MOOC Intro POO C++

Corrigés semaine 2

Les corrigés proposés donnent à chaque fois une solution à laquelle vous pourriez aboutir au moyen des connaissances acquises jusqu'à la semaine correspondante. On rappelle que plusieurs solutions sont en général possibles.

Exercice 5 : Apéritif (niveau 1)

```
class Apero {
public:
    Apero() { cout << "L'heure de l'apero a sonne !" << endl; }
    ~Apero() { cout << "A table !" << endl; }
    void bis() const { cout << "Encore une ?" << endl; }
};
```

Exercice 6 : Un peu de douceur dans ce monde (niveau 1)

```cpp
class Fleur {
public:
  Fleur(const string& espece, const string& couleur) : couleur(couleur)
  {
    cout << espece << " fraichement cueillie" << endl;
  }

  Fleur(const Fleur& f) : couleur(f.couleur)
  {
    cout << "Fragile corolle taillee" << endl;
  }

  ~Fleur() { cout << "qu'un simple souffle..." << endl; }

  void eclore() const { cout << "veine de " << couleur << endl; }

private:
  const string couleur;
};
```

Exercice 7 : Banque (niveau 3)

Voici une solution possible pour la première partie (voir les commentaires) :

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

// ========================================================================
class Compte {
public:

  /* Un compte a au moins un identifiant (= intitule) et un taux de remuneration.
  * J'ai trouve plus naturel de fixer le solde a 0 lors de l'ouverture du compte.
  */
  Compte(string const& nom, double taux): intitule(nom), solde(0.0), taux(taux)
  {}

  // Moyen de recuperer l'intitule du compte
  const string& nom() const {
    return intitule;
  }

  /* Pour ajouter de l'argent sur le compte.
  * Dans un contexte plus large, on fournirait bien sur aussi la methode pour retirer de l'argent.
  */
  void crediter(double somme) {
    solde += somme;
  }

  /* Operations lors du bouclement du compte.
  * Ici : crediter les interets
  */
  void bouclement() {
    crediter(solde * taux);
  }

  // Affichage du compte.
  void afficher() const {
    cout << " Compte " << intitule << " : " << solde << " francs" << endl;
  }

// ----------------------------------------
private:
  const string intitule;
  double solde;
  double taux;
};

// ========================================================================
class Client {
public:

  /* Un client a au moins un nom et une adresse (= ville ici).
  * On imagine egalement que pour etre client, il a au moins un compte.
  * Le taux par defaut de ce compte est ici arbitraire. On pourrait tres
  * bien imaginer ne pas avoir de valeur par defaut ici.
  */
  Client(string const& nom, string const& adresse, double taux_negocie = 0.01)
    : nom(nom), ville(adresse)
    // disons que pour etre client il faut au moins un compte courant :
    , portefeuille(1, Compte("courant", taux_negocie))
```

```
  {}

  // Pour ouvrir un nouveau compte
  void ouvre_compte(string const& nom, double taux) {
    portefeuille.push_back(Compte(nom, taux));
  }

  // Affichage des informations du client
  void afficher() const {
    cout << "Client " << nom << " de " << ville << endl;
    for (const auto & compte : portefeuille) {
      compte.afficher();
    }
  }

  // Bouclement de tous les comptes du client
  void bouclement() {
    for (auto & compte : portefeuille) {
      compte.bouclement();
    }
  }

// Pour ajouter de l'argent sur un compte (donne par son intitule)
    void crediter(string const& intitule, double somme) {
      for (auto & compte : portefeuille) {
        if (compte.nom() == intitule) {
          compte.crediter(somme);
          return;
        }
      }
    }

// ----------------------------------------
private:
  const string nom;
  string ville;
  vector<Compte> portefeuille;
};

// ========================================================================
class Banque {
public:

  // Pour ajouter un nouveau client
  void nouveau_client(Client& quidam) {
    // en toute rigueur, il faudrait ici verifier que quidam n'est pas deja client !
    clients.push_back(&quidam);
  }

  // Pour faire le bouclement
  void bouclement() {
    for(auto & client : clients) {
      client->bouclement();
    }
  }

  // Pour afficher les comptes des clients
  void afficher() const {
    for(auto & client : clients) {
      client->afficher();
    }
  }

```

```
// ---------------------------------------
private:
  /* La banque ne possede pas vraiment ses clients, au mieux "un moyen de les
  * contacter" = pointeurs vers les clients
  */
  vector<Client*> clients;
};

// ========================================================================
int main()
{
  Banque fictive;

  Client pedro ("Pedro" , "Geneve" );
  fictive.nouveau_client(pedro);
  pedro.crediter("courant", 1000.0);
  pedro.ouvre_compte("epargne", 0.02);
  pedro.crediter("epargne", 2000.0);

