



Ministerio de Producción
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



Programa
111
mil
VOS PODÉS
SER UNO.

Swing / AWT



Agenda

- Interfaces de usuario en Java
- AWT
 - Componentes
 - Contenedores
 - Layouts
 - Eventos
- Swing
 - Componentes
 - Contenedores
 - Conversión de aplicaciones AWT a Swing
 - Modelo-Vista-Controlador
 - Componentes avanzados
- Ejemplo de GUI en Netbeans IDE

GUI en Java

- **GUI - Graphical User Interface** (en castellano Interfaz de Usuario Gráfica)
- El usuario interactúa con las aplicaciones usando la interfaz gráfica.
 - Las GUIs nos permiten ingresar datos a través de campos de texto, elegir entre posibles opciones usando una lista desplegable, tildar valores deseados usando check-boxes, cerrar ventanas, etc.
 - Esta interacción del usuario con la aplicación, seguramente tendrá un efecto sobre la lógica de la aplicación.
 - Por ejemplo, en general, cuando se completan datos de un formulario, los valores ingresados son procesados y quizá, almacenados en una tabla de mi base de datos.
- Java provee un conjunto de clases para escribir programas con GUI, llamado **AWT (Abstract Window Toolkit)**.



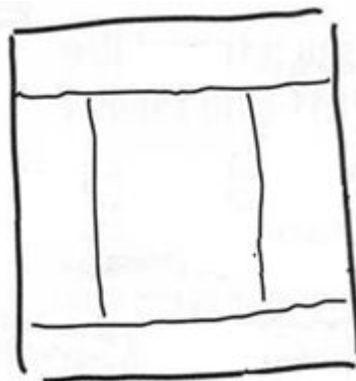
AWT - Abstract Window Toolkit

¿Qué provee AWT?

- **Componentes GUI básicos** como botones, campos de texto, listas desplegables, etc., que pueden ser utilizados en los Applets y en las aplicaciones Java de escritorio.
- **Clases contenedoras** que contienen a otros componentes. Algunas de los componentes que se agregan o sitúan en una clase contenedora, a su vez, pueden contener a otros componentes (con contenedores a su vez).
- Clases que **acomodan las componentes** en un contenedor, conocidos como layouts managers.

AWT - Componentes y Contenedores

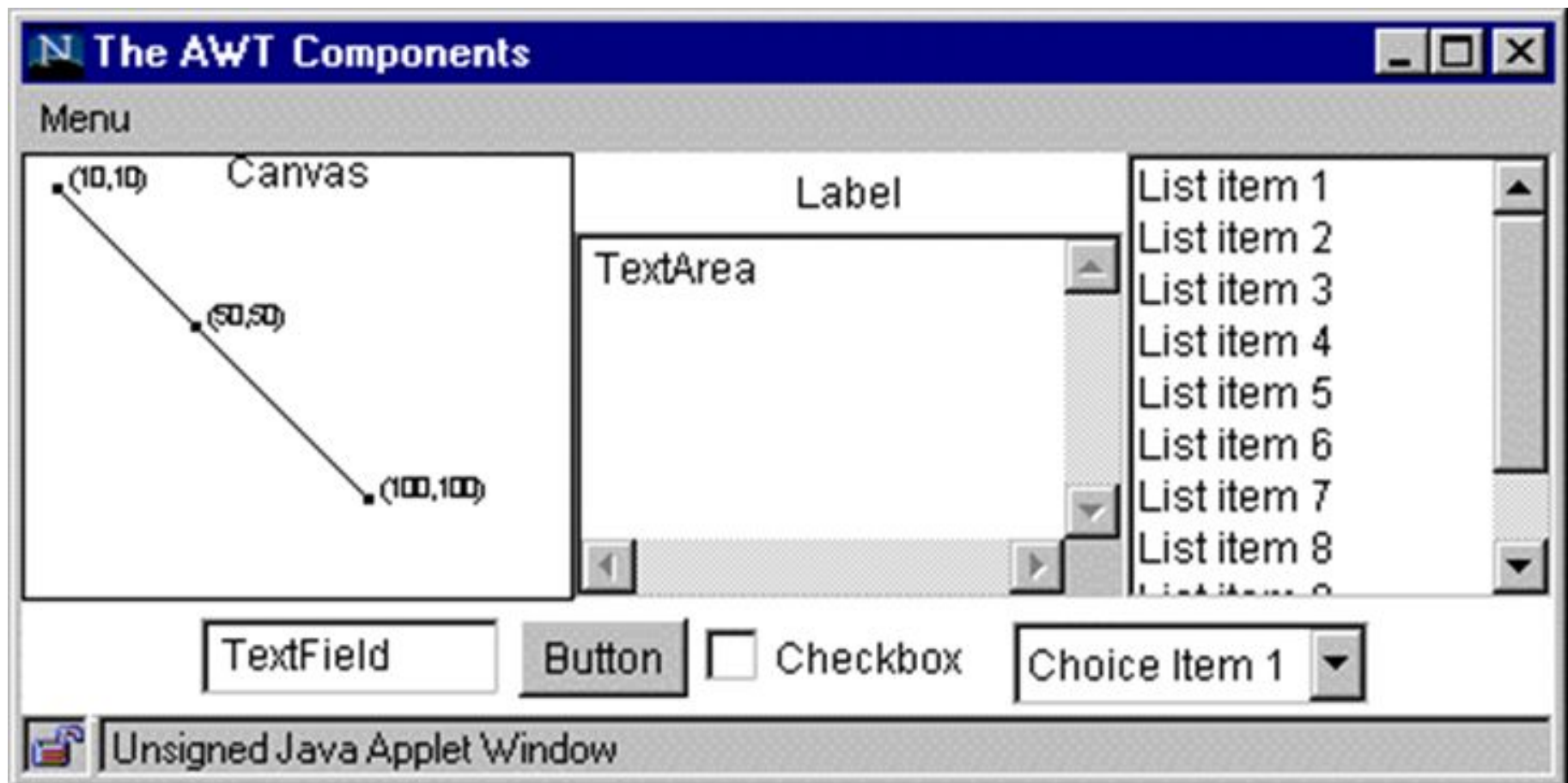
- AWT establece una relación simple y fundamental entre objetos **Component** y **Container**: un Container pueden contener múltiples Components.
- Cada objeto Component tiene un Container padre.
- Todos los Containers tienen asociado un layout manager que establece la ubicación y el tamaño de las componentes dentro del Container. Cada container tiene un layout manager único.



Este podría ser un Contenedor principal, con 5 regiones que a su vez, cada una de ellas podría contener otros contenedores o componentes como botón, lista, etc.

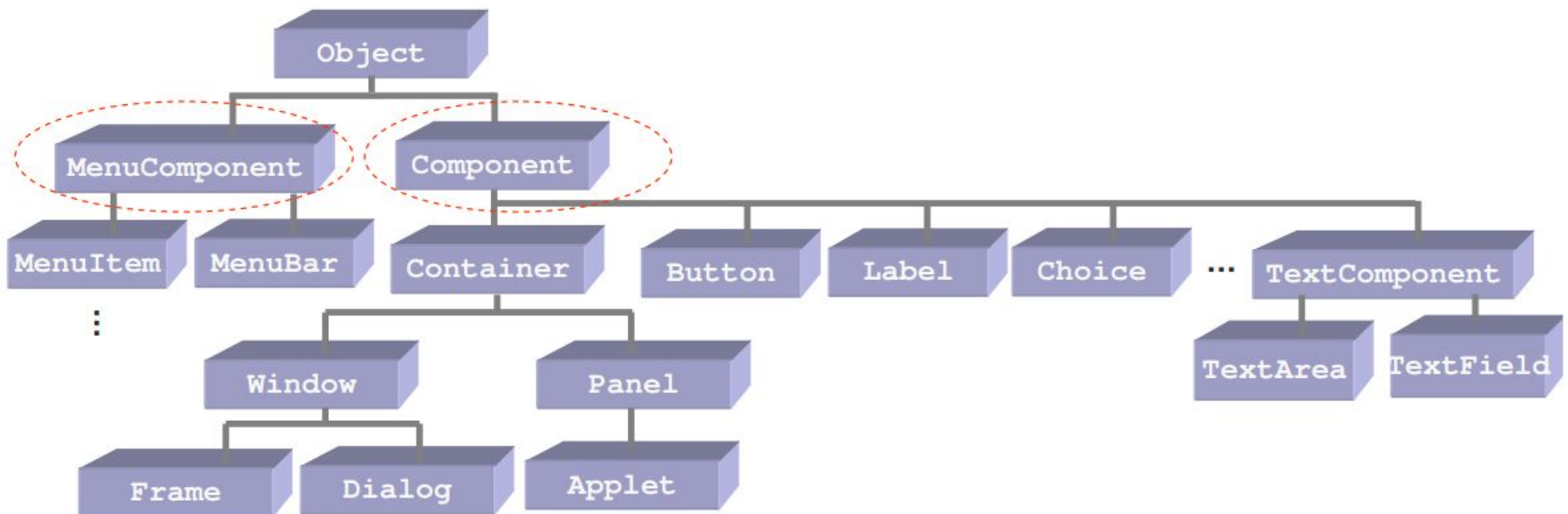
AWT - Componentes y Contenedores

- Esta ventana, que es un contenedor, contiene distintas componentes AWT



AWT - Componentes

- Cada componente GUI es subclase de la clase abstracta **Component** o **MenuComponent**. Las componentes de GUI básicas y los contenedores, heredan de **Component** una cantidad de métodos y variables de instancia.



AWT - Componentes

Botones de Pulsación

- La clase **Button** es una clase que produce una componente de tipo botón con una etiqueta (texto) que se visualizará.
- Métodos más importantes:
 - **addActionListener()**: añade un receptor de eventos de tipo Action producidos por el botón
 - **getLabel()**: devuelve la etiqueta o título del botón.
 - **removeActionListener()**: elimina el receptor de eventos y el botón deja de avisar cuando ocurre un evento.
 - **setLabel()**: fija el título o etiqueta visual del botón

Botones de Elección

- Los botones de selección en una lista (choice) permiten el rápido acceso a una lista de elementos. Por ejemplo, podríamos implementar una selección de colores y mantenerla en un botón Choice:

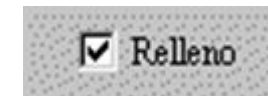


AWT - Componentes

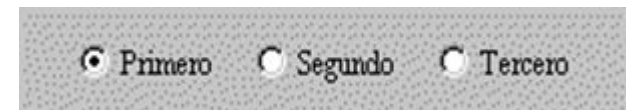
Botones de Marcación

- La de comprobación (Checkbox) se utilizan frecuentemente como botones de estado. Proporcionan información del tipo Sí o No (true o false). El estado del botón se devuelve en el argumento Object de los eventos Checkbox; el argumento es de tipo booleano: verdadero (true) si la caja se ha seleccionado y falso (false) en otro caso.
- Tanto el nombre como el estado se devuelven en el argumento del evento, aunque se pueden obtener a través de los métodos `getLabel()` y `getState()` del objeto Checkbox.

Botones de Selección



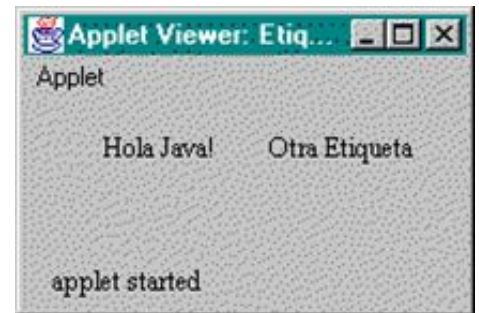
- Los botones de comprobación se pueden agrupar para formar una interfaz de botón de radio (CheckboxGroup), que son agrupaciones de botones Checkbox en las que siempre hay un único botón activo.



AWT - Componentes

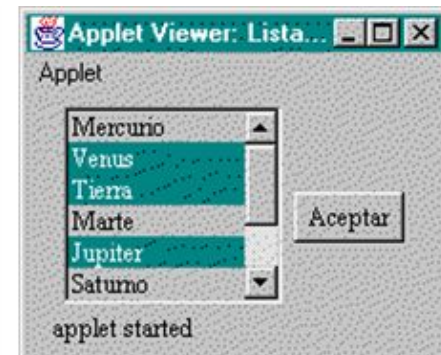
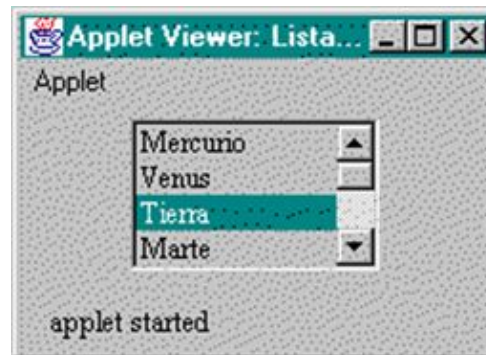
Etiquetas - Labels

- Las etiquetas (*Label*) proporcionan una forma de colocar texto estático en un panel, para mostrar información que normalmente no varía, al usuario.



Listas - List

- Las listas (*List*) aparecen en los interfaces de usuario para facilitar a los operadores la manipulación de muchos elementos. Se crean utilizando métodos similares a los de los botones choice. La lista es visible todo el tiempo, utilizándose una barra de desplazamiento para visualizar los elementos que no caben en el área que aparece en la pantalla.

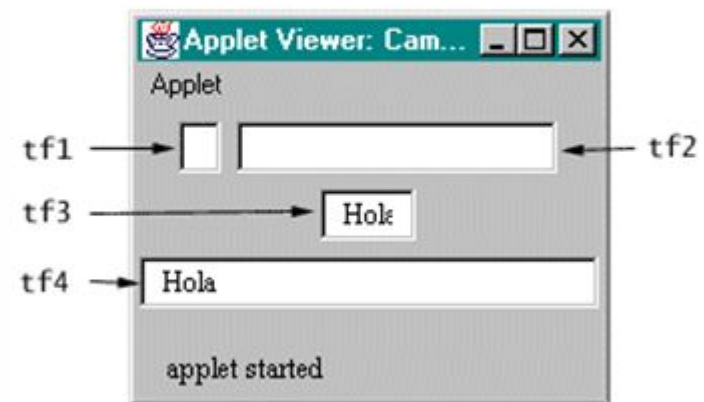


AWT - Componentes

Campos de Texto - TextField

- Para la entrada directa de datos se suelen utilizar los campos de texto, que aparecen en pantalla como pequeñas cajas que permiten al usuario la entrada por teclado.
- Los campos de texto se pueden crear vacíos, vacíos con una longitud determinada, rellenos con texto predefinido y rellenos con texto predefinido y una longitud determinada.

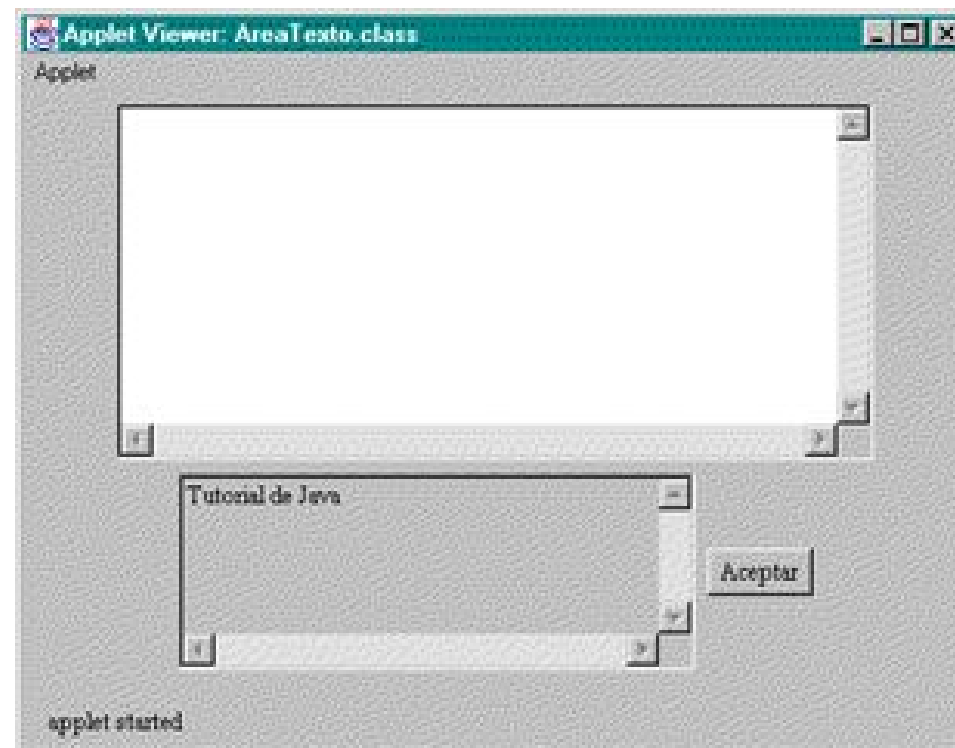
```
import java.awt.*;  
import java.applet.Applet;  
public class CampoTexto {  
    TextField tf1,tf2,tf3,tf4;  
    public void init() {  
        // Campo de texto vacío  
        tf1 = new TextField();  
        // Campo de texto vacío con 20 columnas  
        tf2 = new TextField( 20 );  
        // Texto predefinido  
        tf3 = new TextField( "Hola" );  
        // Texto predefinido en 30 columnas  
        tf4 = new TextField( "Hola",30 );  
        add( tf1 ); add( tf2 ); add( tf3 ); add(tf4);  
    }  
}
```



AWT - Componentes

Áreas de Texto - TextArea

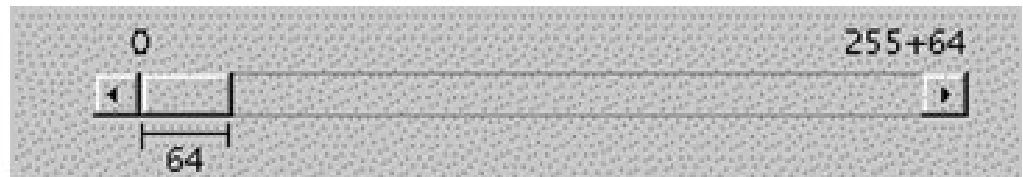
- Permite incorporar texto multilínea dentro de zonas de texto (TextArea). Los objetos TextArea se utilizan para elementos de texto que ocupan más de una línea, como puede ser la presentación tanto de texto editable como de sólo lectura.
- Para las áreas de texto hay que especificar el número de columnas.



