SERIE DES EXERCICES POO 3

EXERCICE 1 : Gestion D'un Parc Informatique

On souhaite développer une application orientée objet à l'instar du système conçu pour la société de fabrication des produits informatiques. L'application contiendra plusieurs classes dont tous les attributs doivent être **privés** (ou **protégés**), ce qui implique le besoin de coder des **accesseurs** dans toutes les classes. Les accesseurs des attributs affectées automatiquement doivent être en **lecture seule**. Chaque classe doit avoir un **constructeur d'initialisation**, et une méthode "**ToString()**".

On suppose que les classes suivantes sont déjà créées :

Classe "Développeur" : Ayant comme attributs un identifiant pour la société de développement (entier positif qui s'incrémente automatiquement), sa raison sociale, son adresse, son numéro de téléphone, son adresse e-mail, et le lien de son site web.

```
public class Développeur
{
   private int IdDéveloppeur;
   private string RaisonSocialeDéveloppeur;
   private string AdresseDéveloppeur;
   private string TélDéveloppeur;
   private string EMailDéveloppeur;
   private string SiteWebDéveloppeur;
                private static int auto;
}
```

Classe "Constructeur" : Ayant comme attributs un identifiant pour la société de fabrication des composants (entier positif qui s'incrémente automatiquement), sa raison sociale, son adresse, son numéro de téléphone, son adresse e-mail, et le lien de son site web**,** et un 2ème constructeur avec un paramètre supplémentaire: **constructeur**, qui exprime la société qui regroupe la filiale en instance.

```
public class Constructeur
{
   private int IdConstructeur;
   private string RaisonSocialeConstructeur;
   private string AdresseConstructeur;
   private string TélConstructeur;
   private string EMailConstructeur;
```

```
    private string SiteWebConstructeur;
    private Constructeur constructeur;


    private static int auto;
}
```

Classe "Système" : Ayant comme attributs la référence du système d'exploitation, la désignation du système, le prix du système, et le développeur du système.

```
public class Système
{
    private string RéfSystème;
    private string DésignationSystème;
    private float PrixSystème;
    private Développeur développeur;
}
```

Classe "Composant" : Ayant comme attributs la référence du composant, la désignation du composant, le prix du composant, et le constructeur du composant.

```
public class Composant
{
    private string RéfComposant;
    private string DésignationComposant;
    private float PrixComposant;
    private Constructeur constructeur;
}
```

Classe "Intégration" : Ayant comme attributs le composant intégré, et la quantité à intégrer de ce composant dans un produit.

```
public class Intégration
{
    private Composant ComposantIntégré;
    private int QuantitéIntegrée;
}
```

Vous devez répondre aux questions suivantes :

Classe "Produit"

1- Créer une classe "Produit" ayant comme attributs la référence du produit, la désignation du produit, une liste des composants**intégrés** dans le produit, et le prix du produit. **(4 pts)**
2- Ajouter une méthode "**IntégrerComposant(composant_intégré)**" permettant d'intégrer un nouveau composant à la liste des composants intégrés dans le produit. si le composant intégré est déjà existant (composant ayant la même référence) affichez message « Produit existe déjà ». **(2,5 pts)**
3- Ajouter une méthode "**EnleverComposant(référence_composant)**" permettant d'enlever un composant de la liste des composants intégrés en fournissant sa **référence**. si la référence du composant est inexistante. Affichez umessage « Produit inexistant »**(2,5 pts)**
4- Ajouter une méthode "**CalculerPrix()**" permettant de calculer le prix d'un produit obtenu par la somme des montants des composants intégrés (en tenant compte de leurs quantités) plus une majoration de 20%. **(4 pts)**

Classe "Pilote"

1- Créer une classe "Pilote" ayant comme attributs la référence du pilote, la désignation du pilote, la liste des composants compatibles, et la liste des systèmes d'exploitation compatibles. **(4 pts)**
2- Ajouter une méthode "**AjouterComposant(composant)**" permettant d'ajouter un nouveau composant à la liste des composants compatibles avec le pilote. si le composant est déjà existant (composant ayant la même référence) affichez message « Composant existe déjà ». **(2,5 pts)**
3- Ajouter une méthode "**AjouterSystème(système)**" permettant d'ajouter un nouveau système à la liste des systèmes d'exploitation compatibles avec le pilote. si la référence du système d'exploitation est déjà existante affichez message « Système existe déjà ». **(2,5 pts)**

Classe "Produit_Pro"

Un produit pro "**est un produit**" destiné aux professionnels, se caractérisant par sa performance, sa résistance et sa durabilité. Pour de tels produits, la société informatique offre des garanties allant de 2 à 5 ans.

