Reuse features in Android applications

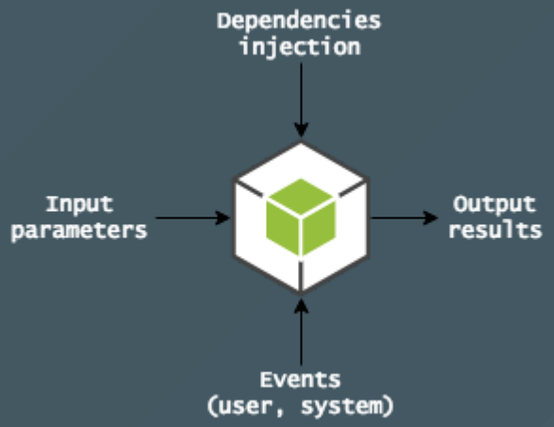
An introduction to component modularity

Romain Rochegude

Introduction

What to reuse?

- mainly focused on UI concerns
- independent, reusable and isolated
- unit of code to compile (i.e., Android Studio module)
- Almost like React Component



Background: key concepts

- Activity (since API level 1)
- Fragment (since API level 11)
- ViewModel (since ACC)

Activity (since API level 1)

“ one of the fundamental building blocks

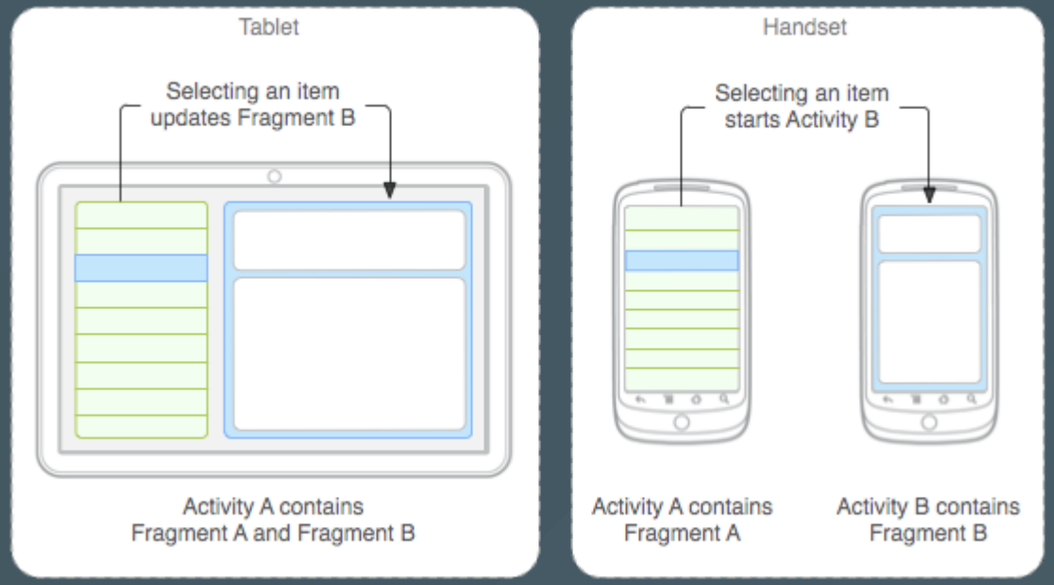
https://developer.android.com/guide/components/activities/index.html ”

- behind every screen stands a single `Activity`

Fragment (since API level 11)

“ portion of user interface in an `Activity`

https://android-developers.googleblog.com/2011/02/android-30-fragments-api.html ”

