Introducción. Este material incorpora al anterior de la Unidad II (2004) del apunte de la Cátedra la creación de la interfase gráfica del usuario, usando el **Constructor de Interfaces Gráficas del NetBeans IDE** (**GUI Builder**), para una aplicación que llamaremos **Contactos.**

El material se elabora a partir de una tutoría (Ingles), a la cual llegamos desde el entorno de NetBeans (se requiere conexión a Internet): <Help>, <Quick Start Guide>, <Gueting Started>, <Java GUI Applications>, <Tutorials, Guides and Demos>, <Designing GUIS>, <**GUI Building in Netbeans IDE**>

Mas material disponible relacionado con este tema se encuentra en www.netbeans.org/kb/60/java, como ser: Introduction to GUI Building, Building a java Desktop Databse Application, Working with the Java DB (Derby) database in Neatbeans IDE, GUI Builder FAQ, Java GUI Application Learning Trail.

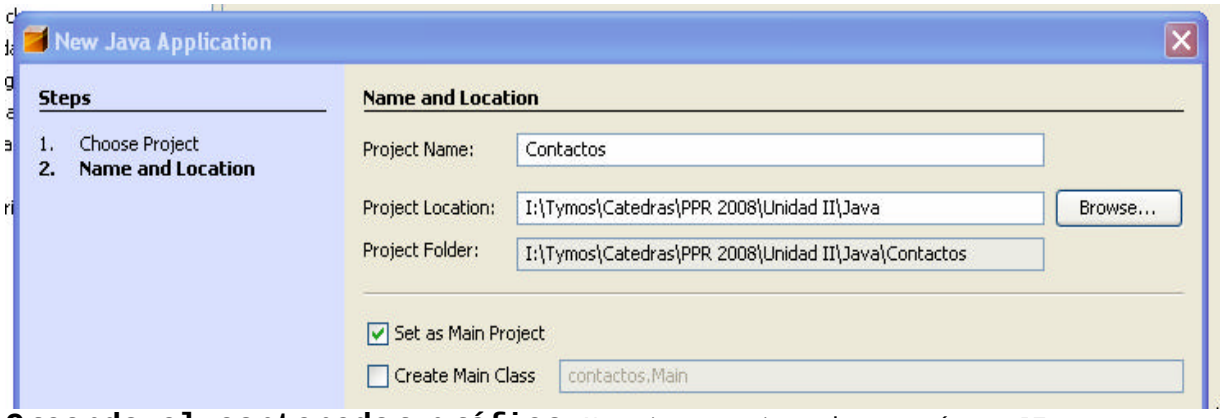
Utilizando este material UD aprenderá a:

- Usar el **GUI Builder**
- Crear un contenedor de GUI
- Agregar componentes
- Redimensionar componentes
- Alinear componentes
- Ajustes de ancho
- Posesionar su comportamiento
- Editar sus propiedades

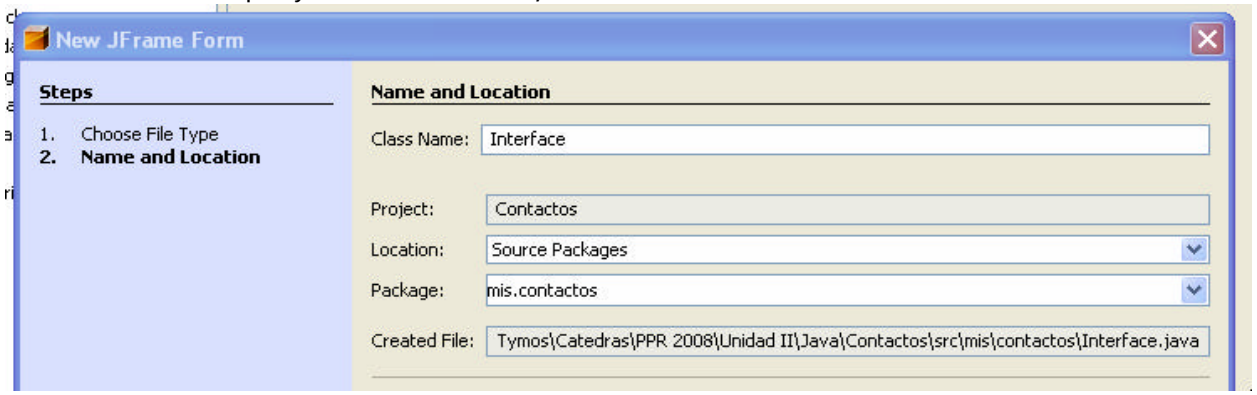
Luego, al código construido por **GUI Builder** le incorporamos nuestro tratamiento de eventos para tratamiento de contactos en una agenda de E_mails, incluyendo su almacenamiento persistente en disco, y su actualización.

Creando el proyecto. Usando el NetBeans IDE,

- seleccione: **<File> <New Proyect> <Java Classes> <Java Class>**
- (Name and Location) Proyect Name **Contactos** , defina localizacion, contenedor(fólder) **Contactos**

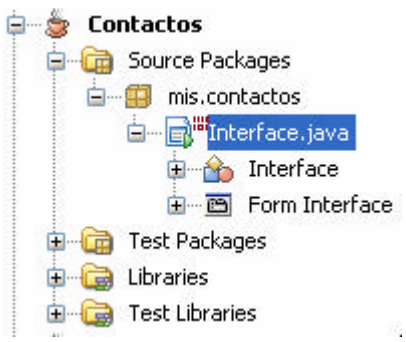


Creando el contenedor gráfico Nuestro contenedor será un JFrame. Posiciónese en proyecto **Contactos, <boton derecho> <new> <Jframe Form>**



