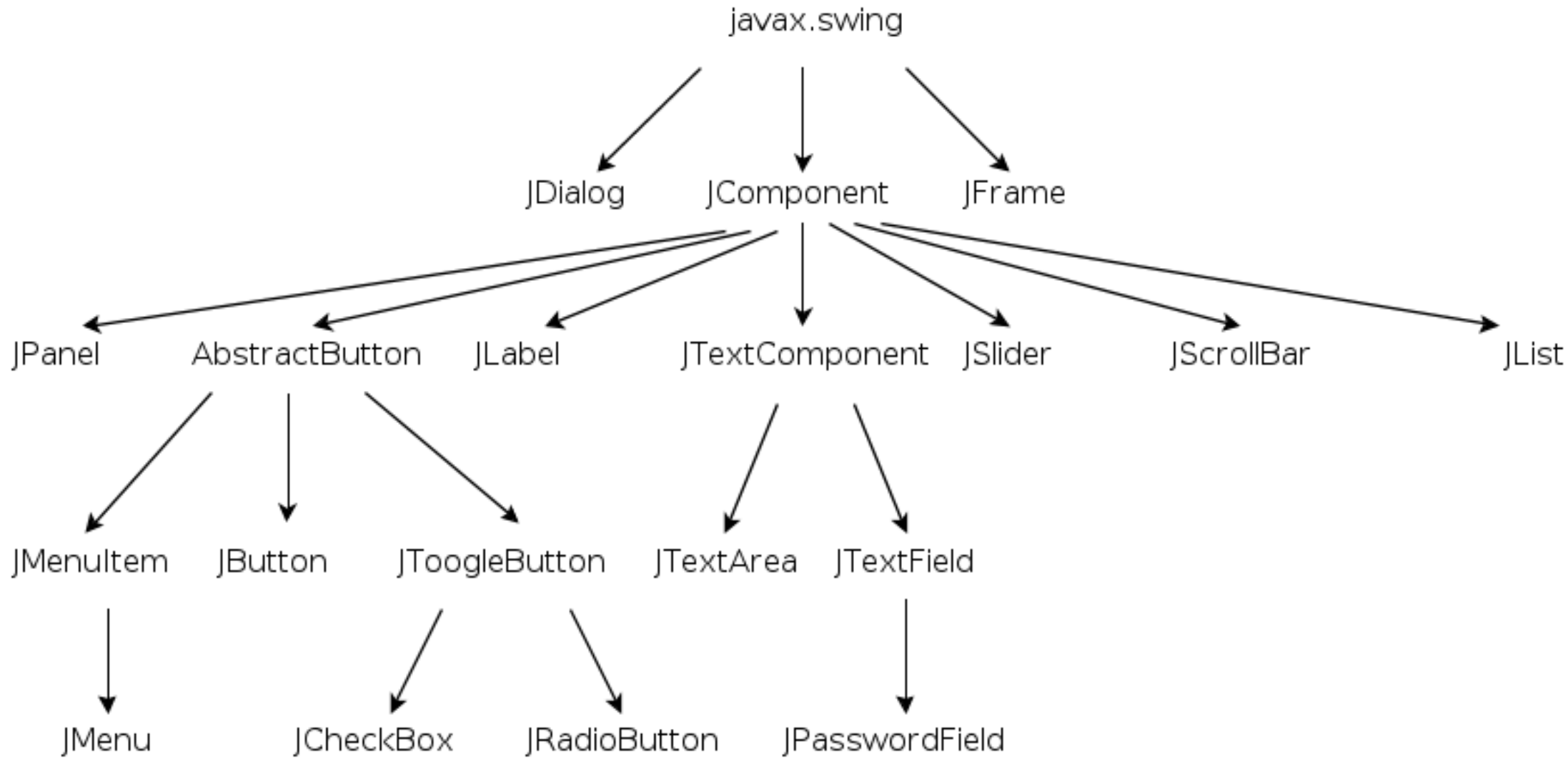
ILERNA Online

CFGS: Desarrollo de Aplicaciones Multiplataforma

Módulo 07: Desarrollo de Interfaces

Modifier	Class	Package	Subclass	World
`public`	✔	✔	✔	✔
`protected`	✔	✔	✔	✘
*`no modifier`**	✔	✔	✘	✘
`private`	✔	✘	✘	✘


javax.swing
JDialog
JComponent
JFrame
JPanel
AbstractButton
JLabel
JTextComponent
JSlider
JScrollBar
JList
JMenuItem
JButton
JToogleButton
JTextArea
JTextField
JMenu
JCheckBox
JRadioButton
JPasswordField

QUÉ VAMOS A VER

UF1. DISEÑO E IMPLEMENTACIÓN DE INTERFACES

1. Confección de interfaces de usuario

1.1. Librerías de componentes disponibles para los distintos sistemas operativos y lenguajes de programación. Características

1.2. Enlaces de componentes a orígenes de datos

1.3. Componentes: características y campos de aplicación

1.4. Eventos, escuchadores y acciones a eventos

1.5. Edición del código generado por las herramientas de diseño

1.6. Clases, propiedades, métodos

2. Generación de interfaces a partir de documentos XML

2.1. Lenguajes de descripción de interfaces basadas en XML. Ámbito de aplicación

2.2. XAML (Extensible Application Markup Language)

2.3. XUL (Extensible User Interface Language)

2.4. SVG (Scalable Vectorial Graphics)

2.5. UIML (User Interface Markup Language)

2.6. MXML (Macromedia eXtensible Markup Language)

2.7. Generación de código para diferentes plataformas

2.8. Herramientas libres y propietarias para la creación de interfaces de usuario multiplataforma. Herramientas para crear interfaces, editor XML

2.9. Edición del documento XML

QUÉ VAMOS A VER

UF1. DISEÑO E IMPLEMENTACIÓN DE INTERFACES

3. Creación de componentes visuales

3.1. Concepto de componente y características

3.2. Propiedades y atributos

3.3. Elementos generales

3.4. Eventos; asociación de acciones a eventos

3.5. Persistencia del componente

3.6. Herramientas para desarrollo de componentes visuales

4. Usabilidad

4.1. Concepto de usabilidad

4.2. Medidas de usabilidad

4.3. Pautas de diseño de interfaces

4.4. W3C

5. Confección de informes

5.1. Informes incrustados y no incrustados en la aplicación

5.2. Creación de parámetros

5.3. Creación de subinformes

5.4. Imágenes

5.5. Informes incrustados y no incrustados en la aplicación

5.6. Herramientas gráficas integradas y externas al IDE

5.7. Filtrado de datos

5.8. Numeración de líneas, recuentos y totales

5.9. Librerías para la generación de informes. Clases, métodos y atributos

ILERNA Online

QUÉ VAMOS A VER

UF2. PREPARACIÓN Y DISTRIBUCIÓN DE APLICACIONES

6. Realización de pruebas

6.1. Objetivo, importancia y limitaciones del proceso de prueba. Estrategias

6.2. Prueba unitaria

6.3. Pruebas de integración: ascendentes y descendentes

6.4. Pruebas del sistema: configuración, recuperación, entre otras

6.5. Pruebas de uso de recursos

6.6. Pruebas funcionales

6.7. Pruebas de simulaciones

6.8. Pruebas de aceptación

6.9. Pruebas alfa y beta

6.10. Pruebas manuales y automáticas

6.11. Herramientas de software para la realización de pruebas

7. Documentación de aplicaciones

7.1. Archivos de ayuda. Formatos

7.2. Herramientas de generación de ayudas

7.3. Tablas de contenidos, índices, sistemas de búsquedas, entre otros

7.4. Tipos de manuales

8. Distribución de aplicaciones

8.1. Componentes de una aplicación. Empaquetado

8.2. Instaladores

8.3. Paquetes autoinstalables

8.4. Herramientas para crear paquetes de instalación

8.5 Personalización de la instalación: logotipos, fondos, diálogos, botones, idioma, entre otros

ILERNA Online

UF1: DISEÑO E IMPLEMENTACIÓN DE INTERFACES

ILERNA Online

Tema 1. Confección de interfaces de usuario.

1. Confección de interfaces de usuario

IDE (Entorno de Desarrollo Integrado).

- Principales características:
  - Codificación.
  - Compilación.
  - Depuración.
  - Testeo.

Visual Studio

- Dispone de una versión gratuita → *express.* Esta versión está más limitada que las de pago (ya que es grauita).
- Permite programar en varios lenguajes de programación (C, C++, C#, etc).

NetBeans

- Desarrollado en Java.
- Cuenta con la posibilidad de programar también en lenguajes como: XML, HTML, PHP, JavaScript, JavaDoc, Groovy y JSP.

Eclipse

- El homologo a NetBeans.
- Sigue el principio WYSWYG (Lo que ves es lo que obtienes).
- Permite plugins para programar en otros lenguajes, al igual que NetBeans.

Visual Studio es una IDE creada por Microsoft

NetBeans gratuita y puede instalarse en cualquier sistema operativo, pero al tratarse de java necesita su máquina virtual

Eclipse, del estilo de NetBeans.

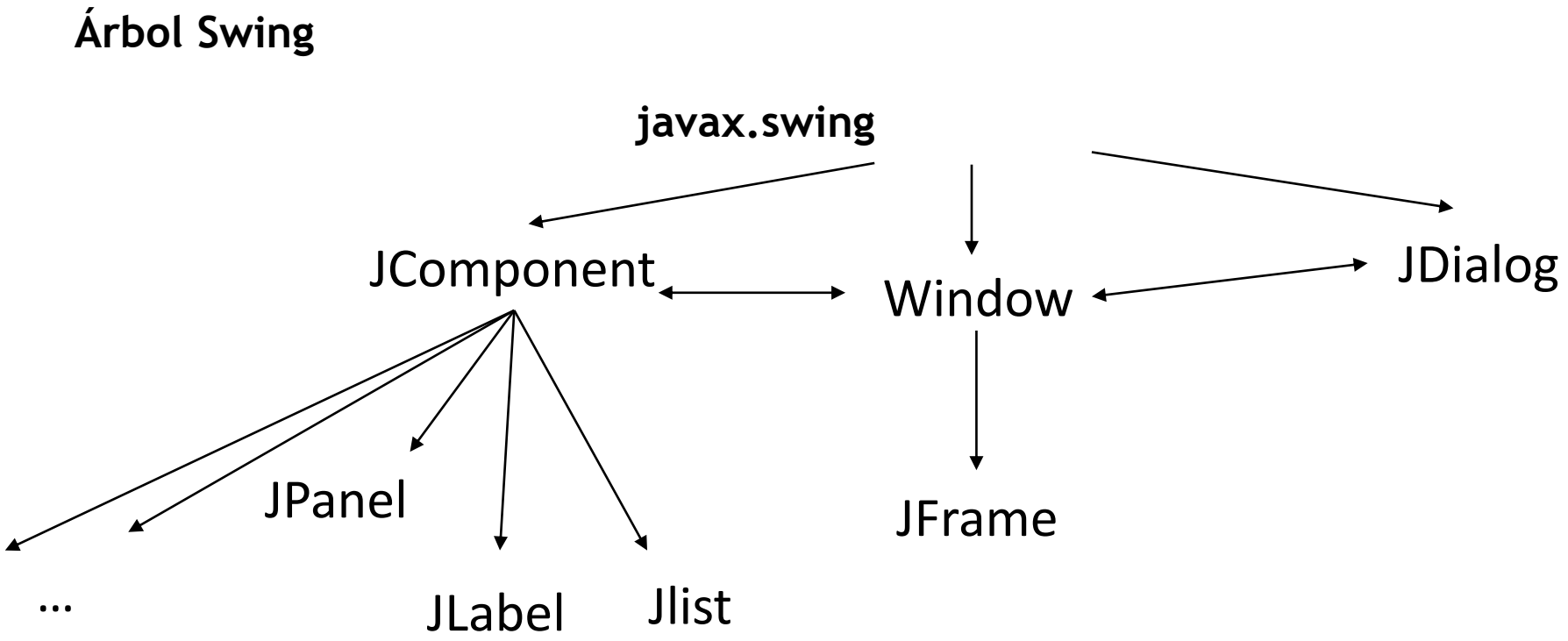
ILERNA Online

1.1. Librerías de componentes disponibles para los distintos sistemas operativos y lenguajes de programación. Características (1)

- **AWT (Abstract Window Toolkit)**
  - Diseñada en Java puro y utilizada como base de la librería Swing.
  - Presenta un buen funcionamiento a pesar de no contar con controles demasiado avanzados.
  - Utilizada para el desarrollo de diseños prácticos y eficientes
  - Su aspecto dependerá del sistema operativo que utilice.
  - Con el paso del tiempo ha perdido protagonismo, por lo que hoy en día se considera obsoleta.
- **Swing**
  - Diseñada en Java puro y creada a partir de la librería AWT.
  - Su función es dar respuesta a todos los inconvenientes que presente AWT.
  - Cuenta con controles de bastante funcionalidad utilizados para aplicaciones.
  - En versiones posteriores, Java 5 y Java 6, se convirtió en un framework de desarrollo de interfaces para nuevas aplicaciones. Cuenta con controles de gran funcionalidad.
  - Presenta carencias como el filtrado y organización de datos en controles como tipo tabla y árbol, con la necesidad de realizar un desarrollo de forma manual en esta funcionalidad.
- **SWT (Standard Widget Toolkit)**
  - Creada por IBM para el entorno de desarrollo Eclipse, mejorando la versión Swing que existía en ese mismo momento.
  - Considerada la tercera generación de librerías (después de AWT y Swing), aunque no está muy relacionada con ellas.
  - Considerada una librería de bajo nivel que utiliza widgets nativos de la misma plataforma que ejecuta mediante JNI (Java Native Interface).
  - Estas interfaces no se pueden ejecutar en todas las plataformas.
  - Uno de sus inconvenientes es que la API que proporciona la librería es bastante complicada de utilizar y no muy intuitiva.
- **JavaFX**
  - Desarrollada por Java/Oracle, siendo una plataforma open source.
  - - Basada principalmente en el desarrollo de aplicaciones **RIA (rich internet application)**, permite ser utilizada en diferentes plataformas y dispositivos.
  - - En esta librería se agrupan las tecnologías conocidas como JavaFX Mobile y JavaFX Script.

Las aplicaciones **RIA** buscan tener la misma funcionalidad y aspecto que las aplicaciones tradicionales de escritorio, buscando una mejor interacción con el usuario.

Árbol Swing

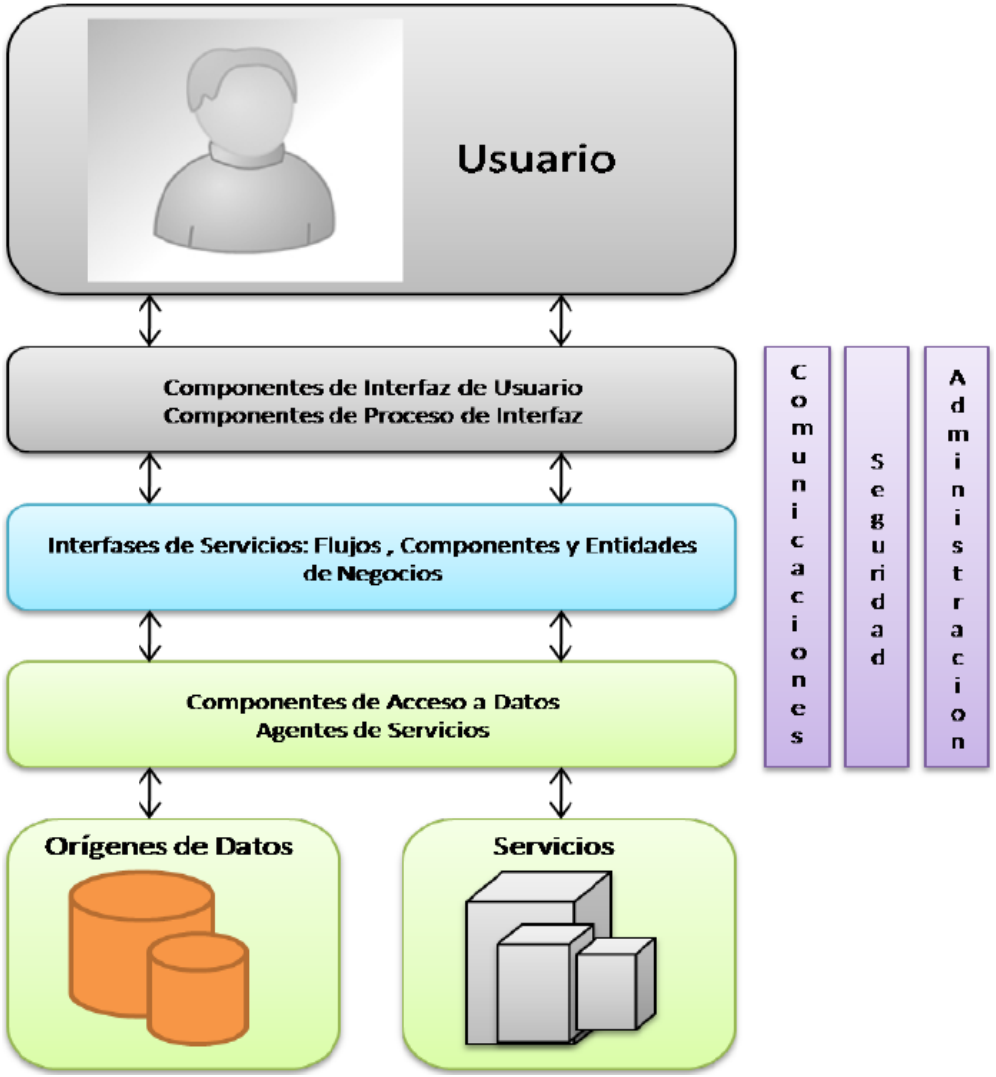


Las clases que empiezan por J son de swing

1.1. Librerías de componentes disponibles para los distintos sistemas operativos y lenguajes de programación. Características (2)

- SwingX
- Apache Pivot
- Qt Jambi
- Librerías OpenGL
- API DirectX
  - Direct2D
  - Direct3D
  - DirectCompute
- Qt
- GTK+

1.2. Enlaces de componentes a orígenes de datos



1.3. Componentes: características y campos de aplicación

Vemos la **Programación orientada a componentes (POC).**

- Hace énfasis en la **descomposición de sistemas ya conformados en componentes funcionales** con interfaces bien definidas usadas para la comunicación entre componentes.
- Los **componentes** tienen un **nivel de abstracción más elevado que los objetos**. Por eso **no comparten estado y se comunican a través de mensajes que contienen datos.**
- Podemos definir un **componente de *software*** como un elemento del sistema que **ofrece servicios predefinidos y puede comunicarse con otros componentes**.

> Un **componente** es un objeto escrito de acuerdo con unas especificaciones, las cuales hacen que el objeto se convierta en componente adquiriendo características como, por ejemplo, la reusabilidad.

- **Objetivo de la POC**: construir una serie de componentes software, permitiendo a los desarrolladores **reutilizar componentes** ya diseñados y testados **para desarrollar** así las **aplicaciones de** una **forma** más **rápida y robusta**.
- Entidad básica de la POC: los componentes.
  - Podemos verlos como cajas negras que encapsulan cierta funcionalidad.
  - Son diseñados sin saber cuando se van a usar.
  - Los servicios de los componentes son conocidos por sus interfaces y requisitos.
- Diseñar un componente reutilizable conlleva gran esfuerzo y atención:
  - Debe estar bien documentado.
  - Está diseñado pensando en su uso de maneras imprevistas.
  - Debe estar probado en profundidad:
    - Debe ser robusto, comprobando la validez de las entradas.
    - Debe ser capaz de enviar mensajes de error apropiados.

1.4. Eventos, escuchadores y acciones a eventos

Un **evento** es una acción que resulta de la interacción de un usuario sobre un componente de la aplicación.

- Es necesario definir cada **evento** con la **acción** a la que va asociada. Esta acción es conocida como administrador de eventos.
- La idea es: cuando se ejecuta el programa, se realizan inicializaciones y después el programa esperará a que ocurra cualquier **evento**.
- Cuando se **dispara un evento** el programa **ejecuta** el código correspondiente del **administrador de eventos** (es decir, esa acción que hemos programado que haga). **Ejemplo**: un botón que cuando el usuario lo pulsa se reproduce un sonido.
- La **programación dirigida a eventos** es la base de la interfaz de usuario (ya que el usuario interactúa con ésta constantemente).
- Un escuchador de eventos (**event listener**) es un mecanismo asíncrono ante ciertas circunstancias que ocurren en clases diferentes a la nuestra. Se utiliza, por ejemplo, para detectar si un botón ha sido pulsado.
- Para usar un escuchador de eventos, se tienen que seguir tres pasos:
  - Implementar la interfaz del escuchador.
  - Registrar el escuchador en el objeto que genera el evento, indicándole el objeto que los recogerá.
  - Implementar los métodos callback correspondientes.