  Client alexandra("Alexandra", "Lausanne");
  fictive.nouveau_client(alexandra);
  alexandra.crediter("courant", 3000.0);
  alexandra.ouvre_compte("epargne", 0.02);
  alexandra.crediter("epargne", 4000.0);

  cout << "Donnees avant le bouclement des comptes :" << endl;
  fictive.afficher();

  fictive.bouclement();

  cout << "Donnees apres le bouclement des comptes :" << endl;
  fictive.afficher();

  return 0;
}
```

et sa révision pour la seconde partie :

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

// ========================================================================
class Compte {
public:

  /* Un compte a au moins un identifiant (= intitule) et un taux de remuneration.
  * J'ai trouve plus naturel de fixer le solde a 0 lors de l'ouverture du compte.
  */
  Compte(string const& nom, double taux): intitule(nom), solde(0.0), taux(taux)
  {}

  // Moyen de recuperer l'intitule du compte
  const string& nom() const {
    return intitule;
  }

  /* Pour ajouter de l'argent sur le compte.
  * Dans un contexte plus large, on fournirait bien sur aussi la methode pour retirer de l'argent.
  */
  void crediter(double somme) {
    solde += somme;
```

```
  }

  /* Operations lors du bouclement du compte.
  * Ici : crediter les interets
  */
  void bouclement() {
    crediter(solde * taux);
  }

  // Affichage du compte.
  void afficher() const {
    cout << " Compte " << intitule << " : " << solde << " francs" << endl;
  }

// ---------------------------------------
private:
  const string intitule;
  double solde;
  double taux;
};

// ========================================================================
class Client {
public:

  /* Un client a au moins un nom et une adresse (= ville ici).
  * On imagine egalement que pour etre client, il a au moins un compte.
  * Le taux par defaut de ce compte est ici arbitraire. On pourrait tres
  * bien imaginer ne pas avoir de valeur par defaut ici.
  */
  Client(string const& nom, string const& adresse, char genre, double taux_negocie = 0.01)
    : nom(nom), ville(adresse), genre(genre)
    // disons que pour etre client il faut au moins un compte courant :
    , portefeuille(1, Compte("courant", taux_negocie))
  {}

  // Pour ouvrir un nouveau compte
  void ouvre_compte(string const& nom, double taux) {
    portefeuille.push_back(Compte(nom, taux));
  }

  // Affichage des informations du client
  void afficher() const {
      cout << "Client" ;
      if (genre == 'F') cout << 'e';
      else if (genre == 'A') cout << "(e)";
      cout << " " << nom << " de " << ville << endl;
      for (const auto & compte : portefeuille) {
         compte.afficher();
      }
  }

  // Bouclement de tous les comptes du client
  void bouclement() {
    for (auto & compte : portefeuille) {
      compte.bouclement();
    }
  }

// Pour ajouter de l'argent sur un compte (donne par son intitule)
    void crediter(string const& intitule, double somme) {
      for (auto & compte : portefeuille) {
        if (compte.nom() == intitule) {
```

```
89          compte.crediter(somme);
90          return;
91        }
92      }
93    }
94
95  // ----------------------------------------
96  private:
97    const string nom;
98    string ville;
99    vector<Compte> portefeuille;
100   char genre;
101 };
102
103 // ========================================================================
104 class Banque {
105 public:
106
107   // Pour ajouter un nouveau client
108   void nouveau_client(Client& quidam) {
109     // En toute rigueur, il faudrait ici verifier que quidam n'est pas deja client !
110     // Par ailleurs, cette logique n'est pas tout a fait safe, puisqu'on stocke un pointeur vers
111     // un objet controle par l'appelant (qui pourrait donc "disparaitre" sans prevenir)
112     clients.push_back(&quidam);
113   }
114
115   // Pour faire le bouclement
116   void bouclement() {
117     for(auto & client : clients) {
118       client->bouclement();
119     }
120   }
121
122   // Pour afficher les comptes des clients
123   void afficher() const {
124     for(auto & client : clients) {
125       client->afficher();
126     }
127   }
128
129 // ----------------------------------------
130 private:
131   /* La banque ne possede pas vraiment ses clients, au mieux "un moyen de les
132   * contacter" = pointeurs vers les clients
133   */
134   vector<Client*> clients;
135 };
136
137 // ========================================================================
138 int main()
139 {
140   Banque fictive;
141
142   Client pedro ("Pedro" , "Geneve", 'M' );
143   fictive.nouveau_client(pedro);
144   pedro.crediter("courant", 1000.0);
145   pedro.ouvre_compte("epargne", 0.02);
146   pedro.crediter("epargne", 2000.0);
147
148   Client alexandra("Alexandra", "Lausanne", 'F');
149   fictive.nouveau_client(alexandra);
150   alexandra.crediter("courant", 3000.0);
151   alexandra.ouvre_compte("epargne", 0.02);
```

```
152   alexandra.crediter("epargne", 4000.0);
153
154   cout << "Donnees avant le bouclement des comptes :" << endl;
155   fictive.afficher();
156
157   fictive.bouclement();
158
159   cout << "Donnees apres le bouclement des comptes :" << endl;
160   fictive.afficher();
161
162   return 0;
163 }
```

