Table des matières

Table des figures

List of Listings

Résumé

Ce projet et la résulta d'un stage que j'avais passe chez la Faculté des Lettres et Science Humaines, El Jadida sous le theme Informatision du service Resourse Humains. Sous l'encadrement de Mr. A. Madani, et la supervision du chef de service ; Mr. Driss Dibaji.

Première partie

Intoduction

Pour obtenir ce besoin, j'avais la responsabilité de développer un environnement pour gérer les différents taches décrites en cahier des charges

Pour realiser cela, il y a deux parties. Stockage des données et l'application bureau. Pour la première, j'ai choisi *XML* ; un langage markup écrit dans un fichier texte. Tout simplement parce qu'il est simple à utiliser et/ou modifier ainsi qu'il gratuit. Et pour la deuxième, j'ai développé une application en *Java*, car il est un langage Oriente-Objet qui facilite le processus de développement.

Deuxième partie

Cahier des Charges

D'après son nom, *Service des ressources humaines* est un service qui est responsable de la gestion des employées et fonctionnaires, leurs diplômes et grades, ainsi que donner des attestations du travail et des autorisations de congé, suivi d'absence, rémunération du travail les jours fériés et finalement une notation annuelle.

Donc, on en déduit que le *cahier des charges* est le suivant :

— Implémenter un système de gestion des employées/fonctionnaires
— Gérer les diplômes et les grades
— Suivi des grades
— Suivi d'absence
— Suivi de rémunération du travail les fériés
— Générer des attestations de travail
— Générer des autorisations de congé
— Générer des notations annuelle pour

Troisième partie

Conception

Chapitre 1

Stockage des données en XML

Les données sont stockées dans un fichier `XML`, `data/xml/hr.xml` puisqu'il est lisible à la fois par la machine et l'humain. Au suivant, les règles du gestion est schéma général du fichier.

1.1 Règles de gestion du fichier `XML`

Le root-tag est `<Employee>` et qui contient plusieurs tags de type `<employee>` qui représente des employées. Chaque tag `<employee>` contient un seul tag `<personal>` et un seul tag `<administrative>` qui peut contient 0 ou plusieurs tags `<uplift>`. Le tag `<employee>` peut aussi avoir 0 ou plusieurs tags de type `<diploma>`, `<medicalcertif>` et `<repayment>`.

Voici la signification de chaque tag des tags déclaré ci-dessus :

`<Employee>` le root-tag, qui contient les tags `employee`

`<employee>` contient tout les information d'un employée particulier est il avait deux attributes :

reference identifiant du employée

departement département du employée. Certains employées n'appartient à aucun département. Ce sont des *fonctionnaire*

`<personal>` contient des informations personnelle comme le *nom*, *prénom*, *date de naissance*, etc.

`<administrative>` contient des informations administrative comme le *SOM*, *CIN*, etc.

`<uplift>` contient les informations des avancements dans le grade, *date*, *indice*, *échelon* et *échelle*. Ce tag avais un seul attribut.

id identifiant du avancement par rapport au avancement précédant

`<diploma>` contient les informations sur les diplômes, *titre*, *mention*, *institue* et *session*. Ce tag a aussi un seul attribue, ainsi que l'enfant `<title>`.

id identifiant du diplôme

mention la mention du diplôme (dans le tag `<title>`)

`<medicalcertif>` contient les informations sur certification médical, *date du certification*, *durée* et *la période*.

id identifiant du certification médical.

`<repayment>` contient les informations sur les remboursements, *la période*, *nombre des jours à rembourser* et *nombre des jours déjà remboursé*

id identifiant du remboursement

1.2 Schéma général du fichier

```
<Employee>                        <!--root-->
  <employee reference="" department="">

    <notes />                      <!--les notes sur l'employé-->

    <personal>                     <!--les informations personnelles-->
    </personal>

    <administrative>               <!--les informations administrative-->
      <uplift id="" state="">      <!--les informations d'avancement-->
      </uplift>

      <uplift id="" state="" />    <!--nous pouvons avoir plus-->
    </administrative>

    <diplomas id="">               <!--les infomration du diplôme-->
    </diplomas>
    <diplomas id="" />             <!--nous pouvons avoir plus-->

    <medicalcertif id="">          <!--information du certificat médical-->
    </medicalcertif>
    <medicalcertif id="" />        <!--nous pouvons avoir plus-->

    <repayment id="">              <!--information du remboursement-->
    </repayment>
    <repayment id="" />            <!--nous pouvons avoir plus-->

  </employee>
</Employee>
```

Listing 1: Schéma général `XML` du fichier `data/xml/hr.xml`

Chapitre 2

Les Paquets system et leurs Classes

Le code source de l'application est divisé en 4 paquets pricipales :

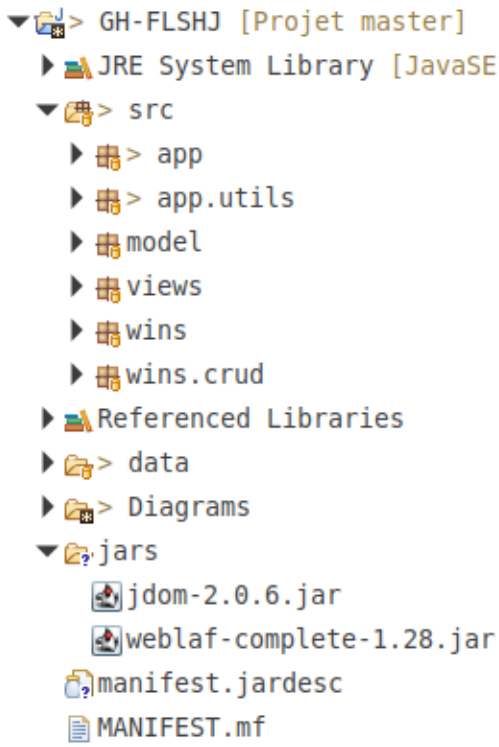
`model` contient les différentes classe pour mobilisé les donnée en objet

`app` contient les différentes énumération utilisées dans l'application. Ce paquet contient aussi `app.utils`, qui contient des utilitaires utiles pour le développement, notamment la gestion du fichier XML.

`wins` contient des interfaces graphiques, y compris celles qui sont responsables des opérations CRUD normales qui existent dans `wins.crud`

`views` contient des pages générées pour l'impression.

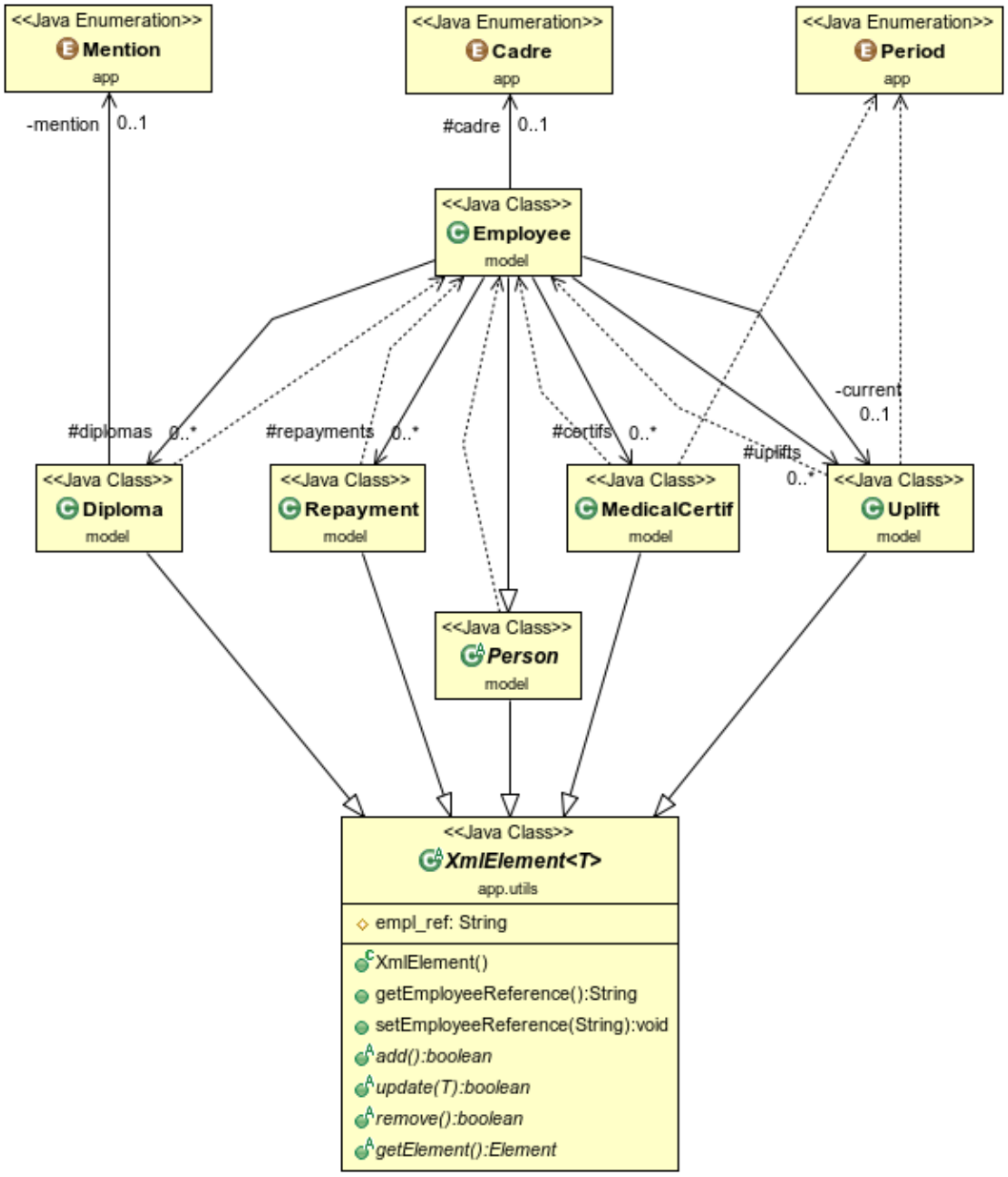
FIGURE 2.1 – Aperçu de code source du projet



2.1 Paquet model

Ce paquet contient les modèles de l'application, ce sont des classes Java pour modéliser les informations stockées dans le fichier XML data/xml/hr.xml.

FIGURE 2.2 – La relation entre les classes du paquet model et la classe abstrait XmlElement du paquet app.utils



Les classes implémentent les méthodes abstrais getElement(), add(), update() et remove() dans la classe générique XmlElement. Ces méthodes sont responsables de la selection, l'ajout, la mise à jour et la suppression du tag correspondant à l'objet concerné dans le fichier xml.

Voici la classe mère de toutes les classes, XmlElement, qui contient en addition,

une chaine de caractères qui représente le référence de l'employé, c.-à-d. L'identifient

```java
import org.jdom2.Element;

public abstract class XmlElement<T> {
    public abstract boolean add();
    public abstract boolean update(T updated);
    public abstract boolean remove();
    public abstract Element getElement();

    /* référence du employée */
    protected String empl_ref;
    public String getEmployeeReference( ) {
        return empl_ref;
    }

    public void setEmployeeReference(String ref) {
        this.empl_ref = ref;
    }
}
```

Listing 2: Extrais du classe générique `XmlElement` du paquet `app.utils`

Les méthodes `add()`, `update()` et `remove()` de `XmlElement` retournent une valeur booléen, qui signifie est ce que l'opération a réussie ou non. Tandis que `getElement()` retourne le tag `XML` correspondant a l'objet.

La raison pour laquelle la classe est générique, c'est que `update()` doit l'être. La méthode `update()` prend un variable de type `T`, ce type est décrit avec un héritage du classe `XmlElement`.

Par exemple, `update()` dans la classe `Diploma` est la suivant :

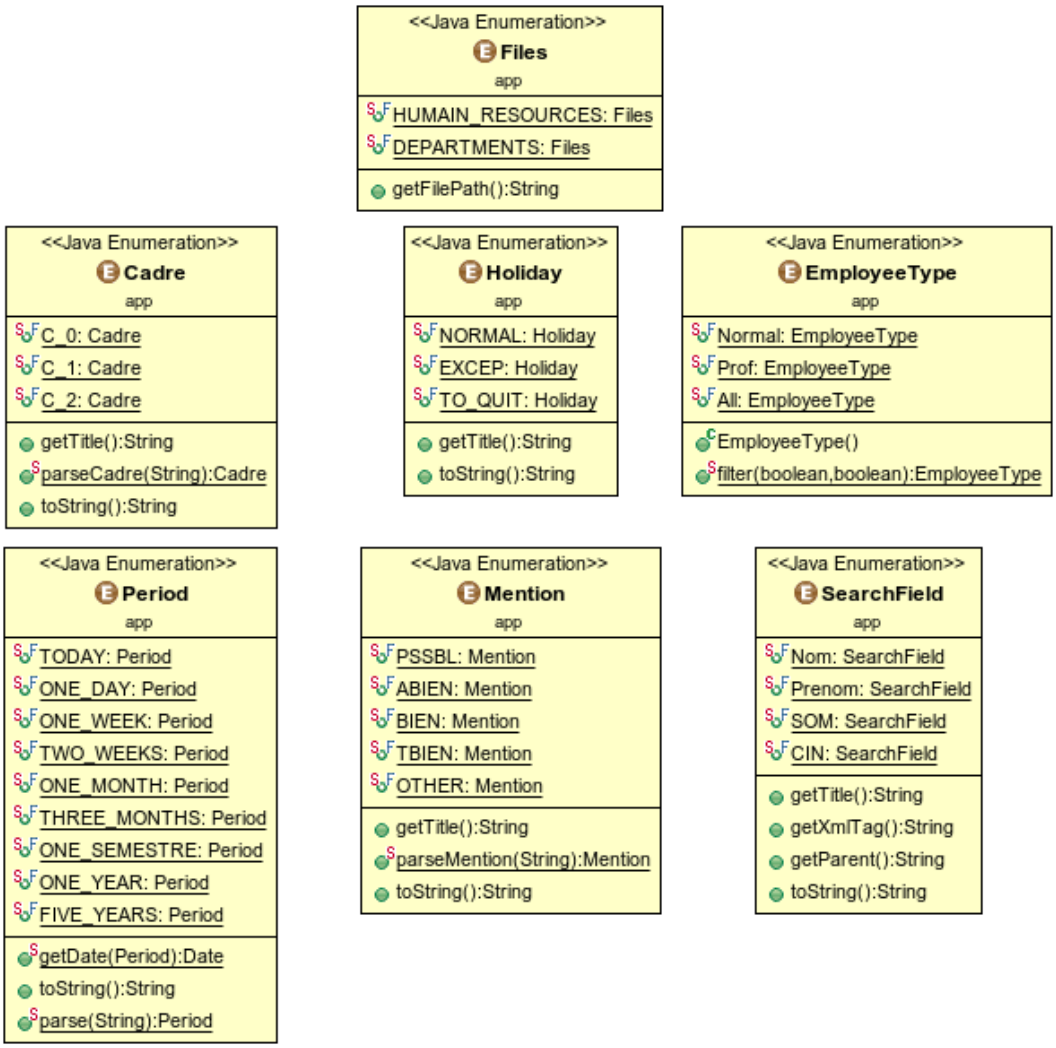
```java
public class Diploma exstends XmlElement<Diploma> {
    /* les attributs du classe */

    @Override
    public boolean update(Diploma updated) {
        /* process la mise à jour */
    }
}
```

Listing 3: Extrais du classe `Diploma` qui montre l'héritage de `XmlElement`

2.2 Paquet app

Le paquet `app` contient que les énumérations, décrites dans le figure 2.3.

FIGURE 2.3 – Les enumeration dans paquet `app`



`Cadre` représente les cadres possibles pour un employée/fonctionnaire

`Mention` les mentions possibles pour un diplôme, utilisés dans `DilpomaCrud.java`

`Period` les différentes périodes utilisées dans l'application

`Holiday` utilisé dans `MainWin.java` pour la génération du congé

`SearchField` utilisé pour la recherche dans `MainWin.java`

`EmployeeType` utilisé pour filtrer les employée et fonctionnaire dans `MainWin.java`

