

Práctica 5. Patrones Estructurales . “Definiendo clases”

Objetivos y Método de trabajo

Con esta práctica se pretende que el alumno detalle las características de las clases y sus relaciones dibujadas en prácticas anteriores.

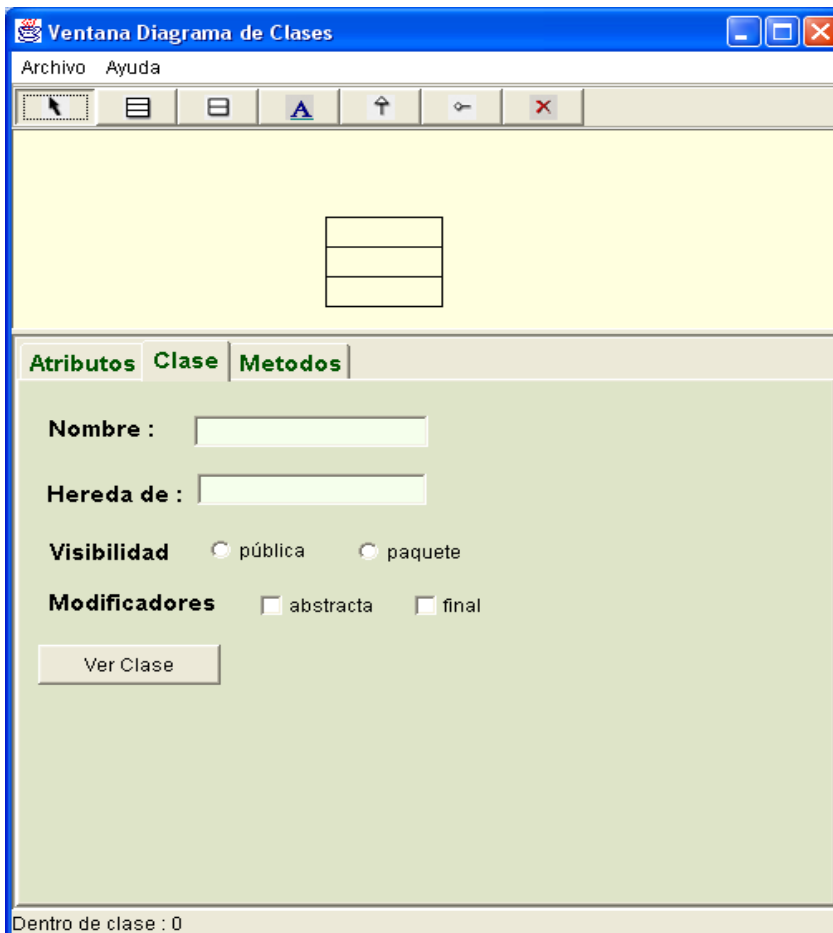
Deberá asimismo identificar los patrones convenientes para la implementación del código.

A desarrollar en 2 turnos.

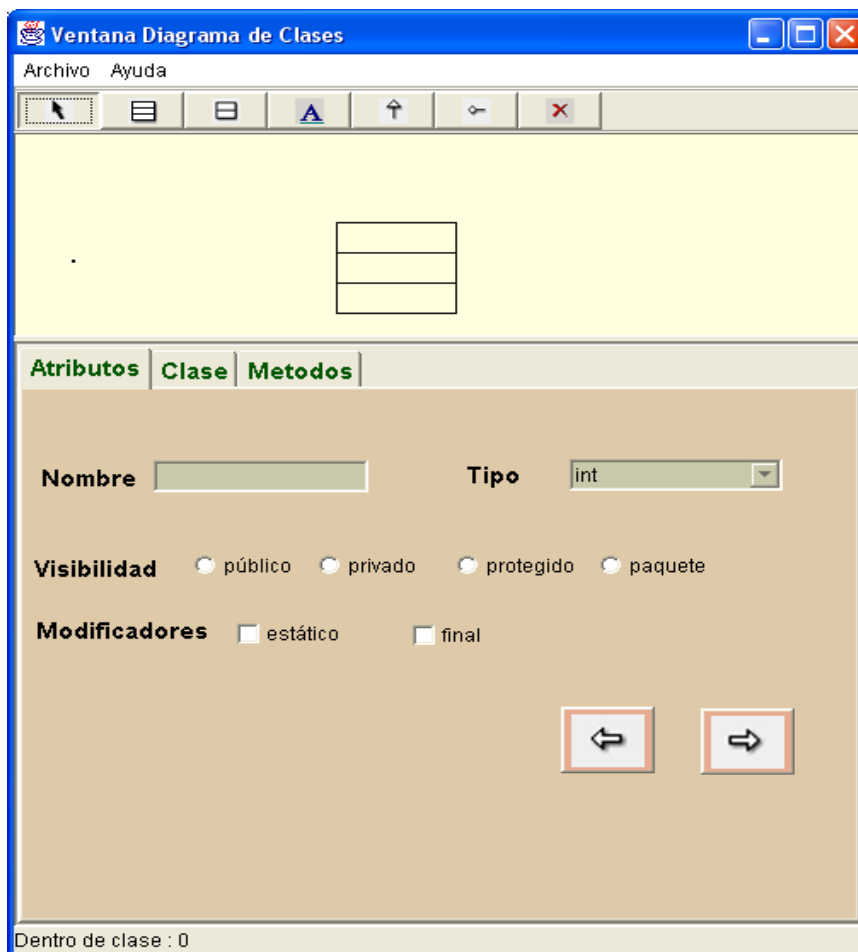
Aspectos a tratar

El diseño es **orientativo** y muy personal, sin que esto implique restar funcionalidad adecuada al componente utilizado.

