Java Graphics Programming

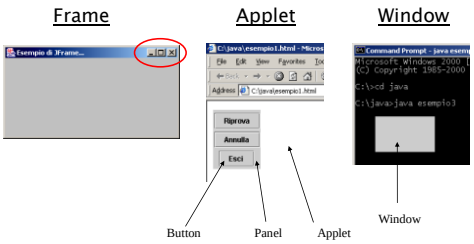
SOftEng

Concepts of graphical programming

- Set a Look & Feel (= Style)
  - Microsoft → *Windows* style
  - Macintosh → *Mac* style
  - Java → *Metal* style
- Define one (or more) principal container
  - Window, Frame, Applet
- add components to the containers
  - Button
  - RadioBox
  - Ecc.
  - Arrange the components/containers according to a layout

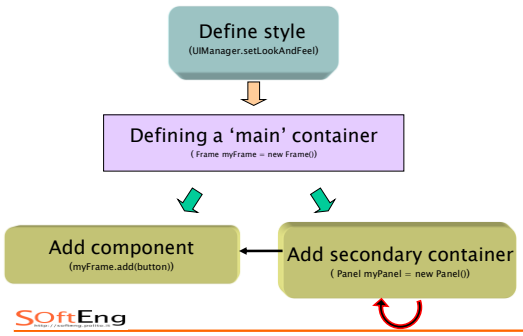
SOftEng

Difference Frame – Applet – Window



SOftEng

Procedure



SOftEng

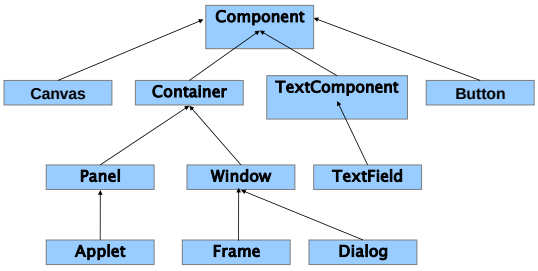
Package java.awt.*

Provides the following features :

- **Components**: button, checkbox, scrollbar, etc.
- Support for secondary "containers": they are still components!
- Management:
  - System events
  - events generated by the users on parts of IU
- Layout: the components are included in the platform

SOftEng

Sub-classes of Component in java.awt



SOftEng

Package javax.swing

- contains the same components of **java.awt**, but with the different name (**jbutton**, **jframe**, etc.)
- All these components derive from **JComponent**
- Advantages:
  - provides a series of components 'light' (light-weight) has the same appearance/behaviour on all platforms
  - Look and feel is changeable to flight
- swing it is an extension of AWT → however management of the events in the two package is different

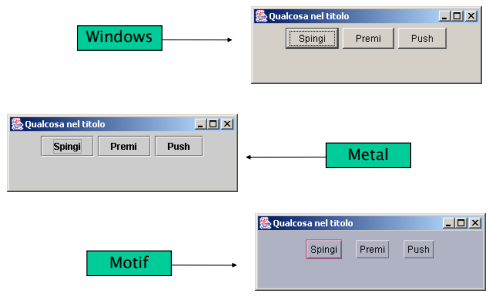
SOftEng

Define a style (step 1 of 4)

- The class of reference is UIManager, which belongs to the package java.lang
- Possibility:
  - UIManager.setLookAndFeel("com.sun.java.swing.plaf.windows.**WindowsLookAndFeel**");
  - UIManager.setLookAndFeel("com.sun.java.swing.plaf.motif .**MotifLookAndFeel**");
  - UIManager.setLookAndFeel("javax.swing.plaf.mac.**MacLoo kAndFeel**"); → only on MACINTOSH platforms
  - UIManager.setLookAndFeel("javax.swing.plaf.metal.**MetalL ookAndFeel**"); [DEFAULT]
- By default the class UIManager must always precede made by a block try.. catch

SOftEng

Example of Look & Feel



SOftEng

Set a container (Step 2 of 4)

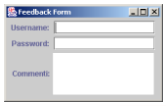
- Steps:
  - the class must extend the container chosen (Class my_container **extends** jframe)
  - It is necessary to create at first a secondary container and describe it as 'inside' (jpanel jpanel = new window (); (…) setcontentpane (window))
  - components must be added to Window: pressbutton window.add (pressbutton);

SOftEng

A complete example

```
import javax.swing.*;

public class MyForm extends JFrame{

private JTextField username = new
JTextField(15);
JPasswordField password = new
JPasswordField(15);
JTextArea textArea = new JTextArea(4,15);

public MyForm() {

        super("Feedback Form");
        setSize(360, 160);
   setDefaultCloseOperation(EXIT_ON_CLOSE);

        JPanel pane = new JPanel();
        JLabel usernameLabel = new
JLabel("Username: ");
        JLabel passwordLabel = new
JLabel("Password: ");
        JLabel commentLabel = new
JLabel("Comments: ");}
```

```
        pane.add(usernameLabel);
        pane.add(username);
        pane.add(passwordLabel);
        pane.add(password);
        pane.add(commentLabel);
        pane.add(commentLabel);
        pane.add(textArea);
        setContentPane(pane);

        setVisible(true);
}

public static void main(String[] arguments) {
        MyForm form1 = new MyForm();
        form1.setVisible(true);
  }
```



SOftEng

Basic Functions of containers

- Messages (MSG) in the closure of the Windows→ in the form "setDefaultCloseOperation(msg)"
  - EXIT_ON_CLOSE
  - DO_NOTHING_ON_CLOSE
  - DISPOSE_ON_CLOSE
  - HIDE_ON_CLOSE
- SetSize(int base, int altezza) → defines the dimensions of the panel outside
- setBounds (int xSupSin, int ySupSin, int base, int height) → it specifies the position in which it is initially the panel
- insert the secondary container in primary: primarycont.setcontentpane (secondarycont);

SOftEng

Insert component (step 3 of 4)

Component Swing – Button

- button is a component: initiates action with the pressure event.
- button is created by the manufacturers:
  - JButton(); creates a button without a text (without label)
  - JButton(String); creates a button with a label containing the text.
- it is a *component* → inherits all the methods of classes JComponent (javax.swing) and component (java.awt)
- It is a *container* → inherits all methods of java.awt.container.class

SOftEng

Component Swing – Label

- JLabel(); create label empty, aligned on the left
- JLabel(String); create label with given text, aligned on the left
- JLabel(String, int); create label with given text, aligned as specified in the second parameter (SwingConstants.LEFT, SwingConstants.RIGHT, SwingConstants.CENTER).
- Activated methods on a label : getText(), setText(String), getAlignement(), setAlignement(int).

SOftEng

Swing Component – Text fields

- The text fields allows the introduction of strings of text to be part of the user.
- To create a text field using the manufacturers :
  - JTextField();

  ✂ JTextField (String), JTextField (String, int), ...

Example

```
tf1 = new JTextField();
tf2 = new JTextField("", 20);
tf3 = new JTextField("Hello");
tf4 = new JTextField("Hello",30);
```

SOftEng

Swing Component – Text fields

- Need to manage more than a line of text for time .
- Builders:
  - JTextArea (int1, int2) → int1: nr. lines, int2: nr columns
  - JTextArea (String, int1, int2) → is a default text
- Useful Metods:
  - getText(), setText(String);
  - append(String), insert(String, int);
  - void setLineWrap(boolean) → if true, the lines will be wrapped if they are too long to fit the allocated space, if false lines will be unwrapped
  - void setWrapStyleWord(boolean) → it must be at the head with the whole word

SOftEng

Swing Component – Control/Option

- check boxes : JCheckBox(String, boolean)
- Option buttons: JRadioButton(String, boolean)
- Useful methods :
  - void setSelected(boolean) → with true, set the boolean component to 'on'
  - boolean isSelected() → return true if the component is turned on
- by default are non-exclusive: can be found more Lit in contemporary radiobutton
- to ensure the mutual exclusion : il RadioButton (or CheckBox) are added to a ButtonGroup

SOftEng

Example

```
public class Authors extends JFrame{

JRadioButton[] list = new JRadioButton[4];
public Authors() {
            super("Select an author");
        setSize(140, 190);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        list[0] = new JRadioButton("Jehoshua", true);
        list[1] = new JRadioButton("McEwan");
        list[2] = new JRadioButton("Tamaro");
        list[3] = new JRadioButton("Steel");
        JPanel panel = new JPanel();
        ButtonGroup group = new ButtonGroup();

        for (int i = 0; i < list.length; i++) {
            group.add(list[i]);
            panel.add(list[i]);
        }
        setContentPane(panel);
        setVisible(true);
}       public static void main (String args[]) {
        Authors newLista = new Authors();   }
}
```

SOftEng

Components Swing – Dialogue box

- They're used to receive input, provide information, advise the user, etc.
- Option:
  - Confirmation windows
  - Input and dialog boxes
  - Message and dialog windows
  - Dialog and option windows
- Methods more efficient than input/output in order to read from keyboard

  Options are a much more efficient method of input/output flow to read from the keyboard
- each window is managed by a different method of the same class (JOptionPane.class)

SOftEng

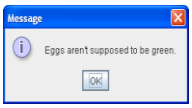
Dialogue windows for confirmation

Every dialog is dependent on a Frame component.

A swing JDialog class inherits this behavior from the AWT Dialog class.

Example:

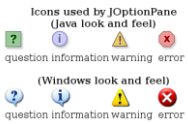
JOptionPane.showMessageDialog(frame, "Eggs are not supposed to be green.");



SOftEng

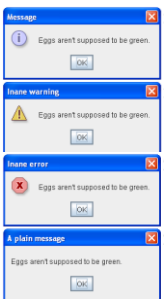
JOptionPane Features

Using JOptionPane, you can quickly create and customize several different kinds of dialogs. JOptionPane provides support for laying out standard dialogs, providing icons, specifying the dialog title and text, and customizing the button text.



SOftEng

JOptionPane Features: examples

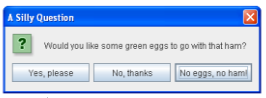



```
//default title and icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Message");

//custom title, warning icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane warning",
    JOptionPane.WARNING_MESSAGE);

//custom title, error icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "Inane error",
    JOptionPane.ERROR_MESSAGE);

//custom title, no icon
JOptionPane.showMessageDialog(frame,
    "Eggs are not supposed to be green.",
    "A plain message",
    JOptionPane.PLAIN_MESSAGE);
```

SOftEng

showOptionDialog

Displays a modal dialog with the specified buttons, icons, message, title, and so on. With this method, you can change the text that appears on the buttons of standard dialogs. You can also perform many other kinds of customization.




```
//Custom button text
Object[] options = {"Yes, please",
                    "No, thanks",
                    "No eggs, no ham!"};
int n = JOptionPane.showOptionDialog(frame,
    "Would you like some green eggs to go "
    + "with that ham?",
    "A Silly Question",
    JOptionPane.YES_NO_CANCEL_OPTION,
    JOptionPane.QUESTION_MESSAGE,
    null,
    options,
    options[2]);
```

SOftEng

Dialogue windows for input

- The reference method is showinputdialog
- Primitive: String showInputDialog(Component, Object)
- Alternative: String showInputDialog(Component, Object, String, int)
  - Component: in which component appears window
  - Object: Request message input
  - String: title
  - int: type of message (Same encoding of the window of confirmation)
- Input is immediate : String answer = JOptionPane.showInputDialog(null, "Your sweet preferred ?", "answers…", JOptionPane.QUESTION_MESSAGE)

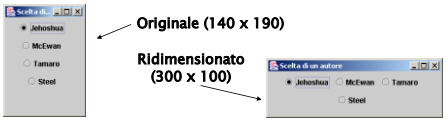
Exercise

- Create Java application able to take input from user information
  - Name of the internet site
  - URL address
  - General information {personal, business, educational}
- with them, create three couples Labels/lines of text (Label = "Name", text = inserted by User)
- try to change the size of main panel
- finally, before adding the components to the panel , use pannel.setLayout(new GridLayout(3,2)), and try again to modify the dimensions

LAYOUT

What is a layout?

