

---

# *Construction et réutilisation de composants logiciel*

Collections et genericité

---

L3 Informatique  
Semestre 6

Cours donné par  
Rédigé par Antoine de ROQUEMAUREL

2014

---

# Table des matières

---

<b>1</b>	<b>Listes : ArrayList et Iterator</b>	<b>3</b>
1.1	Détection d'un Palindrome . . . . .	3
1.2	Gestion de tâches . . . . .	3
<b>2</b>	<b>Généricité</b>	<b>5</b>
<b>3</b>	<b>Files : Queue</b>	<b>8</b>
3.1	Travail sur la collection <code>ArrayQueue</code> . . . . .	8
3.2	Travail sur l'ordre naturel avec <code>Comparable</code> . . . . .	8
3.3	Travail sur la collection <code>PriorityQueue</code> . . . . .	9
3.4	Pour aller plus loin ... Problème : gestion d'un planning . . . . .	9
<b>4</b>	<b>Ensembles : Set</b>	<b>12</b>
4.1	<code>EnumSet</code> . . . . .	12
4.2	Arbre rouge-noir : <code>TreeSet</code> . . . . .	12
4.3	<code>HashSet</code> . . . . .	14
<b>5</b>	<b>Association : Map</b>	<b>16</b>
5.1	<code>EnumMap</code> . . . . .	16
5.2	<code>HashMap</code> . . . . .	16
<b>6</b>	<b>Synthèse : La ludothèque</b>	<b>19</b>
<b>A</b>	<b>Liste des codes sources</b>	<b>22</b>

# 1

## Listes : ArrayList et Iterator

---

### 1.1 Détection d'un Palindrome

```
1 boolean isPalindrome(String s) {  
    ArrayList<Character> l;  
3    // l contient les caractères de la String  
    ListIterator iBegin = l.iterator();  
5    ListIterator iEnd = l.iterator(l.size());  
    boolean out = true;  
7  
    while(iBegin.hasNext() && iEnd.hasPrevious() &&  
9        iBegin.nextIndex() <= iEnd.previousIndex() && out) {  
        out = iBegin.next().equals(iEnd.previous());  
11    }  
13    return out;  
}
```

Listing 1.1 – Palindrome

### 1.2 Gestion de tâches

```
1 public abstract class Tache {  
2     public abstract String toString();  
3 }  
4  
5 public final class TacheCodage extends Tache {  
6     private final String spec;  
7     public TacheCodage(final String spec) {  
8         this.spec = spec;  
9     }  
10  
11     public String getSpec() {  
12         return spec;  
13     }  
14  
15     public String toString() {  
16         return "Code "+spec;  
17     }  
18 }  
19  
20 public final class TacheTelephone extends Tache {  
21     private final String nom;  
22     private final String numero;  
23  
24     public TacheTelephone(final String nom, final String numero) {  
25         this.nom = nom;  
26     }  
27 }
```

```
26     this.numero = numero;
27 }
28
29 public String getNom() {
30     return nom;
31 }
32 public String getNumero() {
33     return numero;
34 }
35
36 public String toString() {
37     return "Telephone "+nom;
38 }
39 }
```

Listing 1.2 – Classes Tache TacheCodage et TacheTelephone

```
1 Tache appelerEric = new TacheTelephone("Ertineric", "0211223344");
2 Tache appelerMartine = new TacheTelephone("", "0211223344");
3
4 Tache coderBd = new TacheCodage("bd");
5 Tache coderIHM = new TacheCodage("ihm");
6 Tache coderLogique = new TacheCodage("logique");
7
8 // Le paramètre correspond au nombre d'élément initialement alloués.
9 // Ca permet d'éviter la réallocation inutile
10 // Par défaut = 10
11 List tachsAppel = new ArrayList();
12 List tachsCodage = new ArrayList();
13 List tachsLundi = new ArrayList(8);
14 List tachsMardi = new ArrayList(8);
15
16 tachesAppel.add(appelerEric);
17 tachesAppel.add(appelerMartine);
18
19 tachesCodage.add(coderBd);
20 tachesCodage.add(coderLogique);
21 tachesCodage.add(1, coderIHM);
22
23 tachesLundi.add(coderLogique);
24 tachesLundi.add(appelerMartine);
25 tachesLundi.set(1, appelerEric);
26
27 Tache toutesLesTaches = new ArrayList(tachesLundi);
28 toutesLesTaches.addAll(tachesMardi);
29
30 tachesLundi.remove(appelerEric);
31
32 List tachesMardiNonAppel = new ArrayList(tachesMardi);
33 tachesMardiNonAppel.removeAll(tachsAppel);
34
35 tachesMardi.contains(appelerMartine);
36 tachesMardi.containsAll(tachesMardiAppel);
```

