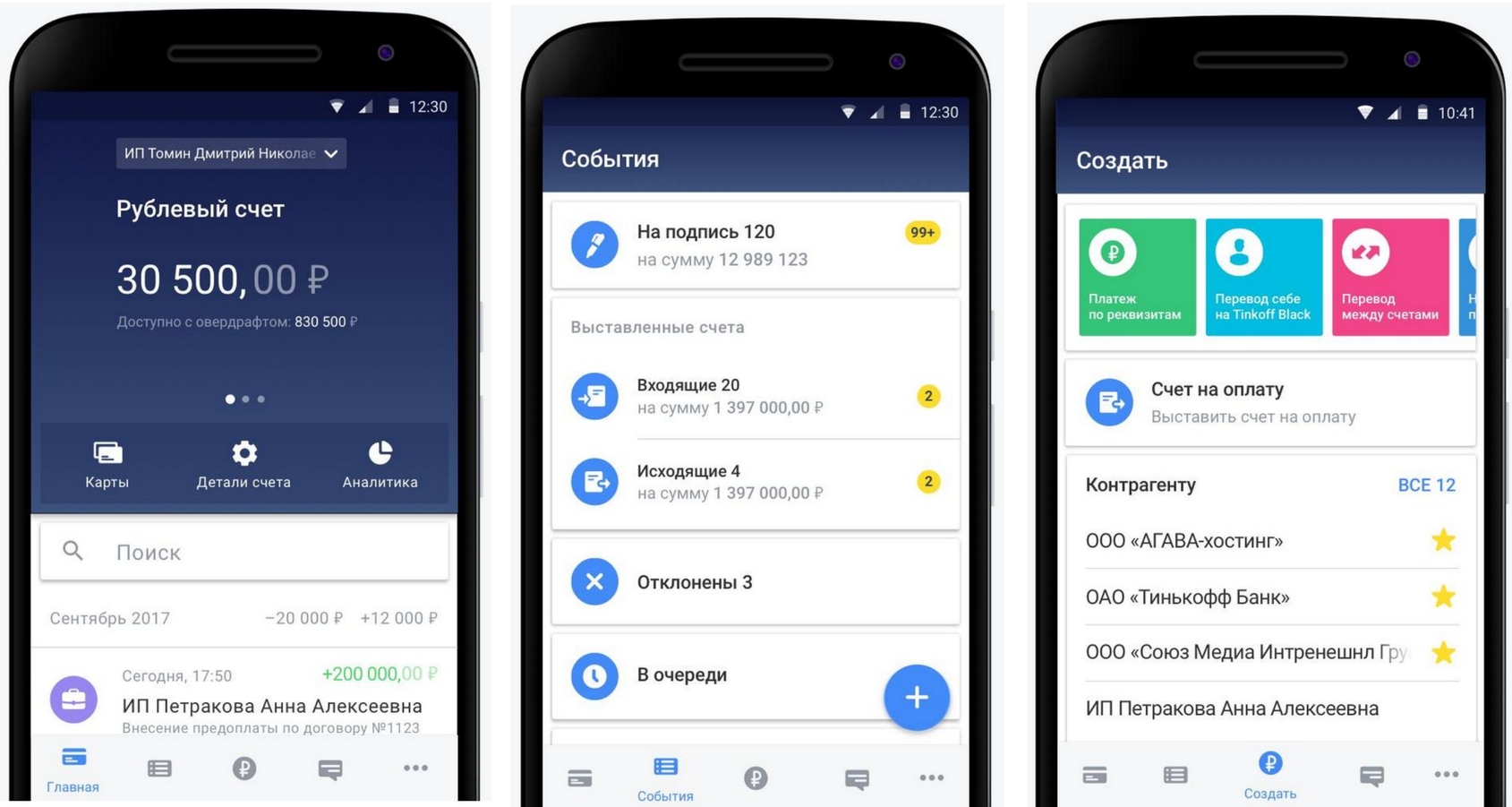
Тинькофф
Бизнес

Тинькофф
Бухгалтерия

Тинькофф
Клиенты и Проекты

Тинькофф Бизнес (Android)

