UNIVERSITÉ TOULOUSE III PAUL SABATIER
Université de Toulouse

Rédigé avec LaTeX
Version du 25 février 2014

Construction et réutilisation de composants logiciel

Collections et généricité

L3 Informatique
Semestre 6

Cours donné par
Rédigé par Antoine de Roquemaurel

2014

Table des matières

1 Listes : ArrayList et Iterator

1.1 Détection d'un Palindrome

```
1  boolean isPalindrome(String s) {
     ArrayList<Character> l;
3    // l contient les charactères 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.netx().equals(iEnd.previous());
11   }

13   return out;
}
```

Listing 1.1 – Palinrome

1.2 Gestion de tâches

```
public abstract class Tache {
2   public abstract String toString();
}
4
public final class TacheCodage extends Tache {
6   private final String spec;
    public TacheCodage(final String spec) {
8     this.spec = spec;
    }
10
    public String getSpec() {
12    return spec;
    }
14
    public String toString() {
16    return "Code "+spec;
    }
18 }

20 public final class TacheTelephone extends Tache {
    private final String nom;
22  private final String numero;

24  public TacheTelephone(final String nom, final String numero) {
      this.nom = nom;
```

```
26      this.numero = numero;
    }
28
    public String getNom() {
30      return nom;
    }
32  pulic String getNumero() {
      return numero;
34  }

36  public String toString() {
      return "Telephone "+nom;
38  }
}
```

Listing 1.2 – Classes Tache TacheCodage et TacheTelephone

```
1  Tache appelerEric = new TacheTelephone("Ertineric", "0211223344");
   Tache appelerMartine = new TacheTelephone("", "0211223344");
3
   Tache coderBd = new TacheCodage("bd");
5  Tache coderIHM = new TacheCodage("ihm");
   Tache coderLogique = new TacheCodage("logique");
7
   // Le paramètre correspond au nombre d'élément intialement alloués.
9  // Ca permet d'éviter la réallocation inutile
   // Par défaut = 10
11 List tachsAppel = new ArrayList();
   List tachsCodage = new ArrayList();
13 List tachsLundi = new ArrayList(8);
   List tachsMardi = new ArrayList(8);
15
   tachesAppel.add(appelerEric);
17 tachesAppel.add(appelerMartine);

19 tachesCodage.add(coderBd);
   tachesCodage.add(coderLogique);
21 tachesCodage.add(1,coderIHM);

23 tachesLundi.add(coderLogique);
   tachesLundi.add(appelerMartine);
25 tachesLundi;set(1,appelerEric);

27 Tache toutesLesTaches = new ArrayList(tachesLundi);
   toutesLesTaches.addAll(tachesMardi);
29
   tachesLundi.remove(appelerEric);
31
   List tachesMardiNonAppel = new ArrayList(tachesMardi);
33 tachesMardiNonAppel.removeAll(tachsAppel);

35 tachesMardi.contains(appelerMartine);
   tachesMardi.containsAll(tachesMardiAppel);
```

Listing 1.3 – Exercices sur les taches

2 Généricité

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

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

  public int getNombre() {
    return nombre;
  }

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

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

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

public class Figure extends Valeur {
  private Image image;
  private String nom;

  public Figure(Image image) {
    this.image = image;
    this.nom = (image.toString().toLowerCase();
  }

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

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

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

```
50    }
  }
52
  abstract public class Genre {
54    private List<Valeur> liste;

56    public Genre(List pValeurs) {
        valeurs = pValeurs;
58    }

60    private int positionCarte(Valeur val) {
        int pos = 0;
62      for(Valeur v : liste) {
          if(val.equals(v)) {
64          break;
          }
66        ++pos;
        }
68
        return pos;
70    }
  }
72
  public class HorsAtout extends Genre {
74    public Atout(List l) {
        super(Arrays.asList(
76            new Figure(Image.AS), new Nombre(10), new Figure(Image.ROI), new ↩
                 Figure(Image.VALET), new Figure(Image.DAME),
              new Nombre(9), new Nombre(8), new Nombre (7)
78            );
    }
80  }

82  public class Atout extends Genre {
    public HorsAtout() {
84    super(Arrays.asList(
            new Figure(Image.VALET), new Nombre(9), new Figure(Image.AS), new ↩
                 Nombre(10), new Figure(Image.ROI),
86          new Figure(Image.DAME), new Nombre(8), new Nombre (7)
            );
88  }
  }
90
  public class Jeu {
92  private Couleur atout;
    private int calculGagnant(Carte c1, Carte c2) {
94
    }
96
    public int levee(Couleur atout, Pair<Carte,Carte> equipeMain, ↩
       Paire<Carte,Carte< equipe) {
98    this.atout = atout;
      int numVainqueurEquipeMain = calculGagnant(equipeMain.getFirst(), ↩
         equipeMain.getSecond());
100   Carte joueurEquipeMain;
      // TODO
102 }
    static class ClassInterne<G extends Genre> {
104   G genre;

106   public classInterne(G genre) {
        this.genre = genre;
```

```
108     }
    }
110 }

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

Listing 2.1 – TD belotte

3 Files : Queue

3.1 Travail sur la collection `ArrayQueue`

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

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

Listing 3.1 – `ArrayDeque`

3.2 Travail sur l'ordre naturel avec `Comparable`

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

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

Listing 3.2 – `Comparable`

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

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

  public String toString() {
    retrn tache+" :"+priorite;
  }
```

```
13
  public boolean equals(Object objetAComparer) {
15   if(objetAComparer instanceof TacheAvecPriorite) {
       TacheAvecPriorite tacheAComparer = (TacheAvecPriorite) objetAComparer;
17     return tache.equals(tacheAComparer.tache) && ↩
           priorite.equals(tacheAComparer.priorite);
     } else {
19     return false;
     }
21 }

23 public int compareTo(TacheAvecPriorite tacheAComparer) {
     int resultat = priorite.compareTo(tacheAComparer.priorite);
25   if(resultat == 0) {
       resultat = tache.compareTo(tacheAComparer.tache);
27   }

29   return resultat;
   }
31
}
```

