EL ALAOUI TALIBI Salma **BinômeB3122**

RAZANAJATOVO Xavier

TP C++ n°3: Héritage
Document de Conception

I. Présentations des fonctionnalités de notre application

L'application *Geometric* est un éditeur qui permet de créer des formes géométriques et de les manipuler dans un modèle. L'interaction avec l'éditeur se fait en mode console.

1. **Manipulations possibles sur les éléments :**

- **Ajout :**

  TypeObjet NomObjet ParamètresObjet

Permet de créer une forme géométrique *NomObjet* de type *TypeObjet* qui sera ajouté au modèle courant.
Le *NomObjet* doit être unique pour un modèle donné.
Si le nom existe déjà, l'ajout ne sera pas effectué, et une erreur sera renvoyée.
Si les paramètres sont incorrects, l'ajout ne sera pas effectué et une erreur sera envoyée.
La réponse est OK si l'ajout s'est bien exécuté et ERR sinon.

- **Suppression :**

  DELETE NomObjet1...NomObjetN

Permet de supprimer les objets spécifiés.
Si un nom est invalide, aucun objet ne sera supprimé, et une erreur est renvoyée.

- **Déplacement :**

  MOVE NomObjet X Y

Permet de déplacer l'objet *NomObjet* de X sur l'axe x et de Y sur l'axe y.
Si *NomObjet* est invalide une erreur est renvoyée.
Si *X* et/ou *Y* ne sont pas des entiers, une erreur est renvoyée.

2. **Manipulation permettant la gestion du modèle :**

- **Énumération :**

  LIST

Permet d'afficher la description des objets existants. La description d'un objet est la commande nécessaire pour le construire. L'affichage se fait par ordre alphabétique des noms d'objet.

- **Charger en mémoire :**

  LOAD fileName

Charge un ensemble d'objets à partir d'un fichier dont le format est le suivant :
Chaque ligne décrit un objet ou est un commentaire.
Une ligne est un commentaire si le premier caractère est le #.
La description d'un objet est la commande qui permet de le créer.
Si une commande présente dans le fichier est invalide, le chargement du fichier est entièrement annulé et une erreur est renvoyée.

- **Sauvegarder :**

  SAVE fileName.txt

Sauvegarde le modèle courant dans le fichier *fileName.txt.*

- **Vider le modèle :**

  CLEAR

Supprime tous les objets composant le modèle actuel.

- **Fermer l'application :**

  EXIT

Ferme l'application.

3. **Gestion de l'historique :**

- **Annuler la dernière opération :**

  UNDO

Annule la dernière opération qui a eu un effet sur le modèle (déplacement, suppression, insertion d'un objet, chargement d'un fichier).

- **Reprendre la dernière modification :**

  REDO

Refaire la dernière opération annulée qui a eu effet sur le modèle. La commande *REDO* a un effet sur le modèle s'il y a eu une commande *UNDO* précédemment et entre cette commande *UNDO* et le moment d 'exécution de la commande *REDO* il n'y a pas eu d'autres commandes qui changent le modèle.

4. **Formes géométriques gérées :**

- **Cercle :**

-Un cercle possède un centre (X1, Y1) et un rayon R qui peut être positif ou nul.
-Dans le cas où le rayon R est nul, la commande revient à la création d'un point.

- **Rectangle :**

-Un rectangle est défini par deux points (X1,Y1) et (X2,Y2).
-Les deux points définissant un rectangle peuvent être superposés.

- **Ligne :**

-Une ligne est un segment défini par deux points (X1,Y1) et (X2,Y2).
-Les deux points définissant une ligne peuvent être superposés.

- **Poly-ligne :**

-Une poly-ligne est définie par les segments (X1,Y1)(X2,Y2), (X2,Y2)(X3,Y3) . . . (Xn-1,Yn-1)(Xn,Yn).
-Une poly-ligne peut être composée d'un seul point (X1,Y1).

- **Objet agrégé :**

-Un objet agrégé est formé de plusieurs objets existants, identifiés par leurs noms.
-Un OA ne peut pas être crée vide.
-Un OA peut se retrouver vide suite à la suppression des éléments qui le composent.
-Un OA peut être composé de plusieurs fois le même élément si l'élément existe bien.

