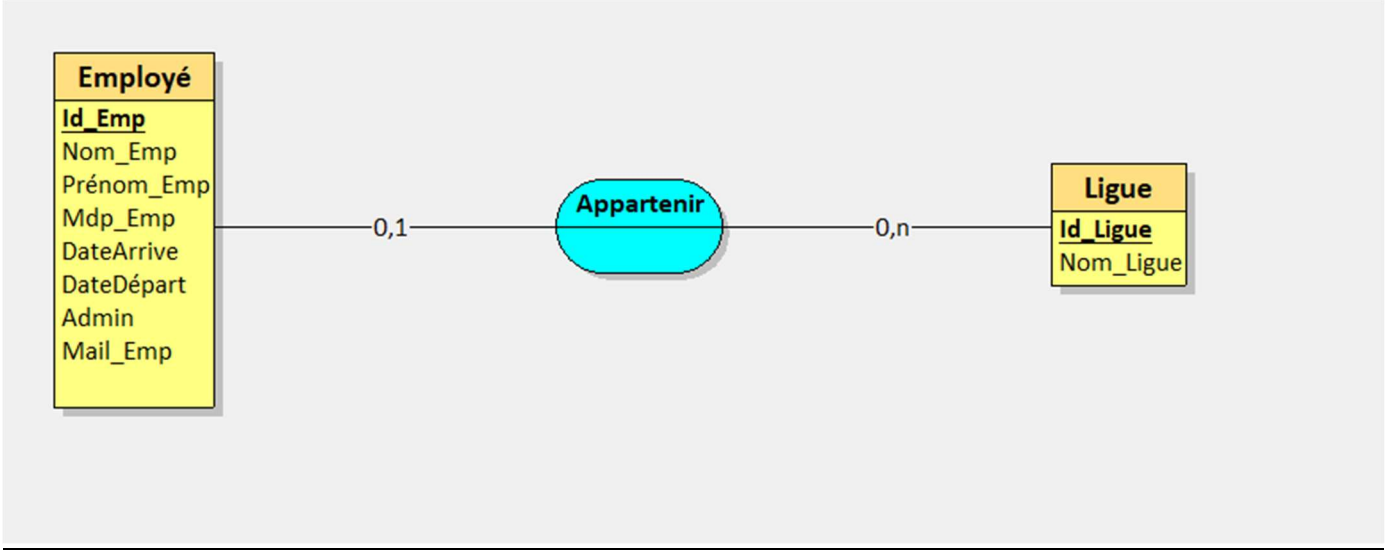
AP-Personnel

Ce projet a consisté en l'amélioration d'une application Java en y ajoutant de nouvelles fonctionnalités, telles que :

- La gestion de l'administrateur en ligne de commande.
- La gestion des dates avec leurs exceptions.
- L'ajout de tests 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 :

