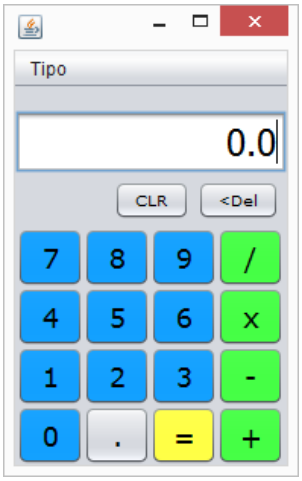
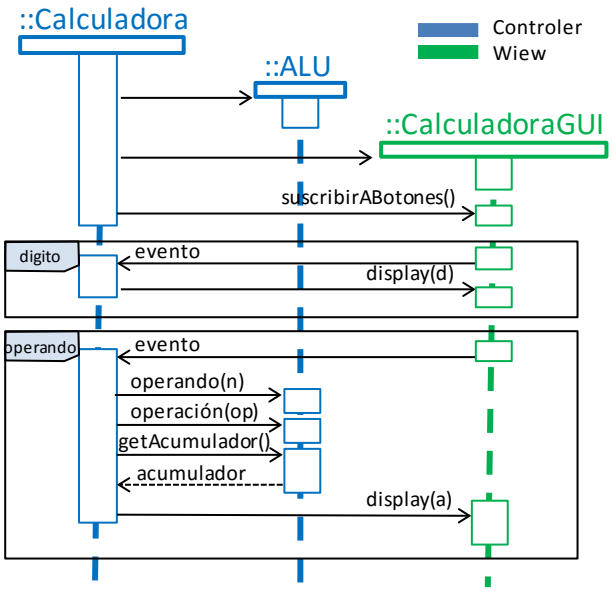
TAP Labo 30/10/2015

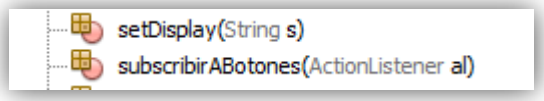


Vamos a usar (o generar) nuestro interfaz de calculadora para hacer una versión que funcione, y con una arquitectura razonable.

La aplicación arrancará de una clase de control que representa a la calculadora electrónica y comienza por generar su propia ALU y por poner en marcha el GUI (instanciar un objeto de nuestra clase gráfica). El código para esto último es un tanto "truculento", por lo que se da en la imagen que se encuentra al final de la página (se explicará más adelante. Es una simplificación de lo que pone Netbeans en nuestra clase gráfica -main-, que debe quitarse y puede traerse a la clase de control "calculadora")



Antes de ver el controlador, pensemos en el JFrame: no vamos a poner ninguna atención a eventos de los botones; dejaremos que de eso se encargue el controlador. El JFrame será sólo "fachada" (bueno, y lanzador de eventos). Será el controlador quien se encargue de atender eventos y decirle al JFrame qué poner en el display, y para ello el JFrame ha de proporcionarle estos dos métodos:



que son el **PROBLEMA 1**

Pistas:

"set Display" es muy fácil, no necesita pistas...

"suscribirABotones" debe recorrer todos los paneles que contengan botones para que a todos los botones contenidos se les suscriba el objeto que le llega como parámetro. Métodos a usar:

jp.getComponentCount() al JPanel (jp) podemos pedirle el número de componentes que tiene;
jp.getComponent(i) al JPanel (jp) podemos pedirle el componente i-ésimo
b.addActionListener(al) a un botón b podemos suscribirle un ActionListener (al)

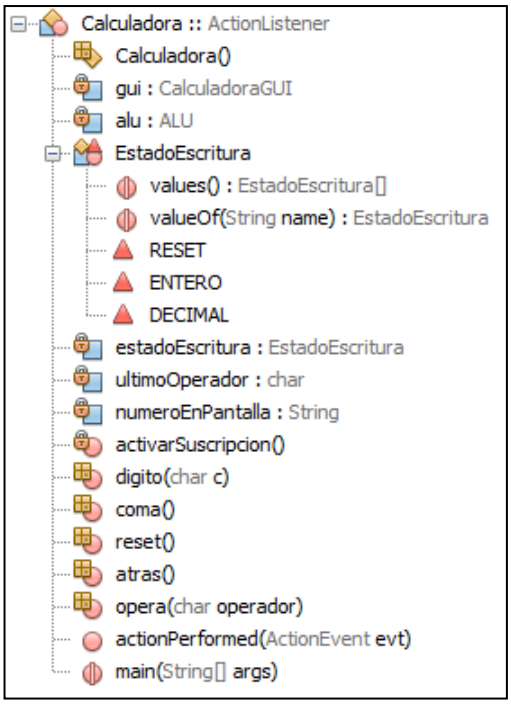
De este modo podemos centrarnos en el controlador.
Por mantener la estructura vista en clase, con una ALU muy simple, podemos mantener aquella:

```
Calculadora() throws Exception {
    //Instancia y muestra el GUI y espera a que esté hecho.
    java.awt.EventQueue.invokeAndWait(new Runnable() {
        @Override public void run() {(gui = new CalculadoraGUI()).setVisible(true);}
    });
}
```

```java
/**
 * Arithmetic-Logical Unit (en realidad sólo aritmética y muy sencilla).
 *
 * Arquitectura basada en acumulador (como las primeras ALUs)
 * Sólo ejecuta las cuatro operaciones aritméticas básicas.
 * Trabaja con aritmética decimal en base 10.
 *
 * @author German
 * @version 1.0 (oct 2012)
 */
public class ALU {
    private double acumulador=0.0;
    private double registro=0.0;

    double getAcumulador() { return acumulador; }
    void   clear()         { acumulador = 0.0; }
    void   setRegistro(double registro) { this.registro = registro; }
    double suma()          { return acumulador+=registro; }
    double resta()         { return acumulador-=registro; }
    double multiplica()    { return acumulador*=registro; }
    double divide()        { return acumulador/=registro; }
}
```

Vamos con el controlador: puede basarse en el visto en clase. Como puede verse, la enumeración interna que pusimos para controlar los estados del número en el Display tiene un par de métodos que no escribimos, y es que se añaden "de oficio", no es preciso escribirlas.



Para escribir el método "actionPerformed" tendremos que "descubrir" cuál ha sido el botón pulsado, cosa que podemos hacer "mirando dentro" del ActionEvent que recibimos... lo que es el **PROBLEMA 2**

Pistas: podremos descubrir quien ha sido el componente gráfico origen ("source") , que será un botón, y a este pedirle el texto que muestra en el GUI, con lo cual decidir a qué método llamar.