EISTI 2008-2009 – Examen

Analyse et conception

2h30 – Tout document autorisé

1. Diagrammes de classes et de séquences : Déroulement d'un examen (8 points)

Dans une école on désire automatiser le déroulement d'un examen d'un étudiant donné, dans une matière donnée et pour une session donnée (dates et heures de début et de fin de l'examen). Un examen est défini par un libellé, est associé à une matière et est composé d'un texte et d'un certain nombre d'annexes. Une session est associée à un examen, un couple (date, heure) de début et un autre couple (date, heure) de fin. Un même examen peut avoir plusieurs sessions.

Le scénario nommé **examenSessionEtudiant** se passe comme suit :

- L'étudiant émarge au début de la session de l'examen en donnant son identifiant et son mot de passe.
- S'il est inscrit pour cette session de l'examen, l'ordinateur lui affiche à l'écran l'énoncé de l'examen. Pendant l'examen il peut consulter à l'écran une série de documents annexes associés à cet examen.
- Un examen est composé de questions. L'étudiant répond à chacune des questions séparément. En d'autres termes pour une question donnée, il rédige une réponse qu'il envoie dès qu'il juge avoir fini de répondre à la question. Il n'est pas obligé de répondre dans l'ordre des questions.
- Quand il a fini l'examen, il le signale en quittant la session.

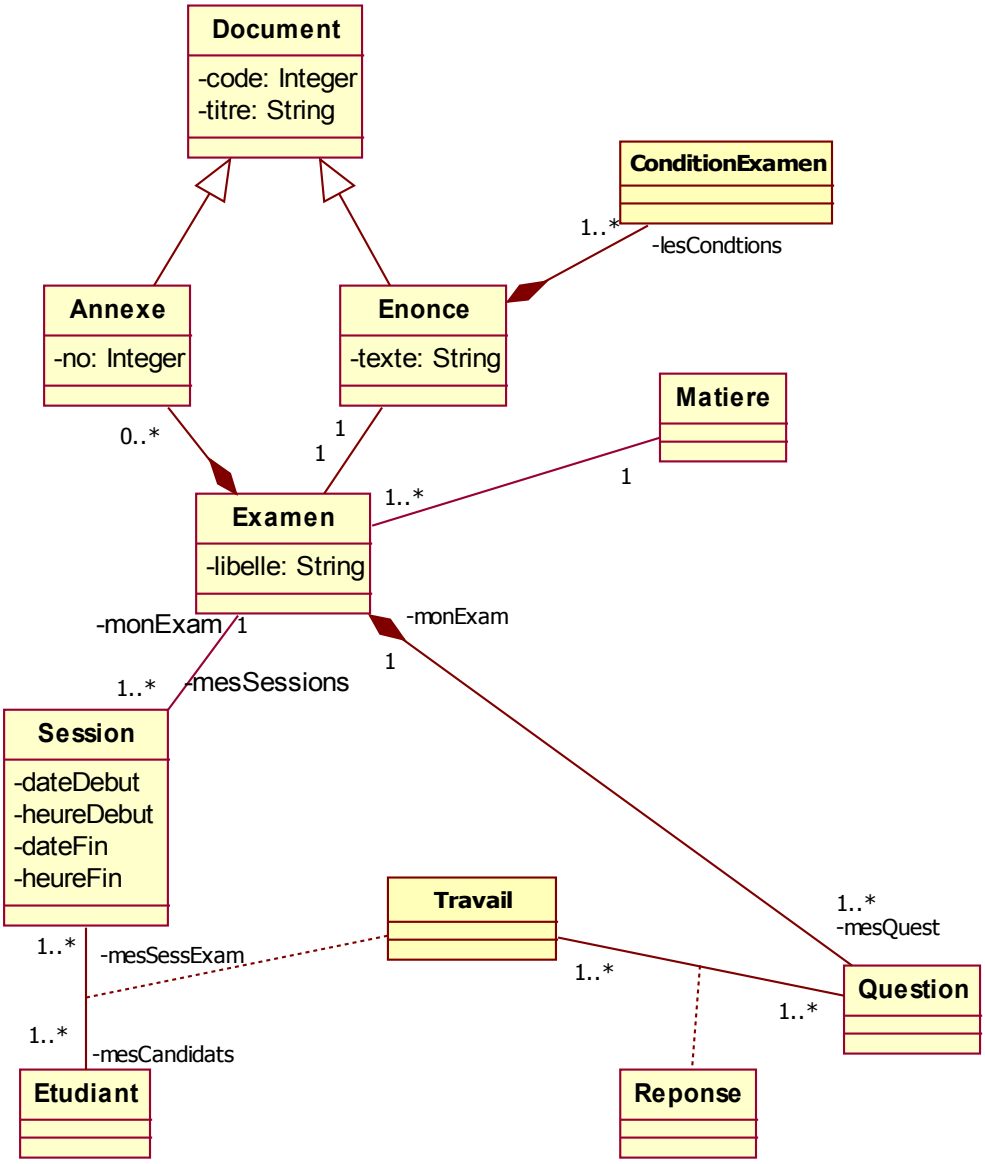
Les annexes comme l'énoncé de l'examen sont des documents. Un document est décrit par un code et un titre. En plus d'être un document une annexe comporte un numéro. De même dans un énoncé au-delà du document on ajoute les conditions d'examen.

