TD 6 - Les associations

Programmation Orientée Objet

Objectif

– Comprendre le principe de l'implémentation d'une association et d'une composition en C++

1 Première partie

Ce sujet est tiré d'un exercice de Benoit Charroux.

Soit le diagramme des classes de la figure 1 modélisant une médiathèque où des adhérents peuvent faire 3 emprunts (ils empruntent un exemplaire d'une œuvre donnée). L'ensemble des adhérents est stocké dans une classe `Adherents`, et l'ensemble des œuvres de la médiathèque est conservé dans une classe `Œuvres`. Il y a deux types d'œuvres : des œuvres interprétées (limitées ici à des CD), et des œuvres non interprétées (uniquement des livres dans notre cas).

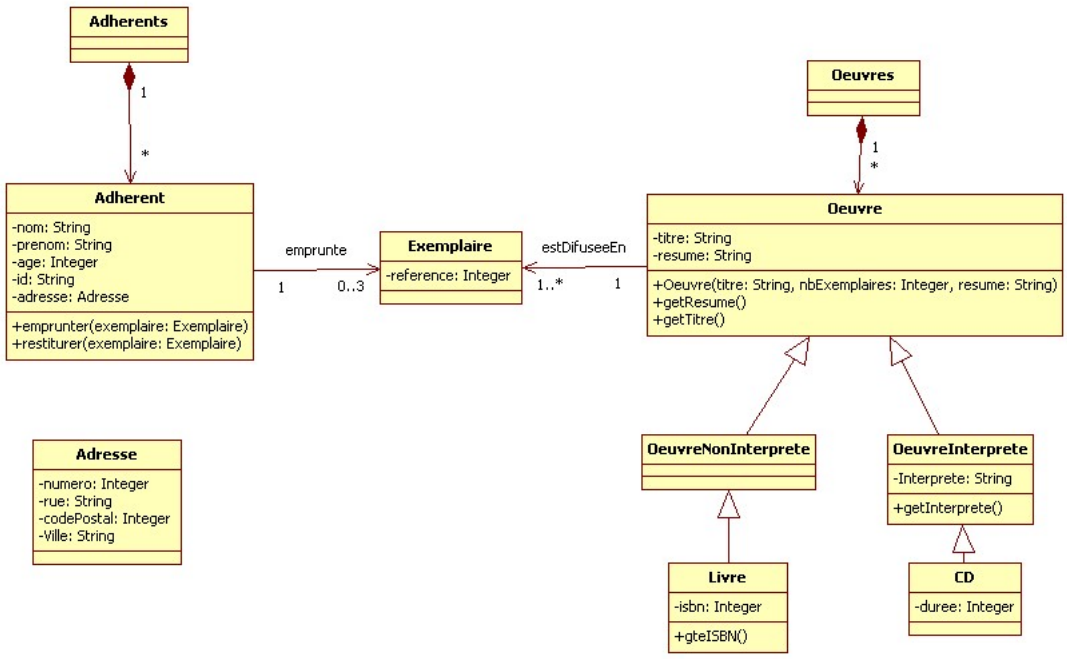

FIGURE 1 – Diagramme de classe de l'application

Ecrivez la classe `Adherent`. Ajoutez-y un constructeur pour initialiser les données membres et éventuellement un destructeur.

Ecrivez la classe `Exemplaire` avec un constructeur pour initialiser la donnée membre reference.

Pour modéliser l'association unidirectionnelle `emprunte` entre les classes `Adherent` et `Exemplaire`, vous pouvez utiliser un tableau de 3 pointeurs. Pour mettre à jour cette association, ajoutez dans la classe `Adherent` une fonction membre appelée `emprunter`, qui ajoute un exemplaire à un adhérent, simulant ainsi un emprunt.

Ajoutez aussi une fonction appelée **restituer** qui modélise la restitution d'un exemplaire dans la médiathèque par un adhérent.

