

Java

INTRODUCTION	2
NOUVEAUTES DE JAVA 1.1	2
LES NOUVEAUTES DES AWT	2
PEERS	3
APERÇU GENERAL.....	4
LES EVENEMENTS.....	6
LE MODELE ANCIEN	6
<i>Capter un évènement.....</i>	<i>7</i>
<i>Exemple.....</i>	<i>8</i>
LE NOUVEAU MODELE.....	10
<i>Listeners.....</i>	<i>11</i>
<i>Méthodes des listeners.....</i>	<i>11</i>
<i>Adaptateurs.....</i>	<i>13</i>
INNER CLASSES.....	15
CONTENEURS.....	17
PRINCIPALES CARACTERISTIQUES DES LAYOUT	18
<i>BorderLayout.....</i>	<i>18</i>
<i>FlowLayout.....</i>	<i>18</i>
<i>GridLayout.....</i>	<i>18</i>
<i>Note sur FlowLayout.....</i>	<i>19</i>
DESSIN.....	21
CHOIX.....	24
COMPORTEMENT COMMUN	24
BOUTON A COCHER.....	24

Introduction

Nouveautés de Java 1.1

- Internationalisation,
- Applettes signées,
- format de fichier JAR,
- amélioration de AWT (window toolkit),
- modèle de composants JavaBeans(tm),
- améliorations de gestion de réseau,
- package mathématique pour grands nombres,
- appel à distance: Remote Method Invocation (RMI),
- réflexion,
- connexion aux bases de données database connectivity (JDBC),
- sérialisation d'objets,
- classes internes.

Les nouveautés des AWT

- Nouveau modèle d'évènements
Le modèle de Java 1.02 est basé sur l'héritage, le nouveau est basé sur la délégation. Il est proche de X-Window.
- Composants légers
Un composant léger se rapproche d'un gadget, alors qu'un composant classique est un widget
- Presse-papier et transfert de données
- Menus jaillissants
- Fenêtres à ascenseur
Les ascenseurs sont intégrés dans certains conteneurs, comme les " ScrolledWindow " de Motif.

Les peers sont présents depuis toujours dans Java.

Les *peers* sont des classes AWT constitués de composants natifs, c'est-à-dire dépendant de la plateforme.

Un composant Java délègue une grande partie de leurs fonctionnalités à son *peer*.

Pour un menu, sous Unix, JDK crée un menu Motif, sous Windows, c'est un menu Windows, etc.

Cette approche a permis un développement rapide de Java. De plus, le look-and-feel d'un programme ou d'une applette correspond au système natif de la machine qui l'exécute.

Remplacé progressivement par des widgets Java purs.

Aperçu général

Les programmes sont dirigés par évènement s'ils font usage de classes awt (abstract windowing toolkit).

La classe de base des awt est la classe abstraite `Component`.

Les classes conteneur sont

- Container (abstraite, responsable du layout)
 - Window (interaction avec le système)
 - Frame (fenêtre principale d'application)
 - Dialog
 - Panel (pour contenir des composants)
 - Applet
 - ScrollPane (enrobe un conteneur d'ascenseurs)

Les classes d'interaction sont

- Canvas (dessin)
- Label
- Button
- TextComponent
 - TextField
 - TextArea
- Checkbox
- Choices (combobox)
- List
- Scrollbar

Les menus sont à part.

Un programme étend `Frame`, une applette étend `Applet`.

Une applette est exécutée à travers un fichier html, du type

```
<title>Essai</title>
<hr>
<applet code="PlusMoins.class", width=300, height=100>
</applet>
<hr>
```

```
import java.awt.*;

class PlusMoins extends Frame {
    Button oui, non;
    Label diff;

    public PlusMoins() {
        super("Oui et non");
        oui = new Button("Plus!");
        non = new Button("Moins!");
        diff = new Label("0", Label.CENTER);
        add(oui, "North");
        add(non, "South");
        add(diff, "Center");
        show();
    }

    public static void main(String[] argv) {
        new PlusMoins();
    }
}
```

