TÍTULO DO SLIDE

Arquitetura em Projetos ANDROID

Carlos Nicolau Galves

Tech Lead Android

TÍTULO DO SLIDE

30 anos, 7 anos desenvolvendo aplicativos Nativos em java, kotlin, swift.

TÍTULO DO SLIDE

Empresas: Livetouch, Bradesco, Ifood, Zup Innovation.

TÍTULO DO SLIDE

Aplicativos para empresas como: Porto Seguro, Mondial, Einstein, Bradesco, Santander, Ifood, Itaú.

Arquiteturas em Projetos Android

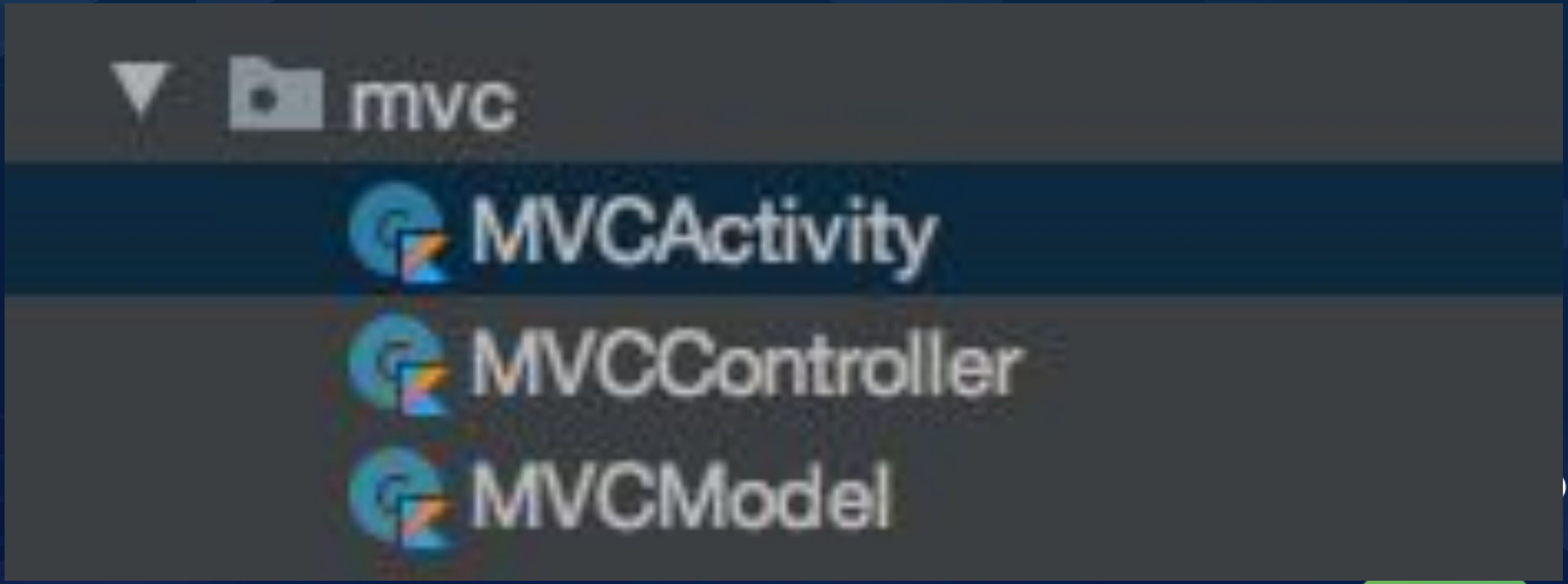
TÍTULO DO SLIDE

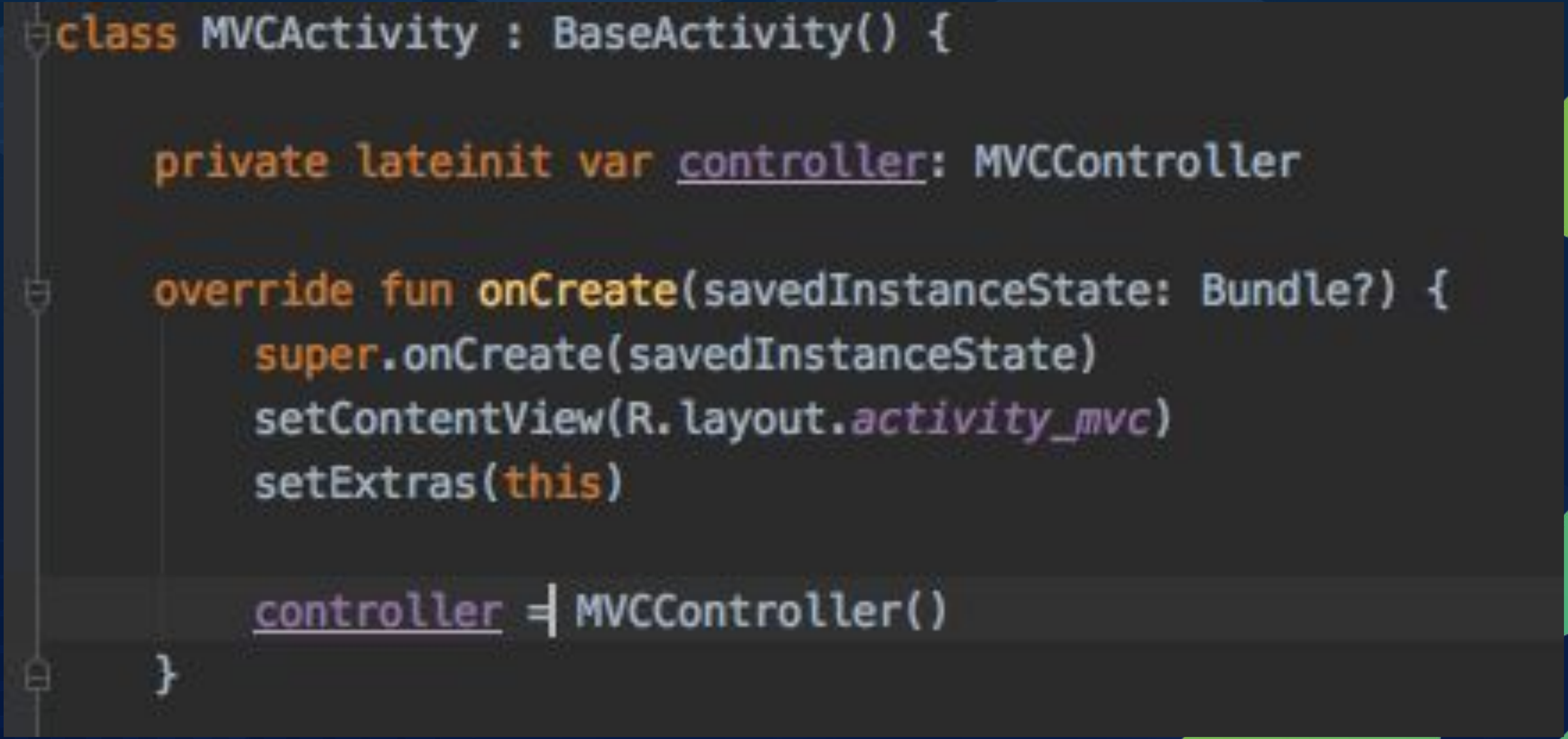
MVC

MVC

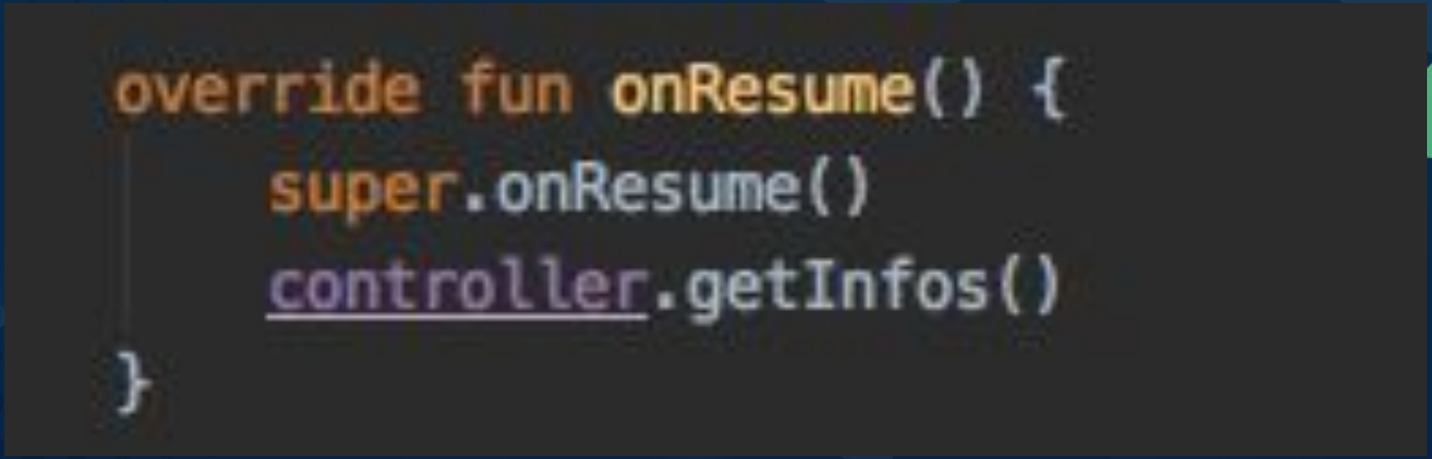
Aqui voce vera mais detalhes sobre o MVC.

https://github.com/nicconicco/Arch-DBZ/tree/master/app/src/main/java/com/nicco/architectures/android/mvc

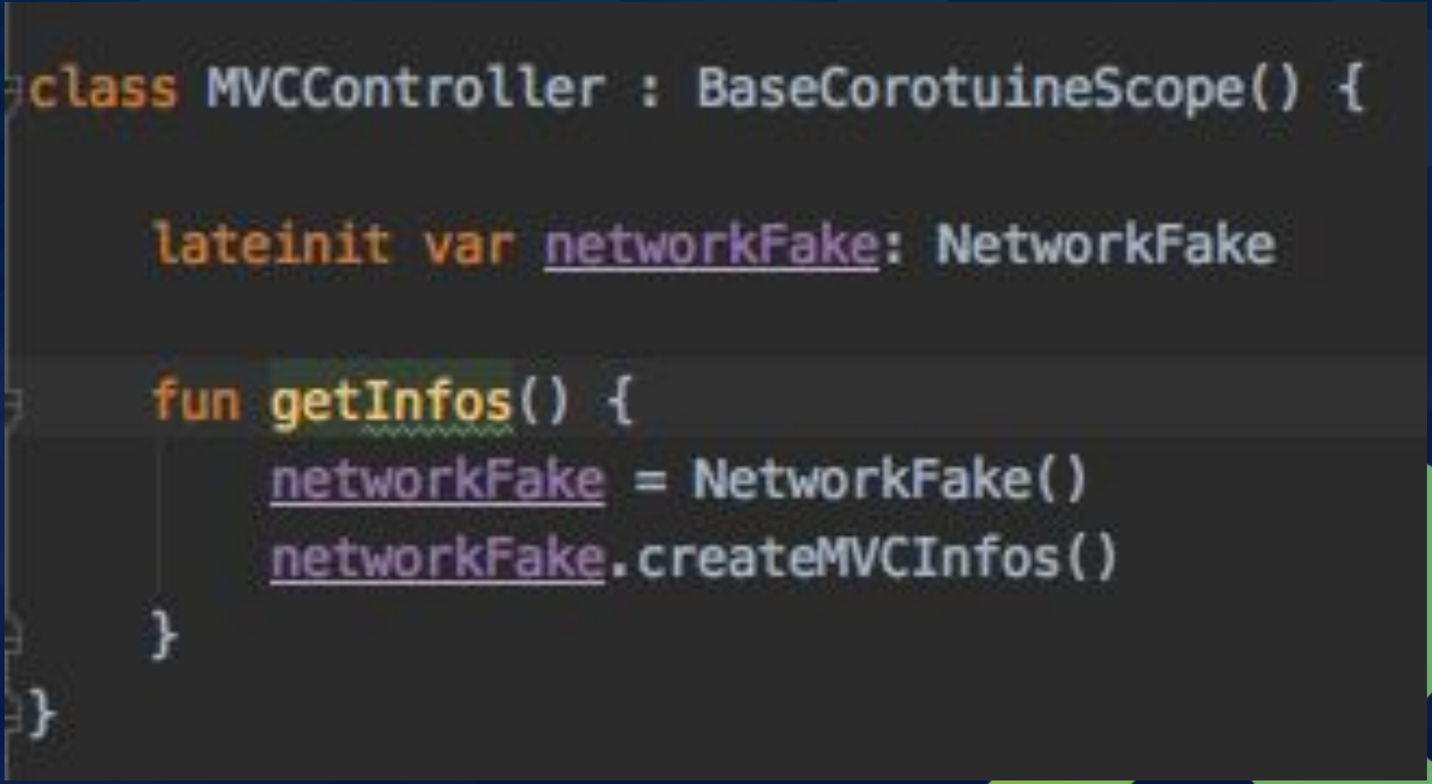

mvc
MVCActivity
MVCController
MVCModel



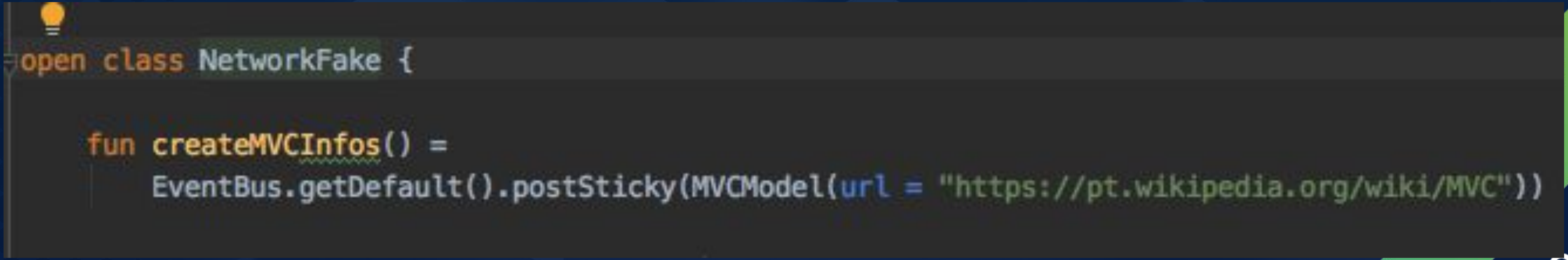
```
class MVCActivity : BaseActivity() {

    private lateinit var controller: MVCController

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mvc)
        setExtras(this)

        controller = MVCController()
    }
```



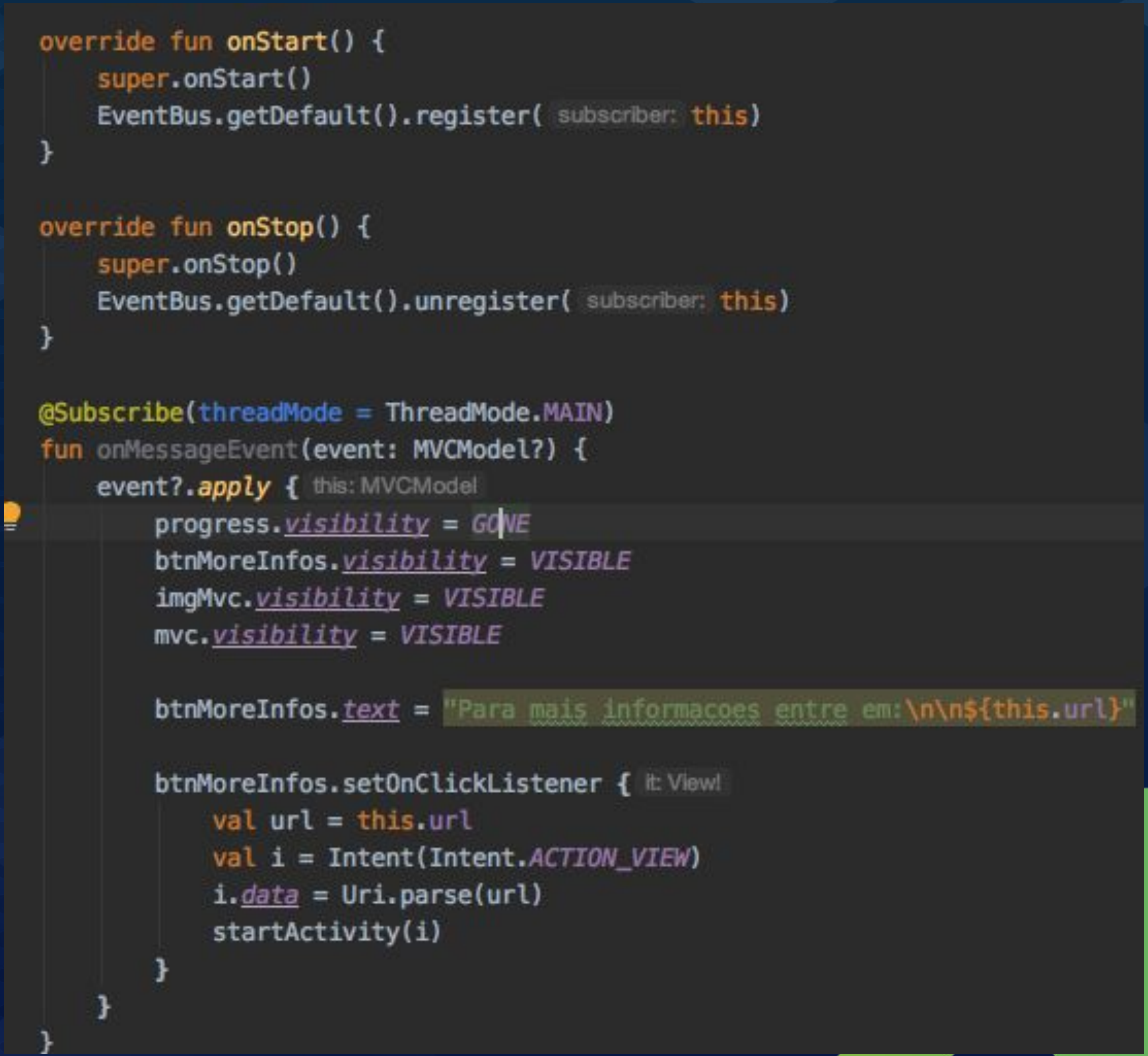
```
override fun onResume() {
    super.onResume()
    controller.getInfos()
}
```



```
class MVCController : BaseCorotuineScope() {

    lateinit var networkFake: NetworkFake

    fun getInfos() {
        networkFake = NetworkFake()
        networkFake.createMVCInfos()
    }
}
```

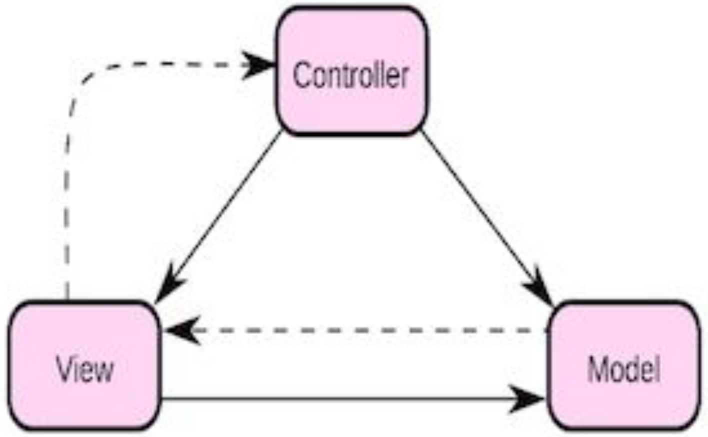


```
open class NetworkFake {

    fun createMVCInfos() =
        EventBus.getDefault().postSticky(MVCModel(url = "https://pt.wikipedia.org/wiki/MVC"))
```



```
override fun onStart() {
    super.onStart()
    EventBus.getDefault().register( subscriber: this)
}

override fun onStop() {
    super.onStop()
    EventBus.getDefault().unregister( subscriber: this)
}

@Subscribe(threadMode = ThreadMode.MAIN)
fun onMessageEvent(event: MVCModel?) {
    event?.apply { this: MVCModel
        progress.visibility = GONE
        btnMoreInfos.visibility = VISIBLE
        imgMvc.visibility = VISIBLE
        mvc.visibility = VISIBLE

        btnMoreInfos.text = "Para mais informacoes entre em:\n\n${this.url}"

        btnMoreInfos.setOnClickListener { it: View!
            val url = this.url
            val i = Intent(Intent.ACTION_VIEW)
            i.data = Uri.parse(url)
            startActivity(i)
        }
    }
}
```

MVC



Para mais informacoes entre em:

https://pt.wikipedia.org/wiki/MVC

Model View Controller

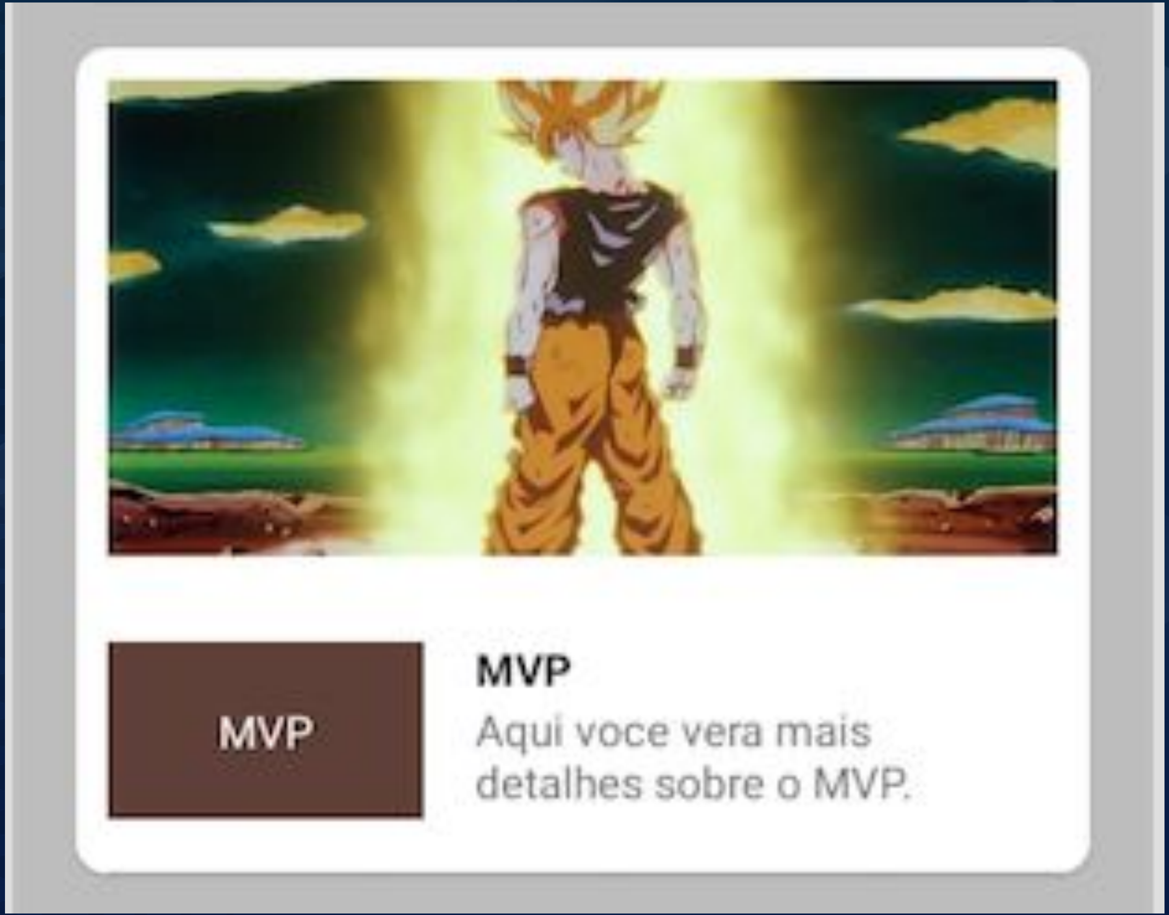
- 1 - Activity /* - View ✓
- 2 - Controller - EventBus ✓
- 3 - Model - MVCModel ✓

Vantagens: Pequeno, Rápido, Direto.

Desvatangens: Muita informação em um só lugar, manutenção disso começa a ficar confusa, Testes unitários?