Complétez le programme suivant, puis testez vos classes avec :

```
void main(){
    Adherent belloir( ... ) ;
    Exemplaire exp1( ... ) ;
    belloir.emprunter( &exp1 ) ;
    belloir.restituer( exp1 ) ;
}
```

On s'intéresse à présent à la relation de composition entre les classes **Adherents** et **Adherent**. Bien que ce soit une relation de type composé / composant, il est possible de la modéliser avec des pointeurs sur des adhérents. Il faudra cependant veiller à écrire correctement le destructeur de la classe **Adherents** pour qu'il détruise l'ensemble des adhérents.

Ecrivez la classe **Adherents**. Par simplicité, vous pouvez modéliser la relation de composition avec la classe **Adherent** par un tableau de pointeurs. Ajoutez à la classe **Adherents** des fonctions appelées **addAdherent** et **removeAdherent** permettant d'ajouter et de supprimer respectivement un adhérent de la médiathèque.

Ecrire un programme de test.

```
#ifndef ADHERENT_H
#define ADHERENT_H
#include <iostream>
using namespace std;

#include "Adresse.h"
#include "Exemplaire.h"

#define NB_EXEMPLAIRES 3

class Adherent{
        string sNom;
        string sPrenom;
        Adresse adresse;
        int iAge;
        int iId;
        int iNbExemplairesEmpruntes;
        Exemplaire * emprunte[NB_EXEMPLAIRES];
public:
        Adherent();
        Adherent(string, string, Adresse, int, int=-1);
        Adherent(const Adherent &);
        Adherent & operator=(const Adherent &);
        ~Adherent();
//      int addExemplaire(const Exemplaire * const ); /* pointeur sur un Exemplaire constant*/
//      int addExemplaire(Exemplaire * const ); /* pointeur constant sur un Exemplaire */
        int emprunter(Exemplaire * const);
        int restituer(Exemplaire * const );
        friend ostream& operator << (ostream&, const Adherent&);
};

#endif
```

```
#include "Adherent.h"
/*******************************/
Adherent::Adherent(){
}
/*******************************/
Adherent::Adherent(string n, string p, Adresse ad, int a, int id): sNom(n),     sPrenom(p), adresse(ad), iAge(
        iNbExemplairesEmpruntes=0;
        for (int i=0; i < NB_EXEMPLAIRES; i++){
                emprunte[i]=NULL;
        }
}
/*******************************/
Adherent::Adherent(const Adherent &adr): sNom(adr.sNom),sPrenom(adr.sPrenom), adresse(adr.adresse), iAge(adr.i
        iNbExemplairesEmpruntes=adr.iNbExemplairesEmpruntes;
        for (int i=0; i < NB_EXEMPLAIRES; i++){
                emprunte[i]=adr.emprunte[i];
        }
}
/*******************************/
Adherent & Adherent::operator=(const Adherent & adr){
        sNom = adr.sNom;
        sPrenom = adr.sPrenom;
        adresse = adr.adresse;
        iAge = adr.iAge;
```

```
        iId = adr.iId;
        for (int i=0; i < iNbExemplairesEmpruntes; i++){
                emprunte[i]->rendre();
        }
        iNbExemplairesEmpruntes=adr.iNbExemplairesEmpruntes;
        for (int i=0; i < iNbExemplairesEmpruntes; i++){
                emprunte[i]=adr.emprunte[i];
        }
        return *this;
}
/*******************************/
Adherent::~Adherent(){
        for(int i=0;i<iNbExemplairesEmpruntes;i++){
                emprunte[i]->rendre();
        }
}
/*******************************/
int Adherent::emprunter(Exemplaire * const e){
        if (iNbExemplairesEmpruntes==2)
                return -1;
        emprunte [iNbExemplairesEmpruntes] = e;
        iNbExemplairesEmpruntes++;
        return 0;
}
/*******************************/
int Adherent::restituer(Exemplaire * const e){
        if (iNbExemplairesEmpruntes==0)
                return 0;
        for (int i=0; i < NB_EXEMPLAIRES; i++){
                if (emprunte[i]==e){
                        /* exemplaire trouve */
                        emprunte[i]->rendre();
                        emprunte[i]=NULL;
                        iNbExemplairesEmpruntes--;

                        for (int j=i; j < NB_EXEMPLAIRES-1; j++)
                        {
                                emprunte[j]=emprunte[j+1];
                                emprunte[j+1]=NULL;
                        }
                        return 0;
                }
        }
        return -1;
}
/*******************************/
ostream& operator << (ostream& os, const Adherent& ad){
        os << ad.sNom << " " << ad.sPrenom ;
        os << " a emprunte " << ad.iNbExemplairesEmpruntes << " exemplaire (s) :\n";
        for (int i =0; i< NB_EXEMPLAIRES;i++){
                if (ad.emprunte [i] != NULL)
                        os << "\t" << *(ad.emprunte [i]);
        }
        return os;
}
```

```
#ifndef EXEMPLAIRE_H
#define EXEMPLAIRE_H
#include <iostream>
using namespace std;


class Exemplaire{
        int iReference;
        bool bLibre;
public:
        Exemplaire(int ref);
        Exemplaire ();
        Exemplaire (const Exemplaire &);
