

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE
MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE
SCIENTIFIQUE

UNIVERSITÉ DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE



RAPPORT

JAVA PROJECT

JAVA ONLY

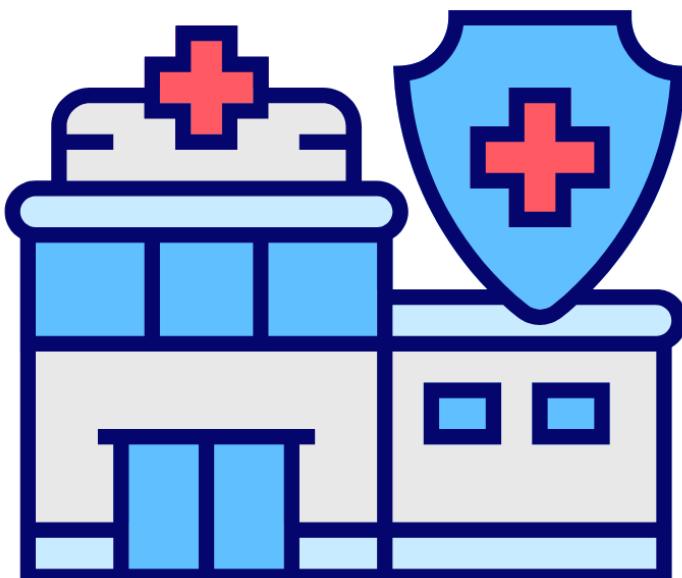
OUALIHINE DOUAA CHARAZED
BOUDJELAL MARIA
DADI SOUMIA
RACHEDI MERIEM

05 / 2024



SOMMAIRE :

1 - Introduction	03
2 - Diagramme de classes	04
3 - Explication du contenu des classes.....	06
4 - Conclusion	32



1 - INTRODUCTION :

Ce rapport concerne un projet JAVA qui gère la gestion d'un cabinet médical.

L'objectif principal de ce projet est de développer une sorte d'application informatique en JAVA qui facilite au patients de voir leurs dossier médicale et prendre des rendez-vous et facilite au médecin de gérer ses patients sans confondre les informations et être plus organiser.

Ce rapport expliquera bien tous le code JAVA de cette gestion de cabinet médical.

2 - DIAGRAMME DE CLASSES :

Patient

```

private String nom;
private String prenom;
private String email;
private String gender;
private String maladies;
private String adresse;
private int numtel;
private String immatricule;
private Date birthdate;
private static int compteurMatricule = 0;

public Patient();
public Patient(String nom, String prenom, String email,
String gender, String maladies, int numtel, Date
birthdate, String adresse);
private String genererMatricule();
public void afficher();
public void ajoutPatient(List patientList);
public static boolean obtenirImmatriculesParNom(String
nomRecherche, List patientList);
public static Patient recherchePatient (String immatricule,
List patientList);
public static boolean suppPatient (String immatricule, List
patientList);
public boolean modifinfo (List patientList, Patient patient2);
    
```

Dossier

```

private Patient patient;
private String groupeSanguin;
public class Consultations implements
Serializable() //Pour créer un type Consultation
private String diagnostic;
private Date date;
private List medicamentsPrescrits;
private ArrayList consultation;

Consultations();
public Consultations(String diagnostic, Date date, List
medicamentsPrescrits);
Dossier();
public Dossier(Patient patient, String groupeSanguin,
ArrayList consultation);
public Dossier(Patient patient);
public void ajouterMedicamentPrescrit(String
medicament);
public void afficher();
public void ajouterConsultations(ArrayList LC,
Consultations consultation);
public void supprimerConsultations(ArrayList LC, Date
dateConsultation);
public void rechercherConsultationsParDate(ArrayList
LC, Date dateConsultation);
public boolean ajoutConsultADossier(ArrayList LD,
Patient patient, Consultations consultation);
public boolean afficher();
public void ajouterDossier(ArrayList LD, Dossier
dossier);
public boolean supprimerDossier(ArrayList LD, Patient
patient);
public boolean rechercherDossier(ArrayList LD);
public static boolean modifPatient(ArrayList LD, Patient
patient1, Patient patient2);
    
```

Medecin

```

private String nom;
private String prenom;
private String status;

public Medecin();
public Medecin(String nom, String prenom, String
status);
public Medecin(String status);
    
```

