

UNIVERSIDAD DE CORDOBA

FACULTAD DE INGENIERIAS

PROGRAMA INGENIERIA DE SISTEMAS

CURSO: Programación I

TEMA: Arreglos o Vectores en Java.

DESCRIPCION: En este documento se aborda el tema de los arreglos o vectores unidimensionales, su conceptualización, características e implementación en el lenguaje Java. Para este último efecto se hace un tratamiento de las reglas sintácticas que el lenguaje provee tanto para la declaración, inicialización y acceso a los elementos de un arreglo, así como también otras características relacionadas tales como la propiedad *length* de un arreglo, el uso de arreglos con contenido explícito. Por otra parte la demostración práctica del concepto de arreglo se desarrollara con una aplicación o proyecto de ventana (interfaz gráfica de usuario), usando el IDE *NetBeans* y los controles visuales de la paleta *Swing*. La implementación se hace aplicando el modelo de orientación a objetos y el diseño de clases en UML.

OBJETIVO: Diseñar en UML e implementar en el lenguaje Java una clase para ilustrar la declaración, inicialización, acceso y otras operaciones sobre vectores unidimensionales, acorde a un problema concreto planteado y usando para ello una aplicación de ventana construida con el IDE *NetBeans*.

PALABRAS CLAVES: Arreglos en Java, arreglos de contenido explícito, arreglos como atributos de una clase en *Java*, aplicaciones de GUI en *Java*, instancias de clases en *Java*, controles de *Swing*.

Luis Roberto Olascoaga Surmay

1. Arreglos o vectores

Para introducir el concepto de arreglo (también conocido como vector unidimensional por tener una sola dimensión) y entender su importancia analicemos la siguiente característica de las variables y atributos que hasta ahora hemos usado. En una variable o atributo de los tipos de datos que hemos estudiado anteriormente, solo podemos almacenar un valor cada vez, un solo valor al mismo tiempo. El valor de una variable o de un atributo, por definición puede cambiar en cualquier momento y cuando esto sucede, el valor anterior se pierde al ser reemplazado por el nuevo dato. Significa que no es posible mantener un "historial" de los valores que se han almacenado anteriormente en una variable.

Consideremos ahora la situación de almacenar la nota definitiva de un estudiante en una variable cualquiera de tipo real (*float*). En dicha variable solo podemos almacenar la nota de un único estudiante, con lo cual si quisiéramos guardar las notas definitivas de todo un curso, deberíamos declarar una variable para cada estudiante en las cuales se almacenen las notas definitivas de cada uno de ellos. Está claro que esta es una situación bastante engorrosa; no solo por la cantidad de variables a definir y por la codificación, compleja por demás, que sería necesaria hacer para gestionar cada variable sin que se presenten errores, por ejemplo, en asignar la nota correspondiente de cada estudiante; sino también por los posibles cambios que pudieran surgir por el ingreso de nuevos estudiantes o por el retiro de algunos, lo cual nos obligaría a modificar gran parte del código. Los arreglos vienen a resolver de manera eficiente y sencilla problemas como el descrito.

Los arreglos son un tipo de datos personalizado, es decir, definidos por nosotros mismos y en el cual se puede almacenar más de un valor al mismo tiempo en una misma variable o en un mismo atributo. Decimos que es un tipo personalizado porque somos nosotros los que indicamos el tipo y la cantidad de datos a contener en el arreglo, así de forma similar a como definimos una clase en la cual es usted quien decide que atributos y métodos va a tener; claro está, en ambos casos todo depende del problema que estemos abordando. De esta manera con un arreglo declaramos variables o atributos en las que puedo contener cualquier cantidad de datos o valores en forma simultánea en una misma o única variable. Por ello una característica a tener en cuenta en un arreglo es su tamaño, que define la capacidad de almacenamiento del mismo; que no es otra cosa que el número de elementos que puede contener en un momento dado. Otra característica del arreglo es que cada dato contenido en

él, debe ser del mismo tipo, razón por la cual decimos que los arreglos son una estructura de datos homogéneas en cuanto al tipo de datos de su contenido. Con una variable de tipo arreglo así definida podemos contener un conjunto de valores, de modo que accedemos a cualquiera de ellos de forma individual; siempre haciendo uso de la misma variable o atributo; evitando así tener que declarar una variable individual para cada dato del conjunto. Con ello para el problema de las notas definitivas de un curso señalado anteriormente, solo necesitaríamos definir un arreglo de números reales (*float*) con tantos elementos como estudiantes tenga el curso.

2. Sintaxis de Arreglos en Java

La sintaxis seguida por *Java* para declarar arreglos es en esencia la misma empleada para declarar atributos o variables normales (recuerde que un arreglo en últimas es también una variable con otras características adicionales); por lo tanto, para definir una variable de tipo arreglo indicamos el tipo de datos a contener en el arreglo, después el nombre de variable de arreglo y seguidamente un par de corchetes []; sin indicar nada entre ellos, que es lo que precisamente diferencia la declaración de un arreglo de una variable "normal". Así la sintaxis para declarar arreglos en *Java* es:

```
Tipo_De_Datos Nombre_Arreglo [ ];
```

Así por ejemplo, en las siguientes líneas de código se declara un vector de cadenas de caracteres llamado *Nombres* y otro de números reales llamado *Notas*:

```
String Nombres [ ];
```

```
float Notas [ ];
```

Observe que el tamaño no se especifica dentro de los corchetes, como sucede en algunos otros lenguajes, por lo que decimos que los arreglos en *Java* son dinámicos en su tamaño; es decir, que la cantidad de elementos que contienen puede cambiar en cualquier momento de acuerdo a sus necesidades, ampliándola o disminuyéndola en tiempo de ejecución.

2. Inicialización de un arreglo.

Esta es la operación mediante la cual establecemos el tamaño o cantidad de elementos que tendrá un arreglo; considerando además, que en algunos casos también podemos definir el

contenido de sus elementos como en la inicialización explícita. En *Java* existen diferentes mecanismos con los cuales inicializar un arreglo, veamos cada uno de ellos.

a) Inicialización de un arreglo con *new*.

La primera forma de inicializar un arreglo es usando el operador ***new***, lo cual se hace siguiendo la siguiente sintaxis:

```
Tipo_De_Datos Nombre_Arreglo[ ] = new Tipo_De_Datos [ Tamaño ];
```

Donde *Tipo_De_Datos* indica el dominio de los valores a guardar en el arreglo; por supuesto que el tipo de datos indicado a la derecha (después de ***new***), debe ser el mismo empleado en el lado izquierdo. En cuanto al tamaño debe ser un número entero positivo; ya sea un valor puntual explícito (como el 10, 20 18 etc.), una variable o atributo de tipo entero, un método que retorne un valor entero, pero en general cualquier expresión que de como resultado un número entero. Veamos algunos ejemplos al respecto:

```
int VecE[ ] = new int [10];
```

```
float VecR[ ] = new float [N];
```

```
char VecC[ ] ;
```

(... otras instrucciones cualesquiera...)

```
VecC[ ] = new char [15];
```

En el primer caso declaramos un arreglo de tipo entero y lo inicializamos con un tamaño de diez elementos. El segundo caso define un arreglo de números reales, pero esta vez su tamaño no es explícito, asumimos por supuesto que *N* es una variable o atributo de tipo entero que previamente se le ha asignado un número entero positivo cualquiera. En estos dos primeros ejemplos, vemos como declarar arreglos que inmediatamente se inicializan con ***new***, dando su tamaño respectivo. Por el contrario para el tercer caso, declaramos el arreglo de tipo carácter (***char***) pero no lo inicializamos enseguida. Esto es posible ya que podemos inicializar el arreglo en un punto o línea de código diferente a aquella en la que se declara. Suponemos entonces que en otra parte del código inicializamos este vector con el operador ***new***, dándole un tamaño de 15; claro está, que este valor no necesariamente tiene que ser explícito como en el ejemplo señalado.

Cabe señalar, que en los tres casos, es posible volver a inicializar el arreglo en cualquier momento, cambiando así su tamaño actual; para este efecto, se sigue la misma sintaxis y especificaciones indicadas anteriormente. Esta característica es la que precisamente hace que los vectores sean dinámicos en tamaño.

b) Inicialización de un arreglo con null.

Por otra parte el valor **null** (nulo, vacío o nada) se puede usar para inicializar un vector, ya sea en el momento en que se declara o en punto posterior a ello. Téngase en cuenta que cuando hemos inicializado un vector con **new** y luego le asignamos el valor **null**, el efecto es que se libera el espacio ocupado en memoria por el arreglo y que su nuevo tamaño será cero; en cuyo estado no está permitido acceder a ningún elemento del vector, puesto que esta vacío. Veamos estos ejemplos al respecto:

```
long VecL[ ] = null;
```

```
String VecS[ ] = new String[25];
```

```
(... otras instrucciones cualesquiera...)
```

```
VecS[ ] = null;
```

c) Inicialización explícita de un arreglo.

A parte de los mecanismos señalados anteriormente, tenemos otra forma de inicializar un arreglo, la cual se suele llamar inicialización explícita; que se emplea cuando conocemos el conjunto de valores que debe almacenar el vector. Esta inicialización debe hacerse obligadamente en el momento en el que se declara el arreglo, pues después no es posible cambiar el tamaño del vector ni con **new** ni tampoco con el valor **null**; sin embargo, es completamente válido cambiar el contenido de sus elementos. Para efectuar esta forma de inicialización de un arreglo, se define sus elementos entre llaves separando cada valor por comas. El número de elementos de la lista especificada determina el tamaño del vector. Por supuesto que los elementos enlistados deben ser del mismo tipo de datos del arreglo. Veamos entonces los siguientes ejemplos:

```
char Vocales[ ] = {'a','e','i','o','u'};
```

```
int Dígitos[ ] = {0,1,2,3,4,5,6,7,8,9};
```

```
String Dias[ ] = {"lunes", "martes", "miércoles", "jueves", "viernes", "sábado", "domingo"}
```

El primer vector es de tipo carácter y almacena el valor de las letras vocales. Como puede verse se listan cinco letras; por lo tanto su tamaño es de cinco. Para el segundo ejemplo tenemos un arreglo de tipo entero que contiene diez números y su tamaño será de diez. En tercer caso, tenemos un arreglo de cadenas que guarda los nombres de los siete días de la semana. Como una característica importante de los arreglos en *Java*, tenemos que cada vez que declaramos uno el lenguaje gestiona de forma automática un atributo de tipo entero en el que se guarda el tamaño actual del vector. Este atributo recibe el nombre de ***length*** y es de solo lectura; lo que significa que no podemos asignarle un valor a dicho atributo en forma directa. Este valor cambia con las operaciones de ***new*** y asignación de ***null*** y se calcula automáticamente en una inicialización explícita. En este orden de ideas, siguiendo las tres declaraciones anteriores, las siguientes líneas imprimen los valores de 5, 10 y 7 como tamaños respectivos de los arreglos en cuestión:

```
System.out.println(Vocales.length);
```

```
System.out.println(Digitos.length);
```

```
System.out.println(Dias.length);
```

3. Acceso a los elementos de un arreglo.

Existe un mecanismo para poder acceder al contenido de un arreglo, ya sea para asignar datos al mismo o para emplear sus valores en alguna operación. Para este efecto, los datos de un arreglo se almacenan en posiciones o "casillas" individuales; razón por la cual, la gestión u operaciones con los arreglos se reducen a hacer referencia a sus elementos en forma individual; es decir, cada elemento se guarda en una única posición o índice que está representado a su vez por un número entero. En *Java* el primer elemento se encuentra en la posición número 0, el segundo en la numero 1, el tercero en la numero 2 y así sucesivamente siguiendo una enumeración consecutiva, en la que los índices empiezan en cero y se incrementan automáticamente de uno en uno. Esta indexación, al ser automática no podemos cambiarla, ya es algo que viene predeterminado por el lenguaje *Java*.

A la luz de la sintaxis en *Java*, hacemos referencia a un elemento del arreglo indicando el nombre del arreglo y entre corchetes [] la posición o índice del elemento al cual queremos acceder. Este valor de posición debe ser un entero número positivo comprendido entre 0 y el

tamaño del arreglo menos 1, ya que como se ha dicho la indexación de los elementos empieza en cero. El valor del índice puede ser un valor explícito, o el contenido de una variable, parámetro, atributo de tipo entero, así como también el resultado de un método o expresión entera. Considerando el atributo **length**, que es implícito a cualquier vector, podemos establecer como regla que una posición válida en un vector, toma un valor comprendido entre 0 y **length** – 1. Para aclarar más este punto, considere el ejemplo anterior con un vector inicializado explícitamente:

```
char Vocales[ ] = {'a','e','i','o','u'};
```

Gráficamente al vector le corresponde la siguiente estructura como la siguiente:



De esta manera, los números en la parte superior son los índices o posiciones que son fijos y empiezan en cero. En cada casilla se guarda un elemento (en este caso un carácter); por lo tanto, representan el contenido almacenado por el vector. Así, si queremos imprimir en consola el valor del elemento de la posición número cuatro haremos esto:

```
System.out.println(Vocales[4]);
```

Un elemento de una posición cualquiera puede participar en cualquier expresión donde se espere o necesite un dato del tipo del arreglo. Por ejemplo:

```
if(Vocales[2]=='e'){  
    System.out.println("La posición 2 tiene la letra e");  
}
```

```
Vocal [1]= 'k'; //Ahora ya no es una vocal, toma el valor de k
```

Normalmente se hace necesario acceder a todos los elementos del vector, “visitando” los elementos de todas sus posiciones para hacer algo con ellos. En estos casos se usan ciclos, que recorren el arreglo; principalmente el ciclo para (**for**). Por ejemplo, si queremos imprimir en consola todo el vector anterior, lo podemos hacer así:

```
int i;  
  
for(i=0;i<5;i++){  
    System.out.println(Vocales[i]);  
}
```

O también:

```
int i;  
  
for(i=0;i<Vocales.length;i++){  
    System.out.println(Vocales[i]);  
}
```

En donde podemos observar, que las posiciones del arreglo están dadas por una variable (i) cuyos valores son asignados por el ciclo **for**. Para resumir y también a manera de conclusión del tema de los arreglos, presentamos la siguiente lista de características relacionadas con este concepto:

- ✓ Es un tipo de datos que permite almacenar "n" valores de un mismo tipo en una misma variable.
- ✓ Todo arreglo tiene un tamaño en un momento dado, el cual representa el número de datos que contiene, es decir, su capacidad de almacenamiento.
- ✓ A los datos o valores contenidos o almacenados en el arreglo también se les llama elementos del arreglo.
- ✓ Cada dato del arreglo está contenido en una posición, que hace referencia al número de celda o casilla donde se guarda.
- ✓ Los índices o posiciones son siempre números enteros positivos y consecutivos, que en *Java* empiezan en 0.
- ✓ Es necesario distinguir entre la posición o índice de un arreglo y el dato guardado o contenido en dicha posición.
- ✓ Es un tipo de datos personalizado por ser definido por usted mismo, dado que usted decide el tipo y cantidad de datos a contener en el arreglo.
- ✓ Es común que al concepto de arreglo también lo encontremos con el sobrenombre de vector.

