**Université de Picardie Jules Verne - INSSET de Saint-Quentin**

**Licence Pro Conception et Développement d'Application Web et Mobile**

# Rapport Projet : Gestion des donneurs de sang

# Application Web en JAVA

Réaliser par : Atmane Abidar

Encadré par Mr. Hamidi Massinissa

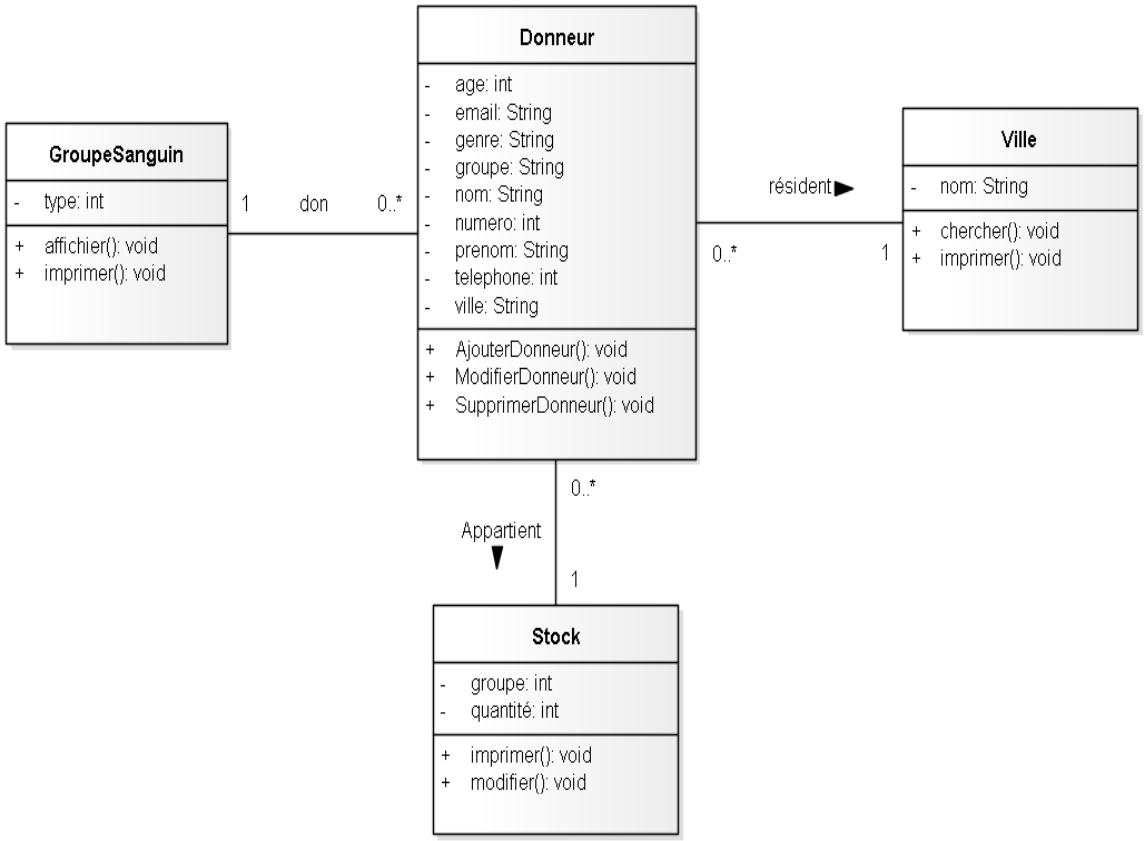
2021/2022

## Description de la problématique :

Le monde du travail actuel tend à migrer vers l'automatisation des tâches par le biais de l'informatisation. Le besoin de facilité, efficacité et fiabilité dans la résolution des problèmes et dans la réalisation des tâches est devenu énorme et pousse les entrepreneurs à doter de leurs entreprises des systèmes de gestion automatisés. De ce fait, j'ai préféré, pour ce projet universitaire, de développer en Java un système de gestion de donneurs de sang en vue d'améliorer mes compétences et mes connaissances en modélisation et programmation, et en vue de continuer le travail sur le programme et de le rendre utile pour un centre de Don de Sang.

