# Séance n°4 - Travaux pratiques : Encore plus de menus et d'outils

## 1) Il faut savoir changer la couleur !

L'utilisateur peut changer de couleur de dessin au clavier par les touches R, V, B, J, N. Mais rien dans l'interface graphique ne suggère cette possibilité.

Dans le paquetage vue, modifier le constructeur de la classe **`BarreDeMenus`** pour ajouter un menu « Couleur » proposant les couleurs « Rouge », « Vert », « Bleu », « Jaune » et « Noir » au moyen d'instances de **`RadioMenuItem`** dont l'exclusion mutuelle sera gérée par un groupe d'options dédié, groupeCouleurs, instance de **`ToggleGroup`**. Associer à ces options de menu les traitements correspondants.

**Compiler et exécuter.**

## 2) Un écouteur clavier qui ne permet plus de changer de couleur !

L'utilisateur peut changer de couleur de dessin au clavier par les touches R, V, B, J, N. Mais ces changements ne sont pas répercutés dans le menu Couleur.

Dans le paquetage controleur, modifier la méthode **`onKeyPressed`** pour qu'il n'y soit plus possible de changer de couleur.

**Compiler et exécuter.**

## 3) Un écouteur clavier qui ne permet plus de changer d'outil… parce que le menu outil propose des raccourcis clavier !

On associe un raccourci clavier à une option de menu avec la méthode **`setAccelerator`**. Celle-ci reçoit en argument une instance de **`KeyCombination`** ou d'une de ses classes dérivées.

On associe par exemple le raccourci Ctrl-Q au choix Quitter comme ceci :

```
KeyCombination raccourci = new KeyCodeCombination(KeyCode.Q,
                                                  KeyCombination.CONTROL_DOWN);
choixQuitter.setAccelerator(raccourci);
```

Dans le paquetage controleur, modifier la méthode **`onKeyPressed`** pour qu'il n'y soit plus possible de changer d'outil.

Dans le paquetage vue, modifier la classe **`BarreDeMenus`** pour que chaque outil ait un raccourci clavier.

**Compiler et exécuter.**

**Séance n°4 - Travaux pratiques : Encore plus de menus et d'outils**

# 4) Un autre mode d'affichage !

On souhaite rétablir l'affichage fugitif en mode brouillon de la version 0_6_4_1 en le modifiant pour que les traces et les étoiles soient dessinées plus sommairement.

Dans le paquetage vue, définir une classe IterateurBrouillon qui implante partiellement l'interface Iterator (seulement les méthodes hasNext() et next()) et dont le constructeur prend en paramètre un itérateur dont le type est Iterator<Point2D>.

Dans la classe AffichageBrouillon, modifier les méthodes opereSur(Trace t) et opereSur(Etoile e) pour qu'elles utilisent une instance d'IterateurBrouillon au lieu d'un itérateur standard ou d'un *for-each*.

**Compiler et exécuter.**

# 5) Un nouvel outil de dessin !

On propose un troisième outil pour permettre à l'utilisateur de dessiner des segments.

Le paquetage modele a été revu en ajoutant une classe Segment et le paquetage contrôleur a été revu en ajoutant une classe EcouteurSegment.

L'interaction utilisateur diffère de celle mise en œuvre pour les outils crayon et étoile.

Lorsque l'utilisateur enfonce le bouton de la souris, seule la première extrémité du segment est connue. Par la suite, lorsqu'il déplace la souris en maintenant le bouton enfoncé, il détermine la seconde extrémité du segment. Cette dernière ne sera fixée que lorsque l'utilisateur relâchera le bouton de la souris.

Le segment peut donc être ajouté au modèle et dessiné dans la vue seulement au relâchement du bouton de la souris. Il faut pourtant dessiner un segment mouvant avec le déplacement de la souris (bouton enfoncé).

La classe **`ZoneDeDessin`** a été modifiée pour pouvoir dessiner provisoirement sans agir sur le dessin définitif. Elle hérite maintenant de **`StackPane`** (au lieu de **`Canvas`**) et contient deux canvas transparents superposés. Le dessin est fait dans le canvas définitif alors que le dessin « mouvant » est fait dans le canvas provisoire qui peut être effacé sans effacer le dessin définitif, comme si on dessinait et gommait sur un calque au dessus du dessin.

Les classes **`ZoneDeDessin`** et **`PanneauDeDessin`** proposent maintenant les méthodes :

- **`void effaceCalque()`**
- **`void dessineTraitCalque(double x1, double y1, double x2, double y2)`**

en plus de :

- **`void dessineTrait(double x1, double y1, double x2, double y2)`**

Avec le nouveau modèle, les dessins enregistrés comportent maintenant des traces, des étoiles et des segments. Il faut donc différencier les fichiers des précédentes versions, ce qu'on fera en commençant les fichiers par un octet à la valeur 2 (nouveau numéro de version de fichier). Mais on souhaite pouvoir lire ceux des anciennes versions.

## Séance n°4 - Travaux pratiques : Encore plus de menus et d'outils

Modifier le paquetage modele pour que le chargement et l'enregistrement soient réalisées par des classes distinctes implantant l'interface Operation.

Définir ainsi les classes ChargemetV1 et ChargementV2 ainsi que EnregistrementV1 et EnregistrementV2.

A la lecture d'un fichier, on instanciera un ChargementV1 ou un ChargementV2 en fonction du numéro de version lu au début du fichier. A cet effet, on définiera une classe implantant une méthode qui renvoie la bonne opération en utilisant le patron de conception Fabrique concrète.

**Compiler et exécuter.**

# 6) Et des préférences !

Au lancement de l'application, on dessine des traces (outil crayon) noires d'épaisseur 1.

Modifier le programme (dans quel paquetage faut-il intervenir ?) pour lire dans un fichier de préférences au format XML, l'outil, la couleur et l'épaisseur par défaut.

En l'absence de fichier de préférences, on établira l'outil crayon, la couleur noire et l'épaisseur 1.

**Compiler et exécuter.**