L3 informatique 2023-2024 : examen de première session

DURÉE : **2H** – DOCUMENTS ET DISPOSITIFS ÉLECTRONIQUES **INTERDITS**

Exercice 1 : classes, héritage, flot de données, exceptions et *threads*

Un *afficheur* est un objet qui affiche le contenu d'un fichier texte sur la sortie standard, ligne par ligne, avec un délai entre l'affichage de chaque ligne.

Exercice 1.1

Définissez la classe `Afficheur` avec les caractéristiques suivantes :

- attribut `source` : flot entrant de données permettant de lire un fichier ligne par ligne
- attribut `d` : délai entre l'affichage d'une ligne et l'affichage de la suivante (en secondes)
- constructeur permettant d'initialiser les attributs à partir d'un nom de fichier et d'un délai
- méthode `demarre`, sans paramètre : lance l'affichage du fichier ligne par ligne

Les éventuelles exceptions de la méthode `demarre` doivent être propagées.

Exercice 1.2

On souhaite pouvoir interrompre l'affichage d'un fichier. Dans ce but, définissez la classe `AfficheurArretable` qui hérite de `Afficheur` et dispose :

- d'un attribut supplémentaire `t` de type `Thread` initialisé dans le constructeur : il doit permettre l'exécution en parallèle de la méthode `demarre` héritée de `Afficheur`
- d'une redéfinition de `demarre` qui lance l'exécution de `t`
- d'une méthode `stop`, sans paramètre, qui interrompt l'exécution de `t`

Quand `t` est interrompu, l'affichage du fichier doit s'arrêter et le message `STOP` doit s'afficher.

Exercice 2 : interfaces, classes abstraites, généricité et collections

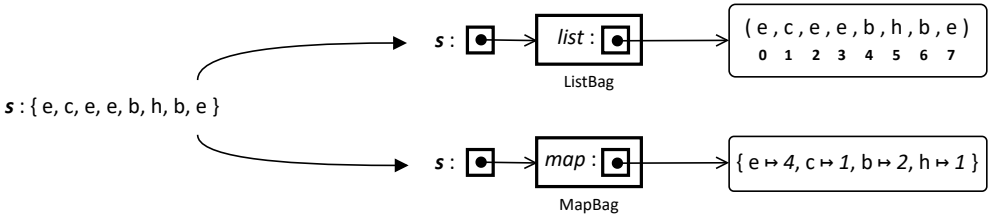
Un *multiensemble*, ou *bag*, est une sorte d'ensemble dont les éléments peuvent apparaître *plusieurs* fois (*e.g.* multiensemble `s` contenant des caractères : `{e, c, e, e, b, h, b, e}`).

On souhaite définir les multiensembles par une interface et trois classes. Les principes généraux sont décrits dans cette introduction. Les détails sont donnés dans les exercices à suivre.

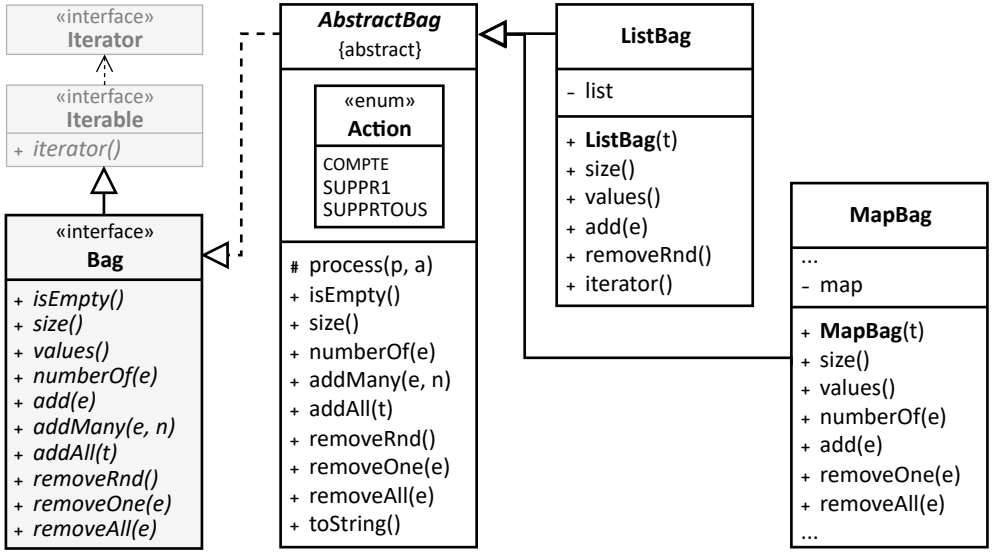
L'interface `Bag` représente un multiensemble *itérable* (*i.e.* qui hérite de l'interface `Iterable` de l'API standard, et peut donc être parcouru par un itérateur) :

```
interface Bag<E> extends Iterable<E> {
  boolean     isEmpty  ();           // indique si le multi-ensemble est vide
  int         size     ();           // nombre total d'éléments stockés
  int         values   ();           // nombre de valeurs différentes
  int         numberOf (E e);        // nombre d'occurrences de l'élément e
  void        add      (E e);        // ajoute l'élément e au multi-ensemble
  void        addMany  (E e, int n); // ajoute n fois l'élément e au multi-ensemble
  void        addAll   (E... te);    // ajoute tous les éléments fournis au multi-ensemble
  E           removeRnd();           // supprime et renvoie un élément choisi au hasard
  boolean     removeOne(E e);        // supprime 1 e si possible (renvoie true dans ce cas)
  int         removeAll(E e);        // supprime tous les e et renvoie leur nombre
  // Méthode héritée de Iterable<E> :
  Iterator<E> iterator ();           // fournit un itérateur du multiensemble
}
```

