Programació multimèdia
i dispositius mòbils

2. Estructura i execució de programes

IES Jaume II El Just
Tavernes de la Valldigna
Departament d'Informàtica
Curs 2021-22

Continguts

1 Estructura i execució d'un programa: Hello World **3**
1.1 Compilació i execució 3
1.2 Ús d'arguments 4
1.3 Gradle, Java i Kotlin 5
1.3.1 Afegint codi Java 13

1 Estructura i execució d'un programa: Hello World

Recordem l'estructura general d'un programa en Java:

```java
public class hello {
   public static void main(String[] args) {
      System.out.println("Hello, World");
   }
}
```

Recordeu que el nom de la classe ha de ser el mateix que el del fitxer, sense l'extensió java (Fitxer hello.java).

Aquest codi, en Kotlin l'expressaríem així (nomenarem al fitxer `helloworld.kt`):

```kotlin
fun main() {
    println("Hello World")
}
```

Com veiem, les principals diferències són:

- La funció principal (main) es declara com una funció de primer nivell, sense necessitat que estiga dins una classe. Per tant, Kotlin no requereix que tot siguen classes, a diferència de Java, i ens permet treballar com en C o C++, amb funcions.
- Aquesta funció `main` pot rebre arguments, però no és necesasri indicar-ho.
- Tampoc cal indicar la visibilitat de la funció (`public`), ni si és estàtica (`static`).
- La funció `println` la tenim disponible directament, sense passar per `System.out`.
- Els `;` de final de línia són opcionals, i solen evitar-se per convenció

1.1 Compilació i execució

Amb java, des de la línia d'ordres, compilem amb:

```
$ javac hello.java
```

Aquesta ordre ens crea el fitxer en bytecode hello.class, que executem amb:

```
$ java hello
```

De la mateixa manera que amb java, disposem d'una ordre per compilar i altra per executar.

Per compilar el programa, utilitzem l'ordre `kotlinc`:

```
$ kotlinc helloworld.kt
```

Amb açò es genera un fitxer bytecode `HelloworldKt.class`. Fixeu-vos que afig la primera lletra en majúscula, i completa el nom de la classe amb Kt, per tal d'indicar que prové d'una compilació de Kotlin.

Per executar aquest bytecode:

```
$ kotlin HelloworldKt
```

Per comprovar que es tracta de bytecode totalment compatible amb Java, podem llançar-lo amb:

```
$ java HelloworldKt
```

1.2 Ús d'arguments

Recordem que els arguments amb els què s'invoca el programa esw recullen en un vector d'strings:

```java
/*
Fitxer hello.java
*/

public class hello {
    // Mètode principal
    public static void main(String[] args) {
       if (args.length!=0)
        System.out.println("Hello "+args[0]);
       else
        System.out.println("Hello World");
    }
 }
```

En Kotlin, aquest exemple seria:

```kotlin
/*
Fitxer hello.kt
*/

// funció principal, de primer nivell.
fun main(args: Array<String>) {
    if (args.size>1)
        println("Hello "+args[0])
    else
        println("Hello World")
}
```

Com podem veure, la forma de rebre arguments és lleugerament diferent. Als paràmetres per a les funcions en Kotlin expressem primer el nom i després el tipus. En aquest cas, per indicar que es tracta d'un vector d'Strings ho fem amb `Array<String>`.

Per altra banda, en lloc d'utilitzar la propietat `length` utilitzem `size`, i accedim de la mateixa manera al vector. Per altra banda, el primer `println` podria haver-se expressat també de la següent manera:

```kotlin
println("Hello ${args[0]}")
```

Que recorda una sintaxi més semblant a *Bash*, i fa ús del que es coneixen com *expressions de plantilla*, que veurem més avant.

Per taltra banda, a l'exemple, podeu veure un parell de comentaris. Aquests s'expressen de la mateixa manera, tant en Java com en Kotlin:

```kotlin
// Comentari d'una línia
```

```kotlin
/*
Comentaris de més d'una línia
*/
```

1.3 Gradle, Java i Kotlin

El sistema d'automatització de construcció de projectes *Gradle* suporta, a més de Java, projectes en Kotlin, així com projectes on es combinen amdós llenguatges.

