	COMPONENTES Y MANEJO DE EVENTOS EN

JPANEL - JCHECKBOX - JRADIOBUTTON – BUTTONGROUP - JCOMBOBOX

Tener presente que el único tipo de datos que manejan los componentes son de tipo **String** y el valor a enviar debe ir entre comillas dobles, para ello se debe tener presente para efectos de procesos los métodos de biblioteca para sus respectivas conversiones:

Métodos del objeto, para extraer su valor existente en el formulario.

- double numero = **Double.parseDouble** (nombreObjeto.getText());
- int numero = **Integer.parseInt** (nombreObjeto.getText());
- float numero = **Float.parseFloat** (nombreObjeto.getText());

Para asignar una cadena al objeto creado en el formulario:

- objeto.setText (**String.valueOf** (ValorNumerico));

Componente	Prefijo	Descripción
JPanel	**pnl**	Contenedor para agrupar componentes en su interior
JTextField	**txt**	Caja de Texto, donde el usuario ingresará valores de entrada, acuerdo al requerimiento, también se pueden mostrar valores de salida, bloqueando para no modificar su resultado
JButton	**btn**	Un botón se requiere para que el usuario al utilizarlo dispare un evento o acción que ejecutará un cálculo o proceso; las acciones con el teclado (key press), sobre el mouse(move mouse, cick)
JCheckBox	**chk**	Casilla de verificación, el usuario puede tildar o des-tildar una o varias opciones de una lista; por ejemplo: Hobbies **(cine, deporte, tv)**
JRadioButton **ButtonGroup**	**rad**	El usuario puede seleccionar una y solamente una opción de una lista; por ejemplo para seleccionar el género de una persona **(Hombre – Mujer---Pocos)**

			Permite la funcionalidad de seleccionar una y solamente una opción de los radios que se incluyan dentro del grupo.
	ComboBox		El usuario puede seleccionar una y solamente una opción de una lista; por ejemplo para seleccionar un país de una lista **(Colombia – Ecuador – Perú…Muchos más)**

Constructores, a medida que se van diseñando los componentes, en el código se van generando las instrucciones respectivas para su creación; por ejemplo:

```
JLabel      lblEtiqueta    = new JLabel();
JTextField  txtCajaTexto = new JTextField();
JButton     btnBoton       = new JButton();
```

En el diseño para cambiar el nombre del objeto creado con el nombre y el prefijo correspondiente es con el mouse derecho sobre el mismo objeto en cambiar nombre de la variable; se recomienda cambiarle sola a los componentes que se usarán en el código para efectos cálculos, condicionales, ciclos.

1. **Contenedores JPanel**: Una vez creada una ventana con **JFrame**, se procede a crear los contenedores para alojar los componentes que se crearán dependiendo de la Interfaz Gráfica del Usuario GUI.

   El contenedor **JPanel** no tiene ningún aspecto visual, solo sirve para alojar en su interior otros componentes y/o contenedores; lo cual nos permite alojar componentes en su interior y no tener que posicionar uno por uno con setBounds, bastaría con aplicárselo al JPanel creado.

2. ***Layout Managers:*** Swing nos ofrece *gestores de aspecto* capaces de organizar, de manera automática, la posición y tamaño de los componentes dentro de los contenedores.

   Los contenedores tienen asignado, por defecto, alguno de estos *gestores de aspecto*. En el caso de **JFrame** este gestor es **BorderLayout**.

   - En Java no es habitual indicar explícitamente la posición de los componentes de la interfaz dentro de la ventana.
   - Los *layout managers* se encargan de colocar los componentes de la interfaz de usuario en la ventana contenedora.

