x

**COMPTE RENDU N 1**

* **Nom:** Karouit
* **Prénom:** Khadija
* **Groupe:** DEV OM202
* **Année de formation:**2025-2026

**Sommaire:**

[**Introduction:**](#_wga4fiu93nei) **4**

[**Exercice 1 – Opérations sur deux nombres**](#_7704fv9syfu7) **5**

[1. Demander à l'utilisateur de saisir deux nombres.](#_4poxa6gyny3r) 5

[2. Calculer et afficher](#_dps7golkj1pq) 5

[**Exercice 2 – Calcul de moyenne et évaluation**](#_yr3wfff6bwh) **7**

[**1-Demander à l'utilisateur de saisir les notes des trois examens.**](#_huhx099jjo3y) **7**

[2-Calculer et afficher la moyenne des trois notes](#_5frwqpktsca) 7

[4-Afficher un message personnalisé selon la performance](#_pliazn5gk9u4) 8

[**Exercice 3 – Threads avec Runnable**](#_kbnpgj8v35mw) **9**

[1-Une classe implémente l’interface Runnable](#_g7ec1isndtwy) 9

[2-La méthode run() affiche un message chaque second](#_byjx25nbwj65) 9

[3-Plusieurs threads sont créés et démarrés :](#_sm954x25c1bd) 10

[**Exercice 4 – Chargement différé avec lazy**](#_jwt1m4fuwsie) **12**

[1-Simulation du chargement d’une configuration complexe](#_k5qfqyfonhkx) 12

[2-Déclaration d’une propriété lazy dans une classe App](#_f3089iq0u5uf) 13

[3-Initialisation de la connexion](#_62howmfui91l) 14

[4-Exécution:](#_kophhy8fyq8h) 14

[**Exercice 5 : Instanciation différée avec lazy**](#_w0ai4deoixzc) **15**

[1-Définition d’une fonction calculLourd() simulant un traitement complexe](#_6t5i5nutr95d) 15

[2:Déclaration d’une propriété lazy pour stocker le résultat du calcul](#_nrv18pm6a7pm) 16

[3-Affichage du résultat uniquement lors du premier accès à la propriété](#_r6196tbk7n4u) 16

[4-Exécution](#_yjiuwevjqvlo) 16

[**Exercice 6 : Initialisation tardive avec lateinit**](#_9vpgs4so9kxq) **17**

[1-La classe UtilisateurService simule un service avec un bloc init](#_y3j7uvghsmsm) 17

[2-La classe Application contient une propriété lateinit pour un objet UtilisateurService](#_kw16kcj1jmkj) 18

[3-Utilisation](#_8f15g880ew12) 19

[4-Exécution](#_soimwibxznd7) 19

[**Exercice 7:Simulation de connexion à une base de données**](#_4r2w7znna8w3) **20**

[1-Classe DatabaseConnection](#_jzx46fekq8tb) 20

[2-Classe DataManager](#_4wgn4qinghqd) 21

[3-Initialisation:](#_56nchxdzm4tg) 22

[4-Exécution:](#_o26826lcgmfx) 22

[**CONCLUSION:**](#_encvmz3zfj26) **23**

**Introduction:**

Ce TP a pour but d’explorer les mécanismes d’instanciation différée en Kotlin, à travers l’utilisation des mots-clés lazy et lateinit. Ces outils permettent de retarder la création d’un objet jusqu’à ce qu’il soit réellement nécessaire, ce qui améliore la gestion des ressources et la performance des applications. Les exercices proposés couvrent des cas concrets comme le chargement de services, la simulation de connexions à une base de données, et l’exécution de tâches en parallèle avec des threads.

**Exercice 1 – Opérations sur deux nombres**

1. **Demander à l'utilisateur de saisir deux nombres.**
2. **Calculer et afficher**

