**Commandes en Java - IHM**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Symbole | Signification | Code | Utilisation | | | | |
|  |  |  |  | | | | |
| Fenêtre |  |  |  | | |  | |
| JFrame | Classe permettant d'instancier une nouvelle fenêtre | JFrame maFenetre = new JFrame(); | javax.swing | | |  | |
| .setVisible( ) | permet d'afficher la fenêtre à l'écran | maFenetre.setVisible(true); |  | | | ATTENTION : cette méthode ne doit être appelée que lorsque tous les paramètres d'affichages de la fenêtre ont étés définis | |
| .setTitle( ) | donne un titre à la fenêtre | maFenetre.setTitle("Titre de ma fenêtre"); |  | | |  | |
| .setDefaultCloseOperation( ) | défini le comportement lorsque la fenêtre est fermée  arret complet du programme  fermeture de la fenetre uniquement | maFenetre.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  JFrame.EXIT\_ON\_CLOSE  JFrame.DISPOSE\_ON\_CLOSE |  | | |  | |
| .setSize( , ) | défini les dimensions (largeur, hauteur) en pixel de la fenêtre | maFenetre.setSize(nb, nb); |  | | |  | |
| .setLocationRelativeTo( ) | défini la position relative de la fenêtre  centrée | maFenetre.setLocationRelativeTo(null); |  | | |  | |
| .setLocation( , ) | défini la postition en pixel du coin sup gauche de la fenêtre par rapport au coin sup gauche de l'écran | maFenetre.setLocation(nb, nb); |  | | |  | |
| .setResizable( ) | empêche le redimensionnement de la fenêtre si false (true par défaut) | maFenetre.setResizable(false); |  | | |  | |
| .setAlwaysOnTop( ) | garde la fenêtre toujours au premier plan si true (false par défaut) | maFenetre.setAlwaysOnTop(true); |  | | |  | |
| .setUndecorated( ) | retire le contour et les boutons de contrôles si true (false par défaut) | maFenetre.setUndecorated(true); |  | | |  | |
|  |  |  |  | | |  | |
| Panel |  |  | javax.swing | | |  | |
| JPanel | objet permettant l'instanciation d'un nouveau panel (= panneau) qui sert à construire notre page avant de l'insérer dans la fenêtre | JPanel monPanel = new JPanel(); | *Portion de code intéressant :*  //Icone  icon = new JLabel(new ImageIcon("imgs/icone.jpg"));  JPanel panIcon = new JPanel();  panIcon.setBackground(Color.WHITE);  panIcon.setLayout(new BorderLayout());  panIcon.add(icon);  //Le nom  JPanel panNom = new JPanel();  panNom.setBackground(Color.white);  panNom.setPreferredSize(new Dimension(220, 60));  nom = new JTextField();  nom.setPreferredSize(new Dimension(100, 25));  panNom.setBorder(BorderFactory.createTitledBorder("Nom du personnage"));  nomLabel = new JLabel("Saisir un nom : ");  panNom.add(nomLabel);  panNom.add(nom); | | |  | |
| .setContentPane( ) | méthode de l'objet JFrame (= fenêtre) permettant d'insérer le contenu du panel dans la fenêtre | maFenetre.setContentPane(monPanel); |  | | | ATTENTION : ici aussi cette méthode ne doit être appliquée qu'une fois toutes les modifications du Panel réalisées | |
|  |  |  |  | | |  | |
| Mise en page |  |  | Java.awt | | |  | |
| Layout managers | mise en page des éléments dans la fenêtre |  |  | | |  | |
| .setLayout( ) | méthode à appliquer aux objets JFrame ? et JPanel pour controler la mise en page  elle prend en paramètre un objet de type Layout | FlowLayout monLayout = new FlowLayout();  maFenetre.setLayout(monLayout); ???  monPanel.setLayout(monLayout); |  | | |  | |
| FlowLayout | centre tous les éléments sur une ligne et passe à la ligne suivante quand plus de place | monPanel.setLayout(new FlowLayout()); |  | | | Remarque : par défaut pour les éléments JFrame | |
| BorderLayout | place les objets selon 4 emplacements cardinaux + centre | monPanel.setLayout(new BorderLayout());  monPanel.add(monBouton, BorderLayout.SOUTH); |  | | | Remarque : par défaut pour les éléments JPanel | |
| GridLayout | découpe la page en un tableau dont on choisit le nb de ligne et colonne | monPanel.setLayout(new GridLayout(nbLi, nbCol));  monPanel.add(monBouton); |  | | |  | |
| .setHgap( )  .setVgap( ) | définit l'espace (en pxl) entre   * les colonnes (horizontal) * les lignes (vertical) | monGridLayout.setHgap(nb); |  | | |  | |
| BoxLayout | range les éléments en une seule :  ligne  colonne | monPanel.setLayout(new BoxLayout(monPanel, BoxLayout.LINE\_AXIS));  monPanel.setLayout(new BoxLayout(monPanel, BoxLayout.PAGE\_AXIS)); |  | | |  | |
| Box | permet aussi de créer des "Box" à utiliser comme des JPanel pour les remplir et les insérer dans le JPanel ou directement dans la fenêtre (ContentPane) | Box b1 = Box.createHorizontalBox();  b1.add(monBouton);  Box b2 = Box.createVerticalBox();  monPanel.add(b1); |  | | |  | |
| CardLayout | créer plusieurs panneaux superposés dont un seul est visible et pouvoir passer de l'un à l'autre facilement grâce à leur nom | CardLayout maCL = new CardLayout();  monConteneur.setLayout(maCL);  monConteneur.add(monPanel1, "Nom panel1"); |  | | |  | |
| .next( ) | affiche dans la page le panel suivant dans la liste | maCL.next(monConteneur); |  | | |  | |
| .show( ) | affiche dans la page le panel correspondant au nom donné | maCL.show(monConteneur, "Nom panelX"); |  | | |  | |
| GridBagLayout | permet de définir avec précision l'emplacement et la taille relative des éléments dans notre fenêtre découpée selon une grille infinie et basée sur le premier élément inséré | monPanel.setPreferredSize(new Dimension(nb, nb);  monPanel.setLayout(new GridBagLayout()); |  | | | Remarque : on doit définir une dimension initiale de du panel pour servir de référence pour les dimensions relatives des éléments | |
| GridBag  Constraints | Classe permettant de définir les propriétés d'affichage d'un élément avant de l'insérer dans le GrigBagLayout | GridBagConstraints gbc = new GridBagConstraints();  gbc.gridx = 0; //1er élément de la ligne  gbc.gridy = 1; //2e colonne  gbc.gridwidth = 1; //largeur d'une case  gbc.gridheight = 2; //hauteur de 2 case  gbc.fill = GridBagConstraints.VERTICAL; //la cellule à fusionnée 2 cases verticales  monPanel.add(panel5, gbc); |  | | | Pour plus de détail sur ce Layout : PDF cours p. 289 et 291 | |
| Onglets |  |  | javax.swing | | |  | |
| JTabbedPane | objet accueillant plusieurs JPanel afin de créer des onglets dans la fenêtre JFrame | JPanel panA = new JPanel(), panB = new JPanel();  mesOnglets = new JTabbedPane();  mesOnglets.add("Onglet A", panA);  mesOnglets.add("Onglet B", panB);  this.getContentPane().add(mesOnglets, BorderLayout.CENTER); | *En précisant la position de la barre d'onglets dans le constructeur :*  mesOnglets = new JTabbedPane(JTabbedPane.LEFT);  RIGHT  BOTTOM | | | Rq: par défaut, la position de la barre d'onglets est en haut de la fenêtre. Elle peut être modifié en le précisant dans le constructeur ou avec la méthode .setTabbedPlacement( ) qui prend les même paramètres. | |
| .setIconAt( , ) | ajout d'une image devant le titre de l'onglet  param : (index de l'onglet, ImageIcon) | mesOnglets.setIconAt(0, new ImageIcon("adresse")); |  | | | Rq: l'index des onglets commence à 0, comme pour les tableaux | |
