R2.01 : cours n°4

I.Borne

- Structure de données : ArrayList
- Parcours d'une collection d'objets
  - *for each*
  - *Iterator*
- Retour sur les packages

Introduction

- Pour traiter une grande quantité de données il est nécessaire de rassembler les valeurs dans une structure de données.
- Les structures de données les plus couramment utilisées en Java sont les tableaux et les *ArrayList*.
- Un tableau a une taille fixée à sa création.
- Les tableaux sont très pratiques pour les valeurs numériques.
- Il existe différentes façons d'initialiser un tableau

```
int[] primes = {2,3,5,7,11} ;
```


Framework des collections : implémentation

General-purpose Implementations

Interfaces	Hash table Implementations	Resizable array Implementations	Tree Implementations	Linked list Implementations	Hash table + Linked list Implementations
Set	HashSet		TreeSet		LinkedHashSet
List		ArrayList		LinkedList	
Queue					
Deque		ArrayDeque		LinkedList	
Map	HashMap		TreeMap		LinkedHashMap

Toutes les collections sont dans java.util

Cette année nous étudierons :

- `ArrayList`
- `HashMap`

Une première collection : java.util.ArrayList

- La classe **`ArrayList`** est une des collections du *framework des collections* de java , qui se trouve dans le package util
- Cette classe permet de gérer des collections d'**objets**, un peu à la manière d'un tableau.

- Un `ArrayList` offre deux avantages :
  - Une taille variable avec une gestion transparente ;
  - De nombreuses méthodes pour des tâches habituelles.

- Attention un `ArrayList` ne peut contenir que des **objets**, pas de types primitifs.

La classe `java.util.ArrayList`

- Chaque instance d'un **`ArrayList`** possède une capacité (taille du tableau utilisé pour stocker les éléments dans la liste)
- La capacité par défaut est 10, et augmente automatiquement au fur et à mesure.
- La classe **`ArrayList`** possède beaucoup de méthodes avec des variantes.
- Une instance d'**`ArrayList`** est un objet à qui on envoie des messages.

Les collections sont des classes génériques

- La classe `ArrayList` est une classe générique :
  - `ArrayList<E>` recueille des objets de type E
- Le type est précisé à la déclaration et à la création

Pour en savoir plus :
https://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html.

```
public class ArrayList<E> extends AbstractList<E>
   implements List<E>, RandomAccess, Cloneable, Serializable
```


Déclaration et construction d'un `ArrayList`

- Nous utiliserons des collections typées.
- Pensez à importer la classe pour plus de lisibilité :

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

- Déclaration d'un attribut de type `ArrayList` dans une classe :

```
private ArrayList<type_des_éléments> maListe;
```

- Création d'un objet `ArrayList` (dans le constructeur)

```
maListe= new ArrayList<type_des_éléments>();
```

- Une capacité supérieure à 10 peut être précisée à la création si on doit gérer un très grand nombre d'éléments, sinon c'est inutile :

```
maGrandeListe= new ArrayList<type_des_éléments>(2000);
```


Exemple

```
import java.util.ArrayList;

public class ArrayListExample {
    // déclaration de la liste
    private ArrayList<String> myList;

    // création de la liste
    public ArrayListExample() {
      this.myList = new ArrayList<String>();
    }
}
```

Respectez ce principe : déclaration puis création dans le constructeur

Une première comparaison avec les tableaux

```
ArrayList<Etudiant> liste;
liste = new ArrayList<Etudiant>() ;

liste.add(new Etudiant("Max" , 3)) ;
liste.add(new Etudiant("Tom" , 5)) ;
liste.add(new Etudiant("Jim" , 7)) ;

System.out.println(liste.size()) ;
```

```
Etudiant[] liste ;
liste = new Etudiant[10] ;

liste[0]= new Etudiant("Max" , 3);
liste[1] = new Etudiant("Tom" , 5);
liste[2]= new Etudiant("Jim" , 7);

System.out.println(liste.length) ;
```


Ajout et retrait d'éléments dans un ArrayList

- Ajout en fin de liste
  - **`public boolean add(E e)`**
  - retourne true (on utilise rarement ce booléen qui est toujours égal à true)
- Insérer un élément à une position précise
  - **`public void add(int index, E element)`**
- Retirer la première occurrence d'un élément
  - **`public boolean remove(E e)`**
  - retourne true si la liste contenait l'élément spécifié
- Retirer un élément à une position précise
  - **`public E remove(int index)`**
  - retourne l'élément qui a été retiré

