

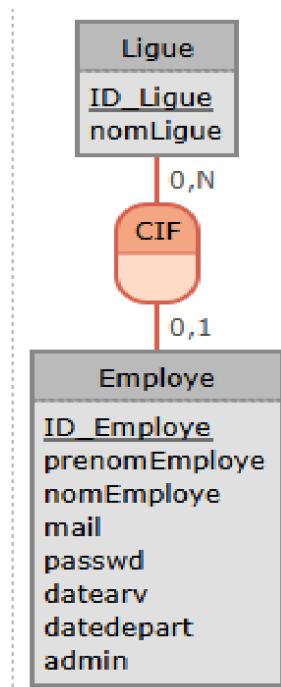
AP-Personnel

Ce projet a consisté à l'amélioration d'une application Java en y ajoutant des nouvelles fonctionnalité tels que :

- Gestion de l'administrateur en ligne de commande .
- La gestion des dates avec leurs Exception
- Ajout de Test Unitaires afin de s'assurer du bon fonctionnement des méthodes
- La gestion des employé (Modifications , Sélection)
- La création d'une base de données liant l'application a une base de donnée(MCD , MLD, MPD)

1) Création de la base de données du MCD au MPD :

a. Le MCD :

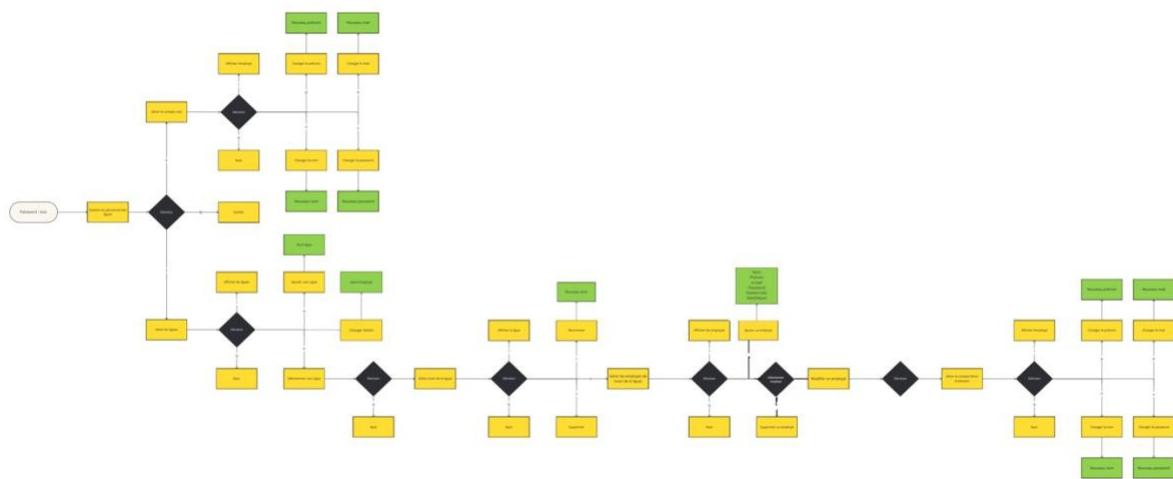


b. Le script de création de se MCD :

```
1  CREATE TABLE Ligue (
2      ID_Ligue INT,
3      nomLigue VARCHAR(100) NOT NULL,
4      ADD CONSTRAINT PK_Ligue PRIMARY KEY (ID_Ligue)
5  )engine = innodb;
6
7  CREATE TABLE Employe (
8      ID_Employe INT,
9      prenomEmploye VARCHAR(50) NOT NULL,
10     nomEmploye VARCHAR(50) NOT NULL,
11     mail VARCHAR(100) UNIQUE NOT NULL,
12     passwd VARCHAR(50) NOT NULL,
13     dateavv DATE NOT NULL,
14     datedepart DATE,
15     Admin BOOLEAN NOT NULL,
16     ID_Ligue INT,
17     ADD CONSTRAINT PK_Employe PRIMARY KEY (ID_Employe)
18 )engine = innodb;
19
20 ALTER TABLE Employe
21 ADD CONSTRAINT fk_Ligue_employe
22 FOREIGN KEY (ID_Ligue)
23 REFERENCES Ligue(ID_Ligue);
```

