ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT
TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

TP2

Programmation Orientée Objet Java

2022-2023

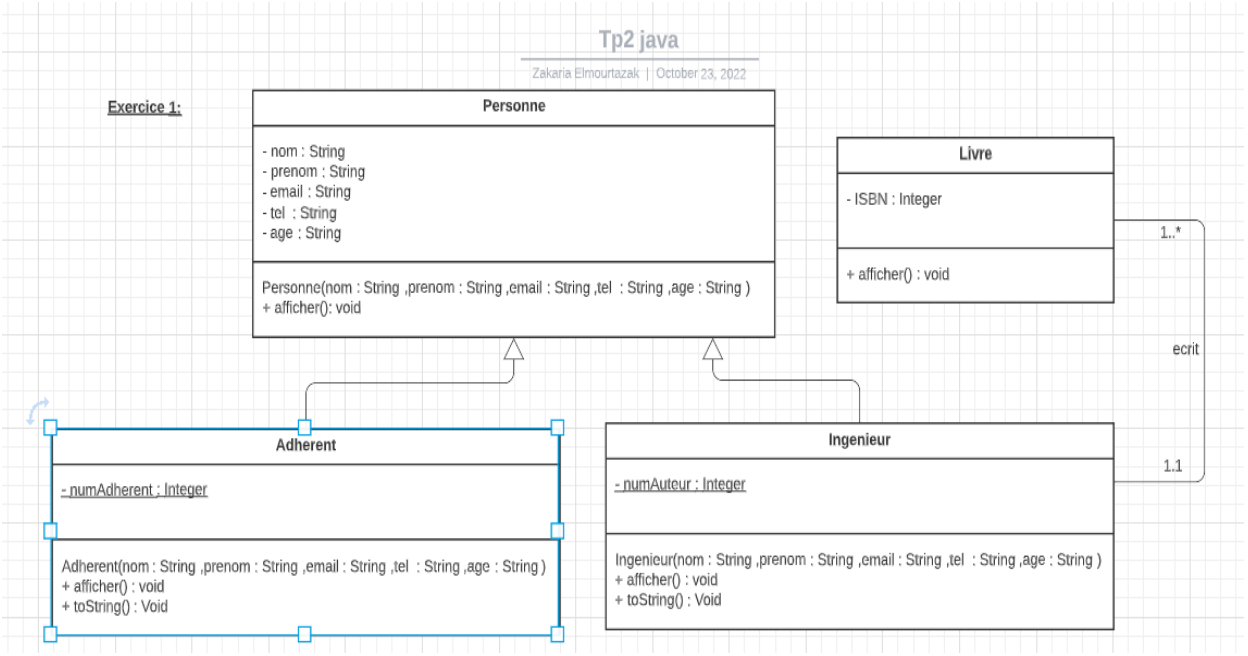
RÉALISÉ PAR : ZAKARIA EL MOURTAZAK
ENSET-M

Exercice 1:

- On souhaite créer une application JAVA pour la gestion des livres et des adhérents d'une bibliothèque.

● L'analyse de l'application :

- Premièrement et avant de partir de la partie développement nous schématisons un diagramme de classes afin de mieux comprendre l'aperçu général le schéma de notre application en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets. avec l'utilisation de site web **lucidchart**



● Développement de l'application :

- Après la partie modélisation du schéma, notre application devient plus claire et on peut commencer la partie développement avec la création des deux packages, un packages **métier** qui va contenir les différentes fonctionnalités de l'application et l'autre package qui est **présentation** qui va contenir l'utilisation de ces fonctionnalités
- on crée maintenant les différentes classes nécessaire de notre application

⇨ Class Personne

- on crée la classe Personne avec les attributs privés : **nom, prénom, email, tel, et âge.**
- on ajoute le **constructeur avec paramètres** pour initialiser les différents attributs et la méthode **afficher**() pour afficher ces attributs. on ajoute aussi une méthode **toString** pour l'utiliser à l'intérieur de la méthode afficher

```java
public class Personne {
    2 usages
    private String nom;
    2 usages
    private String prenom;
    2 usages
    private String email;
    2 usages
    private double tel;
    2 usages
    private int age;
    2 usages
    public Personne(String nom, String prenom, String email, double tel, int age) {
        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
        this.tel = tel;
        this.age = age;
    }
    @Override
    public String toString() {
        return
                "nom='" + nom + '\'' +
                ", prenom='" + prenom + '\'' +
                ", email='" + email + '\'' +
                ", tel=" + tel +
                ", age=" + age;
    }
    4 usages  2 overrides
    public void afficher() { System.out.print(toString()); }
}
```

- Class Adhérent

- On crée une deuxième classe **Adhérent** qui hérite de la classe **Personne** et qui contient l'attribut static **numAdherent** initialisé par zéro car au moment de la création de la class **Adhérent** le nombre adhérent doit être zéro
- Au niveau des méthodes on a le **constructeur avec paramètres** pour initialiser les différents de la class mere et incrémente l'attribut **numAdherent**
- on redéfinit la méthode **afficher()** pour afficher les différents attributes de la class mere, ainsi l'attribut **numAdherent**

```java
package metier;

public class Adherent extends Personne {
    2 usages
    private static int numAdherent =0;

    1 usage
    public Adherent(String nom, String prenom, String email, double tel, int age) {
        super(nom, prenom, email, tel, age);
        numAdherent++;
    }

    4 usages
    public void afficher(){
        super.afficher();
        System.out.println(", numAdherent: "+numAdherent);
    }

}
```

⇨ Class Auteur

- On créez une troisième classe Auteur qui hérite de la classe Personne, qui contient l'attribut **numAuteur** initialisé par zéro car au moment de la création de la class **Adhérent** le nombre adhérent doit être zéro
- Au niveau des méthodes on a le **constructeur avec paramètres** pour initialiser les différents de la class mere et incrémente l'attribut **numAuteur**
- on redéfinit la méthode **afficher()** pour afficher les différents attributes de la class mere, ainsi l'attribut **numAuteur**

```java
package metier;

public class Auteur extends Personne {
    2 usages
    private static int numAuteur=0;

    1 usage
    public Auteur(String nom, String prenom, String email, double tel, int age) {
        super(nom, prenom, email, tel, age);
        numAuteur++;
    }

    4 usages
    public void afficher(){
        super.afficher();
        System.out.println(", numAutheur: "+numAuteur);
    }
}
```

