Ambientes gráficos en Java

Ambientes gráficos (Swing)

Elizabeth León Guzmán, Ph.D.
eleonguz@unal.edu.co

Jonatan Gómez Perdomo, Ph. D.
jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.
aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)
eccubidesg@unal.edu.co

Carlos Andres Sierra, M.Sc.
casierrav@unal.edu.co

Research Group on Data Mining – Grupo de Investigación en Minería de Datos – (Midas)
Research Group on Artificial Life – Grupo de Investigación en Vida Artificial – (Alife)
Computer and System Department
Engineering School
Universidad Nacional de Colombia

Agenda

1. Swing
2. Algunas componentes de Swing
3. Práctica de clase
   - Agregar un botón para detener la ejecución de la animación
   - Agregar un campo de texto para definir la carga del platón

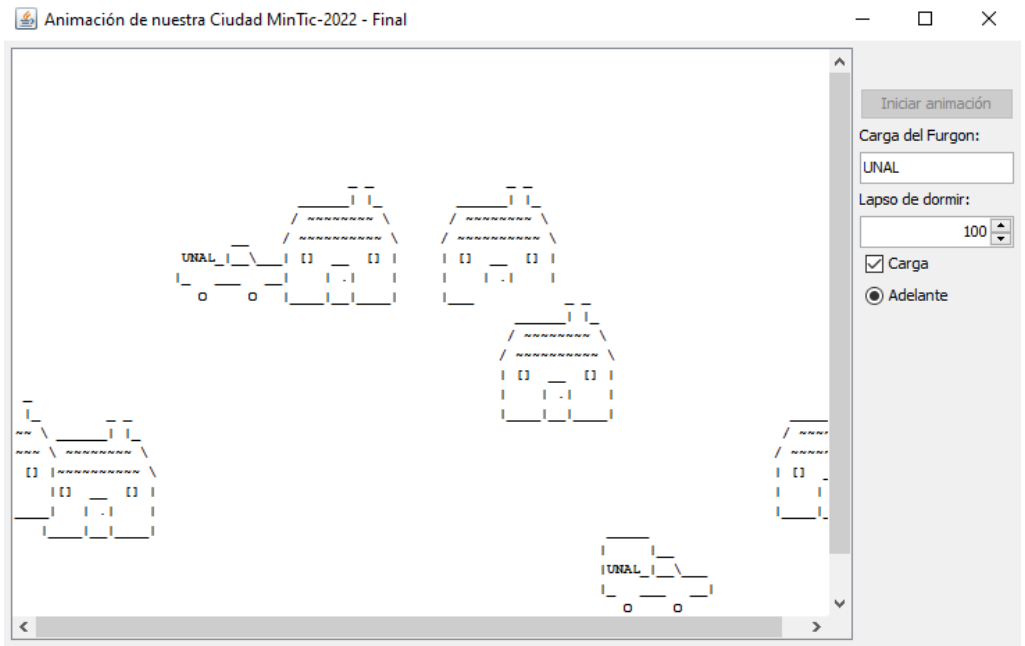
MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Interfaces gráficas con Swing

Definición

Swing es un conjunto de clases (librería) que permite construir en Java interfaces gráficas independientes del sistema operativo, gracias a su modelo vista/controlador de componentes flexibles.



https://github.com/jgomezpe/carros/tree/master/swingcity

Agenda

1. Swing
2. Algunas componentes de Swing
3. Práctica de clase
   - Agregar un botón para detener la ejecución de la animación
   - Agregar un campo de texto para definir la carga del platón

MIDAS

Marcos en Swing (JFrame) I

Definición (JFrame)

Es un contenedor donde colocar componentes. La clase `JFrame` se encuentra en el paquete `javax.swing`. La mayoría de las ventanas son instancias de la clase `JFrame` o subclases de `JFrame`. `JFrame` proporciona los atributos y comportamientos básicos de una ventana

Constructores y métodos:

`JFrame()`: Es el constructor de nuestra ventana, aparece sin título y no puede verse en pantalla hasta que se llame el método para hacerlo visible.

`setVisible(boolean b)`: Define si nuestro `JFrame` puede verse o no y únicamente acepta valores booleanos como `true` o `false`.