- Un ejemplo de programa orientado a objetos con eventos en pseudocódigo:

CÓDIGO:

```
While (true) {
    Switch (event) {
            Case mouse_click ():
            Case Keypressed ():
            Case Else:
    }
}
```

ILERNA Online

Eventos y Listeners (1)

- Los eventos son interacciones del usuario con la interfaz.
- Cada componente esta asociado a un evento
- El desarrollador implementará la acción cuando ocurra ese evento
- Los objetos(implementación) son definiciones de cómo tratar esos eventos
    - Se les notifica ese evento y se relacionan con la acción → Listeners
- Dos tipos eventos principalmente:
    - ActionEvent: Por ejemplo, eventos que se desencadenan con el ratón. (paquete → java.awt.event)
    - WindowEvent: Por ejemplo, eventos que se desencadenan con la ventana. (paquete →java.awt.event)

Eventos y Listeners (2)

java.awt.event

Interface KeyListener

All Superinterfaces:

EventListener

All Known Implementing Classes:

AWTEventMulticaster, BasicComboBoxUI.KeyHandler, BasicComboPopup.InvocationKeyHandler, BasicTableUI.KeyHandler, BasicTreeUI.KeyHandler, KeyAdapter

- **keyTyped** → Realiza una acción al pulsar y soltar la tecla.
- **keyPressed** → Realiza una acción al pulsar la tecla.
- **keyReleased** → Realiza una acción al soltar la tecla.

Method Summary

Methods

Modifier and Type	Method and Description
void	keyPressed(KeyEvent e) Invoked when a key has been pressed.
void	keyReleased(KeyEvent e) Invoked when a key has been released.
void	keyTyped(KeyEvent e) Invoked when a key has been typed.

ILERNA Online

Eventos y Listeners (3)

java.awt.event

Interface MouseListener

All Superinterfaces:

EventListener

All Known Subinterfaces:

MouseInputListener

All Known Implementing Classes:

AWTEventMulticaster, BasicButtonListener, BasicComboPopup.InvocationMouseHandler, BasicComboPopup.ListMouseHandler, BasicDesktopIconUI.MouseInputHandler, BasicFileChooserUI.DoubleC
BasicMenuItemUI.MouseInputHandler, BasicMenuUI.MouseInputHandler, BasicScrollBarUI.ArrowButtonListener, BasicScrollBarUI.TrackListener, BasicSliderUI.TrackListener, BasicSplitPaneDivider.M
BasicTextUI.BasicCaret, BasicToolBarUI.DockingListener, BasicTreeUI.MouseHandler, BasicTreeUI.MouseInputHandler, DefaultCaret, FormView.MouseEventListener, HTMLEditorKit.LinkController, M
MouseInputAdapter, ToolTipManager

Method Summary

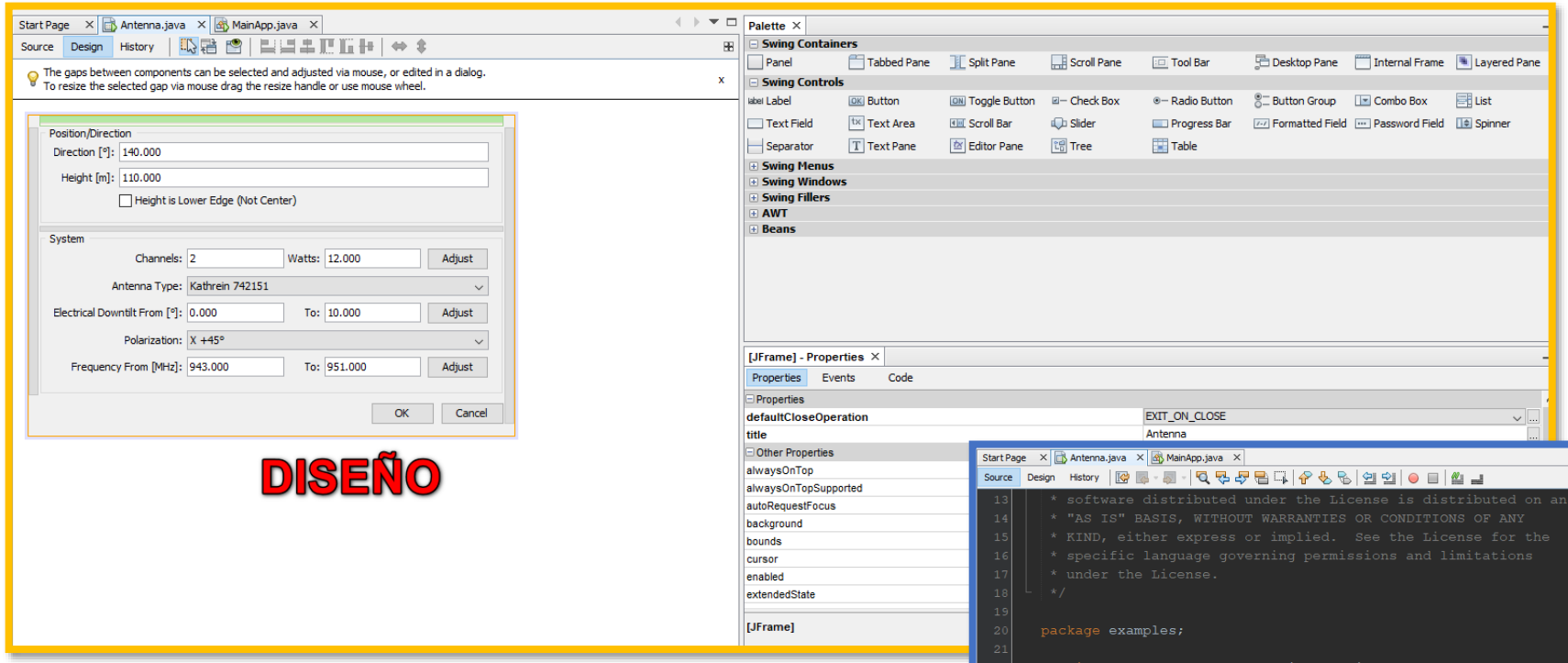
Methods

Modifier and Type	Method and Description
void	`mouseClicked(MouseEvent e)` Invoked when the mouse button has been clicked (pressed and released) on a component.
void	`mouseEntered(MouseEvent e)` Invoked when the mouse enters a component.
void	`mouseExited(MouseEvent e)` Invoked when the mouse exits a component.
void	`mousePressed(MouseEvent e)` Invoked when a mouse button has been pressed on a component.
void	`mouseReleased(MouseEvent e)` Invoked when a mouse button has been released on a component.

- Esta interfaz se encargará exclusivamente de los eventos de ratón, por ejemplo, si queremos realizar una acción apretando únicamente el botón con el ratón.

ILERNA Online

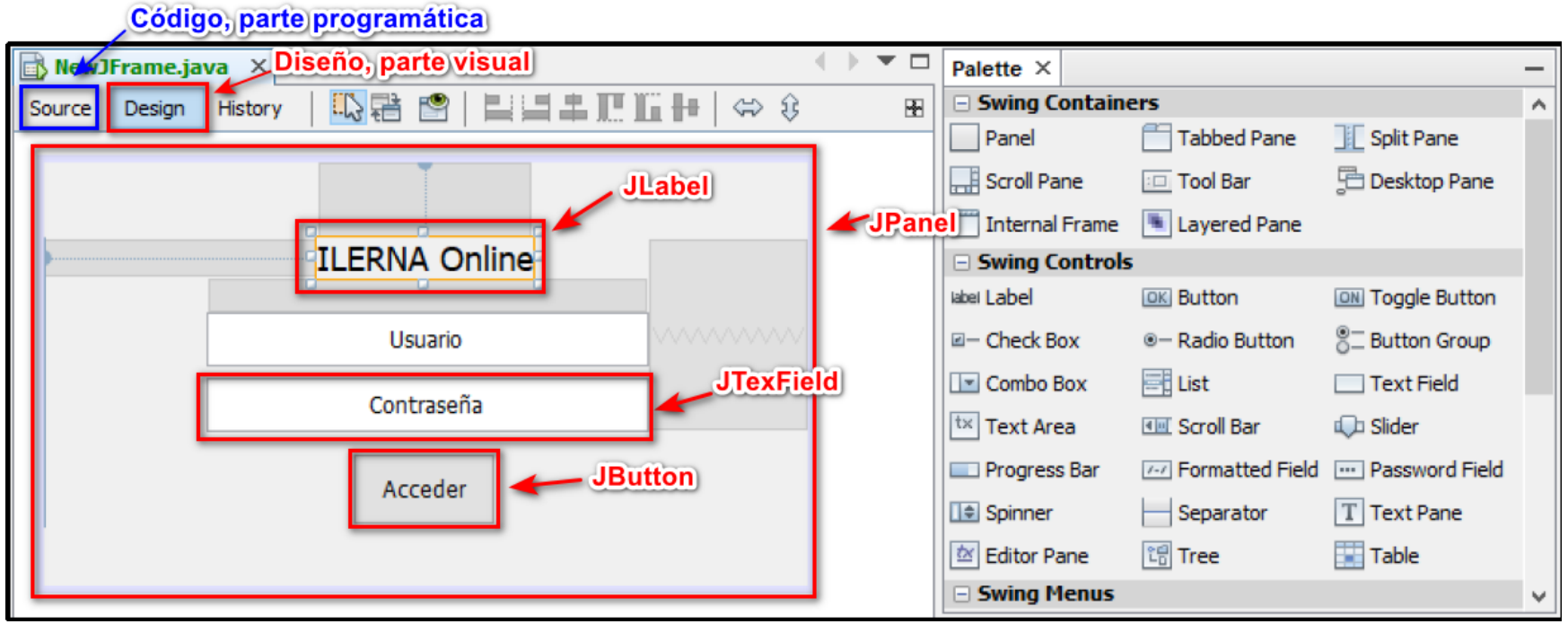
1.5. Edición del código generado por las herramientas de diseño (1)



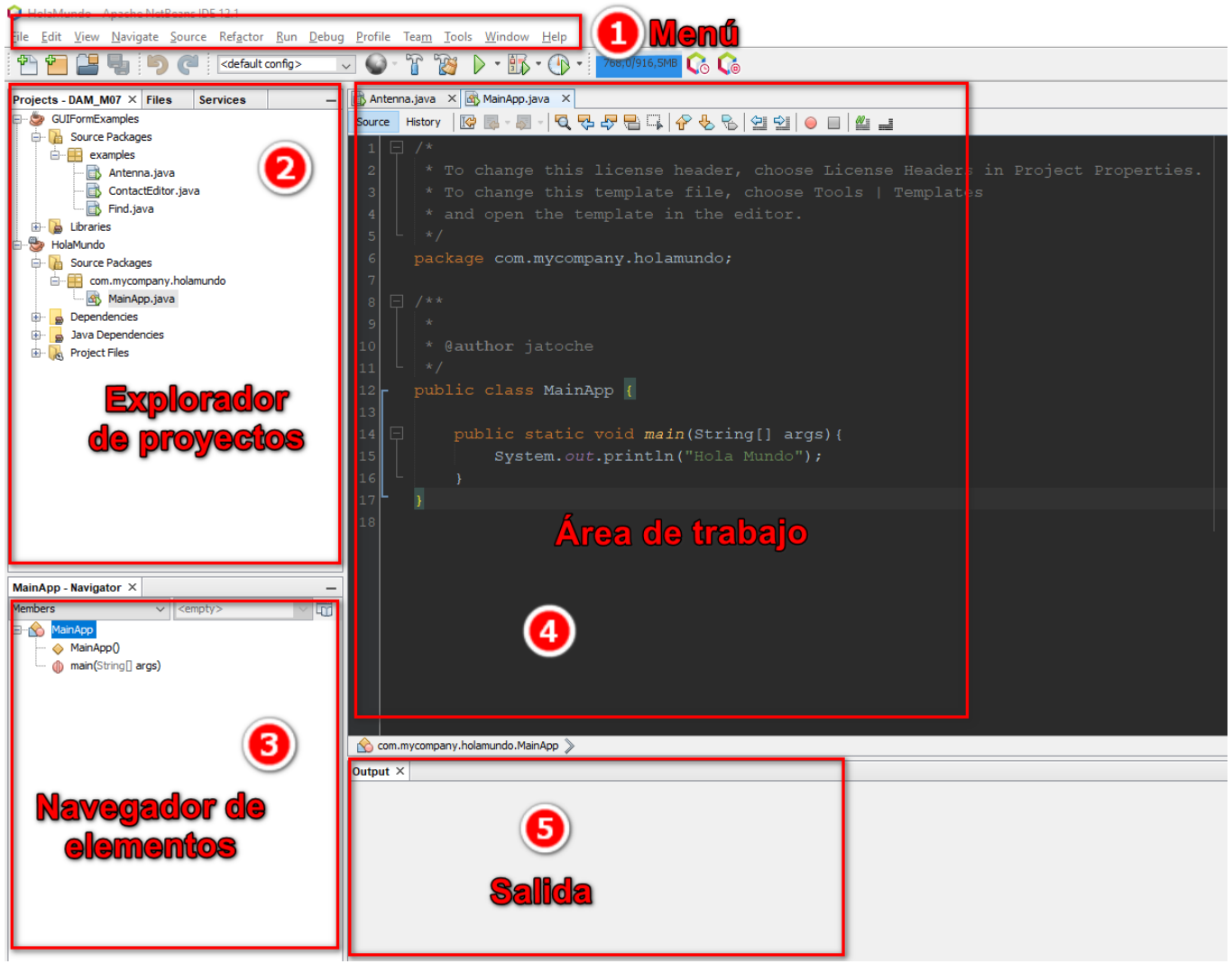
NetBeans

ILERNA Online

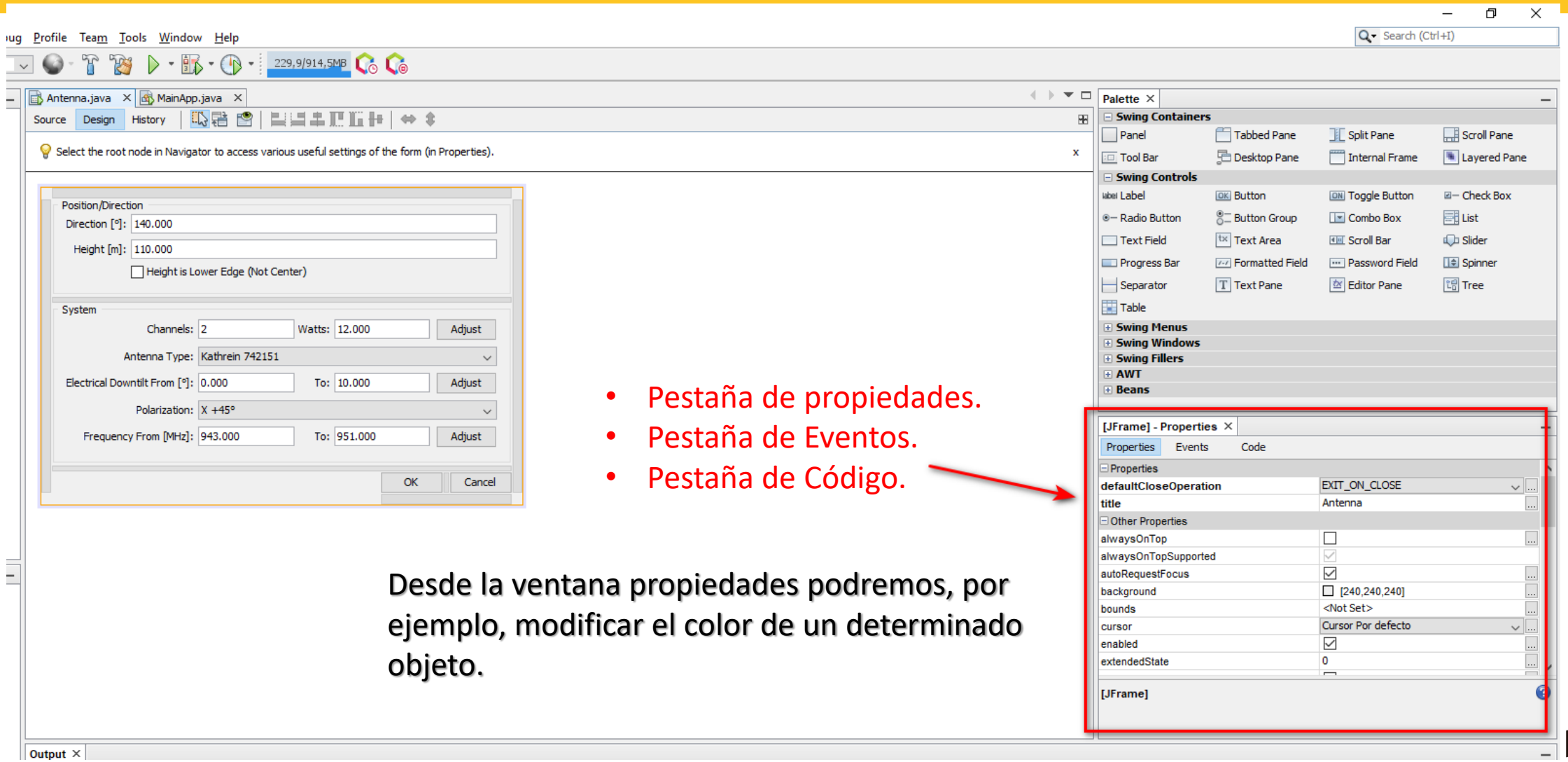
1.5. Edición del código generado por las herramientas de diseño (2)



Ventana de Entorno NetBeans



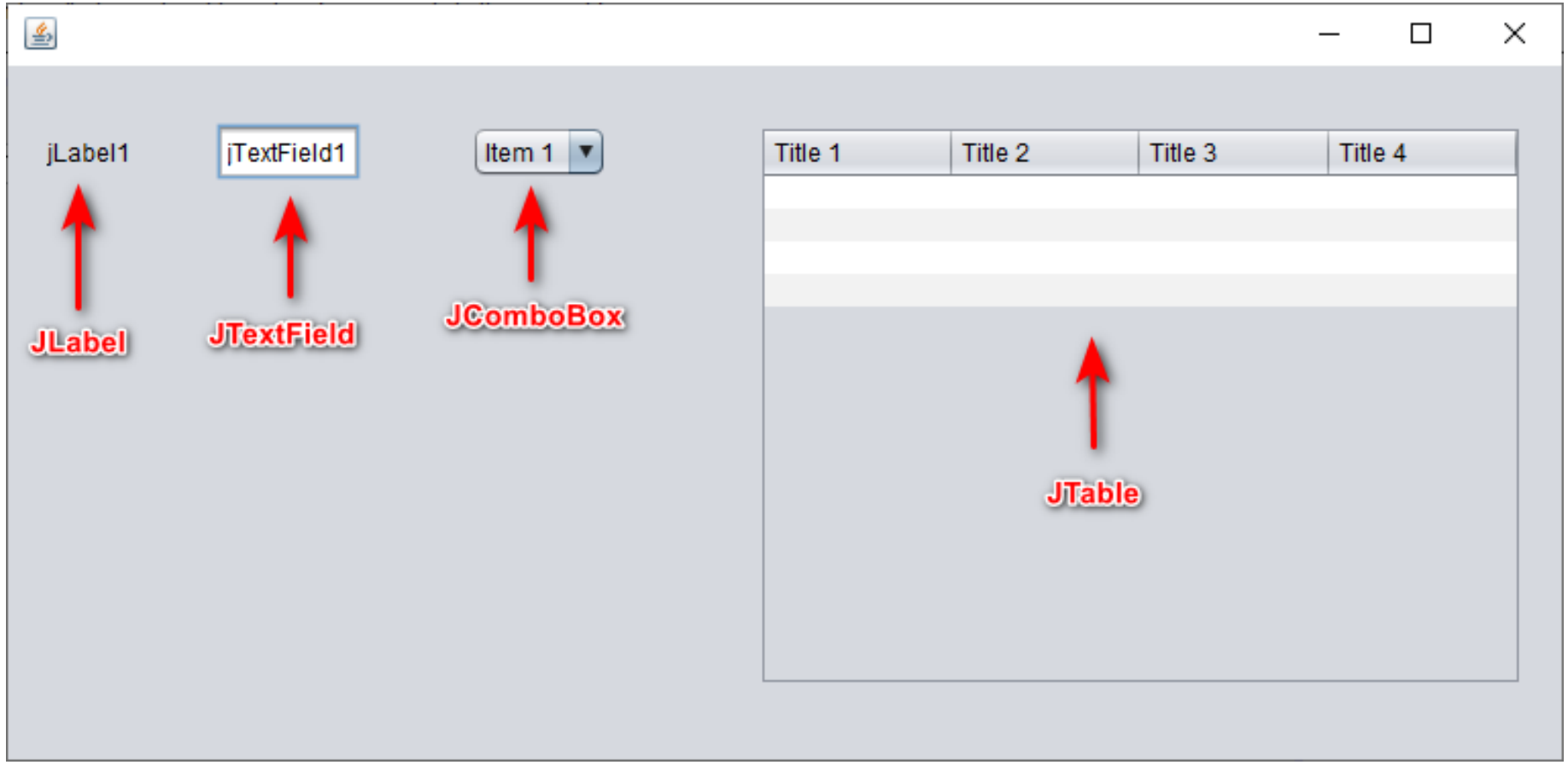
Ventana NetBeans. Design



- Pestaña de propiedades.
- Pestaña de Eventos.
- Pestaña de Código.

Desde la ventana propiedades podremos, por ejemplo, modificar el color de un determinado objeto.

