Classes (suite) [cm]
Exercices de cours

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 20 mai 2018

Table des matières

C++ - Exercices de cours (Solution)

1 Appréhender le cours

1.1 Membres partagés / qzstatic1

Créez une classe `KStatic` muni d'un attribut partagé `cpt` de type `Entier`.

Que devez-vous obligatoirement faire concernant un attribut partagé.

Solution simple
Il faut l'instancier et l'initialiser.

Instanciez l'attribut partagé à zéro.

Ajoutez une méthode `m1` et une méthode partagée `m2`.
Écrivez un corps vide pour chacune.

Validez votre classe et vos méthodes avec la solution.

Solution C++ @[KStatic.cpp]

```
class KStatic
{
  public:
    void m1();
    static void m2();
  private:
    static int cpt;
};

int KStatic::cpt = 0;
void KStatic::m1()
{}
void KStatic::m2()
{}
```

Déclarez une instance `o` de type `K` puis écrivez tous les appels licites de méthodes.

Validez votre programme avec la solution.

Solution C++ @[qzstatic.cpp]

```
#include "KStatic.hpp"
int main()
{
  KStatic o;
  o.m1();
```

```
  o.m2();
  KStatic::m2();
}
```

1.2 Classe Dessin / pgdessin

1.2.1 Classe Dessin

Écrivez une classe `Dessin` contenant :

- Un attribut `n` (entier) du nombre de replicatats.
- Un attribut `motif` (chaîne de caractères) du motif à reproduire.
- Un attribut partagé public `total` (entier) du nombre d'instances.

Initialisez l'attribut partagé à zéro.

Écrivez un constructeur normal à deux paramètres qui :

- Initialise les attributs.
- Et incrémente le `total` du nombre d'instances.

Écrivez une méthode `afficher` qui affiche une ligne de `n` tirets suivi du `motif`.

Validez votre classe et vos méthodes avec la solution.

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

```
#include <iostream>
#include <string>
using namespace std;
/**
  Dessin
*/
class Dessin
{
  public:
    Dessin(int n, const string& motif);
    void afficher() const;
    static int total;
  private:
    int m_n;
    string m_motif;
};
/**
  Initialise les attributs partages
*/
int Dessin::total = 0;

/**
  Constructeur normal
  @param[in] n - nombre de replications
  @param[in] motif - le motif
*/
Dessin::Dessin(int n, const string& motif)
: m_n(n), m_motif(motif)
```

```
{
  ++total;
}

/**
  Methode d'affichage
*/
void Dessin::afficher() const
{
  cout << string(m_n, '-') << m_motif << endl;
}
```

1.2.2 Programme de test

On considère le programme suivant :

Programme C++ @[qzdessin.cpp]

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

#include "Dessin.hpp"

void afficher(int n)
{
  Dessin d(n,"*");
  d.afficher();
}

void afficherDessin(int n)
{
  for (int k = 0; k < n; ++k)
  {
    afficher(k);
  }
  for (int k = n; k >= 0; --k)
  {
    afficher(k);
  }
}

int main()
{
  cout<<"Voici un dessin"<<endl;
  afficherDessin(5);
  cout<<"Je viens d'afficher "<<Dessin::total<<" motifs"<<endl;
}
```

Sans l'exécuter, qu'affiche-t-il ?

Solution simple

Résultat d'exécution :

