Correction Série 4
Les threads

Exercice 1:

Question 1:

Version 1: La première version où les threads sont créés avec une classe fille de Thread.

```
public class Compteur extends Thread {
  private String nom;   // nom du Thread Compteur
  private int max;  // Le Compteur va afficher des nombres de 1 à max
  public Compteur(String nom, int max) {
    this.nom = nom;
    this.max = max;
  }

  public void run() {
    for (int i = 1; i <= max; i++) {
        System.out.println(nom + " : " + i);
        try {
            // Le Compteur marque une pause aléatoire ( de 0 et 5 secondes)
            // après chaque affichage
            sleep((int)(Math.random() * 5000));
        }catch(InterruptedException e) {
            System.err.println(nom + " a été interrompu.");
        }
    }
    System.out.println(nom + " a fini de compter jusqu'à " + max);
  }
```

```
    public static void main(String[] args) {
        Compteur toto=new Compteur("toto",10);
        Compteur titi=new Compteur("titi",20);
        toto.start();
        titi.start();
    }
}
```

Version 2: La deuxième version où les threads sont créés avec une instance d'une classe à part qui implémente l'interface Runnable.

```
public class Compteur implements Runnable{
  private String nom;   // nom du Thread Compteur
  private int max;  // Le Compteur va afficher des nombres de 1 à max
  public Compteur(String nom, int max) {
    this.nom = nom;
    this.max = max;
  }
  public void run() {
    for (int i = 1; i <= max; i++) {
        System.out.println(nom + " : " + i);
        try {
            // Le Compteur marque une pause aléatoire ( de 0 et 5 secondes)
            // après chaque affichage
            Thread.currentThread().sleep((int)(Math.random() * 5000));
        }catch(InterruptedException e) {
            System.err.println(nom + " a été interrompu.");
        }
    }
```

```
    System.out.println(nom + " a fini de compter jusqu'à " + max);
  }
     public static void main(String[] args) {
       Compteur toto=new Compteur("toto",10);
       Thread totoThread=new Thread(toto);
       Compteur titi=new Compteur("titi",20);
       Thread titiThread=new Thread(titi);
       totoThread.start();
       titiThread.start();
    }
}
```

NB: Pour la suite de ce TP, nous allons travailler avec la première version pour la creation des threads (i.e : les threads qui sont créés avec une classe fille de Thread.).

Question 2:

```
public class Compteur extends Thread {
  private String nom;   // nom du Thread Compteur
  private int max;  // Le Compteur va afficher des nombres de 1 à max
  private static int pos=0; //La position de terminaison de chaque Thread
  public Compteur(String nom, int max) {
    this.nom = nom;
    this.max = max;
  }
  public void run() {
    for (int i = 1; i <= max; i++) {
       System.out.println(nom + " : " + i);
       try {
          // Le Compteur marque une pause aléatoire ( de 0 et 5 secondes)
          // après chaque affichage
          sleep((int)(Math.random() * 5000));
```

```
        }catch(InterruptedException e) {
            System.err.println(nom + " a été interrompu.");
        }
    }
    pos++;
    System.out.println(nom + " a fini de compter jusqu'à " + max+" en position "+pos);
  }

     public static void main(String[] args) {
        Compteur toto=new Compteur("toto",10);
        Compteur titi=new Compteur("titi",20);
        toto.start();
        titi.start();
    }
}
```

Question 3:

```
public class Compteur extends Thread {
  private String nom;   // nom du Thread Compteur
  private int max;  // Le Compteur va afficher des nombres de 1 à max
  public Compteur(String nom, int max) {
    this.nom = nom;
    this.max = max;
  }
  public void run() {
    for (int i = 1; i <= max; i++) {
        System.out.println(nom + " : " + i);
        yield();
    }
    System.out.println(nom + " a fini de compter jusqu'à " + max);
```

```
  }
     public static void main(String[] args) {
       Compteur toto=new Compteur("toto",10);
       Compteur titi=new Compteur("titi",20);
       toto.start();
       titi.start();
    }
}
```

