TP Héritage, polymorphisme

Objectif : Familiarisation avec les notions d'héritage entre classes C++, polymorphisme, méthodes virtuelles, classe abstraite, opérations d'entrée/sortie. D'autres notions seront revisitées dans ce TP : l'utilisation des "design patterns", la mise au point d'un logiciel, l'évaluation des performances d'un logiciel, respecter un cahier des charges, intégration avec des modules existants.

Problème

Vous devez implémenter un éditeur de formes géométriques et de les manipuler simplement. Nous ne vous demandons pas de réaliser une interface graphique, l'interaction avec l'éditeur se fera en mode console. L'éditeur doit permettre la gestion de formes géométriques suivantes :

- Segment
- Rectangle
- Polygone convexe

Il permet également d'effectuer :

- L'ajout d'un nouvel objet (avec l'une de formes présentées précédemment) ;
- La création d'un nouveau objet comme résultat d'une opération sur des objets existants (réunion, intersection) ;
- La suppression d'un objet ;
- Le déplacement d'un objet ;
- Vérifier si un point se trouve à l'intérieur d'un objet ;
- La persistance d'un ensemble d'objets construit de cette manière ; le fichier de sauvegarde est un fichier de format texte ;

Pour des raisons d'intégration avec d'autres modules existants, votre application doit implémenter d'une manière très stricte l'interface en mode console en respectant la syntaxe illustrée dans l'annexe A.

Méthode de travail

D'une manière très schématique, nous vous conseillons de diviser votre travail de la façon suivante :

- Compréhension du cahier des charges (CdC) ; éclaircissement des points ambigüs ; vous devez produire un CdC détaillé à partir du CdC initial ;
- Concevoir une solution ; identifier les classes, les méthodes et les attributs ; Produire un document qui spécifie votre future solution, incluant un diagramme de classes ;
- Préparer vos données de test ; éventuelle mise en place d'une procédure de non-régression ;
- Implémenter votre solution ; Documentation de votre code ; Respect d'un guide de style ; Réaliser des tests unitaires pour vos classes ; Paralléliser éventuellement votre travail ; un planning très schématique peut vous aider ;
- Tester votre solution via des tests fonctionnels ;
- Évaluer les performances de votre solution ;
- Préparer correctement les livrables à rendre avant la date limite qui vous sera communiqué en séance.

N'oubliez pas de :

- Relire vos documents d'une manière croisée ;
- Vérifier la validité de votre solution.

Livrables

Tous les livrables seront déposés dans un fichier .zip sur moodle2.insa-lyon.fr, dans l'activité associée au cours *Algorithmique, Programmation Orientée Objet - C++*. Le fichier aura le nom B3XYY.zip, où X est votre groupe, YY votre numéro de binôme.

Le fichier B3XYY sera organisé de la manière suivante :

- Exécutable avec le nom B3XYY, dans le répertoire racine (/B3XYY) ;
- Sources de votre projet, permettant de reconstruire l'exécutable (sur une machine Linux du département), dans le sous-répertoire src/
- Document de conception ($\leq$ 4 pages) : Conception_B3XYY.pdf, dans le sous-répertoire doc/
- Cahier des charges détaillé ($\leq$ 6 pages) : CDC_B3XYY.pdf, dans le sous-répertoire doc/
- Procédure de non-régression : fichiers de données + script dans le sous-répertoire tests/
- Document évaluant les performances de votre application, en principal le temps de lecture/écriture d'un fichier : Perfs_B3XYY.pdf, dans le sous-répertoire doc/
- Toute autre information utile : dans un fichier readme.txt situé dans le répertoire racine.

Notation

Pour simplifier la validation de votre application, vous allez recevoir quelques données pour tester le format des entrées/sorties. Après avoir livré votre solution, vous allez recevoir les tests "officiels" que vous pouvez exécuter vous-mêmes sur votre solution. La notation se fait de la manière suivante :

1. (0 pt) Votre livrable est arrivé à temps, en s'exécutant correctement (sans erreur de compilation etc.) :
   (a) OUI : goto 2
   (b) NON : le TP n'est pas validé ;
2. (10 pts) Une batterie de tests est appliquée à votre application.
   (a) Plus de 75% de tests sont passés avec succès pour les objets de type rectangle et segment : goto 3
   (b) Sinon : le TP n'est pas validé ;
3. (3 pts) Respect d'un guide de style, qualité du code, gestion correcte de la mémoire etc. ;
4. (2 pts) Le cahier des charges détaillé ;
5. (2 pts) Document décrivant la conception de votre application ;
6. (1 pt) Document (1 page) décrivant l'évaluation des performances ;
7. (2 pts) Procédure de non-régression via des tests fonctionnels ;

Annexe A

L'interaction avec cette application se fait en mode console. L'application reçoit des commandes en entrée (commande = chaine des caractères terminée avec le symbole de fin de ligne). Après la réception d'une commande, l'application peut répondre avec un texte (qui peut être vide). L'application peut afficher à tout instant (console) de lignes des caractères qui commencent avec le caractère #. Ces lignes sont considérées comme des commentaires, et n'ont pas d'influence sur le fonctionnement des entrées/sorties.

