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.

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.

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] :

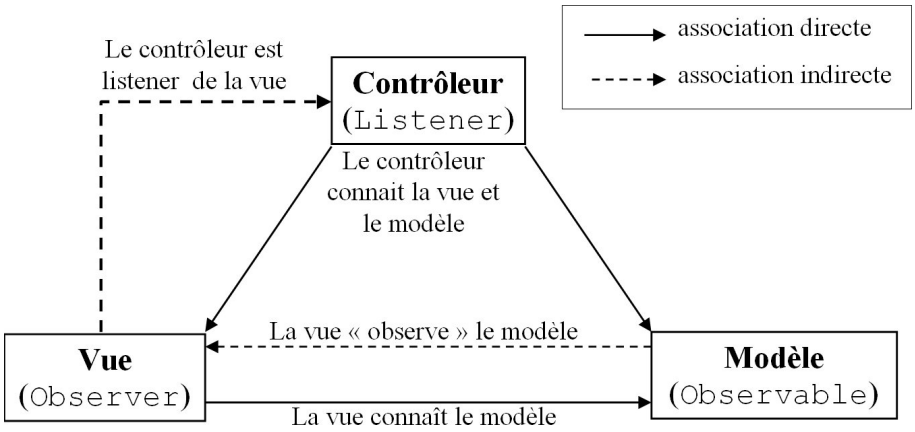


Figure 1 : principe de fonctionnement du modèle MVC

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…)

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) :

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

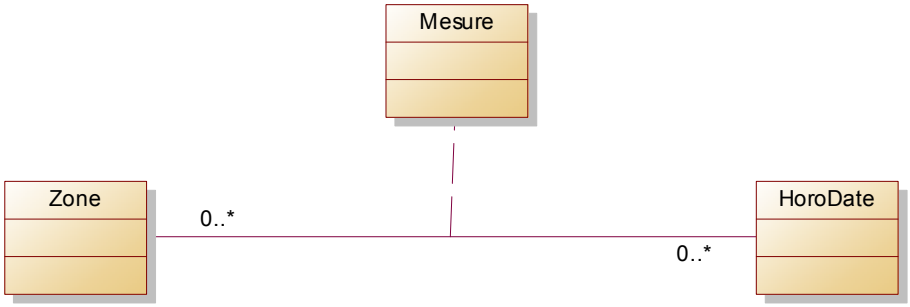


Figure 2 : diagramme des classes métier

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
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;
```

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

```
		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 LIRE UN FICHIER

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/

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/

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() :

```
      /**
       * <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();
                        }
```

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

```
                    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 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);
```

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

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 UN FICHIER XML

@TODO version ultérieure

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);
```

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

```
      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);
                        }
                  }
            }
      }
```

```
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 :

[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

- 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 :

```
/**
 * <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());
```

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

```
                // 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.

[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

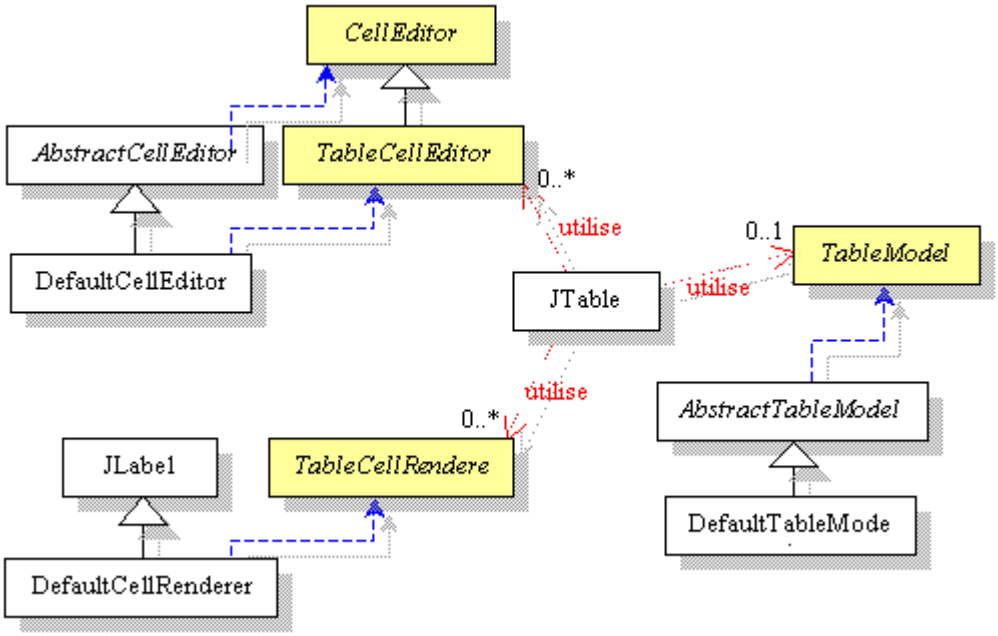


Figure 3 : diagramme des classes associées au JTable

```
/**
 * <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).