- All the former examples graphs, when resized, allow the relocation of the components:



- this behavior is a necessity: Java adapts to many platforms (display in different way for different systems)
- Solution in Visual Basic → available 'absolut'(x,y)

Operators of layout in Java

- *Layout* → indicates where the components are located
- *Operators of layout* → Determining the method of disposal of the same components (import java.awt.*)
- A panel ↔ an operator of layout
- Therefore: different panels can have different operators
- Methodology:
  - Create a body by the class of the operator:
    FlowLayout f = new FlowLayout()
  - Create a panel, and first assign the operator :
    JPanel panel = new JPanel();
    panel.setLayout(f);
  - After, add the components to the panel:
    panel.add(JButton); (…)

Operators of layout – FlowLayout

- It is the base layout of applciations/graphics applets
- Disposition: from left to right, starting from the left most corner in the top
- Builder:
  - FlowLayout f = new FlowLayout();
  - FlowLayout f = new FlowLayout(int align);
  - FlowLayout f = new FlowLayout(int align, int hgap, int vgap);
- Builder elements:
  - align: Alignment of basis (FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER)
  - hgap: Horizontal space between components (default: 3 pixel)
  - vgap: Vertical space between components (default: 3 pixel)

Example of FlowLayout (default in Java)

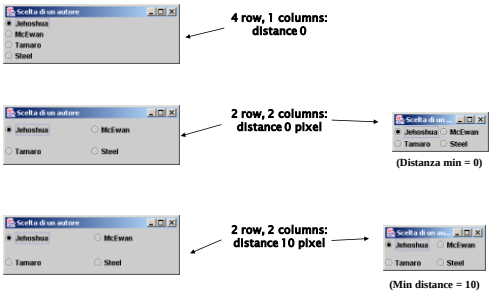
SOftEng

Operators of layout – GridLayout

- splits the screen in a grid of rows and columns
- filling: the box in the top left Builders:
  - GridLayout g = new GridLayout(int rows, int cols);
  - GridLayout g = new GridLayout(rows, cols, hgap, vgap);
- Subjects:
  - rows: number of row;
  - cols: number of columns;
  - hgap: Spacing (in pixels) between two horizontal boxes (default: 0 pixel)
  - vgap: spacing (in pixel) between two vertical boxes (default: 0 pixel)

SOftEng

Example of GridLayout



SOftEng

Operators of layout – BorderLayout

- split into five areas (“North”, “South”, “East”, “West”, “Center”)
- The filling is ‘targeted on’:

```
JPanel pannel = new JPanel();
BorderLayout b = new BorderLayout();
pannel.setLayout(b);
pannel.add(“North”, buttonNord);
pannel.add(“West”, buttonOvest);
Etc…
```

What area of the screen

Which component add

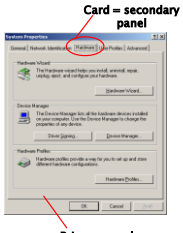
- Alternative builder: BorderLayout(int1, int2), Where the two issues are the spaces between the components related horizontal and vertical

SOftEng

Operators of layout – CardLayout

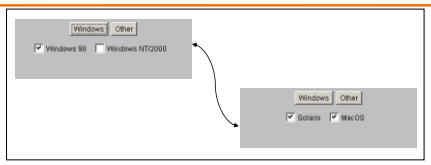
- The last two operators have properties advanced
- With CardLayout is possible to have different panels in the frame, but only one show to time
- the panels are called *cards*
- Metodology:
  - Create a primary panel
  - Its layout is CardLayout
  - create secondary panels
  - add them to primary :

    panPrimary.add(“Secondary Title”, secondaryName)



SOftEng

Example of CardLayout



- The two cards are mutually exclusive: when the first becomes visible, the second shall enter in the background and vice versa
- How make them visible?

We must act through the operator itself:

```
CardLayout c = new CardLayout;
c.show(OneOfPanel, “PanelName”);
```

SOftEng

Operators of layout – GridBagLayout

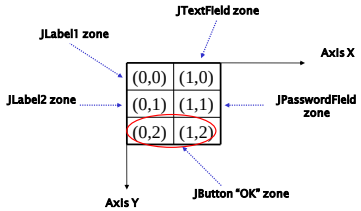
- Extension of the layout to grid (GridLayout)
- is possible to adjust the elements of the grid with mechanisms of personalization
- METHODOLOGY OF USE :
  - Create a body of class GridBagLayout
  - Create a body of 'regolation tool' (class GridBagConstraints)
  1. Regular each component
  2. Inform the operator of adjustments
  3. Made add components to the panel

Exercise

- To understand the meaning of regulations, try to create a component of this type:

GridBagConstraints in detail

- The previous project can be seen in a schematic way in the following way :



Regolation on GridBagConstraints (1)

1. Components are put in the cells of positions (x, y)
2. "OK" button must occupy two cells: the other components are in a single cell
3. breadth of the components is variable (the label "name" occupies about 30% of line…)
4. Cells are positioned (the "OK" button is centered, etc.)

Regolation on GridBagConstraints (2)

✂Constructor GridBagConstraints has different parameters:

- **gridx – The initial gridx value.**
- **gridy – The initial gridy value.**
- **gridwidth – The initial gridwidth value.**
- **gridheight – The initial gridheight value.**
- **weightx – The initial weightx value.**
- **weighty – The initial weighty value.**
- **anchor – The initial anchor value.**
- **fill – The initial fill value.**
- **insets – The initial insets value.**
- **ipadx – The initial ipadx value.**
- **ipady – The initial ipady value.**

Regolation on GridBagConstraints (3)

- The values of fill are : BOTH, NONE, HORIZONTAL, VERTICAL
- The values of anchor are: CENTER, NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST
- Therefore…

```
GridBagLayout grid = new GridBagLayout();
pannel.setLayout(grid);
GridBagConstraints Gbc = new GridBagConstraints();
JLabel label1 = new JLabel ("Name:", JLabel.LEFT);
Gbc.gridx = 0;
Gbc.gridy = 0;
Gbc.gridwidth = 1;
Gbc.gridheigth = 1;
Gbc.weightx = 30;
Gbc.weigthy = 40;
Gbc.fill = GridBagConstraints.NONE;
Gbc.anchor = GridBagConstraints.EAST;
grid.setConstraints(Gbc, label1);
pannello.add(label1);
```

Java Events

SOftEng http://softeng.polito.it

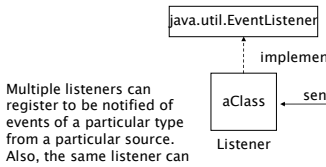
Event Delegation Model

- From Java1.1
  - Events are classified by type (MouseEvent, KeyEvent, ecc.)
  - Events are generated in components sources
  - An object can be registered as listener (listener) of a type of event by sending a message to the component source

Event Delegation Model

- Whenever an event occurs the AWT thread send a message to all the registered listener objects (the event is passed as a parameter)
- A listener object must implement appropriate interface (to make possible the call-back)

Event Delegation Model

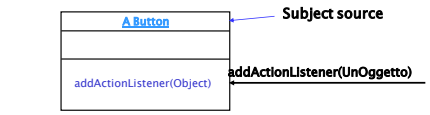


Multiple listeners can register to be notified of events of a particular type from a particular source. Also, the same listener can listen to notifications from different objects.

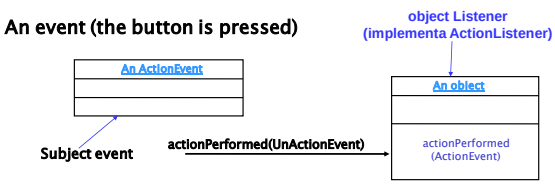
Event

- The events are represented by a hierarchy of classes. Each class is defined by the data representing that type of event.
- Some of the classes that are a set of events (mouseevent) MAY CONTAIN AN ID that identifies the exact class.

Example



An event (the button is pressed)

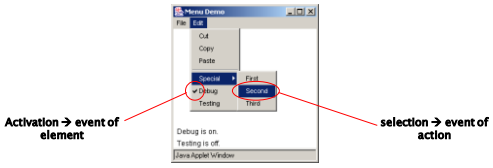


Events in Java: sources and types

Source	Event
Button	ACTION events → when the button is push
Box of choice	ELEMENT events → select/deselect
Menu	ACTION event → when you select a menu item; ELEMENT event → when a selectable menu item is activated
Window	WINDOW events → when the window is activated,maximized,minimized,...
etc.	

SOftEng

Difference between 'selection – activation'



SOftEng

Management of the events in Java

- Problem: We must make receptive interface Java
- Events covered in Java :
  - Action event → *click* a button
  - Adjustment event → Actions on scroll bars
  - Focus event → Point the mouse on a text field
  - Item event → *clic* on RadioButton, CheckBoxButton
  - Key event → keyboard input
  - Mouse event → *click* (not covered above)
  - Mouse-motion event → Simple displacement of the mouse
  - Window event → Enlarge, close a window

SOftEng

An example of management of the events (1/4)

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

public class MouseEvents extends JFrame implements MouseListener,
    MouseMotionListener {

    int mouseX = 0, mouseY = 0;

    public MouseEvents() {
        super("Mouse Events");
        setSize(300,120);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        JPanel pannello = new JPanel();
        pannello.addMouseListener(this);
        pannello.addMouseMotionListener(this);
        setContentPane(pannello);
        setVisible(true);
    }
```

SOftEng

An example of management of the events (2/4)

```
public void mouseClicked(MouseEvent me) {
        mouseX = 0;
        mouseY = 10;
        System.out.println("noted mouse click.");
    }

    public void mouseEntered(MouseEvent me) {
        mouseX = 0;
        mouseY = 10; // Position of the entry' = in the top left (under the edge
    of the frame)
    System.out.println("Mouse is in sensible zone.");
    }

    public void mouseExited(MouseEvent me) {
        mouseX = 0;
        mouseY = 10;
        System.out.println("Mouse is out of sensible zone.");
    }
```

SOftEng

An example of management of the events (3/4)

```
public void mousePressed(MouseEvent me) {
        mouseX = me.getX();
        mouseY = me.getY();
        System.out.println("Mouse push");
    }

    public void mouseReleased(MouseEvent me) {
        mouseX = me.getX();
        mouseY = me.getY();
        System.out.println("Mouse release");
    }

    public void mouseDragged(MouseEvent me) {
        mouseX = me.getX();
        mouseY = me.getY();
        System.out.println("You are dragging the mouse in personal data " +
    mouseX + ", " + mouseY);
    }
```

SOftEng

An example of management of the events (4/4)

```
    public void mouseMoved(MouseEvent me) {
        System.out.println(" You are moving the mouse in personal data " +
    me.getX() + ", " + me.getY());
    }

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

SOftEng

How to manage events in Java

- The principle underlying the events is quite similar to the exceptions :
  - the class declares which event is able to deal with (one or more) → implements one or more interfaces
  - joins a listener to components that are source of events (jbutton, jtextfield, etc..) → JButton.addActionListener(this)
  - Pay attention!You're implementing interfaces, so you must overwrite all methods of those interfaces!

SOftEng

How to manage the events in Java

```
class FrameWithEvents extends JFrame implements
InterfaceWithEvents {
    JComponent componentSourceofEvents = new
JComponent();
    componentSourceOfEvents.addListener(this);
    void methodOfTheInterfaceWithEvents() {…}
    void anotehrMethodOfTheInterfaceWithEvents() {…}
}//end class
```

SOftEng

Event Interfaces (1)

§ **ActionListener** → Methods to override :
- void actionPerformed (ActionEvent evt)

§ **FocusListener** → Methods to overwrite :
- void focusGained (FocusEvent evt)
- void focusLost (FocusEvent evt)

§ **ItemListener** → Methods to rewrite :
- void itemStateChanged (ItemEvent evt)

SOftEng

Event Interfaces (2)

4.MouseListener (→ Methods to rewrite :
- void mouseClicked (MouseEvent evt)
- void mouseEntered (MouseEvent evt)
- void mouseExited (MouseEvent evt)
- void mousePressed (MouseEvent evt)
- void mouseReleased (MouseEvent evt)

5.MouseMotionListener (→ Methods to rewrite :
- void mouseDragged (MouseEvent evt)
- void mouseMoved (MouseEvent evt)

SOftEng

Event Interfaces (3)

6. **KeyListener** → Methods to rewrite :
   - void keyPressed (KeyEvent evt)
   - void keyReleased(KeyEvent evt)
   - void keyTyped(KeyEvent evt)
7. **WindowListener** (→ Methods to rewrite :
   - void windowActivated(WindowEvent evt)
   - void windowClosed (WindowEvent evt)
   - void windowClosing (WindowEvent evt)
   - void windowDeactivated (WindowEvent evt)
   - void windowDeiconified (WindowEvent evt)
   - void windowIconified (WindowEvent evt)
   - void windowOpened (WindowEvent evt)

SOftEng

Add a listener

- Two equivalent methods :
  - the component adds on itself listener
    - JButton.addActionListener(this)
  - The main panel (ad es. JFrame) adds listeners to components JFrame.addActionListener(JButton)
- how to deal with the event ?
  1. find out who has created the event

```
Object ob = evt.getSource();
if (ob == ButtonSelfDestruction ) // Please note the
    operator '=='
  destroyAll();
```

  - Manage the event with the methods of classes relating (KeyEvent.class, WindowEvent.class, ecc.)