        Exemplaire & operator=(const Exemplaire &);
        ~Exemplaire();

        int getReference() const;
        void setReference(int);

        bool estLibre() const;
        void emprunter();
        void rendre();

        friend ostream& operator << (ostream& os, const Exemplaire& e);
};
```

```
#endif
```

```
#include "Exemplaire.h"

/*********************************/
Exemplaire::Exemplaire (int ref):iReference(ref),bLibre(true){}
/*********************************/
Exemplaire::Exemplaire (){
        iReference=-1;
        bLibre=true;
}
/*********************************/
Exemplaire::Exemplaire (const Exemplaire & e):iReference(e.iReference),bLibre(e.bLibre){}
/*********************************/
Exemplaire & Exemplaire::operator=(const Exemplaire & e){
        iReference=e.iReference;
        bLibre=e.bLibre;
        return *this;
}
/*********************************/
Exemplaire::~Exemplaire(){
}
/*********************************/
int Exemplaire::getReference() const{
        return(iReference);
}
/*********************************/
void Exemplaire::setReference(int Reference){
        iReference=Reference;
}
/*********************************/
bool Exemplaire::estLibre() const{
        return bLibre;
}

/*********************************/
void Exemplaire::emprunter(){
        bLibre = false;
}
/*********************************/
void Exemplaire::rendre(){
        bLibre = true;
}
/*********************************/
ostream& operator << (ostream& os, const Exemplaire& e){
        os << "Exp : "<<e.iReference;
        if (e.estLibre())
                os<< "Libre ";
        else
                os<< "Sorti ";
        os<< endl;
        return os;
}
```

```
#ifndef ADHERENTS_H
#define ADHERENTS_H
#include <iostream>
using namespace std;

#include "Adherent.h"

#define NB_ADH_DEFAULT 10        // Nombre d adherent par defaut

class Adherents{
        int iNbAdh;
        Adherent * lesAdherents[NB_ADH_DEFAULT];
public:
        Adherents(int iNbA=0);
        ~Adherents();

        void addAdherent(Adherent * );
        void removeAdherent(Adherent * const );
        int getNbAdh() const;
        friend ostream& operator << (ostream&, const Adherents&);
};
#endif
```

```
#include "Adherents.h"
/*********************************/
```

```
Adherents::Adherents(int iNbA):iNbAdh(iNbA){}
/*********************************/
Adherents::~Adherents(){
        for (int i=0; i<iNbAdh;i++)
                if (lesAdherents[i]!=NULL)
                        delete lesAdherents[i];
}
/*********************************/
void Adherents::addAdherent(Adherent * const adh){
        lesAdherents[iNbAdh]= adh;
        iNbAdh++;
}
/*********************************/
void Adherents::removeAdherent( Adherent * const a){
        for (int i=0; i < NB_ADH_DEFAULT; i++){
                if (lesAdherents[i]==a){
                        /* exemplaire trouve */
                        lesAdherents[i]=NULL;
                        iNbAdh--;

                        for (int j=i; j < NB_ADH_DEFAULT-1; j++)
                        {
                                /* decalage vers la gauche */
                                lesAdherents[j]=lesAdherents[j+1];
                                lesAdherents[j+1]=NULL;
                        }
                }
        }
}
/*********************************/
int Adherents::getNbAdh() const{
        return iNbAdh;
}
/*********************************/
ostream& operator << (ostream & os, const Adherents & a){
        os << "Liste des Adherents ("<< a.getNbAdh() <<" adherents): \n";
        for (int i=0; i<a.getNbAdh();i++)
                os << "\t" << (*(a.lesAdherents[i])) << "\n";
        return os;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "Adherent.h"
#include "Adherents.h"
#include "Livre.h"
#include "CD.h"

int main(void) {
        //Test classes Adherent, Adherents et Exemplaire
        Adresse a(10, "du Jardin", 64000, "Pau");
        Adherent belloir("Belloir", "Nicolas", a, 33, 1234);
        Adresse b(12, "du Manoir", 64530, "Pontacq");
        Adherent marcillac("Marcillac", "Charly", a, 35, 1247);

        Exemplaire exp1 (1);
        Exemplaire exp2 (2);

        Exemplaire exp3 (3);

        belloir.emprunter(&exp1);
        belloir.emprunter(&exp3);
        cout << belloir;

        marcillac.emprunter(&exp2);
        cout << marcillac;

        Adherents DptInfo;
        DptInfo.addAdherent (&belloir);
        DptInfo.addAdherent (&marcillac);

        cout << DptInfo;

        DptInfo.removeAdherent(&belloir);
        DptInfo.removeAdherent(&marcillac);

/*
        Livre liv1("Capital de la douleur", 1, "Recueuil de poeme de Paul Eluard", 1234556778);
        CD cd1("Le coquelicot", 2, "Deuxieme album de Jamait", "Jamait", 48);
        cout << liv1;
        cout << cd1;
```

```
*/
        cout << "Fin programme" <<endl;

        return EXIT_SUCCESS;
}
```

