ISI5
Développement d'interfaces Homme-Machine

Manuele Kirsch Pinheiro
Maître de conférences en Informatique
Centre de Recherche en Informatique
Université Paris 1 – Panthéon Sorbonne
Manuele.Kirsch-Pinheiro@univ-paris1.fr
http://mkirschp.free.fr

Présentation

- **Contenu prévisionnel**
  - Gestionnaires de placement (*Layout Managers*)
  - Java Swing
  - Révision dernier cours
    - Programmation événementiel avec AWT
    - Architecture 1 et 2 couches

Gestionnaires de placement
LayoutManager

Gestionnaires de placement

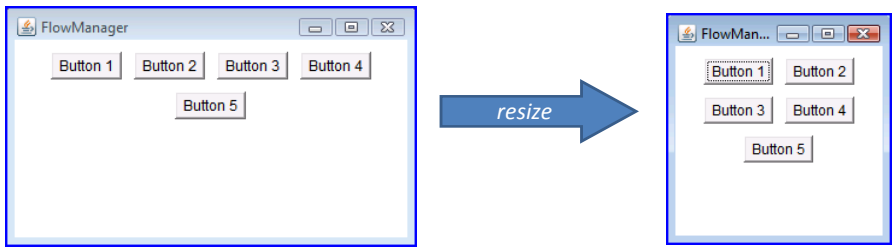
- Interface *java.awt.**LayoutManager***
- Objectif :
  - Gérer l'**emplacement** des composants dans un container
- **Positionner** les **composants** d'interface graphique
  - Calcul automatique de la position
  - En cas de redimensionnement, le repositionnement des composants est automatique
  - *Pas besoin d'affecter des positions précises aux composants*

Gestionnaires de placement

- Chaque container possède un **LayoutManager**
  - Méthode **setLayout**(LayoutManager) de **Container**
- Différents implémentations, dont
  - *FlowLayout : présentation en ligne*
  - *BorderLayout : présentation en blocs nord, sud…*
  - *CardLayout : présentation en pile*
  - *GridLayout : présentation en grille*
  - *GridBagLayout: présentation en grille composite*
  - …

FlowLayout

- Le plus simple
- Organisation dans une ligne
  - Gauche → droite, droite → gauche
  - Centralisée ou pas



Exemple

```
public class FlowLayoutExampleFrame extends Frame {

    public FlowLayoutExampleFrame(String title) {
        super(title);
        initComponents();
    }

    protected void initComponents () {
        this.addWindowListener(new ExternalWindowListner ());

        /* on definit le layoutmanager centralise, avec 2x plus de espace
        vertical et horizontal entre les composants */
        FlowLayout flow = new FlowLayout(FlowLayout.CENTER);
        flow.setHgap(flow.getHgap() * 2);
        flow.setVgap(flow.getVgap() * 2);

        this.setLayout(flow);

        //on ajoute 5 boutons
        for (int i=1; i<=5; i++) {
            Button b = new Button("Button "+i);
            this.add(b);
        }
        this.setSize(this.guessSize());
    }
```

FlowLayout centralisé
Définition des espaces entre les composants :
- Espace horizontal (hgap)
- Espace vertical (vgap)

Un Frame avec 5 buttons

Exemple

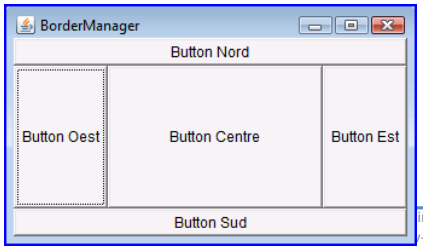
```
    protected Dimension guessSize () {
        Dimension d = null;

        //on recupere un lien vers le gestionnaire
        Toolkit kit = Toolkit.getDefaultToolkit();
        d = kit.getScreenSize();

        double wdt = d.getWidth() / 4;
        double hgt = d.getHeight() / 4;

        //taille mminimale WIDTH, HEIGHT
        if (wdt < WIDTH) wdt = WIDTH;
        if (hgt < HEIGHT) hgt = HEIGHT;

        d.setSize(wdt, hgt);
        return (d);
    }

    public static void main(String[] args) {
        FlowLayoutExampleFrame frame = new FlowLayoutExampleFrame ("FlowManager");
        frame.setVisible(true);
    }

    static final int WIDTH = 300;
    static final int HEIGHT = 200;

}
```