Rendezvous

```
private Patient patient;
private Medecin medecin;
private Date dateRendezvous;
private String heure;

public Rendezvous();
public Rendezvous(Patient patient, Medecin medecin,
Date dateRendezvous, String heure);
public Rendezvous(Patient patient);
public Rendezvous(Date dateRendezvous);
public Rendezvous(Date dateRendezvous);
public void afficher();
public static void ajouterRen(ArrayList listeRendezvous,
Rendezvous nouveauRendezvous);
public boolean supprimerRen(ArrayList listeRendezvous);
public boolean modifierRen(ArrayList listeRendezvous,
Date dateRendezvous, String heure);
public boolean afficherListeRen(ArrayList
listeRendezvous);
public boolean rechercherrendezvous(ArrayList
listeRendezvous);
public boolean rechercherrendezvousM(ArrayList
listeRendezvous);
public boolean rechercherrendezvousDate(ArrayList
listeRendezvous);
public static boolean modifPatient(ArrayList
listeRendezvous, Patient patient1, Patient patient2);
```

SessionCompte

```
private String utilisateur;
private String mdp;
private Patient patient;

public SessionCompte();
public SessionCompte(String utilisateur, String mdp);
public SessionCompte(String utilisateur, String mdp,
Patient patient);
public boolean ajoutCompte(List listeCompte,
SessionCompte compte);
public boolean supprimerCompte(List listeCompte, String
utilisateur);
public boolean modifInfoC(List listeCompte, Patient
patientToReplace, Patient newPatient);
public Patient rechercherPatientCompte(List
listeCompte);
public boolean rechercheUtil(List listeCompte, String
utilisateur);
public boolean verifMDP(List listeCompte, String
utilisateur, String mdp);
public boolean connexion(List listeCompte);
public boolean creerCompte(List listeCompte);
```

DataHandler

```
// Méthode pour sauvegarder les données
```

```
public static void saveData(ArrayList<Dossier> LD,
ArrayList<Rendezvous> LR, List<Patient> LP,
List<SessionCompte> LC, String fileName) ;
public static void loadData(ArrayList<Dossier> LD,
ArrayList<Rendezvous> LR, List<Patient> LP,
List<SessionCompte> LC, String fileName);
```

3- EXPLICATION DU CONTENUE DES CLASSES :

1- Classe Patient :

1.1-Les attributs :

```
private String nom;  
private String prenom;  
private String email;  
private String gender;  
private String maladies;  
private String adresse;  
private int numtel;  
private String immatricule;  
private Date birthdate;  
private static int compteurMatricule = 0;
```

1.2-Les constructeurs :

```
public Patient() {  
//Constructeur par défaut permet de créer un patient  
avec un matricule unique sans fournir manuellement  
toutes les informations. }
```

```
public Patient(String nom, String prenom, String email, String gender, String maladies, int numtel, Date birthdate, String adresse) {  
    //Permet de créer un patient avec des détails spécifiques tels que le nom, le prénom, l'email, etc., en plus de générer automatiquement un matricule unique.  
}
```

1.3-Les méthodes :

```
private String genererMatricule() {  
    //Cette méthode est la responsable de la génération des immatricules des patients }  
public void afficher() {  
    //Cette méthode affiche les informations du patient sur la console, elle facilite la visualisation des détails d'un patient pour le personnel }  
public void ajoutPatient(List<Patient> patientList) {  
    //Cette méthode ajoute un nouveau patient à une liste de patients fournie en paramètre. Permet d'ajouter un patient à la base de données du système }
```

```
public static boolean obtenirImmatriculesParNom  
(String nomRecherche, List patientList) {  
//Cette méthode recherche et affiche les matricules  
des patients ayant un nom spécifique dans une liste de  
patients donnée.-Fonctionnement : Parcourt la liste de  
patients, compare les noms et affiche les matricules  
correspondants.-Utilité : Aide à retrouver les patients  
en fonction de leur nom. }
```

```
public static Patient recherchePatient (String  
immatricule, List patientList) {  
//Cette méthode recherche un patient par son  
matricule dans une liste donnée et affiche ses  
informations.-Utilité : Permet de retrouver rapidement  
les détails d'un patient à partir de son matricule. }
```

```
public static boolean suppPatient (String immatricule,  
List patientList) {  
//Cette méthode supprime un patient de la liste en  
fonction de son matricule.-Fonctionnement : Utilise un  
itérateur pour parcourir la liste et supprimer le patient  
trouvé.-Utilité : Permet de supprimer les  
enregistrements des patients du système. }
```

```
public boolean modifinfo (List patientList, Patient  
patient2) {  
//Cette méthode modifie les informations d'un patient  
existant en remplaçant ses données par celles d'un  
autre patient fourni.-Utilité : Facilite la mise à jour des  
informations des patients enregistrés. }
```

