PROGRAMACIÓN EN JAVA

TEMA 4: Swing (cont.)

Bibliografía

- "JAVA 2 Interfaces gráficas y aplicaciones para Internet – 2ª Edición"
  Fco. Javier Ceballos. Ed. Ra-Ma, 2006
  Capítulo: 2 y parte del 4

Contenido

- Cajas de texto, etiquetas y botones
- Tecla de acceso
- Botón por omisión
- Enfocar un objeto
- Seleccionar el texto de una caja de texto
- Validar interceptando la tecla pulsada
- Ejercicio 1
- Solución al ejercicio 1
- Casillas de verificación
- Ejercicio 2
- Botones de opción
- Ejercicio 3

Cajas de texto, etiquetas y botones

- Los componentes más comunes en una aplicación *Swing* son las cajas de texto, las etiquetas y los botones de pulsación
- Las cajas de texto (componentes *JTextField* o *JTextArea*): Permiten realizar la entrada de datos para una aplicación y visualizar los resultados producidos por la misma
- Las etiquetas (componentes *JLabel*): Son cajas de texto no modificables por el usuario. Informan al usuario de qué tiene que hacer y cuál es la función de cada componente
- Los botones de pulsación (componentes *JButton*): Permite ejecutar una acción cuando sea preciso
- Las clases *JTextField*, *JTextArea*, *JLabel* y *JButton* se derivan directa o indirectamente de la clase *JComponent*

Tecla de acceso

- Aparece la tecla de acceso con el carácter subrayado
- Esto significa que el usuario podrá también ejecutar la acción especificada por un determinado componente, pulsando las teclas *Alt + <letra>*
- Esta asociación tecla-componente recibe el nombre de nemónico y se realiza mediante la siguiente sentencia:

<nombre_componente>.setMnemonic('a');

NOTA: El argumento puede ser el código *ASCII* de la letra o la letra encerrada entre comillas simples

Botón por omisión

- En los botones de las ventanas de nuestro sistema operativo existe generalmente un botón con un borde más resaltado que los demás; se trata del botón por omisión
- Este es el botón que es automáticamente pulsado cuando se presiona la tecla *enter*
- Para informar al panel raíz ¿Cuál será el botón por omisión de entre todos los que contenga?, hay que invocar al método *setDefaultButton*:

```
getRootPane().setDefaultButton(btSaludo);
```

Enfocar un objeto

- Cuando un componente posee el punto de insercción, se dice que está enfocado o que tiene el foco
- ¿Cómo enfocar un determinado componente?

  R = 1. Haciendo clic sobre él

  2. Pulsando la tecla *Tab* una o más veces hasta situar el foco sobre él

  3. Desde la propia aplicación (Invocando al método *requestFocus*)
- Ejemplo:

```
//...
addWindowListener(new WindowAdapter()
{
        public void windowOpened(WindowEvent e)
        {
                frameWindowOpened(e);
        }
        //...
});
//...
public void frameWindowOpened(WindowEvent e)
{
        txtSaludo.requestFocus();
}
//...
```

Enfocar un objeto (cont.)

- También en ocasiones necesitaremos obtener el identificador del componente que tiene el foco
- Para ello debemos invocar al método *getFocusOwner*
- Ejemplo:

```
//Obtener el identificador del componente que tiene el foco
Object jcomp = this.getFocusOwner();
if(jcomp == jTextField1)
//…
```

- El método *getFocusOwner* devuelve el identificador del componente que actualmente está enfocado

Seleccionar el texto de una caja de texto

- En la mayoría de las ocasiones requerimos que el contenido de una caja de texto sea seleccionado automáticamente cuando esta reciba el foco
- ¿Cómo hacemos para que cuando una caja de texto obtenga el foco, todo su contenido sea seleccionado?

  R = Es fácil si sabemos que cuando un componente obtiene el foco genera el evento “foco obtenido” y cuando lo pierde, “foco perdido”

- Ejemplo:

```
//...
jTextField1.addFocusListener(new FocusAdapter()
{
        public void focusGained(WindowEvent e)
        {
                jTextFieldFocusGained(e);
        }
});
//...
public void jTextFieldFocusGained(WindowEvent e)
{
        jTextField1.selectAll();
}
//...
```

Seleccionar el texto de una caja de texto (cont.)