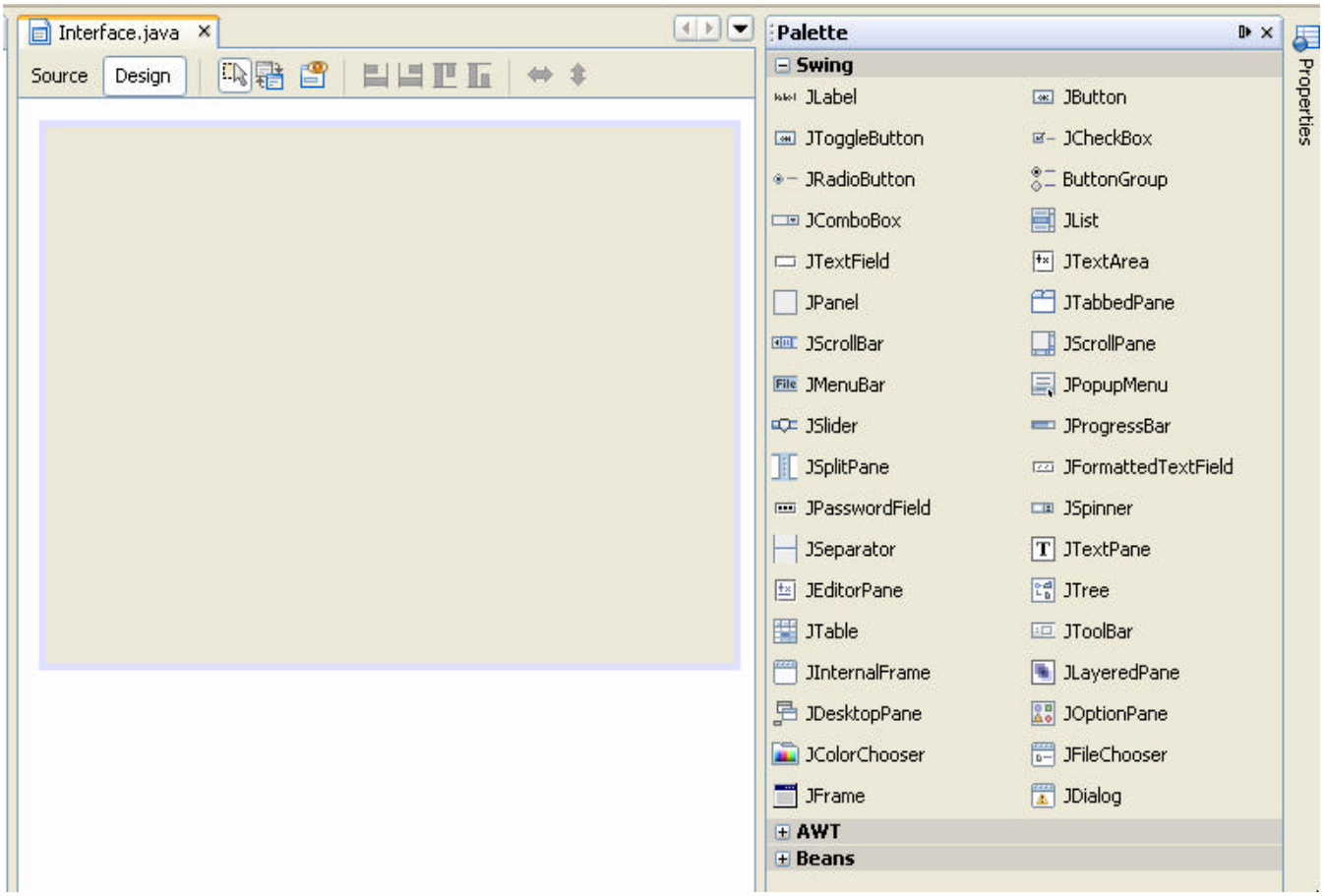
Cuando <Finalizar>, NetBeans IDE nos muestra:

Como quedó el proyecto?



Dentro del proyecto **Contactos** se ha generado una estructura. El package **mis.contactos** contiene Interface.java, constituida por Interface (Fuente en codigo Java) y Form Interface, que es un archivo .form generado por **GUI Builder**
Interface no debe ser modificada, salvo en los lugares específicamente permitidos.

Y que nos ofrece el IDE?



A la izquierda esta la ventana contenedora JFrame. (Interface.java, design)
A la derecha se nos ofrece la ventana Palette, con componentes Swing: JLabel, Jbutton, JRadioButton, JChekBox, etc, etc. UD. puede ver la descripción y su utilidad en el apunte Unidad II (2004), pgs 6 a 14. No profundice sobre los métodos, UD no los necesitará si dispone de **GUI Builder**.

ED nos proporciona cuatro ventanas.

- **Design Area**. Es la que está viendo arriba, a la izquierda. Allí incluiremos los componentes de nuestra interface. Para facilitarnos la tarea, nuestra Design Area incluye, arriba, una barra (toolbar) conteniendo botones para:
  - Source. Para ver el código java generado por ED
  - Design. Vista gráfica de los componentes GUI
  - Selection mode

- Connection mode
- Preview design

- **Inspector**. Provee una representación de los componentes, visual y no visual, en una jerarquía en árbol.
- **Palette**. Una lista de todos los componentes disponibles, (Swing, AWT, JavaBeans)
- **Properties**. Muestra las propiedades del componente corrientemente seleccionado en ED.

Cuando estemos en la vista Source, vemos el código.