| .setTabPlacement( ) | modifie la position de la barre d'onglets | mesOnglets.setTabPlacement(JTabbedPane.LEFT);  TOP, RIGHT, BOTTOM |  | | |  | |
| .remove( ) | retire l'onglet de l'index indiqué | mesOnglets.remove(1); |  | | |  | |
| .getSelectedIndex() | renvoi l'index de l'onglet actuellement actif dans la fenêtre | int indexOngletActif = mesOnglets.getSelectedIndex(); |  | | |  | |
| Fenêtre interne |  |  |  | | |  | |
| JDesktopPane  JInternalFrame | conteneur permettant d'insérer des fenêtre mobiles dans la page | class MaFenetrePrincipale extends JFrame {  desktop = new JDesktopPane();  miniFenetre = new JIntenalFrame();  desktop.add(miniFenetre);  this.getContentPane().add(desktop, Layout…);  } | classe interne automatisant la création de mini-fenetres :  class MaFenetrePrincipale extends JFrame {  private static nbFenetre = 0;  private static int xy = 10;  class MiniFenetre extends JInternalFrame {  public MiniFenetre() {  xy += 4;  this.setTitle("MiniFenetre" + (++nbFenetre));  this.setSize(150,80);  this.setLocation(xy, xy);  this.setResizable(true);  this.setClosable(true);  this.setVisible(true);  }  }  } | | |  | |
| JWindow | fenêtre sans contour et les controles classique (fermer, réduire, …) souvent utilisé pour les splash screens (écran de chargement, …) |  | class Window extends JWindow {  public Window() {  this.setSize(220, 170);  this.setLocationRelativeTo(null);  Jpanel pan = new JPanel();  JLabel img = new JLabel(new ImageIcon("adresse"));  img.setVerticalAlignment(JLabel.CENTER);  img.setHorizontalAlignment(JLabel.CENTER);  pan.setBorder(BorderFactory.createLineBorder(Color.BLUE);  pan.add(img);  this.getContentPane().add(pan);  this.setVisible(true);  }  } | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Boutons |  |  | javax.swing | | |  | |
| JButton | classe permettant l'instanciation d'un bouton | JButton monBouton = new JButton("Etiquette"); |  | | |  | |
| .setText( ) | modifie l'étiquette du bouton | monBouton.setText("Nouvelle étiquette"); |  | | |  | |
| .setEnabled( ) | rend le bouton cliquable si true (par défaut) ou non si false | monBouton.setEnabled(false); |  | | |  | |
|  |  |  |  | | |  | |
| Texte |  |  | javax.swing | | |  | |
| JLabel | classe permettant l'instanciation d'un élément de type texte | JLabel monLabel= new JLabel("Mon Texte"); |  | | |  | |
| .setText( ) | modifie le texte du label | monLabel.setText("Mon nouveau texte"); |  | | |  | |
| .setFont( ) | définie la police du texte en prenant en paramètre un objet de type Font | Font maPolice = new Font("nomPolice", Font.BOLD, 30);  monLabel.setFont(maPolice); |  | | |  | |
| .setHorizontalAlignment( )  .setVerticalAlignment( ) | définit la position du label dans son conteneur | monLabel.setHorizontalAlignment(JLabel.CENTER); |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Décoration des éléments | méthodes communes à quasiment tous les composant et permettant leur mise en page, mise en forme, décoration, … | si non précisé = valable pour monConteneur et monElement |  | | |  | |
| .setPreferredSize( ) | défini la taille par défaut à l'affichage | .setPreferedSize(new Dimension(Xpx, Ypx); |  | | |  | |
| .setForeground( ) | modifier la couleur de la police | .setForeground(Color.BLUE); |  | | |  | |
| .setVerticalAlignment( )  .setHorizontalAlignment( ) | contrôle l'alignement d'un objet dans son conteneur | monElement.setVerticalAlignment(JLabel.CENTER);  monElement.setHorizontalAlignment(JLabel.LEFT); |  | | |  | |
| .setBorder( ) | bordure des éléments | .setBorder(BorderFactory.createTitledBorder("Titre de la bordure"));  .setBorder(BorderFactory.createLineBorder(Color.BLUE)); | *Autres types de bordures :*  .createBevelBorder(BevelBorder.LOWERED, Color.black, Color.red),  .createEtchedBorder(Color.BLUE, Color.GRAY),  .createLineBorder(Color.green),  .createMatteBorder(5, 2, 5, 2, Color.MAGENTA),  .createRaisedBevelBorder(),  .createTitledBorder("Titre"),  .createCompoundBorder(borderA, borderB) | | | Rq: ici Compound = Bevel + Matte | |
|  |  |  |  | | |  | |
| .setToolTipText( ) | affiche une bulle d'information au survol prolongé de la souris sur l'élément | .setToolTipText("Info complémentaires sur l'élément"); |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| JScrollPane | ajoute des ascenseurs au panel lorsque le contenu dépasse la taille de la page | monPanel = new JPanel();  JScrollPane scroll = new JScrollPane(monPaneau);  this.getContentPane().add(scroll, BorderLayout.CENTER); |  | | | ATTENTION : l'objet JScrollPane encapsule le Panel sur lequel il est appliqué. Pour afficher cet élément, il faut afficher le Scroll. | |
| .setHorizontalScrollBarPolicy( )  .setVerticalScrollBarPolicy( ) | défini le comportement du scroll horizontal du composant  3 paramètres possibles :  scroll apparait quand c'est nécessaire  n'apparait jamais  reste tjs affiché | monScroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS);  JScrollPane.HORIZONTAL\_SCROLLBAR\_AS\_NEEDED  JScrollPane.HORIZONTAL\_SCROLLBAR\_NEVER  JScrollPane.HORIZONTAL\_SCROLLBAR\_ALWAYS | (par défaut) | | | Rq: idem pour Vertical. | |
|  |  |  |  | | |  | |
| Barre de séparation mobile |  |  |  | | |  | |
| JSplitPane | défini une barre de séparation mobile entre deux éléments de type JComponent (JPanel, JButton, JComboBox, JRadioButton, …) | JPanel pan1 = new JPanel(), pan2 = new JPanel();  split = new JSplitPane(JSplitPane.HORIZONTAL\_SPLIT, pan1, pan2);  this.getContentPane().add(split, BorderLayout.CENTER); |  | | | ATTENTION : une fois le split appliqué sur les Panel, ces derniers appartiennent au composant split (le Split encapsule les autres). Pour les afficher tous les trois, il suffit donc d'insérer le split dans la page. | |
| .setOneTouchExpandable( ) | affiche les boutons sur la barre pour pouvoir faire complètement disparaitre l'un ou l'autre des conteneurs | split.setOneTouchExpandable(true); |  | | |  | |
| .setDividerSize( ) | défini la taille de la barre (en pixel) | split.setDividerSize(5); |  | | |  | |
| .setDividerLocation( ) | défini la position de la barre lors de l'affichage de la fenêtre (en pixel par rapport à la gauche ou au haut) | split.setDividerLocation(10); |  | | |  | |
|  |  |  |  | | |  | |
| JSlider | barre permettant de suivre en direct la valeur et de sélectionner un nb dans un intervalle donné | slider = new JSlider();  slider.setMaximum(100);  slider.setMinimum(0);  slider.setValue(30);  slider.setPaintTicks(true);  slider.setPaintLabels(true);  slider.setMinorTickSpacing(10);  slider.setMajorTickSpacing(20);  slider.addChangeListener(new ChangeListener() {  public void stateChanged(ChangeEvent e) {  label.setText("Valeur actuelle : " + ((JSlider)e.getSource()).getValue());  }  });  monPanel.add(slider, BorderLayout.CENTER); |  | | |  | |
|  |  |  |  | | |  | |