- Otras funciones relacionadas con la selección de texto en un componente de texto son las siguientes:

Método	Descripción
String getSelectedText()	Devuelve el texto seleccionado
int getSelectionEnd()	Devuelve la posición final del texto seleccionado
int getSelectionStart()	Devuelve la posición inicial del texto seleccionado
void select(int pos_inicial, int pos_final)	Selecciona el texto que se encuentra entre las posiciones especificadas
void setSelectionEnd(int pos_final)	Fija la posición final de la selección
void setSelectionStart(int pos_inicial)	Fija la posición inicial de la selección

Validar interceptando la tecla pulsada

- Validar equivale a restringir un contenido según un conjunto de caracteres considerados como validos
- Si la validación de los datos se hace después de pulsar *enter* (evento de acción: *ActionEvent*), podrían existir datos no válidos, pero podrían ser validados antes de utilizarlos
- Si la validación se hace verificando la validez de cada tecla pulsada (evento de pulsación: *KeyEvent*), los datos ya estarán validados una vez finalizada la entrada
- ¿ Qué sucede cuando un componente de texto tiene el foco y el usuario pulsa una tecla?

  R = Se generan tres eventos: *keyPressed* (se pulsa la tecla), *keyTyped* (se escribe el carácter) y *keyReleased* (se suelta la tecla)

Validar interceptando la tecla pulsada (cont.)

- Ejemplo:

```
//...
KeyAdapter objKA = new KeyAdapter()
{
        public void keyPressed(KeyEvent e)
        {
                txtKeyPressed(e);
        }
        public void keyTyped(KeyEvent e)
        {
                txtKeyTyped(e);
        }
        public void keyReleased(e)
        {
                txtKeyReleased(e);
        }
};
jTextField1.addKeyListener(objKA);
jTextField2.addKeyListener(objKA);
//...
```

- Creamos un manejador de eventos del tipo *KeyAdapter* denominado *objKA*
- Asociamos dicho manejador a las cajas de texto para manipular los eventos: *keyPressed*, *keyTyped* y *keyReleased*
- Todos los métodos tienen un argumento *e* de la clase *KeyEvent*. El método *getSource* de esta clase devuelve el identificador del componente que genera el evento, *getKeyCode* devuelve el código asociado con la tecla pulsada y *getKeyChar* el carácter asociado con la tecla pulsada

Validar interceptando la tecla pulsada (cont.)

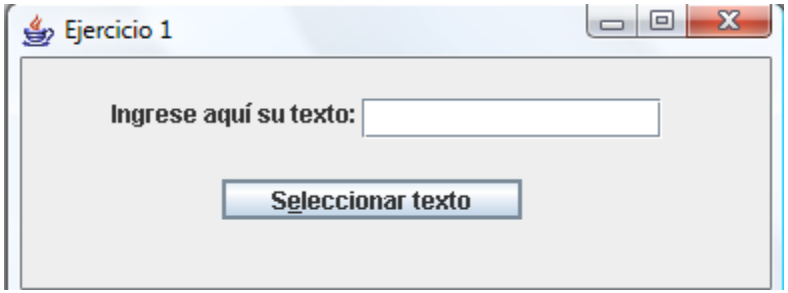
- Supongamos que los tres métodos (*keyPressed*, *keyTyped* y *keyReleased*) incluyen el siguiente código:

```
JTextField objJTF = (JTextField)e.getSource();
String str = objtJTF.getText();          //Contenido de la caja de texto
char car = e.getKeyChar();               //Carácter tecleado
```

- Si ejecutamos el código anterior podremos observar que el contenido de la caja de texto (almacenado en la variable *str*) obtenido de los métodos *keyPressed* y *keyTyped* no incluye el carácter *car* correspondiente a la tecla que generó el evento (último carácter tecleado) y sí lo hace el método *keyReleased*

Ejercicio 1

- Realizar una aplicación con la siguiente interfaz gráfica:



- Añadir a dicha aplicación las siguientes propiedades y funcionalidades:
  - La funcionalidad de que el botón pueda ser activado con las teclas *Alt + e*
  - Asociar una breve descripción al botón de pulsación que diga: *clic aquí para seleccionar el texto de la caja de texto*
  - Establecer el botón como botón predeterminado
  - Añadir la funcionalidad para que al dar clic sobre el botón se seleccione todo el texto dentro de este
  - Añadir la funcionalidad para que dentro de la caja de texto solamente se puedan escribir letras entre la *a-z* ó *A-Z*

