TP n° 2
Licence Informatique (L2)
« Programmation objet avancée »

F. BERTRAND

Année universitaire 2020–2021

Éléments de correction

1 Hiérarchie d'héritage

À partir des classes `BiblioMM`, `CD`, et `DVD` (présentes dans le code source disponible sur Moodle, répertoire `Biblio`) :

1. Construire une hiérarchie d'héritage (comme illustré par la figure 1) en introduisant une classe `EltMM` super-classe de `CD` et `DVD` permettant de factoriser les éléments communs aux deux classes.

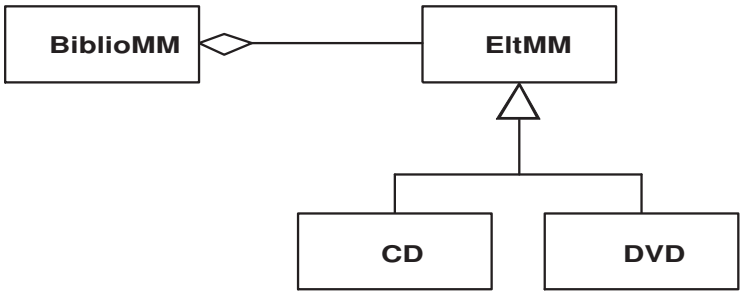

FIGURE 1 – La hiérarchie d'héritage à construire

2. Ajouter dans la classe `EltMM` une méthode `toString()` permettant d'afficher les attributs propres à `EltMM`. Puis utiliser cette méthode dans les versions de `toString()` définies dans `CD` et `DVD`.
3. Modifier la représentation interne de `BiblioMM` de manière à n'avoir plus qu'une seule liste ;
4. Ajouter une méthode `rechercherTitre()` prenant en paramètre une chaîne de caractères, correspondant au titre d'un élément multimédia et retournant une liste d'éléments multimédia dont le titre correspond au titre recherché.
5. Ajouter une méthode `emprunterTitre(String titre)` à la classe `BiblioMM` permettant d'emprunter un élément multimédia. Cette méthode fera appel à la méthode `rechercherTitre()`. L'emprunt ne pourra se faire que si le résultat de la recherche est constitué d'un seul élément et si celui-ci est disponible.

6. Construire une classe de test possédant une méthode `main` qui, après avoir créer une bibliothèque, insérera un `CD` et un `DVD` dans cette bibliothèque, puis testera la méthode `rechercherTitre()` et affichera l'objet retourné. Puis tester deux emprunts successifs d'un même `CD` (sans qu'il soit rendu) en s'assurant que cela n'est pas possible.
7. Définir une méthode `donneType` ayant la signature suivante :

```
1 public String donneType()
```

Cette méthode retournera une chaîne de caractères indiquant le type de l'objet. Par exemple, pour la classe `EltMM`, elle retournera la chaîne de caractères `"EltMM"`.

Tester cette méthode en créant un CD et un DVD avec le même titre puis appeler `rechercherTitre()` avec ce titre en paramètre :
— si un élément du résultat est un `CD`, alors on appellera la méthode `donneArtiste`;
— si un élément du résultat est un `DVD`, alors on appellera `donneRealisateur`.
Exemple :

```
1 ArrayList<EltMM> elts = bibliotheque.rechercherTitre("Moby Dick");
2 for(EltMM elt : elts) {
3   System.out.println(elt.donneType()); // ex. affichage "DVD"
4 }
```

Éléments de correction

Commentaires sur les différentes questions :