2 Deuxième partie

Ecrivez à présent la classe `Œuvre` ainsi que ses classes dérivées

Comment modéliser l'association de 1 à n entre les classes `Œuvre` et `Exemplaire` ? Cette association doit être mise à jour dès la création d'une oeuvre comme dans le programme suivant où une nouvelle œuvre portant le titre de Don Juan et diffusée à 2 exemplaires est créée. C'est le constructeur de la classe `Œuvre` qui se charge de créer les exemplaires :

```
    void main(){
        OEuvre donJuan(``Don Juan'', 2, ``C'est l'histoire...'' ) ;
    }
```

On remarque que l'association de 1 à n entre les classes `Œuvre` et `Exemplaire` doit être mise à jour quand un adhérent emprunte un exemplaire : il faut alors commencer par mettre à jour la relation entre un adhérent et l'exemplaire qu'il désire, puis mettre à jour la relation entre l'exemplaire et l'oeuvre correspondante (modélisant ainsi qu'il y a un exemplaire de moins dans la médiathèque, et un exemplaire de plus pour un adhérent) ! Ecrivez alors les fonctions membres `emprunter` et `restituer` de la classe `Adherent`.

```
#ifndef ADHERENT_H
#define ADHERENT_H
#include <iostream>
using namespace std;

#include "Adresse.h"
#include "Exemplaire.h"

#define NB_EXEMPLAIRES 3

class Adherent{
        string sNom;
        string sPrenom;
        Adresse adresse;
        int iAge;
        int iId;
        int iNbExemplairesEmpruntes;
        Exemplaire * emprunte[NB_EXEMPLAIRES];
public:
        Adherent();
        Adherent(string, string, Adresse, int, int=-1);
        Adherent(const Adherent &);
        Adherent & operator=(const Adherent &);
        ~Adherent();
//      int addExemplaire(const Exemplaire * const ); /* pointeur sur un Exemplaire constant*/
//      int addExemplaire(Exemplaire * const ); /* pointeur constant sur un Exemplaire */
        int emprunter(Exemplaire * const);
        int restituer(Exemplaire * const );
        friend ostream& operator << (ostream&, const Adherent&);
};

#endif
```

```
#include "Adherent.h"
/*******************************/
Adherent::Adherent(){
}
/*******************************/
Adherent::Adherent(string n, string p, Adresse ad, int a, int id): sNom(n),     sPrenom(p), adresse(ad), iAge(
        iNbExemplairesEmpruntes=0;
        for (int i=0; i < NB_EXEMPLAIRES; i++){
                emprunte[i]=NULL;
        }
}
/*******************************/
Adherent::Adherent(const Adherent &adr): sNom(adr.sNom),sPrenom(adr.sPrenom), adresse(adr.adresse), iAge(adr.i
        iNbExemplairesEmpruntes=adr.iNbExemplairesEmpruntes;
        for (int i=0; i < NB_EXEMPLAIRES; i++){
                emprunte[i]=adr.emprunte[i];
```

```
        }
}
/*******************************/
Adherent & Adherent::operator=(const Adherent & adr){
        sNom = adr.sNom;
        sPrenom = adr.sPrenom;
        adresse = adr.adresse;
        iAge = adr.iAge;
        iId = adr.iId;
        for (int i=0; i < iNbExemplairesEmpruntes; i++){
                emprunte[i]->rendre();
        }
        iNbExemplairesEmpruntes=adr.iNbExemplairesEmpruntes;
        for (int i=0; i < iNbExemplairesEmpruntes; i++){
                emprunte[i]=adr.emprunte[i];
        }
        return *this;
}
/*******************************/
Adherent::~Adherent(){
        for(int i=0;i<iNbExemplairesEmpruntes;i++){
                emprunte[i]->rendre();
        }
}
/*******************************/
int Adherent::emprunter(Exemplaire * const e){