Marcos en Swing (`JFrame`) II

`setTitle(String str)`: Define el nombre del `JFrame` (el título), y lo que acepta es una cadena para nombrarla.

`setSize(int ancho, int largo)`: Es la definición del ancho y largo del `JFrame` que vamos a crear.

`pack()`: Redefine la ventana para que entre en un espacio determinado.

`setLocation(int horizontal, int vertical)`: Definición de la posición en pantalla donde va a estar nuestro `JFrame` una vez que lo creemos.

`setDefaultCloseOperation(int operation)`: Define la operación que va a realizar la ventana cuando se oprime el botón de cerrar.

Véase y analice la clase `Ventana`

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Paneles en Swing (JPanel) I

Definición (JPanel)

Los paneles en Java son objetos contenedores, la finalidad de estos objetos es la agrupación de otros objetos tales como botones, campos de texto, etiquetas, selectores, etc; una gran ventaja de usar `JPanel` en Java es que podemos manejar la agrupación de una mejor forma, supongamos que tenemos una serie de botones en un panel, y deseamos desactivarlos todos a las vez, en lugar de hacerlo individualmente con los botones, podemos desactivar el panel y con esto los botones.

Constructores y métodos:

- `JPanel()`: Crea un panel vacío.

Paneles en Swing (`JPanel`) II

`JPanel(Layout gestor)`: Crea un `JPanel` con el gestor de organización proporcionado. `Layout` es un gestor de organización, es decir, es el que determina como van a estar distribuidos los componentes en el panel, dentro de estos distribuidores esta el `FlowLayout`, el `BorderLayout`, el `BoxLayout`, el `GridLayoud`, etc.

`add(Component componente)`: Adiciona un componente al panel.

Véase y analice las clases `PanelCiudad` y `PanelControles`

MIDAS

Áreas de texto en Swing (JTextArea) I

Definición

Las áreas de texto son un sencillo editor multilíneas que utiliza un sólo tipo de fuente.

Constructores y métodos:

`JTextArea()`: Crea un campo de texto por defecto.

`JTextArea(int numLines, int numChars)`: Permite crear un área de texto con el alto y ancho definidos.

`JTextArea(String str)`: Permite crea un área de texto con un texto inicial especificado.

`JTextArea(String str, int numLines, int numChars)`: Este método permite crear un área de texto con el alto y ancho definidos y además con un texto inicial especifico.

Áreas de texto en Swing (`JTextArea`) II

`void setText(String str)`: Este método permite establecer un determinado texto.

`String getText()`: Con este método se puede obtener la cadena del área de texto.

`String getSelectedText()`: Con este método puede obtenerse el texto actualmente seleccionado.

`void select(int startIndex, int endIndex)`: Con este método se puede seleccionar una parte del texto del área de texto creado.

`void setEditable(boolean b)`: Este método permite controlar si el contenido de área de texto puede ser modificado por el usuario, `true` para que el texto se pueda cambiar y `false` para que no se pueda editar.

Áreas de texto en Swing (`JTextArea`) III

`boolean isEditable()`: Este método permite determinar si un área de texto es editable o no, retorna `true` si es editable y `false` si no lo es.

`void append(String str)`: Este método permite añadir una cadena especifica al final del texto actual.

`void insert(String str, int index)`: Este método permite insertar una cadena de caracteres en un punto especificado.

`void replaceRange(String str, int startIndex, int endIndex)`: Este método permite reemplazar los caracteres dados en `str` desde `startIndex` hasta `endIndex`.

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Etiquetas en Swing (JLabel) I

Definición

Las etiquetas en Java permiten agregar texto o imágenes en una ventana que sólo pueden ser editados por funciones del programa.

Constructores y métodos:

`JLabel()`: Crea una etiqueta vacía.

`JLabel(String str)`: Crea una etiqueta con el `String str` justificado a la izquierda.

`JLabel(String str, int alignment)`: Crea una etiqueta con el `String str`, justificado con `alignment`. Los valores de `alignment` son: `JLabel.LEFT`, `JLabel.RIGHT` y `JLabel.CENTER`.

Etiquetas en Swing (JLabel) II

`void setText(String str)`: Cambia el texto de la etiqueta por el `String str`.

`String getText()`: Retorna el texto en la etiqueta.