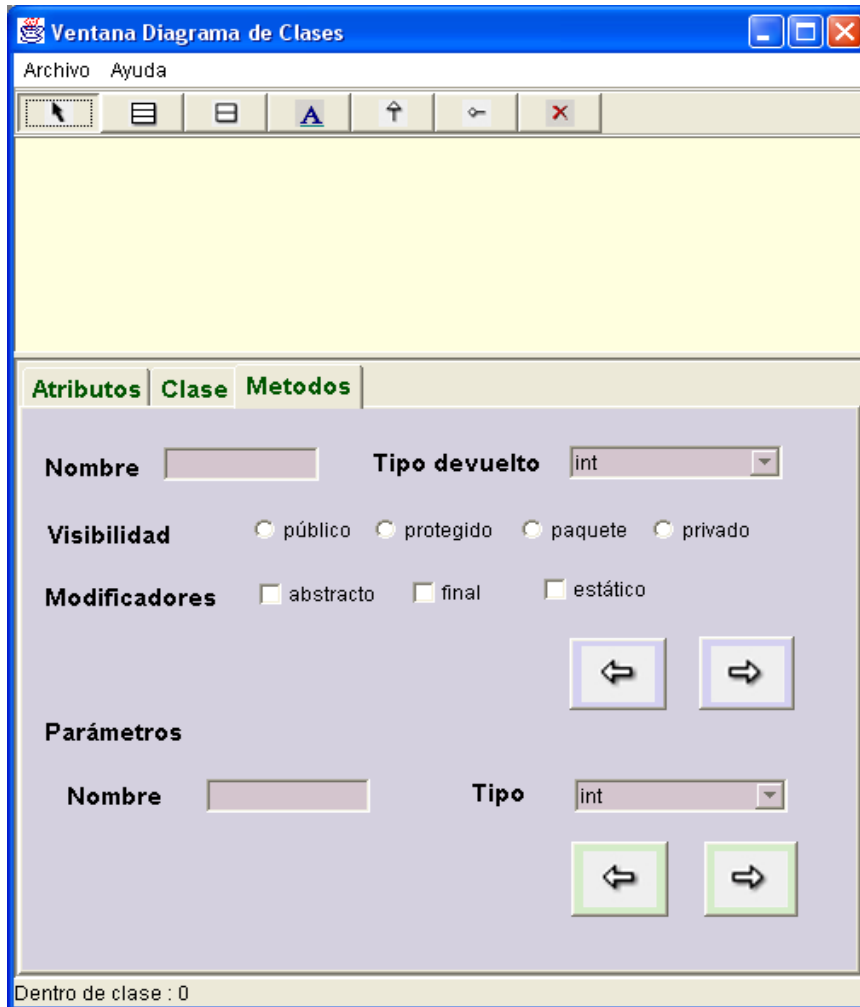
Con una determinada clase seleccionada desde el panel de dibujo, se deben considerar , las siguientes características de dicha clase :



Panel que describe básicamente las características generales de una clase o interfaz



Panel que describe las características de los atributos de la clase seleccionada, las flechas permiten la edición de atributos ya guardados para su modificación. También se puede emplear un JList para editar todos los atributos guardados.



Panel que describe las características de los métodos de la clase, así como los parámetros. Al igual que los atributos, tienen flechas para "recorrer" los distintos métodos de la clase. De igual forma, al tener un número indeterminado de parámetros, existen unas flechas a tal efecto.

También se podría utilizar JList en vez de las flechas.

Controles utilizados

Todos los nombres de identificadores son JTextField.

Para la visibilidad he utilizado JRadioButton.

Los modificadores son JCheckBox

Para dar valor a un tipo de dato se utiliza JComboBox.

Las flechas pueden ser JButton con iconos.

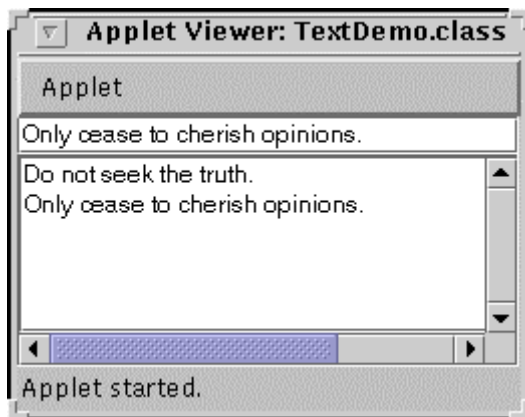
Algunas aclaraciones sobre los distintos componentes :

¿Cómo usar JTextField?

