PROGRAMACIÓN ORIENTADA A OBJETOS
(Laboratorio de Prácticas)

Titulaciones de Grado en Ingeniería de Computadores, Ingeniería Informática y Sistemas de Información

Sesión 7
INTERFACES GRÁFICAS CON SWING Y POO

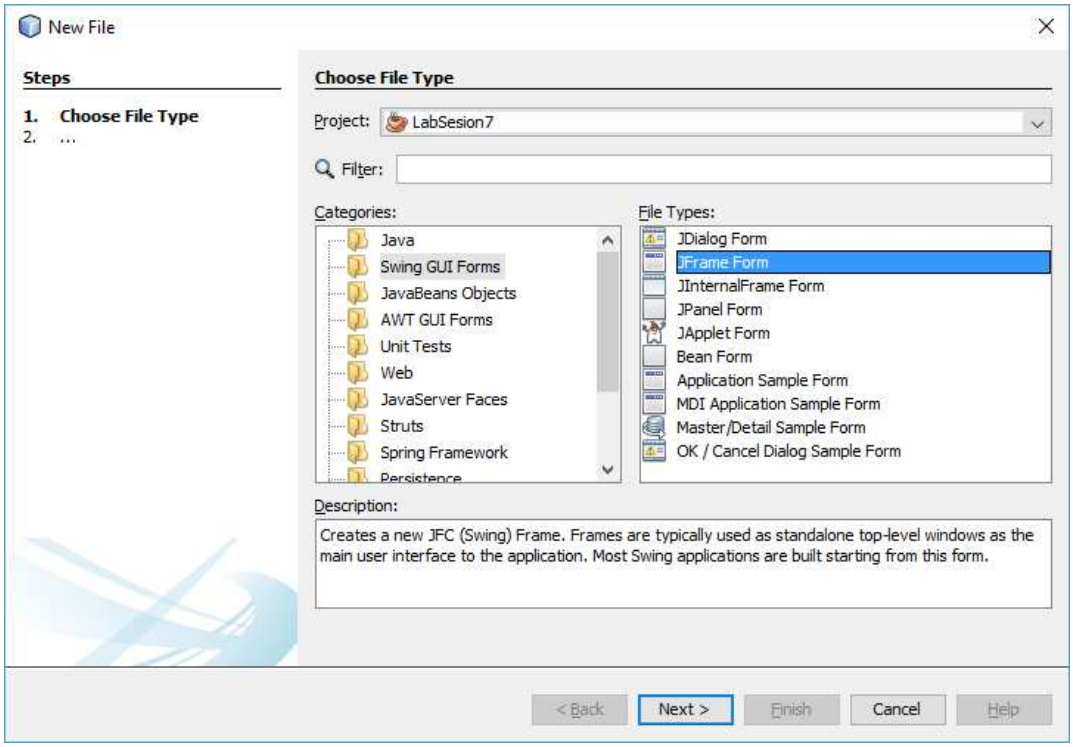
En esta sesión realizaremos interfaces gráficas de usuario con Swing y programas orientados a objetos avanzados que se comunican con estas interfaces.

