Numéro d'anonymat :

Examen terminal

Tous documents sur support papier autorisés. Durée : 2h.
L'ensemble des réponses sera à donner sur les feuilles d'énoncé. Ne pas dégrapher les feuilles.

Ce sujet développe quelques éléments d'un logiciel destiné à gérer des factures.
On vous donne les interfaces `IClient`, `IFournisseur` et `Item` suivantes.

```
public interface IClient {String getNom();}
public interface IFournisseur {String getNom();}
public interface Item {String getNom(); double prixUnitaire();}
```

Question 1. Ecrivez en Java une interface `IFacture`. A ce stade, une facture est décrite par différentes informations :

- le fournisseur concerné,
- le client concerné,
- une liste d'items,
- un taux de TVA (rapporté entre 0 et 1) qui sera variable suivant les factures,
- un montant à payer, qui est par défaut la somme des prix unitaires des items, à laquelle est appliquée ensuite le taux de TVA.

(2)

Réponse à la question 1 :

```
0,25 ... public interface IFacture {
0,25 ..... IFournisseur getFournisseur();
0,25 ..... IClient getClient();
0,25 ..... ArrayList<Item> getListeItems();
0,25 ..... double getTauxTva();
0,25 ..... default double montantAPayer(){
0,5 ..... double somme=0;
         for(Item i: this.getListeItems())
             somme += i.prixUnitaire();
         somme = somme * (1+this.getTauxTva());
         return somme;
    }
}
```

Question 2. Ecrivez une première version de la classe `Facture` qui respecte `IFacture`. Cette classe ne doit pas pouvoir être instanciée et à ce stade le taux de TVA n'est pas connu pour une facture. Ecrivez la classe avec :

(35)

- les attributs nécessaires pour implémenter l'interface,
- les méthodes cohérentes avec l'interface,
- une méthode pour ajouter un item,
- un constructeur avec paramètres. Lors de la construction, la liste d'items est vide.

Réponse à la question 2 :

0,5 public abstract class Facture
0,5 ----------> implements IFacture {
0,5 private IFournisseur fournisseur;
private IClient client;
private ArrayList<Item> listeItems;
0,5 public Facture (IFournisseur f, IClient c)
{ this.fournisseur = f;
this.client = c;
listeItems = new ArrayList<>();
}
public IFournisseur getFournisseur()
{ return this.fournisseur; }
public IClient getClient()
0,5 { return this.client; }
public ArrayList<Item> listeItems()
{ return this.listeItems; }
0,5 // on n'écrit rien concernant getTauxTVA
0,5 public void ajoute(Item i) { if(!listeItems.contains(i))
listeItems.add(i); }

Question 3. Ecrivez une classe `FactureHotel` qui peut être instanciée, incluant ses attributs, un constructeur avec paramètre et les méthodes nécessaires en utilisant les informations qui suivent. Le taux de TVA est de 0,1. Dans l'hôtellerie, le montant à payer de la facture est augmenté, pour chaque nuit d'hébergement, d'une taxe de séjour dont le montant est relatif à la catégorie de la facture (cette catégorie est exprimée en nombre d'étoiles) :

- 1 étoile : 80 centimes.
- 2 étoiles : 90 centimes.
- 3 étoiles : 1,50 euros.
- 4 étoiles : 2,3 euros.
- 5 étoiles et plus : 3 euros.

Dans la classe `FactureHotel`, vous devrez stocker un nombre de nuits d'hébergement, la catégorie de la facture (en nombre d'étoiles), et les montants possibles de taxe de séjour suivant les catégories. Proposer également (1) une méthode retournant la taxe de séjour pour une nuitée, suivant la catégorie stockée, (2) la méthode retournant le taux de TVA et (3) redéfinissez la méthode retournant le montant à payer de la facture.

