IN201 : 7ter – GUI (nuage)

Author : Christophe Garion <garion@isae.fr>
Public : SUPAERO 2A
Date :

Solution

Résumé

Ce TP a pour but de manipuler quelques aspects avancés de l'API Swing : affichage personnalisé d'un composant, utilisation d'horloges etc.

1 Contenu

Ce document est un corrigé succint du dernier TP sur les interfaces graphiques. Tous les fichiers source sont disponibles sur le site http://www.tofgarion.net/lectures/IN201.

2 Problématique

On souhaite pouvoir créer une application graphique permettant d'afficher une carte de France et de déplacer un mobile (un nuage) sur cette carte. En cliquant sur la carte, on matérialise le chemin du nuage à sa destination par une ligne rouge et un ovale se dessine sur la destination. Un exemple est présenté sur la figure 1.

3 Conception et implantation de la fenêtre

La fenêtre de l'application est très simple : il s'agit d'une `JFrame` possédant une instance de `JScrollPane` et un `JLabel` contenant l'image de fond et le mobile. Vous pouvez l'implanter avec les classes de base dans un premier temps. On choisira une taille préférée de l'instance de `JScrollPane` égale à $(500, 500)$.

Solution :
Pas de problème particulier ici, on pouvait effectivement tester simplement la vue en la construisant de cette façon.

4 Conception du `JLabel`

On souhaite utiliser une instance de `JLabel` pour afficher la carte et le mobile. La méthode `setIcon` de `JLabel` nous permet d'afficher la carte sur l'objet, mais on ne peut pas afficher le mobile en même temps. De plus, il va falloir dessiner non seulement le mobile, mais également une ligne vers sa destination et un ovale. On va donc devoir redéfinir la méthode `paintComponent` de `JLabel` pour réaliser cet affichage. Une spécialisation de `JLabel` en une classe `Carte` est donc nécessaire.

1. quels sont les attributs nécessaires à `Carte` ? Faut-il des accesseurs et des modifieurs à ces méthodes ? On ne s'intéressera dans un premier temps qu'au dessin du nuage et de sa destination.

   Solution :
   Les attributs nécessaires à `Carte` sont les suivants :
   - deux entiers représentant les coordonnées du nuage ;
   - deux entiers représentant les coordonnées de la destination ;
   - éventuellement, un entier représentant la modification de la taille de l'ovale d'arrivée (pour simuler la pulsation).

   Nous avons juste besoin de modifieurs pour les coordonnées de la destination (on les modifie depuis l'extérieur de la classe lorsque l'on clique sur la carte).
   J'ai choisi de passer en constructeur de `Carte` le nom du fichier contenant l'image de fond, la position initiale du nuage et la vitesse de déplacement de celui-ci (qui sera stockée comme un attribut).

2. on utilise un objet de type `Timer` pour pouvoir réaliser les déplacements du mobile. Cet objet aura un *listener* qui devra être prévenu lors des « clics » de l'horloge. Proposer une solution simple pour que le *listener* puisse utiliser la méthode `paintComponent` de `Carte`.