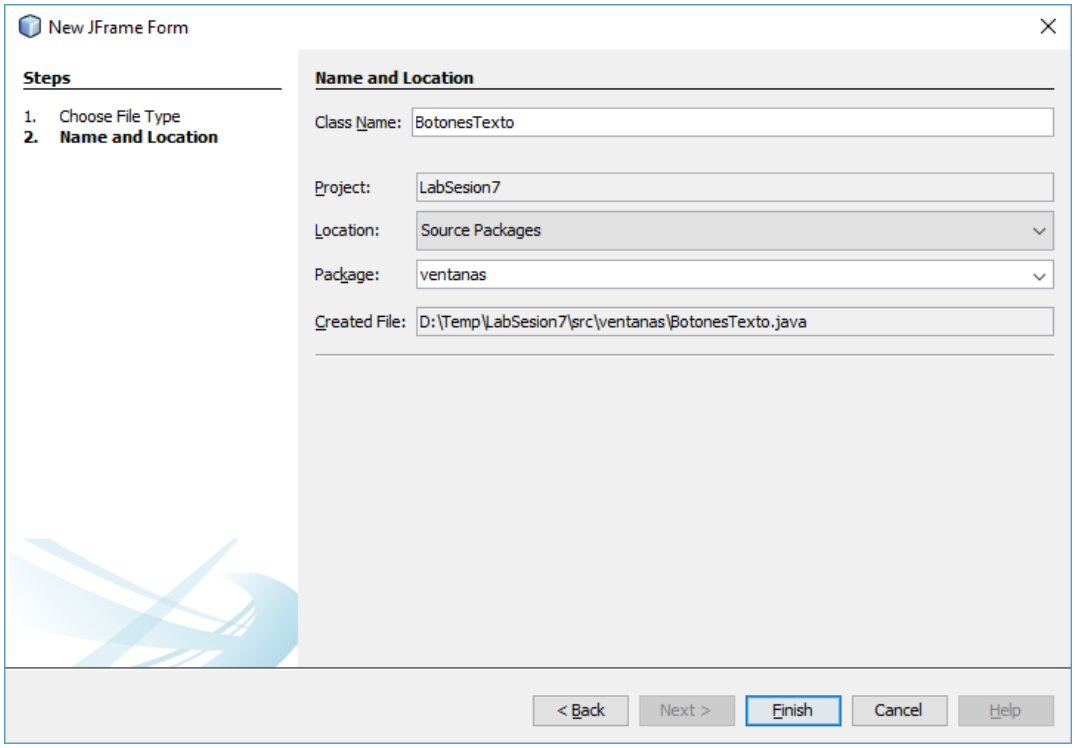
1. Ejemplos Swing

Lo primero que haremos será crear un proyecto llamado LabSesion7 donde meteremos todos nuestros ejemplos y ejercicios.

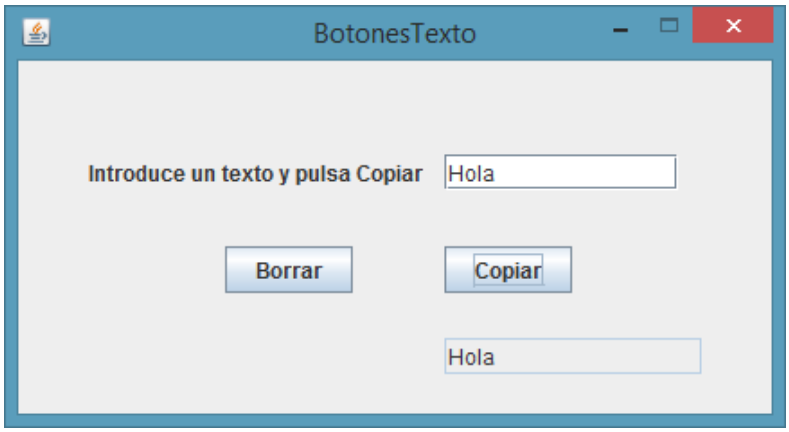
1.1 BotonesTexto

La primera aplicación que haremos consistirá en una ventana con una caja de texto en la que se introducirá un texto que copiaremos en otra caja mediante la pulsación de un botón. Para realizarla tenemos que crear en el proyecto un nuevo fichero de tipo JFrame que llamaremos BotonesTexto.



