Etablissement : ISET-Charguia	**Département :** Technologies de l'Informatique
Matière : Atelier POO Avancée	**Année :** 2ème année
	Année Universitaire : 2023-2024

TP n° 5.2 : Les interfaces graphiques sous JAVA

Objectif du TP :
- *Programmation événementielle*
- *Application suivant le pattern MVC- Model View Control-*

Exercice 1

Écrire un programme qui crée une fenêtre (de type JFrame) et qui détecte les événements "appui" et "relâchement" associés à la souris et ayant la fenêtre comme source. On se contentera de signaler chacun de ces deux événements en affichant en fenêtre console un message précisant sa nature (appui ou relâchement), ainsi que les coordonnées correspondantes.

On proposera quatre solutions :

1. la fenêtre a son propre écouteur de souris et elle implémente l'interface MouseListener,
2. on utilise un écouteur différent de la fenêtre, objet d'une classe implémentant l'interface MouseListener,
3. on utilise un objet écouteur différent de la fenêtre en recourant à l'adaptateur MouseAdapter,
4. on utilise un écouteur différent de la fenêtre, objet d'une classe anonyme dérivée de MouseAdapter.

Exercice 2

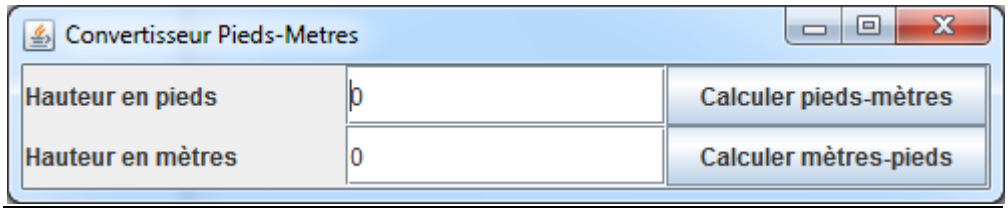
On se propose de réaliser une application permettant de mettre en forme (Couleur et Casse) un texte saisi comme l'illustre l'interface illustrée par la figure:

Interface de mise en forme d'un texte saisi

Fonctionnement :

- L'utilisateur introduit un texte dans la zone de saisie.
- Lorsque l'utilisateur sélectionne le bouton radio intitulé « Rouge » le texte saisi est affiché dans la zone d'affichage en rouge, la sélection du bouton radio « Bleu » rend le texte bleu.
- L'utilisateur coche la case Majuscule, le texte saisi est affiché dans la zone d'affichage en majuscule. Lorsque le bouton est décoché le texte est affiché en minuscule.

Exercice 3 - Partie Vue d'un convertisseur « pieds-mètres »
Soit l'interface graphique suivante :



1. Créer une classe *ConvertisseurView* qui hérite de la classe *JFrame*. Cette classe gèrera la partie Vue, et la création d'une instance de cette classe devra générer l'interface suivant :
   Pour cela, récupérer le conteneur de la fenêtre avec la méthode *this.getContentPane()* et associer une mise en page en appelant le constructeur *GridLayout(2,3)* comme vous l'avez appris dans l'atelier 4.1. Dans le conteneur, rajouter les Widgets suivant :
   - deux objets *piedsLabel* et *metresLabel* de la classe *JLabel*
   - deux objets *piedsTextField* et *metresTextField* de la classe *JTextField* (ayant chacun 0 comme valeur par défaut).
   - deux objets *piedsButton* et *metresButton* de la classe *JButton*
2. Créer une classe principale *ConvertisseurRun* qui crée un objet de la classe ConvertisseurView.

Exercice 4 – Partie Ecouteur (Listener)
3. Si vous avez déclaré les variables *piedsTextField* et *metresTextField* en variables locales dans le constructeur de la classe ConvertisseurVue, déplacer les pour qu'ils deviennent des attributs privés de cette même classe.

4. Dans la classe *ConvertisseurVue*, rajouter les méthodes suivantes :

```
public void afficherPieds() {
	float pieds = Float.parseFloat(metresTextField.getText()) / 0.305f;
	String piedsText = String.valueOf(pieds);
	piedsTextField.setText(String.valueOf(piedsText));
}

public void afficherMetres() {
	float metres = Float.parseFloat(piedsTextField.getText()) * 0.305f;
	String metresText = String.valueOf(metres);
	metresTextField.setText(String.valueOf(metresText));
}
```

