ircam Centre Pompidou

UPMC SORBONNE UNIVERSITÉS

# JAVA
# - Interface Graphique 2 -

Ninon Devis: ninon.devis@ircam.fr

Philippe Esling: esling@ircam.fr

License 3 Professionnelle - Multimédia

# Plan du cours

*Mini-Projet tout au long du cours: Création d'un Paint*

I. Rappels: `JFrame, JPanel, JButton, JLabel`
II. Action Listener
III. Mouse Listener
IV. Dessin dans le Panel
V. Classe Anonyme
VI. Dessin Libre
VII. Dialogue et Toolbox

# Rappels

- Dans le main, ajouter une frame avec titre. Ne pas oublier:
  - `setDefaultCloseOperation()`
  - `setVisible()`
- Que doit-on ajouter ensuite immédiatement dans l'optique de pouvoir dessiner dedans ?
- Créer un "main" `JPanel` et lui assigner un layout.
- Ajouter deux labels et deux boutons de votre choix à ce panel.

# Rappels

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

public class Main {

    public static void main(String[] args) {
        JFrame window = new JFrame("Java Paint");
        window.setSize(1000, 1000);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);
        // Always add a "main panel" in the frame (default container)
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new FlowLayout());
        // Labels and buttons
        JLabel mainLabel = new JLabel("Welcome to my Paint");
        JLabel secondLabel = new JLabel("Catch me");
        JButton button1 = new JButton("I am first");
        JButton button2 = new JButton("I am second");
        // Add objects to the panel
        mainPanel.add(mainLabel);
        mainPanel.add(button1);
        mainPanel.add(button2);
        mainPanel.add(secondLabel);
        window.add(mainPanel);
    }
}
```

# Action Listener

*Objectif: Lorsque l'utilisateur clique sur un bouton, le second label est remplacé par le label du bouton cliqué.*

- Que doit-on utiliser pour récupérer l'action de clic de l'utilisateur ?
- Créer une nouvelle classe `ButtonAction` qui implémente `ActionListener`.
- Afficher sur la console les informations contenues dans `getActionCommand()`, `paramString()` et `getSource()`.
- Ajouter au main `window.validate()` afin de forcer la mise à jour de l'affichage.
- Quelles informations s'affichent dans la console ?

```
I am first

ACTION_PERFORMED,cmd=I am first,when=1605115909925,modifiers=Button1

javax.swing.JButton[,432,5,96x25,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundBo
rderUIResource@6b05d69,flags=296,maximumSize=,minimumSize=,preferredSize=,defaultIcon=,disabledIcon=,disabledSelec
tedIcon=,margin=javax.swing.plaf.InsetsUIResource[top=2,left=14,bottom=2,right=14],paintBorder=true,paintFocus=tru
e,pressedIcon=,rolloverEnabled=true,rolloverIcon=,rolloverSelectedIcon=,selectedIcon=,text=I am
first,defaultCapable=true]


I am second

ACTION_PERFORMED,cmd=I am second,when=1605115924086,modifiers=Button1

javax.swing.JButton[,533,5,118x25,alignmentX=0.0,alignmentY=0.5,border=javax.swing.plaf.BorderUIResource$CompoundB
orderUIResource@6b05d69,flags=296,maximumSize=,minimumSize=,preferredSize=,defaultIcon=,disabledIcon=,disabledSele
ctedIcon=,margin=javax.swing.plaf.InsetsUIResource[top=2,left=14,bottom=2,right=14],paintBorder=true,paintFocus=tr
ue,pressedIcon=,rolloverEnabled=true,rolloverIcon=,rolloverSelectedIcon=,selectedIcon=,text=I am
second,defaultCapable=true]
```

# Action Listener

*Objectif: Lorsque l'utilisateur clique sur un bouton, le second label est remplacé par le label du bouton cliqué.*

## Classe ButtonAction:

```java
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class ButtonAction implements ActionListener
{
    @Override
    public void actionPerformed(ActionEvent actionEvent)
    {
        System.out.println(actionEvent.getActionCommand());
        System.out.println(actionEvent.paramString());
        System.out.println(actionEvent.getSource());
    }
}
```

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

public class Main {

    public static void main(String[] args) {
        JFrame window = new JFrame("Java Paint");
        window.setSize(1000, 1000);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setVisible(true);
        // Always add a "main panel" in the frame (default container)
        JPanel mainPanel = new JPanel();
        mainPanel.setLayout(new FlowLayout());
        // Labels and buttons
        JLabel mainLabel = new JLabel("Welcome to my Paint");
        JLabel secondLabel = new JLabel("Catch me");
        JButton button1 = new JButton("I am first");
        JButton button2 = new JButton("I am second");
        // Add response object to the click on any button
        ButtonAction infoButtonAction = new ButtonAction();
        // Add the same action on the two buttons
        button1.addActionListener(infoButtonAction);
        button2.addActionListener(infoButtonAction);
        // Add objects to the panel
        mainPanel.add(mainLabel);
        mainPanel.add(button1);
        mainPanel.add(button2);
        mainPanel.add(secondLabel);
        window.add(mainPanel);
        window.validate();
    }
}
```

# Action Listener

Objectif: Lorsque l'utilisateur clique sur un bouton, le second label est remplacé par le label du bouton cliqué.

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

public class ButtonAction implements ActionListener
{
    private JLabel label;

    public ButtonAction(JLabel lab){
        this.label = lab;
    }

    @Override
    public void actionPerformed(ActionEvent actionEvent)
    {
        System.out.println(actionEvent.getActionCommand());
        System.out.println(actionEvent.paramString());
        System.out.println(actionEvent.getSource());
        if (actionEvent.getSource().getClass() == JButton.class) {
            JButton btn = ((JButton) actionEvent.getSource());
            this.label.setText(btn.getText());
        }
    }
}
```

Que restera t-il à changer dans le main ?

```java
ButtonAction infoButtonAction = new ButtonAction(secondLabel);
```

# Mouse Listener

- L'interface `MouseListener` propose 5 méthodes représentant les différentes actions d'une souris.

```java
// ... in main
JPanel main = new JPanel(new FlowLayout());
main.addMouseListener(new PrintCoordinate());
// ...

class PrintCoordinate implements MouseListener
{
    public void mouseClicked(MouseEvent e)
    {
        System.out.println(e.getPoint());
    }
    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
}
```

- L'objet passé en paramètre, `MouseEvent`, contient entre autres:
  - `getClickCount()`, nombre de cliques pour cet événement
  - `getPoint()`, les coordonnées du clique.
  - …

# Mouse Listener

*Objectif: Afficher dans la console les coordonnées de la souris lorsqu'elle entre dans le panel, lorsqu'elle en sort et lorsqu'elle est cliquée.*

- Créer une nouvelle classe `MousePanelInfo` qui implémente `MouseListener`.
- Quelles sont les trois méthodes à surcharger pour notre objectif?
- Ajouter au main le `MouseListener` qui vient d'être créé.
- Vous devriez obtenir: (cf .getX et .getY !)

```
ENTERED
x: 938 - y : 271

CLICKED
x: 350 - y : 452

EXITED
x: 460 - y : 973
```

# Mouse Listener

*Objectif: Afficher dans la console les coordonnées de la souris lorsqu'elle entre dans le panel, lorsqu'elle en sort et lorsqu'elle est cliquée.*

```java
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;

public class MousePanelInfo implements MouseListener {
    @Override
    public void mouseClicked(MouseEvent mouseEvent) {
        System.out.println("CLICKED");
        System.out.println("x: " + mouseEvent.getX() + " - y : " + mouseEvent.getY());
    }

    @Override
    public void mousePressed(MouseEvent mouseEvent) {
    }

    @Override
    public void mouseReleased(MouseEvent mouseEvent) {
    }

    @Override
    public void mouseEntered(MouseEvent mouseEvent) {
        System.out.println("ENTERED");
        System.out.println("x: " + mouseEvent.getX() + " - y : " + mouseEvent.getY());
    }

    @Override
    public void mouseExited(MouseEvent mouseEvent) {
        System.out.println("EXITED");
        System.out.println("x: " + mouseEvent.getX() + " - y : " + mouseEvent.getY());
    }
}
```

Et ajouter simplement dans le main:

```java
mainPanel.addMouseListener(new MousePanelInfo());
```

# Mouse Listener

*Objectif: Afficher dans la console les coordonnées de la souris lorsqu'elle se déplace et lorsqu'elle est cliquée et déplacée.*

- Créer une nouvelle classe `MousePanelDraw` qui implémente `MouseMotionListener`.
- Sur la base de l'exemple précédant, répondre à l'objectif.