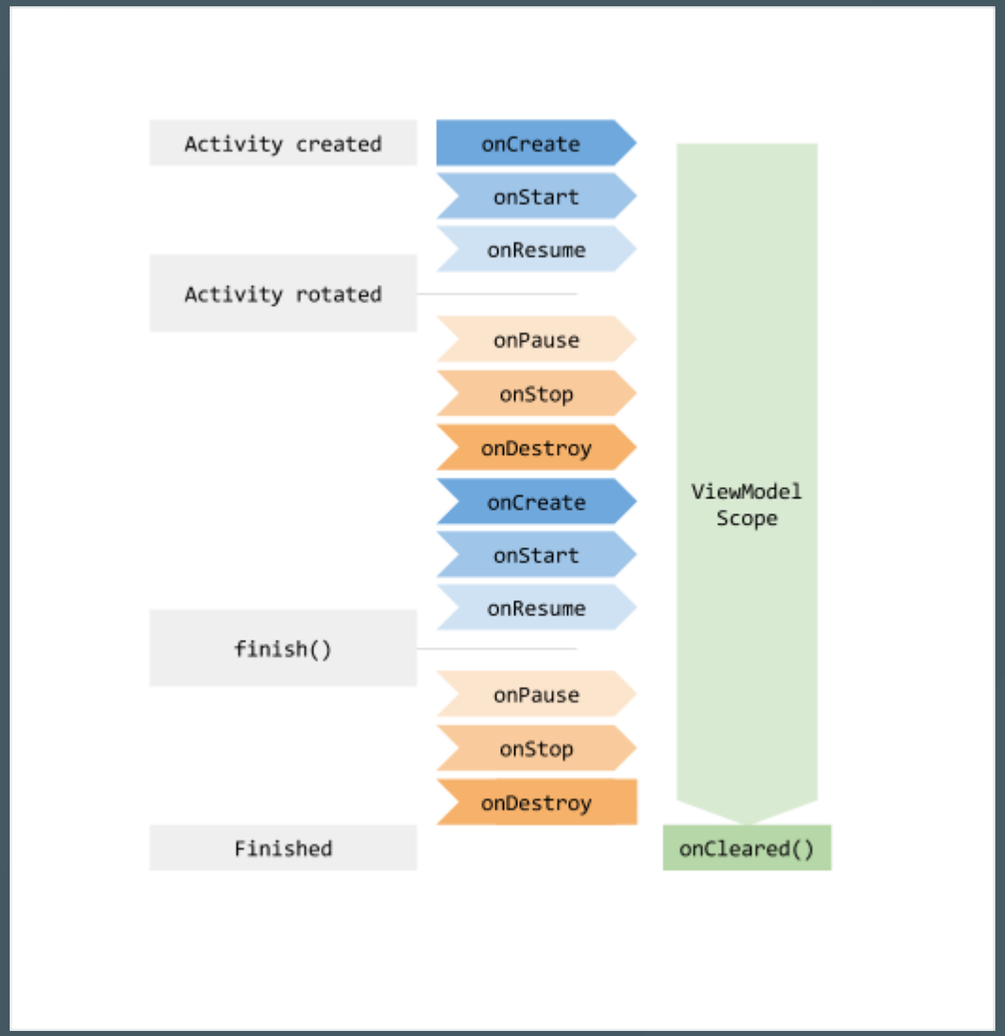


ViewModel (since ACC)

“ The `ViewModel` class

- is designed to store and manage UI-related data in a lifecycle conscious way.
- allows data to survive configuration changes such as screen rotations.

https://developer.android.com/topic/libraries/architecture/viewmodel.html ”


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

```
fun onCreate(savedInstanceState: Bundle) {
// Create a ViewModel the first time the system calls an activity's onCreate
// Re-created activities receive the same MyViewModel instance
    val viewModel = ViewModelProviders.of(this).get(MyViewModel::class.java)
}
```

