INF224
Travaux Pratiques Java/Swing

Eric Lecolinet - **Télécom ParisTech** - **Dept. INFRES**

Liens utiles

- **Page de INF224**, **transparents du cours**
- **TP C++**, **ancien TP Swing (plus complet)**
- **Documentation : Swing SE 8**, **Java SE 8**, **Swing SE 7**, **Java SE 7**
- **Tutoriels : Tutoriels Oracle**, **the Really Big Index**, **page de sites Java**
- **Démos : Swing**, **Java2D**

Exercices

Le but de cet exercice est de créer une interface graphique Java/Swing qui permettra à terme d'interagir avec le logiciel déjà créé lors du **TP C++/Objet**. Comme précédemment, ce programme Java sera réalisé par étapes en ajoutant les fonctionnalités nécessaires petit à petit. Pensez à lire les "notes" ou "remarques" qui vous donneront des indications utiles avant de faire chaque étape. Vous pouvez utilisez l'IDE de votre choix pour programmer.

1ere Etape: Fenêtre principale et quelques interacteurs

Créez une fenêtre principale (dérivant de **JFrame**) contenant une zone de texte multi-lignes (JTextArea) et trois boutons (JButton). Faites en sorte que, lorsqu'on les active, les deux premiers boutons ajoutent une ligne de texte (différente pour chaque bouton) à la zone de texte et que le dernier bouton termine l'application (cf. méthode exit(int)() de la classe **System**).

Notes :

- Un JFrame est par défaut associé à un LayoutManager de type **BorderLayout**. Ceci permet un positionnement de ses enfants de type "points cardinaux" en appelant la méthode add() avec les arguments adéquats (voir la doc. de BorderLayout). Mettez par exemple la zone de texte dans la zone centrale et les boutons dans un conteneur JPanel lui-même situé dans la zone sud du JFrame.
- Pour spécifier le comportement des boutons quand on les active vous pourrez vous inspirer de la "2e ou 3e stratégie" vues en cours (si vous avez le temps, faites les deux pour comparer).
- Pensez à donner une taille suffisante au **JTextArea** en spécifiant un nombre de lignes et de colonnes adéquats à sa création.
- N'oubliez pas d'appeler les méthodes suivantes de JFrame:
  - setDefaultCloseOperation(int) pour que la fermeture de l'interface entraîne la terminaison de l'application
  - pack() pour calculer la disposition spatiale des composants graphiques
  - setVisible(boolean) pour faire apparaître l'interface
- Votre classe devra comprendre une variable serialVersionUID définie comme suit:
  ```
  private static final long serialVersionUID = 1L;
  ```
  Cette variable est réclamée par le compilateur pour préciser la version de la classe (1 dans ce cas). De plus cette classe doit etre publique et le fichier qui la contient doit avoir le même nom.
- Documentez votre code au fur et à mesure de manière à pouvoir générer automatiquement la documentation grâce à JavaDoc ou Doxygen.

Lancez votre programme, cliquez plusieurs fois sur les deux premiers bouton, retaillez la fenêtre. Que constate-t'on ?

En fait, il est généralement nécessaire d'ajouter des ascenseurs à un JTextArea pour le rendre réellement utilisable. Pour ce faire, mettez ce composant dans un **JScrollPane** (en utilisant le constructeur adéquat de JScrollPane).

2eme Etape: Menus, barre d'outils et actions

Nous allons maintenant rajouter une barre de menus (**JMenuBar**) comprenant un menu déroulant (**JMenu**) ainsi qu'une barre d'outils (**JToolBar**). Il faudra, comme de juste, placer la barre d'outils dans la zone nord de la fenêtre principale. La barre de menus sera ajoutée à la fenêtre via sa méthode setJMenuBar().

Le menu déroulant et la barre d'outils contiendront tous les deux les mêmes boutons que précédemment:

- Une première solution consiste à procéder de la même manière qu'à l'étape précédente (à ceci près que les menus doivent contenir des JMenuItems et non des JButtons). Mais cette technique n'est pas particulièrement optimale car elle conduit à dupliquer les boutons qui sont dans les menus déroulants et ceux qui sont dans la barre d'outils alors qu'ils font la même chose.
- Une autre possibilité (encouragée) est d'utiliser les Actions, ou, plus exactement, de créer des sous-classes de **AbstractAction**. Contrairement aux autres composants de Java Swing, les Actions peuvent être inclues simultanément dans plusieurs composants graphiques (dans notre cas, à la fois dans un JMenu et un JToolbar). Cette classe permet de spécifier toutes les caractéristiques d'un bouton (son nom et, optionnellement, une image, un mnémonique, un raccourci clavier, l'état du bouton s'il en a un) ainsi que son comportement en redéfinissant sa méthode actionPerformed().

