eBook Gratuit

APPRENEZ swing

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#swing

Table des matières

À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: swing

It is an unofficial and free swing ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official swing.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapitre 1: Commencer avec le swing

Remarques

Swing a été remplacé par JavaFX . Oracle recommande généralement de développer de nouvelles applications avec JavaFX. Still: Swing sera supporté en Java dans un avenir prévisible. JavaFX s'intègre également bien avec Swing, pour permettre aux applications de transition de se dérouler sans heurts.

Il est fortement recommandé d'avoir la plupart de vos composants Swing sur le thread de répartition des événements. Il est facile d'oublier de regrouper votre configuration graphique dans un appel `invokeLater` . De la documentation Java:

> Le code de gestion des événements Swing s'exécute sur un thread spécial appelé thread de distribution des événements. La plupart des codes qui invoquent les méthodes Swing s'exécutent également sur ce thread. Cela est nécessaire car la plupart des méthodes d'objet Swing ne sont pas "thread-safe": les invoquer à partir de plusieurs threads risque de provoquer des erreurs d'interférence ou de cohérence de la mémoire. Certaines méthodes de composant Swing sont étiquetées "thread safe" dans la spécification de l'API; ceux-ci peuvent être invoqués en toute sécurité à partir de n'importe quel thread. Toutes les autres méthodes du composant Swing doivent être appelées à partir du thread de distribution des événements. Les programmes qui ignorent cette règle peuvent fonctionner correctement la plupart du temps mais sont sujets à des erreurs imprévisibles difficiles à reproduire.

De plus, à moins que ce ne soit pour une bonne raison, assurez-vous *toujours* que vous avez appelé `setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)` sinon vous pourriez avoir à gérer une fuite de mémoire si vous oubliez de détruire la JVM.

Examples

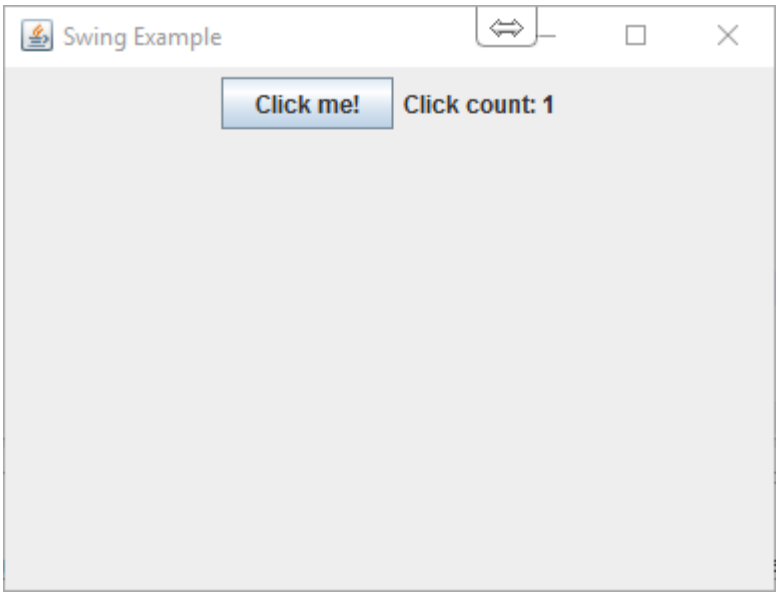
Incrémenter avec un bouton

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

/**
 * A very simple Swing example.
 */
public class SwingExample {
    /**
     * The number of times the user has clicked the button.
     */
    private long clickCount;
```

```
    /**
     * The main method: starting point of this application.
     *
     * @param arguments the unused command-line arguments.
     */
    public static void main(final String[] arguments) {
        new SwingExample().run();
    }

    /**
     * Schedule a job for the event-dispatching thread: create and show this
     * application's GUI.
     */
    private void run() {
        SwingUtilities.invokeLater(this::createAndShowGui);
    }

    /**
     * Create the simple GUI for this application and make it visible.
     */
    private void createAndShowGui() {
        // Create the frame and make sure the application exits when the user closes
        // the frame.
        JFrame mainFrame = new JFrame("Counter");
        mainFrame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        // Add a simple button and label.
        JPanel panel = new JPanel();
        JButton button = new JButton("Click me!");
        JLabel label = new JLabel("Click count: " + clickCount);
        panel.add(button);
        panel.add(label);
        mainFrame.getContentPane().add(panel);

        // Add an action listener to the button to increment the count displayed by
        // the label.
        button.addActionListener(actionEvent -> {
            clickCount++;
            label.setText("Click count: " + clickCount);
        });

        // Size the frame.
        mainFrame.setBounds(80, 60, 400, 300);
        //Center on screen
        mainFrame.setLocationRelativeTo(null);
        //Display frame
        mainFrame.setVisible(true);
    }
}
```

Résultat

Comme le bouton intitulé "Cliquez moi!" est appuyé sur le nombre de clics va augmenter d'un:



"Bonjour le monde!" sur le titre de la fenêtre avec lambda

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Hello World!");
            frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
            frame.setSize(200, 100);
            frame.setVisible(true);
        });
    }
}
```

Dans la méthode `main` :
Sur la première ligne, `SwingUtilities.invokeLater` est appelée et une expression lambda avec un bloc de code `() -> {...}` est transmise. Cela exécute l'expression lambda passée sur l'EDT, qui est l'abréviation de Event Dispatch Thread, au lieu du thread principal. Cela est nécessaire car dans le bloc de code de l'expression lambda, des composants Swing doivent être créés et mis à jour.

Dans le bloc de code de l'expression lambda:
Sur la première ligne, une nouvelle instance de `JFrame` appelée `frame` est créée à l'aide du `new JFrame("Hello World!")` . Cela crée une instance de fenêtre avec "Hello World!" sur son titre. Ensuite, sur la deuxième ligne, le `frame` est configuré sur `EXIT_ON_CLOSE` . Sinon, la fenêtre sera simplement fermée, mais l'exécution du programme restera active. La troisième ligne configure l'instance du `frame` sur 200 pixels de largeur et 100 pixels de hauteur à l'aide de la méthode `setSize` . Jusqu'à présent, l'exécution ne montre rien du tout. Seulement après avoir appelé `setVisible(true)` sur la quatrième ligne, l'instance du `frame` est configurée pour apparaître à l'écran.

"Bonjour le monde!" sur le titre de la fenêtre avec compatibilité

Utiliser `java.lang.Runnable` nous faisons notre "Hello World!" exemple disponible pour les utilisateurs de Java avec des versions datant de la version 1.2:

```
import javax.swing.JFrame;
import javax.swing.SwingUtilities;
import javax.swing.WindowConstants;

public class Main {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JFrame frame = new JFrame("Hello World!");
                frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
                frame.setSize(200, 100);
                frame.setVisible(true);
            }
        });
    }
}
```

Lire Commencer avec le swing en ligne: https://riptutorial.com/fr/swing/topic/2191/commencer-avec-le-swing

Chapitre 2: Disposition de la grille

Examples

Comment GridLayout fonctionne

Un `GridLayout` est un gestionnaire de disposition qui place des composants dans une grille de taille égale. Vous pouvez définir le nombre de lignes, de colonnes, l'espace horizontal et l'espace vertical à l'aide des méthodes suivantes:

- `setRows(int rows)`
- `setColumns(int columns)`
- `setHgap(int hgap)`
- `setVgap(int vgap)`

ou vous pouvez les définir avec les constructeurs suivants:

- `GridLayout(int rows, int columns)`
- `GridLayout(int rows, int columns, int hgap, int vgap)`

Si le nombre de lignes ou de colonnes est inconnu, vous pouvez définir la variable respective sur `0` . Par exemple:

```
new GridLayout(0, 3)
```

Cela fera que `GridLayout` aura 3 colonnes et autant de lignes que nécessaire.

L'exemple suivant montre comment un `GridLayout` présente des composants avec des valeurs différentes pour les lignes, les colonnes, les `GridLayout` verticaux et la taille de l'écran.

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.EventQueue;
import java.awt.GridLayout;

import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;
import javax.swing.WindowConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class GridLayoutExample {

    private GridLayout gridLayout;
    private JPanel gridPanel, contentPane;
    private JSpinner rowsSpinner, columnsSpinner, hgapSpinner, vgapSpinner;
```

```
    public void createAndShowGUI() {
        gridLayout = new GridLayout(5, 5, 3, 3);

        gridPanel = new JPanel(gridLayout);

        final ChangeListener rowsColumnsListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setRows((int) rowsSpinner.getValue());
                gridLayout.setColumns((int) columnsSpinner.getValue());
                fillGrid();
            }
        };

        final ChangeListener gapListener = new ChangeListener() {
            @Override
            public void stateChanged(ChangeEvent e) {
                gridLayout.setHgap((int) hgapSpinner.getValue());
                gridLayout.setVgap((int) vgapSpinner.getValue());
                gridLayout.layoutContainer(gridPanel);
                contentPane.revalidate();
                contentPane.repaint();
            }
        };

        rowsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getRows(), 1, 10, 1));
        rowsSpinner.addChangeListener(rowsColumnsListener);

        columnsSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getColumns(), 1, 10,
1));
        columnsSpinner.addChangeListener(rowsColumnsListener);

        hgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getHgap(), 0, 50, 1));
        hgapSpinner.addChangeListener(gapListener);

        vgapSpinner = new JSpinner(new SpinnerNumberModel(gridLayout.getVgap(), 0, 50, 1));
        vgapSpinner.addChangeListener(gapListener);

        JPanel actionPanel = new JPanel();
        actionPanel.add(new JLabel("Rows:"));
        actionPanel.add(rowsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Columns:"));
        actionPanel.add(columnsSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Horizontal gap:"));
        actionPanel.add(hgapSpinner);
        actionPanel.add(Box.createHorizontalStrut(10));
        actionPanel.add(new JLabel("Vertical gap:"));
        actionPanel.add(vgapSpinner);

        contentPane = new JPanel(new BorderLayout(0, 10));
        contentPane.add(gridPanel);
        contentPane.add(actionPanel, BorderLayout.SOUTH);

        fillGrid();

