Le Langage Java

1ère année

M. Bastreghi J. Beleho P. Bettens M. Codutti
A. Hallal C. Leruste D. Nabet N. Pettiaux A. Rousseau

Haute École de Bruxelles — École Supérieure d'Informatique

Année académique 2011 / 2012

Leçon 3

Les listes

- La machine virtuelle
- Les outils de développement
- Analyse lexicale
- Les espaces
- Les commentaires
- Les tokens
- Motivation
- La Javadoc
- Les tags
- Le code HTML
- Produire la documentation
- Pour une «bonne» documentation

Présentation

En Logique, vous avez vu le concept de *Liste*

Liste : Séquence d'éléments (ordonnés mais pas nécessairement triés) auxquels on accède via leur **position**

Ce concept est présent en Java

- Pas dans le langage
- Mais via l'API standard
  (classe java . util . ArrayList)

Déclaration / Création

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

- On spécifie le type des éléments (via les <>)
- Crée une liste **vide**
- Elle pourra évidemment grandir au besoin (pas de limite)

Déclaration / Création

Écriture viellie (**non recommandée**) :
Ne pas mettre les <>

```
ArrayList liste = new ArrayList ();
```

En Java7, on peut ne pas spécifier le type des éléments à droite (déduit par le compilateur)

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

Ajout d'éléments

add(E)	ajoute en fin
add(**int**, E)	ajoute (insère) en position donnée

- Le premier élément est en position **0**
- Une insertion provoque un décalage des éléments suivants

Exemple :

```
ArrayList<String> dictionnaire = new ArrayList<String>();
dictionnaire.add("zèbre");
dictionnaire.add("éléphant");
dictionnaire.add(1, "girafe");
// contient : [ "zèbre", "girafe", "éléphant" ]
```

Affichage

La méthode toString a été récrite pour afficher les éléments

Exemple :

```
ArrayList<String> dictionnaire = new ArrayList<String>();
dictionnaire.add("zèbre");
dictionnaire.add("éléphant");
dictionnaire.add(1, "girafe");
System.out.println( dictionnaire );
// affiche : [ "zèbre", "girafe", "éléphant" ]
```

Taille de la liste

size ()	donne la taille
isEmpty()	indique si c'est vide

Exemple :

```
ArrayList<String> dictionnaire = new ArrayList<String>();
System.out.println( dictionnaire.size() );      // 0
System.out.println( dictionnaire.isEmpty() );   // true
dictionnaire.add("zèbre");
dictionnaire.add("éléphant");
dictionnaire.add(1, "girafe");
System.out.println( dictionnaire.size() );      // 3
System.out.println( dictionnaire.isEmpty() );   // false
```

Accès aux éléments

get(**int**) | demande un élément

- Permet notamment le parcours

Exemple :

```
public static void monAffichage( ArrayList<String> liste ) {
  for(int i=0; i<liste.size(); i++) {
    System.out.println( i + ": " + liste.get(i) );
  }
}
```

Parcours

Parcours d'une liste via le **foreach** :

Exemple :

```
for (String mot : dictionnaire){
  System.out.println(mot);
}
```

- La variable `mot` prend chaque valeur de la liste
- La position de `mot` est inconnue : remplacement/suppression impossibles

Remplacement

set(**int**, E) | remplace l'élément en position donnée

Exemple :

```
public static void remplacer( ArrayList<String> dico ) {
   String mot ;
   for (int i=0; i<dico.size (); i++ ) {
      mot = dico.get(i );
      if (mot.charAt(0) == 'a') {
         dico.set(i ,"remplacé");
      }
   }
}
```

Recherche

boolean contains(E)	indique si l'élément est présent
int indexOf(E)	donne l'indice de la 1ère occurence de l'élément dans la liste

Recherche un élément de **même valeur**

- Utilisation de la méthode equals
- Nécessité de la redéfinir pour nos propres classes

Suppression

remove(**int**)	enlève l'élément en position donnée
remove(E)	enlève un élément donné

- Comme pour la recherche, se base sur la méthode equals
- Pas de méthode pour supprimer le dernier mais on peut écrire `liste.remove( liste.size()-1 )`

Wrapper

Seuls les objets sont permis dans les listes.

- Et pour les types primitifs (**int**, **boolean**, ...) ?
- Existence de *wrapper* (*enveloppe*)
  - Englobe une valeur primitive dans un objet
  - Conversion automatique de l'un à l'autre

Type primitif	Enveloppe
int	Integer
boolean	Boolean
char	Character
...	...

