TP3
Héritage & mapReduce

Dans ce travail, l'objectif n'est pas de créer une application iOS. Vous allez concevoir quelques classes dans une application console OSX. Les classes auront une hiérarchie, et les résultats s'afficheront dans la console. L'intérêt et de manipuler le langage Swift à travers la POO, l'héritage, les protocoles, et le stockage des objets dans une structure.

1. Héritage et classe fille

A. Créer une classe de base nommée « Forme ». Cette classe aura les attributs suivants :

- nbDroite : Float, la forme est composée de X segments
- nbCourbe : Float, la forme est composée de X courbes
- formeFerme : Bool, la forme a un tracé ouvert ou fermé

Vous aurez les méthodes suivantes :

- Init : Constructeur paramétré
- getForme : affichage de tous les attributs dans la console avec println()
- isQuadrilatere -> Bool : détermine si la forme est un quadrilatère
- isCerle -> Bool : détermine si la forme est un cercle
- isTriangle -> Bool : détermine si la forme est un triangle
- Surface -> Float : calcul de la surface, renvoie 0 pour l'instant
- Perimètre -> Float : calcul du périmètre, renvoie 0 pour l'instant

Cette classe forme hérite de la classe couleur :

```
import Foundation

class couleur  {

    var couleur : String?

    init(couleur : String) {
        if couleur == "rouge" {
            self.couleur = "🟥"
        }
        if couleur == "vert" {
            self.couleur = "🟩"
        }
        if couleur == "bleu" {
           self.couleur = "🟦"
        }
        if couleur == "yellow" {
           self.couleur = "🟨"
        }
        if couleur == "gris" {
            self.couleur = "⬛"
        }
        if couleur == "orange" {
            self.couleur = "🟧"
        }
    }
}
```

Implémenter les méthodes et ajouter les propriétés nécessaires dans la classe

« Forme ». Prenez en compte l'héritage.

B. Les classes « cercle », « carré » et « rectangle » héritent de la classe Forme. Les méthodes suivantes pourront donc être surchargées (override)

- Surface() qui retournera la surface de la forme concernée (Float)
- Perimètre() qui retournera le périmètre de la forme concernée (Float)

La classe « carré » possède un attribut « côté », la classe rectangle deux attributs « longueur » et « largeur ». La classe cercle possède un attribut « rayon ».

Implémenter les méthodes et ajouter les propriétés nécessaires dans les classes « cercle », « carré » et « rectangle ».

C. Tester votre travail à travers un code principal.

Le fichier main.swift permet de tester vos classes :
Créer deux formes A et B avec :
A : 3 droites, 0 courbes, formeFerme : true, couleur : vert
B : 0 droite, 1 courbe, formeFerme : false, couleur bleu

2. Héritage et protocole

A. Le protocole « sommets » est le suivant :

```
import Foundation

protocol sommets {
    var lesSommets : [String] { set get }
    var nbSommets : Int { set get }
    func listeSommets() -> String
}
```

Ce protocole permet de connaître les sommets d'une figure géométrique, et son nombre. Une méthode permet de lister les sommets.

Ce protocole est ajouté uniquement aux classes « carré » et « rectangle ». La classe « cercle » et la classe « Forme » ne sont pas concernées.

Réaliser l'héritage multiple sur les deux classes « carré » et « rectangle ». Faites les modifications qui s'imposent sur ces deux classes.

B. Vous pouvez tester votre code à travers un code principal.

Créer un carré ABCD de côté 3, de couleur orange avec les sommets « A », « B », « C », « D »
Créer un rectangle EFGH de longueur 2, largeur 3, couleur rouge avec les sommets « E », « F », « G », « H »

3. Travailler avec les tableaux et le polymorphisme.

A. Créer un cercle C de rayon 4 et de couleur grise.

B. Ajoutez toutes les instances créées A, B, ABCD, EFHG, C dans un tableau de « Formes ».

C. Utilisez le polymorphisme pour trouver le total des surfaces des formes « carrées », des formes « rectangles », faites ensuite le total des deux valeurs

Ici, vous pouvez réaliser les calculs à l'aide d'une simple addition des surfaces de chaque forme.

4. Utilisation d'une technique d'analyse de données

Reprenez la question 3, est au lieu, de travailler avec les formes successives, vous allez parcourir l'ensemble des formes, et regrouper celles qui sont identiques. C'est la technique du mapReduce.

mapReduce est un algorithme utilisé en Big Data pour faire de la tendance sur les données. Il s'agit donc de regrouper les formes identiques, et de travailler par famille (suivant le type de l'objet)

A. Réaliser le filtre des figures « carrées », « rectangles », « cercles », « quadrilatères », « triangles ». Je regroupe chaque type de figure dans un tableau spécifique avec la méthode (filter).

B. Appliquer la transformation (map) de chaque figure filtrée, à l'aide de la surface.
Sur chaque tableau spécifique je remplace la figure par sa surface calculée. Cela est valable pour les « carrés », « rectangles » et « quadrilatères ».

C. Réduire (reduce) chaque tableau à sa valeur finale. Je fais la somme des surfaces de chaque tableau. Faites le calcul uniquement pour les « carrés » et les rectangles ».

Filter, Map, Reduce used in functional language and their use in Swift.

Swift standard library contains these three functions as an extension to Array(List) type and as well as separate functions. Lets see what we can do with these functions.

Map :

Map is a function that takes a list and an unary function, applies this function to each element of the list and outputs a modified list.

Example 1 :

Map each value in array to twice of the value. The new values are contained in a new arraymappedArray.
let array: [Int] = [1,2,3]
let mappedArray: [Int] = map(array, {$0 * 2})

Filter :

Filter function takes a unary function returning a boolean(predicate) and some list of values, produces a new list containig those values from the original list for which the predicate returns true.

Example 2 :

Filter out even numbers from a list.
let array: [Int] = Array(1...10)
let filteredArray: [Int] = filter(array, {return (($0 % 2) == 0)})
result:
array = [1,2,3,4,5,6,7,8,9,10]filteredArray = [2,4,6,8,10]

Reduce :

Reduce (alist, initialValue, combineFunc) function combines the values to produce a single value by applying combineFunc function on the list alist recursively.
If the list is [l1, l2, l3, l4] and the initial value l0After the first step the list looks like: [combineFunc(l0, l1), l2, l3, l4], and the next step [combineFunc(combineFunc(l0, l1), l2), l3, l4] and so on.

Example 3 - Sum of numbers in a list

let array: [Int] = Array(1...10)
let result: Int = reduce(array, 0, {$0 + $1})

Example 4 - Maximum of numbers in a list.

let array: [Int] = Array(1...10)
let result: Int = reduce(array, 0, {if $0 > $1 {return $0;} else {return $1}})

Remark :
`($0 is carre ==  true)`
Vérifie si la forme $0 est un carré est vrai.
`($0.isTriangle() == true)`
Vérifie si la forme $0 est un triangle est vrai.