

TD5 – Introduction au JavaBean

Ce TD est une première approche des JavaBeans. On y crée notre premier Bean, élémentaire. Pour en savoir plus, consultez le document « Guide du développeur JavaBean » [GDD].

INTRODUCTION

Selon la définition de Sun, un JavaBean est un « *composant logiciel réutilisable utilisable visuellement dans un outil de conception* ». Pour cela, une série de conventions devront être respectées.

On cite souvent le cas des composants graphiques (comme ceux de Swing) pour lesquels un environnement de développement intégré (comme Netbeans) offre des outils pour faciliter le développement. Ainsi, dans ces environnements, on peut glisser-déposer les composants sur un canevas et les configurer aisément sans taper de ligne de code (le code est généré automatiquement pour nous). La liste des composants et leurs propriétés n'est pas « *hardcodée* » dans l'IDE mais appris par *interrogation automatique* des composants.

Notons toutefois que cette notion de composant existe aussi pour des éléments non graphiques comme nous le verrons aussi.

UN BEAN SIMPLE

Théorie

Toute classe peut à priori être définie comme un JavaBean. Toutefois, une série de règles syntaxiques et conceptuelles doivent être respectées. Pour votre premier Bean, vous n'avez qu'à respecter les suivantes :

1. Un Bean doit impérativement proposer un constructeur public par défaut (sans paramètres)
2. Un Bean doit implémenter l'interface `Serializable` ou hériter d'un composant qui l'implémente (ce sera le cas de tous les composants Swing)
3. Un bean possède des attributs qui sont identifiés par un couple de getter/setter.

Mise en pratique

Nous allons créer un bean représentant un simple LED.

- Créez un nouveau projet et une classe de type `JPanel` que vous appellerez « `Led` ».
- Créez la propriété « `couleur` » de type `Color` et la propriété « `allumé` » de type `boolean` (attention aux conventions; n'oubliez pas d'utiliser les facilités de l'éditeur !)
- Redéfinissez la méthode `paint` (`void paintComponent(Graphics g)`) pour tracer un rond au centre du panel en fonction des propriétés « `couleur` » et « `allumé` ».



- Affichez le dans une Fenêtre pour vérifier son bon fonctionnement. Un bouton « On/off » fera basculer son état « allumé/éteint ».
- Vous pouvez aussi ajouter votre nouveau composant à la palette de Matisse (clic droit sur le javabean / Tools / Add to palette...)

LE BEAN COMMUNIQUANT

Théorie

Un bean n'a évidemment aucun intérêt s'il ne dispose pas de moyen de communiquer avec l'extérieur. Essentiellement, un composant doit pouvoir avertir d'autres composants intéressés de tout changement dans son état.

Un changement intéressant est celui affectant une propriété. On parle de propriété « liée » lorsqu'un changement de sa valeur entraîne une notification aux écouteurs.

Afin de simplifier la tâche du programmeur, le framework des JavaBeans propose une classe (`java.beans.PropertyChangeSupport` [GDD]) permettant de gérer l'inscription de composants intéressés (« écouteurs ») par le changement de valeur d'une propriété. Remarquez qu'un écouteur ne pourra pas spécifier quelle propriété précise il veut écouter. Il sera averti lors de tout changement d'une propriété liée mais il lui sera évidemment possible de savoir quelle propriété précise a été modifiée.

Mise en pratique

Nous allons permettre à d'autres composants d'écouter tout changement des propriétés définies plus haut.



- Normalement, il vous faudrait définir un objet de type `PropertyChangeSupport`, ainsi que deux fonctions permettant d'ajouter ou de retirer un écouteur. Mais comme vous héritez d'un composant Swing, tout ceci est déjà hérité.
- Dans les « modificateurs » d'attributs, il vous faudra appeler la fonction `firePropertyChange` pour avertir tous les composants inscrits du changement.

Pour vérifier le bon fonctionnement de votre composant, on vous demande de coder la petite application suivante :

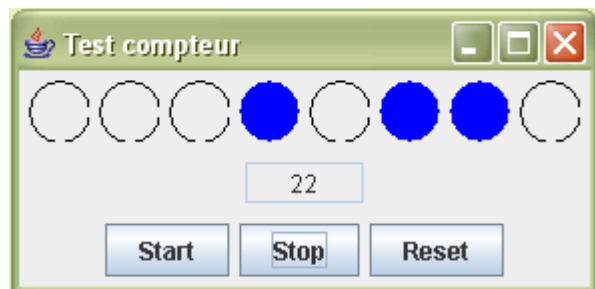
- La fenêtre affiche 2 leds (appelons-les A et B). Le led A est vert et le B est rouge. Un est éteint et l'autre allumé.
- Un bouton « Inverser » est présent. Lorsqu'on clique dessus, il bascule le led A.
- Un observateur du led A va permettre la synchronisation. Lorsqu'il bascule (passe de « allumé » à « éteint » ou inversement) il met le led B dans l'autre état. Il y aura ainsi toujours un led allumé et l'autre éteint.
- Pour aller plus loin, on propose de remplacer le bouton par un timer qui bascule l'état du led A (regardez la classe « Timer » de Swing).

APPLICATION : UN COMPTEUR BINAIRE

Écrivons un bean représentant un compteur binaire avec une série de LED.

Les propriétés sont :

- Le nombre de leds utilisés
- La valeur actuellement représentée (une propriété liée)
- Une propriété « timer », un entier qui indique le nombre de millisecondes entre 2 avancements du compteur..



Les méthodes sont :

- `reset()` qui réinitialise le compteur à 0.
- `start()` qui démarre le compteur à partir de la valeur courante.
- `stop()` qui arrête le compteur.

En ce qui concerne l'affichage et pour s'habituer à la gestion des propriétés liées, le led de poids faible sera contrôlé par le timer alors que chacun des autres sera un observateur de celui qui se trouve à sa droite et s'inversera lorsque le précédent s'éteindra.

Pour le tester, écrivez une fenêtre reprenant un compteur binaire, 3 boutons (reset, start et stop) et un textfield affichant la valeur du compteur en décimal.