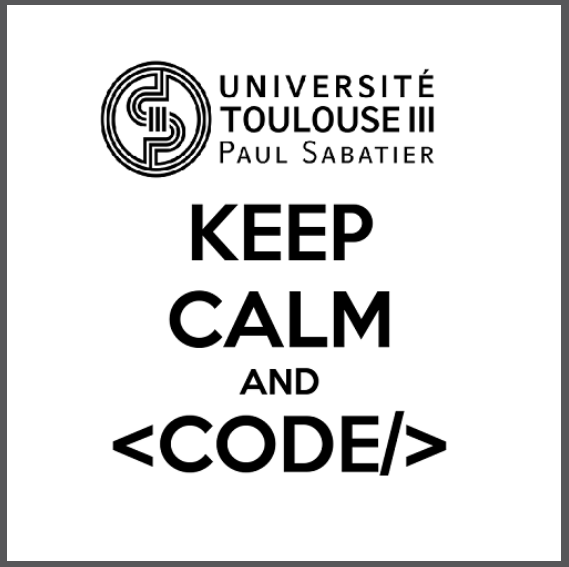


Programmation orientée objet – Application en Kotlin

Présentation du langage Kotlin : langage et premiers programmes

Sylvie TROUILHET - www.irit.fr/~Sylvie.Trouilhet

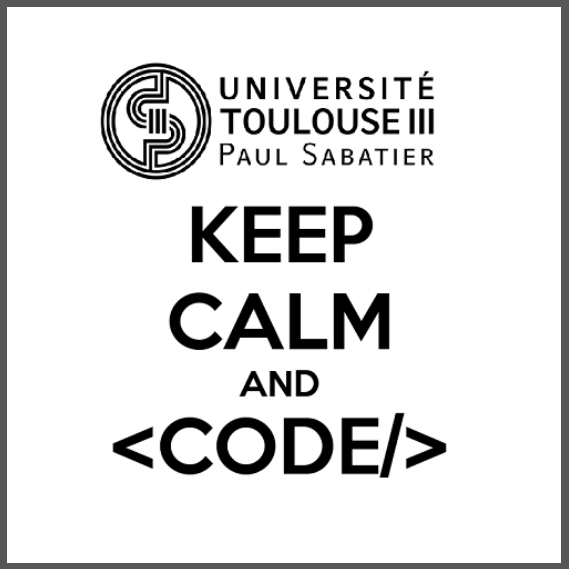
Présentation du langage

Créé par une équipe de JetBrains
Première version stable : 2016
Compilation en byte-code Java

Site de référence : **https://kotlinlang.org**

Écriture plus concise (par exemple, pas de ; en fin d'instruction)
Minimise les risques d'erreur d'exécution (par exemple les « null pointer exception »)

Version actuelle : 1. 7 (2022)



Programmation orientée objet – Application en Kotlin

PLAN

Partie 1 : syntaxe de base

Partie 1 : les variables

Types simples
Opérateur de comparaison
Tableaux
Opérateurs particuliers

Déclaration de variable

`val` : variable ne pouvant être affectée qu'une fois

```
val v1:Int
v1=12

val v2:Int=2
```

`var` : variable « mutable »

Déclaration de constante

`const` : à déclarer en dehors de toute fonction

```
const val NB_MOIS:Int=12
```

Types numériques

Représentation des nombres :

Type	Size (bits)	Min value	Max value
Byte	8	-128	127
Short	16	-32768	32767
Int	32	-2,147,483,648 (-2^{31})	2,147,483,647 ($2^{31} - 1$)
Long	64	-9,223,372,036,854,775,808 (-2^{63})	9,223,372,036,854,775,807 ($2^{63} - 1$)

Type	Size (bits)	Significant bits	Exponent bits	Decimal digits
Float	32	24	8	6-7
Double	64	53	11	15-16

Pas de conversion implicite entre types numériques :

`nb.toInt()` `nb.toDouble()` …

Booléen

Représentation des booléens :

`Boolean`
`true false`
Opérateurs logiques : `&&` `||` `!`

Types symboliques : caractères

Représentation des caractères :

```
Char
'A'       '\u0041'  (unicode)
```

Types symboliques : chaînes de caractères

Représentation des chaînes :

```
String
"A"          "Bonjour"
```

Template- le mot-clé $

```
var v1:Int=12
val s1:String= " a vaut ${v1} ou $v1 "
```

