



sOOP

Compte rendu de projet

T. Lecointre – F. Everaert (Avril 2016)

SOMMAIRE

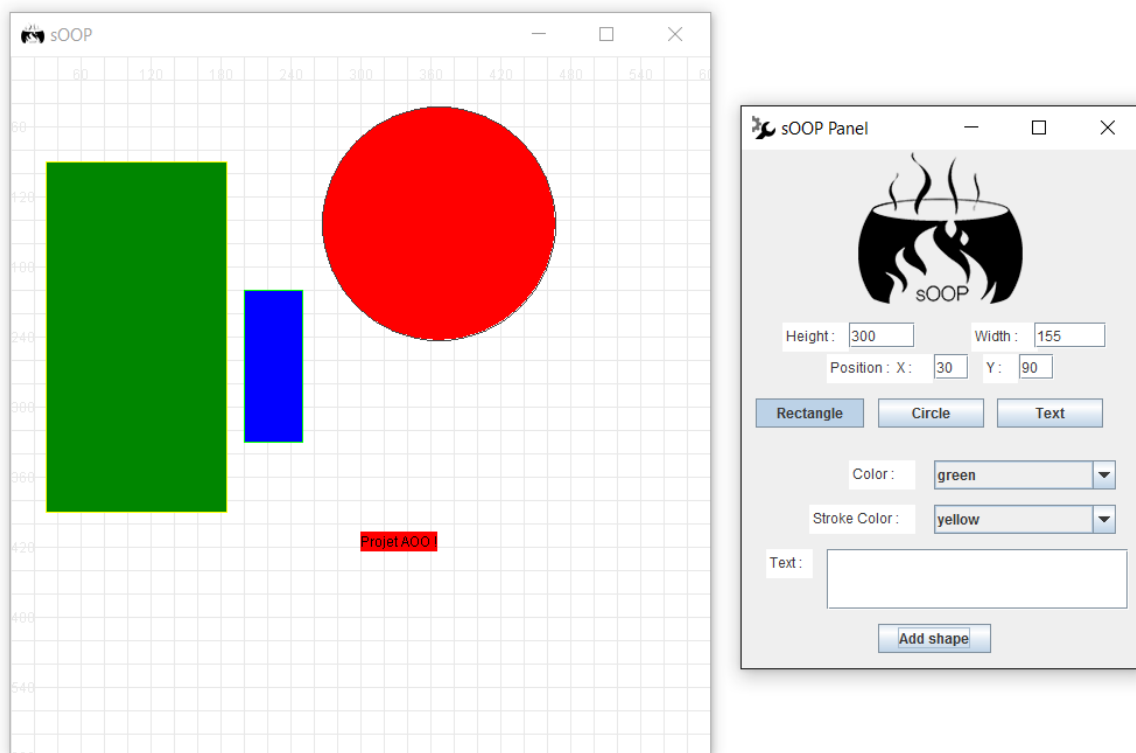
| | |
|----------------------------------|---|
| Descriptif de l'application..... | 3 |
| Fonctionnalités..... | 4 |
| Implémentation..... | 6 |
| Annexes..... | 9 |
| • Diagramme UML..... | 9 |

DESCRIPTIF DE L'APPLICATION

sOOP (shape Object Oriented Project) est un programme java minimaliste permettant d'afficher et déplacer des formes :

- rectangles
- cercles
- textes
- compositions de formes (collections)

Au démarrage, un ensemble de formes de départ est affiché. Il se compose de quelques formes élémentaires qu'il est possible de déplacer ou supprimer.



