Université Paris Diderot **Programmation Orientée Objet**
L2-L3 **Année 2016-2017**

TP n° 10

Interfaces graphiques 2 : utilisation de la souris

Dans ce TP, nous allons implémenter l'interface `MouseInputListener`. Nous illustrons cela sur un petit jeu : on veut créer une fenêtre avec plusieurs carrés de couleurs aléatoires, cette couleur pouvant changer en fonction des actions faites avec la souris. Le jeu se finit lorsque tous les carrés sont de la même couleur. On rappelle qu'il faut tester systématiquement les méthodes créées.

1. Pour commencer, créer une classe `Cadre extends JFrame` ainsi qu'un constructeur sans argument. Ce constructeur affiche simplement une fenêtre de taille 600×600. De plus, le programme doit s'arrêter lorsque l'on ferme la fenêtre (on utilise pour cela la méthode `setDefaultCloseOperation`).
2. Appeler ce constructeur dans un `main`. Pour rappel, il faut utiliser :

```
javax.swing.SwingUtilities.invokeLater(new Runnable() {
     public void run() {   /*Votre code ici*/   }   } );
```

3. Rajouter un conteneur principal dans le constructeur. Celui-ci n'utilisera pas de `LayoutManager` (c'est-à-dire que le `LayoutManager` est initialisé à `null`).
4. Créer une classe interne `Carre extends JPanel` avec un constructeur sans argument. Pour le moment, on demande que le constructeur crée un carré de couleur bleue en position (100,200) dont les côtés sont de taille 50. Utiliser pour cela la méthode `setBounds` de la classe `Component`.

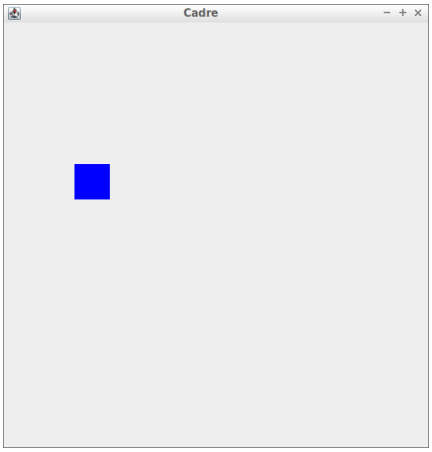

FIGURE 1 – Affichage du premier carré

La méthode `setBounds(int x, int y, int width, int height)` donne les dimensions `width` × `height` à un rectangle dont le coin en haut à gauche est en (`x`,`y`). Le point de coordonnées (`0`,`0`) est en haut à gauche de la fenêtre.

La gestion de la souris se fait par l'écoute d'évènements de type `MouseEvent`. Les composants graphiques détectent un certains nombre d'évènements, et s'ils sont écoutés par un `MouseInputListener` (qu'on leur aura rajouté par les méthodes `addMouseListener` et `addMouseMotionListener`) ils appelleront les méthodes appropriées de l'écouteur en cas d'évènement. Pourquoi deux et non une méthode ? Car l'interface `MouseInputListener` hérite en fait de deux interfaces, `MouseListener` et `MouseMotionListener`, qui détectent des types d'évènements différents : la première des évènements plutôt ponctuels, la seconde des évènements plutôt continus.

Clicked	Entered	Exited	Pressed	Released	Dragged	Moved
ML	ML	ML	ML	ML	MML	MML

FIGURE 2 – Les méthodes imposées par l'interface `MouseInputListener`. Pour chacune, on indique si elle vient de l'interface `MouseListener` (ML) ou `MouseMotionListener` (MML) Elles prennent toutes un `MouseEvent` en argument et elles sont à compléter par le préfixe `mouse` (ex : `public void mouseClicked(mouseEvent e)`

5. On demande à présent que la classe `Carre` implémente l'interface `MouseInputListener` (de façon à définir les actions faites par la souris). Plusieurs méthodes sont à redéfinir, pour le moment nous ne demandons pas de leur donner de corps (elles ne font rien). Que faut-il faire pour que (une fois les méthodes redéfinies) les actions voulues aient effectivement lieu ?
6. On souhaite qu'un carré puisse être déplacé à l'aide du click-and-drag (on clique avec le bouton de la souris sur le carré, on déplace le carré en gardant le bouton enfoncé et lorsqu'on le lâche le carré est à la nouvelle position de la souris). On souhaite de plus qu'une fois déplacé le carré soit de couleur rouge. Quelle(s) méthode(s) faut-il implémenter ?

