



Interfaces Graphiques

Jean-Baptiste.Yunes@u-paris.fr

Université Paris Cité

©2025



- GUI Graphical User Interface, Interface Utilisateur Graphique, un type d'interface :
- plus riche que le simple clavier et l'affichage en mode texte
- un paradigme
- une représentation picturale
- un mode d'interaction gestuel

- c'est un paradigme très répandu aujourd'hui
- mais qui n'a pas que des avantages...
- Neal Stephenson
 - In the beginning was the command line



- on considère habituellement que le père des GUI est Douglas Engelbart (1962)
- **AUGMENTING HUMAN INTELLECT: A Conceptual Framework**
- mais que dire du projet SAGE 1950-1983 ?



- on co
est D
- AU
Cor
- mais c



es GUI

A

- qu'est ce que ce type d'application a de particulier ?
 - son architecture
 - c'est l'utilisateur qui décide...
 - ...ou plutôt ce sont les évènements qui décident
- event-driven programming, event-driven design

- Event-driven programming ?
- une boucle principale :
 - (event detection) retirer un évènement
 - (event handling) distribuer l'évènement à la partie logicielle concernée

- La clé de tout lorsqu'on conçoit un programme utilisant une interface graphique c'est de suivre le conseil suivant :
- une interface est bien conçue lorsque le programme se comporte comme l'utilisateur pense qu'il doit : user centric
- ceci nécessite une conception soignée

- Quelle architecture ?
 - un système de fenêtrage (windowing system)
 - un gestionnaire de fenêtres (window manager)
 - des outils et bibliothèques d'objets (tools & API)

- système de fenêtrage
 - X Window System (monde Unix) *aka* X11R7.7
 - Wayland
 - Quartz (Apple)
 - Windows (Microsoft)
- fournit aux applications un espace (graphique) d'accueil propre

- gestionnaire de fenêtre (window manager)
 - kde
 - gnome
 - twm
 - motif
 - etc.
 - aqua
 - dwm (Desktop Window Manager)
- gère l'ensemble des fenêtres et uniformise leur manipulation

- outils et bibliothèque d'objets (toolkits)
 - Xt, Xaw, Motif
 - GTK+
 - FLTK
 - Qt
 - wxWidgets
 - Cocoa
- uniformise la représentation visuelle des objets standards

