eig–hes

POO

Java

Swing

MOTS CLES

JFrame

JPanel

Light & heavyweight

plaf

Annexe(s)

`IHMelem.java`

eig–hes

Swing

- 1 – Qu'est-ce ?
- 2 – Mise en page
- 3 – Les composants
- 4 – Modèle du contrôleur
- 5 – GUI contraints
- 6 – Exemple

Swing est le paquetage de composants (javax.swing.*) qui permet de développer des IHM (GUI= graphic user interface) évolués.

Historiquement, il ne date pas de la première version. Le x du suffixe de son nom, signifie qu'il s'agit d'une extension et est aussi une obligation car , pour des raisons de sécurité, Java ne supportait pas l'ajout de paquetage avec le préfixe « java » !

Nous verrons plus avant que pour une JVM standard, ce paquetage devrait être utilisé de manière systématique et être préféré à awt partout là ou il a ses équivalents.

Note: dès 2005 des paquetages plus « génériques », tels SWT, remportent un certains succès.

eig–hes

1 – Qu'est-ce ?

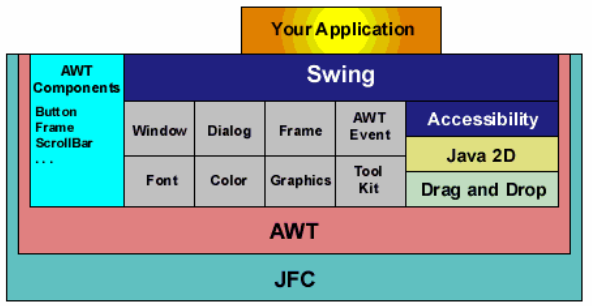
- Bibliothèque de composants graphiques
  - Dès 1.2
  - Au-dessus de awt
  - Contient:
    - Graphisme
    - Couleur
    - Fontes
    - Composants « réactifs »

Pour les deux premières éditions majeures de Java (1.0 et 1.2), la seule façon de faire du graphisme était d'utiliser java.awt.* . Quand il a fallu reconsidérer les fondements de l'implémentation, il n'a pas été possible de faire abstraction de l'ancienne bibliothèque. Ainsi, pour des raisons essentielles de compatibilité Swing s'appuie sur awt. Le paquetage javax.swing.* est en partie similaire à awt. Respectivement, nous trouvons JPanel et Panel, JButton et Button, etc. Mais il ne faut pas croire qu'il suffit toujours de remplacer un composant awt pas son correspondant préfixé par J appartenant à Swing.

Swing présente l'avantage de fournir des composants légers (*lightweight GUI component*) qui sont indépendants de gestionnaire de fenêtre de la machine hôte. Un composant léger émule le composant de gestion d'événements. Au contraire des composants lourds (*heavyweight GUI component*) de awt qui ont leur pair (peer) du système d'exploitation de la machine hôte. Initialement ce choix avait été fait pour assurer la portabilité de Java. C'est JNI (Java Native Interface) qui assurait la « communication » avec le système d'exploitation au sein même de la machine virtuelle (sous UNIX, un java.awt.Button était ainsi un vrai bouton Motif).

Notons que les composants Swing JApplet, JDialog, JFrame et JWindow sont les exceptions de composants lourds.

1 – Qu'est-ce ?



Un des atouts de Swing réside dans son offre de choix de l'aspect des composants puisque ceux-ci sont émulés. C'est ce que l'on appelle le *pluggable look and feel* avec le … joli acronyme de *plaf*. Sun a choisi son propre *look and feel* : Metal; qui donne une personnalité Sun aux GUI.

Nous voyons sur la diapositive que Swing coiffe et étend awt (avec Java2D par exemple). Rappelons que le paquetage est préfixé avec javax. Pourquoi? Parce qu'une JVM 1.1 refuse de charger tout paquetage de préfixe java !

eig–hes

1 – Qu'est-ce ?

