УДК 004.421

DOI 10.31651/2076-5886-2019-2-96-103

УЖВА Дмитро Геннадійович
магістрант спеціальності «Прикладна математика» Черкаського національного університету імені Богдана Хмельницького
e-mail: kraativ@gmail.com
ORCID 0000-0003-0073-3928

МЕТОД ОРГАНІЗАЦІЇ КРОСПЛАТФОРМЕННОГО ПРОЕКТУ

У статті розглядається метод створення кроссплатформного проекту для подальшого створення програм для різних платформ з використанням спільного програмного коду. Велика кількість різноманітних платформ та операційних систем вимагають створення нових методів та технологій розробки коду. Кроссплатформні рішення дозволяють швидше та менш затратно отримати готовий продукт, що буде доступний користувачам з різними цільовими машинами, що є важливим фактором для програм та ігор. Використовується середовище програмування IntelliJ IDEA, система автоматичного збирання Gradle та мова програмування Kotlin, а також середовище XCode для запуску на платформі iOS.

Ключові слова: *кроссплатформне програмування, kotlin, gradle.*

Вступ. У наш час існує багато платформ для різних пристроїв, наслідком чого є створення нових методів та технологій створення коду, який можна використати на багатьох платформах, що значно прискорює розробку під велику кількість аудиторії та скорочує витрати.

Метою статті є ознайомлення з одним з підходів до створенням кросплатформного проекту та перевикористанням коду, використовуючи мову програмування Kotlin та технологій Kotlin Native, Kotlin Multiplatform.

Виклад основного матеріалу

Для створення проекту використовується програмне середовище IntelliJ IDEA, що має вбудовану підтримку мови програмування Kotlin. Це середовище дозволяє автоматично створювати один з запропонованих варіантів кросплатформних “helloworld” проектів, що відрізняються платформами та призначенням. Такі проекти використовують систему автоматичного збирання Gradle, що має спеціальні плагіни для мови програмування Kotlin, технологій кросплатформного програмування Kotlin Multiplatform та нативної компіляції Kotlin Native.

Кросплатформне рішення за допомогою технологій Kotlin Multiplatform та Kotlin Native дозволяє зробити значну частину коду, у спільній частині, залишивши для реалізації на кожній платформі окремо специфічних частин. Прикладами задач, які потребують специфічних рішень є інтерфейси мобільних додатків для Android та iOS, де використовуються різні компоненти, або робота з графікою OpenGL, Direct3D.

Для створення потрібного Gradle проекту були розглянуті та розібрані стандартні проекти, що створює програмне середовище IntelliJ IDEA, серед них:

- Multiplatform Library – проект, що дозволяє перевикористовувати код котліна між трьома основними платформами: JVM, JS і Native;
- Mobile Shared Library – проект, що дозволяє будувати кросплатформну бібліотеку для мобільних платформ Android і iOS
- Mobile Android/iOS - проект, що дозволяє створювати додатки, перевикористовуючи код котліна між мобільними платформами Android і iOS;

Після дослідження проектів, що пропонує IntelliJ IDEA було створено проект, що містить два кросплатформних модулі library та app - бібліотеку та додаток, що її використовує. Кожен з цих модулів має під модулі для кожної платформи, а також модуль зі спільним кодом. Загальна структура проекту подана на рис. 1.

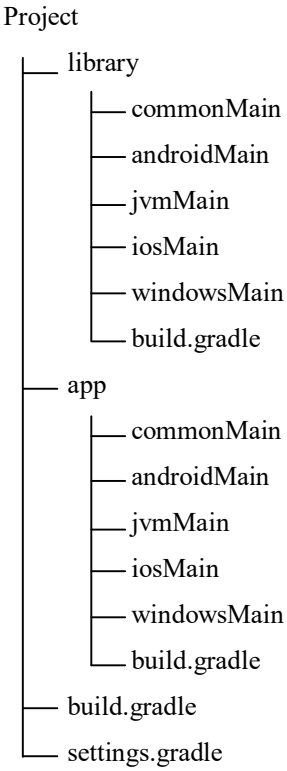

Рис. 1

Кореневий файл build.gradle містить репозиторії бібліотек та залежності інструментів та бібліотек, а settings.gradle назви модулів, ці файли використовують стандартні рішення при створенні Gradle проектів.

