Initiation à la programmation Java
IP2 - Séance No 5

Yan Jurski

Université de Paris

(Rappels) Modélisation d'une liste d'éléments de type E

Fichier MaList.java

```
public class MaList implements ListIP2{
 private Cellule first;
 ...
}
```

Fichier Cellule.java

```
public class Cellule{
 private E content;
 private Cellule next;
 public Cellule getNext(){ return this.next; }
 public E getContent(){ return this.content; }
 ...
}
```

Université de Paris

Petit exercice Liste (révision)

Fichier ListIP2.java – Ecrite pour des éléments de type E

```
public interface ListIP2 { // position starts at 0
boolean remove(E x); // Removes the first occurrence of the element
...
}
```

- La cellule précédente est concernée
- Pensez que `first` peut changer
- Dessinez d'abord ...

Université de Paris

Fichier MaList.java

```
public class MaList implements ListIP2{
 private Cellule first;
 public boolean remove(E x){
  if ( this.isEmpty() ) return false;
  if ( first.getContent()== x) {
   first=first.getNext();
   // remarquer l'état de la mémoire (garbage collector ?)
   return true;
  }
  else return first.remove(x); // on peut choisir ce nom sans ambiguïté
}
```

Fichier Cellule.java

```
public class Cellule{
 private E content;
 private Cellule next;
 public Cellule getNext(){ return this.next; }
 public E getContent(){ return this.content; }
 ...
}
```

Fichier MaList.java

```
public class MaList implements ListIP2{
 ...
}
```

Fichier Cellule.java

```
public class Cellule{
 private E content;
 private Cellule next;
 public Cellule getNext(){ return this.next; }
 public E getContent(){ return this.content; }
 boolean remove(E x){ // on sait (voir MaListe) : this ne contient pas x
  Cellule tmp=this;
  // tmp s'arrête éventuellement sur la cellule précédent x
  while( (tmp.next != null) && (tmp.next.content != x) ){
   tmp=tmp.next;
  }
  if (tmp.next==null) return false;
  else {
   tmp.next=tmp.next.next;
   return true;
  }
 }
}
```

Fichier MaList.java

```
public class MaList implements ListIP2{
 ...
}
```

Fichier Cellule.java

```
public class Cellule{
 private E content;
 private Cellule next;
 public Cellule getNext(){ return this.next; }
 public E getContent(){ return this.content; }
 boolean remove(E x){ // on sait (voir MaListe) : this ne contient pas x
  Cellule tmp=this;
  // tmp s'arrête éventuellement sur la cellule précédent x
  while( (tmp.next != null) && (tmp.next.content != x) ){
   tmp=tmp.next;                          // et si on essayait avec un for ?
  }
  if (tmp.next==null) return false;
  else {
   tmp.next=tmp.next.next;
   return true;
  }
 }
}
```

Fichier MaList.java

```
public class MaList implements ListIP2{
 ...
}
```

Fichier Cellule.java

```
public class Cellule{
 private E content;
 private Cellule next;
 public Cellule getNext(){ return this.next; }
 public E getContent(){ return this.content; }
 boolean remove(E x){ // on sait (voir MaListe) : this ne contient pas x
  Cellule tmp;
  // tmp s'arrête éventuellement sur la cellule précédent x
  for( tmp=this; (tmp.next != null) && (tmp.next.content != x) ;
       tmp=tmp.next){}
  if (tmp.next==null) return false;
  else {
   tmp.next=tmp.next.next;
   return true;
  }
 }
}
```

Petits exercices Liste

- Vous devez absolument écrire rapidement ces méthodes
- En 5 à 10 minutes maximum ! Entrainez vous !

- Si vous voulez des exercices courts regardez par exemple www.hackerrank... où il y a toute une banque d'exercices.
  Le style qui y est suggéré est orienté `static`, j'aurais préféré des versions dynamiques, mais après tout l'exercice de style reste utile.
  Vous pouvez essayer d'écrire des solutions dans des versions avec `for` ou `while` ; et sur votre machine essayer la même chose en version dynamique.

Université de Paris

Nouveau chapitre : la Récursion !