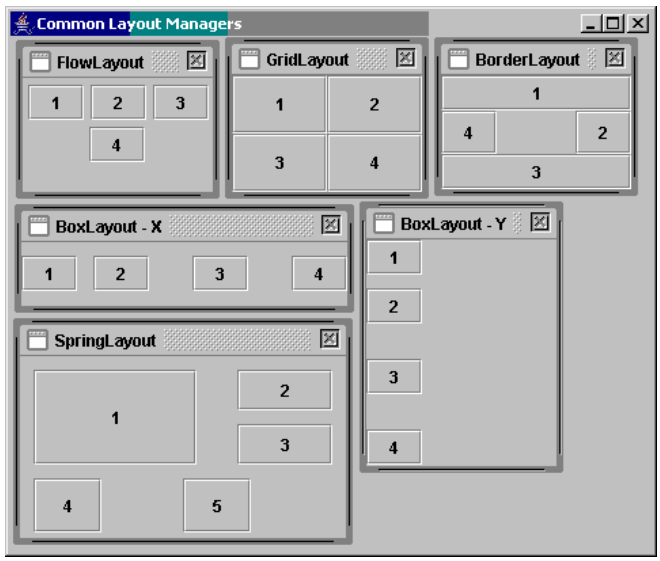
- Collection de gestionnaires de placements
- Propriétés relativement aux composants
  - Libres, Java s'en occupe
  - Contraintes, Java les applique
  - Tailles préférées, Java les respecte

Les gestionnaires de placement sont des éléments fondamentaux pour les IHM. Ce sont eux qui sont responsables de la mise en page des composants (des dérivés de Jcomponent) qui sont accessibles à l'utilisateur. De plus, une pratique courante de commodité est celle du redimensionnement d'une fenêtre qui doit être pris en charge justement par le getionnaire de placement lors d'une telle opération.

La mise en page ou le Layout (terme anglais consacré) est un choix primordial de la présentation qui joue un rôle important pour la visualisation des communications avec un applicatif.

eig–hes

2 – Mise en page



Le *layout* est la mise en page. Les pages d'un journal sont passées au *layout*; quelqu'un peut avoir mis les 4 articles en lignes, en colonnes, sur 2 lignes de 2 colonnes chacune, ou encore autrement. Java propose des `Layouts` tout prêts. Ils sont bien utiles pour des interfaces simples mais ils montrent très vite leur limite si nous voulons imposer nos contraintes quand nous redimenssionons les fenêtres. Les *layouts* reçoivent des composants. C'est le `Container` qui reçoit les composants et c'est donc à lui que l'on « impose » son layout.

En trop bref:

-FlowLayout: les composants sont ajoutés depuis l'est et poussés vers l'ouest (notion géographique cardinale)

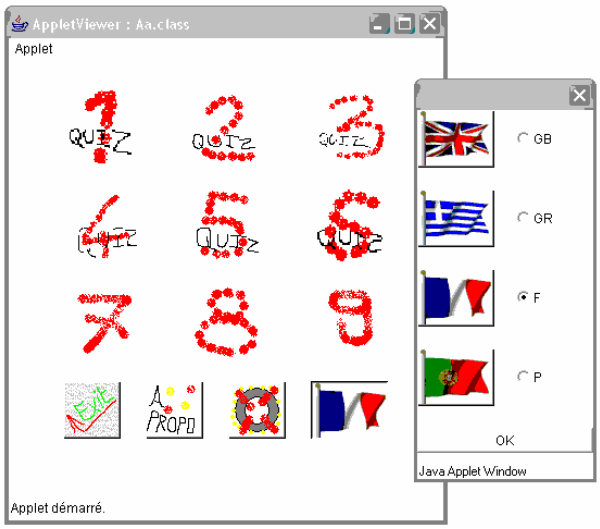
-GridLayout: les composants sont ajoutés en occupant les cases d'une matrice déclarée à la construction de l'objet

-BorderLayout: les quatre points cardinaux entourent une position majeure nommée CENTER (il y a donc 5 possibilités)

-BoxLayout: les composants sont mis en boîte (selon X ou Y)

-SpringLayout: les composants sont contraints par quatre valeurs (x, y, largeur, hauteur)

2 – Mise en page



Le CardLayout: une pile de cartes dont une seule est visible à la fois. Les méthodes de navigation:

