Les SWING

Intérêt

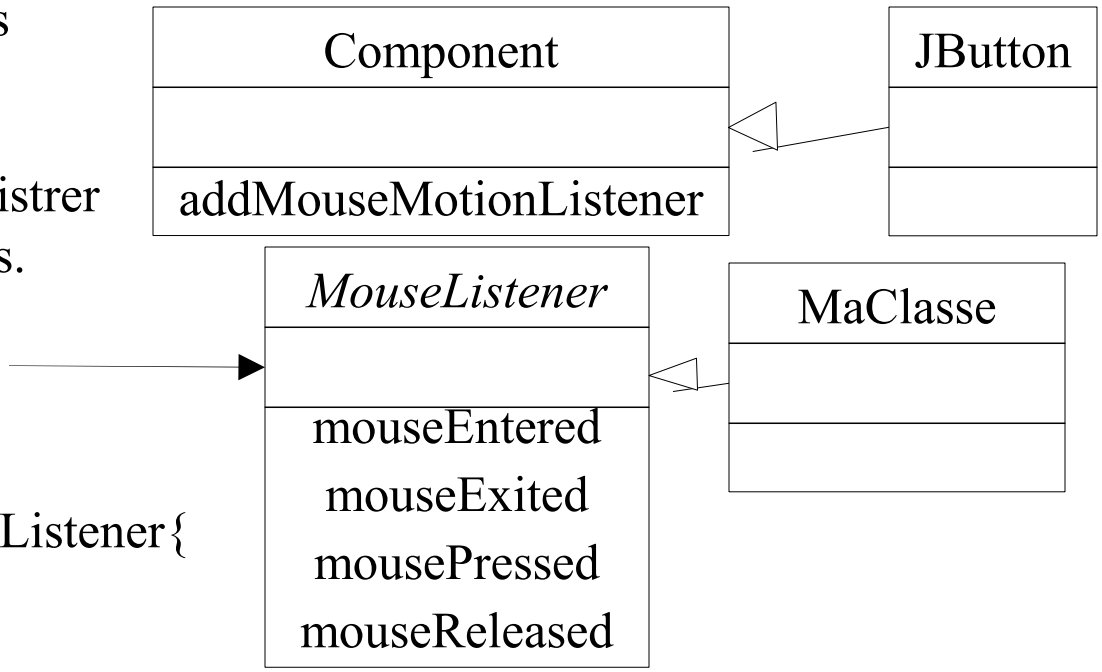
- Les Swing est un ensemble de classes permettant de construire des interfaces utilisateur en Java :
  - JFrame,
  - JDialog,
  - JButton
  - JPanel,
  - ...
- Les actions de l'utilisateur sur les composants SWING sont gérés par des événements (clic sur la souris, appui sur une touche, ...)

Le principe des événements

- Gérer les événements de la souris sur un composant.

1/ Un composant doit s'enregistrer comme récepteur d'événements.

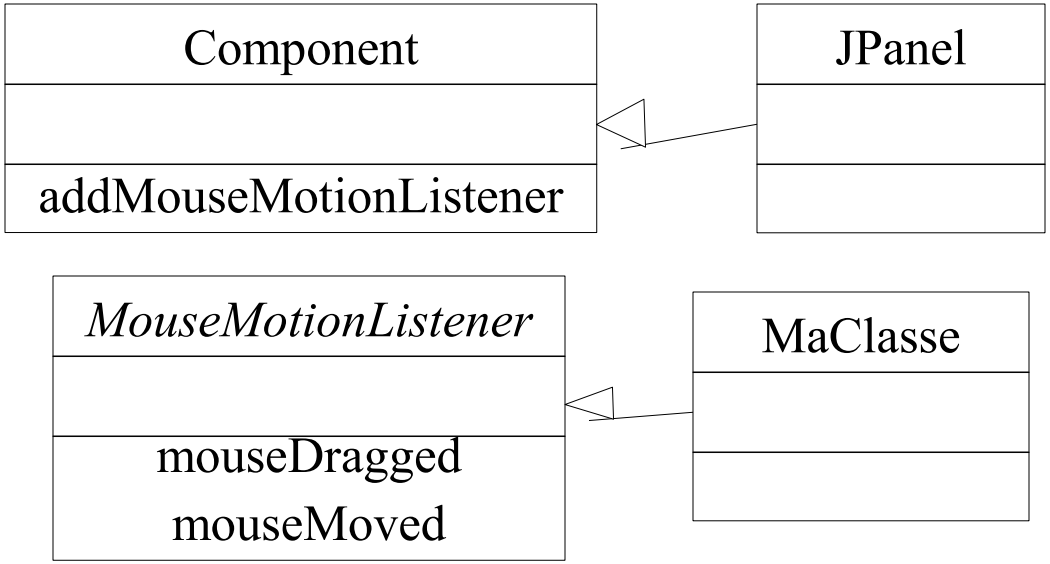
2/ Un objet doit traiter les événements reçus.




```
class MaClasse implements MouseListener{
      // ...
      {
            JButton boutonCommande = new JButton(  "Texte du bouton" ) ;
            bouton.addMouseListener( this ) ;      // s'enregistre en récepteur
      }
      public void mouseClicked( MouseEvent e ){ // … }     // recevoir les événements
      public void mouseEntered(MouseEvent e){} ;
      public void mouseExited(MouseEvent e){} ;
      public void mousePressed(MouseEvent e){} ;
      public void mouseReleased(MouseEvent e){} ;
}
```

Le principe des événements

- Gérer les événements de la souris pour faire des dessins sur un objet de type JPanel.



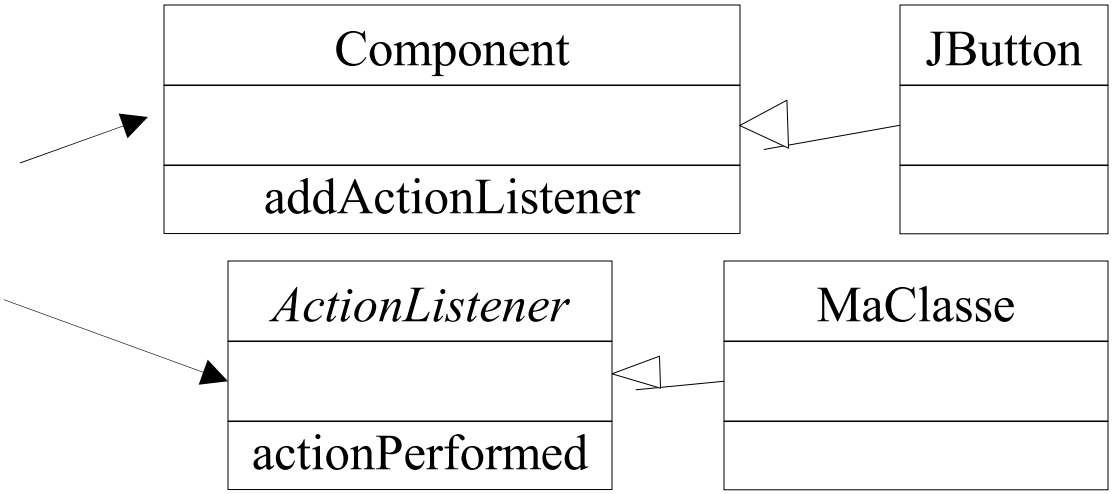

```
class MaClasse extends JPanel implements MouseMotionListener{
      // ...
      {
            addMouseMotionListener( this ) ;                    // s'enregistre en récepteur
      }
      public void mouseDragged( MouseEvent e ){ // … } ;   // recevoir les événements
      public void mouseMouved(MouseEvent e){ // … } ;
}
```

Le principe des événements

- Gérer les événements des composants : JButton, JList, JMenu, JTextField.

1/ Un Button peut s'enregistrer comme récepteur d'événements.

2/ Un objet avec un bouton doit traiter les événements reçus.



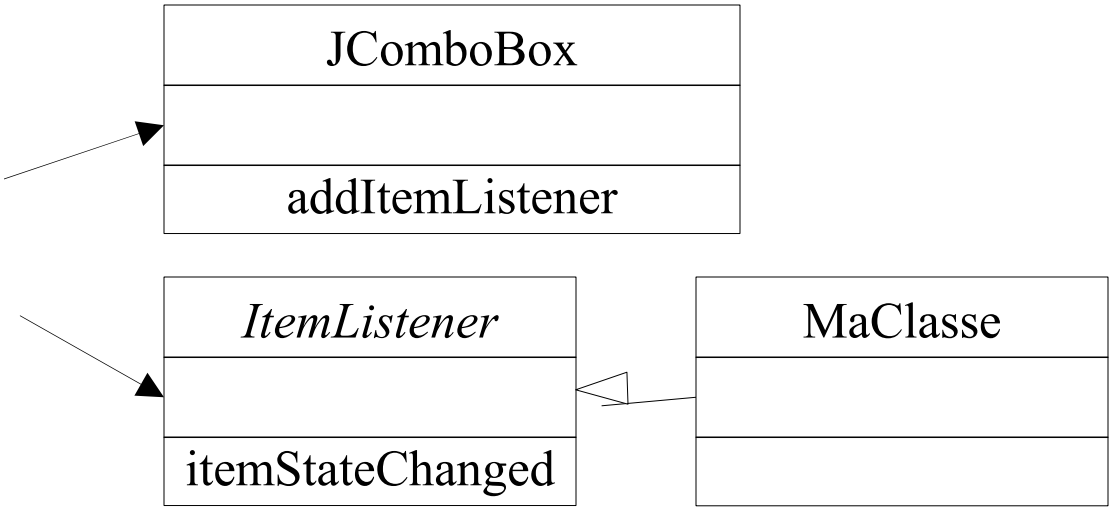

```
class MaClasse implements ActionListener{
      // ...
      {
            JButton bouton = new JButton(  "Texte du bouton" ) ;
            bouton.addActionListener( this ) ;                  // s'enregistre en récepteur
      }
      public void actionPerformed( ActionEvent e ){       // recevoir les événements
            // …
      }
}
```

Les événements

- Gérer les événements des composants : JComboBox, JCheckBox.