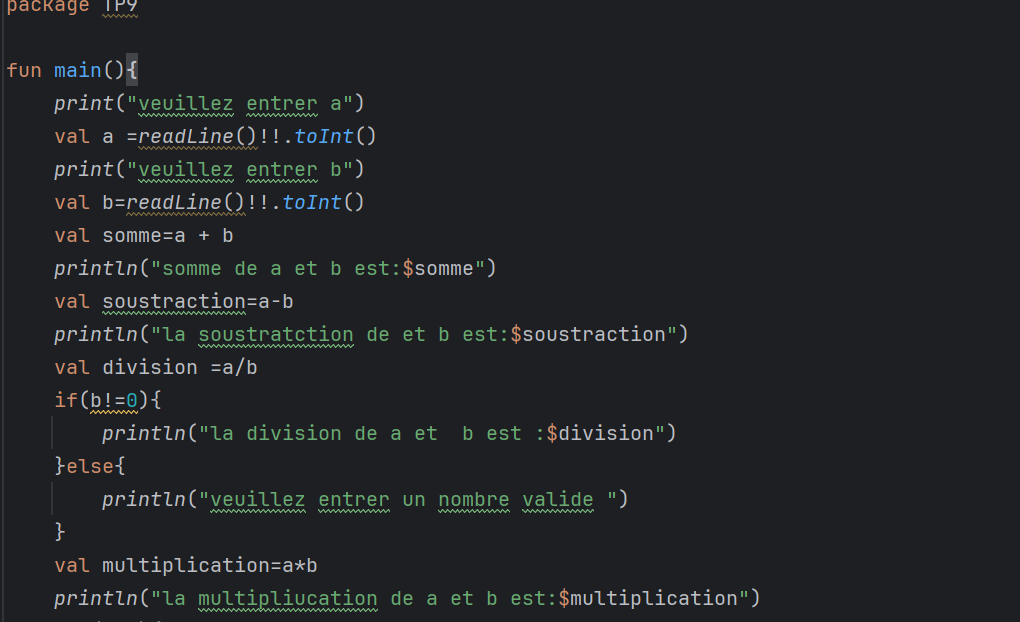
****

Figure1:Demande et calcule

* Demande à l’utilisateur d’entrer deux entiers a et b. Lit les valeurs avec readLine()!!.toInt().
* Calcule et affiche :
* la somme a + b
* la soustraction a - b
* la multiplication a \* b
* la division a / b (seulement si b ≠ 0, sinon affiche un message d’erreur)

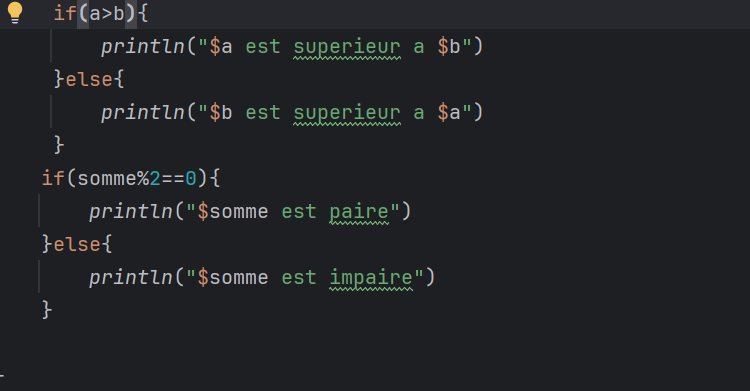


Figure2:verification

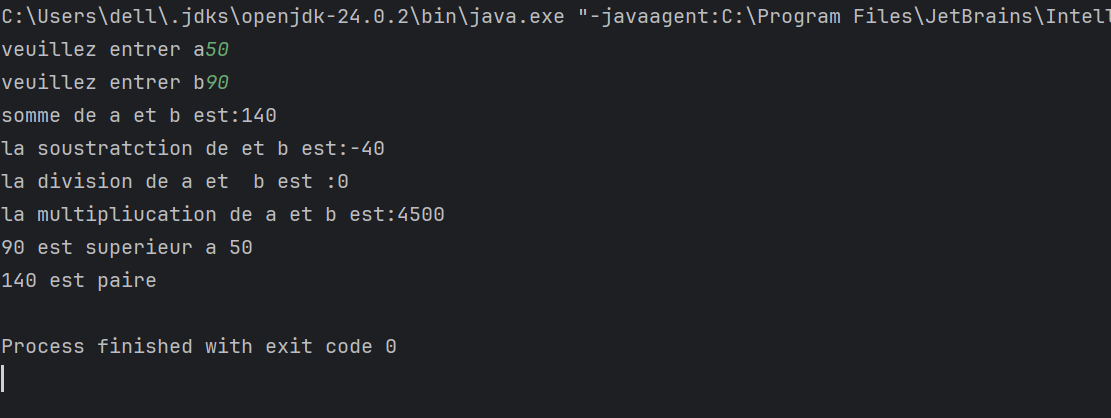
* Compare les deux nombres aet b :Affiche lequel est supérieur.
* Vérifie si la somme de a et b est paire ou impaire :Utilise l’opérateur % pour tester la divisibilité par 2.

Figure3:affichage

# **Exercice 2 – Calcul de moyenne et évaluation**

# **1-Demander à l'utilisateur de saisir les notes des trois examens.**

# 

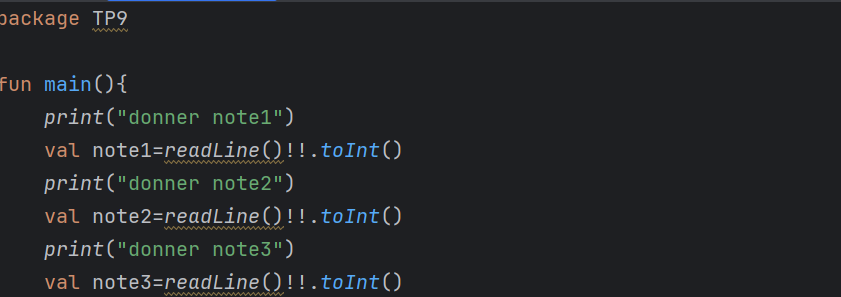


Figure4:saisir les notes

On demande à l’utilisateur d’entrer trois notes : note1, note2 et note3. On lit chaque note avec readLine()!!.toInt() et on les stocke dans des variables.

