Interagir

Modèles d’évènement

- souris
- clavier

…

Composants

- boutons
- textes
- champs
- menus

…

Conteneurs et mise en pages

- protocoles de mise en page
- récursivité de la mise en page

Modèle d'événements de Java 1.1

Basé sur le principe général de la délégation.

Chaque événement est représenté par un objet

Contenu de l'objet évènement :

- la source de l'événement
- le moment auquel il a eu lieu
- des informations spécifiques (coordonnées souris, touche enfoncée, etc.).

Pour recevoir des évènements un objet doit :

- s'inscrire comme écouteur auprès d'un objet source d'événement ;
- posséder les méthodes requises pour traiter ce type d'événement.

Pour être source d'événements un objet doit :

- posséder une méthode d'inscription et de désincription d'écouteurs ;
- transmettre les événements en appelant les méthodes requises des écouteurs.

Gestion de la souris (1)

Pour recevoir des événements de type *MouseEvent*

Il faut

- implémenter l'interface *MouseMotionListener*.
  c-à-d les méthodes
  mouseMoved() et *mouseDragged()*.
- appeler la méthode *addMouseMotionListener()* de la source d'évènements pour s'inscrire.

Cas des applets :

une applet est sa propre source d'événements souris

On écrira donc l'appel d'inscription :

```
this.addMouseMotionListener(this);
```

Afficher en permanence les coordonnées de la souris :

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

public class Interaction extends java.applet.Applet implements MouseMotionListener
{
    int sourisX, sourisY;
    long quand;

   public void init() {
     this.addMouseMotionListener(this);
   }

   public void mouseMoved(MouseEvent e){
     // extrait les informations de l'événement
     quand=e.getWhen(); sourisX=e.getX(); sourisY=e.getY();
     repaint();
     }
   public void mouseDragged(MouseEvent e)  {}

   public void paint(Graphics g){
     g.drawString(quand+
     "=("+sourisX+","+sourisY+")",10,20);
   }
}
```

À l’aide d’un adaptateur (2)

Classes toutes faites, appelées **adaptateurs**, qui réalisent chacune un écouteur inactif.

On définit un écouteur par extension d’un adaptateur.

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

public class InteractionI extends java.applet.Applet{

   int sourisX, sourisY; long quand;

   class EcouteSouris extends MouseMotionAdapter {
     public void mouseMoved(MouseEvent e){
       quand=e.getWhen(); sourisX=e.getX();
       sourisY=e.getY();
       repaint();
     }
   }
   public void init() {
     this.addMouseMotionListener(new EcouteSouris());
   }
   public void paint(Graphics g) {
      g.drawString(quand+"=("+sourisX+
      ","+sourisY+")",10,20);
   }
}
```

Avec des classes anonymes (3)

pour éviter de définir explicitement une classe écouteur :

```
public class InteractionA extends java.applet.Applet {
   int sourisX, sourisY;
   long quand;

   public void init() {
     this.addMouseMotionListener(
      new MouseMotionAdapter(){
        public void mouseMoved(MouseEvent e){
         quand=e.getWhen(); sourisX=e.getX();
         sourisY=e.getY();
         repaint();
        }
      });
   }


   public void paint(Graphics g) {
     g.drawString(quand+"=("+sourisX+","+sourisY+")",10,20);
   }
}
```

Définition d’une classe anonyme

new MouseMotionAdapter() { … } =

- définit une classe qui étend *MouseMotionAdapter*
- crée un objet de cette classe (l’écouteur).

Ecriture compacte mais plus très lisible

On mélange des déclarations de méthodes à l’intérieur d’expressions.

Interface *MouseListener*

Clics:
mousePressed(), *mouseReleased()*, *mouseClicked()*,

Entrés/sortie du composant:
mouseEntered(), *mouseExited()*.

Extraire les information de l'événement de type *MouseEvent* reçu :

getWhen() : le temps auquel s'est produit l'événement ;

getX(), *getY()* : la position du pointeur ;

getClickCount() : le nombre de clicks sur ce même point ;

isAltDown(), *isAltGraphDown()*, *isControlDown()*, *isMetaDown()*, *isShiftDown()* : vrai si la touche (modificateur) Alt, AltGr, Ctrl, Meta ou Shift était enfoncée lors de l'événement ;

Souris à plusieurs boutons

getModifiers() : un codage sous forme d'un *int* des touches enfoncées.