1/ Un Choix peut s'enregistrer comme récepteur d'événements.

2/ Un objet avec un choix doit traiter les événements reçus.




```
class MaClasse implements ItemListener{
      // ...
      {
            JComboBox box = new JComboBox() ;          // crée un sélecteur
            box.addItemListener( this ) ;              // s'enregistre en récepteur
      }
      public void itemStateChanged( ItemEvent e ){    // gère les événements
            // ...
      }
}
```

Les frames


```
public class Main{
   public static void main( String[] argv ){
       Fenêtre frame = new Fenêtre() ;
   }
}
public class Fenêtre extends JFrame impements WindowListener{
      public Fenêtre(){
            setTitle( "titre de la fenêtre" ) ;
            setSize( 200, 200 ) ;
            addWindowListener( this );
            setVisible( true ) ;
      }
      public void windowActived( WindowEvent e ){}
      public void windowClosed( WindowEvent e ){}
      public void windowClosing( WindowEvent e ){ dispose(); System.exit(0); }
      public void windowDeactived( WindowEvent e ){}
      public void windowDeiconified( WindowEvent e ){}
      public void windowIconified( WindowEvent e ){}
      public void windowOpened( WindowEvent e ){}
}
```

Les menus

```
public class Fenêtre extends JFrame {
    public Fenêtre(){
        // ...
        JMenu menu = new JMenu( "Fichier" ) ;

        JMenuItem item = new JMenuItem( "item" ) ;
        menu.add( item ) ;

        JMenuItem quitter = new JMenuItem( "Quitter" ) ;
        menu.add( quitter ) ;

        JMenuBar menuBarre = new JMenuBar() ;          // créé une barre de menu
        menuBarre.add( menu ) ;                        // ajoute un menu à la barre
        setJMenuBar( menuBarre ) ;                     // ajoute la barre à la Frame
    }
}
```

Gérer les événements des menus

```
public class Fenêtre extends Frame implements ActionListener {
   public Fenêtre(){
      // ...
      JMenuItem item = new JMenuItem( "item" ) ;
      item.addActionListener( this ) ;          // enregistrer en récepteurs d'événements
      menu.add( item ) ;
      JMenuItem quitter = new JMenuItem( "Quitter" ) ;
      quitter.addActionListener( this ) ;       // enregistrer en récepteurs d'événements
      menu.add( quitter ) ;
   }

   public void actionPerformed( ActionEvent e ){
      if( e.getActionCommand().equals( "Quitter" ) )  // item quitter :
         dispose() ;                                  // libérer ressource
      if( e.getActionCommand().equals( "item" ) )
         System.out.println( "item choisit" ) ;
   }
}
```

Les sélecteurs de choix


```
class Fenêtre extends JFrame implements ItemListener {
   Fenêtre(){
      // ...
      getContentPane().add( new Label( "choisir :" ) ) ;

      JComboBox choix = new JComboBox() ;
      choix.addItem( "choix 1" ) ;
      choix.addItem( "choix 2" ) ;
      choix.addItemListener( this ) ;
      getContentPane().add( choix ) ;
   }

    public void itemStateChanged( ItemEvent e ){      // gère les événements
      if( e.getItem().equals( "choix 1" ) )           // si premier item
            System.out.println( "premier choix" ) ;
      if( e.getItem().equals( "choix 2" ) )           // si deuxieme item
            System.out.println( "deuxieme choix" ) ;
   }
}
```

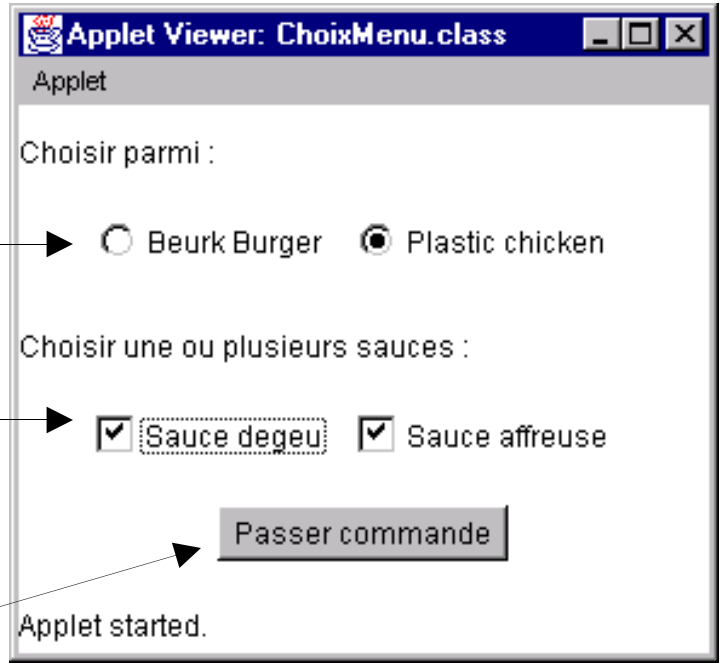
Les cases à cocher 1/4

JLabel

JCheckbox à choix unique :
CheckboxGroup

JCheckbox à choix multiple :
JCheckbox regroupée dans un Panel

JBouton dans un JPanel pour qu'il n'occupe pas la largeur de la fenêtre



Les cases à cocher 2/4

```
public class ChoixMenu extends JApplet{
   private JPanel saucePanel = new JPanel() ;
   private CheckboxGroup platGroup = new CheckboxGroup() ;

   public void init(){
      JCheckbox c ;
      JPanel platPanel = new JPanel() ;
      platPanel.add( c = new JCheckbox( "Beurk Burger" ) ) ;
      c.setCheckboxGroup( platGroup ) ;
      platPanel.add( c = new JCheckbox( "Plastic chicken" ) ) ;
      c.setCheckboxGroup( platGroup ) ;
      c.setSelected( true ) ;

      saucePanel.add( new JCheckbox( "Sauce degeu" ) ) ;
      saucePanel.add( new JCheckbox( "Sauce affreuse" ) ) ;
   }
}
```

Les cases à cocher : mise en forme 3/4

```
public class ChoixMenu extends JApplet{
   private JPanel saucePanel = new JPanel() ;
   private CheckboxGroup platGroup = new CheckboxGroup () ;

   public void init(){

      // ...
      JPanel platPanel = new JPanel() ;
      JPanel commandePanel = new JPanel() ;

      setLayout( new GridLayout( 5, 1 ) ) ;
      add( new JLabel( "Choisir parmi :" ) ) ;
      add( platPanel ) ;
      add( new JLabel( "Choisir une ou plusieurs sauces :" ) ) ;
      add( saucePanel ) ;
      add( commandePanel ) ; // voir page suivante
      resize( getPreferredSize() ) ;
   }
}
```

Les cases à cocher : gérer les événements 4/4

```
public class ChoixMenu extends JApplet implements ActionListener {

   public void init(){
      JPanel commandePanel = new JPanel() ;
      JButton boutonCommande = new JButton( "Passer commande" ) ;
      boutonCommande.addActionListener( this ) ;
      // ...
  }
    public void actionPerformed( ActionEvent e ){
      if( e.getActionCommand().equals( "Passer commande" ) ){
         JCheckbox c = platGroup.getCurrent() ;
         System.out.println( c.getLabel() ) ;
         Component [] composants = saucePanel.getComponents() ;
         for( int i=0; i<composants.length; i++ )
            if( (c = (JCheckbox)composants[i]).getState() )
               System.out.println( "Avec " + c.getLabel() ) ;
         System.out.println( "Merci !" ) ;
      }
   }
}
```