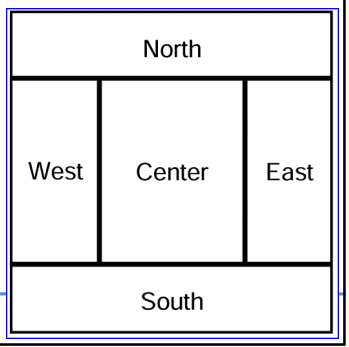
Taille du Frame calculé en fonction de taille de l'écran

BorderLayout

- Layout par défaut
- On choisit dans que région le composant est attachée
  - CENTER, NORTH, SOUTH, EAST, WEST
  - Layout composé par de sous-panels

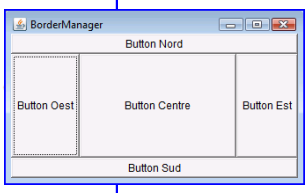
Source : Core Java ®





Exemple

```
public class BorderLayoutExampleFrame  extends FlowLayoutExampleFrame {
    public BorderLayoutExampleFrame(String title) {
        super(title);
    }

    @Override
    protected void initComponents () {
        this.addWindowListener(new ExternalWindowListner ());

        /* on definit le layoutmanager avec 2x plus d'espace vert.
        BorderLayout border = new BorderLayout();
        border.setHgap(border.getHgap() * 2);
        border.setVgap(border.getVgap() * 2);
        this.setLayout(border);

        //on ajoute 5 boutons
        Button b = new Button("Button Nord");
        this.add(b, BorderLayout.NORTH);
        b = new Button("Button Oest");
        this.add(b, BorderLayout.WEST);
        b = new Button("Button Est");
        this.add(b, BorderLayout.EAST);
        b = new Button("Button Sud");
        this.add(b, BorderLayout.SOUTH);
        b = new Button("Button Centre");
        this.add(b, BorderLayout.CENTER);

        this.setSize(this.guessSize());
    }
```



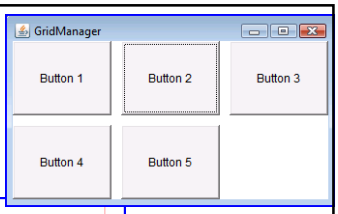
→Chaque composant est attaché à une zone

→S'il n'y a pas d'autres composants, il l'occupe entièrement

GridLayout

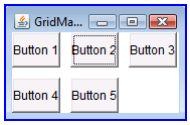
- Organisation des composants en **lignes** et **colonnes**
  - Grille rectangulaire composé par des cellules identiques
  - Remplissage droite → gauche ou droite → gauche selon propriété ComponentOrientation du container
- Composants avec une **taille identique**

Exemple




```
public class GridLayoutExampleFrame extends FlowLayoutExampleFrame {

    public GridLayoutExampleFrame(String title) {...}

    @Override
        protected void initComponents () {
        this.addWindowListener(new ExternalWindowListner ());

        /* on definit une grille de 2x3, avec 10pts entre les cellules */
        GridLayout grid = new GridLayout(2,3, 10, 10);
        this.setLayout(grid);

        //on ajoute 5 boutons
        for (int i=1; i<=5; i++) {
            Button b = new Button("Button "+i);
            this.add(b);
        }
        this.setSize(this.guessSize());
    }

    /**...*/
    public static void main(String[] args) {
        GridLayoutExampleFrame frame = new GridLayoutExampleFrame ("GridManager");
        frame.setVisible(true);
    }
}
```

Resize garde la grille



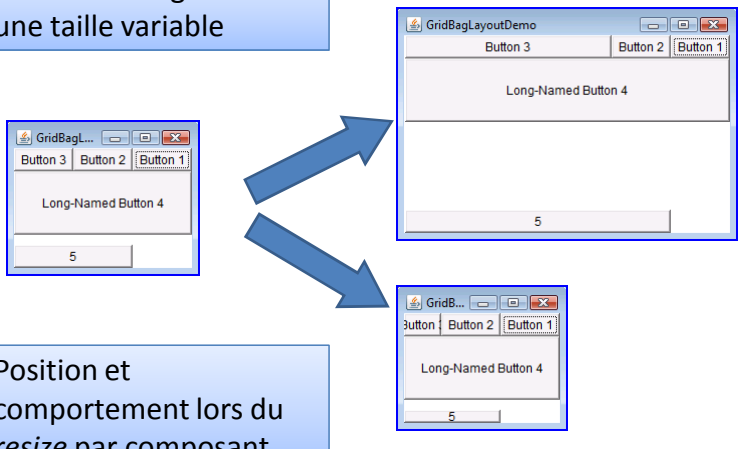
→Grille de 2 lignes et 3 colonnes
→Composants disposés automatiquement

