Série 4
Les threads

Exercice 1

Un compteur a un nom (Toto par exemple) et il compte de 1 à n (nombre entier positif quelconque). Il marque une pause aléatoire entre chaque nombre (de 0 à 5000 millisecondes par exemple).

Un compteur affiche chaque nombre (Toto affichera par exemple, "Toto : 3") et il affiche un message du type " Toto a fini de compter jusqu'à 10" quand il a fini.

Question 1 : Des threads indépendants

Ecrivez la classe compteur et testez-la en lançant plusieurs compteurs qui comptent jusqu'à 10. Voyez celui qui a fini le plus vite. Faites 2 versions : une où les threads sont créés avec une classe fille de Thread, et une où ils sont créés avec une instance d'une classe à part qui implémente Runnable.

Question 2 : Des threads un peu dépendants

Modifiez la classe Compteur pour que chaque compteur affiche son ordre d'arrivée : le message de fin est du type : "Toto a fini de compter jusqu'à 10 en position 3".

Question 3 : Des threads un peu plus dépendants

Modifiez la classe Compteur pour que chaque compteur donne un coup de pouce à un des autres compteurs (choisi au hasard) juste après avoir affiché un nombre : il interrompt la pause de cet autre compteur (pause entre l'affichage de 2 nombres), ce qui raccourcit le délai entre l'affichage de 2 nombres.

Exercice 2 : Un problème d'accès concurrent

Voici 2 classes Compte (correspond à un compte bancaire) et Operation (thread qui effectue des opérations sur un compte bancaire).

1. Examinez le code et faites exécuter la classe Opération. Constatez le problème : opération effectue des opérations qui devraient laisser le solde du compte inchangé, et pourtant, après un moment, le solde ne reste pas à 0. Expliquez.
2. Modifiez le code pour empêcher ce problème.
3. Dans le code de Operation, remplacez l'opération nulle par 2 opérations ajouter et retirer qui

devraient elles aussi laisser le solde du compte à 0 (elles sont en commentaire dans le code). Lancez l'exécution et constatez le problème. Modifiez le code pour que ça marche.

Exercice 3 (Tris parallèles)

Voici un algorithme de tri en ordre croissant d'une tranche de tableau comprise entre les éléments d'indices debut et fin :

```
trier(debut, fin) {
  si (fin - debut < 2) {
    si (t[debut] > t[fin])
      echanger(t[i], t[j])
  }
  sinon {
    milieu = (i + j) / 2
    trier(debut, milieu)
    trier(milieu + 1, fin)
    triFusion(milieu) // tri fusion des 2 moitiés du tableau
  }
}
```

On remarque que les 2 tris qui sont effectués avant la fusion sont indépendants l'un de l'autre et il est donc facile de les faire exécuter en parallèle par 2 threads.

On vous donne une version Java mono-tâche de cet algorithme. En utilisant la méthode join(), codez une version multi-tâche de cet algorithme. Si vous lancez des threads depuis la méthode main(), n'oubliez pas d'attendre la fin de leur exécution avant d'afficher le résultat final (ceux pour lequel le tableau "trié" s'affiche tel qu'il était au début, pas trié).

Codez une nouvelle version en utilisant cette fois-ci wait() - notify() au lieu de join().

Exercice 4 : Produit de 2 matrices en multi-tâches

Rappel : (Aij) . (Bkl) = (Pmn) avec Pmn = S (Amj . Bjn)

On remarque que les calculs des Pmn sont indépendants les uns des autres. On peut donc facilement effectuer ces calculs en parallèles.

Ecrivez une classe Matrice (n'implantez que les méthodes utiles pour effectuer le produit de 2 matrices en parallèle).