⇨ Class Livre

- On crée une troisième classe livre qui contient un attribut **ISBN** (entier) et un **auteur**.
- Au niveau des méthodes on a le **constructeur avec paramètres** pour initialiser les différents attributes de la class
- on redéfinit la méthode **afficher()** qui affiche l' **ISBN**, le titre et les informations de l'auteur.

```
package metier;

public class Livre {
    2 usages
    private int ISBN;
    2 usages
    private  Auteur auteur;
    1 usage
    public void afficher(){
        this.auteur.afficher();
        System.out.println("ISBN: "+ISBN);
    }

    1 usage
    public Livre(int ISBN, Auteur auteur) {
        this.ISBN = ISBN;
        this.auteur = auteur;
    }
}
```

⇨ Class main

- finalement on crée la méthode main() pour tester les différentes classes,

dans laquelle :

— déclarez et intenter un adhérent ;

— déclarez et instanciez un livre qui est écrit par un auteur ;

— affichez les informations de l'adhérent et du livre.

```java
//EXERCICE 1
Adherent adherent = new Adherent( nom: "adsa",  prenom: "asda",  email: "dasdas@gmail.com",  tel: 3234243, age: 12);
Auteur auteur =new Auteur( nom: "dsad", prenom: "dsad", email: "dasdas@gmail.com", tel: 2133123, age: 21);
Livre livre  =new Livre( ISBN: 1313,  auteur);
adherent.afficher();
livre.afficher();
```

- **L'exécution du code est donné par:**

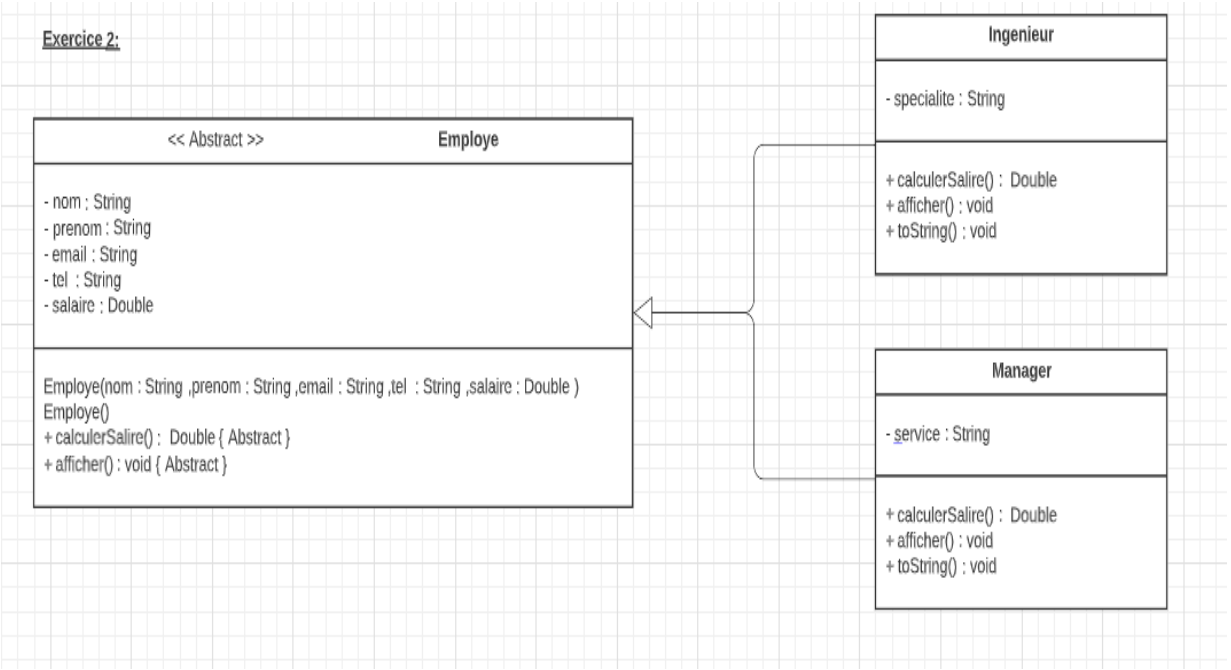
```
"C:\Program Files\Java\jdk1.8.0_341\bin\java.exe" ...
Connected to the target VM, address: '127.0.0.1:62560', transport: 'socket'
nom='adsa', prenom='asda', email='dasdas@gmail.com', tel=3234243.0, age=12, numAdherent: 1
nom='dsad', prenom='dsad', email='dasdas@gmail.com', tel=2133123.0, age=21, numAutheur: 1
ISBN: 1313
Disconnected from the target VM, address: '127.0.0.1:62560', transport: 'socket'
```

Exercice 2:

- On souhaite créer une application en java qui permet de gérer les salaires des ingénieurs et des managers d'une entreprise de développement informatique.

- **L'analyse de l'application :**

- Premièrement et avant de partir de la partie développement nous schématisons un diagramme de classes afin de mieux comprendre l'aperçu général le schéma de notre application en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets. avec l'utilisation de site web **lucidchart**



● Développement de l'application :

- Après la partie modélisation du schéma, notre application devient plus claire et on peut commencer la partie développement avec la création des deux packages, un packages **métier** qui va contenir les différentes fonctionnalités de l'application et l'autre package qui est **présentation** qui va contenir l'utilisation de ces fonctionnalités
- on crée maintenant les différentes classes nécessaire de notre application

⇨ Class Employee

- On crée la classe abstraite **Employé** avec les attributs nom, prenom, email, telephone, et salaire.
- on ajoute les constructeurs avec et son paramètres, puis la méthode abstraite **calculerSalire**() qui retourne le salaire d'un employé.

```java
package metier;

public abstract class Employee {
    3 usages
    protected String nom;
    3 usages
    protected String prenom;
    3 usages
    protected String email;
    3 usages
    protected double telephone;
    7 usages
    protected double salaire;

    2 usages
    public Employee() {
    }

    2 usages
    public Employee(String nom, String prenom, String email, double telephone, double salaire) {
        this.nom = nom;
        this.prenom = prenom;
        this.email = email;
        this.telephone = telephone;
        this.salaire = salaire;
    }
    2 implementations
    public abstract double calculerSalire();
}
```

