Práctica VI - Cajas de Selección (JComboBox)

Muchas veces requerimos mostrar un conjunto de opciones para que el usuario seleccione alguna opción. El control `JComboBox` es ideal para estos casos. El control `JComboBox` tiene por defecto de elementos fijo, sin embargo, podemos cambiar esto vía código mediante un modelo. El evento que nos permite determinar cuando cambió el elemento es `itemStateChanged` que se invoca cada que el usuario cambia un item en la caja de selección, este evento sin embargo se ejecuta dos veces, la primera vez antes de que ocurra el cambio (`State: 1`) y la segunda vez cuando ocurre el cambio (`State 2`).

Manipular el evento de cambio de elemento de un `JComboBox`

`JComboBox` posee el evento `itemStateChanged` pero se manda a llamar dos veces por lo que temos que recuperar el valor del estado con `evt.getStateChange()` y comprobar que sea 1. Luego podemos recuperar el elemento seleccionado con `jComboBox1.getSelectedItem()` . El código final debería ser similar al siguiente:

```
private void jComboBox1ItemStateChanged(java.awt.event.ItemEvent evt) {
    if (evt.getStateChange() == 1) {
        System.out.println(this.jComboBox1.getSelectedItem());
    }
}
```

Ajustar el modelo de `JComboBox`

Podemos asociar un modelo de elementos a través de `DefaultComboBoxModel` . Si por ejemplo queremos una caja de selección que contenga datos de tipo `String` usamos `DefaultComboBoxModel<String>` . El modelo nos permite manipular elementos dinámicos que puedan cambiar en transcurso de la aplicación, por ejemplo si tenemos que agregar o quitar elementos de la caja de selección. El modelo puede ser un atributo de nuestra clase `JFrame` y luego enlazarlo en el constructor con `jComboBox1.setModel(model);` :

```
public class MiVentana extends javax.swing.JFrame {

    DefaultComboBoxModel<String> model = new DefaultComboBoxModel();

    public MiVentana() {
        initComponents();

        this.jComboBox1.setModel(this.model);

        this.model.addElement("Hola");
        this.model.addElement("Mundo");
    }

    ...
}
```

Así por ejemplo si queremos agregar otro elemento desde un botón en su evento `mouseClicked` hacemos:

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    this.model.addElement("Otro elemento");
}
```

Crear una ventana para agregar elementos a un `JComboBox` dinámicamente

- Crea un ventana que contenga una caja de selección (`JComboBox`)
- Agrega una caja de texto (`JTextField`)
- Agrega una botón (`JButton`)
- En el código agrega un modelo `DefaultComboBoxModel<String> model = new DefaultComboBoxModel();` como atributo de la clase (`JFrame`)
- Enlaza el modelo a la caja de selección `this.jComboBox1.setModel(this.model);` en el constructor de la clase
- En el evento `mouseClicked` del botón, recupera el valor de la caja de texto y agrega el texto al modelo:

```
private void jButton1MouseClicked(java.awt.event.MouseEvent evt) {
    String texto = jTextField1.getText();
    this.model.addElement(texto);
}
```

Ahora cada que el usuario esciba un texto y de clic sobre el botón las opciones de la caja de selección crecerán.

Problemas

- Modifica el código para evitar que agregue cadenas vacías.
- Guarda las opciones que vienen desde la caja de texto en mayúsculas.
- Impide que se agreguen más de 5 opciones.
- Muestra en un `JLabel` el valor de la selección y actualiza cada que ocurra el evento `itemStateChanged` en la caja de selección.