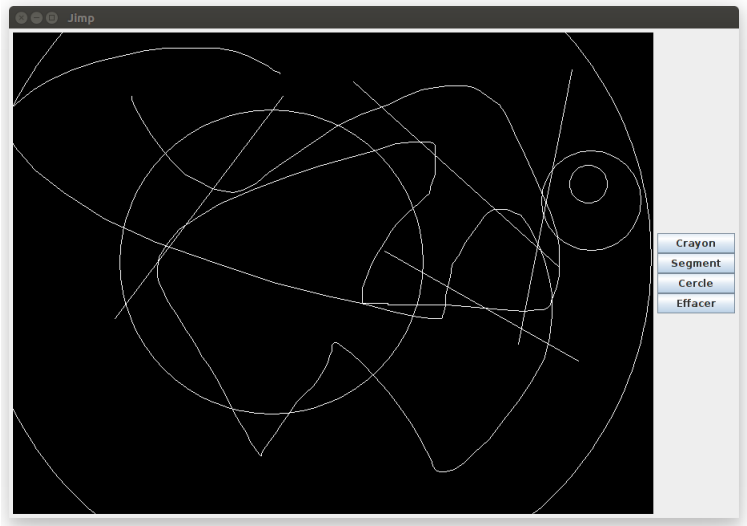
Programmation avancée
TP 6 – Swing !

V. Padovani, PPS - IRIF



Le but de ce TD est décrire un mini-logiciel de dessin à l'aide de la librairie Swing. L'utilisation d'eclipse y est presque indispensable : le nombre d'imports nécessaires est trop important pour se servir d'emacs. L'énoncé est conceptuellement simple – et vous serez beaucoup guidés – mais plutôt long : il peut être fait sur deux semaines (au moins). Chaque texte en bleu foncé est un lien http vers le site d'Oracle.

Il est conseillé de garder en permanence un onglet ouvert sur l'API du langage, en mode "frames". Au besoin, les exemples du tutorial officiel de Swing sont librement consultables et utilisables, à condition de prendre le temps les comprendre et de me poser des questions si nécessaire. La Programmation Par Google [1] est ici hautement déconseillée, étant donnée la quantité énorme sur le web d'informations sur swing mal digérées.

Exercice 1 : Fenêtre et panneaux principaux

1. Créez une classe principale `Jimp` extension de `JPanel`. Dans sa méthode `main`, créez une fenêtre avec un titre (`JFrame`). A l'aide de la méthode `setDefaultCloseOperation` , faites en sorte que la fermeture de cette fenêtre termine l'application.
2. Toujours dans `main`, insérez dans la fenêtre une instance de `Jimp` [2]. Faites en sorte que la fenêtre adapte sa taille à celle de cette instance quelles que soient ses dimensions futures (`pack()`) – c'est-à-dire quels que soient les composants internes ajoutés à cette instance. Dans les questions et exercices qui suivent, nous l'appellerons *panneau principal* de l'application.

1. Chercher, copier, coller.
2. Cet ajout ne se fait pas directement sur la fenêtre, mais sur l'un de ses composants internes, sa "content pane" : `frame.getContentPane().add(jimp)`

3. Testez l'application. Le panneau principal étant vide, les dimensions de la partie intérieure de la fenêtre devraient être réduites à quelques pixels. La fenêtre est néanmoins redimensionnable, et fermable – avec dans ce cas terminaison de l'application.

Exercice 2 : L'ardoise (Board)

L'*ardoise* sera un panneau inséré dans le panneau principal de l'application, dans lequel seront capturés les événements–souris – repercutés en dessins visibles dans l'ardoise.

1. Créez une nouvelle classe `Board`, extension de `JPanel`. Ajoutez à cette classe un constructeur à deux arguments entiers `width` et `height`. Dans ce constructeur, spécifiez la taille préférée de votre ardoise comme étant `width` × `height`, à l'aide de la méthode `setPreferredSize` et de la classe `Dimension`,
2. Dans le constructeur du panneau principal, ajoutez au panneau (`add`) une nouvelle instance de `Board`, par exemple de taille 800 × 600.
3. Testez l'application : la fenêtre devrait avoir ajusté sa taille à celle de l'ardoise (plus une marge de quelques pixels, due à la présence du panneau principal).

Exercice 3 : Echauffement – écouteurs de boutons

Uniquement à titre de test.

1. Dans le constructeur du panneau principal, créez deux instances de `JButton` étiquetées par les chaînes "Hello" et "World !". Ajoutez ces deux boutons au panneau.
2. Toujours dans le constructeur, créez deux objets de classe anonyme implémentant l'interface `ActionListener`. La définition de chaque listener sera de la forme :

```
ActionListener al = new ActionListener () {
  public void actionPerformed (ActionEvent e) {
    // code de la reponse a une action.
  }
};
```

3. Complétez le code des méthodes `actionPerformed` des listeners de manière à ce qu'ils affichent sur la console des messages distincts, par exemple "Bonjour" et "Le Monde !". Ajouter à chaque bouton son écouteur (`addActionListener`).
4. Observez avec bonheur le résultat, puis commentez le code de cet exercice.