On peut tester quel bouton a déclenché l'événement avec l'expression.

```
(getModifiers() & BUTTONi_MASK) // i = 1, 2 ou 3
```

Le résultat est différent de 0 si le bouton *i* a été pressé, relâché ou cliqué (c'est-à-dire pressé puis relâché).

Cependant, pour être vraiment portable, une application ne devrait pas faire d'hypothèses sur le nombre de boutons dont est munie la souris.

Gestion du clavier

Les événements clavier sont représentés par des objet de la classe *KeyEvent* qui étend *InputEvent*.

***keyPressed* ()**
invoquée lorsqu'une touche est enfoncée.

classe *keyEvent*

getKeyCode()
fournit le code de la touche.

pour un *clavier virtuel* indépendant de la machine utilisée.

Codes définis par une liste de constantes *KeyEvent.VK_xxx*.

getKeyChar()
fournit un caractère lorsque la ou les touches pressées correspondent à un caractère.

keyTyped()
n'est invoquée que lorsqu'un caractère est généré par la ou les touches pressées.

isAltDown()*, *isAltGraphDown()*, *isControlDown()*, *isMetaDown() et isShiftDown()
pour savoir quelle(s) touche(s) étaient enfoncée(s) au moment du clic.

Applet

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

public class KB extends java.applet.Applet {
   int taille=12;
   char toucheCourante='?';
   boolean shiftPresse;

   class Clavier extends KeyAdapter {
     public void keyPressed(KeyEvent e){
       int touche = e.getKeyCode();
       shiftPresse = e.isShiftDown();

       switch(touche){
         case KeyEvent.VK_DOWN: --taille; break;
         case KeyEvent.VK_UP: ++taille; break;
         default: toucheCourante=e.getKeyChar();
       }
       setFont(new Font("Helvetica", Font.BOLD,
               taille));
       repaint();
     }
   }
```

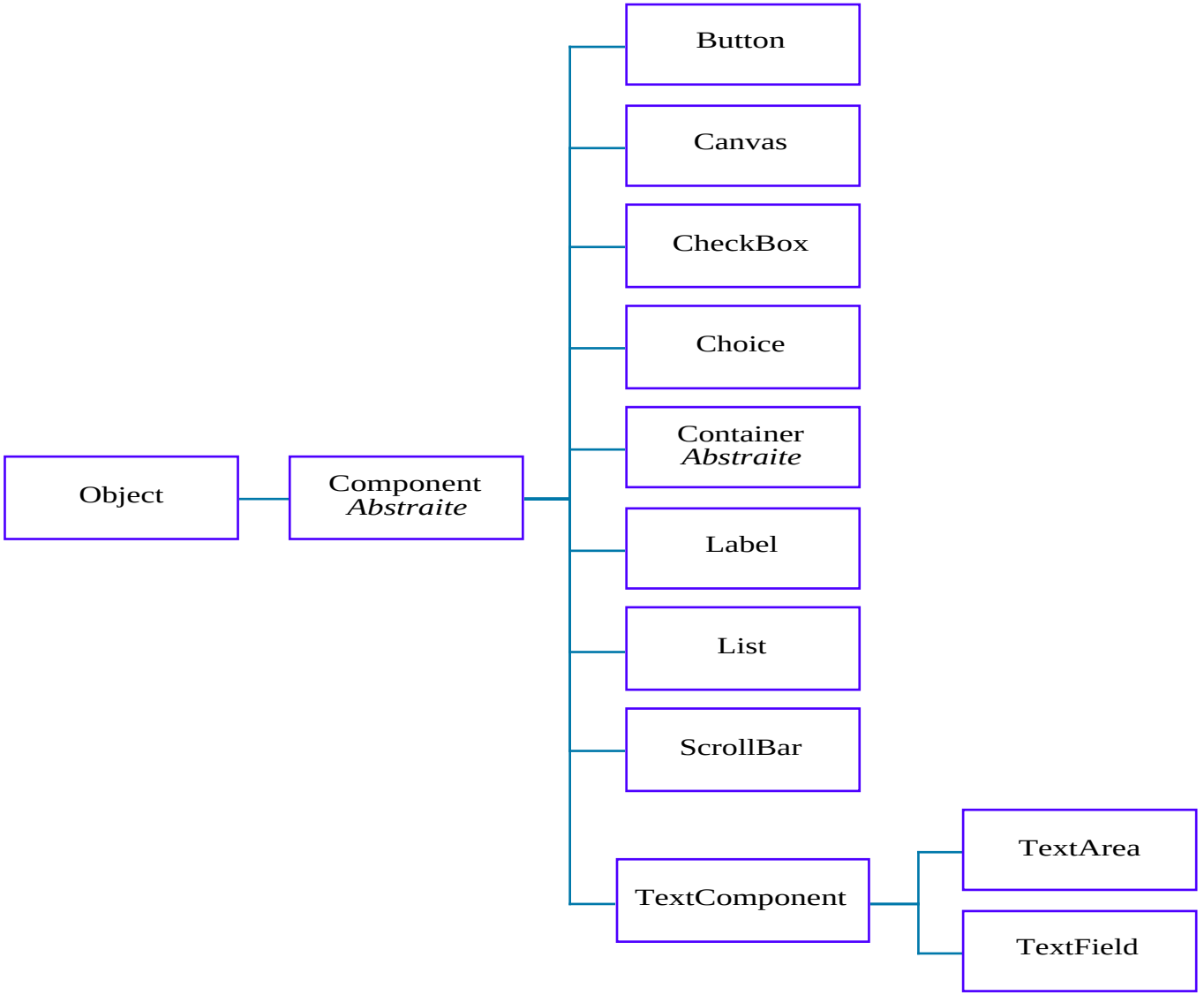
Applet -- suite

