

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 :

- Cercle
- Rectangle
- Ligne
- Poly-ligne

Il permet également d'effectuer :

- L'ajout d'un nouvel objet (avec l'une de formes présentées précédemment) ;
- La sélection de plusieurs objets ;
- La suppression d'un ou plusieurs objets ;
- Le déplacement d'un ou plusieurs objets ;
- 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 ambigus ; Vous pouvez 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

Vous devez fournir les fichiers suivants, organisés dans le répertoire BXXX qui vous sera précisé en séance :

- Exécutable avec le nom B3XXX, dans le répertoire racine (/BXXX) ;
- Sources de votre projet, permettant de reconstruire l'exécutable (sur une machine Linux du département), dans le sous-répertoire src/
- Document de conception (≤ 6 pages) : Conception_B3XXX.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_B3XXX.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. (11 pts) Une batterie de tests est appliquée à votre application.
 - (a) Plus de 50% de tests sont passés avec succès : 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) Document décrivant la conception de votre application ;
5. (1 pt) Document (1 page) décrivant l'évaluation des performances ;
6. (2 pts) Procédure de non-régression via des tests fonctionnels ;
7. (1pt) Application correcte de la batterie de tests (point 2 ; attention, les tests vous les recevez après avoir rendu le TP !) sur votre application et estimation correcte du résultat ; ce résultat vous allez le communiquer par mail à : Marian.Scuturici@insa-lyon.fr et Youakim.Badr@insa-lyon.fr et nous allons le comparer avec notre résultat :
 - (a) valeur identique : 1 pt
 - (b) valeur différente : 0 pt

Annexe A

L'interaction avec cette application se fait en mode console. L'application reçoit des commandes en entrée (commande = chaîne 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 cercle

Commande :

C Name X1 Y1 R

Réponse :

[OK|ERR]

Description : Ajoute un cercle avec un centre (X1,Y1) et un rayon R. 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: C c11 12 15 45
R: OK
R: #New object: c11
```

Exemple :

```
C: C 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 (X1, Y1) et (X2, Y2).

Exemple :

C: R rectangle1 56 46 108 4536

R: OK

Ajouter une ligne

Commande :

L Name X1 Y1 X2 Y2

Description : Ajoute un segment défini par les deux points (X1, Y1) et (X2, Y2).

Exemple :

C: L l172v 56 46 108 4536

R: OK

Ajouter une poly-ligne

Commande :

PL Name X1 Y1 X2 Y2 ... Xn Yn

Description : Ajoute une poly-ligne définie par les segments : (X1,Y1), (X2,Y2), (X2,Y2), (X3,Y3) ... (Xn-1,Yn-1), (Xn,Yn), avec $n \geq 1$.

Exemple :

C: PL Name12 56 46 108 4536 80 100

R: OK

Sélection

Commande :

S Name x1 y1 x2 y2

Description : Sélectionne les objets se trouvant dans le rectangle défini par les deux points. Une sélection à un nom, et plusieurs sélections peuvent coexister au même moment. Une sélection n'est pas persistante (il n'y a pas de sauvegarde pour la sélection).

Exemple :

```

C: PL Name12 56 46 108 4536 80 100
R: OK
C: L 1172v 56 46 108 4536
R: OK
C: R rectangle1 56 46 108 4536
R: OK
C: S sel1 0 0 10000 10000
R: OK

```

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. Le suppression d'une sélection entraine la suppression des objets la composant, et il n'y a pas d'erreur si un objet de la sélection est invalide.

Exemple :

```

C: DELETE Name12 1172v
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, un par ligne ; le descripteur d'un objet est la commande nécessaire pour le construire. L'ordre d'affichage est l'ordre alphabétique des noms d'objet. La commande LIST n'affiche pas les sélections.

Exemple :

```
C: LIST
R: PL Name1 56 46 108 4536 80 100
R: L Name2 56 46 108 4536
R: L Name23 -52 46 108 4536
```

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). Une sélection n'est pas "undo-able".

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 ; le format du fichier : 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 c'est la commande qui permet de le créer.

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, incluant les sélections.

Exemple :

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

Fermer l'application

Commande :

```
EXIT
```

Description : ferme l'application.

Exemple :

```
C: EXIT
R:
```