TP3 : Éditeur – Figures Géométriques

Dans ce TP, nous allons travailler sur la modélisation d'un éditeur (non graphique…) qui va permettre la manipulation simple de formes géométriques 2D, et leur export vers un fichier SVG pour la visualisation.

On vous demande donc de proposer une hiérarchie de figures géométriques dans laquelle figureraient les Cercles, les Rectangles et les Carrés. Définissez les attributs nécessaires et spécifiques à chaque classe, notamment pour le stockage de vos figures, ainsi que le calcul de l'aire de chacune de ces figures.

Pour la position du centre de la Figure, Vous pourrez par exemple vous aider d'une struct/classe Point pour laquelle les attributs x et y sont publics et donc directement accessibles.

Constructeur et Héritage

Mettre en place les différents éléments dont vous aurez besoin pour les figures, pensez bien à créer 2 fichiers (.h et .cpp) pour chaque classe. Rajoutez les accesseurs/mutateurs nécessaires.

1. Quel est le statut de la classe Figure Géométrique ? Et de quel type d'héritage s'agit-il pour les classes filles ?
2. Pour la classe Mère et chaque classe Fille, écrire un constructeur par défaut (par exemple Carré de centre (0,0) et de côté 1). Rajouter les destructeurs pour toutes les classes de la hiérarchie, même s'ils ne sont pas forcément nécessaires, et rajoutez des traces d'exécution.
3. Testez la création d'un objet simple, et vérifiez la bonne création et destruction des objets :2 appels à des constructeurs, et 2 appels à des destructeurs par objet. Dans quel ordre ? Commentez.
4. Rajoutez maintenant les constructeurs avec paramètres. Faites appel au constructeur de la classe mère pour les attributs communs (la position du centre de la figure, par exemple). Comment faire ? Comme pour la question 2, validez le cycle de vie de vos objets.

Héritage et Surcharge

Nous allons maintenant calculer l'aire des figures géométriques.

1. Rajoutez une méthode pour chaque classe Fille. Un prototype possible serait `virtual float getAire() override`, commentez. Testez dans le main sur des figures de différentes formes, tailles et positions.
2. Peut-on calculer l'aire d'un objet Figure Géométrique (classe mère) ? Par exemple, adaptez à votre projet et testez le code suivant :

FG forme= Cercle() ;

cout << forme.getAire() ;

Que faudrait-il changer pour que cela fonctionne ?

Classe Éditeur

Nous allons maintenant créer une classe pour manipuler les Figures Géométriques.

1. Écrire la classe Éditeur, qui contient une liste (std::list<FigureGeometrique*>) de Figures. Composition ou Agrégation ?
2. Écrire une méthode *addFG(FigureGeometrique*)* qui ajoute une figure en fin de liste ; Créez un certain nombre de figures et ajoutez-les à votre Éditeur. Vérifiez aussi la bonne destruction de vos objets lors de la suppression de l'Éditeur (Composition) ou fin de programme (Agrégation).

Polymorphisme et Héritage

Nous allons maintenant commencer à travailler sur des manipulations simples des Figures Géométriques (FG) à travers l'éditeur.

1. Écrire une fonction qui calcule l'aire totale des FG d'un Éditeur. Quel est ici l'intérêt d'avoir une liste de pointeurs sur FG ? Quel mécanisme est mis en jeu ?
2. Écrire la surcharge de l'opérateur<< pour les FG. Tester pour une classe Fille. Pourquoi le mécanisme précédent ne fonctionne-t-il pas si on veut afficher toutes les FG de l'éditeur avec l'opérateur<< de FG ? Comment contourner ce problème ?

Export SVG

Nous allons sauvegarder les figures dans un fichier au format SVG, qui permettra de visualiser vos œuvres dans Inkscape !

1. Commencez par étudier le format SVG, par exemple : http://tutorials.jenkov.com/svg/simple-svg-example.html ou encore https://www.w3schools.com/graphics/svg_intro.asp même si nous n'utiliserons pas l'intégration dans HTML5.
2. Rajoutez des attributs de couleur de remplissage (*fill*:struct RGB), épaisseur du trait (*stroke-width* :int) et couleur du trait (*stroke :* struct RGB) à la figure géométrique ; et accesseurs/mutateurs correspondants.
3. Ajoutez un attribut pour la zone d'affichage (*viewport*). De quel type ?
4. Écrire une méthode ExportSVG() dans l'Éditeur qui permette de sauver les figures dans un fichier dont on passera le nom en paramètre. Écrire les méthodes nécessaire dans la hiérarchie de FG. Là encore, quel mécanisme est mis en jeu ?
5. Testez le fichier créé dans Inkscape, et admirez !

Couper/Copier/Coller

Nous allons maintenant manipuler les FG au sein l'éditeur, avec les fonctionnalités classiques de Couper/Copier/Coller. En plus des traces dans les constructeurs/destructeurs, pensez à exportez systématiquement les FG dans un fichier SVG pour tester le bon fonctionnement.

1. Rajouter une fonctionnalité pour sélectionner une FG. On pourra par exemple simuler un clic dans l'Éditeur, et tester si le point cliqué est à l'intérieur d'une des FG.
2. Rajouter les constructeurs par copie et surcharge de l'opérateur= pour les FG. Tester. Dans quel cas ces méthodes sont-elles utiles ? Justifiez.
3. Créer une zone *tampon* pour stocker une FG à copier. De quel Type est cette zone ?
4. Écrire maintenant la méthode cut() qui stocke dans le *tampon la FG passée en paramètre*, et qui supprime de la liste de FG de l'Éditeur la FG. Cette FG est-elle toujours accessible ? Comment ?
5. Écrire la méthode paste() qui colle (ajoute à l'Éditeur) la FG présente dans le *tampon* (si il est non vide...) à la position (x,y) passée en paramètre. Si on ne passe pas de paramètres, la FG est décalée de (10,10) par rapport à l'originale.
6. Rajouter une méthode clone() dans FG et les classes Filles. Son prototype est `virtual FigureGeometrique* clone() const`. Utilisez cette méthode pour écrire la fonction copy() qui stocke la nouvelle FG dans le tampon. Commentez, notamment pourquoi on ne peut pas utiliser directement le constructeur par copie disponible dans FG ? Tester le copier/coller.

Remarque : cette technique est liée au Design Pattern *Factory*, très utilisé pour créer génériquement des objets de différents types.