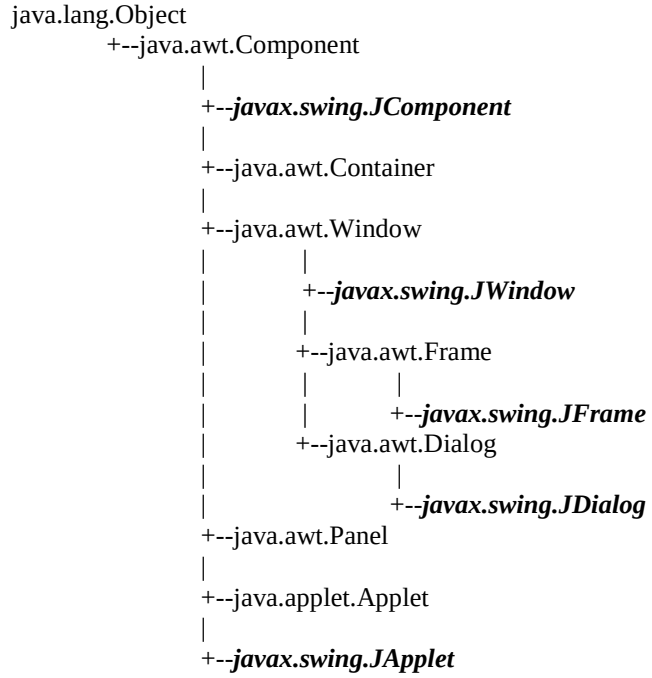
Les IHM en java

1] Organisation des packages et composants

- ⇨ Deux bibliothèques stockées dans deux packages :
  - AWT Abstract Windows Toolkit
    - Composants lourds dépendants de l'OS
  - Swing : plus récent
    - Composants légers écrits en java
    - Sauf les 4 containers JFrame, JWindow, JDialog, JApplet
    - Meilleure portabilité que AWT
    - La classe lookAndFeel détermine l'aspect des composants (classique, métalique…)
- ⇨ Il est déconseillé de mélanger les deux familles de composants.
- ⇨ Chacune de ces familles gère l'aspect graphique des composants, et surtout les événements sur ces composants.
- ⇨ La disposition des composants est gérée par des **layoutManager**



Pour utiliser ces composants, il faut importer **javax.swing** ou **java.awt**.

Les événements sont gérés par les **Listener** du package **java**.awt.event.

Les composants légers héritent tous directement ou indirectement de la classe **javax.swing.JComponen**t qui hérite elle-même de la classe **java.awt.Component** :

2] Les fenêtres en java

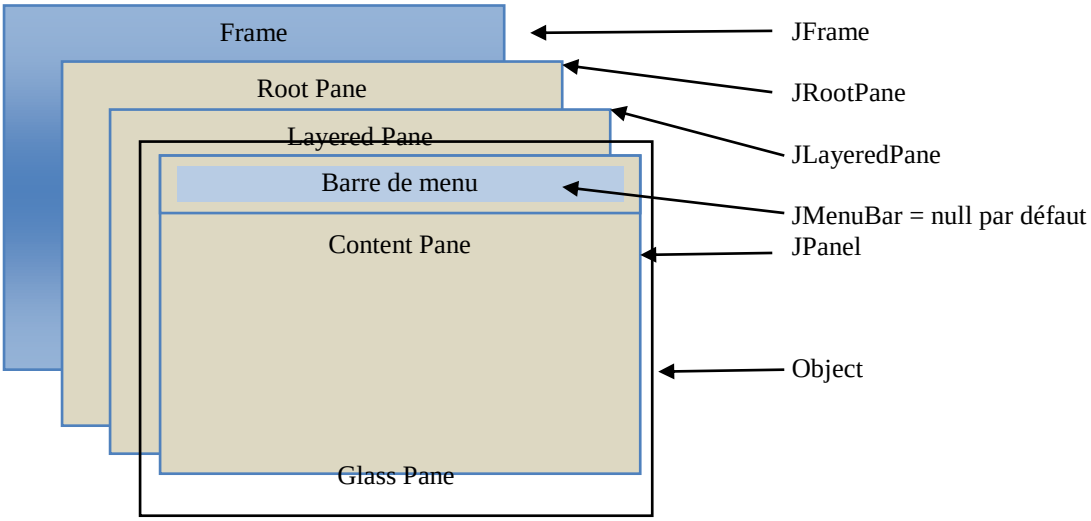
Deux possibilités :

- ⇨ Utiliser la classe Frame de java.awt
- ⇨ Utiliser la classe JFrame de javax.swing

2.1] Le JFrame du package SWING :

Le composant **JFrame** est un composant graphique permettant la réalisation d'IHM.
Associé à un **JPanel**, container capable d'accueillir d'autres composants graphiques (JButton, JEdit…) ou non.
Permet de gérer les événements se produisant.

Constitué de 4 couches



Une **JFrame** contient un **JRootPane** qui lui même contient un **JLayeredPane**.
Le **JLayeredPane** organise la fenêtre en 2 composants :

=> la barre de menu **JMenuBar inexistante par défaut**.

