Chapitre 12

Comparaison et tri d'objets

1. Intérêt et définition
2. Comparaison et tri de données primitives
3. Comparaison et tri d’objets complexes

1 Intérêt et définition

2 Comparaison et tri de données primitives

3 Comparaison et tri d’objets complexes

1 Intérêt et définition

- La **comparaison** entre deux objets (primitifs ou non) est le fait de déterminer s'ils sont équivalents ou non
- Le **tri** est une notion basée sur la comparaison d'objets avec pour but de les classifier par ordre croissant ou décroissant dans un tableau ou une collection
- La comparaison d'objets et leur tri dans des tableaux ou des collections sont des problématiques que l'on rencontre très souvent dans le traitement des données dans les programmes
- Cependant, il n'existe aucune structure de données automatiquement capable de trier des objets que l'on y stocke.
  Ceci est dû au fait que la comparaison est basée sur des critères qui ne peuvent être déterminés à l'avance.
  Car dépendant du contexte et du type de données manipulées
  > Il s'agit donc d'une tâche que Java délègue à la responsabilité du programmeur
- Mais, pour éviter au programmeur de re-implémenter de zéro des algorithmes de comparaison et de tri, Java propose des fonctionnalités natives abstrayant ces algorithmes, mais réclamant à celui-ci de définir certaines règles en amont
- Un des intérêts principaux de la comparaison et du tri d'objets, est l'amélioration significative des performances d'un programme dans les opérations de recherche sur des ensembles très volumineux de données (primitives ou complexes)

1 Intérêt et définition

2 Comparaison et tri de données primitives

3 Comparaison et tri d’objets complexes

2 Comparaison et tri de données simples (1/2)

- Il n'y a pas particulièrement de difficulté à comparer et trier des données de type primitif
- La comparaison des données de type primitif se fait à l'aide des opérateurs déjà étudies : **==, >, >=, <, <=**
  > byte, short, char, int, long, float, double
- Java fourni dans la classe Arrays de la bibliothèque *java.util,* un ensemble de fonctionnalités qui permettent de trier, de rechercher un élément dans un tableau et même de comparer deux tableaux entre eux, etc. Voici quelques fonctions utiles :

fonction	Description	Exemple	Résultat
void sort(T[] tab)	Tri les éléments du tableau tab par ordre croissant	tab = [4, 10, 3, 2] => Arrays.sort (tab)	tab = [2, 3, 4, 10]
boolean equals(T[] tab1, T[] tab2)	Compare si les tableaux *tab1* et *tab2* sont équivalents case par case et retourne true ou false	tab1= [4, 10, 3, 2] tab2 = [2, 3, 4, 10] => Arrays.equals(tab1, tab2)	false
int binarySearch(T[] tab, T key)	Retourne le numéro de la case de *tab* si l'élément *key* existe	tab = [4, 10, 3, 2], key = 3 => Arrays.binarySearch(tab, 3)	2
List<T> asList(T… a)	Crée une List<T> à partir des d'éléments passés en paramètres	Arrays.asList(2, 10, 3, 2)	[4, 10, 3, 2]

T correspond à un des types primitifs ci-dessus listé

La liste exhaustive des fonctionnalités se trouve ici

2 Comparaison et tri de données simples (2/2)

- Cas spécifique des chaînes de caractères (type **String**) :
- Les chaînes de caractères sont de type de référence, mais qui par leur simplicité sont souvent vus comme un type de données simple
- Cependant, on ne peut comparer deux chaines de caractères au moyen de l'opérateur **==**
  > Exemple : String s1 = ''toto''; String s2 = ''toto''; s1 == s2 => false
  > La comparaison de deux objets de type String à l'aide de == revient à comparer leurs références et non leurs valeurs
- La comparaison en valeur de deux chaînes de caractères se fait au travers de la méthode equals directement implémentée dans la classe String
  > Exemple : String s1 = ''toto''; String s2 = ''toto''; s1.equals(s2) => true
- Le tri d'un tableau de String peut se faire au travers de la classe Arrays et des méthodes que nous avons étudiées à la slide précédente
  > Exemple : tab = [''toto'', ''tata'', ''titi'']; Arrays.sort(tab) => [''tata'', ''titi'', ''toto'']