        JFrame frame = new JFrame("GridLayout Example");
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setContentPane(contentPane);
        frame.setSize(640, 480);
```

```
        frame.setLocationByPlatform(true);
        frame.setVisible(true);
    }

    private void fillGrid() {
        gridPanel.removeAll();
        for (int row = 0; row < gridLayout.getRows(); row++) {
            for (int col = 0; col < gridLayout.getColumns(); col++) {
                JLabel label = new JLabel("Row: " + row + " Column: " + col);
                label.setHorizontalAlignment(JLabel.CENTER);
                label.setBorder(BorderFactory.createLineBorder(Color.GRAY));
                gridPanel.add(label);
            }
        }
        contentPane.revalidate();
        contentPane.repaint();
    }

    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run() {
                new GridLayoutExample().createAndShowGUI();
            }
        });
    }

}
```

Lire Disposition de la grille en ligne: https://riptutorial.com/fr/swing/topic/2780/disposition-de-la-grille

Chapitre 3: Disposition GridBag

Syntaxe

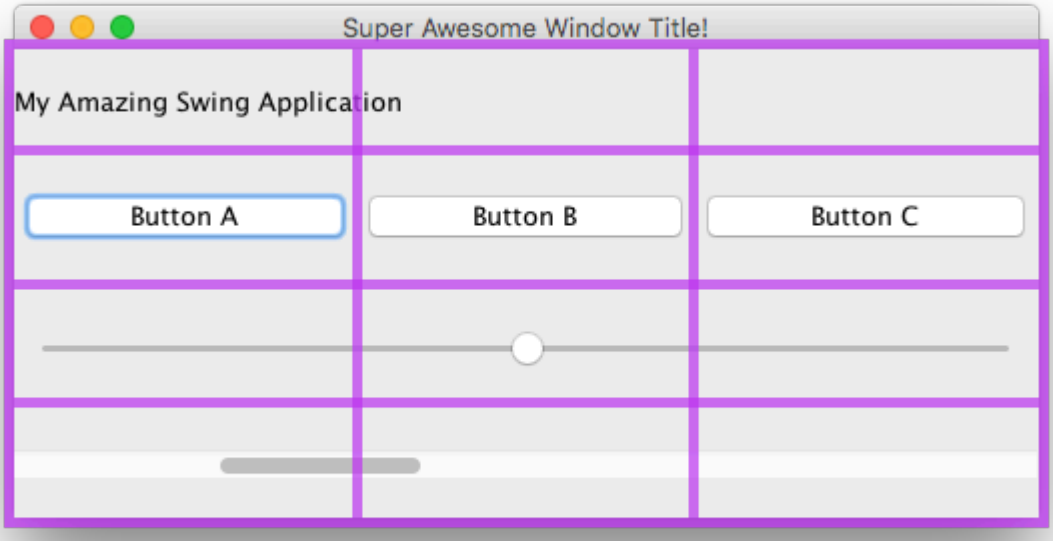
- frame.setLayout (new GridBagLayout ()); // Définit GridBagLayout pour le cadre
- pane.setLayout (new GridBagLayout ()); // Définit GridBagLayout pour Panel
- JPanel pane = new JPanel (new GridBagLayout ()); // Définit GridBagLayout pour Panel
- GridBagConstraints c = new GridBagConstraints () // Initialise un GridBagConstraint

Examples

Comment GridBagLayout fonctionne-t-il?

Les mises en page sont utilisées chaque fois que vous souhaitez que vos composants ne soient pas simplement affichés les uns à côté des autres. `GridBagLayout` est utile, car il divise votre fenêtre en lignes et en colonnes, et vous décidez de la rangée et de la colonne dans lesquelles placer les composants, ainsi que du nombre de lignes et de colonnes du composant.

Prenons cette fenêtre comme exemple. Les lignes de quadrillage ont été marquées pour indiquer la disposition.



Ici, j'ai créé 6 composants, conçus à l'aide d'un GridBagLayout.

Composant	Position	Taille
`JLabel` : "Mon application étonnante Swing"	`0, 0`	`3, 1`

Composant	Position	Taille
`JButton` : "Bouton A"	`0, 1`	`1, 1`
`JButton` : "Button B"	`1, 1`	`1, 1`
`JButton` : "Bouton C"	`2, 1`	`1, 1`
`JSlider`	`0, 2`	`3, 1`
`JScrollBar`	`0, 3`	`3, 1`

Notez que la position `0, 0` est en haut à gauche: les valeurs `x` (colonne) augmentent de gauche à droite, les valeurs `y` (ligne) augmentent de haut en bas.

Pour commencer à `GridBagLayout` des composants dans un `GridBagLayout` , commencez par définir la disposition de votre volet `JFrame` ou contenu.

```
frame.setLayout(new GridBagLayout());
//OR
pane.setLayout(new GridBagLayout());
//OR
JPanel pane = new JPanel(new GridBagLayout()); //Add the layout when creating your content
pane
```

Notez que vous ne définissez jamais la taille de la grille. Cela se fait automatiquement lorsque vous ajoutez vos composants.

Ensuite, vous devrez créer un objet `GridBagConstraints` .

```
GridBagConstraints c = new GridBagConstraints();
```

Pour vous assurer que vos composants remplissent la taille de la fenêtre, vous pouvez définir le poids de tous les composants sur 1. Le poids est utilisé pour déterminer comment répartir l'espace entre les colonnes et les lignes.

```
c.weightx = 1;
c.weighty = 1;
```

Vous pouvez également faire en sorte que les composants occupent autant d’espace horizontal que possible.

```
c.fill = GridBagConstraints.HORIZONTAL;
```

Vous pouvez également définir d'autres options de remplissage si vous le souhaitez.

```
GridBagConstraints.NONE //Don't fill components at all
GridBagConstraints.HORIZONTAL //Fill components horizontally
GridBagConstraints.VERTICAL //Fill components vertically
GridBagConstraints.BOTH //Fill components horizontally and vertically
```

Lors de la création de composants, vous souhaitez définir l'emplacement de la grille et le nombre de cases à utiliser. Par exemple, pour placer un bouton dans la 3ème ligne de la 2ème colonne et occuper un espace de 5 x 5, procédez comme suit. Gardez à l'esprit que la grille commence à `0, 0` et non `1, 1` .

```
JButton button = new JButton("Fancy Button!");
c.gridx = 2;
c.gridy = 1;
c.gridwidth = 5;
c.gridheight = 5;
pane.add(buttonA, c);
```

Lorsque vous ajoutez des composants à votre fenêtre, n'oubliez pas de transmettre les contraintes en tant que paramètre. Cela peut être vu dans la dernière ligne de l'exemple de code ci-dessus.

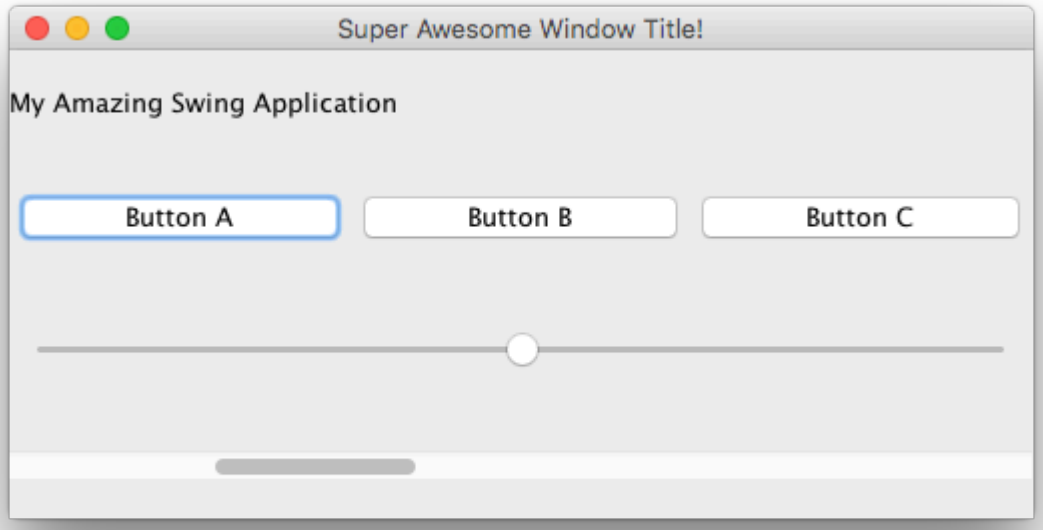
Vous pouvez réutiliser les mêmes `GridBagConstraints` pour chaque composant - le changer après l'ajout d'un composant ne change pas le composant ajouté précédemment.

Exemple

Voici le code de l'exemple au début de cette section.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(new GridBagLayout()); //Create a pane to house all content, and give
it a GridBagLayout
frame.setContentPane(pane);

GridBagConstraints c = new GridBagConstraints();
c.weightx = 1;
c.weighty = 1;
c.fill = GridBagConstraints.HORIZONTAL;

JLabel headerLabel = new JLabel("My Amazing Swing Application");
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 0;
pane.add(headerLabel, c);

JButton buttonA = new JButton("Button A");
c.gridx = 0;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonA, c);

JButton buttonB = new JButton("Button B");
c.gridx = 1;
c.gridwidth = 1;
c.gridy = 1;
```

```
pane.add(buttonB, c);

JButton buttonC = new JButton("Button C");
c.gridx = 2;
c.gridwidth = 1;
c.gridy = 1;
pane.add(buttonC, c);

JSlider slider = new JSlider(0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 2;
pane.add(slider, c);

JScrollBar scrollBar = new JScrollBar(JScrollBar.HORIZONTAL, 20, 20, 0, 100);
c.gridx = 0;
c.gridwidth = 3;
c.gridy = 3;
pane.add(scrollBar, c);

frame.setVisible(true); //Show the window
```



