Récapitulatif JAVA :

Table des matières

[JFrame : 3](#_Toc472084074)

[JPanel : 3](#_Toc472084075)

[Graphics : 3](#_Toc472084076)

[Animation : 4](#_Toc472084077)

[JButton : 5](#_Toc472084078)

[Agir avec le bouton : 5](#_Toc472084079)

[Thread : 7](#_Toc472084080)

[FORK/JOIN : 8](#_Toc472084081)

[Listes, JComboBox : 11](#_Toc472084082)

[CheckBox : JCheckBox : 13](#_Toc472084083)

[Radio : JRadioButton : 13](#_Toc472084084)

[Textes : JTextField : 14](#_Toc472084085)

[Texte avec réstriction : JFormattedTextField : 15](#_Toc472084086)

[Contrôle du clavier : interface KeyListener : 15](#_Toc472084087)

# JFrame :

Par défaut une fenêtre a une taille minimale et n’est pas visible.

import javax.swing.JFrame;

public class Fenetre extends JFrame {

public Fenetre(){

this.setTitle("Ma première fenêtre Java");

this.setSize(100, 150);

this.setLocationRelativeTo(null);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setContentPane(new Panneau());

this.setVisible(true);

}

}

# JPanel :

A pour but d’accueillir d’autres objets, c’est un container.

import java.awt.Color;

import javax.swing.JFrame;

import javax.swing.JPanel;

public class Fenetre extends JFrame {

public Fenetre(){

this.setTitle("Ma première fenêtre Java");

this.setSize(400, 100);

this.setLocationRelativeTo(null);

//Instanciation d'un objet JPanel

JPanel pan = new JPanel();

//Définition de sa couleur de fond

pan.setBackground(Color.ORANGE);

//On prévient notre JFrame que notre JPanel sera son content pane

this.setContentPane(pan);

this.setVisible(true);

}

}

# Graphics :

Permet de faire des dessins :

import java.awt.Graphics;

import javax.swing.JPanel;

public class Panneau extends JPanel {

public void paintComponent(Graphics g){

//x1, y1, width, height

g.drawRect(10, 10, 50, 60);

g.fillRect(65, 65, 30, 40);

}

}

# Animation :

On utilise une boucle infinie pour répéter l’animation :

private void go(){

//Les coordonnées de départ de notre rond

int x = pan.getPosX(), y = pan.getPosY();

//Le booléen pour savoir si l'on recule ou non sur l'axe x

boolean backX = false;

//Le booléen pour savoir si l'on recule ou non sur l'axe y

boolean backY = false;

//Dans cet exemple, j'utilise une boucle while

//Vous verrez qu'elle fonctionne très bien

while(true){

//Si la coordonnée x est inférieure à 1, on avance

if(x < 1)

backX = false;

//Si la coordonnée x est supérieure à la taille du Panneau moins la taille du rond, on recule

if(x > pan.getWidth()-50)

backX = true;

//Idem pour l'axe y

if(y < 1)

backY = false;

if(y > pan.getHeight()-50)

backY = true;

//Si on avance, on incrémente la coordonnée

//backX est un booléen, donc !backX revient à écrire

//if (backX == false)

if(!backX)

pan.setPosX(++x);

//Sinon, on décrémente

else

pan.setPosX(--x);

//Idem pour l'axe Y

if(!backY)

pan.setPosY(++y);

else

pan.setPosY(--y);

//On redessine notre Panneau

pan.repaint();

//Comme on dit : la pause s'impose ! Ici, trois millièmes de seconde

try {

Thread.sleep(3);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

# JButton :

Il existe plusieurs façons de les placer :

* BorderLayout (N-S-E-O)
* GridLayout (tableau)
* BoxLayout (alligner en ligne et colonne)
* CardLayout (les uns au-dessus des autres dans un certain ordre) GridBagLayout (façon bataille navale)

Exemple avec BorderLayout :

import java.awt.BorderLayout;

import javax.swing.JButton;

import javax.swing.JFrame;

public class Fenetre extends JFrame{

public Fenetre(){

this.setTitle("Bouton");

this.setSize(300, 300);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

//On définit le layout à utiliser sur le content pane

this.setLayout(new BorderLayout());

//On ajoute le bouton au content pane de la JFrame

//Au centre

this.getContentPane().add(new JButton("CENTER"), BorderLayout.CENTER);

//Au nord

this.getContentPane().add(new JButton("NORTH"), BorderLayout.NORTH);

//Au sud

this.getContentPane().add(new JButton("SOUTH"), BorderLayout.SOUTH);