-next()

-previous()

-first()

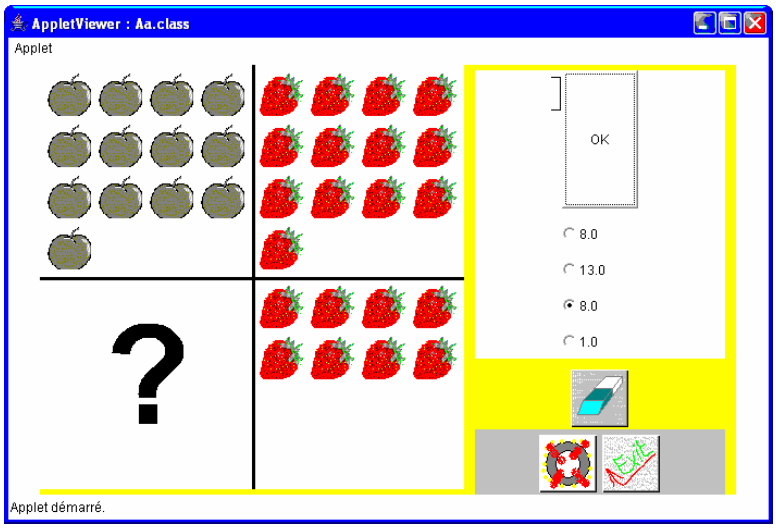
-last()

Note:

Très souvent la navigation d'une carte à l'autre doit être étendue au-delà des simples possibilités énoncées ci-dessus.

Nous verrons plus loin que des composants tels que le `JTabbedPane` est peut-être mieux adapté pour ce genre de navigation (par essence, il indique les cartes qu'il est possible d'atteindre).

2 – Mise en page



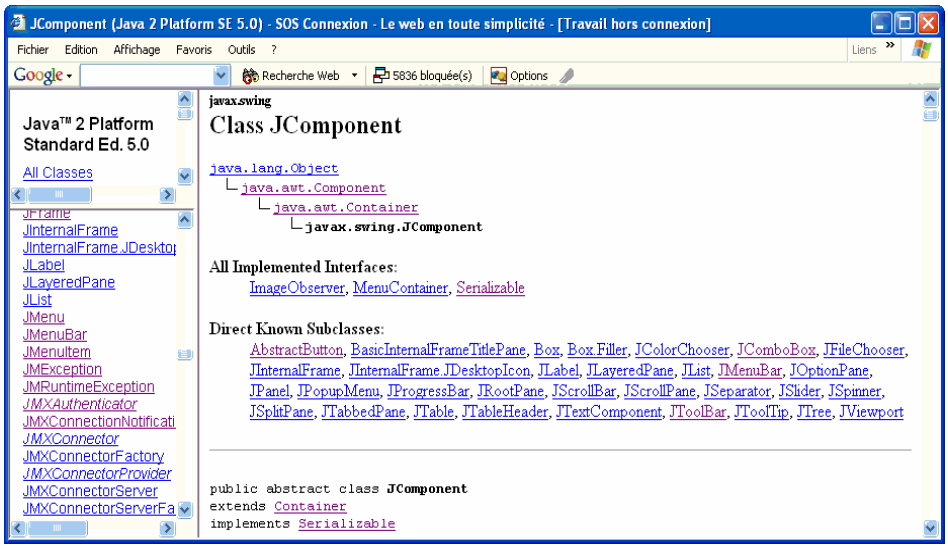
Cette diapositive montre une des cartes accessibles sur l'image précédente.

La navigation (`next()`, `previous()`, …) est ici « masquée » avec un bouton « Exit » qui vous renverra à la carte maître (souvent trop contraignant!).

Note:

Le `CardLayout` possède un concurrent majeur: les onglets qui sont aujourd'hui très répandus dans les GUI. Leur mise en œuvre se fait avec les objets `JTabbedPane` qui sont très particularisables en fonction des besoins.

