Capa de Presentación

- Java 2 con JFC/SWING
- Componentes visuales
- Construcción de la interfaz
- Gestión de la interfaz

Java 2 JFC/Swing

- JFC (Java Foundation Class) - Framework gráfico que proporciona herramientas para construir interfaces gráficas de usuario (GUI)

- Java proporciona clases para conseguir:
  - Programación de interfaces fáciles y rápidas
  - Programación de applets para la web

- Swing es el nombre del proyecto que desarrolló los últimos componentes que se añadieron a JFC, que dio lugar a la API

- Clases AWT (Abstract Windowing Toolkit)
- Clases SWING: posteriores a AWT, más portables y funcionales

AWT vs Swing

- AWT está implementado con código nativo, Swing no.
- Comportamiento AWT depende del S.O., el de Swing no.
- Swing da más funcionalidad:
- Los botones y etiquetas pueden mostrar imágenes (+texto).
- Facilidad en la edición de bordes de los componentes.
- Tecnologías asistivas (lector de pantalla), pueden obtener fácilmente información de componentes.
- Permite especificar el aspecto y comportamiento de la IGU.

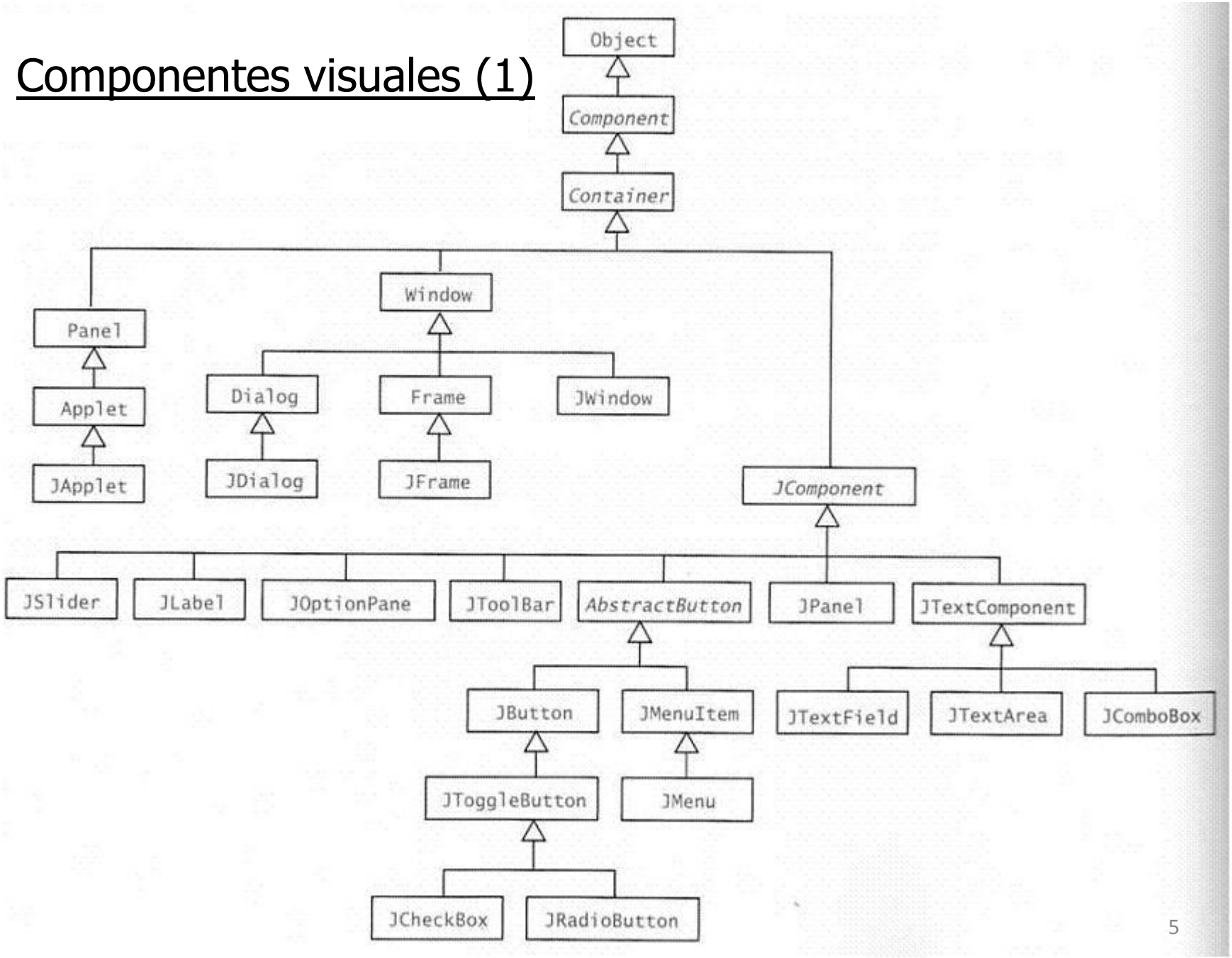
Objetivos

- Entender la jerarquía de clases diseñada en Java que permiten construir interfaces de usuario

- Entender cómo se realiza la gestión de eventos

- No es objetivo aprender los nombres de todas las clases, etc. ya que pueden construirse GUIs usando herramientas visuales (p.e. Jdeveloper, Eclipse Visual Editor)

Componentes visuales (1)



Componentes visuales (2)

- Un contenedor se compone de varios componentes, los cuales pueden ser componentes concretos o pueden ser contenedores.

- **Contenedores** de alto nivel (Window):
  - JFrame: ventana con marco, título, botones, etc.
  - Único contenedor al que se le pueden insertar menús
  - Incluye automáticamente un JPanel
  - JDialog: ventana más limitada que la anterior con un pequeño texto.

- **Contenedores** de propósito general (JComponent):
  - JPanel: contenedor para añadir más componentes
  - JScrollPanel: añade un scroll sobre un componente
  - JToolBar: agrupa diversos componentes en una fila o columna

Componentes visuales (3)

- **Componentes** básicos (JComponent): obtener info. del usuario
  - JButton: botón
  - JComboBox, JList, JMenu: menús desplegables de elementos
  - JCheckBox, JRadioButton: activar/desactivar opciones
  - JTextField, JPasswordField, JTextArea: introducir texto
- **Componentes** con información no editable (JComponent):
  - JLabel: muestra un texto o una imagen no seleccionable
- **Componentes** con información editable (JComponent):
  - JTable: muestra tablas de datos, opcionalmente editables
  - JTextComponent (JTextField, JTextArea): muestra texto, opcionalmente editable
- **Componentes** para confirmación de datos/acciones (JComponent):
  - JOptionPane: muestra una nueva ventana (ligada al padre)

Construcción de la GUI

- Escoger los contenedores de alto nivel (el más usual es JFrame)
- Escoger los contenedores de propósito general, si son necesarios
- Escoger el resto de componentes de la interfaz (JButton, JLabel,...)
- Añadir los componentes a los contenedores y mostrar las interfaces, mediante operadores de añadir y mostrar que proporcionan las clases JFrame, ...

Regla de Diseño

- "Hacer fáciles las cosas simples y hacer posibles las cosas difíciles"

- Principio de menor estupefacción: "No sorprendas al usuario"

Componentes

Clases Button/JButton

- Se usa para construir Botones.
- Al pulsar un botón se generará un evento, que habrá que tratar.

Clases Label/JLabel

- Utilizado para mostrar información.
- Se usan junto a los cuadros de texto.

Componentes

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

public class Botoiak extends JFrame
{
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();
  JButton jButton3 = new JButton();

  public Botoiak() {
    this.setTitle("Botoien adibidea");
    jButton1.setText("Ireki");
    jButton2.setText("Gorde");
    jButton3.setText("Ezeztatu");
    this.getContentPane().add(jButton3, BorderLayout.EAST);
    this.getContentPane().add(jButton2, BorderLayout.CENTER);
    this.getContentPane().add(jButton1, BorderLayout.WEST);
    pack();
  }