1. *Cette question permet de vérifier la bonne compréhension de l'enchaînement des appels des constructeurs lors de la construction d'une instance d'une classe héritant (directement et indirectement) de plusieurs autres;*
2. *Comme la règle générale qui précise que chaque classe doit s'occuper de l'initialisation de ses attributs (via le chaînage des constructeurs avec `super`), cela vaut également pour l'affichage via la méthode `toString()` : comme les attributs hérités sont privés, le seul moyen de les afficher est, dans la méthode `toString` de `CD` (par ex.) de faire appel à la méthode `toString` de la classe `EltMM` (via `super`) dans lesquels ils sont définis;*
3. *Utilisation du type `EltMM` pour stocker à la fois des `CD` et des `DVD` et illustration du principe de substitution de type : une instance de `CD` (ou `DVD`) peut être utilisée lorsqu'une méthode attend une instance de `EltMM`;*
4. *Pour `rechercherTitre` le principe de substitution de type s'applique au type de retour qui est ici défini comme `ArrayList<EltMM>` et qui pourra donc contenir soit des `CD`, soit des `DVD`. Or ce qui sera réellement retourné sera soit un `CD`, soit un `DVD`, tous deux compatible avec le type `EltMM`;*
5. *Rien de particulier à noter ici...*
6. *Toujours le même principe des tests unitaires : on teste ce qu'on reçoit avec ce qu'on devrait recevoir... Si c'est la même chose, OK sinon il y a un problème : soit ce qu'on reçoit est incorrect, soit (plus rare) ce qu'on s'attendait à recevoir est incorrect.*
7. *Ici la méthode `donneType()` permet de vérifier dynamiquement (à l'exécution) le type d'un objet. On verra dans le prochain cours, qu'il existe un opérateur Java permettant de faire cela (déterminer le type dynamique d'une variable). Ici noter l'utilisation du transtypage pour « récupérer » le type original et ainsi pouvoir accéder aux méthodes spécifiques du type original...*

```
1
2   import java.util.ArrayList;
3
4   /**
5    * Cette classe fournit un moyen de stocker des objets
6    * CD et DVD. Une liste de tous les CD et DVD peut etre affichee
7    * en mode texte.
8    *
9    *
10   * @author Michael Kolling and David J. Barnes
11   * @version 2006.03.30
12   */
13  public class BiblioMM
14  {
15
16      private ArrayList<EltMM> items;
17
18      /**
19       * Construit une bibliotheque vide.
20       */
21      public BiblioMM()
22      {
23          this.items = new ArrayList<EltMM>();
24      }
25
26      /**
27       * Ajoute un element multimedia a la bibliotheque.
28       * @param unElt L'element a ajouter.
29       */
30      public void ajouter(EltMM unElt)
31      {
32          this.items.add(unElt);
33      }
34
35      /** Rechercher un element multimedia dans la bibliotheque
36       *
37       * @param titre de l'element recherche
38       * @return une liste d'elements multimedia
39       */
40      public ArrayList<EltMM> rechercherTitre(String titre)
41      {
42          ArrayList<EltMM> resultats = new ArrayList<EltMM>();
43          for (EltMM e : this.items)
44          {
45              if (e.donneTitre().equals(titre))
46              {
47                  resultats.add(e);
48              }
49          }
50          return resultats;
51      }
52
53      /**
54       * Emprunte un element dans la bibliotheque.
55       * @param  eltEmprunte Element a emprunte.
56       * @return true si l'emprunt a pu avoir lieu
57       */
58      public boolean emprunterTitre(String titre)
59      {
60          ArrayList<EltMM> resultats = this.rechercherTitre(titre);
61          if (resultats.size() == 1)
62          {
63              return resultats.get(0).emprunter();
64          }
65          else
66          {
67              return false;
68          }
69      }
70
71      /**
72       * Affiche une liste de tous les CD et DVD actuellement dans
73       * la bibliotheque.
74       */
75      public String toString()
76      {
77          String s = "";
78          // nouvelle forme de la boucle for a partir de la version 1.5
79          for (EltMM elt : items)
```

```
        {
            s += elt + "\n";
        }
        return s;
    }

    public static void main(String[] args)
    {
        BiblioMM biblio = new BiblioMM();
        CD unCD = new CD("The Joshua Tree", "U2", 12, 60);
