Structures de données: les listes, piles..

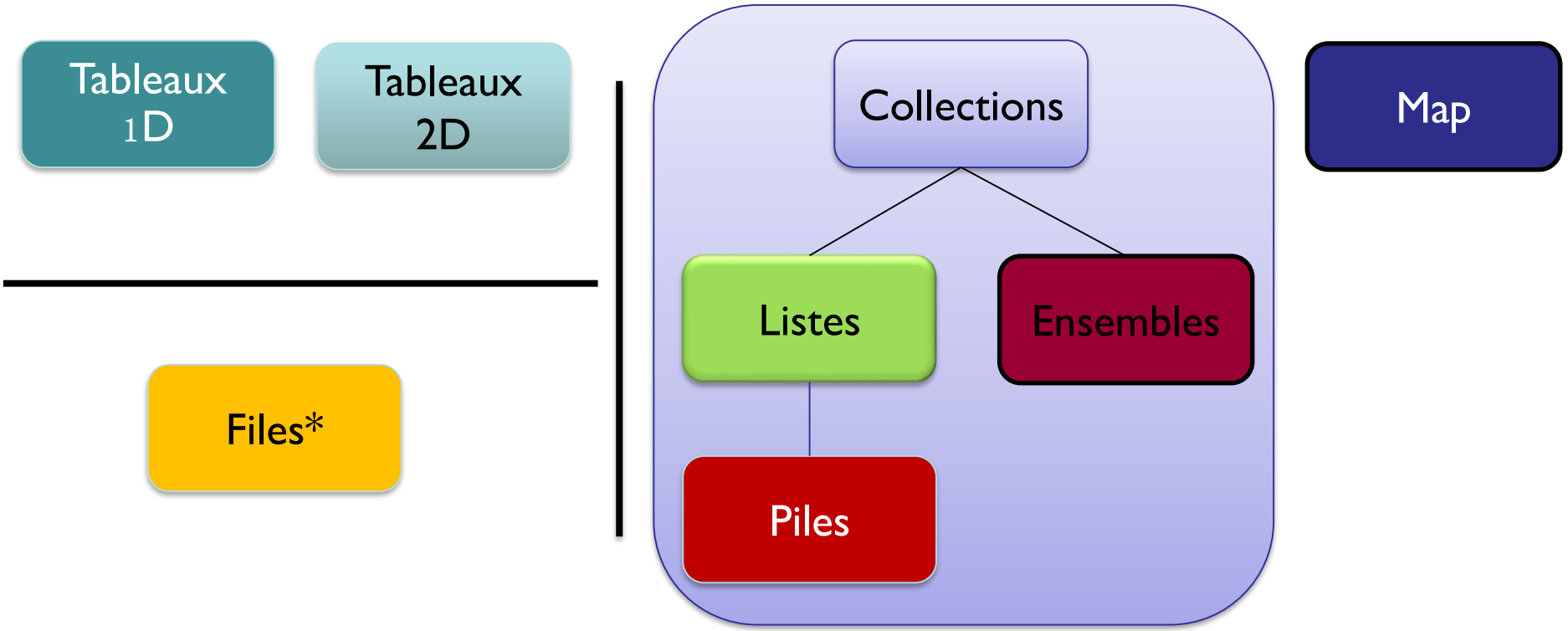
Objectifs du cours

- Présenter les structures de données essentielles en algorithmique que sont les:

Listes	Ensembles
Piles	Map
Files	

- Leur principe de fonctionnement, et leur manipulation en Java.

Objectifs du cours



* N'est pas fournie par Java.

