Le langage Java

Les collections et les listes

Programme détaillé ou sommaire

Présentation
- Les limites des tables
- Les collections

Les Listes
- Les listes les plus courantes
- Ajout d'élément
- Suppression d'élément
- Autres méthodes
- Parcours d'une liste
- Gestion des doublons
- ConcurrentModificationException

TP

Annexe: Focus sur l'Itérator

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Chapitre 1
Présentation

Les limites des tableaux

diginamic
FORMATION
« Votre passeport pour l'emploi numérique »

Limites des tableaux

Les collections sont l'évolution logique des tableaux.

Les tableaux peuvent être utilisés pour des opérations simples, mais:

- L'agrandissement du tableau doit être géré
- Le décalage dans le cas d'un élément supprimé doit être géré
- Les éléments sont obligatoirement indexés par des entiers

```
int[] tab = new int[10];
tab[0]=1; tab[1]=2;…
```

exemple d'utilisation d'un tableau

Les collections (1/2)

Les collections sont des classes Java permettant de faciliter la gestion d'ensembles d'éléments.

Opérations courantes :

- *Ajout d'élément*
- *Suppression d'élément*
- *Parcours de la Collection*

Les collections (2/2)

De nombreuses collections sont fournies avec Java Standard Edition :

- Package java.util
- Elles permettent de stocker des références vers tous types d'objets
- Elles n'ont pas de taille maximum prédéfinie
- Elles sont optimisées en fonction de besoins précis.

Chapitre 2
Les listes

List et ArrayList

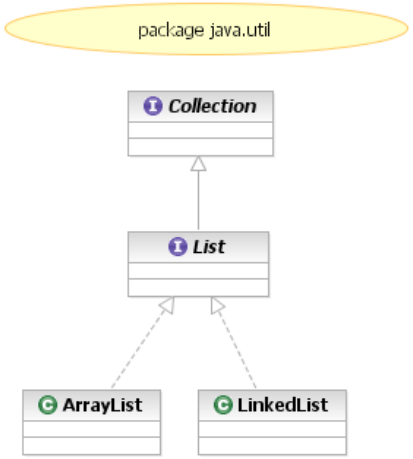
diginamic
FORMATION
« Votre passeport pour l'emploi numérique »

Les listes

Les listes sont des classes

- Qui implémentent l'interface java.util.List
- java.util.List hérite de l'interface Collection

Les listes sont indexées



Les listes les plus courantes

ArrayList

- La plus utilisée
- Souvent vue comme un "tableau dynamique"
- Excellentes performances pour le parcours d'éléments

LinkedList

- Liste chaînée
- Excellentes performances lors d'insertion/suppression d'éléments.
- Mauvaises performances pour le parcours d'éléments

Il existe également la classe Vector (généralement déconseillée)

Ajout d'éléments (1/3)

Les éléments sont ajoutés avec la méthode **add(…)**

```
ArrayList list = new ArrayList();
list.add(new Integer(3));
list.add("petit");
list.add(new Date());
```

Il est possible (mais rare) d'insérer des éléments hétérogènes dans une même liste : String, Integer, CompteCourant…

Ajout d'éléments (2/3)

Pour **éviter les listes hétérogènes**, il faut **typer** la liste (cf. Java 5)

Notation diamant <>

```
ArrayList<String> list = new ArrayList<>();
list.add("le");
list.add("petit");
list.add("chat");
```

Avec une liste typée, l'ajout d'un type non attendu provoque une erreur de compilation

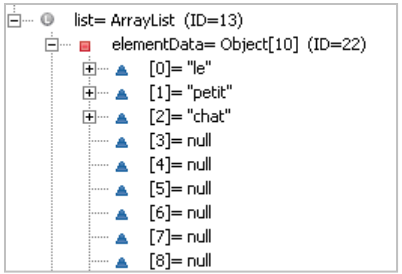
Ajout d'éléments (3/3)

Exemple : ArrayList

- Éléments stockés dans un tableau à l'intérieur de l'objet ArrayList.
- A chaque ajout, vérification que la taille maximum du tableau n'a pas été atteinte.

  Si la taille maximum est atteinte, un nouveau tableau est créé.

```
ArrayList<String> list = new ArrayList<>();
list.add("le");
list.add("petit");
list.add("chat");
```



Contenu de l'objet list

Suppression d'élément

Méthode **remove(…)**

- Supprime l'objet à partir de l'objet lui-même ou à partir de son index.

```
User u1 = new User("jean", "dupont");
User u2 = new User("jean", "durand");
User u3 = new User("jean", "martin");

ArrayList<User> list = new ArrayList<>();
list.add(u1);
list.add(u2);
list.add(u3);

list.remove(u3);      // suppression de la référence vers u3
                      // Attention ! Ne marche que si equals est redéfinie

list.remove(0);       // suppression de la référence vers
                      // l'élément en position 0
```

Autres méthodes

size()

- Renvoie le nombre d'éléments de la collection

isEmpty()

- Renvoie 'true' si la collection est vide

toArray()

- Créé un tableau contenant tous les éléments de la liste

Ces méthodes sont déclarées dans l'interface Collection. Elles ne sont pas spécifiques aux listes.

