Vous devez compléter le système modélisé lors de la première séance de TD en ajoutant un *Contrôleur* chargé de recevoir les requêtes de l'utilisateur et interagissant avec le modèle et la vue afin de répondre à ces requêtes. Vous supposerez que l'instance de la classe *ObjetGéométrique* ainsi que l'instance correspondante de la classe *Image* ont déjà été créées.

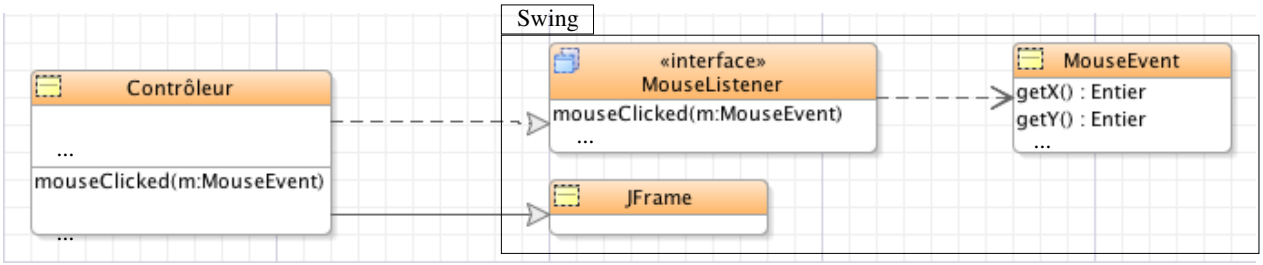
L'utilisateur interagit avec le système en utilisant le clavier et la souris.
- Quand l'utilisateur clique sur la souris : si les coordonnées du clic correspondent à la vue d'un élément géométrique, alors l'élément géométrique le moins profond se trouvant à ces coordonnées est sélectionné ; sinon aucun élément n'est sélectionné.
- Quand l'utilisateur appuie sur les touches h, b, g, ou d : si un élément est sélectionné, alors cet élément est translaté de 1 cm vers le haut, le bas, la gauche ou la droite en fonction de la touche appuyée. Si aucun élément n'est sélectionné, alors tous les éléments composant l'objet géométrique sont translatés. Dans tous les cas, la vue est redessinée.
- Quand l'utilisateur appuie sur la touche a : la dernière translation effectuée est annulée et la vue est redessinée.

Votre travail : Lors de la séance d'aujourd'hui [1], vous devez :
- dessiner le diagramme de séquence associé à un clic de souris ;
- compléter en conséquence le diagramme de classes élaboré lors du premier TD (cf page suivante).

Pour détecter les clics de souris dans une fenêtre graphique et gérer la liste des instances de la classe *VueEltGéo*, vous utiliserez des classes Java brièvement décrites ci-dessous.

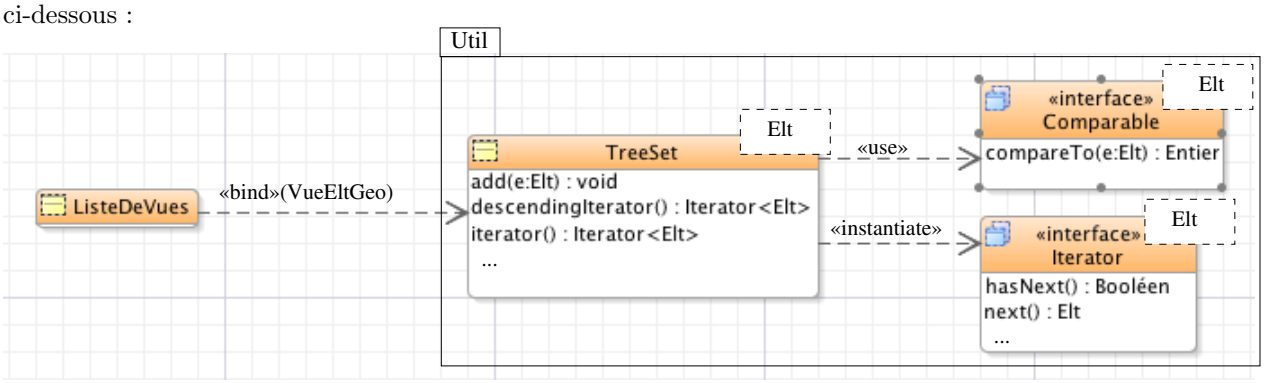
Détection d'événements dans une fenêtre graphique en Java : Une fenêtre graphique est définie en héritant de la classe *JFrame* de l'API *Swing*. Pour détecter les événements en provenance de la souris, il faut implémenter des méthodes déclarées dans l'interface *MouseListener* de l'API *Swing*, et notamment la méthode *mouseClicked(m : MouseEvent)* qui est appelée à chaque fois que l'utilisateur clique sur la souris dans la fenêtre graphique. Les méthodes *getX()* et *getY()* de la classe *MouseEvent* retournent les coordonnées du pixel cliqué. Au moment de la création de la fenêtre graphique, il faut appeler la méthode *addMouseListener(this)* pour indiquer que la fenêtre est son propre écouteur d'événements souris.

