Exploration

Les collections en java

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision

Sommaire

Nomenclature :

- **Assimiler** : cours pur. Explication théorique et détaillée (globalement supérieur à 4 pages).
- **Décoder** : fiche de cours, généralement inférieure à 5 pages.
- **Découvrir** : Travaux dirigés. Faisable sans matériel.
- **Explorer** : Travaux pratiques. Nécessite du matériel ou des logiciels.
- **Mission** : Projet encadré ou partie d'un projet.
- **Voyager** : Projet en autonomie totale. Environnement ouvert : Vous êtes le capitaine !

A Introduction

Entre une base de données et un tableau (taille fixe) il existe le concept de collection.

L'objectif de ce cours est simplement de comprendre la différence entre un tableau et une collection, puis de tester une implémentation en Java, sur un exemple de gestion d'IMC.

A.1 Prérequis

aucun

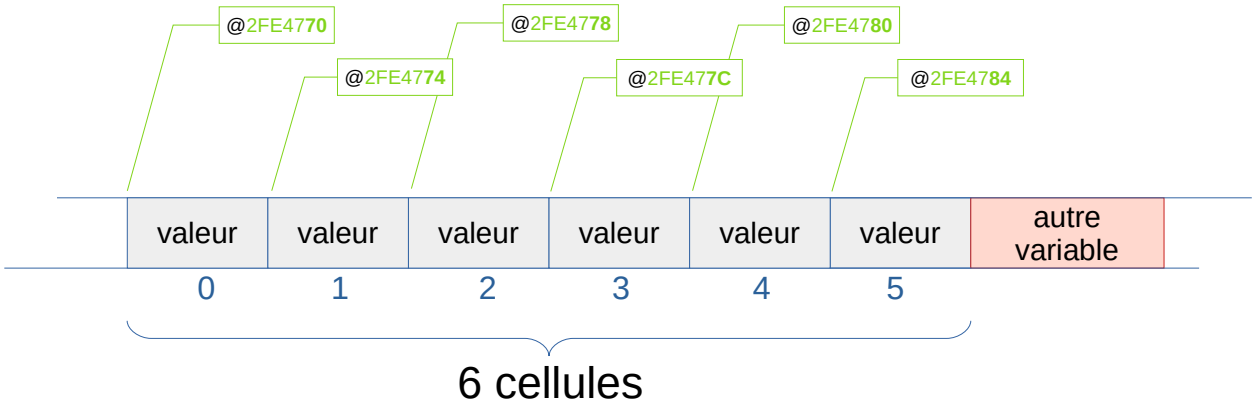
B Différence entre une collection et un tableau

Bien que les deux puisse fonctionner à partir d'un index pour atteindre une cellule en particulier, les deux sont différents et ne fonctionnent pas de la même manière.

- Dans un tableau, les cellules sont contiguës et de taille fixe
- Dans une collection, la recherche est plus longue, car il faut trouver la cellule dans la mémoire

B.1 Tableau

Voici la représentation d'un tableau d'entier (4 octets) dans la mémoire d'un ordinateur :



B.1.1 Explication

On constate que l'adresse de base du tableau dans la mémoire correspond à l'adresse de la première cellule. D'autre part, on comprend que pour accéder au contenu de la troisième cellule, il faut ajouter 3x4 octets à cette adresse de base.

Par exemple : si l'adresse de base est @2FE4770 et que j'ajoute 12 octets (soit C en hexadécimal), j'obtiens @2FE477C

Ce principe permet à l'ordinateur d'atteindre très rapidement n'importe quelle cellule du tableau. L'inconvénient est que le tableau ne peut pas être étendu à volonté, surtout si d'autres variables sont présentes en mémoire, après le tableau.

La taille des données est importante : un booléen ne devrait tenir qu'un bit (1/8^{e} d'octet), un double prendra 8 octets, un caractère occupera un seul octet, etc.

⚠ Une exception pour les chaînes de caractères, qui sont gérées différemment (une zone spécifique leur est attribuée, un tableau de String ne contiendra alors que les adresses de chaque zone).

