

UNIVERSIDAD DE CORDOBA

FACULTAD DE INGENIERIAS

PROGRAMA INGENIERIA DE SISTEMAS

CURSO: Programación I

TEMA: Vectores de objetos en Java.

DESCRIPCION:

Este documento desarrolla el tema de vectores unidimensionales de objetos como complemento a los arreglos de tipo primitivo o básico tratados anteriormente en clase, abordando los conceptos relativos al tema, las relaciones de clases en UML para modelar vectores de objetos, la declaración e inicialización de vectores de objetos en el lenguaje Java, las reglas para encapsular atributos de esta naturaleza y un ejemplo de problema resuelto usando una aplicación de consola creada en el IDE *NetBeans* y probada en Microsoft Windows.

OBJETIVOS:

- ✓ Comprender el concepto de vectores o arreglos de objetos y su aplicabilidad en la resolución de problemas de programación.
- ✓ Reconocer los tipos de relación de agregación entre clases para modelar en UML diagramas de clases para vectores de objetos.
- ✓ Diseñar e implementar en el lenguaje Java las clases requeridas para vectores de objetos en la solución de problema ejemplo.


PALABRAS CLAVES: Clases en Java, arreglos de objetos en Java, relaciones de agregación entre clases en UML, aplicaciones GUI en Java, control *JTable*, IDE *NetBeans*.



1. Concepto vectores de objetos

Un vector o arreglo de objetos es aquel en el que podemos almacenar un conjunto de objetos, es decir instancias de una clase cualquiera. De esta manera el tipo de datos almacenado en el vector ya no será de tipo primitivo (**int**, **float**, **char** etc.) como en los casos anteriores, sino que esta vez su tipo de datos será del tipo de una clase que hemos definido anteriormente. En este sentido cada elemento del arreglo contendrá una instancia de una misma clase y por supuesto dicha instancia tendrá su propio estado (conjunto de valores almacenados en sus atributos) eventualmente diferente a los estados de los objetos almacenados en otras posiciones del vector. Así mismo desde cada elemento del arreglo podemos invocar los métodos de un objeto de la clase ubicado en una posición particular del vector, independientemente del resto de objetos del arreglo.

Ahora bien el arreglo de objetos en cuanto a su tamaño siempre será dinámico como en el caso de los arreglos de tipo primitivo, conservando las mismas características ya explicadas en caso para este tipo de vectores; no obstante cuando se crea un arreglo de objetos es necesario tener en cuenta que cada una de las instancias que se pueden almacenar en él no están inicializadas; es decir, todas toman por defecto el valor de nulo (**null**). En este orden de ideas, el lenguaje Java permite definir vectores de objetos de diferentes maneras así:

En cuanto a su representación en UML, el diseño del diagrama de clases se hace considerando la naturaleza de la relación entre el vector de objetos y los objetos propiamente dichos que contiene. En este sentido a nivel del diseño de clases en UML las relaciones entre la clase que contiene al vector de objetos y la clase que define a los objetos contenidos en dicho vector será representada mediante una **relación de agregación** representada en UML por un rombo; recordando que este tipo de relación entre clases se interpreta como una como una relación de la forma todo/parte; en donde evidentemente la clase todo (en la que va el símbolo del rombo) será aquella que contiene como atributo al vector de objetos y la clase parte (hacia donde apunta la línea del rombo) entonces será aquella que modela a los objetos contenidos en el vector.

Este tipo de relación es la apropiada ya que tiene dos formas que se adaptan perfectamente para representar la relación de clases que modela a un vector de objetos. Por un lado con la **relación de agregación por referencia**, (simbolizada por el rombo transparente ) representamos a un vector de objetos cuyos elementos son creados y destruidos por fuera de la clase (el todo) que declara como atributo a un vector de

instancias de clase (las partes); toda vez que como ya se explicó en clases en esta forma de agregación las partes existen independientemente del todo. Por otra parte con una **relación de agregación por valor** también llamada **composición** y simbolizada por un rombo relleno , representamos a un vector de objetos cuyos elementos son creados y destruidos por la misma la clase (el todo) que contiene como atributo a un vector de instancias de clase (las partes); ya que en la relación de composición las partes existen dependiendo  de la existencia del todo.

Sintaxis para declarar vectores de objetos.

En cuanto a la sintaxis en el lenguaje Java para dedarar vectores de objetos de es la siguiente:

```
NombreClase NombreVector [ ];
```

Donde *NombreClase* es el tipo de la clase que le corresponde a cada uno de los objetos del vector. La inicialización del vector sigue las mismas reglas usadas para inicializar vectores de tipos primitivos, para ello la sintaxis es la siguiente:

```
NombreVector = new NombreClase [Tamaño];
```

Donde *NombreClase* es la clase a la que pertenecen los objetos del vector y tamaño es la cantidad de objetos (instancias) que almacenara el vector. En cuanto al tamaño se refiere éste debe ser una constante, valor numérico fijo o expresión numérica constante y siempre de tipo entero (**int**, **long**, o **byte**). Así por ejemplo, digamos que tenemos una clase llamada *TPersona* que representa la información básica de una persona y que requerimos un vector (que llamaremos *VecPer*) para contener instancias de esa clase, para lo cual lo podemos definir así:

```
TPersona VecPer[ ];
```

Si queremos inicializar este arreglo de objetos con una capacidad para contener 100 instanciase de la clase *TPersona*, la sentencia en Java seria como se muestra a continuación:

```
VecPer = new TPersona[100];
```

En este punto vale la pena recordar que la inicialización de un vector de objetos no incluye la inicialización individual de cada uno de sus elementos (objetos o instancias), ya que la inicialización presentada anteriormente solo reserva espacio para cada instancia y no ejecuta el constructor por cada una de ellas. Una vez se reserva el espacio para los objetos

en el vector cada uno de dichos objetos se inicializa en **null** (nulo) de forma automática y por ello debemos inicializar "manualmente" cada elemento del vector. Seguidamente presentamos el fragmento de código necesario para declarar e inicializar cada objeto de un vector de 100 objetos de la clase *TPersona* considerando anteriormente:

```
TPersona VecPer [ ]; //Declaración del vector de objetos
VecPer = new TPersona[100]; //Inicialización del vector de objetos
int i;
for(i=0;i<100;i++){
    VecPer[i]=new Persona(); //Inicialización cada instancia del vector
}
```

Como puede observar cada elemento del vector se referencia por su posición mediante un numero entero indicado entre corchetes, que para el caso va de 0 a 99 y es representado por la variable *i*, al igual que en un vector normal. Pero al contenido de cada posición (elementos del vector) por ser objetos se les debe inicializar con el constructor de la clase, como si se tratase de una variable de tipo clase (instancia u objeto) normal, similar a las que ya hemos usado en otras ocasiones. La inicialización de cada instancia del vector realizada por el ciclo es obligatoria antes de efectuar cualquier otra operación con los elementos del vector.

Para ejecutar métodos de cualquier instancia accedemos a su posición indicada entre corchetes, seguido del punto y del nombre del método a usar. Así por ejemplo para asignar el valor de 25 como edad a la persona del puesto 5 e imprimir los apellidos de la persona del puesto 20 procederíamos así:

```
VecPer[5].setEdad(25);
System.out.println("Apellidos: " + VecPer[20].getApellidos());
```

De igual manera a cualquier elemento del arreglo se le puede asignar cualquier objeto o instancia de la clase con la que se declaró el vector; incluso podemos usar el valor de **null** para los objetos del vector; para el caso por ejemplo de liberar memoria, tal como se aprecia en la siguiente línea para el caso del elemento del puesto número diez en la línea siguiente:

```
VecPer[10] = null;
```

En cuanto al encapsulado de un vector de objetos el prototipo (diseño) e implementación de los métodos modificadores y selectores es el mismo, ya que cada uno de ellos recibe un parámetro de tipo entero (**int**) que especifica la posición del elemento (objeto) dentro del vector. Lo que cambia es el tipo de datos al cual hace referencia el elemento del vector, pues ya no se trata de un valor de tipo primitivo sino de una instancia de clase; por lo tanto, al método modificador se le pasa por parámetro una instancia del tipo de clase con la cual

se declaró el vector, y el método selector retorna un objeto de dicha clase. De esta forma continuando con las declaraciones hechas anteriormente, los métodos modificadores y selectores para este vector de objetos son como siguen:

```
public void setVecPer(int pos, Persona Per){
    VecPer[pos] = Per;
}

public Persona getVecPer(int pos){
    return VecPer[pos];
}
```

2. Presentación ejemplo vectores de objetos

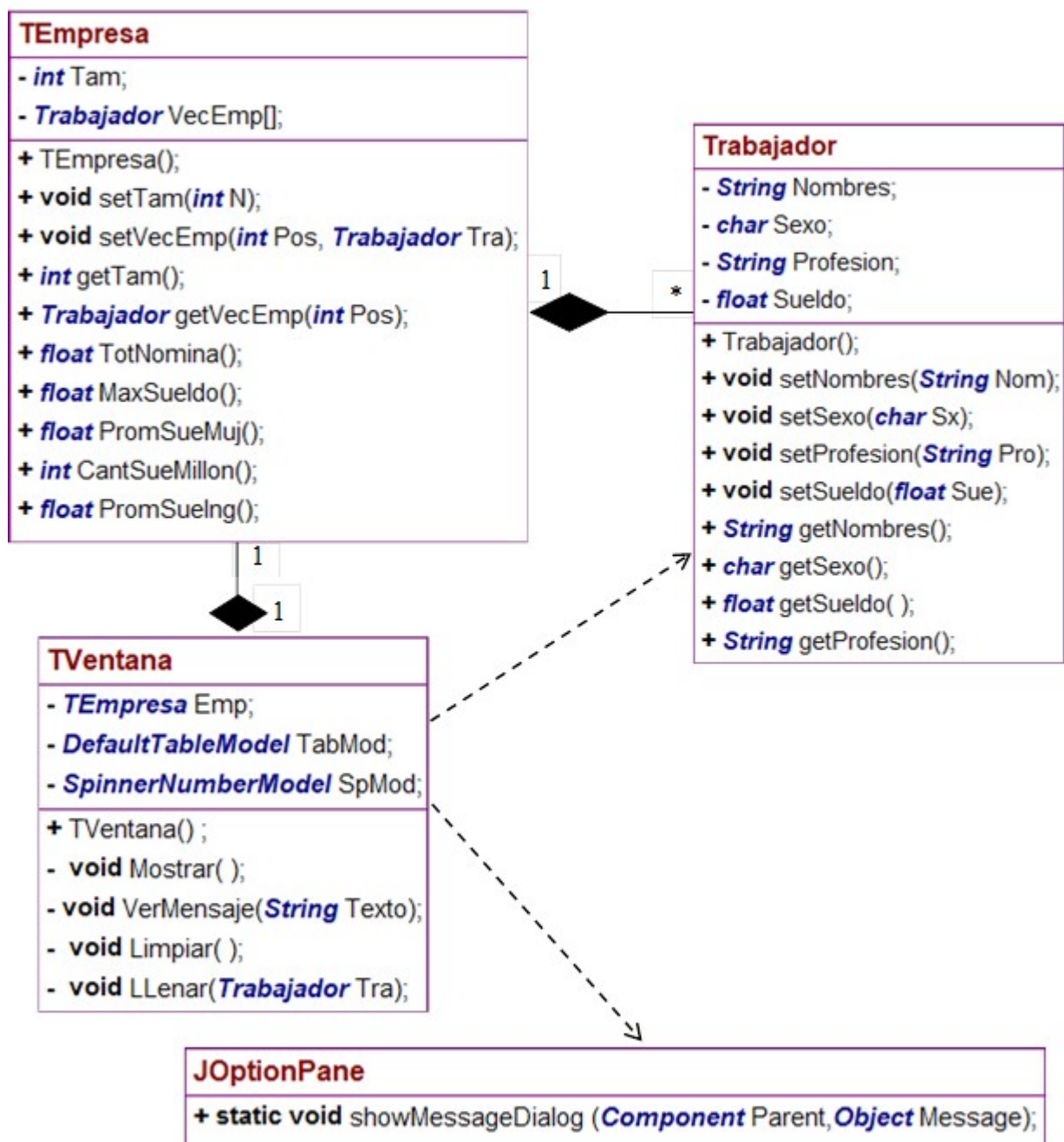
Para este ejemplo vamos a considerar una empresa que tiene un conjunto de trabajadores de los que se registran sus nombres, sexo, profesión y sueldo. Con esta información se pide diseñar e implementar las clases necesarias en Java para representar un vector de objetos de tamaño indicado por el usuario, que además permita realizar las siguientes operaciones:

- ♦ Valor total de la nómina.
- ♦Cuál es el sueldo más alto.
- ♦ Determinar el promedio de sueldo de las mujeres.
- ♦ Cuantos empleados ganan más de un millón.
- ♦Cuál es el promedio de sueldo de los ingenieros de sistemas.

La clase de la ventana principal tendrá una lista desplegable (**JComboBox**) para todas estas operaciones, además de controles para llamar a métodos que llenen el arreglo de trabajadores y mostrar los datos de cada empleado en un control **JTable**.

3. Diagrama UML de clases para el vector de objetos.

En la siguiente imagen de abajo observamos el diagrama de clases UML que implementaremos como solución del problema planteado anteriormente.



En el diseño UML del diagrama de clases vemos una relación de composición entre la clase *TEmpresa* (clase todo) y la clase *Trabajador* (clase parte), toda vez que la primera clase contiene un vector de objetos de la segunda clase. En este sentido esperamos que todas las instancias de la clase *Trabajador* contenidas en el vector de la clase todo, sean creadas automáticamente por el método modificador del atributo tamaño de esta última. Por otra

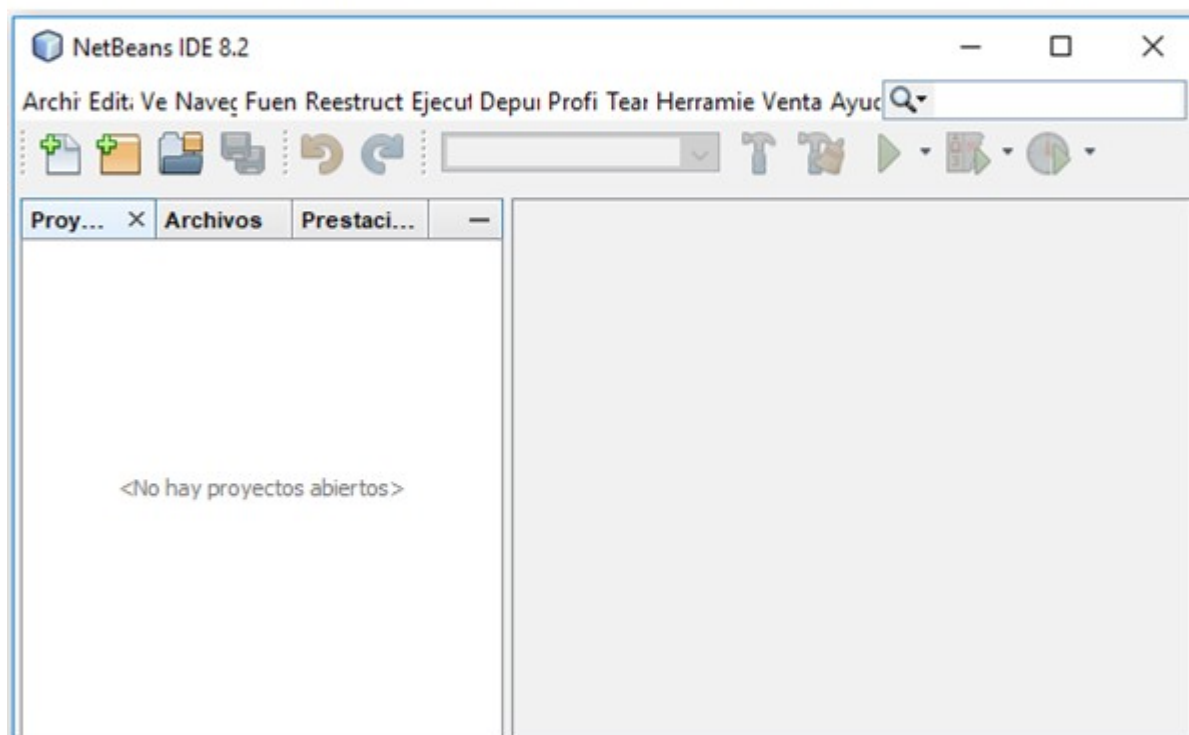
tiene el método *Llenar* que requieren una instancia de la clases *Trabajador*. La segunda dependencia que presenta clase *TVentana* es hacia la clase ***JOptionPane***, pues en la implementación del método *VerMensaje*, usaremos su método *showMessageDialog*.