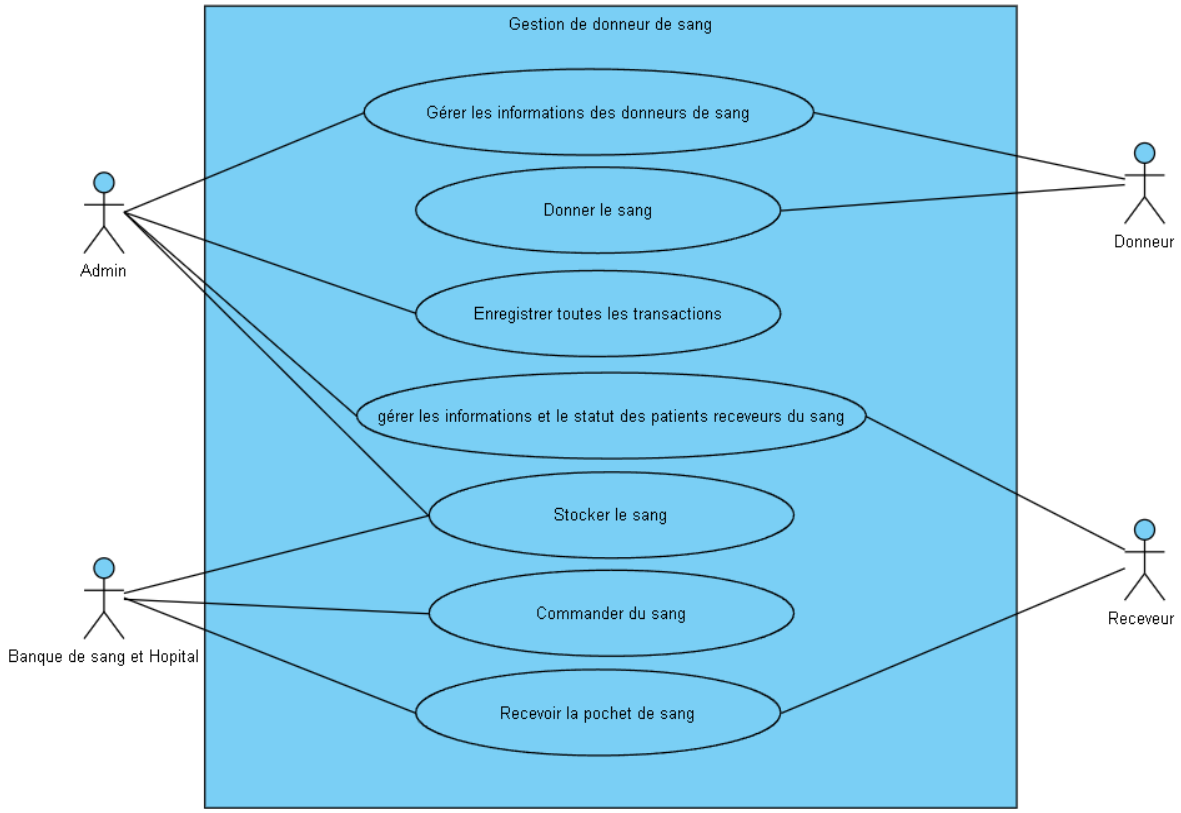
## Diagramme de classes :

Le diagramme de classes est un type de diagramme UML qui décrit un système en visualisant les différents types d'objets au sein d'un système et les types de relations statiques qui existent entre eux. Il illustre également les opérations et les attributs des classes. J'ai réalisé ce diagramme de classes en utilisant le programme Enterprise Architect.