```java
import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;

public class MousePanelDraw implements MouseMotionListener {

    @Override
    public void mouseDragged(MouseEvent mouseEvent) {
        System.out.println("Dragged");
        System.out.println("x: " + mouseEvent.getX() + " - y : " + mouseEvent.getY());
    }

    @Override
    public void mouseMoved(MouseEvent mouseEvent) {
        System.out.println("Moved");
        System.out.println("x: " + mouseEvent.getX() + " - y : " + mouseEvent.getY());
    }
}
```

Et dans le main: `mainPanel.addMouseMotionListener(new MousePanelDraw());`

# JPanel

*Objectif: Dessiner dans le JPanel des formes géométriques*

- A la place de votre objet `JPanel`, créer une classe `PaintPanel` qui étend `JPanel`.
- Certaines méthodes peuvent alors être surchargées, et notamment paintComponent qui permet d'ajouter des composants graphiques.
- Vous pouvez y ajouter des String, définir la couleur, dessiner des formes géométriques...
- Ajouter un rectangle rouge dans votre panel avec un message de votre choix.

# JPanel

*Objectif: Dessiner dans le JPanel des formes géométriques*

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

public class PaintPanel extends JPanel
{
    protected int x, y;

    public PaintPanel() {
        this.x = 50;
        this.y = 50;
    }

    @Override
    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        // Add a String, set the color to red and draw the rectangle
        g.drawString("Add a rectangle here", 55, 65);
        g.setColor( Color.RED );
        g.drawRect(this.x, this.y,200,50);
    }
```

# JPanel

*Objectif: Dessiner dans le JPanel des formes géométriques à l'aide de la souris*

- On veut pouvoir changer les coordonnées du rectangle par la position de la souris.
- Ajouter une nouvelle méthode `mouseClic()` dans votre classe `PaintPanel` qui prend en paramètre un événement de la souris et agit sur les coordonnées du rectangle.
- Où faut-il ensuite appeler cette nouvelle méthode ?

# JPanel

*Objectif: Dessiner dans le JPanel des formes géométriques à l'aide de la souris*

## Dans `PaintPanel`:

```
19      public void mouseClick(MouseEvent me)
20      {
21          this.x = me.getX();
22          this.y = me.getY();
23          this.repaint();
24      }
25  }
```

Permet de valider et dessiner

## Dans `MousePanelInfo`:

```
1  import java.awt.event.MouseEvent;
2  import java.awt.event.MouseListener;
3
4  public class MousePanelInfo implements MouseListener {
5      @Override
6      public void mouseClicked(MouseEvent mouseEvent) {
7          System.out.println("CLICKED");
8          System.out.println("x: " + mouseEvent.getX() + " - y : " + mouseEvent.getY());
9          if (mouseEvent.getSource().getClass() == PaintPanel.class){
10             PaintPanel pn = (PaintPanel)mouseEvent.getSource();
11             pn.mouseClick(mouseEvent);
12         }
13     }
```

# Classes de formes

*Objectif: Pouvoir ajouter facilement d'autres formes*

- Comment faire pour pouvoir ensuite ajouter d'autres types de dessin ?
  - Ajoutez une arrayList des formes
  - Une méthode addShape() qui remplira la liste
  - Surchargez la méthode paintComponent en conséquence.
- Implémentez une classe abstraite Shape: quels seront ses attributs?
- Implémentez la classe fille `Rectangle`.

```
1  public class PaintPanel extends JPanel
2  {
3      protected int x, y;
4      protected ArrayList<Shape> shapes;
5
6      public PaintPanel() {
7          this.x = 50;
8          this.y = 50;
9          this.shapes = new ArrayList<Shape>();
10     }
11
12     @Override
13     protected void paintComponent(Graphics g)
14     {
15         super.paintComponent(g);
16         g.setColor( Color.BLACK );
17         for (Shape s : shapes)
18             s.draw(g);
19     }
20
21     protected void addShape(Shape f)
22     {
23         this.shapes.add(f);
24         this.repaint();
25     }
```

# Classes de formes

*Objectif: Pouvoir ajouter facilement d'autres formes*

Classe Shape:

```java
public abstract class Shape
{
    protected int x, y;
    protected Color color;

    public Shape(int x, int y, Color c)
    {
        this.x = x;
        this.y = y;
        this.color = c;
    }

    public abstract void draw(Graphics g);
}
```

Classe RectangleShape:

```java
public class RectangleShape extends Shape
{
    protected int width, height;

    public RectangleShape (int x, int y, Color c, int w, int h)
    {
        super(x, y, c);
        this.width = w;
        this.height = h;
    }

    @Override
    public void draw(Graphics g)
    {
        g.setColor(this.color);
        g.drawRect(this.x, this.y, this.width, this.height);
    }
}
```