Listing 1.3 – Exercices sur les taches

# 2

## Généricité

---

```
public abstract class Valeur {
2   public abstract String toString();
   public abstract boolean egale(Valeur valeur);
4 }

6 public class Nombre extends Valeur {
   private int nombre;
8   public Nombre(int nombre) {
       this.nombre = nombre;
10  }

12   public int getNombre() {
       return nombre;
14  }

16   public String toString() {
       return "" + nombre;
18  }

20   public boolean egale(Valeur valeur) {
       return ((Nombre)valeur).getNombre() == nombre;
22  }
}

24 public enum Image { ROI, DAME, VALET, AS };

26 public class Figure extends Valeur {
28   private Image image;
   private String nom;
30
32   public Figure(Image image) {
       this.image = image;
       this.nom = (image.toString().toLowerCase());
34  }

36   public Image getImage() {
       return image;
38  }
   public String toString() {
       return nom;
40  }

42
   public boolean egale(Valeur valeur) {
       return ((Figure)valeur).getImage() == image;
44  }
}

46 public Carte extends Pair<Valeur,Couleur> {
48   public Carte(Valeur valeur, Couleur couleur) {
       super(valeur, couleur);
   }
```

```

50     }
51 }
52
53 abstract public class Genre {
54     private List<Valeur> liste;
55
56     public Genre(List pValeurs) {
57         valeurs = pValeurs;
58     }
59
60     private int positionCarte(Valeur val) {
61         int pos = 0;
62         for(Valeur v : liste) {
63             if(val.equals(v)) {
64                 break;
65             }
66             ++pos;
67         }
68
69         return pos;
70     }
71 }
72
73 public class HorsAtout extends Genre {
74     public Atout(List l) {
75         super(Arrays.asList(
76             new Figure(Image.AS), new Nombre(10), new Figure(Image.ROI), new ←
77                 Figure(Image.VALET), new Figure(Image.DAME),
78             new Nombre(9), new Nombre(8), new Nombre (7)
79         ));
80     }
81 }
82
83 public class Atout extends Genre {
84     public HorsAtout() {
85         super(Arrays.asList(
86             new Figure(Image.VALET), new Nombre(9), new Figure(Image.AS), new ←
87                 Nombre(10), new Figure(Image.ROI),
88             new Figure(Image.DAME), new Nombre(8), new Nombre (7)
89         ));
90     }
91 }
92
93 public class Jeu {
94     private Couleur atout;
95     private int calculGagnant(Carte c1, Carte c2) {
96
97     }
98
99     public int levee(Couleur atout, Pair<Carte, Carte> equipeMain, ←
100         Paire<Carte, Carte< equipe) {
101         this.atout = atout;
102         int numVainqueurEquipeMain = calculGagnant(equipeMain.getFirst(), ←
103             equipeMain.getSecond());
104         Carte joueurEquipeMain;
105         // TODO
106     }
107
108     static class ClassInterne<G extends Genre> {
109         G genre;
110
111         public classInterne(G genre) {
112             this.genre = genre;
113         }
114     }

```

---

```
108     }
110 }

112 public class Test {
113     public static void main(String[]s) {
114         Carte asCarreau = new Carte(new Figure(Image.AS, Couleur.CARREAU));
115         Carte roiPique = new Carte(new Figure(Image.ROI, Couleur.PIQUE));
116     }
}
```

Listing 2.1 – TD belotte

# 3

### 3.1 Travail sur la collection ArrayDeque

```
1 Deque<Tache> fileDeTaches = new ArrayDeque<>();
  fileDeTaches.offer(appelerEric);
3 fileDeTaches.offer(appelerMartine);

5 System.out.println("La tache suivante est : "+fileDeTaches.poll() +
  " est traité");
7 System.out.println("La tache suivante est : "+fileDeTaches.peek());
  System.out.println("La tache suivante est : "+fileDeTaches.remove() +
9     "est traité");
```