- Fondo azul, son áreas de codificación no modificables en este modo. Si algo es necesario modificar, ir al modo design, hacer los ajustes. Al salvar el design, el IDE actualiza sources.
- Fondo blanco, nuestra área de código a completar

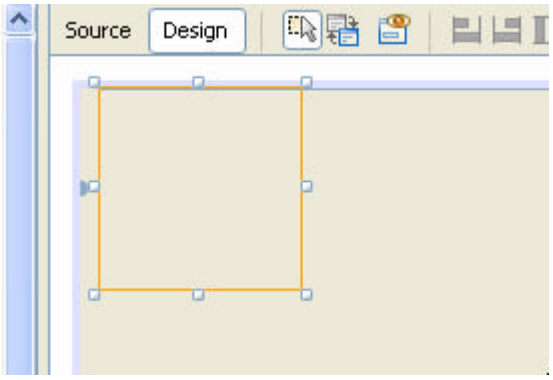
Primeros pasos.

Nos hemos mínimamente familiarizado con ED. Es hora de usar.
Gracias al paradigma de Diseño Libre del IDE, podemos despreocuparnos de la distribución de los componentes, y de los "Gestores de Diseño" (layout managers) que se ocupaban de ello. Todo lo que necesita es arrastrar y soltar (drag and drop) o picar y arrojar (pick and plop) componentes.

Agregando componentes. Lo básico.
Tenemos la ventana JFrame, contenedor de máximo nivel (Form's top level container). Necesitamos de un par de componentes Jpanel (Contenedores intermedios) para agrupar los componentes de nuestro UI (User Interfaz) con bordes titulados (tittled borders)

Agregando un par de JPanel's al JFrame (Mas fácil hacerlo que escribirlo)

- En la ventana Palette, seleccionar JPanel
- Desplace el cursor hacia la esquina superior izquierda del area de diseño. Cuando las líneas guía le muestran una posición adecuada, <click>.
- El JPanel ha sido incluido, si bien no es visible. Solo cuando pasa el cursor sobre esa área, aparece en gris claro. <click> dentro del JPanel.

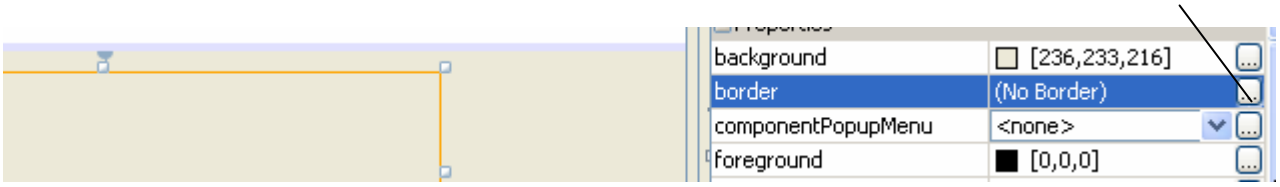


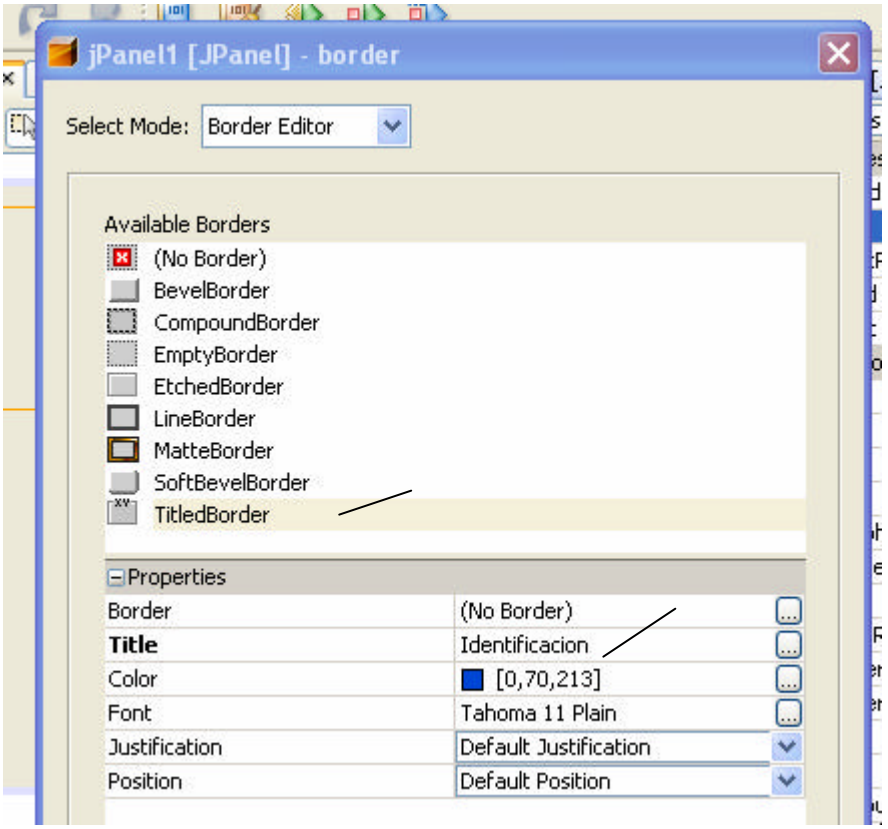
Posicione el cursor sobre el cuadrado central , lado derecho.. El cursor cambia de forma., de flecha inclinada pasa a (<->) Arrastre hasta que el borde del JPanel quede a un par de cms del borde derecho del JFrame. Lo mismo en el lado horizontal inferior del JFrame. Haga que el JPanel cubra cerca de la mitad del JFrame.
Repita el proceso para incorporar un segundo panel en la mitad inferior del JFrame..

Ya tenemos ambos JPanel posicionados y dimensionados dentro de nuestro Júrame. Por ahora no son visibles, excepto cuando deslizamos el cursos arriba de ellos. Usaremos el superior para datos personales y el inferir para el e_mail.
El paso siguiente es incorporarles un borde y un título a cada uno.

