

# Parts of the GUI

## Swing #2

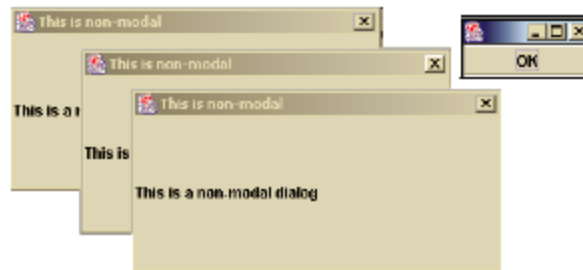
Athanasia Katsouraki

19/10/2017

# JDialog

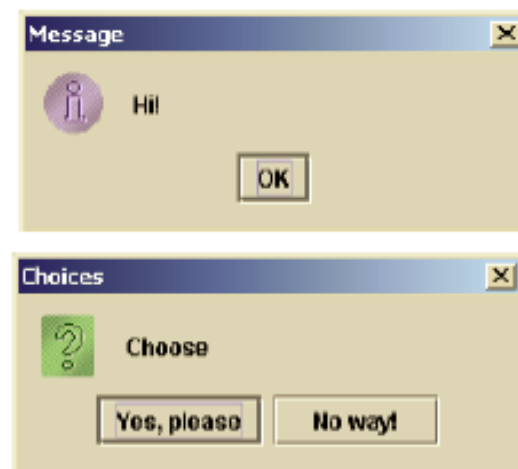
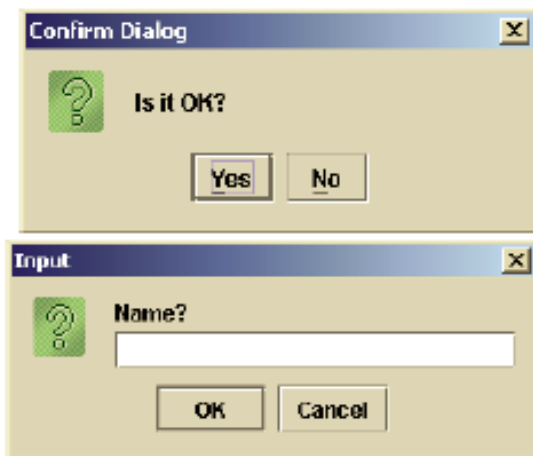
```
JDialog dialog = new JDialog(myFrame, "Dialog Frame");  
JLabel label = new JLabel("This is a message");  
dialog.getContentPane().add(label);  
dialog.setVisible(true);
```

```
JDialog dialog1 = new JDialog(myFrame, "This is modal", true);  
JDialog dialog2 = new JDialog(myFrame, "This is non modal", false);
```



# Dialog Frames

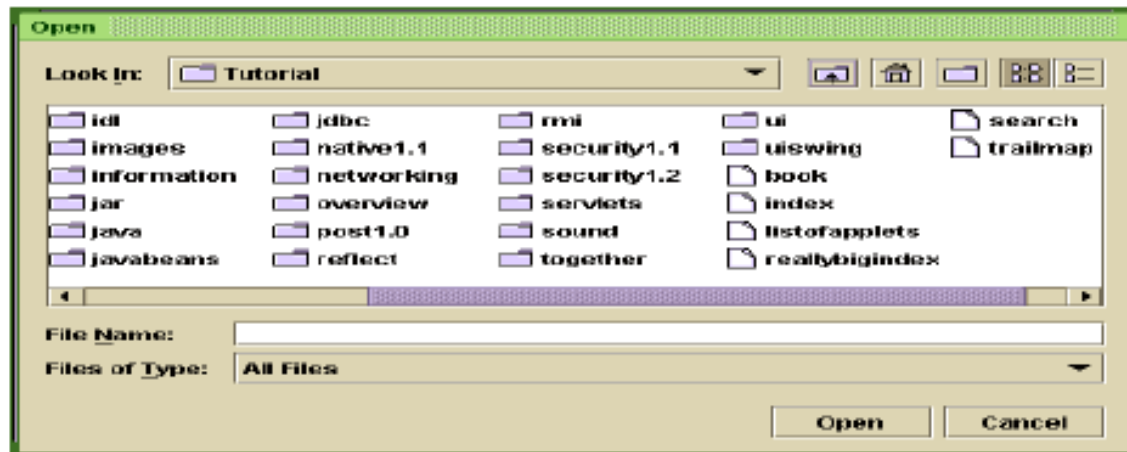
- `int n = JOptionPane.showConfirmDialog(myFrame, "Is it OK?", "Confirm Dialog", JOptionPane.YES_NO_OPTION);`
- `String s = (String)JOptionPane.showInputDialog(myFrame, "Name?");`
- `JOptionPane.showMessageDialog(myFrame, "Hi!");`
- `Object[] options = {"Yes, please", "No way!"};`  
`int n = JOptionPane.showOptionDialog(myFrame, "Choose", "Choices",  
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE,  
null, options, options[0]);`