Listing 3.1 – ArrayDeque

### 3.2 Travail sur l'ordre naturel avec Comparable

```
1 public abstract class Tache implements Comparable<Tache> {
  public abstract toString();
3  public boolean equals(Object objetAComparer) {
    if(objetAComparer instanceof Tache) {
5      Tache tacheAComparer = (Tache)objetAComparer;
      return toString().equals(tacheAComparer.toString());
7    } else {
      return false;
9    }
  }

11
  public int compareTo(Tache tacheAComparer) {
13    return toString().compareTo(tacheAComparer.toString());
  }
15 }
```

Listing 3.2 – Comparable

```
1 public class TacheAvecPriorite implements Comparable<TacheAvecPriorite> {
  private final Tache tache;
3  private Priorite priorite;

5  public TacheAvecPriorite(Tache tache, Priorite priorite) {
    this.tache = tache;
7    this.priorite = priorite;
  }

9
  public String toString() {
11    retn tache+" : "+priorite;
  }
```



```

13 public boolean equals(Object objetAComparer) {
14     if(objetAComparer instanceof TacheAvecPriorite) {
15         TacheAvecPriorite tacheAComparer = (TacheAvecPriorite) objetAComparer;
16         return tache.equals(tacheAComparer.tache) && ←
17             priorite.equals(tacheAComparer.priorite);
18     } else {
19         return false;
20     }
21 }

22
23 public int compareTo(TacheAvecPriorite tacheAComparer) {
24     int resultat = priorite.compareTo(tacheAComparer.priorite);
25     if(resultat == 0) {
26         resultat = tache.compareTo(tacheAComparer.tache);
27     }
28
29     return resultat;
30 }
31 }

```

Listing 3.3 – TachesAvecPriorite

### 3.3 Travail sur la collection PriorityQueue

```

1 final int CONTENANCE_INITIAL = 10;
2
3 Comparator<TacheAvecPriorite> cmpPriorite = new ←
4     Comparator<TacheAvecPriorite>() {
5         public int compare(TacheAvecPriorite tache1,
6             TacheAvecPriorite tache2) {
7             return tache1.getPriorite().compareTo(tache2.getPriorite());
8         }
9     };
10
11 Queue<TacheAvecPriorite> fileAvecPriorite = new ←
12     PriorityQueue<TacheAvecPriorite>(CONTENANCE_INITIAL, cmpPriorite);

```

Listing 3.4 – TachesAvecPriorite

### 3.4 Pour aller plus loin . . . Problème : gestion d'un planning

```

1 public class FileTachesPrioritaires {
2     private Queue<TacheAvecPriorite> fileDeTaches;
3     private boolean estArrete = false;
4
5     public FileTachesPrioritaires() {
6         fileDeTaches = new PriorityQueue<>();
7     }
8
9     public boolean ajouterTache(TacheAvecPriorite tache) {
10         if(estArrete) {
11             return false;
12         } else {
13             fileDeTaches.offer(tache);
14             return true;
15         }
16     }

```

```

18 public TacheAvecPriorite obtenirTache() {
19     return fileDeTaches.poll();
20 }
21
22 public Collection(TacheAvecPriorite> arreter() {
23     estArrete = true;
24     return fileDeTaches;
25 }
26
27 public String toString() {
28     String file = "";
29     int tailleFile = fileDeTaches.size();
30     int i = 1;
31
32     for (TacheAvecPriorite tache : fileDeTaches) {
33         file += tache;
34         if (i < tailleFile) {
35             file += ", ";
36         }
37         ++i;
38     }
39
40     return file;
41 }
42 }

```

Listing 3.5 – FileTachesPrioritaires

```

1 public class OrdonnanceurTaches {
2     private final int DUREE_PLANNING_EN_JOUR = 365;
3
4     private List<FileTachesPrioritaires> planning;
5     private int jourCourant;
6
7     public OrdonnanceurTaches() {
8         jourCourant = 0;
9         planning = new ArrayList<>();
10        for(int i = 0 ; i < DUREE_PLANNING_EN_JOUR ; ++i) {
11            planning.add(new FileTachesPrioritaires());
12        }
13    }
14
15    public void ajouterTache(TachesAvecPriorite tache, int jour) {
16        if(jour < 0 || jour > DUREE_PLANNING_EN_JOUR) {
17            throw new IllegalArgumentException("jour hors planning");
18        }
19
20        FileTachesPrioritaires fileTachesDuJour = planning.get(jour);
21
22        if(!fileTachesDuJour.ajouterTache(tache)) {
23            throw new IllegalArgumentException("impossible d'ajouter la tache"+tache);
24        }
25    }
26
27    public TachesAvecPriorite obtenirTache() {
28        return planning.get(jourCourant).obtenirTache();
29    }
30
31    public void renouveler() {
32        Collectino<TachesAvecPriorite> tacheRstantes = ←
33        planning.get(jourCourant).arreter();