Pour repositionner votre carré dans votre fenêtre, voici quelques éléments :

— les méthodes `getX(` et `getY()` de `MouseEvent` renvoient les coordonnées où l'évènement a eu lieu *par rapport à l'objet écouté* (c'est à dire que dans le cas de nos carrés 50×50, forcément des nombres entre 0 et 50). `getPoint()` renvoie les mêmes informations dans un objet `Point`.

— les méthodes `getXOnScreen()` et `getYOnScreen()` de `MouseEvent` renvoient les coordonnées où l'évènement a eu lieu *par rapport au coin en haut à gauche de l'écran*. `getLocationOnScreen()` renvoie les mêmes informations sous forme d'un `Point`.

— la méthode `setLocation(int x, int y)`, ou `setLocation(Point p)`, de `JPanel`, déplace le point en haut à gauche du carré en `(x,y)` (ou en `p`),

les coordonnées étant par rapport au composant le contenant, comme pour `setBounds` (donc le coin en haut à gauche de l'intérieur de la fenêtre, dans notre cas).

— les méthodes `getX()` et `getY()` de `JFrame` permettent d'obtenir les coordonnées du coin en haut à gauche de la fenêtre Java par rapport à l'écran.

— la méthode `getInsets()` de `Jframe` permet d'obtenir un objet `Insets` dont les attributs `left`, `top`, `right`, `bottom` indiquent le nombre de pixels des différentes bordures de la fenêtre (côtés gauche, haut, droit et bas).

