

## ***TD 4 : Carafes, convives et garçons de café***

### **Objectifs :**

Accès concurrents à une ressource partagée  
Utilisation du mécanisme wait/notifyAll (mise en veille /réveil d'un thread)  
Mise en œuvre d'une tâche planifiée (timer)

### **Enoncé :**

Dans un café, les convives installés à une table se servent à la carafe d'eau sans se soucier les uns des autres : lorsque son verre est vide, un convive saisit la carafe, remplit le verre, repose la carafe et boit son verre. Un garçon de café vient remplir la carafe à intervalles réguliers, puis retourne se reposer.

Le code donné sur le serveur commun modélise une carafe non thread-safe (classe Carafe) et son utilisation (classe AppliCarafe) dans le contexte où un garçon de café (classe GarconDeCafe) et 3 convives (classe Convive) partagent cette carafe.

### **Exercice 1 :**

Vous devez rendre cette application fiable, thread-safe et performante en ajoutant tests, sections critiques et wait/notifyAll :

#### **1<sup>ère</sup> version : la ressource est sécurisée**

sans modifier les classes Convive et GarconDeCafe, en travaillant uniquement sur le code de la classe Carafe (la ressource partagée).

#### **2<sup>ème</sup> version : les accès à la ressource sont sécurisés**

en reprenant la classe Carafe initiale et en travaillant cette fois sur le code des activités Convive et GarconDeCafe.

#### **3<sup>ème</sup> version : un proxy de la ressource gère les accès sécurisés de façon transparente pour les runnable et propage par délégation les actions sur la ressource**

Carafe est désormais une interface et la classe Carafe de l'énoncé est renommée SimpleCarafe et implémente l'interface Carafe ; vous écrirez une classe ConcurrentCarafe implémentant la même interface et gérant les accès concurrents à une Carafe en lui déléguant les opérations dans un contexte thread-safe. Cette technique par « proxy » (représentant de la carafe pour l'application) permet de ne pas revoir le code des Runnable pour qui la carafe reste de type Carafe.

Vous utiliserez vos connaissances sur le découplage pour séparer le cœur du domaine (des Runnable agissant sur une carafe) du contexte spécifique de l'application et de la délégation.

### **Exercice 2 :**

On se propose maintenant de prendre en compte la fin de service d'un garçon de café, soit au bout d'un certain temps d'activité (délai), soit à heure fixe. Deux garçons de café seront créés et l'un des deux servira de test pour la fin de service, l'autre continuant indéfiniment.

Vous utiliserez la classe java.util.Timer (planification) qui permet de prévoir le déclenchement d'une java.util.TimerTask (tâche planifiée) et vous créerez donc une tâche FinService interrompant le thread du garçon de café.

**1<sup>ère</sup> version : fin de service sur délai** on supposera que le garçon de café Brette travaille 1000ms

**2<sup>ème</sup> version : fin de service à heure fixe** on supposera que le garçon de café Brette travaille jusqu'à hh:mm:ss (pour le test, vous pouvez prendre l'heure système + une seconde)