AWT
Interfaces Utilisateur

Introduction

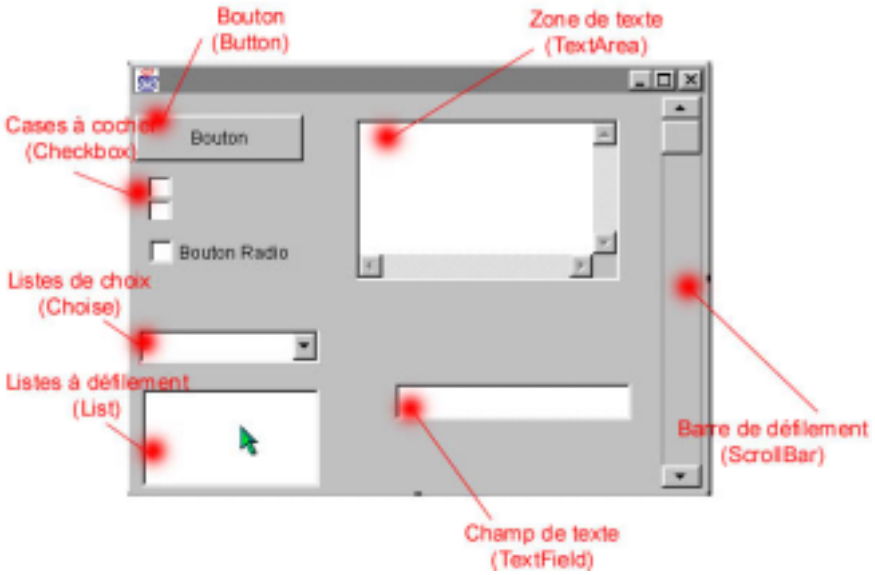
AWT (*Abstract Windowing Toolkit*) est une série de classes Java permettant de construire une interface utilisateur (*UI* en anglais ou *IU*). AWT fut la première UI de Java, elle est maintenant largement remplacé par Swing (Sauf pour les applets car Swing est une nouveauté de Java 2). Une interface utilisateur se compose principalement de :

- Une fenêtre de travail
- Une zone où afficher les composants, dans cette fenêtre de travail
- Une mise en page des composants (en ligne, en colone…)
- Des composants insérés dans cette fenêtre (boutons, cases à cocher, menus, barre de tâches…)
- Des *gestionnaires d'évènements* qui répondent par un comportement particulier aux actions de l'utilisateur.

Tout ceci est un peu compliqué à première vue et il peut paraître maladroit de dire qu'il faut définir une mise en page des composants avant de dire qu'il faut des composants, mais en Java, les opérations se font généralement dans cet ordre là.
Dans la mesure où AWT est utilisé majoritairement pour les applets, vous n'aurez le plus souvent pas à vous occuper de la création de la fenêtre de travail (il s'agit du navigateur) ni de la zone d'affichage des composants (il s'agit de l'applet elle-même). Ces 2 étapes sont cependant nécessaires à l'utilisation d'une UI dans une application.
Nous commencerons par étudier les composants, puis leur mise en page, car même si ces opérations sont inversées en programmation, il est plus facile de les concevoir dans cet ordre là, et en dernier les cadres…

Les Composants

La bibliothèque de composants AWT est assez limitée, Swing la complète. Nous ne verrons donc que les composants « classiques ».



1) Boutons

Il n'y a pas beaucoup d'explications à donner sur un bouton…

1. Button leNomDuBouton = new Button("Texte du bouton ici ! ");
2. add(leNomDuBouton);

2) Label

Un label est une zone d'affichage d'un texte. C'est le programmeur qui définit le texte est l'utilisateur ne peut pas le changer.

- label() // un label tout simple (et vide)
- label(String) // la chaîne String représente les texte à afficher dans ce label
- label(String, int) // int peut *être label.RIGHT*, *label.CENTER* ou *label.LEFT*, ce qui détermine //où le texte devra être affiché dans le label.

Pour le créer :

- label leNomDuLabel = new label("Bonjour! ", label.######);
- add(leNomDuLabel);
- leNomDuLabel.setText("Zoubida !!!!!!!!!! ");

Note: pour insérer un composant, la méthodes est toujours la même, elle ne sera donc par répétée.

On peut utiliser la méthode *setText()* pour redéfinir le texte du label (par exemple en résultat d'une action de l'utilisateur)

3) Cases à cocher

On coche une case pour valider l'option correspondante. On peut cocher toutes les cases, certaines, ou aucune.