Wrapper

Exemple : Combien de conversions automatiques en tout ?

```
import java.util.ArrayList;
public class TestBox {
  public static void main ( String [] args ) {
    ArrayList<Integer> liste = new ArrayList<Integer>() ;
    liste.add(1) ; liste.add(14); liste.add(1) ;
    int premier = liste.get(0);
    liste.set(2, liste.get(2) + 1 );
  }
}
```

LinkedList

La classe LinkedList propose aussi le concept de *Liste*

- On retrouve toutes les méthodes vues pour ArrayList
- Pouquoi plusieurs versions ?
  - Font la même chose
  - **mais** différences en terme de performances

LinkedList

La classe LinkedList propose aussi le concept de *Liste*

- On retrouve toutes les méthodes vues pour ArrayList
- Pouquoi plusieurs versions ?
  - Font la même chose
  - **mais** différences en terme de performances

ArrayList	LinkedList
éléments stockés dans un tableau	éléments stockés dans une liste chainée (cf. Logique 2ème)
recommandé en général (ex : plus rapide pour accéder à un élément)	parfois plus rapide (ex : ajout en début)

Le concept d'interface

Supposons qu'on veuille écrire une méthode qui affiche un élément sur deux

- d'une ArrayList de chaines

```
public static void afficher1Sur2(ArrayList<String> liste) {
    for( int i=0; i<liste.size(); i+=2 ) {
        System.out.println( liste.get(i) );
    }
}
```

- d'une LinkedList de chaines

```
public static void afficher1Sur2(LinkedList<String> liste) {
    for( int i=0; i<liste.size(); i+=2 ) {
        System.out.println( liste.get(i) );
    }
}
```

Le concept d'interface

Le code écrit est presque identique

- Seule la déclaration du paramètre change

On aimerait éviter de dupliquer le code

- Indiquer qu'on accepte une ArrayList **ou** une LinkedList
- Et même n'importe quelle classe qui définit le concept de liste
- En fait, il nous suffit que les méthodes d'une liste existent

Interface

Interface : suite de déclarations de méthodes

Exemple :

```
interface MonInterface {
    void maMéthode1(int a);
    boolean maMéthode2(char c);
}
```

- Notez les ;
- On donne l'entête mais pas le code

Interface

Implémenter une interface : Définir toutes les méthodes d'une interface

Exemple :

```
public MaClasse implements MonInterface {
    public void maMéthode1(int a) {// le corps de la méthode...}
    boolean maMéthode2(char c) {// le corps de la méthode...}
    // + d'autres méthodes si on veut
}
```

- On déclare qu'on implémente l'interface
- Le compilateur vérifie qu'on fournit bien le code de chaque méthode

Interface List

Il existe une interface List

- Définit toutes les méthodes déjà vues pour les listes
- Implémentée par ArrayList et LinkedList
- Se voit facilement dans la *javadoc*

Interface et polymorphisme

Une interface définit un *type*

- On peut donc utiliser une interface dans une déclaration (variable, paramètre, ...)
- Là où une interface est attendue, on peut trouver n'importe quelle classe l'implémentant (polymorphisme)

Exemple :

```
public static void afficher1Sur2 ( List<String> liste ) {
  for (int i=0; i<liste.size(); i+=2) {
    System.out.println( liste.get(i) );
  }
}
```

Programmation par interface

Bon usage : programmer le plus possible avec les interfaces (et pas l'implémentation)

- Choix de l'implémentation uniquement lors de l'instanciation

```
List <String> maListe;
maListe = new ArrayList<String>();
```

- Assure que l'on ne va utiliser que les méthodes définies dans l'interface
- Facilite le changement d'implémentation

Collections

La classe java.util.Collections propose des services pour les listes

max (List)	donne le maximum d'une liste
sort (List)	trie une liste
reverse (List)	inverse une liste
shuffle (List)	mélange une liste
...	...

Collections

Exemple :

```
List<String> animaux = new ArrayList<String>() ;

animaux.add("âne");
animaux.add("zèbre");
animaux.add("alouette");

System.out. println  ( Collections .max(animaux) );
```

affiche *âne* (pourquoi pas *zèbre*) ?

Crédits

Ce document a été produit avec les outils suivants

- La distribution `Ubuntu` du système d'exploitation `Linux`
- `LaTeX` comme système d'édition
- La classe `Beamer` pour les transparents
- Les packages `listings`, `fancyvrb`, ...
- Les outils `make`, `rubber`, `pdfnup`, ...