GridBagLayout

- Disposition des composants également en lignes et colonnes
- Cellules de **taille variable**
  - Composants peuvent s'étendre sur plusieurs cellules
- Définition des contraintes (**GridBagConstraints**)
  - Chaque composant est associé à un GridBagConstraints
  - Les contraintes définissent le positionnement du composant

Exemple

Cellules de la grille ont une taille variable



Position et comportement lors du *resize* par composant

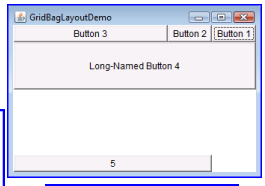
GridBagConstraints

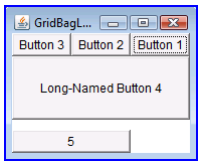
- Différentes contraintes sont possibles
  - Emplacement dans la grille :
    - *gridx* (colonne), *gridy* (ligne)
    - c.gridx = 1; c.gridy = 0; //2nd colonne, 1er ligne
  - Remplissage des cellules adjacentes :
    - *gridwidth* (n° de colonnes occupées)
    - *gridheight* (n° de lignes occupées)
    - c.gridwidth = 3;
    - c.gridwidth = GridBagConstraints.REMAINDER; //dernier de la col.
    - c.gridheight = GridBagConstraints.RELATIVE; //après le dernier
  - Remplissage d'une zone (fill):
    - NONE (aucun), HORIZONTAL, VERTICAL, BOTH
    - c.fill = GridBagConstraints.HORIZONTAL;

GridBagConstraints

  - Remplissage interne :
    - ipadx (horizontal), ipady (vertical)
    - c.ipady = 40; //composant haut
  - Poids :
    - weightx (0 si la colonne doit garder sa taille)
    - weighty (0 si la ligne doit garde sa taille)
    - c.weightx = 0.0; c.weightx = 100;
  - Situation du composant dans la zone (anchor)
    - CENTER, NORTH, NORTHEAST, EAST, PAGE_END…
    - c.anchor = GridBagConstraints.PAGE_END;

Exemple






```
public static void addComponentsToPane(Container pane) {
    pane.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);

    Button button;
    pane.setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();

    button = new Button("Button 1");
    c.fill = GridBagConstraints.HORIZONTAL;
    c.gridx = 0;
    c.gridy = 0;
    pane.add(button, c);

    button = new Button("Button 2");
    c.fill = GridBagConstraints.HORIZONTAL;
    c.weightx = 0.5;
    c.gridx = 1;
    c.gridy = 0;
    pane.add(button, c);

    button = new Button("Button 3");
    c.fill = GridBagConstraints.HORIZONTAL;
    c.weightx = 100;
    c.gridx = 2;
    c.gridy = 0;
    pane.add(button, c);
```

```
    button = new Button("Long-Named Button 4");
    c.fill = GridBagConstraints.HORIZONTAL;
    c.ipady = 40;      //make this component tall
    c.weightx = 0.0;
    c.gridwidth = 3;
    c.gridx = 0;
    c.gridy = 1;
    pane.add(button, c);

    button = new Button("5");
    c.fill = GridBagConstraints.HORIZONTAL;
    c.ipady = 0;       //reset to default
    c.weighty = 1.0;   //request any extra vertical space
    c.anchor = GridBagConstraints.PAGE_END; //bottom of space
    c.insets = new Insets(10, 0, 0, 0);  //top padding
    c.gridx = 1;       //aligned with button 2
    c.gridwidth = 2;   //2 columns wide
    c.gridy = 2;       //third row
    pane.add(button, c);
}
```

