MÉMENTO JAVA SWING

Licence 3 INFORMATIQUE – Interface Humain Machine IHM • KILLIAN REINE

Initialisation globale

Initialiser les paramètres généraux de la fenêtre.

- **Titre de la fenêtre**
```
setTitle("Titre");
```
- **Taille de la fenêtre**
```
setSize(600, 400);
```
- **Taille automatique**
```
pack();
```
Définit la taille minimale de la fenêtre pour que tous les composants s'affichent correctement.
- **Centrer la fenêtre**
```
setLocationRelativeTo(null);
```
- **Comportement à la fermeture**
Après avoir délimité la taille.
```
setDefaultCloseOperation(EXIT_ON_CLOSE);
```

Rendre la fenêtre visible

```
myFrame.setVisible(true);
```

Structure de base

Un template de base pour afficher une `Frame`.

```
public class myFrame extends JFrame{
    public myFrame(){
        init();
        setVisible(true);
    }
    public void init(){
        // initialisation des parametres
    }
}
```

Notion de conteneur

Un conteneur est un composant qui peut contenir d'autres composants graphiques (boutons, champs de texte, panneaux, etc.).
Il sert à organiser l'interface graphique et à appliquer un gestionnaire de présentation.

`JFrame`	fenêtre principale
`JPanel`	panneau de composants
`JDialog`	boîte de dialogue

Ajouter un composant à un conteneur

```
monConteneur.add(composant);
```

Gestionnaire de présentation

Associé à un `JPanel`, un gestionnaire de présentation (`LayoutManager`) est un objet qui gère l'organisation et le positionnement des composantes.

Changer le gestionnaire de présentation

```
container.setLayout(...);
```

Les différents gestionnaires de présentation

- `FlowLayout`
Organisé horizontalement. *Si espace insuffisant : retour à la ligne*
`FlowLayout.[LEFT][RIGHT][CENTER][LEADING][TRAILING]`
```
panel.setLayout(
    new FlowLayout(FlowLayout.#)
);
panel.add(...);
```
- `BorderLayout`
Découpe le conteneur en 5 zones.
`BorderLayout.[NORTH][SOUTH][WEST][EAST][CENTER]`
```
panel.setLayout(new BorderLayout());
panel.add(..., BorderLayout.#);
```
- `BoxLayout`
Organise les composants sur un axe horizontal ou vertical.
`BoxLayout.[X_AXIS][Y_AXIS]`
```
panel.setLayout(
    new BoxLayout(panel, BoxLayout.#)
);
panel.add(...);
```
- `CardLayout`
Superpose plusieurs composants comme des cartes.
Une seule carte visible à la fois.
```
panel.setLayout(new CardLayout());
panel.add(panel1, "card1");
panel.add(panel2, "card2");
cl.show(panel, "card1");
```
- `GridLayout`
Disposition en grille régulière (lignes × colonnes).
Tous les composants ont la même taille.
```
panel.setLayout(
    new GridLayout(lignes, colonnes)
);
panel.add(...);
```

Par défaut, un `JPanel` utilise un `FlowLayout` alors que le `JPanel` par défaut d'une `JFrame` utilise un `BorderLayout`.

Imbrication de LayoutManager

Plusieurs gestionnaires de présentation peuvent être utilisés en combinant plusieurs conteneurs. Chaque `JPanel` doit alors avoir son propre `LayoutManager`. Cela permet de créer des interfaces plus complexes.

Idée de l'imbrication :

conteneur principal	→ Layout A
sous-conteneur	→ Layout B
sous-conteneur	→ Layout C

Composant

`Composant(Component)` est un élément graphique d'une interface graphique qui peut être affiché, cliqué ou édité *(boutons, labels, champs, etc.)*.