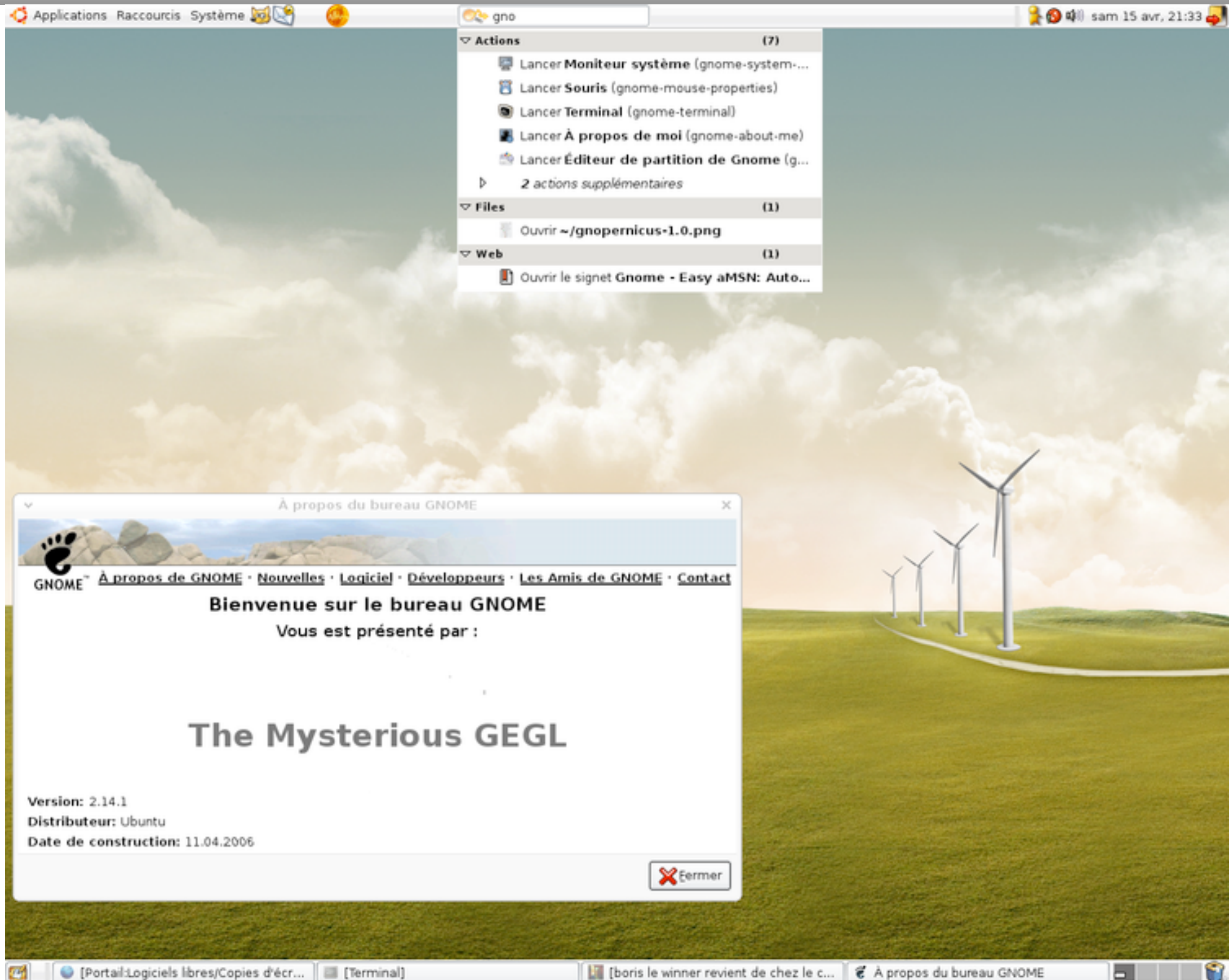
Aqua/Quartz (MacOSX)