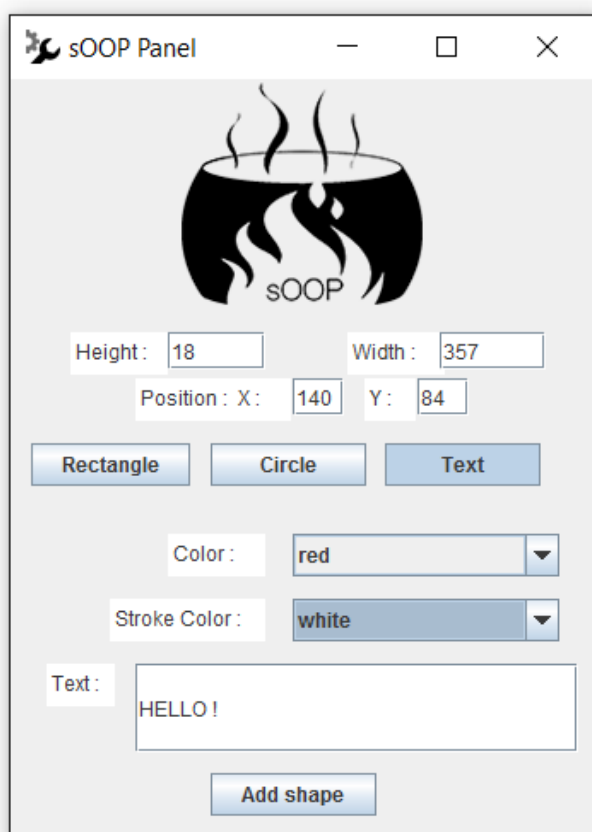
Exemple d'utilisation de sOOP

FONCTIONNALITÉS

L'application se compose de boutons permettant d'**ajouter** de nouvelles formes dans la fenêtre. Pour ceci, il suffit de rentrer dans le Panel les dimensions et la position de la forme choisie ainsi que ses couleurs de fond et de contour, puis de cliquer sur « Add shape ».

Pour **supprimer** une forme, il faut la sélectionner puis d'utiliser la touche « Suppr » du clavier.

On **sélectionne** une forme en cliquant dessus. Il est possible de sélectionner plusieurs formes en enfonçant la touche « Maj » puis en cliquant successivement sur les formes à sélectionner. En retapant plusieurs fois sur des formes tout en enfonçant la touche « Maj » on peut basculer entre la sélection et la « désélection » d'une forme. On peut sélectionner toutes les formes avec la combinaison « Ctrl+A ». Une forme sélectionnée sera entourée de petits marqueurs. On peut aussi **déplacer un ensemble** de formes sélectionnées.



Panneau de contrôle (Panel)

AJOUT D'UN RECTANGLE

Pour ajouter un rectangle, il faut préciser :

- hauteur (Height)
- largeur (Width)
- position X (du coin haut-gauche du rectangle)
- position Y (du coin haut-gauche du rectangle)
- couleur de fond (Color)
- couleur de contour (Stroke color)

AJOUT D'UN CERCLE


Pour ajouter un cercle, il faut préciser :

- rayon (Height divisé par deux)
- position X (du coin haut-gauche du rectangle contenant le cercle)
- position Y (du coin haut-gauche du rectangle contenant le cercle)
- couleur de fond (Color)
- couleur de contour (Stroke color)

AJOUT D'UN TEXTE

Pour ajouter un texte, il faut préciser :

- contenu du texte (Text)
- position X (du coin bas-gauche du rectangle)
- position Y (du coin bas-gauche du rectangle)
- couleur de fond (Color)
- couleur du texte (Stroke color)

 Il est impossible de cliquer sur « Add shape » lorsque deux ou trois types de formes sont sélectionnés.

IMPLÉMENTATION

PROBLÈMES RENCONTRÉS

- `FontRenderContext` lors de la définition des `FontAttributes`.

Au moment où nous avons besoin des dimensions d'un objet texte nous nous sommes aperçus que ceci dépendrait des attributs de la police dans son contexte graphique. L'attribut classe font de la classe *FontAttributes* possède une méthode `getStringBounds()` renvoyant les dimensions d'un texte et dont les arguments sont la chaîne de caractères et le *FontRenderContext* dont on parle avant.

