MOOC Intro. POO C++
Corrigés semaine 7

Les corrigés proposés correspondent à l'ordre des apprentissages : chaque corrigé correspond à la solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante.

Exercice 23 : fichiers et programmes

> Cet exercice correspond à l'exercice n°71 (pages 183 et 379) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

```
#include <iostream>
#include <string>
using namespace std;

class Document {
public:
  Document(int, string, string);
  virtual ~Document();
  void afficher() const;

private:
  int taille;
  string nom;
  string extension;
};

Document::Document(int taille, string nom, string extension)
  : taille(taille), nom(nom), extension(extension)
{
  cout << "creation d'un document" << endl;
}

Document::~Document()
{
 cout << "destruction d'un document" << endl;
}

void Document::afficher() const
{
  cout << "Je suis le fichier " << nom << "." << extension<< endl;
  cout << "ma taille est de : " << taille << "Kb" << endl;
}

class Programme {
public:
  Programme(string, string, string);
  virtual ~Programme();
  void afficher() const;
private:
  string langage;
  string auteur;
  string descriptif;
};

Programme::Programme(string langage, string auteur, string descriptif)
```

```
  : langage(langage), auteur(auteur), descriptif(descriptif)
{
  cout << "creation d'un programme" << endl;
}

Programme::~Programme()
{
 cout << "destruction d'un programme" << endl;
}

void Programme::afficher() const
{
  cout << "Je suis ecrit en : " << langage << endl;
  cout << "mon auteur est   : " << auteur<<  endl;
  cout << "Je fait du : " << descriptif << endl;
}

class FichierCPP: public Programme, public Document {
public:
  FichierCPP(int, string, string, string, string, string);
  ~FichierCPP();
  void afficher () const;
};

FichierCPP::FichierCPP(int taille, string nom, string extension,
                       string langage, string auteur,
                       string descriptif)
  : Programme(langage, auteur, descriptif),
    Document(taille, nom, extension)
{
  cout << "construction d'un fichier C++" << endl;
}

FichierCPP::~FichierCPP()
{
  cout << "destruction d'un fichier C++" << endl;
}

void FichierCPP::afficher() const
{

  Document::afficher();
  Programme::afficher();

}

int main ()
{
  FichierCPP f(1600, "numeric", "cc", "C++", "C.Hacker",
               "calcul numerique");
  f.afficher();
  return 0;
}
```