- Checkbox() // case toute simple
- Checkbox(String) // String est le libellé de la case
-
- // Méthode à appeler sur un Checkbox pour cocher par défaut une option :
- setState(boolean)

4) Boutons Radio

Les boutons radio sont des sortes de cases, on ne peut cocher qu'une seule case dans tout le groupe.
Un exemple :

- CheckboxGroup p = new CheckboxGroup() ;
- Checkbox p1 = new Checkbox("Manger", p, true);
- Checkbox p2 = new Checkbox("Boire", p, false);
- // On rattache chaque bouton radio à un groupe, dans ce groupe, un seul est sélectionnable.
- // p1 est cochée par défaut (boolean = true)

5) Listes de choix

Les listes de choix sont des menus déroulants, on peut sélectionner une seule option.

- Choice c = new Choice() ; // création du menu
- c.addItem("Bonjour! "); // ajout d'une entrée avec Java 1.02
- c.add("Aurevoir!"); // ajout d'une entrée avec Java 2

Voici les méthodes que l'on peut employer avec *Choice* :

- getItem(int) // renvoir le n° de la case sélectionneé par l'utilisateur
- countItems() // renvoie le nombre ce cases, pour Java 2
- getItemsCount() // même chose pour Java 1.02
- getSelectedItem() // renvoie la position de l'objet sélectionné

- Select(int) // sélectionne l'élément n° x
- Select(String) // sélectionne l'élément ayant le nom indiqué pat String

6) Champs de texte

Ceci devient intéressant : ce composant permet de capturer un texte (court) entré par l'utilisateur.

- TextField() // vide
- TextField(int) // spécifie la largeur (déconseillé en Java 2)
- TextField(String) // la chaîne entrés dans le TextField
- TextField(String, int) // les 2 combinés

Méthodes utiles :

- getText() // retourne le texte contenu dans le champ
- setText() // définit un texte
- setEditable(boolean) // définit si l'utilisateur peut éditer le texte du champ, true = il peut
- isEditable() // indique si le champ est éditable (retourne un boolean)
- setEchoCharacter(' * ') // remplace les charactères tapés par une *, pour Java 1.02
- setEchoChar() // même chose pour Java 2

7) Zones de texte

Ce sont de grands champs de texte permettant à l'utilisateur de raconter sa vie ;-)
Les méthodes sont les mêmes que celles du champ de texte (daut setEchoChar()).

- TextArea() // tout simple
- TextArea(int, int) // nombre de lignes et largeur d'une ligne
- TextArea(String) // avec une valeur de départ …
- TextArea(String, int, int) // les 2 combinés

8) Listes à défilement

Ce sont des listes de choix dans lesquelles on peut sélectionner plusieurs entrées.

- List()
- List(int, boolean) // nombre d'entrées visibles en même temps, si plusieurs éléments peuvent //être sélectionnés (true), ou non (false)

9) Barres de défilement

Elles permettent de « scroller » ou de faire défiler le texte pour voir le bas d'une page (les barres à droite dans les traitements de texte, par exemple).

- ScrollBar(int, int, int, int, int)

Dans l'ordre, les *int* correspondent à :

- L'orientation : *Scorllbar.VERTICAL* ou *Scrollbar.HORIZONTAL*
- Valeur initiale du niveau de la barre de défilement
- Largeur globale du curseur, 0 = valeur de départ
- Valeur minimale de la barre
- Valeur maximale de la barre

Layouts

Il existe plusieurs types de *layout* qui correspondent à différentes techniques de disposition des composants. AWT possède par défaut 5 layouts : *FlowLayout, GridLayout, BorderLayout, CardLayout, GridBagLayout.* Si vous utilisez une IDE de développement graphique comme Borland JBuilder, Symantec VisualCafé ou IBM VisualAge, vous rencontrerez de nouveaux *layouts*, plus performants que ceux fournis par défaut. Ils possèdent cependant plusieurs défauts : premièrement il faut que l'utilisateur possède le package

duquel est issu le *layout*. Deuxièmement, les *proportions* dans le placement des composants n'est souvent pas conservé lors du redimentionnement de la fenêtre. Par exemple, un XYLayout spécifie les coordonnées du composant dans un cadre, si ce cadre est redimensionné, les proportions ne sont pas gardées ; alors qu'elles sont conservées avec un *layout* qui placerait un composant à 50% par rapport à la hauteur de la fenêtre et 40% en fonction de sa largeur.

