Rapport de Travail pratique Java

GESTION DE CABINER MÉDICALE

Mohamed EDDAHBY | 3 décembre 2023

Introduction :

L'idée générale de ce projet est mise en œuvre une interface graphique interactive qui permettra la gestion d'un cabinier médicale.

Ce projet est pour le but de se familier avec le concept de patron de conception (design pattern) qui sont concéder comme un bon pratique pour résoudre un problème de modélisation. Pour ce but nous avons intégré par la suit trois patrons de conception :

1. Singleton : pour la connexion avec la base de données, il permet de créer une seule instance d'objet ce qui permet la connexion à la base de données une seule fois. Pour éviter les ambigüités.
2. DAO : pour faciliter la maintenance et la modification de code
3. Factory : pour faciliter l'accès aux données

Ce projet doit vas être diviser au plusieurs parties ou tâche au moment de l'implémentation :

Création des bases de données :

❖ Client

Table structure | Relation view

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)			No	None			Change Drop More
2	titre	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
3	nom	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
4	prenom	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More

❖ Créneaux

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)			No	None			Change Drop More
2	version	int(11)			No	None			Change Drop More
3	hDebut	int(11)			No	None			Change Drop More
4	mDebut	int(11)			No	None			Change Drop More
5	Hfin	int(11)			No	None			Change Drop More
6	mFin	int(11)			No	None			Change Drop More
7	id_medecine	bigint(20)			No	None			Change Drop More

❖ Médicine

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)			No	None			Change Drop More
2	version	int(11)			No	None			Change Drop More
3	titre	varchar(5)	utf8mb4_general_ci		No	None			Change Drop More
4	nom	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More
5	prenom	varchar(30)	utf8mb4_general_ci		No	None			Change Drop More

❖ Rendez-vous

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra	Action
1	id	bigint(20)			No	None			Change Drop More
2	jour	date			No	None			Change Drop More
3	id_client	bigint(20)			No	None			Change Drop More
4	id_creneau	bigint(20)			No	None			Change Drop More

1 : création de connexion :

```java
public class Connections {
    private static Connection conn;
    static  {
        try {
            Class.forName("com.mysql.cj.jdbc.Driver");
            conn = DriverManager.getConnection("jdbc:mysql://localhost/Cabinier","root", "");
        }catch (SQLException e) {
            System.out.println("SQL problems");
        } catch (ClassNotFoundException e) {
            System.out.println("Driver not exist");
        }
    }
    public static Connection  getConnection(){
        return conn;
    }
}
```

Cette classe est une bonne pratique de design pattern Singleton car on 'a limité l'accès aux données de notre classe et aussi le mettre statique pour que l'objet ne peut pas créer une instance de classe chaque fois, par la suit on aura une seule instanciation de cette classe.

2 : création des classes :

Dans cette partie la création des classes nécessaires pour la gestion de cabinier médicale :

Classe Créneaux

```java
public class Creneaux {
    private  int version;
    private int hDebut;
    private int mDebut;
    private int hFin;
    private int mFin;
    private int id_Medecin;

    public int gethFin() { return hFin;}
    public int gethDebut() { return hDebut;}
    public int getId_Medecin() {return id_Medecin;}
    public int getmDebut() { return mDebut;}
    public int getVersion() { return version;}
    public int getmFin() {  return mFin;}

    public String ToString(){
        return "Creneaux{"+getVersion() +"\t"+gethDebut() +"\t"+getmDebut()+"\t"+gethFin() +"\t"+getmFin()+"}";
    }
}
```

Class client

```java
public class Clients {

    private int id;
    private String titre ;
    private String nom;
    private String prenom;

    public Clients(int anInt, String string, String rsString, String s){
        id = anInt;
        titre = string;
        nom = rsString;
        prenom= s;
    }

    public String getPrenom() { return prenom;}
    public String getTitre() { return titre;}
    public int getId() { return id;}
    public String getNom() {return nom;}
    public String ToString(){
        return "client{"+getId() +"\t"+getTitre() +"\t"+getNom()+"\t"+getPrenom() +"}";
    }

}
```

