Android App

Rybak Andriian

Android App

- Kotlin
- MVVM
- LiveData
- ViewModel
- Room

Kotlin

```kotlin
fun sum(a: Int, b: Int): Int {
    return a + b
}

fun sum(a: Int, b: Int) = a + b

fun printSum(a: Int, b: Int) {
    println("sum of $a and $b is ${a + b}")
}

val a: Int = 1  // immediate assignment
val b = 2       // `Int` type is inferred
val c: Int      // Type required when no initializer is provided
c = 3           // deferred assignment
```

Kotlin

```
var x = 5 // `Int` type is inferred
x += 1

var a = 1
// simple name in template:
val s1 = "a is $a"
a = 2
// arbitrary expression in template:
val s2 = "${s1.replace("is", "was")}, but now is $a"

fun parseInt(str: String): Int? {
    // ...

    return null
}
```

Kotlin

```
fun maxOf(a: Int, b: Int): Int {
    if (a > b) {
        return a
    } else {
        return b
    }
}

fun maxOf(a: Int, b: Int) = if (a > b) a else b
```

Kotlin

```
val fruits = listOf("banana", "avocado", "apple", "kiwifruit")
fruits
        .filter { it.startsWith("a") }
        .sortedBy { it }
        .map { it.toUpperCase() }
        .forEach { println(it) }
```

Kotlin

```
object ReportUtil {

    fun filterJobExtIdsUntilDate(jobs: List<Job>, date: Long) = jobs
            .asSequence()
            .filter { job -> job.updatedAt < date }
            .map { job -> job.externalId }
            .toSet()

    fun filterAndSortJobsFromDate(jobs: List<Job>, date: Long) = jobs
            .asSequence()
            .filter { job -> job.updatedAt >= date }
            .sortedBy { job -> job.updatedAt }
            .toList()
}
```

Kotlin: *Null Safety*

```
val a = "Kotlin"
val b: String? = null
println(b?.length)
println(a?.length) // Unnecessary safe call

val listWithNulls: List<String?> = listOf("Kotlin", null)
for (item in listWithNulls) {
    item?.let { println(it) } // prints Kotlin and ignores null
}

submitted_from_list.apply {
    layoutManager = LinearLayoutManager(context)
    adapter = HistoryAdapter(context)
}
```

Android + Kotlin

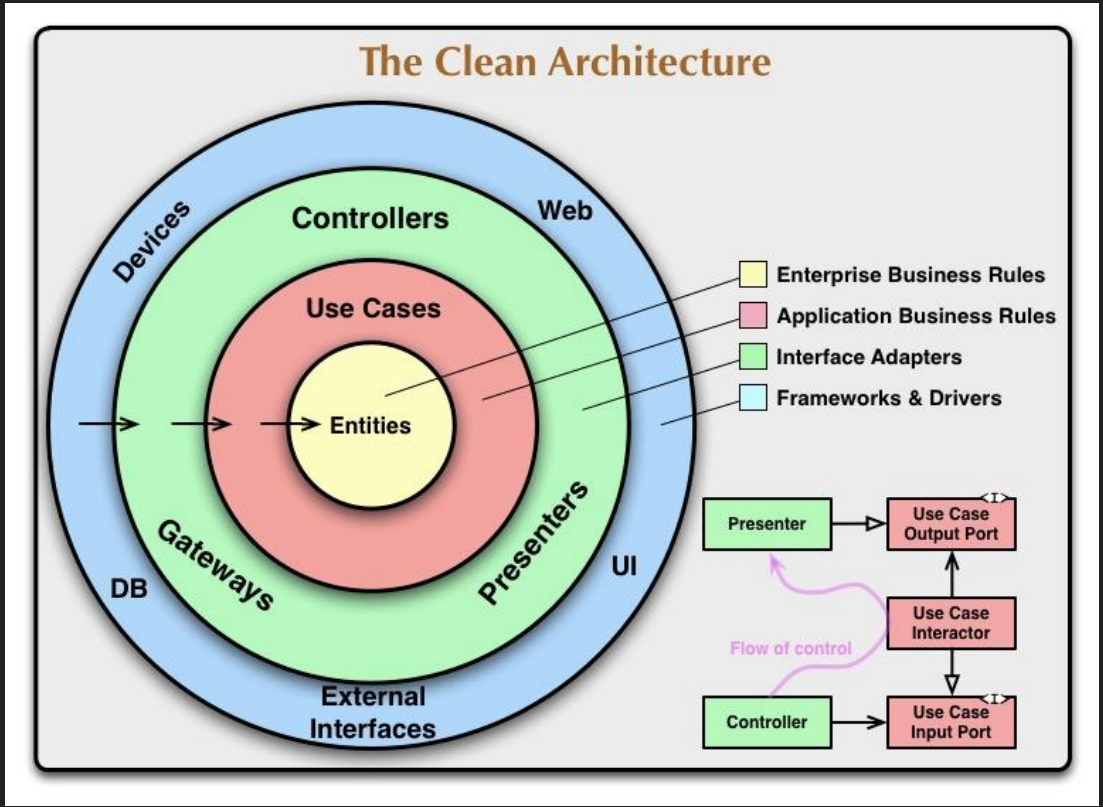
```
<FrameLayout
  android:id="@+id/tasksContainer"
  android:layout_width="match_parent"
  android:layout_height="wrap_content" />

ViewGroup tasksContainer = (ViewGroup) findViewById(R.id.tasksContainer);
```

