Cours Developpement Efficace

Introduction aux Listes simples en Java

En Java, on a une classe Cellule `Cell.java` qui représente une Cellule pour notre Liste

`[1, null] -> [2, null] -> [3, null] -> |///`

```java
// class Cell.java
public class Cell {
    public int value;
    public Cell next;
    public Cell(int value) {
        this.value = value;
        next = null;
    }
}
```

```java
// class List.java
public class List {
    public Cell head;
    public int size;
    public List() {
        head = null;
        size = 0;
    }
    // Reste du code ci-dessous
}
```

Maintenant on souhaite renvoyer une cellule par la méthode `public Cell find (int value) {}`.

```java
public Cell find (int value) {
    Cell c = head;
    while ((c != null) && (c.value != value)) {
        c = c.next;
    }
    return c;
}
```

Maintenant on souhaite renvoyer la valeur d'une cellule par la méthode `public Cell get (int index) {}`.

```java
public Cell get (int index) {
    Cell c = head;
    int i = 0;
    while ((c != null) && (i < index)) {
        c = c.next;
        i += 1;
    }
}
```

On souhaite ajouter une valeur a la liste par la méthode `public Cell append (int value) {}`.

```java
public Cell append (int value) {
    Cell c = null;
    Cell newCell = new Cell (value);
    if (size == 0) {
        head = newCell;
    } else {
        c = get(size - 1);
        c.next = newCell;
    }
    size += 1;
    return newCell;
}
```

On souhaite ajouter une valeur à un index précis par la méthode `public Cell insert (int value, int index) {}`.

```
public Cell insert (int value, int index) {
    Cell c = null;
    Cell newCell = new Cell(value);
    if (index > size) { index = size; }
    if (size == 0) { head = newCell; }
    else if (index == 0) {
        newCell.next = head;
        head = newCell;
    } else {
        c = get(index-1);
        newCell.next = c.next;
        c.next = newCell;
    }
    size += 1;
    return newCell;
}
```

On souhaite maintenant remplacer une valeur par la méthode `public Cell replace (int replace, int index) {}`.

```
public Cell replace (int value, int index) {
    if ((index<0) && (index >= size)) return null;
    Cell c = get(index);
    c.value = value;
    return c;
}
```

Les Collections en Java

- **Set** : ensemble d'objets non indicé, sans doublons
- **List** : ensemble d'objets non indicé, éventuellement avec doublons
- **Map** : ensemble associatif d'objets, non indicé, chaque objet étant associé à une clé
  - Une clé est unique dans **Map** mais plusieurs clés peuvent être associées à un même objet.
- **Queue** : ensemble d'objets, non indicé avec un schéma d'accès de type FIFO/LIFO
  - **FIFO** : First In First out
  - **LIFO** : Last In First Out

Les piles sont souvent utilisés en programmation.

En **réalité**, Set, List, Map et Queue sont des interfaces.

En **pratique**, on utilise les implémentations HashSet, ArrayList, HashMap, ArrayDeque. Ces collections sont génériques :

```
// On ne peut pas écrire cela
Set <int> set = new HashMap<int>();

// On écrit plutôt cela
Set <Integer> set = new HashMap<Integer>();
Set <Integer> set = new HashMap<>();

// ------------------
// Methodes communes
// ------------------
// int size();
// void clear();
// boolean istemply();
```

HashSet: - **boolean add(E e)**: ajoute l'élement e de la classe E. Si E n'est pas de la classe (même classe) indiquée par constructeur, erreur compil. Renvoie vrai si l'insertion a lieu.

- **boolean remove(Object o)**: enlève o s'il existe et renvoie true, sinon false, pas d'erreur de compil si o n'est pas de la même classe.

- **boolean contains(Object o)**: vérifie si l'élément spécifié est présent dans le HashSet et retourne true si l'élément existe, sinon elle retourne false.

Exemple :

```
public class A {}
public class B extends A {}

Set<A> set = new HashSet<>();

set.add(new A());
set.add(new A());
set.add(new B());

set.add(new Date()); // Cette ligne provoque erreur compil car Date pas enfant de A
```

ArrayList: - **boolean add(E e)**: ajoute en fin de liste.

- **void add(int index, E e)**: insertion en index. Si index < 0 ou index $>$ taille liste, ne fait rien et lève une exception.
- **E get(int index)** : retourne l'élément à l'index spécifié.
- **int indexOf(Object o)** : retourne l'index de la première occurrence de l'élément o, ou -1 si l'élément n'est pas trouvé.
- **E remove(int index)** : supprime et retourne l'élément à l'index spécifié.