Componentes Swing. Algunos Ejemplos



Componentes Swing. Algunos Ejemplos

VAMOS A NETBEANS

Componentes Swing. CÓDIGO: MainApp

```
package interfaz.grafica;

import javax.swing.JFrame;


public class MainApp {

    public static void main(String[] args){
        // muestra por consola el texto indicado
        System.out.println("Hola Mundo");

        // crea un nuevo marco
        MiMarco marco1 = new MiMarco();

        // cierra la ventana y deja de ejecutarse el programa
        marco1.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // necesitamos hacer visible el marco
        marco1.setVisible(true);


    }
}
```

Componentes Swing. CÓDIGO: MiMarco

```
package interfaz.grafica;

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

public class MiMarco extends JFrame {


    public MiMarco(){

        // modifica el titulo del marco
        setTitle("MiMarco");

        // Modificamos un JFrame a tamaño 500x300 configura
el tamaño de la ventana
        setSize(500,300);

        // mueve la ventana a la localizacion dada
        //SetLocation(500,300);

        // configura el tamanyo de la ventana y la
localizacion de la misma
        setBounds(500, 200, 400, 400);

        // extiende el marco a la opcion indicada
        //setExtendedState(Frame.MAXIMIZED_BOTH);


        // crea una lamina para el marco
        MiLamina lamina1 = new MiLamina();

        // anyade la lamina al marco
        add(lamina1);
    }

}
```

Componentes Swing. CÓDIGO: MiLamina

```
package interfaz.grafica;

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

public class MiLamina extends JPanel implements ActionListener{

    // creamos un boton
    JButton boton1 = new JButton("Nombre del Botón");

    public MiLamina() {

        // anyade el boton a la lamina
        add(boton1);

        // anyade un escuchador (listener) al boton1
        /*Con esto conseguimos que cuando el boton sea pulsado, la lamina estara a la escucha para recoger ese evento*/
        boton1.addActionListener(this);
    }

    // este metodo define la accion que se llevara a cabo cuando el evento se produzca
    @Override
    public void actionPerformed(ActionEvent e) {
        setBackground(Color.red);
    }
}
```

1.6. Clases, propiedades, métodos (1)

Las **clases** son plantillas que se utilizan para crear objetos de datos según un modelo que definimos.

- Las **clases** representan **entidades** o conceptos. Ejemplo: Un **coche** podría ser representado como una clase, es un objeto que tiene propiedades y métodos.
- Las **clases** definen un conjunto de **variables** (guardan el estado) y **métodos** (referente al comportamiento y que implementan la funcionalidad de una clase).
- Se denomina **instancia** de una clase a un objeto creado a partir de dicha clase.
- Las clases se componen de:
  - **Campos de datos**: Reflejan el estado de la clase y suelen usar variables para almacenarlo. Las variables suelen ser privadas para limitar su acceso.
  - **Métodos**: Implementan la funcionalidad del objeto (de la clase).
  - **Propiedades**: Tipo especial de métodos. Debido a la privacidad de las variables, podemos contar con métodos Get y Set que nos permiten acceder a estas y modificarlas.
    - El **método Set** nos permite modificar una propiedad y le pasamos un valor que modificará el que había en esa propiedad antes.

```java
public class Coche {
    String marca;
    String modelo;
    int potencia;

// Constructor, se llamará cuando se cree la clase
    public Coche() {
        marca = "Ford";
        modelo = "Focus";
        potencia = 150;
    }

// Métodos para insertar valores, conocidos como setters
    public void setMarca(String marca) {
        this.marca = marca;
    }

    public void setPotencia(int potencia) {
        this.potencia = potencia;
    }

// Métodos para obtener valores, conocidos como getters
    public String getMarca() {
        return marca;
    }

    public int getPotencia() {
        return potencia;
    }

}
```

Librerías de java

- Las librerías se componen de clases y cada clase tiene unos métodos para su utilización a la hora de crear interfaces gráficas.
- A continuación, mediante el enlace se pueden comprobar todos los métodos de cada clase en la dirección de la API correspondiente:

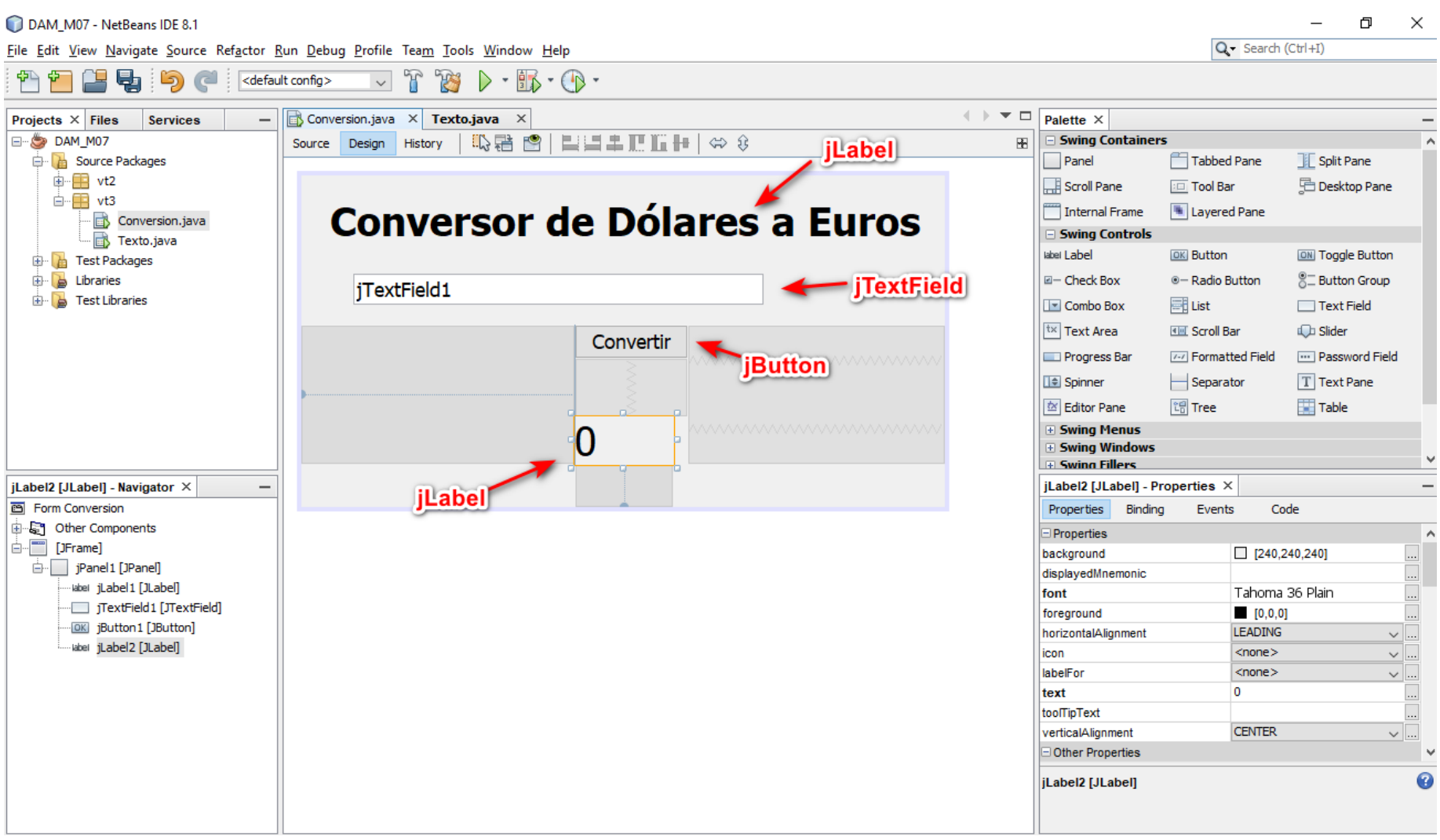
https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html

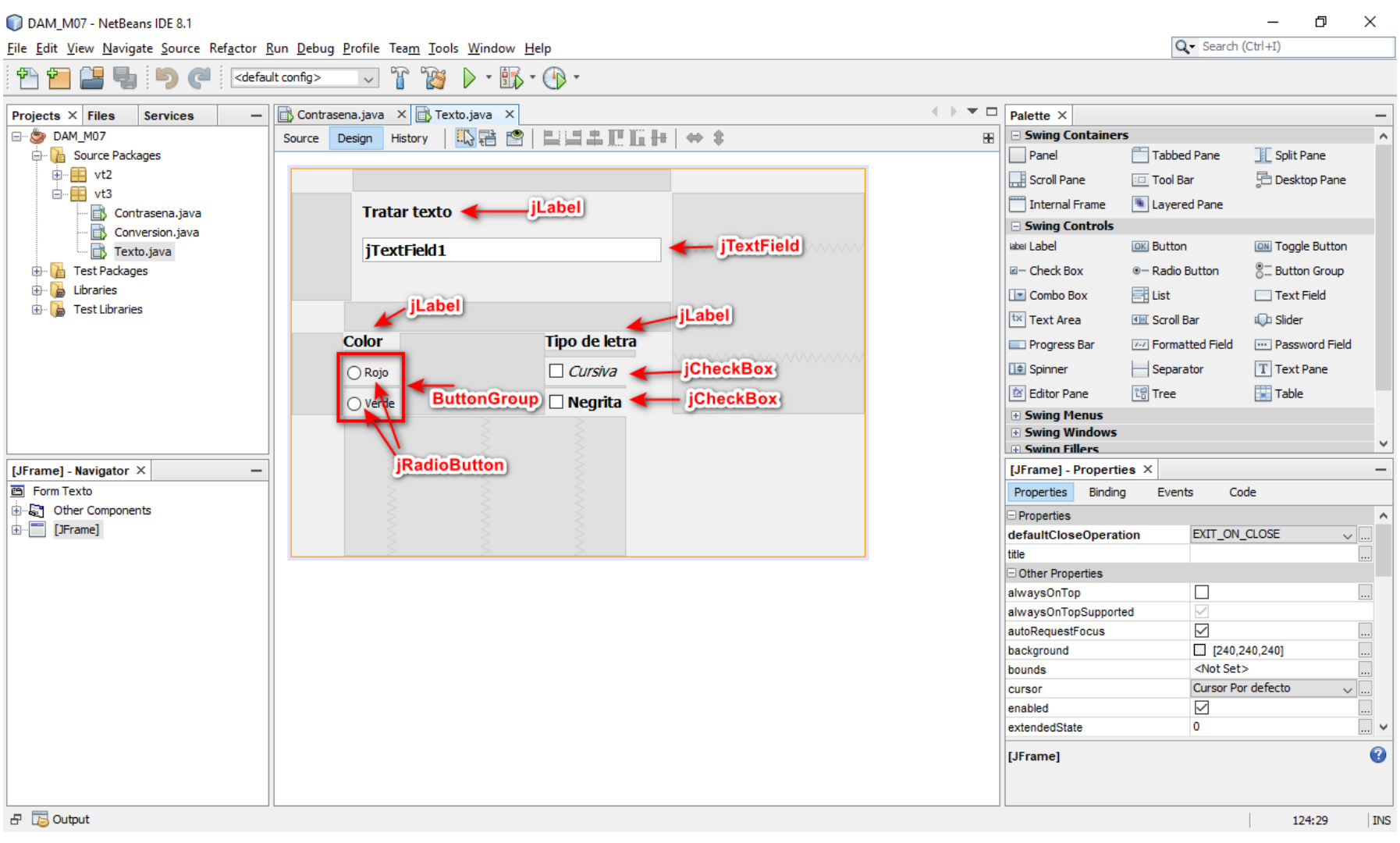
ILERNA Online

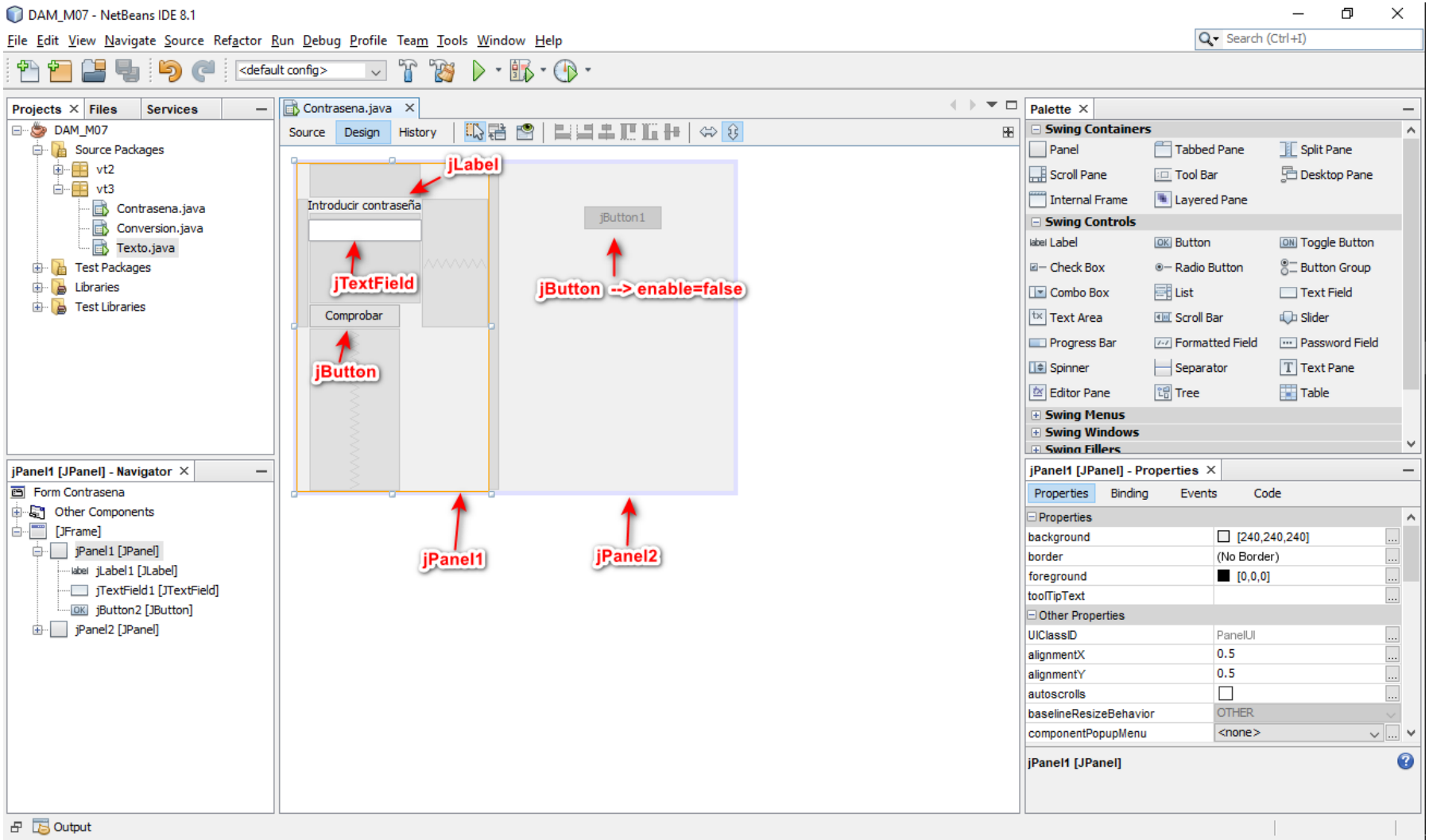
CFGS: Desarrollo de Aplicaciones Multiplataforma

Videotutoría 3

Módulo 07: Desarrollo de Interfaces


DAM_M07 - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects
Files
Services
DAM_M07
Source Packages
vt2
vt3
Conversion.java
Texto.java
Test Packages
Libraries
Test Libraries
Conversion.java
Texto.java
Source
Design
History
jLabel
Conversor de Dólares a Euros
jTextField1
jTextField
Convertir
jButton
0
jLabel
Palette
Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
Spinner
Separator
Text Pane
Editor Pane
Tree
Table
Swing Menus
Swing Windows
Swing Fillers
jLabel2 [JLabel] - Navigator
Form Conversion
Other Components
[JFrame]
jPanel1 [JPanel]
jLabel1 [JLabel]
jTextField1 [JTextField]
jButton1 [JButton]
jLabel2 [JLabel]
jLabel2 [JLabel] - Properties
Properties
Binding
Events
Code
Properties
background
[240,240,240]
displayedMnemonic
font
Tahoma 36 Plain
foreground
[0,0,0]
horizontalAlignment
LEADING
icon
<none>
labelFor
<none>
text
0
toolTipText
verticalAlignment
CENTER
Other Properties
jLabel2 [JLabel]


DAM_M07 - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects
Files
Services
DAM_M07
Source Packages
vt2
vt3
Contrasena.java
Conversion.java
Texto.java
Test Packages
Libraries
Test Libraries
Contrasena.java
Texto.java
Source
Design
History
Tratar texto
jLabel
jTextField1
jTextField
jLabel
jLabel
Color
Tipo de letra
Rojo
Verde
ButtonGroup
Cursiva
Negrita
jCheckBox
jCheckBox
jRadioButton
Palette
Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
Spinner
Separator
Text Pane
Editor Pane
Tree
Table
Swing Menus
Swing Windows
Swing Fillers
[JFrame] - Navigator
Form Texto
Other Components
[JFrame]
[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
[JFrame]
Output
124:29
INS


DAM_M07 - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Projects
Files
Services
DAM_M07
Source Packages
vt2
vt3
Contrasena.java
Conversion.java
Texto.java
Test Packages
Libraries
Test Libraries
Contrasena.java
Source
Design
History
jLabel
Introducir contraseña
jTextField
Comprobar
jButton
jButton1
jButton --> enable=false
jPanel1
jPanel2
Palette
Swing Containers
Panel
Tabbed Pane
Split Pane
Scroll Pane
Tool Bar
Desktop Pane
Internal Frame
Layered Pane
Swing Controls
Label
Button
Toggle Button
Check Box
Radio Button
Button Group
Combo Box
List
Text Field
Text Area
Scroll Bar
Slider
Progress Bar
Formatted Field
Password Field
Spinner
Separator
Text Pane
Editor Pane
Tree
Table
Swing Menus
Swing Windows
Swing Fillers
jPanel1 [JPanel] - Navigator
Form Contrasena
Other Components
[JFrame]
jPanel1 [JPanel]
jLabel1 [JLabel]
jTextField1 [JTextField]
jButton2 [JButton]
jPanel2 [JPanel]
jPanel1 [JPanel] - Properties
Properties
Binding
Events
Code
Properties
background
[240,240,240]
border
(No Border)
foreground
[0,0,0]
toolTipText
Other Properties
UIClassID
PanelUI
alignmentX
0.5
alignmentY
0.5
autoscrolls
baselineResizeBehavior
OTHER
componentPopupMenu
<none>
jPanel1 [JPanel]
Output

Asociación de acciones a eventos

- **Ejemplo de Ratón**

Contrasena.java
→ veremos **mouseClicked**

MÉTODOS	Public void mouseClicked(MouseEvent e)	
	Public void mouseEntered(MouseEvent e)	
	Public void mouseExited(MouseEvent e)	
	Public void mousePressed(MouseEvent e)	
EVENTOS	mouseClicked	Pinchar y soltar
	mouseEntered	Entra en un componente con el puntero
	mouseExited	Sale de un componente con el puntero
	mousePressed	Presiona el botón
	mouseReleased	Suelta el botón
ESCUCHADOR	MOUSELISTENER → Cuando se lleva a cabo alguna acción con el ratón	

Interfaces Microsoft: XAML (XML)

- **XAML** (e**X**tensible **A**pplication **M**arkup **L**anguage o, en español, Lenguaje Extensible de Formato para Aplicaciones) es un lenguaje que permite realizar una descripción gráfica de las interfaces de usuario.
- Se puede aplicar al lenguaje de desarrollo de interfaces para escritorio y para web.
- Este lenguaje está **basado** en **XML** y permite realizar una descripción gráfica de las interfaces de los distintos usuarios (desde el punto de vista gráfico).
- **Uno** de los **principales objetivos**: **SEPARAR** totalmente las **capas de presentación de** la **capa lógica**.
- Elemento raíz: **<Window>…</Window>**
- **Dentro** de esta **etiqueta** es preciso declarar los distintos **espacios de nombres** junto con sus correspondientes referencias.
- Estos espacios de nombres deben asociar los elementos descritos en el documento con los determinados controles de WPF que están en el espacio de nombres System
- **WPF**: Windows Presentation Foundation. Es una tecnología de Microsoft que permite el desarrollo de interfaces de usuario en Windows tomando características de aplicaciones de Windows y de aplicaciones web.