2) L'applications Java :

Arbre Heuristique de l'applications (lien vers une meilleur qualité :
<https://github.com/WassimElArche/Iteration2/blob/main/Maquette.pdf>) :



Le visuel du dialogue Client / Machine sur le terminal :



```
Console < 
PersonnelConsole (1) [Java Application] /Library/Internet Plug-Ins/JavaAppletPlugin.plugin/Contents/Home/bin/java (23 nov. 2024, 21:12:08) [pid: 2705]
password : toor
Gestion du personnel des ligues
c : Gérer le compte root
l : Gérer les ligues
q : Quitter
Select an option :
```

a) Tests unitaires

L'ajout des Test unitaire pour la gestion des dates avec leurs exceptions :

```
    @Test
    void testInvalidDates() throws SauvegardeImpossible {
        Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
        Exception exception = assertThrows(Errurdate.class, () -> {
            ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , LocalDate.of(2023, 12, 31), LocalDate.of(2023, 1, 1));
        });
        assertEquals("La date de départ ne peut pas être avant la date d'arrivée.", exception.getMessage());
    }

    // TEST POUR LES DATES NULL :
    @Test
    void testDateArriveNull() throws SauvegardeImpossible{
        Ligue ligue = gestionPersonnel.addLigue("Football");

        Exception exception1 = assertThrows(Errurdate.class, () -> {
            ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , null , LocalDate.of(2023, 1, 1));
        });
        assertEquals("La date de départ ne peut pas être avant la date d'arrivée.", exception1.getMessage());
    }

    @Test
    void testDateDepartNull() throws SauvegardeImpossible , Errurdate{
        Ligue ligue = gestionPersonnel.addLigue("Football");

        Exception exception1 = assertThrows(Errurdate.class, () -> {
            ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , LocalDate.of(2023, 1, 1) , null);
        });
        assertEquals("La date de départ ne peut pas être avant la date d'arrivée.", exception1.getMessage());
    }

    @Test
    void setDateArriveNull()throws SauvegardeImpossible, Errurdate {
        Ligue ligue = gestionPersonnel.addLigue("Football");

        Employe employe = ligue.addEmploye("a", "a", "a", "a", LocalDate.of(2023, 12, 31), LocalDate.of(2024, 1, 1));
        Exception exception1 = assertThrows(Errurdate.class, () -> {
            employe.setDateArrive(null);
        });
    }

    void setDateDepartNull()throws SauvegardeImpossible, Errurdate {
        Ligue ligue = gestionPersonnel.addLigue("Football");

        Employe employe = ligue.addEmploye("a", "a", "a", "a", LocalDate.of(2023, 12, 31), LocalDate.of(2024, 1, 1));
        Exception exception1 = assertThrows(Errurdate.class, () -> {
            employe.setDateDepart(null);
        });
    }

    @Test
    void testSetDateDepartInvalid() throws SauvegardeImpossible , Errurdate {
        Ligue ligue = gestionPersonnel.addLigue("Fléchettes");

        Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
        Exception exception = assertThrows(Errurdate.class , () -> employe.setDateDepart(LocalDate.of(2022, 1, 1)));
        assertEquals("La date de départ ne peut pas être avant la date d'arrivée." , exception.getMessage());
    }

    @Test
    void testSetDateArriveInvalid() throws SauvegardeImpossible , Errurdate {
        Ligue ligue = gestionPersonnel.addLigue("Fléchettes");

        Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
        Exception erreur = assertThrows(Errurdate.class, () -> employe.setDateArrive(LocalDate.of(2024, 12, 2)));
        assertEquals("La date de départ ne peut pas être avant la date d'arrivée." , erreur.getMessage());
    }
```

Ajout des test unitaires testant le changement d'administrateurs :

```
@Test
void changementetSuppAdmin() throws SauvegardeImpossible, Erreurdate{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe test;
    test = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    //teste personnel
    assertEquals(gestionPersonnel.getRoot(), ligue.getAdministrateur());
    //MODIF ADMIN
    ligue.setAdministrateur(test);
    assertEquals(test, ligue.getAdministrateur());
    //SUPP ADMIN
    test.remove();
    assertFalse(ligue.getEmployes().contains(test));
    //VERIFIE QUE ROOT EST BIEN ADMIN
    assertFalse(ligue.getEmployes().contains(test));
    assertEquals(gestionPersonnel.getRoot(), ligue.getAdministrateur());
}
```

