Classes, instances, objets [oo]
Exercices de cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 20 mai 2018

Table des matières

C++ - Exercices de cours (Solution)

Mots-Clés Classes, instances, objets ■
Difficulté • ○ ○ (1 h 30) ■

DEPLACEZ Le Tour de magie dans oo03 ■

1 Appréhender le cours

1.1 Classe Personne / pgpersonne

Écrivez une classe `Personne` munie d'un `nom` (chaîne de caractères) et d'un `age` (entier).

Écrivez un constructeur qui initialise les attributs.

Écrivez des accesseurs `getNom` du nom et `getAge` de l'âge.

Écrivez un mutateur du nom `setNom(nom)`.

Écrivez une méthode `feterAnniv` qui fait vieillir la personne d'une année.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Personne.hpp] @[Personne.cpp]

```
#ifndef PERSONNE_CLASS
#define PERSONNE_CLASS
#include <string>
using namespace std;

/**
  Classe Personne
*/
class Personne {
public:
  Personne(const string& nom, int age);
  string getNom() const;
  int getAge() const;
  void setNom(const string& nom);
  void feterAnniv();
private:
 string m_nom;
 int m_age;
};
#include "Personne.cpp"
#endif
```

```
/**
  Constructeur normal
  @param[in] nom - nom de la personne
  @param[in] age - age de la personne
*/
Personne::Personne(const string& nom, int age)
: m_nom(nom), m_age(age)
{}

/**
```

```
  Accesseur du nom
  @return nom de la personne
*/
string Personne::getNom() const
{
  return m_nom;
}

/**
  Accesseur de l'age
  @return l'âge de la personne
*/
int Personne::getAge() const
{
  return m_age;
}

/**
  Fixe le nom
  @param[in] nom - nouveau nom
*/
void Personne::setNom(const string& nom)
{
  m_nom = nom;
}

/**
  Fait vieillir d'un an
*/
void Personne::feterAnniv()
{
  ++m_age;
}
```

Écrivez un programme qui instancie une `Personne` de nom `"toto"` qui a 10 ans.

Affichez son nom et son âge.

Faites vieillir la personne jusqu'à 21 ans (exclus) en affichant un point `(.)` à chacun de ses anniversaires puis à 21 ans, elle se marie.

Modifiez son nom en suffisant `-mariage` (à son nom), puis affichez le nouveau nom.

Faites vieillir la personne de 5 ans en affichant un point `(.)` à chacun de ses anniversaires.

Finalement affichez son nom et son âge.

Testez. Résultat d'exécution :

```
toto a 10 ans
quelques annees plus tard
..........
A 21 ans il se marie
Nouveau nom: toto-mariage
cinq annees plus tard
.....
toto-mariage a maintenant 26 ans
```

Validez votre programme avec la solution.

Solution C++ @[pgpersonne.cpp]

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

#include "Personne.hpp"
int main()
{
  Personne p("toto", 10);
  cout<<p.getNom()<<" a "<<p.getAge()<<" ans"<<endl;
  cout<<"quelques annees plus tard"<<endl;
  while (p.getAge() < 21)
  {
    p.feterAnniv();
    cout<<".";
  }
  cout<<endl;
  cout<<"A "<<p.getAge()<<" ans il se marie"<<endl;
  p.setNom( p.getNom()+"-mariage" );
  cout<<"Nouveau nom: "<<p.getNom()<<endl;
  cout<<"cinq annees plus tard"<<endl;
  for (int k = 1; k <= 5; ++k)
  {
    p.feterAnniv();
    cout<<".";
  }
  cout<<endl;
  cout<<p.getNom()<<" a maintenant "<<p.getAge()<<" ans"<<endl;
}
```

1.2 Stockage en mémoire / qstkage

Objectif
Le but de cet exercice est de comprendre les différentes façons de stocker en mémoire une variable de type primitif et une variable de type objet.

Programme C++
On considère le programme suivant :

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

void modifier(int& k)
{
  ++k;
}

void afficher(int k)
{
  cout<<k<<endl;
}

int main()
{
  cout<<"en A: -----"<<endl;
  int k = 1;
  afficher(k);
  modifier(k);
  afficher(k);
}
```

Expliquez le rôle des deux procédures.