AWT - Componentes

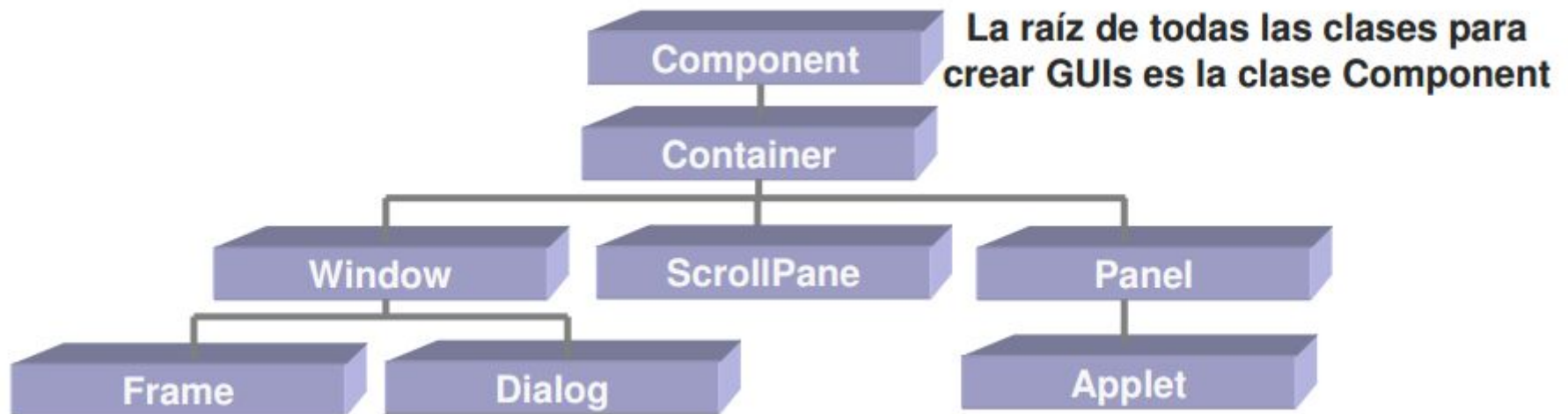
Barras de Desplazamiento - Scroll Bar

- En determinadas situaciones se necesitan realizar el ajuste de valores lineales en pantalla, resulta útil el uso de barras de desplazamiento. Proporcionan una forma de trabajar con rangos de valores o de áreas, como el componente TextArea, que proporciona dos barras de desplazamiento automáticamente.



AWT - Contenedores

- Hay dos tipos principales de Contenedores: Windows y Panel. Un contenedor Window es una ventana que se ubica en un lugar de la pantalla y que es independiente de otras ventanas, un Panel es una componentes que debe ubicarse en un contenedor Window.



Para agregar los componentes a un **Contenedor**, se utiliza el método **add()**

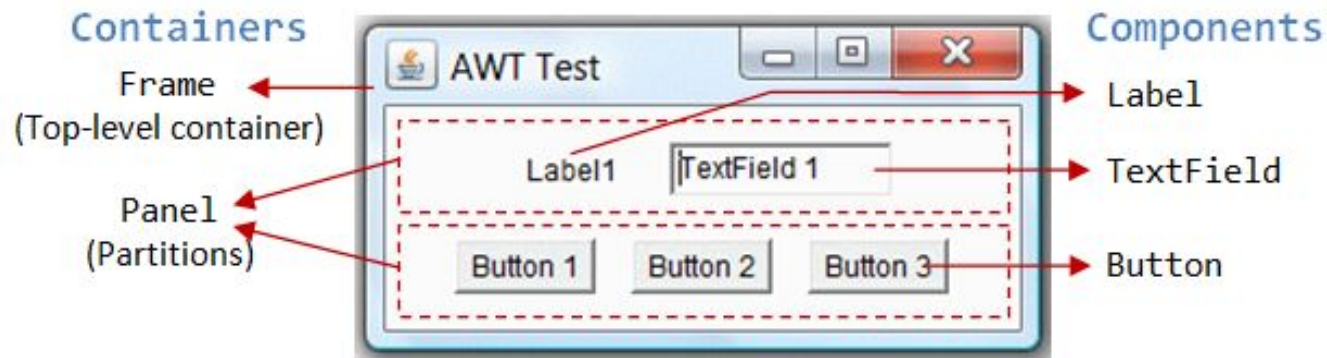
AWT - Tipos de Contenedores

Frame

- Es una ventana con título, es el contenedor de las aplicaciones de escritorio. Tiene borde, puede contener un menubar, agrandarse, achicarse y minimizarse.

Panel

- Debe estar contenido en otro Contenedor o dentro de una ventana de un navegador Web. Un Panel identifica un área rectangular dentro de la cual se pueden incluir otros componentes.



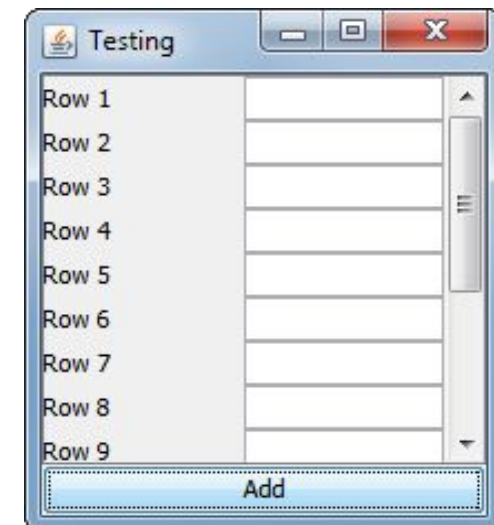
AWT - Tipos de Contenedores

Dialog

- Es una ventana simple y no puede tener una barra de menú. Puede moverse por la pantalla, pero no puede cambiar su tamaño. Puede ser modal o no-modal. Es ideal para capturar entradas del usuario.

ScrollPane

- Permite hacer “scroll” de una componente usando barras de desplazamiento.

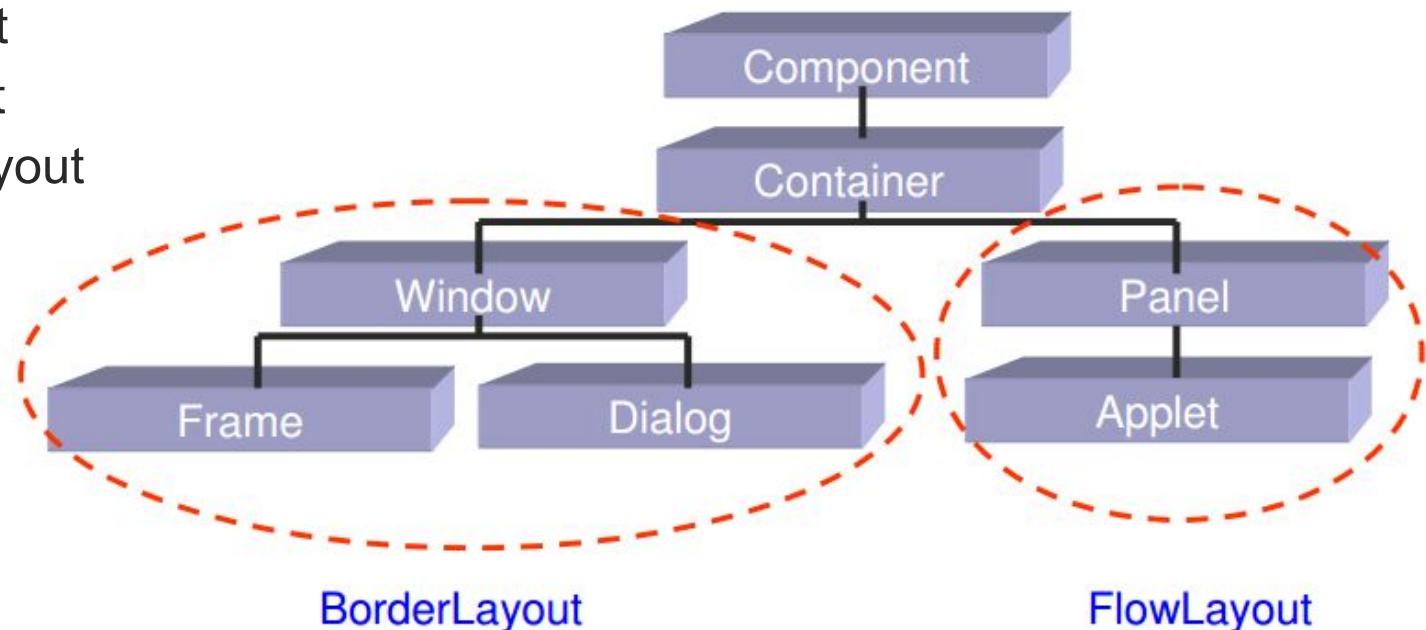


AWT - Disposiciones (Layouts)

- La ubicación y tamaño de un componente en un contenedor están determinados por el administrador de disposición o Layout. Cada contenedor mantiene una referencia a una instancia de un Layout
- El Administrador de Disposición toma el control de todos los componentes en el contenedor. El Layout es responsable de calcular el tamaño preferido del objeto en el contexto del tamaño de la pantalla actual. El programador no necesita definir el tamaño y posición de cada uno de los componentes de las ventanas, esa lógica la tiene incorporada el Administrador de Disposición.
- Sin embargo, es posible controlar el tamaño de los componentes y su ubicación manualmente. Para hacerlo se debe deshabilitar el administrador para ese contenedor así:
 - **`contenedor.setLayout(null);`**

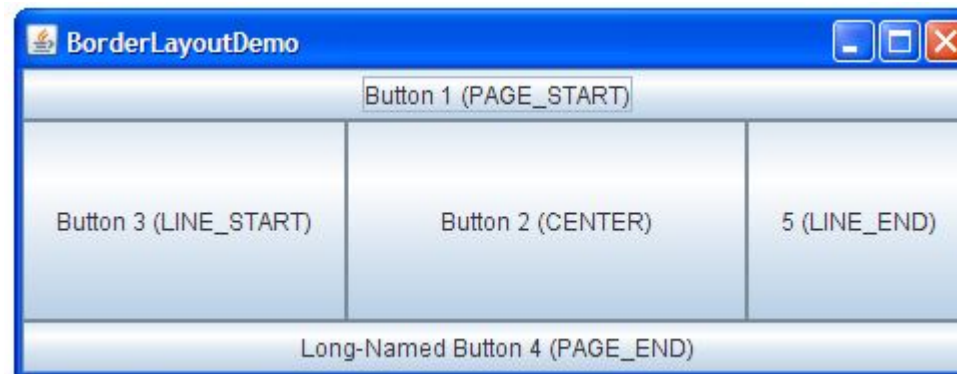
AWT - Tipos de Layouts

- Cada contenedor tiene asociado un administrador por defecto, el cual puede cambiarse invocando el método del contenedor `setLayout()`.
- Cada tipo de contenedor tiene asociado un objeto `Layout` por defecto, que puede cambiarlo utilizando el método `setLayout(nuevoLayout)`.
- Existen 5 tipos de administradores de disposición:
 - `BorderLayout`
 - `CardLayout`
 - `FlowLayout`
 - `GridBagLayout`
 - `GridLayout`



AWT - Tipos de Layouts

- **BorderLayout:** Organiza los componentes en 5 regiones: este, oeste, norte, sur y centro



- **CardLayout:** Trata a cada componente en el contenedor como una tarjeta. Sólo una tarjeta es visible por vez



AWT - Tipos de Layouts (2)

- **FlowLayout:** Layout por defecto. Organiza los componentes en un flujo direccional.



- **GridLayout:** Organiza los componentes en una grilla rectangular.



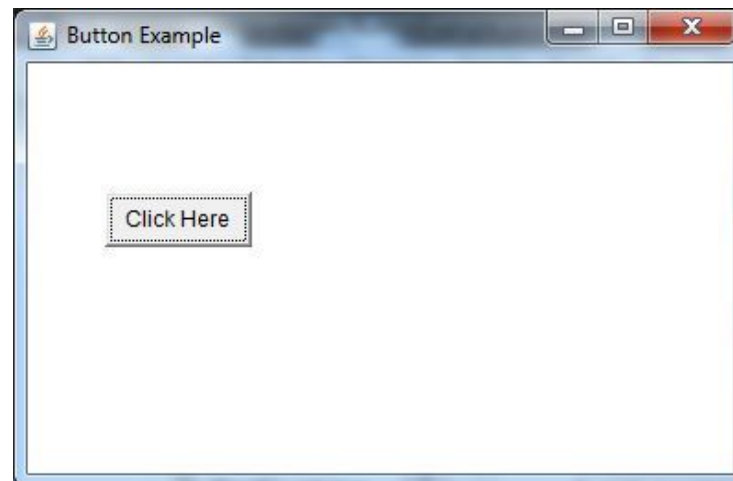
AWT - Tipos de Layouts (3)

- **GridBagLayout:** Es el más flexible. Alinea los componentes verticalmente, horizontalmente o a lo largo de su línea base sin necesidad de que todos los componentes sean del mismo tamaño.



AWT - Eventos

- Cuando el usuario realiza una acción a nivel de la interfaz de usuario (hace *click* o presiona una tecla), esto produce un evento. **Los eventos son objetos que describen lo que ha sucedido.** Hay una gran cantidad de clases de eventos para describir las diferentes categorías de acciones del usuario.
- Si creamos un botón (Button) y luego intentamos clickear en el mismo, por defecto nada sucederá. Por qué? Porque no manejamos los eventos, no indicamos qué hacer cuando se produce un evento.



AWT - Manejo de Eventos

- El manejo de eventos de la GUI está basado en el modelo de delegación. Dicho modelo se basa en objetos que disparan ú originan eventos llamados fuentes de eventos y objetos que escuchan y atienden esos eventos llamados escuchas de eventos o listeners.
- **¿Qué objetos pueden ser las fuentes de eventos?**
 - Las componentes básicas de GUI, como botones, listas, campos de texto, etc. son los objetos que disparan eventos y delegan el manejo de los mismos a otros objetos llamados escuchas.
 - La esencia de este modelo es simple: objetos que disparan eventos atendidos por los listeners (clases que implementan interfaces listeners).

AWT - Manejo de Eventos (2)

Eventos

- Un evento es generado por una componente como consecuencia de una acción iniciada por un usuario (presionar un botón, seleccionar un ítem de una lista, etc).

Fuentes de Eventos

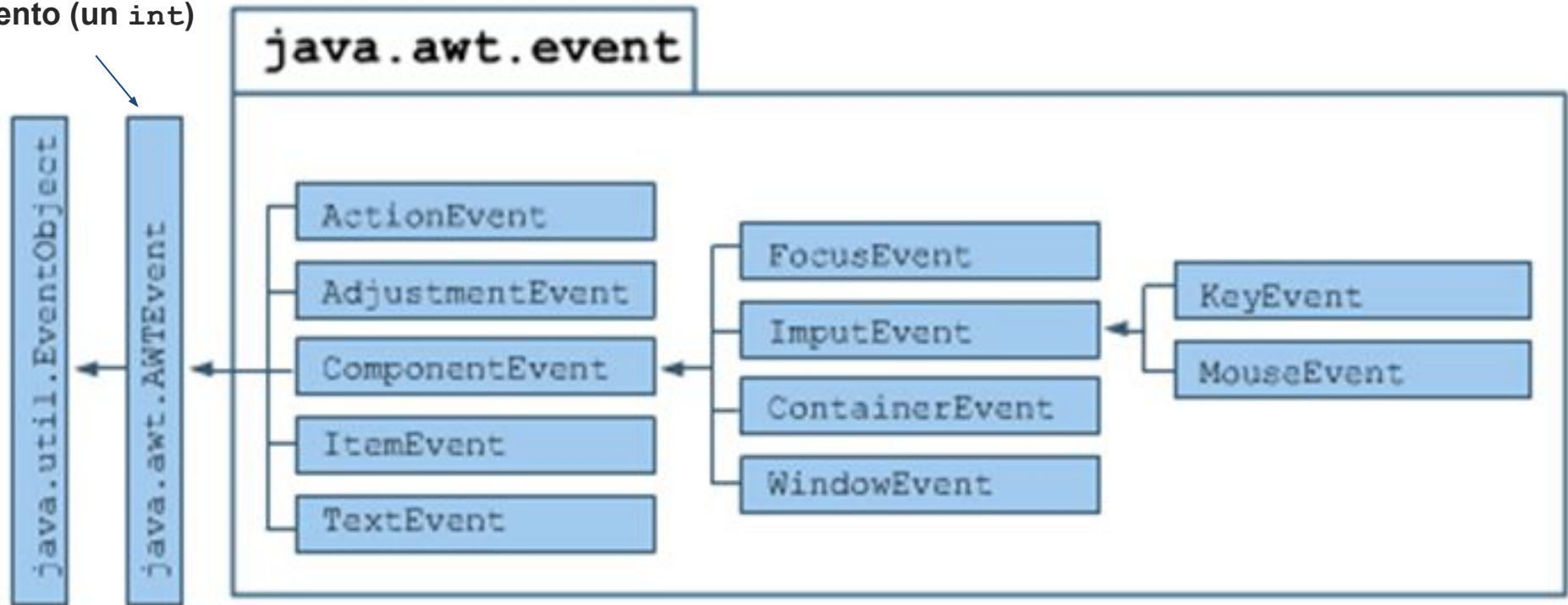
- Las componentes de GUI, son las que generan eventos. Estas componentes además implementan 2 métodos para que los listeners registren interés o quiten el registro de los eventos que ellas generan

Manejadores de Eventos

- Un listener es un objeto que implementa una determinada interface. Los métodos de estos objetos, reciben como parámetro un `AWTEvent` específico, que contiene información del evento y qué objeto AWT lo disparó.
- Sobre una componente AWT, se pueden registrar uno o más escuchas. El orden en que los escuchas son notificados del evento, es indefinido.

AWT - Tipos de Eventos

Mantiene un ID del
evento (un `int`)



Mantiene al objeto que
originó el evento

La clase `java.util.EventObject` junto con las interfaces analizadas, constituyen el fundamento del modelo de delegación de eventos.

AWT - Tipos de Eventos (2)

- JAVA provee clases que representan los eventos de la GUI. Cada una de estas clases agrupa un conjunto de eventos relacionados. La mayoría de estas clases codifica los eventos particulares mediante constantes
- Ejemplo: Constantes de la clase WindowEvent que representan los eventos particulares de ventana: activar, desactivar, cerrar, abrir, minimizar y maximizar, etc.

