

Examen Java

Documents interdits

2h

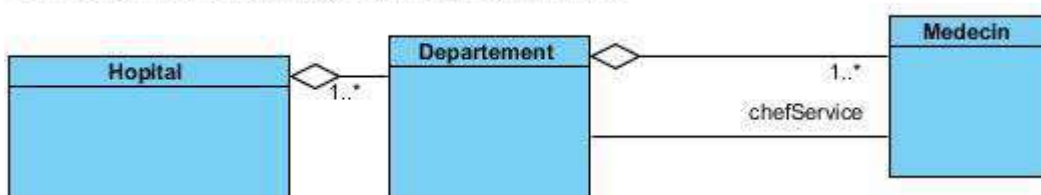
Les écarts syntaxiques ne sont pas pris en compte à condition que le tout ressemble à du Java.

Exercice 1 (22 pts)

On voudrait créer un système de gestion d'un hôpital. L'hôpital contient plusieurs départements et plusieurs médecins y sont rattachés. Un médecin assure la fonction de chef de service. On voudrait pouvoir ajouter/supprimer/rechercher un médecin. De même pour un département. L'ensemble des informations concernant les médecins sont sauvegardés dans un fichier.

1. Dessiner un diagramme explicitant les classes à créer pour réaliser cette application (3 pts)

Visual Paradigm for UML Community Edition [not for commercial use]



2. Ecrire les squelettes de classes, interfaces et placer l'ensemble dans un package « monPackage » (5 pts)
3. Implémenter les classes (implémenter les getters et les setters pour une seule classe). (10 pts)
 - Fichier Hopital.java (avec un main pour des petits tests)

```
package monPackage.efrei.fr;
```

```
import java.io.*;
import java.util.ArrayList;
```

```
public class Hopital implements Serializable {
```

```
    private static final long serialVersionUID = 4014629488557577559L;
```

```

    private String nom;
    private ArrayList<Departement> listDepartements = new
ArrayList<Departement>();

    public Hopital(String _nom) {
        nom = _nom;
    }

    public void ajouterDepartement(Departement m) {
        listDepartements.add(m);
    }

    public void supprimerDepartement(String nom) {
        listDepartements.remove(rechercherDepartement(nom));
    }

    public Departement rechercherDepartement(String nom) {
        for (Departement ref : listDepartements) {
            if (ref.equals(nom)) // on doit surcharger equals
                return ref;
        }

        return null;
    }

// désérialization pour la question 4
    public static Hopital chargerHopital(String fichier) {

        try{
            ObjectInputStream ois = new ObjectInputStream(new
BufferedInputStream(new FileInputStream(fichier)));

            return (Hopital) ois.readObject();
        } catch (Exception e){
            e.printStackTrace();
        } finally {
            System.out.println("test finally");
        }

        return null;
    }

// sérialization pour la partie 4
    public void sauvegarderHopital(String fichier) {

        try {
            ObjectOutputStream oos = new ObjectOutputStream(new
BufferedOutputStream(new FileOutputStream(fichier)));
            oos.writeObject(this);
            oos.close();
        } catch (Exception e) {
            e.printStackTrace();
        } finally {
        }
    }
}

```

```

public static void main(String[] args) {
    Hopital hop = Hopital.chargerHopital("HopitaleEFREI");

    if (hop == null) {
        System.out.println("pas de fichier sauvegarde");
        hop = new Hopital("EFREI");

        // Departements
        Departement dep1 = new Departement("pédiatrie");
        Departement dep2 = new Departement("urgences");

        hop.ajouterDepartement(dep1);
        hop.ajouterDepartement(dep2);

        System.out.println(hop.rechercherDepartement("pédiatrie"));
        //hop.supprimerDepartement("pédiatrie");
        System.out.println(hop.rechercherDepartement("pédiatrie"));

        // Medecins
        Medecin med1 = new Medecin("Bill");
        Medecin med2 = new Medecin("Helen");
        Medecin med3 = new Medecin("Nicolas 1");
        Medecin med4 = new Medecin("Nicolas 2");

        dep1.ajouterMedecin(med1);
        dep1.ajouterMedecin(med2);
        dep1.ajouterMedecin(med3);
        dep1.ajouterMedecin(med4);

        dep2.ajouterMedecin(med1);
        dep2.ajouterMedecin(med4);

        // rechercher un medecin
        System.out.println(dep1.rechercherMedecin("Nicolas 1"));

        dep1.supprimerMedecin("Nicolas 1");

        System.out.println(dep1.rechercherMedecin("Nicolas 1"));
    } else {
        // deserialisation du fichier
        System.out.println(hop.listDepartements.size());
        Departement dep1 = hop.listDepartements.get(0);
        Departement dep2 = hop.listDepartements.get(1);

        System.out.println(dep1.rechercherMedecin("Nicolas 2"));
    }

    hop.sauvegarderHopital("HopitaleEFREI");
}
}