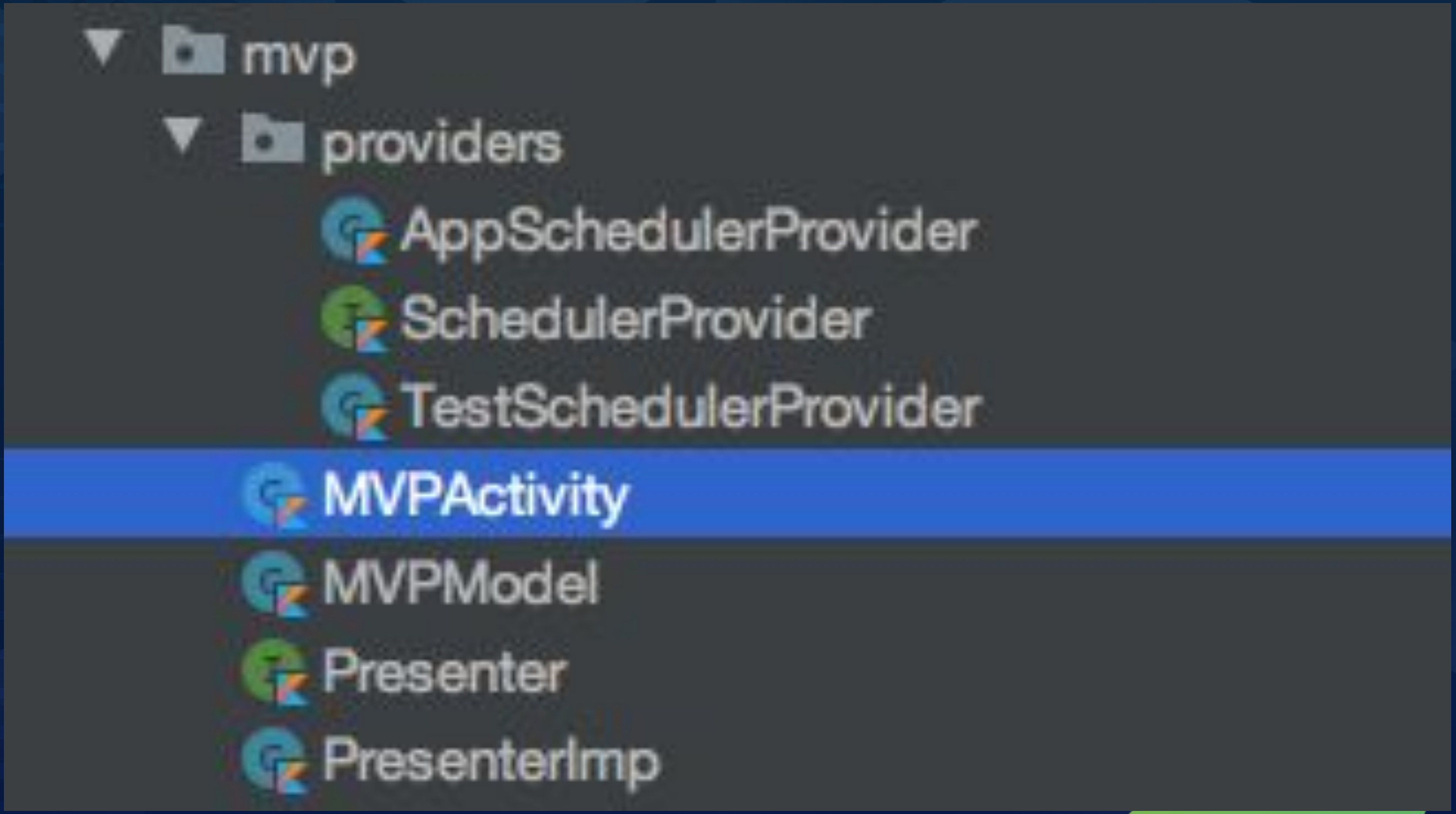
Arquiteturas em Projetos Android

- 1 - MVC ✓
- 2 - MVP
- 3 - MVVM
- 4 - MVP Clean
- 5 - MVVM Clean
- 6 - MVI

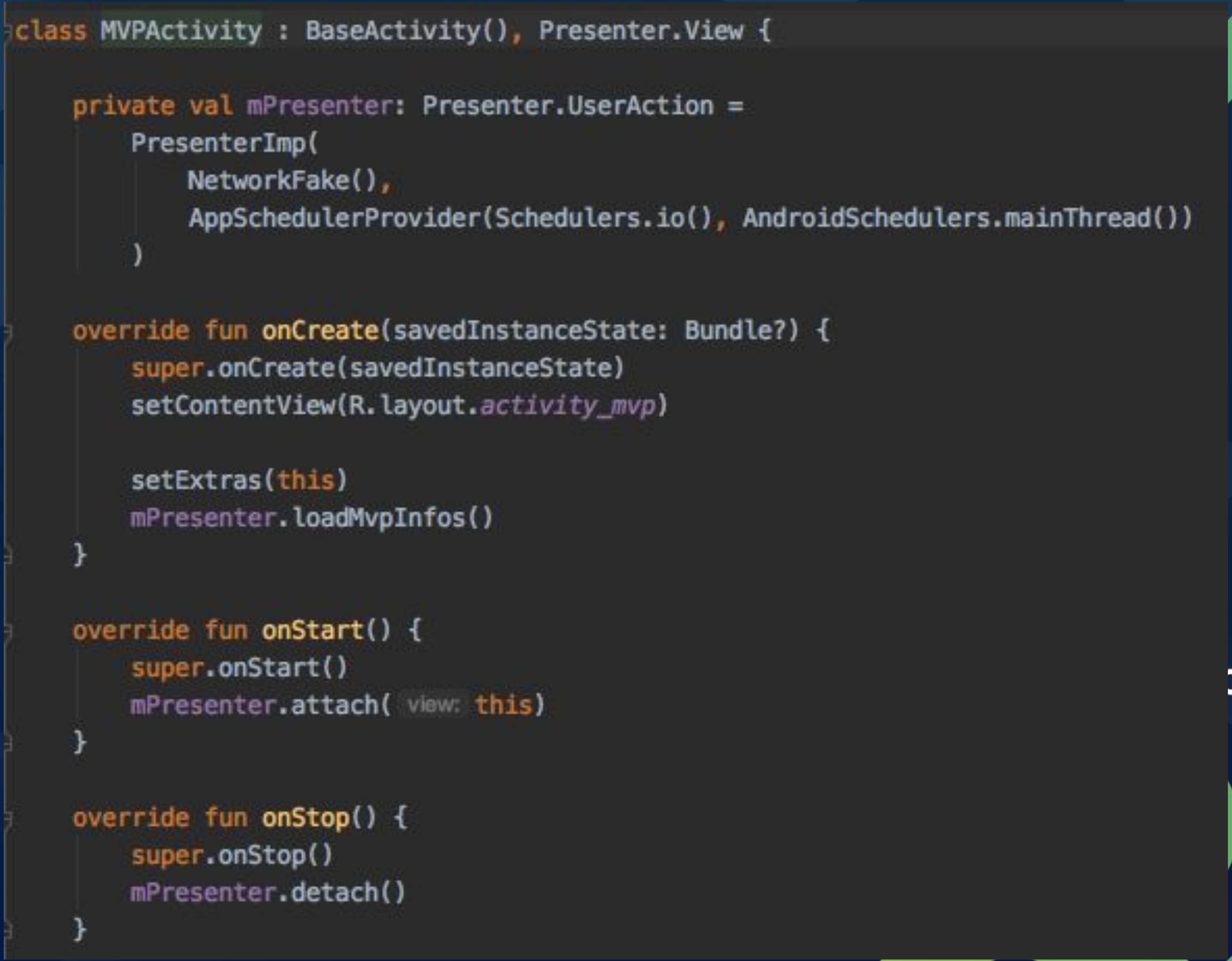

MVP

MVP
Aqui voce vera mais detalhes sobre o MVP.

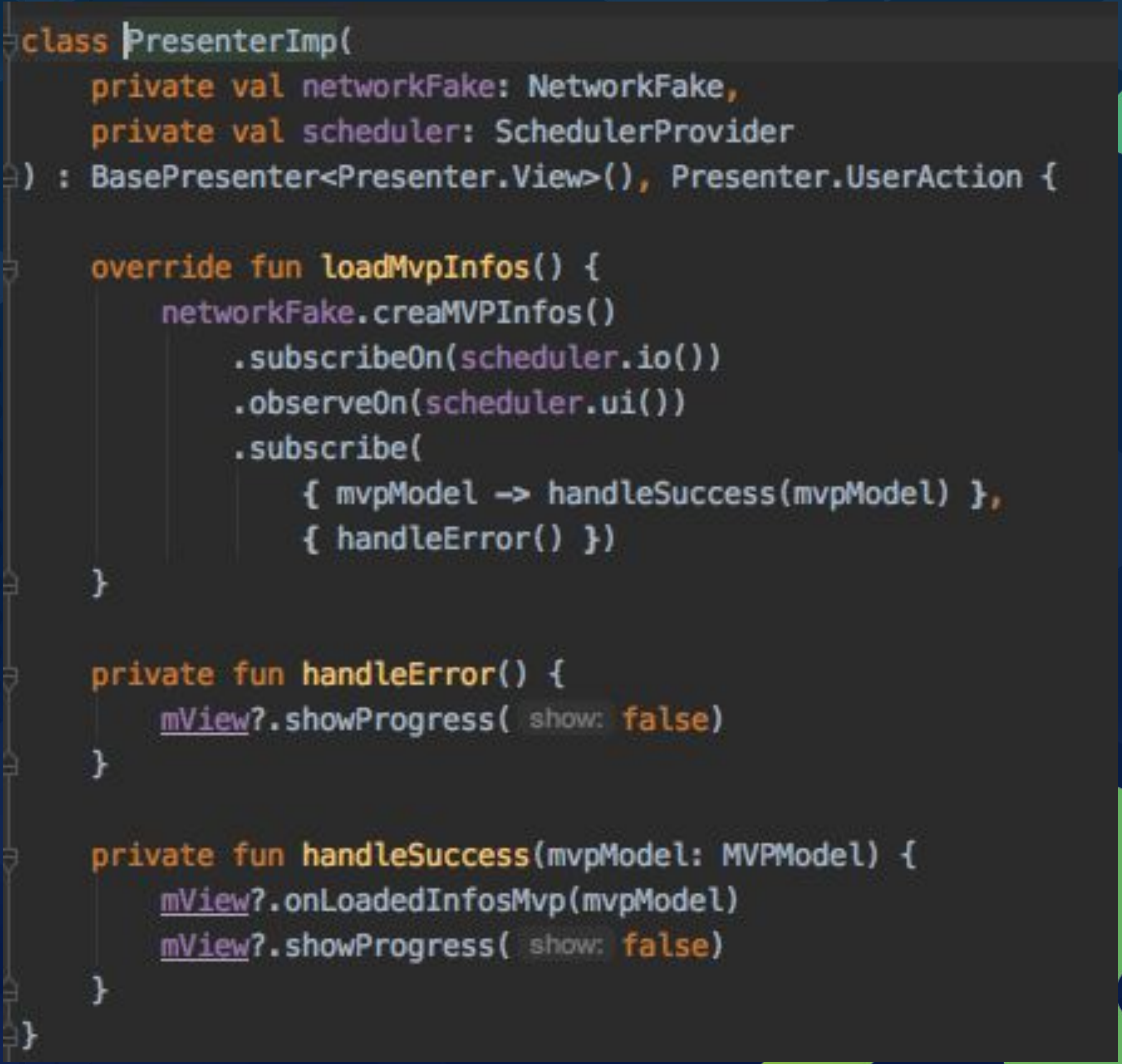
https://github.com/nicconicco/Arch-DBZ/tree/master/app/src/main/java/com/nicco/architectures/android/mvp



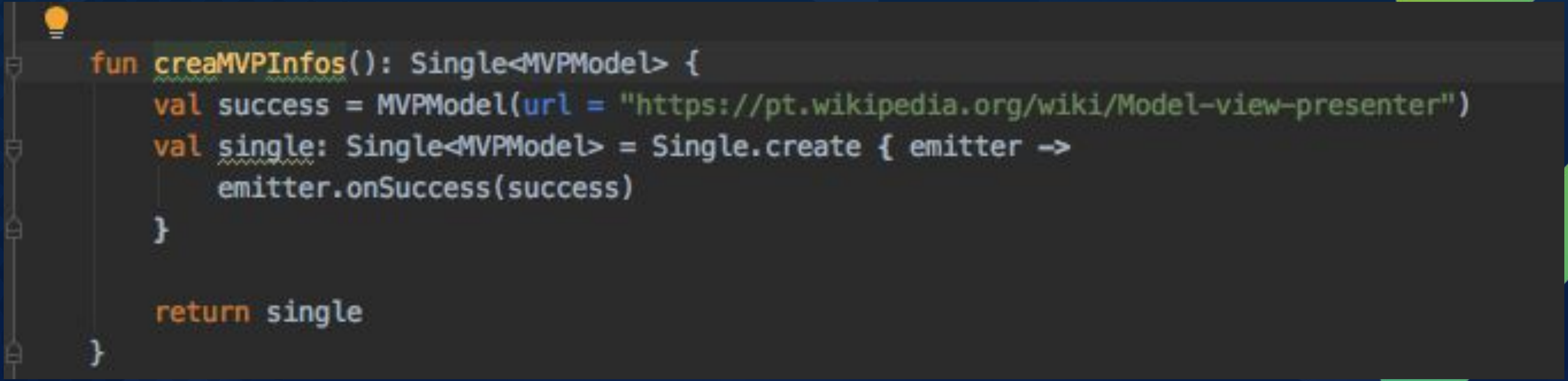
```
▼ mvp
    ▼ providers
        AppSchedulerProvider
        SchedulerProvider
        TestSchedulerProvider
    MVPActivity
    MVPModel
    Presenter
    PresenterImp
```



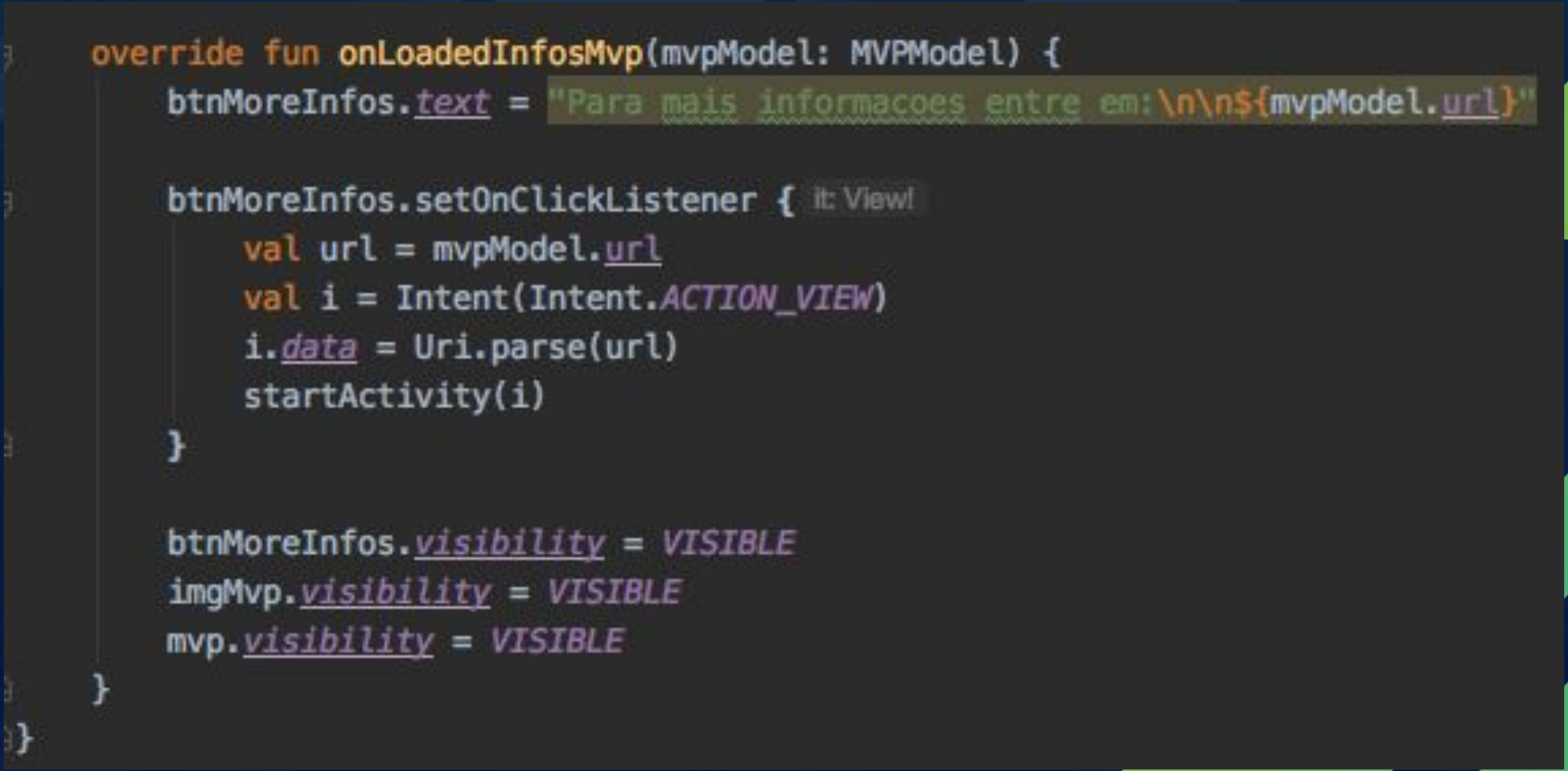
```
class MVPActivity : BaseActivity(), Presenter.View {

    private val mPresenter: Presenter.UserAction =
        PresenterImp(
            NetworkFake(),
            AppSchedulerProvider(Schedulers.io(), AndroidSchedulers.mainThread())
        )

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mvp)

        setExtras(this)
        mPresenter.loadMvpInfos()
    }

    override fun onStart() {
        super.onStart()
        mPresenter.attach( view: this)
    }

    override fun onStop() {
        super.onStop()
        mPresenter.detach()
    }
```



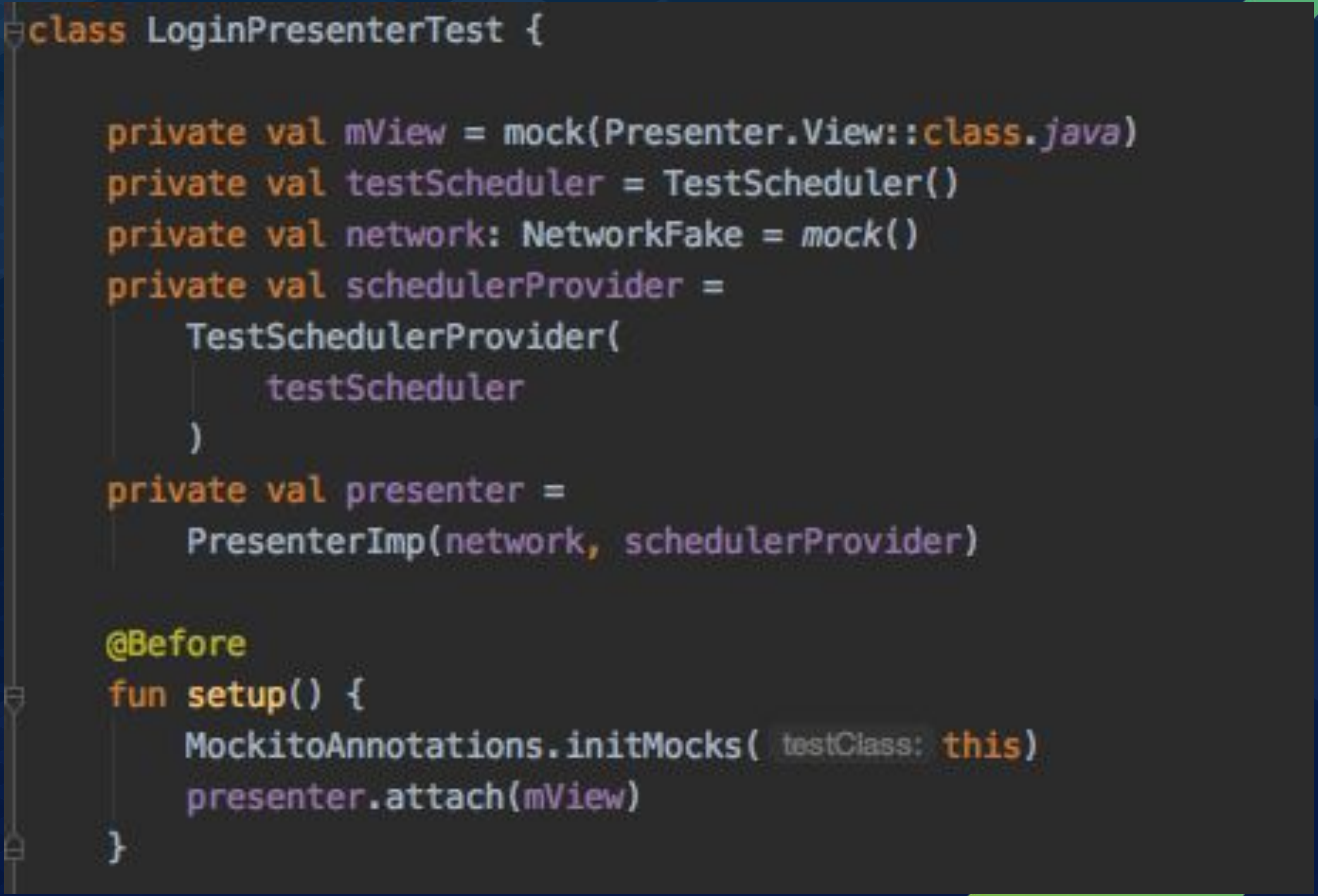
```kotlin
class PresenterImp(
    private val networkFake: NetworkFake,
    private val scheduler: SchedulerProvider
) : BasePresenter<Presenter.View>(), Presenter.UserAction {

    override fun loadMvpInfos() {
        networkFake.creaMVPInfos()
            .subscribeOn(scheduler.io())
            .observeOn(scheduler.ui())
            .subscribe(
                { mvpModel -> handleSuccess(mvpModel) },
                { handleError() })
    }

    private fun handleError() {
        mView?.showProgress( show: false)
    }

    private fun handleSuccess(mvpModel: MVPModel) {
        mView?.onLoadedInfosMvp(mvpModel)
        mView?.showProgress( show: false)
    }
}
```



