

## TP N°3

- Objectifs :
- Mettre en pratique les notions d'héritage et de polymorphisme
  - Définir des classes abstraites
  - Implémenter les interfaces Comparable et Serializable

Nous proposons de coder les classes suivantes pour gérer le service Ressources Humaines de l'université :

Personne	Personnel	Enseignant	Administratif	ContactExt
- nom : String - prenom : String - age : int	- nom : String - prenom : String - age : int - contrat : Contrat	- nom : String - prenom : String - age : int - contrat : Contrat - grade : Grade	- nom : String - prenom : String - age : int - contrat : Contrat - poste : Poste	- nom : String - prenom : String - age : int - type : TypeContact
+ Personne(...) + Getters + Setters + afficher()	+ Personnel(...) + Getters + Setters + afficher()	+ Enseignant(...) + Getters + Setters + afficher()	+ Administratif(...) + Getters + Setters + afficher()	+ ContactExt(...) + Getters + Setters + afficher()

1. Définir les classes énumérations suivantes :

Contrat : CDI, CDD, INTERIM

Grade : MAA, MAB, MCA, MCB, PR

Poste : MAINTENANCE, SECURITE, LOGISTIQUE

TypeContact : CHEF, SALARIE

2. Implémenter chaque classe en définissant au préalable les relations d'héritage permettant d'éviter la redondance des données. Définir en classes abstraites celles qu'on ne peut instancier.

3. Créer une nouvelle classe nommée ServiceRH définie par le nom de l'université et une liste de personnes puis implémenter dans cette classe les méthodes suivantes :

- ajouter(Personne p) : permet d'ajouter une personne au service.
- supprimer(Personne p) : supprime du service la personne entrée en argument.
- typePersonne(Personne p) : affiche le type d'une personne (Enseignant, Adm. ou Contact extérieur).
- afficher() : affiche les personnes et toutes leurs informations selon leur type.
- recherche(String nom,String prenom) : retourne la personne de nom et prénom donnés en paramètre.

4. Définir le programme principal :

- Créer les personnes suivantes puis les ajouter au service RH de l'USTO.

Type	Nom	Prénom	Age	Contrat	Grade	Poste	TypeContact
Enseignant	MOUSSA	Karim	30	CDI	PR	-	-
Enseignant	OUALI	Majda	37	INTERIM	MCA	-	-
Adm	BADJI	Ahmed	28	CDD	-	SECURITE	-
Adm	SAAD	Khalil	40	CDI	-	LOGISTIQUE	-
Adm	MOUAD	Ibrahim	50	CDI	-	MAINTENANCE	-
ContactExt	OUASTI	Halima	30	-	-	-	CHEF
ContactExt	MOUSSA	Imad	28	-	-	-	SALARIE

- Afficher la liste des personnes du service.
- Rechercher une personne (de nom et prénom lus au clavier) dans le service.
- Supprimer une personne (recherchée au préalable) si elle existe et réafficher la liste des personnes.

5. On voudrait trier la liste des personnes du service RH par ordre alphabétique du nom. Implémenter l'interface **Comparable** et afficher le résultat obtenu après le tri.

6. Utiliser l'interface **Serializable** pour stocker la liste des personnes sur fichier.

## **TP N°3**

---

Objectifs :

- Mettre en pratique les notions d'héritage et de polymorphisme
- Définir des classes abstraites
- Implémenter des méthodes abstraites

---

Un parc auto contient 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 privés. Les accesseurs (get...) et les mutateurs (set....) seront créés pour accéder et modifier ses attributs.
- La classe Véhicule possède également deux méthodes abstraites demarrer() et accelerer() 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

## TP N°3

---

Objectifs :

- Mettre en pratique les notions d'héritage et de polymorphisme
  - Manipuler des dates
- 

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*, un constructeur et la redéfinition de la méthode « afficher ».
- Une classe **Chef** qui dérive de la classe *Employé*, avec en plus un champ *Service*, un constructeur et la redéfinition de la méthode « afficher ».
- Une classe **Directeur** qui dérive de la classe *Chef*, avec en plus un champ *Société*, un constructeur et la redéfinition de la méthode « afficher ».

**Travail à faire:**

- Ecrire les classes *Personne*, *Employé*, *Chef* et *Directeur*.
- Créer un programme de test qui comporte un 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*).
- A l'aide de la boucle `for`, afficher l'ensemble des éléments du tableau puis uniquement les chefs.
- A l'aide de la boucle `foreach`, afficher l'ensemble des éléments du tableau puis uniquement les chefs.

## TP N°3

- 
- Objectifs :
- Mettre en pratique les notions d'héritage et de polymorphisme
  - Définir des classes abstraites
  - Implémenter des méthodes abstraites
  - Manipuler des dates
- 

Soit la classe abstraite `Employe` caractérisée par les attributs suivants :

- Matricule
- Nom
- Prénom
- Date de naissance

La classe `Employé` doit disposer des méthodes suivantes :

- un constructeur d'initialisation
- 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=20000 (DA)`
- L'ouvrier a un salaire mensuel qui est :  $\text{Salaire} = \text{SMIG} + (\text{Ancienneté en année}) * 3000$ .
- De plus, le salaire ne doit pas dépasser  $\text{SMIG} * 3$ .

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

- Le cadre a un salaire qui dépend de son indice :
  - 1 : salaire mensuel 33000 DA
  - 2 : salaire mensuel 45000 DA
  - 3 : salaire mensuel 67000 DA
  - 4 : salaire mensuel 80000 DA

Un patron est un employé qui se caractérise par un chiffre d'affaire (CA) et un pourcentage.

- Le chiffre d'affaire est commun entre les patrons.
- Le patron a un salaire annuel qui est égal à un pourcentage du chiffre d'affaire  
 $\text{Salaire} = \text{CA} * \text{pourcentage} / 100$

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.
4. Dans une classe `Lanceur`, tester votre travail en créant une collection de 5 employés :

Classe	Matricule	Nom	Prénom	Né(e) le	Entré le	indice	CA	%age
Ouvrier	0001	Ali	Omar	11/03/2001	04/05/2019		-	
Ouvrier	0002	Taleb	Yacine	05/04/1980	01/01/2021		-	
Ouvrier	0003	Charif	Nadia	01/05/1978	31/12/2000		-	
Cadre	0004	Kamli	Meriem	10/03/2000	-	3		
Patron	0005	Benali	Malik	02/07/2001	-	-	3 000 000	20%

Afficher les employés avec leurs salaires mensuels.