Structures de données

L'écriture sur des exemples simples de plusieurs implémentations d'une même structure de données permet de faire émerger les notions d'interface et d'implémentation, ou encore de structure de données abstraite.

Contenus	Capacités attendues	Commentaires
Structures de données, interface et implémentation.	Spécifier une structure de données par son interface. Distinguer interface et implémentation. Écrire plusieurs implémentations d'une même structure de données.	L'abstraction des structures de données est introduite après plusieurs implémentations d'une structure simple comme la file (avec un tableau ou avec deux piles).
Listes, piles, files : structures linéaires. Dictionnaires, index et clé.	Distinguer des structures par le jeu des méthodes qui les caractérisent. Choisir une structure de données adaptée à la situation à modéliser. Distinguer la recherche d'une valeur dans une liste et dans un dictionnaire.	On distingue les modes FIFO *(first in first out)* et LIFO *(last in first out)* des piles et des files.

Cette leçon sera une introductuion aux Types abstraits de données (TAD) et à la découverte des listes.

Les piles, files et listes seront traitées dans un chapitre prochain, chacune d'elles sera implantée sous forme de TAD.

I. Exemples

1. Liste place avion

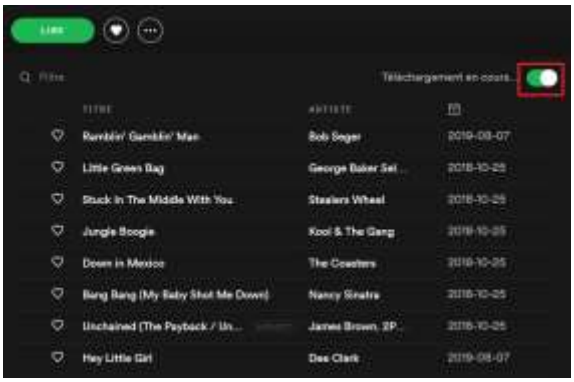
Les compagnies aériennes gèrent la liste d'embarquement sur les avions.

Dans l'image ci-contre l'avion dispose de 72 places numérotées de 1 à 72.
A chaque place correspond le numéro d'un passager.
Au début les sièges ne sont associés à aucun passager.
Les « actions » possible sur une liste sont :

- Ajouter un passager,
- Supprimer un passager,
- Quel passager est à une place donnée ?
- Combien de places sont utilisées dans l'avion ?
- Reste-t-il des places libres ?
- …….

L'avion disposant de 72 places, la taille de liste est fixée à la création de la liste. On parle de liste statique.

2. Play liste musicale



Dans le cas d'une liste musicale personnelle, les « actions » possibles :

- ajouter une chanson,
- supprimer une chanson
- quelle chanson est la $n^{ème}$ dans la liste ?
- lire séquentiellement,
- lire aléatoirement
- …..