10) FlowLayout

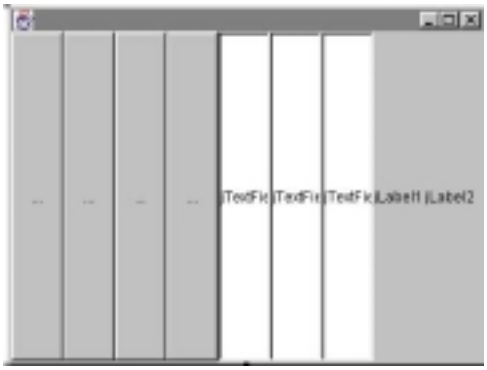
Ce *layout* dispose tous les composants à la suite, en ligne, de gauche à droite.

- FlowLayout flow = new FlowLayout(FlowLayout.LEFT) ;
- setLayout(flow);

11) GridLayout

GridLayout dispose les composants à la suite, sur 1 seule ligne.

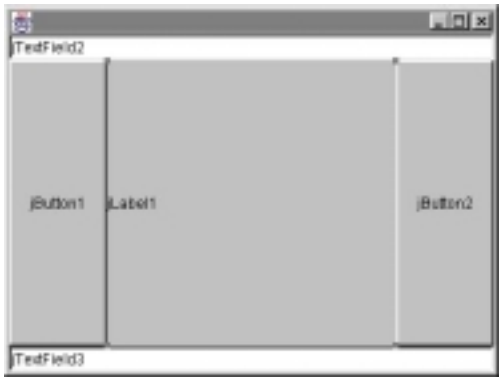
- BridLayout grid = new GridLayout(a,b,c,d) ;
- SetLayout(grid);
- // a = nombre de lignes
- // b = nombre de colones
- // c = espacement horizontal
- // d = espacement vertical



12) BorderLayout

Ce *layout* dispose les éléments en 5 parties : Nord, Sud, Est, Ouest, Centre.

- BorderLayout bd = new BorderLayout(a,b) ;
- SetLayout(bd);
- // a = espacement horizontal
- // b = espacement vertical
- Button b = new Button(" BBBB! ");
- Add("NORTH", b);
- // Au choix: NORTH, SOUTH, EAST, WEST, CENTER



13) CardLayout

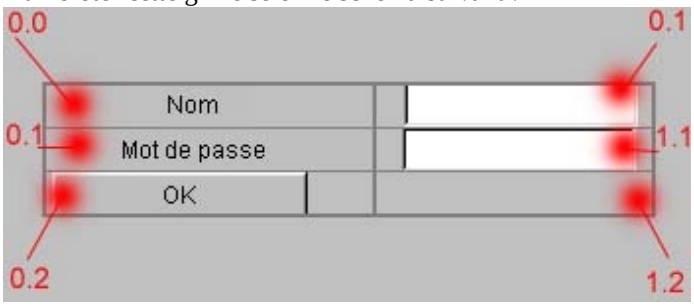
C'est un layout un peu particulier : « à onglets ».

14) GridBagLayout

C'est le plus intéressant des *layouts* proposés par AWT. Il est cependant plus compliqué d'utilisation.

- Concevoir une grille

Concevoir une grille, placer au moins 1 élément par case (1 élément peut prendre plusieurs cases.)
Numéroter cette grille selon le schéma suivant :



- Créer la méthode buildConstraints()

```
Void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int wx, int wy) {
Gbc.gridx = gx;
Gbc.gridy = gy;
Gbc.gridwidth = gw;
Gbc.gridheight = gh;
Gbc.weightx = wx;
Gbc.weighty = wy;
}

// gw et gh = le nombre de cellules sur lesquelles le composant s'étend
//  gx, gy = les coordonnees du composant, par exemple 0.2 pour le bouton OK
// wx et wy = largeur et profondeur des lignes.
```