# JFileChooser

```
JFileChooser fc = new JFileChooser();
```

```
int returnVal = fc.showOpenDialog(aComponent);
```



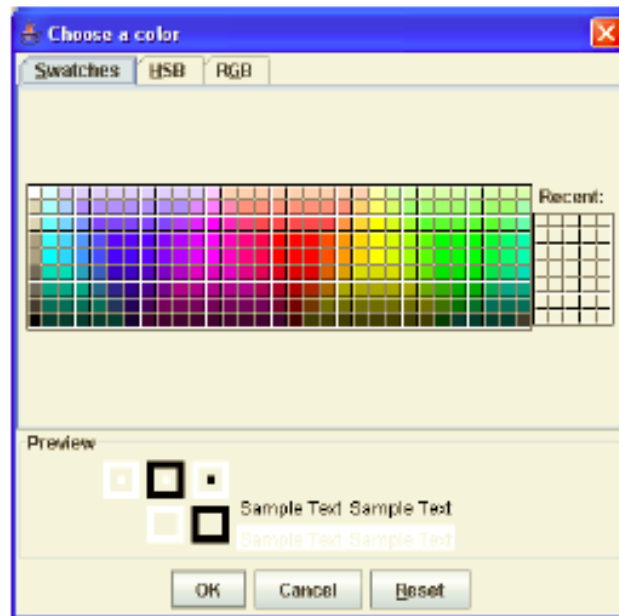
# JFileChooser

```
if (returnVal == JFileChooser.APPROVE_OPTION) {  
    // Open  
    File file = fc.getSelectedFile(); // chosen file  
    log.append("Opening: " + file.getName() + "." + newline);  
}  
else {  
    log.append("Open command cancelled by user." + newline);  
}
```

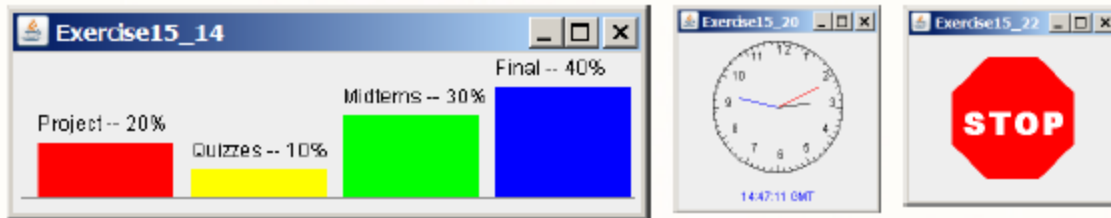
fc.showDialog(*aComponent*, "Save"); // open button ➡ Save

# JColorChooser

```
Color c = JColorChooser.showDialog(myFrame, "Choose a color", myFrame.getContentPane().getBackground( ));  
if (c != null) myFrame.getContentPane().setBackground(c);
```



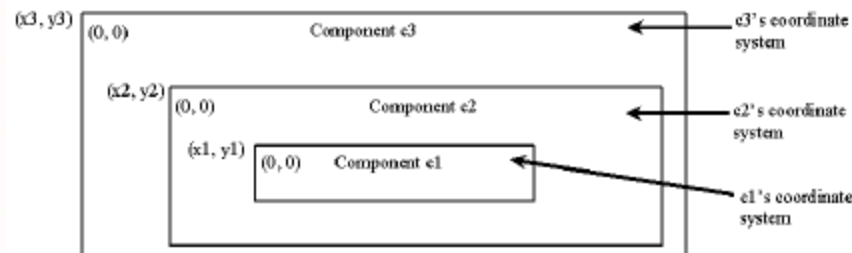
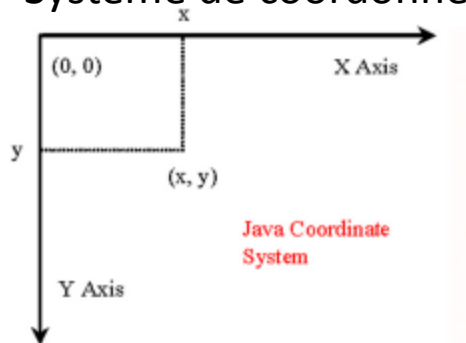
# Graphics (arbitraires)