1.4-Fonctionnement général et logique:

- La classe utilise des variables d'instance pour stocker les détails individuels du patient.
- Les constructeurs permettent d'instancier des objets Patient avec différentes configurations.
- Les méthodes offrent des fonctionnalités pour manipuler les objets Patient, y compris l'ajout, la recherche, la modification et la suppression.
- La logique des méthodes repose sur la manipulation des listes de patients à l'aide de boucles, d'itérateurs et de comparaisons de chaînes.
- Les méthodes de recherche et de suppression utilisent les matricules comme identifiants uniques pour identifier les patients.
- La classe intègre des mécanismes pour gérer les cas où la liste de patients est vide ou lorsque le patient recherché n'est pas trouvé.

2- Classe Dossier :

2.1- Les Attributs :

```
private Patient patient; //Le patient à le quelle appartient le dossier
```

```
private String groupeSanguin;
```

```
public class Consultations implements Serializable0  
//Pour créer un type Consultation
```

- private String diagnostic;
- private Date date;
- private List<String> medicamentsPrescrits;

```
private ArrayList<Consultations> consultation; //La liste de tous les consultations du patient
```

2.2- Les constructeurs :

```
Consultations(){
```

```
    // Constructeur par défaut qui initialise une consultation sans spécifier de détails.
```

```
    // Utilité : Permet de créer une consultation vide qui peut être complétée ultérieurement.
```

```
}
```

```
public Consultations(String diagnostic, Date date,
List<String> medicamentsPrescrits) {
    //Constructeur utilisé pour créer une consultation
    avec un diagnostic, une date et une liste de
    médicaments prescrits.

    //Utilité : Permet de créer une consultation complète
    avec toutes les informations nécessaires.
}

Dossier0{
    //Constructeur par défaut qui initialise un dossier
    sans spécifier de patient ou de consultations.

    //Utilité : Permet de créer un dossier vide qui peut
    être complété ultérieurement.
}

public Dossier(Patient patient, String groupeSanguin,
ArrayList<Dossier.Consultations> consultation) {
    //Constructeur utilisé pour créer un dossier avec un
    patient spécifique, son groupe sanguin et une liste de
    consultations.

    //Utilité : Permet de créer un dossier complet
    contenant les informations d'un patient ainsi que ses
    consultations.}
```

```
public Dossier(Patient patient) {  
    //Constructeur utilisé pour créer un dossier avec  
    seulement un patient spécifique, sans spécifier de  
    groupe sanguin ni de consultations.  
    //Utilité : Utile lorsqu'il est nécessaire de créer un  
    dossier pour un patient sans ajouter immédiatement  
    des consultations ou un groupe sanguin.  
}
```