Vous supposerez que votre classe *Contrôleur* hérite de la classe *JFrame* et implémente l'interface *MouseListener* comme cela est décrit ci-dessous :

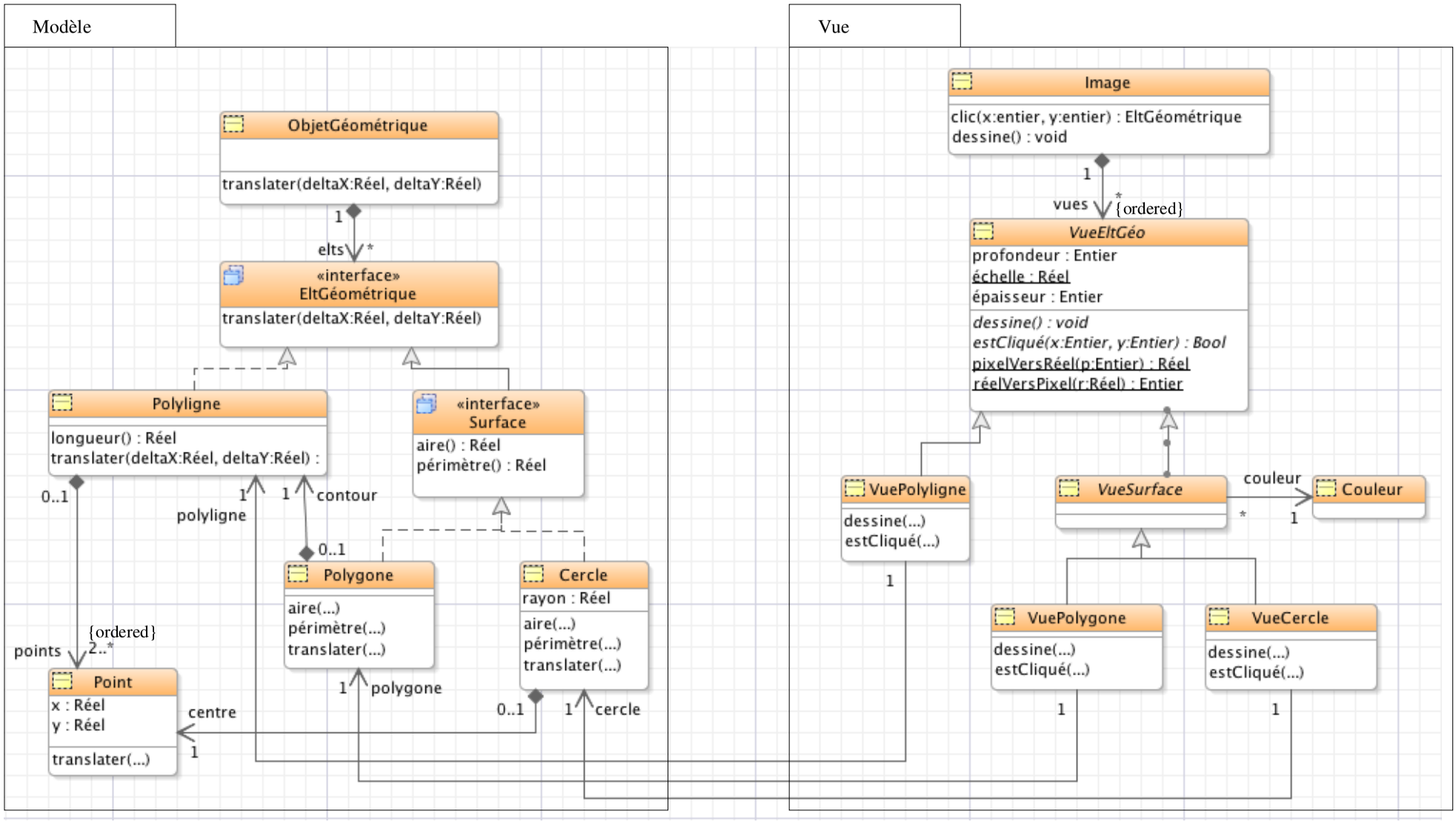


Gestion de collections ordonnées en Java : Le paquetage *java.util* contient des classes utilitaires implémentant les principales structures de données. Il contient notamment la classe *TreeSet* qui permet de gérer une collection ordonnée d'objets à l'aide d'un arbre binaire. Cette classe est générique et est paramétrée par le type *Elt* des objets de la collection. La classe *TreeSet* utilise la méthode *compareTo* de l'interface *Comparable* pour comparer deux instances de la classe *Elt*. La classe *TreeSet* contient deux méthodes créant des itérateurs : *iterator()*, qui crée un itérateur permettant de lister les éléments de *TreeSet* par ordre croissant, et *descendingIterator()*, qui crée un itérateur permettant de lister les éléments par ordre décroissant.

Vous supposerez que l'attribut *vues* de la classe *Image* est une instance de la classe *ListeDeVues* comme cela est décrit ci-dessous :



1. La semaine prochaine, vous terminerez la conception de cette application en dessinant le diagramme de séquence associé à la frappe d'un caractère au clavier.


Modèle
Vue
ObjetGéométrique
translater(deltaX:Réel, deltaY:Réel)
1
elts
*
«interface»
EltGéométrique
translater(deltaX:Réel, deltaY:Réel)
Polyligne
longueur() : Réel
translater(deltaX:Réel, deltaY:Réel) :
«interface»
Surface
aire() : Réel
périmètre() : Réel
0..1
1
polyligne
1
contour
0..1
Polygone
aire(...)
périmètre(...)
translater(...)
Cercle
rayon : Réel
aire(...)
périmètre(...)
translater(...)
points
{ordered}
2..*
Point
x : Réel
y : Réel
translater(...)
centre
1
1
polygone
0..1
1
cercle
Image
clic(x:entier, y:entier) : EltGéométrique
dessine() : void
1
vues
*
{ordered}
VueEltGéo
profondeur : Entier
échelle : Réel
épaisseur : Entier
dessine() : void
estCliqué(x:Entier, y:Entier) : Bool
pixelVersRéel(p:Entier) : Réel
réelVersPixel(r:Réel) : Entier
VuePolyligne
dessine(...)
estCliqué(...)
1
VueSurface
couleur
*
1
Couleur
VuePolygone
dessine(...)
estCliqué(...)
1
VueCercle
dessine(...)
estCliqué(...)
1