Lire Disposition GridBag en ligne: https://riptutorial.com/fr/swing/topic/3698/disposition-gridbag

Chapitre 4: Gestion de la mise en page

Examples

Mise en page frontière

```
import static java.awt.BorderLayout.*;
import javax.swing.*;
import java.awt.BorderLayout;

JPanel root = new JPanel(new BorderLayout());

root.add(new JButton("East"), EAST);
root.add(new JButton("West"), WEST);
root.add(new JButton("North"), NORTH);
root.add(new JButton("South"), SOUTH);
root.add(new JButton("Center"), CENTER);

JFrame frame = new JFrame();
frame.setContentPane(root);
frame.pack();
frame.setVisible(true);
```

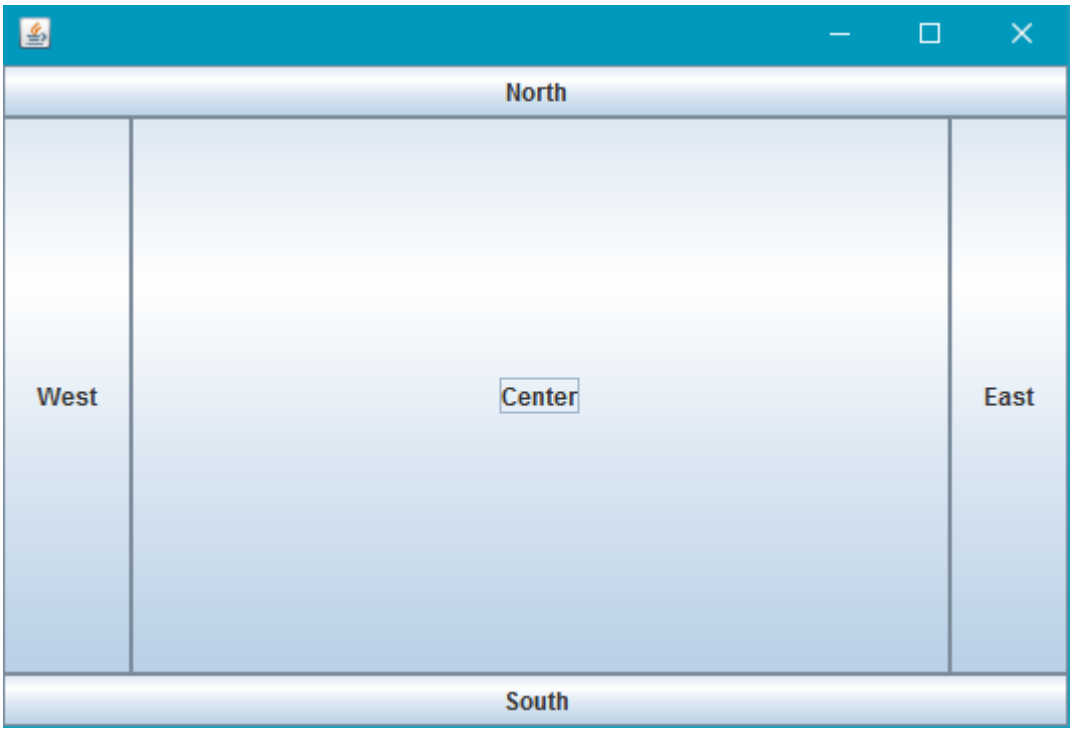
La disposition des bordures est l'un des gestionnaires de mise en page les plus simples. La façon d'utiliser un gestionnaire de disposition est de définir le gestionnaire d'un `JPanel` .

Les emplacements de mise en page de bordure suivent les règles suivantes:

- Nord et Sud: hauteur préférée
- Est et Ouest: largeur préférée
- Centre: espace restant maximal

Dans `BorderLayout` emplacements peuvent également être vides. Le gestionnaire de mise en page compense automatiquement les espaces vides, en les redimensionnant si nécessaire.

Voici à quoi ressemble cet exemple:



Disposition de flux

```
import javax.swing.*;
import java.awt.FlowLayout;

public class FlowExample {
    public static void main(String[] args){
        SwingUtilities.invokeLater(new Runnable(){

            @Override
            public void run(){
                JPanel panel = new JPanel();
                panel.setLayout(new FlowLayout());

                panel.add(new JButton("One"));
                panel.add(new JButton("Two"));
                panel.add(new JButton("Three"));
                panel.add(new JButton("Four"));
                panel.add(new JButton("Five"));

                JFrame frame = new JFrame();
                frame.setContentPane(Panel);
                frame.pack();
                frame.setVisible(true);
            }
        });
    }
}
```

La disposition de flux est le gestionnaire de disposition le plus simple que Swing a à offrir. La disposition de flux essaie de tout placer sur une ligne et, si la disposition déborde de la largeur, elle enveloppera la ligne. L'ordre est spécifié par l'ordre dans lequel vous ajoutez des composants à votre panneau.

Captures d'écran:

Disposition de la grille

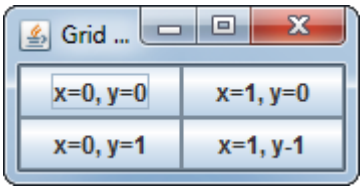
`GridLayout` vous permet d'organiser les composants sous la forme d'une grille.

Vous passez le nombre de lignes et de colonnes que vous voulez que la grille `GridLayout` au constructeur de `GridLayout` . Par exemple, le `new GridLayout(3, 2)` créera un `GridLayout` avec 3 lignes et 2 colonnes.

Lors de l'ajout de composants à un conteneur avec `GridLayout` , les composants seront ajoutés ligne par ligne, de gauche à droite:

```
import javax.swing.*;
import java.awt.GridLayout;

public class Example {
    public static void main(String[] args){
        SwingUtilities.invokeLater(Example::createAndShowJFrame);
    }

    private static void createAndShowJFrame(){
        JFrame jFrame = new JFrame("Grid Layout Example");

        // Create layout and add buttons to show restraints
        JPanel jPanel = new JPanel(new GridLayout(2, 2));
        jPanel.add(new JButton("x=0, y=0"));
        jPanel.add(new JButton("x=1, y=0"));
        jPanel.add(new JButton("x=0, y=1"));
        jPanel.add(new JButton("x=1, y-1"));

        jFrame.setContentPane(jPanel);
        jFrame.pack();
        jFrame.setLocationRelativeTo(null);
        jFrame.setVisible(true);
    }
}
```

Cela crée et affiche un `JFrame` qui ressemble à:



Une description plus détaillée est disponible: GridLayout

Lire Gestion de la mise en page en ligne: https://riptutorial.com/fr/swing/topic/5417/gestion-de-la-mise-en-page

Chapitre 5: Graphique

Examples

Utiliser la classe Graphics

Introduction

La classe `Graphics` vous permet de dessiner sur des composants Java tels qu'un `Jpanel` , il peut être utilisé pour dessiner des chaînes, des lignes, des formes et des images. Cela se fait en *remplaçant* la `paintComponent(Graphics g)` méthode de la `JComponent` vous dessinez sur l' utilisation des `Graphics` objet reçu comme argument pour faire le dessin:

`Board` classe

```
import java.awt.*;
import javax.swing.*;

public class Board extends JPanel{

    public Board() {
        setBackground(Color.WHITE);
    }

    @override
    public Dimension getPreferredSize() {
        return new Dimension(400, 400);
    }

    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        // draws a line diagonally across the screen
        g.drawLine(0, 0, 400, 400);
        // draws a rectangle around "hello there!"
        g.drawRect(140, 180, 115, 25);
    }
}
```

classe wrapper `DrawingCanvas`

```
import javax.swing.*;

public class DrawingCanvas extends JFrame {

    public DrawingCanvas() {

        Board board = new Board();

        add(board); // adds the Board to our JFrame
```

```
        pack(); // sets JFrame dimension to contain subcomponents

        setResizable(false);
        setTitle("Graphics Test");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        setLocationRelativeTo(null); // centers window on screen
    }

    public static void main(String[] args) {
        DrawingCanvas canvas = new DrawingCanvas();
        canvas.setVisible(true);
    }
}
```

Couleurs

Pour dessiner des formes avec des couleurs différentes, vous devez définir la couleur de l'objet `Graphics` avant chaque appel à l'aide de `setColor` :

```
g.setColor(Color.BLUE);  // draws a blue square
g.fillRect(10, 110, 100, 100);

g.setColor(Color.RED);  // draws a red circle
g.fillOval(10, 10, 100, 100);

g.setColor(Color.GREEN);  // draws a green triangle
int[] xPoints = {0, 200, 100};
int[] yPoints = {100, 100, 280};
g.fillPolygon(xPoints, yPoints, 3);
```

Dessin d'images

Les images peuvent être dessinées sur un `JComponent` utilisant la méthode `drawImage` de la classe `Graphics` :

charger une image

```
BufferedImage img;
try {
    img = ImageIO.read(new File("stackoverflow.jpg"));
} catch (IOException e) {
    throw new RuntimeException("Could not load image", e);
}
```

dessiner l'image

```
@Override
protected void paintComponent(Graphics g) {
    super.paintComponent(g);
```

```
    int x = 0;
    int y = 0;

    g.drawImage(img, x, y, this);
}
```

Les `x` et `y` spécifient l'emplacement du **haut à gauche** de l'image.

