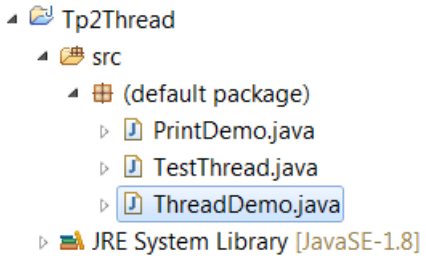
# TP2

## LES TREAD (1)

**Exercice 1 :**

L'objectif de ce TP est de lancer deux thread et synchroniser leur affichage. Pour cela, il vous est demandé de suivre les étapes suivantes :




1) créer un nouveau projet sur Eclispe :
   a. File/new/Java Project,
   b. puis nommer le projet créer `Tp2Thread`
2) créer trois nouvelles classes et nommer les `PrintDemo.java`, `TestThread.java` et `ThreadDemo.java`
   a. clique bouton droit sur le nom du projet `Tp2Thread`, puis cliquer sur `new/class`
3) Ecriture de la classe `PrintDemo.java`
   - Déclarer la méthode `printCount()`, qui affiche les entiers de 5 à 1.

```
class PrintDemo {
   public void printCount(){
    try {
         for(int i = 5; i > 0; i--) {
            System.out.println("Counter   ---   "  + i );
         }
     } catch (Exception e) {
         System.out.println("Thread  interrupted.");
     }
   }
}
```

4) Ecriture de la classe `ThreadDemo` qui hérite de la classe `Thread`
   - Variables d'instances

   déclarer trois variables d'instance privées : `t`, `threadName` et `PD` respectivement de type `Thread`, `String`, `PrintDemo`.
   - Constructeur `ThreadDemo(String name, PrintDemo pd)` permet d'instancier les variable `name` et `PD`
   - Méthode d'instance `void run()` qui permet d'invoquer `PD.printCount()`

5) Ecriture de la classe `TestThread` qui contient la méthode `static void main(String args[])`. Cette méthode contient les instructions suivantes :
   - Déclare et instancie une variable `PD` de type `PrintDemo`.
   - Déclare et instancie une variable `T1` de type `ThreadDemo` avec les paramètres `"Thread - 1 "` et `PD`.
   - Déclare et instancie une variable `T2` de type `ThreadDemo` avec les paramètres `"Thread - 2 "` et `PD`.
   - Lance les threads `T1` et `T2`.

6) modifier la méthode `run` de la classe `ThreadDemo` pour afficher de manière séquentielle les résultats du thread `T1` puis les résultats du thread `T2`.

**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 2 :

Dans cet 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. 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