Compte rendu TP1 :

Initiation thread java

Réalisé par :

Mtibaa Amal
Mhiri Yasmine
Riahi Mohamed Wassim

GL4/groupe1

2017-2018

Exercice 1 :

La configuration logicielle et matérielle de la machine utilisée pour cet exercice :

- Processeur : Inter(R) Core(™) i7
- RAM : 6G
- SE : Windows 10 64bits
- IDE : Eclipse java photon

1) Compilons les deux programmes disposés dans le répertoire exo1 et modifions le path du fichier log.txt.
2) Exécutons le programme Main :

```
Minimum Priority: 1
Normal Priority: 5
Maximun Priority: 10
Thread 'My Thread 0': START
Thread 'My Thread 2': START
Thread 'My Thread 4': START
Thread 'My Thread 6': START
Thread 'My Thread 4': END. Number of Primes: 2263
Thread 'My Thread 2': END. Number of Primes: 2263
Thread 'My Thread 8': START
Thread 'My Thread 0': END. Number of Primes: 2263
Thread 'My Thread 6': END. Number of Primes: 2263
Thread 'My Thread 1': START
Thread 'My Thread 3': START
Thread 'My Thread 5': START
Thread 'My Thread 7': START
Thread 'My Thread 9': START
Thread 'My Thread 8': END. Number of Primes: 2263
Thread 'My Thread 1': END. Number of Primes: 2263
Thread 'My Thread 5': END. Number of Primes: 2263
Thread 'My Thread 3': END. Number of Primes: 2263
Thread 'My Thread 7': END. Number of Primes: 2263
Thread 'My Thread 9': END. Number of Primes: 2263
```

- Extrait du fichier log.txt :

```
Main : Status of Thread 0 : NEW
Main : Status of Thread 1 : NEW
Main : Status of Thread 2 : NEW
Main : Status of Thread 3 : NEW
Main : Status of Thread 4 : NEW
Main : Status of Thread 5 : NEW
Main : Status of Thread 6 : NEW
Main : Status of Thread 7 : NEW
Main : Status of Thread 8 : NEW
Main : Status of Thread 9 : NEW
Main : Id 10 - My Thread 0
Main : Priority: 10
Main : Old State: NEW
Main : New State: RUNNABLE
Main : ************************************
Main : Id 11 - My Thread 1
Main : Priority: 1
Main : Old State: NEW
Main : New State: RUNNABLE
Main : ************************************
Main : Id 12 - My Thread 2
Main : Priority: 10
Main : Old State: NEW
Main : New State: TERMINATED
Main : ************************************
Main : Id 13 - My Thread 3
Main : Priority: 1
Main : Old State: NEW
Main : New State: RUNNABLE
Main : ************************************
```

=> Le rôle de ce programme :

- La création d'un tableau de 10 threads, tels que ceux qui ont un indice impair ont la même priorité minimale égale à 1 et ceux ayant un indice pair ont la même priorité maximale égale à 10.
- Chaque thread va appeler la classe 'Calculator' qui va démarrer le thread en calculant le nombre des entiers premiers compris entre 1 et 20000 et puis l'afficher dans la console.
- Garder les traces des états des threads en précisant leurs états ancien et nouveau, ainsi que leur priorité et l'enregistrer dans le fichier 'log.txt'.

3) a/ Les attributs d'un thread:

- name
- priority
- state

b/ - valeur de la priorité maximale=10

- valeur de la priorité normale = 5
- valeur de la priorité minimale = 1

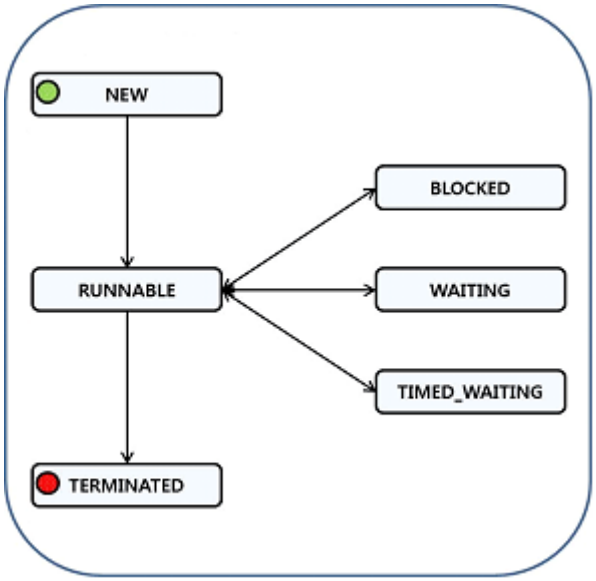
c/ - La priorité des threads ayant un indice pair = 10

- La priorité des threads ayant un indice impair= 1

d/ Le format de nommage par défaut d'un thread : 'Thread-i' avec i in [0,9] dans notre cas.

e/ - Les états d'un thread :

- NEW
- RUNNABLE
- TERMINATED



f/ - On peut modifier le nom d'un thread ainsi que son état. En effet, Java permet d'arrêter (**stop**), de faire attendre (**sleep, yield**), d'interrompre (**interrupt**), de changer la priorité(**setPriority**), de détruire(**destroy**) des threads à travers les méthodes de la classe Thread.

g/ - Pour savoir le groupe d'un thread, exécutons l'instruction suivante:

```
System.out.println(threads[0].getThreadGroup());
```

-> `java.lang.ThreadGroup[name=main,maxpri=10]`

- Par défaut, un thread est créé dans le groupe courant (c'est à dire de celui qui l'a créé). Au démarrage, un thread est créé dans le groupe de nom main.
- On peut créer de nouveaux groupes et créer des threads appartenant à ces groupes par de nouveaux constructeurs:

```
ThreadGroup Groupe = new ThreadGroup("Mon groupe");
Thread Processus = new Thread(Groupe, "Un processus");
```

Exercice 2 :

La configuration logicielle et matérielle de la machine utilisée pour cet exercice :

- Processeur : Inter(R) Core(™) i7
- RAM : 8G
- SE : Windows 8.1 64bits
- IDE : IntelliJ IDEA

4) pour rediriger la sortie vers un fichier log il suffit de créer une variable globale de type PrintWriter puis de l'instancier au début du main comme suit :

-

pw=new PrintWriter (new BufferedWriter(new FileWriter
("C:\\Users\\HP\\Desktop\\output.txt",true)));
Et enfin de remplacer tous les System.out.println(); par des pw.println();

5) Ce programme lance 3 thread en parallèles et chaque thread affiche un message qui lui est spécifique. Exemple d'exécution :

```
Je suis le main :)
main terminé
1 Bonsoir
1 Bonjour
1 à demain
2 à demain
3 à demain
2 Bonjour
4 à demain
3 Bonjour
5 à demain
4 Bonjour
6 à demain
5 Bonjour
7 à demain
6 Bonjour
8 à demain
7 Bonjour
9 à demain
8 Bonjour
10 à demain
9 Bonjour
11 à demain
10 Bonjour
12 à demain
11 Bonjour
13 à demain
12 Bonjour
14 à demain
13 Bonjour
```

6) On remarque que le main lance les 3 thread puis continue son exécution. Ensuite chaque thread exécute une partie de son code et ils alternent entre eux.

7) Lorsqu'on décommente l'instruction System.exit(0); le main va lancer les 3 thread puis il va continuer son exécution et il va exécuter l'instruction System.exit(0) qui va forcer l'arrêt du programme et de toutes les threads qui sont reliées au programme.

8) Lorsqu'on exécute on remarque que maintenant on n'a que les messages du main qui s'affiche mais pas ceux des thread puisque le programme s'est arrêté avant qu'ils ne puissent prendre la main.