- Ajouter les composants

```
// Création du layout
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints constraints  = new GridBagConstraints();
SetLayout(gridbag);


 // Exemple pour le label NOM
buildConstraints(constraints, 0, 0, 1, 1, 10, 40) ;
constraints.fill = GridBagConstraints.NONE ;
constraints.anchor = GridBagConstraints.EAST ;
Label label1 = new Label("Nom ", Label.LEFT) ;
Gridbag.setConstraints(label1, constraints) ;
Add(label1) ;

// Exemple pour le textfiels du mot de passe
buildConstraints(constraints, 1, 1, 1, 1, 0, 0) ;
constraints.fill = GridBagConstraints.HORIZONTAL ;
constraints.anchor = GridBagConstraints.CENTER ;
TextField tpas = new TextField() ;
Tpas.setEchoChar('*');
Gridbag.setConstraints(tpas, constraints) ;
Add(tpas) ;

// Faire de même pour les autres composants
```

Fill détermine dans quel sens l'*étirement* des composants doit se faire (vers la droite, gauche, haut, bas…) BOTH (2 sens), NONE (plus petite taille possible), HORIZONTAL (etirment horizontal), VERTICAL (…).
Anchor définit comment les composants qui ne remplissent pas toute une cellule doivent être disposés. NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTH, SOUTHEAST, WEST, NORTHWEST.

15) Insets

La méthode insets() (ou getInsets() en Java 2) détermine les espacements en HAUT, BAS, GAUCHE, DROITE (dans cet ordre) entre le bord de la fenêtre et le layout. (On l'utilise généralement avec GridLayout)

```
public Insets insets() {
    return new Insets(haut,bas , gauche, droite);
}
```

Exemple complet

Vous devez maintenant être en mesure de comprendre partiquement tout le code écrit pour mon applet *Convert*. Cette applet est disponible à l'adresse suivante : http://perso.wanadoo.fr/guillaume/Programmes/Convert/
Elle ne fonctionne pas avec IE5 ni Communicator 4.7, cependant vous pouvez l'utiliser avec *appletviewer* ou Netscape 6.
Les lignes générales de cette applet :

- BUT : convertir des Francs en Euros et vice-versa
- UI : AWT, GridBagLayout, 4 Labels, 4 TextFields, 1 Bouton
- DONNEES: type *long* pour le taux de conversion: *6,55957*
- METHODES : 1) Conversion d'euros en francs
- 2) Conversion de francs en euros
- METHODES UTILISEES SUR LES COMPOSANTS : *getText()* et *setText()*

Voici le code source :

```
import java.awt.*;


/************************************************************
 *                    Bonjour !                             *
 * Ceci est ma première applet. Sa fonction est de convertir des*
 * francs français en euros et vice-versa.                  *
 *                guillaume.florimond@wanadoo.fr             *
 ************************************************************/


public class Convert extends java.applet.Applet{
  // Définition des variables du programme
    String labelBouton = new String("                    Convertir !
");

// Le bouton contenant le texte "Convertir !"
    Button bouton = new Button(labelBouton);
// Case où taper le nombre * d' Euros à convertir
    TextField textEuro = new TextField(11);
// Case où taper le nombre * de Francs à convertir

TextField textFranc = new   TextField(11);
// Case où lire le résultat de la convertion
```

