

Les listes en Java

Les types de données qu'on a utilisés jusqu'à présent permettent de stocker une seule valeur (un booléen, un entier, un réel, une chaîne de caractères, un objet). Souvent cependant il est nécessaire de regrouper des objets qui ont la même sémantique.

Exemple

Développer une classe modèle `StudentPicker` qui permet de gérer des noms d'élèves et d'en choisir un au hasard. La classe possède les méthodes suivantes :

- une méthode `addName` qui permet d'ajouter un nom d'élève
- une méthode `removeName` qui permet de supprimer un nom dont on n'a plus besoin
- une méthode `getRandomName` qui permet de choisir aléatoirement un des noms

Avec les types de données qu'on a vus jusqu'à présent il est impossible d'écrire la classe `StudentPicker`.

- Il faut mémoriser les noms ajoutés à l'aide de la méthode `addName`.
- Si on voulait utiliser les types de données qu'on connaît, il faudrait ajouter un nouvel attribut pour chaque nom.
- Il faut déclarer ces attributs au moment où on écrit la classe.
- A cet instant on ne sait pas combien de noms l'utilisateur va ajouter.
- Par conséquent on ne sait pas combien d'attributs il faut déclarer.

Solution sans ordinateur

La solution consiste à créer un seul attribut qui contient une liste de tous les noms disponibles. Le principal avantage est que le nombre de noms contenus dans la liste peut varier lors de l'exécution du programme.

Par la suite un type de données qui nous permet de stocker une telle liste en Java sera présenté.

La classe ArrayList

En Java il existe plusieurs classes prédéfinies qui permettent de regrouper des **objets**. On appelle ces classes des collections. Dans ce cours on va se limiter à la classe `ArrayList`.

Les paquets

Afin d'utiliser des collections du type `ArrayList`, il nous faut absolument inclure l'instruction suivante dans la première ligne du fichier de notre classe.

```
import java.util.ArrayList;
```

Java utilise le concept de **paquets** pour regrouper des fonctionnalités qui vont ensemble. Par défaut Java n'importe que les fonctionnalités de base requises dans presque tous les programmes. La raison est que toute fonctionnalité importée est chargée en mémoire vive (RAM) au moment du lancement du programme. Charger des fonctionnalités pas nécessairement requises par le programme ne gaspillerait donc que de la mémoire. C'est pourquoi pour utiliser toute fonctionnalité non importée par défaut il faut d'abord importer le paquet correspondant. Au cas des `ArrayList` le paquet à importer s'appelle `java.util.ArrayList`.

Déclaration et initialisation d'une ArrayList

Pour créer un attribut du type `ArrayList` et l'initialiser avec un nouvel objet il faut utiliser la syntaxe suivante :

```
private ArrayList<String> alNames = new ArrayList<>();
```

De même que pour les autres déclarations il faut **indiquer le type de l'attribut**. Pour une liste il faut cependant aussi indiquer **quel type d'objets elle regroupe**. Dans cet exemple la liste contient plusieurs **chaînes de caractères**. Il faut toujours indiquer le type des objets regroupés par la liste entre `<` et `>`. **Le nom de l'attribut peut être choisi librement**. Cependant dans ce cours on va toujours choisir un nom préfixé par *al*. Ceci nous permet d'identifier plus rapidement les listes. Derrière **le signe d'affectation** il faut faire un appel au **constructeur**. Dans cet appel il ne faut surtout pas oublier les chevrons `<>`. On peut indiquer une deuxième fois **le type des objets stockés dans la liste**...

```
private ArrayList<String> alNames = new ArrayList<String>();
```

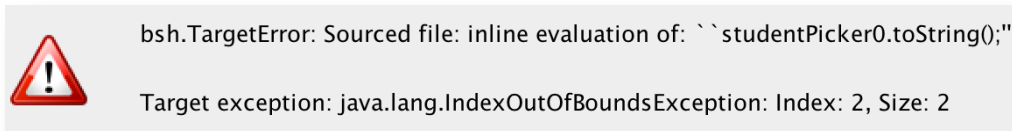
...mais ce n'est pas obligatoire avec les versions plus récentes de Java.

Voici un aperçu des méthodes publiques les plus importantes de la classe `ArrayList` :

Méthodes publiques	Description
<code>boolean add(Object pObj)</code>	Ajouter un nouvel objet à la liste. La valeur retournée est toujours <code>true</code> , mais on ne l'utilisera jamais.
<code>Object get (int pIndex)</code>	Retourner l'objet à l'index spécifié en paramètre.
<code>Object set(int pIndex, Object pObj)</code>	Remplacer l'objet à l'index spécifié <code>pIndex</code> en paramètre par <code>pObj</code> . La méthode retourne l'objet qui a été remplacé.
<code>Object remove(int pIndex)</code>	Enlever l'objet à l'index <code>pIndex</code> La méthode retourne l'objet qui vient d'être enlevé.
<code>void clear()</code>	Enlever tous les objets de la liste
<code>int size()</code>	Retourner le nombre d'objets qui se trouvent dans la liste
<code>boolean isEmpty()</code>	Vérifier si la liste est vide. La méthode retourne <code>true</code> si la liste est vide, <code>false</code> sinon.

L'index d'une liste

L'index indique la position d'un objet dans la liste. Il faut faire attention à toujours utiliser un index valable. Si on essaie d'accéder ou de modifier une page qui n'existe pas, on obtient un message d'erreur (`IndexOutOfBoundsException`) comme dans l'image suivante :



La deuxième ligne indique clairement le problème. On essaie de tourner 2 pages pour accéder au contenu de la troisième page (Index: 2) alors qu'il n'y a pas de troisième page. Il n'y en a que deux (Size: 2).



Le premier index d'une liste non vide vaut 0 tandis que le dernier index est égal à `size() - 1`. L'index est donc toujours strictement plus petit que la valeur retournée par `size()`.



```
public String toString(){
    String names = "";

    for (int i = 0; i < alNames.size(); i++){
        names = names + alNames.get(i) + " ";
    }

    return names;
}
```

Comme illustré par le code ci-dessus, on peut afficher facilement tout le contenu d'une liste à l'aide d'une boucle. Cependant il faut surtout faire attention à utiliser l'opérateur `<` dans la condition `i < alNames.size()`.

`i <= alNames.size()` produirait le message d'erreur d'en haut. Il s'agit là d'une faute très fréquente.