

# 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 et leurs Classes</b>	<b>9</b>
2.1	Paquet <code>model</code> . . . . .	10
2.2	Paquet <code>app</code> . . . . .	12
2.3	Paquet <code>app.utils</code> . . . . .	13
<b>IV</b>	<b>L'Interface Graphique</b>	<b>23</b>
<b>3</b>	<b>La relation entre les différents Paquets graphiques</b>	<b>24</b>
3.1	Paquet <code>wins</code> . . . . .	25
3.2	Paquet <code>wins.crud</code> . . . . .	27
3.3	Paquet <code>views</code> . . . . .	29
<b>V</b>	<b>Conclusion</b>	<b>30</b>
<b>VI</b>	<b>Dépendances</b>	<b>32</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 abstraite <code>XmlElement</code> du paquet <code>app.utils</code> . . . . .	10
2.3	Les énumérations dans le 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	24
3.2	Aperçu sur le démarrage de l'application . . . . .	25
3.3	Aperçu sur le suivi des avancements de grade . . . . .	26
3.4	Aperçu sur la fiche des informations personnelle et administrative .	27

# List of Listings

1	Schéma général XML du fichier <code>data/xml/hr.xml</code> . . . . .	8
2	Extrait du classe générique <code>XmlElement</code> du paquet <code>app.utils</code> . . .	11
3	Extrait du classe <code>Diploma</code> qui montre l'héritage de <code>XmlElement</code> . .	11
4	Extrait de la classe <code>Printer</code> qui montre l'implémentation de la méthode abstraite <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> utilisées dans la méthode <code>propertyChange()</code> . . . . .	17
8	L'implémentation du <code>run()</code> du classe <code>JTableListener</code> . . . . .	18
9	Extrait de la classe <code>DateUtil</code> . . . . .	19
10	La méthode <code>parseDate()</code> de la classe <code>DateUtil</code> . . . . .	19
11	La définition de la méthode <code>diffDays()</code> de la classe <code>DateUtil</code> . . .	19
12	La surcharge de la méthode <code>parseDate()</code> de la classe <code>DateUtil</code> . .	20
13	La définition de la méthode <code>diff()</code> du classe <code>DateUtil</code> . . . . .	20
14	Extrait de la classe <code>XmlFile</code> du paquet <code>app.utils</code> . . . . .	21
15	Définition de la méthode <code>writeXml</code> de la classe <code>XmlFile</code> . . . . .	22

## Résumé

Ce projet est le résultat d'un stage que j'avais passé chez la *Faculté des Lettres et Science Humaines*, El Jadida sous le thème *informatiser le service Ressources Humaines*. Sous l'encadrement de **Mr. A. Madani**, et la supervision du chef de service ; **Mr. Driss DIBAJI**.

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.

Tout d'abord, j'aimerais remercier **Mr. Abdellah MADANI** pour sa patience, pour son suivie interrompu, pour ses conseils judicieux qui m'ont aidé à mener à bout se travail et son appui tout au long de ce projet.

J'adresse mes remerciements au mes professeurs, **Mr. Mohammed Essaid RIFFI**, **Mr. Hassan SILKAN** et finalement **Mme Salwa BELAQZIZ**, qui m'ont beaucoup aidé pendant cette formation, c'était un grand plaisir.

Je tiens à remercier vivement mon maitre de stage, **Mr Driss DIBAJI**, chef du service Ressources Humaines au sein de la *Faculté des Lettres et Science Humaines*, El Jadida, pour son accueil et le partage de son expertise. Grâce aussi à sa confiance j'ai pu m'accomplir totalement dans mes missions. Il fut d'une aide précieuse dans les moments les plus délicats.

Enfin, je tiens à remercier toutes les personnes qui m'ont conseillé et relu lors de la rédaction de ce rapport de stage : ma famille, mon ami **H. Hamza**, **R. Yassine** et **B. Aya**.

# Première partie

## Introduction

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 est gratuit. Et pour la deuxième, j'ai développé une application en *Java*, car il est un langage Orienté-Objet qui facilite le processus de développement.

# Deuxième partie

## Cahier des Charges

*Service des ressources humaines* est un service qui est responsable de la gestion des employés et fonctionnaires, leurs diplômes et grades, ainsi que donner des attestations du travail et des autorisations de congé, suivi d'absence, suivi des remboursements pour travailler les jours de congé et finalement donner des fiches notation annuelle.