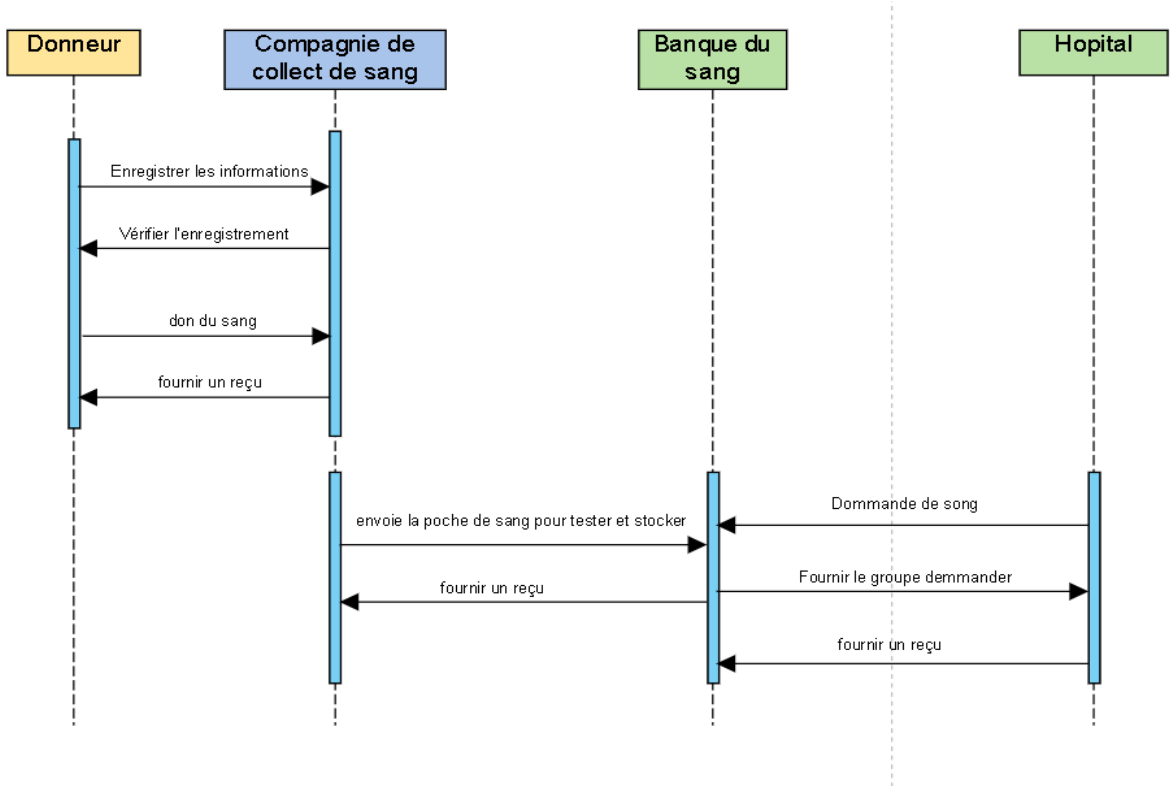
## Diagrammes de cas d'utilisation

Le diagrammes UML utilisés pour une représentation du comportement fonctionnel d'un système logiciel. J'ai réalisé ce diagramme en utilisant le programme Visual Paradigm Free Edition.




## Le diagramme de séquences

C'est la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation




## Design pattern

C'est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel. Il décrit une solution standard, utilisable dans la conception de différents logiciels.

## Singleton pattern :

Mon programme utilise de façon importante la base de données que j'ai nommé "BDD" pour fonctionner. Il me faudrait une façon de faire pour avoir une seule instance d'accès à la base donnée, il faudrait protéger cette instance contre l'écrasement pour être sûr d'utiliser toujours la même.

Dans plein d'endroits différents, Une nouvelle classe est créée à chaque fois et chaque instanciation accède à la même donnée en même temps.