Solución al ejercicio 1

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

class prueba extends JFrame
{
   public prueba() {
      initComponents();
   }
   private void initComponents() {
      //Etiqueta lblTexto
      lblTexto = new JLabel("Ingrese aquí su texto: ");
      lblTexto.setBounds(45,20,125,15);
      //Caja de texto txtTexto
      txtTexto = new JTextField();
      txtTexto.setBounds(170,20,150,20);
      //Añadir manejadores de eventos a la caja de texto
      txtTexto.addFocusListener(new FocusAdapter() {
         public void focusGained(FocusEvent e) {
            txtTextoFocusGained(e);
         }
      });
      txtTexto.addKeyListener(new KeyAdapter() {
         public void keyReleased(KeyEvent e) {
            txtTextoKeyReleased(e);
         }
      });
      //Botón de pulsación btnSeleccionar
      btnSeleccionar = new JButton("Seleccionar texto");
      btnSeleccionar.setBounds(100,60,150,20);
      //Funcionalidad de nemónico
      btnSeleccionar.setMnemonic('e');
      //Funcionalidad de breve descripción
      btnSeleccionar.setToolTipText("Clic aquí para seleccionar el texto de la caja de texto");
      //Añadir manejadores de eventos al botón
      btnSeleccionar.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent e) {
            btnSeleccionarActionPerformed(e);
          }
      });
      //JFrame prueba
      setTitle("Ejercicio 1");
      setSize(390,150);
      getContentPane().setLayout(null);
      //Establecer el botón predeterminado
      getRootPane().setDefaultButton(btnSeleccionar);
      //Añadimos los componentes al formulario
      getContentPane().add(lblTexto);
      getContentPane().add(txtTexto);
      getContentPane().add(btnSeleccionar);
      //Añadir manejadores de eventos al formulario
      addWindowListener(new WindowAdapter() {
         public void windowClosing(WindowEvent e) {
            frameWindowClosing(e);
         }
      });
   }
   public void frameWindowClosing(WindowEvent e) {
      System.exit(0);
   }
   public void btnSeleccionarActionPerformed(ActionEvent e) {
      //Establecemos el foco sobre la caja de texto
      txtTexto.requestFocus();
   }
   public void txtTextoFocusGained(FocusEvent e) {
      txtTexto.selectAll();
   }
   public void txtTextoKeyReleased(KeyEvent e) {
      char[] fuente = txtTexto.getText().toCharArray();
      char[] resultado = new char[fuente.length];
      int j = 0;
      boolean error = false;
      for(int i=0;i<fuente.length;i++) {
         if(Character.isLetter(fuente[i]))
            resultado[j++]=fuente[i];
         else  {
            error = true;
            //Emitimos un pitido indicando el error
            java.awt.Toolkit.getDefaultToolkit().beep();
         }
      }
      if(error)
         txtTexto.setText(new String(resultado,0,j));
   }
   public static void main(String[] args) {
      new prueba().setVisible(true);
   }
   //Declaraciones de variables
   private JLabel lblTexto = null;
   private JTextField txtTexto = null;
   private JButton btnSeleccionar = null;
}
```

Casillas de verificación

- Es un control que indica si una opción particular está activada o desactivada
- Cada casilla de verificación es independiente de las demás, ya que cada una de ellas tiene su propio identificador
- El número de opciones representadas de esta forma puede ser cualquiera, y de ellas el usuario puede seleccionar todas las que desee cada vez
- La funcionalidad para manipular este tipo de controles es proporcionada por la clase *JCheckBox*
- Un clic sobre el objeto genera un evento de la clase *ItemEvent* del paquete *java.awt.event* que será manejado por el método *itemStateChanged* de la interfaz *ItemListener*
- Para saber el tipo de cambio experimentado por la opción que originó el evento, hay que verificar el valor devuelto por el método *getStateChange*, este valor puede ser *DESELECTED* (sin marcar) o *SELECTED* (marcada)
- Otra forma de conocer el estado actual es invocando al método *isSelected*
- Para fijar el estado por medio de código podemos invocar al método *setSelected*

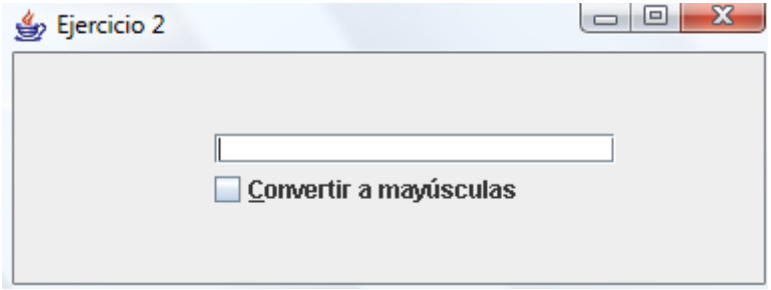
Casillas de verificación (cont.)

- Ejemplo:

