M11 : Programmation Orientée Objet

Exercices TP

Exercice 1 :

1. Écrire une classe Complexes permettant de représenter des nombres complexes. Un nombre complexe Z comporte une partie réelle et une partie imaginaire : Z = PartieRéelle + PartieImaginaire * i
2. Définir à l'aide des propriétés les méthodes d'accès aux attributs de la classe.
3. Définir un **constructeur** par défaut permettant d'initialiser les deux parties du nombre à 0.
4. Définir un **constructeur** d'initialisation pour la classe.
5. Ajouter les méthodes suivantes :

- **Plus(Complexe)** : Elle permet de retourner le nombre complexe obtenu en ajoutant au nombre en cours un nombre complexe passé en argument.
- **Moins(Complexe) :** Elle permet de retourner le nombre complexe obtenu en soustrayant au nombre en cours un nombre complexe passé en argument.
- **Afficher ( ) :** Elle donne une représentation d'un nombre complexe comme suit : a+b*i.

6. Écrire un programme permettant de tester la classe Complexe.

Exercice 2 :

1. Définir une classe Rectangle ayant les attributs suivants : longueur et largeur.
2. Définir à l'aide des propriétés les méthodes d'accès aux attributs de la classe.
3. Ajouter un constructeur d'initialisation.
4. Ajouter les méthodes suivantes :

- Périmètre ( ) : retourne le périmètre du rectangle.
- Aire( ) : retourne l'aire du rectangle.
- EstCarre( ) : vérifie si le rectangle est un carré.
- AfficherRectangle( ) : expose les caractéristiques d'un rectangle comme suit : Longueur : […] - Largeur : […] - Périmètre : […] - Aire : […] - Il s'agit d'un carré / Il ne s'agit pas d'un carré

Exercice 3 :

1. Définir une classe Client avec les attributs suivants : CIN, Nom, Prénom, Tél.
2. Définir à l'aide des propriétés les méthodes d'accès aux différents attributs de la classe.
3. Définir un constructeur permettant d'initialiser tous les attributs.
4. Définir un constructeur permettant d'initialiser le CIN, le nom et le prénom.
5. Définir la méthode Afficher ( ) permettant d'afficher les informations du Client en cours.
6. Créer Une classe Compte caractérisée par son solde et un code qui est incrémenté lors de sa création ainsi que son propriétaire qui représente un client.
7. Définir à l'aide des propriétés les méthodes d'accès aux différents attributs de la classe (le numéro de compte et le solde sont en lecture seule)
8. Définir un constructeur permettant de créer un compte en indiquant son propriétaire.
9. Ajouter à la classe Compte les méthodes suivantes :

- Une méthode permettant de Crediter() le compte, prenant une somme en paramètre.
- Une méthode permettant de Crediter() le compte, prenant une somme et un compte en paramètres, créditant le compte et débitant le compte passé en paramètres.
- Une méthode permettant de Debiter() le compte, prenant une somme en paramètre
- Une méthode permettant de Débiter() le compte, prenant une somme et un compte bancaire en paramètres, débitant le compte et créditant le compte passé en paramètres
- Une méthode qui permet d'afficher le résumé d'un compte.
- Une méthode qui permet d'afficher le nombre des comptes crée.

10. Créer un programme de test pour la classe Compte. somme

Exercice 4 :

Un cercle est défini par :

- point qui représente son centre
- Son rayon r

On peut créer un cercle en précisant son centre et son rayon.

Dans ce problème, nous allons commencer tout d'abord par définir la classe Point définie par :

- Les **attributs**: x et y de type réel
- Un **constructeur** qui permet de définir les valeurs de x et de y.
- Une méthode **Afficher ()** qui affiche une chaîne de caractères POINT(x,y).

Les opérations que l'on souhaite exécuter sur un cercle sont :

- **getPerimetre()**: retourne le périmètre du cercle.
- **getSurface()**: retourne la surface du cercle.
- **appartient (Point p)**: retourne si le point p appartient ou non au cercle.
- **Afficher ()**: Affiche une chaîne de caractères de type CERCLE(x,y,R)

Exercice 5 :

1. Définir une classe **Livre** avec les attributs suivants : Titre, Auteur (Nom complet), Prix.
2. Définir à l'aide des propriétés les méthodes d'accès aux différents attributs de la classe.
3. Définir un **constructeur** permettant d'initialiser les attributs de la méthode par des valeurs saisies par l'utilisateur.
4. Définir la méthode **Afficher ( )** permettant d'afficher les informations du livre en cours.
5. Écrire un programme testant la classe Livre.