        if (iNbExemplairesEmpruntes==2)
                return -1;
        emprunte [iNbExemplairesEmpruntes] = e;
        iNbExemplairesEmpruntes++;
        return 0;
}
/*******************************/
int Adherent::restituer(Exemplaire * const e){
        if (iNbExemplairesEmpruntes==0)
                return 0;
        for (int i=0; i < NB_EXEMPLAIRES; i++){
                if (emprunte[i]==e){
                        /* exemplaire trouve */
                        emprunte[i]->rendre();
                        emprunte[i]=NULL;
                        iNbExemplairesEmpruntes--;

                        for (int j=i; j < NB_EXEMPLAIRES-1; j++)
                        {
                                emprunte[j]=emprunte[j+1];
                                emprunte[j+1]=NULL;
                        }
                        return 0;
                }
        }
        return -1;
}
/*******************************/
ostream& operator << (ostream& os, const Adherent& ad){
        os << ad.sNom << " " << ad.sPrenom ;
        os << " a emprunte " << ad.iNbExemplairesEmpruntes << " exemplaire (s) :\n";
        for (int i =0; i< NB_EXEMPLAIRES;i++){
                if (ad.emprunte [i] != NULL)
                        os << "\t" << *(ad.emprunte [i]);
        }
        return os;
}
```

```
#ifndef EXEMPLAIRE_H
#define EXEMPLAIRE_H
#include <iostream>
using namespace std;


class Exemplaire{
        int iReference;
        bool bLibre;
public:
        Exemplaire(int ref);
        Exemplaire ();
        Exemplaire (const Exemplaire &);
        Exemplaire & operator=(const Exemplaire &);
        ~Exemplaire();

        int getReference() const;
```

```
        void setReference(int);

        bool estLibre() const;
        void emprunter();
        void rendre();

        friend ostream& operator << (ostream& os, const Exemplaire& e);
};

#endif
```

```
#include "Exemplaire.h"

/*********************************/
Exemplaire::Exemplaire (int ref):iReference(ref),bLibre(true){}
/*********************************/
Exemplaire::Exemplaire (){
        iReference=-1;
        bLibre=true;
}
/*********************************/
Exemplaire::Exemplaire (const Exemplaire & e):iReference(e.iReference),bLibre(e.bLibre){}
/*********************************/
Exemplaire & Exemplaire::operator=(const Exemplaire & e){
        iReference=e.iReference;
        bLibre=e.bLibre;
        return *this;
}
/*********************************/
Exemplaire::~Exemplaire(){
}
/*********************************/
int Exemplaire::getReference() const{
        return(iReference);
}
/*********************************/
void Exemplaire::setReference(int Reference){
        iReference=Reference;
}
/*********************************/
bool Exemplaire::estLibre() const{
        return bLibre;
}

/*********************************/
void Exemplaire::emprunter(){
        bLibre = false;
}
/*********************************/
void Exemplaire::rendre(){
        bLibre = true;
}
/*********************************/
ostream& operator << (ostream& os, const Exemplaire& e){
        os << "Exp : "<<e.iReference;
        if (e.estLibre())
                os<< "Libre ";
        else
                os<< "Sorti ";
        os<< endl;
        return os;
}
```

```
#ifndef ADHERENTS_H
#define ADHERENTS_H
#include <iostream>
using namespace std;

#include "Adherent.h"