- ✓ En *Java* todo arreglo es dinámico en cuanto al tamaño, es decir, que la cantidad de elementos que contiene puede cambiar en cualquier momento, cuando así lo necesitemos.

4. Arreglos en una clase

En este punto trataremos la relación de un arreglo con una clase, es decir veremos las formas distintas en las cuales una clase puede hacer uso de un arreglo. Tenga en cuenta que en todos los casos, la sintaxis y características señaladas con los arreglos no cambian en nada. Partiendo del hecho de que un arreglo es en general una variable (el concepto del cual precede), el arreglo puede figurar haciendo cualquiera de las funciones que cumple una variable normal. Por lo tanto, un arreglo puede ser un atributo de la clase, una variable local, un parámetro e incluso el valor retornado por un método; esto significa, que el arreglo hace parte de un atributo o de un método de la clase y con base ello estas pueden hacer uso del arreglo.

Así por ejemplo, el siguiente segmento de código *Java*, ilustra la declaración de un arreglo como variable local (*Vec*) dentro de un método:

```
public void Arreglo_Variable() {  
    int i;  
    float Vec[];  
    ...  
}
```

el siguiente código *Java* tenemos un arreglo (*args*) definido como parámetro, considerando el clásico método ***main*** para ello:

```
public static void main(String args[]) {  
  
  
}
```

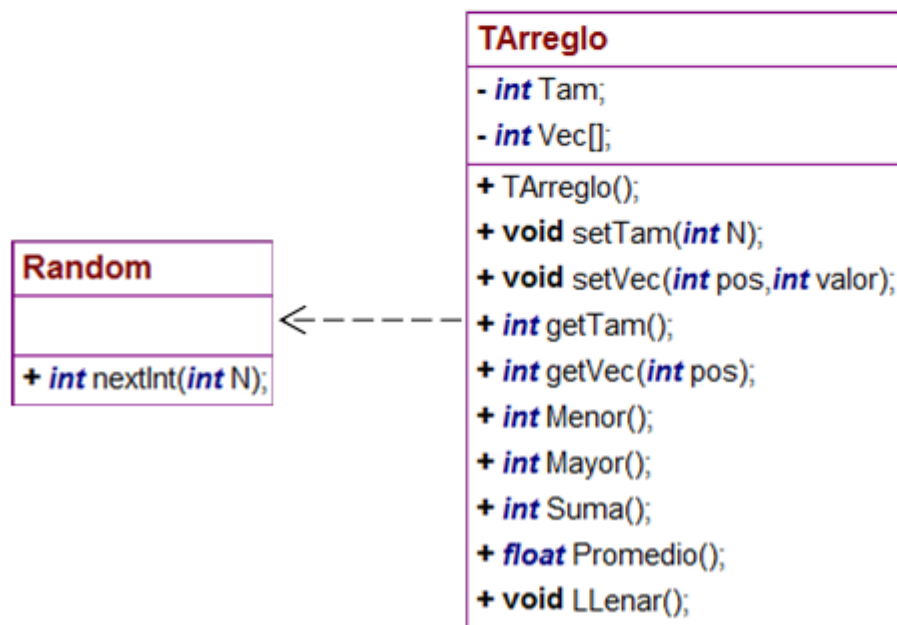
Para el caso de los arreglos como atributos, en el que haremos énfasis, es necesario que en virtud de la primera regla del encapsulado estos sean declarados en el área privada; con lo cual agregaremos dos métodos públicos para su acceso: un método modificador (***set***) y uno selector (***get***) como se sigue convencionalmente, terminando sus nombres con el nombre del arreglo en cuestión. Sin embargo tome muy en cuenta, que para ambos métodos, será necesario añadir un parámetro adicional (especialmente de tipo entero), que indica la posición

del elemento a acceder dentro del arreglo. En otras palabras, la interfaz de la clase que contiene a un arreglo de forma privada, lo expone a su exterior permitiendo el acceso a un solo elemento de aquel a la vez. Seguidamente se presenta el planteamiento y solución de un problema resuelto con arreglos donde se aterrizan los conceptos tratados hasta el momento.

5. Presentación ejemplo arreglos

Diseñe en UML una clase e impléméntela en Java, de modo que tenga como atributo un arreglo de N números enteros, de tal suerte, que determine el valor menor, el valor mayor, la suma y el promedio de sus elementos; además el arreglo debe llenarse con valores aleatorios comprendidos en el rango de 0 a 9999. Por otra parte en la ventana (**JFrame**), los datos del arreglo deben mostrarse en un **JTextArea** en líneas distintas: además el usuario deberá poder asignar el tamaño del arreglo y valores a cada posición.

6. Diagrama UML de clases

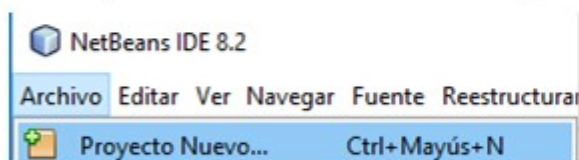


7. Creación del proyecto en NetBeans

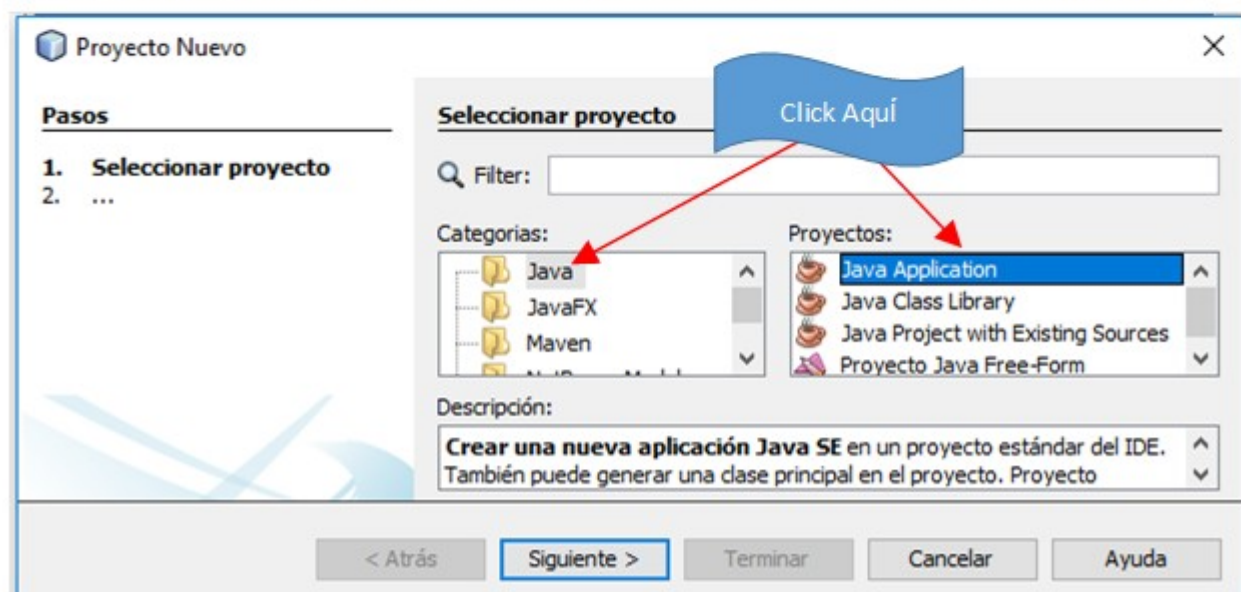
➡ Inicie o ejecute el programa *NetBeans* y observará una pantalla como la siguiente:



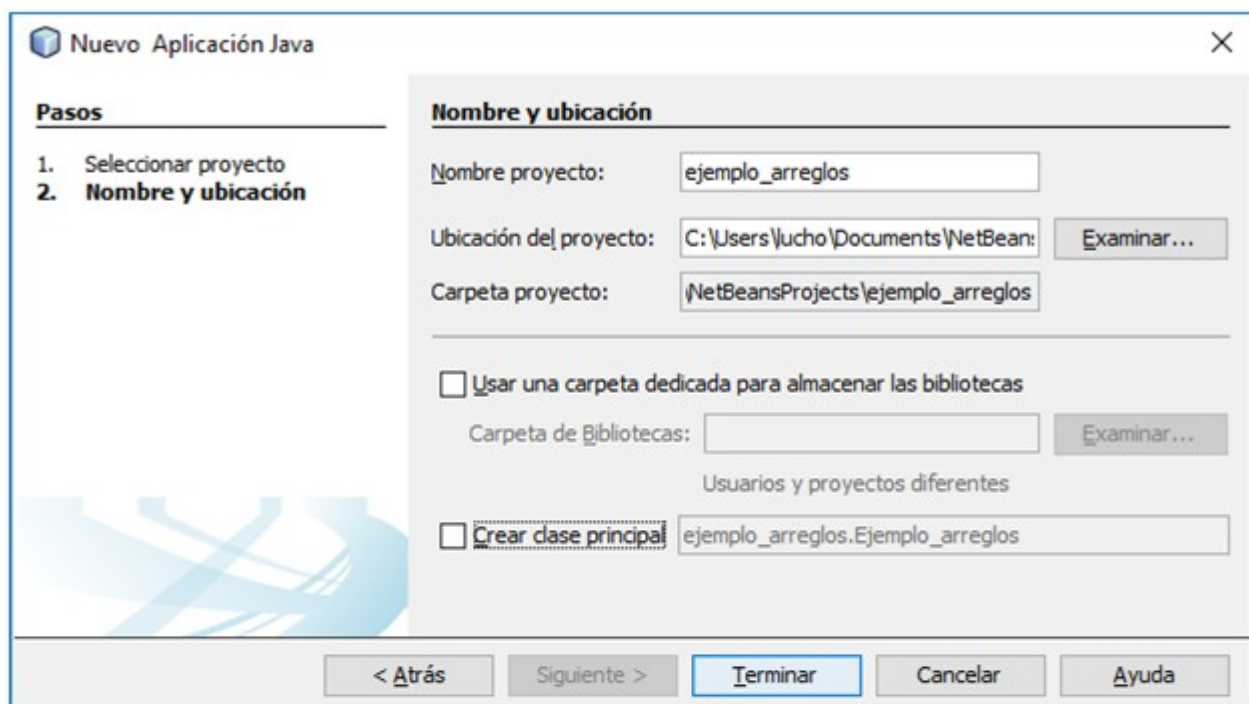
➡ Cree un nuevo proyecto haciendo click en el menú **Archivo** y luego en la opción **Proyecto nuevo**, tal como se muestra en la siguiente imagen:



➡ En la ventana desplegada, seleccione el nodo **java** del panel de **categorías** y el nodo **Java Application** del panel **proyectos**; a continuación haga click en el botón **siguiente**:



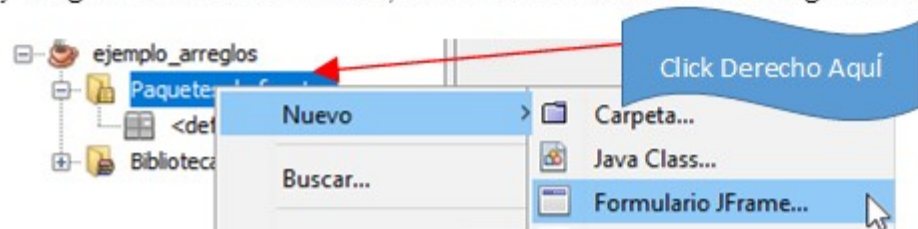
- ➡ En la siguiente ventana, ingrese el nombre del proyecto (**ejemplo_arreglos**), desmarque la casilla de verificación titulada como “**Crear clase principal**” y haga click en el botón **Terminar**.



- ➡ Una vez creado el proyecto en la parte superior ventana. Si el proyecto aparece contraído, vera a su derecha un signo +, haga click en este signo para expandirlo; luego haga click en el signo + de **Paquetes de fuentes** para expandirlo.



- ➡ Haga click derecho en nodo **Paquetes de fuentes** del proyecto, escoja la opción **Nuevo** (o y luego **Formulario JFrame**, tal como se muestra en la siguiente imagen:



- ➡ En la ventana desplegada, en la entrada **Class Name**, ingrese el nombre para la clase (**Ventana**) y haga click en el botón **Terminar**.