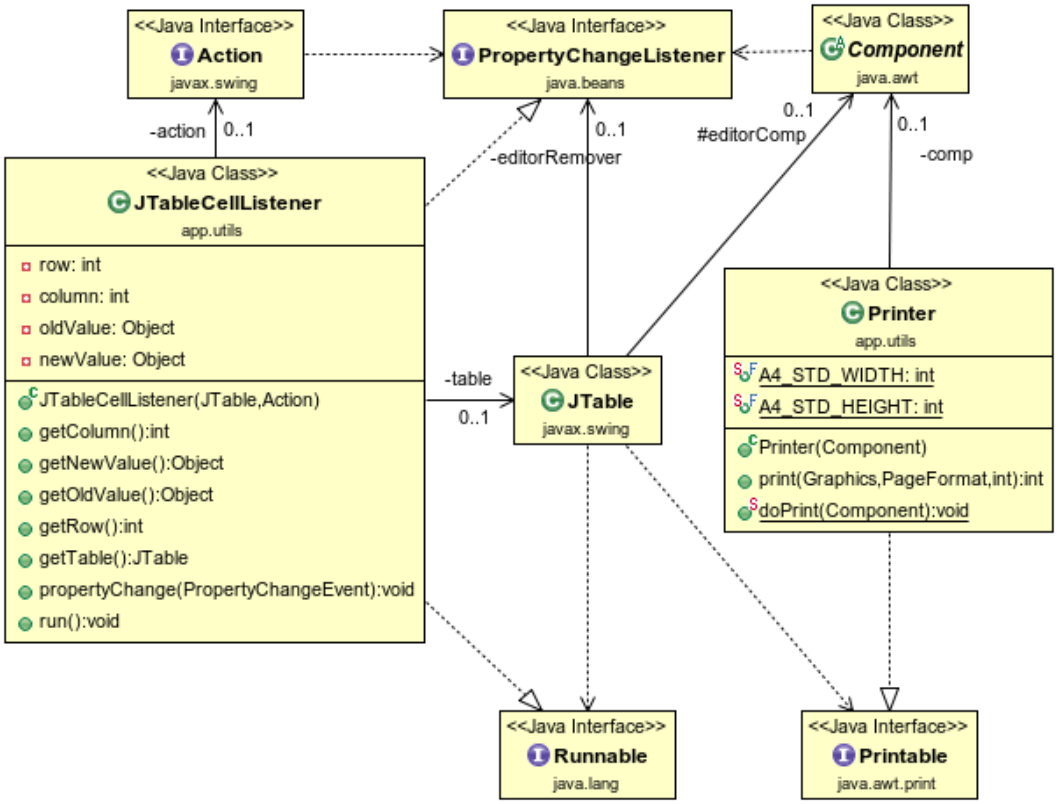
`Files` contient des énumérations qui concernent les différents fichiers `XML` utilisé.

2.3 Paquet app.utils

Alors, ce paquet contient des classes important pour l'application. Le diagramme des classes dans la figure 2.4 explique les différents relations entre ces classes et les classes de système de la d'interface graphique du Java.

La classe Printer est responsable de l'impression d'un Component, la classe des composants graphiques, avec l'aide de la méthode static doPrint() qui prend un Component comme paramètre.

FIGURE 2.4 – La relation entre JTable, JTableListener, et Printer du paquet app.utils



Aussi, Printer a une implémentation de la méthode abstrait print() de l'interface Printable, doPrint() faire un appel à cette méthode avec l'aide des autres classes du paquet java.awt.print comme PrinterJob et PageFormat.

> *Note :* Principalement, L'appelle de la méthode print() se fait par la classe PrinterJob() du paquet java.awt.print.PrinterJob. La méthode utilise dans l'application c'est static doPrint().

Au suivant, un extrait du classe Printer qui montre l'implémentation de la méthode print().

```java
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.print.PageFormat;
import java.awt.print.Printable;

public class Printer implements Printable {
    /**
     * méthode abstrait dans l'interface java.awt.Printable, l'appelle
     * se fait par la classe java.awt.PrinterJob
     *
     * @param g une graphique du classe java.awt.Graphics
     * @param format formatage de la page à imprimer
     * @param page_index l'index de la page
     *
     * @return intègre qui représente l'état de l'impression */
    @Override
    public int print(Graphics g, PageFormat format, int page_index) {
        /* vérification du page, c'est un protocole de l'interface */
        if (page_index > 0) return Printable.NO_SUCH_PAGE;

        /* prend les dimensions du composant */
        Dimension dim = comp.getSize( );
        double cHeight = dim.getHeight( ), cWidth = dim.getWidth( );

        /* initialization de demention du la zone d'impression */
        double pHeight = format.getImageableHeight( );
        double pWidth = format.getImageableWidth( );
        double pXStart = format.getImageableX( );
        double pYStart = format.getImageableY( );

        /* la difference entre la taille du composant et la taille du
         * page pour mettre le composant à l'échelle du page */
        double xRatio = (pWidth / cWidth), yRatio = (pHeight / cHeight);

        /* mettre le composant à jour avec la page à imprimer en utilisent
         * la methode java.awt.Component.paint() */
        Graphics2D g2 = (Graphics2D) g;
        g2.translate(pXStart, pYStart);
        g2.scale(xRatio, yRatio);
        comp.paint(g2);
        return Printable.PAGE_EXISTS;
    }
}
```

Listing 4: Extrais de la classe `Printer` qui montre l'implémentation de la méthode abstrait `print()` de l'interface `java.awt.Printable`

On revient à la figure 2.4, la classe JTableCellListener est responsable à réagir avec une modification qui passe au niveau des cellules d'un JTable. Cette classe est à l'écoute des modifications apportées aux données de la table via TableCellEditor du paquet javax.swing.table avec l'aide du interface PropertyChangeListener du paquet java.beans. Donc, doit implémenter la méthode propertyChange() de l'interface mentionnée.

```
1  import java.awt.event.ActionEvent;
2
3  import java.beans.PropertyChangeEvent;
4  import java.beans.PropertyChangeListener;
5
6  import javax.swing.Action;
7  import javax.swing.JTable;
8  import javax.swing.SwingUtilities;
9
10 public class JTableCellListener implements PropertyChangeListener, Runnable {
11     private JTable table;
12     private Action action;
13     private int row;
14     private int column;
15     private Object oldValue;
16     private Object newValue;
17
18     /**
19      * @param table la table concerne
20      * @param action l'action a invoquée
21      */
22     public JTableCellListener(JTable table, Action action) {
23         this.table = table;
24         this.action = action;
25
26         /* ajouter cette classe à la table pour l'invoquée */
27         this.table.addPropertyChangeListener(this);
28     }
29 }
```

Listing 5: Extrait du classe JTableListener

Lorsque l'édition est démarrée, la valeur de la cellule est enregistrée. Lorsque l'édition est arrêtée, la nouvelle valeur est enregistrée en tant que Object. Lorsque l'ancienne et la nouvelle valeur sont différentes, l'action fournie est invoquée. La classe doit appeller des classes selon l'état de l'édition, alors on doit aussi implémenter la méthode run() dans l'interface Runnable. Au suivant un extrait du classe JTableCellListener. Par la suit, des extraits du classe et ses méthodes.

