Séance d'ED : Mécanisme des traitements d'événements en langage Java : Java Beans

Programmation d'interface graphique
I - Première Partie

Rappeler le mécanisme de délégation proposée par la version 1.1 (et suivantes) de Java SE pour traiter les événements graphiques.
réponse : cette technique consiste à indiquer qu'un objet graphique est écouté par un objet d'une autre classe. Cet autre objet lancera une méthode convenue par Java lors de l'action spécifique de l'utilisateur sur cet objet graphique. Plus précisément on utilise un code comme :

```
import java.awt.*;
import java.awt.event.*;

•••

Button bt = new Button("Mon Bouton");
EcouteurBouton ecouteur = new EcouteurBouton();
bt.addActionListener(ecouteur); // l'ecouteur est associe au bouton
•••
class EcouteurBouton implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        System.out.println("Coucou");
                }
}
```

Ici au bouton repéré par la référence `bt` est associé l'objet repéré par la référence `ecouteur`. Lorsque l'utilisateur clique sur le bouton, la machine virtuelle Java lance la méthode `actionPerformed()` de l'objet repéré par `ecouteur` (et cette méthode affiche la chaîne "Coucou").

II - Seconde Partie
Construction d'une interface en Java

On veut écrire en langage Java un outil qui présente l'interface suivante :

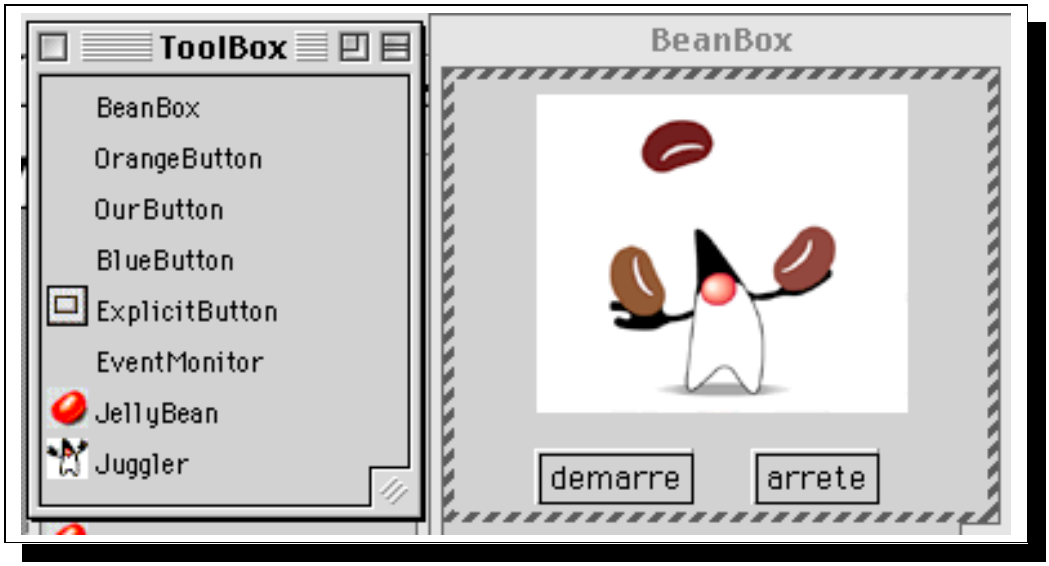


Figure 1

c'est à dire deux fenêtres, l'une de titre Toolbox, l'autre de titre BeanBox. L'utilisateur de cet outil sélectionne l'un des objets proposés par la fenêtre ToolBox et les déposent à l'intérieur de la fenêtre BeanBox (en hachuré) par "glisser-déposer". Par exemple on a déposé, pour obtenir la vue ci dessus, deux objets de la classe `ExplicitButton` (deux boutons poussoir "`demarre`" et "`arrete`") et un objet de l a `Juggler` (un jongleur).

