Université de Nice - Sophia Antipolis
Faculté des Sciences

DEUG MIAS MP1

Programmation 2000-01

8. DESSINER AVEC L'AWT (SUITE)

Dans la feuille 7, vous avez vu comment on construisait une fenêtre [*frame*] contenant un seul composant : une zone de dessin [*canvas*]. La *frame* était une instance de `MP1Frame` est le *canvas* était une instance d'une sous-classe `MP1Canvas` de `Canvas` spécialisée dans un dessin donné, exécuté par les instructions de la méthode `paint(…)`. Nous allons prolonger cette étude en introduisant d'autres composants dans la *frame* : zones de texte [fixes ou modifiables] et boutons à cliquer provoquant une action. Ouvrez *Explorer* sur l'URL : `http://deptinfo.unice.fr/~roy/MP1/8.html`

Nous voudrions par exemple, au lieu de ré-exécuter le programme de l'exercice 6.7, ce qui est fastidieux, disposer de deux boutons en bas de la *frame* : un bouton « Quitter » pour terminer [c'est plus propre qu'une case de fermeture !] et un bouton « Go » pour recommencer une expérience. De plus, nous aimerions avoir les résultats de la simulation [la valeur approchée du nombre π] directement dans une zone réservée de la *frame* et non pas dans cette vilaine petite fenêtre en bas de *RealJ*. Ouf. Programme ambitieux…

1. Ajouter des composants à une fenêtre avec un « Layout Manager »

Notre but est donc de reprendre le programme de Monte-Carlo [exercice 7.7], dont la fenêtre [de type `MP1Frame`] ne contenait qu'un seul composant, un *canvas*. Nous allons rajouter d'autres composants à la fenêtre. Mais comment seront-ils disposés, dans quel ordre ? Ceci est du ressort d'un « gestionnaire de disposition » *[layout manager*]. Il en existe plusieurs, du plus simple `FlowLayout` [qui dispose les composants les uns à la suite des autres] à de plus sophistiqués comme `BorderLayout` [qui les assemble dans 5 régions `North`, `South`, `East`, `West` et `Center`] ou `GridLayout(p,q)` qui divise la fenêtre en une grille matricielle à `p` lignes et `q` colonnes.

Nous commencerons par utiliser la classe `FlowLayout` dont nous disons qu'une de ses instances gèrera la disposition des composants de notre fenêtre `frame` :

```
frame.setLayout(new FlowLayout());
```

A partir de là, il suffit, étant donné un composant `machin`, de l'ajouter à la fenêtre [le *layout manager* le rangera pour vous là où il faut] :

```
frame.add(machin);
```

et une fois que tous les composants seront ajoutés, il suffira de compacter le tout afin de calculer la taille de la fenêtre résultante, et de demander à la fenêtre de se montrer :

```
frame.pack();
frame.show();
```

A partir de là, les composants auront leur vie propre, et réagiront suivant les instructions qu'on leur aura fournies : un *canvas* activera sa méthode `paint(…)`, un bouton attendra patiemment qu'on le clique, etc.

2. La classe « Label » : zones de texte non éditables

La classe `Label` [package `java.awt`] est une sous-classe de la classe `Component`. Une instance de la classe `Label` est donc un composant destiné à être ajouté [`add`] à une fenêtre et contenant une simple ligne de texte :

```
Label label = new Label("Ceci est un texte",Label.CENTER);
```

Les constantes CENTER, RIGHT, LEFT de la classe `Label` permettent de choisir l'*alignement* du texte. Ce texte n'est pas modifiable par l'utilisateur, seulement par le programme, avec la méthode d'instance `setText(…)` :

```
label.setText("Et ceci est le nouveau texte");
```

Vous trouverez les détails dans l'API, classe `Label` du package `java.awt`…

Exercice 8.1 Ecrivez une sous-classe `TP1Exo7Frame` de la classe `MP1Frame`. Dans le constructeur, optez pour un gestionnaire `FlowLayout`, ajoutez deux objets de type `Label`, l'un contenant le texte `"pi ="` en bleu sur fond jaune, et l'autre le texte `"3.02"` en rouge sur fond blanc. Faites afficher le tout dans le `main(…)`. Attendez 5 secondes, puis changez le texte `"3.02"` en `"3.1416"`. Tous les chiffres doivent être visibles…

La solution à compléter est sur la page Web du cours n°7, fichier TP7Exo1Frame.java