Agregando títulos de borde (title borders) al panel

- Seleccione el panel superior
- En el tab menú, <Windows>, <Properties>. <Click>





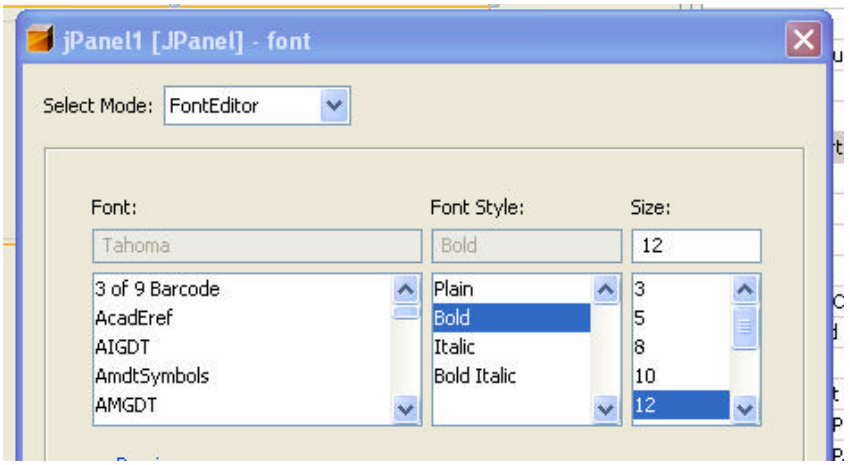
Aparece el editor de bordes

Opte: **TitledBorder**

En Title, tipee

Identificacion

Necesitamos informar tipo y tamaño de letra del título.



Nuevamente <Windows>, <Properties> Clickear en (…) de Font, aparece su editor, Elegir Tahoma, Bold, 12, <ok>

Repetimos todo el proceso para conformar el JPanel inferior, que lleva como titulo E_mail. Al final de esto nuestra incipiente GUI debe aparecer como:

Tenemos un panel Identificación, otro para E_mail.

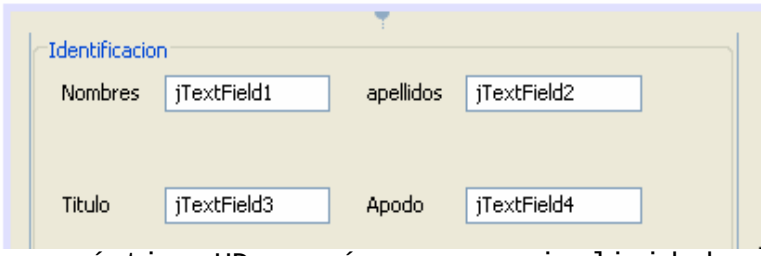
Y espacio adicional en la ventana para otros componentes que necesitaremos incorporar:

En identificación, 4 campos texto (JTextField) y sus 4 respectivos rótulos. (JLabel)

Incorporando componentes individuales

Agregando un rótulo (JLabel).

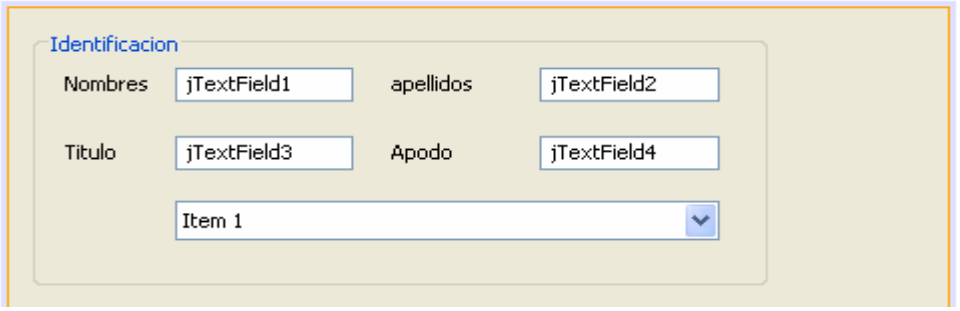
- En la ventana Palette, seleccione JLabel
- Mueva el cursor sobre el panel Identificación, cerca de la esquina superior izquierda. Aparecen líneas guía que ayudan el posicionamiento. <Click> para agregar.
- Necesitamos incorporar el texto de la etiqueta. <Doble click> sobre ella, pasa a modo edición. Tipee "Nombres", <Enter>. Nos queda:



Ahora debemos incorporar el campo de texto respectivo

Selección JTextField en Palette, mueva el cursor a la derecha de Nombres (Líneas guía ayudan), <Click>
Y repetimos todo el proceso para apellido, titulo, apodo.

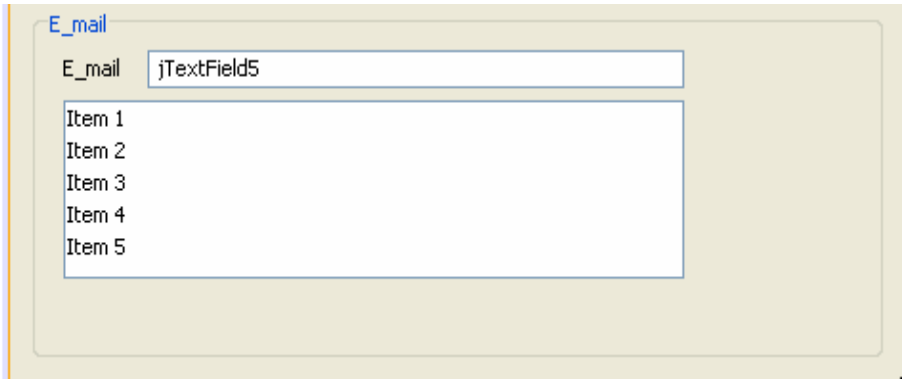
En su práctica UD. verá con que simplicidad se puede desplazar, alinear, modificar tamaño.
Ahora supongamos que necesitamos captar diversos detalles del contacto. Optamos por una lista desplegable (JComboBox). La incorporamos.