Les évènements

Le modèle ancien

Chaque classe dérivant de `awt.Component` hérite d'une méthode

```
boolean handleEvent(Event e)
```

qui se comporte grosso-modo comme la procédure de fenêtres de Windows. Voici la méthode de base

```
//java.awt.Component.handleEvent()  
public boolean handleEvent() {  
    switch (e.id) {  
        case Event.MOUSE_ENTER :  
            return mouseEnter(e, e.x, e.y) ;  
        case Event.MOUSE_EXIT :  
            return mouseExit(e, e.x, e.y) ;  
        case Event.MOUSE_MOVE :  
            return mouseMove(e, e.x, e.y) ;  
        case Event.MOUSE_DOWN :  
            return mouseDown(e, e.x, e.y) ;  
        case Event.MOUSE_DRAG :  
            return mouseDrag(e, e.x, e.y) ;  
        case Event.MOUSE_UP :  
            return mouseUp(e, e.x, e.y) ;  
        case Event.KEY_PRESS :  
        case Event.KEY_ACTION :  
            return keyDown(e, e.key) ;  
        case Event.KEY_RELEASE :  
        case Event.KEY_ACTION_RELEASE :  
            return keyUp(e, e.key) ;  
        case Event.ACTION_EVENT :  
            return action(e, e.arg) ;  
        case Event.GOT_FOCUS :  
            return gotFocus(e, e.arg) ;  
        case Event.LOST_FOCUS :  
            return lostFocus(e, e.arg) ;  
    }  
    return false ;  
}
```

Il y a donc une méthode par évènement (`keyDown`, `mouseDrag`).

Capter un évènement

Il suffit de masquer *une* méthode pour capter un comportement spécifique.

Pour utiliser la méthode de la classe de base (surclasse dans la hiérarchie d'héritage), appeler:

```
super.handleEvent(e);
```

Pour passer l'évènement pour traitement par la surclasse conteneur (surclasse dans la hiérarchie d'instance):

```
return false;
```

Les évènements d'*action* (type `Event.ACTION_EVENT`) sont produits par les classes

Button	Checkbox	Choice
List	MenuItem	TextField

La méthode `action` a le prototype

```
public boolean action (Event e, Object w);
```

L'objet passé en argument dépend du composant qui a déclenché l'évènement:

Button	texte de l'étiquette (String)
Checkbox	état de la boîte (Boolean)
Choice	texte de la sélection (String)
List	id
MenuItem	id
TextField	id

L'évènement `e` a un champ `target` qui retourne la référence du composant. L'inconvénient du modèle est d'obliger d'écrire une classe dérivée dès la moindre modification de méthode de traitement d'un évènement.

Exemple

```
import java.awt.*;

class ActionTest extends Frame {
    Button    button = new Button("Cancel");
    Checkbox  checkbox = new Checkbox("Something to check about");
    TextField textfield = new TextField(25);
    Choice    choice  = new Choice();
    List      list     = new List();
    MenuItem  quitItem = new MenuItem("quit");

    static public void main(String args[]) {
        Frame frame = new ActionTest();
        frame.reshape(100,100,200,200);
        frame.show();
    }
    public ActionTest() {
        super("Action Test");

        MenuBar menubar = new MenuBar();
        Menu      fileMenu = new Menu("File");

        fileMenu.add("menu item");
        fileMenu.add(quitItem);
        menubar.add(fileMenu);
        setMenuBar(menubar);

        choice.addItem("One");   choice.addItem("Two");
        choice.addItem("Three"); choice.addItem("Four");

        list.addItem("item One"); list.addItem("item Two");
        list.addItem("item Three"); list.addItem("item Four");
        list.addItem("item Five"); list.addItem("item Six");

        setLayout(new GridLayout(0,1));
        add(button); add(checkbox);
        add(list); add(textfield); add(choice);
    }
    public boolean action(Event event, Object what) {
        if(event.target == quitItem) {
            System.exit(0);
        }
        System.out.print(event.target.getClass().getName());
        System.out.println(" " + what.getClass().getName() +
                           " = " + what);
        return true;
    }
}
```


Voici les actions effectuées :

coche, double clics, sélection, validation d'un texte, choix d'un item dans un menu.

```
java.awt.Checkbox java.lang.Boolean= true  
java.awt.List java.lang.String= item One  
java.awt.List java.lang.String= item Two  
java.awt.Choice java.lang.String= Three  
java.awt.TextField java.lang.String= machin  
java.awt.Button java.lang.String= Cancel  
java.awt.MenuItem java.lang.String= menu item
```

Le nouveau modèle est en de nombreux point plus proche de X-Window.