=> le ContentPane (JPanel ou Container) sur lequel seront disposés les autres composants graphiques.

Par la suite, un **LayoutManager** sera attribué au **contentPane** afin de gérer la disposition des composants.

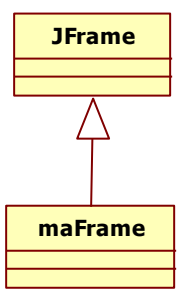
Les constructeurs et méthodes de la classe JFrame :

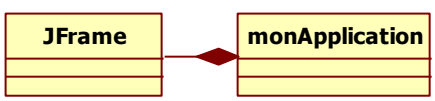
`JFrame()`	crée une fenêtre invisible
`JFrame(String t)`	crée une fenêtre avec un titre = t
`getContentPane()`	retourne le **ContentPane** de la fenêtre
`add(Component c)`	ajoute un composant c sur le ContentPane de la fenêtre
`setDefaultCloseOperation(int)`	spécifie l'action de fermeture par défaut.
`setJMenuBar(JMenuBar menu)`	affecte **menu** comme menu à la fenêtre
`setLayout(LayoutManager lay)`	attribue le **layoutManager** utilisé par le **ContentPane**
`setVisible(boolean)`	
~~`hide()`~~	dépréciée, obsolète ou **deprecated** en anglais
~~`show()`~~	**deprecated**
`setTitle( string t)`	définit et affiche le titre
`setSize( int w , int h)`	définit la taille de la fenêtre, largeur et hauteur en pixel
`setContentPane( Container cp)`	attribue un nouveau ContentPane
`pack()`	la taille et la position sont gérées par la JVM

Création d'une fenêtre JFrame :

Deux possibilités :

- ⇨ Créer une classe qui instancie, configure et utilise un JFrame
- ⇨ Créer une classe qui hérite de la classe JFrame





```
public class Ihm extends JFrame
{
       JButton b1 = new JButton("Valider");
       public Ihm (String titre)
       {
       super(titre);
       …
       }
public static void main(String[] args) {
    Ihm i = new Ihm ("test interface");
    …
       }
```

```
public class ihm1  {

public static void main(String args[])
{
JFrame monCadre = new JFrame("coucou");
monCadre.add(new JButton("Un"));
monCadre.setTitle("un bouton");
monCadre.pack();
monCadre.setVisible(true);
}
}
```

2.2] Gestion de la disposition des composants : les LayoutManager

Les **layoutManager** sont des objets issus de l'interface java.awt.LayoutManager et gère la disposition des composants graphiques sur le **ContentPane**.

Implémentations de java.awt.LayoutManager pour les composants graphiques AWT ou Swing

- **BorderLayout** **(par défaut pour un content pane)**
- **BoxLayout**
- **CardLayout**
- **FlowLayout** **(par défaut pour un JPanel)**
- **GridBagLayout**
- **GridLayout**
- **SpringLayout**

Il est possible de passer outre ces **layoutManager** en le définissant **null**. Dans ce cas, il faut explicitement programmer la taille et la position de chaque composant sur le ContentPane. Ce qui est plus souple mais demande davantage de travail.

Choisir son layout manager :

```
JPanel panel = new JPanel(new BorderLayout());
ou
Container contentPane = frame.getContentPane();
contentPane.setLayout(new FlowLayout());
```

2.2.1] BorderLayout : position par rapport aux bords de la fenêtre. Il existe 5 positions nord ,est ,sud , ouest et centre:

```
public class ihm1 {
public static void main(String[] args) {
 JFrame frame = new JFrame("BorderLayout");
//action par defaut
//frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
JButton b1 = new JButton("nord");
JButton b2 = new JButton("est");
JButton b3 = new JButton("sud");
JButton b4 = new JButton("ouest");
//JButton b5 = new JButton("centre");
JButton b5 = new JButton("-----------centre-------------");
 JPanel p=(JPanel)frame.getContentPane();
 p.add(b1,"North");
 p.add(b2,"East");
 p.add(b3,BorderLayout.SOUTH);
 p.add(b4,"West");
 p.add(b5,"Center");
 frame.pack();
 frame.setVisible(true);
 }
 }
```

⇨ A tester en commentant et décommentant les lignes en commentaires

On remarque que l'on peut disposer un composant de deux manières :

```
p.add(b3, "SOUTH");
p.add(b3,BorderLayout.SOUTH);
```

2.2.2] BoxLayout :

⇨ dispose les composants verticalement ou horizontalement les uns à la suite des autres.

```
public class ihm1 extends JFrame {
       // 6 boutons
       JButton JButton1 = new JButton("1");
       JButton JButton2 = new JButton("2");
       JButton JButton3 = new JButton("3");
       JButton JButton4 = new JButton("quatrième");
       JButton JButton5 = new JButton("5");
       JButton JButton6 = new JButton("6");

public ihm1() {
Container contentpane = getContentPane() ;
setLayout(new BoxLayout(contentpane, BoxLayout.PAGE_AXIS));
       add(JButton1);
       add(JButton6);
       add(JButton5);
       add(JButton4);
       add(JButton3);
       add(JButton2);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       pack() ;
       setVisible(true);
       }
       public static void main(String[] args) {
       ihm1 jf5 = new ihm1();
       }
       }
```

le paramètre permet de définir l'orientation de la disposition :
BoxLayout.X_AXIS
BoxLayout.Y_AXIS
BoxLayout.LINE_AXIS
BoxLayout.PAGE_AXIS

Tester les 4 possibilités.