La classe possède un constricteur privé qui prend en paramètre la table, numéro du ligne et colonne, et les deux valeurs, l'ancien et nouvelle. Ce constricteur est utilisé dans la méthode `processEditingStopped()` pour créer une sauvegarde de la cellule concernée.

```
30 /**
31  * Créé une copie du JTableCellListener avec une sauvegarde des
32  * anciennes/nouvelles données ainsi que la ligne et la colonne
33  *
34  * @param row la ligne de la cellule modifiée
35  * @param column la colonne de la cellule modifiée
36  * @param oldValue l'ancienne valeur de la cellule modifiée
37  * @param newValue nouvelle valeur de la cellule modifiée
38  */
39 private JTableCellListener(JTable table, int row, int column,
40                            Object oldValue, Object newValue) {
41     this.table = table;
42     this.row = row;
43     this.column = column;
44     this.oldValue = oldValue;
45     this.newValue = newValue;
46 }
47
48 /**
49  * Implémentation de l'interface PropertyChangeListener
50  *
51  * @param e l'événement génère par le système
52  */
53 @Override
54 public void propertyChange(PropertyChangeEvent e) {
55     /* tester si l'évent vient d'après la classe TableCellEditor */
56     if ("tableCellEditor".equals(e.getPropertyName( ))) {
57         /* si la table est en coure de la modification */
58         if (table.isEditing( )) processEditingStarted( );
59         /* si l'edition est terminée */
60         else processEditingStopped( );
61     }
62 }
```

Listing 6: L'implémentation du `propertyChange()` de l'interface `Printable` dans la classe `JTableListener`

La méthode `JTable.isEditing()` indique l'état booléen du table, la valeur `true` indique que la table est en une modification actif, alors on fait un appel à `processEditingStarted()`. Lorsque la méthode retourne la valeur `false`, on fait un appel au `processEditingStopped()`, qui est responsable de la vérification est ce que la valeur de la cellule a été modifie ou non par la comparaison entre `oldValue`

et newValue dans la classe JTableCellListener. Si les valeurs sont différentes, on fait un appel à l'action du classe.

```
63  /**
64   * annoncer le démarrage du processus d'édition de cellule
65   */
66  private void processEditingStarted( ) {
67      SwingUtilities.invokeLater(this);
68  }
69
70  /**
71   * vérifier la cellule concernée
72   */
73  private void processEditingStopped( ) {
74      /* sauvegardée la nouvelle valeur */
75      newValue = table.getModel( ).getValueAt(row, column);
76
77      /* si la nouvelle valeur est différente a l'ancienne valeur, alors */
78      if (!newValue.equals(oldValue)) {
79          JTableCellListener tcl;
80          ActionEvent event;
81
82          tcl = new JTableCellListener(getTable( ), getRow( ), getColumn( ),
83                                       getOldValue( ), getNewValue( ));
84          event = new ActionEvent(tcl, ActionEvent.ACTION_PERFORMED, "");
85
86          action.actionPerformed(event); /* exécuter l'action */
87      }
88  }
```

Listing 7: Les fonctions `processEditingStarted()` et `processEditingStopped()` utilisées dans la méthode `propertyChange()` de la classe `JTableListener`

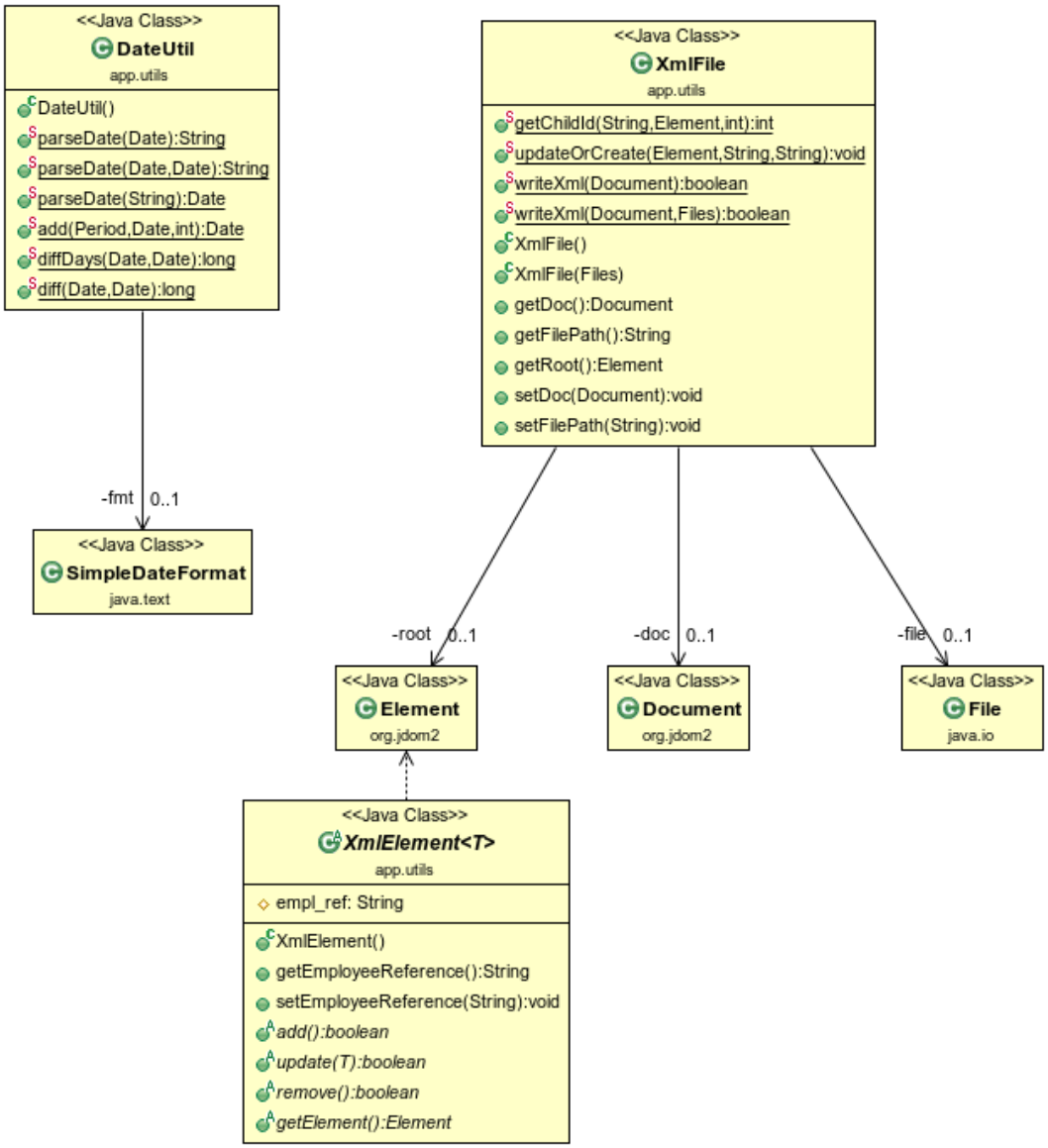
Et finalement, il reste que l'implémentation de la méthode `run()`, qui est été appelée avec le protocole `SwingUtilities.invokeLater()` dans `processEditingStarted()`. Le rôle de cet appel est de récupérer la valeur actuelle de la cellule.