```

```
33     planning.remove(jourCourant);
34     FileTachesPrioritaires premierJour = planning.get(jourCourant);
35     for(TachesAvecPriorite tache : tacheRstantes) {
36         tache.incrementsPriorite();
37         premierJour.ajouterTache(tache);
38     }
39
40     FileTachesPrioritaires dernierJour = new FileTachesPrioritaires();
41     planning.add(jourCourant, dernierJour);
42     jourCourant = (jourCourant+1) % DUREE_PLANNING_EN_JOUR;
43 }
44
45 public ListIterator<FileTachesPrioritaires> obtenirSousPlanning(int ←
46     premierJour, int dernierJour) {
47     return planning.subList(premierJour, dernierJour).listIterator();
48 }
```

Listing 3.6 – OrdonnanceurTaches

# 4

## Ensembles : Set

---

### 4.1 EnumSet

```
1 public enum Apprecation {  
    EXCEPTIONNEL, TRESBIEN, BIEN, PASSABLE, MAUVAIS, TRESMAUVAIS  
3 }  
  
5 EnumSet<Apprecation> e1 = e.allOf(Apprecation.class);  
EnumSet<Apprecation> e2 = e.range(Apprecation.EXCEPTIONNEL, Apprecation.BIEN);  
7 EnumSet<Apprecation> e3 = e.complementOf(e2);  
EnumSet<Apprecation> e34= e.of(Apprecation.EXCEPTIONNEL);
```

Listing 4.1 – EnumSet

### 4.2 Arbre rouge-noir : TreeSet

```
1 public class Video extends Comparable<Video> {  
    private int annee;  
3    private String realiateur;  
    private String titre;  
  
5  
    public Video(String ptitre, String prealisateur, int pannee) {  
7        annee = pannee;  
        realiateur = prealisateur;  
9        titre = ptitre;  
    }  
  
11  
    public String getTitre() {  
13        return titre;  
    }  
  
15  
    public int getAnnee() {  
17        return annee;  
    }  
  
19  
    public String getRealisateur() {  
21        return realiateur;  
    }  
  
23  
    public String toString() {  
25        return titre.toString() + " " +  
            realiateur.toString() + " "+annee.toString();  
27    }  
  
29    public int compareTo(Video v) {  
        int comparaison = titre.compareTo(v.getTitre());  
31    }
```

```

33     if(compairaison != 0) {
34         return compairaison;
35     }
36     compairaison = realiateur.compareTo(v.getRealisateur());
37
38     if(compairaison != 0) {
39         return compairaison;
40     }
41
42     return (new Integer(annee)).compareTo(new Integer(v.getAnnee()));
43 }
44
45 public boolean equals(Object o) {
46     if(!o instanceof Video) {
47         return false;
48     }
49
50     Video v = (Video)o;
51     return (titre.equals(v.getTitre()) &&
52         realiateur.equals(v.getRealisateur()) &&
53         annee.equals(o.getAnnee()));
54 }
55 }

```

Listing 4.2 – Classe Video

```

1 public class Test {
2     public static void afficherElements(NavigableSet<Video> ens) {
3         for(Video v : ens) {
4             System.out.println(v);
5         }
6     }
7
8     public static void afficherElementOrdreInverse(NavigableSet<Video> ens) {
9         for(Video v : ens) {
10             System.out/println(v);
11         }
12     }
13
14     public static void main(String[] args) {
15         NavigableSet<Video> ens = new TreeSet<Video>();
16         ensemble.add(new Video("le jour le plus long", "Ken Annakin", 1962));
17         ensemble.add(new Video("Un pont trop loin", "Richard Attenborough", 1977));
18         ensemble.add(new Video("Platoon", "Olier Stone", 1986));
19         ensemble.add(new Video("Full metal jacket", "Stanley Kubrick", 1987));
20         ensemble.add(new Video("La ligne rouge", "Terrence Malick", 1962));
21         ensemble.add(new Video("The patriot", "Roland Emmerich", 2000));
22
23         afficherElements(ens);
24
25         // tri par réalisateurs
26         NavigableSet<Video> ensRealisateur = new TreeSet<>(
27             new Comparator<Video>() {
28                 public int compare(Video v1, Video v2) {
29                     return v1.getRealisateur().compareTo(v2.getRealisateur());
30                 }
31             });
32         ensRealisateur.addAll(ens);
33         afficherElements(ensRealisateur);
34
35         // tri par annee

