MOOC Intro. POO C++
Corrigés semaine 2

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 5 : apéro

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

Exercice 6 : un peu de douceur…

```
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 : banques

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

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

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

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

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

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

  /* Opérations lors du bouclement du compte.
   * Ici : créditer les intérêts
   */
  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 également que pour être client, il a au moins un compte.
   * Le taux par défaut de ce compte est ici arbitraire. On pourrait très
   * bien imaginer ne pas avoir de valeur par défaut ici.
```

```
   */
  Client(string const& nom, string const& adresse, double taux_negocie = 0.01)
    : nom(nom), ville(adresse)
      // disons que pour être 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 (donné par son intitulé)
  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 vérifier que quidam n'est pas déjà 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 n'A 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"    , "Genève"  );
  fictive.nouveau_client(pedro);
  pedro.crediter("courant", 1000.0);
  pedro.ouvre_compte("épargne", 0.02);
  pedro.crediter("épargne", 2000.0);

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

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

  fictive.bouclement();

  cout << "Données 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 (= intitulé) et un taux
   * de rémunération.
   * J'ai trouvé plus naturel de fixer le solde à 0 lors de l'ouverture
   * du comptes.
   */
  Compte(string const& nom, double taux)
    : intitule(nom), solde(0.0), taux(taux)
  {}

  // Moyen de récupérer l'intitulé du compte
```

```
  const string& nom() const {
    return  intitule;
  }

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

  /* Opérations lors du bouclement du compte.
   * Ici : créditer les intérêts
   */
  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 également que pour être client, il a au moins un compte.
   * Le taux par défaut de ce compte est ici arbitraire. On pourrait très
   * bien imaginer ne pas avoir de valeur par défaut ici.
   */
  Client(string const& nom, string const& adresse, bool homme = true,
         double taux_negocie = 0.01)
    : nom(nom), ville(adresse), homme(homme)
      // disons que pour être 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 (not homme) 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 (donné par son intitulé)
  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;
  const bool homme;
  vector<Compte> portefeuille;
};

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

  // Pour ajouter un nouveau client
  void nouveau_client(Client& quidam) {
    // en toute rigueur, il faudrait ici vérifier que quidam n'est pas déjà 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 n'A 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"    , "Genève"  );
```

```
  fictive.nouveau_client(pedro);
  pedro.crediter("courant", 1000.0);
  pedro.ouvre_compte("épargne", 0.02);
  pedro.crediter("épargne", 2000.0);

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

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

  fictive.bouclement();

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

  return 0;
}
```

Exercice 8 : supermarché

```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 à payer : " << montant << " F." << endl;
  }

private:
  double total;
};

// ============================================================================
int main()
{
  // Les articles vendus dans le supermarché
  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 supermarché, disons 3 ici
  vector<Caddie> caddies(3);

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

  // Les clients font leurs achats :
  // le second argument de la méthode remplir correspond à une quantité

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

```cpp
  // remplissage du 2eme caddie
  caddies[1].remplir(roman        );
  caddies[1].remplir(camembert    );
  caddies[1].remplir(petitspois, 2);
  caddies[1].remplir(poires    , 2);

  // remplissage du 3eme caddie
  caddies[2].remplir(cafe     , 2);
  caddies[2].remplir(pain         );
  caddies[2].remplir(camembert, 2);

  // Les clients passent à la caisse :
  caisses[0].scanner(caddies[0]);
  cout << "========================================" << endl;
  caisses[0].scanner(caddies[1]);
  cout << "========================================" << endl;
  caisses[1].scanner(caddies[2]);
  cout << "========================================" << endl;

  // Affichage du résultat des caisses
  cout << "Résultats du jour :" << endl;
  for (size_t i(0); i < caisses.size(); ++i) {
    cout << "Caisse " << i+1 << " : " ;
    caisses[i].afficher();
    cout << endl;
  }

  return 0;
}
```