- Les évènements sont captés et traités par des *listeners*.
- Les listeners sont *enregistrés* sur les composants en appelant une méthode `AddXYZListener`. C'est donc proche du `XtAddCallback()` de Xt Intrinsics.
- Chaque classe de listeners a des *méthodes de traitement* d'évènements (des callbacks).
- Les classes de listeners sont des *interfaces*.
- Il suffit de définir ces méthodes, par *implémentation* d'un listener.

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

public class ActionExample extends Applet {
    public void init() {
        Button button = new Button("Activate Me");
        button.addActionListener(new ButtonListener());
        add(button);
    }
}

class ButtonListener implements ActionListener {
    public void actionPerformed(ActionEvent event) {
        System.out.println("Button Activated!");
    }
}
```

Le bouton est doté d'un listener d'actions.

Les activités à réaliser lors d'un évènement action sont décrite dans le `ActionListener` : cette classe n'a qu'une seule méthode propre qu'il s'agit d'écrire.

Listeners

Les *listeners* ont pour classe de base `java.util.EventListener`.

Les 11 listeners eux-mêmes sont des interfaces. Ce sont

```
ActionListener
AdjustmentListener
ComponentListener
FocusListener
ContainerListener
ItemListener
KeyListener
TextListener
MouseListener
MouseMotionListener
WindowListener
```

Ils sont définis dans `java.event`.

Chaque composant a ses méthodes d'enregistrement de listener,

Button	<code>void addActionListener(ActionListener)</code>
Checkbox	<code>void addItemListener(ItemListener)</code>
Choice	<code>id</code>
Component	<code>void addComponentListener(ComponentListener)</code> <code>void addFocusListener(FocusListener)</code> <code>void addKeyListener(KeyListener)</code> <code>void addMouseListener(MouseListener)</code> <code>void addMouseMotionListener(MouseMotionListener)</code>
Container	<code>void addContainerListener(ContainerListener)</code>
List	<code>void addItemListener(ItemListener)</code> <code>void addActionListener(ActionListener)</code>

On peut ajouter plusieurs *listener* à un composant.

Toute classe des awt dérivant de `Component` possède les listeners de cette classe. On peut par exemple ajouter un `MouseListener` à un bouton.

Méthodes des listeners

- Chaque listener a des méthodes.
- Les méthodes ont des noms convenus.
- Ces méthodes sont appelées pour le traitement des évènements associés.

Pour une action, l'ActionListener appelle sa méthode `actionPerformed()`. Voici d'autres exemples.

ActionListener	<code>void actionPerformed(ActionEvent)</code>
ComponentListener	<code>void componentResized(ComponentEvent)</code> <code>void componentMoved(ComponentEvent)</code> <code>void componentShown(ComponentEvent)</code> <code>void componentHidden(ComponentEvent)</code>
ContainerListener	<code>void componentAdded(ContainerEvent)</code> <code>void componentRemoved(ContainerEvent)</code>
ItemListener	<code>void itemStateChanged(ItemEvent)</code>
KeyListener	<code>void keyTyped(KeyEvent)</code> <code>void keyPressed(KeyEvent)</code> <code>void keyReleased(KeyEvent)</code>

Les évènements eux-mêmes sont des classes dérivées de `AWTEvent`. Le nouveau modèle fournit des classes là où l'ancien avait des constantes pour différencier les évènements.

Les évènements ont des méthodes et conservent encore des constantes.

Pour `MouseEvent` par exemple, méthodes

```
int getClickCount()
int getX()
int getY()
Point getPoint()
```

et constantes

```
MOUSE_FIRST, MOUSE_LAST, MOUSE_CLICKED,
MOUSE_PRESSED, MOUSE_RELEASED, MOUSE_MOVED, etc.
```

Adaptateurs

L'implémentation d'une interface demande l'écriture de *toutes* ses méthodes.

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

public class ButtonTest extends Applet {
    public void init() {
        Button button = new Button("Press Me");
        button.addMouseListener(new ButtonMouseListener());
        add(button);
    }
}

class ButtonMouseListener implements MouseListener {
    public void mouseEntered(MouseEvent event) {
        System.out.println("Mouse Entered Button");
    }
    public void mouseExited(MouseEvent event) {
        System.out.println("Mouse Exited Button");
    }
    public void mousePressed (MouseEvent event) { }
    public void mouseClicked (MouseEvent event) { }
    public void mouseReleased(MouseEvent event) { }
}
```

