2024 -2025

TP 2 - JAVA

Exercice 1: Tableaux d'entiers

Dans cet exemple, on s'intéresse à la manipulation des tableaux d'entiers. On considère que chaque élément du tableau représente les nombre de pages des livres disponibles dans la bibliothèque de l'EFREI. (Classe TestTabelau qui contient une méthode main avec tout le code)

1. Déclarer un tableau contenant les nombre de pages suivantes : 120, 350, 200, 500, 175, 90, 300.
2. Afficher le contenu du tableau.
3. Calculer et afficher la somme totale des pages de tous les livres.
4. Trouver et afficher le nombre de pages minimum.
5. Trouver et afficher le nombre de pages moyen.
6. Trier le tableau (ordre croissant) puis ré-afficher après le trie.

Résultat attendu:

```
Affichage des nombres de pages:
Livre 1: 120 pages
Livre 2: 350 pages
Livre 3: 200 pages
Livre 4: 500 pages
Livre 5: 175 pages
Livre 6: 90 pages
Livre 7: 300 pages
Somme totale des pages: 1735
Nombre  minimum : 90
Moyenne:  247
```

Exercice 2 : ArrayList

Utiliser des **ArrayList** en Java peut rendre la gestion de collections d'objets plus flexible par rapport aux tableaux classiques. Voici un rappel sur les **ArrayList** et leurs différences par rapport aux tableaux d'objets.

☐ *Tableaux d'objets*

1. **Taille fixe**
2. **Accès direct** : Vous pouvez accéder aux éléments d'un tableau par leur index (par exemple, tableau[0]).

3. **Stockage** : Les tableaux peuvent contenir des types primitifs et des objets, et ils sont plus efficaces en termes de mémoire lorsque vous connaissez le nombre d'éléments à l'avance.

- ☐ ***ArrayList***

1. **Taille dynamique** : ArrayList peut grandir ou rétrécir dynamiquement. Vous pouvez ajouter ou supprimer des éléments sans avoir à créer un nouveau tableau.
2. **Méthodes pratiques** : ArrayList fournit de nombreuses méthodes utiles (comme add(), remove(), contains(), etc.) qui facilitent la gestion des éléments.
3. **Type d'objets** : ArrayList ne peut contenir que des objets. Pour les types primitifs, Java utilise des wrappers (comme Integer pour int, Double pour double, etc.).
4. **Performance** : L'accès par index est toujours rapide, mais l'ajout d'éléments peut être plus coûteux si l'ArrayList doit être redimensionné.

Travail à faire: Exécutez le code suivant:

```
import java.util.ArrayList;
import java.util.Collections;

public class Bibliotheque {
	public static void main(String[] args) {
	// Déclaration de l'ArrayList et ajout des nombres de pages
	ArrayList<Integer> pages = new ArrayList<>();
	pages.add(120);
	pages.add(350);
	pages.add(200);
	pages.add(500);
	pages.add(175);
	pages.add(90);
	pages.add(300);

	// Affichage du contenu de la liste
	System.out.println("Nombre de pages des livres : " + pages);

	// Calcul et affichage de la somme totale des pages
	int somme = 0;
	for (int page : pages) {
	somme += page;
	}
	System.out.println("Somme totale des pages : " + somme);

	// Trouver et afficher le nombre de pages minimum
	int minPages = Collections.min(pages);
	System.out.println("Nombre de pages minimum : " + minPages);

	// Trier et afficher la liste
	Collections.sort(pages);
	System.out.println("Liste triée des nombres de pages : " + pages);
```

```
        }
}
```

Exercice 3: Tableaux d'objets (contient 4 parties)

Dans le cadre de la gestion des livres d'une bibliothèque, on souhaite organiser les livres sur des étagères selon des domaines spécifiques (littérature, chimie, informatique, etc.). Pour cela, on va utiliser deux classes : une classe Livre pour représenter chaque livre, et une classe Etagere pour gérer les livres placés sur une étagère donnée.

Partie 1 : Classe Livre Créez une classe Livre qui représente un livre avec les attributs suivants :

- **titre** : Le titre du livre (chaîne de caractères).
- **code** : Le code unique du livre (entier).
- **auteur** : L'auteur du livre (chaîne de caractères).
- **nbPages** : Le nombre de pages du livre (entier).
- **anneeEdition** : L'année d'édition du livre (entier).

Exigences :

- Implémentez un **constructeur** qui permet d'initialiser tous ces attributs lors de la création d'un livre.
- Implémentez les **méthodes getters** et **setters** pour accéder à ces attributs.
- Implémentez une méthode **toString**() pour afficher les informations d'un livre dans un format lisible.

Partie 2 : Classe Etagere Créez une classe Etagere qui représente une étagère dans une bibliothèque. Cette classe doit contenir :

- **code** : Un code unique pour identifier l'étagère (entier).
- **domaine** : Le domaine des livres sur cette étagère (chaîne de caractères). Par exemple : "Littérature", "Chimie", "Informatique".
- capacite: nombre max de livres à mettre dans cette étagère
- **livres** : Un tableau de livres placés sur cette étagère.

1. **Ajout d'un livre** :
    - Implémentez une méthode ajouterLivre(Livre livre) pour ajouter un livre à l'étagère.

