UNIVERSITE CHEIKH ANTA DIOP
DE DAKAR

ECOLE SUPERIEURE
POLYTECHNIQUE

DEPARTEMENT GENIE INFORMATIQUE

DIC2

Projet de Java

TITRE
GéoAnalytique

DATE DE PRESENTATION ET DE LIVRAISON
16/04/2024

GROUPES
Le mini projet peut être réalisé par groupes de cinq à six étudiants. Chaque étudiant devra être en mesure d'expliquer l'intégralité du code de l'application ainsi que les différents choix opérés (chaque mini projet devra être présenté)**.**

LIVRABLES

Chaque groupe devra fournir:

- un document contenant le diagramme de classes demandé
- le code source clair et bien **documenté**,

OU

- une page Web donnant accès à la documentation API de toutes vos classes.

DESCRIPTION

1 Introduction

1.1 Présentation générale

Pour ce travail on vous propose de concevoir une application permettant de réaliser des dessins géométriques en utilisant des calculs analytiques.
L'outil consiste à définir un repère orthogonal. A partir de ce repère nous avons la possibilité d'ajouter des formes géométriques quelconques. En mathématiques, toute forme géométrique a des caractéristiques qui lui sont particulières. Ainsi, plusieurs opérations peuvent être appelées sur chaque forme géométrique. Bien sûr, ces opérations peuvent différer d'une forme à l'autre. Par exemple, les caractéristiques d'un segment peuvent être les coordonnées de ces deux points définissant ses extrémités. Les opérations associées à un segment peuvent être le calcul de sa longueur, sa pente, la construction de sa médiatrice, etc.

1.2 Présentation de l'architecture du logiciel

Nous vous proposons d'implémenter cette application graphique en utilisant une variante du modèle Model-View-Controller (MVC). En effet, dans le cas de cette application, l'architecture MVC est plus difficile à mettre en place à cause des changements de repère entre les coordonnées (réelles) des figures géométriques et les coordonnées entières dans le canevas graphique. Pour cela nous allons ajouter une quatrième entité qui est le Présenteur, son rôle est de transformer et préparer les coordonnées pour la zone d'affichage. Dans un souci de simplification, les rôles du Présenteur et du Contrôleur sont souvent fusionnés en une unique entité.
Pour résumer ce que nous venons de dire, la **figure 1** représente la nouvelle architecture à utiliser dans notre application.

- Modèle: décrit les modèles mathématiques des objets géométriques.
- Présenteur+Contrôleur: permet de préparer les coordonnées pour l'affichage et permet de retrouver les objets géométriques à partir de la Vue. Gère les évènements provenant de la vue.
- Vue: la vue affiche toutes les figures demandées, sans aucun lien avec le modèle (elle ne peut communiquer uniquement avec le contrôleur).

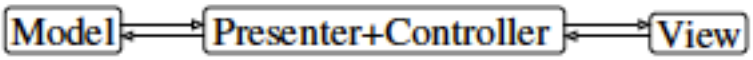

FIG. 1 – L'architecture de l'application

1.3 Le squelette fourni

Le squelette qui vous est fourni contient plusieurs paquetages. Avant de vous lancer dans le code, il vous est fortement suggéré de bien étudier la structure du squelette.
Le premier paquetage est *geoanalytique*. Ce package contient la classe *Main* qui est le lanceur principale de l' application (unique classe possédant un main). En plus, ce package contient plusieurs sous-packages:

- Le package *geoanalytique.model* définit les objets géométriques de manière mathématique. Il contient la classe *GeoObject*, qui est la classe mère (abstraite) de tous les objets géométriques. Vous y trouvez la classe *Point* qui est presque entièrement implémentée, ainsi que d'autres classes qui devront être complétées par vos soins (*Droite, Segment, Surface, Ellipse, Cercle* et *Polygone*). Ce package contient aussi la classe *ViewPort* qui définit la zone de dessin virtuel. En effet, en mathématiques un repère est infini sur l'ensemble des réels, cependant nous ne pouvons afficher qu'une certaine partie dans notre Vue. Cette classe constitue un moyen pour convertir les coordonnées réelles du repère en coordonnées entières de la Vue, et vice versa.
  Ce package contient un sous-package *geoanalytique.model.geoobject.operation* qui doit définir toutes les opération réalisables sur les objets géométriques. Nous vous donnons comme exemple la classe *ChangeNomOperation* qui permet de renommer un objet.
- Le package *geoanalytique.controleur* contient deux classes: la première est *GeoAnalytiqueControleur* qui sert de Contrôleur/Présenteur principal. Tous les événements importants émanant de l'interface graphique doivent passer par cette classe. Et la deuxième classe est *OperationControleur* gère les événements pour les opérations sur un objet.
- Le package *geoanalytique.view* contient la classe *GeoAnalytiqueView* qui représente la zone de dessin pour afficher les objets géométriques. Cette classe étend le composant

javax.swing.JPanel, et possède l'attribut graphiques regroupant tous les objets *Graphique* à afficher.