5. Créer une classe *ConvertisseurEvent* ayant le code suivant :

```
import java.awt.event.*;

public class ConvertisseurEvent implements ActionListener {
	private ConvertisseurView convertisseurView;

	public ConvertisseurEvent(ConvertisseurView convertisseurViewParameter) {
		convertisseurView = convertisseurViewParameter;
	}

	public void actionPerformed(ActionEvent e) {
		if (e.getActionCommand().equals("piedsmetres")) {
			System.out.println("piedsmetres");
```

```
                convertisseurView.afficherMetres();
            } else if (e.getActionCommand().equals("metrespieds")) {
                System.out.println("metrespieds");
                convertisseurView.afficherPieds();
            }
        }
}
```

6. Dans le constructeur de la classe *ConvertisseurVue*, rajouter les lignes suivantes pour rattacher des événements définis dans la classe *ConvertisseurEvent* aux boutons :

```
        piedsButton.addActionListener(new ConvertisseurEvent(this));
        piedsButton.setActionCommand("piedsmetres");
        metresButton.addActionListener(new ConvertisseurEvent(this));
        metresButton.setActionCommand("metrespieds");
```

7. Tester et essayer de comprendre les codes des fichiers .java.

<u>Exercice 5</u> – Fenêtre d'authentification –MVC-

Vous allez créer 5 classes dans 5 fichiers séparés :

- *LoginView* Cette classe créera la partie Vue de l'application, et plus précisément une fenêtre contenant un champ texte JTextField pour le login et un champ JPasswordField pour le mot de passe, et un bouton JButton pour valider ainsi que deux étiquettes JLabel pour décrire les champs textes. Le constructeur de cette classe prendra comme paramètre un objet de la classe LoginController.
  Dans cette classe LoginView, rajouter les méthodes d'accesseurs String getLogin() et String getPassword() qui renverront les valeurs des champs respectives.
  Rajouter également une méthode void afficher(boolean reponse) qui affichera la fenêtre si reponse est true, et la masque sinon.
  Finalement rajouter la méthode void afficherMessage(String message) ayant le code suivant :

```
        protected void afficherMessage(String msg) {
                JOptionPane.showMessageDialog(this, msg, "Information",
                                JOptionPane.INFORMATION_MESSAGE);
        }
```

- Créer un fichier login-password.txt avec des données exemples en utilisant la structure suivant :

```
login1:password1
login2:password2
login3:password3
```

- *LoginModel* Cette classe créera la partie Model de l'application. Cette classe aura comme attribut la variable nomFichier avec le nom du fichier contenant les logins et leurs mots de passes respectives, qui sera affectée dans le constructeur de la classe. De plus, cette classe contient une méthode public boolean estValide(String login, String pwd) qui retournera true si les login/pwd sont corrects, et false sinon.

```
import java.io.*;

public class LoginModel {
        private String nomFichier;

        /* l'argument est le nom du fichier contenant les inforamtions */
        public LoginModel(String nomFichier) {
                this.nomFichier = nomFichier;
        }
```

```
	/* la fonction retourne vrai si le mot de passe correspond au login */
	public boolean estValide(String login, String pwd) {
		String ligne = null;
		String[] temp = null;
		try {
			BufferedReader entree = new BufferedReader(new FileReader(
					_nomFichier));
			while ((ligne = entree.readLine()) != null) {
				temp = ligne.split(":");
				if (temp[0].compareTo(login) == 0) {
					if (temp[1].compareTo(pwd) == 0) {
						return true;
					} else {
						return false;
					}
				}
			}
		} catch (IOException e) {
			System.out.println("erreur dans: " + e);
		}
		return false;
	}
}
```

- *LoginEvent* Cette classe gèrera l'événement du bouton de validation. Cette classe a comme attribut la variable loginController de la classe LoginController qui est reçue comme paramètre dans le constructeur. La classe implémentera l'interface ActionListener, et notamment la méthode actionPerformed qui fera appel à boutonValider() de la classe LoginController.

- LoginController Cette classe créera la partie Controller de l'application. Cette classe aura comme attributs privés les deux objets loginView de la classe LoginView et loginModel de la classe LoginModel. Dans le constructeur, les attributs seront instanciés. La méthode start() visualise la fenêtre en appelant la méthode afficher(true) de l'attribut loginView de la classe LoginView.
  La méthode clé de cette classe sera la méthode void boutonValider() qui fera la communication entre la partie Vue et la partie Model du pattern MVC. Elle testera si les login/password rentrés dans la fenêtre correspondent à une entrée du fichier login-password.txt, et affichera une fenêtre en appelant la méthode afficherMessage(String msg) avec les messages « Login correct » ou « incorrect ».

- *LoginRun* Cette classe principale permet qui crée un objet de la classe LoginController et appelle la méthode start() de cet objet.