5. **Cas d'utilisation :**

- **Cas d'erreur :**

-Non respect de la casse (majuscules) dans le type de commande (C ,R ,L ,PL ,OA ,DELETE ,CLEAR ,LOAD ,SAVE ,MOVE , LIST ,UNDO ,REDO ,EXIT).
- Ajout d'un cercle (commande C) ne respectant pas la syntaxe *C Name X1 Y1 R.*
- Ajout d'un cercle avec un rayon négatif (R<0).
- Ajout d'un rectangle (commande R) ne respectant pas la syntaxe *R Name X1 Y1 X2 Y2.*
- Ajout d'une ligne (commande L) ne respectant pas la syntaxe *L Name X1 Y1 X2 Y2.*
- Ajout d'une poly-ligne (commande PL) ne respectant pas la syntaxe *PL Name X1 Y1 X2 Y2 . . . Xn Yn.*
- Ajout d'un objet agrégé (commande OA) ne respectant pas la syntaxe *OA Name Name1 Name2 . . . NameN.*
- Ajout d'un objet agrégé contenant un ou plusieurs objet(s) inexistant(s).
- Ajout d'un objet agrégé ne contenant aucun objet.
- Suppression (commande DELETE) d'un ou plusieurs objet(s) inexistant(s).
- Déplacement (commande MOVE) d'un objet inexistant.
- Chargement (commande LOAD) d'un fichier inexistant.
- Chargement (commande LOAD) d'un fichier contenant des éléments qui n'existent ni dans le

modèle courant, ni précédemment dans le fichier à charger).
-Commande UNDO lorsqu'il n'y a aucune commande à défaire.
-Commande REDO lorsqu'il n'y a aucune commande à refaire.

- **Cas limites :**

- Ajout d'un cercle de rayon R=0 => on obtient un point.
- Ajout d'un rectangle avec deux points de mêmes coordonnées (*R Name X1 Y1 X1 Y1*) => on obtient un point.
- Ajout d'une ligne avec deux points de mêmes coordonnées (*L Name X1 Y1 X1 Y1*) => on obtient un point.
- Ajout d'une poly-ligne ne contenant que deux points (*PL Name X1 Y1 X2 Y2*) => on obtient une ligne.
- Ajout d'une poly-ligne contenant des points de mêmes coordonnées (*PL Name X1 Y1 X2 Y2 X2 Y2 . . . Xn Yn*).
-Ajout d'un OA contenant plusieurs fois le même élément géométrique (*OA Name EG1 EG1*) => revient à créer un OA contenant l'élément géométrique EG1.

II. Architecture du programme

Nous avons organisé notre application de la manière suivante pour limiter le couplage, en suivant le design pattern Commande :

- **Couche pour la gestion des éléments géométriques :**

Contient les classes permettant de créer les différentes formes géométriques supportées par notre application, et d'implémenter les différentes opérations qui leurs sont associées.

- **Gestion du modèle :**

Contient la classe modèle, qui représente le modèle courant, et contient la collection des éléments géométriques présents dans le modèle. Cette classe nous permet d'implémenter également toutes les manipulations qu'on pourra faire sur notre modèle, sans avoir à passer par les méthodes de chaque éléments géométriques.

- **Gestion des commandes :**

Contient les différentes classes qui représentent chacune des actions possibles, permettant l'annulation et le rétablissement des actions quand il s'agit d'une commande « historizable », ainsi que le commande manager qui permet de gérer l'historique,

- **Application principale :**

Contient l'application principale qui permet de traiter les commandes reçues et crée la commande adéquate, puis appelle les méthodes du CommandManager correspondantes s'il s'agit d'une annulation ou d'un rétablissement.

(cf Diagrammes UML: "TP-H - UML1.jpg" et "TP-H - UML2.jpg" dans le répertoire doc)

III. Spécifications des classes

- **EltGeo**
  Classe abstraite
  Cette classe représente un élément géométrique. Elle implémente la méthode déplacer qui est virtuelle et qui sera spécifiée par chacun des élément géométrique concrets (Rectangle, Ligne, Polyligne, Cercle,ObjAgr).

- **ObjAgr**
  Hérite de EltGeo
  Cette classe représente un agrégat d'éléments géométriques. Elle contient une map de pointeurs sur EltGeo permettant qui a comme clé leur nom, permettant l'unicité sur le nom. Cette classe implemente la méthode déplacer qui permet de déplacer tous les éléments géométriques simples contenus dans l'objet agrégé(qui peut aussi contenir des objets agrégés).

- **Modèle**
  Cette classe représente le modèle courant. Elle contient une map d'éléments géométriques qui pointe vers tous les éléments géométriques contenus dans notre modèle(qu'il s'agisse d'objets agrégés ou non). Elle contient également une map qui pointe uniquement sur les objets agrégés, et un vecteur de noms d'éléments qui nous permet de garder l'ordre chronologique d'ajout des éléments géométriques. Elle contient les méthodes permettant d'ajouter, de supprimer, de déplacer un élément géométrique du modèle.