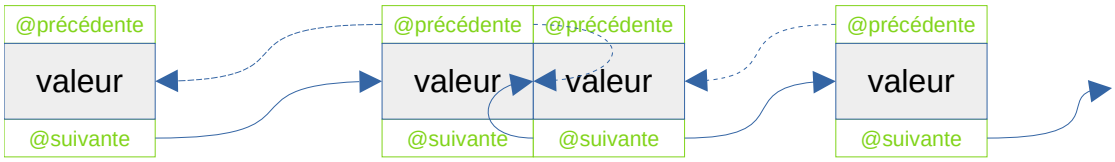
B.1.2 Utilisation

Voici un exemple de code utilisant un tableau.

```
class TestTableau
{
  public static void main(String[] args)
  {
      // Tableau de quatre caractères
      char[] tab = new char[4];

      tab[0]='a';
      tab[1]='b';
      tab[2]='c';
      tab[3]='d';

      // accéder aux éléments du tableau
      for(int i=0; i < tab.length; i++)
      {
          System.out.println(tab[i]);
      }
   }
}
```

B.2 Collection ou liste

Une collection (aussi appelé liste) fonctionne différemment ; Le nom qui évoque le principe est "liste chaînée", car les valeurs sont stockées dans un objet contenant deux adresses et la valeur, comme le schéma suivant :



B.2.1 Explication

Ainsi, pour stocker un entier sur 4 octets, une cellule occupera 8 octets pour l'adresse suivante, 8 octets pour l'adresse précédente et enfin 4 octets pour l'entier lui-même, soit 20 octets.

La perte de place est élevée et il y a un autre inconvénient : pour atteindre la troisième cellule, il faut lire toutes les cellules précédentes, ce qui peut prendre beaucoup de temps si la liste est très grande ;

Le gain est en revanche de pouvoir étendre la taille de la liste, sans se soucier de l'emplacement en mémoire.

B.2.2 Utilisation

Voici un exemple d'utilisation d'une liste :

```
import java.util.ArrayList;

class TestArrayList
{
   public static void main(String[] args)
   {
       // Création d'une ArrayList de taille dynamique.
       ArrayList<string> arr = new ArrayList<string>();

       arr.add("Ma première chaîne de caractères");
       arr.add("Ma deuxième chaîne de caractères");
       arr.add("Ma troisième chaîne de caractères");

       // Itérateur pour itérer entre les éléments de ArrayList
       Iterator it = arr.iterator();

       // accéder aux éléments du tableau en utilisant l'itérateur
       while(it.hasNext())
             System.out.println(it.next());
   }
}
```

L'exemple ci-dessus utilise un itérateur automatique pour utiliser la liste, mais il est possible d'accéder aux éléments avec une boucle for classique :

```
import java.util.ArrayList;

public class TestArrayList {
    public static void main(String[] args) {
        // Création d'une ArrayList de taille dynamique.
        ArrayList<String> arr = new ArrayList<String>();

        arr.add("Ma première chaîne de caractères");
        arr.add("Ma deuxième chaîne de caractères");
        arr.add("Ma troisième chaîne de caractères");

        // Itérer sur les éléments de l'ArrayList avec une boucle for
        for (int i = 0; i < arr.size(); i++) {
            System.out.println(arr.get(i));
        }
    }
}
```

Ainsi, le comportement d'une collection est souvent considéré comme plus souple et est donc plus utilisé que les tableaux.

B.3 Liste ou tableau

Le choix est donc une question d'utilité : dans un jeu vidéo, un tableau sera plus efficace s'il est souvent accédé (plusieurs fois par seconde), en revanche une liste sera préférée pour la gestion d'un nombre d'élément indéterminé.

C Exploration d'une collection d'IMC

Nous allons maintenant créer un projet simple, permettant de gérer une collection d'objets.

C.1 Création du projet GestionIMC

Pour cela, nous allons utiliser une classe IMC, dans un nouveau projet (GestionIMC) à Java/IntelliJ :

IMC.java

```
class IMC {
    private int id;
    private String nom;
    private double poids;
    private double taille;
    private double imc;

    public IMC(int id, String nom, double poids, double taille) {
        this.id = id;
        this.nom = nom;
        this.poids = poids;
        this.taille = taille;
        this.imc = calculerIMC();
    }

    private double calculerIMC() {
        return poids / (taille * taille);
    }

    public int getId() {
        return id;
    }

    public String getNom() {
        return nom;
    }

    public double getPoids() {
        return poids;
    }

    public double getTaille() {
        return taille;
    }

    public double getIMC() {
        return imc;
    }
}
```

Vous pouvez ajouter les mutateurs (setTaille(double taille), setPoids(double poids) et setNom(String nom)) si vous le souhaitez, cela peut servir par la suite et est un bon entraînement.

