

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

Un JavaBean¹ est un composant Java (une classe) réutilisable et automatiquement reconnaissable et manipulable par un environnement de développement.

Pour qu'une classe Java soit un JavaBean elle doit respecter certaines conventions:

- implémenter `Serializable`
- proposer un constructeur sans argument
- à moins de se présenter dans un `BeanInfo`², elle ne peut exposer ses propriétés privées qu'au travers de méthodes publiques (accesseurs) dont les noms commencent par 'get' ou 'set' suivi du nom de la propriété dont la première lettre est écrite en majuscule (`private type maPropriété` sera donc éventuellement accessible par `public type getMaPropriété()` et/ou `public void setMaPropriété(type p)`. Attention, si `type` est `boolean` ou `Boolean`, le getteur s'écrira `public boolean isMaPropriété()`).
- elle doit être totalement indépendante de la plateforme
- elle doit prévoir les notifications nécessaires d'évènements

On cite souvent comme exemples de JavaBeans les composants graphiques (comme ceux de Swing) pour lesquels un environnement de développement intégré (comme Netbeans) offre des facilités graphiques de mise en œuvre (cf. Matisse). 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 consultation des `BeanInfo` ou, à défaut, par *interrogation automatique*³ des composants.

Ne croyons pas qu'un JavaBean est nécessairement un composant visuel même si les seuls que nous ayons utilisés actuellement sont des composant Swing (`JButton`, `JTextField`, ...).

UN BEAN SIMPLE

Nous allons créer un bean représentant un simple LED⁴.

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 !)

1 Définition de Sun: ...JavaBeans... are reusable software programs that you can develop and assemble easily to create sophisticated applications.

2 `BeanInfo`: classe de description d'un javaBean, nous présenterons cette notion ultérieurement si nous en avons le temps

3 cf. introspection que nous aborderons prochainement

4 LED: light-emitting diode ou, en français, diode électroluminescente.

- Redéfinissez la méthode `paint` (`void paintComponent(Graphics g)`) pour tracer un rond au centre du panel en fonction des propriétés « couleur » et « allumé ».
- Prévoyez une méthode publique `inverser()` qui inverse l'état du Led.
- Affichez le LED 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 `javabeans / Tools / Add to palette...`)

LE BEAN COMMUNIQUANT

Théorie

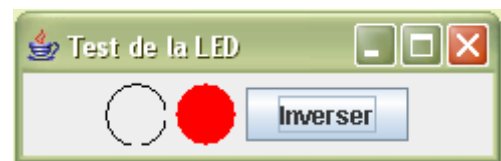
Un bean n'a que rarement un intérêt s'il ne dispose pas de moyen de communiquer avec l'extérieur et offrira généralement la possibilité à des composant extérieurs d'être à l'écoute de changements de son état.

Un changement intéressant est celui affectant une propriété. On parle de propriété « liée » (*bound property*) 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 ou plusieurs propriétés.

Mise en pratique

Nous allons permettre à tout objet implémentant l'interface `PropertyChangeListener` d'être à l'écoute de changements des propriétés liées.



- 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 écouteurs 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 quand il est allumé et le B est rouge. Un est éteint et l'autre allumé.
- Un bouton « Inverser » est présent. Lorsqu'on clique dessus, le led A s'inverse.
- 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

Nous allons utiliser le JavaBean créé au point précédent (n'en modifiez aucune ligne!) pour en définir un nouveau [CompteurBinaire].



Les propriétés liées sont :

- nbLeds : entier donnant le nombre de leds apparents (la valeur est limitée à $[4, 16]^5$). Lors d'un changement de nombre de leds.
- valeur : entier donnant la valeur représentée en binaire (une propriété liée)[cette propriété est read-only: elle n'offre pas de setter].
- délai: entier qui indique le nombre de millisecondes entre 2 avancements du compteur (la valeur est limitée à $[100, 10000]$).
- timerStarted : booléen indiquant si le timer est actif ou non[read-only].

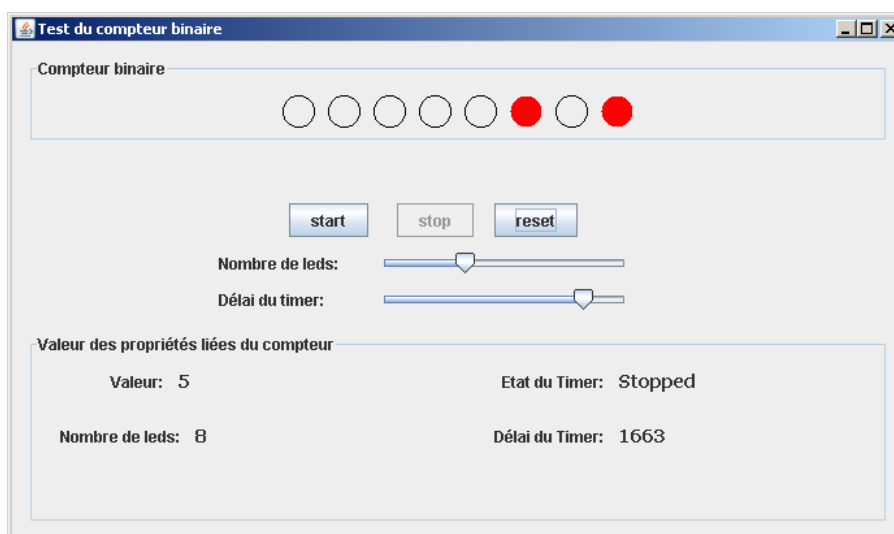
Les méthodes sont :

- 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 [en pratique ce mode de réalisation n'a guère de sens], le led de poids faible sera contrôlé par le timer alors que chacun des autres s'inversera lorsque celui à sa droite s'éteindra.

Test du JavaBean:

Pour tester le compteur binaire, écrivez une fenêtre reprenant un compteur binaire, 3 boutons 'reset', 'start' et stop [les boutons start et stop ne peuvent pas être actifs ensemble], 2 JSlider pour permettre de faire évoluer le nombre de leds et le délai du timer. De plus, prévoyez de permettre de visualiser l'état des différentes propriétés liées.



5 Quel type d'exception envisagez-vous de lancer si cette contrainte n'est pas respectée?