1 Intérêt et définition

2 Comparaison et tri de données primitives

3 Comparaison et tri d’objets complexes

3 Comparaison et tri de données complexes (1/8)

- On qualifie un objet de complexe, lorsqu'il est instance d'un type de référence et plus particulièrement d'une Classe
- On ne peut comparer que des objets de même type et la comparaison ne concerne ici que l'égalité ou la non égalité.
- La comparaison des objets complexes n'est pas aussi simple que les données de types simples
- Une classe étant constituée d'un ensemble d'attributs (simples ou complexes), rendre ses instances comparables entre elles, doit suivre la procédure suivante :

  1/ Choisir l'ensemble des attributs pertinents, qui selon le développeur, comptent dans la différentiation entre ses objets

  2/ Redéfinir la méthode *equals()* dont toute classe hérite par défaut de la classe de base Object

  > C'est dans cette méthode *equals()* que les tests de différentiation entre les attributs seront effectués

  ➔ Une fois cela fait, il sera possible d'appliquer l'opération *obj1.equals(obj2)* pour comparer les deux objets *obj1* et *obj2*

- Pour trier des instances d'une classe stockées dans une collection ou un tableau, alors de deux choses l'une :

  > utiliser la stratégie de l'interface *Comparable* en définissant sa méthode abstraite *compareTo()*

  > utiliser la stratégie de l'interface *Comparator* en définissant sa méthode abstraite *compare()*

  ➔ une fois cela fait, il sera possible de les trier dans une collection à l'aide de la méthode *java.util.Collections.sort(..)* ou dans un tableau à l'aide de la méthode *java.util.Arrays.sort(..)*

Comparaison d'objets

3 Comparaison et tri de données complexes (2/8)

Comparaison d'instances de classe (1/3)

- Considérons la classe Livre suivante :

```
public class Livre {

  String titre;
  String auteur;
  int nbrePage;
  String prefacier;


  public String getTitre() {
    return titre;
  }

  public String getAuteur() {
    return auteur;
  }

  Public int getNbrePage() {
    return nbrePage;
  }

  public String getPrefacier() {
    return prefacier;
  }

}
```

- Pour rendre les objets de cette classe comparables entre eux, nous devons choisir parmi ses 4 attributs ceux que nous estimons pertinents dans la différentiation.
  => Pour cet exemple nous choisissons : *titre*, *auteur* et *nbrePage*

- Ensuite nous allons y implémenter la méthode public boolean equals(Object obj)
  > la variable obj correspond à l'objet comparé
  > l'instance courante est l'objet avec qui on essaie de comparer obj
  Exemple : dans l'écriture obj1.equals(obj2), obj1 correspond à l'instance courante et obj2 correspond à l'objet comparé
  > La définition du contenu de la méthode equals() est libre au développeur, mais dans notre exemple, on dira que deux objets obj1 et obj2 de la classe Livre sont égaux :
  - s'ils sont de même type et ont la même référence (adresse mémoire)
  - ou si leurs attributs respectifs ***titre***, ***auteur*** et ***nbrePage*** se valent

3

Comparaison et tri de données complexes (3/8)

Comparaison d'instances de classe (2/3)

```java
public class Livre {

  String titre;
  String auteur;
  int nbrePage;
  String prefacier;

  public String getTitre() {
    return titre;
  }
  public String getAuteur() {
    return auteur;
  }
  public int getNbrePage() {
    return nbrePage;
  }
  public String getPrefacier() {
    return prefacier;
  }

  public boolean equals(Object obj) {
    // si l'instance courant this et obj ont la même référence mémoire, ils sont forcément égaux. On retourne true
    if (this == obj) {
       return true;
    }

    // si obj n'est même pas de type Livre, alors il ne peut pas être égal à l'instance courante qui est de type Livre. On retourne false
    if (!(obj instanceof Livre)) {
       return false;
    }

    // si tous les attributs de obj ne sont pas null et sont égaux aux attributs de l'instance courante this, alors ils les deux objets sont égaux. Sinon, ils ne le sont pas
    Livre other = (Livre) obj;
    if (other.getTitre() != null && other.getAuteur() != null && other.getNbrePage() != 0) {
       return other.getTitre().equals(this.titre) && other.getAuteur().equals(this.auteur) && (other.getNbrePage() == this.nbrePage);
    } else {
       return false;
    }
  }
}
```

