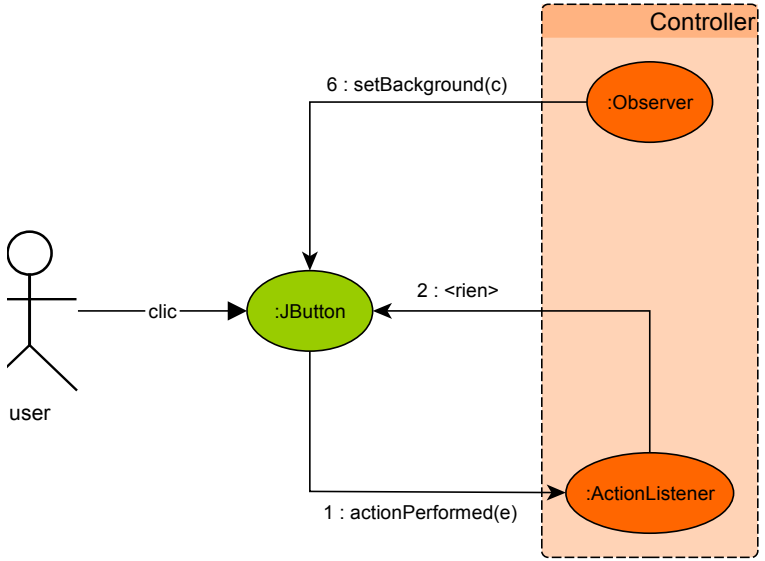
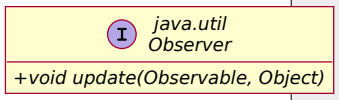
Applications graphiques Java

- javax.swing.JComponent
  - javax.swing.AbstractButton
    - javax.swing.JButton
    - javax.swing.JMenuItem
      - ■ ■ ■
      - javax.swing.JMenu
      - ■ ■ ■
    - javax.swing.JToggleButton
      - javax.swing.JCheckBox
      - javax.swing.JRadioButton
  - ■ ■ ■
  - javax.swing.JComboBox
  - javax.swing.JFileChooser
  - ■ ■ ■
  - javax.swing.JLabel
    - ■ ■ ■
  - ■ ■ ■
  - javax.swing.JMenuBar
  - javax.swing.JOptionPane
  - javax.swing.JPanel
    - ■ ■ ■
  - ■ ■ ■
  - javax.swing.JRootPane
  - javax.swing.JScrollBar
    - javax.swing.JScrollPane.ScrollBar
  - javax.swing.JScrollPane
  - javax.swing.JSeparator
    - ■ ■ ■
  - javax.swing.JSlider
  - javax.swing.JSpinner
  - ■ ■ ■
  - javax.swing.text.JTextComponent
    - javax.swing.JEditorPane
      - javax.swing.JTextPane
    - javax.swing.JTextArea
    - javax.swing.JTextField
      - javax.swing.JFormattedTextField
      - javax.swing.JPasswordField
  - javax.swing.JToolBar
  - ■ ■ ■
  - javax.swing.JViewport

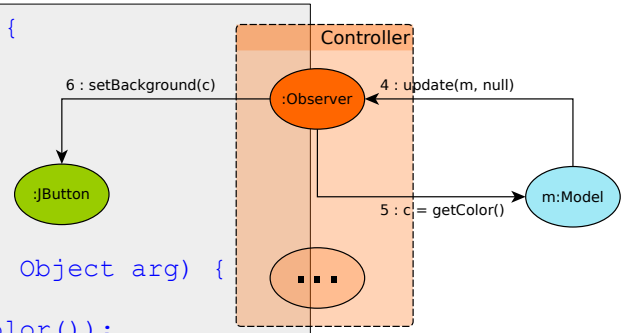
Codage de la vue :
l'API Java est généralement suffisante



Codage du contrôleur : l'observateur du modèle...

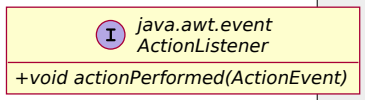



```
public class MO implements Observer {
    // ATTRIBUTS D'INSTANCE
    private JButton button;
    // CONSTRUCTEURS
    public MO(JButton b) {
        button = b;
    }
    // COMMANDES
    public void update(Observable o, Object arg) {
        Model m = (Model) o;
        button.setBackground(m.getColor());
        button.setText("go to " + m.nextColorName()) ;
    }
}
```

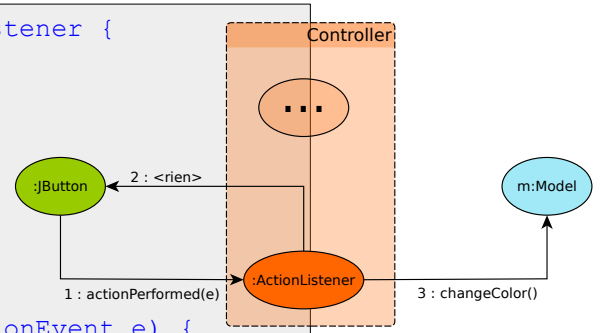


action → demande d'actrualisation de la vue
réaction → changement de couleur du bouton

