VINCI THERMO GREEN
Réalisation

PAGE DE SERVICE

Référence : Vinci Thermo Green
Plan de classement : stadium-technic-analyse-conception-thermo-green
Niveau de confidentialité : confidential

Mises à jour

Version	Date	Auteur	Description du changement
1.0.0	12-04-2016	Jérôme VALENTI	création Networking Inc.
1.1.0	01-10-2022	Jérôme VALENTI	maj
2.0.0	05-11-2023	Lucas Chabanne	maj
3.0.0	04-01-2024	Lucas Chabanne	maj

Validation

Version	Date	Nom	Rôle
1.0.0	18-04-2016	Delphine TALARON	Direction Technique Vinci Thermo Green Project

Diffusion

Version	Date	Nom	Rôle
1.0.0	20-04-2016	All	SLAM Networking Inc.
1.1.0	01-10-2022	All	SLAM Networking Inc.
2.0.0	02-11-2023	Lucas Chabanne	SLAM Networking Inc.
3.0.0	04-01-2024	Lucas Chabanne	SLAM Networking Inc.

OBJET DU DOCUMENT

Ce document décrit l'implémentation Java du projet Vinci Thermo Green.
Ce projet vise à produire une application réalise l'implémentation du diagramme des classes métier et du diagramme de séquence objet présenté dans la documentation d'analyse conception.

Sommaire

1 ARCHITECTURE

Conformément à l'analyse conception, la réalisation de l'application est structurée en trois couches selon une structure qui ressemble à un design-pattern MVC (Model View Controler) mais qui en réalité ne l'est pas vraiment. Cependant cette structure du code permet d'envisager dans une version ultérieure une évolution vers un modèle réellement MVC.

Le modèle MVC fonctionne selon le principe illustré par le schéma ci-dessous[1] :

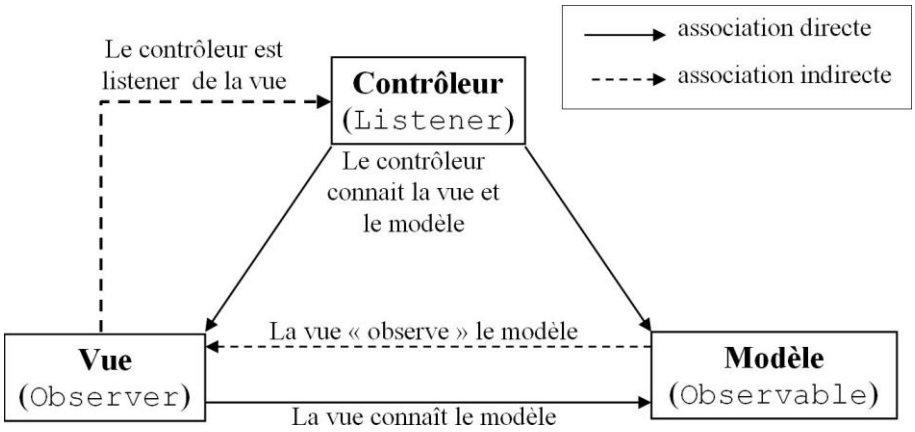


Figure 1. Principe de fonctionnement du modèle MVC

La conception de la v.2.0.0 de l'application Vinci Thermo Green prévoit la mise en œuvre d'un contrôleur. Ce contrôleur représente non pas le "listener" au sens MVC du terme mais le Data Access Object (DAO)[2] d'une architecture n-tiers.
Cependant, l'utilisation de la bibliothèque graphique Swing permet d'écouter la vue au sens propre du design-pattern MVC. Cela pourra être abordé lors d'une version ultérieure.

L'utilisation d'un DAO permet de s'abstraire de la façon dont les données sont stockées au niveau des objets métier. Ainsi, le changement du mode de stockage ne remet pas en cause le reste de l'application. Seules les classes dites "techniques" seront à modifier.

Les objets en mémoire vive sont liés à des données persistantes (stockées en base de données, dans des fichiers, dans des annuaires, etc…). Le modèle DAO regroupe les accès aux données persistantes dans des classes techniques spécifiques, plutôt que de les disperser. Il s'agit surtout de ne pas écrire ces accès dans les classes "métier", qui ne seront modifiées que si les règles de gestion métier changent.

Ce modèle complète le modèle MVC (Modèle - Vue - Contrôleur), qui préconise de séparer dans des classes les différentes problématiques :

- des "vues" (charte graphique, ergonomie)
- du "modèle" (cœur du métier)
- des "contrôleurs" (tout le reste : l'enchaînement des vues, les autorisations d'accès, etc…)

[1] http://www.infres.enst.fr/~hudry/coursJava/interSwing/boutons5.html
[2] https://fr.wikipedia.org/wiki/Objet_d'acc%C3%A8s_aux_donn%C3%A9es

1.1 ARCHITECTURE DE L'APPLICATION

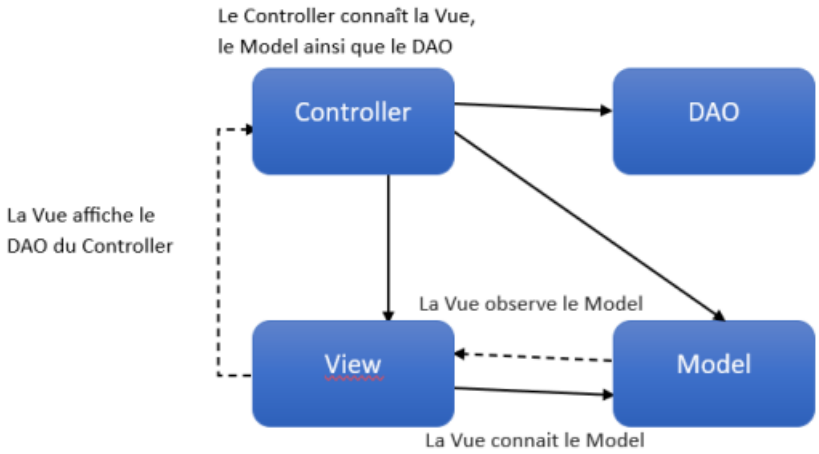


Figure 2. Architecture MVC de l'application

Le Controller instancie des objets liés à la Vue, au Model et au DAO.

Tout d'abord, il importe les packages DAO, model, et view, ce qui indique que le Controller connaît les classes de ces différents éléments. Ensuite, dans le constructeur du Controller, on peut voir qu'il crée une instance de DAOmySQL, ce qui implique que le Controller a une connaissance directe du DAO. En utilisant cette instance de DAO, il lit une base de données et extrait un ensemble d'utilisateurs, créant des objets User pour chacun d'eux. Cela montre que le Controller a une connaissance du Model, en l'occurrence de la classe User.

```
import DAO.*
import model.*;
import view.*;
import control.DAOmySQL;
```

Le Controller initialise également des objets de type Login et ConsoleGUI, indiquant qu'il a accès à la Vue. Le fait qu'il puisse créer ces objets implique une familiarité avec la façon dont ils sont conçus et manipulés. Ainsi, le Controller démontre une connaissance directe de la Vue, du Model et du DAO, car il utilise activement des éléments de ces trois parties pour coordonner le fonctionnement global de l'application.

```
public Controller() throws ParseException, SQLException {
      this.theDAOMesures = new FileMesures(this);
      this.myDAOmySQL = new DAOmySQL(this);
      theDAOMesures.readDB();

      this.userSet = myDAOmySQL.listUsers();

      while(userSet.next()) {
            this.theUser        =        new        User(userSet.getString("login"),
userSet.getString("password"));
            this.lesUsers.add(theUser);
      }

      this.theLogin = new Login(this);
      this.theLogin.setBounds(100, 100, 450, 300);
      this.theLogin.setVisible(true);

      this.theConsole = new ConsoleGUI(this);
}
```

La Vue a connaissance du Model et affiche certaines informations concernant le DAO du Controller.

Tout d'abord, l'importation de la classe Controller dans la Vue indique une certaine dépendance de la Vue vis-à-vis du Controller. Ensuite, on peut voir que la méthode listStadiumNames() du DAO est appelée à partir de l'instance

myController.getMyDAOmySQL() dans la Vue. Cela démontre que la Vue a une certaine connaissance de la manière dont le DAO est utilisé dans le Controller.

```
import control.Controller;
import model.Mesure;

ResultSet resultSet = myController.getMyDAOmySQL().listStadiumNames();

while (resultSet.next()) {
      choixZone.addItem(resultSet.getString("name"));
}
```

La Vue manipule activement des objets du Model, en l'occurrence des objets de la classe Mesure. La méthode setTable prend une liste d'objets Mesure en argument, et en fonction de certaines conditions, elle extrait et affiche des informations spécifiques de ces objets, telles que les températures en Celsius ou Fahrenheit, la date et l'heure. Cela montre que la Vue a une connaissance directe des attributs de la classe Mesure.

En somme, la Vue semble avoir une connaissance du Model, comme en témoigne son interaction avec des objets de la classe Mesure, et elle affiche également une certaine connaissance du DAO du Controller en appelant sa méthode listStadiumNames().

```
/**
 * <p>Transfert les donn?es de la collection vers un tableau d'objets</p>
 * <p>La temp?rature est en degr? Fahrenheit</p>
 *
 * @param ArrayList<Mesure>
 * @return Object[][]
 */
public JTable setTable(ArrayList<Mesure> mesures) {

      float min = 0;
      float max = 0;
      float moy = 0;

      DecimalFormat round = new DecimalFormat("0.##");
      Object[][] dataTable = new Object[mesures.size()][3];

      if (rdbtnCelsius.isSelected()) {

            System.out.println("Celsius : " + rdbtnCelsius.isSelected() + " | " +
mesures.size());

            // Initialisation de min et max
            min = mesures.get(0).getCelsius();
            max = mesures.get(0).getCelsius();

            for (int i = 0; i < mesures.size(); i++) {

                  uneMesure = mesures.get(i);
                  dataTable[i][0] = uneMesure.getNumZone();
                  dataTable[i][1] = uneMesure.getHoroDate();
                  dataTable[i][2] = round.format(uneMesure.getCelsius());

                  // Min, max et moy
                  moy = moy + uneMesure.getCelsius();

                  if (uneMesure.getCelsius() < min) {
                        min = uneMesure.getCelsius();
                  }
                  if (uneMesure.getCelsius() > max) {
                        max = uneMesure.getCelsius();
                  }
            }
      } else {
```

```
            System.out.println("Celsius : " + rdbtnCelsius.isSelected() + " | " +
mesures.size());

            // Initialisation de min et max
            min = mesures.get(0).getFahrenheit();
            max = mesures.get(0).getFahrenheit();

            for (int i = 0; i < mesures.size(); i++) {
                  uneMesure = lesMesures.get(i);
                  dataTable[i][0] = uneMesure.getNumZone();
                  dataTable[i][1] = uneMesure.getHoroDate();
                  dataTable[i][2] = round.format(uneMesure.getFahrenheit());

                  // Min, max et moy
                  moy = moy + uneMesure.getFahrenheit();

                  if (uneMesure.getFahrenheit() < min) {
                        min = uneMesure.getFahrenheit();
                  }
                  if (uneMesure.getCelsius() > max) {
                        max = uneMesure.getFahrenheit();
                  }
            }
      }

      String[] titreColonnes = { "Zone", "Date-heure", "T°" };
      JTable uneTable = new JTable(dataTable, titreColonnes);
      // Les donn°es de la JTable ne sont pas modifiables
      uneTable.setEnabled(false);

      // Arroundi et affecte les zones texte min, max et moy
      tempMin.setText(round.format(min));
      tempMax.setText(round.format(max));
      moy = moy / mesures.size();
      tempMoy.setText(round.format(moy));

      return uneTable;
}
```