Un campo de texto es un control básico que permite al usuario teclear una pequeña cantidad de texto y dispara un evento `action` cuando el usuario indique que la entrada de texto se ha completado (normalmente pulsando `Return`). Generalmente se usa la clase `JTextField` para proporcionar campos de texto..

Si queremos un campo de texto que también proporcione un menú de cadenas desde la que elegir una, podemos considerar la utilización de un `combo box` editable. Si necesitamos obtener más de una línea de texto desde el usuario deberíamos utilizar una de las `glases` que implementan `text area` para propósito general.

El Applet siguiente muestra un campo de texto básico y un área de texto. El campo de texto es editable y el área de texto no lo es. Cuando el usuario pulse `Return` en el campo de texto, el campo dispara un `action event`. El applet reacciona al evento copiando el contenido del campo de texto en el área de texto y seleccionando todo el texto del campo de texto.



Esta es una imagen del GUI del applet. Para ejecutar el applet, pulsa sobre la imagen. El Applet aparecerá en una nueva ventana del navegador.

Puedes encontrar el programa fuente en `TextDemo.java`. Aquí está el código de `TextDemo` que crea el campo de texto del applet.

```
textField = new JTextField(20);
textField.addActionListener(this);
...
contentPane.add(textField);
```

El argumento entero pasado al constructor de `JTextField`, **20** en el ejemplo, indica el número de columnas del campo, que es usada junto con la métrica proporcionada por el font actual del campo, para calcular la anchura preferida por el campo. Como varios controladores de disposición ignoran los tamaños preferidos y como los márgenes, los bordes y otros factores afectan al tamaño del componente, toma este número como una aproximación, no un número absoluto.

Las siguientes líneas de código registran el applet como oyente de `action` para el campo de texto y añade el campo de texto al panel de contenidos del applet. Aquí está el método `actionPerformed` que maneja los eventos `action` del campo de texto.

```
public void actionPerformed(ActionEvent evt) {
    String text = textField.getText();
    textArea.append(text + newline);
    textField.selectAll();
}
```

Observa el uso del método **getText** de **JTextField** para recuperar el contenido actual del campo de texto. El texto devuelto por este método no incluye un carácter de nueva línea para la tecla Return que disparó el evento **action**.

Este ejemplo ilustra usando un campo de texto básico para introducir datos textuales y realizar algunas tareas cuando el campo de texto dispara un evento **action**. Otros programas, sin embargo, necesitan un comportamiento más avanzado. Una subclase de **JTextComponent**, **JTextField** puede ser configurada y personalizada. Una personalización común es proporcionar un campo de texto cuyos contenidos sean validados. Esta sección cubre los siguientes tópicos de los campos de texto avanzados. Para entender toda la información, necesitas haber comprendido el material presentado en Reglas Generales para el uso de Componentes.

■ Crear un Text Field Validado

Muchos programas requieren que el usuario introduzca un dato textual de un cierto tipo o formato. Por ejemplo, un programa podría proporcionar un campo de texto para entrar una fecha, un número decimal, o un número de teléfono. Los contenidos de dichos campos como campos de texto deben ser validados antes de ser utilizados para cualquier propósito. Un campo de texto puede ser validado cuando se dispare el evento **action** o el evento **keystroke**.

El dato en un campo validado-en-action se chequea cada vez que el campo dispara un evento **action** (cada vez que el usuario pulsa la tecla Return). Un campo validado-en-action podría, en un momento dado, contener datos no válidos. Sin embargo, el dato será validado antes de ser utilizado. Para crear un campo validado-en-action, necesitamos proporcionar un oyente **action** para nuestro campo e implementa su método **actionPerformed** de la siguiente forma.

Usa **getText** para obtener el contenido del campo de texto.

Evalúa el valor devuelto por **getText**.

Si el valor es válido, realiza cualquier tarea de cálculo que sea requerida. Si el campo es nulo, reporta el error y retorna sin realizar ninguna tarea de cálculo.

El dato en un campo validado-en-pulsación se chequea cada vez que el campo cambia. Un campo validado-en-pulsación nunca puede contener datos no válidos porque cada cambio (pulsación, cortar, copiar, etc.) hace que el dato no válido sea rechazado. Para crear un campo de texto validado-en-pulsación necesitamos proporcionar un documento personalizado para nuestro campo de texto. Si no estás familiarizado con los documentos, puedes ir a Trabajar con el Documento de un Componente de Texto

■ El API de JTextField

Las siguientes tablas listan los constructores y métodos más comúnmente utilizados de **JTextField**. Otros métodos a los que se podría llamar están definidos en las clases **JComponent** y **Component**. Estos incluyen los métodos **setForeground**, **setBackground**, y **setFont**.

Además, podrías querer llamar a algunos de los métodos definidos en la clase padre de **JTextField**, **JTextComponent**.