... et l'écouteur de la vue




```
public class AL implements ActionListener {
    // ATTRIBUTS D'INSTANCE
    private Model model;
    /* le lien vers la vue
       n'est pas nécessaire ici */
    // CONSTRUCTEURS
    public AL(Model m) {
        model = m;
    }
    // COMMANDES
    public void actionPerformed(ActionEvent e) {
        model.changeColor();
    }
}
```



action → clic sur le bouton
réaction → changement d'état du modèle

```java
public class ColoredAppli {
    // ATTRIBUTS D'INSTANCE
    private final JFrame frame;
    private final JButton coloredButton;
    private final Model model;
    // CONSTRUCTEURS
    public ColoredAppli() {
        // MODELE
        model = new Model();
        // VUE
        frame = new JFrame("Exemple");
        coloredButton = new JButton();
        placeComponents();
        // CONTROLEUR
        connectControllers();
    }
    // COMMANDES
    public void display() {
        frame.pack();
        frame.setLocationRelativeTo(null);
        refreshView();
        frame.setVisible(true);
    }
    // OUTILS
    private void placeComponents() {
        // Panneau central contenant coloredButton
        JPanel p = new JPanel();
        { //--
            p.add(coloredButton);
        } //--
        frame.add(p, BorderLayout.CENTER);
    }
    private void connectControllers() {
        // écoute les demandes de fermeture de l'application
        frame.setDefaultCloseOperation(EXIT_ON_CLOSE);
        // écoute les demandes de changement de couleur
        coloredButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                model.changeColor();
            }
        });
        // observe les changements de couleur du modèle
        model.addObserver(new Observer() {
            public void update(Observable o, Object arg) {
                refreshView();
            }
        });
    }
    private void refreshView() {
        coloredButton.setBackground(model.getColor());
        coloredButton.setText("go to " + model.nextColorName());
    }
    // POINT D'ENTREE
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new ColoredAppli().display();
            }
        });
    }
}
```

1. déclarer les composants graphiques majeurs et le modèle

2. créer le modèle, la vue et le contrôleur... ... à l'aide de méthodes outils si nécessaire

```java
private final JTextField pwd;
public MonAppli() {
    ...
    pwd = createPwd(Color.RED);
    ...
}
private JTextField createPwd(Color c) {
    JTextField tf = new JTextField();
    tf.setEditable(true);
    tf.setBackground(c);
    return tf;
}
```

3. construire l'arborescence des composants graphiques de la vue

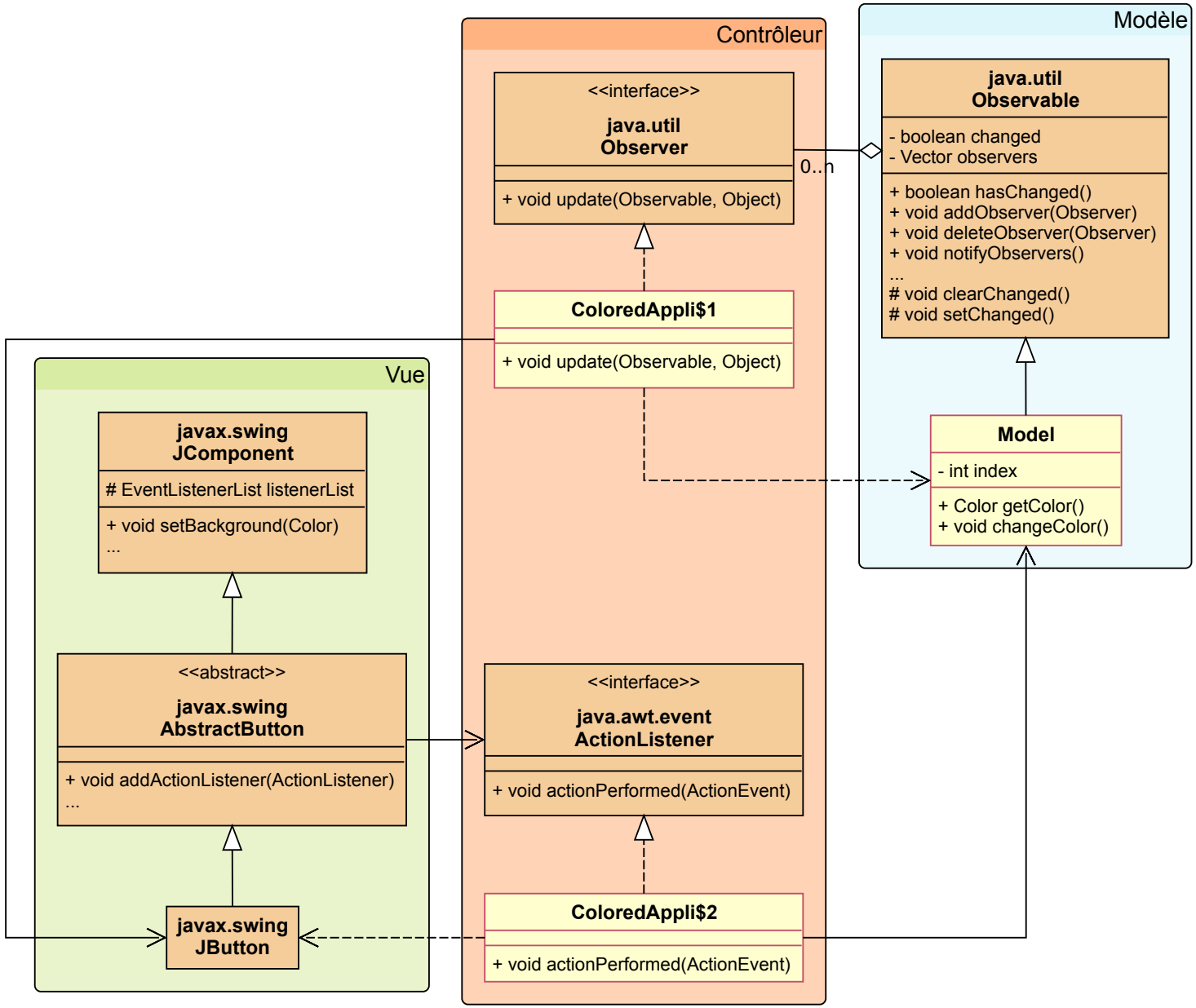
4. construire les contrôleurs et les connecter avec la vue et le modèle