 public static void main(String[] args){
    Frame frame = new Botoiak();
    frame.setVisible(true);
 }
}
```

Componentes

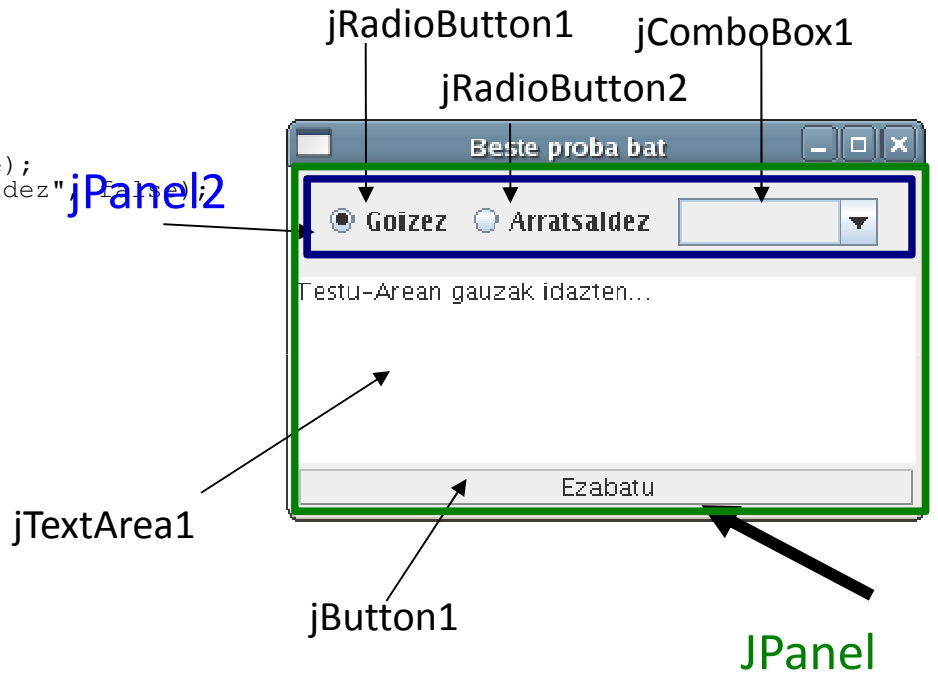
Clases CheckBox/JCheckBox/JRadioButton

- Casillas de verificación. Ofrecen funcionalidad para activar y desactivar opciones.

- Los componentes JRadioButton los agruparemos en un **ButtonGroup** para conseguir seleccionar una (y sólo una) opción del grupo. ButtonGroup no es un componente visual.

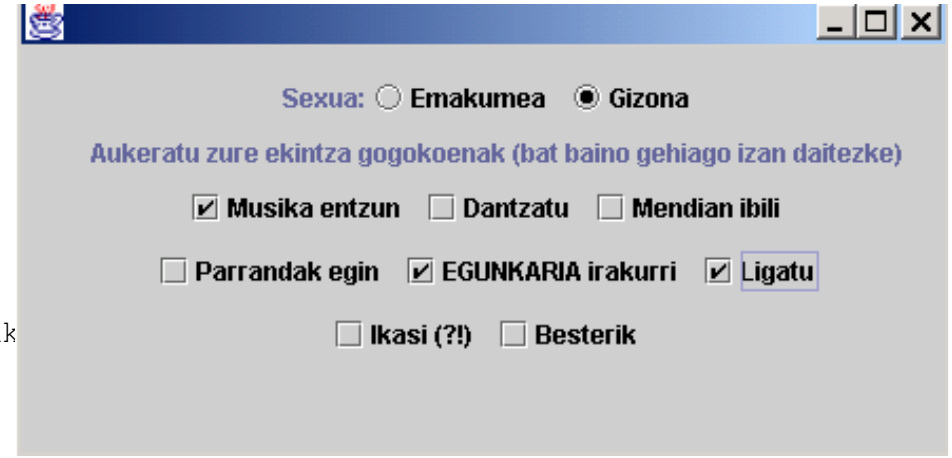
Contenedores

```
public class Complejo extends JFrame {

Button aButton = new Button("Ezabatu");
JPanel jPanel2 = new JPanel();
JTextArea jTextArea1 = new JTextArea();

// Crear botones radiales
JRadioButton goizez = new JRadioButton("Goizez", true);
JRadioButton arratsaldez = new JRadioButton("Arratsaldez");

public Complejo() {

  ...

// Crear agrupación de botones
ButtonGroup aukeraBotoiMultzoa = new ButtonGroup();
aukeraBotoiMultzoa.add(goizez);
aukeraBotoiMultzoa.add(arratsaldez);

// Crear lista desplegable y cargarla de datos
Vector<String> dias_semana = new Vector<String>();
JComboBox jComboBox1 = new JComboBox(dias_semana);

dias_semana.addElement("Astelehena");
dias_semana.addElement("Asteartea");
dias_semana.addElement("Asteazkena");
dias_semana.addElement("osteguna");
dias_semana.addElement("Ostirala");


 jPanel2.add(BorderLayout.EAST, goizez);
 jPanel2.add(BorderLayout.EAST, arratsaldez);
 jPanel2.add(BorderLayout.EAST, jComboBox1);

setSize(320, 200);

// Meter en panel princ. el JPanel2, el área de texto y el botón
getContentPane().add(BorderLayout.NORTH, box);
getContentPane().add(BorderLayout.CENTER, jTextArea1);
getContentPane().add(BorderLayout.SOUTH, aButton);

}
```



Componentes

```
public class Aukerak extends JFrame{
  JPanel jPanel1 = new JPanel();
  JLabel jLabel1 = new JLabel();
  JRadioButton jRadioButton1 = new JRadioButton();
  JRadioButton jRadioButton2 = new JRadioButton();
  ButtonGroup g = new ButtonGroup();
  JLabel jLabel2 = new JLabel();
  JCheckBox jCheckBox1 = new JCheckBox();
  JCheckBox jCheckBox2 = new JCheckBox(); //Besteak
  public Aukerak()  {
    jLabel1.setText("Sexua:");
     jRadioButton1.setText("Gizona");
    jRadioButton1.setSelected(true);
     jRadioButton2.setText("Emakumea");
     jLabel2.setText("Aukeratu zure ekintza gogokoenak (bat baino
    gehiago izan daitezke)");
     jCheckBox1.setText("Musika entzun");
    jCheckBox1.setSelected(true);
     jCheckBox2.setText("Dantzatu");//Besteenak testuak gehitu
     jPanel1.setLayout(new FlowLayout());
    jPanel1.add(jLabel1, null);
     jPanel1.add(jRadioButton2, null);
     jPanel1.add(jRadioButton1, null);
     jPanel1.add(jLabel2, null);
     jPanel1.add(jCheckBox1, null); //Beste JCheckBox-ak gehitu
    g.add(jRadioButton1);
     g.add(jRadioButton2);
     this.getContentPane().add(jPanel1, null);
  }
```

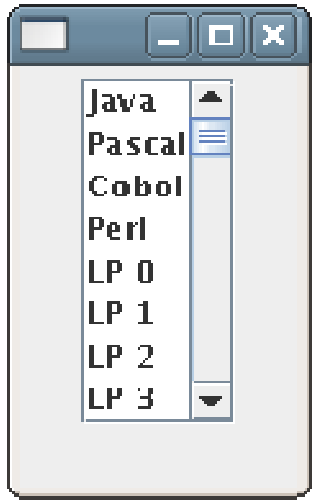


Componentes

Clases List/JList

- Por medio de las listas desplegables, mostraremos al usuario un grupo de opciones. A menudo se usan para evitar la saturación de información en pantalla.

- JList no dispone de scroll por defecto. Para ello, se debe añadir a un JScrollPane.

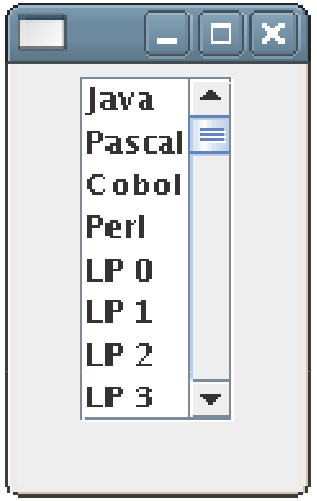


Componentes


```
public class Listak extends JFrame
{
  JList jList1; //new gero egingo da
  Vector elementuak = new Vector();
  JPanel jPanel1 = new JPanel();
  public Listak(){
   this.setTitle("Listen adibidea");
    elementuak.addElement("Java");
    elementuak.addElement("Pascal");
    elementuak.addElement("Cobol");
    elementuak.addElement("Perl");
    jList1 = new JList(elementuak);
   jPanel1.add(jList1, null);
    this.getContentPane().add(jPanel1, null);
    elementuak.addElement("LISP"); // gehitzean, JLIST-a aldatzen da!!
  }
   public static void main(String[] args){
    Frame frame = new Listak();
    frame.setVisible(true);
 }
}
```

Componentes

```
public class ZerrendaScrollekin extends JFrame {
  JPanel jPanel1 = new JPanel();
  JList jList1;
  Vector <String> elementuak = new Vector<String>();
  public ZerrendaScrollekin(){
    this.getContentPane().add(jPanel1, null);
    elementuak.addElement("Java");
    elementuak.addElement("Pascal");
    elementuak.addElement("Cobol");
    elementuak.addElement("Perl");
    jList1 = new JList(elementuak);
    jList1.setVisibleRowCount(8);
 //Lista scroll-a duen panel batean sartzen dugu
    JScrollPane j = new JScrollPane(jList1);
    jPanel1.add(j,null);
    for (int i=0;i<50;i++) elementuak.addElement("LP "+i);
   pack();
   }
   public static void main(String[] args){
    ZerrendaScrollekin frame = new ZerrendaScrollekin();
    frame.setVisible(true);
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   }
}
```



Componentes

- **Clases Choice/JComboBox**
  - Son listas (desplegables) de opciones.
  - Las ventajas de esto: las listas de opciones no ocupan demasiado espacio en pantalla.

Componentes

