

A3P/IPO 2023/2024

Cours 6

© Denis BUREAU, ESIEE Paris

Sommaire

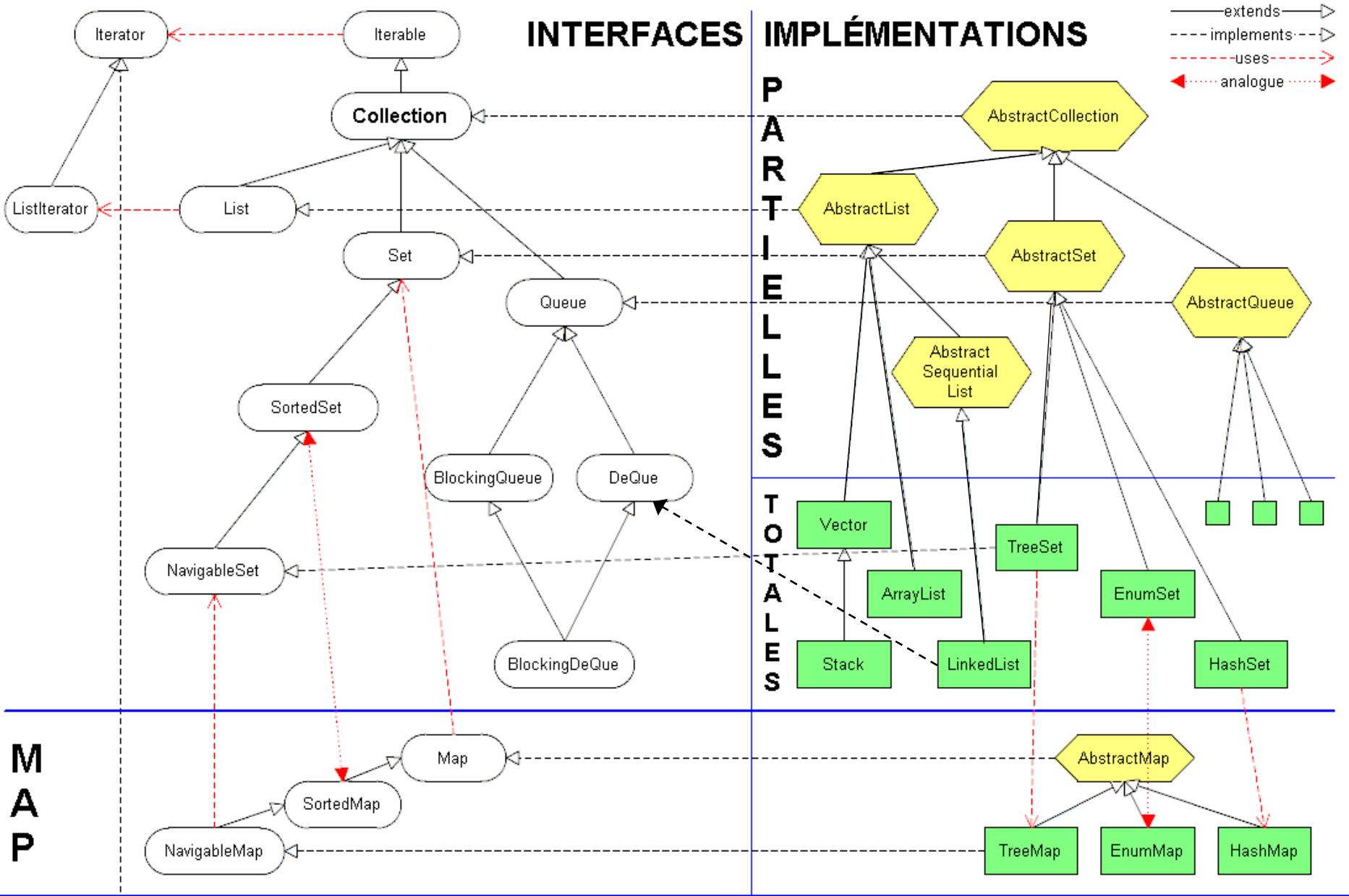
1. Le framework *Collections*
2. La hiérarchie
3. Le type des éléments
- 4. La généricité**
- 5. Les parcours** (simple et non simple)
6. Les itérateurs
7. Scanner
8. HashMap **et** Set

1. Framework Collections

- = ensemble de :
 - classes abstraites ou non,
 - d'interfaces,
 - et d'algorithmes.
- Exemples :
 - `List`, `ArrayList`, `LinkedList`,
 - `Stack`, `Set`, `HashMap`
- dans `java.util` => nombreux `import`

2. Hiérarchie

- Interface la plus haute = **Collection** :
size, add, iterator, contains, xxxAll, ...
- **List** extends Collection :
+ get, set, listIterator
- **AbstractList** implements List :
+ equals, toString
- **ArrayList** / **LinkedList** extends AbstractList :
performances différentes selon opérations
- **Collections** :
méthodes statiques telles que shuffle