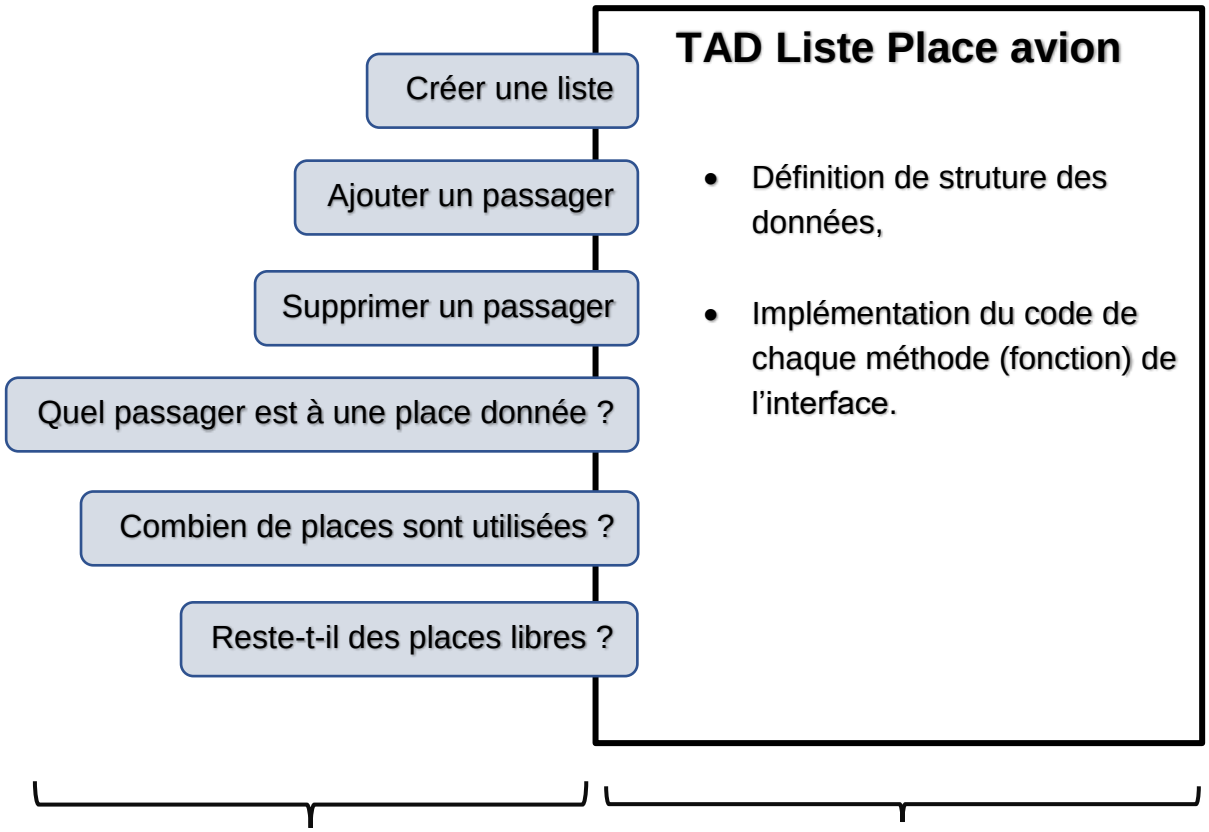
A la création de la liste, la taille (nombre de chansons) n'est pas connue, sa taille évolue au cours de sa « vie de liste », on parle de liste dynamique.

II. Les Types Abstraits de données : TAD

Dans les deux exemples précédents les « actions » possibles sur les listes ont été définies.
La « mécanique interne » n'a pas été présentée et l'organisation des données n'a pas été donnée.

Il s'agit donc d'un **Type Abstrait de Donnée**, pour les deux exemples.

On peut représenter le TAD liste place avion de la manière ci-dessous :



Interface :
Le « programmeur **utilsateur »** du TAD, ne voit pas comment sont organisées les données, ni le code qui y est associé. C'est **abstrait** pour lui : c'est une « **boite noire** ».
Avant d'être « utilisée » la liste doit être créée.
L'action sur le TAD se fait par les méthodes de **interface**.

Le « programmeur **concepteur »** du TAD, chosit la structure des données, et l'**implémentation** du **code** correspondant à l'interface.
C'est dans le choix de l'organisation des données et du code qui y est associé, que la liste sera statique ou dynamique.

Ce TAD peut être utilisé pour autant de liste que l'on veut : on a réalié un composant logiciel , on peut en faire un package.
Le langage Ada est un des premiers langages à utiliser la notion de TAD.

III. Les listes :

1. Définition

Propriétés d'une liste :

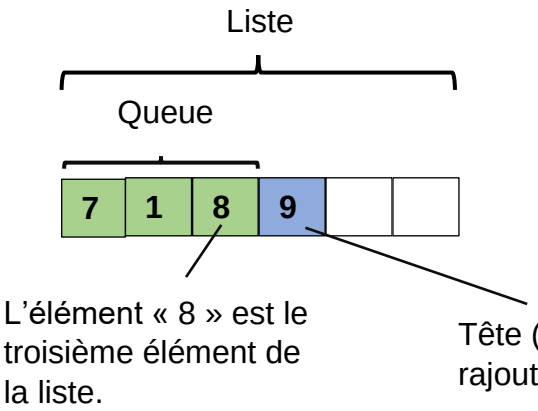
- C'est une structure abstraite de données permettant de regrouper des données sous une forme séquentielle.
- Elle est constituée d'éléments d'un même type, chacun possédant un rang. Une liste est évolutive : on peut ajouter ou supprimer n'importe lequel de ses éléments.

