Práctica 7
Interfaces gráficas de usuario (GUI)

Diseñada por: Carlos Alfredo Campos De la Garza

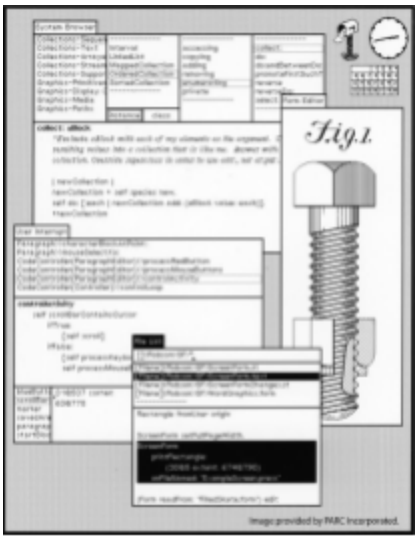
Objetivos de la práctica

- Aprender a crear una interfaces gráficas sencillas en Java.
- Conocer algunos de los componentes de Swing.
- Reconocer las diferencias entre los diferentes manejadores de layout.
- Aprender a atender eventos de botones y de componentes de texto.
- Programar un juego sencillo con interfaz gráfica.

Introducción

Interfaces gráficas de usuario

Las interfaces gráficas de usuario o GUI (por las siglas en inglés de Graphical User Interface) son programas de computadora que permiten la interacción del usuario con la computadora a través de elementos visuales (como símbolos, imágenes o íconos), metáforas y dispositivos de señalamiento (como un mouse o una pluma). Estas interfaces disminuyeron considerablemente la dificultad que existía al utilizar las computadoras con las interfaces de texto e incrementaron la posibilidad de que las computadoras fueran utilizadas por personas no especializadas.



El lenguaje Smalltalk introducía un estilo "orientado a objetos" de programación. En los sistemas actuales permanecen las ventanas con un título en la parte superior izquierda y que se superponen. Imagen propiedad de Palo Alto Research Center, Incorporated (1980).

Las interfaces gráficas se han diseñado a partir de las ideas sobre cómo debería interactuar el ser humano con una computadora. Desde la invención de las computadoras han habido visiones de las interfaces gráficas y algunas de ellas se han vuelto realidad. En el año de 1945, Vannevar Bush, publicó el artículo *As we may think* donde él describe un dispositivo llamado "memex" que podía almacenar la información del usuario y que podía recuperarla y ponerla a la disposición del usuario en cuestión de segundos. Bush dio origen a la palabra hipertexto e inspiró a otras personas para el desarrollo de innovaciones en la computación.

Influenciado por las ideas de Bush, Ivan Sutherland, un estudiante de doctorado en el MIT, desarrolló en su tesis doctoral un programa llamado *Sketchpad* que permitía la manipulación de elementos gráficos en un monitor CRT a partir de un lápiz óptico. En el programa de Sutherland se podían dibujar líneas, arcos, realizar transformaciones, aumentar o alejar un dibujo y almacenar objetos en memoria.

Video de demostración de *Sketchpad* https://youtu.be/6orsmFndx_o

Primer prototipo de un mouse. Desarrollado por Douglas Engelbart en el SRI.

Alguien que también fue encantado por el artículo de Vannevar Bush y que fue inspirado por el trabajo de Sutherland fue Douglas Engelbart. Él, a inicios de los años sesenta en el Stanford Research Institute (SRI), fundó el Augmented Research Center donde se encargaría de desarrollar innovaciones para la computadora. En ese periodo, Engelbart desarrolló un prototipo del dispositivo apuntador más popular hasta nuestros tiempos: el mouse. Junto con su equipo, él desarrolló un sistema llamado *oN-Line System* (NLS) que buscaba llevar a la realidad las ideas de Bush. El NLS se conectaba en red, tenía ventanas, hipertexto, gráficos, entrada de comandos, videoconferencia, un mouse, control de versiones de archivos, editor de texto, entre otras novedosas características. Engelbart hizo la demostración del NLS frente al público en la que se conoce com "la madre de todas las demostraciones".

Video de "la madre de todas las demostraciones": https://youtu.be/yJDv-zdhzMY

El NLS dejó de tener popularidad debido a la alta dificultad para aprender a usar sus aplicaciones, a que comenzaron a surgir las minicomputadoras y a la aparición del internet. Debido a la dificultad de Engelbart por continuar con un NLS mejorado en nuevas computadoras, muchos de sus desarrolladores dejaron el SRI y fueron al Xerox Palo Alto Research Center (PARC) y se llevaron la idea exitosa del mouse con ellos. Uno de los integrantes del PARC fue Alan Kay, quien trabajó con Sutherland y había visto el demo de Engelbart. Él creó el lenguaje y ambiente de programación Smalltalk que estaba orientado a objetos y contaba con un sistema de ventanas. Además, él desarrolló junto con su equipo la computadora Alto que contaba el primer sistema operativo con interfaz gráfica de usuario e incluía la metáfora de *escritorio*. Este sistema tenía un cliente de correos, un procesador de textos WYSIWYG, un editor de gráficos vectoriales, un editor de dibujos (paint) y un juego multijugador en red.

WYSIWYG es el acrónimo de *What You See Is What You Get* que hace referencia a los programas en donde el producto final se visualiza todo el tiempo en pantalla. Actualmente muchos programas siguen esta idea, por ejemplo Word, PowerPoint y Photoshop.

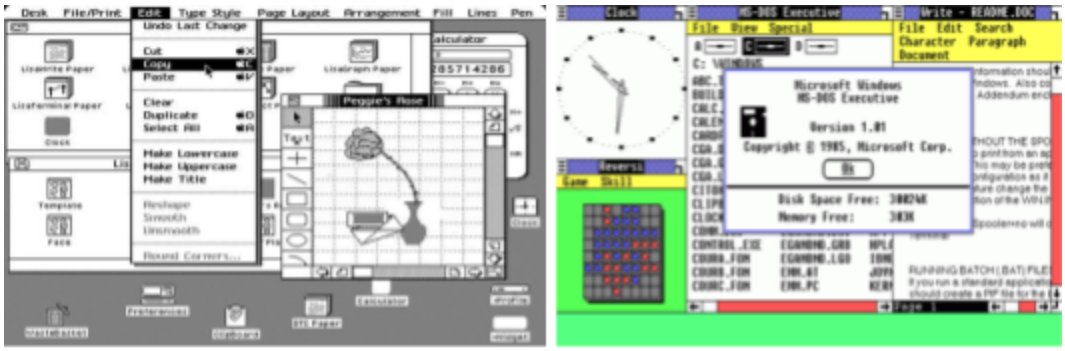
La computadora Alto con su sistema operativo innovador fue un proyecto muy atractivo que no estuvo a la venta. En 1979, Steve Jobs visita el PARC para ver una demostración de Alto (que contaba con mouse) y su GUI elaborada con Smalltalk. Jobs quedó fascinado por el proyecto por lo que invitó a varios desarrolladores de Xerox para trabajar en algo similar para su computadora Lisa. Aunque Xerox distribuyó una versión comercial de su sistema llamado Xerox Star, fue un fracaso comercial mientras que la computadora Lisa es recordada como la primera computadora que contenía la primera GUI distribuida comercialmente. Dentro de las innovaciones de Apple en el campo de las GUI se encuentran las barras de menú y los menús desplegables. Un año después de Lisa, en 1984, Apple lanza al mercado su computadora Macintosh de bajo costo que logró acercar a la población una computadora con GUI.

Cartel publicitario de la Macintosh de Apple (1984).

Video promocional de la Macintosh, considerado como una obra maestra por su contenido metafórico https://youtu.be/zIE-5hg7FoA

Aunque la Macintosh fue un éxito comercial, los expertos en computación se veían muy limitados al usarla por lo que la consideraban un juguete. Las computadoras de IBM (International Bussiness Machines Corporation) con el sistema operativo DOS (Disk Operative System) con interfaz de línea de comandos seguían siendo utilizadas

por los profesionales. Fue hasta 1990 cuando Microsoft lanzó su sistema Windows 3.0 con una interfaz gráfica aceptable para las computadoras IBM.



Las primeras GUI de los sistemas operativos de Apple, Microsoft y Linux en un solo video: https://youtu.be/JcOifOFZgE0

(a) Imagen de la primera GUI comercial de Apple. Influenciados por el trabajo del centro de investigación de Xerox, Apple desarrolló su propia GUI para su computadora Lisa. Aunque Lisa fracasó por su alto precio y mal desempeño, tenía características sobresalientes. (1984)
(b) Microsoft no tardó demasiado en lanzar al mercado su nuevo sistema operativo con interfaz gráfica inicialmente llamada VisiOn pero renombrada a Windows. En su versión 1.0 ya tenía gráficos a color. (1985)

Como consecuencia del surgimiento de interfaces gráficas para los sistemas DOS (que después sería Windows) de las computadoras IBM y para las computadoras Apple, comenzaron a surgir interfaces gráficas para los sistemas basados en UNIX. Para estos sistemas, las GUI fueron pensadas como "envolventes" de tal manera que los usuarios acostumbrados a la interfaz de línea de comandos pudieran seguir usándola y los usuarios menos experimentados tuviera acceso al sistema mediante la GUI. Debido a esta característica opcional, en sistemas basados en UNIX se cuenta con un sistema de ventanas (que establece cómo se deberán desplegar las ventanas) y con un gestor de ventanas (que define el comportamiento de las ventanas al interactuar con el usuario).