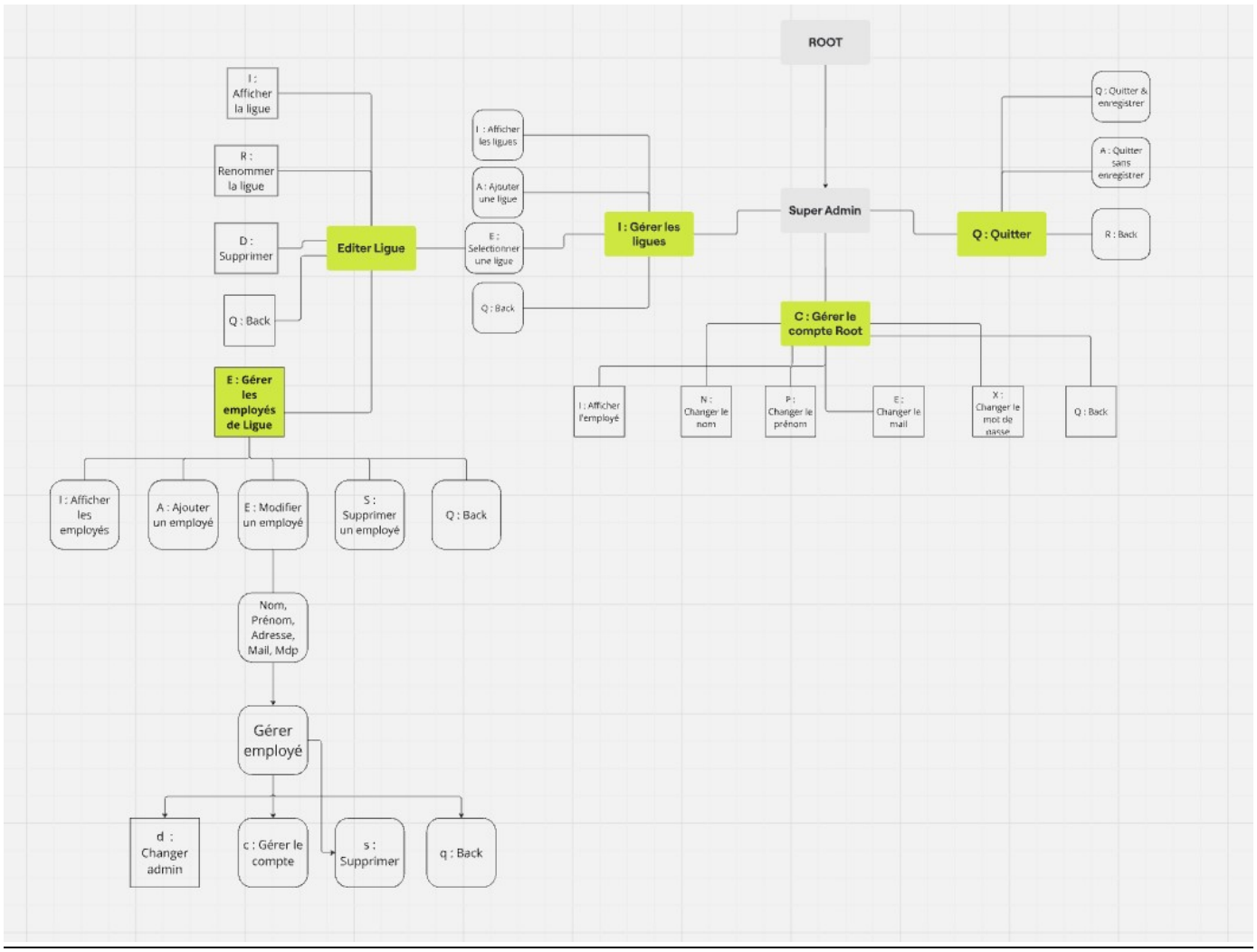


b. Le script de création de ce MCD:

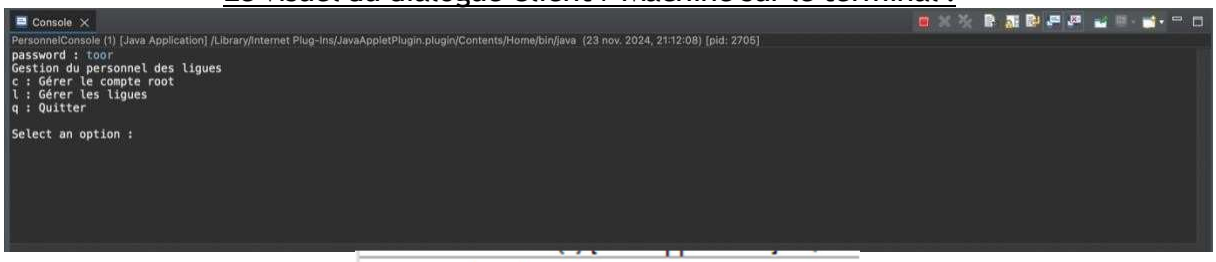
```sql
CREATE TABLE Ligue (
    ID_Ligue INT,
    nomLigue VARCHAR(100) NOT NULL,
    ADD CONSTRAINT PK_Ligue PRIMARY KEY (ID_Ligue)
)engine = innodb;

CREATE TABLE Employe (
    ID_Employe INT,
    prenomEmploye VARCHAR(50) NOT NULL,
    nomEmploye VARCHAR(50) NOT NULL,
    mail VARCHAR(100) UNIQUE NOT NULL,
    passwd VARCHAR(50) NOT NULL,
    datearv DATE NOT NULL,
    datedepart DATE,
    Admin BOOLEAN NOT NULL,
    ID_Ligue INT,
    ADD CONSTRAINT PK_Employe PRIMARY KEY (ID_Employe)
)engine = innodb;

ALTER TABLE Employe
ADD CONSTRAINT fk_Ligue_employe
FOREIGN KEY (ID_Ligue)
REFERENCES Ligue(ID_Ligue);
```