⇨ Class Ingénieur

- On crée la classe **Ingénieur** avec l'attribut spécialité.
- On redéfinit la méthode **calculerSalire**() sachant qu'on prévoit une augmentation de 15% par rapport à son salaire
- On crée aussi la méthode toString qui l'on utilisé à l'intérieur de la méthode **afficher**() pour afficher les different attributes de la classe

```java
package metier;

public class Ingenieur extends Employee{
    3 usages
    private String specialite;
    @Override
    public double calculerSalire() { return salaire =salaire*.15; }
    public Ingenieur(String specialite) { this.specialite = specialite; }
    @Override
    public String toString() {
        return "Ingenieur{" +
                "nom='" + nom+ '\'' +
                ", prenom='" + prenom + '\'' +
                ", email='" + email + '\'' +
                ", telephone=" + telephone +
                ", specialite=" + specialite + '\'' +
                ", salaire=" + salaire +
                '}';
    }
    1 usage
    public void afficher(){

        System.out.println(toString());
    }
    1 usage
    public Ingenieur(String nom, String prenom, String email, double telephone, double salaire, String specialite) {
        super(nom, prenom, email, telephone, salaire);
        this.specialite = specialite;
    }
}
```

⇨ Class Manager

- On crée la classe **Manager** avec l'attribut service.
- On redéfinit la méthode **calculerSalire**() sachant qu'on prévoit une augmentation de 20% par rapport à son salaire normal.
- On crée aussi la méthode toString qui l'on utilisé à l'intérieur de la méthode **afficher**() pour afficher les different attributes de la classe

```java
package metier;

public class Manager extends Employee {
    3 usages
    private String service;
    public Manager(String service) {
        this.service = service;
    }
    1 usage
    public Manager(String nom, String prenom, String email, double telephone, double salaire, String service) {
        super(nom, prenom, email, telephone, salaire);
        this.service = service;
    }
    @Override
    public String toString() {
        return "Manager{" +
                "Ingenieur{" +
                "nom='" + nom+ '\'' +
                ", prenom='" + prenom + '\'' +
                ", email='" + email + '\'' +
                ", telephone=" + telephone +
                ", service='" + service + '\'' +
                ", salaire=" + salaire +
                '}';
    }
    1 usage
    public void afficher() { System.out.println(toString()); }

    @Override
    public double calculerSalire() { return salaire + salaire * .20; }
}
```

⇨ Class Main

- finalement on crée la méthode main() qui contient une méthode **main**() pour tester les différentes

  classes, dans laquelle :

  — On déclare et initialise un ingénieur ;

  — On déclare et initialise un manager ;

  — On affiche les informations de l'ingénieur et du manager (nom, prénom, salaire,

  service, et spécialité).

```
EXERCICE 2
Ingenieur ingenieur  =new Ingenieur( nom: "dsadd", prenom: "dsada", email: "dasd", telephone: 20, salaire: 20, specialite: "dsada");
Manager manager  = new Manager( nom: "dsad", prenom: "dsahaskd", email: "dasd", telephone: 222, salaire: 22, service: "dasd");
ingenieur.afficher();
System.out.println();
manager.afficher();
```

- **L'exécution du code est donné par :**

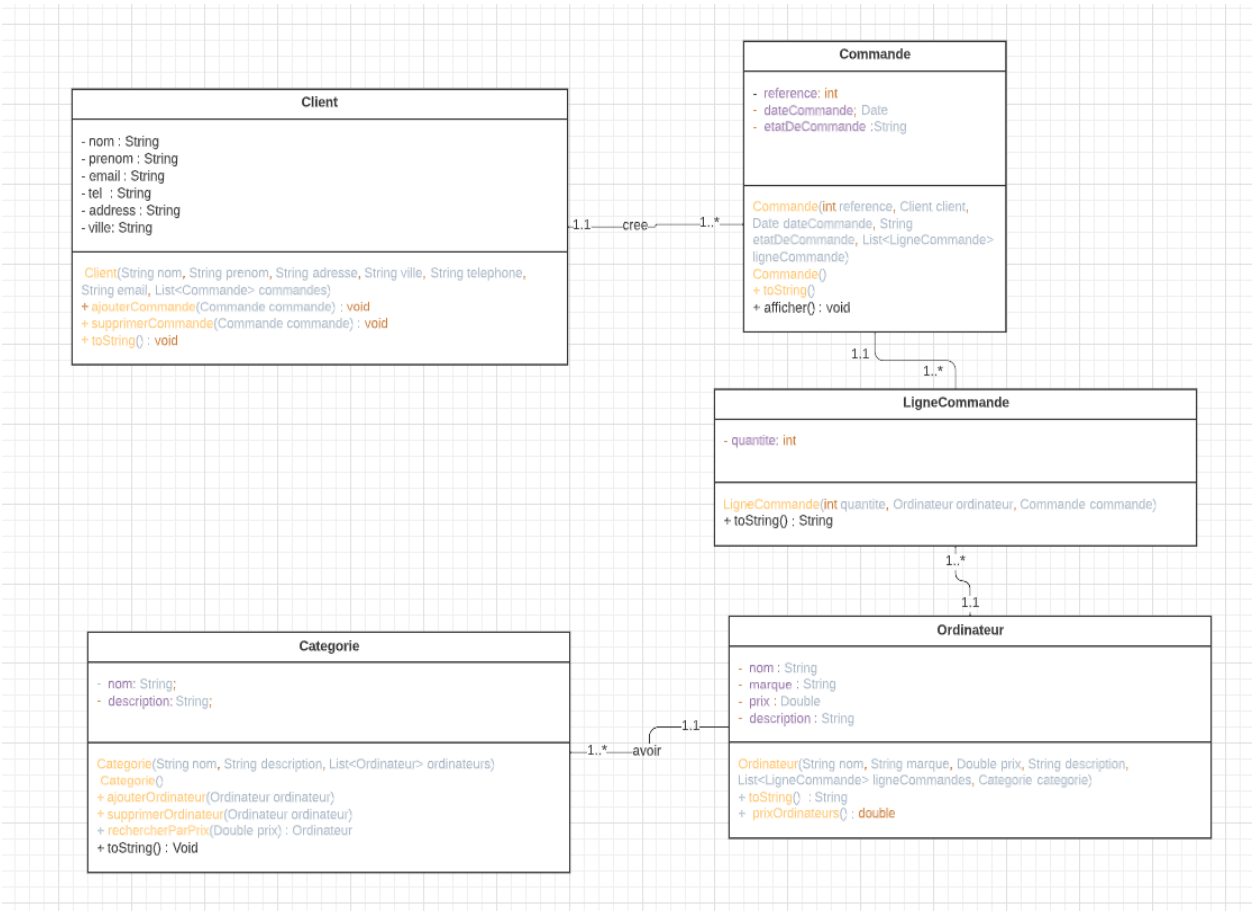
```
Connected to the target VM, address: '127.0.0.1:63259', transport: 'socket'
Ingenieur{nom='dsadd', prenom='dsada', email='dasd', telephone=20.0, specialite=dsada', salaire=20.0}

Manager{Ingenieur{nom='dsad', prenom='dsahaskd', email='dasd', telephone=222.0, service='dasd', salaire=22.0}
Disconnected from the target VM, address: '127.0.0.1:63259', transport: 'socket'
```

