Cités des Métiers et des Compétences
La formation nouvelle génération

OFPPT
La voie de l'avenir

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

Compte-rendu
TP 14 - 1

Nom: Boulahya

Prénom: Asma

Groupe: DEVOAM 202

Année de formation:2025/2026

Sommaire

Introduction

Ce travail pratique a pour objectif de nous familiariser avec le concept des **coroutines en Kotlin**, un outil puissant permettant d'exécuter des tâches **asynchrones** sans bloquer le programme principal.
L'exercice proposé simule la gestion d'une commande dans un restaurant : vérification des ingrédients, préparation du repas et livraison.
À travers ce TP, nous allons apprendre à organiser plusieurs opérations simultanément tout en respectant un ordre logique d'exécution.

Objectif du TP

- Comprendre l'utilisation des **coroutines** dans un programme Kotlin.
- Apprendre à exécuter des tâches **en parallèle** de manière **ordonnée**.
- Utiliser des **délais simulés (`delay()`)** pour imiter des actions réelles (vérification, préparation, livraison).
- Découvrir le **Dispatchers.IO**, utilisé pour les opérations d'entrée/sortie.

Exercice1: Système de gestion de bibliothèque

Question 1 : Créer une fonction `verifierDisponibilite()`

```kotlin
import kotlinx.coroutines.*

suspend fun verifierDisponibilite() {  1 Usage
    println("Vérification des ingrédients en cours...")
    delay( timeMillis = 2000)
    println("Ingrédients disponibles ")
}
```

Figure 1: Exécution de la fonction verifierDisponibilite()

Cette fonction simule la vérification des ingrédients nécessaires pour une commande.
Elle doit durer **2 secondes**, ce qui est simulé grâce à la fonction `delay(2000)`.

- Le mot-clé **suspend** indique que cette fonction peut être suspendue (utilisée dans une coroutine).
- **delay(2000)** met en pause la coroutine pendant 2 secondes sans bloquer le thread principal.
- **println()** affiche des messages simulant le processus de vérification.

Question 2 : Créer une fonction preparerCommande()

```kotlin
suspend fun preparerCommande() {  1 Usage
    println("Préparation de la commande...")
    delay( timeMillis = 5000)
    println("Commande prête ")
}
```

Figure 2: Exécution de la fonction preparerCommande()

Cette fonction simule la **préparation de la commande**, qui prend **5 secondes**.

- La fonction utilise le même principe que la première.
- Le temps d'attente est de 5 secondes pour imiter la préparation d'un repas.
- Les messages permettent de suivre la progression dans la console.

Question 3 : Créer une fonction livrerRepas()

```
suspend fun livrerRepas() = withContext( context = Dispatchers.IO) {  1 Usage
    println("Livraison du repas en cours...")
    delay( timeMillis = 3000)
    println("Repas livré ")
}
```

Figure 3: Exécution de la fonction livrerRepas()

Cette fonction simule la **livraison du repas**. Elle doit durer **3 secondes** et utiliser le **Dispatcher.IO**, car cette tâche représente une opération d'entrée/sortie (comme une connexion réseau ou une requête API).

- **withContext(Dispatchers.IO)** permet d'exécuter la tâche dans un contexte optimisé pour les opérations d'E/S.
- La fonction attend 3 secondes avant d'afficher le message final.

Question 4 : Exécuter les trois fonctions avec des coroutines

```
fun main() = runBlocking {
    println("---- Application de gestion de commandes ----")

    val job1 = launch { verifierDisponibilite() }
    job1.join()

    val job2 = launch { preparerCommande() }
    job2.join()

    val job3 = launch { livrerRepas() }
    job3.join()

    println("Toutes les étapes sont terminées SS")
}
```

Figure 4 : Résultat final dans la console

Dans la fonction principale `main`, on lance les trois étapes **dans des coroutines séparées** mais **dans un ordre logique**, grâce à `join()`.

- **runBlocking** démarre un environnement de coroutine.
  launch crée une nouvelle coroutine pour chaque tâche.
  join() garantit que la tâche précédente se termine avant de commencer la suivante.
- Cela permet une exécution fluide et organisée.

Execution :

```
---- Application de gestion de commandes ----
Vérification des ingrédients en cours...
Ingrédients disponibles
Préparation de la commande...
Commande prête
Livraison du repas en cours...
Repas livré
Toutes les étapes sont terminées SS
```

Figure 5 :Résultat

Conclusion

Ce TP m'a permis de découvrir la **programmation asynchrone avec les coroutines en Kotlin**.
J'ai compris comment organiser des tâches parallèles tout en maintenant un ordre d'exécution logique.
Les coroutines facilitent la gestion du multitâche et sont très utiles pour les **applications Android**, où il faut éviter de bloquer le thread principal.
Grâce à cet exercice, j'ai appris à utiliser :

- Les fonctions **suspendues**,
- Le **Dispatchers.IO**,
- Et la synchronisation des coroutines avec **launch** et **join()**.

En conclusion, les coroutines sont un outil essentiel pour améliorer la **performance**, la **réactivité** et la **fluidité** des applications Kotlin et Android.