```
   public void init() {
     this.addKeyListener(new Clavier());
   }
   public void paint(Graphics g) {
     g.drawString("Taille="+taille+" majuscule="+
       shiftPresse, 10, 25);
     g.drawString(String.valueOf(toucheCourante),
       100,60);
   }
}
```

Événements et autres composants d'interaction.

Action qui crée l'événement	Type d'événement
Un utilisateur clique sur un bouton, choisit un article dans un menu, tape la touche *Return* dans un champ de texte	`ActionEvent`
Un utilisateur ferme une fenêtre	`WindowEvent`
Un utilisateur presse un bouton de la souris quand le curseur est sur un composant	`MouseEvent`
Un utilisateur bouge la souris sur un composant	`MouseMotionEvent`
Un composant se cache ou devient visible	`ComponentEvent`
Un composant reçoit l'attention du clavier	`FocusEvent`
Une table ou une liste de sélection est modifiée	`ListSelectionEvent`

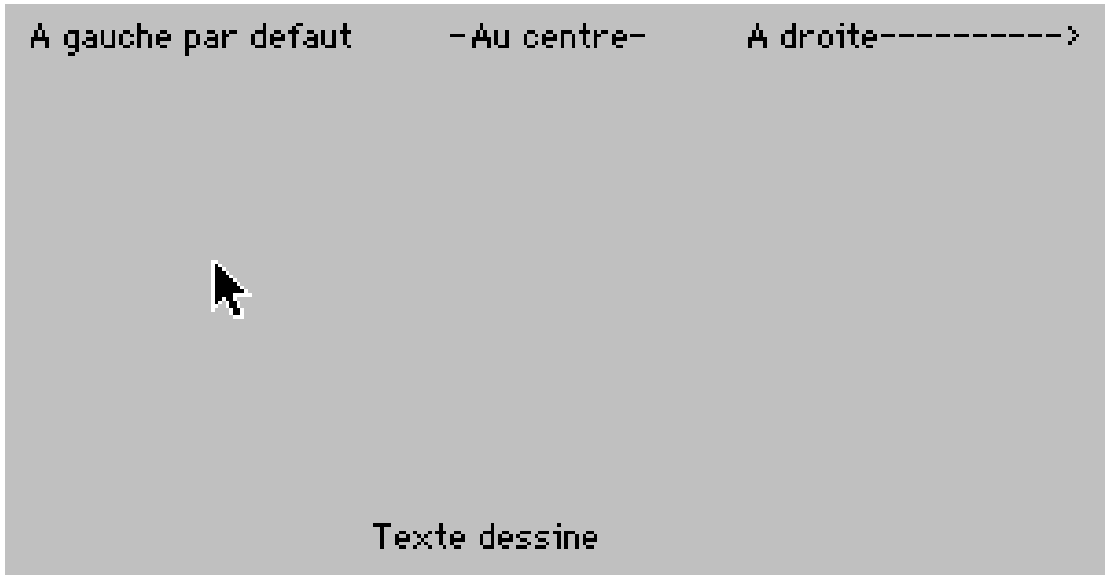
Composants d’interaction graphique



Les libellés