Ajout des tests unitaires testant la suppression ligues et employés :

```
@Test
void Suppression() throws SauvegardeImpossible, Erreurdate {
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe;
    employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    Employe employe1 = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    Employe employe2 = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    employe.remove();
    assertFalse(ligue.getEmployes().contains(employe));
    ligue.remove();
    assertFalse(gestionPersonnel.getLigues().contains(ligue));
}
```

Ajout des tests unitaires testant les getteurs et setteurs employés :

```
@Test
void Employe() throws SauvegardeImpossible, Erreurdate
{
//GETTEUR
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe test = ligue.addEmploye("El Arche", "Wassim", "mail", "azerty", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    assertEquals("Wassim", test.getPrenom());
    assertEquals("mail", test.getMail());
    assertEquals(ligue, test.getLigue());
// SETTEUR
    test.setMail("nouveauemail");
    assertEquals("nouveauemail", test.getMail());

    test.setNom("NvNom");
    assertEquals("NvNom", test.getNom());

    test.setPassword("nvmdp");
    assertTrue(test.checkPassword("nvmdp"));

    test.setPrenom("Nvprenom");
    assertEquals("Nvprenom", test.getPrenom());
}
```

Ajout des tests unitaires testant la création d'un employés :

```
@Test
void addEmploye() throws SauvegardeImpossible, ErreurDate {
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe;
    employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    assertEquals(employe, ligue.getEmployes().first());
}
```

b) Modifications ligne de commande

Ajout de la possibilité de changer d'administrateur :

```
private Menu suppOuEditEmploye(Employe employe) {
    Menu menu = new Menu("Editer Employe "+ employe.getNom() + " " + employe.getPrenom() + " de chez " + employe.getLigue().getNom());
    menu.add(modifierEmploye(employe));
    menu.add(supprimerEmploye(employe));
    menu.add(changerAdmin(employe));
    menu.addBack('q');
    return menu;
}

private void setAdmin(Employe employe ) {
    employe.getLigue().setAdministrateur(employe);
    System.out.println("Administrateur bien modifié");
}

private Option changerAdmin(Employe employe) {
    return new Option("Le nommer administrateur" , "n" , () -> setAdmin(employe));
}
```

Ajout de la possibilité de sélectionner un employé avant de le modifier ou supprimer :

```
private List<Employe> selectEmploye(Ligue ligue){
    return new List<Employe>("Selectionner un employer" , "s" , () -> new ArrayList(ligue.getEmployes()) , (nb) -> suppOuEditEmploye(nb))
}

private Menu suppOuEditEmploye(Employe employe) {
    Menu menu = new Menu("Editer Employe "+ employe.getNom() + " " + employe.getPrenom() + " de chez " + employe.getLigue().getNom());
    menu.add(modifierEmploye(employe));
    menu.add(supprimerEmploye(employe));
    menu.add(changerAdmin(employe));
    menu.addBack('q');
    return menu;
}
```

Ajout des dates et possibilité de la modifier :

```
Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrive, LocalDate dateDepart)
{
    this.gestionPersonnel = gestionPersonnel;
    this.nom = nom;
    this.prenom = prenom;
    this.password = password;
    this.mail = mail;
    this.ligue = ligue;
    if (dateArrive == null || dateDepart == null || dateDepart.isBefore(dateArrive) ) {
        throw new Erreurdate();
    }
    this.dateArrive = dateArrive;
    this.dateDepart = dateDepart;
}
```

```
{
    return (employe) -> editerEmploye(employe);
}

Option editerEmploye(Employe employe)
{
    Menu menu = new Menu("Gérer le compte " + employe.getNom(), "c");
    menu.add(afficher(employe));
    menu.add(changerNom(employe));
    menu.add(changerPrenom(employe));
    menu.add(changerMail(employe));
    menu.add(changerPassword(employe));
    menu.add(changerDateArriver(employe));
    menu.add(changerDateDepart(employe));
    menu.addBack("q");
    return menu;
}

private Option changerDateArriver(Employe employe) {
    return new Option("Changer date d'arriver", "a",
        () ->
    {
        try {
            employe.setDateArrive(LocalDate.parse(getString("Nouvelle date")));
        } catch (Erreurdate e) {
            // TODO Auto-generated catch block
            System.out.println("Les dates ne sont pas cohérentes : La date de départ ne peut pas être avant la date d'arriver ");
        } catch (DateTimeParseException s) {
            System.out.println("Veuillez fournir le bon format de date sous cette forme : AAAA-MM-JJ");
        }
    });
}

private Option changerDateDepart(Employe employe) {
    return new Option("Changer date Depart", "d",
        () ->
    {
        try {
            employe.setDateDepart(LocalDate.parse(getString("Nouvelle date")));
        } catch (Erreurdate e) {
            // TODO Auto-generated catch block
            System.out.println("Les dates ne sont pas cohérentes : La date de départ ne peut pas être avant la date d'arriver ");
        } catch (DateTimeParseException s) {
            System.out.println("Veuillez fournir le bon format de date sous cette forme : AAAA-MM-JJ");
        }
    });
}
```