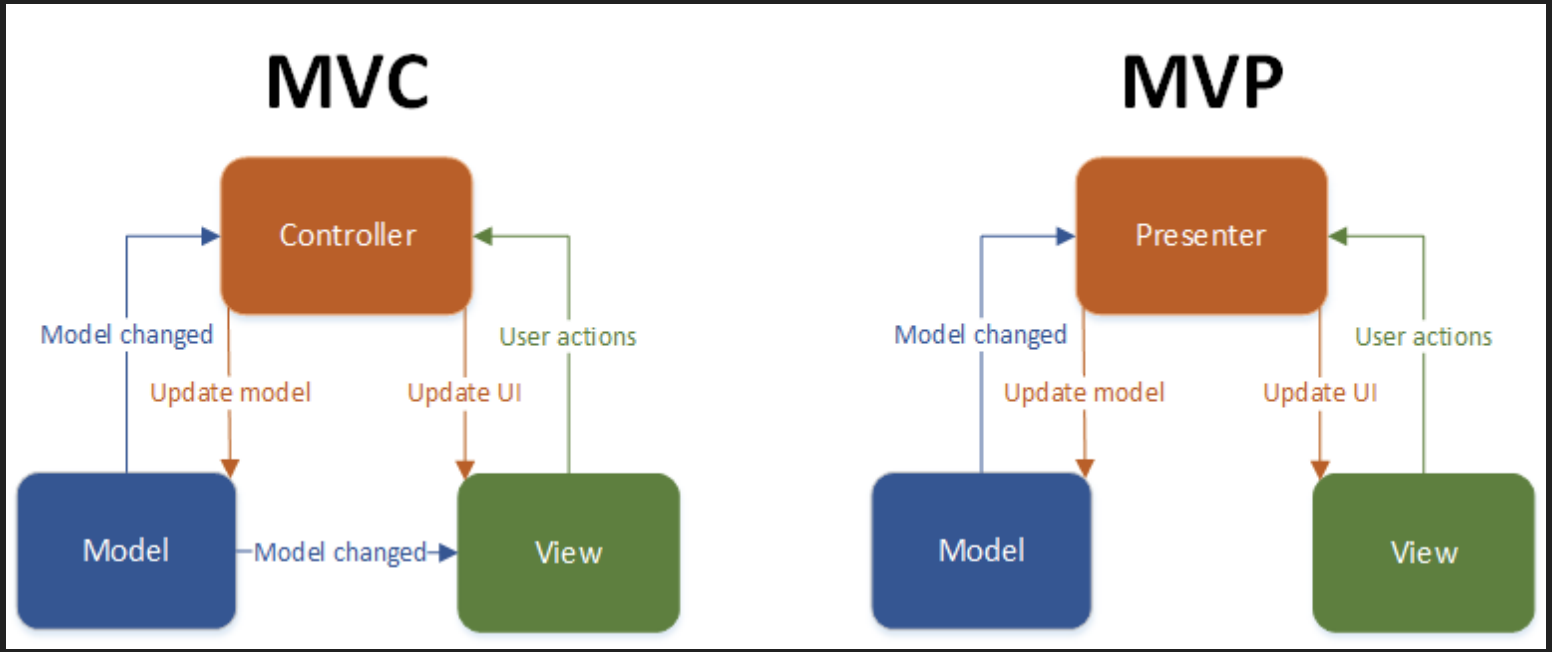
```
<android.support.v7.widget.RecyclerView
  android:id="@+id/submitted_from_list"
  android:layout_width="match_parent"
  android:layout_height="match_parent" />

submitted_from_list.visibility = View.GONE
```

