Seul document autorisé : une feuille A4 recto-verso manuscrite (non photocopiée). Aucune machine.
Le barème est indicatif. Une question peut toujours être traitée en utilisant les précédentes (traitées ou non).
Les morceaux de code Java devront être clairement présentés, indentés et commentés.
L'utilisation de collections de l'API Java (en particulier `LinkedList` et `ArrayList`) est interdite, sauf mention explicite contraire.

- Tous les attributs d'instance de toutes les classes introduites doivent être privés.
- Il est possible d'écrire des méthodes non spécifiquement demandées si c'est plus pratique, en spécifiant pour chaque méthode dans quelle classe elle se trouve.

Le gérant du "gland de chêne" a trouvé très efficace de passer à une gestion informatisée des commandes. Il voudrait améliorer maintenant la productivité en cuisine et éviter les erreurs de composition de sandwichs. Le but de ce sujet est de lui proposer une implémentation de "recettes" de sandwichs.

Ingrédient

Un ingrédient est représenté par son nom, son stock courant et son stock d'alerte (ces deux derniers étant représentés par des entiers, sans se soucier des unités). La classe **`Ingredient`** intègre par ailleurs un mécanisme qui permet de connaître à chaque instant la liste des ingrédients dont le stock est passé en dessous du seuil d'alerte. Cette liste peut être modélisée en utilisant la classe **`java.util.ArrayList`**.

La classe **`Ingredient`** contient donc au minimum un attribut de type chaîne de caractères et deux attributs entiers représentant le stock courant et le stock d'alerte. Son interface est la suivante (le caractère statique ou non des méthodes n'est pas spécifié, ce sera à vous de le donner, en justifiant votre choix) :

```
/** affiche la liste des ingrédients à acheter (ceux dont le stock courant est strictement
inférieur au stock d'alerte) avec la quantité à acheter: pour chaque ingrédient, la quantité
à acheter doit permettre de mettre le stock courant au stock d'alerte plus la quantité
passée en paramètre */
void listeDeCourse(int surplus);

/** simule l'achat d'un ingrédient en ajoutant la valeur du paramètre au stock courant */
void achat(int quantite);

/** simule la consommation d'une unité de l'élément, quand c'est possible */
void consomme();
```

Cette classe possède un constructeur qui prend en argument le nom d'un ingrédient et son stock d'alerte.

Exercice 1. (3 points) Écrire la classe **`Ingredient`** (attributs, constructeurs et méthodes). N'oubliez pas d'expliquer ce qui doit être statique. Notez bien que le surplus en argument de **`listeDeCourse`** est commun à tous les ingrédients : ce n'est pas réaliste, mais c'est pour simplifier l'implémentation.

Recettes

Une recette est une suite ordonnée d'ingrédients. On peut voir apparaître plusieurs fois le même ingrédient dans une recette (ex : dans un double cheeseburger, il y a deux steaks hachés séparés par une tranche de fromage). Pour simplifier, on supposera qu'une apparition d'un ingrédient dans une recette correspond à une unité de cet ingrédient : quand on réalise la recette, chaque apparition décrémente donc le stock de cet ingrédient d'une unité.

L'ensemble des recettes est représenté dans un arbre binaire. L'idée étant qu'en suivant un chemin depuis la racine dans l'arbre jusqu'à un nœud reconnu comme étant un sandwich (mais qui n'est pas nécessairement une feuille), on a la liste des ingrédients *dans l'ordre* pour ce sandwich. Chaque nœud représente une option sur un ingrédient : s'il est intégré au sandwich la recette se poursuit en explorant le fils gauche ; et s'il n'est pas intégré, une autre option d'ingrédient est envisagée : celle du fils droit.

Pour celà, on crée les classes suivantes :

- **`Recette`** : référence la racine de l'arbre ;
- **`Creation`** : référence un nœud de l'arbre et contient :

  - une référence vers un ingrédient
  - deux références vers des nœuds,
  - une référence vers le nœud père dans l'arbre (nulle dans le cas de la racine de l'arbre),
  - une référence vers un sandwich, non nulle si et seulement si le nœud correspond à un sandwich ;
- **`Sandwich`** : référence un sandwich et contient :
  - un attribut de type chaîne de caractères contenant le nom du sandwich,
  - une référence vers le nœud de l'arbre qui correspond à ce sandwich (c'est-à-dire celui qui contient l'instance courante de **`Sandwich`** comme attribut).

Exercice 2. (1 point) Écrire les déclarations des attributs des classes **`Recette`**, **`Creation`** et **`Sandwich`**.

Exercice 3. (1 point) Dessiner une instance de **`Recette`** qui contient les recettes des trois sandwichs suivants (les ingrédients sont notés dans l'ordre) :

- le *matelot* : beurre, saumon, aneth ;
- le *parigot* : beurre, jambon ;
- le *grassouillet* : saumon, mayonnaise.

L'interface de la classe **`Recette`** est la suivante :

```
/** renvoie le nombre de sandwichs différents représentés dans l'arbre */
int nbSandwichs();

/** renvoie le nombre de sandwichs contenant l'ingrédient passé en paramètre représentés
dans l'arbre */
int nbSandwichs(Ingredient i);

/** supprime les branches qui ne contiennent pas de sandwich */
void grandNettoyage();

/** ajoute une recette dans l'arbre */
void sandwichDuMois(String nom, java.util.ArrayList<Ingredient> recette);
```

Exercice 4. (4,5 points) Écrire les deux méthodes **`nbSandwichs`** de la classe **`Recette`**. On rappelle qu'un sandwich peut contenir plusieurs fois le même ingrédient.

Exercice 5. (2,5 points) Écrire la méthode **`grandNettoyage`** de la classe **`Recette`** : toute branche de l'arbre qui ne contient pas de sandwich disparaît.

Exercice 6. (3,5 points) Écrire la méthode **`sandwichDuMois`** de la classe **`Recette`** : il s'agit d'ajouter un sandwich dont le nom et la liste dans l'ordre des ingrédients sont donnés en paramètre, en se contentant d'ajouter ce qui est nécessaire dans l'arbre des recettes, tout en tenant compte de ce qui existe déjà et sans modifier les autres recettes contenues dans cet arbre. On suppose que la classe **`Sandwich`** dispose d'un constructeur qui prend en argument une chaîne de caractères pour le nom et une instance de **`Creation`** pour la référence au nœud.

L'interface de la classe **`Sandwich`** est la suivante :

```
/** teste si tous les ingrédients pour le sandwich sont disponibles */
boolean ok();

/** affiche la liste des ingrédients dans l'ordre et les enlève des stocks, si le sandwich
peut être préparé */
void preparer();
```

Exercice 7. (2 points) Écrire la méthode **`ok`** de la classe **`Sandwich`**.

Exercice 8. (2,5 points) Écrire la méthode **`preparer`** de la classe **`Sandwich`** : celle-ci affiche la liste des ingrédients, dans l'ordre, pour faire le sandwich, s'il y en a suffisament en stock, et met le stock à jour ; sinon elle affiche un message disant qu'il faut faire les courses.