Exemple :

```
import java.util.ArrayList;

public class A {}
public class B extends A {}

List<A> list = new ArrayList<>();
A a1 = new A();
A a2 = new A();
Date d = new Date();

list.add(a1);
list.add(a2);
list.add(d);
list.add(15, a1); // Erreur d'execution

A aa = null;
aa = list.get(0); // aa référence le même objet que a1
aa = list.remove(1); // aa référence le même objet que a2
int pos = list.indexOf(a1); // Ok, renvoie 0
pos = list.indexOf(d); // Ok, renvoie -1
```

HashMap : - **V put (K key, V value)**: rajoute/modifie couple {clé; valeur}. Si clé existe déjà l'ancienne valeur est écrasée par la nouvelle. Renvoie l'ancienne valeur ou null su elle n'existe pas.

- **V get(Object key)**: renvoie l'objet associé à la clé key si elle existe ou null.
- **V remove(Object key)**: supprime la valeur associée à la clé.
- **boolean containsKey(Object key) / boolean containsValue(Object value)**
- **Set keySet()**: renvoie un set des clés.

Exemple :

```
Map<String, A> map1 = new HashMap<>();
Map<A, Integer> map2 = new HashMap<>();

A a1 = new A();
A a2 = new A();
Date d = new Date();
```

```
map1.put("toto", a1);
map1.put("toto", a2); // écrasement de a1 par a2
map1.put(a1, "tutu"); // erreur compil
map2.put(a1, new Integer(10));
map2.put(d, new Integer(5));
map2.containsKey(d); // renvoie False
map2.containsKey(a1); // rnevoie True
map2.remove(d); // Ne fait rien
map1.remove("toto"); // Renvoie a1
map2.remove(a1); // renvoie 10
```

ArrayDeque: - **File**: - **boolean offer(E e)**: ajoute e en fin de queue - **E pull()**: supprime et renvoie l'élément en tête de queue. Si la queue est vide, renvoie null. - **Pile**: - **void push(E e)**: ajoute en tête de queue. - **E pop()**: supprime et renvoie l'élément de tête de queue. Contrairement à pull provoque erreur si queue vide.

- **boolean offerFirst(E e)/boolean offerLast(E e)**
- **void addFIrst(E e)/void addLast(E e)**
- **E getFirst()/E getLast()**
- **E peekFirst()/E peekLast()**

Exemple:

```
import java.util.ArrayDeque;

Queue<Double> q = new ArrayDeque<>();
// FIFO
q.offer(1);
q.offer(2);
int val = q.pull(); // renvoie 1

// LIFO access
q.push(3);
val = q.pop(); // renvoie 3
val = q.pop(); // renvoie 2
```

Parcourir une collection en Java

Par le biais de : for

Contrainte: on ne peut pas modifier la collection que l'on parcoure.

Exemple :

```
import java.sql.Array;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.HashSet;

Set<String> set = new HashSet<String>();
List<Date> list = new ArrayList<>();
Map<Integer, String> map = new HashMap<>();

for(String s : get) {
    // s ...
}
for (Date d : list){
    // d ...
}
for (MapEntry<Integer, String> e : map.entrySet()){
    Integer k = e.getKey();
    String v = e.getValue();
}
```

Par le biais de : iterator

Exemple :

```
import java.sql.Array;
import java.util.*;

Set<String> set = new HashSet<String>();
List<Date> list = new ArrayList<>();
Map<Integer, String> map = new HashMap<>();

// SET
Iterator<String> is = set.iterator();
while(is.

hasNext()){
String s = is.next();
// ...
if(...) is.remove();

remove();
}

// LIST
ListIterator il = list.listIterator();
while (il.hasNext()) {
    Date d = il.next();
    // ...
    il.previous();
    il.add(E e);
    il.remove();
}

// MAP
Iterator<Integer> in = map.keySet().iterator();
while (in.hasNext()) {
    Integer k = in.next();
    String v = map.get(k);
    // ...
}
```

Astuce:

```
Map<Integer, Double> map = new HashMap<>();
// Mauvaise pratique : attention aux types ! Fonctionne quand même en Java
map.put(5, 1.234); // le compilateur va faire le nécessaire pour transform int en Integer et double en Double
// Bonne pratique
double val = map.get(5); // val = 1.234
Double d = 6.6;
if (d == 6.6) {
    // ...
}
```