Ajout des exceptions pour les date (dans le constructeur , setteur):

1) Création de l'exception :

```
package personnel;

public class Erreurdate extends Exception {

    public String getMessage(){
        return "La date de départ ne peut pas être avant la date d'arrivée.";
    }
}
```

2) Exception dans le constructeur :

```
Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrive, LocalDate dateDepart)
{
    this.gestionPersonnel = gestionPersonnel;
    this.nom = nom;
    this.prenom = prenom;
    this.password = password;
    this.mail = mail;
    this.ligue = ligue;
    if (dateArrive == null || dateDepart == null || dateDepart.isBefore(dateArrive) ) {
        throw new Erreurdate();
    }
    this.dateArrive = dateArrive;
    this.dateDepart = dateDepart;
}
```

3) Exceptions dans les setteurs :

```
public void setDateArrivee(LocalDate dateArrive)
    throws Erreurdate
{
    if (dateArrive == null || dateDepart.isBefore(dateArrive)) {
        throw new Erreurdate();
    }
    else {
        this.dateArrive = dateArrive;
    }
}

public void setDateDepart(LocalDate dateDepart)
    throws Erreurdate
{
    if (dateDepart == null || dateDepart.isBefore(dateArrive)) {
        throw new Erreurdate();
    }
    else {
        this.dateDepart = dateDepart;
    }
}
```

C) JDBC

Mettre à jour un employé dans la DB :

```
public void update(Employe employe) throws SauvegardeImpossible {
    try
    {
        String dateArrv = null;
        String dateDepart = null;

        if(employe.getLigue() != null) {
            dateArrv = employe.getDateArrivee().toString();
            dateDepart = employe.getDateDepart().toString();
        }

        PreparedStatement instruction;
        instruction = connection.prepareStatement("update employe set prenomEmploye = (?) , nomEmploye = (?) , mail = (?) , passwd = (?) , datearrv = (?) , datedepart = (?) where id_employe = (?)");
        instruction.setString(1, employe.getPrenom());
        instruction.setString(2, employe.getNom());
        instruction.setString(3, employe.getMail());
        instruction.setString(4, employe.getPassword());
        instruction.setString(5, dateArrv);
        instruction.setString(6, dateDepart );
        instruction.setBoolean(7, false);
        instruction.setInt(8, employe.getId());

        //System.out.println(employe.getLigue().getAdministrateur().getId() + " ET EMPLOYE : " + employe.getId());
        if(employe.estRoot() == false && employe.getLigue().getAdministrateur().getId() == employe.getId())
        {
            PreparedStatement instruction1;
            instruction1 = connection.prepareStatement("update employe set Admin = false where ID_Ligue = (?)", Statement.RETURN_GENERATED_KEYS);
            instruction1.setInt(1, employe.getLigue().getId());
            instruction1.executeUpdate();

            instruction.setBoolean(7, true);
        }
        instruction.executeUpdate();

    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

Supprimer dans la DB :