Les collections
Création 1

Je définis quelle structure de données je souhaite utiliser

```
NomCol<Type> nom= new NomCol<Type>();
```

Je dis à Java que tous les éléments de la collection sont du type Type (entiers, nombres à virgule…)

Je nomme ma collection

Les collections
Création 1

- Type peut prendre les valeurs suivantes

Si je veux des	Je mets le mot
`int`	Integer
`double`	Double
`boolean`	Boolean
`char`	Character
`String`	String

Les collections
Création 2

- NomCol peut prendre les valeurs suivantes

Si je veux	Je mets
`une liste`	LinkedList ArrayList

Création d'une liste d'entiers:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
ArrayList<Integer> maListe= new ArrayList<Integer>();
```

Les collections
Création 2

- NomCol peut prendre les valeurs suivantes

Si je veux	Je mets
`une liste`	LinkedList ArrayList
`une pile`	Stack

Création d'une pile d'entiers:

```
Stack<Integer> maPile= new Stack<Integer>();
```

Les collections
Création 2

- NomCol peut prendre les valeurs suivantes

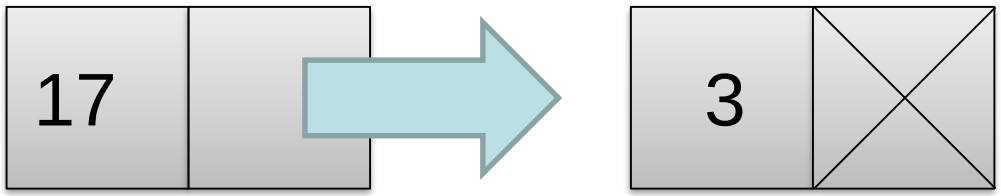
Si je veux	Je mets
`une liste`	LinkedList ArrayList
`une pile`	Stack
`un ensemble`	HashSet

Création d'un ensemble d'entiers:

```
HashSet<Integer> monSet= new HashSet<Integer>();
```

Les collections
Modèle théorique

- Composée de maillons ordonnées
- Chaque maillon est un couple :
  - ✓Variable
  - ✓Pointeur vers le maillon suivant



- Donc taille infinie (contrairement aux tableaux) : il suffit de rajouter un maillon

Les collections : listes
Différentes impl.

Si je veux	Je mets
`une liste`	LinkedList ArrayList

- Les LinkedList Java fonctionnent quasiment sur le modèle théorique : ce sont des listes (doublement) chainées, et la queue de la liste pointe vers sa tète (= anneau).
- Les ArrayList reposent sur des tableaux automatiquement redimensionnés au besoin.

Les listes

Méthodes

boolean	**add(Object o)**	Ajouter un objet à la collection.
void	**add(int index, Object o)**	Ajouter un objet à la liste en position donnée.
Object	**get(int index)**	Retourne l'élément à la position index
boolean	**isEmpty()**	Tester si la collection est vide.
boolean	**contains(Object o)**	Tester si la collection contient l'objet indiqué.
int	**size()**	Retourne la taille de la collection
void	**clear()**	Vider la collection
Object	**remove(int index)**	Retourne l' objet retiré à la position index de la collection.
boolean	**set(int index, Object element)**	Remplace l'élément à la position index.

Les listes

Méthodes

boolean	**add(Object o)**	Ajouter un objet à la collection.
void	**add(int index, Object o)**	Ajouter un objet à la liste en position donnée.

Exemple :

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(0,10)
```

Les listes

Méthodes

Object	**get(int index)**	Retourne l'élément à la position index

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);

int valeurRecuperee= maListe.get(0);
```

Les listes

Méthodes

boolean	**isEmpty()**	Tester si la collection est vide.

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
boolean res= maListe.isEmpty();
```

Les listes

Méthodes

boolean **contains(Object o)** Tester si la collection contient l'objet indiqué.