Les boîtes de dialogues 1/2

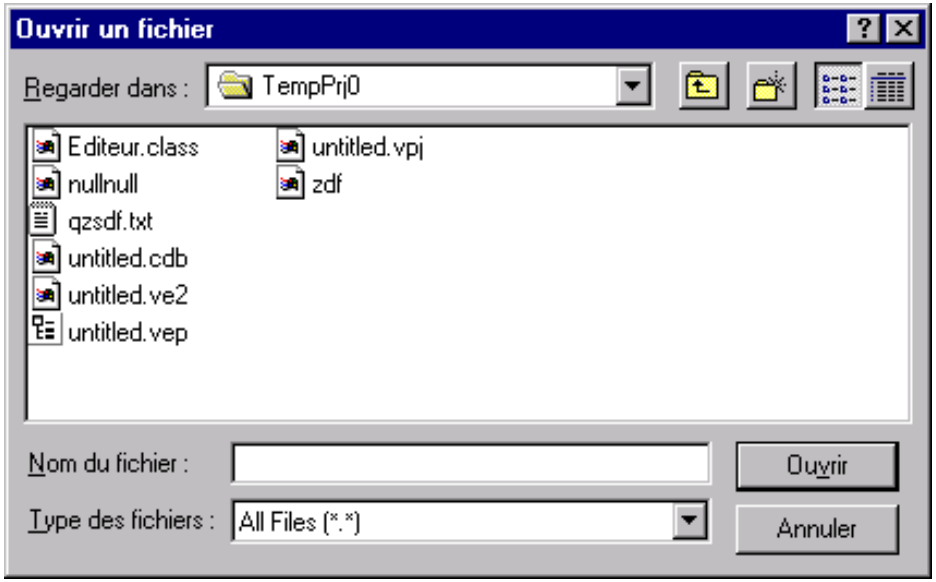

```
class MaBoite extends JDialog implements ActionListener{

    MaBoite(){
        setModal( true ) ;
        setLayout( new BorderLayout() );
        getContentPane().add( new JLabel( "Bart dit" ),
        BorderLayout.Center ) ;
        JPanel panel = new JPanel() ;
        JButton boutonCompris = new JButton( "Cool mec !" ) ;
        boutonCompris.addActionListener( this ) ;
        panel.add( boutonCompris ) ;
        JButton boutonPasCompris = new JButton(  "D'ho !" ) ;
        boutonPasCompris.addActionListener( this ) ;
        panel.add( boutonPasCompris ) ;
        getContentPane(). add( panel, BorderLayout.SOUTH ) ;
        pack() ;
        setVisible( true );
    }
  // …
}
```

Les boîtes de dialogues 2/2

```
class MaBoite extends JDialog implements ActionListener{

   public void actionPerformed( ActionEvent e ){
       if( e.getActionCommand().equals( "Compris !" ) )
          setVisible( false ) ;
          dispose();
       else
          System.out.println( "Comprends rien !" ) ;
    }

   public static void main( String [] argv ){
       MaBoite boite = new MaBoite()
    }
}
```

Les boîtes de sélection de fichiers : ouvrir un fichier 1/3

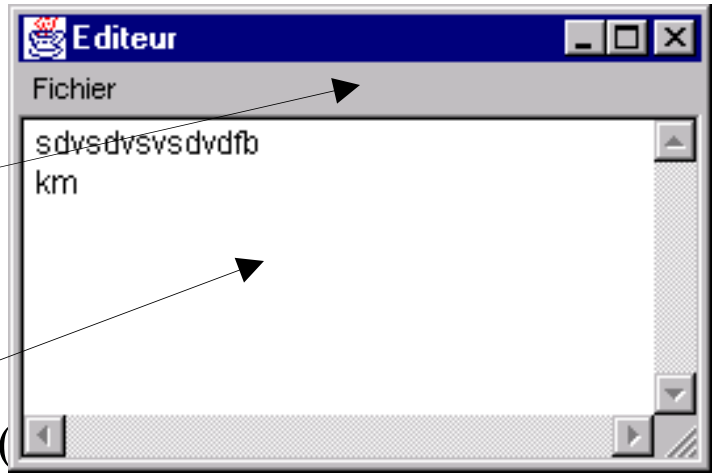
```
public class Editeur extends JFrame {
        public Editeur(){
                // gestion d'un menu
        }
        public void loadFile(){
                JFileChooser f = new JFileChooser();
                chooser.showOpenDialog( this );
        }
        public static void main( String [] argv ){
                new Editeur();
        }
}
```



Les boîtes de sélection de fichiers : ouvrir un fichier 2/3

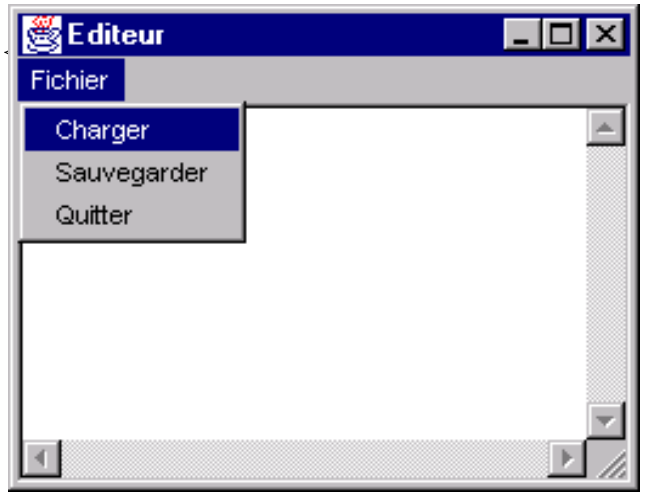
```
public class Editeur extends JFrame {

    public void loadFile(){
        JFileChooser chooser = new JFileChooser();
        int returnVal = chooser.showOpenDialog( this );
        if( returnVal == JFileChooser.APPROVE_OPTION ){
        String fileName = chooser.getSelectedFile().getName();
            try{
                FileInputStream fis = new FileInputStream( fileName ) ;
                byte [] donnees = new byte[ fis.available() ] ;
                fis.read( donnees ) ;
                fis.close() ;
                // ...
            } catch( IOException e ) {
                // ...
            }
        }
    }
}
```

Les boîtes de sélection de fichiers : ouvrir un fichier 3/3




```
public class Editeur extends JFrame {

    private JTextArea zoneTexte = new JTextArea(

    public void loadFile(){
        // ...
        try{
            // ...
            zoneTexte.setText( new String( donnees, 0 ) ) ; // affiche dans zoneTexte
        } catch( IOException e ) {
            zoneTexte.setText( "Impossible de charger le fichier : " + e.toString() ) ;
        }
    }
}
```

Les boîtes de sélection de fichiers : utiliser un menu




```
class Editeur extends JFrame implements ActionListener {

  public Editeur(){
        super( "Editeur" ) ;
        add( "Center", zoneTexte ) ;        // ajout du texte

        JMenu menu = new JMenu( "Fichier" ) ;
        JMenuItem charger = new JMenuItem( "Charger" ) ;
        charger.addActionListener( this ) ;       // s'enregistre en récepteur d'événements
        menu.add( charger ) ;                     // ajoute le menuItem au menu
        // idem pour Sauvegarder
        // idem pour Quitter

        JMenuBar menuBar = new JMenuBar() ;  // créé une barre de menu
        menuBar.add( menu ) ;                     // ajoute le menu
        setJMenuBar( menuBar ) ;                  // ajoute la barre à la frame
        pack() ;                                  // ajuste la taille des composants
  }
}
```

Les boîtes de sélection de fichiers : intercepter les événements

```
public class Editeur extends JFrame implements ActionListener{


    public void actionPerformed( ActionEvent e ){

        if( e.getActionCommand().equals( "Charger" ) )
            loadFile() ;
        }

}
```

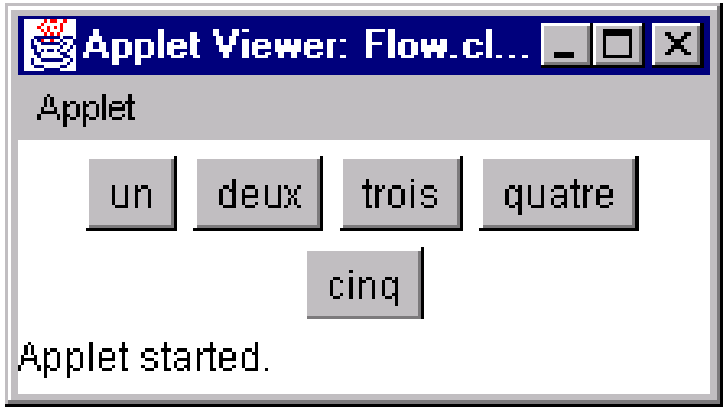
Le gestionnaire de placement

- Le gestionnaire de placement (*Layout) permet de disposer le composants (Component) dans un conteneur (Container) :
  - flow layout,
  - border layout,
  - grid layout,
  - Card layout,
  - Box layout,
  - ...