```
Voici un dessin
*
-*
--*
---*
----*
-----*
----*
---*
--*
-*
*
Je viens d'afficher 11 motifs
```

Dans ce programme il y en a $2n + 1$ (avec $n = 5$) soit 11 motifs.

Exécutez-le afin de confirmer vos résultats.

On supprime le qualificatif `static` de l'attribut partagé `total`.

Quelles seront les modifications à apporter au programme ? Justifiez.

Solution simple

La suppression du qualificatif induit que l'attribut `total` est de la durée de vie de l'instance : il faut un objet pour pouvoir y faire référence. D'où :

- L'initialisation des attributs partagés n'est plus valide : ligne à supprimer.
- L'expression `Dessin::total` n'est plus valide : elle nécessite un objet et l'opérateur point (`.`).

1.3 Variables statiques / qzstatic2

Téléchargez le programme suivant :

C++ @[qzstatic2.cpp]

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

class Figure
{
  public:
    Figure()
    : csurf("blanc"), ccont("jaune")
    {}

    Figure(const string& cs, const string& cc)
    : csurf(cs), ccont(cc)
    {
      if (ccont == "noir")  { ++nfgCN; }
      if (csurf == "blanc") { ++nfgSB; }
      ++nfigs;
    }

    void afficher() const
    { cout << "csurf = " << csurf << ", ccont = " << ccont << endl; }

    void setCC(const string& x)
    { ccont = x; }

    void setCS(const string& x)
    { csurf = x; }

    static unsigned nfigs;      // # de figures
    static unsigned nfgCN;      // # de figures Contour Noir
    static unsigned nfgSB;      // # de figures Surface Blanc

  private:
    string csurf;               // couleur de la surface
    string ccont;               // couleur du contour
};

unsigned Figure::nfigs = 0;
unsigned Figure::nfgCN = 0;
unsigned Figure::nfgSB = 0;

int main(int, char*[])
{
  // Instanciation des objets
  Figure f1("rouge", "noir");
  Figure f2("bleu", "blanc");
  Figure f3;
  Figure f4("violet", "orange");

  // Exécution des traitements
  f2.setCC("noir");

  // Affichage des attributs:
```

```
  cout << "f3: "; f3.afficher();
  cout << "f2: "; f2.afficher();

  cout << "Figure::nfigs = " << Figure::nfigs << endl;
  cout << "f2.nfigs = " << f2.nfigs << endl;
  cout << "f1.nfgCN = " << f1.nfgCN << endl;
  cout << "f4.nfgSB = " << f4.nfgSB << endl;
}
```

Sans l'exécuter, indiquez l'affichage exact produit par son exécution.

Aide simple

Il s'agit de faire attention à ce qui est **partagé** par la classe de ce qui ne l'est pas. Pour ce genre d'exercices, aidez-vous de petits schémas associés à vos objets et décrivant leur contenu, puis déroulez le programme pas à pas en mettant à jour le contenu des objets dans vos schémas.

Exécutez-le afin de vérifier vos résultats.
Résultat d'exécution :

```
f3: csurf = blanc, ccont = jaune
f2: csurf = bleu, ccont = noir
FigureP1::nfigs = 3
f2.nfigs = 3
f1.nfgCN = 1
f4.nfgSB = 0
```

1.4 Les tirelires / qztirelire

Toto achète deux tirelires et les remplit avec le même montant. Il écrit ensuite un programme orienté objet pour simuler la gestion de ses tirelires.

Voici le programme : @[qztirelire1.cpp]

```
#include <iostream>
using namespace std;
#include "Tirelire1.hpp"
#include "Tirelire2.hpp"

int main(int, char*[])
{
  gestion1();
  gestion2();  //<- AJOUT
  gestion3();  //<- AJOUT
}
```

Voici le Fichier Implémentation : @[Tirelire1.hpp]

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

// Représente des tirelires
class Tirelire1
{
  public:
    // Constructeur normal
    Tirelire1()
    : m_v()
    {}

    // Ajoute le montant à la tirelire
    void gagner(double montant)
    { m_v.push_back(montant); }

    // Retire le montant de la tirelire
    void depenser(double montant)
    {
      double somme = 0.;
      do{
        somme += m_v.back();
        m_v.pop_back();
      } while (somme < montant);

      if (somme > montant)
      { m_v.push_back(somme - montant); }
    }

    // Retourne la somme contenue dans la tirelire
    double eval() const
    {
      if (m_v.empty()) { return 0.; }

      double s = 0.;
```

```
      for (unsigned ix = 0; ix < m_v.size(); ++ix)
      { s += m_v[ix]; }

      return s;
    }

  private:
    vector<double> m_v;
};

// Remplit une tirelire
void remplir(Tirelire1& b)
{
  b.gagner(10.0);
  b.gagner(5.0);
  b.gagner(2.0);
  b.gagner(0.5);
  b.gagner(100.0);
}

// Retire d'une tirelire
void retirer1(Tirelire1& b)
{ b.depenser(15.0); }

// Retire d'une tirelire
void retirer2(Tirelire1& b)
{
  retirer1(b);
  b.depenser(10.0);
}