Dans notre fenêtre, nous avons une vue sur les formes (*ShapesView*). Cette classe hérite la méthode `getGraphics` de *JPanel* qui renvoi un objet *Graphics*, c'est concrètement ce que l'on voit apparaître sur l'écran. Il se met à jour chaque fois qu'il est sollicité. Concrètement, on ne peut pas connaître les dimensions d'un objet texte sans connaître les propriétés de l'objet *Graphics* associé. Autre problème, il faut d'abord afficher le texte avant que l'on puisse connaître ses dimensions.

- Ajout de la grille (*Grid*)

La grille de fond a été relativement simple à mettre en place. La classe *Graphics* possède les méthodes `drawLine()` et `drawString()` permettant respectivement d'afficher des traits et des textes.

Essayant d'abord avec une grille de 300 000 pixels en largeur et en longueur nous avons voulu prévoir une fonctionnalité permettant de déplacer notre vue et non les formes elles mêmes. Nous nous sommes rapidement aperçus que le logiciel avait du mal à redessiner rapidement autant de pixels lors des déplacements de formes (déplacer une forme nécessitant elle même de effacer puis redessiner plusieurs fois les formes et la grille par dixième de seconde). Nous nous sommes finalement limités à 2000x2000 pixels (les ordinateurs actuels dépassent rarement cette dimension).

- Suppression des formes « précédentes » lors des déplacements.

Lorsque l'on déplace une forme sélectionnée il faut effacer la forme à sa position antérieure. L'affichage n'est pas cadencé comme on pourrait le voir dans un jeu vidéo ce qui génère des durées de rafraîchissements plus ou moins fluides et rapides. C'est pour cette raison que nous avons choisi de limiter la taille de la grille.

- Ajout du Panel avec Eclipse WindowBuilder.

Eclipse et le WindowBuilder de la librairie Swing offrent la possibilité de concevoir rapidement des fenêtres comportant plusieurs *JComponent* comme des boutons ou des champs textes. C'est en utilisant cet outil que nous avons programmé le *Panel*. Nous avons ensuite copié dans une classe « normale » le code généré afin qu'il puisse être lu par toutes les versions d'Eclipse. (La classe créée avec le WindowBuilder est toujours disponible dans le projet sous le nom de *ButtonsWindowBuilder*.)

L'un des plus gros problèmes rencontrés lors de l'élaboration du panel fut de créer les *ComboBox* pour les couleurs et d'appliquer celles-ci. En effet, pour créer la *ComboBox* correspondante au choix des couleurs, il faut une liste de couleurs à sélectionner, chaque couleur étant au format String. Pour appliquer ces couleurs aux formes il faut convertir les *String* en objet *Color*. Pour cela nous avons dû générer un *Stylesheet* car Java ne fait pas la conversion naturellement.

L'interaction utilisateur/logiciel se fait la plupart du temps avec la souris et le clavier. Les événements comme des clicks ou des déplacements de souris sont captés par notre logiciel et peuvent dans certains cas actionner du code.

- Création de nouvelles formes avec le Panel

Le second problème rencontré pour le *Panel* était de mettre à jour la fenêtre principale après avoir créé une forme. Le *Panel* étant une classe et une fenêtre différente, il est évident que les formes créées dans le *Panel* n'allaient pas apparaître immédiatement dans la fenêtre principale. Nous avons dû récupérer les formes générées pour les ajouter au *model*. Pour cela, nous avons passé le *model* et le *ShapesView* en paramètres de la méthode *affiche()* de la classe *Buttons* après l'instanciation du *Panel* dans *Editor*. On met à jour ensuite la fenêtre principale après chaque interaction avec le *Panel*.

- Ajout des icônes

Nous avons également créé des icônes pour la fenêtre principale et pour le *Panel* afin de donner au programme une identité visuelle.