Los gestores de ventana de UNIX están basados en un sistema de ventanas llamado *X System* que fue desarrollado en el MIT en 1984 con la finalidad de estandarizar un sistema de ventanas.

Entorno de escritorio KDE Plasma (izquierda) y GNOME 3.0 (derecha).
KDE fue fundado por Matthias Ettrich en 1996 mientras que GNOME fue fundado por Miguel de Icaza y Fernando Mena en 1997.

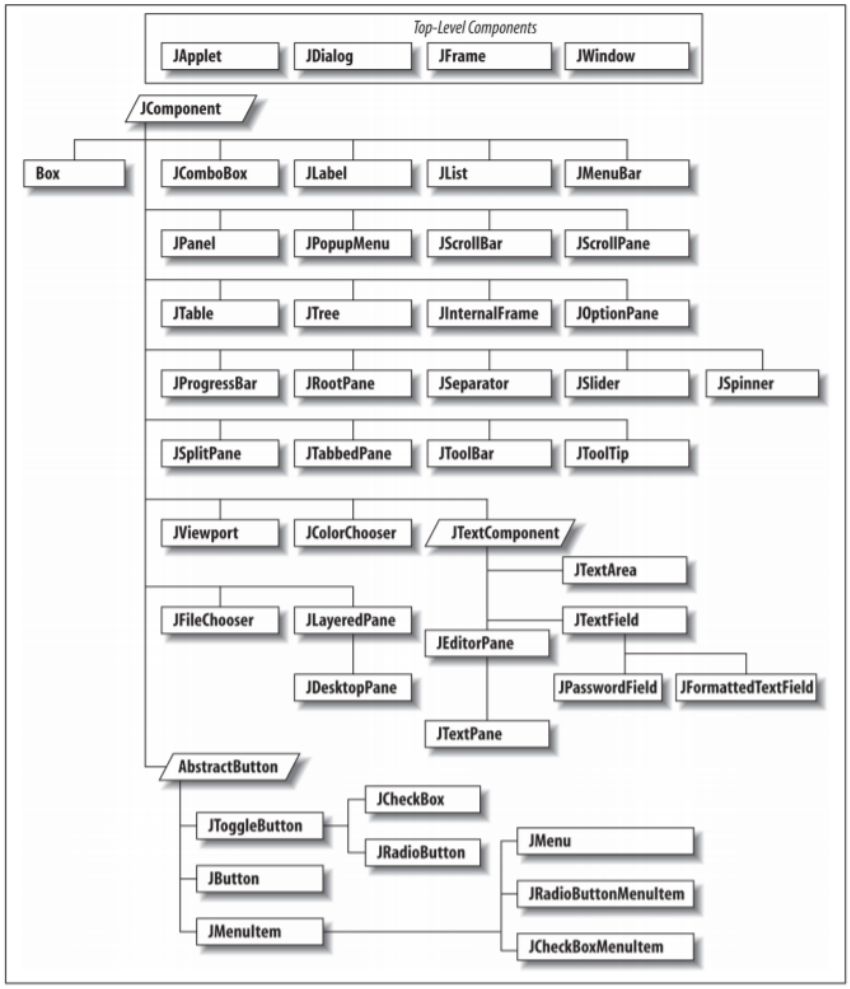
Las interfaces gráficas de la actualidad conservan muchas características de las primeras y han evolucionado para ser cada vez más intuitivas y simplificar la interacción del usuario con la computadora. Con el surgimiento de nuevas tecnologías como los dispositivos móviles, las GUI han tenido que adaptarse para mostrarse en tamaños y orientaciones variables de pantalla, así como para reaccionar a dispositivos diferentes del mouse y el teclado, como ahora ocurre con las pantallas táctiles.

AWT y Swing

En Java, las interfaces gráficas de usuario se crean a partir de las clases de los paquetes java.awt y java.swing. En versiones antiguas de Java (antes de Java 2) sólo existían las clases de AWT (Abstract Window Toolkit) para crear programas con GUI. Después, surgió Swing como una alternativa a AWT pero este último no desapareció con la finalidad de conservar compatibilidad con programas creados en otras versiones de Java. Todas las clases de AWT tienen una contraparte en Swing y es preferible utilizar las clases de Swing.

Las clases de AWT utilizan los objetos de GUI nativos de cada sistema operativo. Un botón de AWT se desplegará y comportará como un botón de Windows en un sistema operativo Windows y como un botón Macintosh para los sistemas Mac, por ejemplo. En contraste, las clases de Swing están implementadas completamente en Java, tienen mayor compatibilidad y se comportan igual en diferentes sistemas operativos. Además, al ser independientes del sistema operativo, las clases de Swing incluyen capacidades adicionales a las que presentan las clases de AWT.

Las clases de Swing suelen llamarse clases de peso ligero mientras que las de AWT se les llama clases de peso pesado. Por la diferencia en la implementación no se recomienda combinar las clases de los dos paquetes.



Clases del paquete Swing. Imagen tomada de Niemeyer y Leuck (2013).

Ventanas, marcos, etiquetas y botones

Las ventanas y los marcos son los únicos componentes que pueden desplegarse en pantalla sin ser añadidos a otro contenedor. El siguiente código crea un objeto JWindow y un objeto JFrame y los despliega juntos para visualizar sus diferencias:

```
import javax.swing.*;
public class Interfaces1 {
   public static void main(String[] args){
        JFrame marco = new JFrame("Un marco");
        marco.setSize(300, 300);
        marco.setLocation(100,100);
        JWindow ventana = new JWindow();
        ventana.setSize(300,300);
        ventana.setLocation(500, 100);
        //Hacer visibles los componentes
        marco.setVisible(true);
        ventana.setVisible(true);
   }
}
```

Tanto el JWindow como el JFrame son ventanas. La diferencia se encuentra en que JWindow sólo representa la región donde aparece la ventana mientras que un JFrame es una ventana que tiene bordes y una barra de título y dentro de ella contiene algunos botones. Los JFrame se pueden comportar como las ventanas de la GUI del sistema operativo. En este documento nos referiremos a los JFrame como ventanas.

La ventana de inicio de una aplicación se conoce como SplashScreen y sirve para mostrarle al usuario que la aplicación está iniciando y cargando los componentes necesarios. Debido a que swing carga después de iniciar la aplicación, no es posible hacer una Splashscreen con un objeto JWindow. Si deseas conocer más de esto consulta los capítulos 3 y 16 de *Learning Java* de Niemeyer y Leuck (2013).

Las ventanas tienen algunos métodos que permiten configurar sus propiedades. Algunas de ellos se muestran en la siguiente tabla:

Método	Descripción
setTitle	Establece el título de la ventana
setIconImage	Define el ícono de la ventana
setSize	Define el tamaño de la ventana. El tamaño se define en píxeles.
setLocation	Indica dónde se ubicará la esquina superior izquierda de la ventana.
setLocation	Establece el tipo de manejador de layout que organizará los componentes de la ventana.
setVisible	Determina si la ventana se debe ver en la pantalla.
setDefaultCloseOperation	Indica qué debe suceder con el programa cuando la ventana se cierra.

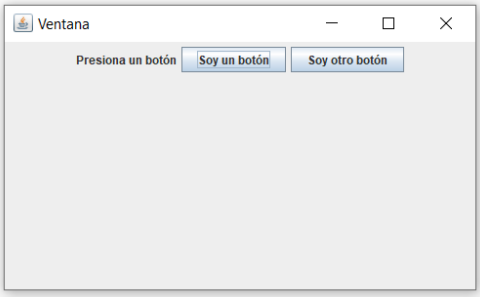
Un método que es importante mencionar es setDefaultCloseOperation. Este método indica qué debe suceder con el programa cuando la ventana se cierre. Los posibles valores de acción son los siguientes:

- JFrame.EXIT_ON_CLOSE: Cierra el programa.
- JFrame.HIDE_ON_CLOSE: Oculta el JFrame pero mantiene el programa ejecutándose.
- JFrame.DISPOSE_ON_CLOSE: Se destruye el objeto, pero mantiene el programa ejecutándose.
- JFrame.DO_NOTHING_ON_CLOSE: Ignora el click.

