Chapitre 1: Listes

Christophe Morvan

Université Paris-est, Marne-la-Vallée

8 septembre 2015

Plan

Préambule

- La plupart des chapitres reprendront une structure similaire au plan indiqué précédemment.
- Nous utiliserons une syntaxe semblable à celle de Java pour présenter les exemples ou les signatures de fonctions. Néanmoins, le contenu de ce cours n'est pas propre à Java.
- Les types génériques seront beaucoup utilisés.

Progression

Premier exemple

Types de base

- Caractères
- Entier (différentes tailles)
- Flottants
- Autres...

Premier exemple

Types de base

- Caractères
- Entier (différentes tailles)
- Flottants
- Autres...

Tableau

La structure la plus simple pour mémorier des collections est constituée du tableau, une série d'enregistrements *contigus* en mémoire.

Problèmes des tableaux

Problèmes des tableaux

- Agrandir/réduire

Problèmes des tableaux

Problèmes des tableaux

- Agrandir/réduire
- Décalage

Problèmes des tableaux

Problèmes des tableaux

- Agrandir/réduire
- Décalage
- Insertion

Problèmes des tableaux

Problèmes des tableaux

- Agrandir/réduire
- Décalage
- Insertion

Solution : la liste

La liste est une *structure de donnée* permettant de stocker des collection d'objets (en général de même type).

Progression

Définitions

Soit `T` un type quelconque.

Définition 1 (Récursive)

Une liste de `T` est, soit la *liste vide*, soit composée d'une *tête de liste* de type `T` avec une *queue de liste*, de type liste de `T`.

Définitions

Soit `T` un type quelconque.

Définition 1 (Récursive)

Une liste de `T` est, soit la *liste vide*, soit composée d'une *tête de liste* de type `T` avec une *queue de liste*, de type liste de `T`.

Définition 2 (séquentielle)

Une liste de `T` est une suite d'éléments de type `T`, ainsi qu'un pointeur vers un élément distingué, l'*éléments courant*.

Interface récursive

Interface récursive – `ListR<T>`

- `static nil` : la liste vide
- `static ListR<T> cons (T head , ListR<T> tail)` : construit la liste avec `head` comme tête de liste et `tail` comme queue de liste.
- `boolean isEmpty()` : fournit `true` lorsque la liste courante est vide
- `T car()` : fournit comme résultat la tête de liste de la liste courante.
- `ListR<T> cdr ()` : fournit comme résultat la queue de liste de la liste courante.

Exemple

```
ListR<Integer> l_1=ListR.nil(), l_2=ListR.cons(59, l_1);
l_2 = ListR.cons(32, l_2); l_1 = l_2.cdr();
int val = 17 + l_2.car());
```

Interface séquentielle

Interface Séquentielle – `ListS<T>`

- Constructeur classique.
- `boolean isEmpty()` : fournit `true` lorsque la liste courante est vide
- `void add(T elem)` : ajoute `elem` après l'élément courant
- `T current()` : fournit l'élément courant
- `boolean hasNext()` / `boolean hasPred()` : fournit `true` lorsque l'élément courant possède un successeur/prédécesseur
- `void next()` / `void pred()` : déplace l'élément courant sur son successeur/prédécesseur
- `void remove()` : supprime l'élément courant, le nouvel élément courant est son prédécesseur, à défaut son successeur.
- `void onHead()` / `void onTail()` : l'élément courant devient la tête/queue de la liste.

Exemple

Exemples séquentiels

```
ListeS<Integer> l_1=new ListeS<Integer>();
l_1.add(59); l_1.add(32);
l_1.pred(); l_1.remove();
System.out.println(l_1.current());
```

Affichage ?

Exemple

Exemples séquentiels

```
ListeS<Integer> l_1=new ListeS<Integer>();
l_1.add(59); l_1.add(32);
l_1.pred(); l_1.remove();
System.out.println(l_1.current());
```

Affichage ? 32

Variantes classiques

Pile (*Stack*) – LIFO

Les piles des cas particuliers de listes avec des accès restreints. Ajout *et* retrait d'élément *exclusivement* en tête de liste. En général pas de navigation dans la pile.
LIFO : Last In Fist Out