- Мобильное приложение для Юр. лиц
- 15 разработчиков
- 182K java + 220K kotlin LoC
- Многомодульность (120+ модулей)
- ~5 минут – clean build



Поддерживаем актуальность

#https://blog.jetbrains.com/kotlin/2018/01/kotlin-1-2-20-is-out
kotlin.incremental.usePreciseJavaTracking=true

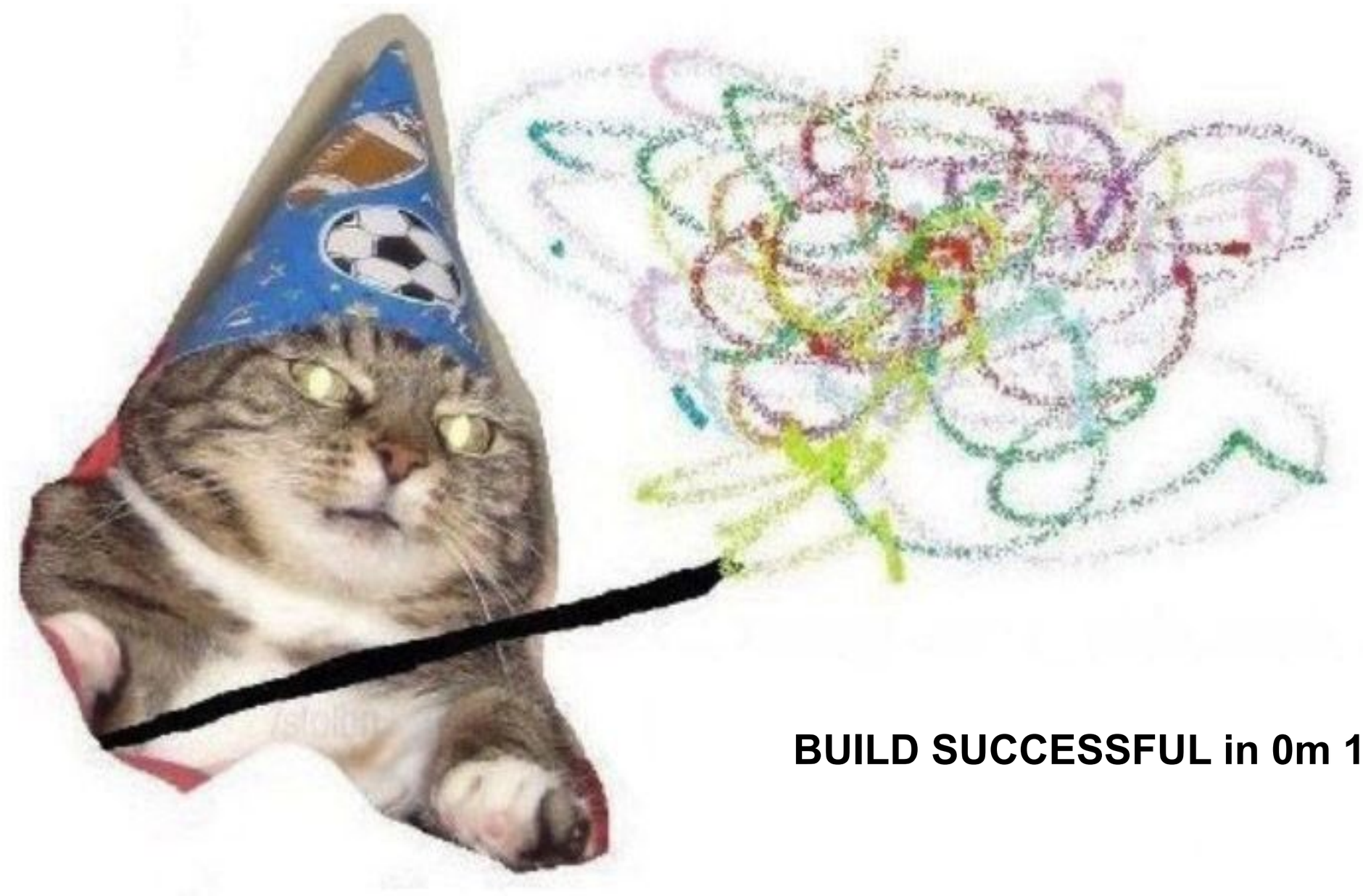
#https://kotlinlang.org/docs/reference/kapt.html#compile-avoidance-for-kapt-since-1320
kapt.include.compile.classpath=false