2.2.3] FlowLayout

LayoutManager par défaut pour les composant JPanel.
Les composants sont placés les uns après les autres sur la même ligne. Passage à la ligne suivante ci nécessaire.

Reprendre l'exemple ci-dessus en remplaçant

```
setLayout(new BoxLayout(contentpane, BoxLayout.PAGE_AXIS));
```

par

```
setLayout(new FlowLayout()) ;
```

2.2.4] CardLayout

Ce layoutManager permet de disposer de plusieurs panel ou un seul est visible. Ce qui ressemble à un jeu de carte où on ne voit que la carte au dessus du paquet.
Il est bien sur possible de choisir le panel que l'on souhaite visualiser.
Le principe est de créer un CardLayout parent puis de lui ajouter des sous-container qui seront les layout enfants. L'ensemble formera un jeu de container.
Chaque sous-container sera nommé.

Quelques méthodes utiles :

void first (Container parent) //affiche le premier sous-container, le parent est le cardlayout principal
void last (Container parent) //affiche le dernier sous-container du jeu.
Dimension minimumlayoutSize(Container target)
void next (Container parent) //affiche le sous-container suivant
Dimension preferredLayoutSize(Container target)

public void previous (Container parent) //affiche le sous-container précédent.

void removeLayoutComponent(Component comp)
public void show (Container parent, String name) //sélectionne le sous-container à afficher

`public class ihm1 extends JFrame implements MouseListener{`	
`static String s1 = "carte 1"; //nom du sous-container`	static car les noms ne doivent pas être modifiés.
`  static String s2 = "carte 2";`	
`Panel Cards;`	Panel parent
`Panel p1;`	
`Panel p2;`	
`JButton b1 ;`	
`JButton b2 ;`	
`JTextField t1 ;`	
`JLabel t2 ;`	
`public ihm1() {`	
`this.setLayout(new FlowLayout());`	
`this.setTitle("fenetre avec CardLaout");`	
`b1 = new JButton("Afficher Card2");`	
`b1.addMouseListener(this);`	
`b2 = new JButton("Afficher Card1");`	
`b2.addMouseListener(this);`	
`t1 = new JTextField("Card1");`	
`t2 = new JLabel("Card2");`	
`Cards = new Panel();`	
`p1 = new Panel();`	
`p2 = new Panel();`	
`Cards.setLayout(new CardLayout());`	création du cardLayout
`p1.add(t1);`	
`p1.add(b1);`	
`Cards.add(s1,p1);`	ajout du 1er sous-container nommé s1
`p2.add(t2);`	
`p2.add(b2);`	
`Cards.add(s2,p2);`	ajout du 2ème sous-container
`add(Cards);`	ajout du CardLayout
`}`	
`public void mouseClicked(MouseEvent arg0) {`	
`/* if (arg0.getSource() == b1)   {`	
`((CardLayout) Cards.getLayout()).show(Cards,s2);`	sélection du sous-container à afficher
`};`	
`if (arg0.getSource() == b2)   {`	
`((CardLayout) Cards.getLayout()).show(Cards,s1);`	idem
`};`	
`*/`	
`((CardLayout) Cards.getLayout()).next(Cards);`	affichage du prochain sous-container
`}`	
`public void mouseEntered(MouseEvent arg0) {  }`	
`public void mouseExited(MouseEvent arg0) {  }`	
`public void mousePressed(MouseEvent arg0) {  }`	
`public void mouseReleased(MouseEvent arg0) { }`	
`static public void main(String[] args)  {`	
`ihm1 f = new ihm1();`	
`f.setVisible(true);`	
`f.pack();`	
`}`	
`}`	