Le package `java.awt.event` contient des classes appelées *adapters*, et qui implémentent les interfaces. Par exemple, `MouseAdapter` implémente `MouseListener`. L'implémentation consiste à ne rien faire.

```

import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class ButtonTest2 extends Applet {
    public void init() {
        Button button = new Button("Press Me");
        button.addMouseListener(new ButtonMouseListener());
        add(button);
    }
}

class ButtonMouseListener extends MouseAdapter {
    public void mouseEntered(MouseEvent event) {
        System.out.println("Mouse Entered Button");
    }
    public void mouseExited(MouseEvent event) {
        System.out.println("Mouse Exited Button");
    }
}

```

Depuis la version 1.1, une classe peut être définie à l'intérieur d'une autre classe. Elle a alors un accès direct à l'objet qui provoque l'évènement.

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

public class ButtonTest2 extends Applet {
    public void init() {
        Button button = new Button("Press Me");
        button.addMouseListener(new ButtonMouseListener());
        add(button);
    }

    class ButtonMouseListener extends MouseAdapter {
        public void mouseEntered(MouseEvent event) {
            System.out.println("Mouse Entered Button");
        }
        public void mouseExited(MouseEvent event) {
            System.out.println("Mouse Exited Button");
        }
    }
}
```

La classe `ButtonMouseListener` est locale à la classe de l'applette (*inner class*), et son nom n'a d'autre rôle que de servir à la création d'un objet.

On peut économiser cette construction par une inner classe *anonyme*. (Ceci rappelle la possibilité, en C usuel, de ne pas nommer une struct, mais d'en utiliser la déclaration.)

On définit la classe au moment de l'instanciation; un groupe

```
new MaClasse()  
...  
class MaClasse extends ClasseBase {  
    ...  
}
```

devient

```
new ClasseBase() {  
    ...  
}
```

Dans notre exemple, ceci donne:

```
import java.awt.*;  
import java.awt.event.*;  
import java.applet.Applet;  
  
public class ButtonTest2 extends Applet {  
    public void init() {  
        Button button = new Button("Press Me");  
        add(button);  
        button.addMouseListener(new MouseAdapter() {  
            public void mouseEntered(MouseEvent event) {  
                System.out.println("Mouse Entered Button");  
            }  
            public void mouseExited(MouseEvent event) {  
                System.out.println("Mouse Exited Button");  
            }  
        });  
    }  
}
```

Ceci est une écriture bien plus lisible, et proche de la programmation Motif !

Conteneurs

Les conteneurs sont dérivés de la classe abstraite `Container`. La classe `Container` est responsable de la disposition des composants qu'elle contient. Elle est faite selon un `LayoutManager`.

Chaque conteneur dispose d'un `LayoutManager` par défaut:

```
Panel : FlowLayout  
Window, Dialog, Frame : BorderLayout
```

Les applettes et les applications autonomes ne fonctionnent donc pas de la même manière par défaut.

On modifie le gestionnaire par

```
setLayout(new BorderLayout());
```

pour avoir un gestionnaire de style `BorderLayout`. Les gestionnaires de géométrie sont:

```
BorderLayout  
FlowLayout  
GridLayout  
GridBagLayout  
CardLayout
```

Les deux derniers sont compliqués.

BorderLayout

divise ses composants en 5 régions : nord, sud, ouest, est, et centre.

Les composants "nord" et "sud" occupent toute la largeur, les composants "ouest" et "est" la hauteur qui reste, les composants "centre" la place restante.

FlowLayout

affiche les composants de la gauche vers la droite, en passant à la ligne s'il n'y a pas assez de place. C'est un *RowColumn* à orientation horizontale.

GridLayout

dispose se composants dans une grille. Une Form avec deux fractionBase différents.

`GridLayout(3,1)` : 3 lignes et une colonne.

Note sur FlowLayout

Tous les conteneurs prennent par défaut la taille minimale pour contenir leur composants.

Pour suggérer ou imposer une autre taille, on retaille par

`setSize(largeur, hauteur)`

ou en réécrivant la méthode `getPreferredSize()`, par exemple

```
public Dimension getPreferredSize() {  
    return new Dimension(100,150);  
}
```

Ceci est nécessaire lorsque l'on ajoute un Canvas, dont la taille est nulle par défaut.