Étiquettes et champs de saisie

`JLabel`	texte/icône non éditable (affichage)
`JTextField`	saisie texte sur une ligne
`JPasswordField`	saisie masquée (mot de passe)
`JTextArea`	zone texte multi-lignes

Boutons et éléments interactifs

`JButton`	bouton déclenchant une action
`JCheckBox`	choix multiple (true/false)
`JRadioButton`	choix unique dans un groupe

Remarque sur `JRadioButton`

Dans un groupe de bouton radio **un seul et unique bouton** peut être coché à la fois. On utilise alors un objet `ButtonGroup` pour satisfaire cette propriété.
Il faut donc ajouter les boutons au `Container` mais aussi au `ButtonGroup`.

```
JButton bt1 = new JButton("...");
JButton bt2 = new JButton("...");
ButtonGroup bg = new ButtonGroup();
bg.add(bt1); bg.add(bt2);
panel.add(bt1); panel.add(bt2);
```

Listes et choix multiples

`JList`	liste sélection simple/multiple
`JComboBox`	liste déroulante

Création `JList`

```
// (1) ajout un par un
JList<String> jls = new JList<>();
jls.add(...);
// (2) avec un tableau
T[] tab = {...};
JList<T> jlsT = new JList<>(tab);
```

Création `JComboBox` comme `JList`.

Composants de conteneur

`JPanel`	conteneur générique
`JScrollPane`	ajoute des barres de défilement
`JTabbedPane`	interface par onglets
`JSplitPane`	séparation ajustable

Création JScrollPane

```
JScrollPane sp = new JScrollPane(component);
```

Création JTabbedPane

```
JTabbedPane tabs = new JTabbedPane();
tabs.add("Onglet1", panel1);
tabs.add("Onglet2", panel2);
```

Création JSplitPane

```
JSplitPane split =
 new JSplitPane(
  JSplitPane.HORIZONTAL_SPLIT,
  panel1, panel2);
```

`JSplitPane.[HORIZONTAL_SPLIT][VERTICAL_SPLIT]`

Dialogues et fenêtres secondaires

`JOptionPane`	message, saisie, confirmation
`JDialog`	fenêtre secondaire personnalisée

Utiliser une boîte de dialogue

```
JDialog dlg = new JDialog(parent, "Titre");
dlg.setSize(...,...);
dlg.setModal([true][false]);
dlg.setVisible([true][false]);
```

Utiliser `OptionPane` – Saisie utilisateur

```
String saisie =
    JOptionPane.showInputDialog(
        null,
        " crit   qqch"
    );
JOptionPane.showMessageDialog(
        null,
        saisie
    );
```

ici, on affiche ce que l'utilisateur a écrit.

Utiliser `OptionPane` – Confirmer une action

```
// Affichage de la boite
int reponse =
    JOptionPane.showConfirmDialog(
        null,
        "Veux-tu continuer ?",
        "Confirmation",
        JOptionPane.YES_NO_CANCEL_OPTION
    );
// Traitement de la reponse
String message = "";
if(reponse == JOptionPane.YES_OPTION){
    message ="Tu as clique Oui !";
} else if(reponse == JOptionPane.NO_OPTION){
    message = "Tu as clique Non !";
} else if(reponse == JOptionPane.
     CANCEL_OPTION){
    message = "Tu as clique Annuler !";
} else {
    message = "Tu as ferme la boite";
}
JOptionPane.showMessageDialog(null, message);
```

Utiliser `OptionPane` – Liste de choix

```
String[] options = {...};

String choix =
    (String) JOptionPane.showInputDialog(
    null,
    "Choisis une option :",
    "Selection",
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[0]
);

String message = "";
if (choix != null) {
    message = choix;
} else {
    message = "Aucun choix selectionne !";
}
JOptionPane.showMessageDialog(
    null,
    "Choix : " + message);
```

Les actions

Une **action** est une opération déclenchée par l'utilisateur.

