Classes géométriques [oo02] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 20 mai 2018

Table des matières

C++ - Classes géométriques (Solution)

Mots-Clés Classes ■
Requis Structures de base, Structures conditionnelles, Algorithmes paramétrés, Classes ■
Difficulté ● ○ ○ (1 h) ■

Objectif
Cet exercice réalise des classes géométriques.

1 Classe Cercle / pgcercle

1.1 Classe Cercle (du plan)

Écrivez une classe `Cercle` qui modélise des cercles du plan. Incluez un réel `rayon`.

Écrivez un constructeur à un paramètre `r` (réel) initialisant son attribut par appel à la méthode `assign(r)` définie ci-après.

Écrivez un accesseur `getRayon` du rayon.

Écrivez un mutateur `assign(r)` qui fixe le rayon au réel `r`|. La méthode doit fixer le rayon à zéro si \lstinliner@ n'est pas positif.

Écrivez une méthode `afficher` qui affiche :

```
Cercle: R=[rayon]
```

Écrivez une méthode `surface` qui calcule et renvoie la surface du cercle.

De même, écrivez une méthode `perimetre` qui calcule et renvoie le périmètre du cercle.

Validez votre classe et vos méthodes avec la solution.

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

```cpp
#ifndef CERCLE_CLASS
#define CERCLE_CLASS

/**
  Cercle (rayon)
*/
class Cercle
{
  public:
    explicit Cercle(double r);
    double getRayon() const;
    void assign(double r);
    void afficher() const;
    double surface() const;
    double perimetre() const;
  private:
    double m_rayon;
```

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

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

const double M_PI = 3.1415;
/**
  Constructeur normal
  @param[in] r - rayon du cercle
*/
Cercle::Cercle(double r)
{
  assign(r);
}

/**
  Accesseur du rayon
  @return le rayon
*/
double Cercle::getRayon() const
{
  return m_rayon;
}

/**
  Mutateur de l'attribut
  @param[in] r - rayon du cercle
*/
void Cercle::assign(double r)
{
  m_rayon = (r >= 0 ? r : 0.0);
}

/**
  Méthode d'affichage
*/
void Cercle::afficher() const
{
  cout<<"Cercle: R="<<getRayon()<<endl;
}

/**
  Méthode pour le calcul de la surface
  @return surface du cercle
*/
double Cercle::surface() const
{
  return M_PI * getRayon() * getRayon();
}

/**
  Méthode pour le calcul du périmètre
  @return périmètre du cercle
*/
double Cercle::perimetre() const
{
```

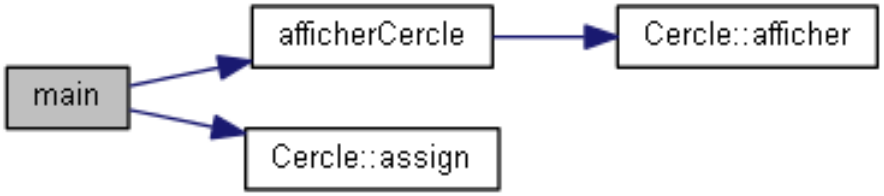
```
  return 2.0 * M_PI * getRayon();
}
```

1.2 Programme de test

Écrivez une procédure `afficherCercle(txt,obj)` qui affiche une chaîne de caractères `txt` puis les propriétés d'un \lstinlineCercle obj@.

Écrivez un programme de sorte à obtenir le résultat d'exécution suivant :

```
==> Etat initial
Cercle: R=1.5
==> Après assign valide
Cercle: R=3.4
==> Après assign invalide
Cercle: R=0
```



Validez votre procédure et programme avec la solution.

Solution C++ @[pgcercle.cpp]

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

/**
  Affiche les proprietes d'un cercle
  @param[in] txt - texte à afficher
  @param[in] obj - un Cercle
*/
void afficherCercle(const string& txt, const Cercle& obj)
{
  cout<<txt<<endl;
  obj.afficher();
}

int main(int, char*[])
{
```

```
  Cercle c(1.5);
  afficherCercle("==> Etat initial", c);
  c.assign(3.4);
  afficherCercle("==> Apres assign valide", c);
  c.assign(-5);
  afficherCercle("==> Apres assign invalide", c);
}
```

2 Classe Couronne / pgcouronne

2.1 Classe Couronne (du plan)

Ce problème utilise la classe `Cercle`.

