المدرسة العليا للتكنولوجيا - الداخلة
ⵜⵉⵎⵙⵙⵓⵔⵉⵜ ⵜⴰⵣⵣⵓⵔⵜ ⵏ ⵜⵙⵙⵏⵉⵡⵉⵏ - ⴰⴷⴰⵅⵍⴰ
ÉCOLE SUPÉRIEURE DE TECHNOLOGIE - DAKHLA

Advanced object programming

2nd year GI

Prof. Naoual Nassiri

N.Nassiri@uiz.ac.ma

Academic year 2023-2024

JAVA graphics APIs

• We will see two APIs for programming graphical interfaces:

1. AWT (Abstract Windowing Toolkit);

2. Swing.

• The awt API was introduced in Java from version 1.0.

• Swing was introduced with version 1.1 of the jdk as part of of the Java Foundation Classes (JFC).

• The JFC includes in addition to Swing JAVA2D, accessibility, internationalization, ...

JAVA graphics APIs

- Third-party editors offer graphics APIs other than awt and swing.
- This is particularly the case of Eclipse with its swt framework and of Google with its GWT api.

Composition of AWT

- The AWT API is made up of 370 classes distributed over 12 packages. We

  We will focus on the most important of them, namely:

  o java.awt

  o and java.awt.event.

- The first contains the graphic classes forming the core of awt such as:

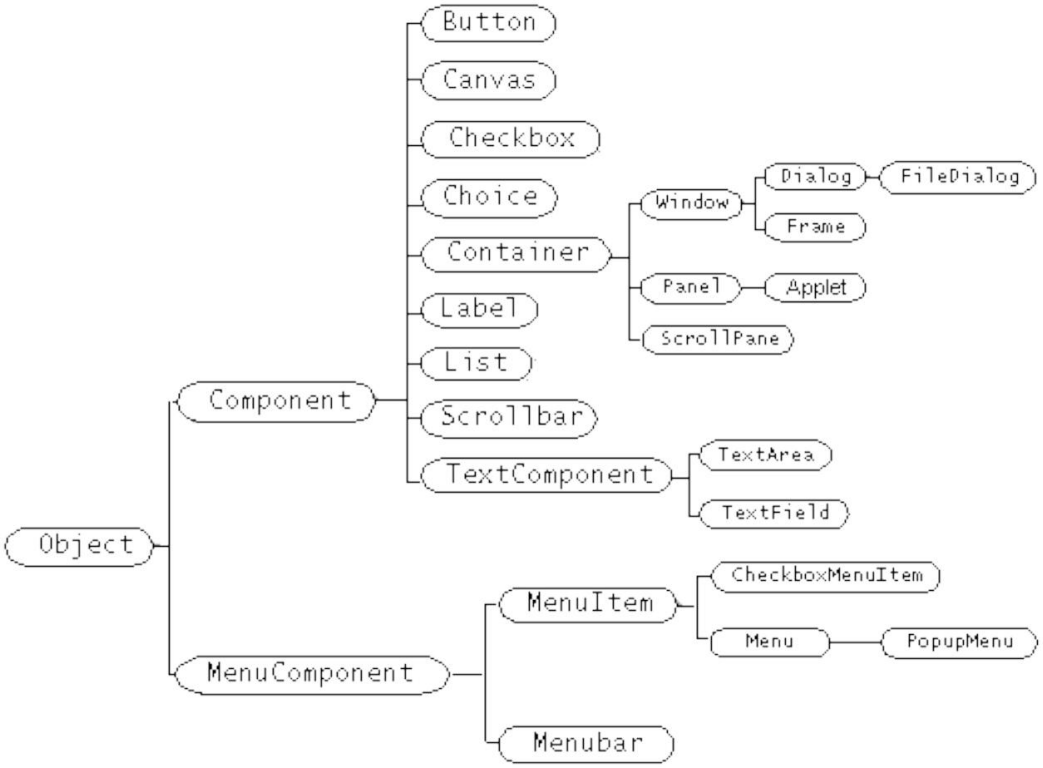
  components (buttons, labels, etc.), containers (Frame and Panel),

  positioning managers and special classes for managing

  fonts and colors.