MouseEvent

```
MOUSE_CLICKED  
MOUSE_ENTERED  
MOUSE_EXITED  
MOUSE_PRESSED  
MOUSE_RELEASED
```

WindowEvent

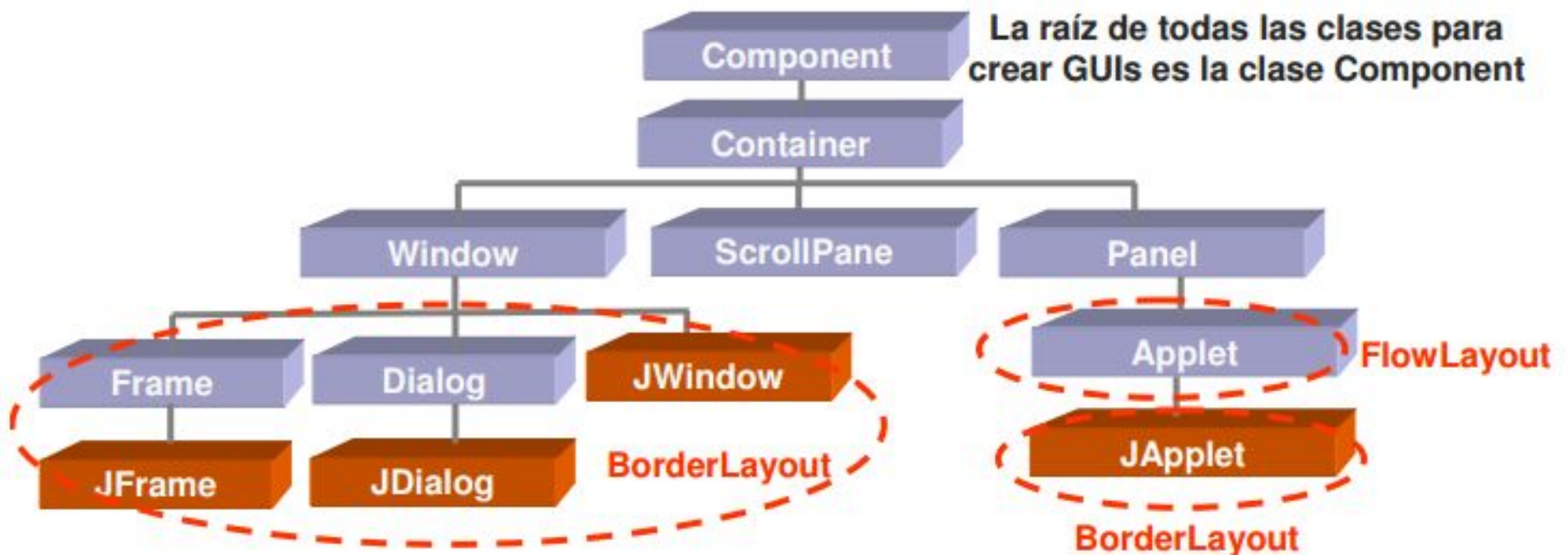
```
WINDOW_CLOSING  
WINDOW_CLOSED  
WINDOW_ACTIVATED  
WINDOW_DEACTIVATED  
WINDOW_OPENED  
WINDOW_ICONIFIED  
WINDOW_DEICONIFIED  
WINDOW_GAINED_FOCUS  
WINDOW_LOST_FOCUS
```

Swing

- JFC (Java Foundation Classes) – Swing es una segunda generación de herramientas para el desarrollo de GUIs.
- Está incluida en la J2SDK.
- Tiene muchas mejoras con respecto a AWT sobre el cual está construido.
- Las componentes Swing reemplazan a las AWT y permiten construir interfaces de usuario de alta calidad.
- Swing provee múltiples componentes de GUI que no están presentes en el AWT: fichas, barras de herramientas, tablas, árboles, cajas de diálogo, etc.

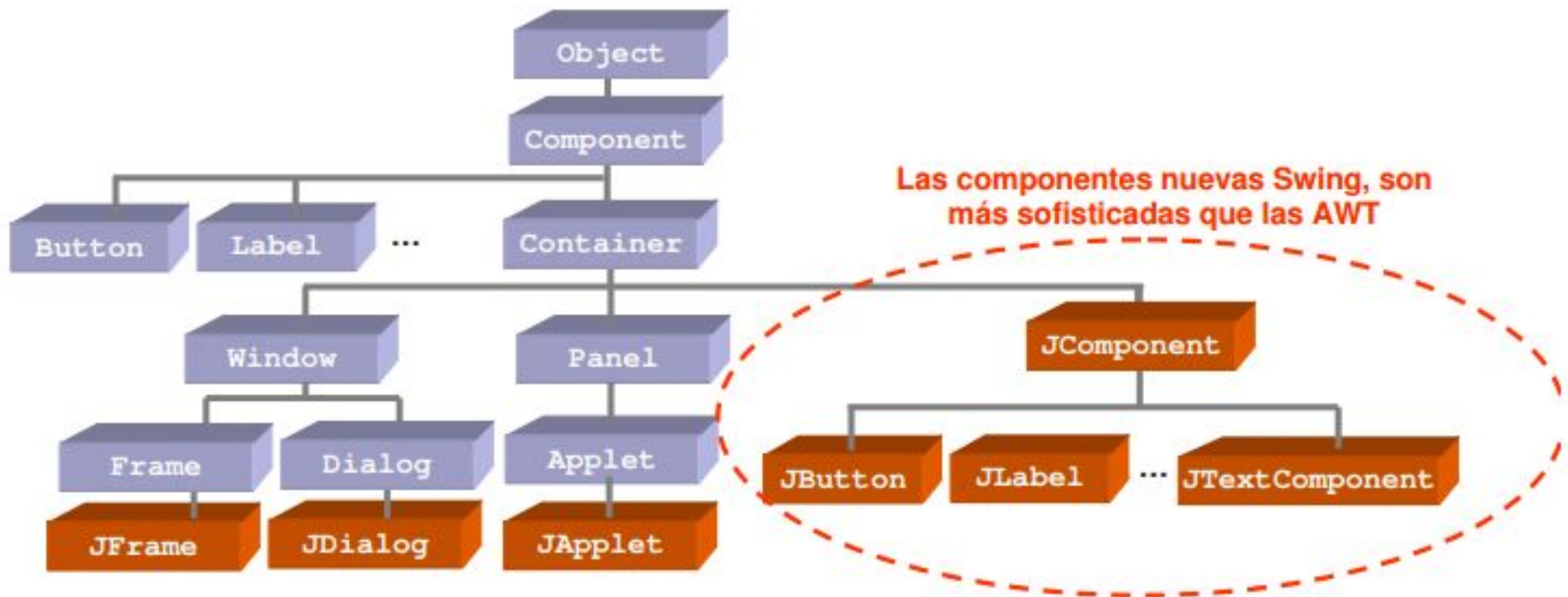
Swing - Jerarquía de contenedores

- Los contenedores Swing a nivel de ventana, son subclases de los contenedores AWT
- Los Layouts son los mismos que en AWT con excepción de Applet que cambió de layout



Swing - Jerarquía de componentes

- Las componentes básicas de Swing son subclases de “Container” (las AWT son subclases de Component) por lo tanto están capacitadas para contener a otras componentes.



Swing - Conversión desde AWT

Básicamente para transformar una aplicación AWT a Swing, se debe:

1. Cambiar las librería que deben importarse: **import javax.swing.*;**
2. Cambiar los nombres de las componentes: cada nombre de clase de componente y de contenedor, debe anteponerse una J

```
Button → JButton  
TextField → JTextField  
Frame → JFrame  
Panel → JPanel  
Checkbox → JCheckBox
```

3. Cambiar el receptor de los `||` los **setLayout(LayoutManager)** o **add(Component)**. El manejo de eventos de componentes es idéntico al manejo de AWT.

```
Frame.add(Component) → JFrame.getContentPane().add(Component)  
Frame.setLayout(LayoutManager) → JFrame.getContentPane().setLayout(LayoutManager)
```



Swing - Conversión desde AWT (2)

¿Por qué debemos cambiar el receptor de los llamados a los métodos `setLayout(LayoutManager)` o `add(Component)` ?

- Todos los contenedores Swing de nivel de ventana (**JApplet**, **JFrame**, **JDialog**, **JWindow**) contienen una instancia de la clase **JRootPane**. Esta instancia incluye un contenedor llamado **content pane** (instancia de **JPanel**), donde estarán contenidas todas las componentes.



```
public class PruebaJFrame {  
    private JFrame f = new JFrame();  
    private JButton b = new JButton("Hola");  
    public PruebaJFrame () {  
        Container contentPane = f.getContentPane();  
        contentPane.setLayout(new BorderLayout());  
        contentPane.add(b);  
        f.pack();  
        f.show();  
    }  
    public static void main(String[] args) {  
        PruebaJFrame vent = new PruebaJFrame();  
    }  
}
```

Swing - Conversión desde AWT: Ejemplo

```
package vistas;
import java.awt.*;

public class AwtLayoutExample {
    private Frame f;
    private Button b1;
    private Button b2;

    public AwtLayoutExample() {
        f = new Frame("Ejemplo de GUI");
        b1 = new Button("Presionar");
        b2 = new Button("No presionar");
    }

    public void launchFrame() {
        f.setLayout(new FlowLayout());
        f.add(b1);
        f.add(b2);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[]) {
        AwtLayoutExample guiWindow = new AwtLayoutExample();
        guiWindow.launchFrame();
    }
}
```



Swing - Conversión desde AWT: Ejemplo

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

public class SwingLayoutExample2 {
    private JFrame f;
    private JButton b1;
    private JButton b2;

    public SwingLayoutExample2() {
        f = new JFrame("Ejemplo de GUI con Swing");
        b1 = new JButton("Presionar");
        b2 = new JButton("No presionar");
    }

    public void launchFrame() {
        f.getContentPane().setLayout(new FlowLayout());
        f.getContentPane().add(b1);
        f.getContentPane().add(b2);
        f.pack();
        f.setVisible(true);
    }