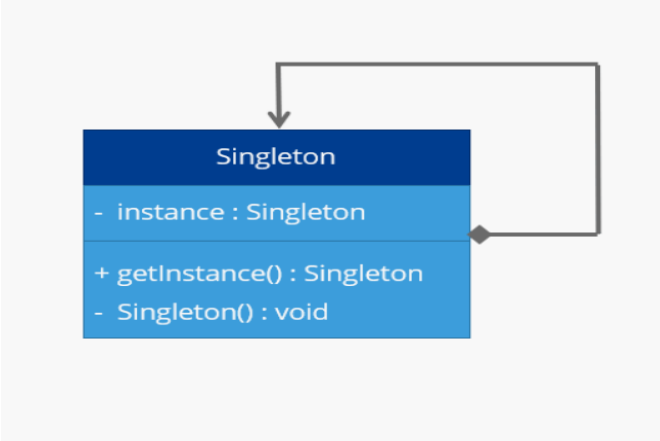
Le but du singleton pattern est d'éviter qu'une classe ne crée plus d'un objet. Pour ce faire, on crée l'objet souhaité dans une classe propre, puis on le récupère sous forme d'instance statique. Le singleton est l'un des patrons les plus simples, mais les plus puissants dans le développement de logiciels.

```java
public class Singleton {
    private static Singleton instance; // protégé contre l'accès externe et statique
    private Singleton() {} // constructeur privé avec protection d'accès externe
    public static Singleton getInstance() { // méthode publique, appel par code
        if (instance == nul) { // seulement si aucune instance n'existe, en créer une nouvelle
            instance = new Singleton();
        }
        return instance;
    }
```

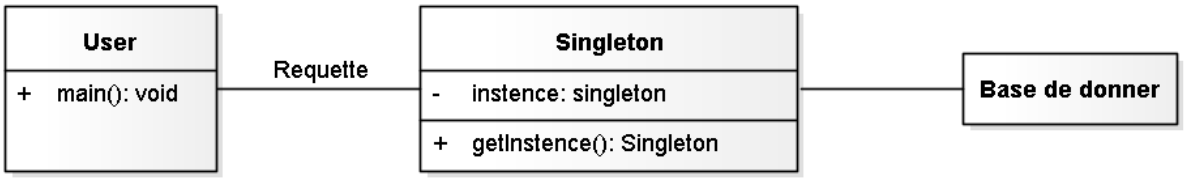
Structure de base d'un patron de conception singleton en UML, qui, une fois créé, ne subsiste que pour lui-même. Il n'est **pas possible** de changer quoi que ce soit de l'extérieur dans le singleton créé. C'est exactement le but de l'utilisation du singleton Pattern.

Le singleton complet est constitué d'un seul objet, car il n'y a qu'une seule instance d'une classe à créer.

L'utilisateur n'est censé utiliser que la fonction getInstance, Dans cette fonction, on vérifie si l'instance de la base de données existe déjà.




Pour s'assurer qu'il reste avec une seule instance unique, il faut empêcher les utilisateurs de créer de nouvelles instances. Pour ce faire, le c**onstructeur** doit déclarer le patron comme « **privé** ». De cette façon, seul le code du Singleton peut instancier le Singleton lui-même. Cela garantit qu'un seul et même objet peut atteindre l'utilisateur. Si cette instance existe déjà, aucune nouvelle n'est créée.



## Design pattern Factory Method

Ce design pattern permet de créer une instance à partir d'une classe dérivée d'un type abstrait, selon un critère donné et en évitant systématiquement d'avoir à évaluer le critère de choix. La classe exacte utilisée pour produire l'objet n'est donc pas connue par l'appelant.

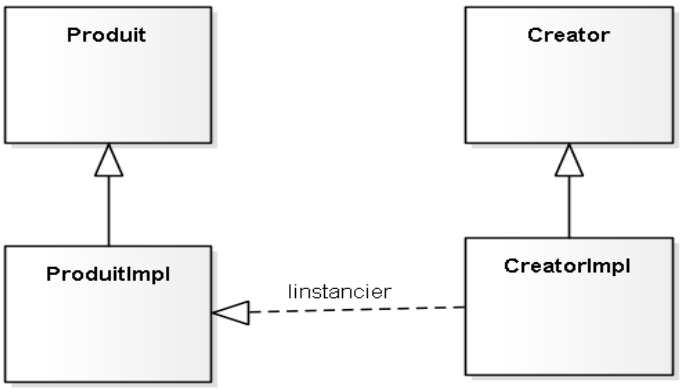
La construction d'un objet d'image dans le programme à partir de son nom de fichier : le problème étant que la manière de charger (ou sauvegarder) une image est dépendante de son type (gif, jpg, png...) et donc de l'extension du fichier.