| JProgressBar | barre de progression / de chargement  progression en % visible  valeur entre 0 et 500 liée à la progression du traitement | class ProgressFenetre extends JFrame {  public ProgressFenetre() {  bar = new JProgressBar();  bar.setMaximum(500);  bar.setMinimum(0);  bar.setStringPainted(true);    this.getContentPane().add(bar, BorderLayout.CENTER);  t = new Thread(new Traitement());  t.start();  this.setVisible(true);  class Traitement implements Runnable {  public void run() {  instruction(s);  bar.setValue(valeurDeProgression);  }  }  }  } |  | | | Rq: la barre de progression ne peut suivre que les calculs d'un autre Thread, sinon on n'observera pas la progression (elle passera directement de 0 à 100%) | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Graphics | les objets de type Graphics sont générés automatiquement par le programme lors de l'affichage d'un composant dans la fenêtre mais aussi à chaque redimensionnement, …  Ces objets vont alors être utilisés pour modifier et personnaliser nos éléments (panneau, bouton, …) | | java.awt | | |  | |
| paintComponent(Graphics g) | méthode qui sera lancé à chaque fois qu'un objet Graphics sera généré pour l'objet auquel elle appartient | public class monElement extends JXXX {  public void paintComponent(Graphics g) {  instruction(s) pour modifier l'apparence;  }  } |  | | |  | |
| .repaint() | méthode à appeler sur un objet contenant un élément Graphics pour lancer la méthode paintComponent( ) | monElement.repaint(); |  | | |  | |
|  | ATTENTION : toutes les méthodes suivantes sont à utiliser dans la méthode paintComponent( ) { |  |  | | |  | |
| .getWidth() | donne la largeur de l'élément | this.getWidth(); |  | | |  | |
| .getHeight() | donne la hauteur de l'élément |  |  | | |  | |
| .setColor( ) | modifie la couleur de l'élément tracé en précisant une couleur reconnue par le programme | g.setColor(Color.maCouleur);  *g.setColor(Color.RED);* |  | | |  | |
| .drawOval( , , , ) | dessine un ovale vide avec (abscisse, ordonnée, largeur, hauteur) en pixel | g.drawOval(x, y, w, h);  *g.drawOval(10, 10, 50, 75);* |  | | |  | |
| .fillOval( , , , ) | dessine un ovale plein | g.fillOval(x, y, w, h |  | | |  | |
| .drawRect( , , , ) | dessine un rectangle vide | g.drawRect(x, y, w, h) |  | | |  | |
| .fillRect( , , , ) | dessine un rectangle plein | g.fillRect(x, y, w, h) |  | | |  | |
| .drawRoundRect( , , , , , ) | dessine un rectangle vides avec bords arrondis, les 2 derniers paramètres sont (arcWidth, arcHeight) | g.drawRoundRect(x, y, w, h, aw, ah) |  | | |  | |
| .fillRoundRect(, , , , , ) | dessine un rectangle plein avec bords arrondis | g.fillRoundRect(x, y, w, h, aw, ah) |  | | |  | |
| .drawLine( , , , ) | trace une ligne droite depuis le point initial jusqu'au point final | g.drawLine(xi, yi, xf, yf)  *g.drawLine(0, 10, this.getWidth(), 10);* |  | | |  | |
| .drawPolyline( , , ) | trace une ligne brisée passant par tous les points de coordonnées définies par les tableaux x et y, nbS indique le nb de point à utiliser | g.drawPolyline(x[ ], y[ ], nbS)  *int[] abs = {50, 100, 75, 25, 0};*  *int[] ord = {0, 33, 100, 100, 33};*  *g.drawPolyline(abs, ord, 5);* |  | | |  | |
| .drawPolygon( , , ) | trace un polygone dont on définit les coordonnées de tous les points dans 2 tableaux (un pour x, l'autre pour y), nbS indique le nb de sommet du polygone (la figure se referme automatiquement) | g.drawPolygon(x[ ], y[ ], nbS) |  | | |  | |
| .fillPolygon( , , ) | idem en plein | g.fillPolygon(x[ ], y[ ], nbS) |  | | |  | |
| .drawString( , , ) | affiche du texte aux coordonnées indiquées (coin bas. gauche du texte) | g.drawString("Ma phrase à afficher", x, y); |  | | |  | |
| .setFont( ) | modifie la police du texte en prenant en paramètre un objet de type Font | Font maPolice = new Font("nomPolice", Font.BOLD, 20);  g.setFont(maPolice); |  | | |  | |
| .drawImage( , , , , , ) | affiche une image dans notre élément  elle comprend 4 param. obligatoires et 2 facultatifs :   * l'image * coord. du coin sup gauche * *(les dimensions)* * son conteneur (souvent : this) | try {  Image monImg = ImageIO.read(new File("adresse/image.jpg");  g.drawImage(monImg, x, y, *w, h,* cont);  } catch (IOException e) {  e.printStackTrace();  } |  | | |  | |
| Graphics2D | permet d'utiliser des effets plus avancés que Graphics simple | public void paintComponent(Graphics g) {  Graphics2D g2d = (Graphics2D)g;  g2d. …  } |  | | | Remarque : pour pouvoir utiliser les effets de Graphics2D il faut caster notre objet Graphics en objets Graphics2D | |
| GradientPaint( , , , , , , ) | Classe permettant de définir un gradiant de couleur avant de l'appliquer à un élément | GradientPaint gp = new GradientPaint(x1, y1, Color.couleur1, x2, y2, Color.couleur2, bool);  g2d.setPaint(gp);  g2d.fillRect(0, 0, this.getWidth(), this.getHeight()); |  | | | * x1, y1 coord. de départ de 1re couleur * x2, y2 coord. de départ de 2e couleur * booléen true : répétition gradient, false sinon | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |
| Prog. Evènementielle |  |  | java.awt | | |  | |
|  | la mise en place d'un gestionnaire d'événement sur un élément se fait en 3 étapes :   * implémentation de sa classe avec l'interface de l'évènement * ajout du gestionnaire sur l'élément lui-même * redéfinition de la/des méthodes qui nous intéresse | public class Bouton implements MouseListener {  this.addMouseListener(this);  public void mouseEntered(MouseEvent e) {  instruction(s);  } |  | | |  | |
| MouseListener | gestionnaire d'événement souris  clic sur l'élément  survol de l'élément  curseur sort de la zone de l'élément  bouton gauche enfoncé sur l'élément  bouton gauche de la souris relâché | implements MouseListener {  this.addMouseListener(this);  public void mouseClicked(MouseEvent e) {  instruction(s);  }  mouseClicked( ) {  mouseEntered( ) {  mouseExited( ){  mousePressed( ){  mouseReleased( ){ | *Avec classe anonyme :*  JPanel pan = new JPanel();  pan.addMouseListener(new MouseAdapter() {  public void mouseReleased(MouseEvent e) {  instruction(s);  }  }); | | | Dans une classe anonyme, la déclaration d'un MouseListener peut se faire avec MouseListenr(), mais MouseAdapter() présente l'avantage de ne pas avoir à redéfinir toutes les méthodes de MouseListener, seulement celle(s) qui nous intéresse(nt) (plus pertinent pour une classe anonyme réservée aux cas les plus simples de gestionnaires) | |
| MouseMotionListener | gestionnaire d'événement suivant le mouvement de la sourie  suit les mvts de la souris dans le composant sans clic  suit les mvmts de la souris dans le composant qd clic gauche maintenu enfoncé | implements MouseMotionListener {  public void mouseMoved(MouseEvent e) {  instruction(s);  }  public void mouseDragged(MouseEvent e) {  instruction(s);  } |  | | |  | |