Exercise

- CardLayout presented before has an internal management of the Events :
  - pressing the button "windows" , the box with options related to Windows is shown
  - by pressing the "other" the box with options related to Windows is shown
- try to obtain this result

How to manage events

- Every function that appears at the interfaces presents a common argument (KeyEvent, MouseEvent, etc.)
- Each argument is an object and it provides methods to get information about the event:
- Esempi :
  - ActionListener
    - → String getActionCommand(): returns a string identifying the component which generated the command
    - → String paramString(): returns a string describing the event type (common to all event objects)

Methods associated with the event objects

- ItemEvent:
  - int getStateChange(): return SELECTED or DESELECTED on whether the RadioButton or the CheckBox is turned or less
- KeyEvent:
  - char getKeyChar(): Returns the character typed by keyboard → also useful in the case of a case from the keyboard
  - int getKeyCode(): Unicode returns the code of the character
  - String getKeyText(): STRESSES the events due to keys as "Insert", "PageUp", etc. [NOT modifiers as "Shift" or "Ctrl"]
- WindowEvent:

Possible events:

WINDOW_ACTIVATED
WINDOW_CLOSED
WINDOW_CLOSING
WINDOW_DEACTIVATED
WINDOW_DEICONIFIED
WINDOW_GAINED_FOCUS
WINDOW_ICONIFIED

Possible methods:

Window getWindow()

Window getOtherWindow()

Exercise

- Build a graphic interface for the exercise of voting :
  - Which portions of the Code must be maintained intact?
  - What you must rewrite ?
  - Which may be adapted ?

For Who has patience (and desire)...

Exercise:

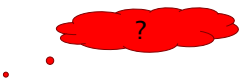
- Write a program Calcolator.java that realizes the functionality of a simple calculator. Requirements of the graphics interface:
  - 10 buttons with the figures from 0 to 9 prepared as in a traditional calculator;
  - Buttons relating to the operations of sum, subtraction, multiplication and division;
  - button "CE" To cancel the last number wrought ;
  - button "C" To clear any operation ;
  - button"=" To claim the result ;
  - button "." To insert decimal places;
  - label to represent the display the calculator.

Observation

- Model the behavior of a "simple" Pocket calculator is not a simple activity : in order to make this exercise not too heavy from the point of view of the algorithms, you can simplify the following algorithm of calculation of arithmetic expressions introduced with the following hypotheses:
  - The expressions involve always and only 2 operandi ;
  - The user inserts always and only the first working operator, then the second working and the key "="; the result becomes the first working operator for the next operation
  - The only exception to the rule (2) is the case for buttons "C", "EC" AND "off" that can be pressed at any time.

Algorithm of management (1)

```
state = NO_OPERATOR
buffer = 0
For each button pressed :
    if the state is NO_OPERATOR then
        If the button pressed is a figure then
            queues to buffer the figure
            view the buffer
        If the button pressed is a sign then
            operator1 = buffer
            operando = segno
            buffer = 0
            state = AN_OPERATOR
        if the botton pressed is C or CE
            buffer = 0
            view the buffer
    otherwise, if the State is AN_OPERATOR then
        If the button pressed is a figure then
            queues to buffer the figure
            view the buffer
        If the button pressed is "=" then
            operator2 = buffer
            Calculate the term operatore1, a sign, operatore2 and visualizzala  buffer =
0
            state = NO_OPERATORE
```



Algorithm of management (2)

```
If the button pressed is C
        buffer = 0
        view the buffer
        state = NO_OPERATORE
If the button pressed is CE
        buffer = 0
        view the buffer
```

Applet

Motivation

- Html pages are static
- applet added
  - capacity for processing
  - dynamic interaction with the user

Java applet

- programs that require a browser to be carried out (using environment Run-time of the browser)
- generally small
- Subject to restrictions of security (sandbox): the browser may prevent them from launch other applications
  - Access to local file system
  - access to information on the system that runs freely
  - communicate via network

Constraints

- For matters of security
  - No reading/writing on local file system
  - No communication if not server from which have been downloaded
  - No fork processes in local, no execution local programmes

SOftEng

Working

- tag <applet> in the html page
- The browser ago downloads of Java classes necessary
- VM generates instance of the class page
- <applet code = “x.class”>
  - There is no “main” method
- Browser call the methods on this application, in function call events or other methods

SOftEng

Characteristics of applet

Java applets are carried out within the WWW browser (hotjava, Netscape), or with the appletviewer provided in JDK.

- Inside a HTML page To insert a pointer to applet, through a specific HTML tags.
- When the browser, reading the html page, finds the tag <APPLET>, Download the applet from the web server .
- The applet is carried out on the local system, the client, where the browser resides.

The applet, because `perform in a graphical environment supported in the browser can use all the capacity of management of the graphics, images, user interfaces and access network in the browser.

SOftEng

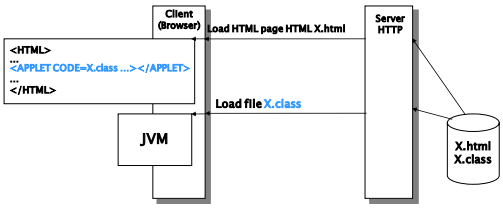
Limits of applet

- applets are subject to many restrictions on their capacity for security reasons, because `carry out on the local machine.
- May not read and write the file system the local machine, with the exception of some Directory specifically indicated by the user;
- cannot communicate with other server other than that from which arose;
- may not run programs on local file system (for example may not do fork processes);
- May not load programs local natives of the platform, including libraries DLL .

=> The compiler and the performer Java perform several checks of consistency and security.

SOftEng

Implementation of a Java applet



SOftEng

Implementation of Java applet \

- <html>
- <applet code="WelcomeApplet.class" width=300 height=30>
- </applet>
- </html>

```
import javax.swing.JApplet;  // import class JApplet
import java.awt.Graphics;    // import class Graphics

public class WelcomeApplet extends JApplet {
   public void paint( Graphics g )
   {
      g.drawString( "Welcome to Java Programming!", 25, 25 );
      g.drawLine( 15, 10, 210, 10 );
      g.drawLine( 15, 30, 210, 30 );     }}
```

SOftEng

Es: I am a simple program

- As an application

```
import java.awt.*; import java.awt.event.*;import javax.swing.*;
public class SwingUI extends JFrame implements ActionListener {
   JLabel text, clicked;   JButton button, clickButton;   JPanel panel;
        private boolean clickMeMode = true;
   public SwingUI() {  //Begin Constructor
       text = new JLabel("I'm a Simple Program");
       button = new JButton("Click Me");
       button.addActionListener(this);

       panel = new JPanel();     panel.setLayout(new BorderLayout());
       panel.setBackground(Color.white);     getContentPane().add(panel);
       panel.add(BorderLayout.CENTER, text);
       panel.add(BorderLayout.SOUTH, button);
   } //End Constructor
```


Es: I am a simple program

```
import java.awt.*; import java.awt.event.*;import javax.swing.*;
public class SwingUI extends JFrame implements ActionListener {
   JLabel text, clicked;   JButton button, clickButton;   JPanel panel;
        private boolean clickMeMode = true;
   public SwingUI() {  //Begin Constructor
       text = new JLabel("I'm a Simple Program");
       button = new JButton("Click Me");
       button.addActionListener(this);

       panel = new JPanel();     panel.setLayout(new BorderLayout());
       panel.setBackground(Color.white);     getContentPane().add(panel);
       panel.add(BorderLayout.CENTER, text);
       panel.add(BorderLayout.SOUTH, button);
   } //End Constructor
```

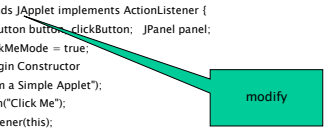

Es: I am a simple program

```
public void actionPerformed(ActionEvent event){
   //    Object source = event.getSource();
      if (clickMeMode) {
        text.setText("Button Clicked");
        button.setText("Click Again");
        clickMeMode = false;
      } else {
        text.setText("I'm a Simple Program");
        button.setText("Click Me");
        clickMeMode = true;
      }       }
}
```


Es: I am a simple program

- Processing in applet

```
import java.awt.*; import java.awt.event.*;import javax.swing.*;
public class SwingUI extends JApplet implements ActionListener {
   JLabel text, clicked;   JButton button, clickButton;   JPanel panel;
        private boolean clickMeMode = true;
   public SwingUI() {  //Begin Constructor
       text = new JLabel("I'm a Simple Applet");
       button = new JButton("Click Me");
       button.addActionListener(this);

       panel = new JPanel();     panel.setLayout(new BorderLayout());
       panel.setBackground(Color.white);     getContentPane().add(panel);
       panel.add(BorderLayout.CENTER, text);
       panel.add(BorderLayout.SOUTH, button);
   } //End Constructor
```



Es: I am a simple program

```
public void actionPerformed(ActionEvent event){
   //    Object source = event.getSource();
      if (clickMeMode) {
        text.setText("Button Clicked");
        button.setText("Click Again");
        clickMeMode = false;
      } else {
        text.setText("I'm a Simple Program");
        button.setText("Click Me");
        clickMeMode = true;
      }    }
public void init(){
    SwingUI s = new SwingUI();
}
}
```

```
<html>
<applet code="SwingUI.class" width=300 height=30>
</applet>
</html>
```


Es: I am a simple program

```
public class AdditionApplet extends JApplet {
double sum;
public void init()
{     String firstNumber,  secondNumber;   double number1,  number2;
        // read in numbers from user
   firstNumber = JOptionPane.showInputDialog("Enter  floating-point value" );
secondNumber =JOptionPane.showInputDialog("Enter floating-point value" );
   // convert numbers from type String to type double
   number1 = Double.parseDouble( firstNumber );
   number2 = Double.parseDouble( secondNumber );
   sum = number1 + number2; }

public void paint( Graphics g )
{     g.drawRect( 15, 10, 270, 20 );
   g.drawString( "The sum is " + sum, 25, 25 );   }}
```

Basic methods

- Inherited empty (da javax.swingJApplet o java.awt.Applet) o override
  - init() // Initialize confronted (in practice replaces manufacturer)
  - start() // Starts implementation application- generally called after init () or after stop () (User back to page)

Basic methods

- stop()// suspending implementation (called when user abandons the page where applet is running)
- destroy() // destroys confronted, shall issue the resources (when browser ends)
- paint() // Update the part of screen controlled by applet

Order called

- init()
- start()
- paint()

- called on the order from appletviewer or browser

Passage parameters browser-applets

- In the browser: use tag PARAM
- <Applet Code="myApplet.class" width=100 height=100 >
- <param name=font value="TimesRoman">
- <param name=size value="36">
- </applet>

Passage parameters browser-applets

- In the applet
- method getParameter() // typically in init()
- Receives string with name parameter, makes string with value , or null
- String s1 = getParameter("font");
- String s2 = getParameter("size");

