ESISAR

NE449 – Programmation répartie

TDM numéro 8

Table des matières

1 Exercice 0 – Le problème de la concurrence.

Executez le code ci dessous :

```
public class Calculatrice extends Thread
{
	private Somme somme;

	public Calculatrice(Somme somme)
	{
		this.somme = somme;
	}

	@Override
	public void run()
	{
		int res = 0;
		for (int i = 0; i < 100; i++)
		{
			res= somme.somme(res, i);
		}
		System.out.println("La somme est 1 et 99 est :"+res);
	}


	public static void main(String[] args) throws InterruptedException
	{
		Somme somme = new Somme();
		Calculatrice c1 = new Calculatrice(somme);
```

```
            Calculatrice c2 = new Calculatrice(somme);

            c1.start();
            c2.start();
      }


      static  class Somme
      {

            int c;

            public int somme(int a, int b)
            {
                  c=a+b;
                  System.out.println("c="+c);
                  return c;
            }

      }

}
```

Les résultats affichés sont ils justes ? Comment corriger ?

2 Exercice 1 : Le problème des philosophes - simple

Le problème des philosophes est un problème classique inventé par le mathématicien hollandais E Dijskstra, pour tester les algorithmes de synchronisation.

Un ensemble de N philosophes (N vaut par exemple 5) sont réunis autour d'une table dans un restaurant chinois. Chaque philosophe boucle sur deux tâches :

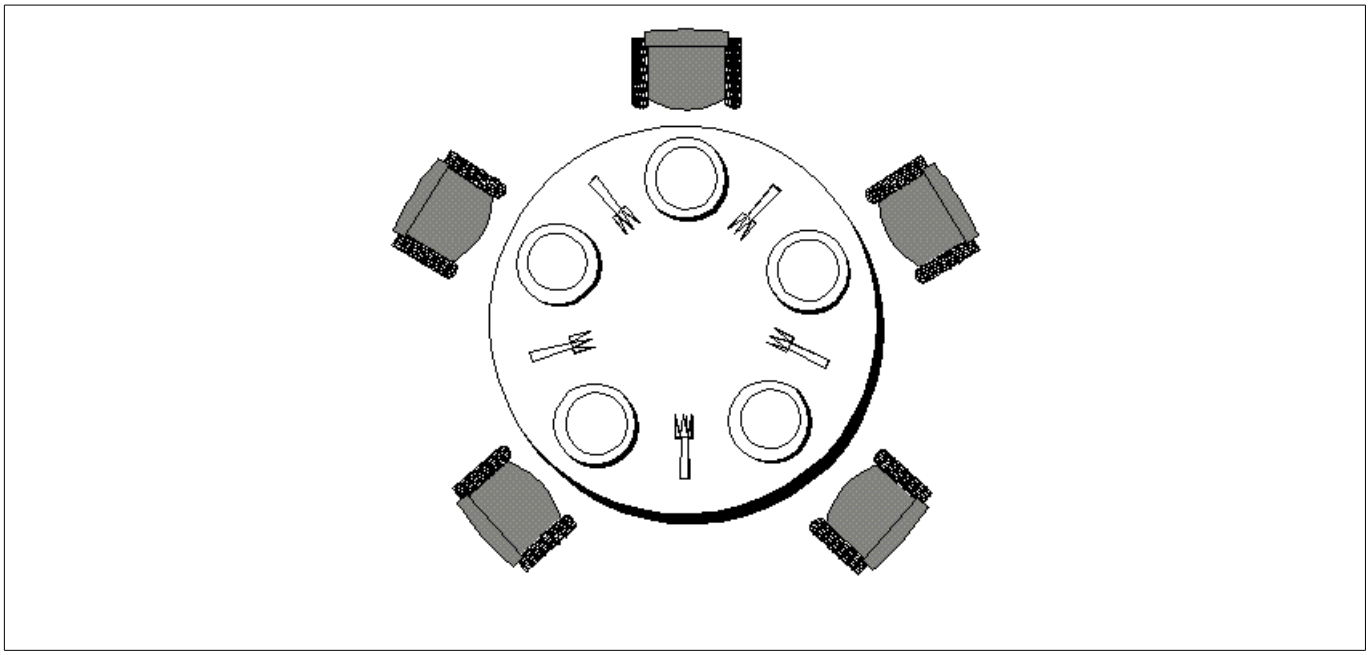
- discuter un temps aléatoire compris entre 0 et 10 secondes
- manger un temps aléatoire compris entre 0 et 10 secondes

Faites un premier programme avec 5 thread, chaque thread représentant un philosophe. Vous afficherez une trace quand un philosophe commence de manger, et quand il commence à discuter, avec aussi le numéro du philosophe.

3 Exercice 2: Le problème des philosophes avec synchronisation des fourchettes

Maintenant, on admet que les assiettes sont séparées de ses voisines par une baquette

seulement.



Pour pouvoir manger, un philosophe doit acquérir au préalable les 2 baguettes situées de part et d'autre de son assiette (il est donc en compétition avec ses voisins pour l'obtention des baquettes).

Il doit les libérer lorsqu'il se met à discuter pour permettre à ses voisins de manger à leur tour.

Le problème consiste à synchroniser correctement le comportement des philosophes. La solution envisagée consiste à disposer d'un arbitre, qui autorisera ou non les philosophes à prendre les baquettes.

Créez donc une classe Arbitre avec deux méthodes :

- **boolean autorisation(int numPhilo)** : un philosophe appellera cette fonction pour savoir si il a le droit de manger. Cette méthode retourne true si les deux baquettes sont libres, false sinon
- **void liberation(int numPhilo)** : un philosophe appellera cette fonction quand il aura fini de manger, pour indiquer à l'arbitre que les baquettes sont maintenant disponibles.

Votre classe Philosophe aura donc maintenant le comportement suivant :

- discuter un temps aléatoire compris entre 0 et 10 secondes
- demander les baquettes à l'arbitre
    - si l'arbitre répond ok : manger un temps aléatoire compris entre 0 et 10 secondes puis retourner à l'état « je discute » puis remanger ...

- si l'arbitre répond non : attendre 1 seconde, puis demander de nouveau les baguettes à l'arbitre. Continuez ainsi jusqu'à obtenir les baguettes. Après avoir mangé, retourner à à l'état « je discute » puis remanger ...

Faites fonctionner le tout. Afficher des statistiques : quel est le philosophe qui mange le plus ? Qui mange le moins ? Le temps d'attente moyen pour avoir les fourchettes ?