TD sur l'héritage/ Abstract / Polymorphisme /Interface

Exercice 1 :

1. En considérant la signature de la méthode public double moyenne(double note_examen, int coefficient), indiquer les définitions qui permettent de surcharger correctement cette méthode.

   a) public double moyenne(double note_tp, int coefficient)
   b) public double moyenne(int coefficient, double note_exmn, int coefficient1, double note_ds)
   c) protected int moyenne(double note_tp, int coefficient)
   d) public double moy(double note_examen, int coefficient)
   e) public double moyenne(String cours, double note_examen, int coefficient)
   f) public double moyenne(int coefficient, double note_examen)

2. Trouver la phrase qui n'est pas une caractérisation correcte de polymorphisme :

   a) Le Polymorphisme est un mécanisme consistant à rassembler les données et les méthodes au sein d'une structure en cachant l'implémentation de l'objet.
   b) Le Polymorphisme signifie que la même opération peut se comporter différemment sur différentes classes de la hiérarchie.
   c) Le Polymorphisme offre la possibilité à plusieurs objets de natures différentes d'exposer une interface identique au système, et ainsi répondre à un même message d'une manière qui leur est propre
   d) Le Polymorphisme consiste à autoriser le même code à être utilisé avec différents types, ce qui permet des implémentations plus abstraites et générales.

3. Laquelle des opérations ci-dessous est interdite en Java ?

   a) Le Upcasting implicite
   b) Le Upcasting explicite
   c) Le Downcasting implicite
   d) Le Downcasting explicite

4. Etant donné que la classe Chien (classe fille) étend la classe Animal (classe mère), trouvez la(les) lignes correctes parmi les suivantes et justifiez votre réponse :

   a) Animal y =new Animal(); Chien x= (Chien)y; Animal z=x;
   b) Chien x= new Chien(); Animal y = x; Chien z=(Chien)y
   c) Chien x= new Chien(); Animal y = x; Chien z=y;
   d) Animal y =new Animal(); Chien x= (Chien)y; Animal z=(Animal)x;

5. Quelle classe n'a pas de classe mère ?

a) Orpheline
b) String
c) Object
d) Exception

6. Qu'est-ce qui est faux pour les interfaces ?

a) Une Interface peut être le type d'une référence
b) Une Interface déclare des méthodes sans les implémenter
c) Une Interface peut être implémentée
d) Les attributs d'une interface sont implicitement Public, final
e) Une classe implémentant une interface doit implémenter au moins une méthode d'interface
f) Une Interface peut être instanciée
g) Une interface peut hériter de plusieurs interfaces

7. Qu'est-ce qui est vrai pour les classes abstraites?

a) Une classe abstraite contient au moins une méthode abstraite
b) On ne peut pas instancier une classe abstraite.
c) Une classe abstraite n'a jamais de constructeur
d) Une classe abstraite ne peut pas avoir des classes filles
e) Une classe fille d'une classe abstraite est forcément abstraite

8. Static permet de:
a) Définir des classes statiques
b) Partager des variables de classe entre toutes les instances de classe
c) Définir des méthodes d'instance
d) D'appeler des variables de classes sans créer des objets

Exercice 2 :

Vous êtes chargé de développer un système de gestion des employés pour une entreprise. Chaque employé a un salaire, mais la manière dont le salaire est calculé peut varier en fonction du type d'employé. Vous devez donc créer une hiérarchie de classes pour gérer les employés avec différents types de contrats.

1. Créer une classe `Employe` qui définira les propriétés communes de tous les employés (nom, prénom, identifiant) et une méthode `calculerSalaire()`,`toString()`.

2. Créer des sous-classes de la classe `Employe` pour gérer différents types d'employés.
3. Implémenter la méthode `calculerSalaire()` dans ces sous-classes pour déterminer comment le salaire est calculé en fonction du type d'employé.