Nouveau chapitre : la Récursion !

- La récursion est une caractéristique syntaxique :
  un objet ou une méthode fait référence à sa propre définition dans son expression.
- Elle se retrouve en
  - linguistique
  - biologie
  - mathématique
  - informatique
  - en psycho ...
- Et caractérise des objets finis !

Université de Paris

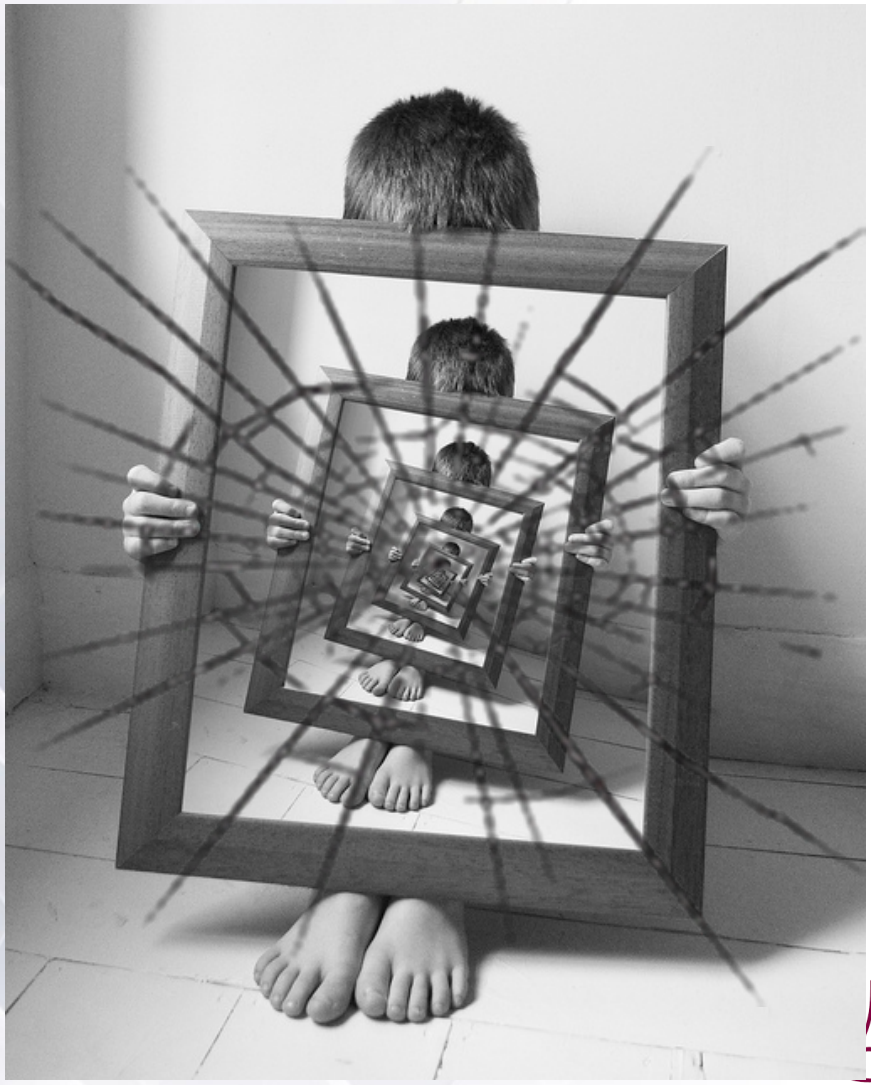
Se perdre avec la récursion ?

TO UNDERSTAND
what ***recursion*** *is*
YOU MUST FIRST
understand recursion

- Non !

