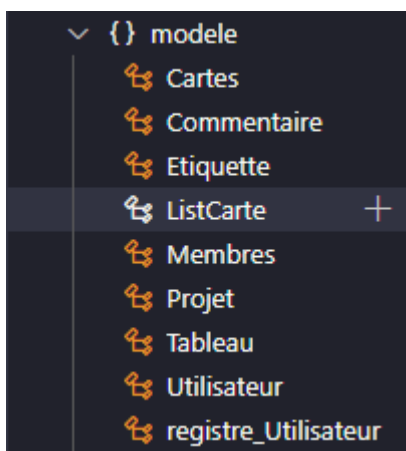


SAE DOO Partie Java

Nous allons vous expliquer comment nous avons procédé à la réalisation de l'application **Trello Lite**.

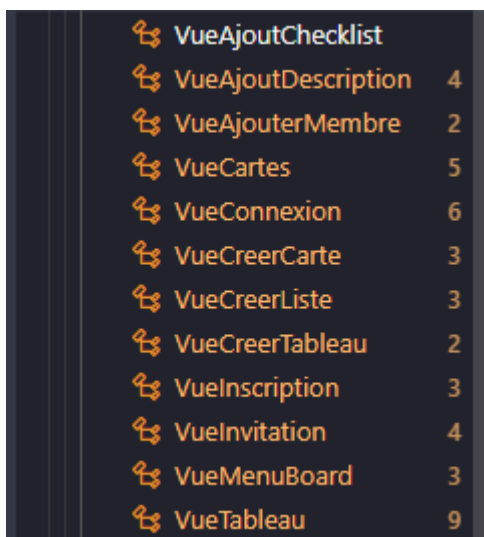
Tout d'abord, nous avons commencés par la partie modèle.

Nous avons réaliser les différent classe de l'application, cela comprend les classes *Cartes*, *Commentaire*, *Étiquette*, *ListCarte* etc.



Nous avons , à chacun, donné leur variable et caractéristique qui leur appartient et les différences.

Par la suite, nous avons réalisé les vus de l'application, ce qui concerne la fenêtre de connexion, de création de table, de liste etc.



Chacun crée à base d'une fenêtre (*JFrame/JDialog*), de *panel* et des élément tels que des *bouton*, *labels*, *menu*, etc.

```
public class VueTableau extends JPanel
```

```
private JButton ajouterListeButton;
```

```
private JFrame fenetreTableau;
```

```
private JMenuBar menuBar;  
private JMenu Listetableau,nameboard
```

Pour **les vus**, nous somme partie d’une base simple, sobre et minimaliste pour réaliser cette application.

Connexion

Identifiant :

Mot de passe:

Guest

Inscription

Connexion

Annuler

Creation Tableau

Entrez le nom du tableau :

Ajoutez un fond au Tableau ? :