2.3- Les méthodes :

```
public void ajouterMedicamentPrescrit(String  
medicament) {  
    //Ajoute un nouveau médicament prescrit à la liste  
    des médicaments prescrits pour cette consultation.  
    //Utilité : Permet d'ajouter des médicaments à la  
    prescription médicale lors de la consultation.  
}  
  
public void afficher() {  
    //Affiche les détails de la consultation, y compris le  
    diagnostic, la date et les médicaments prescrits.  
    //Utilité : Fournit une vue détaillée des informations  
    de la consultation pour référence médicale.  
}
```

```
public void ajouterConsultations
(ArrayList<Consultations> LC, Consultations
consultation) {
    //Ajoute une nouvelle consultation à une liste de
    consultations donnée.
    //Utilité : Permet d'ajouter les consultations à la liste
    des consultations dans le dossier médical.
}
public void supprimerConsultations
(ArrayList<Consultations> LC, Date dateConsultation) {
    //Supprime une consultation de la liste en fonction
    de la date de la consultation.
    //Utilité : Facilite la suppression des consultations
    antérieures ou spécifiques.
}
public void rechercherConsultationsParDate
(ArrayList<Consultations> LC, Date dateConsultation) {
    //Recherche et affiche les consultations pour une
    date spécifique dans une liste de consultations donnée.
    //Utilité : Permet de retrouver rapidement les
    consultations pour une date donnée dans le dossier
    médical.
}
```

```
public boolean ajoutConsultADossier  
(ArrayList<Dossier> LD, Patient patient, Consultations  
consultation) {  
    //Ajoute une nouvelle consultation au dossier d'un  
    patient spécifique dans une liste de dossiers donnée.  
    //Utilité : Permet d'enregistrer les consultations  
    médicales dans le dossier approprié du patient. }  
public boolean afficher0 {  
    //Affiche les détails du dossier, y compris le groupe  
    sanguin et les consultations.  
    //Utilité : Fournit une vue détaillée des informations  
    médicales du patient pour le personnel médical. }  
public void ajouterDossier(ArrayList<Dossier> LD,  
Dossier dossier) {  
    //Ajoute un nouveau dossier à une liste de dossiers  
    donnée.  
    //Utilité : Permet d'ajouter de nouveaux dossiers au  
    système de gestion des dossiers médicaux. }  
public boolean supprimerDossier(ArrayList<Dossier>  
LD, Patient patient) {  
    //Supprime le dossier d'un patient spécifique de la  
    liste de dossiers.  
    //Utilité : Facilite la suppression des dossiers des  
    patients qui ne sont plus suivis ou qui quittent le  
    système. }
```

```
public boolean rechercherDossier(ArrayList<Dossier>
LD) {
    //Recherche et affiche le dossier d'un patient
    spécifique dans la liste de dossiers.
    //Utilité : Permet de retrouver rapidement les
    informations médicales d'un patient enregistré.
}
public static boolean modifPatient(ArrayList<Dossier>
LD, Patient patient1, Patient patient2) {
    //Modifie les informations du patient dans le dossier,
    en remplaçant le patient spécifié par un nouveau
    patient.
    //Utilité : Facilite la mise à jour des informations des
    patients dans leurs dossiers.
}
```

2.4- Fonctionnement général et logique :

Consultation :

- La classe Consultations utilise des variables d'instance pour stocker les détails spécifiques de chaque consultation, tels que le diagnostic, la date et les médicaments prescrits.
- Les constructeurs permettent de créer des consultations avec différentes configurations, en fonction des informations disponibles.
- Les méthodes offrent des fonctionnalités pour manipuler les consultations, y compris l'ajout, la suppression et la recherche.
- La logique des méthodes repose sur la manipulation des listes de consultations à l'aide de boucles, de comparaisons et de méthodes de liste.
- Les méthodes de recherche et de suppression utilisent la date de la consultation comme critère pour identifier les consultations à manipuler.
- La classe intègre des mécanismes pour gérer les cas où la liste de consultations est vide ou lorsque la consultation recherchée n'est pas trouvée.

Dossier :

- La classe utilise des variables d'instance pour stocker les informations du patient, y compris son groupe sanguin et ses consultations.
- La classe Consultations est une classe interne utilisée pour représenter les détails de chaque consultation médicale.
- Les constructeurs permettent de créer des dossiers avec différentes configurations, en fonction des informations disponibles.
- Les méthodes offrent des fonctionnalités pour manipuler les dossiers et les consultations, y compris l'ajout, la suppression, la recherche et la modification.
- La logique des méthodes repose sur la manipulation des listes de dossiers et de consultations à l'aide de boucles, de comparaisons et de méthodes de listes.
- Les méthodes de recherche et de suppression utilisent l'identifiant unique du patient pour identifier les dossiers à manipuler.
- La classe intègre des mécanismes pour gérer les cas où la liste de dossiers est vide ou lorsque le dossier recherché n'est pas trouvé.

3- Classe Rendezvous :

3.1- Les Attributs :

```
private Patient patient;  
private Medecin medecin;  
private Date dateRendezvous;  
private String heure;
```

3.2- Les constructeurs :

```
public Rendezvous() {  
    //Constructeur par défaut qui crée un rendez-vous  
    sans spécifier de détails.  
    //Utilité : Permet d'instancier un rendez-vous vide qui  
    peut être complété ultérieurement.  
}  
public Rendezvous(Patient patient, Medecin medecin,  
Date dateRendezvous, String heure) {  
    //Constructeur utilisé pour créer un rendez-vous  
    avec toutes les informations nécessaires.  
    //Utilité : Permet de créer un rendez-vous complet  
    avec les détails du patient, du médecin, de la date et  
    de l'heure.  
}
```

```
public Rendezvous(Patient patient) {}  
public Rendezvous(Date dateRendezvous) {}  
public Rendezvous(Date dateRendezvous) {
```

//Ces constructeurs sont utilisés pour créer des rendez-vous avec une seule information spécifiée (patient, médecin ou date).

