

Graphical User Interfaces

User Interface Development

- Generalization of use \Rightarrow Increasing importance of the computer of the ease of use
- The user interface does not add proper functionality to an application
 - However it is a decisive aspect for the success of an application
- Very costly development
 - Typically, 50% of the development effort is devoted to the UI
- Tools that help in the development of UIs

2

A Tiny Bit of History

- 1981 – Xerox Star
- 1984 – Apple Macintosh, X Windows
- 1985 – MS Windows

3

Tools: Software for UI Development (I)

- Based on a higher-level understanding of UIs
 - Windows, buttons, menus, text fields, etc., rather than plain pixels
 - User actions, UI behaviour, etc., rather than input device signals
- Easier UI development and maintenance, reduced cost
 - Provide part of the code
 - Help organize and structure the code written by the programmer
- Consistency of UIs across applications
- Tradeoff: expressive limitations
- Without toolkits: draw and redraw pixels on the screen, track mouse movement, check screen regions (coordinates), manage input signals

4

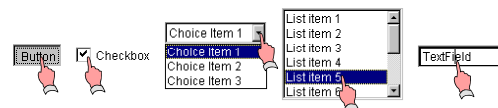
Tools: Software for UI Development (II)

- Window systems / toolkits: MS Windows, X + Motif, Mac/OS, OS/2 PM, etc.
 - Interface to the OS message system
 - Libraries for GUI development
 - Low level, high expressive power, hard to use
- Object libraries: Java AWT, MS MFC, Borland OWL, etc.
 - Code easier to produce and maintain
 - Usually, interactive interface builders available
- Interface builders: Visual Basic, Visual C++ App Studio, NetBeans, etc.
 - Visual programming
 - High ease of use, limited scope
- Building more advanced tools is an open research field

5

¿What Have GUIs Got to Do with OOP?

- The user sees objects on the screen, s/he can manipulate the objects
- GUI objects have their own behaviour: different response to same user actions
 - Example: click the mouse on a GUI object



- No “click mouse” function that knows what to do depending on the element on which it acts
- Each GUI element has in itself the functionality for the mouse clicking action (the responsibility resides in the object)
- Event-based (message-based) programming

6

A New Programming Model: Event-Based Programming

- The "execution" of a UI does not follow a strictly sequential control flow
- The user has a high degree of freedom most of the time: wide set of possible actions available
- It would be extremely difficult to account for all possible execution paths in a traditional programming model

7

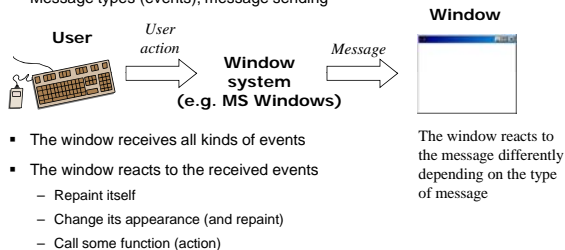
Event-Based Programming

- Used in modern window-based UIs
- The user takes the initiative, not the program
- The program is broken down into subprograms associated to different windows or UI components
- Components remain waiting for user actions
- User actions generate events that are pushed into an event queue
- The event processing system pops events from the queue and dispatches them to the proper programs
- The programs process the received events, responding differently depending on the event type
- Each type of component is characterized by its own way to respond to events

8

Window Systems

- Graphic drawing primitives: geometric forms, text, etc.
- Message types (events), message sending



- The window receives all kinds of events
- The window reacts to the received events
 - Repaint itself
 - Change its appearance (and repaint)
 - Call some function (action)

9

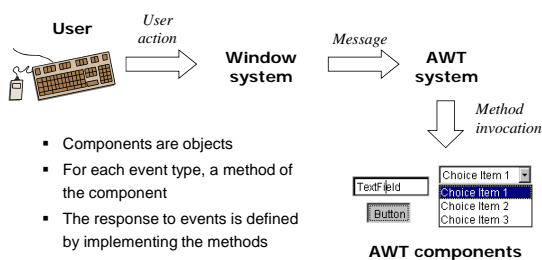
No Objects in MS Windows

No concept of UI object exists, but it is implicit

- Window class \approx OO class
- Case clauses in window procedure \approx methods
- Process that runs the window procedure \approx object of the class

10

Java Abstract Window Toolkit (AWT)



- Components are objects
- For each event type, a method of the component
- The response to events is defined by implementing the methods

11

Abstract Window Toolkit (AWT)

Introduction

The AWT Library

`java.awt`, `java.awt.event`

- **Components**
 - Predefined components
 - Component aggregation
 - GUIs paint themselves: drawing methods
 - Advanced: creation of custom components
- **Interaction** with the user: event handling
 - Predefined components generate events in response to user actions
 - Listening to and processing events
 - Advanced: sending events
- **Component layout**
 - Manual or automatic

13

Building a UI

- Compose UIs with predefined component classes
 - The `Container` class for aggregating subcomponents
 - The appearance of components is controlled by manipulating their properties
- Define the layout of container parts
 - Absolute position
 - Layout managers
- Handle events: emitter / listener model
 - Action events generated by predefined classes
 - Direct user input handling
- Define custom components
 - The `Canvas` class
 - Drawing methods

14

Example: a Simple Text Editor

15

Predefined Components

16

Component Layout

17

Interactive Features

18

```
import java.awt.*;
import java.awt.event.*;
import java.io.*;

public class Editor extends Frame implements WindowListener,
                                           ActionListener {

    TextField fileName;
    TextArea fileBuffer;
    Button load, save, quit;

    Editor () {
        setLayout (null);

        Label label = new Label ("File Name: ");
        label.setBounds (10, 30, 300, 20);
        add (label);

        fileName = new TextField ();
        fileName.setBounds (10, 50, 290, 20);
        add (fileName);
        ...
    }
}
```

19

```
...
load = new Button ("Load");
load.setBounds (40, 80, 60, 20);
add (load);

save = new Button ("Save");
save.setBounds (120, 80, 60, 20);
add (save);

quit = new Button ("Quit");
quit.setBounds (200, 80, 60, 20);
add (quit);

fileBuffer = new TextArea ();
fileBuffer.setBounds (10, 110, 300, 200);
add (fileBuffer);

this.addWindowListener (this);
load.addActionListener (this);
save.addActionListener (this);
quit.addActionListener (this);
} // End constructor
...
```

20

```
...
public void actionPerformed (ActionEvent e) {
    String command = e.getActionCommand ();
    if (command.equals ("Quit")) dispose ();
    else if (command.equals ("Load")) load ();
    else if (command.equals ("Save")) save ();
}

public void windowClosing (WindowEvent e) { dispose (); }
public void windowActivated (WindowEvent e) {}
public void windowClosed (WindowEvent e) {}
public void windowDeactivated (WindowEvent e) {}
public void windowDeiconified (WindowEvent e) {}
public void windowIconified (WindowEvent e) {}
public void windowOpened (WindowEvent e) {}
...
```

21

```
...
void load () {
    try {
        RandomAccessFile input =
            new RandomAccessFile (fileName.getText (), "r");
        byte buffer[] = new byte [(int) input.length ()];
        input.read (buffer);
        input.close ();
        fileBuffer.setText (new String (buffer));
    } catch (IOException e) { System.out.println (e); }
}

void save () {
    try {
        FileWriter output =
            new FileWriter (fileName.getText ());
        output.write (fileBuffer.getText ());
        output.close ();
    } catch (IOException e) { System.out.println (e); }
}
...
```

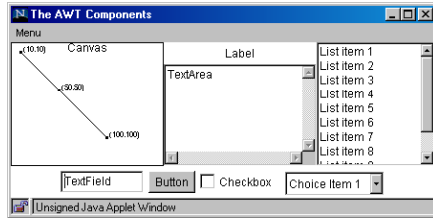
22

```
...
public static void main (String args[]) {
    Editor edit = new Editor ();
    edit.setSize (320, 320);
    edit.setVisible (true);
}
} // End class Editor
```

23

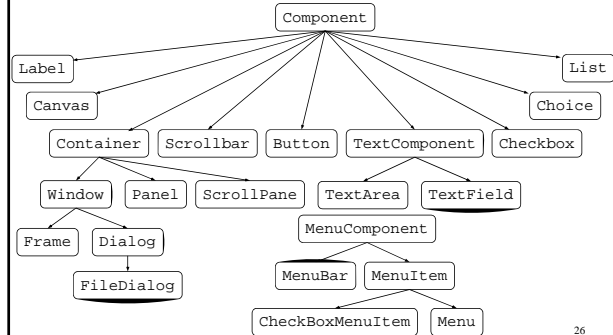
Components

Predefined AWT Components



25

AWT Component Hierarchy: the java.awt Package



26

Predefined Components: Example

```
import java.awt.*;

public class MyWin extends Frame {
    public MyWin () {
        setLayout (new FlowLayout ());

        //Set up the menu bar
        Menu m = new Menu ("Menu");
        m.add (new MenuItem ("Menu item 1"));
        m.add (new CheckboxMenuItem ("Menu item 2"));
        m.add (new MenuItem ("Menu item 3"));
        m.add (new MenuItem ("-"));

        MenuBar mb = new MenuBar ();
        mb.add (m);
        setMenuBar (mb);
        ...
    }
}
```