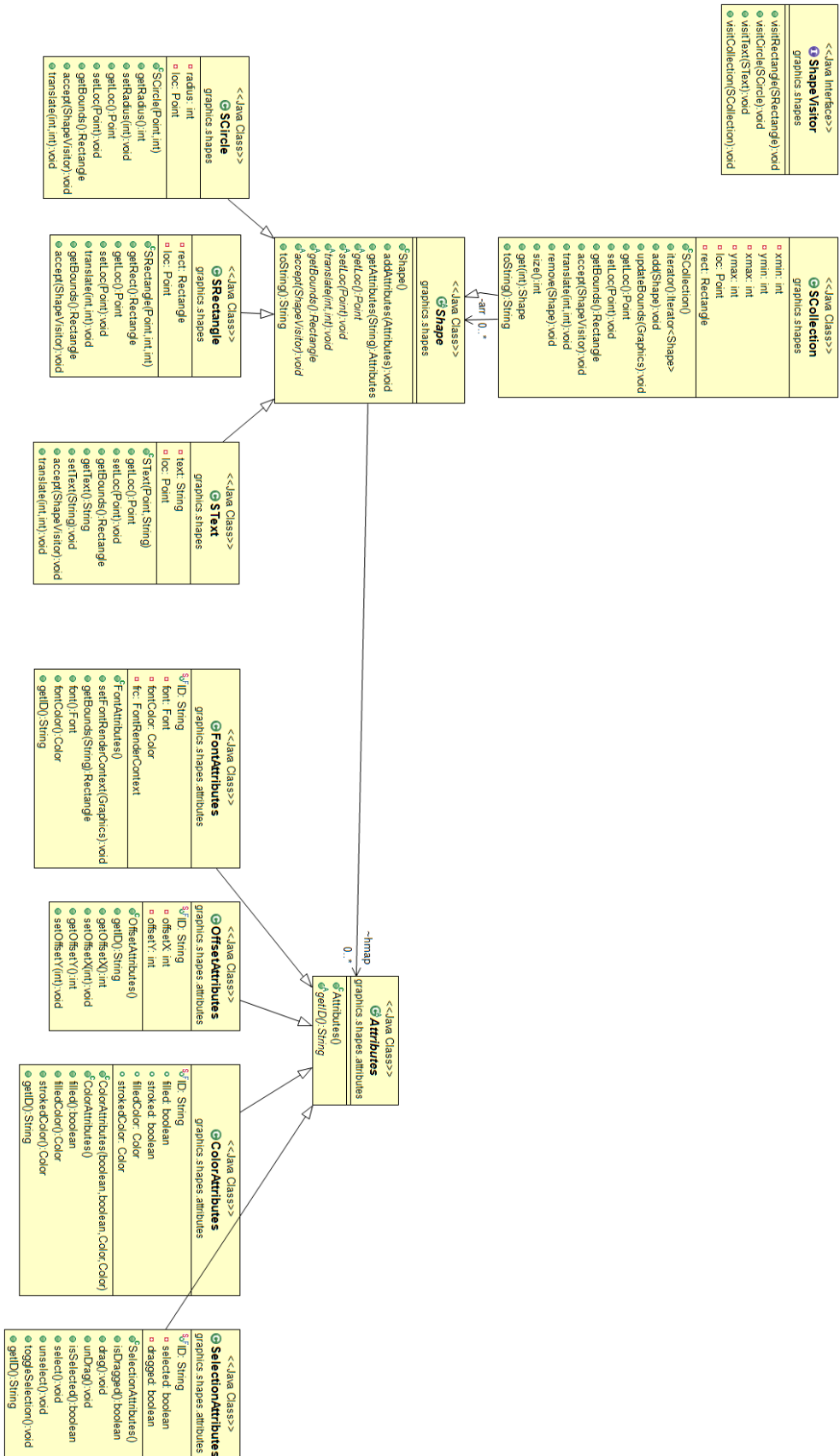


Diagramme des classes – UML – « Shapes et Attributes »

