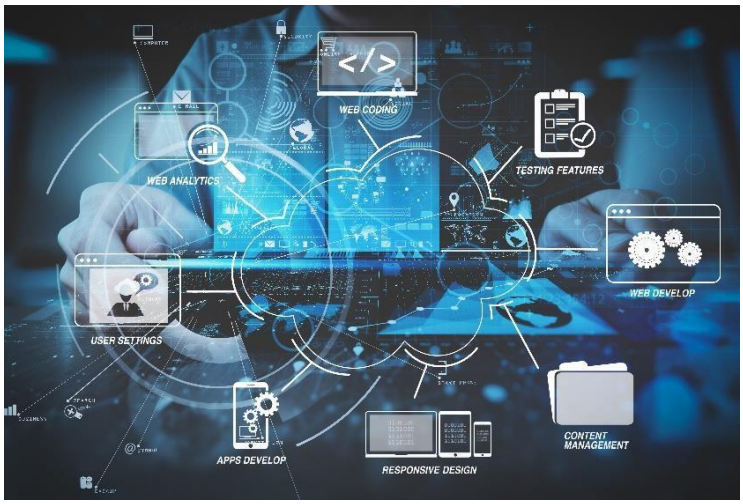
ROYAUME DU MAROC

مكتب التكوين المهني وإنعاش الشغل

Office de la Formation Professionnelle et de la Promotion du Travail

OFPPT

Rapport



Sommaire

- Définition de package
- Contrôle n 2
- Question de cours & correction
- Contrôle n 1
- Question de cours & correction

DÉFINITION DE PACKAGE :

- En programmation orientée objet (POO), un "package" fait référence à un mécanisme de regroupement de classes, interfaces, et autres éléments de code liés, permettant ainsi une meilleure organisation, modularité et encapsulation du code. Un package peut être considéré comme un conteneur logique qui rassemble des éléments de code apparentés, facilitant ainsi la gestion et la réutilisation du code.

Les packages permettent de :

✓ **Organiser le code** :

Les packages permettent de structurer le code en regroupant des classes et des interfaces apparentées, ce qui facilite la navigation et la compréhension du code pour les développeurs.

✓ **Encapsulation** :

Les classes et les interfaces peuvent être regroupées dans des packages, permettant de limiter l'accès à certaines parties du code en les déclarant comme privées ou en utilisant des modificateurs d'accès tels que `public`, `protected`, ou `private`.

✓ **Réutilisation du code** :

Les packages facilitent la réutilisation du code en permettant aux développeurs d'importer et d'utiliser des classes et des interfaces définies dans d'autres packages.

✓ **Gestion des dépendances** :

Les packages peuvent être utilisés pour gérer les dépendances entre les différents composants logiciels en spécifiant les dépendances dans les fichiers de configuration de dépendances.

Control v2

Enonce :

Etude de cas:

Partie1:

1- Definissez une classe abstraite Produit avec les attributs id_p,nomProduits et prix,incluant deux méthodes abstraites Afficher_details() et Appliquer_réduction() et ajoutez un constructeur pour initialiser les attributs.

2- implimenez une méthode statique Somme_inverse qui prend en arguments une liste, et qui renvoi la somme inverse des éléments de cette liste.

Exemple: liste=[4,7,5,2,3,10,17]

Somme croisée= -4+7-5+2-3+10-17=-10

Correction :

```
from abc import ABC,abstractmethod class
Produit(ABC):     def
_init_(self,id_produit,nom,prix):
self.id_produit = id_produit
self.nom = nom          self.prix = prix
@abstractmethod     def
Afficher_detail(self):          pass
@abstractmethod     def
appliquer_reduction(self):
        pass
@staticmethod     def
somme_inverse(liste):
        somme = 0          for i, nombre
in enumerate(liste):              if i % 2
== 0:                   somme -= nombre
else:
                somme += nombre
return somme
```

Enonce :

Partie2:

1- Créez une classe dérivée Machine qui n'est pas abstraite et qui hérite de la classe

Produit avec les attributs marque , quantité_machine, et total_prix qui bénéficie d'une réduction de 7% lorsque la quantité des machines dépasse 3,et ajoutez un constructeur pour initialiser les attributs sans passer total_prix en argument

2- Implimentez une méthode de classe pour afficher le nombre total des machines

Correction :

```
class Machine(Produit):
    nbr_machine = 0     def __init__(self, id_produit, nom, prix,
marque, quantite_machine):          super().__init__(id_produit, nom,
prix)          self.marque = marque
        self.quantite_machine = quantite_machine
        self.total_prix = self.prix * self.quantite_machine
self.total_prix = self.appliquer_reduction(7)
Machine.nbr_machine += 1     def Afficher_detail(self):
return f"id_produit: {self.id_produit}\nnom: {self.nom}\nprix:
{self.prix}\nmarque:
        {self.marque}\nquantite_machine: {self.quantite_machine}\ntotal_prix:
{self.total_prix}"     def
appliquer_reduction(self, poucentaReduire):
if self.quantite_machine > 3:
            self.total_prix = self.total_prix - (self.total_prix*(poucentaReduire/100))
return self.total_prix
    @classmethod     def nombreMachine(cls):          return f"le
nombre total des aliment est: {cls.nbr_machine}" m = Machine(123,
"sabone" ,8, "qsdfghu", 12)
print(f"le somme inverse: {Produit.somme_inverse([4, 7, 5, 2, 3, 10, 17])}")
```