27

```
...
add (new TextField ("TextField"));
add (new Button ("Button"));
add (new Checkbox ("Checkbox"));

Choice c = new Choice ();
c.add ("Choice Item 1");
c.add ("Choice Item 2");
c.add ("Choice Item 3");
add (c);

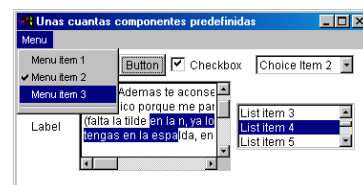
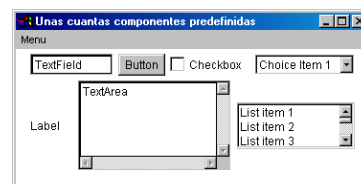
add (new Label ("Label"));
add (new TextArea ("TextArea", 5, 20));

List l = new List (3, false);
for (int i = 1; i <= 10; i++) {
    l.add ("List item " + i);
}
add (l);
} // End MyWin constructor
...
```

28

```
...
public static void main (String[] args) {
    Mywin w = new MyWin ();
    w.setTitle ("A few predefined components");
    w.pack ();
    w.setVisible (true);
}
} // End MyWin class
```

29



30

Using Predefined Components

- Predefined components provide:
 - A visual aspect based on their self-drawing method (`paint`)
 - A predefined response to events from the user
 - An API to control the specific properties of the component: `setFont()`, `setBackground()`, `setForeground()`, and others
- The programmer:
 - Creates instances of the predefined classes
 - The appearance is controlled by setting the component properties (state)
 - Subclasses predefined components overriding their methods and/or defining a constructor to create subcomponents and define an initial state
 - Usually `paint()` is not redefined, unless custom graphics are needed
 - Can create her own components from scratch implementing `paint()` in a `Canvas` to draw the components

31

Component Aggregation: the `java.awt.Container` Class

- UI are built by aggregating components as interface parts
- The component hierarchy must be rooted in a window or applet
- A component can only be added to a `Container`
 - `add(Component)`
 - `remove(Component)`
- Windows cannot be parts of another component
- The layout of components in a container can be defined:
 - With different types of layout manager
 - Manually, with absolute positions

32

Component Aggregation (contd.)

- Components can be added:
 - In the constructor of the container (the usual way)
 - In main
 - In any method
- } Fixed structure
} Dynamic structure
- Traversing the component tree:
 - Access parts: `getComponent(int)`, `getComponentAt(int, int)`, `getComponentCount()`, `getComponents()`
 - Access the container from one of its parts: `getParent()`

33

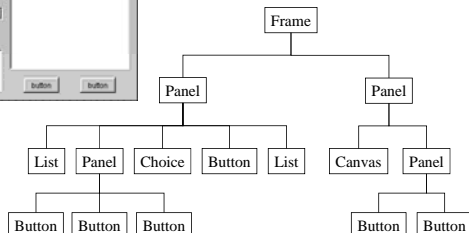
Subcontainers: What for?

- Move a group of components as a single block
- Add, remove, show, hide groups of components
- Simplify layout: different layouts in different window regions

34



Example



35

Component Appearance: `paint(Graphics)` of Component

- The appearance on the screen of each subclass of `Component` is defined by the implementation of `paint(Graphics)`
- The implementation of `paint` consists of calls to drawing methods on the object of type `Graphics`
- It is not very practical to redefine the `paint` method of predefined components (In contrast, it is easy to change their response to events)
 - Usually the `paint` implementation of predefined components is not overridden
 - To change their appearance, change their state (the system shows the changes on screen automatically)
 - In custom components that are defined from scratch, `paint` is redefined
- It is important that `paint` is not slow to execute
 - Use threads if necessary to avoid blocking component redraw

36

Component Refresh: update(Graphics) and repaint()

Components draw themselves when:

- The AWT system automatically decides so, by calling `update(Graphics)`
 - This happens when:
 - The window system generates certain type of update events as a consequence of the manipulation of windows by the user
 - The program changes visual properties of components
 - This call is made from the event processing thread
 - `update(Graphics)` erases the area to refresh and calls `paint(Graphics)`
- The program requests it: `repaint()`
 - Request to the AWT system to call `update` as soon as possible
 - The program must call `repaint` to refresh its custom components when they undergo state changes that must be shown in their appearance
 - The program must never call `paint` directly

37

Basic Classes: java.awt.Component

- Draw itself on the screen: `paint()`, `update()`, `repaint()`
- Event processing: methods for different types of events (*deprecated*)
- Visual appearance:
 - Color: `setForeground(Color)`, `getForeground()`, `setBackground(Color)`, `getBackground()`
 - Font: `setFont(Font)`, `getFont()`
 - Cursor: `setCursor(Cursor)`, `getCursor()`
- Size and position: `setSize(int, int)`, `getSize()` → `Dimension`, `getLocation()` → `Point`, `getLocationOnScreen()` → `Point`, `setBounds(int, int, int, int)`, `getBounds()` → `Rectangle` (The layout manager may alter these values)
- Show details in `System.out: list()`

38

Basic Classes: java.awt.Graphics

- Passed as a parameter to `paint()` and `update()` by the AWT system
- Drawing primitive graphic forms:
 - `drawLine(int, int, int, int)`, `drawRect(int, int, int, int)`, `drawArc(int, int, int, int, int, int)`, `fillRect(int, int, int, int)`
 - `drawString(String, int, int)`
 - `drawImage(Image, int, int, int, int, ImageObserver)`
- State:
 - A `Graphics` object is associated 1-1 to a `Component`
 - `getGraphics()` of `Component`
 - Origin of coordinate system: `translate(int, int)`
 - Clip area: `getClip()`, `setClip(int, int, int, int)`
 - Color: `setColor(Color)`, `getColor()`
 - Font: `setFont(Font)`, `getFont()`
 - XOR mode: `setXORMode(Color)`

39

Basic Classes: Auxiliary Classes in java.awt

- Size and position:
 - `Dimension: width, height`
 - `Point: x, y`
 - `Rectangle: x, y, width, height, contains(Point)`
 - `Polygon: npoints, xpoints, ypoints, addPoint(Point)`
 - Color:
 - `new Color(0.8f, 0.3f, 1.0f)` in RGB
 - Constants of type `Color: Color.white, Color.blue, etc.`
 - Functions for RGB ↔ HSB conversion
 - Font:
 - `new Font("Helvetica", Font.BOLD + Font.ITALIC, 18)`
 - `getName(), getStyle(), getSize()`
 - Constants for font styles: `Font.BOLD, Font.ITALIC, Font.PLAIN`
 - Cursor:
 - `new Cursor(Cursor.HAND_CURSOR), Cursor.CROSSHAIR_CURSOR, etc.`
- They are not graphic objects*

40

Layout

Layout Management

- ¿What does layout mean?
 - Spatial arrangement of a set of components
- ¿What is a layout manager?
 - An object of a class implementing the `LayoutManager` interface
 - It globally controls the size and position of the components of a `Container`
 - There are different manager classes for different types of layout
 - `setLayoutManager(LayoutManager)` of `Container`
 - Each class of container has a type of layout manager by default
 - For example: `Window` → `BorderLayout`, `Panel` → `FlowLayout`
- ¿Why use layout managers?
 - The size and position of a component often depends of other surrounding components and the available space for the whole group
 - A layout manager applies layout rules on top of the components
 - The components negotiate their position and size with the layout manager

42

Types of Layout Manager

- Basics: BorderLayout, FlowLayout, GridLayout
- Advanced:
 - CardLayout: two or more components share the same region
 - GridBagLayout:
 - Rows and columns of variable size
 - Components may span several cells

43

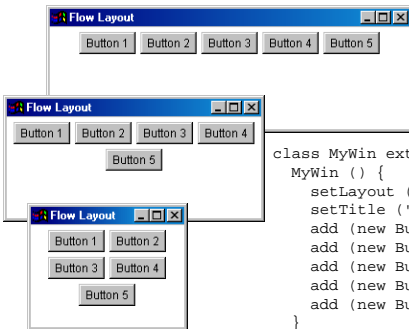
BorderLayout



```
class MyWin extends Frame {
    MyWin () {
        setLayout (new BorderLayout ());
        setTitle ("Border Layout");
        add("North", new Button("North"));
        add("South", new Button("South"));
        add("East", new Button("East"));
        add("West", new Button("West"));
        add("Center", new Button("Center"));
    }
}
```

44

FlowLayout



```
class MyWin extends Frame {
    MyWin () {
        setLayout (new FlowLayout ());
        setTitle ("Flow Layout");
        add (new Button ("Button 1"));
        add (new Button ("Button 2"));
        add (new Button ("Button 3"));
        add (new Button ("Button 4"));
        add (new Button ("Button 5"));
    }
}
```

45

FlowLayout (II)

