MOOC Intro POO C++

Corrigés semaine 4

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 9 : Véhicules (niveau 1)

> Cet exercice correspond à l'exercice n°55 (pages 139 et 315) de l'ouvrage *C++ par la pratique (3e édition, PPUR).*

Pour la classe `Vehicule` :

```
class Vehicule {
protected:
    string marque ;
    unsigned int date_achat ;
    double prix_achat ;
    double prix_courant ;
};
```

Ces attributs sont « `protected` » car on ne souhaite pas pouvoir y accéder en dehors de la classe, mais on imagine tout de même pouvoir y accéder depuis « les héritiers ».

Pour le constructeur, voici une solution possible :

```
class Vehicule {
public:
    Vehicule(string marque, unsigned int date, double prix)
    : marque(marque), date_achat(date), prix_achat(prix),
      prix_courant(prix)
    {}
    //...suite comme avant
```

Pour la méthode `affiche` :

```
class Vehicule {
public:
    // ...
    void affiche(ostream& affichage) const;
    // ...
};
void Vehicule::affiche(ostream& affichage) const {
    affichage << "marque : " << marque
              << ", date d'achat : " << date_achat
              << ", prix d'achat : " << prix_achat
              << ", prix actuel : " << prix_courant
              << endl;
}
```

Cette méthode est `const` car elle ne modifie pas l'état de l'objet.

NOTE : Il serait préférable ici de surcharger l'opérateur externe `ostream& operator<<(ostream&, const Vehicule&)` mais ce n'est pas le sujet de cet exercice.

Dans la direction opposée, on pourrait aussi déclarer `affiche` sans lui passer de paramètre et la faire directement opérer sur `cout`.

Pour la classe `Voiture`, elle doit hériter de la classe `Vehicule` ; on ajoute ensuite ses champs spécifiques :

```
class Voiture : public Vehicule {
protected:
    double cylindree ;
    unsigned int nb_portes ;
    double puissance ;
    unsigned int kilometrage ;
};
```

On procède de même pour la classe `Avion` :

```
enum Type_Avion { HELICES, REACTION };

class Avion : public Vehicule {
protected:
    Type_Avion moteur ;
    unsigned int heures_vol ;
};
```

Pour les constructeurs et méthodes d'affichage :

```
class Voiture : public Vehicule {
public:
    Voiture(string marque, unsigned int date, double prix,
         double cylindree, unsigned int portes, double cv, unsigned int km);
    void affiche(ostream&) const;
protected:
    double cylindree ;
    unsigned int nb_portes ;
    double puissance ;
    unsigned int kilometrage ;
};
```

Ces deux méthodes doivent bien entendu être publiques puisqu'elles sont précisément faites pour être utilisées hors de la classe. Voici un exemple possible pour leur définition :

```
Voiture::Voiture(string marque, unsigned int date, double prix,
       double cylindree, unsigned int portes, double cv,
       unsigned int km)
    : Vehicule(marque, date, prix), cylindree(cylindree),
      nb_portes(portes), puissance(cv), kilometrage(km)
{}

void Voiture::affiche(ostream& affichage) const {
    affichage << " ---- Voiture ----" << endl;
    Vehicule::affiche(affichage);
    affichage << cylindree << " litres, "
        << nb_portes << " portes, "
        << puissance << " CV, "
        << kilometrage << " km." << endl;
}
```

Notons que pour le constructeur de `Voiture`, on fait appel au constructeur de `Vehicule`, et que `affiche` appelle la méthode du même nom de la classe `Vehicule` en la « démasquant » à l'aide de l'opérateur « : : ».

Les méthodes de la classe `Avion` s'implémentent de même :

```
class Avion : public Vehicule {
public:
    Avion(string marque, unsigned int date, double prix,
```

```
        Type_Avion moteur, unsigned int heures);
    void affiche(ostream&) const;

protected:
    Type_Avion moteur ;
    unsigned int heures_vol ;
};

Avion::Avion(string marque, unsigned int date, double prix,
         Type_Avion moteur, unsigned int heures)
    : Vehicule(marque, date, prix), moteur(moteur), heures_vol(heures)
{}

void Avion::affiche(ostream& affichage) const {
    affichage << " ---- Avion a ";
    if (moteur == HELICES) affichage << "helices";
    else affichage << "reaction";
    affichage << " ----" << endl;
    Vehicule::affiche(affichage);
    affichage << heures_vol << " heures de vol." << endl;
}
```

et pour `calculePrix()` (exemple pour l'année 2015) :

```
class Vehicule {
public:
    Vehicule(string marque, unsigned int date, double prix);
    void affiche(ostream&) const;
    void calculePrix();
...
};

...
void Vehicule::calculePrix() {
    double decote((2015 - date_achat) * .01);
    prix_courant = std::max(0.0, (1.0 - decote) * prix_achat);
}
```

Le prototype est le même pour `Voiture` et `Avion` que pour `Vehicule`, par contre les définitions diffèrent :

```
void Voiture::calculePrix() {
    double decote((2015 - date_achat) * .02);
    decote += 0.05 * (kilometrage / 10000.0);
    if (marque == "Fiat" || marque == "Renault")
        decote += 0.1;
    else if (marque == "Ferrari" || marque == "Porsche")
        decote -= 0.2;
    prix_courant = std::max(0.0, (1.0 - decote) * prix_achat);
}

...

void Avion::calculePrix() {
    double decote;
    if (moteur == HELICES)
        decote = 0.1 * heures_vol / 100.0;
    else
        decote = 0.1 * heures_vol / 1000.0;

    prix_courant = std::max(0.0, (1.0 - decote) * prix_achat);
}
```