Après avoir étudié ce cahier de charges on a identifié l'acteur et les classes suivantes :

- L'acteur ActEtudiant.
- Les classes : Examen, Session, Matiere, Etudiant, Question, Annexe, Reponse, Enonce, Document.

Q1 (3pts) : Faire le diagramme de **ces classes**. Dans ce diagramme, on fera apparaître les liens entre les classes ainsi que les informations citées dans l'énoncé. Dans un premier temps on ne demande pas de déclarer les opérations des classes.

Réponse



Q2 (3pts) : Ecrire le diagramme de séquences associé à ce scénario **examenSessionEtudiant**. Dans ce diagramme, **on ne devra utiliser que des objets des classes définies ci-dessus**.

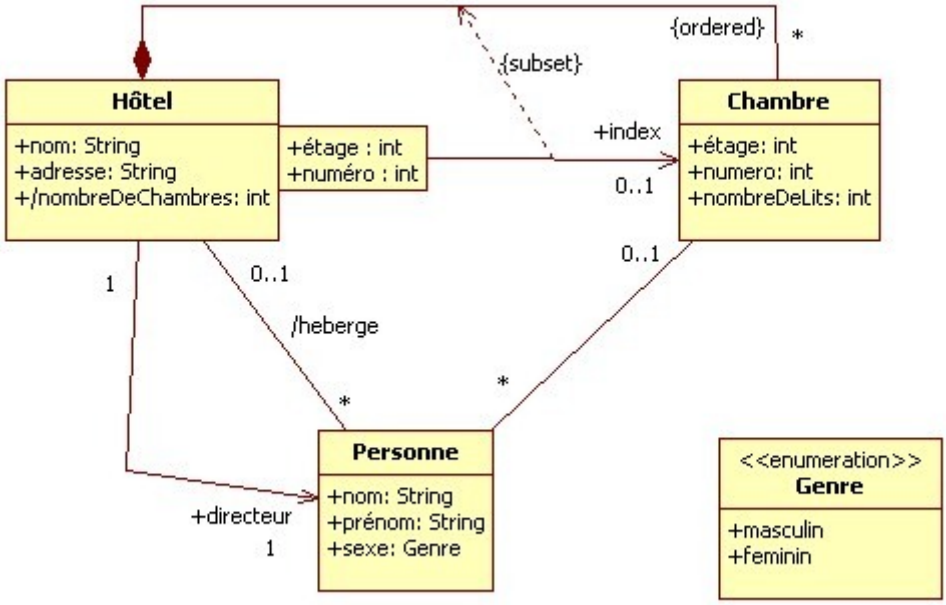
Q3 (2pts) : Compléter le diagramme de classes en ajoutant les opérations. Il faudra bien sur être cohérent avec votre diagramme de séquences.