Le gestionnaire de placement FlowLayout

- Les conteneurs Panel et Applet ont le gestionnaire de placement FlowLayout par défaut : les composants sont placés de gauche à droite et de haut en bas.

```
public class Flow extends JApplet{
    public void init(){
        add( new JButton( "un" ) ) ;
        add( new JButton( "deux" ) ) ;
        add( new JButton( "trois" ) ) ;
        add( new JButton( "quatre" ) ) ;
        add( new JButton( "cinq" ) ) ;
    }
}
```

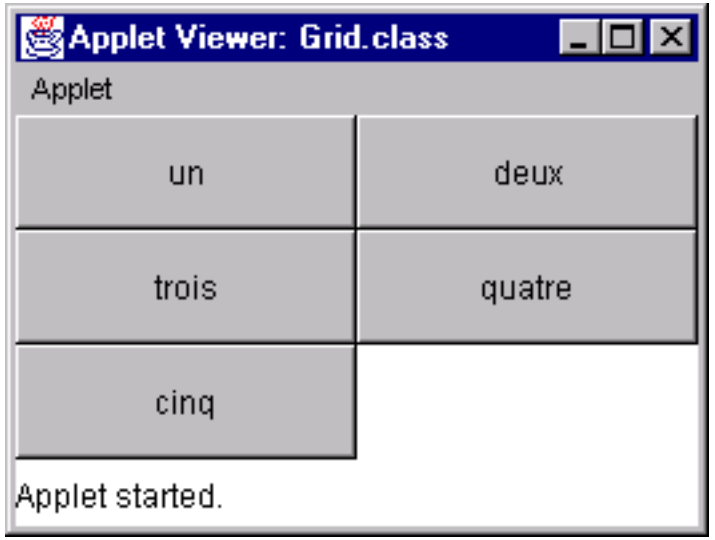


Le gestionnaire de placement GridLayout

- Le gestionnaire de placement GridLayout place les composants selon une grille ;
- mettre en place un nouveau gestionnaire se fait avec

  setLayout(new GridLayout(3, 2)) ;

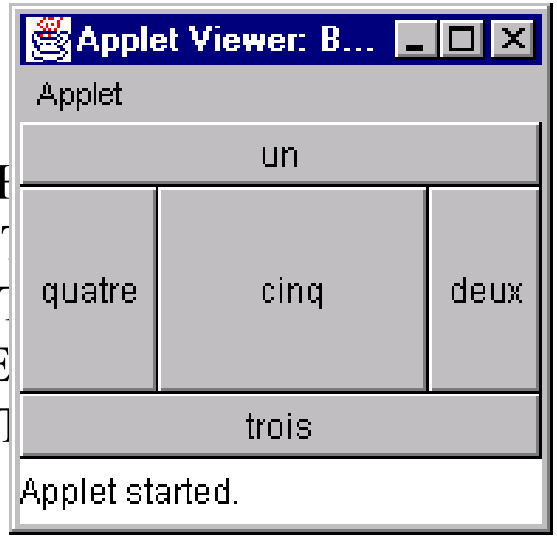
```
public class Grid extends JApplet{
    public void init(){
        setLayout( new GridLayout( 3, 2 ) ) ;
        add( new JButton( "un" ) ) ;
        add( new JButton( "deux" ) ) ;
        add( new JButton( "trois" ) ) ;
        add( new JButton( "quatre" ) ) ;
        add( new JButton( "cinq" ) ) ;
    }
}
```



Le gestionnaire de placement BorderLayout

- Le gestionnaire de placement GridLayout place les composants selon une position géographique par rapport au bord de la fenêtre.

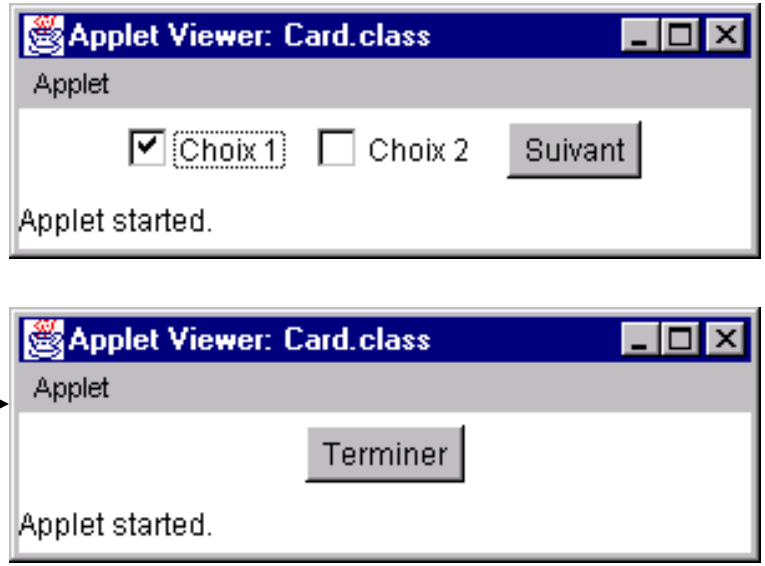
```
public class Border extends JApplet{
    public void init(){
        setLayout( new BorderLayout() ) ;
        add( new JButton( "un" ), BorderLayout.NOTRH
        add( new JButton( "deux" ), BorderLayout.EAS
        add( new JButton( "trois" ), BorderLayout.SOU
        add( new JButton( "quatre" ), BorderLayout.WE
        add( new JButton( "cinq" ), BorderLayout.CENT
    }
}
```



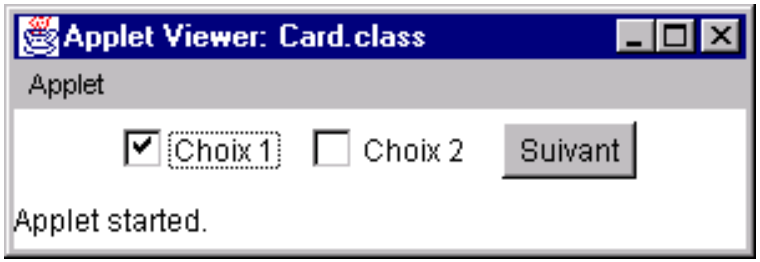
Le gestionnaire de placement CardLayout 1/2

- Le gestionnaire de placement CardLayout permet d'enchaîner des écrans.

```
public class Card extends JApplet implements ActionListener{
   private CardLayout cartes = new CardLayout() ;
   private JPanel panel1 = new JPanel() ;
   private JPanel panel2 = new JPanel() ;

   public void init(){
      setLayout( cartes ) ;
      add( "un", panel1 ) ;
      add( "deux", panel2 ) ;
   }

   public void actionPerformed( ActionEvent e ){
      if( e.getActionCommand().equals( "Suivant" ) )
         cartes.next( this ) ;              // passe au panel suivant
      if( e.getActionCommand().equals( "Terminer" ) ){
         // ...
      }
   }
}
```



Le gestionnaire de placement CardLayout 2/2




```
public class Card extends JApplet implements ActionListener{
   private JPanel panel1 = new JPanel() ;

   public void init(){
      panel1.add( new JCheckbox( "Choix 1" ) ) ;
      panel1.add( new JCheckbox( "Choix 2" ) ) ;
      JButton bouton = new JButton( "Suivant" ) ;
      bouton.addActionListener( this ) ;
      panel1.add( bouton ) ;
   }

   // ...
}
```

Les
view
et les
Model

Le view et les model illustrés par une JTable


```
public class MixerInfoPanel extends JPanel{
        public MixerInfoPanel(){

                super(new GridLayout(1,0));

                JTable table = new JTable( new MyTableModel() );
                table.setPreferredScrollableViewportSize(new Dimension(700, 70));
                table.setDefaultRenderer( Mixer.Info.class, new MixerRenderer() );
                table.setDefaultEditor( Mixer.Info.class, new MixerEditor() );
                JScrollPane scrollPane = new JScrollPane( table );

                add( scrollPane );
        }
}
```

Le view et les model illustrés par une JTable

```
class MyTableModel extends AbstractTableModel {
	private String[] columnNames = { "", "nom", "description", "vendeur", "version" };

	private Object[][] data;

	public MyTableModel(){
		Mixer.Info[] infos = AudioSystem.getMixerInfo();
		data = new Object[infos.length][5];
		for( int i=0; i<infos.length; i++ ){
			data[i][0] = infos[i];
			data[i][1] = infos[i].getName();
			data[i][2] = infos[i].getDescription();
			data[i][3] = infos[i].getVendor();
			data[i][4] = infos[i].getVersion();
		}
	}

}
```

Le view et les model illustrés par une JTable

```
class MyTableModel extends AbstractTableModel {
        public int getColumnCount() {
                return columnNames.length;
        }
        public int getRowCount() {
                return data.length;
        }
        public String getColumnName(int col) {
                return columnNames[col];
        }
        public Object getValueAt(int row, int col) {
                return data[row][col];
        }
        public Class getColumnClass(int c) {
                return getValueAt(0, c).getClass();
        }
        public boolean isCellEditable(int row, int col) {
        if (col > 0)  return false;
        else return true;
        }
}
```