5. initialiser l'application

6. factoriser judicieusement le code des contrôleurs

7. lancer l'application à l'aide d'une "formule magique"

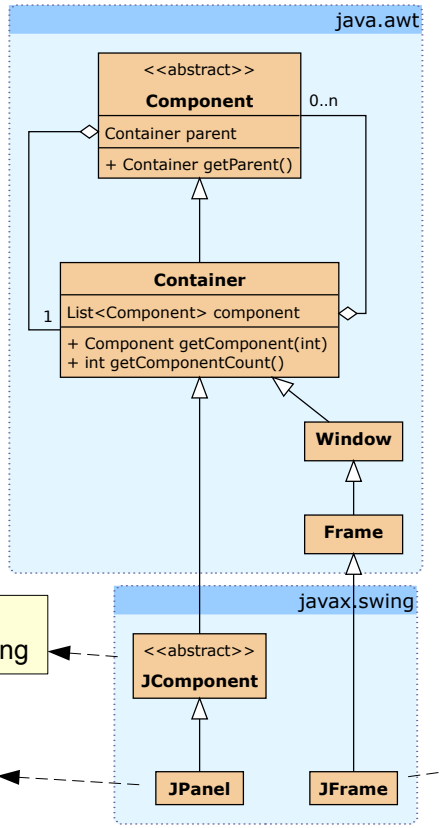
Applications graphiques Java



Applications graphiques Java

nb méthodes	public	protected
Object	9 / 0 / 0	2 / 0 / 0
Component	166 / 8 / 1	19 / 2 / 0
Container	33 / 148 / 27	3 / 19 / 2
JComponent	68 / 167 / 41	11 / 19 / 5
JPanel	2 / 273 / 3	0 / 34 / 1

légende : définies / héritées / redéfinies



classe de base pour les composants graphiques Swing

classe de base pour les conteneurs graphiques Swing

classe de base pour les fenêtres graphiques Swing

Component

```
+ Color getBackground()
+ void setBackground(Color)

+ Color getForeground()
+ void setForeground(Color)

+ Graphics getGraphics()
+ void paint(Graphics)
+ void repaint()

+ boolean isEnabled()
+ void setEnabled(boolean)

+ boolean isVisible()
+ void setVisible(boolean)

+ int getHeight()
+ int getWidth()

+ Dimension getMaximumSize()
+ void setMaximumSize(Dimension)

+ Dimension getMinimumSize()
+ void setMinimumSize(Dimension)

+ Dimension getPreferredSize()
+ void setPreferredSize(Dimension)

+ Dimension getSize()
+ void setSize(Dimension)
```