3 – Composants

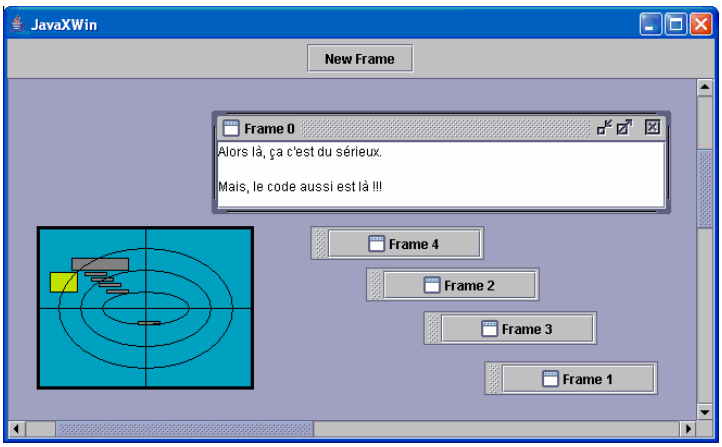


Comme vous pouvez le constater, vous avez le choix !

De nombreux composants permettent à eux seuls de constituer des éléments d'une interface homme-machine.

Mais, il ne faut pas croire que cette offre est restrictive; Java possède et offre tous les mécanismes de développement de base pour faire des interfaces plus évoluées. Mais, qu'est-ce à dire ?

3 – Composants



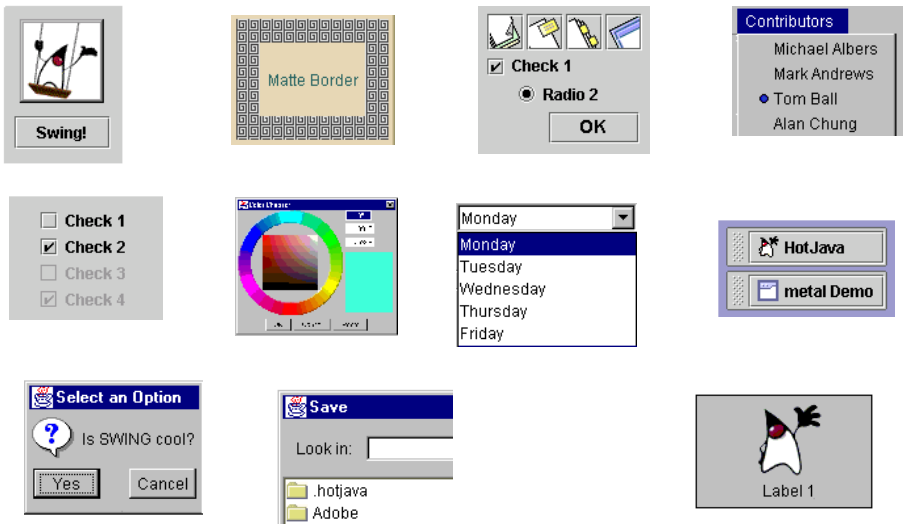
Cette interface tirée de [Swing, M. Robinson & P. Vorobiev, Manning] propose de visualiser le contenu entier d'une fenêtre « trop grande pour l'écran ».

Sur la gauche de la capture vous voyez une incrustation de l'espace de dessin. Cette dernière est toujours en avant-plan et affiche sous forme de pictogramme des *frames* internes que l'on peut ajouter à volonté et positionner dans un plan à sa propre convenance. Cette interface présente un avantage certain pour retrouver des items propres à un développement (nœuds de réseau, emplacement géographique, etc.)

Attention, ce genre d'interface à un coût: environ 1000 lignes de code ici (et … sans action de GUI)