Université de Paris

Sortir du paradoxe

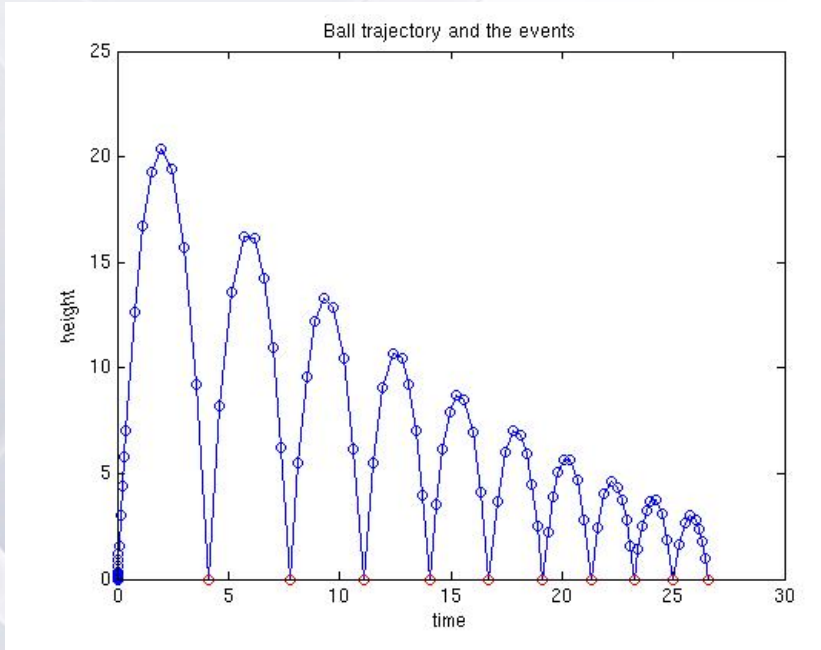


- La récursion termine !

Université de Paris

Il faut penser à une représentation finie !

Raisonner en anticipant la fin



- La dernière poupée existe
- La balle arrête de rebondir

Exemple de récursion en linguistique

- Un fait : dans la construction d'une phrase on peut ajouter un nombre arbitraire de fois le mot de liaison 'de', et écrire :
  la clé de la serrure de la porte de l'appartement de l'immeuble
- La grammaire autorisant cette construction, définit une phrase par :
  - soit un objet précis
  - soit un objet précis suivit de la liaison de puis d'une phrase
- On ne conçoit pas de phrase infinie pour autant :
  - potentiellement arbitrairement longue
  - mais finie

Université de Paris

En mathématiques

- Vous avez vu les preuves par récurrence :
  - prouver $\mathcal{P}(0)$
  - prouver $\mathcal{P}(n) \Rightarrow \mathcal{P}(n+1)$
  - on en déduit que $\mathcal{P}$ est vrai sur $\mathbb{N}$, puis on peut oublier la preuve
  - mais $\mathcal{P}(100)$ se justifie par $\mathcal{P}(99)$ qui se justifie par $\mathcal{P}(98)$ ...
- Vous les avez vu également avec les suites :
  - $u_0 = 5$
  - $u_{n+1} = (u_n)^2 + u_n$ pour tout $n \geq 0$
- En informatique on va :
  - "refaire" la preuve à chaque fois (par l'exécution du calcul)
  - généraliser un peu la notion de récurrence
    - pourquoi partir de 0 ?
    - pourquoi se contenter de $n+1$?
    - affiner le point de vue descendant/ascendant
  - implémenter un mécanisme (pile d'appels - voir CI2 pour les détails)

Université de Paris

En informatique - Une méthode récursive :

1. A la possibilité de faire appel à elle même (éventuellement indirectement)
2. VOUS devez concevoir la suite d'appels en garantissant qu'elle termine !

Fichier Test.java

```
public class Test {
 public static void f ( int n ){
  if ( n <= 10){ // on veut s'assurer que l'exécution termine !
   System.out.println ( n );
   f ( n+1 ); // f se définit en fonction de f
  }
 }
 public static void main(String [] args){
  f(5);
 }
}
```

- Ici la suite des valeurs prises par n est croissante ET majorée

En informatique - Une méthode récursive (autre exemple)

1. A la possibilité de faire appel à elle même (éventuellement indirectement)
2. VOUS devez concevoir la suite d'appels en garantissant qu'elle termine !