Longueur d'une chaîne : `s1.length`

Types symboliques : chaînes de caractères

Méthodes de la bibliothèque
`https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-string/`

```
replace()
split()
toUperCase() …
```

Les objets de cette classe sont constants ; chaque traitement qui modifie une chaîne laisse l'original inchangé et renvoie un nouvel objet modifié

Opérateurs de comparaison

```
< > <= >=

== !=
```

Tableaux

Méthodes de la bibliothèque
`https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-array/`

Création d'un tableau de taille fixe:

```
val t1:Array<Int>
t1=arrayOf(1,2,3)
```

```
val t1:Array<Int>
t1=arrayOf(1,2,3)
```

Création d'un tableau vide de taille fixe:

```
val t2:Array<Int?>
t2=arrayOfNulls(3)
```

Taille d'un tableau : `size` Copie : `source.copyOf(taille)`

Objet : marqueur « nullable »

Par défaut, la référence à un objet ne peut valoir null

```
var i: Int = null
```

Il faut utiliser le marqueur ? :

```
var i : Int?
```

À éviter dès que c'est possible

Partie 1 : structures de contrôle

Sélection
Répétition

Structures de contrôle : sélection

```
if (v1>0) {
 …
}
else {
 …
}
```

Imbrications possibles

Expression conditionnelle :

```
val type:String= if (v1>0) "+" else "-"
```

Structures de contrôle : sélection

Choix multiples :

```
when (v1) {
    1 -> print("1")
    2 -> print("2")
    else -> print("autre")
}
```

Structures de contrôle : répétition

while et do-while:

```
while (condition) {
    …
}
```

```
do {
    …
}
while (condition)
```

Structures de contrôle : répétition

For in :

```
for (i in 0..6) {
    …}
for (i in 0..6 step 2) {
    …}

for (i in 0 until 6) {
    …}

for (i in 6 downTo 0) {
    …}
```

i>=0 et i<=6

i>=0 et i<6

Partie 1 : les fonctions

main()
Paquetages
Fonction d’entrée/sortie

Déclaration

Mot-clé fun

```
fun somme(p1:Int, p2:Int):Int{
return p1+p2
}
```

Fonction ne retournant rien : type `Unit`

```
fun afficher(p1:Int, p2:Int):Unit{
    print("$p1 et $p2 donne ${somme(p1,p2)}")
}
```

NB : on peut ne pas préciser Unit

Valeur par défaut pour un paramètre

```
fun somme(p1:Int, p2:Int=0):Int{
return p1+p2
}
```

L'appel somme(5) retourne 5

Nombre de paramètres variable

```
fun somme(vararg p:Int):Int{
…
}
```

somme(5,12,9) somme(5,12,9, 13, 17)

Paramètres nommés

```
fun construireVoiture(marque:String,
couleur:String="noir", immatriculation:Int=0,
puissance:Int=4):Unit {

}
```

Appel : `construireVoiture(marque ="panhard", immatriculation=45)`

Fonction à 1 instruction

```
fun identique(p1: Int, p2: Int): Boolean {
return p1==p2
}

fun identique2(p1: Int, p2: Int): Boolean =
p1==p2
```

Point d’entrée du programme

Une fonction de nom main

```
fun main() {
println("Hello DReAM")
}
```

Paquetage et import de librairies

La définition du paquetage se met en haut du fichier
Elle n’est pas obligatoire

```
package monProjet
```

Utilisation d’un paquetage :

```
import kotlin.math.*
```

Lecture / Affichage

Pas d'utilité pour les applications en production

Lecture au clavier :

```
import java.util.*
val reader = Scanner(System.`in`)
val num = reader.nextInt()
```