// Effectue une gestion de tirelires
void gestion1()
{
  typedef Tirelire1 Tirelire;

  // Instancie une Tirelire
  Tirelire b1;

  // Effectue quelques opérations
  remplir(b1);

  // Une autre Tirelire
  Tirelire &b2 = b1;
  retirer1(b1);
  retirer2(b2);

  // Affiche les sommes contenues dans les tirelires
  cout << "Gestion1 :\n";
  cout << "b1 contient " << b1.eval() << " euros\n";
  cout << "b2 contient " << b2.eval() << " euros\n";
}
#endif
```

Son code compile et s'exécute sans message d'erreur.
Qu'affiche-t-il ? Justifier votre réponse en **explicitant** les principales étapes.

Solution simple

On aura :

```
Gestion1 :
b1 contient 77.5 euros
b2 contient 77.5 euros
```

Et oui ! car `b1` et `b2` font référence au même objet (en raison de la ligne `Tirelire1 &b2 = b1`) : l'objet `b2` est un synonyme de l'objet `b1` : ils réfèrent le **même** endroit en mémoire et donc toute modification de l'un se voit chez l'autre.

Il y a deux problèmes importants avec ce programme : l'un dans la classe `Tirelire1` et l'autre dans le programme principal. Proposez des corrections à la fois sous forme d'explications en français **et** de code.

Solution simple

Le premier problème est illustré par le comportement ci-dessus. Il faut donc créer une copie de l'objet `b1` (non pas référencer) et remplacer la ligne par `Tirelire1 b2(b1);` [ici il y a effectivement une vraie copie `b2| de \lstinline`b1@].

Le deuxième problème réside dans le fait qu'aucun contrôle des cas limites n'est effectué. Ainsi si l'on dépense trop (`b1.depenser(100);`), une exception non-traitée est déclenchée. Si d'un autre côté on ne dépense rien (`b2.depenser(0);`), le montant restant dans la tirelire sera incorrectement calculé. Étant donné qu'il s'agit de situations typiques et prévisibles, on peut se permettre de traiter ces cas sans utiliser le mécanisme des exceptions en vérifiant le montant.

Proposez une méthode `depenser` plus simple et plus efficace (texte en français sur comment on pourrait la coder autrement). Implémentez votre méthode et/ou nouvelle classe `Tirelire2`.

Solution simple

Il existe plusieurs façons d'implémenter la méthode `depenser`. Une première façon consiste à insérer une valeur négative dans le vecteur en cas de dépense : ceci simplifie alors passablement le contrôle des erreurs. On peut aussi implémenter la classe des tirelires en se passant du vecteur et en ne stockant que le montant de la tirelire (un `double` au lieu d'un `vector`). Dans ce cas la méthode consiste simplement à soustraire le montant dépensé au montant de la tirelire. C'est la bonne **stratégie** : voir la classe `Tirelire2`.

Testez. Exemple d'exécution :

```
Gestion1 :
b1 contient 77.5 euros
b2 contient 77.5 euros

Gestion2 :
b1 contient 102.5 euros
```

```
b2 contient 92.5 euros

Gestion3 :
b1 contient 10 euros
OUPS... montant negatif !
b1 contient 5 euros
b1 contient 4.5 euros
OUPS... montant negatif !
b1 contient 6 euros
OUPS... montant negatif !
b1 contient -4 euros
```

Validez votre classe avec la solution.

Solution C++ @[Tirelire2.hpp]

```cpp
#ifndef TIRELIRE2_CLASS
#define TIRELIRE2_CLASS
#include <iostream>
#include <vector>
using namespace std;
#include "Tirelire1.hpp"

// Représente des tirelires (version simplifiée)
class Tirelire2
{
  public:
    // Constructeur normal
    Tirelire2()
    : m_s(0.0)
    {}

    // Ajoute le montant à la tirelire
    void gagner(double montant)
    { m_s += montant; verif(montant); }

    // Retire le montant de la tirelire:
    void depenser(double montant)
    { m_s -= montant; verif(montant); }

    // Retourne la somme de la tirelire:
    double eval() const
    { return m_s; }

  protected:
    // Vérifie que le montant est valide et que le contenu
    // est positif : on ne peut retirer plus qu'il n'y en a:
    void verif(double montant)
    {
      if (montant <= 0.0)
      { cout << "\aOUPS... montant negatif !\n"; }

      if (m_s < 0.0)
      { cout << "\aOUPS... contenu negatif !\n"; }
```

```
    }

  private:
    double m_s;     // montant dans la tirelire
};