Exercice 6 :

Créer une Classe nommée **MyDateTime,** qui se compose de deux parties : La partie date et la partie heure. (A vous de choisir les attributs correspondants, ne pas oublier les constructeurs !)

Ajouter par la suite les méthodes **AjouterDateTime(int annees, int mois, int jours)** et **Afficher()**.

Exercice 7 :

Ecrivez une classe Bâtiment avec les attributs suivants:

- adresse

La classe Bâtiment doit disposer des constructeurs suivants:

- Batiment(),
- Batiment (adresse).

La classe Bâtiment doit contenir des accesseurs et mutateurs (ou propriétés) pour les différents attributs. La classe Bâtiment doit contenir une méthode ***ToString ()*** donnant une représentation du Bâtiment.

Ecrivez une classe Maison héritant de Bâtiment avec les attributs suivants:

- NbPieces: Le nombre de pièces de la maison.

La classe Maison doit disposer des constructeurs suivants:

- Maison(),
- Maison(adresse, nbPieces).

La classe Maison doit contenir des accesseurs et mutateurs (ou des propriétés) pour les différents attributs. La classe Maison doit contenir une méthode ***ToString ()*** donnant une représentation de la Maison.

Ecrivez aussi un programme afin de tester ces deux classes.

Exercice 8 :

- Un compte bancaire possède à tout moment une donnée : son solde. Ce solde peut être positif (compte créditeur) ou négatif (compte débiteur).
- Chaque compte est caractérisé par un code incrémenté automatiquement.
- le code et le solde d'un compte sont accessibles en lecture seulement.
- A sa création, un compte bancaire a un solde nul et un code incrémenté.
- Il est aussi possible de créer un compte en précisant son solde initial.
- Utiliser son compte consiste à pouvoir y faire des dépôts et des retraits. Pour ces deux opérations, il faut connaître le montant de l'opération.
- L'utilisateur peut aussi consulter le solde de son compte par la méthode ToString().
- Un compte Epargne est un compte bancaire qui possède en plus un champ « Taux Intérêt=6 » et une méthode calculIntérêt() qui permet de mettre à jour le solde en tenant compte des intérêts.
- Un ComptePayant est un compte bancaire pour lequel chaque opération de retrait et de versement est payante et vaut 5 DH.

Travail à faire :

1. Définir la classe Compte.

2. Définir la classe CompteEpargne.
3. Définir la classe ComptePayant.
4. Créer un programme permettant de tester ces classes avec les actions suivantes:

- Créer une instance de la classe Compte, une autre de la classe CompteEpargne et une instance de la classe ComptePayant
- Faire appel à la méthode déposer() de chaque instance pour déposer une somme quelconque dans ces comptes.
- Faire appel à la méthode retirer() de chaque instance pour retirer une somme quelconque de ces comptes.
- Faire appel à la méthode calculInterêt() du compte Epargne.
- Afficher le solde des 3 comptes.

Exercice 9 :

Définir une classe **Vecteurs2D** caractérisée par l'abscisse **X** et l'ordonné **Y**, ainsi qu'un attribut qui renseigne sur le nombre de vecteurs créés lors de l'exécution du programme.

Définir les constructeurs (par défaut, d'initialisation et de recopie), et les accesseurs aux différents attributs.

Définir les méthodes **ToString** et **Equals**, la première retourne une chaine de caractères représentant l'abscisse et l'ordonné du vecteur comme suit : X = 1.5 - Y=2, la deuxième retourne **True** lorsque l'abscisse et l'ordonné des deux vecteurs sont égaux.

Définir une méthode Norme qui retourne la norme d'un vecteur, cette méthode peut être redéfinie dans les classes dérivées.

Définir une classe **Vecteurs3D** dérivée de la classe **Vecteur2D** qui contient, en plus des propriétés de la classe de base, la propriété **Z** symbolisant la troisième dimension.

Définir les constructeurs (par défaut, d'initialisation et de recopie) de cette classe, ainsi que les accesseurs des propriétés.

Redéfinir les méthodes **ToString** et **Equals** pour intégrer la troisième propriété.

Redéfinir la méthode Norme pour qu'elle retourne la norme d'un vecteur dans l'espace.