//À l'ouest

this.getContentPane().add(new JButton("WEST"), BorderLayout.WEST);

//À l'est

this.getContentPane().add(new JButton("EAST"), BorderLayout.EAST);

this.setVisible(true);

}

}

# Agir avec le bouton :

Si on a plusieurs boutons, on les met dans un JPanel et on insère le JPanel dans le JFrame.

import java.awt.Color;

import java.awt.FontMetrics;

import java.awt.GradientPaint;

import java.awt.Graphics;

import java.awt.Graphics2D;

import java.awt.Image;

import java.awt.event.MouseEvent;

import java.awt.event.MouseListener;

import java.io.File;

import java.io.IOException;

import javax.imageio.ImageIO;

import javax.swing.JButton;

public class Bouton extends JButton implements MouseListener{

private String name;

private Image img;

public Bouton(String str){

super(str);

this.name = str;

try {

img = ImageIO.read(new File("fondBouton.png"));

} catch (IOException e) {

e.printStackTrace();

}

this.addMouseListener(this);

}

public void paintComponent(Graphics g){

Graphics2D g2d = (Graphics2D)g;

GradientPaint gp = new GradientPaint(0, 0, Color.blue, 0, 20, Color.cyan, true);

g2d.setPaint(gp);

g2d.drawImage(img, 0, 0, this.getWidth(), this.getHeight(), this);

g2d.setColor(Color.black);

//Objet permettant de connaître les propriétés d'une police, dont la taille

FontMetrics fm = g2d.getFontMetrics();

//Hauteur de la police d'écriture

int height = fm.getHeight();

//Largeur totale de la chaîne passée en paramètre

int width = fm.stringWidth(this.name);

//On calcule alors la position du texte, et le tour est joué

g2d.drawString(this.name, this.getWidth() / 2 - (width / 2), (this.getHeight() / 2) + (height / 4));

}

public void mouseClicked(MouseEvent event) {

//Inutile d'utiliser cette méthode ici

}

public void mouseEntered(MouseEvent event) {

//Nous changeons le fond de notre image pour le jaune lors du survol, avec le fichier fondBoutonHover.png

try {

img = ImageIO.read(new File("fondBoutonHover.png"));

} catch (IOException e) {

e.printStackTrace();

}

}

public void mouseExited(MouseEvent event) {

//Nous changeons le fond de notre image pour le vert lorsque nous quittons le bouton, avec le fichier fondBouton.png

try {

img = ImageIO.read(new File("fondBouton.png"));

} catch (IOException e) {

e.printStackTrace();

}

}

public void mousePressed(MouseEvent event) {

//Nous changeons le fond de notre image pour le jaune lors du clic gauche, avec le fichier fondBoutonClic.png

try {

img = ImageIO.read(new File("fondBoutonClic.png"));

} catch (IOException e) {

e.printStackTrace();

}

}

public void mouseReleased(MouseEvent event) {

//Nous changeons le fond de notre image pour l'orange lorsque nous relâchons le clic avec le fichier fondBoutonHover.png si la souris est toujours sur le bouton

if((event.getY() > 0 && event.getY() < this.getHeight()) && (event.getX() > 0 && event.getX() < this.getWidth())){

try {

img = ImageIO.read(new File("fondBoutonHover.png"));

} catch (IOException e) {

e.printStackTrace();

}

}

//Si on se trouve à l'extérieur, on dessine le fond par défaut

else{

try {

img = ImageIO.read(new File("fondBouton.png"));

} catch (IOException e) {

e.printStackTrace();

}

}

}

}

# Thread :

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class Fenetre extends JFrame{

private Panneau pan = new Panneau();

private JButton bouton = new JButton("Go");

private JButton bouton2 = new JButton("Stop");

private JPanel container = new JPanel();

private JLabel label = new JLabel("Le JLabel");

private int compteur = 0;

private boolean animated = true;

private boolean backX, backY;

private int x, y;

private Thread t;

public Fenetre(){

//Le constructeur n'a pas changé

}

private void go(){

//La méthode n'a pas changé

}

public class BoutonListener implements ActionListener{

public void actionPerformed(ActionEvent arg0) {

animated = true;

t = new Thread(new PlayAnimation());

t.start();

bouton.setEnabled(false);

bouton2.setEnabled(true);

}

}

class Bouton2Listener implements ActionListener{

public void actionPerformed(ActionEvent e) {

animated = false;

bouton.setEnabled(true);

bouton2.setEnabled(false);