Exercice 24 : Échecs

Cet exercice correspond à l'exercice n°73 (pages 186 et 384) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

```
#include <iostream>
#include <vector>
#include <cstdlib> // pour la fonction abs(int)
using namespace std;

class Position {
public:
  Position(unsigned char li, unsigned char co) : x(li), y(co)
     { check(x); check(y); }
  void affiche() const;
  unsigned char ligne()   const { return x; }
  unsigned char colonne() const { return y; }
protected:
  unsigned char x;
  unsigned char y;
private:
  void check(unsigned char&);
};

void Position::check(unsigned char& pos) {
  if (pos < 1) pos = 1;
  else if (pos > 8) pos = 8;
}

void Position::affiche() const {
  cout << char('a' + x-1) << int(y);
}

// ---------------------------------------------------------------------
enum Couleur { Blanc, Noir };

// ---------------------------------------------------------------------
class Piece {
public:
  bool deplace(const Position&);
  virtual void affiche() const { affiche_type(); p.affiche(); }
  Piece(Position p, Couleur c) : p(p), c(c) {}
protected:
  Position p;
  Couleur c;
  virtual void affiche_type() const = 0;
  virtual bool deplacement_possible(const Position&) const = 0;
};

bool Piece::deplace(const Position& new_p) {
  if (deplacement_possible(new_p)) {
    affiche_type();
    cout << " déplacé de "; p.affiche(); cout << " à ";
    new_p.affiche(); cout << endl;
    p = new_p;
    return true;
  } else {
    return false;
  }
}
```

```
// ----------------------------------------------------------------------
class Cheval : public Piece {
public:
  using Piece::Piece; // héritage du constructeur (on n'a pas de nouvel attribut)
private:
  void affiche_type() const { cout << 'C'; }
  bool deplacement_possible(const Position& new_p) const {
    return (abs(p.ligne() - new_p.ligne())
            * abs(p.colonne() - new_p.colonne()) == 2);
  }
};

// ----------------------------------------------------------------------
class Tour : public Piece {
public:
  using Piece::Piece;
private:
  void affiche_type() const { cout << 'T'; }
  bool deplacement_possible(const Position& new_p) const {
    return (((p.ligne() == new_p.ligne()) and (p.colonne() != new_p.colonne())) or
            ((p.ligne() != new_p.ligne()) and (p.colonne() == new_p.colonne())) );
  }
};

// ----------------------------------------------------------------------
class Jeu
{
public:
  Jeu();
  void affiche() const;
  Piece* piece(size_t nb) const;
  virtual ~Jeu();
private:
  vector<Piece*> contenu;
};

Jeu::Jeu() {
  contenu.push_back(new Cheval(Position(5,6), Noir  ));
  contenu.push_back(new Cheval(Position(1,2), Blanc ));
  contenu.push_back(new Tour  (Position(7,6), Noir  ));
  contenu.push_back(new Tour  (Position(3,2), Blanc ));
}

Piece* Jeu::piece(size_t nb) const {
  if (nb < contenu.size()) {
    return contenu[nb];
  }
  return nullptr;
}

Jeu::~Jeu() {
  for (auto& piece : contenu) { delete piece; }
}

void Jeu::affiche() const {
  for (auto piece : contenu) {
    piece->affiche();
    cout << endl;
  }
}

// ======================================================================
int main()
```

```
{
  Jeu mon_jeu;

  mon_jeu.affiche();
  mon_jeu.piece(1)->deplace(Position(6, 4));
  mon_jeu.piece(0)->deplace(Position(6, 4));
  mon_jeu.piece(3)->deplace(Position(7, 1));
  mon_jeu.piece(2)->deplace(Position(7, 1));
  cout << "----------" << endl;
  mon_jeu.affiche();

  return 0;
}
```

Exercice 25 : bibliothèque

Cet exercice correspond à l'exercice n°74 (pages 188 et 386) de l'ouvrage *C++ par la pratique* (3e édition, PPUR).

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

typedef string Date;

// -----------------------------------------------------------------------
class Ouvrage {
public :
  Ouvrage() : emprunte(false) {}
  virtual ~Ouvrage() {}
  void emprunt(string nom, Date d);
  void rendu();
  bool dispo() const { return not emprunte; }
  virtual void affiche() const;
protected :
  bool emprunte;
  string nom_emprunt;
  Date date_emprunt;
  virtual void info() const = 0;
};

void Ouvrage::affiche() const
{
  info();
  if (emprunte) {
    cout << "Emprunté le " << date_emprunt
         << " par " << nom_emprunt << endl;
  } else {
    cout << "Disponible." << endl;
  }
}

void Ouvrage::rendu() {
  nom_emprunt  = "";
  date_emprunt = Date(); // appel du constructeur par defaut
  emprunte = false;
}

void Ouvrage::emprunt(string nom, Date d) {
  nom_emprunt  = nom;
  date_emprunt = d;
  emprunte = true;
}

// -----------------------------------------------------------------------
class Periodique : public virtual Ouvrage
{
public:
  Periodique(string t, Date d, string e)
    : titre(t), date(d), editeur(e) {}
  virtual ~Periodique() {}
protected:
  string titre;
  Date date;
```

```
  string editeur;
  void info() const;
};

void Periodique::info() const
{
  cout << "Journal :" << endl;
  cout << '\t' << titre << " de " << date << endl;
  cout << "\tpublié par " << editeur << endl;
}

// ---------------------------------------------------------------------
class Livre : public virtual Ouvrage
{
public:
  Livre(string t, string auth, Date d, string e)
    : titre(t), auteurs(auth), date(d), editeur(e) {}
  virtual ~Livre() {}
protected:
  string titre;
  string auteurs;
  Date date;
  string editeur;
  void info() const;
};

void Livre::info() const
{
  cout << "Livre :" << endl;
  cout << '\t' << titre << endl;
  cout << "\tpar " << auteurs << endl;
  cout << "\tpublié en " << date << " par " << editeur << endl;
}