XAML (II)

- **Sintaxis:**
  - https://docs.microsoft.com/es-es/xamarin/xamarin-forms/xaml/xaml-basics/essential-xaml-syntax

```
<Label Text="Hello, XAML!"
       VerticalOptions="Center"
       FontAttributes="Bold"
       FontSize="Large"
       TextColor="Aqua" />
```

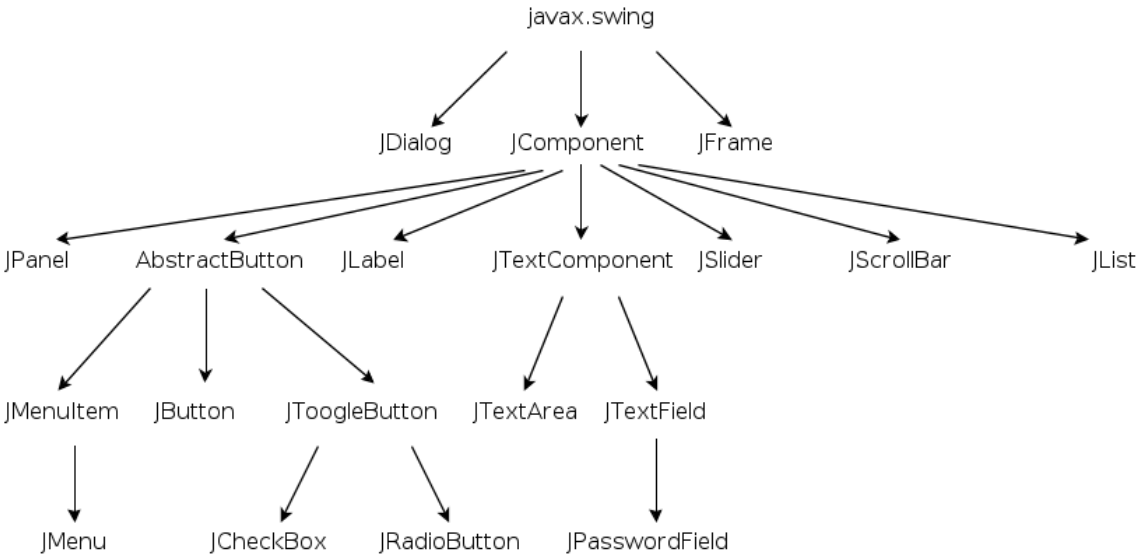
```
<Label Text="Hello, XAML!"
       VerticalOptions="Center"
       FontAttributes="Bold"
       FontSize="Large">
    <Label.TextColor>
        Aqua
    </Label.TextColor>
</Label>
```

Ejemplo XAML

```
<Label>
    <Label.Text>
        Hello, XAML!
    </Label.Text>
    <Label.FontAttributes>
        Bold
    </Label.FontAttributes>
    <Label.FontSize>
        Large
    </Label.FontSize>
    <Label.TextColor>
        Aqua
    </Label.TextColor>
    <Label.VerticalOptions>
        Center
    </Label.VerticalOptions>
</Label>
```

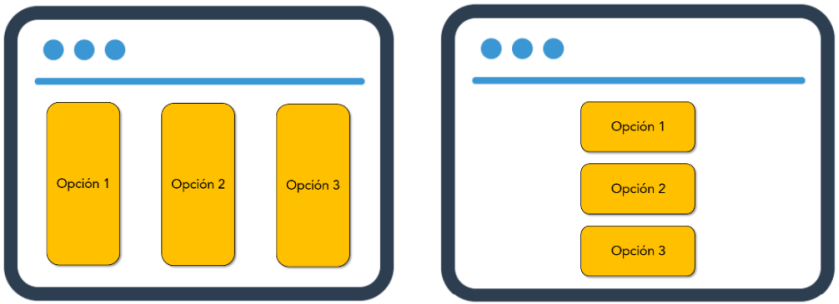
ACLARACIÓN

```
<Label>
    <Label.Text>
        Hello, XAML!
    </Label.Text>
    <Label.FontAttributes>
        Bold
    </Label.FontAttributes>
    <Label.FontSize>
        Large
    </Label.FontSize>
    <Label.TextColor>
        Aqua
    </Label.TextColor>
    <Label.VerticalOptions>
        Center
    </Label.VerticalOptions>
</Label>
```



Contenedores

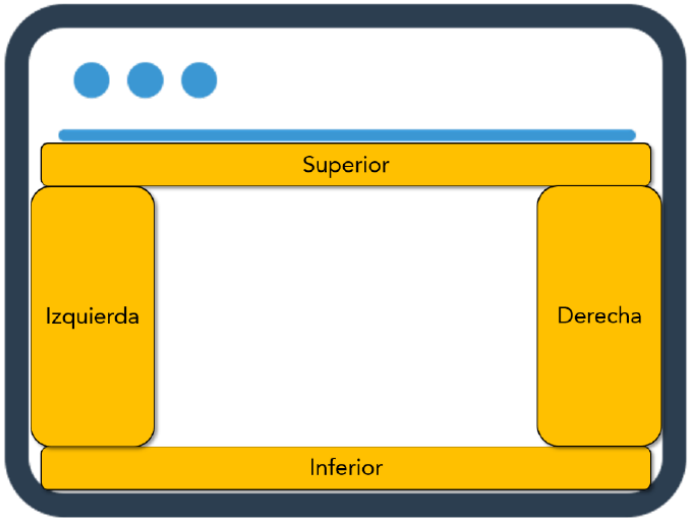
- **StackPanel**: Muy utilizado a la hora de diseñar listas, ya que ofrece la posibilidad de apilar los diferentes elementos de dos formas diferentes:
  - **Orientación horizontal**: uno al lado de otro.
  - **Orientación vertical**: uno encima de otro.



Orientación horizontal *Orientación vertical*

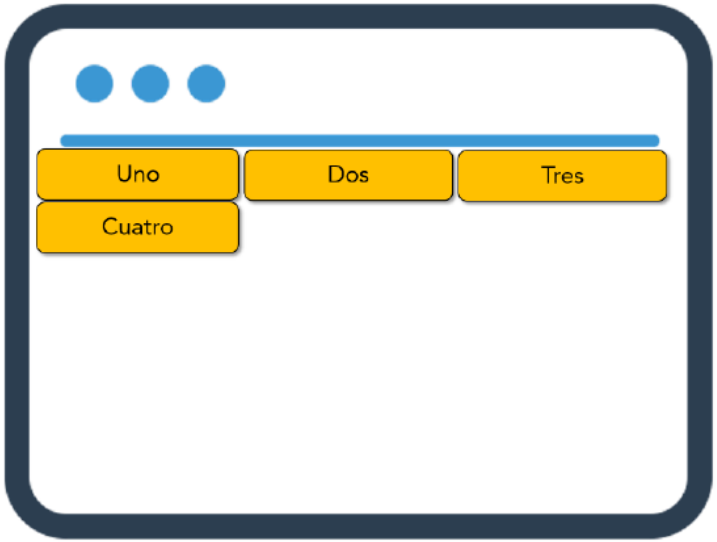
Contenedores

- **DockPanel:** Define un área en la que puede organizar elementos secundarios de forma horizontal o vertical, en relación unos con otros.



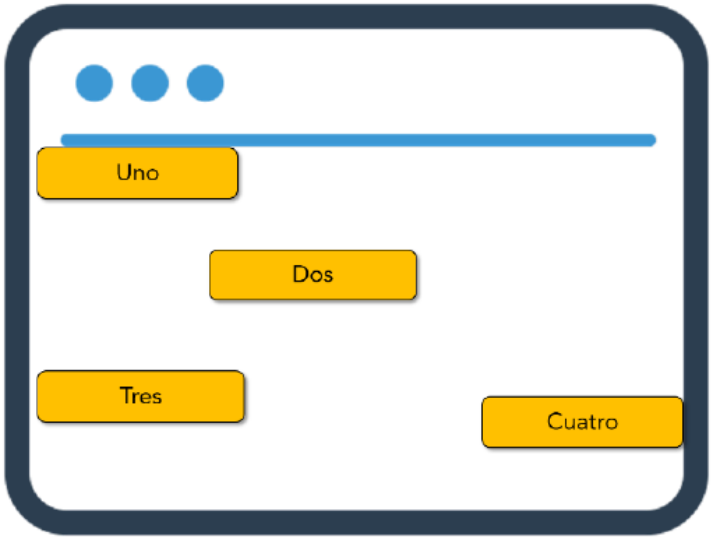
Contenedores

- **WrapPanel:** elemento coloca los elementos secundarios secuencialmente de izquierda a derecha y traslada el contenido a la línea siguiente en el borde del cuadro contenedor.



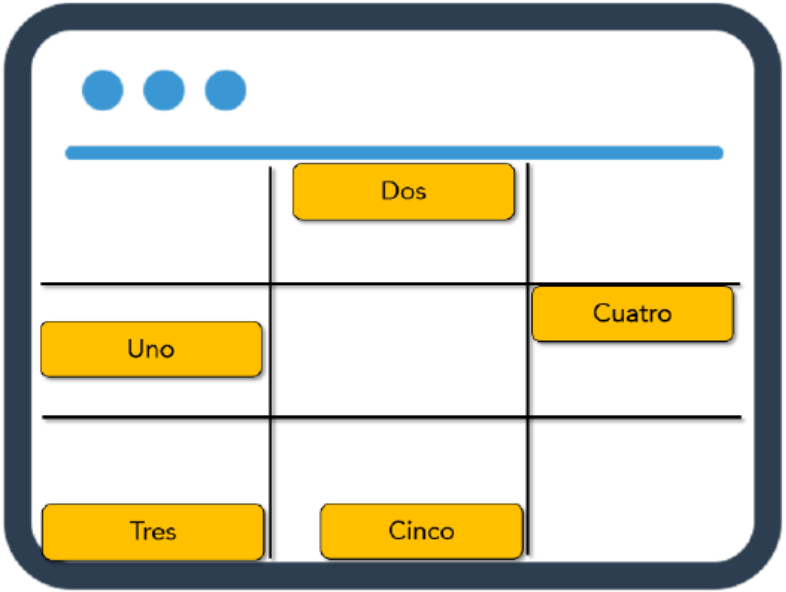
Contenedores

- **Canvas:** Define un área en la cual puede colocar elementos secundarios explícitamente mediante coordenadas relativas al área.



Contenedores

- **Grid:** Define un área de cuadrícula flexible que consta de columnas y filas.



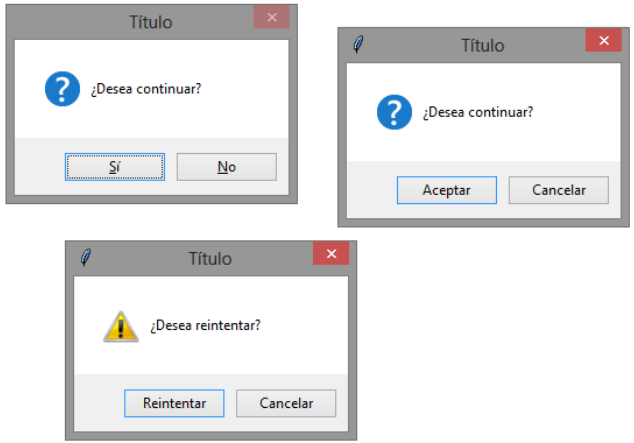
Cuadros de diálogo

- Los **cuadros de diálogo** definen la relación que existe entre la interfaz y el usuario.
- Los principales son:

1. **Mensajes**

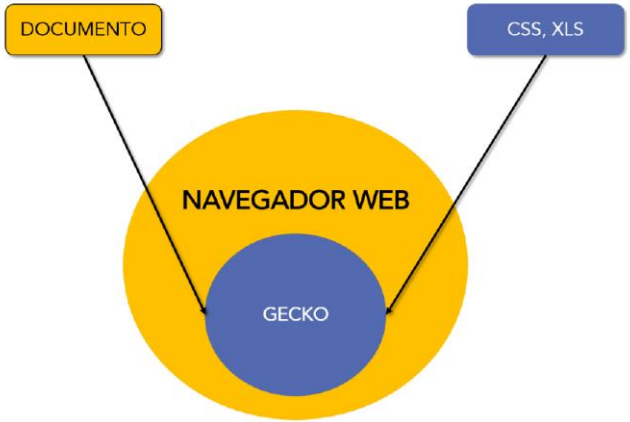
2. **Controles de Diálogo:**

    - Los **botones de diálogo** suelen estar asociados a estas ventanas para notificar, mediante la pulsación, una respuesta de la ventana de diálogo.



XUL

- Aunque hoy en día sigue desarrollándose por parte de Mozilla, es posible **comprobar el funcionamiento de este lenguaje** comúnmente denominado **Gecko**.
- **Toma** datos tipo **HTML**, **XML y** diferente información de formato tipo **CSS o XLS para**, a continuación, **visualizar en pantalla el resultado** que se ha obtenido tras aplicar el formato a los distintos datos.



SVG

- **SVG (Scalable Vectorial Graphics)**

- Lenguaje **basado en XML** que se emplea para **definir gráficos vectoriales** en 2D.

- **SVG ofrece** la posibilidad de **definir** distintas **figuras simples o complejas** que son posibles de especificar **a través de ecuaciones matemáticas o** incluso **expresiones algebraicas.**

UIML

- **UIML (User Interface Markup Language)**
- Fue **creado por** software **Harmonia**, **compañía** que ha **publicado UIML como** un **lenguaje** de **código abierto**.
- UIML es un lenguaje descriptivo que ofrece la posibilidad de crear una página web para utilizarla en cualquier tipo de interfaz o dispositivo.
- UIML se basa en un lenguaje de marcado extensible (XML). Necesita determinar una serie de elementos de la interfaz de usuario (widgets), por lo que es conveniente identificar el conjunto de elementos existentes y sus propiedades.

MXML

- Este **lenguaje se suele utilizar unido a aplicaciones Adobe Flex para llevar a cabo el diseño de interfaces**. Cuenta con una **estructura jerárquica**, en la que el **elemento raíz es el Application** y, **a partir de esta, cuelgan el resto de elementos** de la interfaz.

- **Cuando se compila un archivo MXML, se obtiene** como resultado **un fichero *.swf.**

Parser

- El **parser o procesador de XML** es la **herramienta principal de cualquier aplicación XML**.
- El **parser puede comprobar si nuestros documentos están bien formados o válidos.**
- **También podemos incorporarlos a nuestras aplicaciones**, de manera que estas se puedan manipular y trabajar con documentos XML.

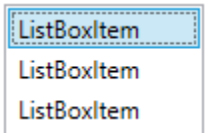
Componentes

- Un **componente** es un **elemento que cuenta con suficiente autonomía y funcionalidad para existir por sí solo**, pudiendo adaptarse a diferentes situaciones.
- Se le denomina **desarrollo de software basado en componentes a** la **manera de unir distintos componentes y desarrollar un determinado código para conseguir un correcto funcionamiento.**
- Sus **principales ventajas**:
  - Reutiliza software.
  - Permite simplificar las pruebas.
  - Permite simplificar el mantenimiento del sistema.
  - Mayor calidad

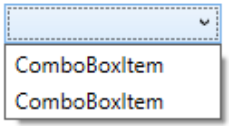
Elementos generales

- **Button**: permite realizar una acción u otra en exclusividad
  - **Content**: selecciona el texto que se va a mostrar en el botón.
  - **IsCancel**: ofrece la posibilidad de asociar el botón a una acción determinada al pulsar la tecla esc del teclado…
  - …

- **ListBox**: lista con una serie de elementos que pueden ser seleccionados por el usuario.
  - **Content**: selecciona el texto que se va a mostrar en el botón.



- **ComboBox**: unión de una caja de texto a una lista con sus propiedades correspondientes y ventajas.

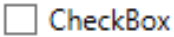


ILERNA Online

Elementos generales(II)

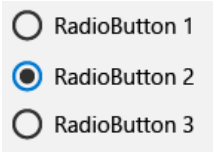
- **TextBox**: caja de texto que ofrece la posibilidad de realizar una determinada operación en la que el usuario actúa de forma interactiva a la hora de introducir algún tipo de información.

- **CheckBox**: determina el elemento que se va a utilizar para definir las diferentes opciones. Es posible seleccionar más de una opción.



Con checkbox no hace falta Buttongroup.

- **RadioButton**: es un elemento muy similar al anterior, en este caso, las distintas selecciones son excluyentes unas de otras.



Buttongroup para seleccionar solo uno de ellos.

- **Image**: este control permite añadir imágenes en una interfaz haciendo referencia a los diferentes recursos.

Componentes en WPF y eventos

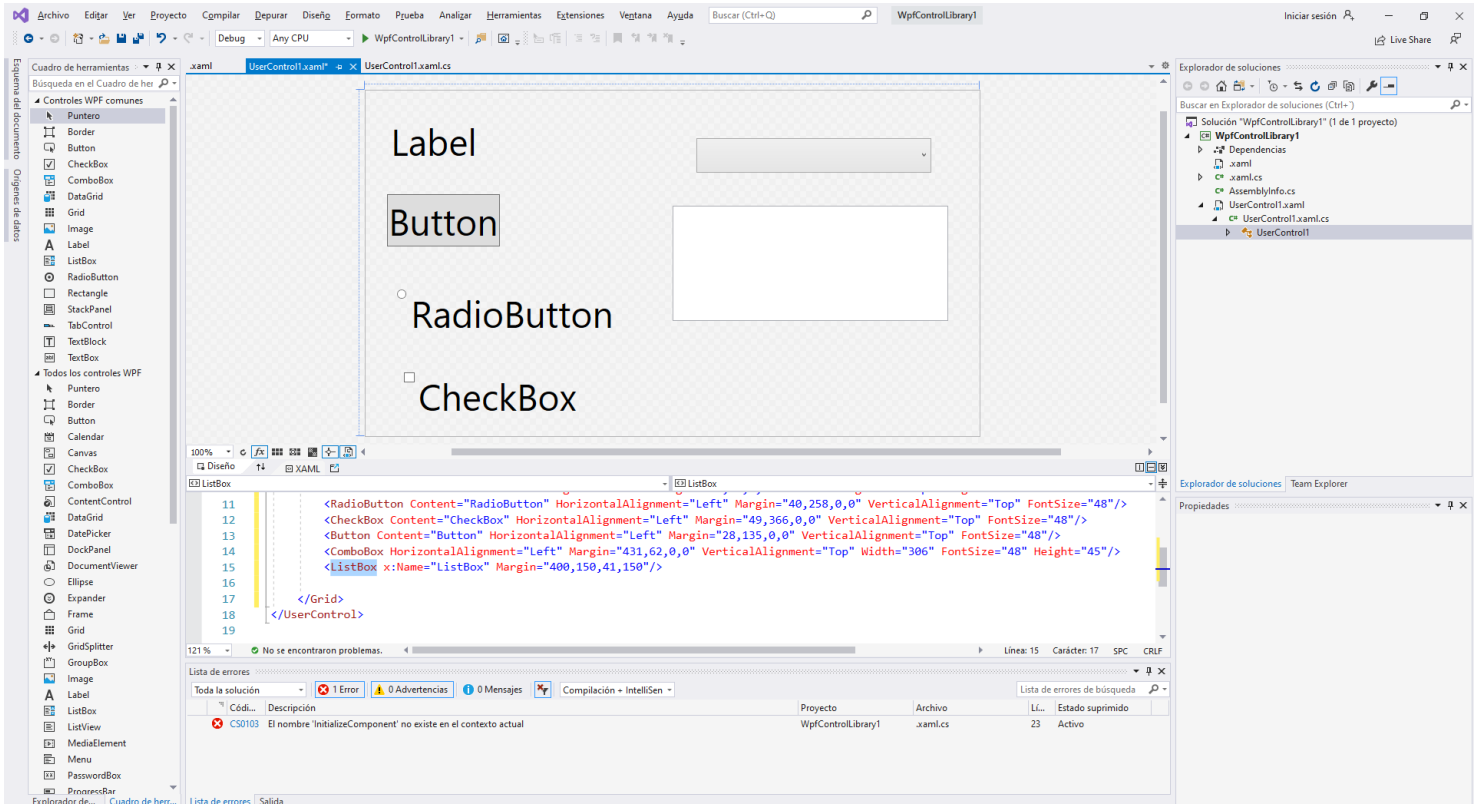
- A la hora de crear un nuevo componente gráfico **WPF (Window Presentation Foundation), se ofrece al desarrollador la opción de partir de la clase UserControl** o **reutilizar algún control o incluso varios**.

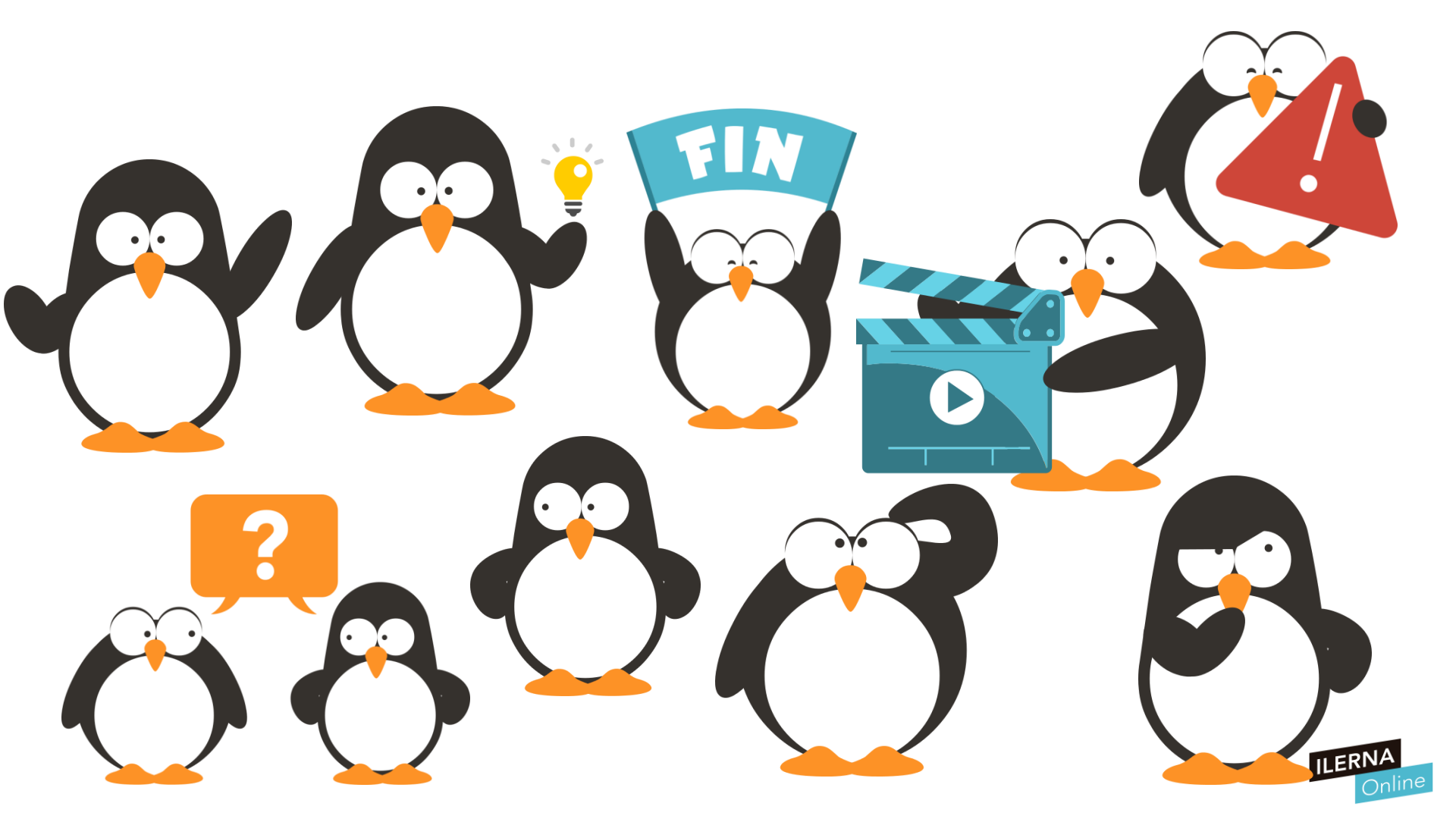
- **Definición de propiedades:** las propiedades asociadas a los componentes gráficos no son muy complejas de definir, solo es necesario utilizar aquellas propiedades determinadas de una clase para hacer uso de sus comportamientos.