3

Réponse à la question 3 :

```
0,5  public class FactureHotel -> extends Facture {
0,5      private static double tauxTVA = 0.1;
         private int nbNuits, nbEtoiles;
0,5      private static double[] taxeSejour
                 = { 0.8, 0.9, 1.5, 2.3, 3 };
0,5      public FactureHotel (IFournisseur f, IClient c,
                              int nbNuits, int nbEtoiles)
         { super (f, c);
           this.nbNuits = nbNuits;
           this.nbEtoiles = nbEtoiles;
         }
0,5      public double getTaxeSejour()
           { return taxeSejour[nbEtoiles-1]; }
         public double getTauxTVA ()
           { return FactureHotel.tauxTVA; }
0,5      public double montantAPayer()
           { return super.montantAPayer()
                    + nbNuits * getTaxeSejour();
           }
     }
```

Question 4. On propose le schéma incomplet de code suivant dans `Facture` pour construire et retourner une description d'une facture sous forme de chaîne de caractères :

```
public String description(){
   String d = "";

   if (this.fournisseur != null) d += this.fournisseur.getNom()+"\n";
   if (this.client != null) d += this.client.getNom()+"\n";
   d+="montant="+this.montantApayer();

   if (this instanceof FactureHotel)
       // .... ajouter le montant pour les nuitées de la taxe de séjour

   return d;
}
```

4

a- Expliquez en quoi il n'est pas rédigé dans l'esprit de la programmation par objets et quel problème peut se poser.

Réponse à la question 4.a :

0,75 - Le test de type (`instanceof`) demande de connaître à l'avance les sous-classes, ce qui est contraire à l'esprit objet : quand on écrit une classe on n'anticipe pas sur toutes les extensions - Sinon en cas d'ajout de nouvelle sous-classe on doit revoir le code de la super classe.
- De plus les responsabilités de manipulation des attributs sont mal distribuées.

b- Une partie de ce code peut-elle figurer dans l'interface `IFacture` ? Justifiez.

Réponse à la question 4.b :

0,75 - on peut placer dans IFacture la partie du code avant le `if(instanceof...)` puisqu'elle ne fait appel qu'à des méthodes visibles de l'interface.

Condition → utilise getFournisseur, getClient → fournisseur, client

à la place de

c- Proposez une solution correctement rédigée et complète en Java.
Réponse à la question 4.c :

```
public interface IFacture {
    ...
    default String description() {                    (Enlever 0,25 si dans classe Facture et non default)
        String d = "";
        if (getFournisseur() != null)
            d += getFournisseur().getNom() + "\n";
        if (getClient() != null)
            d += getClient().getNom() + "\n";
        d += "montant = " + this.montantAPayer();
        return d; }
}
public ... FactureHotel ... {
    ...
    public String description() {
        return super.description() +
            " taxe sejour " + this.taxeSejour();
    }
    // ou " taxes sejournées " + this.taxeSejour() * nbNuitees
}
```

