Université de Paris **Programmation Orientée Objet**
L2 **Année 2020-2021**

TP n° 12

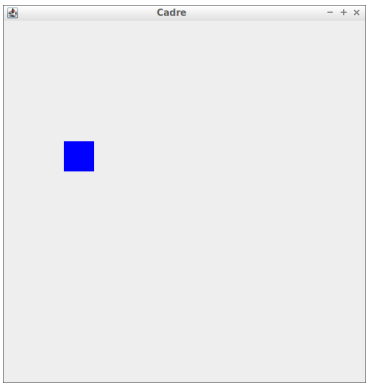
Petit jeu autour de l'utilisation de la souris

Dans une fenêtre graphique nous allons disposer plusieurs carrés de couleur. En utilisant la souris nous allons effectuer des actions qui permettrons de modifier la couleurs de ces carrés. Le but de ce petit jeu est qu'au final ils aient tous la même couleur. Nous nous intéresserons également au déplacement des carrés.

1. Pour commencer, créez une classe `Cadre` qui étend `JFrame`. Munissez la d'un constructeur sans argument qui fixe la taille de la fenêtre à 600×600. Paramétrez là pour que le programme s'arrête lorsque l'on ferme la fenêtre (voir `setDefaultCloseOperation`). Testez ce premier affichage en reprenant la séquence :

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

2. Pour travailler sur une page précise, ajoutez un champs a votre classe de type `JPanel` qui jouera le rôle d'un conteneur principal. Initialisez le dans le constructeur sans utiliser de `LayoutManager` (c'est-à-dire que le `LayoutManager` sera initialisé à `null`).
3. Créez une classe interne `Carre extends JPanel` disposant d'un constructeur sans argument. Pour le moment, on se contentera de fixer ses paramètres pour obtenir un carré bleu positionné en (100,200) dont les côtés sont de taille 50. (Utilisez pour cela la méthode `setBounds(int x, int y, int width, int height)` de la classe `Component` sachant que le point de coordonnées (0,0) est en haut à gauche de la fenêtre. Testez votre travail afin d'obtenir :



Nous avons déjà utilisé des évènements liés à la souris dans des TP précédents. On rappelle que 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 si on leur a déclaré qu'ils sont écoutés par tel ou tel `MouseInputListener` ils appelleront les méthodes appropriées des écouteurs en cas d'évènement. Cette déclaration se fait par les méthodes `addMouseListener` et `addMouseMotionListener`. Pourquoi deux et non une méthode ? Car l'interface `MouseInputListener` hérite de deux interfaces, `MouseListener` et `MouseMotionListener`, la première s'occupe 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 1 – 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)`

4. On demande que la classe `Carre` implémente l'interface `MouseInputListener`. Plusieurs méthodes seront à redéfinir, mais pour le moment il suffira qu'elles ne fassent rien. C'est le carré lui même qui prendra en charge les actions faites par la souris. Déclarez cet écouteur au JPanel/JFrame approprié.

   Pour tester progressivement la mise en place des actions, commençons par faire en sorte que lors d'un click un message "click !" s'affiche sur la console. Assurez vous que cet affichage ne se produise que lors d'un click précisément sur le carré.

5. On voudrait qu'un carré puisse être déplacé à l'aide de la souris. Pour dire les choses plus précisément : lorsqu'on cliquera une première fois sur un carré, celui ci se mettra à suivre les mouvements de la souris, jusqu'au moment où un second click le libèrera. Pour faire cela, la première étape consiste à introduire une variable d'état qui précisera si le carré sera en mode "mouvement" ou non. Procédez à un test en affichant chaque changements d'état.

Pour qu'ensuite le carré se mette à suivre les mouvements de la souris, il va falloir le repositionner. Notez que le mouvement n'est pas un évènement "ponctuel" mais "continu" (voir ce qui a été dit plus haut). Il faut également prendre en compte quelques règles de disposition relative pour calculer proprement ses nouvelles coordonnées. Voici quelques éléments à connaître :

— 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 `JPanel` 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).