Classe `java.awt.Graphics`

Fournit des méthodes pour tracer des lignes, des rectangles,

Système de coordonnées JAVA

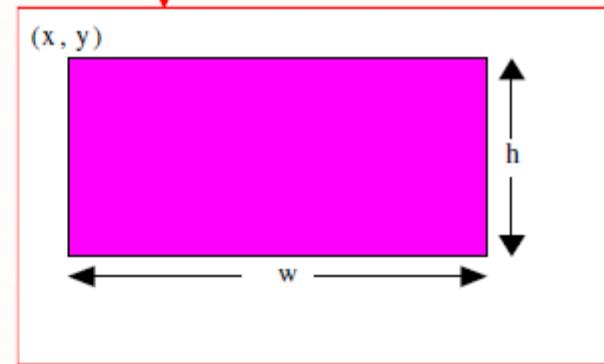
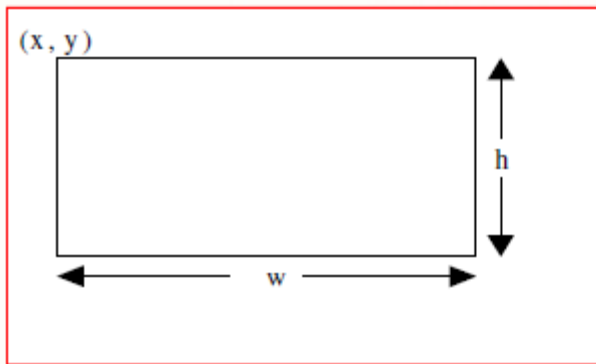


# Example

```
drawRect(int x, int y, int w, int h);
```

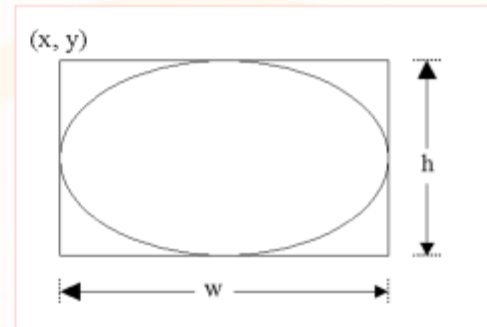


```
fillRect(int x, int y, int w, int h);
```



```
drawOval(int x, int y,  
          int w, int h);
```

```
fillOval(int x, int y,  
          int w, int h);
```





# Exemple

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

public class HandleEvent extends JFrame {
    public HandleEvent() {

        // Create and add button ok
        JButton jbtOK = new JButton("OK");
        add(jbtOK);

        // Register listeners
        OKListenerClass listener1 = new OKListenerClass();
        jbtOK.addActionListener(listener1);
    }

    public static void main(String[] args) {
        JFrame frame = new HandleEvent();
        frame.setTitle("Handle Event");
        frame.setSize(200, 150);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}

class OKListenerClass implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}
```

HandleEvent  
c'est un  
JFrame

Creation  
listener.  
O listener  
"ecoute" les  
even. differents

Comment on  
declare?

# Example

```
JButton jbtOK = new JButton("OK");
add(jbtOK);
// Register listeners
OKListenerClass listener1 = new OKActionListener();
jbtOK.addActionListener(listener1);