Variantes classiques

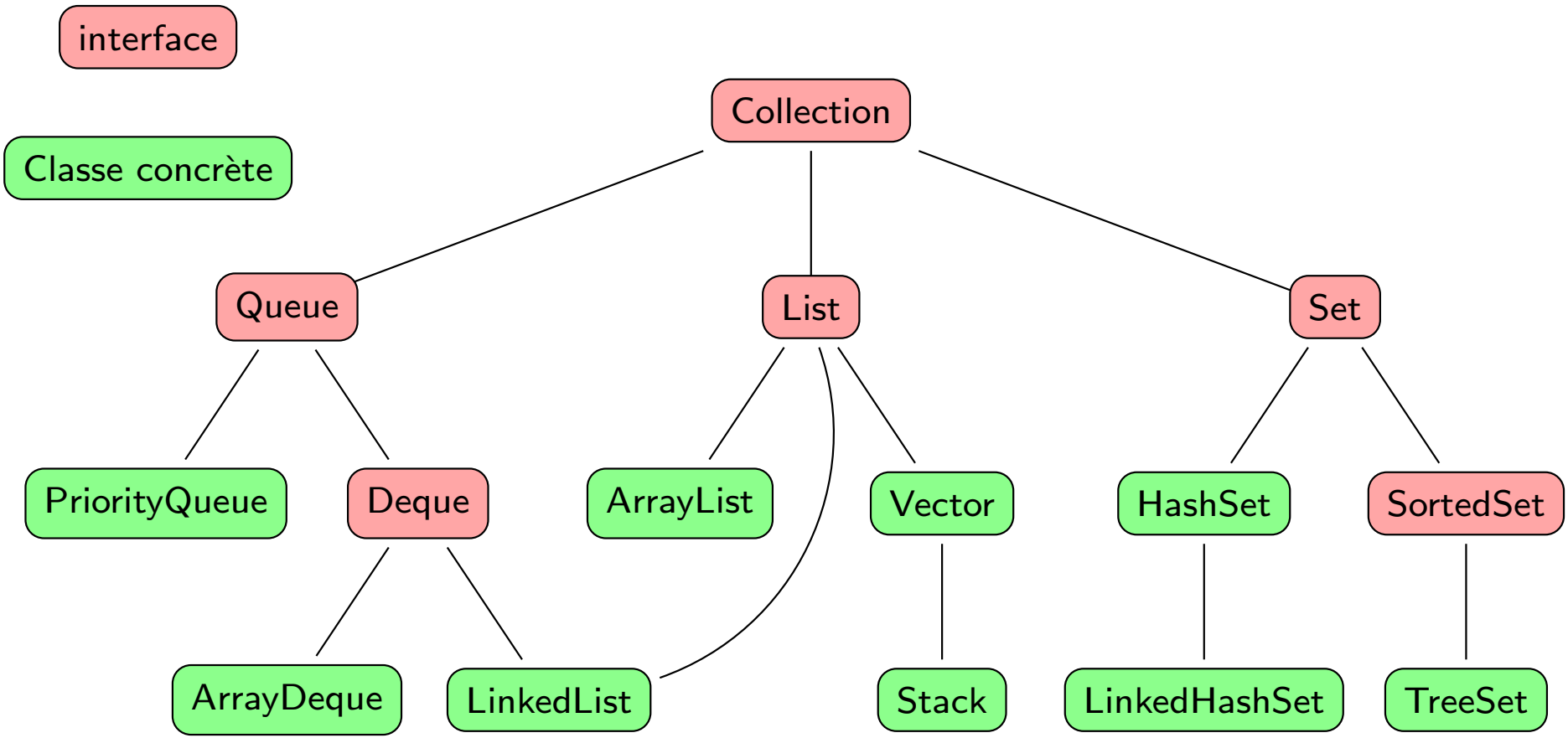
Pile (*Stack*) – LIFO

Les piles des cas particuliers de listes avec des accès restreints. Ajout *et* retrait d'élément *exclusivement* en tête de liste. En général pas de navigation dans la pile.
LIFO : Last In Fist Out

File (*Queue*) – FIFO

Les files sont des cas particuliers de listes avec des accès restreints. Les insertions sont faites en tête de liste et les retraits sont faits en queue de liste. En général pas de navigation dans la file.
FIFO : First In First Out

API Java : les collections



Vue d'ensemble

Les collections sont rassemblées dans `java.util`

Vue d'ensemble

Les collections sont rassemblées dans `java.util`

Interfaces

Générale : `Collection<T>` ensembles d'objets avec ou sans répétitions

Vue d'ensemble

Les collections sont rassemblées dans `java.util`

Interfaces

Générale : `Collection<T>` ensembles d'objets avec ou sans répétitions
Spécifiques :