Agregando, alineando, anclando

El editor gráfico permite distribuir los componentes rápida y facilmente perfeccionando flujos de trabajo típicos. Cuando UD. agrega un componente a una forma, el editor automáticamente lo posiciona en la posición preferida y crea las relaciones que sean necesarias. UD. puede concentrarse en diseñar su forma más bien que sufrir en complicados detalles.

A continuación trabajamos en el JPanel E_mail. Hay una primera parte parecida a la que ya hemos realizado en el JPanel Identificación, por lo que no repetiremos los detalles. Consiste en agregar un JLabel E_mail, un JTextField para la captura de este dato, y un JList para mostrar nuestra agenda de contactos completa. Esta primera parte debe aparecer así:



Observe que hemos dejado espacios en blanco:

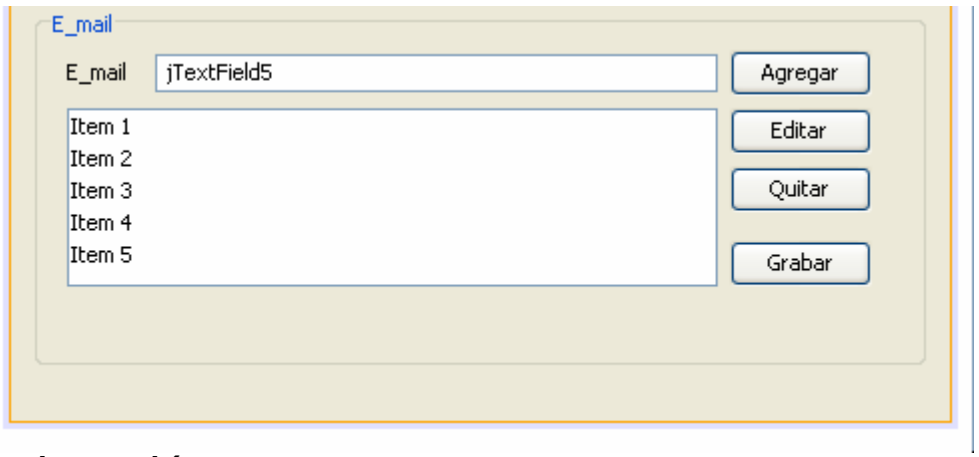
a) En la parte inferior del Jpanel
b) En la parte inferior del JFrame
c) A la derecha del JPanel.

Los necesitaremos

Dimensionado de componentes (Component sizing)

Frecuentemente es conveniente ajustar varios componentes relacionados al mismo tamaño, por motivos de consistencia visual. Para demostrar esto agregaremos cuatro JButtons a nuestra interfase para permitirnos agregar, editar, quitar y grabar en nuestra lista de contactos.
Para ello, los pasos son:

- En la paleta seleccione JButton.
- Posicione el cursor a la derecha del JTextField5 y <shift click>
- Posicione el cursor a la altura del item1 y <shift clic>.
- Repita esto para agregar dos JButtons mas, distribuyéndolos.
- Reemplace los rótulos por Agregar, Editar <doble clic>. Nos queda:

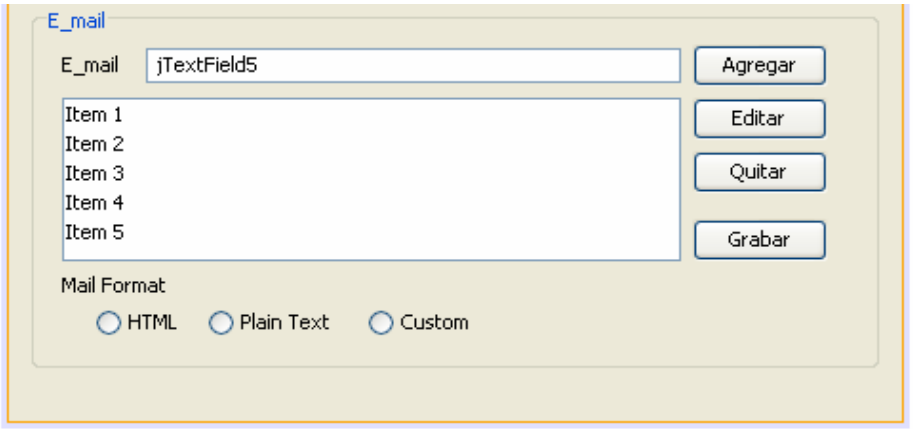


Indentación (Sangría)

A menudo es necesario agregar múltiples componentes bajo otro componente de manera que quede claro que pertenecen a una operación relacionada. Como ejemplo de lo dicho agregaremos varios JRadioButtons debajo de un JLabel para permitir al usuario personalizar la forma en que la apicacion muestra los datos.

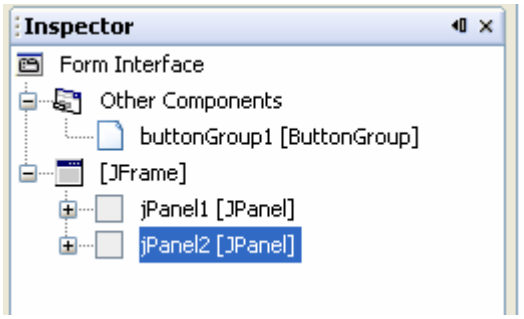
Indentar varios JRadioButtons debajo de un JLabel

- Agregue un Jlabel debajo del JList, a derecha.
- Identifíquelo como Mail Format
- Seleccione JRadioButton en la paleta
- Ubique cursor debajo del JLabel. Use líneas guía para posicionarse ligeramente indentado a derecha del JLabel. <shift click>
- Desplace cursor a derecha. <shift click> para posicionar
- Repita una última vez, fijamos tercer JRadioButtom
- <Doble clic> sobre cada JRadioButtom. Identifiquemos como HTML, Plain Text y Custom. Debemos ver lo siguiente:

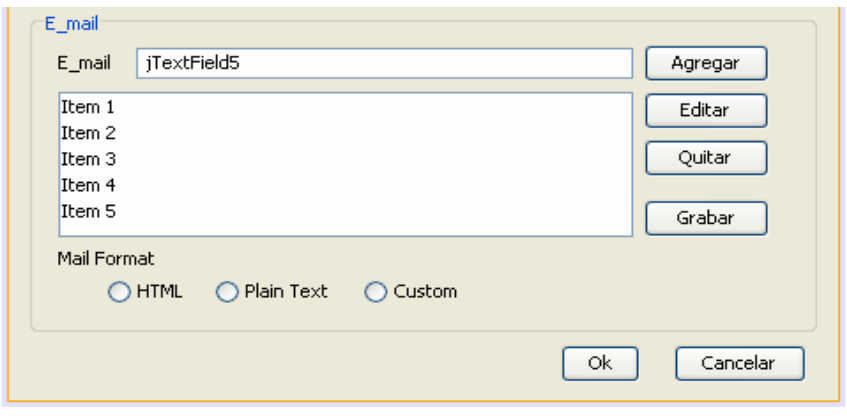


El paso siguiente es agregar los JRadioButtoms a un ButtomGroup:

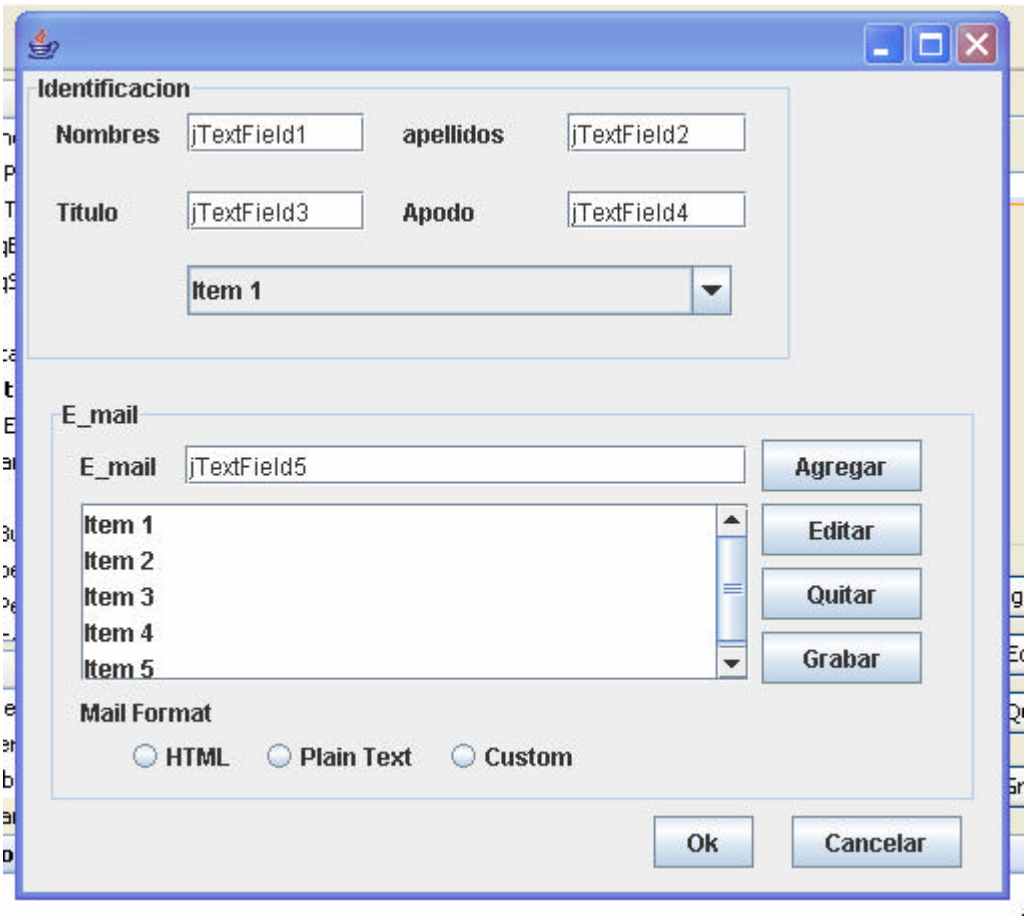
- Seleccione Buttom Group en la paleta.
- Clickee en cualquier lugar del Jpanel E_mail. No aparece en la paleta, pero es visible el el área de inspección de otros componentes (Inspector Other Component area)
- Seleccione los tres JRadioButtoms (<shift click>)
- En la ventana Properties, seleccione buttomGroup1.
- Los tres JRadioButtoms son incorporados al ButtomGroup.



Necesitamos incorporar un par de botones para confirmar la información introducida, o cancelarla. Agregamos estos botones en la parte inferior del JFrame y mostramos como nos finalmente queda



Compilamos, ejecutamos. Nos aparece la interface. Si clickeamos en cualquiera de sus componentes no ocurre nada aparte del comportamiento de focalización propio de la interfase, pero ninguna respuesta. Todavía no tenemos **tratamiento de eventos**



Tratamiento de eventos

Hasta ahora, hemos construido una interfase totalmente inoperante. Lo único que funciona es lo que viene en el comportamiento "automático" de los componentes.