Note : Le but du création de cette classe `JTableCellListener` *c'est que dans* `Java` *il n'y a pas d'un listener native, ou par défaut qui suit l'état des cellules d'un* `JTable`.

```
89  @Override
90  public void run( ) {
91      row = table.convertRowIndexToModel(table.getEditingRow( ));
92      column = table.convertColumnIndexToModel(table.getEditingColumn( ));
93      oldValue = table.getModel( ).getValueAt(row, column);
94      newValue = null;
95  }
```

Listing 8: L'implémentation du `run()` du classe `JTableListener`

FIGURE 2.5 – La relation entre `DateUtils`, `XmlFile` et `XmlElement`



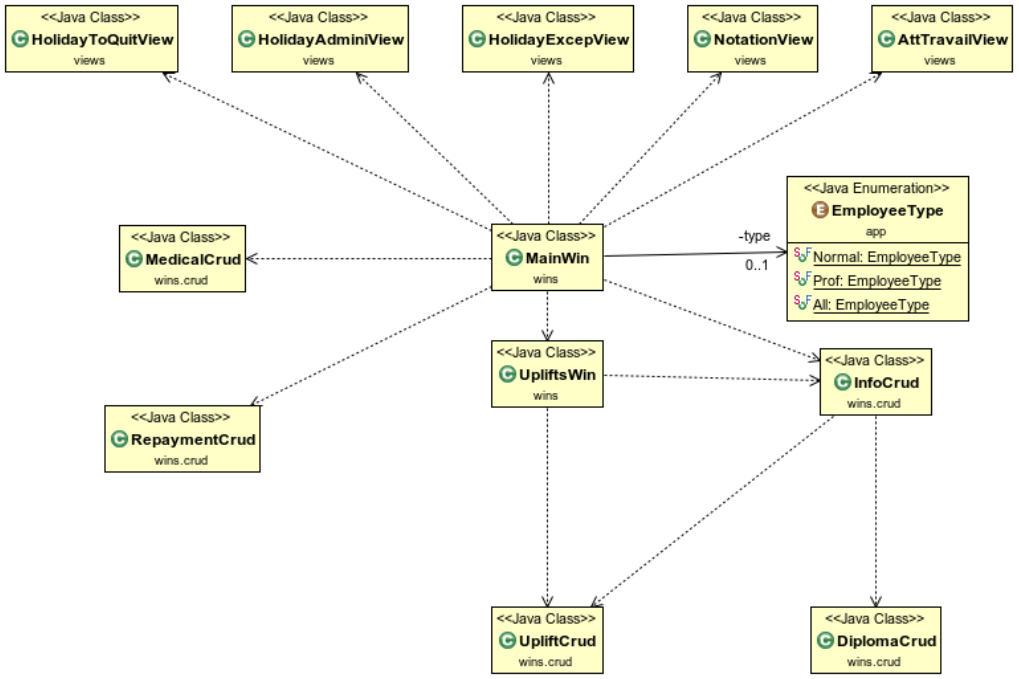
Quatrième partie

L'Interface Graphique

Chapitre 3

La relation entre les différent Paquet graphique

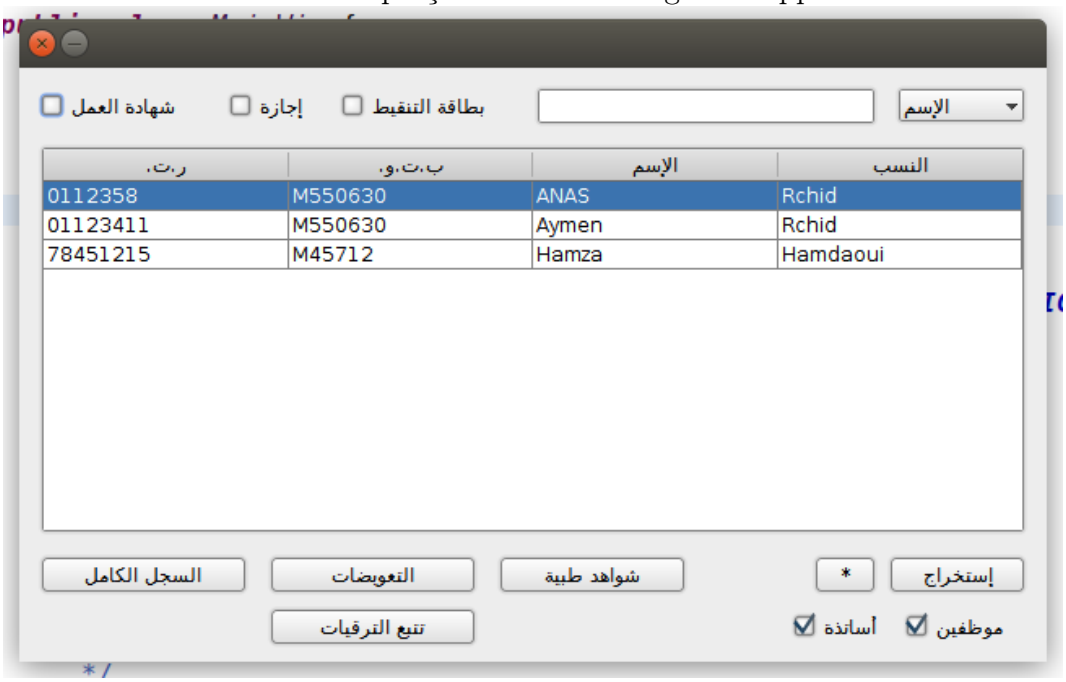
FIGURE 3.1 – Aperçu général sur les interfaces graphiques utilisé dans l'application