Per fer un exemple il·lustratiu, anem a fer un projecte Gradle, de tipus aplicció Kotlin, però que també utilitze codi java. Utilitzarem la versió 7.2 de Gradle, que podeu instal·lar amb *SDKMAN!*:

```
$ sdk install gradle
```

Per crear el projecte de manera senzilla, farem ús de l'ordre `gradle init`, en un directori per al projecte anomenat `helloGradle`, que haurem de crear:

```
$ mkdir helloGradle
$ cd helloGradle
helloGradle$ gradle init

Welcome to Gradle 7.2!

Here are the highlights of this release:
 - Toolchain support for Scala
 - More cache hits when Java source files have platform-specific line endings
 - More resilient remote HTTP build cache behavior

For more details see https://docs.gradle.org/7.2/release-notes.html

Starting a Gradle Daemon (subsequent builds will be faster)

Select type of project to generate:
  1: basic
  2: application
  3: library
  4: Gradle plugin
Enter selection (default: basic) [1..4] 2
```

Com veiem, ens indica la versió de Gradle, i en primer lloc ens demana el tipus de projecte que volem. Indicarem que volem generar una aplicacio `2: applicaction`.

Tot seguit ens demanarà el llenguatge d'implementació. Triarem `4: Kotlin`.

```
Select implementation language:
  1: C++
  2: Groovy
  3: Java
  4: Kotlin
  5: Scala
```

```
  6: Swift
Enter selection (default: Java) [1..6] 4
```

El següent pas ens pregunta si el nostre projecte serà un d'úna sola aplicació o estarà dividit en subprojectes. En cas que un projecte es dividisca en subprojectes, contindrà un fitxer `build.gradle` comú a tots i un fitxer `settings.gradle` amb els subprojectes que conté; ambdós fitxers es trobaran a l'arrel del projecte. A més, tindrem una carpeta per cada subrpojecte, dins la qual tindrem també el fitxer `build.gradle` específic del subprojecte.

Al nostre cas, contestarem que **no**, que volem un projecte amb una única aplicació.

> Teniu més informació relativa a multiprojectes amb Gradle a l'article Multiproyectos con Gradle.

```
Split functionality across multiple subprojects?:
  1: no - only one application project
  2: yes - application and library projects
Enter selection (default: no - only one application project) [1..2] 1
```

I per a l'script de configuració del projecte, tot i que podem utilitzar una notació basada en Kotlin, farem ús de *Groovy*, amb el què ja hem treballat en altres ocasions.

```
Select build script DSL:
  1: Groovy
  2: Kotlin
Enter selection (default: Kotlin) [1..2] 1
```

Ara ens queda per indicar el nom del projecte, on triarem el que ens suggereix per defecte (*helloGradle*, com la carpeta on ens trobemm), i com a paquet font, indicarem `com.ieseljust.dam`:

```
Project name (default: helloGradle):
Source package (default: helloGradle): com.ieseljust.dam

BUILD SUCCESSFUL in 1m 20s
2 actionable tasks: 2 executed
```

Recordeu que `gradle init` també admet paràmetres per configurar el projecte, de manera que ens podem estalviar l'assistent anterior amb:

```
$ gradle init   --type kotlin-application \
                --dsl groovy \
                --project-name helloGradle \
                --package com.ieseljust.dam
```

Si havérem contestat que sí a dividir el projecte en múltiples projectes, hauríem d'haver afegit l'opció `--split-project`.

Amb qualsevol de les dues formes, haurem generat la següent estructura de directoris:

```
.
.
├── app
│   ├── build.gradle
│   └── src
│       ├── main
│       │   ├── kotlin
│       │   │   └── com
│       │   │       └── ieseljust
│       │   │           └── dam
│       │   │               └── App.kt
│       │   └── resources
│       └── test
│           ├── kotlin
│           │   └── com
│           │       └── ieseljust
│           │           └── dam
│           │               └── AppTest.kt
│           └── resources
├── gradle
│   └── wrapper
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradlew
├── gradlew.bat
└── settings.gradle
```

Com a fitxers i carpetes interessants tenim el fitxer `app/build.gradle`, amb la configuració del projecte, i la carpeta `src`. Fixeu-vos que ara, dins la carpeta `src/main`, en lloc del directori `java`

tenim el directori `kotlin`, amb l'estructura de directoris determinada pel nom del paquet que li hem dit (`com/ieseljust/dam`).