Oui

Non

Ajouter un Membre

Creer

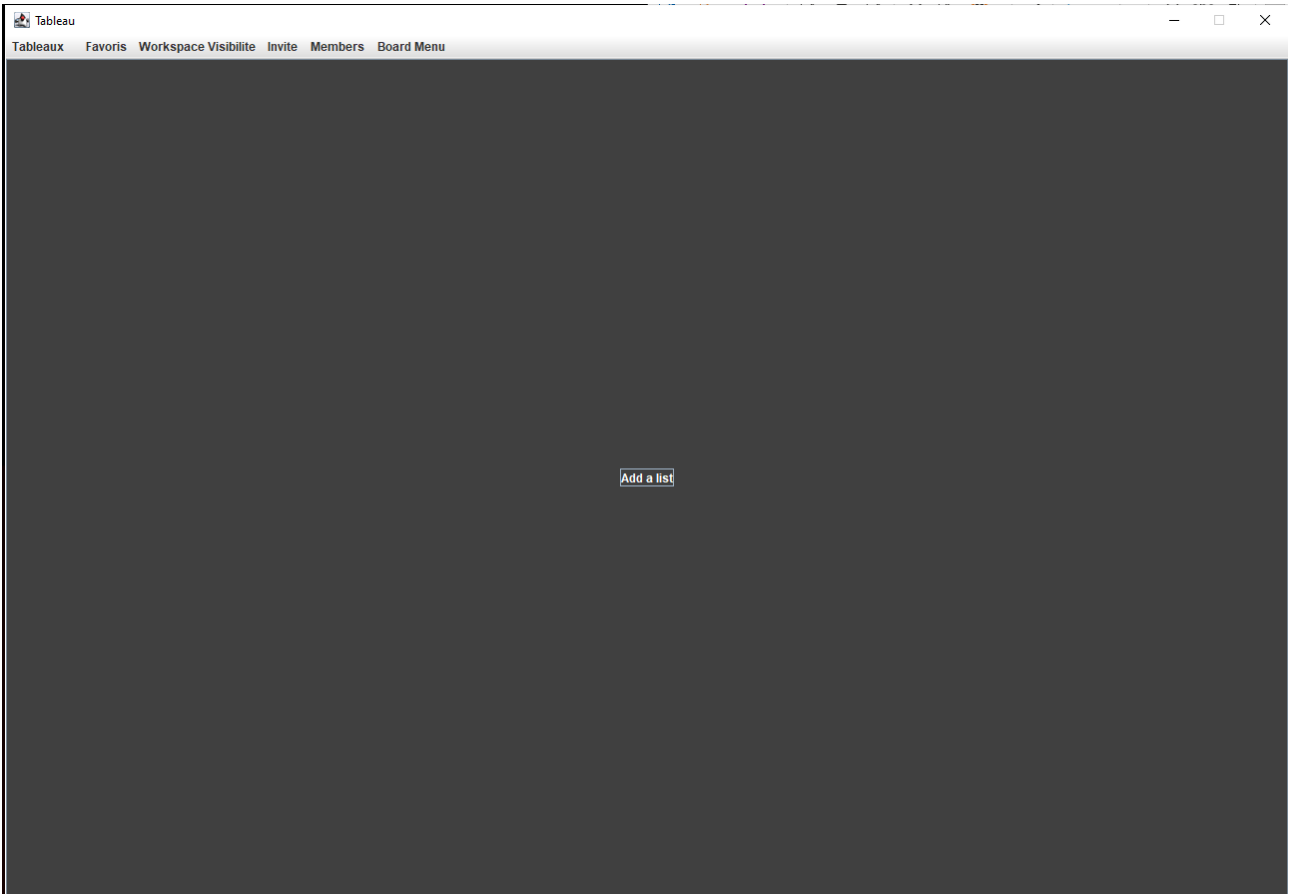
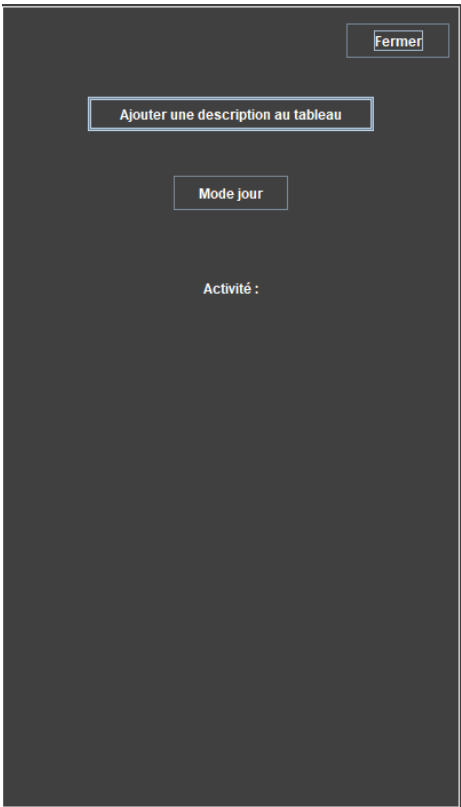
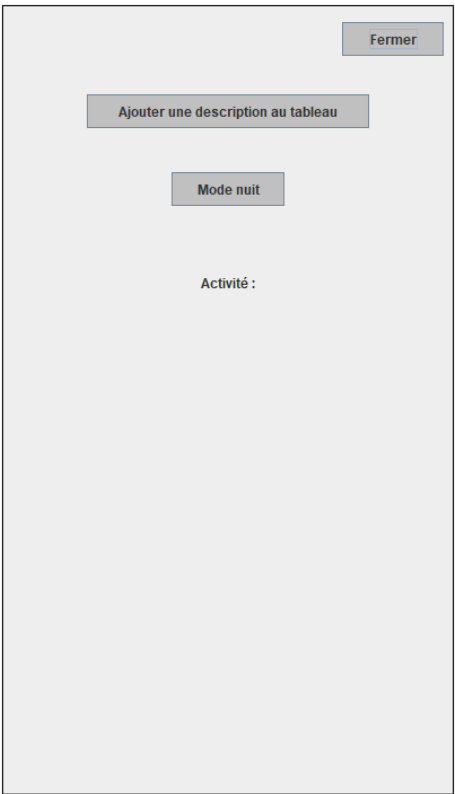
Annuler