2. **Suppression d'un livre** :
    - Implémentez une méthode supprimerLivre(int code) pour supprimer un livre en utilisant son code unique.
3. **Somme des pages** :
    - Implémentez une méthode sommePages() pour calculer et retourner la somme du nombre total de pages de tous les livres de l'étagère.
4. **Recherche d'un livre par titre** :
    - Implémentez une méthode chercherLivreParTitre(String titre) qui retourne le **code** du livre recherché. Si le livre n'existe pas, retournez un message indiquant que le livre n'a pas été trouvé.
5. **Recherche des petits livres** :
    - Implémentez une méthode chercherPetitsLivres(int seuil) qui affiche tous les livres ayant un nombre de pages inférieur à un seuil donné.
6. **Affichage des nouveaux livres** :
    - Implémentez une méthode afficherNouveauxLivres(int anneeActuelle) qui affiche les **titres** et **codes** des livres édités durant l'année en cours.
7. **Changer le nombre de pages d'un livre** :
    - Implémentez une méthode changerNbPages(int code, int nbPages) pour modifier le nombre de pages d'un livre identifié par son code.
8. **Affichage des livres** :
    - Implémentez une méthode afficherLivres() qui affiche les détails de tous les livres présents sur l'étagère.
9. **Trie:**
    - Trier les livres par nombre de pages
10. **Recherche :**
    - la méthode **chercherTitresParAuteur**, qui retourne *un tableau de chaînes de caractères* contenant les titres des livres d'un auteur spécifié.

Partie 3 : Programme principal

Créez un programme dans une classe BibliothequeTest qui permet de :

1. Créer une étagère pour un domaine donné (par exemple, "Littérature").
2. Ajouter plusieurs livres à cette étagère. (ajouter les livres de l'exemple d'utilisation ci-dessous)
3. Utiliser les différentes méthodes de la classe Etagere pour :
    - Afficher les livres présents sur l'étagère.
    - Calculer et afficher la somme totale des pages.
    - Rechercher un livre par son titre.
    - Afficher les livres ayant un nombre de pages inférieur à un seuil donné.
    - Afficher les titres et codes des livres édités durant l'année en cours.
    - Modifier le nombre de pages d'un livre en fonction de son code.
    - Supprimer un livre par son code.

Afficher de nouveau la liste des livres après suppression.

Exemple d'utilisation :

- Créez deux étagères :
  - Une pour le domaine **"Chimie"**.
  - Une pour le domaine **"Informatique"**.

Livres à ajouter :

Etagère Chimie :

1. **"La chimie organique"**, code : 201, auteur : John McMurry, 1200 pages, édition : 2019
2. **"Introduction à la chimie analytique"**, code : 202, auteur : Daniel C. Harris, 800 pages, édition : 2021
3. **"Chimie générale"**, code : 203, auteur : Raymond Chang, 1000 pages, édition : 2018

Etagère Informatique :

1. **"Introduction aux algorithmes"**, code : 301, auteur : Thomas H. Cormen, 1300 pages, édition : 2020
2. **"Design Patterns"**, code : 302, auteur : Erich Gamma, 400 pages, édition : 1994
3. **"Java pour les débutants"**, code : 303, auteur : Herbert Schildt, 700 pages, édition : 2019

Pour aller plus loin…

Partie 4: arrayList

Utiliser des **ArrayList** en Java peut rendre la gestion de collections d'objets plus flexible par rapport aux tableaux classiques. Voici un rappel sur les **ArrayList** et le

Travail à faire: Refaire la partie 2 en créant la classe EtagereV2 utilisant un ArrayList au lieu du tableau de livre. N'oubliez pas de tester la nouvelle classe.

Exercice 4

Créez une classe `Etudiant` ayant les attributs suivants :

- `numEtudiant` : un identifiant unique pour l'étudiant (de type entier).
- `nom` : le nom de l'étudiant (de type chaîne de caractères).
- `prenom` : le prénom de l'étudiant (de type chaîne de caractères).

- `age` : l'âge de l'étudiant (de type entier).
- `specialite` : la spécialité de l'étudiant (de type chaîne de caractères).
- `moyenne` : la moyenne des notes de l'étudiant (de type décimal).

La classe `Etudiant` doit également comporter :

1. Deux constructeurs :
    - Un constructeur par défaut qui initialise les attributs à des valeurs par défaut.
    - Un constructeur qui prend tous les attributs en paramètres et les initialise.
2. Une méthode `toString()` qui retourne une représentation sous forme de chaîne de caractères de l'objet `Etudiant`, incluant tous ses attributs.
3. Des méthodes **getters** et **setters** pour accéder et modifier les attributs.

Partie 2 : Classe `Groupe`

Créez une classe `Groupe` qui représente un groupe d'étudiants(vous pouvez utiliser un tableau ou une collection adaptée). Ajouter puis tester des methodes que vous trouvez utiles.

Exemple de méthodes

- **Affichage des étudiants** : une méthode pour afficher tous les étudiants du groupe en utilisant la méthode `toString()` de la classe `Etudiant`.
- **Trier par âge** : une méthode qui trie les étudiants du groupe en fonction de leur âge, du plus jeune au plus âgé.
- **Rechercher par spécialité** : une méthode qui prend un numéro d'étudiant en paramètre et qui retourne la spécialité de l'étudiant correspondant. Si l'étudiant n'existe pas, un message approprié doit être affiché.
- **Afficher <u>les prénoms</u> par âge** : une méthode qui prend un âge en paramètre et qui affiche les prénoms des étudiants ayant cet âge (notez que **le type de retour est une liste)**