Exercices POO (1/2)

A l'aide du cours sur le POO et des diagrammes UML, établir les différentes classes des exercices ci-dessous, puis valider votre programme en langage python.

Exercice 1 : Classe Cercle

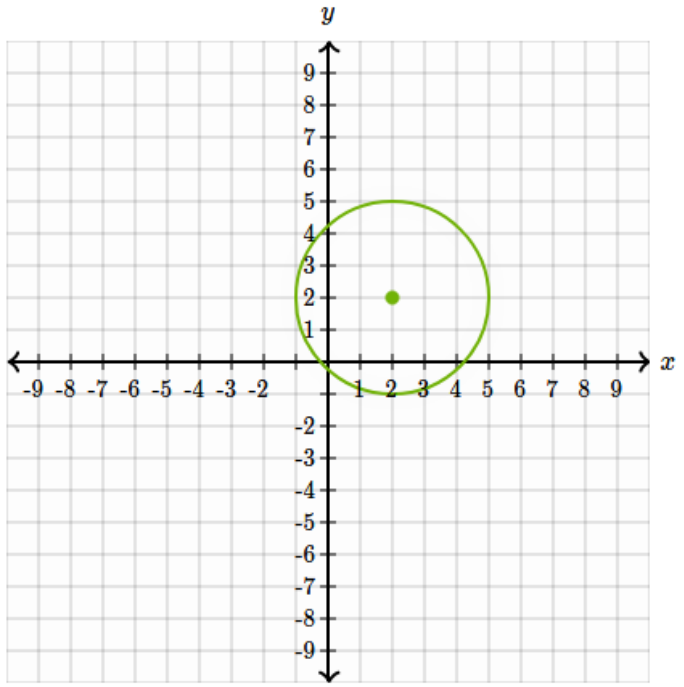
Définir une classe Cercle permettant de créer, dans un plan cartésien, un cercle C(O,r) de centre O(a,b) et de rayon r.

- Le centre O(a,b) est un tuple.
- Définir une méthode aire() de la classe qui permet de calculer la surface du cercle.
- Définir une méthode perimetre() de la classe qui permet de calculer le périmètre du cercle.
- Définir une méthode testAppartenance(A) de la classe qui permet de tester si un point A(x,y) se situe à l'intérieur du cercle C(O,r). Le point A est un tuple.

Pour tester si le point se situe dans le cercle, il suffit de savoir si la distance du point au centre du cercle est inférieure au rayon du cercle comme le montre le pseudo-code suivant :

```
si racine_carre((x_point - x_centre)² + (y_ point - y_ centre)²) <= rayon alors
        dansLeCercle← vrai          #le point est dans le cercle
sinon
        dansLeCercle← faux          #le point n'est pas dans le cercle
finsi
```

Vérification :

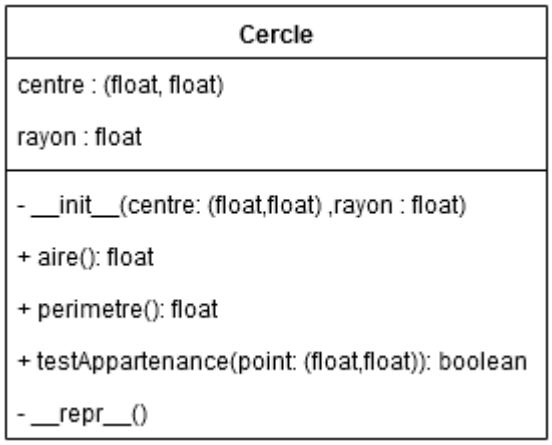



```
c1=Cercle((2,2),3)
print(c1)
print(c1.testAppartenance((0,0)))
print(c1.testAppartenance((0,-1)))
```

Résultat dans la console

P=18.84955592153876 , A=28.274333882308138
True
False

Diagramme UML :