#https://kotlinlang.org/docs/reference/kapt.html#running-kapt-tasks-in-parallel-since-1260
kapt.use.worker.api=true

#https://kotlinlang.org/docs/reference/kapt.html#incremental-annotation-processing-since-1330
kapt.incremental.apt=true

~~Идеальный~~ билд

Идеальный билд



BUILD SUCCESSFUL in 0m 1s

~2k16 Больше кодогенерации!

- Project Lombok
- Icepick (saveInstance/restoreInstance-State)
- Google’s AutoValue
- Android annotations

- Dagger 2
- etc

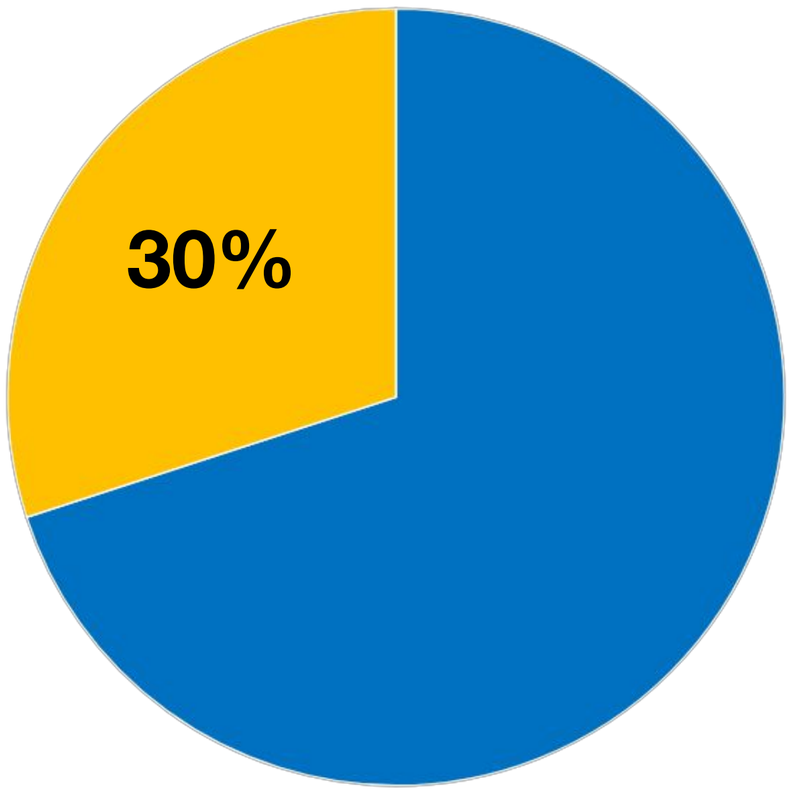
Рефлексии – «Нет»!

	NEXUS 5 (6.0) ART	GALAXY S5 (5.0) ART	GALAXY S3 mini (4.1.2) Dalvik
getFields	1108	1626	27083
getDeclaredFields	347	951	7687
getGenericInterfaces	16	23	2927
getGenericSuperclass	247	298	
makeAccessible	14	147	
getObject	21	167	
setObject	21	201	
createDummyItems	312	358	
createDummyItemsWithReflection	1332	6384	

https://blog.nimbledroid.com/2016/02/23/slow-Android-reflection.html

Кодогенерации – «Нет»?