OKListenerClass listener2 = new OKMouseListener();
jbtOK.addMouseListener(listener2);
}

public static void main(String[] args) {
    JFrame frame = new HandleEvent();
    ...
}
}

class OKActionListener implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}

class OKMouseListener implements MouseListener {
    public void mouseClicked(MouseEvent arg0) {
        System.out.println("mouseClicked");
    }
    public void mouseEntered(MouseEvent arg0) {
        System.out.println("mouseEntered");
    }
}
```

Listener1  
ActionListener

Listener2  
MouseListener

# Event Handling

Question: Disons que nous voulons gérer ce qui se passe quand l'utilisateur appuie sur un bouton. Comment est-ce fait?



Exemple: lorsque j'appuie sur le bouton "OK" pour exécuter `System.out.println ("OK cliqué");`

- Réponse: Utilisation « event handling » (programmation événementielle)

Programmation procédurale vs événements:

- Programmation procédurale: l'exécution est une opération en série  
flux
- Programmation axée sur les événements: Exécuter lorsque cela se produit  
un événement, par exemple un bouton

# ActionListener....

La fonction:

**void addActionListener (ActionListener l)**

ajoute un Listener (ActionListener) au bouton jbtOK.

- java.awt.event.ActionListener: est une interface qui doit implémenter chaque classe Listener
- La classe Listener que nous créons fournit des instructions qui le définissent

Que se passe-t-il lorsqu'un événement se produit?

```
class OKListenerClass implements ActionListener {  
    public void actionPerformed(ActionEvent e) {  
        System.out.println("OK button clicked");  
    }  
}
```

- Un événement peut être considéré comme un signal du programme déclarant que quelque chose est arrivé
- Nous avons encore une autre classe de haut niveau dans le même fichier!

# Events – Objects - Actions

User Action	Source Object	Event Type generated
Click a button	JButton	ActionEvent
Click a check box	JCheckBox	ItemEvent, ActionEvent
Click a radio button	JRadioButton	ItemEvent, ActionEvent
Press return on a text field	JTextField	ActionEvent
Select a new item	JComboBox	ItemEvent, ActionEvent
Window opened, closed, etc.	Window	WindowEvent
Mouse pressed, released, etc.	Component	MouseEvent
Key released, pressed, etc.	Component	KeyEvent

# Events – Event Handlers

Event Class	Listener Interface	Listener Methods (Handlers)
ActionEvent	ActionListener	actionPerformed(ActionEvent)
ItemEvent	ItemListener	itemStateChanged(ItemEvent)
WindowEvent	WindowListener	windowClosing(WindowEvent)
		windowOpened(WindowEvent)
		windowIconified(WindowEvent)
		windowDeiconified(WindowEvent)
		windowClosed(WindowEvent)
		windowActivated(WindowEvent)
		windowDeactivated(WindowEvent)
ContainerEvent	ContainerListener	componentAdded(ContainerEvent)
		componentRemoved(ContainerEvent)
MouseEvent	MouseListener	mousePressed(MouseEvent)
		mouseReleased(MouseEvent)
		mouseClicked(MouseEvent)
		mouseExited(MouseEvent)
		mouseEntered(MouseEvent)
KeyEvent	KeyListener	keyPressed(KeyEvent)
		keyReleased(KeyEvent)
		keyTyped(KeyEvent)

# Gestion des événements

## **L'élément GUI (par exemple un JButton)**

Produit des événements dans des circonstances spécifiques selon les activités de l'utilisateur

L'événement (par exemple un événement MouseEvent)

Un objet contenant des informations sur l'événement

## **Listeners (par exemple un MouseListener)**

Comprendre un événement

Ils ont des méthodes qui prennent les événements comme argument

# Créer des événements

Les événements sont des objets de commande:

`java.util.EventObject` (par ex. `AWTEvent`)

Transférer des informations sur le type d'événement,  
l'action qui l'a causé, etc.

Par exemple

un événement `MouseEvent` est généré par un élément lorsque l'utilisateur déplace la souris dans la zone de l'élément (par ex. `JButton`).

L'objet `MouseEvent` inclut:

des informations sur les coordonnées de la souris (x, y) pour les boutons, etc.

Un `ActionEvent` peut être produit à partir du même élément (par exemple, `JButton`) en vous informant que quelque chose est arrivé à l'élément



# Listen un evenement

Appuyer sur un JButton crée un événement (objet `ActionEvent`).

Pour le gérer, nous devons ajouter à JButton une façon de gérer les événements.

Les méthodes qui gèrent chaque événement : `Listeners`

