SN/UCAD/ESP/DGI/DIC2

TD TP Sur les threads :

Exercice 1 : (Synchronisation de threads : blocs synchronisés).
Réaliser une classe Banque permettant de créer et de manipuler des comptes en banque.
Un compte sera identifié par le prénom du titulaire, le nom, le numéro de compte, le solde initial et le découvert.
Prévoir :
- un constructeur qui crée convenablement des comptes en banque
- une méthode pour créditer sur le solde. Elle prend en paramètre le montant à créditer
- une méthode pour débiter sur le solde. Elle prend en argument le montant à débiter.
- une méthode pour récupérer le prénom du titulaire du compte
- une méthode pour récupérer le nom du titulaire du compte
- une méthode pour récupérer le solde du titulaire du compte

- une méthode pour récupérer le découvert du titulaire du compte
- une méthode pour modifier le montant du découvert du compte
- une méthode getHistorique pour imprimer le solde du compte.

Maintenant, on veut créer deux processus qui gèrent les transactions que le client veut faire sur son compte. Pour simplifier, on suppose que le client peut réaliser deux opérations :
Créditer ou débiter.
Mais attention ces deux opérations ne devront jamais et en aucun cas se réaliser simultanément, sinon on risque d'avoir un historique sur le compte qui sera faux.
Créer alors deux classes de thread :
- l'une, ProcessBankCredit, pour gérer l'opération créditer
- l'autre, **ProcessBankDebit**, pour gérer l'opération débiter.

Expliquer par des commentaires javadoc précis (sur ces deux classes) les dispositions que vous prenez pour que les données (du solde) ne soient pas corrompues si un caissier lance simultanément ces deux opérations sur un même compte.
Reprendre *l'exercice* en implantant cette fois-ci l'interface Runnable.

Exercice 2 : (course de 1000 m)
Réaliser une classe **Coureur** en étendant la classe **Thread** qui simule une compétition dans une course de 1000 m.
Un coureur n'est identifié que par son nom. Chaque coureur prenant part à cette compétition sera identifié à un processus.
Créer :
- un constructeur qui instancie correctement un coureur
- on redéfinit la méthode d'exécution du thread pour décrire le comportement d'un coureur quelconque : on fixe une pause aléatoire de 1 seconde maximum qui simule le temps mis par chaque coureur pour franchir 100 m.

A chaque fois qu'un coureur fait i*100 m, il faut l'afficher comme le montre la sortie de l'exemple.
Voici une classe de test et un exemple de sortie :

```
public class TestCoureur {
public static void main (String [ ] args) {
// Il s'agit d'une classe de coureurs
System.out.println ("Passage aux : ");
Coureur j = new Coureur ("Margaret");
Coureur p = new Coureur ("Irene");
// On lance les deux coureurs.
j.start ( );
p.start ( );
try { j.join ( ); p.join( ); // attendre la mort de j et p
// avant de commencer k
}
catch (InterruptedException e) {};
Coureur k = new Coureur ("Janet");
k.start ( ) ;
}
}
```

NB : ces deux classes doivent etendre la classe **Thread.**

```
Passage aux :
100m par Margaret
100m par Irene
200m par Margaret
200m par Irene
300m par Margaret
400m par Margaret
300m par Irene
500m par Margaret
600m par Margaret
700m par Margaret
400m par Irene
800m par Margaret
900m par Margaret
1000m par Margaret
500m par Irene
Margaret est arrivé
600m par Irene
700m par Irene
800m par Irene
900m par Irene
1000m par Irene
Irene est arrivé
100m par Janet
200m par Janet
300m par Janet
400m par Janet
500m par Janet
600m par Janet
700m par Janet
```

Exercice 3 :

Ecrire une classe **Trieur** contenant les méthodes suivantes :

- **public int [ ] triTableau (int t [ ])**, qui prend en argument un tableau d'entiers etrenvoie un tableau dont les éléments sont triés par ordre croissant.
- **public void echanger (int t [ ], int i, int j)**, cette méthode prend en paramètre un tableau d'entiers et deux indices i et j qui sont les deux indices du tableau dont on veut échanger le contenu. Cette méthode sera utilisée dans la méthode triTableau pour échanger les valeurs de deux cases du tableau à trier.
- **public void affiche ( )**, qui affiche les éléments du tableau trié.

Créer maintenant deux classes de threads : **ThreadTrieTableau** qui permet de lancer l'opération de trie d'un tableau d'entiers et **ThreadAfficheTableau** qui lance l'affichage des éléments du tableau trié.

Gérer ces deux processus de telle sorte que des ***problèmes d'accès concurrents*** (que vous identifierez) sur une donnée partagée, ne puissent jamais se rencontrer.

Créer une classe **TestTrieur** où vous créerez un tableau en demandant de saisir ses éléments au clavier et vous lancerez les deux threads sur ce dernier.

NB : ces deux classes doivent implanter l'interface **Runnable.**