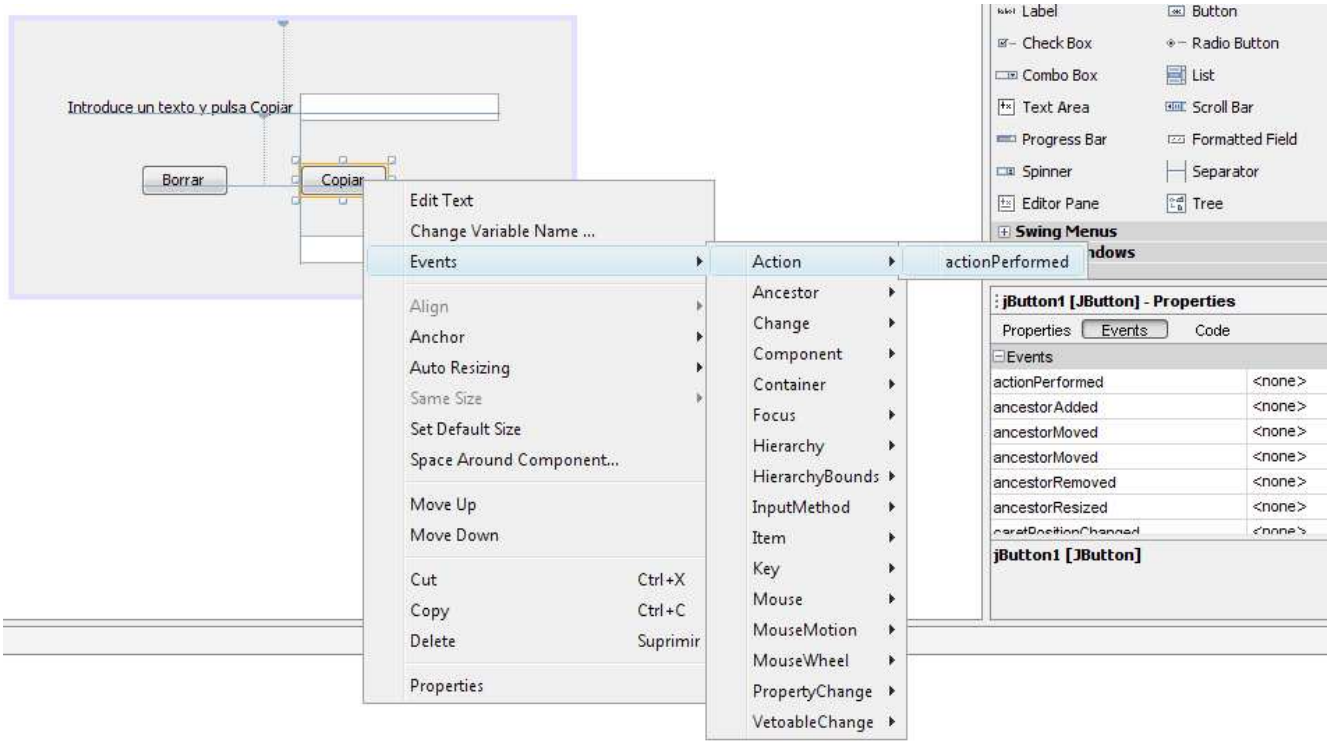


La apariencia final de la ventana es la siguiente:



Para realizar la aplicación tendremos que ir arrastrando componentes a la ventana. NetBeans nos proporciona un Layout propio que irá situando los componentes en el contenedor en las posiciones y tamaños que nosotros deseemos. Situaremos un JLabel con el texto correspondiente, dos JTextField (en uno de ellos no se permite la edición) y dos botones.

Para el tratamiento de los eventos tenemos dos opciones: una es pulsar dos veces en el botón lo que nos llevará directamente al código de tratamiento del evento, la otra forma es pulsar con el botón derecho del ratón en cada uno de los botones y seleccionar *"Events – Action – actionPerformed"* y rellenar con el siguiente código marcado en negrita.



En las propiedades del JFrame debemos poner como título de la ventana "BotonesTexto" y desmarcar la casilla "resizable". Para que la ventana salga centrada tenemos que pulsar en la pestaña código y seleccionar "Form Size Policy - Generate Resize Code".

```
import javax.swing.JOptionPane;
public class BotonesTexto extends javax.swing.JFrame {

    /** Creates new form BotonesTexto */
    public BotonesTexto() {
        initComponents();
    }

    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
            ...
    }// </editor-fold>

    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        jTextField1.setText("");
        jTextField2.setText("");
    }

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        String txt = jTextField1.getText();
        if (txt.equals("")) {
                JOptionPane.showMessageDialog(this,"Rellene                    el
texto.","Mensaje",JOptionPane.INFORMATION_MESSAGE);
                return;
        }
        jTextField2.setText(txt);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
```

```
            public void run() {
                new BotonesTexto().setVisible(true);
            }
        });
    }

    // Variables declaration - do not modify
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JTextField jTextField1;
    private javax.swing.JTextField jTextField2;
    // End of variables declaration

}
```

1.2 Ejercicio 1: Botones

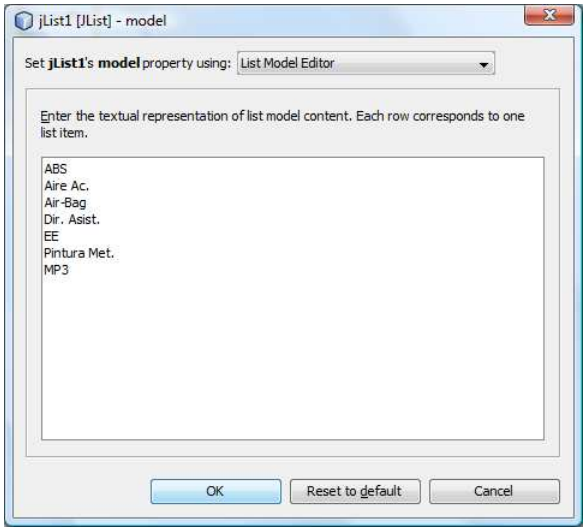
Diseñar una ventana que contenga dos cajas de texto y dos botones, uno etiquetado con "MAYÚSCULAS" y otro con "minúsculas". Cuando se pulse el botón de mayúsculas se deberá convertir el contenido de la primera caja de texto a mayúsculas y si se pulsa el otro a minúsculas y presentarlo en la segunda caja de texto. Si no hay texto se debe presentar un mensaje de error.

