SERIE 2 D'EXERCICES POO

Exercice 1 : Gestion cabinet Médical

L'objet de cette application est la gestion informatisée du cabinet médical. Elle doit procéder à la gestion informatisée des dossiers médicaux des patients et elle s'occupe également de la gestion des rendez-vous et des visites.

1. **Classe Patient. (2,5 pts)**
   a. Définir une classe Patient dont les caractéristiques sont : Code patient (affecté de façon incrémentale par rapport au nombre de patients), Nom, Prénom, Date de naissance, Adresse, Tél, E-mail. (0.5 pt)
   b. Ecrire un constructeur à deux paramètres : Nom et prénom. (0.5 pt)
   c. Écrire un deuxième constructeur avec tous les paramètres. (0.5 pt)
   d. Écrire les accesseurs des champs et la méthode **toString()** qui renverra les informations d'un patient . (1 pt)
2. **Classe Visites. (2,5 pts)**
   a. Définir une classe Visites dont les caractéristiques sont Date visite, Heure visite, Code patient, Montant payé. (0.5 pt)
   b. Ecrire un constructeur à deux paramètres : Date visite et Heure visite. (0.5 pt)
   c. Écrire un deuxième constructeur sans paramètres qui appelle le premier avec la date et l'heure actuelle. (0.5 pt)
   d. Écrire les accesseurs des champs et la méthode **toString()** qui renverra les informations de cette visite . (1 pt)
3. **Classe RendezVous (2 pts)**
   a. Définir une classe RendezVous dont les caractéristiques sont : Date RendezVous, Heure RendezVous , Code patient, Observation. (0.5 pt)
   b. Ecrire un constructeur à trois paramètres : Date RendezVous , Heure RendezVous et code de patient. (0.5 pt)
   c. Écrire les accesseurs des champs et la méthode **toString()** qui renverra les informations de ce rendez-vous. (1 pt)
4. **Classe CabinetMedical (18 pts)**
   a. Définir une classe **CabinetMedical** dont les caractéristiques sont : une liste des patients, une liste des visites et une liste des rendez-vous
   b. Ajouter une méthode **ajouterPatient** qui ajoute un patient à l'ensemble des patients du cabinet. (2 pts)
   c. Ajouter une méthode **patientExitant** ayant comme paramètre le nom et le prénom et qui retourne le code du patient s'il existe et le chiffre « -1 » sinon. (2 pts)

d. Ajouter une méthode **ajouterRDV(RDV)** qui ajoute un rendez-vous. Si le rendez-vous coïncide avec un autre un message s'affiche : « Rendez-vous déjà occupé »

e. Ajouter une méthode **afficherRDVduJour** qui va pour une date donnée comme paramètre, afficher la liste des rendez-vous de ce jour.

f. Ajouter une méthode **patientAyantDesVisites** qui affiche la liste des patients (codes, nom, prénom) ayant visité le cabinet pendant la dernière semaine.

g. Ajouter une méthode **supprimerPatient** qui permet de supprimer un patient ainsi que ses visites et rendez-vous.

h. Ajouter une méthode **annulerRDV** qui permet d'annuler un rendez-vous

Exercice 2 : Gestion des Congés

L'objectif de cet exercice est de développer une application de gestion des demandes de congé des salariés d'une entreprise.