Todas las ventanas pueden contener otros elementos de Swing como botones, etiquetas o cuadros de texto. Para poder insertarlos, éstos se deben añadir a un contenedor. Las ventanas pueden contener varias capas de contenedores llamadas paneles o cristales (traducción del inglés pane) que se superponen. Un JFrame contiene un ContentPane al cual se le pueden insertar componentes con el método add. Si se cuenta con un JFrame llamado ventana, el panel contenedor se puede obtener con el método ventana.getContentPane() y se puede insertar un elemento al panel con la línea ventana.getContentPane().add(). Para mayor facilidad, Java permite que se inserten los elementos de un JFrame simplemente llamando el método add del objeto JFrame. A continuación se muestra un ejemplo para crear una ventana que incluya una etiqueta y dos botones:

Para cambiar el color de fondo de un panel existe el método setBackground(). Este método recibe como parámetro un objeto de tipo Color del paquete awt.

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

public class Interfaces1 {
   public static void main(String[] args){
        JFrame ventana = new JFrame("Ventana");
        ventana.setSize(500, 300);
        ventana.setLocation(200,100);
        ventana.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        ventana.setLayout(new FlowLayout());

        JLabel etiqueta = new JLabel("Presiona un botón");
        JButton boton1 = new JButton("Soy un botón");
        JButton boton2 = new JButton("Soy otro botón");
        ventana.add(etiqueta);
        ventana.add(boton1);
        ventana.add(boton2);
        ventana.setVisible(true);
    }
}
```



En el código anterior se crearon tres objetos, dos de tipo JButton y uno de tipo JLabel y se agregaron al JFrame llamado ventana. Para que éstos se organicen correctamente en la ventana se debe definir un layout para el JFrame mediante el método setLayout. Más adelante se profundizará en este tema.

Existe otro tipo de panel que resulta muy útil en la práctica. El nombre de este panel es JOptionPane y permite mostrarle al usuario una ventana en la que se le muestre un mensaje, se le solicite seleccionar una opción o se le pida introducir un texto. Las ventanas de diálogo de este panel más usadas se muestran con los métodos:

- showInputDialog: muestra una ventana con un cuadro de texto para solicitar una cadena de texto.
- showConfirmDialog: muestra una ventana con botones con las opciones sí, no o cancelar.
- showMessageDialog: muestra una ventana de diálogo con un mensaje.

A diferencia de las clases que leían datos del teclado y que podían devolver valores de tipo numérico, en los componentes de Swing siempre se proporcionará un objeto de tipo String. Para convertir a un valor numérico se deberá usar el método parse de las clases envolventes.

El cuadro de diálogo de confirmación puede configurarse para mostrar uno, dos o los tres botones. Las posibles opciones son YES_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION y OK_CANCEL_OPTION. Dependiendo del botón que presione el usuario el valor que devolverá el método podrá ser YES_OPTION, NO_OPTION, CANCEL_OPTION, OK_OPTION y CLOSED_OPTION.

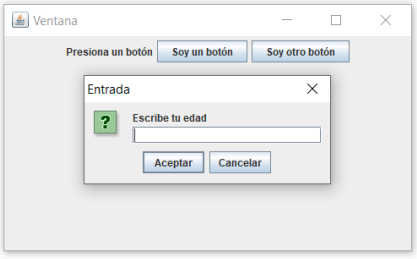
El cuadro de diálogo de mensaje puede modificar el ícono que muestra al usuario. Las opciones posibles son las que se muestran en la siguiente tabla:

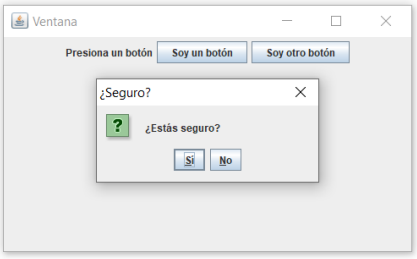
Tipo de diálogo	Ícono	Descripción
ERROR_MESSAGE		Un diálogo que indica un error al usuario.
INFORMATION_MESSAGE		Un diálogo que muestra un mensaje informativo.
WARNING_MESSAGE		Un diálogo que advierte sobre un problema.
QUESTION_MESSAGE		Diálogo con una pregunta cerrada para el usuario.
PLAIN_MESSAGE		Diálogo con un mensaje pero sin ícono.

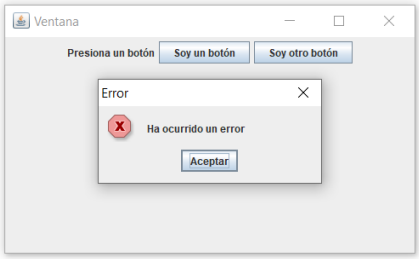
Para probar el funcionamiento del panel de opciones se pueden agregar las siguientes líneas al ejemplo anterior:

Para mayor información de los parámetros de los métodos de JOptionPane consulta la documentación de Java.

```
JOptionPane.showInputDialog(ventana, "Escribe tu edad");
JOptionPane.showMessageDialog(ventana, "Ha ocurrido un error", "Error",
        JOptionPane.QUESTION_MESSAGE);
JOptionPane.showConfirmDialog(ventana, "¿Estás seguro?", "¿Seguro?",
        JOptionPane.YES_NO_OPTION);
```







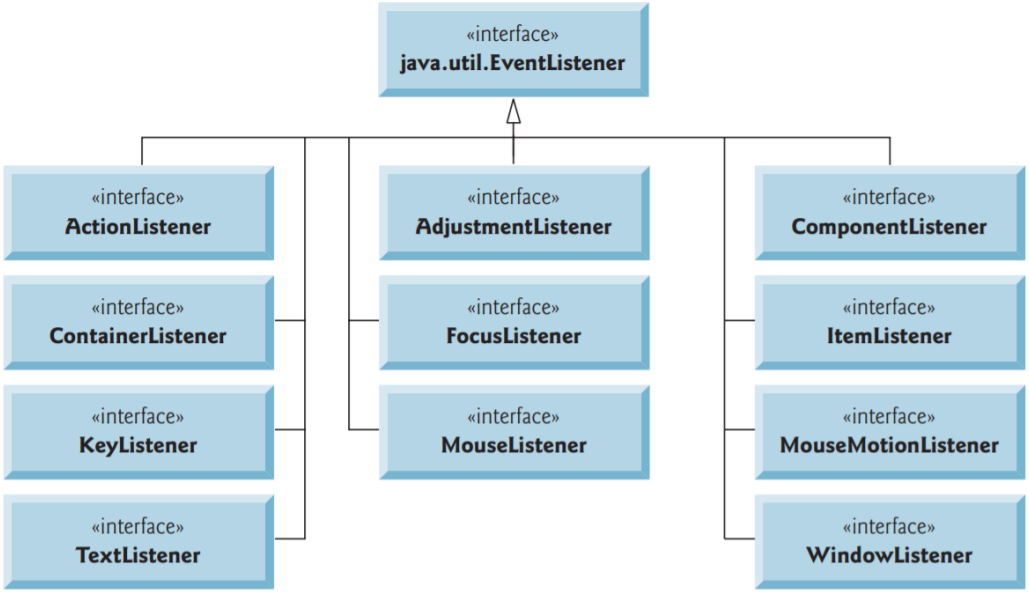
Programación dirigida por eventos

Hasta el momento se ha mostrado cómo insertar componentes de Swing de una ventana y cómo mostrar ventanas de diálogo. Sin embargo, en los ejemplos anteriores los objetos JButton no realizan alguna acción. En Java, así como en otros lenguajes de programación, cuando se trabaja con interfaces gráficas se utiliza el paradigma de **programación dirigida por eventos**. Este paradigma se basa en la idea de que el orden de ejecución de un programa depende de los eventos que sucedan y que son frecuentemente desencadenados por el usuario.

Un **evento** puede ser cualquier cambio de estado de un objeto, ya sea temporal o permanente, como el clic en un botón, presionar una tecla, cambiar el contenido de un cuadro de texto, recibir un nuevo correo en la bandeja de correo electrónico, etc. Los objetos de la GUI donde puede suceder un evento se denominan como **fuente de eventos** *(event source)* mientras que los objetos que se encargarán de definir las acciones que se realizarán al momento de que un evento suceda se llaman **manejadores de eventos** (*event handler),* los cuales implementan una interfaz de **escucha u oyente de eventos** *(event listener)* específicos de un tipo de componente. Es posible que un mismo objeto sea tanto como la fuente como el oyente.

Por defecto los componentes de GUI de Java no tienen acciones definidas cuando un evento sucede. Para definir las acciones para un determinado evento (atender un evento) se deben seguir los siguientes pasos:

1. Crear una clase que será el manejador de eventos.
2. Implementar la interfaz del oyente de métodos apropiada para el tipo de evento que se desea atender.
3. Indicar que el componente debe notificar a un objeto de la clase manejadora de eventos cuando un evento suceda. A esto se le conoce como registrar el manejador de eventos.



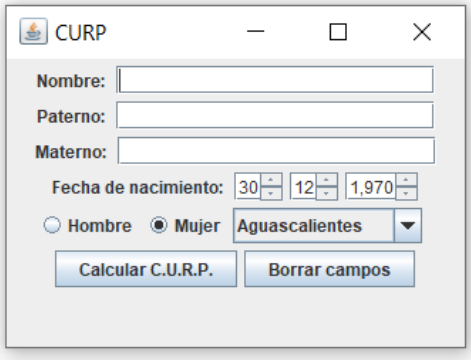
Algunos *event listeners* más comunes. Diagrama tomado de Deitel y Deitel (2012).

Manejo de eventos

Para comenzar esta sección se desarrollará una interfaz gráfica para construir los primeros dieciséis caracteres de la CURP a partir de los apellidos, nombre,fecha de nacimiento, sexo y lugar de nacimiento de una persona. En este ejemplo, se utilizará una técnica de desarrollo de GUI que es muy común: la clase principal será una subclase de JFrame. Esto nos permite crear un nuevo objeto que tenga como atributos otros elementos de Swing y poderlos modificar en diferentes instantes, como se verá después cuando se añadan los manejadores de eventos.

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.TreeMap;

public class CURPFrame extends JFrame{
   private JTextField txtNombre;
   private JTextField txtApePat;
   private JTextField txtApeMat;
   private JButton botonOK;
   private JButton botonReset;
   private JSpinner spinDia;
   private JSpinner spinMes;
   private JSpinner spinAño;
   private JRadioButton opHombre;
   private JRadioButton opMujer;
   private JComboBox<String> selectEstado;
   private JLabel lblCURP;
   private TreeMap<String, String> estados;

   public static void main(String[] args){
       CURPFrame ventana = new CURPFrame();
   }

   private void cargarClaves(){
       String[] nombres = {"Aguascalientes", "Baja California", "Baja
              California Sur", "Campeche", "Chiapas", "Chihuaha", "Ciudad de
              México", "Coahuila", "Colima", "Durango", "Guanajuato",
              "Guerrero", "Hidalgo", "Jalisco", "México", "Michoacán", "Morelos",
              "Nayarit", "Nuevo León", "Oaxaca", "Puebla", "Querétaro",
              "Quintana Roo", "San Luis Potosí", "Sinaloa", "Sonora", "Tabasco",
              "Tamaulipas", "Tlaxcala", "Veracruz", "Yucatán",     "Zacatecas"};
        String[] clave = {"AS", "BC", "BS", "CC", "CS", "CH", "DF", "CL",
                          "CM", "DG", "GT", "GR", "HG", "JC", "EM", "MI",
                          "MO", "NA", "NL", "OA", "PU", "QT", "QR", "SL",
                          "SI", "SO", "TB","TM", "TL", "VE", "YU", "ZA"};
       estados = new TreeMap<String, String>();
       for(int i=0; i<nombres.length; i++)
           estados.put(nombres[i],clave[i]);
   }
```

```
public CURPFrame(){
    super("CURP");
    cargarClaves();
    setSize(340,250);
    setLocation(500,200);
    JLabel lblNombre = new JLabel("Nombre: ");
    JLabel lblApePat = new JLabel("Paterno: ");
    JLabel lblApeMat = new JLabel("Materno: ");
    JLabel lblFechaNac = new JLabel("Fecha de nacimiento: ");
    txtNombre = new JTextField(20);
    txtApePat = new JTextField(20);
    txtApeMat = new JTextField(20);
    ButtonGroup opSexo = new ButtonGroup();
    opHombre = new JRadioButton("Hombre");
    opMujer = new JRadioButton("Mujer", true);
    opSexo.add(opHombre);
    opSexo.add(opMujer);
    selectEstado = new JComboBox<>();
    estados.forEach((llave, valor)->selectEstado.addItem(llave));
    botonOK = new JButton("Calcular C.U.R.P.");
    botonReset = new JButton("Borrar campos");
    spinDia = new JSpinner();
    spinDia.setValue(30);
    spinMes = new JSpinner();
    spinMes.setValue(12);
    spinAño = new JSpinner();
    spinAño.setValue(1970);
    lblCURP = new JLabel(" ");

    setLayout(new FlowLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    add(lblNombre);
    add(txtNombre);
    add(lblApePat);
    add(txtApePat);
    add(lblApeMat);
    add(txtApeMat);
    add(lblFechaNac);
    add(spinDia);
    add(spinMes);
    add(spinAño);
    add(opHombre);
    add(opMujer);
    add(selectEstado);
    add(lblCURP);
    add(botonOK);
    add(botonReset);
    setVisible(true);
}
```