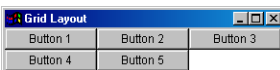


```
class MyWin extends Frame {
    MyWin () {
        FlowLayout f = new FlowLayout ();
        f.setAlignment(FlowLayout.RIGHT);
        f.setHgap (20);
        setLayout (f); setTitle ("Flow Layout");
        add (new Button ("Button 1"));
        add (new Button ("Button 2"));
        add (new Button ("Button 3"));
        add (new Button ("Button 4"));
        add (new Button ("Button 5"));
        add (new Button ("Button 6"));
    }
}
```

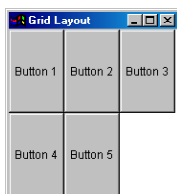
All layout managers have setHgap, setVgap

46

GridLayout



- setRows(int), setColumns(int)
- GridLayout(0,n) → as many rows as needed



```
class MyWin extends Frame {
    MyWin () {
        setLayout (new GridLayout (2,3));
        setTitle ("Grid Layout");
        add (new Button ("Button 1"));
        add (new Button ("Button 2"));
        add (new Button ("Button 3"));
        add (new Button ("Button 4"));
        add (new Button ("Button 5"));
    }
}
```

47

Interaction with the User

Event Handling

Responding to Events: Example

```
class MyWin extends Frame implements MouseListener {
    MyWin () { addMouseListener (this); }

    public void mouseClicked (MouseEvent e) {
        System.out.println ("The user has clicked on me");
    }

    public void mouseEntered (MouseEvent e) {}
    public void mouseExited (MouseEvent e) {}
    public void mousePressed (MouseEvent e) {}
    public void mouseReleased (MouseEvent e) {}
}
```

49

Responding to Events

1. Implement the listener interface that corresponds to the type of event
 - There is a correspondence between type of event and type of listener
2. Implement all the methods of the interface
 - Each method corresponds to a subvariety of the event type
3. Register as listener of an event source component
 - Each type of component can fire certain types of events
4. The AWT system takes care of the rest

The classes implementing listeners can be components, or any other class
- it is up to the choice of the programmer

50

The Event Model

- Events are objects of different subclasses of `AWTEvent`
- Events are generated when:
 - User input comes in: `MouseEvent`, `KeyEvent`
 - A widget is acted upon: `ActionEvent`, `TextEvent`, `AdjustmentEvent`
 - A window is manipulated: `WindowEvent`
 - Other causes: `ContainerEvent`, `ComponentEvent`, `PaintEvent`, etc.
- Events take place in the context of a component: the event **source**
- Other components can register for different types of events fired by different source components: **listeners**
- To be a listener of a certain type of events, a class must implement the corresponding **listener** interface

51

Elements Involved in Handling a Type of Event

For each event type `xxxEvent` we have:

- A type of listener `xxxListener` (except `MouseEvent` which has 2 listeners)
- A list of component classes that can fire the event
- An `addxxxListener` method to register listeners of the event type
 - This method is defined in the classes that can fire the event
 - A component may only register listeners of the event types that the component can fire

52

Example

Event class: `ActionEvent`

Source components: `Button`, `TextField`, `List`, `MenuItem`

Listener interface: `ActionListener`

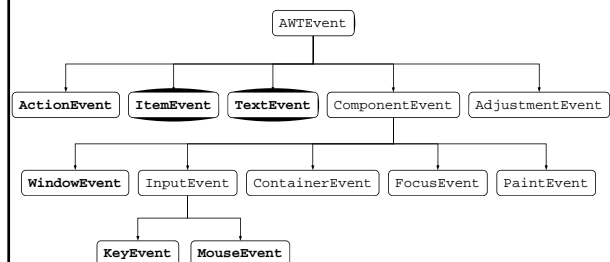
Methods to implement in the listener class: `actionPerformed (ActionEvent)`

Method to register a listener: `addActionListener (ActionListener)`

A component can only take listeners of the event types the component can fire

53

Event Classes: the `java.awt.event` package



54

Events Fired by Each Component Type

Component type	Event type										
	Mouse	MouseMotion	Key	Action	Window	Text	Item	Container	Component	Adjustment	Focus
Component	X	X	X						X		X
Canvas	X	X	X						X		X
Label	X	X	X						X		X
Button	X	X	X	X					X		X
Checkbox	X	X	X				X		X		X
Choice	X	X	X				X		X		X
List	X	X	X	X			X		X		X
TextField	X	X	X	X		X			X		X
TextArea	X	X	X			X			X		X
TextComponent	X	X	X			X			X		X
Scrollbar	X	X	X						X	X	X
MenuItem				X							
CheckboxMenuItem				X			X				

Component type	Event type										
	Mouse	MouseMotion	Key	Action	Window	Text	Item	Container	Component	Adjustment	Focus
Component	X	X	X						X		X
Container	X	X	X						X	X	X
Panel	X	X	X						X	X	X
ScrollPane	X	X	X						X	X	X
Window	X	X	X		X				X	X	X
Frame	X	X	X		X				X	X	X
Dialog	X	X	X		X				X	X	X

56

- ### Methods of Listener Interfaces (I)
- **MouseListener**
 - mouseClicked(MouseEvent)
 - mousePressed(MouseEvent)
 - mouseReleased(MouseEvent)
 - mouseEntered(MouseEvent)
 - mouseExited(MouseEvent)
 - **MouseMotionListener**
 - mouseMoved(MouseEvent)
 - mouseDragged(MouseEvent)
 - **KeyListener**
 - keyTyped(KeyEvent)
 - keyPressed(KeyEvent)
 - keyReleased(KeyEvent)
 - **ActionListener**
 - actionPerformed(ActionEvent)
- 57

- ### Methods of Listener Interfaces (II)
- **WindowListener**
 - windowActivated(WindowEvent)
 - windowDeactivated(WindowEvent)
 - windowOpened(WindowEvent)
 - windowClosing(WindowEvent)
 - windowClosed(WindowEvent)
 - windowIconified(WindowEvent)
 - windowDeiconified(WindowEvent)
 - **TextListener**
 - textValueChanged(TextEvent)
 - **ItemListener**
 - itemStateChanged(ItemEvent)
- 58

- ### Methods of Listener Interfaces (III)
- **ContainerListener**
 - componentAdded(ContainerEvent)
 - componentRemoved(ContainerEvent)
 - **ComponentListener**
 - componentShown(ComponentEvent)
 - componentHidden(ComponentEvent)
 - componentMoved(ComponentEvent)
 - componentResized(ComponentEvent)
 - **AdjustmentListener**
 - adjustmentValueChanged(AdjustmentEvent)
 - **FocusListener**
 - focusGained(FocusEvent)
 - focusLost(FocusEvent)
- 59

- ### ¿What Should Listener Methods Do?
- Change interface characteristics, appearance, or state
 - Change colors, fonts, labels, etc.
 - Move components, change their size
 - Hide, show, add, remove components
 - Open a dialog box
 - etc.
 - Call application programs
 - Usually some result is shown in the interface
- 60

¿Process or Ignore Events?

- Low-level events captured in widgets and elaborated into higher-level events
 - Buttons: MouseEvent → ActionEvent
 - Text widgets: MouseEvent, KeyEvent → TextEvent, ActionEvent
 - Selection widgets: MouseEvent → ItemEvent, ActionEvent
 - Etc.
- Events that denote component state changes: process the events immediately vs. access the state when needed
 - TextEvent, ItemEvent, ComponentEvent, ContainerEvent, AdjustmentEvent, etc.

61

Examples

```
class CoolButtons extends Editor implements MouseListener {
    CoolButtons () {
        super ();
        load.addMouseListener (this);
        save.addMouseListener (this);
        quit.addMouseListener (this);
    }
    public void mouseClicked (MouseEvent e) {}
    public void mousePressed (MouseEvent e) {
        ((Component) e.getSource()).setForeground (Color.green);
    }
    public void mouseReleased (MouseEvent e) {
        ((Component) e.getSource()).setForeground (Color.blue);
    }
    public void mouseEntered (MouseEvent e) {
        ((Component) e.getSource()).setForeground (Color.blue);
    }
    public void mouseExited (MouseEvent e) {
        ((Component) e.getSource()).setForeground (Color.black);
    }
}
```

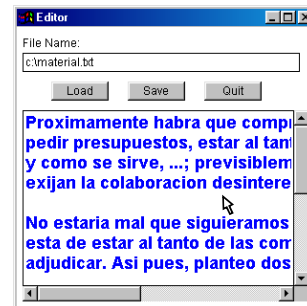
1. Buttons change color when pointing and clicking

62

```
class CoolFonts extends CoolButtons {
    CoolFonts () {
        super ();
        fileBuffer.addMouseListener (this);
    }
    public void mouseEntered (MouseEvent e) {
        super.mouseEntered (e);
        Component c = (Component) e.getSource ();
        if (c == fileBuffer) {
            Font f = c.getFont ();
            c.setFont (new Font (f.getName (), Font.BOLD,
                f.getSize () + 6));
        }
    }
    public void mouseExited (MouseEvent e) {
        super.mouseExited (e);
        Component c = (Component) e.getSource ();
        if (c == fileBuffer) {
            Font f = c.getFont ();
            c.setFont (new Font (f.getName (), Font.PLAIN,
                f.getSize () - 6));
        }
    }
}
```