Méthodes ou actions :

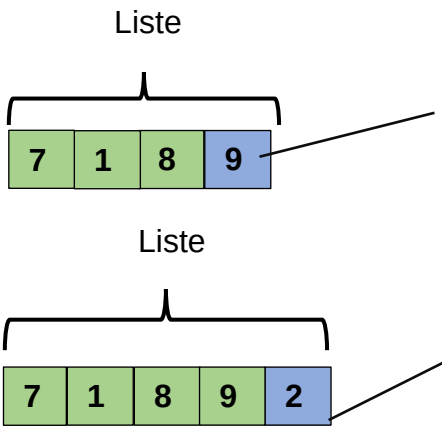
- Les principales actions sur les listes sont : Créer_liste(), ajouter_un_element() Lire_element(), taille_liste()
- Il est possible de rajouter des actions « sur la liste » à condition de ne pas changer ses propriétés.

Remarque :

La liste peut-être de taille fixe ou dynamique,



Cette liste est une liste de taille fixe.
Elle peut contenir 6 éléments.
Les deux dernières cases sont vides.
En mémoire, l'ordinateur range chaque case du tableau dans une zone « contigüe »



Cette liste est une liste de taille dynamique.

« 9 » est le dernier élément rajouté à la liste, sa taille est de 4.

L'élément « 2 » a été rajouté à la liste, sa taille est de 5,
en mémoire, l'ordinateur range chaque case du tableau dans des zones non « contigües».

Le langage de programmation Lisp (inventé par John McCarthy en 1958) a été l'un des premiers langages de programmation à introduire cette notion de liste (Lisp signifie " list processing ").

2. Exercices

Exercice 1 :

Soit le TAD liste qui a pour interface les méthodes suivantes :

- Creer_liste() -> liste
- Ajouter (liste,element) ->liste
- Lire (liste, rang) -> element
- Supprimer (liste, element) -> liste

Un programme contient les lignes de codes suivantes :

```
L1= Creer_liste()
L1= Ajouter (L1, 4)
L1= Ajouter (L1, 7)
L1= Ajouter (L1, 12)
```

Q 1. Quel est le style de programmeur qui écrit ces lignes ? (« codeur utilisateur de TAD » ou « codeur concepteur de TAD »).

Q 2. Que contient L1 à la fin de ces quatre lignes de code ?

La suite du programme est donné ci-dessous :

```
L2= Creer_liste()
L2= Ajouter (L1, 25)
Element = Lire (L1, 2)
L2= Ajouter (L2, Element)
```

Q 3. Que contient L2 à la fin de ces quatre lignes de code supplementaires ?

Exercice 2 : Création d'un TAD liste

Le script « TAD_liste.py » est donné ci-dessous :

```
''' Ce type abstrait Liste permet de travailler sur les listes de nombres ,
1 avant d'utiliser la liste,
il faut créer une liste vide avec la fonction creer_liste(taille) -> liste
2 pour inserer un élément ajouter (liste, element) -> liste
'''

def creer_liste(taille:int )->list:
    ''' crée un liste de taille fixe précisé en parametre.
        et retourne la liste (vide)
    '''
    # secret de fabrication:
    # la structure des données est la suivante:
    # l = [taille, [nb_éléments, élément1, élément2,élément3,élément4,.....]]:
    # A la créartion de la liste :
    #l = [taille, [0,None, None,None,....]
    l = [taille,[None]*(taille+1)]
    l[1][0] =0
    #ajouter un print pour afficher la structure des données
    return l
```

```python
def ajouter  (l:list, element:int)->list:
    ''' Ajoute element à la condition qu'il reste de la place dans la liste.
        Si la liste est pleine elle reste inchangée.
    '''
    la_liste = l[1]
    taille = l[0]
    nb_element = la_liste [0]
    if taille == nb_element:
        #ajouter un print pour afficher la structure des données
        return l
    else:
        nb_element +=1
        la_liste[0] = nb_element
        la_liste[nb_element] = element
        l[1] = la_liste
        #ajouter un print pour afficher la structure des données
        return l
```

```python
def afficher(l):
    ''' Affiche la liste créee lors de la mise en oeuvre du TAD'''
    #  ajouter un print pour afficher les données seules.
    # ne pas afficher la structure interne du TAD!!! SERCRET DE FABRICATION


if __name__ == "__main__":
    liste1 = creer_liste(3)
    liste2 = creer_liste(3)

    ajouter  (liste1, 4)
    ajouter  (liste1, 5)
    ajouter  (liste1, 2)
    ajouter  (liste1, 3)
```