1- Créer une classe "Produit_Pro" ayant comme attributs la garantie qui doit être entre 2 et 5 ans, et la date de début de garantie. **(4 pts)**
2- Redéfinir la méthode "**CalculerPrix()**" en ajoutant au prix du produit le montant de garantie égale à 750 MAD par année de garantie. **(3 pts)**
3- Ajouter une méthode "**Fin_garantie()**" qui retourne la date de fin de la garantie d'un produit pro. **(3 pts)**
4- Ajouter une méthode "**Etendre_garantie(nombre_années)**" permettant d'augmenter le nombre d'années de garantie pour un produit ayant moins de 5 ans de garantie, vers une garantie inférieure ou égale à 5 ans (en tenant compte que si la garantie totale risque d'excéder 5 ans). **(3 pts)**

Exercice 2 : Gestion Magasin

On souhaite informatiser la gestion des ventes au sein d'un magasin. On considère alors qu'un article est caractérisé par son numéro de série, son prix hors taxe, sa quantité en stock, et la quantité minimale

1) a) Ecrire la classe **« Article »**. **(2 pts)**
   Ajouter à cette classe un constructeur permettant d'instancier des objets de la classe « Article » dont on précisera le numéro de série, le prix hors taxe, la quantité en stock, la quantité minimale et un constructeur sans paramètres. **(2 pts)**

   b) Réécrire la méthode ***ToString()*** pour afficher les caractéristiques d'un article. **(2 pts)**
   c) Ajouter à la classe Article les méthodes suivantes:
   - **S'approvisionner (int qte) :** qui permet d'approvisionner le stock par une quantité donnée. **(2pts)**
   - **Achat (int qte)** permet de traiter un achat d'un article par un client. Une opération d'achat aura pour effet de déduire la quantité achetée du stock. Si la quantité qui reste est inférieure à la quantité minimale on avise par un message. **(2 pts)**

2) Un habit est un article qui a une taille et une couleur :
   a) Ecrire la classe **« Habit »** héritant de la classe « **Article** ». **(2 pts)**
   b) Récrire le constructeur de cette classe afin d'initialiser, en plus, la couleur et la taille avec des valeurs passées en paramètre. . **(2 pts)**
   c) Réécrire la méthode ***toString()*** pour afficher les caractéristiques de l'habit. . **(2 pts)**

3) Un électroménager est un article qui a un poids et une durée de garantie.
   a) Ecrire la classe **« Electroménager »** héritant de la classe « **Article** ». . **(2 pts)**
   b) Récrire le constructeur de cette classe pour définir, en plus, le poids et la durée de garantie en mois. **(2 pts)**
   c) Ajouter la méthode **datefinGarantie ()** : retourne la date de fin de la garantie à partir de la date actuelle. **(2 pts)**
   d) Réécrire la méthode ***toString()*** donnant les caractéristiques d'un électroménager et la date de fin de sa garantie à partir de la date courante. **(2 pts)**

4) **Classe Program :** Tester ces trois classes dans un programme principal.

   a) Créer un article de type habit **(1 pt)**
   b) Approvisionner le stock de cet article et l'afficher. **(1,5 pts)**
   c) Créer un article de type électroménager **(1 pt)**
   d) Effectuer un achat de cet article. **(1 pt)**
   e) Afficher la date fin de garantie de cet article. **(1 pt)**
   f) Afficher cet article. **(0,5 pt)**

Exercice 3 : Gestion des appareils Electriques :

On désire réaliser une application pour une société qui fabrique et commercialise des appareils électriques :

Classe Appareil : On considère qu'un appareil est caractérisé par une référence, une puissance (exprimée en watt), un poids et un prix.