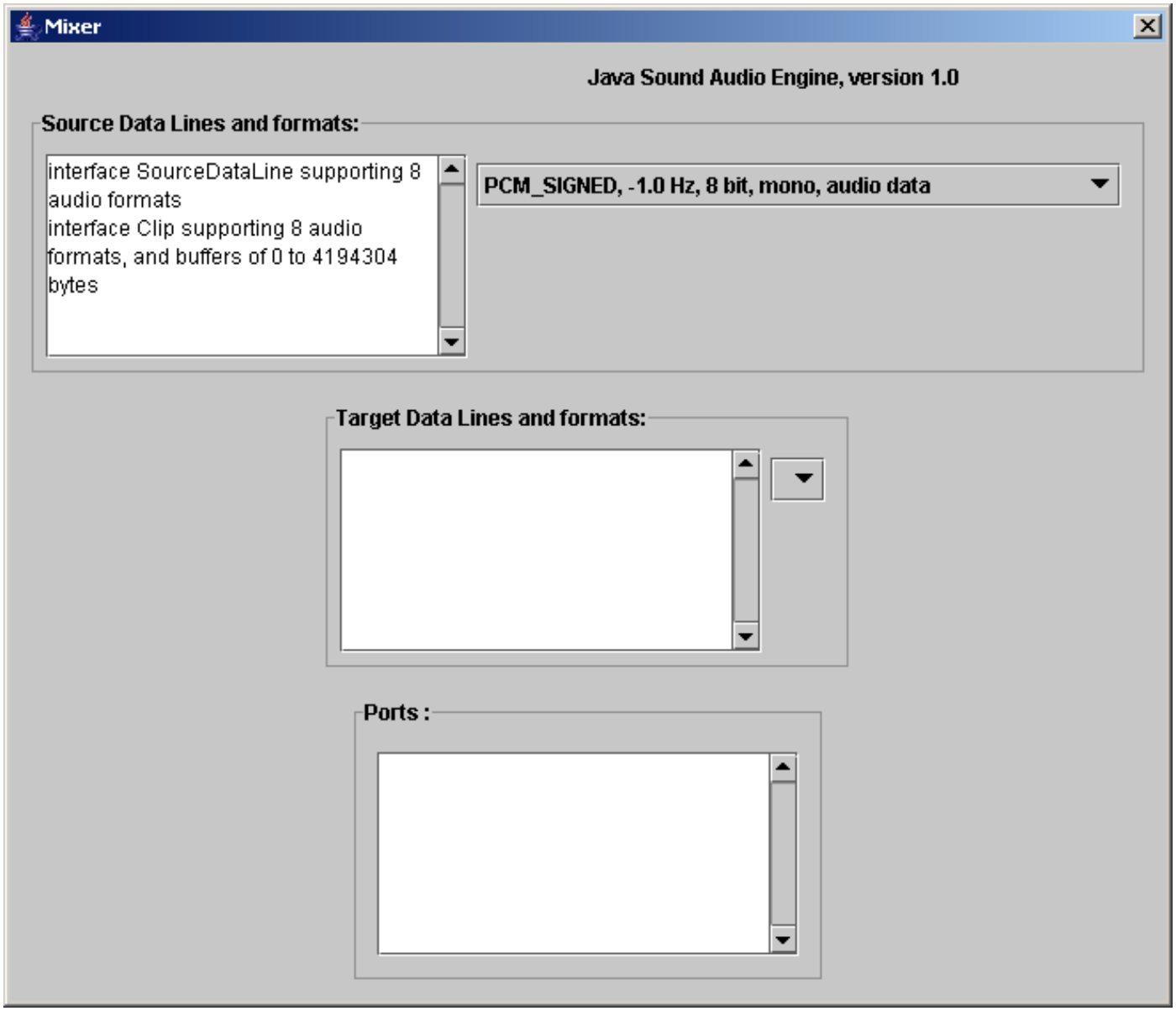
Le view et les model illustrés par une JTable

```
class MixerEditor extends AbstractCellEditor implements TableCellEditor, ActionListener{
   Mixer.Info info;
   JButton button;
   protected static final String EDIT = "afficher les caractéristiques";
    public MixerEditor() {
        button = new JButton();
        button.setActionCommand(EDIT);
        button.addActionListener(this);
    }
    public void actionPerformed(ActionEvent e) {
        if (EDIT.equals(e.getActionCommand())) {
                JDialog dialog = new JDialog();
                dialog.setTitle( "Mixer" );
                MixerPanel mixerPanel = new MixerPanel( (Mixer.Info)getCellEditorValue() );
                dialog.getContentPane().add( mixerPanel );
                dialog.pack();
                dialog.setVisible( true );
                fireEditingStopped();
        }
    }
}
```

Le view et les model illustrés par une JTable

```
class MixerRenderer extends JLabel implements TableCellRenderer {

    public MixerRenderer() {
            setOpaque(true);
    }

    public Component getTableCellRendererComponent
( JTable table, Object color, boolean isSelected, boolean hasFocus, int row, int column){
        setBackground( Color.ORANGE );
        setToolTipText( "Afficher les caractéristiques du mixer" );
        return this;
    }
}
```

Le view et les model illustrés par une JTable



Le view et les model illustrés par une JTable

```
public class MixerPanel extends JPanel{
    public MixerPanel( Mixer.Info mixerInfo ){
        setLayout( new BoxLayout(this, BoxLayout.PAGE_AXIS) );
        setBorder( BorderFactory.createEmptyBorder(5,5,5,5) );

        add( Box.createRigidArea(new Dimension(0, 5)) );
        add( new JLabel(mixerInfo.toString()) );

        add( Box.createRigidArea(new Dimension(0, 5)) );
        SourceDataLinePanel sourceDataLinePanel = new SourceDataLinePanel( mixerInfo );
        add( sourceDataLinePanel );

        add( Box.createRigidArea(new Dimension(0, 5)) );
        TargetDataLinePanel targetDataLinePanel = new TargetDataLinePanel( mixerInfo );
        add( targetDataLinePanel );

        add( Box.createRigidArea(new Dimension(0, 5)) );
        PortPanel portPanel = new PortPanel( mixerInfo );
        add( portPanel );

        add( Box.createGlue() );
    }
}
```