[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

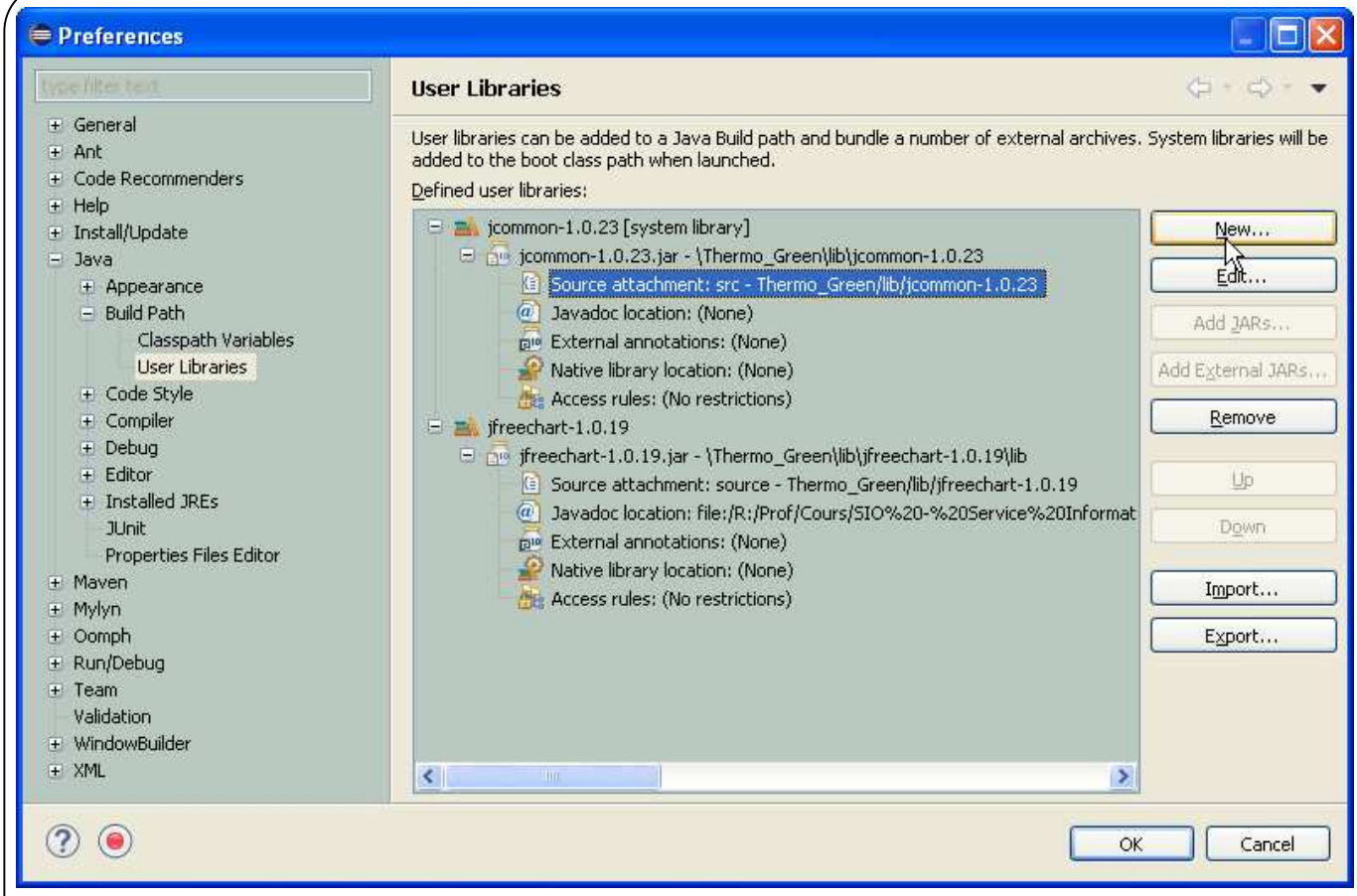


Figure 4 : installation de JFreeChart dans Eclipse

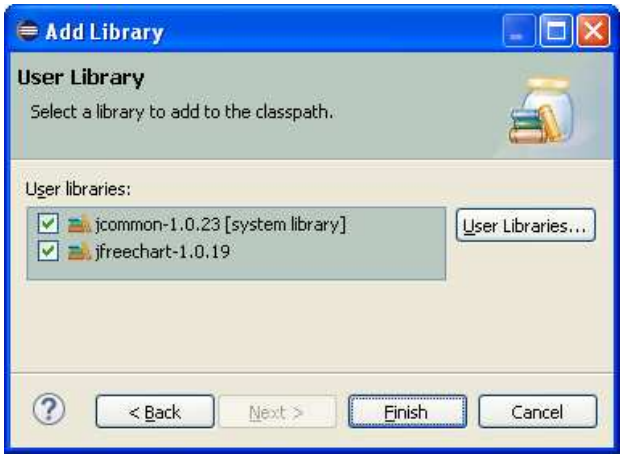


Figure 5 : ajout de JFreeChart au projet

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>
```

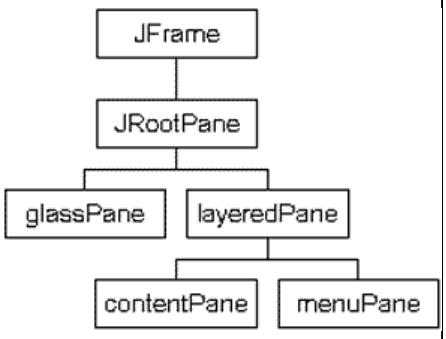
```
	 * <li>alimente le container des donn&eacute;es</li>
	 * <li>Fabrique un graphique lin&eacute;aire -&gt;
ChartFactory.createLineChart</li>
	 * <li>Englobe le graphique dans un panel sp&eacute;cifique -&gt; new
ChartPanel(chart)</li>
	 * <li>Englobe ce panel dans un JPanle de l'IHM -&gt;
pnlGraph.add(chartPanel)<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.



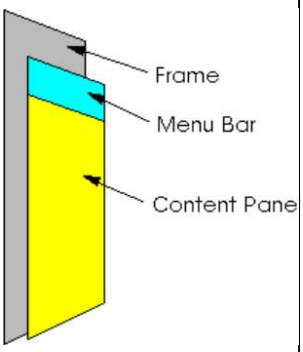