2.2.5] GridBagLayout

Quadrillage flexible, les composants pourront être disposés sur plusieurs lignes et/ou plusieurs colonnes.
Ce quadrillage est flexible car les cases n'ont pas nécessairement les mêmes dimensions.
Il est donc nécessaire d'utiliser conjointement un objet **java.awt.GridBagConstraints** pour définir la taille de chaque cellule.

```
public class ihm1 extends JFrame {
public ihm1() {
JButton button;
GridBagLayout gridbag = new GridBagLayout();
GridBagConstraints c = new GridBagConstraints();
setLayout(gridbag);
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
c.weightx = 0.5 ;
c.weighty = 1 ;
button= new JButton("Button 1");
gridbag.setConstraints(button, c);
add(button);
button = new JButton("2");
c.gridx = 1;
c.gridy = 0;
gridbag.setConstraints(button, c);
add(button);
button = new JButton("Button 3");
c.gridx = 2;
c.gridy = 0;
gridbag.setConstraints(button, c);
add(button);
button = new JButton("beaucoup de texte Button 4");
c.ipady = 40; //Taille verticale = taille verticale minimale + 40 pixels
//c.gridwidth = 3; ou // Largeur du bouton = toute la largeur, ici 3 cellules
c.gridwidth = GridBagConstraints.REMAINDER ;
c.gridx = 0;
c.gridy = 1;
gridbag.setConstraints(button, c);
add(button);
button = new JButton("Button 5");
c.ipady = 0; //Remet à 0 l'allongement vertical du bouton
c.anchor = GridBagConstraints.SOUTH; // en bas de la cellule
c.insets = new Insets(10,0,0,0); // espace avec la cellule supérieure
c.gridx = 1; // aligné avec le 2ème bouton
c.gridwidth = 2; // 2 colonnes de largeur
c.gridy = 2; // 3ème ligne
gridbag.setConstraints(button, c);
add(button);
}
public static void main(String args[]) {
ihm1 window = new ihm1();
window.setTitle("GridBagLayout");
window.pack();
window.setVisible(true);
}
}
```

2.2.6] GridLayout

Position des composants sur une grille de n lignes et m colonnes.

```
public class ihm1 extends JFrame {
       // Grille de 3 lignes et 2 colonnes,
       //espacement de 10 pixels entre les composants
       GridLayout gridLayout1 = new GridLayout(3,2,10,10);
       // 6 boutons
       JButton JButton1 = new JButton();
       JButton JButton2 = new JButton();
       JButton JButton3 = new JButton();
       JButton JButton4 = new JButton();
       JButton JButton5 = new JButton();
       JButton JButton6 = new JButton();
       /**Construire le cadre**/
       public ihm1() {
       setTitle("JFrame avec un grig layout");
       setLayout(gridLayout1);
       JButton1.setText("b1");
       JButton2.setText("b2 ");
       JButton3.setText("b3");
       JButton4.setText("bbbbbbbbbbbbbb4");
       JButton5.setText("b5");
       JButton6.setText("b6");
       add(JButton1);
       add(JButton6);
       add(JButton5);
       add(JButton4);
       add(JButton3);
       add(JButton2);
       pack() ;
       setVisible(true) ;
       }
       /**Méthode principale*/
       public static void main(String[] args)
       {
       new ihm1();
       }
       }
```

2.2.7] SpringLayout

```
public class ihm1 {
        public static void main(String args[]) {
          JFrame frame = new JFrame("SpringLayout");
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          Container contentPane = frame.getContentPane();

          SpringLayout layout = new SpringLayout();
          contentPane.setLayout(layout);

          Component left = new JLabel("Left");
          Component right = new JTextField(15);

          contentPane.add(left);
          contentPane.add(right);
layout.putConstraint(SpringLayout.WEST, left, 10, SpringLayout.WEST, contentPane);
layout.putConstraint(SpringLayout.NORTH, left, 25, SpringLayout.NORTH, contentPane);
layout.putConstraint(SpringLayout.NORTH, right, 25, SpringLayout.NORTH, contentPane);
layout.putConstraint(SpringLayout.WEST, right, 20, SpringLayout.EAST, left);

          frame.setSize(300, 100);
          frame.setVisible(true);
        }
       }
```

2.2.8] Null layout

Il est possible de ne définir aucun layout particulier. Dans il faut indiquer la taille et la position de chaque composants.
Les méthodes à utiliser sont issues des classes mères des composants graphiques et sont :

```
setBounds( Rectangle r )                          //fixe la position et la taille en pixel
setBounds ( int x, int y ,int width , int height )
setSize( int width , int height )                 //fixe la taille
setLocation ( Point p )                           //fixe la position
setLocation ( int x , int y )
```