Définir un programme de test permettant de créer deux objets de type **Vecteurs2D** et deux autres de type **Vecteurs3D**. Afficher les informations de tous les objets, et tester les méthodes Equals et Norme. Afficher le nombre d'objet créés.

Exercice 10 :

Soit les classes suivantes :

Une classe **Personne** qui comporte trois champs privés, nom, prénom et date de naissance. Cette classe comporte un constructeur pour permettre d'initialiser les données. Elle comporte également une méthode polymorphe Afficher pour afficher les données de chaque personne.

Une classe **Employé** qui dérive de la classe Personne, avec en plus un champ *Salaire* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.

Une classe **Chef** qui dérive de la classe Employé, avec en plus un champ *Service* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.

Une classe **Directeur** qui dérive de la classe Chef, avec en plus un champ *Société* accompagné de sa propriété, un constructeur et la redéfinition de la méthode Afficher.

1. Ecrire les classe Personne, Employé, Chef et Directeur.
2. créez un programme de test qui comporte tableau de huit personnes avec cinq employés, deux chefs et un directeur (8 références de la classe Personne dans lesquelles ranger 5 instances de la classe Employé, 2 de la classe Chef et 1 de la classe Directeur).
   Affichez l'ensemble des éléments du tableau à l'aide de for.
   Affichez l'ensemble des éléments du tableau à l'aide de foreach.

Exercice 11 :

1) Créer la classe Fournisseur :

Cette classe possédera 3 propriétés :

Nom des propriétés	Définitions	Types
IdF	Identifiant du fournisseur	Integer
NomF	Nom du fournisseur	String
PrénomF	Prénom du fournisseur	String

2) Créer un constructeur par défaut

3) Créer un constructeur permettant d'initialiser tous les membres de la classe Fournisseur.

4) Créer la classe Auteur :

Cette classe possédera 3 propriétés :

Nom des propriétés	Définitions	Types
IdA	Identifiant de l'auteur	Integer
NomA	Nom de l'auteur	String
PrénomA	Prénom de l'auteur	String

5) Créer un constructeur par défaut

6) Créer un constructeur permettant d'initialiser tous les membres de la classe Auteur

7) Créer la classe Livre permettant de gérer des livres

Cette classe possédera 6 propriétés de visibilité privée :

Nom des propriétés	Définitions	Types
Titre	Titre	String
Annee	Année de parution	String
NPage	Nombre page	Integer
Prix	prix du livre	Integer
Fournisseur	Founisseur du livre	Classe Fournisseur
Auteur	Auteur du livre	Classe Auteur

8) Créer un constructeur permettant d'initialiser tous les membres de la classe Livre

9) Créer un constructeur de recopie

10) Implémenter une méthode statique à qui on donne comme paramètre un livre et affiche les informations de son fournisseur

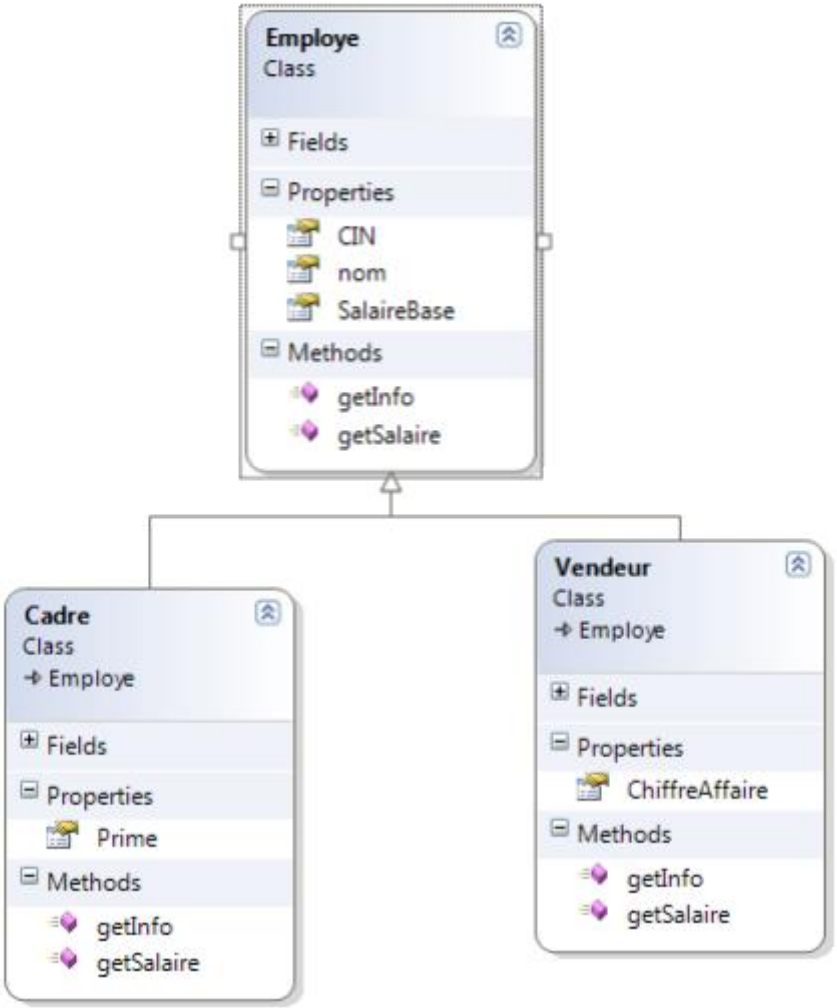
11) Implémenter une méthode statique à qui on donne comme paramètre un livre et affiche les informations de son auteur

