

# Table des matières

<b>I</b>	<b>Introduction</b>	<b>2</b>
<b>II</b>	<b>Cahier des Charges</b>	<b>4</b>
<b>III</b>	<b>Conception</b>	<b>6</b>
<b>1</b>	<b>Stockage des données en XML</b>	<b>7</b>
1.1	Règles de gestion du fichier XML . . . . .	7
1.2	Schéma général du fichier . . . . .	8
<b>2</b>	<b>Les Paquets system et leur Classes</b>	<b>9</b>
2.1	Paquet model . . . . .	10
2.2	Paquet app . . . . .	12
2.3	Paquet app.utils . . . . .	13
<b>IV</b>	<b>L'Interface Graphique</b>	<b>19</b>
<b>3</b>	<b>La relation entre les différent Paquet graphique</b>	<b>20</b>
3.1	Paquet wins . . . . .	21
3.2	Paquet wins.crud . . . . .	21
3.3	Paquet views . . . . .	21
<b>V</b>	<b>Dependencies</b>	<b>24</b>

# Table des figures

2.1	Aperçu de code source du projet . . . . .	9
2.2	La relation entre les classes du paquet <code>model</code> et la classe abstrait <code>XmlElement</code> du paquet <code>app.utils</code> . . . . .	10
2.3	Les enumeration dans paquet <code>app</code> . . . . .	12
2.4	La relation entre <code>JTable</code> , <code>JTableListener</code> , et <code>Printer</code> du paquet <code>app.utils</code> . . . . .	13
2.5	La relation entre <code>DateUtils</code> , <code>XmlFile</code> et <code>XmlElement</code> . . . . .	18
3.1	Aperçu général sur les interfaces graphiques utilisé dans l'application	20
3.2	Aperçu sur le démarrage de l'application . . . . .	21
3.3	Aperçu sur la génération d'un congé . . . . .	22
3.4	Aperçu sur le suivi des avancements de grade . . . . .	22
3.5	Aperçu sur le suivi des avancements de grade . . . . .	23

# List of Listings

1	Schéma général XML du fichier <code>data/xml/hr.xml</code> . . . . .	8
2	Extraits du classe générique <code>XmlElement</code> du paquet <code>app.utils</code> . . .	11
3	Extraits du classe <code>Diploma</code> qui montre l'héritage de <code>XmlElement</code> . .	11
4	Extraits de la classe <code>Printer</code> qui montre l'implémentation de la méthode abstrait <code>print()</code> de l'interface <code>java.awt.Printable</code> . . . . .	14
5	Extrait du classe <code>JTableListener</code> . . . . .	15
6	L'implémentation du <code>propertyChange()</code> de l'interface <code>Printable</code> dans la classe <code>JTableListener</code> . . . . .	16
7	Les fonctions <code>processEditingStarted()</code> et <code>processEditingStopped()</code> utilisée dans la méthode <code>propertyChange()</code> de la classe <code>JTableListener</code>	17
8	L'implémentation du <code>run()</code> du classe <code>JTableListener</code> . . . . .	18

## **Résumé**

Ce projet est le résultat d'un stage que j'ai passé à la Faculté des Lettres et Sciences Humaines, El Jadida sous le thème Informatisation du service Ressources Humaines. Sous l'encadrement de Mr. A. Madani, et la supervision du chef de service ; Mr. Driss Dibaji.

# Première partie

## Introduction

Pour obtenir ce besoin, J'avais la responsabilité de développer un environnement pour gérer les différentes tâches décrites en [cahier des charges](#)

Pour réaliser, il y a deux parties. Stockage des données et l'application bureau. Pour la première, J'ai choisi *XML* tout simplement parce qu'il est gratuit, alors c'est un langage de balises écrit dans un fichier texte, ainsi que la simplicité d'utilisation et de modification. Et pour la deuxième, j'ai développé une application en *Java*, parce qu'il est un langage Orienté-Objet, donc il facilite le processus de développement.

# Deuxième partie

## Cahier des Charges

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

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

- implémenter un système de gestion des employées/fonctionnaires
- gérée les diplômes et les grades
- suivi des grades
- suivi d'absence
- suivi de rémunération du travail les fériés
- générée des attestations de travail
- générée des autorisations de congé
- générée 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`. La raison de choisir ce type de stockage c'est que il est lisible soit au niveau du machine soit au niveau d'humain. Au suivant, les règles de gestion est le schéma général du fichier.