`void setAlignment(int alignment)`: cambia la justificación de la etiqueta según el velor de `alignment`.

`int getAlignment()`: Retorna la justificación actual del objeto.

Botones en Swing (JButton) I

Definición

Los botones en Java sirven como mecanismo de interacción entre el usuario y el programa por medio de la interfaz gráfica que representan, ya que pueden realizar tareas programadas, verificar condiciones o seleccionar elementos en grupos específicos.

Botones en Swing (`JButton`) II

Constructores y métodos:

`JButton()`: Crea un botón sin etiqueta.

`JButton(String str)`: Crea un botón con la etiqueta `str`.

`void setLabel(String str)`: Modifica la etiqueta del botón.

`String getLabel()`: Retorna la etiqueta asociada al botón.

`addActionListener(ActionListener listener)`: Añade un auditor al botón, es decir, `ActionListener`, escucha los eventos. Cuando se da clic al botón, el suceso será gestionado por el `ActionListener listener`.

Campos de texto en Swing (`JTextField`) I

Definición

Permite usar un campo de texto de una sola línea, que es editable. Los campos de texto permiten al usuario introducir cadenas y editar texto utilizando los cursores, las teclas de cortar y pegar, y las selecciones que se hacen con el ratón.

Constructores y métodos:

`JTextField()`: Crea un campo de texto por omisión.

`JTextField(int numChars)`: Crea un campo de texto con tamaño definido por `n` caracteres.

`JTextField(String str)`: Crea un campo de texto inicializado con la cadena de caracteres dada `String str`.

Campos de texto en Swing (`JTextField`) II

`JTextField(String str, int numChars)`: Crea un campo de texto inicializado con la cadena de caracteres y además especifica su tamaño.

`void setText (String str)`: Este método permite establecer un determinado texto.

`String getText()`: Con este método se puede obtener la cadena del campo de texto.

`String getSelectedText()`: Con este método puede obtenerse el texto actualmente seleccionado.

`void select(int startIndex, int endIndex)`: Con este método se puede seleccionar una parte del texto del campo de texto creado.

Campos de texto en Swing (`JTextField`) III

`void setEditable(boolean b)`: Este método permite controlar si el contenido de un campo de texto puede ser modificado por el usuario, `true` para que el texto se pueda cambiar y `false` para que no se pueda editar.

`boolean isEditable()`: Este método permite determinar si un campo de texto es editable o no, retorna `true` si es editable y `false` si no lo es.

`void setEchoChar(char ch)`: Este método permite que el texto que se introduce no sea visible, como cuando se introduce una clave de acceso.

`boolean echoCharIsSet()`: Este método permite revisar si el campo de texto está en modo ocultar caracteres.

`char getEchoChar()`: Este método permite obtener los caracteres que están ocultos.

Cajas de chequeo en Swing (JCheckBox) I

Definición

Las cajas de chequeo permiten que se ejecute un procedimiento dependiendo de la verificación de la caja, es decir, si está o no activo. También permite al usuario hacer selecciones múltiples de un conjunto de opciones. Por convención, se usan cajas de chequeo para crear listas abiertas en las que el usuario puede escoger varias opciones.

Constructores y métodos:

`JCheckBox()`: Crea un botón `JCheckBox` sin seleccionar, sin texto y sin icono.

`JCheckBox(String str)`: Crea un `JCheckBox` inicialmente deseleccionado con el texto `str`.

Cajas de chequeo en Swing (JCheckBox) II

`JCheckBox(String str, boolean b)`: Crea un `JCheckBox` con el texto `str` y especifica si está inicialmente seleccionado o no, mediante el parámetro `b`.

`setSelected(boolean b)`: Permite establecer si una caja de chequeo debe seleccionarse o no.

`isSelected()`: Permite determinar si una caja de chequeo fue seleccionada o no.

`String getText()`: Obtiene el texto de la caja de chequeo y lo utiliza para definir el texto dentro de la etiqueta.

`setText(String str)`: Establece el texto `str` de la etiqueta de la caja de chequeo.

Botones de opción o selección en Swing (`JRadioButton`) I

Definición

Los botones de opción o selección a diferencia de las cajas de chequeo en donde pueden seleccionarse múltiples opciones, estos botones se usan en listas que sólo permitan una opción activa.