// Effectue une gestion de Tirelire
// (version modifiée)
void gestion2()
{
  typedef Tirelire1 Tirelire;

  // Instancie une Tirelire:
  Tirelire b1;

  // Effectue quelques opérations:
  remplir(b1);

  // Une autre Tirelire:
  Tirelire b2 = b1;                 //<- MODIF
  retirer1(b1);
  retirer2(b2);

  // Affiche les sommes contenues dans les tirelires:
  cout << "Gestion2 :\n";
  cout << "b1 contient " << b1.eval() << " euros\n";
  cout << "b2 contient " << b2.eval() << " euros\n";
}

// Effectue une gestion de TirelireP1b:
// avec test de vérification des montants
void gestion3()
{
  typedef Tirelire2 Tirelire;      //<- MODIF

  // Instancie une Tirelire
  Tirelire b1;

  // Effectue quelques opérations
  cout << "Gestion3 :\n";
  b1.gagner(10.0);
  cout << "b1 contient " << b1.eval() << " euros\n";
  b1.gagner(-5.0);                  //<- ALERTE
  cout << "b1 contient " << b1.eval() << " euros\n";
  b1.depenser(0.5);
  cout << "b1 contient " << b1.eval() << " euros\n";
  b1.depenser(-1.5);                //<- ALERTE
  cout << "b1 contient " << b1.eval() << " euros\n";
  b1.depenser(10.0);                //<- ALERTE
  cout << "b1 contient " << b1.eval() << " euros\n";
}
#endif
```

2 Appliquer le cours

2.1 Classe Hasard / pghasard

Créez une classe `Hasard` modélisant des générateurs de nombres pseudo-aléatoires.

Écrivez un constructeur par défaut.

Écrivez une méthode partagée `randomize` qui initialise le générateur.

Écrivez une méthode partagée `random` qui renvoie un entier pseudo-aléatoire.

Écrivez une méthode partagée `random(n)` qui renvoie un entier pseudo-aléatoire dans `[0..n[`.

Écrivez une méthode partagée `frandom` qui renvoie un réel pseudo-aléatoire.

Validez votre classe et vos méthodes avec la solution.

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

```
#ifndef HASARD_CLASS
#define HASARD_CLASS

/**
  Générateur de nombres pseudo-aléatoires
*/
class Hasard
{
  public:
    Hasard();
    static void randomize();
    static int random();
    static int random(int n);
    static double frandom();
};
#include "Hasard.cpp"
#endif
```

```
#include <ctime>
#include <cstdlib>
using namespace std;
/**
  Constructeur normal
*/
Hasard::Hasard()
{
  randomize();
```

```
}

/**
  Initialise le générateur
*/
void Hasard::randomize()
{
  srand(time(0));
}

/**
  Entier pseudo-aléatoire
  @return Entier pseudo-aléatoire
*/
int Hasard::random()
{
  return rand();
}

/**
  Entier pseudo-aléatoire dans [0..n[
  @param[in] n - borne supérieure
  @return Entier pseudo-aléatoire dans [0..n[
*/
int Hasard::random(int n)
{
  return rand() % n;
}

/**
  Réel pseudo-aléatoire
  @return Réel pseudo-aléatoire
*/
double Hasard::frandom()
{
  return static_cast<double>(rand()) / RAND_MAX;
}
```

Écrivez un programme qui saisit la borne supérieure des entiers pseudo-aléatoires dans un entier `vmax` et le nombre de tirages dans un entier `n`.

Instanciez un générateur de générateur de nombres pseudo-aléatoires.

Effectuez `n` tirages d'entiers dans l'intervalle `[0..vmax[`.

Testez. Résultats d'exécution montrant que le caractère pseudo-aléatoire :

```
Borne superieure vmax? 10
Nombre de tirages? 20
3 0 7 8 4 3 0 8 8 3 2 5 4 1 6 5 4 1 5 5
```

Validez votre programme avec la solution.

Solution C++ @[pghasard.cpp]

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

#include "Hasard.hpp"
int main()
{
  int vmax;
  cout<<"Borne superieure vmax? ";
  cin>>vmax;
  int n;
  cout<<"Nombre de tirages? ";
  cin>>n;
  Hasard::randomize();
  for (int j = 1; j <= n; ++j)
  {
    cout<<Hasard::random(vmax)<<" ";
  }
  cout<<endl;
}
```

2.2 Jeu de la fourchette OO / pgjeufourche

Objectif
Cet exercice réalise une classe du jeu de la fourchette : l'algorithme tire un entier au hasard entre 1 et 100 puis demande à l'utilisateur de le trouver.