### 1.1 Règles de gestion du fichier XML

Le root-tag est `<Employee>` qui contient plusieurs tags de type `<employee>` qui représente des employés. Et chaque tag `<employee>` contient un seul tag `<personal>`, un seul tag `<administrative>` qui peut contenir 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 déclarés ci-dessus :

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

`<employee>` contient toute l'information d'un employé particulier. Il a deux attributs :

**reference** identifiant de l'employé

**département** département de l'employé. Certains employés n'appartiennent à aucun département. Ce sont des *fonctionnaires*

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

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

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

**id** identifiant de l'avancement par rapport à l'avancement précédent

`<diploma>` contient les informations sur les diplômes, *titre*, *mention*, *institué* et *session*. Ce tag a aussi un seul attribut, 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

```

1 <Employee>                                <!--root-->
2   <employee reference="" department="">
3
4   <notes />                                <!--les note sur l'employé-->
5
6   <personal>                                <!--les information personnelles-->
7   </personal>
8
9   <administrative>                          <!--les information administrative-->
10    <uplift id="" state="">                <!--les information d'avancement-->
11    </uplift>
12
13    <uplift id="" state="" /> <!--nous pouvons avoir plus-->
14  </administrative>
15
16  <diplomas id="">                          <!--les infomration du diplôme-->
17  </diplomas>
18  <diplomas id="" />                        <!--nous pouvons avoir plus-->
19
20  <medicalcertif id="">                     <!--information du certification médical-->
21  </medicalcertif>
22  <medicalcertif id="" />                  <!--nous pouvons avoir plus-->
23
24  <repayment id="">                         <!--information du remboursement-->
25  </repayment>
26  <repayment id="" />
27
28 </employee>
29 </Employee>

```

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

# Chapitre 2

## Les Paquets system et leur Classes

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

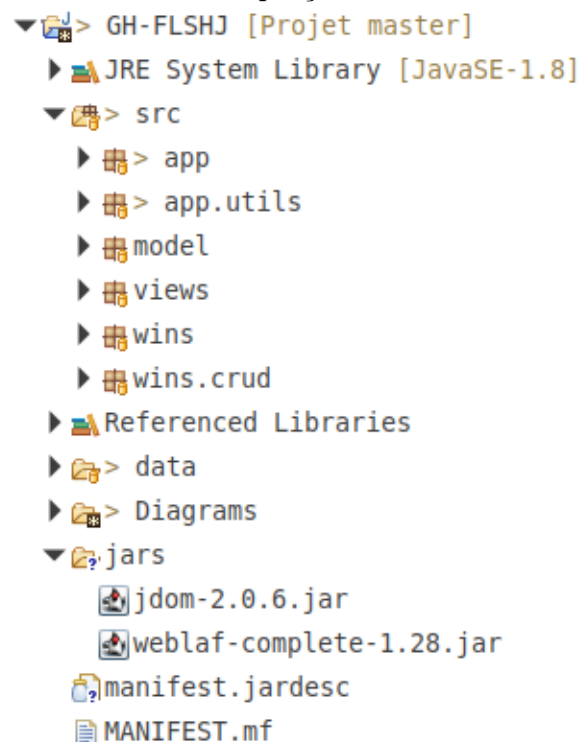
`model` contient les différent classe pour mobilisé les donnée en objet

`app` contient les différent énumération utilisé 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 existe dans `wins.crud`