Annexe

Exercice 2 :

Classe Compte :

```
public class Compte {
  private int solde = 0;

  public void ajouter(int somme) {
    solde += somme;
    System.out.print(" ajoute " + somme);
  }

  public void retirer(int somme) {
    solde -= somme;
    System.out.print(" retire " + somme);
  }

  public void operationNulle(int somme) {
    solde += somme;
    System.out.print(" ajoute " + somme);
    solde -= somme;
    System.out.print(" retire " + somme);
  }

  public int getSolde() {
    return solde;
  }
}
```

Classe Operation :

```
public class Operation extends Thread {
  private Compte compte;

  public Operation(String nom, Compte compte) {
    super(nom);
    this.compte = compte;
  }

   public void run() {
    while (true) {
      int i = (int) (Math.random() * 10000);
      String nom = getName();
      System.out.print(nom);
//        compte.ajouter(i);
//        compte.retirer(i);
      compte.operationNulle(i);
      int solde = compte.getSolde();
      System.out.print(nom);
      if (solde != 0) {
      System.out.println(nom + ":**solde=" + solde);
      System.exit(1);
      }
    }
  }

  public static void main(String[] args) {
```

```
    Compte compte = new Compte();
    for (int i = 0; i < 20; i++) {
      Operation operation = new Operation("" + (char)('A' + i), compte);
      operation.start();
    }
  }
}
```

Exercice 3 :

Classe Trieur :

```
/**
 * Tri d'un tableau d'entiers
 * Version mono-thread
 */
public class Trieur {

  private int[] t;

  private Trieur(int[] t) {
    this.t = t;
  }

  /**
   * Trie un tableau d'entiers par ordre croissant
   * @param t tableau à trier
   */
  public static void trier(int[] t) {
    Trieur tableau = new Trieur(t);
    tableau.trier(0, t.length - 1);
  }

  /**
   * Trie une tranche de t
   * @param debut indice du début de la partie à trier
   * @param debut indice de la fin de la partie à trier
   */
  private void trier(int debut, int fin) {
    if (fin - debut < 2) {
      if (t[debut] > t[fin]) {
        echanger(debut, fin);
      }
    }
    else {
      int milieu = debut + (fin - debut) / 2;
      trier(debut, milieu);
      trier(milieu + 1, fin);
      triFusion(debut, fin);
    }
  }

  /**
   * Echanger t[i] et t[j]
   */
  private void echanger(int i, int j) {
    int valeur = t[i];
    t[i] = t[j];
```

```
    t[j] = valeur;
  }

  /**
   * Fusionne 2 tranches déjà triées du tableau t.
   *   - 1ère tranche : de debut à milieu
   *   - 2ème tranche : de milieu + 1 à fin
   * @param milieu indique le dernier indice de la 1ère tranche
   */
  private void triFusion(int debut, int fin) {
    // tableau où va aller la fusion
    int[] tFusion = new int[fin - debut + 1];
    int milieu = (debut + fin) / 2;
    // Indices des éléments à comparer
    int i1 = debut,
        i2 = milieu + 1;
    // indice de la prochaine case du tableau tFusion à remplir
    int iFusion = 0;
    while (i1 <= milieu && i2 <= fin) {
      if (t[i1] < t[i2]) {
        tFusion[iFusion++] = t[i1++];
      }
      else {
        tFusion[iFusion++] = t[i2++];
      }
    }
    if (i1 > milieu) {
      // la 1ère tranche est épuisée
      for (int i = i2; i <= fin; ) {
        tFusion[iFusion++] = t[i++];
      }
    }
    else {
      // la 2ème tranche est épuisée
      for (int i = i1; i <= milieu; ) {
        tFusion[iFusion++] = t[i++];
      }
    }
    // Copie tFusion dans t
    for (int i = 0, j = debut; i <= fin - debut; ) {
      t[j++] = tFusion[i++];
    }
  }

  public static void main(String[] args) {
    int[] t = {5, 8, 3, 2, 7, 10, 1};
    Trieur.trier(t);
    for (int i = 0; i <t.length; i++) {
      System.out.print(t[i] + " ; ");
    }
    System.out.println();
  }
}
```