RAPPORT DE PROJET EN ARCHITECTURE LOGICIELLE

ABDOULAYE SOUMAH
OUSSAMA RADOUCHE

Université de Bordeaux
Avril 2024

1 Présentation du projet

On désire créer un logiciel de dessin vectoriel dont le principe est de créer de nouveaux dessins en utilisant, modifiant, combinant des dessins existants.
Plus précisément, nous voulons pouvoir :

- Sélectionner un objet depuis un menu graphique (toolbar), et le positionner sur notre dessin – (whiteboard) en utilisant le glisser-déposer (drag and drop).
- Créer des groupes d'objets et sous-groupes d'objets, sous-sous-groupes etc...
- Dissocier un groupe d'objet.
- Modifier la taille, position, etc... de nos objets ou groupes d'objets une fois ceux-ci incorporés dans le dessin.
- Ajouter des groupes d'objets ou des objets paramétrés à notre toolbar en les déposants sur la toolbar (drag and drop).
- Annuler ou refaire une opération.
- Sauvegarder un document et charger un document.
- Sauvegarder l'état du logiciel (toolbar) et le recharger au démarrage.

2 Conception

Pour la conception de ce projet nous avons utilités différents design patterns pour modéliser chaque composant de notre Editeur de figure.

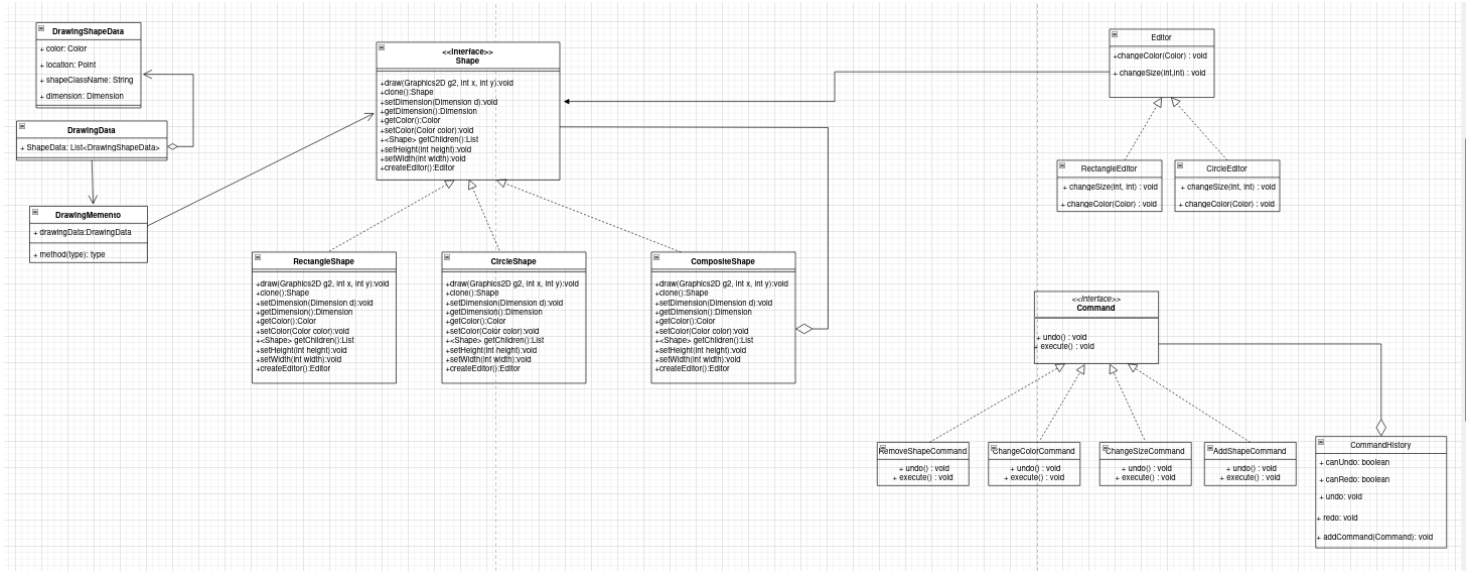

Figure 1: Class Diagram in Landscape

Analysons de près quelques éléments.

2.1 Pattern Command :

Nous voulons que nos actions à éffectuer soient des objets autonomes en contenant les détails de cette action pour ainsi pour l'exécuter ou annuler. Ainsi nous retrouvons des classes suivantes : InterfaceCommand : Une interface fondamentale qui déclare les méthodes execute() et undo(). Ce modèle permet de transformer des actions comme l'ajout ou la suppression de formes en objets exécutables et réversibles, facilitant ainsi les opérations d'annulation. AddShapeCommand: Implémente Command pour ajouter des formes au WhiteBoard. Lors de l'exécution, cette commande ajoute un ShapePanel au Panel spécifié, et inversement pour undo(), en retirant le ShapePanel. RemoveShapeCommand: Parallèlement à AddShapeCommand, cette classe gère la suppression de formes. Elle montre comment les commandes peuvent manipuler les éléments d'interface pour refléter les changements de l'état interne de l'application.

2.2 Pattern Composite:

La classe CompositeShape Implémente l'interface SHAPE, cette classe permet de composer des formes individuelles en un tout unifié. Elle est utilisée pour grouper des formes et les manipuler comme une seule entité, simplifiant la gestion de formes complexes comme les dessins composés de multiples éléments.