    public static void main(String args[]) {
        SwingLayoutExample2 guiWindow = new SwingLayoutExample2();
        guiWindow.launchFrame();
    }
}
```



Se obtiene el objeto
ContentPane, para luego
configurarlo un Layout y
agregarle componentes

Swing - Ejemplos de Componentes



Campos de texto (JTextField)



Botones (JButton)



Bordes (JBorder)



Selectores
(JTabbedPane)



Paneles dimensionables
(JSplit Pane)



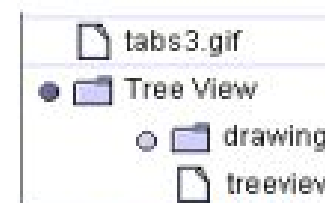
Tablas (JTable)



Menús
(JMenuBar y JMenuItem)



Barra de Herramientas
(JToolBar)



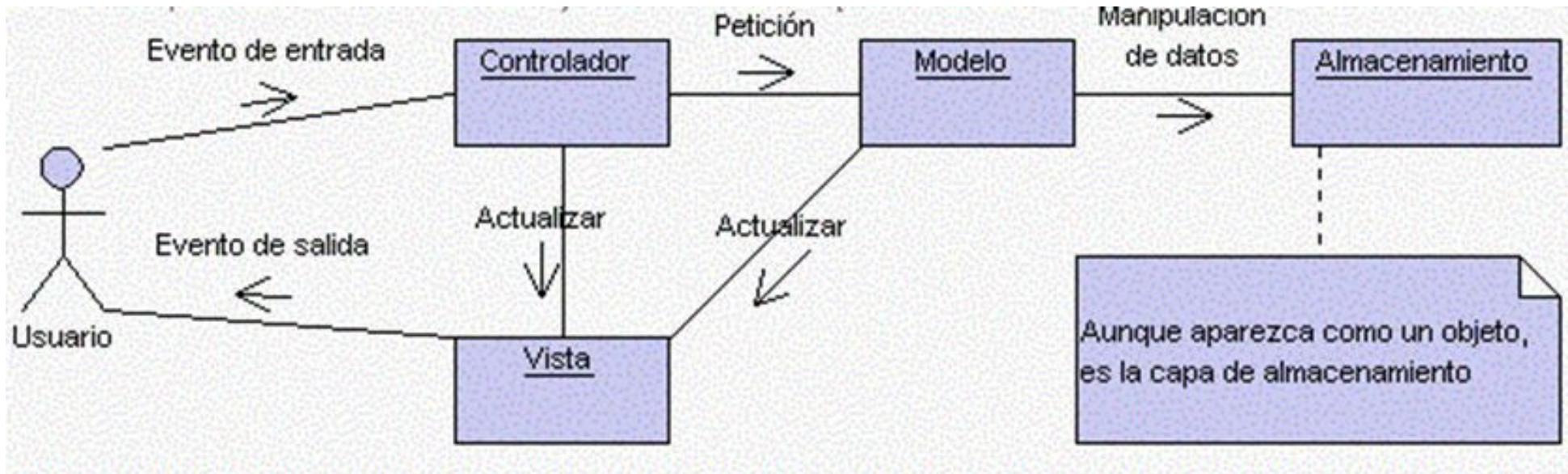
Arboles (Jtree)

Swing - Modelo-Vista-Controlador

- Para el diseño de aplicaciones con interfaces sofisticados se utiliza el patrón de diseño **Modelo Vista Controlador (MVC)**.
 - La lógica de un interfaz de usuario cambia con más frecuencia que los datos que tenemos almacenados y la lógica de negocio.
 - Si realizamos un diseño complicado, es decir, una mezcla de componentes de interfaz y de negocio, entonces cuando necesitemos cambiar la interfaz, tendremos que modificar trabajosamente los componentes de negocio à mayor trabajo y más riesgo de error.
- MVC fomenta un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.
- **Elementos del patrón:**
 - **Modelo:** datos y reglas de negocio
 - **Vista:** muestra la información del modelo al usuario
 - **Controlador:** gestiona las entradas del usuario

Swing - Modelo-Vista-Controlador (2)

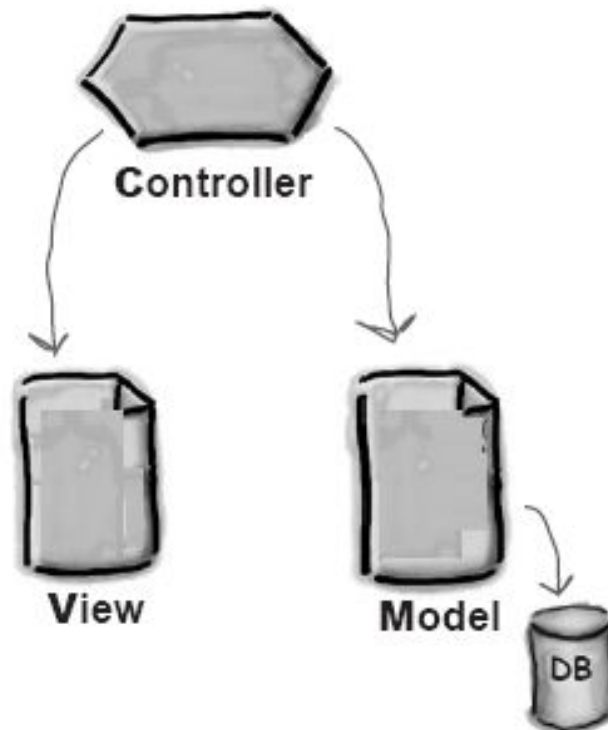
- En el siguiente diagrama se puede observar que los eventos de entrada del usuario son procesados por el controlador, quien hace invocaciones al modelo y actualiza la Vista, que será mostrada al usuario.
- ¿Cuál es el rol de cada una de las componentes del MVC?



Swing - Modelo-Vista-Controlador (3)

Recibe los eventos de entrada (un click, un cambio en un campo de texto, etc.). Contiene reglas de gestión de eventos, del tipo "Si el evento Z, entonces la acción W". Estas acciones pueden suponer peticiones al modelo o a las vistas. Una de estas peticiones a las vistas puede ser una llamada al método "Actualizar()". Una petición al modelo puede ser "Obtener_tiempo_de_entrega(nueva_orden_de_venta)".

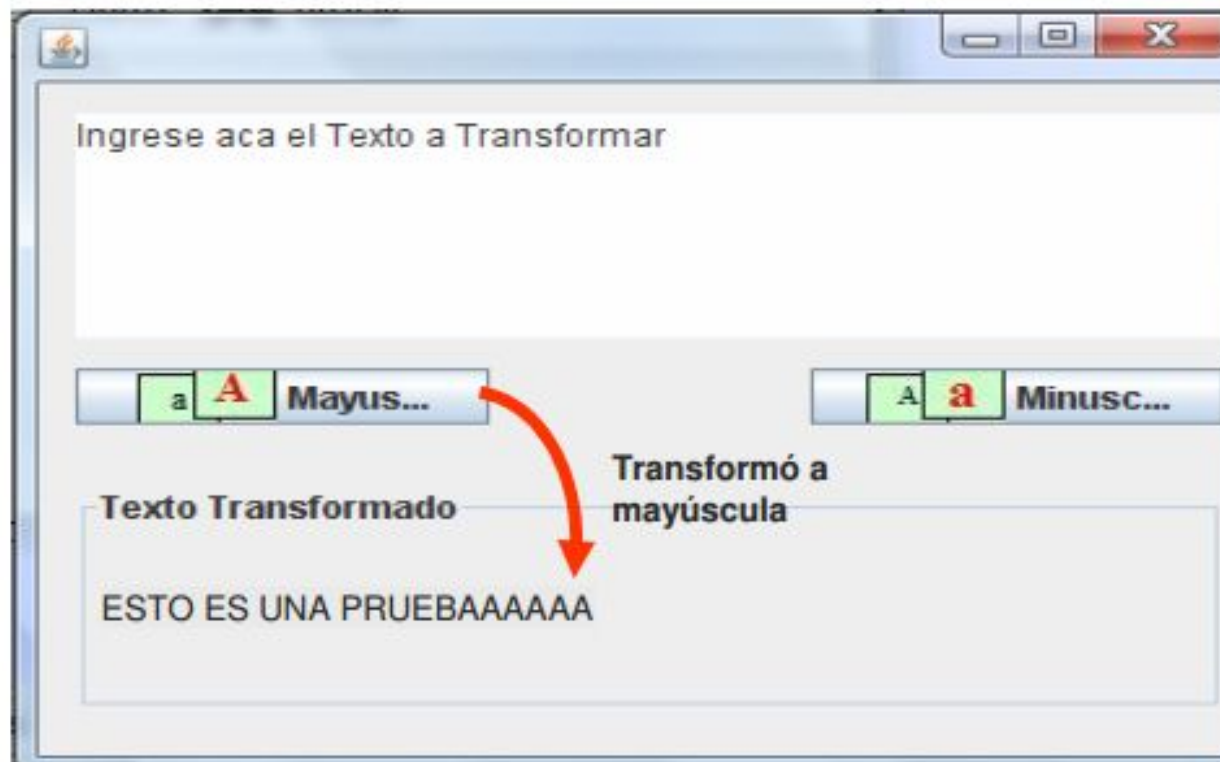
Recibe datos del modelo y los muestra al usuario. Tienen un registro de su controlador asociado. Pueden dar el servicio de "Actualización()", para que sea invocado por el controlador o por el modelo



