

INF3034 : Programmation Orientée Objet

TP n°5 : Interface graphique

A. Gademer *

2009

Vous devez me rendre les résultats des deux TP 4 et 5 (le moteur de jeu et l'interface) par mail avant le **15 Janvier 23h59** à l'adresse suivante : gademer@esiea.fr.

Votre rendu comprendra le projet Netbeans dans une archive `NOM1_NOM2_C13X.zip` où `NOM1` et `NOM2` sont les noms des deux binomes et `C13X` votre classe.

1 Objectifs

L'objectif de ce TP est de faire une interface graphique pour le jeu d'échecs réalisé au TP n°4. Nous reprendrons donc le code de celui-ci pour nous servir de base.

2 Présentation

Nous désirons faire une application graphique Java qui permet de jouer de manière simplifiée aux échecs.

Si vous ne l'avez pas encore fait, commencez par lire la fiche consacrée aux interfaces graphiques.

On peut distinguer deux parties à ce TP :

- d'une part, nous allons créer une fenêtre contenant l'application et détourner un composant pour fabriquer une zone de dessin.
- d'autre part, nous allons modifier certaines classes des packages `game` et `game.chess` écrites au TP n°4 pour les adapter au mode graphique.

Tout ceci est décrit par la figure 1.

Les nouvelles classes seront regroupées dans un nouveau package `game.gui` (click droit > New > J

- La classe `Fenetre` correspond à notre fenêtre d'application. C'est le composant principal de notre interface graphique.
- La classe `ZoneDeDessin` est une sous-classe de `JPanel` qui correspond à une zone de dessin. C'est ce composant qui nous permettra de dessiner l'échiquier et les pièces placées dessus.

*gademer@esiea.fr

3 Interface graphique de l'application

3.1 Fenetre

La classe `Fenetre` hérite de la classe `JFrame` et possède un certain nombre de champs privés dont on a besoin de retenir la référence :

- `ZoneDeDessin dessin` qui correspond à la zone de dessin.
- `JLabel message` qui est le champ texte qui indique tour à tour quelle couleur doit jouer.
- `JLabel etat` qui indique la `Position` de la case survolée, ainsi que les messages d'erreurs.
- `Position start` qui mémorise la case sur laquelle a commencé le clic souris.
- `boolean turnNoir` qui mémorise à qui est le tour.

La première méthode de la classe est le constructeur par défaut `Fenetre(String name, Plateau p)` qui initialise les composants dont la fenêtre est constituée.

`name` correspond au titre de la fenêtre (propriété héritée de `JFrame`) et `p` correspond au plateau de jeu du TP n°4.

Le plan de construction de la fenêtre est indiqué comme suit :

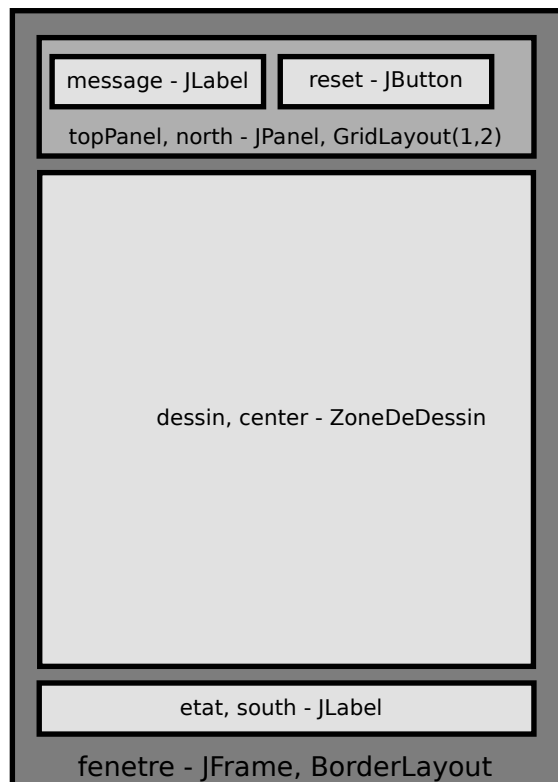


FIG. 2 – Organisation de la fenêtre d'application

Certains composants correspondent aux champs de la classe (`message`, `dessin`...), d'autres ne sont définis que dans le constructeur (`reset`, `topPanel`) car ils n'ont pas vocation à être modifiés dans les autres fonctions.

`message` doit être initialisé avec le texte "Blanc joue :", et `turnNoir` à `false`.

3.2 ZoneDeDessin

Regardons de plus près cette étrange classe qui doit dessiner notre échiquier. Elle étend `JPanel` ce qui lui fait de lui un composant graphique avec les mêmes propriétés que les autres composants de *javax.swing*.

Pour pouvoir afficher ce que l'on désire, on redéfinit sa méthode `paintComponent(Graphics g)` qui prend en argument le contexte graphique et fait appel à ses méthodes pour dessiner.

Dans notre cas, nous nous contenterons d'appeler la méthode `paintComponent(Graphics g)` de la classe mère (avec le mot clef `super`) et la méthode `paint(Graphics g)` de notre classe `Plateau` (nous verrons cela dans la partie ??).

Enfin, notre constructeur `ZoneDeDessin(Plateau p)` se contentera d'initialiser le champ `p` avec l'argument passé et de fixer la taille du composant en fonction de la taille prévue par le plateau. (`p.getSize()` comme nous le verrons dans la partie ??)...

La classe est si petite qu'en voici le code :

```
package game.gui;

import game.Plateau;
import java.awt.Graphics;
import javax.swing.JPanel;

public class ZoneDeDessin extends JPanel {

    public Plateau p;

    public ZoneDeDessin(Plateau p) {
        super();
        this.p = p;
        setPreferredSize(p.getSize());
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        p.paint(g);
    }
}
```

4 Quelques modifications...

4.1 Dessiner le plateau

Nous avons vu que nous aimerions que notre classe `Plateau` possède un certain nombre de méthodes supplémentaires :

```
public void paint(Graphics g);
public Dimension getSize();
```

qui seront appelées dans `ZoneDeDessin`.

Ces méthodes resteront abstraites pour `Plateau` et seront implémentées dans chaque sous-classe, ici la classe `Echiquier`.

La méthode `paint(Graphics g)` correspond au tracé de l'échiquier.

Quelques outils de base de la classe `Graphics` que nous pourrions utiliser :

```
g.setColor(Color c)
g.drawString(String str, int x, int y)
g.drawLine(int x1, int y1, int x2, int y2)
g.drawOval(int x, int y, int width, int height)
g.fillOval(int x, int y, int width, int height)
g.drawRect(int x, int y, int width, int height)
g.fillRect(int x, int y, int width, int height)
```

Je vous conseille de vous fixer une taille de carreaux (50 pixel par exemple) puis de tracer le damier en gris (`Color.GRAY`) et blanc (`Color.WHITE`). Si chaque case nous voulons un symbole correspondant à chaque pièce... il nous faudra donc implémenter une nouvelle méthode dans la classe `Piece`. Nous verrons cela juste après.

La méthode `getSize()` devrait découler facilement une fois que vous aurez fixé la taille d'un carreau.

4.2 Identifier les pièces

Nous avons besoins de charger en mémoire un symbole unique pour chaque pièce. Pour cela nous allons rajouter la méthode suivante à la classe `Piece`

```
ImageIcon getImage();
```

`ImageIcon` (*javax.swing*) est un objet simple contenant une image chargé à partir d'un fichier image sur le disque dur (`.bmp`, `.jpg` ou `.png` par exemple).

On l'instancie avec une `String` contenant le chemin vers l'image.

Pour retourner une `ImageIcon` correspondant à notre pièce on cherchera un fichier image du type `img/xxxx_noir.png` ou `xxxx_blanc.png` où `xxxx` correspond au nom de la pièce en minuscule.

Pour vérifier que l'image à bien été chargé on utilisera la méthode `getImageLoadStatus()` et la constante `MediaTracker.COMPLETE`.

Vous trouverez sur le blog une archive avec les fichiers images correspondants.

Nous pouvons alors utiliser la méthode `paintIcon(Component c, Graphics g, int x, int y)` pour "peindre" l'image de chaque pièce sur l'échiquier.

5 Prendre un peu de recul

À ce stade vous devriez avoir une fenêtre qui affiche l'ensemble des composants comme sur la figure ci-dessous :

Mais rien ne fonctionne! Il va nous falloir rajouter un peu d'action dans tout ça. Et c'est le rôle des gestionnaires d'événements (`EventManager`).

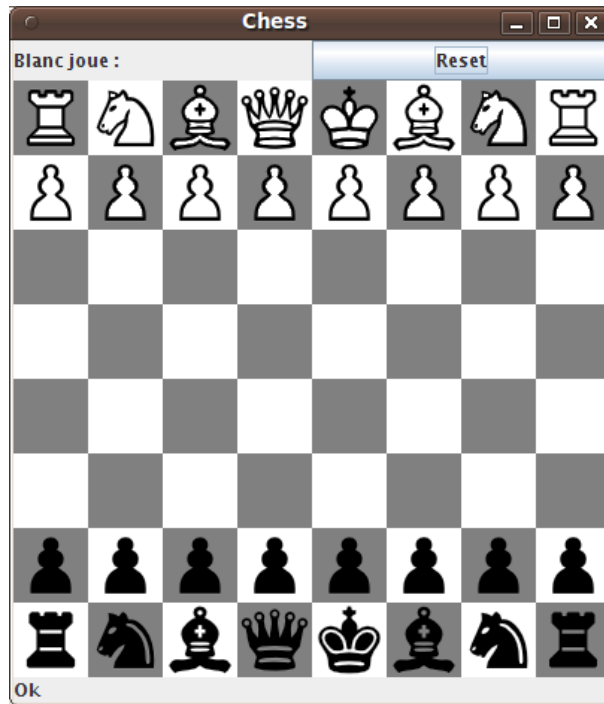


FIG. 3 – Interface graphique

6 Gestion des événements

Les gestionnaires d'événements sont des interfaces qui permettent de définir des classes redéfinissant des méthodes d'action et pouvant être associée à des composants.

Par exemple l'interface `MouseEventListener` permet de définir une classe qui l'implémentera et pourra être associé à un conteneur (`JFrame` ou `JPanel`). Dès lors, à chaque mouvement de la souris, le conteneur invoquera les méthodes de la nouvelle classe.

Rien ne vaut l'exemple.

6.1 `MouseEventListener`

Dans notre cas c'est la classe `Fenetre` qui implémentera l'interface `MouseEventListener`. Elle devra donc définir les méthodes associée à l'interface :

```
void mouseDragged(MouseEvent e)
void mouseMoved(MouseEvent e)
```

En associant, dans le constructeur, la classe `Fenetre` à la zone de dessin :

```
dessin.addMouseListener(this);
```

on s'assure que la méthode `mouseMoved(MouseEvent e)` sera appelée à chaque fois que la souris passera au dessus de la zone de dessin.

Et justement, on aimerai, qu'à chaque fois que cela se produit le label `etat` reflète les coordonnées (ex : D6) de la case survolée.

Comment faire ?

L'argument `e` contient les coordonnées en pixel par rapport au coin en haut à gauche du composant associé (la zone de dessin).

```
e.getPoint()
```

Mais comment les convertir en `Position` ?

Il va nous falloir écrire une nouvelle méthode pour la classe `Plateau` et la classe `Echiquier`.

```
Position getPosition(Point p);
```

Ce qui ne devrait pas être insurmontable connaissant la taille d'un carreau. **Attention** cependant ! La méthode doit retourner `null` si les coordonnées du `Point` sont en dehors de l'échiquier.

6.2 MouseListener

Nous allons rajouter un deuxième gestionnaire d'événement pour gérer les clics souris : `MouseListener`.

On rajoute donc cette interface à notre classe `Fenetre` ainsi que les méthodes associées :

```
mouseClicked(MouseEvent e)  
mouseEntered(MouseEvent e)  
mouseExited(MouseEvent e)  
mousePressed(MouseEvent e)  
mouseReleased(MouseEvent e)
```

et la petite ligne dans le constructeur

```
dessin.addMouseListener(this);
```

On désire simuler l'action du *Drag & Drop*. Le joueur cliquera sur la pièce qu'il désire déplacer et ne relâchera la pression qu'une fois sur la case à atteindre.

Nous allons exploiter les méthodes `mousePressed(MouseEvent e)` et `mouseReleased(MouseEvent e)`.

Dans `mousePressed(MouseEvent e)` nous allons récupérer la position de la souris et sauver la `Position` associée dans le champ `start`.

Dans `mouseReleased(MouseEvent e)` nous allons récupérer la position de fin et tenter de jouer. Si le coup est valide, la pièce sera déplacée et nous changeons le boolean `turnNoir` et le texte de `message` et nous rafraîchissons le dessin avec la méthode `updateUI()`. Sinon nous affichons le message d'erreur sur le label `etat`. Dans tout les cas nous remettons `start` à `null`.

Cela devrait nous permettre de jouer.

6.3 ActionListener

Pour finir, il nous faut nous occuper du bouton `reset`. Pour cela nous ajoutons l'interface `ActionListener` à notre `Fenetre`, la méthode associée :

```
public void actionPerformed(ActionEvent e)
```

et les deux lignes dans le constructeur :

```
reset.setActionCommand("reset");  
reset.addActionListener(this);
```

Dans la méthode `actionPerformed(ActionEvent e)` nous vérifions la valeur du mot-clé avec la méthode `getActionCommand()` de l'argument `e` puis nous remettons le plateau dans sa condition initiale, la main aux blancs avec le bon texte pour `message`. Enfin, nous actualisons `dessin` avec la méthode `updateUI()`.

7 Programme

La classe `Programme` devient presque obsolète. Il suffit d'y créer un nouvel objet `Fenetre` avec un titre et un `Echequier` pour pouvoir commencer à jouer.

8 Au travail !