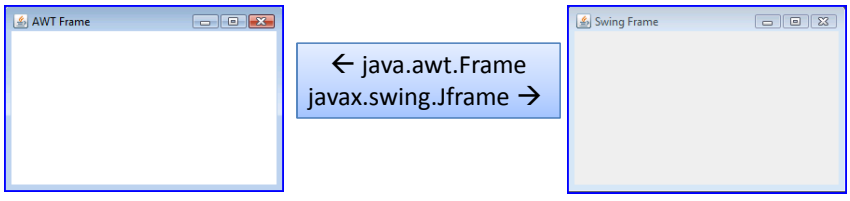

Swing

Swing

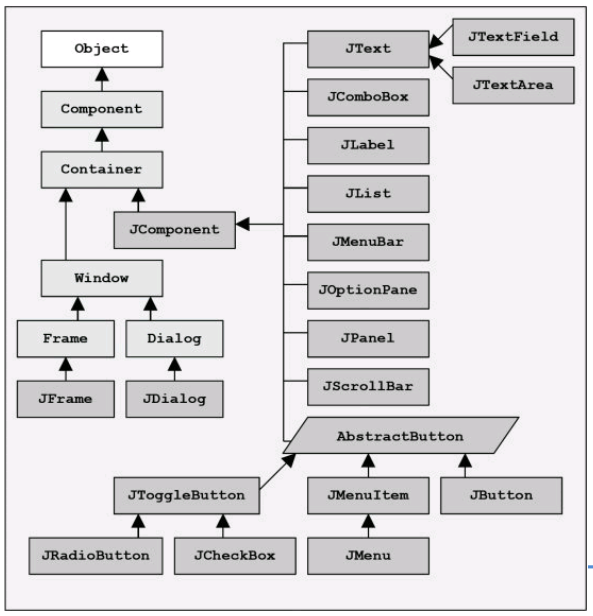
- Développé en collaboration avec Netscape
- Fondé sur AWT
  - Même mode de traitement des événements
- Implémentation Java, non native
  - Peu de dépendances par rapport à la plateforme
- Implémentation basée sur le modèle MVC
- *Look & Feel* configurable
- Ensemble de composants étendu

Swing

- Organisation similaire à AWT
  - Notion de composant (**JComponent**) et de container (**JFrame**)
- Package javax.swing.*
  - Composants se nomment ***J****



Hiérarchie Swing



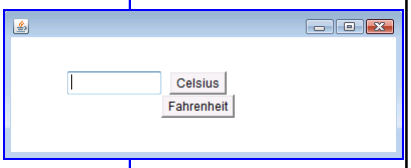
Source : C.S. Lindsey, *"JavaTech: An Introduction to Scientific and Technical Computing with Java"*

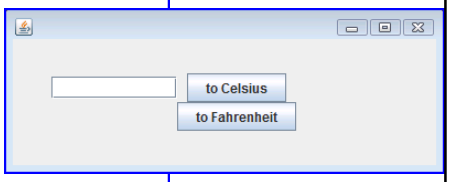
http://www.particle.kth.se/~lindsey/JavaCourse/Book/Part1/Java/Chapter06/swing.html

21

AWT x Swing

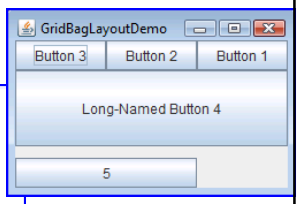
```
public class TempConverterSwingGUIv2 extends javax.swing.JFrame {

    private javax.swing.JButton optCelsius;
    private javax.swing.JButton optFahrenheit;
    private javax.swing.JLabel resultat;
    private javax.swing.JPanel panel1;
    private javax.swing.JTextField textField1;

    /**...*/
    public TempConverterSwingGUIv2() {...}

    private void initComponents() {

        panel1 = new javax.swing.JPanel();
        textField1 = new javax.swing.JTextField();
        optCelsius = new javax.swing.JButton();
        optFahrenheit = new javax.swing.JButton();
        resultat = new javax.swing.JLabel();

        addWindowListener(new java.awt.event.WindowAdapter() {...});

        java.awt.GridBagConstraints gridBagConstraints;

        panel1.setLayout(new java.awt.BorderLayout());

        textField1.setText("");
        textField1.setColumns(10);

        optCelsius.setText("to Celsius");
        optFahrenheit.setText("to Fahrenheit");
```