Détails des classes :

1. Classe Employe :

- **Attribut :** Nom,identifiant
- **Méthodes :**

– Un constructeur pour initialiser les attributs.
– Une méthode `calculerSalaire()` qui sera implémentée par toutes les sous-classes.
– Une méthode `toString()` pour retourner une chaine de caractère décrivant un employé :

```
Nom – Identifiant - Salaire
```

2. Classe EmployePermanent: La classe **EmployePermanent** représente un employé ayant un contrat à durée indéterminée avec l'entreprise. Un employé permanent bénéficie d'un salaire fixe mensuel.

- **Attribut** : double salaireFixe (salaire mensuel fixe).
- **Méthodes** :
  – Constructeur
  – ...

3. Classe EmployeContractuel : La classe **EmployeContractuel** représente un employé ayant un contrat à durée déterminée avec l'entreprise. L'employé contractuel est payé en fonction du nombre d'heures qu'il a travaillées.Le salaire est égale au produit du tauxHoraire par le nombre d'heures de travail.

- **Attribut** : double tauxHoraire (taux horaire) et int heuresTravaillees (nombre d'heures travaillées).
- **Méthodes** :

– Constructeur
– toString()
– ...

4. Classe EmployeCommercial Représente un employé commercial qui est payé en fonction de son pourcentage sur les ventes réalisées. Le salaire d'un employé commercial est le salaire de base auquel on ajoute la commission. La commission est calculée comme un pourcentage des ventes effectuées par l'employé. Par exemple, un commercial qui réalise 10 000 de ventes avec un pourcentage de 5 % touchera 500 en commission.

- **Attribut** : `double pourcentageVentes` (pourcentage du chiffre d'affaires généré par l'employé) et `double ventesEffectuees` (montant des ventes réalisées par l'employé), `double SalaireDeBase` (le salaire de base de l'employé).
- **Méthodes** :

– Double commission : retourne la commission touchée par l'employé.
– Constructeur
– toString()
– ...

5. Dans la méthode **main**, créez un tableau **Employe[]** et remplissez-le avec des objets des différentes sous-classes, puis affichez les détails de chaque employé à l'aide de la méthode toString().

6. Définir l'interface **TypeEmploye** contenant la Méthode : *afficherTypeEmploye()*. Cette méthode doit être implémentée par les 3 classes d'employé pour afficher le type d'employé (par exemple : "Employé Permanent", "Employé Contractuel", "Employé Commercial", etc.).

7. Dans cette question, vous devez modifier la solution de la question précédente pour que l'interface TypeEmploye soit implémentée dans la classe Employe, et non dans les sous-classes. Ainsi, la classe Employe devra contenir l'implémentation de la méthode afficherTypeEmploye(), qui Affichera le type de l'employé.

Exercice 3 :

On vous demande de développer une petite application en *Java* pour assurer la gestion d'une agencede location immobilière. Les immobiliers à louer peuvent être des appartements ou bien desmaisons.

Interface Louable et Exception

On définit l'interface ***louable*** comme suit :

public interface louable{

public void louer() throws ImmobilierReserveException;

public void liberer() ;}

Cette interface gère les objets qui peuvent être loués : appartement, maison, etc. Les deux

méthodes mettent à jour l'***état*** de l'objet à louer ou à libérer. L'exception *ImmobilierReserveException* se déclenche si l'objet est déjà loué.

1) Définir la classe ***ImmobilierReserveException.***

Classe Personne

1) Définir une classe ***Personne***, possédant des attributs ***nom_prenom*** et ***adresse*** de type ***String***,***contact*** de type ***int***. L'attribut ***nom_prenom***, une fois initialisé, ne peux plus changer de valeur.
2) Ecrire un constructeur avec trois paramètres permettant d'initialiser tous les attributs de la classe ***Personne***.
3) Ecrire les accesseurs nécessaires pour tous les attributs de la classe et les modificateurs des attributs ***contact*** et ***adresse***.
4) Ecrire la méthode ***toString*** permettant de retourner une chaîne décrivant une ***Personne***.

On suppose par la suite que tous les accesseurs « getNomAttribut() » et les modificateurs « setNomAttribut() » des classes ci-dessous (BienImmobilier, Maison, Appartement) sont déjà définis.