Parcours d'une liste (1/5)

Iterator

- Permet de parcourir une collection en récupérant les éléments successivement
- Méthode de parcours homogène pour tous les types de collection
- Garantit la cohérence de la collection parcourue

    Pas de modifications externes pendant le parcours

Parcours d'une liste (2/5)

Récupération d'un **Iterator** sur une collection donnée :

```
Iterator<T> iterator = maCollection.iterator()
```

Iterator contient 3 méthodes :

- boolean hasNext() : renvoie true s'il reste des éléments à parcourir dans la liste.
- T next() : renvoie le prochain objet stocké dans la liste.
- void remove() : supprime l'élément en cours de la liste.

Parcours d'une liste (3/5)

Exemple

```
User u1 = new User("jean", "dupont");
User u2 = new User("jean", "durand");
User u3 = new User("jean", "martin");

ArrayList<User> list = new ArrayList<>();
list.add(u1);
list.add(u2);
list.add(u3);

Iterator<User> iterator = list.iterator();
while (iterator.hasNext()) {
    User myUser = iterator.next();
    System.out.println(myUser);
}
```

Parcours d'une liste (4/5)

Parcours avec une boucle objet

```
User u1 = new User("jean", "dupont");
User u2 = new User("jean", "durand");
User u3 = new User("jean", "martin");

ArrayList<User> list = new ArrayList<>();
list.add(u1);
list.add(u2);
list.add(u3);

for (User user: list) {
    System.out.println(user);
}
```

Ci-contre la liste est parcourue via une référence de type User.

Parcours d'une liste (5/5)

Parcours avec une boucle indexée

```
User u1 = new User("jean", "dupont");
User u2 = new User("jean", "durand");
User u3 = new User("jean", "martin");

ArrayList<User> list = new ArrayList<>();
list.add(u1);
list.add(u2);
list.add(u3);

for ( int i=0; i<list.size(); i++) {
    User user = list.get(i);
    System.out.println(user);
}
```

Gestion des doublons

Les listes acceptent les doublons

- Les doublons sont positionnés à un index différent
- La liste stocke plusieurs références vers le même objet

```
User u1 = new User("jean", "dupont");

ArrayList<User> list = new ArrayList<>();
list.add(u1);
list.add(u1); //deux références pointent vers le même objet
```

Suppression d'élément: ConcurrentModificationException

Exception renvoyée par la méthode remove(…)

- Suppression d'un élément dans une collection en cours de parcours (boucle).

```
ArrayList<User> list = new ArrayList<>();
list.add(new User("jean", "dupont"));
list.add(new User("jean", "durand"));
list.add(new User("jean", "martin"));
list.add(new User("marcel", "ferrand"));

for (User user : list) {
	if (user.getNom().equals("durand")) {
		list.remove(user);
	}
}
```

Eviter la ConcurrentModificationException

Parcourir la collection avec un **iterator** et utiliser la méthode **remove()**

```
ArrayList<User> list = new ArrayList<>();
list.add(new User("jean", "dupont"));
list.add(new User("jean", "durand"));
list.add(new User("jean", "martin"));
list.add(new User("marcel", "ferrand"));

Iterator<User> iter = list.iterator();
while (iter.hasNext()) {
	User user = iter.next();
	if (user.getNom().equals("durand")) {
		iter.remove();
	}
}
```

Atelier (TP)

Objectifs du TP: manipuler les collections et plus particulièrement les List et ArrayList

Description du TP:

Dans ce TP n°10, vous allez créer diverses listes et apprendre à les utiliser.

Annexe
Zoom sur l'Iterator

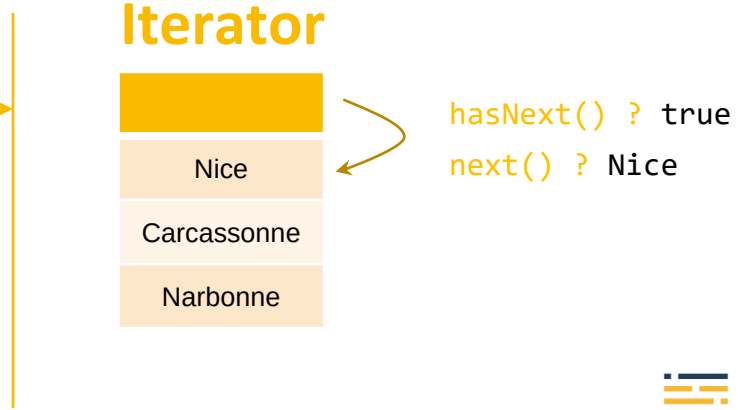
Iterator – Première itération

- Zoom sur le fonctionnement de l'**Iterator**

```
ArrayList<String> liste = new ArrayList<>();
liste.add("Nice");
liste.add("Carcassonne");
liste.add("Narbonne");

Iterator<String> iter = liste.iterator();
while (iter.hasNext()) {
    String nomVille = iter.next();
}
```