```
import java.awt.*;

public class Composant extends java.applet.Applet {

   public void init() {
     add(new Label ("A gauche par defaut"));
     add(new Label ("-Au centre-",Label.CENTER));
     add(new Label ("A droite---------->", Label.RIGHT));
   }

   public void paint(Graphics g) {
     g.drawString("Texte dessine", 100, 150);
   }
}
```

Effet



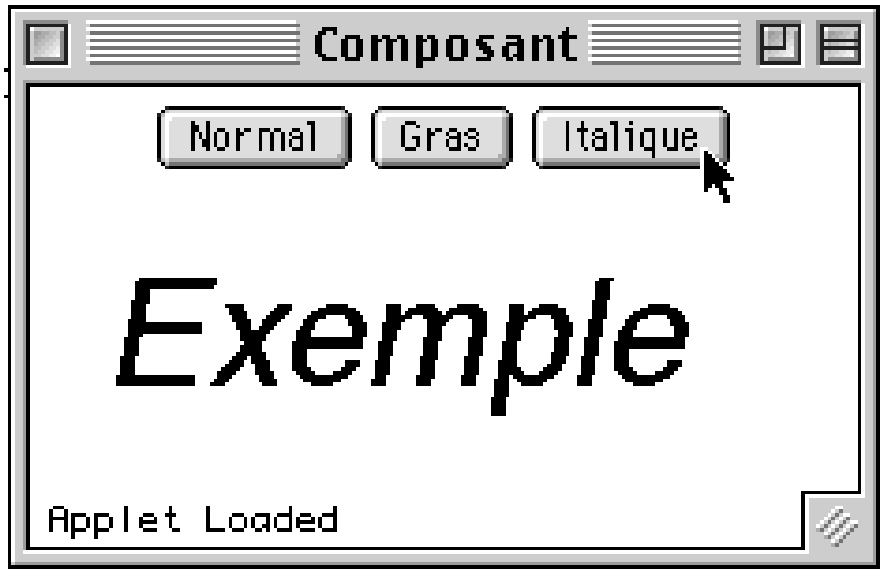
Les paramètres d'alignement sont des variables de classe constantes : *Label.CENTER*, *Label.LEFT*, *Label.RIGHT*.

Les boutons

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

public class Composant extends Applet implements ActionListener {

   String commande = "Normal";
   Button bn, bg, bi;
   Font fgras, fnormal, fitalic, fdef;

   public void init() {
     add(bn = new Button ("Normal"));
     add(bg = new Button("Gras"));
     add(bi = new Button("Italique"));
     bn.addActionListener(this);
     bg.addActionListener(this);
     bi.addActionListener(this);
     fgras   = new Font("Helvetica",Font.BOLD,40);
     fnormal = new Font("Helvetica",
                 Font.PLAIN,40);
     fitalic = new Font("Helvetica",
                 Font.ITALIC,40);
     fdef    = new Font("Helvetica",
                 Font.PLAIN,12);
   }
```

(suite)

```
 public void actionPerformed(ActionEvent e) {
   commande = e.getActionCommand();
   repaint();
 }
 public void paint(Graphics g) {
   if (commande.equals("Normal")) g.setFont(fnormal);
   if (commande.equals("Gras")) g.setFont(fgras);
   if (commande.equals("Italique"))g.setFont(fitalic);
   g.drawString("Exemple", 20, 80);
   g.setFont(fdef);
 }
}
```