Fichier Test.java

```
public class Test {
 public static int question ( int n , int p ){
  if ( n <= 0) return p ;
  if ( p <= 0) return n ;
  return question (n-1 , p-2) + 2;
 }
 public static void main(String [] args){
  question(7,5);
 }
}
```

- Ici la *suite d'appels est "globalement"* décroissante ET minorée
- A suivre ... plus de détails en CI2 ...

Université de Paris

La récursion - Mécanisme

Basé simplement sur le mécanisme connu des appels de fonctions

Rappels important :

- le contrôle du flux. Les instructions sont :
  - exécutées en séquence jusqu'au moment d'un appel de méthode
  - alors, l'exécution courante est suspendue
  - la méthode appelée commence dans un nouvel environnement
  - lorsqu'elle termine, retour à la méthode appelante
- la portée, la visibilité des variables
  - chaque appel de méthode isole les variables actuellement utilisées
  - la nouvelle exécution travaille avec de nouvelles variables
- ces mécanismes sont des rappels de CI2. Reste la difficulté de concevoir vos algorithmes

Révisions amusante :

- Lien : CoderTheTyler Vidéo tout en 5mn
- Lien : BlondieByte Vidéo plus de détails en 30 mn
- Lien (plus avancé)

Université de Paris

Conception d'algorithmes récursifs = Décomposer en problèmes plus petits

C'est le seul secret de la programmation récursive

Conception d'algorithmes récursifs = Décomposer en problèmes plus petits

C'est le seul secret de la programmation récursive

Exercice de style : affichage des entiers de 1 à n

Conception d'algorithmes récursifs = Décomposer en problèmes plus petits

C'est le seul secret de la programmation récursive

Exercice de style : affichage des entiers de 1 à *n*

1. afficher les entiers de 1 à n-1
2. puis afficher n

Université de Paris

Décomposer en problèmes plus petits

C'est le seul secret de la programmation récursive

Exercice de style : affichage des entiers de 1 à *n*

1. afficher les entiers de 1 à n-1
2. puis afficher n

Fichier Test.java (incorrect)

```
public class Test {
 public static void main(String [] args){
  afficheEntiers(10);
 }
 public static void afficheEntiers(int n){
  afficheEntier(n-1);
  System.out.println(n);
 }
}
```

Université de Paris

Décomposer en problèmes plus petits

C'est le seul secret de la programmation récursive

Exercice de style : affichage des entiers de 1 à *n*

1. afficher les entiers de 1 à n-1
2. puis afficher n seul

Fichier Test.java

```
public class Test {
 public static void main(String [] args){
  afficheEntiers(10);
 }
 public static void afficheEntiers(int n){
  if (n<=0) return;          // ne pas oublier la condition de terminaison
  afficheEntier(n-1);
  System.out.println(n);
 }
}
```

- Illustré par un arbre des appels

Université de Paris

Décomposer en problèmes plus petits

C'est le seul secret de la programmation récursive

Exercice de style : affichage des entiers de 1 à *n*

1. afficher les entiers de 1 à n-1
2. puis afficher n seul

Fichier Test.java

```
public class Test {
 public static void main(String [] args){
  afficheEntiers(10);
 }
 public static void afficheEntiers(int n){
  if (n<=0) return;          // ne pas oublier la condition de terminaison
  afficheEntier(n-1);
  System.out.println(n);
 } // Bien sûr vous saviez déjà l'écrire sans récursion,
  // dans un autre style plus rassurant peut être
 public static void afficheEntierVersionNonRecursive(int n){
  for(int i=1;i<=n;i++) System.out.println(i);
 }
```

Récursion, simple figure de style ?

Quelques résultats de CI2

- tout ce qui est fait par une boucle peut être fait en récursif ?
- tout ce qui est fait en récursif peut être fait avec des boucles ?
- Exact : du point de vue de l'expressivité la récursion n'apporte rien

Université de Paris

Récursion, simple figure de style ?