2)

a. Ecrire la classe **« AppElectrique »** permettant de modéliser ces objets. **(2 pts)**

Ajouter à cette classe un constructeur permettant d'instancier des objets de la classe « ***AppElectrique*** » dont on précisera la référence, la puissance, le poids, et un constructeur sans paramètres. **(2 pts)**.

b. Réécrire la méthode ***ToString()*** pour afficher les caractéristiques de l'appareil. **(2 pts)**
c. Ecrire la méthode **ClasseEneergetique() :** qui permet d'afficher la classe énergétique de l'appareil selon sa puissance : (**2 pts)**
    - Si la puissance est inférieure strictement à 300 watt c'est la classe A
    - Si la puissance est comprise entre 300 watt et 1000 watt c'est la classe B
    - Si la puissance est supérieure à 1000 watt c'est la classe C

3) **Classe Television :** une télévision est un appareil qui possède un type d'écran (LCD, LED …), et une fréquence(en hertz) :
    a) Ecrire une classe **« Television »** héritant de la classe « **AppElectrique** ». **(2 pts)**
    b) Récrire le constructeur de cette classe afin d'initialiser, en plus, le type d'écran et la fréquence avec des valeurs passées en paramètre. **(2 pts)**
    c) Réécrire la méthode ***toString()*** affichant les caractéristiques de la télévision. **(2 pts)**
4) **Classe VeloElec :** un vélo électrique est un appareil qui a une autonomie et un kilométrage.
    a) Ecrire une classe **VeloElec** héritant de la classe « ***AppElectrique*** ». **(2 pts)**
    b) Récrire le constructeur de cette classe pour définir, en plus, l'autonomie (exprimée en Km) et le kilométrage. **(2 pts)**
    c) Ajouter les méthodes suivantes :
        i) **Rouler (float distance) :** permettant au vélo d'avancer et retournant le nouveau kilométrage. **(2 pts)**
        ii) **Charger** (**int nbrminute**) : permettant de charger la batterie et renvoyer la nouvelle valeur de l'autonomie (on suppose qu'une heure de charge donne 10 km en autonomie). **(2 pts)**

d) Réécrire la méthode ***toString()*** donnant les caractéristiques d'un vélo électrique. **(2 pts)**

5) **Classe Program :** Tester ces trois classes dans un programme principal.

   a) Créer un appareil électrique de type télévision **(1 pt)**

   b) Afficher cet article et afficher sa classe énergétique. **(1,5 pts)**

   c) Créer un article de type vélo électrique **(1 pt)**

   d) Faites rouler ce vélo **(1 pt)**

   e) Charger ce vélo **(1 pt)**

   f) Afficher le vélo **(0,5 pt)**

EXERCICE 4 : Gestion des Factures :

On souhaite créer une application pour la gestion des factures des clients dans une entreprise.

Nous aurons besoin des classes suivantes :

1. La classe **Client** : **(2 Points)**
   a. Définir la liste des attributs suivants : Numéro de client (auto-incrément), nom, prénom.
   b. Pour chaque attribut, Ajouter des propriétés de type lecture/écriture.
   c. Ajouter un constructeur par défaut et un constructeur d'initialisation de tous les attributs.
   d. Redéfinir la méthode ToString et la méthode Equals (deux clients sont égaux s'il possède le même Numéro de client)
2. La classe **Facture : (2,5 Points)**
   a. Définir la liste des attributs suivant : Numéro de Facture, Montant de facture, Client, Date de facture.
   b. Pour chaque attribut, Ajouter des propriétés de type lecture/écriture sachant que :
      i. La date de facture doit être inférieur à la date du jour.
      ii. Le montant de la facture doit être supérieur ou égale à 0.
   c. Ajouter un constructeur par défaut et un constructeur d'initialisation de tous les attributs.
   d. Ajouter la méthode polymorphe **GetRemise()**, qui permet de calculer la remise qui peut être appliquée a la facture.
   e. Ajouter une méthode **GetAncienneteFacture(),** qui retourne l'ancienneté de la facture qui se calcule par la différence par année entre la date du jour et la date de la facture.
3. La classe **FactureLocale** qui hérite de la classe Facture : **(2,5 Points)**
   a. Définir la liste des attributs suivants : ville(string).
   b. Ajouter une propriété de type lecture/Ecriture pour l'attribut ville.
   c. Ajouter un constructeur par défaut et un constructeur d'initialisation de tous les attributs.
   d. Redéfinir la méthode **GetRemise()**, qui retourne la valeur suivante :

si Montant de la facture < 200 alors Remise = 0

sinon Remise = 10%

4. La classe **FactureCourant** qui hérite de la classe Facture : **(2,5 Points)**
    a. Définir la liste des attributs suivants : Pays(String).
    b. Ajouter une propriété de type lecture/Ecriture pour l'attribut Pays.
    c. Ajouter un constructeur par défaut et un constructeur d'initialisation de tous les attributs.
    d. Redéfinir la méthode **GetRemise()**, qui retourne la valeur suivante :

