

TECNICATURA
UNIVERSITARIA
EN PROGRAMACIÓN
UTN-FRC



UTN 
Facultad Regional Córdoba

TECNICATURA UNIVERSITARIA EN PROGRAMACIÓN

LABORATORIO DE COMPUTACIÓN III

Unidad Temática 4: Interfaces Gráficas de Usuario

Material de Estudio

2^{do} Año - 3^{er} Cuatrimestre

2020



V.0.1

Índice

1. INTERFACES GRÁFICAS DE USUARIO	2
Introducción	2
Librerías	2
Creación de cuadros de diálogo	3
Presentación de texto	3
Ingreso de datos	4
Cuadro de mensaje con botoneras predefinidas	5
2. CREACIÓN DE VENTANAS	6
3. DISEÑADOR DE VENTANAS	9
Uso del diseñador	9
Creación de una ventana	9
4. AGREGADO DE CONTROLES	11
5. MANIPULACIÓN DE EVENTOS	13
Manipulación de atributos por código	15
BIBLIOGRAFÍA	26

1. INTERFACES GRÁFICAS DE USUARIO

Introducción

Cuando se desea programar aplicaciones con interfaces gráficas de usuario con ventanas, Java ofrece un amplio abanico de posibilidades y un gran conjunto de herramientas que permiten crear tales interfaces con un adecuado esfuerzo y en poco tiempo.

Dado que Java es multiplataforma, los programas que presenten ventanas también van a poder ejecutarse en forma idéntica y con una presentación visual muy similar en todos los sistemas operativos, principalmente Windows, Linux y macOS.

A lo largo de la vida del lenguaje, se ofrecieron tres librerías diferentes para la creación de aplicaciones con ventanas. Todas ellas están disponibles para ser utilizadas pero en esta asignatura se estudiará la más utilizada y que por lo tanto posee mayor documentación disponible.

Librerías

- **AWT:** Es la primera librería que ofreció Java para la creación de ventanas. Muchos de los conceptos que incluyó son utilizados por las tres librerías disponibles. Tiene una desventaja principal que reside en el hecho de que cada objeto se dibuja en la pantalla con su propio diseño gráfico, sin utilizar las funcionalidades básicas del sistema operativo donde se ejecute el programa. Esto lleva a que los programas sean más lentos para presentar cada ventana y además la interfaz no respeta los estilos visuales del escritorio del sistema operativo.
- **Swing:** Es la segunda librería disponible en Java y fue presentada algunos años después que AWT. Una gran cantidad de las clases disponibles son heredadas de clases similares de la librería AWT. O sea que cuando se use Swing, también se está usando porciones de la librería anterior. Swing agrega y mejora muchas funcionalidades con respecto a su antecesora. Por un lado, utiliza al sistema operativo para poder dibujar los componentes visuales, es decir, que si se quiere mostrar una ventana, en lugar de dibujar con líneas, curvas o letras propias, simplemente le solicita al escritorio del sistema operativo que lo haga. De esta maneja no son más lentas las ventanas de Java que las de programas desarrollados en otros lenguajes. Además que no se distinguen especialmente porque se respetan los estilos visuales configurados en el escritorio. Por otro lado, entre otras ventajas, en Swing se incluyeron muchos controles nuevos de interacción con el usuario, se agregaron gestores de distribución física del contenido de las ventanas y se

permite cambiar el estilo de las ventanas aplicando un concepto similar al de las pieles o temas que ofrecen muchos programas. Esta biblioteca es la que se va a analizar en la materia.

- **JavaFX:** Es la tercera librería de interfaces gráficas que se incluyó en Java. No tiene código heredado de las anteriores y fue planteada para poder programar software con ventanas que se ejecute en una computadora y también para crear mini aplicaciones que corran dentro de un navegador, con un concepto similar al de Adobe Flash o Silverlight. A pesar de ofrecer unas capacidades gráficas muy superiores a AWT y Swing, es considerablemente más difícil de aprender y no hay tanta documentación disponible. Tras la desaparición de Flash en favor de HTML5, el uso de JavaFX también decayó rápidamente al punto que desde la versión 11 de Java ya no es parte de la biblioteca estándar y para ser utilizada debe descargarse en forma independiente.

Creación de cuadros de diálogo

El mecanismo más simple para presentar ventanas al usuario es mediante el uso de unos cuadros de diálogo con diseño predefinido denominados cuadros de mensajes.

Para presentar un cuadro de mensaje se utiliza la clase JOptionPane la cual ofrece diversos métodos, cada uno de ellos para los diferentes diseños disponibles.

Presentación de texto

En el caso más básico se puede utilizar el método showMessageDialog que muestra al usuario una ventana con un texto y un botón OK o Aceptar.

```
JOptionPane.showMessageDialog(contenedora, texto);
```

Por ejemplo, la línea

```
JOptionPane.showMessageDialog(null, "Hola, soy una ventana Java");
```

presenta la siguiente ventana:

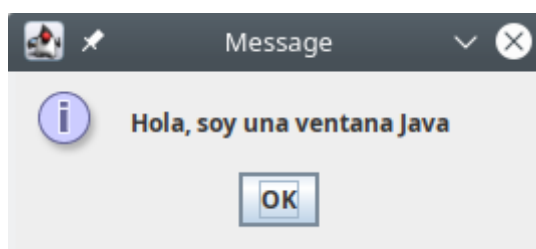


Imagen 1: Extraído de JAVA

Para poder indicar el título de la ventana y el tipo de icono presentado a la izquierda se debe utilizar la siguiente sobrecarga:

```
JOptionPane.showMessageDialog(contenedora, texto, titulo, tipo);
```

En donde el tipo de mensaje debe ser una de las siguientes constantes:

```
JOptionPane.ERROR_MESSAGE  
JOptionPane.INFORMATION_MESSAGE  
JOptionPane.WARNING_MESSAGE,  
JOptionPane.QUESTION_MESSAGE  
JOptionPane.PLAIN_MESSAGE
```

Por ejemplo:

```
JOptionPane.showMessageDialog(null,"El apellido es obligatorio","Error de carga",JOptionPane.ERROR_MESSAGE);
```

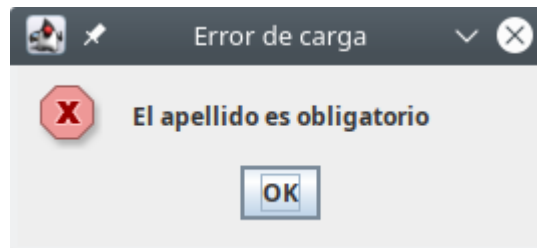


Imagen 2: Extraído de JAVA

Ingreso de datos

Por otro lado, también se puede indicar que el cuadro de mensaje posea cuadro de ingreso de texto para que el usuario pueda proveer un dato.