- **Eventos:** Se refieren a una acción determinada que puede desempeñar un usuario. Esta acción está asociada a un componente específico.

- Un evento es la función que se desencadena tras haber realizado una acción determinada sobre un componente en cuestión.

Componentes en WPF y eventos (II)




FIN
?
ILERNA
Online

ILERNA Online

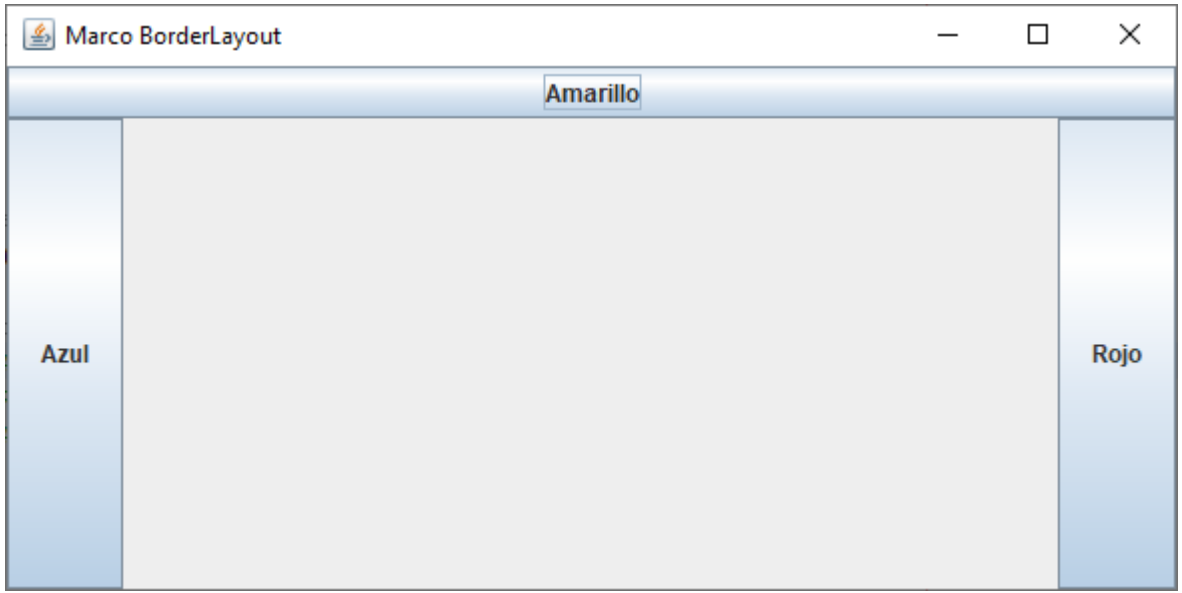
CFGS: Desarrollo de Aplicaciones Multiplataforma

Videotutoría 4

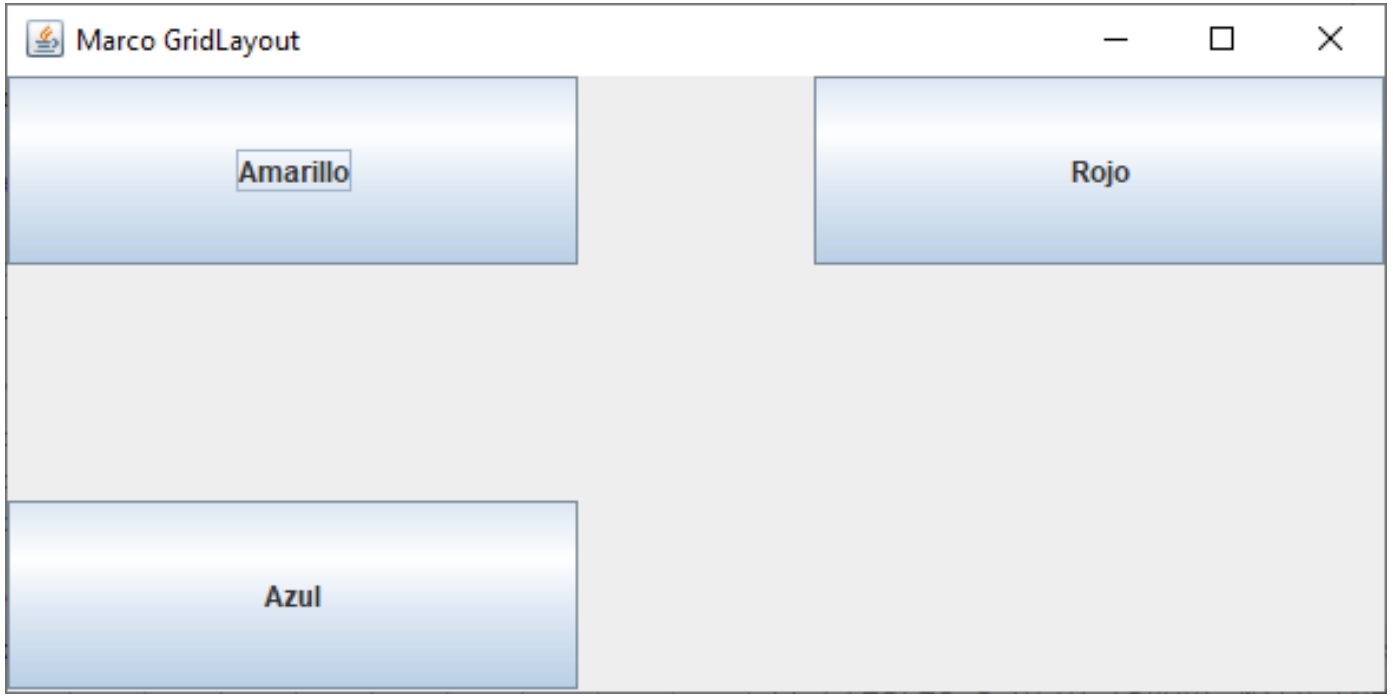
Módulo 07: Desarrollo de Interfaces

FlowLayout (SWING)

BorderLayout (SWING)

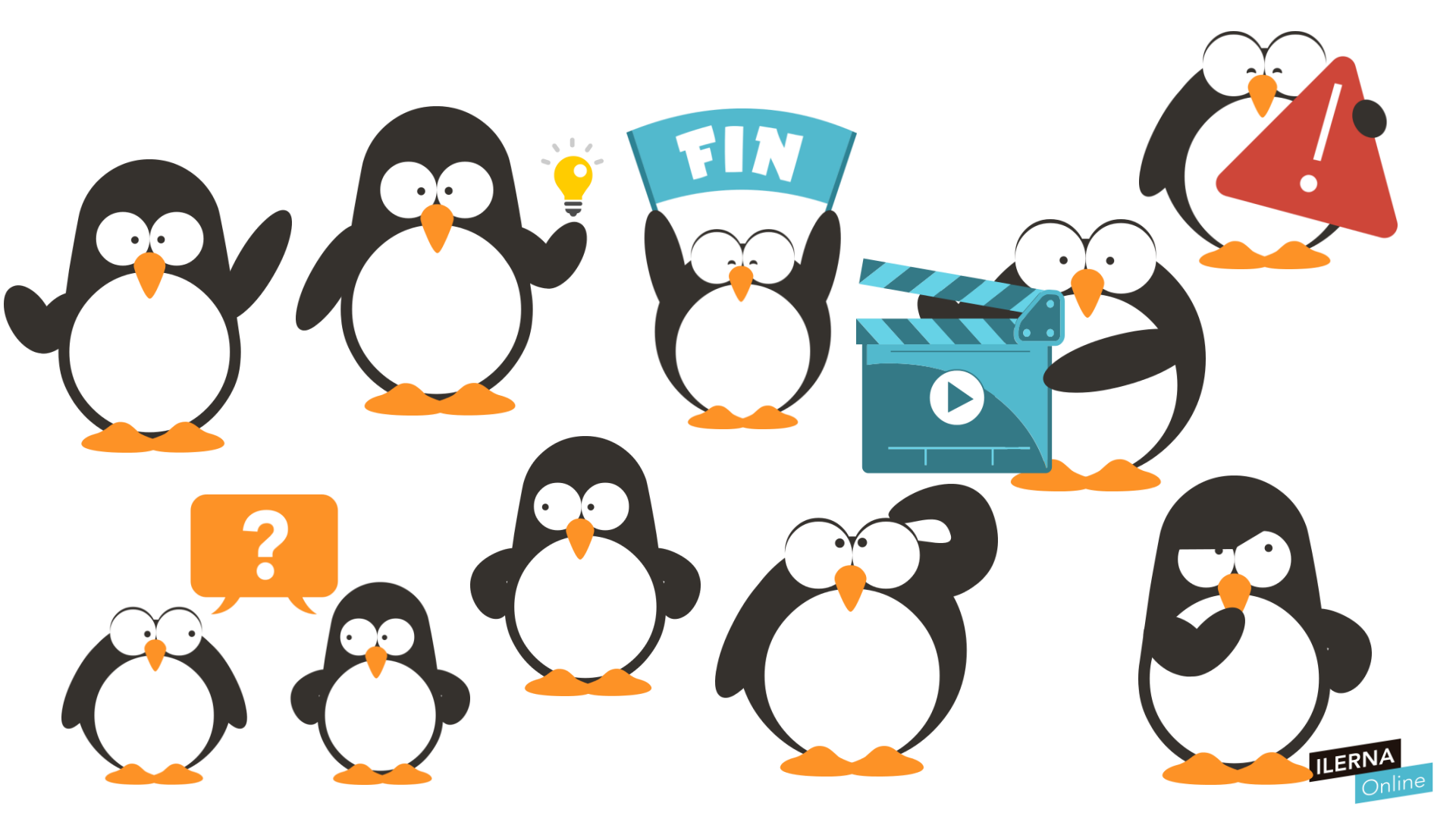


GridLayout (SWING)



Extra

ILERNA Online


FIN
?
ILERNA
Online

ILERNA Online

CFGS: Desarrollo de Aplicaciones Multiplataforma

Videotutoría 5

Módulo 07: Desarrollo de Interfaces

UF1: DISEÑO E IMPLEMENTACIÓN DE INTERFACES

Tema 4. Usabilidad.

Concepto de usabilidad

Usabilidad se refiere a la facilidad con que las personas pueden utilizar una herramienta particular o cualquier otro objeto fabricado por humanos con el fin de alcanzar un **objetivo** concreto.

En la interacción persona ordenador, la usabilidad se refiere a la claridad y elegancia con que se diseña la interacción con un programa de ordenador o un sitio web.

La usabilidad **puede ser medida y evaluada**.

Podemos definir la usabilidad como la medida en la cual un producto puede ser usado por usuarios específicos para conseguir objetivos específicos con efectividad, eficiencia y satisfacción en un contexto de uso especificado.

¿Qué nos proporciona la usabilidad?

- **Facilidad de aprendizaje**: los usuarios nuevos deben poder interactuar de forma efectiva con el sistema.
- **Eficiencia**: el usuario debe poder hacer uso de la herramienta de forma sencilla y utilizando el menor número de pasos posibles para conseguir llegar a un sitio determinado.
- **Cualidad de ser recordado:** si al volver a usarla, el usuario se adapta rápidamente.
- **Eficacia**: grado de ayuda con la que cuenta el usuario para realizar las diferentes acciones en el programa gráfico.
- **Satisfacción**: ¿Es agradable para el usuario?

Usabilidad

- La interfaz gráfica de usuario (GUI) hace referencia a cómo los distintos elementos gráficos permiten comunicarse con un determinado sistema informático.
- **Objetivos:**
  - **Funcionalidad:** se encarga de facilitar las tareas que se llevan a cabo.
  - **Amistosa:** cuenta con un manejo bastante sencillo.
- **Diseño de una interfaz:**
  - Sencilla/Clara/Predecible/Flexible/Consistente/Intuitiva/Coherente

Atributos más usados en una Interfaz

- Tiempo de aprendizaje.
- Eficiencia de uso.
- Retención a través del tiempo.
- Confiabilidad en el uso.
- Satisfacción a largo plazo.
- Eficacia.
- Entendimiento.
- Adaptabilidad.
- Primera impresión.
- Elementos extra.
- Desempeño inicial.
- Evolucionable.

Problemas: Jakob Nielsen

Los 10 principios de Jakob Nielsen:

1. **Visibilidad del estado del sistema**: la página web debe siempre hacer una retroalimentación al usuario para hacerle saber que está pasando en todo momento y en qué punto de navegación se encuentra. Por ejemplo, indicar con un color diferente el elemento de menú en el cual nos encontramos.
2. **Relación entre el sistema y el mundo real**: para que el sistema se pueda comunicar con el mundo real, estos deben hablar el mismo idioma. Por tanto, será preciso unificar la información con la que se cuenta para conseguir establecer esta comunicación de forma natural.
3. **Control y libertad de usuario**: es conveniente ofrecer las opciones de “deshacer” y “rehacer” para volver fácilmente a un estado anterior en caso de error.
4. **Consistencia y estándares**: no es tarea de los usuarios cuestionar si las acciones y las palabras diferentes significan lo mismo.
5. **Prevención de errores**: es más importante prevenir los errores que diseñar unos buenos mensajes de error.

Problemas: Jakob Nielsen(II)

6. **Reconocimiento antes que recuerdo**: tanto los objetos como las acciones y las opciones deben ser visibles. No es tarea del usuario el recordar aquella información que le permite continuar. Para ello, debe contar con instrucciones visibles.
7. **Flexibilidad y eficiencia de uso**: mediante el teclado y combinando distintas teclas, es posible interactuar de forma más rápida.
8. **Estética y diseño minimalista**: los diálogos no deben contar con información que no sea importante.
9. **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores**: los distintos mensajes de error deben ser entregados en un lenguaje conciso y claro.
10. **Ayuda y documentación**: es recomendable contar con una serie de ayuda a la que poder recurrir en caso de error. Esta información no debe ser muy complicada y debe ser de fácil acceso.

- https://www.nngroup.com/people/jakob-nielsen/

Beneficios

- Reducción de costes
- Disminuye la tasa de errores
- Optimiza costes y mantenimiento
- Calidad
- Mejora imagen y prestigio
- Estándares de la usabilidad:
    - Se pueden encontrar una gran cantidad de estándares relativos a la usabilidad (estándar ISO 9241 y otros cuantos más).

Medidas de usabilidad

La **función principal** de las **medidas de usabilidad** es valorar el grado al que han conseguido llegar tanto el usuario como las organizaciones para que se pueda proporcionar una información a fin de ser utilizada por los diseñadores y desarrolladores con la intención de mejorar el estado de la interfaz.

La evaluación de estas medidas se basan en:

- El **usuario**: ofrece información relacionada con la tarea que se va a realizar.
- El **experto**: describe la falta de conformidad que existe con las normas o directrices de diseño de la interfaz.

Medidas de usabilidad

- **2 objetivos** principales:

  1.- Diagnóstico de problemas:

  - Aquellos métodos que están basados en el **usuario**, como:
    - Evaluación participativa.
    - Evaluación de diagnóstico.
    - Análisis de incidentes críticos.
  - Aquellos que se pueden completar por el **experto**

Medidas de usabilidad(II)

- **2 objetivos** principales:

  2.- Evaluación para comprobar si se han conseguido o no los objetivos referentes a la usabilidad

  • Los requisitos para satisfacción y desarrollo del usuario se pueden evaluar mediante la utilización de pruebas de rendimiento, carga de trabajo cognitivo, etc.

  • Otros objetivos diferentes de usabilidad se puedan evaluar a través de la evaluación de expertos.

  Las herramientas más utilizadas para la evaluación suelen ser, en muchos de los casos, las *entrevistas, test y cuestionarios*

Tipos de métricas

- **Directa**:
  - Los atributos se miden a través de métricas directas:
    - **Longitud del texto del cuerpo de una página**: medido por la cantidad de palabras.
    - **Cantidad de enlaces rotos internos de un sitio web**: medido según la presencia de errores del tipo 404.
    - **Cantidad máxima de frames que tiene un sitio web**: medidos por la presencia de etiquetas <FRAMESET>.
    - **Cantidad de imágenes con texto alternativo de un sitio web**: medido por las propiedades ALT que existen en cada una de las imágenes que están vinculadas al sitio web.
- **Indirecta**: se refiere a la relación que existe entre dos o más atributos.
  - **- Porcentaje de los enlaces rotos de un determinado sitio**
    - *(N° enlaces rotos internos + N° enlaces rotos externos) * 100 / N° total enlaces*
  - **- Porcentaje de la presencia de la propiedad ALT**
    - *(N° imágenes ALT / N° total imágenes) * 100*

Tipos de métricas(II)

- **Interna**
  - Se refiere a un valor numérico del atributo que involucra al valor en sí, independientemente de si es obtenido por una métrica directa o indirecta.
- **Externa**
  - Se refiere a un valor resultante del atributo cuando se aplica una métrica indirecta. En este caso, involucra al sujeto junto al comportamiento con el entorno.
- **Objetiva**
  - Se refiere a un valor resultante del atributo de un determinado sujeto. En este tipo de métrica, es posible distinguir entre diferentes grados de objetividad.
- **Subjetivas**
  - Por último, la métrica subjetiva se refiere a un valor numérico que siempre involucra al usuario mediante heurísticas o distintos criterios de preferencia.

Pautas de diseño de interfaces

- Existen una serie de **recomendaciones** que se pueden verificar de forma gráfica:
  - **Jerarquía**: clasificación del orden de los elementos principales.
  - **Foco**: determina una zona de bastante importancia. Se sitúa en la esquina superior izquierda.
  - **Homogeneidad**: la estructura de las diferentes ventanas debe ser uniforme para conseguir una mejor interpretación de la interfaz.
  - **Relaciones entre elementos**: es conveniente organizar los elementos que estén relacionados en la misma zona de forma limpia y concisa. De esta manera, se facilitará la lectura de la interfaz.

Cuadro de diálogo y desplegables

- Es conveniente que los **cuadros de diálogo** sean lo más sencillos posibles, ya que su función principal es simplificar la resolución de una función que se encuentra dentro de la interfaz principal.
- Es conveniente hacer uso de las **listas desplegables** ya que ofrecen la posibilidad de poder seleccionar uno o algunos de los elementos, según las necesidades que existan en cada momento.
    - ***Flujo de ejecución***: ofrece al usuario, de forma clara, el flujo de ejecución que puede existir entre las diferentes ventanas.
    - ***Integración***: permite crear diferentes elementos gráficos en lugares en los que se va a utilizar.

Fuentes, colores, gráficos

- **Fuentes**:
  - Es conveniente utilizar una **semántica clara** para el texto. Si se van a referenciar elementos iguales, es aconsejable la utilización de la misma semántica y así dejar constancia de que se va a realizar la misma acción, evitando la ambigüedad.
  - Es preciso tener **estandarizado el uso de fuentes**, ya que, si se emplean muchos tipos distintos, se dificulta la lectura al usuario.
  - Se debe utilizar un **lenguaje** que sea apto para todo el público, buscando los conceptos clave como los más significativos y evitando el uso de palabras negativas como error, fallo, etc.
  - En caso de que se produzca algún error, es preferible resaltar cómo se va a solucionar, en lugar de a qué corresponde ese tipo de error.