Gradle Build Scan



Annotation Processing

Other tasks

А что у нас?

- Google’s AutoValue (MVI’s States)
- Room
- Dagger 2

Сажаем Dagger на диету

- ~~Subcomponents~~
- static provides в модулях
- Экономим на @Provides-методах
- Переиспользуем модули (!)
- Переиспользуем компоненты (?!)
- Не используем @Inject в поля

DI-контейнер на минималках

Что такое Component?

```
interface FeatureComponent {
    val presenter: Presenter
    val repository: Repository
}
```

DI-контейнер на минималках

Реализация FeatureComponent

```
class FeatureComponentImpl(
    private val deps: FeatureComponentDependencies
) : FeatureComponent {

    override val presenter: Presenter
        get() = Presenter(repository)

    override val repository: Repository
        get() = Repository(deps.retrofitBuilder.buildApi())
}
```

```
interface FeatureComponentDependencies {

    val retrofitBuilder: Retrofit.Builder

}
```

DI-контейнер на минималках

Выносим зависимости в модули

```
interface FeatureComponent {
    val presenter: Presenter
    val repository: Repository
}
```

=

```
interface FeatureModule {
    val presenter: Presenter
}
```

+

```
interface DataModule {
    val repository: Repository
}
```

+

```
interface FeatureComponent
    : FeatureModule, DataModule {
}
```

DI-контейнер на минималках

Реализация модулей

```
class DataModuleImpl(
    private val deps: FeatureComponentDependencies
) : DataModule {

    override val repository: Repository
        get() = Repository(deps.retrofitBuilder.buildApi())
}


class FeatureModuleImpl(
    private val dataModule: DataModule
) : FeatureModule {

    override val presenter: Presenter
        get() = Presenter(dataModule.repository)
}
```

DI-контейнер на минималках

Реализация компоненты

```
class FeatureComponentImpl(

    deps: FeatureComponentDependencies,

    dataModule: DataModule,
    featureModule: FeatureModule

) : FeatureComponent,

    DataModule by dataModule,
    FeatureModule by featureModule {
}
```

Implementation by delegation

DI-контейнер на минималках

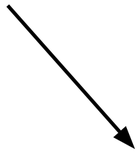
```
class DataModuleImpl(
    private val deps: FeatureComponentDependencies
) : DataModule {

    override val repository: Repository
        get() = Repository(deps.retrofitBuilder.buildApi())
}
```

```
fun DataModule(deps: FeatureComponentDependencies): DataModule {
    return object : DataModule {
        override val repository: Repository
            get() = Repository(deps.retrofitBuilder.buildApi())
    }
}
```

DI-контейнер на минималках

fun DataModule(deps: FeatureComponentDependencies): DataModule



D

Exception: factory functions used to create instances of classes can have the same name as the class being created:

```
abstract class Foo { /*...*/ }

class FooImpl : Foo { /*...*/ }

fun FooImpl(): Foo { return FooImpl() }
```

DI-контейнер на минималках

```
class FeatureComponentImpl(

    deps: FeatureComponentDependencies,

    dataModule: DataModule,
    featureModule: FeatureModule

) : FeatureComponent,

    DataModule by dataModule,
    FeatureModule by featureModule {
    }
```

```
fun FeatureComponent(deps: FeatureComponentDependencies): FeatureComponent {
    val dataModule = DataModule(deps)
    return object : FeatureComponent,
        DataModule by dataModule,
        FeatureModule by FeatureModule(dataModule) {}
}
```

