JavaBeans

JavaBeans @ SUN
http://java.sun.com/products/javabeans

JavaBeans : les composants réutilisables avec Java
http://cui.unige.ch/java/cours/notes/beans.pdf

Programmation avancée en Java
http://w3.ift.ulaval.ca/~abali/ift-21133/21133-Supplement.pdf

Des objets aux composants
http://www.essi.fr/~riveill/cours/car/07-99-slides-composants.pdf

Outils

JavaBeans Development Kit (BDK)
http://java.sun.com/beans/software/bdk_download.html

Forte for Java
http://www.sun.com/forte/ffj/index.html

Programmation Orientée Objet

- les programmes sont organisés comme des ensembles d'objets coopérant ensemble.
- chaque objet représente une instance d'une classe.
- les classes appartiennent à une hiérarchie suivant la relation d'héritage.
- La connaissance des objets qu'on manipule est indispensable avant l'utilisation de ces objets.

- Structure d'une application objet ⇒ flots de messages entre un certain nombre d'objets, les objets sont « presque » indépendants les uns des autres.

Cette indépendance (l'une des grandes forces de l'approche orientée objet) permet la réutilisation des objets par de nombreuses applications.

- Les objets sont plus stables que les spécifications qui définissent leurs interactions ⇒ les applications sont plus simples à écrire et à faire évoluer.

Évolution de la réutilisation

Motivation

intégration de modules logiciels existants
construction d'applications réparties par assemblage de modules logiciels existants

1. Copie de code source

2. Bibliothèques de fonctions

3. Bibliothèques de classes (modules)

4. Composants logiciels

Approche

description de l'architecture de l'application à l'aide d'un langage déclaratif

modèle de construction des composants

composants : interfaces, attributs, implémentation

description des interactions entre composants (connecteurs)

description de variables d'environnement (placement, regroupement, sécurité, etc.)

Les composants...

Intérêt : être des briques de base configurables pour permettre la construction d'une application par composition

Qu'est-ce que c'est ?

C'est un module logiciel qui a les propriétés suivantes :

Indépendant de la plate-forme (internet qui est un environnement réseau hétérogène),

Indépendant du langage (les technologies corba ou DCOM qui ajoutent une couche logicielle supplémentaire entre le langage et le moteur),

Basé sur un modèle standard (permet l'interaction entre composants s'ils répondent à un même standard),

Encapsulé.

Ces composants devraient en plus :

- Permettre la lecture ou la modification de leurs propriétés.

- Avoir la faculté d'être scrutables pour révéler leur interface.

- Être sensibles à l'aspect réseau.

- Être sécuritaires pour assurer la confidentialité des transactions qu'ils gèrent.

- Offrir un support aux outils de développement d'applications.

- Être persistants, i.e. capables de sauvegarder leur état une fois à l'autre.

Architectures des composants logiciels

Plug-ins:
composants additionnels pour des applications (Photoshop, Word, etc.)

Applets Java:
parties actives dans des documents, pas de communication

LiveConnect (Netscape):
communication entre applets, javascripts, plug-ins

OpenDoc/SOM:
communications entre parties hétérogènes de documents

ActiveX:
communications entre "contrôles" à travers DCOM

Java Beans:
composants actifs dans l'outil de développement et dans l'application finale

AppleScript:
langage d'interconnexion de composants génériques, publication des capacités de chaque composant (terminologie), terminologies standardisées : BD, tableur, texte, comptabilité, etc.