Écrivez une classe `Couronne` comprenant un `Cercle` interne `c1` et un `Cercle` externe `c2`.

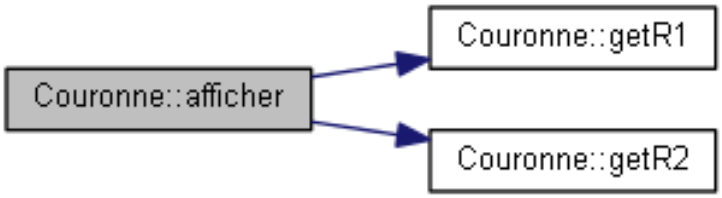
Écrivez un constructeur à deux paramètres réels initialisant ses attributs.

Écrivez des accesseurs `getR1` du rayon du cercle interne et `getR2` du rayon du cercle externe.

Écrivez un mutateur `assign(r1,r2)` qui fixe les rayons `r1` (réel) et `r2` (réel) des cercles interne et externe.

Écrivez une méthode `afficher` qui affiche :

```
couronne: R1=[r1] R2=[r2]
```



Validez votre classe et vos méthodes avec la solution.

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

```cpp
#ifndef COURONNE_CLASS
#define COURONNE_CLASS

/**
  Couronne par composition de deux Cercle
*/
#include "Cercle.hpp"
class Couronne
{
  public:
    Couronne(double r1, double r2);
    double getR1() const;
    double getR2() const;
```

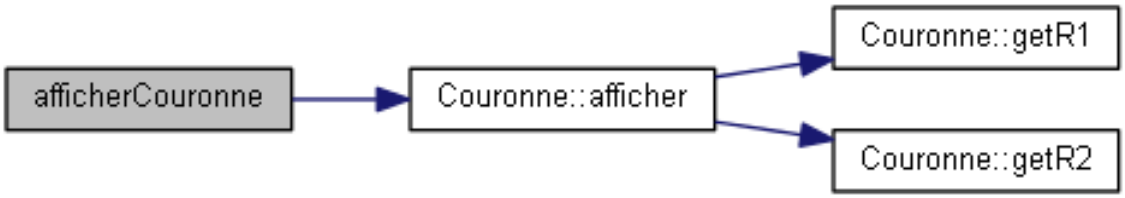
```
    void assign(double r1, double r2);
    void afficher() const;
  private:
    Cercle m_c1; // cercle interne
    Cercle m_c2; // cercle externe
};
#include "Couronne.cpp"
#endif
```

```
#include <iostream>
using namespace std;
/**
  Constructeur (assign assure la validite des parametres)
  @param[in] r1 - rayon interne
  @param[in] r2 - rayon externe
*/
Couronne::Couronne(double r1, double r2)
{
  assign(r1,r2);
}

/**
  Accesseur du rayon interne
  @return rayon interne
*/
double Couronne::getR1() const
{
  return m_c1.getRayon();
}

/**
  Accesseur du rayon externe
  @return rayon externe
*/
double Couronne::getR2() const
{
  return m_c2.getRayon();
}

/**
  Methode d'assignation
  @param[in] r1 - rayon interne
  @param[in] r2 - rayon externe
*/
void Couronne::assign(double r1, double r2)
{
  if (r1 <= r2)
  {
    m_c1 = Cercle(r1);
    m_c2 = Cercle(r2);
  }
  else
  {
    m_c1 = Cercle(r2);
    m_c2 = Cercle(r1);
  }
}
```

```
/**
  Methode d'affichage
*/
void Couronne::afficher() const
{
  cout<<"couronne: R1="<<getR1()<<" R2="<<getR2()<<endl;
}
```

2.2 Programme de test

Écrivez une procédure `afficherCouronne(txt,obj)` qui affiche une chaîne de caractères `txt` puis les propriétés d'une `Couronne obj`.