- Especifican la posición y el tamaño de dichos componentes.
  - FlowLayout (en forma horizontal en el orden que se adicionen)
  - BoxLayout (coloca los componentes en forma horizontal o Vertical)
  - GridLayout (distribución filas x columnas)
  - BorderLayout (Se indica los puntos cardinales: este, oeste, norte y sur)
  - GridBagLayout (es el más complejo)

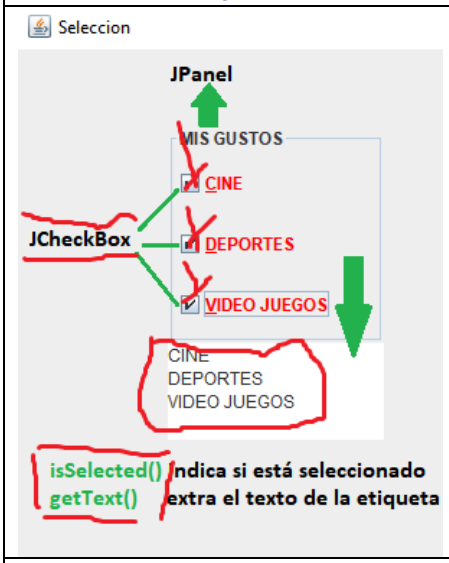
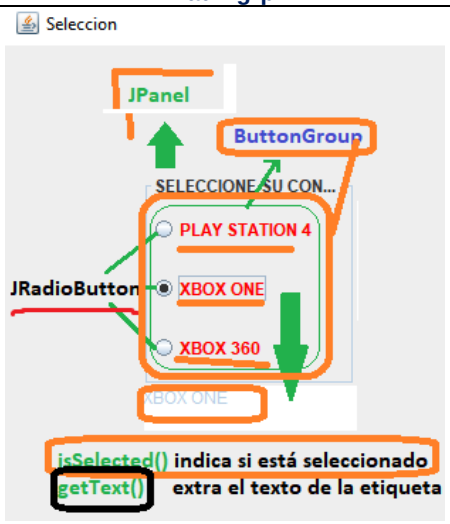
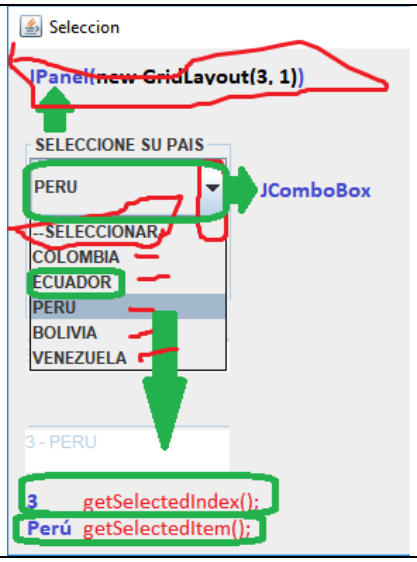
3. Si bien el **JButton** tradicional o clásico es uno de los más utilizados, hay otros tipos de botones que se requieren dominar, ya que su incorporación en las diferentes interfaces es bastante útil.

Entre los botones que debemos aprender a manejar tenemos los **Checkbox**, que son como una especie de interruptores que tienen dos estados que al presionarlos se le coloca una marca o check, de ahí su nombre.

El otro botón es el **Radio Button**, este tiene la característica que nos permite solo marcar una opción entre las disponibles, por lo que es perfecto para hacer selecciones únicas, como por ejemplo seleccionar el género de una persona (Másculino, Femenino) o tal vez alguna opción excluyente, recomendado para listas cortas diríamos que no más de 10.

El otro componente es el **ComboBox**, este tiene la característica que nos permite solo selccionar una opción entre las disponibles en una lista, por lo que es perfecto para hacer selecciones únicas, como por ejemplo seleccionar un país (……).

4.

JCHECKBOX chk	JRADIOBUTTON – BUTTONGROUP rad - grp	JCOMBOBOX cbo
		
El **JCheckbox**, permite seleccionar una, varios o ninguna opción de la lista	El **JRadioButton**, permite seleccionar una y solamente una opción de la lista, se utiliza para	El **JComoboBox**, permite seleccionar una y solamente una opción de la lista, se utiliza para listas muy largas, la lista se

		listas cortas y siempre visibles para el usuario en la GUI	despliega al momento del usuario hacer clic sobre el mismo
	JCheckBox (String, boolean);	ButtonGroup(); JRadioButton(String, boolean);	JComboBox(new Object[] {String, String, String});

5. **El checkbox**, permite seleccionar una, varias o ninguna opción de una la lista; también es conocido como casillas de verificación y es un botón tipo interruptor que maneja dos estados, encendido y apagado o en ingles podemos conseguirlo como *Checked* y ***Unchecked,*** su forma es rectangular y cuando está marcado tiene un visto bueno en el centro del rectangulo de la opción.