`views` contient des pages générées pour l'impression

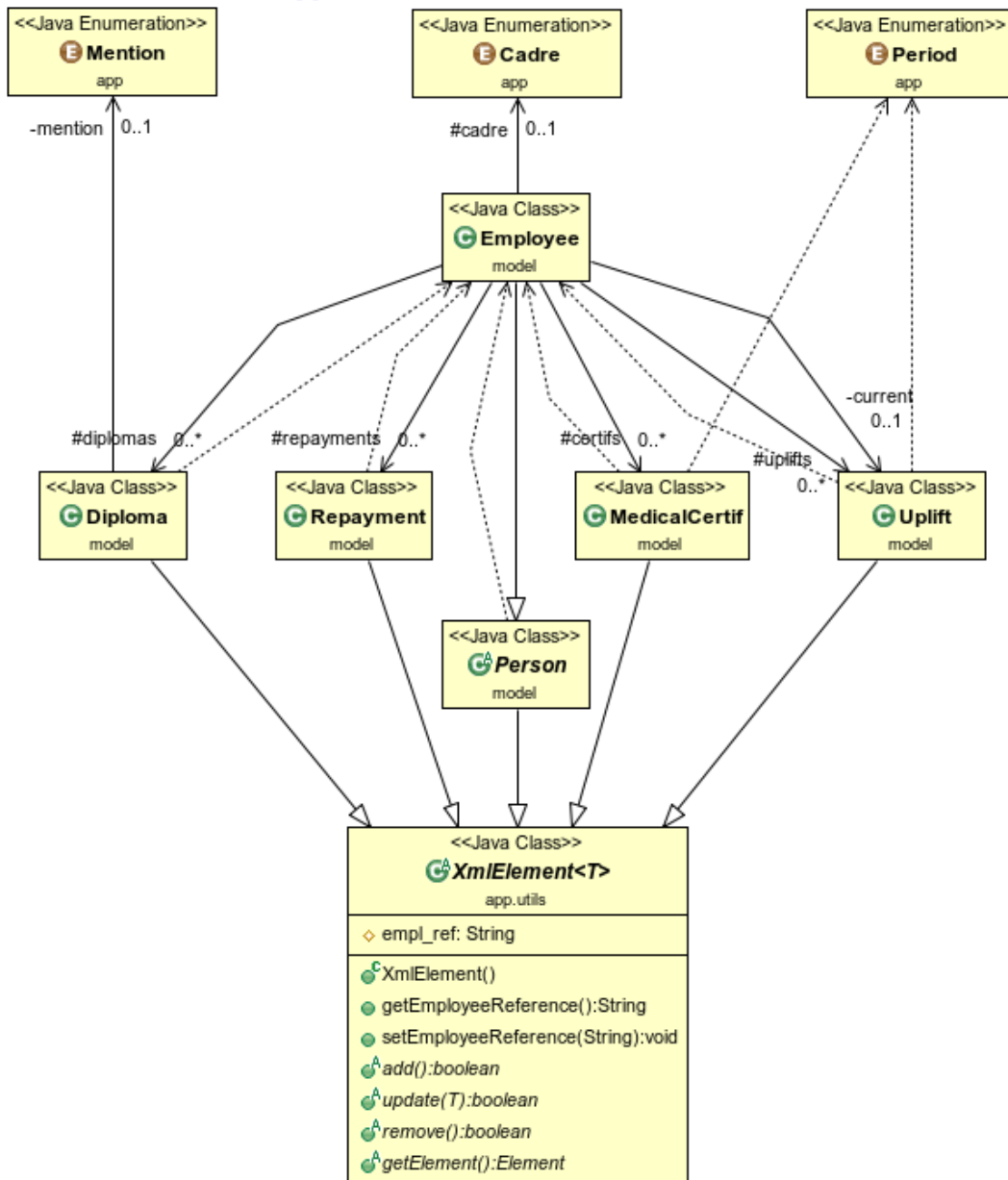
FIGURE 2.1 – Aperçu de code source du projet



## 2.1 Paquet model

Ce paquet contient les modèles de l'application, ce sont des classes Java pour modéliser les informations stockée dans le fichier XML [data/xml/hr.xml](#).

FIGURE 2.2 – La relation entre les classes du paquet [model](#) et la classe abstraite 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 a l'objet concerné dans le [fichier xml](#).