```

```

36     NavigableSet<Video> ensAnnee = new TreeSet<>(
37         new Comparator<Video>() {
38             public int compare(Video v1, Video v2) {
39                 return (new Integer(v1.getAnnee())).compareTo(v2.getAnnee());
40             }
41         });
42     ensAnnee.addAll(ens);
43     afficherElements(ensAnnee);
44
45     System.out.println("Le remier film tourné à partr de 1977" +
46         ensAnnee.ceiling(new Video("", "", 1977));
47
48
49     System.out.println("Le remier film tourné avant 1977" +
50         ensAnnee.lower(new Video("", "", 1977));
51
52     afficherElementOrdreInverse(ensAnnee);
53     // afficher la selection correspondant aux films tournés
54     // après "Full metal jacket"
55     SortedSet selection1 = ensAnnee.tailSet(new Video("Full metal jacket",
56         "Stanley Kubric", 1987));
57     System.out.print("affichage selection1 ");
58
59     // afficher l'ensemble correspondant aux fils tournées
60     // après "Full metal jacket"
61     NavigableSet<Video> ensembleSelection1 = new TreeSet<> {
62         ensembleSelection1.retainAll(selection1);
63     }
64
65
66     ensAnnee.add(new Video("Sherlock Holmes", "Guy Ritchie", 2010));
67
68     NavigableSet<Video> selection2 = ensAnnee.subset(new Video("", "", 1987),
69         true, new Video("", "", 2000), false);
70     afficherElements(selection2);
71
72     NavigableSet<Video> selection3 = ensAnnee.subset(new Video("", "", 1992),
73         true, new Video("", "", 2012), true);
74     NavigableSet<Video> ensembleSelectionne = new TreeSet<Video>(ensemble);
75     ensembleSelectionne.retainAll(selection3);
76     afficherElements(selection3);
77
78 }
79
80 }

```

Listing 4.3 – Affichage

## 4.3 HashSet

```

1  Set<Video> ensembleAlouer = new HashSet<>(ensembleTrie);
2  Set<Video> ensembleLoue = new HashSet<>();
3
4
5  public int hashCode() {
6      return titre.hashCode() * annee * realisateur.hashCode();
7  }
8
9
10 public boolean equals(Object o) {
11     if(o instanceof Video) {
12         Video videoToCompare = (Video) o;

```

```

    return (titre.equals(videoToCompare.getTitre()) &&
10         realiseur.equals(videoToCompare.getRealisateur()) &&
        annee == videoToCompare.annee);
12 } else {
    return false
14 }
}
```

Listing 4.4 – Ajout du hashCode et equals dans Video

```

System.out.println("", !ensembleLoue.isEmpty());
2
// Sans utiliser add
4 ensembleAlouer.remove(new video("Le jour le plus long", "Ken Annakin", 1962));
ensembleLoue.addAll(ensembleTrie);
6 ensembleLoue.removeAll(ensembleAlouer);
```

On doit ajouter dans le equals la possibilité de comparer une VideoAppreciation.

```

2 public boolean equals(Object o) {
    if(o instanceof VideoAppreciation) {
4         VideoAppreciation videoToCompare = (VideoAppreciation) o;
        return (titre.equals(videoToCompare.getVideo().getTitre()) &&
6             realiseur.equals(videoToCompare.getVideo().getRealisateur()) &&
            annee == videoToCompare.getVideo().annee);
8     if(o instanceof Video) {
        Video videoToCompare = (Video) o;
10        return (titre.equals(videoToCompare.getTitre()) &&
            realiseur.equals(videoToCompare.getRealisateur()) &&
12            annee == videoToCompare.annee);
    } else {
14        return false
    }
16 }

1 Set<Video> ensembleLie = new LinkedHashSet<>(ensembleTrie);
```

Les vidéos sont affichés triés car `ensembleTrie` était trié. Si on ajoute une nouvelle vidéo celle-ci sera à la fin : affichage dans l'ordre d'insertion.

