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	24-09-2023	Enzo Marion	Analyse conception du code

Validation

Version	Date	Nom	Rôle

Diffusion

Version	Date	Nom	Rôle
1.0.0	24-09-2023	Jérôme VALENTI	Professeur

OBJET DU DOCUMENT

L'objectif principal du projet Vinci Thermo Green est de créer une application permettant de collecter, filtrer, afficher et analyser les mesures de température de différentes zones de pelouses. Les principales fonctionnalités de l'application incluent :

- Collecte des Mesures : L'application doit être capable de lire les mesures de température à partir d'un fichier de données CSV. Chaque mesure est associée à une zone, une date et une température en degrés Fahrenheit.

- Filtrage des Données : Les utilisateurs doivent pouvoir filtrer les données en fonction de critères tels que la zone, la plage de dates, etc.

- Affichage des Données : Les données filtrées doivent être affichées dans une interface utilisateur graphique (GUI) conviviale, permettant aux utilisateurs de visualiser les mesures sous forme de tableaux et de graphiques.

- Conversion de Température : L'application doit prendre en charge la conversion automatique de la température entre Fahrenheit et Celsius, selon les préférences de l'utilisateur.

Sommaire

1 Documentation du Modele-Vue-Controleur (MVC)

1.1 Introduction au Modele-Vue-Controleur (MVC) :

Le modèle MVC est un modèle architectural qui sépare une application en trois composants principaux : le Modèle, la Vue et le Contrôleur. Cette séparation permet de mieux organiser le code, d'améliorer la maintenabilité et de favoriser la réutilisation du code.

- Modèle (Model) : Le modèle représente les données et la logique métier de l'application. Il est responsable de la gestion des données et de leur manipulation. Dans le contexte de mon application, les classes dans le package model (comme Mesure) sont des exemples de Modèle.

- Vue (View) : La vue est responsable de l'affichage des données à l'utilisateur. Elle présente les informations du modèle de manière adaptée à l'interface utilisateur. Dans mon code, la classe ConsoleGUI dans le package view est la Vue. Elle gère l'interface graphique et l'affichage des données à l'utilisateur.

- Contrôleur (Controller) : Le contrôleur agit comme un intermédiaire entre la Vue et le Modèle. Il reçoit les entrées de l'utilisateur de la Vue, traite ces entrées et met à jour le Modèle en conséquence. Dans mon code, la classe Controller dans le package control est le Contrôleur. Il traite les demandes de l'utilisateur, filtre les données et interagit avec le Modèle.

1.2 Application du Modele-Vue-Controleur a mon code :

Dans mon code, l'application du modèle MVC peut être expliquée comme suit :

- Modèle (Model) :
    - Les classes du package model (comme Mesure) représentent le Modèle de mon application. Elles contiennent les données relatives aux mesures de température.
- Vue (View) :
    - La classe ConsoleGUI dans le package view est la Vue de mon application. Elle gère l'interface utilisateur graphique (GUI) à l'aide de composants Swing.
    - La classe ConsoleGUI affiche les données au format graphique, permet à l'utilisateur de filtrer les données, et présente un graphique des températures.
- Contrôleur (Controller) :
    - La classe Controller dans le package control est le Contrôleur de mon application. Elle traite les demandes de l'utilisateur à partir de l'interface graphique (ConsoleGUI).
    - Le Contrôleur interagit avec le Modèle pour récupérer les données de mesure de température et les filtre en fonction des critères fournis par l'utilisateur.

2 Analyse du code

2.1 Partie 1 : Modele (Model)

2.1.1 Extrait de Code pour le Modele :

```
package model;

import java.util.Date;

public class Mesure {
private String numZone;
private Date horoDate;
private float fahrenheit;

// Constructeurs, getters et setters...
}
```

Figure 1 : portion de code Mesure.java

2.1.2 Analyse du Modele :

- Le Modèle est représenté par la classe Mesure dans le package model. Cette classe contient les données relatives aux mesures de température.
- Les attributs numZone, horoDate et fahrenheit stockent respectivement le numéro de la zone, la date et l'heure de la mesure, ainsi que la température en degrés Fahrenheit.
- Cette classe définit des constructeurs, des getters et des setters pour accéder et manipuler ces données.