Écrivez un programme de sorte à obtenir l'exemple d'exécution :

```
==> Etat Initial
couronne: R1=1.5 R2=2.3
==> Apres assign r1 negatif
couronne: R1=0 R2=6.1
==> Après assign avec r1 > r2
couronne: R1=5.2 R2=8
```

Validez votre procédure et programme avec la solution.

Solution C++ @[pgcouronne.cpp]

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

/**
  Affiche les proprietes d'une couronne
  @param[in] txt - texte a afficher
  @param[in] obj - une Couronne
*/
void afficherCouronne(const string& txt, const Couronne& obj)
{
  cout << txt << endl;
  obj.afficher();
}

int main()
{
```

```
  Couronne c(1.5, 2.3);
  afficherCouronne("==> Etat Initial", c);
  c.assign(-3,6.1);
  afficherCouronne("==> Apres assign r1 negatif", c);
  c.assign(8.0,5.2);
  afficherCouronne("==> Apres assign avec r1 > r2", c);
}
```

3 Classe Rectangle / pgrectangle

3.1 Classe Rectangle (du plan)

Ce problème réalise une classe modélisant des rectangles (du plan).

Écrivez une classe `Rectangle` ayant pour attributs un réel `largeur| et un réel \lstinline`hauteur@.

Écrivez un constructeur par défaut initialisant les attributs à zéro.

Écrivez un constructeur à deux paramètres initialisant sa largeur `w` (réel) et sa hauteur `h` (réel) par appel à la méthode `assign(w,h)` définie ci-après.

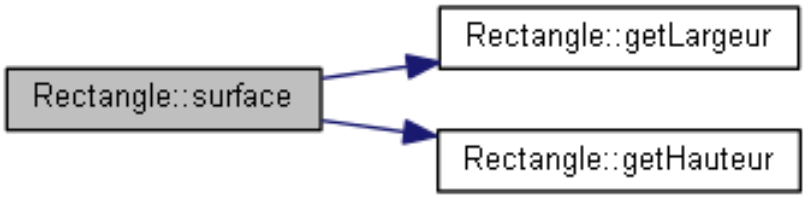
Écrivez des accesseurs `getLargeur` de la largeur et `getHauteur` de la hauteur.

Écrivez un mutateur `assign(w,h)` qui fixe la largeur `w` (réel) et la hauteur `h` (réel) du `Rectangle`. La méthode doit fixer l'attribut correspondant à zéro si le paramètre `w` (largeur) ou `h` (hauteur) n'est pas positif.

Écrivez une méthode `afficher` qui affiche :

```
rect: L=[largeur], H=[hauteur]
```

Écrivez une méthode `surface` qui calcule et renvoie la surface du rectangle.



De même, écrivez une méthode `perimetre` qui calcule et renvoie le périmetre du rectangle.

Validez votre classe et vos méthodes avec la solution.

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

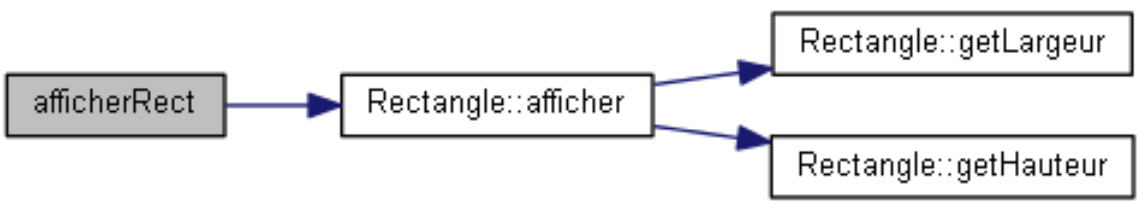
```cpp
#ifndef RECTANGLE_CLASS
#define RECTANGLE_CLASS