On souhaite mettre en œuvre cette interface en utilisant une *liste* (attribut `list` de la classe `ListBag`), et en utilisant un *tableau associatif* (attribut `map` de la classe `MapBag`). Illustration avec le multiensemble `s` et les deux mises en œuvre envisagées :



La plupart des méthodes de ces deux classes peuvent être mises en œuvre de la même façon. La classe abstraite `AbstractBag` rassemble ces méthodes communes :



Exercice 2.1

Définissez la classe abstraite `AbstractBag` qui met en œuvre l'interface `Bag` et contient :

1 – un type énuméré encapsulé `Action` comportant les valeurs `COMPTE`, `SUPPR1` et `SUPPRTOUS` ; et la méthode `process` qui cherche le ou les élément(s) du multiensemble qui satisfont un certain prédicat, et qui selon les cas supprime ce ou ces élément(s) :

```
protected int process(Predicate<E> p, Action a) { /* à compléter... */ }
```

Exemple avec `s` : `{e, c, e, e, b, h, b, e}` et `p` : prédicat testant l'égalité au caractère `'b'`
- `s.process(p, COMPTE)` renvoie le nombre de `'b'` (donc 2) et laisse `s` inchangé
- `s.process(p, SUPPR1)` supprime si elle existe la première occurrence de `'b'` et renvoie le nombre d'éléments supprimés (donc 1) ; à la fin, `s` vaut `{e, c, e, e, h, b, e}`
- `s.process(p, SUPPRTOUS)` supprime toutes les occurrences de `'b'` et renvoie le nombre d'éléments supprimés (donc 2) ; à la fin, `s` vaut `{e, c, e, e, h, e}`

Remarque : pour parcourir le multiensemble, il est nécessaire d'utiliser la méthode `iterator`.

2 – les méthodes `size`, `numberOf`, `removeOne` et `removeAll`, toutes réduites à une instruction `return` faisant appel à `process` avec les paramètres appropriés (lambda expression et action)

3 – les méthodes `isEmpty` (basée sur `size`), `addMany` et `addAll` (basées sur `add`), `removeRnd` (basée sur `size` et `iterator` ; provoque une `IllegalStateException` si le multiensemble est vide), et enfin la méthode `toString` qui renvoie une chaîne constituée des éléments séparés par des espaces et encadrés par des parenthèses (*e.g.* avec `{e, c, e, e, b, h, b, e}` : `"( e c e e b h b e )"`)

Exercice 2.2

Définissez la classe `ListBag` qui hérite de `AbstractBag` et contient les éléments suivants (dont le code est normalement *très court*) :

1 – un attribut `list` de type `List` contenant toutes les valeurs du multiensemble, et un constructeur qui ajoute les valeurs fournies en paramètre en nombre variable (par exemple : `Bag<Character> s = new ListBag<>('e', 'c', 'e', 'e', 'b', 'h', 'b', 'e');`)