Exercice 3:

- On souhaite réaliser une application JAVA qui gère les commandes des clients d'une entreprise qui vend des ordinateurs. L'application demandée doit donner la possibilité de gérer les ordinateurs, les catégories, et les commandes de l'entreprise.

● L'analyse de l'application :

- Premièrement et avant de partir de la partie développement nous schématisons un diagramme de classes afin de mieux comprendre l'aperçu général le schéma de notre application en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets. avec l'utilisation de site web **lucidchart**



● Développement de l'application :

- Après la partie modélisation du schéma, notre application devient plus claire et on peut commencer la partie développement avec la création des deux packages, un packages **métier** qui va contenir les différentes fonctionnalités de l'application et l'autre package qui est **présentation** qui va contenir l'utilisation de ces fonctionnalités
- on crée maintenant les différentes classes nécessaire de notre application

⇨ Class Ordinateur

- On crée la classe **Ordinateur** avec les attributs nom, marque, prix, description, et nombre en stock. Chaque ordinateur appartient à une catégorie. Ajoutez une méthode
- On redéfinit la méthode qui retourne le prix pour une quantité donnée
- On crée aussi la méthode toString qui nous renvoie une chaîne de caractères représentant les attributs de l'objet.

```
package metier;

import java.util.List;

16 usages   zakaria-root *
public class Ordinateur {
    4 usages
    private String nom;
    4 usages
    private String marque;
    5 usages
    private Double prix;
    4 usages
    private String description;
    4 usages
    List<LigneCommande> ligneCommandes;

    4 usages
    private Categorie categorie;
```

```java
    public String getNom() { return nom; }

    zakaria-root
    public String getMarque() { return marque; }

    1 usage  zakaria-root
    public Double getPrix() { return prix; }

    zakaria-root
    public String getDescription() { return description; }

    zakaria-root
    public static int getNombreOrdinateur() { return nombreOrdinateur; }

    3 usages
    static private int nombreOrdinateur = 0;

    zakaria-root
    public void setNom(String nom) { this.nom = nom; }

    zakaria-root
    public void setMarque(String marque) { this.marque = marque; }

    zakaria-root
    public void setPrix(Double prix) { this.prix = prix; }

    zakaria-root
    public void setDescription(String description) { this.description = description; }

    zakaria-root
    public static void setNombreOrdinateur(int nombreOrdinateur) { Ordinateur.nombreOrdinateur = nombreOrdinateur; }
```

```java
3 usages  zakaria-root                                                                                                                                                    ⚠ 11  ✓ 37  ^
public Ordinateur(String nom, String marque, Double prix, String description, List<LigneCommande> ligneCommandes, Categorie categorie) {
    this.nom = nom;
    this.marque = marque;
    this.prix = prix;
    this.description = description;
    this.ligneCommandes = ligneCommandes;
    this.categorie = categorie;
    nombreOrdinateur++;
}
zakaria-root
@Override
public String toString() {
    return "  \nnom d'ordinateur :"+nom +
            " \nla marque d'ordinateur : " + marque +
            " \nle prix d'ordinateur : " + prix +
            " \nla description d'ordinateur : " + description +
            " \nle totale d'ordinateur : " + prixOrdinateurs() +
            "" + categorie ;
}

1 usage  zakaria-root
public double prixOrdinateurs() {
    double somePrix =0;
    for (LigneCommande ligne : ligneCommandes) {
        somePrix +=  ligne.getQuantite() * prix;
    }
    return somePrix;
}
```

- Class Catégorie

- On crée la classe **Categorie** avec les attributs nom, description et une liste d'ordinateurs. pour supprimer un ordinateur, et une méthode
- rechercherParPrix() qui retourne la liste des ordinateurs par un prix donné en paramètres.
- On définit la méthode **ajouterOrdinateur**() pour ajouter un nouvel ordinateur à la liste (vous devez vérifier s'elle existe déjà avant de l'ajouter), une méthode **supprimerOrdinateur**() .
- On crée aussi la méthode toString qui nous renvoie une chaîne de caractères représentant les attributs de l'objet.

```
2 usages
private String nom;
2 usages
private String description;
6 usages
List<Ordinateur> ordinateurs;

1 usage  zakaria-root
public Categorie(String nom, String description, List<Ordinateur> ordinateurs) {
    this.nom = nom;
    this.description = description;
    this.ordinateurs = ordinateurs;
}
 zakaria-root
public Categorie() {
}
```

```
    public void ajouterOrdinateur(Ordinateur ordinateur) {
        if (ordinateurs.contains(ordinateur)) {
            ordinateurs.add(ordinateur);
        } else {
            System.out.println("l'ordinateur existe deja");
        }
    }

     zakaria-root
    public void supprimerOrdinateur(Ordinateur ordinateur) {
        if (ordinateurs.contains(ordinateur)) {
            ordinateurs.remove(ordinateur);
        } else {
            System.out.println("l'ordinateur n'existe pas");
        }

    }

     zakaria-root
    public Ordinateur rechercherParPrix(Double prix) {
        for (Ordinateur ordinateur : ordinateurs) {
            if (ordinateur.getPrix() == prix) {
                return ordinateur;
            }
        }
        return null;
    }
```

```
zakaria-root
@Override
public String toString() {
    return "" +
            "\nnom du categorie : " + nom + "    "+
            "\ndescription du categorie :   " + description ;
}
```