/**
  Rectangle (largeur, hauteur)
*/
class Rectangle
{
  public:
    Rectangle();
    Rectangle(double w, double h);
    double getLargeur() const;
    double getHauteur() const;
    void assign(double w, double h);
    void afficher() const;
    double surface() const;
    double perimetre() const;
  private:
    double m_largeur;
    double m_hauteur;
};
#include "Rectangle.cpp"
#endif
```

```cpp
#include <iostream>
using namespace std;
/**
  Constructeur par defaut
*/
Rectangle::Rectangle()
: m_largeur(0.0), m_hauteur(0.0)
{}

/**
  Constructeur normal
  @param[in] w - largeur
  @param[in] h - hauteur
*/
Rectangle::Rectangle(double w, double h)
{
  assign(w, h);
}

/**
  Accesseur de la largeur
  @return largeur du rectangle
*/
double Rectangle::getLargeur() const
{
  return m_largeur;
}

/**
  Accesseur de la hauteur
  @return hauteur du rectangle
*/
double Rectangle::getHauteur() const
```

```
{
  return m_hauteur;
}

/**
  Methode d'assignation
  @param[in] w - largeur
  @param[in] h - hauteur
*/
void Rectangle::assign(double w, double h)
{
  m_largeur = (w >= 0 ? w : 0.0);
  m_hauteur = (h >= 0 ? h : 0.0);
}

/**
  Methode d'affichage
*/
void Rectangle::afficher() const
{
  cout<<"rect: L="<<getLargeur()<<" H="<<getHauteur()<<endl;
}

/**
  Methode pour le calcul de la surface
  @return surface du rectangle
*/
double Rectangle::surface() const
{
  return getLargeur() * getHauteur();
}

/**
  Methode pour le calcul du perimètre
  @return perimètre du rectangle
*/
double Rectangle::perimetre() const
{
  return 2*(getLargeur() + getHauteur());
}
```

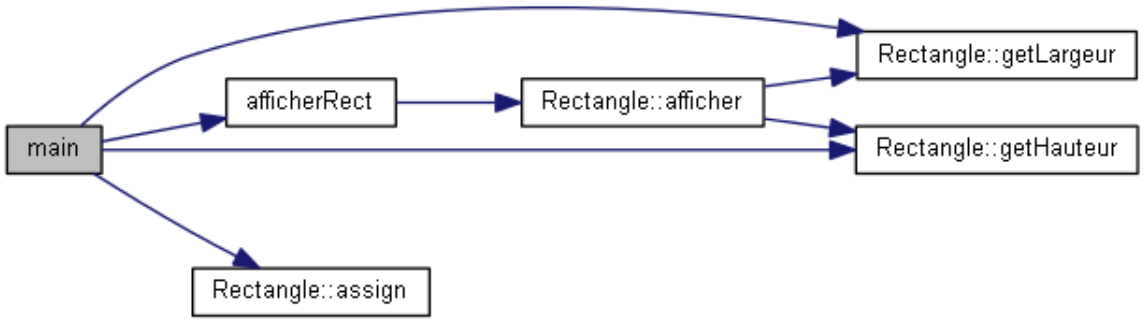
3.2 Programme de test

Écrivez une procédure `afficherRect(txt,obj)` qui affiche une chaîne de caractères `txt` puis les propriétés d'un `Rectangle obj`.



Écrivez un programme de sorte à obtenir le résultat d'exécution suivant :

```
==> Etat Initial
rect: L=1.5 H=12.8
==> Apres assign valide
rect: L=3.2 H=6.9
==> Apres permutation des dimensions
rect: L=6.9 H=3.2
==> Après assign largeur invalide
rect: L=0 H=3.2
```



Validez votre procédure et programme avec la solution.

Solution C++ @[pgrectangle.cpp]

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

/**
  Affiche les proprietes d'un rectangle
  @param[in] txt - texte à afficher
  @param[in] obj - un Rectangle
*/
void afficherRect(const string& txt, const Rectangle& obj)
{
  cout << txt << endl;
  obj.afficher();
}

int main()
{
  Rectangle r(1.5, 12.8);
  afficherRect("==> Etat Initial", r);
  r.assign(3.2, 6.9);
  afficherRect("==> Apres assign valide", r);
  r.assign(r.getHauteur(), r.getLargeur());
  afficherRect("==> Apres permutation des dimensions", r);
  r.assign(-3.4, r.getHauteur());
  afficherRect("==> Apres assign largeur invalide", r);
}
```

