Travaux Dirigés
Mise à niveau en JAVA
Collections & Généricité
–M1–

Exercice 1: Collections

Pour cet exercice, vous pouvez vous aider de la documentation en ligne de la classe ArrayList

- Créez un programme Java qui crée une collection (ArrayList) de noms de pays puis alimenter cette collection avec quelques valeurs et afficher la taille de la collection
- Complétez le programme pour afficher le contenu de la collection.
- Trouvez une méthode pour vider la collection et modifiez votre programme pour afficher un message d'erreur lorqu'elle est vide et afficher le contenu lorsqu'elle n'est pas vide.
- Après avoir de nouveau alimenté votre liste de pays, modifiez le nom d'un pays et affichez de nouveau la liste des pays.

  Conseil : Pour modifier le nom, il faut supprimer un élément (remove) et en ajouter un autre (add)

- Triez votre collection et ré-affichez la liste des pays.
  Pour trier notre collection, il faut utiliser la méthode sort . Si vous regardez la documentation de la classe ArrayList , vous ne trouvez pas cette méthode.

  Mais allez voir la documentation de la classe Collections : cette méthode y est présente.

  Et par chance, une ArrayList est aussi une collection . Comme Médor qui est un Chien est aussi un Animal . Ces notions seront vus ultérieurement. . .

Pour le moment, utilisons la syntaxe suivante Collections.sort(uneArrayList) pour trier une ArrayList.

Exercice 2: Map

On considère la classe Etudiant ci-après:

```
public class Etudiant {
    private String nom, prenom;
    private int age;
    private int [] notes;

    public Etudiant(String _nom, String _prenom, int _age) {
    nom = _nom;
    prenom = _prenom;
    age = _age;
    }
    public String getNom() {
    return nom;
    }
    void setAge(int _age) {
    age = _age;
    }
}
```

modifiez la classe pour pouvoir ajouter une note à un étudiant.

Dans la méthode main(), créez plusieurs étudiants, affichez maintenant pour chaque étudiant la liste de ses notes; affichez aussi la moyennes de ses notes; enfin, affichez la moyenne de tous les étudiants.

On veut intégrer la gestion des étudiant dans celle d'une scolarité. Pour une année particulière, une scolarité gère une collection de promotions. Chaque promotion est nommée et comprend une liste d'étudiants.

- Une promotion d'étudiants représente tous les étudiants d'une même classe (IUP1, IUP2 ou IUP3 par exemple)
- La scolarité gère un ensemble de promotions.

Développez les classes Promotion et Scolarité : outre les fonctions classiques :

- ajout et retrait d'une promotion,
- ajout et retrait d'un étudiant dans une promotion,

la scolarité doit pouvoir :

- retrouver un étudiants par son nom et
- retrouver la promotion d'un étudiant.

Quand un étudiant est ajouté dans une promotion, il faut tester que cet étudiant n'existe pas déjà dans la promotion.

Utilisez un Map pour programmer les recherches et le test d'unicité.

De plus, on on besoin d'une fonction d'affichage pour chacune des classes :

- l'affichage d'une promotion doit afficher le nom de la promotion, la liste des étudiants et la moyenne des notes,
- l'affichage de la scolarite doit afficher l'année et toutes ses promotions.

Pour chacune des classes développées, il est demandé de mettre en oeuvre et d'utiliser equals et toString.

Exercice 3: Jockers

```
1 private static void print(List<Object> list) {
2     for(Object o:list)
3         System.out.println(o);
4 }
5 public static void main(String[] args) {
6     List<String> list=Arrays.asList("foo", "toto");
7     print(list);
```

- Pourquoi le code plus haut ne compile t-il pas ?
- Comment le modifier pour qu'il compile (sachant qu'on veut pouvoir afficher également une liste d'entiers par la même méthode) ?
- Ajoutez une méthode statique prenant en argument une liste d'objets implémentant CharSequence et en affichant la longueur de chaque objet.

Exercice 4: Généricité

On souhaite écrire un programme permettant à l'utilisateur de manipuler des ensembles au sens mathématique :

- Les valeurs contenues dans un ensemble ne sont pas ordonnées,
- Les valeurs ne peuvent être présentes qu'une seule et unique fois dans l'ensemble.

On souhaite pouvoir manipuler des ensembles d'entiers, des ensembles de réels, des ensembles de ... et écrire le moins de code possible !

Les étapes à suivre sont les suivantes :

- Écrivez la déclaration de la classe Ensemble qui contient :
  - Une variable d'instance, liste de type ArrayList, qui contiendra les différentes valeurs de l'ensemble (type générique).
  - Une variable d'instance, card de type entier, qui contiendra le cardinal de l'ensemble.
  - Le constructeur qui initialise un ensemble à l'ensemble vide.
  - Un accesseur privé, setCard, qui recalcule le cardinal de l'ensemble.
  - Une méthode, ajoute, qui permet d'ajouter un élément à un ensemble. Si l'élément n'est pas déjà dans l'ensemble, la fonction retournera VRAI. Elle retournera FAUX dans le cas contraire.
  - Une méthode, enleve, qui permet d'enlever un élément à un ensemble. Si l'élément est présent dans l'ensemble, la fonction retournera VRAI. Elle retournera FAUX dans le cas contraire.
  - Une surcharge de la méthode, toString, qui indique le cardinal de l'ensemble ainsi que la valeur de tous les éléments qui le composent.
- Écrivez la déclaration de la classe Principal qui:
  - Affiche un petit menu :
    1- Ajouter un élément,
    2- Enlever un élément,
    3- Arrêter
  - En fonction du choix de l'utilisateur : Ajoute un élément : si l'ajout a été possible affiche le nouvel état de l'ensemble, puis affiche à nouveau le menu ; si l'ajout n'a pu se faire affiche simplement un message.
  - Enlève un élément : si la suppression a été possible affiche le nouvel état de l'ensemble, puis affiche à nouveau le menu ; si la suppression n'a pu se faire affiche simplement un message.
  - Arrête le programme.