Voici la classe mère de toutes les classes, [XmlElement](#), qui contient en addition,

une chaîne de caractères qui représente le référence de l'employée, c.-à-d. L'identifiant

```
1 import org.jdom2.Element;
2
3 public abstract class XmlElement<T> {
4     public abstract boolean add();
5     public abstract boolean update(T updated);
6     public abstract boolean remove();
7     public abstract Element getElement();
8
9     /* référence du employée */
10    protected String empl_ref;
11    public String getEmployeeReference( ) {
12        return empl_ref;
13    }
14
15    public void setEmployeeReference(String ref) {
16        this.empl_ref = ref;
17    }
18 }
```

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

Les méthodes `add()`, `update()` et `remove()` de `XmlElement` retourne une valeur booléen, qui signifie est ce que l'opération a réussi 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 :

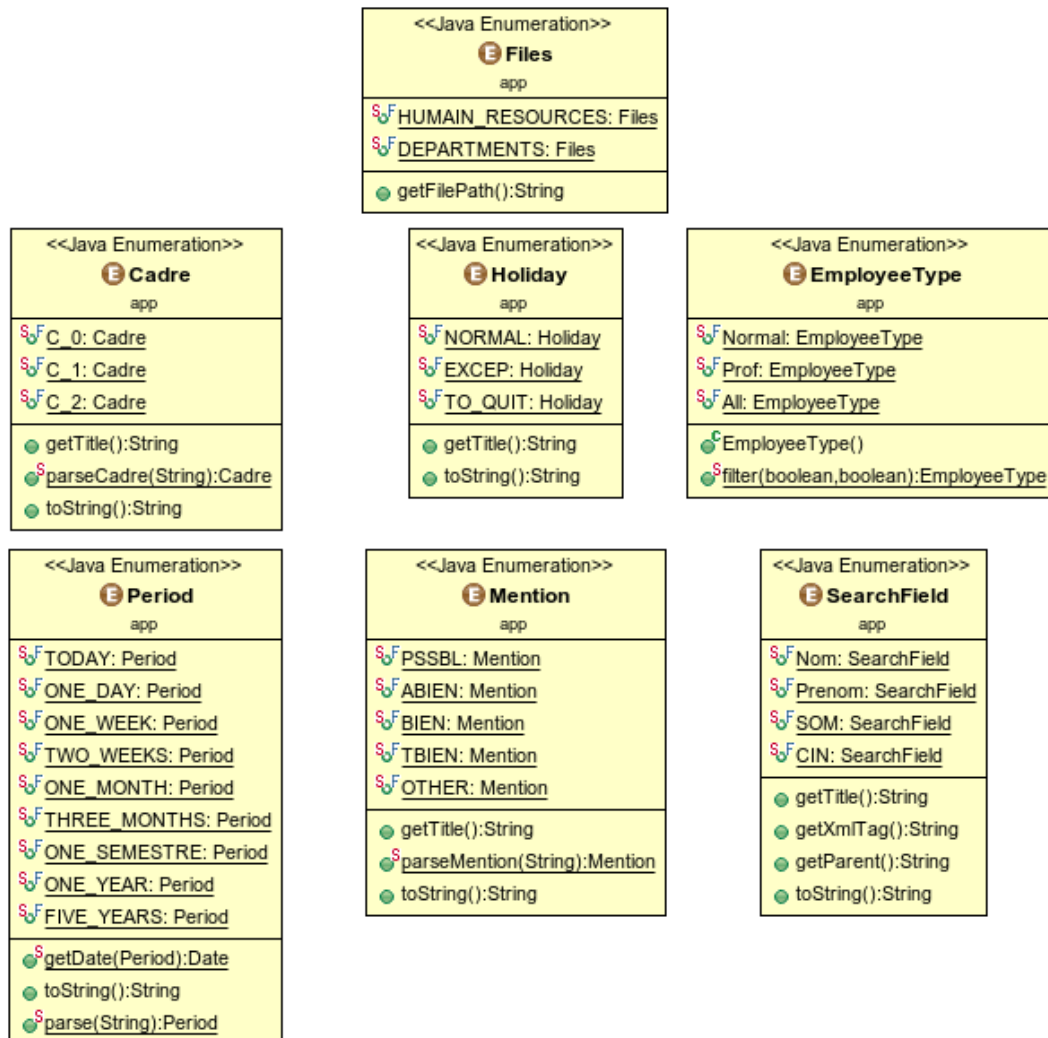
```
1 public class Diploma extends XmlElement<Diploma> {
2     /* les attributs du classe */
3
4     @Override
5     public boolean update(Diploma updated) {
6         /* process la mise à jour */
7     }
8 }
```

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

## 2.2 Paquet app

Le paquet `app` contient que les énumération, décrits dans le figure 2.3.