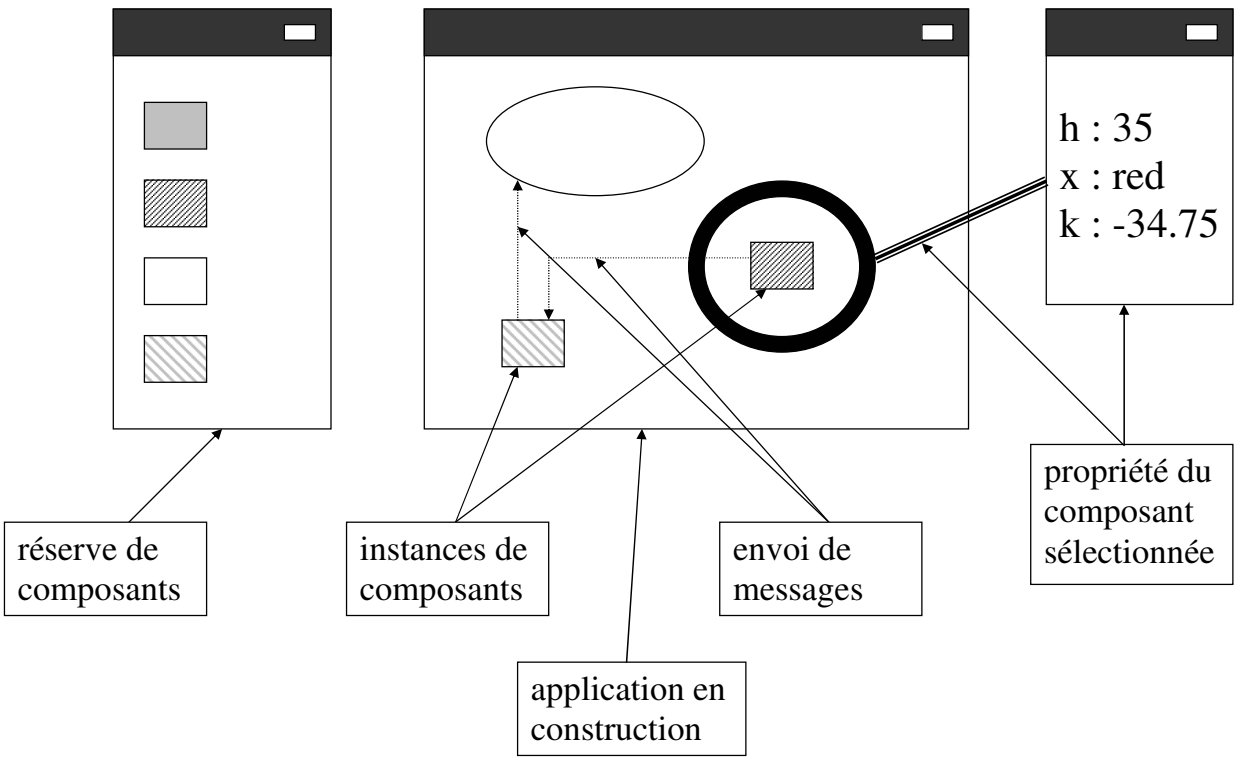
JavaBeans (ou Beans mais … pas un haricot!)

Selon la spécification des Javabeans, une *Bean* est un composant logiciel réutilisable pouvant être manipulé visuellement dans un outil de construction (*builder tool*).

En fait, elles représentent le modèle d'architecture à base de composants logiciels pour le langage Java.

Assemblage dynamique de composants

Visualisation en ligne de l'application réalisée : intéressant pour la construction d'une interface



Un composant possède

- des propriétés persistantes

Une *propriété* est une partie de l'état interne de la *Bean* qui peut être positionnée et interrogée par programmation, habituellement par une paire standard de méthodes d'accès get et set.

La **persistance**, propriété par laquelle on peut enregistrer l'état courant d'un objet et le récupérer par la suite. C'est un cliché de l'objet à un moment donné qui peut être récupéré plus tard de façon à recréer l'objet et à l'exécuter à partir de sa valeur au moment du cliché. La persistance s'appelle aussi **sérialisation**, parce qu'elle convertit un objet en un flux (*stream*) série.

- des événements reconnus

une *Bean* peut générer des **événements** de la même façon qu'un composant AWT, par exemple, un bouton qui génère des événements ActionEvent. Une *Bean* définit un événement si elle fournit des méthodes pour ajouter et enlever des objets écouteurs d'événements d'une liste d'écouteurs intéressés par cet événement.

- des méthodes de traitement des événements

les **méthodes** exportées par une *Bean* sont simplement toute méthode publique définie par la *Bean*, excluant celles qui sont utilisées pour positionner ou interroger les valeurs des propriétés et enregistrer ou enlever des écouteurs d'événements.

L'introspection

Un outil de développement (OD) peut interroger un composant (quelles sont tes méthodes, tes variables, les évènements que tu traites, etc.) => pas de fichier de configuration.

Grâce à l'usage de modèles bien définis l'OD peut reconnaître les propriétés, évènements et méthodes du composant => utilisation dynamique du composant (le composant est "vivant") pendant le développement

L'outil peut créer des classes de connexion entre composants (associer des méthodes aux évènements)

On associe habituellement la JavaBean à un objet **BeanInfo** qui définit la *Bean*, ses propriétés, ses événements, ses méthodes, ses éditeurs de propriétés et ses interfaces de personnalisation. La classe **BeanInfo** fournit cette information sous forme d'un certain nombre d'objets **FeadureDescriptor**, chacun dévrivant un seul aspect d'une Bean.