Файли build.gradle модулів core та run використовують плагін Kotlin Multiplatform. Плагін дозволяє розділяти налаштування та залежності для кожної платформи в замиканні(Closure) kotlin. Використовуючи замикання kotlin {} потрібно вказати цільові платформи та назви модулів, а також додати залежності бібліотек Kotlin та довільні потрібні виристовуючи замикання kotlin.sourseSets.

За замовчуванням Kotlin Multiplatform має модуль для спільного коду, а у замиканні kotlin.sourseSets вказана залежність від основної бібліотеки Kotlin:

```
[app,library]/build.gradle - Groove code
apply plugin: "org.jetbrains.kotlin.multiplatform"

kotlin {
    sourceSets {
        commonMain {
            dependencies {
                implementation kotlin("stdlib-common")
            }
        }
    }
}
```

Початкове налаштування скриптів build.gradle додатку та бібліотеки відрізняється лише залежністю додатка від бібліотеки, тому у файлі app/build.gradle потрібно вказувати залежність від бібліотеки для кожної платформи:

```
  dependencies {
        implementation kotlin("stdlib-common")
        implementation project(":library")
  }
```

Для додання наступної платформи необхідно додати потрібну платформу в замиканні kotlin та вказати залежності від основних бібліотек котліна, для того щоб мати доступ до реалізації основних функцій на конкретній платформі. Важливо підмітити те, що бібліотеки для конкретної платформи можуть мати різні реалізації, наприклад для платформи JVM використовують «Kotlin Standard Library», «Kotlin Standard Library JDK 7» або «Kotlin Standard Library JDK 8», останні з них відрізняються додатковими функціями та змінами для підтримки та сумісності з цільовими платформами.

Додамо платформу JVM та залежність Kotlin Standard Library JDK 8 до проекту:

```
[app,library]/build.gradle - Groove code
kotlin {
    jvm()

    sourceSets {
        commonMain {
            dependencies {
                implementation kotlin("stdlib-common")
            }
        }
        jvmMain {
            dependencies {
                implementation kotlin("stdlib-jdk8")
            }
        }
    }
}
```

Для написання JVM проектів зазвичай використовуються залежності у вигляді JAR архівів, що містять скомпільовані класи, тому для того щоб наша програма після збірки мала можливість запускатись без додаткових файлів, потрібно налаштувати Gradle додавати бібліотеки до архіву вихідної програми. Також для запуску JAR архіву як програми потрібно вказати головний клас. Виконаємо ці кроки у файлі build.gradle для додатку:

```
app/build.gradle - Groove code
kotlin {
    jvm()

    sourceSets {
        . . .
   }

    configure(jvm()) {
        jvmJar {
            withJava()
            manifest {
                attributes "Main-Class": "com.example.app.Main"
```

```
            }
            from {
                configurations.compileClasspath.collect {
                    it.isDirectory() ? it : zipTree(it)
                }
            }
        }
    }
}
```

Нативні платформи можуть компілюватися під різні архітектури процесора, що вказується у назві платформи, наприклад mingwX64, mingwX86, iosArm32, iosArm64, iosX64. Для зміни назви для розуміння чи розмежування потрібно написати нову назву в дужках. Додамо платформи iOS та MinGW до проекту:

```
[app,library]/build.gradle - Groove code
kotlin {
    iosX64("ios")
    jvm()
    mingwX64("windows")

    sourceSets {
        commonMain {
            dependencies {
                implementation kotlin("stdlib-common")
            }
        }
        iosMain {
            dependencies {
                implementation kotlin("stdlib")
            }
        }
        jvmMain {
            dependencies {
                implementation kotlin("stdlib-jdk8")
            }
        }
        windowsMain {
            dependencies {
                implementation kotlin("stdlib")
            }
        }
    }
}
```

Програма для Windows готова до запуску, в той час як додатки для iOS можна компілювати лише на платформі MacOS та за допомогою середовища XCode. Проте можна налаштувати iOS проект при збірці запускати Gradle задачу. Потрібно скомпілювати iOS фреймворк та скопіювати його до iOS проекту, після чого він буде використовуватись у додатку. Додамо задачу для цього в кінці файлу build.gradle для бібліотеки:

```
library/build.gradle - Groove code
    task buildIOSFramework {
   def buildType = project.findProperty("kotlin.build.type") ?: "DEBUG"
   def target = project.findProperty("kotlin.target") ?: "ios"
   def framework = kotlin.targets."$target"
       .binaries.getFramework(buildType)
```

```
    dependsOn framework.linkTask
    doLast {
        def srcFile = framework.outputFile
        def targetDir = getProperty("configuration.build.dir")
        copy {
            from srcFile.parent
            into targetDir
            include "app.framework/**"
            include "app.framework.dSYM"
        }
    }
}
```