# Classes anonymes

*Objectif: Attribuer le dessin des formes aux boutons*

- Plutôt que de créer des classes de comportement pour chacun des boutons, on utilise des classes anonymes dans le main.
- Surtout utilisées pour implémenter les méthodes d'une interface Listener.
- Il s'agit d'une classe dont les méthodes sont surchargées dans un bloc, et dont la portée est donc limitée au bloc.
- Les méthodes sont redéfinies directement après l'instanciation d'un objet, leur nouveau comportement ne s'appliquera qu'à cet objet.

# Classes anonymes

*Objectif: Attribuer le dessin des formes aux boutons*

- Pour affecter au `button1` le dessin du rectangle, on définit la classe anonyme dans le main de la manière suivante:

```
1 JButton button1 = new JButton("Rectangle");
2 button1.addActionListener(new ActionListener(){
3     @Override
4     public void actionPerformed(ActionEvent actionEvent) {
5         mainPanel.addShape(new RectangleShape(100, 200, new Color(0,0,0), 50, 70));
6     }
7 });
```

- Attention à bien affecter le type statique `PaintPanel` à l'objet `mainPanel` afin qu'il puisse surcharger les méthodes à l'exécution.

```
1 PaintPanel mainPanel = new PaintPanel();
```

# Classes anonymes

*Objectif: Ajouter les formes par clic de la souris*

- Comment modifier `mouseClick` dans `PaintPanel` pour dessiner l'élément à l'emplacement du clic ?
- Modifier également la classe `Shape` en conséquence.

Dans `PaintPanel`:

```
1  public void mouseClick(MouseEvent me)
2  {
3      this.formes.get(this.formes.size() - 1).mouseClick(me);
4      this.repaint();
5  }
```

Dans `Shape`:

```
13     public void mouseClick(MouseEvent me)
14     {
15         this.x = me.getX();
16         this.y = me.getY();
17     }
```

# Dessin Libre

*Objectif: Simuler le pinceau pour du dessin libre*

- Quelle forme choisir pour pouvoir faire du dessin libre ? Comment implémenter notre objectif ?
- Créer une nouvelle classe `FreeHand` qui étend `Shape`.
- Celle-ci permettra de créer une ArrayList de points et de dessiner des lignes entre chaque point de la liste, traçant ainsi une simulation de dessin à main levé.
- Il est possible de définir une classe `Point` dans la classe `FreeHand`: c'est une classe interne.
- Une classe interne peut avoir accès aux méthodes et aux attributs de la classe englobante.

```
1 class Englobante {
2     int a;
3     class Interne{
4         . . .
5         Englobante.this.a = 12
6     }
7     . . .
8 }
```

# Dessin Libre

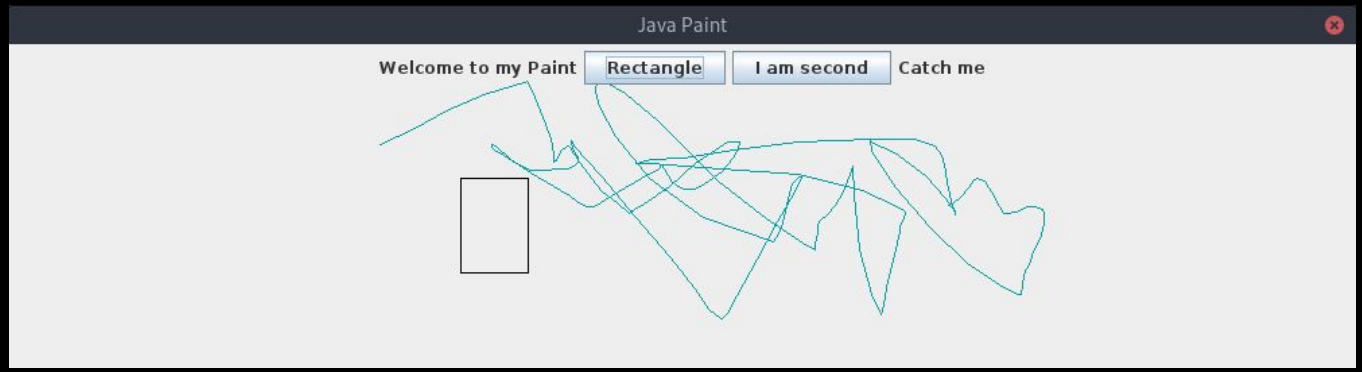
```java
public class FreeHandShape extends Shape {
    private class Point
    {
        public int x, y;
        public Point(int x, int y)
        {
            this.x = x;
            this.y = y;
        }
    }

    public ArrayList<Point> points;

    public FreeHandShape(Color c)
    {
        super(0, 0, c);
        points = new ArrayList<Point>();
    }

    public void addPoint(int x, int y)
    {
        this.points.add(new Point(x, y));
    }

    @Override
    public void draw(Graphics g) {
        for (int i = 0; i < points.size() - 1; i++) {
            Point p1 = points.get(i);
            Point p2 = points.get(i + 1);
            g.setColor(this.color);
            g.drawLine(p1.x, p1.y, p2.x, p2.y);
        }
    }
}
```