        CD unAutreCD = new CD("Rattle and Hum", "U2", 12, 60);
        DVD unDVD = new DVD("Apocalypse now", "Coppola", 190);
        DVD unautreDVD = new DVD("Rattle and Hum", "Phil Joanou", 99);
        biblio.ajouter(unCD);
        biblio.ajouter(unAutreCD);
        biblio.ajouter(unDVD);
        biblio.ajouter(unautreDVD);
        System.out.println("Contenu intial de la bibliotheque : \n" + biblio);
        if (biblio.emprunterTitre("The Joshua Tree"))
        {
            System.out.println("l'emprunt a ete effectue...");
        }
        else
        {
            System.out.println("l'emprunt n'a pas ete effectue...");
        }
        System.out.println("Contenu de la bibliotheque apres 1er emprunt : \n" + biblio);
        // test pour voir si on peut emprunter 2 fois le meme element
        if (biblio.emprunterTitre("The Joshua Tree"))
        {
            System.out.println("l'emprunt a ete effectue...");
        }
        else
        {
            System.out.println("l'emprunt n'a pas ete effectue...");
        }
        ArrayList<EltMM> recherche = biblio.rechercherTitre("The Joshua Tree");
        System.out.println("\nResultat de la recherche :");
        for (EltMM e : recherche)
        {
            System.out.println("Element trouve = " + e);
            if (e.donneType().equals("CD"))
            {
                CD cd = (CD) e;
                System.out.println("Artiste = " + cd.donneArtiste());
            }
            else if (e.donneType().equals("DVD"))
            {
                DVD dvd = (DVD) e;
                System.out.println("Realisateur = " + dvd.donneRealisateur());
            }
        }
        recherche = biblio.rechercherTitre("Apocalypse now");
        System.out.println("\nResultat de la recherche :");
        for (EltMM e : recherche)
        {
            System.out.println("Element trouve = " + e);
            if (e.donneType().equals("CD"))
            {
                CD cd = (CD) e;
                System.out.println("Artiste = " + cd.donneArtiste());
            }
            else if (e.donneType().equals("DVD"))
            {
                DVD dvd = (DVD) e;
                System.out.println("Realisateur = " + dvd.donneRealisateur());
            }
        }
        recherche = biblio.rechercherTitre("Rattle and Hum");
        System.out.println("\nResultat de la recherche :");
        for (EltMM e : recherche)
        {
            System.out.println("Element trouve = " + e);
            if (e.donneType().equals("CD"))
            {
                CD cd = (CD) e;
                System.out.println("Artiste = " + cd.donneArtiste());
            }
            else if (e.donneType().equals("DVD"))
            {
```

```
                DVD dvd = (DVD) e;
                System.out.println("Realisateur = " + dvd.donneRealisateur());
            }
        }
    }
}
```

```
/*
 * Classe permettant de regrouper les attributs communs
 * aux classes CD et DVD.
 *
 * @author fbertran
 */
public class EltMM
{
    private String titre;
    private int duree;
    private boolean presentEnRayon;
    private String commentaires;

    /**
     * Initialise un Element MultiMedia.
     *
     * @param unTitre Le titre du CD.
     * @param uneDuree La duree du CD.
     */
    public EltMM(String unTitre, int uneDuree)
    {
        this.titre = unTitre;
        this.duree = uneDuree;
        this.presentEnRayon = true;
        this.commentaires = "<pas de commentaires>";
    }

    /**
     * Donne le titre de l'element.
     *
     * @return titre de l'element.
     */
    public String donneTitre()
    {
        return this.titre;
    }

    /**
     * Donne la dur&eacute;e de l'element.
     *
     * @return dur&eacute;e de l'element.
     */
    public int donneDuree()
    {
        return this.duree;
    }

    /**
     * Ajoute un commentaire a l'element.
     *
     * @param commentaires Les commentaires devant etre ajoutes.
     */
    public void ajouteCommentaires(String commentaires)
    {
        this.commentaires = commentaires;
    }

    /**
     * Donne les commentaires relatif a l'element
     *
     * @return Les commentaires de cet element.
     */
    public String donneCommentaires()
    {
        return this.commentaires;
    }

    /**
     * Fixe l'indicateur pour indiquer si l'element est dans la bibliotheque.