4°) *(1 point)*

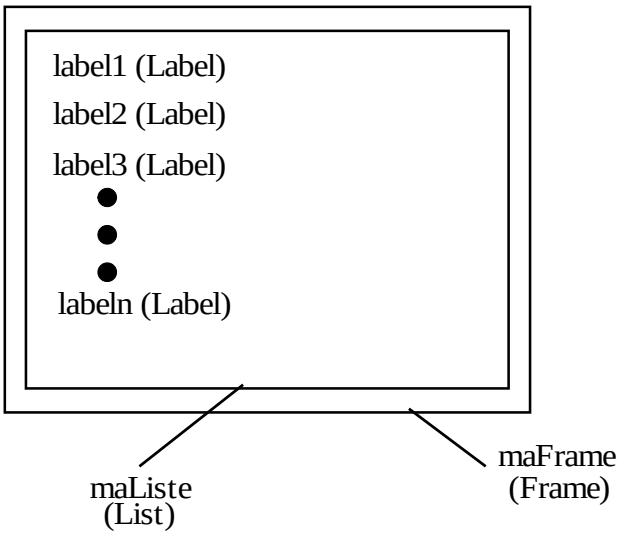
Les deux fenêtres ToolBox et BeanBox sont des fenêtres habillés par le gestionnaire de fenêtre. Quelle est la classe du langage Java qui permet d'obtenir de telles fenêtres ?

C'est la classe `java.awt.Frame` ou plus simplement la classe `Frame` du paquetage `java.awt`.

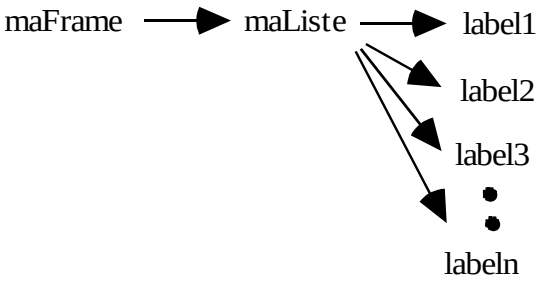
5°) *(2 points)*

Comment le programmeur doit-il s'y prendre pour faire apparaître une fenêtre comme la fenêtre Toolbox ? (on ne demande pas une suite de code mais plutôt les techniques de programmation des interfaces graphiques). On indiquera une arborescence de composants graphiques.

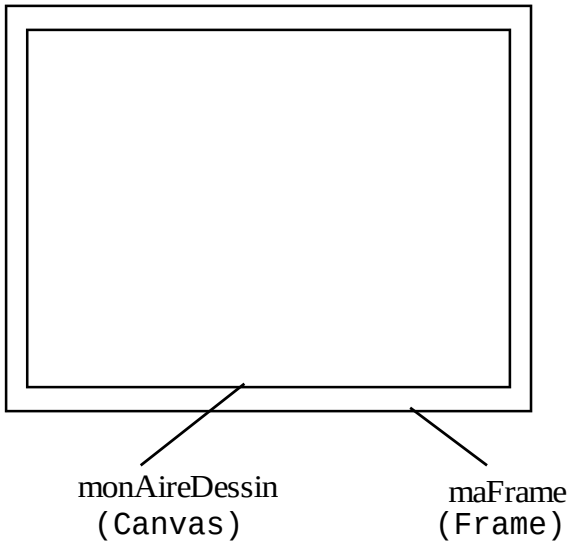
Il faut ajouter dans cette frame, des composants graphiques. On peut envisager que chaque ligne est un composant graphique comme un Label et que l'ensemble de ces labels soient mis dans une liste (c'est une réponse acceptable). On aurait donc une architecture de composants comme :



Ce qui donne une arborescence de composants comme :