2. Text font bigger and bold when mouse over button

63



64

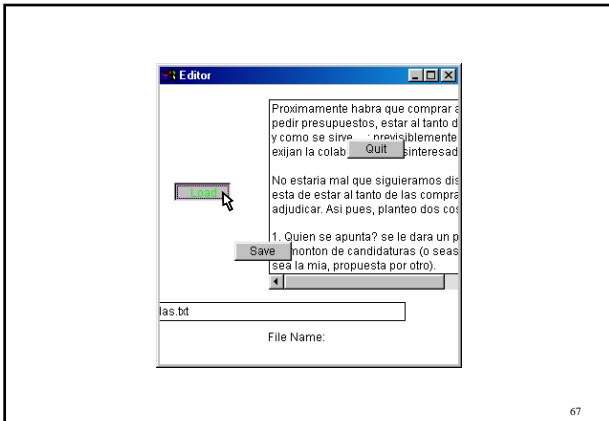
3. All components can move

```
class MovingParts extends CoolFonts
    implements MouseMotionListener {
    Point initialPoint = null;
    MovingParts () {
        super ();
        Component components [] = getComponents ();
        for (int i = 0; i < components.length; i++) {
            components[i].addMouseMotionListener (this);
            components[i].removeMouseListener (this);
            components[i].addMouseListener (this);
        }
    }
    ...
}
```

65

```
...
public void mousePressed (MouseEvent e) {
    super.mousePressed (e);
    initialPoint = e.getPoint ();
}
public void mouseDragged (MouseEvent e) {
    Component c = (Component) e.getSource ();
    Point p = c.getLocation ();
    c.setLocation (p.x + e.getX () - initialPoint.x,
        p.y + e.getY () - initialPoint.y);
}
public void mouseMoved (MouseEvent e) {}
}
```

66



67

Contents of Event Classes

The different event classes include:

- Constants (static final variables)
 - ID of the subtypes of events of an event class
E.g. `MouseEvent.MOUSE_MOVED`, `KeyEvent.KEY_RELEASED`
 - Constants for values of certain event properties (values returned by methods)
E.g. `ItemEvent.SELECTED`, `ItemEvent.DESELECTED`
- Methods
 - Return additional information about the event
E.g. `getX()`, `getY()` of `MouseEvent`, `getKeyChar()` of `KeyEvent`, `getID()` of `AWTEvent`

68

Information Contained in Events (I)

- **AWTEvent**
 - `getID()`, `getSource()`, `toString()`
- **InputEvent**
 - `getWhen()`, `isShiftDown()`, `isControlDown()`, `isAltDown()`
 - `getModifiers()` → `BUTTON1_MASK`, `BUTTON2_MASK`, `BUTTON3_MASK`
- **MouseEvent**
 - `getClickCount()`, `getX()`, `getY()`
- **KeyEvent**
 - `getKeyChar()`, `getKeyText()`
- **ActionEvent**
 - `getActionCommand()` → `String`
 - `getModifiers()` → `ALT_MASK`, `CTRL_MASK`, `META_MASK`, `SHIFT_MASK`
- **WindowEvent**
 - `getWindow()`

69

Information Contained in Events (II)

- **TextEvent**
- **ItemEvent**
 - `getItem()` → `Object` (`String` or `Integer`), `getItemSelectable()`
 - `getStateChange()` → `SELECTED`, `DESELECTED`
- **ContainerEvent**
 - `getChild()`, `getContainer()`
- **ComponentEvent**
 - `getComponent()`
- **AdjustmentEvent**
 - `getValue()`, `getAdjustable()`
 - `getAdjustmentType()` → `UNIT_INCREMENT`, `UNIT_DECREMENT`, `BLOCK_INCREMENT`, `BLOCK_DECREMENT`, `TRACK`
- **FocusEvent**

70

Adapters

- Spare empty implementation of the listener methods that are not needed
- Adapters are predefined implementations of listeners with an empty definition of all methods
 - Java provides the adapters in `java.awt.event` (an adapter class for each listener with more than one method)
 - The programmer can create adapter subclasses, defining only the methods needed
- A subclass of `Component` may not extend an adapter (or any other class)
⇒ Use an auxiliary adapter object to act as event listener
- Often the listener methods need to access variables of the component
 - Include in the adapter a variable that points to the component
 - Or define the adapter as an *internal class* of the component
- Self-sufficient adapters can be defined in a highly reusable way

71

```
class MouseAdapter implements MouseListener {
    public void mouseClicked (MouseEvent e) {}
    public void mousePressed (MouseEvent e) {}
    public void mouseReleased (MouseEvent e) {}
    public void mouseEntered (MouseEvent e) {}
    public void mouseExited (MouseEvent e) {}
}

class MyListener extends MouseAdapter {
    public void mouseEntered (MouseEvent e) {
        ... // Response to the event
    }
}

class MyWin extends Frame {
    MyWin () {
        Button b = new Button ();
        b.addMouseListener (new MyListener ());
        ...
    }
}
```

72

Adapter as Internal Class

```
class MyWin extends Frame {
    private String name;
    MyWin () {
        Button b = new Button ();
        b.addMouseListener (new MyListener ());
        ...
    }
    class MyListener extends MouseAdapter {
        public void mouseEntered (MouseEvent e) {
            ...
            System.out.println (name);
            ...
        }
    }
}
```

73

Adapter as Anonymous Class

```
class MyWin extends Frame {
    private String name;
    MyWin () {
        Button b = new Button ();
        b.addMouseListener (
            new MouseAdapter () {
                public void mouseEntered (MouseEvent e) {
                    ...
                    System.out.println (name);
                    ...
                }
            }
        );
    }
}
```

74

Reusable Adapters

```
class Mover extends MouseAdapter
    implements MouseMotionListener {
    private Point initialPoint;
    public Mover (Component c) { listenTo (c); }
    public void listenTo (Component c) {
        c.addMouseListener (this);
        c.addMouseMotionListener (this);
    }
    public void mousePressed (MouseEvent e) {
        initialPoint = e.getPoint ();
    }
    public void mouseDragged (MouseEvent e) {
        Component c = (Component) e.getSource ();
        Point p = c.getLocation ();
        c.setLocation (p.x + e.getX () - initialPoint.x,
            p.y + e.getY () - initialPoint.y);
    }
    public void mouseMoved (MouseEvent e) {}
}
```

75

AWT Widgets

Predefined Classes for Standard GUI Components

java.awt.Button



State and properties

- Constructors: Button(), Button(String)
The string is used as button label
- Set and get label: GetLabel(), SetLabel(String)
- Enable and disable button: setEnabled(boolean)

Operation

- Fires an ActionEvent when clicked
- Labelling action event: setActionCommand(String)
 - Associates a string to the button (by default, the same as the button label)
 - The string will be part of the information included in the ActionEvents fired by the button (see getActionCommand() of ActionEvent)

77

```
class ButtonDemo extends Frame implements ActionListener {
    Button b1, b2, b3;
    ButtonDemo () {
        setLayout (new FlowLayout ());
        b1 = new Button ();
        b1.setLabel ("Disable middle button");
        b1.setActionCommand ("Disable");
        b2 = new Button ("Middle button");
        b3 = new Button ("Enable middle button");
        b3.setEnabled (false);
        b3.setActionCommand ("Enable");

        add (b1);
        add (b2);
        add (b3);

        b1.addActionListener (this);
        b3.addActionListener (this);
    }
    ...
}
```

78

```

...
public void actionPerformed (ActionEvent e) {
    String command = e.getActionCommand();
    if (command.equals ("Disable")) {
        b1.setEnabled (false);
        b2.setEnabled (false);
        b3.setEnabled (true);
    }
    else {
        b1.setEnabled (true);
        b2.setEnabled (true);
        b3.setEnabled (false);
    }
}
}

```

79

Selection Widgets

Checkbox

 Checkbox

List

List item 1
List item 2
List item 3
List item 4
List item 5
List item 6

Choice

Menu

Menu 1 | Menu 2
Menu item 1_1
Menu item 1_2
Menu item 1_3
Menu item 1_4

ItemEvent carries item number (from List to Checkbox)
Fire ActionEvents (from Choice and Menu to List)

Double click (from List to Choice)
Multiple selection (from List to Checkbox)
Space saving (from Menu to Choice)

Fire ItemEvents (from List to Checkbox)

80

java.awt.Checkbox

Checkbox

Description

- Selectable button with two states: selected / deselected
- Can be grouped in single-selection radio button groups

State and properties

- Checkbox(), Checkbox(String), Checkbox(String, boolean)
- getLabel(), setLabel(String)
- getState(), setState(boolean)

81

java.awt.Checkbox (cont.)