Exercice 8 : Supermarché (niveau 2)

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

// ========================================================================
class Article {
public:
  Article(string const& nom, double prix, bool action = false)
  : nom_(nom), prix_(prix), action_(action)
  {}

  double prix() const { return prix_; }
  string nom() const { return nom_; }
  bool est_en_action() const { return action_; }

private:
  const string nom_ ;
  double prix_ ;
  bool action_ ;
};

// ========================================================================
class Achat {
public:
  Achat(Article const& article, unsigned int quantite = 1)
  : article_(article), quantite_(quantite)
  {}

  double prix() const {
    double prix_( quantite_ * article_.prix() );
    if (article_.est_en_action()) {
      prix_ *= 0.5;
    }
    return prix_;
  }

  void afficher() const {
    cout << article_.nom() << " : "
      << article_.prix() << " x " << quantite_
      << " = " << prix() << " F";
    if (article_.est_en_action()) {
      cout << " (en action)";
    }
    cout << endl;
  }

private:
  const Article article_;
  const unsigned int quantite_;
};

// ========================================================================
class Caddie {
public:
  void remplir(Article const& article, unsigned int quantite = 1) {
    achats.push_back(Achat(article, quantite));
  }

  double total() const {
```

```
    double somme(0.0);
    for (auto const& achat : achats) {
      achat.afficher();
      somme += achat.prix();
    }
    return somme;
  }

private:
  vector<Achat> achats;
};

// ========================================================================
class Caisse {
public:
  Caisse() : total(0.0) {}

  void afficher() const {
    cout << total << " F";
  }

  void scanner(Caddie const& caddie) {
    double montant(caddie.total());
    total += montant;

    cout << "-----" << endl;
    cout << "Total a payer : " << montant << " F." << endl;
  }

private:
  double total;
};

// ========================================================================
int main()
{
  // Les articles vendus dans le supermarche
  Article choufleur ("Chou-fleur extra" , 3.50 );
  Article roman ("Les malheurs de Sophie", 16.50, true );
  Article camembert ("Cremeux 100%MG" , 5.80 );
  Article cdrom ("C++ en trois jours" , 48.50 );
  Article boisson ("Petit-lait" , 2.50, true);
  Article petitspois("Pois surgeles" , 4.35 );
  Article poisson ("Sardines" , 6.50 );
  Article biscuits ("Cookies de grand-mere" , 3.20 );
  Article poires ("Poires Williams" , 4.80 );
  Article cafe ("100% Arabica" , 6.90, true);
  Article pain ("Pain d'epautre" , 6.90 );

  // Les caddies du supermarche, disons 3 ici
  vector<Caddie> caddies(3);

  // Les caisses du supermarche, disons 2
  vector<Caisse> caisses(2);

  // Les clients font leurs achats :
  // le second argument de la methode remplir correspond a une quantite

  // remplissage du 1er caddie
  caddies[0].remplir(choufleur, 2);
  caddies[0].remplir(cdrom );
  caddies[0].remplir(biscuits , 4);
  caddies[0].remplir(boisson , 6);
```

```
124   caddies[0].remplir(poisson , 2);
125
126   // remplissage du 2eme caddie
127   caddies[1].remplir(roman );
128   caddies[1].remplir(camembert );
129   caddies[1].remplir(petitspois, 2);
130   caddies[1].remplir(poires , 2);
131
132   // remplissage du 3eme caddie
133   caddies[2].remplir(cafe , 2);
134   caddies[2].remplir(pain );
135   caddies[2].remplir(camembert, 2);
136
137   // Les clients passent a la caisse :
138   caisses[0].scanner(caddies[0]);
139   cout << "========================================" << endl;
140   caisses[0].scanner(caddies[1]);
141   cout << "========================================" << endl;
142   caisses[1].scanner(caddies[2]);
143   cout << "========================================" << endl;
144
145   // Affichage du resultat des caisses
146   cout << "Resultats du jour :" << endl;
147   for (size_t i(0); i < caisses.size(); ++i) {
148   cout << "Caisse " << i+1 << " : " ;
149   caisses[i].afficher();
150   cout << endl;
151 }
152 return 0;
153 }
```