Ecole Nationale des Sciences Appliquées de safi
A.U :2015-2016

Devoir Surveillé N°1 -Semestre2-

Exercice1 :

Qu'affiche les codes suivants ? Justifier votre réponse.

1)

```
interface C {
void affiche( );
}
-Dans le main C x= new C( );
x.affiche( )
```

2)

```
-Class A {
void f( ) { System.out.println("A");}}
- Class B extends A {
void f( ) { System.out.println("B");}}
-Class C extends B { }
-Dans le main :A x=new C( );
x.f( );
```

Exercice2 : Le but ici est de créer un programme permettant de gérer les notes des étudiants d'une classe.Pour cela on aura :

-Classe Note : contient une suite de notes(Vector<Double>) et une méthode :

+ **void ajouter (double x) :** pour ajouter une note.

-Classe Etudiant : contient le nom et le prénom de l'étudiant. Elle est aussi composée d'une référence sur un objet Note. On veut aussi différencier les notes selon les matières, on va donc créer un objet de type Hashtable< String, Note> qui sera faites d'association (une matière, les notes de l'étudiant dans cette matière).car dans une matière l'étudiant pourra avoir plusieurs notes : Note de examen1,examen2…Cette classe a les méthodes suivantes :

+**Etudiant (String nom,String prenom)** :le constructeur

+**void ajouterNoteMatiere(String matiere,int note) :** ajoute une note de l'étudiant concerné dans une matière.

+ **double moyenDansMatiere(String matiere) :**retourne la moyenne de l'étudiant concerné dans la matière passée en paramètre

-Classe Groupe : On Créer une classe Groupe composée d'une collection des étudiants (Vector<Etudiant>).Cette classe se compose de :

+**void inscrireEtudiant(Etudiant e) :** une méthode inscrire qui permet d'inscrire un étudiant dans le groupe ;

+**void afficherEtudiants(String matiere) :** affiche les notes des étudiants du groupe dans une matière par ordre alphabétique des noms des étudiants. Une exception de type **NullPointerException** doit être traitée au cas où un étudiant n'a eu aucune note dans cette matière.

+une méthode **admis(String matière):** qui retourne l'ensemble des étudiants qui ont validé la matière concernée (la matière est validée si la note >=12);

+afficher un **histogramme** des moyennes du groupe dans une matière. Par exemple, si les étudiants ont obtenu les moyennes suivantes : omar 16,3; sara 18; ahmed 15,7;karim 12,2; nada 15,9; fatima 11,4; mohmad 11; amal 11,1 , on affiche le diagramme suivant (limité entre 10 et 20 ici) :

[10 – 11[

[11 – 12[***

[12 – 13[*

[13 – 14[

[14 – 15[

[15 – 16[**

[16 – 17[*

[17 – 18[

[18 – 19[*

[19 – 20]

Exercice3 : L'objectif de cet exercice est la construction d'une application qui permet de gérer des entreprises avec leurs employés. Un employé est une entité qui a un nom(String), un numéro de téléphone(String) et l'entreprise dans laquelle il travaille. Une entreprise est une entité qui a un nom et une liste de des employés. On peut emboucher et renvoyer des employés d'une entreprise. Attention, on ne peut pas renvoyer un employé d'une entreprise dans laquelle il ne travaille pas ou ajouter un employé déjà embouché dans une entreprise donnée.

L'association entre une entreprise et ses employés sera bidirectionnelle : une entreprise connaît ses employés et un employé connaît son entreprise ; attention de bien gérer les 2 "bouts" de l'association. Par exemple, si vous ajoutez un employé dans une entreprise ETR, l'employé doit être ajouté à la liste des employés de l'entreprise mais l'entreprise de l'employé doit aussi être mise à la valeur ETR.

En plus, on ajoute un employé particulier à chaque entreprise qui prend le poste de directeur. Un directeur a un salaire que l'on donne quand on le crée. On n'a qu'un seul directeur par entreprise. Un problème de conception se pose : comment faire pour empêcher la création de plusieurs directeurs dans une entreprise !

L'ensemble des fonctionnalités demandées sont données par les squelettes des classes à construire:

public class Employe{

pubic String nom;	
public String telephone;	
private Entreprise	doit être null si l'employé n'est pas embouché par aucune entreprise(càd pour un employé en chômage Entreprise=null)
public Employe(String nom, String telephone)	Constructeur de la classe

public Entreprise getEntreprise( )	Getter permettant de retourner la valeur de Entreprise
public void setEntreprise(Entreprise NewEntreprise)	Fonction permettant de modifier l'entreprise employeur d'un employé. La modification se termine correctement si l'employé sera embouché par le paramètre NewEntreprise et si l'ancienne entreprise a viré cet employé(cette dernière vérification se fait si l'entreprise courant est non null).Dans le cas contraire la fonction gère une exception personnalisée de type **EmployeException** .
public void demissionne()	Fonction permettant de gérer la démission d'un employé de son entreprise.On ne peut pas appliquer cette fonction pour un chômeur sinon il faut gérer une **EmployeException**.
public String toString()	Fonction permettant d'afficher les infos d'un employé avec le nom, le numéro de téléphone et le nom de son entreprise employeur s'il n'est pas au chômage