Composition of AWT

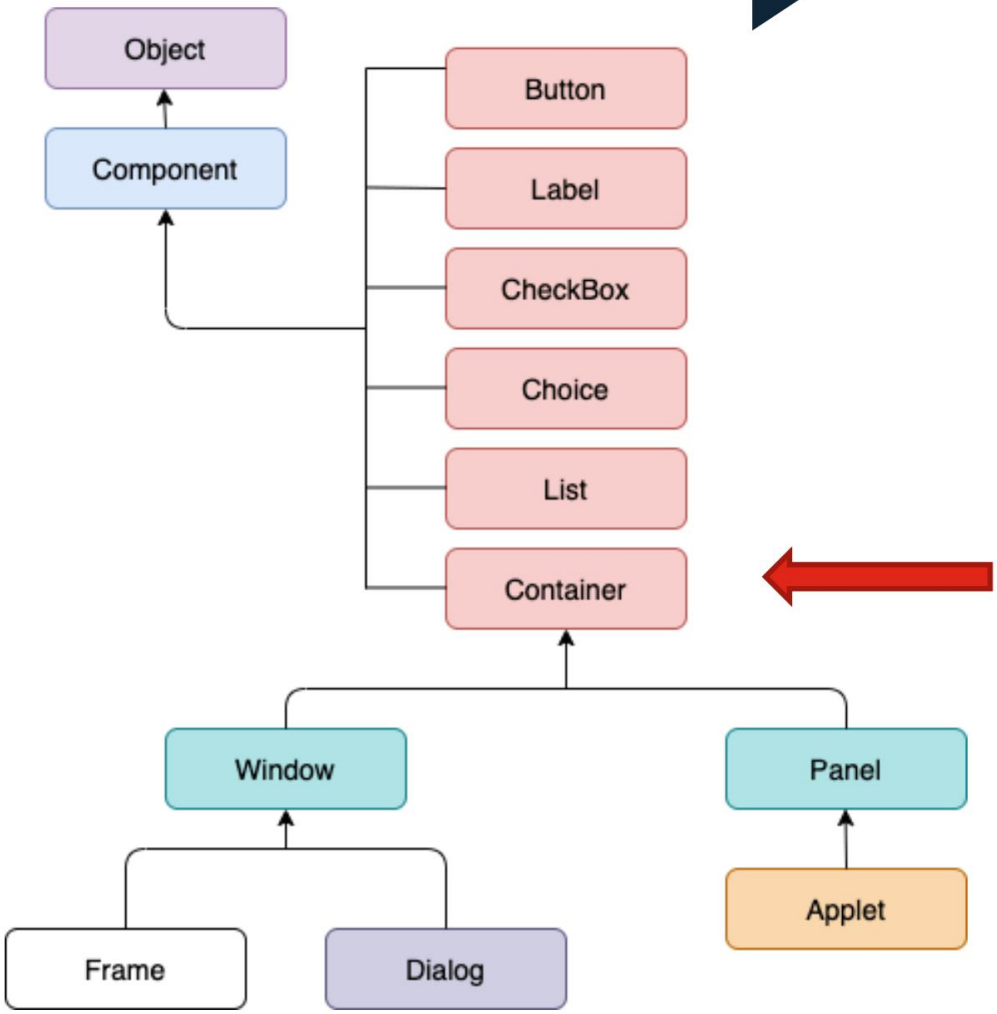
- The second package is used to manage events produced by the interaction between the graphical interface and the user: mouse click, keyboard input, ...



Container awt

- The previous Figure illustrates the hierarchy of awt containers. These containers can be classified into two categories according to the functionalities offered. We thus finds high-level containers and secondary containers.

High level container



Hierarchy of awt containers

High level container

- The Frame, Dialog and Applet classes represent the three main containers by awt. They are also called high-level containers because any program graphic must contain at least one.

- Instances of the Frame class represent an iconified window menu of a title bar of an optional menu bar and a display area of content. To create a program with a graphical interface we start typically by inheriting from the java.awt.Frame class:

High level container

Example:

```
1  import java.awt.Frame;   // Using Frame class in package java.awt
2
3  public class MonProgramGraphique extends Frame {
4      //attributs
5      ...
6      //constructeur
7      public MonProgramGraphique() { ... }
8
9      public static void main(String[] args) {
10         MonProgramGraphique p = new MonProgramGraphique();
11     }
12 }
```

High level container

- The Dialog class represents a "pop-up" window and has a bar title with an icon, a close button and a display area of content.

- Finally, the Applet class of the java.applet package is a container of high level which allows you to execute graphical interfaces within a Web browser.

Secondary container

- Secondary containers are placed within their upper counterparts level or principals. The Panel class represents a multi-use container rectangular without visual characteristics (color, etc.).
- The second ScrollPane container provides a container with a bar of horizontal and/or vertical scrolling for a single child component.

Components awt

- Modern graphical interfaces are built on the notion of components. These are reusable blocks with a graphic rendering and a business logic.

- Awt offers a collection of basic components such as buttons, lists, ... It is also possible to create your own component by combining the elementary components.

Components awt

- List of awt components:
  - oButton
  - o Canvas (drawing area)
  - o Checkbox (check box)
  - o CheckboxGroup
  - o Label
  - o Choice
  - oList
  - o Scrollbar
  - o TextField (1 line input box)
  - o TextArea (multi-line input area)

Summary

- A graphical interface in Java is an assembly of containers (Container) and of components (Component).
- A component is a "visible" part of the Java user interface. It is a subclass of the abstract class java.awt.Component (buttons, text or drawing areas, etc.).
- A container is a space in which we can position several components. Subclass of java.awt.Container class
- The Container class is itself a subclass of the Component class
- For example windows, applets, etc.

Summary

- The two most common containers are the Frame and the Panel.
- A Frame represents a top-level window with a title, border and resizing angles. Most applications use least a Frame as a starting point for their graphical interface.
- A Panel does not have a clean appearance and cannot be used as a standalone window. Panels are created and added to other containers of the same way as components such as buttons
- Panels can then redefine a presentation of their own to themselves contain other components.

Features

- We add a component to a container, with the add() method:

```
Panel p = new Panel();

Button b = new Button();

p.add(b);
```

- Similarly, a component is removed from its container by remove() method:

```
p.remove(b);
```

Features

- A component has (in particular):
  - o a preferred size that we obtain with getPreferredSize()
  - o a minimum size that we obtain with getMinimunSize()
  - o a maximum size that we obtain with getMaximunSize()

Label

- The **java.awt.Label** class represents descriptive text used to label another component (textual fields) or provide a textual description.

