TD2

Exercice 1:
On souhaite définir différents types d'articles dans un magasin. Tous les articles partagent des caractéristiques communes, une référence, un libellé, un fournisseur et un prix d'achat. Des promotions pourront être appliquées à l'ensemble des articles. On enregistrera donc aussi pour chaque article un prix de l'article et un prix de vente pouvant être inférieur au prix de l'article.

1. Créer une classe `Article` caractérisée par une référence de type `int`, un libellé, un fournisseur de type `String`, un `prixAchat`, un `prixArticle` et un `prixVente` de type `double`. Ces attributs auront une visibilité **privée**.

   Cette classe fournira un constructeur ayant pour paramètres la référence, le libellé, le fournisseur, le prix d'achat et le prix de l'article. Le prix de vente sera initialisé au prix de l'article. Elle fournira également les méthodes suivantes :

   a. `public void promotion (int pourcentage)` qui affecte au prix de vente le prix de l'article auquel on applique une réduction du pourcentage indiqué en paramètre.
   b. `public void remonterPrix ()` qui affecte au prix de vente la valeur du prix de l'article.
   c. `public String toString()` qui retourne l'état d'un article sous forme d'une chaîne de caractères décrivant toutes les caractéristiques de celui-ci.

On distingue trois types d'articles, les articles électromenagers, les articles d'habillement et les articles primeurs.

2. Créer une classe `ArticleElectromenager` héritant de la classe `Article` ayant pour caractéristique supplémentaire le poids : `private int poids`.

   Cette classe fournira un constructeur ayant pour paramètres la référence, le libellé, le fournisseur, le prix d'achat, le prix de l'article et le poids.
   Elle redéfinira la méthode `public String toString()` qui retourne l'état d'un article tel que défini dans la classe `Article` suivi, sur une autre ligne, du poids de cet article.

3. Créer une classe `ArticleHabillement` héritant de la classe `Article` ayant pour caractéristique supplémentaire la taille : `private int taille`.

   Cette classe fournira un constructeur ayant pour paramètres la référence, le libellé, le fournisseur, le prix d'achat, le prix de l'article et la taille.
   Elle redéfinira la méthode `public String toString()` qui retourne l'état d'un article tel que défini dans la classe `Article` suivi, sur une autre ligne, de la taille de cet article.

4. Créer une classe `ArticlePrimeur` héritant de la classe `Article` ayant pour caractéristique supplémentaire la date de péremption : `private LocalDate datePeremption`.
   (voir la documentation de `java.time.LocalDate` en annexe1).

   Cette classe fournira un constructeur ayant pour paramètres la référence, le libellé, le fournisseur, le prix d'achat, le prix de l'article et la date de péremption.
   Elle redéfinira la méthode `public String toString()` qui retourne l'état d'un article tel que défini dans la classe `Article` suivi, sur une autre ligne, de la date de péremption de cet article.

   Elle fournira la méthode `public boolean estPerime()` qui retourne `true` si l'article est périmé et `false` sinon.

5. On souhaite gérer les articles d'un magasin. Écrire une classe `Magasin` caractérisée par

- un attribut `lesArticles` de visibilité privée et de type un tableau d'objets de type `Article`.
- un attribut `capacite` fixant la capacité du tableau à 1000 ;
- un attribut `nb` correspondant au nombre d'articles dans le magasin qui sera initialisé à 0.

La classe `Magasin` fournira un constructeur sans paramètre ainsi que les méthodes suivantes:

- `public boolean estPresent (Article a)` qui retourne vrai si l'article est présent dans le magasin et faux sinon. Que faut-il redéfinir dans la classe `Article` ?
- `public boolean ajouterArticle(Article a)` qui ajoute l'article passé en paramètre à la fin du tableau et retourne `true` à condition que l'article ne soit pas déjà présent dans le magasin et que la capacité le permette. Sinon, la méthode retourne `false`. On pensera à incrémenter le nombre d'articles du magasin.
- `public boolean promotion(Article a, int pourcentage)` qui applique à l'article passé en paramètre la promotion correspondant au pourcentage passé en paramètre et retourne vrai. La méthode retourne faux si l'article n'est pas présent dans le magasin.
- `public String toString()` qui retourne l'état de chacun des articles du magasin. Les informations de chaque article apparaissant sur une ligne différente.

6. Créer une classe `Main` contenant le `main (String[] args)`. Créer un article de chaque type en définissant des variables du type général `Article`.
   Exemple : `Article a = new ArticlePrimeur(1, "Salade", "f", 0.4, 0.9, LocalDate.of (2019,10,1))`.
7. Créer un magasin et y ajouter les articles.
   Afficher l'état des articles contenus dans le magasin.
   Appliquer une promotion à certains articles puis afficher de nouveau l'état des articles contenus dans le magasin.
   Tester si l'article `a` est périmé.Est-ce possible ? Que faut-il faire ?

Exercice 2 . On considère l'interface suivante :

```
@FunctionalInterface
public interface Transformation {
     Object transforme (Object o);
}
```

1. Créer une classe `EnsTransformable` permettant de gérer un ensemble d'éléments transformables. Cette classe sera caractérisée par un attribut privé nommé `ensemble` de type `Object[]` et fournira un constructeur ayant un tableau de type `Object` en paramètre.

   Redéfinir la méthode `toString ()` et ajouter la méthode :

   - `public void transformer (Transformation f)` appliquant la transformation `f` à chaque élément de l'ensemble.

2. Créer une classe `TransformationDouble` contenant la méthode `main`. Définir un objet de type `EnsTransformable` à partir d'un tableau de type `double`. Appliquer à l'ensemble les transformations suivantes :

- Prendre la racine carrée.
- Élever au carré.
- Élever à la puissance 3.

Afficher le contenu de l'ensemble après chaque transformation.

Rappel : la classe `Math` fournit les méthodes suivantes :

- `public static double sqrt ( double a)` qui retourne la racine carré du double passé en paramètre
- `public static double pow ( double a, double b)` qui retourne la valeur du premier paramètre élevé à la puissance du second.

3. On souhaite utiliser la classe `EnsTransformable` pour transformer un tableau de chaînes de caractères en majuscule ou en minuscule.

   Construire une classe `TransformationMinMaj` contenant la méthode `main`. Créer un ensemble transformable à partir du tableau de `String` suivant : `String[] lesMots = {"Bonjour", "HIER","Aujourd'hui"}.`

   Interagir avec l'utilisateur de manière à lui proposer de choisir le type de transformation qu'il souhaite appliquer à l'ensemble : **`a`** pour majuscule et **`i`** pour minuscule. Appliquer la transformation demandée à l'ensemble et afficher le résultat.

   La classe `String` fournit la méthode `public String toUpperCase()` qui convertit tous les caractères de la chaîne en majuscule et la méthode `public String toLowerCase()` qui convertit tous les caractères de la chaîne en minuscule.