Para ello se debe invocar al método `JOptionPane.showInputDialog()` que posee los mismos parámetros que `showMessageDialog`. De esta manera para ingresar un dato de tipo `String` se puede invocar a:

```
String apellido = JOptionPane.showInputDialog(null,"Ingrese su apellido","Nueva persona", JOptionPane.QUESTION_MESSAGE);
```

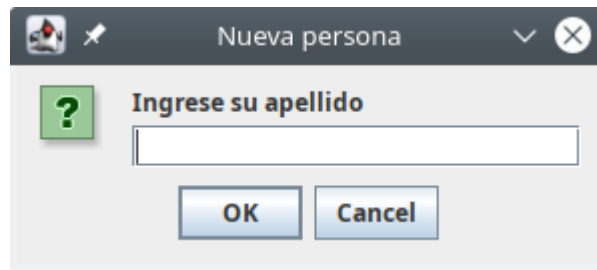


Imagen 3: Extraído de JAVA

En este caso, el texto que el usuario ingrese es devuelto por el método para que sea asignado en una variable. Mientras que si el usuario presiona el botón Cancelar o el de cierre de la ventana se recibe un dato null.

Si se necesita cargar datos de otros tipos, tales como números, se necesita almacenar el retorno en una variable de tipo String y luego realizar una conversión:

```
String texto = JOptionPane.showInputDialog(null,"Ingrese su edad","Nueva persona",
JOptionPane.QUESTION_MESSAGE);
int edad = Integer.parseInt(texto);
```

Cuadro de mensaje con botonerías predefinidas

Además del ingreso de texto, la clase `JOptionPane` posee un método que le presenta al usuario un conjunto de botones para que seleccione uno. Para poder dibujar este tipo de cuadros de mensaje se utiliza el siguiente método:

```
JOptionPane.showMessageDialog(contenedora, texto, titulo, botones, tipo);
```

Los posibles conjuntos de botones son los siguientes:

```
JOptionPane.YES_NO_OPTION
JOptionPane.YES_NO_CANCEL_OPTION
JOptionPane.OK_CANCEL_OPTION
```

El resultado del método es el número de botón presionado, contando desde 0 en adelante según el orden en que se presentan, desde izquierda a derecha. De esta manera, en el caso de la opción `YES_NO_CANCEL`, si el usuario selecciona el botón SI, se retorna un 0, y si presiona `CANCEL` se retorna un 2.

Ejemplo:

```
JOptionPane.showConfirmDialog(null,"¿Desea grabar antes de salir?",  
"Confirmación",  
JOptionPane.YES_NO_CANCEL_OPTION,  
JOptionPane.QUESTION_MESSAGE);
```

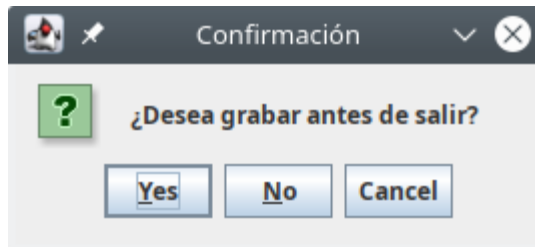


Imagen 4: Extraído de JAVA

2. CREACIÓN DE VENTANAS

Indudablemente no alcanza con los diálogos predefinidos de JOptionPane para desarrollar un programa completo, normalmente se requiere poder presentar al usuario ventanas con muchos controles de interacción y con un diseño propio.

Para crear ventanas propias la librería Swing ofrece una gran cantidad de clases que lo permiten sin un esfuerzo considerable.

Para poder presentar al usuario una ventana solamente se requiere crear una instancia de la clase JFrame y modificar al menos las propiedades size y visible. La propiedad size indica el tamaño inicial en píxeles y al asignar el valor true a la propiedad visible, el programa muestra la ventana.

A continuación, se presenta el programa mínimo necesario para esto:

```
public static void main(String[] args) {  
  
    JFrame ventana = new JFrame("Mi primera ventana");  
    ventana.setSize(300,200);  
    ventana.setVisible(true);  
  
}
```

Al ejecutar se presenta la siguiente ventana:

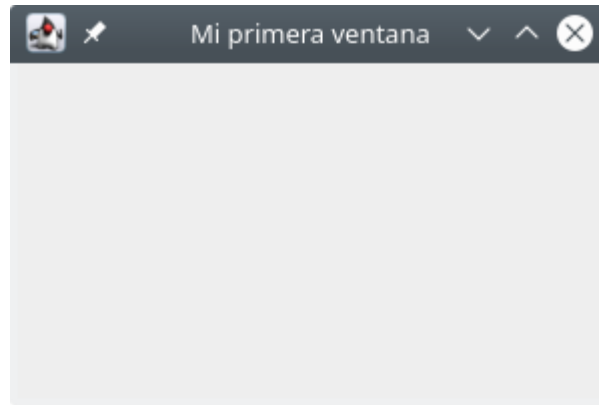


Imagen 5: Extraído de JAVA

La ventana creada de esta manera no tiene ningún contenido, para agregarle elementos visuales debe seguirse un patrón similar, es decir, crear instancias de las clases que representan a cada tipo de control, asignarle propiedades y finalmente agregarlas a la ventana.

Por ejemplo, para agregar a la ventana un botón se necesita crear una instancia de la clase `JButton` y asignarle las propiedades `size` y `location`, esta última necesaria para ubicar el botón en una coordenada en particular dentro de la ventana.