Quelques résultats de CI2

- tout ce qui est fait par une boucle peut être fait en récursif ?
- tout ce qui est fait en récursif peut être fait avec des boucles ?
- Exact : du point de vue de l'expressivité la récursion n'apporte rien

Exercice de style - Jeu : le compte est bon

- Un nombre $N \in [1..999]$ est choisi au hasard
- ainsi que 6 valeurs dans $\{1, 2, ..., 9, 10, 25, 50, 75, 100\}$
- Existe t'il une combinaison des 6 valeurs et des opérations $+, -, *, /$ qui donnent N ? (répondre par oui ou non)

Exemple

- essayez avec $N = 888$
- et les valeurs que vous pouvez utiliser une fois : $\{100, 2, 75, 3, 1, 10\}$
- voir solveur, <ici> ... Essayons donc d'écrire l'algorithme ...

Récursion, simple figure de style ?

Exemple

- $N = 888$
- et des valeurs à utiliser $\{100, 2, 75, 3, 1, 10\}$
- voir solveur, <ici> ... Essayons donc d'écrire l'algorithme ...

- Certains vont penser à chercher les diviseurs de 888, il y a :
  - 8, donc 2 et 4
  - 3 car une propriété dit que si la somme des chiffre est divisible par 3 le nombre l'est
  - et donc 888 est divisible aussi par 6, 12 et 24

Université de Paris

Récursion, simple figure de style ?

Exemple

- $N = 888$
- et des valeurs à utiliser $\{100, 2, 75, 3, 1, 10\}$
- voir solveur, <ici> ... Essayons donc d'écrire l'algorithme ...

- Certains vont penser à chercher les diviseurs de 888, il y a :
  - 8, donc 2 et 4
  - 3 car une propriété dit que si la somme des chiffre est divisible par 3 le nombre l'est
  - et donc 888 est divisible aussi par 6, 12 et 24
- chercher les diviseurs, revient à :
  - envisager que la dernière opération soit une multiplication
  - décomposer en problèmes plus petits
- On cherche presque naturellement une solution récursive...
  En voici une :

Université de Paris

Récursion, simple figure de style ?

Exercice de style - Jeu : le compte est bon

- Un nombre $N \in [1..999]$ est choisi au hasard
- ainsi que 6 valeurs dans $\{1, 2, ..., 9, 10, 25, 50, 75, 100\}$
- Existe t'il une combinaison des 6 valeurs et des opérations $+, -, *, /$ qui donnent N ? (oui ou non)

Solution mélangeant boucles et récursion

- pour toute paire $(i, j), i \neq j$ dans le tableau T des $n = 6$ valeurs
- pour toute opération *opp* dans $+, -, *, /$
- calculer $x = T[i]$ *opp* $T[j]$
- si $x == N$ terminé : répondre oui
- construire une copie T' à partir de T : enlevez $T[i]$ et $T[j]$, ajoutez x
- résoudre le problème à $n - 1$ valeurs pour T' et N
  (sauf si T' est de taille 1, alors répondre non)

Récursion, simple figure de style ?

Exercice de style - Jeu : le compte est bon

- Un nombre $N \in [1..999]$ est choisi au hasard
- ainsi que 6 valeurs dans $\{1, 2, ..., 9, 10, 25, 50, 75, 100\}$
- Existe t'il une combinaison des 6 valeurs et des opérations $+, -, *, /$ qui donnent N ? (oui ou non)

Solution mélangeant boucles et récursion

- pour toute paire $(i, j), i \neq j$ dans le tableau T des $n = 6$ valeurs
- pour toute opération *opp* dans $+, -, *, /$ $\Rightarrow$ 3 boucles imbriquées
- calculer $x = T[i]$ *opp* $T[j]$
- si $x == N$ terminé : répondre oui $\Rightarrow$ condition arrêt
- construire une copie T' à partir de T : enlevez $T[i]$ et $T[j]$, ajoutez x
- résoudre le problème à $n - 1$ valeurs pour T' et N $\Rightarrow$ appel récursif
  (sauf si T' est de taille 1, alors répondre non) $\Rightarrow$ condition arrêt

Exercice de style - Jeu : le compte est bon

- Un nombre $N \in [1..999]$ est choisi au hasard
- ainsi que 6 valeurs dans $\{1, 2, ..., 9, 10, 25, 50, 75, 100\}$
- Existe t'il une combinaison des 6 valeurs et des opérations $+, -, *, /$ qui donnent N ? (oui ou non)