2 – les méthodes `add`, `iterator` et `values` (`values` indique le nombre de valeurs *différentes*, *e.g.* 4 pour `{e, c, e, e, b, h, b, e}`), et les redéfinitions plus efficaces au niveau `ListBag` des méthodes `size` et `removeRnd` héritées de `AbstractBag` (sans appel à `process` ou `iterator`)

Exercice 2.3

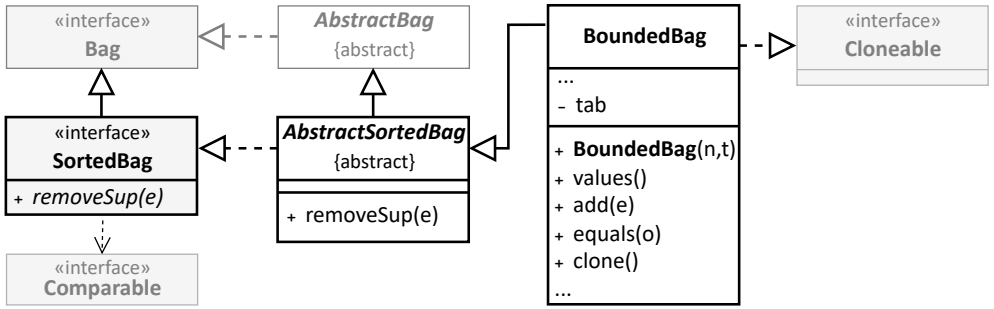
Définissez partiellement la classe `MapBag` qui hérite de `AbstractBag` et contient les éléments suivants (dont le code est également *très court*, à l'exception de `removeOne`) :

1 – un attribut `map`, tableau associatif dont les clés sont les éléments du multiensemble et les valeurs sont leurs nombres d'occurrences (nombres strictement positifs), et un constructeur qui ajoute les valeurs fournies en paramètre en nombre variable

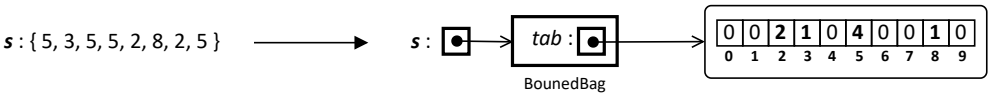
2 – les méthodes `add` et `values`, et les redéfinitions plus efficaces au niveau `MapBag` des méthodes `size`, `numberOf`, `removeOne` et `removeAll` héritées de `AbstractBag` (sans appel à `process`)

Exercice 3 : comparaison, égalité, clonage

On complète l'interface `Bag` avec une interface dérivée `SortedBag`, dont les éléments sont contraints à être comparables entre eux. La classe abstraite `AbstractSortedBag` hérite de `AbstractBag` et met en œuvre cette nouvelle interface. Enfin, la classe `BoundedBag` hérite de `AbstractSortedBag` et est en plus *cloneable* :



La classe `BoundedBag` représente des multiensembles *bornés* d'*entiers*, *i.e.* des multiensembles pouvant contenir des entiers compris entre 0 inclus et une certaine borne M exclue. Dans cette classe, les éléments sont les indices d'un tableau de taille M. La case d'indice i contient le nombre d'occurrences de l'élément i. Illustration avec un multiensemble `s` borné à 10 :



Exercice 3.1

Définissez `SortedBag` et la déclaration de `removeSup`, puis `AbstractSortedBag` et sa définition de `removeSup` : cette méthode prend en paramètre une valeur `e`, supprime les éléments strictement supérieurs à `e`, et renvoie le nombre d'éléments supprimés. Dans `AbstractSortedBag`, cette méthode est réduite à une instruction `return` faisant appel à `process` avec les paramètres appropriés (lambda expression et action).

Exercice 3.2

Définissez partiellement la classe `BoundedBag` qui hérite de `AbstractSortedBag`, met en œuvre `Cloneable`, et contient les éléments suivants :

1 – un attribut `tab`, tableau d'entiers, et un constructeur qui prend en paramètre la taille de `tab` et les valeurs initiales du multiensemble (en nombre variable)

2 – les méthodes `add` et `values`, ainsi que les méthodes `equals` et `clone` ; remarques :
- l'égalité de deux objets de type `BoundedBag` coïncide avec l'égalité de leurs tableaux
- la méthode `clone` doit être publique, ne doit pas provoquer d'exception, et doit renvoyer un objet de type `BoundedBag` complètement séparé de son receveur