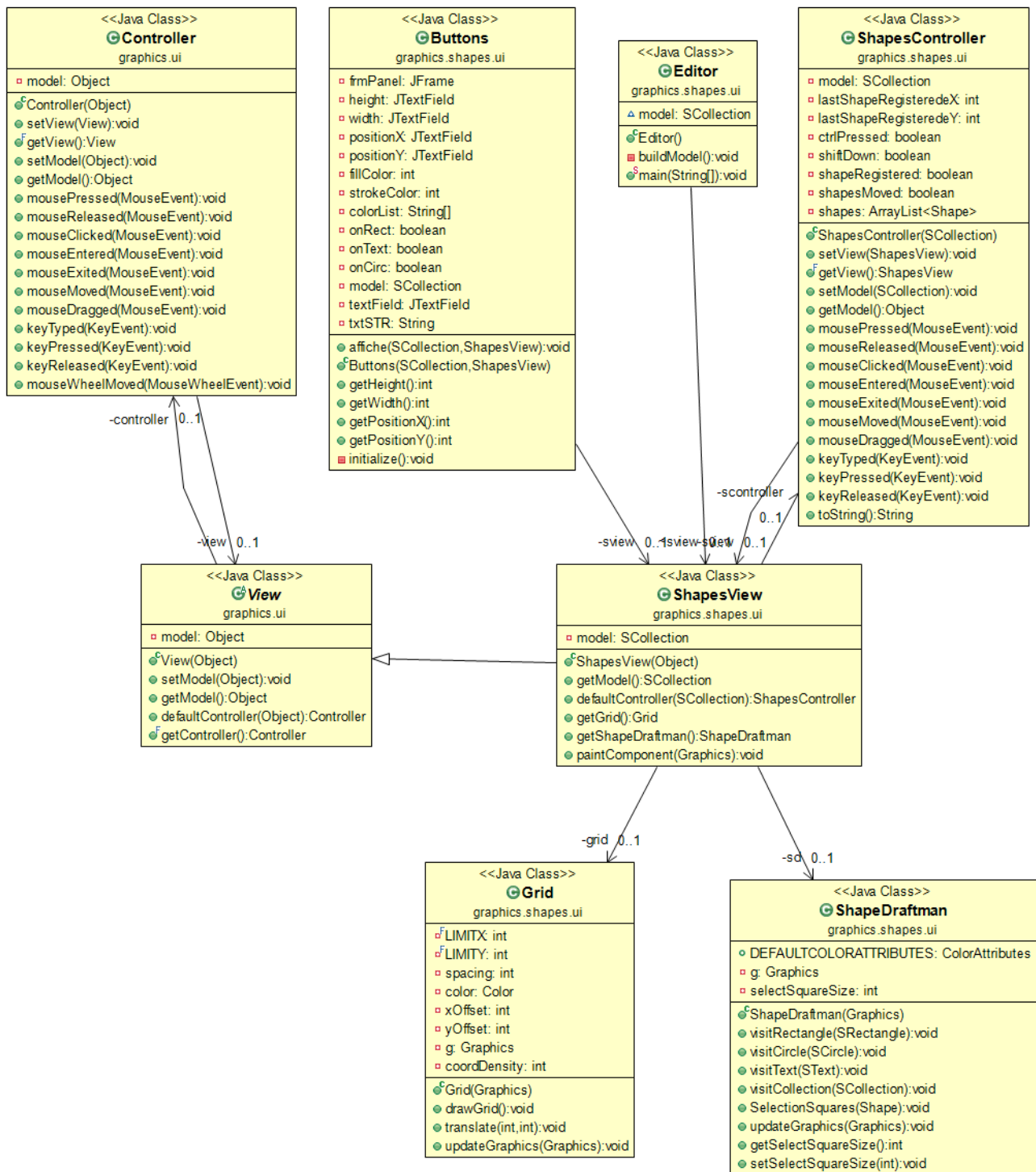


Diagramme des classes – UML – « User Interface »