9)

```
thread1.start();
thread2.start();
thread3.start();
try {
    thread1.join();
    thread2.join();
    thread3.join();
} catch (InterruptedException e) {
    e.printStackTrace();
}
```

Après avoir exécuté on constate que maintenant le main ne finit son exécution qu'après la fin des 3 thread. La fonction join met le thread courant (dans notre cas le main) en mode wait jusqu'à la fin de l'exécution du thread qui a appelé la méthode join (dans notre cas thread1 , thread2 et thread3).

10) Il est possible de lancer plusieur thread en parallèle depuis le main. Ces thread auront comme parent le main thread. Après avoir lancé les différent thread, le main, étant plus prioritaire, continue son exécution jusqu'à la fin mais le programme ne se termine pas car il doit attendre la fin de l'exécution de ces thread fils (sauf si on force l'arrêt du programme avec un System.exit(0), cela force l'arrêt de tous les thread et du programme). Le main va donc exécuter toute ces instruction puis il va céder la main aux thread et attendre la fin de leurs exécution pour terminer le programme.
Il est possible de faire en sorte que le thread main doit attendre la fin de l'exécution de ces thread fils avant de pouvoir finir l'exécution de ces instructions et cela avec la fonction join().

Exercice 3 :

La configuration logicielle et matérielle de la machine utilisée pour cet exercice :

1) Processeur : Inter(R) Core(™) i7
2) RAM : 8G
3) SE : Windows 10 64bits
4) IDE : Eclipse JAVA Neon

ConcurrentLinkedQueue est une file d'attente thread-safe non limitée basée sur des nœuds liés. Lorsque de nombreux threads partageront l'accès à une collection commune, cette classe n'autorise pas l'utilisation d'éléments nuls. ConcurrentLinkedQueue effectue toutes les opérations de verrouillage et autres dont on a besoin en interne ; les producteurs ajoutent des données à la file d'attente et le consommateur les traite.

Le but de cet exercice est de créer un gestionnaire de tâches qui met en oeuvre 2 types de thread:

- ✓ WriterTask qui crée et stocke des tâches dans le queue de type ConcurrentLinkedQueue
- ✓ CleanerTask qui traite les taches et les supprime ; si une tâche devient ancienne (création date de plus de 10 secondes) elle sera supprimée. Après chaque suppression le thread démon affichera la nouvelle taille de la queue.

On définit alors 5 classes :

- ❖ **Event :**représente une tâche, caractérisé par un string event et la date de création
- ❖ **WriterTask**: représente un thread qui écrit dans le queue.

- **CleanerTask :** traite les tâches et les supprime s'ils dépassent 10 seconde à partir de la date de création.
- **TestMain:** pour tester les threads0
- **TestMachinePropreties :** pour vérifier le nombre de cœurs dans la machine utilisé.

Implémentation des classes :

Classe Event : on définit dans cette classe les attributs date et event avec un constructeur et des getters et des setters

```
import java.util.Date;
    public class Event {
    private Date date;
    private String event;

    public Event(Date date,String event){
        this.date=date;
        this.event=event;
    }
    public Date getDate() {
    return date;
    }
    public void setDate(Date date) {
    this.date = date;
    }
    public String getEvent() {
    return event;
    }
    public void setEvent(String event) {
    this.event = event;
    }
```

Classe WriterTask:

On passe au WriterTask la queue de type ConcurentLinkedQueue, dans le run on écrit 100 event.

```java
import java.util.Date;
import java.util.concurrent.ConcurrentLinkedQueue;
//Classe WriterTask
public class WriterTask implements Runnable {

    //concurrentLinkedQueue est la structure qui contient les evénements
    private ConcurrentLinkedQueue<Event> concurrentLinkedQueue;

    public WriterTask(ConcurrentLinkedQueue<Event> concurrentLinkedQueue) {
        this.concurrentLinkedQueue = concurrentLinkedQueue;
        }

    public void run() {
    // Writes 100 events in deque
    try {
        for(int i=0;i<100;i++){
            Event event=new Event(new Date(),"Event coming from "+
                    Thread.currentThread().getName()+" with i= "+i);
            concurrentLinkedQueue.offer(event);
            System.out.println("Adding new Event ="+event.getEvent());
            Thread.currentThread();
            Thread.sleep(100);
        }
    } catch (InterruptedException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    }
}
```