//Utilité : Offre une flexibilité pour créer des rendez-vous en spécifiant uniquement une partie des informations.}

3.3- Les méthodes :

```
public void afficher() {
```

//Affiche les détails du rendez-vous, y compris le patient, le médecin, la date et l'heure.

//Utilité : Fournit une vue détaillée des informations du rendez-vous.}

```
public static void ajouterRen(ArrayList<Rendezvous>  
listeRendezvous, Rendezvous nouveauRendezvous) {
```

//Ajoute un nouveau rendez-vous à une liste de rendez-vous donnée.

//Utilité : Permet d'ajouter de nouveaux rendez-vous à la planification médicale.}

```
public boolean supprimerRen(ArrayList<Rendezvous>  
listeRendezvous) {
```

//Supprime un rendez-vous de la liste en fonction de la date et de l'heure du rendez-vous.

//Utilité : Facilite la gestion et la suppression des rendez-vous planifiés.}

```
public boolean modifierRen(ArrayList<Rendezvous>
listeRendezvous, Date dateRendezvous, String heure) {
    //les argument représente les information du
rendezvous qu'on veut changé
    //Modifie les détails d'un rendez-vous dans une liste
en fonction de la date et de l'heure du rendez-vous.
    //Utilité : Permet de mettre à jour les informations
d'un rendez-vous spécifique.
}
public boolean afficherListeRen
(ArrayList<Rendezvous> listeRendezvous) {
    - //Affiche une liste de rendez-vous.
    //Utilité : Permet de visualiser tous les rendez-vous
planifiés dans une liste donnée.
}
public boolean rechercherrendezvous
(ArrayList<Rendezvous> listeRendezvous) {
    //Recherche et affiche les rendez-vous d'un patient
donné.
    //Utilité : Facilite la recherche des rendez-vous d'un
patient spécifique dans la planification médicale.
}
```

```
public boolean rechercherrendezvousM  
(ArrayList<Rendezvous> listeRendezvous) {  
    //Recherche et affiche les rendez-vous d'un médecin  
    donné.  
    //Utilité : Facilite la recherche des rendez-vous  
    associés à un médecin spécifique dans la planification  
    médicale.  
}  
public boolean rechercherrendezvousDate  
(ArrayList<Rendezvous> listeRendezvous) {  
    //Recherche et affiche les rendez-vous pour une  
    date spécifique.  
    //Utilité : Permet de retrouver rapidement les  
    rendez-vous planifiés pour une date donnée.  
}  
public static boolean modifPatient  
(ArrayList<Rendezvous> listeRendezvous, Patient  
patient1, Patient patient2) {  
    //Modifie le patient associé à un rendez-vous.  
    //Utilité : Permet de mettre à jour les informations du  
    patient associé à un rendez-vous.  
}
```

3.4- Fonctionnement général et logique :

- La classe Rendezvous permet de représenter et de gérer les rendez-vous médicaux entre les patients et les médecins.
- Les constructeurs offrent une flexibilité pour créer des rendez-vous avec différentes configurations en fonction des informations disponibles.
- Les méthodes fournissent des fonctionnalités pour ajouter, supprimer, modifier et rechercher des rendez-vous dans une liste donnée.
- La logique des méthodes repose sur la manipulation des listes de rendez-vous à l'aide de boucles, de comparaisons et de méthodes de liste.
- Les méthodes de recherche utilisent différents critères tels que le patient, le médecin ou la date pour identifier les rendez-vous à afficher.
- La classe intègre des mécanismes pour gérer les cas où la liste de rendez-vous est vide ou lorsque le rendez-vous recherché n'est pas trouvé.