   Para darle forma a estas funcionalidades se utiliza el componente **JCheckBox** que hereda de la clase **JToggleButton**, lo interesante de esto es que podemos heredar todas las propiedades de la clase **AbstractButton** con lo que si sabemos manejar parcialmente los botones podremos aplicar esos conocimientos acá.

   El constructor recibe como parámetros el nombre en forma de *String* y un valor *Bool* en true, esto último significa que el botón aparecerá marcado o **checked** por defecto, por último el método **setMnemonic**, nos permite asignarle un atajo de teclado para poder marcar o no el botón.

   Cuando hacemos click sobre el *Checkbox*, tenemos un método muy útil que es el **isSelected** que nos permite saber si el *Checkbox* está marcado (true – false).

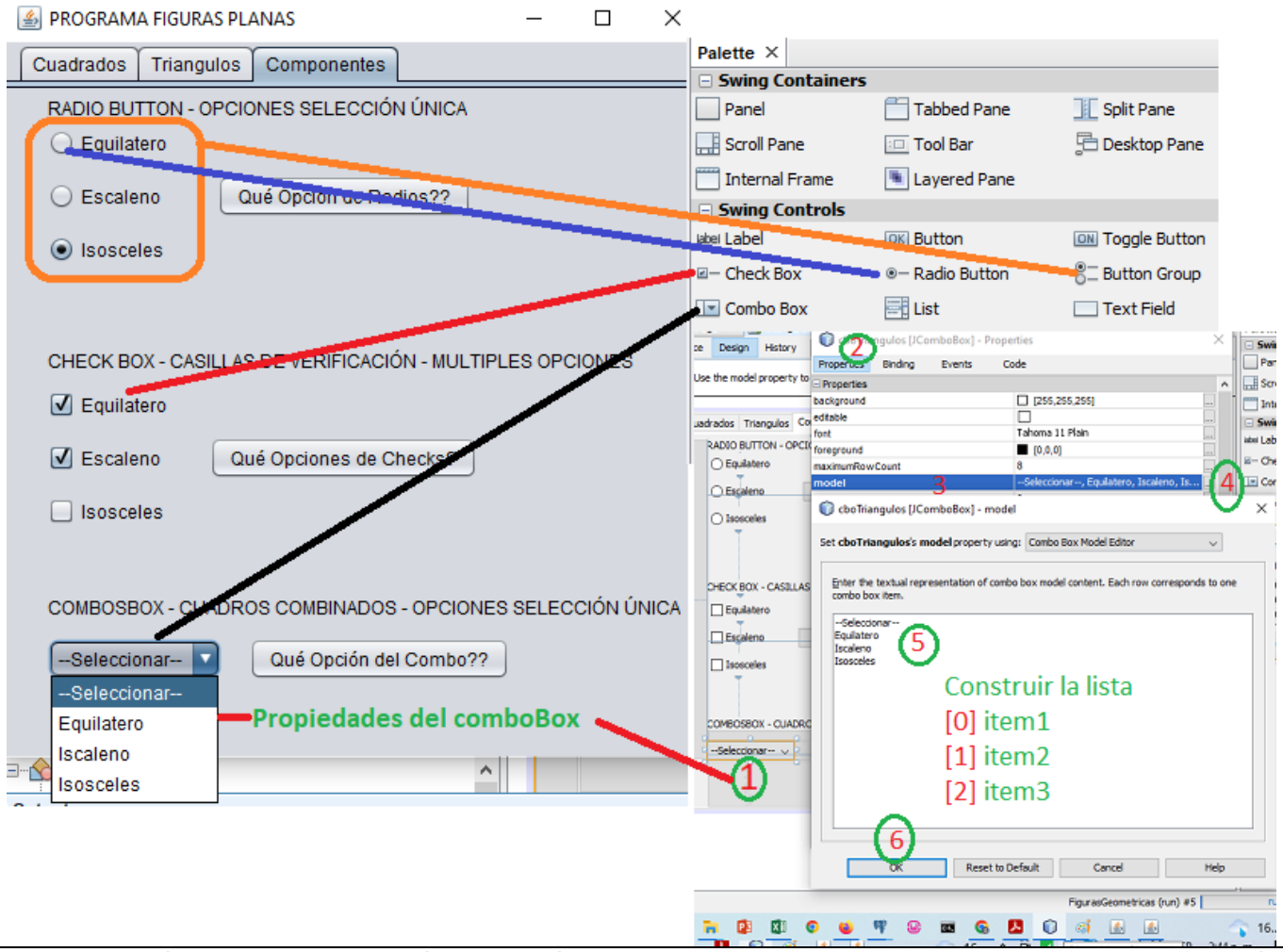
   - boolean seleccionado = chkNombre**.isSelected ();**
   - String cadena = chkNombre**.getText ();**