Le

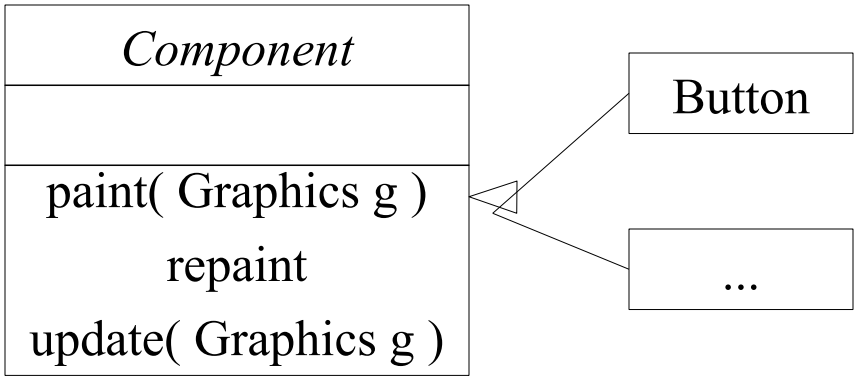
Graphisme

avec les

SWING

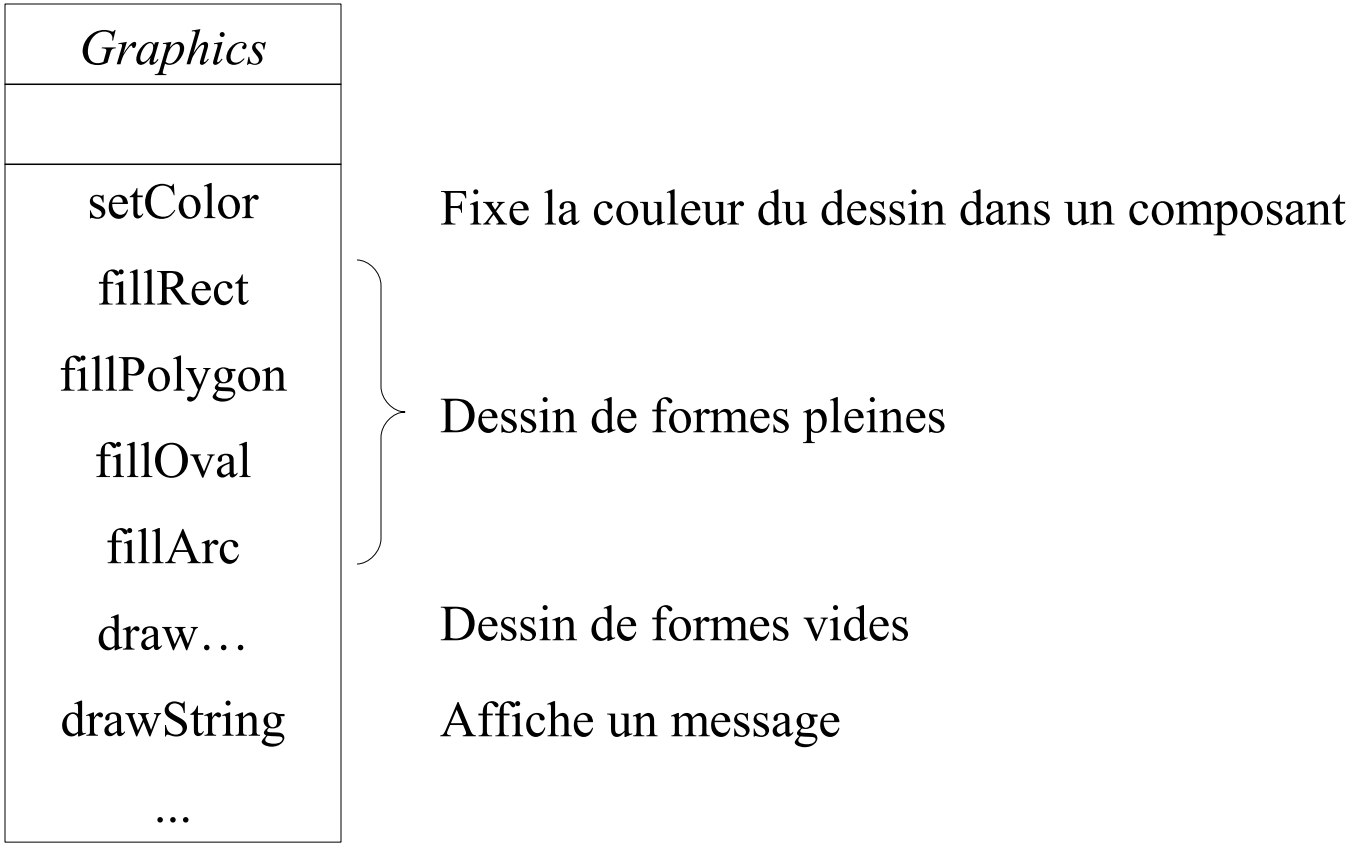
Le dessin des composants

- la méthode paint permet de peindre un composant, elle est appelée la première fois qu'un composant s'affiche et quand c'est nécessaire ;
- si une application veut redessiner un composant, il peut appeler la méthode repaint ;
- repaint est chargé de planifier le dessin du composant qui sera redessiné par appel de la méthode update ;
- update efface le composant puis appelle la méthode paint.

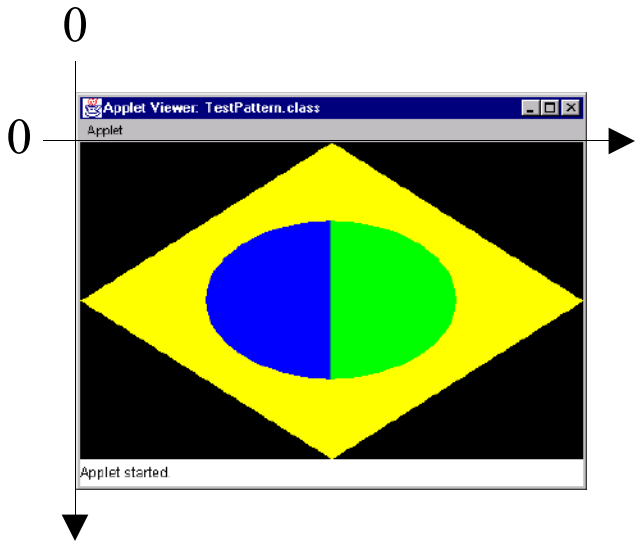


Un objet de type Graphics définit le contexte d'affichage d'un composant, et permet de réaliser des dessins.

Le dessin des composants



Exemple




```
public class Dessin extends JApplet{
    public void paint( Graphics graphic ){
        int appLargeur = getSize().width ;
        int appHauteur = getSize().height ;
        int largeur = appLargeur/2 ;
        int hauteur = appHauteur/2 ;
        int x = (appLargeur - largeur)/2 ;
        int y = (appHauteur - hauteur)/2 ;
        int [] polyX = { 0, appLargeur/2, appLargeur, appLargeur/2 } ;
        int [] polyY = { appHauteur/2, 0, appHauteur/2,  appHauteur } ;
        Polygon poly = new Polygon( polyX, polyY, 4 ) ;
        graphic.setColor( Color.black ) ;
        graphic.fillRect( 0, 0, getSize().width, getSize().height ) ;
        graphic.setColor( Color.yellow ) ;
        graphic.fillPolygon( poly ) ;
        graphic.setColor( Color.green ) ;
        graphic.fillOval( x, y, largeur, hauteur ) ;
        graphic.setColor( Color.blue ) ;
        graphic.fillArc( x, y, largeur, hauteur, 90, 180 ) ;
    }
}
```

Coordonnées des angles du polygone

Les polices de caractères

Graphics
setFont drawString ...

Fixe la police d'un contexte graphique

Affiche un texte

Font
PLAIN BOLD ITALIC
...

Variables publiques

```
Font font = new Font( "TimesRoman", Font.PLAIN, 12 ) ;  // défini une police
```

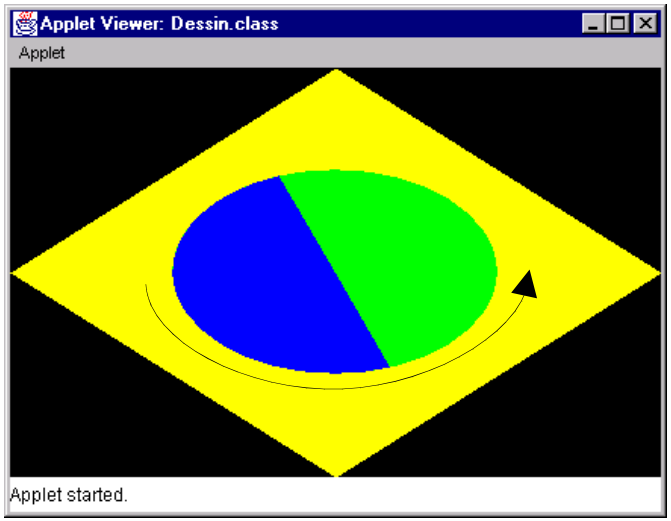
Dialog ;
Helvetica ;
TimesRoman ;
Courier ;
Symbol.

Taille en nombre de points de la police.

Le problème du rafraîchissement des images et des dessins

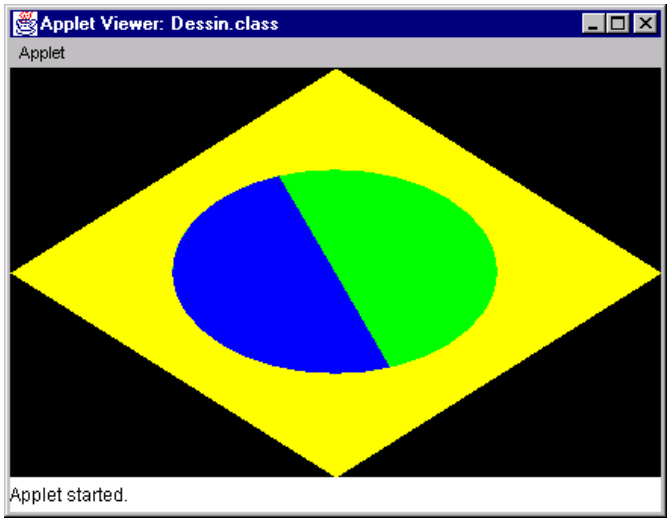
- Cas d'un dessin tournant sur lui même à chaque clic sur la souris (suite du transparent 4) :

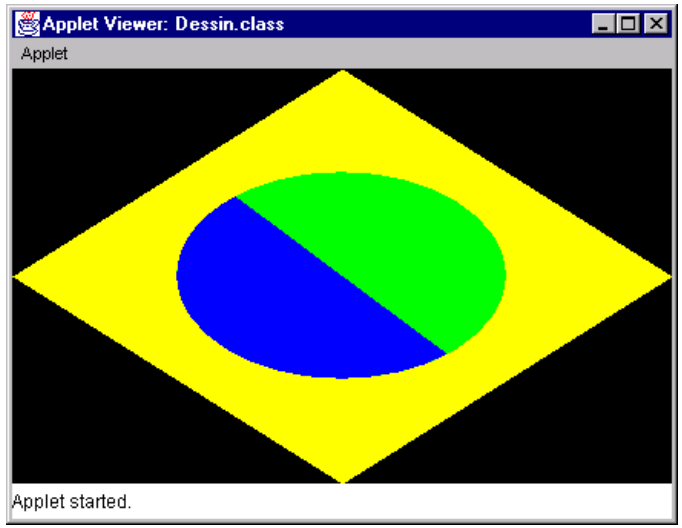
```
public class Dessin extends JApplet implements MouseListener{
    int theta = 90 ;
    int largeur, hauteur, x, y ;

    public void init(){
        addMouseListener( this ) ;
    }
    public void mouseClicked( MouseEvent e ){
        theta = (theta+10) % 360 ;
        repaint() ;
    }
    public void mouseEntered(MouseEvent e){} ;
    // ...

    public void paint( Graphics graphic ){
        // …        graphic.fillArc( x, y, largeur, hauteur, theta, 180 ) ;
    }
}
```



Le problème du rafraîchissement des images et des dessins

- Le passage d'une image à une autre n'est pas fluide, il y a des tremblements !