e i g – h e s

3 – Composants

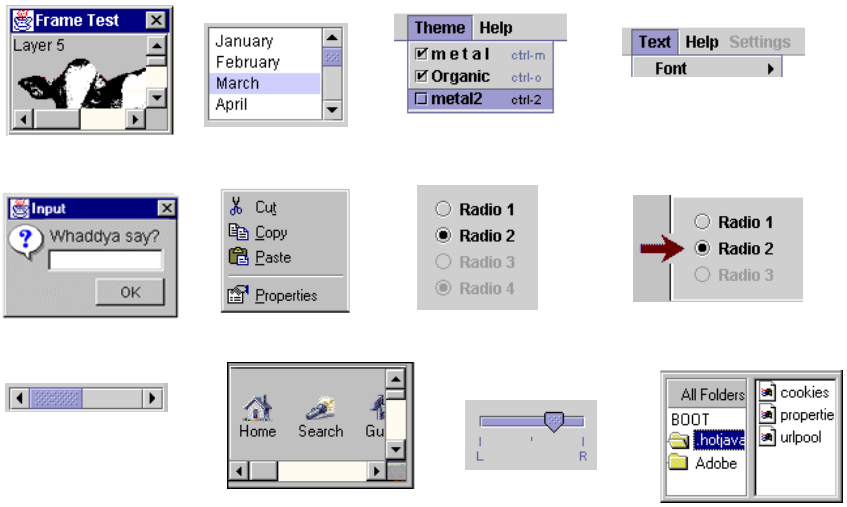


La large palette des composants permet de réaliser des interfaces évoluées, elle est richement dotée en divers éléments qui permettent une interaction facile avec les fonctions que l'on désire mettre en place.

Sur le diapositive, vous voyez: un applet, une bordure, différents boutons, un menu à cocher, un groupe de boutons à cocher, une interface pour choisir les couleurs, ce qu'il est combo box, des icônes pour le bureau, un dialogue de choix d'options et pour finir un label pouvant également contenir un pictogramme qui lui est associé.

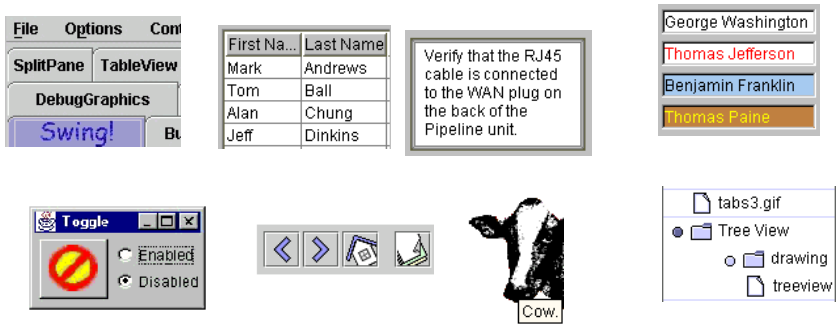
Comprenez bien que sont des objets de Java et qu'il suffit de les instancier pour avoir, en retour, une valeur choisie (par exemple, un triplet RGB).

3 – Composants



Les composants contenus sur cette diapositive, nous pouvons les décrire selon : un *layered frame* qui possède deux ascenseurs afin de se déplacer dans une image, une liste d'items parmi lesquels une ou plusieurs sélections peuvent être faites, un menu avec des boîtes à cocher, une barre de menu avec ses menus déroulants dans lesquels nous nous déplaçons telle une arborescence, un dialogue de saisie, un menu *pop-up*, un groupe de radios bouton, une autre forme pour regrouper des radios boutons avec indication, un ascenseur (une scroll bar),un panneau à défilement (un scroll pane), une jauge de choix (un *slider*) et un panneau divisé (un *split pane*).

3 – Composants



Pour finir cette numération de composants: vous pouvez voir tout d'abord un tableau à onglets (un *tabbed pane*) , un affichage de table(s), une aire de texte (un `TextArea`), des champs de texte avec différentes couleurs, un composant de bascule, une barre d'outils (une *tool bar*) , un texte signalétique (un *tool tip*)et enfin une vue arborescente (un `JTree` view).

Certains de ses composants peuvent être encore agrémentés de ce qui s'appelle des décorateurs. Nous arrivons ainsi à faire des interfaces évoluées et nous ne devons pas oublier un des principes de base qui consiste à conserver une lisibilité aisée.