- To create a Label instance we have the following constructors:

```
1 public Label(String strLabel, int alignment);
2 public Label(String strLabel);
3 public Label();
```

- The **strLabel** and **alignment** parameters respectively describe the text and label alignment. Note that Awt offers values for the second parameter defined as static constants within the class **Label,** it is:

Label

```
public static final LEFT;     // Label.LEFT
public static final RIGHT;    // Label.RIGHT
public static final CENTER;   // Label.CENTER
```

Example:

```
Panel p=new Panel();
Label lblInput;

lblInput = new Label("Enter ID");

p.add(lblInput);
lblInput.setText("Enter password");

String ch= lblInput.getText();
```

Button

- The **java.awt.Button** class represents a button used to trigger an action following a user click. In order to build instances, awt offers to use the following constructors:

```
1 public Button(String btnLabel);
2 public Button();
```

Button

- It is also possible to retrieve/modify the label of a button and enable/disable a button using the following methods:

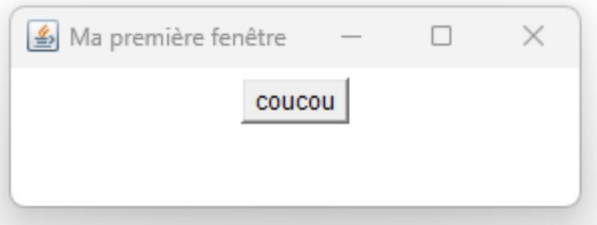
```
1 public String getLabel();
2 public void setLabel(String btnLabel);
3 public void setEnable(boolean enable);
```

Example :

```
Panel p =new Panel();
Button btnColor = new Button("Red");
p.add(btnColor);
btnColor.setLabel("Green");
String lb = btnColor.getLabel();
p.add(new Button("Blue"));
```

Exercise

- Create a “EssaiFenetre1” class representing a first interface with the title “My first window” and a Button with text “hello” as follows:



TextField

- Single-line text boxes, instances of **java.awt.TextField,** are intended to receive the text entered by the user.

- To create them, awt offers a set of its overload constructors to control the default text and/or component size in terms of number of columns. Note that the number of columns determines the size of the TextField but does not impose any constraints on the size of the text entered by the user.

```
public TextField(String initialText, int columns);
public TextField(String initialText);
public TextField(int columns);
```

TextField

- The methods below allow you to manipulate the text contained in a TextField and manage its activation and deactivation.

```
public String getText(); instance
public void setText(String strText);
public void setEditable(boolean editable);
```

TextField

Example:

```
Panel p =new Panel();
TextField tfIn = new TextField(30);
p.add(tfIn);
TextField tfRes = new TextField();
tfRes.setEditable(false) ;
p.add(tfRes);
try{
     int number = Integer.parseInt(tfIn.getText());
     number *= number;
     tfRes.setText(number + "");
}catch(NumberFormatException e){
     System.out.println(e.getMessage());
}
```

TextArea

- The awt API also supports multi-line text boxes across the class **java.awt.TextArea.**

- The developer can control the visual rendering by specifying a number of columns and lines to set the size of the input area. We have the constructors following steps to create TextArea instances:

```
TextArea(int rows, int columns)
TextArea(String text)
TextArea(String text, int rows, int columns)
TextArea(String text, int rows, int columns, int scrollbars)
```

TextArea

- To manipulate the entered text, modify it and or replace it, we have the methods:

```
void setText(String)
String getText()
void   append(String str)
void   insert(String str, int pos)
```

Example:

```
Panel p = new Panel();
TextArea ta = new TextArea("Zone de texte multi-ligne", 5, 40);
p.add(ta);
```

CheckBox and radio buttons

- Check boxes are graphic components with two states (on/off) and which switch from one state to another following a user click.

- Awt does not offer a dedicated class for creating radio buttons. However, we can work around this limitation by grouping checkboxes with an instance of the class **CheckboxGroup.**

- In fact, the latter immediately become exclusive selection simulating thus the behavior of the radio buttons.

CheckBox and radio buttons

- The builders:

```
Checkbox(String label)
Checkbox(String label, boolean state)
Checkbox(String label, boolean state, CheckboxGroup group)
Checkbox(String label, CheckboxGroup group, boolean state)
```

- The methods:

```
String  getLabel()
void    setLabel(String label)
boolean getState()
void    setState(boolean state)
```

Example:

```
Pnael p =new Panel();
CheckboxGroup cbg = new CheckboxGroup();
p.add(new Checkbox("one", cbg, true));
p.add(new Checkbox("two", cbg, false));
p.add(new Checkbox("three", cbg, false));
```

List

- The **java.awt.List** class allows you to choose/select one or more elements at within a fixed list of choices. We distinguish single-choice lists from choice lists. multiple. The constructors of the List class are:

```
List()
List(int rows)
List(int rows, boolean multipleMode)
```

List

- To manipulate List instances we have the following methods:

```
void      setMultipleMode(boolean b)
void      add(String item)
void      add(String item, int index)
String    getItem(int index)
int  getItemCount()
String[]      getItems()
int  getRows()
int  getSelectedIndex()
int[]     getSelectedIndexes()
String    getSelectedItem()
String[]      getSelectedItems()
Object[]      getSelectedObjects()
isIndexSelected(int).
```

List

Example:

```
Panel cnt = new Panel();
 List lst = new List(4, false);
 lst.setMultipleMode(true);//essayez avec false
 lst.add("Mercury");
 lst.add("Venus");
 lst.add("Earth");
 lst.add("JavaSoft");
 lst.add("Mars");
 lst.add("Jupiter");
 lst.add("Saturn");
 lst.add("Uranus");
 lst.add("Neptune");
 lst.add("Pluto");
 cnt.add(lst);
```

Choice

- Drop-down lists are instances of the **java.awt.Choice** class and allow the user to choose an item from a list. To build instances we use: public Choice ( )

- To manipulate drop-down lists we use the methods:

```
void      add(String item)
String    getItem(int index)
int       getItemCount()
int       getSelectedIndex()
String    getSelectedItem()
```

Choice

Example:

```
Panel p= new Panel();
Choice ColorChooser = new Choice();
ColorChooser.add("Green");
ColorChooser.add("Red");
ColorChooser.add("Blue");
p.add(ColorChooser);
```

Presentation Manager

- Each container is associated with a presentation manager (layout manager)
- The presentation manager manages the positioning and (re)sizing the components of a container.
- The rearrangement of components in a container takes place when:
  - o modification of its size,
  - o the change in size or movement of one of the components,
  - o adding, displaying, deleting or hiding a component.