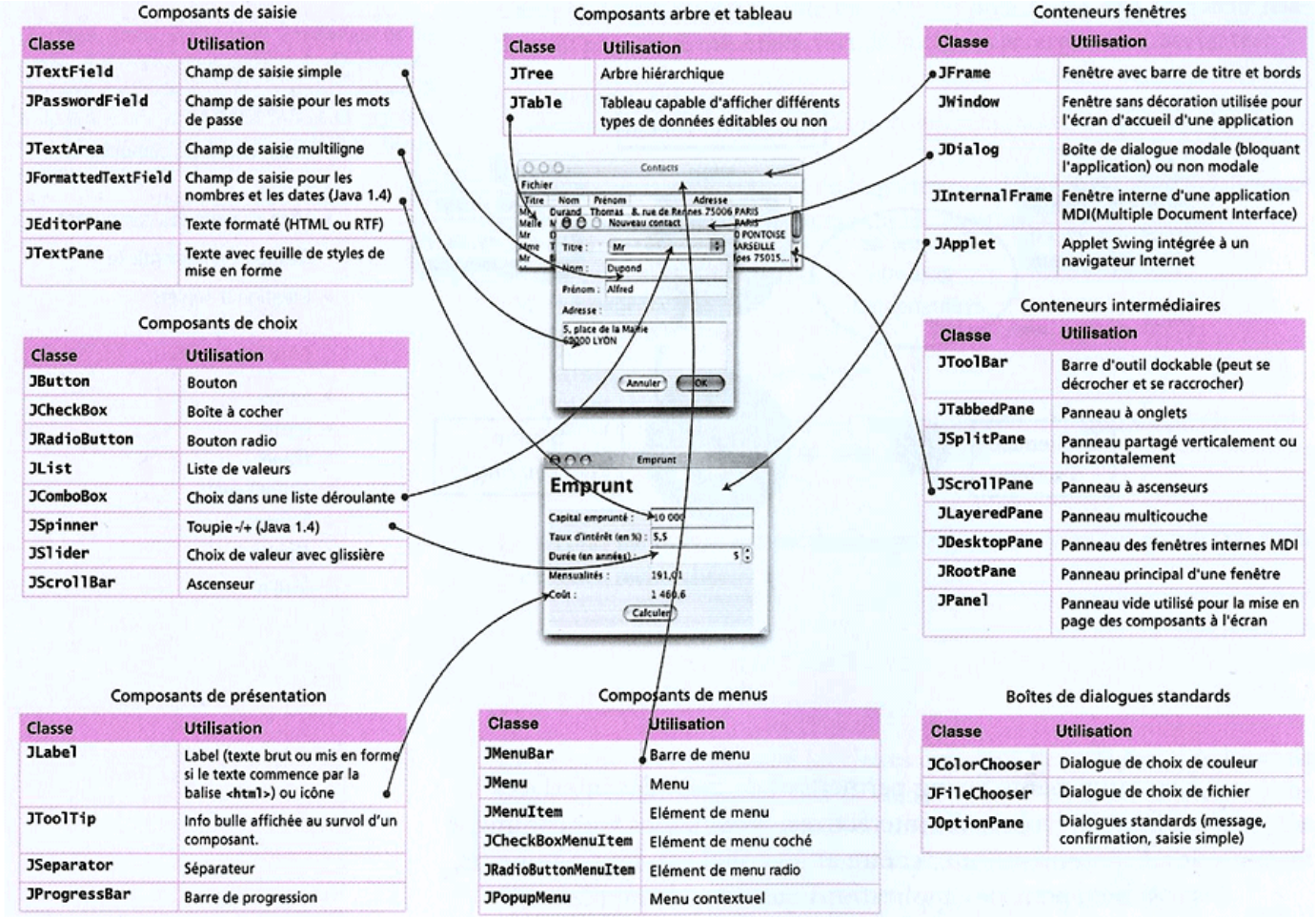
```
public class ihm1 {
        public static void main(String args[]) {
          JFrame frame = new JFrame("SpringLayout");
          frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
          Container contentPane = frame.getContentPane();

          contentPane.setLayout(null);

          Component left = new JLabel("Left");
          //left.setSize(100,30);
          //left.setLocation(10,20);
          left.setBounds(0,0,100,10);
          Component right = new JTextField(15);
          right.setSize(100,30);
          right.setLocation(10,90);

          contentPane.add(left);
          contentPane.add(right);
          frame.setSize(400, 300);
          frame.setVisible(true);
        }
      }
```

2.3] Les composants du package javax.swing



Composants de saisie

Classe	Utilisation
JTextField	Champ de saisie simple
JPasswordField	Champ de saisie pour les mots de passe
JTextArea	Champ de saisie multiligne
JFormattedTextField	Champ de saisie pour les nombres et les dates (Java 1.4)
JEditorPane	Texte formaté (HTML ou RTF)
JTextPane	Texte avec feuille de styles de mise en forme

Composants de choix

Classe	Utilisation
JButton	Bouton
JCheckBox	Boîte à cocher
JRadioButton	Bouton radio
JList	Liste de valeurs
JComboBox	Choix dans une liste déroulante
JSpinner	Toupie -/+ (Java 1.4)
JSlider	Choix de valeur avec glissière
JScrollBar	Ascenseur

Composants arbre et tableau

Classe	Utilisation
JTree	Arbre hiérarchique
JTable	Tableau capable d'afficher différents types de données éditables ou non

Conteneurs fenêtres

Classe	Utilisation
JFrame	Fenêtre avec barre de titre et bords
JWindow	Fenêtre sans décoration utilisée pour l'écran d'accueil d'une application
JDialog	Boîte de dialogue modale (bloquant l'application) ou non modale
JInternalFrame	Fenêtre interne d'une application MDI(Multiple Document Interface)
JApplet	Applet Swing intégrée à un navigateur Internet

Conteneurs intermédiaires

Classe	Utilisation
JToolBar	Barre d'outil dockable (peut se décrocher et se raccrocher)
JTabbedPane	Panneau à onglets
JSplitPane	Panneau partagé verticalement ou horizontalement
JScrollPane	Panneau à ascenseurs
JLayeredPane	Panneau multicouche
JDesktopPane	Panneau des fenêtres internes MDI
JRootPane	Panneau principal d'une fenêtre
JPanel	Panneau vide utilisé pour la mise en page des composants à l'écran