```
public class AukeraZerrendak extends JFrame {

Vector <String> elementuak = new Vector<String>();
  JPanel jPanel1 = new JPanel();
  JComboBox jComboBox1;//new gero egingo da

  public AukeraZerrendak(){
elementuak.addElement("Java");
    elementuak.addElement("Pascal");
    elementuak.addElement("Cobol");
    elementuak.addElement("Perl");
    jComboBox1 = new JComboBox(elementuak);
    jPanel1.add(jComboBox1, null);
    this.getContentPane().add(jPanel1, null);
    elementuak.addElement("LISP"); // beste item bat gehitzean, JLIST-a
   aldatzen da!!
    pack();
  }
  public static void main(String[] args){
    AukeraZerrendak frame = new AukeraZerrendak();
    frame.setVisible(true);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
 }
}
```

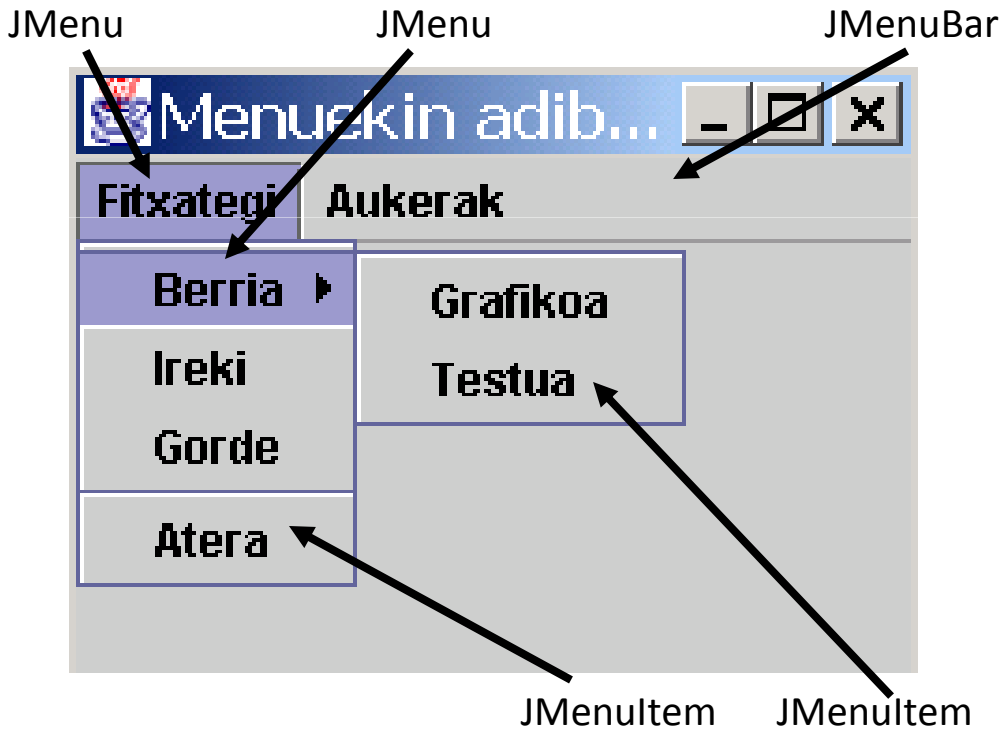
Componentes

Creación de Menús

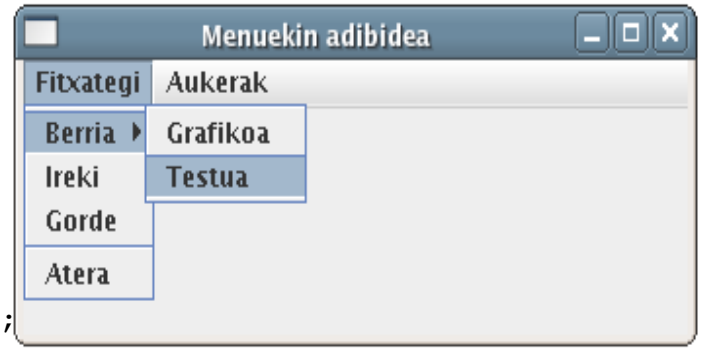
- **En Swing**
  - **El único contenedor que puede alojar una barra de menú es JFrame** .
  - **La clase JMenuBar** crea la barra de menú donde se insertarán las opciones de dicho menú.
  - **La clase JMenu** es la encargada de crear los menús. Estos menús tienen un nombre asociado y muestran una lista desplegable con varios elementos.
  - Los elementos de un menú pueden ser objetos **JmenuItem** u objetos **JMenu** (para crear menús en cascada)

Componentes

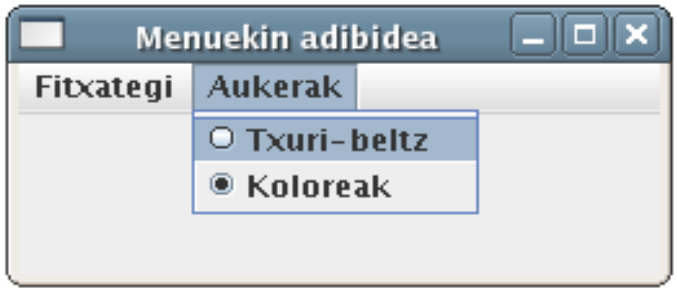
Creación de menús en Swing



Componentes




```
public class Menuak extends JFrame {
  JMenuBar menuBarra = new JMenuBar();
  JMenu fitxategi = new JMenu();  JMenu aukerak = new JMenu();
  JMenu berria = new JMenu(); JMenuItem ireki = new JMenuItem();
  JMenuItem gorde = new JMenuItem(); JMenuItem atera = new JMenuItem();
  JMenuItem koloreak = new JRadioButtonMenuItem();
  JMenuItem txuriBeltz = new JRadioButtonMenuItem();
  ButtonGroup bg = new ButtonGroup();
  JMenuItem testua = new JMenuItem(); JMenuItem grafikoa = new JMenuItem();
  public Menuak() {
this.setJMenuBar(menuBarra); this.setTitle("Menuekin adibidea");
    fitxategi.setText("Fitxategi");    aukerak.setText("Aukerak");
    berria.setText("Berria");    grafikoa.setText("Grafikoa");
    testua.setText("Testua");    ireki.setText("Ireki");
    gorde.setText("Gorde");    atera.setText("Atera");
    koloreak.setText("Koloreak");    txuriBeltz.setText("Txuri-beltz");
    testua.setText("Testua");
    berria.add(grafikoa);    berria.add(testua);
    fitxategi.add(berria); fitxategi.add(ireki);   fitxategi.add(gorde);
    fitxategi.addSeparator();    fitxategi.add(atera);
    menuBarra.add(fitxategi);
    aukerak.add(txuriBeltz);    aukerak.add(koloreak);
    bg.add(txuriBeltz);    bg.add(koloreak);
    menuBarra.add(aukerak);
  }
 }
```



Componentes

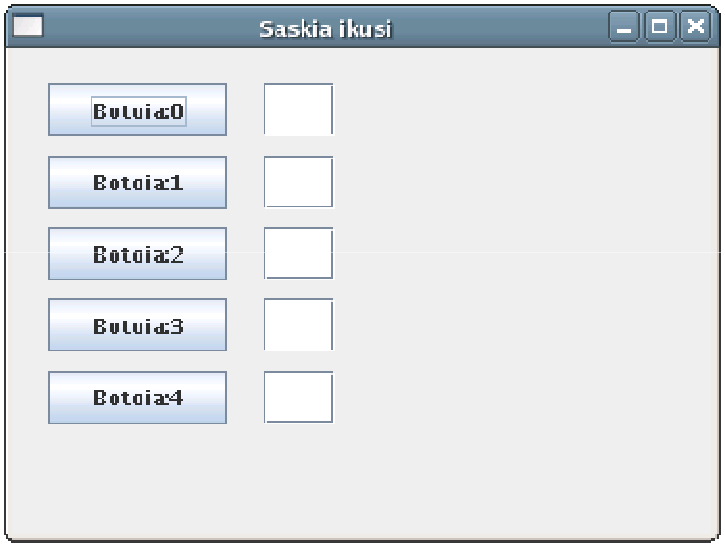
- **Inserción de imágenes**
  - Meteremos la imagen en un JLabel


```
public class FrameIrudiekin extends JFrame {
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();

  public FrameIrudiekin() {
    this.getContentPane().setLayout(null);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Irudiak nola bistaratu");