     * @param etat true si l'element est en rayon, false autrement.
     */
    public void changeEtatRayon(boolean etat)
    {
```

```
        this.presentEnRayon = etat;
    }

    /**
     * @return true si l'element est en rayon.
     */
    public boolean donneEtatRayon()
    {
        return this.presentEnRayon;
    }

    /**
     * Permet d'indiquer que l'element a ete emprunte
     * (il n'est plus en rayon).
     */
    public boolean emprunter()
    {
        if (this.presentEnRayon)
        {
            this.changeEtatRayon(false);
            return true;
        }
        else
        {
            return false;
        }
    }

    /**
     * Donne le type de l'objet
     *
     * @return une chaine de caracteres representant le type de l'objet
     */
    public String donneType()
    {
        return "EltMM";
    }

    /**
     * Renvoie une description textuelle de l'objet
     *
     * @return un chaine de caracteres decrivant l'objet
     */
    public String toString()
    {
        String s = "Titre : " + this.titre + "\n";
        s += "Duree : " + this.duree + "\n";
        s += "Etat : ";
        if (this.presentEnRayon)
        {
            s += "disponible\n";
        }
        else
        {
            s += "emprunte\n";
        }
        s += "Commentaires : " + this.commentaires + "\n";
        return s;
    }
}
```

```
/**
 * La classe CD represente un objet CD.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2006.03.30
 */
public class CD extends EltMM
{
    private String artiste;
    private int nbPistes;

    /**
     * Initialise un CD.
     *
     * @param unTitre Le titre du CD.
     * @param unArtiste Le nom de l'artiste du CD.
     * @param pistes Le nombre de pistes du CD.
     * @param uneDuree La duree du CD.
     */
```

```
    public CD(String unTitre, String unArtiste, int pistes, int uneDuree)
    {
        super(unTitre, uneDuree);
        this.artiste = unArtiste;
        this.nbPistes = pistes;
    }

    /**
     * Donne le nom de l'artiste
     *
     * @return nom de l'artiste
     */
    public String donneArtiste()
    {
        return this.artiste;
    }

    /**
     * Donne le nombre de pistes.
     *
     * @return nombre de pistes
     */
    public int donneNbPistes()
    {
        return this.nbPistes;
    }

    /**
     * Donne le type de l'objet
     * @return une chaine de caracteres representant le type de l'objet
     */
    public String donneType()
    {
        return "CD";
    }

    /**
     * Renvoie une description textuelle de l'objet
     *
     * @return un chaine de caracteres decrivant l'objet
     */
    public String toString()
    {
        String s = "CD :\n";
        s += super.toString(); //  affiche les elts communs
        s += "Artiste : " + this.artiste + "\npistes : " + this.nbPistes;
        return s;
    }
}
```

```
/**
 * La classe DVD represente un objet DVD.
 * Nous considerons ici uniquement les DVD Video.
 *
 * @author Michael Kolling and David J. Barnes
 * @version 2006.03.30
 */
public class DVD extends EltMM
{
    private String realisateur;

    /**
     * Initialise un DVD.
     *
     * @param unTitre Le titre du DVD.
     * @param unArtiste Le nom de l'artiste du DVD.
     * @param pistes Le nombre de pistes du DVD.
     * @param uneDuree La duree du DVD.
     */
    public DVD(String unTitre, String unRealisateur, int uneDuree)
    {
        super(unTitre, uneDuree);
        this.realisateur = unRealisateur;
    }

    /**
     * Donne le nom du realisateur
     *
     * @return le nom du realisateur
     */
```

```
31      public String donneRealisateur()
32      {
33          return this.realisateur;
34      }
35
36      /**
37       * Donne le type de l'objet
38       * @return une chaine de caracteres representant le type de l'objet
39       */
40      public String donneType()
41      {
42          return "DVD";
43      }
44
45      /**
46       * Renvoie une description textuelle de l'objet
47       *
48       * @return un chaine de caracteres decrivant l'objet
49       */
50      public String toString()
51      {
52          String s = "DVD :\n";
53          s += super.toString(); //  affiche les elts communs
54          s += "Realisateur : " + this.realisateur;
55          return s;
56      }
57  }
```