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

- Implémenter un système de gestion des employés/fonctionnaires
- Gérer les diplômes et les grades
- Suivi des grades
- Suivi d'absence
- Suivi des remboursements pour travailler les jours de congé
- Générer des attestations de travail
- Générer des autorisations de congé
- Générer des fiches de notation annuelle



# 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 de gestion et 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 employés. Chaque tag `<employee>` contient un seul tag `<personal>` et 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 déclaré ci-dessus :

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

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

**reference** identifiant de l'employé

**departement** 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*, *institut* 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 notes sur l'employé-->
5
6    <personal>                                <!--les informations personnelles-->
7    </personal>
8
9    <administrative>                          <!--les informations administrative-->
10     <uplift id="" state="">                <!--les informations 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 certificat médical-->
21 </medicalcertif>
22 <medicalcertif id="" />                     <!--nous pouvons avoir plus-->
23
24 <repayment id="">                           <!--information du remboursement-->
25 </repayment>
26 <repayment id="" />                         <!--nous pouvons avoir plus-->
27
28 </employee>
29 </Employee>

```

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

# Chapitre 2

## Les Paquets et leurs Classes

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

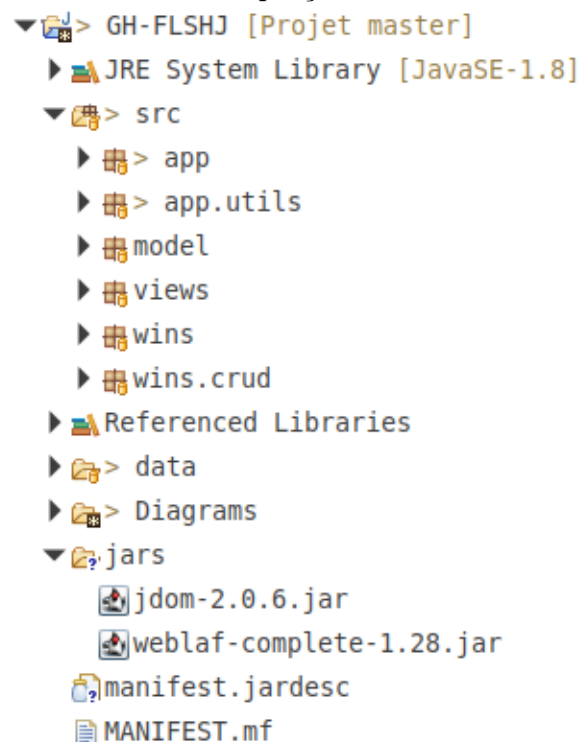
`model` contient les différentes classe pour mobilisé les donnée en objet

`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.

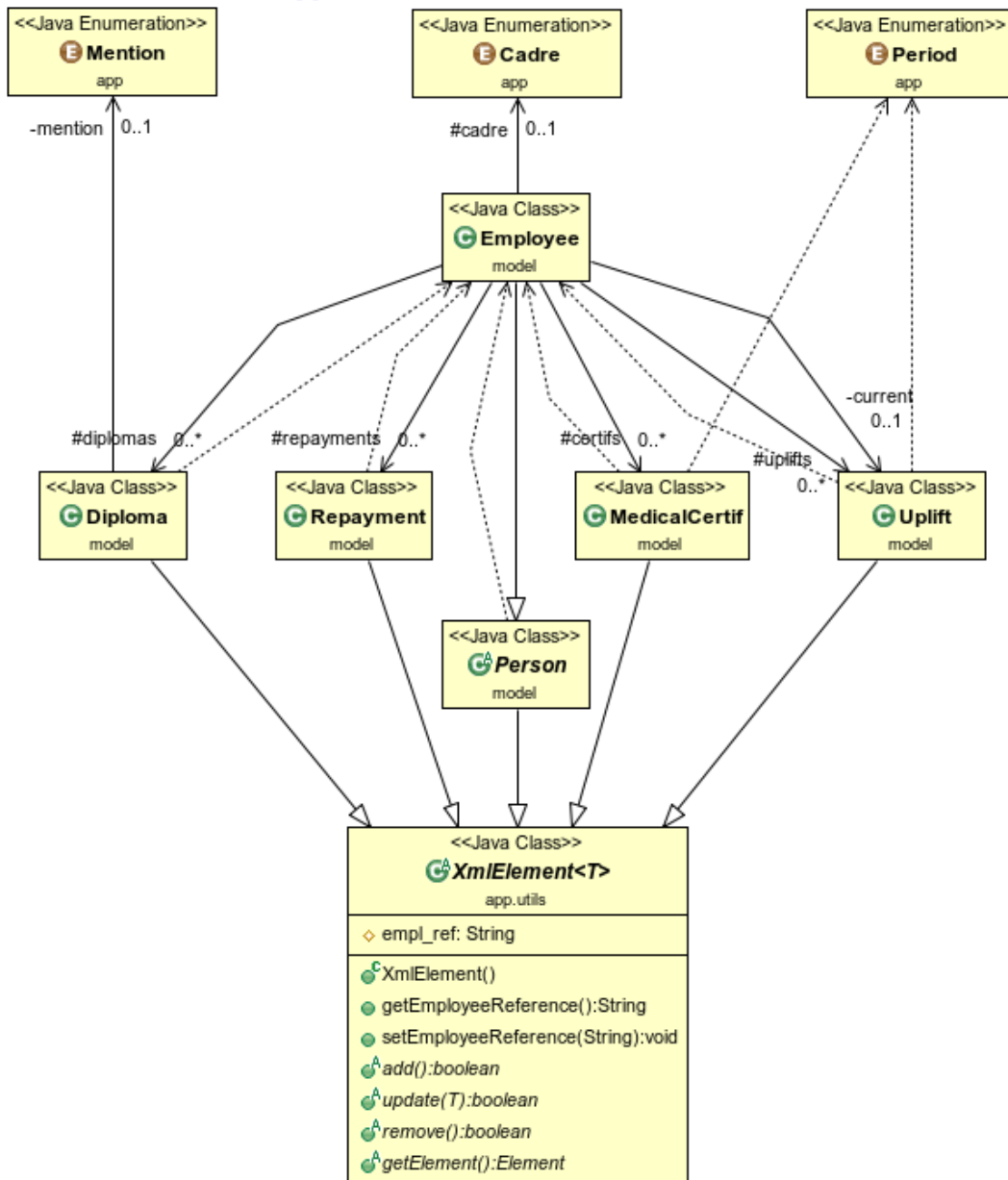
FIGURE 2.1 – Aperçu de code source du projet



## 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 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 à 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'employé, 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 employé */
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: Extrait 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 :

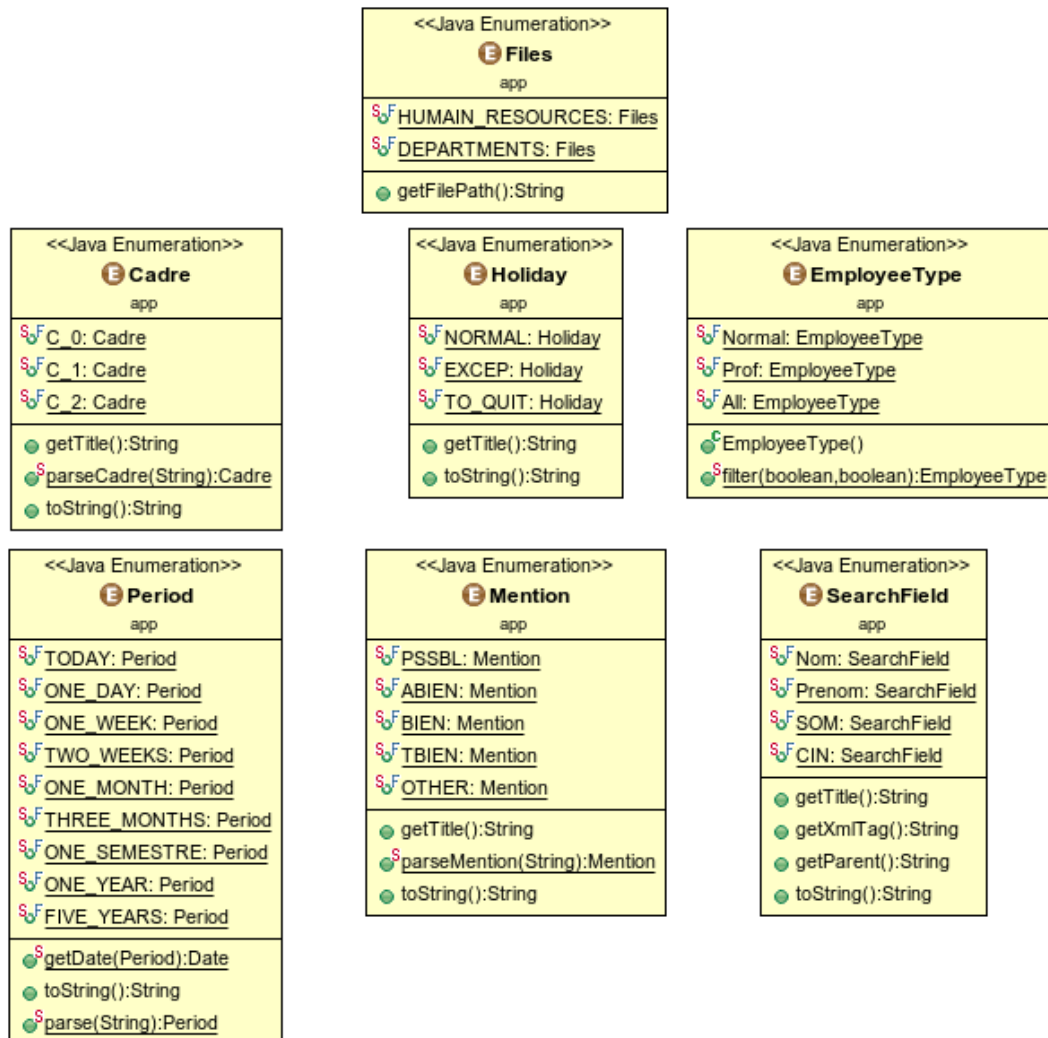
```
1 public class Diploma extends XmlElement<Diploma> {
2     /* les attributs du classe */
3
4     @Override
5     public boolean update(Diploma updated) {
6         /* process la mise à jour */
7     }
8 }
```

Listing 3: Extrait 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 énumérations dans le paquet `app`



`Cadre` représente les cadres possibles pour un employé/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 employé et fonctionnaire dans `MainWin.java`

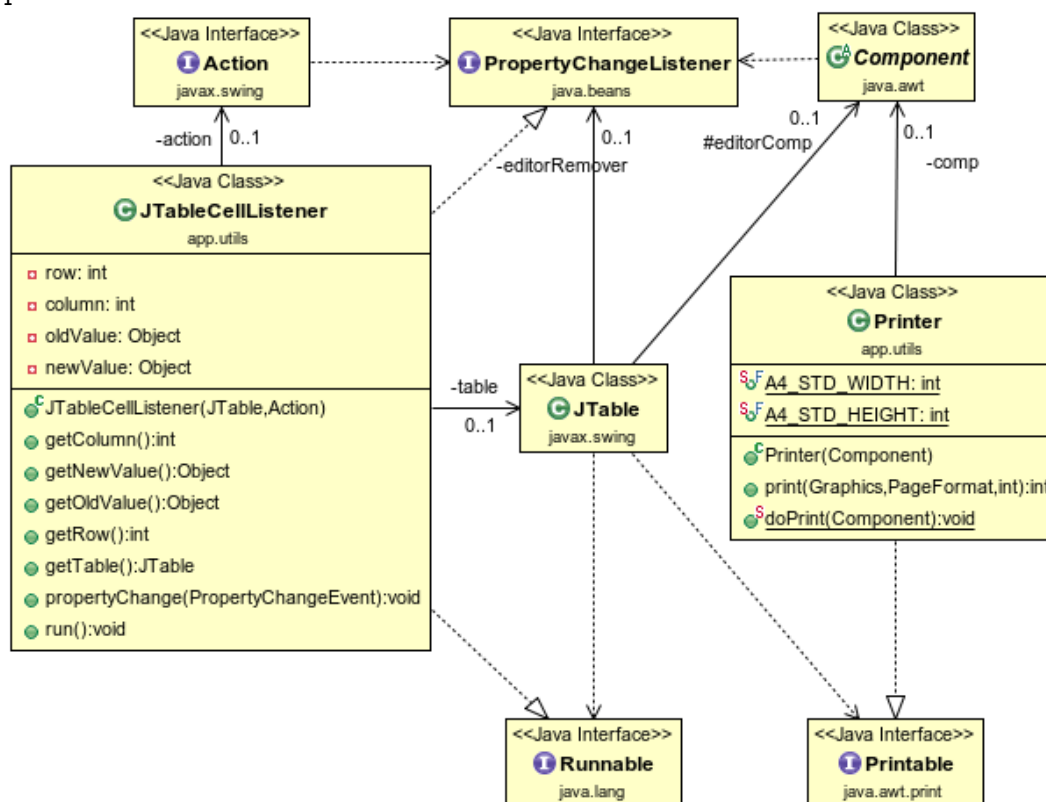
`Files` concernent les différents fichiers XML utilisé.

## 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 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      private final Component comp;
10     /**