Finalmente se necesita agregar el nuevo botón a la ventana con el método `add()` de la clase `JFrame` antes de mostrar la ventana. El programa anterior con el agregado del botón puede quedar como se presenta a continuación:

```
public static void main(String[] args) {  
    JFrame ventana = new JFrame("Mi primera ventana");  
    ventana.setSize(300,200);  
    ventana.setLayout(null);  
  
    JButton boton = new JButton("Aceptar");  
    boton.setLocation(80,50);  
    boton.setSize(100,40);  
    ventana.add(boton);  
    ventana.setVisible(true);  
}
```

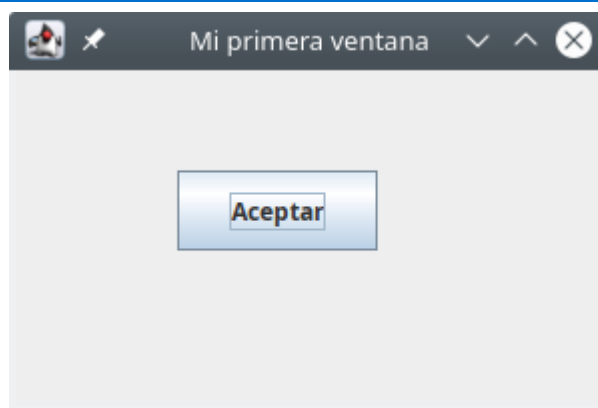



Imagen 6: Extraído de JAVA

Cada uno de los diferentes tipos de controles que se pueden agregar a una ventana están representados por una clase del paquete `javax.swing`. De esta manera, para dibujar un cuadro de texto se dispone de la clase `JTextField`, para una casilla de selección simple la clase `JCheckBox`, para una casilla de selección mutuamente excluyente `JRadioButton`, etc.

Con el mismo criterio que en el ejemplo del botón, asignando diversas propiedades de cada clase y finalmente agregando las instancias a la ventana con el método `add()` es totalmente factible crear ventanas con el contenido y diseño que se requiera.

Sin embargo, esta tarea resulta muy laboriosa, propensa a errores y lenta. A pesar de ser totalmente factible, excepto para casos excesivamente simples, cualquier ventana requeriría centenares o incluso miles de líneas de código fuente en la tarea de crear y agregarle todos los controles y asignarle sus propiedades.

Por otro lado, el cálculo de la ubicación y tamaño de cada control haría necesario llevar adelante un mecanismo muy largo de prueba y error, ejecutando múltiples veces el programa únicamente para verificar que cada control se muestre en el lugar correcto dentro de la ventana.

Finalmente, la asignación de los controladores de eventos requiere elementos del lenguaje relativamente difíciles.

Por todos estos motivos, siempre va a ser recomendable utilizar diseñadores gráficos de las ventanas, que simplifiquen el diseño de las mismas sin obligar al programador a redactar todo el código necesario. Todos los entornos de desarrollo poseen algún mecanismo para diseñar las ventanas, en la siguiente sección se revisará el uso del diseñador de NetBeans.

3. DISEÑADOR DE VENTANAS

Uso del diseñador

Para crear ventanas mediante el diseñador de NetBeans se debe comprender el código fuente que el mismo genera. Por cada ventana diferente que se vaya a diseñar NetBeans crea una clase nueva que hereda de JFrame. Para poder mostrar esas ventanas, el programa necesita únicamente crear una instancia de la clase correspondiente a la ventana que se necesita y asignarle la propiedad visible en true.

En la clase que representa a una ventana, los controles son atributos la misma, y en el constructor se asignan todas las propiedades de los mismos y se los agrega a la ventana.

El diseñador genera todo el código fuente necesario (el cual a veces es muy largo) y deja al programador la responsabilidad de agregarle la funcionalidad. Para ello el programador modifica todo el código según su necesidad, creando nuevos atributos o métodos con su criterio. Únicamente se debe dejar intacto un método denominado initComponents; adecuadamente el editor impide modificaciones en el mismo, pero es muy importante no modificarlo desde afuera del IDE.

Creación de una ventana

Para crear una ventana nueva se debe hacer click derecho en el proyecto o en un paquete de clases y seleccionar la opción de "Formulario JFrame". En este punto es muy importante no seleccionar "Formulario JPanel" ya que dicha opción genera ventanas similares pero que requieren código adicional para poder ser mostradas.