Classe interne

# Dessin Libre

*Objectif: Simuler le pinceau pour du dessin libre*

- Modifier à présent le PaintPanel, notamment:
  - ajouter la possibilité de dessiner en FreeHand à paintComponent()
  - ajouter une méthode addFreeHand
- Ajouter enfin au main un nouvel objet MouseMotionListener qui surcharge les méthodes qui nous intéressent grâce à une classe anonyme.



# Dessin Libre

```java
public class PaintPanel extends JPanel
{
    protected int x, y;
    protected ArrayList<Shape> shapes;
    protected FreeHandShape drawing;


    public PaintPanel() {
        this.x = 50;
        this.y = 50;
        this.shapes = new ArrayList<Shape>();
        this.drawing = new FreeHandShape(new Color(4, 159, 164));
    }

    @Override
    protected void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        // Custom code to paint all the Rectangles from the list
        g.setColor( Color.BLACK );
        for (Shape s : shapes)
            s.draw(g);
        this.drawing.draw(g);
    }

    protected void addShape(Shape f)
    {
        this.shapes.add(f);
        this.repaint();
    }

    protected void addFreeHand(int x, int y)
    {
        this.drawing.addPoint(x, y);
        this.repaint();
    }

    public void mouseClick(MouseEvent me)
    {
        this.shapes.get(this.shapes.size() - 1).mouseClick(me);
        this.repaint();
    }
}
```

# Dessin Libre

Dans le main:

```
mainPanel.addMouseMotionListener(new MouseMotionListener() {
    @Override
    public void mouseDragged(MouseEvent mouseEvent) {
        mainPanel.addFreeHand(mouseEvent.getX(), mouseEvent.getY());
    }
    @Override
    public void mouseMoved(MouseEvent mouseEvent) {}
});
```

# JDialog

*Objectif: Ajouter une boîte de dialogue à notre Paint*


- Créer une nouvelle classe `DialogBox` avec un membre `JDialog`.
- Son constructeur contiendra:
  - Une nouvelle `JFrame`
  - La JDialog
  - Un layout pour la `JDialog`
  - Un bouton
  - Une classe anonyme qui permet de fermer la `DialogBox` si l'utilisateur clic sur le bouton
  - Un label
  - N'oubliez pas de paramétrer la taille et le `setVisible`.
- Ajouter dans le main votre JDialog

```
1 new DialogBox();
```

```
1  public class DialogBox
2  {
3      private static JDialog d;
4
5      public DialogBox()
6      {
7          JFrame f = new JFrame();
8          d = new JDialog(f, "Welcome to my custom Paint", true);
9          d.setLayout(new FlowLayout());
10         JButton b = new JButton("Yep");
11         b.addActionListener(new ActionListener() {
12             public void actionPerformed(ActionEvent e) {
13                 DialogBox.d.setVisible(false);
14             }
15         });
16         d.add(new JLabel("Are you an artist ?"));
17         d.add(b);
18         d.setSize(300, 200);
19         d.setVisible(true);
20     }
21 }
```

# Mini-Projet

*Poursuivre votre Paint*

- Ajouter une option pour tracer des cercles.
- Faites en sorte que le dessin à main levé s'arrête lorsque vous arrêtez de cliquer et recommence sans lien lors d'un nouveau clic.
- Ajoutez des boîtes de dialogue interactives.

*Plus avancé*:

- Ajoutez un bouton clear qui efface toute la page de dessin.
- Ajoutez la possibilité de pouvoir déplacer vos dessins.
- Ajoutez une ToolBox
    - Non noté, projet bonus si rendu.