Exercice 4 : Echauffement – écouteur de souris

Uniquement à titre de test.

1. Les interfaces `MouseListener` et `MouseMotionListener` regroupent tous les noms de méthodes invoquées automatiquement sur un écouteur de souris lorsqu'il observe un composant donné. La classe `MouseInputAdapter` implémente toutes les méthodes de ces deux interfaces avec des implémentations vides. Allez voir la documentation de ces classes.

2. Dans le constructeur du panneau principal, créez une extension anonyme de `MouseInputAdapter`, redéfinissant à la volée les implémentations vides des méthodes `mousePressed`, `mouseDragged` et `mouseReleased`. Cette définition sera donc de la forme :

```
MouseInputAdapter ma = new MouseInputAdapter() {
  public void mousePressed(MouseEvent e) {
    // code de la reponse a un clic
  }
  public void mouseDragged(MouseEvent e) {
    // code de la reponse a un drag
  }
  public void mouseReleased(MouseEvent e) {
    // code de la reponse a un release
  }
};
```

3. Implémentez chaque méthode en affichant son nom (`"pressed :"`, `"dragged :"` ou `"released :"`) suivi des coordonnées de l'événement `e` (*cf.* `MouseEvent`).
4. Il faut maintenant ajouter cet écouteur à la fois aux écouteurs de clics de souris et aux écouteur de mouvement de souris de l'ardoise. Ecrivez ces deux opérations dans le constructeur du panneau, en invoquant sur l'ardoise `addMouseListener` et `addMouseMotionListener`.
5. Observez le résultat, puis commentez le code de cet exercice.

Exercice 5 : Image interne de l'ardoise

Le dessin dans l'ardoise pourrait se faire directement dans les pixels de ce composant, mais cette solution est bancale : le dessin sera dans ce cas perdu à chaque fois que le composant sera masqué par une autre fenêtre puis redécouvert. Une meilleure solution est de dessiner dans une *image* hors-écran et de modifier la méthode de rendu de l'ardoise pour qu'elle affiche cette image à chaque rafraîchissement.

1. Ajoutez à la classe `Board` un champ de type `BufferedImage`. Dans le constructeur, initialisez ce champ à une nouvelle image (à pixels entiers RGB, sans transparence) de même taille que l'ardoise.
2. Redéfinissez la méthode `paintComponent` de l'ardoise[3]. La nouvelle implémentation devra copier l'image dans la fenêtre, à l'aide de la méthode `drawImage`.
3. Testez l'application : la méthode `paintComponent` étant invoquée au premier affichage de la fenêtre, une image entièrement noire devrait apparaître dans celle-ci.
4. Ajoutez à la classe `Board` un champ de type `Graphics`. Dans le constructeur, initialisez ce champ à un contexte graphique cablé aux pixels de l'image (méthode `getGraphics` de `BufferedImage`).
5. Testez à nouveau l'application, en effectuant dans le constructeur quelques opérations graphiques sur l'image, via son contexte graphique : `drawLine`, `drawOval`, etc. Ces modifications de l'image devraient être visibles au démarrage.

3. Cette méthode est invoqué automatiquement à chaque rafraichissement de la fenêtre, ou encore après invocation explicite sur l'ardoise de sa méthode `repaint()`. Son argument (`Graphics`) est alors un point d'accès aux pixels de la fenêtre – un "contexte graphique"

6. En préparation des étapes suivantes : ajoutez un accesseur pour le contexte graphique de l'image courante (`getImageGraphics()`). Ajouter également une méthode `effacer` permettant d'effacer l'image interne de l'ardoise (méthode `clearRect` de `Graphics`).

Exercice 6 : Ecriture des outils

La hiérarchie des *outils* sera celle des objets chargés d'écouter les événements–souris dans l'ardoise et de répercuter ces évenéments en dessins dans son image interne. Selon l'outil actif, il s'agira de dessin libre, de tracés de segments, ou encore de tracés de cercles.