| KeyListener | gestionnaire d'évènement clavier  pression sur une touche  entre deux : touche pressée non relâchée  relâchement de la touche | implements KeyListener {  this.addKeyListener(this);  public void keyPressed(KeyEvent e) {  instuction(s);  }  keyTyped( ) {  keyReleased( ) { | *Avec classe interne :*  class Fenêtre extends JFrame {  …  jtf = new JTextField();  jtf.addKeyListener(new ClavierListener());  }  class ClavierListener implements KeyListener {  public void keyPressed(KeyEvent e) {  …  …  } | | |  | |
| e.getKeyCode() | retourne le code de la touche |  |  | | |  | |
| e.getKeyChar() | retourne le caractère de la touche |  |  | | |  | |
|  |  |  |  | | |  | |
| ActionListener | gestionnaire d'événement qui permet de réagir dans une autre classe à un événement intervenu sur un élément présentant son propre gestionnaire | implements ActionListener {  monObjet.addActionListener(this);  public void actionPerformed(ActionEvent e) {  instruction(s);  } |  | | | Remarque : monObjet est l'élément qui présente déjà un gestionnaire classique (souris, …)  une seule méthode à redéfinir pour l'action | |
| e.getSource() | donne le nom de l'élément sur lequel à eu lieu l'action détectée,  utile si un même ActionListener "écoute" plusieurs événements et y réagit différement | e.getSource(); |  | | |  | |
| classe interne | si une seule classe doit suivre plusieurs événements, il est préférable de déclarer une classe interne pour chaque gestionnaire ActionListener afin de ne pas surcharger la méthode actionPerformed( ) | monBouton.addActionListener(MaClasseInterne); | *Exemple de déclaration avec classe interne :*  class Fenetre extends JFrame {  …  JButton monBouton = new JButton("Go");  monBouton.addActionListener(new BoutonGoListener());  …  }  class BoutonGoListener implements ActionListener {  public void actionPerformed(ActionEvent e) {  instruction(s);  }  } | | |  | |
| classe anonyme | dans un cas simple de gestion d'événement (1 ou 2 lignes de code) il est plus rapide de faire appel à une classe anonyme | monBouton.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent event) {  instruction(s);  }  }); |  | | |  | |
| ItemListener | gestionnaire d'évènement spécifique aux liste déroulantes (JComboBox)  *voir* Cours - Formulaires) |  |  | | |  | |
|  |  |  |  | | |  | |
| Drag 'n Drop | Glisser-Déposer |  |  | | |  | |
| .setDragEnabled( ) | permet d'activer le Drag par défaut pour le contenu de certains éléments  (peut être personalisé en attribuant ensuite un TransferHandler personalisé au composant) | JTextArea monTxtArea = new JTextArea("Contenu déplassable");  monTxtArea.setDragEnable(true);  monTxtAream.setTransferHandler(new MonTransferHandler); | *Composant permettant*   |  |  |  | | --- | --- | --- | | *Drag* | *Drop* | *Les 2* | | JFileChooser  JTable  JTree  JList | JPasswordTextField | JEditorPane  JColorChooser  JTextPane  JTextField  JTextArea  JFormattedTextField | | | | Rq: JLabel ne permet ni l'un, ni l'autre ! | |
| TransferHandler  .exportAsDrag( , , ) | Pour tous les JCompnent ne présentant pas cette option par défaut, on doit :  lui affecter un TransferHandler (qui permet de gérer les données transférées entre les éléments lors du déplacement)  amorcer la procédure du Drag n Drop lors de l'événement souris  crée un racourcis vers les données cibles  n'autorise rien | monComposant.setTransferHandler(new TransferHandler("Text"));  monComposant.addMouseListener(new MouseAdapter() {  public void mousePressed(MouseEvent e) {  JComponent monComp = (JComponent)e.getSource();  TransferHandler tHandler = monComp.getTransferHandler();  tHandler.exportAsDrag(monComp, e, TransferHandler.COPY);  }  });  TransferHandler.COPY  TransferHandler.MOVE  TransferHandler.COPY\_OR\_MOVE  TransferHandler.LINK  TransferHandler.NONE | | *Avec TransferHandler personalisé :*  monComposant.setTransferHandler(new MonTransferHandlerPerso()); | | Rq: possibilité de personnaliser entièrement le TransferHandler *(voir Java-ExempleCodes)*  ATTENTION : si TransferHandler est instancié avec "Text", il n'acceptera que .COPY | |
| DropLocation | objet contenu dans TransferSupport et permettant des actions spécifiques avec les JList, JTree et JTable | *Dans le TransferHandler :*  public boolean importData(TransferHandler.TransferSupport support){  if(!canImport(support))  return false;    JTree.DropLocation dl = (JTree.DropLocation)support.getDropLocation();  *Dans la méthode de construction du JTree :*  tree.setDragEnabled(true);  tree.setTransferHandler(new MonTreeTransferHandler(tree));  tree.setDropMode(DropMode.ON\_OR\_INSERT); | | | |  |  | | --- | --- | | *JList* | *JTable* | | dl.isInsert()  dl.getIndex() | dl.isInsertRow()  dl.isInsertColumn()  dl.getRow()  dl.getColumn() | | *JTree* | | dl.getChildIndex()  dl.getPath() | | | *(voir Java-ExempleCodes)* |
|  |  |  |  | | |  | |
|  |  |  |  | | |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  | | |  | | | | |  | | |
| EDT | Event Dispatch Thread = thread reservé aux composants graphiques |  | | |  | | | | |  | | |
| SwingUtilities | classe offrant plusieurs méthodes statiques permettant d'insérer des instructions liès aux composants graphisques dans l'EDT |  | | | Méthodes thread-safe pouvant être appellées depuis n'importe quel thread :  .paint() .repaint()  .validate() .revalidate() .invalidate() | | | | |  | | |
| .invokeLater( ) | exécute le thread passé en paramètre dans l'EDT et rend immédiatement la main au thread principal | Thread t = new Thread(new Runnable() {  public void run() {  instruction(s);  }  });  SwingUtilities.invokeLater(t); | | | Thread t = new Thread(new Runnable() {  public void run() {  monBouton.setText("Cliqué " + (++count) + " fois");  }  });  if(SwingUtilities.isEventDispathThread()) {  t.start();  } else {  SwingUtilities.invokeLater(t);  } | | | | |  | | |
| .inkokeAndWait( ) | idem, mais attends la fin du traitement avant la reprise du thread principal | InterruptedException  InvocationTargetException | | |  | | |
| .isEventDispatchThread() | retourne true si le thread dans lequel cette méthode est apellé s'éxécute dans l'EDT | SwingUtilities.isEventDispathThread() | | |  | | |
| SwingWorker<T, V> | classe abstraite à utiliser pour réaliser des traitments lourds devant intéragir avec EDT  interaction avec EDT :   * dirrectement dans done() * par setProgress(int nb) dans doInBackground() via un PropertyChangeListener | class MaSwingWorker extends SwingWorker<ObjectA, ObjectB> {  *//méthode appelée par .execute()*  protected ObjectA doInBackground() throws Exception {  *instruction(s) en tâche de fond;*  setProgress(nb); *//changement de la propriété d'état*  publish(monObjectB); *//publication intermédiaire*  return monObjectA;  }  *//méthode appelée à la fin du traitement de doInBackground()*  public void done() {  *instruction(s) à faire dans l'EDT;*  try {  ObjectA monObjectA = get();  } catch (InterruptedException e) {  } catch (ExecutionException e) {  }  }  *//méthode gérant les résultats intemédiaires*  public void process(List<ObjectB> list) {  instruction(s);  }  }  *Dans le thread principal :*  MaSwingWorker maSW = new maSW();  maSW.addPropertyChangeListener(new PropertyChangeListener() {  public void propertyChange(PropertyChangeEvent e) {  if("progress".equals(e.getPropertyName()))  *instruction(s) à faire d'ans l'EDT*  (Integer)e.getNewValue());  }  });  maSW.execute(); | | | | | | *Classe anonyme :*  SwingWorker sw = new SwingWorker() {  protected Object doInBackground() throws Exception {  *instruction(s) en tâche de fond;*  }  public void done() {  *instruction(s) à faire dans l'EDT;*  }  };  sw.execute(); | | ATTENTION : seuls .done() et le PropertyChangeListener s'assurent que ses instructions se fassent dans l'EDT.  Rq: .done() est appelée automatiquement une fois à la fin du traitement de .doInBackground()  Rq: seul .doInBackground() doit être obligatoirement redéfinie, les autres méthades et listeners étants facultatifs. | | |