Demonstration of the methods of an applet

```
import java.awt.*;
import java.applet.*;

public class TestApplet extends Applet {
  String s;
  int inits=0, starts=0, stops=0;  // contatori
  public void init() { inits++; }
  public void start() { starts++; }
  public void stop() { stops++; }
  public void paint (Graphics g) {
    s = "inits: "+ inits + "starts: " + starts +
        "stops: "+ stops;
    g.drawString(s, 10, 10);
  }
}
```

An example: the applet hello again

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;
public class HelloAgainApplet extends
   java.applet.Applet {
   Font f = new Font("TimesRoman", Font.BOLD, 36);
   public void paint(Graphics g) {
      g.setFont(f);
      g.setColor(Color.red);
      g.drawString("Hello World!", 5, 25);
   }
}
```

SOftEng

The applet hello again: HTML

```
<HTML>
<HEAD>
   <TITLE>Hello to Everyone!</TITLE>
</HEAD>
<BODY>
   <P> My applet says:
   <APPLET CODE="HelloAgainApplet.class" WIDTH=250
HEIGHT=25>
   </APPLET>
</BODY>
</HTML>
```

SOftEng

Inclusion of applet in a HTML page

you are using the tag <APPLET>. Its parameters are very similar to those of the tag <IMG>.

- WIDTH e HEIGHT Define the area of the screen (in pixels) dedicated to applet .
- If the applet is small, may be "Included" in a line of text. In this case, Align defines the alignment of applet in line with the other elements of the line. can take the values LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM e ABSBOTTOM.

SOftEng

Inclusion of applet in a HTML page

- HSPACE and VSPACE fissano lo spazio (in pixel) tra la applet ed il testo che la circonda.
- CODE and CODEBASE indicano rispettivamente il nome del file .class che contiene la applet e la directory in cui si trova il file.
- Il testo contenuto tra <APPLET> e </APPLET> e` mostrato dai browser che non interpretano il tag <APPLET>.

In HTML 4.0 si utilizza il tag <OBJECT>. I suoi parametri sono molto simili a quelli del tag <APPLET>, ad eccezione dell'uso del parametro CLASSID:"java:pippo.class" al posto del parametro CODE.

SOftEng

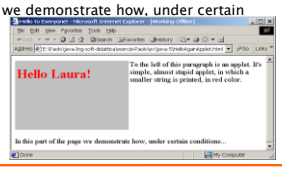
A complete example

```
import java.awt.Graphics;
import java.awt.Font;
import java.awt.Color;
public class MoreHelloApplet extends java.applet.Applet {
    Font f = new Font("TimesRoman",Font.BOLD,36);
    String name;
    public void init() {
        this.name = getParameter("name");
        if (this.name == null)
            this.name = "Paolo";
        this.name = "Hello " + name + "!";
    }
    public void paint(Graphics g) {
        g.setFont(f);
        g.setColor(Color.red);
        g.drawString(this.name, 5, 50);
    }
}
```

SOftEng

A complete example : HTML

```
<HTML>  <HEAD>
  <TITLE>Hello to Everyone!</TITLE>
</HEAD>  <BODY>
  <P><APPLET CODEBASE="." CODE="MoreHelloApplet" WIDTH=300
    HEIGHT=200 ALIGN=LEFT>Hello Again!</APPLET>
To the left of this paragraph is an applet. It's simple, almost stupid
applet, in which a smaller string is printed, in red color.
<BR CLEAR=ALL>
<P>In this part of the page we demonstrate how, under certain
    conditions...
</BODY>
</HTML>
```



SOftEng

Archives: jar files

To avoid duty is to open a connection for each single file necessary to execute the applet (. class files, audio files, images, text files) it is possible to create an archive or a jar files.

- An archive Java is a set of classes and other files (compressed) contained in a single file .
- The JDK programme provides a jar that allows you to create archives .

```
jar cf Animazione.jar  *.class  *.gif    // Create archive
```

Using the parameter archive= in the tag <applet> to allow the browser to transfer the archive. It is however necessary to specify by the parameter code the name the executable .

SOftEng

Graphics

SOftEng

Graphics: the class graphics

Is the class that supports the capacity graphics applets, which draw lines, forms, characters and present images on screen, by means of a series of methods .

It isn't necessary to create an instance of the class graphics to draw on the screen

=> The method paint () provides an object graphics acting on which draws on the screen.

SOftEng

Graphics: the class graphics

- The system of two-dimensional coordinates has
  - origin, tgat is point (0,0) in the top left
  - The positive values of coordinated X is moving on the right
  - The positive values of coordinated y is moving in the lower
- The points point, used as a reference to draw any object, express the coordinates in pixels on the screen and are integer values .

SOftEng

Draw Lines and squares

To draw a line

```
public void paint(Graphics g) {
    g.drawLine(25, 25, 75, 75);
}
```

To Draw a rectangle, specifying the coordinated point in the top left, width and length:

```
public void paint(Graphics g) {
    g.drawRect(20, 20, 60, 60); // x0, y0, width, height
    g.fillRect(120, 20, 60, 60);
}
```

To Draw a rectangle, specifying the coordinated point in the top left, width and length:

```
public void paint(Graphics g) {
    g.drawRoundRect(20, 20, 60, 60, 10, 10);
    g.fillRoundRect(120, 20, 60, 60, 20, 20);
}
```

To Draw a rectangle "3d" (You get a 3d effect style buttons):

```
public void paint(Graphics g) {
    g.draw3DRect(20, 20, 60, 60, true);
    g.fill3DRect(120, 20, 60, 60, false);     }
```

SOftEng

Draw polygon

To draw a polygon and necessary to define a set of coordinates x and y, as array or as instances of the class polygon.

```
public void paint(Graphics g) {
    int x[ ] = {39, 94, 97, 142, 53, 58, 26};
    int y[ ] = {33, 74, 36, 70, 108, 80, 106};
    int points = x.length;
    g.drawPolygon(x,y,points);
}
public void paint(Graphics g) {
    int x[ ] = {39, 94, 97, 142, 53, 58, 26};
    int y[ ] = {33, 74, 36, 70, 108, 80, 106};
    int points = x.length;
    Polygon poly = new Polygon(x,y,points);
    g.fillPolygon(poly);
}
```

The polygon is closed automatically from Java (1.2).

- drawPolyline() allows to have open polygons.

It is possible to add points to a subject Polygon

```
poly.addPoint(20,35);
```

SOftEng

Draw circles, ellipses and strings

To draw rims or ellipses using the oval .

```
public void paint(Graphics g) {
    g.drawOval(20, 20, 60, 60); // x0, y0, width, height
    g.fillOval(120, 20, 100, 60);  colora l'ovale nel rettangolo s
}
```

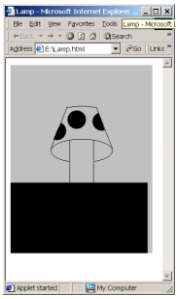
- the strings are defined as pieces of ellipses with the method drawArc()
- you define initially the size of the Circle (ellipse), which otterra' the arc. Finally, it must provide the points at the beginning and end of the ARC through the corner at the beginning and the angle subtended by arc.
- corners defined positive counterclockwise (90 vertical axis).

```
public void paint(Graphics g) {
    g.drawArc(20, 20, 60, 60, 90, 180);
    g.fillArc(120, 20, 60, 60, 90, 180);
}
public void paint(Graphics g) {
    g.drawArc(10, 20, 150, 50, 25, -130);
    g.fillArc(10, 80, 150, 50, 25, -130);
}
```

SOftEng

A simple example

```
import java.awt.*;
public class Lamp extends java.applet.Applet {
    public void init() {
        resize(300,300);
    }
    public void paint(Graphics g) {
        g.fillRect(0,250,290,290); // the base
        g.drawLine(125,250,125,160);//the stem
        g.drawLine(175,250,175,160);
        g.drawArc(85,157,130,50,-65,312);
        g.drawArc(85,87,130,50,62,58);
        g.drawLine(85,177,119,89);
        g.drawLine(215,177,181,89);
        g.fillArc(78,120,40,40,63,-174); //draw
        g.fillOval(120,96,40,40);
        g.fillArc(173,100,40,40,110,180);
    }
}
```



SOftEng

Copy and delete pieces of screen

- to copy a piece of screen using the method copyarea () with the parameters x and y the angle on the top left of the rectangle to copy, width and height of the rectangle, relative distance in X and Y of the area on which copy .

  Example

  g.copyArea (0, 0, 100, 100, 100, 0);
- To cancel a piece of screen using the method that clearrect has the same parameters of drawRect.

SOftEng

Copy and delete pieces of screen

In this way stains with the color of the region specified background as a parameter in the call to the method clearRect().

Example: delete the whole area dedicated to applet .

g.clearRect (0, 0, width, height);

SOftEng

Print text and font management

To write on the screen you must first create an instance of the class font .

The objects fonts are identified by :

- A name: a string that represents the family of font: timesroman, courier, Helvetica. IN THE VERSION 1.2 USING THE NAMES. Serif, monospaced, sanserif .
- Style: a constant whole (Font.PLAIN, Font.BOLD, Font.ITALIC)

  => The constants be added together : Font.BOLD + Font.ITALIC

SOftEng

Print text and font management

- The dimension in points: a number

```
public void paint(Graphics g) {
    Font f = new Font("TimesRoman", Font.BOLD, 72):
    g.setFont(f);
    g.drawString("This is a huge font .", 10, 100);
}
```

- There is also the method drawChars () of the class graphics, which requires as parameters an array of characters, a whole that is the first character to play on the screen, a whole for the last character to represent, and coordinates X and Y.

SOftEng

Font management

The most important methods to obtain information about current font are:

- getFont(): Return the current font
- getName(): ritorna una stringa con il nome del font
- getSize(): Return a string with the name of the font
- getStyle(): Return the style of font
- isPlain()
- isBold()
- isItalic()

For more information more specific on the individual font exploits the class FontMetrics .

Font management

Principal method are:

- stringWidth(): date a string, it will return the width in pixels
- charWidth(): As a character it will return the amplitude
- getAscent(): Return the distance between the base and the far superior of the font
- getDescent(): Return the distance between the base and the lower end of the font
- getLeading(): Return the space between the lower end of a character and the far higher than what in the next line
- getHeight(): Return the total height of the font

Example of font management

- The following applet centra horizontally and vertically written compared to the area dedicated to applet .

```
import java.awt.Font;
import java.awt.Graphics;
import java.awt.FontMetrics;
public class Centered extends java.applet.Applet {
   public void paint(Graphics g) {
        Font f = new Font("TimesRoman", Font.PLAIN, 36);
        FontMetrics fm = getFontMetrics(f);
        g.setFont(f);
        String s = "A bee on a bee makes a baby bee.";
        int xstart = (this.getSize().width - fm.stringWidth(s)) / 2;
        int ystart = (this.getSize().height + fm.getHeight()) / 2;
        g.drawString(s, xstart, ystart);
   }   }
```


Color management

- The methods for managing the colors are contained in the class color .
- The colors are encoded on 24 bit; each color and consists of a combination of red, green and blue .
- Each component is represented with a whole number between 0 and 255 (or with a float line between 0 and 1).

are defined class variables for the main colors .

Color management

Color	RGB code	Color	RGB code
Color.white	255, 255, 255	Color.blue	0, 0, 255
Color.black	0, 0, 0	Color.yellow	255, 255, 0
Color.lightGray	192, 192, 192	Color.magenta	255, 0, 255
Color.gray	128, 128, 128	Color.cyan	0, 255, 255
Color.darkGray	64, 64, 64	Color.pink	255, 175, 175
Color.red	255, 0, 0	Color.orange	255, 200, 0
Color.green	0, 255, 0		

Color management

The methods most important for the management of the colours are :

- setColor(): set current color
- setBackground(): Sets the current Color the wallpaper
- setForeground(): Change the color of the objects of the user interface (for example, buttons) and acts on components of the user interface, not on an instance of the class graphics

All methods there is the corresponding method Get... () that allows to read the current color.

An example of color management

