Fait par:

* Essafi Mustapha
* Benzakour Manal
* Elouatyq Safae

RAPPORT DE PROGRAMMATION ORIENTEE OBJET

**TP N1:**

* EXERCICE 1
  + Comprehension des competences d’une classe
* EXERCICE 2
  + Encapsulation et modificateur
* EXERCICE 3
  + Definition et destruction d’un objet
* EXERCICE 4
  + Methodes et visibilite

**TP N2:**

* EXERCICE 1
  + Modelisation d’une classe
* EXERCICE 2
  + principe de l’encapsulation
* EXERCICE 3
  + Principe de l’Heritage
* EXERCICE 4
  + Principe du Polymorphysme

**TP N3:**

* EXERCICE 1
  + Polymorphisme et redèfinition des méthodes
* EXERCICE 2
  + Sucharge des opérateurs et utilisation des méthodes abstaires

**TABLE DES MATIÈRES**

1.TP N1 EXERCICE 1:

La conception:

|  |
| --- |
| PERSONNE |
| NOM = Chain  AGE = entier |
| Afficher\_détails()  Nt\_personne() |

La correction:

class personne:  
 population=0  
 def \_\_init\_\_(self,nom,age):  
 self.nom=nom  
 self.age=age  
 personne.population+=1  
 def afficher\_details(self):  
 print(f"nom={self.nom},age={self.age}")  
 @classmethod  
 def personne(cls):  
 print(f"{cls.population}")

1.TP N1 EXERCICE 2:

La conception:

|  |
| --- |
| PERSONNE |
| NOM = Chain  AGE = entier  (-)SALAIRE= réel |
| Afficher\_détails()  Get\_salaire()  Set\_salaire()  Augmant\_salaire() |

La correction:

class personne:  
 population=0  
 def \_\_init\_\_(self,porsontage):  
 self.\_\_salaire=self.\_\_salaire + porsontage \* self.\_\_salaire  
   
 population=0  
 def \_\_init\_\_(self,nom,age,\_\_salaire):  
 self.nom=nom  
 self.age=age  
 self.\_\_salaire=:\_\_salaire  
   
 personne.population+=1

@classmethod  
 def personne(cls):  
 print(f"{cls.population}")

def afficher\_details(self):  
 print(f"nom={self.nom},age={self.age},{self.\_\_salaire}")  
def get\_salaire(self):  
 return self.\_\_salaire  
def set\_salaire(self,new):  
 self.\_\_salaire=new

1.TP N1 EXERCICE 3/4:

La conception:

|  |
| --- |
| COURS |
| nom = Chain  prof = Chain  Etudiants = [ ] |
| Quitter\_cours()  Annoncer\_cours()  Cours\_popular() |

class cours:  
 def \_\_init\_\_(self,nom,prof,etudiants):  
 self.nom=nom  
 self.prof = prof  
 self.etudiants = etudiants  
 def quitter\_cours(self):  
 if self.etudiants in self.etudiants:  
 self.etudiants.remove(self.etudiants)

def annoncer\_cours(self,age):  
   
 print(f"nome={self.nom}")  
 self.prof.afficher\_details()  
   
 for x in self.etudiants:  
 if x.age >= age :  
 x.afficher\_details()  
@classmethod  
def cours\_populaire(cls,list\_cours):  
 return max(list\_cours , key=lambda cours.len(cours.etudiants))

2.TP N2 EXERCICE 1:

La conception:

|  |
| --- |
| PRODUIT |
| nom = Chain  prix= réel  quantite\_stock = entier |
| Afficher\_détails() |

La correction:   
class produit:  
 def \_\_init\_\_(self,nom,prix,quantite\_stock):  
 self.nom=nom  
 self.prix=prix  
 self.quantite\_stock=quantite\_stock

def afficher\_details(self):  
 print(f"nom: {self.nom}, \n prix:{self.prix}, \n quantite\_stock:{self.quantite\_stock}")

2.TP N2 EXERCICE 2:

La conception:

|  |
| --- |
| PRODUIT |
| (-)nom = Chain  (-)prix= réel  (-)quantite\_stock = entier |
| Get\_n ()  Set\_n ()  Get\_p ()  Set\_pr ()  Get\_q ()  Set\_q () |

La correction:

class produit:  
 n\_produit = 0  
  
 def \_\_init\_\_(self, nom, prix, quantite\_stock):  
 self.\_\_nom = nom

self.\_\_prix = prix  
 self.\_\_quantite\_stock = quantite\_stock  
 produit.n\_produit += 1  
def afficher\_details(self):  
 print(f"nom: {self.nom}, \n prix:{self.prix}, \n quantite\_stock:{self.quantite\_stock}")  
@classmethod  
def n\_produit(cls):  
 print(f"n\_produit:{cls.n\_produit}")  
def afficher\_details(self):  
 print(f"nom: {self.nom}, \n prix:{self.prix}, \n quantite\_stock:{self.quantite\_stock}")  
def get\_nom(self):  
 return self.\_\_nom  
def set\_nom(self, new):  
 self.\_\_nom = new  
def get\_prix(self):  
 return self.\_\_prix  
def set\_prix(self, price):  
 self.\_\_prix = price  
def get\_quantite(self):  
 return self.\_\_quantite\_stock  
def set\_quantite(self, newq):  
 self.\_\_quantite\_stock = newq