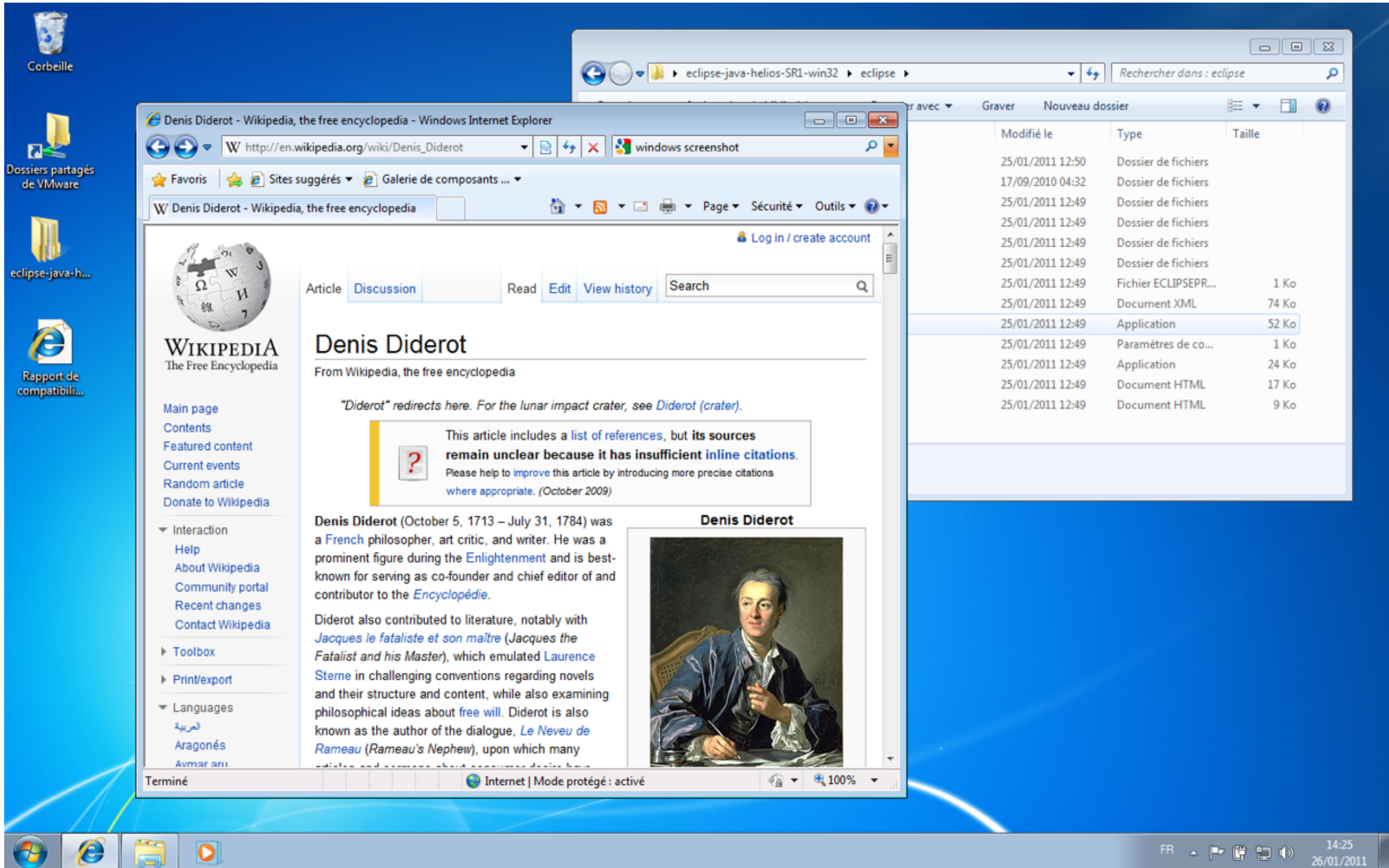


Photo d'écran, source privée



gnome/X11 (Linux)





- idée générale :
 - programmer sans de préoccuper des aspects graphiques
- toutefois il existe des règles à suivre pour créer une interface graphique dans un environnement donné
 - les guidelines

- KDE User Interface Guidelines
- Windows User Experience Interaction Guidelines
- Guide de l'Interface Utilisateur de Gnome
- Apple Human Interface Guidelines
- etc.