Vamos a incorporar eventos únicamente a los cuatro JRadioButtons que están a la derecha del Panel E_mail. Hacerlo en todo la interfase es innecesariamente largo, podemos dejar esta tarea a la investigación de los alumnos. En concreto, el comportamiento que queremos es:

- Al iniciar la aplicación, suponemos que existe una agenda de contactos guardada en disco, "Agenda.dat". Esta agenda es un objeto LinkedList, lo leemos y cargamos en memoria. El resto del comportamiento dependerá de la activación de los siguientes botones
- **Agregar:** Incorpora a la LinkedList en memoria el contacto informado en JTextField5
- **Editar:** Edita sucesivos contactos almacenados en la LinkedList, usa el mismo JTextField5.
- **Quitar:** Quita de la LinkedList el contacto corrientemente editado en JTextField5.
- **Grabar:** Guarda en disco **Agenda.dat**

Todo este comportamiento debe estará adecuadamente "trazado" para su fácil seguimiento, usando System.out.println.

Como ya dijimos, este ejemplo no contiene toda la codificación necesaria para usar todos sus componentes. Inclusive deja de lado el componente JList, que permite resolver este comportamiento de una forma bastante más profesional. La idea es que una vez entendido el

mecanismo de conexión de la interfase con la parte del tratamiento de eventos, se la puede generalizar fácilmente.

Preparando la incorporación del tratamiento de eventos.

Si editamos el contenido de nuestra **Interface.java**:

```
/*
 * Interface.java
 *
 * Created on 19 de febrero de 2008, 19:18
 */

package mis.contactos;

/**
 *
 * @author  Tymoschuk, Jorge
 */
public class Interface extends javax.swing.JFrame {

    /** Creates new form Interface */
    public Interface() {
        initComponents();
    }

    /** This method is called from within the constructor to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */

+     Generated Code    // Código generado por GUI Builder.
                        // No abrirlo, sus modificaciones se pierden.
/**
* @param args the command line arguments
*/
    public static void main(String args[]) throws IOException {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Interface().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.ButtonGroup buttonGroup1;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JButton jButton4;
    private javax.swing.JButton jButton5;
    private javax.swing.JButton jButton6;
    private javax.swing.JComboBox jComboBox1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
```

```
    private javax.swing.JLabel jLabel6;
    private javax.swing.JList jList1;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JRadioButton jRadioButton1;
    private javax.swing.JRadioButton jRadioButton2;
    private javax.swing.JRadioButton jRadioButton3;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    private javax.swing.JTextField jTextField3;
    private javax.swing.JTextField jTextField4;
    private javax.swing.JTextField jTextField5;
    // End of variables declaration

}
```

Que vemos?

- Un constructor **public Interface()**{ initComponents();}, abierto a modificaciones; podemos agregar, quitar contenido Java.

- **Generated Code** // Código generado por GUI Builder.
  // No abrirlo, sus modificaciones se pierden.

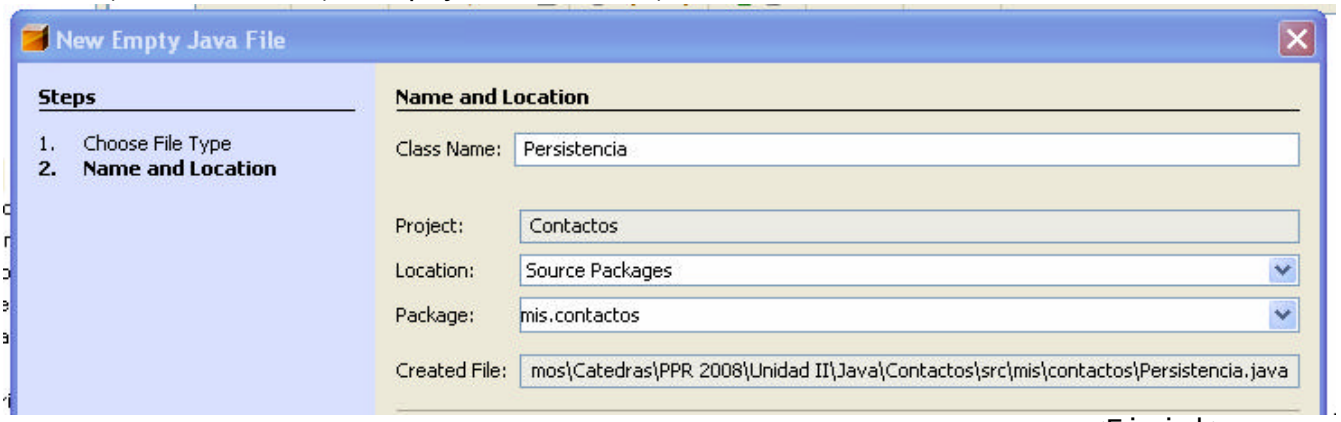
- **public static void main(String args[])** throws IOExceptions{
  // usamos el constructor para generar interface y tornar
  // visibles sus componentes. Nada mas.

- **// Variables declaration - do not modify**

```
private javax.swing.ButtonGroup buttonGroup1;
private javax.swing.JButton jButton1;
private javax.swing.JButton jButton2;
private javax.swing.JButton jButton3;
private javax.swing.JButton jButton4;
```

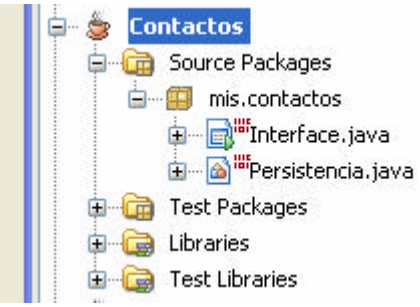
El hecho que el constructor esté abierto nos posibilita que aquí activemos el tema de leer la agenda anterior desde almacenamiento persistente y lo carguemos en la LinkedList. Ahora leemos, luego deberemos grabar, este comportamiento pertenece a objetos de otra clase, Persistencia. Para hacer las cosas bien, en nuestro IDE incorporamos una clase java vacía, la llamamos Persistencia, la incluimos en nuestro packagge …

<File<, <New file<, <Empty Java File>,



<Finish>

Vemos que en nuestro proyecto aparece una clase mas, Persistencia.