Classe Médecine

```java
public class Medecins {

    private int id ;
    private  int version ;
    private String titre;
    private String nom;
    private  String prenom;

    public void setVersion(int version) {this.version = version;    }
    public int getVersion() { return version;}
    public String getPrenom() {return prenom;}
    public int getId() {return id;}
    public String getTitre() {return titre;}
    public String getNom() {return nom;}

    public String ToString(){
        return "Medecine{"+getId() +"\t"+getTitre() +"\t"+getNom()+"\t"+getPrenom() +"}";
    }
}
```

Classe Rendez-vous

```java

public class Rendez {
    private int id ;
    private Date jour ;
    private int id_Client;
    private int id_Creneau;



    public int getId() {return id;}
    public Date getJour() {return jour;}
    public int getId_Client() {return id_Client;}
    public int getId_Creneau() {return id_Creneau;}

    public String ToString(){
        return "Rendez Vous{"+getId() +"\t"+getJour() +"\t"+getId_Client()+"\t"+getId_Creneau() +"}";
    }
}
```

3 : création d'une interface :

Le but de cette interface est séparé entre les classes et leur implémentation, ce qui fait la gestion des classes et leur implémentation sera plus facile est simple, en peut même connecter plusieurs bases de données dans le même code. Tout cela grâce au design pattern DAO.

Interface GestionCabinier

```java
public interface GestionCabinier<T> {

    T getT(int id) throws SQLException;

    List<T> getTs() throws SQLException;

    void AddT(T p) throws SQLException;

    void updateT (T p, int id) throws SQLException;

    void deleteT(int id) throws SQLException;
}
```

Cette interface contient la signature des fonctions de gestion (CRUD).

Ces fonctions seront manipulées par la suite déponde de la classe

L'utilité de cette interface va nous serve à créer plusieurs techniques de stocké notre data par exemple utiliser un fichier :

Cela ce que on va avoir par la suite :

4 : Sauvegarder les données dans un fichier

Pour cette technique sous arons besoins d'une classe hérite de l'interface GestionCabinier. Cette classe doit implémenter tout les fonctione de l'interface :

Pour faciliter la tâche de manipuler les données dans cette class j'ai créé une fonction qui retourne tous les éléments existe dans le fichier.

```
1   public class DataSerialization  implements GestionCabinier<Clients>{
2
3       public  List<Clients> getFromSerialFile() throws IOException {
4           List<Clients> ls = new ArrayList<>();
5           try {
6               FileInputStream fsi = new FileInputStream("Clients"+".serial");
7               ObjectInputStream obi = new ObjectInputStream(fsi);
8               try {
9                   ls = (List<Clients>) obi.readObject();
10              }finally {
11                  obi.close();
12                  fsi.close();
13              }
14          }catch (ClassNotFoundException e) {
15              System.out.println("This class name doesn't exist !");
16          }catch (IOException ee){
17              System.out.println("There is an error some where ! ");
18          }finally {
19              return  ls;
20          }
21      }
```

Puis j'ai implémenté les fonctions de l'interface :

```java

@Override
public Clients getT(int id) throws SQLException, IOException {
    List<Clients> list = getFromSerialFile();
    Clients clients = null;
    for (Clients cl : list){
        if(cl.getId() == id){
            clients= cl;
        }
    }
    return clients;
}

@Override
public List<Clients> getAllTs() throws SQLException, IOException {
    List<Clients> ls= getFromSerialFile();
    return ls;
}

@Override
public void AddT(Clients client) throws SQLException {
    try {
        List ls = getFromSerialFile();
        ls.add(client);
        FileOutputStream fs = new FileOutputStream("Clients"+".serial");
        ObjectOutputStream obo = new ObjectOutputStream(fs);
        try {
            obo.writeObject(ls);
            obo.flush();
        }finally {
            fs.close();
            obo.close();
            System.out.println("Done !");
        }
    }catch (IOException e){
        System.out.println(e.getMessage());
    }
}