```
import java.awt.Graphics;
import java.awt.Color;
public class ColorBoxes extends java.applet.Applet {
    public void paint(Graphics g) {
        int rval, gval, bval;
        for (int j = 30; j < (this.getSize().height -25); j += 30)
        for (int i = 5; i < (this.getSize().width -25); i+= 30) {
            rval = (int) Math.floor(Math.random() * 256);
            gval = (int) Math.floor(Math.random() * 256);
            bval = (int) Math.floor(Math.random() * 256);
            g.setColor(new Color( rval,gval,bval ));
            g.fillRect(i,j,25,25);
            g.setColor(Color.black);
            g.drawRect( i-1, j-1, 27, 27);
        }
    }
}
```


Animation

To run the entertainment of an image must be taken two steps:

1. Definition of the image to animate
2. Recovery of the upgrade of the screen so as to create the illusion of movement

- The method paint () is automatically called from Java all the times that it is necessary to update (cool) the area of video dedicated to applet: the first time an applet is activated, every time you move the browser window, each time another window overlaps that of the browser. . .).

Animation

- It is possible to order explicitly to Java implementation of the method of updating of the screen whenever necessary .
- To change what you see on the screen, and `sufficient to achieve an image of what is to draw, and ask Java refresh the screen.
- all the amendments necessary to create the images are made in a specific method. The method paint () is only to copy on the screen the current image.

Animation

- At the end of the operations of the preparation of the image, is called the method repaint (), which in turn eseguira' the call to paint ().

  => By running in a cyclical and a certain speed the previous steps, it is a simple animation.

Animation: methods start() and stop()

To run applets containing animated must use the methods start() e stop().

- The method start() initiates the execution of applet .
- The method stop() is executed when the applet suspending its implementation (change of HTML page by the browser) and allows you to release the resources used for the performance of applet.

  => It is necessary to run the redefinition of start() and stop()

Example: digital watch

```
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;
public class DigitalClock extends java.applet.Applet {
    Font theFont = new Font("TimesRoman",Font.BOLD,24);
    Date theDate;
    public void start() {
        while (true) {
            theDate = new Date();
            repaint();
            try { Thread.sleep(1000); }
            catch (InterruptedException e) { }
        }
    }
    public void paint(Graphics g){
        g.setFont(theFont);
        g.drawString(theDate.toString(),10,50);
    }
}
```

Multithreading

The previous example does not work !

- The endless cycle this in the method start () monopolizes the resources of the system, impendendo ALSO TO THE METHOD paint () to make the refreshment of the screen .

It is not possible to stop the applet why not you can call the method stop().

=> The applet should be restated using a thread.

- Each time that it is necessary to perform a sequence of operations of a certain length it should create a separate thread run this operation.
- It is always a good idea to use the programming with the thread when you write an applet.

SOftEng

Currect digital watch

```
import java.awt.Graphics;
import java.awt.Font;
import java.util.Date;
public class DigitalClock extends
    java.applet.Applet
    implements Runnable {
    Font theFont =new Font("Arial",Font.BOLD,
    24);
    Date theDate;
    Thread runner;
    public void start() {
          if (runner == null) {
            runner = new Thread(this);
            runner.start(); // Innesca invocazione
                  // method run() of the applet
          }
    }
    public void stop() {
          if (runner != null) {
            runner.stop();
            runner = null;
    }
}
```

```
public void run() {
while (true) {
Thread thisThread = Thread.currentThread();
  while (runner == thisThread) {
    repaint();
    try { Thread.sleep(1000); }
    catch (InterruptedException e) { }
  }
}
}
public void paint(Graphics g){
   theDate = new Date(); // aggiorno data ogni
                  // volta che devo ridipingere
schermo
  g.setFont(theFont);
  g.drawString(theDate.toString(),10,50);
}
}
// Class Date obsolete; change with Calendar
```

SOftEng

The flicker in animation

The implementation of the method repaint() initiates (indirectly) the call to the method paint():

1. repaint() calls the methodupdate()
2. The method update() Delete the part of screen dedicated to applet (painting everything with the background color or background) and calls the method paint()
3. The method paint() refreshes the screen designing the new image

=> The phase which draws the background because the flicker .

SOftEng

The flicker in animation

To avoid the flicker can :

1. redefine (overriding) the method update () so as not redesign the background
2. redefine (overriding) the method update () In order to draw only the part that is changed
3. redefine method is the update () method is the paint () and use the technique of double buffering

SOftEng

Update() method

The standard version of the method update() is:

```
public void update(Graphics g){
    g.setColor(getBackground());
    g.fillRect(0, 0, size().width, size().height);
    g.setColor(getForeground());
    paint(g);
}
```

When it redefines the method update () The new version must perform all the instructions that are necessary for the functioning of applet .

SOftEng

First solution: not redesign the background

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;
public class ColorSwirl extends java.applet.Applet
    implements Runnable {
    Font f = new Font("Arial",Font.BOLD,48);
    Color colors[] = new Color[50];
    Thread runThread;
    public void start() {
      if (runThread == null) {
          runThread = new Thread(this);
          runThread.start();
      }
    }
public void stop() {
    if (runThread != null) {
    runThread.stop();
    runThread = null;
}
}
```

```
public void run() {
// should be better in method init !!!
  float c = 0;
  for (int i = 0; i < colors.length; i++) {
    colors[i]= Color.getHSBColor(c,
(float)1.0,(float)1.0);
    c += .02;
}
  int i = 0;   // cycle through the colors
  while (true) {
    setForeground(colors[i]);
    repaint();
    i++;
    try { Thread.currentThread().sleep(50); }
    catch (InterruptedExceptione) { }
     if (i == (colors.length)) i = 0;
  }
}
public void paint(Graphics g) {
  g.setFont(f);
  g.drawString("All the Swirly Colors", 15,50);
}
public void update(Graphics g){
  paint(g);
} }
```

SOftEng

First solution: not redesign the background

```
import java.awt.Graphics;
import java.awt.Color;
public class Checkers extends
    java.applet.Applet
    implements Runnable {
    Thread runner;
    int xpos;
    public void start() {
      if (runner == null); {
      runner = new Thread(this);
      runner.start();
    }
}
public void stop() {
    if (runner != null) {
      runner.stop();
      runner = null;
    }
}
```

```
public void run() {
  setBackground(Color.blue);
  while (true) {
    for (xpos = 5; xpos <= 105; xpos+=4) {
      repaint();
      try { Thread.sleep(100); }
      catch (InterruptedException e){}
  }
  for (xpos = 105; xpos > 5; xpos -=4) {
    repaint();
    try { Thread.sleep(100); }
    catch (InterruptedException e){}
  }
}
}
public void paint(Graphics g) {
  g.setColor(Color.black); // Draw background
  g.fillRect(0,0,100,100);
  g.setColor(Color.white);
  g.fillRect(101,0,100,100);
  g.setColor(Color.red); // Draw checker
  g.fillOval(xpos,5,90,90);
}     }
```

SOftEng

Checkers applet

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.Font;
public class Checkers extends java.applet.Applet
    implements Runnable {
    Font f = new Font("Arial",Font.BOLD,48);
    Color colors[] = new Color[50];
    Thread runThread;
    public void start() {
      if (runThread == null) {
          runThread = new Thread(this);
          runThread.start();
      }
    }
public void stop() {
    if (runThread != null) {
    runThread.stop();
    runThread = null;
}
}
```

```
public void run() {
// should be better in method init !!!
  float c = 0;
  for (int i = 0; i < colors.length; i++) {
    colors[i]= Color.getHSBColor(c,
(float)1.0,(float)1.0);
    c += .02;  }
  int i = 0;    // cycle through the colors
  while (true) {
    setForeground(colors[i]);
    repaint();
    i++;
    try { Thread.currentThread().sleep(50); }
    catch (InterruptedException  e) { }
     if (i == (colors.length)) i = 0;
  } }
public void paint(Graphics g) {
  g.setFont(f);
  g.drawString("All the Swirly Colors", 15,50);
}
public void update(Graphics g){
  paint(g);
} }
```

SOftEng

Other solution: redesign only the parts necessary

```
import java.awt.Graphics;
import java.awt.Color;
public class Checkers2 extends java.applet.Applet
    implements Runnable {
    Thread runner;
    int xpos,ux1,ux2;
    public void run() {
      setBackground(Color.blue);
      while (true) {
        for (xpos = 5; xpos <= 105; xpos+=4) {
          ux2 = xpos + 90;
          repaint();
          try { Thread.sleep(100); }
          catch (InterruptedException e) { }
          if (ux1 == 0) ux1 = xpos; // Importante
per   assicurare esecuzione del metodo paint
        }
        for (xpos = 105; xpos > 5; xpos -=4) {
          ux1 = xpos;
          repaint();
          try { Thread.sleep(100); }
          catch (InterruptedException e) { }
          if (ux2 == 0) ux2 = xpos + 90;
      } } }
```

```
public void update(Graphics g) {
    g.clipRect(ux1, 5, ux2 - ux1, 95);
    paint(g);
}
public void paint(Graphics g) {
    g.setColor(Color.black);
    g.fillRect(0,0,100,100);
    g.setColor(Color.white);
    g.fillRect(100,0,100,100);
    g.setColor(Color.red);
    g.fillOval(xpos,5,90,90);
    ux1 = ux2 = 0; // calcolo area
}
public void start() {
    if (runner == null); {
      runner = new Thread(this);
      runner.start();
    }
}
public void stop() {
    if (runner != null) {
      runner.stop();
      runner = null; } } }
```

SOftEng

Use of images

- The class image in java.awt provides the methods for managing images .
- Java supports images GIF format and JPEG .
- The most important methods to load images are :
  - getimage() load the image
  - getdocumentbase() return the URL that is the directory in which the html file that contains the applet
  - getcodebase() return, in the form of string, the URL of the applet

SOftEng

Use of images

```
Image img = getImage(getDocumentBase(),
   "image.gif"));
   Image img = getImage(getCodeBase(),
   "image.gif"));
   Image img = getImage(getCodeBase(),
   "images/image.gif"));
```

=> If the File not Found, getimage will return null.

- To draw a picture you use the method drawimage () In paint ().

SOftEng

Example: LadyBug applet

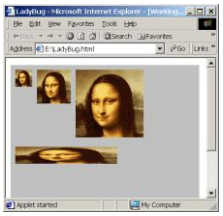
```
import java.awt.Graphics;
import java.awt.Image;
public class LadyBug extends
    java.applet.Applet {
    Image bugimg;
    public void init() {
      bugimg = getImage(getCodeBase(),
      "gioconda.gif");
    // getImage() Torna oggetto Image.
    // Non istanzio bugimg
}
```

```
public void paint(Graphics g) {
  int iwidth = bugimg.getWidth(this);
  int iheight = bugimg.getHeight(this);
  int xpos = 10;    // 25 %
  g.drawImage(bugimg, xpos, 10,
      iwidth /4, iheight /4, this);  // 50 %
  xpos += (iwidth / 4) + 10;
  g.drawImage(bugimg, xpos, 10,
  iwidth/2, iheight /2, this); // 100%
  xpos += (iwidth /2) + 10;
  g.drawImage (bugimg,xpos, 10, this);
       // 150% x, 25% y
  g.drawImage(bugimg, 10, iheight + 30,
    (int) (iwidth * 1.5), iheight /4, this);
}
}
```

SOftEng

Example: LadyBug applet

```
<HTML>
<HEAD>
    <TITLE>Clock</TITLE>
</HEAD><BODY>
<P><APPLET CODEBASE="." CODE="LadyBug"
WIDTH=400 HEIGHT=400 ALIGN=LEFT>
      </APPLET>
</BODY>
</HTML>
```



SOftEng

Example of entertainment: the applet Neko

```
import java.awt.*;
public class Neko extends java.applet.Applet
    implements Runnable {
    Image nekopics[] = new Image[9];
    String nekosrc[]={"right1.gif", "right2.gif",
    ........
    "awake.gif" };
    Image currentimg;
    Thread runner;
    int xpos;
    int ypos = 50;
    public void start() {
        if (runner == null) {
        runner = new Thread(this);
        runner.start();
    }
}
public void stop() {
    if (runner != null) {
    runner.stop();
    runner = null;
}
}
public void run() {
    for (int i=0; i < nekopics.length; i++) {
        nekopics[i] = getImage(getCodeBase(),
        "images/" + nekosrc[i]);
    }
    setBackground(Color.white);
    nekorun(0, this.size().width / 2); // run
    currentimg = nekopics[2]; // stop and pause
    repaint();
    pause(1000);
    currentimg = nekopics[3]; // yawn
     repaint();
     pause(1000);
     nekoscratch(4); // scratch four times
     nekosleep(5); // sleep for 5 seconds
    currentimg = nekopics[8]; // wake up and run
off
    repaint();
    pause(500);
    nekorun(xpos, this.size().width +10);
}
```

SOftEng

Example of entertainment: the applet Neko (follow)