Nuevo Formulario JFrame

Pasos

1. Escoja el tipo de archivo
2. **Name and Location**

Name and Location

Class Name:

Project:

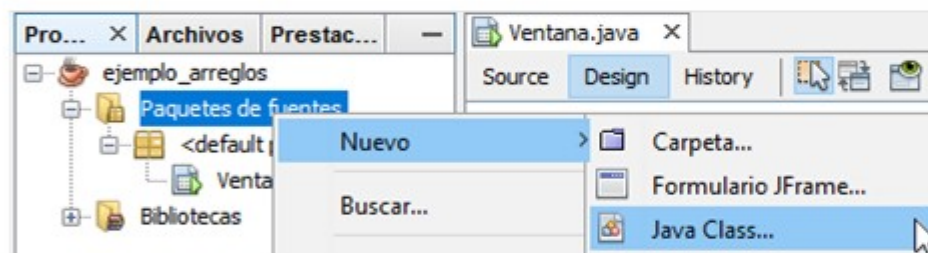
Location:

Package:

Created File:

< Atrás Siguiente > **Terminar** Cancelar Ayuda

- ➡ Para crear la clase **TArreglo**, haga click derecho en nodo **Paquetes de fuentes** del proyecto y escoja la opción **Nuevo** y luego **Java Class**, tal como se muestra abajo:



- ➡ En la entrada **Class Name** presentada, ingrese **TArreglo** y después haga click en el botón **Terminar**.

Nuevo Java Class

Pasos

1. Escoja el tipo de archivo
2. **Name and Location**

Name and Location

Class Name:

Project:

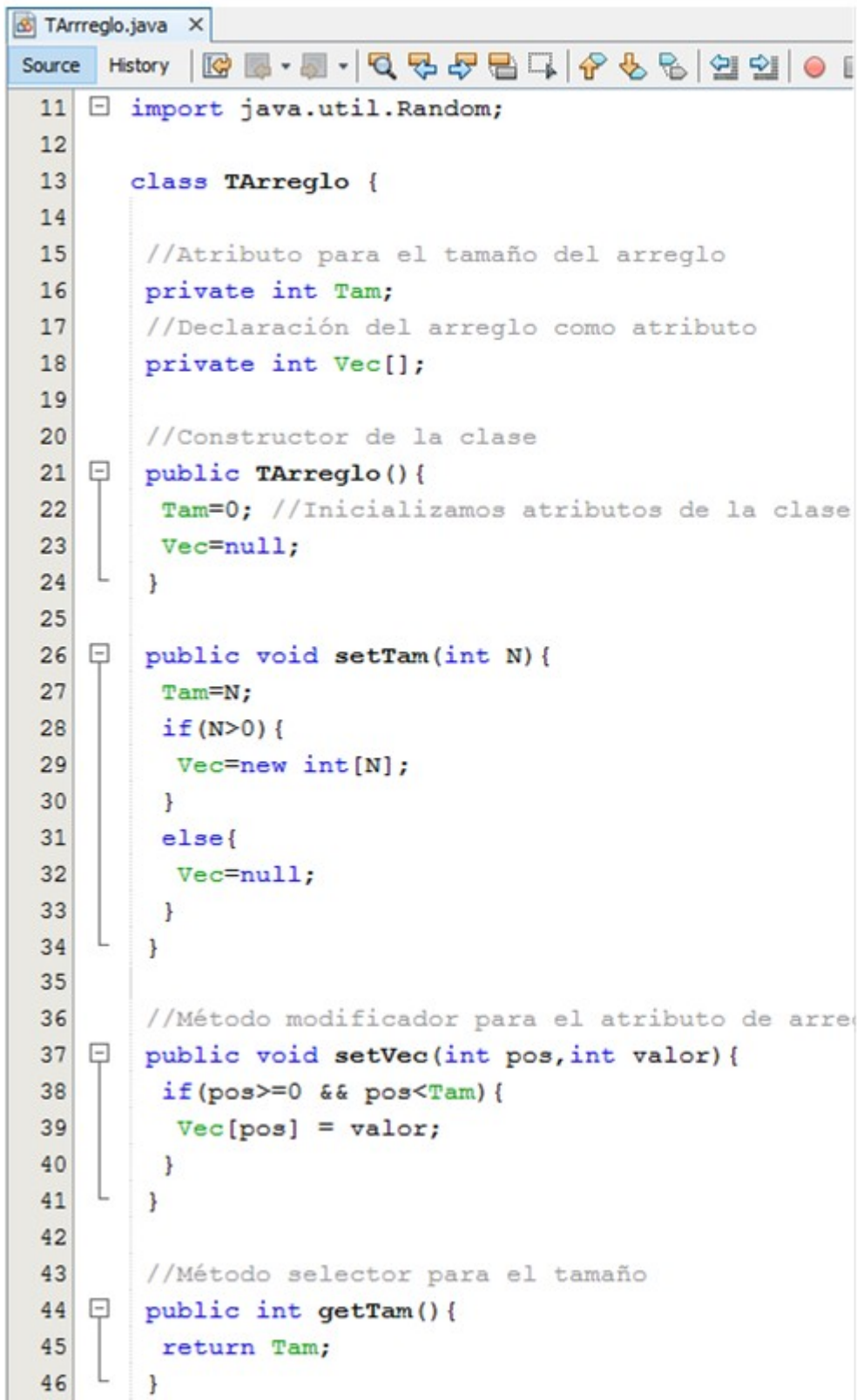
Location:

Package:

Created File:

< Atrás Siguiente > **Terminar** Cancelar Ayuda

- ➡ Implemente la clase **TArreglo** de la siguiente manera, considerando que los comentarios (texto gris precedido por la doble barra //) no es necesario escribirlo:



```
11 import java.util.Random;
12
13 class TArreglo {
14
15     //Atributo para el tamaño del arreglo
16     private int Tam;
17     //Declaración del arreglo como atributo
18     private int Vec[];
19
20     //Constructor de la clase
21     public TArreglo() {
22         Tam=0; //Inicializamos atributos de la clase
23         Vec=null;
24     }
25
26     public void setTam(int N) {
27         Tam=N;
28         if(N>0) {
29             Vec=new int[N];
30         }
31         else{
32             Vec=null;
33         }
34     }
35
36     //Método modificador para el atributo de arreglo
37     public void setVec(int pos,int valor) {
38         if(pos>=0 && pos<Tam) {
39             Vec[pos] = valor;
40         }
41     }
42
43     //Método selector para el tamaño
44     public int getTam() {
45         return Tam;
46     }
```

```
TArreglo.java x
Source History
47
48 //Método selector para el atributo de arreglo
49 public int getVec(int pos){
50     return Vec[pos];
51 }
52
53 //Métodos para resolver el problema...
54 public int Menor(){
55     int i,Min;
56     //Partimos que el menor podría ser el primer elemento (de posición 0)
57     Min = Vec[0];
58     for(i=1;i<Tam;i++){
59         //Ahora buscamos en el resto de elementos (del 1 al 49) otro menor que min
60         if(Vec[i]<Min){//Si encontramos uno menor que min
61             //Actualizamos a min con aquel valor encontrado que es menor que min
62             Min = Vec[i];
63         }
64     }
65     return Min;
66 }
67
68 public int Mayor(){
69     int i,Max;
70     //Partimos que el mayor podría ser el primer elemento (de posición 0)
71     Max = Vec[0];
72     for(i=1;i<Tam;i++){
73         //Ahora buscamos en el resto de elementos (del 1 al 49) otro mayor que max
74         if(Vec[i]>Max){//Si encontramos uno mayor que max
75             //Actualizamos a max con aquel valor encontrado que es mayor que max
76             Max = Vec[i];
77         }
78     }
79     return Max;
80 }
81
```

```

eglo.java X
History
}

public int Suma() {
    int i, sum = 0; //Inicializamos la suma en cero
    for(i=0; i<Tam; i++){ //Recorremos todo el arreglo
        //Incrementamos el valor de sum con el valor de cada elemento del arreglo
        sum = sum + Vec[i];
    }
    return sum;
}

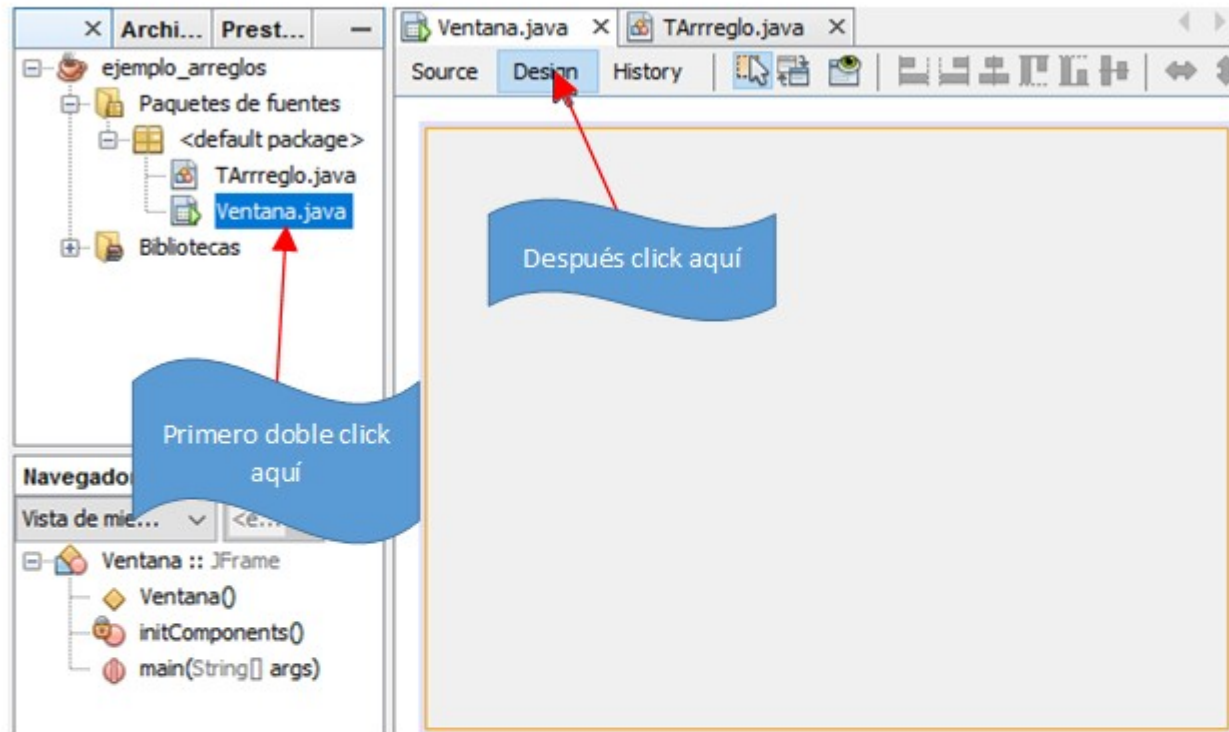
public float Promedio() {
    //Convertimos explícitamente con un cast (float) el resultado de suma
    //de entero a real, para que la parte decimal si la hay no sea truncada
    return (float) Suma() / Tam;
}

//Método para llenar el arreglo números aleatorios entre 0 y 999
public void LLenar() {
    int i;
    //La clase Random permite generar números aleatorios, requiere incluir
    //desde el paquete java.util
    Random R = new Random();
    for(i=0; i<Tam; i++){
        //El método nextInt genera un número al azar entre 0 y 10000-1 (0 a 9999)
        Vec[i] = R.nextInt(10000);
    }
}
}

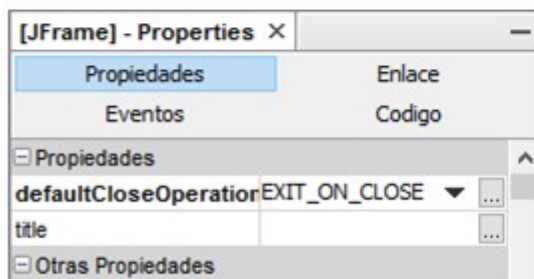
```


8. Diseño gráfico de la ventana

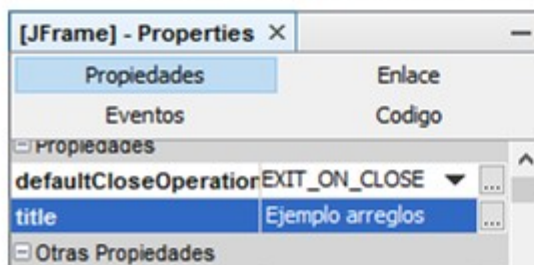
- ➡ Si la ventana (**JFrame**) no está visible, actívelo en modo diseño: para ello primero haga doble click en el archivo *Ventana.java* y después click en la vista diseño (**Design**). Así la vista de diseño de la ventana, es el rectángulo gris de borde naranja que vemos en la siguiente imagen:



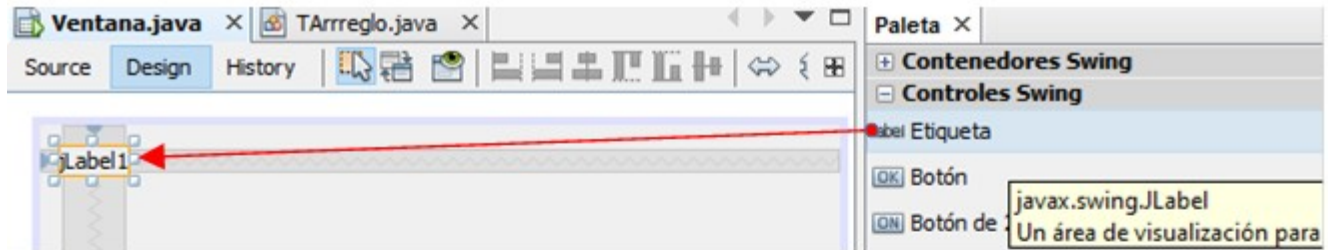
- ➡ Haga click dentro de la ventana y luego vaya al inspector de propiedades; ubique la propiedad **title** y haga click en el recuadro de su derecha:



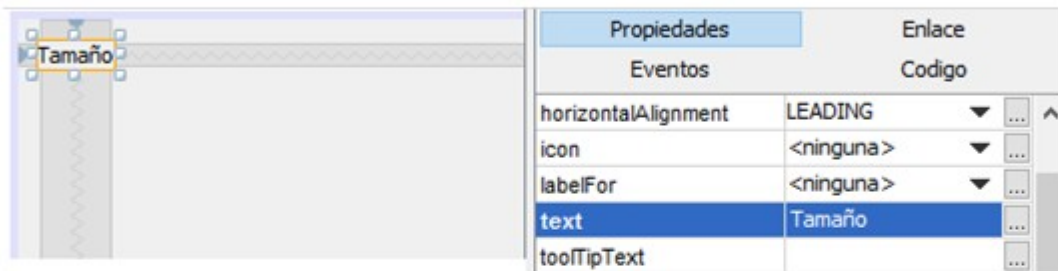
- ➡ Escriba **Ejemplo arreglos** como el título para la ventana y pulse enter.