6. **El JRadioButton**, permite seleccionar una y solamente una opción de la lista, se utiliza para listas cortas y siempre visibles para el usuario en la GUI; con ello podemos lograr que se elijan opciones excluyentes, como por ejemplo género, tipo documento, modelos, colores, etc., su forma es circular y cuando está marcado tiene un punto en el centro del circulo de la opción.

   Para su construcción primero se crea un objeto del tipo **ButtonGroup**, luego se crean los objetos de la clase **JRadioButton y posteriormente se procede a añadir los botones dentro del grupo;** hace que al estar agrupados cuando seleccionemos una opción inmediatamente la otra sea deseleccionada.

   Cuando hacemos click sobre un radio, tenemos un método muy útil que es el **isSelected** que nos permite saber si el *RadioButton* está seleccionado (true – false).

   - boolean seleccionado = radNombre**.isSelected ();**
   - String cadena = radNombre**.getText ();**

7.	**El JComboBox**, permite seleccionar una y solamente una opción de la lista, se utiliza para listas largas que solamente se visualizan sus opciones, cuando el usuario interactua con el combo; por ejemplo Listado de Países, estudiantes, clientes, etc. Cuando hacemos click sobre el ***ComboBox*** podemos obtener mediante métodos el Índice y el Texto de la opción respectiva: • int indice = (int) cboNombre.**getSelectedIndex();** • String cadena = (String) cboNombre.**getSelectedItem();**
8.	La siguiente es la gráfica de diseño con NetBeans explicativa con los componentes previamente definidos y su respectivo código 
9.	A continuación, se presentan las líneas de código que normalmente se requieren implementar en el código al momento en que se dispare un evento por parte del usuario o al ser invocado en el programa:

8.1. Código para las casillas de verificación o JCheckBox

```
private void cmdCheckBoxAccion(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        String cadena = "";
        if (chkEquilatero.isSelected())
            cadena = cadena + chkEquilatero.getText() + " ";
        if (chkEscaleno.isSelected())
            cadena = cadena  + chkEscaleno.getText() + " ";
        if (chkIsosceles.isSelected())
            cadena = cadena  + chkIsosceles.getText() + " ";
        if (cadena == "")
            cadena = "SIN TILDAR OPCIONES DE LAS CASILLAS";

        JOptionPane.showMessageDialog(null, cadena);
    }
```

8.2. Código para los radios JRadioButton

```
private void cmdRadiosAccion(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        if (radEquilatero.isSelected())
                JOptionPane.showMessageDialog(null, "Seleccionó: " + radEquilatero.getText());
        else if (radEscaleno.isSelected())
            JOptionPane.showMessageDialog(null, "Seleccionó: " + radEscaleno.getText());
        else if (radIsosceles.isSelected())
            JOptionPane.showMessageDialog(null, "Seleccionó: " + radIsosceles.getText());
        else
            JOptionPane.showMessageDialog(null, "SIN Seleccion");
    }
```