Listing 3.3 – TachesAvecPriorite

3.3 Travail sur la collection PriorityQueue

```
final int CONTENANCE_INITIAL = 10;
2
Comparator<TacheAvecPriorite> cmpPriorite = new ↩
    Comparator<TacheAvecPriorite>() {
4   public int compare(TacheAvecPriorite tache1,
              TacheAvecPriorite tache2) {
6     return tache1.getPriorite().compareTo(tache2.getPriorite());
    }
8 };

10 Queue<TacheAvecPriorite> fileAvecPriorite = new ↩
    PriorityQueue(TacheAvecPriorite>(CONTENANCE_INITIAL, cmpPriorite);
```

Listing 3.4 – TachesAvecPriorite

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

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

```
18    public TacheAvecPriorite obtenirTache() {
20      return fileDeTaches.poll();
      }

22    public Collection(TacheAvecPriorite> arreter() {
        estArrete  = true;
24      retunr fileDeTaches;
      }
26
      public String toString() {
28      String file = "";
        int tailleFile = fileDeTaches.size();
30      int i = 1;

32      for (TacheAvecPriorite tache : fileDeTaches) {
          file += tache;
34        if (i < tailleFile) {
            file += ", ";
36        }
          ++i;
38      }

40      return file;
      }
42
}
```

Listing 3.5 – FileTachesPrioritaires

```
1  public class OrdonnanceurTaches {
     private final int DUREE_PLANNING_EN_JOUR = 365;
3
     private List<FileTachesPrioritaires> planning;
5    private int jourCourant;

7    public OrdonnanceurTaches() {
       jourCourant = 0;
9      planning  = new ArrayList<>();
       for(int i = 0 ; i < DUREE_PLANNING_EN_JOUR ; ++i) {
11       planning.add(new FileTachesPrioritaires());
       }
13   }

15   public void ajouterTache(TachesAvecPriorite tache, int jour) {
       if(jour < 0 || jour > DUREE_PLANNING_EN_JOUR) {
17       throw new IllegalArgumentException("jour hors planning");
       }
19
       FileTachesPrioritaires fileTachesDuJour = planning.get(jour);
21
       if(!fileTachesDuJour.ajouterTache(tache)) {
23       throw new IllegalArgumentException("impossible d'ajouter la tache"+tache);
       }
25   }

27   public TachesAvecPriorite obtenirTache() {
       return planning.get(jourCourant).obtenirTache();
29   }

31   public void renouveler() {
       Collectino<TachesAvecPriorite> tacheRstantes = ↩
          planning.get(jourCourant).arreter();
```

```
33      planning.remove(jourCourant);
        FileTachesPrioritaires premierJour = planning.get(jourCourant);
35      for(TachesAvecPriorite tache : tacheRstantes) {
          tache.incrementerPriorite();
37        premierJour.ajouterTache(tache);
        }
39
        FileTachesPrioritaires dernierJour = new FileTachesPrioritaires();
41      planning.add(jourCourant, dernierJour);
        jourCourant = (jourCourant+1) % DUREE_PLANNING_EN_JOUR;
43  }

45  public ListIterator<FileTachesPrioritaires> obtenirSousPlanning(int ↩
        premierJour, int dernierJour) {
      return planning.subList(premierJour, dernierJour).listIterator();
47  }
```