Voici un diagramme de classe UML présentant le pattern Factory Method dans sa forme originelle.

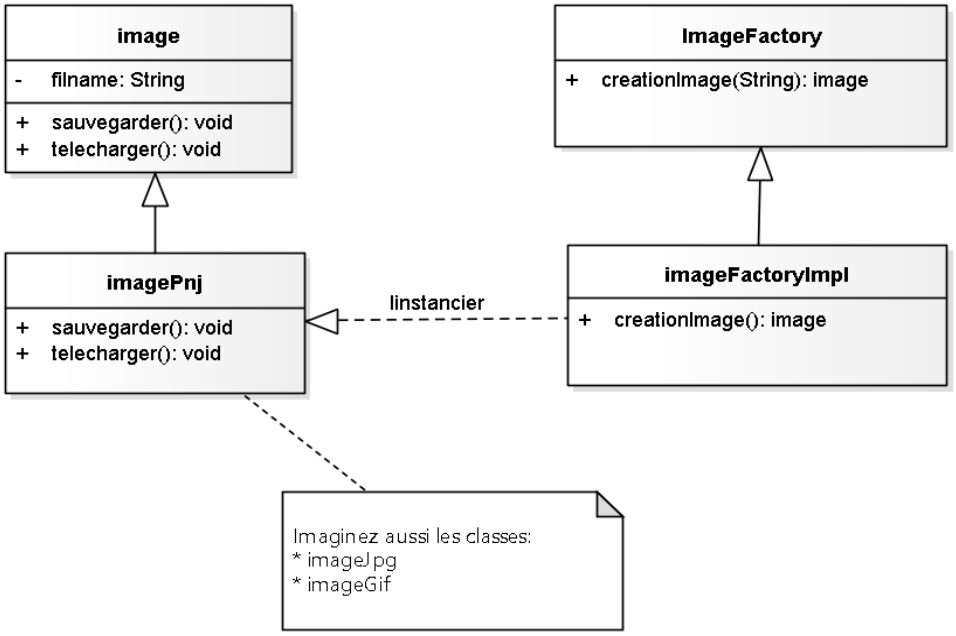
Creator est une class abstrait.

Plusieurs fabriques peuvent être regroupées en une fabrique abstraite Permettant d'instancier des objets dérivant de plusieurs types abstraits différents.

Les Produit étant en général uniques dans un programme, on utilise souvent le patron de conception singleton Pour les implémenter.




Structure appliquer pour Gestionnaire des donneurs du sang :




La définition abstraite du produit :

```java
public abstract class Image {
    private String filename;
    /** Le constructeur de la classe Image */
    public Image(String filename) {
        this.filename = filename;
    }
    /** Pour récupérer le nom du fichier associé à l'image */
    public String getFilename() {
        return filename;
    }
    /** Permet de charger en mémoire l'image */
    public abstract void loadImage();
    /** Permet de sauvegarder l'image sur le disque */
    public abstract void saveImage();
    /** Un petit toString pour les affichages sur la console */
    @Override
    public String toString() {
        String className = this.getClass().getName();
        return className + "image : " + filename;
    }
```

Définir une classe dérivée pour chaque format d'images supporté par l'application :

```java
public class ImagePng extends Image {
    public ImagePng(String filename) {
        super(filename);
    }
    @Override
    public void loadImage() {
        System.out.println( "image au format PNG" );
    }
    @Override
    public void saveImage() {
        System.out.println( "image au format PNG" );
    }
```

La définition de notre interface de fabrique:

```java
public interface ImageFactory {Image createImage( String filename );}
```

La classe d'implémentation pour notre fabrique d'images :

