R308 : Consolidation de la programmation

TDP 1: Classes et objets

1 Équation du second degré (à partir de l'exemple du cours) :

Voici la représentation UML de la classe

```
Classe : Equation2Degre
```

```
Attributs :
__a: float
__b: float
__c: float
__sol1: float
__sol2: float
```

```
Méthodes :
__init__(self):
set(self, coeff_a: float, coeff_b: float, coeff_c: float)-> None:
solutions_reelles(self)-> bool:
calcule_les_solutions(self)-> None:
get_delta(self)-> float:
get_solutions(self)-> list[float]:
get_coeff(self)-> list[float]:
get(self)-> str
```

1. Modifier la classe Equation2ndeDegre :

- définissez la méthode « solutions_reelles() » qui permet de savoir si les solutions de l'équation sont réelles ou non.
- modifiez la classe pour que l'appel de la méthode « calcule_les_solutions() » soit automatique à chaque changement des valeurs des coefficients.
- définissez la méthode get() qui retournera une chaîne de caractères qui récapitule l'équation ainsi que les solutions. La chaîne retournée sera de la forme :

```
l'équation a.x2 + b.x + c =0 a pour solutions : sol1 et sol2
ou
l'équation a.x2 + b.x + c = 0 n'a pas de solution réelle
```

2. Complétez le programme principal suivant :

```
if __name__ == "__main__":
    # Première étape : déclarez 2 références d'objet nommées eq1 et eq2

    # Seconde étape : instanciez les objets

    # troisième étape : instanciez les objets
    # initialisez eq1 avec les paramètres suivants : a=1,0 ; b=-2 ; c=1,0

    # instanciez eq2 avec les paramètres suivants : a=1,0 ; b=2,2 ; c=3,0

    # affichez des solutions des deux équations

    # affichez des paramètres des deux équations

    # afficher toutes les informations de chaque équation (méthode get() )
```

2 Gestion de stock :

Afin de gérer un stock, on se propose d'utiliser des objets de la classe Article.

2.1 Un article sera identifié avec les caractéristiques suivantes :

- Son nom
- La quantité en stock
- Le prix unitaire

2.2 Les méthodes qui permettront de manipuler les articles seront les suivantes :

1. Les modificateurs :

- Un constructeur qui permettra de modifier tous les attributs, il fera appel à la méthode set() ci dessous.
- Le modificateur set() qui permettra de modifier tous les attributs avec une seule méthode.
- La méthode acheter() qui permet d'acheter un certain nombre d'articles (le nombre sera un paramètre de la méthode), la méthode retourne le prix à payer.

2. Les observateurs :

- get() qui retourne l'ensemble des attributs de l'objet sous la forme d'une chaîne de caractères.
- get_nom() qui retourne le nom de l'article
- get_quantite() qui retourne la quantité de l'article.
- get_attributs() qui retourne un tuple avec tous les attributs
- get_dict() qui retourne un dictionnaire avec l'ensemble des attributs (clé = nom de l'attribut, valeur = valeur de l'attribut)

2.3 Écrivez la représentation UML de la classe Article.

2.4 Écrivez la définition de la classe Article.

2.5 Tester votre classe en complétant le programme principal suivant :

```
if __name__ == "__main__":
      print ("------- premiere partie, des objects simples-------")
      # declaration de 3 references d objets nommés a1, a2 et a3
      ...
      # instanciation des 3 objets
      # 100 bols a 10.23€, 50 chemises a 45.32€ et 35 guitares a 150)
      ...
      # affichage des attributs des articles avec la methode get()
      …
      # afficher le dictionnaire de chaque objet avec l'attribut __dict__ puis avec votre méthode
get_dict()
      ...
      # acheter 5 elements au 1er article et afficher de cout correspondant
      ...
      # acheter 3 elements de 2eme article et afficher de cout correspondant
      ...
      # afficher a nouveau les attributs des articles avec la methode get_attributs()
```

2.6 Gestion du stock, une classe Magasin :

Vous allez maintenant gérer l'ensemble du stock en utilisant une classe Magasin dont voici la représentation UML :

Classe Magasin	
Attributs : __liste_articles : list[Article] __chiffre_affaire : float	Le chiffre d'affaires sera mis à jour à chaque achat
Méthodes : __init__(self):	Instanciation de la liste initialisation du chiffre d'affaires
ajouter(self, nom: str, quantite: int, prix: float)-> None:	Pour ajouter un article à la liste
acheter(self, indice: int, quantite: int)-> float:	Pour acheter un article à partir de son indice dans la liste
get(self)-> str:	Retourne tous les attributs de tous les articles de la liste
get_indice(self, nom: str)->int (cette méthode est à écrire à la fin de l'exercice)	Retourne l'indice correspondant au libellé ou -1 en cas d'erreur