Les composants top-level possèdent un content pane qui contient tous les composants visibles d'un top-level. Un composant top-level peut aussi contenir une barre de menu

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 :

- 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();
```

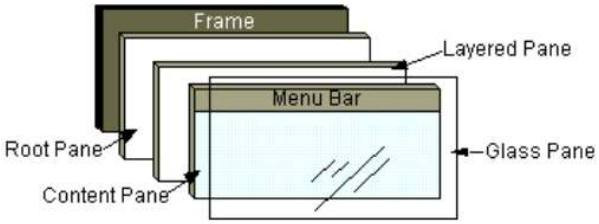


Les conteneurs intermédiaires sont utilisés pour structurer l'application graphique.
Le composant top-level contient des composants conteneur intermédiaires.

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.

[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

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]

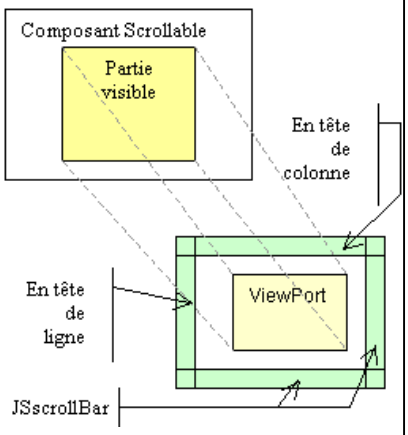
Pour placer des composants dans un container, Java utilise le "Layout".

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.
- 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);
```

