

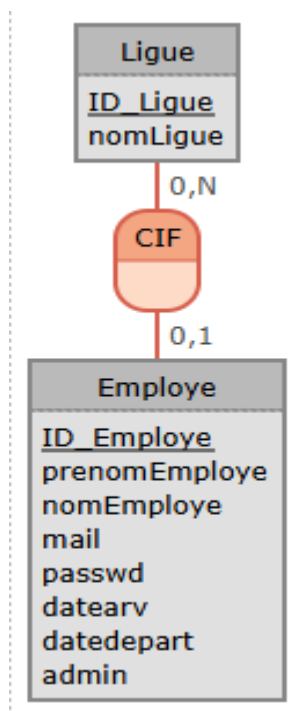
AP-Personnel

Ce projet a consisté à l'amélioration d'une application Java en y ajoutant des nouvelles fonctionnalités 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 employés (Modifications , Sélection)
- La création d'une base de données liant l'application à une base de données (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 :

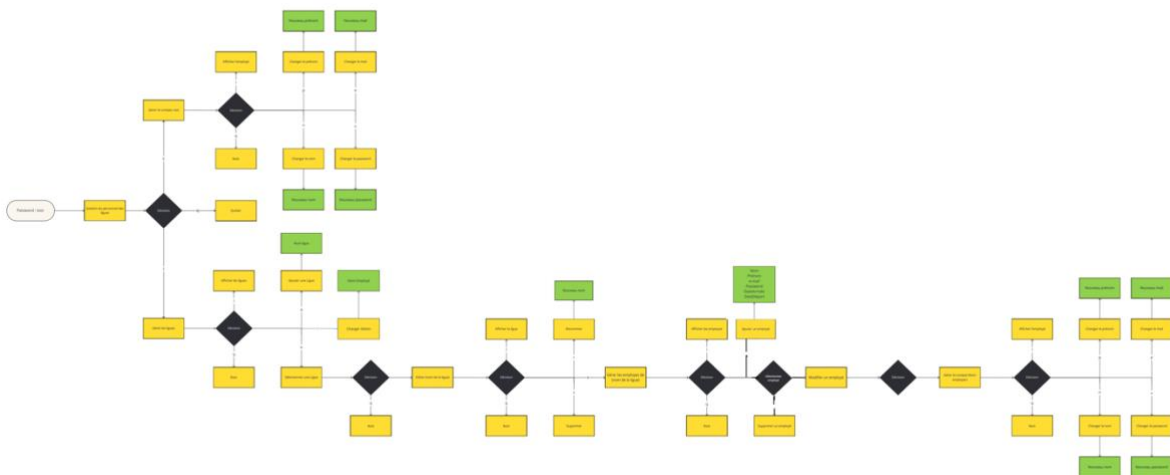
```

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 Employee (
8     ID_Employee INT,
9     prenomEmployee VARCHAR(50) NOT NULL,
10    nomEmployee VARCHAR(50) NOT NULL,
11    mail VARCHAR(100) UNIQUE NOT NULL,
12    passwd VARCHAR(50) NOT NULL,
13    datearv DATE NOT NULL,
14    datedepart DATE,
15    Admin BOOLEAN NOT NULL,
16    ID_Ligue INT,
17    ADD CONSTRAINT PK_Employee PRIMARY KEY (ID_Employee)
18 )engine = innodb;
19
20 ALTER TABLE Employee
21 ADD CONSTRAINT fk_Ligue_employee
22 FOREIGN KEY (ID_Ligue)
23 REFERENCES Ligue(ID_Ligue);

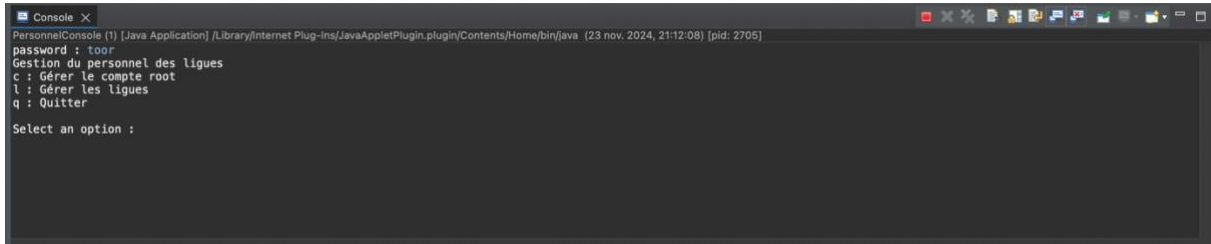
```

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 :