Accede a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento. Define las reglas de negocio (la funcionalidad del sistema). Lleva un registro de las vistas y controladores del sistema. Un ejemplo de regla puede ser: "Si la mercadería pedida no está en el almacén, consultar el tiempo de entrega al proveedor".

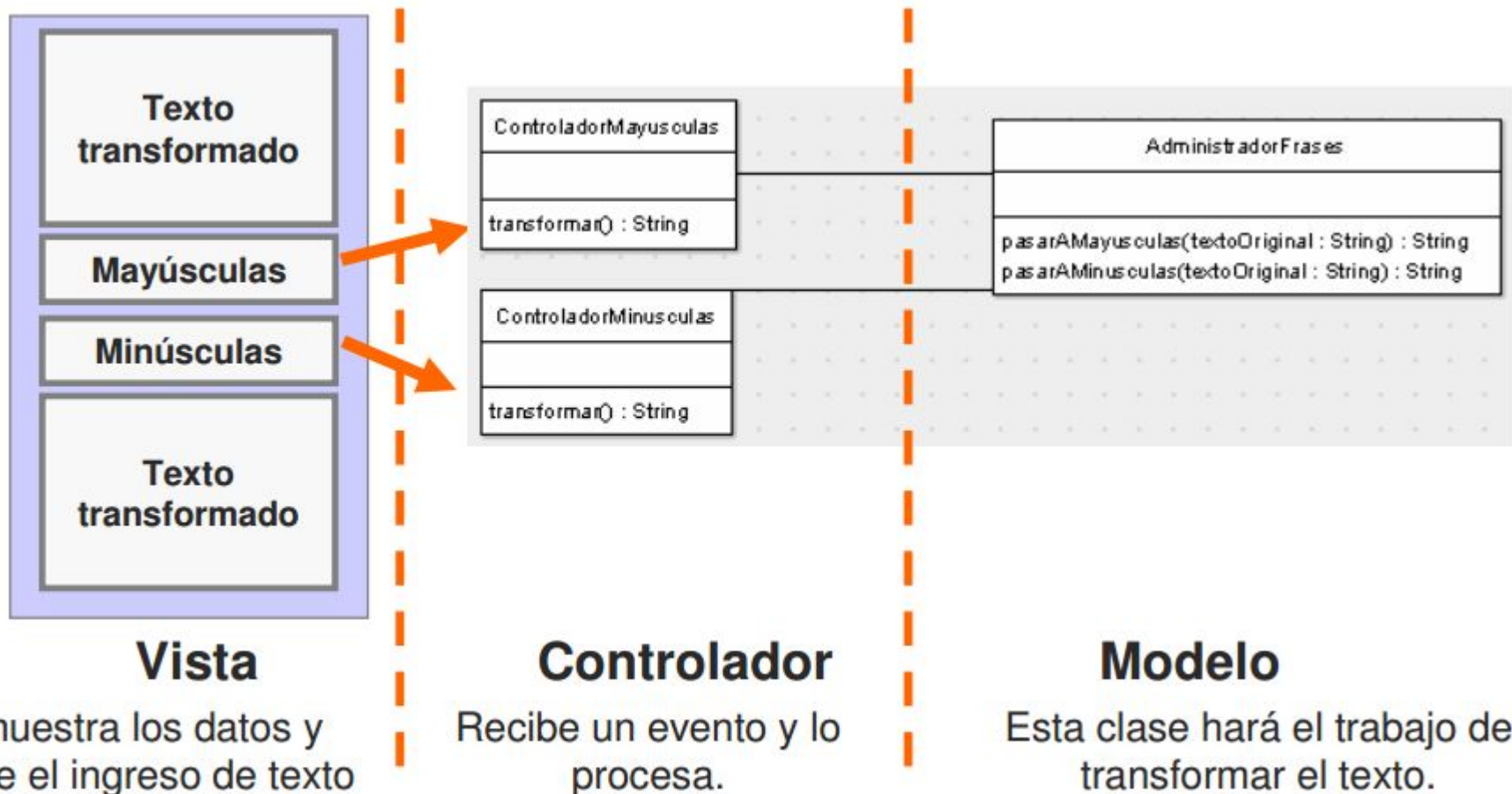
Swing - MVC Ejemplo

- En esta app de ejemplo, se desarrolla una interfaz gráfica que sea capaz de tomar un texto ingresado por el usuario y darle la opción de poder pasar todo el texto a mayúsculas o pasar todo el texto a minúsculas. Una posible vista es esta:



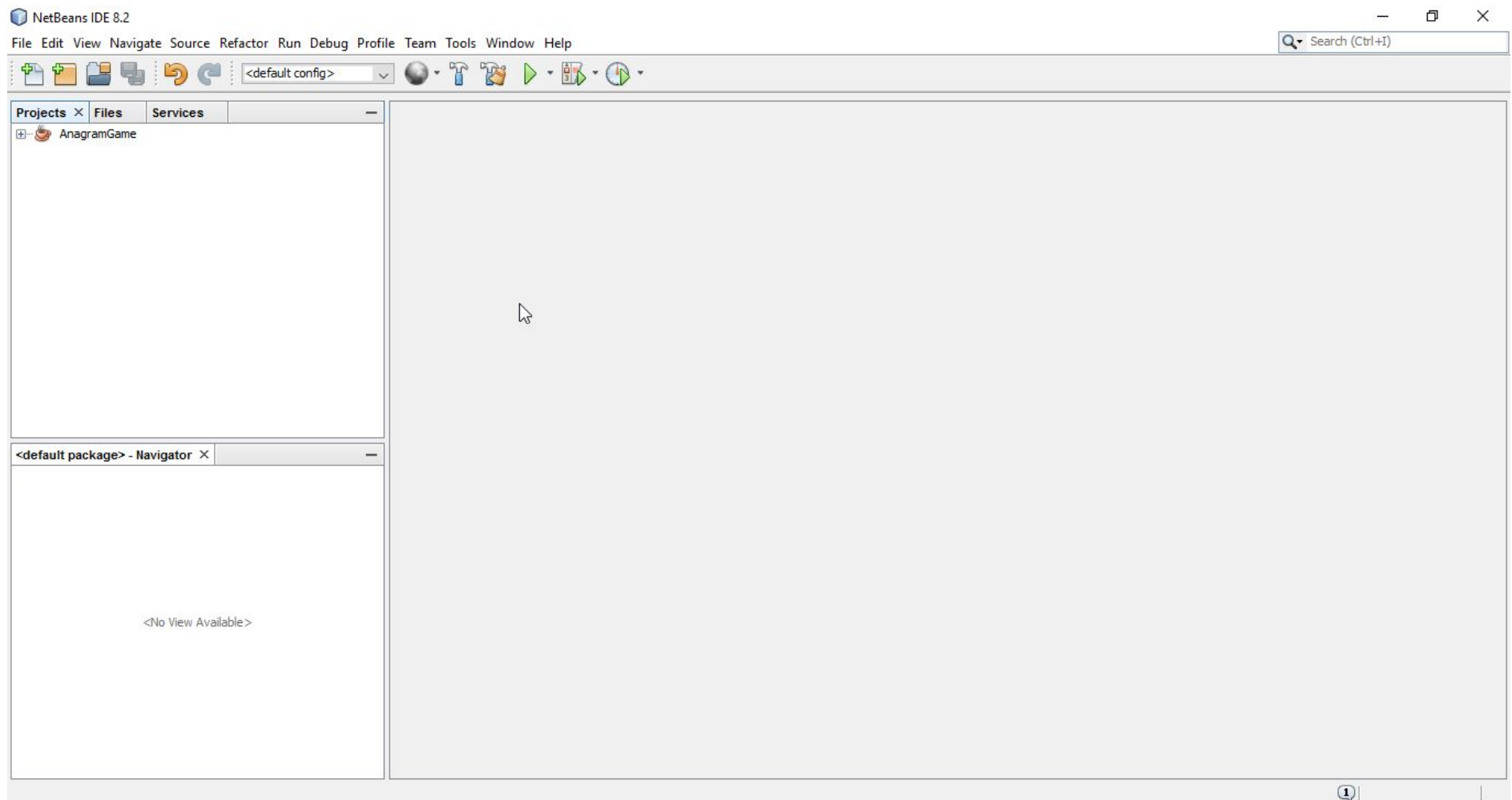
Swing - MVC Ejemplo (2)

- Usando el patrón MVC podemos diseñar/dividir nuestra aplicación en las siguientes partes:



Swing - Netbeans

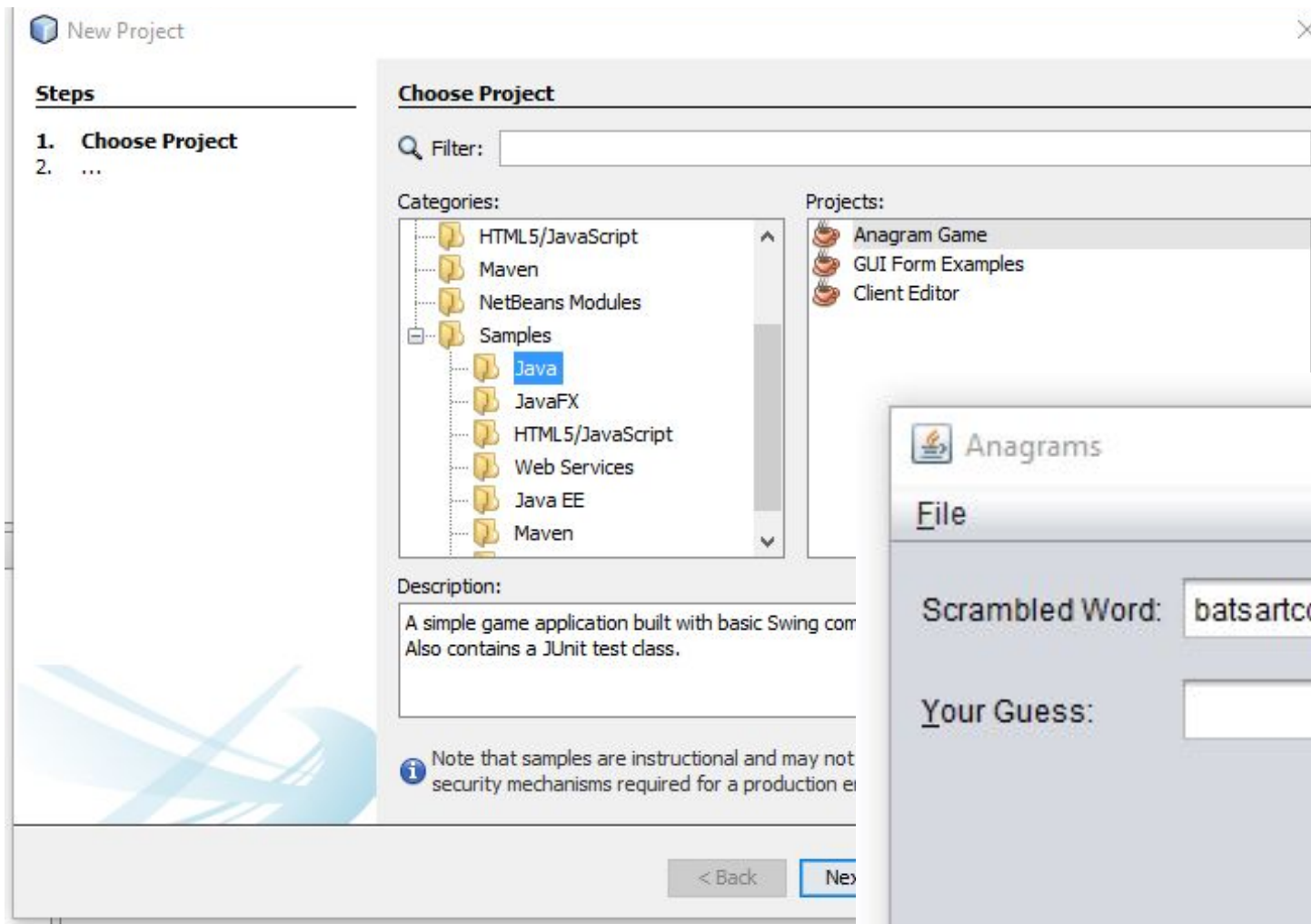
- NetBeans IDE es un entorno integrado de desarrollo gratuito, open-source y multi-plataforma con soporte para el lenguaje Java.



Swing - Netbeans GUI Builder

- Netbeans incluye un editor visual de interfaces Java basadas en AWT/Swing denominado **GUI Builder**. Las principales características son:
 - Creación visual de interfaces mediante un esquema Drag & Drop
 - Generación automática de código que permite al desarrollador concentrarse en la lógica de aplicación en lugar de detalles visuales y complejidad del Layout manager.
 - Diseño libre, mediante el uso de posicionamiento absoluto, desligando al desarrollador de la configuración de los layouts
 - Sincronismo instantáneo, mantiene alineados al código y la interfaz visual del GUI Builder. La aplicación se ve exactamente igual en tiempo de ejecución y diseño.
 - Asistencia para posicionamiento, sugiriendo posiciones y alineación de nuevos componentes en base a los componentes que ya pertenecen a un contenedor

Swing - Ejemplos de Apps visuales

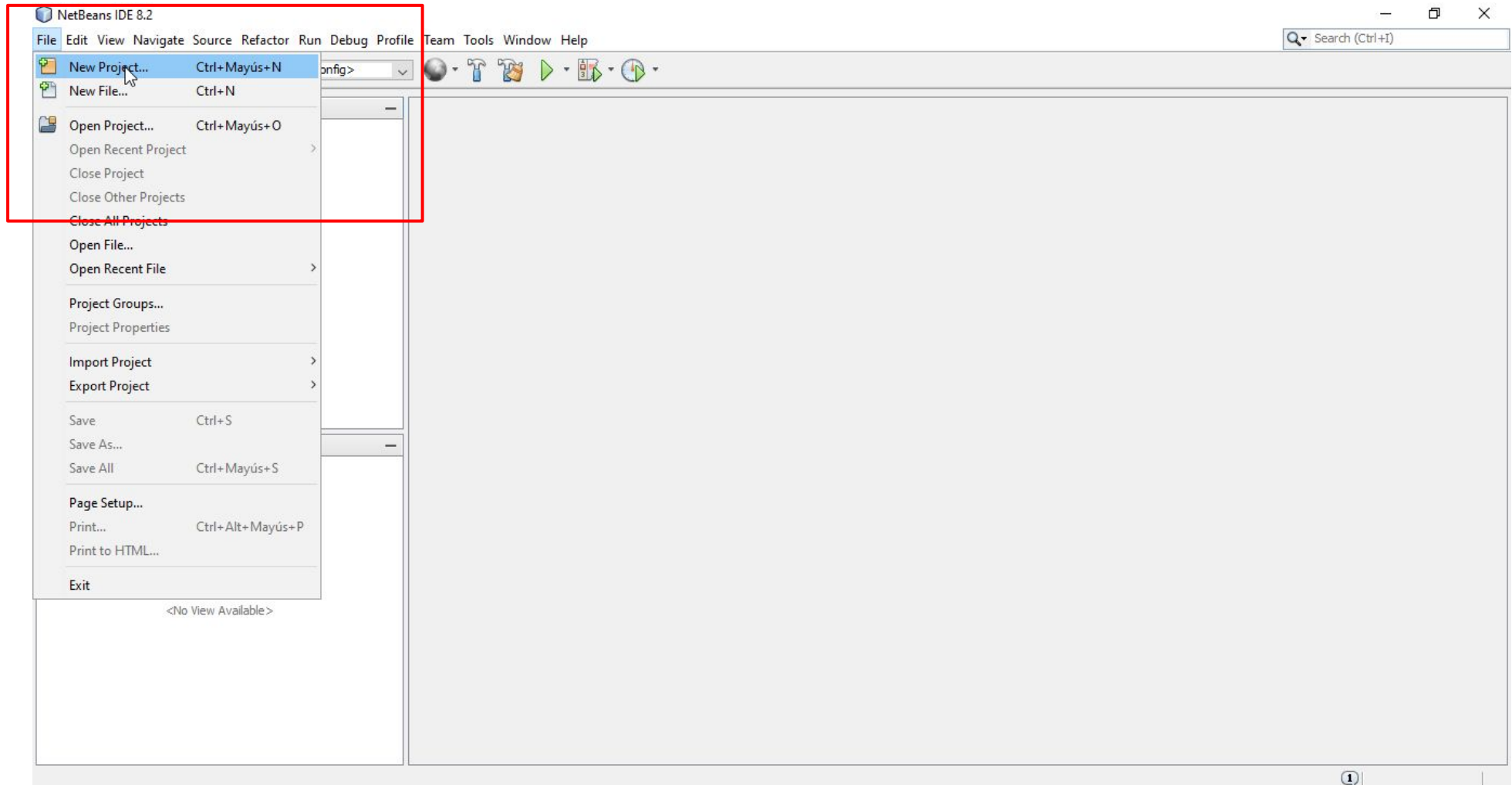


Swing - Ejemplo: Celsius Converter

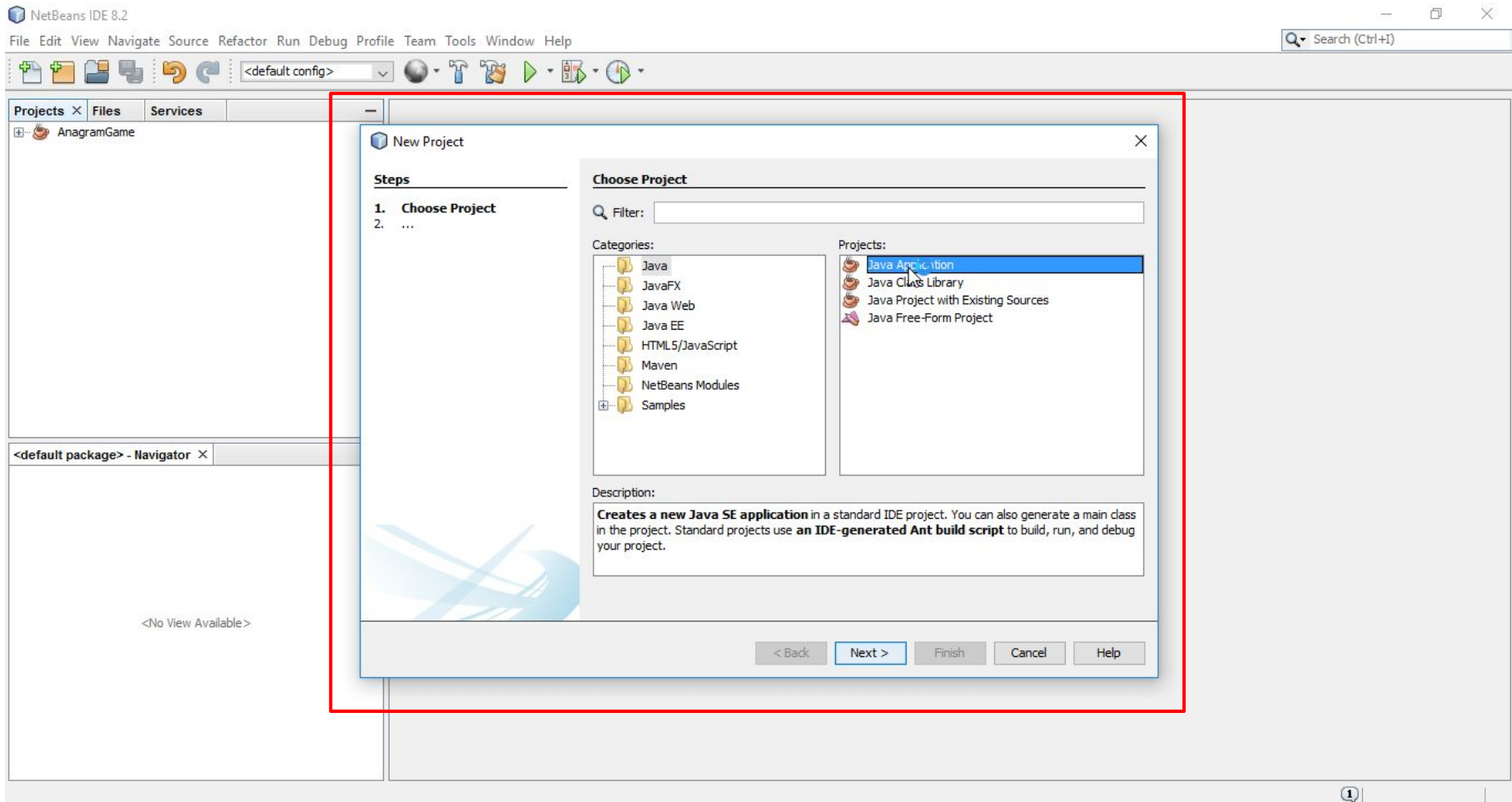
- Aplicación simple para convertir una temperatura indicada en grados celsius a fahrenheit
- Consta de un input de texto, un botón y 2 labels



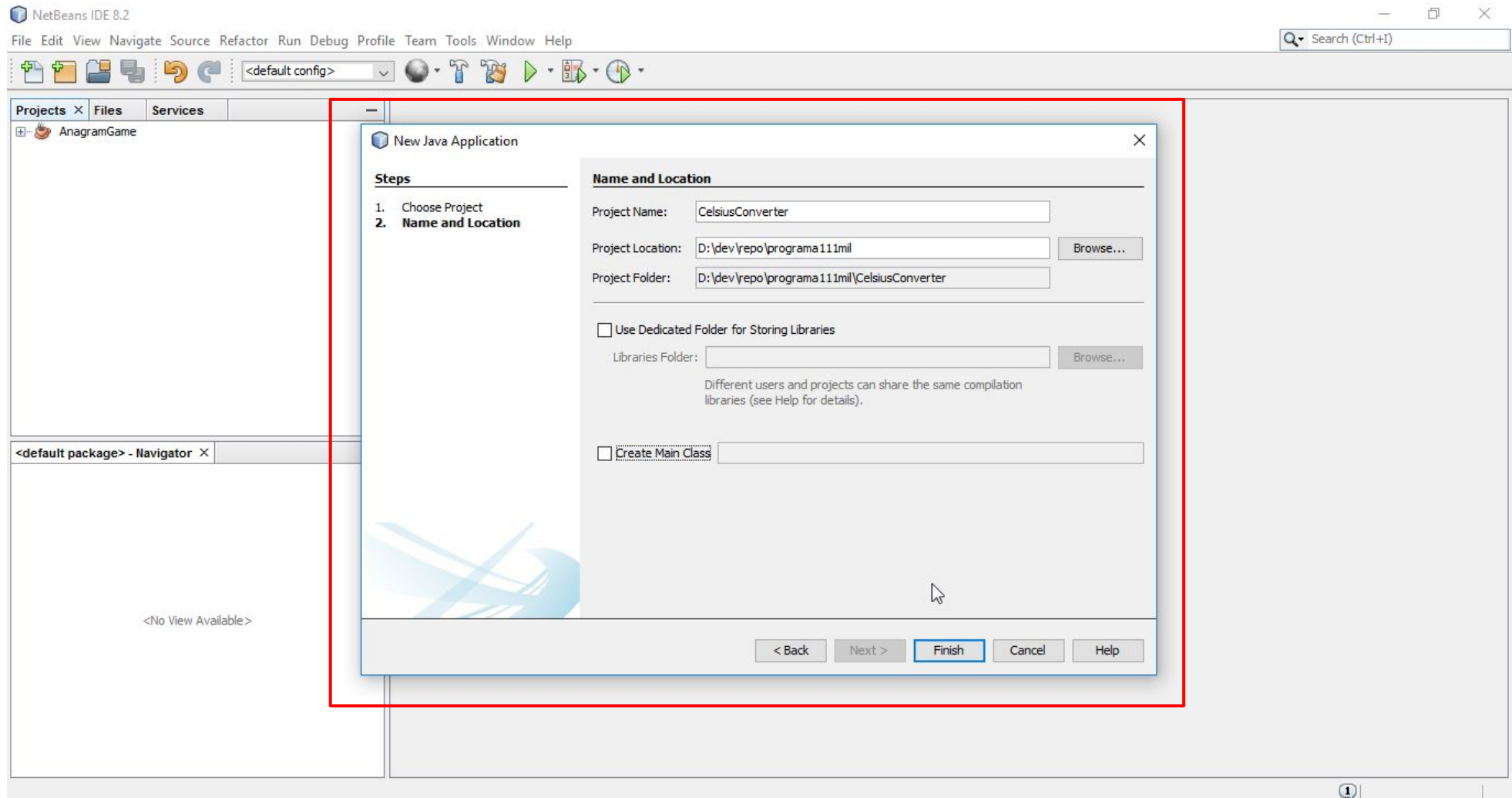
Netbeans - Nuevo Proyecto



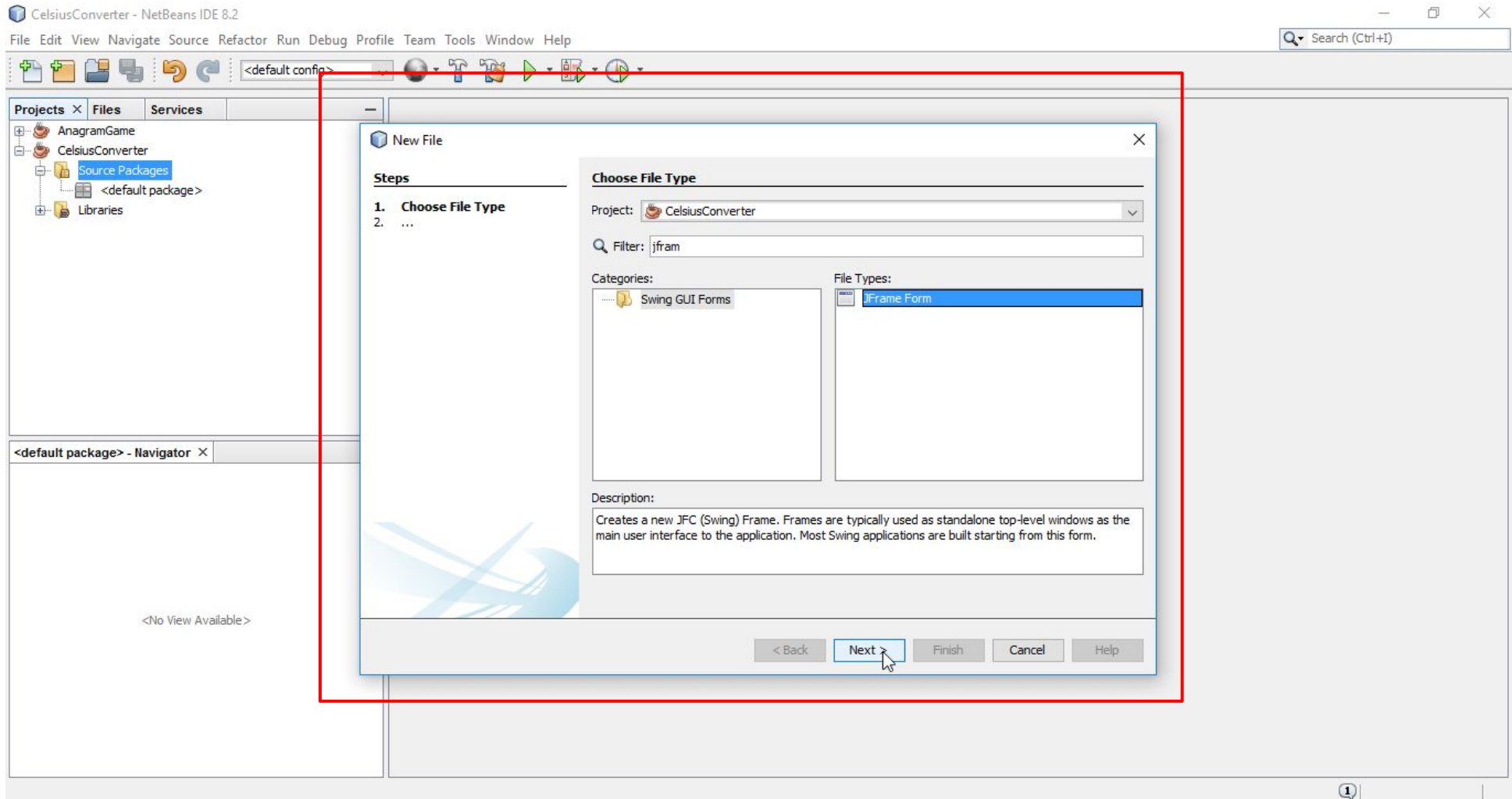
Netbeans - Nuevo Proyecto (1)



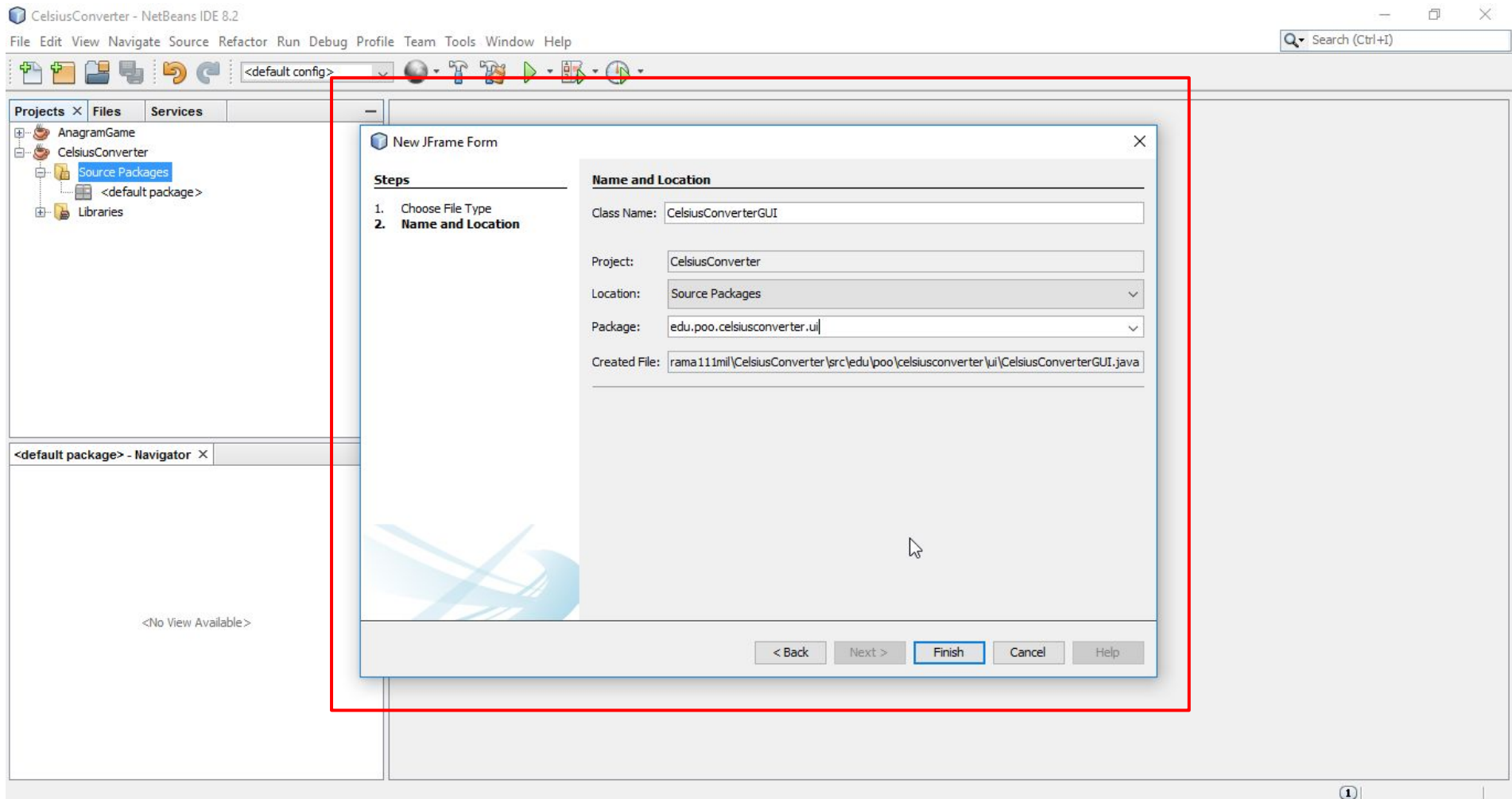
Netbeans - Nuevo Proyecto (2)



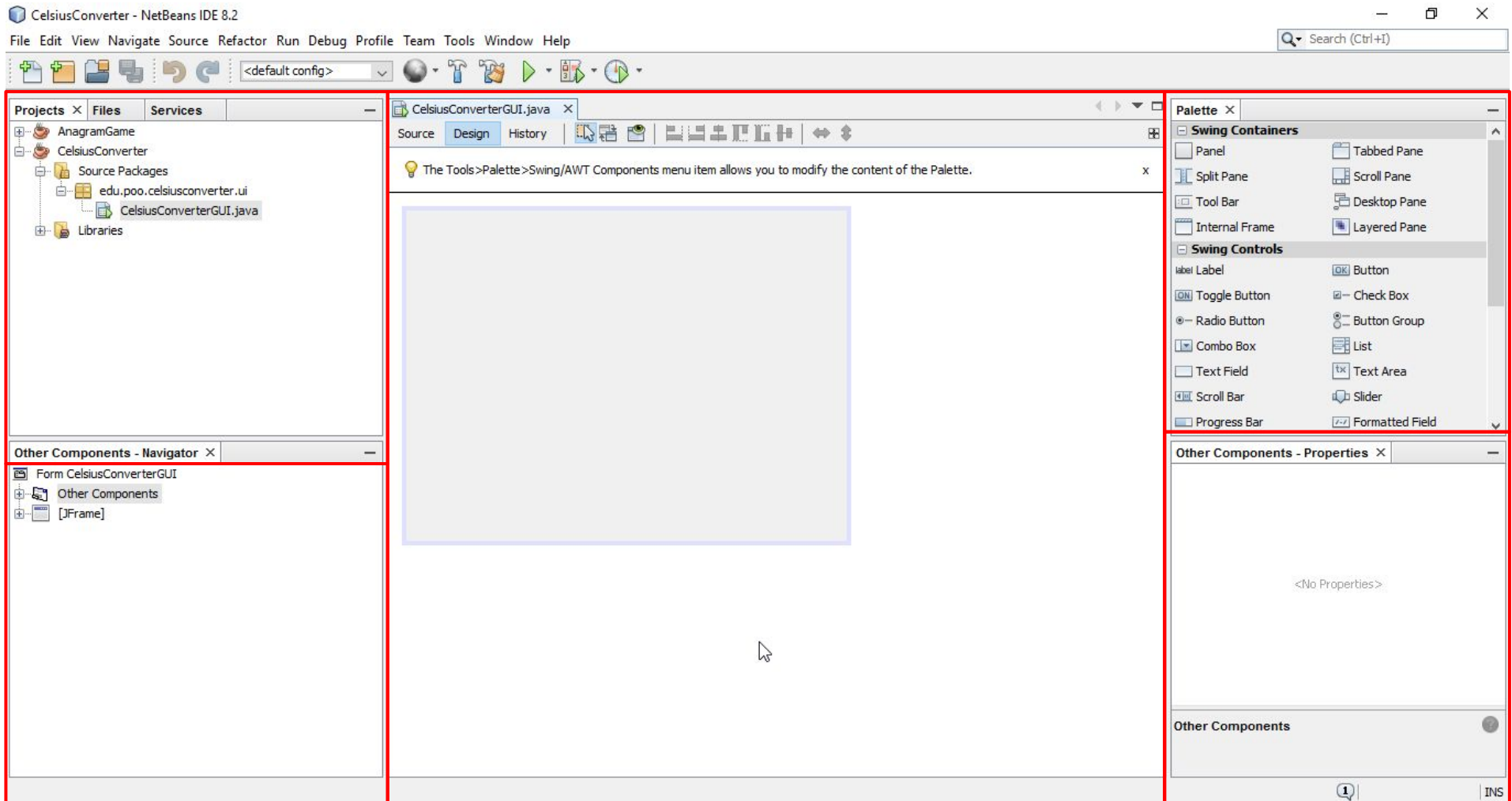
Netbeans - Crear JFrame



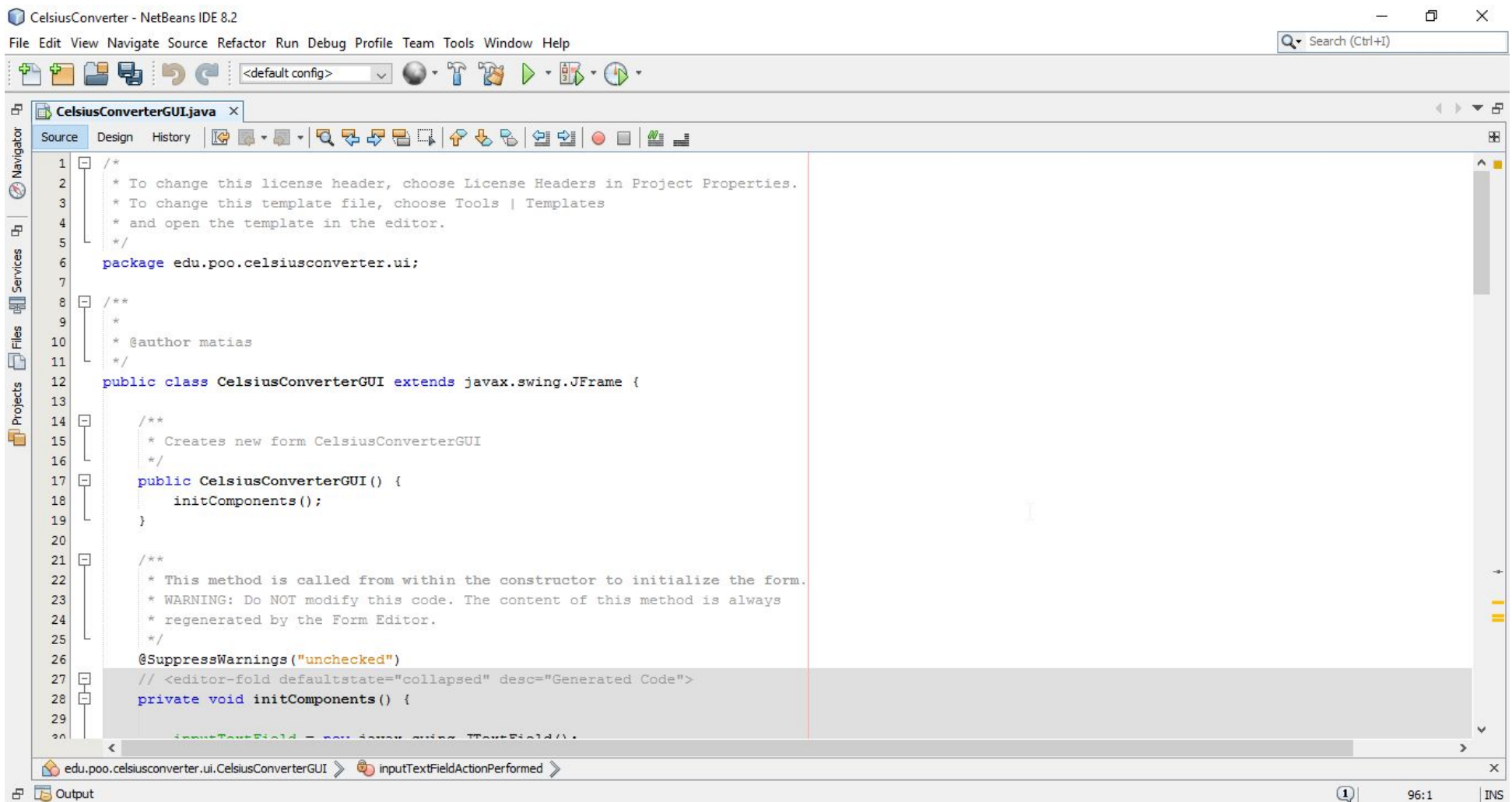
Netbeans - Crear JFrame (2)



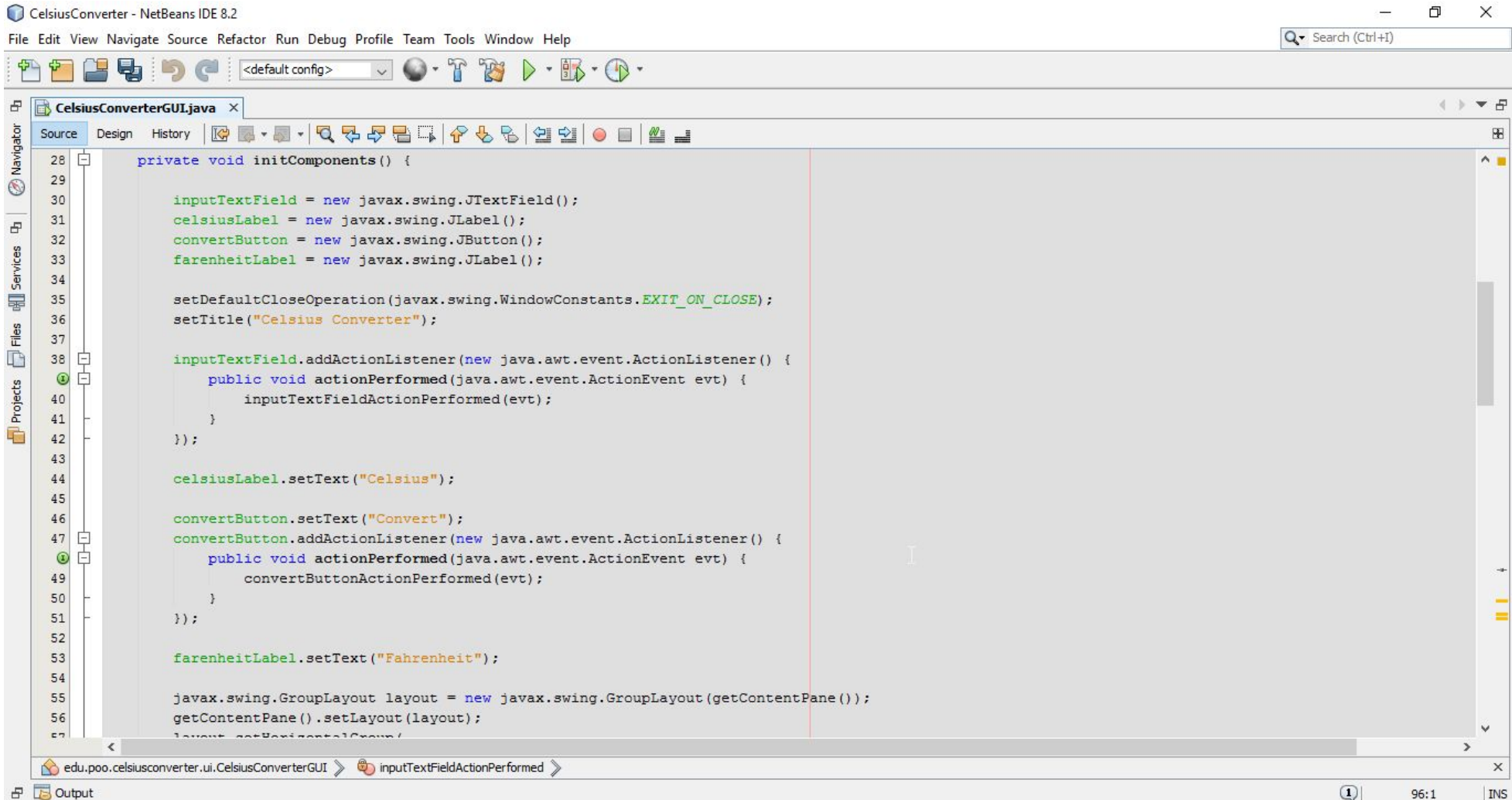
Netbeans - GUI Vista de Diseño



Netbeans - Vista de Código



Netbeans - Vista de Código (2)

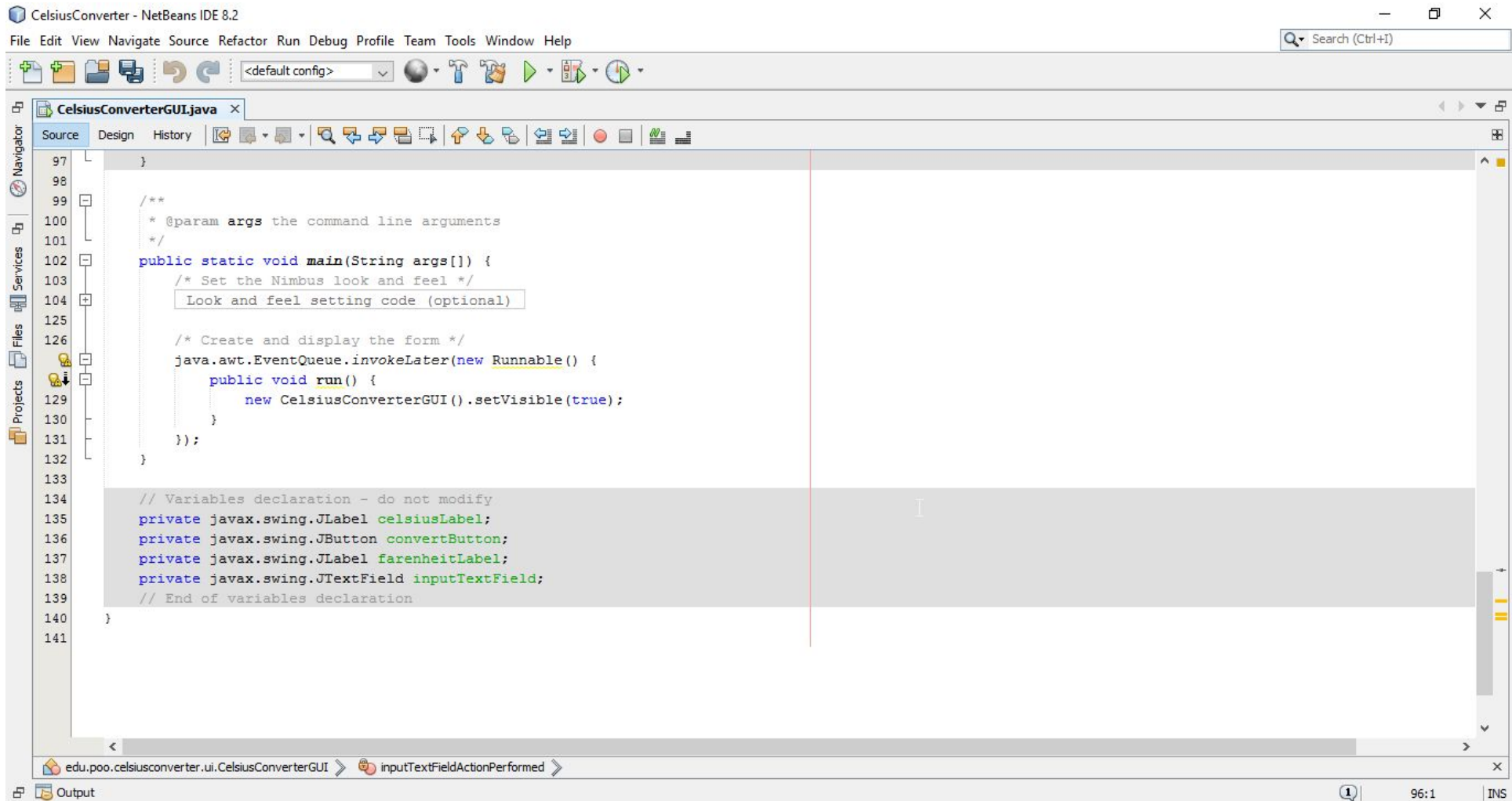


