TP C++ 4 : Gestionnaire de formes

Document de conception

B3334 : VICENTE Olivier, SORIN Nicolas

Rendu : Vendredi 5 février 2016

Introduction

Ce programme a pour but de gérer une collection de formes géométriques de différents types. Il doit permettre d'ajouter des rectangles, segments, polygones convexes et des formes composées d'autres formes. Diverses opérations peuvent être réalisées sur ces formes (voir cahier des charges). Le programme s'utilise en ligne de commande.

Définition du vocabulaire

— Object : Classe générique désignant une forme. Toutes les classes spécifiques en héritent.
— Command : Classe abstraite générique désignant une commande *qui modifie le modèle*. Les commandes informatives (qui ne modifient pas le modèle) n'ont pas de classe associée.
— Controller : Partie du programme qui gère les entrées/sorties et interprète les commandes.
— Model : Partie du programme qui stocke les objets et gère la mémoire. Le modèle est composé d'un dictionnaire d'objets.

Choix généraux

Afin de faciliter l'ajout de nouvelles fonctionnalités, nous avons opté pour le design pattern MVC. Dans ce cas précis, notre programme ne comporte qu'un contrôleur et un modèle, puisqu'il n'y a aucun élément d'affichage complexe. Le contrôleur se charge donc de renvoyer les sorties, qui sont très simples. Nous avons donc implémenté deux types d'objets.

Le contrôleur est composé de la fonction main (interprétation basique des commandes) et de toutes les classes héritant de Command. Une fois l'entrée de l'utilisateur interprétée, deux scénarios sont possibles suivant la commande. Si la commande ne modifie pas le modèle, on appelle simplement une fonction du contrôleur (indépendante de toute classe). Si la commande modifie le modèle, on instancie un objet du type correspondant, on exécute la commande, qui est stockée dans une file double en cas de succès. En cas d'échec, la commande est supprimée. Chacune de ces commandes reçoit une référence vers le modèle pour y effectuer des modifications. Ces modifications peuvent être annulées via la fonctionnalité UNDO et rétablies via la fonctionnalité REDO. Jusqu'à 20 commandes peuvent être stockées dans la file double, et donc annulées. Il n'est possible d'annuler qu'une seule commande.

Le modèle est composé d'un dictionnaire contenant des objets héritant de Object. La mémoire pour ces objets est allouée dynamiquement afin de limiter au maximum les copies pour une gestion de la mémoire au plus juste. Les objets sont manipulés par les commandes du contrôleur.

Spécifications des classes

Dans cette section seront détaillés l'utilisation des méthodes et attributs des différentes classes. Du fait de l'héritage, les classes ont un fonctionnement normalisé.

Command

Rôle

Les classes héritant de Command ont pour rôle de manipuler le modèle, tout en étant stockable pour pouvoir être annulées si besoin. Elles ont une structure commune.

Attributs communs

- *bool undone :* Indique quelle est la dernière méthode appelée (Do ou Undo). Cette information est nécessaire pour la gestion de la mémoire lors de la destruction de certianes commandes.

Méthodes communes

- *bool Do(map<string, Object*> & model) :* Exécute la commande et modifie le modèle si la commande est valide. Renvoie *true* en cas de succès, *false* en cas d'échec.
- *bool Undo(map<string, Object*> & model) :* Annule les changements effectués par Do et rétablit l'état du modèle précédent l'exécution de la commande. Renvoie *true* en cas de succès, *false* en cas d'échec.

Attributs spécifiques aux classes filles

- *bool invalid :* Dans le cas des commandes d'ajout d'objets, stocke si l'objet construit est invalide (et donc inapte à être ajouté au modèle)
- *map<string, Object*> mapObjects :*Dans le cas des commandes impliquant des suppressions d'objets, stocke temporairement les adresses des objets supprimés pour pouvoir les rétablir en cas d'appel de Undo ou les supprimer définitivement lorsque la commande ne peut plus être annulée.
- *Object* object:* Dans le cas des commandes d'ajout d'objets, stocke l'adresse de l'objet pour pouvoir ensuite l'ajouter au modèle ou le retirer sans perte de mémoire.

Object

Rôle

La classe Object permet de stocker un objet et ses attributs, et d'effectuer des opérations simples sur cet objet (déplacement, vérification de l'appartenance d'un point à l'objet).

Attributs communs

- *string name :* Nom de l'objet.

Méthodes communes

- *bool Contains(Point & p) const :* Renvoie *true* si le point est appartient à l'objet, *false* sinon.
- *void Move(int dx, int dy) :* Déplace l'objet suivant les données indiquées en paramètres.
- *string GetName() const :* Renvoie le nom de l'objet.
- *ostream & doPrint(ostream & os) :* Utilisée lors de la surcharge de l'opérateur << (l'héritage empêchant une implémentation directe de cette surcharge)
- *Object * clone() :* Utilisée pour effectuer une copie dynamique d'un objet (nécessaire pour les objets complexes).
- *ostream & operator<<(ostream & stream, Object & object) :* Surcharge de l'opérateur << faisant appel à la méthode doPrint, spécifique à chaque objet. Utilisée pour les commandes LIST et SAVE.

Attributs spécifiques aux classes filles

- *vector<Object *> tabObjects :* Dans le cas des objets complexes, vecteur des objets contenus dans l'objet complexe.
- *Point * tabPoints :* Dans le cas des objets simples, tableau des points constituant l'objet.
- *unsigned int nPoints :* Dans le cas des objets simples, taille du tableau de points. (Cette taille est fixée à 2 pour les segments et rectanles)

Méthodes spécifiques aux classes filles

- *vector<Object *> GetTabObjects :* Dans le cas des objets complexes, renvoie le vecteur des objets contenus dans l'objet complexe.
- *void AddObject(Object * o) :* Dans le cas des objets complexes, ajoute une copie d'un objet au vecteur des objets contenus dans l'objet complexe.
- *void ClearObjects() :* Dans le cas des objets complexes, supprime les objets contenus dans l'objet complexe.
- *Point * GetTabPoints :* Dans le cas des objets simples, renvoie le tableau des points constituant l'objet.
- *unsigned int GetSize()* : Dans le cas des objets simples, taille du tableau de points. (Cette taille est fixée à 2 pour les segments et rectanles)

Types Dépendants

— *map<string, Object* >::iterator iter_ref :* itérateur qui permet de parcourir le modèle.

Point

Rôle

La classe Point contient les coordonnées d'un point et permet son déplacement. Elle est utilisée comme composante des formes simples et dans le cas de la commande HIT.

Attributs

- *int x :* Coordonnée horizontale du point.
- *Int y :* Coordonnée verticale du point.

Méthodes

- *int GetX() :* Renvoie la coordonnée horizontale du point.
- *int GetY() :* Renvoie la coordonnée verticale du point.
- *void SetX(int x) :* Modifie la coordonnée horizontale du point.
- *void SetY(int y) :* Modifie la coordonnée verticale du point.
- *void Move(int dx, int dy) :* Déplace le point suivant un vecteur (dx, dy).