Tableau

TableauxFavorisWorkspace VisibiliteInviteMembersBoard Menu

Add a list

Nous avons aussi réalisés une version « *Mode Nuits* » en noir pour avoir la possibilité de choisir ou pas de le mettre, possible dans le menu du tableau.



Le design et la police des textes de l'application sont ceux par défaut de java, juste la couleur des arrières plans et des polices on été changer pour le « *Mode nuits* » et juste l'arrière plan des bouton pour le mode normal.

Pour réaliser ces modification, nous avons utilisé la classe « UIManager », fournie par **Java Swing** qui nous a permis de faire c'est réalisation seulement en quelque ligne.

Au début de la 1^{ère} vue « VueConnexion » :

```
//inisialisation du designe de l'aplication
//change la couleur en arriere plan des bouton
UIManager.put("Button.background", Color.LIGHT_GRAY);
```

Fonction du bouton « *Mode nuit* » dans « VueMenuBoard » :

- pour le noir :

```
if(!isNights)
{
    isNights = true;
    UIManager.put("OptionPane.background", Color.DARK_GRAY);
    UIManager.put("Panel.background", Color.DARK_GRAY);
    UIManager.put("OptionPane.messageForeground", Color.WHITE);
    UIManager.put("Button.background", Color.DARK_GRAY);
    UIManager.put("Button.foreground", Color.WHITE);
    UIManager.put("Label.foreground", Color.WHITE);
    new VueTableau(vt.getTableau(), isNights);
}
```

- pour le remettre par défaut :

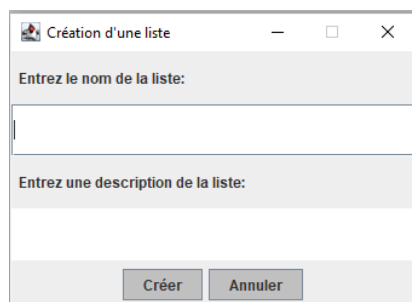
```
else
{
    isNights = false;
    UIManager.put("OptionPane.background", null);
    UIManager.put("Panel.background", null);
    UIManager.put("OptionPane.messageForeground", null);
    UIManager.put("Button.background", Color.LIGHT_GRAY);
    UIManager.put("Button.foreground", null);
    UIManager.put("Label.foreground", null);
    new VueTableau(vt.getTableau(), isNights);
}
```

Pour l'activation de ce mode nuit, nous avons initialisé une variable booléen dans la 1^{ère} vue en false, ce qui est le mode par défaut.