Enonce :

Partie3:

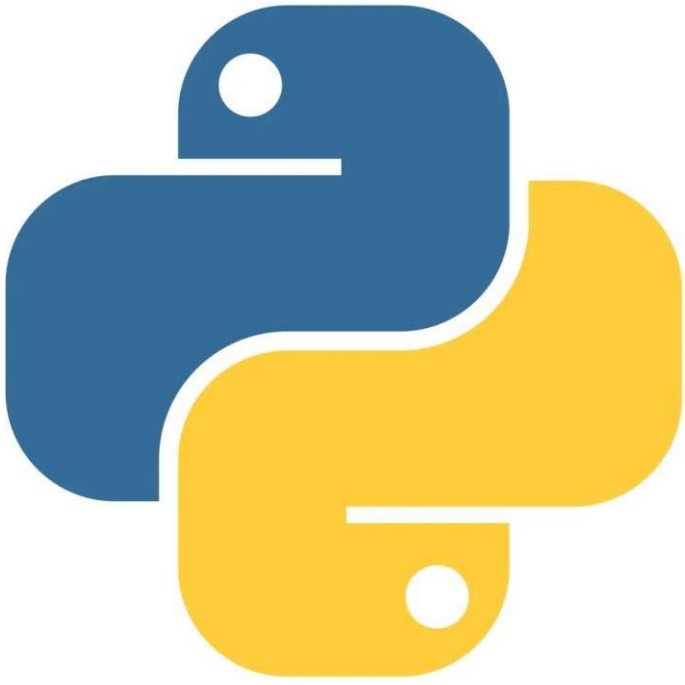
1- Créez une nouvelle classe Commande avec les attributs id_commande,date,et Machines(une liste d'instance de Machine) et prix_total_commande en argument.

2- Implimentez des méthodes getters et setters pour accéder et modifier l'attribut prix_total_commande de maniére sécurisée

3- Modifier la classe Commande dans le but d'afficher la moyenne des prix total de deux commandes par exemple: {Commande C1,Commande C2 -->print(C1+C2),afficher la moyenne des prix total des deux commandes C1 et C2}

4- Implimentez une méthode de classe chère_Commande() qui affiche les détails dee la commande qui à le plus grand prix_total_commande

Correction :



Enonce :

```
class Commande:     TVA = 0.20     def
_init_(self, id_commande, date, machines):
        self.id_commande = id_commande         self.date = date
self.machines = machines          self._prix_total_commande =
self.calculer_prix_total_commande()      def
calculer_prix_total_commande(self):          prix_total =
sum(machine.prix_machine for machine in self.machines)          return
prix_total * (1 + self.TVA)     @property     def
prix_total_commande(self):
        return self._prix_total_commande
    @prix_total_commande.setter      def
prix_total_commande(self, new_price):
        raise AttributeError("Le prix total de la commande ne peut pas être
modifié directement.")     @classmethod     def moyenne_prix_total(cls, commande1,
commande2):
        return (commande1.prix_total_commande + commande2.prix_total_commande) / 2
    @classmethod     def
chere_Commande(cls, *commandes):
        chere_commande = max(commandes, key=lambda x:
x.prix_total_commande)          print("Détails de la commande la plus chère
:")          print(f"ID Commande: {chere_commande.id_commande}")
print(f"Date: {chere_commande.date}")          print("Machines:")
for machine in chere_commande.machines:
            print(f" - {machine.nom_machine}: {machine.prix_machine}")
print(f"Prix total: {chere_commande.prix_total_commande}")
```

```
 # Méthode spéciale pour l'addition de deux
commandes     def _add_(self, other):         if not
isinstance(other, Commande):
            raise TypeError("L'addition ne peut être effectuée qu'avec une autre instance
de Commande.")         return Commande(-1, "", []) # Nous renvoyons une nouvelle commande
avec des valeurs vides pour l'exemple

# Exemple d'utilisation
if_name_ == "_main_":
        # Création de quelques machines
machine1 = Machine(1, "Machine 1", 1000)
machine2 = Machine(2, "Machine 2", 1500)
machine3 = Machine(3, "Machine 3", 2000)

        # Création de deux commandes          commande1 =
Commande(1, "2024-04-02", [machine1, machine2])          commande2
= Commande(2, "2024-04-03", [machine2, machine3])
        # Affichage de la moyenne des prix totaux          moyenne_prix =
 Commande.moyenne_prix_total(commande1, commande2)          print(f"Moyenne des
prix totaux des commandes 1 et 2: {moyenne_prix}")
        # Affichage de la commande la plus chère
        Commande.chere_Commande(commande1, commande2)
# Addition de deux commandes (pour l'exemple)
try:
            resultat_addition = commande1 + commande2
print("Résultat de l'addition de commande 1 et commande 2 :",
resultat_addition)          except TypeError as e:print(e)
```

contrôle v1 :