Mise en place `ActionListener`

- Implémenter `ActionListener` **à ne pas faire !!!**
```
// dans une methode
monBouton.addActionListener(this);
@Override
public void actionPerformed(ActionEvent e)
{
    // code gestion action
}
```
On met `this` car l'action est gérée dans la même classe. Si on gère l'action dans une autre classe `actionClass` nous aurions écrit : `monBouton.addActionListener(new actionClass())`.
- Classe anonyme **Déconseillé depuis Java8**
```
monBouton.addActionListener(
    new ActionListener(){
        @Override
        public void actionPerformed(...){
            // code gestion action
        }
    }
);
```
- Avec une expression lambda
```
monBouton.addActionListener(
    e -> // code gestion action
);
```

Classe externe (ou interne) avec `AbstractAction`

```
MonAction action = new MonAction(...);
// On associe l'action au bouton
bouton = new JButton(action);

class MonAction extends AbstractAction{
    // constructeur si besoin

    @Override
    public void actionPerformed(...){
        // code gestion action
    }
}
```

Les événements

Un **événement** est le **signal généré par le composant Swing**. Les événements sont capturés par des écouteurs.

`ActionListener`	Clic sur bouton
`ItemListener`	Sélection de case à cocher
`DocumentListener`	Modification d'un champ texte
`MouseMotionListener`	Mouvement de souris
`MouseListener`	Clic + entrée/sortie dans un composant
`KeyListener`	Touche du clavier pressée
`FocusListener`	Changement de focus
`ComponentListener`	Redimensionnement/mouvement de fenêtre
`ListSelectionListener`	Changement dans une liste ou combo box
`MenuListener`	Changement dans un menu

Pour gérer un **événement** en Swing, il faut : choisir le composant, sélectionner le Listener approprié, l'implémenter, puis l'attacher au composant. Veiller à ce que le code exécuté reste rapide pour ne pas bloquer l'interface.

Mettre en place un écouteur

Patron général (classe externe)

```
// Dans le conteneur
composant.addXxxListener(
    new MonEcouteur(this)
);

// Classe externe
class MonEcouteur implements XxxListener {
    private MaFenetre fenetre;

    public MonEcouteur(MaFenetre f){
        this.fenetre = f;
    }
    @Override
    public void xxxPerformed(XxxEvent e){
        // acces aux composants via fenetre
    }
}
```

Identifier la source de l'événement — Déconseillé !

```
@Override
public void actionPerformed(ActionEvent e){
    Object src = e.getSource();
    if(src == bouton1){ ... }
    else if(src == bouton2){ ... }
}
```

Tableau des méthodes à implémenter

MouseListener	mouseClicked(e) mouseReleased(e) mouseExited(e)	mousePressed(e) mouseEntered(e)
KeyListener	keyPressed(e) keyTyped(e)	keyReleased(e)
FocusListener	focusGained(e)	focusLost(e)
ActionListener	actionPerformed(e)	
ItemListener	itemStateChanged(e)	
ListSelection...	valueChanged(e)	

■ *plusieurs méthodes → utiliser un* `Adapter`
■ *une seule méthode*

```
composant.addMouseListener( new XxxAdapter(){
        @Override
        public void ...(XxxEvent e){
            // seulement ce dont on a besoin
        }
    }
);
```

L'exemple ci dessus, est à éviter ! L'utilisation d'un `Adapter` évite de devoir implémenter toutes les méthodes du `Listener`. Seulement celles dont on a besoin.

Menus

Hiérarchie des éléments

JMenuBar	barre de menus
JMenu	menu déroulant
JMenuItem	item cliquable
JCheckBoxMenuItem	item à cocher
JRadioButtonMenuItem	item radio

Construction

```
JMenuBar barre = new JMenuBar();
JMenu menu1 = new JMenu("menu1");
JMenu menu2 = new JMenu("menu2");
JMenuItem item = new JMenuItem("sousmenu");
JMenuItem item2 = new JMenuItem("sousmenu2");
...
item.addActionListener(...);
menu1.add(item);     // ajout au menu
menu2.add(item2);
barre.add(menu1); barre.add(menu2);
setJMenuBar(barre);
```

Étape de construction d'un menu

1. Création de la barre de menu
2. Création d'un menu dépliant
3. Création des items
4. Affectation items aux menus déroulant
5. Affectation des menus déroulants à la barre de menu