```
//…
chbCasilla = new JCheckBox("Casilla de verificación");
chbCasilla.setBounds(120,45,150,15);
chbCasilla.addItemListener(new ItemListener()
{
        public void itemStateChanged(ItemEvent e)
        {
                chbCasillaItemStateChanged(e);
        }
});
//…
getContentPane().add(chbCasilla);
//…
public void chbCasillaItemStateChanged(ItemEvent e)
{
        if(e.getStateChange() == ItemEvent.SELECTED)
                JOptionPane.showMessageDialog(null,"Casilla de verificación marcada");
}
//…
private JCheckBox chbCasilla = null;
//…
```

Ejercicio 2

- Realizar una aplicación con la siguiente interfaz gráfica:



- Añadir a dicha aplicación las siguientes propiedades y funcionalidades:
  - La funcionalidad de que el botón pueda ser activado con las teclas *Alt + C*
  - Evitar que el usuario sea capaz de redimensionar el formulario
  - Cuando el usuario seleccione la opción *Convertir a mayúsculas*, todo el texto que haya escrito en la caja de texto aparezca en mayúsculas así como también todo el texto que se escriba a continuación (mientras la opción esté activa)
  - Cuando el usuario des-seleccione la opción, todo el texto que haya escrito en la caja de texto aparezca en minúsculas así como también todo el texto que se escriba a continuación

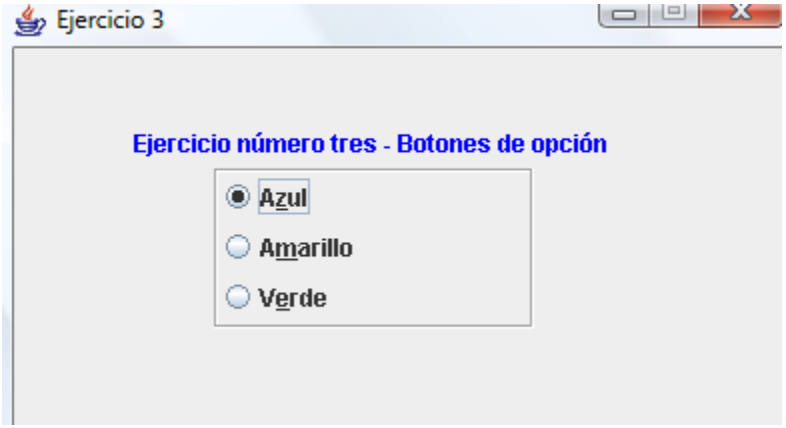
NOTA: Contemplar los cambios en el comportamiento de la aplicación si la tecla *Caps Lock* del teclado está activa

Botones de opción

- Es un control que indica si una determinada opción está activada o desactivada
- La funcionalidad para manipular este tipo de controles es proporcionada por la clase *JRadioButton*
- Un grupos de opciones es un objeto de la clase *ButtonGroup* que agrupa a los botones de opción que pertenecen a un mismo grupo
- Un clic sobre un objeto genera un evento de la clase *ItemEvent* del paquete *java.awt.event* que será manejado por el método *itemStateChanged* de la interfaz *ItemListener*
- Para saber el tipo de cambio experimentado por la opción que originó el evento, hay que verificar el valor devuelto por el método *getStateChange*, este valor puede ser *DESELECTED* (sin seleccionar) o *SELECTED* (seleccionado)
- Otra forma de conocer el estado actual es invocando al método *isSelected*
- Para fijar el estado por medio de código podemos invocar al método *setSelected*
- Cuando se hace clic sobre un botón de opción que pertenece a un grupo, hay dos que cambian de estado: el que estaba seleccionado y el nuevo seleccionado

Ejercicio 3

- Realizar una aplicación con la siguiente interfaz gráfica:



- Añadir a dicha aplicación las siguientes propiedades y funcionalidades:
  - La funcionalidad de que los botones de radio: Azul, Amarillo y Verde puedan ser activados con las teclas *Alt + z*, *Alt + m* y *Alt + e* respectivamente
  - Establecer el botón de radio Azul como el botón seleccionado de forma predeterminada y el color del texto en la etiqueta, al iniciar la aplicación, también deberá corresponder con este
  - Cuando el usuario pulse cualquiera de los botones de radio, el color del texto mostrado en la etiqueta deberá cambiar según el color del botón de radio que haya sido pulsado