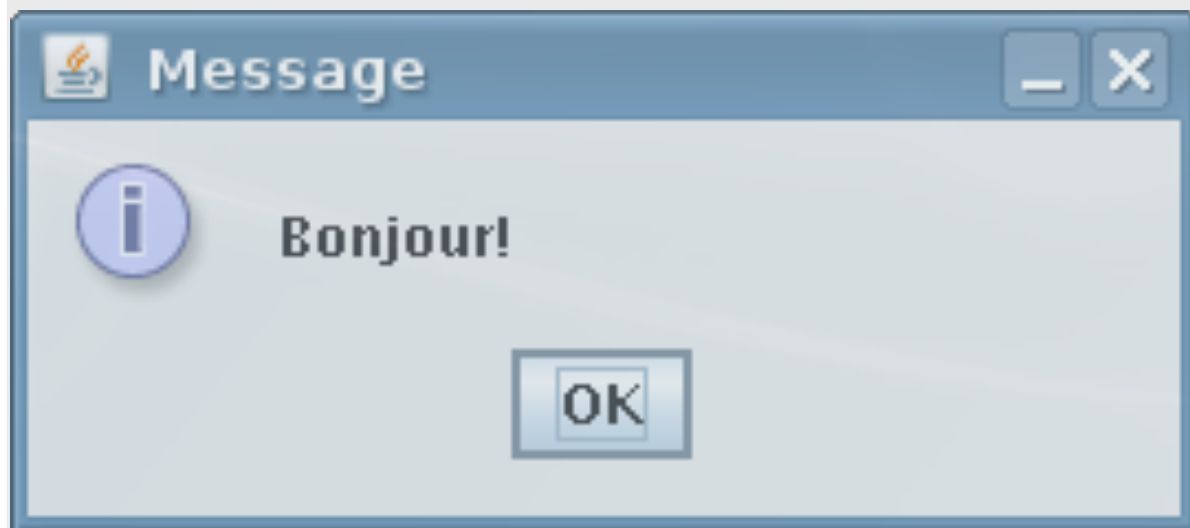
- suivre les règles permet d'obtenir une application qui se comporte comme attendu
- les API permettant en grande partie de garantir la conformité vis-à-vis de ces règles
- il n'y a donc qu'à se focaliser sur les aspects abstraits ou logiques

- un programme utilisant une interface graphique compliquée n'est pas nécessairement complexe
- les API offrent en général des objets tout faits permettant de rendre la plupart des services généraux attendus...
- afficher un message dans une fenêtre, par exemple...

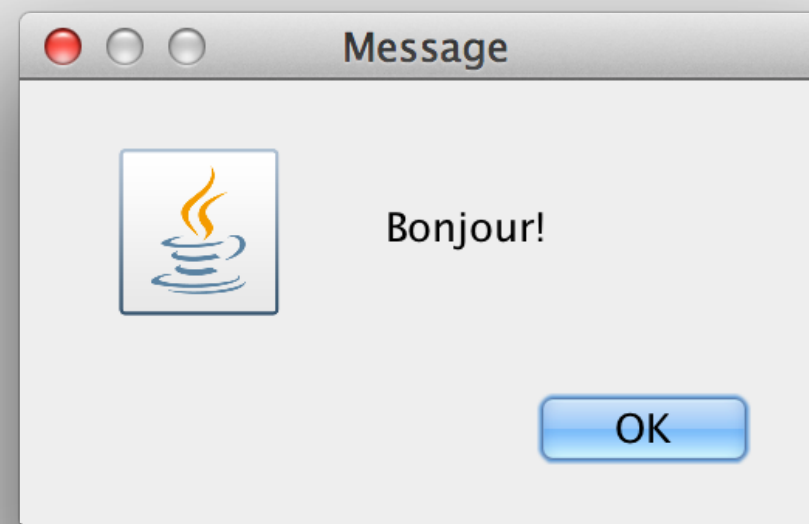
Bonjour (minimal)

```
import javax.swing.JOptionPane; Bonjour.java

public class Bonjour {
    public static void main(String[] args) {
        JOptionPane.showMessageDialog( null, "Bonjour!" );
    }
}
```



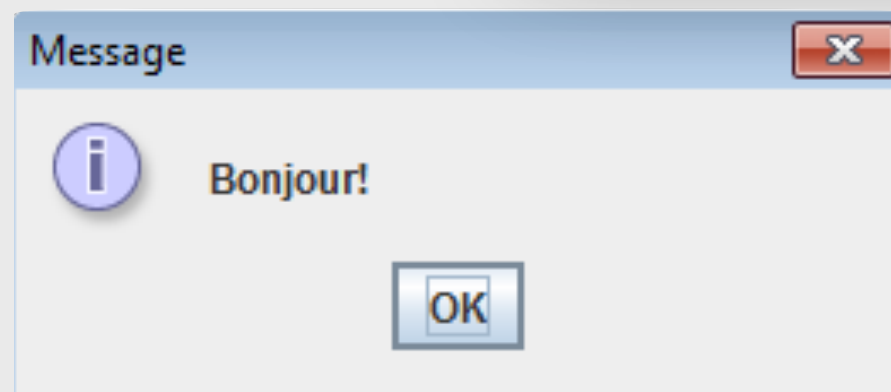
Linux (kde)