2 Transport de marchandises

On souhaite représenter (de manière simpliste) le coût de transport de colis. On dispose d'une classe `Colis` disponible dans le code source fourni.

On souhaite définir une classe `Conteneur` (dont la documentation est fournie, fichier `Conteneur.html`) qui permettra de transporter des colis qui iront tous au même endroit (même distance). La condition de chargement d'un colis dans le conteneur sera de s'assurer que le volume du colis, ajouté aux colis déjà présents, ne dépasse pas le volume maximal (ici le poids n'est pas pris en compte, acheminement par voie maritime). Le coût de transport sera égal à la distance multipliée par le poids.

Puis on souhaite définir une classe `ConteneurUrgent` qui aura pour conditions de chargement d'un colis :

1. la condition de la classe `Conteneur` ;
2. plus celle de ne pas dépasser un poids maximal (les conteneurs urgents seront acheminés par voie aérienne).

Le coût de transport d'un colis pour ce type de conteneur sera égal à deux fois à celui d'un conteneur normal.

Puis définir une classe de test permettant de vérifier que les deux classes `Conteneur` et `ConteneurUrgent`) ont été correctement implémentées (pour chaque méthode, on comparera le résultat de l'appel avec la valeur attendue).

Dans toutes les classes développées, les attributs seront qualifiés `private` et si nécessaire des accesseurs seront définis.

Éléments de correction

Dans cet exercice, l'idée est de vous montrer un exemple de redéfinition de méthodes :

— *la méthode `conditionChargement` qui est redéfinie dans `ConteneurUrgent` mais qui fait appel (via `super`) à la version définie dans `Conteneur` (similaire à l'exemple du cours sur le calcul de l'âge de la retraite pour un pilote par rapport à un salarié) ;*

