Projet Java IHM

*A faire par un groupe de **2 personnes** à rendre au plus tard le dimanche **23 juin 2024 à 23h59***
*Avec un **compte rendu** don le modèle est fourni dans la pièce joint.*

soaformationtp@gmail.com

TRAVAIL A FAIRE

- **L'interface graphique** sera réalisée avec la librairie **swing** de Javax, comme vu en cours
- **L'exercice 1** de la **partie I** doit être fait avant la **partie II**.
  Faire le diagramme de classe *sur feuille*, ou avec un logiciel(StarUML …).
  Ce diagramme de classe sera complété plus tard en tenant compte de la **partie II**.
- C'est un projet, l'objectif est de faire ressortir les notions vues en cours, toute démarche personnelle en plus sera très appréciée. Préciser bien ce que vous avez fait en plus avec une autre couleur lors de la rédaction du compte rendu.

Merci de rédiger un dossier sous Word contenant :
- La copie de l'énoncé
- Le résultat (outil capture)
- Le code java chacun dans un tableau (voir page 3).

→ Utiliser le modèle ***compteRenduJava_EPMI_Nom_Prenom.doc*** ci-joint

PARTIE – I : UML

1.1. Exercice : Diagramme de classe

Dans un magasin,
Les clients peuvent acheter un ou plusieurs produits. Un produit peut être acheté par au moins 2 clients.
Ces produits peuvent être un produit artisanal appelé **ProduitA** ou un produit industriel appelé **produitB.**
En général, un produit est caractérisé par son nom, sa ville de provenance, son prix unitaire et la quantité.
On remarque que le produit artisanal est caractérisé en plus par sa qualité, celle-ci peut être **bas de gamme** ou **haute gamme**.
Alors qu'un produit industriel, se différencie selon le taux de réduction appliqué.
Toutes ces données sont encapsulées public et déclarées au niveau des classes héritières.

Peu importe le produit considéré, on doit mettre en place le **calcul de prix** total dans une méthode, en tenant compte de la quantité et du prix unitaire. Compte tenu du fait que les calculs changent selon le type de produit, nous allons mettre en place une interface **Produit** qui impose ce calcul de prix total, Les détails de calcul (en java) se feront donc dans les classes héritières d'après ce qu'on vu en cours.
Cette interface ne contient que la méthode **calculPrix**().
La classe correspondant au produit industriel admet en plus la méthode **calculPrixReduit**().

Pour information, un client est caractérisé par son nom, son prénom, son âge et sa ville, toutes ces données sont encapsulées privées

1. En utilisant un logiciel **starUML ou sur feuille…**, représenter la situation décrite ci-dessus par un diagramme de classe.

On mettra en place les notions de : **classe - héritage - encapsulation -stéréotype**

- ➔ On implémentera ces classes en java dans la **partie II**
- ➔ Attention à la règle de nommage

PARTIE – II : JAVA

2. RAPPEL : Boucle for each pour les ArrayList d'objet

```
listObjet.add(new Personne("DUBOIS", 23, "Cergy"));
for(type variable : listObjet){
        System.out.println(variable.getNom()) ;
 }
```

```
import java.util.ArrayList;

public class TestArrayList {

    public static void main(String[] args) {

        // creation de la liste
        ArrayList<Personne> listPersonne = new ArrayList<Personne>();

        // instanciation de personne
        Personne pers1 = new Personne();
        Personne pers2 = new Personne();
        // Ajout des personnes dans la liste
        listPersonne.add(pers1);
        listPersonne.add(pers2);

        for (Personne pers : listPersonne){
             System.out.println("Personne : "+ pers);
    }
}
```

Créer un nouveau projet : **projetFinal**

2.1. Exercice : tableau, interface graphique, constructeur

1. Créer l'interface **Iproduit** et les classes héritières décris dans la partie I,

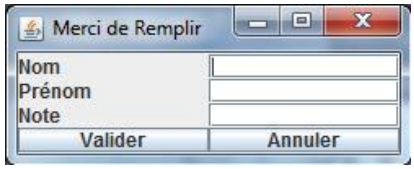
   Les variables sont *publiques* et sont définies dans les classes héritières, en effet une interface permet juste de déclarer les méthodes et éventuellement définir une constante.

