

TP 1 : PRAM & Thread Java

1 Barrières de Synchronisation et PRAMs

A.1 Ecrire une classe `Barrière` qui peut être utilisée comme une barrière de synchronisation réutilisable.

- Constructeur `Barriere(int nb)` : définit une barrière pour `nb` threads,
- Méthode `attendre(void)` : bloque jusqu'à ce `nb` threads aient appelé cette méthode.

Remarque : Il est préférable d'utiliser `Wait` et `notifyAll`

A.2 Ecrire un programme réalisant une course *équitable* entre threads, ie bloquant les threads jusqu'à ce qu'ils soient tous "prêts" à courir.

A.3 Implémenter l'algorithme de tri sur PRAM linéaire vu au TD précédent.

2 Gestion de compte bancaire

B.1 Rappeler les deux méthodes de bases utilisées dans JAVA pour créer un thread. Quelle est la différence entre les méthodes `run` et `start` ?

On considère maintenant le code de la figure 1.

B.2 Ce code permet-il de simuler correctement le fonctionnement d'un compte bancaire ?

Si non, proposez une solution.

```
public class Compte {
    private int valeur;
    Compte(int val) { valeur = val; }
    public int solde() { return valeur;}
    public void depot(int somme) {
        if (somme > 0) { valeur+=somme; }
    }
    public boolean retirer(int somme) throws InterruptedException {
        if (somme > 0)
            if (somme <= valeur) {
                Thread.currentThread().sleep(50);
                valeur -= somme;
                Thread.currentThread().sleep(50);
                return true;
            }
        return false;
    }
}
```

```

public class Banque implements Runnable {
    Compte nom;
    Banque(Compte n) { nom = n; }
    public void liquide (int montant) throws InterruptedException {
        if (nom.retirer(montant)) {
            Thread.currentThread().sleep(50);
            donne(montant);
            Thread.currentThread().sleep(50);
        }
        imprimeRecu();
        Thread.currentThread().sleep(50);
    }
    public void donne(int montant) {
        System.out.println(Thread.currentThread().getName()+
            ": Voici vos " + montant + " euros.");
    }
    public void imprimeRecu() {
        if (nom.solde() > 0)
            System.out.println(Thread.currentThread().getName()+
                ": Il vous reste " + nom.solde()+ " euros
                .");
        else
            System.out.println(Thread.currentThread().getName()+
                ": Vous etes fauches!");
    }
    public void run() {
        try {
            for (int i=1;i<10;i++) {
                liquide(100*i);
                Thread.currentThread().sleep(100+10*i);
            }
        } catch (InterruptedException e) {
            return;
        }
    }
    public static void main(String[] args) {
        Compte Commun = new Compte(1000);
        Runnable Mari = new Banque(Commun);
        Runnable Femme = new Banque(Commun);
        Thread tMari = new Thread(Mari) ; tMari.setName("Conseiller Mari
            ");
        Thread tFemme = new Thread(Femme); tFemme.setName("Conseiller
            Femme");
        tMari.start(); tFemme.start();
    }
}

```

FIG. 1 – Gestion de compte bancaire