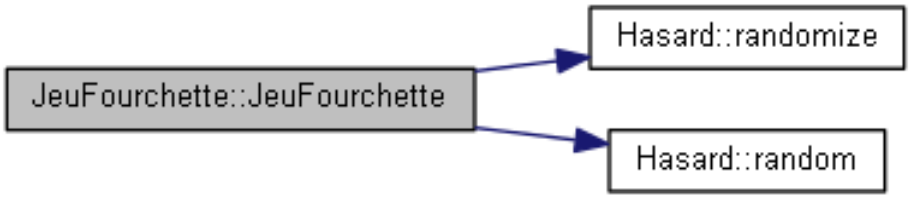
2.2.1 Classe JeuFourchette

Soit la classe `Hasard` modélisant des générateurs de nombres pseudo-aléatoires.

C++ @[Hasard.hpp] @[Hasard.cpp]

Écrivez une classe `JeuFourchette` comprenant l'entier `mystere` à trouver.

Écrivez un constructeur à un paramètre `vmax` qui tire au hasard un entier dans `[0..vmax[` pour initialiser son attribut.



Écrivez un accesseur `getMystere` de l'entier à trouver.

Écrivez une méthode `compare(nombre)` qui teste et renvoie -1, 0 ou 1 selon que l'entier mystère est inférieur, égal, supérieur à un entier `nombre`.

Écrivez une méthode `afficher` qui affiche l'entier mystère.

Validez votre classe et vos méthodes avec la solution.

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

```
#ifndef JEUFOURCHE_CLASS
#define JEUFOURCHE_CLASS
class JeuFourche
{
  public:
    explicit JeuFourche(int vmax);
    int compare(int nombre) const;
    void afficher() const;
    int getMystere() const;
  private:
    int m_mystere;
```

```
};
#include "JeuFourche.cpp"
#endif
```

```
#include <iostream>
using namespace std;
#include "Hasard.hpp"
/**
  Constructeur
*/
JeuFourche::JeuFourche(int vmax)
{
  Hasard::randomize();
   m_mystere = 1 + Hasard::random(vmax);
}

/**
  Accesseur du nombre mystere
  @return l'entier mystere
*/
int JeuFourche::getMystere() const
{
  return m_mystere;
}

/**
  Methode de comparaison
  @param[in] nombre - nombre de l'utilisateur
  @return -1, 0 ou 1 selon que mystere est inferieur, egal, superieur a nombre
*/
int JeuFourche::compare(int nombre) const
{
   if (getMystere() < nombre)
   {
     return -1;
   }
   else if (getMystere() == nombre)
   {
     return 0;
   }
   else
   {
      return 1;
   }
}

/**
  Affiche le nombre mystere
*/
void JeuFourche::afficher() const
{
  cout<<"Mystere = "<<getMystere()<<endl;
}
```

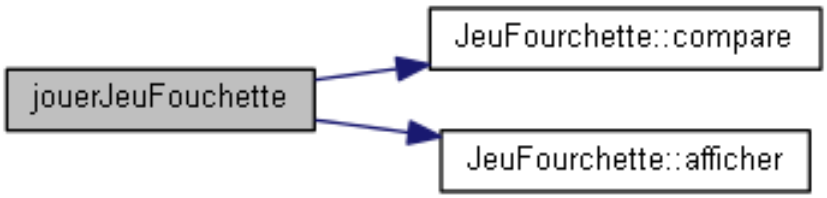
2.2.2 Programme de test

Écrivez une procédure `jouerJeuFourchette(nmax,maxessais)` qui réalise le jeu de la fourchette en tirant un entier entre 1 et `nmax` et demande de le trouver en au plus `maxessais` tentatives. Dans le cas où l'utilisateur trouve le nombre mystère, affichez le message :

```
==> Bravo, Vous avez trouve en [nessais] essai(s)
```

sinon affichez le message :

```
==> Désolé, Mystère = [mystere]
```



Testez. Exemple d'exécution avec `nmax=100` et `maxessais=7` :

```
Trouvez un entier dans [1..100] en 7 tentatives max
Votre entier? 50
C'est moins
Votre entier? 25
C'est plus
Votre entier? 40
C'est plus
Votre entier? 45
C'est moins
Votre entier? 42
==> Bravo, Vous avez trouve en 5 essai(s)
```

