EST Salé
Département Informatique
LP BigData

Programmation Concurrente en Java
TP N°1 Thread

Le multithreading
Le multithreading est une fonctionnalité Java qui permet l'exécution simultanée de deux ou plusieurs parties d'un programme pour une utilisation maximale de l'unité centrale. Chaque partie d'un tel programme est appelée un thread. Les threads sont donc des processus légers au sein d'un processus.

Les threads peuvent être créés en utilisant deux mécanismes :

1. L'extension de la classe Thread.
2. Implémentation de l'interface Runnable

Exercice 1 : création des threads (extends Thread)

Écrivez un programme java qui permet de créer cinq threads en étendant la classe Thread.
Chaque thread affiche le message suivant :
Je suis le thread d'id : id_thread, je suis en cours d'exécution. Création en étendant la classe Thread

Exemple d'exécution

```
Création des threads par Héritage de la classe Thread
Je suis le thread d'id :  21 en cours d'exécution. Création en étendant la classe Thread
Je suis le thread d'id :  24 en cours d'exécution. Création en étendant la classe Thread
Je suis le thread d'id :  23 en cours d'exécution. Création en étendant la classe Thread
Je suis le thread d'id :  22 en cours d'exécution. Création en étendant la classe Thread
Je suis le thread d'id :  25 en cours d'exécution. Création en étendant la classe Thread
```

Exercice 2 : création des threads (implements Runnable)

Écrivez un programme java qui permet de créer cinq threads en étendant la classe Thread.
Chaque thread affiche le message suivant :
Je suis le thread d'id : id_thread, je suis en cours d'exécution. Création en étendant la classe Thread

Exemple d'exécution

```
Création des threads par utilisant une interface exécutable
Je suis le thread d'id : 21 je uis en cours d'exécution. Création en utilisant implements Runnable
Je suis le thread d'id : 23 je uis en cours d'exécution. Création en utilisant implements Runnable
Je suis le thread d'id : 24 je uis en cours d'exécution. Création en utilisant implements Runnable
Je suis le thread d'id : 25 je uis en cours d'exécution. Création en utilisant implements Runnable
Je suis le thread d'id : 22 je uis en cours d'exécution. Création en utilisant implements Runnable
```

Exercice 3 : Nom et priorité d'un thread

Créez une nouvelle classe Test en étendant la classe Thread.
Dans la fonction main de la classe Test :
Créez un objet t (une instance de la classe Test)
Affichez le nom du thread (getName())
Renommez le nom du thread en Principal. Affichez le nouveau nom du thread

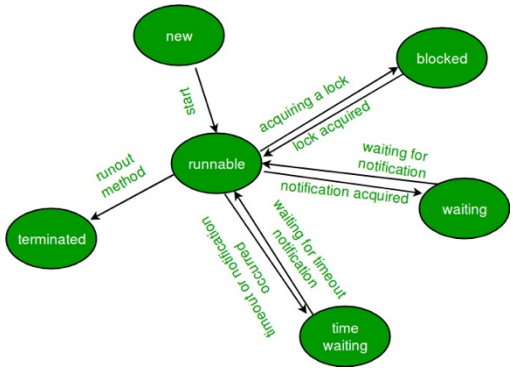
Affichez la priorité du thread (getPriority())
Changez la priorité à MAX_PRIORITY du thread en Principal. Affichez la nouvelle priorité nom du thread

Dans la fonction main, créez cinq nouveau thread (new Thread()). Chaque thread affiche son nom et sa priorité. Changez la priorité de chaque thread à MIN_PRIORITY puis affichez le nom de la nouvelle priorité de chaque thread

Exercice 4 : Cycle de vie et états d'un thread en Java

Un thread en Java à tout moment existe dans l'un des états suivants :

- Nouveau (New)
- Exécutable (Runnable)
- Bloqué (Blocked)
- En attente (Waiting)
- En attente temporisée (Timed Waiting)
- Terminé (Terminated)



Testez le code suivant

```
// Programme Java pour démontrer les états des threads
class thread implements Runnable {
    public void run()
    {
        // moving thread2 to timed waiting state
        try { Thread.sleep(1500);}
        catch (InterruptedException e) {
            e.printStackTrace();}

        System.out.println(
            "State of thread1 while it called join() method on thread2 -"
            + Test.thread1.getState());
        try { Thread.sleep(200);}
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public class Test implements Runnable {
    public static Thread thread1;
    public static Test obj;

    public static void main(String[] args)
    {
        obj = new Test();
        thread1 = new Thread(obj);

        // thread1 created and is currently in the NEW
        // state.
        System.out.println(
            "State of thread1 after creating it - "
```

```
            + thread1.getState());
        thread1.start();

        // thread1 moved to Runnable state
        System.out.println(
            "State of thread1 after calling .start() method on it - "
            + thread1.getState());
    }

    public void run()
    {
        thread myThread = new thread();
        Thread thread2 = new Thread(myThread);

        // thread1 created and is currently in the NEW
        // state.
        System.out.println(
            "State of thread2 after creating it - "
            + thread2.getState());
        thread2.start();

        // thread2 moved to Runnable state
        System.out.println(
            "State of thread2 after calling .start() method on it - "
            + thread2.getState());

        // moving thread1 to timed waiting state
        try {
            // moving thread1 to timed waiting state
            Thread.sleep(200);
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(
            "State of thread2 after calling .sleep() method on it - "
            + thread2.getState());

        try {
            // waiting for thread2 to die
            thread2.join();
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println(
            "State of thread2 when it has finished it's execution - "
            + thread2.getState());
         }
}
```

Exercice 5

Soit la classe suivante :

```
public class Alphabet {
   public void affiche() {
      for (char a = 'A'; a <= 'Z'; a++) {
         System.out.print(a);
         try { Thread.sleep(10); // ms
         } catch (InterruptedException e) {}
}

      System.out.print("\n");
   }
   public static void main(String args[]) {
      Alphabet A = new Alphabet();
      A.affiche();
   }
}
```

Modifier cette classe en utilisant l'interface Runnable pour que l'affichage se fasse dans un thread séparé. On pourra donc écrire le
programme de test suivant :

```
AlphabetThread A1 = new AlphabetThread();
Thread T1 = new Thread(A1);
AlphabetThread A2 = new AlphabetThread();
Thread T2 = new Thread(A2);
T1.start();
T2.start();
```

Exercice 6 : Des threads indépendants

Un "compteur" a un nom (*Programmation Concurrente* 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 itération (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.

Ecrivez la classe compteur et testez-la en lançant plusieurs compteurs qui comptent jusqu'à 10. Voyez celui qui a fini le plus vite.

Exercice 7 : 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".