Cités des Métiers et des Compétences
La formation nouvelle génération

مكتب التكوين المهني و إنعاش الشغل
Office de la Formation Professionnelle et de la Promotion du Travail

Compte-rendu
Devoir n°2

Nom: Boulahya

Prénom: Asma

Groupe: DEVOAM 202

Année de formation:2025/2026

Introduction

Le TP N°10 a pour objectif de renforcer la maîtrise des **threads, Runnable, callbacks et propriétés différées (lateinit et by lazy) en Kotlin**. Les exercices permettent de comprendre comment exécuter des tâches en parallèle, gérer l'interruption d'un thread, utiliser des callbacks pour retourner des résultats et manipuler des propriétés dont l'initialisation est différée.

Exercice1: Affichage de nombres avec un thread

```
fun main() {
    val thread = Thread {
        for (i in 1 ≤ .. ≤ 10) {
            println(i)
            Thread.sleep( millis = 1000)
        }
    }
    thread.start()
}
```

Figure 1: Capture d'écran du programme affichant les nombres de 1 à 10 avec un délai d'une seconde.

Explication :

Dans cet exercice, nous avons créé un thread qui affiche les nombres de 1 à 10 avec un délai d'une seconde entre chaque affichage. Cela permet de voir comment exécuter une tâche en parallèle à la méthode principale (`main`) sans bloquer le reste du programme.

Résultat :

```
1
2
3
4
5
6
7
8
```

Figure2: Résultat de l'exécution de l'exercice 1

Exercice2: Interruption d'un thread

```kotlin
fun main() {
    val thread = Thread {
        try {
            for (i in 1 ≤ .. ≤ 10) {
                println(i)
                Thread.sleep( millis = 1000)
            }
        } catch (e: InterruptedException) {
            println("Comptage interrompu !")
        }
    }

    thread.start()
    Thread.sleep( millis = 5000)
    thread.interrupt()
}
```

Figure3: Capture montrant le thread arrêté après 5 secondes et le message "Comptage interrompu !" affiché.

Explication:

Ce programme démarre un thread qui compte de 1 à 10 mais est interrompu après 5 secondes. L'utilisation de `try-catch` avec `InterruptedException` permet de gérer l'arrêt prématuré du thread et d'afficher un message indiquant que le comptage a été interrompu.

Résultat :

```
1
2
3
4
5
Comptage interrompu !
```

Figure4: Résultat de l'exécution de l'exercice 2

Exercice3: Implémentation de Runnable

```
class AlphabetRunnable : Runnable {  1 Usage
    override fun run() {
        for (c in 'A' ≤ .. ≤ 'Z') {
            println(c)
            Thread.sleep( millis = 500)
        }
    }
}

fun main() {
    val thread = Thread( task = AlphabetRunnable())
    thread.start()
}
```

Figure5: Capture affichant les lettres de A à Z avec un délai de 500 ms.

Explication:

Une classe implémente l'interface Runnable pour définir la tâche à exécuter dans un thread. Dans la méthode run, les lettres de A à Z sont affichées avec un délai de 500 ms entre chaque lettre. Cela illustre la flexibilité d'exécution parallèle grâce à Runnable.

Résultat:

```
M
N
O
P
Q
R
```

Figure6: Résultat de l'exécution de l'exercice 3

Exercice4: Utilisation d'un callback

```
fun calculer(a: Int, b: Int, callback: (Int) -> Unit) {  1 Usage
    val somme = a + b
    callback(somme)
}

fun main() {
    calculer( a = 5,  b = 7) { resultat ->
        println("Résultat du calcul: $resultat")
    }
}
```

Figure7: Capture montrant le résultat du calcul retourné par le callback

Explication:

L'exercice montre comment créer une fonction qui effectue un calcul (addition de deux nombres) et utilise un callback pour retourner le résultat. Cela illustre la communication asynchrone entre une fonction et son appelant.

Résultat:

```
Résultat du calcul: 12
```

Figure8: Résultat de l'exécution de l'exercice 4

Exercice5: Propriétés lateinit et by lazy

```
class Personne {  1 Usage
    lateinit var name: String  2 Usages
    val description: String by lazy { "Nom: $name" }  2 Usages
}

fun main() {
    val p = Personne()
    try {
        println(p.description)
    } catch (e: UninitializedPropertyAccessException) {
        println("Propriété 'name' non encore initialisée")
    }

    p.name = "Asma"
    println(p.description)
}
```

Figure9: Capture montrant l'accès à la propriété `description` avant et après l'initialisation de `name`, avec le message d'erreur géré.

Explication:

Nous avons créé une classe avec une propriété `lateinit` et une autre `by lazy`. La propriété `description` dépend de `name` et est évaluée uniquement lors du premier accès. L'exercice illustre la gestion des initialisations différées et la prévention d'erreurs lorsque la propriété n'est pas encore initialisée.

Résultat:

```
Propriété 'name' non encore initialisée
Nom: Asma
```

Figure10: Résultat de l'exécution de l'exercice 5

Conclusion

Ce TP a permis de se familiariser avec la programmation concurrente en Kotlin, notamment l'utilisation de threads, l'interruption de threads, la mise en œuvre de Runnable, l'utilisation des callbacks et la gestion des propriétés différées (lateinit et by lazy). Ces concepts sont essentiels pour le développement d'applications mobiles performantes et réactives, où l'exécution parallèle et la gestion efficace des ressources sont primordiales.