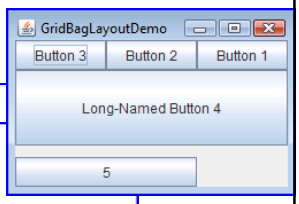
22

AWT x Swing




```
public static void addComponentsToPane(Container pane) {
    pane.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);

    JButton button;
    pane.setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();

    button = new JButton("Button 1");
    c.fill = GridBagConstraints.HORIZONTAL;
    c.gridx = 0;
    c.gridy = 0;
    pane.add(button, c);
```

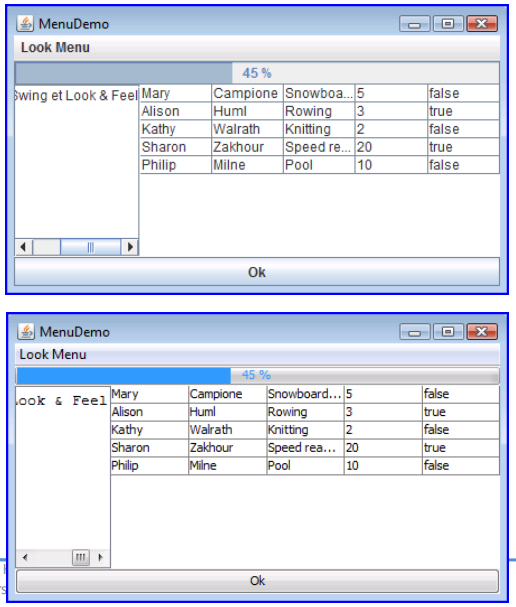

AWT x Swing




```
public static void addComponentsToPane(Container pane) {
    pane.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
private static void createAndShowGUI() {
    //Create and set up the window.
    JFrame frame = new JFrame("GridBagLayoutDemo");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    //Set up the content pane.
    addComponentsToPane(frame.getContentPane());

    //Display the window.
    frame.pack();
    frame.setVisible(true);
}

public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
    } catch (UnsupportedLookAndFeelException ex) {
        ex.printStackTrace();
    } catch (IllegalAccessException ex) {
        ex.printStackTrace();
    } catch (InstantiationException ex) {
        ex.printStackTrace();
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    }
    /* Turn off metal's use of bold fonts */
    UIManager.put("swing.boldMetal", Boolean.FALSE);
```

Quelques composants

- Composants :
  - JButton
  - JTextArea
  - JMenu
  - JTable
  - JProgressBar
- Containers :
  - JFrame
  - JPanel
  - JScrollPane



Création d'un menu…

```
public JMenuBar createMenuBar() {
    JMenuBar menuBar;
    JMenu menu;
    JMenuItem menuItem;

    //Create the menu bar.
    menuBar = new JMenuBar();

    //Build the first menu.
    menu = new JMenu("Look Menu");
    menu.setMnemonic(KeyEvent.VK_A);
    menu.getAccessibleContext().setAccessibleDescription(
            "The only menu in this program that has menu items");
    menuBar.add(menu);

    //a group of JMenuItems
    menuItem = new JMenuItem("Metal", KeyEvent.VK_M);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_1, ActionEvent.ALT_MASK));
    menuItem.getAccessibleContext().setAccessibleDescription("Set the Java (Metal) Look & Feel");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    menuItem = new JMenuItem("Native", KeyEvent.VK_N);
    menuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_2, ActionEvent.ALT_MASK));
    menuItem.getAccessibleContext().setAccessibleDescription("Set the Native Look & Feel");
    menuItem.addActionListener(this);
    menu.add(menuItem);

    return menuBar;
}
```

- Collaboration entre 3 éléments : **MenuBar**, **Menu** et **MenuItem**
- Association à des **raccourcis** clavier
- Production d'**ActionEvents**

