D[i] | Département Informatique

UNIVERSITÉ TOULOUSE III PAUL SABATIER — Université de Toulouse

Faculté Sciences et Ingénierie

Programmation Orientée Objet

Mathieu RAYNAL

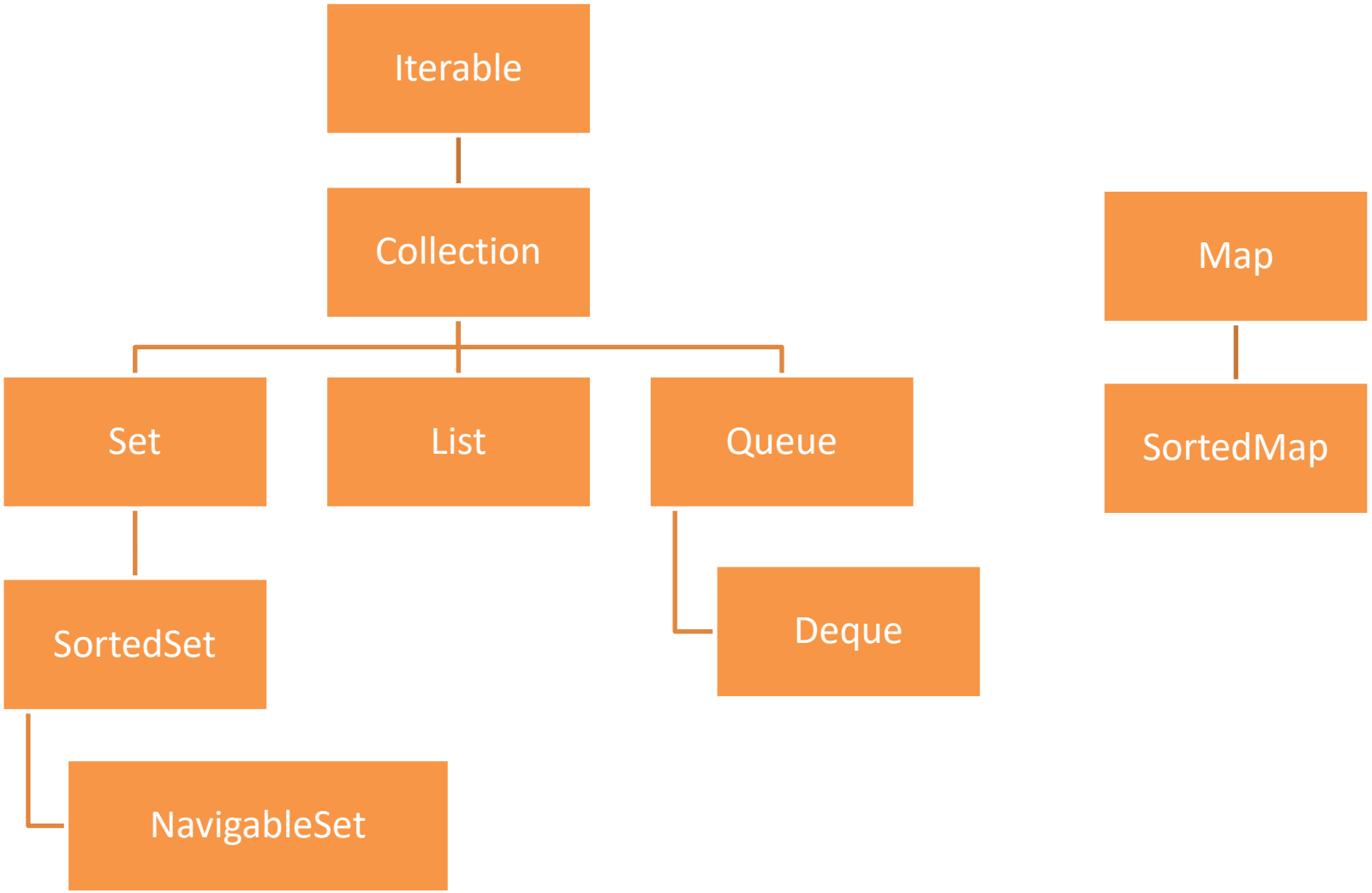
mathieu.raynal@irit.fr

http://www.irit.fr/~Mathieu.Raynal

A quoi ça sert ?

- Stocker des données en mémoire
- Ordonner / Classer
- Les structures peuvent être
  - Statiques
  - Dynamiques

Une histoire d'Interface …



L'interface Iterable

- Une seule méthode

```
Iterator<E> iterator()
```

- ***Iterator*** est aussi une interface qui contient 3 méthodes

```
boolean hasNext()
E next()
void remove()
```

L'interface Collection

- Pas d'association clé/valeur

- Ses principales méthodes

```
boolean add(E e)
boolean addAll(Collection<? extends E> c)
boolean contains(Object o)
boolean equals(Object o)
boolean isEmpty()
Iterator<E> iterator()
boolean remove(Object o)
int size()
Object[] toArray()
```

L'interface List

- Classes qui l'implémentent : LinkedList, ArrayList, Vector
- Ses caractéristiques
  - L'ordre a une importance
  - Accès aux éléments via un indice
  - Accepte les doublons

- Ses principales méthodes

```
void add(int index, E element)
E get(int index)
int indexOf(Object o)
int lastIndexOf(Object o)
E remove(int index)
E set(int index, E element)
```