Affichage à l'écran : fonctions `print()` et `println()`

Commentaires

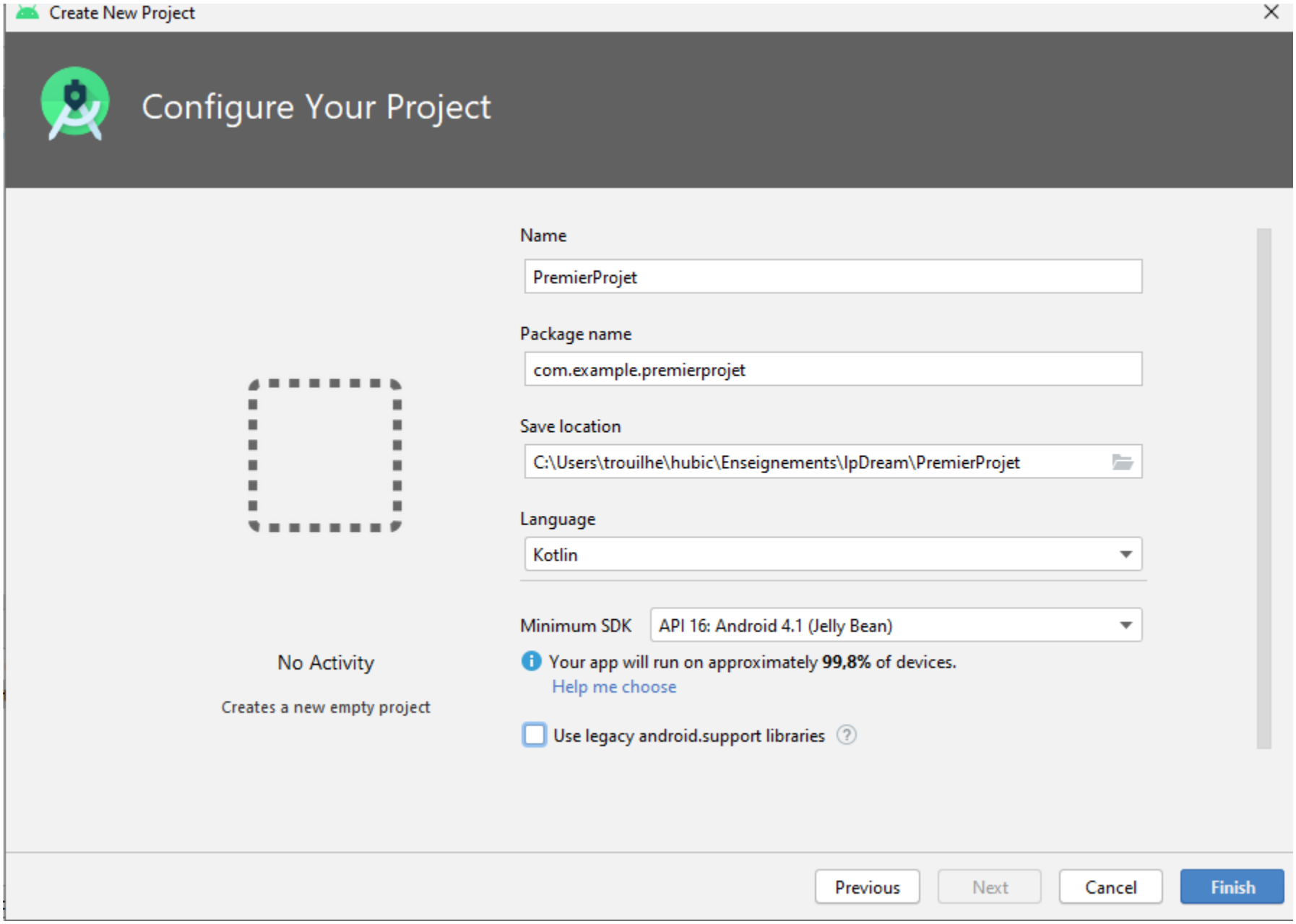
```
// Commentaire sur une ligne

/*
Commentaire sur
    plusieurs lignes
*/
```