2 IMPLEMENTATION DES CLASSES METIERS

L'analyse a permis de modéliser les classes métiers selon le diagramme ci-dessous (non documenté, cf. document d'analyse-conception) :

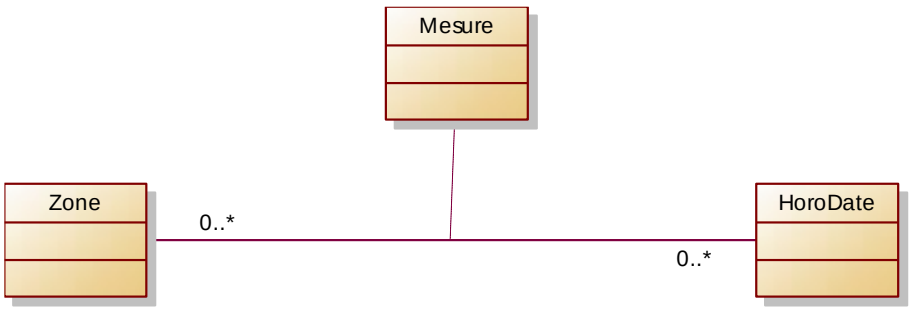


Figure 3. Diagramme des classes métier

L'application lit un fichier texte qui contient les enregistrements de température en degré Fahrenheit[3]. La classe "Mesure" stocke les T°Fahrenheit dans un attribut float et propose une méthode getCelsius() qui retourne la T°Celsius.
L'échelle Fahrenheit est calée sur l'échelle Celsius par la relation :

T(°F) = 1,8 T(°C) + 32

[3] https://fr.wikipedia.org/wiki/Degr%C3%A9_Fahrenheit

T(°C) = (T(°F) -32)/1,8

```
/**
 * @author Jérôme Valenti
 */
package model;

import java.util.Date;

/**
 * <p>Des capteurs mesure régulièrement la température de la pelouse.</p>
 * <p>Pour chaque capteur :</p>
 * <ul>
 * <li>les mesures sont donn&eacute;es en degr&eacute; Fahrenheit;<br /></li>
 * <li>localis&eacute;es par le d&eacute;coupage du terrain en zones;<br /></li>
 * <li>horadat&eacute;ee par la date et l'heure.<br /></li>
 * </ul>
 *
 * @author jvalenti
 * @version 2.0.0
 */
public class Mesure {
	/**
	 * <p>numZone contient le numéro de la zone mesurée</p>
	 */
	private String numZone;
	/**
	 * <p>horoDate contient la date et l'heure de la mesure au format aa-mm-jj
hh:mm</p>
	 */
	private Date horoDate;
	/**
	 * <p>valFahrenheit contient la valeur de la température mesurée en degré
Fahrenheit</p>
	 */
	private float fahrenheit;

	public Mesure() {
		this.numZone = new String();
		this.horoDate = new Date();
		this.fahrenheit = 0.0f;
	}

	public Mesure(String pZone, Date pDate, float pFahrenheit) {

		this.numZone = pZone;
		this.horoDate = pDate;
		this.fahrenheit = pFahrenheit;
	}

	public String getNumZone() {
		return numZone;
	}

	public void setNumZone(String numZone) {
		this.numZone = numZone;
	}

	public Date getHoroDate() {
		return horoDate;
	}

	public void setHoroDate(Date horoDate) {
		this.horoDate = horoDate;
	}
```

```
	public float getFahrenheit() {
		return fahrenheit;
	}

	public void setFahrenheit(float valFahrenheit) {
		this.fahrenheit = valFahrenheit;
	}

	/**
	 * <p>Convertit Fahrenheit en °Celsius</p>
	 * @since 2.0.0
	 * @return float t°Celsius
	 */
	public float getCelsius() {
		//return (float) (valFahrenheit - 32) / 1.8)
		return (fahrenheit - 32.0f) / 1.8f;
	}
}
```

3 ACCES AUX DONNEES

3.1 LIRE UN FICHIER CSV[4]

CSV (Comma Separated Values) est un format de fichiers ouvert qui permet de représenter des données tabulaires sous forme de valeurs séparées par des virgules ou un caractère séparateur.
Ce format ne fait pas l'objet d'une spécification formelle mais la RFC 4180 (Request For Comments) décrit la forme la plus courante et établit son type MIME « text/csv », enregistré auprès de l'IANA.
Un fichier CSV est un fichier texte, par opposition aux formats dits « binaires ». Chaque ligne du texte correspond à une ligne du tableau et les séparateurs correspondent aux séparations entre les colonnes. Les portions de texte séparées par un séparateur correspondent donc aux contenus des cellules du tableau ou au champ d'un enregistrement.

Une ligne est une suite ordonnée de caractères terminée par un caractère de fin de ligne (line break - CRLF), la dernière ligne pouvant en être exemptée.

Dans un premier temps, on lit un fichier texte ligne par ligne.
http://www.ukonline.be/programmation/java/tutoriel/chapitre12/page3.php
https://docs.oracle.com/javase/8/docs/api/

Les classes File, FileReader et BufferedReader fournissent les méthodes pour lire un fichier texte ligne par ligne :

```
[…]
	File f = new File("data\\mesures.csv");
	FileReader fr = new FileReader(f);
	BufferedReader br = new BufferedReader(fr);
[…]
```

3.1.1 MANIPULER UNE CHAINE DE CARACTERES

Pour découper (parser) une chaîne de caractère, on peut entre autres :

- invoquer la méthode split() de la classe String[5]
- ou utiliser la classe Scanner[6]

Avec la méthode Split() :

[4] https://fr.wikipedia.org/wiki/Comma-separated_values
http://thierry-leriche-dessirier.developpez.com/tutoriels/java/charger-donnees-fichier-csv-5-min/
http://thierry-leriche-dessirier.developpez.com/tutoriels/java/csv-avec-java/

[5] https://rachonzedev.wordpress.com/2011/04/21/decouper-une-chaine-de-caracteres-string-en-java/

[6] https://www.tutorielsenfolie.com/tutoriels-103-Parser-une-chaine-de-caracteres-en-Java.html

```
	/**
	 * <p>Lit un fichier de type CSV (Comma Separated Values)</p>
	 * <p>Le fichier contient les mesures de temp&eacute;rature de la pelouse.</p>
	 *
	 * @author Jérôme Valenti
	 * @return
	 * @throws ParseException
	 * @since 2.0.0
	 */
	public void lireCSV(String filePath) throws ParseException {

		try {
			File f = new File(filePath);
			FileReader fr = new FileReader(f);
			BufferedReader br = new BufferedReader(fr);

			try {
			// Chaque ligne est un enregistrement de données
			String records = br.readLine();

			// Chaque enregistrement contient des champs
			String[] fields = null;
			String numZone = null;
			Date horoDate = null;
			float fahrenheit;

			while (records != null) {
				// Affecte les champs de l'enregistrement courant dans un
				// tableau de chaine
				fields = records.split(";");

				// Affecte les champs aux paramètre du constructeur de
				// mesure
				numZone = fields[0];
				horoDate = strToDate(fields[1]);
				fahrenheit = Float.parseFloat(fields[2]);

				// Instancie une Mesure
				Mesure laMesure = new Mesure(numZone, horoDate, fahrenheit);
				lesMesures.add(laMesure);

				// Enregistrement suivant
				records = br.readLine();
				}

			br.close();
			fr.close();
			} catch (IOException exception) {
				System.out.println("Erreur lors de la lecture : " +
exception.getMessage());
			}
		} catch (FileNotFoundException exception) {
			System.out.println("Le fichier n'a pas été trouvé");
		}
	}
```

3.1.2 CONVERTIR UNE CHAINE DE CARACTERE

3.1.2.1 Convertir une chaîne en date heure[7]

La manipulation des dates n'est pas simple à mettre en œuvre :

- Il existe plusieurs calendriers dont le plus usité est le calendrier Grégorien. Le calendrier Grégorien comporte de nombreuses particularités : le nombre de jours

[7] http://www.jmdoudoux.fr/java/dej/chap-utilisation_dates.htm

d'un mois varie selon le mois, le nombre de jours d'une année varie selon l'année (année bissextile), ...

- Le format textuel de restitution des dates diffère selon la Locale utilisée
- L'existence des fuseaux horaires qui donnent une date/heure différente d'un point dans le temps selon la localisation géographique
- La possibilité de prendre en compte un décalage horaire lié aux heures d'été et d'hiver

Pourtant le temps s'écoule de façon linéaire : c'est de cette façon que les calculs de dates sont réalisés avec Java, en utilisant une représentation de la date qui indique le nombre de millisecondes écoulées depuis un point d'origine défini. Dans le cas de Java, ce point d'origine est le 1er janvier 1970. Ceci permet de définir un point dans le temps de façon unique.

Dans l'application, la date-heure est lue sous la forme d'une chaîne de caractère et convertie en Date par une méthode privée ci-dessous qu'on pourrait envisager de la déplacer en public dans la classe Mesure elle-même.

```
        /**
         * <p>Convertion d'une String en Date</p>
         *
         * @param strDate
         * @return Date
         * @throws ParseException
         */
        private Date strToDate(String strDate) throws ParseException {

                SimpleDateFormat leFormat = null;
                Date laDate = new Date();
                leFormat = new SimpleDateFormat("yy-MM-dd kk:mm");

                laDate = leFormat.parse(strDate);
                return laDate;
        }
}
```

3.1.2.2 Convertir en nombre décimal

Les classes Float et Double possèdent une méthode X.parseX(String s) qui permet de convertir une chaîne de caractères en un flottant. Si cette conversion n'est pas possible il y a levée d'une exception NumberFormatException.

```
        //convertir une chaine en float
        float fahrenheit;
        fahrenheit = Float.parseFloat(uneString);
```

Attention, en Java, le caractère séparateur des décimal est le "." Donc inutile de remplacer le "." par la "," dans le fichier :

[…]

```
        for(int i = 0; i < fields.length; i++) {

        //remplace le séparateur décimal par une virgule
        if (i == 2) {
                float a = Float.parseFloat(fields[i]);
                System.out.println(a);
                fields[i] = fields[i].replace(".",",");
                System.out.println("test : " + fields[i]);
                float b = Float.parseFloat(fields[i]);
                System.out.println(b);
        }
        System.out.println("élement n° " + i + "=[" + fields[i]+"]");
        }
```

[…]

3.2 LIRE ET ECRIRE DANS UNE BASE DE DONNEES – JDBC

3.2.1 PRESENTATION DE JDBC

JDBC (Java Database Connectivity) est une API (Application Programming Interface) Java qui permet aux programmes Java d'interagir avec des bases de données. En termes simples, cela signifie qu'il fournit un moyen pour les programmes Java d'envoyer des requêtes à une base de données, d'envoyer des mises à jour aux données de la base de données, et de recevoir les résultats de ces requêtes.