Сравним с Dagger 2

Module

```
@Module
class DataModule {

    @Provides
    fun provideRepository(
        retrofitBuilder: Retrofit.Builder
    ): Repository {
        return Repository(retrofitBuilder.buildApi())
    }
}
```

```
interface DataModule {
    val repository: Repository
}

fun DataModule(deps: FeatureComponentDependencies): DataModule {
    return object : DataModule {
        override val repository: Repository
            get() = Repository(deps.retrofitBuilder.buildApi())
    }
}
```

```
@Module
class FeatureModule {

    @Provides
    fun providePresenter(
        repository: Repository
    ): Presenter {
        return Presenter(repository)
    }
}
```

```
interface FeatureModule {
    val presenter: Presenter
}

fun FeatureModule(dataModule: DataModule): FeatureModule {
    return object : FeatureModule {
        override val presenter: Presenter
            get() = Presenter(dataModule.repository)
    }
}
```

Сравним с Dagger 2

Component

```
@Component(
    dependencies = [FeatureComponentDependencies::class],
    modules = [FeatureModule::class, DataModule::class]
)
interface FeatureComponent {
    val presenter: Presenter
    val repository: Repository
}

fun initComponent(
    deps: FeatureComponentDependencies
): FeatureComponent {
    return DaggerFeatureComponent.builder()
        .featureComponentDependencies(deps)
        .build()
}
```

```
interface FeatureComponent
    : FeatureModule, DataModule

fun FeatureComponent(
    deps: FeatureComponentDependencies
): FeatureComponent {
    val dataModule = DataModule(deps)
    return object : FeatureComponent,
        DataModule by dataModule,
        FeatureModule by FeatureModule(dataModule) {}
}
```

Сравним с Dagger 2

Возможности «из коробки»

- Unscoped

```
override val presenter: Presenter
    get() = Presenter(dataModule.repository)
```

- Scoped

```
override val presenter = Presenter(dataModule.repository)
```

- Ленивая инициализация

```
override val presenter by lazy { Presenter(dataModule.repository) }
```

- Provider<>

```
override val presenter = Provider { Presenter(dataModule.repository) }
```

- Qualifiers

```
override val firstPresenter by lazy { Presenter(dataModule.repository) }
override val secondPresenter by lazy { Presenter(dataModule.repository) }
```

… и так далее

Пример: WeakLazy

```
fun <T> weakLazy(factory: () -> T) = object : ReadOnlyProperty<Any?, T> {
    private var weakRef: WeakReference<T>? = null

    override fun getValue(thisRef: Any?, property: KProperty<*>): T {
        synchronized(this) {
            return weakRef?.get()
                ?: factory().also {
                    weakRef = WeakReference(it)
                }
        }
    }
}
```

Итоги. Плюсы

1. Полный контроль над DI (без «магии»)
2. Возможность постепенного перехода
3. Нет зависимости от di-фреймворка
4. Пропущенные провайды видно сразу в IDE
5. Нет кодогенерации, но compile-time safety*

* Почти 😉, см. минусы

Итоги. Минусы

1. Нет защиты от циклических зависимостей
2. Писать все «руками» (без @Inject и пр.)
3. Нет «шаринга» модулей
4. TBD

Циклические зависимости

```
fun FeatureModule(): FeatureModule {
    return object : FeatureModule {
        override val manager
            get() = Manager(presenter)
        override val presenter
            get() = Presenter(manager)
    }
}
```