```
public void delete(Employe employe) throws SauvegardeImpossible {
    try
    {
        PreparedStatement instruction;
        instruction = connection.prepareStatement("delete from employe where id_employe = (?)", Statement.RETURN_GENERATED_KEYS);
        instruction.setInt(1, employe.getId());
        instruction.executeUpdate();
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}

public void delete(Ligue ligue) throws SauvegardeImpossible {
    try
    {
        PreparedStatement instruction1;
        instruction1 = connection.prepareStatement("delete from ligue where id_ligue = (?)", Statement.RETURN_GENERATED_KEYS);
        instruction1.setInt(1, ligue.getId());
        instruction1.executeUpdate();
    }
    catch (SQLException exception)
    {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

Mettre à jour une ligue dans la DB :

```
public void update(Ligue ligue) throws SauvegardeImpossible {
    try {
        PreparedStatement instruction;
        instruction = connection.prepareStatement("update ligue set nomLigue = (?) WHERE ID_Ligue = (?)", Statement.RETURN_GENERATED_KEYS);
        instruction.setString(1, ligue.getNom());
        instruction.setInt(2, ligue.getId());
        instruction.executeUpdate();

    } catch (SQLException exception) {
        exception.printStackTrace();
        throw new SauvegardeImpossible(exception);
    }
}
```

Insérer un employé dans la DB :

```
public int insert(Employe employe) throws SauvegardeImpossible{

    try {
        PreparedStatement instruction1;
        instruction1 = connection.prepareStatement("select * from employe where id_employe = 1");
        ResultSet t = instruction1.executeQuery();
        boolean verif = employe.getLigue() == null && !t.next();

        String sql = "insert into employe (prenomEmploye , nomEmploye , mail , passwd , dateArrive , datedepart , Admin , ID_Ligue ) values(?,?,?,?,?,?,?,?)";
        if(verif){
            sql = "insert into employe (ID_Employe, nomEmploye , passwd ) values(?, ?, ?)";
        }

        //System.out.println(sql);

        PreparedStatement instruction;
        instruction = connection.prepareStatement(sql, Statement.RETURN_GENERATED_KEYS);
        if(verif) {

            instruction.setInt(1,1);
            instruction.setString(2, employe.getNom());
            instruction.setString(3, employe.getPassword());
        }
        else if (employe.getLigue() != null){

            instruction.setString(1, employe.getPrenom());
            instruction.setString(2, employe.getNom());
            instruction.setString(3, employe.getMail());
            instruction.setString(4, employe.getPassword());
            instruction.setString(5, employe.getDateArrive().toString());
            instruction.setString(6, employe.getDateDepart().toString());
            instruction.setBoolean(7, employe.getAdmin());
            instruction.setInt(8, employe.getLigue().getId());
        }
        else return 0;
        instruction.executeUpdate();
        ResultSet id = instruction.getGeneratedKeys();
        id.next();
        return id.getInt(1);
    }
}
```

```
public GestionPersonnel getGestionPersonnel() throws SauvegardeImpossible
{
    GestionPersonnel gestionPersonnel = new GestionPersonnel();
    try
    {

        PreparedStatement instruction1;
        instruction1 = connection.prepareStatement("select id_employe , nomEmploye , passwd from employe where id_ligue is null");
        ResultSet resultat = instruction1.executeQuery();

        //for( int i = 0 ; i<=3 ; i++){System.out.println(resultat.getString(i));

        if(resultat != null && resultat.next()) {
            gestionPersonnel.addRoot(gestionPersonnel, resultat.getString(2), resultat.getString(3), resultat.getInt(1));
            //System.out.println("Debug");
        }

        String requete = "select * from ligue";
        Statement instruction = connection.createStatement();
        ResultSet ligues = instruction.executeQuery(requete);
        while (ligues.next()) {
            gestionPersonnel.addLigue(ligues.getInt(1), ligues.getString(2));}

        SortedSet<Ligue> liguees = gestionPersonnel.getLigues();

        String requete2 = "select * from employe";
        Statement instruction2 = connection.createStatement();
        ResultSet employes = instruction2.executeQuery(requete2);

        while(employes.next()) {
            for (Ligue ligue : liguees) {
                if(ligue.getId() == employes.getInt("ID_Ligue")) {
                    Employe e = ligue.addEmploye(employes.getString("nomEmploye"), employes.getString("prenomEmploye"), employes.getString("mail"), employes.getBoolean("Admin"));
                    if (employes.getBoolean("Admin"))
                        ligue.setAdministrateur(e);
                }
            }
        }
    }
}
```