1. Classe **Employe (3,5pts)**
   a. Créer une classe **Employe** qui se caractérise par : Numéro de matricule, nom, prénom, date d'embauche et solde de jours de congés de l'employé. (0,5 pt)
   b. Créer un constructeur à 2 paramètres : nom et prénom, ce constructeur initialise le numéro de matricule (affecté de façon incrémentale par rapport au nombre des employés), et la date d'embauche par la date système. (1 pt)
   c. Ajouter des accesseurs pour tous les champs. (1 pt)
   d. Ajouter une méthode **toString** qui renvoie les informations d'un employé. (1 pt)
2. Classe **Manager (8pts)**
   a. Créer une classe **Manager** sous-classe de la classe **Employe**, elle comporte une liste des employés. Ecrire le constructeur correspondant ainsi que sa méthode **toString** qui renvoie les informations du manager. (2 pts)
   b. Ajouter une méthode **listeEmployes** qui affiche la liste des noms des employés. (2 pts)
   c. Ajouter une méthode **ajouterEmploye** qui permet d'ajouter un employé à la liste des employés. (2 pts)
   d. Ajouter une méthode **getEmployeByCode** ayant comme paramètre le numéro de matricule d'un employé et qui renvoie l'employé correspondant. (2 pts)
3. Classe **DemandeConge (2,5pts)**
   a. Créer une classe **DemandeConge** caractérisée par : code employé, date début, durée, motif, Etat (En cours, Validé, Refusé). (0,5 pt)
   b. Ajouter un constructeur à 4 paramètres : code employé, date début, durée et motif, et initialise l'état par la valeur « En cours ». (0,5 pt)
   c. Ajouter des accesseurs pour tous les champs. (0,5 pt)
   d. Ajouter les méthodes **valider** et **refuser** qui permet de modifier l'état de la demande. (1 pt)
4. Classe **GestionCongé (11pts)**
   a. Créer une classe **GestionCongé** qui se caractérise par une liste des **Manager** et une liste des **DemandeConge** et ajouter un constructeur sans paramètre. (1 pt)
   b. Ajouter une méthode **ajouterDemandeConge** qui permet d'ajouter un objet **DemandeConge** à la liste des demandes. (2 pts)
   c. Ajouter une méthode **listeDemandeCongeEnCours** qui retourne la liste des objets **DemandeConge** qui ont un état « en cours ». (2 pts)
   d. Ajouter une méthode **listeDemandeCongeParEmploye** qui retourne une liste des demandes de congé d'un employé, cette méthode a comme paramètre le code de l'employé. (3 pts)
   e. Ajouter une méthode **listeDemandeCongeParManager** qui retourne une liste des demandes de congé des employés d'un manager, cette méthode a comme paramètre le code du manager. (3 pts)

Exercice 3 : Gestion Inventaire des matériels :

L'objectif de cette application est de gérer l'inventaire par un suivi des flux d'équipements (création, modification, sortie) et gérer les équipements durant toute leur durée de vie en terme de planifications d'entretien et de remplacement (maintenance) et disponibilité d'équipement (prêt, utilisation multiple, etc.).

1. Créer une classe abstraite **Equipement** caractérisée par son code, date d'acquisition, Etat (opérationnel ou non), prix d'achat. Ajouter un constructeur à 2 paramètres : code et Etat et des accesseurs pour tous les champs. Et une méthode **toString()** qui renvoie les informations sur l'équipement sous forme d'une chaine de caractère. (2 pts)
2. Créer une classe **Ordinateur** qui hérite de la classe **Equipement**, elle aura comme caractéristiques supplémentaires : une marque et taille de l'écran. Ecrire le constructeur correspondant ainsi que sa méthode **toString**() qui renvoie les informations sur l'équipement sous forme d'une chaine de caractère. (2 pts)
3. Créer une classe **Bureau** qui hérite de la classe **Equipement**, elle aura comme caractéristiques supplémentaires sa longueur et sa largeur. Ecrire le constructeur correspondant ainsi que sa méthode **toString**() qui renvoie les informations sur l'équipement sous forme d'une chaine de caractère. (2 pts)
4. Créer une classe **Placard** qui hérite de la classe **Equipement**, elle aura comme caractéristique supplémentaire une couleur. Ecrire le constructeur correspondant ainsi que sa méthode **toString**() qui renvoie les informations sur l'équipement sous forme d'une chaine de caractère. (2 pts)

Un **établissement** se compose de plusieurs locaux, une fiche d'inventaire est établie pour chaque local, un local est caractérisé par un code, description, et une liste des équipements.