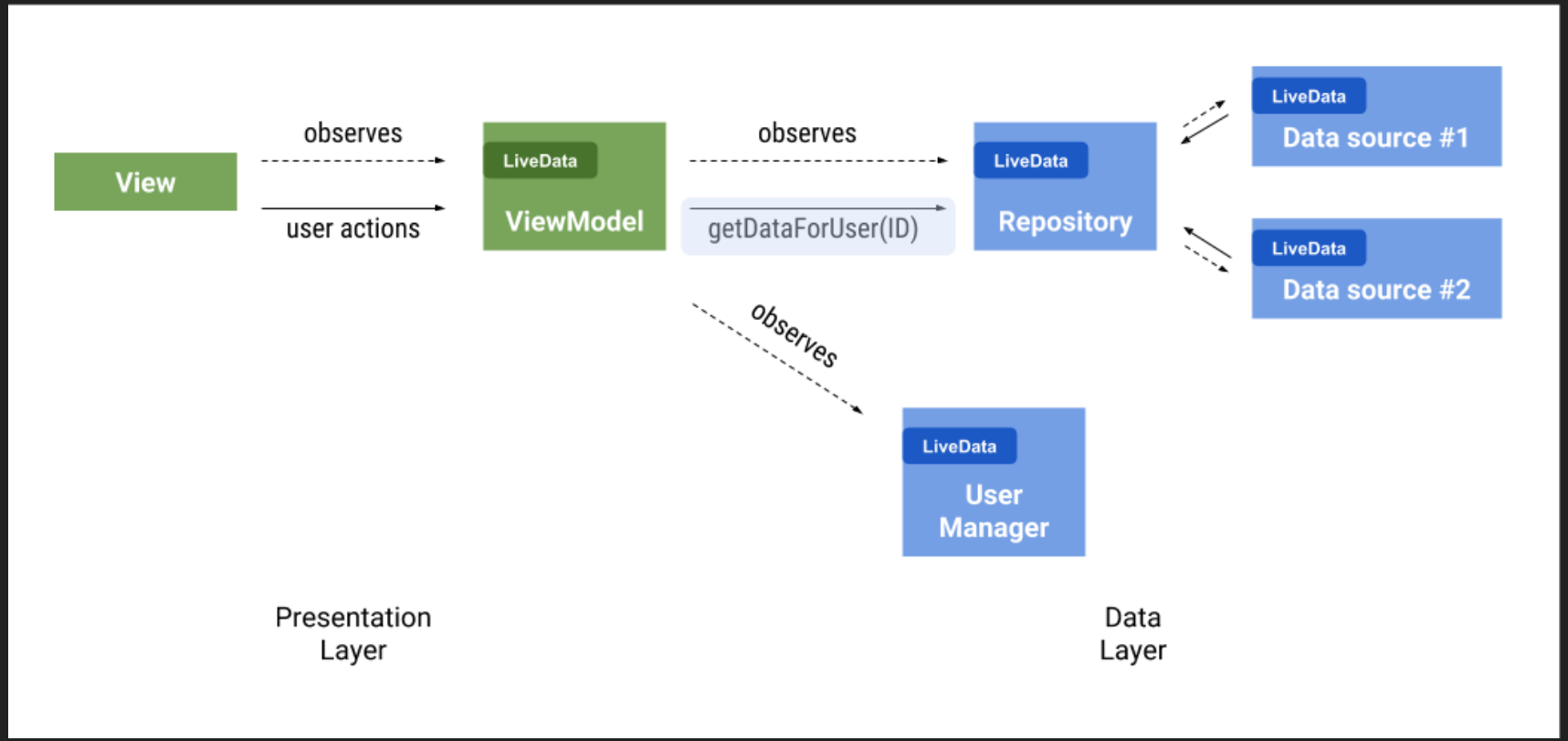
Clean Architecture



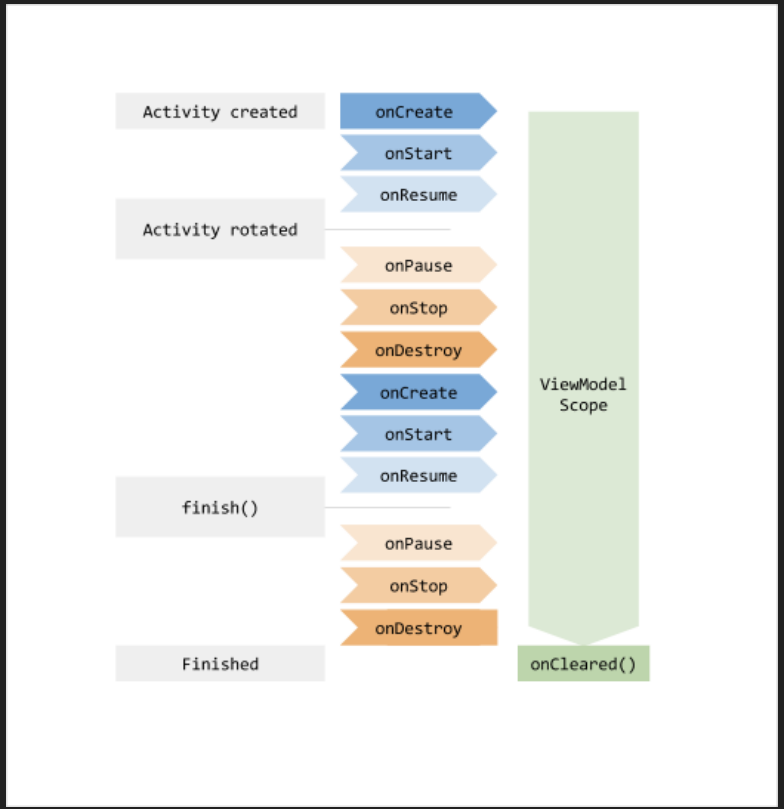
MVC vs MVP


MVC
Controller
Model changed
Update model
Update UI
User actions
Model
Model changed
View
MVP
Presenter
Model changed
Update model
Update UI
User actions
Model
View

Android + MVVM



ViewModel


Activity created
onCreate
onStart
onResume
Activity rotated
onPause
onStop
onDestroy
onCreate
onStart
onResume
finish()
onPause
onStop
onDestroy
Finished
ViewModel Scope
onCleared()

Android + Room

```
@Entity(tableName = SubmittedFormDao.TABLE_NAME)
data class SubmittedFormEntity(

    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = SubmittedFormDao.LOCAL_ID, typeAffinity = ColumnInfo.INTEGER)
    var localId: Long = 0,

    @ColumnInfo(name = SubmittedFormDao.USER_EXTERNAL_ID, typeAffinity = ColumnInfo.TEXT)
    var userExternalId: String,

    @ColumnInfo(name = SubmittedFormDao.SUBMITTED_AT, typeAffinity = ColumnInfo.INTEGER)
    var submittedAt: Long,

    @ColumnInfo(name = SubmittedFormDao.JSON_DATA, typeAffinity = ColumnInfo.TEXT)
    var jsonData: JSONObject
)
```

Android + Room

```
class DataConverter {

    @TypeConverter
    fun toString(json: JSONObject) = json.toString()

    @TypeConverter
    fun toJSONObject(string: String) = try {
        JSONObject(string)
    } catch (e: JSONException) {
        JSONObject()
    }
}
```

Android + Room

```
@Dao
interface SubmittedFormDao {
   @Query("SELECT * FROM $TABLE_NAME WHERE $USER_EXTERNAL_ID = :userExternalId AND
$SUBMITTED_AT >= :fromDate")
   fun getSubmittedForms(userExternalId: String, fromDate: Long): LiveData<List<SubmittedFormEntity>>

   @Insert(onConflict = OnConflictStrategy.REPLACE)
   fun addSubmittedForm(submittedForm: SubmittedFormEntity)

   companion object {
      const val TABLE_NAME = "submitted_form"
      const val LOCAL_ID = "local_id"
      const val USER_EXTERNAL_ID = "user_external_id"
      const val SUBMITTED_AT = "submitted_at"
      const val JSON_DATA = "json_data
   }
}
```