Exercice 8.2 Ecrivez une classe `TP7Exo2Frame` étendant la classe `MP1Frame`. Exercice analogue au précédent, mais vous décrirez une fenêtre contenant un *label* avec le texte `"pi = 3.1416"` et un *canvas* contenant un cercle noir inscrit dans un carré de fond jaune de taille 300x300.
La solution à compléter est téléchargeable, fichier TP7Exo2Frame.java

Exercice 8.3 Modifiez le programme précedent. Dans la méthode `paint(…)` du *canvas*, vous procèderez au jet de 10000 points rouges aléatoires. A la fin de cette expérience de Monte-Carlo, vous affichez la valeur de π dans le *label*.
La solution à compléter est téléchargeable, fichier TP7Exo3Frame.java

Faites une copie de votre fichier TP7Exo3Frame.java. Nommez la copie MonteCarloGUI.java et nommez le canvas MonteCarloCanvas. Jusqu'à la fin du TP, vous allez modifier ce fichier…

3. La classe « Button » : les boutons à cliquer

Un *bouton à cliquer* [ne pas confondre avec un bouton-radio ou une case à cocher] est un composant, instance de la classe `Button` qui est une sous-classe de `Component`, comme `Label` et `Canvas`. Un bouton contient un texte court, comme "Go", "Ok", "Quitter", "Annuler", etc.

```
Button button = new Button("Go");
```

Exercice 8.4 Ajoutez un bouton "Go" à la *frame* comme dernier composant, après le *label* et le *canvas*. Ajoutez aussi un bouton "Quitter". Où le gestionnaire les place-t-il ? Les boutons réagissent-ils lorsque vous cliquez dessus ?

• Oui, les boutons réagissent mais il ne font rien, personne ne leur a encore appris quoi faire. Nous pénétrons là dans un domaine intéressant, celui des *objets réactifs et autonomes*. Adaptons la solution que nous avions utilisée pour gérer la case de fermeture dans la feuille 6. Nous avions évoqué à l'époque la notion d'événement : en branchant un écouteur à la fenêtre [instance d'une sous-classe anonyme de `WindowAdapter`], l'événement `e` était passé à la méthode `windowClosing(e)` que nous forcions à faire un `System.exit(0)`.

Et bien, nous reprenons presque la même idée. Lorsque vous cliquez sur un bouton pour provoquer une action, il y a production d'un événement de type `ActionEvent`. Il faut brancher un écouteur au bouton qui va se mettre à l'écoute des clics dans le bouton et réagir dès qu'un clic se produit.

L'interface `ActionListener` du package `java.awt.event` ne contient qu'une seule méthode à implémenter :

```
public interface ActionListener extends EventListener
{  public void actionPerformed(ActionEvent e);
}
```

Reprenez le fichier `MonteCarloGUI.java` dans lequel vous venez d'ajouter les boutons `"Go"` et `"Quitter"`. Après l'instruction `this.add(buttonGo)` qui rajoutait le composant `buttonGo` à la fenêtre, vous enregistrez ce bouton comme écouteur d'évènements action en instanciant une sous-classe anonyme de `ActionListener` :

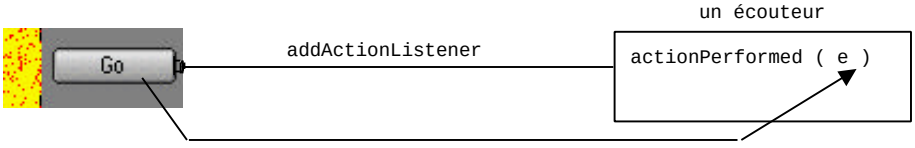
```
buttonGo.addActionListener( new ActionListener()
                            {  public void actionPerformed(ActionEvent e)
                               {  canvas.repaint();
                               }
                            });
```

Enregistrement de l'écouteur

Un écouteur des évènements action

Lorsque l'utilisateur clique dans le bouton `"Go"`, il y a production d'un événement de type `ActionEvent` qui est redirigé vers quiconque est enregistré comme `ActionListener` auprès du bouton, ici l'écouteur anonyme créé par `new`. Cet événement est passé dans le paramètre `e` de la méthode `actionPerformed(…)` qui provoque un ré-affichage [`repaint`] du canvas, c'est-à-dire une ré-activation de sa méthode `paint(…)`.