```

Cette classe implémentes Gestion Cabiner interface puis les méthodes se change pour permettre la sérialisation et les désérialisation des objets sous forme d'une List d'objet.

```java
    Clients c1 = new Clients(2 , "M", "alex", "mali");
    Clients c3 = new Clients(3 , "F", "fatima", "fatima");

    DataSerialization dataSerialization = new DataSerialization();
   List<Clients> list = new ArrayList<>();
    System.out.println("get list of elements before ************* ");
   list =dataSerialization.getAllTs();
   for (Clients c : list){
       System.out.println(c.ToString());
   }
    System.out.println("add new element **************");
    dataSerialization.AddT(c1);
    System.out.println("get list of elements after adding new one *************** ");
    list =dataSerialization.getAllTs();
    for (Clients c : list){
        System.out.println(c.ToString());
    }
    System.out.println("get element bu ID ***************");
   Clients c = dataSerialization.getT(2);
    System.out.println(c.ToString());

}
```

Dane la classe Mains j'ai créé certains objets pour créer une instance de mon class DataSerialization puis j'ai appelé les fonctions pour le test.

Voilà le résultat

```
get list of elements before *************
client{3    F   fatima  fatima}
add new element **************
Done !
get list of elements after adding new one ***************
client{3    F   fatima  fatima}
client{2    M   alex    mali}
get element bu ID ***************
client{2    M   alex    mali}

Process finished with exit code 0
```

5 : Création des interfaces graphique pour le projet :

Pour cette partie nous avons utilisé le Biblio Swing pour créer des interfaces graphiques.

Cette biblistique contient deux parties soit créé dès les éléments en utilisant le code soit utilisant une interface pour créer les éléments.

Pour ce projet nous avons créé les éléments utilisant le code.

Pour bien organiser ce projet nous avons diviser l'interface au plusieurs Panels sous forme des classes :

Prenat l'exemple de la classe Client :

➢ Classe Client

```java
public class ClientsPanel extends JPanel {
    private GestionCabinier<Clients> clientsGestion;

    private JTextField idField;
    private JTextField titleField;
    private JTextField nameField;
    private JTextField firstNameField;
    private JButton addButton;
    private JButton readButton;
    private JButton updateButton;
    private JButton deleteButton;
    private JButton showAllButton;
    private JTextArea outputArea;
```

Cette classe contient les attributs pour construire un panel.

```java
public ClientsPanel() {
    setLayout(new BorderLayout());

    clientsGestion = new ClientImplimentaion();

    idField = new JTextField(15);
    titleField = new JTextField(15);
    nameField = new JTextField(15);
    firstNameField = new JTextField(15);

    addButton = new JButton("Add");
    readButton = new JButton("Read");
    updateButton = new JButton("Update");
    deleteButton = new JButton("Delete");
    showAllButton = new JButton("Show All");

    outputArea = new JTextArea();
    outputArea.setEditable(false);

    addButton.addActionListener(e -> addClient());
    readButton.addActionListener(e -> readClient());
    updateButton.addActionListener(e -> updateClient());
    deleteButton.addActionListener(e -> deleteClient());
    showAllButton.addActionListener(e -> showAllClients());

    JPanel inputPanel = new JPanel();
    inputPanel.setLayout(new GridLayout(6, 2));
    inputPanel.add(new JLabel("ID:"));
    inputPanel.add(idField);
    inputPanel.add(new JLabel("Title:"));
    inputPanel.add(titleField);
    inputPanel.add(new JLabel("Name:"));
    inputPanel.add(nameField);
    inputPanel.add(new JLabel("First Name:"));
    inputPanel.add(firstNameField);
    inputPanel.add(addButton);
    inputPanel.add(readButton);
    inputPanel.add(updateButton);
    inputPanel.add(deleteButton);
    inputPanel.add(showAllButton);
```

Le constricteur de cette classe pour initialiser les éléments président et former une interface avec des Button et des espaces pour ajouter est modifier les données.