3.2.2 L'AUTHENTIFICATION

La méthode listUsers exécute une requête SQL pour récupérer tous les utilisateurs de la table "user" dans la base de données. Elle retourne un objet ResultSet qui contient les résultats de la requête.

```
public ResultSet listUsers() throws SQLException {
      String theQuery = "SELECT * FROM user;";
      ResultSet theResultSet = myStatement.executeQuery(theQuery);
      return theResultSet;
}
```

Dans la classe Controller, une ArrayList appelée lesUsers est créée pour stocker des objets de type User. Les utilisateurs sont extraits d'un objet ResultSet et ajoutés à cette liste.

```
private ArrayList<User> lesUsers = new ArrayList<User>();

while(userSet.next()) {
      this.theUser = new User(userSet.getString("login"),
userSet.getString("password"));
      this.lesUsers.add(theUser);
}
```

Dans la classe Login, l'événement "MouseClicked" du bouton de connexion (btnLogin) est géré. Les informations de connexion fournies par l'utilisateur sont comparées avec les données stockées dans l'ArrayList lesUsers. Si une correspondance est trouvée, la fenêtre de connexion est masquée et la console d'application est affichée.

```
this.btnLogin.addMouseListener(new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent e) {
            for (User user : lesUsers) {
                  if          (user.getLogin().equals(txtLogin.getText())          &&
BCrypt.checkpw(txtPassword.getText(), user.getPassword())) {
                        myController.getTheLogin().setVisible(false);
                        myController.getTheConsole().setVisible(true);
                        break;
                  }
            }
      }
});
```

3.2.3 La liste deroulante

La fonction listStadiumNames de la classe DAOmySQL permet de récupérer les noms des stades présents dans la base de données. Elle exécute une requête SQL "SELECT name FROM stadium;" et renvoie un ensemble de résultats ResultSet contenant les noms des stades.

```
public ResultSet listStadiumNames() throws SQLException {
      String theQuery = "SELECT name FROM stadium;";
      ResultSet theResultSet = myStatement.executeQuery(theQuery);
      return theResultSet;
}
```

L'implémentation pour ajouter les valeurs de la base de données dans une liste déroulante dans la classe ConsoleGUI se realise comme ceci au niveau du code :

```
while (resultSet.next()) {
      choixZone.addItem(resultSet.getString("name"));
}
```

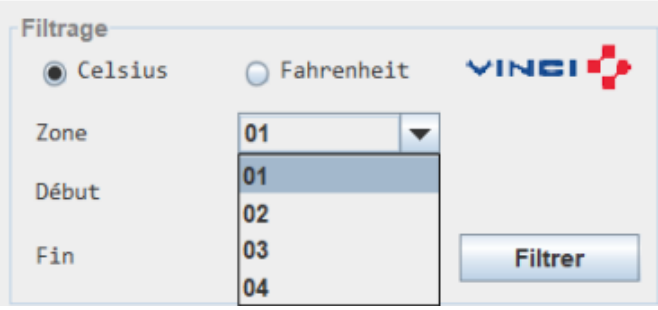


Figure 4. Résultat de la liste déroulante

3.2.4 Le tableau des mesures

De même, la classe DAOmySQL offre la fonction listMeasures, qui exécute une requête SQL "SELECT * FROM measures;" pour récupérer l'ensemble des mesures de la base de données. Cette fonction retourne un objet ResultSet contenant toutes les mesures.

```
public ResultSet listMeasures() throws SQLException {
      String theQuery = "SELECT * FROM measures;";
      ResultSet theResultSet = myStatement.executeQuery(theQuery);
      return theResultSet;
}
```

La méthode readDB de la classe FileMesures remplace la fonction lireCSV. Cette méthode récupère les résultats de la fonction listMeasures de la classe DAOmySQL, puis parcourt les lignes de résultats pour créer des objets Mesure à partir des données extraites de la base de données. Les champs utilisés pour chaque instance de Mesure sont le numéro de zone, la date et la température, extraits des colonnes correspondantes de la base de données.

```
public void readDB() throws ParseException, SQLException {

      ResultSet resultSet = myController.getMyDAOmySQL().listMeasures();

      String numZone = null;
      Date horoDate = null;
      float fahrenheit;

      while (resultSet.next()) {

            // Affecte les champs aux param�tre du constructeur de
            // mesure
            numZone = String.valueOf("0" + resultSet.getInt("zone"));
            horoDate = strToDate(resultSet.getString("date_measure"));
            fahrenheit = resultSet.getFloat("temperature");
            // Instancie une Mesure
```

```
            Mesure laMesure = new Mesure(numZone, horoDate, fahrenheit);
            myController.getLesMesures().add(laMesure);
      }
}
```

4 GERER UNE COLLECTION D'OBJET[8]

Les collections sont des objets qui permettent de gérer des ensembles d'objets. Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, etc. ...

Une collection est un regroupement d'objets qui sont désignés sous le nom d'éléments.

L'API Collections propose un ensemble d'interfaces et de classes dont le but est de stocker de multiples objets. Elle propose quatre grandes familles de collections, chacune définie par une interface de base :

- List : collection d'éléments ordonnés qui accepte les doublons
- Set : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons
- Map : collection sous la forme d'une association de paires clé/valeur
- Queue et Deque : collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement

Dans l'application Thermo Green, on utilise une collection pour les objets "Mesure" instanciés à partir des valeurs lues dans le fichiers des mesures qui est mis à jour par les capteurs de température.

[…]

```
while (records != null)
      {
      //Affecte les champs de l'enregistrement courant dans un tableau de chaine
      fields = records.split(";");

      //Affecte les champs aux paramètres du constructeur de "Mesure"
      numZone = fields[0];
      horoDate = strToDate(fields[1]);
      fahrenheit = Float.parseFloat(fields[2]);

      //Instancie une Mesure
      Mesure laMesure = new Mesure(numZone, horoDate,fahrenheit);
      uneListeMesures.add(laMesure);

      records = br.readLine();
      }
```

[…]

Pour filtrer la collection, on ne modifie pas la collection en supprimant les objets hors critère et ceci pour au moins deux raisons :

Techniquement, modifier une collection pendant son parcours nécessite des précautions sinon on obtient ce genre d'erreur :

```
      public void filtrerLesMesure(String laZone) {
            // Parcours de la collection et suppression des objets qui ne
            // correspondent pas aux paramètres
            for (Mesure mesure : lesMesures) {
                  if (laZone != null) {
                        if (mesure.getNumZone() != laZone) {
                              lesMesures.remove(mesure);
                        }
```

[8] https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-collections-d-objets
http://www.jmdoudoux.fr/java/dej/chap-collections.htm

```
                }
            }
        }
```

```
Exception in thread "AWT-EventQueue-0" java.util.ConcurrentModificationException
```

Pour cause, les index ne sont plus cohérents entre l'iterator et la collection elle-même.

Fonctionnellement, les éléments que l'on supprime lors d'un premier filtrage peuvent être attendus lors d'un autre filtrage.

```
    /**
     * <p>
     * Filtre la collection des mesures en fonction des param&egrave;tres :
     * </p>
     * <ol>
     * <li>la zone (null = toutes les zones)</li>
     * <li>la date de d&eacute;but (null = &agrave; partir de l'origine)</li>
     * <li>la date de fin (null = jusqu'&agrave; la fin)<br />
     * </li>
     * </ol>
     */
    // public void filtrerLesMesure(String laZone, Date leDebut, Date lafin) {
    public ArrayList<Mesure> filtrerLesMesure(String laZone) {
        // Parcours de la collection
        // Ajout à laSelection des objets qui correspondent aux paramètres
        // Envoi de la collection
        ArrayList<Mesure> laSelection = new ArrayList<Mesure>();
        for (Mesure mesure : lesMesures) {
            if (laZone.compareTo("*") == 0) {
                laSelection.add(mesure);
            } else {
                if (laZone.compareTo(mesure.getNumZone()) == 0) {
                    laSelection.add(mesure);
                }
            }
        }
        return laSelection;
    }
```

nota bene : la comparaison entre deux chaînes de caractère à ne pas confondre avec la comparaison entre deux pointeurs sur deux chaines.[9]
La classe String implémente l'interface java.lang.Comparable, et possède donc une méthode "compareTo(String s)" renvoyant :

- un nombre négatif si la chaîne actuelle est placée avant la chaîne passée en paramètre;
- zéro (0) si les deux chaînes sont strictement égales;
- un nombre positif si la chaîne actuelle est placée après la chaîne passée en paramètre.

```
int comparaison = "Hello".compareTo("World");
System.out.println(comparaison); /* Nombre négatif car H < W */
```

4.1 Recherche du min et du max[10] et calcul de la moyenne

On pourrait rechercher le min et le max en implémentant une collection.

Le plus simple dans cette version consiste à repérer min et max à la constitution des données de la JTable :

---

[9] http://blog.lecacheur.com/2006/04/01/stringequals-ou-stringcompareto/
http://thecodersbreakfast.net/index.php?post/2008/02/22/24-comparaison-des-chaines-accentuees-en-java
http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/ChainesDeCaracteres.html

[10] https://docs.oracle.com/javase/8/docs/api/

```
	/**
	 * <p>Transfert les données de la collection vers un tableau d'objets</p>
	 * <p>La température est en degré Fahrenheit</p>
	 *
	 * @param ArrayList<Mesure>
	 * @return Object[][]
	 */
	private static JTable setTable(ArrayList<Mesure> mesures) {

		float min = 0;
		float max = 0;
		float moy = 0;
		DecimalFormat round = new DecimalFormat("0.##");
		Object[][] dataTable = new Object[mesures.size()][3];

		if (rdbtnCelsius.isSelected()) {

			System.out.println("Celsius : " + rdbtnCelsius.isSelected() + " | "
+ mesures.size());

			// Initialisation de min et max
			min = mesures.get(0).getCelsius();
			max = mesures.get(0).getCelsius();

			for (int i = 0; i < mesures.size(); i++) {

				uneMesure = lesMesures.get(i);
				dataTable[i][0] = uneMesure.getNumZone();
				dataTable[i][1] = uneMesure.getHoroDate();
				dataTable[i][2] = round.format(uneMesure.getCelsius());

				// Min, max et moy
				moy = moy + uneMesure.getCelsius();

				if (uneMesure.getCelsius() < min) {
					min = uneMesure.getCelsius();
				}
				if (uneMesure.getCelsius() > max) {
					max = uneMesure.getCelsius();
				}
			}
		} else {

			System.out.println("Celsius : " + rdbtnCelsius.isSelected() + " | "
+ mesures.size());

			// Initialisation de min et max
			min = mesures.get(0).getFahrenheit();
			max = mesures.get(0).getFahrenheit();

			for (int i = 0; i < mesures.size(); i++) {
				uneMesure = lesMesures.get(i);
				dataTable[i][0] = uneMesure.getNumZone();
				dataTable[i][1] = uneMesure.getHoroDate();
				dataTable[i][2] = round.format(uneMesure.getFahrenheit());

				// Min, max et moy
				moy = moy + uneMesure.getFahrenheit();

				if (uneMesure.getFahrenheit() < min) {
					min = uneMesure.getFahrenheit();
				}
				if (uneMesure.getCelsius() > max) {
					max = uneMesure.getFahrenheit();
				}
```

```
                }
            }

            String[] titreColonnes = { "Zone", "Date-heure", "T°" };
            JTable uneTable = new JTable(dataTable, titreColonnes);
            // Les données de la JTable ne sont pas modifiables
            uneTable.setEnabled(false);

            // Arroundi et affecte les zones texte min, max et moy
            tempMin.setText(round.format(min));
            tempMax.setText(round.format(max));
            moy = moy / mesures.size();
            tempMoy.setText(round.format(moy));

            return uneTable;
    }
```

La moyenne est calculée à l'affichage.

5 AFFICHER DES DONNEES

5.1 SOUS FORME TABULAIRE[11]

Le composant JTable permet d'afficher des tables de données, en autorisant éventuellement l'édition de ces données. Un JTable ne contient pas ses données mais les obtient à partir d'un tableau d'objets à 2 dimensions, ou à partir d'un modèle de données. Le rendu et le mode d'édition des cellules de la table peuvent être modifiés.

Le composant JTable est un «visualisateur» qui prend les données à afficher dans un modèle qui implémente l'interface TableModel, ou qui dérive de la classe abstraite AbstractTableModel (javax.swing.table.AbstractTableModel). La classe AbstractTableModel implémente les méthodes de TableModel sauf :

- `public int getRowCount()` : le nombre de lignes.
- `public int getColumnCount()` : le nombre de colonnes.
- `public Object getValueAt(intligne, int colonne)` : l'objet à l'intersection d'une ligne et d'une colonne.