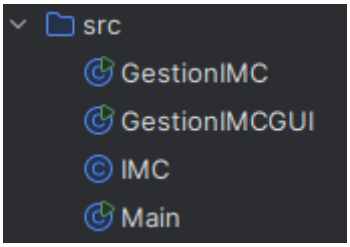
Nous créons ensuite la classe GestionIMC, le code suivant :

gestionIMC.java

```
import java.util.ArrayList;

public class GestionIMC {
    private static ArrayList<IMC> listeIMC = new ArrayList<>();

    public static void ajouterEnregistrement(int id, String nom, double poids, double taille) {
        IMC enregistrement = new IMC(id, nom, poids, taille);
        listeIMC.add(enregistrement);
    }

    public static void afficherCollection() {
        System.out.println("Collection d'enregistrements d'IMC :");
        for (IMC enregistrement : listeIMC) {
            System.out.println("ID : " + enregistrement.getId());
            System.out.println("Nom : " + enregistrement.getNom());
            System.out.println("Poids : " + enregistrement.getPoids() + " kg");
            System.out.println("Taille : " + enregistrement.getTaille() + " m");
            System.out.println("IMC : " + enregistrement.getIMC());
            System.out.println("------------");
        }
    }

    public static void main(String[] args) {
        ajouterEnregistrement(1, "Chuck Norris", 95.0, 1.82);
        ajouterEnregistrement(2, "Sylvester Stallone", 85.0, 1.77);
        ajouterEnregistrement(3, "Arnold Schwarzenegger", 110.0, 1.88);
        afficherCollection();
    }
}
```

Ce code permet d'utiliser une interface en mode console.

Note : Nous ne sommes pas obligés d'utiliser la classe Main prédéfinie, l'important étant de définir les classes dans le répertoire source (c'est une exploration, nous n'utilisons pas le modèle MVC).



C.2 Interface graphique

Java sait utiliser des fenêtres et des boutons et l'ensemble des champs de formulaires, que l'on trouve avec HTML ou d'autres langages (Python, C#, C++, etc.)

En reprenant le fichier de la classe IMC (IMC.java), voici un exemple sur un projet GestionIMCGUI :

GestionIMCGUI.java

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

public class GestionIMCGUI extends JFrame {
    private static ArrayList<IMC> listeIMC = new ArrayList<>();
    private JList<String> listeAffichage;
    private DefaultListModel<String> modeleList;

    public GestionIMCGUI() {
        setTitle("Gestion des enregistrements d'IMC");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(500, 400);
        setLocationRelativeTo(null);

        // Création du modèle de liste
        modeleList = new DefaultListModel<>();
        listeAffichage = new JList<>(modeleList);

        // Ajout de la liste dans un panneau
        JPanel panneauPrincipal = new JPanel(new BorderLayout());
        panneauPrincipal.add(new JScrollPane(listeAffichage), BorderLayout.CENTER);

        // Ajout du panneau principal à la fenêtre
        getContentPane().add(panneauPrincipal);
    }

    public static void ajouterEnregistrement(int id, String nom, double poids, double taille) {
        IMC enregistrement = new IMC(id, nom, poids, taille);
        listeIMC.add(enregistrement);
    }

    public void afficherCollection() {
        modeleList.clear();
        for (IMC enregistrement : listeIMC) {
            modeleList.addElement("ID : " + enregistrement.getId() + ", Nom : " +
enregistrement.getNom() + ", IMC : " + enregistrement.getIMC());
        }
    }

    public static void main(String[] args) {
        ajouterEnregistrement(1, "Chuck Norris", 95.0, 1.82);
        ajouterEnregistrement(2, "Sylvester Stallone", 85.0, 1.77);
        ajouterEnregistrement(3, "Arnold Schwarzenegger", 110.0, 1.88);

        GestionIMCGUI fenetre = new GestionIMCGUI();
        fenetre.afficherCollection();
        fenetre.setVisible(true);
    }
}
```

D Travail à faire

Maintenant que vous savez gérer une liste, voici une liste de tâches à faire (sur la version initiale du programme, en mode console).

D.1 Suppression

La suppression d'un élément dans une liste se fait avec l'instruction `.remove(int index)`.

Voici un exemple de code :

```
    public static void supprimerEnregistrement(int id) {
        for (int i = 0; i < maListe.size(); i++) {
            if (maListe .get(i).getId() == id) {
                maListe.remove(i);
                break;
            }
        }
    }
```

D.2 Suppression par nom

Essayez de créer une méthode de suppression par nom `supprimerEnregistrementNom(String nom)`

D.3 Insérer un enregistrement

Essayez d'insérer des enregistrements au début de la liste ou à une position quelconque de la liste, sachant que la méthode est la suivante : `maListe.add(indexAvant, enregistrement)`.

Insérez Bruce Lee ("Bruce Lee", 70.0 kg, 1.72 m) entre Chuck Norris et Sylverster Stallone.

D.4 Modifier un enregistrement par le nom

Enfin, essayez de retrouver un enregistrement par le nom et changer les valeurs