Classe CleanerTask

On passe aussi la queue de type ConcurentLinkedQueue au cleanerTask, il prend le prend le premier élément de la queue (head). Si sa période a déplacé 10s, on le supprime,
Si non, on l'ajoute à la fin de la queue.

```java
import java.util.Date;
import java.util.concurrent.ConcurrentLinkedQueue;
public class CleanerTask extends Thread {
    private ConcurrentLinkedQueue<Event> concurrentLinkedQueue;

    public CleanerTask(ConcurrentLinkedQueue<Event> concurrentLinkedQueue){
        this.concurrentLinkedQueue=concurrentLinkedQueue;
    }
    public void run() {
        while(true){
            Date date = new Date();
            clean(date);
        }
    }
    public  void clean(Date date){
        try{
        int size=concurrentLinkedQueue.size();
        for (int i=0;i<size;i++){
            Event headEvent=this.concurrentLinkedQueue.poll();
            System.out.println("Traitement Event"+headEvent.getEvent());
            if((date.getTime()-headEvent.getDate().getTime())/1000<10) //if period<10 , add in tail
                this.concurrentLinkedQueue.add(headEvent);
            }
        System.out.println("At the date :"+date.getTime()+" the size of the queue is : "+
            this.concurrentLinkedQueue.size());
        Thread.currentThread();
        Thread.sleep(100);
        }
        catch (InterruptedException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }}
}
```

Classe TestMachinePropreties

```java
import java.util.Enumeration;
import java.util.Hashtable;

import static java.lang.System.out;

public class TestMachineProprties {

        public static void main(String ...args) {
                out.println("Properties : ");
                Hashtable<Object, Object> props = System.getProperties();
                Enumeration<Object> keys = props.keys();
                while(keys.hasMoreElements()) {
                        Object key = keys.nextElement();
                        out.println(key + " : " + props.get(key));
                }
                out.println("availableProcessors : " + Runtime.getRuntime().availableProcessors());
                out.println("NUMBER_OF_PROCESSORS : " + System.getenv("NUMBER_OF_PROCESSORS"));
        }
}
```

Exécution

```
sun.io.unicode.encoding : Unicc
sun.cpu.endian : little
sun.desktop : windows
sun.cpu.isalist : amd64
availableProcessors : 4
NUMBER_OF_PROCESSORS : 4
```

On a ici 4 cœurs.

Classe TestMain : on crée 4 threads de type WriteTask avec un seul cleanerTask

```
import java.util.concurrent.ConcurrentLinkedQueue;

public class TestMain {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

          ConcurrentLinkedQueue<Event> queue = new ConcurrentLinkedQueue<Event>();
          Thread writerTask1 = new Thread(new WriterTask(queue));
          Thread writerTask2 = new Thread(new WriterTask(queue));
          Thread writerTask3 = new Thread(new WriterTask(queue));
          Thread writerTask4 = new Thread(new WriterTask(queue));

          Thread cleanerTask = new Thread(new CleanerTask(queue));


          writerTask1.start();
          writerTask2.start();
          writerTask3.start();
          writerTask4.start();

          cleanerTask.start();
    }
}
```

Exemple d'exécution :

```
Adding new Event =Event coming from Thread-2 with i= 0
Adding new Event =Event coming from Thread-0 with i= 0
Adding new Event =Event coming from Thread-3 with i= 0
Adding new Event =Event coming from Thread-1 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
At the date :1550793352222 the size of the queue is : 4
Traitement EventEvent coming from Thread-2 with i= 0
Adding new Event =Event coming from Thread-3 with i= 1
Adding new Event =Event coming from Thread-1 with i= 1
Traitement EventEvent coming from Thread-2 with i= 0
Adding new Event =Event coming from Thread-0 with i= 1
Adding new Event =Event coming from Thread-2 with i= 1
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
At the date :1550793352322 the size of the queue is : 8
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
Traitement EventEvent coming from Thread-2 with i= 0
```