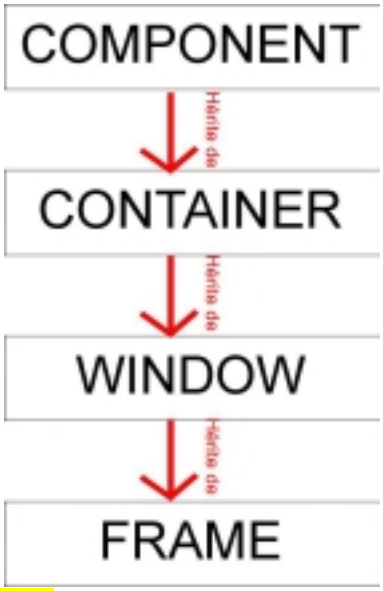
```
TextField textResultat = new TextField(26);

// Franc vers Euro
    TextField textResultat2 = new TextField(26);// Case où lire le résultat de la
convertion
                                                    // Euro vers Franc
// Définition de la police par défaut du programme

    Font f = new Font("Helvetica", Font.BOLD, 15);
Label titre = new Label("Convertisseur de Francs français en Euro",Label.LEFT);

    Label labelEuro = new Label("Valeur en Euro:",Label.RIGHT);
    Label labelFranc = new Label("Valeur en Franc:",Label.RIGHT);
    Label labelResultat = new Label("Résultat de la convertion (en
Euro):",Label.RIGHT);
    Label labelResultat2 = new Label("Résultat de la convertion (en
Franc):",Label.RIGHT);

    String FrancChaine = new String(); // Valeur entrée par l'utilisateur dans le
//TextField "textFranc"
    String EuroChaine = new String(); // Valeur entrée par l'utilisateur dans le
//TextField "textEuro"
    double taux = 6.55957; // Taux définif de convertion de l'euro en franc au
//1er Janvier 2000
    double FrancEntier; // Valeur convertie en "double" que l'utilisateur a entrée
//dans "textFranc"
    double EuroEntier; // Valeur convertie en "double" que l'utilisateur a entrée
//dans "textEuro"
    Label copyright = new Label("Copyright Guillaume Florimond 1999", Label.RIGHT);
// (...) ;-)


  void buildConstraints(GridBagConstraints gbc, int gx, int gy, int gw, int gh, int
wx, int wy) {
    gbc.gridx = gx;
    gbc.gridy = gy;
    gbc.gridwidth = gw;
    gbc.gridheight = gh;
    gbc.weightx = wx;
    gbc.weighty = wy;
  /* Les lignes précédentes représentent la définition des variables necessaires
pour la création
  d'une interface GridBagLayout */
  }

  public void init () {
    /* Les lignes suivantes représentent le paramétrage et l'insertion sur le
container principal Applet
    de la seule police utilisée, des couleurs de premier et second plan, ainsi que
des éléments propres
    à l'interface AWT tels que les TextArea et les Labels crées plus haut. */

    setFont(f); // police

    setBackground(Color.cyan); // couleur arrière plan
    setForeground(Color.magenta); // couleur de la police

    GridBagLayout gridbag = new GridBagLayout();
    GridBagConstraints constraints = new GridBagConstraints();
    setLayout(gridbag);

    // Titre
    buildConstraints(constraints, 1,0,4,1,0,5);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.CENTER;
    gridbag.setConstraints(titre, constraints);
    add(titre);

    // labelFranc
    buildConstraints(constraints, 0,1,1,1,30,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.EAST;
    gridbag.setConstraints(labelFranc, constraints);
```

```
    add(labelFranc);

    // labelEuro
    buildConstraints(constraints, 0,2,1,1,30,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.EAST;
    gridbag.setConstraints(labelEuro, constraints);
    add(labelEuro);

    // bouton
    buildConstraints(constraints, 0,3,2,1,0,0);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.CENTER;
    gridbag.setConstraints(bouton, constraints);
    add(bouton);

    // textFranc
    buildConstraints(constraints, 1,1,1,1,20,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.WEST;
    gridbag.setConstraints(textFranc, constraints);
    add(textFranc);

    // textEuro
    buildConstraints(constraints, 1,2,1,1,20,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.WEST;
    gridbag.setConstraints(textEuro, constraints);
    add(textEuro);

    // labelResultat
    buildConstraints(constraints, 2,1,1,1,30,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.EAST;
    gridbag.setConstraints(labelResultat, constraints);
    add(labelResultat);

    // labelResultat2
    buildConstraints(constraints, 2,2,1,1,30,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.EAST;
    gridbag.setConstraints(labelResultat2, constraints);
    add(labelResultat2);

    // textResultat
    buildConstraints(constraints, 3,1,1,1,20,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.WEST;
    gridbag.setConstraints(textResultat, constraints);
    add(textResultat);

    // textResultat2
    buildConstraints(constraints, 3,2,1,1,20,20);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.WEST;
    gridbag.setConstraints(textResultat2, constraints);
    add(textResultat2);

    // Copyright
    buildConstraints(constraints, 3,3,2,1,0,0);
    constraints.fill = GridBagConstraints.NONE;
    constraints.anchor = GridBagConstraints.CENTER;
    gridbag.setConstraints(copyright, constraints);
    add(copyright);

  }


  public void convertir (Button b) { /* ma méthode convertir() qui permet de
convertir des euros en francs et vice-versa appelée quand une action sur le bouton b
se produit */
    if (b == bouton) {  // si le bouton * est le bouton b
```

```
    FrancChaine = textFranc.getText(); // prendre la chaîne tapée par l'utilisateur
//dans textFranc
    FrancEntier = Double.parseDouble(FrancChaine); // convertir cette valeur en
//Double
    FrancEntier = FrancEntier / taux; // diviser par le taux de convertion officiel
    String resultat = new String(FrancEntier + ""); // convertir la valeur Double
//retournée en String
    textResultat.setText(resultat); // afficher cette valeur dans textResultat


    EuroChaine = textEuro.getText();
    EuroEntier = Double.parseDouble(EuroChaine);
    EuroEntier = EuroEntier * taux;
    String resultat2 = new String(EuroEntier + "");
    textResultat2.setText(resultat2);

    }

  }
  // les lignes suivantes définissent le gestionnaire d'évènement
  public boolean action(Event evt, Object arg) {
    if(evt.target instanceof Button) {
      convertir((Button)evt.target); // appeler la méthode convertir()

      return true;
    } else return false;
  }

}
```