FIGURE 2.3 – Les enumeration dans paquet `app`



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

`Mention` les mentions possibles pour un diplôme, utilisé dans [DilpomaCrud.java](#)

`Period` les différents périodes utilisé 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](#)

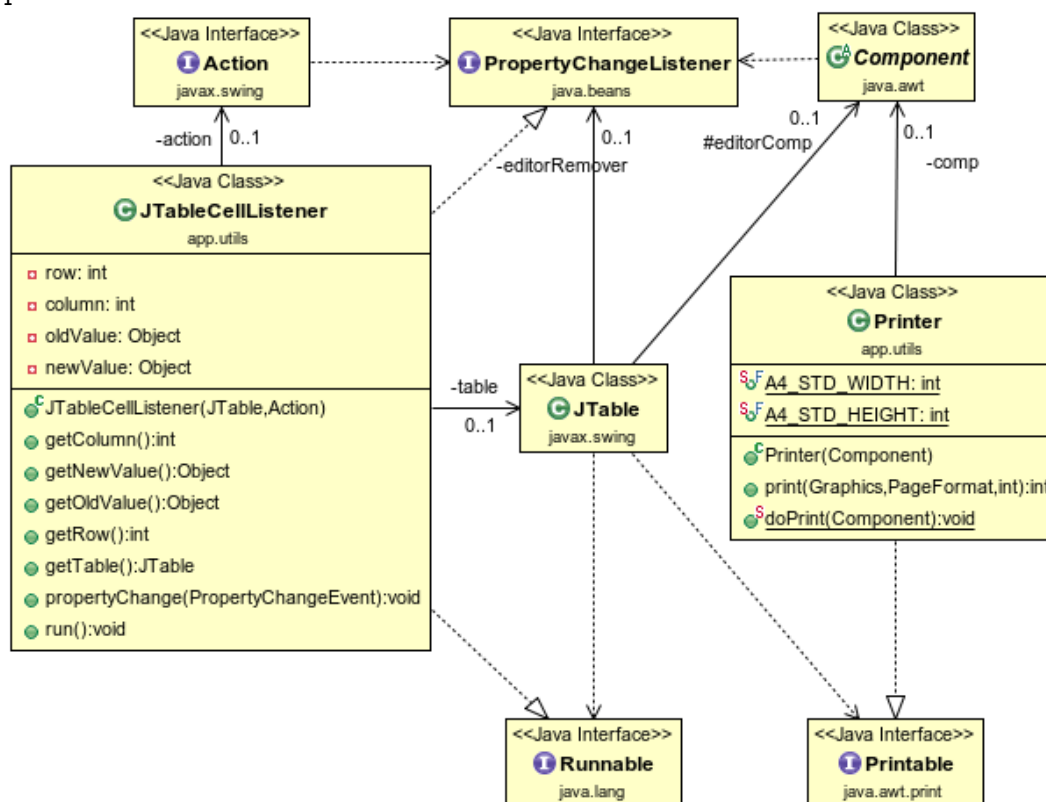
`Files` contient des énumérations qui concerne les différents fichiers XML utilisé.