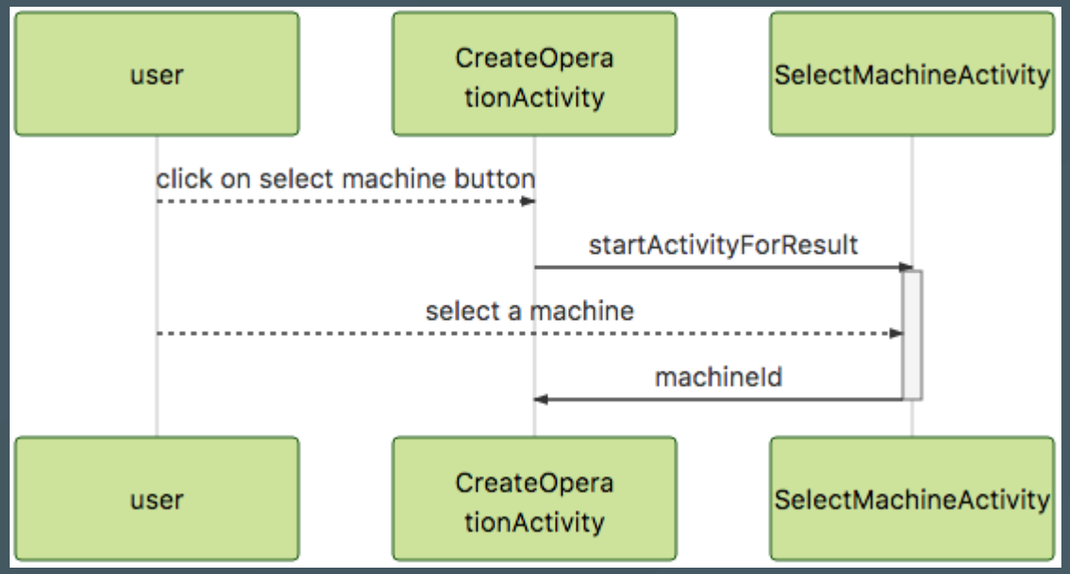
1. Native solutions

Activity + result code

“ Getting a Result from an Activity ”

Example

- First activity: fill a form to create a new operation
- Second activity: select a capable machine



```
sequenceDiagram
  user-->>CreateOperationActivity: click on select machine button
    CreateOperationActivity->>SelectMachineActivity: startActivityForResult
activate SelectMachineActivity
user-->>SelectMachineActivity: select a machine
    SelectMachineActivity->>CreateOperationActivity: machineId
deactivate SelectMachineActivity
```

- With the help of Anko

CreateOperationActivity.kt

```
findViewById<Button>(R.id.button_select_machine).setOnClickListener {
    startActivityForResult<SelectMachineActivity>(
        SELECT_MACHINE,
        SelectMachineActivity.CAPABILITY to typeOfOperation
    )
}
```

SelectMachineActivity.kt

```
class SelectMachineActivity : AppCompatActivity() {

    findViewById<Button>(R.id.select_machine_a).setOnClickListener {
        val intent = Intent()
        intent.putExtra(MACHINE_ID, 1L)
        setResult(
            Activity.RESULT_OK,
            intent
        )
        finish()
    }

    companion object Params {
        val CAPABILITY = "SelectMachineActivity:capability"
        val MACHINE_ID = "SelectMachineActivity:machineId"
    }
}
```

CreateOperationActivity.kt

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?
    if (requestCode == SELECT_MACHINE) {
        if (resultCode == Activity.RESULT_OK) {
            selectedMachineId = data?.getLongExtra(SelectMachineActivity.MACHI
        }
    } else {
        super.onActivityResult(requestCode, resultCode, data)
    }
}
```

Activity + result code: assessments

- pros:
  - stable
  - many libraries written this way
- cons:
  - not composable (1 activity per screen)
  - break the code flow (but rx to the rescue)

Fragment + callbacks

“ Communicating with Other Fragments ”

The embedded `Fragment` defines a callback interface

```
class SelectMachineFragment : Fragment() {
    interface OnFragmentInteractionListener {
        fun onSelectedMachine(selectedMachineId: Long)
    }
}
```

The `Activity` must implement this callback

```
class CreateOperationActivity :
        AppCompatActivity(),
        SelectMachineFragment.OnFragmentInteractionListener {

    override fun onSelectedMachine(selectedMachineId: Long) {
        this.selectedMachineId = selectedMachineId
    }
}
```

The `Fragment` handles a reference to its callback