Au début j'ai les éléments puis j'ai les ajouter au panel

Puis j'ai affecté des fonctions au buttons à partir de addActionListner ().

Chaque Button fait une actions grâce à une fonction.

Voilà les fonctions de cette classe.

- AddClient

```java
private void addClient() {
    try {
        int id = Integer.parseInt(idField.getText());
        String title = titleField.getText();
        String name = nameField.getText();
        String firstName = firstNameField.getText();

        Clients newClient = new Clients(id, title, name, firstName);
        clientsGestion.AddT(newClient);

        outputArea.setText("Client added successfully!");
    } catch (NumberFormatException | SQLException e) {
        outputArea.setText("Error adding client: " + e.getMessage());
    }
}
```

- ReadClient

```java
private void readClient() {
    try {
        int id = Integer.parseInt(idField.getText());
        Clients client = clientsGestion.getT(id);

        outputArea.setText("Read Client:\n" + client);
    } catch (NumberFormatException | SQLException e) {
        outputArea.setText("Error reading client: " + e.getMessage());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

➢ ShowAllClients

```java
private void showAllClients() {
    try {
        List<Clients> clientsList = clientsGestion.getAllTs();

        StringBuilder result = new StringBuilder("All Clients:\n");
        for (Clients client : clientsList) {
            result.append(client).append("\n");
        }

        outputArea.setText(result.toString());
    } catch (SQLException e) {
        outputArea.setText("Error fetching clients: " + e.getMessage());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

➢ updateClient

```java
private void updateClient() {
    try {
        int id = Integer.parseInt(idField.getText());
        String title = titleField.getText();
        String name = nameField.getText();
        String firstName = firstNameField.getText();

        Clients updatedClient = new Clients(id, title, name, firstName);
        clientsGestion.updateT(updatedClient, id);

        outputArea.setText("Client updated successfully!");
    } catch (NumberFormatException | SQLException e) {
        outputArea.setText("Error updating client: " + e.getMessage());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

➢ deleteClient

```java
private void deleteClient() {
    try {
        int id = Integer.parseInt(idField.getText());
        clientsGestion.deleteT(id);

        outputArea.setText("Client deleted successfully!");
    } catch (NumberFormatException | SQLException e) {
        outputArea.setText("Error deleting client: " + e.getMessage());
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

Pour Ajouter son panel a un frame j'ai créé interface générale qui prend une panel chaque fois on clique sur un Button.

Cette interface est entendue de la classe JFrame

La même chose que l'interface président on crée des attributs puis les initialiser dans le constricteur.

```
public class MedicalCenterApp extends JFrame {
    private JPanel mainPanel;
    private JPanel headerPanel;
    private GestionCabinier<Clients> clientsGestion;
    private GestionCabinier<Medicine> medicineGestion;
    private GestionCabinier<Rendez> rendezVousGestion;
    private GestionCabinier<Creneaux> creneauxGestion;

