Implémentation et Utilisation des Listes Chaînées, Files et Piles

Exercice 1 : Implémentation d'une liste chaînée

1. Créer une classe `Noeud` contenant une valeur et une référence vers le suivant.
2. Créer une classe `ListeChainee` avec les méthodes :
   - `ajouter_fin(valeur)` : ajoute un élément à la fin.
   - `afficher()` : affiche tous les éléments.
   - `rechercher(valeur)` : retourne vrai si la valeur est dans la liste.
   - `supprimer(valeur)` : supprime un élément donné.

a. Écrire le code.
b. Tester avec une liste d'entiers : insérer `10, 20, 30`, afficher, rechercher `20`, supprimer `10`, puis afficher de nouveau.

Exercice 2 : Implémentation d'une file

1. Implémenter une classe `File` en utilisant la classe `Noeud`.
2. Méthodes à écrire :
   - `enfiler(valeur)` : ajoute un élément en fin de file.
   - `defiler()` : enlève et retourne l'élément en tête.
   - `est_vide()` : teste si la file est vide.
   - `afficher()` : affiche les éléments de la file.

a. Créer une file vide.
b. Enfiler `"Client1"`, `"Client2"`, `"Client3"`.
c. Afficher la file.
d. Dépiler un élément et réafficher la file.

Exercice 3 : Application – Gestion d'un buffer

Un programme reçoit des messages : `["A", "B", "C", "D"]`.

a. Enfiler chaque message dans une file.
b. Traiter les messages en les retirant un par un de la file et en les affichant.

Exercice 4 : Implémentation d'une pile

1. Implémenter une classe `Pile` en utilisant soit une liste chaînée, soit une simple liste Python. 2. Méthodes à écrire :
   - `empiler(valeur)` : ajoute un élément au sommet.
   - `depiler()` : retire et retourne l'élément au sommet.
   - `est_vide()` : teste si la pile est vide.
   - `sommet()` : retourne l'élément au sommet sans le retirer.

a. Créer une pile vide.
b. Empiler les nombres `1, 2, 3`.
c. Afficher le sommet.
d. Dépiler un élément et réafficher la pile.

Exercice 5 : Application – Annulation d'actions

Simuler un éditeur de texte avec une pile :

a. Les actions `"écrire A"`, `"écrire B"`, `"écrire C"` sont empilées.
b. Lorsque l'utilisateur appuie sur **Annuler**, on dépile la dernière action et on affiche la pile restante.

Exercice 6 : Application - Centre d'Appels

A. Système de Gestion d'Appels

Implémentez une classe `CentreAppels` qui gère des appels téléphoniques avec : • `arriver_appel(self, id_appel, numero, urgence)` : `urgence = True/False` • `traiter_appel(self)` : traite le prochain appel (priorité aux urgents)

• `statistiques(self)` : affiche le nombre d'appels en attente, traités, etc. **Structure :** Utilisez DEUX files : une pour les appels normaux, une pour les urgents. **B. Scénario de Test**

Simulez cette séquence d'événements :

1. `Appel normal #1` (non urgent)
2. `Appel urgent #2`
3. `Appel normal #3`
4. Traitement d'un appel
5. `Appel urgent #4`
6. Traitement de 3 appels

Vérifiez que les urgents sont toujours traités en premier.

Questions

1. Quelle est la différence fondamentale entre une **pile** et une **file** ?
2. Pourquoi la **liste chaînée** est-elle plus flexible qu'un **tableau** ?
3. Donnez un exemple d'application réelle pour :
   - une pile
   - une file
   - une liste chaînée