Presentation Manager

- The main presentation managers of the AWT are: FlowLayout, BorderLayout, GridLayout, CardLayout, GridBagLayout.

- Every container has a default presentation manager.

  - Any instance of Container references an instance of LayoutManager.

  - It is possible to change it using the setLayout() method.

Presentation Manager

- The default presentation managers are:
  - o The BorderLayout for Window and its descendants (Frame, Dialog, etc.)
  - o The FlowLayout for Panel and its descendants (Applet, etc.)
- Once installed, a presentation manager works “by itself” by interacting with the container.

FlowLayout

- The FlowLayout is the simplest of the AWT managers. It's the presentation manager used by default in Panels if none LayoutManager is not specified.

- A FlowLayout can specify:

  - o left, right or centered justification,

  - o horizontal or vertical spacing between two components.

  - o By default, components are centered inside the area that they are is allocated.

FlowLayout

- The FlowLayout layout strategy is as follows:

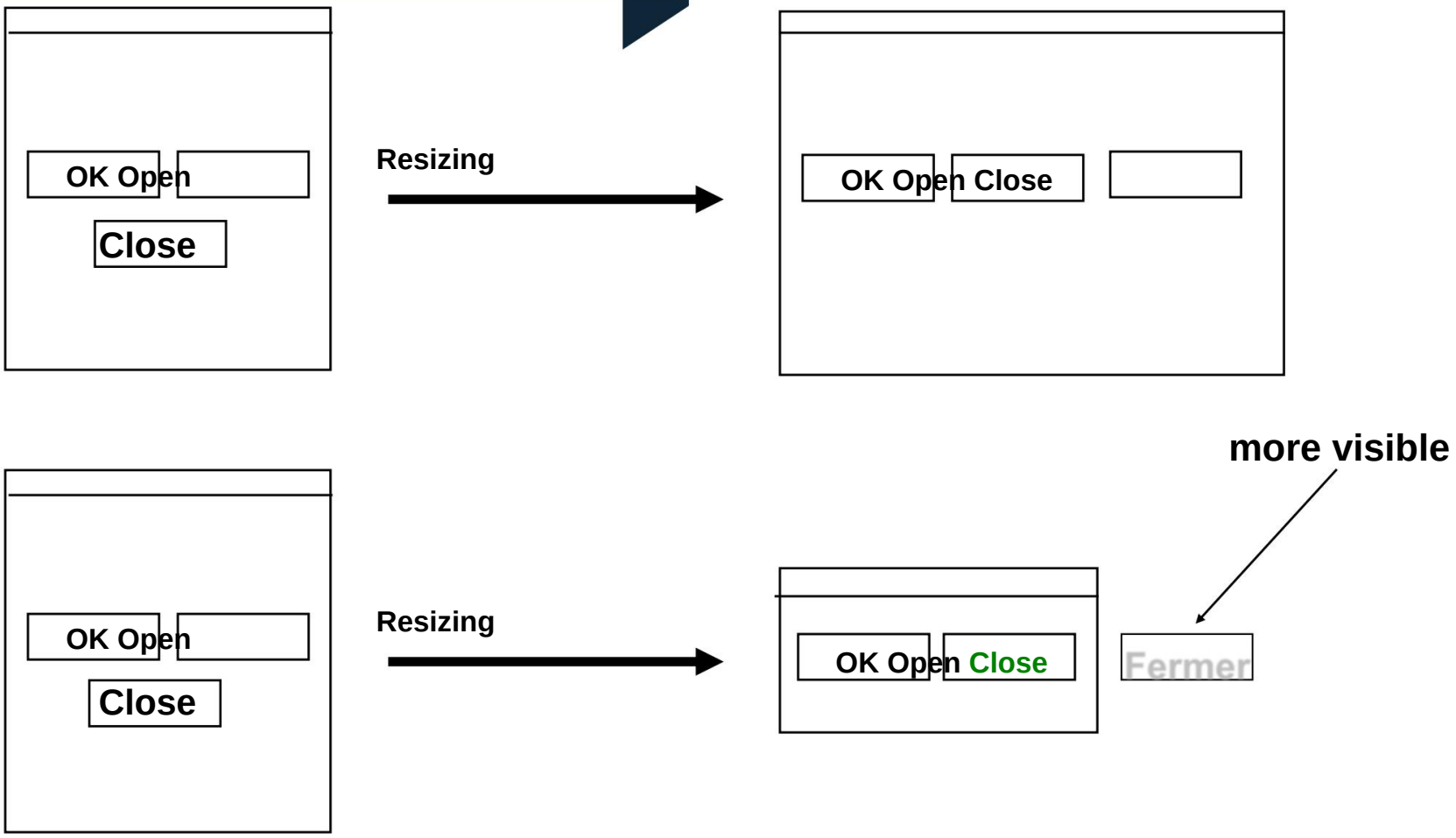
  o Respect the preferred size of all contained components.

  o Arrange as many components as can fit horizontally inside the Container object.