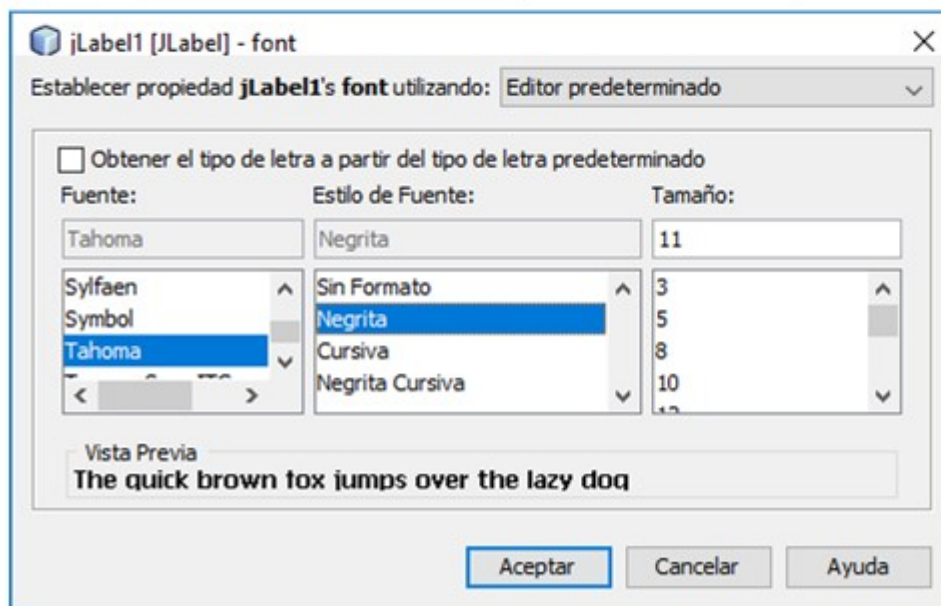
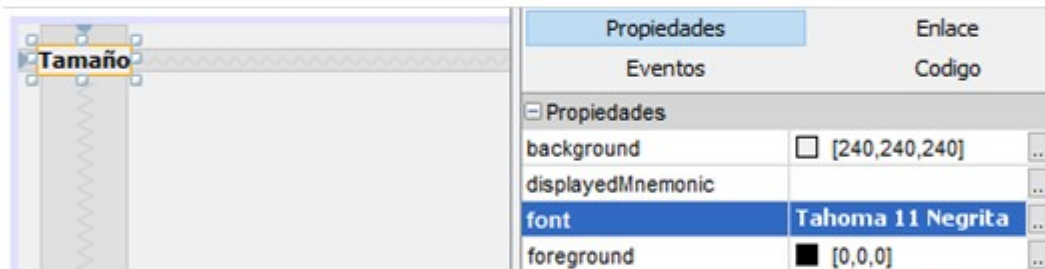
➡ Arrastre la etiqueta sobre la ventana hasta el punto indicado:



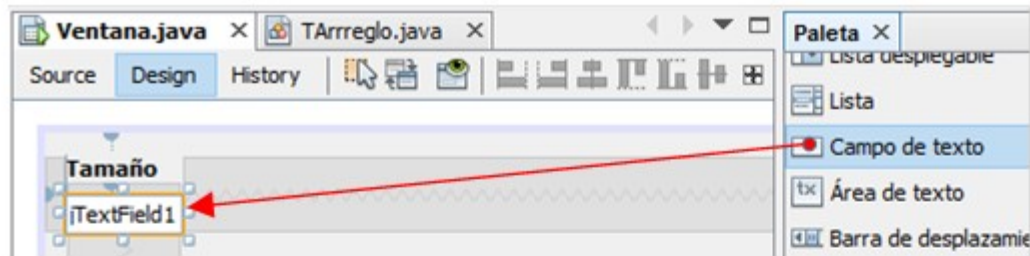
➡ Localice la propiedad **text**, haga click a su derecha borre su texto, escriba **Tamaño** y pulse tecla enter; quedando así:



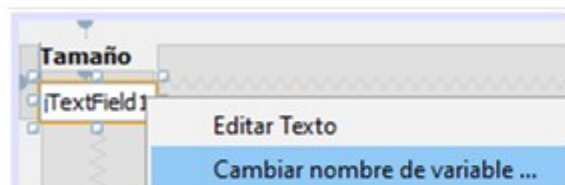
➡ Haga click en el botón de tres puntos (...), que se encuentra a la derecha de la propiedad **font**, seleccione **Negrita** y haga click en el botón **Aceptar**.



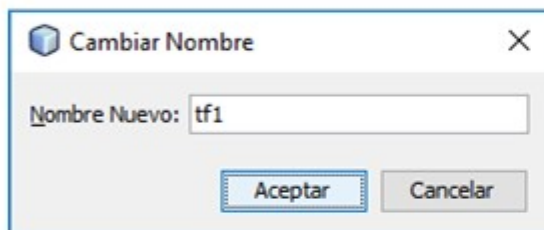
- ➡ Arrastre un *campo de texto* (**JTextField**) hasta la ventana en la posición indicada:



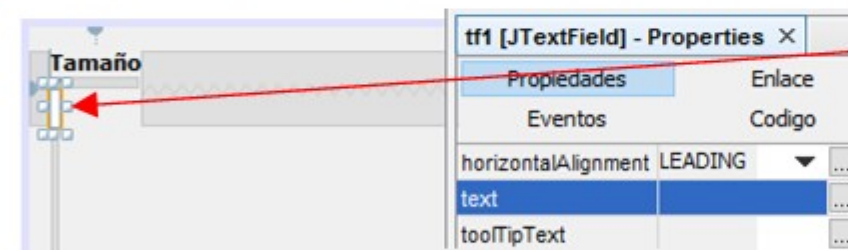
- ➡ Haga click derecho sobre el campo de texto y escoja la opción **Cambiar nombre de variable...**



- ➡ En esta ventana ingrese **tf1** (nombre de instancia) y haga click en el botón **Aceptar**



- ➡ Busque la propiedad **text**, borre su contenido y pulse enter:



Use este nodo para redimensionar

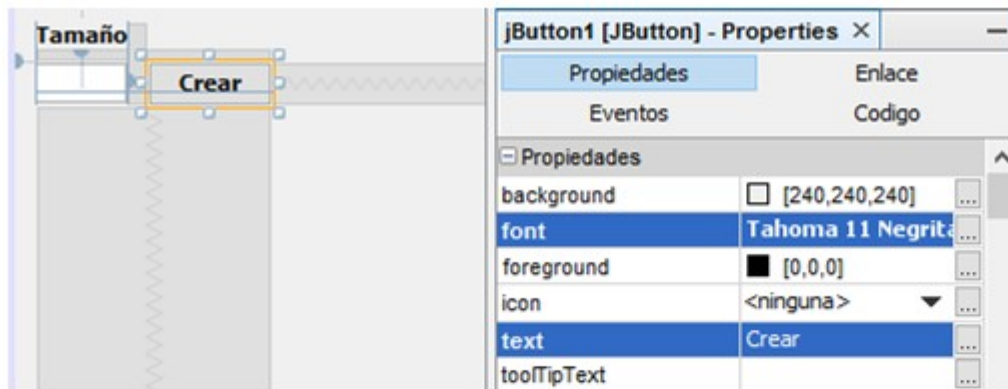
- ➡ Redimensione el campo de texto hasta el ancho indicado, seleccionándolo por el nodo central derecho y arrastrando hacia la derecha con el botón izquierdo pulsado.



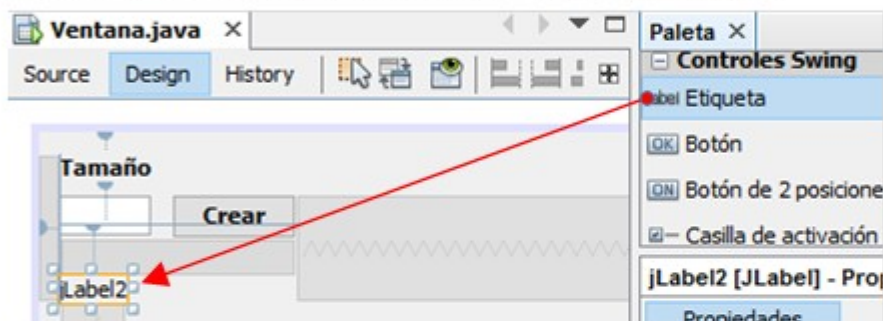
- ➡ Arrastre un botón (**JButton**) hasta este punto:



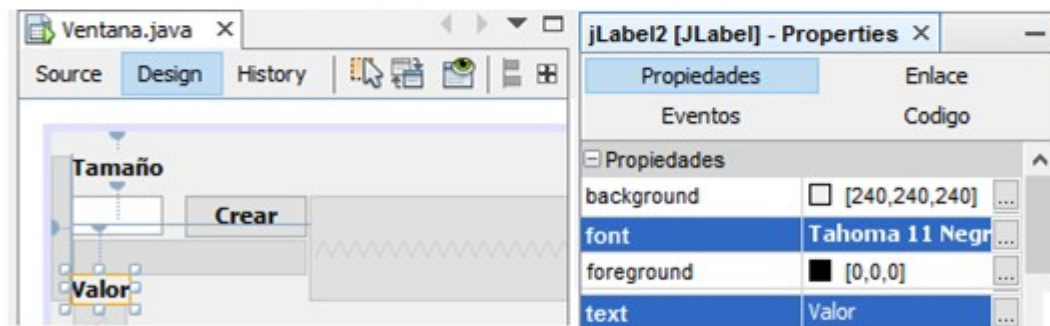
- ➡ En el inspector de propiedades busque la propiedad **text**, ponga en ella el texto **Crear** y pulse enter. Seguidamente busque la propiedad **font** y ponga el texto en negrilla.



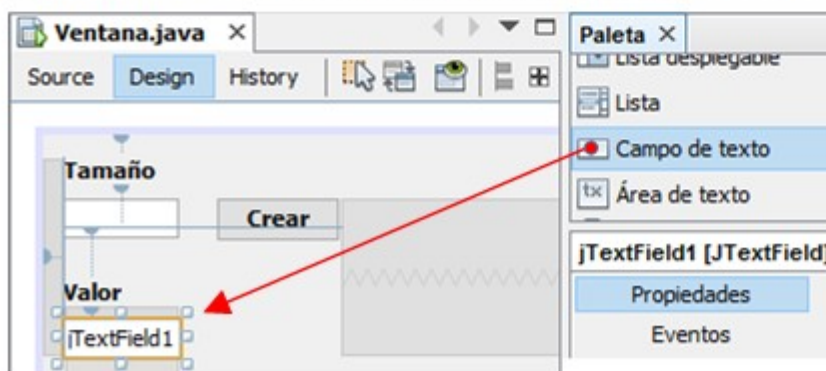
- ➡ Arrastre una etiqueta (**JLabel**) hasta el punto indicado:



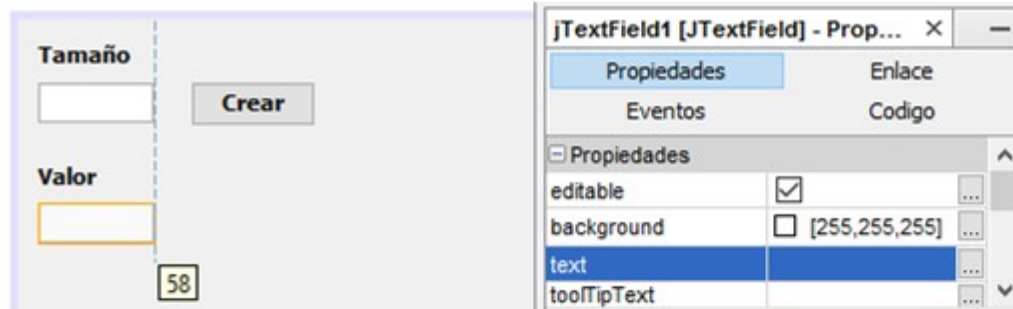
- ➡ La propiedad **font** póngala en negrilla y escriba **valor** en la propiedad **text**.



