



Séries de TDs/TPs N° : (4 &5)

-prépare par : BENHAMMOU MOHAMED (IDAI)

-encadré par: PR. El Mokhtar EN-NAIMI



<https://github.com/mohamedben12369/JAVA-interfaces-classes>

Dans ce lien GitHub, vous trouverez les codes sources des TP 5 et 6.

PARTIE I : CLASSES ABSTRAITES

Exercice 1 :

les classes du système de gestion des personnes dans un établissement:

1. Classe Personne (classe abstraite)

Description:

La classe Personne est une classe abstraite qui sert de base pour représenter les différentes catégories de personnes dans un établissement (secrétaires, enseignants, étudiants). Elle contient les attributs et méthodes communs à toutes les personnes.

Attributs:

- **nom (protected)** : Nom de la personne.
- **prenom (protected)** : Prénom de la personne.
- **rue (protected)** : Rue de l'adresse de la personne.
- **ville (protected)** : Ville de l'adresse de la personne.
- **nbPersonnes (static, protected)** : Variable de classe qui compte le nombre total de personnes créées.

Méthodes:

- **Personne(String nom, String prenom, String rue, String ville)** : Constructeur qui initialise les attributs d'une personne.
- **toString()** : Retourne une chaîne de caractères contenant les informations de la personne.
- **abstract void ecrirePersonne()** : Méthode abstraite à implémenter dans les sous-classes pour afficher les détails d'une personne.
- **static void nbPersonne()** : Affiche le nombre total de personnes créées.
- **void modifierPersonne(String rue, String ville)** : Modifie l'adresse de la personne et affiche les nouvelles informations.

2. Classe Secrétaire (sous-classe de Personne)

Description:

La classe Secrétaire représente une secrétaire dans l'établissement. Elle hérite des attributs et méthodes de la classe Personne et ajoute un attribut spécifique : le numéro de bureau.

Attributs:

- **numeroBureau (private)** : Numéro de bureau de la secrétaire.
- **nbSecretaires (static, private)** : Variable de classe qui compte le nombre de secrétaires créées.

Méthodes:

- **Secrétaire(String nom, String prenom, String rue, String ville, String numeroBureau)** : Constructeur qui initialise les attributs d'une secrétaire.
- **toString()** : Retourne une chaîne de caractères contenant les informations de la secrétaire.
- **void ecrirePersonne()** : Affiche les informations de la secrétaire.
- **static void nbSecretaires()** : Affiche le nombre total de secrétaires créées.

3. Classe Enseignant (sous-classe de Personne)

Description:

La classe Enseignant représente un enseignant dans l'établissement. Elle hérite des attributs et méthodes de la classe Personne et ajoute un attribut spécifique : la spécialité de l'enseignant.

Attributs:

- **specialite (private)** : Spécialité de l'enseignant (exemple : informatique, mathématiques).
- **nbEnseignants (static, private)** : Variable de classe qui compte le nombre d'enseignants créés.

Méthodes:

- **Enseignant(String nom, String prenom, String rue, String ville, String specialite)** : Constructeur qui initialise les attributs d'un enseignant.

- **toString()** : Retourne une chaîne de caractères contenant les informations de l'enseignant.
- **void écrirePersonne()** : Affiche les informations de l'enseignant.
- **static void nbEnseignants()** : Affiche le nombre total d'enseignants créés.

4. Classe Etudiant (sous-classe de Personne)

Description:

La classe Etudiant représente un étudiant dans l'établissement. Elle hérite des attributs et méthodes de la classe Personne et ajoute un attribut spécifique : le diplôme en cours de l'étudiant.

Attributs:

- **diplomeEnCours(private)** : Diplôme que l'étudiant est en train de préparer.
- **nbEtudiants(static, private)** : Variable de classe qui compte le nombre d'étudiants créés.