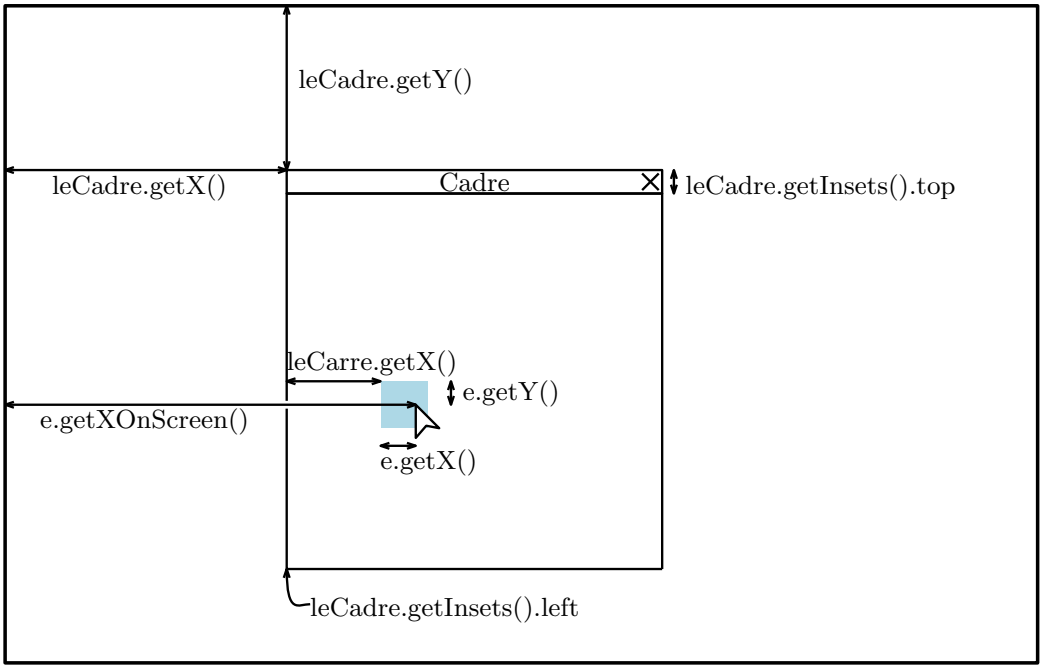

FIGURE 3 – Explication du système de coordonnées. Le grand rectangle représente l'écran, le petit une fenêtre Java désignée par `leCadre`. `e` est un évènement qui vient d'être lancé par la souris (par exemple, on vient de clicker). `leCarre.getX()` et `getY()` sont les coordonnées qui peuvent être changées par la méthode `leCarre.setLocation(int x, int y)`

7. Modifier le constructeur de la classe `Carre` pour que la couleur et la position dans la fenêtre soient aléatoires.
8. Modifier le constructeur de la classe `Cadre` de telle sorte que la fenêtre contienne un nombre aléatoire (compris entre 1 et 10) de carrés (créés en faisant appel au constructeur précédent).
9. Écrire une méthode `boolean gagne()` qui teste si tous les carrés sont de la même couleur.

10. Écrire une méthode `void finJeu()` qui ferme la fenêtre (et donc arrête le programme) une fois que tous les carrés sont de la même couleur.
11. On souhaite que lorsque le curseur est sur un carré (sans que l'on clique sur la souris) celui-ci devienne bleu. Lorsque l'on clique sur un carré celui doit devenir vert. Écrire les méthodes correspondantes.
12. Modifier la méthode `void finJeu()` de telle sorte qu'un message de succès soit affiché sur la fenêtre (on ne demande évidemment plus que la fenêtre se ferme automatiquement). Pour cela on rajoutera dans la classe `Cadre` un attribut `JPanel etiquette`.
13. On souhaite que lorsque le jeu est fini, il suffit de cliquer n'importe où sur la fenêtre (y compris sur les carrés) pour que celle-ci se ferme. On souhaite également que la fenêtre se ferme si le curseur de la souris sort du cadre. Écrire les méthodes correspondantes.
14. Si vous avez du temps, vous pouvez réfléchir à une autre façon de modifier la couleur d'un carré. Par exemple, si un carré passe "au-dessus" d'un autre on peut choisir que le carré du dessous récupère la couleur de celui du dessus. Vous pouvez également demander à ce qu'il y ait un message sur la fenêtre décrivant la situation (par exemple "3 carres bleus, 2 carres rouges, un carre vert").

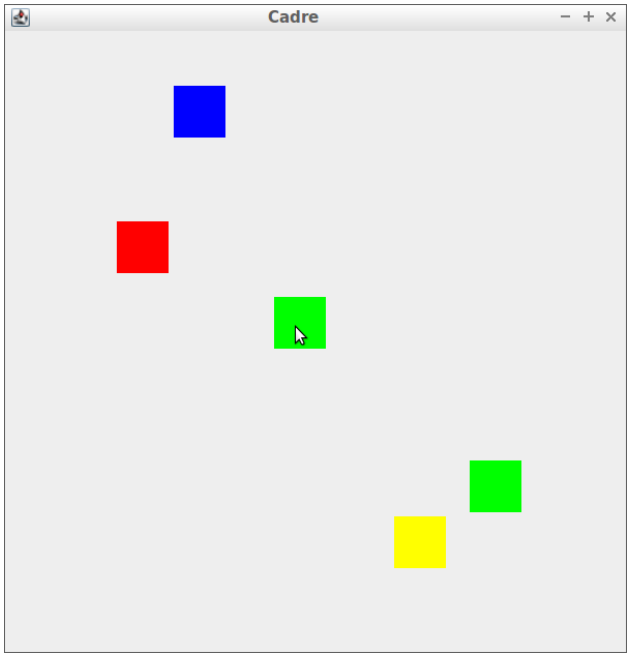

FIGURE 4 – Un exemple du jeu une fois implémenté