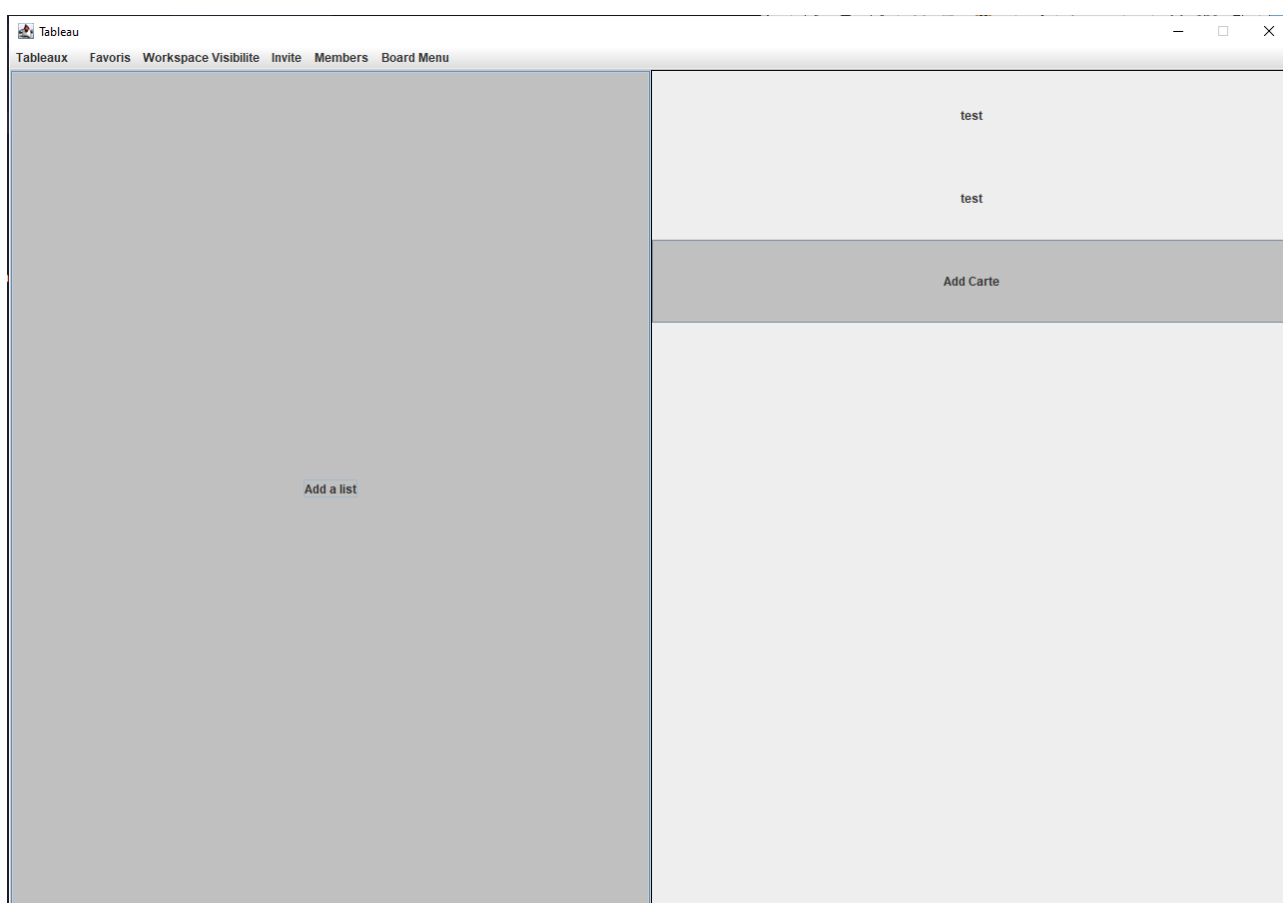
```
boolean isNights = false;
```

Pour que ensuite, qu'il se balade jusqu'à « VueTableau » pour l'utiliser dans le Menu et changer de mode. Obligé de faire de cette manière si nous voulons que **toute l'application garde cette état** et de **vouloir l'enlever quand on le voudra**.