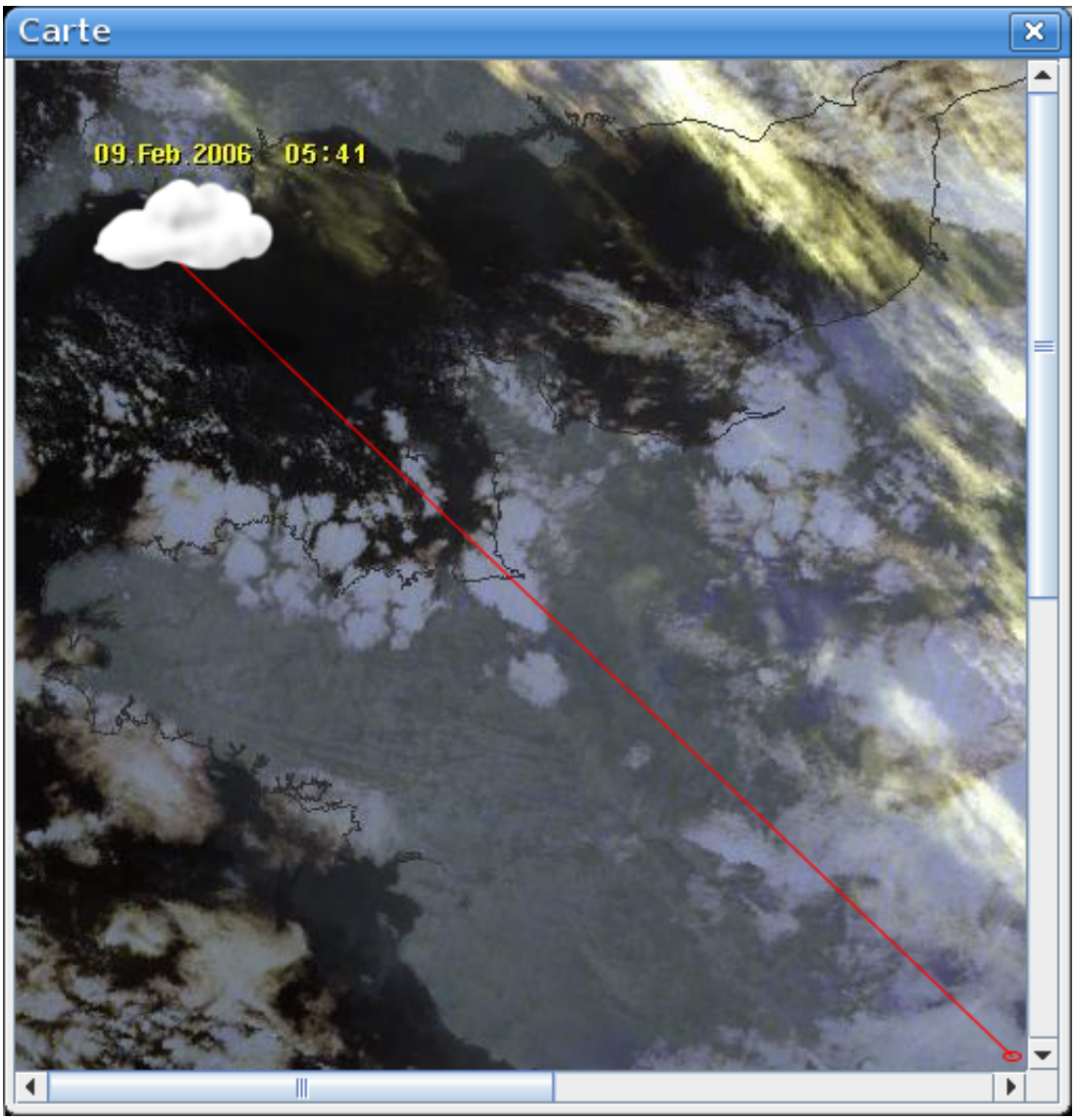

FIGURE 1 – L'interface graphique « en action »

Solution :
Il faut que le *listener* associé à l'horloge puisse appeler la méthode `paintComponent` de `Carte`. Il serait très simple que ce soit l'instance de `Carte` qui soit elle même *listener* de l'horloge. Pour cela, il suffit que `Carte` réalise l'interface `ActionListener` et implante la méthode `actionPerformed`. On va également mettre l'horloge comme attribut de `Carte`. De cette façon, on pourra arrêter l'horloge depuis la classe `Carte` (par exemple lorsque le nuage est arrivé à destination).
Il faut par contre proposer une méthode publique `start` dans `carte` pour pouvoir démarrer l'horloge depuis l'extérieur.

5 Implantation de Carte

Pour réaliser la classe `Carte`, nous vous proposons le plan de développement incrémental suivant (tester l'interface à chaque étape) :

1. spécialiser `JLabel` et afficher la carte en fond.

   Solution :
   Pas de problème particulier, il fallait définir un constructeur propre pour `Carte`.

2. redéfinir `paintComponent` et afficher le nuage à une position de départ, ainsi que la ligne vers sa destination et un ovale.

   Solution :
   Là encore, pas de problème particulier, il fallait suivre ce qui avait été présenté en cours pour l'utilisation d'une instance de `Graphics2D`. Évidemment, si la distance entre le nuage est l'arrivée était inférieure à 10 pixels (choix arbitraire), il ne fallait rien afficher. J'ai utilisé `setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON)` pour mettre en place l'anti-aliasing.

3. implanter l'horloge (avec une période de 50 ms) et tester l'application en donnant des coordonnées d'arrivée « à la main ». On pourra utiliser la méthode statique `calculeNouvellesCoordonnees` fournie dans le fichier `newCoord.java` que vous trouverez dans votre dépôt pour recalculer les positions.

   Solution :
   Rien de bien particulier. La classe `Carte` réalisait l'interface `ActionListener` et était enregistré comme *listener* auprès de l'horloge contenue dans `Carte`. Dans la méthode `actionPerformed`, on recalculait les nouvelles coordonnées grâce à la méthode fournie, éventuellement la pulsation de l'ovale et on appelait `repaint` pour afficher l'instance de `Case`.

4. ajouter un *listener* sur l'instance de `Case` pour pouvoir récupérer les coordonnées du point sur lequel on clique comme destination.

   Solution :
   Rien de bien particulier, il fallait utiliser les méthodes `getX` et `getY` de la classe `MouseEvent` pour récupérer les coordonnées du point de destination.

License CC BY-NC-SA 3.0

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported license (CC BY-NC-SA 3.0)

You are free to Share (copy, distribute and transmite) and to Remix (adapt) this work under the following conditions:


Attribution – You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).


Noncommercial – You may not use this work for commercial purposes.


Share Alike – If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

See http://creativecommons.org/licenses/by-nc-sa/3.0/ for more details.