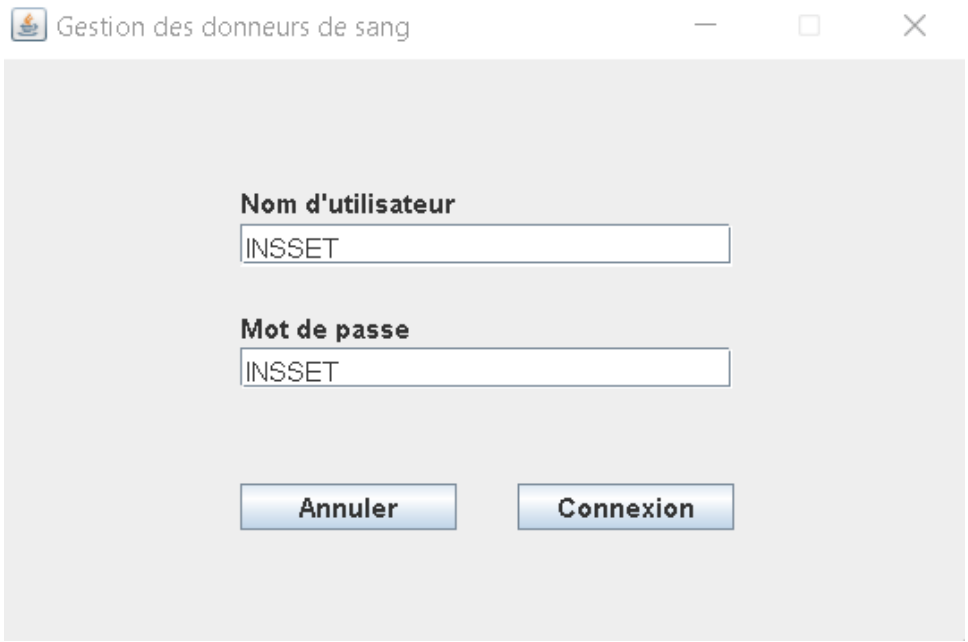
```java
public class ImageFactoryImpl implements ImageFactory {
    @Override
    public Image createImage(String filename) {
        String extention = filename.substring( filename.lastIndexOf(".") );
        Image image = switch( extention.toLowerCase() ) {
            case ".gif" -> new ImageGif( filename );
            case ".jpg" -> new ImageJpg( filename );
            case ".png" -> new ImagePng( filename );
            default -> throw new RuntimeException( "Format " + extention + " not supported" );
        };
        image.loadImage();
        return image;
```

Utilisation de notre « factory »

```java
public class GestionnaireDonSang {

    public static void main(String[] args) {

        ImageFactory factory = new ImageFactoryImpl();
        Image image1 = factory.createImage( "essai.png" );
        Image image2 = factory.createImage( "essai.jpg" );

        System.out.println( image1 );
        System.out.println( image2 );
    }

}
```