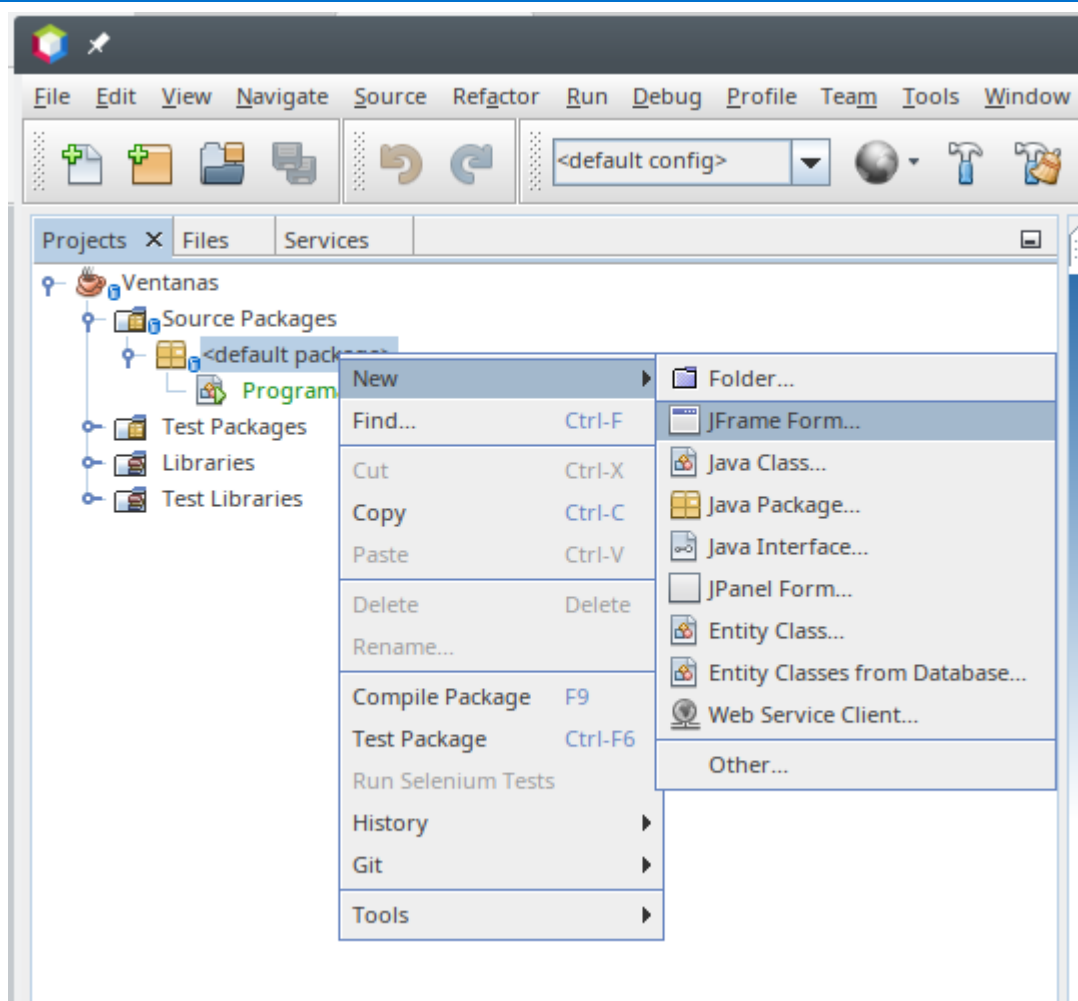


Imagen 7: Extraído de JAVA

A continuación se debe ingresar el nombre de la ventana y al aceptar se crea la nueva clase pero en lugar de abrirse el editor de texto, se presenta el diseñador de formularios.

El mismo muestra una región rectangular vacía que representa a la ventana y por encima dos solapas que permite alternar entre la vista de diseño y la vista de código, es decir, el editor de texto donde se escribe el código fuente.

A la clase de cada ventana se le agrega un método main cuya funcionalidad es exclusivamente la de abrir la ventana a la que pertenece y es útil para poder verla en funcionamiento durante la programación. Cuando el programa se encuentra terminado es una práctica recomendable eliminar todos esos métodos main dejando el principal.

4. AGREGADO DE CONTROLES

A la derecha del diseñador se presenta la paleta de controles, desde la cual se agrega arrastrando a la ventana cada uno de los elementos visuales que se necesiten:

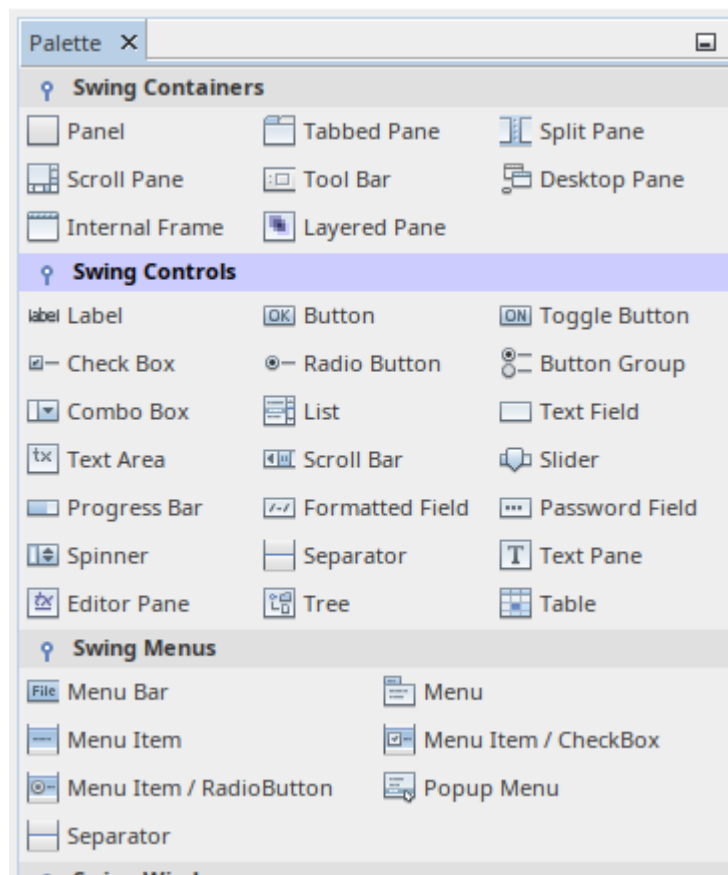


Imagen 8: Extraído de JAVA

En esta paleta se pueden identificar todos los controles disponibles para agregar a las ventanas, para ello se debe arrastrar desde la paleta el control deseado hasta la ubicación necesaria en la ventana o hacer doble click sobre el mismo.

Cada control que se agrega necesita un nombre adecuado para que se lo pueda acceder desde el código. El IDE solamente le asigna un nombre formado por el nombre de la clase y un número correlativo. Este nombre no resulta adecuado para casi ningún escenario por lo tanto debe cambiarse inmediatamente después de ser agregado. El nombre del control se lo modifica haciendo click derecho en el diseñador sobre el mismo, y seleccionando la opción "Cambiar nombre de variable":

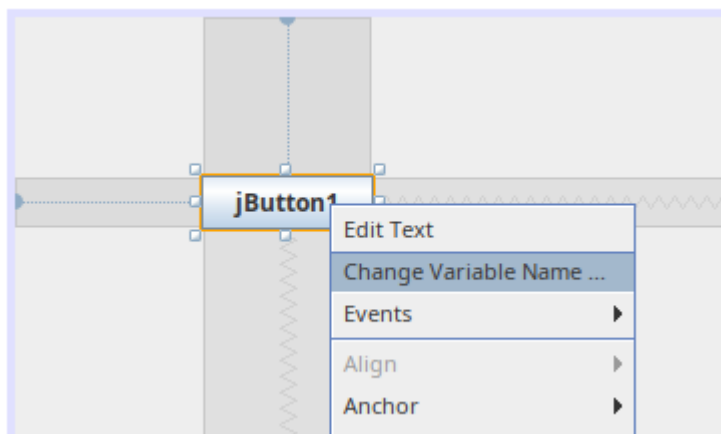


Imagen 9: Extraído de JAVA

Los nombres de los controles deben cumplir con las reglas correspondientes a los atributos de una clase. Sin embargo es una práctica muy habitual tanto en Java como en todos los lenguajes que ofrecen interfaces gráficas aplicar una nomenclatura especial a los controles visuales.