Écrivez une classe `A` contenant un entier `k`.

Écrivez le constructeur.

Écrivez la méthode `modifier` équivalente à la procédure `modifier(k)`.

Écrivez la méthode `afficher` équivalente à la procédure `afficher(k)`.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Stkage.hpp] @[Stkage.cpp]

```
#ifndef STOCKAGE_CLASS
#define STOCKAGE_CLASS

class Stkage
{
  public:
    explicit Stkage(int v);
    void modifier();
    void afficher() const;

  private:
    int k;
};
#include "Stkage.cpp"
#endif
```

```
#include <iostream>
using namespace std;
/**
  Constructeur
  @param[in] v - un entier
*/
Stkage::Stkage(int v)
: k(v)
{}

/**
  Méthode de modification
*/
void Stkage::modifier()
{
  ++k;
}

/**
  Méthode d'affichage
*/
void Stkage::afficher() const
{
  cout<<k<<endl;
}
```

Réécrivez le programme en utilisant la classe `A` de sorte à obtenir le résultat d'exécution suivant :

```
en B: -----
1
2
```

Validez votre programme avec la solution.

Solution C++ @[qzStkageC1.cpp]

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

#include "Stkage.hpp"
int main()
{
  cout << "en B: -----" << endl;
  Stkage a1(1);
  a1.afficher();
  a1.modifier();
  a1.afficher();
}
```

C++

On considère maintenant les instructions suivantes :

```
A a2(2);
A a3(3);
```

Qu'affiche le code suivant ?

```
cout << "en C: -----" << endl;
a2.afficher();
a3.afficher();
```

De même, qu'affichera :

```
cout << "en D: -----" << endl;
a3 = a2;
a2.afficher();
a3.afficher();
```

Enfin qu'affiche ce code ?

```
cout << "en E: -----" << endl;
a2.modifier();
a3.modifier();
a2.afficher();
a3.afficher();
```

Validez vos réponses avec la solution.

Solution simple

(Résultat d'excution]

```
en C: -----
2
3
en D: -----
2
```

```
2
en E: -----
2
3
```

1.3 Le grille pain erroné / qzxconstr1

Objectif
Il s'agit de corriger un programme.

Téléchargez le programme suivant :

C++ @[qzxconstr1.cpp]

Celui-ci contient la définition d'une classe `GrillePain` qui sert à représenter des grille-pains sous forme d'objets dotés de deux variables d'instances : l'`annee` de fabrication et le nombre de tranches `nbt` que l'appareil peut griller. Il y a également une méthode `afficher` que l'on peut appliquer sur un objet de type `GrillePain` afin d'afficher les valeurs de ses attributs.

Compilez le programme. Vous aurez des messages d'erreurs.
Pourquoi ? Corrigez et recompilez.

Solution simple
Le programmeur a omis de spécifier le droit d'accès `public` ; par défaut il est `private` et donc on ne peut ni accéder à la méthode constructeur depuis l'extérieur de la classe, ni à la méthode `afficher`. Il faut rendre ces deux méthodes publiques en ajoutant le mot-clé `public`.

L'affichage lors de l'exécution (voir ci-dessous) montre que la variable d'instance `nbt` pour les deux grilles-pains ne possède pas la valeur envoyée au constructeur ? Pourquoi ? Corrigez le programme.

```
> GrillePain
GrillePain : annee = 1995 et nbt = 1
> GrillePain
GrillePain : annee = 1998 et nbt = 1
< GrillePain
< GrillePain
```

Solution simple
Ni les variables d'instance, ni les variables locales reçoivent des valeurs par défaut. Si la variable d'instance `nbt` n'est pas initialisée, elle aura une valeur aléatoire. Dans la méthode constructeur, il y a une situation de *shadowing* où le paramètre a le même nom que la variable d'instance. Il faut utiliser l'objet `this` pour différencier les deux et ainsi affecter la valeur du paramètre `nbt` à la variable d'instance `nbt`.

Qu'affiche la version corrigée du programme ?

Solution Résultat d'exécution