Commentaire pour la documentation :

```
/**
 * @author
 */
```

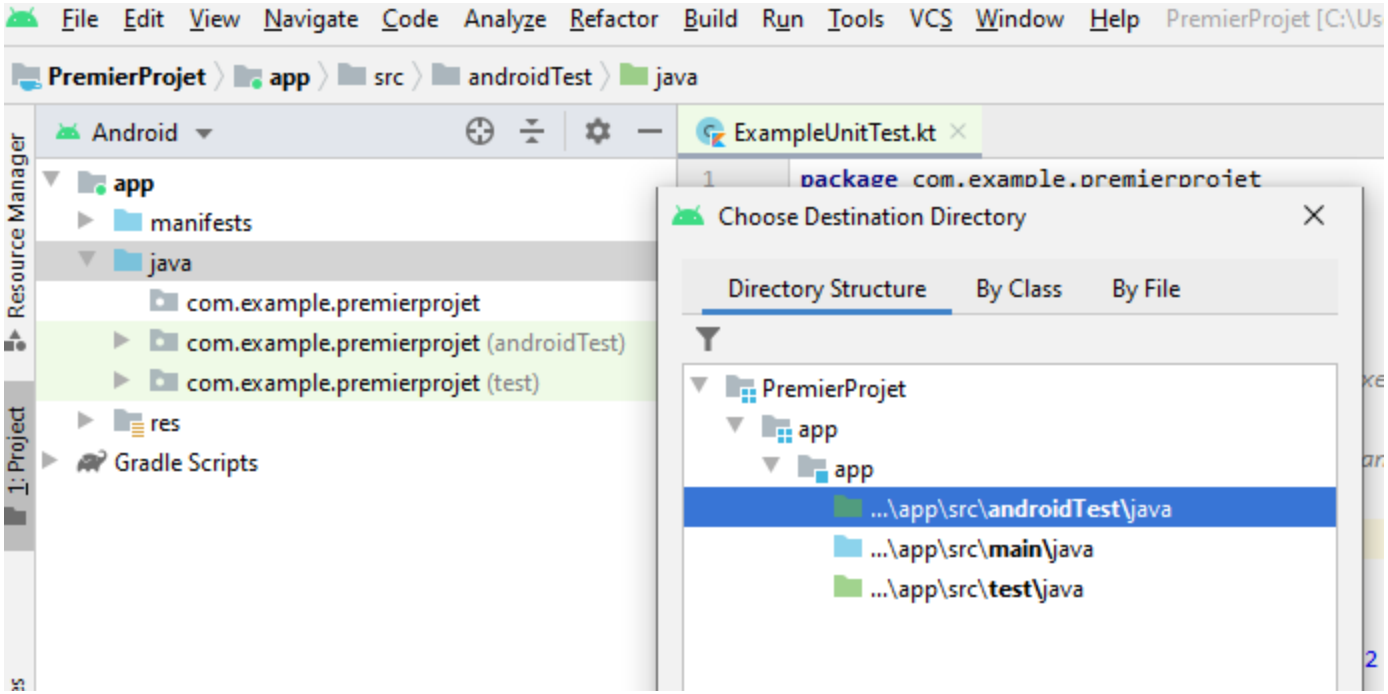
Partie 1 : premier programme

Ouvrir Android studio
File > New Projet
choisir « Empty Activity »


Create New Project
Configure Your Project
Name
PremierProjet
Package name
com.example.premierprojet
Save location
C:\Users\trouilhe\hubic\Enseignements\IpDream\PremierProjet
Language
Kotlin
Minimum SDK
API 16: Android 4.1 (Jelly Bean)
Your app will run on approximately 99,8% of devices.
Help me choose
Use legacy android.support libraries
No Activity
Creates a new empty project
Previous
Next
Cancel
Finish