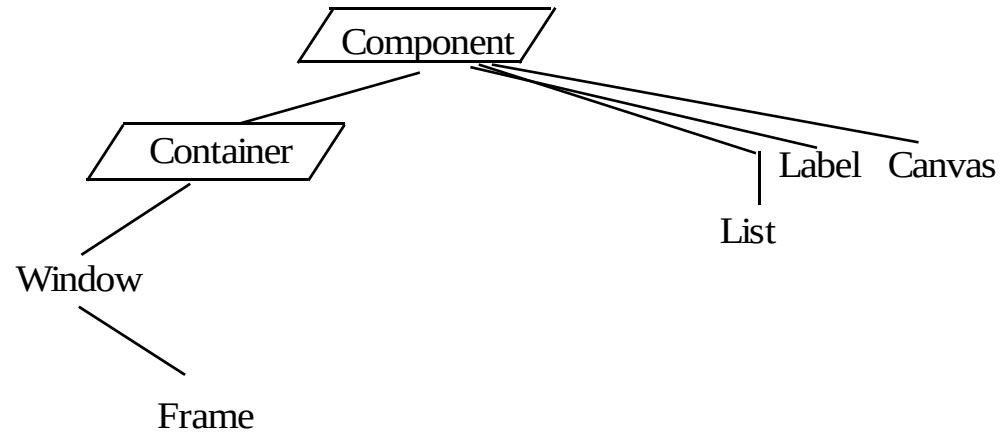


Mais cette liste comporte en fait des couples (une image, un texte). Il faudrait que la bibliothèque de composant dispose de composants listes qui permettent d'insérer de tels couples. Ce n'est pas encore le cas en Java 1.1 mais cela risque de le devenir (composants Swing ?!). En fait, quand on regarde la solution qui a été adoptée, une aire de dessin (`Canvas`) a été mise, qui contient des dessins des images et des chaînes de caractères ("dessinées par des méthodes comme `drawImage()` et `drawString()`). On a donc :



Ce qui donne une arborescence de composants comme :

6°) ***(1 point)***
Indiquer les différentes classes de composants graphiques utilisées pour construire la fenêtre ToolBox et ranger ces classes dans une arborescence d'héritage. Cette arborescence est elle égale à celle que vous avez donnée en réponse à la question précédente ?
En reprenant les diverses classes citées à la question précédente, ces classes sont rangées dans l'arborescence donnée par les concepteurs de Java. On a, comme arborescence des ces classes :



(entre autre `Image` n'est pas un composant graphique). Cette arborescence de classes est différente de l'arborescence des composants graphiques donnée question précédente.

III - Troisième Partie
Les mécanismes des Java Beans

On veut désormais relier les différents objets qui ont été déposés dans la fenêtre BeanBox : on dira désormais dans la BeanBox. Par exemple on veut, à partir des composants déposés dans la BeanBox comme indiqué par la Figure 1, construire un environnement de sorte qu'en cliquant sur le bouton `demarre`, l'objet `Juggler` fasse défiler 3 images qui montrent la mascotte Duke en train de jongler, et en cliquant sur le bouton `arrete` fait arrêter le jonglage (i.e. le défilement des trois images).

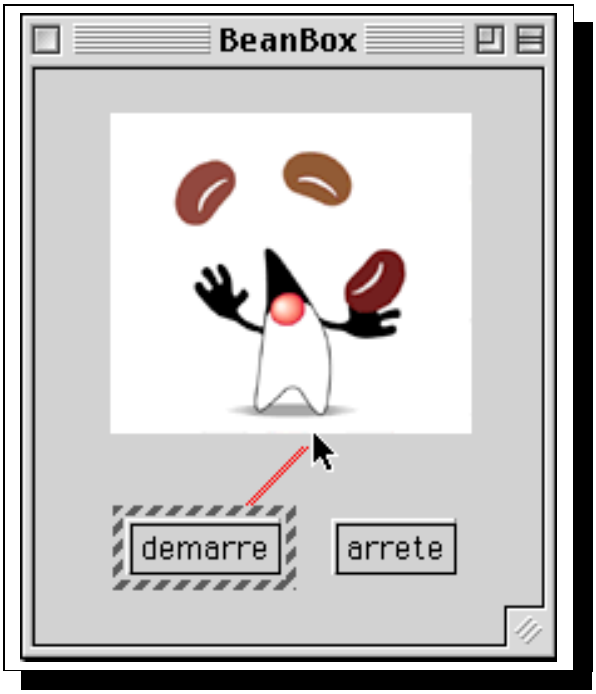
Remarque fondamentale :

On veut assurer cette liaison sans changer quoi que ce soit aux codes des classes `ExplicitButton` ou `Juggler`. Ce sera donc à l'environnement (la Beanbox) de construire tout ce qu'il faut pour assurer ces liaisons.

