Carafes, convives et garçons de café

Objectifs :

Accès thread-safe à une ressource partagée non prévue à cet effet
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 :

1ère 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).

2ème 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.

3ème 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 délégant 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 agissent sur une carafe) du contexte spécifique de l'application et de la délégation.

Exercice 2 :

Le garçon de café désormais ne se déplace que si l'un des convives lui signale que la carafe est vide.
A partir de la seconde version précédente, vous ajouterez les wait() et notifyAll() nécessaires : après remplissage de la carafe, le garçon se met en veille et lorsque la carafe est vide (ou bien est-ce à chaque verre ?), on doit le notifier de l'événement. Vous vérifierez en créant plusieurs garçons de café qu'ils viennent plus ou moins à tour de rôle et qu'un seul vient à chaque remplissage.
Votre code est-il facile à faire évoluer si on décide que les garçons de café anticipent le remplissage est viennent lorsqu'il ne reste qu'1/3 de la carafe (bien sur en ajoutant à Carafe une méthode donnant la proportion de remplissage) ?

Exercice 3 :

On se propose maintenant de prendre en compte la fin de service d'un garçon de café, soit au bout d'un certain temlps 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.
un second garçon de café , et modifierez le code de la classe GarconDeCafe afin de prendre en compte la fin de service ;

1ère version : fin de service sur délai
on supposera que le garçon de café Brette travaille 1000ms.
2ème version : fin de service à heure fixe
en reprenant la classe Carafe initiale et en travaillant cette fois sur le code des activités Convive et GarconDeCafe.

on supposera que le garçon de café Brette travaille 1000ms. Le second garçon de café pourra travailler indéfiniment.

Exercice 4 :

On a maintenant une situation plus proche de la réalité : la salle du café comporte plusieurs tables ; sur chaque table il y a une carafe ; un convive s'assoit à une table donnée ; les garçons de café par contre n'ont pas de carafe dédiée mais sont susceptibles de remplir n'importe quelle carafe lorsqu'elle est vide.

Une liste de carafe devra être créée et conservée par attribut static dans la classe GarconDeCafé puisque ecte liste est commune à tous.
La référence de la carafe d'une table restera par contre attribut d'instance dans Convive (inutile de créer une classe pour la table, le numéro dans la liste pouvant servir d'identifiant de la carafe).