La classe ArrayList

- Tableau d'objets

```
ArrayList<Integer> liste = new ArrayList<Integer>();
```

  - Ajout d'un élément

```
liste.add(12);
```

  - Accéder à un élément

```
int n = liste.get(i);
```

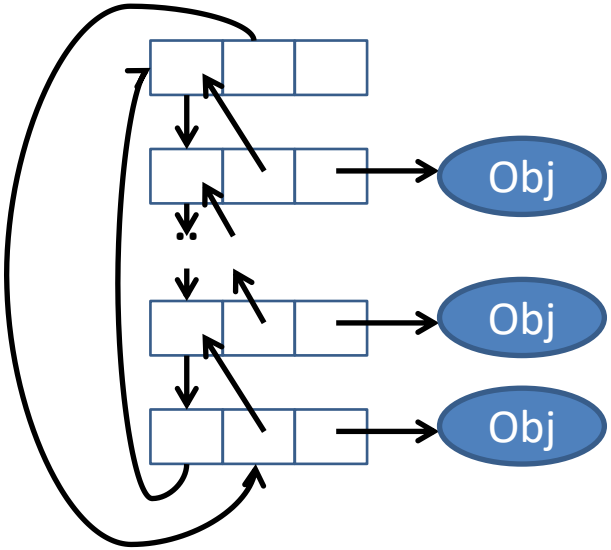
  - Parcourir l'ensemble des éléments

```
for(int i=0;i<liste.size();i++)
   System.out.print(liste.get(i)+" ");

for(Integer entier : liste)
   System.out.print(entier+" ");
```

La classe LinkedList

- Liste chainée
  - Doublement : accès à l'élément précédant et suivant
  - Circulaire :
    - accès au dernier élément depuis le premier
    - accès au premier élément depuis le dernier



- Mêmes méthodes que ArrayList + …
  - Accès au premier

```
liste.addFirst(12);
```

```
int n = liste.getFirst();
```

  - ou dernier élément

```
liste.addLast(12);
```

```
int n = liste.getLast();
```

L'interface Set

- Classes qui l'implémentent : HashSet, TreeSet
- Pas de méthode spécifique par rapport à **Collection**
- Caractéristiques
  - L'ordre n'a pas d'importance
  - Accepte les doublons

L'interface SortedSet

- Trie automatiquement les éléments et les retourne dans l'ordre

```
Comparator<? super E> comparator()
E first()
E last()
SortedSet<E> headSet(E toElement) // Elements < toElement
SortedSet<E> subSet(E fromElement, E toElement)
SortedSet<E> tailSet(E fromElement) // Elements >= from Element
```

L'interface NavigableSet

- Méthodes permettant de trouver les éléments les plus proche d'un élément donné

E	ceiling(E e)
Iterator<E>	descendingIterator()
NavigableSet<E>	descendingSet()
E	floor(E e)
SortedSet<E>	headSet(E toElement)
NavigableSet<E>	headSet(E toElement, boolean inclusive)
E	higher(E e)
Iterator<E>	iterator()
E	lower(E e)
E	pollFirst()
E	pollLast()
NavigableSet<E>	subSet(E fromElt, boolean fromInclu, E toElt, boolean toInclu)
SortedSet<E>	subSet(E fromElement, E toElement)
SortedSet<E>	tailSet(E fromElement)
NavigableSet<E>	tailSet(E fromElement, boolean inclusive)

Besoin de comparaison …

- Soit les objets sont « Comparable »
  - Interface Comparable<E>
    - public int compareTo(E e)

- Soit il faut un comparateur
  - Interface ***Comparator<E>***
  - int **compare**(E e1, E e2)
    - Nombre négatif si e1 est inférieur à e2
    - 0 si e1 = e2
    - Nombre positif si e1 est supérieur à e2

La classe TreeSet

- Implémente SortedMap et NavigableMap
- Arbre (ordonné) d'éléments

```
TreeSet<Integer> arbre = new TreeSet<Integer>();
```

  - Ajouter un élément

```
arbre.add(12);
```

  - Accéder à un élément

```
arbre.first();
```

```
arbre.last();
```

  - Parcourir l'ensemble des éléments

```
Iterator<Integer> i = arbre.iterator();
while(i.hasNext())
  System.out.print(i.next()+" ");
```

```
Iterator<Integer> i = arbre. descendingIterator();
while(i.hasNext())
  System.out.print(i.next()+" ");
```

Exercice

- Créer une classe Produit qui a un nom et un prix
  - Méthode toString()
  - Comparable sur le nom
- Créer une classe ListeProduit
  - Avec un TreeSet pour l'ensemble des produits
- Créer un comparateur de produit
  - Comparaison par prix, puis par ordre alpha si égalité

L'interface Map

- Association clé-valeur

- Ses principales méthodes

```
boolean containsKey(Object key)
boolean containsValue(Object value)
boolean equals(Object o)
V get(Object key)
boolean isEmpty()
V put(K key, V value)
V remove(Object key)
int size()
Collection<V> values()
```

La classe HashMap

- Implémente l'interface Map
- Table de hachage

```
HashMap<Matiere,Integer> dico = new HashMap<Matiere,Integer>();
```

  - Ajouter un élément

```
dico.put(Matiere.MATHS, 10);
```

  - Accéder à un élément

```
int n = dico.get(Matiere.ANGLAIS);
```

  - Parcourir l'ensemble des éléments

```
for(Matiere m : dico.keySet())
   System.out.println(m+" : "+dico.get(m));
```

```
for(Map.Entry<Matiere, Integer> m : dico.entrySet())
   System.out.println(m.getKey()+" : "+m.getValue());
```

Exercice

- Comparaison des structures ArrayList, LinkedList, TreeSet
  - Ajout de 500 000 à 5 000 000 d'éléments par pas de 500 000
  - Recherche d'un nombre
  - Récupérer le premier élément, le dernier élément et l'élément au milieu de la structure