Note: il existe encore un composant appelé barre de progression (*progress bar*) qui devient de plus en plus utile puisque le téléchargement fait souvent partie intégrante d'une session de travail.

eig–hes

3 – Composants : exemple JSpinner

- **javax.swing**
  class JSpinner

java.lang.Object
 java.awt.Component
  java.awt.Container
   javax.swing.Jcomponent
    javax.swing.JSpinner

- All Implemented Interfaces:
  - ImageObserver, MenuContainer, Serializable

Comment choisir dans une liste énumérée ? (sans loi ordinale). Le composant JSpinner est un de ceux qu'il ne faut pas oublier de rechercher quand un besoin particulier de ce type se fait sentir. Il est formé d'une seule ligne de saisie qui permet à l'utilisateur de sélectionner un nombre ou tout autre objet ayant une valeur à partir d'une séquence ordonnée.

Il est typiquement associé à une paire de boutons qui nous permettent de naviguer pas à pas entre les éléments qui sont mis en séquence; les touches du clavier flèche vers le haut et flèche vers le bas sont fonctionnelles. Un utilisateur peut également saisir une valeur légale directement dans le champ du JSpinner.

Le JSpinner est très similaire au JComboBox mais il lui est parfois préféré parce qu'il n'y a pas de liste qui se déroule vers le bas est ainsi ne cache pas d'autres éléments de l'interface. L'autre aspect des plus significatifs pour le JSpinner est, à l'instar d'autres composants d'ailleurs, que celui-ci possède des modèles. Il existe même pour affecter le *spinner* de propriétés une interface abstraite SpinnerModel dont une instance pourra être mise en relation avec l'objet graphique.

3 - JSpinnerDemo2

```
import java.awt.*;
import java.util.*;
import javax.swing.*;
import javax.swing.border.*;

class SpinnerDemo2 extends JFrame {

 public SpinnerDemo2() {
  super("Demo JSpinner + SpinnerDateModel");

  JPanel p = new JPanel();
   p.setLayout(new BoxLayout(p, BoxLayout.X_AXIS));
   p.setBorder(new EmptyBorder(10, 10, 10, 10));
   p.add(new JLabel("Sélectionnez la date: "));
```


Voici un exemple complet de mise en œuvre d'un de ces composants de Swing qui permet de lui attacher un modèle particulier. L'objet `JSpinner` peut recevoir différents types de modèle:

`java.lang.Object`

`javax.swing.AbstractSpinnerModel`

`javax.swing.SpinnerListModel`

`javax.swing.SpinnerNumberModel`

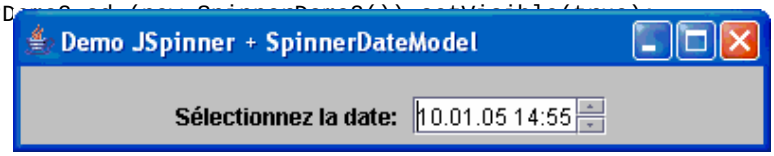
`javax.swing.SpinnerDateModel`

Bien sûr, si ces modèles ne vous conviennent pas, rien ne vous interdit de développer le votre tout en respectant les contraintes imposées par l'interface `SpinnerModel`.

Remarquez également que sur cette première partie du code nous avons mis en œuvre la classe `EmptyBorder` afin de mettre des marges au panneau qui sera ajouté au conteneur du `JFrame`. Ce qui permet de mettre « de l'espace » autour d'un composant.

3 - JSpinnerDemo2 (suite)

```
SpinnerModel modele = new SpinnerDateModel(
       new Date(),            // valeur initiale
       null,                  // valeur Minimum
       null,                  // valeur Maximum
       Calendar.DAY_OF_MONTH // pas du spin
   );
   JSpinner spn = new JSpinner(modele);
   p.add(spn);
   getContentPane().add(p, BorderLayout.SOUTH);
}
 public static void main( String args[] ) {
  SpinnerD
 }
}
```



Le modèle que nous voulons implémenter ici est un dérouleur de dates. Nous pouvons choisir sa valeur initiale, sa valeur minimale et sa valeur maximale ainsi que le pas du déroulement ici il s'agit d'un jour.