3 Comparaison et tri de données complexes (4/8)

Comparaison d'instance des classe (3/3)

- Considérons la classe ComparaisonLivre qui va tester l'égalité entre plusieurs Livres. On suppose que la classe Livre contient les bon constructeurs nécessaires :

```
public class ComparaisonLivre {

     public static void main(String[] args) {

        Livre livre1 = new Livre("Développons en Java", "Jean Michel Doudoux", 3549, null);
        Livre livre2 = new Livre("Apprendre le C++", "Claude Delannoy", 743, null);
        Livre livre3 = new Livre("Développons en Java", "Jean Michel Doudoux", 3549, "Antonio Goncalves");

        System.out.println(livre1.equals(livre2));
        System.out.println(livre1.equals(livre3));
        System.out.println(livre2.equals(livre3));
     }

}
```

> Le résultat affiché est :

```
false
true
false
```

Tri d’objets

3 Comparaison et tri de données complexes (5/8)

Tri d'instances de classe – Stratégie de l'interface Comparable (1/2)

- Considérons la classe Livre suivante :

```
public class Livre {

   String titre;
   String auteur;
   int nbrePage;
   String prefacier;


   public String getTitre() {
     return titre;
   }

   public String getAuteur() {
     return auteur;
   }

   Public int getNbrePage() {
     return nbrePage;
   }

   public String getPrefacier() {
     return prefacier;
   }

}
```

- Pour rendre les objets de cette classe « triables » dans une collection ou un tableau, nous devons choisir parmi ses 4 attributs, l'attribut estimé pertinent qui sera utilisé comme le critère d'ordre de classement croissant ou décroissant entre objets

  => Pour cet exemple nous choisissons l'attribut *nbrePage* pour trier les livres par ordre croissant du nombre de pages

- L'interface ***Comparable<T>*** expose la méthode public int compareTo(T obj)

  > Cette méthode permet de comparer l'objet courant avec l'objet passé en paramètre obj (au travers de l'attribut choisi) et retourne :

  - un nombre positif, si l'objet courant est plus grand que l'objet obj (ordre décroissant)
  - un nombre négatif, si l'objet courant est plus petit que l'objet obj (ordre croissant)
  - 0, si l'objet courant est égal à l'objet obj

  > Comme vous l'avez remarqué, Comparable est une interface générique et T correspond à la classe concerné par le tri. C'est une interface native de Java appartenant à la bibliothèque *java.lang*

3 Comparaison et tri de données complexes (6/8)

Tri d'instances de classe – Stratégie de l'interface Comparable (2/2)

```
public class Livre implements Comparable<Livre> {

   String titre;
   String auteur;
   int nbrePage;
   String prefacier;

   public String getTitre() {
     return titre;
   }
   public String getAuteur() {
     return auteur;
   }
   public int getNbrePage() {
     return nbrePage;
   }
   public String getPrefacier() {
     return prefacier;
   }

   public int compareTo(Livre obj) {
     if(this.nbrePage == obj.getNbrePage()) {
        return 0;
     }else if(this.nbrePage > obj.getNbrePage()) {
        return 1;
     }else {
        return -1;
     }
   }

}
```

```
import java.util.*;

public class TrierLivre {

public static void main(String[] args) {

        Livre livre1 = new Livre("Développons en Java", "Jean Michel Doudoux", 3549, null);
        Livre livre2 = new Livre("Apprendre le C++", "Claude Delannoy", 743, null);
        Livre livre3 = new Livre("Développez un site web en PHP", "Olivier Heurtel", 660, null);

        ArrayList<Livre> listeLivres = new ArrayList<>();
        listeLivres.add(livre1);  listeLivres.add(livre2);  listeLivres.add(livre3);

        //tri par ordre croissant des livres - 1ère Méthode :
        Collections.sort(listeLivres);
        for(Livre livre : listeLivres) {
        System.out.println(livre.getTitre() + " - " + livre.getNbrePage() + " pages");
        }

        //tri par ordre croissant des livres - 2ème Méthode :
        /*
        Livre[] tableauLivres = new Livre[3];
        tableauLivres[0] = livre1; tableauLivres[1] = livre2;  tableauLivres[2] = livre3;
        Arrays.sort(tableauLivres);
        for(Livre livre : tableauLivres) {
        System.out.println(livre.getTitre() + " - " + livre.getNbrePage() + " pages");
        }
        */
    }
}
```