- Class Commande

- On crée une classe Commande avec les attributs référence, le client, la date de commande, et l'état de la commande.
- On crée aussi la méthode toString qui nous renvoie une chaîne de caractères représentant les attributs de l'objet.

```java
import java.util.Date;
import java.util.List;

12 usages  zakaria-root *
public class Commande {
    4 usages
    private int reference;
    4 usages
    private Client client;
    4 usages
    private Date dateCommande;
    4 usages
    private String etatDeCommande;

    4 usages
    private List<LigneCommande> ligneCommandes;

    1 usage  new *
    public Commande(int reference, Client client, Date dateCommande, String etatDeCommande, List<LigneCommande> ligneCommande) {
        this.reference = reference;
        this.client = client;
        this.dateCommande = dateCommande;
        this.etatDeCommande = etatDeCommande;
        this.ligneCommandes = ligneCommande;
    }

    new *
    public Commande() {
    }

    1 usage  zakaria-root
    public List<LigneCommande> getLigneCommandes() { return ligneCommandes; }

    zakaria-root
    public void setLigneCommandes(List<LigneCommande> ligneCommandes) { this.ligneCommandes = ligneCommandes; }

    2 usages  zakaria-root
    public int getReference() { return reference; }

    zakaria-root
    public void setReference(int reference) { this.reference = reference; }

    zakaria-root
    public Client getClient() { return client; }

    zakaria-root
    public void setClient(Client client) { this.client = client; }

    zakaria-root
    public Date getDateCommande() { return dateCommande; }

    zakaria-root
    public void setDateCommande(Date dateCommande) { this.dateCommande = dateCommande; }

    zakaria-root
    public String getEtatDeCommande() { return etatDeCommande; }

    zakaria-root
    public void setEtatDeCommande(String etatDeCommande) { this.etatDeCommande = etatDeCommande; }
```

```java
zakaria-root *
@Override
public String toString() {
    StringBuilder str = new StringBuilder();
    int i = 0;
    for (LigneCommande ligneCommande : ligneCommandes) {
        i++;
        str.append("\nl'ordinateur n "+i+":\n*************" + ligneCommande.toString());
    }
    return "\nreference de la commande : " + reference + "    " +
            "\ndate de commande :  " + dateCommande +
            "\netat de commande : " + etatDeCommande +
            "\nles ordinateurs de la commande : "+
            str.toString() + "" + client + '\'';

}
```

⇨ Class LineCommande

- On crée une classe LigneCommande avec les attributs quantité, la commande et l'ordinateur commandé.
- On crée aussi la méthode toString qui nous renvoie une chaîne de caractères représentant les attributs de l'objet.

```java
import com.sun.media.sound.WaveFloatFileReader;

17 usages  zakaria-root
public class LigneCommande {
    4 usages
    private int quantite;
    4 usages
    private Ordinateur ordinateur;
    1 usage
    private  Commande  commande;

    1 usage  zakaria-root
    public int getQuantite() { return quantite; }

    zakaria-root
    public Ordinateur getOrdinateur() { return ordinateur; }

    zakaria-root
    public void setQuantite(int quantite) { this.quantite = quantite; }

    zakaria-root
    public void setOrdinateur(Ordinateur ordinateur) { this.ordinateur = ordinateur; }

    3 usages  zakaria-root
    public LigneCommande(int quantite, Ordinateur ordinateur, Commande commande) {
        this.quantite = quantite;
        this.ordinateur = ordinateur;
        this.commande = commande;
    }
```

```java
zakaria-root
@Override
public String toString() {
    return
            "\nla quantite d'ordinateur : "+quantite +"    "+
           "" + ordinateur ;
}
```

⇨ Class Client

- On crée une classe Client avec les attributs nom, prénom, adresse, email, ville, téléphone, et une liste de commandes effectuées. Ajoutez la méthode
- On définit la méthode **ajouterCommande**() pour ajouter une nouvelle commande à la liste (vous devez vérifier s'elle existe déjà avant de l'ajouter), et une méthode **supprimerCommande**() pour supprimer une commande.

```java
6 usages  zakaria-root *
public class Client {
    4 usages
    private String nom;
    4 usages
    private String prenom;
    4 usages
    private String adresse;
    4 usages
    private String ville;
    4 usages
    private String telephone;
    4 usages
    private String email;
    7 usages
    List<Commande> commandes;
    zakaria-root
    public List<Commande> getCommandes() { return commandes; }
    zakaria-root
    public void setCommandes(List<Commande> commandes) { this.commandes = commandes; }
    zakaria-root
    public String getNom() { return nom; }
    zakaria-root
    public String getPrenom() { return prenom; }
    zakaria-root
    public String getAdresse() { return adresse; }
    zakaria-root
```

```java
zakaria-root
public String getVille() { return ville; }
zakaria-root
public String getTelephone() {
    return telephone;
}
zakaria-root
public String getEmail() { return email; }
zakaria-root
public void setNom(String nom) { this.nom = nom; }
zakaria-root
public void setPrenom(String prenom) {
    this.prenom = prenom;
}
zakaria-root
public void setAdresse(String adresse) { this.adresse = adresse; }
zakaria-root
public void setVille(String ville) { this.ville = ville; }
zakaria-root
public void setTelephone(String telephone) { this.telephone = telephone; }
zakaria-root
public void setEmail(String email) { this.email = email; }
1 usage  zakaria-root *
public Client(String nom, String prenom, String adresse, String ville, String telephone, String email, List<Commande> commandes) {
    this.nom = nom;
    this.prenom = prenom;
    this.adresse = adresse;
    this.ville = ville;
    this.telephone = telephone;
    this.email = email;
    this.commandes = commandes;
```

```java
public void ajouterCommande(Commande commande) {
    Boolean trouve = false;
    for (Commande c : commandes) {
        if (c.getReference() == commande.getReference()) {
            trouve = true;
            return;
        }
    }
    if (trouve) {
        System.out.println("La commande existe déjà");
    } else {
        commandes.add(commande);
    }

}
```

zakaria-root

```java
public void supprimerCommande(Commande commande) {
    if (commandes.contains(commande)) {
        System.out.println("la commande n'existe pas");
    } else {
        commandes.remove(commande);
    }
}
```

zakaria-root *

```java
@Override
public String toString() {
    return "\nles informations de client: \n**************************" +
            "\nnom de client : " + nom +
            "\nprenom de client : " + prenom +
            "\naddress de client : " + adresse +
            "\nville de client : " + ville +
            "\ntelephone de client : " + telephone +
            "\nemail de client : " + email;
}
```