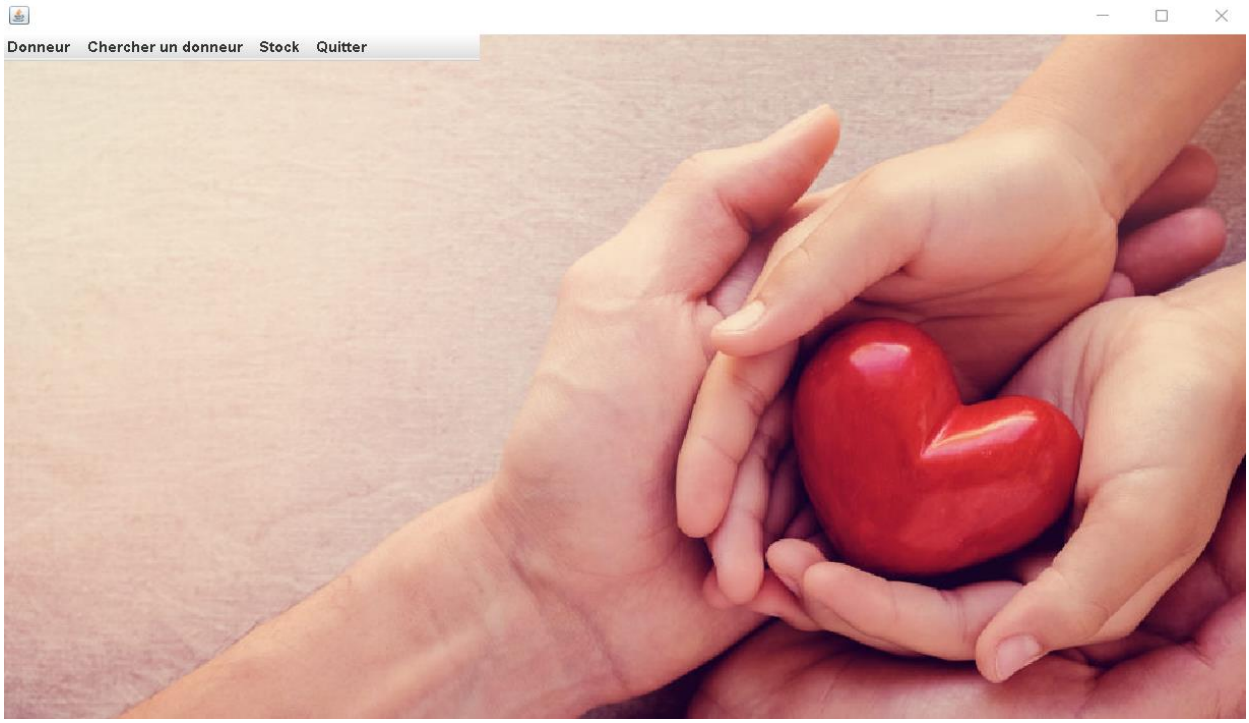
## Les Interfaces

Connection de l'admin



```java
JButton btnConnexion = new JButton("Connexion");
btnConnexion.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        // Valider la connexion: user---> INSSET  mot de passe---> INSSET
        if( txtUser.getText().equals("INSSET") && txtPassword.getText().equals("INSSET")) {
            Acceuil Acceuil = new Acceuil();
            Acceuil.setVisible(true);
            dispose();
        }else{
                    JOptionPane.showMessageDialog(null, " Le nom d'utilisateur ou le mot de passe est incorrect ");
                }
    }
});
```

La vue Accueil du programme :