Android + Room

```
@Database(entities = [SubmittedFormEntity::class], version = 1, exportSchema = false)
@TypeConverters(DataConverter::class)
abstract class AppDatabase : RoomDatabase() {

    abstract fun submittedFormDao(): SubmittedFormDao

    companion object {
        fun buildDatabase(context: Context) = Room
                .databaseBuilder(context.applicationContext, AppDatabase::class.java, DATABASE_NAME)
                .build()
    }
}
```

Android + Room

```
interface IDbManager {
    fun addSubmittedForm(submittedForm: SubmittedFormEntity)
    fun getSubmittedForms(userExternalId: String, fromDate: Long): LiveData<List<SubmittedFormEntity>>
}

class DbManager(context: Context) : IDbManager {
    private val database = AppDatabase.buildDatabase(context)

    override fun addSubmittedForm(submittedForm: SubmittedFormEntity) {
        database.submittedFormDao().addSubmittedForm(submittedForm)
    }

    override fun getSubmittedForms(userExternalId: String, fromDate: Long):
LiveData<List<SubmittedFormEntity>> {
        return database.submittedFormDao().getSubmittedForms(userExternalId, fromDate)
    }
}
```

Android + Kotlin + MVVM

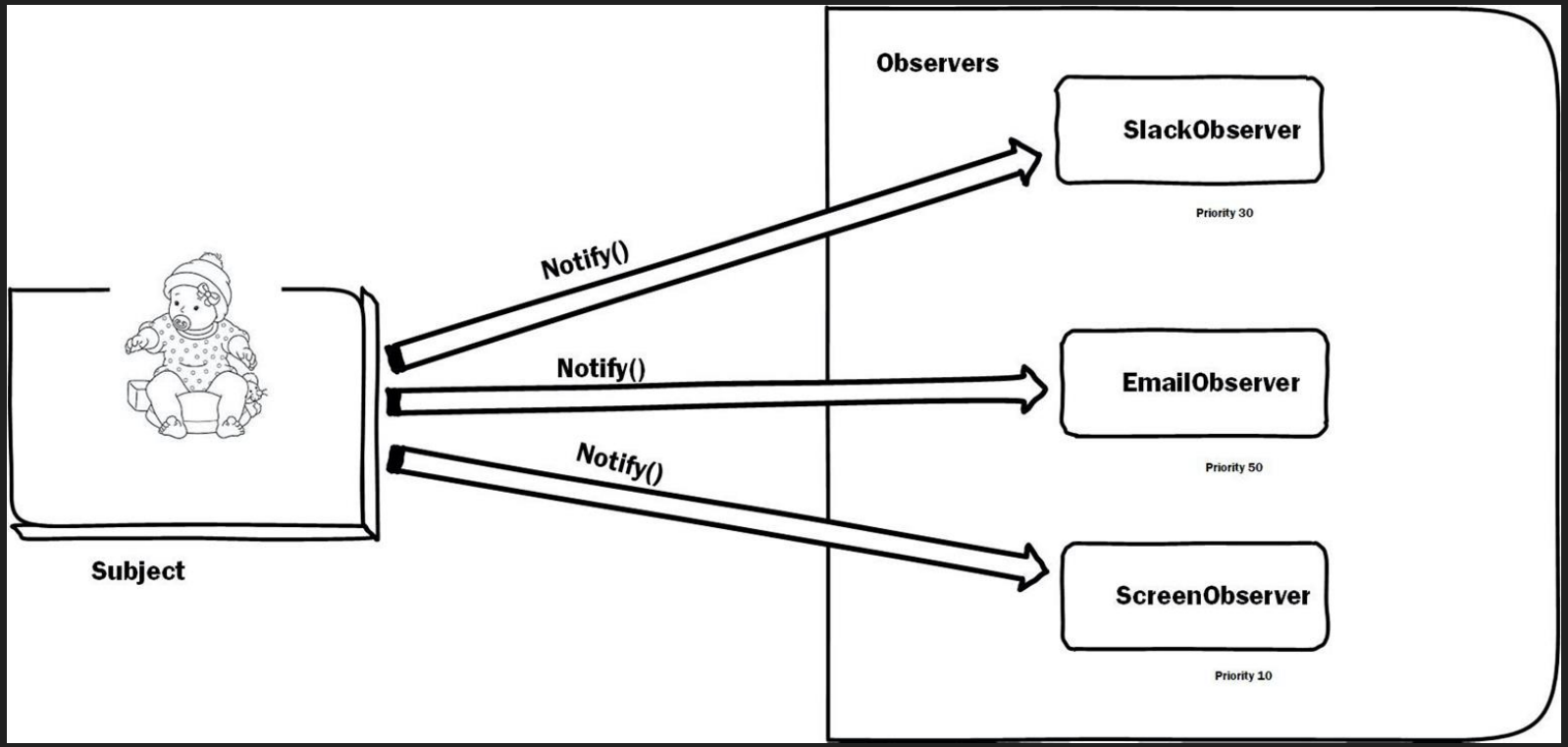
```
interface IBaseModelController {
    /*add specific items*/
}

abstract class BaseViewModel<M : IBaseModelController>(protected val modelController: M) : ViewModel()

interface IModelController : IBaseModelController {
    val activeUserId: Long

    fun getSubmittedFormsByLastWeek(userId: Long): LiveData<List<FormHistory>>
}
```