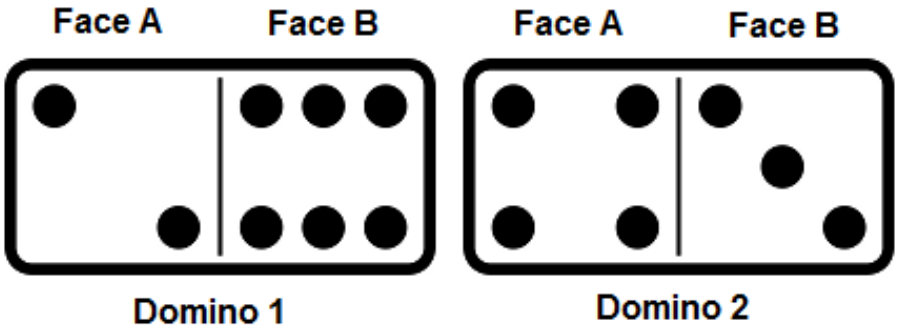


Correction :

```
from math import pi,sqrt
class Cercle:
    """
    Calculs du périmètre et de l'aire d'un cercle.
    test d'appartenance à un point
    """
    def __init__(self, centre, rayon):
        """
        Initialise le cercle avec le centre et le rayon
        """
        self.centre=centre
        self.rayon=rayon

    def perimetre(self):
        """
        calcul du périmètre du cercle
        """
        return self.rayon*2*pi

    def aire(self):
        """
        calcul de l'aire du cercle
        """
        return pi*self.rayon**2

    def testAppartenance(self,point):
        """
        tester si un point(x,y) appartient ou non au cercle
        """
        a,b=self.centre
        x,y=point
        distance=sqrt((x-a)**2+(y-b)**2)
        if distance<=self.rayon:
            dansLeCercle=True
        else:
            dansLeCercle=False
        return dansLeCercle

    def __repr__(self):
        """
        Affichage du périmètre P et de l'aire A
        """
        return ("P=" + str(self.perimetre()) + " , " + "A=" + str(self.aire()))

c1=Cercle((2,2),3)
print(c1)
print(c1.testAppartenance((0,0)))
print(c1.testAppartenance((0,-1)))
```

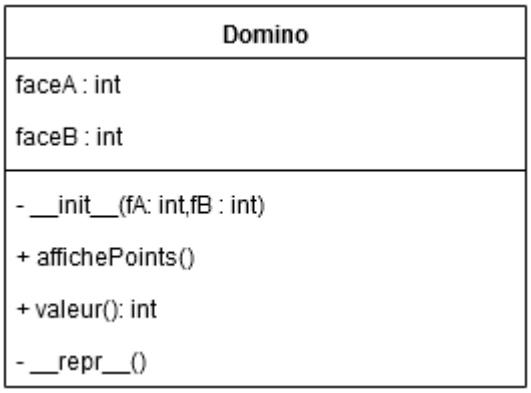
Exercice 2 : Classe Domino

Vous devez définir une classe Domino qui permette d'instancier des objets simulant les pièces d'un jeu de dominos.



- Le constructeur de cette classe initialisera les valeurs des points présents sur les deux faces A et B du domino (valeurs par défaut à 0).
- La méthode affichePoints() affiche les points présents sur les deux faces d'un domino.
- La méthode valeur() renvoie la somme des points présents sur les 2 faces des deux dominos.
- La méthode __repr__() affiche un tuple des points présents sur les deux faces A et B d'un domino.

Diagramme UML :



Vérification :

```
d1=Domino(2,6)
d2=Domino(4,3)
d1.affichePoints()
d2.affichePoints()
print("total des points :", d1.valeur() +
d2.valeur())
print(d1)
```

Résultat dans la console

face A : 2 face B : 6
face A : 4 face B : 3
total des points : 15
domino (2, 6)