## 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 a 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 abstraite `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()`.



```
1  import java.awt.Component;
2  import java.awt.Dimension;
3  import java.awt.Graphics;
4  import java.awt.Graphics2D;
5  import java.awt.print.PageFormat;
6  import java.awt.print.Printable;
7
8  public class Printer implements Printable {
9      /**
10       * méthode abstrait dans l'interface java.awt.Printable, l'appelle
11       * se fait par la classe java.awt.PrinterJob
12       *
13       * @param g une graphique du classe java.awt.Graphics
14       * @param format formatage de la page à imprimer
15       * @param page_index l'index de la page
16       *
17       * @return intègre qui représente l'état de l'impression */
18     @Override
19     public int print(Graphics g, PageFormat format, int page_index) {
20         /* vérification du page, c'est un protocole de l'interface */
21         if (page_index > 0) return Printable.NO_SUCH_PAGE;
22
23         /* prend les dimensions du composant */
24         Dimension dim = comp.getSize( );
25         double cHeight = dim.getHeight( ), cWidth = dim.getWidth( );
26
27         /* initialisation de demention du la zone d'impression */
28         double pHeight = format.getImageableHeight( );
29         double pWidth = format.getImageableWidth( );
30         double pXStart = format.getImageableX( );
31         double pYStart = format.getImageableY( );
32
33         /* la difference entre la taille du composant et la taille du
34          * page pour mettre le composant à l'échelle du page */
35         double xRatio = (pWidth / cWidth), yRatio = (pHeight / cHeight);
36
37         /* mettre le composant à jour avec la page à imprimer en utilisent
38          * la methode java.awt.Component.paint() */
39         Graphics2D g2 = (Graphics2D) g;
40         g2.translate(pXStart, pYStart);
41         g2.scale(xRatio, yRatio);
42         comp.paint(g2);
43         return Printable.PAGE_EXISTS;
44     }
45 }
```

Listing 4: Extraits 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 appeler des classe selon l'état de l'édition, donc on doit aussi implémenter la méthode `run()` dans l'interface `Runnable`. Au suivant un extrait du classe `JTableCellListener`. Au suivant, des extraits du classe et ses méthodes.

La classe possède un constructeur privé, qui prend en paramètre, la table, numéro du ligne et colonne, et les deux valeurs, l'ancien et nouvelle. Ce constructeur 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   * ancienne/nouvelle 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 cours de la modification */
58          if (table.isEditing( )) processEditingStarted( );
59          /* si l'édition 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 modifiée 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   * annoncée 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()` utilisée 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`.*