```
> GrillePain
GrillePain : annee = 1995 et nbt = 2
> GrillePain
GrillePain : annee = 1998 et nbt = 4
< GrillePain
< GrillePain
```

Validez votre programme avec la solution.

Solution C++ @[qzxconstr1SL.cpp]

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

// Activer l'option pour l'emploi de l'initialiseur
// de constructeur
#define USE_INITIALISEUR

// Représente des grilles-pains (VERSION CORRIGEE)
class GrillePain
{
  public:                                          //<- AJOUT
  // Constructeur normal
  #if !defined(USE_INITIALISEUR)
  GrillePainP1(unsigned a, unsigned nbt)
  {
    annee = a; this->nbt = nbt;                    //<- MODIF
    cout << "> GrillePain" << endl;
  }

  #else
  // NOTE Il est plus simple et plus EFFICACE de passer
  // par liste d'initialisation
  GrillePain(unsigned a, unsigned nbt)
  : annee(a), nbt(nbt)
  { cout << "> GrillePain" << endl; }
  #endif

  // Destructeur
  ~GrillePain()
  {
    cout << "< GrillePain" << endl;
  }

  // Méthode d'affichage des variables d'instance
  void afficher() const
  {
    cout << "GrillePain :"
         << " annee = " << annee
         << " et nbt = " << nbt
         << endl;
  }
```

```
  private:
    unsigned annee;
    unsigned nbt;
};

int main(int, char*[])
{
  {
    GrillePain g1(1995, 2);
    g1.afficher();

    GrillePain g2(1998, 4);
    g2.afficher();
  }
}
```

2 Appliquer le cours

2.1 Apéritif / pgaperitif

Objectif
Cet exercice est relatif aux constructeurs et destructeur.
Soit le programme suivant : @[pgaperitif.cpp]

```
#include <iostream>
using namespace std;
#include "Aperitif.hpp"

int main(int, char*[])
{
  {
    Aperitif bic;
    cout << "Super !\n";
    bic.bis();
    cout << "Non merci.\n";
  }
}
```

Écrivez une classe `Aperitif` de sorte qu'il affiche le texte suivant :

```
L'heure de l'apero a sonne !
Super !
Encore un ?
Non merci.
A table !
```

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Aperitif.hpp]

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

class Aperitif
{
  public:
    // Constructeur par défaut
    Aperitif()
    { cout << "L'heure de l'apero a sonne !\n"; }

    // Destructeur
    ~Aperitif()
    { cout << "A table !\n"; }

    // Méthode publique
    void bis() const
    { cout << "Encore un ? \n"; }
};
```

2.2 Un peu de douceur / pgfleur

Objectif
Cet exercice est relatif aux constructeurs et destructeur.
Soit le programme suivant : @[pgfleur.cpp]

```
#include <iostream>
using namespace std;
#include "Fleur.hpp"

int main(int, char*[])
{
  {
    Fleur f1("Violette", "bleu");
    Fleur f2(f1);
    cout << "dans un cristal ";
    f2.eclore();
    cout << "Donne un poeme un peu fleur bleue" << endl;
    cout << "ne laissant plus ";
  }
}
```

Écrivez une classe `Fleur` de sorte qu'il affiche le texte suivant (d'après un poème arabe du 5è siècle de l'HÉGIRE) :

```
Violette fraichement cueillie
Fragile corolle taillee
dans un cristal veine de bleu
Donne un poeme un peu fleur bleue
ne laissant plus qu'un simple soufle...
qu'un simple soufle...
```

La solution n'est pas forcément unique.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Fleur.hpp]

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

class Fleur
{
  public:
    // Constructeur normal
    Fleur(const string& espece, const string& couleur)
    : m_couleur(couleur)
    { cout << espece << " fraichement cueillie\n"; }

    // Constructeur de recopie
    Fleur(const Fleur& obj)
    : m_couleur(obj.m_couleur)
    { cout << "Fragile corolle taillee\n"; }
```

```
    // Destructeur
    ~Fleur()
    { cout << "qu'un simple soufle...\n"; }

    // Méthode d'éclosion
    void eclore() const
    { cout << "veine de " << m_couleur << "\n"; }

  private:
    const string m_couleur;
};
```

3 Approfondir le cours

3.1 Classe Consom / pgconsom

Objectif
Cet exercice traite des structeurs (constructeur, constructeur de recopie, destructeur). Il est relatif à la consommation en combustible d'un véhicule. Celle-ci est caractérisée par :

- Le nombre de litres/100 km (réel) consommé en trajet urbain.
- Le nombre de litres/100 km (réel) consommé en trajet autoroutier.
- La capacité (réel) de son réservoir (« autonomie »).