Méthodes:

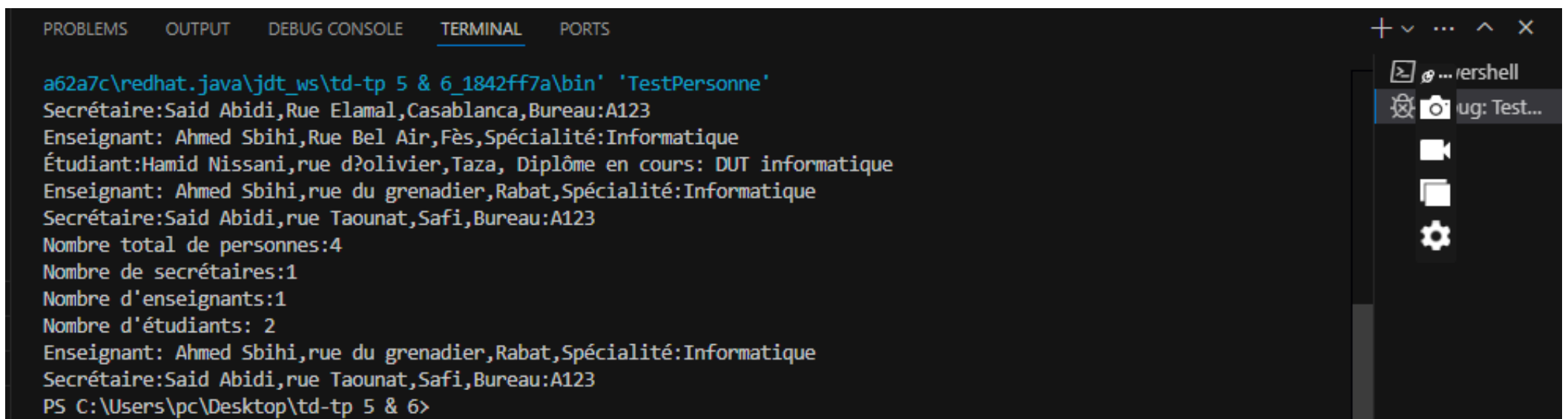
- **Etudiant(String nom,String prenom,String rue,String ville, String diplomeEnCours)** : Constructeur qui initialise les attributs d'un étudiant.
- **toString()** : Retourne une chaîne de caractères contenant les informations de l'étudiant.
- **void écrirePersonne()** : Affiche les informations de l'étudiant.
- **static void nbEtudiants()** : Affiche le nombre total d'étudiants créés.

5. Classe TestPersonne (programme principal)Description

La classe TestPersonne contient le point d'entrée du programme. Elle permet de tester les différentes classes et méthodes définies dans le système.

Fonctionnalités:

- Création d'un objet Secrétaire et affichage de ses informations avec écrirePersonne().
- Création d'un objet Enseignant et affichage de ses informations avec écrirePersonne().
- Création de deux objets Etudiant et affichage des informations du deuxième étudiant avec écrirePersonne().
- Modification de l'adresse de l'enseignant et du secrétaire, puis affichage des nouvelles informations.
- Affichage du nombre total de personnes et du nombre de chaque catégorie (secrétaires, enseignants, étudiants).
- Création d'autres objets pour tester les méthodes des différentes classes.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
a62a7c\redhat.java\jdt_ws\td-tp 5 & 6_1842ff7a\bin' 'TestPersonne'
Secrétaire:Said Abidi,Rue Elamal,Casablanca,Bureau:A123
Enseignant: Ahmed Sbihi,Rue Bel Air,Fès,Spécialité:Informatique
Étudiant:Hamid Nissani,rue d'olivier,Taza, Diplôme en cours: DUT informatique
Enseignant: Ahmed Sbihi,rue du grenadier,Rabat,Spécialité:Informatique
Secrétaire:Said Abidi,rue Taounat,Safi,Bureau:A123
Nombre total de personnes:4
Nombre de secrétaires:1
Nombre d'enseignants:1
Nombre d'étudiants: 2
Enseignant: Ahmed Sbihi,rue du grenadier,Rabat,Spécialité:Informatique
Secrétaire:Said Abidi,rue Taounat,Safi,Bureau:A123
PS C:\Users\pc\Desktop\td-tp 5 & 6>
```

Conclusion:

Ce système permet de gérer les différentes catégories de personnes dans un établissement d'enseignement en utilisant la programmation orientée objet. Chaque classe représente une catégorie spécifique avec ses propres attributs et méthodes. Le programme principal TestPersonne permet de tester les fonctionnalités du système.

Exercice 2 :

1. ObjetGraphique.java

Description : Le fichier ObjetGraphique.java contient la définition de la classe abstraite ObjetGraphique, qui sert de base à toutes les formes géométriques dans ce projet. Cette classe définit les attributs de position (x, y) ainsi que des méthodes pour déplacer les objets et les afficher à l'écran.

- **Attributs :** x et y sont des entiers représentant la position de l'objet dans un espace bidimensionnel.
- **Constructeurs:** La classe propose trois constructeurs :
 1. Un constructeur par défaut initialisant x et y à 0.
 2. Un constructeur qui prend des valeurs spécifiques pour x et y.
 3. Un constructeur de copie pour créer une copie d'un objet existant.
- **Méthodes:**
 - **deplacer(int dx, int dy) :** Modifie la position de l'objet en déplaçant sa position de dx et dy.
 - **setPosition(int x, int y) :** Définit une nouvelle position pour l'objet.
 - **affiche() :** Méthode abstraite qui doit être implémentée par les sous-classes pour afficher les informations spécifiques à l'objet.

2. Rectangle.java:

Description : Le fichier Rectangle.java contient la classe Rectangle, qui hérite de la classe abstraite ObjetGraphique. Cette classe représente un rectangle et ajoute des attributs pour la largeur et la hauteur du rectangle.

- **Attributs :** largeur et hauteur représentent respectivement la largeur et la hauteur du rectangle.
- **Constructeurs:**
 1. Le constructeur par défaut initialise la position à (0, 0) et la taille à 0.0.
 2. Le constructeur avec des paramètres permet de définir la position, la largeur et la hauteur.

3. Le constructeur de copie permet de créer une copie d'un rectangle **existant**.

- **Méthodes:**

- `setObjetGraphique(int x, int y, double largeur, double hauteur)` : Permet de définir la position et les dimensions du rectangle.
- `setTaille(double largeur, double hauteur)` : Modifie uniquement la taille du rectangle.
- `affiche()` : Implémentation de la méthode abstraite `affiche()` qui affiche les informations spécifiques du rectangle, incluant sa position, largeur et hauteur.

3. Bouton.java:

Description : Le fichier `Bouton.java` contient la classe `Bouton`, qui est une spécialisation de la classe `Rectangle`. Elle ajoute un attribut supplémentaire, le texte affiché sur le bouton, et l'implémentation spécifique de la méthode `affiche()`.

- **Attribut** : `text` est une chaîne de caractères qui représente le texte affiché sur le bouton.

- **Constructeurs:**

1. Le constructeur par défaut initialise le texte à une chaîne vide.
2. Le constructeur avec paramètres permet de définir la position, la taille et le texte du bouton.
3. Le constructeur de copie permet de créer une copie d'un bouton existant.

- **Méthodes:**

- `setText(String text)` : Permet de définir le texte du bouton.
- `affiche()` : Implémentation de la méthode `affiche()` qui affiche la position, la taille et le texte du bouton.

4. Cercle.java

Description : Le fichier `Cercle.java` contient la classe `Cercle`, qui est également une spécialisation de la classe `ObjetGraphique`. Elle représente un cercle et ajoute un attribut pour son rayon.

- **Attribut** : `rayon` représente le rayon du cercle.

- **Constructeurs:**

- Le constructeur par défaut initialise le rayon à 0.0.
- Le constructeur avec paramètres permet de définir la position et le rayon du cercle.
- Le constructeur de copie permet de créer une copie d'un cercle existant.

- **Méthodes:**

- `setRayon(double rayon)` : Permet de définir le rayon du cercle.
- `affiche()` : Implémentation de la méthode `affiche()` qui affiche la position et le rayon du cercle.

5. ListeObjetGraphique.java

Description : Le fichier `ListeObjetGraphique.java` contient la classe `ListeObjetGraphique`, qui est responsable de la gestion d'une collection d'objets `ObjetGraphique`. Cette classe permet d'ajouter des objets graphiques à une liste et de les afficher.

- **Attribut** : `liste` est une liste d'objets `ObjetGraphique`.

- Constructeur : Le constructeur initialise la liste vide.
- Méthodes:
 - ajouter(ObjetGraphique obj) : Ajoute un objet graphique à la liste.
 - afficheObjetsGraphique() : Affiche tous les objets graphiques dans la liste en appelant leur méthode affiche().

6. Test.java

Description : Le fichier Test.java est le programme principal qui permet de tester les classes définies dans le projet. Il crée une liste d'objets graphiques, ajoute des objets de type Rectangle, Bouton et Cercle, puis affiche les objets avant et après les déplacements.

- **Méthodes:**
 - Création d'objets : Des objets de type Rectangle, Bouton et Cercle sont créés et ajoutés à la liste d'objets.
 - Affichage avant et après déplacement : Le programme affiche les objets graphiques avant et après un déplacement, en utilisant la méthode déplacer() sur chaque objet.

Cela permet de vérifier que les objets graphiques sont correctement déplacés et leurs informations mises à jour.