Validez votre procédure avec la solution.

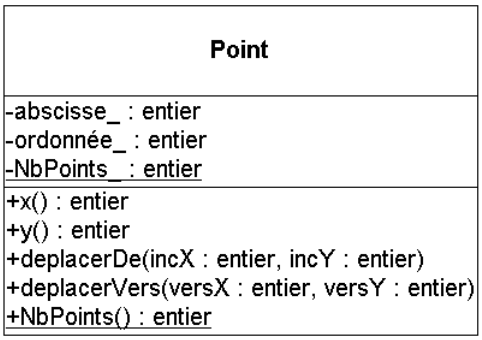
Solution C++ @[pgjeufourche.cpp]

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

#include "JeuFourche.hpp"
/**
  Lance le jeu de la Fourchette
  @param[in] nmax - borne maximale du jeu
  @param[in] maxessais - nombre maximum d'essais
*/
void jouerJeuFouche(int nmax, int maxessais)
{
  cout<<"Trouvez un entier dans [1.."<<nmax
      <<"] en "<<maxessais<<" tentatives max"
      <<endl;
  JeuFourche jeu(nmax);
  int nessais = 0;
```

```
  int nombre;
  bool succes = false;
  while (not succes and nessais < maxessais)
  {
    cout<<"Votre entier? ";
    cin>>nombre;
    int result = jeu.compare(nombre);
    if (result < 0)
    {
      cout<<"C'est moins"<<endl;
    }
    else if (result > 0)
    {
      cout<<"C'est plus"<<endl;
    }
    else
    {
      succes = true;
    }
    ++nessais;
  }
  if (succes)
  {
    cout<<"==> Bravo, Vous avez trouve en "<<nessais<<" essai(s)"<<endl;
  }
  else
  {
    cout<<"==> Desole, ";
    jeu.afficher();
  }
}

int main()
{
  jouerJeuFouche(100,7);
}
```

2.3 Classe Point (avec compteur d'instances)

Définissez une classe `Point` telle qu'explicitée sur la figure ci-dessous :



Écrivez un programme de test.

Testez.

Validez votre classe avec la solution.

Solution C++ @[Point.hpp]

```
#ifndef POINT_CLASS
#define POINT_CLASS
class Point
{
  public :
    Point(int x=0, int y=0)
    : x_(x), y_(y)
    {
      NombrePoints_ += 1;
    }

    Point(const Point& p)
    : x_(p.x()), y_(p.y())
    {
      NombrePoints_ += 1;
    }