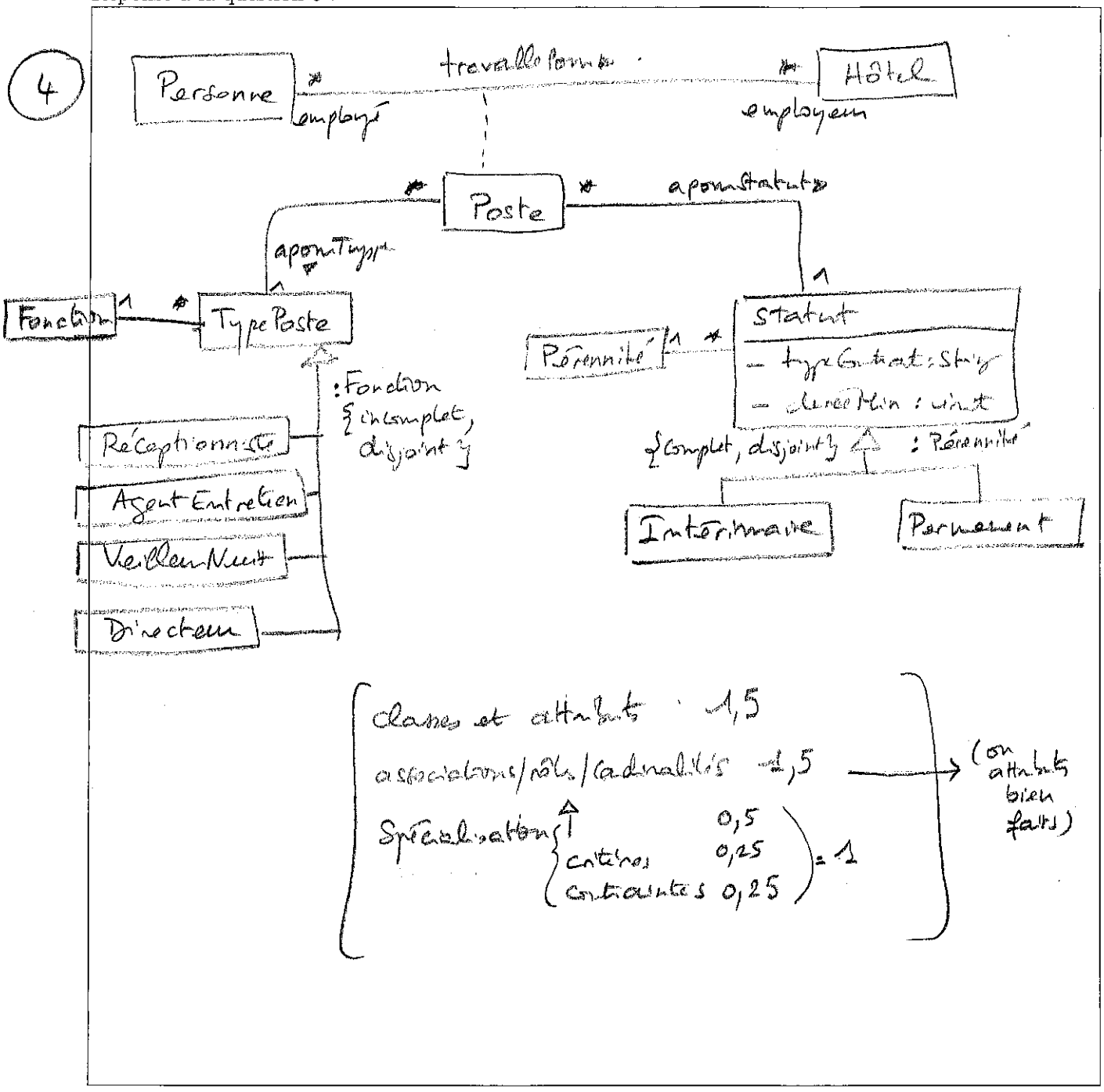
0,75 — default String description() {
String d = "";
0,5 — if (getFournisseur() != null) d += getFournisseur().getNom() + "\n";
0,5 — if (getClient() != null) d += getClient().getNom() + "\n";
0,5 — d += "montant = " + this.montantAPayer();
return d; }
}
public ... FactureHotel ... {
...
0,75 — public String description() {
return super.description() +
" taxe sejour " + this.taxeSejour();
}
// ou " taxes sejournées " + this.taxeSejour() * nbNuitees

Question 5. Ecrire l'entête et les attributs d'une classe `Hotel` qui implémente IFournisseur et stocke une liste de factures en cours. Ajoutez-lui une méthode prenant en paramètre un client, un nombre d'étoiles à considérer et permettant de créer une facture en cours, vide d'items et de nuitées.
(2)
Réponse à la question 5 :

```
0,5 — public class Hotel implements IFournisseur {
0,5 — private ArrayList<IFacture> factEnCours;
0,25 — public void ajoutFactEnCours(IClient c,
                                    int nbEtoiles)
{
0,5 — Facture_Hotel f = new FactureHotel
                        (this, c, 0, nbEtoiles);
0,25 — factEnCours.add(f);
}
```

On se place maintenant dans le contexte de la description d'hôtels et de restaurants.

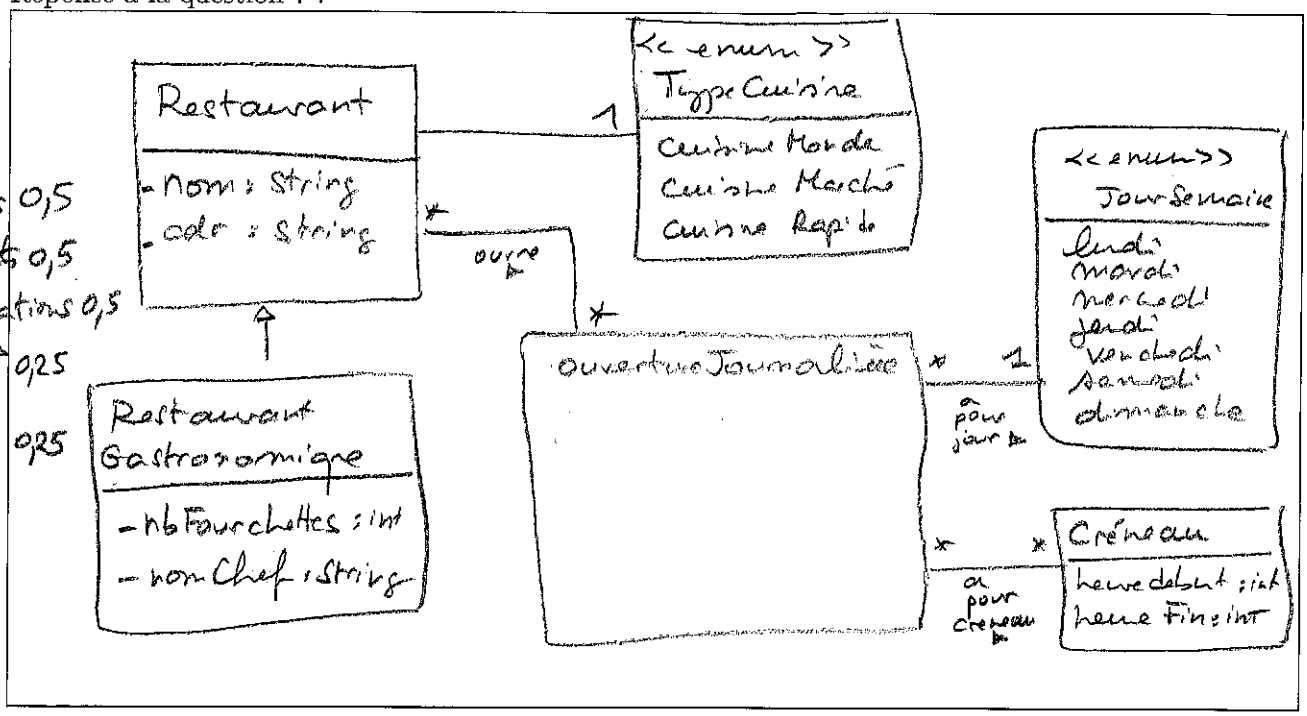
Question 6. Dessinez un diagramme de classes UML correspondant à la situation suivante. Dans l'énoncé de cette question, les noms des classes que vous devrez organiser sont écrits en italiques.
Une *personne* peut exercer dans un ou plusieurs *hôtels* dans le cadre d'un ou plusieurs *postes*, chaque poste correspondant à un type de poste et à un statut. Une personne peut occuper plusieurs postes dans chaque hôtel. Il y a plusieurs *types de postes* dans l'hôtellerie, qui peuvent se spécialiser suivant la fonction occupée. Ces fonctions incluent : *réceptionniste*, *agent d'entretien*, *veilleur de nuit*, *directeur*, mais il pourrait y en avoir d'autres. Différents *statuts* peuvent être utilisés, spécialisés selon leur pérennité : *intérimaire* ou *permanent*. Le statut est décrit par un type de contrat (chaîne de caractères) et une durée minimale d'emploi.
Réponse à la question 6 :



Question 7. Dessinez un diagramme de classes UML correspondant à la situation suivante. Un restaurant est décrit par un nom, une adresse, un type de cuisine (les types sont limités à "Cuisine du monde", "Cuisine du marché", "Cuisine Rapide") et une liste d'ouvertures journalières. Une ouverture journalière est composée d'un jour de la semaine et d'une liste de créneaux horaires pour ce jour. Un restaurant gastronomique est un restaurant qui est de plus décrit par un nombre de fourchettes et par le nom de son chef.

2

Réponse à la question 7 :



Question 8. Dessinez un diagramme d'instances UML correspondant à la situation suivante. Le restaurant "l'Huîtrière" à Lille, 3, rue des chats bossus, tenu par le chef Stéphane, est un restaurant gastronomique qui propose une cuisine du marché, qui a 4 fourchettes et est ouvert tous les vendredis de 19h à 22h et tous les samedis de 11h à 13h et de 18h à 23h.

2

Réponse à la question 8 :