}

}

class PlayAnimation implements Runnable{

public void run() {

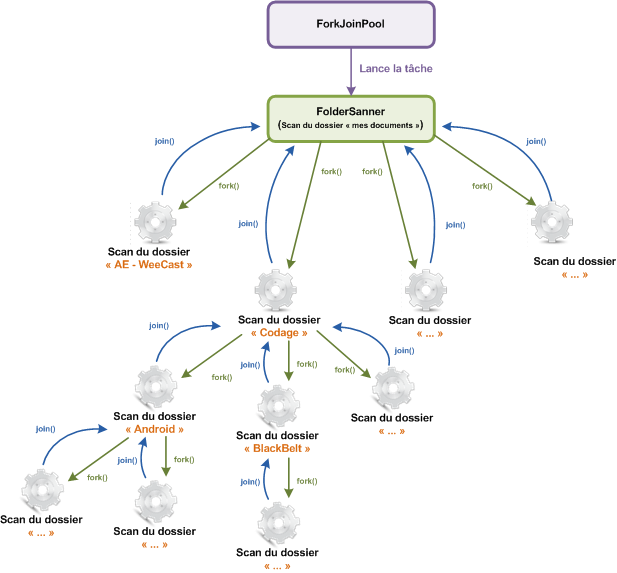
go();

}

}

}

# FORK/JOIN :



public class FolderScanner extends RecursiveTask<Long>{

private Path path = null;

private String filter = "\*";

private long result = 0;

public FolderScanner(){ }

public FolderScanner(Path p, String f){

path = p;

filter = f;

}

/\*\*

\* Notre méthode de scan en mode mono thread

\* @throws ScanException

\*/

public long sequentialScan() throws ScanException{

//Si le chemin n'est pas valide, on lève une exception

if(path == null || path.equals(""))

throw new ScanException("Chemin à scanner non valide (vide ou null) !");

System.out.println("Scan du dossier : " + path + " à la recherche des fichiers portant l'extension " + this.filter);

//On liste maintenant le contenu du répertoire pour traiter les sous-dossiers

try(DirectoryStream<Path> listing = Files.newDirectoryStream(path)){

for(Path nom : listing){

//S'il s'agit d'un dossier, on le scan grâce à notre objet

if(Files.isDirectory(nom.toAbsolutePath())){

FolderScanner f = new FolderScanner(nom.toAbsolutePath(), this.filter);

result += f.sequentialScan();

}

}

} catch (IOException e) { e.printStackTrace(); }

//Maintenant, on filtre le contenu de ce même dossier sur le filtre défini

try(DirectoryStream<Path> listing = Files.newDirectoryStream(path, this.filter)){

for(Path nom : listing){

//Pour chaque fichier correspondant, on incrémente notre compteur

result++;

}

} catch (IOException e) { e.printStackTrace(); }

return result;

}

/\*\*

\* Méthode que nous allons utiliser pour les traitements

\* en mode parallèle.

\* @throws ScanException

\*/

public long parallelScan() throws ScanException{

//List d'objet qui contiendra les sous-tâches créées et lancées

List<FolderScanner> list = new ArrayList<>();

//Si le chemin n'est pas valide

if(path == null || path.equals(""))

throw new ScanException("Chemin à scanner non valide (vide ou null) !");

System.out.println("Scan du dossier : " + path + " a la recherche des fichiers portant l'extension " + this.filter);

//Nous listons, comme précédemment, le contenu du répertoire

try(DirectoryStream<Path> listing = Files.newDirectoryStream(path)){

//On parcourt le contenu

for(Path nom : listing){

//S'il s'agit d'un dossier, on crée une sous-tâche

if(Files.isDirectory(nom.toAbsolutePath())){

//Nous créons donc un nouvel objet FolderScanner

//Qui se chargera de scanner ce dossier

FolderScanner f = new FolderScanner(nom.toAbsolutePath(), this.filter);

//Nous l'ajoutons à la liste des tâches en cours pour récupérer le résultat plus tard

list.add(f);

//C'est cette instruction qui lance l'action en tâche de fond

f.fork();

}

}

} catch (IOException e) {

e.printStackTrace();

}

//On compte maintenant les fichiers, correspondant au filtre, présents dans ce dossier

try(DirectoryStream<Path> listing = Files.newDirectoryStream(path, this.filter)){

for(Path nom : listing){

result++;

}

} catch (IOException e) {

e.printStackTrace();

}

//Et, enfin, nous récupérons le résultat de toutes les tâches de fond

for(FolderScanner f : list)

result += f.join();

//Nous renvoyons le résultat final

return result;

}

/\*\*

\* Méthode qui défini l'action à faire

\* dans notre cas, nous lan çons le scan en mode parallèles

\*/

protected Long compute() {

long resultat = 0;

try {

resultat = this.parallelScan();