1. Définissez une classe abstraite `Tool` extension de `MouseAdapter`, contenant un champ de type `Board`[4] et un champ `name` de type `String`. Ajoutez un constructeur permettant d'initialiser ces champs. Ajoutez aussi un accesseur pour `name`.
2. Toujours dans la classe `Tool`, écrivez les méthodes suivantes :
   (a) `void activate()` permettant d'ajouter un contrôleur aux écouteurs de son ardoise associée (`addMouseListener`, `addMouseMotionListener`).
   (b) `void deactivate()` permettant de supprimer un contrôleur des écouteurs de son ardoise associée (`removeMouseListener`, `removeMouseMotionListener`).
3. Definissez une extension concrète `ToolPen` de `Tool` munie d'un constructeur invoquant le super-constructeur précédent. Redéfinissez ses méthodes `mousePressed` et `MouseDragged` de telle sorte que chacune affiche dans l'ardoise associée un simple point aux coordonnées de chaque évènement reçu (il suffit de demander à l'ardoise associée le contexte graphique de son image interne courante, de faire un `drawLine` sur place, puis de demander à l'ardoise associée de se mettre à jour par `repaint()`).
4. Testez cette classe en effectuant dans le constructeur du panneau principal les opérations suivantes :
   – création d'une instance de `ToolPen` liée à l'ardoise (de nom `"Crayon"`) ;
   – activation de ce contrôleur par `activate`.
   Les points tracés devraient suivre les mouvements de la souris lorsque celle-ci est maintenue pressée. Noter que ce tracé est discontinu lorsque les mouvements sont trop rapides. Trouvez le moyen de résoudre ce problème en modifiant les méthodes `mousePressed` et `MouseDragged` de `ToolPen`.
5. Définissez une nouvelle extension `ToolLine` de `Tool` permettant de tracer un segment entre les coordonnées de deux clics de souris (`drawLine`). Testez cette nouvelle classe de la même manière.
6. Définissez et testez une nouvelle extension `ToolCircle` de `Tool` permettant de tracer un cercle : premier clic au centre souhaité, second clic sur le cercle souhaité (trouver le moyen d'effectuer ce tracé avec `drawOval`).

4. Il est raisonnable que les outils ne connaissent que l'ardoise courante auquels ils sont associés, et pas le contexte graphique de son image : ce contexte pourrait varier au cours du temps, par exemple en cas de chargement d'une image.

Exercice 7 : La boîte à Outils (ToolBox)

La *boîte à outils* sera le composant contenant les boutons permettant de basculer d'un outil à l'autre. Ces boutons seront créés par la boîte elle-même, à partir des outils fournis à la première méthode `addTool` ci-dessous – la seconde ne servira qu'à créer le bouton d'effacement, qui n'est pas un outil en lui-même, mais une simple action disjointe du choix de l'outil courant.

1. Definissez une nouvelle extension `ToolBox` de (`JPanel`), munie d'un constructeur à deux arguments entiers `lines` et `columns`. Dans ce constructeur, remplacez l'agencement d'une boîte à outils (`setLayout`) par un `GridLayout` de taille `lines` × `columns`.
2. Toujours dans le constructeur, ajustez la taille d'une boîte de manière à ce qu'elle puisse contenir `lines` lignes de `columns` colonnes boutons de taille suiffsante – par exemple `(100 * columns)` × `(20 * lines)` (noter que `100` et `20` peuvent être définis comme des constantes statiques).
3. Ajoutez à la classe un champ privé `current` de type `Tool`. Ajoutez une méthode privée `void switchTool(Tool tool)` : si `current` n'est pas à `null`, la méthode doit d'abord désactiver l'outil référencé (`deactivate ()`) ; elle doit ensuite activer `tool` (`activate ()`), puis remplacer `current` par `tool`.
4. Ajoutez à la classe une méthode `void addTool(Tool tool, boolean set)`. Cette méthode doit : créer un `JButton` étiqueté par le champ `name` de `tool`, et ajouter ce bouton à la boîte ; créer pour le bouton un `ActionListener` anonyme dont l'action sera d'invoquer la méthode `switchTool` sur `tool` ; si `set` est à `true`, invoquer immédiatement `switchTool` sur `tool`
5. Ajoutez à la classe une méthode `void addTool(String name, ActionListener al)`. Cette méthode doit : créer un `JButton` étiqueté par `name` ; ajouter `al` aux écouteurs du bouton ; ajouter ce bouton à la boîte.

Exercice 8 : Assemblage final

Il ne reste plus qu'à compléter le constructeur du panneau principal, après la création de l'ardoise et son ajout au panneau.

1. Créez une instance de `ToolBox` de taille `4` × 1 et ajoutez-là au panneau.
2. Créez une instance liée à l'ardoise de chaque sorte d'outils ("Crayon", "Segment", "Cercle"). Ajoutez chaque instance à la boîte par la méthode `addTool`, en ajustant la valeur de `set` de manière à activer l'outil–crayon comme outil initial.
3. Créez enfin un `ActionListener` de classe anonyme dont l'action sera d'effacer l'ardoise. Servez-vous de la seconde méthode `addTool` pour ajouter un bouton d'effacement à la boîte.