En el código anterior se creó una ventana del tipo JFrame que contiene tres instancias de la clase JTextField, tres objetos de tipo Spinner, dos objetos JRadioButton, un objeto JComboBox y dos botones JButton. Además, para mostrar los textos auxiliares se utilizaron etiquetas JLabel. Debido a que la clase RFCFrame hereda de la clase JFrame, dentro del constructor se llama al constructor de la superclase y se define el título de la ventana. Por la misma razón, en el constructor se llama a los métodos que se heredaron de la clase JFrame (como los métodos add y setVisible).

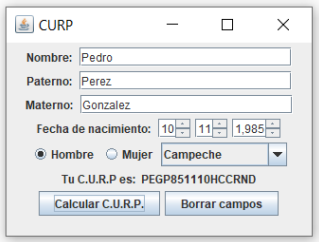
A continuación se registrará un manejador de eventos para los botones de la GUI. Los botones pueden provocar un evento del tipo ActionEvent por lo que la clase que se definirá como manejador de eventos deberá implementar la interfaz ActionListener. Ésta contiene un solo método llamado actionPerformed que se ejecutará cada vez que un evento del tipo ActionEvent suceda en un objeto que tenga registrado este manejador de eventos. El manejador que se definirá en este ejemplo será una clase llamada ManejadorClick y se encontrará dentro de la clase principal con el fin de tener un acceso fácil a los componentes del formulario. Las clases que se encuentran dentro de otra clase se conocen como **clases internas o anidadas**. El código que le dará funcionalidad a los botones será el siguiente:

Una clase interna puede utilizar todos los métodos y los atributos de la clase que la contiene, incluso las que tienen el modificador private.

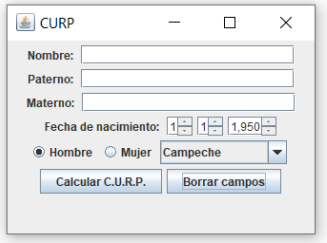
```
class ManejadorClick implements ActionListener{
   @Override
   public void actionPerformed(ActionEvent e) {
     JButton boton = (JButton) e.getSource();
     if(boton.equals(botonOK)) {
      try {
        String curp = txtApePat.getText().substring(0, 2).toUpperCase() +
               txtApeMat.getText().substring(0, 1).toUpperCase() +
               txtNombre.getText().substring(0, 1).toUpperCase() +
               spinAño.getValue().toString().substring(2) +
               spinMes.getValue().toString()+spinDia.getValue().toString();
        curp += (opHombre.isSelected())? "H" : "M";
        curp += estados.get(selectEstado.getSelectedItem());
        curp += txtApePat.getText().substring(1).replaceAll("[aeiou]", "").
              substring(0,1).toUpperCase();
        curp += txtApeMat.getText().substring(1).replaceAll("[aeiou]", "").
              substring(0,1).toUpperCase();
        curp += txtNombre.getText().substring(1).replaceAll("[aeiou]", "").
              substring(0,1).toUpperCase();
        lblCURP.setText("Tu C.U.R.P es:  " + curp);
      } catch (StringIndexOutOfBoundsException ex){
        lblCURP.setText("Complete los campos");
      }
     }else{
           lblCURP.setText("");
           txtApeMat.setText("");
           txtApePat.setText("");
           txtNombre.setText("");
           spinDia.setValue(1);
           spinMes.setValue(1);
           spinAño.setValue(1950);
       }
   }
}
```

El objeto de tipo ActionEvent contiene información sobre el evento. Con el método getSource se puede obtener la instancia que fue la fuente del evento.

Lo único que hace falta para que los botones funcionen es registrar el manejador de eventos en cada uno de los botones. Los objetos de tipo JButton cuentan con el método addActionListener para registrar manejadores de eventos. Para que los botones del ejemplo adquieran funcionalidad bastará con incluir las siguientes dos líneas en el constructor de la ventana:



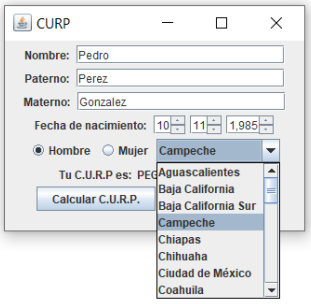

```
botonOK.addActionListener(new ManejadorClick());
botonReset.addActionListener(new ManejadorClick());
```



Con el código que se desarrolló en este ejemplo los botones son capaces de reaccionar a un clic. En el ejemplo se usó un solo manejador de eventos para los dos botones. Para que cada botón tuviera un comportamiento diferente se necesitó conocer el botón de donde provino el evento mediante el método getSource del objeto de tipo ActionEvent. El mismo resultado se pudo haber logrado definiendo dos diferentes clases manejadoras de eventos y registrar cada una con un botón diferente. Aunque cualquiera de las dos técnicas tienen el mismo efecto, es recomendable usar clases diferentes cuando la acción que realice cada botón sea demasiado compleja y requiera demasiadas líneas de código.