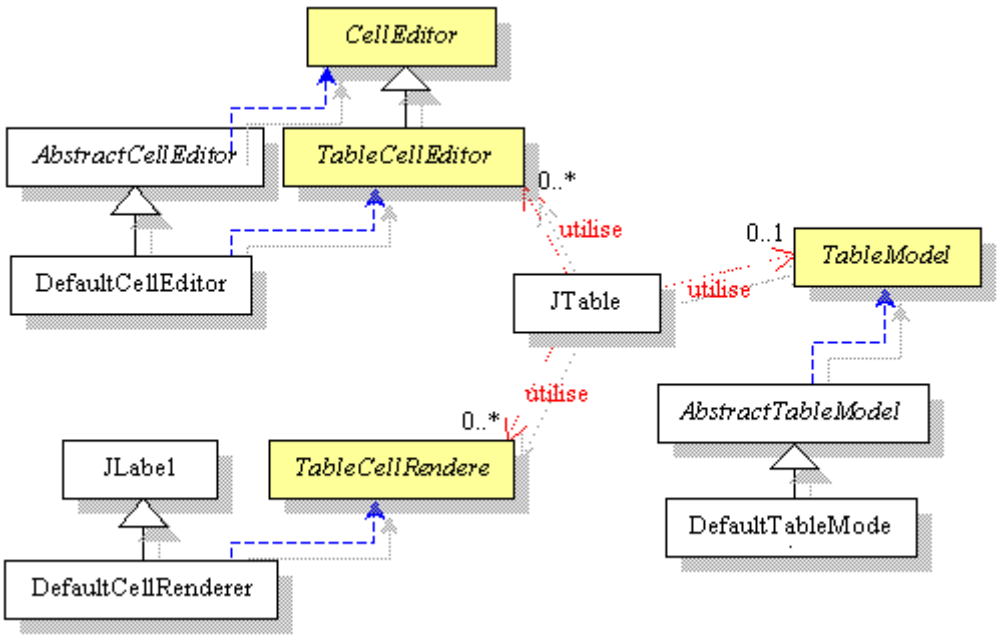


Figure 5. diagramme des classes associées au JTable

[11] http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/swing/JTable.html
https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-interfaces-de-tableaux
http://docs.oracle.com/javase/tutorial/uiswing/components/table.html

```
/**
 * <p>Transfert les données de la collection vers un tableau d'objets</p>
 * <p>La température est en degré Fahrenheit</p>
 *
 * @param ArrayList<Mesure>
 * @return Object[][]
 */
private static JTable setTable(ArrayList<Mesure> mesures) {

	float min = 0;
	float max = 0;
	float moy = 0;
	DecimalFormat round = new DecimalFormat("0.##");
	Object[][] dataTable = new Object[mesures.size()][3];

	if (rdbtnCelsius.isSelected()) {

		// Initialisation de min et max
		min = mesures.get(0).getCelsius();
		max = mesures.get(0).getCelsius();

		for (int i = 0; i < mesures.size(); i++) {

			uneMesure = lesMesures.get(i);
			dataTable[i][0] = uneMesure.getNumZone();
			dataTable[i][1] = uneMesure.getHoroDate();
			dataTable[i][2] = round.format(uneMesure.getCelsius());

			// Min, max et moy
			moy = moy + uneMesure.getCelsius();

			if (uneMesure.getCelsius() < min) {
				min = uneMesure.getCelsius();
			}
			if (uneMesure.getCelsius() > max) {
				max = uneMesure.getCelsius();
			}
		}
	} else {

		// Initialisation de min et max
		min = mesures.get(0).getFahrenheit();
		max = mesures.get(0).getFahrenheit();

		for (int i = 0; i < mesures.size(); i++) {
			uneMesure = lesMesures.get(i);
			dataTable[i][0] = uneMesure.getNumZone();
			dataTable[i][1] = uneMesure.getHoroDate();
			dataTable[i][2] = round.format(uneMesure.getFahrenheit());

			// Min, max et moy
			moy = moy + uneMesure.getFahrenheit();

			if (uneMesure.getFahrenheit() < min) {
				min = uneMesure.getFahrenheit();
			}
			if (uneMesure.getCelsius() > max) {
				max = uneMesure.getFahrenheit();
			}
		}
	}

	String[] titreColonnes = { "Zone", "Date-heure", "T°" };
	JTable uneTable = new JTable(dataTable, titreColonnes);
	// Les données de la JTable ne sont pas modifiables
```

```
            uneTable.setEnabled(false);

            // Arroundi et affecte les zones texte min, max et moy
            tempMin.setText(round.format(min));
            tempMax.setText(round.format(max));
            moy = moy / mesures.size();
            tempMoy.setText(round.format(moy));

            return uneTable;
    }
```

5.2 SOUS FORME GRAPHIQUE[12]

JFreeChart est une bibliothèque open source qui permet d'afficher des données numériques sous la forme de graphiques. Elle possède plusieurs formats dont le camembert, les histogrammes ou les lignes et propose de nombreuses options de configuration pour personnaliser le rendu des graphiques. Elle peut s'utiliser dans des applications standalones ou des applications web et permet également d'exporter le graphique sous la forme d'une image.

Les données utilisées dans le graphique sont encapsulées dans un objet de type Dataset. Il existe plusieurs sous-types de cette classe en fonction du type de graphique souhaité.
Un objet de type JFreechart encapsule le graphique. Une instance d'un tel objet est obtenue en utilisant une des méthodes de la classe ChartFactory.

La bibliothèque JFreeChart ne fait pas partie du JDK et doit donc être installée dans l'environnement de développement : l'EDI Eclipse par exemple (cf.: Gilbert, David - Installation Guide).

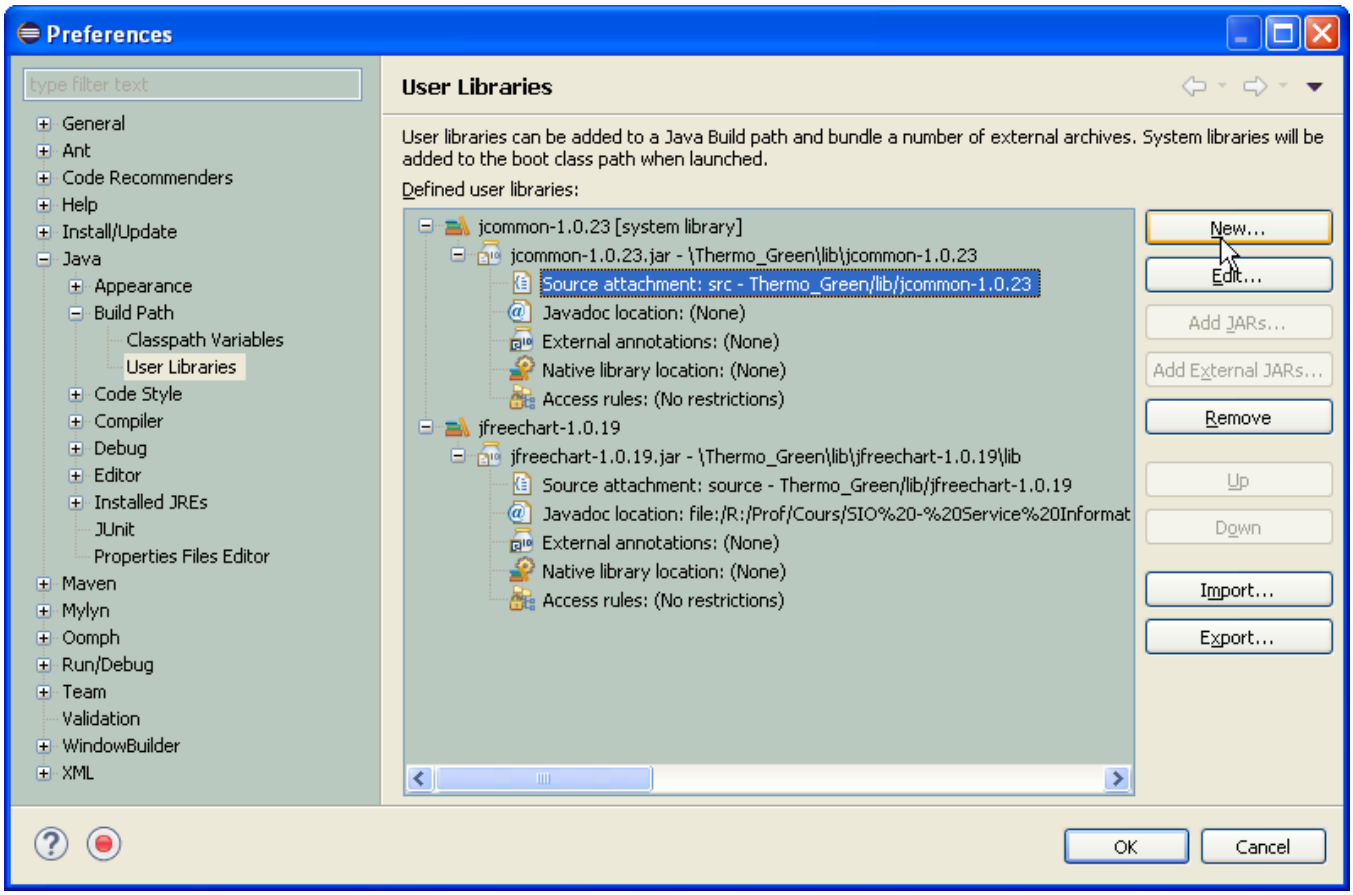


Figure 6. installation de JFreeChart dans Eclipse

[12] http://thierry-leriche-dessirier.developpez.com/tutoriels/java/afficher-graphe-jfreechart-5-min/
http://www.jmdoudoux.fr/java/dej/chap-bibliotheques-free.htm
http://www.jfree.org/jfreechart/
http://www.renaudguezennec.eu/programmation,3-12.html
http://lrie.efrei.fr/2009/11/jfreechart-creer-des-graphes-et-diagrammes-en-java/
http://codes-sources.commentcamarche.net/source/51950-creer-des-graphiques-utilisation-de-jfreechart
http://www.jfree.org/jfreechart/api/javadoc/index.html

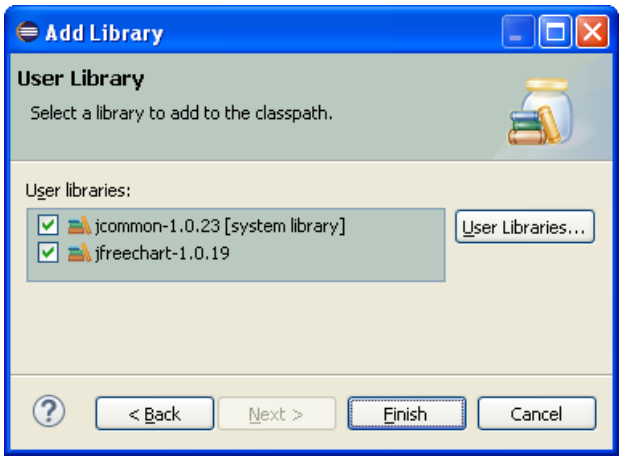


Figure 7. Ajout de JFreeChart au projet

Dans l'application, on alimente un jeu de données à partir de la collection d'objets "Mesure".

```
DefaultCategoryDataset dataChart = new DefaultCategoryDataset();
[…]
dataChart.addValue((Number)uneMesure.getValFahrenheit(), uneMesure.getNumZone(), i1);
```