Raccourcis clavier

Code mnémonique *(Alt + touche, menu ouvert)*

```
// Sur un JMenu ou JMenuItem
menu.setMnemonic(KeyEvent.VK_F);
item.setMnemonic(KeyEvent.VK_O);
```

Souligne la lettre correspondante dans le libellé.
Fonctionne sur : `JMenu`, `JMenuItem`, `JButton`, `JCheckBox`, `JRadioButton`

Accélérateur *(combinaison directe, sans ouvrir)*

```
// Sur un JMenuItem uniquement
item.setAccelerator(
    KeyStroke.getKeyStroke(
        KeyEvent.VK_S,
        InputEvent.CTRL_DOWN_MASK
    )
);
```

Modificateurs disponibles

CTRL_DOWN_MASK	touche Ctrl
SHIFT_DOWN_MASK	touche Shift
ALT_DOWN_MASK	touche Alt
META_DOWN_MASK	touche Meta/Cmd

Combinaison avec | :

```
KeyStroke.getKeyStroke( KeyEvent.VK_G,
    InputEvent.CTRL_DOWN_MASK |
    InputEvent.SHIFT_DOWN_MASK
);
// ou KeyStroke.getKeyStroke("ctrl shift G");
```

Conteneurs avancés

JDesktopPane + JInternalFrame – Bureau virtuel

Un bureau virtuel permet de gérer plusieurs **sous-fenêtres** à l'intérieur d'une seule fenêtre principale, comme un gestionnaire de fenêtres embarqué.

```
JDesktopPane bureau = new JDesktopPane();
setContentPane(bureau);
JInternalFrame f = new JInternalFrame(
    "Titre", true, true, true, true
);
f.add(...);
f.pack(); // ou setSize
f.setVisible(true);
bureau.add(f);
```

Un `JInternalFrame` s'utilise comme une `JFrame` classique : on peut y ajouter une `JMenuBar`, des composants, des layouts...

JToolBar – Barre d'outils

Une `JToolBar` est une barre d'outils contenant des boutons d'accès rapide aux actions fréquentes. Elle doit obligatoirement être placée dans un conteneur avec un `BorderLayout`.

```
JToolBar palette = new JToolBar("Palette");
palette.add(new JButton("b1"));
palette.add(new JButton("b2"));
palette.addSeparator();
add(palette, BorderLayout.NORTH);
add(contenu, BorderLayout.CENTER);
```

setFloatable(false)	palette non déplaçable
setRollover(true)	survol mis en évidence
setBorderPainted(true)	affiche le cadre

Par défaut, la `JToolBar` est déplaçable : l'utilisateur peut la glisser vers `SOUTH`, `EAST`, `WEST` ou la détacher en fenêtre flottante.

Aide au survol (Tooltip)

Bulle d'aide affichée au survol d'un composant. Fonctionne sur tous les composants Swing.

```
component.setToolTipText("Aide ici");
```

Internationalisation – Ressources

Séparer les ressources (textes, images...) du code pour adapter le logiciel à différentes langues sans recompiler.

Locale – Spécifier une localisation

```
Locale ici = new Locale("fr", "FR");
// ou Locale ici = Locale.FRANCE;
Locale us = new Locale("en", "US");
Locale.setDefault(ici); // par defaut
```

ResourceBundle – Charger les ressources

```
ResourceBundle res =
    ResourceBundle.getBundle(
        "mesTextes",   // nomFichier
        ici            // locale utilisee
    );
String texte = res.getString("cle");
```

Fichier `.properties` dans le dossier `./resources`

```
cle=valeur
// pour en : cle=value
```

mesTextes_fr_FR.properties	français France
mesTextes_fr.properties	français général
mesTextes_en_US.properties	anglais USA
mesTextes_en.properties	anglais général
mesTextes.properties	version par défaut

Toujours prévoir une version non suffixée pour éviter une exception.