4. Creación del proyecto para el programa de consola.

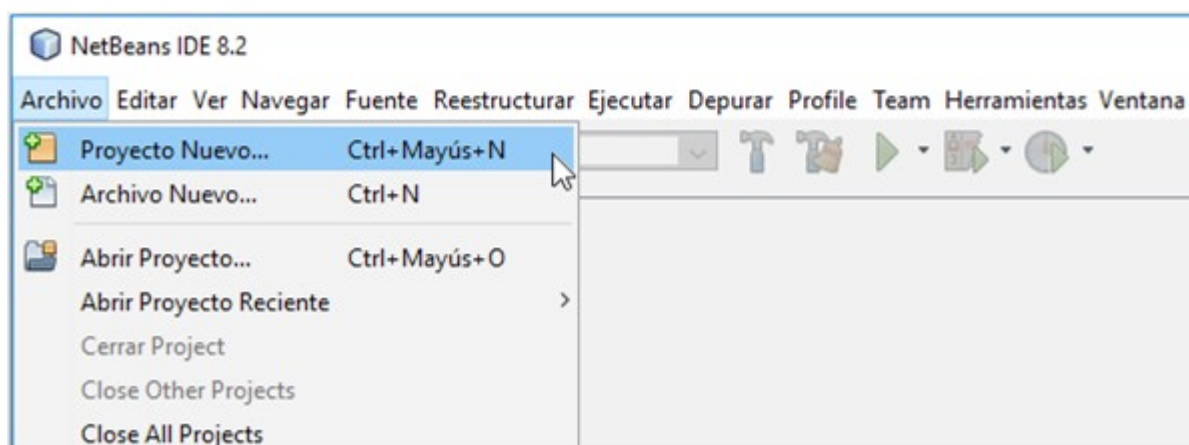
Seguidamente se presentan los pasos para crear un proyecto de consola empleando el IDE *NetBeans*.

a. Creación del proyecto de consola en NetBeans

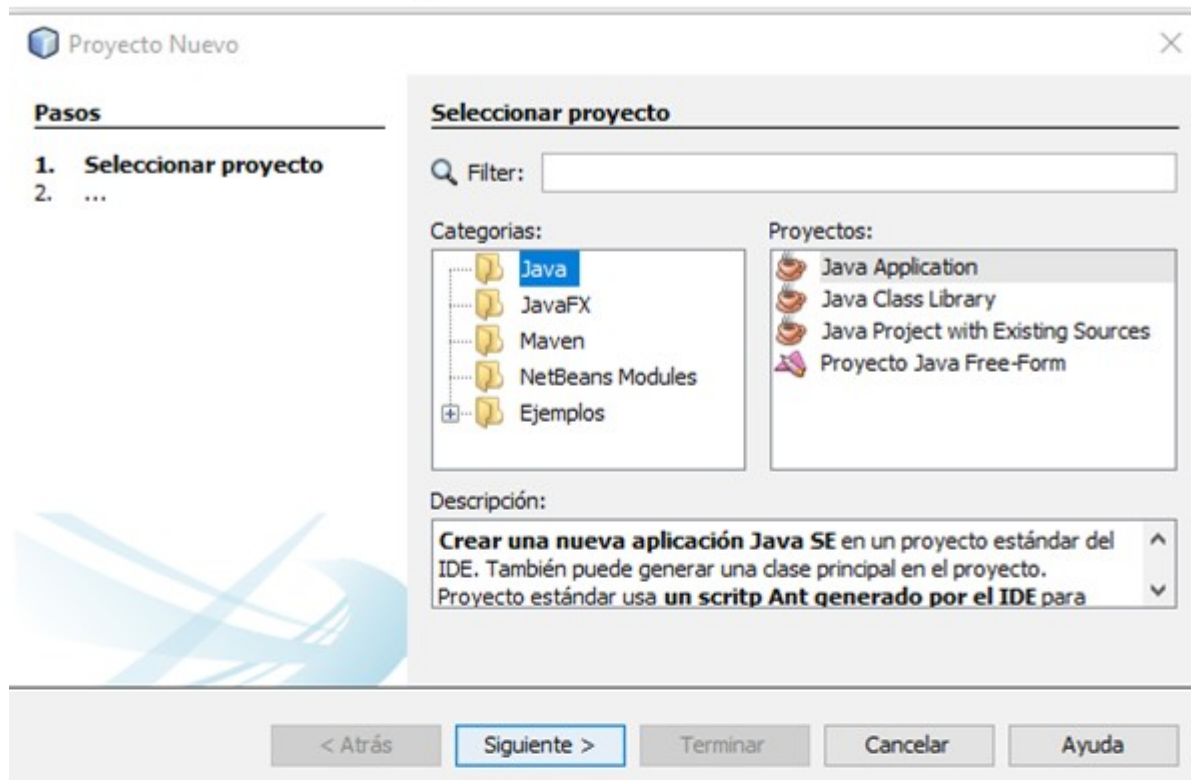
Inicie el IDE de *NetBeans* con lo cual se muestra una pantalla como siguiente:



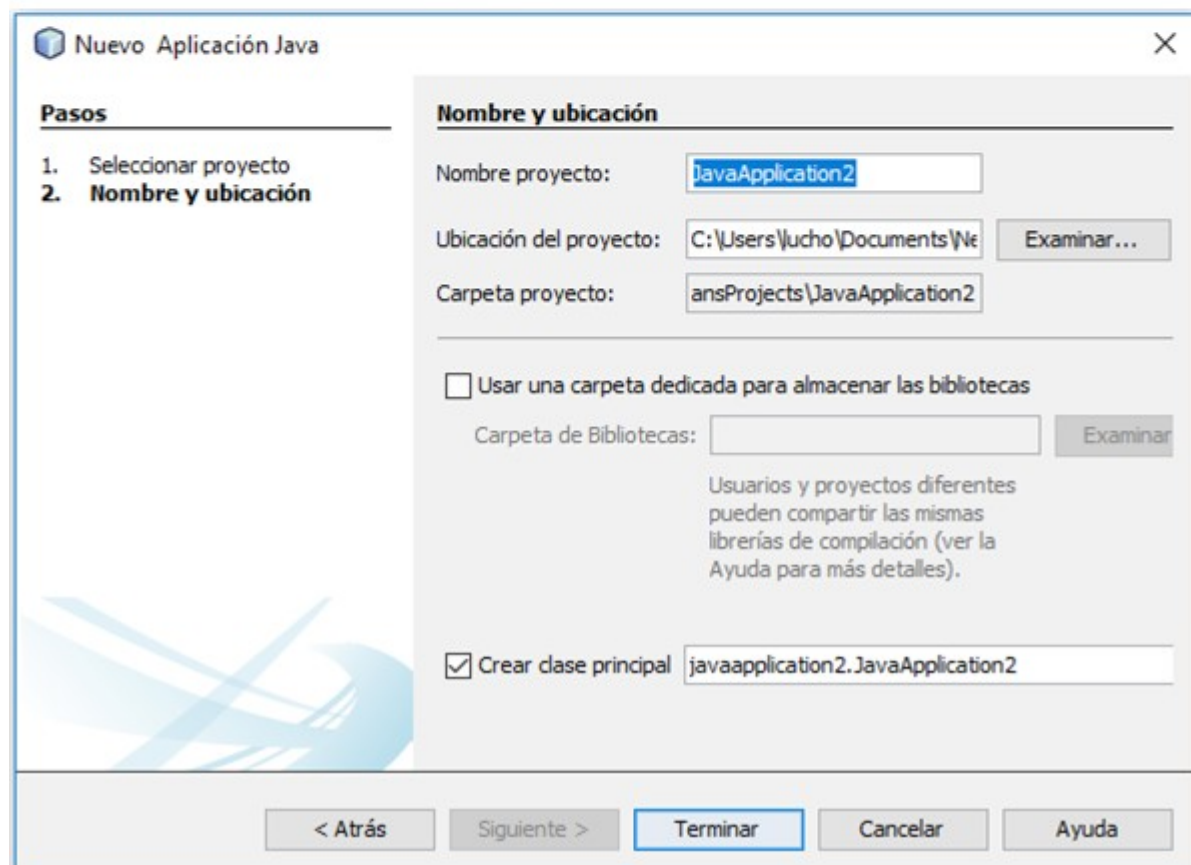
Para crear el proyecto entre por la opción ***Archivo + Nuevo Proyecto***, tal como lo indica la siguiente imagen:



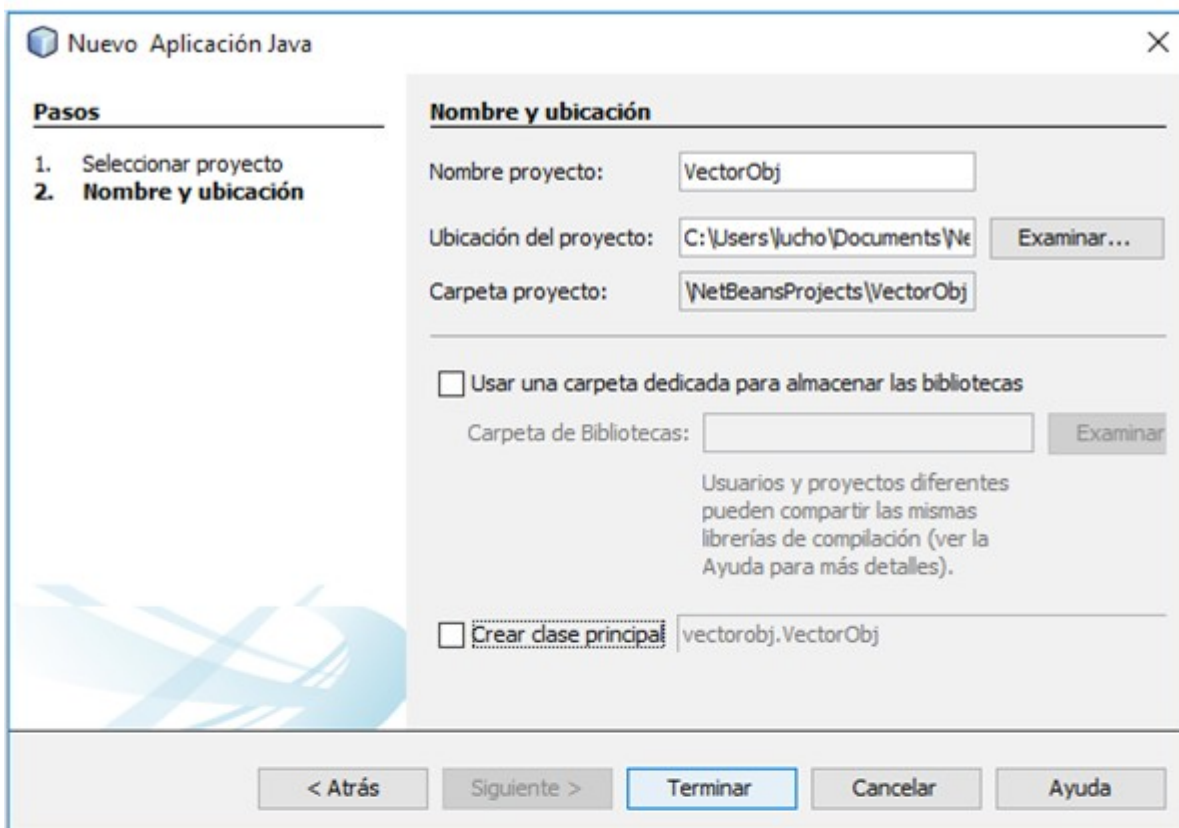
Presentándose la ventana siguiente:



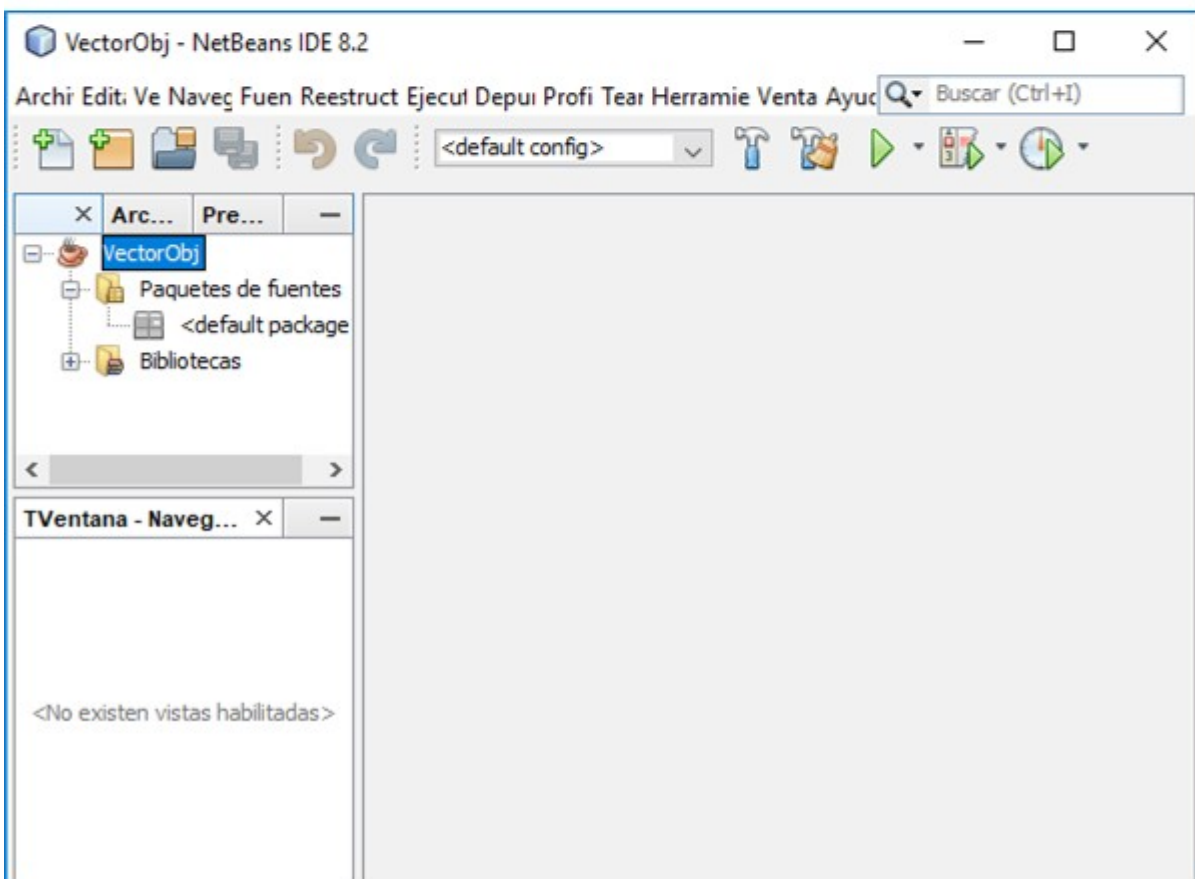
En esta ventana seleccione el nodo Java en panel de categorías, en el panel de proyectos seleccione el nodo "**Java Application**" y pulse el botón "**Siguiente**": Con ello se despliega la ventana ilustrada en la imagen siguiente:



En esta ventana en *nombre del proyecto* ponga **VectorObj**, desmarque la entrada "Crear clase principal" y haga click en el botón "Terminar".