8.3. Código para los combos o cuadros combinados JComboBox

```
private void cmdCombosAccion(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        int    indice = (int)   cboTriangulos.getSelectedIndex();
        String cadena = (String)cboTriangulos.getSelectedItem();
        JOptionPane.showMessageDialog(null, "INDICE[" + indice + "] \nCADENA:"+cadena);
    }
```

	COMPONENTES Y MANEJO DE EVENTOS EN **CRUD CON EL COMPONENTE TABLE**
	Un JTable es un componente visual de java que nos permite dibujar una tabla, de forma que en cada fila/columna de la tabla podamos poner el dato que queramos; un nombre, un apellido, una edad, un número, etc, etc. Un registro corresponde a una sola fila, que a su vez son datos (columnas) que hacen referencia a una sola entidad (Persona, Lugar, Objeto, tangible o intangible).
	
1.	Dibujar la tabla desde la paleta de componentes
2.	Crear un botón para cada una de las operaciones del CRUD • cmdInsertarCuadrado • cmdEliminarCuadrado • cmdActualizarCuadrado • cmdLimpiarCuadrados

3.	Instanciar un obejeto de la clase DefaultTableModel, que sea global a la clase • DefaultTableModel dtm = new DefaultTableModel();

4.	En el constructor de la clase inicializar los títulos de la tabla, cada columna tomará un índice a partir de [0], [1], [2], los cuales se deben tener en cuenta para las operaciones correspondientes.
	
	initComponents();
	String[] titulos = new String[] {"Ancho", "Alto", "Area", "Perímetro"};
	dtm.setColumnIdentifiers(titulos);
	tblCuadrados.setModel(dtm);

5.	**Código para Insertar**
	private void cmdInsertarActionPerformed(java.awt.event.ActionEvent evt) { **insertarRectangulo ();** **}**
	public void insertarRectangulo(){ **//alternative 1 creando todo al tiempo** **/*dtm.addRow(new Object[]{** **txtAncho.getText(),** **txtLargo.getText(),** **txtArea.getText(),** **txtPerimetro.getText()** **});*/** **//alternative 2, creando por partes** **String[] datos = new String[5];** **datos[0] = txtAncho.getText();** **datos[1] = txtLargo.getText();** **datos[2] = txtArea.getText();** **datos[3] = txtPerimetro.getText();** **dtm.addRow(datos);** **limpiarFormulario();** **}**
6.	**Código para Eliminar**
	private void cmdEliminarCuadradoAccion(java.awt.event.ActionEvent evt) { **// TODO add your handling code here:** **eliminarRectangulo ();** **}**
	public void eliminarRectangulo(){ **int fila = tblRectangulos.getSelectedRow();** **//confirmar**

```
    if (fila >=0){
      int opcion = JOptionPane.showConfirmDialog(null, "seguro");
      //System.out.println("opcion: " + opcion);
      if (opcion == 1) dtm.removeRow(fila);
    }
    else
      JOptionPane.showMessageDialog(null, "SELECCIONAR LA FILA A ELIMINAR");
  }
```

7. **Código para Actualizar**

```
private void cmdActualizarCuadradoActionPerformed(java.awt.event.ActionEvent evt) {
      actualizarRectangulos ();
}
```

```
public void actualizarRectangulos(){
    int fila = tblRectangulos.getSelectedRow();
    //dtm.setValueAt(objeto, fila, columna);
    dtm.setValueAt(txtAncho.getText(), fila, 0);
    dtm.setValueAt(txtLargo.getText(), fila, 1);
    dtm.setValueAt(txtArea.getText(), fila, 2);
    dtm.setValueAt(txtPerimetro.getText(), fila, 3);
  }
```

8. **Código para Limpiar la Tabla**

```
public void eliminarCuadrado(){
    int fila = tblCuadrados.getSelectedRow();
    dtm.removeRow(fila);
  }
```

```
public void limpiarCuadrado(){
    //int filas = tblCuadrados.getRowCount();
    int filas = dtm.getRowCount();
    for (int i=0; i<filas; i++){
      dtm.removeRow(0);
    }
  }
```