Fuentes, colores, gráficos(II)

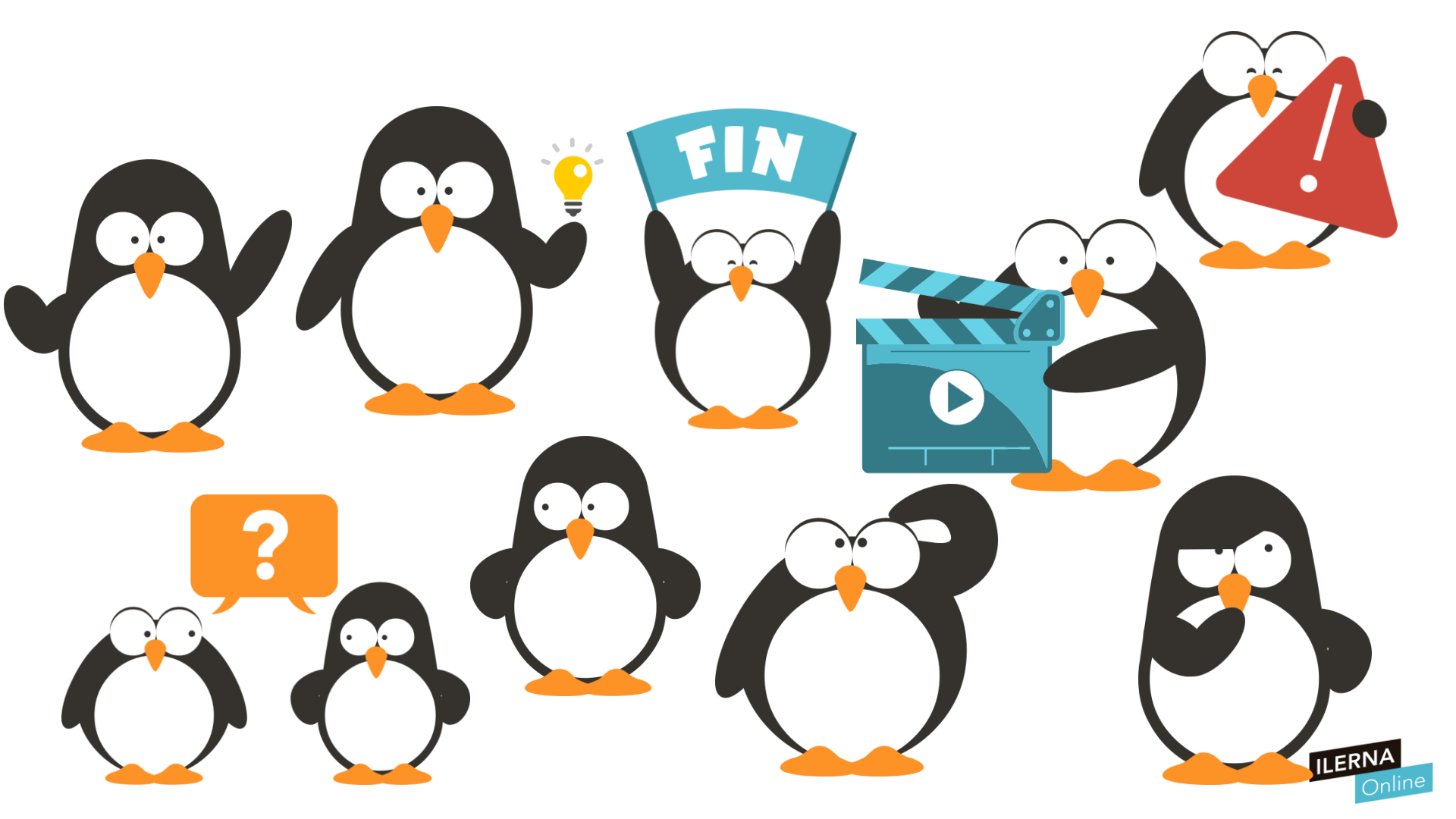
- **Colores**:
  - El uso de los diferentes colores puede provocar una mejor percepción de la interfaz, ya que permite resaltar aquellas tareas principales. Aunque, un abuso de ellos, produce un efecto rebote, haciendo que sea más complicada la lectura de la interfaz.

- **Gráficos**:
  - Deben conseguir captar la atención del usuario.
  - Optimizar el espacio: transmiten una determinada función sin llegar a especificarlo mediante un texto.

W3C

- Se tienen en cuenta la web que ya se encuentra diseñada, para que todos los usuarios puedan interactuar con ella, independientemente del software que se utilice, del lenguaje y la localización
- *"El poder de la web radica en su universalidad. Su acceso universal a pesar de las diferentes discapacidades es un elemento esencial".*
- Estos **criterios de éxito** están ordenados según su **nivel de cumplimiento (A, AA, AAA)**. Para que cada página web cumpla con estas **Pautas WCAG 2.0**, tiene que cumplir una serie de requisitos de conformidad:
  - **Nivel de conformidad "A":** se satisfacen todos los puntos de verificación correspondientes a la prioridad 1.
  - **Nivel de conformidad "AA" (Doble A):** se satisfacen todos los puntos de verificación correspondientes a las prioridades 1 y 2.
  - **Nivel de conformidad "AAA" (Triple A):** se satisfacen todos

http://accesibilidadweb.dlsi.ua.es/?menu=dec-configuracion


FIN
?
ILERNA
Online

VIDEOCLASE 6: métricas

Módulo 07: Desarrollo de interfaces

UF1:Diseño e implementación de interfaces

Métrica y Usabilidad

Definiciones

Medida!=Métrica!=Indicador

Medida: indicio cuantitativo de la extensión, cantidad, dimensión, capacidad o tamaño de algún atributo de un producto o proceso

- Ejemplo.: Número de errores dentro de un diagrama UML
- **Medición**: acto de determinar una medida

No solo se mide código, también se puede medir documentación

Métrica: "una medida cuantitativa del grado en el que un sistema, componente o proceso posee un atributo determinado"

- Relaciona en alguna forma las medidas individuales:
- E.g.: Número promedio de errores que se encuentran por revisión o tasa de fallos, tiempo medio entre fallas, etc.

Indicador: métrica o combinación de métricas que proporcionan comprensión acerca del proceso de software, el proyecto de software o el producto en sí.

- Si la tasa de fallos de las pruebas > 0.2 se revisará el producto SW

Atributos de las métricas

- Las métricas solo son útiles si están caracterizadas y se validan
- Según Lethbridge, una métrica tiene que tener propiedades matemáticas deseables.
    - Por ejemplo, si estamos representando un valor entre 0 y 1, significa que 0 es el valor mínimo, 1 el valor máximo y 0,5 el valor medio
- **Simple y calculable**: fácil de calcular sin mucho esfuerzo ni tiempo
- **Intuitiva**: si medimos el acoplamiento entre módulos, debe ser el menor posible
- **Sin ambigüedades**
- **Constante en su uso:** ¿tiene que ver el número de personas de un proyecto con el número de variables de un módulo?
- **Independiente del lenguaje de programación:** se basa en los requerimientos y en el modelo de diseño
- **Medir →¿Implica calidad?** **Revisar Kahoot para el examen. También métricas.**

Tipos de métricas

- **Directa:**
  - Los atributos se miden a través de métricas directas:
    - Longitud del texto del cuerpo de una página: medido por la cantidad de palabras.
    - Cantidad de enlaces rotos internos de un sitio web: medido según la presencia de errores del tipo 404.

- **Indirecta: se refiere a la relación que existe entre dos o más atributos.**

  - - Porcentaje de los enlaces rotos de un determinado sitio

    - (Nº enlaces rotos internos + Nº enlaces rotos externos) * 100 / Nº total enlaces

  - - Porcentaje de la presencia de la propiedad ALT

    - (Nº imágenes ALT / Nº total imágenes) * 100

Otras métricas

Métricas asignadas como cuantitativas por **Halstead***, se derivan después de que se ha generado el código o se estima una vez que el diseño esté completo.

Un programa esta compuesto de "tokens", las instrucciones del lenguaje, los identificadores, constantes, operadores delimitadores de comentario y signos especiales del mismo. De esta forma se obtiene una medida más realista de la cantidad de información contenida en el código fuente.

* https://www.rcac.purdue.edu/compute/halstead/bio/

Otras métricas(II)

Las medidas son:

n1: el número de operadores diferentes que aparecen en el programa. ! && || + IF == ()

n2: el número de operandos diferentes que aparecen en el programa. Las variables

N1: el número total de ocurrencias de operadores

N2: el número total de ocurrencias de operandos

Cálculos

Longitud (N):

- *Se calcula como, N = N1 + N2*
- *Es una simple medida del tamaño de un programa.*
- *Cuanto más grande sea el tamaño de N, mayor será la dificultad para comprender el programa y mayor el esfuerzo para mantenerlo.*

Volumen (V):

- *Da un peso extra al número de operadores y operandos únicos. Por ejemplo, si dos programas tienen la misma longitud N pero uno tiene mayor número de operadores y operandos únicos, que naturalmente lo hacen más difícil de entender y mantener, este tendrá un mayor volumen.*
- *Se calcula como V = N x log2(n) donde n = n1 + n2*

Cálculos(II)

Dificultad (D):

- *Para definir la dificultad D del programa, se usa la fórmula siguiente: D = (n1 * N2) / (n2 *2).*

Esfuerzo (E):

- *El esfuerzo es otra medida estudiada por Halstead que ofrece una medida del trabajo requerido para desarrollar un programa.*
- *Desde el punto de vista del mantenimiento, el esfuerzo se puede interpretar como una medida del trabajo requerido para comprender un software ya desarrollado.*
- *La fórmula es la siguiente: E = D * V ó V / L*

Ejemplo

```
if (N < 2)
{
A = B * N;
System.out.println("El resultado es : " + A);
}
```

n_1 = IF, <, () **(los de la salida, los otros son obligatorios)**, *, ;, =, salida, +, {} **= 9**
n_2 = N, 2, A, B, String **= 5**
n_1 = 1 + 1 +1 +1 +2 +1 +1 +1 +1 **= 10** (número de veces que aparece cada uno de los elementos contados en el las líneas anteriores)
n_2 = 2 + 1 + 2 + 1 + 1 **= 7**
Longitud = 17

Solución

Ejercicio

```
int sort (int x[ ], int n)

{
    int i, j, save, im1;
    /*This function sorts array x in ascending order */
    If (n< 2) return 1;
    for (i=2; i< =n; i++)
    {
        im1=i-1;
        for (j=1; j< =im1; j++)
            if (x[i] < x[j])
            {
                Save = x[i];
                x[i] = x[j];
                x[j] = save;
            }
    }
    return 0;
}
```

Métricas de orientación a objetos

- **Complejidad**: características estructurales al examinar cómo se relacionan mutuamente las clases de un diseño OO
- **Acoplamiento**. Las conexiones físicas entre elementos del diseño OO (por ejemplo, el numero de colaboraciones entre clases o el de mensajes que pasan entre los objetos) representan el acoplamiento dentro de un sistema OO.
- **Suficiencia**. Un componente de diseño (e.j.: clase) es suficiente si refleja por completo todas las propiedades del objeto de dominio de aplicación que se modela
- **Completitud**. Más general que la suficiencia que sólo se centra en la aplicación actual que se desarrolla.
- **Cohesión**. Un componente OO debe diseñarse de manera que tenga todas las operaciones funcionando en conjunto para lograr un solo propósito bien definido

Métricas de orientación a objetos

- **Primitivismo**: grado en el que una operación es atómica La operación no puede construirse a partir de una secuencia de otras operaciones contenidas dentro de una clase (clase con alto grado de primitivismo encapsula sólo operaciones primitivas

- **Similitud.** El grado en el que dos o mas clases son similares en términos de su estructura, función, comportamiento o propósito se indica mediante esta medida.

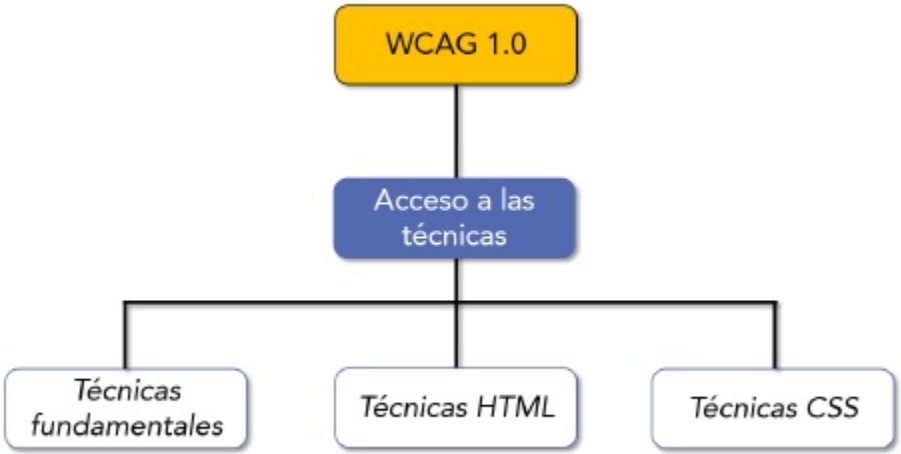
- **Volatilidad.** De un componente de diseño OO mide la probabilidad de que ocurrirá un cambio

W3C

- *Se tienen en cuenta la web que ya se encuentra diseñada, para que todos los usuarios puedan interactuar con ella, independientemente del software que se utilice, del lenguaje y la localización*
- *"El poder de la web radica en su universalidad. Su acceso universal a pesar de las diferentes discapacidades es un elemento esencial".*
- *Estos criterios de éxito están ordenados según su nivel de cumplimiento (A, AA, AAA). Para que cada página web cumpla con estas Pautas WCAG 2.0, tiene que cumplir una serie de requisitos de conformidad:*
    - *• Nivel de conformidad "A": se satisfacen todos los puntos de verificación correspondientes a la prioridad 1.*
    - *• Nivel de conformidad "AA" (Doble A): se satisfacen todos los puntos de verificación correspondientes a las prioridades 1 y 2.*
    - *• Nivel de conformidad "AAA" (Triple A): se satisfacen todos*

http://accesibilidadweb.dlsi.ua.es/?menu=criterios-2.0

W3C(II)




FIN
?

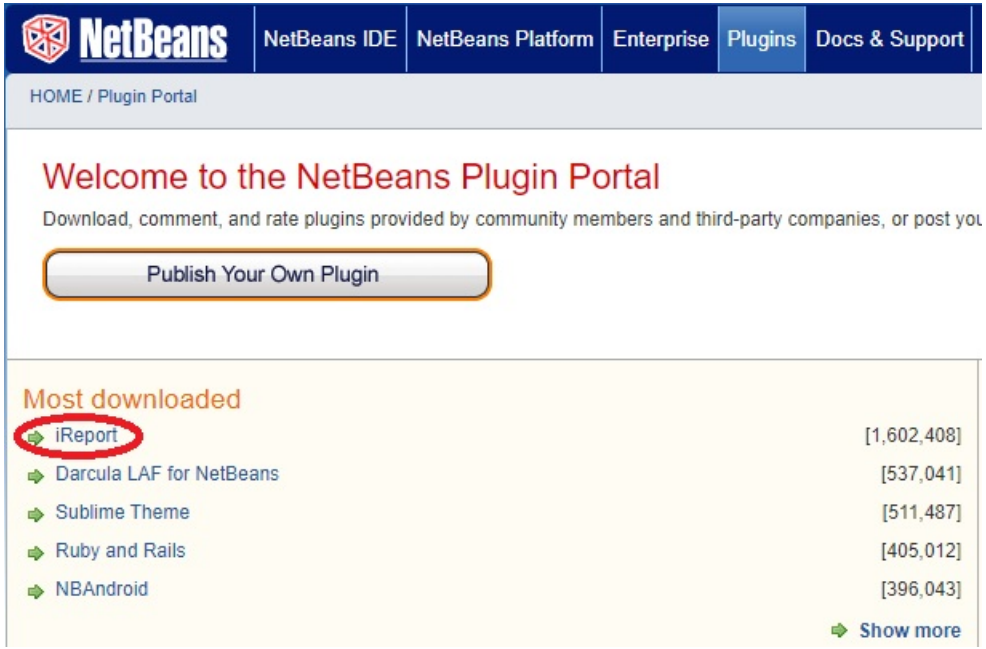
VIDEOCLASE 7: informes (reportes)

Módulo 07: Desarrollo de interfaces

UF1:Diseño e implementación de interfaces

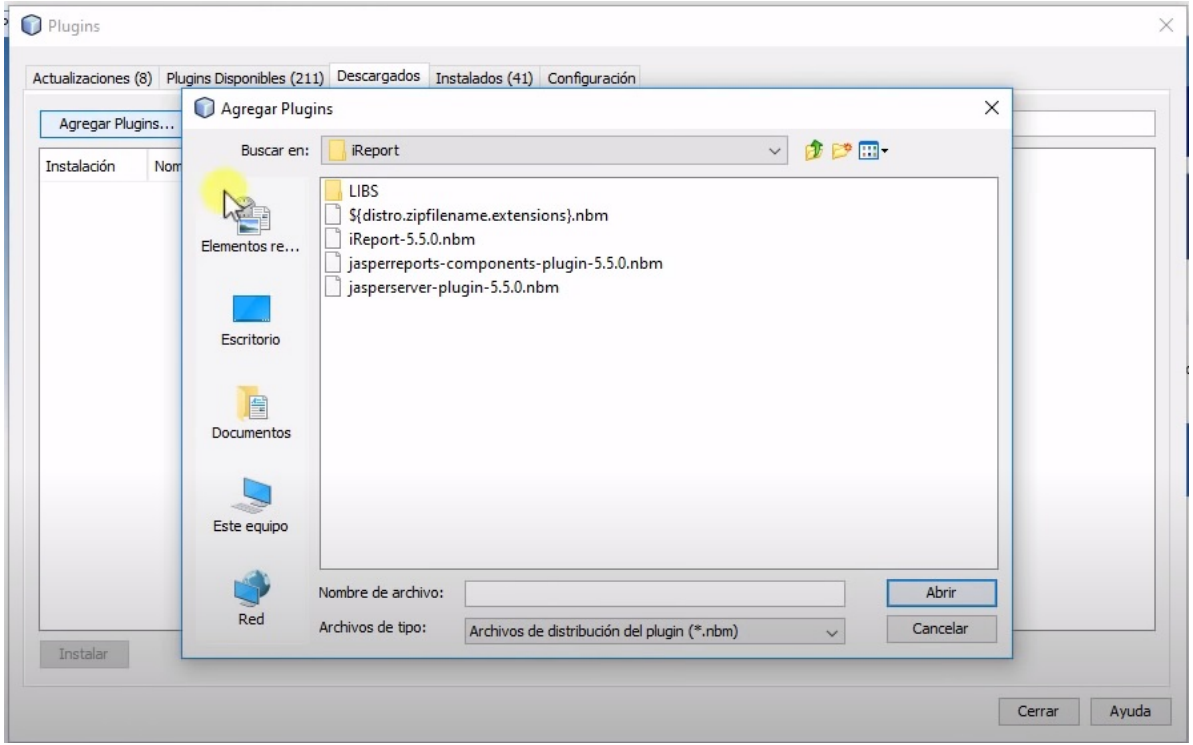
Reportes con Netbeans y iReports

Plugin: iReports



ILERNA Online

Plugin: iReports



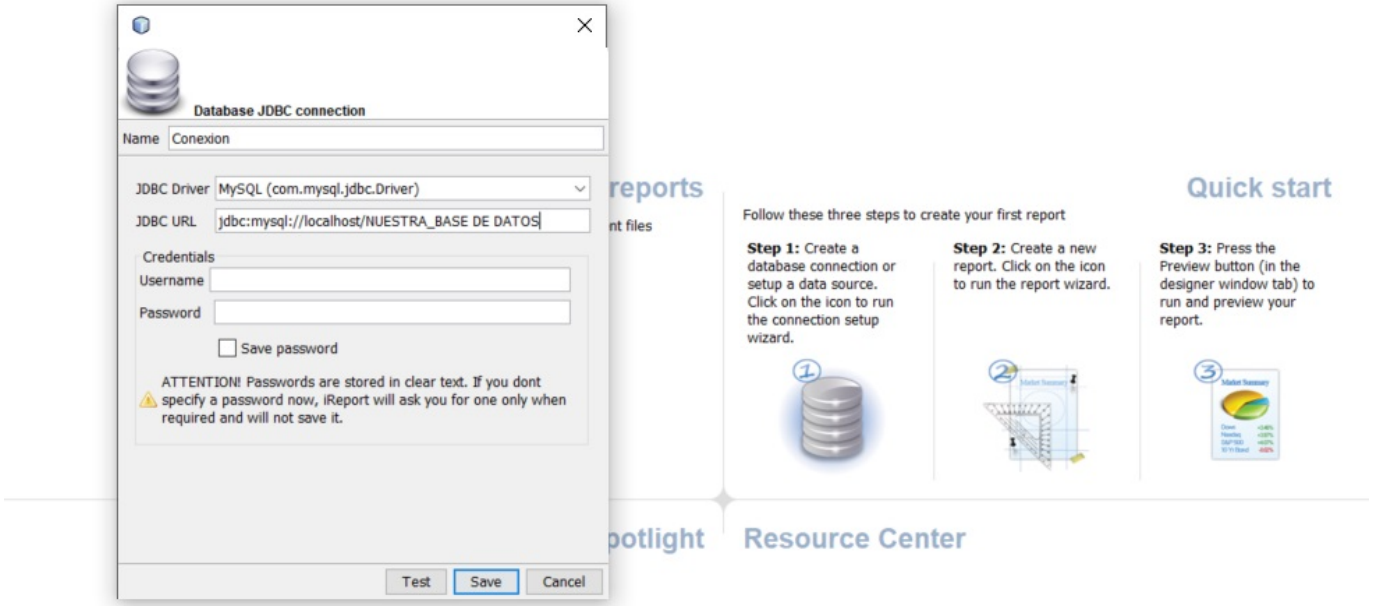
ILERNA Online

Inicio de iReports

Vamos a conectar a una BBDD

- En este ejemplo, trabajaremos con MySQL y con el IDE MySQL Workbench
- Desde una conexión local, crearemos una base de datos con las tablas que queramos mostrar en nuestro reporte
- Dentro de netbeans, seleccionaremos en la ventana “Welcome Window” la conexión hacia esa bbdd
- Nuestro driver será el JDBC de MySQL
- Y nuestra URL, donde tengamos nuestra base de datos