2) L'applications Java :

Arbre Heuristique de l'applications :
https://github.com/Kadeyofficiel/Personnelbis/blob/main/ARBRE_HEURISTIQUE%20V2.png


ROOT
Super Admin
I : Gérer les ligues
I : Afficher les ligues
A : Ajouter une ligue
E : Selectionner une ligue
Q : Back
Editer Ligue
I : Afficher la ligue
R : Renommer la ligue
D : Supprimer
Q : Back
E : Gérer les employés de Ligue
I : Afficher les employés
A : Ajouter un employé
E : Modifier un employé
S : Supprimer un employé
Q : Back
Nom, Prénom, Adresse, Mail, Mdp
Gérer employé
d : Changer admin
c : Gérer le compte
s : Supprimer
q : Back
Q : Quitter
Q : Quitter & enregistrer
A : Quitter sans enregistrer
R : Back
C : Gérer le compte Root
I : Afficher l'employé
N : Changer le nom
P : Changer le prénom
E : Changer le mail
X : Changer le mot de passe
Q : Back

Le visuel du dialogue Client / Machine sur le terminal :




```
password : toor
```

```
Gérer le compte root
l : Afficher l'employé
n : Changer le nom
p : Changer le prénom
e : Changer le mail
x : Changer le password
q : Back
```

```
Gérer les ligues
l : Afficher les ligues
a : Ajouter une ligue
e : Sélectionner une ligue
q : Back
```

a) Tests unitaires + getters setters

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

```java
@Test
void createLigue() throws SauvegardeImpossible
{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    assertEquals("Fléchettes", ligue.getNom());
}

@Test
void addEmploye() throws SauvegardeImpossible
{
    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,
    assertEquals(employe, ligue.getEmployes().first());
}

//Test pour la date de départ antérieure à la date d'arrivée
@Test
void testDateDepartAnterieureDateArrivee()throws IllegalArgumentException, SauvegardeImpossible {
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2025, 01, 24), null);
    IllegalArgumentException exception = assertThrows(IllegalArgumentException.class, () -> employe.setDateDepart(LocalDate.of(2024, 01, 24)
    assertEquals("La date de départ doit être postérieure ou égale à la date d'arrivée.", exception.getMessage());
}

//Test pour la date de départ postérieure à la date d'arrivée
@Test
void testDateDepartPosterieureDateArrivee()throws IllegalArgumentException, SauvegardeImpossible{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2025, 01, 24), null);
    assertDoesNotThrow(() -> employe.setDateDepart(LocalDate.of(2026, 10, 27)));
    assertEquals(LocalDate.parse("2026-10-27"), employe.getDateDepart());
}
```

```java
//Test pour le setNom
@Test
void testSetNom() throws SauvegardeImpossible {
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2025, 01, 24), null);
    employe.setNom("Bouchard");
    assertEquals("Bouchard", employe.getNom());
}

//Test pour le setPrenom
@Test
void testSetPrenom() throws SauvegardeImpossible{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2025, 01, 24), null);
    employe.setPrenom("Gérard");
    assertEquals("Gérard", employe.getPrenom());
}

//Test pour le setMail
@Test
void testSetEmail() throws SauvegardeImpossible{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2025, 01, 24), null);
    employe.setMail("g.bouchard@gmail.com");
    assertEquals("g.bouchard@gmail.com", employe.getMail());
}

//Test pour le setPassword
@Test
void testSetPassword() throws SauvegardeImpossible{
    Ligue ligue = gestionPersonnel.addLigue("Fléchettes");
    Employe employe = ligue.addEmploye("Bouchard", "Gérard", "g.bouchard@gmail.com", "azerty", LocalDate.of(2025, 01, 24), null);
    employe.setPassword("azerty");
    assertEquals("azerty", employe.getPassword());
}
```

```java
// Getters de dateArrive
public LocalDate getDateArrive()
{
    return dateArrive;
}
// Getters de dateDepart
public LocalDate getDateDepart()
{
    return dateDepart;
}

// Setters de dateArrive
public void setDateArrive(LocalDate dateArrive)
{
    this.dateArrive = dateArrive;
}

// Setters de dateDepart + Exception
public void setDateDepart(LocalDate dateDepart) {
    // Vérifie que DateDépart est non null + avant date arrivé et provoque l'exception
    if (dateDepart != null && dateDepart.isBefore(this.dateArrive)) {
        throw new IllegalArgumentException("La date de départ doit être avant ou égale à la date d'arrivé.");
    }
    this.dateDepart = dateDepart; // Met à jour dateDépart
}
```

```java
// Importation du Package LocalDate
import java.time.LocalDate;
```

```java
// Ajout des variables d'instance private de dateArrive et dateDepart
private LocalDate dateArrive;
private LocalDate dateDepart;
```

```java
// Ajout des variables d'instance de type LocalDate
Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrive,
{
    this.gestionPersonnel = gestionPersonnel;
    this.nom = nom;
    this.prenom = prenom;
    this.password = password;
    this.mail = mail;
    this.ligue = ligue;
    // Initialisation des variables d'instances
    this.dateArrive = dateArrive;
    this.dateDepart = dateDepart;
```

```java
LocalDate dateDepart)
```