```

```java
public MedicalCenterApp() {
    setTitle("Medical Center Application");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    mainPanel = new JPanel();
    headerPanel = new JPanel();
    headerPanel.setLayout(new GridLayout(1, 4));

    clientsGestion = new ClientImplimentaion();
    medicineGestion = new MedicineImplementation();
    rendezVousGestion = new RendezVousImplementation();
    creneauxGestion = new CreneauxImplementation();

    // Create instances of the graphical interfaces
    ClientsPanel clientsPanel = new ClientsPanel();
    MedicineForm medicineForm = new MedicineForm();
    RendezVousForm rendezVousForm = new RendezVousForm();
    CreneauxForm creneauxForm = new CreneauxForm();

    // Add buttons to switch between panels
    addButton("Clients", clientsPanel);
    addButton("Medicine", medicineForm);
    addButton("RendezVous", rendezVousForm);
    addButton("Creneaux", creneauxForm);

    // Show ClientsPanel by default
    showPanel(clientsPanel);

    getContentPane().setLayout(new BorderLayout());
    getContentPane().add(headerPanel, BorderLayout.NORTH);
    getContentPane().add(mainPanel, BorderLayout.CENTER);

    pack();
    setLocationRelativeTo(null);
    setVisible(true);
}
```

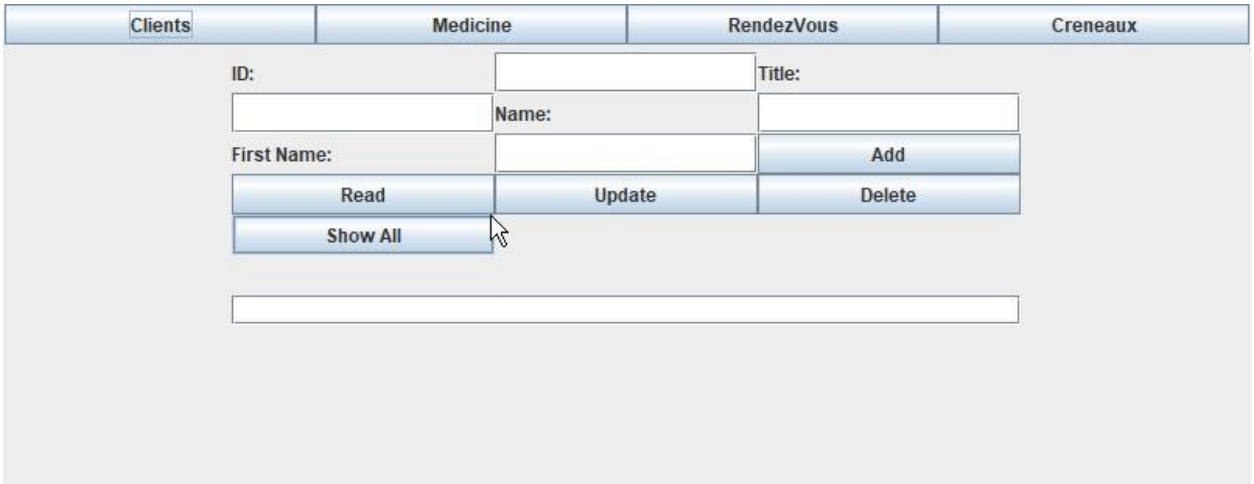
```java
private void addButton(String text, JPanel panel) {
    JButton button = new JButton(text);
    button.addActionListener(e -> showPanel(panel));
    headerPanel.add(button);
}

private void showPanel(JPanel panel) {
    mainPanel.removeAll();
    mainPanel.add(panel);
    mainPanel.revalidate();
    mainPanel.repaint();
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(MedicalCenterApp::new);
}
```

Cette partie de code est la responsable de changer le panel dépond de Button qui cliquer.