⇨ Class Main

- finalement on crée la classe main() qui contient une méthode **main**() pour tester les différentes

  classes, dans laquelle :

  — On déclare une liste de trois ordinateurs ;

  — On déclare et initialise une catégorie ;

  — On déclare et initialise un client ;

  — On déclare et initialise une commande du client ;

  — On déclare et initialise une liste de trois lignes de commandes pour la commande

  et les ordinateurs créés ;;

  — affichez toutes les informations de la commande.

```
public static void main(String[] args) {

    //initilise la class categorie

//      déclarez et intentiez une liste de trois ordinateurs ;
    List<Ordinateur> ordinateurs = new ArrayList<>( initialCapacity: 3);
    ordinateurs.add(new Ordinateur( nom: "nom1",  marque: "marque1",  prix: 100.0,  description: "description1", new ArrayList<>(),  categorie: null));
    ordinateurs.add(new Ordinateur( nom: "nom2",  marque: "marque2",  prix: 200.0,  description: "description2", new ArrayList<>(),  categorie: null));
    ordinateurs.add(new Ordinateur( nom: "nom3",  marque: "marque3",  prix: 300.0,  description: "description3", new ArrayList<>(),  categorie: null));

//- déclarez et intentiez une catégorie ;
    Categorie categorie = new Categorie( nom: "nom",  description: "description", ordinateurs);
    for (Ordinateur o : ordinateurs) {
        o.setCategorie(categorie);
    }
//- déclarez et intentiez un client ;
    Client client = new Client( nom: "nom",  prenom: "prenom",  adresse: "adresse",  ville: "ville",  telephone: "telephone",  email: "email", new ArrayList<Commande>());
//- déclarez et instanciez une commande du client ;
    Commande commande = commande = new Commande( reference: 1, client, new Date(),  etatDeCommande: "etatDeCommande", new ArrayList<>());
    client.ajouterCommande(commande);

//- déclarez et instanciez une liste de trois lignes de commandes pour la commande et les ordinateurs créés ;
    List<LigneCommande> ligneDeCommandes = new ArrayList<>( initialCapacity: 3);


    LigneCommande ligneDeCommande = new LigneCommande( quantite: 1, ordinateurs.get(0), commande);
    LigneCommande ligneDeCommande1 = new LigneCommande( quantite: 2, ordinateurs.get(1), commande);
    LigneCommande ligneDeCommande2 = new LigneCommande( quantite: 3, ordinateurs.get(2), commande);
    ligneDeCommandes.add(ligneDeCommande);
    ligneDeCommandes.add(ligneDeCommande1);
    ligneDeCommandes.add(ligneDeCommande2);
    commande.getLigneCommandes().addAll(ligneDeCommandes);

    ordinateurs.get(0).getLigneCommandes().add(ligneDeCommande);
    ordinateurs.get(1).getLigneCommandes().add(ligneDeCommande1);
    ordinateurs.get(2).getLigneCommandes().add(ligneDeCommande2);
//- affichez toutes les informations de la commande.

    System.out.println(commande);
}
```

- **L’exécution du code est donné par :**

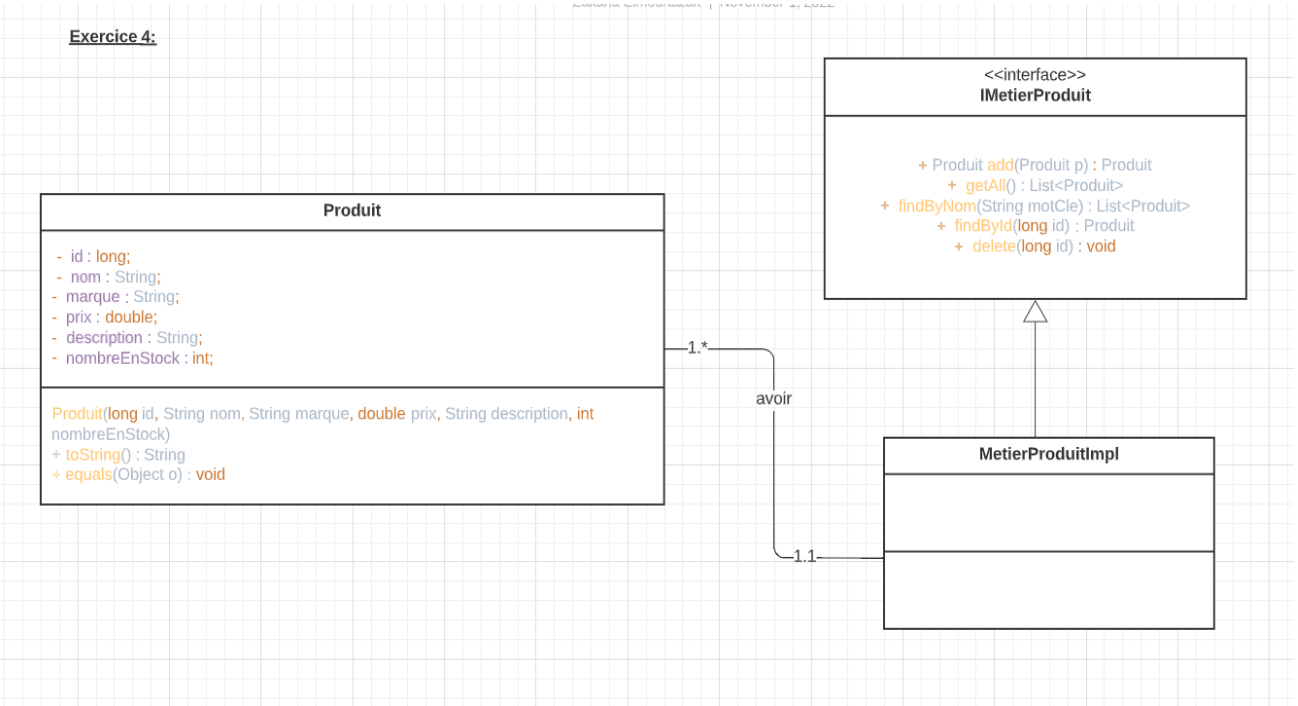
```
reference de la commande : 1
date de commande :  Thu Oct 27 23:44:07 WEST 2022
etat de commande : etatDeCommande
les ordinateurs de la commande :
l'ordinateur n 1:
*************
la quantite d'ordinateur : 1
nom d'ordinateur :nom1
la marque d'ordinateur : marque1
le prix d'ordinateur : 100.0
la description d'ordinateur : description1
le totale d'ordinateur : 100.0
nom du categorie : nom
description du categorie :   description
l'ordinateur n 2:
*************
la quantite d'ordinateur : 2
nom d'ordinateur :nom2
la marque d'ordinateur : marque2
le prix d'ordinateur : 200.0
la description d'ordinateur : description2
le totale d'ordinateur : 400.0
nom du categorie : nom
description du categorie :   description
l'ordinateur n 3:
*************
la quantite d'ordinateur : 3
nom d'ordinateur :nom3
la marque d'ordinateur : marque3
le prix d'ordinateur : 300.0
la description d'ordinateur : description3
le totale d'ordinateur : 900.0
nom du categorie : nom
description du categorie :   description
les informations de client:
**************************
nom de client : nom
prenom de client : prenom
```