FeatureDescriptor possède plusieurs sous-classes :
BeanDescriptor, **Property-Descriptor**,
IndexedPropertyDescriptor, **EventSetDescriptor**,
MethodDescriptor, et **ParameterDescriptor**.

BDK

Pour travailler avec des JavaBeans, vous pouvez utiliser le *bdk* (Bean Development Kit) de Sun pour développer et agencer vos JavaBeans. Il est disponible à l'adresse :

http://java.sun.com/beans/software/bdk_download.html

Vous pouvez l'utiliser pour créer une interface (par exemple ajouter des boutons) et gérer les événements pour accéder aux méthodes des différentes JavaBeans.

Exécuter BDK :

%PATH_BDK%\bdk1_1\beans\beanbox\run.bat

Attention sur l'emplacement: Il y a un bug sur la résolution des chemins (pas d'espace! Donc pas de /My Documents/ ni /Program Files/ etc.).

Vous allez obtenir 4 fenêtres :

BeanBox, ToolBox, Properties, Method Tracer :

BeanBox

Permet à l'utilisateur de personnaliser une Bean en positionnant les valeurs de ses propriétés.

Une BeanBox définit donc des éditeurs de propriétés pour les types courants de propriétés tels que des nombres, les chaînes, les fontes et les couleurs.

Properties

Si, toutefois, une *Bean* a des propriétés de type plus complexe, elle pourra définir une classe **PropertyEditor** qui permettra à la *BeanBox* de laisser l'utilisateur changer les valeurs de telles propriétés.

ToolBox

Contient 16 JavaBeans de démonstration.

Ils peuvent servir aussi à interagir avec n'importe quel Bean que vous testez.

Method Tracer

affichage des messages de débogage simple.

Fenêtre BeanBox :

un objet de la classe java.awt.Panel

la fenêtre properties affiche les propriétés du Bean actuellement sélectionné.

Ces propriétés sont :

background (arrière plan)
forground (avant plan)

name (le nom)

font (la police)

mettre background yellow, puis done.

Plaçons un javabean :

ToolBox, ExplicitButton

au lieu de press, démarrer l'animation.

ToolBox, ExplicitButton

au lieu de press, arrêter l'animation.

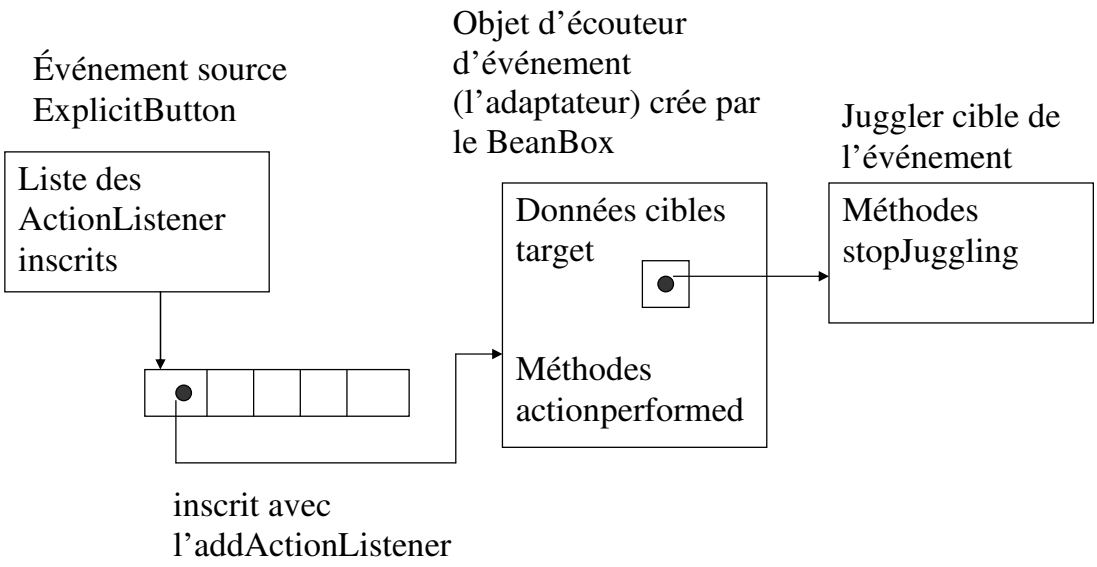
Sélectionnez le Juggler de ToolBox,

modifier l'animation rate (10 et 200) pour voir les différences,

ajouter les événements au Juggler.