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(18);
maListe.add(9);
boolean res= maListe.contains(20);
```

Les listes
Méthodes

int	**size()**	Retourne la taille de la collection

Exemple :

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(8);
maListe.add(20);
maListe.add(18);
maListe.add(9);

int taille= maliste.size();
```

Les listes

Méthodes

void	**clear()**	Vider la collection

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(17);
int res= maListe.size();
System.out.println(res);    //Affichera
maListe.clear();
System.out.println(maListe.size()); //Affichera
```

Les listes
Méthodes

Object **remove(int index)** Retourne l' objet retiré à la position index de la collection.

Exemple:

```
LinkedList<Integer> maListe= new
LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(20);
int val= maListe.remove(2);
maListe.remove(1);//peut être utilisée comme ça
```

Les listes

Méthodes

`boolean set(int index, Object element)`	Remplace l'élément à la position index.

Exemple:

```
LinkedList<Integer> maListe= new LinkedList<Integer>();
maListe.add(8);
maListe.add(20);
maListe.add(20);
maListe.set(0,9);
```

Liste
Exo 1

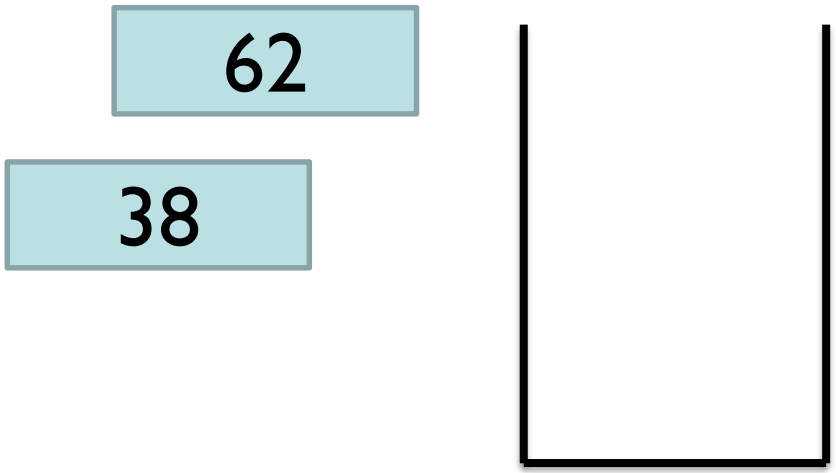
- Dans un programme principal, dans lequel vous créez une liste d'entiers appelée « toto ».
- Remplissez là avec 10 entiers générés aléatoirement.
- Dans un premier temps affichez la en entier.
- Dans un second temps affichez les éléments d'indice paire.

Les collections : piles
Modèle théorique

- Dernier arrivé, premier sorti

  "Last In, First Out" (LIFO)

- Structure ordonnée et homogéne



Les collections : piles
Méthodes de piles

- Les méthodes qui permettent de manipuler les piles sont :

Object	**push(Object o)**	Ajouter au sommet de la pile
Object	**pop()**	Retirer le sommet de la pile
boolean	**isEmpty()**	Tester si la collection est vide.
boolean	**contains(Object o)**	Tester si la collection contient l'objet indiqué.
int	**size()**	Retourne la taille de la collection

Les collections : piles
Méthodes de piles

Object	**push(Object o)**	Ajouter au sommet de la pile

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
```

Les collections : piles
Méthodes de piles

Object	**pop()**	Retirer le sommet de la pile

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
```

Les collections : piles
Méthodes de piles

boolean	**isEmpty()**	Tester si la collection est vide.

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
boolean res=maPile.isEmpty();
```

Les collections : piles
Méthodes de piles

boolean	**contains(Object o)**	Tester si la collection contient l'objet indiqué.