2. Diagramme d'états/transitions : File d'attente dans un magasin (2pts)

On considère une file d'attente dans un grand magasin. Une file d'attente est associée à une caisse. Une file d'attente a une capacité maximum de 10 personnes (y compris la personne à la caisse). Une file d'attente ne peut se remplir que si sa caisse est ouverte. Une file d'attente est détruite quand la caisse est fermée et que le dernier client est servi.

Q : Donner le diagramme d'états/transitions d'un objet file d'attente. On devra préciser les attributs de la classe dont on a besoin dans le diagramme.

3. Mapping UML-Java : Chambres d'hôtel (5 points)

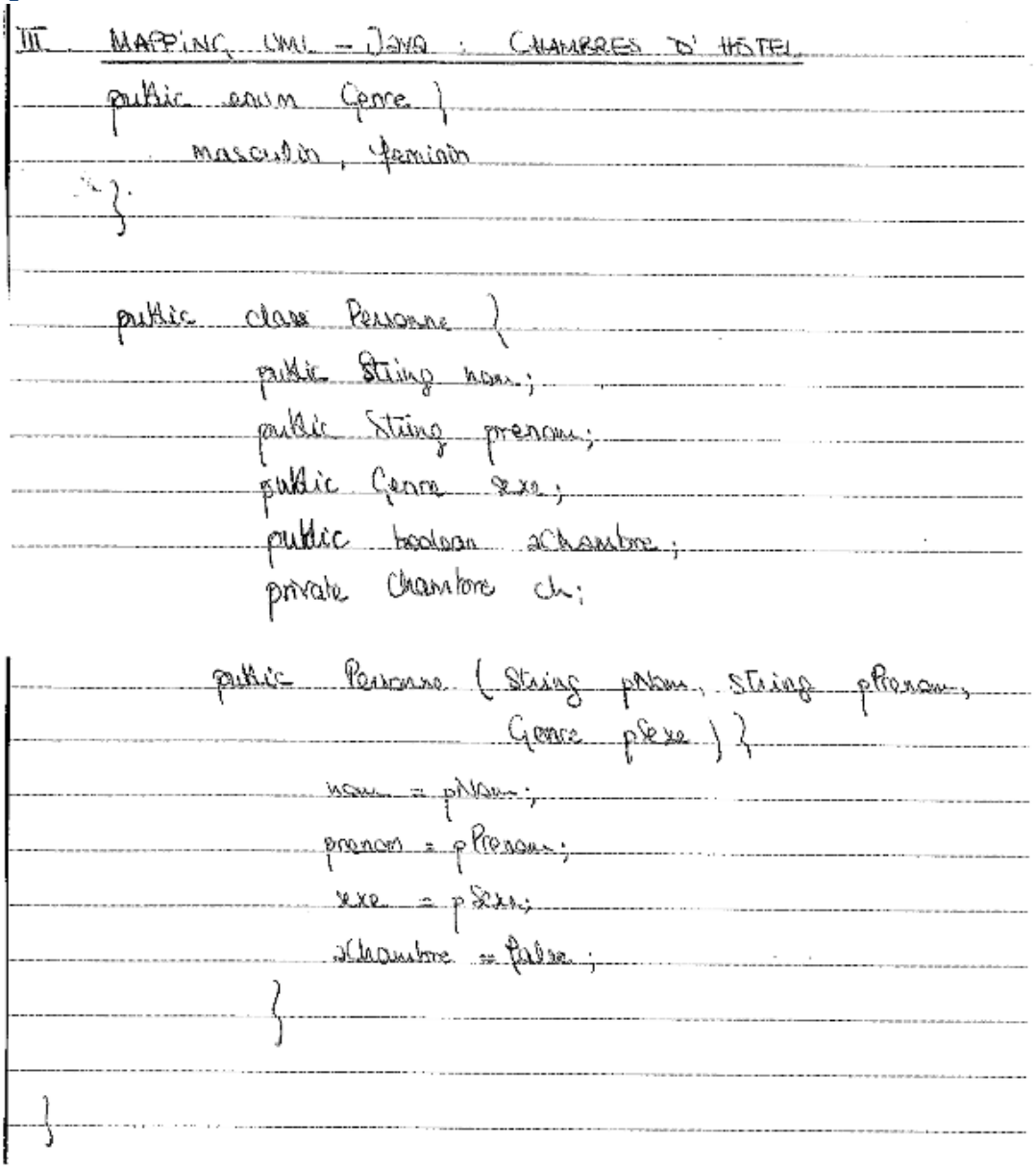


Réalisez le mapping Java du diagramme de classes ci-dessus, en prenant en compte les contraintes OCL suivantes (également traduites en français) :

```
-- à partir de l'étage et du numéro on retrouve via index la chambre
-- correspondante dans l'hôtel
-- et il n'existe pas de 13e étage (superstition oblige)
context Hôtel :
inv: self.chambre->forAll(ch:Chambre | self.index[ch.étage,ch.numéro] = ch)
inv: self.chambre->forAll(ch:Chambre | ch.étage <> 13)

-- pour un Hôtel, le nombre de chambres est calculé à partir du nombre
-- d'associations qu'il possède avec des Chambres
context Hôtel::nombreDeChambres : int
derive: self.chambre->size()

-- pour une chambre, le nombre de personnes doit être inférieur ou égal au
-- nombre de lits
context Chambre :
inv: self.personne->size <= self.nombreDelits
```

```
-- l'association Hôtel-Personne est obtenue par la navigation suivante :
-- Hôtel-Chambre-Personne