Ajout des test unitaires testant le changement d'administrateurs :

```java
    @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:

```java
    @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 getters et setters employés:

```java
    @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("nouveaumail");
        assertEquals("nouveaumail" , 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)
        throws Erreurdate
{
    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.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(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 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):

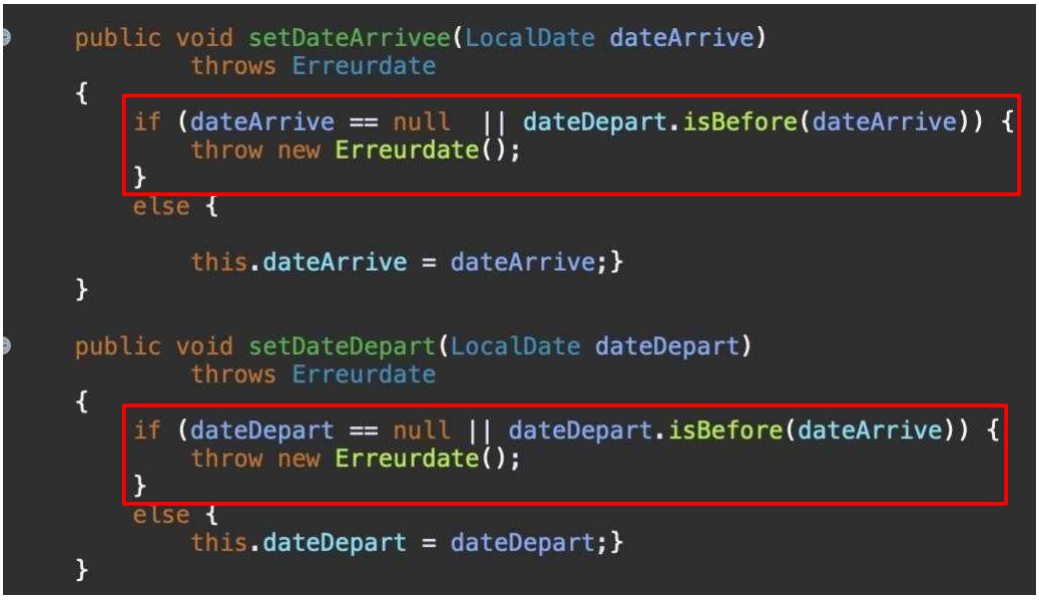
1) Création de l'exception :

```java
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 :

```java
Employe(GestionPersonnel gestionPersonnel, Ligue ligue, String nom, String prenom, String mail, String password, LocalDate dateArrive, LocalDate dateDepart)
        throws Erreurdate
{
    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 :



```java
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;}
}
```