Constructores y métodos:

`JRadioButton(String str)`: Crea un botón de selección con el nombre del texto `str` proporcionado

`JRadioButton(String str, boolean b)`: Crea un botón de selección con el nombre del texto `str` proporcionado y se marca como seleccionado dependiendo del valor del booleano `b`.

Botones de selección en Swing (`JRadioButton`) II

`setSelected(boolean b)`: Permite establecer si un botón de selección debe seleccionarse o no.

`isSelected()`: Permite determinar si un botón de selección fue seleccionado o no.

`String getText()`: Obtiene el texto del botón de selección y lo utiliza para definir el texto dentro de la etiqueta.

`setText(String str)`: Establece el texto `str` de la etiqueta del botón de selección.

Grupo de botones en Swing (`ButtonGroup`)

Definición

Es la clase que permite agrupar de forma correcta un conjunto de botones de selección, para asegurar que únicamente se escogerá una opción.

Constructores y métodos:

`ButtonGroup()`: Crea un objeto que puede considerarse como un arreglo de botones.

`add(button)`: Agrega botones al arreglo, de tal manera que soló se pueda escoger un botón de selección del grupo de botones adicionados.

MIDAS

Tablas en Swing (JTable) I

Definición

Un JTable es una componente visual de Java que permite dibujar una tabla mostrando un conjunto de datos en forma de filas y columnas. Los datos de la tabla se encuentran empaquetados como objetos.

Constructores y métodos:

`JTable(Object[][] data, Object[] names)`: donde `data` es un arreglo bidimensional que contiene la información de las celdas de la tabla y `names` es un arreglo unidimensional que contiene los nombres de las columnas.

`Object getValueAt(int row, int column)`: Retorna el valor (objeto) almacenado en la celda de la fila `row` y columna `column`.

`void setValueAt(Object value, int row, int column)`: Establece el valor `value` en la celda de la fila `row` y columna `column`.

Tablas en Swing (JTable) II

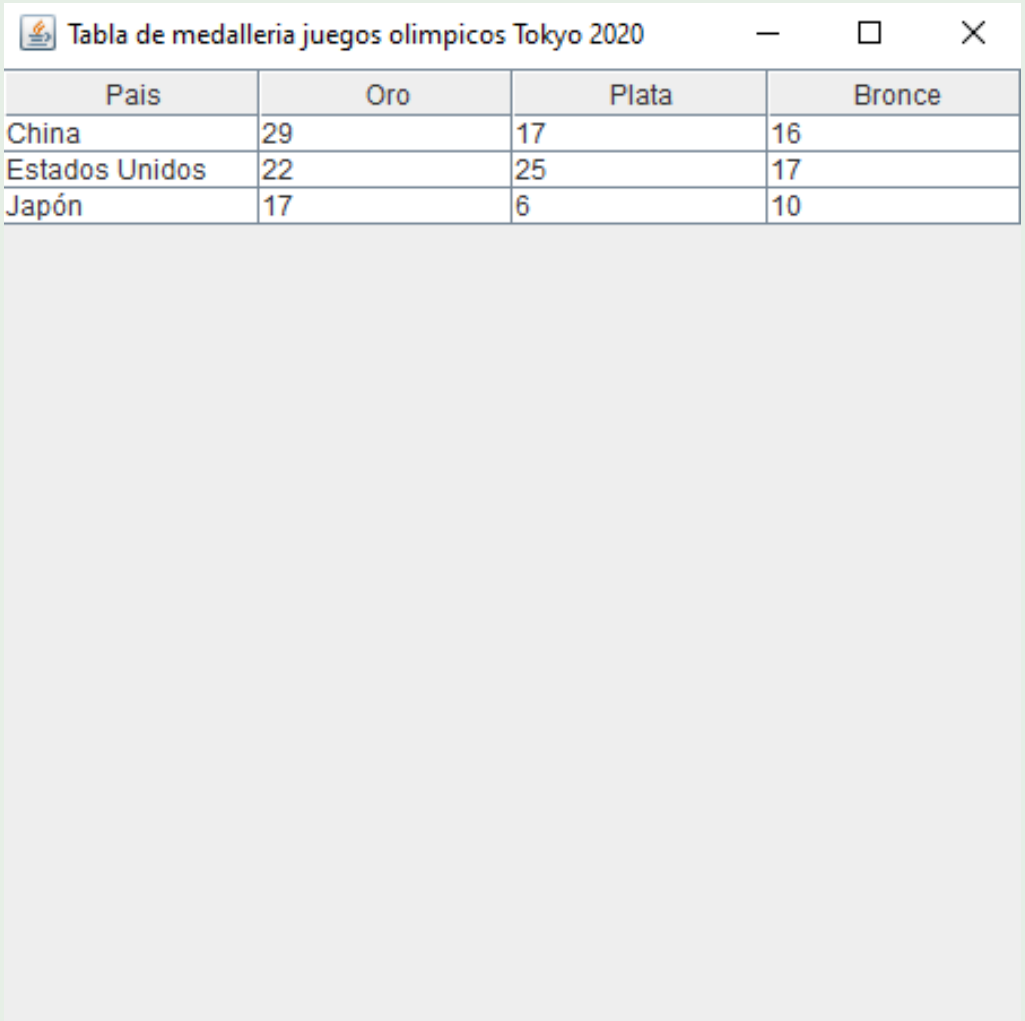
Ejemplo

```
import javax.swing.*;

public class EjemploTabla extends JFrame {
    public EjemploTabla() {
        setTitle("Tabla de medalleria juegos olimpicos Tokyo 2020");
        String[] encabezados = {"Pais", "Oro", "Plata", "Bronce"};
        String[][] valores = {
            {"China", "29", "17","16"},
            {"Estados Unidos", "22", "25", "17"},
            {"Japón", "17", "6", "10"}
        };
        JTable table = new JTable(valores, encabezados);
        JScrollPane jsp = new JScrollPane(table);
        add(jsp);
        pack();
        setVisible(true);
    }
    public static void main(String[] args) {
        new EjemploTabla();
    }
}
```