Classe BienImmobilier

1) Définir une classe ***BienImmobilier***, ***non instanciable***, possédant les attributs ***code*** de type ***int***, ***ville*** de type ***String***, ***surface*** de type ***double***, ***libre*** de type ***boolean*** décrivant l'état de l'immobilier indiquant si l'immobilier est libre ou loué. Définir aussi un l'attribut ***proprietaire*** de type ***personne*** indiquant le propriétaire du bien immobilier.
2) Ecrire un constructeur avec cinq paramètres permettant de créer un nouveau ***BienImmobilier*** en initialisant tous les attributs de la classe.
3) Ecrire un autre constructeur avec quatre paramètres permettant de créer un nouveau ***BienImmobilier*** libre (non loué). Ce constructeur appelle le constructeur avec cinq paramètres.
4) Définir également la méthode ***toString***.

N.B Tous les ***BienImmobilier*** sont **louables.**

Classe maison

1) Sachant qu'une maison est un bien immobilier, définir, une classe ***maison*** possédant trois attributs supplémentaires ***Rue*** de type ***String*** ; ***num_maison*** de type ***int*** et ***nbpieces*** de type ***int*** .
2) Ecrire un constructeur à sept paramètres permettant d'initialiser une instance de la classe ***Maison***. Une maison nouvellement créée est libre (non louée).
3) Ecrire la méthode ***toString*** permettant de retourner une chaîne décrivant une maison.

Classe Appartement

1) Sachant qu'un appartement est un bien immobilier, définir, une classe ***appartement*** possédant trois attributs supplémentaires ***Residence*** de type ***String, etage*** et ***num_app*** de type ***int***.
2) Ecrire un constructeur à sept paramètres permettant d'initialiser une instance de la classe ***Appartement***. Un appartement nouvellement créé est libre (non louée).
3) Définir également la méthode ***toString***.

Classe Agence

1) Créer la classe ***Agence*** assurant la gestion des biens immobiliers. La classe a trois attributs ***Adresse*** de type ***String***, un attribut ***Gerant*** de type ***personne*** et un attribut ***Tab***, un tableau polymorphe de ***BienImmobilier*** afin de ranger tous les biens immobiliers de l'agence.
2) Définir un constructeur avec deux paramètres pour créer le tableau ***Tab*** de ***10 BienImmobilier*** et pour initialiser l'attribut ***Adresse*** et ***Gerant***.

3) Définir la méthode ***toString*** permettant de décrire l'agence : ***Gerant***, l'***adresse*** et tous les ***BienImmobilier*** du tableau ***Tab.***
4) Définir la méthode ***boolean exist(BienImmobilier J)*** recherche le ***BienImmobilier J*** dans le tableau ***Tab.*** La recherche se fait avec l'attribut ***code***.
5) Définir la méthode ***ajout (BienImmobilier B)*** permettant l'ajout d'un ***BienImmobilier B*** dans le tableau.

Classe TestAgence

Ecrire une classe de test ***TestAgence*** avec la méthode principale ***main :***

1) Créer cinq personne ***p1,p2,p3,p4*** et ***p5*** .
2) Créer une Agence ***Agence_Zaghouan*** à *Zaghouan.* Le gérant de l'agence est ***p1***.
3) Définir quatre objets polymorphes de la classe ***BienImmobilier A1 ,A2 ,A3 ,M1*** avec***A1 ,A2 ,A3*** sont des appartement et ***M1*** une maison (choisissez vous-même les villes, etage, rue, nbpieces, Etc.). Les propriétaire de ***A1 ,A2 ,A3*** et ***M1*** sont respectivement ***p2, p3, p4 et p5***.
4) ***A1*** et ***M1*** sont loués. Faites les mises à jour nécessaires. N'oublier pas de gérer les exceptions personnalisées.
5) Ajouter ***A1 , A2 ,A3*** et ***M1*** à l'agence ***Agence_Zaghouan***.
6) Afficher toutes les informations relatives à ***Agence_Zaghouan*** : gérant, adresse et tous les biens immobiliers.