context Hôtel::heberge : Set(Personne)
derive: self.chambre.personne->asSet()
```

Réponse



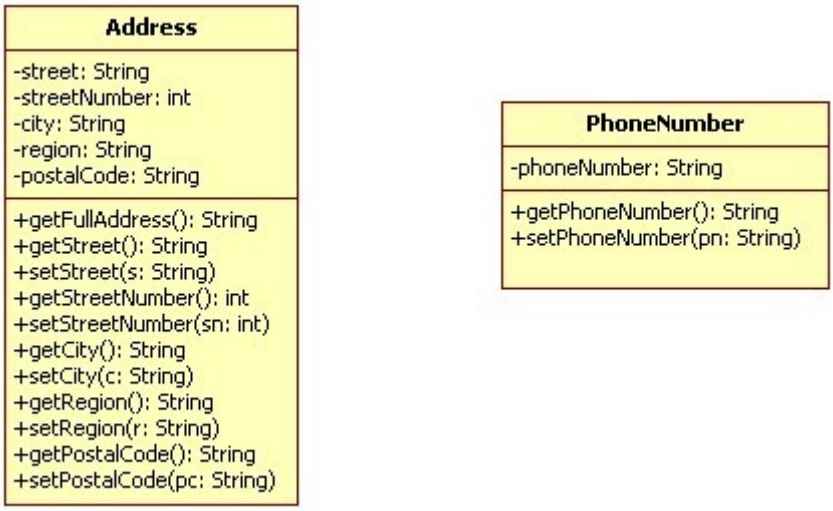
```
public class Chambre {
    public int etage;
    public int numero;
    public int nombreDeLits;
    public Set<Personne> personnes;

    public Chambre ( int pEtage, int pNumero, int pNbLits ) {
        if (pEtage == 13)
            System.out.println("Erreur de superstition");
        } else {
            etage = pEtage;
            numero = pNumero;
            nombreDeLits = pNbLits;
            personnes = new HashSet<Personne>();
        }
    }

    public void loger ( Personne p) {
        if ( personnes.size() == nombreDeLits ) {
            System.out.println("Il n'y a plus de place");
        } else {
            personnes.add (p);
            p.aChambre = true;
            p.Ch = this;
        }
    }
}
```

Très bien !

```
public class Hotel {
    public String nom;
    public String adresse;
    public int nombreDeChambres; --> à calculer
    public Personne directeur;
    private List<Chambre> chambres;
    public Map<int[], chambre> index;

    public Hotel (String pNom, String pAdresse, Personne pDir){
        nom = pNom;
        adress = pAdresse;
        nbrDeChambres = 0;
        directeur = pDir;
        chambres = new ArrayList<Chambre>();
        index = new HashMap<int[], chambre>();
    }

    public Set<Personne> lister() {
        Set<Personne> sP = new HashSet<Personne>();
        for (Chambre ch: chambres) {
            for (Personne p: personnes) {
                sP.add(p);
            }
        }
        return (sP);
    }
}
```