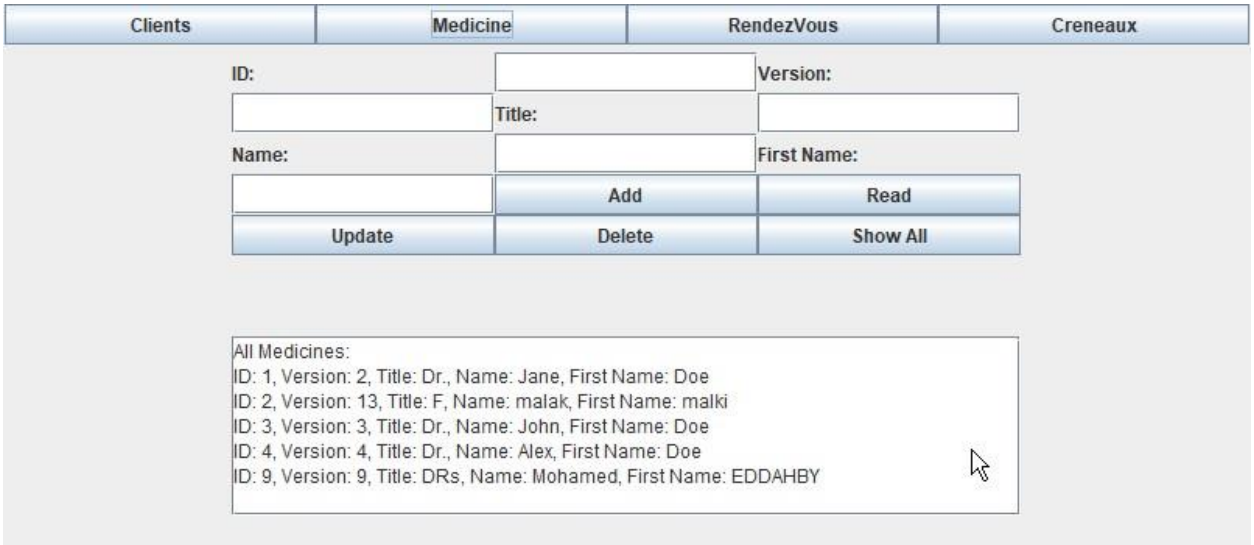
showPanel : permet d'effacer le panel existe et ajout une nouvelle au frame.

Les resultas :

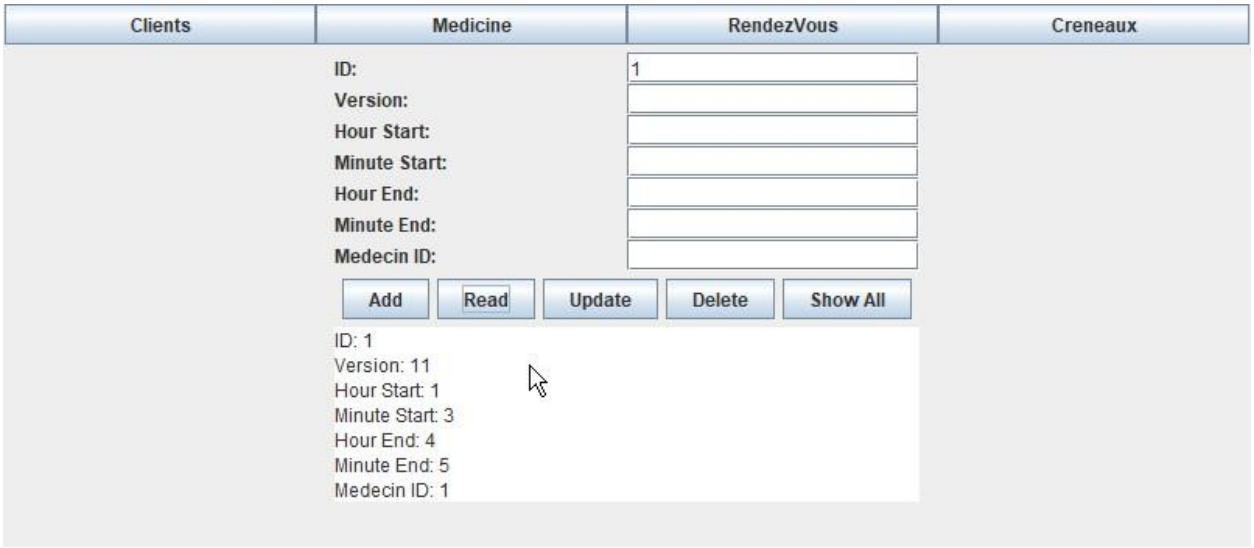
- Interface Générale :