Esta nomenclatura consiste en anteceder al nombre del control un prefijo de tres letras que indique el tipo de control. De esta manera los cuadros de texto usan el prefijo **txt**, las etiquetas el prefijo **lbl**, etc. La tabla a continuación presenta los prefijos de los tipos de controles más habituales:









Tipo de control	Prefijo	Uso
Cuadro de texto  Text Field	txt	Ingreso de texto de una sola línea
Etiqueta  Label	lbl	Texto sin interacción
Botón  Button	btn	Botón simple
Casilla de selección  Check Box	chk	Selección simple por sí o no
Selección excluyente  Radio Button	rad	Casilla de selección mutuamente excluyente
Lista simple  List	lst	Lista de items para seleccionar
Lista desplegable  Combo Box	cbo	Lista desplegable de items
Tabla  Table	tbl	Grilla o tabla para consulta de datos

Tabla 1: Elaboración propia

Luego de agregar los controles se pueden modificar sus propiedades, especialmente aquellas que afectan la presentación visual de los mismos (texto, tipo de letra, colores, etc.) desde la solapa de propiedades del diseñador la cual se encuentra debajo de la paleta:

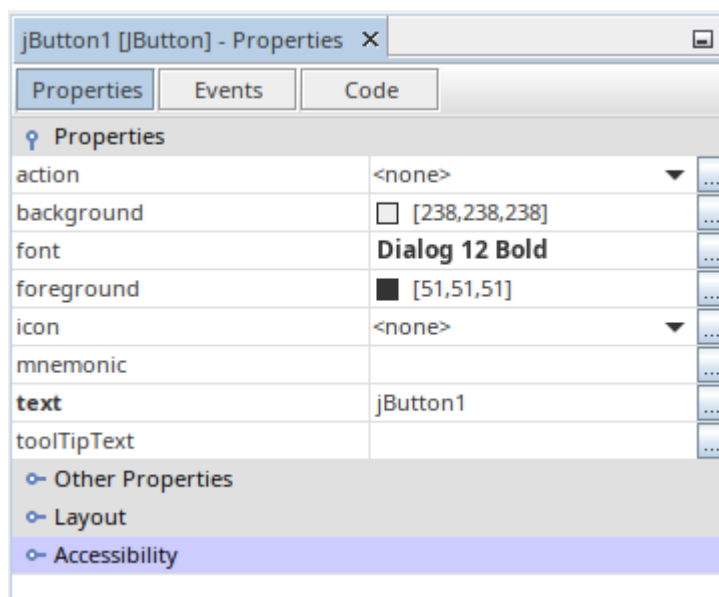


Imagen 10: Extraído de Java

5. MANIPULACIÓN DE EVENTOS

Cuando se presenta un programa con interfaz gráfica el usuario es quién decide el orden de ejecución a través de la interacción con los controles. En estos escenarios, la programación consiste principalmente en asignar porciones de código que se deben ejecutar cuando el usuario efectivamente realice ciertas acciones, tales como presionar un botón, marcar un cuadro de selección, cerrar una ventana, elegir una opción de menú, etc.

Las acciones que el usuario aplica sobre los controles se denominan eventos. Ocasionalmente otros actores generan eventos, por ejemplo el sistema operativo cuando va a apagar la computadora le genera al programa un evento de cierre de todas las ventanas que tenga abiertas. Sin embargo, estas situaciones son particularmente infrecuentes y prácticamente el total de los eventos son generados por el usuario.

Por cada evento que el programa necesite atender se debe programar un método denominado manejador de evento y se debe asociar este manejador con el evento concreto que efectivamente debe manipular. Cuando el usuario realice la acción correspondiente, en forma automática se ejecuta el método manejador.

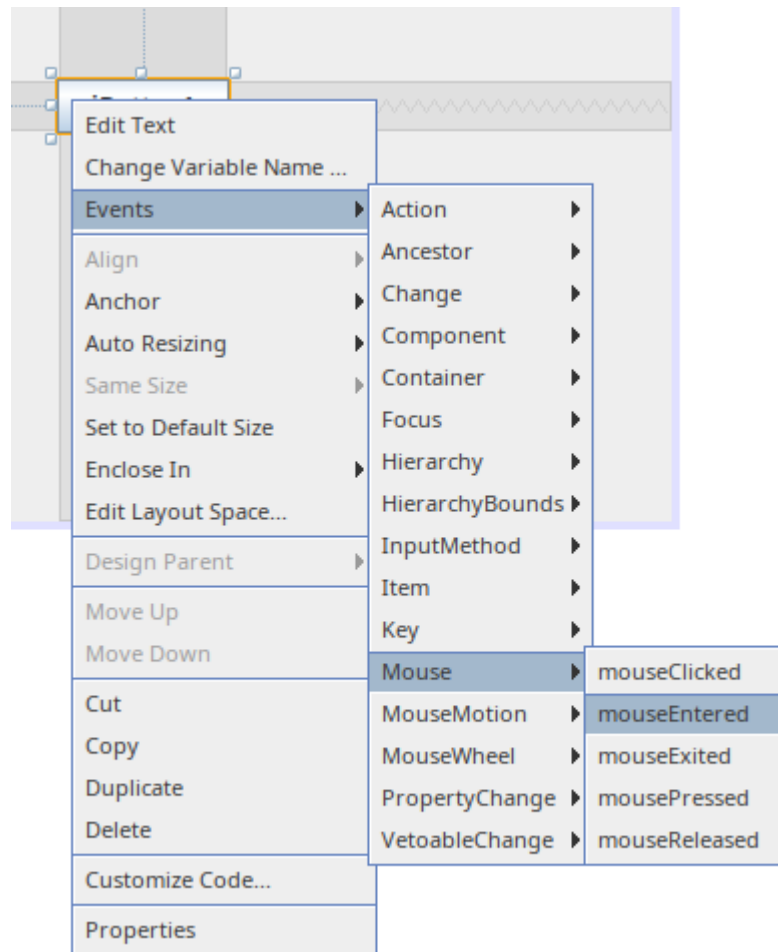
Cada control puede detectar una variedad de acciones, tales como el clic sobre la superficie del mismo, el paso del puntero de mouse, la pulsación de una tecla, etc. y por cada uno de tales eventos se puede asociar un manejador (o muchos, pero en situaciones muy ocasionales).