Operation

- Fires an ItemEvent when its state changes
The item associated to the event is the checkbox label
- Identification of the item from the event:
 - getItem() → String (the checkbox label)
 - getItemSelectable() → Object (the checkbox component)
- Excluding buttons: groups of Checkboxes
 - Create an object of class CheckboxGroup
 - Use the constructor Checkbox(String, boolean, CheckboxGroup)
 - The checkboxes created with the same group as argument form a group
 - Only one checkbox in a group can be selected at any time

82

```

class CheckboxDemo extends Frame {
    CheckboxDemo () {
        setLayout (new GridLayout (1, 2));

        Panel p1 = new Panel ();
        Checkbox cb = new Checkbox ();
        cb.setLabel ("Checkbox 1");
        cb.setState (true);
        p1.add (cb);
        p1.add (new Checkbox ("Checkbox 2"));
        p1.add (new Checkbox ("Checkbox 3"));
        add (p1);

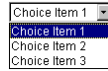
        Panel p2 = new Panel ();
        CheckboxGroup group = new CheckboxGroup ();
        p2.add (new Checkbox ("Checkbox 4", group, false));
        p2.add (new Checkbox ("Checkbox 5", group, false));
        p2.add (new Checkbox ("Checkbox 6", group, false));
        add (p2);
    }
}

```

83

84

java.awt.Choice



Description

- Pull-down menu, a.k.a. "combo box"
- One item in the list is selected

State and properties

- `add(String)`, `getItem(int)`, `getItemCount()`
- `select(int)`, `select(String)`, `isIndexSelected(int)`, `getSelectedItem() → String`, `getSelectedIndex() → int`

Operation

- Fires a `ItemEvent` when selection changes
The item associated to the event is the **string** of the selected item

85

java.awt.List



Description

- Selection list with scroll bar
- The selection mode can be simple or multiple

State and properties

- `List()`, `List(int)`, `List(int,boolean)`
- `add(String)`, `add(String,int)`, `remove(String)`, `remove(int)`, `getItem(int)`, `getItems()`, `getItemCount()`, `getRows()`
- `select(int)`, `select(String)`, `isIndexSelected(int)`, `getSelectedItem() → String`, `getSelectedIndex() → int`, `getSelectedObjects() → Object[]`
- `isMultipleMode()`, `setMultipleMode(boolean)`

86

java.awt.List (cont.)

Operation

- Select / deselect item: fires `ItemEvent`
The item associated to the event is the **position** of the selected item (an int)
- Double click: fires an `ActionEvent` with the text of the selected item as action command string

87

```
class ListDemo extends Frame
    implements ActionListener, ItemListener {
    TextArea output;
    List spanish, italian;
    ListDemo () {
        setLayout (new FlowLayout ());
        spanish = new List (3, true);
        spanish.add ("uno");
        spanish.add ("dos");
        spanish.add ("tres");
        spanish.add ("cuatro");
        spanish.addActionListener (this);
        spanish.addItemListener (this);

        italian = new List ();
        italian.add ("uno");
        italian.add ("due");
        italian.add ("tre");
        italian.add ("quattro");
        italian.addActionListener (this);
        italian.addItemListener (this);
        ...
    }
}
```

88

```
...
Panel p = new Panel ();
p.setLayout (new GridLayout (2, 1, 10, 10));
p.add (spanish);
p.add (italian);

output = new TextArea (10, 40);
output.setEditable (false);

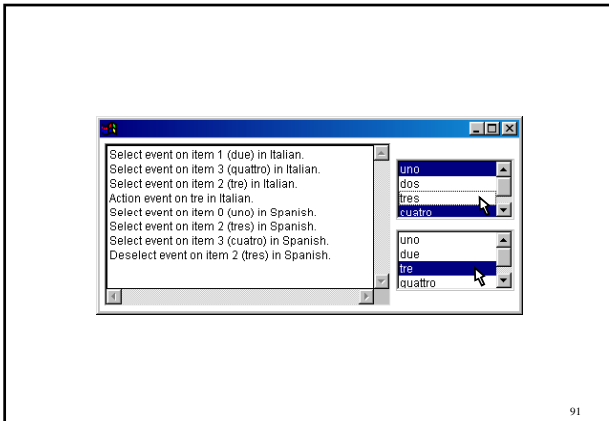
add (output);
add (p);
} // Fin del constructor
...
```

89

```
...
public void actionPerformed (ActionEvent e) {
    List list = (List) (e.getSource ());
    String language = (list == spanish)? "Spanish" : "Italian";
    output.append ("Action event on " + list.getSelectedItem ()
        + " in " + language + ".\n");
}

public void itemStateChanged (ItemEvent e) {
    List list = (List) (e.getItemSelectable ());
    String language = (list == spanish)? "Spanish" : "Italian";
    int index = ((Integer) (e.getItem ())) .intValue ();
    if (e.getStateChange () == ItemEvent.SELECTED) {
        output.append ("Select event on item " + index
            + " (" + list.getItem (index) + ") "
            + " in " + language + ".\n");
    }
    else output.append ("Deselect event on item " + index
        + " (" + list.getItem (index) + ") "
        + " in " + language + ".\n");
}
}
```

90



91

java.awt.Menu

Menu 1	Menu 2
Menu item 1_1	
Menu item 1_2	
Menu item 1_3	
Menu item 1_4	

Description

- Set of different modalities of items
 - MenuItem: button-like
 - CheckboxMenuItem: checkbox-like
 - Menu: submenu

```

classDiagram
    class MenuItem
    class Menu
    class CheckboxMenuItem
    MenuItem <|-- Menu
    MenuItem <|-- CheckboxMenuItem
            
```

Construction

- Menu(), Menu(String), MenuItem(), MenuItem(String), CheckboxMenuItem(), CheckboxMenuItem(String)
- Adding items: add(String|MenuItem|CheckboxMenuItem|Menu), insert(String|MenuItem|... ,int), getItem(int), addSeparator()
- A menubar is required to add menus to a window
 - setMenuBar(MenuBar) of Frame
 - add(Menu), getMenu(int), getMenuCount() of MenuBar

92

java.awt.Menu (cont.)

State and properties

- getLabel(), setLabel(String) of Menu / MenuItem
- Enable / disable item: isEnabled(), setEnabled(boolean) of Menu / MenuItem
- Selection of CheckboxMenuItem: getState(), setState(boolean)

Operation

- MenuItem only throws ActionEvent like a button, with the label of the item clicked upon as action command string
Event source: the item or the Menu that contains the item
- CheckboxMenuItem only fires ItemEvent like a checkbox, with the label of the selected / deselected item as event label
Event source: the CheckboxMenuItem itself

93

Text Widgets

TextComponent

Label

Texto estático

TextField

Campo de texto

TextArea

Area de texto

94

java.awt.Label

Label

Description

- Static text

State and properties

- Label(), Label(String), Label(String, LEFT|RIGHT|CENTER)
- getText(), setText(String)
- getAlignment(), setAlignment(LEFT|RIGHT|CENTER)

Operation

- None

95

java.awt.TextComponent

Description

- Superclass of TextField and TextArea
- Editable, selectable text

State and properties

- getText(), setText(String)
- isEditable(), setEditable(boolean)
- getCaretPosition(), setCaretPosition(int)
- getSelectedText(), select(int, int), selectAll(), getSelectionStart(), getSelectionEnd(), setSelectionStart(int), setSelectionEnd(int)

96

java.awt.TextField

Campo de texto

Description

- Single-line editable text

State and properties (in addition to those of TextComponent)

- TextField(), TextField(String), TextField(String, int)
- getColumns(), setColumns(int)
- getEchoChar(), setEchoChar(char), echoCharIsSet()

Operation

- Fires a TextEvent whenever a character is changed in the text
May want to process KeyEventS if willing to get each typed character
- When pressing 'Enter' fires an ActionEvent with the widget text as action command string

97

java.awt.TextArea

Area de texto

Description

- Multiline editable text with scrolling

State and properties (in addition to those of TextComponent)

- TextArea(), TextArea(String), TextArea(String, int, int)
- Scrollbars: TextArea(String, int, int, int),
TextArea.SCROLLBARS_NONE, SCROLLBARS_VERTICAL_ONLY...
- getColumns(), setColumns(int), getRows(), setRows(int)
- append(String), insert(String, int),
replaceRange(String, int, int)

Operation

- Fires a TextEvent whenever a character is changed in the text
- Does not fire ActionEvents

98

```
class TextDemo extends Frame implements ActionListener {
    TextField textField;
    TextArea textArea;
    TextDemo () {
        setLayout (new FlowLayout ());
        textField = new TextField (20);
        textArea = new TextArea (5, 20);
        textArea.setEditable (false);
        add (textField);
        add (textArea);
        textField.addActionListener (this);
    }
    public void actionPerformed (ActionEvent evt) {
        String text = textField.getText ();
        textArea.append (text + "\n");
        textField.selectAll ();
    }
}
```

99

100

java.awt.Dialog

Description

- Depends on another window:
 - Is destroyed when the main window is destroyed
 - Disappears when the main window is minimized

State and properties

- Dialog (Frame [,String] [,boolean])
- Modal / non-modal: isModal(), setModal(boolean)
- isResizable(), setResizable(boolean) Of Window
- FileDialog extends Dialog
 - getDirectory(), setDirectory(String)
 - getFile(), setFile(String)