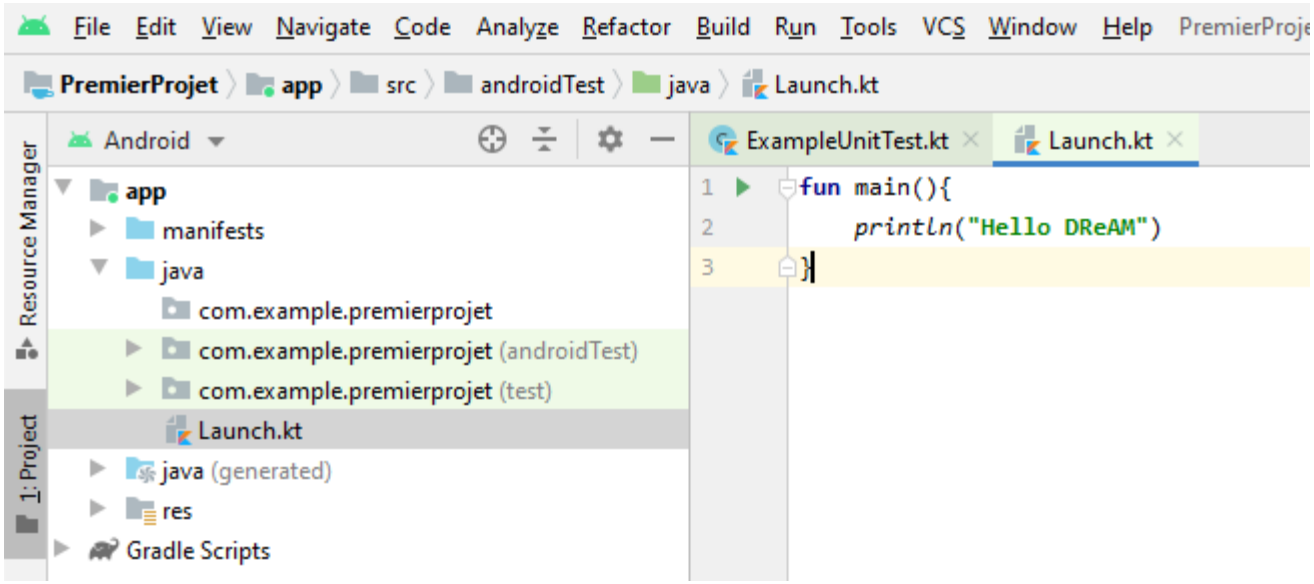
Créer un fichier contenant le main()

Bouton droit sur le répertoire « java » et choisir New > Kotlin File/Class – puis choisir File