- ➡ Arrastre un campo de texto (**JTextField**) hasta la ventana en la posición señalada:



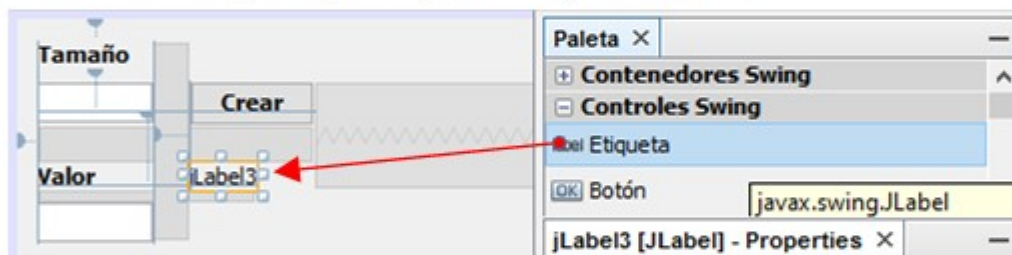
- ➔ Busque la propiedad **text**, borre su contenido y ajuste el ancho del control a un valor adecuado:



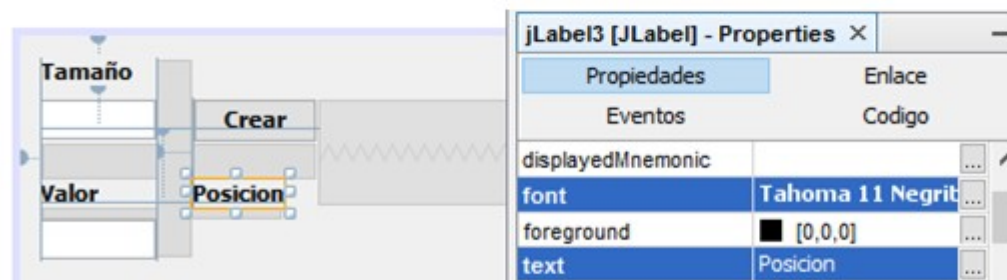
- ➔ Haga click derecho sobre el campo de texto, escoja la opción **Cambiar nombre de variable...** ingrese **tf2** como nombre de instancia y haga click en el botón **Aceptar**



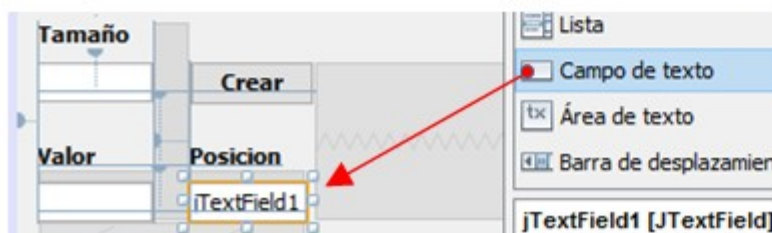
- ➔ Arrastre otra etiqueta (**JLabel**) hasta el punto indicado:



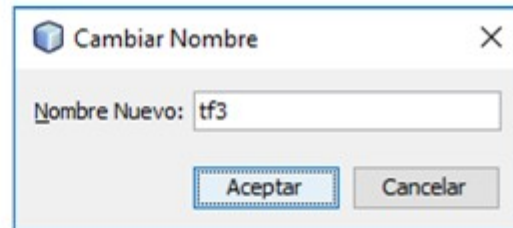
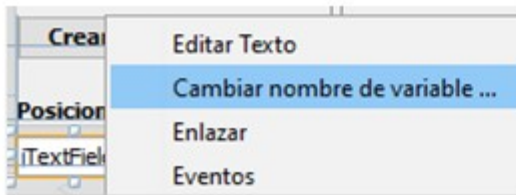
- ➔ En la propiedad **text** escriba **posición** y en la propiedad **font** seleccione en negrilla



- ➔ Arrastre un campo de texto (**JTextField**) hasta la ventana en la posición mostrada:



- ➡ Haga click derecho sobre el campo de texto, escoja la opción **Cambiar nombre de variable...** ingrese **tf3** como nombre de instancia y haga click en el botón **Aceptar**



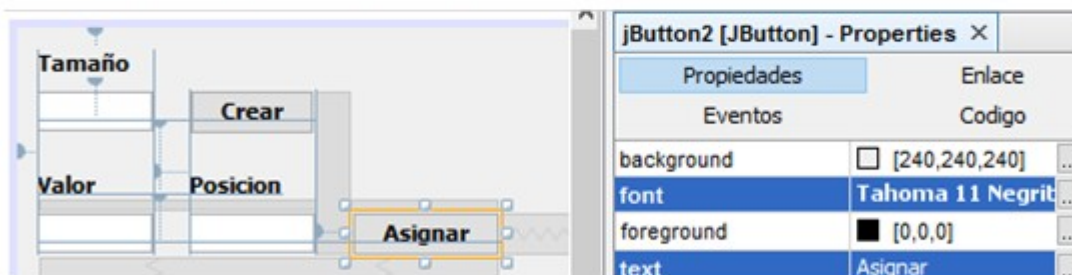
- ➡ Busque la propiedad **text**, borre su contenido y ajuste el ancho a un valor apropiado:



- ➡ Arrastre un **JButton** hasta el punto indicado:



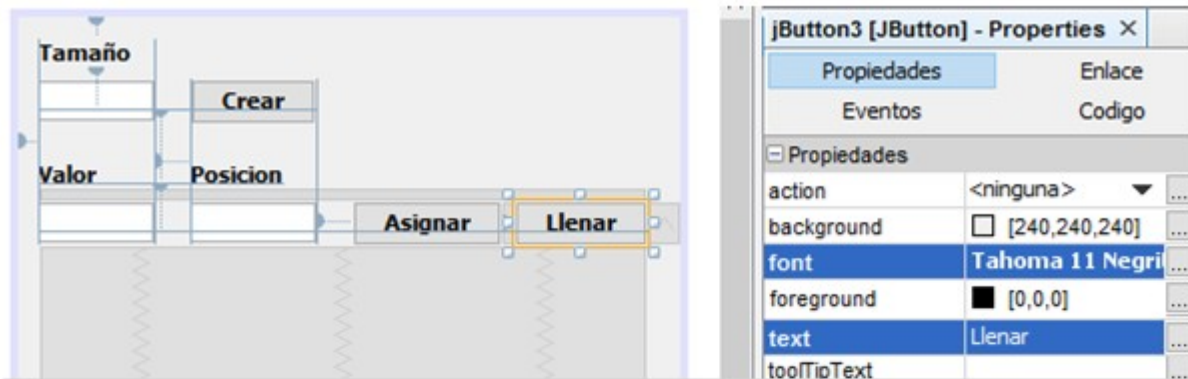
- ➡ En el inspector de propiedades, ponga **Asignar** en la propiedad **text**, pulse enter y después ponga negrilla en la propiedad **font**.



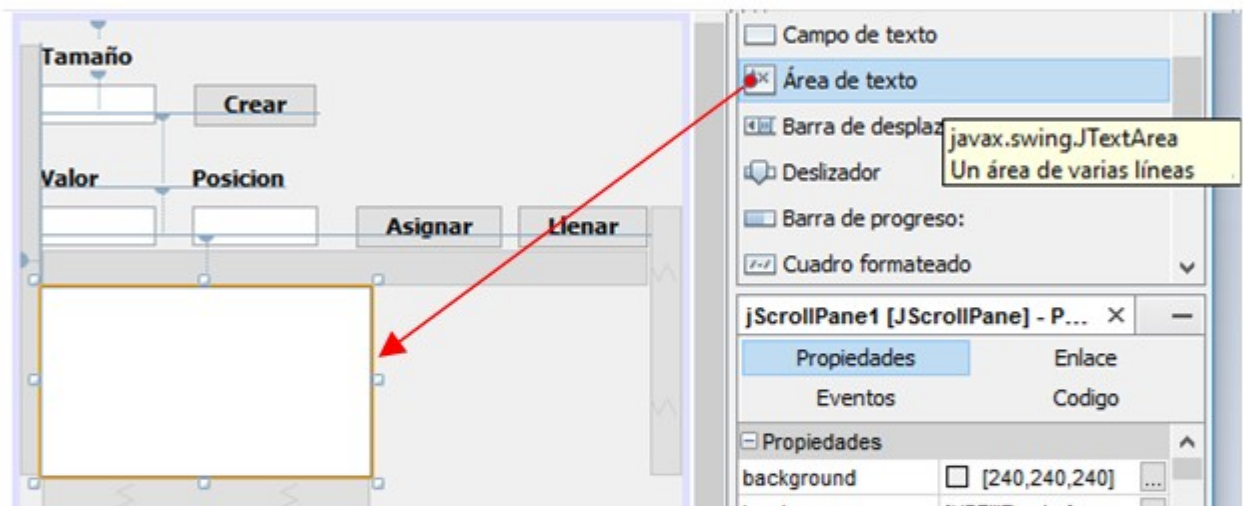
- ➡ Arrastre un **JButton** hasta el puesto señalado:



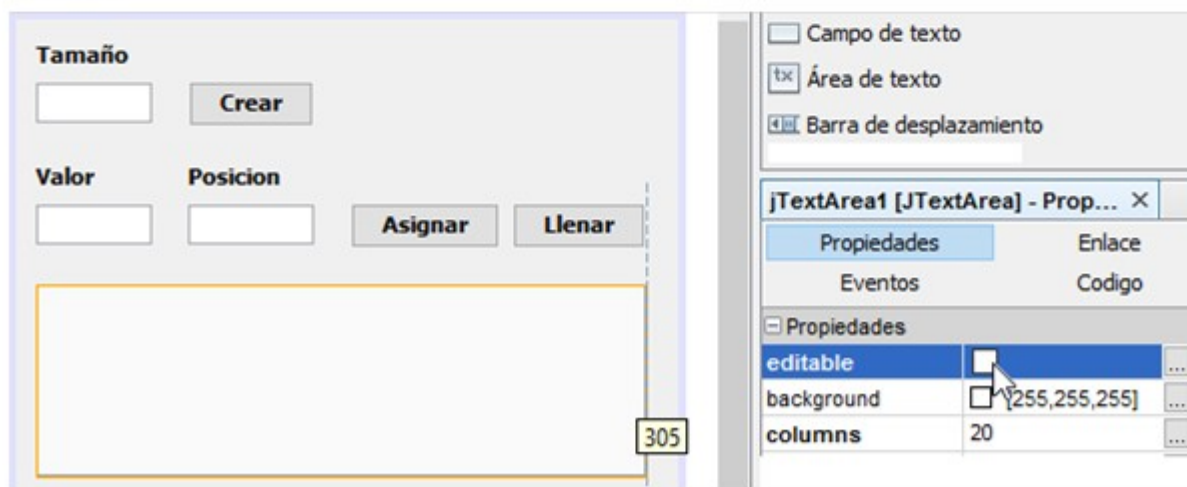
- ➡ En el inspector de propiedades, ponga **Llenar** en la propiedad **text**, pulse enter y después ponga negrilla en la propiedad **font**.



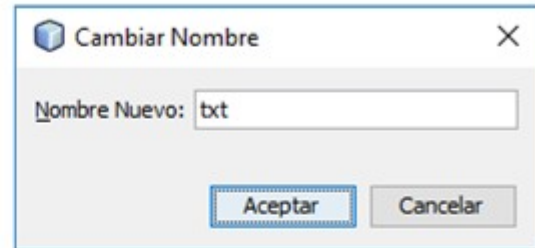
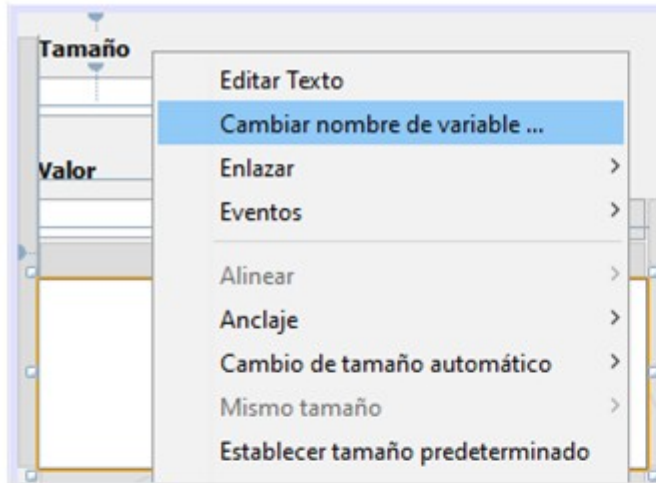
- ➡ Arrastre un **JTextArea** (Area de texto) hasta el punto señalado:



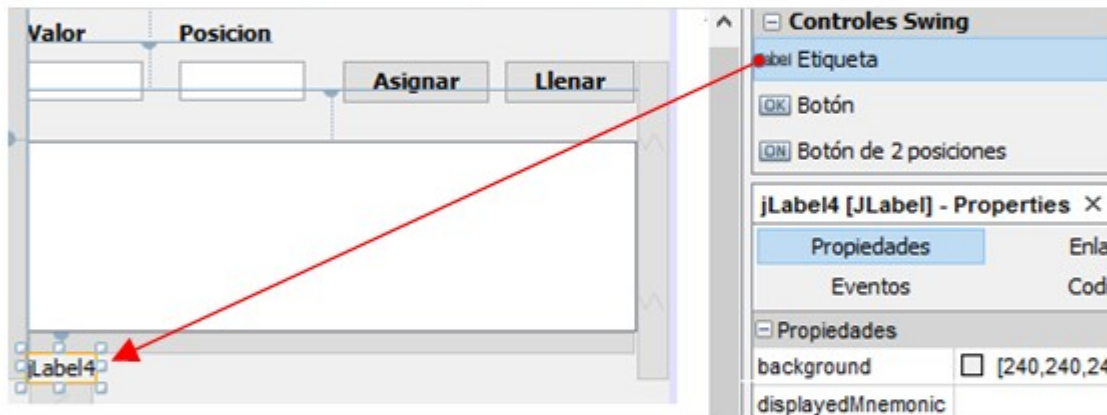
- ➡ Haga click dentro del área de texto, rediménsionela hasta el ancho indicado y en el inspector de propiedades, desmarque la casilla de la propiedad editable (para que sea de solo lectura).