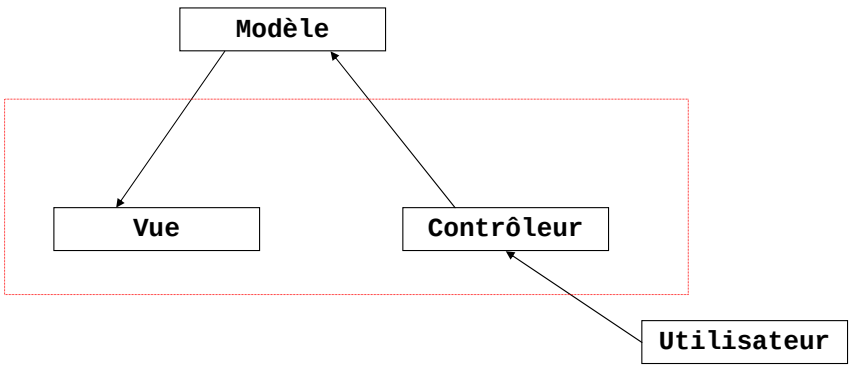
Bizarre: une entrée manuelle d'un numéro de mois plus grand que douze vous fera passer à l'année suivante conformément au modulo 12 !

Pour terminer, ce type de composant peut rendre de grands services en particulier dans le cas de listes discontinues (non ordinales) ou simplement discrètes; alors le déroulement passe d'un événement à l'autre selon l'ordre dans lequel la liste a été construite.

eig–hes

4 – Modèle du contrôleur

- Architecture MVC
  - Model-View-Controller



Les composants de swing sont réalisés selon une architecture dite « modèle, vue et contrôleur »; souvent, on utilise l'acronyme MVC. C'est une architecture MVC qui permet notamment de gérer les différents *look-and-feel* pour un même composant.

Un objet de l'interface graphique est distingué selon trois parties différentes:

- Le modèle qui représente le contenu de l'objet données et protocoles de manipulation

- La vue qui présente les informations contenues dans le modèle à l'utilisateur

- Le contrôleur qui permet à l'utilisateur de manipuler les données

Cette architecture est relativement peu couplée et, si l'on veut changer l'une de ses parties, cela pose moins de problèmes que si l'objet était un tout.

(le `JSpinner` fonctionne également comme cela avec les modèles).

eig–hes

4 – Modèle du contrôleur

- Avec Swing
  - Modèle: `setModel()` et `getModel()`
  - Vue: `setUI()` et `getUI()`

- UIObject
  - Modèle + contrôleur
  - Exemple: **`MetalButtonUI`**

En fait, Java regroupe le modèle et le contrôleur. C'est la classe `javax.swing.plaf.ComponentUI` qui va être spécialisée pour chacun des composants.

Par exemple, à un `JButton` est associé un ou plusieurs modèles; par exemple `DefaultButtonModel` pour modèle de bouton et toute une série de vue. Cette série de vue fut dérive elle-même d'une classe mère qui est `javax.swing.plaf.basic.BasicButtonUI` .

C'est le UI Manager qui est responsable du *look-and-feel*.

eig–hes

5 – GUI contraints

- **`GridBagConstraints`**
  - Pour
    - configurer la grille en position (en X et en Y)
    - configurer la grille en extension (en X et en Y)
    - pondérer (en X et en Y)
    - ancrer
    - remplir ou occuper
    - border le conteneur (**`Container`**)
    - Compléter ou *padding* (en X et en Y)

Les contraintes de placement se font rapidement sentir lorsque nous développons des interfaces. Voir « se balader » des composants lorsque la fenêtre est retaillée devient vite insupportable; fixer la taille peut parfois convenir mais cela ne doit devenir une systématique.