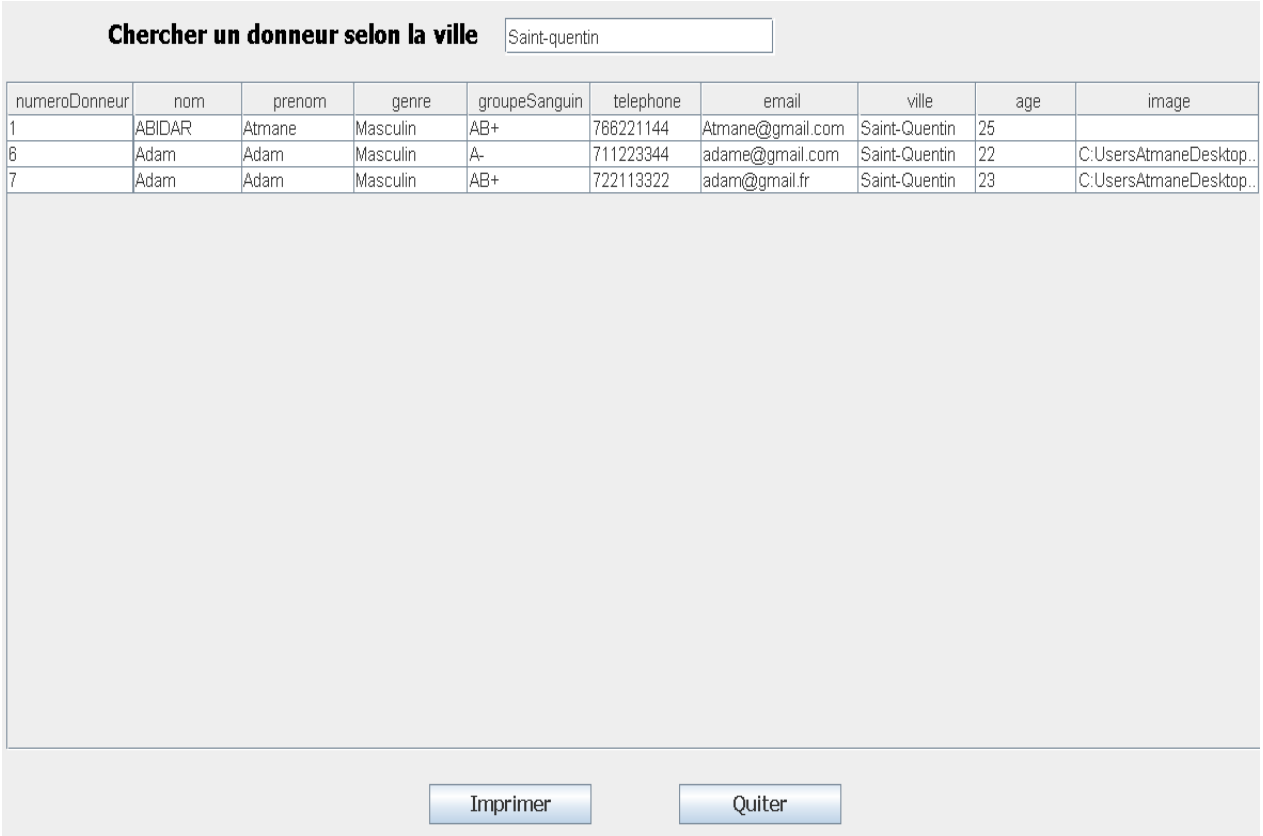



La vue Ajouter un donneur :

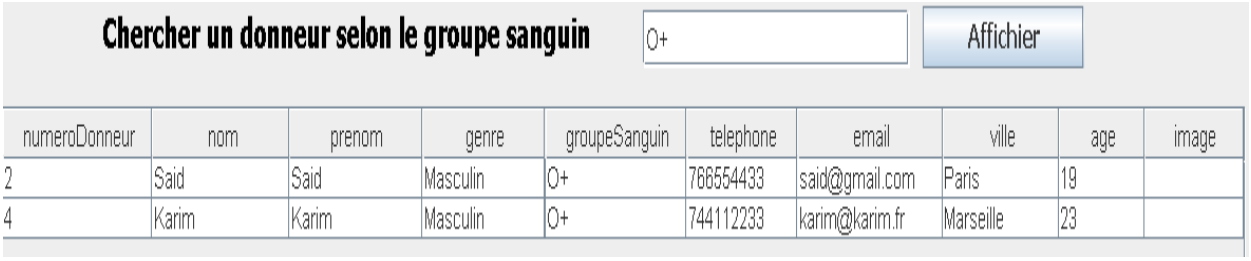

La base de donner :

| numeroDonneur | nom | prenom | genre | groupeSanguin | telephone | email | ville | age | image |
|---|---|---|---|---|---|---|---|---|---|
| 1 | ABIDAR | Atmane | Masculin | AB+ | 766221144 | Atmane@gmail.com | Saint-Quentin | 25 | |
| 2 | Said | Said | Masculin | O+ | 766554433 | said@gmail.com | Paris | 19 | |
| 3 | Sara | Sara | Féminin | A- | 788996633 | sara@gmail.fr | Nantes | 22 | |
| 4 | Karim | Karim | Masculin | O+ | 744112233 | karim@karim.fr | Marseille | 23 | |
| 5 | Lina | Lina | Féminin | A+ | 744112233 | lina@gmail.com | Caen | 20 | |
| 6 | Adam | Adam | Masculin | A- | 711223344 | adame@gmail.com | Saint-Quentin | 22 | C:UsersAtmaneDesktop estImage.png |
| 7 | Adam | Adam | Masculin | AB+ | 722113322 | adam@gmail.fr | Saint-Quentin | 23 | C:UsersAtmaneDesktop estImage.png |

La vue Chercher donneur selon la ville :




La vue Chercher donneur selon le groupe Sanguin :




Conclusion :

Ce programme aide à gérer les aspects des besoins en systèmes de données informatiques des centres de collecte et des banques de sang, Je souhaite travailler encore sur l'amélioration de ce projet, le but c'est qu'il soit capable du recrutement des donneurs, la livraison et la traçabilité des produits sanguins aux patients.