PROGRAMMATION ORIENTÉE OBJETS

LISTES JAVA

ET LEUR IMPLÉMENTATION ARRAYLIST

LISTES

- Les listes sont une implémentation plus propre et plus fonctionnelle des tableaux
- Nous verrons le type de liste (largement) le plus utilisé en Java : `ArrayList`
  - déclaration
  - initialisation
  - méthodes usuelles de manipulation

LES PROBLÈMES DU TABLEAU EN JAVA

- Rappel : un tableau est une collection ordonnée d'objets du même type

```
Employe[] employes = new Employe[2];
employes[0] = new Employe("Laure", "Loge");
employes[1] = new Employe("Maude", "Zarella");
System.out.println("1. " + employes[0].getPrenom() + " " + employes[0].getNom());
System.out.println("2. " + employes[1].getPrenom() + " " + employes[1].getNom());
```

PROBLÈMES DES TABLEAUX EN JAVA

- Taille fixée à l'instanciation
- Insertions/suppressions compliquées et coûteuses
- Tableau plein ? ⇒ Tout recopier dans un tableau plus grand
- Pas de méthodes pour les traitements usuels (insertion, suppression, recherche…)

ARRAYLIST RÉSOUD TOUS CES PROBLÈMES

- Un tableau Java est **encapsulé** dans la classe `ArrayList`
- Pas de taille fixe
- Taille augmente automatiquement si trop petit
- Méthodes publiques fournies par la classe permettent de facilement ajouter, insérer, supprimer, etc.
- Ces méthodes s'occupent elles-mêmes de gérer le tableau sous-jacent, *mais le client ne sait pas (et ne veut pas savoir) comment ça fonctionne*

DÉCLARATION ET INITIALISATION

- `ArrayList` est une classe **générique**
  - on déclare un objet `ArrayList` (comme d'habitude)
  - on l'initialise en instanciant un nouvel objet (comme d'habitude)
  - **on doit spécifier le type des éléments entre chevrons <> (généricité)**

ARRAYLIST : EXEMPLE

```
ArrayList<Employe> employes;
employes = new ArrayList<>();

// ou bien : déclaration + initialisation
ArrayList<Employe> employes = new ArrayList<>();
```

CONTRAINTE SUR LE TYPE DES ÉLÉMENTS

- Le type générique est **obligatoirement un type objet**
  - les types primitifs (`int`, `double`, `boolean`…) sont interdits
- Pas vraiment embêtant, puisqu'on peut utiliser les types *wrappers* à la place : `Integer`, `Double`, `Boolean`…
  - `ArrayList<Integer> mesEntiers = new ArrayList<>();`

UTILISATION

- Là où les tableaux utilisent une syntaxe définie spécialement pour eux en Java, les `ArrayList` utilisent des méthodes classiques
- `add(elt)` : ajouter *elt* en fin de liste
- `get(index)` : récupérer l'élément à la position *index*
- `set(index, elt)` : insérer *elt* à la position *index*

UTILISATION (SUITE)

- `remove(elt)` : supprime *elt* de la liste
- `remove(index)` : supprime l'élément à la position *index*
- `clear()` : vide la liste
- `isEmpty()` : vérifie si la liste est vide ou non
- `indexOf(elt)` : renvoie la position de *elt* dans la liste
- `contains(elt)` : vérifie si *elt* est dans la liste ou non
- `size()` : renvoie la taille de la liste

DOCUMENTATION

- Dans la documentation officielle, le type générique des éléments de la liste est noté `E` (`ArrayList<E>`)
  - cela veut dire qu'à chaque fois que l'on voit `E` dans la page, on peut remplacer mentalement par le type des éléments réels dans notre propre liste (ex: le type `Employe`)
  - `boolean add(E e)`: la méthode prend en paramètre un objet noté *e* du type des éléments `E` de la liste (ex: un objet de type `Employe`)
  - `E get(int index)`: la méthode renvoie un objet du type des éléments de la liste

```java
public static void main(String[] args) {
  // déclaration + initialisation
  ArrayList<Double> valeurs = new ArrayList<>();
  // affichage taille
  System.out.println("Taille : " + valeurs.size());
  // ajouts d'éléments
  valeurs.add(1.23);
  valeurs.add(2.34);
  valeurs.add(3.45);
  System.out.println("Taille : " + valeurs.size());
  // recherche de position
  int position = valeurs.indexOf(2.34);
  System.out.println("2.34 trouvé à la position : " + position);
  // appel de méthode qui doit se trouver dans la même classe
  afficherListe(valeurs);
  valeurs.add(1, 5.67);
  afficherListe(valeurs);
  valeurs.remove(0);
  afficherListe2(valeurs);
}
```

```java
// Utilisation d'une boucle indexée pour le parcours
public static void afficherListe(ArrayList<Double> liste) {
  System.out.println("_____________________________");

  for (int i = 0; i < liste.size(); i++) {
    System.out.println("index " + i + " : " + liste.get(i));
  }

  System.out.println("_____________________________");
}
```

```java
// Utilisation d'un "foreach" Java pour le parcours
public static void afficherListe2(ArrayList<Double> liste) {
  System.out.println("_____________________________");

  for (double element : liste) {
    // note : on n'a plus l'index
    System.out.println(element);
  }

  System.out.println("_____________________________");
}
```

EXERCICE - GESTION D'EMPLOYÉS

- Écrire et tester une classe `GestionEmployes` capable de:
  - embaucher/virer des employés
  - afficher la liste des employés actuels, triés par nom
  - afficher la liste des employés actuels, triés par date d'embauche
  - augmenter le salaire d'un employé (d'un pourcentage donné)
  - augmenter le salaire de tous les employés (d'un pourcentage donné)
  - rechercher des salariés par nom
- Implémentation sur console (avec un menu), **puis** JavaFX