Composants de présentation

Classe	Utilisation
JLabel	Label (texte brut ou mis en forme si le texte commence par la balise <html>) ou icône
JToolTip	Info bulle affichée au survol d'un composant.
JSeparator	Séparateur
JProgressBar	Barre de progression

Composants de menus

Classe	Utilisation
JMenuBar	Barre de menu
JMenu	Menu
JMenuItem	Elément de menu
JCheckBoxMenuItem	Elément de menu coché
JRadioButtonMenuItem	Elément de menu radio
JPopupMenu	Menu contextuel

Boîtes de dialogues standards

Classe	Utilisation
JColorChooser	Dialogue de choix de couleur
JFileChooser	Dialogue de choix de fichier
JOptionPane	Dialogues standards (message, confirmation, saisie simple)

3] Les événements et la programmation événementielle

La programmation événementielle consiste à réaliser une IHM à l'écoute des différents événements pouvant se produire. Lorsqu'un événement est détecté et géré, il peut déclencher un gestionnaire d'événement.
Les événements peuvent être des clicks ou double click, appui sur une touche, le remplissage d'une zone d'édition, l'expiration d'un timer…

En java, pour traiter un événement particulier sur un composant particulier, il faut associer un '**écouteur'** ou **Listener** au composant cible.
Cet écouteur, lorsqu'il détecte un événement déclenche un gestionnaire d'événement en lui passant un argument contenant la source de l'événement (le composant qui reçoit l'événement) et les valeurs particulières liées à l'événement (coordonnées x et y s'il s'agit d'un événement issu de la souris, le code de la touche s'il s'agit d'un appui sur le clavier…).

Java possède de nombreux écouteurs sous forme d'interfaces à implémenter ou des classes abstraites du package (**Adapter**) implémentant ces interfaces.

En résumé :

- ⇨ Un composant qui crée des événements est appelé ***source***.
- ⇨ Le composant source délègue le traitement de l'événement au composant écouteur.
- ⇨ Un composant qui traite un événement est appelé ***écouteur ou*** **listener**
- ⇨ Un composant auditeur doit s'inscrire auprès du composant source des événements qu'il veut traiter.

Ci-dessous, les interfaces à implémenter en fonction des événements les plus utilisés, les classes d'événements associés gérés par le gestionnaire implémenté par les **adapters**.

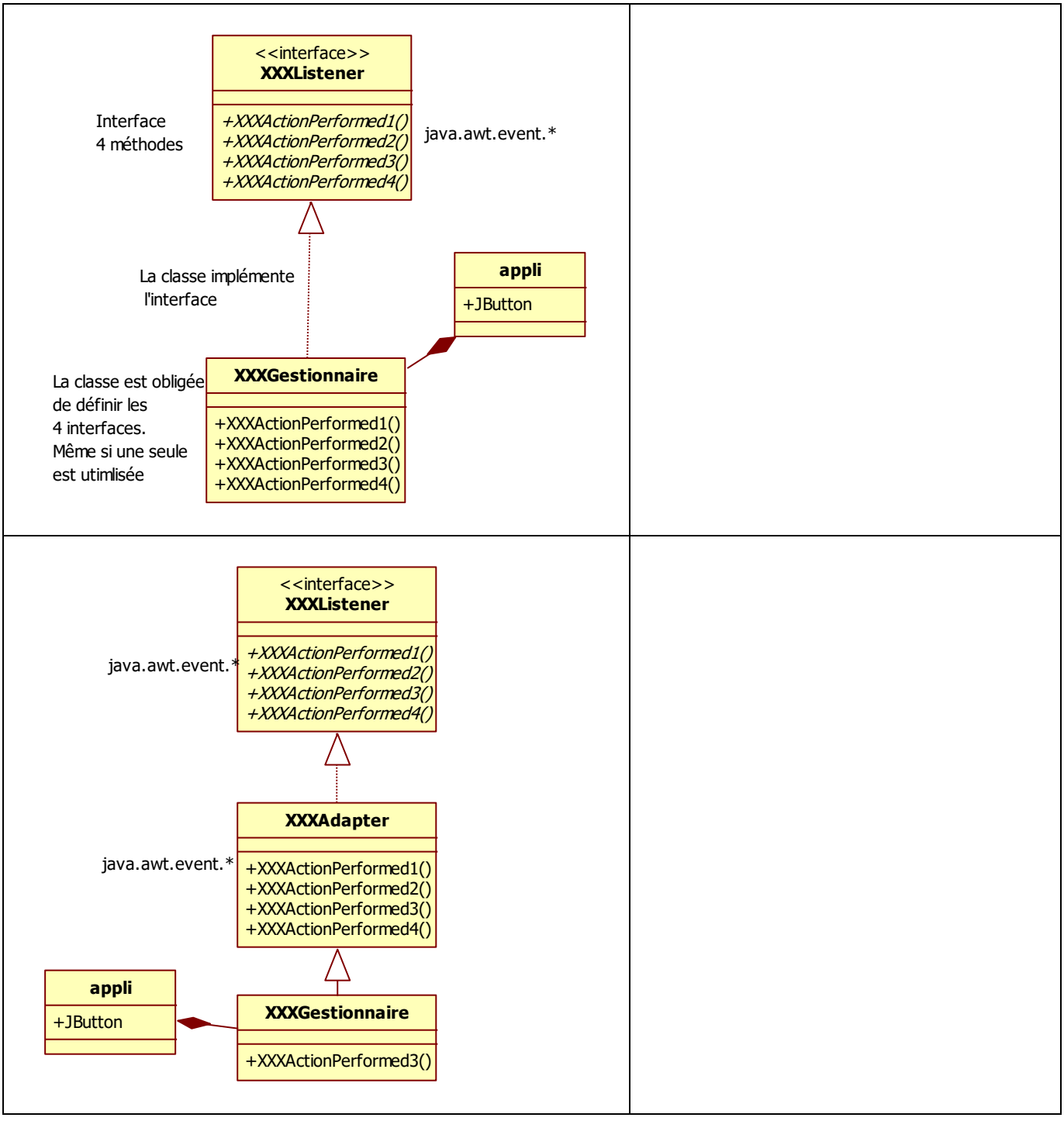
les interfaces	les classes d'objets d'événements	les adapters (classes abstraites)
MouseListener	MouseEvent	MouseAdapter
ActionListener	ActionEvent	
ItemListener	ItemEvent	
KeyListener	KeyEvent	KeyAdapter
WindowListener	WindowEvent	WindowAdapter
WindowFocusListener	''	''
TextListener	TextEvent	