    JLabel lb1 = new JLabel(new ImageIcon(getClass().getResource("katua.jpg")));
    this.getContentPane().add(lb1);
    lb1.setSize(lb1.getPreferredSize());
    lb1.setLocation(20,20);
    JLabel lb2 = new JLabel(new ImageIcon(getClass().getResource("armiarma.jpg")));
    this.getContentPane().add(lb2);
    lb2.setSize(lb2.getPreferredSize());
    lb2.setLocation(20,120);

    jButton1.setText("Katua erosi");
    jButton1.setBounds(new Rectangle(180, 40, 160, 30));
    this.getContentPane().add(jButton1, null);
    jButton2.setText("Armiarma erosi");
    jButton2.setBounds(new Rectangle(180, 140, 160, 30));
    this.getContentPane().add(jButton2, null);
  }
```

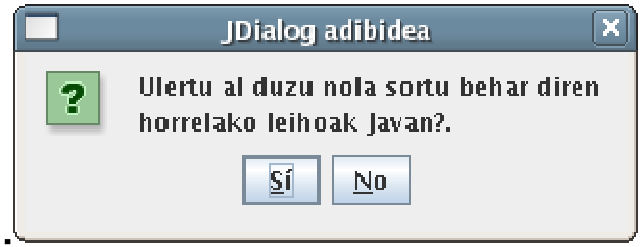
Componentes

- **Creación dinámica de componentes**

```
public class Saskia extends JFrame {
  int N=5;
  JButton[] botoiak= new JButton[N];
  JTextField[] testuKutxak = new JTextField[N];
  public Saskia(){
    for (int i=0; i<N; ++i){
      botoiak[i] = new JButton();
      testuKutxak[i] = new JTextField();
    }
    this.getContentPane().setLayout(null);
    this.setSize(new Dimension(400, 300));
    this.setTitle("Saskia ikusi");
   for (int i=0; i<N; ++i){
      botoiak[i].setBounds(new JButton(20, 20+i*40, 80, 30));
      botoiak[i].setText("Botoia:"+i);
      this.getContentPane().add(botoiak[i], null);
      testuKutxak[i].setBounds(new Rectangle(110, 20+i*40, 40, 30));
      this.getContentPane().add(testuKutxak[i], null);
    }
  }
```

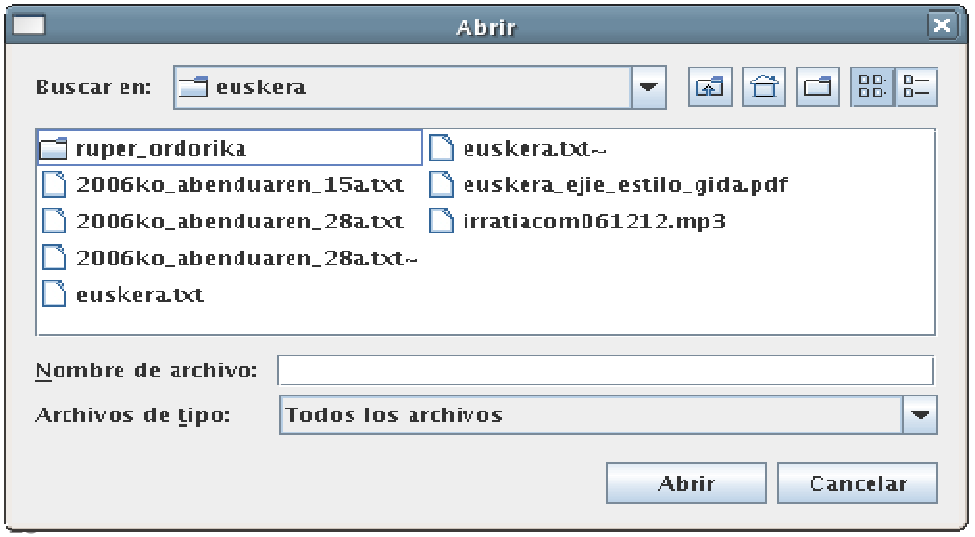


Otros contenedores



- **Clases Dialog/JDialog/JOptionPane**
  - Ventana para leer o mostrar datos del usuario.
  - Si las hacemos MODAL , no podremos cambiar de ventana mientras el diálogo siga activo.

- **Clase JFileChooser** (Swing)



Gestores de Diseño: Layout

- Se usan para definir dónde colocar un componente dentro de un contenedor.
  - `contenedor.add(componente);`
- **FlowLayout** (Gestor predeterminado para Panel)
  - Los componentes se van añadiendo a una línea. Al completar la línea, se pasa a la siguiente.
- **BorderLayout** (Gestor predeterminado para Frame y Dialog)
  - Los componentes se añaden en una de estas 5 zonas: norte, sur, este, oeste y centro.
- Se puede cambiar el gestor predeterminado:
  - `contenedor.setLayout(new BorderLayout());`

Gestores de Diseño: Layout

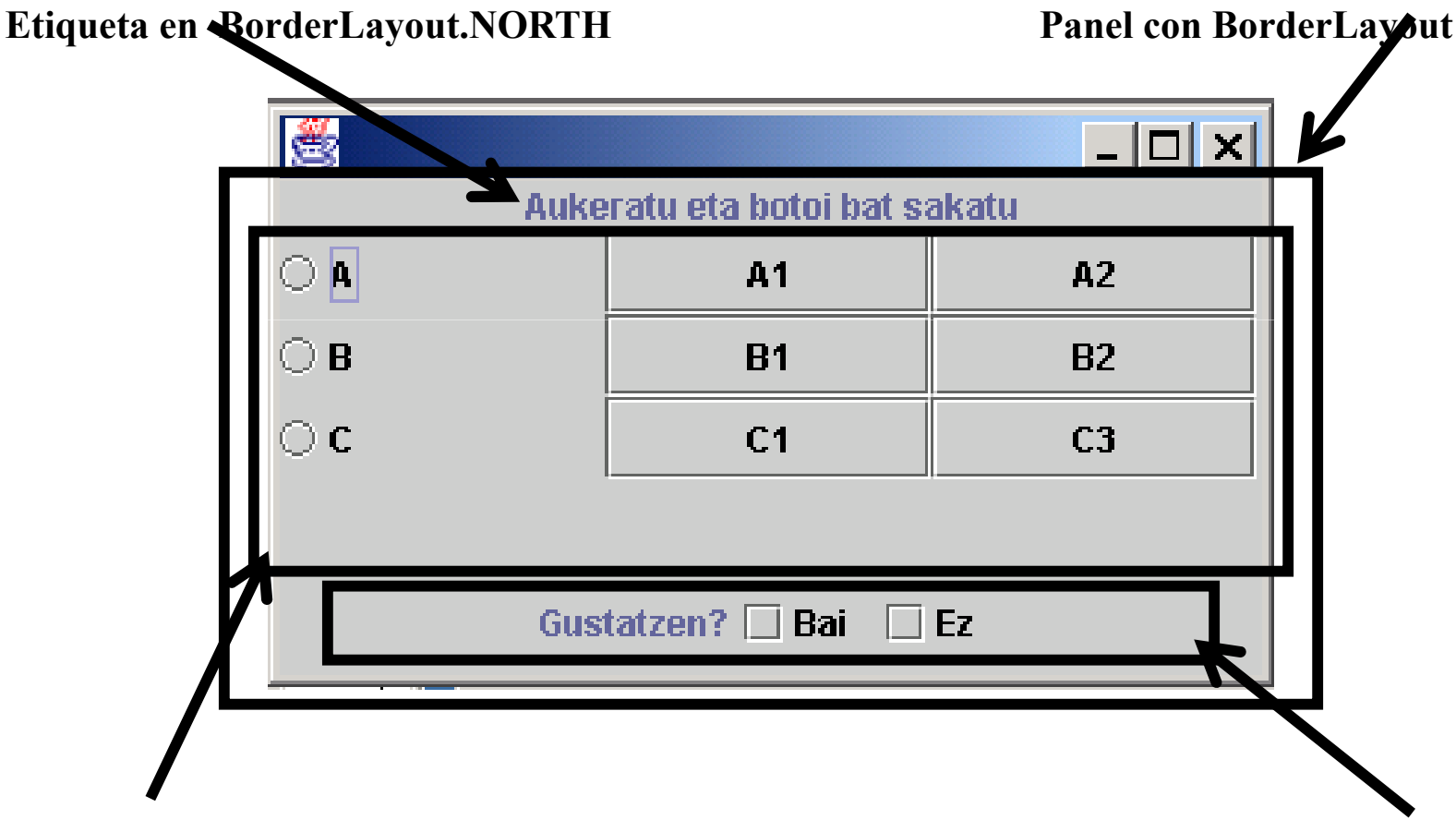
- Si se quiere poner el componente en unas coordenadas concretas, se debe de eliminar el gestor de diseño.