```

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class CanvasTest extends Applet {
    public void init() {
        Canvas canvas = new ExampleCanvas();
        canvas.addComponentListener(new
DbgComponentListener());
        add(canvas);
    }
}

class ExampleCanvas extends Canvas {
    public void paint(Graphics g) {
        Dimension size = getSize();
        g.drawRect(0,0,size.width-1,size.height-1);
        g.setColor(Color.lightGray);
        g.draw3DRect(1,1,size.width-3,
            size.height-3,true);

        g.setColor(Color.blue);
        g.drawString("Canvas!",20,20);

        g.setColor(Color.orange);
        g.fillRect(10,40,20,20);
        g.setColor(Color.red);
        g.drawRect(9,39,22,22);

        g.setColor(Color.gray);
        g.drawLine(40,25,80,80);
        g.setColor(Color.black);
        g.drawLine(50,50,20,90);

        g.setColor(Color.cyan);
        g.fillArc(60,25,30,30,0,270);
    }
    public Dimension getPreferredSize() {
        return new Dimension(100,100);
    }
}

```

Dessin

La classe Component (et donc ses sous-classes) ont les méthodes

```
paint(Graphics)  
repaint()  
update(Graphics)
```

pour dessiner.

paint()	peint le document
repaint()	poste un update()
update()	repeint le fond puis appelle paint().

Chaque objet d'une classe dérivant de Component se voit affilié un objet de la classe Graphics. Cet objet est passé à la méthode paint(). On y accède aussi par getGraphics().

La classe Graphics possède 47 méthodes pour dessiner et peindre.

```

import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

public class CanvasTest extends Applet {
    public void init() {
        Canvas canvas = new ExampleCanvas();
        canvas.addComponentListener(new
DbgComponentListener());
        add(canvas);
    }
}

class ExampleCanvas extends Canvas {
    public void paint(Graphics g) {
        Dimension size = getSize();
        g.drawRect(0,0,size.width-1,size.height-1);
        g.setColor(Color.lightGray);
        g.draw3DRect(1,1,size.width-3,
            size.height-3,true);

        g.setColor(Color.blue);
        g.drawString("Canvas!",20,20);

        g.setColor(Color.orange);
        g.fillRect(10,40,20,20);
        g.setColor(Color.red);
        g.drawRect(9,39,22,22);

        g.setColor(Color.gray);
        g.drawLine(40,25,80,80);
        g.setColor(Color.black);
        g.drawLine(50,50,20,90);

        g.setColor(Color.cyan);
        g.fillArc(60,25,30,30,0,270);
    }
    public Dimension getPreferredSize() {
        return new Dimension(100,100);
    }
}

```

avec

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

public class DbgComponentListener implements
ComponentListener {
    public void componentResized(ComponentEvent event) {
        System.out.println("");
    }
    public void componentShown(ComponentEvent event) {
        System.out.println(event.toString());
    }
    public void componentMoved(ComponentEvent event) {
        System.out.println(event.toString());
    }
    public void componentHidden(ComponentEvent event) {
        System.out.println(event.toString());
    }
}
```

Choix

- Checkbox boutons à cocher et boutons radio
- Choice les combobox de Windows
- List les listes usuelles.

Comportement commun

Implémentent l'interface `ItemSelectable`, composé des méthodes

- `addItemListener(ItemListener)`
- `removeItemListener(ItemListener)`
- `Object[] getSelectedObjects()` retourne les objets sélectionnés ou null

L'interface `ItemListener` a la méthode

```
public abstract void itemStateChanged(ItemEvent e)
```

et un `ItemEvent` est engendré lorsqu'un item est sélectionné ou désélectionné.

Composé de (public static final int)

```
ITEM_FIRST  
ITEM_LAST  
ITEM_STATE_CHANGED  
SELECTED  
DESELECTED
```

Et de méthodes d'accès à l'origine de l'information.

Bouton à cocher

Un bouton à cocher a deux états.


```
setLayout(new GridLayout(3, 1));  
add(new Checkbox("one", null, true));  
add(new Checkbox("two"));  
add(new Checkbox("three"));
```

Un groupe de boutons est CheckboxGroup. Se comporte comme des boutons radio.

```
setLayout(new GridLayout(3, 1));  
CheckboxGroup cbg = new CheckboxGroup();  
add(new Checkbox("one", cbg, true));  
add(new Checkbox("two", cbg, false));  
add(new Checkbox("three", cbg, false));
```

public Checkbox getSelectedCheckbox() retourne le bouton coché.