- Le package *geoanalytique.graphique* contient les classes qui redéfinissent les éléments graphiques basiques préfournis par JAVA (*GCoordonnee*, *GLigne*, *GOvale* et *GTexte*). Toutes ces classes héritent de la classe abstraite *Graphique*.
- Le package *geoanalytique.util* contient la classe *Dessinateur*, qui est utilisée par le Présenteur, pour convertir les modèles en objet graphique utilisable par la Vue. Cette classe implémente l'interface *GeoObjectVisitor<Graphique>*, qui est l'interface de base pour les visiteurs du modèle. (Une presentation de ce design pattern vous a été faite en cours de DIC1, chers informaticiens du groupe). Ce package contient aussi l'interface *Operation*, qui est l'interface générique pour toutes les opérations possibles sur un objet *GeoAnalytique* (par exemple la classe *geoanalytique.model.geoobject.operation.ChangeNomOperation* implémente cette interface).
- Le package *geoanalytique.gui* contient la classe *GeoAnalytiqueGUI* qui représente l'interface graphique principale de l'application.
- Le package *geoanalytique.exception* contient les classes définissant les différentes exceptions possibles pour notre application (*ArgumentOperationException*, *IncorrectTypeOperationException* et *VisiteurException*).

2 Le Modèle

Dans cette partie nous nous intéressons aux modèles internes pour représenter les objets géométriques.

2.1 Architecture des objets

La **figure 2** présente le diagramme UML des objets géométriques. La classe de base contient les méthodes pour prévenir le Contrôleur principal d'une modification.

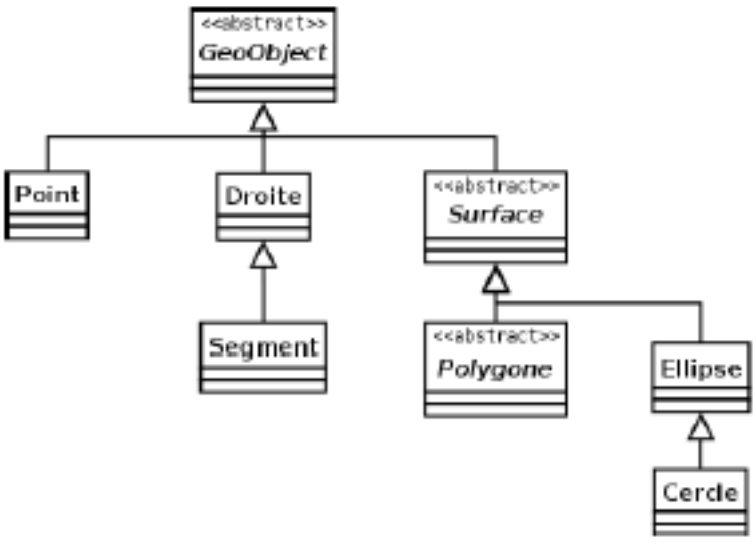

FIG. 2 – Le diagramme UML des objets géométriques

Question 1
Proposer les attributs pour chacune des classes concrètes pour représenter au mieux le modèle

mathématique des objets géométriques. (Par exemple un point est défini par son abscisse et son ordonnée).

Question 2

Implémenter les classes comme illustré dans la **figure 2**.

Important

Ne pas oublier de bien commenter le code et surtout de compléter les commentaires javadoc.

Question 3

La base que nous proposons est minimale et vous pouvez et devez ajouter d'autres objets dans la hiérarchie selon vos désirs. (par exemple : triangle équilatéral, triangle isocèle, triangle rectangle, triangle irrégulier, carré, rectangle, losange, etc.)

2.2 Opérations sur les objets

L'interface de base renvoie une collection d'opérations réalisables sur les objets géométriques. Cette interface est décrite dans la **figure 3**. Il est important de noter qu'une opération ne peut renvoyer qu'un unique résultat.

Question 4

Suivant le modèle d'opération proposée (renommage des objets), implémentez les opérations suivantes pour la classe Point :

- déplacer le point
- calculer la distance à un autre point
- calculer le milieu de deux points

Question 5

Proposer d'autres opérations pour les objets géométriques. Ces opérations peuvent bien être le calcul et l'affichage des médiatrices, bissectrices, médianes, centre de gravité, cercle circonscrit, cercle inscrit, calcul de la surface, contour, etc.

