# M1 Programmation avancée et répartie en Java : TP 4

Jean Fortin et Frédéric Gava
gava@u-pec.fr ou jean.fortin@gmail.com

## 1 Exercice 1

1. Ecrire une classe `Exo1`, contenant un champ privé $i$, de type entier, initialisé à 1.
2. Ajouter une méthode `testModif`, qui effectue les trois lignes de code suivantes :

   ```
   System.out.println(i);
   i = 2*i;
   i = i-1;
   ```

3. Créer une classe interne implémentant `Runnable`, dont la méthode run appelle la méthode `testModif` dans une boucle infinie.
4. Créer une méthode `lancerThread` (dans la classe principale) créant un Thread basé sur le Runnable ci-dessus, et le démarrant.
5. Dans une méthode `main`, créer un objet de la classe, et tester en lançant 3 threads en parallèle.
6. Dans la méthode `testModif`, ajouter entre les deux modifications de $i$ la création et suppression d'un fichier temporaire. Tester, et interpréter le résultat.
7. Modifier le programme pour éviter le problème constaté à la question précédente.

## 2 Exercice 2

On considère l'exemple d'un Thread Patience.

```
Thread patience = new Thread() {
  public void run() {
    while(!isInterrupted()) {
      System.out.print('.') ;
    }
  }
};
patience.start() ;
..... //on fait des tas de choses
patience.interrupt() ;
try {
  patience.join() ;
} catch(InterrruptedException xc) { ...}
```

Ce thread va afficher des points pendant le traitement lourd effectué dans le thread principal, afin que l'utilisateur voit que le programme n'est pas bloqué.

1. Compiler et tester un programme avec ce code.
2. Le Thread “patience” ne marche pas de façon complètement satisfaisante. Le modifier de façon à ce qu'il tente d'alterner son exécution avec le Thread principal qui l'a créé (pour un test essayer par exemple que l'un affiche “ping” et l'autre “pong” -essayer d'abord avec yield puis avec sleep)
3. Faire en sorte que chaque fois qu'un des threads commence à afficher un “Ping” (ou un “Pong”) il soit garanti qu'il puisse afficher 5 fois le même message consécutivement. (exemple : 5 Pings, 5 Pongs, 5 Pings, 5 Pings, etc...) -bien choisir l'objet support de la synchronisation
4. S'assurer que les messages vont alterner ( 5 pings, 5 pongs, etc.). Alternativement chaque processus se mettra en veille pour ensuite attendre que l'autre le “réveille”. (la solution est très simple, mais n'est pas immédiatement évidente !).

# 3 Exercice 3

En utilisant la classe Fifo (vue précédemment) concevoir une file spéciale qui contient des Threads (appellons la ThreadPool). Ces threads sont particuliers :

- leur méthode run() est une boucle infinie
- en début de boucle le thread se met de lui-même dans le ThreadPool et se bloque en attente.
- maintenant lorsqu'on soumet un Runnable au ThreadPool, Ce dernier retire un thread, le réactive et lui fait exécuter la méthode run du Runnable. Quand cette tâche est terminée le thread se replace en début de boucle. etc.
- (question subsidiaire : à quoi peut servir ce dispositif ?)

# 4 Exercice 4

- Créer une classe représentant une interface graphique pour un éditeur de message basique, avec une zone de texte au centre, et deux boutons “Envoyer” et “Quitter” en bas. Dans cette question, on s'intéresse simplement à la disposition des composants, pas à la gestion des évenements.
- Créer une classe de test avec une méthode main pour tester votre interface. Bien penser à utiliser la méthode invokeLater pour cela.
- Ajouter les gestionnaires d'évenements pour les boutons. Le bouton Envoyer se contente d'afficher un pop-up avec le message “Pas disponible pour l'instant”. (Utiliser la classe JOptionPane)
- A l'aide d'un Worker Swing, sauvegarder toute les 10 secondes le texte écrit dans un fichier “temp.txt”.

# 5 Exercice 5

Reprendre l'exercice 3 du TP3, mais utiliser cette fois un Worker swing pour effectuer le déroulement.