Ici "i1" est un compteur qui simplifie la représentation de la date. Ceci sera amélioré et remplacé ultérieurement.
On fabrique le graphique à partir des données. Puis un container spécifique qu'on place dans un JPanel lui-même contenu par "pane" le contentPane de la frame :

```
      /**
       * <p>Impl&eacute;mente la biblioth&egrave;que JFreeChart :</p>
       * <ol>
       * <li>d&eacute;finit le type de container de donn&eacute;es -&gt;
DefaultCategoryDataset</li>
       * <li>alimente le container des donn&eacute;es</li>
       * <li>Fabrique un graphique lin&eacute;aire -&gt;
ChartFactory.createLineChart</li>
       * <li>Englobe le graphique dans un panel sp&eacute;cifique -&gt; new
ChartPanel(chart)</li>
       * <li>Englobe ce panel dans un JPanle de l'IHM -&gt;
pnlGraph.add(chartPanel)<br /></li>
       * </ol>
       * @author Jérôme Valenti
       * @see JFreeChart
       */
      public void setChart (){

            int i1 = 0,i2 = 0,i3 = 0,i4 = 0;
            DefaultCategoryDataset dataChart = new DefaultCategoryDataset();

            // Set data ((Number)temp,zone,dateHeure)
            for (int i = 0; i < lesMesures.size(); i++) {

                  uneMesure = lesMesures.get(i);

                  switch (uneMesure.getNumZone()) {
                  case "01":

      dataChart.addValue((Number)uneMesure.getCelsius(),uneMesure.getNumZone(),i1);
                        i1++;
                        break;
                  case "02":
```

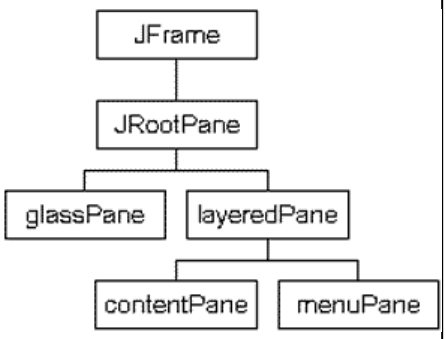
```
                dataChart.addValue((Number)uneMesure.getCelsius(),uneMesure.getNumZone(),i2);
                        i2++;
                        break;
                    case "03":

                dataChart.addValue((Number)uneMesure.getCelsius(),uneMesure.getNumZone(),i3);
                        i3++;
                        break;
                    case "04":

                dataChart.addValue((Number)uneMesure.getCelsius(),uneMesure.getNumZone(),i4);
                        i4++;
                        break;
                    default:
                        break;
                    }
            }

            JFreeChart chart = ChartFactory.createLineChart(
                null,               // chart title
                "Heure",            // domain axis label
                "Températures",     // range axis label
                dataChart,          // data
                PlotOrientation.VERTICAL, // orientation
                true,               // include legend
                true,               // tooltips
                false               // urls
            );
          ChartPanel chartPanel = new ChartPanel(chart);
          chartPanel.setBounds(5, 20, 320, 190);
          chartPanel.setVisible(true);
          pnlGraph.add(chartPanel);
        }
```

6 L'Interface Homme Machine

6.1 Gerer plusieurs JPanel dans une JFrame[13]

Une application graphique doit avoir un composant top-level comme composant racine, c'est-à-dire un composant qui inclut tous les autres composants.
Dans Swing, il y a 3 composants top-level:

1. JFrame,
2. JDialog,
3. JApplet.



Les composants top-level possèdent un content pane qui contient tous les composants visibles d'un top-level. Un composant top-level peut aussi contenir une barre de menu

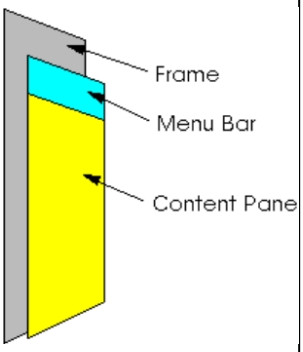
Une JFrame est une fenêtre avec un titre et une bordure.
Chaque JFrame de Swing possède une "vitre", un container racine dans lequel on peut poser tous les autres composants.
Tous les composants associés à un objet JFrame sont gérés par un objet de la classe JRootPane. Un objet JRootPane contient plusieurs Panes. Tous les composants ajoutés au JFrame doivent être ajoutés à un des Pane du JRootPane et non au JFrame directement. C'est aussi à un de ces Panes qu'il faut associer un layout manager si nécessaire. Le Layout manager par défaut du contentPane est BorderLayout.

Le JRootPane se compose de plusieurs éléments :

[13] http://icps.u-strasbg.fr/~bastoul/teaching/java/docs/Swing.pdf
http://www.jmdoudoux.fr/java/dej/chap-swing.htm
http://codes-sources.commentcamarche.net/faq/360-swinguez-jframe-jpanel-jcomponent-layoutmanager-borderlayout

- glassPane : par défaut, le glassPane est un JPanel transparent qui se situe au-dessus du layeredPane. Le glassPane peut être n'importe quel composant : pour le modifier il faut utiliser la méthode setGlassPane() en fournissant le composant en paramètre.
- layeredPane qui se compose du contentPane (un JPanel par défaut) et du menuBar (un objet de type JMenuBar). Le contentPane est par défaut un JPanel opaque dont le gestionnaire de présentation est un BorderLayout. Ce panel peut être remplacé par n'importe quel composant grâce à la méthode setContentPane().

```
Container pane = monIHM.getContentPane();
```

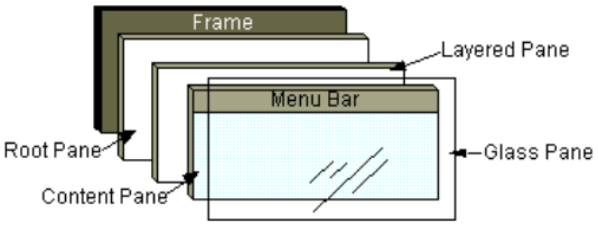


Les conteneurs intermédiaires sont utilisés pour structurer l'application graphique.
Le composant top-level contient des composants conteneur intermédiaires.

Un conteneur intermédiaire peut contenir d'autres conteneurs intermédiaires
Swing propose plusieurs conteneurs intermédiaires :

- JPanel le conteneur intermédiaire le plus neutre.
- JScrollPane offre des ascenseurs, il permet de visionner un composant plus grand que lui.
- JSplitPane est un panel coupé en deux par une barre de séparation.
- JTabbedPane permet d'avoir des onglets.
- JToolBar est une barre d'icones.
- etc…

Swing offre également des conteneurs Intermédiaires spécialisés qui offrent des propriétés particulières aux composants qu'ils accueillent :



- JRootPane est obtenu à partir d'un top-level et composé de:
  - glass pane
  - layered pane
  - content pane
  - menu bar
- JLayeredPane permet de positionner les composants dans un espace à trois dimensions
- JInternalFrame permet d'afficher des petites fenêtres dans une fenêtre.

Swing fournit des composants atomiques qui sont les éléments d'interaction de l'IHM :

- boutons, CheckBox, Radio
- Combo box
- List, menu
- TextField, TextArea, Label
- FileChooser, ColorChooser,
- etc…

6.1.1 Layout[14]

Pour placer des composants dans un container, Java utilise le "Layout".

Un layout est une entité Java qui place les composants les uns par rapport aux autres.
Le layout réorganise les composants lorsque la taille du container varie.
Il y a plusieurs layouts :

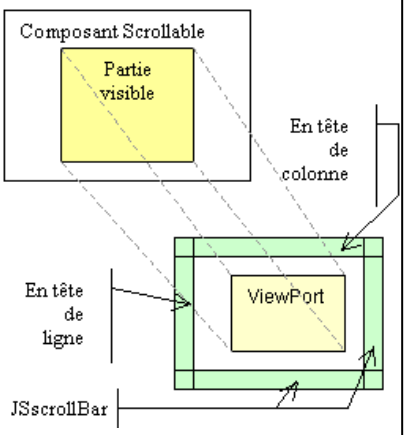
- AbsoluteLayout qui permet de placer les composant par leu coordonnées (x,y).
- BorderLayout sépare un container en cinq zones: NORTH, SOUTH, EAST, WEST et CENTER.
- BoxLayout empiler les composants du container verticalement ou horizontalement.
- CardLayout permet d'avoir plusieurs conteneurs les uns au dessus des autres (comme un jeux de cartes).
- FlowLayout range les composants sur une ligne. Si l'espace est trop petit, une autre ligne est créée. Le FlowLayout est le layout par défaut des JPanel.

[14] http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html

- GridLayout positionne les composants sur une grille.
- GridBagLayout place les composants sur une grille, mais des composants peuvent être contenus dans plusieurs cases. Pour exprimer les propriétés des composants dans la grille, on utilise un GridBagConstraints. Un GridBasConstraints possède :
    - gridx, gridy pour spécifier la position.
    - gridwidth, gridheight pour spécifier la place.
    - fill pour savoir comment se fait le remplissage.

Un layout n'est pas contenu dans un container, il gère le positionnement des composants à l'intérieur du container.

6.2 Le cas particulier du JScroolPane[15]



Dans une interface graphique, il est parfois nécessaire d'afficher des composants plus grands que la fenêtre. Un JScrollPane fournit une vue défilante d'un composant.

Le JScroolPane est un container intermédiaire au même tire qu'un JPanel.

L'application alimente un JTable à partir de la collection des "Mesure" retournée par le contrôleur qui prend en charge l'IHM.

[…]

```
public static void main(String[] args)  throws ParseException {

        //Construit et affiche l'IHM
        ConsoleGUI monIHM = new ConsoleGUI();
        monIHM.setLocation(100,100);

        //Instancie un contrôleur pour prendre en charge l'IHM
        control = new Controller();
        //Demande l'acquisition des data
        lesMesures = control.getLesMesures();

        //Construit le tableau d'objet
        laTable = setTable(lesMesures);

        //Definit le JScrollPane qui va recevoir la JTable
        scrollPane.setViewportView(laTable);

        System.out.println("Before set chart in main()");
        //affiche le graphique
        monIHM.setChart();
        System.out.println("After set chart in main()");
        monIHM.setVisible(true);
}
```

[…]

6.3 Divers composants "atomiques"

L'IHM de l'application est composée de 3 JPanel pour la saisie et un JScrollPane pour l'affichage de la table. Dans chaque JPanel, la saisie est validée par un bouton.
Les calendriers ont été remplacé temporairement par deux simples zones de texte.