Création d'un menu…

```
public JMenuBar createMenuBar() {
    JMenuBar menuBar;
    JMenu menu;
    JMenuItem menuItem;

    //Create the menu bar.
    menuBar = new JMenuBar();

    //Build the first menu.
    menu = new JMenu("Look Menu");
    menu.setMnemonic(KeyEvent.VK_A);
    menu.getAccessib
            "The onl
    menuBar.add(menu

    //a group of JMe
    menuItem = new J
    menuItem.setAcce
    menuItem.getAcce
    menuItem.addActi
    menu.add(menuIte

    menuItem = new J
    menuItem.setAcce
    menuItem.getAcce
    menuItem.addActi
    menu.add(menuIte

    return menuBar;
}
```

```
public void actionPerformed(ActionEvent e) {
    JMenuItem source = (JMenuItem) (e.getSource());

    try {
        if (source.getText().equals("Metal")) {
            UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
        } else {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        }
    } catch (UnsupportedLookAndFeelException ex) {
        ex.printStackTrace();
    } catch (IllegalAccessException ex) {
        ex.printStackTrace();
    } catch (InstantiationException ex) {
        ex.printStackTrace();
    } catch (ClassNotFoundException ex) {
        ex.printStackTrace();
    }

    SwingUtilities.updateComponentTreeUI(frame); //update screen
}
```

- Collaboration entre 3 éléments : **MenuBar**, **Menu** et **MenuItem**
- Association à des **raccourcis** clavier
- Production d'**ActionEvents**

Autres composants…

```
JTable table = new javax.swing.JTable(data, columnNames);
contentPane.add(table, BorderLayout.CENTER);

javax.swing.JProgressBar pbar;
pbar = new JProgressBar(0, 100);
pbar.setValue(45);
pbar.setStringPainted(true);
contentPane.add(pbar, BorderLayout.NORTH);

JButton Ok = new javax.swing.JButton("Ok");
contentPane.add(Ok, BorderLayout.SOUTH);
```

Révision

Architecture en 1 ou 2 couches ?

Listener dans une classe dédiée

Windows Adapter

Architecture en 1 ou 2 couches ?

- Comment les distinguer ?
  - Une architecture en 2 couches présuppose une **séparation métier x UI**
  - Si un calcul / raisonnement / action propre au métier se fait dans les classes d'UI → 1 Couche !
- Exemple : Convertisseur de température
  - Si le calcule se fait dans le Frame : 1 couche
  - Si le calcule se fait dans une classe à part : 2 couches

Listener dans une classe dédiée

- Une classe dédiée au traitement des événements (d'un type donné, d'un composant ou ensemble de composants donnés)
- **Réutilisation** du traitement
- **Isoler** le traitement aussi bien du métier que des composants graphiques
- Implémentation
  - Implémentation d'une interface *XXXXListener*
  - Extension d'une classe *XXXXAdapter*

Listener dans une classe dédiée

- Une classe dédiée au traitement des événements (d'un type donné, d'un composant ou ensemble de composants donnés)
- **Réutilisation** du traitement
- **Isoler** le traitement aus
  des composants graphi
- Implémentation
  - Implémentation d'une interface *XXXXListener*
  - Extension d'une classe *XXXXAdapter*

Un **Adapter** est une classe qui implémente tous les méthodes d'une interface *xxxlistener* en comptant plusieurs

Exemple

- Définition : *class yyyyAdapter extends XXXAdapter*

```
class ExternalWindowListner extends WindowAdapter {
    @Override
    public void windowClosing(java.awt.event.WindowEvent e) {
        System.exit(0);
    }
}
```

- Usage : *comp.addXXXListener (new yyyyAdapter ())*

GridLayoutExampleFrame

```
protected void initComponents () {
    this.addWindowListener(new ExternalWindowListner ());

    /* on definit une grille de 2x3, avec 10pts entre les cellulles */
    GridLayout grid = new GridLayout(2,3, 10, 10);
    this.setLayout(grid);
```

BorderLayoutExampleFrame

```
this.addWindowListener(new ExternalWindowListner ());

/* on definit le layoutmanager avec 2x plus d'espace vert. et horiz. */
BorderLayout border = new BorderLayout();
```

Exercices

Exercices

1) Implémenter le convertisseur avec Swing
   - Utiliser une architecture en 2 COUCHES
     - Classe métier (conversion) + classe UI
     - Le calcul n'est pas fait sur la même classe

2) Jeu de scrabble : version 2 couches
   - Classe métier :
     - Dictionnaire avec les mots dans l'ordre
     - Méthode pour mettre les mots dans le désordre
   - Classe UI :
     - AWT ou Swing : au choix
     - Modèle de *listener* : au choix