```
contenedor.setLayout(null);
```

- Sean **this** un contenedor y **textField1** uno de sus componentes:

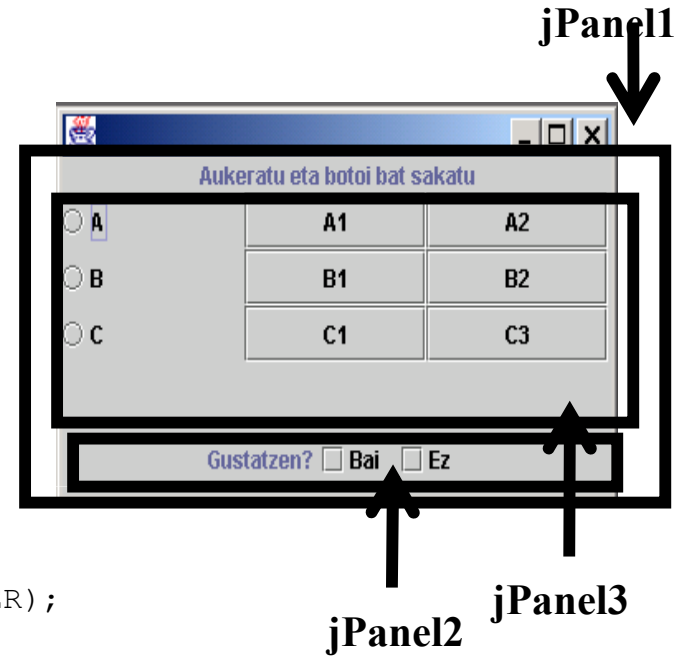
```
setLayout(null);
textField1.setBounds(15,20,50,60);
                          x,y,ancho,alto
```

Gestores de Diseño: Layout



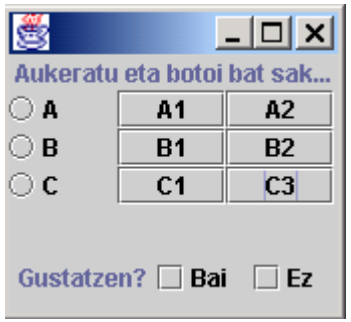
Gestores de Diseño: Layout

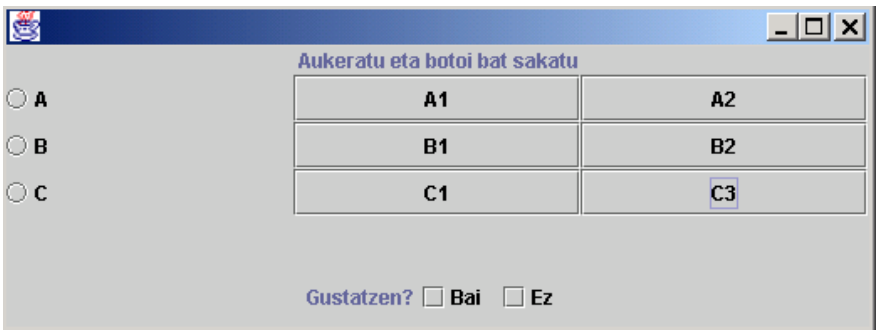
```
public class Layoutak extends JFrame {
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JPanel jPanel3 = new JPanel();
  BorderLayout borderLayout1 = new BorderLayout();
  GridLayout gridLayout1 = new GridLayout(4,3);
  JLabel jLabel1 = new JLabel();//...
  public Layoutak(){
    jPanel1.setLayout(borderLayout1);
    jLabel1.setText("Gustatzen?");
    jCheckBox1.setText("Bai");
    jCheckBox2.setText("Ez");
    jLabel2.setText("Aukeratu eta botoi bat sakatu");
    jLabel2.setHorizontalAlignment(SwingConstants.CENTER);
    jPanel3.setLayout(gridLayout1);
    jRadioButton1.setText("A");//...
    jPanel3.add(jRadioButton1, null);
    jPanel3.add(jButton1, null);
    jPanel3.add(jButton2, null);
    jPanel3.add(jRadioButton2, null);//...
    jPanel2.add(jLabel1, null);
    jPanel2.add(jCheckBox1, null);
    jPanel2.add(jCheckBox2, null);
    jPanel1.add(jLabel2, BorderLayout.NORTH);
    jPanel1.add(jPanel3, BorderLayout.CENTER);
    jPanel1.add(jPanel2, BorderLayout.SOUTH);
    this.getContentPane().add(jPanel1, null);
  }
```

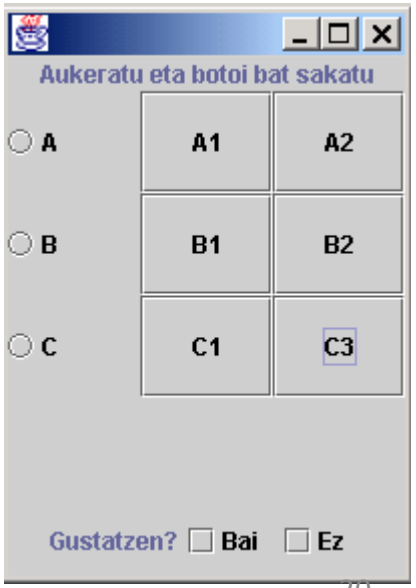


Gestores de Diseño: Layout

- Ventajas de definir un GUI con gestor de diseño:
  - los componentes se redibujan automáticamente al ajustar el tamaño de la ventana (ajustándose al tamaño disponible).



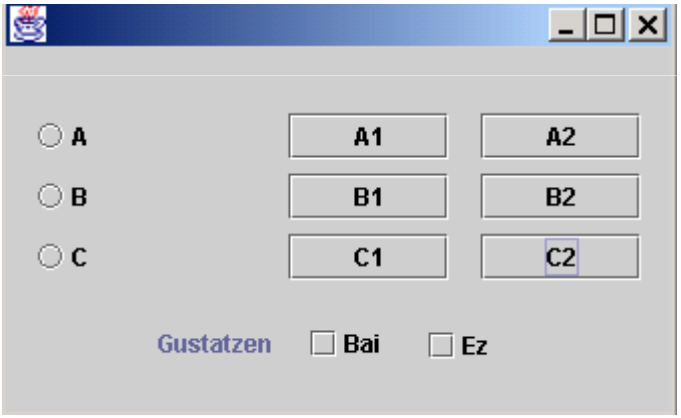




Gestores de Diseño: Layout

Sin gestor de diseño se definen las coordenadas de todos los componentes - fácil de hacer con herramientas visuales.

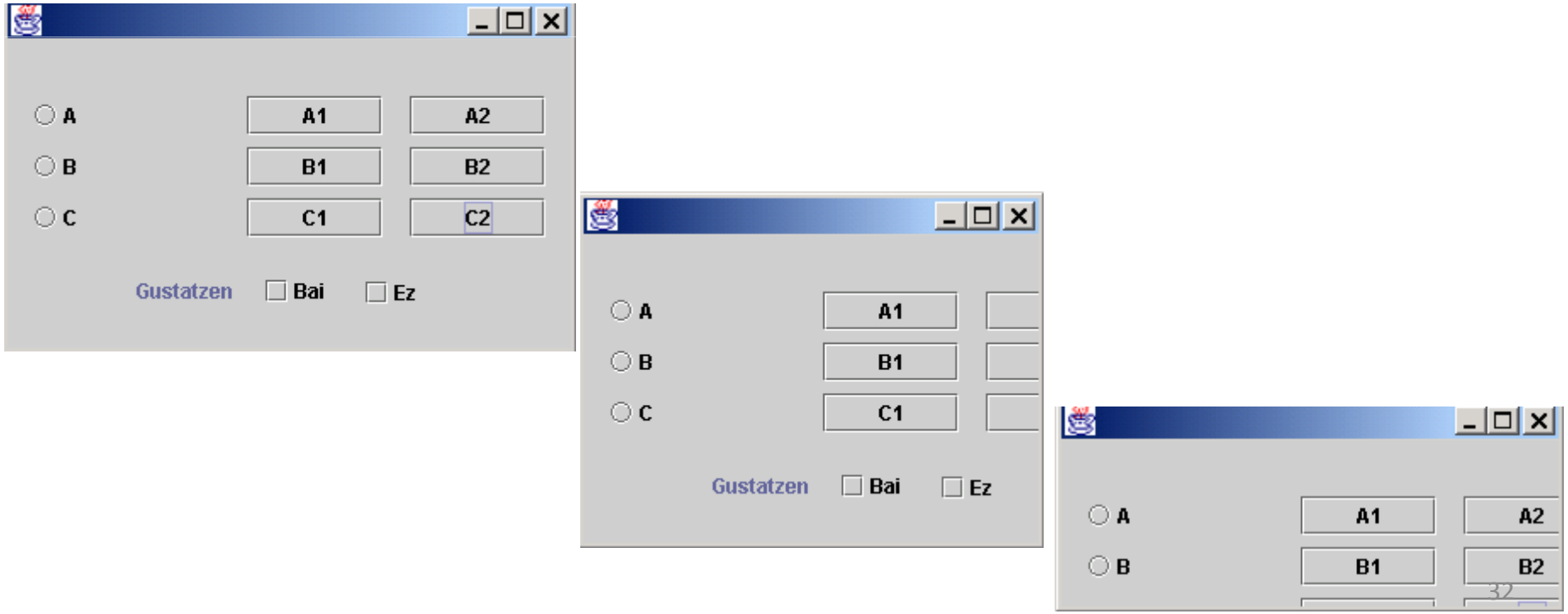
```
jPanel1.setBounds(new Rectangle(3, 0, 333, 170));
jPanel1.setLayout(null);

jLabel1.setText("Gustatzen");
jLabel1.setBounds(new Rectangle(66, 18, 62, 16));
jPanel1.add(jLabel1,null);