Écrivez une classe `Consom` représentant des consommations en carburant. Munissez-la des attributs spécifiés.

Rappel de cours
Les attributs de votre classe doivent être privés.

Écrivez un constructeur qui **initialise** les attributs et **affiche** le message « Vroum ».

Écrivez un constructeur de recopie qui affiche le message « re-Vroum ».

Écrivez le destructeur qui affiche le message « Arret moteur ».

Écrivez une méthode `eval(px,vkm,akm)` qui, pour le prix (par litre) `px` (réel) d'un combustible, un nombre de kilomètres effectuées en ville `vkm` (réel) et un nombre de kilomètres effectués sur autoroute `akm` (réel), calcule et renvoie l'évaluation du prix du trajet. La formule est la suivante :

$$(vkm * consomUrbaine/100 + akm * consomAutoroute/100) * px$$

Écrivez une méthode `afficher` qui affiche les caractéristiques de la consommation (où `[x]` désigne le contenu de `x`) :

```
consommation ville: [consomUrbaine] litres/100km
consommation autoroute: [consomAutoroute] litres/100km
autonomie: [autonomie] litres
```

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[Consom.hpp] @[Consom.cpp]

```
#ifndef CONSOM_CLASS
#define CONSOM_CLASS

// Consommateur (petrole)
class Consom
{
  public:
    Consom(double u, double a, double c);
    ~Consom();
    Consom(const Consom& arg);
    double eval(double px, double vkm, double akm) const;
    void afficher() const;

  private:
    double m_ub; // consommation urbaine (/100km)
    double m_ar; // consommation sur autoroute (/100)
    double m_nomie; // autonomie du réservoir
};
#include "Consom.cpp"
#endif
```

```
#include <iostream>
using namespace std;
/**
  Constructeur normal
  @param[in] u - consommation urbaine (/100km)
  @param[in] a - consommation sur autoroute (/100)
  @param[in] c - autonomie du reservoir
*/
Consom::Consom(double u, double a, double c)
: m_ub(u), m_ar(a), m_nomie(c)
{
  cout<<"Vroum"<<endl;
}

/**
  Constructeur de recopie
  @param[in] arg - un Consom
*/
Consom::Consom(const Consom& arg)
: m_ub(arg.m_ub), m_ar(arg.m_ar), m_nomie(arg.m_nomie)
{
  cout<<"re-Vroum"<<endl;
}

/**
  Destructeur
*/
Consom::~Consom()
{
  cout<<"Arret moteur"<<endl;
}

/**
  Méthode d'evaluation du prix d'un trajet
  @param[in] px - prix d'un litre d'essence
  @param[in] vkm - distance parcourue en ville
```

```
  @param[in] akm - distance parcourue sur autoroute
  @return évaluation du prix d'un trajet
*/
double Consom::eval(double px, double vkm, double akm) const
{
  return (vkm * m_ub / 100 + akm * m_ar / 100) * px;
}

/**
  Methode d'affichage
*/
void Consom::afficher() const
{
  cout<<"  consommation ville: "<<m_ub<<" litres/100km"<<endl;
  cout<<"  consommation autoroute: "<<m_ar<<" litres/100km"<<endl;
  cout<<"  autonomie: "<<m_nomie<<" litres"<<endl;
}
```

Écrivez un programme qui définit la constante `PRIX_ESSENCE` (prix de l'essence) de valeur 1.5.

Instanciez une voiture `v1` de paramètres de consommation `(10.5,7.5,50.0)`.

Instanciez une voiture `v2` par recopie de `v1`.