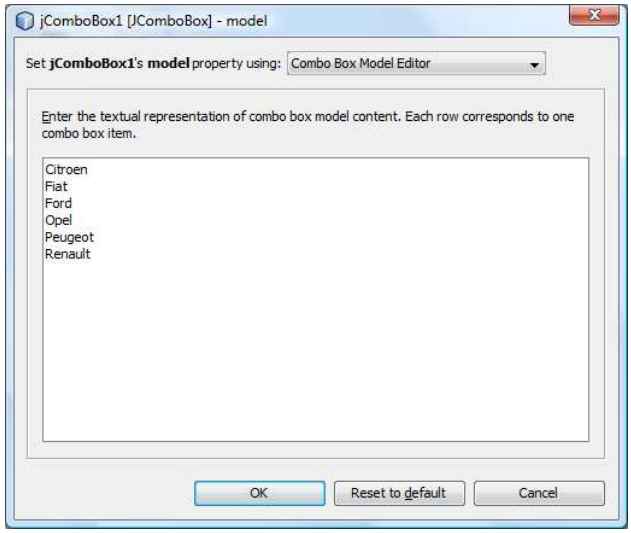
1.3 Listas

En este ejemplo utilizaremos las listas como componentes principales. Para realizarlo tenemos que crear en el proyecto un nuevo fichero de tipo JFrame.

La apariencia final de la ventana es la siguiente:

En la ventana de aplicación pondremos un JLabel con el texto correspondiente, un JComboBox, un JList y un JTextField (no se permite la edición). Para dar los datos iniciales de las listas debemos abrir sus propiedades y editar el campo "model".





Para el tratamiento de los eventos, o pulsamos dos veces sobre los componentes o pulsamos con el botón derecho del ratón en cada una de las listas y seleccionamos: para el JComboBox: *"Events – Action – actionPerformed"* y para el JList: *"Events – ListSelection – valueChanged"*, por último, rellenamos con el código en negrita.

En las propiedades del JFrame debemos poner como título de la ventana "Listas" y desmarcar la casilla "resizable". Para que la ventana salga centrada tenemos que pulsar en la pestaña código y seleccionar "Form Size Policy - Generate Resize Code".

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

    /** Creates new form Listas */
    public Listas() {
        initComponents();
        jComboBoxMarcas.setSelectedIndex(0);
    }

    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
            ...
    }// </editor-fold>

    private String marca_selec, equipamiento;

    private void jListEquipValueChanged(javax.swing.event.ListSelectionEvent evt) {
// TODO add your handling code here:
        //borramos texto e inicializamos equipamiento
        jTextField1.setText("");
        equipamiento = " - ";
        //capturamos los items seleccionados
        //y los convertimos en cadena
        List<String> equipo_selec = jListEquip.getSelectedValuesList();
        Iterator items = equipo_selec.iterator();
        while (items.hasNext()) {
            equipamiento += items.next() + " - ";
        }
        //presentamos la selección de marca y equipamiento
        jTextField1.setText(marca_selec + equipamiento);
    }

    private void jComboBoxMarcasActionPerformed(java.awt.event.ActionEvent evt) {
// TODO add your handling code here:
        //borramos texto
        jTextField1.setText("");
        //deseleccionamos el equipamiento
```

