Introduction

Ce document présente une série d'exercices sur les structures de données avancées en Java, avec un rappel des concepts clés pour chaque structure : **Liste, Pile, File et Arbre**.

Listes

Rappel du concept

Une **liste** est une collection ordonnée d'éléments où chaque élément peut être accédé par un indice. Java propose plusieurs implémentations de listes, comme `ArrayList` ou `LinkedList`.

Exercices

1. Créer une `ArrayList` d'entiers et ajouter les nombres de 1 à 10. Afficher la liste.
2. Supprimer tous les nombres pairs de la liste.
3. Inverser l'ordre des éléments de la liste.
4. Utiliser une `LinkedList` pour implémenter une liste d'attente (simuler un système de tickets).

Pile (Stack)

Rappel du concept

Une **pile** suit le principe **LIFO** (Last In, First Out). Le dernier élément ajouté est le premier à être retiré. En Java, on peut utiliser la classe `Stack` ou les `Deque` pour implémenter une pile.

Exercices

1. Créer une pile de chaînes de caractères et y empiler les jours de la semaine.
2. Dépiler tous les éléments et les afficher.
3. Écrire une fonction qui vérifie si une expression entre parenthèses est bien équilibrée en utilisant une pile.

File (Queue)

Rappel du concept

Une **file** suit le principe **FIFO** (First In, First Out). Le premier élément ajouté est le premier à être retiré. En Java, on utilise `Queue`, `LinkedList` ou `PriorityQueue`.

Exercices

1. Créer une file d'attente de nombres entiers et y ajouter 5 nombres.
2. Retirer les éléments un par un et les afficher.
3. Simuler un système de gestion de clients avec une file, où chaque client est servi dans l'ordre d'arrivée.

Arbres

Rappel du concept

Un **arbre** est une structure hiérarchique composée de **nœuds**, avec un nœud racine et des sous-arbres. Exemple : arbre binaire de recherche (Binary Search Tree, BST) où chaque nœud à gauche est plus petit et à droite plus grand.

Exercices

1. Créer une classe `Node` représentant un nœud d'arbre avec un entier et deux enfants.
2. Construire un petit arbre binaire avec au moins 5 nœuds.
3. Écrire une fonction pour parcourir l'arbre en **in-order** et afficher les valeurs.
4. Écrire une fonction pour rechercher une valeur dans un arbre binaire.

Exercices avancés combinés

1. Utiliser une `Stack` pour inverser une `Queue`.
2. Utiliser un `BST` pour stocker des nombres et afficher la liste triée des nombres.
3. Implémenter une `LinkedList` circulaire et simuler un jeu de rotation de joueurs.

Exemple de code Java pour rappel

```java
// Liste avec ArrayList
import java.util.ArrayList;

public class ListExample {
    public static void main(String[] args) {
        ArrayList<Integer> liste = new ArrayList<>();
        for(int i=1; i<=10; i++) {
            liste.add(i);
        }
        System.out.println("Liste: " + liste);
        liste.removeIf(x -> x % 2 == 0); // supprime pairs
```



```
12          System.out.println("Liste apr s suppression des pairs: " +
                liste);
13      }
14  }
```