

Classes géométriques [oo02] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel

sciel

algotprog

UNIVERSITÉ
HAUTE-ALSACE

Version 20 mai 2018

Table des matières

1	Classe Cercle / pgcercle	2
1.1	Classe Cercle (du plan)	2
1.2	Programme de test	2
2	Classe Couronne / pgcouronne	4
2.1	Classe Couronne (du plan)	4
2.2	Programme de test	4
3	Classe Rectangle / pgrectangle	6
3.1	Classe Rectangle (du plan)	6
3.2	Programme de test	6
4	Classe Carre / pgcarre	8
4.1	Classe Carre (du plan)	8
4.2	Programme de test	8

C++ - Classes géométriques (TP)



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.

1.2 Programme de test



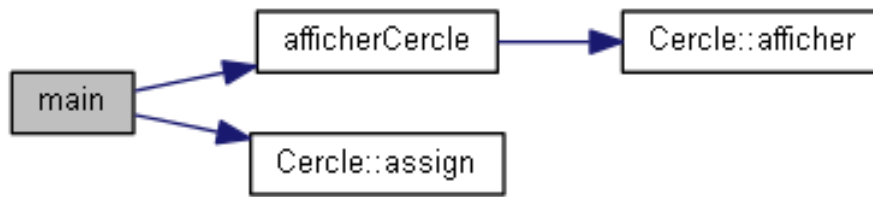
Écrivez une procédure `afficherCercle(txt,obj)` qui affiche une chaîne de caractères `txt` puis les propriétés d'un `\lstinlineCercle obj@`.



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
  
```



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.

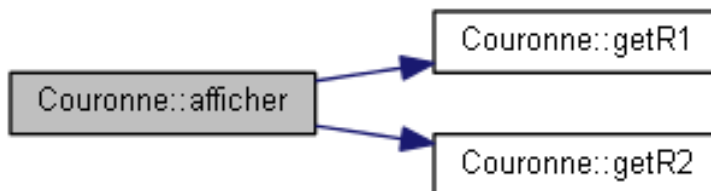


Écrivez un mutateur `assign(r1,r2)` qui fixe les rayons `r1` (réel) et `r2` (réel) des cercles interne et externe.



Écrivez une méthode `afficher` qui affiche :

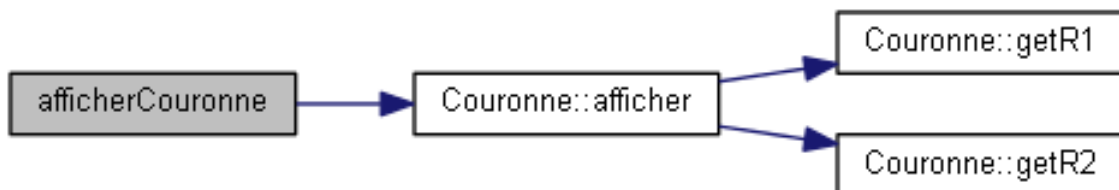
```
couronne: R1=[r1] R2=[r2]
```



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  
==> Après assign r1 négatif  
couronne: R1=0 R2=6.1  
==> Après assign avec r1 > r2  
couronne: R1=5.2 R2=8
```

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 `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.

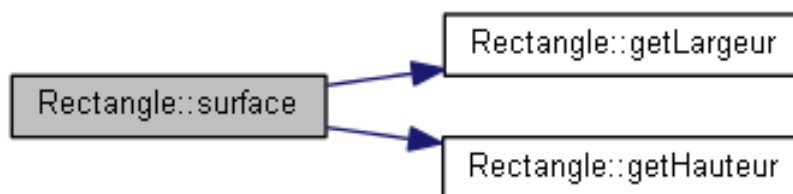


Écrivez une méthode `afficher` qui affiche :

```
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 périmètre du rectangle.

3.2 Programme de test

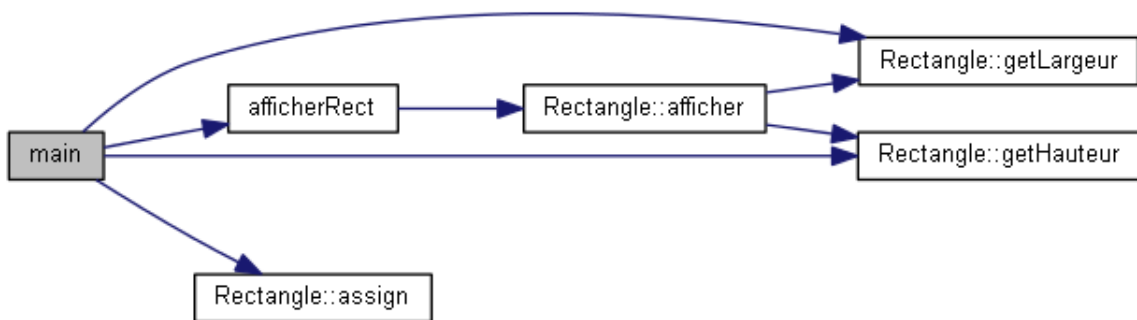


Écrivez une procédure `afficherRect(txt,obj)` qui affiche une chaîne de caractères `txt` puis les propriétés d'un `Rectangle obj`.



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

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



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.

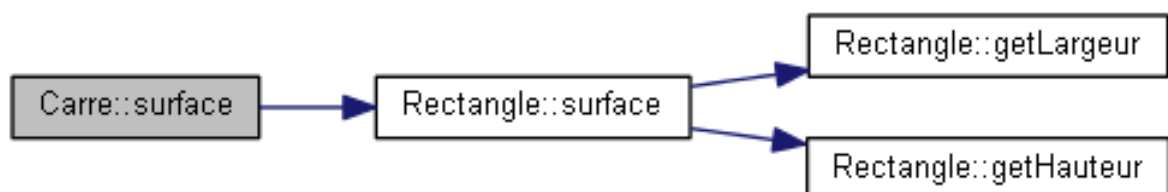


Écrivez une méthode [afficher](#) qui affiche :

```
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.

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
```