Exemple:

```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
boolean res1=maPile.contains("BONJOUR");
```

Les collections : piles
Méthodes de piles

int	**size()**	Retourne la taille de la collection

Exemple:

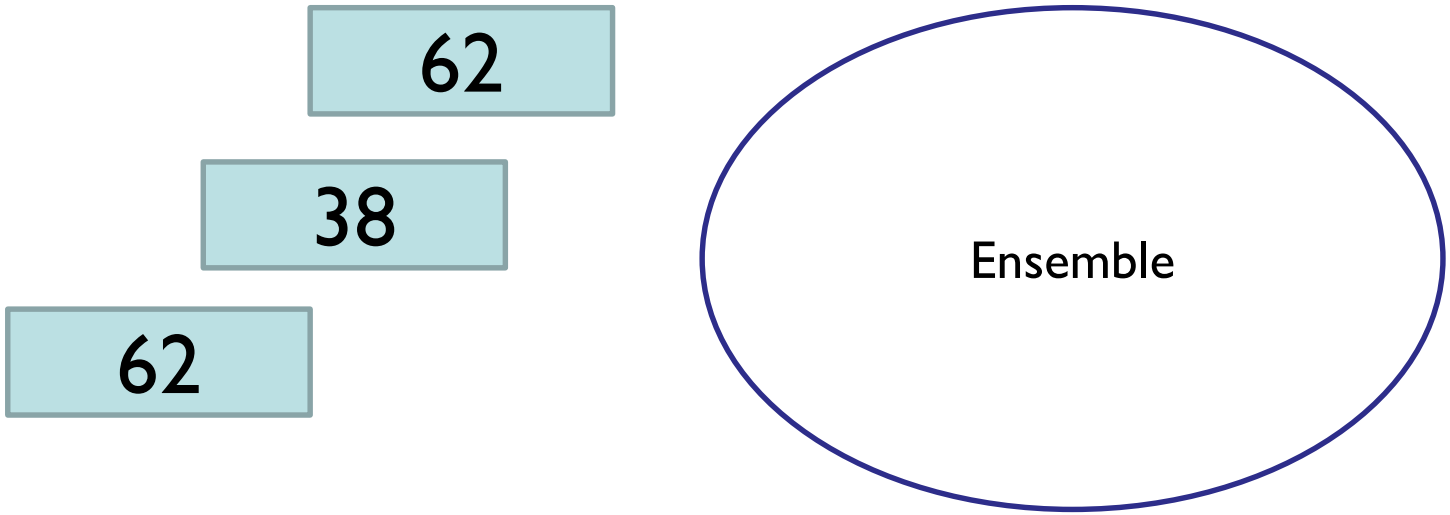
```
Stack<String> maPile= new Stack<String>();
maPile.push("Bonjour");
maPile.push("hei");
maPile.push("ça va?");
mapile.pop();
boolean res1=maPile.contains("BONJOUR");
int taille= maPile.size();
```

Les collections : ensemble
Modèle théorique

- Ensemble de valeurs dont l'unicité est garantie.

