Explication et utilisation des patrons pour MiniDesign

Polytechnique Montréal

Membre de l'équipe

Jasmin Letendre - 2375392 Alexandre Girouard - 2392299

Date de remise : 1er Décembre 2025

Liste des patrons prévus

1. **Composite**

Intention. Avoir une structure uniforme pour manipuler à la fois les points, les nuages de points et les nuages de nuages.

Application. On définit une interface commune (par ex. Element) avec des opérations comme deplacer(), afficher(), etc. Les points sont des feuilles, les nuages sont des composites qui contiennent une collection d'Element. Ça permet par exemple à la commande de fusion de traiter indifféremment des points ou des nuages.

2. **Strategy**

Intention. Pouvoir changer d'algorithme sans modifier le reste du code, en particulier pour la création des surfaces et pour l'affichage. Application. Une famille de stratégies pour les surfaces (SurfaceCreationStrategy) avec au moins deux implémentations : c1 (ordre des IDs) et c2 (distance minimale). Une famille de stratégies pour l'affichage (AffichageStrategy) avec o1 (affichage par textures) et o2 (affichage par IDs et surfaces).

3. **Decorator**

Intention. Ajouter et retirer des textures à un point de manière dynamique, sans multiplier les sous-classes de points. Application. Un point de base (ex. PointNu) représente la position. Des décorateurs (ex. PointAvecTexture) s'empilent autour du point pour ajouter des textures ('o', '#', etc.). Comme un point peut appartenir à plusieurs nuages, il peut avoir plusieurs décorateurs et donc plusieurs textures.

4. **Command**

Intention. Encapsuler les actions de l'utilisateur et permettre d'implanter proprement undo et redo. Application. Une classe abstraite Commande avec les méthodes execute() et undo(). Des commandes concrètes comme CmdDeplacerPoint, CmdSupprimerPoint, CmdFusionnerNuage contiennent toutes les infos nécessaires pour appliquer et annuler l'action. Un gestionnaire de commandes garde deux piles (undo/redo) pour les touches u et r.

5. **Fabrique**

Intention. Centraliser la création des objets Commande à partir du code saisi par l'utilisateur, au lieu d'avoir un gros switch dispersé. Application. Une classe CommandFactory reçoit le caractère de commande (par ex.

‘d’, ‘s’, ‘f’) et retourne l’objet Commande correspondant. La boucle principale ne connaît que la fabrique et la commande abstraite, ce qui facilite l’ajout de nouvelles commandes.

6. **Façade**

Intention. Séparer la logique de l’application de la fonction main et offrir un point d’entrée simple au reste du système. Application. Une classe MiniDesignApp (ou similaire) sert de façade. Elle contient le modèle (points, nuages, surfaces), le gestionnaire de commandes, les stratégies d’affichage et de création de surfaces. main se contente d’initialiser cette classe avec les points passés en argument et d’appeler une méthode du style executerBoucle().

7. **Patron de méthode**

Intention. Factoriser le “squelette” commun de l’exécution d’une commande, tout en laissant les détails aux sous-classes. Application. Dans la classe abstraite Commande, la méthode publique execute() suit toujours les mêmes étapes (préparation, appel à doExecute(), enregistrement pour undo). Les commandes concrètes redéfinissent seulement doExecute() et doUndo(). On peut appliquer la même idée côté affichage si nécessaire (préparer la grille, remplir, afficher).

Diagramme de classe

Voici le diagramme de classe que nous avons fait pour la conception de notre application: