IUT- Département d'Informatique
G. Rey
2016

Introduction à l'IHM – M2105
TD n° 8

1 Objectifs

Ce TD/TP illustre la partie du cours sur l'analyse et la conception des interfaces homme-machine. Les concepts principaux abordés seront :

- Mise en œuvre sur un exemple complet

Un compte rendu donnant la réponse à chacune des questions posées devra être rendu à votre responsable de TD/TP par email à la fin de chacune des séances. L'objet (sujet) de l'email devra être le suivant :

[S2T][IHM][Gx][TDx]Nom-Prénom / Nom-Prénom

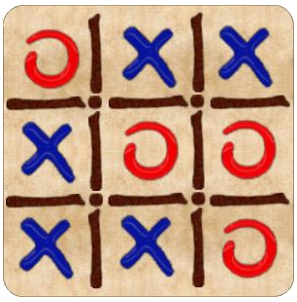
Avec Gx le numéro de votre groupe (exemple G1 pour le groupe 1) et TDx le numéro du TD/TP inscrit sur le sujet du TD/TP (exemple TD1 pour le premier TD).

Vous ne pouvez pas faire 2 séances avec le même binôme. De même, vous ne pouvez pas faire plus de la moitié des séances seul.

Sauf indication contraire, les exercices « fil rouge » (identifiés par le logo et uniquement ceux-ci), pourront être fait par des groupes allant jusqu'à 4 personnes (mais pas plus). De plus ces groupes pourront être les même à chaque séance.

Veuillez, sauf indication contraire de votre responsable de TD, faire l'ensemble des exercices marqués « Obligatoire » dans l'ordre de la feuille de TD/TP. Les exercices « Conseillés » sont à faire si vous avez terminé les précédents ou lors de vos révisions. Les exercices « optionnels » sont là pour les plus passionnés d'entre vous.

2 Rappel : Projet fil rouge



Dans la suite de nos TDs nous allons construire une application en partant de l'analyse des besoins et en allant jusqu'à la réalisation de cette application.
Pour cette année, le choix de l'application est un jeu de tic-tac-toe. Le Tic-tac-toe est un jeu de réflexion se pratiquant à deux joueurs au tour par tour et dont le but est de créer le premier un alignement. [Wikipédia]

Vous avez déjà établi pour ce projet : un scénario partiel (sous la forme d'une description narrative), des modèles de l'utilisateur (Rasmussen et persona), un arbre des tâches, un ensemble de maquettes de l'interface (avec les liens entre les différents « écrans » de l'application) ainsi que l'arbre de widget de ces différents « écrans ».

3 Exercice 1 : Mise en œuvre sur un exemple concret (obligatoire)

L'objectif de ce TD/TP est de concevoir votre application de tic-tac-toe en vous appuyant sur

- Ce que nous avons vu en TD depuis le début de l'année
- Ce que vous avez défini précédemment lors des étapes d'analyses et de conceptions (cf projet fil rouge)
- Ce qui va vous être fourni pour accélérer votre développement.

3.1 Contraintes de développement

Le projet fil rouge que vous allez développer devra respecter les contraintes suivantes :

Introduction à l'IHM – M2105
TD n° 8

- Le projet devra mettre en avant la séparation des préoccupations selon le modèle ARCH et MVC. Pour cela on séparera le code en trois package java : nf, cd et ihm qui correspondrons respectivement au noyau fonctionnel de l'application, a son contrôleur de dialogue et à son interface graphique.
- Le projet devra externaliser l'ensemble des chaines de caractères dans des fichiers de configuration selon le modèle vu en TD précédemment. Au minimum, une version en français et une version en anglais de ce fichier devront être fournis.
- Le projet devra reprendre le code du noyau fonctionnel fourni. Celui-ci ne devra pas être modifié.

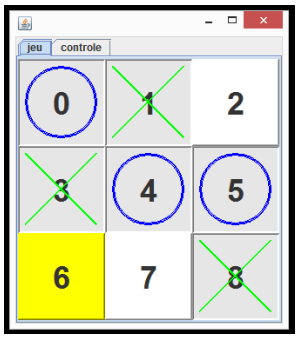
3.2 Analyse de code fourni

- Commencez par étudier la classe abstraite TicTacToe dont le code vous est fourni. Il est conseillé d'ouvrir le projet eclipse fourni.
- Puis testez le jeu local en ligne de commande.
- Maintenant, examinez le code de la classe Jeu et les autres classes qui peuvent être misent à votre disposition.

Quand vous avez compris le fonctionnement du programme :

- Indiquez quelles sont les parties du code fourni qui correspondent au noyau fonctionnel, au contrôleur de dialogue et à l'interface.

3.3 Développement d'une première version



Avant de développer votre application fil rouge comme vous l'avez définie lors de TD précédent, nous allons commencer par créer une version très simple (une seule fenêtre, pas de menu, ...) ni vous aidera à vérifier si vous avez bien compris le principe de séparation des préoccupations et de structuration du code selon les modèles nommés précédemment.

Le code ci-dessous présente un lanceur d'application qui va créer une fenêtre (JFrame) contenant deux panneaux (JPanel) commutable par un mécanisme d'onglets. Le premier panneau contiendra une grille de jeu jouable directement à la souris. Chaque case de la grille étant un JLabel qui contiendra un numéro et une croix ou un cerle (si on y a joué dessus). Le deuxième panneau affichera des informations sur le jeu (qui vient de jouer et où), à qui c'est de jouer et permettra également d'annuler le dernier coup jouer ainsi que recommencer une nouvelle partie.

- Commencez par examiner la classe ci-dessous.

```
import ihm.Case;
import ihm.StatusDisplay;
import cd.CaseListener;
import cd.StatusListener;
import nf.TicTacToe;

import java.awt.GridLayout;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTabbedPane;
import javax.swing.SwingUtilities;

public class Lanceur {

	private static TicTacToe partie;

	public static void main(String[] args) {
		//initialisation du jeu
		partie = TicTacToe.obtenirTicTacToeLocal();
```

Introduction à l'IHM – M2105
TD n° 8


```
                partie.enregistrerJoueur("Asterix");
                partie.enregistrerJoueur("Obelix");
                partie.debuterPartie();

                SwingUtilities.invokeLater(new Runnable(){
                        @Override
                        public void run(){
                                //On crée une nouvelle instance de notre JFrame
                                JFrame window = new JFrame();
                                window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                                //On crée un gestionnaire d'onglets
                                JTabbedPane newpane = new JTabbedPane();

                                //on crée un panel de controle (avec des boutons) ainsi que son contrôleur
                                StatusListener sl = new StatusListener(partie);
                                StatusDisplay  sd = new StatusDisplay(sl);

                                //on crée un panel de jeu (la grille) ainsi que son contrôleur
                                JPanel p = new JPanel(new GridLayout(3,3));
                                CaseListener l = new CaseListener(partie, window);
                                Case [] grilleG = new Case[9];
                                for(int i = 0; i < 9; i++){
                                        grilleG[i] = new Case(l, i); //une Case est un JLabel
                                        p.add(grilleG[i]);
                                }

                                newpane.addTab("jeu", p);
                                newpane.addTab("controle", sd);

                                window.setContentPane(newpane);
                                window.pack();
                                window.setVisible(true);//On la rend visible
                        }
                });
        }
}
```

- Développez les 4 classes utilisées dans le lanceur.java et qui sont manquantes pour son fonctionnement. Pour le moment on pourra oublier la contrainte de régionalisation de l'interface.
- Testez votre programme.

3.4 Construire votre propre application en jeu locale

Maintenant que vous avez réalisé une première application qui respecte les contraintes de séparation des préoccupations, il ne reste plus qu'à continuer le développement de votre propre application en y ajoutant les fonctionnalités que vous aviez identifiées dans vos phases d'analyses et conceptions. Il faudra également intégrer la régionalisation des interfaces.

- Modifiez votre application précédente pour intégrer la régionalisation ainsi que les autres fonctionnalités de votre application.

4 Annexes : Quelques aides pour résoudre des petits problèmes techniques

Le but de ces annexes est de vous donner quelques solutions techniques à des problèmes que vous pourriez rencontrer dans ce dernier TD. Il n'y a aucune obligation de mettre en œuvre ces solutions dans votre projet fil rouge. Votre projet doit correspondre à ce que vous avez modélisé jusqu'à maintenant.

Introduction à l'IHM – M2105
TD n° 8

4.1 Les Images

Il y a plusieurs façons de charger des images. Cela va varier en fonction de l'API utilisée ou du lieu de stockage de l'image.

4.1.1 java.awt.Toolkit

Attention :

- Le chargement est fait de manière asynchrone, en réalité il n'a lieu que lors du premier appel à *drawImage()*.
- L'image renvoyée ne contient pas de donnée (proxy)

Plus d'information sur cette classe dans la JavaDoc.

```
Image image = java.awt.Toolkit.getDefaultToolkit().getImage("fichier");
Image image = java.awt.Toolkit.getDefaultToolkit().getImage(url);

Image image = java.awt.Toolkit.getDefaultToolkit().createImage("fichier");
Image image = java.awt.Toolkit.getDefaultToolkit().createImage(url);
```

getImage() par rapport à createImage() maintient un cache de toutes les images chargées pour partager la même image entre plusieurs appels. Mais en fonction de la version de Java utilisée, il peut y avoir des problèmes de libération mémoire pour les images en cache.