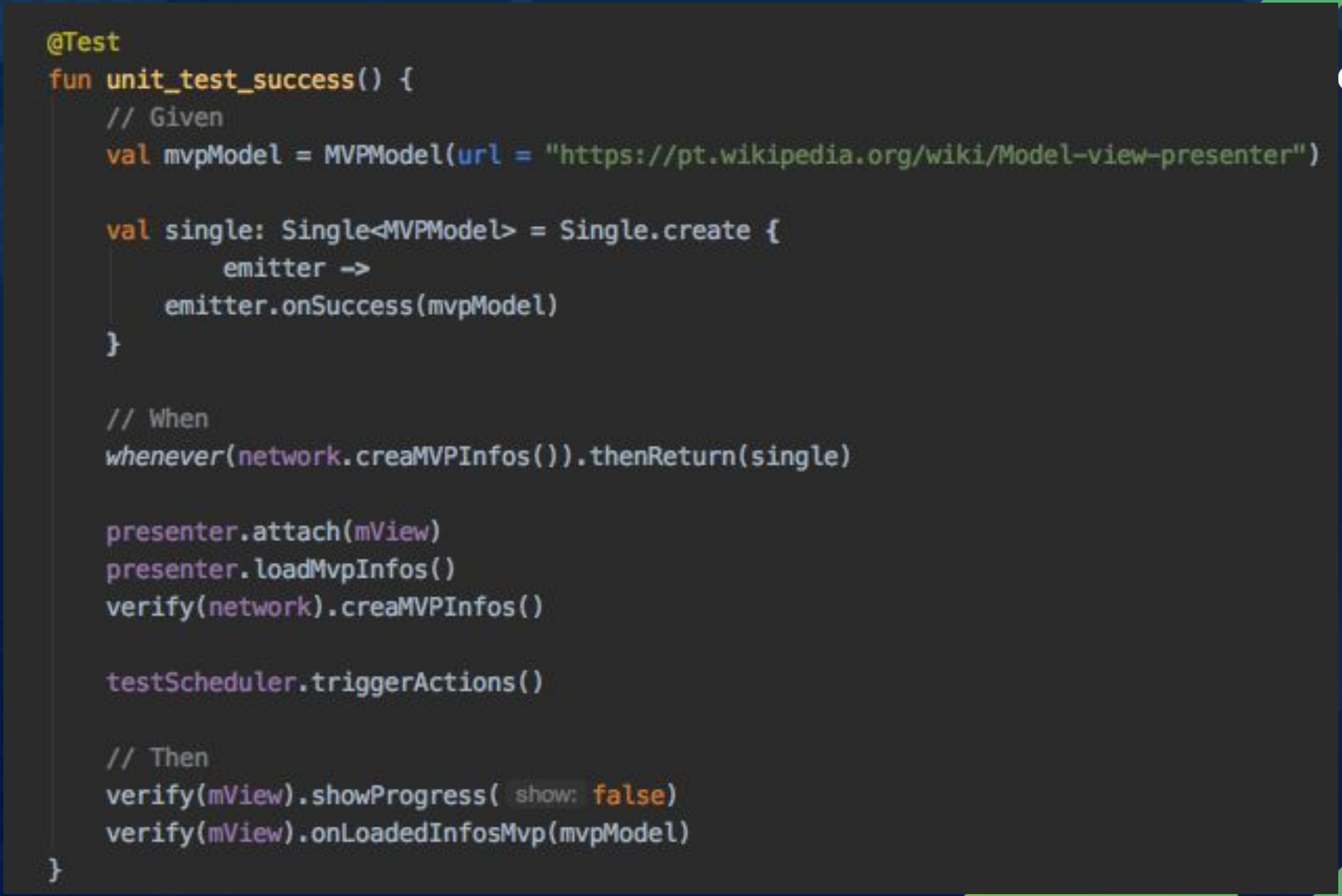
```kotlin
fun creaMVPInfos(): Single<MVPModel> {
    val success = MVPModel(url = "https://pt.wikipedia.org/wiki/Model-view-presenter")
    val single: Single<MVPModel> = Single.create { emitter ->
        emitter.onSuccess(success)
    }

    return single
}
```



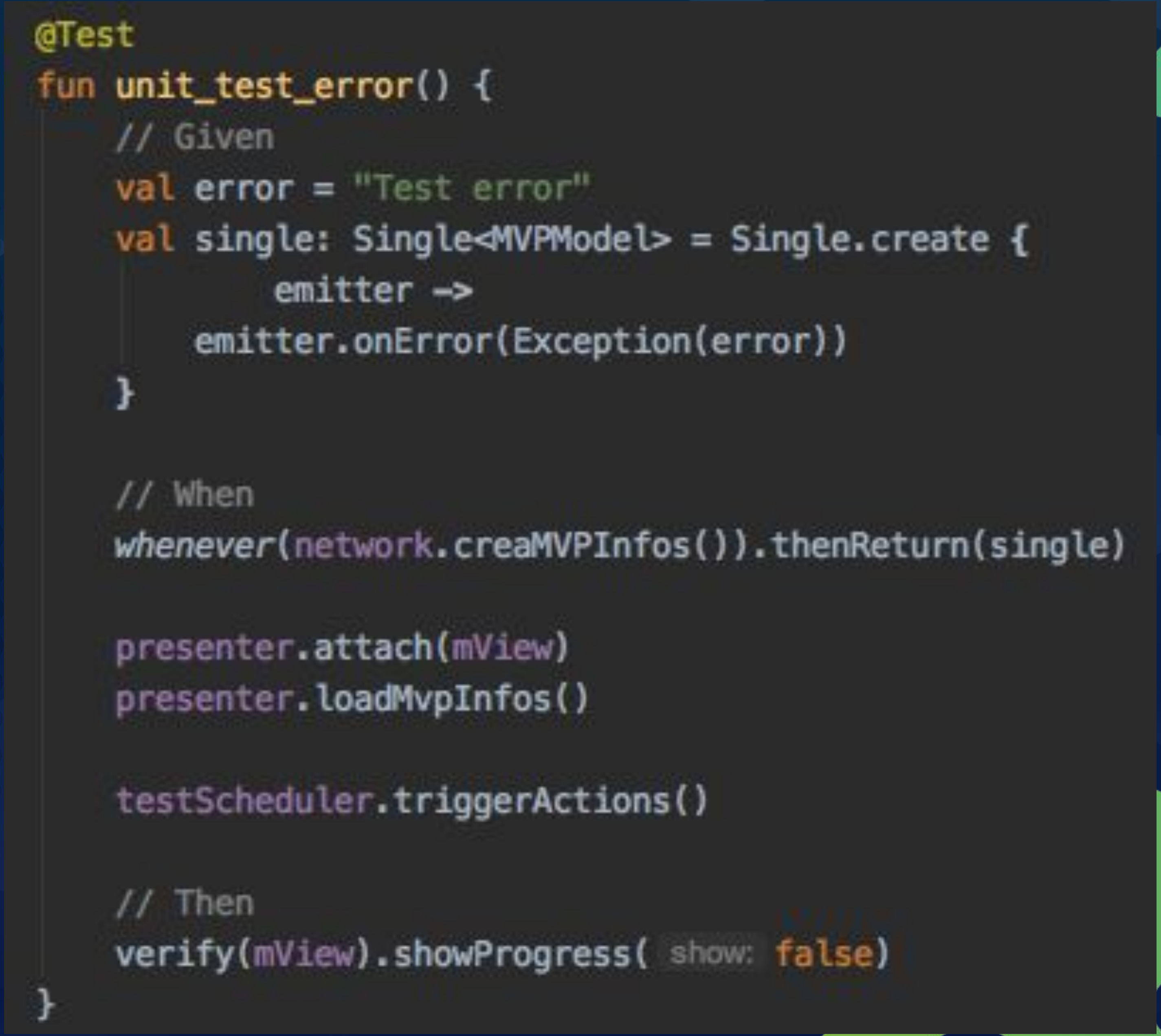
```
    override fun onLoadedInfosMvp(mvpModel: MVPModel) {
        btnMoreInfos.text = "Para mais informacoes entre em:\n\n${mvpModel.url}"

        btnMoreInfos.setOnClickListener { it: View!
            val url = mvpModel.url
            val i = Intent(Intent.ACTION_VIEW)
            i.data = Uri.parse(url)
            startActivity(i)
        }

        btnMoreInfos.visibility = VISIBLE
        imgMvp.visibility = VISIBLE
        mvp.visibility = VISIBLE
    }
}
```



```
class LoginPresenterTest {

    private val mView = mock(Presenter.View::class.java)
    private val testScheduler = TestScheduler()
    private val network: NetworkFake = mock()
    private val schedulerProvider =
        TestSchedulerProvider(
            testScheduler
        )
    private val presenter =
        PresenterImp(network, schedulerProvider)

    @Before
    fun setup() {
        MockitoAnnotations.initMocks( testClass: this)
        presenter.attach(mView)
    }
```



```
@Test
fun unit_test_success() {
    // Given
    val mvpModel = MVPModel(url = "https://pt.wikipedia.org/wiki/Model-view-presenter")

    val single: Single<MVPModel> = Single.create {
            emitter ->
        emitter.onSuccess(mvpModel)
    }

    // When
    whenever(network.creaMVPInfos()).thenReturn(single)

    presenter.attach(mView)
    presenter.loadMvpInfos()
    verify(network).creaMVPInfos()

    testScheduler.triggerActions()

    // Then
    verify(mView).showProgress( show: false)
    verify(mView).onLoadedInfosMvp(mvpModel)
}
```



```
@Test
fun unit_test_error() {
    // Given
    val error = "Test error"
    val single: Single<MVPModel> = Single.create {
            emitter ->
        emitter.onError(Exception(error))
    }

    // When
    whenever(network.creaMVPInfos()).thenReturn(single)

    presenter.attach(mView)
    presenter.loadMvpInfos()

    testScheduler.triggerActions()

    // Then
    verify(mView).showProgress( show: false)
}
```

Model View Presenter

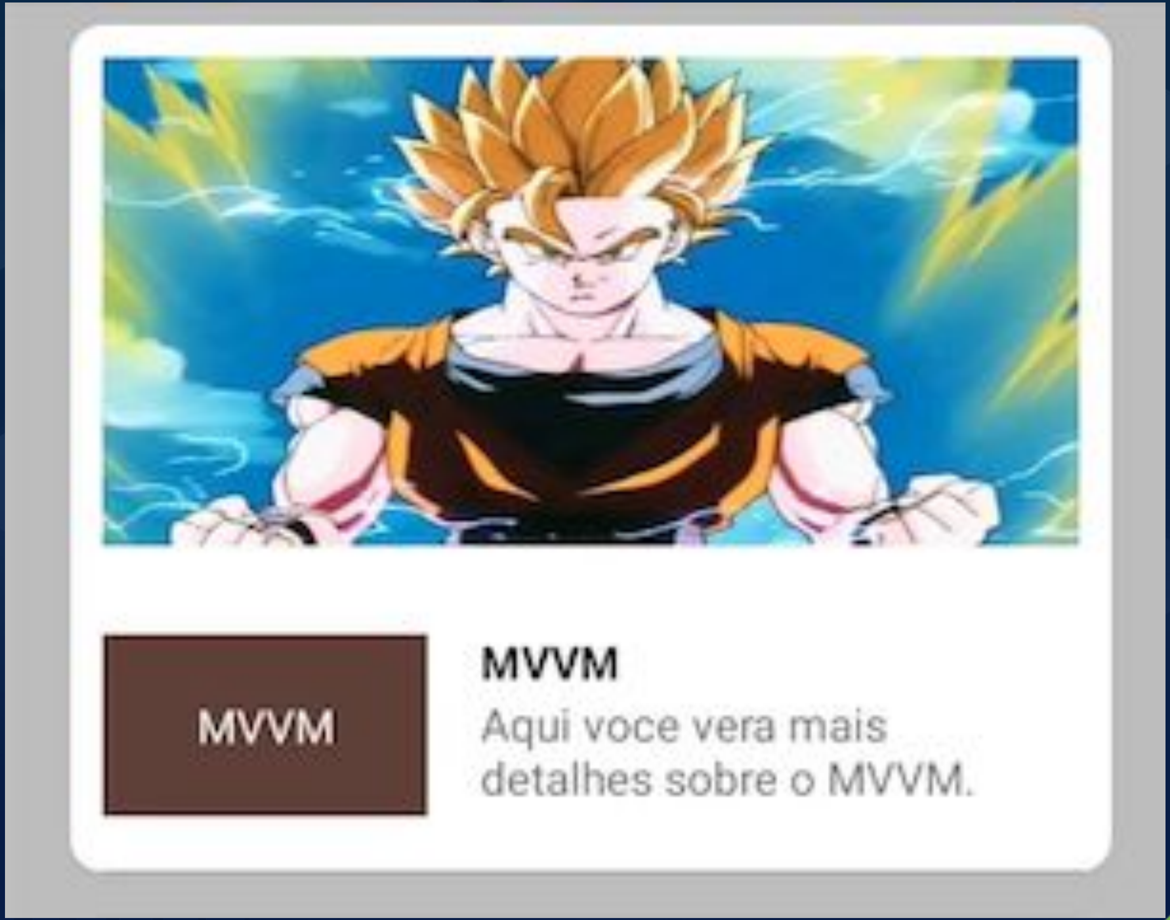
- 1 - Activity /* - View ✓
- 2 - Presenter - Contrato / RxAndroid ✓
- 3 - Model - MVPModel ✓

Vantagens: Fácil de testar, fácil de debugar, Mais clara a manutenção pois tem testes.

Desvantagens: Aumento de classes, repetição de métodos sendo para cada tela um cenário específico.

Arquiteturas em Projetos Android

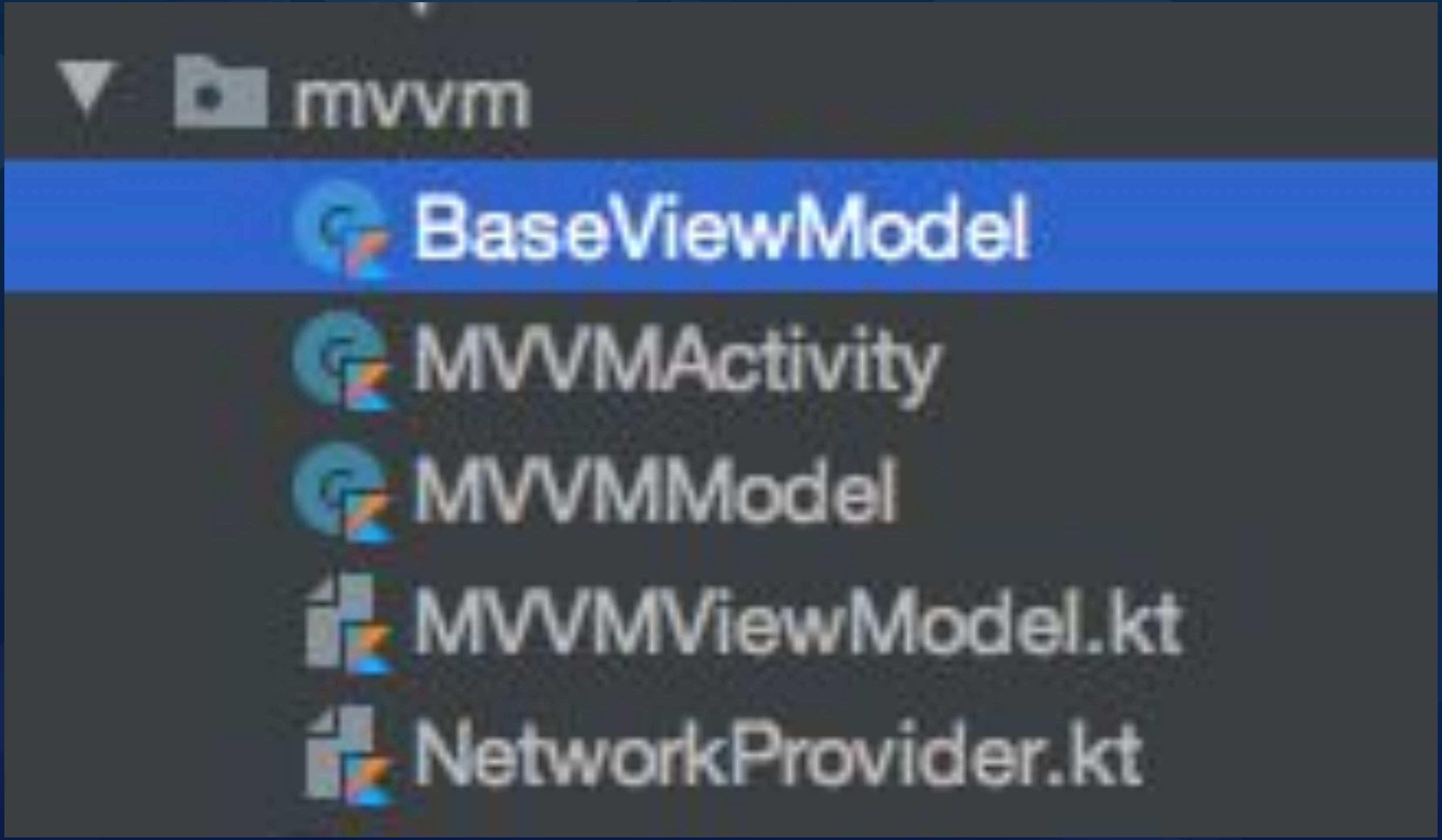
- 1 - MVC ✓
- 2 - MVP ✓
- 3 - MVVM
- 4 - MVP Clean
- 5 - MVVM Clean
- 6 - MVI


MVVM

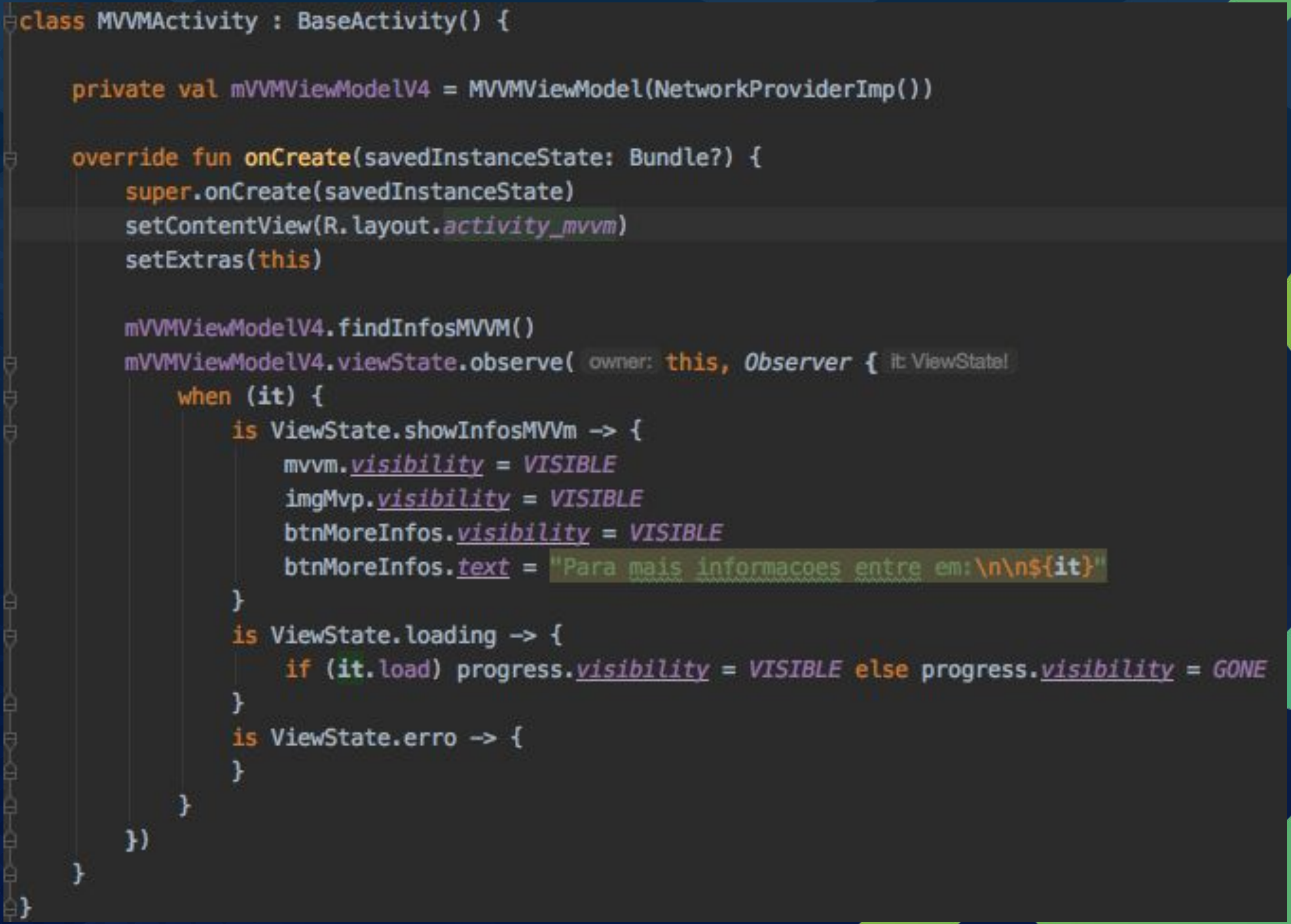
MVVM

Aqui voce vera mais detalhes sobre o MVVM.

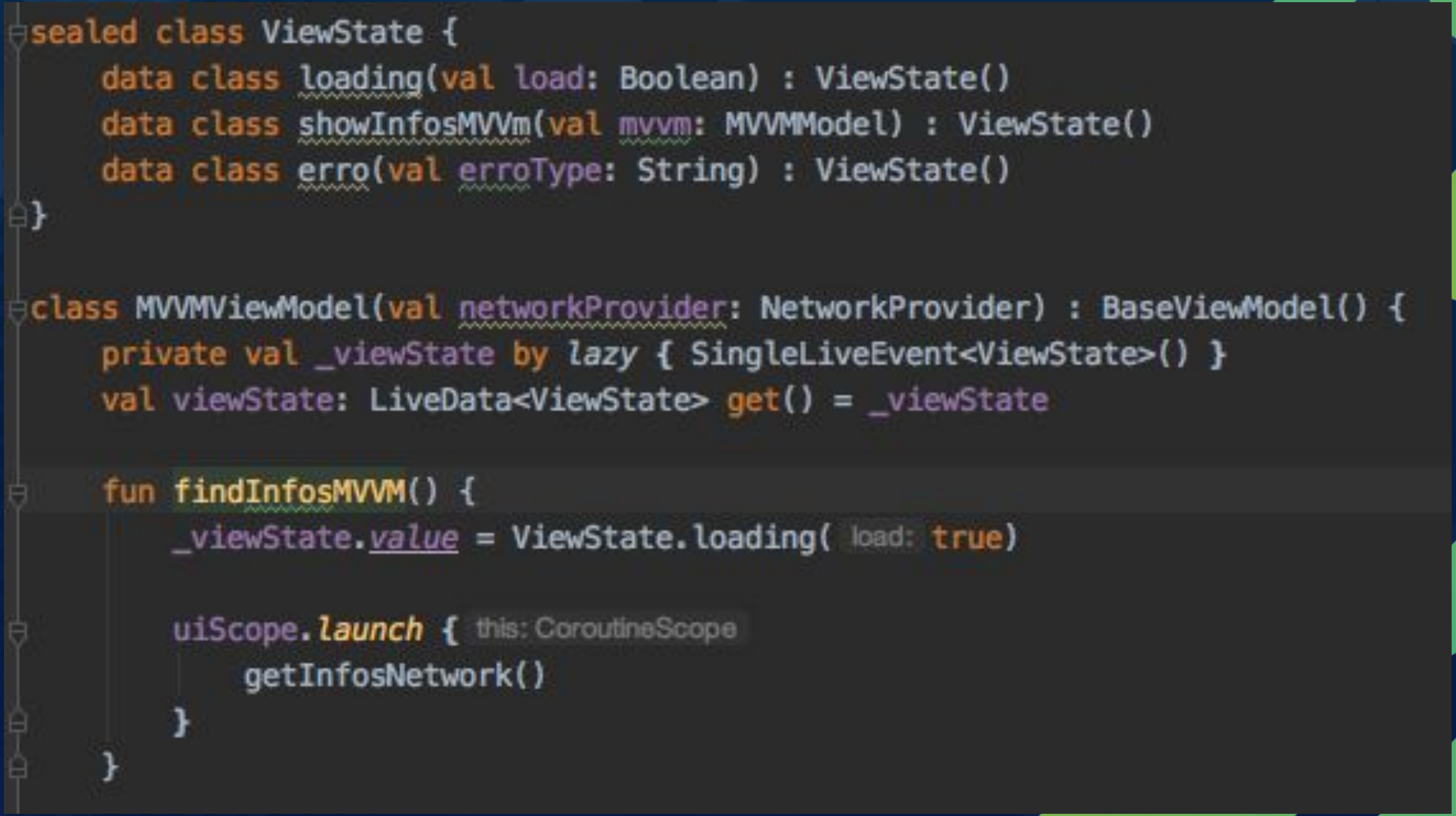
https://github.com/nicconicco/Arch-DBZ/tree/master/app/src/main/java/com/nicco/architectures/android/mvvm