El API para usar campos de texto se divide en tres categorías.

■ Seleccionar u Obtener el Contenido de un Campo de Texto

Método o Constructor	Propósito
JTextField()	Crea un ejemplar de JTextField , inicializando su contenido al texto especificado. El argumento int selecciona el número de columnas. Esto se utiliza para calvar la anchura preferida del componente y podría no ser el número de columnas realmente mostradas.
JTextField(String)	
JTextField(String, int)	
JTextField(int)	
JTextField(Document, String, int)	
void setText(String) String getText()	Selecciona u obtiene el texto mostrado por el campo de texto.

■ Ajuste Fino de la Apariencia de un Campo de Texto

Método o Constructor	Propósito
void setEditable(boolean) boolean isEditable()	Selecciona u obtiene si el usuario puede editar el texto del campo del texto.

void setForeground(Color) Color getForeground()	Selecciona u obtiene el color del texto en el campo de texto.
void setBackground(Color); Color getBackground()	Selecciona u obtiene el color del fondo del campo de texto
void setFont(Font); Font getFont()	Selecciona u obtiene la fuente utilizada por el campo de texto.
void setColumns(int); int getColumns()	Selecciona u obtiene el número de columnas mostradas por el campo de texto.
int getColumnWidth()	Obtiene la anchura de las columnas del campo de texto. Este valor es establecido implícitamente por la fuente usada.
void setHorizontalAlignment(int); int getHorizontalAlignment()	Selecciona u obtiene cómo se alinea el texto horizontalmente dentro de su área. Se puede utilizar JTextField.LEFT , JTextField.CENTER , y JTextField.LEFT como argumentos.

■ Implementar la Funcionalidad del Campo de Texto

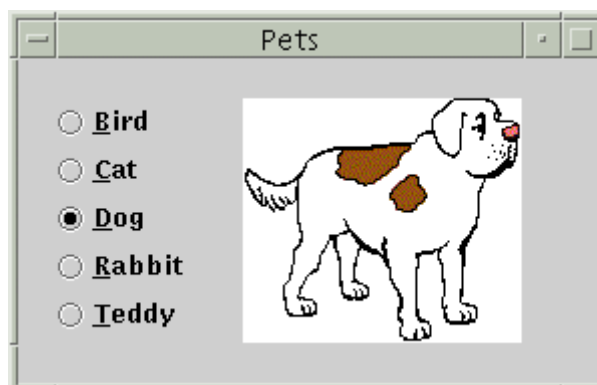
Método o Constructor	Propósito
void addActionListener(ActionListener) void removeActionListener(ActionListener)	Añade o elimina un oyente de action.
Document createDefaultModel()	Sobreescribe este método para proporcionar un documento personalizado.

¿Cómo Usar JRadioButton?

Los Botones de Radio son grupos de botones en los que, por convención, sólo uno de ellos puede estar seleccionado. Swing soporta botones de radio con las clases **JRadioButton** y **ButtonGroup**. Para poner un botón de radio en un menú, se utiliza la clase **JRadioButtonMenuItem**. Otras formas de presentar una entre varias opciones son los combo boxes y las listas. Los botones de radio tienen un aspecto similar a los check boxes, pero, por convención, los checkboxes no tienen límites sobre cuantos ítems pueden estar seleccionados a la vez.

Como **JRadioButton** descende de **AbstractButton**, los botones de radio Swing tienen todas las características de los botones normales. como se explicó en Cómo usar Buttons. Por ejemplo, se puede especificar la imagen mostrada por un botón de radio.

Aquí podemos ver una imagen de una aplicación que utiliza cinco botones de radio para elegir qué tipo de mascota mostrar.



Cada vez que el usuario pulsa un botón de radio, (incluso si ya estaba seleccionado), el botón dispara un **evento action**. También ocurren uno o dos **eventos item** -- uno desde el botón que acaba de ser

seleccionado, y otro desde el botón que ha perdido la selección (si existía). Normalmente, las pulsaciones de los botones de radio se manejan utilizando un oyente de acción.

Abajo está el código de **RadioButtonDemo.java** que crea los botones de radio en el ejemplo anterior y reacciona ante las pulsaciones.

```
//In initialization code:
// Create the radio buttons.
JRadioButton birdButton = new JRadioButton(birdString);
birdButton.setMnemonic('b');
birdButton.setActionCommand(birdString);
birdButton.setSelected(true);

JRadioButton catButton = new JRadioButton(catString);
catButton.setMnemonic('c');
catButton.setActionCommand(catString);

JRadioButton dogButton = new JRadioButton(dogString);
dogButton.setMnemonic('d');
dogButton.setActionCommand(dogString);

JRadioButton rabbitButton = new JRadioButton(rabbitString);
rabbitButton.setMnemonic('r');
rabbitButton.setActionCommand(rabbitString);

JRadioButton teddyButton = new JRadioButton(teddyString);
teddyButton.setMnemonic('t');
teddyButton.setActionCommand(teddyString);

// Group the radio buttons.
ButtonGroup group = new ButtonGroup();
group.add(birdButton);
group.add(catButton);
group.add(dogButton);
group.add(rabbitButton);
group.add(teddyButton);

// Register a listener for the radio buttons.
RadioListener myListener = new RadioListener();
birdButton.addActionListener(myListener);
catButton.addActionListener(myListener);
dogButton.addActionListener(myListener);
rabbitButton.addActionListener(myListener);
teddyButton.addActionListener(myListener);
...
class RadioListener implements ActionListener ... {
    public void actionPerformed(ActionEvent e) {
        picture.setIcon(new ImageIcon("images/" + e.getActionCommand() + ".gif"));
    }
}
```