Cadres

Nous avons vu que dans les cas des applets, un cadre où afficher les composants est par défaut le cadre d'affichage de l'applet. Pour l'instant, toutes les applications présentées dans ce cours étaient de ligne de commande, c'est à dire sans Interface Utilisateur. Nous allons maintenant en créer avec une UI, en définissant un cadre d'affichage grâce à un objet *Frame*. Voici un petit schéma d'héritage de la classe *Frame*.
Un cadre (*Frame*) est une sorte de fenêtre (*Window*), une fenêtre est un conteneur (*Container*) à composants, les cadres, fenêtres et conteneurs sont des composants Java.



Voici un exemple de création de fenêtre :

```
// Crée une nouvelle fenêtre du nom de Win
// "Ma fenêtre" est affiché comme intitulé de la fenêtre
Win = new Frame("Ma fenêtre");
// Assigner un Layout à cette fenêtre
Win.setLayout(new BorderLayout(10, 20);
// Ajouter 2 boutons à cette fenêtre
Win.add("North", new Button("Bonjour"));
Win.add("South", new Button("Aurevoir"));
// La réduire aussi petite que possible en fonction des composants
// qu'elle contient
Win.pack();
// La redimentionner à le taille voulue (en pixels)
Win.resize(100, 200);
// Afficher la fenêtre précédemment crée
win.show() ;
// La masquer
win.hide() ;
```

Gestion d'évènements

Vous savez maintenant créer un nouvelle fenêtre et y insérer des composants suivant le mode de mise en page désiré. Cependant, ces composants ne servent strictement à rien tant que vous ne leur assignez pas un comportement en réponse aux actions de l'utilisateur.
Avec AWT, cette gestion d'évènements se fait en 2 temps : on crée une méthode qui exécute les opérations données avec pour argument, par exemple, un bouton. Ensuite, on crée une seconde méthode qui répond aux actions d'un composant spécifié (pour continuer sur le même exemple, le bouton).
Pour donner un exemple concret, on peut reprendre l'applet Convert :

```
public void convertir (Button b) { /* ma méthode convertir() qui permet de convertir
des euros en francs et vice-versa appelée quand une action sur le bouton b se
produit */
    if (b == bouton) {  // si le bouton * est le bouton b
```

```
// les lignes suivantes définissent le gestionnaire d'évènement
  public boolean action(Event evt, Object arg) {
    if(evt.target instanceof Button) {
      convertir((Button)evt.target); // appeler la méthode convertir()

      return true;
    } else return false;
  }
```

Ce bouton réagit à toutes les actions de la même façon : il exécute le méthode convertir().
Cependant, on peut définir d'autres comportements en fonction du type d'action effectué (RollOver, RollOut…)

- boolean mouseDown(Event evt, int x, int y) // clic
- boolean mouseDrag() // déplacement avec bouton de souris enfoncé
- boolean mouseMove() // souris déplacée
- boolean mouseEnter() // souris entre dans une zone
- boolean mouseExit() // souris sort de cette zone
- boolean KeyUp() // touche du clavier relachée
- boolean KeyDown() // touche appuyée
- // etc…

Il existe bien d'autres comportements mais je ne vais pas les détailler car il ne sont que rarement utiles. Je ne vais pas non-plus parler de l'utilisation des évènements pré-cités car la gestion d'évènements est complètement remaniée avec Swing (celle d'AWT n'est vraiment pas performante). Je vous conseille pour de petits projets d'utiliser une méthode passe-partout comme *action(Event ev, Object arg)* accompagnée d'une instruction de reconnaissance du composant qui a subi l'action de l'utilisateur :
If (evt.target instanceof Button) {} (= si l'élément qui a subi l'action est une instance de la classe Button, exécuter le code entre accolades).

Schéma général d'héritage d'AWT