Esta clase, junto con los objetos LinkedList, Iterator, los declaramos en Interface. En el constructor de Interface invocamos initRest(), quien invoca persiste.leerAgenda.

Todo esto queda:

```
public class Interfase extends javax.swing.JFrame {
    LinkedList agenda;
    Iterator ageIter;
    Persistencia persiste;

    /** Creates new form ContactEditorUI */
    public Interfase() throws IOException{
        initComponents();
        initRest();
    }

    private void initRest() throws IOException{
        try{
            persiste = new Persistencia();
            System.out.println("Instanciado persiste ");
            agenda = new LinkedList();
            System.out.println("Agenda vacia " + agenda);
            agenda = persiste.leerAgenda();
            System.out.println("Agenda leida " + agenda);
            ageIter = agenda.iterator();// Para recorrer agenda
            System.out.println("Instanciado ageIter ");
        }catch(IOException ioe){
            System.out.println("IOException en initObjects()");
        }
    }
```

leerAgenda() lo tenemos en Persistencia, helo aquí:

```
package mis.contactos;
// Cuidando de nuestros contactos
import java.io.Serializable;
import java.io.*;
import java.util.*;
public class Persistencia implements Serializable{

    public LinkedList leerAgenda() throws IOException{
        LinkedList age = new LinkedList();;
        try{
            FileInputStream ageIn = new FileInputStream("Agenda.dat");
            ObjectInputStream objIn = new ObjectInputStream(ageIn);
            if(objIn != null)
                age =  (LinkedList)objIn.readObject();
        }catch (EOFException eof){
            System.out.println("Fin de Agenda.dat");
        }catch (ClassNotFoundException nfe){
            System.out.println("Class Not FoundException ");
```

```
            }catch (IOException ioe){
                System.out.println("IOException en leerAgenda");
            }
            System.out.println("leerAgenda(), tengo " + age);
            return age;
        }   //  void leerAgenda()
```

Bien, con esto tenemos lista la construcción inicial: Nuestra agenda anterior esta cargada en el objeto agenda, atributo de Interfase.

Incorporando el tratamiento de eventos.

Comenzamos por el botón Agregar. Es el primero que hemos incorporado a nuestra interfase, viendo la definición de las variables es el

```
        private javax.swing.JButton jButton1;
```

- Posicionamos el cursos sobre el botón, <click>, <derecho>, <Events>, <Action>, <**ActionPerformed**>

El IDE nos muestra

\+ Generated Code

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt){

// TODO add your handling code here:

}
```

- Reemplazamos la línea de comentarios por **nuestra codificación**

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt){
    //  Agregar contactos
    String contact = jTextField5.getText();
    System.out.println("Antes agenda contiene " + agenda);
    agenda.add(contact);
    jTextField5.setText("");
    System.out.println("Contacto agregado " + contact);
    System.out.println("Ahora agenda contiene " + agenda);
}
```

Repetimos los dos puntos anteriores en los tres botones restantes, luego de bastante trabajo de codificación y prueba va quedando:

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
    //  Grabar mi agenda en disco
    try{
        persiste.grabarAgenda(agenda);
        System.out.println("");
        System.out.println("Salvo en Agenda.dat " + agenda);
   }catch (IOException ioe){
       System.out.println("IOException en persiste.grabarAgenda()");
   }
}
```

```
  private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
      //  Suprimir el contacto en edicion
      String aux = jTextField5.getText();
      jTextField5.setText("");
      ageIter.next(); // salte el siguiente
      ageIter.remove(); // y remuevalo
      System.out.println("Contacto suprimido " + aux);
      System.out.println("Agenda contiene " + agenda);
  }

  private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
      //  Editar contactos
      String contact;
      if (ageIter.hasNext()){
          contact = (String)ageIter.next();
          jTextField5.setText(contact);
      }else  jTextField5.setText("No mas contactos ...");
  }
```

El mensaje **persiste.grabarAgenda(agenda)** de **jButton4ActionPerformed:**

```
  public class Persistencia implements Serializable{

     public LinkedList leerAgenda() throws IOException{// ya la vimos

     public void grabarAgenda(LinkedList cont) throws IOException{
        System.out.println("estoy en grabarAgenda()");
        try{
          FileOutputStream ageOut = new FileOutputStream("Agenda.dat");
          ObjectOutputStream objOut = new ObjectOutputStream(ageOut);
            if( objOut!=null){
               System.out.println("if(objOut != null)");
               objOut.writeObject(cont);
               System.out.println("writeObject " + cont);
               objOut.close();
            }
        }catch (IOException ioe){
           System.out.println("IOException en grabarAgenda()");      }
    }
  }  //  class Persistencia
```

```
run:
Instanciado persiste
Agenda vacia []
leerAgenda(), tengo [pepe@coldMail.com, poto@Fisrtmail.com]
Agenda leida [pepe@coldMail.com, poto@Fisrtmail.com]
Instanciado ageIter
Antes agenda contiene [pepe@coldMail.com, poto@Fisrtmail.com]
Contacto agregado pipo@Lastmail.com
Ahora agenda contiene [pepe@coldMail.com, poto@Fisrtmail.com,
pipo@Lastmail.com]
estoy en grabarAgenda()
if(objOut != null)
writeObject [pepe@coldMail.com, poto@Fisrtmail.com, pipo@Lastmail.com]
Salvo en Agenda.dat [pepe@coldMail.com, poto@Fisrtmail.com,
pipo@Lastmail.com]
BUILD SUCCESSFUL (total time: 43 seconds)
```