11      * méthode abstraite dans l'interface java.awt.Printable, l'appel
12      * se fait par la classe java.awt.PrinterJob
13      *
14      * @param g une graphique du classe java.awt.Graphics
15      * @param format formatage de la page à imprimer
16      * @param page_index l'index de la page
17      *
18      * @return intègre qui représente l'état de l'impression
19      */
20     @Override
21     public int print(Graphics g, PageFormat format, int page_index) {
22         /* vérification du page, c'est un protocole de l'interface */
23         if (page_index > 0) return Printable.NO_SUCH_PAGE;
24
25         /* prend les dimensions du composant */
26         Dimension dim = comp.getSize( );
27         double cHeight = dim.getHeight( ), cWidth = dim.getWidth( );
28
29         /* initialization de demention du la zone d'impression */
30         double pHeight = format.getImageableHeight( );
31         double pWidth = format.getImageableWidth( );
32         double pXStart = format.getImageableX( );
33         double pYStart = format.getImageableY( );
34
35         /* la difference entre la taille du composant et la taille du
36          * page pour mettre le composant à l'échelle du page */
37         double xRatio = (pWidth / cWidth), yRatio = (pHeight / cHeight);
38
39         /* mettre le composant à jour avec la page à imprimer en utilisent
40          * la methode java.awt.Component.paint() */
41         Graphics2D g2 = (Graphics2D) g;
42         g2.translate(pXStart, pYStart);
43         g2.scale(xRatio, yRatio);
44         comp.paint(g2);
45         return Printable.PAGE_EXISTS;
46     }
47 }