Para cada grupo de botones de radio, se necesita crear un ejemplar de **ButtonGroup** y añadirle cada uno de los botones de radio. El **ButtonGroup** tiene cuidado de desactivar la selección anterior cuando el usuario selecciona otro botón del grupo.

Generalmente se debería inicializar un grupo de botones de radio para que uno de ellos esté seleccionado. Sin embargo, el API no fuerza esta regla -- un grupo de botones de radio puede no tener selección inicial. Una vez que el usuario hace una selección, no existe forma para desactivar todos los botones de nuevo.

El API JRadio Button

Puedes ver El API Button para información sobre el API de **AbstractButton** del que descienden **JRadioButton** y **JRadioButtonMenuItem**. Los métodos de **AbstractButton** que más se utilizan son **setMnemonic**, **addItemListener**, **setSelected**, y **isSelected**. Las piezas más utilizadas del API de Radio Button se dividen en dos grupos.

Métodos y Constructores más utilizados de ButtonGroups

Método	Propósito
ButtonGroup()	Crea un ejemplar de ButtonGroup .
void add(AbstractButton)	
void remove(AbstractButton)	Añade un botón a un grupo, o elimina un botón de un grupo.

Constructores de RadioButton

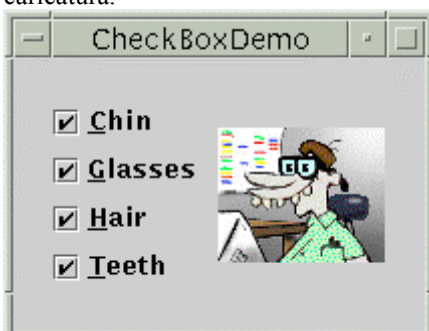
Constructor	Propósito
JRadioButton(String)	Crea un ejemplar de JRadioButton . El argumento string especifica el texto, si existe, que debe mostrar el botón de radio. Similarmente, el argumento, Icon especifica la imagen que debe usar en vez la imagen por defecto de un botón de radio para el aspecto y comportamiento. Si se especifica true en el argumento booleano, inicializa el botón de radio como seleccionado, sujeto a la aprobación del objeto ButtonGroup . Si el argumento booleano esta ausente o es false , el botón de radio está inicialmente deseleccionado.
JRadioButton(String, boolean)	
JRadioButton(Icon)	
JRadioButton(Icon, boolean)	
JRadioButton(String, Icon)	
JRadioButton(String, Icon, boolean)	
JRadioButton()	
JRadioButtonMenuItem(String)	Crea un ejemplar de JRadioButtonMenuItem . Los argumentos se interpretan de la misma forma que los de los constructores de JRadioButton .
JRadioButtonMenuItem(Icon)	
JRadioButtonMenuItem(String, Icon)	
JRadioButtonMenuItem()	

¿Cómo Usar JCheckBox?

La versión Swing soporta botones checkbox con la clase **JCheckBox**. Swing también soporta checkboxes en menus, utilizando la clase **JCheckBoxMenuItem**. Como **JCheckBox** y **JCheckBoxMenuItem** descienden de **AbstractButton**, los checkboxes de Swing tienen todas las características de un botón normal como se explicó en Cómo usar Buttons. Por ejemplo, podemos especificar imágenes para ser utilizadas en los checkboxes.

Los Checkboxes son similares a los botones de radio, pero su modelo de selección es diferente, por convención. Cualquier número de checkboxes en un grupo -- ninguno, alguno o todos -- pueden ser seleccionados. Por otro lado, en un grupo de botones de radio, sólo puede haber uno seleccionado.

Aquí podemos ver una imagen de una aplicación que utiliza cuatro checkboxes para personalizar una caricatura.