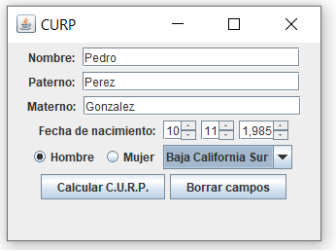
Cuando se le daba clic al botón “Calcular C.U.R.P.” se despliega el CURP calculado a partir de la información que está en los controles en ese instante. Si el usuario modifica la selección del Estado de nacimiento o del sexo, el CURP desplegado no corresponderá con los valores proporcionados hasta que se vuelva a dar clic al botón. Para mejorar la experiencia con la GUI, a continuación se agregará otro tipo de evento para que cuando se cambie la selección del JComboBox o del JRadioButton se oculte el texto del CURP desplegado anteriormente.

Los componentes JComboBox y JRadioButton pueden provocar un evento del tipo ItemEvent que ocurre cuando la selección del elemento es modificada. La clase manejadora deberá implementar la interfaz ItemListener la cual contiene sólo un método llamado itemStateChanged.




```
class ManejadorItem implements ItemListener{
    @Override
    public void itemStateChanged(ItemEvent e) {
        lblCURP.setText("");
    }
}
```

Finalmente, el manejador de eventos ManejadorItem deberá ser registrado en los elementos de selección del sexo y del Estado de nacimiento de la interfaz gráfica:




```
opHombre.addItemListener(new ManejadorItem());
opMujer.addItemListener(new ManejadorItem());
electEstado.addItemListener(new ManejadorItem());
```

Manejadores de layout o gestores de diseño

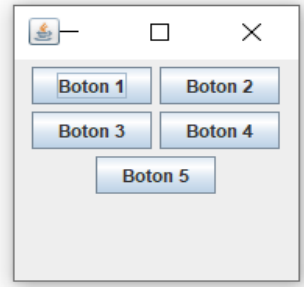
FlowLayout

En este tipo de esquema los componentes son colocados de izquierda a derecha. Cuando el componente no cabe en la misma línea, éste se ubicará en la siguiente línea y se mantendrá el orden de acomodo de izquierda a derecha. Por defecto, todos los componentes de una sola línea aparecen centrados. Cuando la ventana que los contiene cambia de tamaño los elementos son reacomodados para tratar de de completar las líneas.

La alineación de los elementos se puede definir cuando se instancia un objeto FlowLayout. Los posibles valores que se pueden introducir son FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.LEADING, FlowLayout.TRAILING y FlowLayout.CENTER.

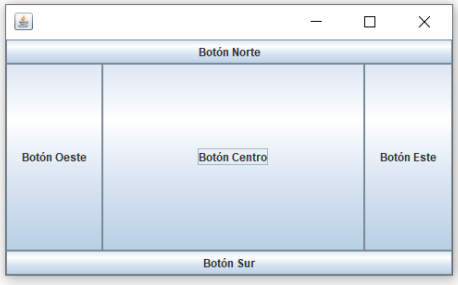
En los sistemas en los que se lee de izquierda a derecha, LEADING indica una alineación izquierda y TRAILING una alineación derecha.

```
import javax.swing.*;
import java.awt.*;
public class VentanaPrincipal {
   public static void main(String[] args){
        JFrame vFlow = new JFrame();
        vFlow.setSize(200,200);
        vFlow.setLayout(new FlowLayout(FlowLayout.CENTER));
        for(int i=1; i<=5; i++){
            vFlow.add(new JButton("Botón "+ i));
        }
        vFlow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        vFlow.setVisible(true);
    }
}
```



BorderLayout

El gestor BorderLayout es el gestor por defecto para los paneles de contenido de los objetos JFrame , JWindow y JDialog. Él provee una manera de organizar los componentes a partir de la definición de su ubicación respecto al borde de la ventana. La ventana se divide en cinco regiones: center, north, south, east y west.




```
import javax.swing.*;
import java.awt.*;
public class VentanaPrincipal {
   public static void main(String[] args){
        JFrame vBorder = new JFrame();
        vBorder.setSize(500,300);
        vBorder.setLayout(new BorderLayout());
        vBorder.add(new JButton("Botón Norte"), BorderLayout.NORTH);
        vBorder.add(new JButton("Botón Sur"), BorderLayout.SOUTH);
        vBorder.add(new JButton("Botón Este"), BorderLayout.EAST);
        vBorder.add(new JButton("Botón Oeste"), BorderLayout.WEST);
        vBorder.add(new JButton("Botón"), BorderLayout.CENTER);
        vBorder.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        vBorder.setVisible(true);
 }
```

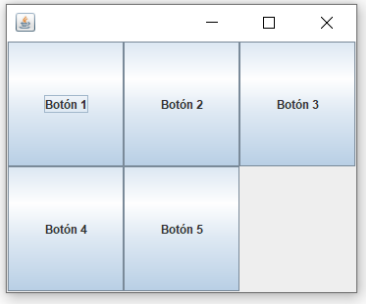
El componente ocupa todo el espacio de la región por lo cual sólo puede haber un elemento en cada región.

Cuando no se indica una región al agregar un componente el gestor de esquemas asume que se debe agregar en la región CENTER

No es necesario ocupar todas las regiones definidas por este esquema. Las regiones no ocupadas serán aprovechadas por las regiones que sí contienen elementos

GridLayout

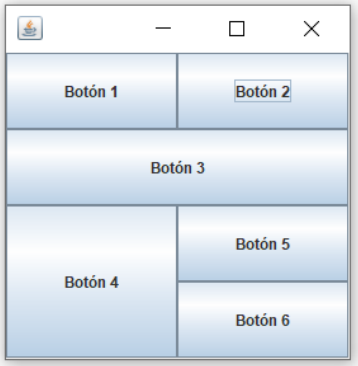
El gestor GridLayout define un esquema de diseño similar al de una cuadrícula. El contenedor se divide en una cuadrícula de N filas por M columnas y el gestor se encarga de definir el ancho y el alto de cada celda para que todas tengan el mismo tamaño. Los componentes se agregan al contenedor comenzando en la celda superior izquierda de la cuadrícula yendo de izquierda a derecha hasta llenar la fila y continúan agregándose en las siguientes filas de la misma manera.




```
import javax.swing.*;
import java.awt.*;
public class VentanaPrincipal {
   public static void main(String[] args){
       JFrame vGrid = new JFrame();
       vGrid.setSize(500,300);
       vGrid.setLayout(new GridLayout(2,3));
       for(int i=1; i<=5; i++){
          vGrid.add(new JButton("Botón "+ i));
       }
       vGrid.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       vGrid.setVisible(true);
    }
}
```

GridBagLayout

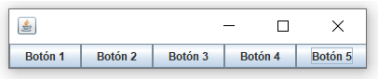
GridBagLayout es el gestor de esquema más complejo y flexible de los gestores. Aunque puede pensarse que este gestor es una subclase del gestor GridLayout no lo es y se comporta diferentemente. Mientras que con GridLayout todos los elementos son colocados en celdas de igual tamaño, con GridBagLayout los componentes pueden tener diferentes tamaños y pueden ocupar múltiples filas y columnas. La posición y el comportamiento de cada elemento se especifica por una instancia de la clase GridBagConstraints.

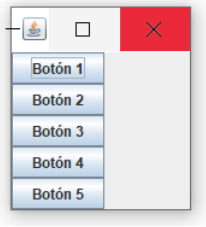



```
public class VentanaPrincipal {
   public static void main(String[] args){
       Insets inset = new Insets(0,0,0,0);
       JFrame vGridBag = new JFrame();
       vGridBag.setSize(500,500);
       vGridBag.setLayout(new GridBagLayout());
       vGridBag.add(new JButton("Botón 1"),new GridBagConstraints(0,0,1,1,1.0,1.0,
              GridBagConstraints.CENTER, GridBagConstraints.BOTH, inset,0,0));
       vGridBag.add(new JButton("Botón 2"), new GridBagConstraints(1,0,1,1,1.0,1.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH, inset,0,0));
       vGridBag.add(new JButton("Botón 3"), new GridBagConstraints(0,1,2,1,1.0,1.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH, inset,0,0));
       vGridBag.add(new JButton("Botón 4"), new GridBagConstraints(0,2,1,2,1.0,1.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH, inset,0,0));
       vGridBag.add(new JButton("Botón 5"), new GridBagConstraints(1,2,1,1,1.0,1.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH, inset,0,0));
       vGridBag.add(new JButton("Botón 6"), new GridBagConstraints(1,3,1,1,1.0,1.0,
                  GridBagConstraints.CENTER, GridBagConstraints.BOTH, inset,0,0));
       vGridBag.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       vGridBag.setVisible(true);
   }
}
```

BoxLayout

El gestor de esquema de diseño BoxLayout permite colocar los componentes alineados en un eje horizontal (BoxLayout. X_AXIS) o vertical (BoxLayout.Y_AXIS). Además, se puede seleccionar la alineación que tendrán estos componentes respecto al eje seleccionado.