Question 4: (Question supplémentaire) :

Modifier la classe Compteur pour que chaque Compteur donne la main à un autre Compteur sauf s'il a terminé de compter jusqu'à n.

```
public class classObject {
   public classObject() {
   }
}

public class Compteur extends Thread {
  private String nom;   // nom du Thread Compteur
  private int max;  // Le Compteur va afficher des nombres de 1 à max
  private classObject obj;
  public Compteur(String nom, int max, classObject obj) {
    this.nom = nom;
    this.max = max;
    this.obj=obj;
  }
  public void run() {
    synchronized (obj){
    for (int i = 1; i <= max; i++) {
        System.out.println(nom + " : " + i);
```

```
        try {
            // Le Compteur marque une pause aléatoire ( de 0 et 5 secondes)
            // après chaque affichage
            sleep((int)(Math.random() * 5000));
        }catch(InterruptedException e) {
            System.err.println(nom + " a été interrompu.");
        }
    }
    System.out.println(nom + " a fini de compter jusqu'à " + max);
  }
  }
     public static void main(String[] args) {
       //Création d'une instance de la classe classObject
       //afin que les Threads créés partage le même objet "objPartage"
       classObject objPartage=new classObject();
       Compteur toto=new Compteur("toto",10,objPartage);
       Compteur titi=new Compteur("titi",20,objPartage);
       toto.start();
       titi.start();
    }
}
```

Exercice 2:

Question 1:

Après l'exécution de la classe Operation, vous remarquez que le solde n'est pas resté à 0, bien que la classe Operation appelle juste la méthode operationNulle de la classe Compte (cette méthode ajoute et retire la même somme au solde).

Question 2:

Il faut mettre la méthode operationNulle à synchronized.

NB : Il faut aussi mettre la méthode getSolde à synchronized. (Il faut éviter que getSolde ne soit exécutée en même temps que operationNulle).

```
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 < 2; i++) {
      Operation operation = new Operation("" + (char)('A' + i), compte);
```

```
      operation.start();
    }
  }
}

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

Exercice 3:

Question 1:

```
/**
```

```
 * Tri d'un tableau d'entiers multi-thread.
 * Version qui utilise join().
 */
public class Trieur extends Thread {
  private int[] t; // tableau à trier
  private int debut, fin; // tranche de ce tableau qu'il faut trier
  public Trieur(int[] t) {
    this(t, 0, t.length - 1);
  }
  private Trieur(int[] t, int debut, int fin) {
    this.t = t;
    this.debut = debut;
    this.fin = fin;
  }
  public void run() {
    if (fin - debut < 2) {
      if (t[debut] > t[fin]) {
        echanger(debut, fin);
      }
    }
    else {
      int milieu = debut + (fin - debut) / 2;
      Trieur trieur1 = new Trieur(t, debut, milieu);
      trieur1.start();
      Trieur trieur2 = new Trieur(t, milieu + 1, fin);
      trieur2.start();
      try {
        trieur1.join();
        trieur2.join();
```

```
    }
    catch(InterruptedException e) {}
    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 trieur = new Trieur(t);
    trieur.start();
    try {
```

```
      trieur.join();
    }
    catch(InterruptedException e) {}
    for (int i = 0; i <t.length; i++) {
      System.out.print(t[i] + " ; ");
    }
    System.out.println();
  }
}
```

Question 2:

```
/**
 * Tri d'un tableau d'entiers multi-thread.
 * Utilisation de wait() et notify() au lieu de join()
 */
public class Trieur extends Thread {
  private int[] t; // tableau à trier
  private int debut, fin; // tranche de ce tableau qu'il faut trier
  private Trieur parent;  // thread Trieur qui a lancé ce (this) Trieur
  private int nbNotify = 0; // La Condition est materialisee ainsi
                            //:"nombre de notifications de terminaison=2"
                            // Initialement, la condition est fausse (nbNotify=0)

  public Trieur(int[] t) {
    this(null, t, 0, t.length - 1);
  }
  private Trieur(Trieur parent, int[] t, int debut, int fin) {
    this.parent = parent;
    this.t = t;
    this.debut = debut;
```