```
void nekorun(int start, int end) {
    for (int i = start; i < end; i+=10) {
    this.xpos = i;
    // swap images
    if (currentimg == nekopics[0])
        currentimg = nekopics[1];
    else if (currentimg == nekopics[1])
        currentimg = nekopics[0];
        else currentimg = nekopics[0];
        repaint();
        pause(150);
    }
}
void nekoscratch(int numtimes) {
    for (int i = numtimes; i > 0; i--) {
      currentimg = nekopics[4];
      repaint();
      pause(150);
      currentimg = nekopics[5];
      repaint();
      pause(150);
    }
}
```

```
void nekosleep(int numtimes) {
    for (int i = numtimes; i > 0; i--) {
      currentimg = nekopics[6];
      repaint();
      pause(250);
      currentimg = nekopics[7];
      repaint();
      pause(250);
    }
}
void pause(int time) {
    try { Thread.sleep(time); }
    catch (InterruptedException e) { }
}
public void paint(Graphics g){
    g.drawImage(currentimg,xpos, ypos,this);
}
}
```

SOftEng

Inclusion of sounds in applet

- Java 1.2 Supports sound formats AIFF, WAV, MIDI0, MIDI1 e RMF.
- It is always possible to include sounds through pointers to the external in HTML page.
- To generate a sound

  through class method applet :

  play(getCodeBase(), "audio/meow.au"); // plays the sound once

  Through methods of class AudioClip:

  AudioClip clip = getAudioClip(getCodeBase(), "audio/loop.au");

  clip.play(); // plays the sound once

  clip.stop(); // stops the sound

  clip.loop(); // plays the sound repeatedly }

SOftEng

Inclusion of sounds in applet

We must stop explicitly a sound of background (or a sound that use the method loop ()) in the method stop (), otherwise the sound continues even when the applet finished run.

```
public void stop() {
if (runner != null) {
  if (bgsound !=null)
        bgsound.stop();
  runner.stop();
  runner = null;
}
```

SOftEng

Reduce the flicker: Double buffering

The method more complex to reduce the flicker and `the double buffering .

The method is to create a second area off the screen to design the new image to display; at the end of this process, the surface will be displayed in a blow only on the screen.

=> It is not likely to view parts of the image intermediate thus disturbing the effect of entertainment since and is a technical costly in terms of memory and efficiency, and `well use it only if none of the other technical works.

SOftEng

Checkers applet modified with double buffering

1. Add the variables of application for the external image and its contents chart

```
Image offscreenImg;
Graphics offscreenImg;
```

2. Add a method init () to initialize the external image

```
public void init() {
    offscreenImg = createImage(this.size().width, this.size().height);
    offscreenG = offscreenImg.getGraphics();
}
```

3. Modify the method paint to design the external image

```
public void paint(Graphics g) {
    // Draw the background
    offscreenG.setColor(Color.black);
    offscreenG.fillRect(0, 0, 100, 100);
    offscreenG.setColor(Color.white);
    offscreenG.fillRect(100, 0, 100, 100);
    // Draw the pawn
    offscreenG.getColor(Color.red);
    offscreenG.fillOval(xpos, 5, 90, 90);
    g.drawImage(offscreenImg, 0, 0, this);    }
```

SOftEng

Observations on applet

- The method showstatus () allows you to view information on applet of method getappletcontext(), which will return an object of type appletcontext and enables the applet to access some first of the browser that contains

```
getAppletContext().showStatus("Cambio il colore");
```

Per fornire informazioni associate alla applet

```
public String getAppletInfo() {
    return "Empty applet, Copyright 2002 Paolo Falcarin";
}
```

SOftEng

Observations on applet

to exchange information between applet belonging to the same page HTML :

- Assigning a name to the applet by the parameter NAME= tag APPLET
- The method getapplet () allows the access to the methods and to the variables of instances of other applet .

```
<APPLET CODE="MyApplet1" WIDTH=100 HEIGHT=150 NAME="Trasmitter">
</APPLET>
<APPLET CODE= "MyApplet2" WIDTH=100 HEIGHT=150 NAME="Receiver">
</APPLET>
Applet receiver = getAppletContext().getApplet("Ricevitore");
receiver.update(text, value); // Initiation of the update applets receiver
```

Per caricare un documento HTML e farlo visualizzare dal browser:

```
String url = "http://www.polito.it/~falcarin";
theURL = new URL(url);
getAppletContext().showDocument(theURL); // open a document in
                                         // the same windows
getAppletContext().showDocument(theURL,"_blank"); // open a document
                                         // in a new windows
```

SOftEng

Swing Classes and Events

Management of interactivity: Events

The events are the ways of communication between the user (system) and the program running .

Type of events:

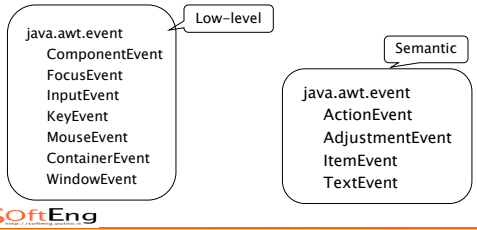
- input from the user :
  - pressure of the buttons of the mouse
  - Movement of the mouse
  - pressure of a key of the keyboard
- Events of user interface:
- - click on button
  - Movement of a scroll bar
  - Viewing menu
- Events of the windows: opening, closing and exit from a window

The management of the events within an application allows you to change the behaviour of applet (or application) in the information supplied by the user.

SOftEng

Low-level events and semantic

- A low-level event is a simple input or an event in the system chart .
- Events semantic way the semantics of a component of user-interface .



SOftEng

Management of the events

- The model of the events introduces the concept of listener of events (listener), which is responsible for the management of specific events .
- The events are separated in different classes, with listeners separate dealing in each Class.
  - There is a package dedicated to the events (java.awt.event)
- The preparation of the events and `in two separate parts:
  - The subject on which it created the event (an applet or a part)
  - the listener in events (which may be different for categories of various events) that performs actions in response to specific events covered
- The two elements are connected through the registration of the listener: are transmitted the listener only events that interested him

SOftEng

Managements of events in Java

To manage the events and `necessary to run the following operations:

1. decide which events must be managed by applet and identify appropriate listeners
2. Define the code for the preparation of the events of the listeners

- a manager of events and `a class that implements one or more listeners interfaces
- Two possibilities :

  (a) creazione di una classe di gestione degli eventi separata

  (b) Introduction of the management of the events in applet

3. Book The listener with the receiver of the events (example: the applet)

SOftEng

Identification of the events

The different listeners are defined by interfaces on the package java.awt.event

Interface	Event	Method
MouseListener	button depressed	void mousePressed(MouseEvent e)
	Button issued	void mouseReleased(MouseEvent e)
	Entrance mouse cursor	void mouseEntered(MouseEvent e)
	Exit mouse cursor	void mouseExited(MouseEvent e)
	mouse click	void mouseClicked(MouseEvent e)
	click = press + release in same position	
MouseMotionListener	Movement of the mouse	void mouseMoved(MouseMotionEvent e)
	Drag of the mouse	void mouseDragged(MouseMotionEvent e)
KeyListener	key pressed	void keyPressed(KeyEvent e)
	Key issued	void keyReleased(KeyEvent e)
	Key typed	void keyTyped(KeyEvent e)
	Key typed = key pressed + key issued	

SOftEng

Creation of a separate class of listening

- It defines the new class as subclass of an adapter of events
  - CLASSES DEFINED IN java.awt.event; there is one for every interface a listener
- for mouse and keyboard are available in three adapters :

  MouseAdapter implement MouseListener

  MouseMotionAdapter implement MouseMotionListener

  KeyAdapter implement KeyListener
- Examole:

```
import java.awt.event.*;
class MyMouseListenerClass extends MouseAdapter {
   public void mousePressed(MouseEvent e) {
   // Instructions for the management of the pressure the mouse button }
   public void mouseReleased(MouseEvent e) {
   // Instructions for the management of the issue of mouse button
   }}
```

SOftEng

Management of the events in applet

Modify the applet executing the following steps:

- Import java.awt.event
- Specify as interfaces for the management of events will be implemented

```
public class MyApplet extends java.applet.Applet implements
    MouseListener {
    ...
}
```

- you define all methods of each interface (for the methods which are not interested, you must specify the method with the body empty)

```
public void mouseClicked(MouseEvent e) { }
public void mousePressed(MouseEvent e) {
  // Instructions for the management of the pressure of the mouse
}
public void mouseReleased(MouseEvent e) { }
public void mouseEntered(MouseEvent e) { }
public void mouseExited(MouseEvent e) { }
```

SOftEng

Registration of the listener

- using special methods defined in class component
- For each type of listener there is the corresponding method (examples: addmouselistener (), addmousemotionlistener (), addkeylistener ())
  - If was defined a class of separate management of the Events:

    miaclassediascoltomouse ml = New ();
    addmouselistener (ml);
  - If was introduced the management of the events in addmouselistener applet (This);

SOftEng

Management of the mouse

· Events of the mouse belong to classes mouseevent and mousemotionevent, which are sub-classes of inputevent

· The interfaces of the listeners are defined in mouselistener and mousemotionlistener

· The methods used are :

mousePressed() mouseReleased()

mouseDragged() mouseMoved()

mouseEntered() mouseExited()

· The method mouseclicked () it is generated when occur a pressure and subsequent release the mouse button in the same position

· To know the position of the mouse is using the methods include () and gety () defined on the subject event

·To manage Triple and double clicks, using the method getclickcount ()

Management of the keyboard

- Events of the keyboard belong to the class keyevent, subclass of inputevent
- The interface of the listener it is defined in keylistener
- The methods are :
  keyPressed(KeyEvent e)
  keyReleased(KeyEvent e)

Note: the encoding of the keys depends on the platform

- The method keytyped (keyevent) corresponds to typing of a key (pressure followed by issue)
  – Only this method of coding platform-independent of the various characters
- The keys modifiers have occurred with the methods isshiftdown (), iscontroldown (), isaltdown () and ismetadown () defined in class keyevent, returning an boolean

Management of the keyboard

- To manage special keys (example: function keys, pgup) using the virtual keys, class variables DEFINED IN CLASS keyevent
  => Allow to make independent Java code platform (keyboards different can generate different numerical values for the same key)
- It is possible testing them using the method getkeycode () As defined in class keyevent:
  if (e.getKeyCode() == KeyEvent.VK_PAGE_DOWN) {
  // istruzioni gestione tasto PgDn }
- Some of deniti special keys :

Variable class	Key	Variable key	Key
VK_HOME	Home	VK_UP	Freccia in su
VK_END	End	VK_DOWN	Freccia in giu'
VK_PAGE UP	Page Up	VK_LEFT	Freccia a sinistra
VK_PAGE_DOWN	Page Down	VK_RIGHT	Freccia a destra
VK_F1 - VK_F12	Tasti funzione	VK_INSERT	Insert
VK_PAUSE	Tasto pausa	VK_ESCAPE	Escape

Management of the Windows

The system to Windows of AWT is based on primary nesting of components, from external window until you reach the components (normally more simple) Internal .

⇒ It defines a hierarchy of components which determines the provision of the elements on the screen, the order in which are displayed. The components are more important :

- Container: are generic components that contain within them other components. applets are a subclass of panel (containers represented on the screen) that are themselves a subclass of containers .
- Canvas: Are areas dedicated to the representation of images .
- Components of User Interface: Button, list, popup menu, checkbox, text eld, label .
- elements for the construction of the windows: frame, menubar, dialog window .

Management of basic components of the interface towards the User

The procedure for the inclusion of a component in a container is independent of the component considered and consists in :

1. Creation of the component elementary
2. Inclusion in the container which contains the