Un Checkbox genera un evento ítem y un evento action por cada pulsación. Normalmente, solo escucharemos los eventos de ítem, ya que nos permiten determinar si el click selecciona o desactiva el checkbox. Abajo puedes ver el código de **CheckBoxDemo.java** que crea los checkboxes del ejemplo anterior y reacciona ante las pulsaciones.

```
//In initialization code:
chinButton = new JCheckBox("Chin");
chinButton.setMnemonic('c');
chinButton.setSelected(true);

glassesButton = new JCheckBox("Glasses");
glassesButton.setMnemonic('g');
glassesButton.setSelected(true);

hairButton = new JCheckBox("Hair");
hairButton.setMnemonic('h');
hairButton.setSelected(true);

teethButton = new JCheckBox("Teeth");
teethButton.setMnemonic('t');
teethButton.setSelected(true);

// Register a listener for the check boxes.
CheckBoxListener myListener = new CheckBoxListener();
chinButton.addItemListener(myListener);
glassesButton.addItemListener(myListener);
hairButton.addItemListener(myListener);
teethButton.addItemListener(myListener);
...
class CheckBoxListener implements ItemListener {
    public void itemStateChanged(ItemEvent e) {
        ...
        Object source = e.getItemSelectable();

        if (source == chinButton) {
            //...make a note of it...
        } else if (source == glassesButton) {
            //...make a note of it...
        } else if (source == hairButton) {
            //...make a note of it...
        } else if (source == teethButton) {
            //...make a note of it...
        }

        if (e.getStateChange() == ItemEvent.DESELECTED)
            //...make a note of it...
            picture.setIcon(/* new icon */);
        ...
    }
}
```

■ El API JCheckBox

Puedes ver El API Button para información sobre el API de **AbstractButton** del que descienden **JCheckBox** y **JCheckBoxMenuItem**. Los métodos de **AbstractButton** que son más usados son **setMnemonic**, **addItemListener**, **setSelected**, y **isSelected**. El único API definido por **JCheckBox** y **JCheckBoxMenuItem** que utilizaremos son los constructores.

■ Constructores de CheckBox

Constructor

Propósito

JCheckBox(String)
JCheckBox(String, boolean)
JCheckBox(Icon)
JCheckBox(Icon, boolean)
JCheckBox(String, Icon)
JCheckBox(String, Icon, boolean)
JCheckBox()

Crea un ejemplar de **JCheckBox**. El argumento string especifica el texto, si existe, que el checkbox debería mostrar. De forma similar, el argumento **Icon** especifica la imagen que debería utilizarse en vez de la imagen por defecto del aspecto y comportamiento. Especificando el argumento booleano como **true** se inicializa el checkbox como seleccionado. Si el argumento booleano no existe o es **false**, el checkbox estará inicialmente desactivado.

JCheckBoxMenuItem(String)
JCheckBoxMenuItem(String, boolean)
JCheckBoxMenuItem(Icon)
JCheckBoxMenuItem(String, Icon)
JCheckBoxMenuItem(String, Icon, boolean)
JCheckBoxMenuItem()

Crea un ejemplar de **JCheckBoxMenuItem**. Los argumentos se interpretan de la misma forma que en los constructores de **JCheckBox**.

¿Cómo usar JComboBox?

Con un **JComboBox** editable, una lista desplegable, y un text field, el usuario puede teclear un valor o elegirlo desde una lista. Un **ComboBox** editable ahorra tiempo de entrada proporcionando atajos para los valores más comunmente introducidos.

Un **ComboBox** no editable desactiva el tecleo pero aún así permite al usuario seleccionar un valor desde una lista. Esto proporciona un espacio alternativo a un grupo de radio buttons o una list.

Aquí puedes ver una imagen de una aplicación que utiliza un **ComboBox** editable para introducir un patrón con el que formatear fechas.

Abajo podemos ver el código de **ComboBoxDemo.java** que crea y configura el **ComboBox**.

```
String[] patternExamples = {
    "dd MMMMM yyyy",
    "dd.MM.yy",
    "MM/dd/yy",
    "yyyy.MM.dd G 'at' hh:mm:ss z",
    "EEE, MMM d, 'yy",
    "h:mm a",
    "H:mm:ss:SSS",
    "K:mm a,z",
    "yyyy.MMMMM.dd GGG hh:mm aaa"
};

currentPattern = patternExamples[0];
...
JComboBox patternList = new JComboBox(patternExamples);
patternList.setEditable(true);
patternList.setSelectedIndex(0);
patternList.setAlignmentX(Component.LEFT_ALIGNMENT);
PatternListener patternListener = new PatternListener();
patternList.addActionListener(patternListener);
```

Este programa proporciona los valores para la lista desplegable del ComboBox con un array de strings. Sin embargo, los valores de la lista pueden ser cualquier **Object**, en cuyo caso el método **toString** de la clase **Object** proporciona el texto a mostrar. Para poner una imagen u otro valor que no sea texto en una lista ComboBox, sólo debemos proporcionar un celda personalizada renderizada con **setRenderer**.

Observa que el código activa explícitamente la edición para que el usuario pueda teclear valores. Esto es necesario porque, por defecto, un ComboBox no es editable. Este ejemplo particular permite editar el ComboBox porque su lista no proporciona todos los patrones de formateo de fechas posibles.

El código también registra un oyente de acción con el ComboBox. Cuando un usuario selecciona un ítem del ComboBox, se llama a este método.

```
public void actionPerformed(ActionEvent e) {
    JComboBox cb = (JComboBox)e.getSource();
    String newSelection = (String)cb.getSelectedItem();
    currentPattern = newSelection;
    reformat();
}
```

El método llama a **getSelectedItem** para obtener el nuevo patrón elegido por el usuario, y utilizarlo para reformatear la fecha y la hora actuales.