2.2 Partie 2 : Vue (View)

2.2.1 Extrait de Code pour la Vue :

```
package View;

import java.awt.Color;
import java.awt.Container;
import java.awt.Font;
import java.awt.Toolkit;
import javax.swing.*;

import control.Controller;
import model.Mesure;

public class ConsoleGUI extends JFrame {
// ... Déclarations d'attributs, de composants graphiques et de méthodes ...

class filtrerData implements ActionListener {
public void actionPerformed(ActionEvent e) {
// Traitement des actions de l'utilisateur...
}
}
}
```

Figure 2 : Portion de code ConsoleGUI.java

2.2.2 Analyse de la Vue :

- La Vue est représentée par la classe ConsoleGUI dans le package View. Cette classe gère l'interface utilisateur graphique (GUI) de l'application.
- Elle utilise la bibliothèque Swing pour créer des composants graphiques tels que des boutons, des étiquettes, des zones de texte, des graphiques, etc.

- Les composants GUI sont configurés avec des attributs tels que la police, la couleur et la disposition.
- La classe ConsoleGUI contient également une classe interne filtrerData qui implémente l'interface ActionListener. Cela permet de gérer les actions de l'utilisateur, comme le filtrage des données, lorsqu'un bouton est cliqué.

2.3 Partie 3 : Controleur (Controller)

2.3.1 Extrait de Code pour le Controleur :

```
package control;

import java.util.ArrayList;
import model.Mesure;

public class Controller {
private ArrayList<Mesure> lesMesures = new ArrayList<>();

public ArrayList<Mesure> filtrerLesMesure(String string) {
ArrayList<Mesure> mesuresFiltrees = new ArrayList<>();
// Filtrage des données en fonction du critère fourni...
return mesuresFiltrees;
}

// ... Autres méthodes et logique du contrôleur ...
}
```

Figure 3 : Portion de code Controller.java

2.3.2 Analyse du Controleur :

- Le Contrôleur est représenté par la classe Controller dans le package control. Cette classe gère la logique de l'application et sert d'intermédiaire entre la Vue (ConsoleGUI) et le Modèle (Mesure).
- La classe Controller contient une liste lesMesures qui stocke les données de mesure de température.
- La méthode filtrerLesMesure est utilisée pour filtrer les données en fonction du critère fourni en argument. Dans cet extrait, elle retourne une liste vide, mais vous pouvez implémenter la logique de filtrage en utilisant les données du Modèle.
- Le Contrôleur est responsable de l'interaction avec le Modèle pour récupérer les données et de l'envoi des données filtrées à la Vue.

2.4 Analyse Globale :

- Votre code suit le modèle MVC en séparant clairement les responsabilités du Modèle, de la Vue et du Contrôleur.
- Le Modèle (Mesure) contient les données métier.
- La Vue (ConsoleGUI) gère l'interface utilisateur et l'affichage des données.
- Le Contrôleur (Controller) traite les actions de l'utilisateur et interagit avec le Modèle pour obtenir et manipuler les données.
- Cette séparation permet de rendre votre code modulaire, évolutif et plus facile à maintenir. Les interactions entre les composants MVC sont bien définies, ce qui améliore la clarté et la compréhension du code.

3 Conception du Code

La conception de mon code suit largement les principes du modèle MVC. Voici comment chaque composant MVC est représenté dans mon code :

- Modèle (Model) :
    - Les classes dans le package model (comme Mesure) contiennent les données métier (mesures de température).

- Vue (View) :
    - La classe ConsoleGUI dans le package view gère l'interface utilisateur graphique (GUI) et affiche les données au format graphique.
    - Elle utilise des composants Swing pour créer une interface utilisateur conviviale.
    - La Vue permet également à l'utilisateur de filtrer les données et affiche un graphique des températures.

- Contrôleur (Controller) :
    - La classe Controller dans le package control agit comme le Contrôleur de l'application. Elle reçoit les actions de l'utilisateur depuis la Vue (ConsoleGUI).
    - Le Contrôleur interagit avec le Modèle pour récupérer et filtrer les données.
    - Il met ensuite à jour la Vue avec les données filtrées et le graphique approprié.

VTG - Vinci Thermo Green - Documentation

4 Introduction

4.1 Objectif du Document

Ce document a pour but de guider et de documenter le développement du projet VTG (Vinci Thermo Green). Il couvre les différentes phases du projet, de la conception à l'implémentation, ainsi que les options offrant des points supplémentaires.

4.2 Structure du Projet

Le projet VTG se compose de plusieurs phases, chacune se concentrant sur des aspects spécifiques du développement. Ce document décrit les étapes de chaque phase et fournit des ressources utiles pour l'implémentation.