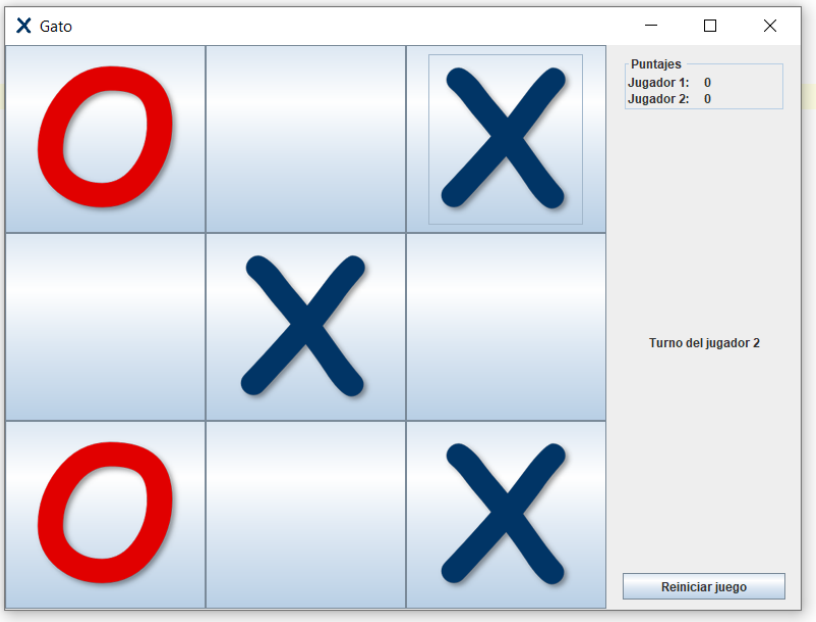




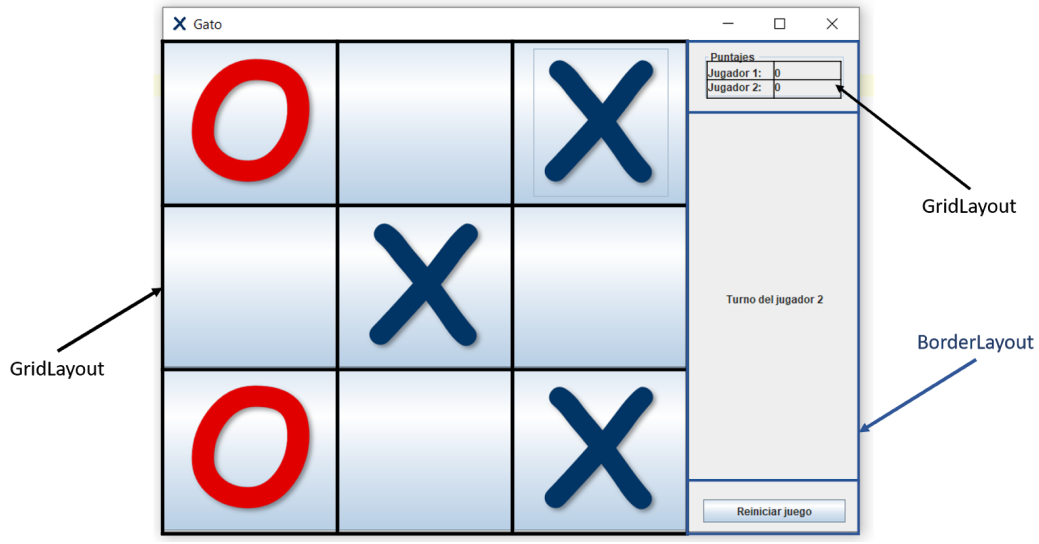

```
public class VentanaPrincipal {
   public static void main(String[] args){
       JFrame vBox = new JFrame();
       vBox.setSize(200, 500);
       vBox.setLayout(new BoxLayout(vBox.getContentPane(),
                                    BoxLayout.Y_AXIS));
       for(int i=1; i<=5; i++){
          vBox.add(new JButton("Botón "+ i));
       }
       vBox.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       vBox.setVisible(true);
   }
}
```

Diseños más complejos. Paneles e íconos.

Desarrollar interfaces gráficas con acomodos más complejos requiere la combinación de varios manejadores de esquema. Para poner en práctica los contenidos revisados en este documento se desarrollará el juego de “Gato” con que tendrá una GUI como la de la siguiente imagen:



En la imagen se puede ver que existe una cuadrícula en la sección del tablero del juego la cual se puede lograr con un gestor del tipo GridLayout y nueve botones. Del lado derecho de la interfaz se cuenta con un recuadro para los puntajes, una etiqueta para indicar el jugador en turno y un botón para reiniciar el juego. La disposición de los elementos de esta sección se puede conseguir con un esquema BorderLayout. Finalmente, las etiquetas dentro del cuadro de puntajes se pueden ordenar con un esquema GridLayout.



Para comenzar con el desarrollo del juego se creará una clase que modele la apariencia del botón con el símbolo de cruz o círculo en la pantalla. Esta clase será una subclase de la clase JButton y se llamará Celda.

Archivo Celda.java

```
import javax.swing.*;

public class Celda extends JButton {
   public static final int CIRCULO = 0;
   public static final int CRUZ = 1;
   public static final int BLANCO = 2;
   public static ImageIcon[] iconos = new ImageIcon[3];

   private int valor;

   public Celda() {
       valor = BLANCO;
       iconos[CIRCULO] = new ImageIcon("img/O.png");
       iconos[CRUZ] = new ImageIcon("img/X.png");
       iconos[BLANCO] = null;
   }

   public void setValor(int valor){
       this.valor = valor;
       setIcon(iconos[valor]);
   }

   public int getValor(){
       return valor;
   }

   public boolean estaLibre(){
       return getIcon() == null;
   }

   public void reiniciar(){
       setValor(BLANCO);
   }
}
```

La clase ImageIcon permite cargar una imagen en forma de ícono para incluirlo en un componente como un botón.

A continuación se incluye el código de la interfaz gráfica de la ventana principal y de la lógica del juego. La clase principal es una subclase de la clase JFrame y tiene dos clases anidadas (o internas) que se encargarán de manejar los eventos de los botones del tablero y del botón de reinicio. En el constructor de la clase se crean objetos de tipo JPanel que sirven para agrupar componentes y tener diferentes tipos de layout. Los paneles *cuadrícula* y *puntajes* tienen un esquema GridLayout mientras que el panel *controles* tiene un esquema BorderLayout.

Archivo Tablero.java

```
import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.*;

public class Tablero extends JFrame {
   public final int GANA_CIRCULO = 1;
   public final int GANA_CRUZ = 2;
   public final int EMPATE = 3;
   public final int EN_PROGRESO = 0;

   private int estado;
   private boolean turnoCruz;
   private int puntosCruz;
   private int puntosCirculo;
   private JPanel cuadricula;
   private JPanel controles;
   private JLabel etP1;
   private JLabel etP2;
   private JLabel etEstado;

   public static void main(String[] args){
       Tablero tab = new Tablero();
   }

   Tablero(){
       super("Gato");
       setSize(800,600);
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       setLayout(new BorderLayout());
       setIconImage(new ImageIcon("img/X.png").getImage());
       cuadricula = new JPanel();
       cuadricula.setLayout(new GridLayout(3,3));
       for(int i=1; i<=9; i++){
           Celda cuadro = new Celda();
           cuadro.addActionListener(new ManejadorCelda());
           cuadricula.add(cuadro);
       }
       add(cuadricula, BorderLayout.CENTER);
       controles = new JPanel();
       controles.setLayout(new BorderLayout());
       controles.setBorder(new EmptyBorder(10,15,10,15));
       JPanel puntajes = new JPanel();
       puntajes.setLayout(new GridLayout(2,2));
       puntajes.setBorder(BorderFactory.createTitledBorder("Puntajes "));
       JLabel etJ1 = new JLabel("Jugador 1:     ");
       JLabel etJ2 = new JLabel("Jugador 2:     ");
       etP1 = new JLabel("0");
       etP2 = new JLabel("0");
```