#define NB_ADH_DEFAULT 10        // Nombre d adherent par defaut

class Adherents{
        int iNbAdh;
        Adherent * lesAdherents[NB_ADH_DEFAULT];
public:
        Adherents(int iNbA=0);
        ~Adherents();

        void addAdherent(Adherent * );
        void removeAdherent(Adherent * const );
```

```
        int getNbAdh() const;
        friend ostream& operator << (ostream&, const Adherents&);
};
#endif
```

```
#include "Adherents.h"
/***********************************/
Adherents::Adherents(int iNbA):iNbAdh(iNbA){}
/***********************************/
Adherents::~Adherents(){
        for (int i=0; i<iNbAdh;i++)
                if (lesAdherents[i]!=NULL)
                        delete lesAdherents[i];
}
/***********************************/
void Adherents::addAdherent(Adherent * const adh){
        lesAdherents[iNbAdh]= adh;
        iNbAdh++;
}
/***********************************/
void Adherents::removeAdherent( Adherent * const a){
        for (int i=0; i < NB_ADH_DEFAULT; i++){
                if (lesAdherents[i]==a){
                        /* exemplaire trouve */
                        lesAdherents[i]=NULL;
                        iNbAdh--;

                        for (int j=i; j < NB_ADH_DEFAULT-1; j++)
                        {
                                /* decalage vers la gauche */
                                lesAdherents[j]=lesAdherents[j+1];
                                lesAdherents[j+1]=NULL;
                        }
                }
        }
}
/***********************************/
int Adherents::getNbAdh() const{
        return iNbAdh;
}
/***********************************/
ostream& operator << (ostream & os, const Adherents & a){
        os << "Liste des Adherents ("<< a.getNbAdh() <<" adherents): \n";
        for (int i=0; i<a.getNbAdh();i++)
                os << "\t" << (*(a.lesAdherents[i])) << "\n";
        return os;
}
```

```
#ifndef OEUVRE_H
#define OEUVRE_H
#include <iostream>
using namespace std;
#include "Exemplaire.h"

class Oeuvre{
        string sTitre;
        string sResume;
        int iNbExemplaires;
        Exemplaire * EstDiffuseEn;  //tableau d'exemplaires
public:
        Oeuvre();
        ~Oeuvre();
        Oeuvre(string, int, string);
        Oeuvre(const Oeuvre&);
        Oeuvre & operator=(const Oeuvre &);

        string getResume() const;
        string getTitre() const;
        int getNbExemplaires() const;

        friend ostream& operator << (ostream&, Oeuvre&);
        virtual void afficher(ostream & os);
};
#endif
```

```
#include "Oeuvre.h"

Oeuvre::Oeuvre(){
        sTitre="";
        iNbExemplaires=0;
```

```
        sResume="";
        EstDiffuseEn=NULL;
}

Oeuvre::~Oeuvre(){
        delete [] EstDiffuseEn;
}

Oeuvre::Oeuvre(string tit, int iNbEx, string res){
        sTitre=tit;
        sResume=res;
        iNbExemplaires=iNbEx;
        EstDiffuseEn = new Exemplaire [iNbExemplaires];
}

Oeuvre::Oeuvre(const Oeuvre &o){
        sTitre=o.sTitre;
        sResume=o.sResume;
        iNbExemplaires=o.iNbExemplaires;
        for (int i; i< o.iNbExemplaires; i++)
                EstDiffuseEn[i]=o.EstDiffuseEn[i];
}

Oeuvre & Oeuvre::operator=(const Oeuvre & o){
        sTitre=o.sTitre;
        sResume=o.sResume;
        iNbExemplaires=o.iNbExemplaires;
        for (int i; i< o.iNbExemplaires; i++)
                EstDiffuseEn[i]=o.EstDiffuseEn[i];

        return *this;
}