```
28 private void initComponents() {
29
30     inputTextField = new javax.swing.JTextField();
31     celsiusLabel = new javax.swing.JLabel();
32     convertButton = new javax.swing.JButton();
33     fahrenheitLabel = new javax.swing.JLabel();
34
35     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
36     setTitle("Celsius Converter");
37
38     inputTextField.addActionListener(new java.awt.event.ActionListener() {
39         public void actionPerformed(java.awt.event.ActionEvent evt) {
40             inputTextFieldActionPerformed(evt);
41         }
42     });
43
44     celsiusLabel.setText("Celsius");
45
46     convertButton.setText("Convert");
47     convertButton.addActionListener(new java.awt.event.ActionListener() {
48         public void actionPerformed(java.awt.event.ActionEvent evt) {
49             convertButtonActionPerformed(evt);
50         }
51     });
52
53     fahrenheitLabel.setText("Fahrenheit");
54
55     javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
56     getContentPane().setLayout(layout);
57     layout.setHorizontalGroup(
```

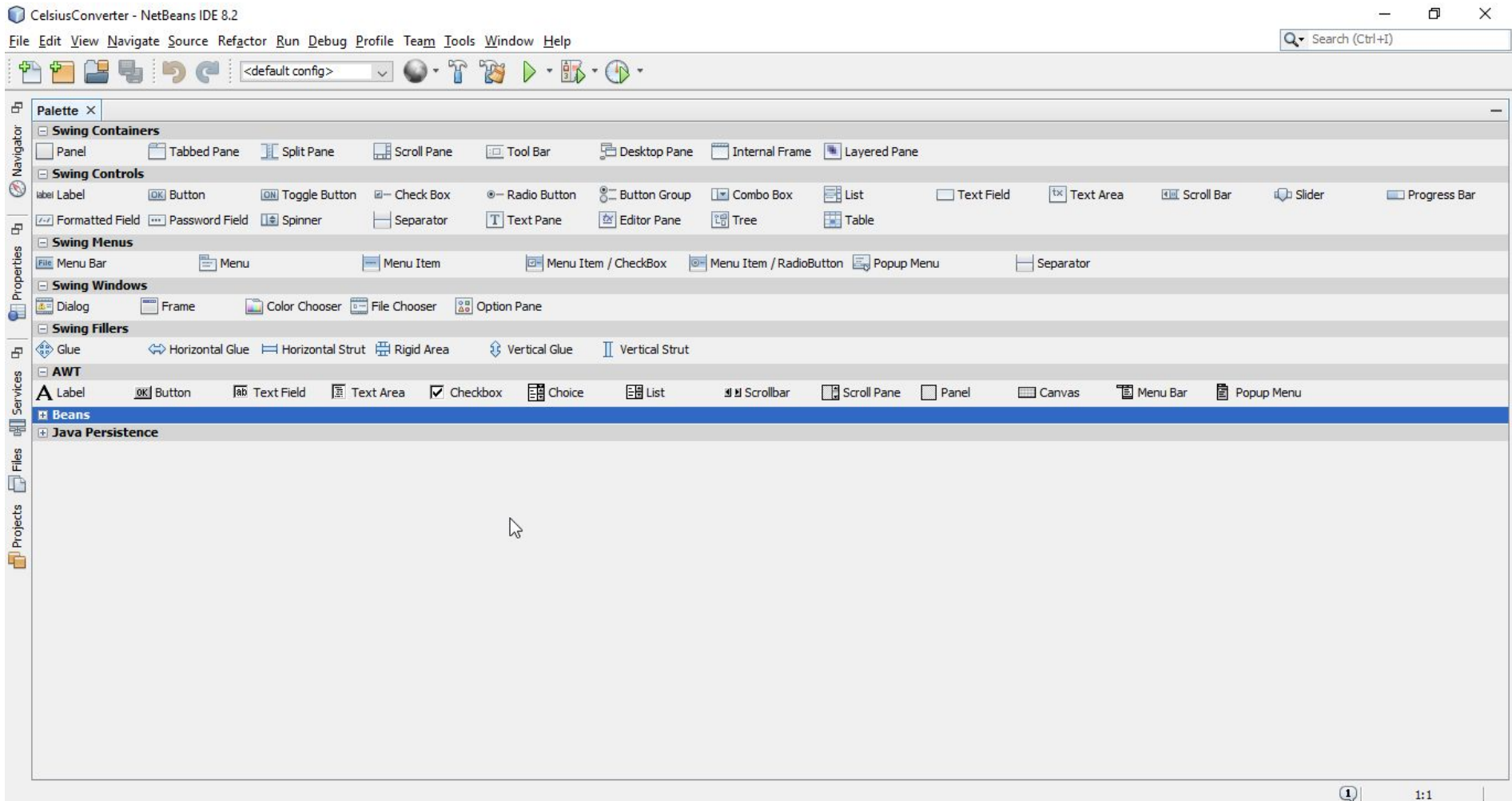
edu.poo.celsiusconverter.ui.CelsiusConverterGUI | inputTextFieldActionPerformed

Output 96:1 INS

Netbeans - Vista de Código (3)



Netbeans - Paleta de Componentes



Netbeans - Editor de propiedades

CelsiusConverter - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

<default config>

[JFrame] - Properties

Properties Binding Events Code

Properties

defaultCloseOperation EXIT_ON_CLOSE

title

Other Properties

alwaysOnTop ☐

alwaysOnTopSupported ☒

autoRequestFocus ☒

background ☐ [240,240,240]

bounds <Not Set>

cursor Cursor Por defecto

enabled ☒

extendedState 0

focusCycleRoot ☒

focusTraversalPolicy <default>

focusTraversalPolicyProvider ☐

focusable ☒

focusableWindowState ☒

font null

foreground null

graphics <none>

iconImage <none>

iconImages <default>

insets [0, 0, 0, 0]

location <Not Set>

locationByPlatform ☐

maximizedBounds null

maximumSize [2147483647, 2147483647]

[JFrame]

1:1

Netbeans - Creamos Componentes

CelsiusConverter - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects Files Services

- AnagramGame
- CelsiusConverter
 - Source Packages
 - edu.poo.celsiusconverter.ui
 - CelsiusConverterGUI.java
 - Libraries

CelsiusConverterGUI.java

Source Design History

The gaps between components can be selected and adjusted via mouse, or edited in a dialog. Double-click the gap or invoke *Edit Layout Space* from context menu to show the customization dialog.

Design View

Components in Design View:

- TextField1
- Label1

[JFrame] - Navigator

- Form CelsiusConverterGUI
 - Other Components
 - [JFrame]
 - TextField1 [JTextField]
 - Label1 [JLabel]

Palette

- Swing Controls
 - Label
 - Toggle Button
 - Radio Button
 - Combo Box
 - Text Field
 - Scroll Bar
 - Progress Bar
 - Password Field
 - Separator
 - Editor Pane
 - Table
 - Button
 - Check Box
 - Button Group
 - List
 - Text Area
 - Slider
 - Formatted Field
 - Spinner
 - Text Pane
 - Tree

[JFrame] - Properties

Properties Binding Events Code

Properties

- defaultCloseOperation: EXIT_ON_CLOSE
- title: Celsius Converter

Other Properties

- alwaysOnTop: ☐
- alwaysOnTopSupported: ☒
- autoRequestFocus: ☒
- background: ☐ [240,240,240]
- bounds: <Not Set>
- cursor: Cursor Por defecto
- enabled: ☒

[JFrame]

1:1

Netbeans - Creamos Componentes (2)

CelsiusConverter - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects Files Services

- AnagramGame
- CelsiusConverter
 - Source Packages
 - edu.poo.celsiusconverter.ui
 - CelsiusConverterGUI.java
 - Libraries

CelsiusConverterGUI.java

Source Design History

To add a component multiple times, select it via click in palette and then Shift-click on design canvas.

Design Canvas:

- jTextField1
- jLabel1
- jButton1
- jLabel2

Palette

- Swing Controls
 - Label
 - Toggle Button
 - Radio Button
 - Combo Box
 - Text Field
 - Scroll Bar
 - Progress Bar
 - Password Field
 - Separator
 - Editor Pane
 - Table
 - OK Button
 - Check Box
 - Button Group
 - List
 - Text Area
 - Slider
 - Formatted Field
 - Spinner
 - Text Pane
 - Tree

Other Components - Navigator

- Form CelsiusConverterGUI
 - Other Components
 - JFrame
 - jTextField1 [JTextField]
 - label jLabel1 [JLabel]
 - OK jButton1 [JButton]
 - label jLabel2 [JLabel]

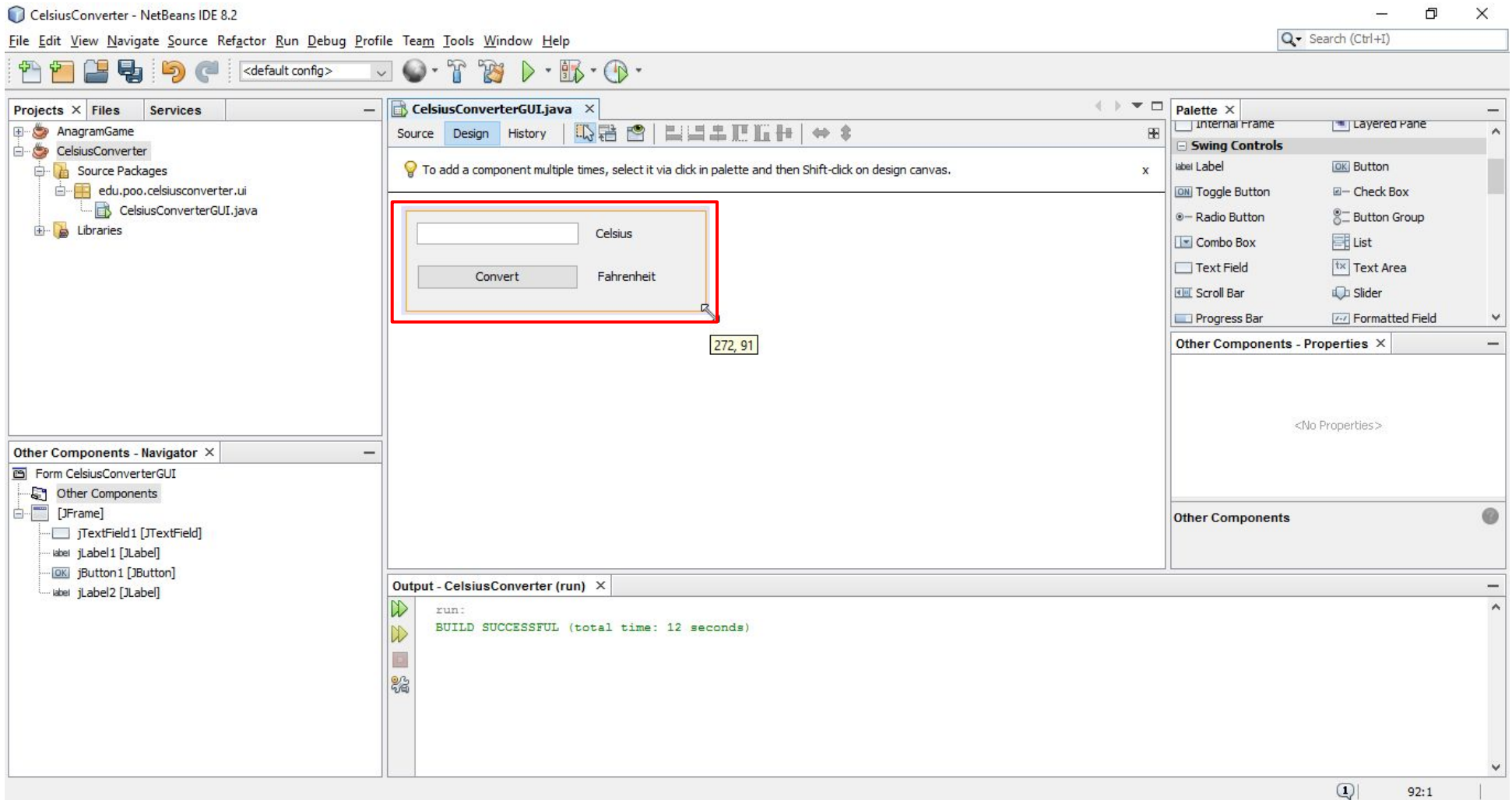
Other Components - Properties

<No Properties>

Other Components

1:1

Netbeans - Redimensionamos el JFrame



CelsiusConverter - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects Files Services

- AnagramGame
- CelsiusConverter
 - Source Packages
 - edu.poo.celsiusconverter.ui
 - CelsiusConverterGUI.java
 - Libraries

CelsiusConverterGUI.java

Source Design History

To add a component multiple times, select it via click in palette and then Shift-click on design canvas.

Celsius

Convert Fahrenheit

272, 91

Palette

- Internal Frame
- Layered Pane
- Swing Controls
 - Label Label
 - Toggle Button
 - Radio Button
 - Combo Box
 - Text Field
 - Scroll Bar
 - Progress Bar
 - OK Button
 - Check Box
 - Button Group
 - List
 - Text Area
 - Slider
 - Formatted Field

Other Components - Properties

<No Properties>

Other Components

Other Components - Navigator

- Form CelsiusConverterGUI
 - Other Components
 - JFrame
 - JTextField1 [JTextField]
 - JLabel1 [JLabel]
 - JButton1 [JButton]
 - JLabel2 [JLabel]

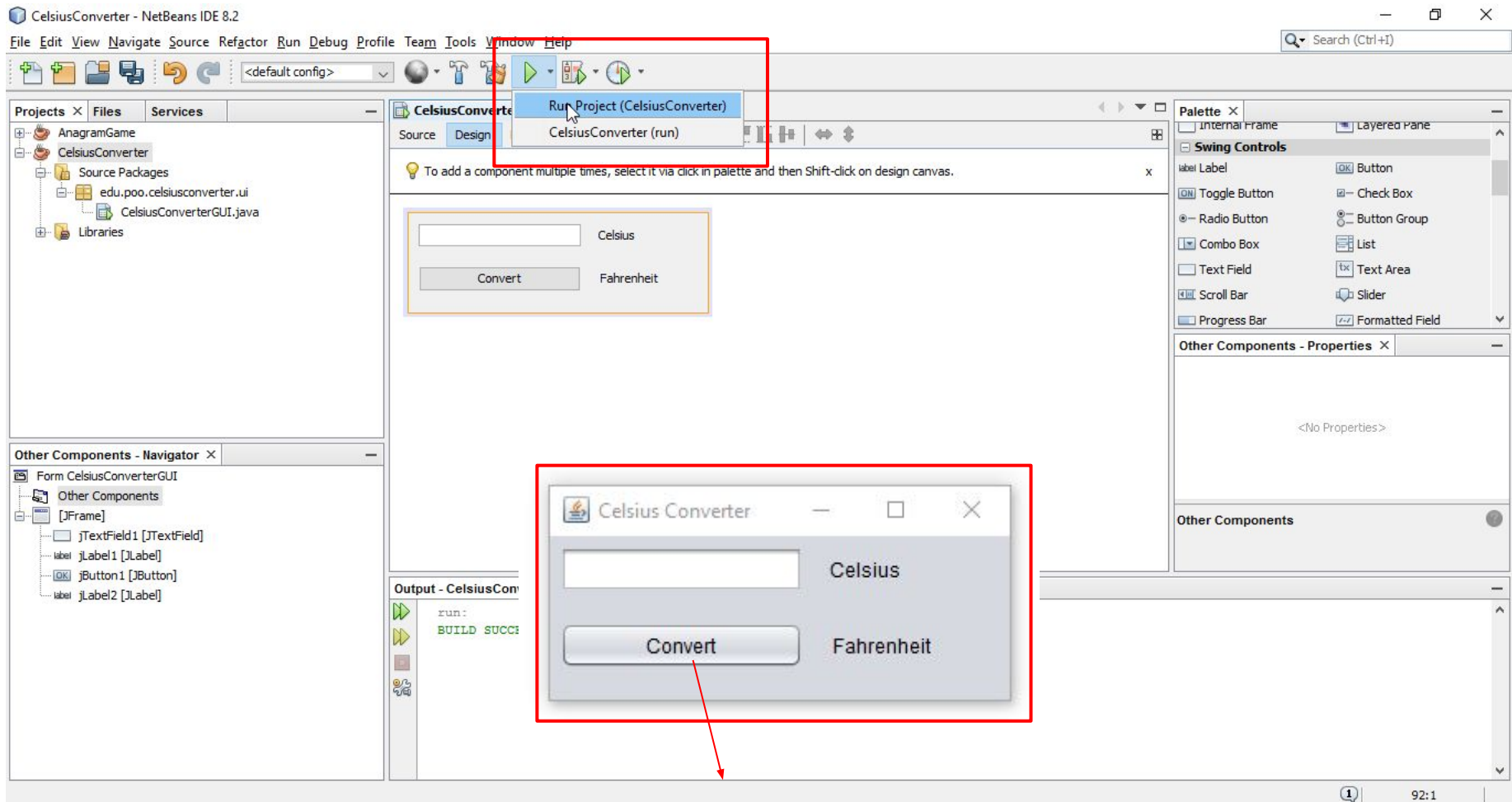
Output - CelsiusConverter (run)

run:

BUILD SUCCESSFUL (total time: 12 seconds)

92:1

Netbeans - Ejecución de la App



El botón no hace nada!

Swing - Eventos

- Similar al manejo de eventos en AWT

Acción

Pulsar un botón

Cambio del texto

Pulsar Intro en un campo de texto

Selección de un nuevo elemento

Selección de elemento(s)

Pulsar una casilla de verificación

Pulsar un botón de radio

Selección de una opción de menú

Mover la barra de desplazamiento

Abrir, cerrar, minimizar,
maximizar o cerrar la ventana

Objeto origen

JButton

JTextComponent

JTextField

JComboBox

JList

JCheckBox

JRadioButton

JMenuItem

JScrollBar

JWindow

Tipo de evento

ActionEvent

TextEvent

ActionEvent

ItemEvent

ActionEvent

ListSelection-
Event

ItemEvent

ActionEvent

ItemEvent

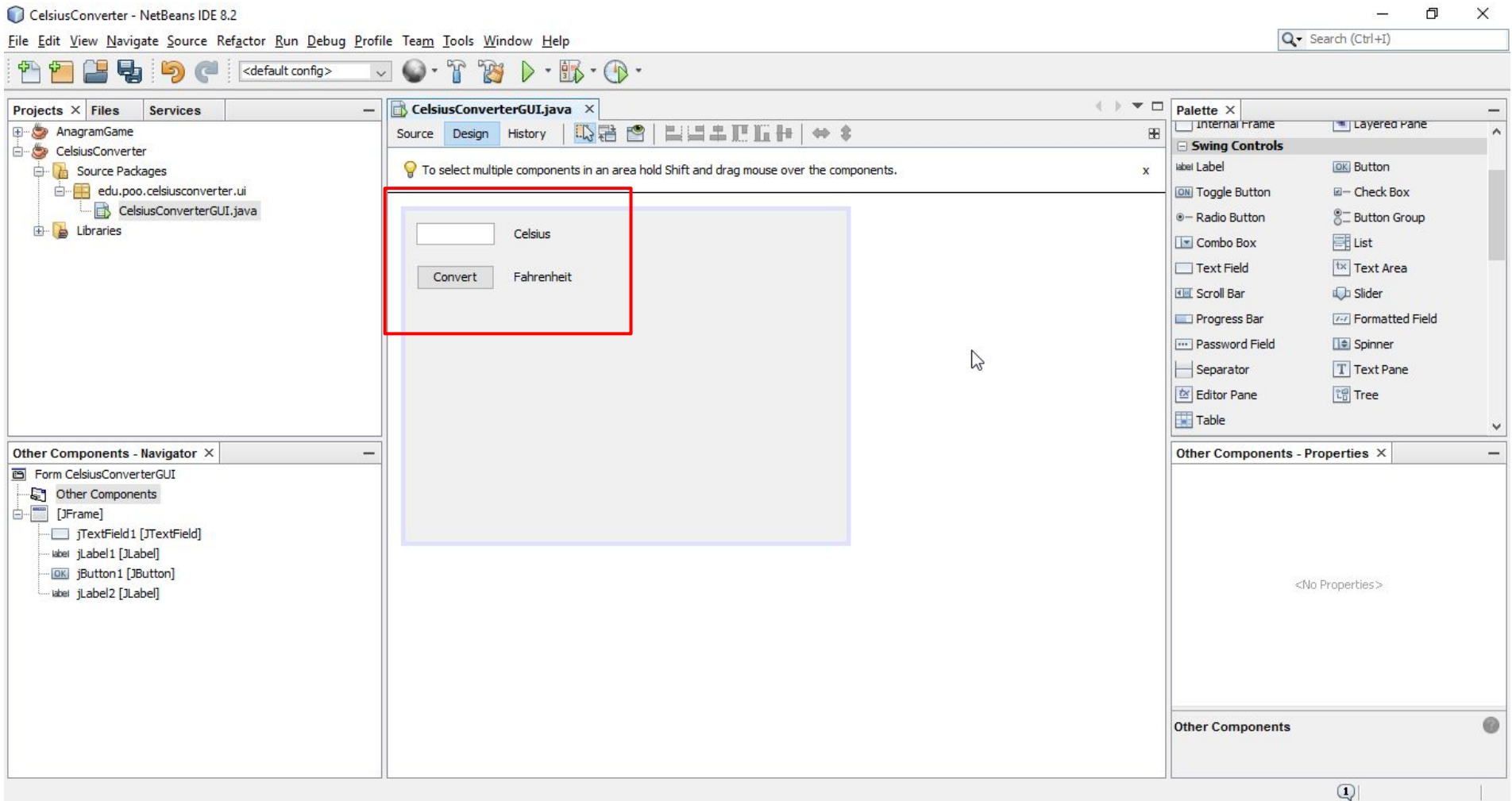
ActionEvent

ActionEvent

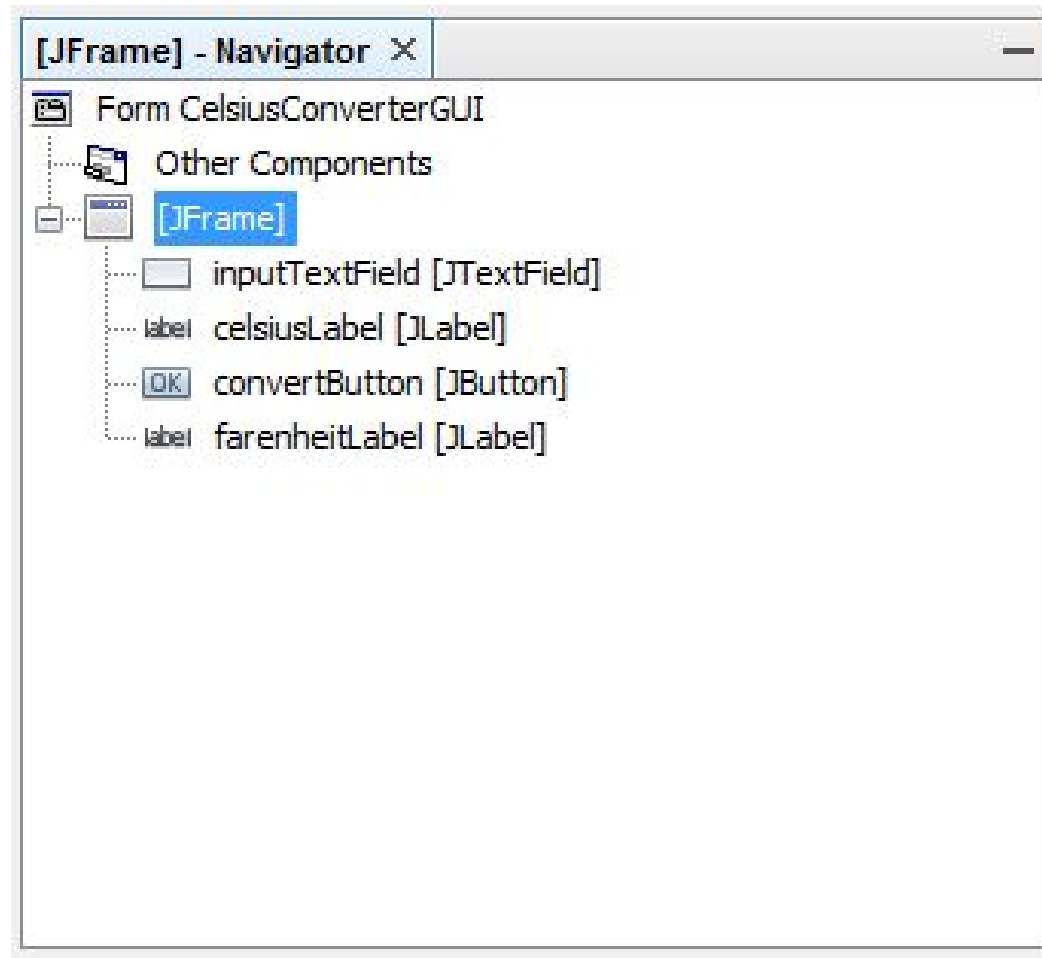
AdjustmentEvent

WindowEvent

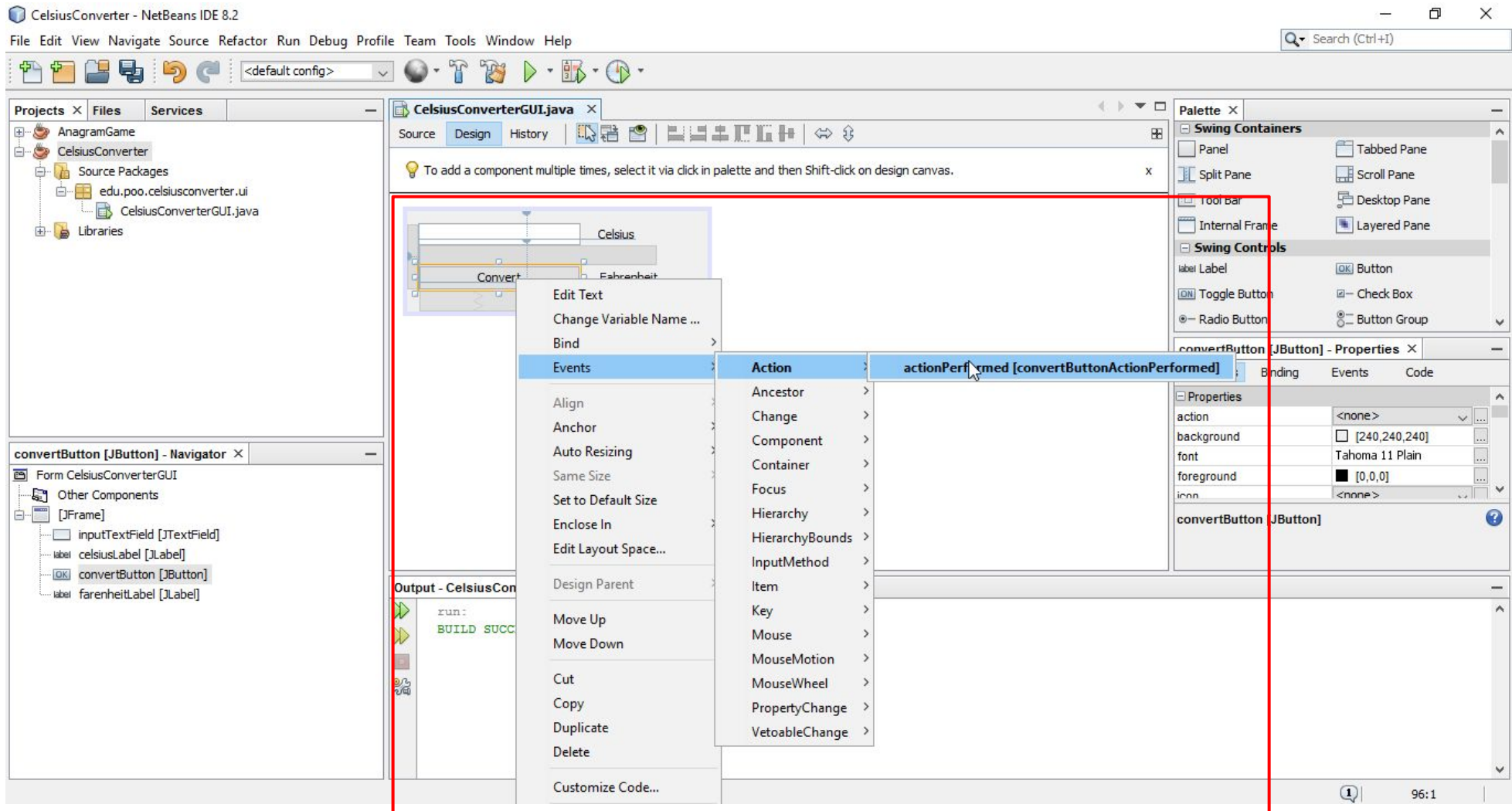
Netbeans - Nombres y etiquetas



Netbeans - Nombres de Variables



Netbeans - ActionListener



The screenshot shows the NetBeans IDE 8.2 interface with the 'CelsiusConverter' project open. The 'CelsiusConverterGUI.java' file is selected in the 'Files' tab. The 'Design' view is active, showing a visual representation of the GUI with a 'Celsius' label, a 'Fahrenheit' label, and a 'Convert' button. A right-click context menu is open over the 'Convert' button, with the 'Events' option selected. This opens a sub-menu where the 'Action' option is chosen. The 'ActionPerformed [convertButtonActionPerformed]' event is selected from the list. The 'Properties' window on the right shows the 'convertButton [JButton]' with its 'action' property set to '<none>'. The 'Other Components' window at the bottom left shows the 'convertButton [JButton]' selected. The 'Output' window at the bottom center shows the message 'BUILD SUCCESS'.

Netbeans - ActionListener (2)

CelsiusConverter - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Search (Ctrl+I)

Projects Files Services

- AnagramGame
- CelsiusConverter
 - Source Packages
 - edu.poo.celsiusconverter.ui
 - CelsiusConverterGUI.java
 - Test Packages
 - Libraries
 - Test Libraries

convertButtonActionPerformed - Navigator

Members

- CelsiusConverterGUI :: JFrame
 - CelsiusConverterGUI()
 - convertButtonActionPerformed(ActionEvent evt)
 - initComponents()
 - inputTextFieldActionPerformed(ActionEvent evt)
 - main(String[] args)
 - celsiusLabel : JLabel
 - convertButton : JButton
 - fahrenheitLabel : JLabel
 - inputTextField : JTextField

CelsiusConverterGUI.java