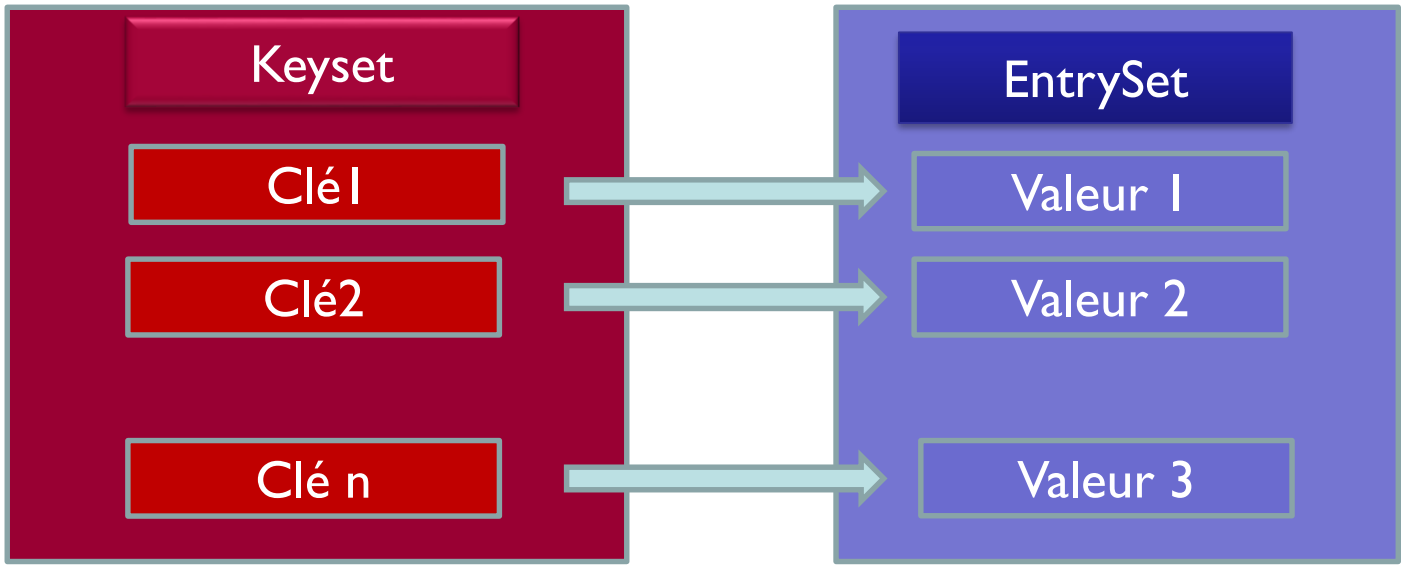
Si je veux	Je mets
`un ensemble`	HashSet

- Non ordonnée



Les maps ou table de hachage

- Structure de données qui fonctionne selon le principe de paires Clé / Valeur
- Implémentation : HashMap



Les maps : HashMap Méthodes

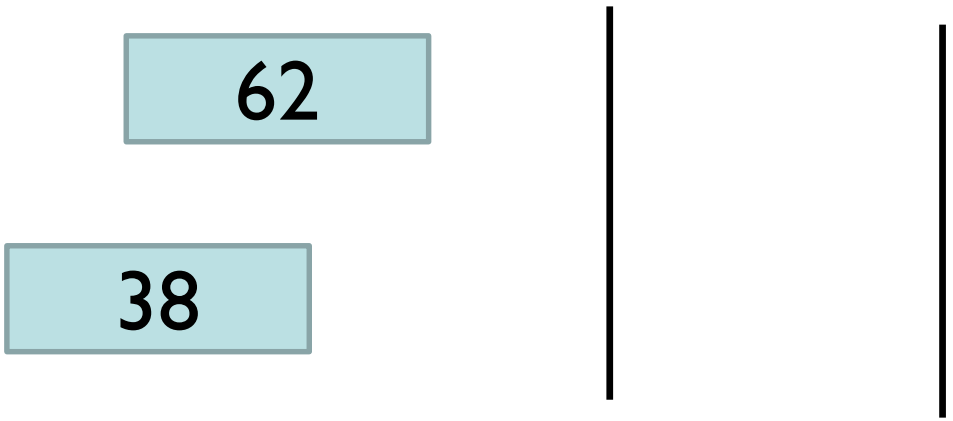
boolean	put(K key,V value)	Ajouter un objet à la collection.
boolean	isEmpty()	Tester si la collection est vide.
boolean	containsKey(Object o)	Tester si la clé existe
boolean	containsValue(Object o)	Tester si la valeur existe
int	size()	Retourne la taille de la collection
void	clear()	Vider la collection
boolean	remove(Object o)	Retirer la clé + valeur si la clé existent

Les files ou queue

- Premier arrivé, premier sorti

  "First In, First Out" (FIFO)

- *Ne sont pas fournies par Java*
- Ex : modélisation des files d'attente



Les piles
exercice

- Ecrivez une méthode qui prend une pile en paramètre et retourne le nombre d'élèments qu'elle contient.

- Attention:
  - ✓Ne pas détruire la pile en question
  - ✓Ne pas faire appel à la méthode size() qui existe déjà, faites votre propre fonction