string Oeuvre::getTitre() const{
        return sTitre;
}

string Oeuvre::getResume() const{
        return sResume;
}

int Oeuvre::getNbExemplaires() const{
        return iNbExemplaires;
}

/*
*************************************************************************
PROBLEME DU POLYMORPHISME POUR L OPERATEUR DE SORTIE
*************************************************************************
* Le probleme est que ma surcharge de l operator<< se fait grâce à une
* fonction friend. Or on ne peut pas surcharger une fonction friend. Si on
* veut du polymorphisme sur l'affichage il faut donc passer par un autre
* moyen. La solution est de :
*   - surcharger la fonction operator<< au niveau de la classe mere
*       - appeler une fonction polymorphe virtuelle (dans la classe mere) par
*               operator<<.
*       - developper les fonctions polymorphes appelee par operator<<.
* Dans l exemple suivant, la fonction operator<< appelle la fonction afficher()
* et celle-ci est declaree en virtuelle dans Oeuvre. Elle est implementee par
* exemple dans la classe Livre.
*************************************************************************
*/
ostream& operator << (ostream & os, Oeuvre & o){
        o.afficher(os);
        return os;
}

void Oeuvre::afficher(ostream & os){
        os << "Oeuvre : " << sTitre << "\n---nb exemplaires : " << iNbExemplaires << "\n---Resume :" << sResum
}
```

```
#ifndef OEUVRE_NON_INTERPRETEE_H
#define OEUVRE_NON_INTERPRETEE_H
#include <iostream>
using namespace std;

#include "Oeuvre.h"

class OeuvreNonInterpretee: public Oeuvre{
public:
        OeuvreNonInterpretee(string, int, string);
```

```
        virtual void afficher(ostream & os);
};
#endif
```

```
#include "OeuvreNonInterpretee.h"

OeuvreNonInterpretee::OeuvreNonInterpretee(string t, int nb, string resume):Oeuvre(t, nb, resume){
}

void OeuvreNonInterpretee::afficher(ostream & os){
        Oeuvre::afficher(os);
}
```

```
#ifndef LIVRE_H
#define LIVRE_H
#include <iostream>
using namespace std;

#include "OeuvreNonInterpretee.h"

class Livre: public OeuvreNonInterpretee{
        int iISBN;
public:
        Livre(string , int , string , int );
        int getISBN() const;
//      friend ostream& operator << (ostream&, const Livre&);
        void afficher(ostream & os);
};
#endif
```

```
#include "Livre.h"

Livre::Livre(string t, int nb, string resume, int isbn):OeuvreNonInterpretee(t, nb, resume){
        iISBN=isbn;
}

int Livre::getISBN() const{
        return iISBN;
}
void Livre::afficher(ostream & os){
        Oeuvre::afficher(os);
        os << "---ISBN : " << getISBN() << "\n";
}
```

```
#ifndef OEUVRE_INTERPRETEE_H
#define OEUVRE_INTERPRETEE_H
#include <iostream>
using namespace std;

#include "Oeuvre.h"

class OeuvreInterpretee: public Oeuvre{
        string sInterprete;
public:
        OeuvreInterpretee(string, int, string, string);
        string getInterprete() const;
        virtual void afficher(ostream & os);
};
#endif
```

```
#include "OeuvreInterpretee.h"

OeuvreInterpretee::OeuvreInterpretee(string t, int nb, string resume, string i):Oeuvre(t, nb, resume){
        sInterprete = i;
}

void OeuvreInterpretee::afficher(ostream & os){
        Oeuvre::afficher(os);
        os << "---Interprete : " << getInterprete() << "\n";
}

string OeuvreInterpretee::getInterprete() const{
        return sInterprete;
}
```

```
#ifndef CD_H
#define CD_H