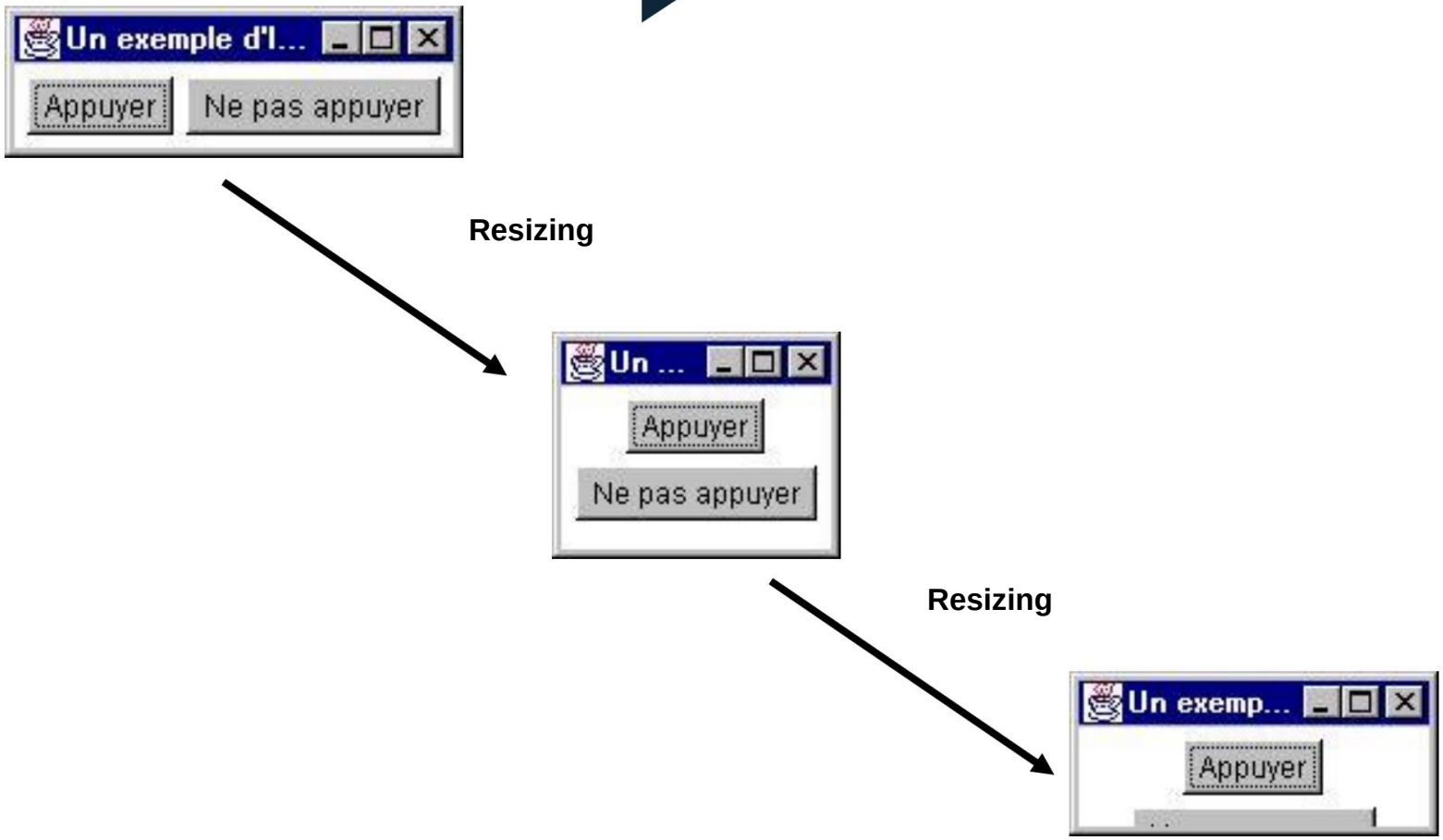
  o Start a new row of components if you cannot fit in a single row.

  o If all components cannot fit in the Container object, this is not managed (that is, components may not appear).