[15] https://docs.oracle.com/javase/tutorial/uiswing/components/scrollpane.html

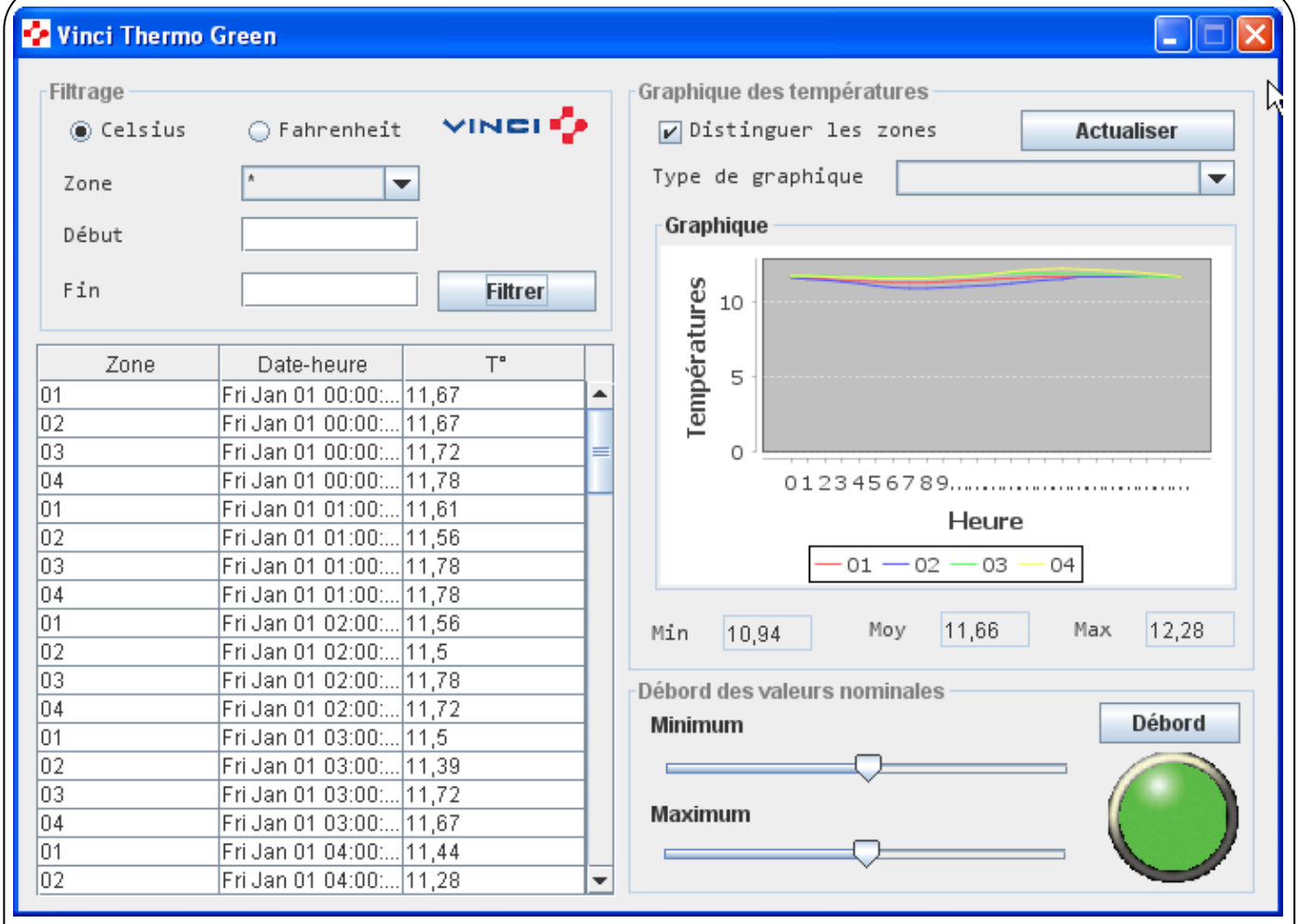


Figure 8. IHM cible du prototype

6.3.1 Les boutons pour valider les saisies[16]

Gérer les interactions avec un JButton impose d'utiliser le modèle Observer-Observable de Swing avec l'implémentation d'un listener ce qui sous-entend également de comprendre ce qu'est une interface au sens Java du terme[17].
La notion d'interface est absolument centrale en Java, et massivement utilisée dans le design des API du JDK (cf. la vidéo de Langlet, Étienne - 10. Java, Interfaces). En Java, une interface est une sorte de classe qui spécifie et "contractualise" un comportement que les classes filles doivent mettre en œuvre. En cela, elles ressemblent aux protocoles réseaux. Les interfaces sont déclarées en utilisant le mot-clé interface et ne peuvent contenir que la signature des méthodes et les déclarations des constantes (variables qui sont déclarées à la fois statique et finale).

Java et Swing offrent plusieurs possibilités pour réaliser une interaction avec un composant, un JButton par exemple. Cette diversité nécessite de comprendre les concepts de classe interne, classe locale (interne de méthode) et classe anonyme[18].

On peut directement à partir de la classe signer un contrat avec une interface en codant par exemple :

---

[16] http://java.developpez.com/faq/gui?page=Les-listeners
http://codes-sources.commentcamarche.net/faq/369-swing-partie-2-actionlistener-listener-jbutton

[17] https://en.wikipedia.org/wiki/Interface_%28Java%29
https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-classes-abstraites-et-les-interfaces
https://fr.wikibooks.org/wiki/Programmation_Java/Interfaces
https://docs.oracle.com/javase/tutorial/java/concepts/interface.html
https://docs.oracle.com/javase/tutorial/java/IandI/usinginterface.html
http://blog.paumard.org/cours/java/chap07-heritage-interface-interface.html

[18] https://fr.wikipedia.org/wiki/Classe_interne
https://fr.wikibooks.org/wiki/Programmation_Java/Classes_internes
http://imss-www.upmf-grenoble.fr/prevert/Prog/Java/CoursJava/classes3.html#locale

```
public class LaClasse extend JFrame implements ActionListener { […] }
```

On peut déclarer une classe interne qui implémentera l'interface ad hoc. Une classe interne a accès aux méthodes et attributs de la classe englobante.
Par exemple :

```
public class TestBouton extends JFrame {

      JPanel panel_1 = new JPanel();
      JPanel panel_2 = new JPanel();

      public TestBouton() {
       getContentPane().setLayout(null);

       panel_1.setBorder(new   TitledBorder(null,   "Commande",   TitledBorder.LEADING,
TitledBorder.TOP, null, null));
       panel_1.setBounds(10, 11, 422, 117);
       getContentPane().add(panel_1);
       panel_1.setLayout(null);

       JButton btnStop = new JButton("Stop");
       btnStop.setBounds(323, 83, 89, 23);
       panel_1.add(btnStop);
       btnStop.addActionListener(new changeColor());

       […]

      }

      class changeColor implements ActionListener {
         public  void  actionPerformed(ActionEvent e)
         {
             panel_2.setBackground(Color.red);
             System.out.println("Stop !");
         }
      }

[…] }
```

Enfin, on peut utiliser des classes anonymes. Par exemple :

```
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;

/**
 * Classe englobante
 */
public class ClasseEnglobante{
        /**
         * Méthode englobant l'appel à une classe anonyme
         */
        public void methodeEnglobante(){

        /**
         * Déclaration et instanciation de la classe anonyme pour un bouton
         * Le bouton est déclaré 'final' afin que la classe anonyme puisse y accéder
         */
                final JButton bouton = new JButton("monBouton");
                bouton.addActionListener(new ActionListener(){
                        public void actionPerformed(ActionEvent e){
                                System.out.println(bouton.toString());
                        }
                });
        }
```

```
}
```

Dans l'application, l'IHM fournit trois JButton d'action qui déclenche chacun un traitement associé. Pour factoriser le code et le structurer, les interfaces sont portées par des classes internes. Chaque classe interne centralise et spécialise l'interaction avec un JButton spécifique.

```
/**
* <p>Classe interne qui gère le clique sur le bouton filtrer
* @author Jérôme Valenti
*/
class filtrerData implements ActionListener {

public void actionPerformed(ActionEvent e){

	lesMesures = control.filtrerLesMesure(choixZone.getSelectedItem().toString());

	//Construit le tableau d'objet
	laTable = setTable(lesMesures);

	//Definit le JScrollPane qui va recevoir la JTable
	scrollPane.setViewportView(laTable);

	//affiche le graphique
	setChart();

	}
}
```

6.3.2 Les boutons radio pour la conversion des températures[19]

L'utilisateur peut choisir l'échelle de degré des températures entre Celsius et Fahrenheit grâce à deux boutons radio disposé dans le JPanel qui regroupe les critères de filtrage.

Déclaration du JPanel et des deux boutons radio :

```
	/**
	 * <p>Container intermédiaire JPanel</p>
	 * <p>Contient les critères de filtrage des données de la table</p>
	 * @see JPanel
	 */
	JPanel pnlCriteria = new JPanel();

	/**
	 * <p>Bouton radio pour le choix de conversion</p>
	 */
	private static JRadioButton rdbtnCelsius = new JRadioButton("Celsius");
	JRadioButton rdbtnFahrenheit = new JRadioButton("Fahrenheit");
```

Dans le constructeur de la classe qui définit l'IHM :

```
	//Ajoute deux boutons radio au JPanel pnlCriteria
	rdbtnCelsius.setFont(new Font("Consolas", Font.PLAIN, 12));
	rdbtnCelsius.setBounds(15, 20, 100, 23);
	pnlCriteria.add(rdbtnCelsius);
	//Sélectionne la convertion celsius par défaut
	rdbtnCelsius.setSelected(true);

	rdbtnFahrenheit.setFont(new Font("Consolas", Font.PLAIN, 12));
	rdbtnFahrenheit.setBounds(115, 20, 100, 23);
	pnlCriteria.add(rdbtnFahrenheit);
```

Dans main(String[] args), on teste quel bouton radio à été activé :

```
	//Test si la conversion est demandée
```

[19] https://docs.oracle.com/javase/tutorial/uiswing/components/button.html#radiobutton

```
	if (monIHM.rdbtnCelsius.isSelected()) {
		data = monIHM.setTableCelsius(lesMesures);
	} else {
		data = monIHM.setTableFahrenheit(lesMesures);
	}
```

6.3.3 Les listes deroulantes[20]

En Swing, il existe 2 sortes de listes déroulantes :

1. JList, liste déroulante qui permet d'afficher et sélectionner plusieurs éléments à la fois.
2. JComboBox, liste de choix.

Il existe 2 manières de manipuler des JComboBox, soit on utilise directement les méthodes de manipulations des éléments de la JComboBox soit on développe son propre modèle de liste.

Dans l'application, on peut faire un "bouchon" pour peupler la liste avec la méthode "addItem". C'est solution rapide mais sale qu'on remplacera par la suite par un peuplement directement à partir des classes métier.

```
		JComboBox<String> choixZone = new JComboBox<String>();
		choixZone.setBounds(115, 50, 100, 20);
	[…]
		pnlCriteria.add(choixZone);

		//un bouchon "Quick & Dirty" pour peupler la liste déroulante
		//TODO peupler la liste avec un équivalent de SELECT DISTINCT
		//TODO implémenter la classe métier Zone pour peupler une JComboBox<Zone>
		choixZone.addItem(null);
		choixZone.addItem("01");
		choixZone.addItem("02");
		choixZone.addItem("03");
		choixZone.addItem("04");
	[…]
```

[20] https://docs.oracle.com/javase/tutorial/uiswing/components/combobox.html
http://baptiste-wicht.developpez.com/tutoriels/java/swing/debutant/?page=listes
https://openclassrooms.com/courses/apprenez-a-programmer-en-java/les-champs-de-formulaire

7 CRYPTOGRAPHIE[21]

7.1 PRESENTATION DE JBCRYPT

JBCrypt est une bibliothèque de hachage de mots de passe populaire utilisée dans de nombreux langages de programmation, y compris Java. Il est utilisé pour sécuriser les mots de passe en les hachant de manière sécurisée, ce qui rend extrêmement difficile pour quiconque, y compris les hackers, de les décoder.

En termes simples, JBCrypt prend un mot de passe comme entrée et le transforme en une chaîne de caractères illisible, appelée hachage, en utilisant un algorithme de hachage sécurisé. Ce hachage peut ensuite être stocké en toute sécurité dans une base de données. Lorsqu'un utilisateur tente de se connecter, JBCrypt compare le hachage du mot de passe entré avec celui stocké dans la base de données pour vérifier si le mot de passe est correct, sans jamais avoir besoin de stocker le mot de passe réel lui-même. Cela aide à protéger les informations sensibles des utilisateurs, telles que les mots de passe, contre les attaques de pirates informatiques.