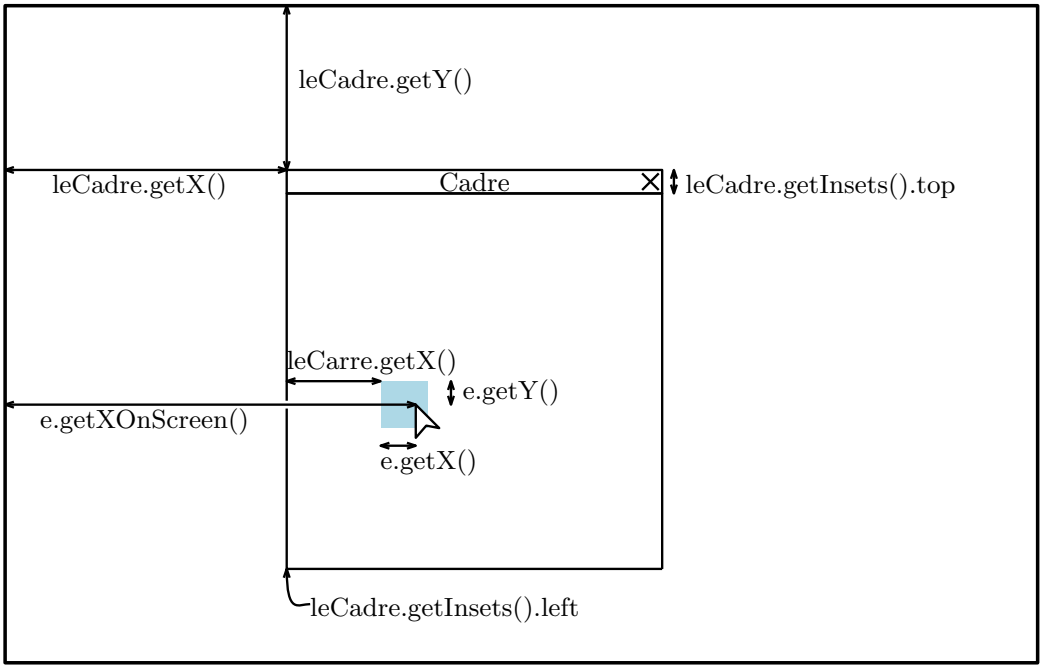


FIGURE 2 – Explication du système de coordonnées.

— Le grand rectangle représente l'écran,

— le petit une fenêtre Java désignée par `leCadre`.

— `e` désigne un évènement de click qui vient d'être lancé par la souris.

— `leCarre.getX()` (et `getY()` non représenté) sont les coordonnées qu'on voudrait voir changer

Or pour changer les coordonnées du coin supérieur gauche du carré, on ne peut le faire que relativement au cadre par la méthode `leCarre.setLocation(int x, int y)`

Il va donc falloir, pour les calculer la coordonnée y il faut retrancher à la position courante de la souris : la position du cadre, l'épaisseur du cadre, et prendre en compte l'endroit où on a cliqué au sein du carré.

6. Terminez l'implémentation du déplacement d'un carré.
7. Modifier le constructeur de la classe `Carre` pour que la couleur et la position dans la fenêtre soient choisies aléatoirement.
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. On souhaite à présent revenir à notre idée initiale de jeu, le Carré que nous avons écrit jusqu'à présent représente uniquement une vue. Ecrivez une classe pour le modèle dans lequel les positions des objets n'auront pas d'importance, mais les couleurs oui, et ajustez votre code pour séparer vue et modèle.
10. Écrire une méthode `boolean gagne()` qui teste si tous les carrés sont de la même couleur.
11. 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.
12. On souhaite que lorsque le curseur survole un carré (sans que l'on clique sur la souris) celui-ci devienne bleu, et que si on clique sur un carré celui devienne vert. Écrire les méthodes correspondantes.
13. 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`.
14. On souhaite que lorsque le jeu est fini, n'importe quel click provoque la fermeture de la fenêtre. (N'importe où sur la fenêtre, y compris sur les carrés). On souhaite qu'elle se ferme également si on fait simplement sortir le curseur de la souris en dehors du cadre. Écrire les méthodes correspondantes.
15. 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. Ce serait appréciable aussi qu'il y ait un message sur la fenêtre qui décrive la situation (par exemple "3 carres bleus, 2 carres rouges, un carre vert").

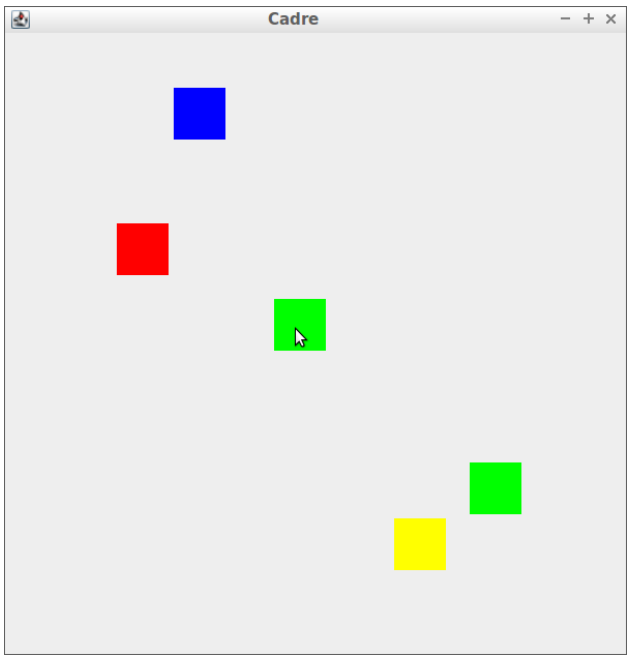


FIGURE 3 – Un exemple du jeu une fois implémenté