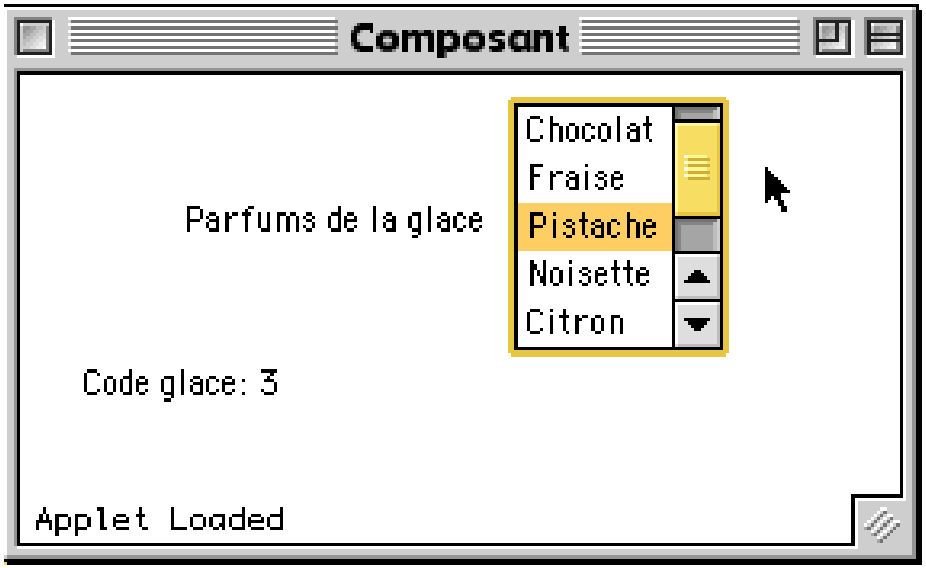


Les listes de choix

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

public class Composant extends Applet implements ItemListener  {

   List parfum;
   public void init() {
     parfum = new List(5,true);
     parfum.add("Vanille");
     parfum.add("Chocolat");
     parfum.add("Fraise");
     parfum.add("Pistache");
     parfum.add("Noisette");
     parfum.add("Citron");
     parfum.add("Orange");
     parfum.add("Abricot");
     parfum.setMultipleMode(true);
     add(new Label("Parfums de la glace"));
     add(parfum);
     parfum.select(3); // préselectionne Pistache
     parfum.addItemListener(this); // l'applet écoute le menu
   }
   public void itemStateChanged(ItemEvent e)  {
     repaint();
   }
```

suite

```
  public void paint(Graphics g) {
    int[] parfumNos = parfum.getSelectedIndexes();
    String message = "Code glace: ";
    for (int i=0; i<parfumNos.length; i++)
      message = message + parfumNos[i];
    g.drawString(message, 20, 100);
  }

}
```

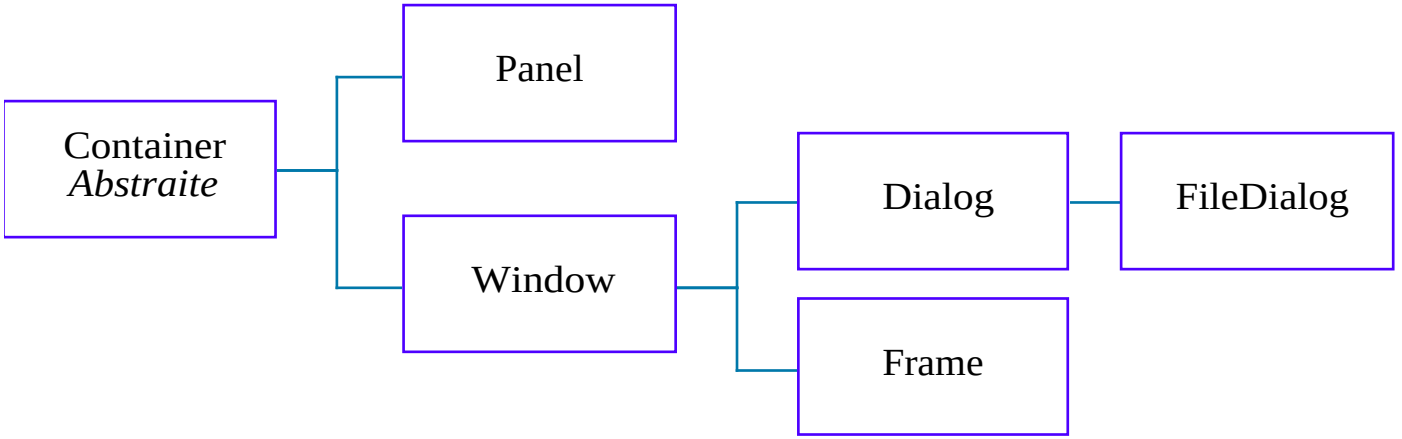
Les fonds

Les fonds sont des objets de la classe *Canvas*. Ils permettent de créer une surface de travail pour dessiner, afficher des images, etc. En étendant cette classe, il est possible de redéfinir le comportement d'un fond en surchargeant la méthode *paint()*.

Un *Canvas* est sensible aux événements générés par la souris et par le clavier.

Conteneurs

Un conteneur est un espace graphique destiné à recevoir plusieurs composants (boutons, fonds, listes, boîtes à cocher, etc.)



Ouvrir une autre fenêtre

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

class Fenetre extends Frame {
   TextArea t;

   Fenetre(String titre) {
     super(titre); // constructeur de Frame
     t = new TextArea("(x, y)", 5, 20);
     add("Center", t);
     pack();
     show();
   }
   public void afficher(String s) {
     t.append(s);
   }
}
```