```
interface publique ActionListener extends java.util.EventListener {  
    public void actionPerformed (ActionEvent e);  
}
```

Un élément peut avoir beaucoup des listeners. Chaque listener peut suivre de nombreux éléments.

# Types des Listeners

---

## Listeners

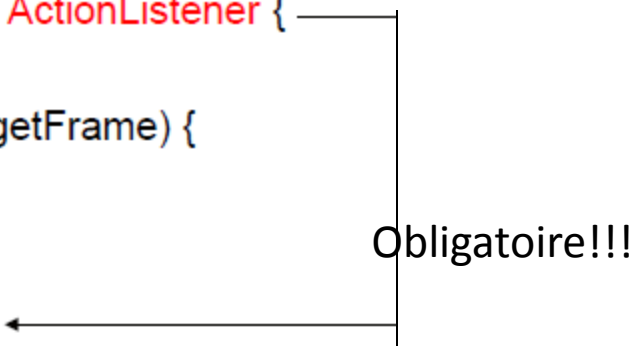
## Methodes

ActionListener	actionPerformed(ActionEvent)
MouseListener MouseAdapter	mouseClicked(MouseEvent) mouseEntered(MouseEvent) mouseExited(MouseEvent) mousePressed(MouseEvent) mouseReleased(MouseEvent)
MouseMotionListener	mouseDragged(MouseEvent) mouseMoved(MouseEvent)
KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)
ItemListener	itemStateChanged(ItemEvent)

ComponentListener ComponentAdapter	componentHidden(ComponentEvent) componentShown(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent)
ContainerListener ContainerAdapter	componentAdded(ContainerEvent) componentRemoved(ContainerEvent)
FocusListener FocusAdapter	focusGained(FocusEvent) focusLost(FocusEvent)
WindowListener WindowAdapter	windowActivated(WindowEvent) , windowClosed windowClosing, windowDeactivated, windowDeiconified, windowGainedFocus windowIconified, windowLostFocus windowOpened, windowStateChanged

# Exemple

```
public class CopyButtonActionListener implements ActionListener {  
    TestFrame targetFrame;  
    public CopyButtonActionListener(TestFrame targetFrame) {  
        this.targetFrame=targetFrame;  
    }  
    public void actionPerformed(ActionEvent e) {  
        String password = targetFrame.pf.getText();  
        targetFrame.ta.append(password);  
    }  
}
```



Obligatoire!!!

```
copyButton.addActionListener(new CopyButtonActionListener(this));
```

# MouseListeners...

Obligatoire!!!