У середовищі XCode потрібно створити новий або відкрити вже існуючий проект, перейти до етапів збірки(Build Phases), та створити новий скріпт, що буде запускати Gradle task:

```
Run Script - Command line code
    # move to the xcode-frameworks folder
    cd "$SRCROOT/../shared/build/xcode-frameworks"

    # run the gradle task
    ./gradlew :library: buildIOSFramework -
PXCODE_CONFIGURATION=${CONFIGURATION}
```

Тепер потрібно впевнитися, що цей етап збірки виконується раніше ніж компілюються вихідні файли проекту середовища XCode, після чого додаток готовий до запуску.

Перш ніж додавати платформу Android до плагіну Kotlin Multiplatform, необхідно підключити та налаштувати плагіни Android для додатку та для бібліотеки. В даному випадку їх налаштування відрізняється лише відсутністю ідентифікатора додатка у бібліотеки. Додамо підключення та налаштування плагіну перед налаштуванням плагіну Kotlin Multiplatform, вказуючи потрібні версії бібліотек та додатку:

```
[app,library]/build.gradle - Groove code
    apply plugin: "org.jetbrains.kotlin.multiplatform"
    // Для додатку та бібліотеки використовуємо різні плагіни
    // apply plugin: "com.android.application"
    apply plugin: "com.android.library"

    android {
        compileSdkVersion 29

        defaultConfig {
            // Для додатку потрібно вказати ідентифікатор
            // applicationId "com.example.app"
            minSdkVersion 14
            targetSdkVersion 29
            versionCode 1
            versionName "1.0"
        }
    }

    kotlin {
        . . .
    }
```

Додаємо платформу Android до плагіну Kotlin Multiplatform:

```
[app,library]/build.gradle - Groove code
kotlin {
    android()
    iosX64("ios")
    jvm()
    mingwX64("windows")

    sourceSets {
        commonMain {
            dependencies {
                implementation kotlin("stdlib-common")
            }
        }
        androidMain {
            dependencies {
                implementation kotlin("stdlib")
            }
        }
        iosMain {
            dependencies {
                implementation kotlin("stdlib")
            }
        }
        jvmMain {
            dependencies {
                implementation kotlin("stdlib-jdk8")
            }
        }
        windowsMain {
            dependencies {
                implementation kotlin("stdlib")
            }
        }
    }
}
```

Плагіни налаштовані та готові до збірки проекта. Для перевірки результату напишемо спільний код, що використовуватиме код, що знаходитиметься в модулях платформ. Спільна частина бібліотеки:

```
library/src/commonMain/kotlin/CommonSample.kt - Kotlin code
expect object Platform {
  val name: String
}

fun hellotext(): String = "Hello from ${Platform.name}!"
```

Платформозалежна частина:

```
library/src/jvmMain/kotlin/ JVMSample.kt - Kotlin code
actual object Platform {
  actual val name: String = "JVM"
}

library/src/androidMain/kotlin/AndroidSample.kt - Kotlin code
actual object Platform {
  actual val name: String = "Android"
```

```
}

library/src/iosMain/kotlin/IOSSample.kt - Kotlin code
actual object Platform {
    actual val name: String = "iOS"
}

library/src/windowsMain/kotlin/WindowsSample.kt - Kotlin code
actual object Platform {
    actual val name: String = "Windows"
}
```

У настільних платформах точкою входу в програму є функція main(), в той час як на мобільних платформах створюються специфічні екрани. Для цього ми створюємо потрібні речі у платформозалежних модулях програми. У випадку з настільними платформами запуск та використання кросплатформної бібліотеки виглядає так:

```
app/src/windowsMain/kotlin/WindowsApp.kt - Kotlin code
    object Main {
        fun main() {
            println(hellotext())
        }
    }
```

У потрібному місці, в даному випадку у функції main, викликається кросплатформна функція hellotext, яка, в залежності від платформи, працює з різними об'єктами Platform. В результаті виклику функції hellotext отримаємо рядок з текстом, частина якого відрізняється на кожній платформі:

```
Output:
JVM: Hello from JVM!
Android: Hello from Android!
iOS: Hello from iOS!
Windows: Hello from Windows!
```

Отримано проект, що дозволяє будувати кросплатформну бібліотеку та додаток, що її використовує. Бібліотека та додаток можуть мати спільну та окрему, платформозалежну частину коду.