```
  this.fin = fin;
 }
 public synchronized void notifier() {
    this.nbNotify++;
    this.notifyAll();
   /**
     * Cette méthode Notifie tout les Threads en attente sur ce (this) moniteur
     * Attention, quand le message sera envoyé au parent (dans run()),
     * on incrémentera la variable nbNotify du parent (qui sera le this implicite)
     * et on notifiera le parent.
     */
 }
 public void run() {
  if (fin - debut < 2) {
    if (t[debut] > t[fin]) {
      echanger(debut, fin);
    }
  }
  else {
    int milieu = debut + (fin - debut) / 2;
    Trieur trieur1 = new Trieur(this, t, debut, milieu);
    Trieur trieur2 = new Trieur(this, t, milieu + 1, fin);
    trieur1.start();
    trieur2.start();
    // attend les 2 threads fils par le biais du test d'une condition
    // qui, si non verifiee, entraine l'utilisation de wait() sur
    // le moniteur associe implicitement a l'objet courant (càd à this)
    // jusqu'à ce qu'elle soit verifiée
    synchronized(this) {
```

```
      try {
          // Tant que deux notifications n'ont pas été reçues, on attend
        while (nbNotify <2)
           wait();
       }catch(InterruptedException e) {}
    }
    triFusion(debut, fin);
  }
  if(parent!=null){
     parent.notifier();// indique qu'il a fini au parent qu'il attend
  }
  }
/**
 * 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 = (debut + fin) / 2
 *   - 2ème tranche : de milieu + 1 à fin
 * @param debut premier indice de la 1ère tranche
 * @param fin dernier indice de la 2ème 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 trieur = new Trieur(t);
    trieur.start();
    try { // on continue d'utiliser un join() pour etre sur que le tri
            // complet est termine avant d'afficher le resultat du tri
      trieur.join();
    }
    catch(InterruptedException e) {}
    for (int i = 0; i <t.length; i++) {
      System.out.print(t[i] + " ; ");
    }
    System.out.println();
  }
}
```

Exercice 4:

```
public class CalculThread extends Thread{
    private int [][] A;
    private int [][] B;
    private int [][] P;
    private int m;
    private int n;
    private int colA;
    public CalculThread(int[][] A,int[][] B,int[][] P, int colA, int m,int n) {
        this.A=A;
```

```
        this.B=B;
        this.P=P;
        this.colA=colA;
        this.m=m;
        this.n=n;
    }
    public void run(){
        calculElement();
    }
    public synchronized void calculElement(){
        for (int i=0; i<colA;i++){
            P[m][n]=P[m][n]+A[m][i]*B[i][n];
        }
    }
}

public class Main {
    public static void main(String[] args) {
        int [][] A= {{1,2,3} ,{ 1, 2, 3}};
        int [][] B= {{1,2,3} ,{ 1, 2, 3},{1, 2, 3}};
        int ligneA = A.length; // Nombre de lignes de A
        int colA = A[0].length; // Nombre de colonnes de A
        //NB: le nombre de lignes de B doit être égale au nombre de colonne de A
        int colB = B[0].length; // Nombre de colonnes de B
        int [][] P = new int [ligneA][colB];// Matrice produit
        for(int m=0; m<ligneA;m++){
            for (int n=0; n<colB;n++){
                CalculThread c=new CalculThread(A,B,P,colA,m,n);
                c.start();
```

```
            }
        }
        // Affichage de la matrice Produit
        for(int i=0;i<ligneA;i++){
            for(int j=0;j<colB;j++){
                System.out.print(P[i][j]+"   ");
            }
            System.out.println();
        }
    }
}
```