101

```
class DialogWindow extends Frame implements ActionListener {
    private SimpleDialog dialog;
    private TextArea textArea;
    public DialogWindow () {
        textArea = new TextArea (5, 40);
        textArea.setEditable (false);
        add ("Center", textArea);
        Button button = new Button ("Click to bring up dialog");
        button.addActionListener (this);
        Panel panel = new Panel ();
        panel.add (button);
        add ("South", panel);
    }
    public void actionPerformed (ActionEvent event) {
        if (dialog == null)
            dialog = new SimpleDialog (this, "A Simple Dialog");
        dialog.setVisible (true);
    }
    public void addLine (String text) {
        textArea.append(text + "\n");
    }
}
```

102

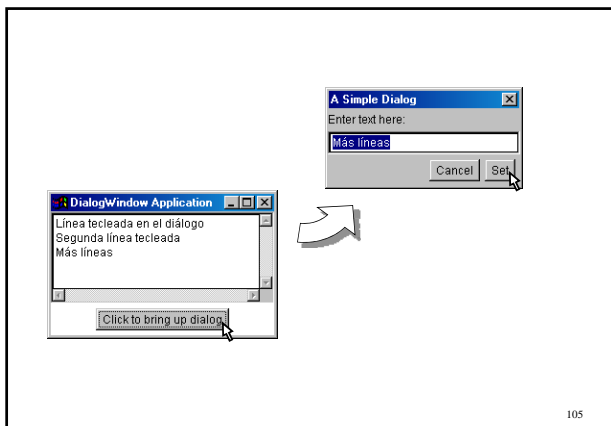
```
class SimpleDialog extends Dialog implements ActionListener {
    TextField field;
    DialogWindow parent;
    Button setButton;
    SimpleDialog (Frame w, String title) {
        super (w, title, false);
        parent = (DialogWindow) w;
        Panel p1 = new Panel ();
        p1.setLayout (new GridLayout (2, 1));
        Label label = new Label ("Enter text here:");
        p1.add (label);
        field = new TextField (40);
        field.addActionListener (this);
        p1.add (field);
        add ("Center", p1);
        ...
    }
}
```

103

```
...
Panel p2 = new Panel();
p2.setLayout (new FlowLayout (FlowLayout.RIGHT));
Button b = new Button ("Cancel");
b.addActionListener (this);
setButton = new Button ("Set");
setButton.addActionListener (this);
p2.add (b);
p2.add (setButton);
add ("South", p2);
pack ();
}

public void actionPerformed (ActionEvent event) {
    Object source = event.getSource();
    if ((source == setButton) || (source == field))
        parent.addLine (field.getText());
    field.selectAll ();
    setVisible (false);
}
}
```

104



105

Custom Widgets

```
class MyButton extends Canvas implements MouseListener {
    private boolean selected = false;
    protected String label;
    public MyButton (String str) {
        label = str;
        addMouseListener (this);
        new Mover (this);
    }
    public void paint (Graphics g) {
        Dimension dim = getSize ();
        FontMetrics metrics = g.getFontMetrics ();
        g.drawRect (0, 0, dim.width-1, dim.height-1);
        g.drawString (label,
            (dim.width - metrics.stringWidth (label)) / 2,
            (dim.height - metrics.getHeight ()) / 2
            + metrics.getMaxAscent ());
    }
    ...
}
```

Define
aspecto visual

106

Methods to negotiate size with layout managers

```
...
public Dimension getPreferredSize () {
    FontMetrics metrics = getGraphics ().getFontMetrics ();
    return new Dimension (metrics.stringWidth (label) + 20,
        metrics.getHeight () + 10);
}

public Dimension getMinimumSize () {
    Dimension dim = getPreferredSize ();
    return new Dimension (dim.width / 2, dim.height / 2);
}

public Dimension getMaximumSize () {
    Dimension dim = getPreferredSize ();
    return new Dimension (dim.width * 2, dim.height * 2);
}
...
```

107

```
...
public void mouseClicked (MouseEvent e) {
    selected = !selected;
    if (selected) {
        setBackground (Color.black);
        setForeground (Color.white);
    }
    else {
        setBackground (Color.white);
        setForeground (Color.black);
    }
}

public void mousePressed (MouseEvent e) {}
public void mouseReleased (MouseEvent e) {}
public void mouseEntered (MouseEvent e) {}
public void mouseExited (MouseEvent e) {}
...
```

Visual feedback
to user action

108

```

...
public String getLabel () { return label; }
public void setLabel (String str) {
    label = str;
    repaint ();
}
public boolean getState () { return selected; }
public void setState (boolean state) {
    selected = state;
    if (selected) {
        setBackground (Color.black);
        setForeground (Color.white);
    }
    else {
        setBackground (Color.white);
        setForeground (Color.black);
    }
}
// public void mouseClicked (MouseEvent e) {
//     setState (!selected);
// }
}
    
```

Interface to control widget state

109

```

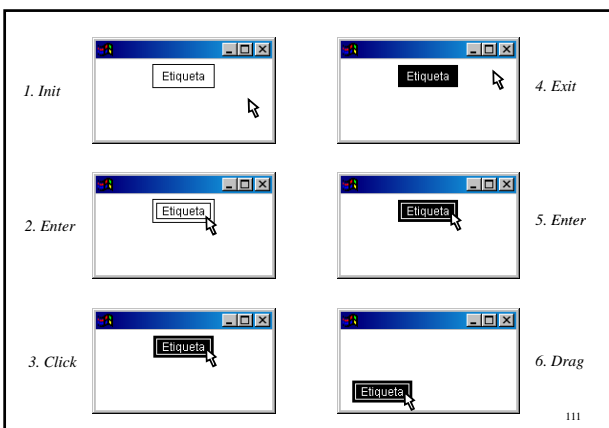
class MyButton2 extends MyButton {
    boolean pointedAt = false;
    public MyButton2 (String str) { super (str); }

    public void paint (Graphics g) {
        super.paint (g);
        Dimension dim = getSize ();
        if (pointedAt)
            g.drawRect (4, 4, dim.width-9, dim.height-9);
    }

    public void mouseEntered (MouseEvent e) {
        pointedAt = true;
        repaint ();
    }
    public void mouseExited (MouseEvent e) {
        pointedAt = false;
        repaint ();
    }
}
    
```

Further feedback to user actions

110



```

class ButtonAction extends MyButton {
    private ActionListener multicaster = null;
    public ButtonAction (String str) { super (str); }
    public void mouseClicked (MouseEvent e) {
        super.mouseClicked (e);
        processEvent ( ← Generation of events in response to interaction
            new ActionEvent (
                this, ActionEvent.ACTION_PERFORMED, label));
    }
    public void addActionListener (ActionListener listener) {
        multicaster =
            AWTEventMulticaster.add (multicaster, listener);
    }
    public void processEvent (AWTEvent e) {
        if (e.getID () == ActionEvent.ACTION_PERFORMED
            && multicaster != null)
            multicaster.actionPerformed ((ActionEvent) e);
        super.processEvent (e);
    }
}
    
```

112

Internal Event System

From the Window System
to the Listener Methods

Event IDs

Correspond to event types in the underlying window system

- Event class vs. event type
 - Event type: concrete type represented by an ID number
To get the event ID: `getID()` de `AWTEvent`
 - Event class: event categories, each comprises a set of IDs
The classes contain constants that define the the IDs they comprise
e.g. `MouseEvent.MOUSE_DRAGGED`, `ActionEvent.ACTION_PERFORMED`
- Each method of a listener corresponds to an event ID
 - Therefore, using listeners, the programmer does not need to use the IDs
 - IDs are used by the AWT system to determine what methods to call on the listeners
- IDs are used to create and fire events from the program
 - E.g. to define new widgets
 - The ID is passed as argument to the constructor of the different event classes

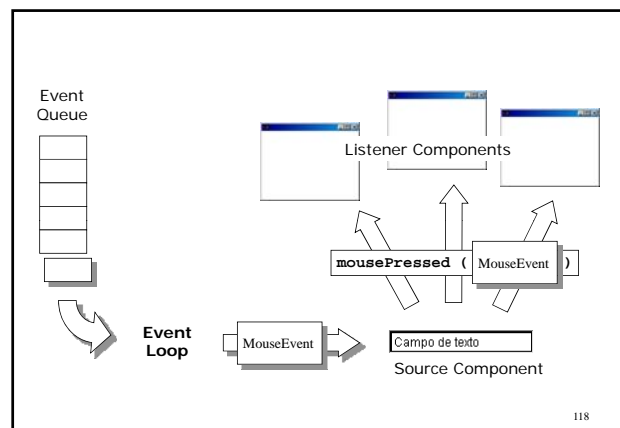
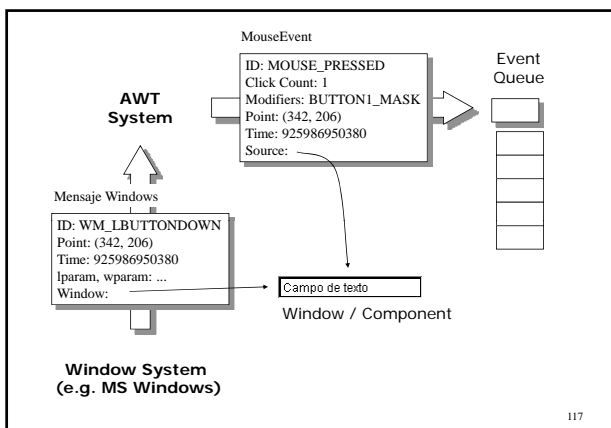
114