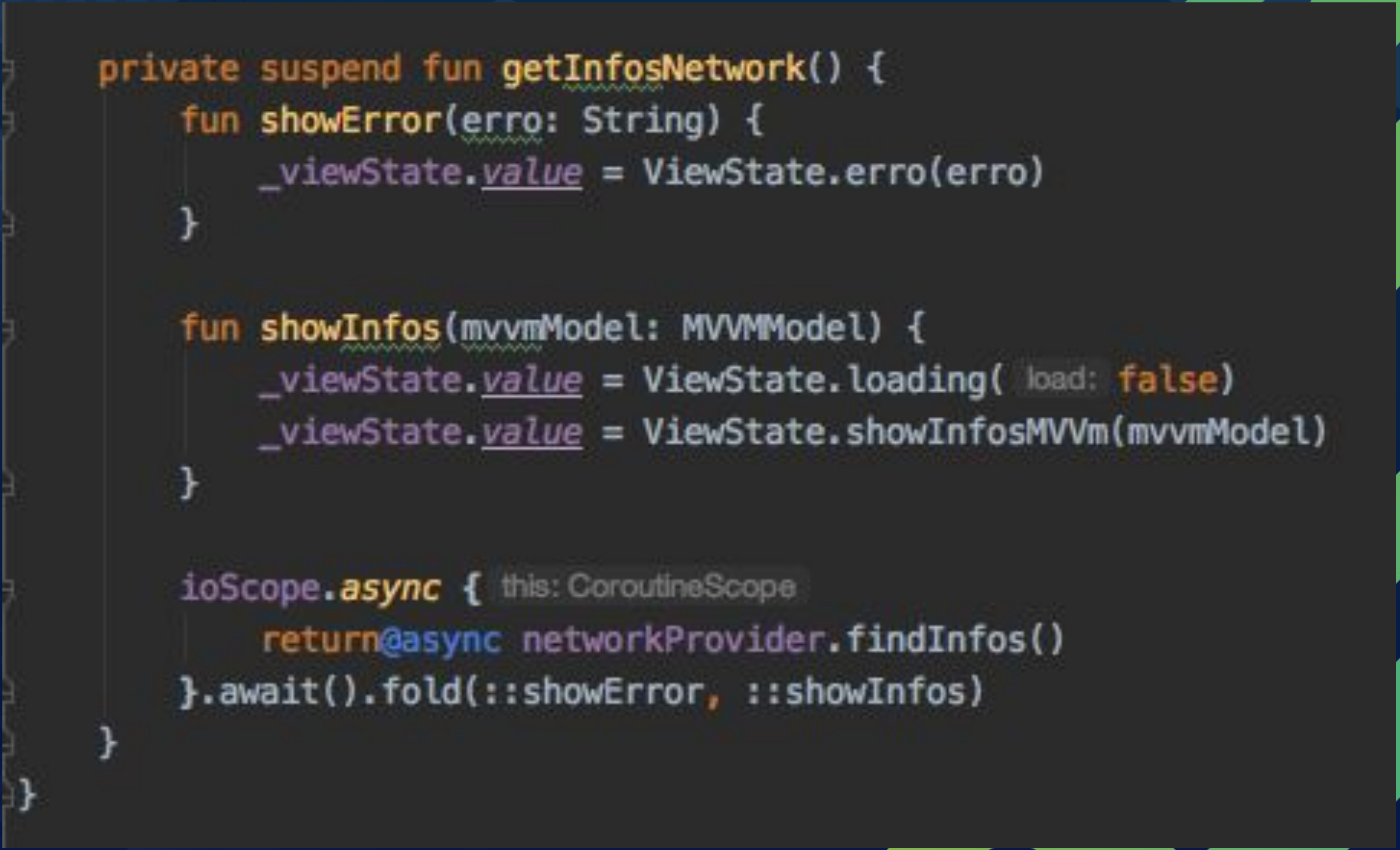
mvvm
BaseViewModel
MVVMActivity
MVVMModel
MVVMViewModel.kt
NetworkProvider.kt

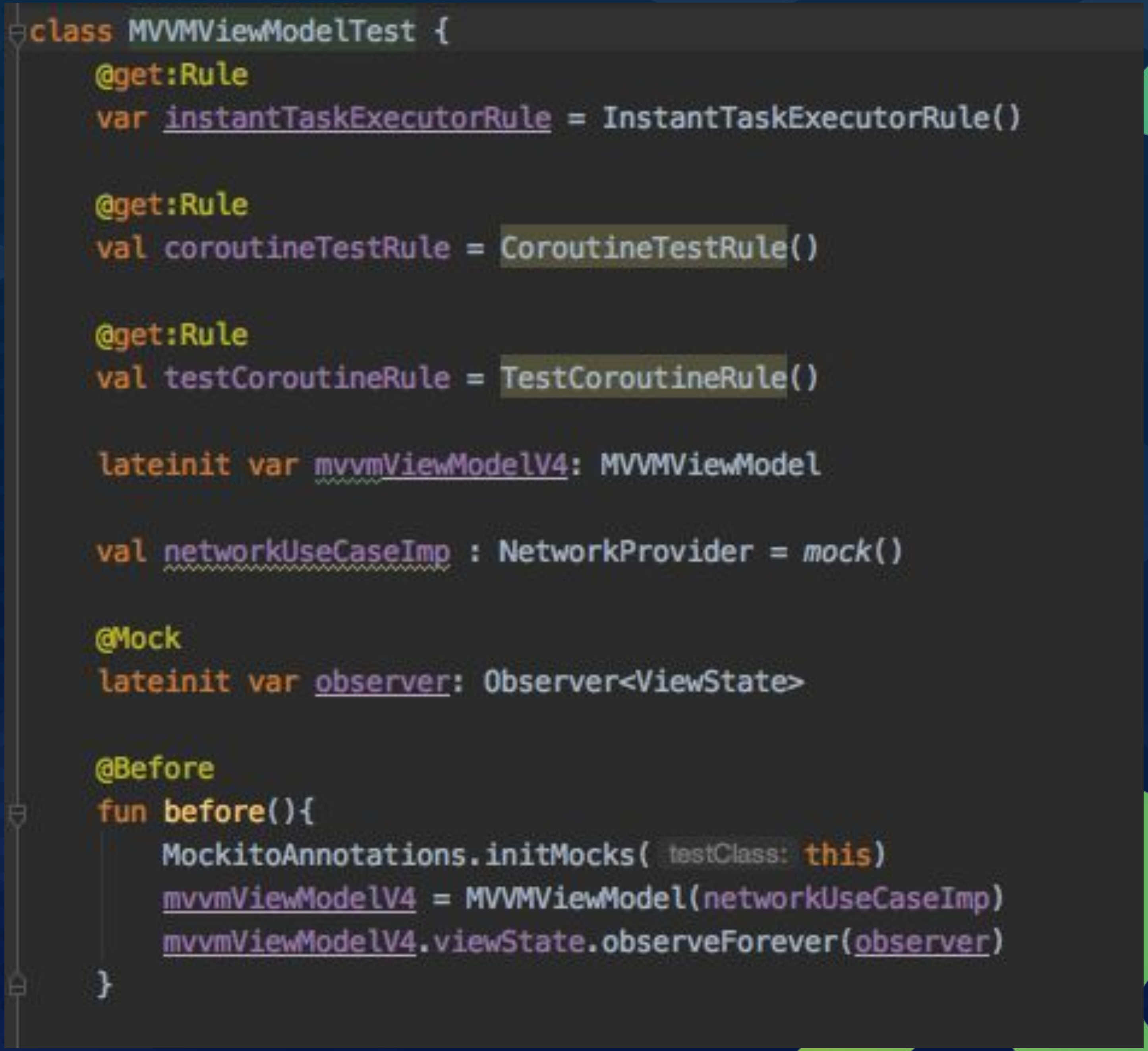


```
class MVVMActivity : BaseActivity() {

    private val mVVMViewModelV4 = MVVMViewModel(NetworkProviderImp())

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mvvm)
        setExtras(this)

        mVVMViewModelV4.findInfosMVVM()
        mVVMViewModelV4.viewState.observe( owner: this, Observer { it: ViewState!
            when (it) {
                is ViewState.showInfosMVVm -> {
                    mvvm.visibility = VISIBLE
                    imgMvp.visibility = VISIBLE
                    btnMoreInfos.visibility = VISIBLE
                    btnMoreInfos.text = "Para mais informacoes entre em:\n\n${it}"
                }
                is ViewState.loading -> {
                    if (it.load) progress.visibility = VISIBLE else progress.visibility = GONE
                }
                is ViewState.erro -> {
                }
            }
        })
    }
}
```

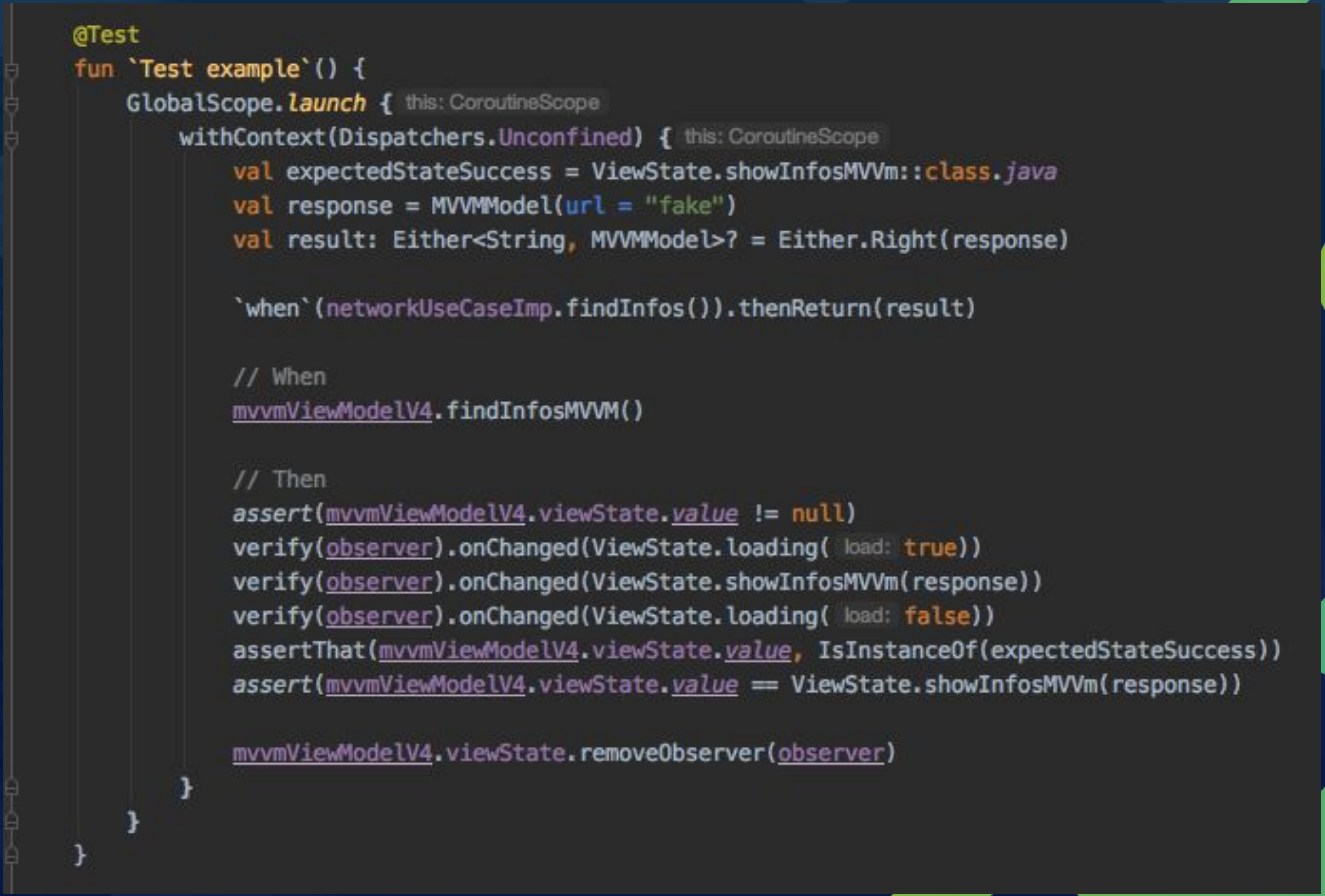


```kotlin
sealed class ViewState {
    data class loading(val load: Boolean) : ViewState()
    data class showInfosMVVm(val mvvm: MVVMModel) : ViewState()
    data class erro(val erroType: String) : ViewState()
}

class MVVMViewModel(val networkProvider: NetworkProvider) : BaseViewModel() {
    private val _viewState by lazy { SingleLiveEvent<ViewState>() }
    val viewState: LiveData<ViewState> get() = _viewState

    fun findInfosMVVM() {
        _viewState.value = ViewState.loading( load: true)

        uiScope.launch { this: CoroutineScope
            getInfosNetwork()
        }
    }
```


```
    private suspend fun getInfosNetwork() {
        fun showError(erro: String) {
            _viewState.value = ViewState.erro(erro)
        }

        fun showInfos(mvvmModel: MVVMModel) {
            _viewState.value = ViewState.loading( load: false)
            _viewState.value = ViewState.showInfosMVVm(mvvmModel)
        }

        ioScope.async { this: CoroutineScope
            return@async networkProvider.findInfos()
        }.await().fold(::showError, ::showInfos)
    }
}
```



```
class MVVMViewModelTest {
    @get:Rule
    var instantTaskExecutorRule = InstantTaskExecutorRule()

    @get:Rule
    val coroutineTestRule = CoroutineTestRule()

    @get:Rule
    val testCoroutineRule = TestCoroutineRule()

    lateinit var mvvmViewModelV4: MVVMViewModel

    val networkUseCaseImp : NetworkProvider = mock()

    @Mock
    lateinit var observer: Observer<ViewState>

    @Before
    fun before(){
        MockitoAnnotations.initMocks( testClass: this)
        mvvmViewModelV4 = MVVMViewModel(networkUseCaseImp)
        mvvmViewModelV4.viewState.observeForever(observer)
    }
```



```kotlin
@Test
fun `Test example`() {
    GlobalScope.launch { this: CoroutineScope
        withContext(Dispatchers.Unconfined) { this: CoroutineScope
            val expectedStateSuccess = ViewState.showInfosMVVm::class.java
            val response = MVVMModel(url = "fake")
            val result: Either<String, MVVMModel>? = Either.Right(response)

            `when`(networkUseCaseImp.findInfos()).thenReturn(result)

            // When
            mvvmViewModelV4.findInfosMVVM()

            // Then
            assert(mvvmViewModelV4.viewState.value != null)
            verify(observer).onChanged(ViewState.loading( load: true))
            verify(observer).onChanged(ViewState.showInfosMVVm(response))
            verify(observer).onChanged(ViewState.loading( load: false))
            assertThat(mvvmViewModelV4.viewState.value, IsInstanceOf(expectedStateSuccess))
            assert(mvvmViewModelV4.viewState.value == ViewState.showInfosMVVm(response))

            mvvmViewModelV4.viewState.removeObserver(observer)
        }
    }
}
```

Vantagens: Reduz o código e dá pra testar, questão de rotacionar a tela, controle de estados.

Desvantagens: Programação reativa (Não tão clara para inciantes) Os testes foram um pouco mais difícil.

Model View ViewModel

- 1 - Activity /* - View ✓
- 2 - ViewModel - Coroutines / States / Livedata ✓
- 3 - Model - MVVMModel ✓

Arquiteturas em Projetos Android

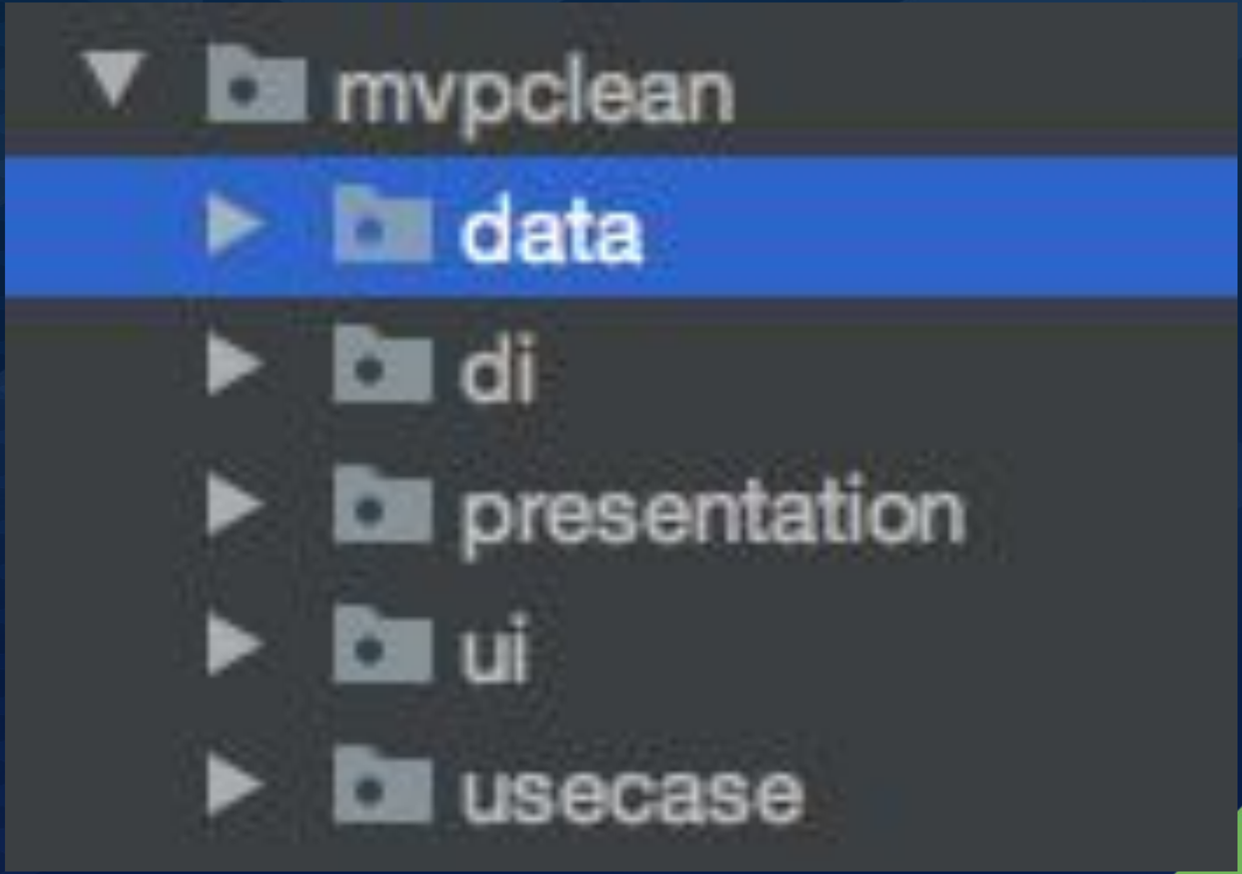
- 1 - MVC ✓
- 2 - MVP ✓
- 3 - MVVM ✓
- 4 - MVP Clean
- 5 - MVVM Clean
- 6 - MVI

MVP Clean

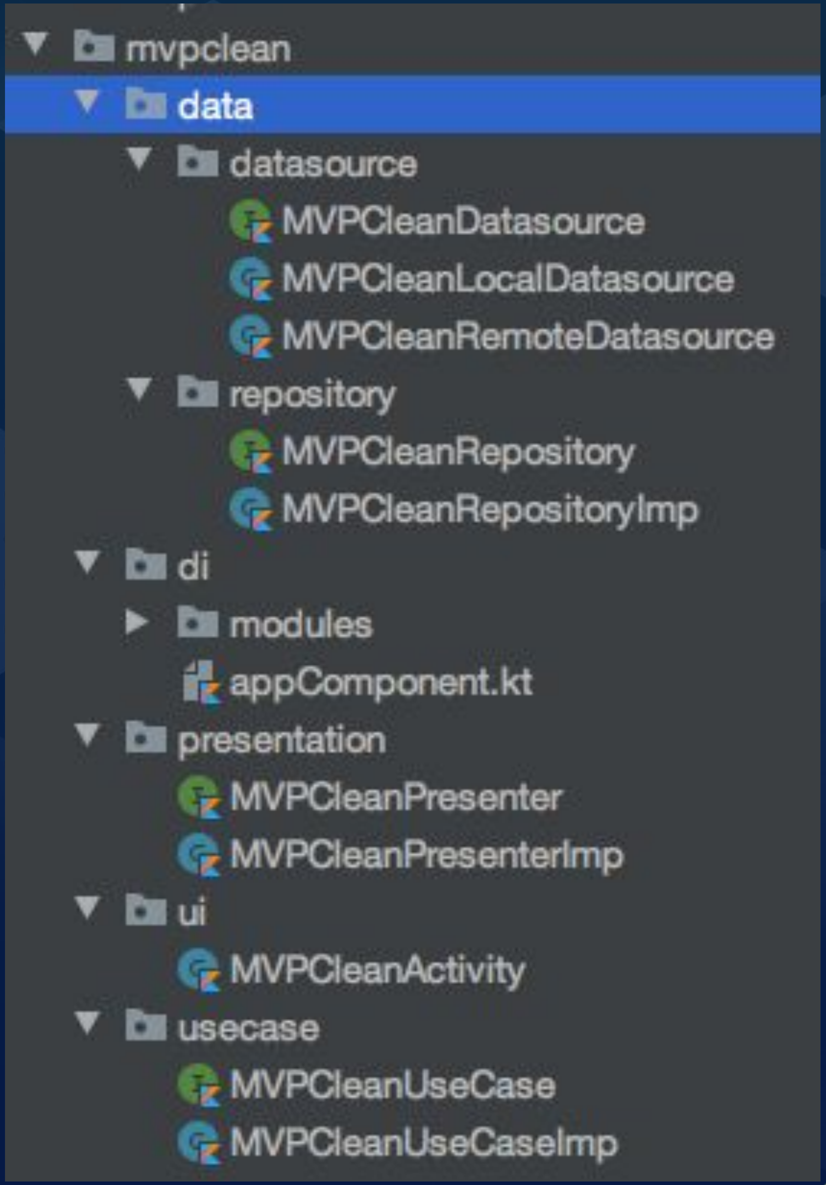
MVP Clean

Aqui voce vera mais detalhes sobre o MVI.