Notes

- On rappelle qu'il ne doit y avoir qu'une seule classe publique dans un fichier Java. Cependant vous pouvez définir d'autres classes non publiques si cela s'avère nécessaire.
- Pour les utilisateurs de MacOSX : pour que les barres de menus apparaissent comme d'habitude sur cette plate-forme il faut faire:

```
System.setProperty("apple.laf.useScreenMenuBar", "true");
```

3eme Etape: Interaction client/serveur

Cette étape vise à faire communiquer votre programme Java avec le programme C++, faisant office de serveur, que vous avez réalisé aux TPs précédents. Tout d'abord, téléchargez le programme **Client.java**. Ce programme fait la même chose que le programme client.cpp vu **à la question 11 du TP C++**. Compilez le et vérifiez qu'il interagit correctement avec le serveur C++.

Adaptez le code que vous avez écrit à l'étape précédente pour créer une interface graphique qui interagisse avec le serveur C++. Les fonctionnalités souhaitées sont les mêmes qu'à la question 13 du TP C++ : pouvoir rechercher un objet multimédia (l'affichage devant se faire sur l'interface graphique Java Swing qui joue le rôle de télécommande), et pouvoir "jouer" un objet multimédia (sur le serveur C++, qui fait office de "set-top box").

Notes

- Vous pouvez bien sûr rajouter toute commande qui vous semble utile pour une télécommande !
- Si vous avez le temps, vous pouvez améliorer l'interface pour la rendre plus facile à utiliser en exploitant les nombreuses possibilités offertes par Java Swing. Outre des **JTexField** vous pourrez par exemple utiliser des onglets (**JTabbedPane**) ou des boîtes de dialogue (**JDialog**) ou encore des boutons exclusifs (cf. **JRadioButton** et **ButtonGroup**).

4eme Etape (obligatoire): Créer un Makefile

Vous avez probablement utilisé un IDE qui a créé des fichiers un peu partout. C'est bien pour développer mais pas forcément pour déployer sur d'autres machines et encore moins ... pour corriger !

On vous demande dans cette question, d'adapter ce **Makefile** qui va permettre de **compiler puis d'executer** automatiquement le programme Java (la télécommande de l'étape 3) juste en tapant make run dans le Terminal.

Pour rendre les TPs

1. Vous devez créer trois répertoires organisés comme suit :

- un **répertoire principal** qui doit s'appeller **Nom__Prenom** (les vôtres bien sûr \ ;-). Si votre nom ou votre prénom a des espaces, mettez un _ à la place.
- un **sous-répertoire** qui doit s'appeller **cpp** et qui doit contenir tout ce qui concerne la partie C++. Le code source doit se trouver dans ce répertoire (pas un sous-répertoire !). De même, la compilation doit produire l'exécutable dans ce répertoire.
- un **sous-répertoire** qui doit s'appeller **swing** et qui doit contenir tout ce qui concerne la partie Java Swing. Comme précedemment, le code source doit se trouver dans ce répertoire et l'exécutable doit être généré dans ce répertoire.

2. Chacun des deux programmes (C++ et Swing) doit pouvoir être **compilé et executé sur une machine Unix de l'Ecole** en allant dans le répertoire correspondant et **en tapant "make run" dans le Terminal** :
   - Les **deux** parties (C++ et Java) doivent donc obligatoirement avoir un **Makefile**.
   - Le programme C++ ne doit pas planter et le programme Java ne doit pas produire de "Null pointer exception".
   - Noter qu'un programme C++ faux peut sembler tourner correctement sous Windows mais planter sous Linus et vice-versa (dans les deux cas il y a une erreur de mémoire, on peut par exemple utiliser la commande **valgrind** pour les détecter).

   Un programme sans Makefile, qui ne compile pas ou qui plante sur une machine de l'Ecole sera considéré non rendu !

3. Documenter votre code en utilisant **Doxygen** et écrire un bref fichier **README** commun aux deux parties précisant quelles questions ont été traitées et donnant les réponses aux questions ainsi que tous les commentaires que vous jugerez utiles.

   Ce fichier doit être en PDF, en HTML (en UTF-8) ou au format texte (impérativement en UTF-8\ ). Il doit être placé dans le répertoire principal. **N'oubliez pas de mettre votre nom**.

4. Enlever les fichiers .o, .class et les executables puis zipper votre répertoire principal, soit au format .zip, soit au format .tar.gz (seuls formats acceptés !)

5. Allez à la page **0EL11** du **Moodle IPP** (attention à choisir la bonne période) puis téléchargez votre fichier zip ou tar.gz

CHECK LIST : ATTENTION A BIEN VERIFIER :

- que vous avez bien sélectionné la **bonne UE et la bonne période**
- que vous n'avez-pas zippé un répertoire vide, ni le répertoire de votre compte Unix, ni votre boîte mail (c'est déjà arrivé !)
- que c'est bien la **dernière version** et qu'il ne manque pas des fichiers
- que votre code **compile et s'exécute** correctement

Eric Lecolinet - http://www.telecom-paristech.fr/~elc - Telecom ParisTech