La clase BorderFactory permite añadirle bordes a los paneles. Un borde de tipo titledBorder le añade un marco y un título al panel.

```
        puntajes.add(etJ1);
        puntajes.add(etP1);
        puntajes.add(etJ2);
        puntajes.add(etP2);
        etEstado = new JLabel("");
        etEstado.setHorizontalAlignment(JLabel.CENTER);
        controles.add(etEstado, BorderLayout.CENTER);
        controles.add(puntajes, BorderLayout.NORTH);
        JButton botonReset = new JButton("Reiniciar juego");
        botonReset.addActionListener(new ManejadorReset());
        controles.add(botonReset, BorderLayout.SOUTH);
        add(controles, BorderLayout.EAST);

        turnoCruz = true;
        puntosCirculo = 0;
        puntosCruz = 0;
        estado = EN_PROGRESO;
        actualizaGUI();
        setVisible(true);
    }

    public void actualizaEstado(){
        Component[] c = cuadricula.getComponents();
        Celda[] celdas = new Celda[9];
        for(int i=0; i<9; i++)
            celdas[i] = (Celda) c[i];
        int[][] combs = {{1,2,3}, {4,5,6},{7,8,9},{1,4,7},{2,5,8},{3,6,9},{1,5,9},{3,5,7}};
        int i=0, suma=0;
        do{
            int res = verificaLinea(combs[i], celdas);
            if(res == 1){
                estado=(celdas[combs[i][0]-1].getValor()==Celda.CIRCULO)? GANA_CIRCULO:GANA_CRUZ;
            }else if(res == 0){
                estado = EN_PROGRESO;
            }else
                suma++;
            i++;
        }while(i<8 && estado != GANA_CIRCULO && estado!=GANA_CRUZ );
        if(suma==8)
            estado = EMPATE;
    }

    public void actualizaGUI(){
        if(estado == GANA_CRUZ || estado == GANA_CIRCULO || estado == EMPATE) {
            String texto = "Ganó Jugador ";
            if(estado==GANA_CRUZ){
                puntosCruz++;
                texto+= "1";
            }else if(estado==GANA_CIRCULO){
                puntosCirculo++;
                texto+= "2";
            }else
                texto = "Hubo un empate";
            etEstado.setText(texto);
            int res = JOptionPane.showConfirmDialog(this, texto + "¿Volver a jugar?",
                    "Fin del juego",JOptionPane.YES_NO_OPTION);
            if(res == JOptionPane.YES_OPTION) {
                limpiaTablero();
                turnoCruz = true;
            }else
                dispose();
```

```
        }else
            etEstado.setText("Turno del jugador "+((turnoCruz)? "1": "2"));
        etP1.setText(String.valueOf(puntosCruz));
        etP2.setText(String.valueOf(puntosCirculo));
    }

    private int verificaLinea(int[] combs, Celda[] celdas){
        int i=0;
        boolean blanco = false;
        do{
            blanco = celdas[combs[i]-1].getValor() == Celda.BLANCO;
        }while(++i<3 && !blanco);

        if(!blanco){
            if(celdas[combs[0]-1].getValor() == celdas[combs[1]-1].getValor()&&
                celdas[combs[1]-1].getValor() == celdas[combs[2]-1].getValor())
                return 1;
            else
                return -1;
        }else
            return 0;
    }

    private void limpiaTablero(){
        Component[] componentes = cuadricula.getComponents();
        for(Component componente : componentes){
            Celda celda = (Celda) componente;
            celda.reiniciar();
        }
        etEstado.setText("Turno del jugador 1");
    }

    class ManejadorCelda implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
            Celda boton = (Celda) e.getSource();
            if(boton.estaLibre()) {
                if(turnoCruz)
                    boton.setValor(Celda.CRUZ);
                else
                    boton.setValor(Celda.CIRCULO);
                turnoCruz = !turnoCruz;
                actualizaEstado();
                actualizaGUI();
            }
        }
    }

    class ManejadorReset implements ActionListener{
        @Override
        public void actionPerformed(ActionEvent e) {
            puntosCirculo = 0;
            puntosCruz = 0;
            turnoCruz = true;
            estado = EN_PROGRESO;
            limpiaTablero();
            actualizaGUI();
        }
    }
}
```

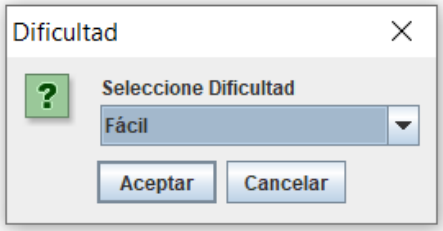
Desarrollo

En esta práctica aplicarás los conceptos revisados en la introducción a través de la realización de una de las dos actividades que se presentan a continuación. En cualquiera de las dos actividades utilizarás los componentes de Swing y los gestores de esquemas de diseño para desarrollar una GUI que permita al usuario jugar un juego sencillo. Para la interacción del usuario con la GUI implementarás algunos manejadores de eventos y programarás la lógica del juego siguiendo los paradigmas de programación dirigida por eventos y orientada a objetos. La actividad que selecciones la podrás realizar junto con un compañero.

Actividad A. Buscaminas.

En esta actividad programarás el juego llamado Buscaminas. Este juego consiste en destapar celdas que no contengan una mina (o bomba). Al destapar una celda se muestra el número de minas que se encuentran alrededor de la celda. Así, si una celda muestra un 4 quiere decir que de las 8 celdas circundantes, 4 tienen una mina.

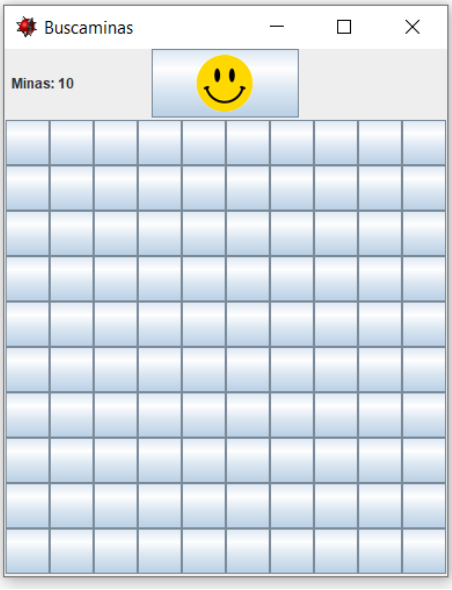
Requisitos de diseño de la interfaz gráfica



Al iniciar la aplicación deberá aparecer una ventana de diálogo (inputDialog) que solicite al usuario seleccionar el nivel de dificultad del juego.

Posteriormente, se deberá mostrar una ventana con un tablero que variará de tamaño dependiendo de la dificultad:

- Fácil: tablero de 10x10 casillas.
- Medio: tablero de 16x16 casillas.
- Difícil: tablero de 20x20 casillas.



La ventana se deberá tener una dimensión diferente de acuerdo con la dificultad seleccionada, con la finalidad de que las celdas se desplieguen con forma cuadrada.

En la parte superior de la ventana se deberá mostrar un texto que indique la cantidad de minas que están ocultas en el tablero, la cual dependerá de la dificultad seleccionada:

- Fácil: 10 minas.
- Medio: 40 minas.
- Difícil: 80 minas.

Además, en la parte superior central se encontrará un botón con una imagen de una carita feliz para reiniciar el juego.

La ventana de la interfaz gráfica deberá tener un título y un ícono distintivo del juego.

Cuando se presione una de las celdas se deberá colorear ésta de un color gris si no se encontrará una mina en esa ubicación y se desplegará el número de minas alrededor de esa celda. En cambio, si se encuentra una mina, se mostrará un ícono similar a la imagen de una bomba y sucederá lo mismo en todas las celdas donde se encuentre una mina.

En la construcción de la interfaz se deberán utilizar componentes de Swing y al menos dos gestores de layout diferentes. Cuando se presione el botón “X” para cerrar la ventana el programa debe terminar.

Requisitos de desarrollo del proyecto

El proyecto se deberá desarrollar siguiendo los fundamentos del paradigma de programación orientada a objetos. En él deberá existir una clase llamada Celda que modele una de las casillas del juego y que además sea una subclase de algún componente de Swing (puede ser un JButton).

Además, se deberá programar la asignación de las minas de manera aleatoria y se deberá calcular la cantidad de minas alrededor de una casilla. En esta actividad no deberá incluirse la característica de marcado de casillas (símbolo de bandera) que existe en otras versiones del juego.

Para llevar a cabo la interacción entre la GUI y el usuario se deberá utilizar el paradigma de programación dirigida por eventos. En este proyecto se deberá programar al menos una clase manejadora de eventos que implemente la interfaz ActionListener.

Actividad B. Batalla Naval.

Esta actividad consiste en desarrollar una versión del juego Batalla Naval en Java. El juego es para dos jugadores y el objetivo es descubrir las naves que ocultó el jugador oponente en un tablero cuadriculado. A partir del “destape” de las casillas, el jugador puede ir descubriendo las naves (que pueden ocupar una o más casillas). El jugador que primero “hunda” (descubra) todas las naves del oponente es el ganador.

En esta versión del juego, la computadora colocará las naves de ambos jugadores. Las naves con las que contará cada jugador y el tamaño de cada uno son los siguientes:

- 1 Portaviones (4 cuadros)
- 1 Submarino (3 cuadros)
- 2 Destructores (2 cuadros)
- 3 Fragata (1 cuadro)

Las naves podrán ubicarse en cualquier parte del tablero, sin empalmarse, en orientación vertical u horizontal.