```
public void init() {
    Button b = new Button("OK");
    add(b);   }
```

- The positioning of the component in the container depends on the definition of the structure (layout) of the container .
- The layout of default is flowlayout with centre alignment. The objects are automatically placed one after another, from left to right, line by line .

Panel

- The positioning of the objects based on absolute coordinates in pixels can give results very different on different screens.
  => The positioning of the objects and is based on
  - Layout of the panel that includes the items
  - order in which these objects are created in the panel

  The five basic layout are: flowlayout, gridlayout, borderlayout, CardLayout, gridbaglayout .
- during initialization of the panel, you select the layout wanted by calling the method setlayout ():

```
public void init() {
    this.setLayout(new FlowLayout());   }
```

- The layout null indicates that the panel must be regarded as a graphical window free (as graphics) editable passing the coordinates

Property of a component

- gridx, gridy. coordinates of the cell (if the component occupies more cells, specify the cell corresponding to the corner in the top left)
- gridwidth, gridheight: number of occupied cells from component (columns for gridwidth, LINES FOR gridheight)
- weightx, weighty: villages of total space (in X and Y) occupied by cell
  - applies zero if the proportions are set elsewhere; indicates to occupy all space is available for the cell
- Fill: determines the direction in which extends the component. None (default): see the component the minimum size

Property of a component

Both: the component extends up to fill the cell in both directions

Horizontal: the component widens in horizontal

Vertical: the component widens in vertical

- Anchor: Position of the component in the cell. Values : CENTER (default), NORTH, NORTHEAST, EAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, NORTHWEST.
- ipadx, ipady: Space around to the component (in x e y)

An example of management of actions

```
import java.awt.*;   import java.awt.event.*;
public class ButtonActionsTest extends java.applet.Applet {
    Button redButton, blueButton, greenButton, whiteButton;
public void init() {
    ButtonHandler bh;
    setBackground(Color.white);
    setLayout( new FlowLayout(FlowLayout.CENTER,10 ,10));
    redButton = new Button("Red");
    bh = new ButtonHandler(this,Color.red);
    redButton.addActionListener(bh);
    add(redButton);
    blueButton = new Button("Blue");
    bh = new ButtonHandler(this,Color.blue);
    blueButton.addActionListener(bh);
    add(blueButton);
    greenButton = new Button("Green");
    bh = new ButtonHandler(this,Color.green);
```


An example of management of actions

```
    greenButton.addActionListener(bh);
    add(greenButton);
    whiteButton = new Button("White");
    bh = new ButtonHandler(this,Color.white);
    whiteButton.addActionListener(bh);
    add(whiteButton);
  }
}
class ButtonHandler  implements ActionListener {
    Color theColor;
    ButtonActionsTest theApplet;
    ButtonHandler(ButtonActionsTest a, Color c) {
        theApplet = a;
        theColor = c;
    }
    public void actionPerformed(ActionEvent e) {
        theApplet.setBackground(theColor);
    }  }
```


Nesting of panels

To use graphical features different (different layout, or different font,. . .) in different areas of the same applet, you create different panel inside the applet.

To create more panel in an applet (itself a subclass of class panel), and `sufficient to add new panel to the panel container, in the same way in which addition to the other items.

```
setLayout(new GridLayout(1,2,10,10);
Panel panel1 = new Panel();
Panel panel2 = new Panel();
add(panel1);
add(panel2);
panel1.setLayout(new FlowLayout() );
panel1.add(new Button("Up");
panel1.add(new Button("Down");
```


Text Area

The TextArea management allows a more complete text fields compared to the component textfield .

Allows you to specify the text on more lines, managing if necessary scrolling text automatically .

The methods builders are:

- TextArea(); Area of size nothing
- TextArea(String); Area of size nothing initialized with the specified text

Text Area

- TextArea(String, int, int); Region with size and text specified initialization
- TextArea(String,int,int,int) : Region with size and text initialization specified. The last parameter describes the state of scrollbar: TextArea.SCROLLBARS BOTH (default): View scrollbar horizontal and vertical
- TextArea.SCROLLBARS HORIZONTAL ONLY: Show only horizontal scrollbar
- TextArea.SCROLLBARS VERTICAL ONLY: Show only scrollbar vertical
- TextArea.SCROLLBARS NONE does not show scrollbar

Text Area

In addition to the many methods applicable to text field, are available:

- getColumns(), getRows(): Returning the number of columns (characters) and lines of text region
- insert(String, int): Insert the text specified in the position date
- replaceRange(String, int start , int end) : replaces the text between the positions specified with the new string

The text area generate the same events of Text Field:

- Events of the selection of selection and deselection
- Event of amendment of the next

Text Area

To manage the events of the selection and deselezione must provide the implementation of methods focusgained () and focuslost ()

to manage events to amend the text it implements the interface textlistener, which contains the method textvaluechanged ().

Scrolling lists

consist of a list of elements, selectable or one at a time (exclusive) or with multiple-choice (nonexclusive).

=> If the number of elements is greater than the number of elements Viewable, a scroll bar and added automatically

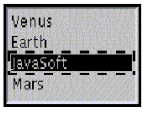
The methods builders are :

- List (); create a list that allows the selection of a single element for time
- List (int): Creates a list with the specified number of visible elements
- List (int, boolean): Creates a list with the specified number of elements and certification (if the second parameter that is true) of multiple selection .

Scrolling list

- After you have created a subject list, you can add elements with the method Add ().
- // 4 lines visible and not abilito multiple selection

```
List lst = new List(4, false);
lst.add("Mercury"); lst.add("Venus");
lst.add("Earth"); lst.add("JavaSoft");
lst.add("Mars");
cnt.add(lst);
```



The methods applicable are similar to those of menu .

Scrolling list

The lists scrolling generate the following events:

- Double-click on an element (event of action)
- Selection and deselezione of an element

To manage the double-click it implements the method actionperformed () In the interface actionlistener for selection.../deselezione it implements the method itemstatechanged (), in the interface itemlistener.

Class itemevent contains the methods getitem (), which gives the element that generated the event, and getstatechange () that describes if it has been selected or deselected.

Scrollbar

It is possible to create scroll bars autonomous, not automatically managed as in lists a scrolling, or in text region .

The methods builders are

- scrollbar (); creates a scroll bar vertical, with a field of minimum and maximum values respectively 0 and 0
- scrollbar (int), creates scrollbar guidance with specicato (scrollbar.horizontal or scrollbar.vertical)
- scrollbar (int, INT, INT, INT, int), and creates scrollbar with guidance, initial position, amplitude of the cursor, minimum and maximum value of scroll bar

Scrollbar

The methods are applicable :

- getMaximum(), getMinimum(), getOrientation(): Return value respectively the minimum or maximum and the guidance of the bar
- getValue(), setValue(): returns or set the current value of the bar

It implements the method adjustmentvaluechanged (), in the interface adjustmentlistener .

The class adjustmentevent includes the method getadjustmenttype (), which describes the type of modification .

Scrollbar: example

```
import java.awt.*;
import java.awt.event.*;
public class ScrollbarTest extends java.applet.Applet implements
    AdjustmentListener {
    Label l;
    public void init() {
        setLayout(new GridLayout(1,2));
        l = new Label("1",Label.CENTER);
        add(l);
        Scrollbar sb = new Scrollbar(Scrollbar.HORIZONTAL,0,0,1,100);
        sb.addAdjustmentListener(this);
        add(sb);
    }
    public void adjustmentValueChanged(AdjustmentEvent e) {
        int v = ((Scrollbar)e.getSource()).getValue();
        l.setText(String.valueOf(v));
        repaint():
    }
}
```


ScrollPane

The scroll bread (sliding panels) are containers in which you can define a single component:

- If the component and largest of the panel that contains, the panel has scrollbar, that allow you to move a "window mobile" on a component, so that we can see all the parties
- The scrolling and managed by

  the methods AWT builders are :
- ScrollPane (); create a panel in which the scrollbar are automatically added if the internal component great there is more of the panel

Scroll Pane

- ScrollPane(int); Create a panel in which the state of scrollbar is determined by the argument, that takes values ScrollPane.SCROLLBARS_ALWAYS: The scrollbar are always present

  ScrollPane.SCROLLBARS_AS_NEEDED: The scrollbar are displayed when is it necessary to see the full component daughter

  ScrollPane.SCROLLBARS_NEVER: The scrollbar are never present

Scroll Pane

To create a Scroll Pane:

```
ScrollPane scroller = new ScrollPane();
Panel panel1 = new Panel();
scroller.add(panel1);
add(scroller);
```

The methods are applicable :

- getScrollPosition(): Returns an object point that is, within the component daughter, the position of the angle in the top left of Scroll Pane
- setscrollposition (int, int), setscrollposition (point): flows the panel until the position specified
- getViewportSize(): Returns an object dimension which is the size of the window display of Scroll Pane

Cursor

The cursor is the image that represents the mouse-pointer (arrow, hand, hourglass,. . .). It is possible to add a cursor to any component and modify it at any time .

The method manufacturer is :

- Cursor (int): create a cursor, the type and determined by parameter that can take the following values :
  - Cursor.DEFAULT_CURSOR: Cursor default (usually the arrow)
  - Cursor.CROSSHAIR_CURSOR: Cursor sign plus
  - Cursor.HAND_CURSOR: Cursor hand
  - Cursor.TEXT_CURSOR Cursor to 'i' for the inclusion of the text

Cursor

- Cursor.WAIT_CURSOR: Indicates that being a very long (hourglass or watch)
- Cursor.MOVE_CURSOR: shows that it is a shift in course of an object
- Cursor.N_RESIZE_CURSOR, Cursor.NE_RESIZE_CURSOR,
- Cursor.E_RESIZE_CURSOR, Cursor.SE_RESIZE_CURSOR,
- Cursor.S_RESIZE_CURSOR, Cursor.SW_RESIZE_CURSOR,
- Cursor.W_RESIZE_CURSOR,
- Cursor.NW_RESIZE CURSOR: Indicate that it is in course the resizing a window

Cursor

Available methods :

- setCursor(Cursor): Set the cursor
- getCursor(): Return the current cursor
- getPredefinedCursor(): returns the type of cursor default
- To create a cursor :

  Cursor cur = new Cursor(Cursor.HAND_CURSOR);

  setCursor(cur);

Component

The class component is the origin of the hierarchy of AWT. Has methods that allow you to modify the appearance of any component

- getBackground(), getForeground(): Returns an object color that is the background color or of the first floor of the component
- setBackground(Color), setForeground(Color): Set the background color or of the first floor of the component
- getFont(); setFont(Font): return the font corrente (in un oggetto Font); Set the font of the component

Component

- getSize() : Returns an object dimension which is the size of the component (width and height are obtained from variables of instance width and height)
- getMinimumSize(): Returns an object dimension which represents the returns in a subject dimension the minimum size of the component (used by the operators of layout). should be redefined for personalized components .
- getPreferredSize() : return in a subject dimension The dimension "ideal" of the component
- setSize(Dimension): Door the size of the component to that in the past as a parameter

Component

- contains(int,int) : Returns True if the coordinates x, y specified are inside of the component
- setVisible(boolean): With parameter false hides the component, with parameter true makes it visible
- isVisible(): Returns True if the component that is visible, FALSE if it is hidden
- setEnabled(boolean): Enables the management of the events for that component with parameter true .
- isEnabled(): Returns True if the component that is empowered, FALSE if it is disabled

A complete example

```
import java.awt.*;
public class ColorTest extends
    java.applet.Applet {
    ColorControls RGBcontrols, HSBcontrols;
    Canvas swatch;
    public void init() {
    setLayout(new GridLayout(1,3,10,10));
    swatch = new Canvas(); // L'area in cui
          // presentare il colore
    swatch.setBackground(Color.black);
    // Il pannello di controllo
    RGBcontrols = new ColorControls(this,
          "Red", "Green", "Blue");
    HSBcontrols = new ColorControls(this,
          "Hue", "Saturation", "Brightness");
    add(swatch);
    add(RGBcontrols);
    add(HSBcontrols);
    }
```