| get() | récupère le résultat (de type ObjetA) retourné à la fin du traitement de doInBackground() | pubic void done() {  ObjetA objA = get(); | | |  | | | | |  | | |
| cancel()  isConcelled() | essaie d'interrompre la tache de doInBackground() en cours  renvoie true si tache intérompue | pubic void done() {  cancel();  if(isCancelled()) { | | |  | | | | |  | | |
| isDone() | retourne true si l'action de doInBackground est finie | pubic void done() {  if(isDone()) { | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Formulaires |  |  | | | java.swing | | | | |  | | |
| JComboBox( ) | liste déroulante  le constructeur peut prendre un tableau d'objet en paramètre  sinon, utilisation de la méthode .addItem( ) | combo = new JComboBox();  combo.addItem(monObjetA);  combo.addItem(monObjetB);  combo.setPreferredSize(new Dimension(100, 20));  String[ ] tableauDeChoix = {"Option A", "Option B", … };  combo = JComboBox(tableauDeChoix);  combo.setPreferredSize(new Dimension(100, 20)); | | | *Avec un type générique :*  JComboBox<String> combo = new JComboBox<String>(); | | | | | Rq: pour afficher l'option dans la liste, JComboBox appellera la méthode .toString() des objets qui lui sont placés en paramètre | | |
| .setSelectedIndex( ) | assigne un choix par défaut à la liste (facultatif) | combo.setSelectedIndex(2); | | |  | | | | |  | | |
| ActionListener | pour obtenir la valeur sélectionnée en direct par l'utilisateur, on utilise une classe implémentant ActionListener | combo.addActionListener(new ItemAction());  class ItemAction implements ActionListener {  public void actionPerformed(ActionEvent e) {  System.out.println("Action sur : " + combo.getSelectedItem());  }  } | | |  | | | | | Rq: la classe peut être déclarée en interne dans le même fichier si instructions pas trop longues.  ATTENTION : la méthode .getSelectedItem() renvoie un objet et non l'intitulé du choix (ici : *Option B*). Pour obtenir ce dernier, on utilise .toString() | | |
|  |  |  | | |  | | | | |  | | |
| JCheckBox | case à cocher | maCheckBox = new JCheckBox("Choix 1"); | | |  | | | | |  | | |
| .setSelected( ) | case cochée par défaut (facult.) | maCheckBox.setSelected(true); | | |  | | | | |  | | |
| ActListener |  | (JCheckBox)e.getSource()).getText();  (JCheckBox)e.getSource()).isSelected();  maCheckBox.isSelected(); | | | => Choix 1  => true / false  => true / false | | | | |  | | |
| JRadioButton | bouton radio | JRadioButton jrb1 = newJRadioButton("Choix A");  JRadioButton jrb2 = newJRadioButton("Choix B");  ButtonGroup bg = new ButtonGroup();  bg.add(jrb1);  bg.add(jrb2); | | | *Insertions des boutons dans le panel :*  monPanel.add(jrb1);  monPanel.add(jrb2); | | | | | ATTENTION : ButtonGroup ne peut être ajouté au panel en une seule étape. On doit ajouter les boutons un à un après avoir lié leur traitement avec ButtonGroup | | |
| .setSelected( ) | case pré-selectionnée (facult.) | jrb2.setSelected(true); | | |  | | | | |  | | |
| ActListener |  | jrb1.getText();  jrb2.isSelected; | | | => Choix A  => true / false | | | | |  | | |
| JTextField | champ de texte | JTextField jtf = new JTextField("Valeur par défaut"); | | |  | | | | |  | | |
| .setText( ) | remplis le contenu du champs | jtf.setText("Nouvelle valeur par défaut"); | | |  | | | | |  | | |
| .getText() | récupère le contenu du champ | String contenu = jtf.getText(); | | |  | | | | |  | | |
| .setPreferedSize( ) | définit les dimensions par défaut | jtf.setPreferedSize(new Dimension(300, 30)); | | |  | | | | |  | | |
| .setForegroung( ) | définit la couleur de la police | jtf.setForeground(Color.BLUE); | | |  | | | | |  | | |
| .setFont( ) | définit la police à utiliser | Font police = new Font("Arial", Font.BOLD, 14);  jtf.setFont(police); | | |  | | | | |  | | |
| JFormattedTextField | champ de texte au contenu formaté (= n'acceptant qu'un nombre entier, une date, un horaire, un pourcentage, …) | jftf = new JFormattedTextField(NumberFormat.getIntegerInstance()); | | |  | | | | | Rq: la méthode .getText() renverra alors   * le contenu du champ s'il est valide * rien si contenu non valide et pas de valeur valide précédente * valeur valide précédente si le champ à déjà été utilisé et que la nouvelle valeur n'est pas valide | | |
|  | formatassions de bases :  nombre entier  pourcentage  nb jusqu'à 3 chiffres après la virgule  avec format à définir | NumberFormat.getIntegerInstance()  NumberFormat.getPercentInstance()  NumberFormat.getNumberInstance()  DateFormat.getTimeInstance()  DateFormat.getDateInstance()  MessageFormat | | |  | | | | |  | | |
| MaskFormatter | objet permettant de définir un format personnalisé grâce aux métacaractères | try {  MaskFormatter monFormat = new MaskFormatter(maChaineMetachar);  JFormattedTextField jftf = new JFormattedTextField(monFormat);  } catch (ParseException e) {  e.printStackTrace();  } | | | *Métacaractères :*   * ' : caractère d'échappement * \* : n'importe quel caractère * # : chiffre * A : chiffre ou lettre * ? : lettre * U : lettre (min -> MAJ) * L : lettre (MAJ -> min) * H : caractère hexadécimal | | | | | *Exemple*  *n° de tel : "## ## ## ## ##"* | | |
| JTextArea | zone de texte étendue  comme un JTextField en plus grand  (méthodes identiques à JTextField) | JTextArea textPane = new JTextArea(); | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Boite de dialogue |  |  | | | javax.swing | | | | |  | | |
| JOptionPane | boite de dialogue | JOptionPane jop = new JOptionPane; | | |  | | | | |  | | |
| types de messages | 5 types de messages permettent de définir les boites de dialogues qui suivent |  | | | *5 types de messages possibles :*  JOptionPane.PLAIN\_MESSAGE  JOptionPane.INFORMATION\_MESSAGE  JOptionPane.QUESTION\_MESSAGE  JOptionPane.WARNING\_MESSAGE  JOptionPane.ERROR\_MESSAGE | | | | |  | | |
| .showMessageDialog( ) | défini une boite de dialogue informative  peut prendre 2, 4 ou 5 paramètres :   * **composant parent (si pas :** null**)** * **corps du message** * titre de la fenêtre * type de message * *image de l'icone* | jop.showMessageDialog(Component parentComponent, String message);  jop.showMessageDialog(Component parentComponent, String message, String titre, int typeDeMessage);  jop.showMessageDialog(Component parentComponent, String message, String titre, int typeDeMessage, ImageIcon image); | | | *Création d'un objet de type* ImageIcon *:*  ImageIcon img = new ImageIcon("Adresse de l'image depuis la racine du projet Eclipse"); | | | | |  | | |