```
public class myMouseListener implements MouseListener {  
    public void mousePressed(MouseEvent e) {  
        textArea.append ("Mouse pressed: # of clicks: "  
            + e.getClickCount(), " detected at position " + e.getX()+ ", "  
            + e.getY() + " of component"  
            + e.getComponent().getClass().getName() + ".\n");  
    }  
    public void mouseReleased(MouseEvent e) {...}  
    public void mouseClicked(MouseEvent e) { ...}  
    public void mouseEntered(MouseEvent e) {  
        textArea.append ("Mouse entered detected on "  
            + e.getComponent().getClass().getName() + ".\n");  
    }  
    public void mouseExited(MouseEvent e) { ...}  
}
```

# Even. → Composants...

ActionEvent	JButton, JCheckBoxMenuItem, JComboBox, JFileChooser, JList, RadioButtonMenuItem, JTextField, JToggleButton
ListSelectionEvent	JList, ListSelectionModel
ItemEvent	JCheckBoxMenuItem, ItemListener, JComboBox, JRadioButtonMenuItem, JToggleButton
MenuEvent	JMenu
MenuKeyEvent	JMenuItem
WindowEvent	JDialog, JFrame, JWindow

# Exemple

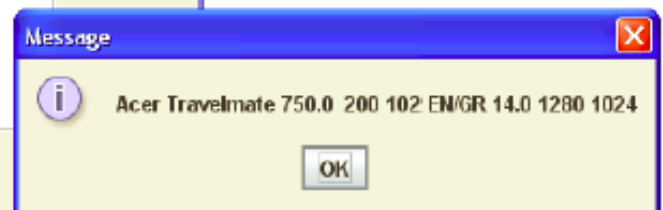
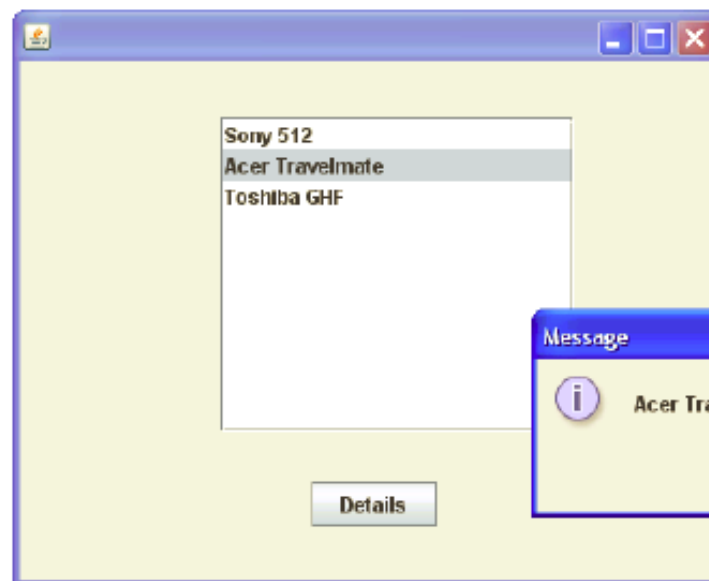
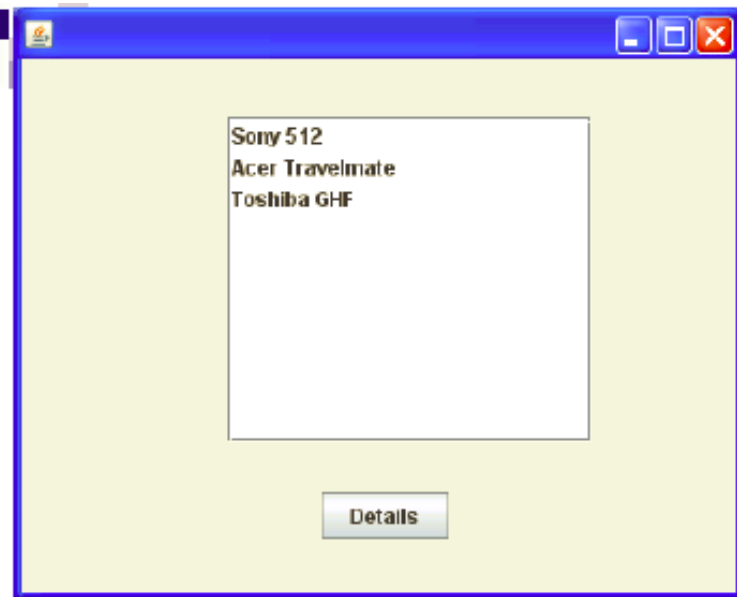
Application pour un magasin d'informatique :

Un Formulaire qui va:

lire tous les ordinateurs et il les affiche dans une liste

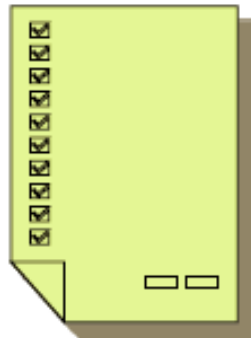
Permettre de sélectionner un seul ordinateur

En cliquant sur le bouton Détails, afficher les détails de l'ordinateur





MainFrame



createOrder

setVisible(true)

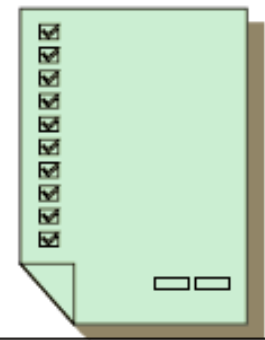
OrderFrame



addProduct

setVisible(true)

ProductFrame



Qu'est-ce  
que  
chaque form  
sait?

`ArrayList<Order> orderList`

`ArrayList<Product> productList`

Qu'est-ce que  
chaque form  
cree?

`Order o = new Order()`

`Product p = new Product()`

...

Dans le constructeur OrderFrame et ProductFrame, on passe comme attributes orderList et productList.

Sur le bouton "Enregistrer ...", nous créons l'objet correspondant et l'ajouter dans une liste que nous avons transmise sur le formulaire.

Nous informons indirectement (par référence) la liste originale

On ferme le formulaire en utilisant >> dispose()