14 http://docs.oracle.com/javase/tutorial/uiswing/layout/index.html
15 https://docs.oracle.com/javase/tutorial/uiswing/components/scrollpane.html

```
        //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.

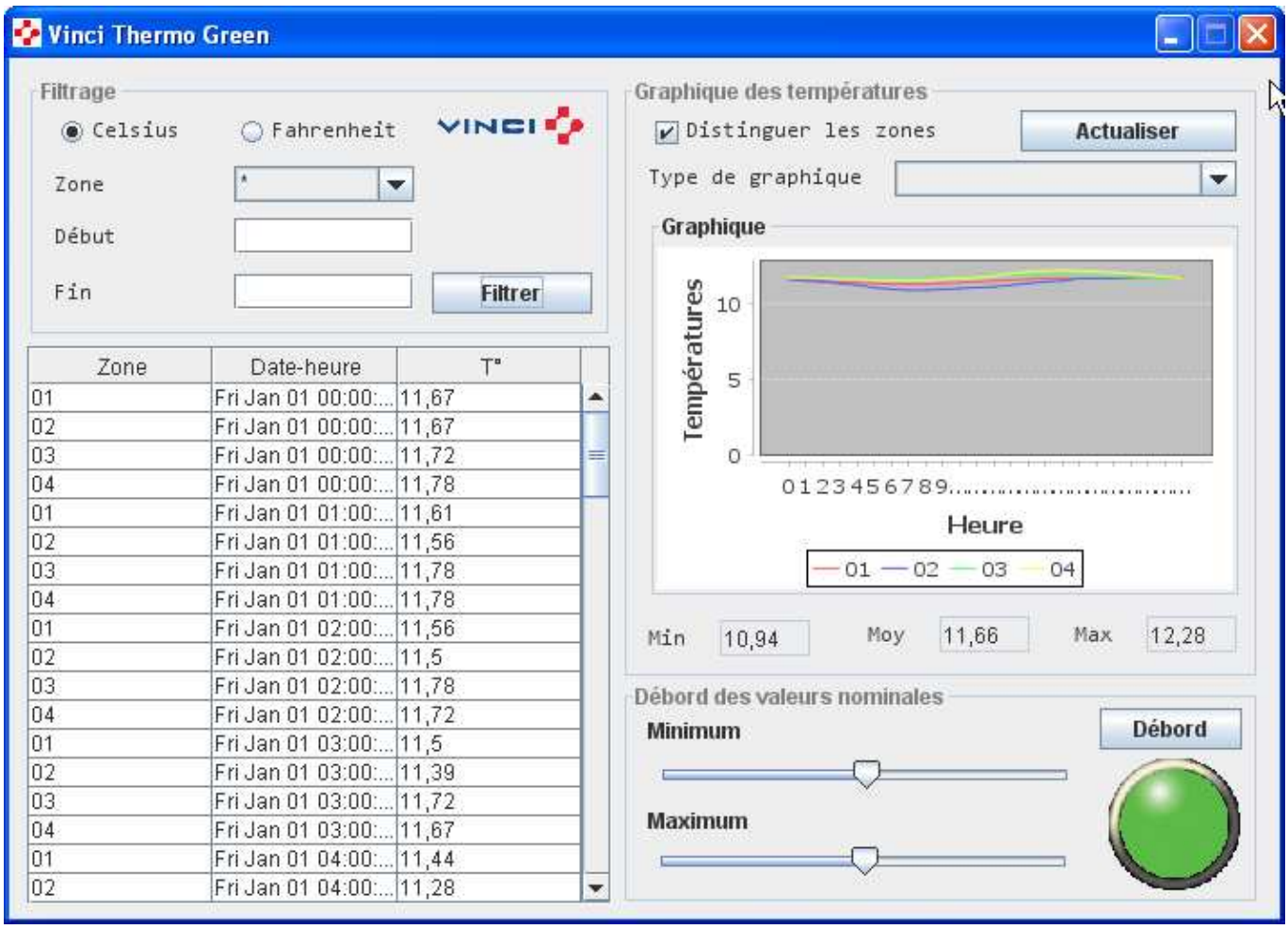


Figure 6 : IHM cible du prototype

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].

[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

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 :

```
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;

/**
```

[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

```
 * 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 TEMPERATURES[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>
       */
```

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

```
	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
	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");
	[…]
```

6.3.4 Les zones de saisie

6.3.5 La case a cocher pour distinguer les zones dans le graphique

6.3.6 Les deux curseurs (JSlider) pour borner les temperatures nominales

[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

6.3.7 La gestion du feu vert – feu rouge

6.3.8 La gestion des calendriers pour saisir les dates