IUT de Blois

R309 - Programmation événementielle

université de TOURS

Les threads

Exercice 1 (Création de threads) :

1. Comment démarrer un thread en Java ?
2. Citer les deux méthodes pour la création d'un thread ? Quelle est la différence entre ces deux méthodes ?
3. Quelle est la méthode principale d'un thread ?
4. Ecrivez en java un programme qui utilise deux threads en parallèle :

   ✓ Le premier affichera les 26 lettres de l'alphabet ;
   ✓ Le second affichera les nombres de 1 à 26.

Exercice 2 (Méthodes de la classe Thread) :

1. *Soit les classes suivantes :*
   - *Exercice1* qui contient une méthode point d'entrée
   - *PremierThread* et *SecondThread* qui définissent des Threads

   a- Affichez ce qui sera écrit dans la console après l'exécution du programme correspondant à la classe Exercice1.

```
public class Exercice1 {

	public static void main(String[] args) {
		PremierThread pth = new PremierThread("Thread 1");
		SecondThread sth = new SecondThread("Thread 2");
		pth.start();
		sth.start();
	}
}
```

```
public class PremierThread extends Thread {
	public PremierThread(final String n) {
		super(n); // nomme le Thread
	}

	@Override
	public void run() {
		for (int  i = 0 ; i<10 ;i++) {
			System.out.println("Bonjour");
			try {
				sleep(2000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

```
public class SecondThread extends Thread {
	public SecondThread(final String n) {
		super(n); // nomme le Thread
	}

	@Override
	public void run() {
		for (int  i = 0 ; i<15 ;i++) {
			System.out.println("TD De programmation");
			try {
				sleep(1000);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}
	}
}
```

b- On modifie maintenant le code de la classe Exercice1 comme ci-dessous. Commentez l'impact de l'ajout du join.

```
public class Exercice1 {

	public static void main(String[] args) {
		PremierThread pth = new PremierThread("Thread 1");
		SecondThread sth = new SecondThread("Thread 2");
		try {
			pth.join();
		} catch (InterruptedException e) {
			e.printStackTrace();
		}
		pth.start();
		sth.start();
	}
}
```

c- Où faudrait-il placer l'appel à la méthode join pour que le Thread pth s'exécute complètement avant le thread sth ?

2. Ecrivez un programme dont le thread principal lance et nomme trois nouveaux threads. Chaque thread ainsi créé doit effectuer 10 fois les actions suivantes :
   - ✓ Attendre un temps aléatoire compris entre 0 et 200 ms,
   - ✓ Puis afficher son nom

Le thread principal devra attendre la fin de l'exécution des trois threads qu'il a créés avant.

Exercice 3 (Synchronisation et coordination de threads) :

Soient les classes suivantes :

- *Compte* : Cette classe représente un compte. Elle possède un attribut représentant le solde et des méthodes permettant d'effectuer des retraits et dépôts.

- *RetraitThread* : Ce thread permet d'effectuer un retrait. Il possède en attributs le compte sur lequel effectuer le retrait et le montant du retrait.
- *DepotThread* : Ce thread permet d'effectuer un dépôt. Il possède en attributs le compte sur lequel effectuer le dépôt et le montant du dépôt.

```
public class Compte {
	private int solde;

	public Compte(int s) {
		solde = s;
	}
	public synchronized void retirer(int r) {
	solde -=r;
	System.out.println("Nouveau solde :"+getSolde());
	}
	public  synchronized void deposer(int d) {
		solde +=d;
		System.out.println("Nouveau solde :"+getSolde());
	}
	public  int getSolde() {
		return solde;
	}

	public static void main(String[] args) {
		Compte c = new Compte(1000);
		DepotThread d1 = new DepotThread(c,500);
		RetraitThread t1 = new RetraitThread(c, 1500);
		RetraitThread t2 = new RetraitThread(c, 2000);
		RetraitThread t3 = new RetraitThread(c, 1500);
		DepotThread d2 = new DepotThread(c,5000);

		t1.start();
		t2.start();
		t3.start();
		d1.start();
		d2.start();

	}
}
```

```
public class RetraitThread extends Thread{
	private int valeur;
	private Compte compte;

	public RetraitThread(Compte c, int v) {
		valeur = v;
		compte = c;
	}

	@Override
	public  void run() {
		compte.retirer(valeur);
	}

}
```

```java
public class DepotThread extends Thread{
	private int valeur;
	private Compte compte;

	public DepotThread(Compte c, int v) {
		valeur = v;
		compte = c;
	}

	@Override
	public  void run() {
		compte.deposer(valeur);
	}

}
```

1. Expliquez l'utilité des mots clés *syncronized* présents dans les déclarations des méthodes *deposer* et *retirer.*
2. Proposez un affichage possible de la console après exécution du programme.
3. Proposez une modification du code pour que l'on puisse effectuer un retrait uniquement si le montant du retrait est inférieur ou égal au solde disponible. Si ce n'est pas le cas, le retrait sera effectué lorsqu'un dépôt suffisant aura eu lieu.

Rappel : L'instruction *wait* met en attente le Thread qui lit l'instruction. L'instruction *notify* réactive un Thread en attente (si il y en a un). L'instruction *notifyAll* réactive l'ensemble des Threads en attente.