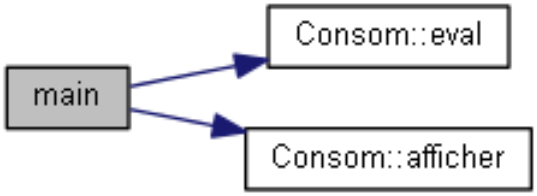
Évaluez et affichez le prix du trajet de `v1` pour 20 km en ville et 100 km sur autoroute.

Affichez les caractéristiques de `v2`.

Testez. Résultat d'exécution :

```
Vroum
re-Vroum
Prix trajet v1 (en euros):
  20km/ville et 100km/autoroute: 14.4
Voiture v2:
  consommation ville: 10.5 litres/100km
  consommation autoroute: 7.5 litres/100km
  autonomie: 50 litres
Arret moteur
Arret moteur
```

Validez votre programme avec la solution.



Solution C++ @[pgconsom.cpp]

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

#include "Consom.hpp"
int main(int, char*[])
{
  const double PRIX_ESSENCE = 1.5;
  {
    Consom v1(10.5, 7.5, 50.0);
    Consom v2(v1);
    cout << "Prix trajet v1 (en euros):" << endl;
    cout << "  20km/ville et 100km/autoroute: "
         << v1.eval(PRIX_ESSENCE, 20., 100.)
         << endl;
    cout << "Voiture v2:" << endl;
    v2.afficher();
  }
}
```

3.2 Petit tour de magie / pgmagicien

Objectif

Cet exercice simule un tour de magie élémentaire suivant :

> Un magicien demande à un spectateur d'écrire sur un papier son âge et la somme qu'il a en poche (moins de 100 €). Il demande ensuite de montrer le papier à son assistant, qui doit le lire (sans rien dire), puis effectuer secrètement le calcul suivant : multiplier l'âge par 2, ajouter 5, multiplier le résultat par 50, ajouter la somme en poche, et soustraire le nombre de jours que contient une année, puis finalement donner le résultat à haute voix. En ajoutant mentalement 115 au chiffre reçu, le magicien trouve l'âge et la somme en poche (qui étaient restés secrets).

Modélisez ce tour de magie, en définissant les classes (simples) `Magicien`, `Assistant` et `Spectateur`. Il pourrait également être utile de disposer d'une classe `Papier`. Pour chaque méthode, effectuez un affichage à l'écran de l'opération en cours et de l'acteur qui la réalise.

Aide simple

Il existe de nombreuses variantes. Commencez par un modèle très simple et faites le évoluer pour vous rapprochez de la situation décrite.

Solution simple

La méthode `arriver` du spectateur correspond en fait à son initialisation.

Validez vos classes et vos méthodes avec la solution.

Solution C++ @[Magicien.hpp]

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

// Représente un bout de papier
class Papier
{
  public:
    // Constructeur par défaut
    Papier()
    : m_age(0), m_argent(0)
    {}

    // Méthodes accesseurs
    unsigned getA() const
    { return m_age; }

    unsigned getS() const
    { return m_argent; }
```

```
    // Méthodes modifieurs
    void set(unsigned a, unsigned s)
    { m_age = a; m_argent = s; }

  private:
    unsigned m_age; // age écrit
    unsigned m_argent; // somme écrite
};

// Représente un spectateur
// NOTE Dans cette version, on fait l'hypothèse que c'est le
// spectateur qui a un papier. Dans d'autres modélisations, ce
// papier pourrait aussi appartenir au magicien, à l'assistant
// ou même "être dans la salle" (i.e. variable du main).
class Spectateur
{
  public:
    // Constructeur par défaut
    Spectateur()
    : m_age(0), m_argent(0)
    {}

    // Lorsqu'il entre dans la salle (avant il n'existe pas)!
    void arriver()
    {
      cout << "[Spectateur] (j'entre en scene)\n";

      cout << "Quel age ai-je? ";
      cin >> m_age;
      cout << "Combien d'argent ai-je en poche? ";
      cin >> m_argent;

      cin.ignore(100, '\n');
      cout << "[Spectateur] (je suis la)\n";
    }

    // Ecrit sur le papier
    void ecrire()
    {
      cout << "[Spectateur] (j'ecris le papier)\n";
      m_paquet.set(m_age, m_argent);
    }

    // Montre le papier
    const Papier& montrer() const
    {
      cout << "[Spectateur] (je montre le papier)\n";
      return m_paquet;
    }

  private:
    unsigned m_age; // age du spectateur
    unsigned m_argent; // somme en poche
    Papier m_paquet; // paquet de cigarettes
};

// Représente un assistant
```

```
class Assistant
{
  public:
    // Constructeur par défaut:
    Assistant()
    : m_papier(0), m_cerveau(0)
    {}

    // Lit un papier
    void lire(const Papier& p)
    {
      cout << "[Assistant] (je lis le papier)\n";
      m_papier = &p;
    }

    // Effectue les calculs
    void calculer()
    {
      cout << "[Assistant] (je calcule mentalement)\n";
      m_cerveau = m_papier->getA() * 2;
      m_cerveau += 5;
      m_cerveau *= 50;
      m_cerveau += m_papier->getS();
      m_cerveau -= 365;
    }

    // Annonce le résultat
    unsigned annoncer() const
    {
      cout << "[Assistant] J'annonce : " << m_cerveau << " !\n";
      return m_cerveau;
    }

  private:
    const Papier* m_papier; // un accès au papier lu
    unsigned m_cerveau; // son cerveau lui sert à retenir...
                        // ... le résultat des calculs
};

// Représente un magicien
class Magicien
{
  public:
    // Constructeur par défaut
    Magicien()
    : m_age(0), m_argent(0)
    {}

    // Pour faire son tour, le magicien a besoin d'au moins
    // un spectateur et d'un assistant
    void tourDeMagie(Assistant& asst, Spectateur& spect)
    {
      cout << "[Magicien] un petit tour de magie...\n";

      // Le magicien donne ses instructions:
      spect.ecrire();
      asst.lire(spect.montrer());
      asst.calculer();
```

```
      calculer(asst.annoncer());
      annoncer();
    }

  protected:
    // Seul le magicien sait ce qu'il doit faire dans son tour
    // Effectue les calculs
    void calculer(unsigned recu)
    {
      recu += 115;
      m_age = recu / 100;
      m_argent = recu % 100;
    }

    // Annonce le résultat
    void annoncer() const
    {
      cout << "[Magicien]\n"
           << " - humm... je vois que vous etes ages de "
           << m_age << " ans\n"
           << "   et que vous avez "
           << m_argent << " euros en poche !"
           << endl;
    }

  private:
    unsigned m_age; // age deviné
    unsigned m_argent; // argent deviné
};
#endif
```