4- Classe Medecin :

4.1- Les Attributs :

```
private String nom;  
private String prenom;  
private String status;
```

4.2- Les constructeurs :

```
public Medecin() {  
    //Constructeur par défaut qui crée un objet  
    Medecin sans spécifier de détails.  
    //Utilité : Permet d'instancier un médecin vide qui  
    peut être complété ultérieurement.  
}  
public Medecin(String nom, String prenom, String  
status) {  
    //Constructeur utilisé pour créer un médecin avec  
    toutes les informations nécessaires.  
    //Utilité : Permet de créer un médecin complet avec  
    les détails du nom, du prénom et du statut.  
}  
public Medecin(String status) {  
    //Constructeur utilisé pour créer un médecin en  
    spécifiant uniquement le statut.  
    //Utilité : Offre une flexibilité pour créer des  
    médecins avec une partie des informations spécifiées.  
}
```

4.3- Fonctionnement général et logique :

Ses données sont importantes à la classe Rendezvous

5- Classe SessionCompte :

5.1- Les Attributs :

private String utilisateur;

private String mdp;

private Patient patient;

5.2- Les constructeurs :

public SessionCompte0{

//Constructeur par défaut qui crée un objet
SessionCompte sans spécifier de détails.

//Utilité : Permet d'instancier un objet
SessionCompte vide qui peut être configuré
ultérieurement.}

public SessionCompte(String utilisateur, String mdp) {

//Constructeur utilisé pour créer un objet
SessionCompte avec un nom d'utilisateur et un mot de
passe.

//Utilité : Facilite la création d'un compte en
spécifiant les informations essentielles.}

public SessionCompte(String utilisateur, String mdp,
Patient patient) {

//Constructeur utilisé pour créer un objet
SessionCompte avec un nom d'utilisateur, un mot de
passe et un patient associé.

//Utilité : Permet de créer un compte lié à un patient
spécifique.}

5.3- Les méthodes :

```
public boolean ajoutCompte(List<SessionCompte>
listeCompte, SessionCompte compte) {
    //Ajoute un compte à la liste des comptes.
    //Utilité : Permet de sauvegarder un nouveau compte
dans la liste des comptes disponibles.}
public boolean supprimerCompte(List<SessionCompte>
listeCompte, String utilisateur) {
    //Supprime un compte de la liste des comptes en
fonction du nom d'utilisateur.
    //Utilité : Permet de supprimer un compte spécifique de
la liste des comptes.}
public boolean modifInfoC(List<SessionCompte>
listeCompte, Patient patientToReplace, Patient
newPatient) {
    //Modifie les informations du patient associé à un
compte spécifique.
    //Utilité : Permet de mettre à jour les informations du
patient associé à un compte.}
public Patient rechercherPatientCompte
(List<SessionCompte> listeCompte) {
    //Recherche et renvoie le patient associé à un compte
spécifique.
    //Utilité : Permet d'obtenir le patient associé à un
compte donné.}
```

```
public boolean rechercheUtil(List<SessionCompte>
listeCompte, String utilisateur){
    //Vérifie si un nom d'utilisateur existe déjà dans la liste
    des comptes.
    //Utilité : Permet de vérifier l'existence d'un nom
    d'utilisateur avant la création d'un nouveau compte.}
public boolean verifMDP(List<SessionCompte>
listeCompte, String utilisateur, String mdp){
    //Vérifie si le mot de passe correspond à celui associé à
    un nom d'utilisateur spécifique.
    //Utilité : Permet de vérifier si le mot de passe entré
    est correct pour un nom d'utilisateur donné.}
public boolean connexion(List<SessionCompte>
listeCompte){
    //Connecte l'utilisateur en vérifiant d'abord son
    existence puis en vérifiant le mot de passe.
    //Utilité : Permet à un utilisateur de se connecter à son
    compte.}
public boolean creerCompte(List<SessionCompte>
listeCompte){
    //Crée un nouveau compte après avoir vérifié
    l'existence du nom d'utilisateur et la longueur du mot de
    passe.
    //Utilité : Permet à un utilisateur de créer un nouveau
    compte.}
```

5.4- Fonctionnement général et logique :

- La classe `SessionCompte` offre des méthodes pour créer, modifier, supprimer et gérer des comptes utilisateur.
- Les méthodes de cette classe sont conçues pour être utilisées dans un environnement de gestion médicale, où les comptes sont associés à des patients.
- Elle assure la sécurité des comptes en vérifiant l'existence des noms d'utilisateur et la correspondance des mots de passe.
- Les constructeurs permettent de créer des objets `SessionCompte` avec différentes configurations en fonction des informations disponibles.