Solution mélangeant boucles et récursion

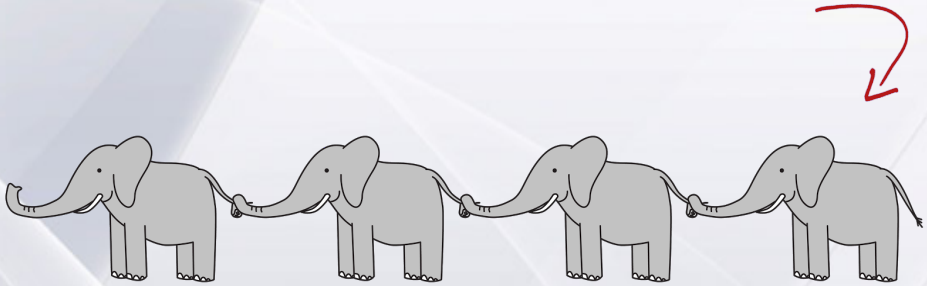
- pour toute paire $(i, j), i \neq j$ dans le tableau T des $n = 6$ valeurs
- pour toute opération *opp* dans $+, -, *, /$ $\Rightarrow$ 3 boucles imbriquées
- calculer $x = T[i]$ *opp* $T[j]$
- si $x == N$ terminé : répondre oui $\Rightarrow$ condition arrêt
- construire une copie T' à partir de T : enlevez $T[i]$ et $T[j]$, ajoutez x
- résoudre le problème à $n - 1$ valeurs pour T' et N $\Rightarrow$ appel récursif
  (sauf si T' est de taille 1, alors répondre non) $\Rightarrow$ condition arrêt

Code complet et présentation pratique du jeu sont laissés en exercice ...

Université de Paris

Applications aux listes

Par leurs formes elles se prêtent à l'écriture de fonctions récursives



- Dans notre implémentation la forme récursive concerne les cellules

Fichier Cellule.java

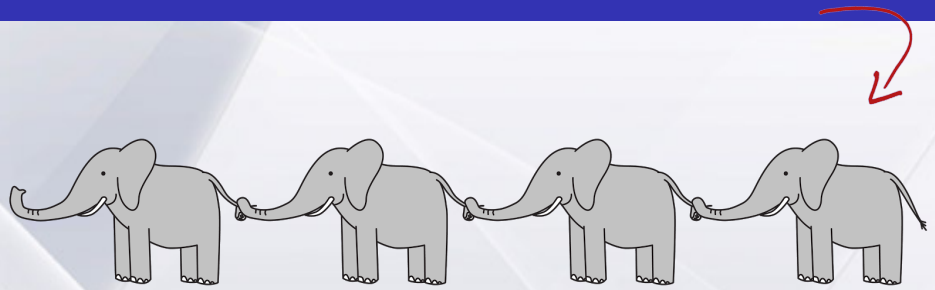
```
public class Cellule{
 private E content;
 private Cellule next;
 ...
}
```

- On préfère un contexte non statique.
  Le passage à la dimension inférieure : `this.next`
- La terminaison se fait sur `null` : plus précisément `this.next==null`

Université de Paris

Applications aux listes

Par leurs formes elles se prêtent à l'écriture de fonctions récursives



- Dans notre implémentation la forme récursive concerne les cellules

Fichier Cellule.java

```
public class Cellule{
 private E content;
 private Cellule next;
 ...
}
```

- On préfère un contexte non statique.
  Le passage à la dimension inférieure : `this.next`
- La terminaison se fait sur `null` : plus précisément `this.next==null`

certains étudiants testent si `this` est `null` c'est un contresens total

Implémentation récursive - même cahier des charges :

On va tout refaire en récursif !
L'idée est que sur cette structure simple vous maîtrisiez cet outil puissant.