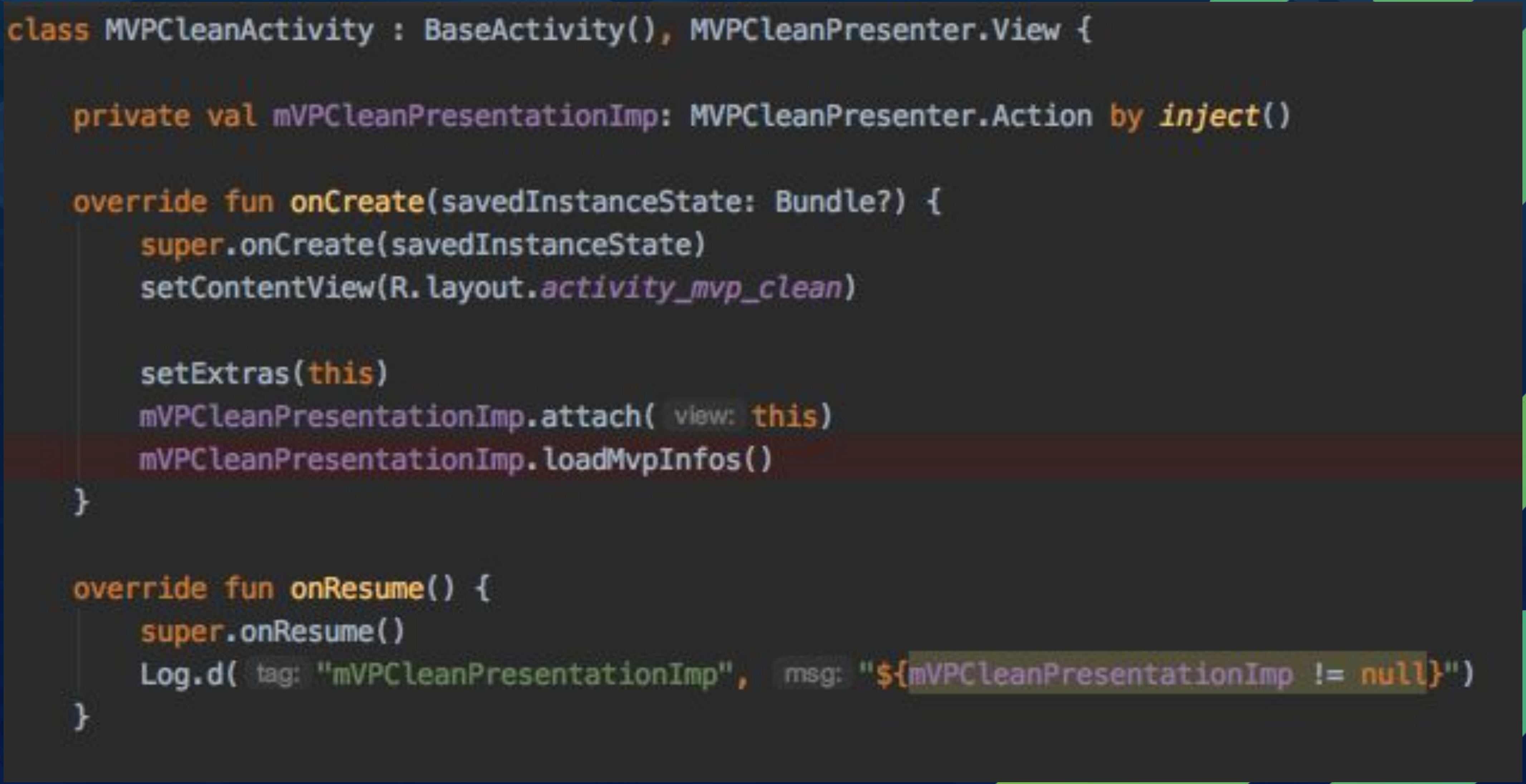
https://github.com/nicconicco/Arch-DBZ/tree/master/app/src/main/java/com/nicco/architectures/android/mvpclean



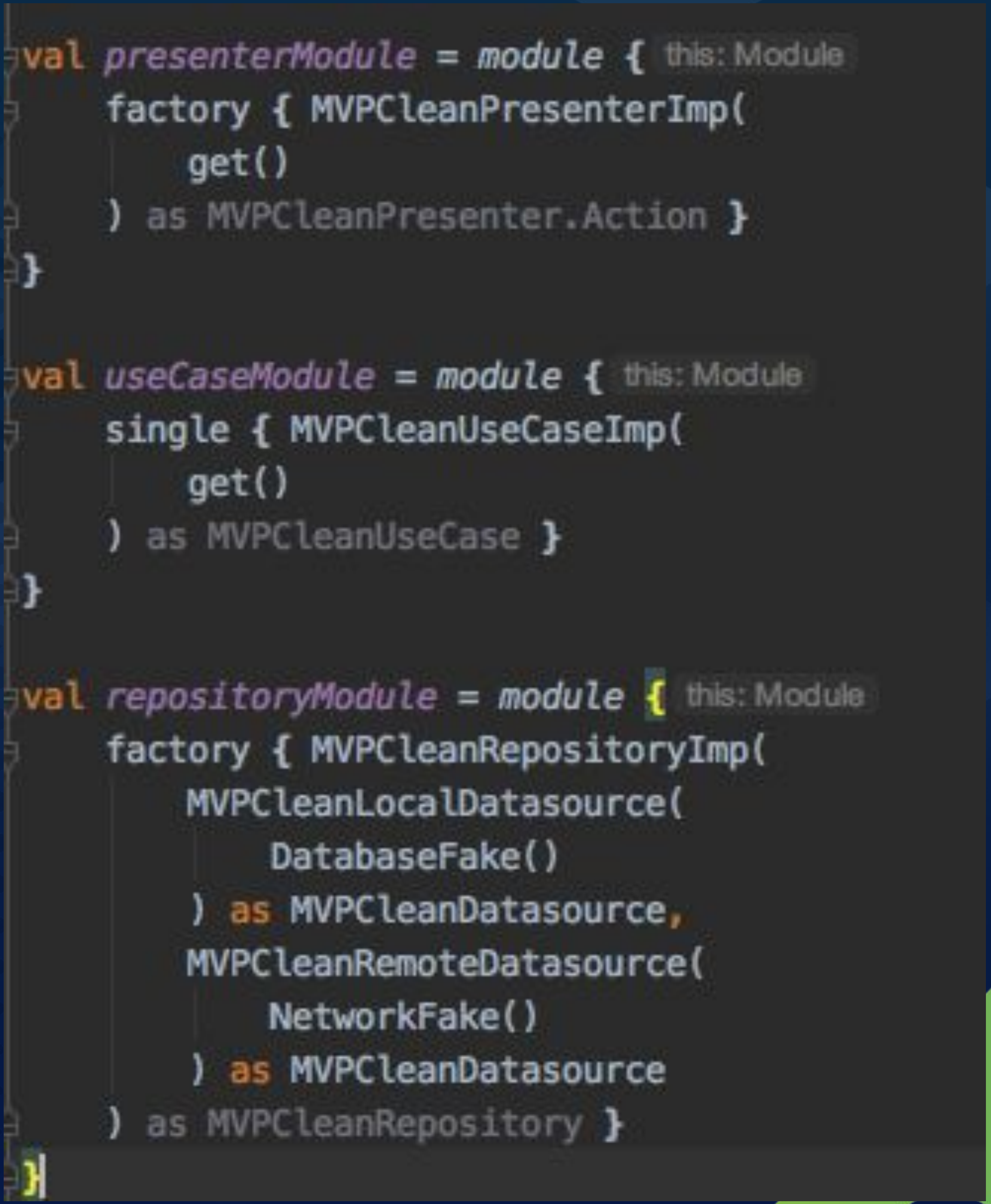
```
▼ mvpclean
    ▶ data
    ▶ di
    ▶ presentation
    ▶ ui
    ▶ usecase
```

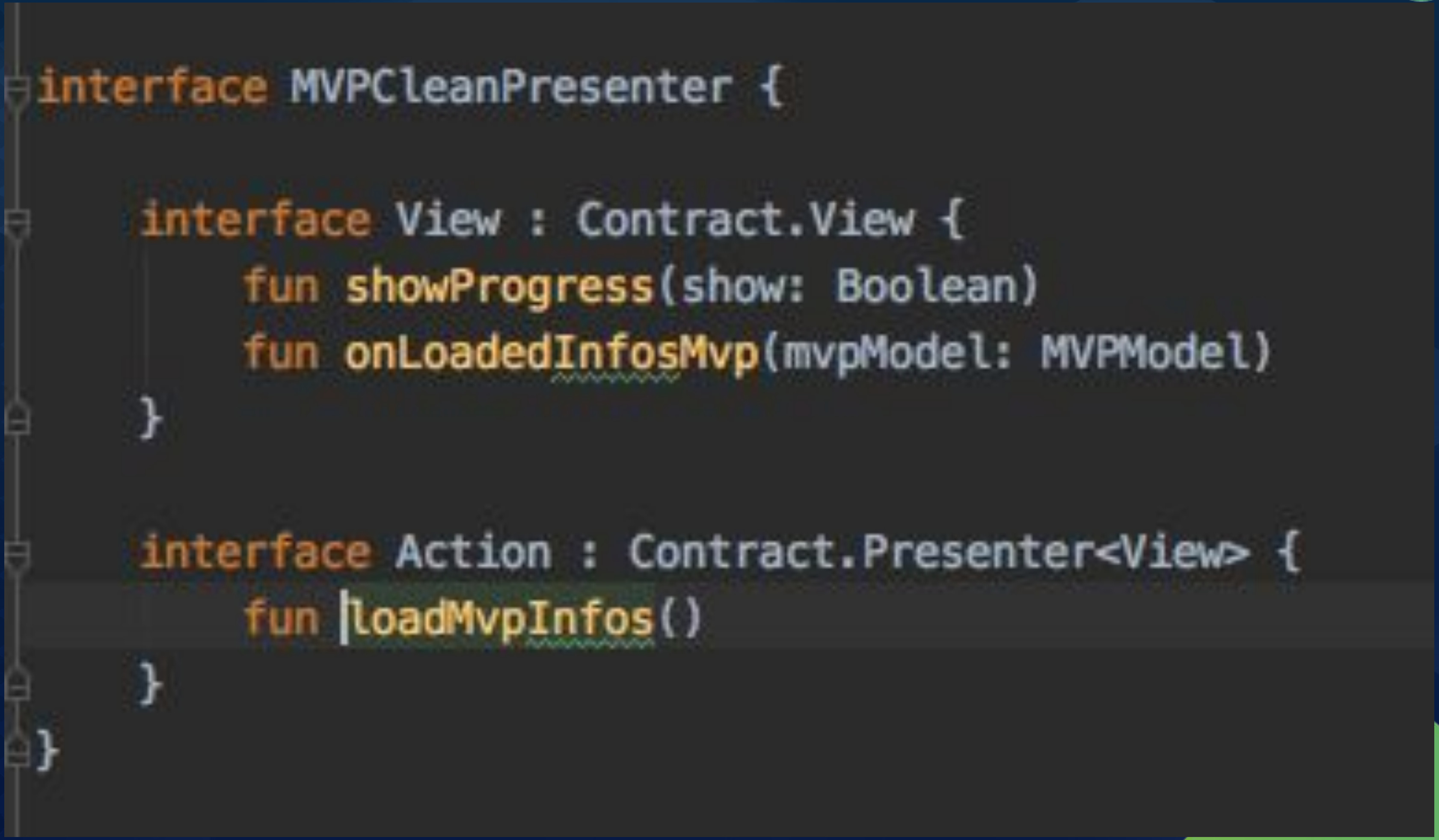


```
▼ mvpclean
  ▼ data
    ▼ datasource
        MVPCleanDatasource
        MVPCleanLocalDatasource
        MVPCleanRemoteDatasource
    ▼ repository
        MVPCleanRepository
        MVPCleanRepositoryImp
  ▼ di
    ▶ modules
      appComponent.kt
  ▼ presentation
      MVPCleanPresenter
      MVPCleanPresenterImp
  ▼ ui
      MVPCleanActivity
  ▼ usecase
      MVPCleanUseCase
      MVPCleanUseCaseImp
```

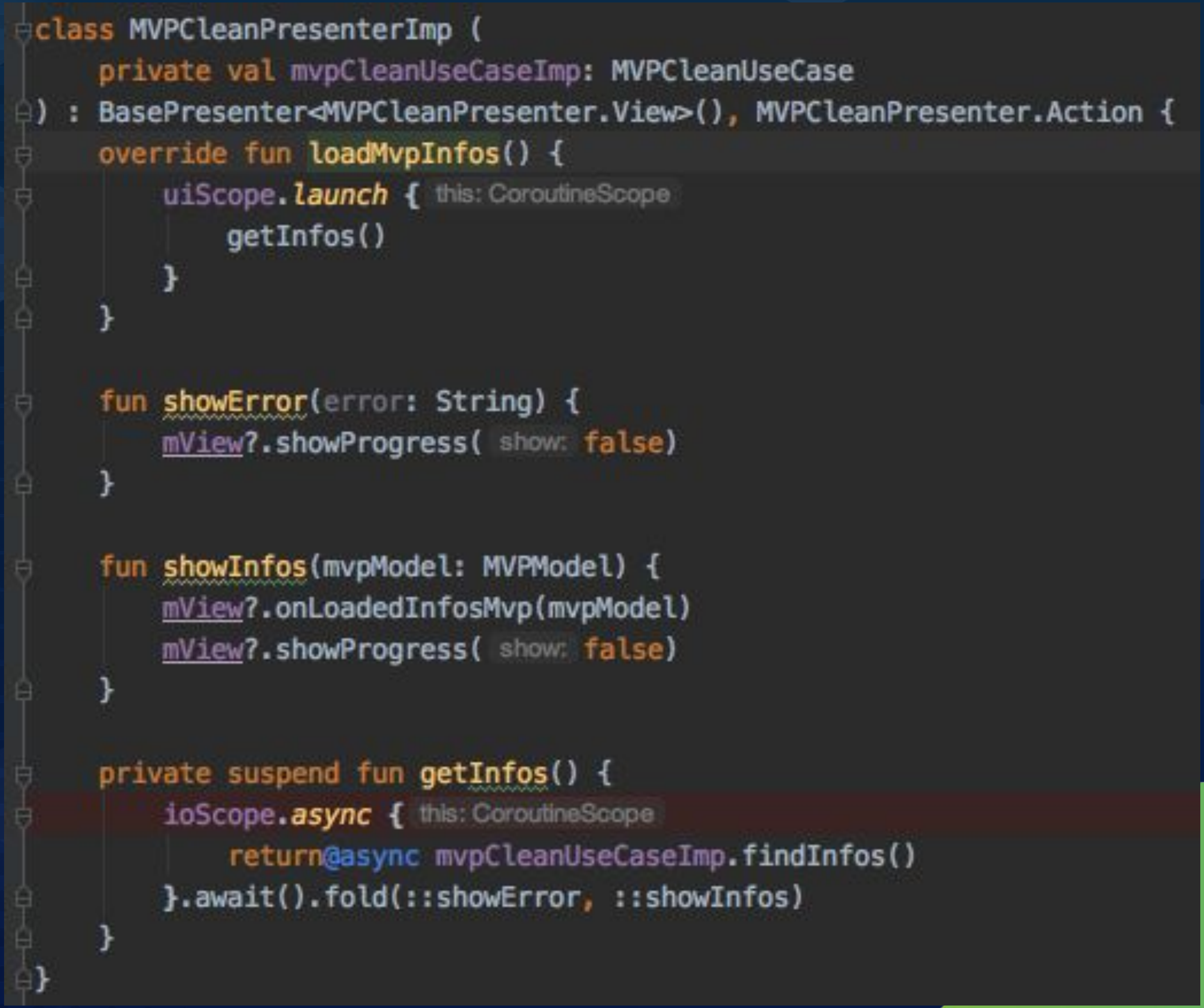


```
class MVPCleanActivity : BaseActivity(), MVPCleanPresenter.View {

    private val mVPCleanPresentationImp: MVPCleanPresenter.Action by inject()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mvp_clean)

        setExtras(this)
        mVPCleanPresentationImp.attach( view: this)
        mVPCleanPresentationImp.loadMvpInfos()
    }

    override fun onResume() {
        super.onResume()
        Log.d( tag: "mVPCleanPresentationImp", msg: "${mVPCleanPresentationImp != null}")
    }
```

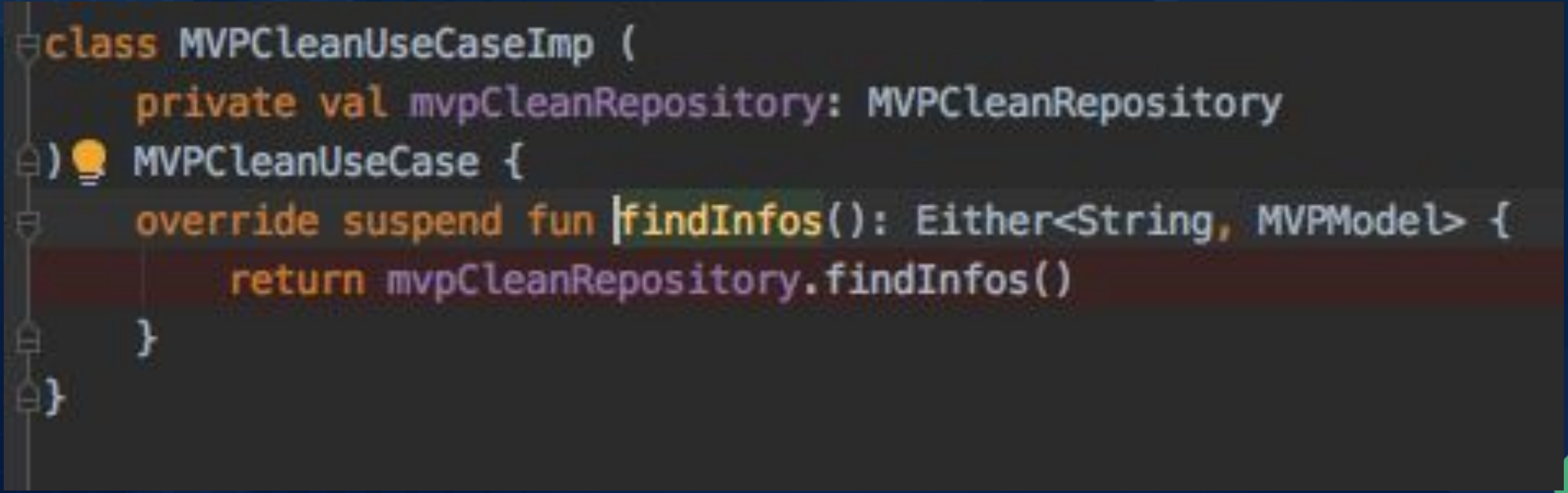

```
val presenterModule = module { this: Module
    factory { MVPCleanPresenterImp(
        get()
    ) as MVPCleanPresenter.Action }
}

val useCaseModule = module { this: Module
    single { MVPCleanUseCaseImp(
        get()
    ) as MVPCleanUseCase }
}

val repositoryModule = module { this: Module
    factory { MVPCleanRepositoryImp(
        MVPCleanLocalDatasource(
            DatabaseFake()
        ) as MVPCleanDatasource,
        MVPCleanRemoteDatasource(
            NetworkFake()
        ) as MVPCleanDatasource
    ) as MVPCleanRepository }
}
```



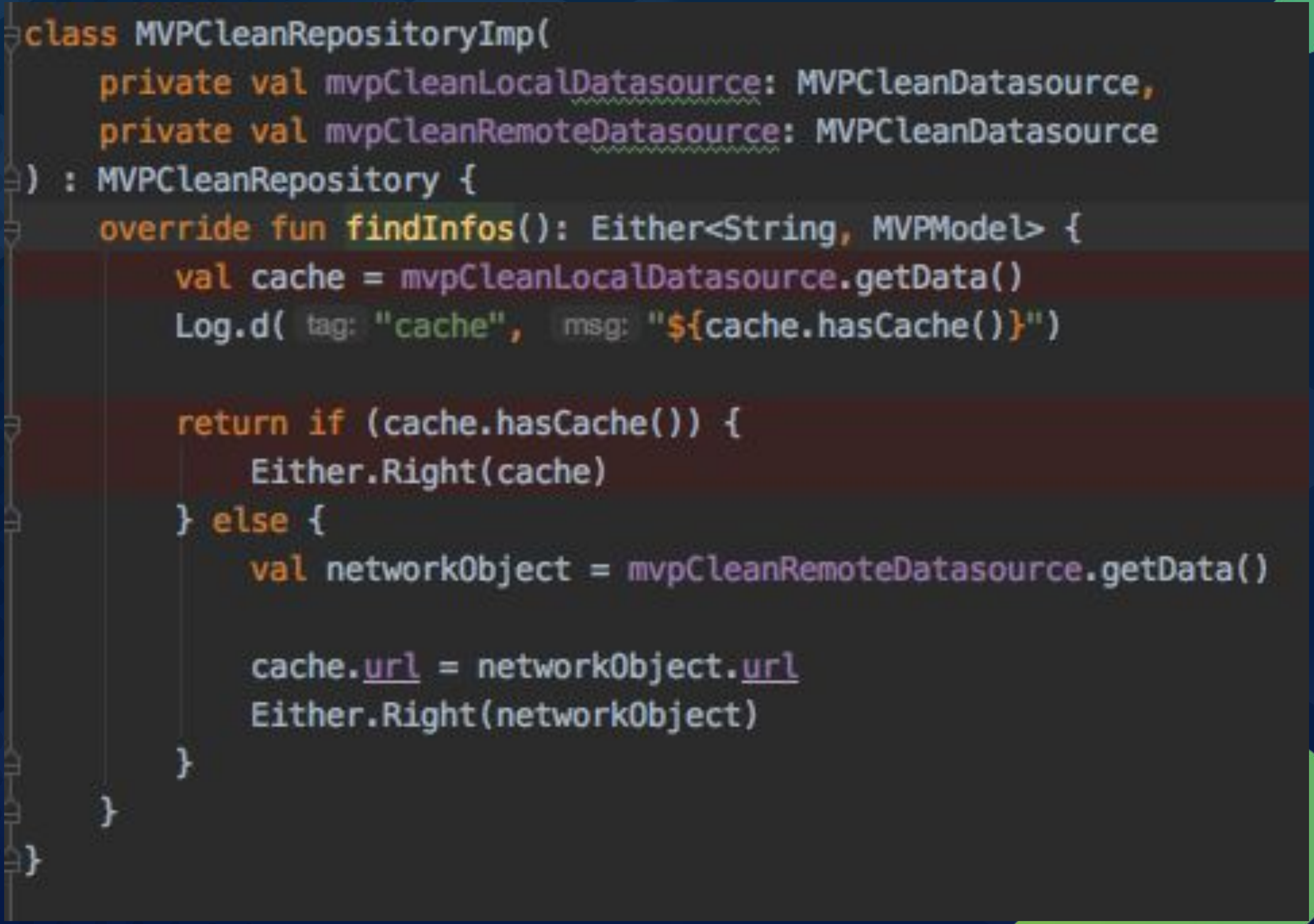
```kotlin
interface MVPCleanPresenter {

    interface View : Contract.View {
        fun showProgress(show: Boolean)
        fun onLoadedInfosMvp(mvpModel: MVPModel)
    }

    interface Action : Contract.Presenter<View> {
        fun loadMvpInfos()
    }
}
```



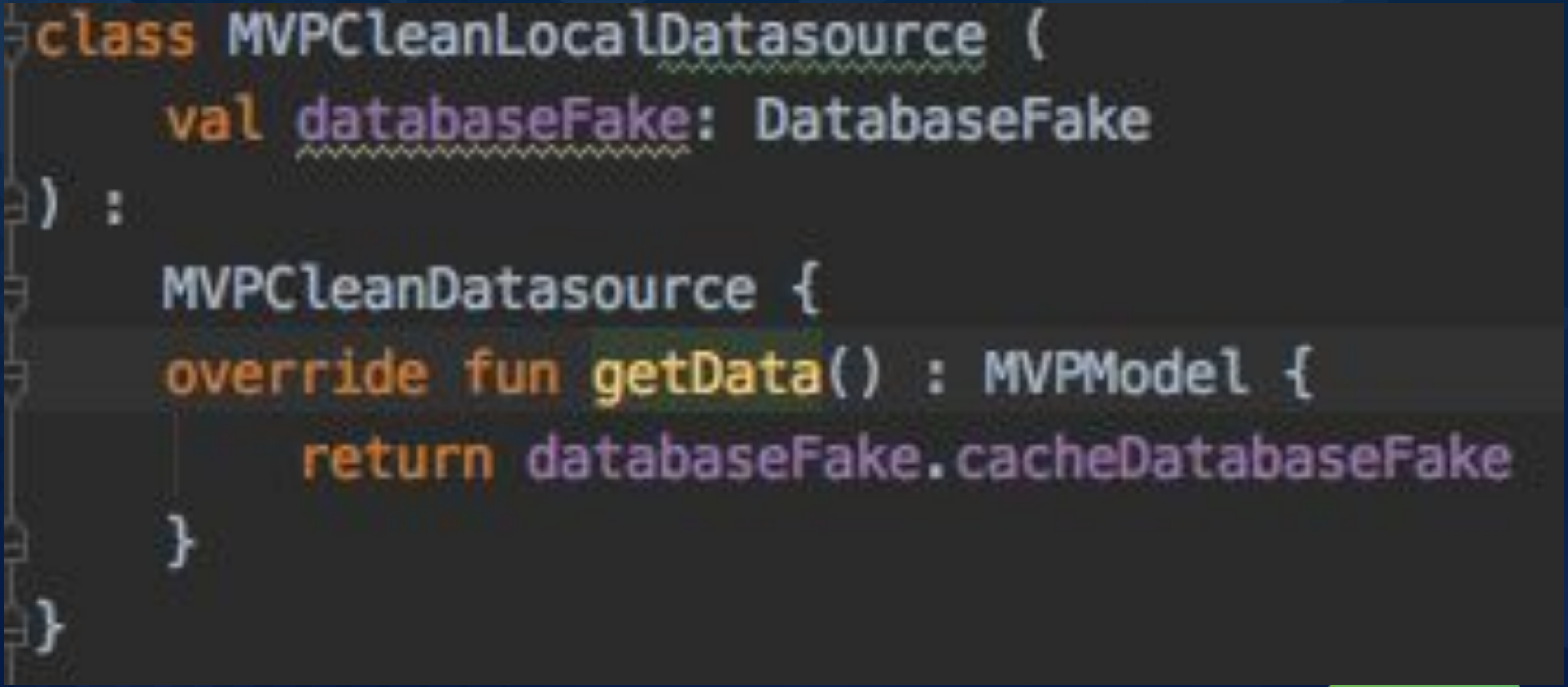
```kotlin
class MVPCleanPresenterImp (
    private val mvpCleanUseCaseImp: MVPCleanUseCase
) : BasePresenter<MVPCleanPresenter.View>(), MVPCleanPresenter.Action {
    override fun loadMvpInfos() {
        uiScope.launch { this: CoroutineScope
            getInfos()
        }
    }

    fun showError(error: String) {
        mView?.showProgress( show: false)
    }

    fun showInfos(mvpModel: MVPModel) {
        mView?.onLoadedInfosMvp(mvpModel)
        mView?.showProgress( show: false)
    }

    private suspend fun getInfos() {
        ioScope.async { this: CoroutineScope
            return@async mvpCleanUseCaseImp.findInfos()
        }.await().fold(::showError, ::showInfos)
    }
}
```