3.1 Paquet wins

1. La fenêtre principale MainWin

FIGURE 3.2 – Aperçu sur le démarrage de l'application



2. La fenêtre de suivi des avancements de grade

3.2 Paquet wins.crud

1. Gestion des Employées InfoCrud
2. Gestion de Diplômes
3. Gestion des Certifications Médical
4. Gestion des Grades

3.3 Paquet views

1. Page de conge

FIGURE 3.3 – Aperçu sur la génération d'un congé

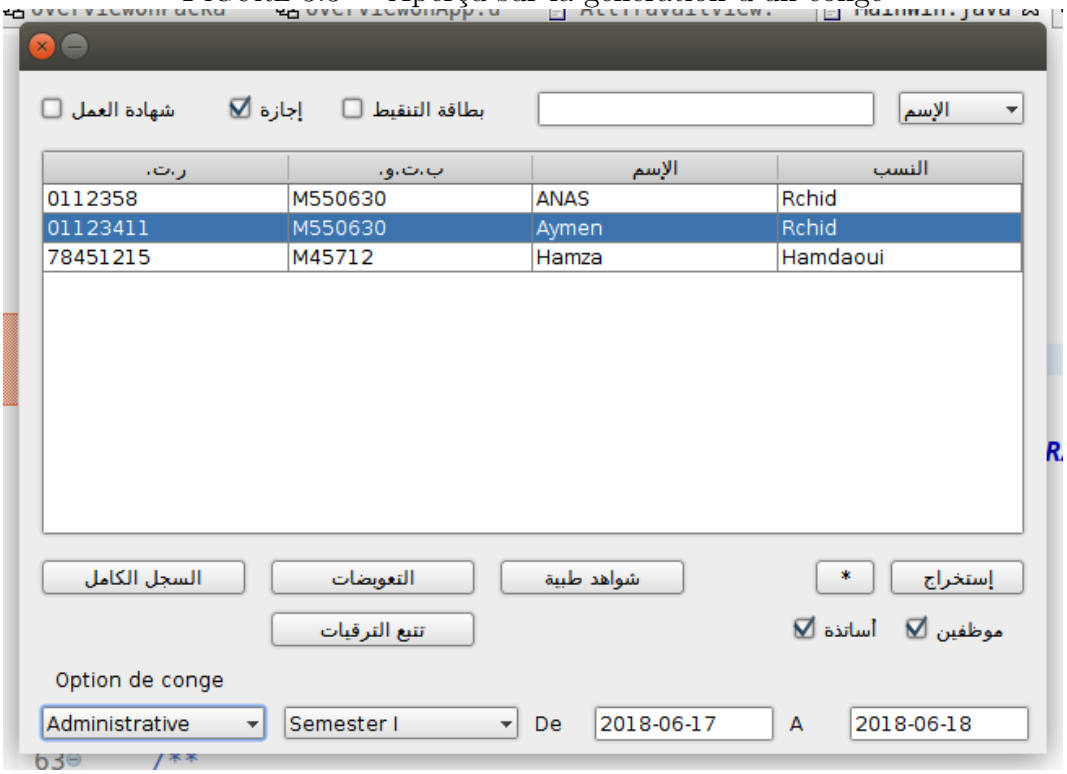


FIGURE 3.4 – Aperçu sur le suivi des avancements de grade

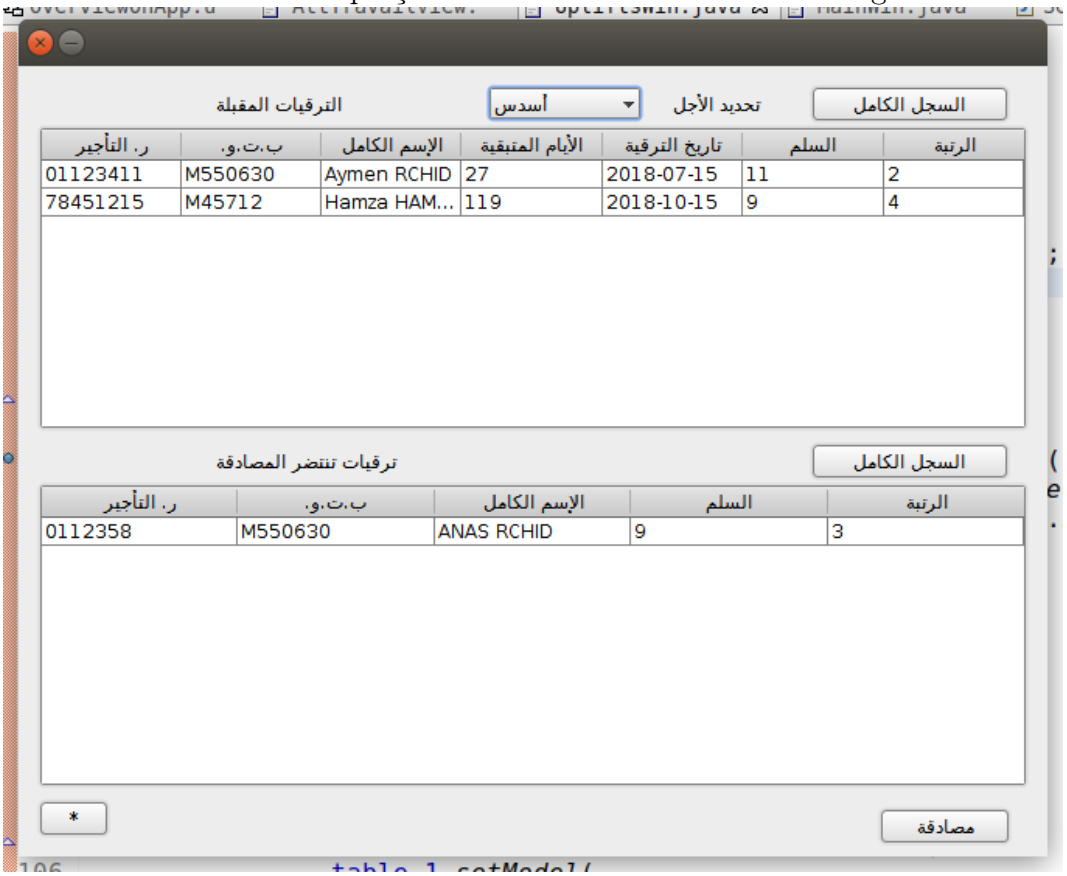
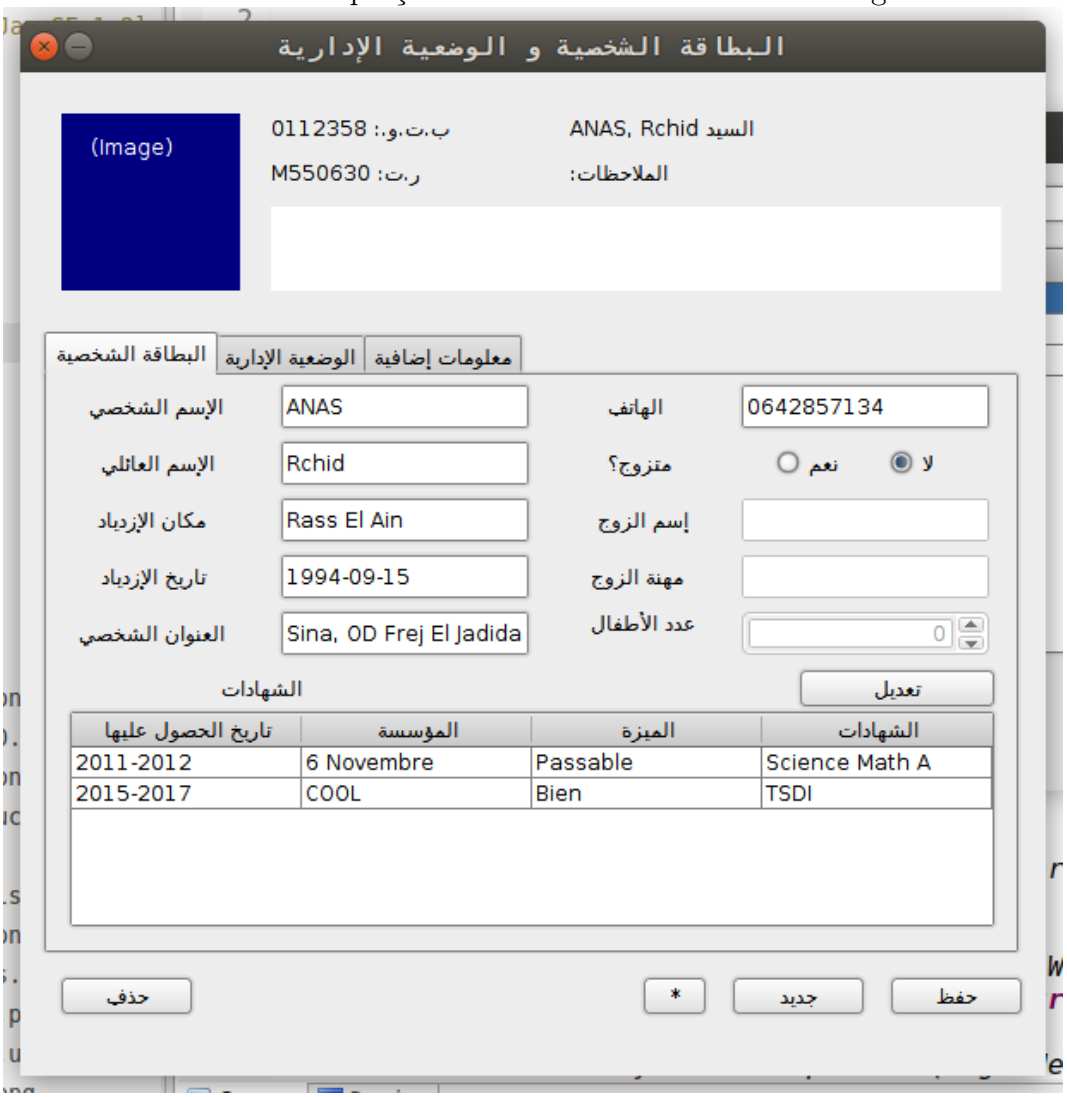


FIGURE 3.5 – Aperçu sur le suivi des avancements de grade



Cinquième partie

Dependencies

JDOM http://jdom.org/ Java library to to parse XML

WebLaF http://weblookandfeel.com/ Java library to enhance the look and feel