```

Listing 4: Extrait de la classe Printer qui montre l'implémentation de la méthode abstraite 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 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` 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   * 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 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   * 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()`

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`.*

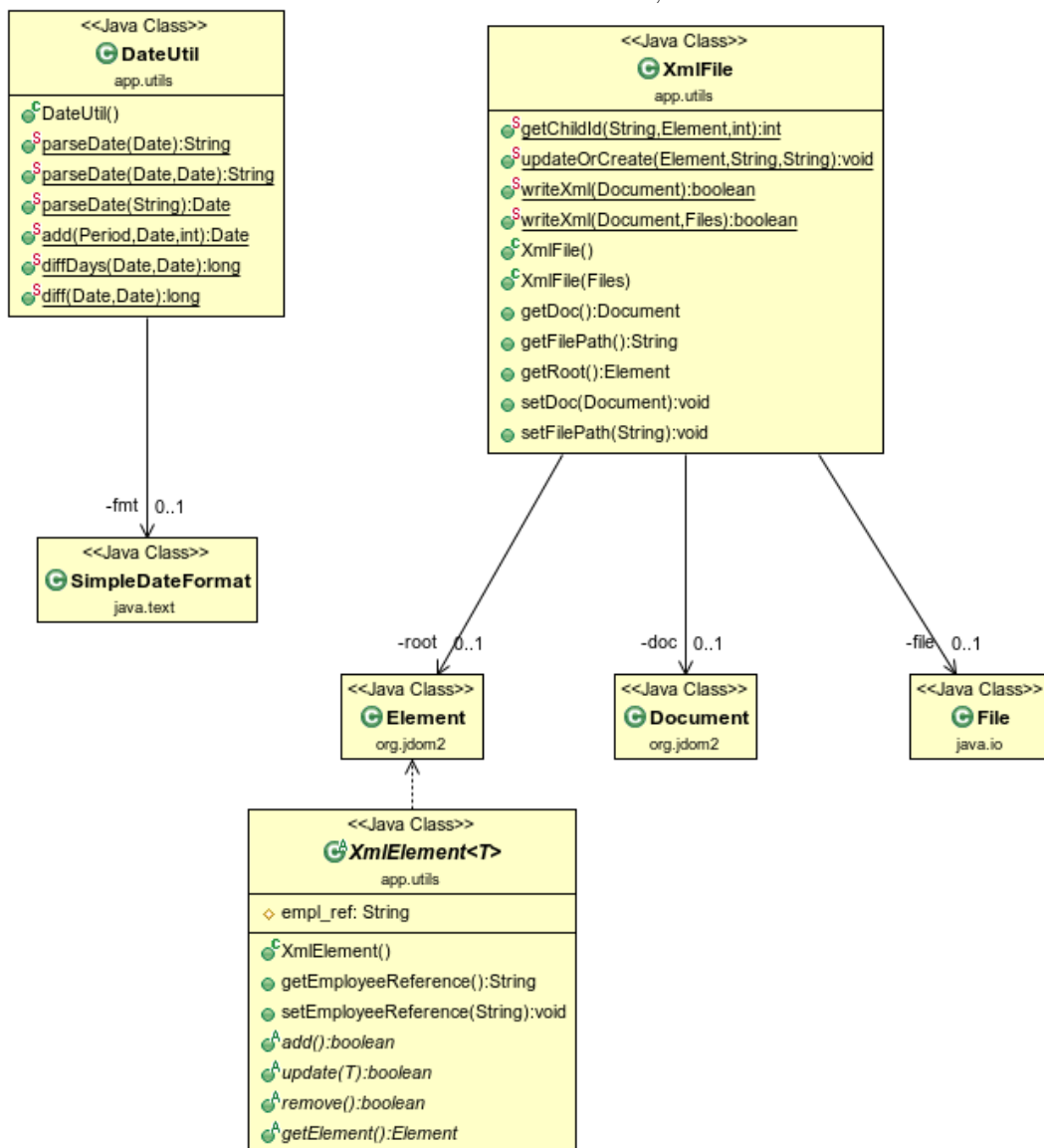
```

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



La classe `DateUtil` dans la figure 2.5 est utilisé pour la manipulation des dates, et la conversion des dates de/en `String` avec l'aide du classe system `SimpleDateFormat`. Pour les dates, j'ai choisi un format standard, `YYYY-MM-DD`, pour tous les dates dans le projet.

```

1  import java.text.ParseException;
2  import java.text.SimpleDateFormat;
3  import java.util.Date;
4
5  public class DateUtil {
6      private SimpleDateFormat fmt;
7
8      public DateUtil() {
9          fmt = new SimpleDateFormat("yyyy-MM-dd");
10     }
11 }

```

Listing 9: Extrait de la classe `DateUtil`

La classe `DateUtil` contient une définition d'une méthode, `static parseDate()`, qui a une surcharge pour la conversion de `Date` vers `String` et vice versa