# 5

## Association : Map

---

### 5.1 EnumMap

```
1 Map<Priorite, ArrayDeque<Tache>> m = new EnumMap<>(Priorite.class);
  for(Priorite p : Priorite.value()) {
3     m.put(p, new ArrayDeque<>());
  }
5
  m.get(Priorite.MOYENNE).offerFirst(appelerEric);
7 m.get(Priorite.HAUTE).offerFirst(appelerMartine);
  m.get(Priorite.HAUTE).offerFirst(coderBD);
9 m.get(Priorite.BASSE).offerFirst(coderLogique);
  m.get(Priorite.MOYENNE).offerFirst(coderIHM);
11
  ArrayDeque<Tache> l = m.get(Priorite.HAUTE);
13 System.out.println("La liste des taches de priorités haute est : "+l);
```

Listing 5.1 – Utilisation de EnumMap

### 5.2 HashMap

```
1 public class Client {
  private String _name;
3
  public Client(String name) {
5     _name = name;
  }
7
  public String getNom() {
9     return name;
  }
11
  @Override
13 public String toString() {
    return _name;
15 }
17
  public void facturer(Tache t) {
    System.out.println("Le client "+nom" recoit la facture concernant "+
19     "la tache "+t);
  }
}
```

Listing 5.2 – Classe Client

```
public abstract class Tache {
2     @Override
    public boolean equals(Object o) {
```



```

4      if(o instanceof Tache) {
        return toString().equals(o.toString());
6      } else {
        return false;
8      }
    }
10
    @Override
12    public int hashCode() {
        return toString().hashCode();
14    }

    @Override
16    public String toString();
18 }

```

Listing 5.3 – Classe Tache; equals et hashCode

```

    public class Clee extends Tache {
2      String chaine;

4      public Clee(String pchaine) {
        chaine = pchaine;
6      }

8      public String toString() {
        return chaine;
10     }
    }

```

Listing 5.4 – Classe Concrète de Tache; equals et hashCode

```

1  Client societeBricolage = new Client("Tous travaux");
   Client bijoutier = new Client("L'étincelant");
3  Client jardinerie = new Client("Espace vert");

5  Map<Tache, Client> m = new HashMap<>();
   m.put(coderBD, bijoutier);
7  m.put(coderLogique, jardinerie);
   m.put(coderIHM, societeBricolage);
9  m.put(appelerMartine, societeBricolage);
   m.put(appelerEric, null);
11
   for(Tache t : m.keySet()) {
13     Client c = m.get(t);
        if(c == null) {
15         System.out.print("pas de client");
        } else {
17         System.out.prnt("Client" + c);
        }
19     System.out.println(", pour la tache : "+t);
   }
21 m.get(CoderIHM).facturer(coderIHM);

23 client = m.get(new Clee("Telephone Martine"));
   client.facturer(appelerMartine);
25
   Collection<Client> clients = m.values();
27 for(Iterator<Client> it = clients.iterator() ; it.hasNext()) {
        if(societeBricolage.equals(it.next())) {
29         it.remove();
        }
31 }

```

```
33 Collection<Client> clients = m.values();  
clients.removeAll(Collections.singleton(societeBricolage));  
35 System.out.println("Facturation des clients en préparation :", m);  
  
37 clients.removeAll(Collections.singleton(null));  
System.out.println("Facturation des clients en prépa :"+m);
```

Listing 5.5 – Utilisation de HashMap

# 6

## Synthèse : La ludothèque

---

```
1 public class Jeu {
2     private String _name;
3
4     public Jeu(String name) {
5         _name = name;
6     }
7
8     @Override
9     public boolean equals(Object o) {
10        return o instanceof Jeu && ((Jeu)o.getName()).equals(_name);
11    }
12
13    @Override
14    public int hashCode() {
15        return _name.hashCode();
16    }
17 }
```