Pour un événement donné **XXX**, il faut :

- ⇨ Créer une classe qui implémente l'interface **XXXListener** correspondante à l'événement.
- ⇨ Ce qui implique de définir toutes les méthodes de l'interface.

Ou

- ⇨ Créer une classe dérivée de la classe **XXXAdapter** qui elle, implémente toutes les méthodes de l'interface
- ⇨ Le corps de ces méthodes est volontairement vide.
- ⇨ Dans la classe dérivée, l'utilisateur ne définira que la méthode nécessaire.



MouseAdapter

void mouseCliked (MouseEvent e)	1 click
void mouseDragged (MouseEvent e)	Bouton souris appuyé et déplacement
void mouseEntered (MouseEvent e)	Curseur présent dans la zone du composant
void mouseExited(MouseEvent e)	Curseur sort de la zone du composant
void mouseMouve(MouseEvent e)	Déplacement de la souris
void mousePressed(MouseEvent e)	Bouton de la souris enfoncé
void mouseRelease (MouseEvent e)	Bouton de la souris relâché

KeyAdapter

void keyPressed(KeyEvent e)	Appui sur le clavier
void keyRelease (KeyEvent e)	La touché appuyée est relâchée
void keyTyped (KeyEvent e)	Touche typée

WindowAdapter

void windowActivate (WindowEvent e)	Fenêtre activée
void windowClosed (WindowEvent e)	La fenêtre a été fermée
void windowClosing (WindowEvent e)	La fenêtre est en train de se fermer
void windowDeactivated (WindowEvent e)	Fenêtre désactivée
void windowDeiconified (WindowEvent e)	
void windowGainedFocus (WindowEvent e)	La fenêtre obtient le focus
void windowIconified (WindowEvent e)	
void windowLostFocus (WindowEvent e)	La fenêtre perd le focus
void windowOpened (WindowEvent e)	
void window StateChanged (WindowEvent e)	

Exemple d'implémentation d'événements :

On propose une IHM composée de d'un label **b2** ,d'une zone de texte **e1** et d'un bouton **b1**.

- ⇨ Lorsque l'on clique sur le bouton, on recopie le texte de **e1** dans le label **b2**.
- ⇨ Lorsque l'on se déplace sur le bouton, on affiche les coordonnées relatives et absolues de la souris.
- ⇨ Lorsqu'on frappe une touche au clavier on l'affiche dans la console.
- ⇨ Si cette touche est un retour chariot, on affiche '**retour chariot'** dans la console.