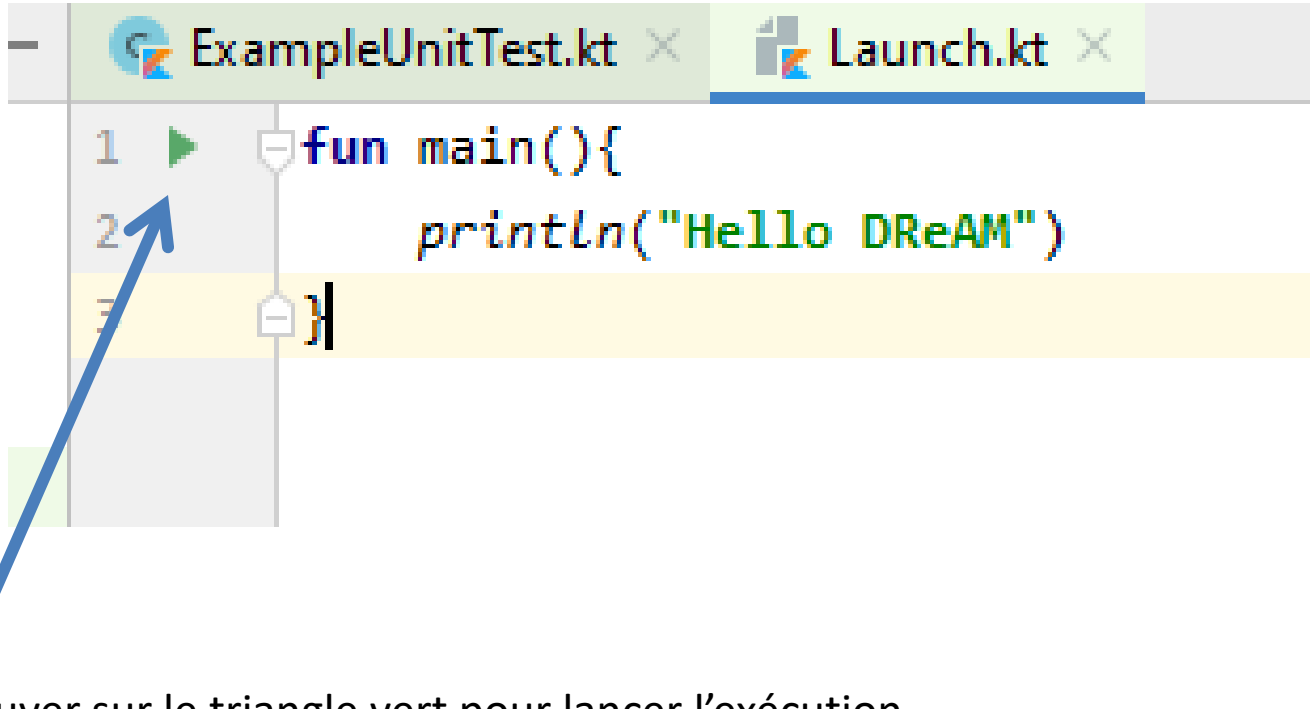


Après avoir fait « ok » choisir « File » et donner un nom (par exemple Launch)

Écrire la fonction main()