} catch (ScanException e) {

e.printStackTrace();

}

return resultat;

}

public long getResultat(){ return this.result; }

}

Et voici maintenant notre classe de test :

import java.nio.file.Path;

import java.nio.file.Paths;

import java.util.concurrent.ForkJoinPool;

public class Main {

public static void main(String[] args) {

Path chemin = Paths.get("E:\\Mes Documents");

String filtre = "\*.psd";

//Création de notre tâche principale qui se charge de découper son travail en sous-tâches

FolderScanner fs = new FolderScanner(chemin, filtre);

//Nous récupérons le nombre de processeurs disponibles

int processeurs = Runtime.getRuntime().availableProcessors();

//Nous créons notre pool de thread pour nos tâches de fond

ForkJoinPool pool = new ForkJoinPool(processeurs);

Long start = System.currentTimeMillis();

//Nous lançons le traitement de notre tâche principale via le pool

pool.invoke(fs);

Long end = System.currentTimeMillis();

System.out.println("Il y a " + fs.getResultat() + " fichier(s) portant l'extension " + filtre);

System.out.println("Temps de traitement : " + (end - start));

}

}

# Listes, JComboBox :

Deux façons :

//Les imports restent inchangés

public class Fenetre extends JFrame {

//Les variables d'instance restent inchangées

public Fenetre(){

//…

combo.setPreferredSize(new Dimension(100, 20));

combo.addItem("Option 1");

combo.addItem("Option 2");

combo.addItem("Option 3");

combo.addItem("Option 4");

//…

}

}

Ou

String[] tab = {"Option 1", "Option 2", "Option 3", "Option 4"};

combo = new JComboBox(tab);

##### Classe Fenetre

. . .

combo.addItem("ROND");

combo.addItem("CARRE");

combo.addItem("TRIANGLE");

combo.addItem("ETOILE");

combo.addActionListener(new FormeListener());

JPanel top = new JPanel();

top.add(label);

top.add(combo);

container.add(top, BorderLayout.NORTH);

this.setContentPane(container);

this.setVisible(true);

}

private void go(){

x = pan.getPosX();

y = pan.getPosY();

while(this.animated){

if(x < 1) backX = false;

if(x > pan.getWidth() - 50) backX = true;

if(y < 1) backY = false;

if(y > pan.getHeight() - 50) backY = true;

if(!backX) pan.setPosX(++x);

else pan.setPosX(--x);

if(!backY) pan.setPosY(++y);

else pan.setPosY(--y);

pan.repaint();

try {

Thread.sleep(3);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

//Classe écoutant notre bouton

public class BoutonListener implements ActionListener{

public void actionPerformed(ActionEvent arg0) {

animated = true;

t = new Thread(new PlayAnimation());

t.start();

bouton.setEnabled(false);

bouton2.setEnabled(true);

}

}

class Bouton2Listener implements ActionListener{

public void actionPerformed(ActionEvent e) {

animated = false;

bouton.setEnabled(true);

bouton2.setEnabled(false);

}

}

class PlayAnimation implements Runnable{

public void run() {

go();

}

}

class FormeListener implements ActionListener{

public void actionPerformed(ActionEvent e) {

//La méthode retourne un Object puisque nous passons des Object dans une liste

//Il faut donc utiliser la méthode toString() pour retourner un String (ou utiliser un cast)

pan.setForme(combo.getSelectedItem().toString());

}

}

}

# CheckBox : JCheckBox :

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JCheckBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

public class Fenetre extends JFrame {

private JPanel container = new JPanel();

private JCheckBox check1 = new JCheckBox("Case 1");

private JCheckBox check2 = new JCheckBox("Case 2");

public Fenetre(){

this.setTitle("Animation");

this.setSize(300, 300);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

container.setBackground(Color.white);

container.setLayout(new BorderLayout());

JPanel top = new JPanel();

check1.addActionListener(new StateListener());

check2.addActionListener(new StateListener());

top.add(check1);

top.add(check2);

container.add(top, BorderLayout.NORTH);

this.setContentPane(container);

this.setVisible(true);

}

class StateListener implements ActionListener{

public void actionPerformed(ActionEvent e) {

System.out.println("source : " + ((JCheckBox)e.getSource()).getText() + " - état : " + ((JCheckBox)e.getSource()).isSelected());

}

}

}

# Radio : JRadioButton :

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.JCheckBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JRadioButton;

public class Fenetre extends JFrame {

private JPanel container = new JPanel();

private JRadioButton jr1 = new JRadioButton("Radio 1");

private JRadioButton jr2 = new JRadioButton("Radio 2");

public Fenetre(){

this.setTitle("Animation");

this.setSize(300, 300);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

container.setBackground(Color.white);

container.setLayout(new BorderLayout());