- `List<T>` : listes (répétitions)
- `SortedSet<T>` : ensembles triés
- `Queue<T>/Dequeue<T>` : files (simple/double)
- `Set<T>` : ensembles (pas de répétitions)

Vue d'ensemble

Les collections sont rassemblées dans `java.util`

Interfaces

Générale : `Collection<T>` ensembles d'objets avec ou sans répétitions
Spécifiques :

- `List<T>` : listes (répétitions)
- `SortedSet<T>` : ensembles triés
- `Queue<T>/Dequeue<T>` : files (simple/double)
- `Set<T>` : ensembles (pas de répétitions)

Exemple

```
Collection<Integer> col = new ArrayList<Integer>();
List<String> li = new LinkedList<String>();
```

Collection<T>

Méthodes

- `boolean add (T elem)` : ajoute l'élément `elem`
- `boolean remove (Object o)` : supprime une instance de l'élément `o` de la collection
- `boolean contains (Object o)` : vérifie la présence de `o`
- `int size()` : donne la taille
- `boolean isEmpty()` : est-elle vide ?
- `Iterator<T> iterator()` : fournit un itérateur sur la collection courante.

List<T>

Méthodes

- `int indexOf (Object o)` : position de l'élément `o` (-1 si absent)
- `T get (int i)` : fournit l'élément à la position `i`
- `T set (int i, T elem)` : fournit l'élément à la position `i` et le remplace par `elem`

List<T>

Méthodes

- `int indexOf (Object o)` : position de l'élément o (-1 si absent)
- `T get (int i)` : fournit l'élément à la position `i`
- `T set (int i, T elem)` : fournit l'élément à la position `i` et le remplace par `elem`

Exemple

```
List<String> li = new LinkedList<String>();
li.add("Bonjour");li.add("Bonsoir");
li.set(1,"Bye");
```

Progression

Implémentations

Deux grandes familles d'implémentations.

Implémentations

Deux grandes familles d'implémentations.

Par tableau

Idée : un tableau contient les éléments de la liste, accompagné de quelques autres attributs : la taille du tableau et celle de la liste, l'indice de l'élément courant.
Point clé : lorsqu'on ajoute un nouvel élément dans la liste et que le tableau est saturé, on agrandit la taille du tableau. On diminue sa taille lorsque le nombre d'éléments dans la liste est trop petit.

Implémentations

Deux grandes familles d'implémentations.

Par tableau

Idée : un tableau contient les éléments de la liste, accompagné de quelques autres attributs : la taille du tableau et celle de la liste, l'indice de l'élément courant.
Point clé : lorsqu'on ajoute un nouvel élément dans la liste et que le tableau est saturé, on agrandit la taille du tableau. On diminue sa taille lorsque le nombre d'éléments dans la liste est trop petit.

Par chaine (liste – doublement – chaînée)

Idée : chaque élément de la liste connaît l'adresse de son successeur.

Schéma d'implémentation – 1

Un maillon

```
class Link<T>{
    /* En général, c'est une classe interne */
    T elem;
    Link<T> next;
    /* Liste doublement chainée : deux liens par maillon next - pred */
    public Link(T elem){
        this.elem =elem;
        next=null;
    }
    public Link(T elem, Link<T> next){
        this.elem =elem;
        this.next=next;
    }
    /* Accesseurs publics : getElem()/getNext()
                  setElem(T elem)/setNext(Link<T> next;)
     */
}
```

Schéma d'implémentation – 2

Une liste

```
class ListC<T>{
    Link<T> head,tail,current;
    int size;
    public List<T>(){
        head=current=tail=null; size=0;
    }
    public boolean isEmpty(){
        return head==null;
    }
    public void add (T elem){ /* Après current */
        this.size++;
        if (this.isEmpty())head=current=tail=new Link<T>(elem);
        else {
            current.next = new Link<T>(elem, current.next);
            if (current==tail) tail=current.next;
        }
        /* Autres méthodes */ }
```

Progression

Récursivité

Observation

Lorsqu'une structure peut se définir de façon récursive, il est intéressant de formuler les algorithmes de façon récursive.

Schéma général

algorithme (objetRec)

Cas d'arrêt, valeur de retour sans appel récursif

Cas récurrent, appel récursif sur la sous-stucture de objetRec

Synthèse du résultat avec le résultat récursif et *l'élément courant*.

Récursivité

Observation

Lorsqu'une structure peut se définir de façon récursive, il est intéressant de formuler les algorithmes de façon récursive.