Listing 3.6 – `OrdonnanceurTaches`

4 Ensembles : Set

4.1 EnumSet

```
public enum Apprecation {
  EXCEPTIONNEL, TRESBIEN, BIEN, PASSABLE, MAUVAIS, TRESMAUVAIS
}

EnumSet<Apprecation> e1 = e.allOf(Apprecation.class);
EnumSet<Apprecation> e2 = e.range(Apprecation.EXCEPTIONNEL, Apprecation.BIEN);
EnumSet<Apprecation> e3 = e.complementOf(e2);
EnumSet<Apprecation> e34= e.of(Apprecation.EXCEPTIONNEL);
```

Listing 4.1 – EnumSet

4.2 Arbre rouge-noir : TreeSet

```
public class Video extends Comparable<Video> {
  private int annee;
  private String realiateur;
  private String titre;

  public Video(String ptitre, String prealisateur, int pannee) {
    annee = pannee;
    realiateur = prealisateur;
    titre = ptitre;
  }

  public String getTitre() {
    return titre;
  }

  public int getAnnee() {
    return annee;
  }

  public String getRealisateur() {
    return realiateur;
  }

  public String toString() {
    return titre.toString() + " " +
      realiateur.toString() + " "+annee.toString();
  }

  public int compareTo(Video v) {
    int compairaison = titre.compareTo(v.getTitre());

```

```
    if(compairaison != 0) {
      return compairaison;
    }
    compairaison = realiateur.compareTo(v.getRealisateur());

    if(compairaison != 0) {
      return compairaison;
    }

    return (new Integer(annee)).compareTo(new Integer(v.getAnnee()));
  }

  public boolean equals(Object o) {
    if(!o instanceof Video) {
      return false;
    }

    Video v = (Video)o;
    return (titre.equals(v.getTitre()) &&
        realiateur.equals(v.getRealisateur) &&
        annee.equals(o.getAnnee());
  }


}
```

Listing 4.2 – Classe Video

```
public class Test {
  public static void afficherElements(NavigableSet<Video> ens) {
    for(Video v : ens) {
      System.out.println(v);
    }
  }

  public static void afficherElementOrdreInverse(NavigableSet<Video> ens) {
    for(Video v : ens) {
      System.out/println(v);
    }
  }

  public static void main(String[] args) {
    NavigableSet<Video> ens = new TreeSet<Video>();
    ensemble.add(new Video("le jour le plus long", "Ken Annakin", 1962);
    ensemble.add(new Video("Un pont trop loin", "Richard Attenborough",1977);
    ensemble.add(new Video("Platoon", "Olier Stone", 1986);
    ensemble.add(new Video("Full metal jacket", "Stanley Kubrick", 1987);
    ensemble.add(new Video("La ligne rouge", "Terrence Malick", 1962);
    ensemble.add(new Video("The patriot", "Roland Emmerich", 2000);

    afficherElements(ens);

    // tri par réalisateurs
    NavigableSet<Video> ensRealisateur = new TreeSet<>(
      new Comparator<Video>() {
        public int compare(Video v1, Video v2) {
          return v1.getRealisateur().compareTo(v2.getRealisateur());
        }
      });
    ensRealisateur.addAll(ens);
    afficherElements(ensRealisateur);

    // tri par annee
```

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

Listing 4.3 – Affichage

4.3 HashSet

```
1   Set<Video> ensembleAlouer = new HashSet<>(ensembleTrie);
    Set<Video> ensembleLoue = new HashSet<>();

    public int hashcode() {
2     return titre.hashcode() * annee * realisateur.hashcode();
    }
4

6   public boolean equals(Objct o) {
      if(o instanceof Video) {
8       Video videoToCmpare = (Video) o;
```

```
    return (titre.equals(videoToCmpare.getTitre()) &&
          realisateur.equals(videoToCompare.getRealisateur()) &&
          annee == videoToCmpare.annee);
  } else {
    return false
  }
}
```

Listing 4.4 – Ajout du hashCode et equals dans `Video`

```
System.out.prntln("", !ensembleLoue.isEmpty());

// Sans utiliser add
ensembleAlouer.remove(new video("Le jour le plus long", "Ken Annakin", 1962);
ensembleLoue.addAll(ensembleTrie);
ensembleLoue.removeAll(ensembleAlouer);
```

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

```

public boolean equals(Objct o) {
  if(o instanceof VideoAppreciation) {
    VideoAppreciation videoToCmpare = (VideoAppreciation) o;
    return (titre.equals(videoToCmpare.getVideo().getTitre()) &&
          realisateur.equals(videoToCompare.getVideo().getRealisateur()) &&
          annee == videoToCmpare.getVideo().annee);
  if(o instanceof Video) {
    Video videoToCmpare = (Video) o;
    return (titre.equals(videoToCmpare.getTitre()) &&
          realisateur.equals(videoToCompare.getRealisateur()) &&
          annee == videoToCmpare.annee);
  } else {
    return false
  }
}
```

```
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.

A Liste des codes sources