7.2 HACHAGE AVEC JBCRYPT

Dans le contexte de la gestion de la sécurité des données et des mots de passe des utilisateurs, la cryptographie de hachage est souvent utilisée pour stocker les mots de passe de manière sécurisée. Ci-dessous se trouve la table User de la base de données utilisée, où les mots de passe sont stockés de manière sécurisée à l'aide de l'algorithme de hachage BCrypt :

```
CREATE TABLE User (
      id INT AUTO_INCREMENT NOT NULL,
      login VARCHAR(30) NOT NULL,
      password VARCHAR(255) NOT NULL,
      CONSTRAINT PK_User PRIMARY KEY(id)
);

INSERT INTO User (login, password) VALUES
      ("Pierre", "$2a$12$BFahOdxHet5ZXfiggU8Md.TtvJXteV0FVu.fqKOAddSGqzreYogoK"),
      ("Jean", "$2a$12$3i0ZpzmGQ8ZKDWRQtArvkeUSUgf1tMz07AcU5jQ50O8ykw9noZ3AC")
;
```

La classe Login implémente la fonctionnalité de vérification des informations d'identification saisies par l'utilisateur. L'exemple suivant montre comment le mot de passe haché est vérifié en utilisant BCrypt.checkpw.

```
this.btnLogin.addMouseListener(new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent e) {
            if (txtLogin.getText().equals(myController.getTheUser().getLogin())) {
                  if                                    (BCrypt.checkpw(txtPassword.getText(),
myController.getTheUser().getPassword())) {
                        myController.getTheLogin().setVisible(false);
                        myController.getTheConsole().setVisible(true);
                  } else {
                        System.out.println("Wrong login or password.");
                  }
            } else {
                  System.out.println("Wrong login or password.");
            }
      }
});
```

[21] https://www.cnil.fr/fr/comprendre-les-grands-principes-de-la-cryptologie-et-du-chiffrement

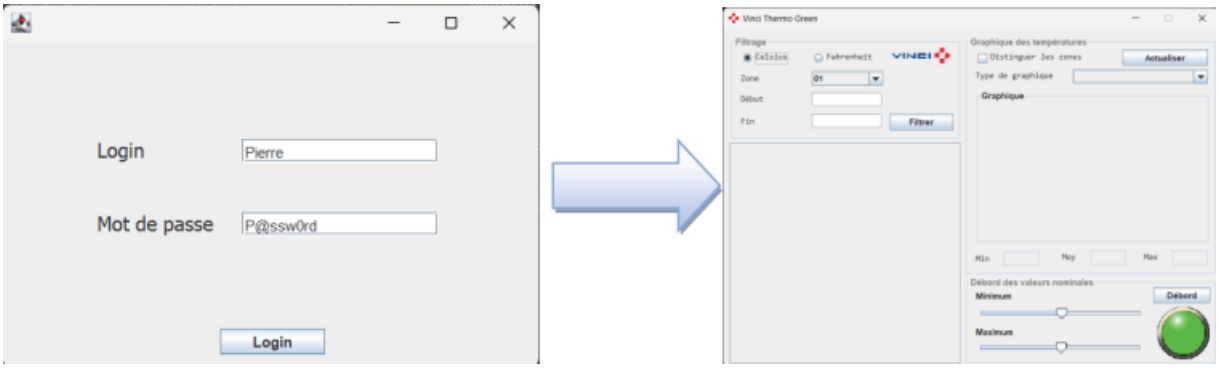


Figure 9. Fenêtre de login avec informations saisies

7.3 PRESENTATION DE JASYPT

Jasypt, acronyme de Java Simplified Encryption, est une bibliothèque open-source en Java qui facilite l'intégration de fonctionnalités de chiffrement dans les applications Java. Elle offre des fonctionnalités simples pour chiffrer des données sensibles, telles que des mots de passe, des configurations, ou d'autres informations confidentielles, de manière transparente.

Voici quelques points clés concernant Jasypt :

1. Chiffrement facile à utiliser : Jasypt simplifie le processus de chiffrement en fournissant des interfaces conviviales pour les développeurs Java. Elle propose une gamme de fonctionnalités de chiffrement pour répondre aux besoins de sécurité.
2. Intégration transparente : Jasypt peut être intégrée facilement dans des applications Java existantes. Les développeurs peuvent l'utiliser pour chiffrer et déchiffrer des données sans avoir à gérer manuellement des détails complexes liés au chiffrement.
3. Support pour divers algorithmes de chiffrement : Jasypt prend en charge plusieurs algorithmes de chiffrement, tels que DES, TripleDES, PBE, et d'autres. Cela permet aux développeurs de choisir l'algorithme qui correspond le mieux à leurs besoins en termes de sécurité.
4. Intégration avec des frameworks de configuration : Jasypt facilite la sécurisation des fichiers de configuration en permettant aux développeurs de chiffrer des propriétés spécifiques. Cela peut être particulièrement utile pour protéger des informations sensibles dans les fichiers de configuration, comme les informations de connexion à une base de données.
5. Support pour différentes sources de clés : Jasypt offre la possibilité d'utiliser diverses sources pour obtenir les clés de chiffrement, y compris des propriétés système, des variables d'environnement, des fichiers de propriétés, etc.

7.4 HACHAGE AVEC JASYPT

Jasypt est utilisé pour le chiffrement de données sensibles, notamment dans le contexte de la configuration d'une base de données. Les paramètres cryptés sont définis dans un fichier de configuration appelé vtg.cfg, et pour assurer la connexion à la base de données, ces paramètres sont décryptés dans la classe DAOmySQL.

Dans cette implémentation, Jasypt facilite le chiffrement des informations sensibles, offrant ainsi une couche de sécurité supplémentaire. Les paramètres tels que l'URL de la base de données, le nom d'utilisateur et le mot de passe sont initialement chiffrés dans le fichier de configuration vtg.cfg à l'aide de Jasypt.

```
# Database Configuration
db.url=Lnn0Go9OONvvWJTCdrzM5MsI1AJxzeVHSL4eU7MSeLBLLy35SmouXhArn8Bq7Y74
db.username=mE2rFwmY2RCocPpyjO0RoQ==
db.password=EraJXCzVDK7EHhfx+2jLlmQMYC/z23Rc
```

La classe DAOmySQL utilise ensuite la fonction de décryptage de Jasypt pour récupérer ces paramètres de manière sécurisée. L'utilisation de BasicTextEncryptor de Jasypt permet de spécifier un mot de passe de chiffrement, dans cet exemple, "P@ssw0rd". Ce mot de passe est utilisé pour initialiser l'objet de chiffrement, qui est ensuite employé pour décrypter les paramètres du fichier de configuration.

Les paramètres décryptés, tels que l'URL, le nom d'utilisateur et le mot de passe, sont ensuite utilisés pour établir une connexion sécurisée à la base de

données MySQL. Cela garantit que les informations sensibles liées à la base de données sont protégées, même si le fichier de configuration est accessible.

```
public class DAOmySQL {

	// Specification
	private Properties theConfig;
	private BasicTextEncryptor theEncryptor;

	Controller myController;
	String URL;
	String username;
	String password;
	Connection myConnection;
	Statement myStatement;

	public DAOmySQL(Controller aController) throws SQLException {
	 this.myController = aController;

	 try {
		Class.forName("com.mysql.cj.jdbc.Driver");
	 } catch (ClassNotFoundException e) {
		e.printStackTrace();
	 }

	 this.theEncryptor = new BasicTextEncryptor();
	 this.theEncryptor.setPassword("P@ssw0rd");

	 this.theConfig = new Properties();

	 try                    (InputStream                    input                    =
getClass().getClassLoader().getResourceAsStream("data/vtg.cfg")) {
	    theConfig.load(input);
	 } catch (IOException e) {
	    e.printStackTrace();
	 }

	 this.URL = theEncryptor.decrypt(theConfig.getProperty("db.url"));

	 this.username = theEncryptor.decrypt(theConfig.getProperty("db.username"));
	 this.password = theEncryptor.decrypt(theConfig.getProperty("db.password"));

	 this.myConnection = DriverManager.getConnection(URL, username, password);

	 this.myStatement = myConnection.createStatement();

	}

	/**
	 * Récupère la liste de tous les utilisateurs de la base de données.
	 * @return ResultSet contenant la liste des utilisateurs.
	 * @throws SQLException si une erreur survient lors de l'exécution de la
requête SQL.
	 */
	public ResultSet listUsers() throws SQLException {
	 String theQuery = "SELECT * FROM user;";
	 ResultSet theResultSet = myStatement.executeQuery(theQuery);
	 return theResultSet;
	}

	public ResultSet listStadiumNames() throws SQLException {
	 String theQuery = "SELECT name FROM stadium;";
	 ResultSet theResultSet = myStatement.executeQuery(theQuery);
	 return theResultSet;
	}
```

```
    public ResultSet listMeasures() throws SQLException {
     String theQuery = "SELECT * FROM measures;";
     ResultSet theResultSet = myStatement.executeQuery(theQuery);
     return theResultSet;
    }

    public void changePWD(String login, String pwd) throws SQLException {
        String theQuery = "UPDATE user SET password = ? WHERE login = ?";

        try (PreparedStatement preparedStatement =
myConnection.prepareStatement(theQuery)) {
            preparedStatement.setString(1, hashPassword(pwd));
            preparedStatement.setString(2, login);

            int rowsAffected = preparedStatement.executeUpdate();

            if (rowsAffected > 0) {
                // Le mot de passe a été changé avec succès
                System.out.println("Mot de passe changé avec succès ! Redémarrez
pour appliquer les changements.");
            } else {
                System.err.println("Aucune ligne mise à jour. Utilisateur non
trouvé.");
            }
        }
    }


    private static String hashPassword(String motDePasse) {
       // Génération d'un sel aléatoire (par défaut, BCrypt utilise un coût de 10)
       String salt = BCrypt.gensalt();

       // Hashage du mot de passe avec le sel
       return BCrypt.hashpw(motDePasse, salt);
    }
}
```

En outre, la classe DAOmySQL contient une méthode changePWD qui utilise Jasypt pour hasher le nouveau mot de passe avant de le mettre à jour dans la base de données. Cette approche renforce la sécurité en stockant les mots de passe de manière sécurisée, ce qui est essentiel pour la protection des informations utilisateur dans les applications Java.

8 Implementation des cas d'utilisation

8.1 Cas d'utilisation « Changer son mot de passe »

8.1.1 Presentation de Passay

La bibliothèque Java Passay est un outil open source qui facilite la validation et la génération de mots de passe conformes à des politiques de sécurité spécifiques. Elle est principalement utilisée pour renforcer la sécurité des applications en appliquant des règles et des contraintes aux mots de passe des utilisateurs.

Voici quelques caractéristiques clés de la bibliothèque Passay :

1. Validation des Mots de Passe : Passay fournit des mécanismes pour valider si un mot de passe répond aux critères de sécurité définis. Ces critères peuvent inclure des exigences telles que la longueur minimale, la présence de caractères spéciaux, la combinaison de majuscules et de minuscules, etc.
2. Génération de Mots de Passe : En plus de la validation, Passay peut également générer des mots de passe aléatoires conformes à des politiques prédéfinies. Cela peut être utile pour créer des mots de passe initiaux ou temporaires.
3. Intégration Facile : La bibliothèque est conçue pour être facilement intégrée dans des applications Java. Elle peut être utilisée avec des frameworks tels que Spring Security pour renforcer les mécanismes d'authentification.
4. Configuration Personnalisée : Passay offre une flexibilité significative en permettant aux développeurs de définir des politiques de mot de passe

personnalisées en fonction des besoins spécifiques de leur application.

5. Support pour Différents Algorithmes de Hachage : Passay prend en charge différents algorithmes de hachage couramment utilisés, améliorant ainsi la sécurité des mots de passe stockés.