*Par défaut **l'Iterator** ne pointe pas sur la première valeur.*



© TOUS DROITS RÉSERVÉS À ODDET TECHNOLOGY

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Iterator – Deuxième itération

- Zoom sur le fonctionnement de l'**Iterator**

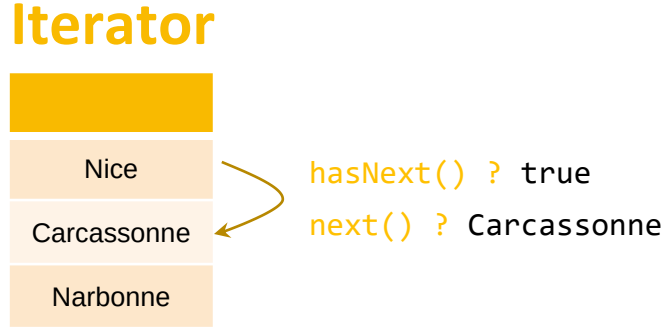
```
ArrayList<String> liste = new ArrayList<>();
liste.add("Nice");
liste.add("Carcassonne");
liste.add("Narbonne");

Iterator<String> iter = liste.iterator();
while (iter.hasNext()) {
    String nomVille = iter.next();
}
```

l'itérator pointe sur la 1ère ligne

hasNext() retourne true.

next() retourne Carcassonne qui est la valeur suivante.



© TOUS DROITS RÉSERVÉS À ODDET TECHNOLOGY

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

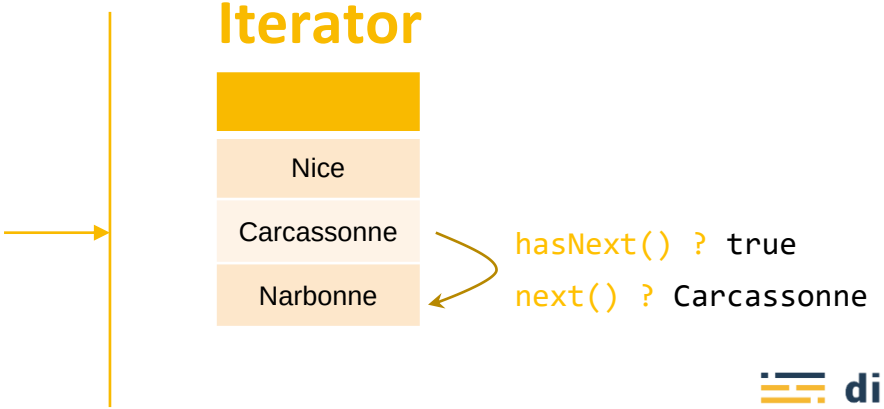
Iterator – Troisième itération

- Zoom sur le fonctionnement de l'**Iterator**

```
ArrayList<String> liste = new ArrayList<>();
liste.add("Nice");
liste.add("Carcassonne");
liste.add("Narbonne");

Iterator<String> iter = liste.iterator();
while (iter.hasNext()) {
    String nomVille = iter.next();
}
```

l'itérator pointe sur la 2ème ligne
hasNext() retourne true.
next() retourne Narbonne



© TOUS DROITS RÉSERVÉS À ODDET TECHNOLOGY

diginamic FORMATION
« Votre passeport pour l'emploi numérique »

Iterator – Dernière itération

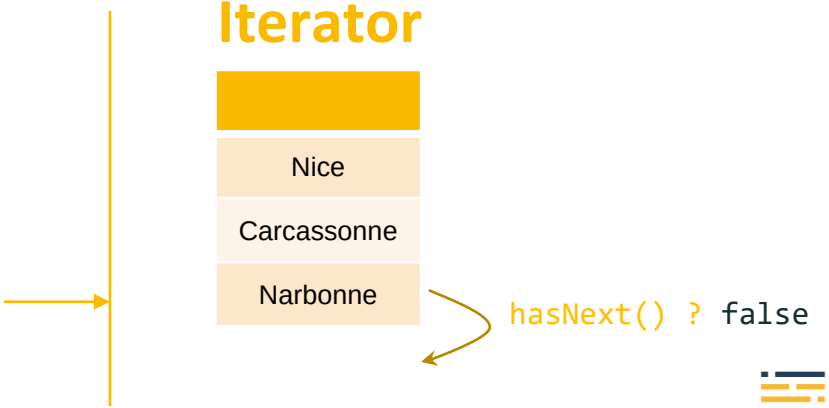
- Zoom sur le fonctionnement de l'**Iterator**

```
ArrayList<String> liste = new ArrayList<>();
liste.add("Nice");
liste.add("Carcassonne");
liste.add("Narbonne");

Iterator<String> iter = liste.iterator();
while (iter.hasNext()) {
   String nomVille = iter.next();
}
```

l'itérator pointe sur la 3ème ligne

hasNext() retourne false.



© TOUS DROITS RÉSERVÉS À ODDET TECHNOLOGY

diginamic FORMATION
« Votre passeport pour l'emploi numérique »