Vamos a conectar a una BBDD

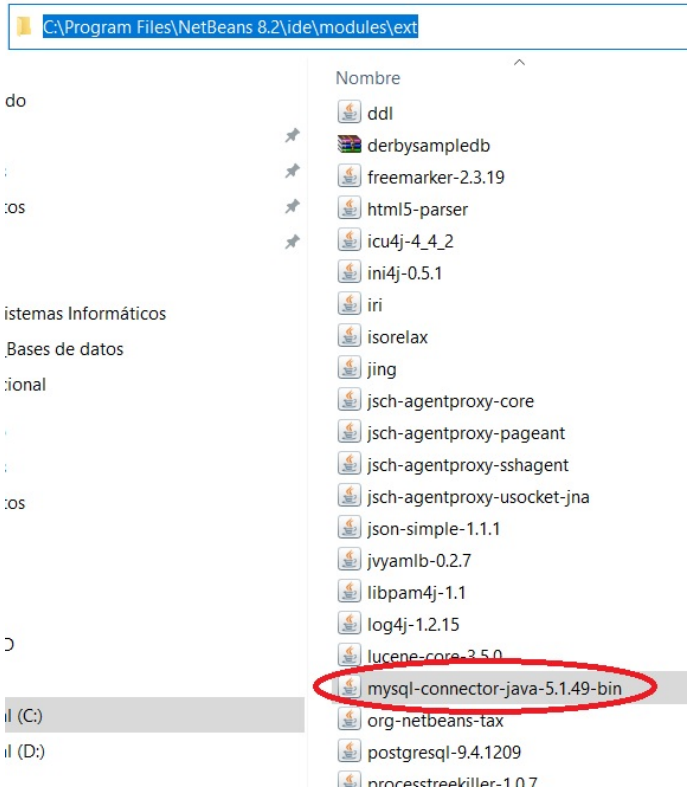


ILERNA Online

Éxito de la conexión

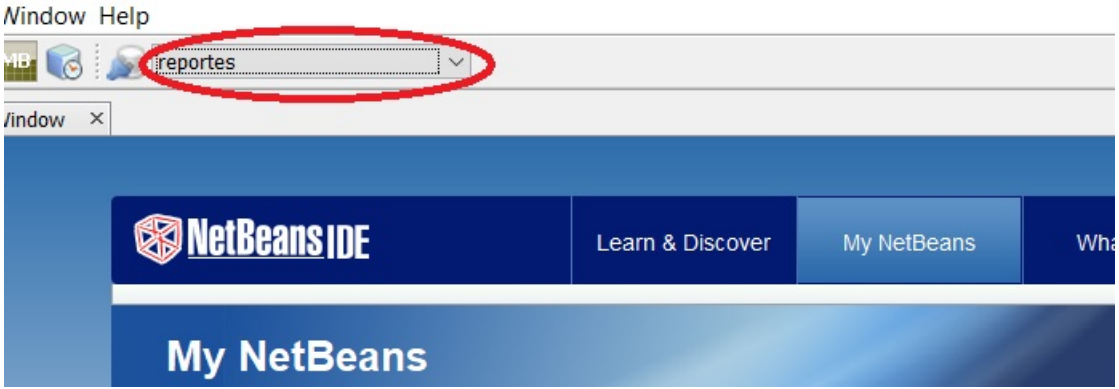
- En la carpeta “ext” de mi ide,
- debemos de tener el conector
- hacia mi base de datos:

Connector/J 5.1.49



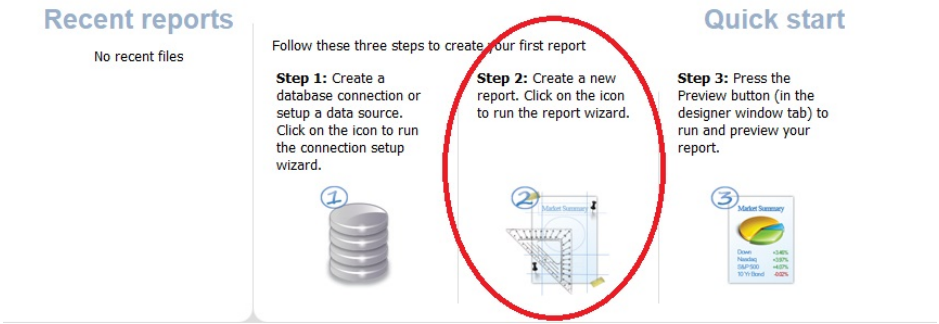
Éxito de la conexión

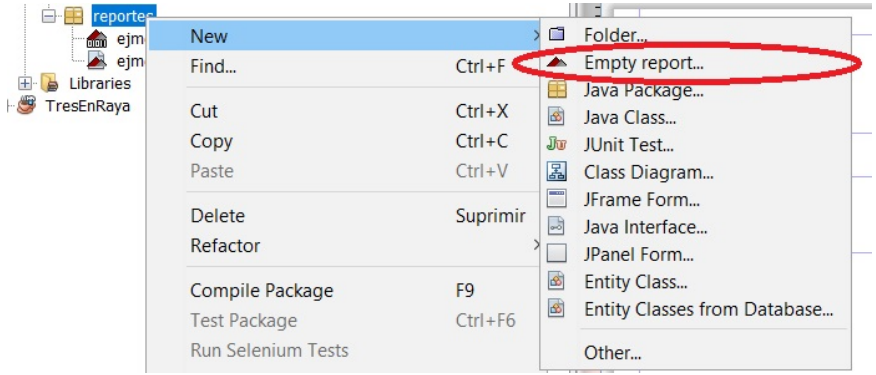
- Una vez creada mi conexión, en el menú superior de arriba, aparecerá creada



ILERNA Online

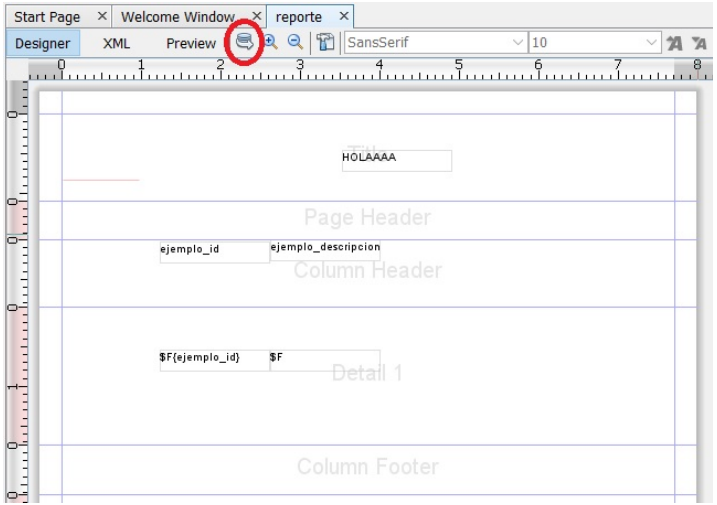
Crear un reporte (o informe)

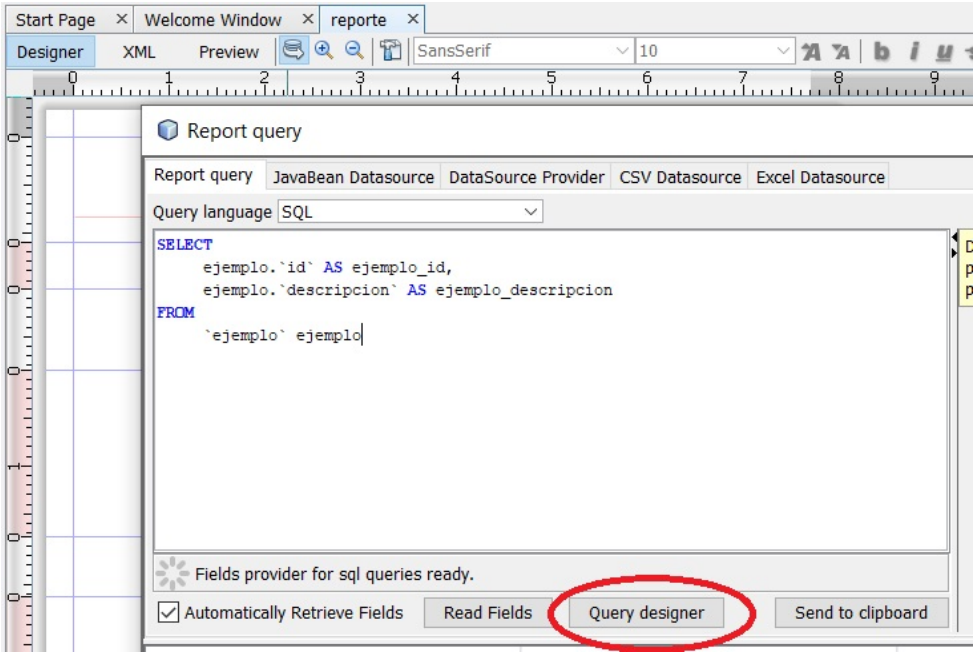




ILERNA Online

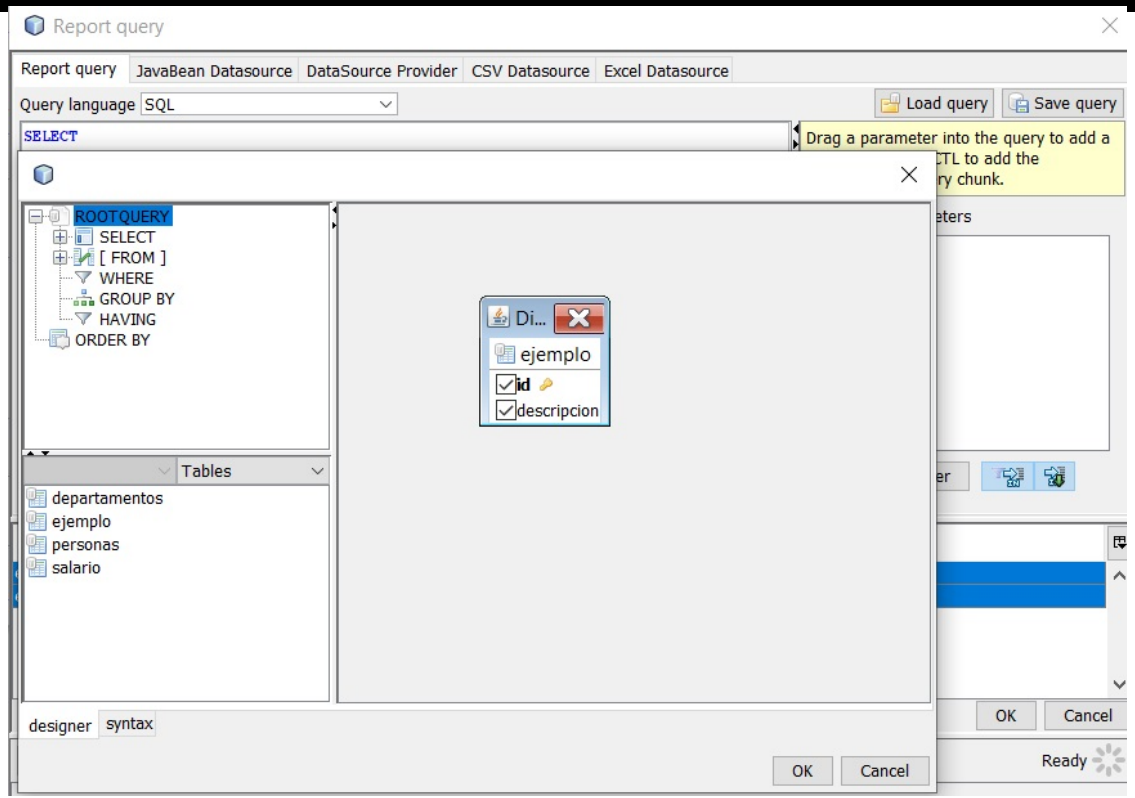
Gestión del reporte: Report query





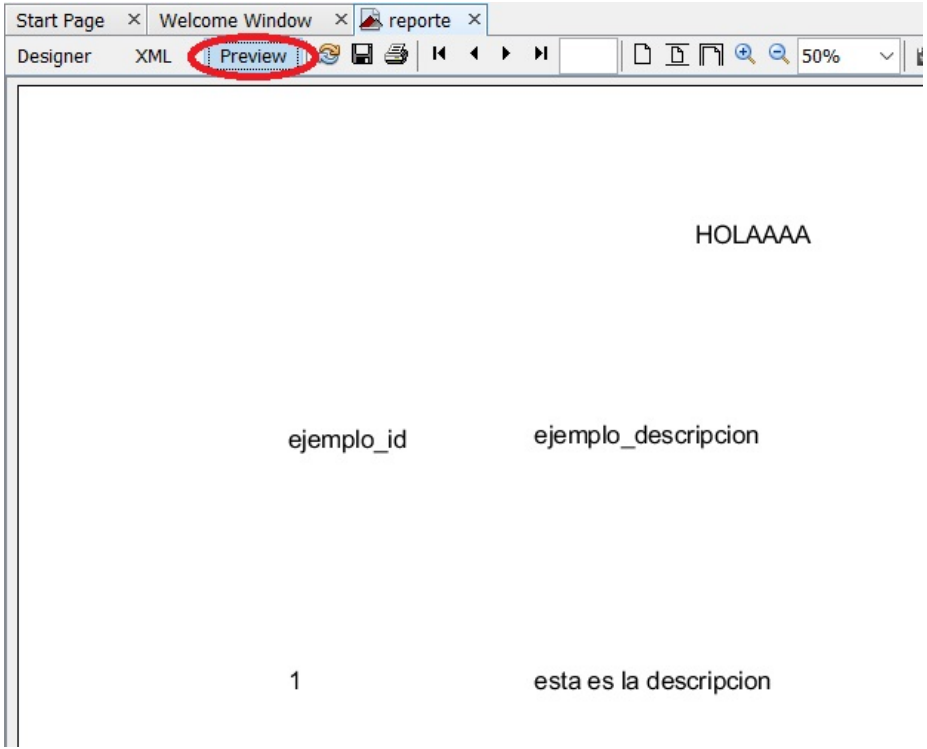
ILERNA Online

Gestión del reporte: Query Designer



ILERNA Online

Gestión del reporte: Preview



ILERNA Online

Gestión del reporte

Es muy importante mostrar a los distintos usuarios un tipo de información organizada en distintos formatos.

Las secciones que ya están definidas son:

- Cabecera
- Cuerpo
- Pie

A partir de estas tres secciones es posible definir la estructura más básica que puede llevar a cabo un informe.

La estructura de un informe puede complicarse a medida que se van añadiendo datos agrupados, gráficos, subinformes, etc.

Ejercicio de reportes

Mostrar un informe con un parámetro que puede ser elegido de un desplegable.


FIN
?

VIDEOCLASE 8: Pruebas

Módulo 07: Desarrollo de interfaces

UF2:Preparación y distribución de aplicaciones

Pruebas

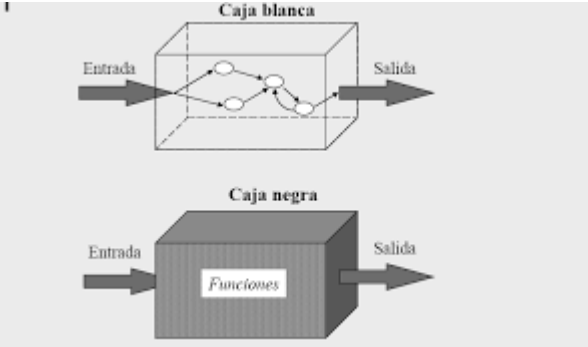
Pruebas

- La realización de diferentes pruebas es un apartado muy importante cuando se lleva a cabo un proceso de desarrollo.
- Mediante estas pruebas, es posible verificar que el producto que se ha diseñado no presenta errores y, sobre todo, que desempeña las distintas tareas para las que ha sido diseñado de forma correcta.
- Una vez que se ha diseñado un producto determinado con un objetivo específico, llega el momento de probar si funciona de forma correcta.

Tipos de pruebas: caja negra

- ***Prueba caja negra***: tienen como objetivo principal especificar las entradas y salidas que produce un determinado producto. En este caso, no es necesario conocer su implementación y se pueden diferenciar los siguientes tipos:

- - Análisis de valores límite.
- - Particiones de equivalencia.



ILERNA Online

Tipos de pruebas: caja negra

Disponemos de un **módulo software** de una aplicación bancaria con siguientes datos de **entrada** con su **tipo**:

- ☐ **Código de área**: número de 3 dígitos que no empieza por 0 ni por 1
- ☐ **Clave identificativa de la operación**: 6 caracteres alfanuméricos
- ☐ **Órdenes posibles**: "cheque", "depósito", "pago factura", "retirada de fondos"

Tipos de pruebas: caja negra

Parámetro de entrada	Regla heurística a aplicar	Clases válidas	Clases inválidas
Código de área	Condición booleana (¿es un número?) + rango valores ([200..999])	**1.** 200<=código<=999	**2.** código < 200 **3.** código > 999 **4.** no es número
Clave identificativa de la operación	Conjunto finito de valores (¿son 6 caracteres?)	**5.** 6 caracteres exactos	**6.** menos de 6 caracteres **7.** más de 6 caracteres
Órdenes posibles	Conjunto de valores admitidos (¿orden válida?)	**8.** "cheque" **9.** "depósito" **10.** "pago factura" **11.** "retirada de fondos"	**12.** no es orden válida

Ejercicio clases de equivalencia

Construcción de una batería de pruebas para detectar posibles errores en la construcción de los identificadores de un hipotético lenguaje de programación. Las reglas que determinan sus construcción sintáctica son:

No debe tener mas de 15 ni menos de 5 caracteres
El juego de caracteres utilizables es: – Letras (Mayúsculas y minúsculas) – Dígitos (0,9) -Guión (-)
Se distinguen las mayúsculas de las minúsculas
El guión no puede estar ni al principio ni al final, pero puede haber varios consecutivos.
Debe contener al menos un carácter alfabético
No puede ser una de las palabras reservadas del lenguaje

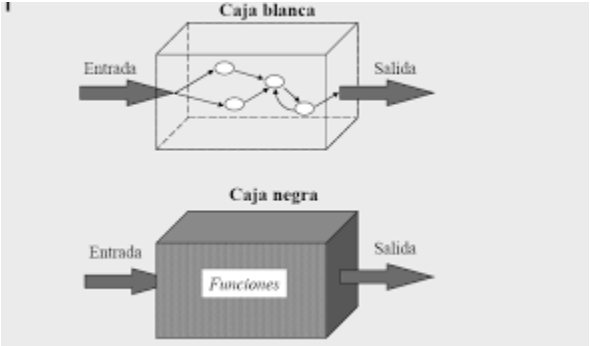
Tipos de pruebas: caja negra

Condiciones de Entrada	Clases de equivalencia válida	Clases de equivalencia no válida
-Entre 5 y 15 caracteres		
-El identificador debe estar formado por letras, dígitos y guión		
-Se diferencia entre letras mayúsculas y minúsculas		
-El guión no puede estar al principio, ni al final -Puede haber varios seguidos en el medio		
-Debe contener al menos un carácter alfabético		
-No se pueden usar palabras reservadas		

ILERNA Online

Tipos de pruebas: caja blanca

- ***Prueba caja blanca***: se tiene en cuenta el valor opuesto a las distintas pruebas que se han realizado, comprobando los elementos internos que son:
- - Cobertura sentencias: decisiones, condiciones…

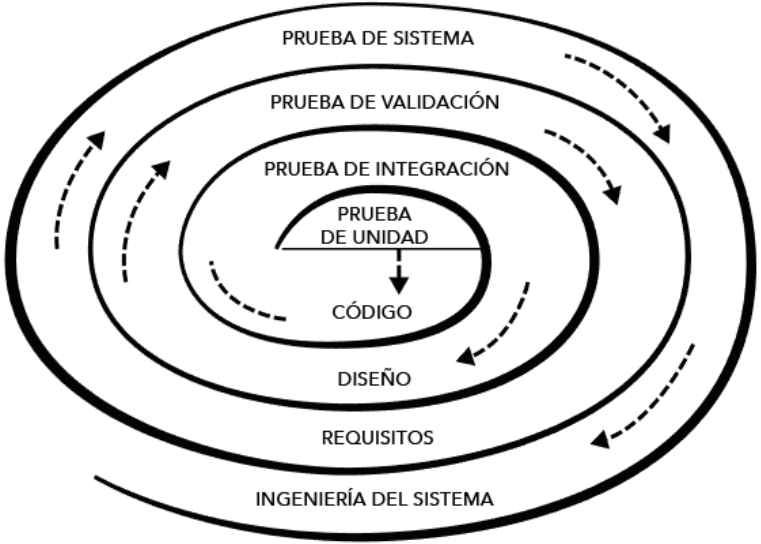


ILERNA Online

Estrategias de pruebas de SW

- En el vértice situaremos la ***prueba de unidad***. Está centrada en la unidad más pequeña, el módulo tal cual está en el código fuente.
- La siguiente es la ***prueba de integración***. Construimos una estructura con los módulos probados en la prueba anterior. El diseño será el foco de atención.
- Seguidamente nos encontramos con la ***prueba de validación***. Es la prueba que realizará el usuario en el entorno final de trabajo.
- La última es la ***prueba del sistema***. Se probará que cada elemento esté construido de forma eficaz y funcional. El software del sistema se prueba como un todo.

Estrategias de pruebas de SW



Pruebas unitarias

•En esta prueba vamos a comprobar cada módulo para eliminar cualquier tipo de error en la interfaz o en la lógica interna. Utiliza ambas técnicas, tanto la prueba de la caja negra como la de la blanca.