Utilisation de la méthode de repeindre pour créer une animation de base

La classe MyFrame étend JFrame et contient également la méthode principale

```
import javax.swing.JFrame;


public class MyFrame extends JFrame{

    //main method called on startup
    public static void main(String[] args) throws InterruptedException {

        //creates a frame window
        MyFrame frame = new MyFrame();

        //very basic game loop where the graphics are re-rendered
        while(true){
            frame.getPanel().repaint();

            //The program waits a while before rerendering
            Thread.sleep(12);
        }
    }

    //the MyPanel is the other class and it extends JPanel
    private MyPanel panel;

    //constructor that sets some basic staring values
    public MyFrame(){
        this.setSize(500, 500);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //creates the MyPanel with paramaters of x=0 and y=0
        panel = new MyPanel(0,0);
        //adds the panel (which is a JComponent because it extends JPanel)
        //into the frame
        this.add(panel);
        //shows the frame window
        this.setVisible(true);
    }

    //gets the panel
    public MyPanel getPanel(){
        return panel;
    }

}
```

La classe MyPanel qui étend JPanel et a la méthode paintComponent

```
import java.awt.Graphics;
import javax.swing.JPanel;


public class MyPanel extends JPanel{

    //two int variables to store the x and y coordinate
    private int x;
    private int y;

    //construcor of the MyPanel class
    public MyPanel(int x, int y){
        this.x = x;
        this.y = y;
    }

    /*the method that deals with the graphics
        this method is called when the component is first loaded,
         when the component is resized and when the repaint() method is
        called for this component
    */
    @Override
    public void paintComponent(Graphics g){
        super.paintComponent(g);

        //changes the x and y varible values
        x++;
        y++;

        //draws a rectangle at the x and y values
        g.fillRect(x, y, 50, 50);
    }

}
```

Lire Graphique en ligne: https://riptutorial.com/fr/swing/topic/5153/graphique

Chapitre 6: JList

Examples

Modifier les éléments sélectionnés dans une liste JList

Étant donné une `JList` comme

```
JList myList = new JList(items);
```

les éléments sélectionnés dans la liste peuvent être modifiés via le `ListSelectionModel` du `JList` :

```
ListSelectionModel sm = myList.getSelectionModel();
sm.clearSelection();                      // clears the selection
sm.setSelectionInterval(index, index);    // Sets a selection interval
                                          // (single element, in this case)
```

`JList` propose également des méthodes pratiques pour manipuler directement les index sélectionnés:

```
myList.setSelectionIndex(index);            // sets one selected index
                                            // could be used to define the Default Selection

myList.setSelectedIndices(arrayOfIndexes);  // sets all indexes contained in
                                            // the array as selected
```

Lire JList en ligne: https://riptutorial.com/fr/swing/topic/5413/jlist

Chapitre 7: Les bases

Examples

Retarder une tâche de l'interface utilisateur pour une période spécifique

Toutes les opérations liées à Swing se produisent sur un fil dédié (EDT - **E** vent **D** ispatch **T** HLit). Si ce thread est bloqué, l'interface utilisateur ne répond plus.

Par conséquent, si vous souhaitez retarder une opération, vous ne pouvez pas utiliser `Thread.sleep` . Utilisez plutôt un `javax.swing.Timer` . Par exemple, le `Timer` suivant inversera le texte de `JLabel`

```
  int delay = 2000;//specify the delay for the timer
  Timer timer = new Timer( delay, e -> {
    //The following code will be executed once the delay is reached
    String revertedText = new StringBuilder( label.getText() ).reverse().toString();
    label.setText( revertedText );
  } );
  timer.setRepeats( false );//make sure the timer only runs once
```

Un exemple complet d'utilisation de cette `Timer` est donné ci-dessous: l'interface utilisateur contient un bouton et une étiquette. Appuyez sur le bouton pour inverser le texte de l'étiquette après un délai de 2 secondes

```
import javax.swing.*;
import java.awt.*;

public final class DelayedExecutionExample {

  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }

  private static void showUI(){
    JFrame frame = new JFrame( "Delayed execution example" );

    JLabel label = new JLabel( "Hello world" );
    JButton button = new JButton( "Reverse text with delay" );
    button.addActionListener( event -> {
      button.setEnabled( false );
      //Instead of directly updating the label, we use a timer
      //This allows to introduce a delay, while keeping the EDT free
      int delay = 2000;
      Timer timer = new Timer( delay, e -> {
        String revertedText = new StringBuilder( label.getText() ).reverse().toString();
        label.setText( revertedText );
        button.setEnabled( true );
      } );
      timer.setRepeats( false );//make sure the timer only runs once
      timer.start();
    } );
```

```
    frame.add( label, BorderLayout.CENTER );
    frame.add( button, BorderLayout.SOUTH );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Répétez une tâche d'interface utilisateur à intervalle fixe

La mise à jour de l'état d'un composant Swing doit avoir lieu sur le thread de répartition des événements (EDT). `javax.swing.Timer` déclenche son `ActionListener` sur l'EDT, ce qui en fait un bon choix pour effectuer des opérations Swing.

L'exemple suivant met à jour le texte d'un `JLabel` toutes les deux secondes:

```
//Use a timer to update the label at a fixed interval
int delay = 2000;
Timer timer = new Timer( delay, e -> {
  String revertedText = new StringBuilder( label.getText() ).reverse().toString();
  label.setText( revertedText );
} );
timer.start();
```

Un exemple complet d'utilisation de cette `Timer` est donné ci-dessous: l'interface utilisateur contient une étiquette et le texte de l'étiquette est annulé toutes les deux secondes.

```
import javax.swing.*;
import java.awt.*;

public final class RepeatTaskFixedIntervalExample {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated task example" );
    JLabel label = new JLabel( "Hello world" );

    //Use a timer to update the label at a fixed interval
    int delay = 2000;
    Timer timer = new Timer( delay, e -> {
      String revertedText = new StringBuilder( label.getText() ).reverse().toString();
      label.setText( revertedText );
    } );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Exécuter une tâche d'interface utilisateur un nombre de fois fixe

Dans `ActionListener` attaché à `javax.swing.Timer` , vous pouvez suivre le nombre de fois que le `Timer` exécuté l' `ActionListener` . Une fois que le nombre de fois requis est atteint, vous pouvez utiliser la méthode `Timer#stop()` pour arrêter le `Timer` .

```
Timer timer = new Timer( delay, new ActionListener() {
  private int counter = 0;
  @Override
  public void actionPerformed( ActionEvent e ) {
    counter++;//keep track of the number of times the Timer executed
    label.setText( counter + "" );
    if ( counter == 5 ){
      ( ( Timer ) e.getSource() ).stop();
    }
  }
});
```

Un exemple complet d'utilisation de cette `Timer` est donné ci-dessous: il montre une interface utilisateur où le texte de l'étiquette comptera de zéro à cinq. Une fois que cinq est atteint, la `Timer` est arrêtée.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public final class RepeatFixedNumberOfTimes {
  public static void main( String[] args ) {
    EventQueue.invokeLater( () -> showUI() );
  }
  private static void showUI(){
    JFrame frame = new JFrame( "Repeated fixed number of times example" );
    JLabel label = new JLabel( "0" );

    int delay = 2000;
    Timer timer = new Timer( delay, new ActionListener() {
      private int counter = 0;
      @Override
      public void actionPerformed( ActionEvent e ) {
        counter++;//keep track of the number of times the Timer executed
        label.setText( counter + "" );
        if ( counter == 5 ){
          //stop the Timer when we reach 5
          ( ( Timer ) e.getSource() ).stop();
        }
      }
    });
    timer.setInitialDelay( delay );
    timer.start();

    frame.add( label, BorderLayout.CENTER );
    frame.pack();
    frame.setDefaultCloseOperation( WindowConstants.EXIT_ON_CLOSE );
    frame.setVisible( true );
  }
}
```

Créer votre premier JFrame

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class FrameCreator {

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame();
            //JFrames will not display without size being set
            frame.setSize(500, 500);

            JLabel label = new JLabel("Hello World");
            frame.add(label);

            frame.setVisible(true);
        });
    }

}
```

Comme vous pouvez le constater, si vous exécutez ce code, l'étiquette se trouve dans un très mauvais endroit. Il est difficile de changer de manière satisfaisante en utilisant la méthode `add` . Pour permettre un placement plus dynamique et plus flexible, consultez les gestionnaires de disposition Swing .

Créer une sous-classe JFrame

```
import java.awt.FlowLayout;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    private static CustomFrame statFrame;

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        JLabel label = new JLabel(labelText);
        add(label);

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    }

    public void addLabel(String labelText) {
        JLabel label = new JLabel(labelText);
        add(label);
        this.validate();
```

```
    }

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            statFrame = frame;
            frame.setVisible(true);
        });

        try {
            Thread.sleep(5000);
        } catch (InterruptedException ex) {
            //Handle error
        }

        SwingUtilities.invokeLater(() -> statFrame.addLabel("Oh, hello world too."));
    }

}
```

Pour plus d'informations sur FlowLayout ici .

Écouter un événement

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;

public class CustomFrame extends JFrame {

    public CustomFrame(String labelText) {
        setSize(500, 500);

        //See link below for more info on FlowLayout
        this.setLayout(new FlowLayout());

        //Tells the JFrame what to do when it's closed
        //In this case, we're saying to "Dispose" on remove all resources
        //associated with the frame on close
        this.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        //Add a button
        JButton btn = new JButton("Hello button");
        //And a textbox
        JTextField field = new JTextField("Name");
        field.setSize(150, 50);
        //This next block of code executes whenever the button is clicked.
        btn.addActionListener((evt) -> {
            JLabel helloLbl = new JLabel("Hello " + field.getText());
            add(helloLbl);
            validate();
        });
        add(btn);
```

```
        add(field);
    }

    public static void main(String args[]) {
        //All Swing actions should be run on the Event Dispatch Thread (EDT)
        //Calling SwingUtilities.invokeLater makes sure that happens.
        SwingUtilities.invokeLater(() -> {
            CustomFrame frame = new CustomFrame("Hello Jungle");
            //This is simply being done so it can be accessed later
            frame.setVisible(true);
        });
    }

}
```

Créez un popup "Veuillez patienter ..."

Ce code peut être ajouté à tout événement tel qu'un écouteur, un bouton, etc. Un `JDialog` bloquant apparaîtra et restera jusqu'à la fin du processus.