12) Ajouter un programme de test où il faut créer différentes objets pour faire l'ensemble de teste

Exercice 12 :

Réalisation d'une application gestion Employés

Soit les classes :



Le salaire se calcule selon les formules suivantes :

Employé : Salaire=SB+SB*10%.

Vendeur : Salaire=SB+CA*5%.

Cadre : Salaire=SB–+prime.

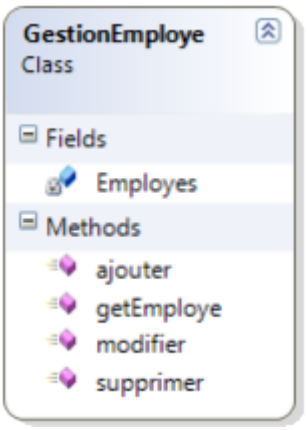
Questions :

1) créer les classes Employé, Vendeur, Cadre.

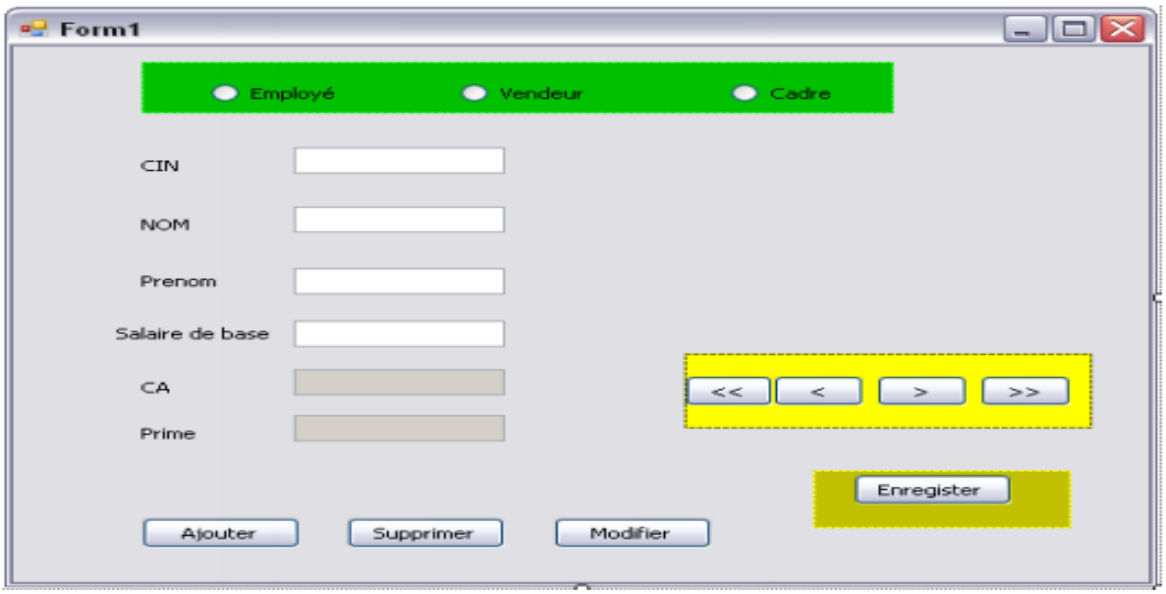
2) Ajouter les constructeurs d'initialisation sur les trois classes.