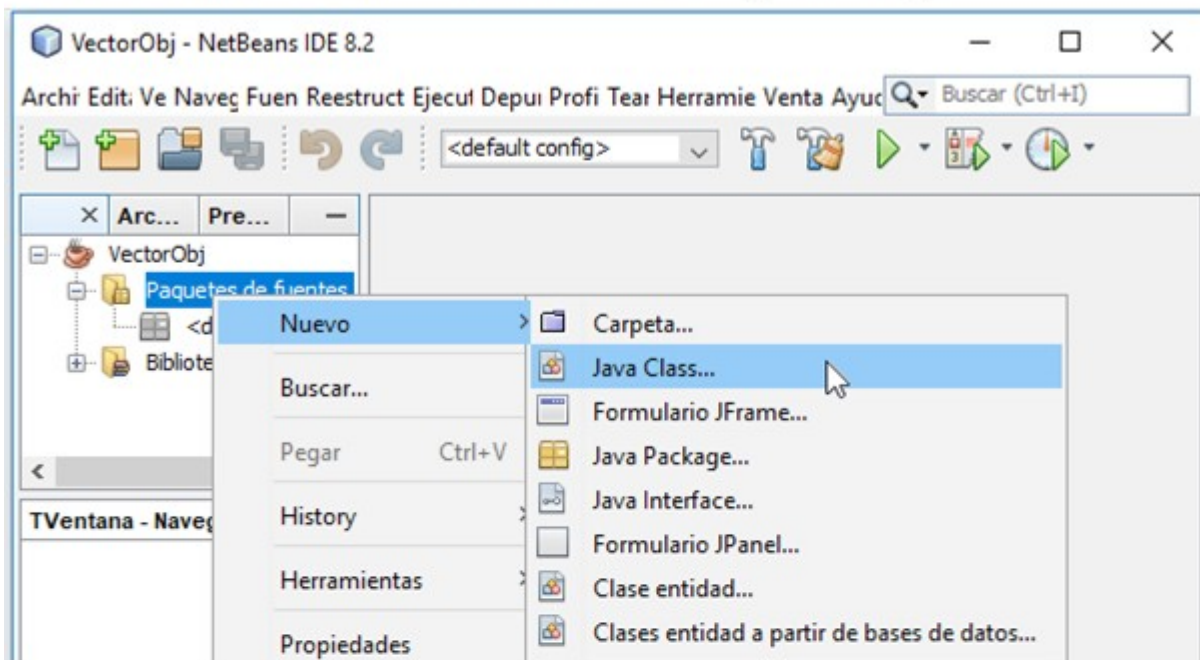


La imagen de abajo muestra el estado inicial del proyecto:

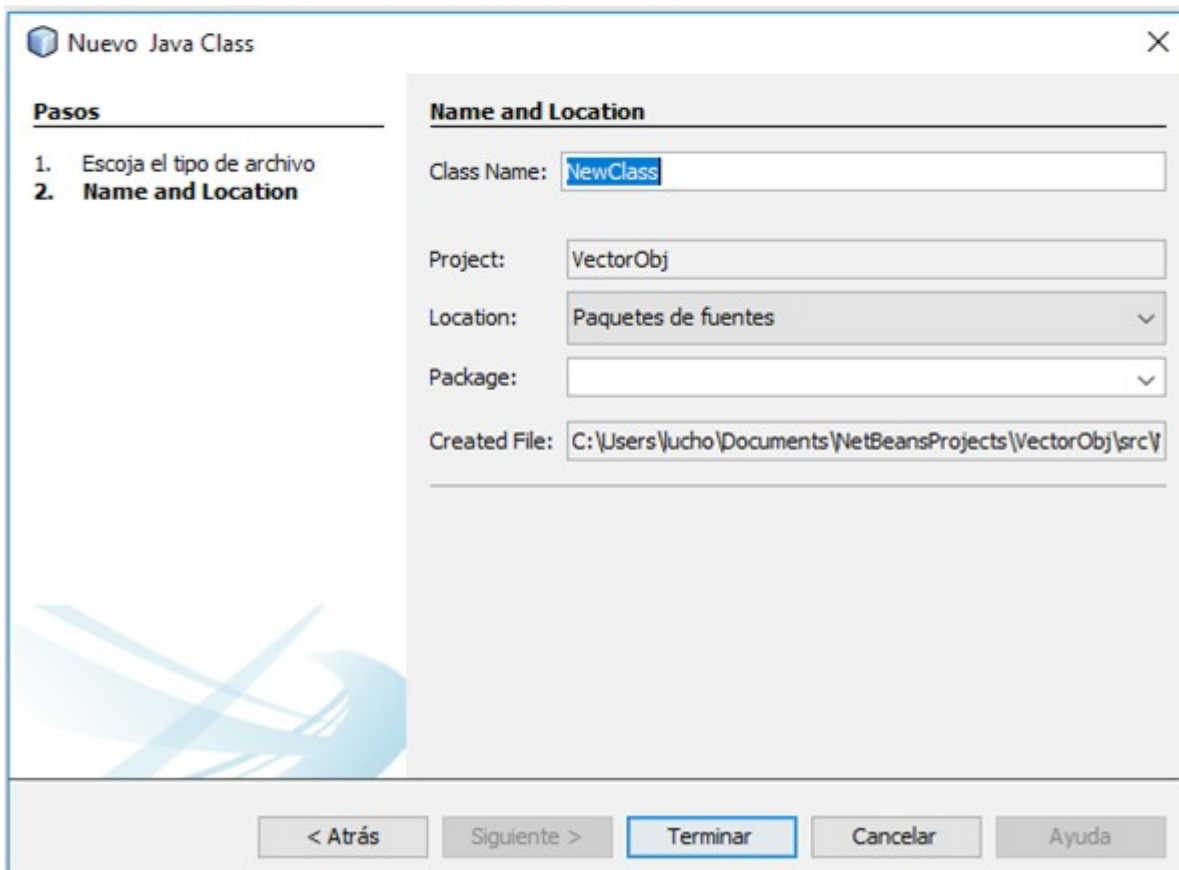


b. Creación de la clase Trabajador

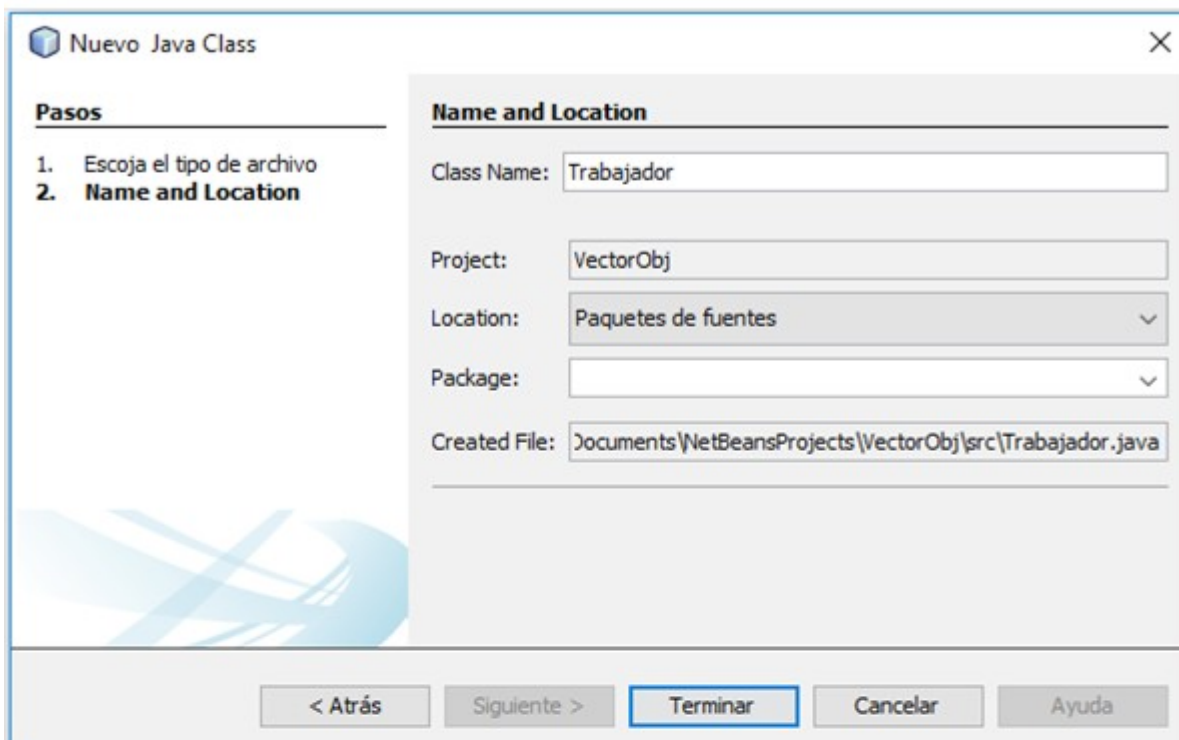
Para crear la clase Trabajador haga click derecho en el nodo paquete de fuentes y seleccione **Nueva + Java Class** como se ve en la siguiente imagen:



Con ello se despliega la ventana ilustrada abajo:



En la ventana anterior ingrese *Trabajador* por nombre de clase (**Class Name**), como se ve en la siguiente imagen. 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:

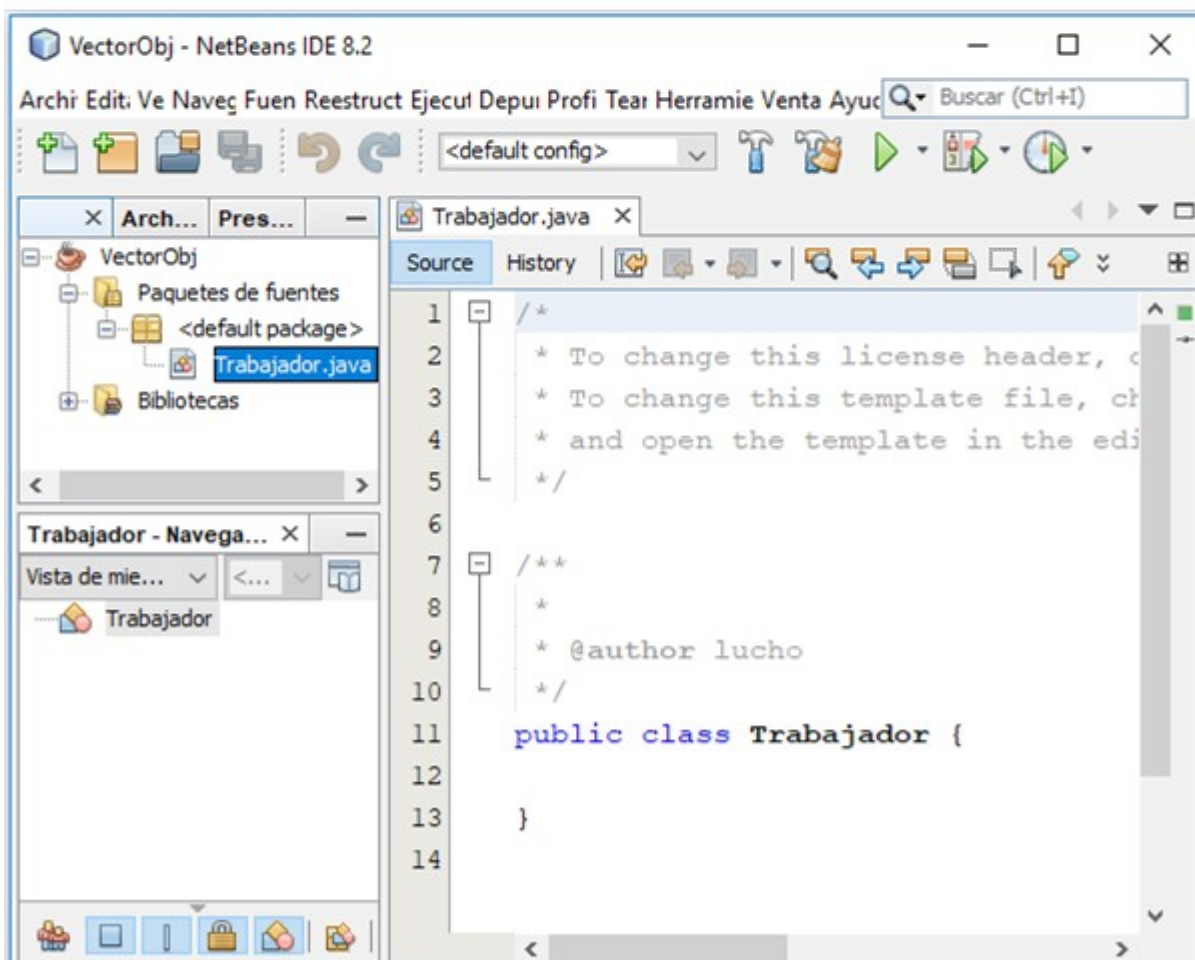
Location:

Package:

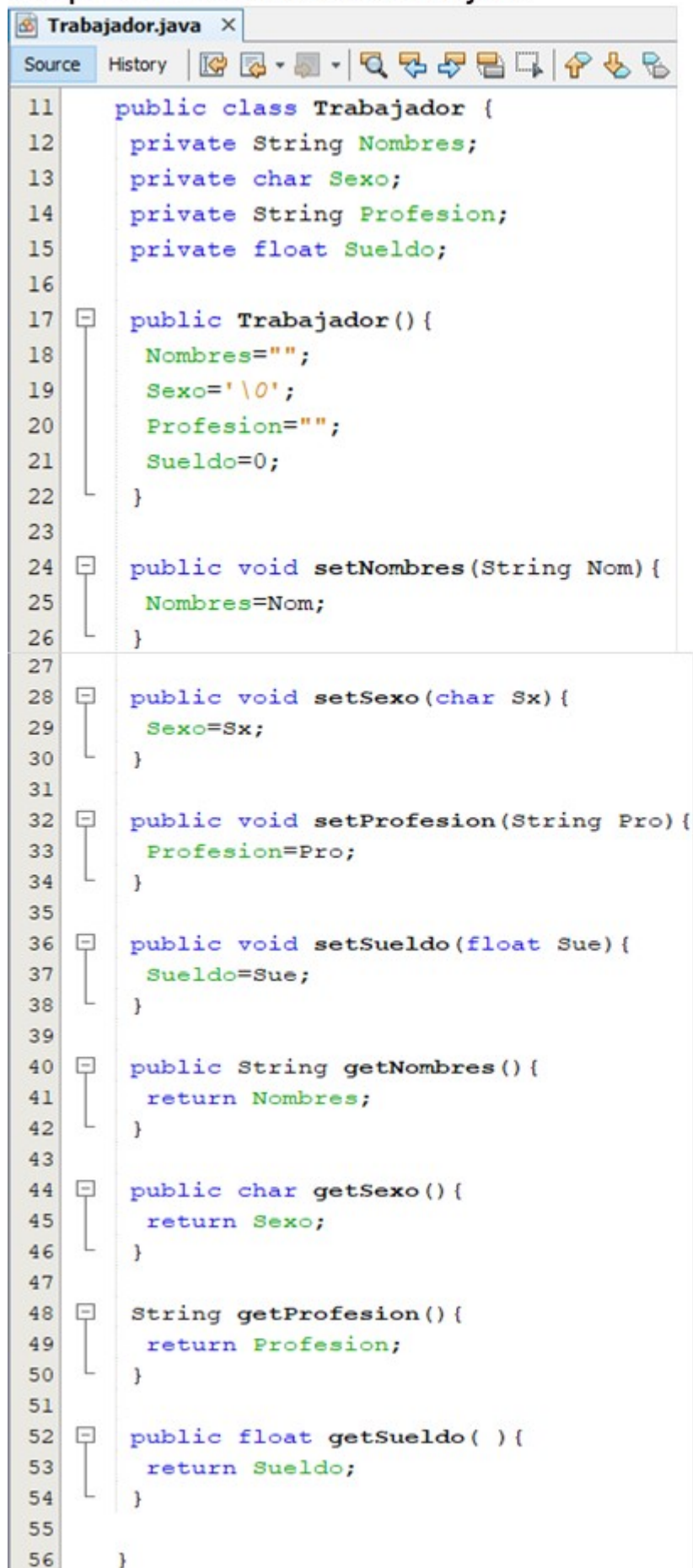
Created File:

< Atrás Siguiete > **Terminar** Cancelar Ayuda

Quedando el proyecto como se aprecia a continuación



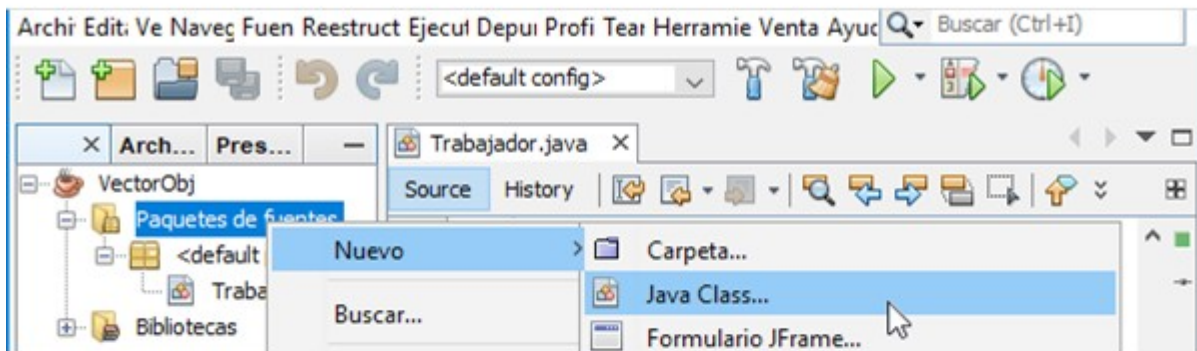
c. Implementación de la clase *Trabajador*



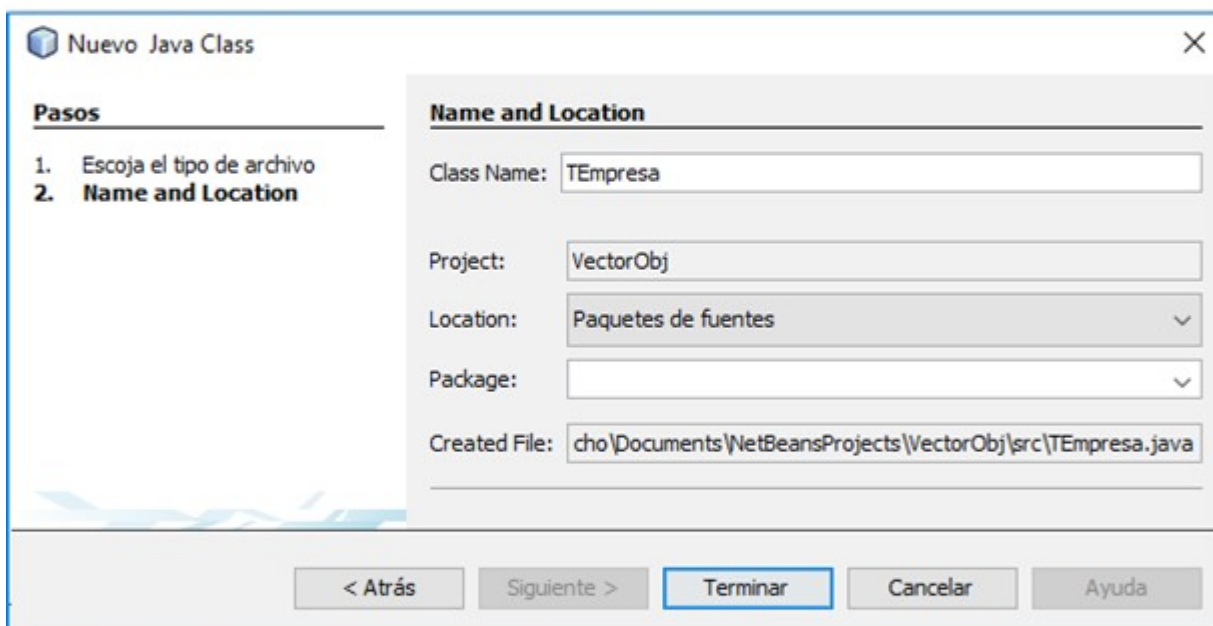
```
Trabajador.java x
Source History
11 public class Trabajador {
12     private String Nombres;
13     private char Sexo;
14     private String Profesion;
15     private float Sueldo;
16
17     public Trabajador() {
18         Nombres="";
19         Sexo='\0';
20         Profesion="";
21         Sueldo=0;
22     }
23
24     public void setNombres(String Nom) {
25         Nombres=Nom;
26     }
27
28     public void setSexo(char Sx) {
29         Sexo=Sx;
30     }
31
32     public void setProfesion(String Pro) {
33         Profesion=Pro;
34     }
35
36     public void setSueldo(float Sue) {
37         Sueldo=Sue;
38     }
39
40     public String getNombres() {
41         return Nombres;
42     }
43
44     public char getSexo() {
45         return Sexo;
46     }
47
48     String getProfesion() {
49         return Profesion;
50     }
51
52     public float getSueldo() {
53         return Sueldo;
54     }
55
56 }
```

d. Creación de la clase *TEmpresa*.

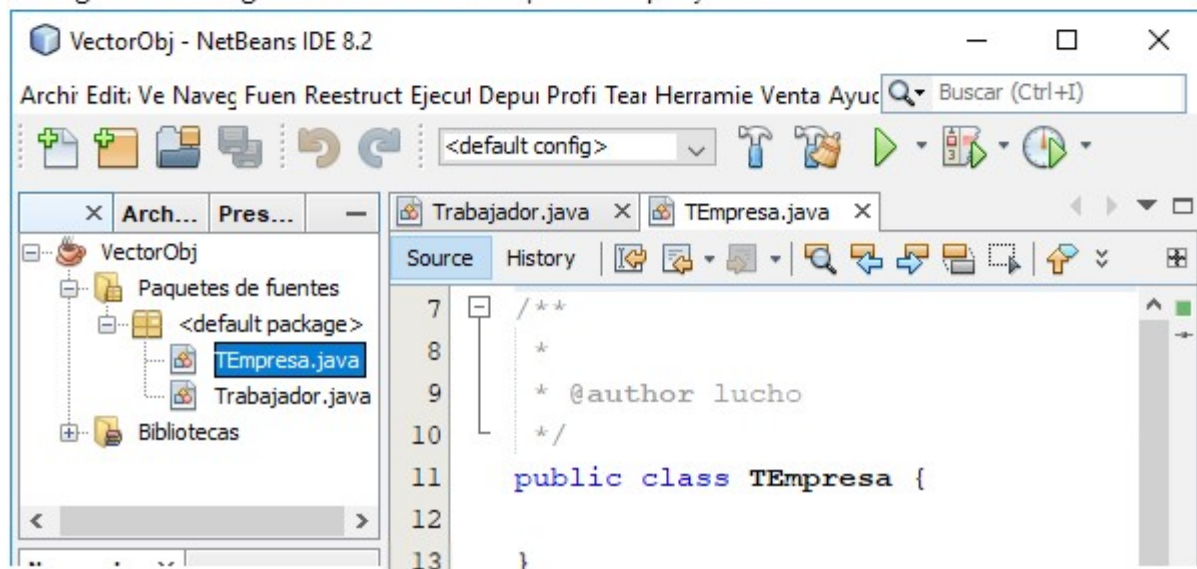
Haga click derecho en el nombre del paquete del proyecto y seleccione las opciones *Nueva + Java Class* como se indica abajo:



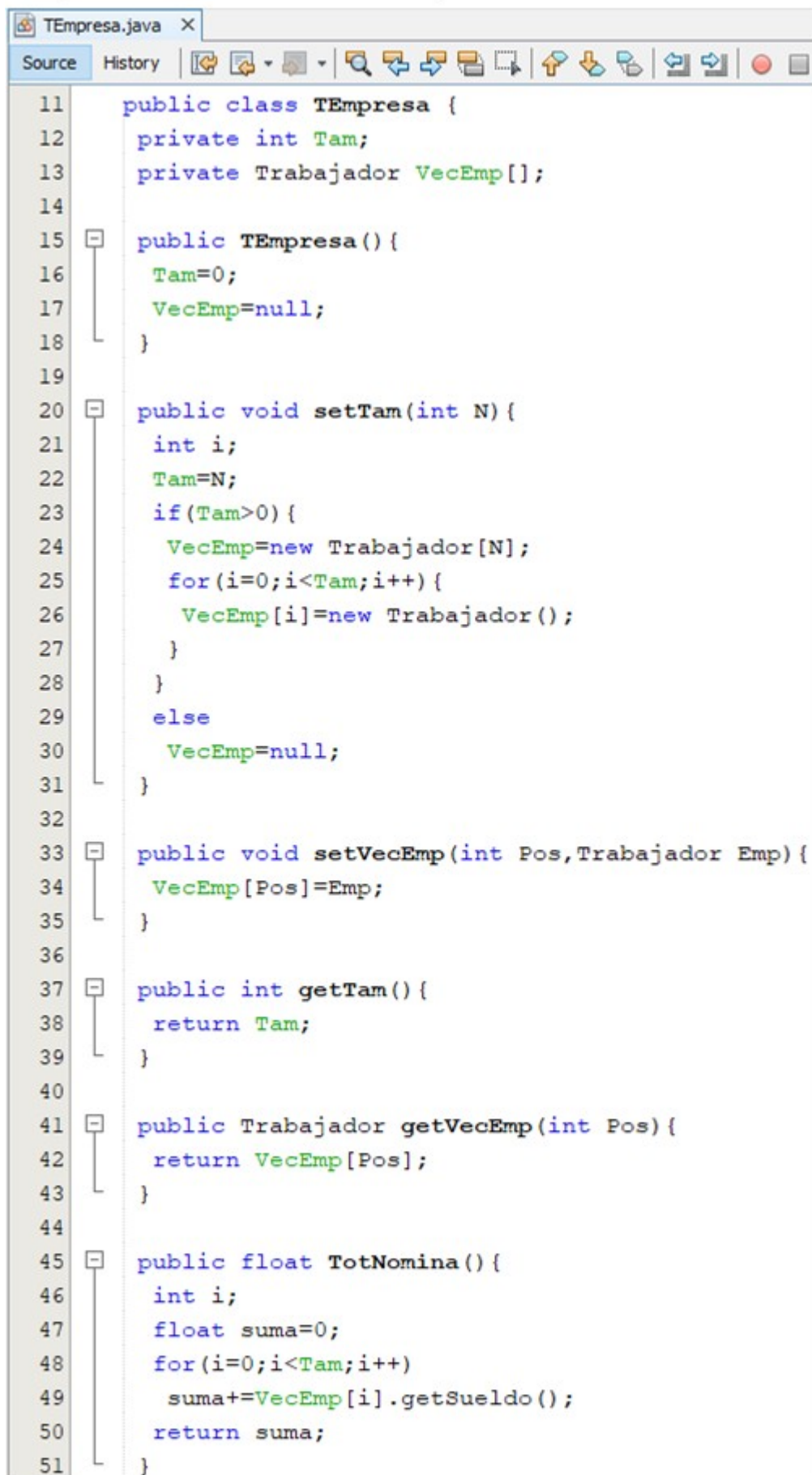
Luego se muestra la ventana de abajo, en la que debe ingresar el nombre *TEmpresa* en "Class Name", y haga click en el botón "Terminar".