El API ComboBox

Las siguientes tablas listan los métodos y constructores más utilizados de **JComboBox**. Otros métodos a los que nos gustaría llamar están definidos por las clases **JComponent** y **Component**.

El API para utilizar ComboBox se divide en dos categorías.

Seleccionar u Obtener Ítems de la Lista del ComboBox

Método	Propósito
JComboBox(ComboBoxModel)	
JComboBox(Object[])	Crea un ComboBox con una lista predeterminada.
JComboBox(Vector)	
void addItem(Object)	Añade o inserta un ítem en la lista.
void insertItemAt(Object, int)	
Object getItemAt(int)	Obtiene un ítem de la lista.
Object.getSelectedItem()	
void removeAllItems()	
void removeItemAt(int)	Elimina uno o más ítems de la lista.
void removeItem(Object)	
int getItemCount()	Obtiene el número de ítems de la lista.
void setModel(ComboBoxModel)	Selecciona u obtiene el modelo de datos que proporciona los ítems de la lista.
ComboBoxModel getModel()	

Personalizar la Configuración del ComboBox

Método	Propósito
void setEditable(boolean)	
boolean isEditable()	Selecciona u Obtiene si el usuario puede teclear en el ComboBox.
void setRenderer(ListCellRenderer)	
ListCellRenderer getRenderer()	Selecciona u obtiene el objeto responsable para crear el ítem seleccionado en el ComboBox. Utilizado cuando el ComboBox no es editable.
void setEditor(ComboBoxEditor)	
ComboBoxEditor getEditor()	Selecciona u obtiene el objeto responsable del pintado y edición del ítem seleccionado en el ComboBox. Esto sólo se utiliza cuando el ComboBox es editable.

¿Cómo Usar JButton?

Para crear un botón, se ejemplariza una de las muchas subclases de la clase **AbstractButton**. Esta sección explica el API básico que define la clase **AbstractButton** -- y lo que todos los botones Swing tienen en común. Como la clase **JButton** desciende de **AbstractButton** define un pequeño API público adicional, esta página lo utiliza para ver cómo funcionan los botones.

La siguiente tabla muestra las subclases de **AbstractButton** definidas en Swing que podemos utilizar.

Clase	Sumario	Dónde se Describe
JButton	Un botón común	En esta sección.
JCheckBox	Un checkbox típico	Cómo usar CheckBox
JRadioButton	Un botón de radio de un grupo.	Cómo usar RadioButton
JMenuItem	Un ítem de un menú.	Cómo usar Menu
JToggleButton	Implementa la funcionalidad heredada de JCheckBox y JRadioButton .	En ningún lugar de este tutorial.

Nota: Si queremos juntar un grupo de botones dentro de una fila o una columna deberíamos chequear Cómo usar toolbar.

Aquí tienes una imagen de una aplicación que muestra tres botones.



Como muestra el ejemplo **ButtonDemo**, un botón Swing puede mostrar tanto texto como una imagen. En **ButtonDemo**, cada botón tiene su texto en un sitio diferente. La letra subrayada de cada texto de botón muestra el **mnemónico** -- la tecla alternativa -- para cada botón.

Cuando un botón se desactiva, le aspecto y comportamiento genera automáticamente la apariencia de botón desactivado. Sin embargo, podríamos proporcionar una imagen para que substituya la imagen normal. Por ejemplo, podría proporcionar versiones en gris de las imagenes utilizadas en los botones de la derecha y de la izquierda.

Cómo se implementa el manejo de eventos depende del tipo de botón utilizado y de cómo se utiliza. Generalmente, implementamos un action listener, que es notificado cada vez que el usuario pulsa el botón, Para un checkbox normalmente se utiliza un item listener, que es notificado cuando el checkbox es seleccionado o deseleccionado.

Abajo podemos ver el código de **ButtonDemo.java** que crea los botones del ejemplo anterior y reacciona a las pulsaciones de los botones. El código en **negrita** es el código que permanecería si los botones no tuvieran imágenes.

//In initialization code:

```
ImageIcon leftButtonIcon = new ImageIcon("images/LEFT.gif");
ImageIcon middleButtonIcon = new ImageIcon("images/middle.gif");
ImageIcon LEFTButtonIcon = new ImageIcon("images/left.gif");
```

```
b1 = new JButton("Disable middle button", leftButtonIcon);
b1.setVerticalTextPosition(AbstractButton.CENTER);
b1.setHorizontalTextPosition(AbstractButton.LEFT);
b1.setMnemonic('d');
b1.setActionCommand("disable");
```

```
b2 = new JButton("Middle button", middleButtonIcon);
b2.setVerticalTextPosition(AbstractButton.BOTTOM);
```

```

    b2.setHorizontalTextPosition(AbstractButton.CENTER);
    b2.setMnemonic('m');

    b3 = new JButton("Enable middle button", LEFTButtonIcon);
    //Use the default text position of CENTER, LEFT.
    b3.setMnemonic('e');
    b3.setActionCommand("enable");
    b3.setEnabled(false);

    //Listen for actions on buttons 1 and 3.
    b1.addActionListener(this);
    b3.addActionListener(this);
    ...
}

public void actionPerformed(java.awt.event.ActionEvent e) {
    if (e.getActionCommand().equals("disable")) {
        b2.setEnabled(false);
        b1.setEnabled(false);
        b3.setEnabled(true);
    } else {
        b2.setEnabled(true);
        b1.setEnabled(true);
        b3.setEnabled(false);
    }
}
}