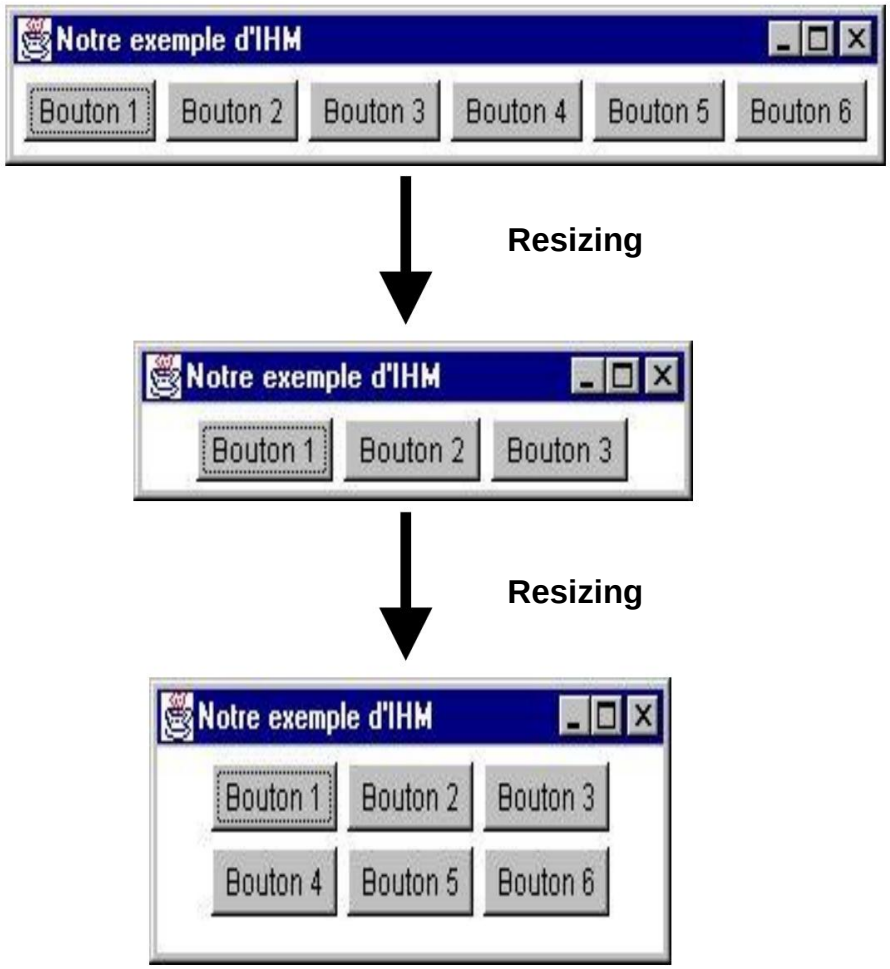
FlowLayout



FlowLayout



FlowLayout



FlowLayout

- The FlowLayout really and effectively hides the components that do not fit in the frame.
- FlowLayout is only of interest when there are few components.
- The vertical equivalent of FlowLayout does not exist
- The FlowLayout layout positions components line by line.
    - Each time a line is filled, a new line is started.
- The FlowLayout manager does not impose the size of the components but allows them to to have the size they prefer.

BorderLayout

- BorderLayout divides its workspace into five geographic zones: North, South, East, West and Center.

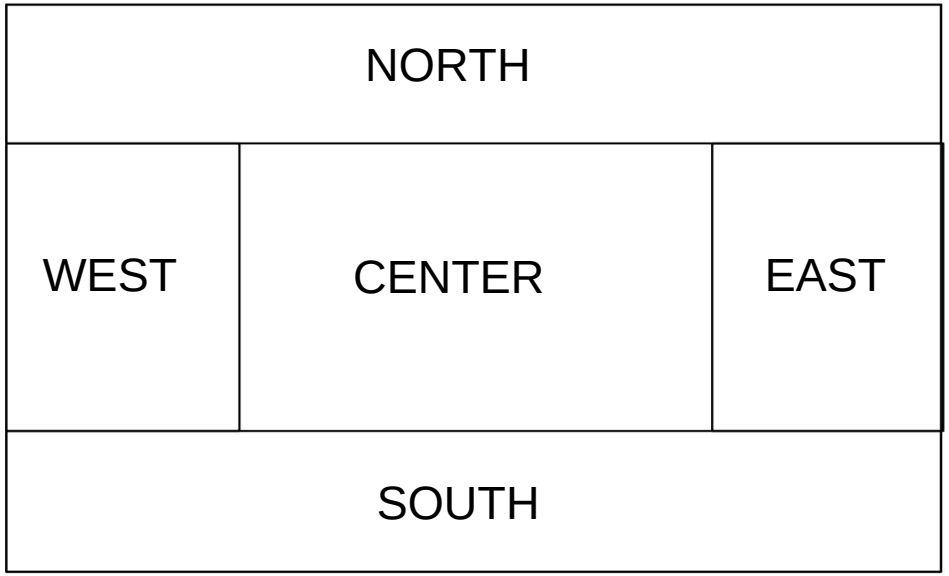
- Components are added by name to these zones (only one component per zone).

  - **Example:** add("North", new Button("The north button!"));

  - If one of the border areas contains nothing, its size is 0.

BorderLayout

- Space division with BorderLayout



BorderLayout

```
import java.awt.*;

public class EssayBorderLayout extends Frame
  { private Button b1,b2,b3,b4,
   b5; public EssayBorderLayout()
        { Panel P = new
        Panel(); P.setLayout(new
        BorderLayout()); b1 = new Button("North"); b2 = new
        Button("South"); b3 = new Button("Is"); b4 = new
        Button("West"); b5 = new
        Button("Center"); P.add(b1,
        BorderLayout.NORTH); P.add(b2,
        BorderLayout.SOUTH);
        P.add(b3, BorderLayout.EAST);
        P.add(b4, BorderLayout.WEST);
        P.add(b5, BorderLayout.CENTER); this.add(P); pack(); setVisible(true) ;}
public static void main(String args[]) {
    TestBorderLayout test = new TestBorderLayout(); }}
```

BorderLayout

- BorderLayout layout strategy
  - o If there is a component in the part placed in the **NORTH part,** it recovers his ***preferred size,*** respects his ***preferred height*** if possible and sets its width to the entire available width of the Container object.
  - o If there is a component in the part placed in the **SOUTH part,** it is the same as in the case of the **NORTH part.**
  - o If there is a component in the part placed in the **EAST part,** it retrieves its **preferred size,** respects its ***preferred width*** if possible and sets its height to the entire height still available.

BorderLayout

- BorderLayout layout strategy
  - o If there is a component in the part placed in the **WEST part,** it makes the same as in the case of the **EAST part.**
  - o If there is a component in the **CENTER part,** it gives it the place that remains, if there is any left.

