Ecole **N**ationale des **S**ciences **A**ppliquées de safi

A.U :2013-2014

Devoir Surveillé N°1 -Semestre2-

'La clarté de vos programmes et sa présentation seront prise en compte dans la notation.'

Exercice1 :

Qu'affiche les codes suivants ? Justifier votre réponse.

1)

```
Class C {
public static int i ;
public int  j;
public C ( ){i++;  j=i; }
 }
C x= new C( );  C  y=new C( );
System.out.println( x.i+ "et" +y.j);
```

2)

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

3) Soient les classes A et B :

```
class A{
public int x ;
public A( ){x=5;}
}
```

```
Class B extends A{
public B( ) {x++;}
public B(int i){
this( ); //appel le constructeur B( )
x=x+i; }
public B(String s){super( );
x--;}
```

```
B b1=new B( );  B b2=new B(2003);
B b3=new B("Bonjour");
System.out.println(b1.x+"et"+b2.x+"et encore"+b3.x) ;
```

Exercice2 :

L'objectif de cet exercice est d'écrire une classe **Annuaire** pour mémoriser des numéros de téléphone et d'adresses.Chaque entrée est représentée par une fiche à plusieurs champs de type String :un nom,un numéro de téléphone et une adresse.La structure des fiches est décrite par une classe **Fiche**,de plus la classe Annuaire à utiliser comporte une table associative Hashtable< String, Fiche> qui sera faites d'association (un nom,une fiche).

1. Construire la classe Fiche,faites simple,ce n'est qu'une classe auxiliaire « constructeur,les accesseurs,toString).

2. Construire la classe Annuaire en ajoutant son propre champ(le hashTable),son constructeur ainsi que les fonctions suivantes :

-void rechercherFicheParNom(String nom) : cette fonction permet de rechercher et d'afficher la fiche concernant le nom indiqué s'il existe sinon afficher un message montrant que cette fiche n'existe pas.

-void ajouterFicheParNom(String nom) :cette fonction permet d'ajouter un nouveau enregistrement(saisi par l'utilisateur) dans l'annuaire si la clé donnée comme argument à la fonction n'existe pas.Si la clé existe, on demande à l'utilisateur s'il veut modifier ou non les valeurs de la fiche de la clé correspondante.

-String toString( ) :retourne une chaine représentant toutes les fiches de l'annuaire.

Remarque : on ne peut pas appeler la fonction binarySearch directement sur une collection de type HashTable .

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 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{

private String nom;	
private 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 String getNom( ) **public String getTelephone( )** **public Entreprise getEntreprise( )**	Les getters permettant de retourner les valeurs des variables d'instance de la classe Employe
public void setNom(String newNom) **public void setTelephone(String NewTelephone)**	Fonctions permettant de modifier le nom et le le numéro de téléphone de l'employé courant.
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 {

private String nom;	Le nom de l'entreprise
private 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 String getNom() **public Vector <Employe> getListEmployes( )** **public Directeur**	Les getters permettant de retourner les valeurs des variables d'instance de la classe Entreprise.

getDirecteur()	
public void setNom(String newNom)	Fonction permettant de modifier le nom de l'entreprise,
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.

-*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).