TP 3: le jus de mangue, quand c'est plus bon, c'est plus bon

notions : Première approche de l'héritage

1 Une classe `BouteilleJus`

Vos classes `Verre`, `Bouteille` et `Bar` réalisées durant le tp Cocktail des deux premières séances sont fort belles, et votre client, le patron du bar "Bu", en est fort satisfait. Mais voila : le bar "Bu" n'a pas de frigo. Ses bouteilles de jus de fruit risquent donc de rapidement devenir peu gouteuses une fois ouvertes...

Pour éviter d'empoisonner les clients, on souhaite implanter un nouveau type de bouteilles qui rende compte de la notion de **bouteille de jus de fruit**. En plus d'une bouteille standard, une bouteille de jus de fruit stockera la date et l'heure de sa première ouverture, connait la durée de validité après ouverture (supposons que c'est écrit sur l'étiquette...) et est capable de signaler qu'il y un risque lorsqu'un verre est servi à courageux buveur alors que le contenu est périmé.

Comme une bouteille de jus de fruit –**est avant toute chose**– une bouteille (à laquelle s'ajoutent d'autres caractéristiques), on aura légitimement recours à l'héritage.

Copiez votre répertoire du TP Cocktail (séances 1 et 2), ou si vous n'avez guère confiance récupérez la correction de ce TP sur le site du module.

En utilisant les principes de l'héritage, implantez une classe `BouteilleJus`.

En plus des attributs de toute `Bouteille`, une `BouteilleJus` dispose des **attributs** suivants :
- `int secondsAvantPeremption ;` Le nombre de seconde après lequel, une fois la bouteille ouverte, la bouteille devient suspect...
- `LocalDateTime dateTimeOuverture ;` Cet attribut vaudra `null` tant que la bouteille n'a pas encore été ouverte, puis stockera la date et l'heure de la 1ère ouverture.

Le **comportement** d'un objet `BouteilleJus` est celui d'un objet `Bouteille`, si ce n'est que :
- A la création, les `BouteilleJus` sont toujours fermées. De plus, il est possible de préciser le nombre de secondes avant péremption. Si il n'est pas précisé, une valeur par défaut de *1 seconde* sera choisie [1].
  La classe déclarera donc 2 constructeurs :
  - `BouteilleJus(String nom, int quantite, int secondsAvantPeremption)`
  - `BouteilleJus(String nom, int quantite)`*`//par défaut : 1 seconde avant péremption.`*
- à la première ouverture de la bouteille (méthode `ouvrir()`), on stocke la date et et l'heure dans l'attribut `dateTimeOuverture`.
  Remarque : la méthode de classe `now()` de la classe `LocalDateTime` devrait être bien utile... Voir exemple en annexe.
- lorsqu'on verse le contenu de la bouteille dans un verre (méthode `verser(Verre v)`), il s'affiche un message `"à vos risques et péril"` dans le Terminal si la bouteille est périmée.

1. Une seconde, c'est absurde, certes... mais ça facilitera les tests !

Enfin, la classe `BouteilleJus` définit la méthode supplémentaire suivante :

— `public boolean estPerimee()` : retourne `true` si le contenu est périmé, c'est à dire si la première ouverture de la bouteille date de plus de `secondsAvantPeremption` secondes.
Remarque : pour calculer une durée écoulée entre deux instances de `LocalDateTime`, on utilisera avec profit la méthode de classe `between()` de la classe Java `Duration`. Voir exemple en annexe.

2 Programme de test

Ecrire un programme de test qui réalise les actions suivantes :

— création d'un verre, d'une bouteille de jus de mangue (périmée au bout d'une seconde après ouverture), d'une bouteille de pastis (non périmable)
— ouverture des bouteilles
— affichage des bouteilles
— attendre 4 secondes (utiliser `Thread.sleep(4000);` Voir exemple en annexe).
— affichage des bouteilles
— faire un bon (????) cocktail dans le verre
— boire le verre (bon courage...)

3 Quelques questions pour mieux comprendre

— A votre avis, serait-il logique qu'on puisse stocker des instances de `BouteilleJus` dans un `Bar`, alors que celui-ci a en attribut un tableau contenant des `Bouteille` simples ?
— pourquoi peut-on appeler la méthode `public static LocalDateTime now()` de la classe `LocalDateTime` en écrivant : `LocalDateTime.now()`, alors que `LocalDateTime` est une classe et non pas une référence sur un objet ?
— A votre avis, pourquoi peut-on passer deux références de type `LocalDateTime` à la méthode
`public static Duration between(Temporal startInclusive, Temporal endExclusive)`
de la classe `Duration`, alors que cette méthode déclare prendre des références de type `Temporal` en paramètre ?

4 Annexe : dates, heures et durées en Java

Pour gérer les questions de date/heure et de durée, on pourra utiliser les classes Java `LocalDateTime` et `Duration`.

Consulter leur Javadoc ici :

— `https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html` .
Voir notamment la méthode de classe `public static LocalDateTime now()`.
— `https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html` .
Voir notamment la méthode de classe
`public static Duration between(Temporal startInclusive, Temporal endExclusive)`
et la méthode `public long toMillis()`

Voici un exemple qui affiche une durée en secondes entre deux `LocalDateTime` :

```java
import java.time.*;

public class TestDuration {
    public static void main(String[] args) {
        LocalDateTime ldt1 = LocalDateTime.now();
        // pour simuler le temps qui passe,
        // on met le programme en pause durant 4 secondes

         try {
            System.out.println("Dodo 4 secondes");
            Thread.sleep(4000); // le programme s'arrête 4 secondes
        } catch(InterruptedException e) {
        }

        LocalDateTime ldt2 = LocalDateTime.now():
        System.out.println("durée écoulée : "
            +  Duration.between(ldt1, ldt2).toMillis() /1000
            + " secondes ");
        // il s'affiche alors "4 secondes"
    }
}
```