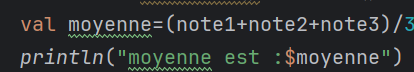
**2-Calculer et afficher la moyenne des trois notes**

Figure5:la moyenne

## 4-Afficher un message personnalisé selon la performance

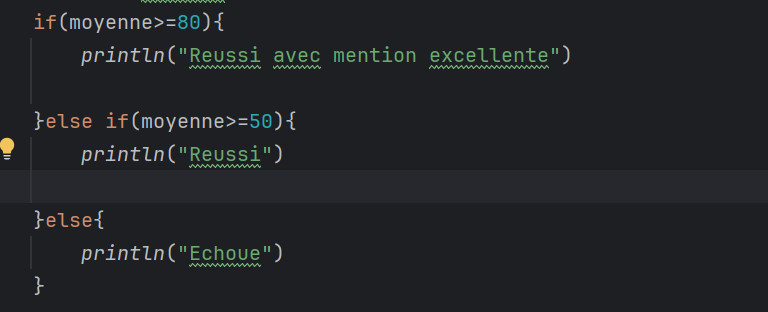
****

Figure6:affichage du message

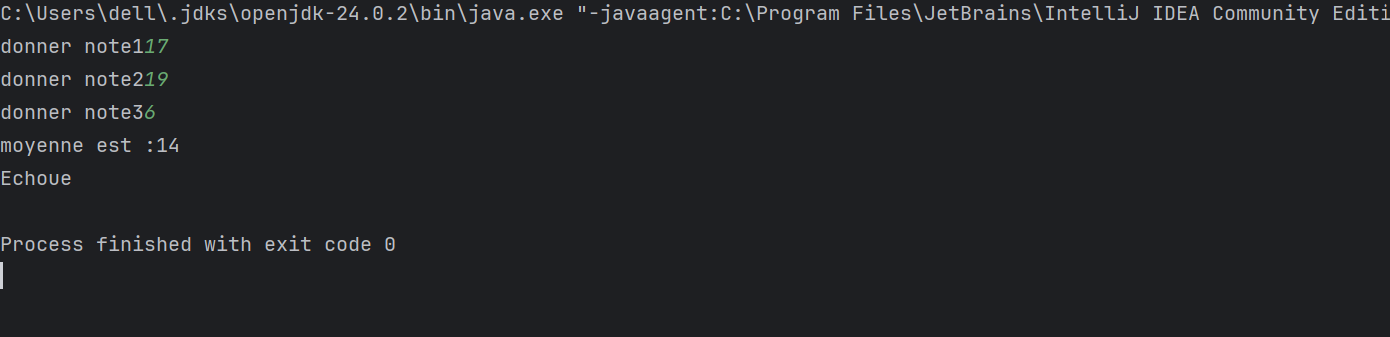
****

Figure7:execution

**Exercice 3 – Threads avec Runnable**

**1-Une classe implémente l’interface Runnable**

**2-La méthode run() affiche un message chaque second**

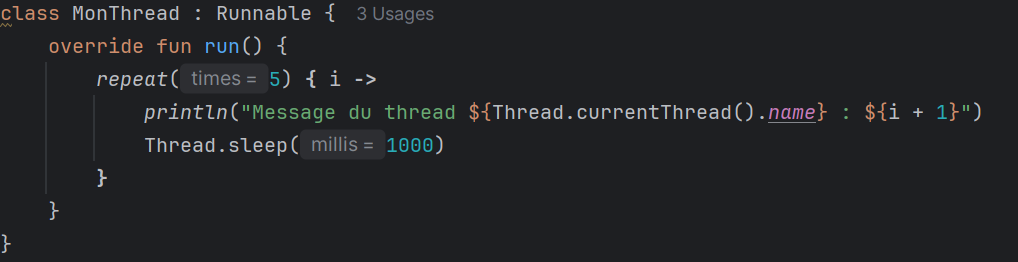


Figure8:interface Runnable

* On crée une classe MonThread qui implémente l’interface Runnable. On redéfinit la méthode run() pour exécuter une tâche répétée 5 fois.
* À chaque répétition :
* On affiche le nom du thread et le numéro de l’itération.
* On met le thread en pause pendant 1 seconde avec Thread.sleep(1000).