> **Le résultat affiché dans l'un ou l'autre des cas est :**

```
Développez un site web en PHP - 660 pages
Apprendre le C++ - 743 pages
Développons en Java - 3549 pages
```

3 Comparaison et tri de données complexes (7/8)

Tri d'instances de classe – Stratégie de l'interface Comparator (1/2)

- Considérons la classe Livre suivante :

```
public class Livre {

   String titre;
   String auteur;
   int nbrePage;
   String prefacier;


   public String getTitre() {
     return titre;
   }

   public String getAuteur() {
     return auteur;
   }

   Public int getNbrePage() {
     return nbrePage;
   }

   public String getPrefacier() {
     return prefacier;
   }

}
```

- Pour rendre les objets de cette classe « triables » dans une collection ou un tableau, nous devons choisir parmi ses 4 attributs, l'attribut estimé pertinent qui sera utilisé comme le critère d'ordre de classement croissant ou décroissant entre objets

  => Pour cet exemple nous choisissons l'attribut *nbrePage* pour trier les livres par ordre croissant du nombre de pages

- L'interface ***Comparator<T>*** expose la méthode public int compare(T obj1, T obj2)

  > Cette méthode permet de comparer l'objet obj1 et l'objet obj2 tous deux passés en paramètre. La méthode prend deux paramètres contrairement à celle de l'interface *Comparable*, parce que la classe concernée (ici Livre) par le tri n'est pas forcément celle qui hérite de l'interface *Comparator*. On passe par une classe intermédiaire

  > La méthode retourne :

  - un nombre positif, si l'objet obj1 est plus grand que l'objet obj2 (ordre décroissant)
  - un nombre négatif, si l'objet obj1 est plus petit que l'objet obj2 (ordre croissant)
  - 0, si l'objet obj1 est égal à l'objet obj2

  > L'interface *Comparator* appartient à la bibliothèque *java.util*

3 Comparaison et tri de données complexes (8/8)

Tri d'instances de classe – Stratégie de l'interface Comparator(2/2)

```java
public class Livre {

  String titre;
  String auteur;
  int nbrePage;
  String prefacier;

  public String getTitre() {
    return titre;
  }
  public String getAuteur() {
    return auteur;
  }
  public int getNbrePage() {
    return nbrePage;
  }
  public String getPrefacier() {
    return prefacier;
  }

}
```

```java
import java.util.Comparator;

public class LivreComparator implements Comparator<Livre> {

  @Override
  public int compare(Livre obj1, Livre obj2) {
    if(obj1.getNbrePage() == obj2.getNbrePage()) {
      return 0;
    }else if(obj1.getNbrePage() > obj2.getNbrePage()) {
      return 1;
    }else {
      return -1;
    }
  }
}
```

```java
import java.util.*;

public class TrierLivre {

public static void main(String[] args) {

        Livre livre1 = new Livre("Développons en Java", "Jean Michel Doudoux", 3549, null);
        Livre livre2 = new Livre("Apprendre le C++", "Claude Delannoy", 743, null);
        Livre livre3 = new Livre("Développez un site web en PHP", "Olivier Heurtel", 660, null);

        ArrayList<Livre> listeLivres = new ArrayList<>();
        listeLivres.add(livre1);  listeLivres.add(livre2);  listeLivres.add(livre3);

        //tri par ordre croissant des livres - 1ère Méthode :
        Collections.sort(listeLivres, new LivreComparator());
        for(Livre livre : listeLivres) {
        System.out.println(livre.getTitre() + " - " + livre.getNbrePage() + " pages");
        }

        //tri par ordre croissant des livres - 2ème Méthode :
        /*
        Livre[] tableauLivres = new Livre[3];
        tableauLivres[0] = livre1; tableauLivres[1] = livre2;  tableauLivres[2] = livre3;
        Arrays.sort(tableauLivres, new LivreComparator());
        for(Livre livre : tableauLivres) {
        System.out.println(livre.getTitre() + " - " + livre.getNbrePage() + " pages");
        }
        */
    }
}
```

> **Le résultat affiché dans l'un ou l'autre des cas est :**

```
Développez un site web en PHP - 660 pages
Apprendre le C++ - 743 pages
Développons en Java - 3549 pages
```


SUCCESS