Classes géométriques [oo02] - Exercice

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 20 mai 2018

Table des matières

Python - Classes géométriques (Solution)

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

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 Python @[Cercle.py]

```python
import math
class Cercle:
   def __init__(self,r):
      self.assign(r)

   def getRayon(self):
      return self.m_rayon

   def assign(self,r):
      self.m_rayon = r if (r >= 0) else 0.0

   def afficher(self):
      print("Cercle: R=",self.getRayon())

   def surface(self):
      return math.pi * self.getRayon() * self.getRayon()
```

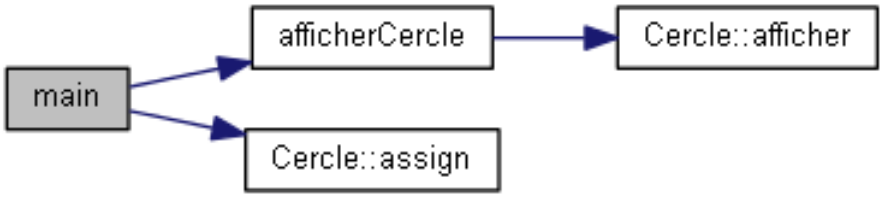
```
  def perimetre(self):
    return 2.0 * math.pi * self.getRayon()
```

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 script 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 script avec la solution.

Solution Python @[pgcercle.py]

```
from Cercle import Cercle

def afficherCercle(txt,obj):
   print(txt)
   obj.afficher()

c = Cercle(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`.

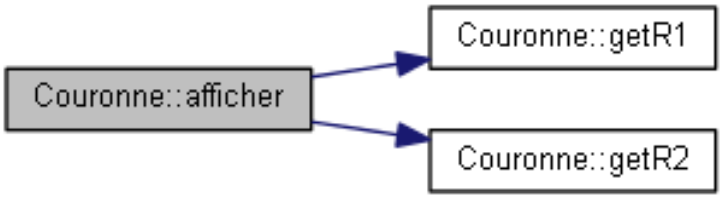
Écrivez un constructeur à deux paramètres réels initialisant ses attributs.

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.

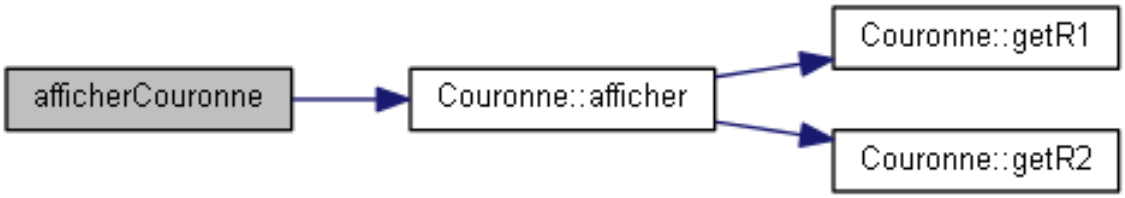
Solution Python @[Couronne.py]

```
from Cercle import Cercle
class Couronne:
   def __init__(self,r1,r2):
      self.assign(r1,r2)

   def getR1(self):
      return self.m_c1.getRayon()

   def getR2(self):
      return self.m_c2.getRayon()

   def assign(self,r1,r2):
      if (r1 <= r2):
```

```
            self.m_c1 = Cercle(r1)
            self.m_c2 = Cercle(r2)
        else:
            self.m_c1 = Cercle(r2)
            self.m_c2 = Cercle(r1)

    def afficher(self):
        print("couronne: R1=",self.getR1()," R2=",self.getR2())
```

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 script 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 script avec la solution.

Solution Python @[pgcouronne.py]

```
from Couronne import Couronne

def afficherCouronne(txt,obj):
    print(txt)
    obj.afficher()

c = Couronne(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 \lstinlinehauteur@`.

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.

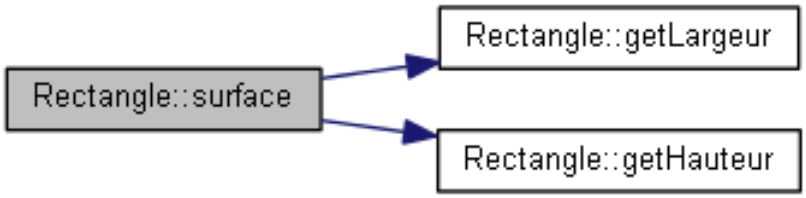
Écrivez des accesseurs `getLargeur` de la largeur et `getHauteur` de la hauteur.

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 Python @[Rectangle.py]

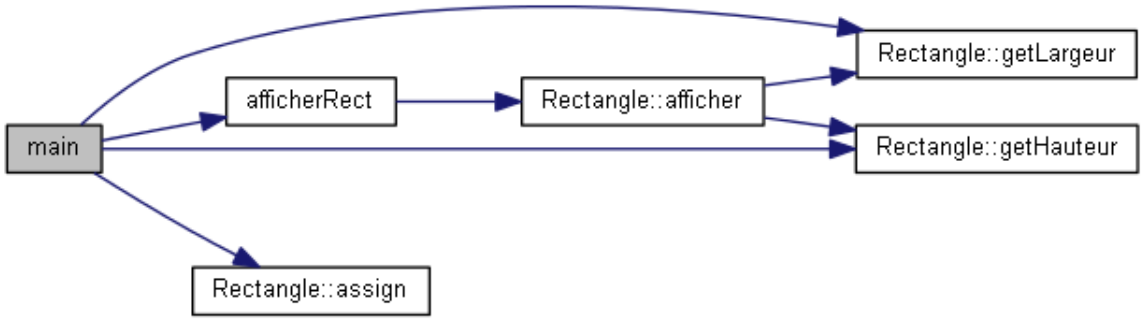
```
class Rectangle:
   def __init__(self,w=0.0,h=0.0):
      self.assign(w,h)

   def getLargeur(self):
      return self.m_largeur

   def getHauteur(self):
      return self.m_hauteur

   def assign(self,w,h):
      self.m_largeur = (w if w >= 0.0 else 0.0)
      self.m_hauteur = (h if h >= 0.0 else 0.0)

   def afficher(self):
      print("rect: L=",self.getLargeur()," H=",self.getHauteur())

   def surface(self):
      return self.getLargeur() * self.getHauteur()

   def perimetre(self):
      return 2*(self.getLargeur() + self.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 script 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 script avec la solution.

Solution Python @[pgrectangle.py]

```
from Rectangle import Rectangle

def afficherRect(txt,obj):
   print(txt)
   obj.afficher()

r = Rectangle(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)

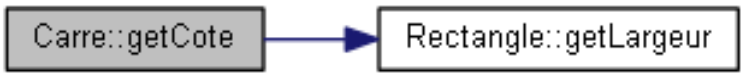
Ce problème utilise la classe `Rectangle`.

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 Python @[Carre.py]

```
from Rectangle import Rectangle
class Carre:
   def __init__(self,lgr):
      self.m_rc = Rectangle(lgr, lgr)

   def getCote(self):
      return self.m_rc.getLargeur()

   def assign(self,lgr):
     self.m_rc.assign(lgr, lgr)

   def afficher(self):
      print("carre: C=",self.getCote())

   def surface(self):
      return self.m_rc.surface()

   def perimetre(self):
      return self.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 script 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 script avec la solution.

Solution Python @[pgcarre.py]

```
from Carre import Carre

def afficherCarre(txt,obj):
   print(txt)
   obj.afficher()

c = Carre(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] ■