**3-Plusieurs threads sont créés et démarrés :**

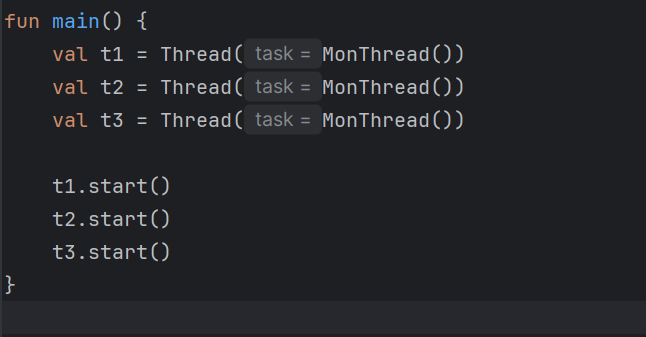


Figure9:Création et démarrage

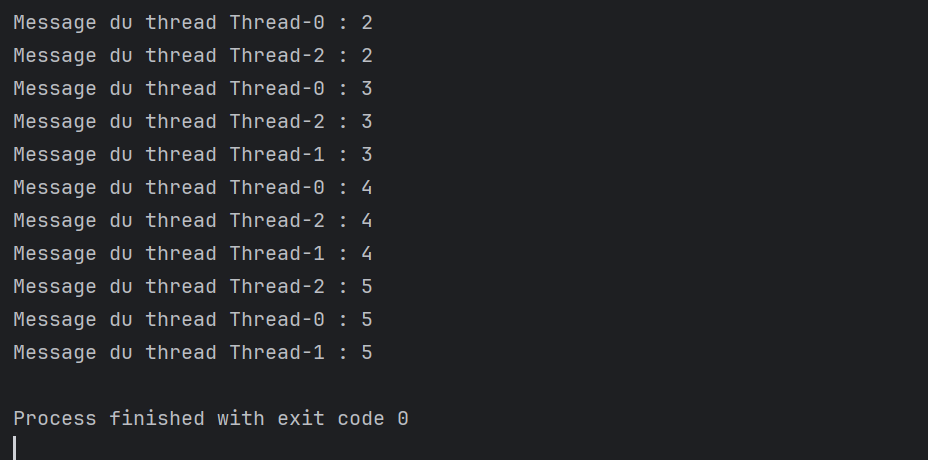


Figure10:execution

**Exercice 4 – Chargement différé avec lazy**

**1-Simulation du chargement d’une configuration complexe**

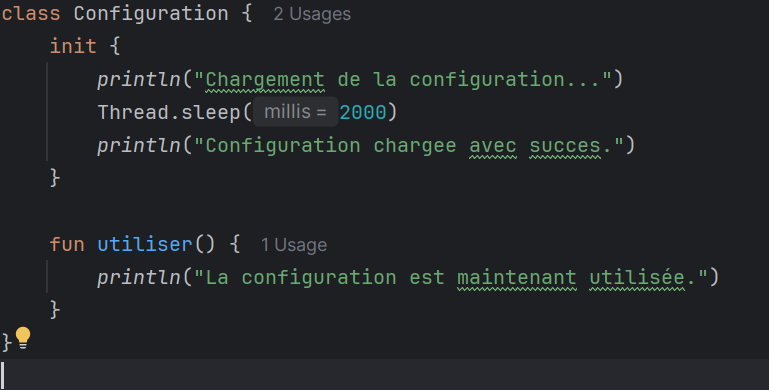


Figure11:classe Configuration

* On crée une classe Configuration avec un bloc init exécuté automatiquement à l’instanciation. On affiche un message de chargement, puis on attend 2 secondes avec Thread.sleep(2000).
* Une fois le délai écoulé, on affiche que la configuration est chargée avec succès.
* La méthode utiliser() affiche que la configuration est en cours d’utilisation.

**2-Déclaration d’une propriété lazy dans une classe App**

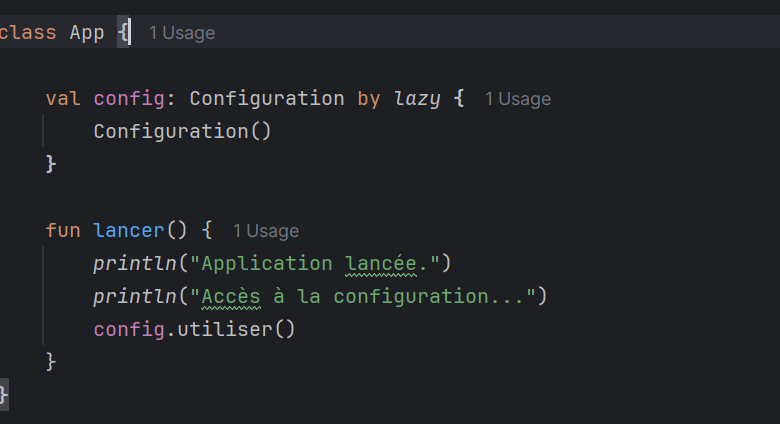


Figure12:classe App

* On crée une classe App avec une propriété config de type Configuration. On utilise by lazy pour que l’objet Configuration soit créé uniquement au premier accès.
* La méthode lancer() :
* Affiche "Application lancée."
* Accède à config, ce qui déclenche son initialisation (et exécute le bloc init de Configuration)
* Appelle la méthode utiliser() sur config

