Threads en JAVA

1. Introduction aux threads

- **Définition :** Un thread est une unité d'exécution légère qui fait partie d'un processus. Chaque programme Java commence par un thread principal appelé **main thread**.
- **Avantages :**
  - Permet le traitement parallèle pour des performances accrues.
  - Utile pour des applications interactives ou multitâches (serveurs, GUI, etc.).
- **Cycle de vie d'un thread :**

1. **Nouveau (New)** : Le thread est créé mais pas encore démarré.
2. **Exécutable (Runnable)** : Le thread est prêt à être exécuté.
3. **En cours d'exécution (Running)** : Le thread est en cours d'exécution.
4. **En attente (Waiting/Blocked)** : Le thread attend une ressource ou une condition.
5. **Terminé (Terminated)** : Le thread a terminé son exécution.

2. Création de threads en Java

Deux méthodes principales pour créer des threads :

a) En étendant la classe Thread

```
class MonThread extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Thread : " + i);
            try {
                Thread.sleep(1000); // Pause d'une seconde
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
```

```
    }
  }
}


public class ExempleThread {
  public static void main(String[] args) {
    MonThread thread = new MonThread();
    thread.start(); // Démarrage du thread
  }
}
```

b) En implémentant l'interface Runnable

```
class MonRunnable implements Runnable {
  @Override
  public void run() {
    for (int i = 1; i <= 5; i++) {
      System.out.println("Runnable : " + i);
      try {
        Thread.sleep(1000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }
}


public class ExempleRunnable {
  public static void main(String[] args) {
    Thread thread = new Thread(new MonRunnable());
    thread.start();
```

```
    }
}
```

c) Avec des expressions lambda (Java 8+)

```
public class ExempleLambda {
    public static void main(String[] args) {
        Thread thread = new Thread(() -> {
            for (int i = 1; i <= 5; i++) {
                System.out.println("Lambda : " + i);
                try {
                    Thread.sleep(1000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                }
            }
        });
        thread.start();
    }
}
```

3. Synchronisation des threads

Lorsque plusieurs threads accèdent à une ressource partagée, il faut les synchroniser pour éviter des comportements imprévisibles.

Problème sans synchronisation :

```
class Compteur {
    private int valeur = 0;

    public void incrementer() {
        valeur++;
    }
```

```
  public int getValeur() {
      return valeur;
    }
}

public class ExempleSansSync {
    public static void main(String[] args) {
      Compteur compteur = new Compteur();

      Thread t1 = new Thread(() -> {
         for (int i = 0; i < 1000; i++) compteur.incrementer();
      });

      Thread t2 = new Thread(() -> {
         for (int i = 0; i < 1000; i++) compteur.incrementer();
      });

      t1.start();
      t2.start();

      try {
         t1.join();
         t2.join();
      } catch (InterruptedException e) {
         e.printStackTrace();
      }

      System.out.println("Valeur finale : " + compteur.getValeur()); // Résultat incorrect
possible
```

```
    }
}
```

Solution : Synchronisation

```
class Compteur {
    private int valeur = 0;

    public synchronized void incrementer() {
        valeur++;
    }

    public synchronized int getValeur() {
        return valeur;
    }
}
```

4. Gestion des états des threads

Méthodes importantes :

1. **start()** : Démarre le thread.
2. **sleep(milliseconds)** : Suspend l'exécution du thread courant.
3. **join()** : Attend la fin d'un thread.
4. **interrupt()** : Interrompt un thread en cours.
5. **isAlive()** : Vérifie si un thread est encore actif.

Exemple : Utilisation de join

```
class MonThread extends Thread {
    @Override
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println(Thread.currentThread().getName() + " : " + i);
            try {
                Thread.sleep(1000);
```

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
  }
}


public class ExempleJoin {
  public static void main(String[] args) {
    MonThread t1 = new MonThread();
    MonThread t2 = new MonThread();


    t1.start();
    try {
      t1.join(); // Attend la fin de t1 avant de lancer t2
    } catch (InterruptedException e) {
      e.printStackTrace();
    }
    t2.start();
  }
}
```

5. Exemples pratiques

Exemple 1 : Calcul parallèle

Deux threads calculent la somme de parties différentes d'un tableau :

```
class SommePartielle implements Runnable {
  private int[] tableau;
  private int debut, fin;
  private int resultat;
```

```
  public SommePartielle(int[] tableau, int debut, int fin) {
      this.tableau = tableau;
      this.debut = debut;
      this.fin = fin;
   }

   @Override
   public void run() {
      for (int i = debut; i < fin; i++) {
         resultat += tableau[i];
      }
   }

   public int getResultat() {
      return resultat;
   }
}

public class CalculParallele {
   public static void main(String[] args) throws InterruptedException {
      int[] tableau = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
      int taille = tableau.length;

      SommePartielle t1 = new SommePartielle(tableau, 0, taille / 2);
      SommePartielle t2 = new SommePartielle(tableau, taille / 2, taille);

      Thread thread1 = new Thread(t1);
      Thread thread2 = new Thread(t2);
```

```
        thread1.start();
        thread2.start();

        thread1.join();
        thread2.join();

        int sommeTotale = t1.getResultat() + t2.getResultat();
        System.out.println("Somme totale : " + sommeTotale);
    }
}
```

6. Exercices

1. **Exercice 1 : Téléchargement parallèle**
    - Simulez le téléchargement de plusieurs fichiers simultanément en utilisant des threads.
    - Affichez un message une fois qu'un fichier est téléchargé.
2. **Exercice 2 : Banque synchronisée**
    - Créez une classe CompteBancaire avec un solde initial.
    - Implémentez deux threads qui effectuent des retraits simultanément.
    - Synchronisez les méthodes pour éviter des retraits dépassant le solde.
3. **Exercice 3 : Producteur-Consommateur**
    - Implémentez une solution avec un thread producteur qui ajoute des éléments dans une liste partagée et un thread consommateur qui consomme ces éléments.