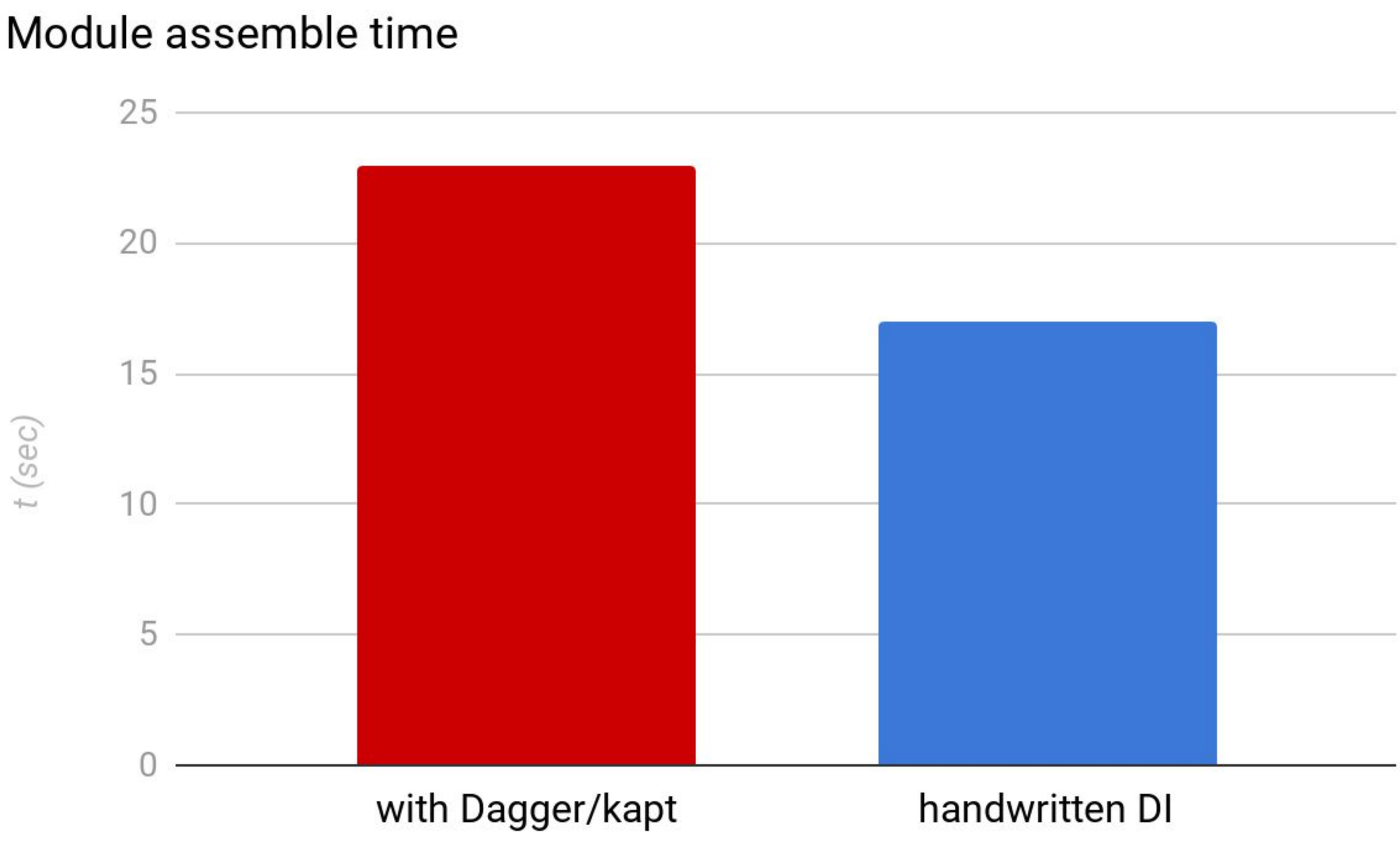
Type checking has run into a recursive problem. Easiest workaround: specify types of your declarations explicitly

А это не тоже самое, что и..?