- ⇨ Implémenter l'interface dans une classe spécifique
- ⇨ Utiliser l'adapter dans une classe spécifique (dérivée de XXXXXAdapter)
- ⇨ Utiliser une classe interne et anonyme.

```
public class ihm1 extends JFrame {
	JButton b1 = new JButton("Valider");
	JLabel b2 = new JLabel("???");
	JTextArea e1 = new JTextArea("entrer le texte");
	click evsouris = new click();
public ihm1()
	{
	super("Titre 1");
	JPanel p=new JPanel(new GridLayout(3,1));
	b1.addMouseListener(evsouris);
	e1.addKeyListener(new KeyAdapter(){
		public void keyPressed(KeyEvent ke)    {
			System.out.println(ke.getKeyChar());
			if(ke.getKeyCode() ==
'\n')System.out.println("retour chariot");
				}
		});
	p.add(b1);
	p.add(e1);
	p.add(b2);
	this.setVisible(true);
	this.setContentPane(p);
	this.setVisible(true);
	this.setSize(250,120);
	}
class click extends MouseAdapter           // (1) ou
//class click implements MouseListener     // (2)
{
public void mouseClicked(MouseEvent me) {
System.out.println(me.toString());
System.out.println("X=" + me.getX() + " y=" + me.getY());
System.out.println("Xabs="+me.getXOnScreen()+" yabs=+
me.getYOnScreen());
String tmp = e1.getText();
b2.setText(tmp);
}
/*	public void mouseEntered(MouseEvent arg0) {
	}
	public void mouseExited(MouseEvent arg0) {
	}
	public void mousePressed(MouseEvent arg0) {
	}
	public void mouseReleased(MouseEvent arg0) {
	}*/
}
	public static void main(String[] args) {
		ihm1 ihmm = new ihm1();
	}
}
```

attachement d'un gestionnaire de saisie via l'adapter.
Classe interne et anonyme

(1) classe gestionnaire dérivée de l'adapter
(2) implémentant le Listener :
si (1) => définir la méthode désirée imposée par l'adapter
si (2) définir toutes les méthodes de l'interface (en commentaire ci-contre)

Voici la liste des interfaces d'écouteurs d'événements tirée du JDK

Action, ActionListener, AdjustmentListener, AncestorListener, AWTEventListener, BeanContextMembershipListener, BeanContextServiceRevokedListener, BeanContextServices, BeanContextServicesListener, CaretListener, CellEditorListener, ChangeListener, ComponentListener, ContainerListener, DocumentListener, DragGestureListener, DragSourceListener, DropTargetListener, FocusListener, HyperlinkListener, InputMethodListener, InternalFrameListener, ItemListener, KeyListener, ListDataListener, ListSelectionListener, MenuDragMouseListener, MenuKeyListener, MenuListener, MouseInputListener, MouseListener, MouseMotionListener, PopupMenuListener, PropertyChangeListener, TableColumnModelListener, TableModelListener, TextListener, TreeExpansionListener, TreeModelListener, TreeSelectionListener, TreeWillExpandListener, UndoableEditListener, VetoableChangeListener, WindowListener.

4] Les menus

La mise en œuvre d'un menu se fait via les objets:

- ⇨ JMenuBar la barre de menu (null par défaut
- ⇨ JMenu titre d'un menu (fichier, edition …)
- ⇨ JMenuItem les choix proposés par le menu
- ⇨ ActionListener écouteur pour les actions sur les items
- ⇨ ActionEvent événements liés aux items des menus.

```
public class menu extends JFrame {
  private JMenuBar menuBar = new JMenuBar();
  private JMenu m1 = new JMenu("Fichier");
  private JMenu m2 = new JMenu("sous menu ");
  private JMenu m3 = new JMenu("Edition");

  private JMenuItem item1 = new JMenuItem("Ouvrir");
  private JMenuItem item2 = new JMenuItem("Fermer");
  private JMenuItem item3 = new JMenuItem("Lancer");
  private JMenuItem item4 = new JMenuItem("Arrêter");

  private JCheckBoxMenuItem jcmi1 = new JCheckBoxMenuItem("Choix 1");
  private JCheckBoxMenuItem jcmi2 = new JCheckBoxMenuItem("Choix 2");

  private JRadioButtonMenuItem jrmi1 = new JRadioButtonMenuItem("Radio 1");
  private JRadioButtonMenuItem jrmi2 = new JRadioButtonMenuItem("Radio 2");

  public static void main(String[] args){
    menu zFen = new menu();
  }

  public menu(){
    this.setSize(400, 200);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.setLocationRelativeTo(null);

    this.m1.add(item1);

    //On ajoute les éléments dans notre sous-menu
    this.m2.add(jcmi1);
    this.m2.add(jcmi2);
    //Ajout d'un séparateur
    this.m2.addSeparator();
    //On met nos radios dans un ButtonGroup
    ButtonGroup bg = new ButtonGroup();
    bg.add(jrmi1);
    bg.add(jrmi1);
    //On présélectionne la première radio
    jrmi1.setSelected(true);

    this.m2.add(jrmi1);
    this.m2.add(jrmi2);

    item4.addActionListener(new ActionListener(){
        public void actionPerformed(ActionEvent arg0) {
                  JDialog d = new JDialog();
                  d.setSize(200, 120);
                  d.setVisible(true);
        }
      });
    this.m1.add(item2);
    //Ajout d'un séparateur
    this.m1.addSeparator();
    //inserer un sous menu
    this.m1.add(this.m2);
    this.m3.add(item3);
```

```
    this.m3.add(item4);

    //L'ordre d'ajout détermine l'ordre d'apparition
    this.menuBar.add(m1);
    this.menuBar.add(m3);
    this.setJMenuBar(menuBar);
    this.setVisible(true);
  }
}
```