4. Interface et Design Patterns : adresse et téléphone (5 points)

Une entreprise américaine a développé une application qui sert à la fois de carnet d'adresses et d'agenda. Cette application utilise de façon intensive des **adresses** et des **numéros de téléphone**. Pour cela, les classes suivantes ont été développées :



La classe Address permet notamment d'obtenir l'adresse complète à partir des différentes informations de l'adresse. Le code suivant décrit la méthode getFullAddress() qui recompose l'adresse au format américain :

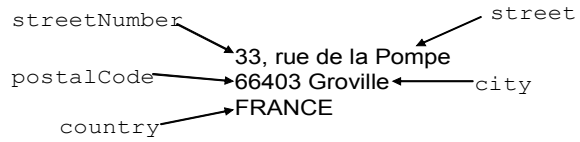
```
public String getFullAddress() {
    return getStreetNumber() + ",” + getStreet() + "\n”
           + getCity() + ", " + getRegion() + " " + getPostalCode()

           + " \n”;
}
```

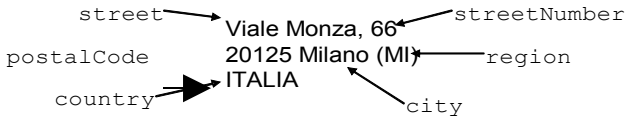
La classe PhoneNumber permet elle de vérifier la validité du numéro de téléphone entré. Le code suivant décrit la méthode setPhoneNumber() qui vérifie que le nombre entré comprend 10 chiffres (format américain) et que ce nombre est bien un nombre :

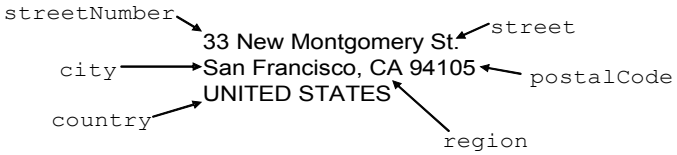
```
public void setPhoneNumber(String pn) {
    try {
        if (pn.length() == 10){
            Long.parseLong(pn);
            phoneNumber = pn;
        }
    catch (NumberFormatException exc) {
    }
}
```

L'entreprise souhaite maintenant s'attaquer au marché européen. Cependant, les formats d'adresses et de numéros de téléphone pour les pays européens sont différents et ne correspondent pas non plus au standard américain. De plus, il faut maintenant gérer les spécificités liées à l'international, i.e. le nom du pays dans l'adresse et le code pays pour le numéro de téléphone.

Voici par exemple, les formats d'adresse US, français et italien :







Voici maintenant les formats de numéro US, français et italien (la lettre C représentant 1 chiffre) :