3 La Vue

3.1 Interface graphique de base

L'interface graphique doit contenir un nombre minimal d'éléments.

Question 6

Nous vous laissons libre pour le design de la fenêtre principale. C'est à vous de décider si vous voulez utiliser plusieurs zones d'affichage, des boutons, des menus déroulants etc. Cependant nous voulons voir la zone à dessiner.

3.2 Canevas

Le canevas est sous un composant graphique où nous avons redéfinit la fonction *void paint(Graphics g)*. Cette fonction est appelée lorsque le composant graphique doit se rafraichir et dessiner la zone cliente. Cette zone se contente d'afficher que des informations graphiques simples sans connaissance du modèle. Pour cela nous utilisons des objets visuels dédiés à cette tâche : *GCoordonnee, GLine, GTexte* et *GOval*.

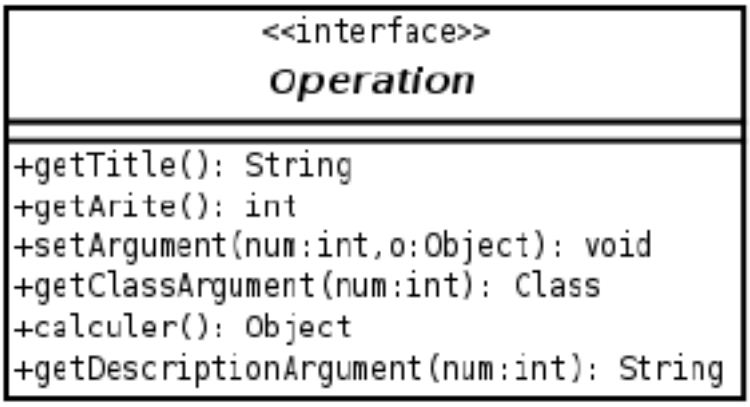

FIG. 3 – L'interface de base pour les opérations

Question 7
Compléter les classes du paquet *geoanalytique.graphique*, afin qu'elles puissent remplir leur rôle (voir commentaire pour chaque classe).

4 Le Contrôleur et le Présenteur

4.1 Le Présenteur

Le Présenteur est l'objet intermédiaire entre la Vue et le Modèle. Il permet, entre autres, de convertir les coordonnées dans le bon repère, et est capable de retrouver l'objet géométrique derrière un clique souris dans l'interface graphique. Cependant le présenteur repose sur d'autres classes, à savoir le *Viewport* qui convertit les coordonnées réelles en coordonnées entières, et le *Dessinateur* qui à partir d'un objet mathématique est capable de renvoyer sa représentation graphique.

Question 8
Compléter la classe *Dessinateur* suivant le modèle des points. Vous pouvez si besoin étendre la hiérarchie des objets graphiques.

Question 9
Compléter dans la classe *GeoAnalytiqueControleur* la fonction *recalculPoints* en ajoutant par défaut deux lignes pour représenter la droite des abscisses (Ox) et la droite des ordonnées (Oy)

4.2 Le Contrôleur

Le contrôleur gère les évènements provenant de la vue et informe/modifie le modèle si nécéssaire.

Question 10
Compléter :

- la fonction *addObjet* qui ajoute un objet dans le système et réalise toutes les opérations nécessaires à cette étape.
- la fonction *selectionner* qui selectionne un objet connu par le système et propose à l'utilisateur les opérations réalisables sur cet objet.
- la fonction *deselectionner* qui désélectionne l'objet en cours de selection et préviens la vue en conséquence.

Question 11
Réaliser toutes les liaisons entre le contrôleur et la vue, afin d'obtenir une application complète (on

complètera les fonctions nécessaires, voir les commentaires dans le code source).

Question 12

Etendre les possibilités de votre application comme vous l'entendez. Par exemple vous pouvez ajouter un zoom, ajouter la possibilité d'enregistrer la zone de dessin dans un fichier image, ajouter la possibilité de "drag and drop" pour les objets graphiques, ajouter des possibilities du style "enregistrer/ouvrir", etc.

Question 13

Modifier l'interprétation du résultat d'une opération (cf. la fonction *lanceOperation* dans la classe *GeoAnalytiqueControleur*) afin que l'on puisse renvoyer plusieurs résultats. On illustrera cette tâche par l'ajout de l'opération calculant les deux points d'intersection entre un cercle et une droite (non tangente et passant par l'intérieur du cercle). Attention, vos modifications ne doivent pas influer sur l'interprétation classique du résultat.