Fichier ListIP2.java – écrite pour des éléments de type E

```
public interface ListIP2 { // position starts at 0
 void add(E x); // Appends the element to the end
 void add(int index, E x); // Inserts the element at the position
 void clear(); // Removes all of the elements from this list
 boolean contains(E x); // Returns true if the list contains the element
 E get(int index); // Returns the element at the specified position
 int indexOf(E x); // Returns the index of the first occurrence of the
     element, or -1 if not there
 boolean isEmpty(); // Returns true if this list contains no elements
 int lastIndexOf(E x); // see indexOf
 E remove(int index); // Removes the element at the specified position
 boolean remove(E x); // Removes the first occurrence of the element
 E set(int index, E x); // Replaces the element at a position
 int size(); // Returns the number of elements in this list
}
```

Modèle à 3 classes

Fichier E.java - le contenu

```
public class E{
 // peu importe
}
```

Fichier Cellule.java - rôle auxiliaire, précise la nature des liaisons

```
public class Cellule{
 private E content;
 private Cellule next;
}
```

Fichier MaListe.java

```
public class MaListe implements ListIP2{
 private Cellule first;
}
```

Université de Paris

Implémentation récursive

Les méthodes simples (celles sans répétitions) restent identiques

Fichier ListIP2.java – écrite pour des éléments de type E

```
public interface ListIP2 { // position starts at 0
 void add(E x); // Appends the element to the end
 void add(int index, E x); // Inserts the element at the position
 void clear(); // Removes all of the elements from this list
 boolean contains(E x); // Returns true if the list contains the element
 E get(int index); // Returns the element at the specified position
 int indexOf(E x); // Returns the index of the first occurrence of the
     element, or -1 if not there
 boolean isEmpty(); // Returns true if this list contains no elements
 int lastIndexOf(E x); // see indexOf
 E remove(int index); // Removes the element at the specified position
 boolean remove(E x); // Removes the first occurrence of the element
 E set(int index, E x); // Replaces the element at a position
 int size(); // Returns the number of elements in this list
}
```

Université de Paris

Fichier MaListe.java

```
public class MaListe implements ListIP2{
 private Cellule first;
 public MaListe(){
  this.first=null;
 }
 public boolean isEmpty() {
  return ( this.first==null );
 }
 public void clear() {
  this.first=null;
 }
}
```

Université de Paris

Implémentation récursive

Les méthodes qui laissent la forme de la liste inchangée se ressembleront beaucoup (basées sur un parcours simple)

Fichier ListIP2.java – écrite pour des éléments de type E

```
public interface ListIP2 { // position starts at 0
 ...
 void add(E x); // Appends the element to the end
 void add(int index, E x); // Inserts the element at the position
 boolean contains(E x); // Returns true if the list contains the element
 E get(int index); // Returns the element at the specified position
 int indexOf(E x); // Returns the index of the first occurrence of the
     element, or -1 if not there
 int lastIndexOf(E x); // see indexOf
 E remove(int index); // Removes the element at the specified position
 boolean remove(E x); // Removes the first occurrence of the element
 E set(int index, E x); // Replaces the element at a position
 int size(); // Returns the number of elements in this list
}
```

Université de Paris

Cas des parcours simples. La liste est vide ou non ?

Sinon on délègue aux cellules, endroit où on résoudra avec un style récursif

Fichier MaListe.java

```
public boolean contains(E x){
 if (this.isEmpty()) return false;
 else // ce else est facultatif
 return this.first.contains(x);
}
public int size(){
 if (this.isEmpty()) return 0;
 return this.first.size();
}
public E get(int index){
 if (this.isEmpty()) return null;
 return this.first.get(index);
}
public int indexOf(E x){
 if (this.isEmpty()) return -1;
 return this.first.indexOf(x);
}
```

Fichier MaListe.java

```
...
public int lastIndexOf(E x){
 if (this.isEmpty()) return -1;
 return this.first.lastIndexOf(x);
}
public E set(int index, E x){
 if (this.isEmpty()) return null;
 return this.first.set(index,x);
}
```

Université de Paris

Fichier Cellule.java

```
public boolean contains(E x){
 if (content==x) return true;
 if (next==null) return false
 return next.contains(x);
}
public int size(){
 if (next==null) return 1;
 return 1+next.size();
}
public E get(int index){
 if (index==0) return content;
 if (next==null) return null;
 return next.get(index-1);
}
public int indexOf(E x){
 if (content==x) return 0;
 if (next==null) return -1;
 int aux = next.indexOf(x);
 if (aux==-1) return -1
 return 1+aux;
}
```