— *la méthode* `cout` *qui elle également, dans sa redéfinition (*`ConteneurUrgent` *utilise le code de sa version initiale (*`Conteneur`*).*

```

/**
 * Represente un colis pouvant etre d'epos'e dans un conteneur.
 *
 */
public class Colis {

    private int poids;
    private int volume;
    private int numero;
    private static int nbColis = 0; // permet de donner un numero aux colis

    /**
     * Cr'ee un colis.
     *
     * @param poids poids en kg
     * @param volume volume en m3
     */
    public Colis(int poids, int volume) {
        this.poids = poids;
        this.volume = volume;
        this.numero = nbColis++;
    }

    /**
     * Retourne le poids du colis en kg
     *
     * @return le poids du colis
     */
    public int donnePoids() {
        return this.poids;
    }

    /**
     * Retourne le volume du colis en m3.
     *
     * @return le volume du colis
     */
    public int donneVolume() {
        return this.volume;
    }

    public String toString() {
        return "poids = " + this.poids + ", volume = " + this.volume;
    }

    /**
     * Compare deux colis
     * @param o l'autre colis
     * @return true si les deux colis sont identiques
     */
    @Override
    public boolean equals(Object o) {
        if (o == null || o.getClass() != Colis.class) {
            return false;
        } else {
            Colis c = (Colis) o;
            if (this.poids == c.poids
                && this.volume == c.volume
                && this.numero == c.numero) {
                return true;
            } else {
                return false;
            }
        }
    }

}
```

```

import java.util.ArrayList;

/**
 * Repr&eacute;sente un conteneur pouvant contenir des colis.
 */
public class Conteneur {
```

```
8
9       private ArrayList<Colis> contenu;
10      private int distance;
11      private int poids;
12      private int volume;
13      private int volumeMax;
14
15      /**
16       * Cr&eacute;e un conteneur
17       *
18       * @param distance distance &agrave; parcourir
19       * @param volumeMax volume maximal des colis
20       */
21      public Conteneur(int distance, int volumeMax) {
22          this.contenu = new ArrayList<Colis>();
23          this.distance = distance;
24          this.volumeMax = volumeMax;
25          this.poids = 0;
26          this.volume = 0;
27      }
28
29      /**
30       * Permet d'ajouter un colis dans le conteneur
31       *
32       * @param c le colis a ajouter
33       * @return true si le colis a pu &ecirc;tre ajout&eacute;
34       */
35      public boolean ajout(Colis c) {
36          if (this.conditionChargement(c)) {
37              this.contenu.add(c);
38              this.poids += c.donnePoids();
39              this.volume += c.donneVolume();
40              return true;
41          } else {
42              return false;
43          }
44      }
45
46      /**
47       * Calcul du cout de transport du conteneur : distance x poids
48       *
49       * @return le cout de transport
50       */
51      public int cout() {
52          return this.distance * this.poids;
53      }
54
55      /**
56       * V&eacute;rifie que le colis respecte les contraintes du conteneur. Cette
57       * m&eacute;thode est utilis&eacute;e par {@link #ajout}. Cette
58       * m&eacute;thode devra etre redefinie dans <code>ConteneurUrgent</code>
59       *
60       * @param c le colis &agrave; tester
61       * @return true si le colis peut &ecirc;tre ajout&eacute; dans le conteneur
62       */
63      public boolean conditionChargement(Colis c) {
64          return (this.volume + c.donneVolume() <= this.volumeMax);
65      }
66
67      /**
68       * Retourne la distance &agrave; parcourir
69       *
70       * @return la distance a parcourir par le conteneur
71       */
72      public int donneDistance() {
73          return this.distance;
74      }
75
76      /**
77       * Retourne le poids courant du conteneur
78       *
79       * @return le poids courant
80       */
81      public int donnePoids() {
82          return this.poids;
83      }
84
85      @Override
86      public String toString() {
87          return "contenu = " + this.contenu + "\n"
```

```
                + "poids = " + this.poids + "\n"
                + "volume = " + this.volume;
    }
}
```

```

public class ConteneurUrgent extends Conteneur {

    private int poidsMax;

    public ConteneurUrgent(int distance, int poidsMax, int volumeMax) {
        super(distance, volumeMax);
        this.poidsMax = poidsMax;
    }

    /**
     * Cette methode r'eutilise et red'efinit la version h'erit'ee de la classe.
     * <code>Conteneur</code>
     *
     * @return true si les conditions sont v'erifi'e'es, false sinon
     */
    @Override
    public boolean conditionChargement(Colis c) {
        return super.conditionChargement(c)
                && (super.donnePoids() + c.donnePoids() <= this.poidsMax);
    }

    /**
     * Cette methode r'eutilise et red'efinit la version h'erit'ee de la classe.
     * <code>Conteneur</code>
     *
     * @return le cout de transport en conteneur urgent
     */
    @Override
    public int cout() {
        return 2 * super.cout();
    }

    /**
     *
     * @return
     */
    public int donnePoidsMax() {
        return this.poidsMax;
    }
}
```

```

public class TestConteneur {

    public static void main(String[] args) {
        // a completer pour qu'elle soit plus exhaustive
        Colis m1 = new Colis(10000, 20);
        Colis m2 = new Colis(20000, 30);
        Conteneur c = new Conteneur(1000, 60);

        if (!c.ajout(m1)) {
            System.out.println("Probleme ajout colis " + m1);
        }
        if (!c.ajout(m2)) {
            System.out.println("Probleme ajout colis " + m2);
        }
        int cout = c.cout();
        if (cout != 30000000) {
            System.out.println("Probleme calcul cout : " + c);
        }

        ConteneurUrgent cu = new ConteneurUrgent(1000, 15000, 60);

        if (!cu.ajout(m1)) {
            System.out.println("Probleme ajout colis " + m1);
        }
        if (cu.ajout(m2)) {
            System.out.println("Probleme ajout colis " + m2);
        }
        int coutUrgent = cu.cout();
        if (coutUrgent != 20000000) {
            System.out.println("Probleme calcul cout : " + coutUrgent);
        }
```

```
33
34      }
35  }
```