#include <iostream>
using namespace std;
```

```
#include "OeuvreInterpretee.h"

class CD: public OeuvreInterpretee{
        int iDuree;
public:
        CD(string , int , string , string, int);
        int getDuree() const;
        void afficher(ostream & os);
};
#endif
```

```
#include "CD.h"

CD::CD(string t, int nb, string resume, string interp, int d):OeuvreInterpretee(t, nb, resume, interp){
        iDuree=d;
}

int CD::getDuree() const{
        return iDuree;
}
void CD::afficher(ostream & os){
        OeuvreInterpretee::afficher(os);
        os << "---Duree : " << getDuree() << "\n";
}
```

```
#ifndef OEUVRES_H
#define OEUVRES_H
#include <iostream>
using namespace std;

#include "Oeuvre.h"
#define NB_MAX_OEUVRES 100      // Nombre d oeuvre par defaut
class Oeuvres{
        Oeuvre * lesOeuvres[NB_MAX_OEUVRES];
        int iNbOeuvres;
public:
        Oeuvres(int iNbOeuvres=0);
        ~Oeuvres();
int addOeuvre(Oeuvre * const o);
int removeOeuvre(Oeuvre * const o);
//      friend ostream& operator << (ostream&, const Oeuvres&);
};
#endif
```

```
#include "Oeuvres.h"

Oeuvres::Oeuvres(int iNbO){
        iNbOeuvres = iNbO;
}
Oeuvres::~Oeuvres(){
        for (int i=0; i<iNbOeuvres;i++)
                if (lesOeuvres[i]!=NULL)
                        delete lesOeuvres[i];
}

int Oeuvres::addOeuvre(Oeuvre * const o){
        if (iNbOeuvres!=NB_MAX_OEUVRES){
                lesOeuvres[iNbOeuvres]= o;
                iNbOeuvres++;
                return 0;
        }
        return -1;

}
int Oeuvres::removeOeuvre(Oeuvre * const o){
        if (iNbOeuvres==0)
                return 0;
        for (int i=0; i < NB_MAX_OEUVRES; i++){
                if (lesOeuvres[i]==o){
                        /* exemplaire trouve */
                        lesOeuvres[i]=NULL;
                        iNbOeuvres--;

                        for (int j=i; j < NB_MAX_OEUVRES-1; j++)
                        {
                                /* decalage vers la gauche */
                                lesOeuvres[j]=lesOeuvres[j+1];
                                lesOeuvres[j+1]=NULL;
                        }
```

```
                        return 0;
                }
        }
        return -1;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "Adherent.h"
#include "Adherents.h"
#include "Livre.h"
#include "CD.h"

int main(void) {
        //Test classes Adherent, Adherents et Exemplaire
        Adresse a(10, "du Jardin", 64000, "Pau");
        Adherent belloir("Belloir", "Nicolas", a, 33, 1234);
        Adresse b(12, "du Manoir", 64530, "Pontacq");
        Adherent marcillac("Marcillac", "Charly", a, 35, 1247);

        Exemplaire exp1 (1);
        Exemplaire exp2 (2);

        Exemplaire exp3 (3);

        belloir.emprunter(&exp1);
        belloir.emprunter(&exp3);
        cout << belloir;

        marcillac.emprunter(&exp2);
        cout << marcillac;

        Adherents DptInfo;
        DptInfo.addAdherent (&belloir);
        DptInfo.addAdherent (&marcillac);

        cout << DptInfo;

        DptInfo.removeAdherent(&belloir);
        DptInfo.removeAdherent(&marcillac);

/*
        Livre liv1("Capital de la douleur", 1, "Recueuil de poeme de Paul Eluard", 1234556778);
        CD cd1("Le coquelicot", 2, "Deuxieme album de Jamait", "Jamait", 48);
        cout << liv1;
        cout << cd1;

*/
        cout << "Fin programme" <<endl;

        return EXIT_SUCCESS;
}
```