Correction :

```
class Domino:
    """
    simulateur de domino
    """
    def __init__(self, fA=0, fB=0):
        """
        Initialise le domino avec des deux faces
        """
        self.faceA=fA
        self.faceB=fB

    def affichePoints(self):
        """
        affiche les points des 2 faces
        """
        print('face A :',self.faceA,'face B :',self.faceB)

    def valeur(self):
        """
        calcul de la valeur des 2 faces A + B
        """
        return self.faceA+self.faceB

    def __repr__(self):
        """
        Affichage des point des deux faces sour forme de tuple
        """
        return "domino " + str((self.faceA,self.faceB))

d1=Domino(2,6)
d2=Domino(4,3)
d1.affichePoints()
d2.affichePoints()
print("total des points :", d1.valeur() + d2.valeur())
print(d1)
```

Exercice 3 : Classe Employé

Vous devez définir une classe Employe caractérisée par les attributs : matricule, nom, prenom, dateNaissance, dateEmbauche, salaire.

- Ajouter à la classe la méthode age( ) qui retourne l'âge de l'employé.
- Ajouter à la classe la méthode anciennete( ) qui retourne le nombre d'années d'ancienneté de l'employé.
- Ajouter à la classe la méthode augmentationDuSalaire( ) qui augmente le salaire de l'employé en prenant en considération l'ancienneté en utilisant le pseudo-code suivant :

```
si ancienneté < 5 ans, alors
        on ajoute 2%
sinon si Ancienneté < 10 ans, alors
            on ajoute 5%
      sinon
            on ajoute 10%
      fin si
fin si
```

- Ajouter la méthode afficherEmploye() qui affiche les informations de l'employé : matricule, nom, prenom, age, ancienneté et salaire en €.

Diagramme UML :

Employe
matricule : int nom : str prenom : str dateNaissance : (int,int,int) dateEmbauche: (int,int,int) salaire: float
- __init__(mat: int, nom : str, prenom : str, dateNaissance:(int,int,int), dateEmbauche:(int,int,int), salaire : float) + age() : int + anciennete( ) : int + augmentationDuSalaire( ) + afficherEmploye()

Vérification :

```
agent=Employe('007','Bond','James',(11,11,1970),(7,4,1995),7500)
agent.augmentationDuSalaire()
agent.afficherEmploye()
```

Résultat dans la console

Matricule : 007
Nom : Bond
Prénom : James
Age : 49
Ancienneté : 24
Salaire en € : 8250.0

Remarque : Instructions permettant de récupérer l'année courante de l'horloge du PC

```
import datetime
date = datetime.datetime.now()
annee=date.year
```

Correction :

```
import datetime

class Employe:
    """
    Gestion d'employés
    """
    def __init__(self, mat, nom, prenom,dateNaissance,dateEmbauche,salaire):
        """
        Initialise un employé
        """
        self.matricule=mat
        self.nom=nom
        self.prenom=prenom
        self.dateNaissance=dateNaissance
        self.dateEmbauche=dateEmbauche
        self.salaire=salaire

    def age(self):
        """
        retourne l'age de l'employé
        """
        date = datetime.datetime.now()
        annee=date.year
        return annee-self.dateNaissance[2]

    def anciennete(self):
        """
        retourne le nombre d'année d'ancienneté de l'employé
        """
        date = datetime.datetime.now()
        annee=date.year
        return annee-self.dateEmbauche[2]

    def augmentationDuSalaire(self):
        """
        Augemente le salaire de l'employé
        """
        nbAnneesAnciennete=self.anciennete()
        if nbAnneesAnciennete<5:
            self.salaire*=1.02
        elif nbAnneesAnciennete<10:
            self.salaire*=1.05
        else:
            self.salaire*=1.10

    def afficherEmploye(self):
        """
        Affiche les informations de l'employé
        """
        print('Matricule :',self.matricule)
        print('Nom :',self.nom)
        print('Prénom :',self.prenom)
        print('Age :',self.age())
        print('Ancienneté :',self.anciennete())
        print('Salaire en €:',self.salaire)

agent=Employe('007','Bond','James',(11,11,1970),(7,4,1995),7500)
agent.augmentationDuSalaire()
agent.afficherEmploye()
```