```
86     );
87
88     pack();
89     */ addition */
90
91     private void convertButtonActionPerformed(java.awt.event.ActionEvent evt) {
92         //Parse degrees Celsius as a double and convert to Fahrenheit.
93         int tempFahr = (int)((Double.parseDouble(inputTextField.getText())) * 1.8 + 32);
94         fahrenheitLabel.setText(tempFahr + " Fahrenheit");
95     }
96
97     private void inputTextFieldActionPerformed(java.awt.event.ActionEvent evt) {
98         // TODO add your handling code here:
99     }
100
101     /**
102      * @param args the command line arguments
103      */
104     public static void main(String args[]) {
```

Output - CelsiusConverter (run)

run:
BUILD SUCCESSFUL (total time: 10 seconds)

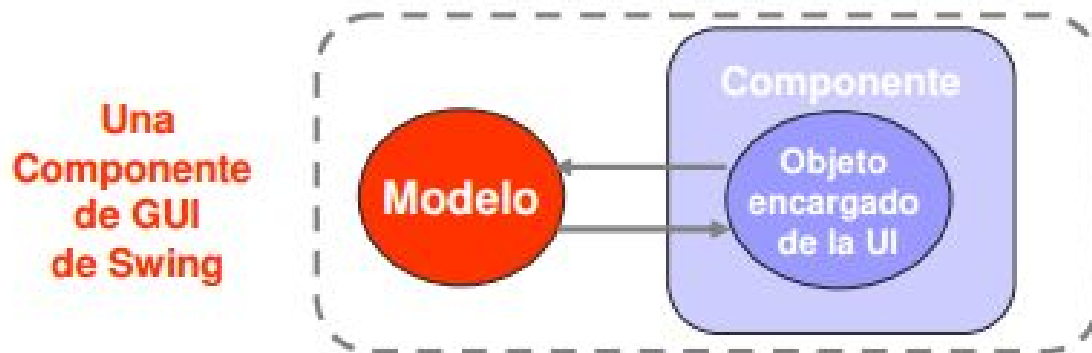
95:51/5:391 | INS

Netbeans - App finalizada



Swing - Componentes avanzadas

- La arquitectura de las componentes Swing responden a una **versión especializada del patrón MVC**. Está basada en 2 objetos: un objeto **Modelo** y un objeto encargado de *display* o apariencia y el manejo de los eventos, que representa la **Vista + el Controlador**.



El Modelo es tratado como un objeto separado

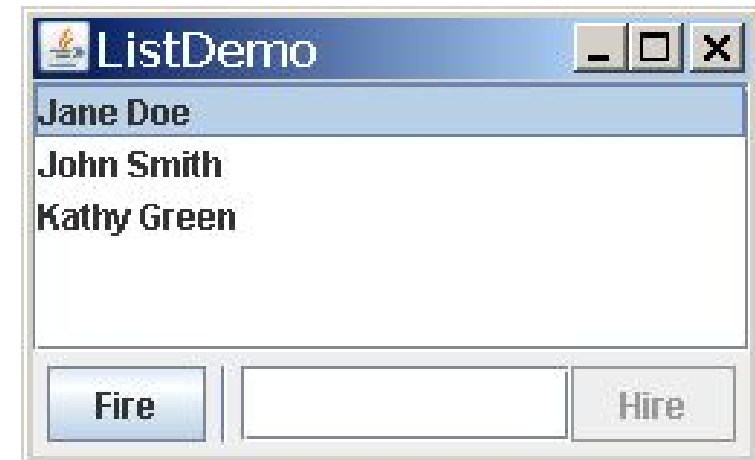
La Vista y Controlador están acoplados en un único objeto encargado de la UI

Swing - Componentes avanzadas (2)

- Cuando se crea un componente Swing, se crea automáticamente su objeto encargado de la UI.
- El objeto **Modelo** es responsable de almacenar el estado del componente. Si una aplicación, no provee un modelo explícito para un componente, Swing le crea un modelo por defecto.
- Swing provee un modelo por defecto para cada componente visual. Por ejemplo: **JTable -> DefaultTableModel, ...**
- El encargado de la UI se crea automáticamente y el modelo por defecto también. Entonces, **¿No hay que hacer nada?**
 - Si el componente es creado sólo para visualizar y permitir seleccionar datos, en general alcanza con el modelo por defecto, pero... si el componente es más complejo, y necesitan manipularse los datos (agregarse o eliminarse del modelo), se debe crear un modelo especial.

Swing - JList

- La clase JList, tiene varios constructores
- Para trabajar con listas, que sólo muestren opciones fijas, NO es necesario crear Modelos. El componente provee uno por defecto.



● `JList()` - `JList`

● `JList(ListModel arg0)` - `JList`

● `JList(Object[] arg0)` - `JList`

● `JList(Vector<?> arg0)` - `JList`

Si inicializamos la lista con estos constructores, podemos agregar y borrar elementos.

Si inicializamos la lista con un arreglo o un vector, el constructor crea un modelo por defecto, que es **IMUTABLE!!!**

Swing - JList (2)

Creación de una lista de selección con un modelo por defecto.