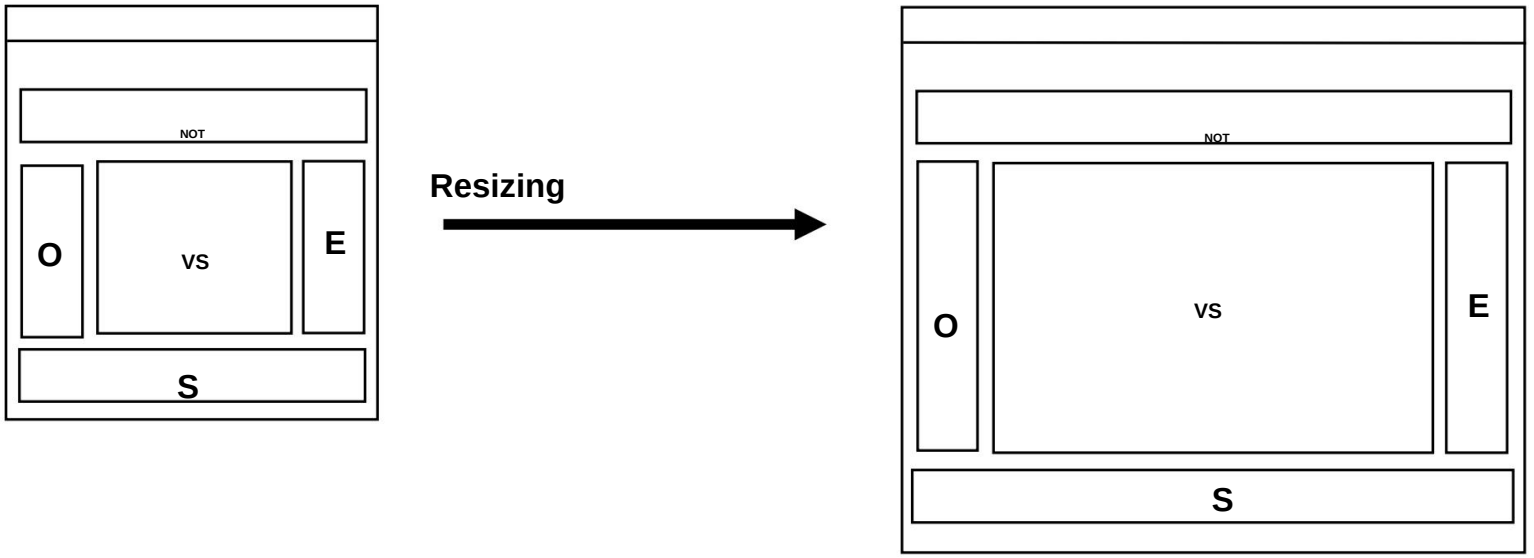
BorderLayout

- When resizing, the component itself is resized according to

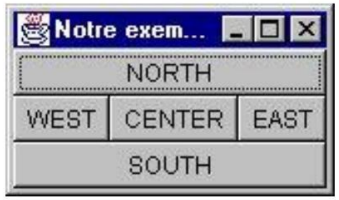
  of the size of the zone, i.e.:

  - the northern and southern zones are possibly widened but not lengthened.
  - the east and west zones are possibly lengthened but not widened,
  - the central area is stretched in both directions.

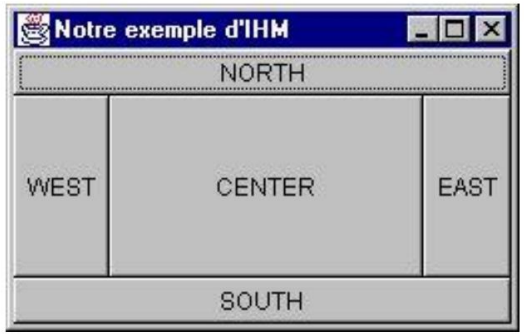
BorderLayout



BorderLayout



Resizing



Resizing

GridLayout

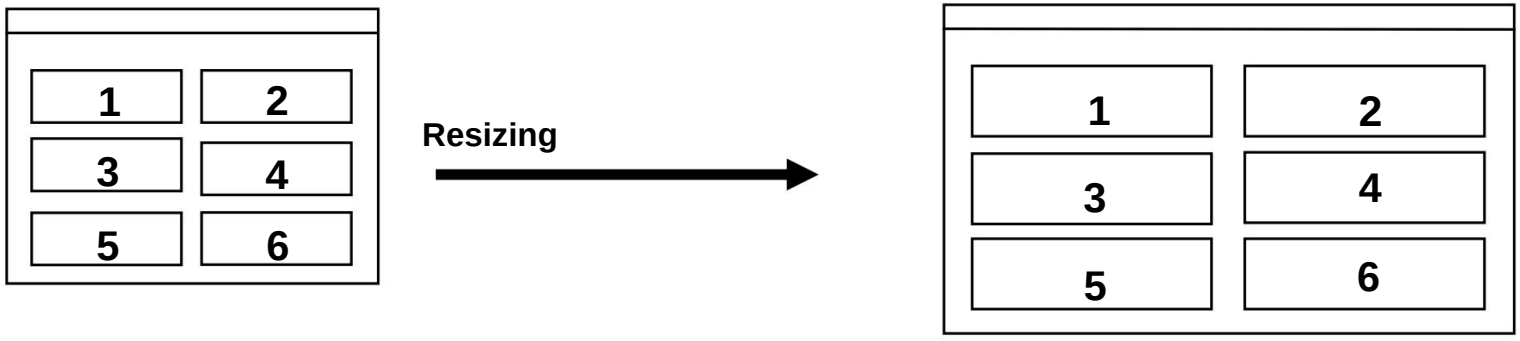
- The GridLayout arranges the components in a grid.
  - o Division of the display area into rows and columns which define cells of equal dimensions.
  - o Each component is the same size
    - ÿ when they are added in the cells the filling is done from the left to the right and from top to bottom.
  - o The 2 parameters are the rows and the columns.
  - o Construction of a GridLayout: **new GridLayout(3,2);**