Висновки. Мова програмування Kotlin має плагіни Kotlin Multiplatform та Kotlin Native, які дозволяють створити кросплатформний проект, писати спільний код для всіх платформ та специфічний для окремих. Отримані додатки запускаються на популярних платформах Andrroid, JVM, iOS, Windows, тощо.

Список використаної літератури:

1. Building Multiplatform Projects with Gradle - Kotlin Programming Language [Електронний ресурс] – Режим доступу: https://kotlinlang.org/docs/reference/building-mpp-with-gradle.html
2. Building your first Kotlin Multiplatform app—Getting Started [Електронний ресурс] – Режим доступу: https://vivekc.xyz/building-your-first-kotlin-multiplatform-app-getting-started-8ad10d7d4e9f
3. Which standard library to use in Kotlin [Електронний ресурс] – Режим доступу: https://stackoverflow.com/questions/51858596/which-standard-library-to-use-in-kotlin/
4. Configure your build | Android Developers [Електронний ресурс] – Режим доступу: https://developer.android.com/studio/build
5. The Build Process objc.io [Електронний ресурс] – Режим доступу: https://www.objc.io/issues/6-build-tools/build-process/

Bibliography:

1. Building Multiplatform Projects with Gradle - Kotlin Programming Language. Retrieved from: https://kotlinlang.org/docs/reference/building-mpp-with-gradle.html
2. Building your first Kotlin Multiplatform app—Getting Started. Retrieved from: https://vivekc.xyz/building-your-first-kotlin-multiplatform-app-getting-started-8ad10d7d4e9f
3. Which standard library to use in Kotlin. Retrieved from: https://stackoverflow.com/questions/51858596/which-standard-library-to-use-in-kotlin/
4. Configure your build | Android Developers. Retrieved from: https://developer.android.com/studio/build
5. The Build Process objc.io. Retrieved from: https://www.objc.io/issues/6-build-tools/build-process/

UZHVA Dmitry,
student, The Bohdan Khmelnytsky National University of Cherkasy

METHOD OF ORGANIZATION OF THE CROSS-PLATFORM PROJECT

Summary. Introduction. *A large number of different platforms and operating systems require the creation of new code creation techniques and technologies. Cross-platform solutions make it faster and less expensive to get a ready-made product that is accessible to users with different targeting machines, which is an important factor for applications and games.*

There is a way to create and organize a cross-platform project to create programs for different platforms using a common source code that uses the IntelliJ IDEA programming environment, Gradle auto-build, and Kotlin programming language, as well as XCode to run on the iOS platform.

This method allows you to split the settings and dependencies for each platform or use common ones for all, allowing you to write the most effective code for a specific platform and reuse as much as possible.

The purpose *of this paper is to define the method that allows you to create and organize a project for writing cross-platform apps and libraries in Kotlin programming language, to explain how tools and plugins works with cross-platform code and controlling it.*

Results. *We have a project that allows us to write cross-platform library and the application that uses it. The library and application has a code that can be shared and separated between platforms, architectures. Each compilation is accompanied by a default source set that stores sources and dependencies specific to that compilation. A Kotlin source set is a collection of Kotlin sources, along with their resources, dependencies, and language settings, which may take part in Kotlin compilations. some platforms require specific setup, use, or launch, but this is all automated in the process of setting up a project build.*

We also have a simple code for an example of how to write cross-platform code, use a common objects to access platform-specific code and try it.

Conclusion. *The Kotlin programming language has Kotlin Multiplatform and Kotlin Native plugins, which allow you to create a cross-platform project, write common code for all platforms, and specific for individual ones. This programming language and tools also have a powerful programming environment that supports building and debugging that combination of solutions.*

Obtained applications are self-sufficient and can run on the most popular platforms like Android, JVM, iOS, Windows, etc.

Keywords: *cross-platform programming, kotiln, gradle.*

Одержано редакцією 27.06.2019 р.
Прийнято до публікації 09.10.2019 р.