**3-Initialisation de la connexion**

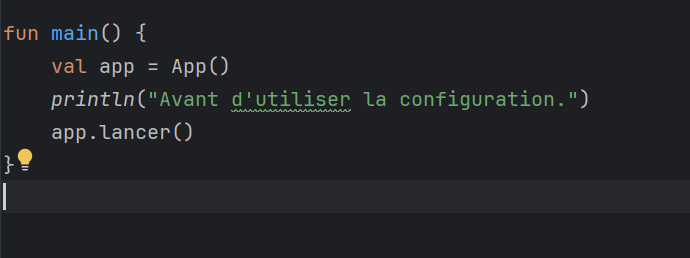


Figure13:utilisation

**4-Exécution:**

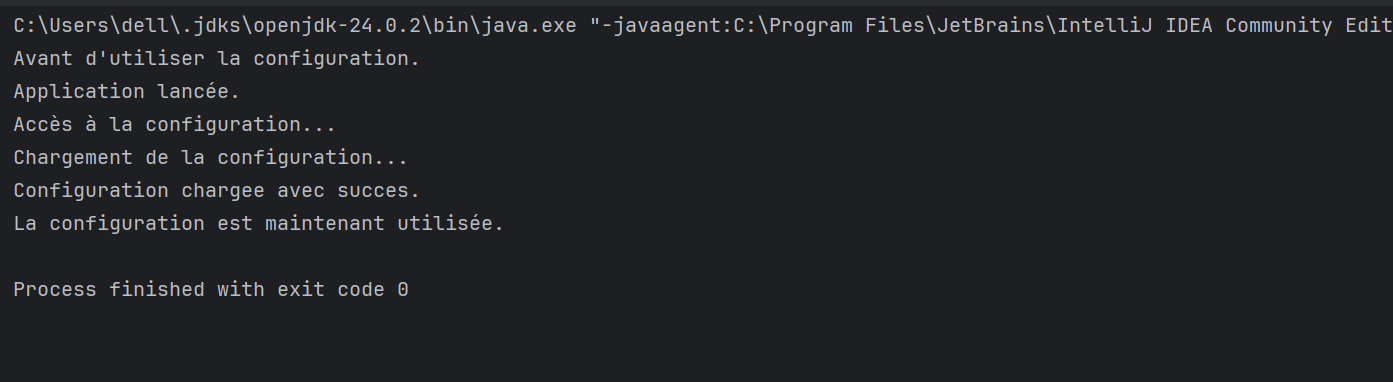


Figure14:execution

**Exercice 5 : Instanciation différée avec lazy**

**1-Définition d’une fonction calculLourd() simulant un traitement complexe**

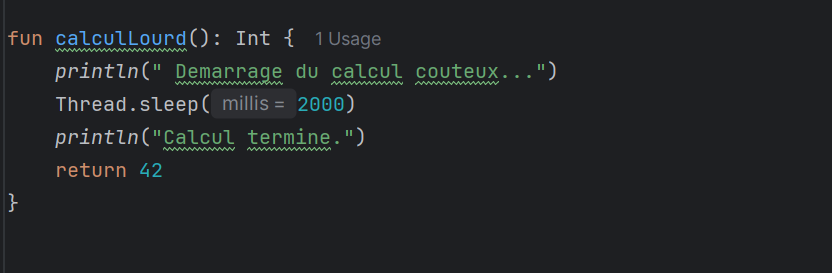


Figure15:classe calculLourd

* On lance un calcul simulé avec le message "Démarrage du calcul coûteux...". On met le programme en pause pendant 2 secondes avec Thread.sleep(2000) pour simuler un traitement long.
* On affiche "Calcul terminé." une fois le délai écoulé.
* On retourne la valeur 42 comme résultat du calcul.

**2:Déclaration d’une propriété lazy pour stocker le résultat du calcul**

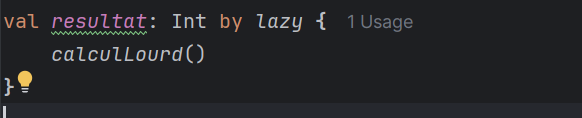


Figure16: propriété lazy

On déclare une variable resultat avec by lazy, ce qui signifie que la fonction calculLourd()ne sera exécutée qu’au premier accès à resultat. Cela permet de retarder le calcul coûteux jusqu’à ce qu’il soit vraiment nécessaire.