```

12  /**
13   * Convertir une date en une chaine des caractères
14   */
15  public static String parseDate(Date date) {
16      try {
17          return new DateUtil( ).fmt.format(date);
18      } catch (ParseException e) {
19          System.err.println(e.getMessage( ));
20          return new DateUtil( ).fmt.format(new Date( ));
21      }
22  }

```

Listing 10: La méthode `parseDate()` de la classe `DateUtil`

La classe aussi contient la méthode `static diffDays()` pour calculer le nombre des jours entre deux dates, avec une aide de la méthode `static diff()`

```

1  /* voir la méthode diff() */
2  public static long diffDays(Date from, Date to) {
3      return TimeUnit.MILLISECONDS.toDays(diff(from, to));
4  }

```

Listing 11: La définition de la méthode `diffDays()` de la classe `DateUtil`

```

1  /**
2   * Convertir une chaîne des caractères vers une Date
3   */
4  public static Date parseDate(String str) {
5      Date d;
6
7      try {
8          d = new DateUtil( ).fmt.parse(str);
9      } catch (ParseException e) {
10         System.err.println(e.getMessage( ));
11         d = new Date( );
12     }
13
14     return d;
15 }

```

Listing 12: La surcharge de la méthode `parseDate()` de la classe `DateUtil`

```

1  /**
2   * Calculé la différence de nombre des seconds
3   * entre les deux dates
4   */
5  public static long diff(Date from, Date to) {
6      if (from != null && to != null) {
7          return to.getTime( ) - from.getTime( );
8      } else return 0;
9  }

```

Listing 13: La définition de la méthode `diff()` du classe `DateUtil`

Dans la figure 2.5 aussi, il y a la classe `XmlFile`, la couche *DAO* de l'application. Elle est responsable a tout interaction avec le `fichier XML`.

Avec l'aide de `static updateOrCreate()` on peut faire une mise à jour a une valeur d'un tag dans le fichier XML.

```

1  public static void updateOrCreate(Element el, String node, String value) {
2      Element foo = el.getChild(node);
3
4      if (foo == null) {
5          el.addContent(new Element(node).addContent(value));
6          writeXml(el.getDocument( ));
7      } else {
8          foo.setText(value);
9      }
10 }

```

```
1  import java.io.File;
2  import java.io.IOException;
3
4  import org.jdom2.Document;
5  import org.jdom2.Element;
6  import org.jdom2.JDOMException;
7  import org.jdom2.input.SAXBuilder;
8
9  import app.Files;
10
11 public class XmlFile {
12     private String filepath;
13     private File file;
14     private Document doc;
15     private Element root;
16
17     public XmlFile(Files file) {
18         setFilePath(file.getFilePath( ));
19     }
20
21     /**
22      * À chaque fois on change l'emplacement du fichier, on doit
23      * initialiser le Document XML ainsi que la racine du document
24      *
25      * @param filepath l'emplacement du fichier
26      */
27     public void setFilePath(String filepath) {
28         try {
29             this.file = new File(this.filepath = filepath);
30             if (file.exists( )) {
31                 this.doc = new SAXBuilder( ).build(this.file);
32                 this.root = doc.getRootElement( );
33             } else {
34                 this.doc = new Document( );
35                 this.root = new Element("Employee");
36             }
37         } catch (JDOMException | IOException e) {
38             System.out.println(e.getMessage( ));
39         }
40     }
41
42 }
```

Listing 14: Extrait de la classe XmlFile du paquet app.utils



Et on peut aussi sauvegarder les changements dans le fichier avec la méthode `static writeXml()`.