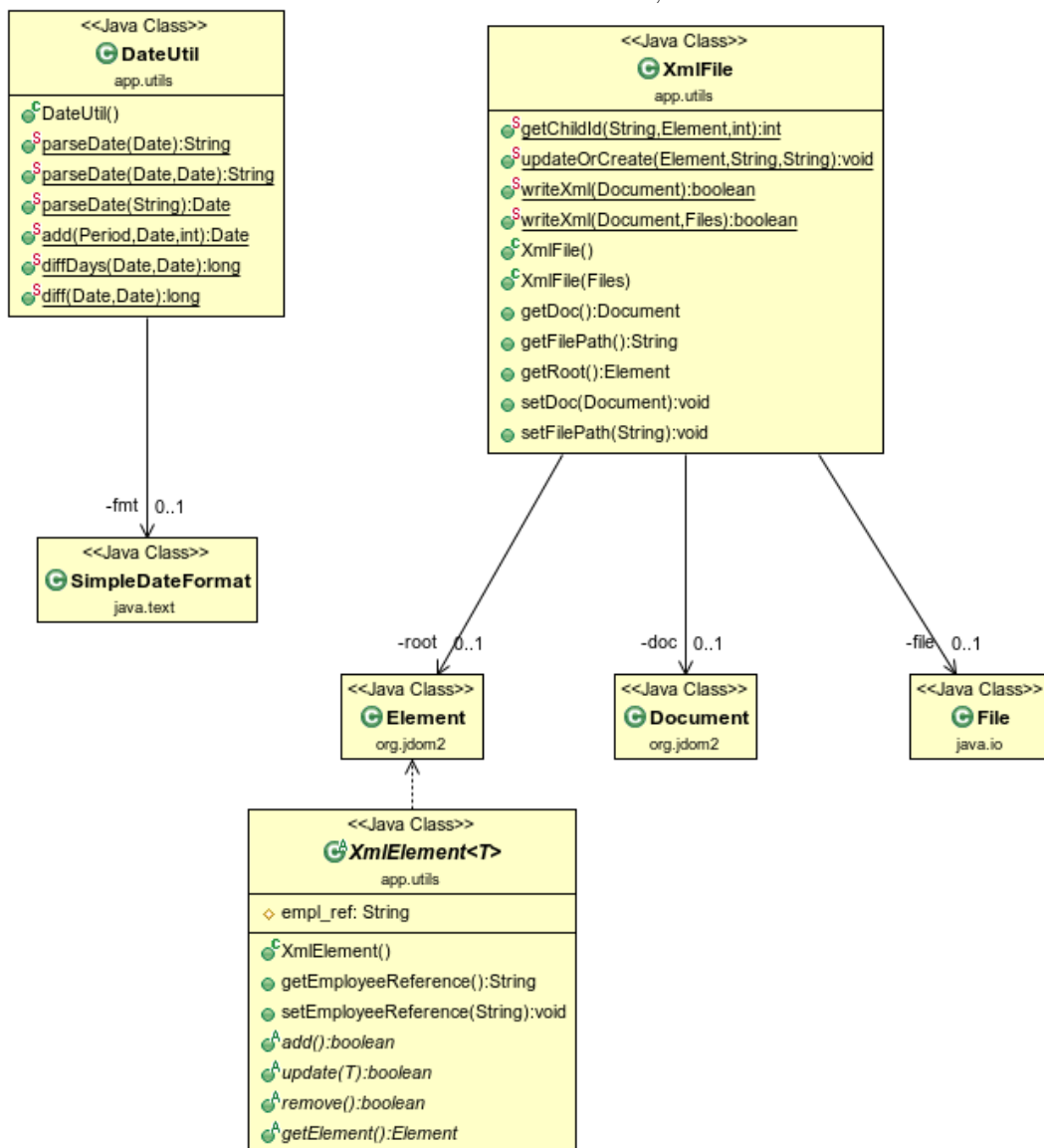
```

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



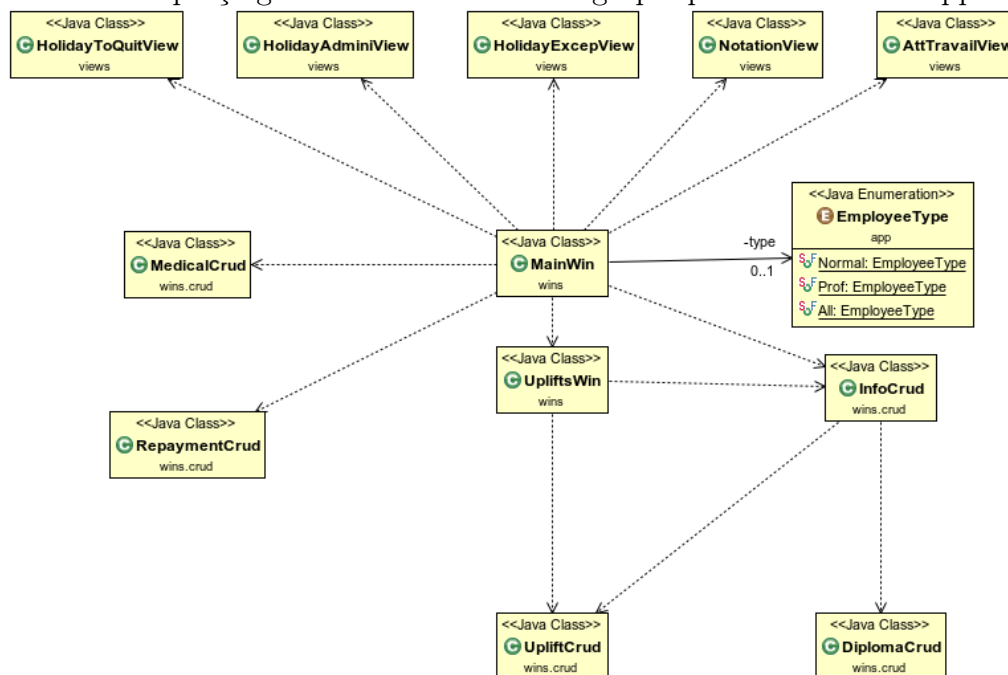
Quatrième partie

L'Interface Graphique

## Chapitre 3

# La relation entre les différent Paquet graphique

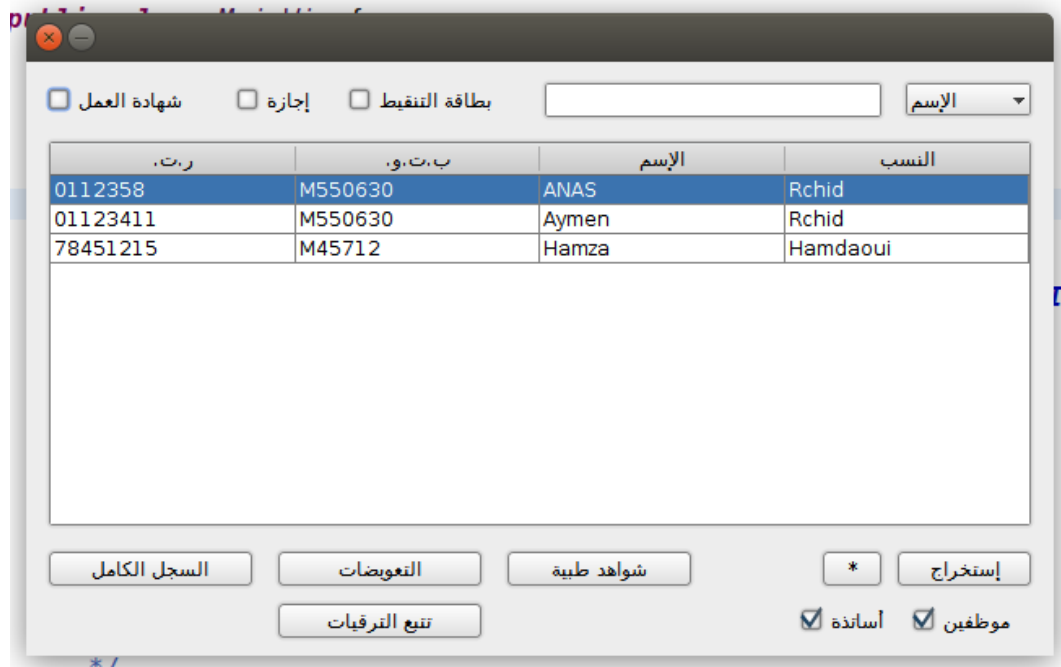
FIGURE 3.1 – Aperçu général sur les interfaces graphiques utilisé dans l'application



## 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

☐ شهادة العمل
 ☒ إجازة
 ☐ بطاقة التنقيط
 
 الإسم ▼

النسب	الإسم	ب.ت.و.	ر.ت.
Rchid	ANAS	M550630	0112358
Rchid	Aymen	M550630	01123411
Hamdaoui	Hamza	M45712	78451215

السجل الكامل    التعويضات    شواهد طبية    \*    إستخراج

تتبع الترقية    ☒ أساتذة    ☒ موظفين

Option de conge

Administrative ▼    Semester I ▼    De    2018-06-17    A    2018-06-18

FIGURE 3.4 – Aperçu sur le suivi des avancements de grade

الترقيات المقبلة    أسدس ▼    تحديد الأجل    السجل الكامل