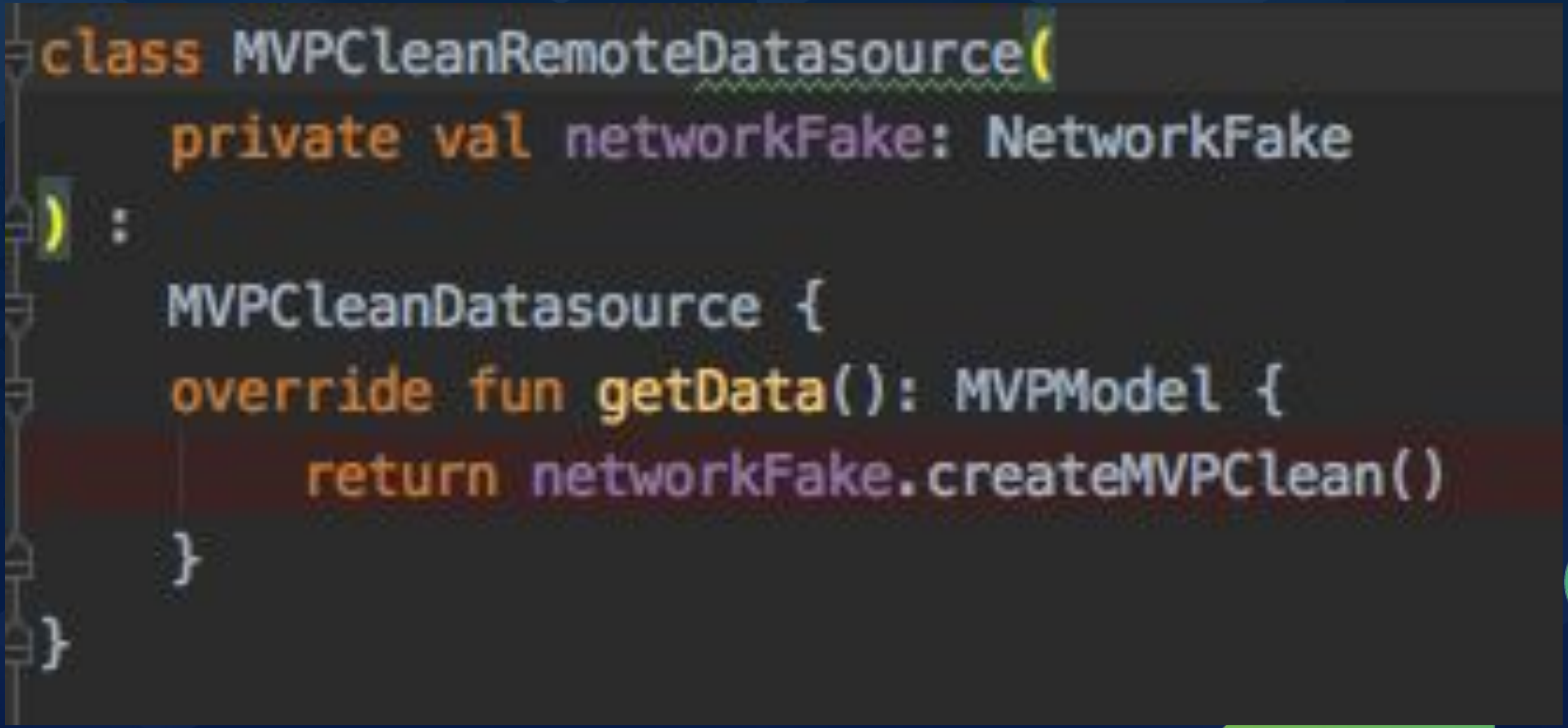
```
class MVPCleanUseCaseImp (
    private val mvpCleanRepository: MVPCleanRepository
) MVPCleanUseCase {
    override suspend fun findInfos(): Either<String, MVPModel> {
        return mvpCleanRepository.findInfos()
    }
}
```



```
class MVPCleanRepositoryImp(
    private val mvpCleanLocalDatasource: MVPCleanDatasource,
    private val mvpCleanRemoteDatasource: MVPCleanDatasource
) : MVPCleanRepository {
    override fun findInfos(): Either<String, MVPModel> {
        val cache = mvpCleanLocalDatasource.getData()
        Log.d( tag: "cache",  msg: "${cache.hasCache()}")

        return if (cache.hasCache()) {
            Either.Right(cache)
        } else {
            val networkObject = mvpCleanRemoteDatasource.getData()

            cache.url = networkObject.url
            Either.Right(networkObject)
        }
    }
}
```



```
class MVPCleanLocalDatasource (
    val databaseFake: DatabaseFake
) :
    MVPCleanDatasource {
    override fun getData() : MVPModel {
        return databaseFake.cacheDatabaseFake
    }
}
```



```
class MVPCleanRemoteDatasource(
    private val networkFake: NetworkFake
) :
    MVPCleanDatasource {
    override fun getData(): MVPModel {
        return networkFake.createMVPClean()
    }
}
```

Vantagens: Projetos grandes, Fácil de mudar os framework que vão se atualizando, testável, SOLID.

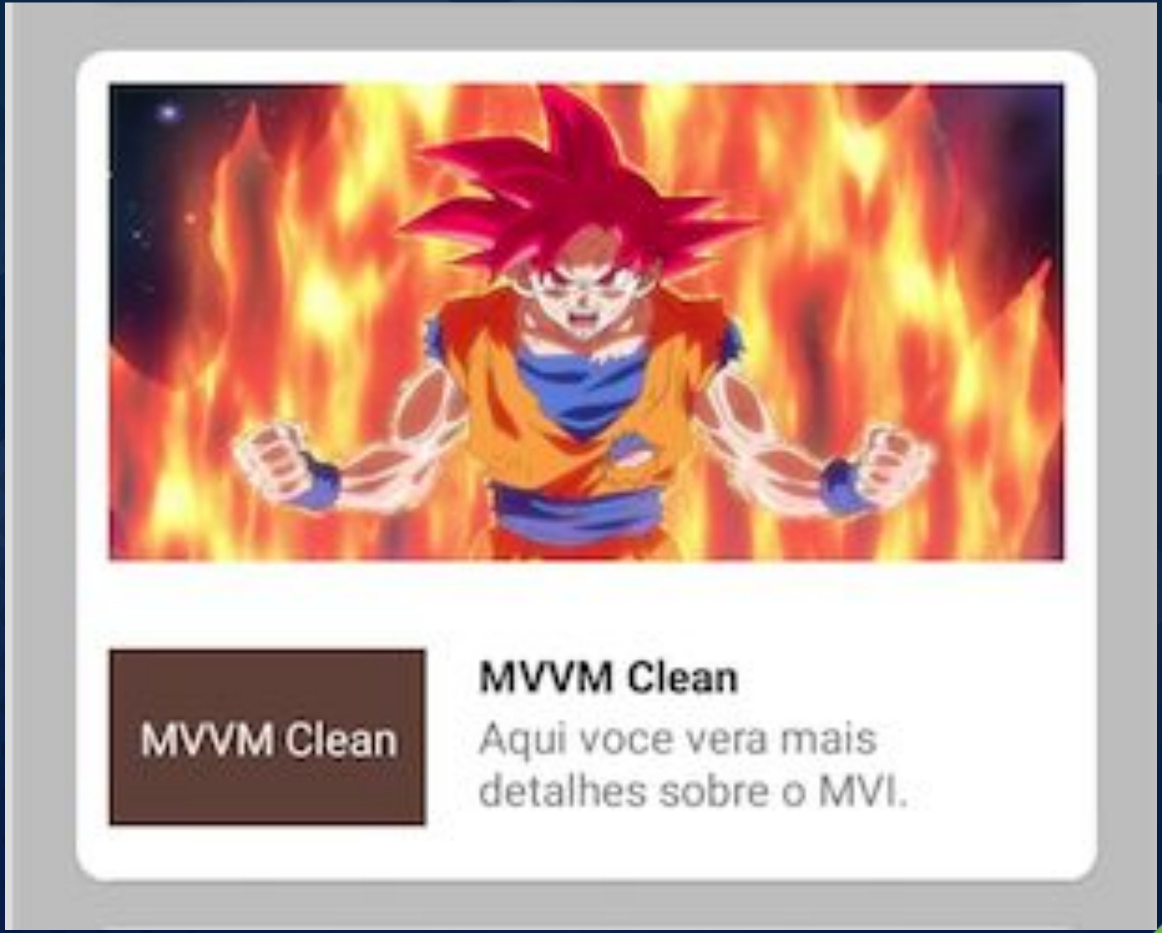
Desvantagens: Código exige um programador mais cuidadoso (Responsabilidades de camadas, onde vai o que?)

Model View Presenter Clean

- 1 - Activity /* - ui ✓
- 2 - Presenter - Coroutines / Either ✓
- 3 - UseCase / Repository / Datasource ✓
- 4 - Dependency Injection: Koin ✓

Arquiteturas em Projetos Android

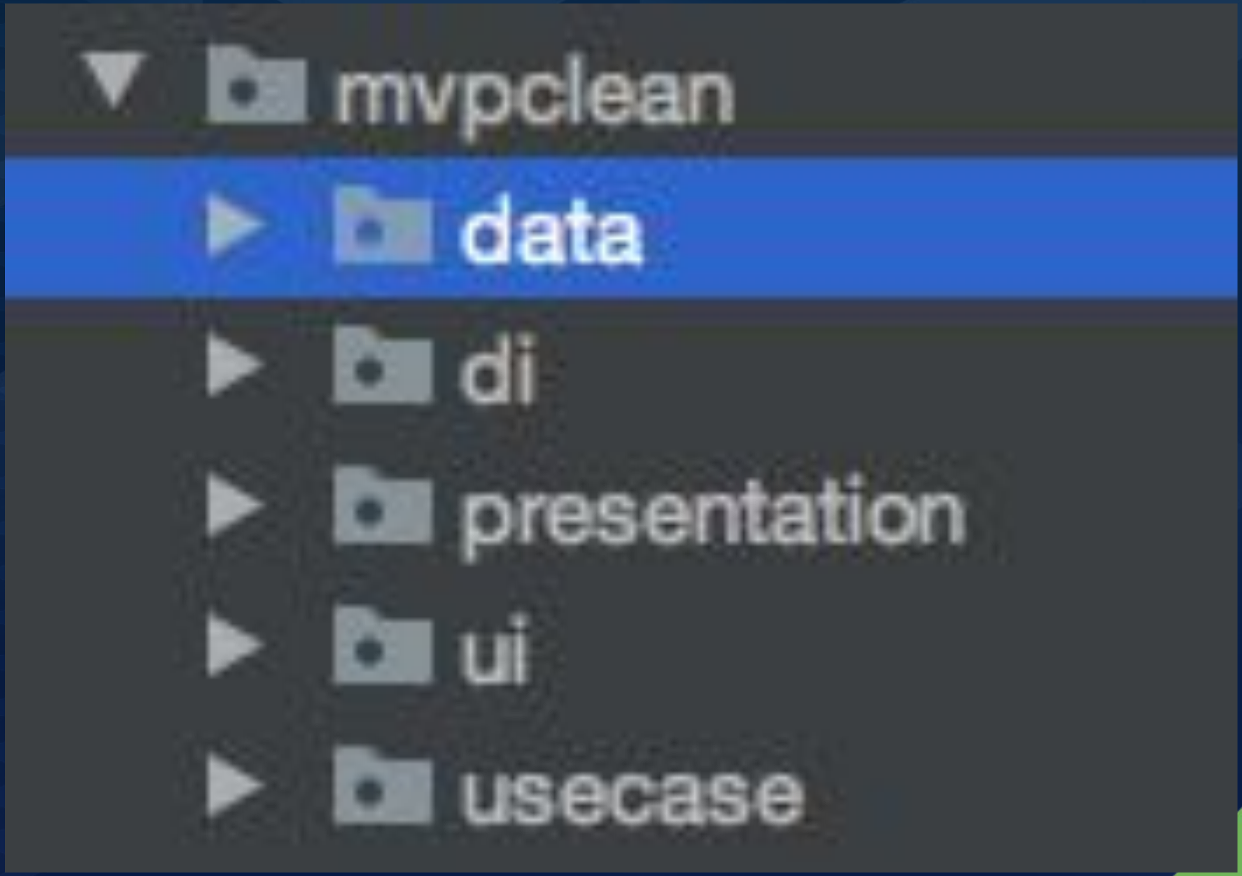
- 1 - MVC ✓
- 2 - MVP ✓
- 3 - MVVM ✓
- 4 - MVP Clean ✓
- **5 - MVVM Clean**
- 6 - MVI

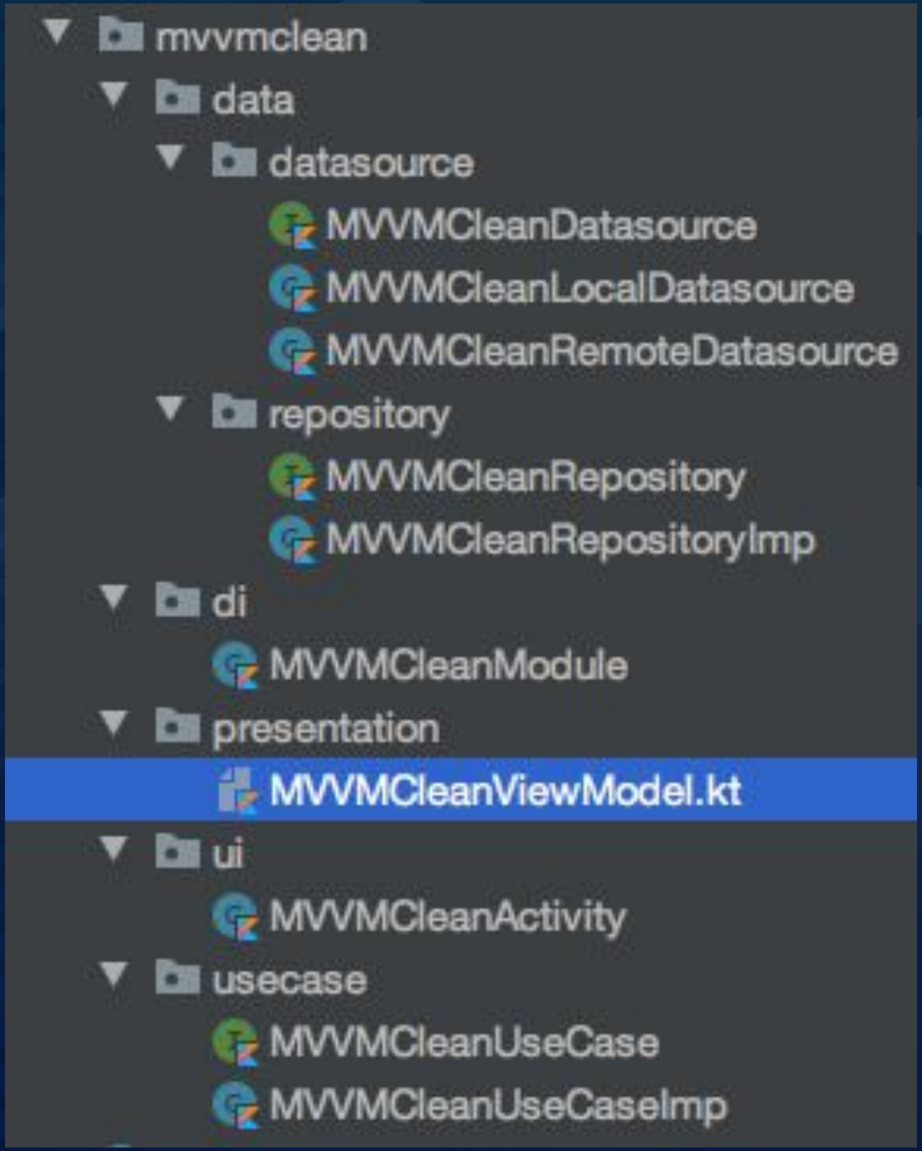

MVVM Clean

MVVM Clean

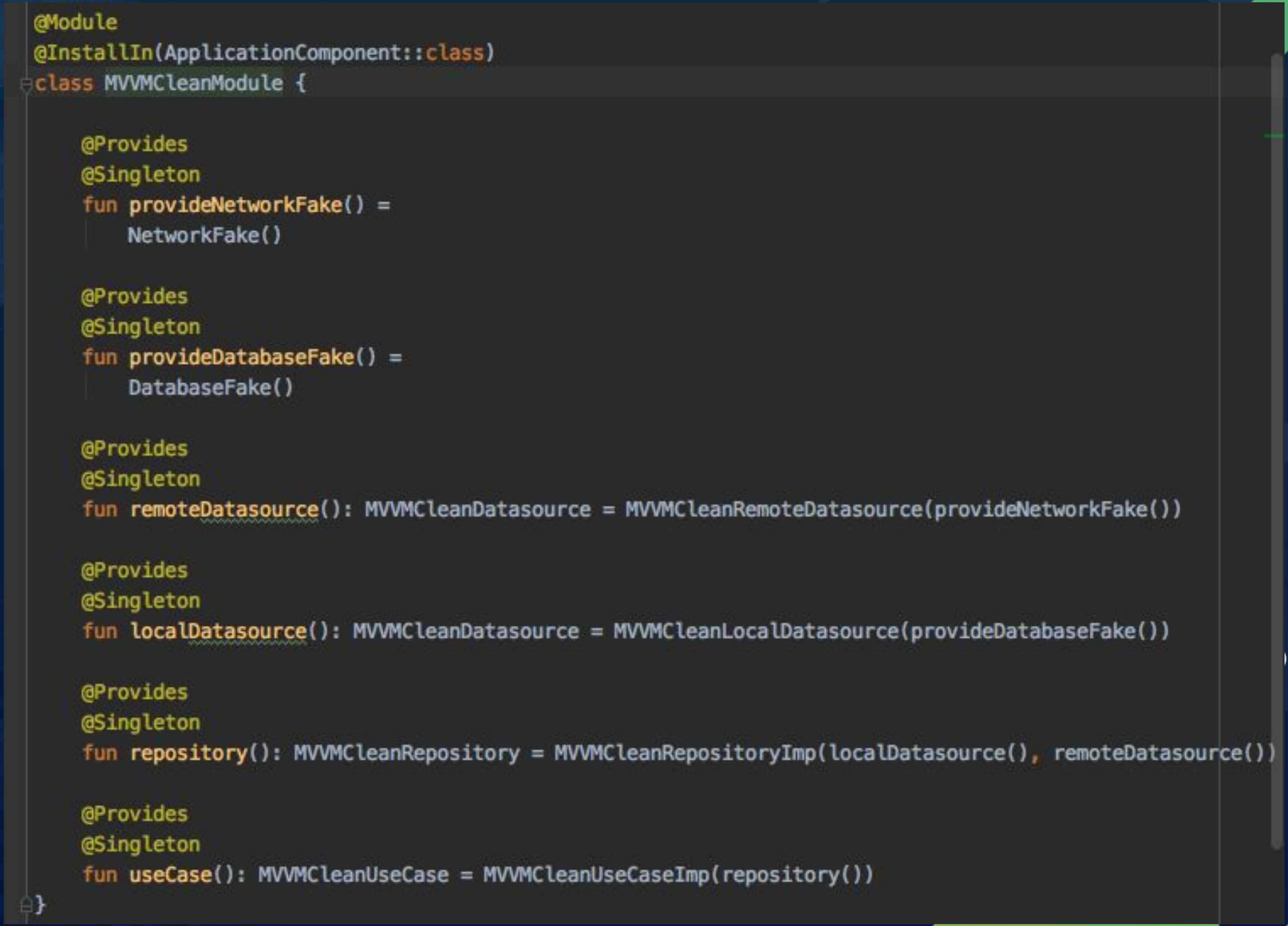
Aqui voce vera mais detalhes sobre o MVI.

https://github.com/nicconicco/Arch-DBZ/tree/master/app/src/main/java/com/nicco/architectures/android/mvvmclean

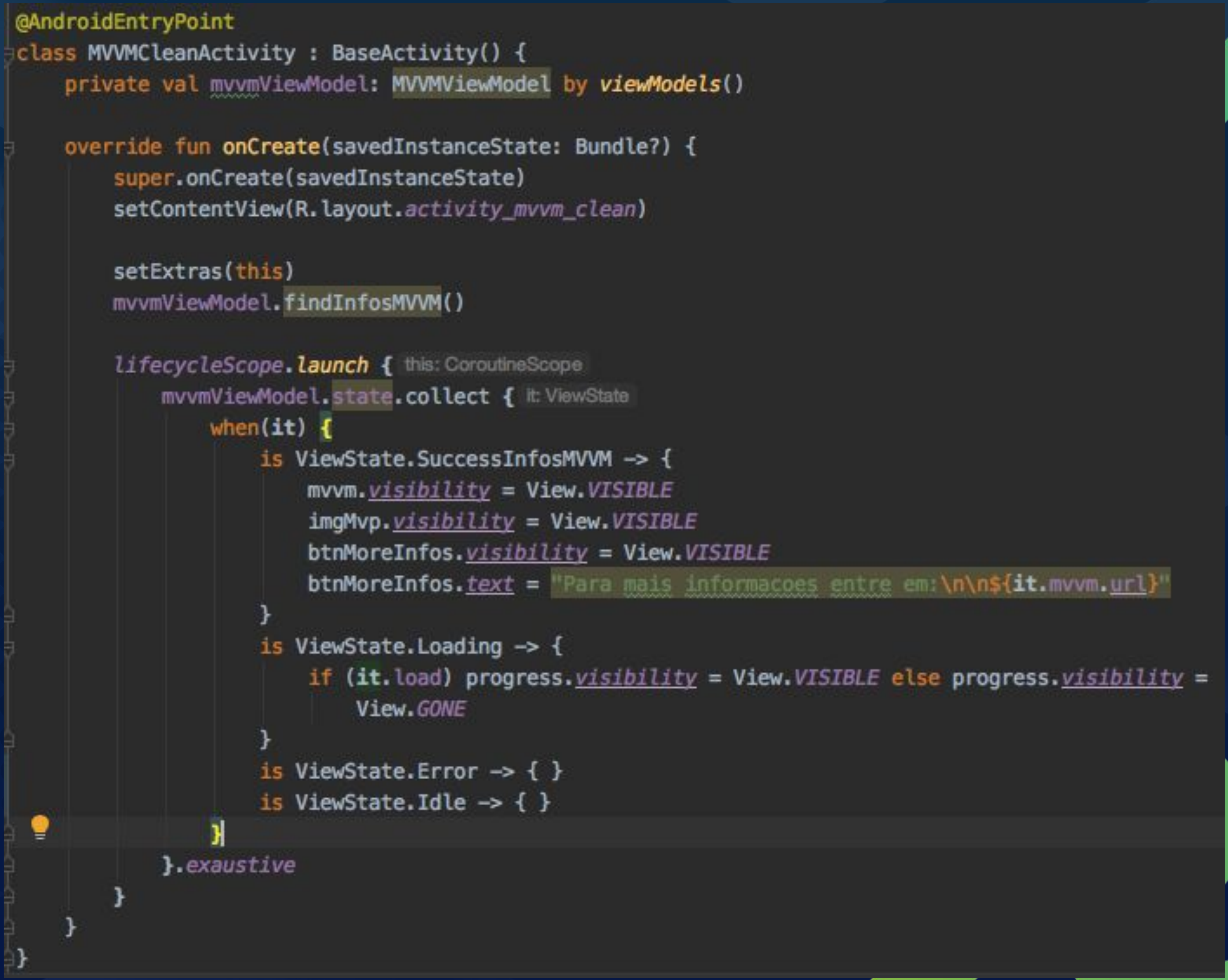

```
▼ mvpclean
    ▶ data
    ▶ di
    ▶ presentation
    ▶ ui
    ▶ usecase
```