GridLayout

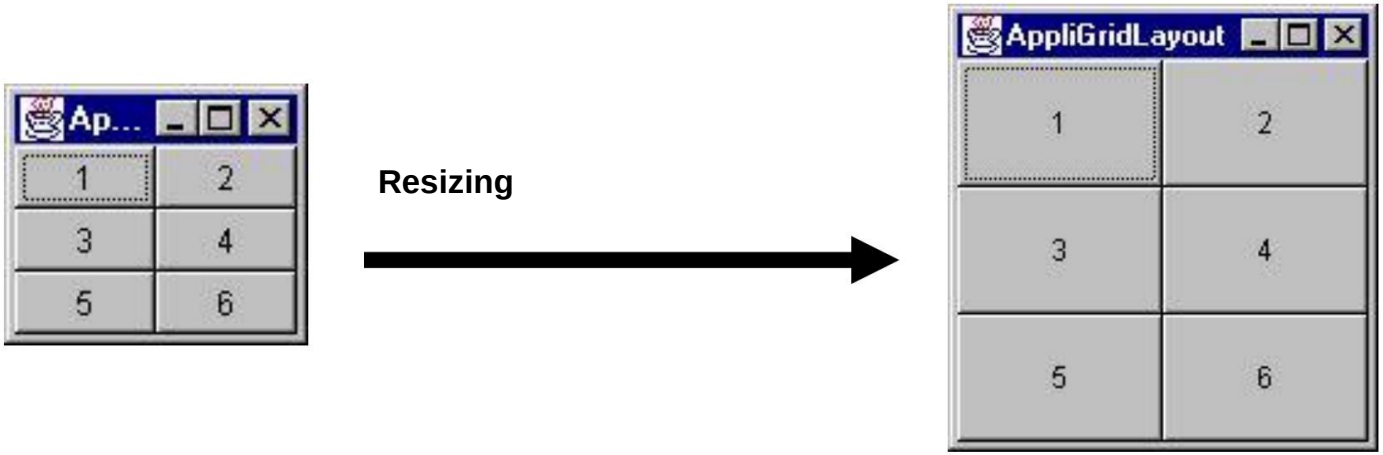
```
import java.awt.*;
public class AppliGridLayout extends Frame
{ public AppliGridLayout()

 { super("AppliGridLayout");
  this.setLayout(new GridLayout(3,2));
  for (int i = 1; i < 7; i++)
    add(new Button(Integer.toString(i)));
  this.pack();
  this.show(); }


public static void main(String args[])
 {
  AppliGridLayout app = new AppliGridLayout(); } }
```

GridLayout

- When resizing, the components all change size but their relative positions do not change.



GridLayout



CardLayout

- The CardLayout only displays one component at a time:
  - o the components are considered stacked, like a pile of cards.
- CardLayout layout allows multiple components to share the same display space so that only one of them is visible at a time.
- To add a component to a container using a CardLayout you must use

  add(String key, Component myComponent)
- Allows you to switch from the display of one component to another by calling the methods

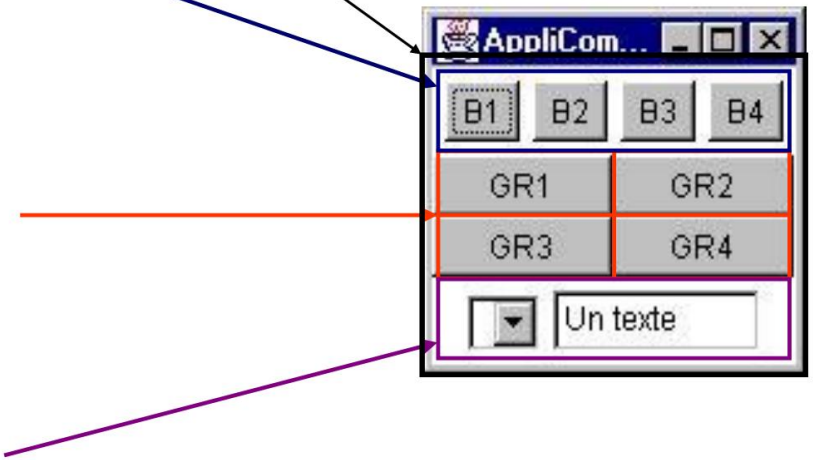
  first, last, next, previous or **show**

GridBagLayout

- GridBagLayout manager provides complex layout functions
  - o based on a grid whose rows and columns are of variable size.
  - o allows simple components to take their preferred size within a cell, instead of filling the entire cell.
  - o also allows the extension of the same component over several cells.
- The GridBagLayout is complicated to manage.
  - o In most cases, it is possible to avoid using it by combining Container objects using different handlers.
- See the docs

Complex formatting

```
super("ComplexAppLayout");
setLayout(new BorderLayout());
Panel pnorth = new Panel();
pnorth.add(b1); pnorth.add(b2);
pnorth.add(b3); pnorth.add(b4);
this.add(pnorth,BorderLayout.NORTH);

Panel pcenter = new Panel();
pcenter.setLayout(new GridLayout(2,2));
pcenter.add(gr1); pcenter.add(gr2);
pcenter.add(gr3); pcenter.add(gr4);
this.add(pcenter,BorderLayout.CENTER);

Panel psouth = new Panel();
psouth.add(ch); psouth.add(tf);
this.add(psouth, BorderLayout.SOUTH);
```



Other managers?

- We can impose on a “container” object not to have a manager by setting its LayoutManager set to null
  - **Frame f = new Frame(); f.setLayout(null);**
  - It is then up to the programmer to position each of the components “manually” by indicating their absolute position in the window marker.
  - This should be avoided, except in special cases.
- It is possible to write your own LayoutManager…

Summary

- FlowLayout
  - o Flow: components placed one behind the other
- BorderLayout
  - o Screen divided into 5 zones (“North”, “West”, “South”, “East”, “Center”)
- GridLayout
  - o Grid: one box per component, each box of the same size
- CardLayout
  - o “Tabs”: we display one element at a time
- GridBagLayout
  - o Complex grid: several boxes per component