Enonce :

Etude de cas:

Partie1:

1- Definissez une classe abstraite Produit avec les attributs id_p,nomProduits et prix, incluant deux méthodes abstraites Afficher_details() et modifier_prix() et ajoutez un constructeur pour initialiser les attributs.

2- implimenez une méthode statique Somme_croisé qui prend en arguments une liste, et qui renvoi la somme croisée des éléments de cette liste.

Exemple: liste=[4,7,0,2,3,10,15]

Somme croisée= 4-7+0-2+3-10+15=3

Correction :

```
from abc import ABC, abstractclassmethod class
Produit(ABC):     def __init__(self, id_p,
nomProduit, prix):self.id_p = id_p
        self.nomProduit =
nomProduit          self.prix = prix
@abstractclassmethod      def
Aficher_detail(self):          pass
    @abstractclassmethod
def modifier_prix(self):
        pass
@staticmethod     def
somme_croisee(liste):
        somme=0         for i in
range(len(liste)):              if
i%2 ==0:
                somme+=liste[i]             else:
somme -=liste[i]         return somme l = [4, 7, 0, 2,
3, 10, 15] print(f"la somme croise est:
{Produit.somme_croisee(l)}")
```

Enonce :

Partie2:

1-Créez une classe dérivée Aliment qui n'est pas abstraite et qui hérite de la classe

Produit avec les attributs date_peremption,quantité et prix_total qui bénéficie d'une réduction de 10% lorsque la quantité d'aliments dépasse 10, et ajoutez un constructeur pour initialiser les attributs sans passer prix_total en argument

2-Implimentez une méthode de classe pour afficher le nombre total des aliments

Correction :

```
class Aliment(Produit):
    nbr_aliment = 0     def _init_(self, id_p, nomProduit, prix,
date_peremption, quantite):         super()._init_(id_p, nomProduit,
prix)         self.date_peremption = date_peremption
self.quantite = quantite
        self.prix_total = self.prix * self.quantite         if
self.quantite > 10:             self.prix_total = self.prix_total -
(self.prix_total*(10/100))          Aliment.nbr_aliment += 1     def
Aficher_detail(self):          return f"id_p: {self.id_p}\nnomProduit:
{self.nomProduit}\nprix:
{self.prix}\ndate_peremption: {self.date_peremption}\nquantite:
{self.quantite}\nprixtotal: {self.prixtotal}"
         def modifier_prix(self,
nouveauPrix):          self.prix =
nouveauPrix
    @classmethod     def
nombreAliment(cls):
        return f"le nombre total des aliment est: {cls.nbr_aliment}"
```

Partie3:

1-Créez une nouvelle classe Panier avec les attributs id_panier,Aliments(une liste d'instance d'Aliment) et prix_total_panier en argument.

2-Implimentez des méthodes getters et setters pour accéder et modifier l'attribut prix_total_panier de maniére sécurisée

3-Modifier la classe Panier dans le but d'afficher la moyenne des prix total de deux panier par exemple: {Panier P1,Panier P2 -->print(P1+P2),afficher la moyenne des prix total des deux panier P1 et P2}

4-Implimentez une méthode de classe chère_Panier() qui affiche les détails du panier qui à le plus grand prix_total_panier

Correction :

```
class panier:      mell_p_t=0
mell_panier=0      mell_aliment=0
nbr=0      def
_init_(self,id_panier,aliments):
self.id_panier=id_panier
        self.aliments=aliments if not None else
[]          self.prix_total_panier=0          ptv=0
panier.nbre+=1          for x in aliments:
ptv +=x.prix_total          self.p_t=ptv
if ptv > panier.mell_p_t:
panier.mell_p_t=ptv
panier.mell_aliment=aliments
panier.mell_panier=id_panier                def
calculer_p_t(self):          p_t=0          for
aliment in self.aliments:
p_t+=aliment.prix_total

self.prix_total_panier=p_t*(17/100)
def get_prix_total_panier(self):
return self.__prix_total_panier()       def
set_prix_total_panier(self,nv):
self._prix_total_panier=nv          def
_add_(self,other):
        return (self.prix_total_panier+other.prix_total_panier)/2
    @classmethod      def chere_panier(cls):       print
("le mielleur panier est:",{cls.mell_panier})
cls.mell_aliment.afficher_detail()       for aliment in
cls.mell_aliment:
        aliment.afficher_detail()
```