```
        jListEquip.clearSelection();
        //capturamos la marca seleccionada
        marca_selec = "Marca: " + (String) jComboBoxMarcas.getSelectedItem() + " # ";
        jTextField1.setText(marca_selec);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Listas().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JComboBox jComboBoxMarcas;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JList jListEquip;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration
}
```

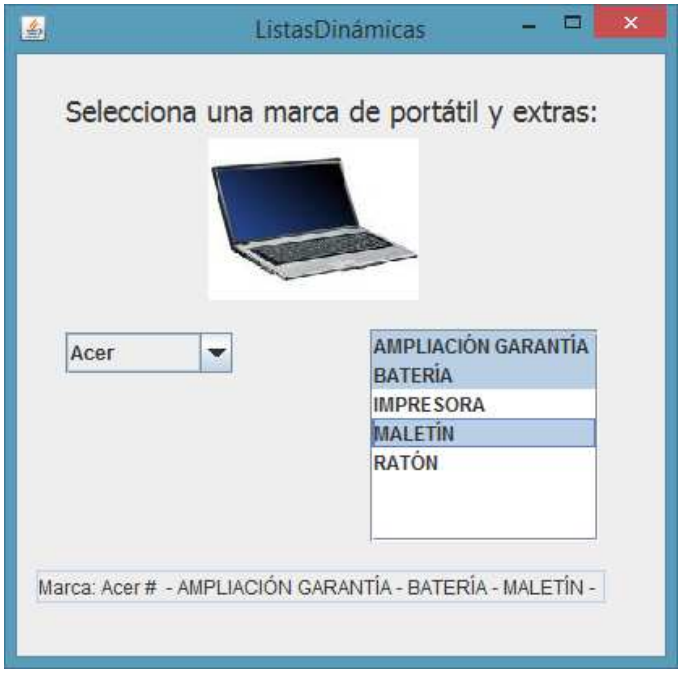
1.4 Ejercicio 2: Calculadora Swing

Escribir una aplicación de ventana Swing que permita realizar las operaciones típicas de una calculadora: suma, resta, multiplicación y división. El usuario introducirá los dos operandos en sendas cajas de texto (se pueden utilizar cajas de texto con formato) y luego seleccionará la operación a realizar (mediante una lista o botones o cualquier componente que permita seleccionar una opción de entre las cuatro posibles), mostrando el resultado correspondiente en otra caja de texto. También debe ofrecer la posibilidad de convertir el resultado de euros a dólares y viceversa (se puede seleccionar la divisa a través de una lista desplegable o un componente equivalente). El programa deberá tratar todos los posibles errores en tiempo de ejecución, dando un mensaje con un diálogo de error. El formato de la interfaz es libre.

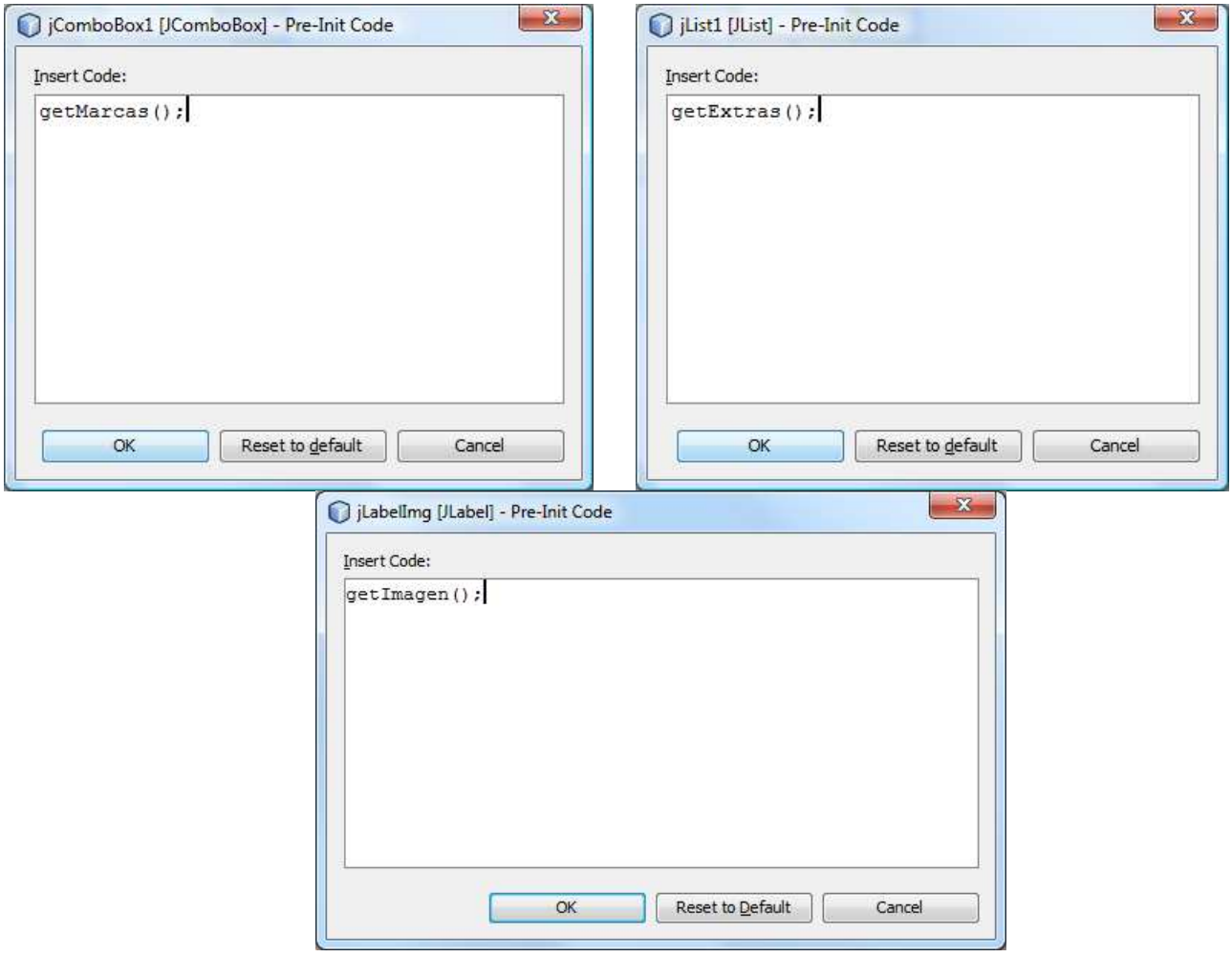
1.5 Listas Dinámicas

En este ejemplo utilizaremos las listas como componentes principales, pero en este caso las opciones que aparecen en las listas se recogerán de ficheros de texto para poder hacer cambios de forma dinámica (el mismo mecanismo se podría utilizar accediendo a datos de una base de datos. Para realizarlo tenemos que crear en el proyecto un nuevo fichero de tipo JFrame.

En la ventana de aplicación pondremos dos JLabel, uno con el texto correspondiente y el otro para la imagen, un JComboBox, un JList y un JTextField (no se permite la edición).



Para dar los datos iniciales de las listas debemos abrir sus propiedades y en la pestaña "code" ir a "Pre-Init Code" y llamar a los métodos correspondientes como se muestra a continuación. Para cargar la imagen del portátil también debemos llamar a un método de inicialización.



Para el tratamiento de los eventos, o pulsamos dos veces sobre los componentes o pulsamos con el botón derecho del ratón en cada una de las listas y seleccionamos: para el JComboBox *"Events – Action –*

actionPerformed" y para el JList *"Events – ListSelection – valueChanged"*, por último, rellenamos con el código en negrita.

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import javax.swing.ImageIcon;
import javax.swing.JComboBox;

public class ListasDinamicas extends javax.swing.JFrame {

    /** Creates new form ListasDinamicas */
    public ListasDinamicas() {
        initComponents();
        jComboBoxMarcas.setSelectedIndex(0);
    }

    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
            ...
    }// </editor-fold>

    private String marca_selec, extras;

    private void jComboBoxMarcasActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        //borramos texto
        jTextField1.setText("");