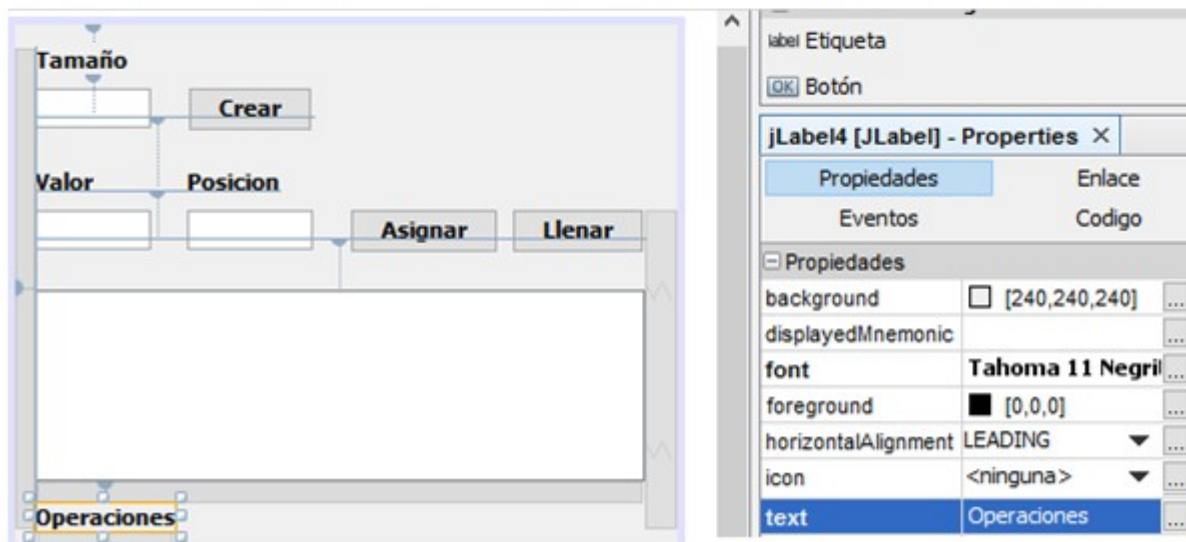
- ➡ Cambie el nombre al control, haciendo click derecho en él y seleccionando la opción **Cambiar nombre de variable...** ingrese **txt** y haga click en el botón **Aceptar**



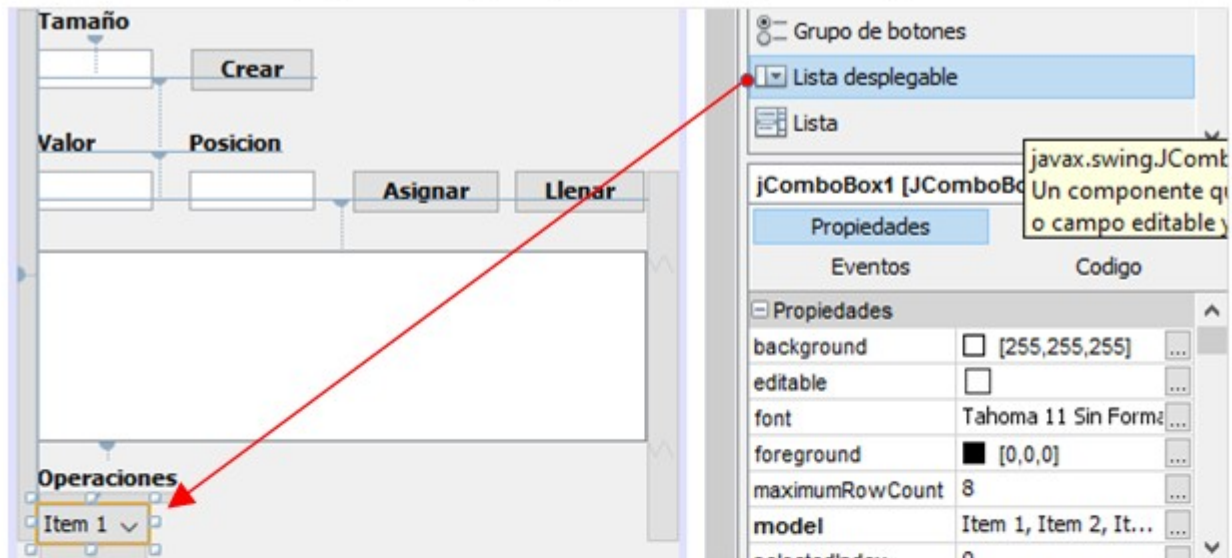
- ➡ Arrastre una etiqueta (**JLabel**) hasta el punto indicado:



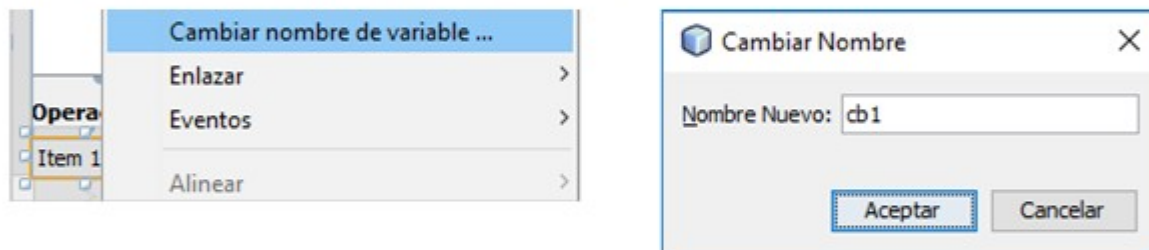
- ➡ En el inspector ponga **Operaciones** en la propiedad **text** y una **font** en negrilla



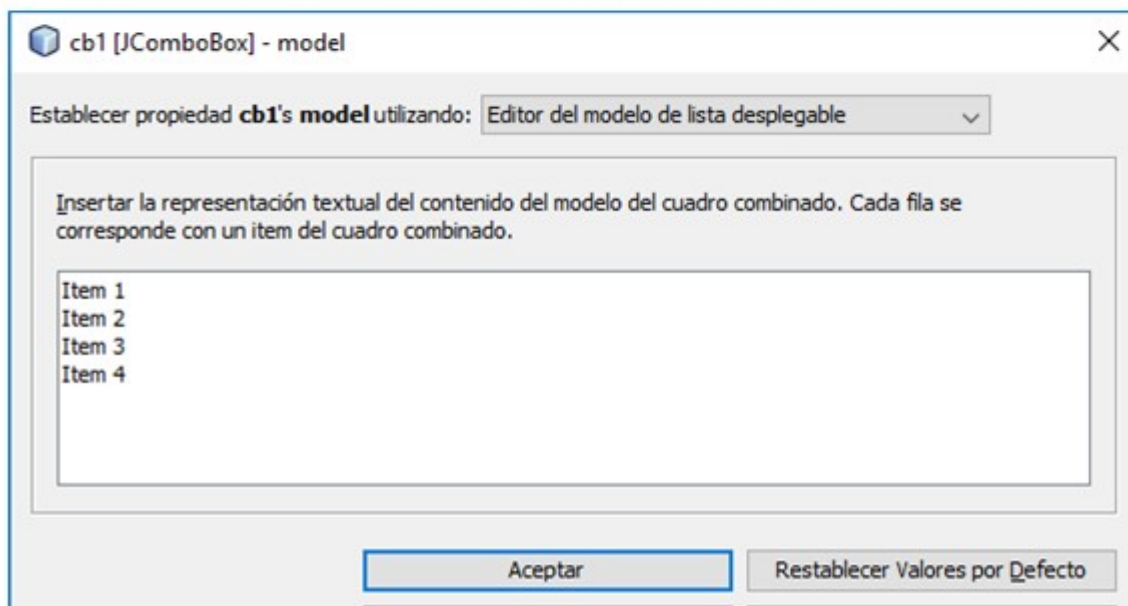
- ➡ Arrastre un *lista desplegable* etiqueta (**JComboBox**) hasta la posición señalada:



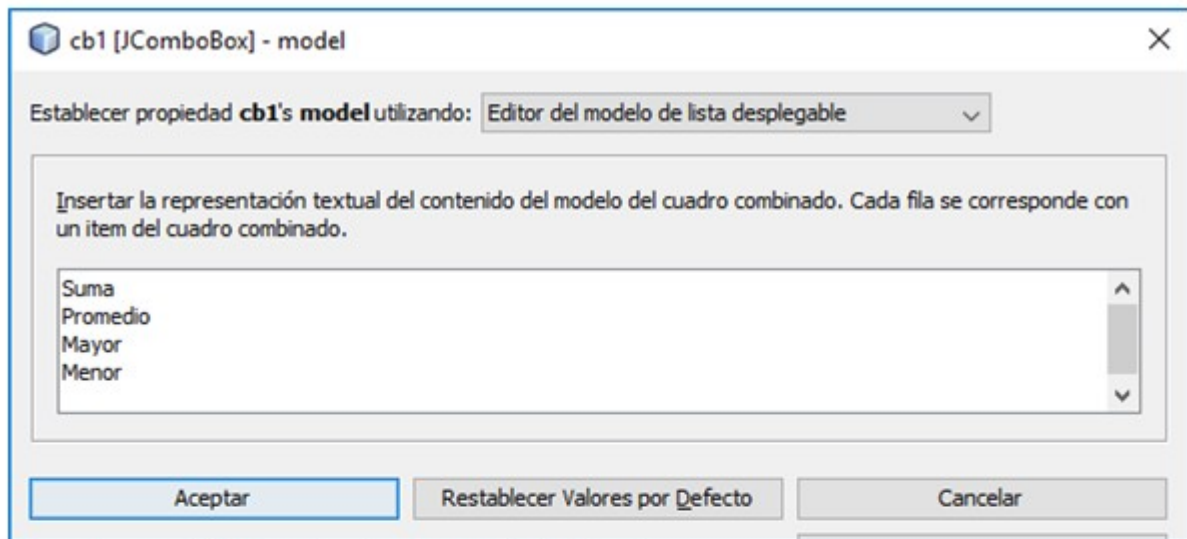
- ➡ Haga click derecho sobre el control, seleccione la opción **Cambiar nombre de variable...** ingrese **cb1** y haga click en el botón **Aceptar**



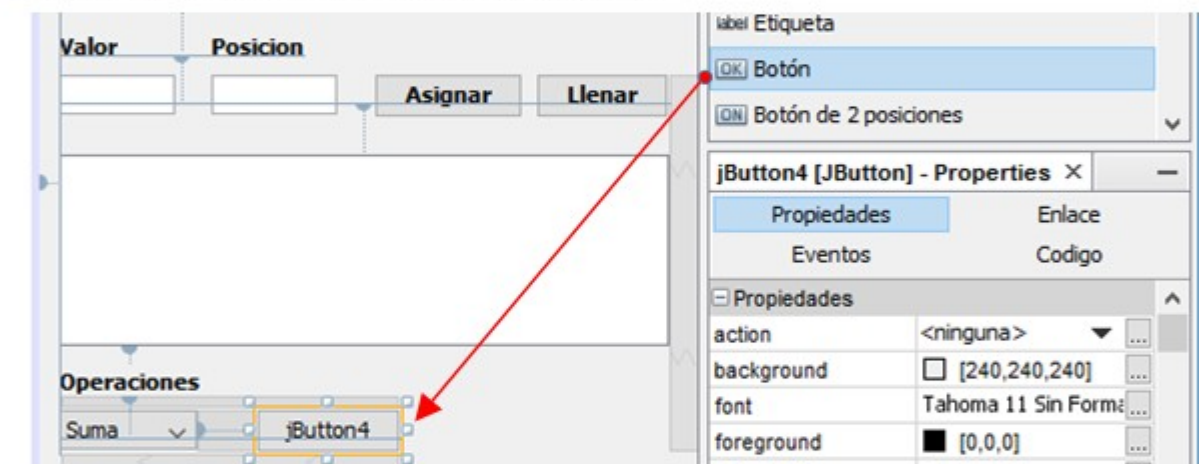
- ➡ Haga click en el botón de tres puntos (...) que está a la derecha de la propiedad **model**,
(**model** | Item 1, Item 2, It... (...)) con lo que vera la siguiente ventana:



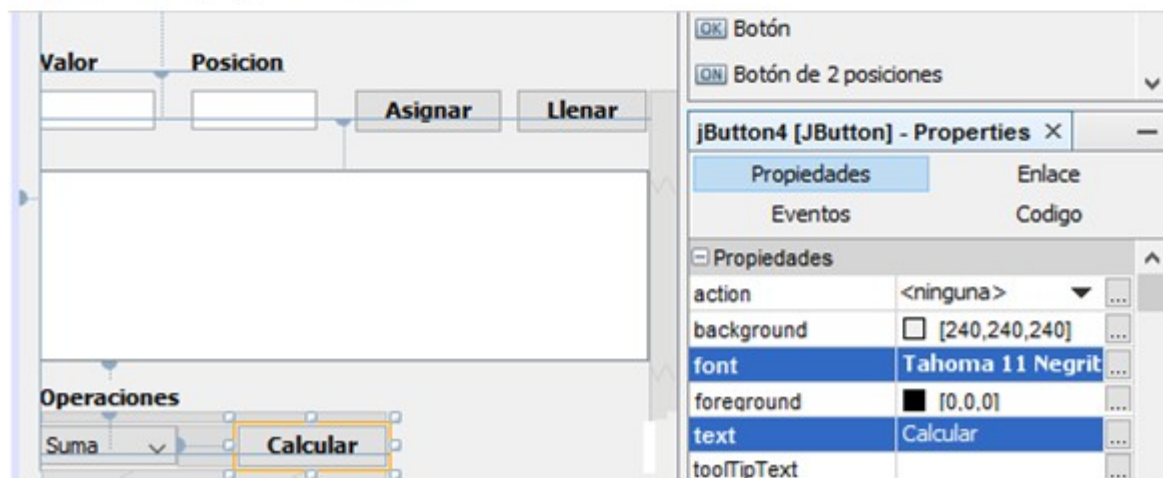
- ➡ Borre el contenido (*Item1*, *Item2*, *Item3* e *Item4*) y en vez de ello escriba las siguientes opciones una debajo de la otra: **Mayor**, **Menor**, **Suma** y **Promedio**



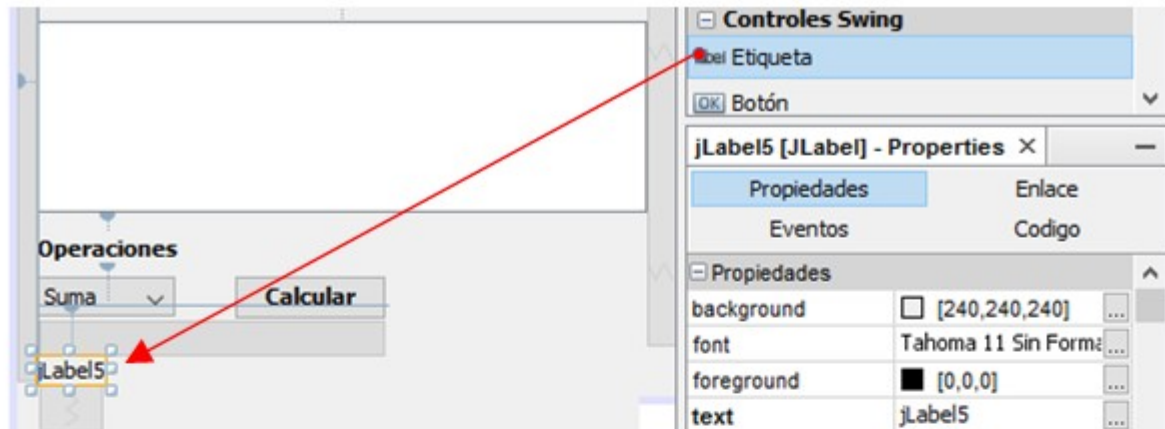
- ➡ Arrastre un **JButton** hasta el puesto indicado:



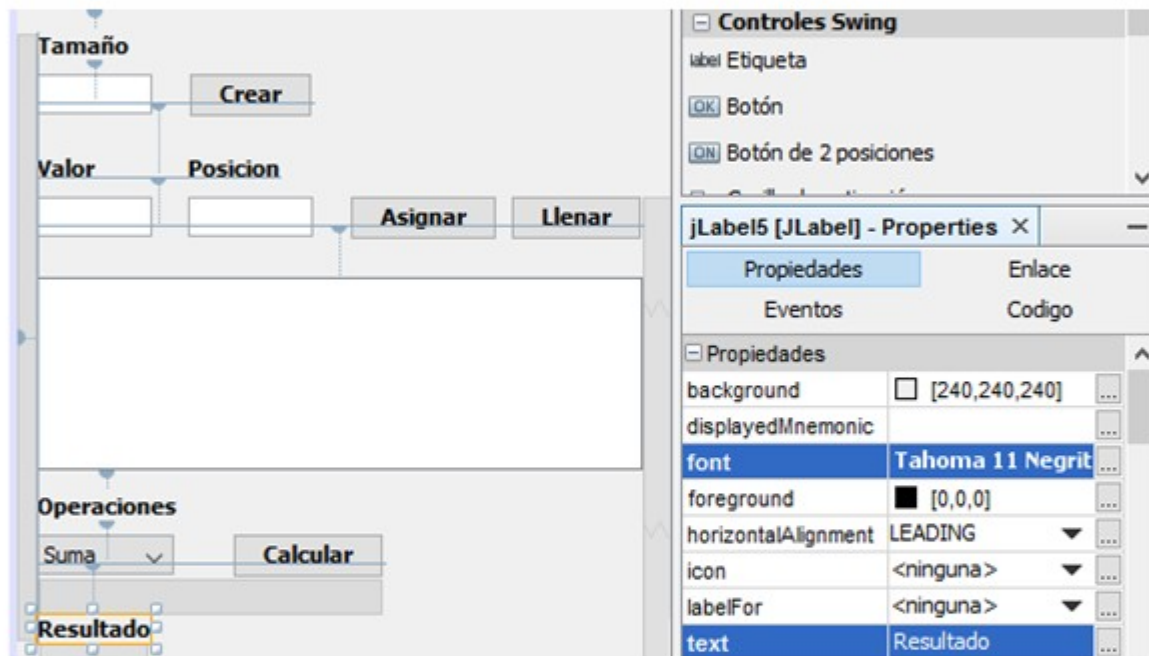
- ➡ En el inspector escriba **Calcular** en la propiedad **text**, pulse enter y después ponga negrilla en la propiedad **font**.



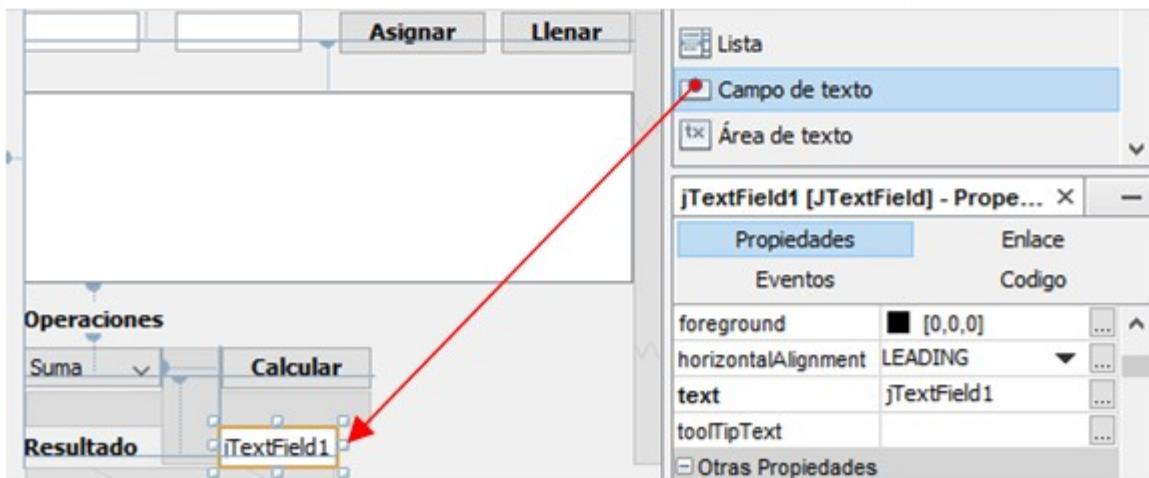
- ➡ Arrastre una etiqueta (**JLabel**) hasta la posición señalada:



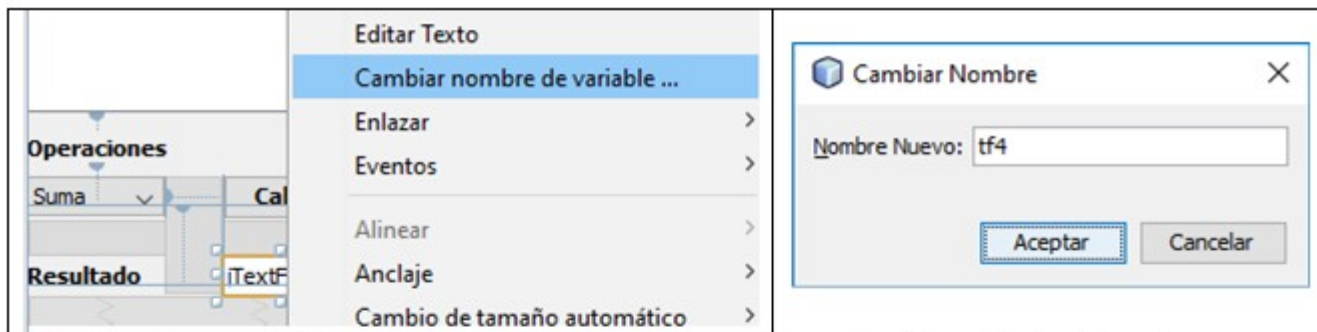
- ➡ En el inspector ponga **Resultado** en la propiedad **text** y una **font** en negrilla



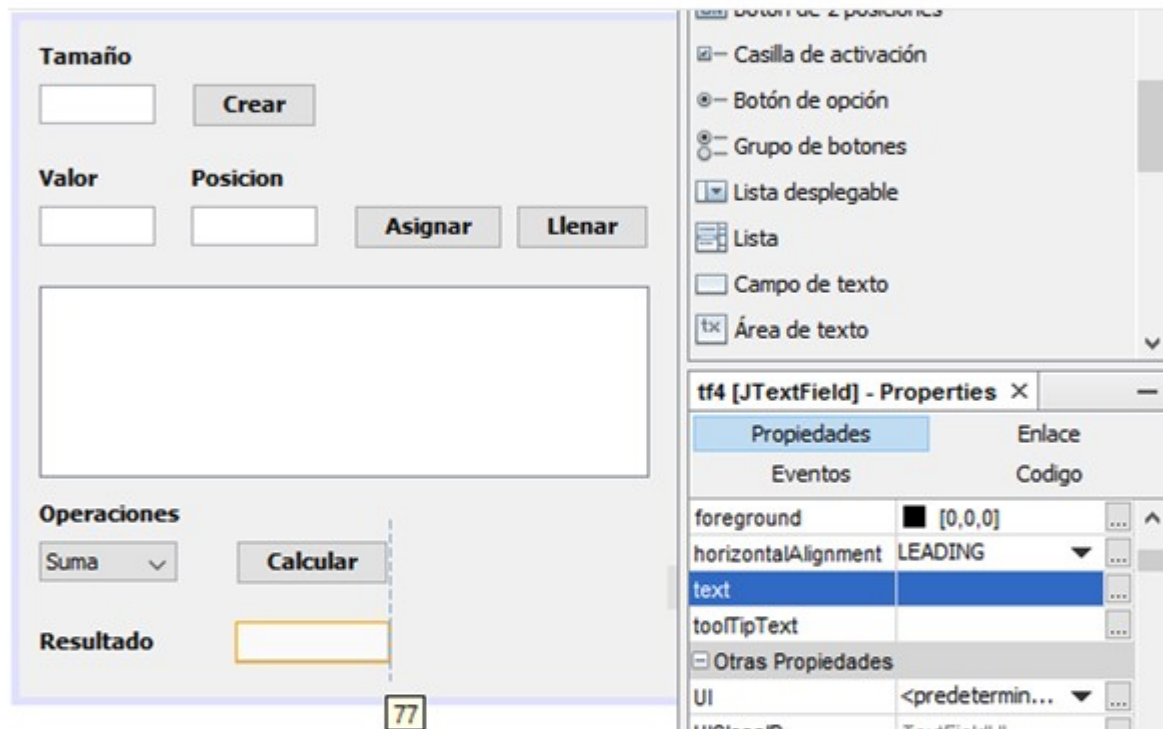
- ➡ Arrastre un campo de texto (**JTextField**) hasta la ventana en la posición mostrada:



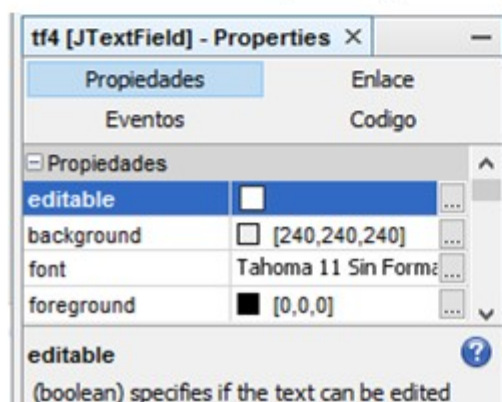
- ➡ Haga click derecho sobre el campo de texto, escoja la opción **Cambiar nombre de variable...** ingrese **tf4** como nombre y haga click en el botón **Aceptar**



- ➡ En el inspector busque la propiedad **text**, borre su contenido y ajuste el ancho:



- ➡ Finalmente desmarque la casilla de la propiedad editable, para que el control sea de solo lectura en tiempo de ejecución.



9. Implementación de la ventana

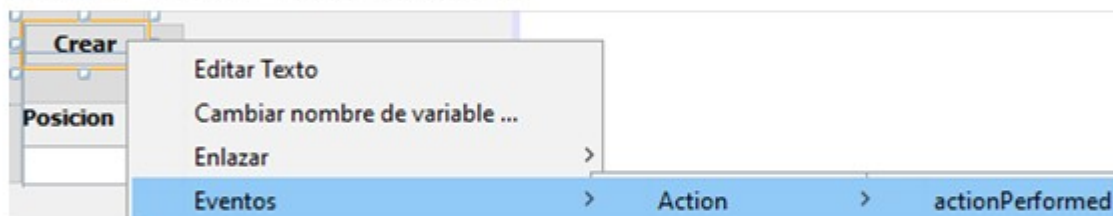
- ➡ Vaya al código fuente de la ventana: debajo de la línea de declaración de la clase, añada la instancia **Ar** de la clase **TArreglo** como atributo de la clase de la ventana (código dentro del cuadro rojo):

```
public class Ventana extends javax.swing.JFrame {  
    private TArreglo Ar=new TArreglo();  
}
```

- ➡ Más debajo de esta línea añada la implementación del método *Mostrar*:

```
private void Mostrar() {  
    int i;  
    txt.setText("");  
    for(i=0;i<Ar.getTam();i++) {  
        txt.append(String.format("Vec[%d]=%d\n",i,Ar.getVec(i)));  
    }  
}
```

- ➡ Vaya al diseño de la ventana y haga click derecho sobre el botón **Crear**, seleccione las opciones **Eventos + Action + actionPerformed**:



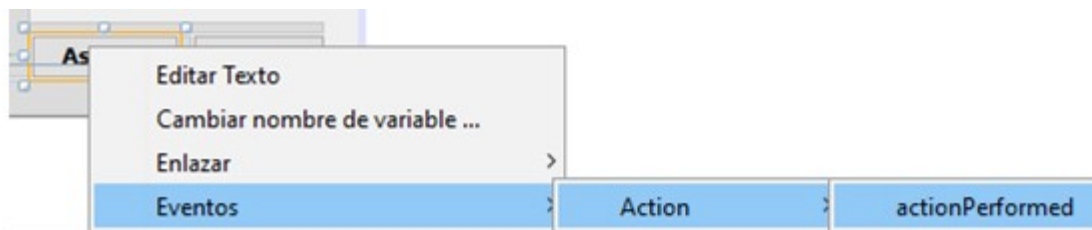
- ➡ El IDE añade un método privado tipo **void** a la ventana, cuyo nombre está compuesto por el nombre del botón (**jButton1**) seguido por el tipo de acción del evento (**ActionPerformed**)

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    // TODO add your handling code here:  
}
```

- ➡ La implementación la hacemos entre las dos llaves { }, es decir, borrando el comentario que dice: **//TODO add your handling code here:** y escribiendo nuestro propio código, que para el caso es el siguiente:

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    Ar.setTam(Integer.parseInt(tf1.getText()));  
    txt.setText("");  
}
```