La siguiente imagen muestra cómo queda el proyecto:



e. Implementación de la clase *TEmpresa*



The screenshot shows a Java IDE window titled "TEmpresa.java". The code defines a class `TEmpresa` with the following attributes and methods:

```
11 public class TEmpresa {
12     private int Tam;
13     private Trabajador VecEmp[];
14
15     public TEmpresa() {
16         Tam=0;
17         VecEmp=null;
18     }
19
20     public void setTam(int N) {
21         int i;
22         Tam=N;
23         if (Tam>0) {
24             VecEmp=new Trabajador[N];
25             for(i=0;i<Tam;i++){
26                 VecEmp[i]=new Trabajador();
27             }
28         }
29         else
30             VecEmp=null;
31     }
32
33     public void setVecEmp(int Pos,Trabajador Emp) {
34         VecEmp[Pos]=Emp;
35     }
36
37     public int getTam() {
38         return Tam;
39     }
40
41     public Trabajador getVecEmp(int Pos) {
42         return VecEmp[Pos];
43     }
44
45     public float TotNomina() {
46         int i;
47         float suma=0;
48         for(i=0;i<Tam;i++)
49             suma+=VecEmp[i].getSueldo();
50         return suma;
51     }
52 }
```



```

52
53 public float MaxSueldo() {
54     int i;
55     float sueldo,max;
56     max=VecEmp[0].getSueldo();
57     for(i=1;i<Tam;i++){
58         sueldo=VecEmp[i].getSueldo();
59         if(sueldo>max)
60             max=sueldo;
61     }
62     return max;
63 }
64
65 public float PromSueMuj() {
66     int i;
67     char sexo;
68     int cant=0;
69     float suma=0;
70     for(i=0;i<Tam;i++){
71         sexo=VecEmp[i].getSexo();
72         if(sexo=='f' || sexo=='F'){
73             cant=cant + 1;
74             suma=suma + VecEmp[i].getSueldo();
75         }
76     }
77     if(cant>0)
78         return suma/cant;
79     else
80         return 0;
81 }
82
83 public int CantSueMillon() {
84     int i;
85     int cant=0;
86     for(i=0;i<Tam;i++)
87         if(VecEmp[i].getSueldo()>1000000)
88             cant=cant + 1;
89     return cant;
90 }
91
92 public float PromSueIng() {
93     int i;
94     int cant=0;

```

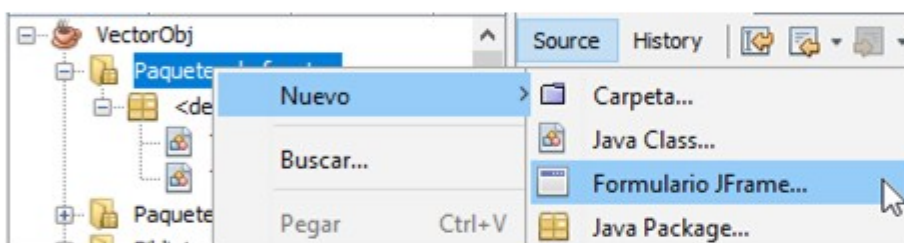
```

float suma=0;
for(i=0;i<Tam;i++)
    if(VecEmp[i].getProfesion().compareTo("ingeniero de sistemas")==0){
        cant=cant + 1;
        suma+=VecEmp[i].getSueldo();
    }
if(cant>0)
    return suma/cant;
else
    return 0;
}

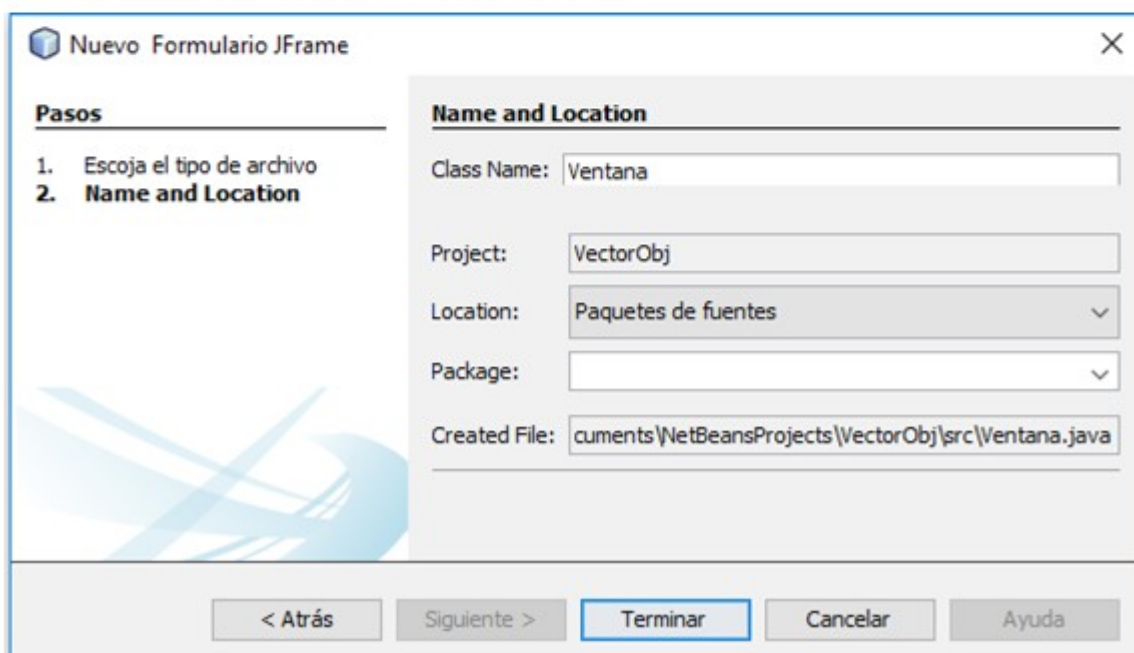
```

f. Diseño gráfico de la ventana

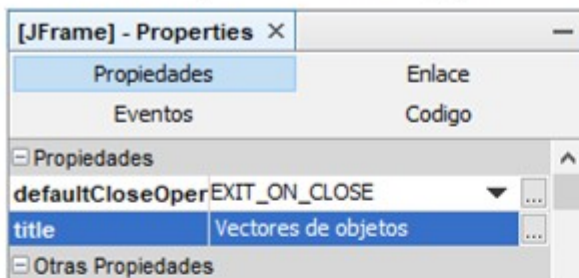
- ➡ Haga click derecho en nodo **Paquetes de fuentes** del proyecto, escoja la opción **Nuevo** y luego **Formulario JFrame**, tal como se muestra en la siguiente imagen:



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



- ➡ Haga click dentro de la ventana y luego vaya al inspector de propiedades; ubique la propiedad **title**, haga click en el recuadro de su derecha y escriba **Vectores de objetos** como el título para la ventana y pulse enter:



- ➡ Cambie el tamaño de la ventana, para ello haga click en el signo más (+) del nodo "**Otras propiedades**" del inspector de propiedades:



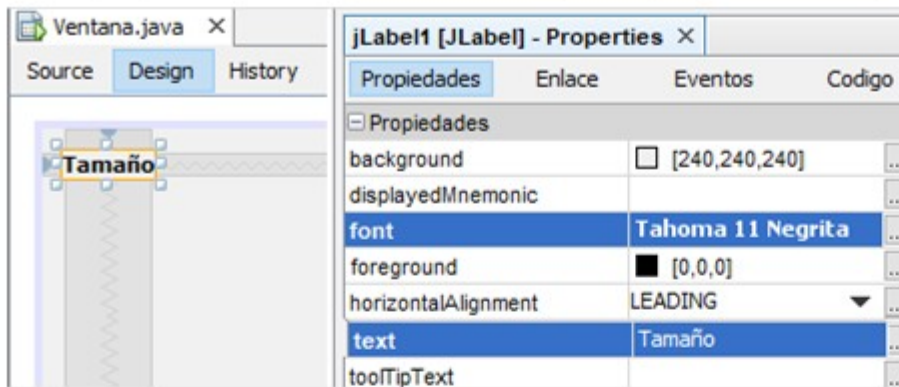
- ➡ Redimensione la ventana (sujetándola por el nodo inferior derecho) hasta que tenga un tamaño de 490 (ancho) por 375 (alto).



- ➡ Arrastre un **JLabel** (etiqueta) sobre la ventana hasta el punto indicado:



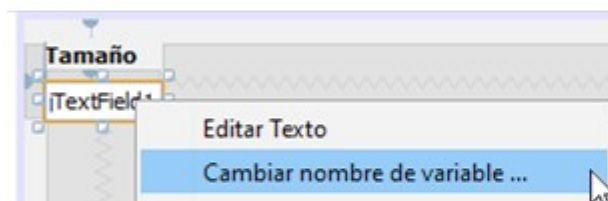
- ➔ En la propiedad **text** escriba **Tamaño** y en la propiedad **font** ponga el texto en negrilla.



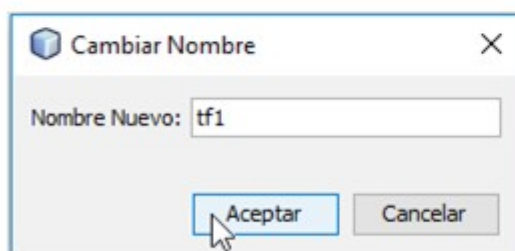
- ➔ Arrastre un **JTextField** hasta el punto indicado:



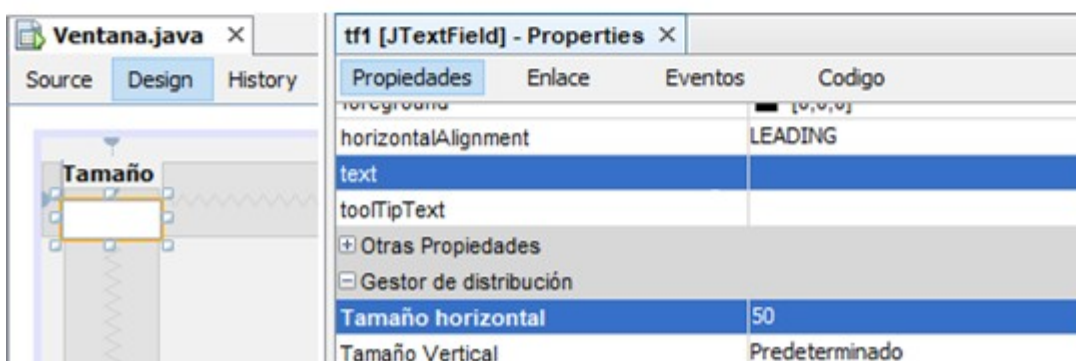
- ➔ Haga click derecho sobre el **JTextField** y seleccione la opción "**cambiar nombre de variable ...**".



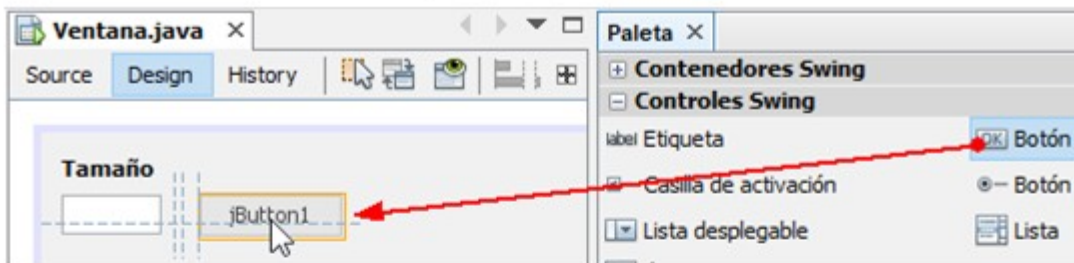
- ➔ Escriba **tf1** como nombre de instancia y haga click en el botón **Aceptar**.



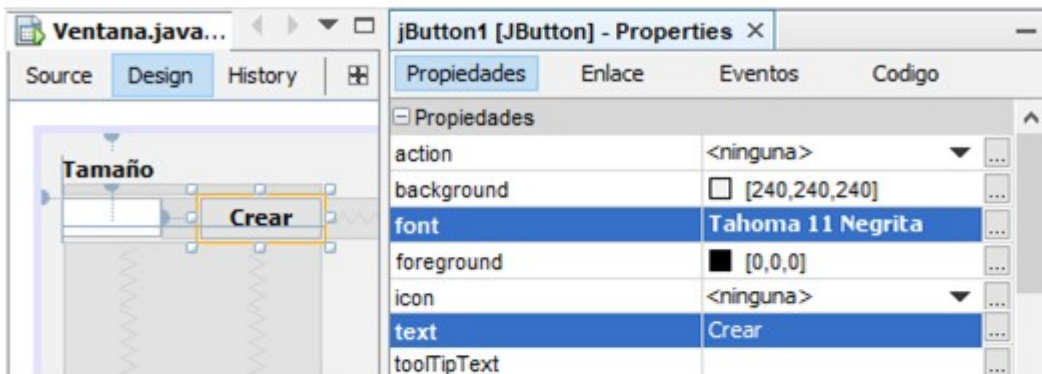
- ➔ En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo "**Gestor de distribución**", ponga 50 como valor para la propiedad **Tamaño horizontal**:



- Arrastre un **JButton** hasta la posición señalada:



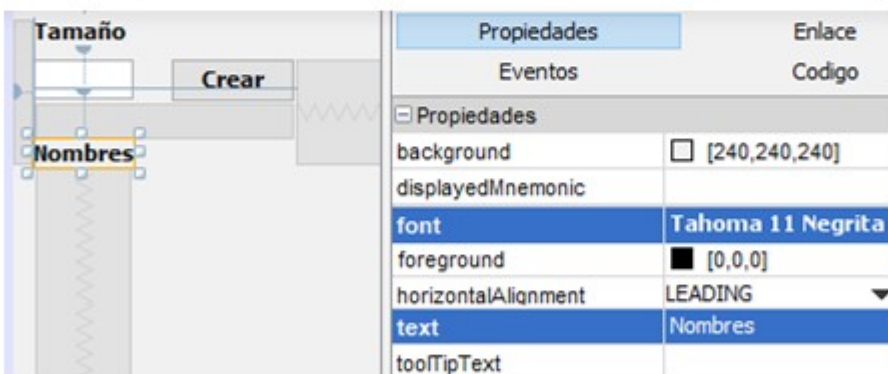
- En el inspector de propiedades, ponga **Crear** en la propiedad **text** y en la propiedad **font** asigne texto en negrilla.



- Arrastre un **JLabel** hasta el punto indicado:



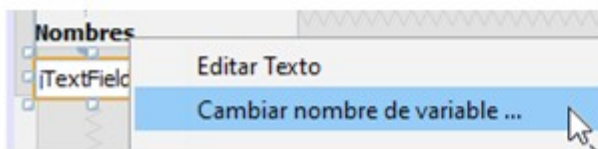
- En la propiedad **text** escriba **Nombres** y en la propiedad **font** ponga el texto en negrilla.



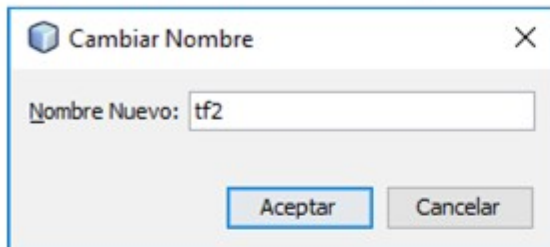
- Arrastre un **JTextField** hasta la posición señalada:



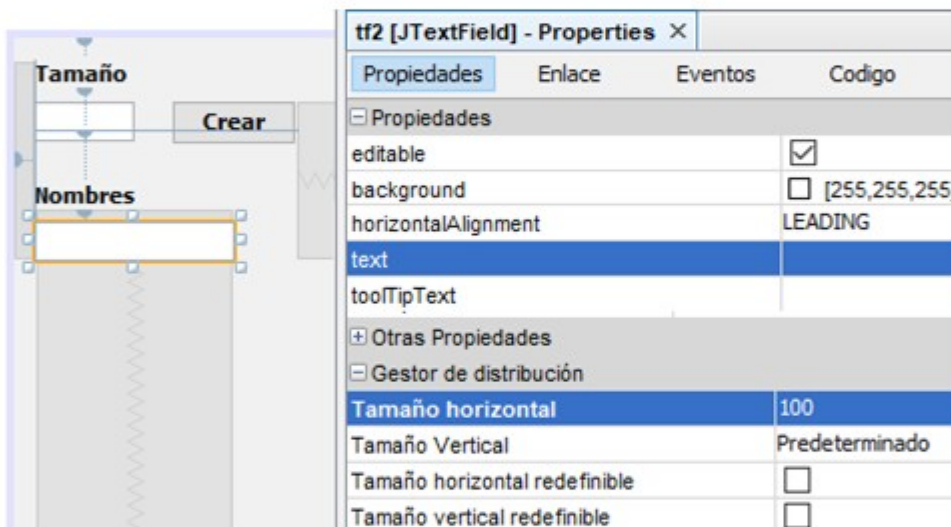
- ➡ Haga click derecho sobre el **JTextField** y seleccione la opción "**cambiar nombre de variable ...**".