```
Avant déplacement :
Rectangle:Position(0,0),Largeur:10.0,Hauteur:20.0
Bouton: Position (5,5),Largeur:15.0,Hauteur:25.0,Text:OK
Cercle:Position(10,10),Rayon:5.0

Après déplacement :
Rectangle:Position(10,10),Largeur:10.0,Hauteur:20.0
Bouton: Position (0,0),Largeur:15.0,Hauteur:25.0,Text:OK
Cercle:Position(13,13),Rayon:5.0
PS C:\Users\pc\Desktop\fd> █
```

Conclusion: Cet exercice illustre bien l'utilisation de l'héritage, des méthodes abstraites et des collections en Java. Les classes Rectangle, Cercle, et Bouton hébergent des caractéristiques spécifiques tout en réutilisant des fonctionnalités définies dans la classe abstraite ObjetGraphique. La gestion de ces objets dans une liste permet de manipuler facilement plusieurs objets graphiques simultanément.

En ajoutant un programme de test, ce projet démontre l'interaction entre ces différentes classes et leur capacité à être manipulées de manière flexible et évolutive.

PARTIE II:INTERFACE

Exercice 3:

1.Interface VendableKilo.java

Objectif :

Cette interface est destinée à représenter les produits qui sont vendus au poids (par kilogramme).

Méthodes :

- **vendre(double quantite)** : Cette méthode reçoit la quantité vendue (en kilogrammes) et retourne le revenu généré par la vente tout en mettant à jour le stock du produit.

2.Interface VendablePiece.java

Objectif :

Cette interface est pour les produits vendus par pièces (et non par poids).

Méthodes :

- **vendre(int quantite)** : Cette méthode reçoit la quantité vendue (en pièces) et retourne le revenu généré par la vente tout en modifiant le stock.

3. Interface Solde.java:

Objectif :

L'interface Solde permet de gérer les produits qui peuvent être vendus avec des réductions, c'est-à-dire ceux qui peuvent voir leur prix modifié par un pourcentage.

Méthodes :

- **lanceSolde(double pourcentage)** : Cette méthode applique une réduction au produit en réduisant son prix du pourcentage donné.
- **termineSolde(double pourcentage)** : Cette méthode annule la réduction en augmentant le prix du produit du pourcentage donné.

4. Classe Article.java

Objectif :

La classe Article est une classe de base pour tous les articles dans le magasin. Elle représente un produit avec ses caractéristiques de prix et de fournisseur.

Attributs:

- **prixAchat** : Prix d'achat de l'article.
- **prixVente** : Prix auquel l'article est vendu.
- **nom** : Nom du produit.
- **fournisseur** : Nom du fournisseur de l'article.

Méthodes :

- **rendement()** : Calcule le rendement, c'est-à-dire la rentabilité, en fonction du prix d'achat et du prix de vente.
- **toString()** : Retourne une représentation sous forme de chaîne de caractères de l'article, incluant ses prix, son nom, son fournisseur et son rendement.

5. Classe ArticleElectromenager.java

Objectif :

Cette classe représente un type d'article spécifique (les articles électroménagers), et elle hérite de la classe Article. Elle permet d'ajouter des articles au stock et de gérer des produits électroménagers.

Attributs:

- **stock** : Nombre d'unités d'articles en stock.

Méthodes :

- **remplirStock(int quantité)** : Permet de remplir le stock en ajoutant une certaine quantité de produits.
- **toString()** : Retourne une chaîne représentant les informations sur l'article, y compris son stock.

Elle implémente l'interface VendablePiece pour les articles qui peuvent être vendus en pièces.

6. Classe ArticlePrimeur.java

Objectif :

Cette classe représente un autre type d'article spécifique, celui des produits primeurs (fruits, légumes, etc.). Elle hérite également de la classe Article.

Attributs::

- **stock** : Quantité d'articles en stock.

Méthodes :

- **remplirStock(int quantité)** : Permet de remplir le stock avec une quantité donnée de produits.
- **toString()** : Retourne une chaîne représentant les informations sur l'article, y compris son stock.

Cette classe implémente l'interface VendableKilo car les produits primeurs sont souvent vendus par kilogramme. Les produits primeurs ne peuvent pas être vendus en solde, donc cette classe ne les implémente pas l'interface Solde.

7. Classe Magasin.java

Objectif :

Cette classe représente un magasin qui contient un stock d'articles (électroménagers et primeurs). Elle est responsable de la gestion du revenu, des dépenses, et du calcul du rendement du magasin.

Attributs:

- **depense** : Le coût d'achat des produits pour le magasin.
- **revenu** : Le revenu généré par les ventes des produits.
- **articlesElectromenager** : Tableau contenant les articles électroménagers.

- **articlesPrimeur** : Tableau contenant les articles primeurs.

Méthodes :

- **ajouterArticleElectromenager()** : Permet d'ajouter un article électroménager au magasin à un index donné.
- **ajouterArticlePrimeur()** : Permet d'ajouter un article primeur au magasin à un index donné.
- **rendement()** : Calcule le rendement du magasin basé sur les revenus et les dépenses.
- **toString()** : Retourne une chaîne détaillant l'état du magasin, incluant les dépenses, les revenus, le rendement, et la liste des articles.

Conclusion:

Ce système de gestion de stock utilise des interfaces pour définir les comportements spécifiques des produits (comme la vente par kilo ou par pièce) et des classes pour représenter des types d'articles (comme les articles électroménagers et primeurs). Le magasin gère ces produits, calculant les dépenses, les revenus, et offrant des méthodes pour ajouter des articles, vendre des produits, et appliquer des soldes lorsque c'est applicable.

Ce projet vous aide à comprendre comment utiliser les interfaces en Java pour modulariser les comportements des classes, et comment structurer une application orientée objet pour gérer des produits dans un magasin.