3) Surcharger avec polymorphisme les méthodes GetInfo () et GetSalaire ().

4) Créer la classe Gestion Employé.



Créer un formulaire qui permet de gérer les employés.



Les boutons radio sont utilisés au moment de l'ajout d'un employé ils permettent d'afficher ou non les zones (chiffre d'affaire CA, Prime) selon le type d'employé (employé, vendeur ou cadre)

Pour vérifier le type d'un objet en utilise le mot clé **is**

Exercice 13 :

Une Voiture est caractérisée par sa marque, couleur, Matricule et Nombre de chevaux

1. Créer la classe voiture

2. Ajouter à la classe le constructeur sans paramètres

3. Ajouter à la classe le constructeur qui permet d'initialiser la marque, la couleur, la matricule et le nombre de chevaux

4. Ajouter à la classe le constructeur qui permet d'initialiser seulement la matricule

5. Encapsuler tous les attributs (les rendre private puis ajouter les accesseurs et modificateurs), l'attribut Matricule ne possède pas sa méthode modificateur en effet on ne peut pas modifier la matricule d'une voiture

6. Ajouter la méthode présentation qui permet d'afficher les informations d'une voiture

7. Sur la classe program.cs créer un tableau qui peut contenir cinq voitures puis remplir le tableau

8. Écrire le code qui permet d'afficher les informations de toutes les voitures du tableau

9. Écrire le code qui permet d'augmenter le nombre des chevaux de 1 cheval toutes les voitures du tableau

10. Afficher les informations des voitures après modification

11. Écrire le code qui permet d'inviter l'utilisateur pour saisir la matricule puis cherche et affiche les informations de la voiture relative, s'il n'existe pas il affiche le message « matricule inexistante »

Exercice 14 :

Un parc auto se compose des voitures et des camions qui ont des caractéristiques communes regroupées dans la classe Véhicule.

- Chaque véhicule est caractérisé par son matricule, l'année de son modèle, son prix.
- Lors de la création d'un véhicule, son matricule est incrémenté selon le nombre de véhicules créés.
- Tous les attributs de la classe véhicule sont supposés privés. ce qui oblige la création des accesseurs (get…) et des mutateurs (set….) ou les propriétés.
- La classe Véhicule possède également deux méthodes abstraites démarrer() et accélérer() qui seront définies dans les classes dérivées et qui afficheront des messages personnalisés.
- La méthode ToString() de la classe Véhicule retourne une chaîne de caractères qui contient les valeurs du matricule, de l'année du modèle et du prix.
- Les classes Voiture et Camion étendent la classe Véhicule en définissant concrètement les méthodes accélérer() et démarrer() en affichant des messages personnalisés.

Travail à faire:

- Créer la classe abstraite Véhicule.
- Créer les classes Camion et Voiture.
- Créer une classe Test qui permet de tester la classe Voiture et la classe Camion

Exercice 15 :

Soit la classe abstraite Employé caractérisée par attributs suivants :

- Matricule
- Nom
- Prénom
- Date de naissance

La classe Employé doit disposer des méthodes suivantes :

- un constructeur d'initialisation
- des propriétés pour les différents attributs
- la méthode ToString
- une méthode abstraite GetSalaire.

Un ouvrier est **un employé** qui se caractérise par sa date d'entrée à la société.

- Tous les ouvriers ont une valeur commune appelée SMIG=2500 DH
- L'ouvrier a un salaire mensuel qui est : Salaire = SMIG + (Ancienneté en année)*100.
- De plus, le salaire ne doit pas dépasser SMIG*2.

Un cadre est **un employé** qui se caractérise par un indice.

- Le chiffre d'affaire est commun entre les patrons.
- Le patron a un salaire annuel qui est égal à x% du chiffre d'affaire : Salaire = CA*pourcentage/100

Travail à faire:

1. Créer la classe abstraite Employé.
2. Créer la classe Ouvrier, la classe Cadre et la classe Patron qui héritent de la classe Employé, et prévoir les Constructeurs et la méthode ToString de chacune des 3 classes.
3. Implémenter la méthode GetSalaire() qui permet de calculer le salaire pour chacune des classes.

Exercice 16 :

Soit à développer une application pour la gestion d'un stock.

Un article est caractérisé par son numéro de référence, son nom, son prix de vente et une quantité en stock.

Le stock est représenté par une collection d'articles**.**

Travail à faire:

Créer la classe article contenant les éléments suivants :

- Les attributs/propriétés.
- Un constructeur d'initialisation.
- La méthode ToString().