Le code du TAD doit être modifié pour compléter des parties de code et rajouter les méthodes à l'interface.

Q 1. Quel est le style de programmeur qui écrit ces lignes ? (« codeur utilisateur de TAD » ou «codeur concepteur de TAD »).

Q 2. Expliquer comment est implantée la liste dans ce TAD

Q 3. Dans la méthode « creer_liste() » remplacer la ligne : « # ajouter un print pour afficher la structure des données » par l'affichage demandé.

Q 4. Même question pour la méthode « ajouter() ».

Q 5. Combien d'opérations élémentaires sont nécessaires pour ajouter à un élément de la liste?

Q 6. Coder la corps de la méthode « afficher() ».

Q 7. Rajouter une méthode qui donne l'élément d'un rang correspondant : « Lire(Liste, ran) -> element ».

Q 8. Rajouter une méthode qui donne la place disponible dans la liste : place_liste(liste) -> int.

Q 9. Combien d'opérations élémentaires sont nécessaires pour accéder à un élément de la liste quel que soit son rang ?

Q 10. Quel avantage possède ce choix d'implantation à la complexité en temps ?

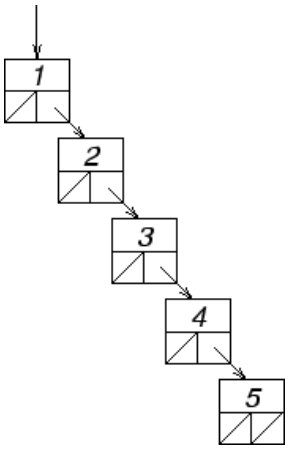
Q 11. Quel inconvenient peut avoir ce type d'implantation ? Peut-on, par exemple, ajouter plus d'éléments que le nombre initialement prévu ?

Exercice 3 : liste d'embarquement d'un Avion

Créer un TAD liste qui gère la liste d'embarquement d'un Avion.
Il devra être testé unitairement au travers « d'assert » (utilisation de if __name__== main) avant d'être utilisé dans dans un programme comme un package.
Les passagers seront répérés par leur nom et non par un numéro.

3. Listes taille dynamique (listes chainées)

Les listes de tailles dynamiques seront traitées après les arbres binaires et les objets.
Elles seront implantées en adaptant un arbre binaire comme un arbre dégénéré : un seul fil droit par nœud comme le montre la figure ci-contre :
Cela fera le sujet d'une activité qui s'appuie fortement sur le cours et les exemples.



IV. Les « listes python », le « couteau suisse de python »

Dans python il existe une classe « list » qui est native.
Ce n'est pas à proprement parlé une liste au sens algorithmique.
En effet ce composant logiciel peut être vu sous les angles d'une liste (avec les méthodes append(), remove()) mais il offre plus de méthodes que celles réservées aux listes algorithmiques. De plus il ne respecte pas les propriétés des listes algotrithmiques.

Toutefois sa mise en œuvre est simple et pratique. Elle pourra donc être utilisée en projet et dans d'autres activités de cours au vue de l'ensemble des fonctionnalités qu'elle intègre.

Soit le code python qui utilise cette classe :

```
liste = [14,7,8,9]
print (liste)
liste.sort()
print(liste)
liste.append("bonjour")
print(liste)
```

Q 1. Dans l'exemple ci-dessus pourquoi la méthode sort() ne peut être utilisée pour une liste.
Q 2. Quelle autre propriété n'est pas vérifiée pour les listes par la classe list de python ?