```

1  /**
2   * Écrire une instance de la classe Document dans un fichier XML
3   *
4   * @param doc le Document XML
5   * @param f les fichiers systèmes de l'application
6   *
7   * @return true si l'écriture est successive
8   */
9  public static boolean writeXml(Document doc, Files f) {
10     try {
11         XMLOutputter xmlout = new XMLOutputter( );
12         xmlout.setFormat(Format.getPrettyFormat( ));
13         xmlout.output(doc, new FileWriter(f.getFilePath( )));
14         System.err.println("success " + f.getFilePath( ));
15         return true;
16     } catch (IOException e) {
17         System.err.println(e.getMessage( ));
18         return false;
19     }
20 }

```

Listing 15: Définition de la méthode `writeXml` de la classe `XmlFile`

La classe `XmlFile` est basée sur un paquet Java appelée `JDOM`, ce dernier contient des classes important comme :

`Element` la représentation des tags XML en objet

`Document` la représentation du document XML en objet

`SAXBuilder` pour initialiser les instances `Document`

`XMLOutputter` pour l'écriture de `Document` et fichier réel

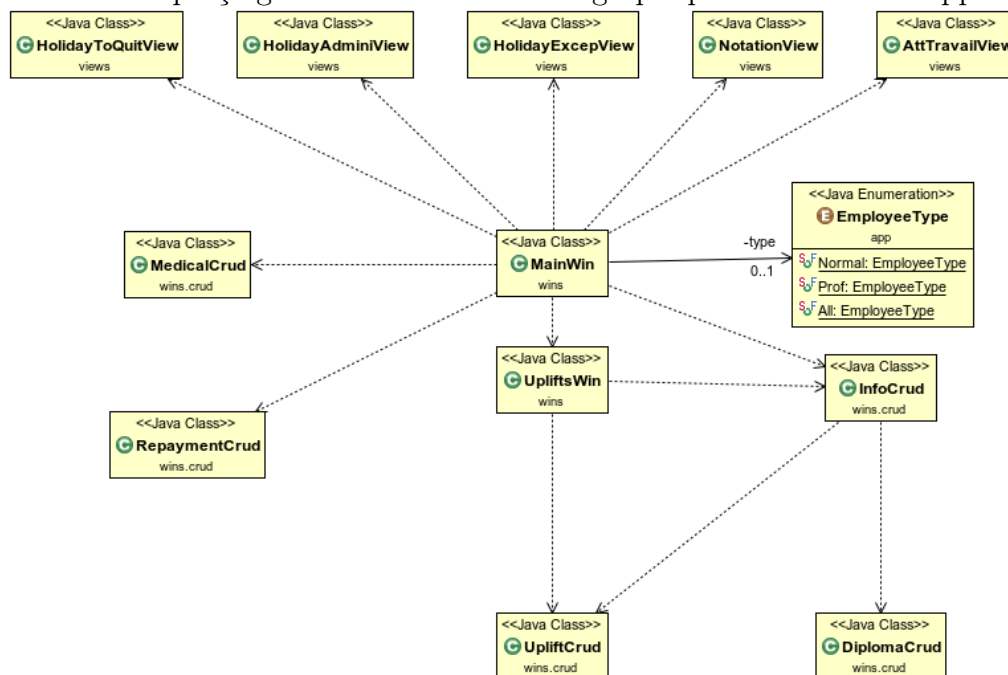
# Quatrième partie

## L'Interface Graphique

## Chapitre 3

# La relation entre les différents Paquets graphiques

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



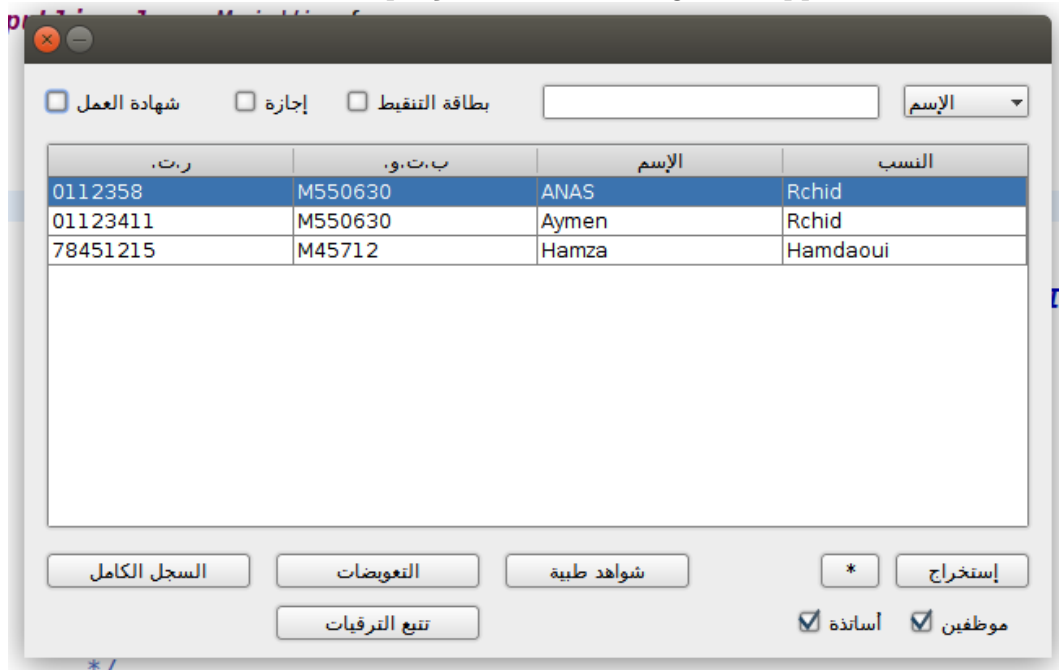
## 3.1 Paquet wins

### 1. La fenêtre principale MainWin

Dans la partie supérieure de la fenêtre, on voit des cases à cocher pour générer l'élément spécifique (voir la section de [views](#)), une zone du texte avec la sélection d'un attribue pour la recherche, c'est un [SearchField](#). Au centre, il y a une table qui montre les informations des employés/fonctionnaires déjà existants.

En bas, il y a des boutons pour afficher le [dossier de chaque personne](#), afficher la fenêtre de [suivi des remboursements](#), afficher suivi les [certificats médicaux](#), et [suivi des avancements de grade](#). En bas à droite, il y a deux boutons, une pour actualiser et une pour générer [la page correspondant](#) aux cases cocher en haut, ainsi que deux autres cases à cocher pour la sélection de type de la personne affichée dans la table.

FIGURE 3.2 – Aperçu sur le démarrage de l'application



### 2. La fenêtre de suivi des avancements de grade UpliftsWin

Dans la figure [3.3](#), on voit que la fenêtre contient deux parties :

**En haut** Les avancements à venir dans la durée sélectionnée avec l'aide de la boîte combo. On peut aussi afficher les informations de la personne avec le bouton à droite.

**En bas** Les avancements que leur temps est venu et en attente d'une confirmation. On peut confirmer par le bouton à droite.

FIGURE 3.3 – 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	الاسم الكامل	ب.ت.و.	ر. التأجير
3	ANAS RCHID	9	M550630	0112358

\*

مصادقة

Après la confirmation d'un avancement, une [fenêtre](#) s'ouvre pour ajouter les informations du avancement.