Requisitos de diseño de la interfaz gráfica

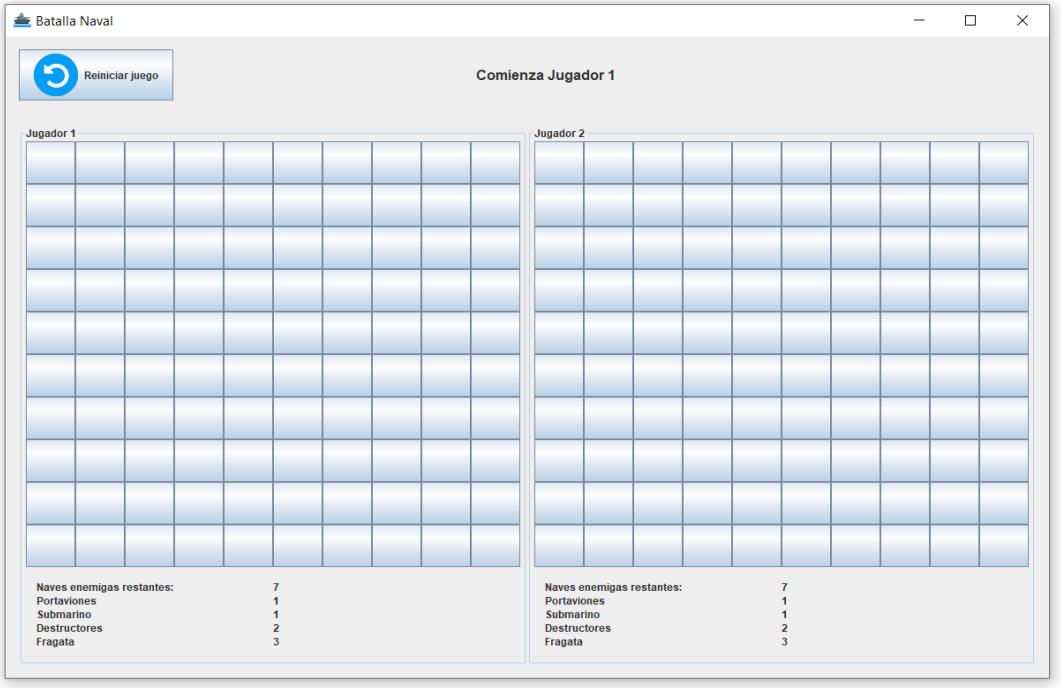
La ventana de la interfaz gráfica deberá tener un título y un ícono distintivo del juego. Además, en la parte superior de ésta se deberá mostrar un botón con un ícono y el texto “Reiniciar juego” que permita restablecer el juego. En el centro de esa sección se deberán mostrar mensajes que indiquen a qué jugador le corresponde el turno.

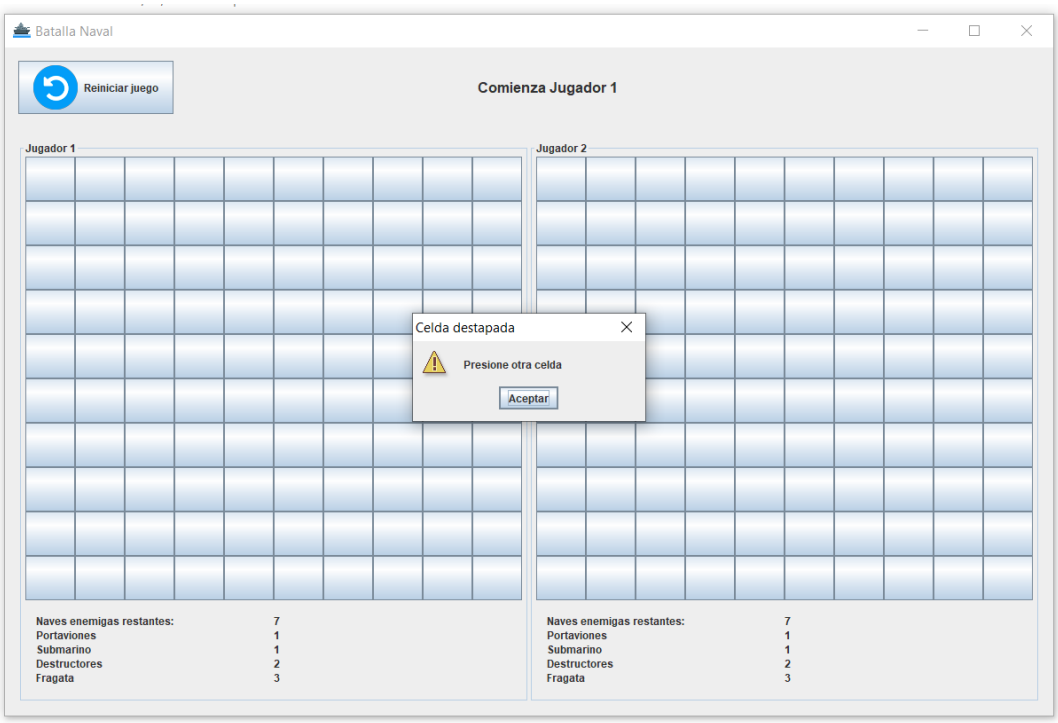
El área de batalla (tablero) de cada jugador será una cuadrícula de 10 x 10 celdas. El jugador, en su turno, podrá dar clic en la celda para tratar de descubrir una nave. Si en la celda no se encuentra una nave, el botón se debe colorear de color azul cielo.

En el caso de que el usuario haya seleccionado una celda que contiene una nave enemiga, dicha celda deberá cambiar de color a un color que identifique que la nave se encuentra “herida” (amarillo, por ejemplo). Si la nave fue hundida se deberán colorear todas las celdas de la nave de un color diferente al utilizado en el caso anterior. Cuando el usuario presione una celda que ya había presionado anteriormente le deberá aparecer al usuario una ventana de diálogo solicitándole que presione otra celda.

Debajo del tablero de cada jugador se deberán desplegar estadísticas para el jugador. Se deberá mostrar, por lo menos, las cantidades de naves enemigas que falta por descubrir, agrupadas por cada tipo de nave.

En la construcción de la interfaz se deberán utilizar componentes de Swing y al menos dos gestores de layout diferentes. Cuando se presione el botón “X” para cerrar la ventana el programa debe terminar.





Requisitos de desarrollo del proyecto

El proyecto se deberá desarrollar siguiendo los fundamentos del paradigma de programación orientada a objetos. Se deberá crear una clase llamada Celda que sea subclase de algún componente de Swing (se recomienda heredar de la clase JButton).

También, se deberá programar un algoritmo que permita asignar la posición de cada una de las naves de manera aleatoria. Las naves no se pueden empalmar y se pueden encontrar tanto en posición horizontal como vertical.

Para llevar a cabo la interacción entre la GUI y el usuario se deberá utilizar el paradigma de programación dirigida por eventos. En este proyecto se deberá programar al menos una clase manejadora de eventos que implemente la interfaz ActionListener.

Entrega de la práctica

La solución al problema de esta práctica deberá entregarse en el horario de clases o en alguna de las horas de asesoría. En la entrega se verificará el cumplimiento de los siguientes puntos:

1. Solución del problema mediante los paradigmas de programación orientado a objetos y programación dirigida por eventos.
2. Cumplimiento de los requisitos de diseño de la interfaz gráfica.
3. Correcto funcionamiento de la solución de acuerdo con el problema planteado.
4. Las clases deben contener los siguientes elementos:
    4.1. Modificadores de acceso tanto en la definición de la clase como en los miembros de ésta.
    4.2. Nombres de los atributos y métodos que cumplan con las convenciones de nombres de Java.
    4.3. Anotación @Override cuando se trate de la sobrescritura de un método.
5. Buen manejo de excepciones (si es que aplica).
6. Buenas técnicas de programación (indentación, nombres descriptivos, uso de estructuras de control).

Referencias

Bush, Vannevar. (julio de 1945). As we may think. *The Atlantic*. p. 101. Recuperado de: https://www.theatlantic.com/magazine/archive/1945/07/as-we-may-think/303881/

Deitel, P. y Deitel, H. (2012). *Java: How to program*. Boston: Pearson-Prentice Hall.

Evans, B. y Flanagan, D. (2015). *Java in a nutshell.* Sebastopol, California: O'Reilly.

Ko, Amy J. y Whitmire, Eric. (s.f.). User Interface Software and Technology. Libro en línea disponible en: https://faculty.washington.edu/ajko/books/uist/index.html

Levy, Steven. (29 de marzo de 2018). Graphical User Interface. Enciclopædia Britannica. Recuperado de: https://www.britannica.com/technology/graphical-user-interface

Liguori, R. y Liguori, R. (2014). *Java 8 Pocket Guide*. Sebastopol, California: O'Reilly.

Niemeyer, P. y Leuck, D. (2013). *Learning Java*. Sebastopol, California: O'Reilly.

Wu, T. (2010). *An introduction to Object-Oriented Programming with Java*. Nueva York: McGraw Hill.