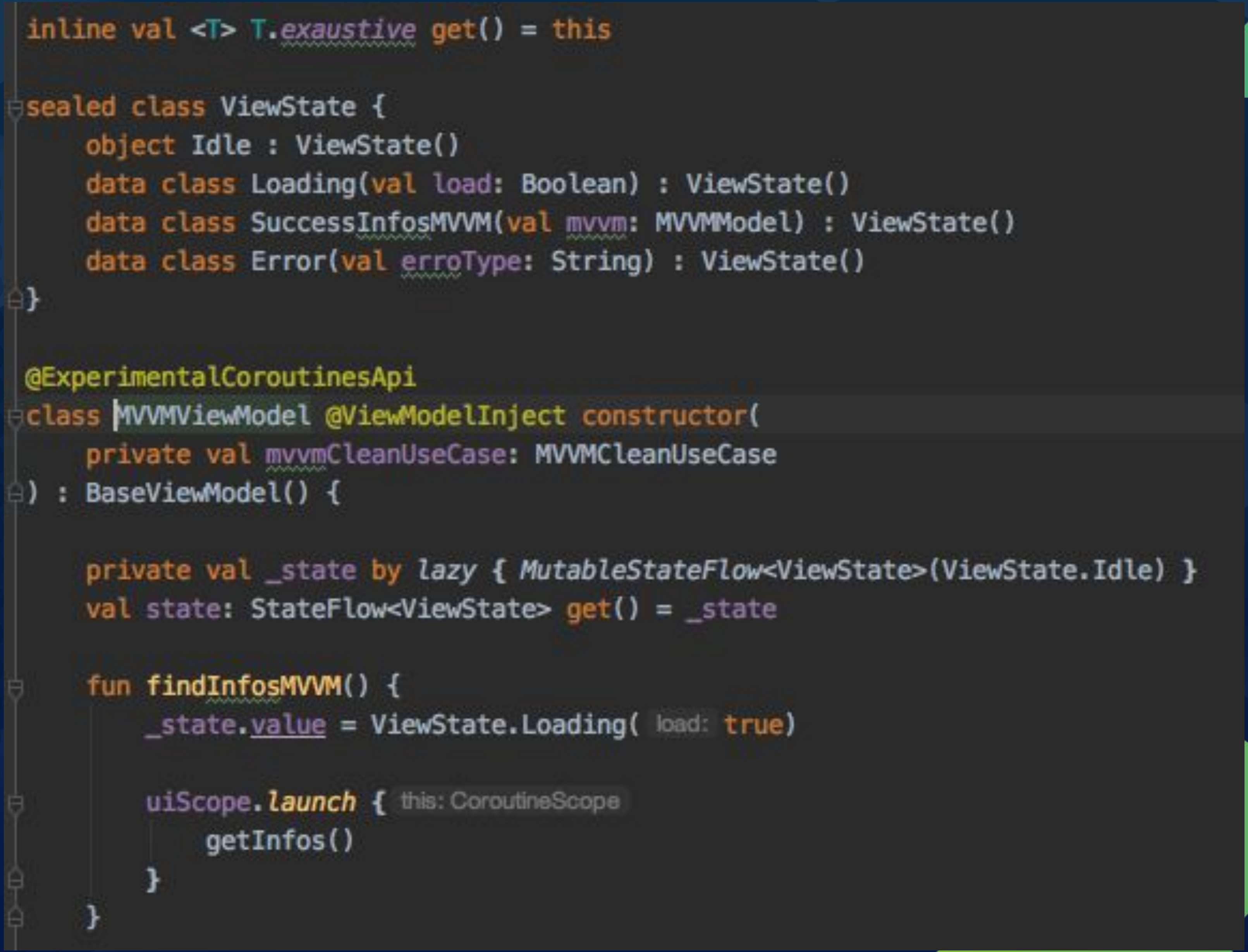
```
▼ mvvmclean
  ▼ data
    ▼ datasource
        MVVMCleanDatasource
        MVVMCleanLocalDatasource
        MVVMCleanRemoteDatasource
    ▼ repository
        MVVMCleanRepository
        MVVMCleanRepositoryImp
  ▼ di
      MVVMCleanModule
  ▼ presentation
      MVVMCleanViewModel.kt
  ▼ ui
      MVVMCleanActivity
  ▼ usecase
      MVVMCleanUseCase
      MVVMCleanUseCaseImp
```



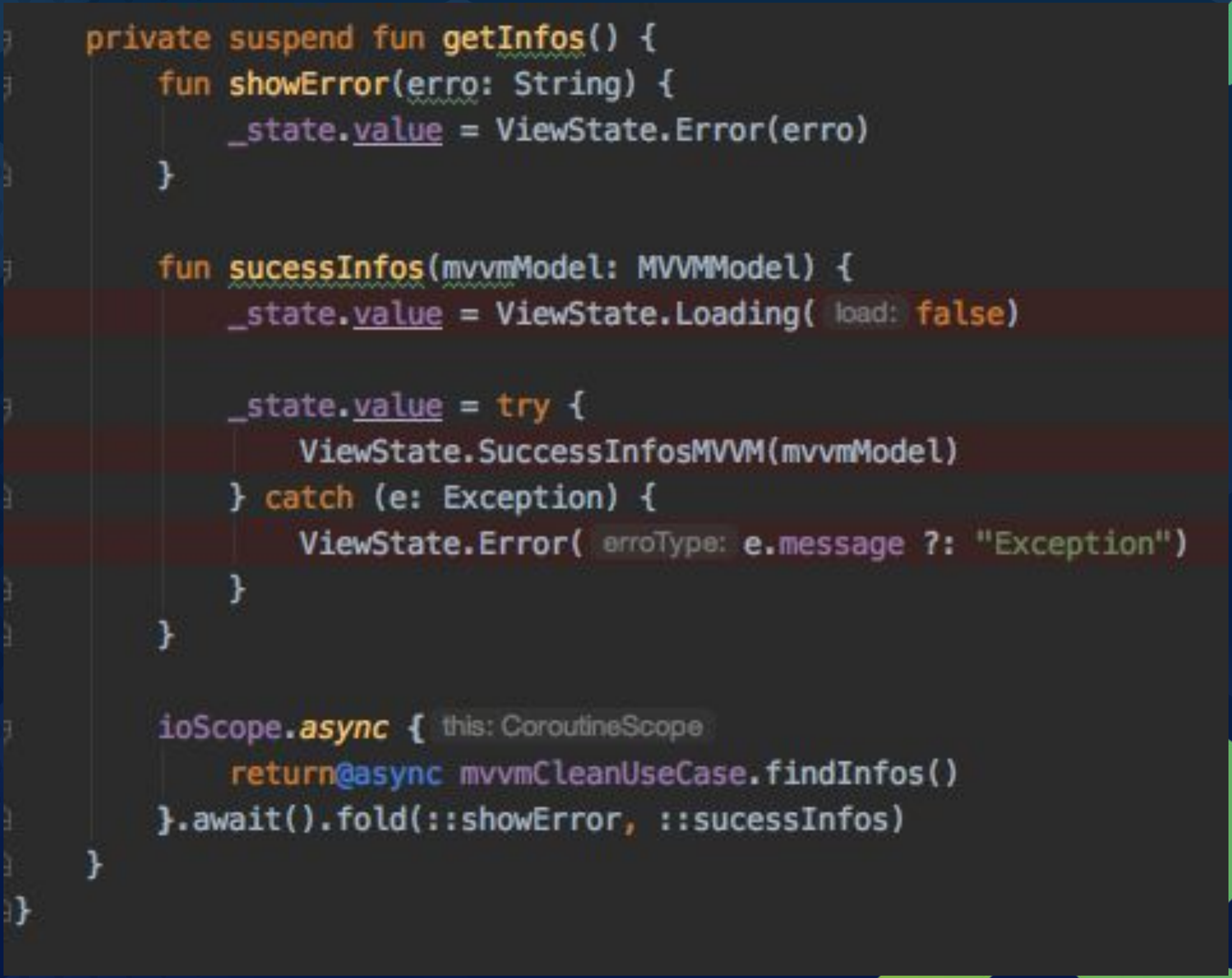
```
@Module
@InstallIn(ApplicationComponent::class)
class MVVMCleanModule {

    @Provides
    @Singleton
    fun provideNetworkFake() =
        NetworkFake()

    @Provides
    @Singleton
    fun provideDatabaseFake() =
        DatabaseFake()

    @Provides
    @Singleton
    fun remoteDatasource(): MVVMCleanDatasource = MVVMCleanRemoteDatasource(provideNetworkFake())

    @Provides
    @Singleton
    fun localDatasource(): MVVMCleanDatasource = MVVMCleanLocalDatasource(provideDatabaseFake())

    @Provides
    @Singleton
    fun repository(): MVVMCleanRepository = MVVMCleanRepositoryImp(localDatasource(), remoteDatasource())

    @Provides
    @Singleton
    fun useCase(): MVVMCleanUseCase = MVVMCleanUseCaseImp(repository())
}
```



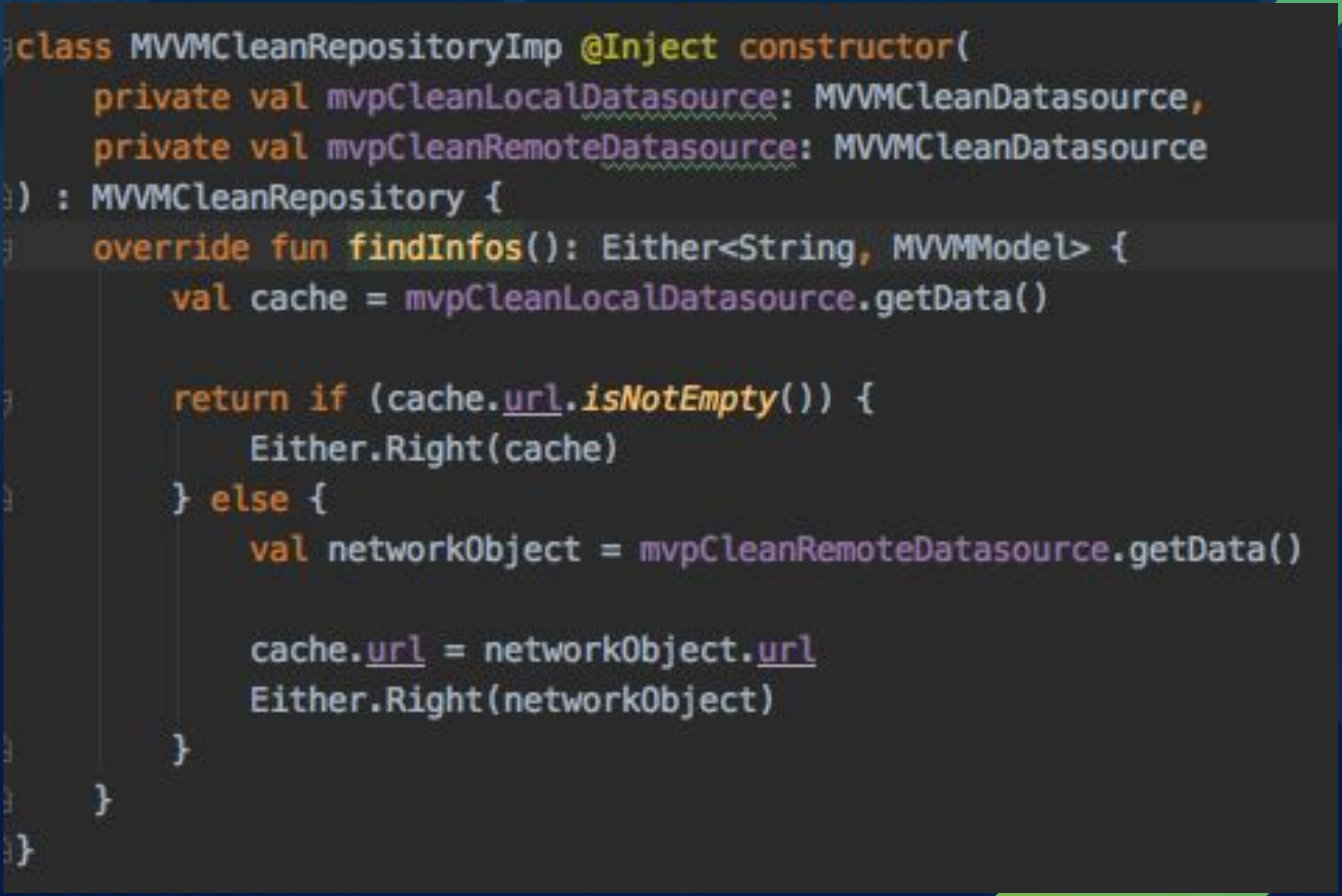
```kotlin
@AndroidEntryPoint
class MVVMCleanActivity : BaseActivity() {
    private val mvvmViewModel: MVVMViewModel by viewModels()

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_mvvm_clean)

        setExtras(this)
        mvvmViewModel.findInfosMVVM()

        lifecycleScope.launch { this: CoroutineScope
            mvvmViewModel.state.collect { it: ViewState
                when(it) {
                    is ViewState.SuccessInfosMVVM -> {
                        mvvm.visibility = View.VISIBLE
                        imgMvp.visibility = View.VISIBLE
                        btnMoreInfos.visibility = View.VISIBLE
                        btnMoreInfos.text = "Para mais informacoes entre em:\n\n${it.mvvm.url}"
                    }
                    is ViewState.Loading -> {
                        if (it.load) progress.visibility = View.VISIBLE else progress.visibility =
                            View.GONE
                    }
                    is ViewState.Error -> { }
                    is ViewState.Idle -> { }
                }
            }.exaustive
        }
    }
}
```



```kotlin
inline val <T> T.exaustive get() = this

sealed class ViewState {
    object Idle : ViewState()
    data class Loading(val load: Boolean) : ViewState()
    data class SuccessInfosMVVM(val mvvm: MVVMModel) : ViewState()
    data class Error(val erroType: String) : ViewState()
}

@ExperimentalCoroutinesApi
class MVVMViewModel @ViewModelInject constructor(
    private val mvvmCleanUseCase: MVVMCleanUseCase
) : BaseViewModel() {

    private val _state by lazy { MutableStateFlow<ViewState>(ViewState.Idle) }
    val state: StateFlow<ViewState> get() = _state

    fun findInfosMVVM() {
        _state.value = ViewState.Loading( load: true)

        uiScope.launch { this: CoroutineScope
            getInfos()
        }
    }
```



```kotlin
    private suspend fun getInfos() {
        fun showError(erro: String) {
            _state.value = ViewState.Error(erro)
        }

        fun sucessInfos(mvvmModel: MVVMModel) {
            _state.value = ViewState.Loading( load: false)

            _state.value = try {
                ViewState.SuccessInfosMVVM(mvvmModel)
            } catch (e: Exception) {
                ViewState.Error( erroType: e.message ?: "Exception")
            }
        }

        ioScope.async { this: CoroutineScope
            return@async mvvmCleanUseCase.findInfos()
        }.await().fold(::showError, ::sucessInfos)
    }
}
```



```
class MVVMCleanRepositoryImp @Inject constructor(
    private val mvpCleanLocalDatasource: MVVMCleanDatasource,
    private val mvpCleanRemoteDatasource: MVVMCleanDatasource
) : MVVMCleanRepository {
    override fun findInfos(): Either<String, MVVMModel> {
        val cache = mvpCleanLocalDatasource.getData()

        return if (cache.url.isNotEmpty()) {
            Either.Right(cache)
        } else {
            val networkObject = mvpCleanRemoteDatasource.getData()

            cache.url = networkObject.url
            Either.Right(networkObject)
        }
    }
}
```

Model View ViewModel Clean

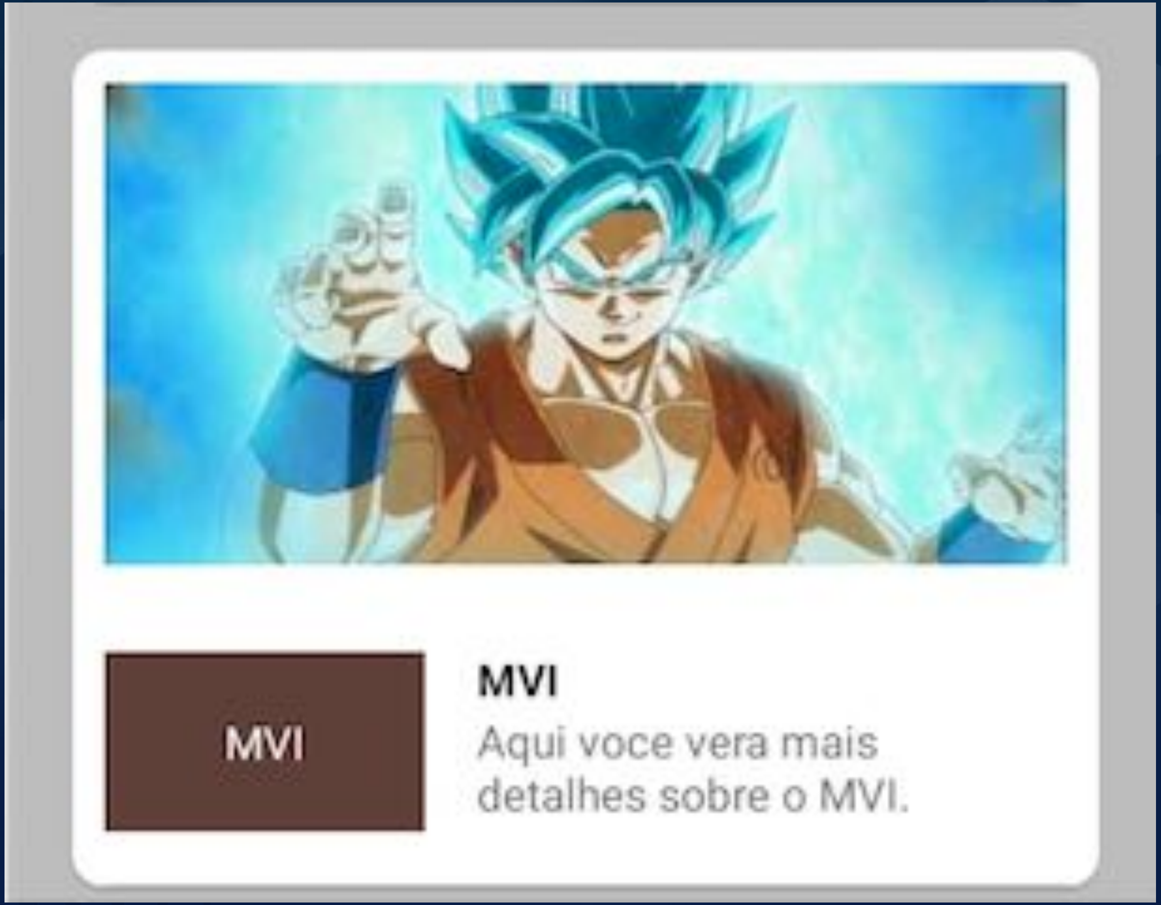
- 1 - Activity /* - ui ✓
- 2 - ViewModel - Coroutines / States / StateFlow ✓
- 3 - UseCase / Repository / Datasource ✓
- 4 - Dependency Injection: Hilt ✓

Vantagens: Para projetos grandes atende bem. Responsabilidades bem definidas. Testável.

Desvantagens: Exige mais do programador. Code Review forte em respeitar as regras.

Arquiteturas em Projetos Android

- 1 - MVC ✓
- 2 - MVP ✓
- 3 - MVVM ✓
- 4 - MVP Clean ✓
- 5 -MVVM Clean ✓
- 6 - MVI

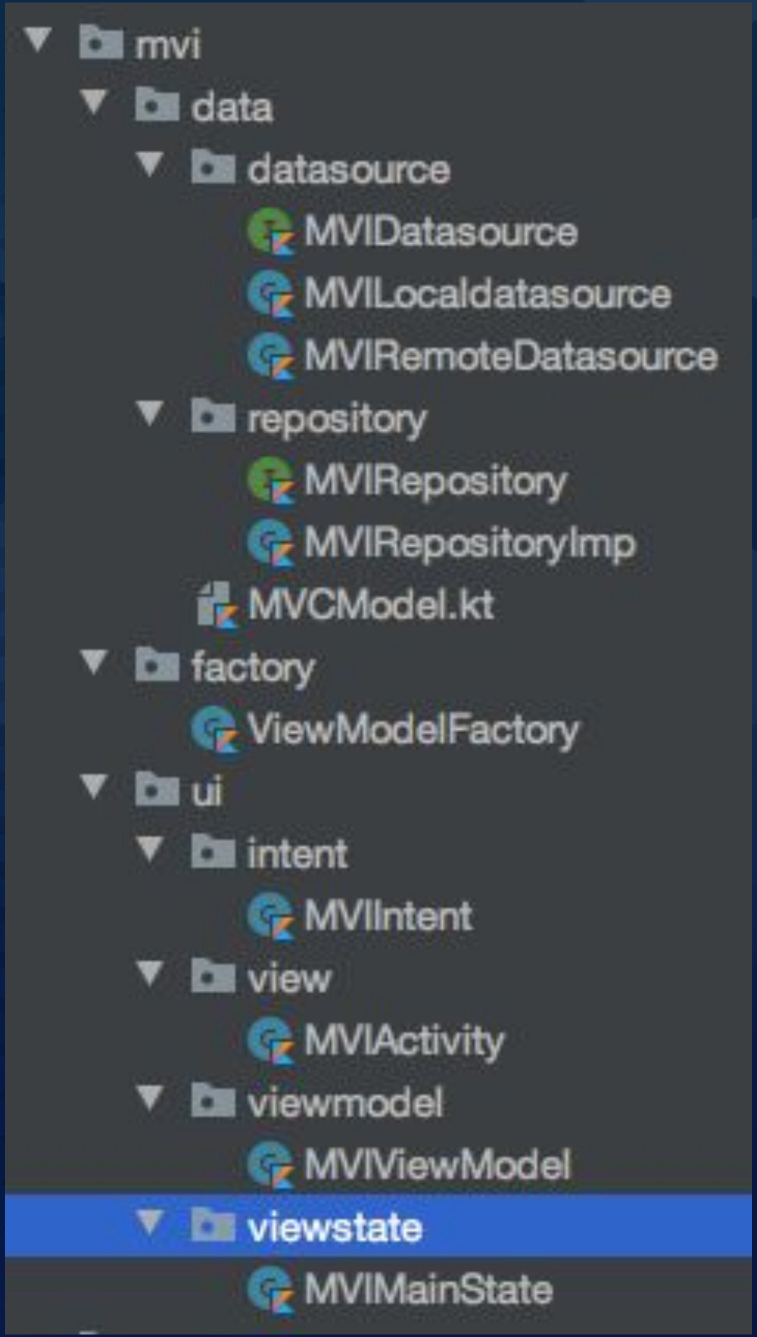


MVI

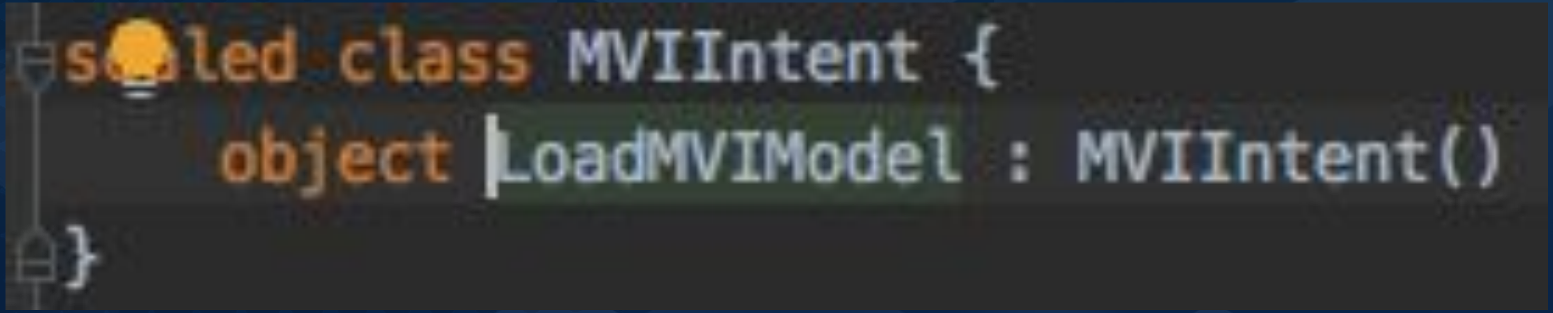
MVI

Aqui voce vera mais detalhes sobre o MVI.