Utilisation

```
class MonApplet ... {

   public void dessiner() {
     ...
     Graphics g = getGraphics();
     g.drawLine(x0, y0, e.getX(), e.getY());
     f1.afficher(" XXXXXX ");
     ...
   }
   public void init() {
     f1 = new Fenetre("f1");
     ...
   }
   public void stop() {
     f1.dispose();
   }
}
```

Un fenêtre plus active

```
class Fenetre extends Frame implements ActionListener {
   String commande;
   TextArea t;
   Button bn, bg, bi;

   Fenetre(String titre) {
     super(titre);
     setLayout (new FlowLayout(FlowLayout.LEFT,8,10));
     t = new TextArea("(x, y)", 5, 20);
     add(t);
     add(bn = new Button ("test")); add(bg = new Button("fermer"));
     add(bi = new Button("rien"));
     bn.addActionListener(this); bg.addActionListener(this);
     bi.addActionListener(this);
     pack();
     show();
   }
   public void afficher(String s) { t.append(s); }
    public void actionPerformed(ActionEvent e) {
     commande = e.getActionCommand();
     afficher("\n --> "+commande+"\n");
     repaint();
   }
}
```

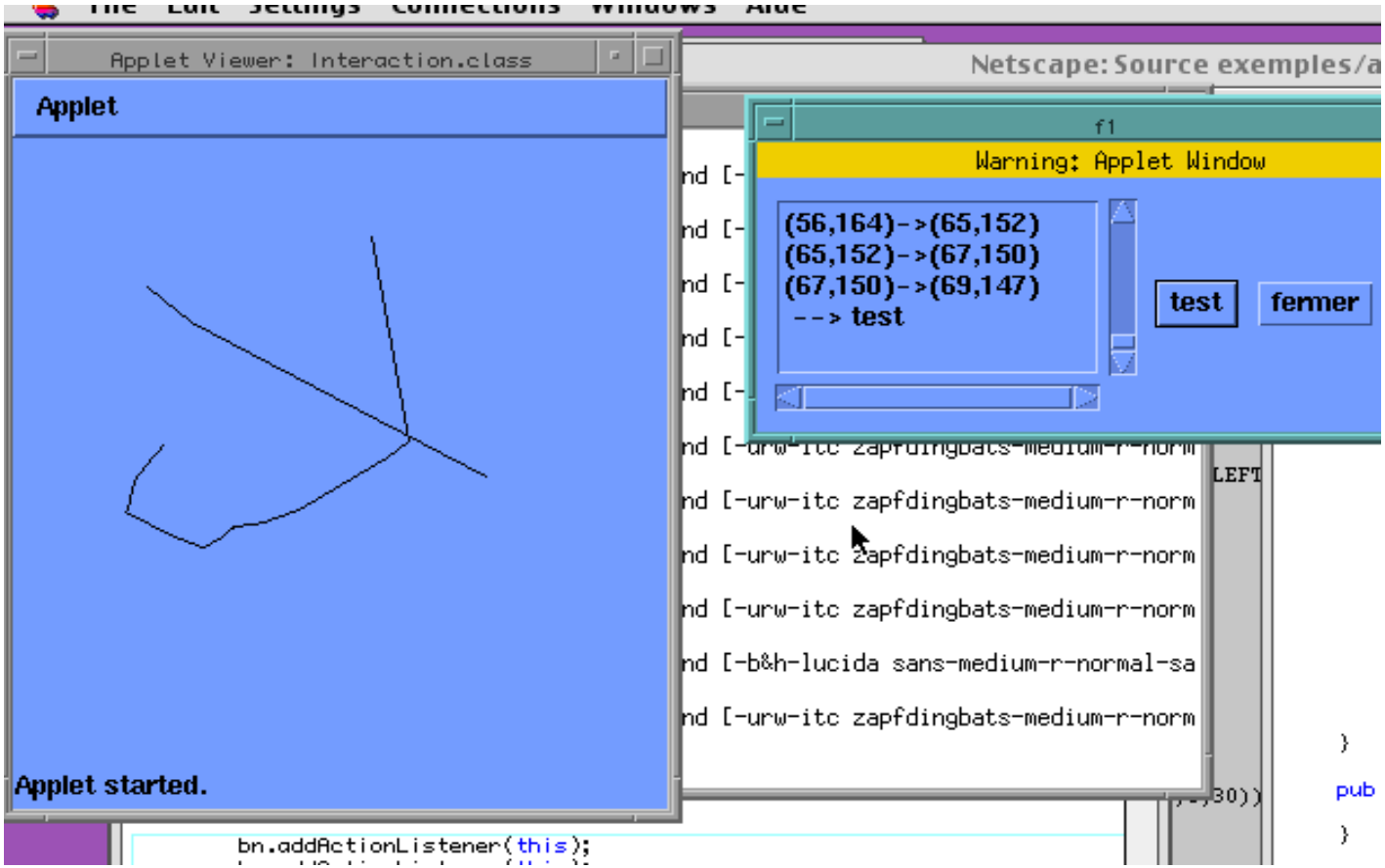
Applet

```
public class Interaction extends Applet {

   int x0 = 0, y0 = 0;
   Fenetre f1;


   class MSouris extends MouseMotionAdapter {

     public void mouseDragged(MouseEvent e) {
       Graphics g = getGraphics();
       g.drawLine(x0, y0, e.getX(), e.getY());
       f1.afficher
         ("\n("+x0+","+y0+")>("+
         e.getX()+","+e.getY()+")");
       x0 = e.getX(); y0 = e.getY();
     }
   }
```