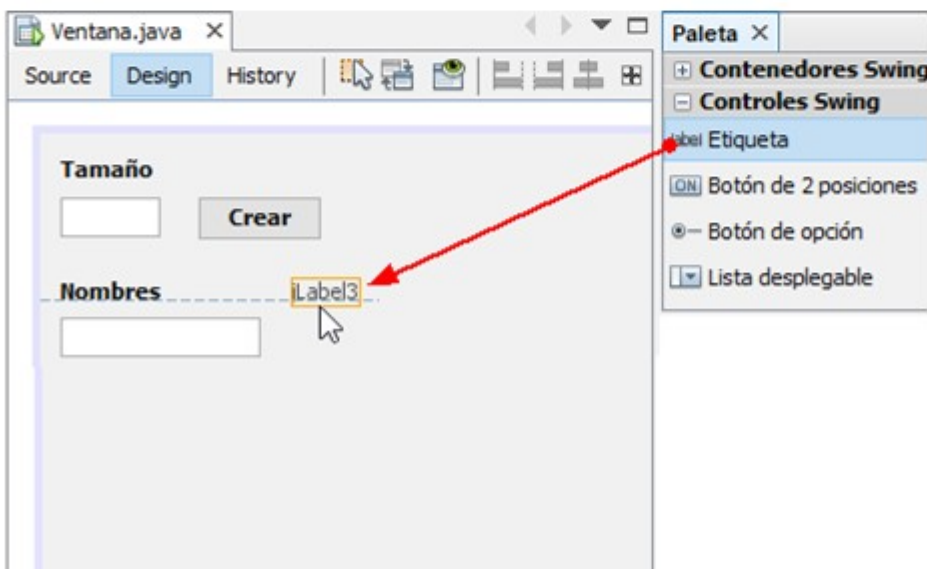
- ➡ Asigne **tf2** como nombre de instancia y haga click en el botón **Aceptar**.



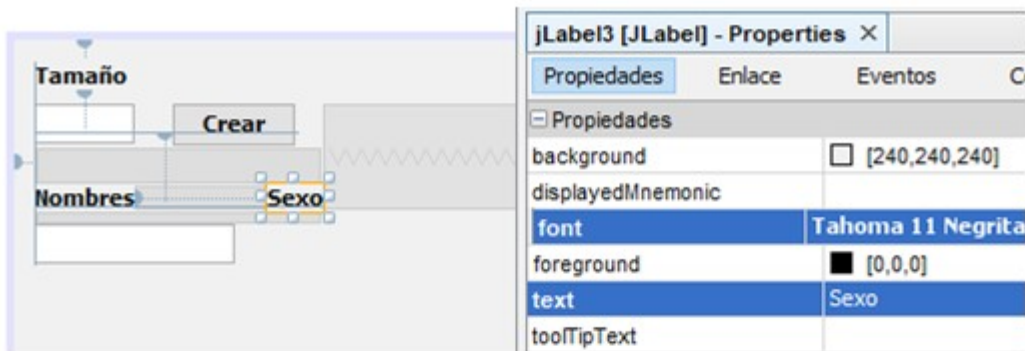
- ➡ En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo "**Gestor de distribución**", ponga 100 como valor para la propiedad **Tamaño horizontal**.



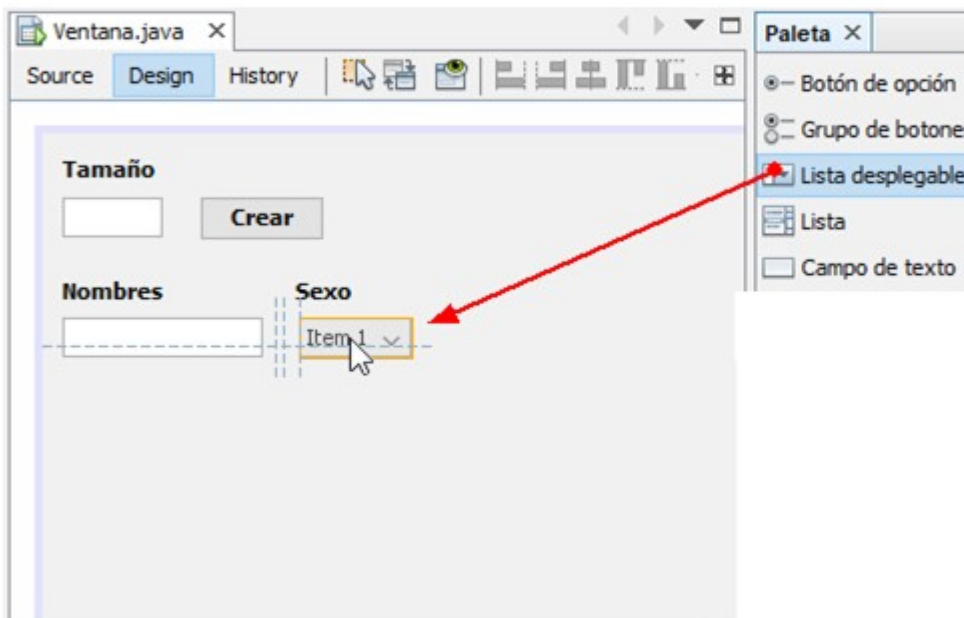
- ➡ Arrastre un **JLabel** hasta el punto señalado:



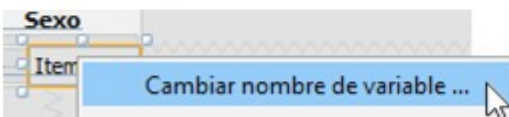
- ➡ En el inspector de propiedades, para la propiedad **text** escriba **Sexo** y en la propiedad **font** ponga el texto en negrilla.



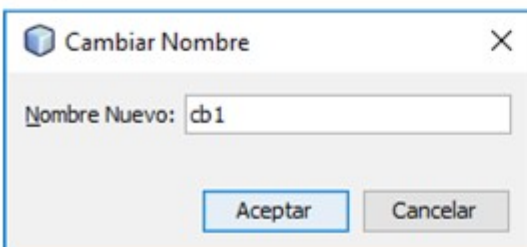
- ➡ Arrastre un **JComboBox** hasta la ventana según la posición indicada:



- ➡ Haga click derecho sobre el **JComboBox** y seleccione la opción "**cambiar nombre de variable ...**".



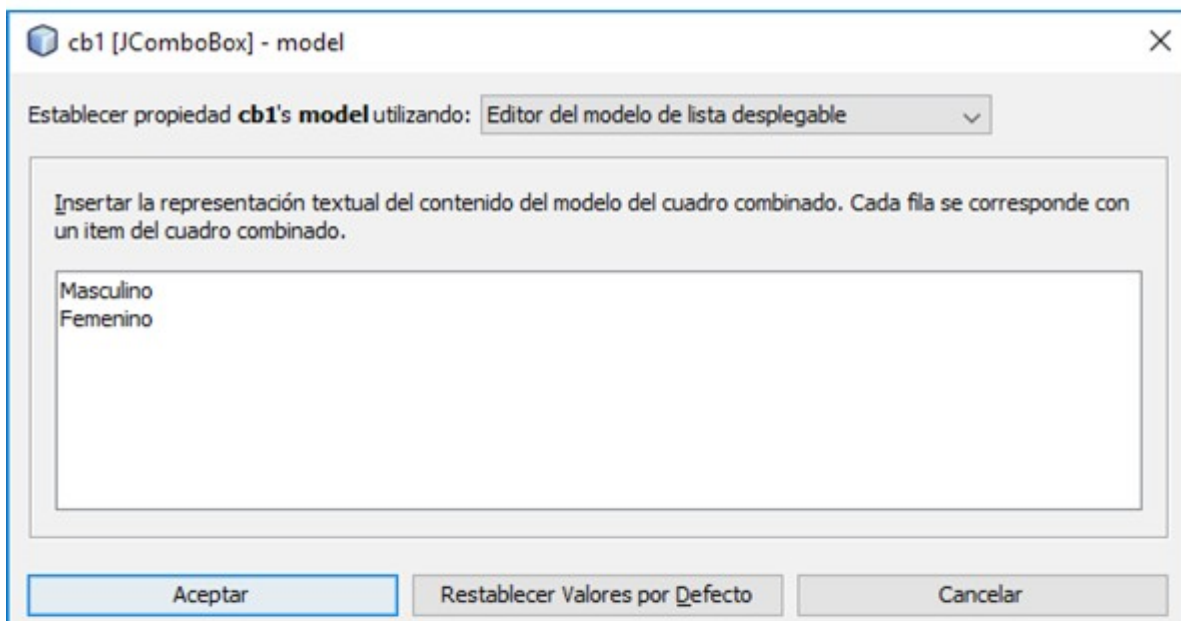
- ➡ Asigne **cb1** como nombre de instancia y haga click en el botón **Aceptar**.



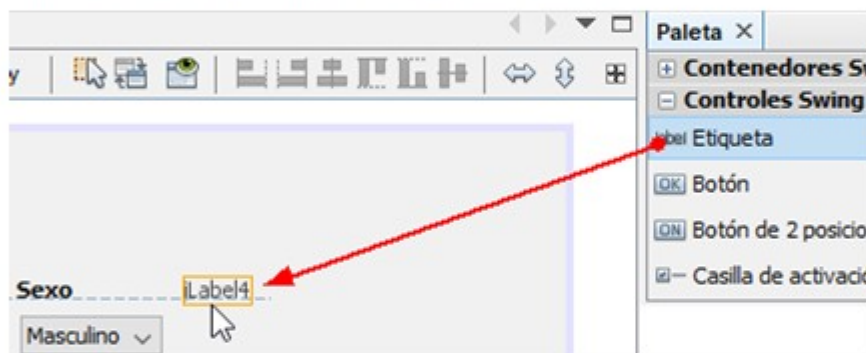
- ➡ En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



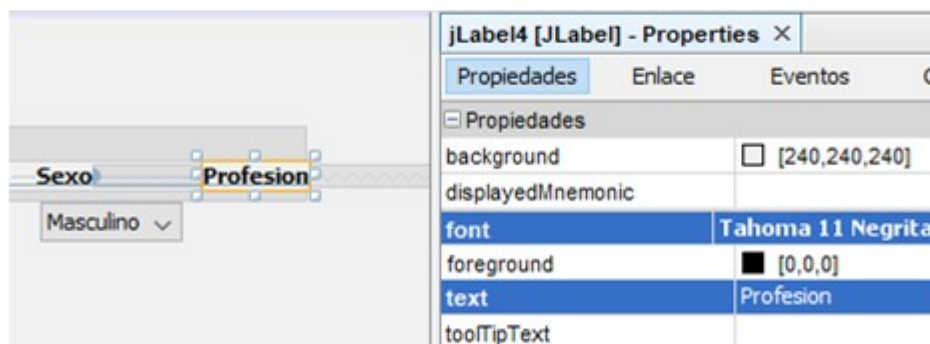
- Borre los textos predeterminados, escriba **Masculino** y **Femenino**, y haga click en el botón **Aceptar**.



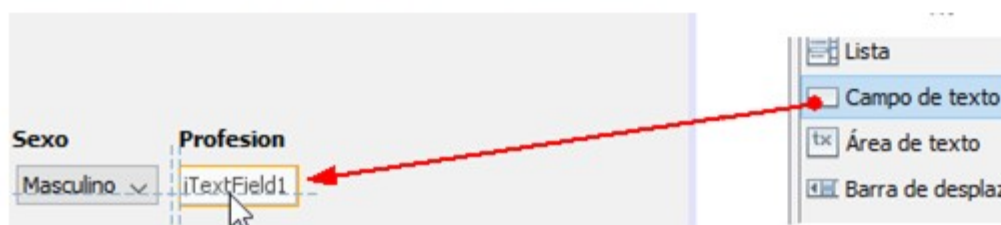
- Arrastre un **JLabel** hasta el punto indicado:



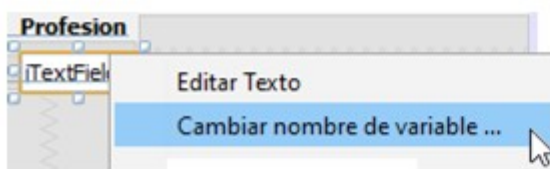
- En el inspector de propiedades, para la propiedad **text** escriba **Profesión** y en la propiedad **font** ponga el texto en negrilla.



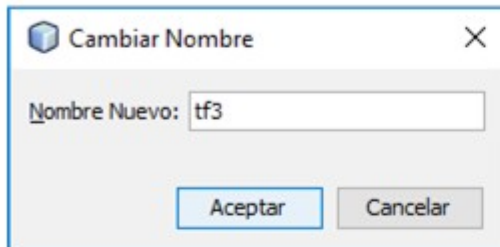
- Arrastre un **JTextField** hasta la posición señalada:



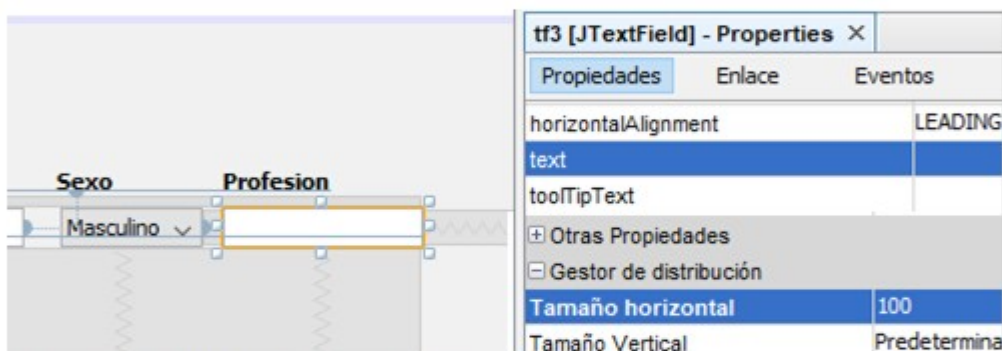
- ➔ Haga click derecho sobre el **JTextField** y seleccione la opción "**cambiar nombre de variable ...**".



- ➔ Asigne **tf3** como nombre de instancia y haga click en el botón **Aceptar**.



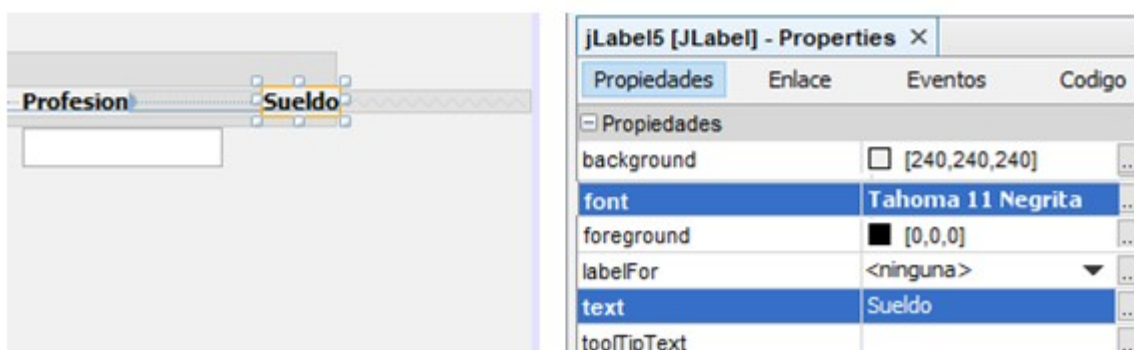
- ➔ En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo "**Gestor de distribución**", ponga 100 como valor para la propiedad Tamaño horizontal:



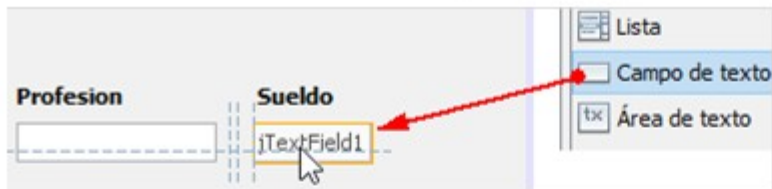
- ➔ Arrastre un **JLabel** hasta el punto indicado:



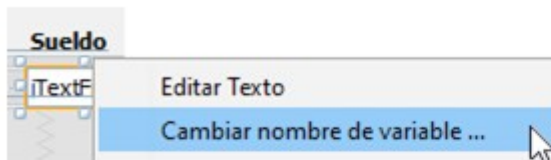
- ➔ En el inspector de propiedades, para la propiedad **text** escriba **Sueldo** y en la propiedad **font** ponga el texto en negrilla.