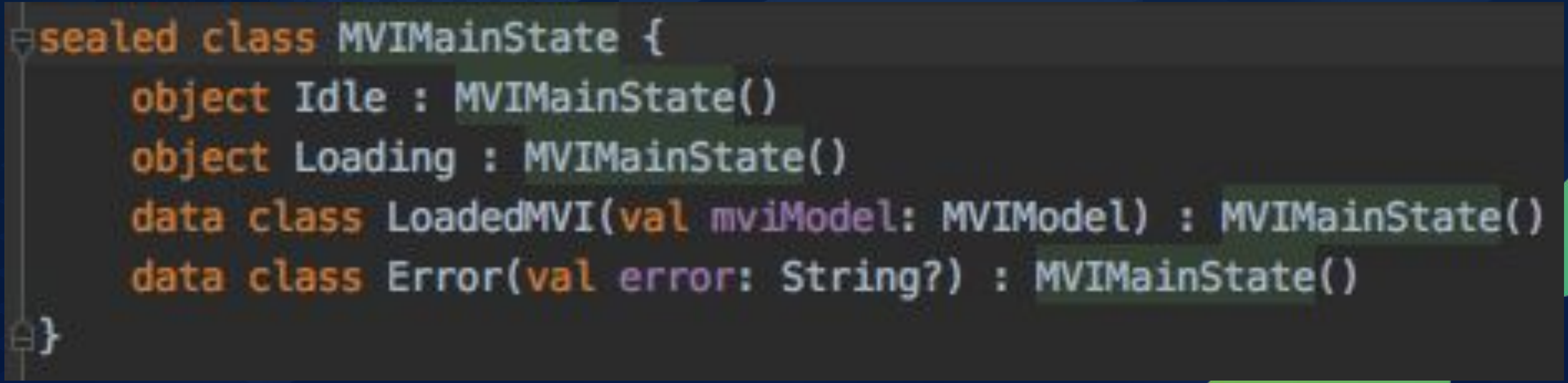
https://github.com/nicconicco/Arch-DBZ/tree/master/app/src/main/java/com/nicco/architectures/android/mvi



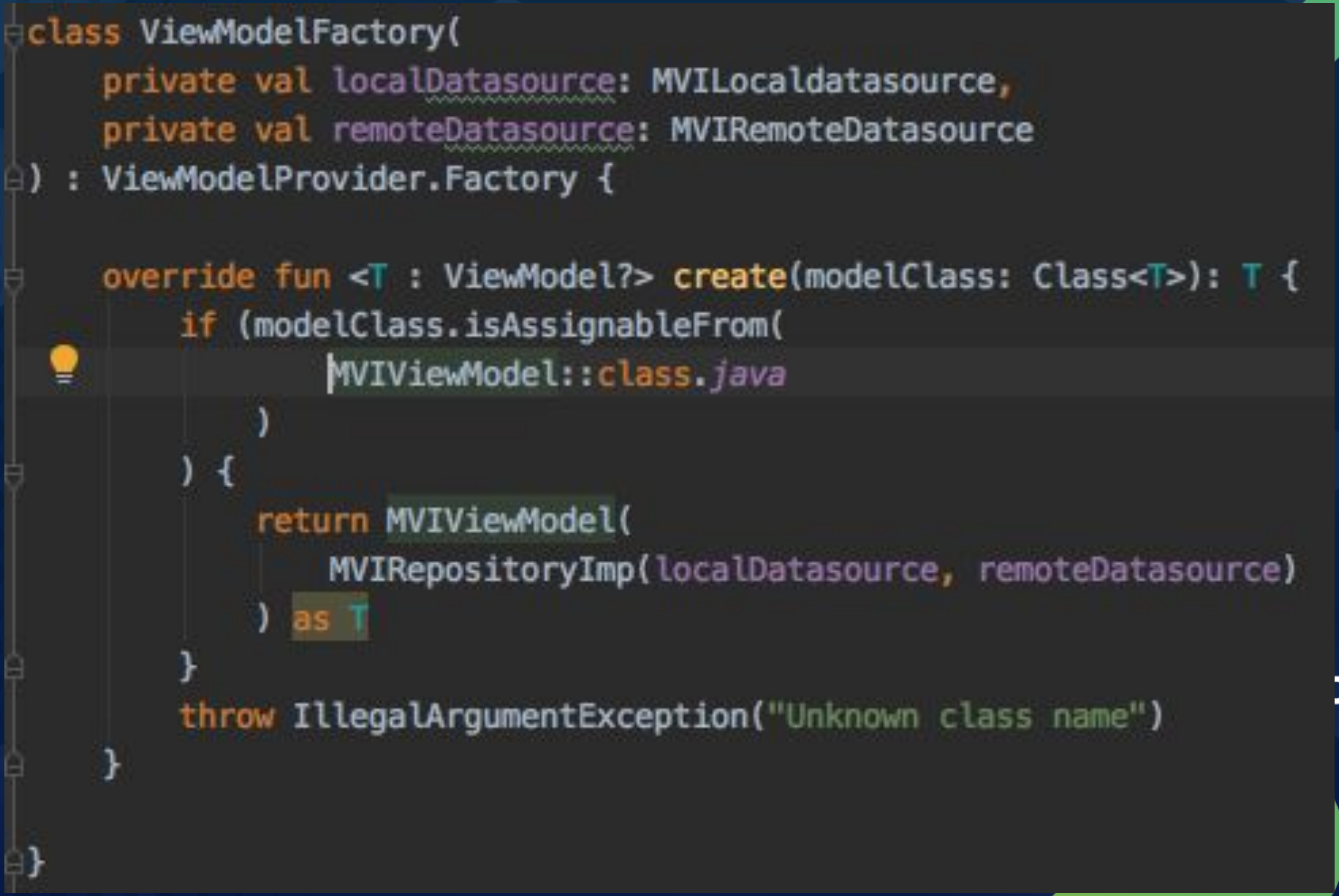
```
▼ mvi
    ▼ data
        ▼ datasource
              MVIDatasource
              MVILocaldatasource
              MVIRemoteDatasource
        ▼ repository
              MVIRepository
              MVIRepositoryImp
          MVCModel.kt
    ▼ factory
          ViewModelFactory
    ▼ ui
        ▼ intent
              MVIIntent
        ▼ view
              MVIActivity
        ▼ viewmodel
              MVIViewModel
        ▼ viewstate
              MVIMainState
```



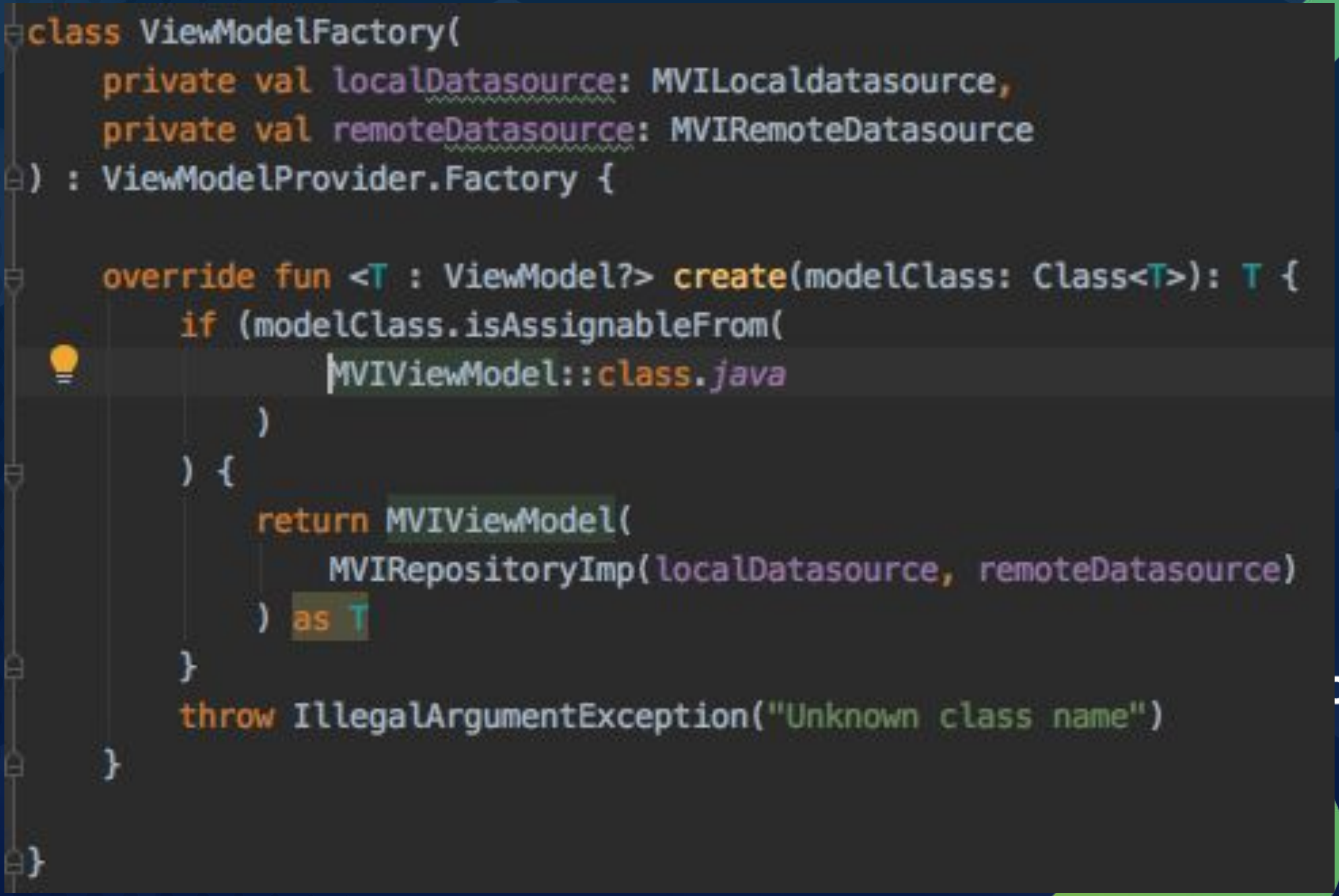
```
sealed class MVIIntent {
    object LoadMVIModel : MVIIntent()
}
```



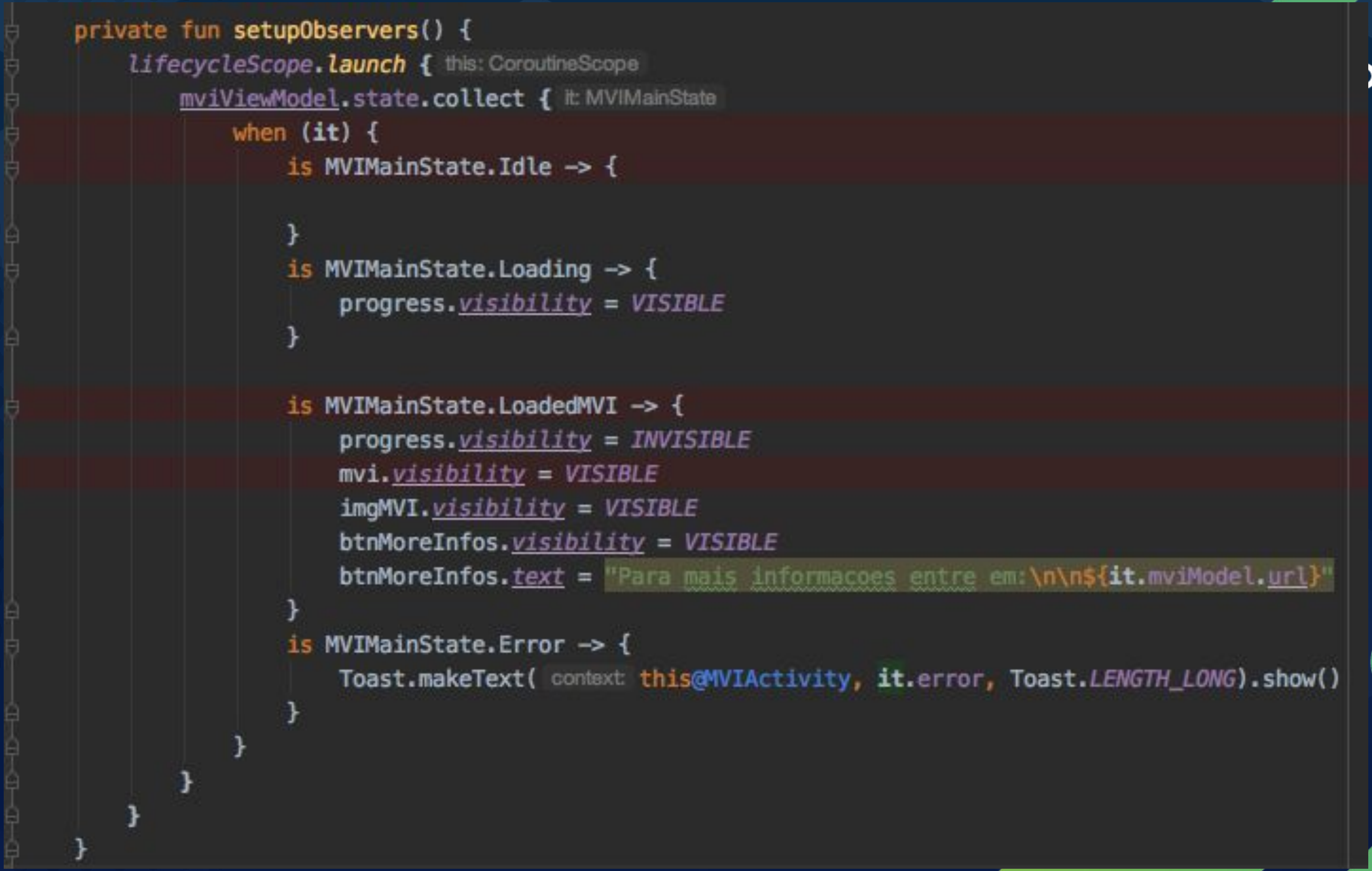
```
sealed class MVIMainState {
    object Idle : MVIMainState()
    object Loading : MVIMainState()
    data class LoadedMVI(val mviModel: MVIModel) : MVIMainState()
    data class Error(val error: String?) : MVIMainState()
}
```



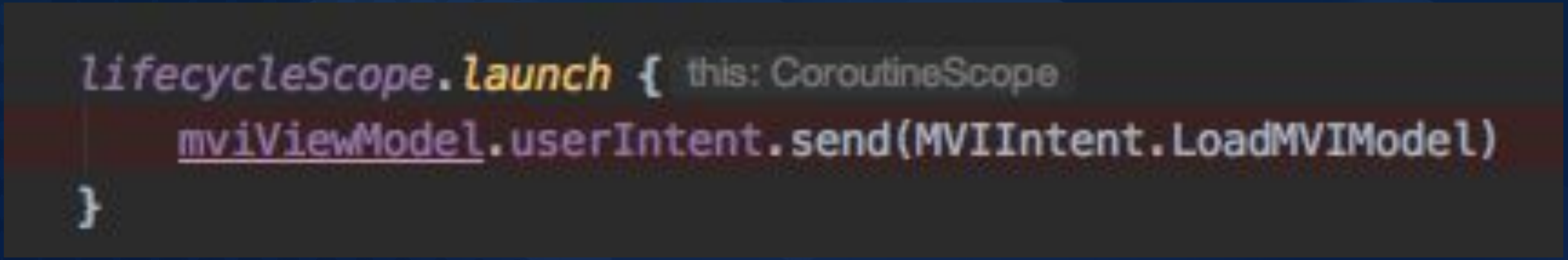
```kotlin
class ViewModelFactory(
    private val localDatasource: MVILocaldatasource,
    private val remoteDatasource: MVIRemoteDatasource
) : ViewModelProvider.Factory {

    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(
                    MVIViewModel::class.java
                )
        ) {
            return MVIViewModel(
                MVIRepositoryImp(localDatasource, remoteDatasource)
            ) as T
        }
        throw IllegalArgumentException("Unknown class name")
    }

}
```



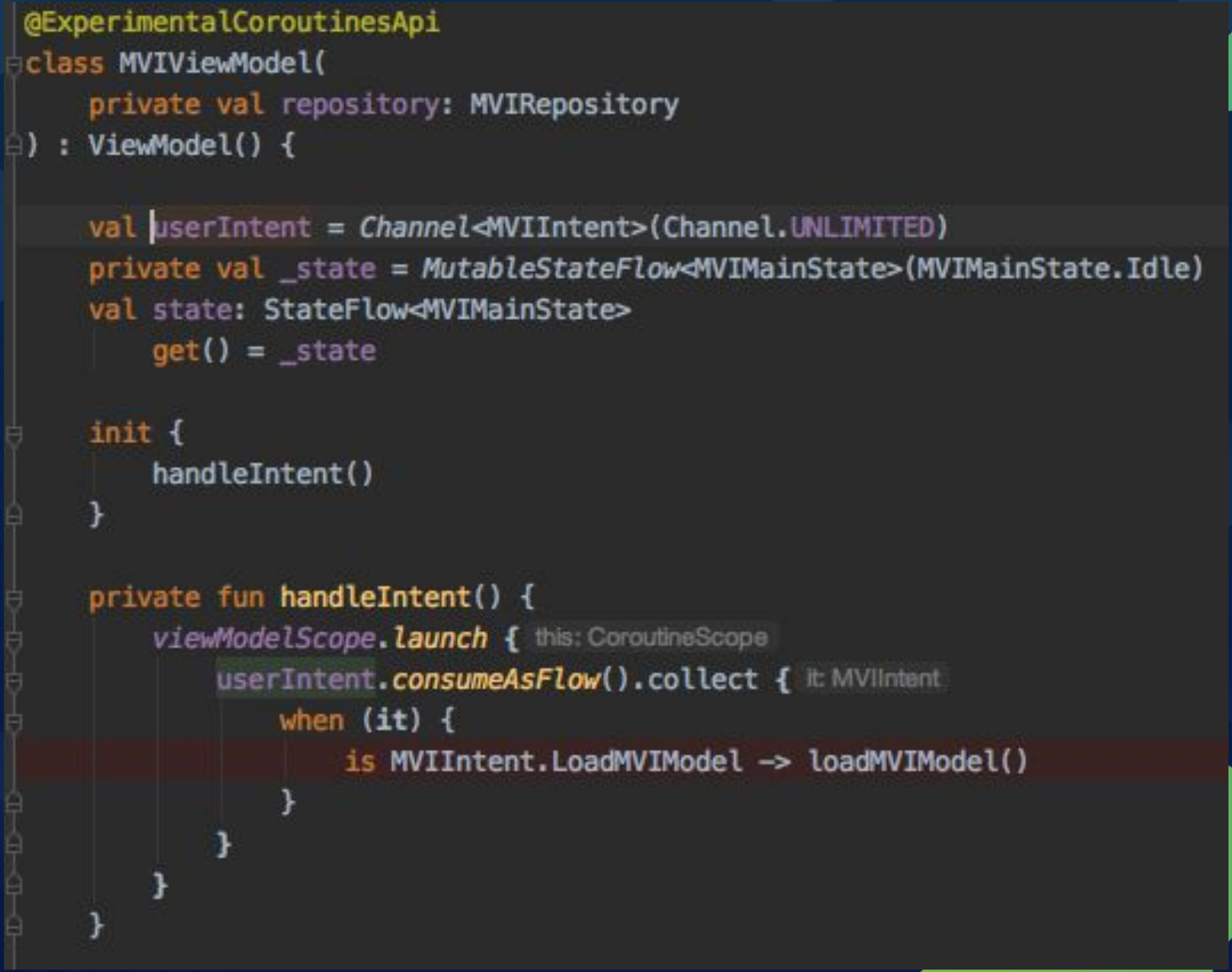
```
class ViewModelFactory(
    private val localDatasource: MVILocaldatasource,
    private val remoteDatasource: MVIRemoteDatasource
) : ViewModelProvider.Factory {

    override fun <T : ViewModel?> create(modelClass: Class<T>): T {
        if (modelClass.isAssignableFrom(
                MVIViewModel::class.java
            )
        ) {
            return MVIViewModel(
                MVIRepositoryImp(localDatasource, remoteDatasource)
            ) as T
        }
        throw IllegalArgumentException("Unknown class name")
    }

}
```



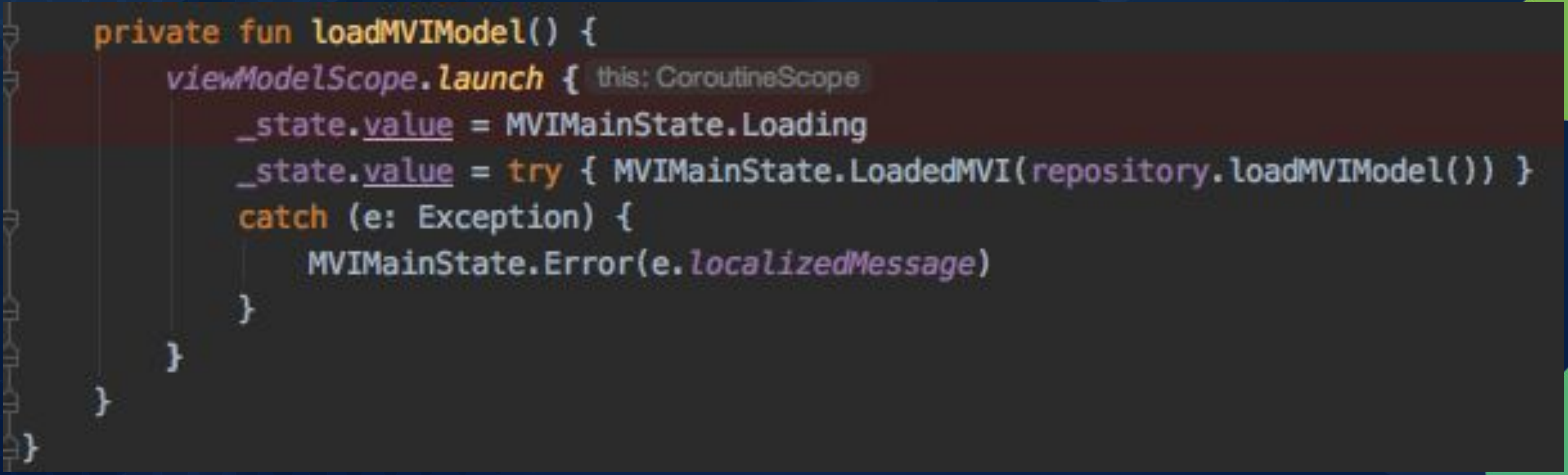
```
private fun setupObservers() {
    lifecycleScope.launch { this: CoroutineScope
        mviViewModel.state.collect { it: MVIMainState
            when (it) {
                is MVIMainState.Idle -> {

                }
                is MVIMainState.Loading -> {
                    progress.visibility = VISIBLE
                }

                is MVIMainState.LoadedMVI -> {
                    progress.visibility = INVISIBLE
                    mvi.visibility = VISIBLE
                    imgMVI.visibility = VISIBLE
                    btnMoreInfos.visibility = VISIBLE
                    btnMoreInfos.text = "Para mais informacoes entre em:\n\n${it.mviModel.url}"
                }
                is MVIMainState.Error -> {
                    Toast.makeText( context: this@MVIActivity, it.error, Toast.LENGTH_LONG).show()
                }
            }
        }
    }
}
```



```
lifecycleScope.launch { this: CoroutineScope
    mviViewModel.userIntent.send(MVIIntent.LoadMVIModel)
}
```



```
@ExperimentalCoroutinesApi
class MVIViewModel(
    private val repository: MVIRepository
) : ViewModel() {

    val userIntent = Channel<MVIIntent>(Channel.UNLIMITED)
    private val _state = MutableStateFlow<MVIMainState>(MVIMainState.Idle)
    val state: StateFlow<MVIMainState>
        get() = _state

    init {
        handleIntent()
    }

    private fun handleIntent() {
        viewModelScope.launch { this: CoroutineScope
            userIntent.consumeAsFlow().collect { it: MVIIntent
                when (it) {
                    is MVIIntent.LoadMVIModel -> loadMVIModel()
                }
            }
        }
    }
```



```
private fun loadMVIModel() {
    viewModelScope.launch { this: CoroutineScope
        _state.value = MVIMainState.Loading
        _state.value = try { MVIMainState.LoadedMVI(repository.loadMVIModel()) }
        catch (e: Exception) {
            MVIMainState.Error(e.localizedMessage)
        }
    }
}
```

Model View ViewModel Clean

- 1 - Activity /* - ui - intent - viewstate ✓
- 2 - ViewModel - Coroutines / States / StateFlow / Channel ✓
- 3 - UseCase / Repository / Datasource ✓
- 4 - Dependency Injection: Nativo ✓

Vantagens: Ação do usuário bem definida, testável, clean.

Desvantagens: Muitas ações serem muito parecidas causando um aumento de código se não se tomar cuidado.

Arquiteturas em Projetos Android

- 1 - MVC ✓
- 2 - MVP ✓
- 3 - MVVM ✓
- 4 - MVP Clean ✓
- 5 -MVVM Clean ✓
- 6 - MVI ✓

Carlos Nicolau Galves

Tech Lead Android na Zup Innovation | Co-organizer Android Curitiba

São Paulo, São Paulo, Brasil · + de 500 conexões ·

Informações de contato

Zup Innovation

Universidade Positivo

Github: github.com/nicconicco/Arch-DBZ

Carlos N. Galves

Tech Lead

carlos.galves@zup.com.br

@niccocwb -> Telegram

Twitter: niccocwb
Medium: @nicolaugalves

Obrigado!