```

- Fichier Departement.java

```
package monPackage.efrei.fr;

import java.io.Serializable;
import java.util.ArrayList;

public class Departement implements Serializable {

    private static final long serialVersionUID = 389450930388029932L;

    private String nom;
    private ArrayList<Medecin> listMedecins = new ArrayList<Medecin>();
    private Medecin chefService = null;

    public Departement(String _nom) {
        nom = _nom;
    }

    public Medecin getChefService() {
        return chefService;
    }

    public void setChefService(Medecin m) {
        chefService = m;
    }

    public void ajouterMedecin(Medecin m) {
        listMedecins.add(m);
    }

    public void supprimerMedecin(String nom) {
        listMedecins.remove(rechercherMedecin(nom)); // on devrait
surcharger equals dans Medecin !
    }

    public Medecin rechercherMedecin(String nom) {
        for (Medecin ref : listMedecins) {
            if (ref.equals(nom))
                return ref;
        }

        return null;
    }

    public boolean equals(String obj) {
        return nom.equals(obj);
    }

}
```

- Fichier Medecin.java

```
package monPackage.efrei.fr;

import java.io.Serializable;

public class Medecin implements Serializable {

    private static final long serialVersionUID = 2074352674966098888L;

    private String nom;

    public Medecin(String _nom) {
        nom = _nom;
    }

    public boolean equals(String obj) {
        return nom.equals(obj);
    }

}
```

4. Sauvegarder et charger le fichier contenant les médecins (4 pts) **[BONUS]**

Voir question précédente

Exercice 2 (8 pts)

Expliquez SUCCINCTEMENT les codes des lignes 3, 4, 5, 7, 12, 14.

```
1. public class MaClasse implements ActionListener {
2. ...
3. button1.addActionListener(this);
4. button2.addActionListener(this);

5. button2.addActionListener(new Eavesdropper(bottomTextArea));
6. }

7. public void actionPerformed(ActionEvent e) {
8. topTextArea.append(e.getActionCommand() + newline);
9. }
10. }
11.
12. class Eavesdropper implements ActionListener {
```

```

13.     ...
14.     public void actionPerformed(ActionEvent e) {
15.         myTextArea.append(e.getActionCommand() + newline);
16.     }
17. }

```

- 3,4 : ajout d'un écouteur d'évènements aux boutons (1,2) et cet écouteur n'est autre que l'objet appelant lui-même de type MaClasse (implémente ActionListener)
- 5 : ajout d'un autre écouteur d'évènements au bouton 2 de type Eavesdropper et qui réagira à l'évènement sur le bouton 2
- 7 : méthode exécutée lors de la réception d'un évènement sur les boutons 1 et 2
- 12 : classe Eavesdropper qui implémente l'interface ActionListener nécessaire pour qu'un objet de la classe puisse être utilisé comme un écouteur
- 14 : action exécutée par l'écouteur à la réception de l'évènement

Exercice 3 (10 pts)

Expliquez ligne par ligne et SUCCINCTEMENT le code suivant :

1. `InetAddress adresseServeur = InetAddress.getByName("localhost");`
2. `Socket client = new Socket(adresse, 2010);`
3. `OutputStream outputStream = client.getOutputStream();`
4. `outputStream.write(...);`

5. `ServerSocket serveur = new ServerSocket(2010);`
6. `Socket socketClient = serveur.accept();`
7. `InputStream inputStream = socketClient.getInputStream();`
8. `byte[] donneesRecus = new byte[1024];`
9. `inputStream.read(...);`

1. on récupère l'adresse IP (InetAddress) de "localhost"
2. une connexion duplex (socket) est établie avec l'adresse IP sur le port 2010 (on est côté client)
3. on récupère (on ne crée rien !) la référence du flux sortant de la connexion
4. on écrit sur ce flux et cela doit être reçu de l'autre côté de la socket (côté serveur)
5. création d'un serveur sur le port 2010
6. le serveur attend une connexion client avec la méthode bloquante « accept » ; à la réception d'une connexion, une socket client est créée avec une communication bidirectionnelle (en duplex)
7. on récupère le flux entrant sur la connexion établie (on va pouvoir récupérer tout le flux sortant du client)

8. on initialise un tableau pour stocker ce qui va être reçu
9. on lit le flux et logiquement, il sera conservé dans le tableau précédent.