5 Phase 3 : IHM avec Swing

5.1 Analyse Conception

5.1.1 Cas d'Utilisation : "S'Identifie"

Le cas d'utilisation "S'Identifie" est essentiel pour permettre aux utilisateurs d'accéder à l'application en fournissant leurs informations d'identification.

```
// Création de la fenêtre d'identification
JFrame frame = new JFrame("Identification");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(400, 200);

// Ajout des composants Swing pour la saisie du nom
d'utilisateur et du mot de passe
JLabel labelNomUtilisateur = new JLabel("Nom
d'utilisateur:");
JTextField champNomUtilisateur = new JTextField(20);
JLabel labelMotDePasse = new JLabel("Mot de passe:");
JPasswordField champMotDePasse = new
JPasswordField(20);
JButton boutonConnexion = new JButton("Se Connecter");
```

Figure 4 : IHM avec Swing

5.1.2 Scenario Objet

Le scénario objet décrit comment les objets interagissent lors de la phase d'identification des utilisateurs.

```
boutonConnexion.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        String nomUtilisateur = champNomUtilisateur.getText();
        String motDePasse = new
String(champMotDePasse.getPassword());

        // Ici, vous effectuez la vérification de l'identité de
l'utilisateur
        // en vérifiant le nom d'utilisateur et le mot de passe dans
la base de données.
        boolean utilisateurValide =
verifierUtilisateur(nomUtilisateur, motDePasse);

        if (utilisateurValide) {
            // L'utilisateur est authentifié, vous pouvez afficher
l'interface principale.
            afficherInterfacePrincipale();
            frame.dispose(); // Fermer la fenêtre d'identification
        } else {
            // Afficher un message d'erreur si l'authentification a
échoué.
            JOptionPane.showMessageDialog(null, "Identifiant ou mot
de passe incorrect.");
        }
    }
});
```

Figure 5 : Gère les interactions de l'utilisateur

5.2 Programmer's Handbook

Ce manuel du programmeur fournit des directives pour la création de l'IHM (Interface Homme Machine) avec Swing. Il couvre des aspects tels que la gestion des composants Swing, la disposition, et bien plus encore.

6 Phase 4 : jBCrypt - Hachage des Mots de Passe

6.1 Programmer's Handbook

6.1.1 Introduction a jBCrypt

Cette section explique les concepts fondamentaux de jBCrypt, une bibliothèque utilisée pour le hachage des mots de passe. Le hachage sécurisé des mots de passe est essentiel pour protéger les données sensibles des utilisateurs.

6.1.2 Utilisation de jBCrypt

Vous apprendrez comment utiliser jBCrypt pour hacher les mots de passe des utilisateurs dans votre application. Le hachage des mots de passe est un moyen efficace de sécuriser les informations d'identification.

```
String motDePasse = "motDePasseDeLUtilisateur";
String motDePasseHache = BCrypt.hashpw(motDePasse, BCrypt.gensalt());
```

Figure 6 : Démo de JbCrypt

6.1.3 Ressources Utiles

- [GitHub jBCrypt Repository](https://github.com/jeremyh/jBCrypt)
- [Site Officiel de jBCrypt](https://www.mindrot.org/projects/jBCrypt/)
- [Générateur de Hachage BCrypt](https://bcrypt-generator.com/)

7 Phase 5 : JDBC - La Persistance des Donnees

7.1 Programmer's Handbook

Cette phase se concentre sur l'accès et la manipulation des données à l'aide de JDBC (Java Database Connectivity). Vous découvrirez comment établir une connexion avec la base de données, exécuter des requêtes et mettre en œuvre la persistance des données.

```
// Création de la connexion à la base de données
String url = "jdbc:mysql://localhost/thg_db";
String utilisateur = "root";
String motDePasse = "coucu";
Connection connexion = DriverManager.getConnection(url, utilisateur, motDePasse);
```

Figure 7 : Connexion a la bd mysql

8 Options (Donnant des Points Supplementaires)

8.1 Option 1 : Internationalisation de l'Application

Cette option consiste à rendre l'application capable de supporter plusieurs langues et régions, offrant ainsi une meilleure expérience utilisateur aux utilisateurs de différentes origines.

8.2 Option 2 : Cas d'Utilisation "Changer son Mot de Passe" avec la Bibliotheque Passay

L'option 2 consiste à implémenter le cas d'utilisation "Changer son Mot de Passe" en

utilisant la bibliothèque Passay pour appliquer des politiques de sécurité strictes aux mots de passe.

8.3 Option 3 : Application Multi-Stades

Cette option vise à développer une application qui peut gérer plusieurs étapes ou phases. Cela peut être utile pour les processus complexes ou les applications à plusieurs niveaux.

8.4 Conclusion

Mon code suit une architecture MVC qui permet de bien séparer les responsabilités entre le Modèle, la Vue et le Contrôleur. Cela facilite la maintenance, la réutilisation et l'extension de mon application. La Vue (GUI) est clairement séparée du Modèle, ce qui rend mon application plus modulaire et flexible. Le Contrôleur assure la communication entre la Vue et le Modèle, garantissant ainsi une interaction utilisateur fluide.

9 Legende des ilustrations :