Kotlin Ecosystems

Author: Daria Shutina

23-02-01

Org stuff

Homeworks

If homework is given at date DATE, the soft deadline is DATE + 2 weeks and the hard deadline is at DATE + 3 weeks.

Some homeworks will be connected to each other, so it is bad to skip one of them.

Homeworks will appear after lectures. `git push` to a separate branch, then open a PR.

Grade

To pass the course you have to score ≥ 45 points.

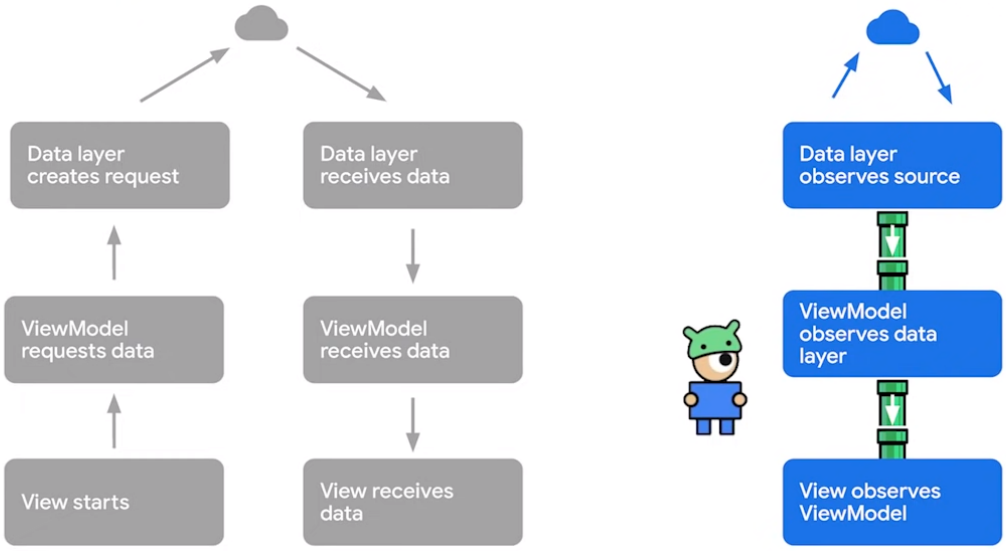
To get A+ you have to score ≥ 95 points.

There are 130 points you can score in this course: 74 for home assignments, 44 for quizzes, 12 for the test.

Cold flows

Main idea

Ordinary threads work in the way that, first, the user sends the request, then they get the answer from the server. Kotlin flows simplify this scheme for the user. Data is "preloaded" to the user.



This type of flows is called "cold", because they are created on demand and emit data only when they are being observed.

Flows should be used in a Coroutine Scope.

Flow builder

```
class UserMessageDataSource(
    private val messageApi: MessagesApi,
    private val refreshIntervalMs: Long = 5000
) {
    val latestMessages: Flow<List<Message>> = flow {
        while(true) { // in order to infinitely fetch messages
            val userMessages = messageApi.fetchLatestMessages()
            emit(userMessages) // emit the result to the flow
            delay(refreshIntervalMs) // wait
        }
    }
}
```

The code inside `while(true)` cycle is called *the producer block*.

Modifying flows

`flow.map`, `flow.filter` and `flow.catch`

```
val userMessages: Flow<MessagesUiModel> =
    UserMessageDataSource.latestMessages
        .map { it. toUiModel() }
        .filter { it.containsImportantNotifications() }
        .catch { e ->
            analytics.log("Error loading reserved event")
            if (e is IllegalArgumentException) throw e
            else emit(emptyList())
        }
```

Observing flows

`flow.collect`

```
userMessages.collect { listAdapter.submitList(it) }
```

Every time `flow.collect` is called on `userMessages`, a new flow will be created. And its producer block will start refreshing messages from the API at its own interval (ничево не понятно)

23-02-08

Gradle

Settings

If there are several modules in a project, each module should have its own settings. The settings are saved in a directory `build.gradle.hts`.

Data in `settings.gradle.kts` will be executed before other settings.

To add a module, use `Project -> New -> Module`.

Tasks

There are default and custom tasks, tasks from plugins. Custom tasks can be defined in the build configuration.

```
task.register("name") {
    group = "useless"
    dependsOn(tasks.named("othername"))
    println("${this.name}, configuration")
    doFirst {
        println("${this.name}, first in execution")
    }
}

task.register("othername") {
    println("${this.name}, configuration")
    doFirst {
        println("${this.name}, first in execution")
    }
    doLast {
        println("${this.name}, last in execution")
    }
}
```

`dependsOn` means that tasks with the name `othername` will be executed before the `name` tasks.

You can also play with tasks:

```
tasks
    .filter { task ->
        task.group?. let { it == "useless" } ?: false
    }
    .forEach {
        it.dependsOn(task3)
    }
```

Throwback

`./gradlew build` will use all components. Use a flag `-x test` to exclude tests, for example.

Plugins

Most useful features are added by plugins.

There are binary plugins and script plugins. If you set up a plugin correctly, you will see additional tasks.

Applying a community plugin:

```
plugins {
    kotlin("jvm") version "1.8.0"
    id("io.gitlab.arturbosch.detekt") version "1.21.0" apply false
}
```

`apply false` means the plugin will not be added in the beginning of the project build. By default, the value is `true`.