6- Classe DataHandler :

La classe `DataHandler` est conçue pour gérer la sauvegarde et le chargement des données d'un système de gestion médicale. Elle utilise la sérialisation pour stocker les données dans des fichiers binaires. Voici une analyse détaillée de chaque aspect de la classe :

6.1- Méthodes :

```
public static void saveData (ArrayList<Dossier>
LD, ArrayList<Rendezvous> LR, List<Patient> LP,
List<SessionCompte> LC, String fileName) {
```

//Sauvegarde les données dans un fichier binaire spécifié.

//Utilité : Permet de stocker les dossiers, les rendez-vous, les patients et les comptes dans un fichier pour une utilisation ultérieure.}

```
public static void loadData(ArrayList<Dossier>
LD, ArrayList<Rendezvous> LR, List<Patient> LP,
List<SessionCompte> LC, String fileName) {
```

//Charge les données à partir d'un fichier binaire spécifié.

//Utilité : Permet de récupérer les données précédemment sauvegardées pour les utiliser dans l'application.}

6.2- Fonctionnement général et logique :

- La classe `DataHandler` utilise des flux d'entrée/sortie d'objets pour sérialiser et désérialiser les données.
- Les données sont sauvegardées et chargées dans des fichiers binaires, ce qui permet de les stocker de manière sécurisée et efficace.
- Les méthodes `saveData` et `loadData` acceptent des listes d'objets correspondant aux différents types de données du système : dossiers, rendez-vous, patients et comptes.
- En cas de problème lors de la sauvegarde ou du chargement des données, des messages d'erreur appropriés sont affichés pour informer l'utilisateur.

6.3- Utilisation :

- Pour sauvegarder les données, l'utilisateur appelle la méthode `saveData` en spécifiant les listes d'objets à sauvegarder et le nom du fichier.
- Pour charger les données, l'utilisateur appelle la méthode `loadData` en spécifiant les listes d'objets à remplir et le nom du fichier contenant les données sauvegardées.

En conclusion, la classe `DataHandler` offre des fonctionnalités essentielles pour la sauvegarde et le chargement des données d'un système de gestion médicale, facilitant ainsi la persistence des données entre les sessions d'utilisation de l'application.

Le Main :

La classe `Main` est l'entrée principale de l'application de gestion médicale. Voici son fonctionnement et sa logique :

1- Initialisation des listes et chargement des données :

- Au début du programme, les listes pour les dossiers, les rendez-vous, les patients et les comptes sont créées et vidées.
- Ensuite, la méthode `loadData` de la classe `DataHandler` est appelée pour charger les données sauvegardées depuis le fichier "data.ser" (s'il existe).

2- Interface utilisateur :

- L'utilisateur est invité à choisir son rôle parmi patient (P), médecin (M), ou secrétaire (S).
- Selon le rôle choisi, différentes actions sont proposées à l'utilisateur.

3- Interface patient :

- Si l'utilisateur est un patient, il est invité à créer un compte (S) ou à se connecter (L).
- S'il choisit de créer un compte, il lui est demandé de saisir ses informations personnelles et de les associer à un compte utilisateur.
- S'il choisit de se connecter, il doit saisir son nom d'utilisateur et son mot de passe pour accéder à son compte.

4- Services pour les patients :

- Après la connexion ou la création du compte, le patient peut accéder à différents services tels que l'affichage de ses informations, de son dossier médical, la prise de rendez-vous, etc.
- Les services sont implémentés dans une boucle while qui continue jusqu'à ce que l'utilisateur décide de se déconnecter.

5- Interface médecin :

- Si l'utilisateur est un médecin, il est invité à saisir ses informations (nom, prénom, statut principal ou remplaçant).
- Ensuite, le médecin peut choisir parmi différents services tels que la création de dossiers pour les patients, la recherche et la suppression de dossiers, etc.
- Les services sont également implémentés dans une boucle while qui continue jusqu'à ce que le médecin décide de se déconnecter.

6- Interface secrétaire :

- Si l'utilisateur est un secrétaire, il peut accéder à des services tels que l'ajout de rendez-vous, la suppression de patients, etc.
- Les services sont proposés dans une boucle while jusqu'à ce que le secrétaire décide de se déconnecter.

7- Sauvegarde des données :

- À la fin du programme, les données sont sauvegardées dans le fichier "data.ser" en appelant la méthode `saveData` de la classe `DataHandler`.

4 - CONCLUSION :

La classe `Main` de l'application de gestion médicale offre une interface utilisateur intuitive et propose une gamme complète de services pour les patients, les médecins et les secrétaires. Elle garantit également la sauvegarde des données pour une utilisation ultérieure.