Exercice 4:

- On souhaite réaliser une application JAVA dont l'objectif de cet exercice est de manipuler une collection d'objets de type produit en utilisant les listes et les interfaces.

● L'analyse de l'application :

- Premièrement et avant de partir de la partie développement nous schématisons un diagramme de classes afin de mieux comprendre l'aperçu général le schéma de notre application en modélisant ses classes, ses attributs, ses opérations et les relations entre ses objets. avec l'utilisation de site web **lucidchart**



● Développement de l'application :

- Après la partie modélisation du schéma, notre application devient plus claire et on peut commencer la partie développement avec la création des deux packages, un packages **métier** qui va contenir les différentes fonctionnalités de l'application et l'autre package qui est **présentation** qui va contenir l'utilisation de ces fonctionnalités
- on crée maintenant les différentes classes nécessaire de notre application

⇨ Class Produit

- On crée la classe **Produit** avec les attributs id, nom, marque, prix, description, et
- nombre en stock.
- On crée aussi la méthode toString qui nous renvoie une chaîne de caractères représentant les attributs de l'objet.

```java
import java.util.Objects;

22 usages
public class Produit {
    6 usages
    private long id;
    6 usages
    private String nom;
    6 usages
    private String marque;
    6 usages
    private double prix;
    6 usages
    private String description;
    6 usages
    private int nombreEnStock;


    4 usages
    public Produit(long id, String nom, String marque, double prix, String description, int nombreEnStock) {
        this.id = id;
        this.nom = nom;
        this.marque = marque;
        this.prix = prix;
        this.description = description;
        this.nombreEnStock = nombreEnStock;
    }

    public Produit() {
    }
```

```
2 usages
public long getId() { return id; }

public void setId(long id) { this.id = id; }

1 usage
public String getNom() { return nom; }

public void setNom(String nom) { this.nom = nom; }

public String getMarque() { return marque; }

public void setMarque(String marque) { this.marque = marque; }

public double getPrix() { return prix; }

public void setPrix(double prix) { this.prix = prix; }

public String getDescription() { return description; }

public void setDescription(String description) { this.description = description; }

public int getNombreEnStock() { return nombreEnStock; }

public void setNombreEnStock(int nombreEnStock) { this.nombreEnStock = nombreEnStock; }

@Override
public String toString() {
    return "Produit " +
            " id : " + id +
            "\n---------------"+
            "\nnom : '" + nom + '\'' +
            "\nmarque : '" + marque + '\'' +
            "\nprix : " + prix +
            "\ndescription : '" + description + '\'' +
            "\nquatite : " + nombreEnStock
            ;
}
```

- Class IMetierProduit

- On Crée l'Interface **IMetierProduit** qui va déclarer les méthodes pour gérer nos objets
- Produit. Cette interface contient les méthodes suivantes :

  - ☐ public Produit add(Produit p) : qui permet d'ajouter un produit à la liste.
  - ☐ public List<Produit> getAll() : qui retourne les produits sous forme d'une liste.
  - ☐ public list<Produit> findByNom(String motCle) : qui retourne une liste de produits dont le nom contient le mot clé passé en paramètre.
  - ☐ public Produit findById(long id) : qui retourne un produit par id.
  - ☐ public void delete(long id) : qui supprime un produit par id.

```java
import java.util.List;

1 usage  1 implementation
public interface IMetierProduit {
    4 usages  1 implementation
    public Produit add(Produit p);

    3 usages  1 implementation
    public List<Produit> getAll();

    1 usage  1 implementation
    public List<Produit> findByNom(String motCle);

    2 usages  1 implementation
    public Produit findById(long id);

    1 usage  1 implementation
    public void delete(long id);
}
```

- Class MetierProduitImpl

- On crée la classe **MetierProduitImpl** qui implémente l'interface **IMetierProduit** et qui va déclarer comme attribut une liste de produits.
- On redéfinit tout les méthodes de l'interface **IMetierProduit** que nous avons implémenté

```java
import java.util.ArrayList;
import java.util.List;

2 usages
public class MetierProduitImpl implements IMetierProduit {
    9 usages
    private List<Produit> produits;
    1 usage
    public MetierProduitImpl(List<Produit> produits) { this.produits = produits; }
    4 usages
    @Override
    public Produit add(Produit p) {
        if (produits.contains(p)) {
            System.out.println(" le prosuit existe deja");
            return null;
        } else {
            produits.add(p);
            return p;
        }

    }
    3 usages
    @Override
    public List<Produit> getAll() { return produits; }

    1 usage
    @Override
    public List<Produit> findByNom(String motCle) {

        this.produits.forEach(( Produit p) -> {
            if (p.getNom().contains(motCle)) {
                produits.add(p);
            }
        });

        return produits;
    }

    2 usages
    @Override
    public Produit findById(long id) {

        for (Produit p : produits) {
            if (p.getId() == id) {
                return p;
            }
        }
        return null;
    }
    1 usage
    @Override
    public void delete(long id) { produits.removeIf(p -> p.getId() == id); }
```

- ⇨ Class Main

- Finalement on crée la classe main() qui contient une méthode **main**() qui propose à l'utilisateur dans une boucle while le menu suivant :
-