suite

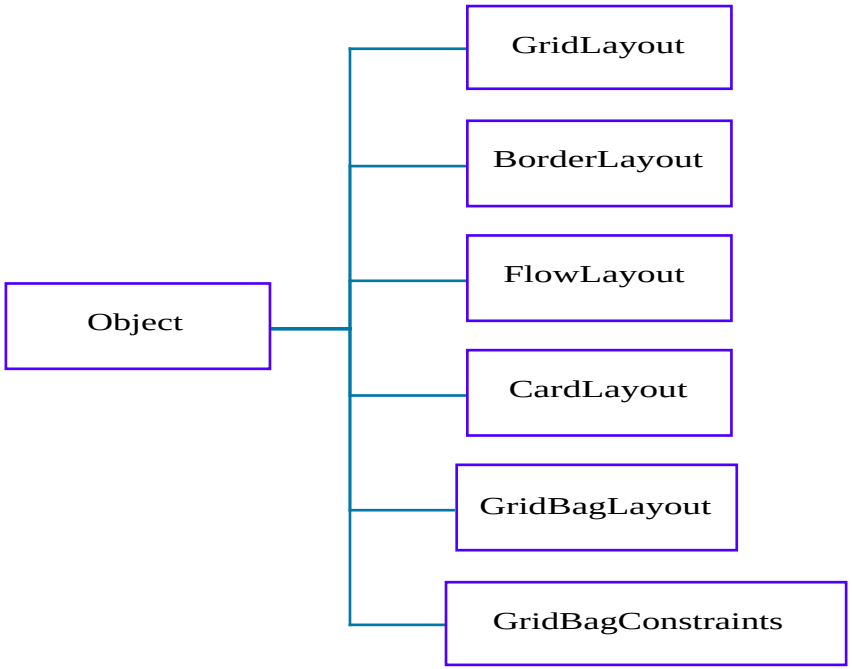
```
  class CSouris extends MouseAdapter {
    public void mousePressed(MouseEvent e) {
      x0 = e.getX(); y0 = e.getY();
    }
  }

  public void init() {
    f1 = new Fenetre("f1");
    addMouseListener(new CSouris());
    addMouseMotionListener(new MSouris());
  }
  public void stop() {
    f1.dispose();
  }
}
```

Apparence



Protocoles de mise en pages

Un protocole définit la méthode de placement des composants d'un conteneur les uns par rapport aux autres.



Chaque conteneur est associé à un seul protocole de mise en pages.

Les sous-composants qui sont des conteneurs peuvent avoir d'autres protocoles.

Permet de rester indépendant de la plate-forme physique.

On spécifie les contraintes de la mise en pages plutôt que des pixels sur la surface de travail.