BeanBox, Edit, Events, button push, action performed, stop

BeanBox, Edit, Events, button push, action performed, start



Modèle d'évènements

Modèle de délégation
Classe source d'évènements invoque des méthodes des classes destinations

Un objet Evenement est transmis

Les objets destinations doivent s'inscrire auprès de l'objet source

Adaptateurs

Intermédiaire entre un ou des objets écouteur qui n'implémentent pas l'interface *MyEventListener* et une source d'évènements du type *MyEvent*.

save puis load

makeapplet

Au plan de **l'emballage**…

une JavaBean est un programme Java pouvant être composé de plusieurs fichiers .class encapsulés sous forme de fichier .jar (Java ARchive). Ce format de fichier est basé sur le format *zip*.

Pour créer un fichier .jar :

```
jar cvf nom.jar liste_de_fichiers
```


Exemple :

```
jar cvf monFichier.jar *.class *.gif
```

Pour afficher le contenu d'un fichier JAR :

```
jar tvf file.jar
```

Pour décompacter un fichier .jar et récupérer tous les fichiers qu'il contient :

```
jar xvf file.jar
```

Pour extraire un fichier donné d'un fichier JAR :

jar xvf file.jar nomDuFichier

myApplet.html

```
<applet

    archive="./myApplet.jar,./support.jar
        ,./buttons.jar
        ,./juggler.jar
    "
    code="MyApplet" ← nom de la classe qui amorce
                      l'exécution de l'applet.
    width=395

    height=275

>
```


Construire un Bean

On peut distinguer quatre types possibles de JavaBeans :

Composant
Conteneur
Invisible
Applet

Une JavaBean est un composant si elle hérite de la classe java.awt.Component.

Une JavaBean est invisible lorsqu'elle n'hérite d'aucune classe dérivée de la classe Component. Elle peut toutefois générer des composants basés sur la classe Frame tels que des fenêtres et des

dialogues. Une JavaBean invisible peut être n'importe quoi, par exemple, une connexion à une base de données, un filtre ou un algorithme.

Une JavaBean Applet peut être traitée comme un conteneur ou comme un composant, puisque la classe Applet dérive de la classe Container et qu'un *Container* est un composant.

Classe et JavaBean

Pour qu'une classe puisse constituer une JavaBean, elle doit satisfaire à quatre conditions :

• Une JavaBean doit pouvoir être instanciée (ne pas être une classe abstraite ou une interface).

• Le constructeur d'une JavaBean ne doit pas avoir de paramètres.

• Une JavaBean doit implémenter la classe Serializable ou io.Externalizable.

• Les JavaBeans doivent obéir aux règles de signature de méthode, pour que la classe Introspector puisse correctement catégoriser les méthodes. Les signatures de méthodes sont des règles de nomenclature de méthodes spécifiant les types retournés, les paramètres et les exceptions. Si vous n'adhérez pas à ce standard, la classe Introspector ne peut pas comprendre les méthodes de votre JavaBean.

Les champs de données doivent être privés (encapsulation)

- les champs ne doivent être accessibles que par un get ou un set.
- Les champs sont mode de lecture pour l'OD si c'est un get, ou en mode d'écriture si c'est un set.

Exemples

SimpleBean.java

```
// un SimpleBean pour afficher un Hello World …

package simple1;

import java.awt.*;
import java.io.Serializable;

public class SimpleBean extends Canvas
                        implements Serializable{

    private Color color;
    private String msg = "Hello, World";
```

```
    //Constructor sets inherited properties
    public SimpleBean(){
        setSize(100,40);
        setBackground(Color.red);
    }

    public Color getColor() {
        return color;
    }

    public void setColor(Color c) {
        color = c;
        repaint();
    }

    public void paint(Graphics g) {
        g.setColor(color);
        g.drawString(msg, 20, 20);
    }
}
```


Compilation

javac –d . SimpleBean.java

-d . le point représente le répertoire où doit être placé le paquetage.

créer d'un manifeste

Soit le fichier manifest1.txt, permettant de décrire le contenu du fichier .jar

il doit contenir ce qui suit :

Main-Class: simple1.SimpleBean <= la classe SimpleBean contient le main.

Name: simple1/SimpleBean.class <= le nom du fichier qui contient la classe du bean.

Java-Bean: True <= la classe nommée à la ligne 3 est en réalité un JavaBean.

manifest1.txt

```
Manifest-Version: 1.0

Name: simple1/SimpleBean.class
Java-Bean: True
```


Création du fichier jar

jar cfm SimpleBean.jar manifest1.txt simple1/*.*

Chargement du fichier SimpleBean.jar dans bdk

File
LoadJar … puis charger le fichier SimpleBean.jar

Utilisation dans le bdk

Choisir SimpleBean du ToolBox et l'insérer dans le BeanBox.