**3-Affichage du résultat uniquement lors du premier accès à la propriété**

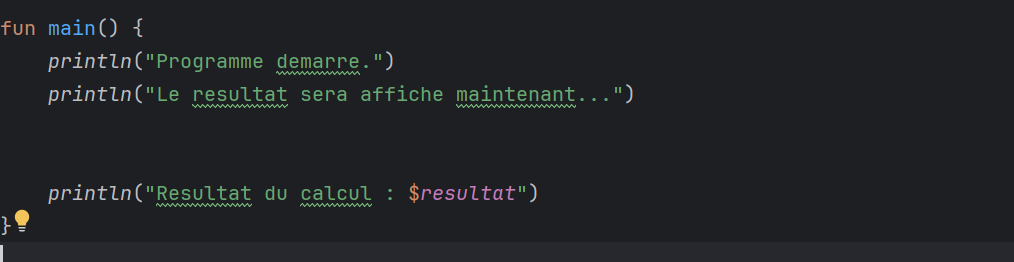


Figure17: utilisation

**4-Exécution**

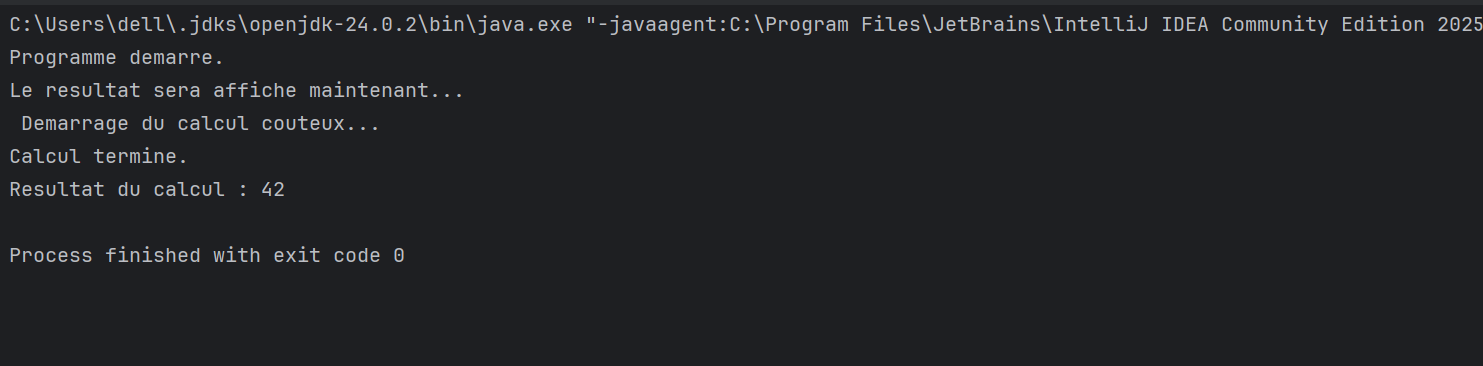


Figure18: execution

**Exercice 6 : Initialisation tardive avec lateinit**

**1-La classe UtilisateurService simule un service avec un bloc init**

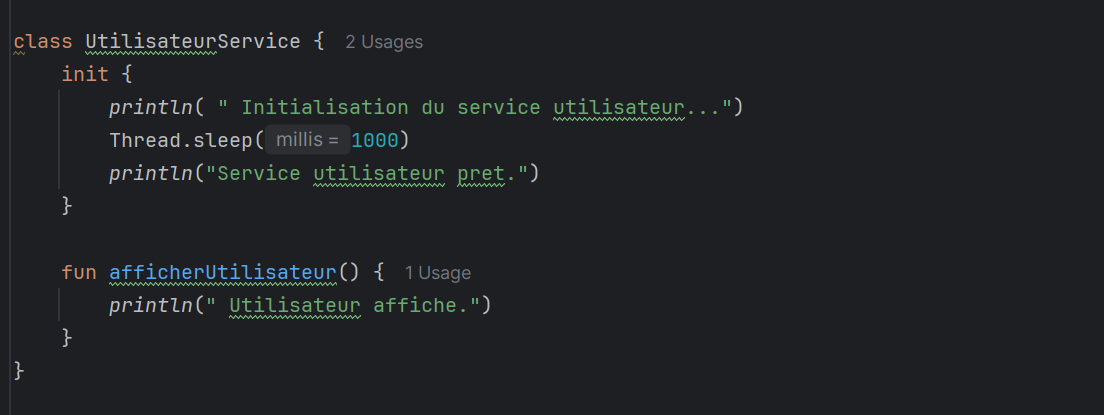


Figure19: classe UtilisateurService

* On crée une classe UtilisateurService qui simule l’initialisation d’un service. Le bloc init s’exécute automatiquement à la création d’un objet :
  + On affiche "Initialisation du service utilisateur...".
  + On attend 1 seconde avec Thread.sleep(1000) pour simuler un chargement.
  + On affiche "Service utilisateur prêt." une fois l’attente terminée.
* La méthode afficherUtilisateur() affiche "Utilisateur affiché." quand elle est appelée.

**2-La classe Application contient une propriété lateinit pour un objet UtilisateurService**



Figure20: lateinit

* On déclare une variable service avec lateinit, ce qui permet une initialisation tardive. On initialise service dans la méthode initialiserService() en créant un objet UtilisateurService.
* On utilise ::service.isInitialized pour vérifier si le service est prêt avant de l’utiliser.
* Si le service est initialisé, on appelle afficherUtilisateur() ; sinon, on affiche un message d’erreur.

