Conception orientée objet

Application du cours 5 : Marché aux poissons et comparaison de gaulois

---

Cet énoncé vient en appui des diapositives du Cours.

I. Comparaison d'objets

1. Ordre naturel

a) Implémenter l'interface « *Comparable* » afin que les poissons puissent être triés selon leur date de pêche puis s'ils ont la même date, selon leur poids. Deux poissons seront considérés identiques si leurs deux attributs sont identiques.

```
public class Poisson                                                    {
  private Date datePeche;
  private float poids;

  public Poisson(Date datePeche, float poids) {
    this.datePeche = datePeche;
    this.poids = poids;
  }

}
```

b) Ecrire les méthodes nécessaires à votre implémentation précédente.

```
public class Date {
   int annee;
   int mois;
   int jour;

   public Date(int jour, int mois, int annee) {
      this.jour = jour;
      this.mois = mois;
      this.annee = annee;
   }

   public boolean equals( ) {
   }

   public int compareTo( ) {
   }
}
```

2. Classe « Gaulois »

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

   private String nom;
   private int age;

   public Gaulois(String nom, int age) {
      this.nom = nom;
      this.age = age;
   }

   public boolean equals(Object obj) {
      if (obj instanceof Gaulois) {
         Gaulois gaulois = (Gaulois) obj;
         return nom.equals(gaulois.nom);
      }
      return false;
   }

   public String toString() {
      return nom + " à " + age + " ans";
   }

   public int compareTo(Gaulois gauloisToCompare) {
      return nom.compareTo(gauloisToCompare.nom);
   }
}
```

3. TreeSet

Dans le main d'une classe « Test » nous avons créé les objets suivants :

```
Gaulois asterix = new Gaulois("Astérix", 35);
Gaulois obelix = new Gaulois("Obélix", 30);
Gaulois abraracourcix = new Gaulois("Abraracourcix", 40);
Gaulois bonemine = new Gaulois("Bonemine", 36);
Gaulois panoramix = new Gaulois("Panoramix", 90);
```

Puis nous les avons ajoutés dans un **TreeSet** ensemble :

```
NavigableSet<Gaulois> ensemble = new TreeSet<>();
ensemble.add(asterix);
ensemble.add(obelix);
ensemble.add(abraracourcix);
ensemble.add(bonemine);
ensemble.add(panoramix);
```

Le main est complété par :

System.***out***.println("Tri par ");
System.***out***.println(ensemble);

Par quelle fin de phrase doit-on compléter la sortie écran (« Tri par … ») pour donner l'ordre de l'affichage de l'ensemble des gaulois ensemble

Ecrire la méthode *afficherGaulois* dans la classe **Test** qui affiche chacun des gaulois contenu dans le **TreeSet** :
- en retournant à la ligne entre chaque gaulois,
- en utilisant un itérateur.

```
public static void afficherGaulois(NavigableSet<Gaulois> ensemble) {

}
```

4. Ordre impose

Créer un ensemble ensembleGaulois qui sera ordonné du plus ancien au plus jeune et s'ils ont le même âge selon l'ordre alphabétique.

Ajouter l'ensemble des Gaulois de la collection ensemble, ainsi que deux nouveaux gaulois : Ordralfabétix et Cétautomatix qui ont tous deux 41 ans.

II. Les ensembles triés

1. NavigableSet

Dans cette partie seules les méthodes de l'interface **NavigableSet** devront être utilisées. Avec ensembleGaulois = {Bonnemine à 36 ans, Astérix à 35 ans, Obélix à 30 ans}

a) Donner le premier gaulois qui a plus de 35 ans.

b) Donner le premier gaulois qui a 35 ans au moins.

c) Que changer dans la méthode *afficherGaulois* écrite précédemment pour afficher les gaulois en ordre inverse ?

Modifier :

Par :

2. Les vues

Créer et afficher la vue selection correspondant aux gaulois ayant plus de 40 ans (40 ans exclus).

Créer et afficher l'ensemble ensembleSelection aux gaulois ayant plus de 40 ans (40 ans exclus).

L'affichage de la vue selection et l'affichage de l'ensemble ensembleSelection est le même :

```
[Panoramix à 90 ans, Cétautomatix à 41 ans, Ordralfabétix à 41 ans]
```

On ajoute le père d'Obélix « Obélodalix » à l'ensemble des gaulois trié par âge :

```
ensembleGaulois.add(new Gaulois("Obélodalix",62));
```

Quel est l'affichage correspondant aux instructions suivantes :
System.*out*.println("Affichage Selection :");
System.*out*.println(selection);

Quel est l'affichage correspondant aux instructions suivantes :
System.*out*.println("Affichage Ensemble :");
System.*out*.println(ensembleSelection);

III. TreeSet (extrait)

Constructor and Description
TreeSet() Constructs a new, empty tree set, sorted according to the natural ordering of its elements.
TreeSet(Collection<? extends E> c) Constructs a new tree set containing the elements in the specified collection, sorted according to the *natural ordering* of its elements.
TreeSet(Comparator<? super E> comparator) Constructs a new, empty tree set, sorted according to the specified comparator.

Modifier and Type	Method and Description
boolean	add(E e) Adds the specified element to this set if it is not already present.
boolean	addAll(Collection<? extends E> c) Adds all of the elements in the specified collection to this set.
E	ceiling(E e) Returns the least element in this set greater than or equal to the given element, or **null** if there is no such element.
void	clear() Removes all of the elements from this set.
boolean	contains(Object o) Returns **true** if this set contains the specified element.
Iterator<E>	descendingIterator() Returns an iterator over the elements in this set in descending order.
E	first() Returns the first (lowest) element currently in this set.
E	floor(E e) Returns the greatest element in this set less than or equal to the given element, or **null** if there is no such element.
NavigableSet<E>	headSet(E toElement, boolean inclusive) Returns a view of the portion of this set whose elements are less than (or equal to, if **inclusive** is true) **toElement**.
E	*higher*(*E* e) Returns the least element in this set strictly greater than the given element, or **null** if there is no such element.
boolean	*isEmpty*() Returns **true** if this set contains no elements.
Iterator<*E*>	*iterator*() Returns an iterator over the elements in this set in ascending order.
E	*last*() Returns the last (highest) element currently in this set.
E	*lower*(*E* e) Returns the greatest element in this set strictly less than the given element, or **null** if there is no such element.
E	*pollFirst*() Retrieves and removes the first (lowest) element, or returns **null** if this set is empty.
E	*pollLast*() Retrieves and removes the last (highest) element, or returns **null** if this set is empty.
int	*size*() Returns the number of elements in this set (its cardinality).
NavigableSet<*E*>	*subSet*(*E* fromElement, boolean fromInclusive, *E* toElement, boolean toInclusive) Returns a view of the portion of this set whose elements range from **fromElement** to **toElement**.
NavigableSet<*E*>	*tailSet*(*E* fromElement, boolean inclusive) Returns a view of the portion of this set whose elements are greater than (or equal to, if **inclusive** is true) **fromElement**.

1. *Methods inherited from class java.util.*AbstractSet
equals, ***hashCode***, ***removeAll***
2. ***Methods inherited from class java.util.***AbstractCollection
containsAll, ***retainAll***, ***toArray***, ***toArray***, ***toString***
3. ***Methods inherited from class java.lang.***Object
finalize, ***getClass***, ***notify***, ***notifyAll***, ***wait***, ***wait***, ***wait***
4. ***Methods inherited from interface java.util.***Set
containsAll, ***equals***, ***hashCode***, ***removeAll***, ***retainAll***, ***toArray***, ***toArray***