        //deseleccionamos el equipamiento
        jListExtras.clearSelection();
        //capturamos la marca seleccionada
        marca_selec = "Marca: " + (String) jComboBoxMarcas.getSelectedItem() + " # ";
        jTextField1.setText(marca_selec);
    }

    private void jListExtrasValueChanged(javax.swing.event.ListSelectionEvent evt) {
        // TODO add your handling code here:
        //borramos texto e inicializamos equipamiento
        jTextField1.setText("");
        extras = " - ";
        //capturamos los items seleccionados
        // y los convertimos en cadena
        List<String> extras_selec = jListExtras.getSelectedValuesList();
        Iterator items = extras_selec.iterator();
        while (items.hasNext()) {
            extras += items.next() + " - ";
        }
        //presentamos la selección de marca y equipamiento
        jTextField1.setText(marca_selec + extras);
    }

    // cargamos la imagen del portátil
    private void getImagen() {
        try {
            jLabelImg.setSize(130, 100);
            ImageIcon imagen = new ImageIcon("imagenes/portatil.jpg");
            //Se redimensiona
            ImageIcon               imgRedimensionada               =               new
ImageIcon(imagen.getImage().getScaledInstance(jLabelImg.getWidth(),
jLabelImg.getHeight(), 1));
            jLabelImg.setIcon(imgRedimensionada);
        } catch (Exception e) {
            System.out.println("Error: " + e.toString());
        }
    }
```

```
    // rellenamos las opciones de la lista desplegable
    private void getMarcas() {
        String cad;
        ArrayList<String> marcas = new ArrayList<String>();
        try {
            FileInputStream fis = new FileInputStream("marcas.txt");
            InputStreamReader is = new InputStreamReader(fis, "ISO-8859-1");
            BufferedReader br = new BufferedReader(is);
            while ((cad = br.readLine()) != null) {
                marcas.add(cad);
            }
            //Cerramos el stream
            br.close();
            jComboBoxMarcas = new JComboBox(marcas.toArray());
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }

    // rellenamos las opciones de la lista
    private void getExtras() {
        String cad;
        ArrayList<String> extrasAL = new ArrayList<String>();
        try {
            FileInputStream fis = new FileInputStream("extras.txt");
            InputStreamReader is = new InputStreamReader(fis, "ISO-8859-1");
            BufferedReader br = new BufferedReader(is);
            while ((cad = br.readLine()) != null) {
                extrasAL.add(cad);
            }
            //Cerramos el stream
            br.close();
            jListExtras.setListData(extrasAL.toArray());
        } catch (Exception e) {
            System.out.println(e.toString());
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {

            public void run() {
                new ListasDinamicas().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JComboBox jComboBoxMarcas;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabelImg;
    private javax.swing.JList jListExtras;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextField jTextField1;
    // End of variables declaration
}
```