si Montant de la facture < 100 alors Remise = 0

sinon Remise = 15%

5. Ecrire la classe **ListeFacture** qui se caractérise par une liste de Facture **(10,5 points)** :
    a. Définir les attributs suivants : Liste des Factures.
    b. Ajouter un constructeur par défaut et un constructeur d'initialisation.
    c. Ajouter la méthode **ajouter(Facture c)** qui permet d'ajouter la Facture passée en paramètre dans la liste. si la Facture est déjà présente dans la liste avec le même numéro de Facture affichez message.
    d. Ajouter la méthode **supprimer(int NumFacture)** qui permet de supprimer la Facture dont le numéro est passé en paramètre.
    e. Ajouter la méthode **ListeFacture(int NumClient)** qui retourne la liste des Factures d'un client dont le numéro est passé en paramètre.
    f. Ajouter la méthode **ListeFactureLocale()** qui retourne la liste des Factures de type Facture Locale.
    g. Ajouter la méthode **ListeFactureEtranger()** qui retourne la liste des Factures de type Facture Etranger.

Exercice 5 : Gestion des Salariés

La classe "Salarié" possédera 5 propriétés de visibilité privée :

matricule	matricule	Int
nom	nom	String
Prénom	prenom	String
salaire	salaire	Double
Taux Charges Sociales	tauxCS	Double

1. Créez la classe Salarié avec ces propriétés.(visibilité privée)
2. Implémentez deux constructeurs (par défaut, d'initialisation)
3. Implémenter les getters et les setters.
4. Implémentez la méthode AfficherSalarie() qui permet d'afficher les informations du salarie.
5. Implémentez la méthode CalculerSalaireNet().

    Elle doit calculer le salaire net d'un employé qui équivaut à :

salaire – (salaire*TauxCS) et retournera la valeur calculée.

6. Implémentez le constructeur par copie
7. Créez une classe **Commercial** en dérivant la classe **salarie.** Cette classe aura 2 propriétés supplémentaires pour calculer la commission :
   - **chiffre d'affaire** chiffreAffaire Double
   - **commission en %** pourcentageCommission Double
8. Créez les deux constructeurs standards de la classe **Commercial** (par défaut et initialisation). Ne pas oublier d'appeler les constructeurs équivalents de la classe de base.
9. Créez les méthodes d'accès aux propriétés supplémentaires.
10. Surchargez la méthode **CalculerSalaireNet** pour calculer le salaire réel (fixe + commission).
11. Changez la visibilité des propriétés de la classe mère en Protected.
12. Ajoutez à la classe **salarie** les méthodes **Equals** et **Tostring.** La règle d'égalité pour la classe **salarie** peut s'énoncer de la façon suivante : deux salariés sont égaux s'ils ont le même numéro de matricule. **Tostring** doit renvoyer toutes les propriétés Matricule et Nom séparées par des virgules.
13. Surchargez les méthodes de la classe de base pour lesquelles on jugera nécessaire de faire ainsi.(ToString et Equals)(Deux commerciaux sont égaux s'ils ont le même matricule et le même chiffre d'affaire).
14. Ecrire un programme pour tester la définition de votre classe et son bon fonctionnement. Pour ce faire :
    - Déclarez une variable de type Salarie en l'instanciant respectivement par le constructeur par défaut et en permettant à l'utilisateur de saisir les propriétés.
    - Déclarez un deuxième objet de type Salarié en l'instanciant avec le constructeur d'initialisation. Initialisez ses propriétés en choisissant des valeurs significatives pour les salaires et le taux de charge sociale afin de tester correctement le calcul réalisé par la méthode CalculerSalaireNet. Pour des salaires respectifs de 5 000 et 10 000 dirhams un taux de charges sociales de 25 %, vous devez trouver 3 750 et 7 500.
    - Afficher les informations du premier salarié en utilisant les getters puis afficher celles du deuxième avec la méthode AfficherSalarie().

- Affichez ensuite le salaire net du deuxième salarié.
- Créez un troisième salarié en copiant les informations du deuxième puis changer son matricule, nom et prénom à travers les méthodes Set en affichant à la fin ses informations.
- Déclarez une variable Commercial puis instanciez-la en utilisant le constructeur d'initialisation.
- Affichez les informations du Commerical en utilisant toString().
- Affichez son salaire net.
- Déclarez et instanciez une deuxième variable (cette fois en utilisant le constructeur par défaut) puis testez l'égalité de ces deux variables