4 Classe Carre / pgcarre

4.1 Classe Carre (du plan)

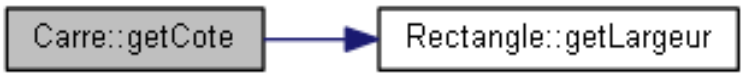
Ce problème utilise la classe `Rectangle`.

Écrivez une classe `Carre` comprenant un `Rectangle rc`.

Écrivez un constructeur à un paramètre `lgr` (réel) initialisant son attribut par par appel à la méthode `assign(lgr)`.

Écrivez un accesseur `getCote` de la longueur des côtés.

Écrivez un mutateur `assign(lgr)` qui fixe la longueur `lgr` (réel) des côtés.



Écrivez une méthode `afficher` qui affiche :

```
carre: C=[lgr]
```

Écrivez une méthode `surface` qui calcule et renvoie la surface du carré.

De même, écrivez une méthode `perimetre` qui calcule et renvoie le périmètre du carré.

Validez votre classe et vos méthodes avec la solution.

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

```
#ifndef CARRE_CLASS
#define CARRE_CLASS
#include <cmath>
using namespace std;

/**
  Carre par composition d'un Rectangle
*/
#include "Rectangle.hpp"
class Carre
{
  public:
    explicit Carre(double lgr);
    double getCote() const;
    void assign(double lgr);
    void afficher() const;
    double surface() const;
    double perimetre() const;
  private:
    Rectangle m_rc;
};
#include "Carre.cpp"
#endif
```

```
#include <iostream>
using namespace std;
/**
  Constructeur
  Initialisation par liste (et non methode assign) parce que la
  classe Rectangle ne dispose pas d'un constructeur par defaut
  @param[in] lgr - longueur du côte
*/
Carre::Carre(double lgr)
: m_rc(lgr, lgr)
{}

/**
  Accesseur de la longueur du côte
  @return longueur du côte
*/
double Carre::getCote() const
{
  return m_rc.getLargeur();
}

/**
  Methode d'assignation
  @param[in] lgr - longueur du cote
*/
void Carre::assign(double lgr)
{
  m_rc.assign(lgr, lgr);
}

/**
  Methode d'affichage
*/
```

```
void Carre::afficher() const
{
  cout<<"carre: C="<<getCote()<<endl;
}

/**
  Methode pour le calcul de la surface
  @return surface du carre
*/
double Carre::surface() const
{
  return m_rc.surface();
}

/**
  Methode pour le calcul du perimetre
  @return perimètre du carre
*/
double Carre::perimetre() const
{
  return m_rc.perimetre();
}
```

4.2 Programme de test

Écrivez une procédure `afficherCarre(txt,obj)` qui affiche une chaîne de caractères `txt` puis les propriétés d'un `Carre obj`.

Écrivez un programme de sorte à obtenir l'exemple d'exécution :

```
==> Etat Initial
carre: C=1.5
==> Apres assign valide
carre: C=6.5
==> Après assign invalide
carre: C=0
```

Validez votre procédure et programme avec la solution.

Solution C++ @[pgcarre.cpp]

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

/**
  Affiche les proprietes d'un carre
  @param[in] txt - texte a afficher
  @param[in] obj - un Carre
*/
void afficherCarre(const string& txt, const Carre& obj)
```

```
{
  cout<<txt<<endl;
  obj.afficher();
}

int main()
{
  Carre c(1.5);
  afficherCarre("==> Etat Initial", c);
  c.assign(6.5);
  afficherCarre("==> Apres assign valide", c);
  c.assign(-3.4);
  afficherCarre("==> Apres assign invalide", c);
}
```

5 Références générales

Comprend [Chappelier-CPP1 :c8 :et, c8 :ex46..47] ■