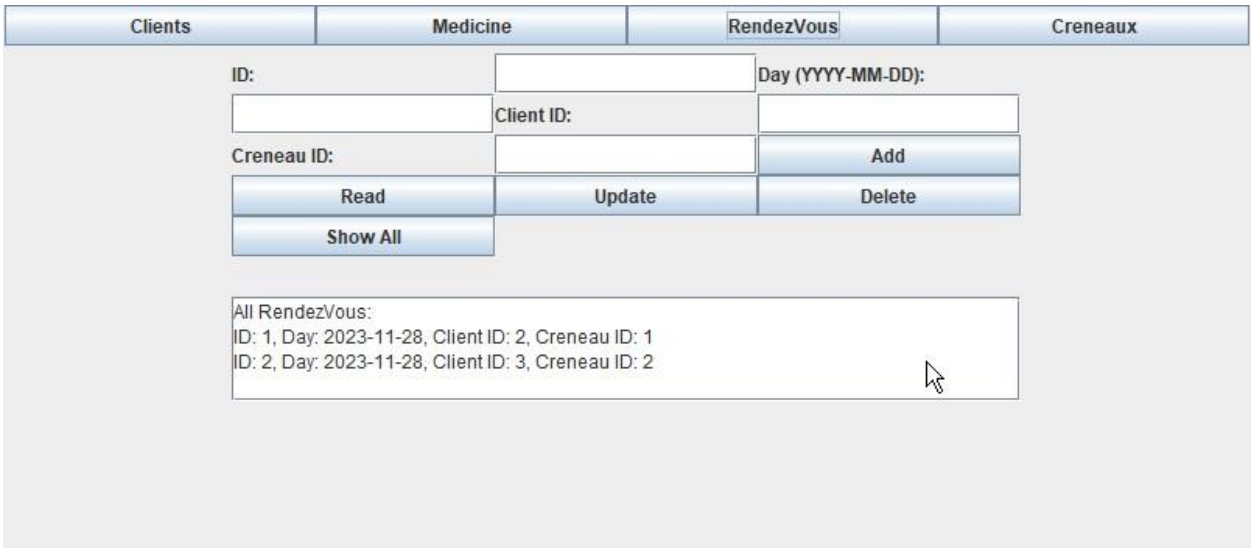
- showAll Medicine



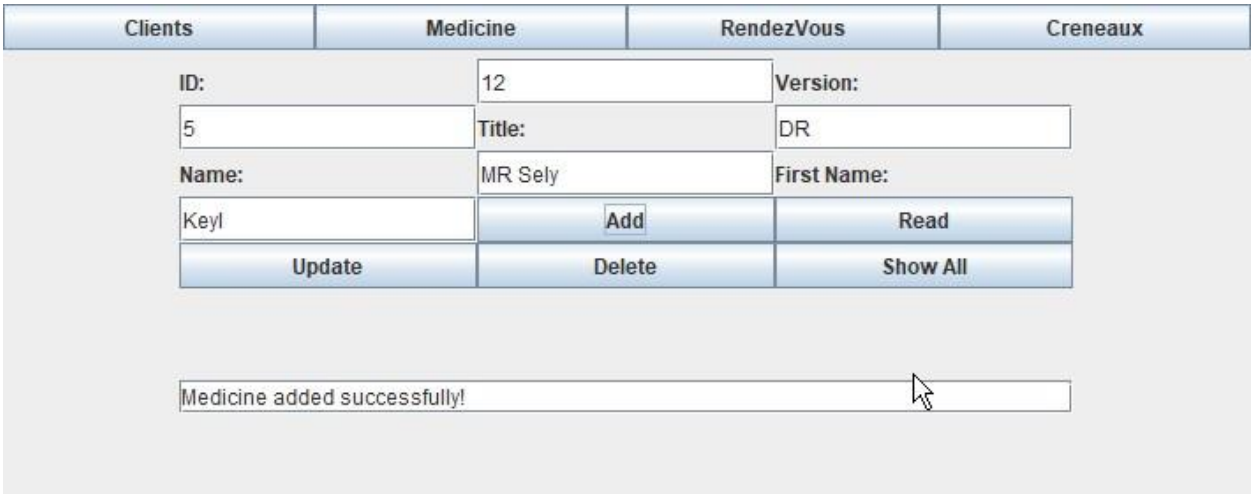
- Read Medicine par ID



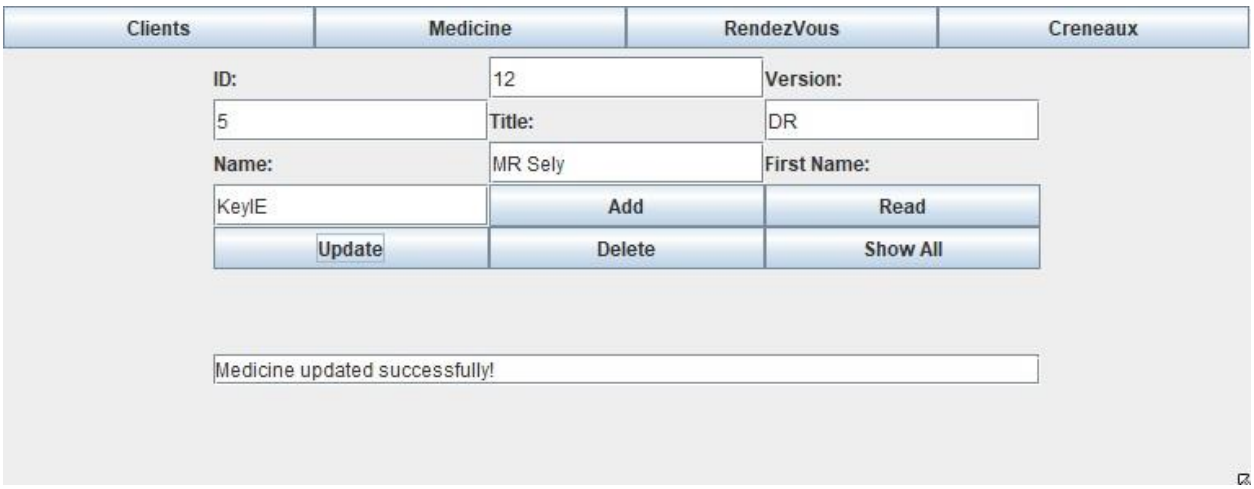
- List des Rendez-vous:



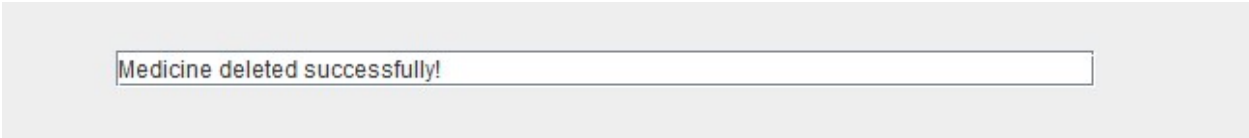
- Ajouter un medicine



- Modifier un médicine



- Supprimer un médicine :



Conclusion:

Ce projet est pour le but de crée une interface graphique qui permet la gestion d'une cabiner médicale.

Les réalisations de ce Project doivent respecter plusieurs normes :

- Utiliser les patrons de conception DAO, Factory, Singleton.
- Crée une interface graphique utilisant le code.
- Utiliser plusieurs moyennes pour stocke les données : fichier, base de données.

En fine le projet a été réaliser, se qui nous permettre d'améliorer notre compétence et notre connaissance au niveau de langage de programmation Java.