Ergonomie – Principes clés

Charte de couleurs

- Limiter à **7 ± 2** couleurs ; assigner un rôle précis à chacune
- Fond : couleur peu saturée (≈ gris) — fatigue peu, neutre
- Éviter couleurs trop proches ou trop soutenues en fond

Charte de polices

- Limiter à **3 polices** ; utiliser du `sans-serif`
- Texte : minuscules, initiale en majuscule ; pas de justification totale
- Lignes : 30–80 caractères selon mise en page

Disposition & boîtes de dialogue

- Regrouper par thème ; éléments principaux en haut, boutons en bas à droite
- Espacement basé sur des multiples de **6 px**
- Largeur uniforme pour tous les boutons d'une même boîte
- Préférer plusieurs fenêtres simples à une seule fenêtre complexe

Menus

- Ordre standard : *Fichier / Édition / Format / Affichage / Aide*
- Max **10** items (débutant) ou **20** (expert) ; 1 niveau d'imbrication max
- Griser les options indisponibles (ne pas les masquer)
- Séparer les items destructeurs (*Supprimer*, etc.)

Attente & erreurs

$t < 2s$	rien
$2s \leq t \leq 6s$	curseur d'attente
$t > 6s$	barre de progression

- Message d'erreur : décrire le problème **et** le moyen de le corriger
- Toujours autoriser le **retour arrière**

Chargement d'un fichier

Ouvrir / Enregistrer

```
JFileChooser fc = new JFileChooser();
int res = fc.showOpenDialog(parent);
int res = fc.showSaveDialog(parent);

if(res == JFileChooser.APPROVE_OPTION){
    File f = fc.getSelectedFile();
    String chemin = f.getAbsolutePath();
}
```

Options utiles

setCurrentDirectory(new File("."))	dossier initial
setMultiSelectionEnabled(true)	sélection multiple
getSelectedFiles()	tableau de fichiers

Filtrer les fichiers

```
fc.setFileFilter(
  new FileNameExtensionFilter(
    "Images (*.png, *.jpg)",
    "png", "jpg"
  )
);
```

Personnalisation des composants

Couleurs & police

```
// couleur de fond
component.setBackground(Color.LIGHT_GRAY);
// couleur du texte
component.setForeground(Color.DARK_GRAY);
// Police de caracteres
component.setFont(
    new Font("Dialog", Font.BOLD, 12)
);
// Activer le fond sur un JLabel/JPanel
component.setOpaque(true);
```

Taille & positionnement

```
component.setPreferredSize(
    new Dimension(120, 30)
);
component.setMinimumSize(
    new Dimension(80, 20)
);
component.setMaximumSize(
    new Dimension(200, 40)
);
component.setAlignmentX(
    Component.CENTER_ALIGNMENT
);
```

Bordure

```
// Bordure simple
component.setBorder(
    BorderFactory.createLineBorder(
        Color.GRAY,
        1
    )
);
// Bordure double
component.setBorder(
    BorderFactory.createCompoundBorder(
        BorderFactory.createLineBorder(
            Color.BLUE, 2
        ),
        BorderFactory.createLineBorder(
            Color.RED, 2
        )
    )
); // (exter, inter)
// Bordure avec titre
component.setBorder(
    BorderFactory.createTitledBorder("Titre")
);
// Marge interieure (H, G, B, D)
component.setBorder(
    BorderFactory.createEmptyBorder(5,5,5,5)
);
```

État du composant

```
comp.setEnabled(false);   // griser
comp.setVisible(false);   // masquer
comp.setFocusable(true); // peut etre focus
comp.requestFocusInWindow(); // donner focus
```

Curseur souris

```
component.setCursor(
    Cursor.getPredefinedCursor(
        Cursor.HAND_CURSOR
    )
);
```

DEFAULT_CURSOR	curseur normal
HAND_CURSOR	main
WAIT_CURSOR	sablier
CROSSHAIR_CURSOR	croix
TEXT_CURSOR	texte (I)