Fichier Cellule.java

```
public int lastIndexOf(E x){
if (content==x) {
 if (next==null) return 0;
 return Math.max(0,
      1+next.lastIndexOf(x)
 );
}
if (next==null) return -1;
int aux = next.lastIndexOf(x);
if (aux==-1) return -1;
else return 1+aux;
}
public E set(int index, E x){
 if (index==0){
  E tmp=content;
  content=x;
  return tmp;
 }
 if (next==null) return null;
 return next.set(index-1,x);
}
```

Implémentation récursive

Pour les méthodes qui changent la forme de la liste, il faut faire plus attention

Fichier ListIP2.java – écrite pour des éléments de type E

```
public interface ListIP2 { // position starts at 0
 ...
 void add(E x); // Appends the element to the end
 void add(int index, E x); // Inserts the element at the position
 E remove(int index); // Removes the element at the specified position
 boolean remove(E x); // Removes the first occurrence of the element
}
```

- traitons par exemples deux de ces 4 méthodes
  (les deux autres sont laissées en exercice)

Université de Paris

Fichier MaListe.java

```
public class MaListe implements ListIP2{
 public void add(int index, E x){
  if ( (index<=0) || (first==null) ) first=new Cellule(x , first);
  else first.add(index , x); // avec index >=1
 }
}
```

Fichier Cellule.java

```
public class Cellule{
 private E content;
 private Cellule next;
 public Cellule(E x, Cellule suiv){
  this.content=x;
  this.next=suiv;
 }
 public void add(int index, E x){
  if ( (next !=null) && (index > 1) ) next.add( index-1 , x);
  else this.next=new Cellule(x , next);
 }
}
```

Fichier MaListe.java

```
public class MaListe implements ListIP2{
 public boolean remove(E x){
  if ( first.getContent()==x ) { // accesseur supposé écrit
   first=first.getNext(); // accesseur supposé écrit
   return true;
  }
  return first.remove(x);
 }
}
```

Fichier Cellule.java

```
public class Cellule{
 ...
 public boolean remove(E x){ // rq : on a l'invariant this.content != x
  if (next==null) return false;
  if (next.content != x) return next.remove(x);
  // reste le cas ou next.content == x
  next=next.next;
  return true;
 }
}
```

Conclusion

- On a changé plusieurs fois de style de programmation :
  - style statique (vu en IP1)
  - style dynamique (c.à.d non statique)
  - parfois on préfère utiliser des itérations `while`
  - parfois on préfère utiliser des itérations `for`
  - à présent : vous avez la possibilité d'écrire des solutions récursives
- Un exemple (*le compte est bon*) illustre que la maîtrise d'un style permet d'orienter la recherche d'une solution.
  (Et je vous met au défi d'en trouver une non récursive !)
- Il vous faudra acquérir ces compétences dans les semaines à venir !

Université de Paris

Annonces

- Les deux cours que nous venons de faire étaient denses.
- Les deux suivants seront simplement des variations, pour vous permettre d'assimiler.
- Mais c'est bien maintenant qu'il vous faut fournir un travail !

Partiel samedi 20/03

- Définitions de classes
- Méthodes statique/dynamique
- Modificateurs `static`, `final`, `private`, `public`
- Politique getters/setters, notion d'interface
- Listes simplement chaînées et opérations liées
- Récursion (formes simples)
- Autres formes de listes (prochain cours)
- Nous aurons une séance de révisions

Exercices ...

1. Rappelez les classes définissant une liste chaînée d'éléments E
2. Ecrivez une méthode non statique qui teste s'il y a bien unicité de tous les éléments dans une liste.
3. Ecrivez une méthode non statique qui simplifie les doublons, en n'en laissant qu'une occurrence
4. Reprenez vos solution en essayant d'écrire des versions récursives/non récursives

Vous pouvez reprendre également des exercices de hackerrank

Université de Paris