```
final JDialog loading = new JDialog(parentComponent);
JPanel p1 = new JPanel(new BorderLayout());
p1.add(new JLabel("Please wait..."), BorderLayout.CENTER);
loading.setUndecorated(true);
loading.getContentPane().add(p1);
loading.pack();
loading.setLocationRelativeTo(parentComponent);
loading.setDefaultCloseOperation(JDialog.DO_NOTHING_ON_CLOSE);
loading.setModal(true);

SwingWorker<String, Void> worker = new SwingWorker<String, Void>() {
    @Override
    protected String doInBackground() throws InterruptedException
        /** Execute some operation */
    }
    @Override
    protected void done() {
        loading.dispose();
    }
};
worker.execute(); //here the process thread initiates
loading.setVisible(true);
try {
    worker.get(); //here the parent thread waits for completion
} catch (Exception e1) {
    e1.printStackTrace();
}
```

Ajout de JButtons (Hello World Pt.2)

En supposant que vous avez créé un JFrame et que Swing a été importé ...

Vous pouvez importer le Swing entièrement

```
import javax.Swing.*;
```

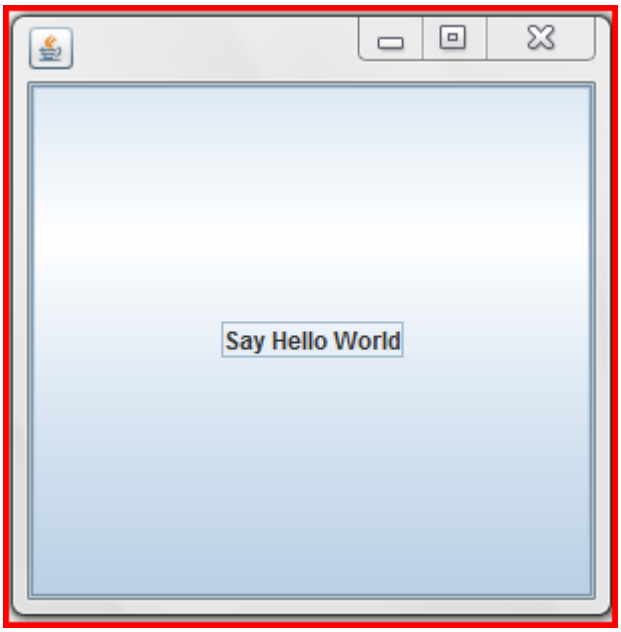
ou Vous pouvez importer les composants / cadres Swing que vous souhaitez utiliser

```
import javax.Swing.Jframe;
import javax.Swing.JButton;
```

Maintenant, il faut ajouter le Jbutton ...

```
public static void main(String[] args) {

    JFrame frame = new JFrame(); //creates the frame
    frame.setSize(300, 300);
    frame.setVisible(true);

    ////////////////////////ADDING BUTTON BELOW//////////////////////////////
    JButton B = new JButton("Say Hello World");
    B.addMouseListener(new MouseAdapter() {

        public void mouseReleased(MouseEvent arg0) {
            System.out.println("Hello World");
        }

    });
    B.setBounds(0, 0,frame.getHeight(), frame.getWidth());
    B.setVisible(true);
    frame.add(B);
    ////////////////////////////////////////////////////////////////////////
}
```

En exécutant / Compilant ce code, vous devriez obtenir quelque chose comme ça ...



Lorsque le bouton est cliqué, "Hello World" devrait également apparaître dans votre console.

Lire Les bases en ligne: https://riptutorial.com/fr/swing/topic/5415/les-bases

Chapitre 8: MigLayout

Examples

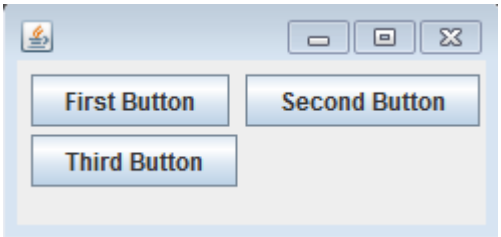
Éléments d'emballage

Cet exemple montre comment placer 3 boutons au total avec 2 boutons dans la première ligne. Ensuite, un retour à la ligne se produit, le dernier bouton est dans une nouvelle ligne.

Les contraintes sont des chaînes simples, dans ce cas "wrap" lors du placement du composant.

```
public class ShowMigLayout {

    // Create the elements
    private final JFrame demo = new JFrame();
    private final JPanel panel = new JPanel();
    private final JButton button1 = new JButton("First Button");
    private final JButton button2 = new JButton("Second Button");
    private final JButton button3 = new JButton("Third Button");

    public static void main(String[] args) {
        ShowMigLayout showMigLayout = new ShowMigLayout();
        SwingUtilities.invokeLater(showMigLayout::createAndShowGui);
    }

    public void createAndShowGui() {
        // Set the position and the size of the frame
        demo.setBounds(400, 400, 250, 120);

        // Tell the panel to use the MigLayout as layout manager
        panel.setLayout(new MigLayout());

        panel.add(button1);
        // Notice the wrapping
        panel.add(button2, "wrap");
        panel.add(button3);

        demo.add(panel);
        demo.setVisible(true);
    }
}
```

Sortie:



Lire MigLayout en ligne: https://riptutorial.com/fr/swing/topic/2966/miglayout

Chapitre 9: minuterie dans JFrame

Examples

Timer In JFrame

Supposons que votre programme Java comporte un bouton qui décompte un temps. Voici le code pour 10 minutes.

```
private final static long REFRESH_LIST_PERIOD=10 * 60 * 1000; //10 minutes

Timer timer = new Timer(1000, new ActionListener() {

   @Override
   public void actionPerformed(ActionEvent e) {
    if (cnt > 0) {
     cnt = cnt - 1000;
     btnCounter.setText("Remained (" + format.format(new Date(cnt)) + ")");
    } else {
     cnt = REFRESH_LIST_PERIOD;
     //TODO
    }

   }
  });

  timer.start();
```

Lire minuterie dans JFrame en ligne: https://riptutorial.com/fr/swing/topic/6745/minuterie-dans-jframe

Chapitre 10: Modèle MVP

Examples

Exemple MVP simple

Pour illustrer un exemple simple d'utilisation du modèle MVP, considérez le code suivant qui crée une interface utilisateur simple avec uniquement un bouton et une étiquette. Lorsque l'utilisateur clique sur le bouton, l'étiquette se met à jour avec le nombre de clics sur le bouton.

Nous avons 5 classes:

- Model - Le POJO pour maintenir l'état (M dans MVP)
- View - La classe avec code UI (V dans MVP)
- ViewListener - Interface fournissant des méthodes pour répondre aux actions dans la vue
- Présentateur - Répond aux entrées et met à jour la vue (P dans MVP)
- Application - La classe "principale" pour tout rassembler et lancer l'application

Une classe "modèle" minimale qui ne fait que maintenir une variable de `count` unique.

```
/**
 * A minimal class to maintain some state
 */
public class Model {
    private int count = 0;

    public void addOneToCount() {
        count++;
    }

    public int getCount() {
        return count;
    }
}
```

Une interface minimale pour notifier les auditeurs:

```
/**
 * Provides methods to notify on user interaction
 */
public interface ViewListener {
    public void onButtonClicked();
}
```

La classe de vue construit tous les éléments de l'interface utilisateur. La vue, et *uniquement* la vue, doit faire référence à des éléments de l'interface utilisateur (par exemple, aucun bouton, champ de texte, etc. dans le présentateur ou dans d'autres classes).

```
/**
 * Provides the UI elements
```

```
 */

import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;

public class View {
    // A list of listeners subscribed to this view
    private final ArrayList<ViewListener> listeners;
    private final JLabel label;

    public View() {
        final JFrame frame = new JFrame();
        frame.setSize(200, 100);
        frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        frame.setLayout(new GridLayout());

        final JButton button = new JButton("Hello, world!");

        button.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(final ActionEvent e) {
                notifyListenersOnButtonClicked();
            }
        });
        frame.add(button);

        label = new JLabel();
        frame.add(label);

        this.listeners = new ArrayList<ViewListener>();

        frame.setVisible(true);
    }

    // Iterate through the list, notifying each listner individualy
    private void notifyListenersOnButtonClicked() {
        for (final ViewListener listener : listeners) {
            listener.onButtonClicked();
        }
    }

    // Subscribe a listener
    public void addListener(final ViewListener listener) {
        listeners.add(listener);
    }

    public void setLabelText(final String text) {
        label.setText(text);
    }
}
```

La logique de notification peut également être codée comme ceci dans Java8:

```
        ...
        final Button button = new Button("Hello, world!");
        // In order to do so, our interface must be changed to accept the event parametre
        button.addActionListener((event) -> {
            notifyListeners(ViewListener::onButtonClicked, event);
            // Example of calling methodThatTakesALong, would be the same as callying:
            // notifyListeners((listener, long)->listener.methodThatTakesALong(long), 10L)
            notifyListeners(ViewListener::methodThatTakesALong, 10L);
        });
        frame.add(button);
        ...

/**
 * Iterates through the subscribed listeneres notifying each listener individually.
 * Note: the {@literal '<T>' in private <T> void} is a Bounded Type Parametre.
 *
 * @param <T>      Any Reference Type (basically a class).
 *
 * @param consumer A method with two parameters and no return,
 *                 the 1st parametre is a ViewListner,
 *                 the 2nd parametre is value of type T.
 *
 * @param data     The value used as parametre for the second argument of the
 *                 method described by the parametre consumer.
 */
private <T> void notifyListeners(final BiConsumer<ViewListener, T> consumer, final T data) {
    // Iterate through the list, notifying each listener, java8 style
    listeners.forEach((listener) -> {

        // Calls the funcion described by the object consumer.
        consumer.accept(listener, data);

        // When this method is called using ViewListener::onButtonClicked
        // the line: consumer.accept(listener,data); can be read as:
        // void accept(ViewListener listener, ActionEvent data) {
        //     listener.onButtonClicked(data);
        // }

    });
}
```

L’interface doit être restructurée pour que l’ActionEvent devienne un paramètre:

```
public interface ViewListener {
    public void onButtonClicked(ActionEvent evt);
    // Example of methodThatTakesALong signature
    public void methodThatTakesALong(long );
}
```

Ici, une seule méthode notify est nécessaire, la méthode d'écoute réelle et son paramètre sont transmis en tant que paramètres. Dans le cas où cela est nécessaire, cela peut aussi être utilisé pour quelque chose d'un peu moins astucieux que le traitement des événements, tout fonctionne tant qu'il y a une méthode dans l'interface, par exemple:

```
        notifyListeners(ViewListener::methodThatTakesALong, -1L);
```

Le présentateur peut prendre la vue et s'ajouter en tant qu'auditeur. Lorsque le bouton est cliqué dans la vue, la vue avertit tous les écouteurs (y compris le présentateur). Maintenant que le présentateur est averti, il peut prendre les mesures appropriées pour mettre à jour le modèle (c'est-à-dire l'état de l'application), puis mettre à jour la vue en conséquence.