Ens queda per veure el contingut del fitxer de configuració `app/build.gradle`. Aquest fitxer, en versions anteriors es trobava a l'arrel, però ara, en projectes d'una única aplicació, s'emmagatzema en la carpeta app.

```
/*
 * This file was generated by the Gradle 'init' task.
 *
 * This generated file contains a sample Kotlin application project to get
 ↪  you started.
 * For more details take a look at the 'Building Java & JVM projects'
 ↪  chapter in the Gradle
 * User Manual available at
 ↪  https://docs.gradle.org/7.2/userguide/building_java_projects.html
 */

plugins {
    // Apply the org.jetbrains.kotlin.jvm Plugin to add support for Kotlin.
    id 'org.jetbrains.kotlin.jvm' version '1.5.0'

    // Apply the application plugin to add support for building a CLI
     ↪  application in Java.
    id 'application'
}

repositories {
    // Use Maven Central for resolving dependencies.
    mavenCentral()
}

dependencies {
    // Align versions of all Kotlin components
    implementation platform('org.jetbrains.kotlin:kotlin-bom')

    // Use the Kotlin JDK 8 standard library.
    implementation 'org.jetbrains.kotlin:kotlin-stdlib-jdk8'

    // This dependency is used by the application.
    implementation 'com.google.guava:guava:30.1.1-jre'

    // Use the Kotlin test library.
```

```
    testImplementation 'org.jetbrains.kotlin:kotlin-test'

    // Use the Kotlin JUnit integration.
    testImplementation 'org.jetbrains.kotlin:kotlin-test-junit'
}

application {
    // Define the main class for the application.
    mainClass = 'com.ieseljust.dam.AppKt'
}
```

Revisem les seccions de què consta aquest fitxer:

- **plugins**, on s'inclouen els plugins de suport per a Kotlin a la JVM i per generar una aplicació de tipus CLI (`application`),
- **repositories**, on s'afig directament el repositori de `mavenCentral()`, igual que als projectes amb Java,
- **depencencies**, on s'inclouen algunes dependències necessàries per a Kotlin i els seus tests,
- **application**, on s'indica el nom, completament qualificat (amb el prefix del paquet), de **la classe que llançarà l'aplicació: `AppKt`**.

Veiem ara el contingut del fitxer font `App.kt`:

```
/*
 * This Kotlin source file was generated by the Gradle 'init' task.
 */
package com.ieseljust.dam

class App {
    val greeting: String
        get() {
            return "Hello World!"
        }
}

fun main() {
    println(App().greeting)
}
```

Com veiem, presenta una sintaxi molt semblant a Java, però amb algunes peculiaritats a destacar:

- S'ha definit la classe App, amb un atribut `greeting`, de tipus *String*, i que és definit com un *atribut immutable*. Un *atribut immutable* seria equiparable a una constant en Java, i es defineix amb `val`, mentre que els atributs mutables es definiran amb `var`. A més, hi ha un mètode `get()`, definit a continuació d'aquest atribut, i que és el mètode accessor, per a lectura d'aquest atribut. Com que es tracta d'un atribut només de lectura, només té el mètode `get` i no `set`.
- Es defineix la funció `main`, com a funció de primer nivell, pel que no està definida dins de cap classe. Aquesta funció el que fa és escriure el valor de l'atribut `greeting` d'un objecte de la classe App. Veiem alguns detalls d'esta línia que ens poden resultar curiosos:
  - No hem utilitzat l'operador new per declarar un objecte de tipus App, sinò que hem utilitzat el nom de la classe com si fos una funció. **En Kotlin, l'operador new no existeix.**
  - **Accedim al valor de `greeting` com si estiguérem accedint directament a l'atribut**. Tot i que no ho indiquem explícitament, el que està fent Kotlin realment és invocar al mètode `get` associat a aquest atribut.
- Per altra banda, recordem que la classe principal que hem indicat al fitxer `build.gradle` és `com.ieseljust.dam.AppKt`. Aquest fitxer `com/ieseljust/dam/AppKt.class` serà el resultat de la compilació d'aquest fitxer `App.kt`. Però... si la funció main no està dins de cap classe, per què s'indica com a classe principal? Aprofundirem més avant en açò, però de moment, avancem que Kotlin, en fer la compilació per a la JVM, generarà automàticament la classe per nosaltres, per tal que el fitxer *bytecode* siga compatible totalment amb la JVM.