    ~Point()
    {
       NombrePoints_ -= 1;
    }

    int x() const
    {
      return x_;
    }
```

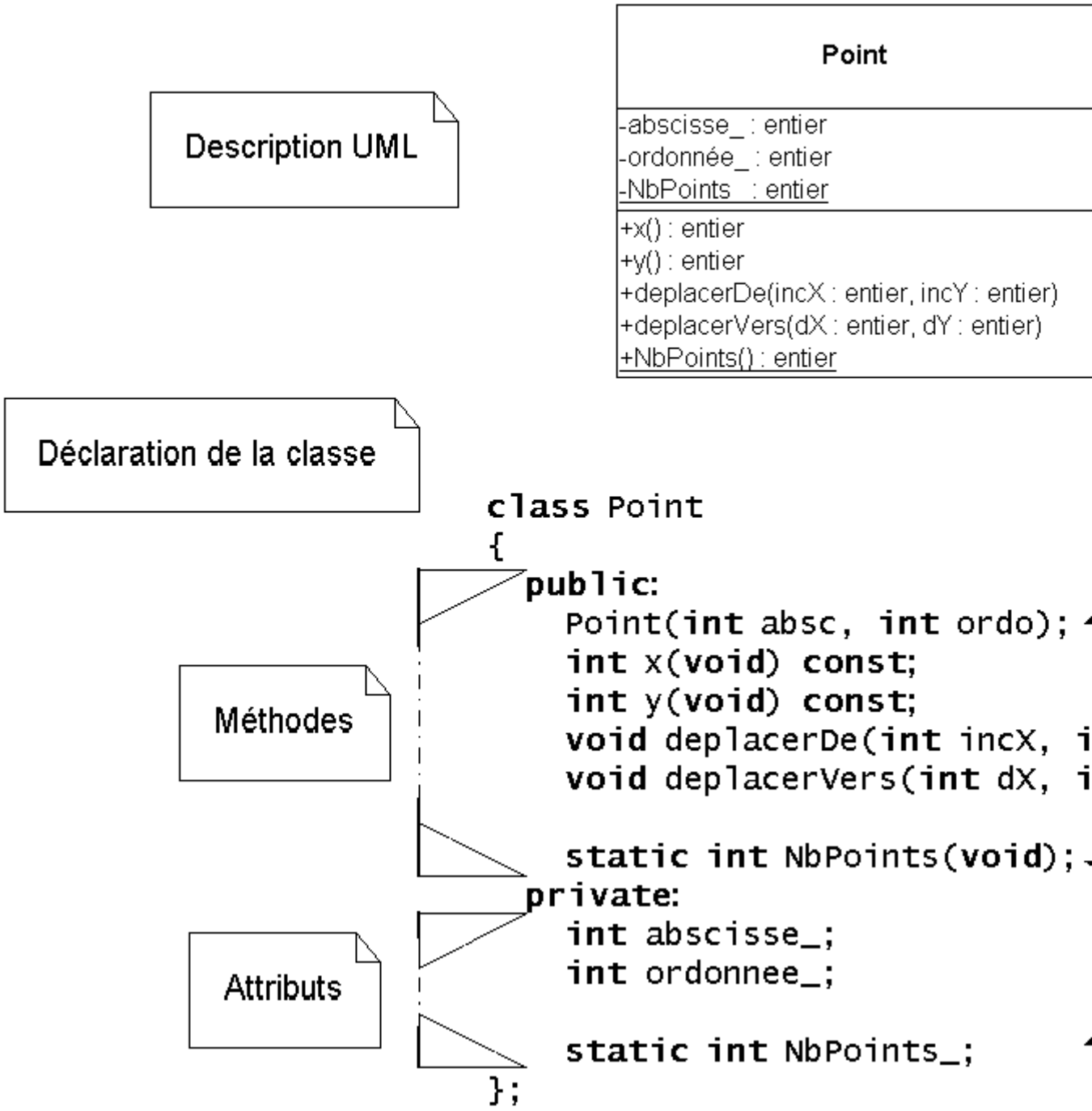
```
    int y() const
    {
      return y_;
    }

    void deplacerVers(int versX,int versY)
    {
      x_ = versX;
      y_ = versY;
    }

    void deplacerDe(int surX,int surY)
    {
      deplacerVers(x_ + surX, y_ + surY);
    }

    static int NombrePoints()
    {
      return NombrePoints_;
    }

    void print(void) const
    {
      cout<<"Abscisse "<<x()<<" Ordonnee "<<y()<<endl;
    }

  private :
    int x_, y_;
    static int NombrePoints_;
};
int Point::NombrePoints_ = 0;
#endif
```

Validez votre programme avec la solution.

Solution C++ @[pgpoint.cpp]

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

int main()
{
   Point p1;
   cout<<"Nombre d'instances = "<<Point::NombrePoints()<<" (1)"<<endl;
   {
      Point p2(5,2);
      Point p3(p2);
      Point tab[10];
      cout<<"Nombre d'instances = "<<Point::NombrePoints()<<" (13)"<<endl;
   }
   cout<<"Nombre d'instances = "<<Point::NombrePoints()<<" (1)"<<endl;
}
```

Solution C++, UML




```
int Point::NbPoints_=0;

Point::Point(int absc, int ordo)
{
  abscisse_=absc;
  ordonnee_=ordo;
}

int Point::x(void) const
```

3 Approfondir le cours