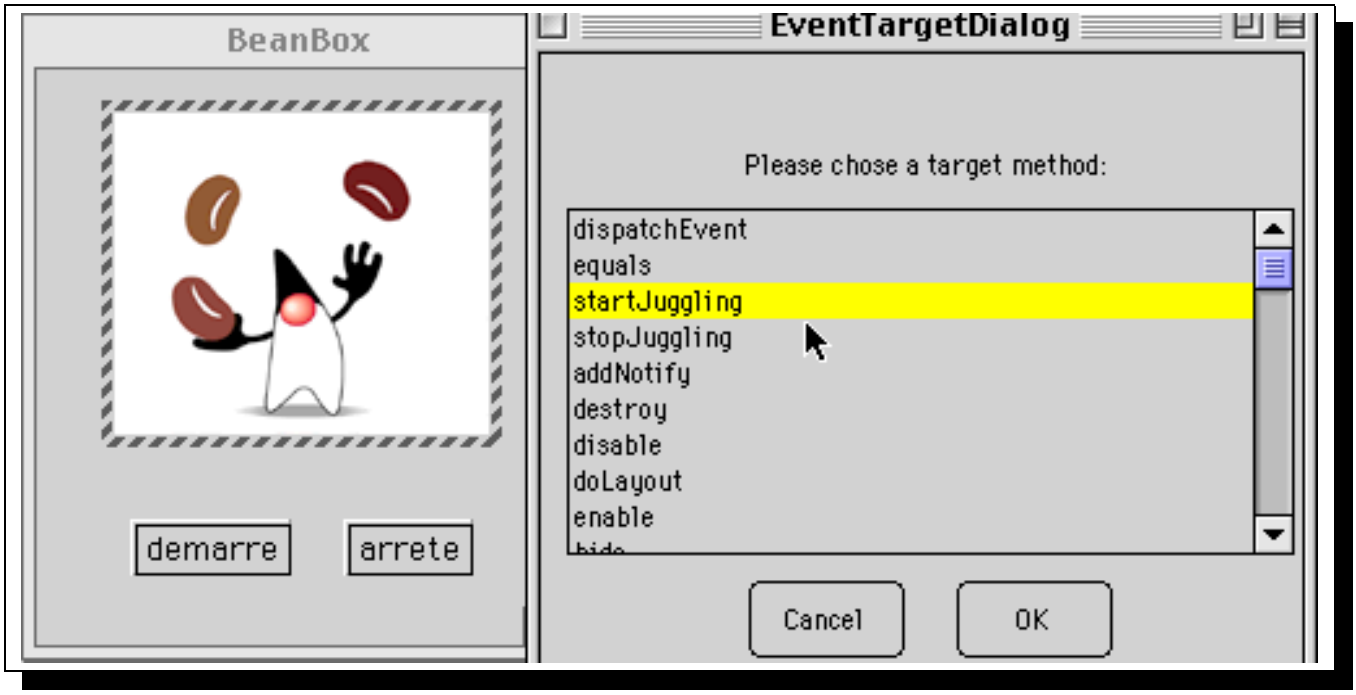
La BeanBox garde trace des objets qui ont été déposés. Lorsqu'on sélectionne un objet dit Bean source, Java nous permet de connaître les écouteurs qui sont susceptibles d'être associés à ce bean source et donc susceptibles d'être déclenchés (par le mécanisme d'introspection qu'on ne demande pas d'étudier ici).