```

■ El API JButton

Las siguientes tablas listan los métodos y constructores más utilizados de **AbstractButton** y **JButton**. Podemos ver la mayoría de este API jugando con el panel de botones del ejemplo **SwingSet** que forma parte de la versión Swing.

El API para utilizar botones se divide en tres categorías.

■ Seleccionar u Obtener el Contenido de un Botón

Método o Constructor	Propósito
JButton(String, Icon) JButton(String) JButton(Icon) JButton()	Crea un ejemplar de JButton , lo inicializa para tener el texto/imagen especificado.
void setText(String) String getText()	Selecciona u obtiene el texto mostrado en el botón.
void setIcon(Icon) Icon getIcon()	Selecciona u obtiene la imagen mostrada por el botón cuando está seleccionado o pulsado.
void setDisabledIcon(Icon) Icon getDisabledIcon()	Selecciona u obtiene la imagen mostrada por el botón cuando está desactivado. Si no se especifica una imagen, el aspecto y comportamiento crea una por defecto.
void setPressedIcon(Icon) Icon getPressedIcon()	Seleccion u obtiene la imagen mostrada por el botón cuando está pulsado.
void setSelectedIcon(Icon) Icon getSelectedIcon() void setDisabledSelectedIcon(Icon) Icon getDisabledSelectedIcon()	Selecciona u obtiene la imagen mostrada por el botón cuando está seleccionado. Si no se especifica una imagen de botón desactivado seleccionado, el aspecto y comportamiento crea una manipulando la imagen de seleccionado.

setRolloverEnabled(boolean)
boolean getRolloverEnabled()
void setRolloverIcon(Icon)
Icon getRolloverIcon()
void
setRolloverSelectedIcon(Icon)
Icon getRolloverSelectedIcon()

Utiliza **setRolloverEnabled(true)** y **setRolloverIcon(someIcon)** para hacer que el botón muestre el icono especificado cuando el cursor pasa sobre él.

■ Ajuste Fino de la Apariencia del Botón

Método o constructor

Propósito

void setHorizontalAlignment(int)
void setVerticalAlignment(int)
int getHorizontalAlignment()
int getVerticalAlignment()

Selecciona u obtiene dónde debe situarse el contenido del botón. La clase **AbstractButton** permite uno de los siguientes valores para alineamiento horizontal: **LEFT**, **CENTER** (por defecto), y **RIGHT**. Para alineamiento vertical: **TOP**, **CENTER** (por defecto), y **BOTTOM**.

void
setHorizontalTextPosition(int)
void setVerticalTextPosition(int)
int getHorizontalTextPosition()
int getVerticalTextPosition()

Selecciona u obtiene dónde debería situarse el texto del botón con respecto a la imagen. La clase **AbstractButton** permite uno de los siguientes valores para alineamiento horizontal: **LEFT**, **CENTER** (por defecto), y **RIGHT**. Para alineamiento vertical: **TOP**, **CENTER** (por defecto), y **BOTTOM**.

void setMargin(Insets)
Insets getMargin()

Selecciona u obtiene el número de pixels entre el borde del botón y sus contenidos.

void setFocusPainted(boolean)
boolean isFocusPainted()

Selecciona u obtiene si el botón debería parecer diferente si obtiene el foco.

void setBorderPainted(boolean)
boolean isBorderPainted()

Selecciona u obtiene si el borde del botón debería dibujarse.

■ Implementar la Funcionalidad del Botón

Método o Constructor

Propósito

void setMnemonic(char)
char getMnemonic()

Selecciona la tecla alternativa para pulsar el botón.

void setActionCommand(String)
String getActionCommand(void)

Selecciona u obtiene el nombre de la acción realizada por el botón.

void
addActionListener(ActionListener)
ActionListener removeActionListener()

Añade o elimina un objeto que escucha eventos action disparados por el botón.

void addItemListener(ItemListener)
ItemListener removeItemListener()

Añade o elimina un objeto que escucha eventos items disparados por el botón.

void setSelected(boolean)
boolean isSelected()

Selecciona u obtiene si el botón está seleccionado. Tiene sentido sólo en botones que tienen un estado on/off, como los checkbox.

void doClick()
void doClick(int)

Programáticamente realiza un "click". El argumento opcional especifica el tiempo (en milisegundos) que el botón debería estar pulsado.