programmation concurrente en Java

Thread, Runnable, Callable, Future, ExecutorService, et CompletableFuture.

TP : Programmation Concurrente en Java

Exercice 1 – Création de Threads

réer plusieurs threads simples qui affichent des messages simultanément.

Code Exemple :

```
class MonThread extends Thread {
    private String nom;

    public MonThread(String nom) {
        this.nom = nom;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(nom + " - Compteur : " + i);
            try {
                Thread.sleep(500); // Pause 0,5 sec
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

public class TestThread {
    public static void main(String[] args) {
        MonThread t1 = new MonThread("Thread A");
        MonThread t2 = new MonThread("Thread B");
```

```
        t1.start();
        t2.start();
    }
}
```

À faire :

1. Modifier la durée du sleep pour voir les effets.
2. Ajouter un troisième thread.
3. Observer que l'exécution est non déterministe (ordre variable).

Exercice 2 – Interface Runnable

Créer un thread via l'interface Runnable.

```
class TacheRunnable implements Runnable {
    private String nom;

    public TacheRunnable(String nom) {
        this.nom = nom;
    }

    @Override
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println("Exécution " + nom + " - étape " + i);
            try {
                Thread.sleep(400);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

```
public class TestRunnable {
   public static void main(String[] args) {
      Thread t1 = new Thread(new TacheRunnable("T1"));
      Thread t2 = new Thread(new TacheRunnable("T2"));

      t1.start();
      t2.start();
   }
}
```

À faire :

- Lancer 5 tâches en parallèle.
- Ajouter un message final dans le main pour montrer que le thread principal continue.

Exercice 3 – Callable et Future

Retourner un résultat à partir d'une tâche exécutée en parallèle.

Code Exemple :

```
import java.util.concurrent.*;

class Calcul implements Callable<Integer> {
   private int n;

   public Calcul(int n) {
      this.n = n;
   }

   @Override
   public Integer call() throws Exception {
      System.out.println("Calcul de la somme jusqu'à " + n);
      int somme = 0;
      for (int i = 1; i <= n; i++) {
         somme += i;
         Thread.sleep(100);
```

```
        }
        return somme;
    }
}

public class TestCallable {
    public static void main(String[] args) throws Exception {
        ExecutorService executor = Executors.newSingleThreadExecutor();

        Future<Integer> resultat = executor.submit(new Calcul(10));

        System.out.println("En attente du résultat...");
        Integer somme = resultat.get(); // Bloquant
        System.out.println("Résultat = " + somme);

        executor.shutdown();
    }
}
```

À faire :

1. Modifier le code pour calculer la somme de plusieurs valeurs (5, 10, 15) en parallèle.
2. Utiliser invokeAll() pour exécuter plusieurs Callable en une seule commande.

Exercice 4 – ExecutorService et ThreadPool

Utiliser un pool de threads pour exécuter plusieurs tâches en parallèle.

Code Exemple :

```
import java.util.concurrent.*;

class Tache implements Runnable {
    private int id;

    public Tache(int id) {
        this.id = id;
```

```
    }

    @Override
    public void run() {
        System.out.println("Tâche " + id + " exécutée par " + Thread.currentThread().getName());
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class TestExecutor {
    public static void main(String[] args) {
        ExecutorService pool = Executors.newFixedThreadPool(3); // 3 threads

        for (int i = 1; i <= 7; i++) {
            pool.execute(new Tache(i));
        }

        pool.shutdown();
    }
}
```

À faire :

- Modifier la taille du pool (1, 3, 5) et observer les changements.
- Ajouter un awaitTermination() pour attendre la fin de toutes les tâches.