```
class SelectMachineFragment : Fragment() {
    private var listener: OnFragmentInteractionListener? = null

    override fun onAttach(context: Context) {
        super.onAttach(context)
        if (context is OnFragmentInteractionListener) {
            listener = context
        } else {
            throw RuntimeException(context.toString() +
                    " must implement OnFragmentInteractionListener")
        }
    }

    override fun onDetach() {
        super.onDetach()
        listener = null
    }
}
```

The `Fragment` uses the callback interface to deliver the event to the parent activity

```
override fun onCreateView(inflater: LayoutInflater,
                          container: ViewGroup?,
                          savedInstanceState: Bundle?): View? {
    val view = inflater.inflate(
        R.layout.fragment_select_machine,
        container,
        false
    )
    view.findViewById<Button>(R.id.select_machine_a).setOnClickListener {
        listener?.onSelectedMachine(1L)
    }
    return view
}
```

The `Activity` can deliver a message to another `Fragment`

```
class AnotherFragment : Fragment() {
    fun updateUi(selectedMachineId: Long) {
        TODO("update UI with selectedMachineId")
    }
}
```

```
class CreateOperationActivity :
        AppCompatActivity(),
        SelectMachineFragment.OnFragmentInteractionListener {

    override fun onSelectedMachine(selectedMachineId: Long) {

        val anotherFragment = supportFragmentManager.findFragmentById(
            R.id.another_fragment_container_id
        ) as AnotherFragment

        if (anotherFragment == null) {
            anotherFragment.updateUi(selectedMachineId)
        } else {
            TODO("create and display AnotherFragment with selectedMachineId")
        }
    }
}
```

Fragment + callbacks: assessments

- pros:
  - composable
  - now compatible with the ACC ViewModel
- cons:
  - boilerplate code
  - no compile-time checking

Gradle product flavors

“ Configure Build Variants ”

```
productFlavors {
    application1 {
        applicationId "fr.romain.application1"
    }
    application2 {
        applicationId "fr.romain.application2"
    }
}
```

Gradle product flavors

- Pros:
  - few code
  - easy to extend styles
- Cons:
  - one single project
  - applications must be very similar

Native solutions: assessments

- pros:
  - native solutions are possible
- cons:
  - difficult to setup
  - difficult to compose
  - no navigation concerns

2. Introduce a finite state machine (FSM)

Foreword: key concepts

- states, events, etc.
- application as a FSM

Setup with Android components

- `Fragment` to define a state of the application (i.e., a use case) and output event(s)
- `Activity` to manage states and how to navigate (i.e., the flow of events to change application state)

Specific cases

Orientation changes

- Use of the ACC `ViewModel`
  - Define and share a specific `ViewModel` between `Fragment`s

Dependency injection

- The hard case of Dagger 2
  - Pros: code generation, hosted by Google
  - Cons: many concepts to know and huge amount of code to write
- A nice way with **Koin**

Final MVVM architecture

- AAC
- Data Binding Library

Conclusion

Benefits

- relevant MVVM architecture
- power of the Kotlin language
- an elegant way to define the application flow
- no explicit coupling between screens
- increase testability
  - test at module level
  - test at application level
- adjustable to technical stack

Main used Kotlin concepts

- Extensions (functions, properties)
- Object declarations
- Delegated Properties
- Data classes
- Default and named arguments

What's next?

Practical

- Syntax enhancement thanks to Kotlin
- Group redundant concerns in Java/Android libraries
- Expose features through a repository

What's next?

Ideal

- Front-end with drag&drop feature to build application flow?
- Kotlin: build iOS application and share common modules?
- React-native: write and share common modules (mobile and desktop)?

Thanks

- Macoscope for many relevant articles
  - Applications as State Machines
  - Introducing SwiftyStateMachine
- Nicolas Chassagneux for many enriching discussions