| .showConfirmDialog( ) | défini une boite de confirmation permettant de valider, invalider ou annuler l'action en cours  elle retourne un entier en fonction du choix de l'utilisateur  Paramètres :   * composant parent (si pas : null) * corps du message * titre de la fenêtre * **type d'options dans la fenêtre** * type de message * *image de l'icone* | int option = jop.showConfirmDialog(null, "Etes-vous sûr ?", "Demande de confirmation", JOptionPane.YES\_NO\_OPTION, JOptionPane.QUESTION\_MESSAGE);  if(option == JOptionPane.OK\_OPTION) {  instruction(s);  } | | | *Types d'options proposés par la fenêtre :*  JOptionPane.YES\_NO\_OPTION  JOptionPane.YES\_NO\_CANCEL\_OPTION  *4 valeurs de renvoie sont possibles :*  JOptionPane.OK\_OPTION = JOptionPane.YES\_OPTION = 0  JOptionPane.NO\_OPTION = 1  JOptionPane.CANCEL\_OPTION = 2  JOptionPane.CLOSED\_OPTION = -1 | | | | |  | | |
| .showInputDialog( ) | défini une boite de saisie  retourne une String  Paramètres :   * composant parent (si pas : null) * corps du message * titre de la fenêtre * type de message * *image de l'icone* | String maChaine = jop.showInputDialog(null, "Entrez une valeur", "Demande info", JOptionPane.INFORMATION\_MESSAGE); | | |  | | | | |  | | |
|  | pour définir une liste déroulante dans une boite de saisie, il suffit de rajouter des paramètres :   * 6e : tableau de valeur * 7e : valeur par défaut de la liste | String[ ] monTableau = {"Choix 1", "Choix 2", … };  String maChaine = jop.showInputDialog(null, "Entrez une valeur", "Demande info", JOptionPane.INFORMATION\_MESSAGE, null, monTableau, monTableau[2]); | | |  | | | | | Rq: ici, le 2nd null dans les paramètres correspond à l'image de l'icône si on ne souhaite pas lui en attribuer une.  ATTENTION : comme avec une ComboBox, la valeur retournée est l'objet du tableau lui-même et non pas une String | | |
| JDialog | classe permettant de créer une boite de dialogue personnalisée  initComponent() contiens ensuite les instructions pour remplir la boite de dialogue avec les mêmes éléments qu'une fenêtre classique (JPanel, Layout, JButton, JTextField, JComboBox, JRadioButton, ...)  et les gestionnaires associés pour la récupération des données saisies dans cette boite de dialogue  Le dialogue débute quand la fenêtre devient visible : maDialogBox.setVisible(true);  et se termine quand elle disparait :  maDialogBox.setVisible(false);  instructions gérées par des Listeners | class maDialogBox extends JDialog {  public maDialogBox(JFrame parent, String title, boolean modal) {  //utilise le contructeur de JDialog  super(parent, title, modal);  this.setSize(550, 270);  this.setLocationRelativeTo(null);  this.setResizable(false);  this.setDefaultCloseOperation(  JDialog.DO\_NOTHING\_ON\_CLOSE);  this.initComponent();  }  private void initComponent() {  instruction(s);  } | | | *Exemple complet de boite de dialogue personnalisée :*  *voir* Eclipse-Java projet : cours 10-boiteDial  *Portion de code intéressant :*  //Icone  icon = new JLabel(new ImageIcon("imgs/icone.jpg"));  JPanel panIcon = new JPanel();  panIcon.setBackground(Color.WHITE);  panIcon.setLayout(new BorderLayout());  panIcon.add(icon);  //Le nom  JPanel panNom = new JPanel();  panNom.setBackground(Color.white);  panNom.setPreferredSize(new Dimension(220, 60));  nom = new JTextField();  nom.setPreferredSize(new Dimension(100, 25));  panNom.setBorder(BorderFactory.createTitledBorder("Nom du personnage"));  nomLabel = new JLabel("Saisir un nom : ");  panNom.add(nomLabel);  panNom.add(nom); | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Menus |  |  | | | javax.swing | | | | |  | | |
| JMenuBar | barre de menus |  | | | public class maFenetre extends JFrame {  private JMenuBar menuBar = new JMenuBar();  private JMenu menuA = new JMenu("Fichier");  private JMenu menuAa = new JMenu("Sous-fichier");  private JMenu menuB = new JMenu("Edition");  private JMenuItem itemA1 = new JMenuItem("Ouvrir");  private JMenuItem itemAa1 = new JMenuItem("Sous-option 1");  private JMenuItem itemAa2 = new JMenuItem("Sous-option 2");  private JMenuItem itemA2 = new JMenuItem("Fermer");  …  public maFenetre() {  …  this.menuA.add(itemA1);  this.manuAa.add(itemAa1);  this.menuAa.add(itemAa2);  this.menuA.add(menuAa);  this.menuA.addSeparator();  this.itemA2.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  System.excit(0);  }  });  this.menuA.add(itemA2);  this.menuB.add(…);  this.menuBar.add(menuA);  this.menuBar.add(menuB);  this.setJMenuBar(menuBar);  this.setVisible(true);  } | | | | |  | | |
| JMenu | menu ou sous-menu |  | | |  | | |
| JMenuItem | élément du menu induisant une action lorsqu'il est sélectionné |  | | | *Dans un Listener :*  ((JMenuItem)e.getSource()).getText() | | |
| .add( ) | permet d'insérer un élément dans un menu ou dans la barre de menu |  | | |  | | |
| .addSeparator() | ajoute un séparateur entre deux éléments ou deux menus |  | | |  | | |
| .setJMenuBar( ) | A appliquer sur l'objet maFenetre pour ajouter la barre de menu préalablement définie à la fenêtre |  | | |  | | |
| JCheckBoxMenuItem | case à cocher spécifique aux menus |  | | |  | | | | |  | | |
| JRadioButtonMenuItem | boutons radio spécifiques aux menus |  | | |  | | | | | *Dans un Listener :*  ((JRadioButtonMenuItem)e.getSource()).getText() | | |
|  |  |  | | |  | | | | |  | | |
| Raccourcis claviers |  |  | | |  | | | | |  | | |
| .setMnemonic( ) | définit un raccourci clavier mnémonique (accessible par alt + touche) donnant accès aux menus  pour JMenu et JMenuItem  paramètre : char | monJMenu = new JMenu("Fichier", 'F');  monJMenu.setMnemonic('F');  monJMenuItem.setMnemonic('I'); | | |  | | | | |  | | |
| .setAccelerator( ) | définit un raccourci clavier accélérateur (combinaison de touches) effectuant une action directe  pour JMenuItem | monJMenuItem.setAccelerator(KeyStroke.getKeyStroke('a'));  monJMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_A, KeyEvent.CTRL\_DOWN\_MASK));  monJMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK\_A, KeyEvent.CTRL\_DOWN\_MASK + KeyEvent.SHIFT\_DOWN\_MASK)); | | |  | | | | | ATTENTION : dans le premier cas, 'a' ≠ 'A' (= Maj + a)  Rq: VK\_ = Value of Key  ATTENTION : .getKeyStroke ne prend   * qu'un param : char * 2 param : VK\_ lettre/chiffre , CTRL   si on veut Ctrl + Maj + lettre : VK\_ , CTRL + SHIFT | | |
|  |  |  | | |  | | | | |  | | |
| Menu contextuel | = menu clic droit |  | | |  | | | | |  | | |