// ---------------------------------------------------------------------
class Video : public virtual Ouvrage
{
public:
  Video(string t, string auth, Date d, string e, unsigned int min)
    : titre(t), auteurs(auth), date(d), editeur(e), minutes(min) {}
  virtual ~Video() {}
protected:
  string titre;
  string auteurs;
  Date date;
  string editeur;
  unsigned int minutes;
  void info() const;
};

void Video::info() const
{
  cout << "Vidéo :" << endl;
  cout << '\t' << titre << endl;
  cout << "\tpar " << auteurs << endl;
  cout << "\tpubliée en " << date << " par " << editeur << endl;
  cout << "\tdurée : " << minutes << " minutes" << endl;
}

// ---------------------------------------------------------------------
class OuvrageExterne : public virtual Ouvrage
{
public:
```

```
  OuvrageExterne(string nom, Date d, string where);
  virtual ~OuvrageExterne() {}
  void affiche() const;
protected:
  string origine;
};

OuvrageExterne::OuvrageExterne(string nom, Date d, string where)
  : origine(where)
{
  emprunt(nom, d);
}

void OuvrageExterne::affiche() const
{
  Ouvrage::affiche();
  cout << "provenant de " << origine << endl;
}

// ---------------------------------------------------------------------
class LivreExterne : public Livre, public OuvrageExterne
{
public:
  LivreExterne(string t, string auth, Date dp, string e, Date de,
               string where, string nom)
    : Livre(t, auth, dp, e), OuvrageExterne(nom, de, where)
  {}
  virtual ~LivreExterne() {}
};

// ---------------------------------------------------------------------
class Bibliotheque
{
public:
  virtual ~Bibliotheque() {}
  size_t ajoute(Ouvrage*);
  Ouvrage* ouvrage(size_t nb) const;
  void supprime(size_t);
  void affiche() const;
  void vider();
private:
  vector<Ouvrage*> contenu;
};

Ouvrage* Bibliotheque::ouvrage(size_t nb) const {
  if (nb < contenu.size()) {
    return contenu[nb];
  }
  return nullptr;
}

void Bibliotheque::vider()
{
  for (auto& ouvrage : contenu) delete ouvrage;
  contenu.clear();
}

void Bibliotheque::affiche() const
{
  for (size_t i(0); i < contenu.size(); ++i) {
    cout << i << "- ";
    contenu[i]->affiche();
  }
```

```
}

size_t Bibliotheque::ajoute(Ouvrage* o)
{
  contenu.push_back(o);
  return (contenu.size()-1);
}

void Bibliotheque::supprime(size_t del)
  /* Note : si on connait bien la bibliothèque standard,
     on peut aussi le faire avec les iterateurs et la methode erase().  */
{
  if (del < contenu.size()) {
    delete contenu[del];
    const size_t end(contenu.size()-2);
    for (size_t i(del); i <= end; ++i)
        contenu[i] = contenu[i+1];
    contenu.pop_back();
  }
}

// ========================================================================
int main()
{
  Bibliotheque mabib;

  mabib.ajoute(new Livre("Programmation orientée objets en C++",
                         "Micheloud et Rieder", "2003", "PPUR"));

  mabib.ajoute(new Periodique("Computational Linguistics",
                              "12/2003", "MIT Press"));

  size_t o1, o2;
  o1 = mabib.ajoute(new Video("Nanometers and Gigabucks",
                              "G. E. Moore", "1996",
                              "University Video Communications", 61));

  o2 = mabib.ajoute(new LivreExterne("Elements of Information Theory",
                                     "Cover and Thomas", "1991",
                                     "Wiley", "25/02/2004",
                                     "ETHZ-BC", "C. Chatnonne"));

  mabib.affiche();
  cout << "=================================================" << endl;

  mabib.ouvrage(o1)->emprunt("J.-C. Chappelier", "05/07/2015");
  mabib.supprime(o2);

  mabib.affiche();

  mabib.vider();

  return 0;
}
```