jCheckBox1.setText("Bai");
jCheckBox1.setBounds(new Rectangle(142, 17, 48, 19));
jPanel1.add(jCheckBox1,null);
jCheckBox2.setText("Ez");
jCheckBox2.setBounds(new Rectangle(201, 17, 44, 21));

jRadioButton1.setText("A");
jRadioButton1.setBounds(new Rectangle(14, 10, 49, 23));
jRadioButton2.setText("B");
jRadioButton2.setBounds(new Rectangle(14, 40, 58, 23));
jRadioButton3.setText("C");
jRadioButton3.setBounds(new Rectangle(14, 70, 55, 23));
jButton1.setText("A1");
jButton1.setBounds(new Rectangle(139, 10, 80, 23));
//...
```



Gestores de Diseño: Layout

Desventajas de no usar un gestor de diseño:

–Al redimensionar el Frame, los componentes se mantienen sin cambiar su posición (no se ajustan al tamaño disponible).



Documentación Swing:

http://download.oracle.com/javase/tutorial/uiswing/

http://www.programacion.com/java/tutorial/swing

API de JAVA:

http://download.oracle.com/javase/1.5.0/docs/api/

Gestión de la Interfaz

- Al diseñar una interfaz gráfica hay que tener en cuenta que se producirán ciertos eventos del usuario
  - Un evento es un suceso generado por una acción del usuario que afecta a algún componente de la interfaz
  - Por ejemplo: pulsar una tecla, mover el ratón, cambiar el formato de la ventana, cerrar una ventana, pulsar un botón, elegir una opción de un menú, etc.

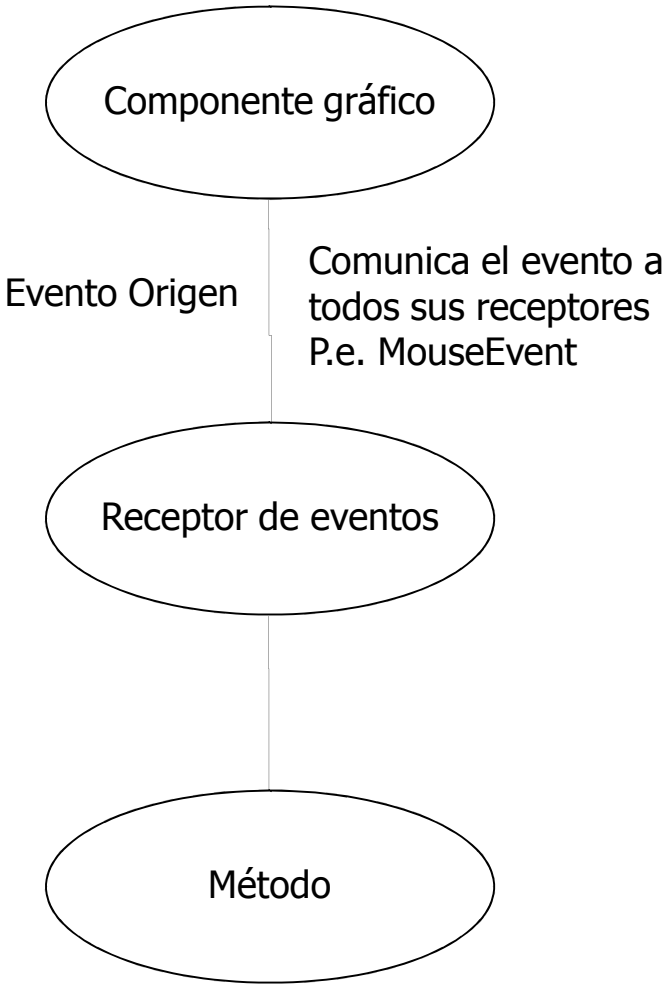
- La implementación deberá contemplar una serie de acciones de respuesta para procesar los eventos del usuario

Modelo de eventos

Sucede un evento que afecta a objetos
P.e. JButton, JFrame

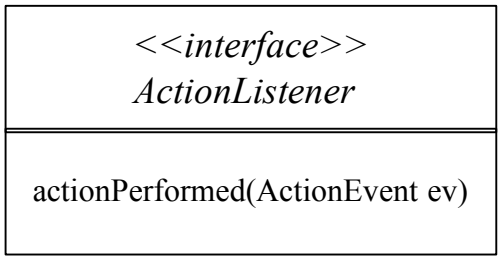
Se crea un objeto evento y se pasa el control al objeto oyente
P.e. MouseListener
Para ser receptor de un evento origen debe registrarse al componente gráfico que lo genera con:
Add<EventType>Listener(receptor)

Método para responder al evento
P.e. mouseClicked(MouseEvent)



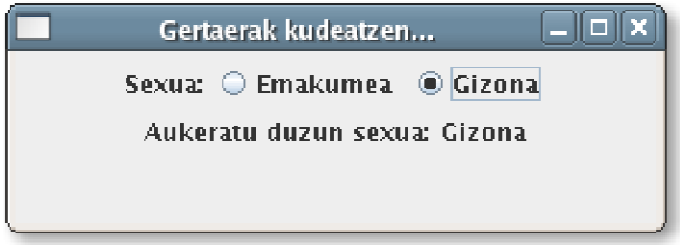
Modelo de Eventos-Implementación

Para estar informados sobre los eventos que ocurran en un botón, debemos implementar una **interfaz listener** *(listener interface)*




```
public class SimpleGUI extends JFrame implements ActionListener {

JButton button;

 public void ekin(){
button = new JButton("sakatu hemen");
button.addActionListener(this);
 }


 public void actionPerformed(ActionEvent e) {
   button.setText("ados! botoia sakatu duzu!");

 }
}
```

Modelo de Eventos-Implementación




```
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.*;

public class Aukerak2 extends JFrame{

   JLabel jLabel1 = new JLabel("Sexua:");
   JLabel jLabel2 = new JLabel("Aukeratu duzun sexua:");
   JLabel emaitza = new JLabel();
   JRadioButton emakumea = new JRadioButton("Emakumea", true);
   JRadioButton gizona = new JRadioButton("Gizona", false);
   ButtonGroup bg = new ButtonGroup();

public Aukerak2() {
   super("Gertaerak kudeatzen...");
}
```

```
public void go(){
  bg.add(emakumea);
  bg.add(gizona);
  emakumea.addActionListener(new GestorEvento());
  gizona.addActionListener(new GestorEvento());
  this.getContentPane().setLayout(new FlowLayout());
  getContentPane().add(jLabel1,null);
  getContentPane().add(emakumea,null);
  getContentPane().add(gizona,null);
  getContentPane().add(jLabel2,null);
  getContentPane().add(emaitza,null);
  setSize(300,200);
  setVisible(true);
}
```

```
public class GestorEvento  implements ActionListener {

  public void actionPerformed(ActionEvent e) {
    emaitza.setText(e.getActionCommand());
  }
}
```

```
public static void main(String[] args){
 Aukerak2 proba = new Aukerak2();
 proba.go();
 proba.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}
}
```

Modelo de Eventos-Implementación

```
public void go(){
  bg.add(emakumea);
  bg.add(gizona);
  emakumea.addActionListener(new
GertaeraKudeatzaile());
  gizona.addActionListener(new GertaeraKudeatzaile());
  this.getContentPane().setLayout(new FlowLayout());
  getContentPane().add(jLabel1,null);
  getContentPane().add(emakumea,null);
  getContentPane().add(gizona,null);
  getContentPane().add(jLabel2,null);
  getContentPane().add(emaitza,null);
  setSize(300,200);
  setVisible(true);
}

public class GertaeraKudeatzaile implements
ActionListener {

  public void actionPerformed(ActionEvent e) {
    emaitza.setText(e.getActionCommand());
  }
}
```

```
public void go(){
bg.add(emakumea);
bg.add(gizona);
emakumea.addActionListener(new ActionListener(){
  public void actionPerformed(ActionEvent e) {
    emaitza.setText(e.getActionCommand());
  }
});