```
/**
 * Responsible to responding to user interaction and updating the view
 */
public class Presenter implements ViewListener {
    private final View view;
    private final Model model;

    public Presenter(final View view, final Model model) {
        this.view = view;
        view.addListener(this);
        this.model = model;
    }

    @Override
    public void onButtonClicked() {
        // Update the model (ie. the state of the application)
        model.addOneToCount();
        // Update the view
        view.setLabelText(String.valueOf(model.getCount()));
    }
}
```

Pour tout rassembler, la vue peut être créée et injectée dans le présentateur. De même, un modèle initial peut être créé et injecté. Bien que les deux *puissent* être créés dans le présentateur, leur injection dans le constructeur permet des tests beaucoup plus simples.

```
public class Application {
    public Application() {
        final View view = new View();
        final Model model = new Model();
        new Presenter(view, model);
    }

    public static void main(String... args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new Application();
            }
        });
    }
}
```

Lire Modèle MVP en ligne: https://riptutorial.com/fr/swing/topic/5154/modele-mvp

Chapitre 11: StyledDocument

Syntaxe

- doc.insertString (index, texte, attributs); // les attributs doivent être un attributSet

Examples

Créer un DefaultStyledDocument

```
try {
    StyledDocument doc = new DefaultStyledDocument();
    doc.insertString(0, "This is the beginning text", null);
    doc.insertString(doc.getLength(), "\nInserting new line at end of doc", null);
    MutableAttributeSet attrs = new SimpleAttributeSet();
    StyleConstants.setBold(attrs, true);
    doc.insertString(5, "This is bold text after 'this'", attrs);
} catch (BadLocationException ex) {
    //handle error
}
```

DefaultStyledDocuments sera probablement vos ressources les plus utilisées. Ils peuvent être créés directement et sous- `StyledDocument` la classe abstraite `StyledDocument` .

Ajout de StyledDocument à JTextPane

```
try {
        JTextPane pane = new JTextPane();
        StyledDocument doc = new DefaultStyledDocument();
        doc.insertString(0, "Some text", null);
        pane.setDocument(doc); //Technically takes any subclass of Document
    } catch (BadLocationException ex) {
        //handle error
    }
```

Le JTextPane peut ensuite être ajouté à n'importe quel formulaire de l'interface graphique Swing.

Copie de DefaultStyledDocument

`StyledDocuments` généralement pas le clone et doivent donc les copier différemment si cela est nécessaire.

```
try {
        //Initialization
        DefaultStyledDocument sourceDoc = new DefaultStyledDocument();
        DefaultStyledDocument destDoc = new DefaultStyledDocument();
        MutableAttributeSet bold = new SimpleAttributeSet();
        StyleConstants.setBold(bold, true);
        MutableAttributeSet italic = new SimpleAttributeSet();
```

```
        StyleConstants.setItalic(italic, true);
        sourceDoc.insertString(0, "Some bold text. ", bold);
        sourceDoc.insertString(sourceDoc.getLength(), "Some italic text", italic);

        //This does the actual copying
        String text = sourceDoc.getText(0, sourceDoc.getLength()); //This copies text, but
loses formatting.
        for (int i = 0; i < text.length(); i++) {
            Element e = destDoc.getCharacterElement(i); //A Elment describes a particular part
of a document, in this case a character
            AttributeSet attr = e.getAttributes(); //Gets the attributes for the character
            destDoc.insertString(destDoc.getLength(), text.substring(i, i+1), attr); //Gets
the single character and sets its attributes from the element
        }
    } catch (BadLocationException ex) {
        //handle error
    }
```

Sérialisation d'un DefaultStyledDocument en RTF

À l'aide de la bibliothèque AdvancedRTFEditorKit , vous pouvez sérialiser un `DefaultStyledDocument` en une chaîne RTF.

```
try {
   DefaultStyledDocument writeDoc = new DefaultStyledDocument();
   writeDoc.insertString(0, "Test string", null);

   AdvancedRTFEditorKit kit = new AdvancedRTFEditorKit();
   //Other writers, such as a FileWriter, may be used
   //OutputStreams are also an option
   Writer writer = new StringWriter();
   //You can write just a portion of the document by modifying the start
   //and end indexes
   kit.write(writer, writeDoc, 0, writeDoc.getLength());
   //This is the RTF String
   String rtfDoc = writer.toString();

   //As above this may be a different kind of reader or an InputStream
   StringReader reader = new StringReader(rtfDoc);
   //AdvancedRTFDocument extends DefaultStyledDocument and can generally
   //be used wherever DefaultStyledDocument can be.
   //However for reading, AdvancedRTFDocument must be used
   DefaultStyledDocument readDoc = new AdvancedRTFDocument();
   //You can insert at different values by changing the "0"
   kit.read(reader, readDoc, 0);
   //readDoc is now the same as writeDoc
} catch (BadLocationException | IOException ex) {
    //Handle exception
    ex.printStackTrace();
}
```

Lire StyledDocument en ligne: https://riptutorial.com/fr/swing/topic/5416/styleddocument

Chapitre 12: Swing Workers et l'EDT

Syntaxe

- classe publique abstraite SwingWorker <T, V>
- T - le type de résultat renvoyé par le doInBackground de ce SwingWorker et les méthodes get.
- V - le type utilisé pour effectuer des résultats intermédiaires par les méthodes de publication et de traitement de SwingWorker.
- T doInBackground () - La fonction abstraite qui doit être remplacée. Le type de retour est T.

Examples

Fil de distribution principal et d'événement

Comme tout programme java, chaque programme de swing commence par une méthode principale. La méthode principale est initiée par le thread principal. Cependant, les composants Swing doivent être créés et mis à jour sur le thread de distribution des événements (ou short: EDT). Pour illustrer la dynamique entre le thread principal et l'EDT, jetez un coup d'œil à ce Hello World! Exemple.

Le thread principal est juste utilisé pour déléguer la création de la fenêtre à l'EDT. Si l'EDT n'est pas encore lancée, le premier appel à `SwingUtilities.invokeLater` configurera l'infrastructure nécessaire au traitement des composants Swing. En outre, l'EDT reste active en arrière-plan. Le thread principal va mourir directement après le lancement de la configuration EDT, mais l'EDT restera actif jusqu'à ce que l'utilisateur quitte le programme. Cela peut être réalisé en appuyant sur la case de `JFrame` instance `JFrame` visible. Cela va arrêter l'EDT et le processus du programme va complètement.

Recherchez les N premiers nombres pairs et affichez les résultats dans un JTextArea où les calculs sont effectués en arrière-plan.

```
import java.awt.EventQueue;
import java.awt.GridLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
import java.util.ArrayList;
import java.util.List;

import javax.swing.JFrame;
import javax.swing.JTextArea;
import javax.swing.SwingWorker;

class PrimeNumbersTask extends SwingWorker<List<Integer>, Integer> {
```

```
    private final int numbersToFind;

    private final JTextArea textArea;

    PrimeNumbersTask(JTextArea textArea, int numbersToFind) {
        this.numbersToFind = numbersToFind;
        this.textArea = textArea;
    }

    @Override
    public List<Integer> doInBackground() {
        final List<Integer> result = new ArrayList<>();
        boolean interrupted = false;
        for (int i = 0; !interrupted && (i < numbersToFind); i += 2) {
            interrupted = doIntenseComputing();
            result.add(i);
            publish(i); // sends data to process function
        }
        return result;
    }

    private boolean doIntenseComputing() {
        try {
            Thread.sleep(1000);
        } catch (InterruptedException e) {
            return true;
        }
        return false;
    }

    @Override
    protected void process(List<Integer> chunks) {
        for (int number : chunks) {
            // the process method will be called on the EDT
            // thus UI elementes may be updated in here
            textArea.append(number + "\n");
        }
    }
}

public class SwingWorkerExample extends JFrame {
    private JTextArea textArea;

    public SwingWorkerExample() {
        super("Java SwingWorker Example");
        init();
    }

    private void init() {
        setSize(400, 400);
        setLayout(new GridLayout(1, 1));
        textArea = new JTextArea();
        add(textArea);

        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent e) {
                dispose();
                System.exit(0);
            }
        });
    }
```

```
    public static void main(String args[]) throws Exception {

        SwingWorkerExample ui = new SwingWorkerExample();
        EventQueue.invokeLater(() -> {
            ui.setVisible(true);
        });

        int n = 100;
        PrimeNumbersTask task = new PrimeNumbersTask(ui.textArea, n);
        task.execute(); // run async worker which will do long running task on a
        // different thread
        System.out.println(task.get());
    }
}
```

Lire Swing Workers et l'EDT en ligne: https://riptutorial.com/fr/swing/topic/3431/swing-workers-et-l-edt

Chapitre 13: Utilisation de l'apparence

Examples

Utiliser le système L & F

Swing supporte pas mal de L & F natifs.
Vous pouvez toujours facilement en installer un sans faire appel à une classe L & F spécifique:

```
public class SystemLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
                try
                {
                    // Installing native L&F as a current application L&F
                    // We do not know what exactly L&F class is, it is provided by the
UIManager
                    UIManager.setLookAndFeel ( UIManager.getSystemLookAndFeelClassName () );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some natively-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
```

```
                content.add ( new JButton ( "Native-looking button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Ce sont les L & F natifs pris en charge par JDK (OS -> L & F):

OS	Nom de L & F	Classe L & F
Solaris, Linux avec GTK +	GTK +	com.sun.java.swing.plaf.gtk.GTKLookAndFeel
Autre Solaris, Linux	Motif	com.sun.java.swing.plaf.motif.MotifLookAndFeel
Windows classique	les fenêtres	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows XP	Windows XP	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Windows Vista	Windows Vista	com.sun.java.swing.plaf.windows.WindowsLookAndFeel
Macintosh	Macintosh	com.apple.laf.AquaLookAndFeel *
IBM UNIX	IBM	javax.swing.plaf.synth.SynthLookAndFeel *
HP UX	HP	javax.swing.plaf.synth.SynthLookAndFeel *

* Ces L & F sont fournis par le fournisseur du système et le nom de la classe L & F peut varier

Utiliser L & F personnalisé

```
public class CustomLookAndFeel
{
    public static void main ( final String[] args )
    {
        // L&F installation should be performed within EDT (Event Dispatch Thread)
        // This is important to avoid any UI issues, exceptions or even deadlocks
        SwingUtilities.invokeLater ( new Runnable ()
        {
            @Override
            public void run ()
            {
                // Process of L&F installation might throw multiple exceptions
                // It is always up to you whether to handle or ignore them
                // In most common cases you would never encounter any of those
```

```
                try
                {
                    // Installing custom L&F as a current application L&F
                    UIManager.setLookAndFeel ( "javax.swing.plaf.metal.MetalLookAndFeel" );
                }
                catch ( final ClassNotFoundException e )
                {
                    // L&F class was not found
                    e.printStackTrace ();
                }
                catch ( final InstantiationException e )
                {
                    // Exception while instantiating L&F class
                    e.printStackTrace ();
                }
                catch ( final IllegalAccessException e )
                {
                    // Class or initializer isn't accessible
                    e.printStackTrace ();
                }
                catch ( final UnsupportedLookAndFeelException e )
                {
                    // L&F is not supported on the current system
                    e.printStackTrace ();
                }

                // Now we can create some pretty-looking UI
                // This is just a small sample frame with a single button on it
                final JFrame frame = new JFrame ();
                final JPanel content = new JPanel ( new FlowLayout () );
                content.setBorder ( BorderFactory.createEmptyBorder ( 50, 50, 50, 50 ) );
                content.add ( new JButton ( "Metal button" ) );
                frame.setContentPane ( content );
                frame.setDefaultCloseOperation ( WindowConstants.EXIT_ON_CLOSE );
                frame.pack ();
                frame.setLocationRelativeTo ( null );
                frame.setVisible ( true );
            }
        } );
    }
}
```

Vous pouvez trouver une liste énorme des Swing L & F disponibles dans le sujet ici: Java Look and Feel (L & F)
Gardez à l'esprit que certains de ces L & F peuvent être assez obsolètes à ce stade.

Lire Utilisation de l'apparence en ligne: https://riptutorial.com/fr/swing/topic/3627/utilisation-de-l-apparence

Chapitre 14: Utiliser Swing pour les interfaces utilisateur graphiques

Remarques

Quitter l'application sur la fenêtre close

Il est facile d'oublier de quitter l'application lorsque la fenêtre est fermée. N'oubliez pas d'ajouter la ligne suivante.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the JFrame is closed
```

Examples

Création d'une fenêtre vide (JFrame)

Créer le JFrame

Créer une fenêtre est facile. Il suffit de créer un `JFrame` .

```
JFrame frame = new JFrame();
```

Titrer la fenêtre

Vous voudrez peut-être donner un titre à votre fenêtre. Vous pouvez le faire en passant une chaîne lors de la création de `JFrame` ou en appelant `frame.setTitle(String title)` .

```
JFrame frame = new JFrame("Super Awesome Window Title!");
//OR
frame.setTitle("Super Awesome Window Title!");
```

Définition de la taille de la fenêtre

La fenêtre sera aussi petite que possible lorsqu'elle aura été créée. Pour l'agrandir, vous pouvez définir explicitement sa taille:

```
frame.setSize(512, 256);
```

Ou vous pouvez avoir la taille du cadre elle-même en fonction de la taille de son contenu avec la méthode `pack()` .

```
frame.pack();
```

Les méthodes `setSize()` et `pack()` s'excluent mutuellement, utilisez donc l'une ou l'autre.

Que faire sur Window Close

Notez que l'application ne **se** fermera **pas** lorsque la fenêtre aura été fermée. Vous pouvez quitter l'application après la fermeture de la fenêtre en indiquant à `JFrame` de le faire.

```
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

Alternativement, vous pouvez dire à la fenêtre de faire autre chose quand elle est fermée.

```
WindowConstants.DISPOSE_ON_CLOSE //Get rid of the window
WindowConstants.EXIT_ON_CLOSE //Quit the application
WindowConstants.DO_NOTHING_ON_CLOSE //Don't even close the window
WindowConstants.HIDE_ON_CLOSE //Hides the window - This is the default action
```

Création d'un volet de contenu

Une étape facultative consiste à créer un volet de contenu pour votre fenêtre. Ce n'est pas nécessaire, mais si vous le souhaitez, créez un `JPanel` et appelez `frame.setContentPane(Component component)` .

```
JPanel pane = new JPanel();
frame.setContentPane(pane);
```

Montrer la fenêtre

Après l'avoir créé, vous souhaiterez créer vos composants, puis afficher la fenêtre. Montrer la fenêtre est fait comme tel.

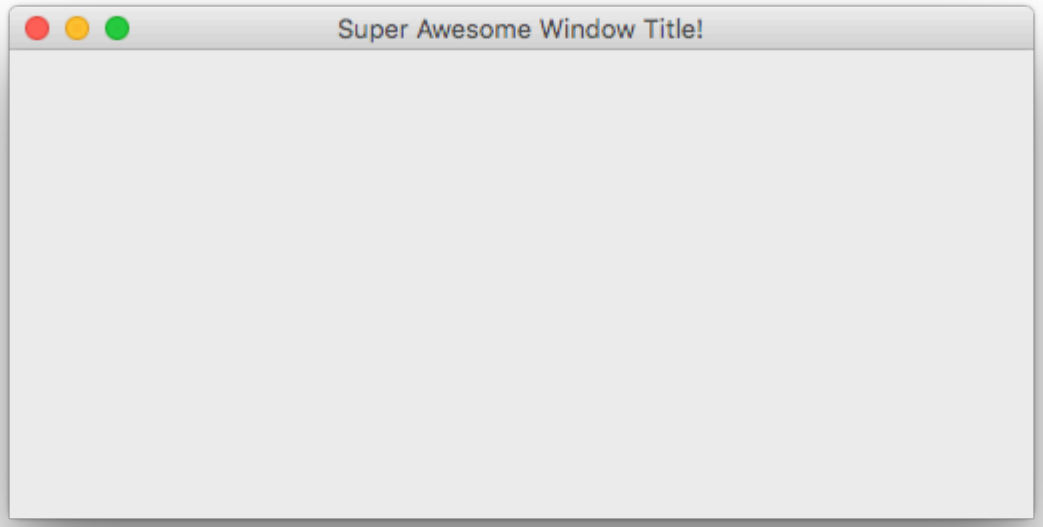
```
frame.setVisible(true);
```

Exemple

Pour ceux d'entre vous qui aiment copier et coller, voici un exemple de code.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
```

```
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

frame.setVisible(true); //Show the window
```



Ajout de composants

Un composant est une sorte d'élément d'interface utilisateur, tel qu'un bouton ou un champ de texte.

Créer un composant

La création de composants est presque identique à la création d'une fenêtre. Au lieu de créer un `JFrame` , vous créez ce composant. Par exemple, pour créer un `JButton` , procédez comme suit.

```
JButton button = new JButton();
```

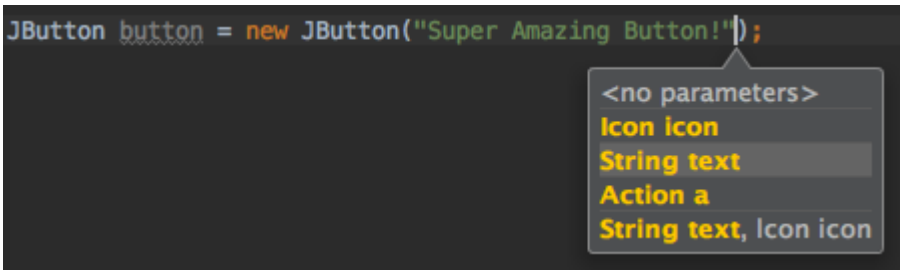
De nombreux composants peuvent recevoir des paramètres lors de leur création. Par exemple, un bouton peut être doté d'un texte à afficher.