```
void update(ColorControls in) {
   Color c;
   int v1 = Integer.parseInt(in.f1.getText());
   int v2 = Integer.parseInt(in.f2.getText());
   int v3 = Integer.parseInt(in.f3.getText());
   if (in == RGBcontrols) { // converto a RGB
     c = new Color(v1,v2, v3);
     float[] HSB= Color.RGBtoHSB( c.getRed(),
c.getGreen(),c.getBlue(), (new float[3]));
     HSB[0] *= 360;
     HSB[1] *= 100;
     HSB[2] *= 100;
     HSBcontrols.f1.setText(String.valueOf((int)HSB[0]));
     HSBcontrols.f2.setText(String.valueOf((int)HSB[1]));
     HSBcontrols.f3.setText(String.valueOf((int)HSB[2]));
   }
   else { // converto a HSB
     c=Color.getHSBColor((float)v1/360, (float)v2/100,
           (float)v3/100);
     RGBcontrols.f1.setText(String.valueOf(c.getRed()));
     RGBcontrols.f2.setText(String.valueOf(c.getGreen()));
     RGBcontrols.f3.setText(String.valueOf(c.getBlue()));
   }
   swatch.setBackground(c);    swatch.repaint();
   }    }
```

SOftEng

A complete example

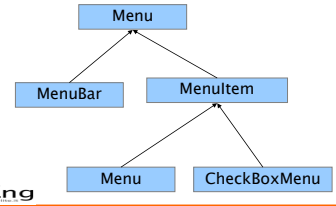
```
import java.awt.*;
import java.awt.event.*;
class ColorControls extends Panel implements
    FocusListener,ActionListener {
    TextField f1, f2, f3;
 ColorTest outerparent; //permette la notifica
          //di update dello schermo alla
    applet
    ColorControls(ColorTest target, String l1,
    String l2, String l3) {
    outerparent = target;
    setLayout(new GridLayout(3,2,10,10));
    f1 = new TextField("0");
    f2 = new TextField("0");
    f3 = new TextField("0");
    add(new Label(l1, Label.RIGHT));
    f1.addFocusListener(this);
    f1.addActionListener(this);
    add(f1);
    add(new Label(l2, Label.RIGHT));
    f2.addFocusListener(this);
     f2.addActionListener(this);
     add(f2);
    add(new Label(l3, Label.RIGHT));
    f3.addFocusListener(this);
    f3.addActionListener(this);
    add(f3);
}
public void focusGained(FocusEvent e) {
}
public void focusLost(FocusEvent e) {
    outerparent.update(this);
}
public void actionPerformed(ActionEvent
e) {
    if (e.getSource() instanceof TextField)
       outerparent.update(this);
}    }
```

SOftEng

MenuBar

are available the following types of menu:

- menus, contained in a menu bar prepared in the upper part of the window
- popup menu, which may appear at any point of applet or applications
- The AWT provides the following classes for the management of menu :



SOftEng

MenuBar

Each window has a bar containing the menu commands.
To create the bar of commands

```
MenuBar mb = new MenuBar();
win.setMenuBar(mb); // method defined in the class frame
```

- To add the bar menu commands

```
Menu m = new Menu("File");
mb.add(m);
```

- To specify the menu dihelp

```
Menu hm = new Menu("Help");
mb.add(hm);
mb.setHelpMenu(hm); // Menu of help to the right in fund
```

- To enable or disable a menu

```
m.enable();
m.disable();
```

SOftEng

Menu

You can define how menu

- **voci normali**, As instances of class menuitem

```
Menu m = new Menu("Tools");
m.add(new MenuItem("Info"));
m.add(new MenuItem("Colors"));
```

- menu entries in two states (toggle), as instances of class CheckboxMenuItem

```
CheckboxMenuItem coords = new
CheckboxMenuItem("Mostra coordinate");
m.add(coords);
```

SOftEng

Menu

- **sub-menu**

```
Menu sb = new Menu("Sizes");
m.add(sb);
sb.add(new MenuItem("Small"));
sb.add(new MenuItem("Medium"));
sb.add(new MenuItem("Large"));
```

- **seperator**

```
MenuItem separator = new MenuItem("-");
m.add(separator);
```

SOftEng

actions arising from the menus

The selection of a menu entry generates an event of action
⇒ it redefines the method **actionPerformed**()

```
public void actionPerformed(ActionEvent e){
    if (e.getSource() instanceof MenuItem) {
        String label = ((MenuItem)e.getSource()).getLabel();
        if (label.equals("view details "))
            toggleCoords();
        else if (label.equals("Fill"))
        fillcurrentArea();
    }
}
```

=> Since checkboxmenuitem and `` a subclass of menuitem, it is not necessary to treat the voices in two states in a different way from the other.

SOftEng

The popup menu

the popup menu is displayed in response to events of the mouse
=> It is possible to create context sensitive menus (to the component on which it was created the event of the mouse)

To create a popup menu
- You create an instance of the class new :

```
PopupMenu pm = new PopupMenu("Edit");
```

- There are the items such as in the case of normal menu

```
pm.add(new MenuItem("Cut"));
pm.add(new MenuItem("Copy"));
pm.add(new MenuItem("Paste"));
```

- is added to the menu to the component

```
add(pm);
```

SOftEng

The popup menu

To show a popup menu
- using the method processmouseevent () of the class component that allows to manage events of the generic mouse
- using the method ispopuptrigger () of Class mouseevent to recognize the request of the popup menu
- Use the show () method of class to show the new menu

```
public void processMouseEvent(MouseEvent e){
  if (e.isPopupTrigger())                    pm.show(
e.getComponent(), e.getX(), e.getY());
super.processMouseEvent(e);
}
```

SOftEng

Example: window with menu

```
import java.awt.*;
class MyFrame extends Frame {
  TextField td;
  Label l;
  String msg = "This is a window";
  MyFrame(String title) {
  super(title);
  setLayout(new BorderLayout());
  l = new Label(msg, Label.CENTER);
  l.setFont(new
Font("Helvetica",Font.PLAIN,12));
  add("Center", l );
  // make dialog for this window
  td = new TextDialog(this, "Enter Text",true);
  td.setSize(150,100);
  // button for showing Dialog
  Button b = new Button("Enter Text");
  MyFrameAction ha = new
MyFrameAction(this);
  b.addActionListener(ha);
  add("South",b);
```

```
MenuBar mb = new MenuBar();
Menu m = new Menu("Colors");
MenuItem i = new MenuItem("Red");
i.addActionListener(ha);
m.add(i);
i = new MenuItem("Green");
i.addActionListener(ha);
m.add(i);
i = new MenuItem("Blue");
i.addActionListener(ha);
m.add(i);
m.add(new MenuItem("-"));
CheckboxMenuItem c =
    new CheckboxMenuItem("Bold");
c.addActionListener(ha);
m.add(c);
mb.add(m);
setMenuBar(mb); setSize(300,200);
setVisible(true);
}
public static void main(String
args[]) {
new MyFrame("My Frame");
}
```

SOftEng

Example: window with menu

```
class MyFrameAction implements ActionListener {
MyFrame theWin;
MyFrameAction(MyFrame win)  {
theWin = win;
}
public void actionPerformed(ActionEvent e) {
if (e.getSource() instanceof Button)
    theWin.td.show();
else if (e.getSource() instanceof MenuItem) {
   String label = ((MenuItem)e.getSource()).getLabel();
   if (label.equals("Red"))
     theWin.l.setbackground(Color.red);
   else if (label.equals("Green"))
     theWin.l.setbackground(Color.green);
     else if (label.equals("Blue"))
      theWin.l.setbackground(Color.blue);
         else if (label.equals("Bold")) {
      if (theWin.l.getFont().isPlain())
         theWin.l.setFont(new Font("Helvetica", Font.BOLD, 12));
      else     theWin.l.setFont(new Font("Helvetica", Font.PLAIN, 12));
          }     }     }     }
```

SOftEng

Use of the windows in applications

- The main class of an application must be a subclass of frame
  Class MyAppAWT extends Frame implements Runnable
- Inside of the method main () The application you create an instance of the class
  =>a normal window
- In the method manufacturer of myappawt to set the characteristics of the window

```
import java.awt.*;
class MyAppAWT extends Frame {
    MyAppAWT(String title) {
        super(title);
        add(new Button("OK"));
        add(new Button("Cancel"));
    }
    public static void main(String args[]) {
        MyAppAWT a = new MyAppAWT("This is an application");
        a.setSize(300,300);
        a.show();  }    }
```

SOftEng

Use of the windows in applications

• You must manage the event of closing the window: hides or destroys the window and recalls the method system.exit (0) to submit to the System and `leaving the application
• the main frame must implement the interface cu ic'e windowlistener in `the method called windowclosing when you close a window .

```
public void windowClosing(WindowEvent e) {
win.setVisible(false);
win.destroy();
System.exit(0);
}
```

SOftEng

Library javax.swing

- extensions in the graphic libraries from Java-Version 1.2
- Many classes include graphic interface of the corresponding class AWT but changes the creation and there are new methods
- La classe swing di solito aggiunge una 'J' davanti al nome della corrispondente classe AWT (es: JFrame, JButton, JLabel, …)
- Look and Feel selectable (stile Java,Windows,Mac)
- The use of jcomponent simplifies the management of the events of the keyboard ;
- Container nested: both more "heavy" (jwindow, jframe, jdialog and japplet) that the more "light" (jinternalframe and jcomponent) delegate the operations to a jrootpane. Any combination that is permitted .

SOftEng

Library javax.swing

- Windows dialog customizable
- JOptionPane, JFileChooser,JColorChooser
- JTable and Jtree
- JEditorPane To edit text fonts with different
- Management of undo

SOftEng

Example with Swing

```
import java.awt.*; import java.awt.event.*;import javax.swing.*;
public class SwingUI extends JFrame implements ActionListener,
WindowListener {
   JLabel text, clicked;   JButton button, clickButton;   JPanel panel;
private boolean clickMeMode = true;
   public SwingUI() {       //Begin Constructor
      text = new JLabel("I'm a Simple Program");
      button = new JButton("Click Me");
      button.addActionListener(this);
addWindowListener(this);
      panel = new JPanel();     panel.setLayout(new BorderLayout());
      panel.setBackground(Color.white);     getContentPane().add(panel);
      panel.add(BorderLayout.CENTER, text);
      panel.add(BorderLayout.SOUTH, button);    } //End Constructor
```

SOftEng

Example with swing

```
public void actionPerformed(ActionEvent event){
      Object source = event.getSource();
      if (clickMeMode) {
        text.setText("Button Clicked");
        button.setText("Click Again");
        clickMeMode = false;
      } else {
        text.setText("I'm a Simple Program");
        button.setText("Click Me");
        clickMeMode = true;
      }  }
}
```

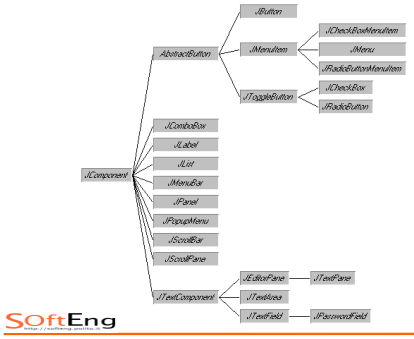
SOftEng

Example with swing

```
public static void main(String[ ] args){
    SwingUI frame = new SwingUI();
    frame.setTitle("Example");
    frame.pack();
    frame.setVisible(true);
 }
public void windowClosing(WindowEvent e) {
       System.exit(0);    }
public void windowActivated(WindowEvent e) { }
public void windowClosed (WindowEvent e) { }
public void windowDeactivated(WindowEvent e) { }
public void windowDeiconified(WindowEvent e) { }
public void windowOpened(WindowEvent e) { }
public void windowIconified(WindowEvent e) { }
}
```

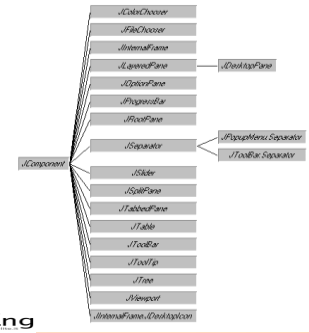
SOftEng

Hierarchy of swing (similar to AWT)



SOftEng

Hierarchy swing (New components)



SOftEng