4.1.2 javax.imageio.ImageIO

Depuis Java 1.4, une nouvelle API de gestion des images a été introduite. On peut donc charger les images avec les instructions ci-dessous.

```
BufferedImage image = ImageIO.read("fichier");
BufferedImage image = ImageIO.read(url);
BufferedImage image = ImageIO.read(InputStream);
```

Contrairement à la solution précédente, avec *ImageIO*, les images sont complètement chargées lors du retour de la méthode *read()*. On n'a pas de chargement asynchrone.

Par contre, pour des questions de performance, il faut penser à adapter l'image obtenu avec cette méthode pour qu'elle compatible avec le format des images à afficher à l'écran. Plus d'information sur ce point dans la FAQ de java.developpez.com.

4.1.3 Comment gérer une image dans un Jar ?

Le plus simple est de laisser le classLoader s'occuper de trouver le fichier. On peut par exemple

```
java.net.URL url = getClass().getResource("fichier");
BufferedImage image = ImageIO.read(url);
```

4.1.4 Peut-on mettre une image en arrière-plan d'une fenêtre ou d'un panneau ?

Oui, cela est possible. Il suffit de surcharger la méthode de dessin du composant.

En SWING, il s'agit de la méthode *paintComponent()* de la classe *JComponent*.

```
public void paintComponent(Graphics g) {
    g.drawImage(image,0,0,null);
}
```

En AWT, il s'agit de la méthode *paint()* de la classe *Container*.

```
public void paint(Graphics g) {
     g.drawImage(image,0,0,null);
}
```

Introduction à l'IHM – M2105
TD n° 8

4.2 Le dessin en Java

Les informations qui suivent viennent en partie du cours disponible sur cette page. N'hésitez pas à le consulter pour avoir plus de détail.

L'outil de dessin est le *contexte graphique*. C'est un objet de la classe *Graphics* ou de sa classe dérivée *Graphics2D* pour Java 2. Cet objet encapsule toute l'information nécessaire au dessin sous forme d'un état graphique qui comporte en autres les propriétés suivantes :

- zone de dessin, c'est-à-dire le composant où s'effectue le dessin,
- une transformation affine des coordonnées,
- une zone de découpe (clipping),
- le trait (Stroke),
- la couleur courante et la texture,
- la fonte courante,
- le mode de dessin.

4.2.1 Méthodes paintComponent et repaint

En Swing, c'est dans la *paintComponent()* définie dans la classe JComponent que doit être fait le dessin d'un composant.

En AWT, on utilisera la méthode *paint()* qui doit être redéfinie pour ajouter du dessin.

La règle à toujours respecter : *Tout programmeur qui redéfinit la méthode paint(AWT) ou paintComponent(Swing) s'engage à redessiner l'intégralité de la surface du composant. En effet, l'objet Graphics qui est transmis au composant dessine toujours dans le même espace mémoire et cet espace n'est jamais nettoyé (pour des raisons de performance). Le programmeur du composant a donc la charge de nettoyer l'espace et de redessiner son composant. [java.Developpez.com]*

Par défaut, la méthode paintComponent appelle la méthode ComponentUI.update() qui efface et redessine le fond si le composant est opaque (comme JPanel par défaut). Lorsque la méthode paintComponent est redéfinie, la méthode de la classe mère doit être appelée par super.paintComponent pour conserver l'appel à ComponentUI.update().

```
public void paintComponent(Graphics g) {
    // Appel de la méthode paintComponent de la classe mère
    super.paintComponent(g);
    // Conversion en un contexte 2D (en réalité g est un objet de type Graphics2D.
    Graphics2D g2 = (Graphics2D) g;
    ...
```

4.2.2 Que puis-je faire comme dessin ?

Il est possible de dessiner des formes géométriques (vide ou pleine), des images et des textes. En plus de cela, il existe un ensemble de méthodes pour faire du traitement sur les intersections des formes, pour leur appliquer des transformations, …

Vous trouverez des détails sur ce sujet dans la JavaDoc ainsi que sur ce cours en ligne. Y a aussi ces quelques tutoriels d'Oracle sur le sujet qui pourront vous aider en cas de besoin.

4.2.3 Le double-buffering

Le double buffering consiste à dessiner une scène en mémoire puis à copier cette image à l'écran. L'avantage de cette méthode est que l'on ne voit pas la scéne lors de sa phase de dessin mais seulement à la fin.

Introduction à l'IHM – M2105
TD n° 8

Vous pouvez consulter le tutoriel Java sur le sujet ou consulter directement la page sur la classe createBufferStrategy qui va vous permettre de faire du double (ou triple) buffering avec une accélération matérielle de cette solution. Vous trouverez aussi des explications et un exemple sur la FAQ de java.developpez.com.