TD sur les threads

Objectifs : Les `Threads` – Synchronisation avec le mot-clé **synchronized** et les méthodes **`wait()`** et **`notify()`** de la classe **`Object`** - Utilisation de la classe `java.util.Vector`.

Exercice 1 : Dans ce premier exercice deux threads tentent de travailler simultanément sur une même ressource (ici : la sortie standard System.out).
Le code du premier thread sera définit dans une classe implémentant l'interface Runnable et affichera toutes les 100 ms (on utilisera pour cela la méthode sleep de Thread) un texte composé de 6 lignes :
Affichage du thread A :
2ème ligne du thread A
3ème ligne du thread A
4ème ligne du thread A
5ème ligne du thread A
6ème ligne du thread A
Le code du second thread sera définit dans une classe héritant de Thread et affichera toutes les 100 ms un texte composé de 6 lignes :
Affichage du thread B :
2ème ligne du thread B
3ème ligne du thread B
4ème ligne du thread B
5ème ligne du thread B
6ème ligne du thread B

1. Comment doit-on faire pour éviter que les affichages des threads soient mélangés ?
2. Donner le code des deux classes.

Exercice 2 : Nous allons présenter dans cet exercice une interaction classique entre deux entités : un `Producteur` et un `Consommateur`. Un `Producteur` crée des messages et les place dans une file, tandis qu'un `Consommateur` les lit et les affiche.
Pour être réalistes, nous donnerons à la file une taille maximale. Et pour rendre les choses plus intéressantes, nous rendrons le consommateur un peu feignant, en faisant en sorte qu'il s'exécute plus lentement que le producteur. Cela signifie que le producteur doit parfois s'arrêter et attendre que le consommateur se mette à travailler.

1. Identifier sur un diagramme UML les classes de cette application (propriétés et méthodes).
2. Dans une première version de cet exercice et pour des raisons de simplification, nous allons dériver les classes `Producteur` et `Consommateur` de la classe `Thread`.
   2.1. Quel problème soulève l'interaction Production/Consommation sur l'objet unique « file de message ». Comment le résoudre en java ?
   2.2. Que signifie « thread-safe » ? Vector est-il « thread-safe » ? Est-ce que ça résout le problème de synchronisation ?
   2.3. Après avoir commenté le code des classes Producteur et Consommateur (donné), écrire la classe principale Distributeur qui simule l'application.
3. Reprendre l'exercice en implémentant pour les classes `Producteur` et `Consommateur` l'interface `Runnable` (2ième version).
4. Développer la version « N Files – X Producteurs – Y Consommateurs » de cet exercice

```
import java.util.Vector ;

class Producteur extends Thread
{
    static final int MAXFILE = 5;
    static final int MAXMESSAGES = 50;
    private Vector messages;
    private int nbMessage;

    public Producteur() {
        messages = new Vector();
                  nbMessage = 0;
    }
```

```
    public void run(){
        try {
            while (nbMessage < MAXMESSAGES) {
                insererMessage();
                System.out.println("Message numéro "+nbMessage+" produit");
                nbMessage++;
                sleep(1000);
            }
        }
        catch (InterruptedException e) {}
    }
    private void insererMessage () {
        while (messages.size() == MAXFILE) {
            System.out.println("File pleine");
        }

        messages.addElement(new java.util.Date().toString());
    }
    public String recupererMessage() {
        while (messages.size() == 0) {
            System.out.println("File vide");
        }
        String message = (String)messages.firstElement();
        messages.removeElement(message);
        return message;
    }
}

class Consommateur extends Thread
{
    static final int MAXMESSAGES = 50;
    private int nbMessage;

    Producteur producteur;
    Consommateur(Producteur p) {
            producteur = p ;
                      nbMessage++;
    }

    public void run(){
        try {
            while (nbMessage < MAXMESSAGES) {
                String message = producteur.recupererMessage();
                System.out.println("Message numéro "+nbMessage+" recu : " +message);
                nbMessage++;
                sleep(2000);
            }
        }
         catch (InterruptedException e) {}
     }
}
```