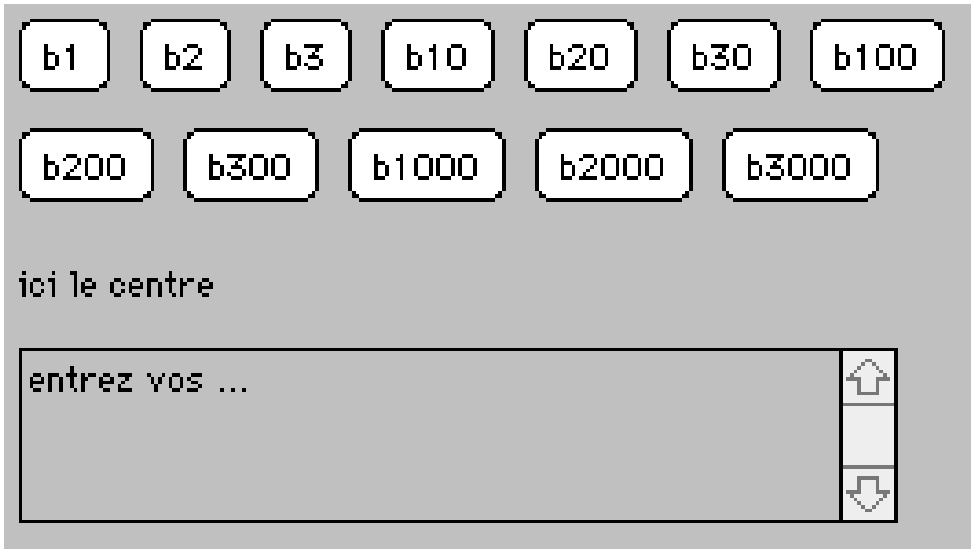
Mise en pages « glissante »

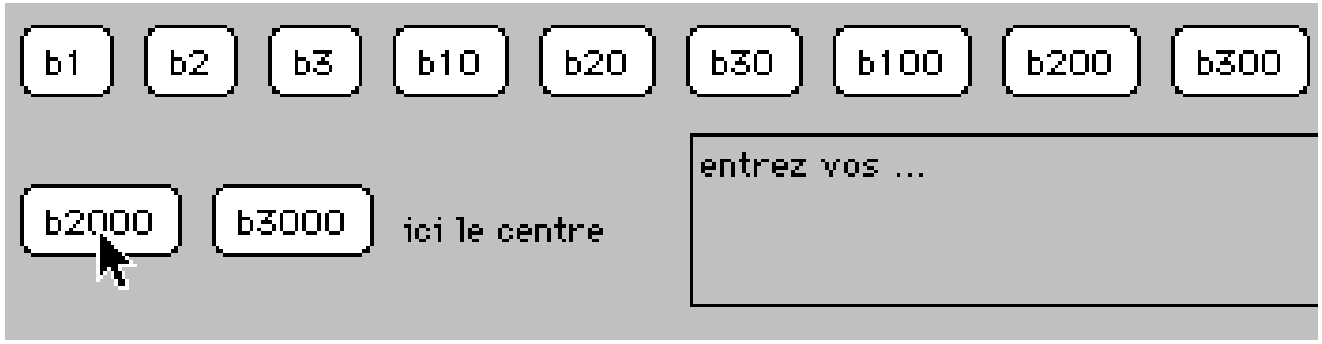
Les composants graphiques sont ajoutés l'un après l'autre de gauche à droite, avec un saut à la ligne dès qu'il ne reste plus d'espace suffisant à droite.

```
import java.awt.*;

public class Composant extends java.applet.Applet {

   public void init() {
     setLayout (new FlowLayout(FlowLayout.LEFT,8,10));
     add(new Button ("b1"));
     add(new Button ("b2"));
     add(new Button ("b3"));
     add(new Button ("b10"));
     add(new Button ("b20"));
     add(new Button ("b30"));
     add(new Button ("b100"));
     add(new Button ("b200"));
     add(new Button ("b300"));
     add(new Button ("b1000"));
     add(new Button ("b2000"));
     add(new Button ("b3000"));
     add(new Label ("ici le centre"));
     add(new TextArea ("entrez vos ...",4,30));
   }  }
```

Aspect

L’utilisation de protocoles de mise en pages permet d’adapter celle-ci dynamiquement lors du changement de la taille du conteneur.





Autres protocoles

Mise en pages par spécification des bords (5 zones North, South, East, West, Center)

Mise en pages avec une grille (n lignes m colonnes)

Mise en pages sous forme de cartes

Mise en pages avec un quadrillage et des contraintes

Récursivité de la mise en pages