```
JButton button = new JButton("Super Amazing Button!");
```

Si vous ne souhaitez pas créer de bouton, vous trouverez une liste de composants communs dans un autre exemple de cette page.

Les paramètres qui peuvent leur être transmis varient d'un composant à l'autre. Un bon moyen de vérifier ce qu'ils peuvent accepter est de regarder les paramètres de votre IDE (si vous en utilisez un). Les raccourcis par défaut sont répertoriés ci-dessous.

- IntelliJ IDEA - Windows / Linux: `CTRL + P`
- IntelliJ IDEA - OS X / macOS: `CMD + P`
- Eclipse: `CTRL + SHIFT + Space`
- NetBeans: `CTRL + P`



Affichage du composant

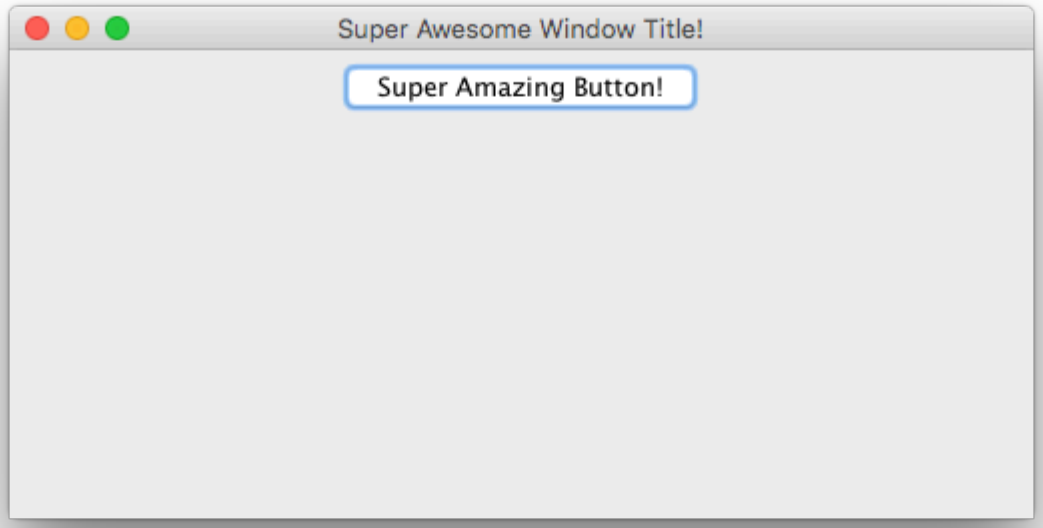
Une fois qu'un composant a été créé, vous définissez généralement ses paramètres. Après cela, vous devez le placer quelque part, par exemple sur votre `JFrame` ou sur votre volet de contenu si vous en avez créé un.

```
frame.add(button); //Add to your JFrame
//OR
pane.add(button); //Add to your content pane
//OR
myComponent.add(button); //Add to whatever
```

Exemple

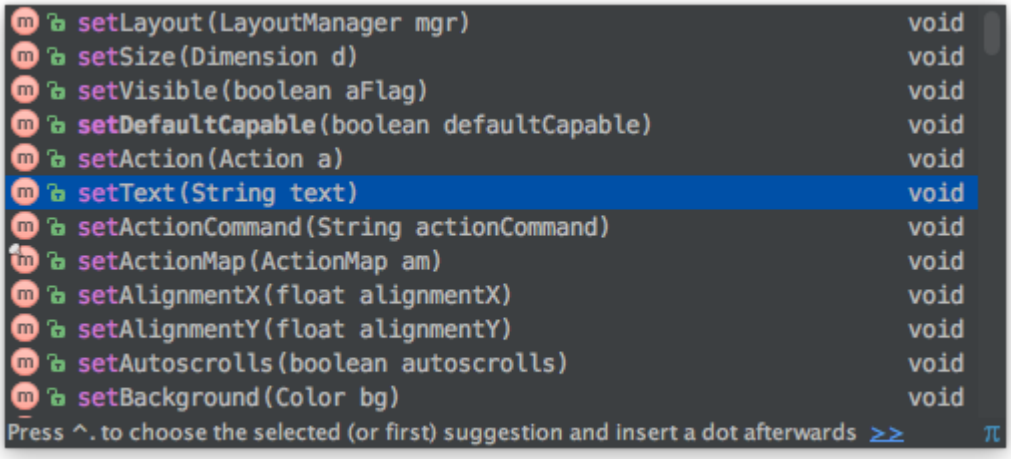
Voici un exemple de création d'une fenêtre, de définition d'un volet de contenu et d'ajout d'un bouton.

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create the content pane
frame.setContentPane(pane); //Set the content pane

JButton button = new JButton("Super Amazing Button!"); //Create the button
pane.add(button); //Add the button to the content pane

frame.setVisible(true); //Show the window
```



Définition des paramètres pour les composants

Les composants ont différents paramètres pouvant être définis pour eux. Ils varient d'un composant à l'autre. Par conséquent, un bon moyen de définir les paramètres des composants est de commencer à saisir `componentName.set` et de laisser les méthodes de saisie semi-automatique (si vous utilisez un IDE) de votre IDE suggérer. Le raccourci par défaut dans de nombreux IDE, s'il ne s'affiche pas automatiquement, est `CTRL + Space` .



Paramètres communs partagés entre tous les composants

La description	Méthode
Définit la plus petite taille possible du composant (uniquement si le gestionnaire de disposition respecte la propriété minimumSize)	`setMinimumSize(Dimension minimumSize)`

La description	Méthode
Définit la taille maximale du composant (uniquement si le gestionnaire de disposition respecte la propriété maximumSize)	`setMaximumSize(Dimension maximumSize)`
Définit la taille préférée du composant (uniquement si le gestionnaire de disposition respecte la propriété preferredSize)	`setPreferredSize(Dimension preferredSize)`
Affiche ou masque le composant	`setVisible(boolean aFlag)`
Définit si le composant doit répondre à une entrée utilisateur	`setEnabled(boolean enabled)`
Définit la police de texte	`setFont(Font font)`
Définit le texte de l'info-bulle	`setToolTipText(String text)`
Définit la couleur d'arrière-plan du composant	`setBackground(Color bg)`
Définit la couleur de premier plan (couleur de police) du composant	`setForeground(Color bg)`

Paramètres communs dans d'autres composants

Composants communs	La description	Méthode
`JLabel` , `JButton` , `JCheckBox` , `JRadioButton` , `JToggleButton` , `JMenu` , `JMenuItem` , `JTextArea` , `JTextField`	Définit le texte affiché	`setText(String text)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Définir une valeur numérique entre les valeurs min et max du composant	`setValue(int n)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Définit la plus petite valeur possible pour la propriété `value`	`setMinimum(int n)`
`JProgressBar` , `JScrollBar` , `JSlider` , `JSpinner`	Set est la plus grande valeur possible que la propriété `value` puisse être	`setMaxmimum(int n)`
`JCheckBox` , `JToggleBox`	Définir si la valeur est vraie ou fausse (par ex.: Une case à cocher doit-elle être cochée?)	`setSelected(boolean b)`

Composants communs

La description	Classe
Bouton	`JButton`
Case à cocher	`JCheckBox`
Menu déroulant / Zone de liste déroulante	`JComboBox`
Étiquette	`JLabel`
liste	`JList`
Barre de menu	`JMenuBar`
Menu dans une barre de menu	`JMenu`
Article dans un menu	`JMenuItem`
Panneau	`JPanel`
Barre de progression	`JProgressBar`
Bouton radio	`JRadioButton`
Barre de défilement	`JScrollBar`
Curseur	`JSlider`
Spinner / Number picker	`JSpinner`
Table	`JTable`
Arbre	`JTree`
Zone de texte / zone de texte multiligne	`JTextArea`
Champ de texte	`JTextField`
Barre d'outils	`JToolBar`

Création d'interfaces utilisateur interactives

Avoir un bouton là-bas est bien, mais à quoi ça sert si cliquer ne fait rien? `ActionListener` s sont utilisés pour indiquer à votre bouton ou à un autre composant de faire quelque chose lorsqu'il est activé.

L'ajout d' `ActionListener` s se fait comme tel.

```
buttonA.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        //Code goes here...
        System.out.println("You clicked the button!");
    }
});
```

Ou, si vous utilisez Java 8 ou supérieur ...

```
buttonA.addActionListener(e -> {
    //Code
    System.out.println("You clicked the button!");
});
```

Exemple (Java 8 et supérieur)

```
JFrame frame = new JFrame("Super Awesome Window Title!"); //Create the JFrame and give it a
title
frame.setSize(512, 256); //512 x 256px size
frame.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE); //Quit the application when the
JFrame is closed

JPanel pane = new JPanel(); //Create a pane to house all content
frame.setContentPane(pane);

JButton button = new JButton("Click me - I know you want to.");
button.addActionListener(e -> {
    //Code goes here
    System.out.println("You clicked me! Ouch.");
});
pane.add(buttonA);

frame.setVisible(true); //Show the window
```

Organisation des composants

L'ajout de composants les uns après les autres entraîne une interface utilisateur difficile à utiliser, car les composants sont tous **quelque part** . Les composants sont classés de haut en bas, chaque composant dans une "rangée" distincte.

Pour remédier à cela et vous fournir en tant que développeur une possibilité de mettre en page facilement des composants, Swing a `LayoutManager` .

Ces LayoutManagers sont abordés plus en détail dans Introduction aux gestionnaires de mise en page ainsi que dans les rubriques distinctes de Layout Manager:

- Disposition de la grille
- Disposition GridBag

Lire Utiliser Swing pour les interfaces utilisateur graphiques en ligne:
https://riptutorial.com/fr/swing/topic/2982/utiliser-swing-pour-les-interfaces-utilisateur-graphiques

Crédits

S. No	Chapitres	Contributeurs
1	Commencer avec le swing	Community, Freek de Bruijn, Petter Friberg, Vogel612, XavCo7
2	Disposition de la grille	Lukas Rotter, user6653173
3	Disposition GridBag	CraftedCart, Enwired, mayha, Vogel612
4	Gestion de la mise en page	explv, J Atkin, mayha, pietrocalzini, recke96, Squidward, XavCo7
5	Graphique	Adel Khial, Ashlyn Campbell, Squidward
6	JList	Andreas Fester, Squidward, user6653173
7	Les bases	DarkV1, DonyorM, elias, Robin, Squidward
8	MigLayout	hamena314, keuleJ
9	minuterie dans JFrame	SSD
10	Modèle MVP	avojak, ehzawad, Leonardo Pina, sjngm, Squidward
11	StyledDocument	DonyorM, Squidward
12	Swing Workers et l'EDT	dpr, isaias-b, rahul tyagi
13	Utilisation de l'apparence	Mikle Garin
14	Utiliser Swing pour les interfaces utilisateur graphiques	CraftedCart, mayha, Michael, Vogel612, Winter