JPanel top = new JPanel();

jr1.addActionListener(new StateListener());

jr2.addActionListener(new StateListener());

top.add(jr1);

top.add(jr2);

container.add(top, BorderLayout.NORTH);

this.setContentPane(container);

this.setVisible(true);

}

class StateListener implements ActionListener{

public void actionPerformed(ActionEvent e) {

System.out.println("source : " + ((JRadioButton)e.getSource()).getText() + " - état : " + ((JRadioButton)e.getSource()).isSelected());

}

}

}

# Textes : JTextField :

//Les imports habituels

import javax.swing.JTextField;

public class Fenetre extends JFrame {

private JPanel container = new JPanel();

private JTextField jtf = new JTextField("Valeur par défaut");

private JLabel label = new JLabel("Un JTextField");

public Fenetre(){

this.setTitle("Animation");

this.setSize(300, 300);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

container.setBackground(Color.white);

container.setLayout(new BorderLayout());

JPanel top = new JPanel();

Font police = new Font("Arial", Font.BOLD, 14);

jtf.setFont(police);

jtf.setPreferredSize(new Dimension(150, 30));

jtf.setForeground(Color.BLUE);

top.add(label);

top.add(jtf);

container.add(top, BorderLayout.NORTH);

this.setContentPane(container);

this.setVisible(true);

}

}

# Texte avec réstriction : JFormattedTextField :

//Les imports habituels

public class Fenetre extends JFrame {

private JPanel container = new JPanel();

**private JFormattedTextField jtf = new JFormattedTextField(NumberFormat.getIntegerInstance**());

private JFormattedTextField jtf2 = new JFormattedTextField(NumberFormat.getPercentInstance());

private JLabel label = new JLabel("Un JTextField");

private JButton b = new JButton ("OK");

public Fenetre(){

this.setTitle("Animation");

this.setSize(300, 300);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

container.setBackground(Color.white);

container.setLayout(new BorderLayout());

JPanel top = new JPanel();

Font police = new Font("Arial", Font.BOLD, 14);

jtf.setFont(police);

jtf.setPreferredSize(new Dimension(150, 30));

jtf.setForeground(Color.BLUE);

jtf2.setPreferredSize(new Dimension(150, 30));

b.addActionListener(new BoutonListener());

top.add(label);

top.add(jtf);

top.add(jtf2);

top.add(b);

this.setContentPane(top);

this.setVisible(true);

}

class BoutonListener implements ActionListener{

public void actionPerformed(ActionEvent e) {

System.out.println("TEXT : jtf " + jtf.getText());

System.out.println("TEXT : jtf2 " + jtf2.getText());

}

}

}

# Contrôle du clavier : interface KeyListener :

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.Font;

import java.awt.event.KeyEvent;

import java.awt.event.KeyListener;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import javax.swing.JTextField;

public class Fenetre extends JFrame {

private JPanel container = new JPanel();

private JTextField jtf;

private JLabel label = new JLabel("Téléphone FR");

private JButton b = new JButton ("OK");

public Fenetre(){

this.setTitle("Animation");

this.setSize(300, 150);

this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

this.setLocationRelativeTo(null);

container.setBackground(Color.white);

container.setLayout(new BorderLayout());

jtf = new JTextField();

JPanel top = new JPanel();

Font police = new Font("Arial", Font.BOLD, 14);

jtf.setFont(police);

jtf.setPreferredSize(new Dimension(150, 30));

jtf.setForeground(Color.BLUE);

//On ajoute l'écouteur à notre composant

jtf.addKeyListener(new ClavierListener());

top.add(label);

top.add(jtf);

top.add(b);

this.setContentPane(top);

this.setVisible(true);

}

class ClavierListener implements KeyListener{

public void keyPressed(KeyEvent event) {

System.out.println("Code touche pressée : " + event.getKeyCode() + " - caractère touche pressée : " + event.getKeyChar());

pause();

}

public void keyReleased(KeyEvent event) {

System.out.println("Code touche relâchée : " + event.getKeyCode() + " - caractère touche relâchée : " + event.getKeyChar());

pause();

}

public void keyTyped(KeyEvent event) {

System.out.println("Code touche tapée : " + event.getKeyCode() + " - caractère touche tapée : " + event.getKeyChar());

pause();

}

}

private void pause(){

try {

Thread.sleep(1000);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

public static void main(String[] args){

new Fenetre();

}

}