| JPopupMenu | menu contextuel se remplissant avec des JMenu et JMenuItem | jpm = new JPopupMenu();  jpm.add(jmenuA);  jpm.add(jmenuitem1); | | | *Déclaration d'un menu contextuel s'affichant au clic droit sur un JPanel :*  monPanel = new JPanel();  monPanel.addMouseListener(new MouseAdapter() {  public void mouseReleased(MouseEvent e) {  if(e.isPopupTrigger()) {  jpm.show(monPanel, e.getX(), e.getY());  }  }  });  container.add(monPanel, BorderLayout.CENTER);  maFenetre.setContentPane(container); | | | | | ATTENTION : pour une même instruction dans la barre de menus et dans le menu contextuel, DEUX JMenuItem doivent être créés et insérés un dans chaque menu (car les JMenuItem n'apparaissent qu'une seule fois dans la fenêtre : dans le menu auquel il est affecté en dernier dans le code)  En revanche, ils peuvent être suivit par la même instance de Listener ! | | |
| .show( , , ) | affiche le menu contextuel  param :   * composant parent qui invoque le menu * position en X * position en Y |  | | |  | | |
| .getInvoker() | permet d'appeler un 2nd JPopupMenu à partir du premier | jpm.getInvoker();  jpm.getInvoker().getX();  jpm.getInvoker().getY(); | private void initPopupMenu() {  JMenuItem ajouterEmplMI = new JMenuItem("Ajouter Employé");  ajouterEmplMI.addActionListener(new ActionListener() {  public void actionPerformed(ActionEvent e) {  initEmplPotentielsMenu();  emplPotentielsMenu.show(jpm.getInvoker(), me.getX(), me.getY());  }  });  jpm.add(ajouterEmplMI);  } | | | | | | |  | | |
| Barre d'outils |  |  | | |  | | | | |  | | |
| JToolBar | barre d'outil se remplissant avec des JButton | JToolBar maBarreOutils = new JToolBar();  JButton bouton1 = new JButton(new ImageIcon("imgs/monimage.jpg"));  Color fondBouton = Color.WHITE;  bouton1.setBackground(fondBouton);  bouton1.addActionListener(monListener);  maBarreOutils.add(bouton1);  maFenetre.add(maBarreOutil, BorderLayout.NORTH); | | *Cas où les boutons de la barre d'outils font référence à des boutons radio dans le menu : appel à ces bouton radio par la méthode* .doClick()  class FormeListener implements ActionListener {  public void actionPerformed(ActionEvent e) {  //si l'action vient d'un bouton radio, ok  if(e.getSource().getClass().getName().equals("javax.swing.JRadioButtonMenuItem"))  pan.setForme((JRadioButtonMenuItem)e.getSource()).getText());  else { //si vient de barre d'outils : appel artificiel au bouton radio correspondant  if(e.getSource() == square)  carre.doClick();  else if(e.getSource() == tri)  triangle.doClick();  else  rond.doClick();    }  }  } | | | | | | | | Rq: ici, carre, triangle et rond sont des JRadioButtonMeniItem, alors que square et tri sont des JButton insérés dans la barre d'outils |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
| Arbres |  |  | | |  | | | | |  | | |
| JTree  DefaultMutableTreeNode | défini une arborescence (= succession de nœuds qui peuvent être indifféremment des dossier ou des fichiers) | racine = new DefaultMutableTreeNode("Racine");  doss = new DefaultMutableTreeNode("DossierA");  fichierA1 = new DefaultMutableTreeNode("Fichier1");  fichier2 = new DefaultMutableTreeNode("Fichier 2");  doss.add(fichierA1);  racine.add(doss);  racine.add(fichier2);  arbre = new JTree(racine);  maFenetre.getContentPane().add(new JScrollPane(arbre)); | | | | *Création arbre à partir lecteurs racines de l'ordinateur :* ***voir Java-ExempleCodes*** | | | |  | | |
| .setRootVisible( ) | appliqué à l'objet JTree avec un booléen pour ne pas afficher l'élément "racine" s'il ne correspond à rien | arbre.setRootVisible(false); | | |  | | | | |  | | |
| TreeSelectionListener | listener de l'objet JTree permettant d'interagir avec lui  ici en classe anonyme  mais possible avec implémentation | arbre.addTreeSelectionListener(new TreeSelectionListener() {  public void valueChanged(TreeSelectionEvent e) {  if(arbre.getLasteSelectedPathComponent() != null) {  instruction(s);  }  }  }); | | | | |  | | |  | | |
| .getLastSelectedPathComponent().toString() | appliqué sur JTree : donne le nom de l'élément selectionné | arbre.getLastSelectedPathComponent().toString() | | |  | | | | |  | | |
| e.getPath() | retourne un objet TreePath contenant une liste d'objet correspondant aux nœuds du chemin d'accès à un élément de l'arbre | arbre.getSelectedPathComponent | | | *Donne le chemin d'accès complet à l'élément sélectionné :*  *(dans* valueChanged(…) *du Listener :)*  getAbsolutePath(e.getPath());  *(2nd méthode du Listener :)*  private String getAbsolutePath(TreePath tp) {  String str = "";  //on balaie le contenu de l'objet TreePath  for(Object name : tp.getPath()) {  //si l'objet a un nom, on l'ajoute au chemin d'accés  str += name.toString();  }  return str;  } | | | | | Rq : avec cette méthode dans le Listener, pour récupérer un objet, il suffit de faire :  File file = new File(getAbsolutePath(e.getPath)); | | |
| Modifier le contenu d'un arbre | ajouter/supprimer/renommer les éléments  => voir ***cours - Modifier le contenu de nos arbres*** |  | | |  | | | | |  | | |
| DefaultTreeCellRenderer | modifier la mise en forme de l'arbre  icone de dossier fermé  icone de dossier ouvert  icone de fichier | dtcr = new DefaultTreeCellRenderer();  dtcr.setClosedIcon(new ImageIcon("adresse"));  dtcr.setOpenIcon(…);  dtcr.setLeafIcon(…);  arbre.setCellRenderer(dtcr); | | | *Utiliser le "look and feel" de notre système d'exploitation :*  *(a insérer dans constructeur de la fenêtre)*  try {  //On force à utiliser le « look and feel » du système  UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  //Ici on force tous les composants de notre fenêtre (this) à se redessiner avec le « look and feel » du système  SwingUtilities.updateComponentTreeUI(this);  } catch (InstantiationException e) {  } catch (ClassNotFoundException e) {  } catch (UnsupportedLookAndFeelException e) {  } catch (IllegalAccessException e) {  }  this.setVisible(true); | | | | | |  | |
|  |  |  | | |  | | | | |  | | |
| Interfaces Tableaux |  |  | | |  | | | | |  | | |
| JTable | défini un tableau pouvant être affiché à partir des données d'un tableau à double entrée d'objets  ATTENTION : tjs insérer le tableau dans un JScrollPane avant de le placer dans la fenêtre, sinon il n'affichera pas le titre des colonnes ! | Object[ ][ } data = {  {objetA, "String A", 1},  {objetB, "String B", 2}  };  String[ ] titles = {"Colonne1", "Colonne2"};  tableau = new JTable(data, titles);  maFenetre.getContentPane.add(new JScrollPane(tableau)); | | | *Constructeur à partir d'un JTableModel :*  modelT = new ModelTPerso(data, titles);  tableau = new JTable(modelT);  *(voir classe abstraite de JTableModel plus bas)* | | | | | ATTENTION : l'objet JTable n'accepte que des données sous forme de tableau d'Object !  Rq : les titres des colonnes peuvent être sous forme de tableau d'Object ou de tableau de String. | | |
| .getValueAt( , ) | retourne le contenu de la cellule de la ligne et colonne indiquée  Param : ligne, colonne | tableau.getValueAt(0, 1); => "String A" | | |  | | | | |  | | |
| .getRowCount()  .getColumnCount() | retourne le nombre de ligne/colonne contenu dans le tableau | tableau.getRowCount();  tableau.getColumnCount(); | | |  | | | | |  | | |