Écrivez un programme qui simule le jeu. L'instance de `Spectateur` devra demander son âge à l'utilisateur ainsi que la somme d'argent en poche et s'assurer de la validité de la valeur (entre 1 et 99).

Testez. Exemple d'exécution :

```
[Spectateur] (j'entre en scene)
Quel age ai-je (>0)? 35
Combien d'argent ai-je en poche dans [0,99]? 112
Combien d'argent ai-je en poche dans [0,99]? 12
[Spectateur] (je suis la)
[Magicien] un petit tour de magie...
[Spectateur] (j'ecris le papier)
[Spectateur] (je montre le papier)
[Assistant] (je lis le papier)
[Assistant] (je calcule mentalement)
[Assistant] J'annonce : 3397 !
[Magicien]
 - humm... je vois que vous etes ages de 35 ans
   et que vous avez 12 euros en poche !
```

Validez votre programme avec la solution.

Solution C++ @[pgmagicien.cpp]

```cpp
#include <iostream>
using namespace std;
#include "Magicien.hpp"

int main(int, char*[])
{
  // L'histoire générale
  // Il était une fois un spectateur...
  Spectateur thorin;

  // ... qui venait voir un spectacle
  thorin.arriver();

  // ... où un magicien
  Magicien gandalf;

  // ... et son assistant
  Assistant bilbo;

  // ... lui firent un tour fantastique
  gandalf.tourDeMagie(bilbo, thorin);
}
```