Итоги

15 files changed, 124 insertions(+), 124 deletions(-)

Итоги. Profit

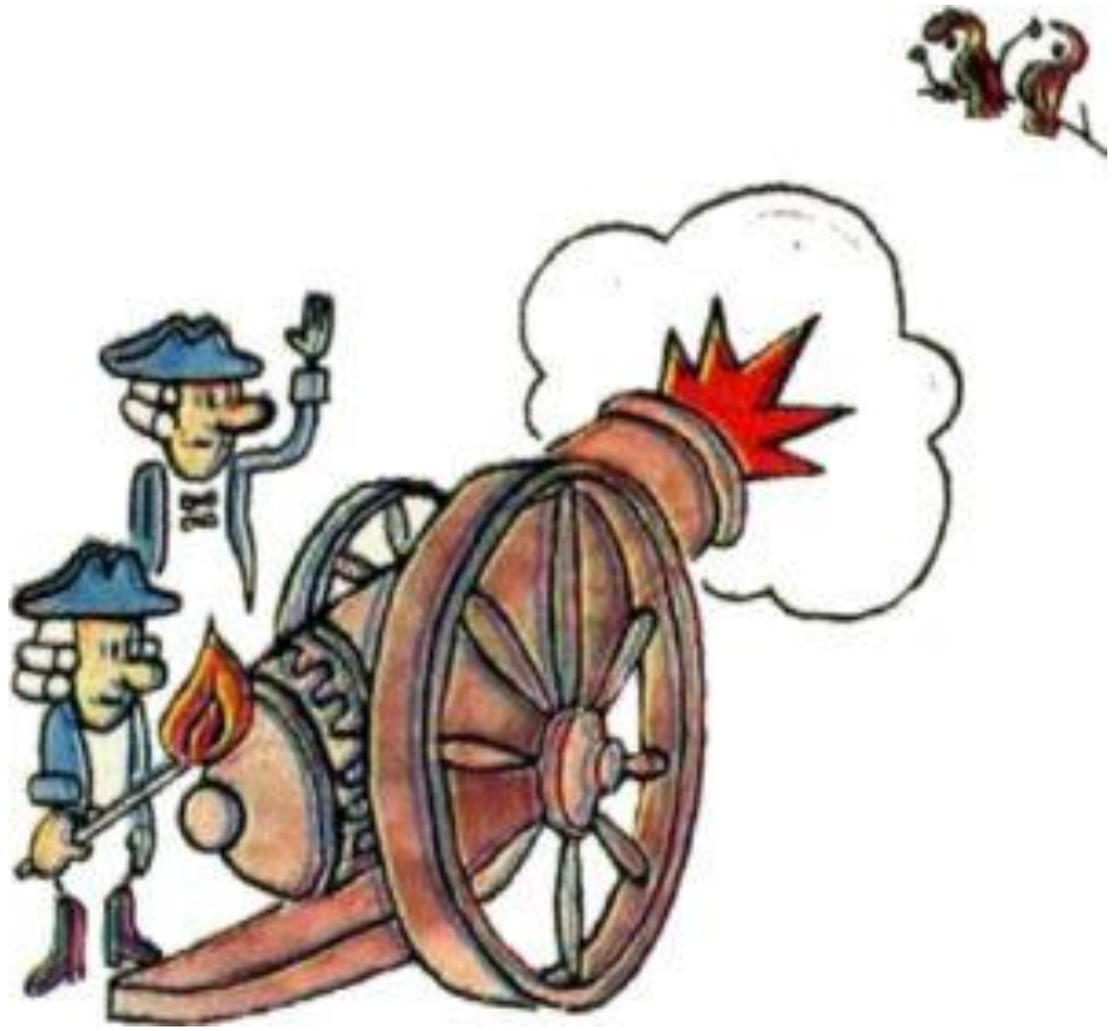


Не делайте культ из инструмента

Dagger – отличный инструмент

Без него можно обойтись

Это не сложно



Исхаков
Марат

Android-разработчик в Тинькофф Бизнес

m.r.iskhakov@tinkoff.ru

+7(905)024-57-20

…/m.r.iskhakov

@RinaToWitch

@mriskhakov

…/m.r.iskhakov

Дальше действовать будем мы!

Tinkoff.ru