1.6 Ejercicio 3: Imágenes (ejercicio obligatorio para II y SI, voluntario para IC)

Diseñar una ventana que contenga una lista desplegable con nombres de imágenes y cuando se pulse una de ellas se muestre en una etiqueta de la ventana. Las imágenes deben estar almacenadas en un directorio llamado "imagenes" que se encuentra en la raíz del proyecto. La lista de imágenes se debe cargar o desde un fichero de texto que contiene los nombres de las imágenes o bien sacando el listado de los ficheros del propio directorio.
Nota: En el directorio "imágenes" del proyecto de ejemplos de la sesión hay una serie de fotos que se pueden utilizar.

2. POO Y SWING

El siguiente ejemplo muestra el uso de la herencia con clases abstractas y polimorfismo y su gestión desde una ventana Swing.

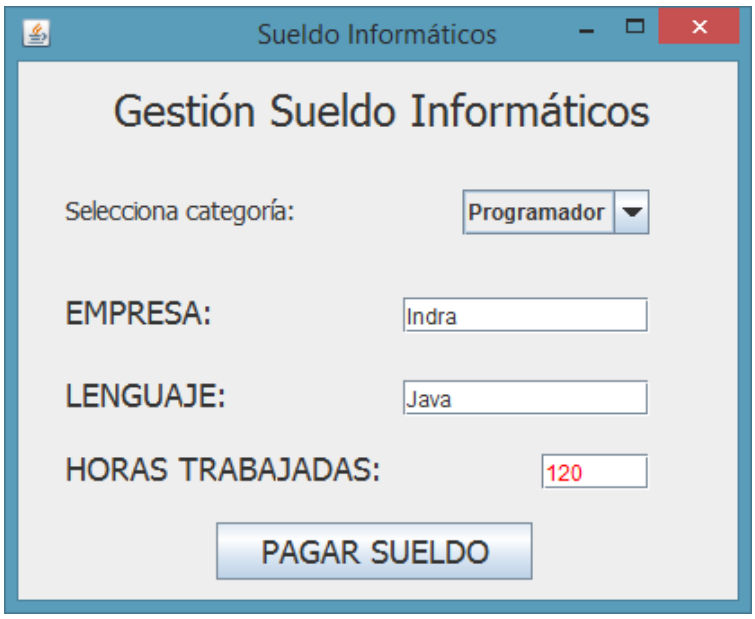
Lo primero que se debe desarrollar antes de la interfaz gráfica son las clases de lógica de negocio que en este ejemplo representan distintas categorías de informáticos a los que hay que pagar el sueldo. Para realizarlo se ha creado una jerarquía de clases donde tenemos una clase padre llamada InformaticoAbs que representa una clase abstracta que contiene un método abstracto "pagarSueldo" que se debe implementar en las clases hijas que es donde se determina a cuanto se paga la hora trabajada dependiendo de la categoría del informático y que dará lugar a utilizar polimorfismo en tiempo de ejecución.

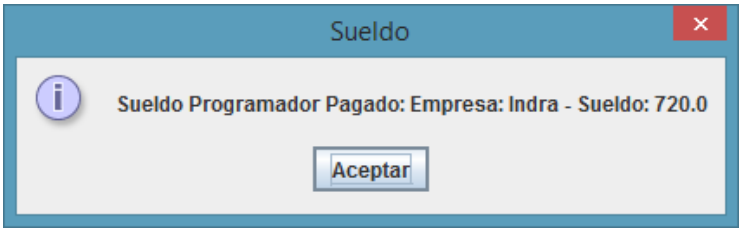
```
public abstract class InformaticoAbs {

    //Atributos
    private String empresa;
    private double sueldo;

    //Constructor
    public InformaticoAbs(String empresa) {
        this.empresa = empresa;
    }

    //Métodos
    public String getEmpresa() {
        return this.empresa;
    }

    public void setEmpresa(String nueva) {
        this.empresa = nueva;
    }

    public double getSueldo() {
        return this.sueldo;
    }

    public void setSueldo(double euros) {
        this.sueldo = euros;
    }

    @Override
    public String toString() {
        return "Empresa: " + this.empresa + " - Sueldo: " + this.sueldo;
    }

    //método abstracto
    public abstract double pagarSueldo(int horas);
}
```

```
public class Analista extends InformaticoAbs {

    private String especialidad;

    public Analista(String empresa, String especialidad) {
        super(empresa);
        this.especialidad = especialidad;
    }

    public String getEspecialidad() {
        return especialidad;
    }

    public void setEspecialidad(String especialidad) {
        this.especialidad = especialidad;
    }

    @Override
    public double pagarSueldo(int horas) {
        return (double) horas * 12;
    }
}
```

```
public class Programador extends InformaticoAbs {

    private String lenguaje;

    public Programador(String empresa, String lenguaje) {
        super(empresa);
        this.lenguaje = lenguaje;
    }

    public String getLenguaje() {
        return lenguaje;
    }

    public void setLenguaje(String lenguaje) {
        this.lenguaje = lenguaje;
    }

    @Override
    public double pagarSueldo(int horas) {
        return (double) horas * 6;
    }
}
```