2.7 Complétez la programme principal aux endroits indiqués :

```
if __name__ == "__main__":
      print ("------- premiere partie, des objects simples-------")
      # déclarer de 3 références d objets
      ...
      # instancier les 3 objets
      # 100 bols a 10.23€, 50 chemises a 45.32€ et 35 guitares a 150)
      …
```

```
    # afficher les attributs des articles avec la methode get()
    ...
    # acheter 5 éléments au 1er article et afficher de cout correspondant
    …
    # acheter 3 éléments de 2eme article et afficher de cout correspondant
    ...
    # afficher a nouveau les attributs des articles avec la méthode get_attributs
    ...
    print("-------- un magasin, une liste d'articles ----------------")
    # déclare une variable de Type Magasin appeler magasin
    ...
    # instancier l'objet magasin
    ...
    # ajouter 3 articles au magasin
    # 100 bols a 10.23€, 50 chemises a 45.32€ et 35 guitares a 150)
    ...
    # afficher tous les attributs des articles du magasin
    ...
    # acheter 5 éléments au 1er article et afficher de cout correspondant
    ...
    # acheter 3 éléments de 2eme article et afficher de cout correspondant
    ...
    # afficher de chiffre d'affaires global (tous les articles)
    ...
    print("---------- acheter a partir du nom -----------")
    # acheter 2 guitares et afficher le cout
    ...
    # afficher à nouveau tous les attributs des articles du magasin
    …
```

3 Points et courbe (à partir de l'exemple du cours) :

Définissez un programme principal qui permettra d'afficher la courbe suivante :

$$f(x) = e^{-t/tau}.\cos(2.PI.f.t)$$ (sinusoïde amortie)

avec :

tau = 100 ms, f=50Hz

de 1000 points séparés de 5 ms.

4 Manipulation de complexes (à partir de l'exemple du cours) :

4.1 Écrivez la définition complète de la classe Complexe dont la représentation UML est la suivante

Classe : Complexe
Attributs : __reel: float __img: float **Attributs statiques:** MODE_REPRESENTATION : str = "math" # ou "elec"
Méthodes : __init__(self,x: float, y: float): set_rectangulaire(self,reel: float, img: float)->None: set_polaire(self,ro: float, teta: float)->None: get_reel(self)-> float get_img(self)-> float get_ro(self)-> float get_teta(self)-> float get(self)-> str: **Méthode statique :** change_mode(nouveau_mode: str)-> None

Rappel : l'attribut statique permettra de définir le contexte de travail et de modifier le format d'affichage ("math" sous la forme a + i.b et "elec" pour la forme [ro ; téta]).

4.2 Écrivez la définition de la classe MaClasseMath dont la représentation UML est la suivante :

Classe : MaClasseMath
Attributs :
Méthodes statiques multiplier(c1: Complexe, c2: Complexe)-> Complexe diviser(c1: Complexe, c2: Complexe)-> Complexe

Remarque : vous utiliserez les méthodes de la classe Complexe pour simplifier au maximum les calculs.

4.3 Le programme principal :

Complétez le programme principal au endroits indiqués :

```
if __name__ == "__main__":
      # déclarez variables de type Complexe nommés z1, z2, z3, z4 et z5
      ...
      # instanciez z1, z2, z3
      ...
      # initialisez z1 avec les valeurs [1.0 ; 45.0°]
      ...
      # instanciez z2 avec les valeurs Re= 0.7071 Im=-0.7071
      ...
      # afficher les 3 complexes
      ...
      print("---------suite------------")
      # Modifier les coordonnées de z3 avec les valeurs : Ro=2.0 Téta=90.0
      ...
      # copier z3 dans z4 en utilisant la méthode copy du module copy
      ...
      # modification de z3 pour vérification avec les valeurs Re=0.0 et Im=0.0
      ...
      # afficher z4
      ...
      # Multiplier z1 par z2 et mettre le résultat dans z5
      ...
      # Ajouter z1 par z2 et mettre le résultat dans z6
      ...
      print("---------nouveau mode affichage------------")
      # changer le mode d'affichage et afficher les – complexes
```

5 Initialisation d'attribut à partir d'un dictionnaire (A partir de l'exemple du cours) :

5.1 Soit la représentation UML suivante :

Classe : Employe
Attributs : self.__nom: str self.__age: int self.__pays: str self.__anniversaire: str self.__id: int self.__internet: bool self.__langue: str self.__genre: str
Méthodes : _init__(self, dictionnaire: dict)->None: get_nom(self)-> str: get_age(self)-> int: get_id(self)-> int: ...

5.2 Écrivez la définition de la classe Personne2

5.3 Complétez le programme suivant :

```
if __name__ == "__main__":
    dict_p = {
        "_Employe__nom": "etienne",
        "_Employe__age": 34,
        "_Employe__pays": "france",
        "_Employe__anniversaire": "1988-12-27",
        "_Employe__id": 227417393,
        "_Employe__internet": False,
        "_Employe__langue": "francais",
        "_Employe__genre": "homme"
    }
    # déclarez une variable de type Personne2
    ...
    # instanciez la variable en utilisant de dictionnaire ci dessus
    ...
    # affichez les attributs en utilisant les observateurs
    ...
```