Pour ajouter une nouvelle liste, on clique sur la fenetre qui mettra une nouvelle fenetre en face de nous :



A modal window titled "Création d'une liste" with standard window controls (minimize, maximize, close). It contains two text input fields: "Entrez le nom de la liste:" and "Entrez une description de la liste:". At the bottom, there are two buttons: "Créer" and "Annuler".



Une nouvelle interfaces s'affiche à droite du bouton qui est la carte qu'on vient de crée.

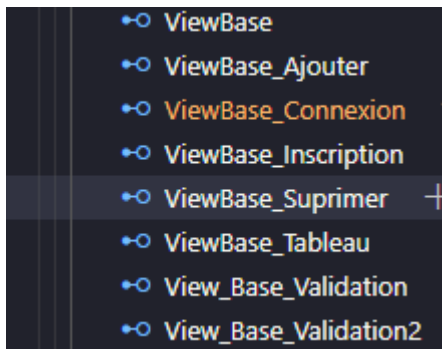
Pour la vue « VueMenuBoard », nous avons utilisé un `JDialog` au lieu d'un `JFrame`. Le but est de pouvoir empêcher l'utilisateur de faire autre chose avant d'avoir fini ce qu'elle a fait dans le menu et de le fermer avec une fonction propre à `JDialog` :

```
JDialog fenetreMenu = new JDialog ();
```

```
fenetreMenu.setModal(true); //bloc l'utilisateur
```

Passons maintenant **au contrôleur**

Pour pouvoir réaliser **les contrôleurs**, nous avons réalisés des interfaces, pour définir des fonctions ou nous allons écrire les codes des événements.



Crée dans le package « vues » et qui permettra de **réutiliser** c'est fonction déjà défini dans plusieurs vue.

```
package vues;

public interface ViewBase_Ajouter {
    public void ajouter();
}
```

```
package vues;

public interface View_Base_Validation {
    public void Valider();
}
```

Dans la vue « VueCreeTableau » :

```
public void Valider() {
    String nomTableau = SaisienomTableau.getText();
    Tableau tableau = new Tableau(nomTableau, fondTableau: false);
    fenetreCreerTableau.dispose();
    JOptionPane.showMessageDialog(null, "Tableau cree avec succes !");
    VueTableau vt = new VueTableau(tableau, isNights);
}

@Override
public void ajouter() {
    new VueAjouterMembre(fenetreCreerTableau);
}
```

Dans la vue « VueMenuBoard » :

```
@Override
public void ajouter() {
    // TODO Auto-generated method stub
    fenetreMenu.dispose();
    new VueAjoutDescription(fenetreMenu, vt, isNights, descriptionTab);
}
```

Ensuite, nous créons des classes dans le package « controleur » qui vont avoir comme fonction de liée ces événements sur des boutons et de les réaliser.

La classe « `EcouteAjouterActionEvent` » exécutant les méthodes « `ajouter()` » :

```

public class EcouteAjouterActionEvent implements ActionListener {

    ViewBase_Ajouter parent;

    public EcouteAjouterActionEvent(ViewBase_Ajouter parent){
        this.parent = parent;
    }

    @Override
    public void actionPerformed(ActionEvent arg0) {
        // TODO Auto-generated method stub
        parent.ajouter();
    }

}

```

L'ajout de l'action dans le bouton :

```

boutonAjouter.addActionListener(new EcouteAjouterActionEvent(this));

```

Donc, cette méthode, permet aussi d'utiliser **plusieurs fois le même contrôleur** pour **différent action** car le code définit pour chacune des classes est écrit directement dans leur classe.