8.1.2 IMPLEMENTATION

La classe MDPChange est conçue pour permettre aux utilisateurs de changer leur mot de passe en suivant des étapes sécurisées. L'interface utilisateur de cette classe inclut des champs pour le nom d'utilisateur actuel, l'ancien mot de passe, le nouveau mot de passe, et la confirmation du nouveau mot de passe. Lorsque l'utilisateur clique sur le bouton "Confirmer", le programme parcourt la liste des utilisateurs pour valider les informations fournies. Si les données sont valides et conformes aux règles de sécurité, le mot de passe est mis à jour dans la base de données.

```
public class MDPChange extends JFrame {

	private static final long serialVersionUID = 1L;
	private JPanel contentPane;

	private Controller myController;

	private JPanel thePanel;
	private URL vinciIcon;

	private JLabel lblUsername;
	private JLabel lblPassword;
	private JLabel lblNewPassword;
	private JLabel lblPasswordConfirm;

	private JTextField txtUsername;
	private JTextField txtPassword;
	private JTextField txtNewPassword;
	private JTextField txtPasswordConfirm;

	private JButton confirmBtn;
	private JButton cancelBtn;

	private ArrayList<User> lesUsers = new ArrayList<User>();

	/**
	 * Create the frame.
	 * @param controller
	 */
	public MDPChange(Controller aController) throws SQLException {

	 super();

	 this.myController = aController;
	 this.lesUsers = aController.getLesUsers();

	 vinciIcon = ConsoleGUI.class.getResource("/img/vinci_ico.jpg");
	 setIconImage(Toolkit.getDefaultToolkit().getImage(vinciIcon));
	 setTitle("Vinci Thermo Green - Changement de Mot de Passe");

	 setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	 setBounds(100, 100, 450, 300);

	 // Initialisation du panneau
	 this.thePanel = new JPanel();
	 this.getContentPane().add(thePanel, BorderLayout.CENTER);
	 thePanel.setLayout(null);

	 // Création du champ de nom d'utilisateur
	 lblUsername = new JLabel("Nom d'utilisateur");
	 lblUsername.setFont(new Font("Tahoma", Font.PLAIN, 15));
```

```
        lblUsername.setBounds(47, 46, 113, 19);
        thePanel.add(lblUsername);

        txtUsername = new JTextField();
        txtUsername.setBounds(244, 48, 135, 19);
        thePanel.add(txtUsername);
        txtUsername.setColumns(10);

        // Création du champ de l'ancien mot de passe
        lblPassword = new JLabel("Mot de Passe");
        lblPassword.setFont(new Font("Tahoma", Font.PLAIN, 15));
        lblPassword.setBounds(47, 87, 113, 19);
        thePanel.add(lblPassword);

        txtPassword = new JTextField();
        txtPassword.setColumns(10);
        txtPassword.setBounds(244, 89, 135, 19);
        thePanel.add(txtPassword);

        // Création du champ pour confirmer le nouveau mot de passe
        lblNewPassword = new JLabel("Nouveau Mot de Passe");
        lblNewPassword.setHorizontalAlignment(SwingConstants.LEFT);
        lblNewPassword.setFont(new Font("Tahoma", Font.PLAIN, 15));
        lblNewPassword.setBounds(47, 129, 161, 19);
        thePanel.add(lblNewPassword);

        txtNewPassword = new JTextField();
        txtNewPassword.setColumns(10);
        txtNewPassword.setBounds(244, 131, 135, 19);
        thePanel.add(txtNewPassword);

        // Création du champ du nouveau mot de passe
        lblPasswordConfirm = new JLabel("Confirmer le Mot de Passe");
        lblPasswordConfirm.setFont(new Font("Tahoma", Font.PLAIN, 15));
        lblPasswordConfirm.setBounds(47, 170, 181, 19);
        thePanel.add(lblPasswordConfirm);

        txtPasswordConfirm = new JTextField();
        txtPasswordConfirm.setColumns(10);
        txtPasswordConfirm.setBounds(244, 172, 135, 19);
        thePanel.add(txtPasswordConfirm);

        // Ajout du bouton Confirmer
        confirmBtn = new JButton("Confirmer");
        confirmBtn.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                // Parcours de la liste des utilisateurs
                for (User user : lesUsers) {
                    // Vérification des informations de connexion
                    if (user.getLogin().equals(txtUsername.getText())
                            &&                    BCrypt.checkpw(txtPassword.getText(),
user.getPassword())
                            && !(txtNewPassword.getText().equals(""))
                            && !(txtPasswordConfirm.getText().equals(""))
                            &&
txtNewPassword.getText().equals(txtPasswordConfirm.getText())) {

                        // Mise à jour du mot de passe dans la base de données
                        try {

aController.getMyDAOmySQL().changePWD(txtUsername.getText(),
txtNewPassword.getText());
                        } catch (SQLException e1) {
                            // Gestion de l'exception SQL
                            e1.printStackTrace();
```

```
                    }

                    // Affichage de la fenêtre de la console et masquage de la
fenêtre de connexion
                    aController.getTheMDPChange().setVisible(false);
                    break;
                }
            }
        }
    });

    confirmBtn.setFont(new Font("Tahoma", Font.PLAIN, 15));
    confirmBtn.setBounds(87, 220, 123, 21);
    thePanel.add(confirmBtn);

    // Ajout du bouton Annuler
    cancelBtn = new JButton("Annuler");
    cancelBtn.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
        }
    });
    cancelBtn.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            aController.getTheMDPChange().setVisible(false);
            aController.getTheLogin().setVisible(true);
        }
    });
    cancelBtn.setFont(new Font("Tahoma", Font.PLAIN, 15));
    cancelBtn.setBounds(221, 220, 123, 21);
    thePanel.add(cancelBtn);
   }

   private void pwdVerify(String newPassword) {
    PasswordValidator validator = new PasswordValidator(
            new LengthRule(8, 30),
            new CharacterRule(EnglishCharacterData.UpperCase, 1),
            new CharacterRule(EnglishCharacterData.Digit, 1),
            new CharacterRule(EnglishCharacterData.Special, 1),
            new WhitespaceRule(),
            new IllegalSequenceRule(EnglishSequenceData.USQwerty)
      );

        RuleResult result = validator.validate(new PasswordData(newPassword));

        if (result.isValid()) {
            // Le nouveau mot de passe est valide, vous pouvez le mettre à jour
dans votre système.
            System.out.println("Le nouveau mot de passe est valide.");
            // Code pour mettre à jour le mot de passe dans votre système
        } else {
            // Le nouveau mot de passe ne respecte pas les règles de validation
            System.out.println("Le nouveau mot de passe n'est pas valide. Règles
non respectées :");
            for (String message : validator.getMessages(result)) {
                System.out.println(message);
            }
        }
    }
}
```

Le processus de validation du nouveau mot de passe est renforcé grâce à l'intégration de la bibliothèque Passay. Cela garantit que le nouveau mot de passe respecte les critères de sécurité tels que la longueur minimale, la présence de majuscules, de chiffres, de caractères spéciaux, etc. Si le nouveau mot de passe ne respecte pas ces règles, l'utilisateur est informé des critères non respectés.

8.2 Cas d'utilisation « Alerter le personnel d'astreinte »

8.2.1 Presentation de l'API Vonage

Vonage est une entreprise spécialisée dans les solutions de communication cloud, offrant une gamme de services tels que la messagerie texte, la voix, la vidéo et les API de communication. L'API principale de Vonage était l'API Nexmo, qui a été renommée Vonage API.

Voici quelques-unes des fonctionnalités clés de l'API Vonage (anciennement Nexmo) :

- SMS (Short Message Service) : Permet d'envoyer et de recevoir des messages texte à travers le monde.
- Voice API : Offre des services de téléphonie vocale pour la programmation d'appels vocaux, de la synthèse vocale, des réponses vocales interactives (IVR), etc.
- Verify API : Permet la vérification en deux étapes via SMS ou appel vocal pour renforcer la sécurité des utilisateurs.
- Number Insight API : Fournit des informations détaillées sur un numéro de téléphone, telles que le type de ligne, la localisation, etc.
- Conversation API : Facilite la création d'applications de messagerie instantanée avec des fonctionnalités de chat en temps réel.
- Video API : Permet d'intégrer des fonctionnalités de vidéoconférence dans des applications.
- Messages API : Permet d'envoyer des messages multimédias tels que des images, vidéos et fichiers audio.

L'API Vonage est conçue pour être facile à intégrer dans différentes applications, offrant des fonctionnalités de communication avancées. Les développeurs peuvent l'utiliser pour enrichir leurs applications avec des fonctionnalités de messagerie, de voix et de vidéo, créant ainsi des expériences utilisateur plus interactives.

8.2.2 Implementation

La classe SMSSender a été implémentée pour tirer parti de l'API Vonage (anciennement Nexmo) afin d'envoyer des alertes par SMS au personnel d'astreinte. L'API Vonage offre divers services de communication cloud, dont la fonction SMS est utilisée ici. La classe SMSSender prend en charge la création d'un message texte avec un numéro de téléphone destinataire et un message, puis envoie ce message via l'API Vonage.

```
package control;

import com.vonage.client.VonageClient;
import com.vonage.client.sms.MessageStatus;
import com.vonage.client.sms.SmsSubmissionResponse;
import com.vonage.client.sms.messages.TextMessage;

public class SMSSender {

    private Controller myController;
    private VonageClient client;

    public SMSSender(Controller aController) {
      this.myController = aController;

      this.client                                                    =
VonageClient.builder().apiKey("23b2778c").apiSecret("gDJj7VDjytNPedBk").build();
    }

    public void sendSMS(String aPhoneNum, String aMessage) {
      TextMessage    message    =    new    TextMessage("Vonage    APIs",    "33"    +
aPhoneNum.substring(1, 10), aMessage);

      SmsSubmissionResponse response = client.getSmsClient().submitMessage(message);

      if (response.getMessages().get(0).getStatus() == MessageStatus.OK) {
          System.out.println("Message sent successfully.");
      } else {
```

```
            System.out.println("Message failed with error: " +
response.getMessages().get(0).getErrorText());
        }
    }
}
```

Dans la classe Alert, lorsqu'un utilisateur saisit un numéro de téléphone valide et un message, et clique sur le bouton "Envoyer", le programme vérifie la validité du numéro de téléphone et l'existence du message. Si ces conditions sont remplies, le message est envoyé via la classe SMSSender. Cela permet au personnel d'astreinte de recevoir des alertes importantes via des messages texte, renforçant ainsi la communication en cas d'urgence.

Figure 10. Frame de la classe Alert

```
// Création du bouton Envoyer
sendBtn = new JButton("Envoyer");
sendBtn.addMouseListener(new MouseAdapter() {
      @Override
      public void mouseClicked(MouseEvent e) {
            if (telText.getText().matches("0[0-9]{9}")
                        && !(msgText.getText().equals(""))) {
                  aController.getMySMSSender().sendSMS(telText.getText(),
msgText.getText());
                  aController.getTheAlert().setVisible(false);
                  aController.getTheConsole().setVisible(true);
            }
      }
});
sendBtn.setBounds(109, 217, 85, 21);
thePanel.add(sendBtn);
```

L'expression régulière utilisée dans la méthode matches() (à savoir « 0[0-9]{9} ») indique que la chaîne de caractère saisie dans le champ de texte du numéro de téléphone doit avoir un 0 pour premier caractère (d'où le chiffre 0 en début d'expression), suivi de 9 autres chiffres (d'où le chiffre {9} entre accolades) compris entre 0 et 9 (d'où la composante [0-9] de l'expression régulière).

Ces implémentations démontrent comment les fonctionnalités de sécurité et de communication sont intégrées dans l'application, assurant ainsi une gestion robuste des mots de passe et la capacité d'alerter le personnel d'astreinte de manière efficace.

9 Bibliographie