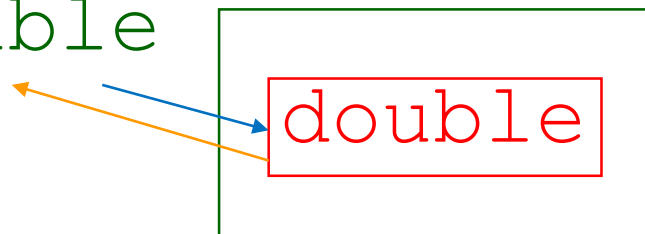


3. Type des éléments

- Par rapport aux tableaux :
 - seulement des objets
 - + extensible
 - + propriétés/méthodes

- Types primitifs interdits
=> classes enveloppes, auto-(un)boxing

Double



```
Double vD = 1.2;  
... Math.sqrt(vD) ...
```

4. Généricité

- généricité systématique => `Coll<TypeE>`
(à la déclaration *ET* à la création)
- `List<String> vL;`
`vL = new ArrayList<String>();`
`...methode(final Stack<Room> pS)`
`public List<Integer> fonc() {...`
- `HashMap<String, Item> vH;`

5.1. Les parcours 1/2

- **Simple** (*et à partir du JDK 5*) =
du début à la fin,
et sans ajout/suppression d'élément,
et sans synchronisation (ou parallélisme)
avec une autre collection

- boucle *for each* :

```
for ( TypeE vElt : vMaCollection ) {  
    System.out.println(vElt.toString());  
} // for each
```

- **Fonctionne aussi sur les tableaux !**

5.2. Les parcours 2/2

- **Non simple** (*ou avant le JDK 5*) =
ne commençant pas au début,
ou avec ajout ou suppression d'élément(s),
ou progressant en parallèle
avec une autre collection
- Utilisation d'un itérateur
Iterator<TypeE> : pas new, iterator()
permet de parcourir une
Collection<TypeE>

6.1. Itérateur

- fournit 3 méthodes **sans paramètre** :
 - **hasNext()** (*peut retourner faux dès la 1^{ère} fois*)
 - **next()** (*retourne directement un TypeE*)
 - **remove()** (*toujours précédée d'un next !*)

```
Iterator<TypeE> vIt =  
    vMaCollection.iterator();  
while ( vIt.hasNext() ) {  
    if ( estMauvais ( vIt.next() ) )  
        vIt.remove(); } // while
```

(toujours la séquence

hasNext() -> next() -> remove() !)

6.2. ListIterator

- Est une sorte d' `Iterator` (*→ les 3 méthodes*)
- Complété pour les `List` par les méthodes
 - `set(TypeE pE)` :
remplace l'élément courant par `pE`
 - `add(TypeE pE)` :
ajoute `pE` juste après l'élément courant
- `ListIterator<TypeE> listIterator()`
permet de parcourir une `List<TypeE>`

7. Scanner

- **Classe du JDK =>** `import java.util.Scanner;`
- **1^{ère} utilisation : lire au clavier =>**
`vSc1 = new Scanner(System.in);`
`String vLigne = vSc1.nextLine();`
- **2^{ème} utilisation : découper une chaîne =>**
`vSc2 = new Scanner(vLigne);`
`String vMot = vSc2.next();`
- **3^{ème} utilisation : extraire des nombres =>**
`int vI = vSc1ou2.nextInt();`
`double vD = vSc1ou2.nextDouble();`
- **4^{ème} utilisation : lire dans des fichiers de texte**

8.1. HashMap

- HashMap \approx « tableau associatif » qui contient un ensemble d'associations clé \rightarrow valeur
- Classe du JDK \Rightarrow `import java.util.HashMap;`
- Plus général qu'un tableau classique qui impose le type `int` pour les indices (clé, valeur) \Rightarrow 2 types objets
- Donc **pas** `typeValeurs[]` mais
`HashMap<typeClés, typeValeurs> vMaHM;`
- `vMaHM.put(maClé, maValeur);`
- `quelleValeur = vMaHM.get(maClé);`

8.2. HashMap : exemple

- Plutôt que de stocker les sorties d'une pièce dans un tableau de 4 Rooms :
tab[0] serait le nord ? **tab[2] l'est ou l'ouest ?**
- Associons chaque sortie Room à la direction "North" ou "South" ...
- Donc dans la classe Room : un attribut
`HashMap<String, Room> aSorties;`
- `aSorties.put("North", vCuisine);`
- `quelleRoom = aSorties.get("North");`
- **null** si la clé n'est pas présente

8.3. Set, keySet, for each

- `keySet` est une fonction de `HashMap` qui retourne l'ensemble des clés :

```
??? = vMaHM.keySet();
```

- Donc pour `aSorties`, un ensemble de `String`, classe du JDK : `Set<String> vMesClés;`

- On peut facilement parcourir tous ses éléments :

```
for ( String vS : vMesClés ) {  
    S.o.p( vS );  
} // appelée boucle for each
```