Event ID ↔ Listener Method

Event class	Event ID	Listener method	Listener
MouseEvent	MOUSE_CLICKED	mouseClicked(MouseEvent)	MouseListener
	MOUSE_PRESSED	mousePressed(MouseEvent)	
	MOUSE_RELEASED	mouseReleased(MouseEvent)	
	MOUSE_ENTERED	mouseEntered(MouseEvent)	
	MOUSE_EXITED	mouseExited(MouseEvent)	
	MOUSE_MOVED	mouseMoved(MouseEvent)	
KeyEvent	KEY_TYPED	keyTyped(KeyEvent)	KeyListener
	KEY_PRESSED	keyPressed(KeyEvent)	
	KEY_RELEASED	keyReleased(KeyEvent)	

115

- ### The Event Queue and the Event Loop
- **Initialisation**
 - The AWT system creates an event queue (internal)
 - The system starts a thread with a loop that pops and processes events from the queue (internal)
 - **Entry of events**
 - The AWT system captures the messages sent by the window system, creates AWT events and pushes them into the event queue (internal)
 - The queue insertion is optimised to remove redundancies in consecutive events of type `MOUSE_MOVE`, `MOUSE_DRAG`, `PAINT`, and `UPDATE`
 - **Event processing: the event loop**
 - Invokes `dispatchEvent` of the popped event onto its source component
- 116



```

public class EventQueue {
    private static int threadInitNumber;
    private static synchronized int nextThreadNum() {
        return threadInitNumber++;
    }
    private EventQueueItem queue;
    public EventQueue() {
        queue = null;
        String name = "AWT-EventQueue-" + nextThreadNum();
        new EventDispatchThread(name, this).start();
    }
    public synchronized void postEvent(AWTEvent theEvent) {
        ...
    }
}

class EventQueueItem {
    AWTEvent event; int id;
    EventQueueItem next;
    EventQueueItem(AWTEvent evt) {
        event = evt; id = evt.getID();
    }
}
    
```

119

```

// Entry of events in the queue
public synchronized void postEvent(AWTEvent theEvent) {
    EventQueueItem eqi = new EventQueueItem(theEvent);
    if (queue == null) { queue = eqi; notifyAll(); }
    else {
        EventQueueItem q = queue;
        for (; q.next != null; q = q.next) {
            if (q.id == eqi.id) {
                switch (q.id) {
                    case Event.MOUSE_MOVE:
                    case Event.MOUSE_DRAG:
                        MouseEvent e = (MouseEvent)q.event;
                        if (e.getSource() ==
                            ((MouseEvent)theEvent).getSource()
                            && e.getModifiers() ==
                            ((MouseEvent)theEvent).getModifiers()) {
                            q.event = eqi.event;
                            return;
                        }
                        break;
                }
            }
        }
    }
}
    
```

120

```

EventQueue.java
...
case PaintEvent.PAINT: } ← Component-refresh
case PaintEvent.UPDATE: } ← event type
    PaintEvent pe = (PaintEvent)q.event;
    if (pe.getSource() == theEvent.getSource()) {
        Rectangle rect = pe.getUpdateRect();
        Rectangle newRect =
            ((PaintEvent) theEvent).getUpdateRect ();
        if (!rect.equals (newRect))
            pe.setUpdateRect (rect.union (newRect));
        return; ← Discard previous event
    }
    break;
} // End switch
}
q.next = eq1; ← Other event types:
                push into queue
} // End postEvent
    
```

Create a refresh event for the union of both regions

```

EventDispatchThread.java
// Event loop
public void run () {
    while (doDispatch) {
        try {
            AWTEvent event = theQueue.getNextEvent ();
            Object src = event.getSource ();
            if (src instanceof Component)
                ((Component) src).dispatchEvent (event);
            else if (src instanceof MenuComponent)
                ((MenuComponent) src).dispatchEvent( event);
        } catch (Exception e) {
            System.err.println (
                "Exception occurred during event handling:");
            e.printStackTrace ();
        }
    }
}
    
```

```

EventDispatchThread.java
// The whole class
class EventDispatchThread extends Thread {
    private EventQueue theQueue;
    private boolean doDispatch = true;

    EventDispatchThread (String name, EventQueue queue) {
        super (name);
        theQueue = queue;
    }

    public void stopDispatching () {
        doDispatch = false;
    }

    public void run () {
        ... // See previous page
    }
}
    
```

```

Component.java
void dispatchEventImpl(AWTEvent e) {
    int id = e.getID();
    // 1. Pre-process any special events before delivery
    switch(id) {
        case PaintEvent.PAINT:
        case PaintEvent.UPDATE:
            Graphics g = getGraphics();
            if (g == null) return;
            Rectangle r = ((PaintEvent)e).getUpdateRect();
            g.clipRect(r.x, r.y, r.width, r.height);
            if (id == PaintEvent.PAINT) paint(g);
            else update(g);
            g.dispose();
            return;
        case FocusEvent.FOCUS_GAINED:
            if (parent != null && !(this instanceof Window))
                parent.setFocusOwner(this);
            break;
    }
    ...
}
    
```

```

Component.java
...
// 2. Deliver event for normal processing
if (newEventsOnly) {
    if (eventEnabled(e)) processEvent(e);
}
else if (!(e instanceof MouseEvent
    && !postsOldMouseEvents())) {
    // backward compatibility...
}

// 3. Propagate key events to parent
if (!e.isConsumed()
    && e instanceof java.awt.event.KeyEvent) {
    Container p = parent;
    if (p != null) p.postProcessKeyEvent((KeyEvent)e);
}

// 4. Allow the peer to process the event
if (peer != null) peer.handleEvent(e);
}
    
```

```

Component.java
protected void processEvent(AWTEvent e) {
    if (e instanceof MouseEvent) {
        switch(e.getID()) {
            case MouseEvent.MOUSE_PRESSED:
            case MouseEvent.MOUSE_RELEASED:
            case MouseEvent.MOUSE_CLICKED:
            case MouseEvent.MOUSE_ENTERED:
            case MouseEvent.MOUSE_EXITED:
                processMouseEvent((MouseEvent)e); break;
            case MouseEvent.MOUSE_MOVED:
            case MouseEvent.MOUSE_DRAGGED:
                processMouseEvent((MouseEvent)e); break;
        }
    }
    else if (e instanceof KeyEvent)
        processKeyEvent((KeyEvent)e);
    else if (e instanceof ComponentEvent)
        processComponentEvent((ComponentEvent)e);
    else if (e instanceof FocusEvent)
        processFocusEvent((FocusEvent)e);
}
    
```

```
Component.java

protected void processKeyEvent(KeyEvent e) {
    if (keyListener != null) {
        int id = e.getID();
        switch(id) {
            case KeyEvent.KEY_TYPED:
                keyListener.keyTyped(e); break;
            case KeyEvent.KEY_PRESSED:
                keyListener.keyPressed(e); break;
            case KeyEvent.KEY_RELEASED:
                keyListener.keyReleased(e); break;
        }
    }
}
127
```

```
Component.java

protected void processComponentEvent(ComponentEvent e) {
    if (componentListener != null) {
        int id = e.getID();
        switch(id) {
            case ComponentEvent.COMPONENT_RESIZED:
                componentListener.componentResized(e); break;
            case ComponentEvent.COMPONENT_MOVED:
                componentListener.componentMoved(e); break;
            case ComponentEvent.COMPONENT_SHOWN:
                componentListener.componentShown(e); break;
            case ComponentEvent.COMPONENT_HIDDEN:
                componentListener.componentHidden(e); break;
        }
    }
}
128
```

```
Component.java

protected void processFocusEvent(FocusEvent e) {
    if (focusListener != null) {
        int id = e.getID();
        switch(id) {
            case FocusEvent.FOCUS_GAINED:
                focusListener.focusGained(e); break;
            case FocusEvent.FOCUS_LOST:
                focusListener.focusLost(e); break;
        }
    }
}
129
```

```
Component.java

protected void processMouseEvent(MouseEvent e) {
    if (mouseListener != null) {
        int id = e.getID();
        switch(id) {
            case MouseEvent.MOUSE_PRESSED:
                mouseListener.mousePressed(e); break;
            case MouseEvent.MOUSE_RELEASED:
                mouseListener.mouseReleased(e); break;
            case MouseEvent.MOUSE_CLICKED:
                mouseListener.mouseClicked(e); break;
            case MouseEvent.MOUSE_EXITED:
                mouseListener.mouseExited(e); break;
            case MouseEvent.MOUSE_ENTERED:
                mouseListener.mouseEntered(e); break;
        }
    }
}
130
```

```
Component.java

protected void processMouseMotionEvent(MouseEvent e) {
    if (mouseMotionListener != null) {
        int id = e.getID();
        switch(id) {
            case MouseEvent.MOUSE_MOVED:
                mouseMotionListener.mouseMoved(e); break;
            case MouseEvent.MOUSE_DRAGGED:
                mouseMotionListener.mouseDragged(e); break;
        }
    }
}
131
```

```
List.java

protected void processEvent(AWTEvent e) {
    if (e instanceof ItemEvent) {
        processItemEvent((ItemEvent)e);
        return;
    } else if (e instanceof ActionEvent) {
        processActionEvent((ActionEvent)e);
        return;
    }
    super.processEvent(e);
}

protected void processItemEvent(ItemEvent e) {
    if (itemListener != null)
        itemListener.itemStateChanged(e);
}

protected void processActionEvent(ActionEvent e) {
    if (actionListener != null)
        actionListener.actionPerformed(e);
}
132
```