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(Erreurdate.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(Erreurdate.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 , Erreurdate{
    Ligue ligue = gestionPersonnel.addLigue("Football");

    Exception exception1 = assertThrows(Erreurdate.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, Erreurdate {
    Ligue ligue = gestionPersonnel.addLigue("Football");

    Employee employee = ligue.addEmploye("a", "a", "a", "a", LocalDate.of(2023, 12, 31), LocalDate.of(2024, 1, 1));
    Exception exception1 = assertThrows(Erreurdate.class, () -> {
        employee.setDateArrivee(null);
    });
}

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

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

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

    Employee employee = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    Exception exception = assertThrows(Erreurdate.class , () -> employee.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 , Erreurdate {
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");

    Employee employee = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty" , LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    Exception erreur = assertThrows(Erreurdate.class, () -> employee.setDateArrivee(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.getEmail());
    assertEquals(ligue, test.getLigue());

// SETTEUR
    test.setEmail("nouveauemail");
    assertEquals("nouveauemail", test.getEmail());

    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 addEmployee() throws SauvegardeImpossible, Erreurdate
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employee employee;
    employee = ligue.addEmployee("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2023, 1, 1), LocalDate.of(2023, 12, 31));
    assertEquals(employee, ligue.getEmployes().first());
}
```

b) Modifications ligne de commande

Ajout de la possibilité de changer d'administrateur :

```
private Menu suppOuEditEmployee(Employee employee) {
    Menu menu = new Menu("Editer Employee "+ employee.getNom() + " " + employee.getPrenom() + " de chez " + employee.getLigue().getNom());
    menu.add(modifierEmployee(employee));
    menu.add(supprimerEmployee(employee));
    menu.add(changerAdmin(employee));
    menu.addBack("q");
    return menu;
}

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

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

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

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

private Menu suppOuEditEmployee(Employee employee) {
    Menu menu = new Menu("Editer Employee "+ employee.getNom() + " " + employee.getPrenom() + " de chez " + employee.getLigue().getNom());
    menu.add(modifierEmployee(employee));
    menu.add(supprimerEmployee(employee));
    menu.add(changerAdmin(employee));
    menu.addBack("q");
    return menu;
}
```

Ajout des dates et possibilité de la modifier :

```
Employee(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrivee, LocalDate dateDepart)
throws Erreurdate
{
    this.gestionPersonnel = gestionPersonnel;
    this.nom = nom;
    this.prenom = prenom;
    this.password = password;
    this.mail = mail;
    this.ligue = ligue;

    if (dateArrivee == null || dateDepart == null || dateDepart.isBefore(dateArrivee) ) {
        throw new Erreurdate();
    }

    this.dateArrivee = dateArrivee;
    this.dateDepart = dateDepart;
}
```

```
{
    return (employee) -> editEmployee(employee);
}

Option editEmployee(Employee employee)
{
    Menu menu = new Menu("Gérer le compte " + employee.getNom(), "c");
    menu.add(afficher(employee));
    menu.add(changerNom(employee));
    menu.add(changerPrenom(employee));
    menu.add(changerMail(employee));
    menu.add(changerPassword(employee));
    menu.add(changerDateArrivee(employee));
    menu.add(changerDateDepart(employee));
    menu.addBack("q");

    return menu;
}

private Option changerDateArrivee(Employee employee) {
    return new Option("Changer date d'arriver", "a",
        () ->
        {
            try {
                employee.setDateArrivee(LocalDate.parse(getString("Nouvelle date")));
            } catch (Erreurdate e) {
                // TODO Auto-generated catch block
                System.out.println("Les dates ne sont pas coherente : La date de depart ne peut pas etre 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(Employee employee) {
    return new Option("Changer date Depart", "d",
        () ->
        {
            try {
                employee.setDateDepart(LocalDate.parse(getString("Nouvelle date")));
            } catch (Erreurdate e) {
                // TODO Auto-generated catch block
                System.out.println("Les dates ne sont pas coherente : La date de depart ne peut pas etre 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 :

```
Employee(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrivee, LocalDate dateDepart)
    throws Erreurdate
{
    this.gestionPersonnel = gestionPersonnel;
    this.nom = nom;
    this.prenom = prenom;
    this.password = password;
    this.mail = mail;
    this.ligue = ligue;

    if (dateArrivee == null || dateDepart == null || dateDepart.isBefore(dateArrivee) ) {
        throw new Erreurdate();
    }

    this.dateArrivee = dateArrivee;
    this.dateDepart = dateDepart;
}
```

3) Exceptions dans les setteurs :

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

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