5. classe **Local (6pts)**
   a. Créer une classe **Local,** écrire le constructeur correspondant ainsi que sa méthode **toString**() qui renvoie le code et la description du local ainsi que le nombre d'équipements sans forme d'une chaine de caractères. (1 pt)
   b. Créer une classe d'exception **EquipementexistantException.** (1 pt)
   c. Ajouter une méthode **AjouterEquipenent** permet d'ajouter un équipement (ordinateur, Placard ou Bureau) à la liste de ce local
   d. Ajouter une méthode **RechercheEquipement** qui permet de rechercher un équipement par son code et qui le renvoie. (2 pts)
   e. Ajouter une méthode **FicheInventaire** qui affiche la liste des équipements. (1 pt)
6. classe **Etablissement (11pts)**
   a. Crée une classe **Etablissement** qui se caractérise par un nom, adresse, téléphone et une liste des locaux. écrire le constructeur correspondant ainsi que sa méthode **toString**() qui renvoie le nom de l'établissement ainsi que le nombre de locaux sans forme d'une chaine de caractères. (2 pts)
   b. Ajouter une méthode **AjouterLocal** qui permet d'ajouter un local. Le code est attribué automatiquement par l'application. (1 pt)
   c. Ajouter une méthode **RechercheEquipement** qui permet de rechercher un équipement par son code dans tous les locaux de la liste. (2 pts)
   d. Ajouter une méthode **EquipementNonOpérationnel** qui recherche et affiche le code local et le code d'équipement non opérationnel. (1,5 pts)
   e. Ajouter une méthode **TransferEquipement** qui permet de transférer un équipement d'un local à un autre. (1,5 pts)
   f. Ajouter une méthode **SupprimerLocal** qui permet de supprimer un local par code. (1,5 pts)

Exercice 4 : Gestion des voyages :

L'objet de cette application est la gestion informatisée des voyages organisés par une société possédant des bus. Chaque voyage est assuré par un seul chauffeur.

1- Classe Chauffeur.

a. Définir une classe Chauffeur dont les caractéristiques sont : CIN, Nom, Prénom. **(0,5 pt)**
b. Écrire un constructeur avec tous les paramètres. **(0,5 pt)**
c. Écrire les accesseurs des champs et la méthode **toString()** qui renverra tous les champs séparés par tabulation. **(0,5 pt)**

2- Classe Bus.

a. Définir une classe Bus dont les caractéristiques sont : Immatriculation, Marque, Type. **(0,5pt)**
b. Écrire un constructeur avec tous les paramètres. **(0,5 pt)**
c. Écrire les accesseurs des champs. **(0,5 pt)**
d. Écrire la méthode **toString()** qui renverra les informations du bus séparées par tabulation. **(0,5 pt)**

3- Classe Voyage

a. Définir une classe Voyage dont les caractéristiques sont : **(0,75 pt)**
- Numéro voyage : le numéro du 1er voyage créé est 1 et à chaque création d'un nouveau voyage, ce numéro doit être automatiquement incrémenté de 1. Faire le nécessaire (données+ code) pour y arriver.
- Vchauffeur : c'est **le chauffeur** qui a assuré le voyage courant.
- Vbus : C'est **le bus** conduit par Vchauffeur au cours de ce voyage.
- Date Voyage : date où a eu lieu ce voyage.
- Ville de départ.
- Ville d'arrivée.
- Nombre de voyageurs : nombre des personnes qui ont effectué ce voyage.
- Prix du billet : prix payé par chaque voyageur de ce voyage (le même prix pour tous).

b. Ecrire un constructeur sans aucun paramètre permettant de définir la valeur du numéro du voyage et d'affecter la date système à Date voyage. **(0,75 pt)**
c. Ecrire un constructeur permettant de définir les valeurs des autres champs. Ce constructeur doit faire d'abord appel au constructeur précédent. **(0,75 pt)**
d. Écrire les accesseurs des champs. **(0,5 pt)**
e. Ecrire la méthode **toString()** qui renverra le numéro du voyage, la date du voyage, le nom et le prénom du chauffeur, l'immatriculation et la marque du bus, ville de départ, ville d'arrivée et recette du voyage tous séparés par tabulation. **(1 pt)**
La recette du voyage = Nombre de voyageurs * Prix du billet du voyage en cours.