Advanced Layout

CardLayout and GridBagLayout

java.awt.CardLayout

- The components take the same space, the whole container, as a deck of cards
- Only one is visible an any time
 - Move up / move down:
 - next (Container)
 - previous (Container)
 - Show first, show last
 - first (Container)
 - last (Container)
 - Show by name components that were added with name
 - public void show (Container, String)
 - add (String, Component) de Component

134

```
class CardDemo extends Frame
    implements ActionListener, ItemListener {
    Choice choice;
    CardLayout panelLayout = new CardLayout ();
    Panel cardPanel;
    CardDemo () {
        cardPanel = new Panel ();
        cardPanel.setLayout (panelLayout);
        cardPanel.add ("One",
            new Label ("Component 1", Label.CENTER));
        cardPanel.add ("Two",
            new Label ("Component 2", Label.CENTER));
        cardPanel.add ("Three",
            new Label ("Component 3", Label.CENTER));
        cardPanel.add ("Four",
            new Label ("Component 4", Label.CENTER));
        add ("Center", cardPanel);
        ...
    }
}
```

135

```
...
Button first = new Button ("First");
Button last = new Button ("Last");
Button previous = new Button ("Previous");
Button next = new Button ("Next");

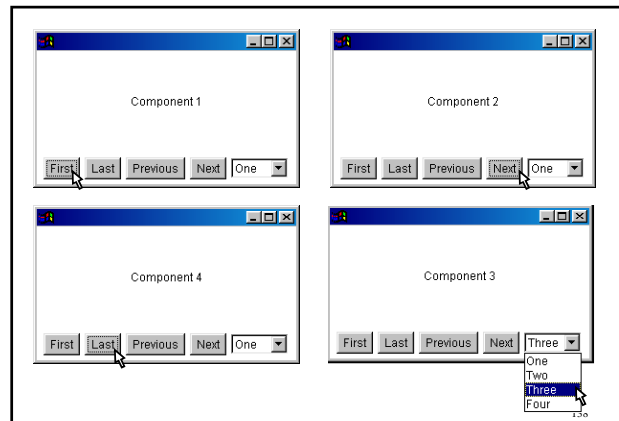
choice = new Choice ();
choice.add ("One");
choice.add ("Two");
choice.add ("Three");
choice.add ("Four");

Panel p = new Panel ();
p.add (first);
p.add (last);
p.add (previous);
p.add (next);
p.add (choice);
add ("South", p);
...
} // Fin del constructor
...
first.addActionListener (this);
last.addActionListener (this);
previous.addActionListener (this);
next.addActionListener (this);
choice.addItemListener (this);
...
}
```

136

```
...
public void actionPerformed (ActionEvent e) {
    String command = e.getActionCommand ();
    if (command.equals ("First"))
        panelLayout.first (cardPanel);
    else if (command.equals ("Last"))
        panelLayout.last (cardPanel);
    else if (command.equals ("Previous"))
        panelLayout.previous (cardPanel);
    else if (command.equals ("Next"))
        panelLayout.next (cardPanel);
}
public void itemStateChanged (ItemEvent e) {
    String item = (String) e.getItem ();
    panelLayout.show (cardPanel, item);
}
} // Fin de CardDemo
```

137



138

java.awt.GridBagLayout

- Similar to `GridLayout`
 - Components placed in a grid
 - Rows and columns width is proportional to container size
- Much more flexible and complex than `GridLayout`
 - Rows and columns may have different widths
 - Components may take more than one grid cell
- The layout (position and size) of each component is set with an object of type `GridBagConstraints`
 - `setConstraints (Component, GridBagConstraints)` of `GridBagLayout`
 - The layout of the object to which the constraint is associated is defined through the variables of the `GridBagConstraints` object

139

Variables of java.awt.GridBagConstraints

- **gridx, gridy:** column and row position of the component
 - After the previous component: `GridBagConstraints.RELATIVE` (default value)
- **gridwidth, gridheight:** number of rows and columns taken by the component
 - Default value: 1
 - Last of row / column: `GridBagConstraints.REMAINDER`
 - Before last of row / column: `GridBagConstraints.RELATIVE`
- **fill:** component flexibility / stiffness
 - Rigid size: `GridBagConstraints.NONE` (default value)
 - Flexible size: `GridBagConstraints.VERTICAL, HORIZONTAL, BOTH`
- **weightx, weighty:** relative size of rows and columns
 - Default value: 0.0 (rigid cells)
 - Values between 0.0 and 1.0

140

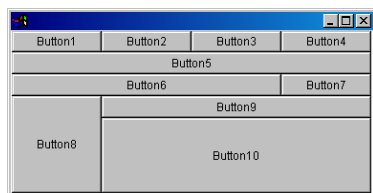
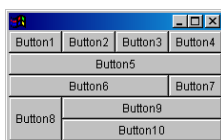
```
public class GridBagWindow extends Frame {
    protected void makebutton (String name,
                               GridBagConstraints gridbag,
                               GridBagConstraints c) {
        Button button = new Button (name);
        gridbag.setConstraints (button, c);
        add(button);
    }
    public GridBagWindow () {
        GridBagLayout gridbag = new GridBagLayout ();
        setLayout (gridbag);

        GridBagConstraints c = new GridBagConstraints ();
        c.fill = GridBagConstraints.BOTH;
        c.weightx = 1.0;
        makebutton ("Button1", gridbag, c);
        makebutton ("Button2", gridbag, c);
        makebutton ("Button3", gridbag, c);
        ...
    }
}
```

141

```
...
c.gridwidth = GridBagConstraints.REMAINDER;
makebutton ("Button4", gridbag, c);
makebutton ("Button5", gridbag, c);
c.gridwidth = GridBagConstraints.RELATIVE;
makebutton ("Button6", gridbag, c);
c.gridwidth = GridBagConstraints.REMAINDER;
makebutton ("Button7", gridbag, c);
c.gridwidth = 1; // valor por defecto
c.gridheight = 2;
c.weighty = 1.0;
makebutton ("Button8", gridbag, c);
c.weighty = 0.0; // valor por defecto
c.gridwidth = GridBagConstraints.REMAINDER;
c.gridheight = 1; // valor por defecto
makebutton ("Button9", gridbag, c);
makebutton ("Button10", gridbag, c);
}
}
```

142



143

```
class ListDemo2 extends Frame {
    TextArea output;
    List spanish, italian;
    ListDemo2 () {
        spanish = new List (4, true);
        spanish.add ("uno"); spanish.add ("dos");
        spanish.add ("tres"); spanish.add ("cuatro");
        spanish.add ("cinco"); spanish.add ("seis");

        italian = new List ();
        italian.add ("uno"); italian.add ("due");
        italian.add ("tre"); italian.add ("quattro");
        italian.add ("cinque"); italian.add ("sei");

        output = new TextArea (10, 40);
        output.setEditable (false);

        add (output);
        add (spanish);
        add (italian);
        ...
    }
}
```

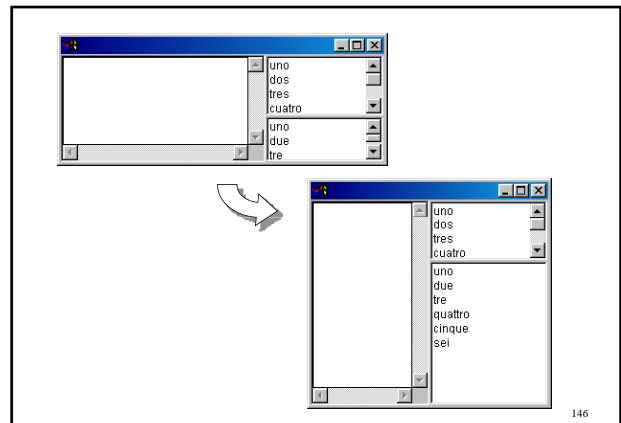
144

```
...
GridBagLayout gridBag = new GridBagLayout ();
setLayout (gridBag);

GridBagConstraints tc = new GridBagConstraints ();
tc.fill = GridBagConstraints.BOTH;
tc.weightx = 1.0;
tc.weighty = 1.0;
tc.gridheight = 2;
gridBag.setConstraints (output, tc);

GridBagConstraints lc = new GridBagConstraints ();
lc.fill = GridBagConstraints.VERTICAL;
lc.gridwidth = GridBagConstraints.REMAINDER;
gridBag.setConstraints (spanish, lc);
gridBag.setConstraints (italian, lc);
}
}
```

145



146