Accès aux éléments dans un ArrayList

- Accès à un élément à une position donnée :
  - **`public E get(int index)`**
  - retourne l'élément à la position spécifiée
- Remplacer un élément à une position donnée :
  - **`public E set(int index, E element)`**
  - remplace l'élément à la position spécifiée avec l'élément passé en paramètre
- Autres méthodes pratiques :
  - **`public int size()`**
  - **`public boolean contains(Object o)`**
  - **`public int indexOf(Object o)`**
  - **`public int lastIndexOf(Objet o)`**
  - **`public boolean isEmpty()`**

Obtenir le nombre d'éléments dans les structures de données

Data type	Nombre d'éléments
Tableau	a.length
ArrayList	a.size()
String	a.length()

Une forme différente pour la boucle Loop

- La boucle appelée *for each* parcoure tous les éléments d'une collection ou d'un tableau.
- Elle n'utilise pas d'indice comme la forme classique.
- La syntaxe :

```
for (Type variable : collection)
```

L'objectif est d'exécuter une boucle pour chacun des éléments de la collection. A chaque itération, la variable est affectée au prochain élément de la collection. Ensuite le bloc de message est exécuté.

Suite de la comparaison avec les tableaux

```
ArrayList<Etudiant> liste;
liste = new ArrayList<Etudiant>() ;
liste.add(new Etudiant("Max" , 3)) ;
liste.add(new Etudiant("Tom" , 5)) ;
liste.add(new Etudiant("Jim" , 7)) ;
Etudiant etud = liste.get(1) ;

System.out.println("taille de la liste :"
+ liste.size()) ;

for (Etudiant e : liste) {
        System.out.println(e) ;
}
```

```
Etudiant[] liste ;
liste = new Etudiant[10] ;
liste[0]= new Etudiant("Max" , 3);
liste[1] = new Etudiant("Tom" , 5);
liste[2]= new Etudiant("Jim" , 7);
Etudiant etud = liste[1] ;

System.out.println("taille de la liste :"
+ liste.length) ;

for (Etudiant e : liste) {
    System.out.println(e) ;
}
```


Exemple d'une classe Promotion avec un ArrayList

- Il faut penser à
  - importer la classe
  - déclarer l'ArrayList
  - Créer l'objet l'ArrayList (dans le constructeur)

```
import java.util.ArrayList;
public class Promotion {
  private ArrayList<Etudiant>listeEtudiants;
  private String nom;

  public Promotion (String nom) {
    if (nom!=null){
       this.nom = nom;
       this.listeEtudiants = new ArrayList<Etudiant>() ;
    }
    else {System.out.println("Promotion: name ne peut pas
être null");}
  } ...
```


(suite) la méthode moyenne

```
public void add(Etudiant e) {
       this.listeEtudiants.add(e) ;
}
public double moyenne() {
    double moyenne = 0;
    double total = 0;
    if (!listeEtudiants.isEmpty()){
      for (Etudiant e : listeEtudiants) {
           total = total + e.moyenne();
      }
      moyenne = total/listeEtudiants.size();
    }
    else {
     System.out.println("moyenne : listeEtudiants est vide");}
    return moyenne;
}
```

Boucle et itération sur un ArrayList

- Pour parcourir un `ArrayList` pour une recherche on utilise généralement un itérateur.
- On est dans le cas où l'on veut pouvoir sortir d'une boucle avant la fin.
- Un itérateur permet d'énumérer une collection et s'utilise avec une boucle *while*.
- Un `ArrayList` possède une méthode qui retourne un itérateur pour ses éléments :
  - `public Iterator<E> iterator()`
- Exemple :

```
Iterator<Etudiant> it= listeEtudiants.iterator();
```


La manipulation d'un itérateur

- Il s'agit de l'interface :

`java.util.Iterator`

- Le type des éléments est aussi précisé à la création de l'itérateur.
- Les deux méthodes principales :
  - `E next()` : retourne le prochain élément de l'itération
  - `boolean hasNext()` : retourne vrai s'il reste des éléments
- Il existe également une méthode qui permet d'enlever de la collection le dernier élément retourné par l'itérateur :

```
void remove()  // à utiliser avec précaution
```


Exemple : recherche d'un étudiant par son nom dans une promotion

```
public Etudiant getEtudiant(String nom) {
    Etudiant resultat=null;
    boolean trouve= false;
    if (!listeEtudiants.isEmpty()){
      Iterator<Etudiant> it = listeEtudiants.iterator();
      while ((it.hasNext()) && (!trouve)) {
          Etudiant  e = it.next();
          if (e.getNom().equals(nom)){
              trouve =true;
              resultat =e;
          }
      }
    }

    return resultat;
  }
```


Synthèse sur l'utilisation des boucles

Parcours d'une collection ou un tableau d'objets

- du premier élément jusqu'au dernier : utilisez une boucle *for* classique ou la forme *for each*
- du dernier au premier : utiliser la boucle *for* classique
- Depuis un indice quelconque jusqu'au dernier : utiliser la boucle *for* classique