Ce mécanisme d'introspection est aussi utilisé pour connaître les méthodes des "Beans but". Par exemple la beanbox affiche le fait que les objets de la classe `Juggler` possède une méthode `startJuggling(...)` et on se doute que cette méthode lance l'animation de jonglage.

L'interface graphique de la BeanBox permet à l'utilisateur d'indiquer quel est l'objet bean associé (l'écouteur donc) du bean source grâce à un mécanisme de "droite élastique".



(Ci dessus l'objet bouton `demarre` est associé au `jongleur`).
Ensuite un fenêtre est affichée indiquant les méthodes susceptibles d'être lancées. Par exemple ci dessous la méthode `startJuggling()`.



Les deux objets "bouton `demarre`" et le jongleur sont alors reliés par la beanbox et pour réaliser cette liaison, une classe auxiliaire (de nom unique et construit par la beanbox par exemple `___Hookup_14e8012560`) est créée et compilée. Voici l'essentiel du code d'une telle classe :

Fichier `___Hookup_14e8012560.java`

```
import sunw.demo.juggler.Juggler;
import java.awt.event.ActionListener;
```

```
public class ___Hookup_14e8012560 implements
java.awt.event.ActionListener{

    public void setTarget(sunw.demo.juggler.Juggler t) {
        target = t;
    }

    public void actionPerformed(java.awt.event.ActionEvent arg0) {
        target.startJuggling(arg0);
    }

    private sunw.demo.juggler.Juggler target;
}
```

La beanbox construit ensuite un objet `objHook` de cette classe ___Hookup_14e8012560.

7°) ***(1 point)***

Que se passe t'il lorsque la méthode `actionPerformed()` de cette classe est lancée sur l'objet référencé par `objHook` ?

Cette méthode lance la méthode `startJuggling()` sur l'objet `target` qui a été initialisé par `setTarget()`. Intuitivement cette méthode lance l'exécution du jonglage de l'objet de la classe `Juggler`.

8°) ***(1 point)***

Cette classe implémente l'interface `ActionListener`. Indiquer à quel objet parmi ceux visualisés Figure 1 et déposés dans la beanbox, l'objet écouteur référencé par `objHook` doit être associé.

Cet objet `objHook` doit être associé au bouton "`demarre`". Ainsi lorsqu'on clique sur le bouton démarre, `actionPerformed()` de `objHook` est lancé qui lance alors l'exécution du jonglage de l'objet de `Juggler`.

9°) ***(1 point)***

Juste après la création de l'objet référencé par `objHook`, la méthode `setTarget()` a été lancé sur cet objet. A quoi cela a t-il servi ?

La méthode `setTarget()` permet de repérer l'objet but (target !!) sur lequel sera lancée la méthode appropriée. Cette méthode à lancer est liée à une action sur le bean source. Ici cet objet but est le jongleur (objet de la classe sunw.demo.juggler.Juggler)

10°) ***(2 points)***

Indiquer le type de code généré par l'environnement BeanBox (on ne demande pas d'écrire du code Java) pour que lorsqu'on clique sur le bouton `demarre` l'objet `jongleur` se mette à jongler. On indiquera la suite des instructions généréees juste après la sélection de la méthode `startJuggling()`. On rappelle qu'à ce moment ci la beanbox connaît les deux objets à relier ainsi que leurs méthodes.

La beanbox crée alors un classe `__Hookup_`..., la compile, crée un objet `hookup` de cette classe et ajoute une ligne de code indiquant que le Bean Source a pour écouteur cet objet `hookup`. Elle lance la méthode `setTarget()` sur cet objet en lui passant l'objet Bean but et tout est alors relié et construit.

Remarque finale :

Ce qui est remarquable en Java c'est que tout ce code est construit de manière dynamique par le mécanisme d'introspection qui permet de construire dynamiquement un objet d'une classe sans connaître le nom de cette classe à la compilation, mais seulement à l'exécution, puis de connaître les méthodes d'un objet en le lui demandant au moment de l'exécution (quel type de listener as tu ?).