Tablas en Swing (`JTable`) III

Ejemplo (continuación)



Pais	Oro	Plata	Bronce
China	29	17	16
Estados Unidos	22	25	17
Japón	17	6	10

Agenda

Agenda

1. Swing
2. Algunas componentes de Swing
3. Práctica de clase
   - Agregar un botón para detener la ejecución de la animación
   - Agregar un campo de texto para definir la carga del platón

MIDAS

Agregar un botón para detener la ejecución (I)

1. Declarar un atributo privado botón en la clase `PanelControles` mediante la siguiente instrucción

```
private JButton jButtonParar;
```

2. Definir y agregar un botón en el método `initComponents()` de la clase `PanelControles`, después del botón `jButtonIniciar`, así

```
jButtonIniciar = new JButton("Iniciar animación");
add(jButtonIniciar);

jButtonParar = new JButton("Parar animación");
add(jButtonParar);
```

Agregar un botón para detener la ejecución (II)

3. Agregar al botón `jButtonParar` el auditor `eventoClic` después de haber definido éste y de haberlo agregado al botón `jButtonIniciar`

```
EventoClic eventoClic = new EventoClic(this);

jButtonIniciar.addActionListener(eventoClic);
jButtonParar.addActionListener(eventoClic);
```

4. Agregar un método *getter* para el botón `jButtonParar` en la clase `PanelControles`, así

```
public JButton getjButtonParar() {
    return jButtonParar;
}
```

Agregar un botón para detener la ejecución (III)

5. Declarar un atributo privado booleano `parar` en la clase `Hilo` mediante la siguiente instrucción

```
private boolean parar;
```

6. Agregar un método *setter* para el atributo booleano `parar` en la clase `Hilo`, así

```
public void setParar(boolean parar) {
    this.parar = parar;
}
```

Agregar un botón para detener la ejecución (IV)

7. Agregar la instrucción

```
parar = false;
```

antes del último ciclo `for` del método `run()` de la clase `Hilo`, así

```
parar = false;
for (int i = 0; i < ParametrosDibujo.ITERACIONES_CIUDAD; i++) {
     if (panelControles.getjRadioButtonAdelante().isSelected()) {
        ciudad.mover();
     }
     ...
}
```

Agregar un botón para detener la ejecución (V)