public class Entreprise {

pubic String nom;	Le nom de l'entreprise
public Vector<Employe> ListEmployes;	Liste des employés de l'entreprise courante
private Directeur directeur	Directeur de l'entreprise qui est unique.Une entreprise sans directeur :Directeur=null
public Entreprise(String nom)	Constructeur de la classe (initialise le nom et ListEmployes par un Vector <Employe>)
-public Directeur getDirecteur()	Getter permettant de retourner la valeur de Directeur
public void setDirecteur(Directeur dir)	Fonction permettant de modifier le directeur d'une entreprise (Faites attention la modification n'est pas par une simple affectation mais le directeur est déjà un employé de l'entreprise+l'ancien directeur devra être viré de l'entreprise si il est différent de null)
public boolean estUnEmploye(Employe emp)	Fonction permettant de vérifier si l'employé donné comme paramètre est un employé de l'entreprise courante ou non
public void emboucherEmploye(Employe emp) throws EmployeException	Fonction permettant d'emboucher un employé, s'il ne l'est pas, dans l'entreprise courante. Si l'employé est déjà embouché par une autre entreprise non null alors il faut qu'il démissionne de cette dernière avant que l'entreprise courante puisse l'emboucher.En plus si l'employé est une instance de Directeur il faut l'affecter dans la propriété « directeur »
public void virerEmploye(Employe emp) throws EmployeException	Fonction permettant de virer un employé de l'entreprise courante. Si l'employé est une instance de Directeur il faut affecter null dans la propriété « directeur »
public String toString()	Fonction permettant d'afficher le nom et la liste des employés de l'entreprise courante trié par ordre croissant des noms.

public class Directeur {
}???
A déterminer son squelette. Cette classe doit contenir une méthode **creerDirecteur** qui prend en paramètre un directeur et une entreprise afin de créer pour cette dernière un directeur(N'oubliez pas que l'entreprise a un seul directeur)

1.Ecrivez un programme implémentant les trois classes précédentes et la classe EmployeException.

2. Ecrivez un exemple d'une classe main pour appeler ces classes.

Annexe :

Les méthodes de la classe Vector :
-void addElement(Object obj) : ajoute un nouvel élément au vecteur. **-void remove (Object obj) :** supprime l'élément qui passe en paramètres. **-Object elementAt(int index) :** retourne l'élément demandé. **-Enumeration elements() :** retourne l'ensemble des éléments du vecteur **-int size() :** retourne le nombre d'éléments du vecteur.
Les méthodes de la classe HashTable :
-void put(Object key, Object value) : Insère une clé et une valeur dans la table de hachage. **-Object get (Object key) :** Renvoie l'objet qui contient la valeur associée à la clé. Si la clé n'est pas dans la table de hachage, un objet null est renvoyé. **-Enumeration elements() :** retourne sous forme d'une énumération tous les éléments de la table, **-Enumeration keys() :** retourne cette fois ci toutes les clefs, **-int size() :** nombre d 'éléments dans la table
Enumération :
-boolean hasMoreElements(): retourne « true » si l'énumération comporte encore des éléments. **-Object nextElement():** récupère l'élément suivant de l'énumération.
Comparable/CompareTo
-Comparable : interface pour définir un ordre de tri, contient la méthode compareTo (Object o). **-Comparator :** interface pour définir un ordre de tri quelconque, contient la méthode compare (Object o1, Object o2)
Collections :
-Collections.Sort(Liste list) :Permet de trier une collection des objets. **-Collections.binarySearch(Liste list, élément) :** rend la position de élément dans la collection ou une valeur < 0 sinon. La collection doit être auparavant trié(par ordre croissant), la recherche est une recherche dichotomique. **-Collections.max(Liste list):** rend la valeur maximale dans une liste
Les Exceptions :
- Le mot-clé "throw" permet de soulever une exception ex: if(b==0) throw new ArithmeticException();
Autres méthodes :
-instanceOf( ) : Permet de savoir si l'objet référencé est une instance d'une classe donnée(exemple -if objet1 instanceOf(ClasseX)-) **-La lecture en java** :utilise un objet de la classe Scanner : Scanner sc=new Scanner(System.in). **-L'écriture en java:**System.out.prinln("la valeur="+x); **-String substring(int debut, int fin):**extrait une sous-chaîne va du caractère en position du premier argument jusqu' au caractère en position du 2ème argument moins 1: Exemple: chaine1="Bonjour" String chaine3= chaine1.substring(0,3) ; chaine3=Bon **-public static void** main(String[] args): programme principal.