gizona.addActionListener(new ActionListener(){
   public void actionPerformed(ActionEvent e) {
      emaitza.setText(e.getActionCommand());              }
});
this.getContentPane().setLayout(new FlowLayout());
getContentPane().add(jLabel1,null);
getContentPane().add(emakumea,null);
getContentPane().add(gizona,null);
getContentPane().add(jLabel2,null);
getContentPane().add(emaitza,null);
setSize(300,200);
setVisible(true);
}
```

Modelo de eventos

- Eventos de bajo nivel
  - Relacionados con los aspectos físicos de la GUI.
  - Por ejemplo: pulsar una tecla, mover un ratón, ...

- ComponentEvent: mover, modificar la medida de un componente
- ContainerEvent: añadir o borrar componentes de un contenedor
- FocusEvent: un componente recibe/pierde el foco
- KeyEvent: pulsar/soltar una tecla
- MouseEvent: pulsar/solar botones ratón, desplazar el ratón
- WindowEvent: cerrar, activar, minimizar, ... ventanas

Modelo de eventos

- Eventos de alto nivel
  - Relacionados con la semántica del componente y generalmente combinaciones de eventos de bajo nivel
  - Por ejemplo: pulsar un botón, cambiar el texto de un campo, seleccionar un elemento de un menú desplegable, ...

- ActionEvent: pulsar botón, seleccionar menú, pulsar ENTER
- AdjustementEvent: mover la barra de desplazamiento
- ItemEvent: seleccionar entre una lista de opciones
- TextEvent: introducir texto

Modelo de eventos

Componente Gráfico	Evento Origen	Receptor de eventos	Métodos
Jbutton, JtextFIeld...	ActionEvent	ActionListener	actionPerformed(ActionEvent)
Componentes	ComponentEvent	ComponentListener	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
Componentes	FocusEvent	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Componentes	KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)

Modelo de eventos

Componente Gráfico	Evento Origen	Receptor de eventos	Métodos
Componentes	MouseEvent	MouseInputListener	MouseClicked(MouseEvent) MouseEntered(MouseEvent) MouseExited(MouseEvent) MousePressed(MouseEvent) MouseReleased(MouseEvent) MouseDragged(MouseEvent) MouseMoved(MouseEvent)
		MouseListener	MouseClicked(MouseEvent) MouseEntered(MouseEvent) MouseExited(MouseEvent) MousePressed(MouseEvent) MouseReleased(MouseEvent)
		MouseMotionListener	MouseDragged(MouseEvent) MouseMoved(MouseEvent)

Modelo de eventos

Componente Gráfico	Evento Origen	Receptor de eventos	Métodos
Contenedores	ContainerEvent	ContainerListener	ComponentAdded(ContainerEvent) ComponentRemoved(ContainerEvent)
Windows	WindowEvent	WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

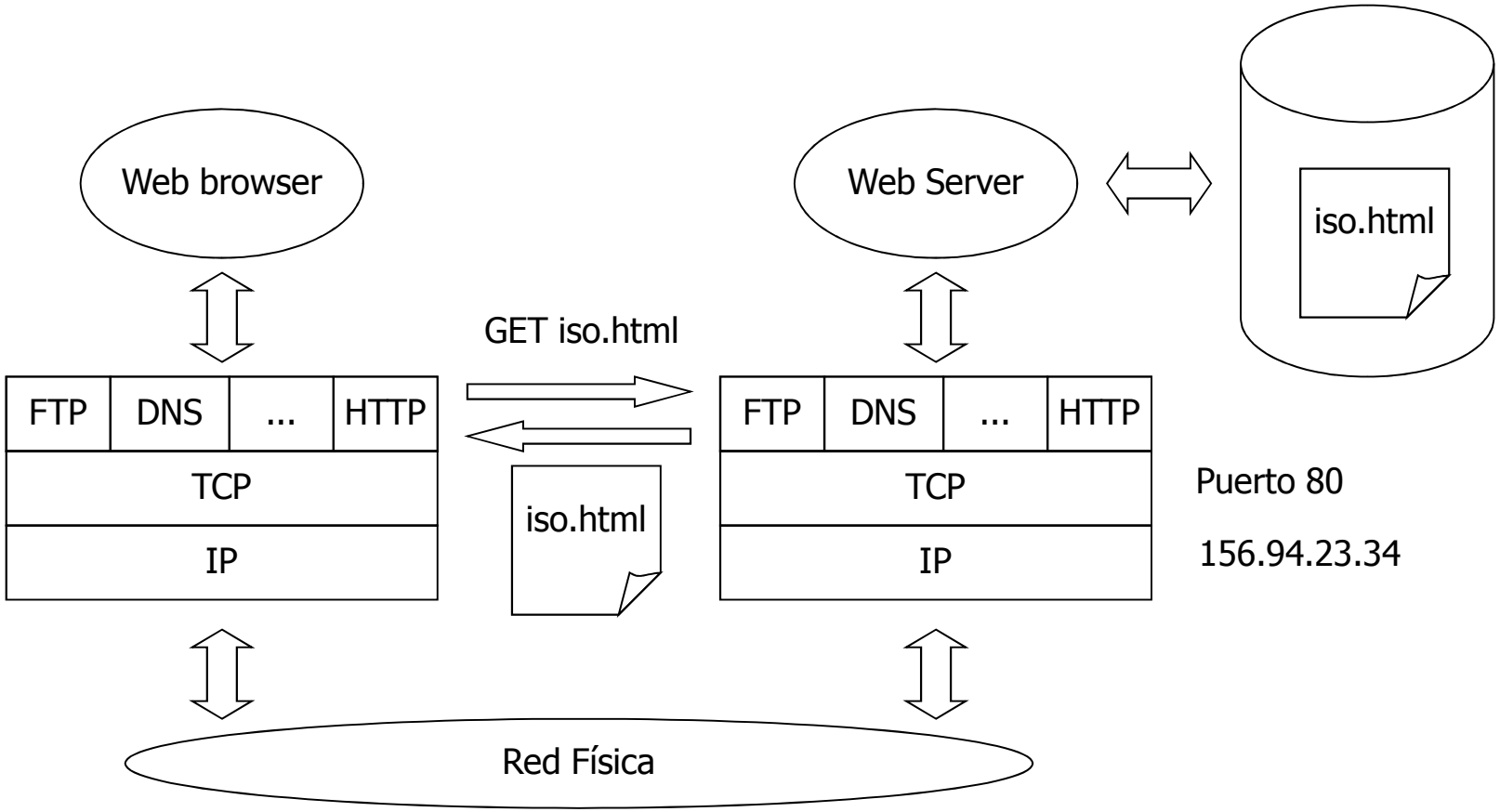
SI Cliente/servidor sobre la web

- World Wide Web
- Aplicaciones estáticas
- Aplicaciones dinámicas web server
- Aplicaciones dinámicas web browser

World Wide Web

- El proyecto WWW (web, w3) comenzó el año 1989 en el CERN de la mano de Tim Berners-Lee. El objetivo inicial era desarrollar un "sistema de hipertexto" que permitiera el intercambio eficiente y fácil de información entre los equipos de investigación, dispersos geográficamente.
- WWW opera sobre el protocolo TCP/IP y comprende las siguientes tecnologías:
  - Web servers
  - Web browsers (navegador)
  - URL (Uniform Resource Locator)
  - HTTP (Hypertex Transfer Protocol)
  - HTML (Hypertex Markup Language)
  - Etc.

World Wide Web



El cliente universal: web browser (WB)

- El WB proporciona una interfaz, normalmente gráfica, para obtener, interpretar, formatear y finalmente presentar documentos –páginas- HTML, con enlaces a otras páginas o documentos, que encapsulan URLs
- Un URL proporciona al WB la información necesaria para obtener un documento de la web
- Otras funcionalidades:
  - Otros protocolos: POP3 y/o IMAP, SMTP, FTP
  - Soporte para extensiones no estándar de HTML: Javascript
  - Plugins
  - Java applets, Controles ActiveX

El servidor: web server (WS)

- El WS es un proceso servidor que "escucha" un puerto TCP, generalmente el 80, esperando a un cliente WB
- Una vez establecida la conexión, el WB envía una petición WS y le devuelve una respuesta, liberando la conexión
- El protocolo que define las peticiones y respuestas legales es HTTP
- Normalmente, los recursos que se solicitan son ficheros accesibles por el WS
- Otras funcionalidades:
  - Control y registro, diario de accesos
  - Protocolos y servicios Internet: FTP, News, etc.
  - Paso de peticiones a otros procesos usando CGI
  - Active Server Pages (ASP) / Servlets

Uniform Resource Locator (URL)

- El URL informa al WB de:
  - En qué máquina está el recurso
  - Qué protocolo se usará para obtener el recurso
  - Cómo localizar el recurso en la máquina servidora

- Protocolo://host[:puerto]/[path]

- Protocolo: http, ftp, file, news, gopher, mailto, telnet, ...

HyperText Transfer Protocol (HTTP)

- HTTP es un protocolo sobre TCP/IP

- Versiones: 0.9, 1.0, 1.1

- Esquema de una sesión HTTP:
  - Se establece una conexión TCP/IP entre el WB y el WS
  - WB realiza una petición al WS
  - WS devuelve una respuesta al WB
  - WB y WS cierran la conexión

- HTTP es un protocolo sin estado: el WS no "recuerda" nada de las sesiones HTTP anteriores

Aplicaciones Estáticas

- HTML
- XHTML
- CSS

HyperText Markup Language (HTML)

- Lenguaje para la publicación de contenidos hypermedia en web
- Basado en SGML (Standard Generalized Markup Language)
- Versiones: 2.0, 3.2, 4.0, 4.01, 5 (en desarrollo)
- Página HTML: contenido + marcaje
- El marcaje permite definir:
  - Cómo se estructura el contenido
  - Cómo se debe presentar el contenido+estructura
  - Referencias a URLs
  - Objetos "incrustados": imágenes, audio, video, scripts, applets, ...
- Las marcas son etiquetas o TAGs:
  - <TAG (atributo=valor)*> Contenido+marcaje </TAG>

Extensible HyperText Markup Language (XHTML)

- XHTML = XML (adaptación de SGML)+HTML
- "Reformulación de HTML 4.0"
- Versiones: 1.0, 2.0
- Normas:
  - Etiquetas en minúsculas
  - Abrir y cerrar etiquetas siempre y no anidarlas
  - Atributos entre comillas
- Ventajas:
  - Compatibilidad (en navegadores + dispositivos)
  - XHTML mal formado da error en el navegador
  - Sólo está accesible código bien escrito
  - Navegadores 'viejos' no fallan si encuentran etiquetas desconocidas (nuevas); XML ignora las etiquetas desconocidas

Cascading Style Sheets (CSS)

- Separan estructura y presentación
- Definen la presentación de documentos HTML, XML y XHTML
- Información de estilo en el mismo documento o por separado
- Ventajas:
  - Control centralizado de un sitio web completo
  - Facilidades de modificación
  - Mayor accesibilidad (los navegadores permiten especificar al usuario su propia hoja de estilo local)
  - Diferentes necesidades, diferentes hojas de estilo: pc, netbook, móvil, impresión, sintetizador de voz,...

Aplicaciones Dinámicas

- Common Gateway Interface (CGI)
- Formularios HTML
- Javascript
- Servlets
- Java Server Pages (JSP)
- Applets

SI Cliente/Servidor sobre la web

- Aplicaciones estáticas: muestran documentos HTML disponibles en el WS

- Aplicaciones basadas en WS: usan WB para obtener y mostrar información HTML y/o rellenar formularios generados por procesos servidores que interaccionan con los WB via los WS: CGI

- Aplicaciones basadas en WB: Java applets que se ejecutan en la MVJ del WB

- Aplicaciones Cliente/Servidor OO: Java applets que son clientes de objetos CORBA

Common Gateway Interface (CGI)

- Permite al WB solicitar datos de un programa externo ejectuado en un WS.

- CGI es un protocolo que define como el WB puede interaccionar mediante el mecanismo HTTP, con programas (llamados CGIs) situados en un WS

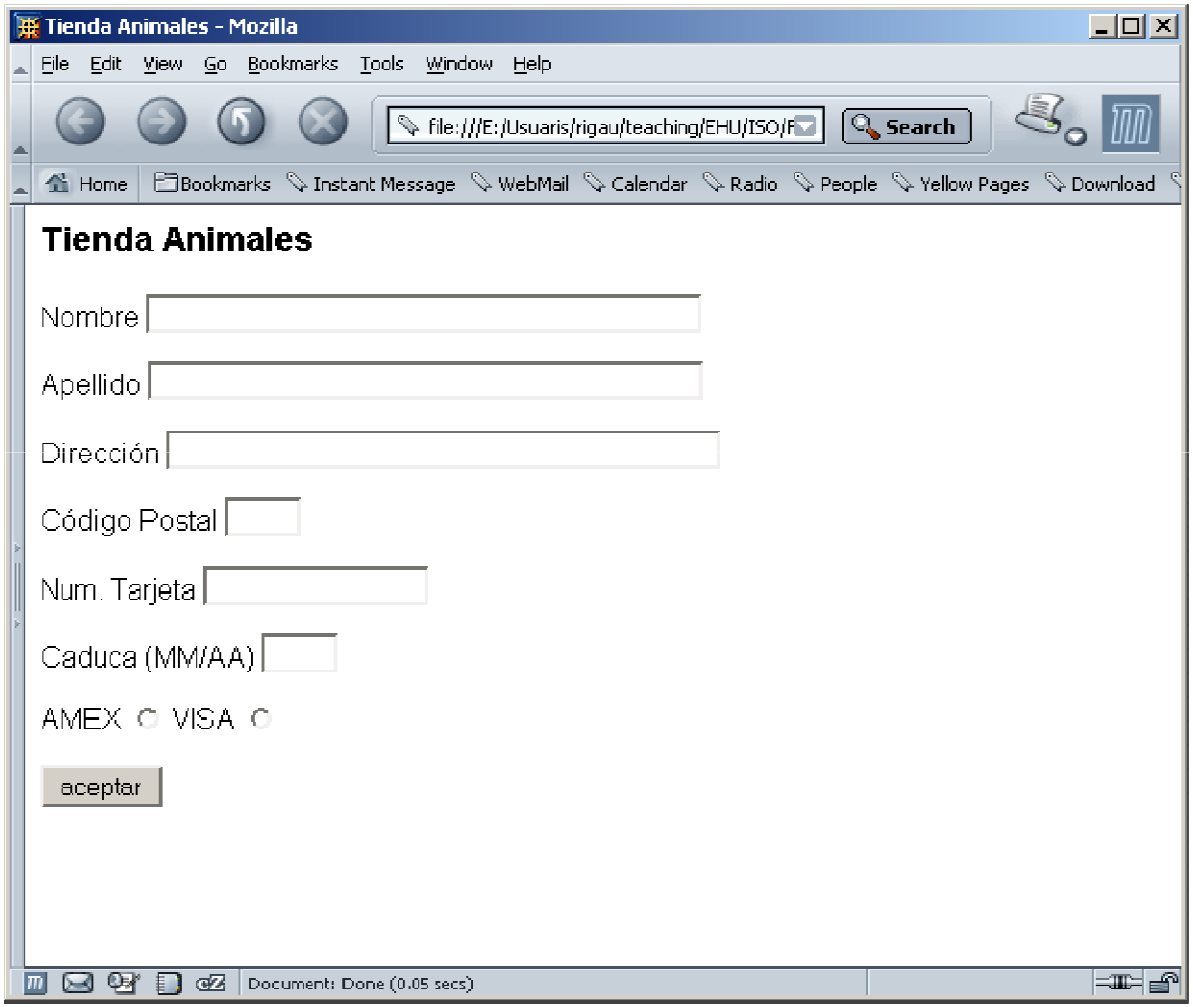
- Los programas CGI pueden ser:
  - Scripts (+portabilidad): shell, perl, phyton, ...
  - Ejecutables: C, C++, Ada, ...

- Suelen estar accesibles en un directorio especial en el WS:
  - www.lawebquesea.com/cgi-bin/loscgis

Formularios HTML