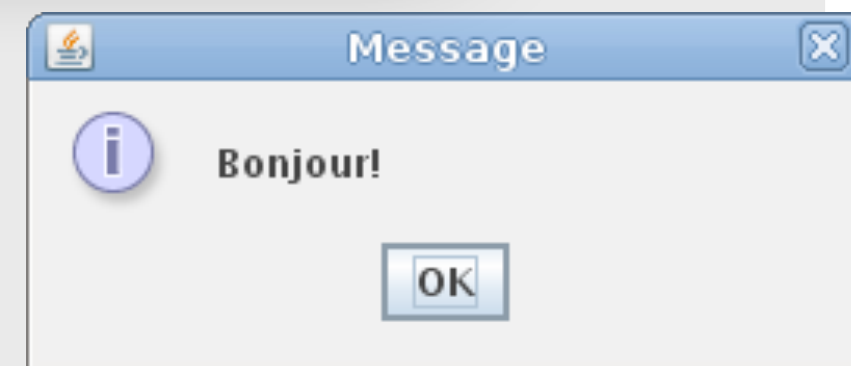
MacOSX



Linux (twm)



Windows 7



Linux (gnome)

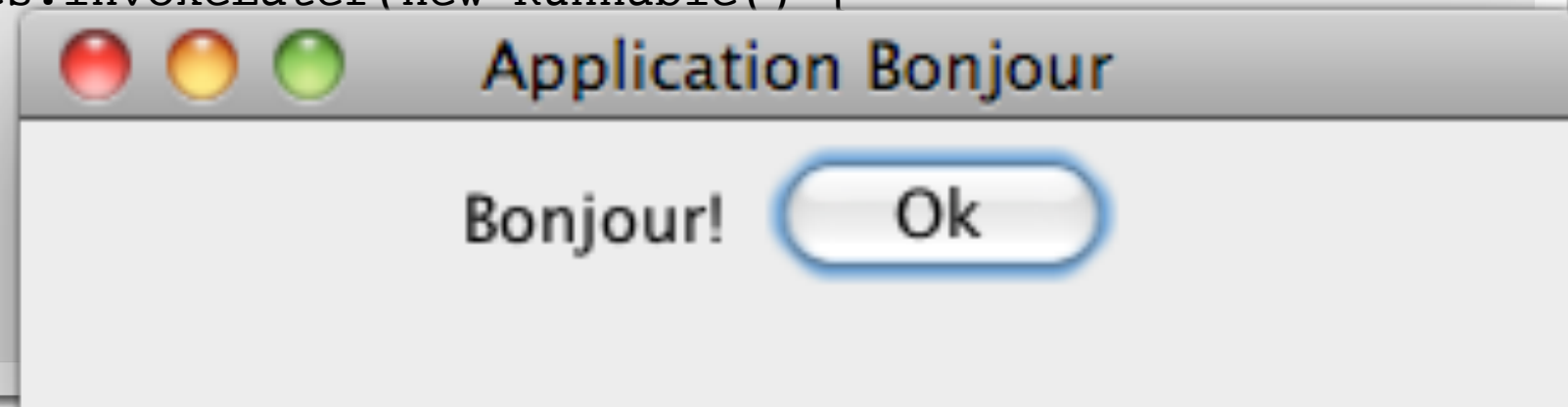
Bonjour (à la main)