Container

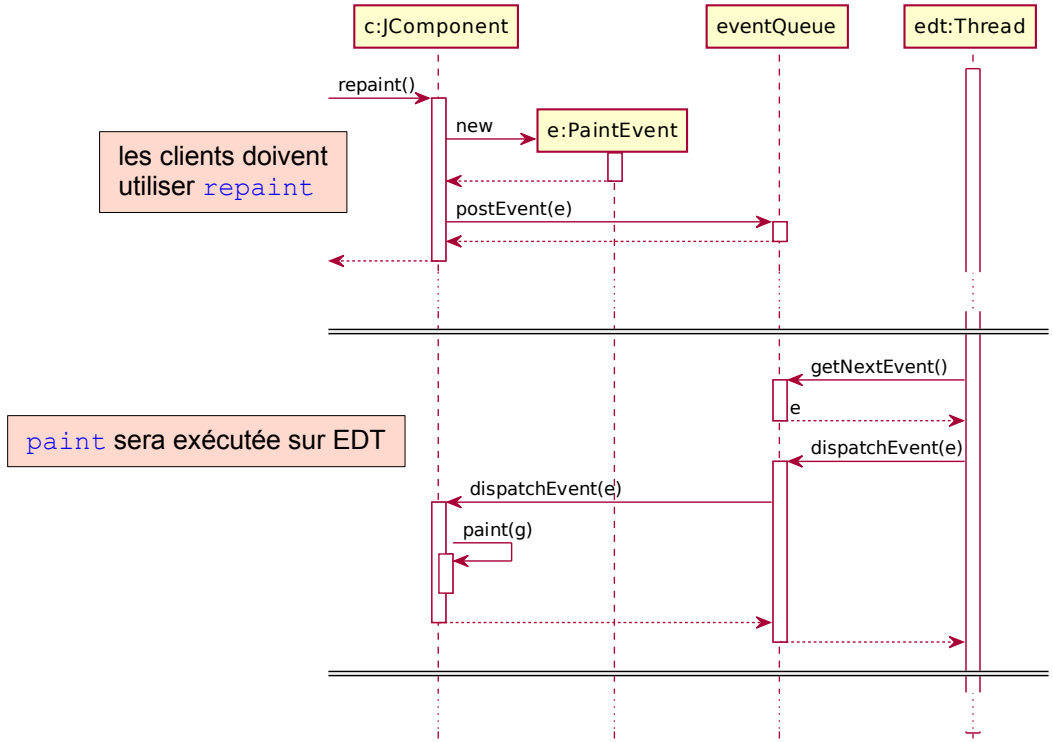
```
+ void add(Component)
+ void add(Component, int)
+ void add(Component, Object)
+ void add(Component, Object, int)
```

JComponent

```
+ boolean isDisplayable()

# void paintBorder(Graphics)
# void paintChildren(Graphics)
# void paintComponent(Graphics)
```

Applications graphiques Java



feuille de dessin allouée au composant

le composant doit définir son code de dessin ici

```
public void paint(Graphics g) {
    ...
    paintComponent(g);
    paintBorder(g);
    paintChildren(g);
    ...
}
```

Applications graphiques Java


```
class DemoApp {
    ...
    private void placeComponents() {
        frame.add(new DemoComp());
    }
    ...
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                new DemoApp().display();
            }
        });
    }
}
```

```
class DemoComp extends JComponent {
    public DemoComp() {
        setPreferredSize(new Dimension(110, 150));
    }
    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.RED);
        g.fillRect(0, 0, getWidth(), getHeight());
        g.setColor(Color.BLACK);
        g.drawString("Py Le Maléfique", 10, 50);
    }
}
```

appel à `setPreferredSize` placé dans le constructeur du composant

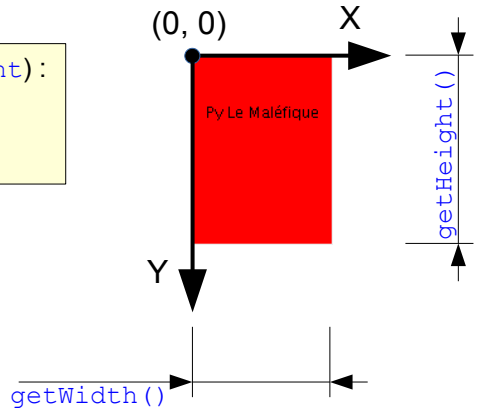
cas simple

il faut toujours définir la taille du composant par appel à `setPreferredSize`

cas complexe

appel à `setPreferredSize` placé dans la méthode `update` d'un `Observer`

Tailles initiales (`JComponent`) :
- minimum (0, 0)
- **préférée (0, 0)**
- maximum (∞, ∞)




```
class DemoComp2 extends JComponent {
    private Model model;
    public DemoComp2() {
        model = new Model();
        model.addObserver(new Observer() {
            @Override
            public void update(Observable obs, Object arg) {
                Dimension dim = model.getSize();
                DemoComp2.this.setPreferredSize(dim);
            }
        });
    }
    ...
}
```