```
<html><head><title>Tienda Animales</title></head>
<body><h3>Tienda Animales</h3>
<form action='http://www.tienda.es/cgi-bin/tienda.pl' method=post>
Nombre <input name='nombre' size=50><p>
Apellido <input name='apellido' size=50><p>
Dirección <input name='direccion' size=50><p>
Código Postal <input name='cp' size=5><p>
Num. Tarjeta <input name='nt' size=19><p>
Caduca (MM/AA) <input name='ft' size=5><p>
AMEX <input name='cc' type=radio value='amex'>
VISA <input name='cc' type=radio value='visa'><p>
<input type=submit value='aceptar'>
</form>
</body>
</html>
```

Formularios HTML



Paso de parámetros

- METHOD=GET
  - Los parámetros aparecen encadenados en la URL que invoca el WB

    http://www.tienda.es/cgi-bin/tienda.pl?nombre=German&...
  - Cuando recibe la respuesta, el WS almacena la cadena de parámetros en la variable de entorno QUERY_STRING

- METHOD=POST
  - La cadena de parámetros tiene el mismo formato que el anterior, pero aparece en en cuerpo de la petición HTTP
  - El WS prepara los parámetros como entrada estándar al programa CGI

Aplicaciones basadas en WB: Java applets

- No se corresponde al paradigma clásico de Cliente/Servidor
- El WS es literalmente un servidor de aplicaciones
- Cualquier WB que acepte applets es un cliente potencial
- No hay problemas de portabilidad de código, ni de actualización del software del cliente (siempre tiene la última versión)
- Pero ... sólo es válido para aplicaciones pequeñas debido a las múltiples restricciones de seguridad (se ejecutan completamente en la máquina del cliente)

Creación de un applet

- Compilamos un programa.java
- El correspondiente programa.class es un conjunto de bytecodes: código de la máquina virtual java JVM
- Se "incrusta" el código en la página html

```
<html>
<head> ... </head>
<body>
<applet codebase="http://www.sc.ehu.es/"
 code="programa.class" height=1000 width=400>
<param name=param1 value=100>
Su navegador no soporta Java
</applet>
</body>
```

Restricciones de seguridad

- A un applet no le está permitido:
  - Comunicarse con ninguna otra máquina que no sea el WS (sólo conexiones con el nodo de donde se descarga)
  - Leer, crear, modificar o ejecutar ningún fichero del cliente
  - Obtener información del cliente: nombre, versión del SO, direcciones e-mail, etc.
- La seguridad la garantiza el *applet security manager*
- Los applets no se ejecutan directamente en el cliente
- Son interpretados por una JVM
- Control de seguridad antes y durante su ejecución
- Hay situaciones en los que la seguridad se puede relajar:
  - Applets con "certificado"
  - Configuramos el WB para autorizar a determinados applets certificados su ejecución sin algunos niveles de seguridad

http://www.ted.com/talks/lang/en/fabian_hemmert_the_shape_shifting_future_of_the_mobile_phone.html

http://www.ted.com/talks/john_underkoffler_drive_3d_data_with_a_gesture.html