Lo siguiente a desarrollar es la interfaz gráfica que tiene la siguiente apariencia:





En el código fuente se puede observar la utilización del polimorfismo para pagar los sueldos:

```
import javax.swing.JOptionPane;

public class VentanaPrincipal extends javax.swing.JFrame {

    private String tipo = "Analista"; //tipo de informático

    /** Creates new form VentanaPrincipal */
    public VentanaPrincipal() {
        initComponents();
        jComboBoxInf.setSelectedIndex(0);
    }

    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {
      ...
    }// </editor-fold>

    private void jComboBoxInfActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        tipo = (String) jComboBoxInf.getSelectedItem();
        if (tipo.equals("Analista")) {
            jLabelVar.setText("ESPECIALIDAD:");
        } else {
            jLabelVar.setText("LENGUAJE:");
        }
    }

    private void jButtonPagarActionPerformed(java.awt.event.ActionEvent evt) {
        // TODO add your handling code here:
        try {
            String empresa = jTextFieldEmpresa.getText();
            String var = jTextFieldVar.getText();
```

```
            int horas = ((Long)jFormattedTextField1.getValue()).intValue();

            if (tipo.equals("Analista")) {
                Analista a = new Analista(empresa, var);
                //aplicamos el método polimórfico
                double sueldo = a.pagarSueldo(horas);
                a.setSueldo(sueldo);
                JOptionPane.showMessageDialog(this, "Sueldo Analista Pagado: " +
a.toString(), "Sueldo", JOptionPane.INFORMATION_MESSAGE);
            } else {
                Programador p = new Programador(empresa, var);
                //aplicamos el método polimórfico
                double sueldo = p.pagarSueldo(horas);
                p.setSueldo(sueldo);
                JOptionPane.showMessageDialog(this, "Sueldo Programador Pagado: " +
p.toString(), "Sueldo", JOptionPane.INFORMATION_MESSAGE);
            }
        } catch (Exception e) {
            JOptionPane.showMessageDialog(this, "Error al pagar sueldo: " +
e.toString(), "Mensaje", JOptionPane.ERROR_MESSAGE);
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new VentanaPrincipal().setVisible(true);
            }
        });
    }
    // Variables declaration - do not modify
    private javax.swing.JButton jButtonPagar;
    private javax.swing.JComboBox jComboBoxInf;
    private javax.swing.JFormattedTextField jFormattedTextField1;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabelVar;
    private javax.swing.JTextField jTextFieldEmpresa;
    private javax.swing.JTextField jTextFieldVar;
    // End of variables declaration
}
```

2.1 Ejercicio Cuentas Swing

Realizar una pequeña aplicación en Java utilizando polimorfismo, como en el ejemplo anterior, para la gestión de los intereses de distintos tipos de cuentas de un banco. La aplicación se compondrá de una clase padre abstracta *CuentaAbs* que contendrá un método abstracto *calculaInteres()* que devolverá un double con el cálculo del interés aplicado al saldo actual de la cuenta, es decir, nos devolverá como quedaría el saldo de la cuenta si se aplicara ese interés. La clase *CuentaAbs* debe contar como mínimo con los atributos: número de cuenta (String), titular (String), saldo (double) y fecha de apertura (LocalDate) y sus correspondientes métodos get y set. De la clase *CuentaAbs* heredarán dos clases llamadas *CuentaCorriente* y *CuentaPlazo* que implementarán el método abstracto dando un interés del 1.5 % la primera y un 3.5 % la segunda. La clase *CuentaCorriente* incorporará un nuevo atributo llamado *numTarjCredito* de tipo long y *CuentaPlazo* el atributo *numAños* de tipo int.

Para probar las clases anteriores se debe realizar una interfaz gráfica que permita crear distintos objetos de los dos tipos de cuentas y aplicarles el interés correspondiente al saldo que tienen.

2.2 Ejercicio Aparcamiento Swing

Realizar la aplicación del Aparcamiento vista en la sesión 4 y 6, reutilizar las clases de lógica de negocio de la sesión 6 y realizar una aplicación Swing de forma que permita la gestión del aparcamiento. Se tienen que distinguir dos funcionalidades distintas en la aplicación, por un lado, se podrá introducir un nuevo vehículo en el aparcamiento indicando todos sus datos, y por otro se podrá sacar un vehículo del aparcamiento con tan solo su matrícula. Para esto último se pueden dar dos opciones, o poner en una caja de texto la matrícula del vehículo o seleccionarla de una lista despegable.