A pesar de eso, normalmente cada tipo de control posee una acción más frecuente, por ejemplo, de un botón casi siempre es necesario detectar cuando es presionado y muy rara vez es relevante que el mouse pase por encima del mismo sin hacerle clic. De una forma similar, a los cuadros de texto el evento más importante es que el usuario le escriba texto, o en el caso de los controles de listas el evento principal es la selección de uno de sus ítems.

El evento más frecuente de cada control siempre se denomina `actionPerformed`. En la gran mayoría de los casos, ese es el único evento que se necesita manipular.

Para crear un manejador del evento `actionPerformed` de un control se debe hacer doble clic sobre el mismo desde el diseñador. En ese momento se agrega a la clase de la ventana un nuevo método con un nombre especial, el cual no puede ser cambiado. Dentro del mismo se agrega toda la funcionalidad que deba ejecutarse al ocurrir la acción.

Para crear manejadores para otros eventos se hace clic derecho sobre el control y se selecciona desde la opción “Eventos” el que se desea atender. Dicha opción muestra el listado completo con todos los eventos que el control puede reconocer.



Manipulación de atributos por código

Las propiedades de los controles son simplemente atributos de instancias y por lo tanto pueden ser accedidas mediante los métodos `get` y `set`. En el caso de los controles de ingreso de datos, una propiedad permite obtener el dato que el usuario haya ingresado.

De esta manera, los cuadros de texto poseen una propiedad llamada `text` que en todo momento posee como valor el texto que el usuario haya ingresado. Por lo tanto una llamada al método `getText()` permite acceder a ese dato o almacenarlo en una variable. De forma semejante, el método `setText` permite modificar el contenido del control.

Cada tipo de control tiene una propiedad diferente para obtener el ingreso del usuario. En el caso de los `checkbox` y `radio buttons`, la propiedad se denomina `selected` y es de tipo `Boolean`. Para accederla se usa el método `isSelected`, el cual puede ser llamado directamente dentro de la condición de un condicional `simple`.

Los controles de listas en cambio poseen dos propiedades útiles para conocer la selección del usuario. Por un lado existe el método `getSelectedIndex()` que retorna el número de orden de la opción seleccionada, como si las opciones estuvieran guardadas en un arreglo en el orden en que se presentan en la pantalla. La primera opción tiene asignado el índice 0, la segunda el 1 y así sucesivamente. Si el usuario no seleccionó ninguna opción, el método retorna -1.

Pero también existe el método `getSelectedValue()`, que en muchos casos es más recomendable. El mismo retorna una referencia al objeto que representa el ítem seleccionado. Esto se debe a que los controles de listas pueden recibir como elementos a mostrar referencias a objetos. Si se presenta tal situación el control muestra como texto de cada ítem el resultado de llamar al método `toString` de cada objeto. Cuando el usuario selecciona un ítem, el método `getSelectedValue` retorna una referencia al objeto, permitiendo acceder a cualquiera de sus atributos, a pesar de no haberlos mostrado a todos en la ventana. `GetSelectedValue` retorna `null` si no hay ningún ítem seleccionado.

Los controles de listas también permiten selecciones múltiples. Cuando se los configuran de esa manera, los métodos descriptos anteriormente indican los datos del primer ítem seleccionado pero no de todos. Para obtenerlos a todos se disponen los métodos `getSelectedIndices` y `getSelectedValues`, que retornan colecciones con los índices y las referencias de todos los ítems seleccionados, respectivamente.

Las etiquetas de texto (los `JLabel`) son controles que normalmente presentan texto estático que le sirven al usuario para conocer el uso de los otros controles de la ventana. Normalmente cada cuadro de texto posee a un lado una etiqueta que le indica al usuario qué datos debe ingresar en aquél. Como ese texto no cambia nunca y tampoco es habitual atender eventos en esos controles, casi nunca se le asigna nombre a los `label` y se acepta el nombre por defecto que le asigne el IDE.

Pero también son muy usados estos controles para presentar al usuario información no estática tales como cálculos o reportes que se obtienen de los datos del programa pero que no admiten interacción por parte del usuario. Para tal uso sí es imprescindible que las etiquetas tengan asignadas un nombre adecuado para que desde el código se les pueda cambiar el valor con el método `setText`. Dado que la propiedad `text` es de tipo `String`, para asignarle valores numéricos se debe efectuar una conversión. Pero como los datos primitivos no son objetos no tienen método `toString` y para obtener una representación en cadenas de su valor debe ejecutarse `String.valueOf()`.

Hay dos propiedades comunes a todos los controles que suelen ser muy relevantes para desarrollar interfaces correctamente validadas y claras. La propiedad `enabled` indica o establece si el control está habilitado, es decir, disponible para interactuar. Si un control está

deshabilitado el usuario puede verlo pero de color gris y sin que reciba acciones de mouse o teclado.

La propiedad visible en cambio permite ocultar controles hasta que se desee volver a mostrarlos. Ambas son de tipo boolean por lo tanto el método de consulta lleva el prefijo **is** en lugar de **get**.

Ejercicios resueltos

1- Desarrollar un programa con interfaz de ventanas que permita ingresar los datos de un conjunto de alumnos en un arraylist. De cada alumno ingresar nombre, legajo y promedio.