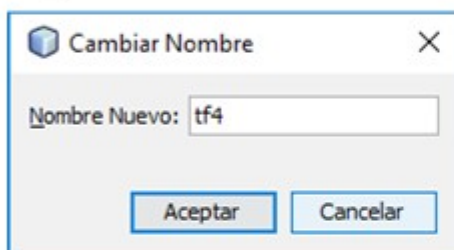
- ➡ Arrastre un **JTextField** hasta la posición mostrada:



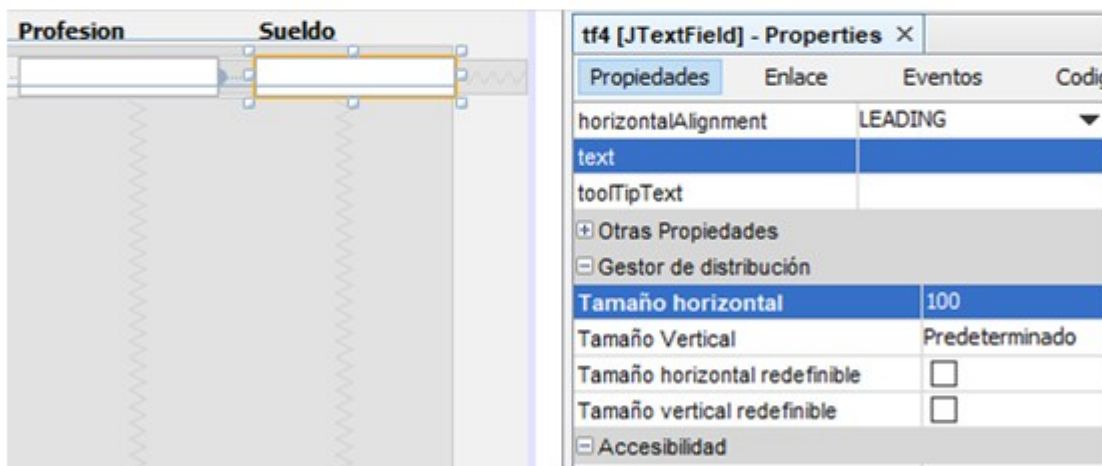
- ➡ Haga click derecho sobre el **JTextField** y seleccione la opción "**cambiar nombre de variable ...**".



- ➡ Asigne **tf4** como nombre de instancia y haga click en el botón **Aceptar**.



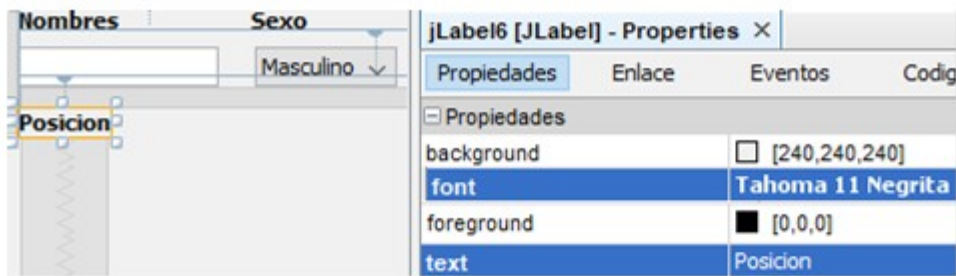
- ➡ En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo "**Gestor de distribución**", ponga 100 como valor para la propiedad Tamaño horizontal:



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



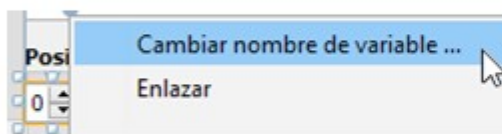
- ➡ En el inspector de propiedades, en la propiedad **text** escriba **Posición** y en la propiedad **font** ponga el texto en negrilla.



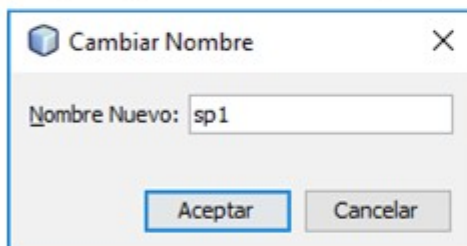
- ➡ Arrastre un **JSpinner** hasta la posición señalada:



- ➡ Haga click derecho sobre el **JSpinner** y seleccione la opción "**cambiar nombre de variable ...**".



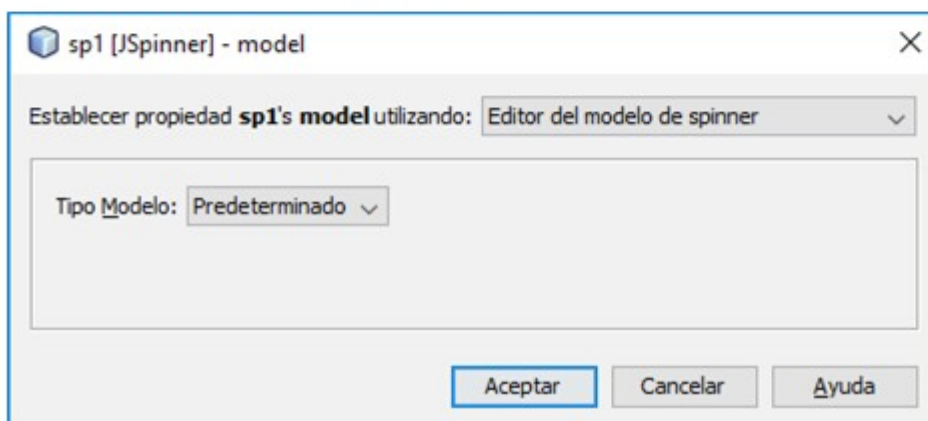
- ➡ Asigne **sp1** como nombre para el **JSpinner** y haga click en el botón **Aceptar**.



- ➡ En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



- ➡ Con ello vera una pantalla como esta:



- ➡ En **tipo de modelo** seleccione **Número**:

sp1 [JSpinner] - model

Establecer propiedad **sp1's model** utilizando: Editor del modelo de spinner

Tipo Modelo: **Número**

Propiedades de modelo

Numero de tipo: **Integer**

Valor inicial: **0**

☐ Mínimo: **0**

☐ Maximizo: **0**

Tamaño del paso: **1**

Aceptar Cancelar Ayuda

- ➡ En **Número de tipo** seleccione **Integer**, en **Valor inicial** ponga 0, marque la casilla **Mínimo** y asegúrese de que su valor sea 0; finalmente haga click en el botón **Aceptar**.

sp1 [JSpinner] - model

Establecer propiedad **sp1's model** utilizando: Editor del modelo de spinner

Tipo Modelo: **Número**

Propiedades de modelo

Numero de tipo: **Integer**

Valor inicial: **0**

☒ Mínimo: **0**

☐ Maximizo: **0**

Tamaño del paso: **1**

Aceptar Cancelar Ayuda

- ➡ En el inspector de propiedades en el nodo "**Gestor de distribución**", ponga 50 como valor para la propiedad **Tamaño horizontal**:

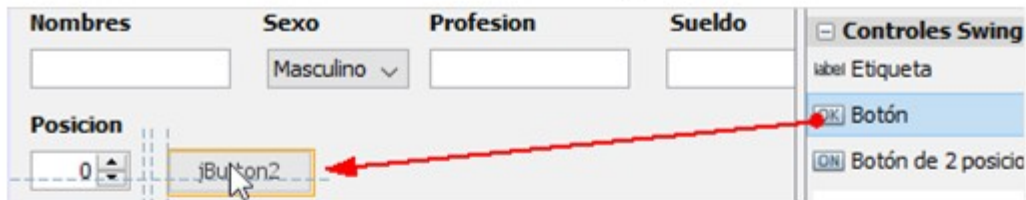
Nombres Sexo Profesion

Posicion

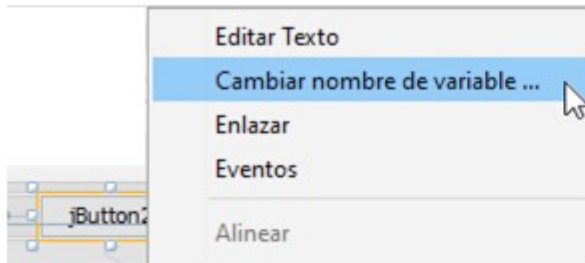
sp1 [JSpinner] - Properties

Propiedades	Enlace	Eventos	Cc
model			[SpinnerModel]
toolTipText			
+ Otras Propiedades			
- Gestor de distribución			
Tamaño horizontal			50
Tamaño Vertical			Predeterminado

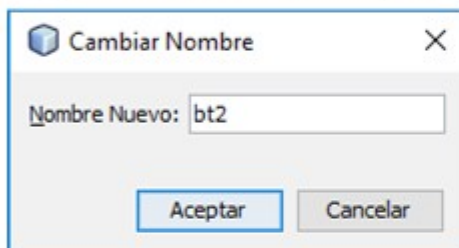
- ➡ Arrastre hasta la ventana un **JButton** en el punto señalado:



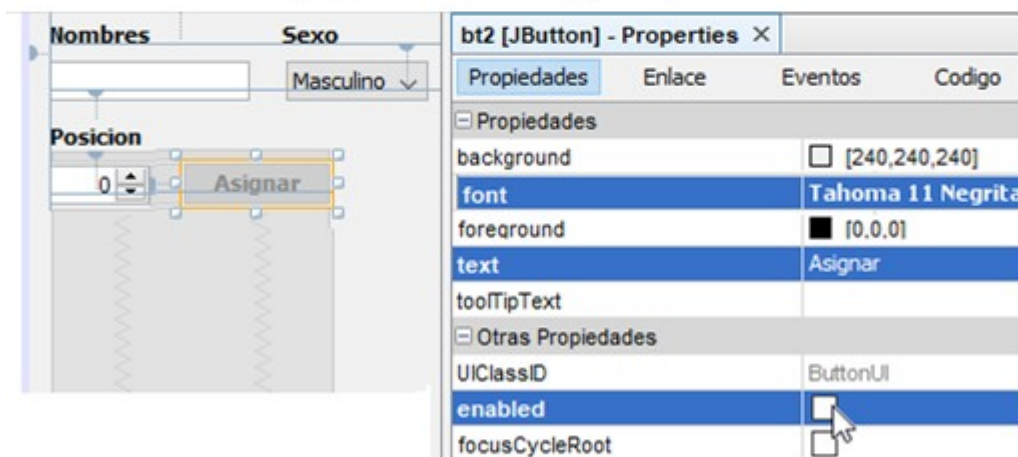
- ➡ Haga click derecho sobre el **JButton** y seleccione la opción "**cambiar nombre de variable ...**"



- ➡ Asigne **bt2** como nombre para el **JButton** y haga click en el botón **Aceptar**.



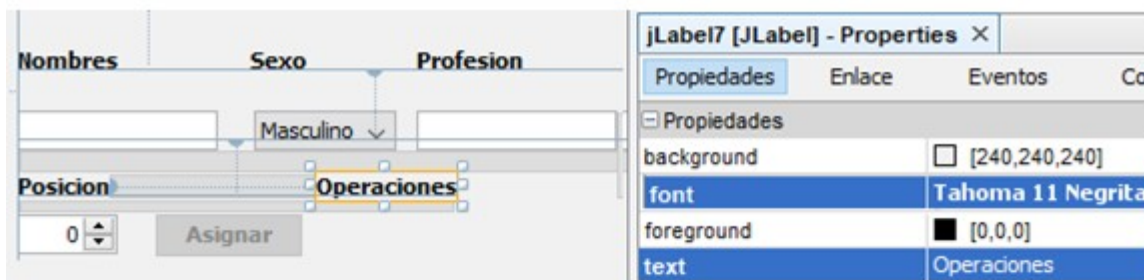
- ➡ En el inspector de propiedades, ponga **Asignar** en la propiedad **text**, en la propiedad **font** asigne texto en negrilla y en el nodo "**Otras propiedades**" desmarque la casilla a la derecha de la propiedad **enabled**, para que el botón inicialmente este deshabilitado.



- ➡ Arrastre un **JLabel** hasta la posición señalada:



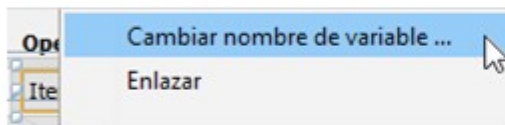
- En el inspector de propiedades, en la propiedad **text** escriba **Operaciones** y en la propiedad **font** ponga el texto en negrilla.



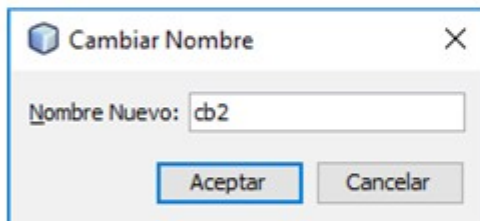
- Arrastre un **JComboBox** hasta la ventana según la posición indicada:



- Haga click derecho sobre el **JComboBox** y seleccione la opción "**cambiar nombre de variable ...**"



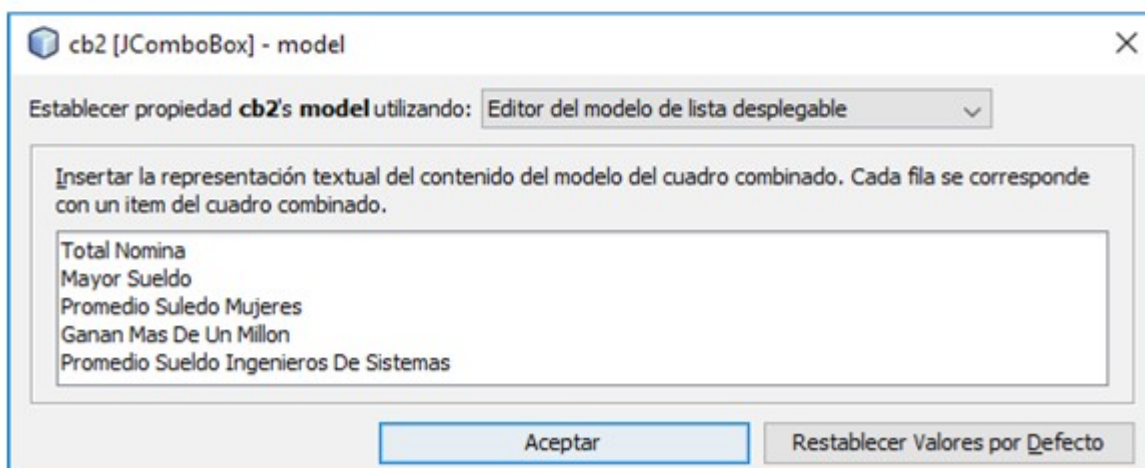
- Asigne **cb2** como nombre de instancia y haga click en el botón **Aceptar**.