Listing 6.1 – Classe Jeu

```
1 public class Personne {
2     private String _firstName;
3     private String _lastName;
4
5     public Personne(String firstName, String lastName) {
6         _firstName = firstName;
7         _lastName = lastName;
8     }
9
10    public String getFirstName() {
11        return _firstName;
12    }
13
14    public String getLastName() {
15        return _lastName;
16    }
17
18    @Override
19    public boolean equals(Object o) {
20        return o instanceof Personne &&
21            _firstName.equals(((Personne)o.getFirstName())) &&
22            _lastName.equals(((Personne)o.getLastName()));
23    }
24
25    @Override
26    public int hashCode() {
27        return _firstName.hashCode() + _lastName.hashCode();
28    }
29 }
```

29 | }

Listing 6.2 – Personne

```
public class Membre extends Personne {
2   private Map<Jeu, Integer> _jeux;

4   public Membre(String firstname, String lastname) {
        super(firstname, lastname);
6       _jeux = new HashMap<Jeu, Integer>();
    }

8   public void ajouterJeu(final Jeu jeu) {
10      _jeux.put(jeu, (_jeux.get(jeu) != null ? _jeux.get(jeu)+1 : 1));
    }

12  public void supprimerJeu(final Jeu jeu) {
14      if(_jeux.get(jeu) > 1) {
16          _jeux.put(jeu, (_jeux.get(jeu))-1);
18      } else {
19          _jeux.remove(jeu);
20      }
21  }

22  public void possedeJeu(final Jeu jeu) {
23      return _jeux.get(jeu) != null;
24  }

25  public Jeu chercherJeu(final String nom) {
26      Jeu ret = new Jeu(nom);
27      boolean found = false;
28      if(!possedeJeu(ret)) {
29          return null;
30      }

31      Jeu j;
32      for(Iterator it = _jeux.keySet().iterator() ; it.hasNext() && !found ;
33          j = ((Jeu)it.next())) {
34          if(j.equals(ret)) {
35              ret = j;
36              found = true;
37          }
38      }

39      return (found) ? ret : null;
40  }

41  public Set<Jeux> getJeux() {
42      return _jeux.keySet();
43  }
44  }
```

Listing 6.3 – Membre

```
public class LudotqueDistribuee {
2   private List<Membre> _membres;

4   public LudothequeDistribuee() {
        _membres = new ArrayList<Membre>();
6   }

8   public Membre ajouterMembre(String prenom, String nom) {
        Membre ret = new Membre(prenom, nom);
    }
```

```

    _memres.add(ret);
10
    return ret;
12
}

14 public boolean retirerMembre(Membre m) {
    return _membres.remove(m);
16
}

18 public List<Membre> membresPossedantJeu(Jeu j) {
    List<Membre> ret = new ArrayList<Membre>();
20
    for(Membre m : _membres) {
22        if(m.possedeJeu(j)) {
            ret.add(m);
24        }
    }

26    return ret;
28
}

30 public Set<Jeu> listeJeux() {
    Set<Jeu> ret = new HashSet<Jeu>();
32    for(Membre m : _membres) {
        ret.addAll(m.getJeux());
34    }

36    return ret;
38
}

38 public double cotisationMembre() {
40    double ret;

42
    return ret;
44
}
}

```

Listing 6.4 – Ludothèque

# A

## Liste des codes sources

---

1.1	Palinrome . . . . .	3
1.2	Classes Tache TacheCodage et TacheTelephone . . . . .	3
1.3	Exercices sur les taches . . . . .	4
2.1	TD belotte . . . . .	5
3.1	ArrayDeque . . . . .	8
3.2	Comparable . . . . .	8
3.3	TachesAvecPriorite . . . . .	8
3.4	TachesAvecPriorite . . . . .	9
3.5	FileTachesPrioritaires . . . . .	9
3.6	OrdonnanceurTaches . . . . .	10
4.1	EnumSet . . . . .	12
4.2	Classe Video . . . . .	12
4.3	Affichage . . . . .	13
	codes/11.java . . . . .	14
4.4	Ajout du hashCode et equals dans Video . . . . .	14
	codes/11-2.java . . . . .	15
	codes/11-3.java . . . . .	15
	codes/11-4.java . . . . .	15
5.1	Utilisation de EnumMap . . . . .	16
5.2	Classe Client . . . . .	16
5.3	Classe Tache; equals et hashCode . . . . .	16
5.4	Classe Concrète de Tache; equals et hashCode . . . . .	17
5.5	Utilisation de HashMap . . . . .	17
6.1	Classe Jeu . . . . .	19
6.2	Personne . . . . .	19
6.3	Membre . . . . .	20
6.4	Ludotheque . . . . .	20