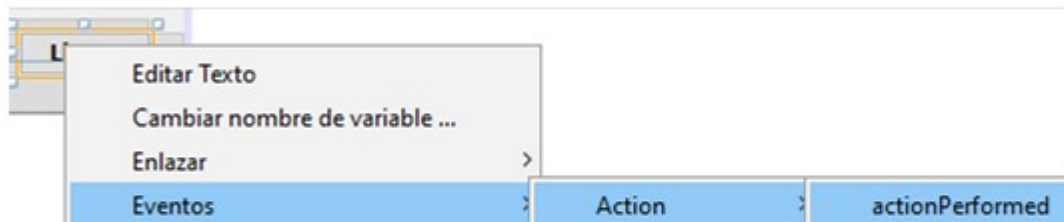
- ➡ Vuelva nuevamente a la vista diseño, haga click derecho sobre el botón **Asignar** y seleccione las opciones **Eventos + Action + actionPerformed**:



➡ El código para este botón es el siguiente:

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    int val=Integer.parseInt(tf2.getText());
    int pos=Integer.parseInt(tf3.getText());
    Ar.setVec(pos, val);
    Mostrar();
}
```

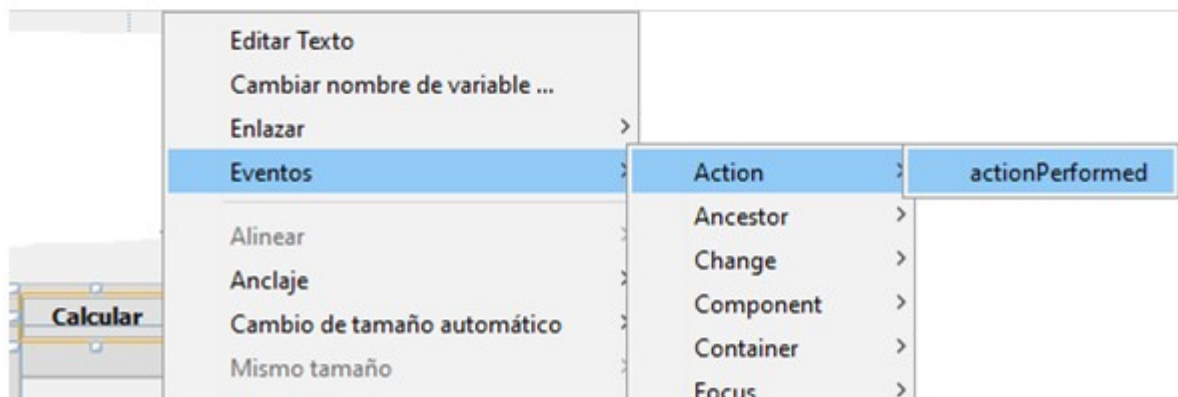
➡ Vaya a la vista diseño, haga click derecho sobre el botón **Llenar** y seleccione las opciones **Eventos + Action + actionPerformed** :



➡ Asegúrese que el código para este botón sea como sigue:

```
private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
    Ar.LLenar();
    Mostrar();
}
```

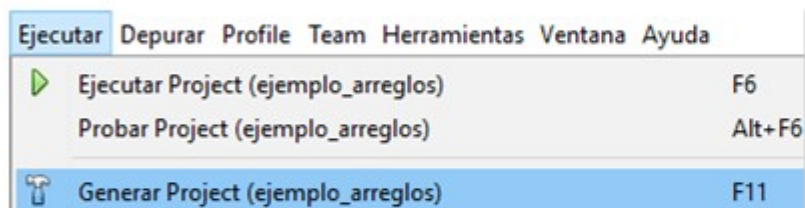
➡ Vaya a la vista diseño, haga click derecho sobre el botón **Calcular** y seleccione las opciones **Eventos + Action + actionPerformed** :



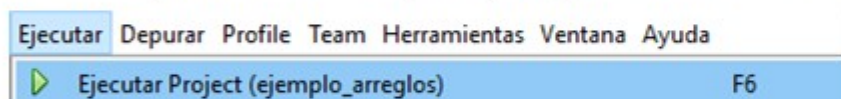
- ➡ El código para el evento de este botón es como sigue:

```
private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {  
    String res="";  
    switch(cb1.getSelectedIndex()) {  
        case 0:res=Ar.Suma()+"";break;  
        case 1:res=Ar.Promedio()+"";break;  
        case 2:res=Ar.Mayor()+"";break;  
        case 3:res=Ar.Menor()+"";break;  
    }  
    tf4.setText(res);  
}
```

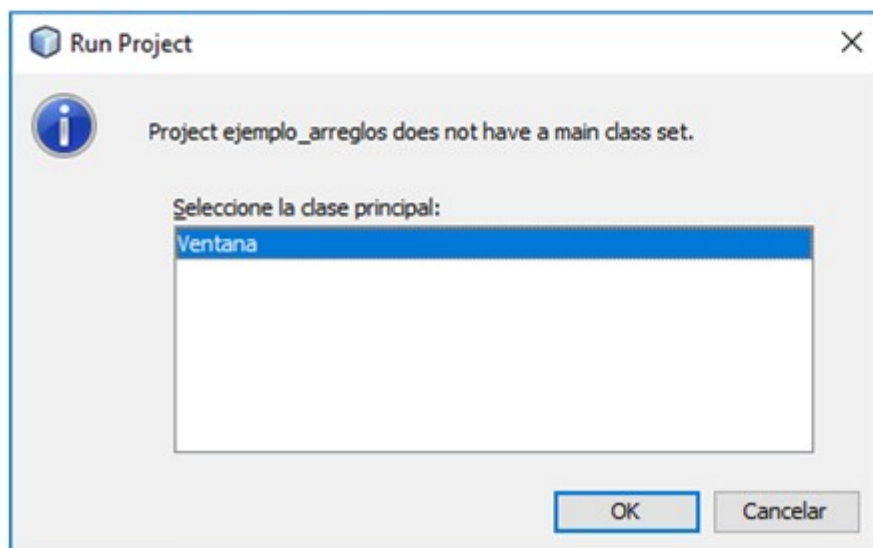
- ➡ Una vez hecho esto puede compilar el programa con la opción **Generar Project** del ítem **Ejecutar** del menú principal o pulsando la tecla F11.



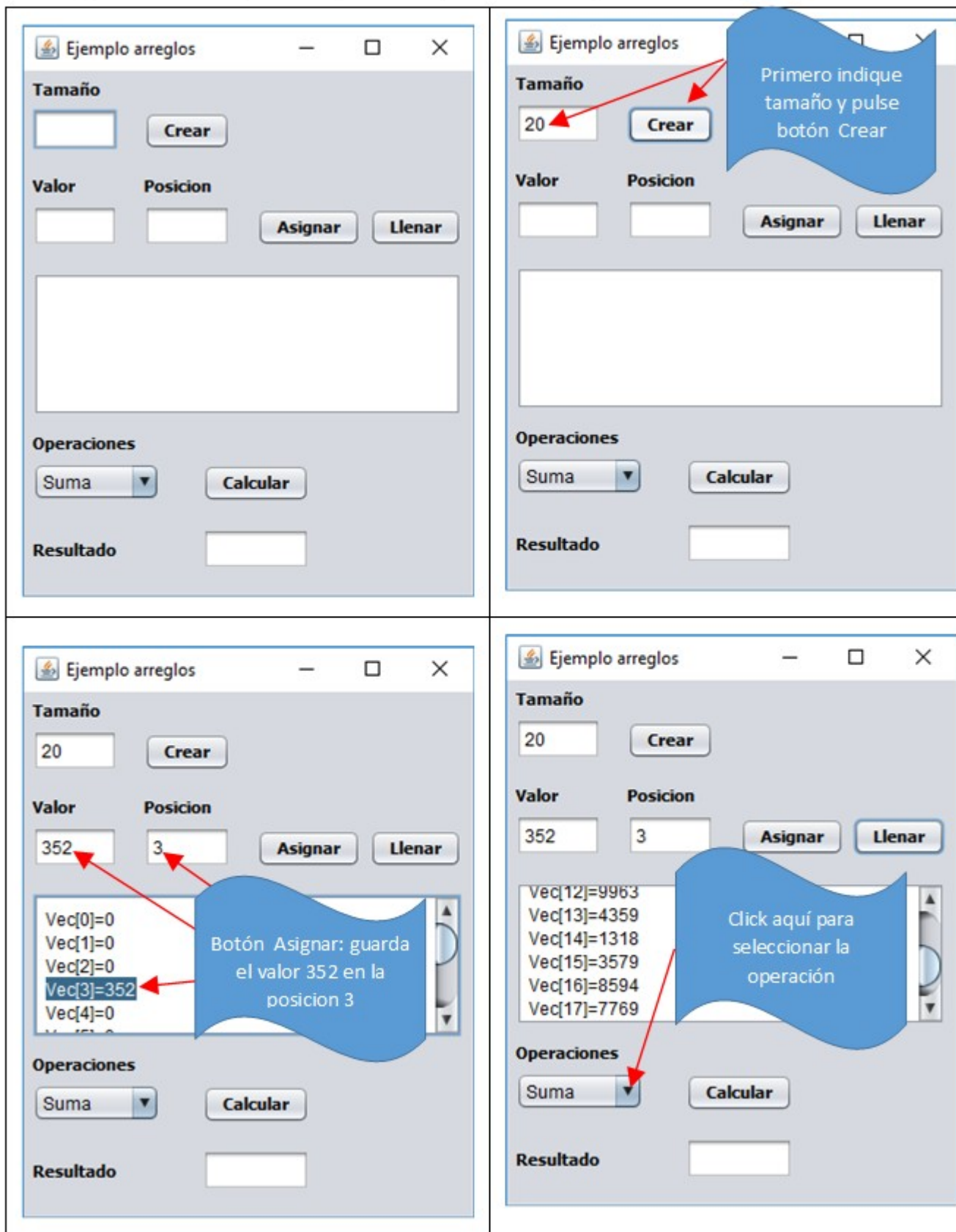
- ➡ Una vez pasada la compilación puede ejecutar el programa, con la opción **Ejecutar Project** del ítem **Ejecutar** del menú principal o pulsando la tecla F6.



- ➡ Cuando se ejecuta el programa por primera vez, *NetBeans* le mostrará el siguiente cuadro de dialogo para usar como clase principal la clase de la ventana (el **JFrame**), de modo que haremos click en el botón **OK**:



➡ Las siguientes son algunas capturas de pantalla de la aplicación en ejecución:



Ejemplo arreglos

Tamaño
20 Crear

Valor Posicion
352 3 Asignar Llenar

Vec[4]=1469
Vec[5]=795
Vec[6]=4039
Vec[7]=9614
Vec[8]=6443
Vec[9]=5895

Operaciones
Suma Calculador

Resultado 118937

Click en calcular para ejecutar la operación

Resultado de suma

Ejemplo arreglos

Tamaño
20 Crear

Valor Posicion
352 3 Asignar Llenar

Vec[0]=4234
Vec[1]=5464
Vec[2]=9400
Vec[3]=8647
Vec[4]=1469
Vec[5]=795

Operaciones
Promedio Calcular

Resultado 5946.85

Resultado de promedio

Ejemplo arreglos

Tamaño
20 Crear

Valor Posicion
352 3 Asignar Llenar

Vec[4]=1469
Vec[5]=795
Vec[6]=4039
Vec[7]=9614
Vec[8]=6443
Vec[9]=5895

Operaciones
Mayor Calcular

Resultado 9963

Resultado del mayor

Ejemplo arreglos

Tamaño
20 Crear

Valor Posicion
352 3 Asignar Llenar

Vec[4]=1469
Vec[5]=795
Vec[6]=4039
Vec[7]=9614
Vec[8]=6443
Vec[9]=5895

Operaciones
Menor Calcular

Resultado 795

Resultado del menor

Ejercicios Propuestos

Para cada uno de los siguientes enunciados, se pide diseñar en UML el diagrama de clases e implementarlo en *Java* con una aplicación de ventana y empleando el IDE de *NetBeans*; tome en cuenta que es suficiente usar los mismos componentes de *Swing*, que empleados en ejemplo presentado anteriormente; es decir, solo usando controles *JLabel*, *JButton*, *JTextField*, *JComboBox* y *JTextArea*.

1. Tenemos un arreglo con las **N** notas de un curso de programación y se desea saber cuántos estudiantes obtuvieron una nota entre 3.5 y 4.5, el total de reprobados, el total de aprobados, el promedio de aprobados, el promedio de los reprobados y el promedio general del curso..
2. En un vector de **N** números enteros, se desea obtener el total de pares e impares, el promedio de los múltiplos de tres que no son múltiplos de 5, la suma de los positivos y la suma de los negativos.
3. Se requiere un arreglo de **M** números enteros; se pide mostrar los valores de las posiciones impares, mostrar los números pares cuyo valor esta entre 100 y 200 e indicar el total de negativos múltiplos de un valor **X** y el total de valores positivos no múltiplos del valor **X**.
4. En un arreglo de tamaño dado por el usuario y números reales, se desea hallar la cantidad de números negativos mayores que -100, la sumatoria de los valores que son inferiores al promedio general y el total de números cuyo primer dígito decimal sea igual a 4.
5. Dado un número **Q** cualquiera, se desea saber cuántas veces se encuentra en un arreglo de tamaño **K** de números reales, cuantos valores del arreglo son mayores, cuantos son menores que dicho número, cuántos de ellos son el inverso aditivo de **Q**.
6. Se tiene un arreglo compuesto **N** caracteres, para el que se pide contar todas las vocales que sean minúsculas, obtener el porcentaje de vocales mayúsculas con respecto al total de caracteres, contabilizar cuantos caracteres son dígitos numéricos, invertir el contenido del arreglo y mostrar por pantalla el arreglo resultante.
7. Se quiere llenar un vector de números enteros de tamaño **M**, con los **M** primeros términos de la serie de fibonacci (1, 1, 2, 3, 5, 8, 13, 21, 34 ...). Además se debe mostrar el arreglo resultante en la ventana y calcular cuántos valores de dos cifras hay en el arreglo.