4- Programme principal

a. Déclarer une liste de chauffeurs, une liste de bus et une liste de voyages accessibles par toutes les méthodes du programme principal. **(0,5 pt)**
b. Ajouter une méthode **rechercherChauffeur()** qui recherche un chauffeur par son CIN. Si trouvé, elle retourne **le chauffeur** correspondant dans la liste des chauffeurs. Sinon, elle retourne la valeur **null**. **(0,5 pt)**
c. Ajouter une méthode **rechercherBus()** qui recherche un bus par son immatriculation. Si trouvée, elle retourne **le bus** correspondant dans la liste des bus. Sinon, elle retourne la valeur **null**. **(0,5 pt)**

d. Ajouter une méthode **rechercherVoyage()** qui recherche un voyage par son numéro. Si trouvé, elle retourne **l'indice** correspondant dans la liste des voyages. Sinon, elle retourne la valeur **-1**. **(0,5 pt)**
e. Ajouter par code (en mode conception) trois chauffeurs à la liste des chauffeurs. **(0,5 pt)**
f. Ajouter par code trois bus à la liste des bus. **(0,5 pt)**
g. Ajouter par code deux voyages à la liste des voyages. Utiliser certains constructeurs et certaines méthodes déjà conçus. **(0,75 pt)**
h. Concevoir le menu suivant : (en prenant en compte les indications ci-dessous)
    1- Ajouter bus. **(0,75 pt)**
    2- Ajouter voyage. **(1 pt)**
    3- Lister tous les voyages. **(0,75 pt)**
    4- Lister les voyages passés entre deux dates. **(0,75 pt)**
    5- Nombre de voyageurs de l'année en cours. **(0,75 pt)**
    6- Fin. **(0,25 pt)**

Indications concernant les traitements à réaliser dans le menu ci-dessus :

- Ajouter bus :
  Saisir les données nécessaires et faire le nécessaire pour que deux bus dans la liste n'aient pas la même immatriculation.
- Ajouter voyage :
  - Le numéro du nouveau voyage est automatiquement affecté par programme et la date voyage correspond à la date système.
  - Saisir le CIN du chauffeur qui doit correspondre à un chauffeur de la liste des chauffeurs et si trouvé, le chauffeur ainsi trouvé correspond à Vchauffeur du voyage à ajouter, sinon le programme vous demandera de saisir un autre CIN.
  - Saisir l'immatriculation du bus qui doit correspondre à un bus de la liste des bus et si trouvée, le bus ainsi trouvé correspond à Vbus du voyage à ajouter, sinon le programme vous demandera de saisir une autre immatriculation.
  - Saisir les autres données restantes et chaque fois qu'une donnée saisie n'est pas valide, le programme vous demandera de la saisir de nouveau.
- Lister tous les voyages : Imprimer la liste de tous les voyages de la manière suivante :

							Date : ../../….	
N°	Date voyage	Nom	Prénom	Immatric.	Marque	Ville départ	Ville arrivée	Recette
…	../../….	…….	……..	……..	……	……..	……	……..

Indication : la date en haut à droite correspond à la date système.

- Lister les voyages passés entre deux dates : Imprimer une liste des voyages semblable au niveau de la présentation à celle de la question précédente mais contenant uniquement les voyages qui se sont déroulés entre deux dates que l'utilisateur de l'application devra saisir. En bas de la liste ainsi imprimée, indiquer le nombre des voyages listés.
- Nombre de voyageurs de l'année en cours :
  Le programme va calculer et afficher le nombre de voyageurs de l'année en cours (utiliser la date système pour récupérer l'année en cours). Ce nombre correspond au nombre total des voyageurs ayant utilisé les bus de la société pendant l'année en cours.