Première solution

- Ne pas rafraîchir toute l'image mais définir un rectangle autour de la partie de l'image qui change ;
- Définir le rectangle de rafraîchissement dans la méthode update qui est appelée par repaint.

```
public class Dessin extends JApplet implements MouseListener{
   // ...
   public void mouseClicked( MouseEvent e ){
            // ...
            repaint() ;
   }
   public void paint( Graphics graphic ){          // repeint l'image précisée dans le
            // …                                   // le contexte graphique g
   }
   public void update( Graphics g ){
            g.clipRect( x, y, largeur, hauteur ) ; // définition du rectangle à rafraîchir
            paint( g ) ;                           // repeint uniquement le rectangle
   }
}
```

Deuxième solution : le double buffering

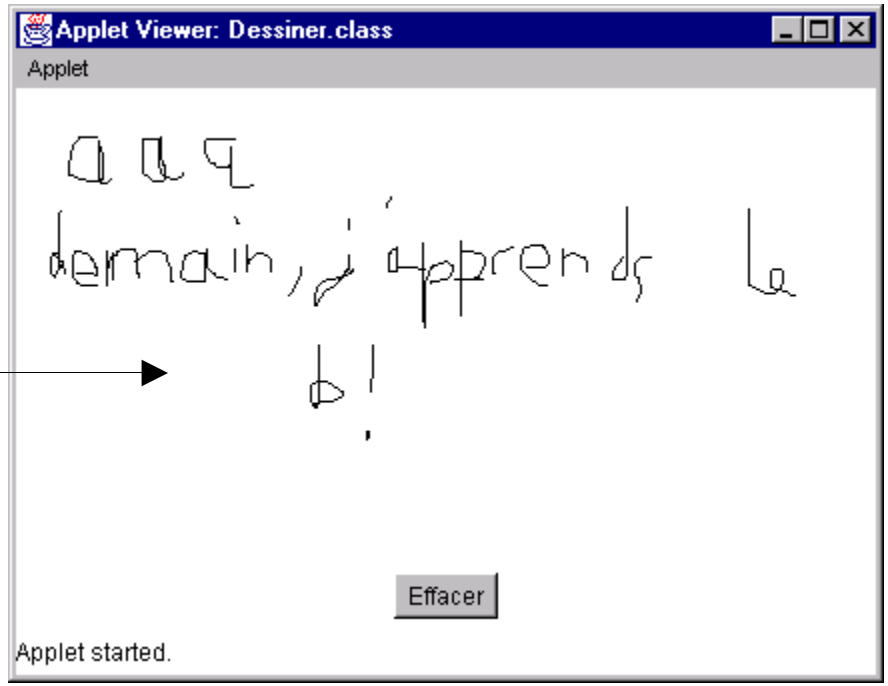
- Ce qui prend du temps, ce n'est pas d'afficher mais de dessiner : dessiner dans une zone tampon (ce qui peut prendre du temps) puis n'afficher que le résultat (ce qui est toujours rapide).

```
public class Dessin extends java.applet.Applet implements MouseListener{
    // ...
    public void update( Graphics g ){
            // créé une image à la taille de l'applet
            Image tampon = createImage( getSize().width, getSize().height ) ;
            // récupère son contexte graphique
            Graphics graphic = tampon.getGraphics() ;
            // dessine l'image avec le contexte graphique
            paint( graphic ) ;
            // quand l'image est prête … on l'affiche
            g.drawImage( tampon, 0, 0, this ) ;
            // le contexte graphique ne peut être utilisé qu'une seule fois
            graphic.dispose() ;
    }
}
```

Application du double buffering au dessin avec la souris 1/4

- Comment reproduire fidèlement les efforts d'un utilisateur faisant n'importe quoi avec la souris ?

Un objet de type Canvas



- Dissocier la gestion de la souris de l'affichage.

Dessin avec la souris : définition d'une zone de dessin 2/4

```
public class Dessiner extends JFrame implements ActionListener{
      ZoneDeDessin zoneDeDessin ;                  // la zone de dessin
      public Dessiner {
            setLayout( new BorderLayout() ) ;      // pour positionner sur les bords
            add( zoneDeDessin = new ZoneDeDessin(), BorderLayout.CENTER );
            JPanel p = new JPanel() ;              // zone basse de l'applet avec un bouton
            JButton boutonClear = new JButton( "Effacer") ;
            boutonClear.addActionListener( this ) ;// récepteur d'événements
            p.add( boutonClear ) ;                                  // ajout du bouton ...
            add( "South", p ) ;                                     // à la zone basse de l'applet
            setVisible( true );
      }
      public void actionPerformed( ActionEvent e ){          // clic sur le bouton
            if( e.getActionCommand().equals( " Effacer " ) )
               zoneDeDessin.clear() ;                       // efface le dessin
      }
}
class ZoneDeDessin extends JPanel {
      // ...
}
```

Dessin avec la souris : gérer l'affichage 3/4

```
class ZoneDeDessin extends JPanel {
        Image drawImg ;
        Graphics drawGr ;
        public void update( Graphics g ) {
                paint(g) ;                          // update ne fait que peindre
        }
        public void paint( Graphics g ) {                   // création d'un buffer, récupère son
                if ( drawImg == null ) {                    // contexte, et prépare un crayon noir
                        drawImg = createImage( getSize().width, getSize().height ) ;
                        drawGr = drawImg.getGraphics() ;
                        drawGr.setColor( Color.black ) ;
                }
                g.drawImage(drawImg, 0, 0, null) ;      // affiche le buffer
        }
        public void clear() {                               // efface la zone de dessin
                drawImg.getGraphics().clearRect(0, 0, getSize().width, getSize().height) ;
                repaint() ;
        }
}
```

Dessin avec la souris : gérer la souris 4/4

```
class ZoneDeDessin extends JPanel implements MouseMotionListener, MouseListener{
      int xpos, ypos, oxpos, oypos ;                  // coordonnées d'un trait

      ZoneDeDessin() {
            setBackground( Color.white ) ;
            addMouseMotionListener( this ) ;      // pour l'événement mouseDragged
            addMouseListener( this ) ;            // pour l'événement mousePressed
      }
      public void mouseDragged( MouseEvent e ){
            xpos=e.getX(); ypos=e.getY() ;        // mémorise la fin du trait
            if ( drawGr != null )                 // trace si nécessaire dans le buffer
                  drawGr.drawLine( oxpos, oypos, oxpos=xpos, oypos=ypos ) ;
            repaint() ;                           // affiche le dessin avec un nouveau trait
      }
      public void mousePressed(MouseEvent e){           // mémorise le début du trait
          oxpos=e.getX() ; oypos=e.getY() ;
      }
}
```

Les images

Les images sont traités de manière asynchrone : les méthodes qui utilisent des objets Image retournent immédiatement, mais continuent à s'exécuter en tâche de fond :

```
public class MonApplet extends java.applet.Applet{
    // ...
    Image img = getImage( "http://monserveur/image.jpg" ) ;
}
```

La méthode getImage retourne immédiatement, le téléchargement de l'image n'a lieu que si on en a besoin :

```
public void paint( Graphics g ){
    g.drawImage( img, 0, 0, this ) ;
}
```

- Avantages : l'application n'est pas bloquée ;

- Inconvénient : comment savoir quand l'image est disponible pour l'affichage ?

Le MediaTracker

- Les limites des ImageObserver est qu'il faut 1 spectateur par image à charger !
- Comment suivre le chargement de plusieurs images pour une seule application ?

MediaTracker

ABORTED

ERRORED

COMPLETE

LOADING

→ status des images.

MediaTracker(Component) → Création d'un MadiaTracket pour les images d'un composant

addImage → Ajout d'images avec un identificateur.

checkAll - checkID → Vérifie si les images sont entièrement chargées

statusAll - statusID → Vérifie le status des images.

waitForAll - waitForID → Commence et attend la fin du chargement d'images.

...