```
Magasin:  
Dépenses:0.0  
Revenu: 0.0  
Rendement:NaN%  
Articles électroménagers:  
Article:Téléviseur  
Fournisseur:Sony  
Prix d'achat:200.0  
Prix de vente: 300.0  
Rendement:50.0%  
Stock:10
```

```
Rendement:50.0%  
Stock:10  
Article:Réfrigérateur  
Fournisseur:LG  
Prix d'achat:400.0  
Prix de vente: 600.0  
Rendement:50.0%  
Stock:5  
Articles primeurs:  
Article:Pommes  
Fournisseur:Producteur local  
Prix d'achat:1.0
```

Prix d'achat:1.0
Prix de vente: 2.0
Rendement:100.0%
Stock:50.0kg
Article:Carottes
Fournisseur:Producteur local
Prix d'achat:0.8
Prix de vente: 1.5
Rendement:87.49999999999999%
Stock:30.0kg

Après ventes:

Après ventes:
Magasin:
Dépenses:0.0
Revenu: 640.0
Rendement:Infinity%
Articles électroménagers:
Article:Téléviseur
Fournisseur:Sony
Prix d'achat:200.0
Prix de vente: 300.0
Rendement:50.0%
Stock:8
Article:Réfrigérateur

Article:Réfrigérateur
Fournisseur:LG
Prix d'achat:400.0
Prix de vente: 600.0
Rendement:50.0%
Stock:5
Articles primeurs:
Article:Pommes
Fournisseur:Producteur local
Prix d'achat:1.0
Prix de vente: 2.0
Rendement:100.0%
Stock:30.0kg

Prix de vente: 2.0
Rendement:100.0%
Stock:30.0kg
Article:Carottes
Fournisseur:Producteur local
Prix d'achat:0.8
Prix de vente: 1.5
Rendement:87.49999999999999%
Stock:30.0kg

PS C:\Users\pc\Desktop\partie 2>