Synthèse sur l'utilisation des boucles(2)

Recherche d'un élément dans un tableau

- Le parcours du tableau doit s'arrêter si on a trouvé l'élément
  - Utiliser une boucle ***while (obligatoirement)***

Recherche d'un élément dans un ArrayList

- Pour pouvoir interrompre le parcours d'un ArrayList quand une condition est satisfaite :
  - Utiliser une boucle ***while*** avec un itérateur

Retour sur les packages

La nécessité des packages

- Un programme Java est constitué par une collection de classes.
- Les classes correspondent à des fichiers.
- Quand les programmes sont plus gros, le nombre de classes augmente.
- Un mécanisme de structuration supplémentaire est nécessaire.
- Java propose les packages qui permettent de regrouper des ensembles de classes qui collaborent.

Toutes les classes sont dans un package

- Toutes les classes de l'API de Java sont organisées en packages :
  - **`java.lang`** : classes fondamentales du langage.
  - **`java.util`** : classes utilitaires et collections,
  - **`java.io`** : classes gérant les entrées/sorties
  - **`java.awt`** : classes pour les interfaces graphiques
  - etc..
- Il existe un package spécial « default package » qui n'a pas de nom et qui contient les classes qui ne précisent pas de nom de package.

Package = espace de noms

- Deux classes Java peuvent avoir le même nom si elles appartiennent à des packages différents :

  `java.util.List`

  `java.awt.List`

Syntaxe : *nomPackage1.nomPackage2. ... .NomClasse*

Pour mettre une classe dans un package, il faut insérer comme première instruction dans le fichier source :

`package` *`nomPackage;`*

- Les noms de packages commencent par une minuscule.

Exemple

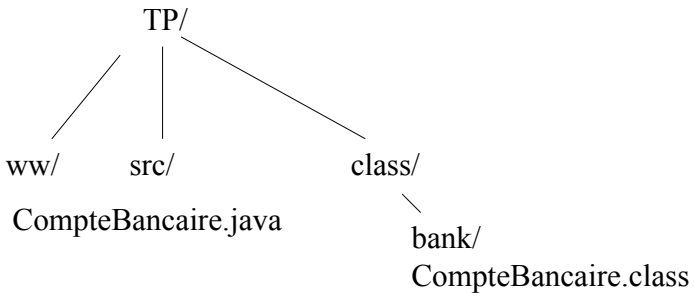
```
package bank;
/**
  La classe CompteBancaire modelise un compte possedant
  un numero et un solde.
*/
public class CompteBancaire {

    //partie prive : variables d'instance
    private int numero;
    private double solde;
. . .
}
```


Organisation de mes propres packages

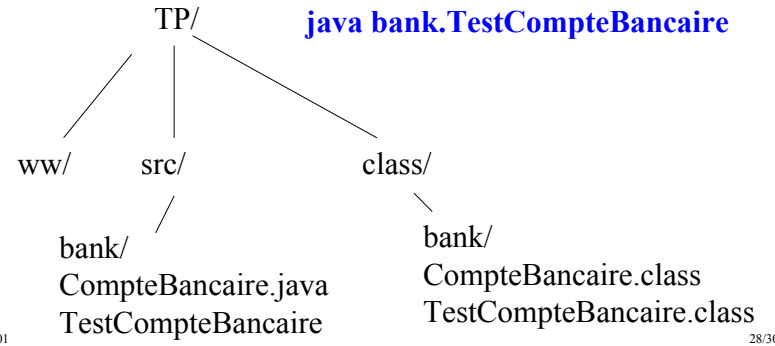
- Le nom du package servira de nom de sous-répertoire pour les classes compilées :

```
~/TP/ww/javac -d ../class ../src/CompteBancaire.java
```



Organisation des sources avec les packages

Pour qu'une classe se trouvant dans un package soit utilisable il est nécessaire que son chemin d'accès soit conforme au nom du package.



Importer des packages

Règle 1 : sans précision d'appartenance à un package, une classe appartient au package « default »

Règle 2 : toutes les classes appartenant à un même package se connaissent.

- Conséquence 1 : à l'intérieur d'un même package l'importation n'est pas nécessaire.
- Conséquence 2 : si une classe fait appel à une classe d'un autre package, **il faut l'importer**, exception faite pour les classes du paquetage *java.lang*.

Package + import

```
package bank;
import java.util.ArrayList ;
public class Bank {

    //partie prive : variables d'instance
    private String nom;
    private ArrayList<BankAccount> accounts;
. . .
}
```

Si je ne mets pas l'import que se passe-t-il ?

Comment faire sans import ?