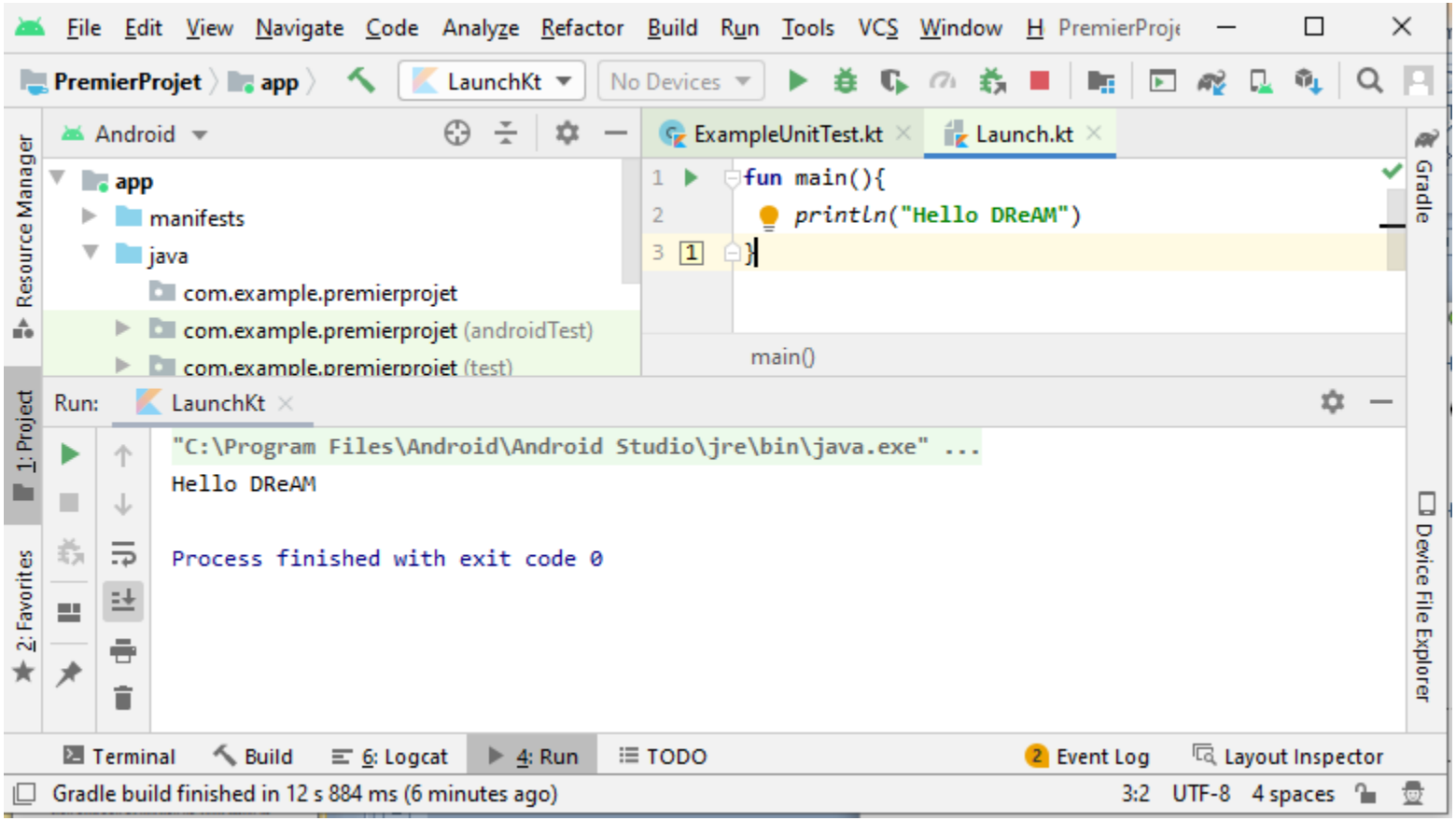


Exécuter le programme



Appuyer sur le triangle vert pour lancer l'exécution

Les entrées et sorties se font dans la fenêtre d'exécution



Exercices

1- Écrire un main qui lit une valeur, calcule l’arrondi supérieur et l’affiche.

2- Écrire un main qui lit une phrase, change toutes les lettres minuscules a en lettre majuscule A, sépare chaque mot et les met dans une liste, puis affiche le dernier élément de la liste.

3- Écrire un fonction qui calcule la somme d’un tableau passé en paramètre. Utiliser cette fonction dans un main pour calculer la somme de 6 valeurs réelles entrées au clavier.

Correction

```
// exercice 1

    val reader = Scanner(System.`in`)
    print("Entrer un nombre : ")
    val num:Float = reader.nextFloat()

    println(round(num))
```

```
"C:\Program Files\JetBrains\IntelliJ
Entrer un nombre : 12,6
13.0

Process finished with exit code 0
```

Correction

```
// exercice 2

    val reader = Scanner(System.`in`)
    print("Entrer la phrase: ")
    val phrase:String = reader.nextLine()

    val phraseBis=phrase.replace('a', 'A')
    println(phraseBis)

    val t=phraseBis.split(" ")
    println(t[t.lastIndex])
```

Correction

```
// exercice 3

fun somme(t: Array<Float>): Float {
    var s:Float=0.0F
    for (i in 0..t.size-1) s+=t[i]
    return s
}

val reader = Scanner(System.`in`)
val t:Array<Float?>
t=arrayOfNulls(3)
print("Entrer les 3 valeurs: ")
for (i in 0..2)   t[i]=reader.nextFloat()
println(message = somme(t as Array<Float>))
```

Partie 1 : quelques compléments

Énumération

Cast automatique

Fonction anonyme et lambda

Les énumérations

```
enum class Chiffre {
   un, deux, trois
}


Chiffre.un
```

Cast automatique : **`is`**

```
if (obj is String) { // obj est
automatiquement casté en string
}
else // obj ne change pas de type
```

Fonction anonyme

```
fun identique(p1: Int, p2: Int): Boolean {
                return p1==p2
}
```

```
fun (p1: Int, p2: Int): Boolean {
                return p1==p2
}
```

Lambda expression

```
fun identique(p1: Int, p2: Int): Boolean =
                p1==p2


val identique: (Int, Int) -> Boolean =
               { p1:Int, p2:Int -> p1==p2}
```

Lambda expression avec 1 seul paramètre

```
val doubler: (Int) -> Int = {p1:Int -> p1*2}
```

Le paramètre p1 peut ne pas être indiqué et on utilise **`it`** :

```
val doubler: (Int) -> Int = {it*2}
```

Une fonction peut prendre une lambda en paramètre

```
fun essai(p1 : Int,
                  doubler:(Int)->Int):Unit {
    println("double de $p1 : ${doubler(p1)}")
}

essai(6, {x:Int -> x*2})
```

Exercices

Ces programmes compilent-ils correctement ?

fun f1(x:Int=6, y: (Int) -> Int =7):Int { return y(x)}

fun f2(x:Int=6, y: (Int) -> Int ={it}) { return y(x)}

fun f3(x:Int=6, y: (Int) -> Int ={x:Int->x+6}):Int{ return y(x)}

fun f4(x: (Int) -> Int ={println(it)
it+7}){x(4)}

Exercices

Utiliser le constructeur de tableau

```
<init>(size: Int, init: (Int) -> T)
```

Pour créer un tableau de 8 valeurs correspondant aux premiers nombres pairs : 2 4 6 8 10 12 14 16

Correction

```
val a: Array<Int>

 a=Array(10, {2*it+2})


 for (i in 0..a.size-1) print("${a[i]} ")
```