La même chose à *la main*

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

public class Bonjour {
    private static void creeInterface() {
        JFrame frame = new JFrame("Application Bonjour");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Bonjour!");
        JButton button = new JButton("Ok");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().add(label);
        frame.getContentPane().add(button);
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creeInterface();
            }
        });
    }
}
```



- Que sont/ont tous ces objets ?
- Comment interfèrent-ils les uns avec les autres ?
- C'est le sujet central de ce cours
 - Comprendre la structure d'une API d'une interface graphique
 - nous étudierons celle de Java : Swing
 - mais les autres sont assez semblables au moins dans leur principe

- En préliminaire on peut dire que les interfaces graphiques :
- constituent un domaine pour lequel le paradigme objet est assez efficace
- font une utilisation intensive de divers *design pattern*
- si vous avez déjà tout oublié : révisez...

Décomposition d'une interface

The screenshot shows the Eclipse IDE with the following components:

- Menu Bar:** Eclipse, File, Edit, Source, Refactor, Navigate, Search, Project, Run, Sample, Window, Help.
- Package Explorer:** Shows a project structure with folders: ACRIZO10, CAVE, Graf, GRIDS, and IcePileModel.
- Editor:** Displays the source code for `CasierFrigo.java`. The code includes:


```

ResourceBundle res = ResourceBundle.getBundle("Messages", Start.loc);

NumberFormat nf = NumberFormat.getInstance(Locale.FRANCE);
//DateFormat df = DateFormat.getDateInstance(DateFormat.FULL, Locale.FRANCE);
SimpleDateFormat df = new SimpleDateFormat("EEEE dd MMMM yyyy");
for(Vin v : list){
    System.out.println(res.getString("Knom") + ": " + v.getNom() );
    System.out.println(res.getString("Kannee") + ": " + v.getAnnee());
    System.out.println(df.format(v.getDateAquisition()));
    for (Entry<Cepage, Float> ent : v.getCepages().entrySet()) {
        System.out.println("\t" + ent.getKey().getNom() + ": " +
            nf.format(ent.getValue()));
    }
    System.out.println();
}
}

/*class CompVin implements Comparator<Vin>{

    public int compare(Vin o1, Vin o2) {
        if(o1.getNom().equals(o2.getNom()))
            return 0;
        else
            return 1;
    }
}*/

```
- Outline:** Shows the class hierarchy and methods for `CasierFrigo`, including `list : List<Vin>`, `cases : int`, `CasierFrigo(int)`, `AjouterVin(Vin) :`, `RetirerVin(Vin) : v`, `new Comparat`, `CaseDispo() : int`, `refroidir(Vin, int)`, and `AfficherCasier() :`.
- Problems/Console/Properties:** Shows "No consoles to display at this time."

- GUI Java
- Historiquement :
- AWT Abstract Window Toolkit
 - son utilisation directe est considérée comme obsolète
 - sa disponibilité est conservée car :
- Swing, basé sur AWT...
 - il existe aussi SWT (Eclipse Foundation)...

- AWT
 - les composants sont **lourds** (*heavyweight*), i.e. ils ont tous un pair natif attaché...
 - la conception d'AWT est *thread-safe*, trop lourd...
- Swing
 - sauf les composants racine, ils sont **légers** (*lightweight*)
 - pas *thread-safe*

- JFC Java Foundation Classes
 - composants Swing
 - look-and-feel
 - accessibilité
 - API 2D
 - internationalisation

- 12 packages AWT
 - dans le module `java.desktop`

`java.awt`

`java.awt.color`

`java.awt.datatransfer`

`java.awt.dnd`

`java.awt.event`

`java.awt.font`

`java.awt.geom`

`java.awt.im`

`java.awt.im.spi`

`java.awt.image`

`java.awt.image.renderable`

`java.awt.print`

- 18 packages Swing
- dans le module `java.desktop`

`javax.swing`

`javax.swing.border`

`javax.swing.colorchooser`

`javax.swing.event`

`javax.swing.filechooser`

`javax.swing.plaf`

`javax.swing.plaf.basic`

`javax.swing.plaf.metal`

`javax.swing.plaf.multi`

`javax.swing.plaf.nimbus`

`javax.swing.plaf.synth`

`javax.swing.table`

`javax.swing.text`

`javax.swing.text.html`

`javax.swing.text.html.parser`

`javax.swing.text.rtf`

`javax.swing.tree`

`javax.swing.undo`

JFrame