LiveData as observable pattern



Android + Kotlin + MVVM

```
class HistoryViewModel(modelController: IModelController) :
BaseViewModel<IModelController>(modelController) {

    fun getFormHistory(): LiveData<List<SubmittedFormViewData>> {
        val userId = modelController.activeUserId
        val submittedFormsLiveData = modelController.getSubmittedFormsByLastWeek(userId)
        return Transformations.map(submittedFormsLiveData) { data ->
            data?.let { HistoryViewDataConverter.convertToSubmittedFormsViewData(it) }
        }
    }
}
```

Android + Kotlin + MVVM

```
class HistoryViewModel(modelController: IModelController) :
BaseViewModel<IModelController>(modelController) {

    fun getFormHistory(): LiveData<List<SubmittedFormViewData>> {
        val userId = modelController.activeUserId
        val submittedFormsLiveData = modelController.getSubmittedFormsByLastWeek(userId)
        return Transformations.map(submittedFormsLiveData) { data ->
            data?.let { HistoryViewDataConverter.convertToSubmittedFormsViewData(it) }
        }
    }
}
```

Android + Kotlin + MVVM

```
class Factory(private val historyRepository: IHistoryRepository) : ViewModelProvider.Factory {

@Suppress("UNCHECKED_CAST")
   override fun <T : ViewModel?> create(modelClass: Class<T>): T {
      return HistoryViewModel(object : IModelController {
         override val activeUserId: Long
            get() = historyRepository.activeUserId

         override fun getSubmittedFormsByLastWeek(userId: Long): LiveData<List<FormHistory>> {
            return historyRepository.getSubmittedFormsByLastWeek(userId)
         }
      }) as T
   }
}
```

Android + Kotlin + MVVM

```
class HistoryFragment : BaseFragment() {
    private val viewModel: HistoryViewModel by lazy {
        val repository = Application.getInstance(context).repository
        ViewModelProviders.of(this, HistoryViewModel.Factory(repository)).get(HistoryViewModel::class.java)
    }
    override fun onViewCreated(view: View?, savedInstanceState: Bundle?) {
        subscribeUi(viewModel)
    }
    private fun subscribeUi(viewModel: HistoryViewModel) {
        viewModel.getFormHistory().observe(this, Observer { data ->
            data?.run {
                historyAdapter.setData(this)
                empty_content.visibility = if (this.isEmpty()) View.VISIBLE else View.GONE
            }
        })
    }
}
```

Android + Kotlin + MVVM

```
class Application : android.app.Application() {
    private lateinit var macroFacade: MacroFacade
    val repository: IRepository
        get() = macroFacade

    override fun onCreate() {
        macroFacade = MacroFacade(applicationContext)
        macroFacade.init()
    }

    companion object {
        fun getInstance(context: Context): Application {
            val appContext = context.applicationContext
            if (appContext is Application) return appContext
            throw IllegalStateException("Wrong application class is applied.")
        }
    }
}
```

Thank you for attention!