| .setRowHeight( , ) | défini la hauteur (en pixel) de la cellule d'index donné | tableau.setRowHeight(2, 40); | | |  | | | | |  | | |
| TableColumn | objet regroupant les informations liées à chaque colonne du tableau | TableColumn tCol = tableau.getColumnModel().getColumn(2); | | |  | | | | |  | | |
| .setPreferredWidth( ) | défini la largeur (en px) de la colonne | tCol.setPreferredWidth(70); | | |  | | | | |  | | |
| Mise en forme |  |  | | |  | | | | |  | | |
| JTableModel | objet regroupant toutes les informations liées au tableau. Il fait le lien entre les données et le tableau. |  | | |  | | | | |  | | |
| AbstractTableModel | pour définir/redéfinir/contrôler le contenu des cellules du tableau en fonction de leur type, on crée une classe (interne ou non) héritant de cette classe abstraite   * affiche booléen en case cochée * permet l'édition des données des cellules | ModelTPerso modelT = new ModelTPerso(data, titles);  tableau = new JTable(modelT);  - - - - - - - - - - - -  class ModelTPerso extends AbstractTableModel {  private Object[][] data;  private String[] title;    //Constructeur  public ModelTPerso(Object[][] data, String[] title) {  this.data = data;  this.title = title;  }    public int getColumnCount() {  return this.title.length;  }  public int getRowCount() {  return this.data.length;  }  public Object getValueAt(int row, int col) {  return this.data[row][col];  }  public String getColumnName(int col) {  return this.title[col];  }  instruction(s);  } | | | *méthodes facultatives pouvant être utiles de rajouter dans la classe précédente :*  //transforme les booléens en case à cocher  public Class getColumnClass(int col) {  //le type de contenu ne dépends que de la colonne (il est id pour toutes les lignes)  return this.data[0][col].getClass();  }    //retourne vrai si la cellule est éditable (cad tjs sauf pour un bouton)  public boolean isCellEditable(int row, int col) {  if(getValueAt(0, col) instanceof JButton)  return false;  return true;  }  //interdit la modification du contenu même si action dessus (important pour les boutons !)  public void setValueAt(Object value, int row, int col) {  //on interdit la modification sur certaines colonnes  if(!this.getColumnName(col).equals("Bouton") && !this.getColumnName(col).equals("Supression"))  this.data[row][col] = value;  } | | | | |  | | |
| DefaultTableCellRender | objet gérant l'affichage (MeP) du tableau, du contenu des cellules  (qd on rajoute getColumnClass() à la classe abstraite ci-dessus, c'est lui qui utilise cette info pour transformer les booléens en case à cocher)   * affiche les JButton comme bouton * affiche les JComboBox comme liste déroulantes | *Classe héritant de TableCellRenderer et permettant d'afficher automatiquement les JButton et ComboBox dans les cellules concernées :*  tableau.setDefaultRenderer(JComponent.class, new TableComponent());  - - - - - - - - - - -  class TableComponent extends DefaultTableCellRenderer {  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {  //si la valeur de la cellule est un JButton ou JComboBox, on transtype cette valeur  if (value instanceof JButton)  return (JButton) value;  else if(value instanceof JComboBox)  return (JComboBox) value;  else  return this;  }  } | | | | | | | *Idem avec classes internes :*  //MaJ du tableau en remplaçant le contenu de la colonne choisie avec les boutons  tableau.getColumn("TitreColB").setCellRenderer(new ButtonRenderer());  //idem pour les ComboBoxde  tableau.getColumn("TitreColC").setCellRenderer(new ComboRenderer());  - - - - - - - - - - - - -  class ButtonRenderer extends JButton implements TableCellRenderer {  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean isFocus, int row, int col) {  //on écrit dans le bouton ce que contient la cellule  this.setText((value != null) ? value.toString() : "");  //on retourne le bouton  return this;  }  }    class ComboRenderer extends JComboBox implements TableCellRenderer {  public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean isFocus, int row, int col) {  //on remplie la liste  this.addItem("Choix1");  this.addItem("Choix2");  //on retourne la ComboBox  return this;  }  } | | | |
| Action sur les éléments du tableau |  |  | | |  | | | | |  | | |
| DefaultCellEditor | objet permettant de suivre les actions sur le tableau   * action sur les boutons * action sur les liste déroulantes   ATTENTION : DefaultCellEditor ne gère que la partie action de l'utilisateur. Pour afficher les éléments sous le bon format il faut au préalable les avoir mis en forma grâce aux méthodes vues ci-dessus.  Rq: le constructeur de DefaultCellEditor peut prendre en paramètre   * JComboBox * ou JTextField   mais pas un JButton | *Pour des listes déroulantes :*  private String[ ] comboData = {"Choix 1", "Choix 2"};  JcomboBox combo = new JComboBox'comboData);  tableau.getColumn("TitreColC").setCellEditor(new DefaultCellEditor(combo)); | | | *Le constructeur ne prenant pas en charge les* JButton*, on doit nécessairement pour eux passer par une classe externe héritant de cette classe :*  tableau.getColumn("TitreColB").setCellEditor(new ButtonEditor());  - - - - - - - - - - - - -  public class ButtonEditor extends DefaultCellEditor {  protected JButton button;  private ButtonListener bListener = new ButtonListener();  public ButtonEditor() {  super(new JCheckBox()); //Contructeur avec une CheckBox par défaut  button = new JButton(); //on crée à nouveau un bouton  button.setOpaque(true);  button.addActionListener(bListener); //on lui attribue un listener  }    public Component getTableCellEditorComponent(JTable table, Object value, boolean isSelected, int row, int column) {  bListener.setRow(row); //donne les infos au Listener  bListener.setColumn(column);  bListener.setTable(table);    //on affecte le libélé au bouton  button.setText((value == null) ? "" : value.toString());  return button;  }    class ButtonListener implements ActionListener {  private int column, row;  private JTable table;  private JButton button;  public void setColumn(int col) {this.column = col;}  public void setRow(int row) {this.row = row;}  public void setTable(JTable table) {this.table = table;}  public JButton getButton() {return this.button;}    public void actionPerformed(ActionEvent e) {  System.out.println("Action sur le bouton : " + ((JButton)e.getSource()).getText());  //on affecte un nouveau libélé à une autre cellule de la ligne  table.setValueAt("New value " + (++nb), this.row, (this.column - 1));  //si affichage des boutons controlé par un Modele personnalisé (classe héritée de AbstractTableModel), contrôle l'affichage de ces derniers en faisant le lien avec ce modele  ((AbstractTableModel)table.getModel()).fireTableCellUpdated(this.row, this.column - 1);  this.button = ((JButton)e.getSource());  }  }  } | | | | | | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |
|  |  |  | | |  | | | | |  | | |