•Se realizarán pruebas sobre:

- La interfaz del módulo.
- La estructura de datos locales: comprobación de integridad.
- Las condiciones límite: comprobación de que funciona en los límites establecidos.
- Caminos independientes de la estructura de control, lo que implica asegurar de que se ejecutan las sentencias al menos una vez.
- Todos los caminos de manejo de errores

Pruebas de integración

- Se comprobará la interacción de los distintos módulos del programa.
- Se realizarán de dos formas distintas:
- 1) ***Integración no incremental o big bang***. Comprobación de cada módulo por separado y después se prueba de forma conjunta. Se detectan muchos errores y la corrección es difícil.
- 2) ***Integración incremental***. En este caso el programa se va creando y probando en pequeñas secciones por lo que localizar los fallos es más sencillo.
- En esta integración podemos optar por dos estrategias:
    - a. **Ascendente**. Se comienza con los módulos más bajos del programa.
    - b. **Descendente**. Se empieza por el módulo principal descendiendo por la jerarquía de control.

Pruebas de validación

• **La prueba de validación** cuando el programa funcione de acuerdo con las expectativas expuestas por el cliente y cuando, además, cumpla con lo indicado en el documento de especificación de requisitos del software .

• Se llevarán a cabo pruebas con **la técnica de caja negra**. Se podrán usar estas técnicas:

- **– Prueba Alfa**: realizada por el cliente o usuario en el lugar de desarrollo. Usará el programa bajo la observación del desarrollador que irá registrando los errores.
- **– Prueba Beta**: realizada por los usuarios finales en su lugar de trabajo sin la presencia del desarrollador. En este caso será el usuario el que registre los errores y se los comunique al desarrollador para que realice las modificaciones correspondientes y cree una nueva versión del producto

Pruebas de sistema

•Esta prueba está formada por varias pruebas que tendrán como misión ejercitar en profundidad el software.

•Serán las siguientes:

- **Prueba de recuperación**: se fuerza el fallo del software y que la recuperación se realice correctamente.
- **Prueba de seguridad**: se comprueba que el sistema esté protegido frente a acciones ilegales.
- **Prueba de resistencia (Stress).** se realizan acciones que requieran una gran cantidad de recursos. (identificar cuellos de botella, reducir el riesgo de caídas del sistema, conocer los límites del sistema…)

Junit: Métodos

MÉTODOS	MISIÓN
assertTrue(boolean expresión) **assertTrue(String mensaje, boolean expression)**	Comprueba que la expresión se evalúe *true*. Si no es *true* y se incluye el String, al producirse error se lanzará el *mensaje*.
assertFalse(Boolean expresión) **assertFalse(String mensaje, Boolean expresión)**	Comprueba que la expresión se evalúe *false*. Si no es *false* y se incluye el String, al producirse error se lanzará el *mensaje*.
assertEquals(valorEsperado, valorReal), **assertEquals(String mensaje, valorEsperado, valorReal)**	Comprueba que el *valorEsperado* sea igual al *valorReal*. Si no son iguales y se incluye el String, entonces se lanzará el *mensaje*. *ValorEsperado* y *ValorReal* pueden ser de diferentes tipos.

Ejemplo prueba unitaria

```
 * @author mouse
 */
public class MiClase {

    public int numero_mayor(int a, int b, int c) {
        if (a > b && a > c) {
            return a;
        } else if (c > b) {
            return c;
        } else {
            return b;
        }
    }

}
```

```
1   @Test
2   public void testNumero_mayor() {
3       System.out.println("numero_mayor");
4       int a = 0;
5       int b = 0;
6       int c = 0;
7       MiClase instance = new MiClase();
8       int expResult = 0;
9       int result = instance.numero_mayor(a, b, c);
10      assertEquals(expResult, result);
```

Ejercicio

Crea una prueba unitaria (en Netbeans con JUnit) que evalúe un método recursivo que saber el palíndromo de una palabra.

PROXIMA VIDEOCLASE:

DESARROLLO DE APLICACIONES MULTIPLATAFORMA

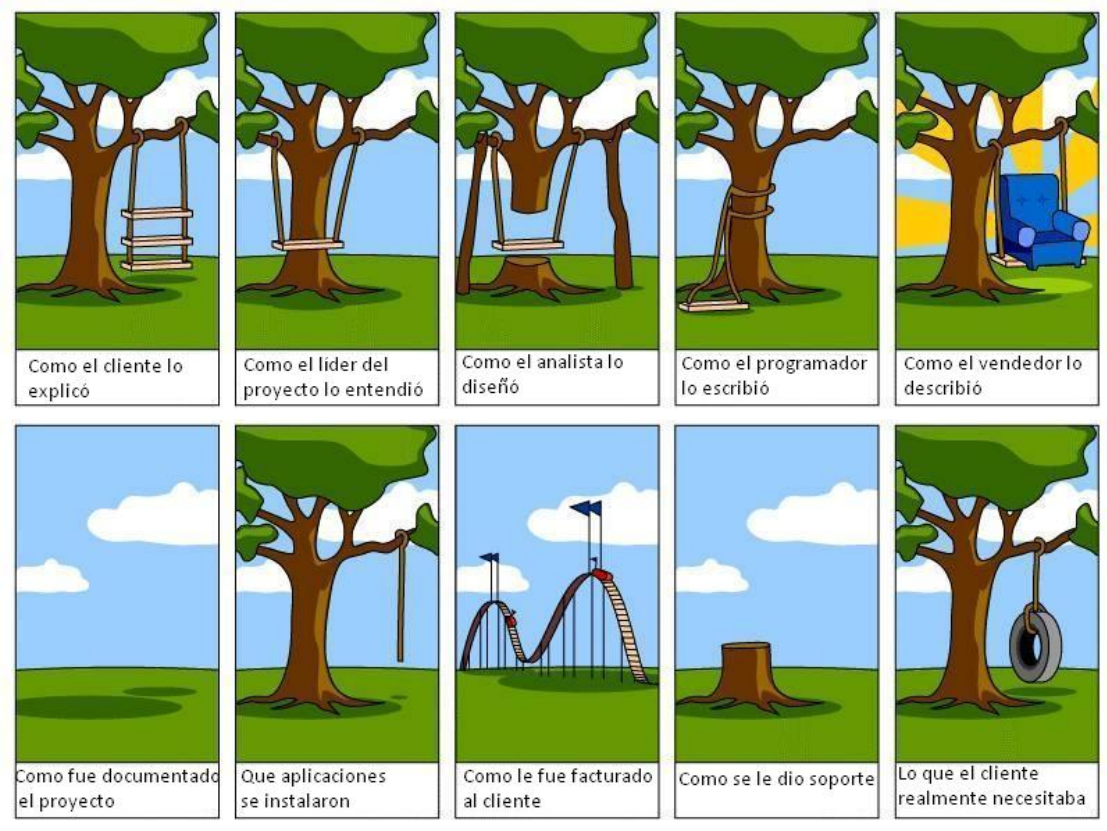
ILERNA Online

M07: DESARROLLO DE INTERFACES

UF2: Preparación y distribución de aplicaciones

Tema 2. Documentación de aplicaciones

> Visión de un proyecto



Se refiere a la producción de un documento

Que puede ser revisado, evaluado y aprobado.

Software Requirements Specification SRS

Este documento es la base para el acuerdo entre clientes y desarrolladores o suministradores sobre lo que hará el producto.

https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf

> Archivos de ayuda

La documentación de las distintas aplicaciones es una tarea muy importante en cualquier proyecto. Es preciso señalar que, si se cuenta con la documentación adecuada y detallada, es posible acceder a parte del código necesario en un determinado momento para después ser reutilizado en otro proyecto distinto.

Esta tarea de documentación por parte de los desarrolladores es bastante tediosa, por lo que no se le dedica demasiado tiempo. Este es uno de los motivos por el que se cree conveniente automatizar el desarrollo de la distinta documentación de ayuda.

Ayuda contextual

Ofrece al usuario, a través de diferentes elementos visuales o de texto, un acceso más rápido y directo a la función que desee realizar.

- **ToolTips**: son una serie de elementos que aparecen en la interfaz cuando el ratón se posiciona sobre algún elemento determinado. El objetivo de estos elementos es brindar al usuario alguna ayuda sobre ese elemento en cuestión.

- **Ficheros ayuda**: son aquellos que permiten ofrecer una ayuda al usuario al pulsar la tecla F1. Gracias a estos ficheros de ayuda, se pretende facilitar la tarea del usuario sobre una función específica.

Manual de instalación

- **Reducida**: se lleva a cabo una instalación rápida con los valores seleccionados por defecto sin entrar en muchos más detalles.

- **Detallada**: esta versión explica de forma más detallada todos los aspectos que intervienen en el proceso de instalación.

Documentación de configuración

- Configuración **normal**.

- Configuración **avanzada**: es muy parecida a la anterior, aunque, en este caso, se detallan de forma más exhaustiva distintos aspectos más complejos que necesitan que los usuarios tengan una mayor experiencia y conocimiento (administradores).

Manual de usuario

Al igual que en otros modelos de manuales, en el manual de usuario también se diferencian dos versiones:

- **Rápida**.

- **Detallada**: esta versión amplía las diferentes opciones a la hora de realizar las diversas posibilidades de aplicación.

Guía de usuario

Igual que manual de usuario pero cambia la extensión y la cantidad de información disponible

Guía rápida

En esta guía rápida se dispone de un resumen con todos los aspectos más importantes que se han desarrollado en la documentación previa. Se detalla de forma breve y muy sintetizada.

> Herramientas generación de ayuda

A la hora de desarrollar un código, es posible ayudarse de comentarios que se detallan sobre este mismo código para ofrecer documentación sobre los temas tratados. Se conoce como meta-información y se escribe precedido de los caracteres /// delante del código.

GhostDoc corresponde a la extensión de Visual Studio que permite generar, de forma automática, documentación XML que contiene toda la información sobre los comentarios de un código determinado, junto con sus principales funciones.

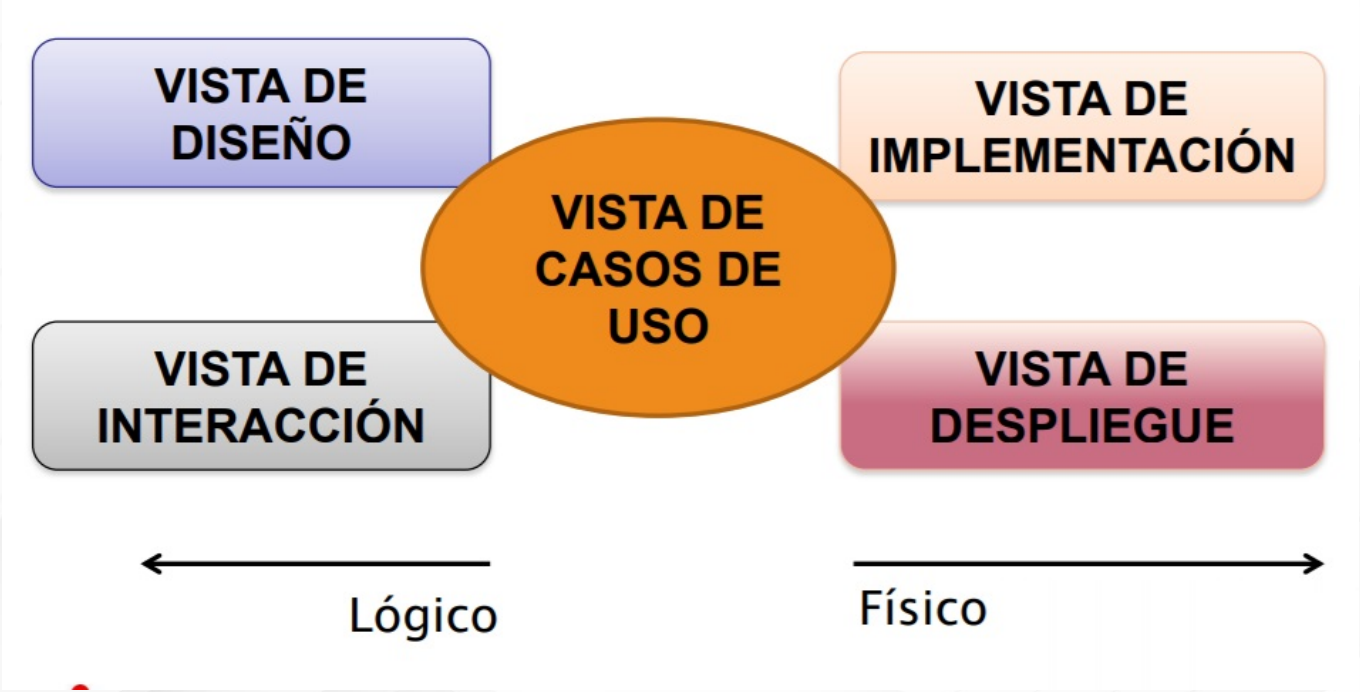
```
///<summary>
...
///</summary>
```

> Javadoc

Etiqueta	*Descripción*
@author	Indica el nombre de quien desarrollo el componente.
@deprecated	Indica que algún método o clase u otro componente está obsoleto.
@param	Indica un parámetro de un método, se tiene que usar para todos los parámetros del método.
@return	Indica el valor de retorno de un método
@see	Indica que el componente puede hacer referencia a otro. Ejemplo: # método(); clase#método();
@throw	Indica que excepción lanza el método.
@version	Indica la versión actual del componente

Tema 2. Distribución de aplicaciones

> 4+1 Vistas

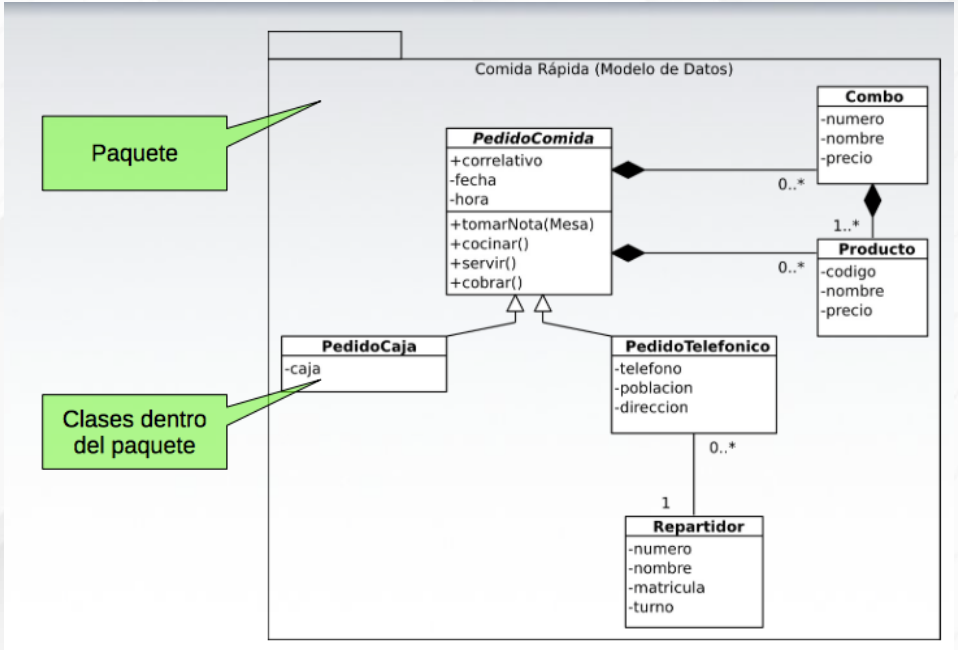

VISTA DE DISEÑO
VISTA DE IMPLEMENTACIÓN
VISTA DE CASOS DE USO
VISTA DE INTERACCIÓN
VISTA DE DESPLIEGUE
Lógico
Físico

¿Diagramas de diseño?

¿Diagramas de interacción?

> Diagrama de paquetes

- Organizan elementos de un modelado en diferentes grupos (incluidos los diagramas)
- También organizan modelos grandes, agrupar elementos relacionados o crear namespaces.



> Dependencias

Para que un elemento de un paquete tenga acceso (permisos) a otro elemento de otro paquete se debe crear una relación entre ambos paquetes.

Dependencia: algún elemento de un paquete depende de los elementos de otro paquete

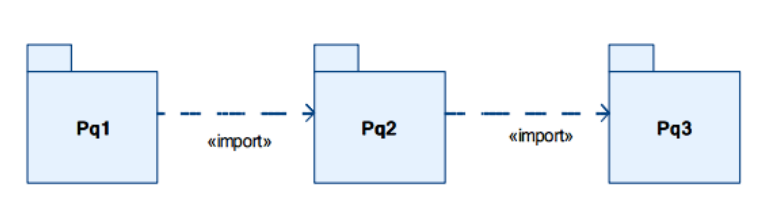
Importación: <<Import>>

La importación de paquetes añade los elementos públicos del paquete destino al espacio de nombres público del origen

> Dependencias

Las relaciones de permiso no son simétricas ni transitivas.

- Pq1 puede acceder al contenido público de Pq2 y Pq2 al de Pq3 pero:
- Pq2 no puede acceder a ningún contenido de Pq1.
- Pq1 no puede acceder a ningún contenido de Pq3.



> Paquetes

Existen distintos paquetes que constan de componentes individuales, mientras que otros están formados por un conjunto de aplicaciones relacionadas entre ellas.
Una vez que se crea un paquete ya es posible distribuirlo a otros usuarios y organizaciones.

Componentes

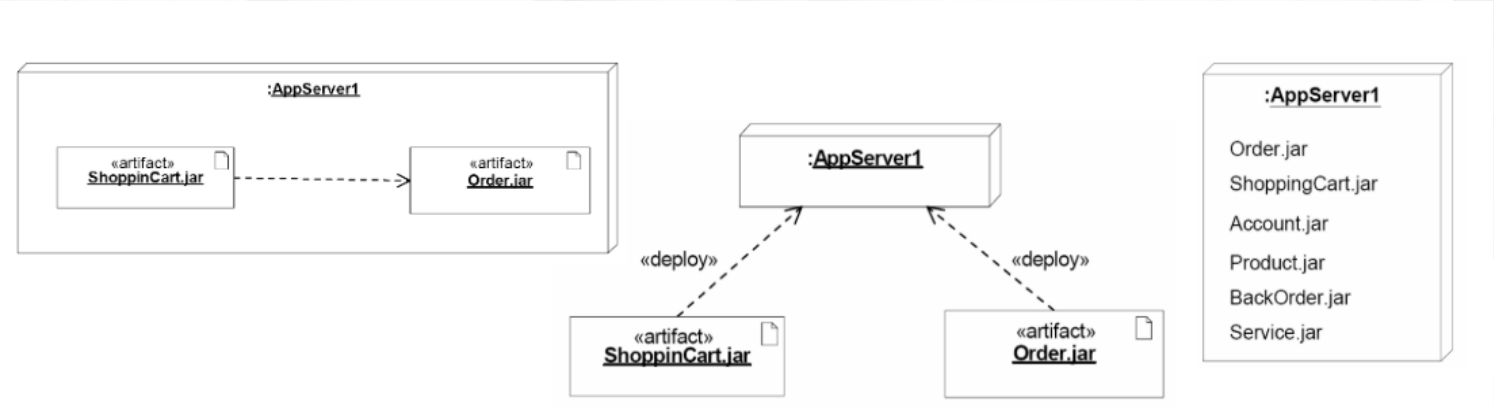
Los componentes son el conjunto de elementos que forman un paquete, teniendo en cuenta que un elemento es un objeto personalizado. Existe la posibilidad de unificar distintos componentes de un paquete para conseguir crear determinadas funciones.
En el caso de los paquetes sin gestionar, los componentes que lo forman no se pueden actualizar. Sin embargo, en los paquetes gestionados, es posible actualizar solo algunos de sus componentes, pero no todos.

> Despliegue

- Los diagramas de despliegue muestran un conjunto de nodos, sus relaciones y los artefactos que residen en ellos.
- Los diagramas de despliegue muestran:
  - El hardware sobre el que se ejecutará el sistema.
  - La configuración de los nodos que participan en la ejecución del software.
  - Los artefactos que residen en cada nodo.
- Describen la vista de despliegue estática.

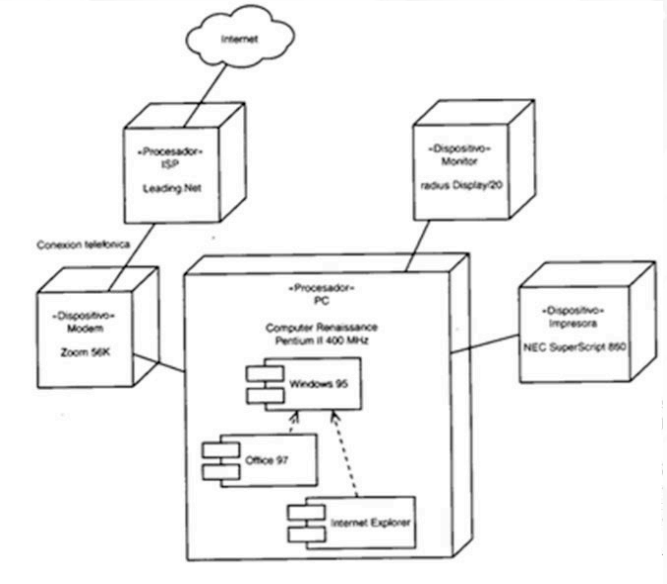
> Despliegue

- El software que se ejecuta en los nodos se modela con artefactos, que son ficheros físicos que el software usa o ejecuta
- Implementación física de un conjunto de elementos lógicos tales como clases y componentes.
- Los artefactos residen en nodos.



> Ejemplo de Despliegue

- Para modelar mi equipo he incluido al procesador y los dispositivos a la vez que he modelado mi conexión telefónica con mi proveedor de servicios de Internet y su conexión.




FIN