Commandes possibles

Ajouter un segment

Commande :

```
S Name X1 Y1 X2 Y2
```

Réponse :

```
[OK|ERR]
```

Description : Ajoute un segment entre les points (X_1, Y_1), (X_2, Y_2). L'objet à un nom (*Name*), qui est un mot composé de lettres et/ou chiffres, sans séparateurs à l'intérieur. Toutes ces valeurs sont des entiers. Dans ce document tous les paramètres sont des entiers, sauf si nous spécifions explicitement un autre type. La réponse est *OK* si la commande s'est bien exécutée, *ERR* dans le cas contraire. La réponse peut s'accompagner d'un commentaire (ligne qui commence avec le caractère #).
Exemple :

```
C: S c11 12 15 45 20
R: OK
R: #New object: c11
```

Exemple :

```
C: S 12 15
R: ERR
R: #invalid parameters
```

Ajouter un rectangle

Commande :

```
R Name X1 Y1 X2 Y2
```

Réponse : voir le point précédent.
Description : Ajoute un rectangle défini par les deux points gauche-haut = (X1, Y1) et droite-bas = (X2, Y2).
Exemple :

```
C: R rectangle1 56 46 108 4536
R: OK
```

Ajouter un polygone convexe

Commande :

```
PC Name X1 Y1 X2 Y2 ... Xn Yn
```

Description : Ajoute un polygone défini par les points : $(X_1, Y_1), (X_2, Y_2), (X_3, Y_3) \ldots (X_n, Y_n)$, avec $n \geq 3$. Si le polygone n'est pas convexe, une erreur sera générée.
Exemple :

```
C: PC Name12 56 46 108 4536 80 100
R: OK
```

Opération de réunion

Commande :

```
OR Name Name1 Name2 ... NameN
```

Description : Permet de construire un nouveau objet Name comme la réunion de la liste d'objets existants $Name_1 \ldots Name_n$.
Exemple :

```
C: PC Name12 56 46 108 4536 80 100
R: OK
C: S seg1 56 46 108 4536
R: OK
C: OR re1 seg1 Name12
R: OK
```

Opération d'intersection

Commande :

```
OI Name Name1 Name2 ... NameN
```

Description : Permet de construire un nouveau objet Name via l'intersection de la liste d'objets existants $Name_1 \ldots Name_n$.
Exemple :

```
C: PC Name12 56 46 108 4536 80 100
R: OK
C: S seg1 56 46 108 4536
R: OK
C: OI int1 seg1 Name12
R: OK
```

Opération d'appartenance

Commande :

```
HIT Name X Y
```

Réponse :

```
[YES|NO]
```

Description : Permet de vérifier si le point (X, Y) se trouve à l'intérieur de l'objet *Name*.
Exemple :

```
C: R rectangle1 56 46 108 4536
R: OK
C: HIT rectangle1 57 50
R: YES
```

Suppression

Commande :

```
DELETE Name1 Name2 ... NameN
```

Description : Supprime les objets identifiés. Si un nom est invalide, aucun objet n'est supprimé, et une erreur est renvoyé.
Exemple :

```
C: DELETE Name12 l172v
R: OK
```

Déplacement

Commande :

```
MOVE Name dX dY
```

Description : Déplace l'objet *Name* avec dX sur l'axe x et dY sur l'axe y.
Exemple :

C: MOVE Name12 100 −25
R: OK

Énumération

Commande :

LIST

Réponse :

Desc1
Desc2
...
DescN

Description : Affiche les descripteurs d'objets existants, dans un format à définir.

Annuler la dernière opération

Commande :

UNDO

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

C: UNDO
R: OK

Reprendre la dernière modification

Commande :

REDO

Description : Refaire la dernière opération annulée qui a eu un 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.
Exemple :

C: REDO
R: OK

Charger en mémoire un modèle

Commande :

```
LOAD filename
```

Description : charge un ensemble d'objets à partir d'un fichier texte ; le format du fichier est à définir.
Exemple :

```
C: LOAD a.txt
R: OK
```

Sauvegarder le modèle courant

Commande :

```
SAVE filename
```

Description : sauvegarde le modèle courant dans un fichier ; le format du fichier est décrit au point précédent (LOAD).
Exemple :

```
C: SAVE a.txt
R: OK
```

Vider le modèle actuel

Commande :

```
CLEAR
```

Description : supprime tous les objets composant le modèle actuel.
Exemple :

```
C: CLEAR
R: OK
```

Fermer l'application

Commande :

```
EXIT
```

Description : ferme l'application.
Exemple :

```
C: EXIT
R:
```