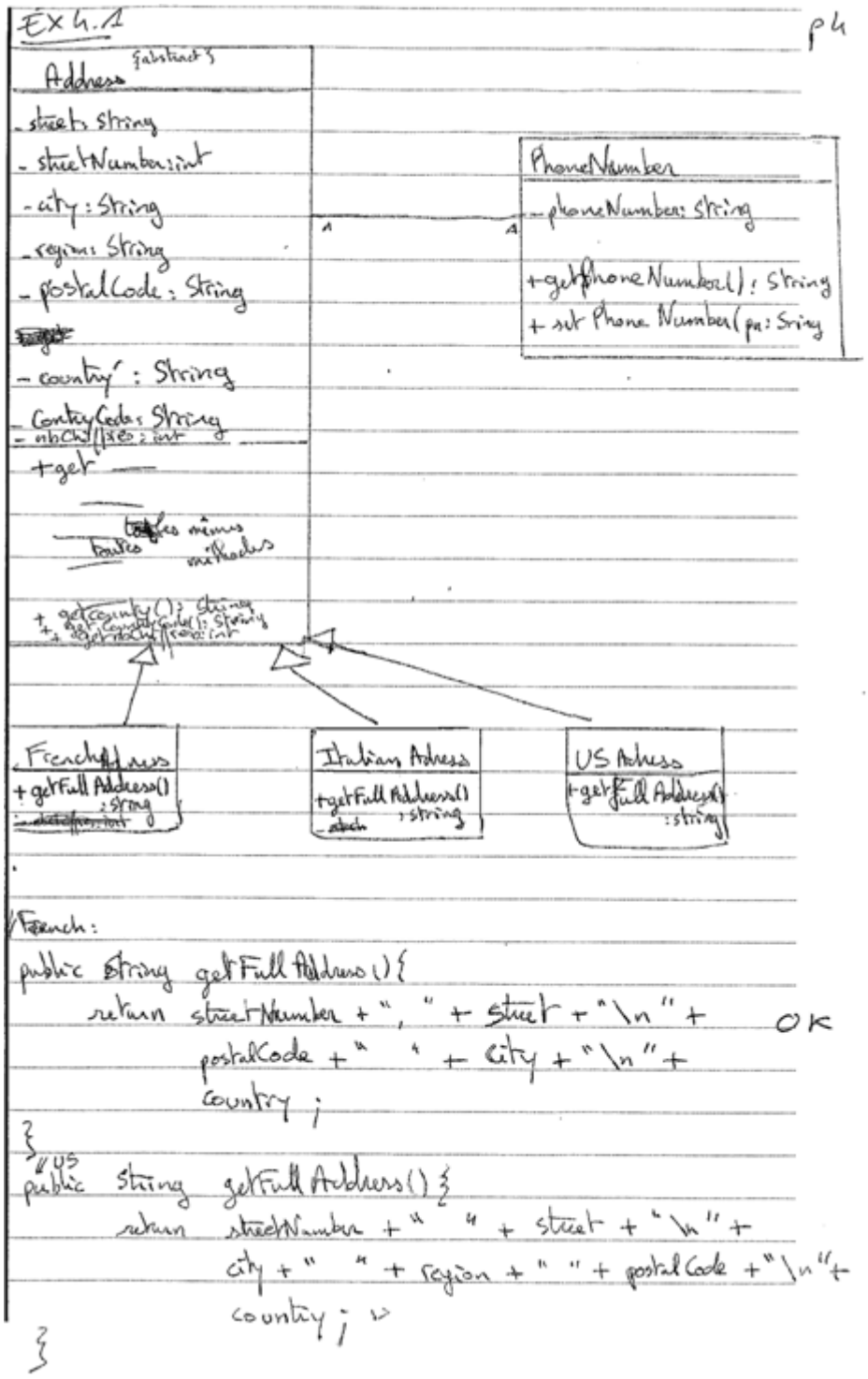
US : +01 CCCCCCCCCC

FR : +33 CCCCCCCCC

IT : +34 CCCCCCCC

1. A l'aide d'un diagramme de classes, proposez une refonte de la modélisation des adresses et des numéros de téléphone pour que l'application puisse prendre en compte de manière uniforme les différents formats utilisés par les pays. Vous vous limiterez aux exemples de pays proposés et donnerez pour chacun comment est pris en compte ses spécificités à travers le code des méthodes getFullAdress(), getPhoneNumber() et setPhoneNumber().

Réponse



```
//Iteba
public String getFullAddress () {

    return street + ", " + streetNumber + "\n" +
           postalCode + " " + city + " (" +
           region + ") " + "\n" + country ;
}

//Pour tous
public String getPhoneNumber () {

    return "+" address.countryCode + phoneNumber ;
}

//pour tous

public void setPhoneNumber ( String pn) {
    try {
        if (pn.length() == address.getnbChi||res ) {
            Long.parseLong (pn);
            phoneNumber = pn;
        }
    catch (NumberFormatException exc) {
    }
}
```

2. Proposez une conception orientée objet, permettant à l'application de créer de manière uniforme un objet adresse et un objet numéro de téléphone correspondant pour chaque pays. Vous donnerez le nom du design pattern que vous utilisez et le diagramme de classes UML correspondant adapté à la situation (i.e. en utilisant un nommage approprié pour les constituants du pattern).

Réponse