1. Afficher la liste des produits.

2. Rechercher des produits par mot clé.

3. Ajouter un nouveau produit dans la liste.

4. Récupérer et afficher un produit par ID.

5. Supprimer un produit par id.

6. Quitter ce programme.

```
1 usage
private static void menue() {
    System.out.println("-----------------------------------------------");
    System.out.println("menue");
    System.out.println(
            "1. Afficher la liste des produits.\n" +
                    "2. Rechercher des produits par mot clé.\n" +
                    "3. Ajouter un nouveau produit dans la liste.\n" +
                    "4. Récupérer et afficher un produit par ID.\n" +
                    "5. Supprimer un produit par id.\n"+
                    "6. Quitter le programme.\n"
    );
    System.out.println("-----------------------------------------------");
}
```

```java
public class Main {
    11 usages
    private static MetierProduitImpl metierProduit= new MetierProduitImpl(new ArrayList<>());
    11 usages
    private static Scanner sc  = new Scanner(System.in);
    public static void main(String[] args) {
        metierProduit.add(new Produit( id: 1,  nom: "Ordinateur",  marque: "HP",  prix: 1000,  description: "Ordinateur portable",  nombreEnStock: 10));
        metierProduit.add(new Produit( id: 2,  nom: "Imprimante",  marque: "HP",  prix: 200,  description: "Imprimante laser",  nombreEnStock: 10));
        metierProduit.add(new Produit( id: 3,  nom: "Smartphone",  marque: "Samsung",  prix: 500,  description: "Smartphone 4G",  nombreEnStock: 10));

        boolean quit;
        do {

            quit = false;
            menue();

            int choix = sc.nextInt();

            switch (choix) {
                case 1:
                    System.out.println("la liste des produits");
                    if (metierProduit.getAll().isEmpty()) {
                        System.out.println("la liste est vide");
                    } else {
                      metierProduit.getAll().forEach(p -> {
                            System.out.println(p);
                      });
                    }

                    break;
                case 2:
                    System.out.println("Rechercher des produits par mot clé");
                    System.out.println("Entrer le mot clé");
                    sc.nextLine();
                    String motCle = sc.nextLine();
                    System.out.println("la liste des produits de mot clé " + motCle);
                    System.out.println("---------------------------------");
                   metierProduit.findByNom(motCle).forEach(p -> {
                        System.out.println(p);
                    });
```

```java
                    break;
                case 3:
                    System.out.println("Ajouter un nouveau produit dans la liste");
                    System.out.println("---------------------------------");
                    System.out.println("Entrer l'id de produit ");
                    long idProduit = sc.nextLong();
                    System.out.println("Entrer le nom du produit");
                    String nom = sc.next();
                    System.out.println("Entrer la marque du produit");
                    String marque = sc.next();
                    System.out.println("Entrer le prix du produit");
                    double prix = sc.nextDouble();
                    System.out.println("Entrer la description du produit");
                    String description = sc.next();
                    System.out.println("Entrer le nombre en stock du produit");
                    int nombreEnStock = sc.nextInt();
                    Produit p = new Produit(idProduit, nom, marque, prix, description, nombreEnStock);
                    metierProduit.add(p);
                    break;
                case 4:
                    System.out.println("Récupérer et afficher un produit par ID");
                    System.out.println("Entrer l'id du produit");
                    long findId = sc.nextLong();
                    Produit findProduit = metierProduit.findById(findId);
                    if (findProduit == null) {
                        System.out.println("le produit n'existe pas");
                    } else {
                        System.out.println(findProduit);
                    }
                    break;
```

```
                break;
            case 5:
                System.out.println("Supprimer un produit par id");
                System.out.println("Entrer l'id du produit");
                if (metierProduit.getAll().isEmpty()) {
                    System.out.println("la liste est vide");
                } else {
                    long deleteId = sc.nextLong();
                    if (metierProduit.findById(deleteId) == null) {
                        System.out.println("le produit n'existe pas");
                    } else {
                        metierProduit.delete(deleteId);
                    }
                }

                break;
            case 6:
                System.out.println("aurevoir ...");
                quit = true;
                break;
            default:
                System.out.println("Choix invalide");
        }
    } while (quit != true);
```

- **L'exécution du code est donné par :**

```
-----------------------------------------------
menue
1. Afficher la liste des produits.
2. Rechercher des produits par mot clé.
3. Ajouter un nouveau produit dans la liste.
4. Récupérer et afficher un produit par ID.
5. Supprimer un produit par id.
6. Quitter le programme.

-----------------------------------------------
```

```
1
la liste des produits
Produit  id : 1
--------------
nom : 'Ordinateur'
marque : 'HP'
prix : 1000.0
description : 'Ordinateur portable'
quatite : 10
Produit  id : 2
--------------
nom : 'Imprimante'
marque : 'HP'
prix : 200.0
description : 'Imprimante laser'
quatite : 10
Produit  id : 3
--------------
nom : 'Smartphone'
marque : 'Samsung'
prix : 500.0
description : 'Smartphone 4G'
quatite : 10
----------------------------------------------
```

```
--------------------------------------------------
2
Rechercher des produits par mot clé
Entrer le mot clé
ma
la liste des produits de mot clé ma
-----------------------------------
Produit  id : 2
---------------
nom : 'Imprimante'
marque : 'HP'
prix : 200.0
description : 'Imprimante laser'
quatite : 10
Produit  id : 3
---------------
nom : 'Smartphone'
marque : 'Samsung'
prix : 500.0
description : 'Smartphone 4G'
quatite : 10
```

```
Ajouter un nouveau produit dans la liste
--------------------------------
Entrer l'id de produit
5
Entrer le nom du produit
dsad
Entrer la marque du produit
dsa
Entrer le prix du produit
23
Entrer la description du produit
sada
Entrer le nombre en stock du produit
3
------------------------------------------------
------------------------------------------------
4
Récupérer et afficher un produit par ID
Entrer l'id du produit
1
Produit  id : 1
--------------
nom : 'Ordinateur'
marque : 'HP'
prix : 1000.0
description : 'Ordinateur portable'
quatite : 10
------------------------------------------------
------------------------------------------------
5
Supprimer un produit par id
Entrer l'id du produit
1
------------------------------------------------

------------------------------------------------
6
aurevoir ...
```

Fin.