2.TP N2 EXERCICE 3/4:

La conception:

|  |
| --- |
| PRODUIT |
| nom = Chain  prix= réel  quantite\_stock = entier |
| Prix\_total() |

|  |
| --- |
| ALIMENT |
| date\_permetion = date |
| Prix\_total() |

La correction:  
class produit:  
 def \_\_init\_\_(self,nom,prix,quantite\_stock):  
 self.\_\_nom=nom  
 self.\_\_prix=prix

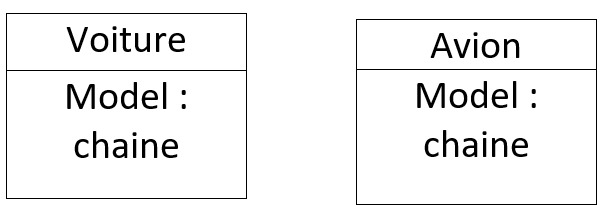
self.\_\_quantite\_stock=quantite\_stock  
  
def prix\_totale(self,q):  
 return self.prix\*q

class aliment(produit):  
 def \_\_init\_\_(self,nom,prix,quantite\_stock,date\_peremition):  
 super().\_\_init\_\_(nom,prix,quantite\_stock)  
 self.date\_peremition=date\_peremition  
  
 def prix\_totale(self,q):  
 if self.date\_peremition < "2024-09-09":  
 return super().prix\_totale(q)\* 0.10  
 else:  
 return super().prix\_totale(q)

3.TP N3 EXERCICE 1:

La conception:





La correction:

from abc import ABC , abstractmethod

class vehicule(ABC):

    @abstractmethod

    def deplacer (self):

        pass

    @staticmethod

    def prime(n):

        if n <= 1:

            return False

        for i in range(2, int(n \*\* 0.5) + 1):

            if n % i == 0:

                return False

        return True

class voiture(vehicule):

    def deplacer(self):

        print("la voiture se deplace")

class avion(vehicule):

    def deplacer(self):

        print("l'avion se deplace")

print(vehicule.prime(7))

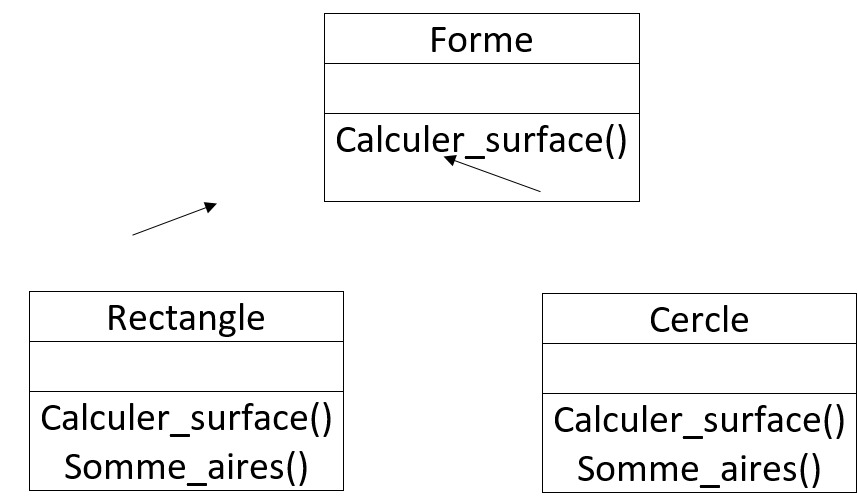
print(vehicule.prime(15))

p1=voiture()

print(p1.prime(37))

3.TP N3 EXERCICE 2:

La conception :



Le rayon:reel

Calculer\_surface()

Somme\_aires()

Longeur :reel

Largeur :reel

La correction :

from abc import ABC , abstractmethod

class form (ABC):

    @abstractmethod

    def caluler\_surface(self):

        pass

class rectangle(form):

    def \_\_init\_\_(self,largeur,hauteur):

        self.largeur=largeur

        self.hauteur=hauteur

    def \_\_add\_\_(self,x):

        return self.caluler\_surface()+x.surface()

    def caluler\_surface(self):

        return self.largeur\*self.hauteur

import math

class Cercle(form):

    def \_\_init\_\_(self,rayon):

        self.rayon=rayon

    def caluler\_surface(self):

        return math.p1\*(self.rayon\*\*2)

    def \_\_add\_\_(self,z):

        return self.caluler\_surface()+z.surface()

print(x+z)