**Exemple 03 : Threads ou les Processus Légers**

D'une manière générale une application est constituée d'un ou de plusieurs processus qui doivent communiquer entre eux afin de réaliser une tâche (simple ou compliquée) bien définie. Ceci nous mis dans une situation de la programmation concurrentielle (il faut de la communication et de la synchronisation).

Le langage Java offre aussi la possibilité de créer cette situation au sein même d'un seul processus, on peut créer des sous-processus à l'intérieur d'un processus. Ces sous-processus sont connus par le nom : t*hread*. Par conséquent, on déduit qu'un processus contient au moins un thread, c'est le *thread principal* (*main thread*). Ce dernier a la possibilité de créer d'autres threads.

Dans cette séance, on réalisera quelques exemples pour la manipulation des threads en Java (création et lancement)

***N.B. :***

Bien évidemment, dans le T.P. on utilisera les threads pour permettre à un processus (qui représente un véhicule) de réaliser une autre action en parallèle avec la tâche principale de ce processus (qui est de faire rouler le véhicules).

***La premier programme***

Dans cet exemple (simple), on va voir comment créer un thread. Dans Java un thread est une instance de la classe *Thread* qui incorpore une instance de l'interface *Runnable*. Runnable est une interface (classe spéciale contenant uniquement des méthodes et qui sont non implémentées) que nous devons implémenter avant de l'instancier. Cette interface (Runnable) contient une seule méthode qui est *run()*.

- ✗ La première étape est de créer une nouvelle classe qui implémente l'interface Runnable ;
- ✗ la seconde est de créer une instance de Thread à partir ce la classe créer précédemment ;
- ✗ finalement, on lance le thread avec la méthode *start()*.

Dans ce premier exemple, le processus principale affiche périodiquement dans la sortie standard (**System.out**) une message qui indique qu'il est toujours en cours de l'exécution. En parallèle à ça, un autre thread affiche aléatoirement sur la sortie des erreurs

(**System.err**) un autre message qui indique qu'il y a un *parallélisme* au sein du même processus.

***L'implémentation de l'interface Runnable (ImpRunnable.java) :***

```
public class ImplRunnable implements Runnable{
	public void run(){
		Random random = new Random();

		while (true){
			try{
			 Thread.sleep(random.nextInt(3001)+1000);
				System.err.println("Je suis le thread parallele !!!");
			}
			catch (Exception e){
				e.printStackTrace();
			}
		}
	}
}
```

***La méthode principale (MainExemple1.java):***

```
public static void main(String args[]) {
	Thread thread = new Thread(new ImplRunnable()); // la creation du Trehad
	thread.start();   // lancement du thread

	try{
		while (true){
		  System.out.println("Hello world !!! : Je suis toujours en execution ...");
		  Thread.sleep(1000); // attente d'une seconde...
		}
	}
	catch (Exception e){
		e.printStackTrace();
	}
}
```

***La deuxième programme***

Ce deuxième exemple est adaptation du premier exemple de telle sorte qu'il y aura :

- ✗ Création (aléatoire) de plusieurs threads et leur lancement ;
- ✗ Le thread principale (processus) affiche périodiquement un message accompagné avec le nombre de threads existant
- ✗ Chaque thread peut s'arrêter d'une manière aléatoire
- ✗ Un thread spéciale a pour rôle de lancer les autres threads
- ✗ Chaque thread possède un nom unique

Dans ce programme, il y deux types de thread : le premier (aucun affichage sur la console) est un lanceurs de threads de second type. Les threads de deuxième type affichent un message avec leurs noms et il quitte leurs exécution (avec un message Bye Bye !!!) avec

une probabilité de *0.3*.

***LancementRunnable.java***

```
public class LancementRunnable implements Runnable{
	private static int nbreThread = 0;
	private static int cptThread = 0;


	public LancementRunnable() {
		Thread thread = new Thread(this);
		thread.start();
	}

	public void run(){
		Random random = new Random();

		while (true){
			try{
				Thread.sleep( random.nextInt(5001)+2000 );
				Thread th = new Thread(new ImplRunnable(), "Thread "+cptThread);

				nbreThread++;
				cptThread++;

				th.start();
			}
			catch (Exception e){
				e.printStackTrace();
			}
		}
	}

	// Les accesseurs et les modificateurs ...
	public static int getNbreThread() {
		return nbreThread;
	}

	public static void setNbreThread(int nbreThread) {
		LancementRunnable.nbreThread = nbreThread;
	}

	public static void decrementerThread(){
		nbreThread--;
	}

	public static int getCptThread() {
		return cptThread;
	}
}
```

***ImplRunnable.java***

```
public class ImplRunnable implements Runnable{
	public void run(){
		Random random = new Random();

		String myName = Thread.currentThread().getName();
		while (true){
			try{
				Thread.sleep(random.nextInt(3001)+1000);
```

```
                        System.err.println(myName+" : Je suis le thread parallele !!!");

                        if (random.nextInt(10)<3){ // quitter la boucle une p = 0.3
                              LancementRunnable.decrementerThread();
                              System.err.println(myName+" : Bye Bye !!!");
                              break;
                        }
                  }
                  catch (Exception e){
                        e.printStackTrace();
                  }
            }
      }
}
```

***MainExemple2.java***

```
public static void main(String args[]) {
      try{
            new LancementRunnable();

            while (true){
                  System.out.println("Hello world !!! : Le nombre de threads en cours
                                    d'execution : "+LancementRunnable.getNbreThread()+".
                                    Total cree : "+LancementRunnable.getCptThread());
                  Thread.sleep(1000);
            }
      }
      catch (Exception e){
                  e.printStackTrace();
      }
}
```

***Récapitulatif***

Les threads nous permettent de réaliser plusieurs tâches en parallèle dans un même processus. Dans le T.P. à réaliser, nous n'utiliserons pas les threads pour représenter les entités concurrentes (on utilisera à la place des processus). Par contre, l'utilisation de threads sera utile pour implémenter la réception de messages d'une manière indépendante du processus lui même.