Exemple d'utilisation du MediaTracker 1/4

Une applet qui charge plusieurs images pour une animation, attend que toutes les images soient chargées avant d'afficher l'animation.

```
public class Animation extends JApplet implements Runnable {
    MediaTracker tracker ;
    Image fond ;                                    // image de fond
    Image anim[] = new Image[5] ;                   // les images animées

    public void init(){
       tracker = new MediaTracker( this ) ;         // créé un MediaTracker pour l'applet
       fond = getImage( getCodeBase(), "fond.jpg" ) ;   // l'image de fond
       tracker.addImage( fond, 0 );                 // id = 0 pour l'image de fond
       for (int i = 0; i < 5; i++){                 // Les images de l'animation
          anim[i] = getImage( getCodeBase(), "anim"+i+".gif" );
          tracker.addImage( anim[i], 1 );           // id = 1 pour les animations
       }
    }
    // ...
}
```

Affiche l'animation quand elle est prête 2/4

```
public class Animation extends JApplet implements Runnable {
      int index ;
      // ...
      public void update( Graphics g ) {
         paint( g ) ;              // le fond occupe toute l'applet, pas la peine de l'effacer
      }
      public void paint( Graphics g ) {              // si erreur de chargement
         if ( (tracker.statusAll(false) & MediaTracker.ERRORED) != 0 ) {
               g.setColor( Color.red ) ;              // affiche un rectangle rouge
               g.fillRect( 0, 0, size().width, size().height ) ;
               return ;
         }
         g.drawImage( fond, 0, 0, this ) ;    // affiche le fond quand il arrive
         // si toutes les animations sont pretes ...
         if( tracker.statusID(1, false) == MediaTracker.COMPLETE )
               g.drawImage( anim[index], 10, 10, this ) ; // affiche les animations
      }
}
```

Créer un thread pour le chargement et l'animation 3/4

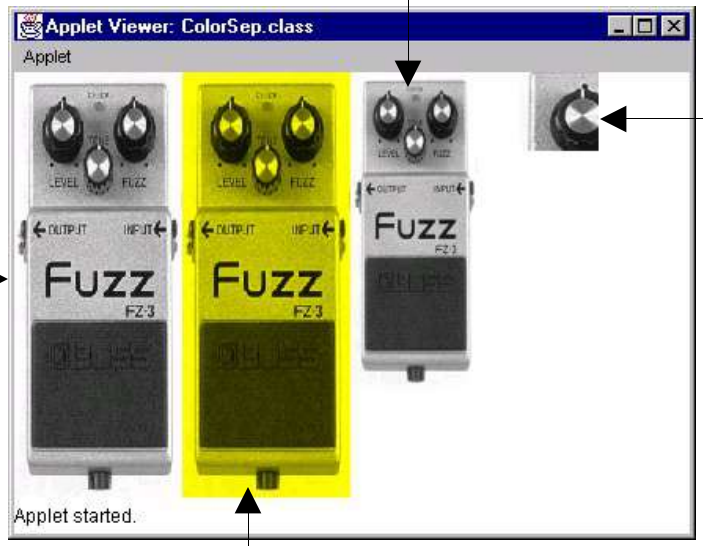
```
public class Animation extends JApplet implements Runnable {
    Thread animation ;
    // ...
    public void start() {
        animation = new Thread( this ) ;
        animation.start() ;                          // lance l'animation
    }
    public void stop() {
        animation.stop() ;                           // stoppe l'animation
        animation = null ;
    }
    public void run() {
        // ...
    }
}
```

Créer un thread pour le chargement et l'animation 4/4

```
public class Animation extends JApplet implements Runnable {
    // ...
    public void run() {
        try {
            tracker.waitForID( 0 ) ;        // attend le chargement de l'image de fond
            tracker.waitForID( 1 ) ;        // attend le chargement des images de l'animation
        } catch( InterruptedException e ){ return ; }
        Thread me = Thread.currentThread() ;
        while( animation == me ) {      // tant que le thread s'exécute
            try{
                Thread.sleep( 100 ) ;   // endort le thread
            }catch( InterruptedException e ){ break ; }
            synchronized( this ){
                index++ ;                   // passe à l'image suivante
                if( index >= anim.length ) index = 0 ;
            }
            repaint() ;                     // appel à paint
        }
    }
}
```

Le filtrage d'images

Image changée d'échelle

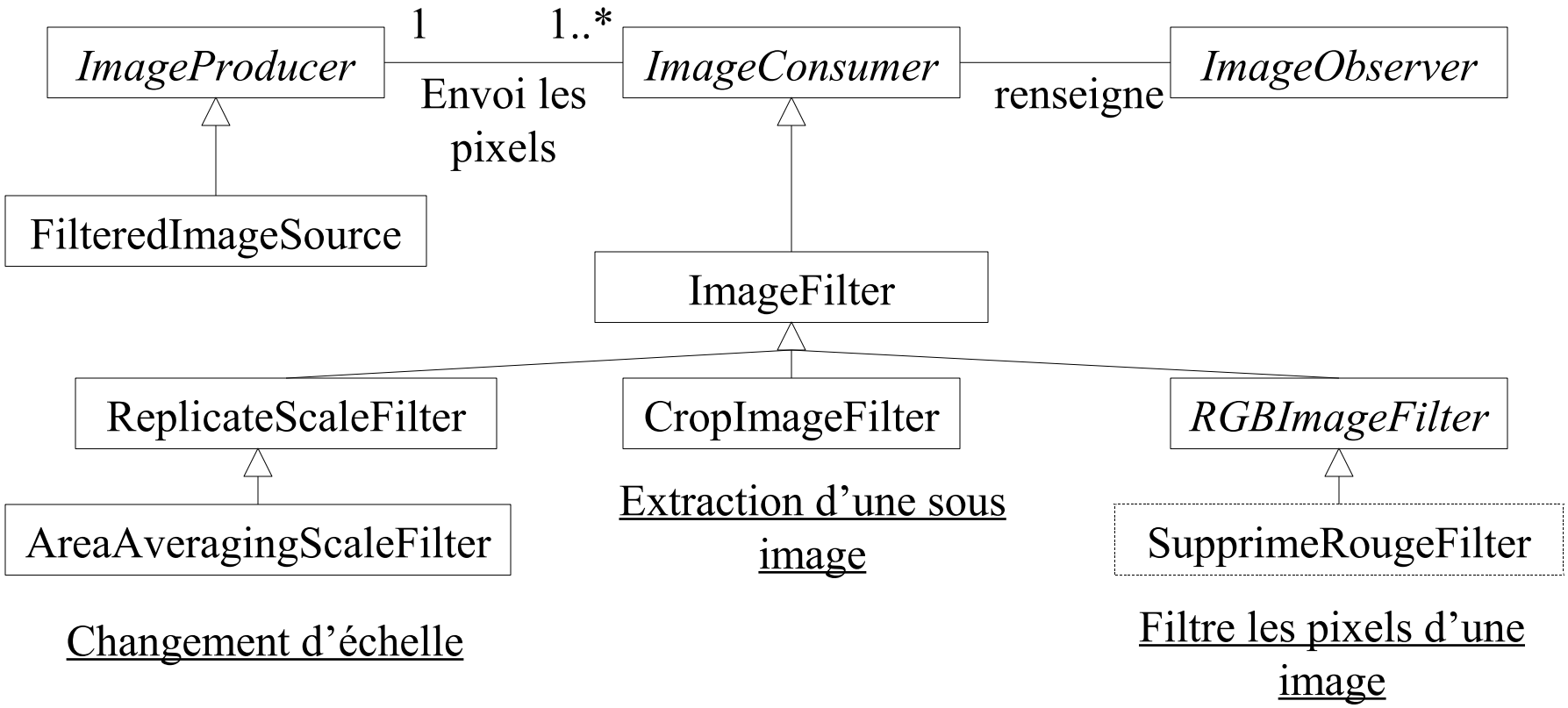


Partie de l'image

Image originale

Image dont on a supprimé la composante rouge

Le filtrage d'images




```
Image original = getImage( getCodeBase(), "fz3.jpg" ) ;
Image ImageFiltree  = createImage(
      new FilteredImageSource( original.getSource(), new SupprimeRougeFilter())
) ;
```

producteur producteur consommateur

Filtrer des images

```
public class Filtage extends JApplet {

    Image original, vert, echelle, partie ;

    public void init(){
        // charge l'image originale
        original = getImage( getCodeBase(), "fz3.jpg" ) ;
        // créé une image dont on supprimé la composante rouge
        vert = createImage( new FilteredImageSource( original.getSource(),
                new SupprimeRougeFilter())) ;
        // créé une image avec changement d'échelle
        echelle = createImage( new FilteredImageSource( original.getSource(),
                new AreaAveragingScaleFilter( 80, 200 ))) ;
        // créé une sous image
        partie = createImage( new FilteredImageSource( original.getSource(),
                new CropImageFilter( 10, 10, 50, 50 ))) ;
        setLayout( null ) ;
        setSize( 430, 270 ) ;
    }
}
```

Afficher les images filtrées

```
public class Filtage extends JApplet {

    Image original, vert, echelle, partie ;

    public void init(){
        // ...
    }
    public void paint( Graphics g ) {
        int largeur = getSize().width ;
        int hauteur = getSize().height ;
        g.drawImage( original, 0, 0, largeur/4, hauteur, this ) ;
        g.drawImage( vert, largeur/4, 0, largeur/4, hauteur, this ) ;
        g.drawImage( echelle, 2*largeur/4, 0, 80, 200, this ) ;
        g.drawImage( partie, 3*largeur/4, 0, 50, 50, this ) ;
    }
}
```

Supprimer la composante rouge d'une image

```
public class Filtage extends java.applet.Applet{
        // ...
        vert = createImage( new FilteredImageSource( original.getSource(),
                                        new SupprimeRougeFilter()));
}

class SupprimeRougeFilter extends RGBImageFilter{
        public SupprimeRougeFilter(){
                canFilterIndexColorModel = true;      // le filtrage ne dépend pas
        }                                             // de la position des pixels

        public int filterRGB( int x, int y, int rgb){
                return ((rgb & 0xff00ff00) | ((rgb & 0xff) << 16));
        }
}
```