- **Commande**
  Classe abstraite
  Cette classe permet de représenter une commande de notre application, càd les différentes actions qu'elle permet de réaliser. Elle implémente la méthode Execute qui sera par la suite spécialisée dans chaque commande concrète. C'est dans la méthode Executer des ajout qu'on crée les différents éléments géométriques.

- **CommandeHis**
  Classe abstraite, hérite de Commande
  Cette classe permet de représenter les commandes qui permettent des undo et des redo. En plus de la méthode Executer, elle implemente les méthode Undo et Redo.

- **CommandManager**
  Cette classe permet de stocker les commandes qui ont été exécutées dans le but de pouvoir les annuler à l'aide de la commande UNDO et éventuellement de les refaire à l'aide de la commande REDO.
  Pour cela, elle contient deux vecteurs undoCommandes et redoCommandes qui contiennent des pointeurs vers les commandes (de type CommandeHis).
  Lorsqu'une commande est exécutée (avec succès), elle est ajoutée à undoCommandes, les commandes s'empilent donc dans le vecteur et lorsqu'une commande UNDO est lancée, la dernière commande ajoutée est annulée à l'aide de la méthode Undo() correspondant au type de commande. Cette commande est ensuite supprimée de

undoCommandes et ajoutée à redoCommandes. Le même principe est appliqué lorsqu'une commande REDO est lancée. A noter: lorsqu'une commande est executée avec succès, toutes les commandes présentes dans redoCommandes sont supprimées.

- **Application**
  Cette classe permet de récupérer la commande entrée par l'utilisateur.
  Cette commande est ensuite traitée:
  -Vérification de la validité de la commande
  -Vérification de la validité des paramètres en fonction de la commande demandée (nombre et types)
  Si toutes les vérifications se sont bien passées, la commande est créée, exécutée, puis ajoutée à la classe CommandesManager si elle est historizable.

IV. Tests :

Voici un tableau permettant de référencer les différents tests que nous avons a mis en place et qui se trouvent dans le répertoire tests.

TESTS	DESCRIPTIONS
Test1	Tests sur l'ajout de cercles.
Test2	Tests sur l'ajout de rectangles.
Test3	Tests sur l'ajout de lignes.
Test4	Tests sur l'ajout de poly-lignes.
Test5	Tests sur l'ajout d'objets agrégés.
Test6	Tests sur la suppression simple (sans objets agrégés) d'un seul élément.
Test7	Tests sur la suppression simple (sans objets agrégés) de plusieurs éléments.
Test8	Tests sur la suppression simple (sans objets agrégés) d'un seul élément. (Avec erreurs de syntaxe et éléments inexistants)
Test9	Tests sur la suppression simple (sans objets agrégés) de plusieurs éléments. (Avec erreurs de syntaxe et éléments inexistants)
Test10	Tests sur la suppression d'objets agrégés et d'éléments composant des objets agrégés.
Test11	Tests sur le déplacement d'éléments (sauf objets agrégés).
Test12	Tests sur le déplacement d'éléments et d'objets agrégés simples (qui ne sont pas composés d'autres objets agrégés).
Test13	Tests sur le déplacement d'éléments et d'objets agrégés complexes (composés d'autres objets agrégés).
Test14	Tests de la commande LOAD sans objets agrégés (fichiers avec et sans erreurs) avec utilisation de la commande CLEAR.
Test15	Tests de la commande LOAD avec des objets agrégés (fichiers avec et sans erreurs) avec utilisation de la commande CLEAR.
Test16	Tests de la commande SAVE.

Test17	Tests de la commande UNDO sur des ajouts d'éléments géométriques (hors objets agrégés).
Test18	Tests des commandes UNDO/REDO sur des ajouts d'éléments géométriques (avec objets agrégés).
Test19	Tests des commandes UNDO/REDO sur des commandes LOAD.
Test20	Tests des commandes UNDO/REDO sur des commandes MOVE.
Test21	Tests des commandes UNDO/REDO sur des commandes MOVE avec des éléments créés a partir d'un fichier chargé.
Test22	Tests des commandes UNDO/REDO sur des commandes DELETE.
Test23	Tests des commandes UNDO/REDO sur la commande CLEAR.
Test24	Tests de 20 commandes UNDO d'affilée.