Atelier 1 : Généricité et Collections

Objectif Général	Paramétrer une classe ou une méthode, manipuler une collection
Objectifs spécifiques	-Comprendre et implémenter des classes paramétrables -Connaitre et utiliser les collections pour gérer les données applicatives
Volume Horaire	2 séances (6 Heures)

1. Rappel

On rappelle quelques interfaces de collections importantes :

- List<E> : liste d'éléments avec un ordre donné, accessibles par leur indice.
- Set<E> : ensemble d'éléments sans doublons
- Map<K,E> : ensemble d'associations (clé dans K, valeur dans E), tel qu'il n'existe qu'une seule association faisant intervenir une même clé.

Ces interfaces peuvent évidemment être composées les unes avec les autres.
Exemple : un ensemble de séquences d'entiers se note Set<List<Integer>>.

2. Série d'exercices

1. Exercice 1

1) Écrire une classe générique ***Triplet*** permettant de manipuler des triplets d'objets d'un même type. On la dotera :
   - d'un constructeur à trois arguments (les objets constituant le triplet),
   - de trois méthodes d'accès *getPremier*, *getSecond* et *getTroisieme*, permettant d'obtenir la référence de l'un des éléments du triplet,
   - d'une méthode *affiche* affichant la valeur des éléments du triplet.

2) Écrire un petit programme utilisant cette classe générique pour instancier quelques objets et exploiter les méthodes existantes.

2. Exercice 2

Écrire une classe générique ***TripletH*** semblable à celle de l'exercice précédent, mais permettant cette fois de manipuler des triplets d'objets pouvant être chacun d'un type différent.

Écrire un petit programme utilisant cette classe générique pour instancier quelques objets et exploiter les méthodes existantes.

3. Exercice 3

Écrire un programme permettant de remplir une liste de chaine de caractère. Ensuite, afficher les éléments de cette liste en utilisant :

- La boucle for.
- La boucle foreach.
- Iterator.

Afficher les éléments de cette liste dans un ordre croissant.

4. Exercice 4

1. Définir une classe Stagiaire avec les attributs suivants : *nom et prénom.*
2. Définir un **constructeur** permettant d'initialiser les attributs d'un objet stagiaire par des valeurs saisies par l'utilisateur.
3. Définir la méthode **toString ( )** permettant d'afficher les informations d'un stagiaire.
4. Écrire un programme testant la classe Stagiaire.
5. Déclarer une collection d'objet stagiaire.
6. Ajouter des stagiaires dans la collection.
7. Afficher la liste des stagiaires.
8. Modifier la classe de teste pour afficher la liste des stagiaires triée dans un ordre alphabétique des noms, si des stagiaires ont le même nom, ils seront triés en ordre alphabétique des prénoms.
   Pour définir un ordre de tri, la classe stagiaire doit implémenter l'interface Comparable, ensuite redéfinir la méthode compareTo ().

5. Exercice 5

Écrire un programme qui construit une collection triée contenant n nombres entiers (représentés par des objets Integer) tirés au hasard dont la valeur est comprise entre 0 et 1000. A votre choix, la valeur de n est lue au début de l'exécution ou est un argument du programme. Ensuite, ce dernier affiche la collection construite afin qu'on puisse constater qu'elle est bien triée.

- Dans la première version du programme la collection est une sorte de List<Integer> (par exemple un ArrayList ou une LinkedList) que vous triez, après la construction, en utilisant une méthode statique ad hoc de la classe Collection ns.

- Dans une deuxième version, la collection est une sorte de Set<Integer> (c'est-à-dire un HashSet ou un TreeSet, mais avez-vous le choix ?) si bien qu'elle est constamment triée

6. Exercice 6

Nous disposons d'une classe générique Rayon destinée à représenter des rayons dans un magasin. Un rayon contient des produits. L'une des méthodes consiste en particulier à réaliser un listing des produits en rayon et pour cela nous supposons que les éléments dans les rayonnages ont une méthode String etiquette().

```
public class Produit{
      .......
      public String etiquette(){.....}
      }
      class Rayon{
            private ArrayList<Produit> contenu=new ArrayList<Produit>(); ....
            public String listingContenu()
            {
                  String listing="";
                  for (Produit c:contenu)
                      listing += c.etiquette();
                      return listing;
             }
        ....
}
```

1) Complétez la classe Produit par un attribut représentant son étiquette, des accesseurs et des constructeurs. Complétez la classe Rayon par un constructeur et une méthode permettant de mettre un produit en rayon.
2) Transformez la classe Rayon en classe générique, paramétrée par le type des éléments que l'on veut mettre en rayon. Ces éléments doivent disposer d'une étiquette mais ne sont pas forcément des produits.
3) Dans la fonction pricipale main, créez une classe représentant des rayons de Livres, puis des rayons de Produits.

7. Exercice 7

- **Ecrire les classes Employe et departement**

  Un employé est caractérisé par un cin, un matricule, un nom et un prenom.
  Un département est caractérisé par son identifiant et son nom. Chaque classe possède :
  - Deux constructeurs dont un sans paramètre.
  - Les getters et les setters o La méthode toString()
  - La méthode equals

- **Créez l'interface InterfaceSociete public interface InterfaceSociete {**

```
    public void ajouterEmployeDepartement(Employe e,Departement d);
    public void supprimerEmploye(Employe e);
    public void afficherLesEmployesLeursDepartements();
    public void afficherLesEmployes();
    public void afficherLesDepartements();
    public void afficherDepartement(Employe e);
    public boolean rechercherEmploye(Employe e);
    public boolean rechercherDepartement(Departement e);
}
```

- **Créer les classes SocieteHashMap et SocieteTreeMap**

  Construire les classes SocieteHashMap et SocieteTreeMap qui implémentent l'interface InterfaceSociete et qui reposent sur l'utilisation respective d'un HashMap et d'un TreeMap pour la gestion des employés ainsi que leurs départements respectifs.