   On mettra en place au niveau des classes héritières des *constructeurs* avec et sans paramètres afin de rajouter un produit.

   Sans écrire le constructeur avec paramètre on peut faire ceci :

   `Clic droit dans le code > source > Generate constructor using field`

   - Attention, *commenter* les codes générés



2. a. Mettre en place **une interface graphique** (voir TP →)
   qui permet de rajouter un produit
   dans un tableau (voir l'exemple ci-dessous).
   On utilisera un *constructeur* avec paramètre.

   b. Afficher les caractéristiques du produit que vous venez d'ajouter

   c. Afficher le prix correspondant

→ ICI C'EST **JUSTE UN EXEMPLE**, L'INTERFACE GRAPHIQUE SERA FAIT A VOTRE CONVENANCE

Exemple de valeur ProduitA

Nom	Ville	PrixUnitaire	Quantité	Qualité
PC	Paris	700	2	Haute de gamme

Exemple de valeur ProduitB

Nom	Ville	PrixUnitaire	Quantité	TauxReduction
PC	Paris	500	3	5%

```
//Création de l'interface graphique: button1 ...

    button1.addActionListener(new ActionListener( ){

        public void actionPerformed(ActionEvent evnt){
            //getText()-parseInt()-rajout de produit- saisie par user dans la liste-affiche du résultat

        }
    }
);
```

2.2. Exercice : Client, Arraylist d'objets, CRUD

Nous allons mettre en place le pattern DAO (D : Data, A : Access, O : Object)
C'est une bonne pratique, qui permet de bien séparer chaque couche (métier – dao - test)

0. Rajouter dans votre diagramme de classe, la classe **ClientDao** qui contient les 3 méthodes, précisée ci-dessous
1. Créer dans le **projetFinal** 3 packages : **metier, dao, test**
2. Le package **metier** contiendra la couche la plus basse à savoir la classe *metier* : **Client** décrite dans **la partie** I.

Les variables sont encapsulées **private**, penser à mettre en place les getters et les setters en effectuant l'opération ci-dessous :

Clic droit dans le code > source > Generate Getters an setters

3. Le package **dao** permet de faire des traitements : Mettons en place le **CRUD** (Create Read Update Delete**)**

   a. Créer la classe **ClientDao** avec les 2 méthodes ci-dessous :
   - **createClient( ) :** *enregistrement de client dans un tableau dynamique*
     Cette une méthode sans paramètre retourne un arraylist de type **Client,**
     Cette méthode permet d'insérer le client qu'on va demander par saisie utilisateur dont les valeurs seront récupérées grâce au *getter* et *setter* de la classe **Client.**
   - **getClientParMC( ) :** recherche de client par nom (par exemple **DUBOIS**) Cette méthode admet 2 *paramètres* le mot clé **mc**, et une **liste de clients**. Cette méthode *retourne* le 1er client dont le nom commence par le mot clé, ceci dans un objet de type **Client**.
   - **deleteClientParMC( ) :** liste de clients qui ne sont pas de la famille **DUBOIS**
     Cette méthode admet 2 *paramètres* le mot clé **mc**, et une **liste de clients**, cette méthode *retourne* un arraylist de type **Client** qui contient une liste restante sans les clients supprimés.

4. Le package **test** contient la classe principale qui Va faire appel aux méthodes de la classe **ClientDao**
   - Créer le tableau dynamique de client
   - Demander à l'utilisateur le mot clé recherché afin de lister les clients dont le nom est par exemple **DUBOIS**
   - Lister les clients qui ne sont pas de la famille **DUBOIS**
     La liste de client après modification sera affichée.

   On mettra en place la boucle **for each** qui est pratique pour les arraylist de type objet.