Imagen 12: Extraído de JAVA

```
public class Alumno {  
    private String nombre;  
    private int legajo;  
    private float promedio;  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public int getLegajo() {  
        return legajo;  
    }  
  
    public void setLegajo(int legajo) {  
        this.legajo = legajo;  
    }  
  
    public float getPromedio() {  
        return promedio;  
    }  
  
    public void setPromedio(float promedio) {
```

```
        this.promedio = promedio;
    }

    public Alumno(String nombre, int legajo, float promedio) {
        this.nombre = nombre;
        this.legajo = legajo;
        this.promedio = promedio;
    }

    @Override
    public String toString() {
        return "Alumno{" + "nombre=" + nombre + ", legajo=" + legajo + ", promedio=" +
promedio + '}';
    }
}

public class Principal {

    public static void main(String[] args) {

        AltaAlumnos v = new AltaAlumnos();
        v.setVisible(true);
    }
}

public class AltaAlumnos extends javax.swing.JFrame {

    private ArrayList<Alumno> listaAlumnos;

    public AltaAlumnos() {
        initComponents();
        listaAlumnos = new ArrayList<>();
    }

    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
        .....
    }

    private void btnAgregarActionPerformed(java.awt.event.ActionEvent evt) {
        int legajo = Integer.parseInt(txtLegajo.getText());
        String nombre = txtNombre.getText();
        float promedio = Float.parseFloat(txtPromedio.getText());

        Alumno nuevo = new Alumno(nombre, legajo, promedio);
        listaAlumnos.add(nuevo);
    }
}
```

```
String mensaje = "Cantidad de alumnos: " + listaAlumnos.size() + "\n" +  
listaAlumnos.toString();
```

```
JOptionPane.showMessageDialog(null,mensaje);
```

```
txtNombre.setText("");  
txtLegajo.setText("");  
txtPromedio.setText("");  
}
```

```
private javax.swing.JButton btnAgregar;  
private javax.swing.JLabel jLabel1;  
private javax.swing.JLabel jLabel2;  
private javax.swing.JLabel jLabel3;  
private javax.swing.JTextField txtLegajo;  
private javax.swing.JTextField txtNombre;  
private javax.swing.JTextField txtPromedio;  
}
```

2- Un banco necesita almacenar la información de las tarjetas que poseen sus clientes. Hay 2 tipos de tarjetas: crédito y débito.

Ambas tarjetas presentan el comportamiento de extracción y depósito de dinero.

Tarjeta de Credito. El saldo 0 significa que no tiene compras hechas, y puede comprar hasta llegar hasta su límite.

Atributos:

- int numero
- float saldo
- String titular
- float limite

Metodos:

- Depositar (float importe): método que representa el pago de la tarjeta, o sea que disminuye el saldo en base al importe por parametro.
- Extraer (float importe): método que representa una compra con la tarjeta, validar con el límite para saber si puede realizar o no la compra.

Tarjeta de Débito.

Atributos:

- int numero
- float saldo
- String titular

Metodos:

- Depositar(float importe): método que representa un depósito en el banco a la cuenta del titular
- Extraer(float importe): método que representa una extracción de dinero. Validar que la tarjeta tenga saldo suficiente para realizar la extracción.
Necesitan un sistema que permita (realizar estas acciones desde una clase Banco):
- Agregar una nueva tarjeta

- Depositar y extraer importes en base a número y tipo de tarjeta
- Mostrar el saldo total de las tarjetas de débito
- Mostrar el promedio de saldo de las tarjetas de crédito

Tipo Tarjeta: El saldo total de tarjetas de debito es de : 0.0

Numero: El promedio de saldo de tarjetas de credito es de: 0.0

Saldo:

Titular:

Limite:

☐ Extraccion ☐ Deposito

Tipo Tarjeta

Numero Tarj

Importe

MENSAJES

Imagen 14: Extraído de JAVA

```
public abstract class Tarjeta {  
    private int numero;  
    private float saldo;  
    private String titular;  
  
    public int getNumero() {  
        return numero;  
    }  
  
    public void setNumero(int numero) {  
        this.numero = numero;  
    }  
  
    public float getSaldo() {  
        return saldo;  
    }  
  
    public void setSaldo(float saldo) {  
        this.saldo = saldo;  
    }  
  
    public String getTitular() {  
        return titular;  
    }  
}
```

```
public void setTitular(String titular) {
    this.titular = titular;
}

public Tarjeta(int numero, float saldo, String titular) {
    this.numero = numero;
    this.saldo = saldo;
    this.titular = titular;
}

public abstract boolean depositar(float importe);
public abstract boolean extraer(float importe);
}

public class TarjetaCredito extends Tarjeta {
    private float limite;

    public TarjetaCredito(int numero, float saldo, String titular, float limite)
    {
        super(numero, saldo, titular);
        this.limite = limite;
    }

    public float getLimite() {
        return limite;
    }

    public void setLimite(float limite) {
        this.limite = limite;
    }

    @Override
    public boolean extraer(float importe)
    {
        boolean operacionExitosa = false;

        if(importe + getSaldo() <= limite)
        {
            float nuevImporte = getSaldo() + importe;
            setSaldo(nuevolImporte);
            operacionExitosa = true;
        }
        return operacionExitosa;
    }

    @Override
    public boolean depositar(float importe)
```

```
{
    float saldoNuevo = getSaldo() - importe;
    setSaldo(saldoNuevo);
    return true;
}
}
```

```
public class TarjetaDebito extends Tarjeta {
    public TarjetaDebito(int numero, float saldo, String titular)
    {
        super(numero, saldo, titular);
    }

    @Override
    public boolean extraer(float importe)
    {
        boolean operacionExitosa = false;

        if(getSaldo() >= importe)
        {
            float nuevoSaldo = getSaldo() - importe;
            setSaldo(nuevoSaldo);
            operacionExitosa = true;
        }

        return operacionExitosa;
    }

    @Override
    public boolean depositar(float importe)
    {
        float nuevoSaldo = getSaldo() + importe;
        setSaldo(nuevoSaldo);
        return true;
    }
}
```

```
public class Banco {
    private ArrayList<Tarjeta> tarjetas;

    public Banco()
    {
        tarjetas = new ArrayList<Tarjeta>();
    }

    public void agregarTarjeta(Tarjeta t)
    {
        tarjetas.add(t);
    }
}
```

```
}

//0 debito - 1 credito
public boolean extraer(int numero, int tipo, float importe)
{
    boolean operacionExitosa = false;
    for (Tarjeta tarjeta : tarjetas) {
        if(tipo == 0)
        {
            if(tarjeta instanceof TarjetaDebito && tarjeta.getNumero() == numero)
            {
                operacionExitosa = tarjeta.extraer(importe);
                break;
            }
        }
        else
        {
            if(tarjeta instanceof TarjetaCredito && tarjeta.getNumero() == numero)
            {
                operacionExitosa = tarjeta.extraer(importe);
                break;
            }
        }
    }
    return operacionExitosa;
}

public boolean depositar(int numero, int tipo, float importe)
{
    boolean operacionExitosa = false;
    for (Tarjeta tarjeta : tarjetas) {
        if(tipo == 0)
        {
            if(tarjeta instanceof TarjetaDebito && tarjeta.getNumero() == numero)
            {
                operacionExitosa = tarjeta.depositar(importe);
                break;
            }
        }
        else
        {
            if(tarjeta instanceof TarjetaCredito && tarjeta.getNumero() == numero)
            {
                operacionExitosa = tarjeta.depositar(importe);
                break;
            }
        }
    }
    return operacionExitosa;
}
```

```
}

public float saldoTotalDebito()
{
    float saldoTotal = 0f;
    for (Tarjeta tarjeta : tarjetas) {
        if(tarjeta instanceof TarjetaDebito)
            saldoTotal += tarjeta.getSaldo();
    }
    return saldoTotal;
}

public float promedioCredito()
{
    float saldoTotal = 0f;
    float cantidadTarjetasCredito = 0;

    for (Tarjeta tarjeta : tarjetas) {
        if(tarjeta instanceof TarjetaCredito)
        {
            saldoTotal += tarjeta.getSaldo();
            cantidadTarjetasCredito++;
        }
    }

    return saldoTotal / cantidadTarjetasCredito;
}
}

public class VentanaPrincipal extends javax.swing.JFrame {
    private Banco banco;

    public VentanaPrincipal() {
        initComponents();

        banco = new Banco();
        txtLimite.setEnabled(false);
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        int tipoTarjeta = cmbTipo.getSelectedIndex();
        int numero = Integer.parseInt(txtNumero.getText());
        float saldo = Float.parseFloat(txtSaldo.getText());
        String titular = txtTitular.getText();

        if(tipoTarjeta == 0)
        {
            TarjetaDebito debito = new TarjetaDebito(numero, saldo, titular);
            banco.agregarTarjeta(debito);
        }
    }
}
```



```
}
else
{
    float limite = Float.parseFloat(txtLimite.getText());
    TarjetaCredito credito = new TarjetaCredito(numero, saldo, titular, limite);
    banco.agregarTarjeta(credito);
}
}

private void cmbTipoActionPerformed(java.awt.event.ActionEvent evt) {
    int tipoTarjeta = cmbTipo.getSelectedIndex();
    //Si esta en 0 es debito, bloqueo JTextField limite
    if(tipoTarjeta == 0)
        txtLimite.setEnabled(false);
    //Si esta en 1 es credito, habilito el campo limite
    else
        txtLimite.setEnabled(true);
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    float saldoTotal = banco.saldoTotalDebito();
    float promedio = banco.promedioCredito();

    lblSaldoDebito.setText("El saldo total de tarjetas de debito es de : " + saldoTotal);
    lblPromedioCredito.setText("El promedio de saldo de tarjetas de credito es de: " +
promedio);
}

private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    float importe = Float.parseFloat(txtImporteOperacion.getText());
    int numero = Integer.parseInt(txtNroOperacion.getText());
    int tipo = cmbTipoOperacion.getSelectedIndex();
    boolean exito = false;
    if(rbExtraccion.isSelected())
    {
        exito = banco.extraer(numero, tipo, importe);
    }
    else
    {
        exito = banco.depositar(numero, tipo, importe);
    }

    if(exito)
        lblMensajes.setText("La operacion se registro exitosamente");
    else
        lblMensajes.setText("La operacion no se pudo concretar");
}
```

```
private javax.swing.JComboBox<String> cmbTipo;  
private javax.swing.JComboBox<String> cmbTipoOperacion;  
private javax.swing.JLabel lblMensajes;  
private javax.swing.JLabel lblPromedioCredito;  
private javax.swing.JLabel lblSaldoDebito;  
private javax.swing.JRadioButton rbDeposito;  
private javax.swing.JRadioButton rbExtraccion;  
private javax.swing.JTextField txtImporteOperacion;  
private javax.swing.JTextField txtLimite;  
private javax.swing.JTextField txtNroOperacion;  
private javax.swing.JTextField txtNumero;  
private javax.swing.JTextField txtSaldo;  
private javax.swing.JTextField txtTitular;  
}
```

BIBLIOGRAFÍA

- Ceballos Sierra, F. (2010). *Java 2. Curso de Programación*. 4ta Edición. Madrid, España. RA-MA Editorial.
- Corso, C; Colaccioppo, N. (2012). “*Apunte teórico-práctico de Laboratorio de Computación III*”. Córdoba, Argentina. Edición digital UTN-FRC.
- Oracle Java Documentation. (febrero/ 2020). Título: “The Java Tutorials. Creating a Gui with JFC/Swing”. Lugar de publicación: docs.oracle.com. Recuperado de <https://docs.oracle.com/javase/tutorial/uiswing/start/index.html>



Atribución-NoComercial-SinDerivadas

Se permite descargar esta obra y compartirla, siempre y cuando no sea modificado y/o alterarse su contenido, ni se comercializarse. Referenciarlo de la siguiente manera:

Universidad Tecnológica Nacional Regional Córdoba (2020). Material para la Tecnicatura en Programación Semipresencial de Córdoba. Argentina.