Nous allons voir que les contraintes ne se définissent pas simplement mais font intervenir plusieurs objets.

eig–hes

5 – GUI contraints

- **public GridBagConstraints(**
  int gridx, int gridy,
  int gridwidth, int gridheight,
  double weightx, double weighty,
  int anchor,
  int fill,
  Insets insets,
  int ipadx, int ipady)

La mise en page de composants graphiques et facilitée avec les classes `*Layout`. Il n'en reste pas moins que lors des redimensionnements de fenêtres il est fréquent que les composants se placent à des endroits qui ne conviennent pas à l'interface. Il y a deux moyens de remédier à cet inconvénient : soit fixer une fois pour toute la taille de la fenêtre soit développer l'interface avec des contraintes. En fait, toutes les contraintes de placement dans un plan fait de deux dimensions sont contenues dans une classe générale qui se nomme `GridBagConstraints`. Cet objet demande également l'instanciation de nombreux autres objets afin de contrôler le placement sur des positions x,y d'une part et d'ordonnancement sur un axe d'autre part. Nous allons le voir, il est aussi possible de définir correctement des marges et des bordures pour que les objets graphiques ne se chevauchent pas.

Un `Insets` est un bord de `Container` (un bord, un blanc ou un titre)

eig–hes

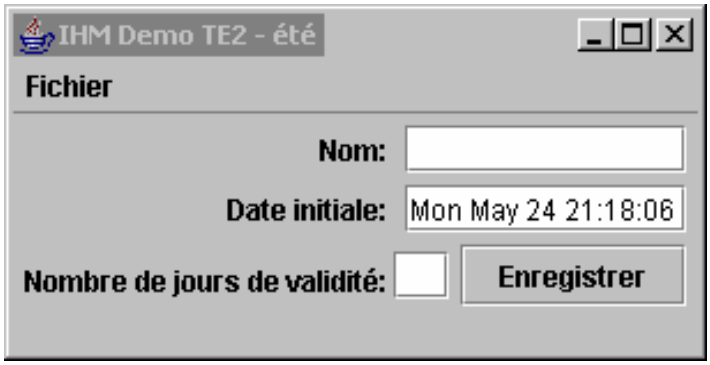
6 – Exemple

- Principe:
  - Configurer l'objet **`GridBagConstraints`**
  - Ajouter le premier composant
  - Reconfigurer l'objet **`GridBagConstraints`** ou modifier les contraintes
  - Ajouter le deuxième composant
  - etc

Pour appliquer ce principe, il est indispensable de faire un croquis de ce que vous voulez afficher.

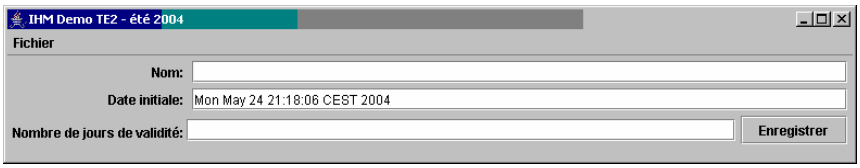
En effet, le énième composant aura des valeurs de grille quelques fois particulières (autre que ligne-colonne visible).

6 – Exemple



L'image ci-dessus donne idée de ce que nous voulons obtenir.

6 – Exemple



Sur cette image étirée par rapport à la précédente, les alignements sont conservés.

Nous obtenons ceci uniquement grâce aux contraintes et aux bordures.

6 – Exemple

- D'abord mettre un *layout* selon :

```
GridBagLayout layout = new    GridBagLayout();
```

- Puis mettre ce layout

```
lePanel.setLayout(layout);
```

- Puis définir les contraintes qui ne sont pas celles par défaut et les mettre avec:

```
layout.setConstraints(labelDeNom,
  fixe(GridBagConstraints.EAST,0,1,new Insets(5,
  5, 0, 0), 0));
```


Le placement des composants n'est pas déterministe si l'on ne prend pas soin d'y apporter des contraintes. C'est la classe GridBagConstraints qui permet d'initialiser ces contraintes.

Ici, la méthode fixe() affecte les contraintes (voir le code après).

eig_hes

6 – Exemple

- **private GridBagConstraints**
  - **fixe(int a, int f, int gw, Insets ist, double wx)**
    {
- **GridBagConstraints**
- **contraintes = new GridBagConstraints();**
- **contraintes.anchor = a;**
- **contraintes.fill = f;**
- **contraintes.gridwidth = gw;**
- **contraintes.insets = ist;**
- **contraintes.weightx = wx;**
- **return contraintes;**
- **}**

C’est pour chaque composant que nous allons fixer ses propres contraintes.