```
public class ListaPlanetas extends JFrame {
    private JList lista;
    private JButton imprimir = new JButton("imprimir selección");

    public void init() {
        String items[] = {"Mercurio", "Tierra", "Saturno", "Júpiter"};
        lista = new JList(items);
        lista.setVisibleRowCount(4);
        Container cp = this.getContentPane();

        imprimir.addActionListener(new MiActionListener());

        cp.setLayout(new FlowLayout());
        cp.add(new JScrollPane(lista));
        cp.add(imprimir);
    }

    class MiActionListener implements ActionListener {
        public void actionPerformed(ActionEvent e) {
            System.out.println(lista.getSelectedValue());
        }
    }

    public static void main(String[] args) {

```

Se crea el arreglo con los valores

El constructor crea un modelo por defecto con los datos del arreglo

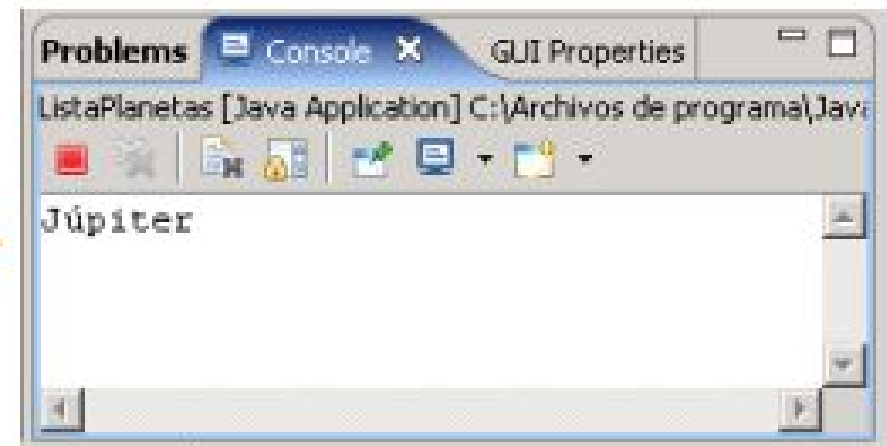
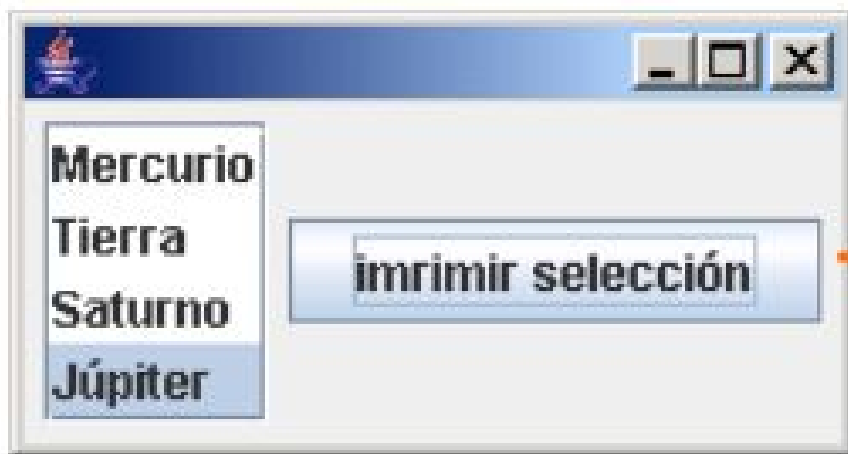
Las listas (JList) no poseen barras de desplazamiento, por lo tanto se las debe colocar adentro de un JScrollPane, quien la hace desplazarse.

Si ejecutamos una main() como este ...

```
public static void main(String[] args) {
    ListaPlanetas app = new ListaPlanetas();
    app.init();
    app.pack();
    app.setVisible(true);
}
```

Swing - JList (3)

- ¿Cómo funciona la aplicación?
- Sólo se pueden visualizar datos fijos, y seleccionar items de la lista. No se pueden eliminar, reemplazar, ni agregar valores al Modelo por defecto creado automáticamente



Swing - JList Modelo actualizable

- ¿Cómo creo una lista de selección con un modelo actualizable?
- ¿Puedo tener un campo de entrada y agregar el valor ingresado a la lista?, ¿y eliminar algún valor de la lista?
- Hay 2 alternativas:
 - Se crea usando el constructor por defecto y luego se le setea un **Modelo**:

```
JList lista = new JList();  
lista.setModel(modelo);
```
 - Se crea usando el constructor que tiene un **Modelo** como argumento:

```
JList lista = new JList(modelo);
```

Swing - JList Modelo actualizable (2)

```
*Lista.java x ListaPlanetas.java
import java.awt.Container;

public class Lista extends JFrame{
    private JList lista= new JList();
    private DefaultListModel modelo = new DefaultListModel();
    private JButton agregar = new JButton("Agregar");
    private JButton quitar = new JButton("Eliminar");
    private JTextField editor = new JTextField("");

    public void init() {
        String items[] = {"Mercurio", "Tierra", "Saturno", "Júpiter"};
        for (int i = 0; i < items.length; ++i)
            modelo.addElement(items[i]);
        lista.setVisibleRowCount(4);
        lista.setModel(modelo);

        Container cp = this.getContentPane();
        agregar.addActionListener(new MiActionListenerAgrega());
        quitar.addActionListener(new MiActionListenerBorra());
        cp.setLayout(new FlowLayout());
        cp.add(new JScrollPane(lista));
        editor.setColumns(15);
        cp.add(editor);
        cp.add(agregar);
        cp.add(quitar);
    }

    class MiActionListenerAgrega implements ActionListener{
    class MiActionListenerBorra implements ActionListener{
    public static void main(String[] args) {
```

Se usa el constructor sin argumentos!!

Se instancia un objeto DefaultListModel (**Modelo**)

Se crea el arreglo

Se itera el arreglo y se guardan los valores en el **Modelo**

Se liga el **Modelo** a la componente **JList** !!

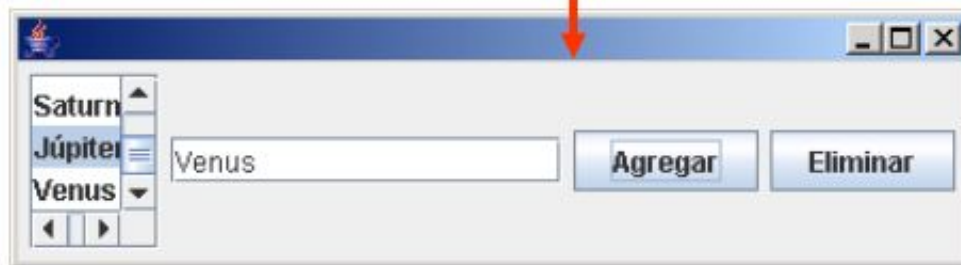
Registramos listeners en los botones

Si ejecutamos una main() como este ...

```
    public static void main(String[] args) {
        ListaPlanetas app = new ListaPlanetas();
        app.init();
        app.pack();
        app.setVisible(true);
    }
```

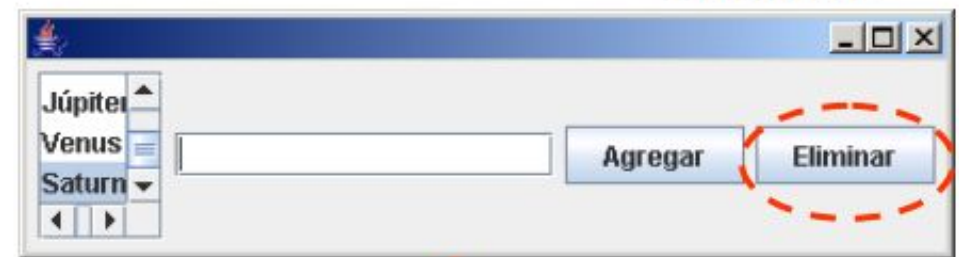

Swing - JList Modelo actualizable (3)

¿Cómo funciona la aplicación?



Al insertar en el Modelo de la componentes Swing, se actualiza la Vista. Además como no pueden visualizarse todos los valores, aparecen las barras de desplazamiento.

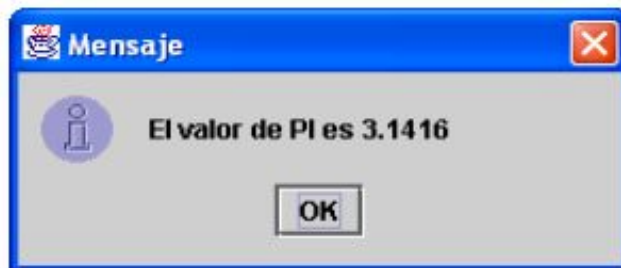
Elimina el item
seleccionado



Al eliminar del Modelo el item seleccionado de la JList, se actualiza la Vista. Además como pueden visualizarse todos los valores, desaparecen las barras de desplazamiento.

Swing - Ventanas de Diálogo

- Swing también provee una clase que automatiza muchas de las actividades que un programador haría para crear ventanas de diálogo: [JOptionPane](#). Esta clase tiene muchos métodos de clase que permiten crear diálogos con íconos, mensajes, campos de entrada y botones.



```
JOptionPane.showMessageDialog(  
    null,                          //padre  
    "El valor de PI es "+pi,        //mensaje  
    "Mensaje",                     // título  
    JOptionPane.INFORMATION_MESSAGE); //hay 5 tipos de mensajes  
}
```



```
JOptionPane.showConfirmDialog(  
    boton,                          //padre  
    "¿Salva antes de salir?",        //mensaje  
    "Mensaje",                     //título  
    JOptionPane.YES_NO_CANCEL_OPTION, //tipo de opción  
    JOptionPane.WARNING_MESSAGE,     // tipo de mensaje  
    new ImageIcon("duke.gif"));      //ícono
```

Todos los diálogos creados con [JOptionPane](#), son modales.