الرتبة	الاسم الكامل	الأيام المتبقية	تاريخ الترقية	الاسم الكامل	ب.ت.و.	ر. التأجير
2	Aymen RCHID	27	2018-07-15	M550630	01123411	
4	Hamza HAM...	119	2018-10-15	M45712	78451215	

ترقيات تنتظر المصادقة

السجل الكامل

الرتبة	الاسم الكامل	ب.ت.و.	ر. التأجير
3	ANAS RCHID	M550630	0112358

\*    مصادقة

FIGURE 3.5 – Aperçu sur le suivi des avancements de grade

البطاقة الشخصية و الوضعية الإدارية

(Image) السيد Rchid, ANAS 0112358 ب.ت.و.:  
الملاحظات: ر.ت: M550630

البطاقة الشخصية الوضعية الإدارية معلومات إضافية

الاسم الشخصي ANAS الهاتف 0642857134  
الاسم العائلي Rchid متزوج؟ ☐ نعم ☒ لا  
مكان الإزدياد Rass El Ain إسم الزوج  
تاريخ الإزدياد 1994-09-15 مهنة الزوج  
العنوان الشخصي Sina, OD Frej El Jadida عدد الأطفال 0

الشهادات تعديل

تاريخ الحصول عليها	المؤسسة	الميزة	الشهادات
2011-2012	6 Novembre	Passable	Science Math A
2015-2017	COOL	Bien	TSDI

حذف \* جديد حفظ

# 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