Dans la classe Program créer :

Le stock sous forme d'une collection (Liste) d'articles de votre choix.

Un menu présentant les fonctionnalités suivantes :

1. Rechercher un article par référence.
2. Ajouter un article au stock en vérifiant l'unicité de la référence.
3. Supprimer un article par référence.
4. Modifier un article par référence.
5. Rechercher un article par nom.
6. Rechercher un article par intervalle de prix de vente.
7. Afficher tous les articles.
8. Quitter

Exercice 17 :

1. Écrire un programme dans lequel on demande à l'utilisateur de saisir un entier en gérant l'exception dans le cas où il ne saisit pas un entier correctement.
2. Écrire un programme dans lequel on demande à l'utilisateur de saisir un entier en gérant l'exception dans le cas où il ne saisit pas un entier correctement en lui demandant de refaire la saisie.
3. Écrire un programme dans lequel on demande à l'utilisateur de saisir sa date de naissance en gérant l'exception dans le cas où il ne saisit pas une date valide en lui demandant de refaire la saisie.
4. Ecrire un programme qui demande à l'utilisateur une date de départ et une date d'arrivée, générer une exception si la date d'arrivée et inférieure à la date de départ.
5. Créer une classe Élèves caractérisée par nom, âge et moyenne.

   - L'âge doit être entre 18 et 26 sinon l'exception InvalidAgeException (elle affiche le message "L'âge doit être entre 18 et 26") est générée.
   - La note doit être entre 0 et 20 sinon l'exception InvalidNoteException est générée (elle affiche le message "La note doit être entre 0 et 20").

   Définir les constructeurs de la classe, les accesseurs et les méthodes ToString.

Exercice 18 :

Soit à développer une application de gestion des adhérents qui sont inscrits dans une Médiathèque. Lorsqu'un adhérent est inscrit à la Médiathèque, on lui affecte automatiquement un numéro et on fixe sa cotisation. L'adhèrent qui le souhaite peut ne plus appartenir à la médiathèque, il démissionne.

1. Créer la classe Adhérent.
2. Ajouter à la classe Adhérent les méthodes :
    a. ToString () : affichage des attributs de la classe Adhérent sous forme de chaine de caractères.
    b. Modifie (Double cotisation) : modification de la cotisation.
3. Ajouter un constructeur par défaut qui permet de créer un objet Adhérent dont le nom est « anonyme ».
4. Ajouter un constructeur qui permet de crée un objet adhèrent en générant un numéro aléatoire.
5. Ecrire le code permettant de saisir un adhèrent et prévoir les cas d'exception.
6. Ajouter une méthode de modification d'un adhèrent.
7. Ajouter une méthode d'affichage et de recherche et d'affichage d'un adhèrent.
8. Ajouter une méthode de suppression d'un adhérent.
9. Ajouter une méthode d'affichage de tous les adhérents.

Exercice 19 :

Chaque enseignant de l'université effectue un certain nombre d'heures d'enseignement dans une année. Suivant le statut de l'enseignant, un certain nombre de ces heures peut être considéré comme complémentaire. Les heures complémentaires sont payées séparément à l'enseignant. Les volumes horaires sont exprimés en heures entières et le prix d'une heure complémentaire est de 100DH.

- Le nom et le nombre d'heures total d'un enseignant sont fixés à sa création, puis seul le nom peut être librement consulté (méthode getNom()).
- D'autre part on veut pouvoir librement consulter un enseignant sur son volume d'heures complémentaires (méthode getHc()) et sur la rétribution correspondante (méthode getRetribution()).

Il y a deux types d'enseignants :

- Les intervenants extérieurs : toutes les heures effectuées sont complémentaires.
- Les enseignants de la fac : seules les heures assurées au-delà d'une charge statutaire de 192h sont complémentaires.

Q1. Modéliser les enseignants : quelles sont les classes ? Ou sont implémentées les méthodes ? Lesquelles sont nouvelles, redéfinies ?
Q2. Ecrire les classes.
Q3. Comment modifier le modèle pour y introduire les étudiants de troisième cycle qui assurent des enseignements : toutes les heures effectuées sont complémentaires mais dans la limite de 96 heures.
Q4. Cela pose-t-il un problème de prendre en compte le fait que les étudiants, n'ayant pas d'employeur, voient leur rétribution diminuée de 18% ?
Q5. Tester le programme.