## 3.2 Paquet wins.crud

Toutes les formes suivantes avaient le même princip : des données du texte (ou combos) qui permet de la modification de employé existant déjà, l'ajout d'un nouveau, ou la suppression d'un ancien.

### 1. Gestion des Employés InfoCrud

C'est la fenêtre qui permet de changer les informations de chaque employé ou fonctionnaire. En haut, on voit la place de l'image, au centre on voit les zones de textes pour la modification des informations divisées en trois sections, les informations personnelles, administrative et autres (qui contient des informations additionnelle comme le nom et prénom en arabe).

Pour l'image, le programme lit une image qui doit existe dans le répertoire data/imgs et contient le même CIN de l'employé concerné.

FIGURE 3.4 – Aperçu sur la fiche des informations personnelle et administrative

البطاقة الشخصية و الوضعية الإدارية

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

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: 99 Ibn Sina, OD Frej El عدد الأطفال: 0

الشهادات

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

حذف \* جديد حفظ

## 2. Gestion des Remboursements



السبب	عدد الأيام	المعوضة	الباقى
Semestre I	5	4	1
Semestre II	60	50	10

54 مجموع الأيام المعوضة

حذف تعديل إضافة

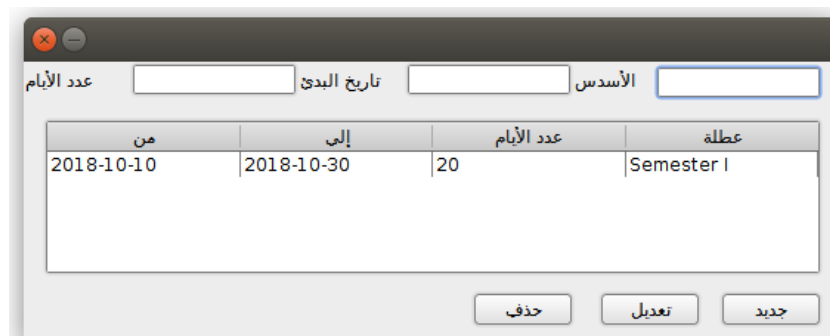
## 3. Gestion des Diplômes



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

حذف تعديل إضافة

## 4. Gestion des Certificats Médicaux



من	إلى	عدد الأيام	عطلة
2018-10-10	2018-10-30	20	Semester I

حذف تعديل جديد

## 5. Gestion des Grades



التاريخ	الرقم الاستدلالي	الرتبة	السلم
2017-06-12	12121	2	9
2017-06-15	451245	1	9

حذف تعديل إضافة

### 3.3 Paquet views

Ce paquet est initialisé par les informations de l'employé, voici un exemple :

Royaume du Maroc

Université Chouaib Doukkali

Faculté des Lettres et des Sciences

Humaines - El Jadida



المملكة المغربية

جامعة شبيب دوكالي

كلية الآداب والعلوم الإنسانية

الجبدة

---

#### **Attestation De Travail**

Le Doyen de la faculté des Lettres et des Sciences Humaines  
atteste que Mr:

Nom complet: Rchid ANAS

Nationalité: Marocaine

Grade: 9

S.O.M.: 0112358

Exerce ses fonctions dans cet établissement  
depuis le 15/10/2018 au département Informatique

La présente attestation est délivrée à l'intéressé(e) pour servir  
et valoir ce que de droit

El Jadida le: 17/06/2018



## Cinquième partie

### Conclusion

*Ce projet a été sous plusieurs aspects riches d'enseignements. Le projet consistait à réaliser une application permettant la gestion des carrières des ressources humaines. C'était une opportunité pour améliorer mes connaissances au matière de codage **Java**.*

*En conclusion, mon projet ma permetais de mettre en oeuvre mes competances scolaires, professionnelles et humaines pour un sujet intéressant. J'ai acquis des nouvelles compétences dans le domain de développement*

# Sixième partie

## Dépendances

**JDOM** <http://jdom.org/> Bibliothèque Java pour analyser XML

**WebLaF** <http://weblookandfeel.com/> Bibliothèque Java pour améliorer l'apparence