N.B. On n'invoque jamais la méthode `paint(…)` directement, on passe par `repaint(…)` qui fait un peu de ménage…



Exercice 8.5 Une fois que le bouton `"Go"` réagit correctement, faites de même avec le bouton `"Quitter"`…

4. La classe « `TextField` » : des zones de texte éditables

La classe `Label` fournissait des composants textuels destinés à l'affichage en sortie. La classe `TextField` procure des zones de texte court destinées à être remplies par l'utilisateur. Adieu donc la `Console` qui n'a pas lieu d'être dans une interface graphique moderne. Nous allons demander un *textfield* pour pouvoir modifier le nombre de points jetés au hasard, et un clic sur le bouton `"Go"` permettra de relancer la simulation de Monte Carlo.
Si vous maîtrisez ce TP, vous aurez donc entre les mains un outil puissant et convivial pour concevoir et écrire des programmes avec entrées, sorties et graphisme. Il faudra vous entraîner à construire vous-même une petite interface graphique sans avoir recours à vos notes, ce sera bien évidemment une question d'examen.

Exercice 8.6 Ajoutez un *textfield* à la fenêtre, avec une valeur par défaut de 10000. Regardez au besoin la doc de l'API pour savoir comment on fait…

Le *textfield* n'est pas actif, on peut entrer des valeurs mais tout le monde s'en moque. Ici, il y a deux manières de programmer :

- soit on procède comme avec les boutons, et on enregistre un écouteur auprès du *textfield* qui lancera un `repaint()` sur appui de la touche *Entrée*
- soit on le laisse complètement passif et lors d'un clic dans le bouton `"Go"`, on demandera le contenu du *textfield*. Nous allons procéder de cette manière pour changer [mais libre à vous, en-dehors du TP…].

Exercice 8.7 a) Quelle est la méthode d'instance de la classe `TextField` qui permet de récupérer le contenu d'un *textfield* ? Quel est le type de ce contenu ?

b) Cherchez dans la classe `Integer` la méthode `parseInt(…)`. Que fait-elle ? Quel est le rapport avec la question a) ?

c) Modifiez dans la méthode `paint(…)` du *canvas* l'initialisation de la variable représentant le nombre total de points en lisant le contenu du *textfield. Il vous faudra peut-être modifier un peu la classe `MonteCarloGUI` pour pouvoir depuis le canvas accéder au textfield. Mais n'oubliez pas que le canvas a connaissance de la frame dans laquelle il est installé !*

5. Un « BorderLayout » : vers une meilleure gestion de l'espace !

Le programme est complet et opérationnel, mais la disposition des composants, laissée à l'appréciation de `FlowLayout`, n'est guère esthétique. Vous allez changer de gestionnaire de disposition et opter pour un `BorderLayout`. Celui-ci divise la fenêtre en 5 zones, à la chinoise [*fig. 1*] :

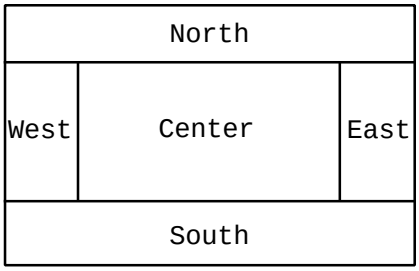


fig. 1

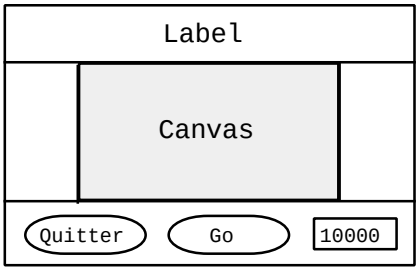


fig. 2

Une fois un objet de type `BorderLayout` créé et mis à la disposition de la fenêtre pour en gérer les composants : il est facile de l'utiliser. Par exemple, pour déposer le label au Nord :

```
this.add(label,"North");          // cf. la doc de l'API
```

Vous mettrez par exemple les boutons à gauche et à droite, et le *textfield* en bas, pour voir : pas très joli…

Plus agréable, la *fig. 2* ci-dessus montre la disposition finale souhaitée. Mais problème : les deux boutons et le *textfield* doivent être dans la zone Sud ! Comment faire ? Vous vous documenterez sur la classe `Panel` [panneau] dans l'API, qui permet de construire un nouveau composant à partir d'autres composants [comme la loi o construit une fonction composée à partir de deux fonctions]. Une fois le panneau à 3 composants construit, il suffira de le placer au Sud…

Exercice 8.8 ***Nous vous laissons terminer ce TP comme travail personnel en-dehors de la séance, et comme préparation au contrôle continu numéro 2, qui consistera à construire une interface graphique avec MP1Frame, Canvas, Label, Button et TextField, et qui proposera une autre expérience que la simulation de Monte-Carlo que vous aurez menée à terme. Vous aurez droit lors du contrôle à la doc en ligne de l'API, sachez donc l'utiliser vite !***