```

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

public class Bonjour {
    private static void creeInterface() {
        JFrame frame = new JFrame("Application Bonjour");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Bonjour!");
        JButton button = new JButton("Ok");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().add(label);
        frame.getContentPane().add(button);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creeInterface();
            }
        });
    }
}

```

- JFrame ?
- un des quatres conteneurs racine (top-level containers) JFrame, JDialog, JWindow, ~~JApplet~~
- il faut au moins un conteneur pour afficher quelque chose...

JLabel

```

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

public class Bonjour {
    private static void creeInterface() {
        JFrame frame = new JFrame("Application Bonjour");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Bonjour!");
        JButton button = new JButton("Ok");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().add(label);
        frame.getContentPane().add(button);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creeInterface();
            }
        });
    }
}

```

- JLabel ?
- un composant permettant d'afficher un texte court, une image ou les deux
- il est passif

JButton

```

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

public class Bonjour {
    private static void creeInterface() {
        JFrame frame = new JFrame("Application Bonjour");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Bonjour!");
        JButton button = new JButton("Ok");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().add(label);
        frame.getContentPane().add(button);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creeInterface();
            }
        });
    }
}

```

- JButton ?
- un composant permettant d'obtenir une interaction basique avec l'application

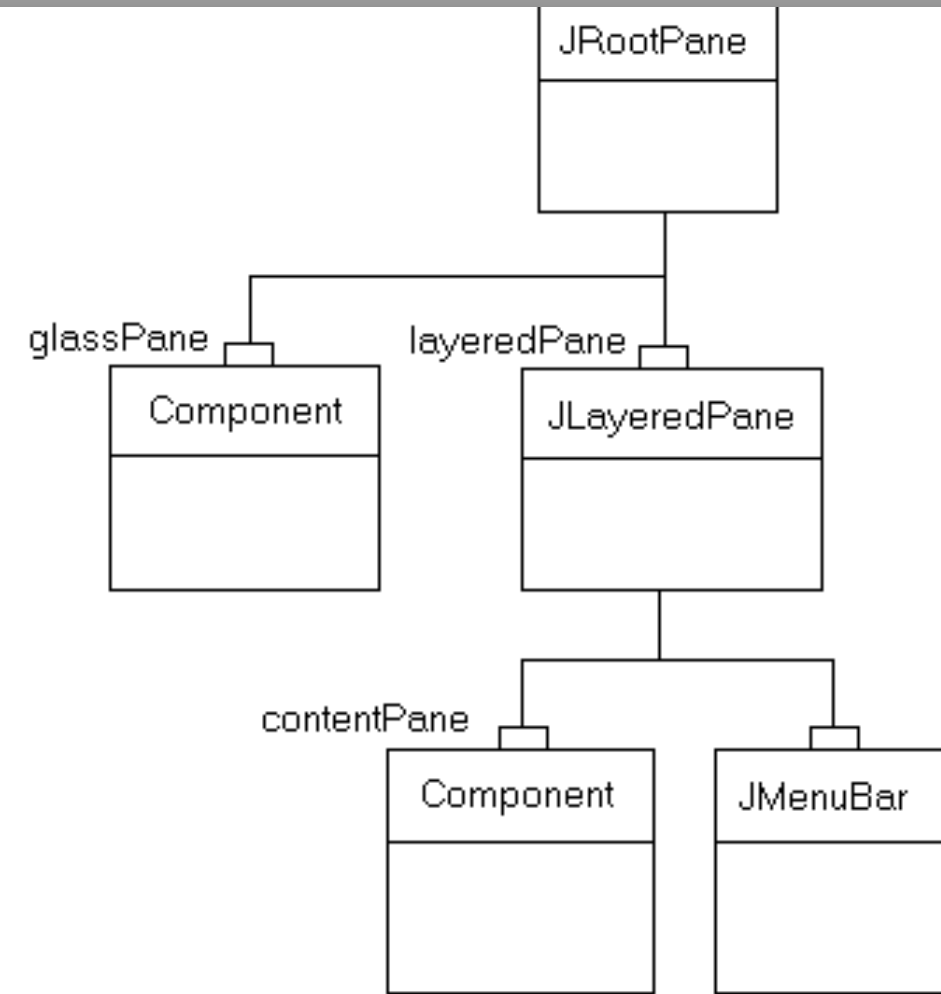
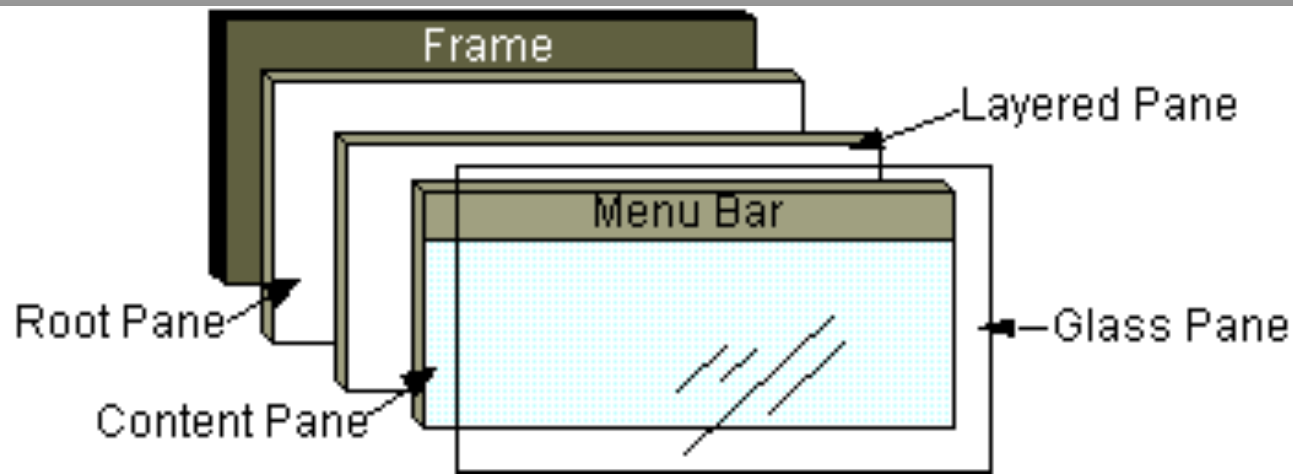
Listener

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

public class Bonjour {
    private static void creeInterface() {
        JFrame frame = new JFrame("Application Bonjour");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel label = new JLabel("Bonjour!");
        JButton button = new JButton("Ok");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.exit(0);
            }
        });
        frame.getContentPane().setLayout(new FlowLayout());
        frame.getContentPane().add(label);
        frame.getContentPane().add(button);
        frame.pack();
        frame.setVisible(true);
    }
    public static void main(String[] args) {
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                creeInterface();
            }
        });
    }
}
```

- JButton ?

- `addActionListener` permet d'associer une **réaction** au clic, ici on termine l'application



- **JFrame** contient :
- un **JRootPane** qui lui-même est composé :
 - d'un **glassPane** (**Component**) qui recouvre un **JLayeredPane** lequel contient :
 - un **contentPane** (**Container**)
 - éventuellement une **JMenuBar**

- le `ContentPane` par défaut est
 - descendant de `JComponent`
 - lequel utilise un `BorderLayout`
 - note : les composants Swing sont des Containers AWT...

- `pack()` :
- calcule les tailles adéquates permettant une représentation raisonnable à l'écran de la hiérarchie construite