Schéma général

algorithme (objetRec)

Cas d'arrêt, valeur de retour sans appel récursif

Cas récurrent, appel récursif sur la sous-stucture de objetRec

Synthèse du résultat avec le résultat récursif et *l'élément courant*.

Recherche simple

```
boolean rechRec(ListR<T> list, T elem){
    if (list.isEmpty())return false;
    if (elem == list.car())return true;
    else return rechRec(list.cdr(), elem);}
/*return (!list.isEmpty())&&((elem == list.car())||
                                    rechRec(list.cdr(), elem))*/
```

Tri à bulles

Tri le plus *naïf*
Performances médiocres (pour un algorithme de tri)

Tri à bulles

Tri le plus *naïf*
Performances médiocres (pour un algorithme de tri)

Schéma

On pose n égal à la taille de la liste

1) Chercher l'élément le plus grand de la liste parmis les n premières positions
2) Déplacer cet élément à la position n
3) Si $n > 1$, reprendre l'étape 1) avec $n = n - 1$

Tri à bulles

Tri le plus *naïf*
Performances médiocres (pour un algorithme de tri)

Schéma

On pose n égal à la taille de la liste

1) Chercher l'élément le plus grand de la liste parmis les n premières positions
2) Déplacer cet élément à la position n
3) Si $n > 1$, reprendre l'étape 1) avec $n = n - 1$

Complexité ?

Tri à bulles

Tri le plus *naïf*
Performances médiocres (pour un algorithme de tri)

Schéma

On pose n égal à la taille de la liste

1) Chercher l'élément le plus grand de la liste parmis les n premières positions
2) Déplacer cet élément à la position n
3) Si $n > 1$, reprendre l'étape 1) avec $n = n - 1$

Complexité ?

$n + (n-1) + \cdots + 2 =$

Tri à bulles

Tri le plus *naïf*
Performances médiocres (pour un algorithme de tri)

Schéma

On pose n égal à la taille de la liste

1) Chercher l'élément le plus grand de la liste parmis les n premières positions
2) Déplacer cet élément à la position n
3) Si $n > 1$, reprendre l'étape 1) avec $n = n - 1$

Complexité ?

$$n + (n-1) + \cdots + 2 = (n(n+1)/2) - 1$$

Tri fusion

Forme de tri particulièrement adapté aux listes.
Exemple typique de diviser pour régner.

Tri fusion

Forme de tri particulièrement adapté aux listes.
Exemple typique de diviser pour régner.

Schéma – Tri

On pose n égal à la taille de la liste

1) Si $n == 1$ la liste est triée.
2) Sinon, partager la liste en deux listes de $n/2$ éléments
3) Trier chacune des deux listes (appel récursif)
4) Fusionner les deux listes *triées*

Tri fusion

Forme de tri particulièrement adapté aux listes.
Exemple typique de diviser pour régner.

Schéma – Tri

On pose n égal à la taille de la liste

1) Si $n == 1$ la liste est triée.
2) Sinon, partager la liste en deux listes de $n/2$ éléments
3) Trier chacune des deux listes (appel récursif)
4) Fusionner les deux listes *triées*

Schéma – Fusion

1) Réaliser un parcours simultané des deux listes. Ajouter en fin de liste résultat le plus petit des deux.
2) Lorsqu'une des deux listes est terminée ajouter la seconde liste à la suite de la liste résultat.

Complexité fusion

Analyse

1) Partages de la liste

Complexité fusion

Analyse

1) Partages de la liste $O(\ln(n))$

Complexité fusion

Analyse

1) Partages de la liste $O(\ln(n))$
2) Fusion

Complexité fusion

Analyse

1) Partages de la liste $O(\ln(n))$
2) Fusion $O(n)$

Complexité fusion

Analyse

1) Partages de la liste $O(\ln(n))$
2) Fusion $O(n)$
3) Au total

Complexité fusion

Analyse

1) Partages de la liste $O(\ln(n))$
2) Fusion $O(n)$
3) Au total $O(n \ln(n))$

Progression

Quelques exemples

Liste

- Liste de clients
- Liste d'items sélectionnés
- Liste d'*observateurs* dans un patron observateur
- Listes des *composants* dans le patron composite
- ...

Quelques exemples

Liste

- Liste de clients
- Liste d'items sélectionnés
- Liste d'*observateurs* dans un patron observateur
- Listes des *composants* dans le patron composite
- ...

Pile/File

- Files d'attentes
- Files de priorités
- File de successeurs (parcours en largeur)
- Pile de successeurs (parcours en profondeur)