Constatation

Le message ne peut pas être changé par l'utilisateur durant l'exécution. La variable msg n'apparaît pas sur le panneau des propriétés.

Une solution …… SimpleBean2.java

```
// SimpleBean2
package simple2;

import java.awt.*;
import java.io.Serializable;

public class SimpleBean2 extends Canvas
                         implements Serializable{

    private Color color;
    private String msg = "Hello, World";
```



```
    //Constructor sets inherited properties
    public SimpleBean2(){
        setSize(100,40);
        setBackground(Color.red);
    }

    public Color getColor() {
        return color;
    }
    public void setColor(Color c) {
        color = c;
        repaint();
    }

    // You need both a setter and a getter to have
    // the property appear in the property sheet
    // in the Beanbox

    public void setMsg(String msg) {
        this.msg = msg;
    }
```

```
    public String getMsg() {
        return msg;
    }

    public void paint(Graphics g) {
        g.setColor(color);
        g.drawString(msg, 20, 20);
    }
}
```

Créer puis charger SimpleBean2.jar dans le BeanBox de la même manière que pour l'exemple SimpleBean.jar

Java Bean Events

Répondre à un événement

Méthodes publics

Un bean répond aux événements en utilisant les méthodes publics

MAIS ne tient pas compte des « setter » et « getter ».

La classe SimpleBean (SimpleBean.java ou SimpleBean2.java) n'a pas de méthodes publics (autre que setter et getter) donc ne peut répondre à un événement généré par un autre bean.

La méthode la plus simple consiste à définir quelques méthodes publics sans aucun argument.

Par exemple,

```
public void dumb() {
    setMsg("dumb");
    repaint();
}
```

Faire le test SimpleBean3.java

Créer et charger SimpleBean3.jar de la même manière que SimpleBean.jar, puis faites ce qui suit :

- sélectionnez dans le ToolBox SimpleBean et insérez le dans le BeanBox.

- sélectionnez dans le ToolBox SimpleBean3 et insérez le dans le BeanBox.

- sélectionnez l'objet SimpleBean dans le BeanBox, puis appuyez sur Edit/Events/Mouse/MouseClicked ; une flèche rouge apparaît. Dirigez la vers SimpleBean3 puis cliquez dessus.

- La fenêtre EventTargetDialog va apparaître, sélectionnez la méthode dumb, puis cliquez sur ok.

- bdk va compiler et générer l'adaptateur associé.

- vérifiez dans la fenêtre dos intitulée javac, s'il n'y a pas un message d'erreur.

- si pas de message d'erreur, maintenant cliquez sur SimpleBean, vous allez remarquer que le message (msg) associé à SimpleBean3 va changer.

- refaire la même opération, en changeant le contenu du msg associé à SimpleBean3 puis cliquez encore sur SimpleBean, le message associé à SimpleBean3 va encore être modifié.

Créer un événement pour un bean

un compteur bean.

un bean qui compte de 0, jusqu'à un nombre prédéfini. À ce niveau il compte de nouveau de 0 ou bien il génère un événement.

il peut recevoir une valeur de départ d'un autre bean, et l'événement qu'il génère peut être reçu par un autre bean.

charger counter.jar dans dbk.
tictoc, juggler et counter beans dans beansbox
maxValue = 20

Connecter maxValue/maxValueReached du counter à la méthode stopjuggling de juggler bean

Connecter PropertyChange event de tictoc bean à la méthode increment() de counter bean.

Mettre tictoc de 1 à 5 secondes.

À ce niveau, vous allez remarquer que le compteur compte de 0 à 20, puis recommence à nouveau de 0. Et le jongleur n'est pas affecté par le passage par zéro.

Essayez maintenant de mettre :

rollover à false.

Dès que le chiffre 20 est atteint, arrêt total de l'activité du jongleur.

Examinez les programmes se trouvant dans counter.jar

Applet & Bean

Faire ce qui suit …

- Dans ToolBox, sélectionnez juggler et deux fois ExplicitButton.
- Renommez les ExplicitButton, arrêter et démarrer.

Pour chacun des boutons,

- Edit/Events/button push/actionPerformed
- dirigez la flèche rouge sur le jongleur puis, sélectionnez :
  Pour arrêter: stopjuggling
  Pour démarrer: startjuggling

Puis dans BeanBox sélectionnez edit /make applet, il préférable de « sérialiser » avec « seralize component » avant de créer l'applet.

Vous devez choisir un endroit où enregistrer l'applet, et le tour est joué. Appletviewer ou IE ou Netscape pour la visualiser.