**3-Utilisation**

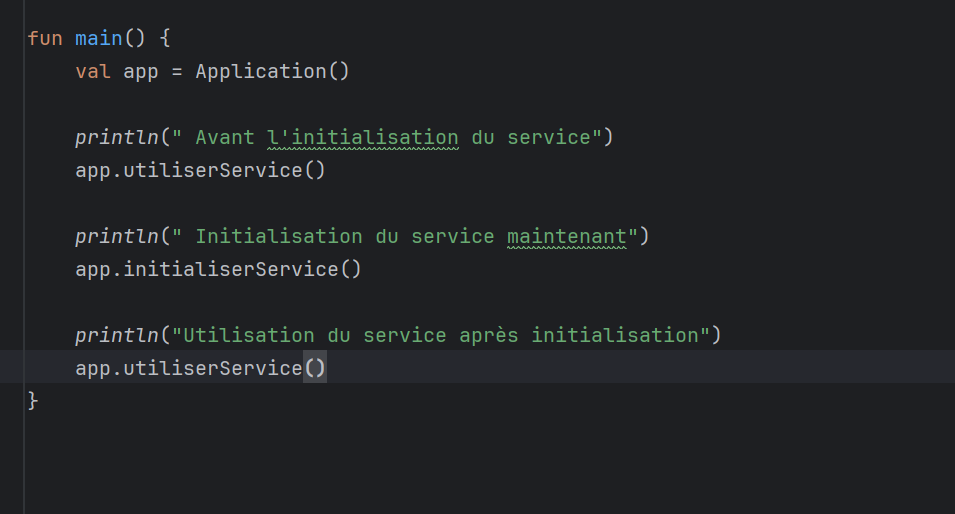


Figure21: utilisation

**4-Exécution**

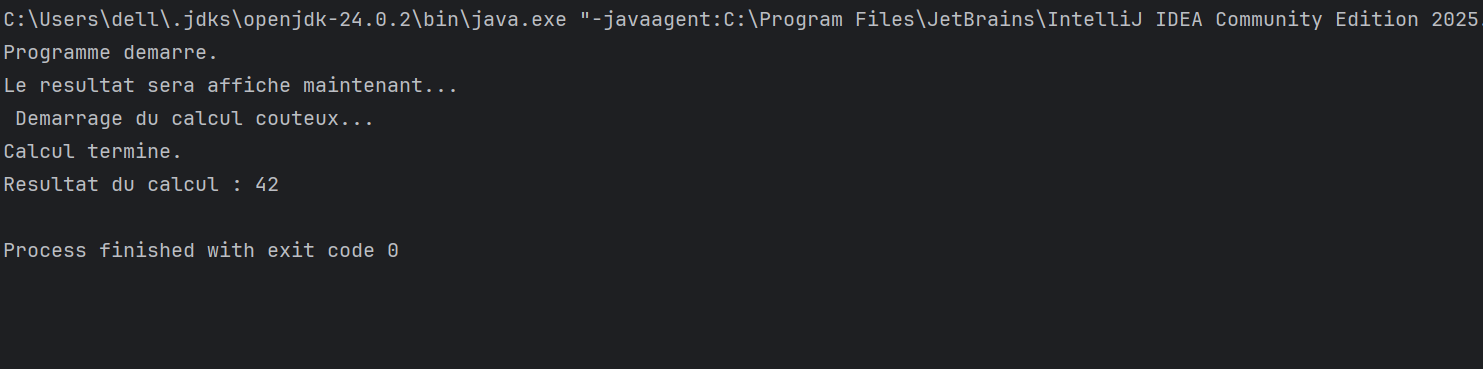


Figure22: execution

**Exercice 7:Simulation de connexion à une base de données**

**1**-**Classe DatabaseConnection**



Figure23: classe DatabaseConnection

* La classe DatabaseConnection simule une connexion avec un délai de 1,5 seconde. La méthode executerRequete(sql) affiche la requête SQL exécutée.