2.3 Pattern Memento:

Ce pattern permet de sauvegarder l'état de nos objets mais sans reveler les détails de son implémentation. CommandHistory et DrawingMemento Ces classes sont essentielles pour implémenter l'annulation et la rétablissement des actions. Elles stockent l'état des commandes et permettent de naviguer à travers l'historique des actions effectuées, démontrant une application du modèle memento.

2.4 Pattern Prototype:

Ce pattern nous permet de cloner nos figures de base lors du drag and drop de la toolbar vers le WhiteBoard. Ainsi on pourra le modifier figure dans le WhiteBoard indépendamment du code de base dans la toolbar.

2.5 Factory Method

Lorsqu'on voudra changer les paramètres d'une figures (couleur ou taille) une fois dans le whiteBoard, ce pattern nous sera utilise dans ce cas. Ce pattern nous sera utile dans l'édition des paramètres des figures une fois dans le canvas

2.6 Interface utilisateur:

L'interface utilisateur comprend une barre d'outils permettant de sélectionner le type de figure à dessiner, ainsi que des boutons qui représentent des figures à dessiner et de suppression. La zone de dessin est représentée par un composant WhiteBoard où les figures sont affichées. On y trouve également deux autres boutons qui permettent de sauvegarder ou charger un fichier.

3 Implémentation

Technologies utilisées: Nous avons utilisé le langage java ainsi que sa bibliothèque AWT.

Fonctionnalités implémentées:

- Cliquer-glisser depuis la toolbar vers la WhiteBoard ;
- Éditer la couleur et la taille des figures dans le WhiteBoard ;
- Supprimer un élement du WhiteBoard ;
- Annuler une action ;
- Refaire une action ;
- Sauvegarder l'état de notre éditeur dans un fichier ;
- Charger un fichier un fichier ;

Fonctionnalités non implémentées:

- Regrouper des figures ;
- Dissocier des figures ;
- Cliquer-glisser du whiteBoard vers la toolbar ;

Extraction des données afin de permettre la Sérialisation :
Les éléments graphiques n'étant pas sérialisables, On parcourt notre WhiteBoard pour créer une liste d'informations à enregistrer.

```java
private DrawingData extractDrawingData() {
    List<DrawingShapeData> shapesData = new ArrayList<>();
    for (Component component : canvas.getComponents()) {
        if (component instanceof ShapePanel) {
            ShapePanel shapePanel = (ShapePanel) component;
            Shape shape = shapePanel.getShape();
            Point location = shapePanel.getLocation();
            Color color = shapePanel.getColor();
            Dimension dimension = shape.getDimension();
            shapesData.add(new DrawingShapeData(shape.getClass().getName(), location, color, dimension));
        }
    }
    return new DrawingData(shapesData);
}

private void saveToFile() {
    FileDialog fileDialog = new FileDialog(this, "Save", FileDialog.SAVE);
    fileDialog.setVisible(true);
    String filename = fileDialog.getFile();
    String directory = fileDialog.getDirectory();
    if (filename != null && directory != null) {
        File file = new File(directory + filename);
        try (FileOutputStream fileOut = new FileOutputStream(file);
                ObjectOutputStream out = new ObjectOutputStream(fileOut)) {
            DrawingMemento memento = new DrawingMemento(extractDrawingData());
            out.writeObject(memento);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Figure 2: Extraction des données

Chargement de fichier:

4 Validation

L'application a été testée en utilisant des scénarios d'utilisation simulés, en vérifiant notamment que les figures sont correctement affichées et déplaçables, la sauvegarde et le chargement et que les opérations de modification et de suppression fonctionnent comme prévu.

5 Résultats

L'application fonctionne à peu près comme prévu, permettant aux utilisateurs de créer, modifier et supprimer, d'enregistrer et charger des figures graphiques de manière interactive. Malgré quelques fonctionnements, l'application aurait dû bénéficier d'autres fonctionnalités comme le stipulait le cahier des charges. Malheureusement par contre temps nous n'avons pas pu le faire.

```java
private void loadFile() {
    FileDialog fileDialog = new FileDialog(this, "Load", FileDialog.LOAD);
    fileDialog.setVisible(true);
    String filename = fileDialog.getFile();
    String directory = fileDialog.getDirectory();
    if (filename != null && directory != null) {
        File file = new File(directory + filename);
        try (FileInputStream fileIn = new FileInputStream(file);
                ObjectInputStream in = new ObjectInputStream(fileIn)) {
            DrawingMemento memento = (DrawingMemento) in.readObject();
            loadDrawingData(memento.getDrawingData());
        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
        }
    }
}

private void loadDrawingData(DrawingData drawingData) {
    canvas.removeAll();
    for (DrawingShapeData shapeData : drawingData.getShapesData()) {
        try {
            Class<?> shapeClass = Class.forName(shapeData.getShapeClassName());
            Constructor<?> constructor = shapeClass.getConstructor(int.class, Color.class);
            Shape shape = (Shape) constructor.newInstance(shapeData.getDimension().width, shapeData.getColor());
            ShapePanel shapePanel = new ShapePanel(shape, this, commandHistory, trashButton);
            shapePanel.setColor(shapeData.getColor());
            shapePanel.setSize(shapeData.getDimension());
            shapePanel.setLocation(shapeData.getLocation());
            canvas.add(shapePanel);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    canvas.validate();
    canvas.repaint();
```

Figure 3: Chargement de fichier

6 Conclusion

Le projet d'édition de figures avec Java AWT a été une expérience enrichissante, permettant de mieux comprendre la manipulation des composants graphiques et la gestion des interactions utilisateur en Java ainsi que l'utilisation des modéle de conception.