TD2 : Java « avancé »
V2.0.2

Document en ligne : www.mickael-martin-nevot.com

1 Généralités

Écrivez les applications ci-dessous en Java et en respectant la norme de programmation donnée en cours puis testez-les.

2 Création de threads

En Java, il existe deux façons de créer des *threads*. La première, en étendant la classe `Thread` :

```
public class MyThread extends Thread {
    public void run() {
        // Ici se trouve la description du comportement du thread.
    }
    public static void main(String[] args) {
        // On crée une instance de l'objet MyThread.
        MyThread t1 = new MyThread();
        // Puis on lance le thread : t1 exécute alors la méthode run().
        t1.start();
    }
}
```

La seconde façon, en implémentant l'interface `Runnable` :

```
public class MyJob implements Runnable {
    public void run() {
        // Ici se trouve la description du comportement du thread.
    }
    public static void main(String[] args) {
        // On crée une tâche pour un thread.
        MyJob job = new MyJob();
        // On crée une instance de Thread avec la tâche job.
        Thread t1 = new Thread(job);
        // Puis on lance le thread : t1 exécute alors la méthode run()
        // de la tâche job.
        t1.start();
    }
}
```

Déterminez pourquoi il existe deux manières de faire et quelles sont les différences entre elles.

3 Threads et JVM

Écrivez en Java un programme qui utilise deux *threads* en parallèle :
- le premier affichera les 26 lettres de l'alphabet (dans l'ordre lexicographique) ;
- le second affichera les nombres de 1 à 26 (dans l'ordre lexicographique).

Déterminez le résultat de ce programme lors de son exécution.

4 Méthodes de la classe Thread

Écrivez un programme dans lequel un *thread* principal lance trois nouveaux *threads* qui effectuent dix fois les actions suivantes (dans l'ordre) :
- attendre un temps aléatoire compris entre 0 ms et 200 ms ;
- afficher son nom.

Le *thread* principal doit attendre la fin de l'exécution des trois *threads* avant de terminer son exécution.

5 Threads et concurrence

Vous allez mettre en évidence ce qui peut se passer quand deux *threads* (représentés par deux personnes : Juliette et Roméo) partagent un même objet (représenté par un compte en banque).

5.1 Classe Account

La classe `Account` contient :
- la variable d'instance : `balance` (initialisée à 100, représentant le solde courant du compte) ;
- la méthode `withdraw(int amount)` permettant de faire un retrait.

5.2 Classe JulietteAndRomeoJob

La classe `JulietteAndRomeoJob` est une tâche (elle implémente l'interface `Runnable`) et représente aussi bien Juliette que Roméo. Elle contient :
- les variables d'instance : `name`, `account` (de type `Account`) ;
- la méthode `doWithdraw(int amount)` permettant à Roméo ou à Juliette d'effectuer un retrait sur leur compte en banque : la personne souhaitant le faire vérifie `account` (et affiche sa valeur), s'endort durant 500 ms, puis (à son réveil) effectue le retrait en signalant son nom ;
- la méthode `run()` permettant d'effectuer dix retraits de 10 € ;
- la méthode statique `main(String[] args)` permettant de créer deux tâches `romeoJob` et `julietteJob` qui seront utilisées par deux *threads* `romeo` et `juliette` (de type `JulietteAndRomeoJob`), de les nommer puis de les exécuter.

5.3 Test

Après avoir écrit le programme, testez-le et examinez son comportement.