**2-Classe DataManager**

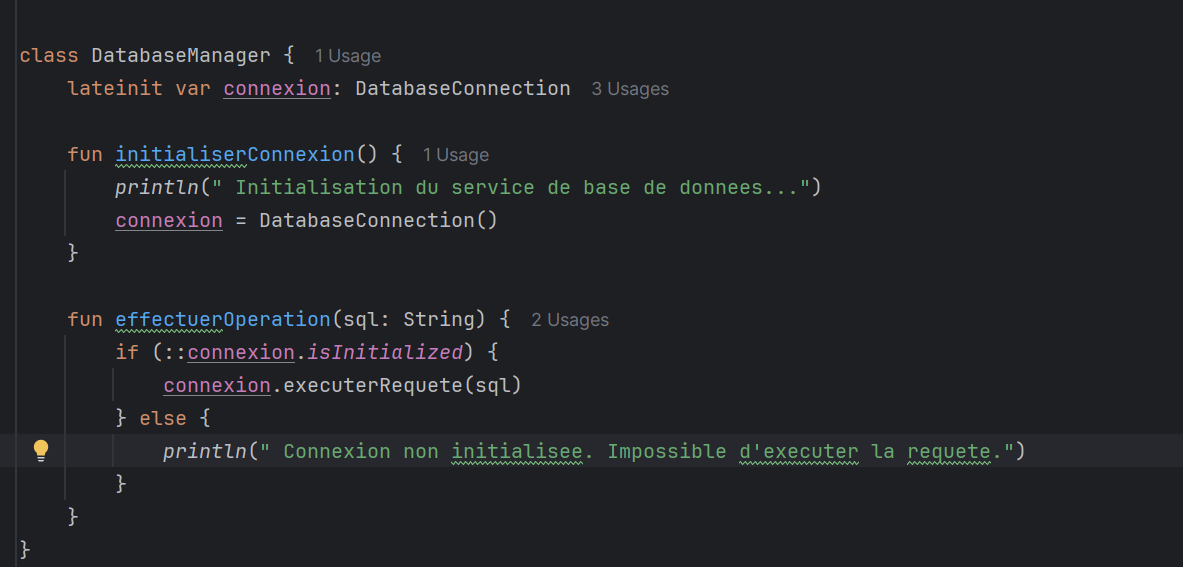


Figure24: classe DataManager

* La classe DatabaseManager utilise lateinit pour initialiser la connexion plus tard. initialiserConnexion() crée l’objet DatabaseConnection.
* effectuerOperation(sql) vérifie si la connexion est prête avant d’exécuter la requête SQL.

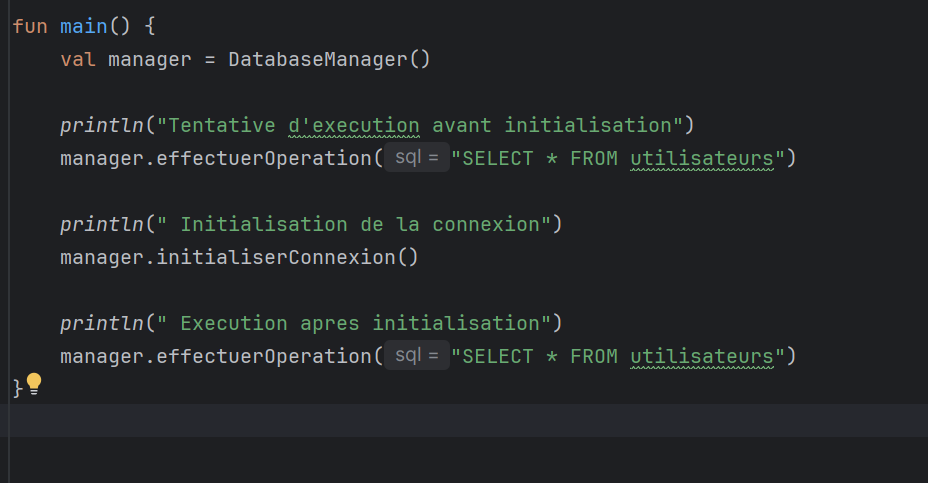
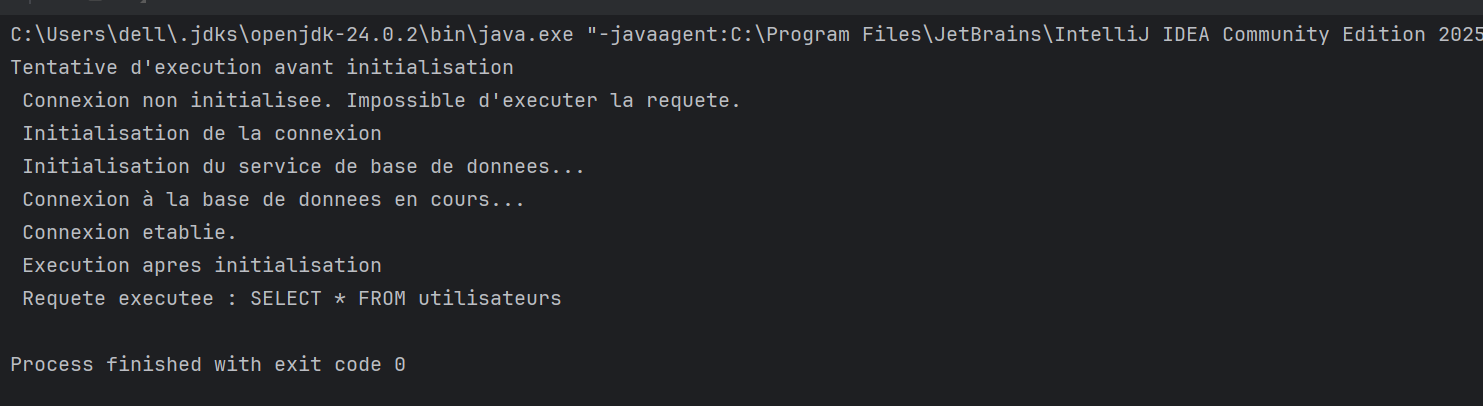
**3-Initialisation:**

Figure25: initialisation

## **4-Exécution:**

Figure26: execution

# **Conclusion:**

Ce TP a permis de comprendre comment Kotlin offre un contrôle précis sur le cycle de vie des objets grâce à lazy et lateinit. En différant l’instanciation, on évite les calculs inutiles et on optimise l’utilisation de la mémoire. Les exercices ont également mis en évidence l’importance de vérifier l’état d’un objet avant son utilisation, ce qui renforce la fiabilité du code. Ces pratiques sont essentielles pour développer des applications mobiles robustes et bien structurées.

Github ; https://github.com/kkkaktyq/Karouit\_Khadija\_DEV-OM202.git