8. Agregar el condicional

```
if(parar){
    i = ParametrosDibujo.ITERACIONES_CIUDAD;
}
```

al final del último ciclo `for` del método `run()` de la clase `Hilo`, así

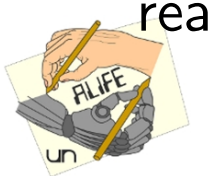
```
parar = false;
for (int i = 0; i < ParametrosDibujo.ITERACIONES_CIUDAD; i++) {
    ...
    ...
    if(parar){
        i = ParametrosDibujo.ITERACIONES_CIUDAD;
    }
}
```

Agregar un botón para detener la ejecución (VI)

9. Agregar un condicional alternativo al condicional del método `actionPerformed(ActionEvent ae)` de la clase `EventoClic`

```
public void actionPerformed(ActionEvent ae) {
    if (ae.getSource() == panelControles.getjButtonIniciar()) {
        hilo = new Hilo(this.panelControles);
        panelControles.getjButtonIniciar().setEnabled(false);
        hilo.start();
    } else if(ae.getSource() == panelControles.getjButtonParar()) {
        if(hilo != null){
            hilo.setParar(true);
            hilo = null;
        }
    }
}
```

10. Compilar, ejecutar el programa, probar y analizar los cambios realizados.



Agenda

MIDAS

UNIVERSIDAD NACIONAL DE COLOMBIA un

Agregar un campo de texto para definir la carga (I)

1. Declarar dos atributos: etiqueta y campo de texto privados en la clase `PanelControles` mediante las siguientes instrucciones

```
private JLabel jLabelPlaton;
private JTextField jTextFieldPlaton;
```

MIDAS

Agregar un campo de texto para definir la carga (II)

2. Definir y agregar la etiqueta `jLabelPlaton` y el campo de texto `jTextFieldPlaton` en el método `initComponents()` de la clase `PanelControles` después del campo de texto `jTextFieldCarga`, así

```
jTextFieldCarga = new JTextField(ParametrosDibujo.CONTENIDO_CARGA);
add(jTextFieldCarga);

jLabelPlaton = new JLabel("Carga del Platon: ");
add(jLabelPlaton);

jTextFieldPlaton = new JTextField(
                Integer.toString(ParametrosDibujo.CONTENIDO_PLATON));
add(jTextFieldPlaton);
```

MIDAS

Agregar un campo de texto para definir la carga (III)

3. Agregar un método *getter* para el campo de texto `jTextFieldPlaton` en la clase `PanelControles`, así

```
public JTextField getjTextFieldPlaton() {
    return jTextFieldPlaton;
}
```

4. En el método `run()` de la clase `Hilo` sobreescribir la invocación del método `llevar` del atributo `camioneta` con la siguiente instrucción

```
camioneta.llevar(Integer.parseInt(
      panelControles.getjTextFieldPlaton().getText()));
```

5. Compilar, ejecutar el programa, probar y analizar los cambios realizados.

Problemas I

Problemas

1. Inserte un botón de chequeo que cuando esté seleccionado permita agregar a la lista de vehículos las variables

```
Automovil automovil = new Automovil(.);
Deportivo deportivo = new Deportivo(.);
```

   y por lo tanto puedan aparecer los vehículos `automóvil` y `deportivo` en la animación, independiente de los vehículos de carga.

2. Inserte un botón de selección que permita establecer si los vehículos se mueven hacia adelante o hacia atrás (sólo una elección es posible). Utilice el método `moverReversa()` de la clase `Ciudad` para alcanzar su objetivo.

MIDAS

Problemas II

Problemas (continuación)

3. Inserte una lista de números (`JSpinner`) que permita modificar la cantidad de iteraciones que realiza la aplicación en cada ejecución de la animación.
4. Inserte una lista de números (`JSpinner`) que permita modificar la cantidad de sitios (casitas) que son insertados dentro de la ciudad.
5. Inserte la instrucción

```
JOptionPane.showMessageDialog(null,
                              "La animación fue detenida");
```

en la versión del condicional alternativo del método `actionPerformed(ActionEvent ae)` de la clase `EventoClic` cuando `hilo != null`, con el fin de que aparezca una ventana emergente que informe que la animación fue detenida anticipadamente.