Ara només ens quedaria veure el resultat de la compilació i execució. Ens situem en l'arrel del projecte, i llancem:

```
$ gradle build
```

La primera vegada que el llancem, veurem que tarda una miqueta a descarregar totes les llibreríes necessàries.

Si donem una ullada a l'arbre del projecte (`tree`) veurem que genera moltes més coses que quan treballàvem amb Java, ja que necessita tot l'entorn per treballar amb Kotlin des de Gradle. Anem a centrar-nos en part de la carpeta `app/build`:

```
app/build
.
├── app
│   ├── build
│   │   ├── classes
│   │   │   └── kotlin
```

```
.   │   │       ├── main
.   .   │       │   ├── com
    .   .       │   │   └── ieseljust
        .       .   │       └── dam
                .   .           ├── App.class
                    .           └── AppKt.class
```

Aci veiem que s'han generat dues classes: la classe App, que és exactament la classe que hem generat creat al fitxer `App.kt`, i la classe `AppKt.class`, que és la que conté el mètode `main` i si es fixeu, la que es llança des del `build.gradle` (`com.ieseljust.dam.AppKt`). Com hem comentat, és una classe que genera Kotlin automàticament, amb el nom del fitxer sense extensió *App* més la cadena *Kt*.

Ara llancem l'aplicació i veurem que ens mostra el misstge esperat:

```
$ gradle run

> Task :run
Hello world.
```

> **gradle vs gradlew**
>
> Els fitxers gradlew, gradle.bat i la carpeta gradle/wrapper fan referència al *wrapper* o *envoltori* de Gradle, i no é smés que una instal·lació portable d'aquest dins del projecte, de manera que no siga necessari tindre instal·lat Gradle a l'equip per treballar amb el projecte. A més, també té l'avantatge que ens aporta una versió fixa de Gradle per al projecte, amb la qual cosa, no tindrem incompatibilitats entre versions.
>
> Per utilitzar aquest wrapper, com que es tracta d'un fitxer que no és al PATH, caldría invocar-lo de forma local amb:
>
> ```
> ./gradlew build
> ./gradlew run
> ```
>
> Comprovarem que la primera vegada que l'invoquem es descarrega la versió portable corresponent.
>
> *¿Quan convé utilitzar un o altre?*
>
> Per crear projecte i per generar nous *wrappers* (`gradle wrapper`), utilitzarem l'ordre `gradle`, mentr que per a la construcció i execució és més recomanable el wrapper.

1.3.1 Afegint codi Java

Anem a veure ara com afegiríem codi Java a la nostra aplicació. Per a això, anem a eliminar la classe *App* del fitxer *App.kt*, i la definirem en un fitxer Java.

En primer lloc, crearíem el directori `java` dins la carpeta `src/main`, amb l'estructura de directoris del paquet:

```
$ mkdir -p app/src/main/java/com/ieseljust/dam
```

I creem el fitxer `app/src/main/java/com/ieseljust/dam/App.java` amb:

```java
package com.ieseljust.dam;

public class App {
  public String greeting = "Hola Kotlin, des de Java";

  public String getGreeting() {
    return this.greeting;
  }
}
```

I modifiquem el fitxer `App.kt` amb:

```kotlin
package com.ieseljust.dam

fun main() {
    println(App().getGreeting());
}
```

Com que es tracta del mateix `package`, no hem d'importar la classe `App`.

L'estructura de fonts quedaria:

```
app/src/main/
├── java
│   └── com
│       └── ieseljust
│           └── dam
│               └── App.java
```

```
├── kotlin
│   └── com
│       └── ieseljust
│           └── dam
│               └── App.kt
└── resources
```

Com veiem, dupliquem l'estructura de directoris per a Java i per a Kotlin, duplicitat que també es vorà en les classes que generem en construir el paquet:

```
app/build/classes/
├── java
│   └── main
│       └── com
│           └── ieseljust
│               └── dam
│                   └── App.class
└── kotlin
    ├── main
    │   ├── com
    │   │   └── ieseljust
    │   │       └── dam
    │   │           └── AppKt.class
    │   └── META-INF
    │       └── app.kotlin_module
    └── test
        ├── com
        │   └── ieseljust
        │       └── dam
        │           └── AppTest.class
        └── META-INF
            └── app.kotlin_module
```

I l'execució seria l'esperada:

```
 ./gradlew run

> Task :app:run
Hola Kotlin, des de Java
```