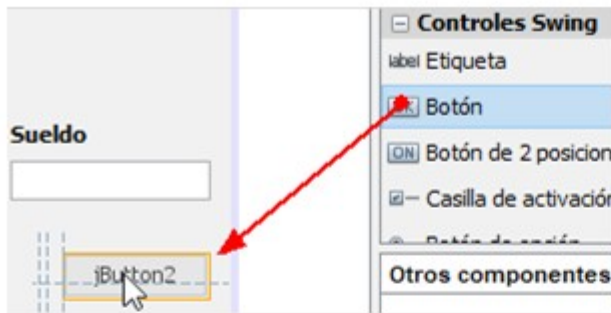
- En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



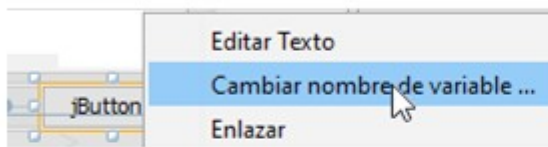
- Borre los textos predeterminados, escriba las opciones señaladas abajo y haga click en el botón **Aceptar**.



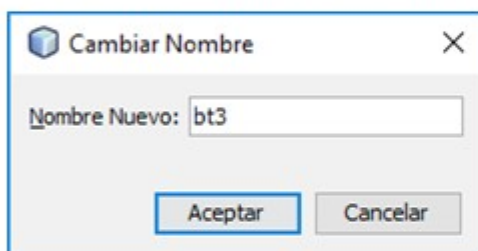
- ➡ Arrastre hasta la ventana un **JButton** en el punto señalado:



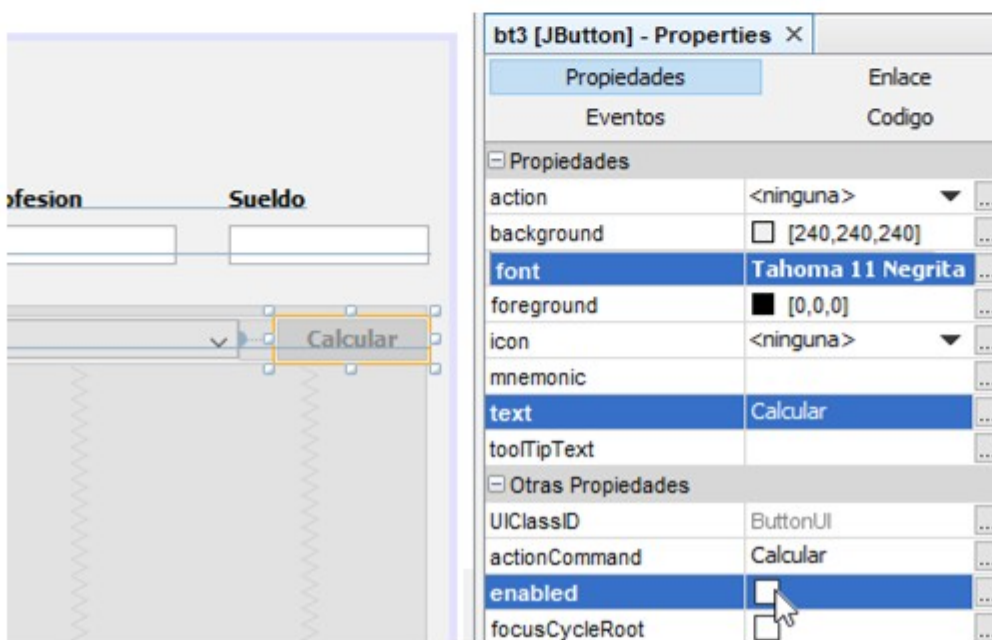
- ➡ Haga click derecho sobre el **JButton** y seleccione la opción "**cambiar nombre de variable ...**".



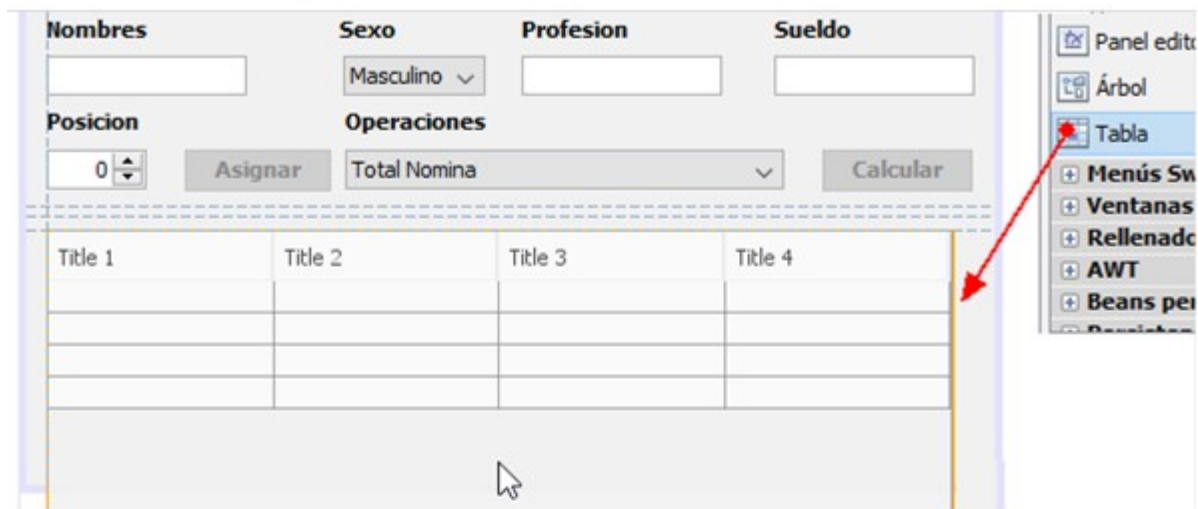
- ➡ Asigne **bt3** como nombre para el **JButton** y haga click en el botón **Aceptar**.



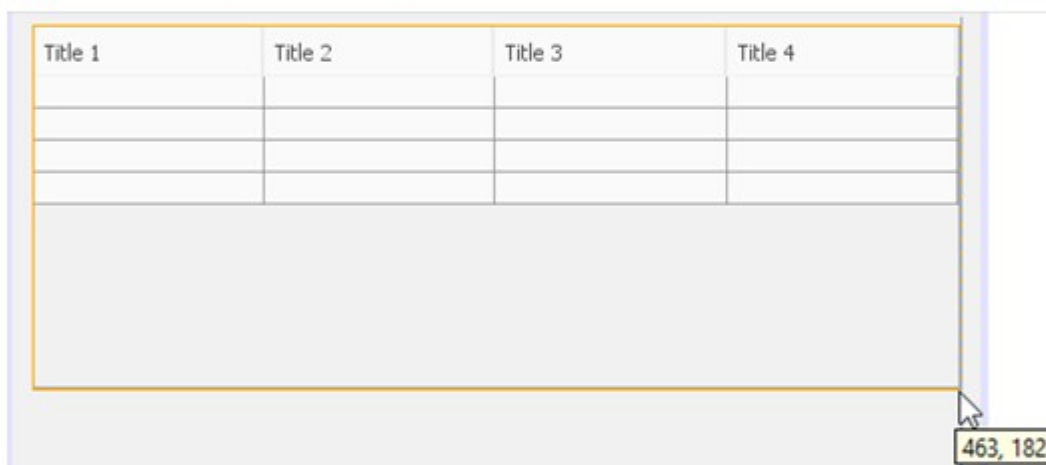
- ➡ En el inspector de propiedades, ponga **Calcular** en la propiedad **text**, en la propiedad **font** asigne texto en negrilla y en el nodo "**Otras propiedades**" desmarque la casilla a la derecha de la propiedad **enabled**, para que el botón inicialmente este deshabilitado.



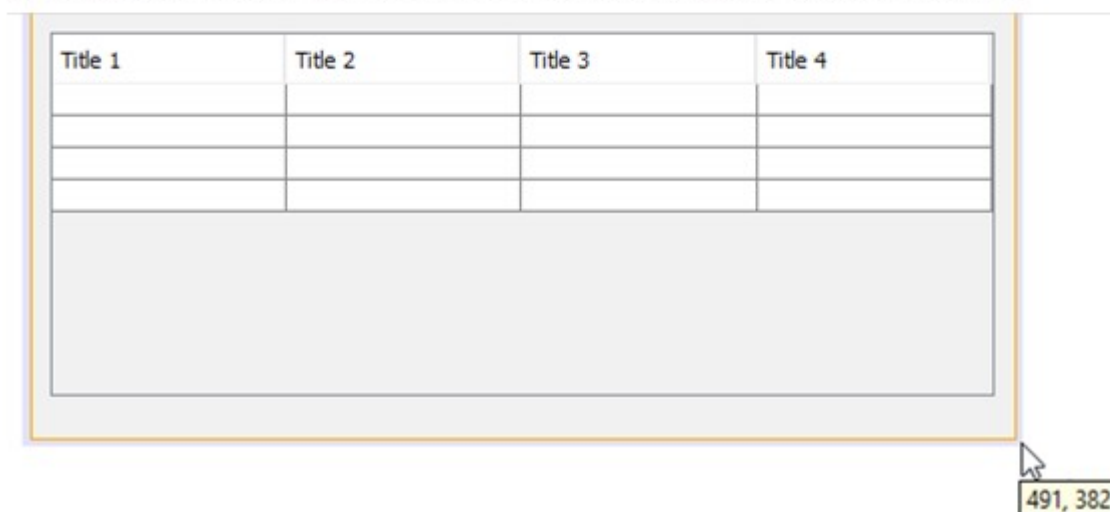
- ➡ Arrastre hasta el punto señalado un control **JTable**



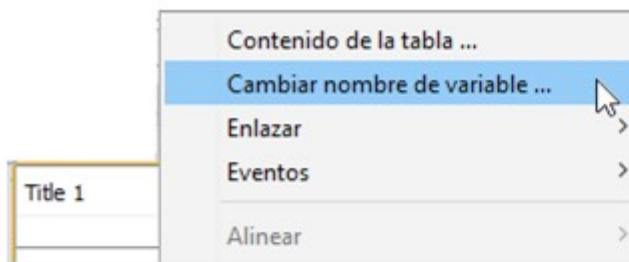
- ➡ Tome la tabla por el nodo inferior derecho y rediménsionela para que tenga el tamaño indicado en el cuadro amarillo de abajo (463,182).



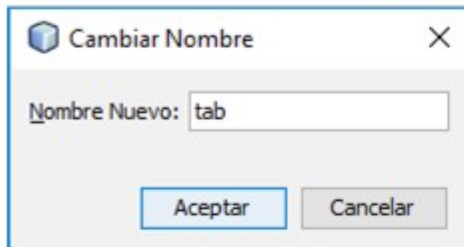
- ➡ De igual manera, tome el **JFrame** por el nodo inferior derecho y rediménsionela para que tenga el tamaño indicado en el cuadro amarillo de abajo (491,382).



- Haga click derecho sobre el **JTable** y seleccione la opción "**cambiar nombre de variable ...**".



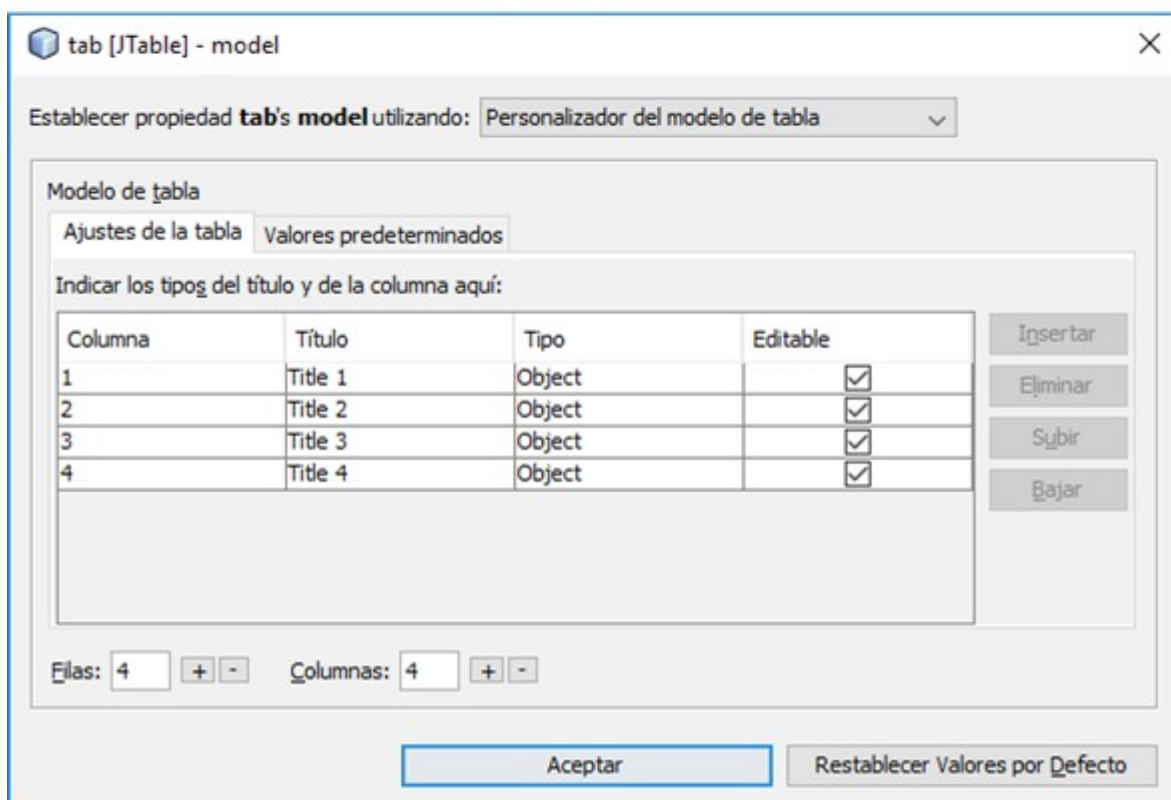
- Asigne **tab** como nombre para el **JTable** y haga click en el botón **Aceptar**.



- En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



- Con ello vera una ventana como la siguiente, en la que se configuran la cantidad de filas y columnas, los títulos y tipos de columnas, además de a cuales columnas se les puede modificar su contenido en tiempo de ejecución:



- ➡ Ahora escriba los títulos indicados en la imagen de abajo; para lo cual, por ejemplo, haga doble click sobre el texto **Title 1** y modifíquelo escribiendo **Nombres**, luego repita el proceso para los demás. En la columna tipo haga click sobre **Object** y escoja el tipo adecuado según la imagen de abajo. Luego desmarque las casillas **Editable** con lo cual el usuario no podrá cambiar el contenido de las columnas. En la entrada **Filas** ponga 0 y **Columnas** deje el valor de 4. Finalmente haga click en el botón **Aceptar**.

tab [JTable] - model

Establecer propiedad **tab's model** utilizando: Personalizador del modelo de tabla

Modelo de tabla

Ajustes de la tabla Valores predeterminados

Indicar los tipos del título y de la columna aquí:

Columna	Título	Tipo	Editable
1	Nombres	String	<input type="checkbox"/>
2	Sexo	String	<input type="checkbox"/>
3	Profesion	String	<input type="checkbox"/>
4	Sueldo	Float	<input type="checkbox"/>

Filas: 0 + - Columnas: 4 + -

Aceptar Restablecer Valores por Defecto

- ➡ La ventana debería tener un aspecto como el siguiente:

Tamaño

Crear

Nombres

Sexo Masculino

Profesion

Sueldo

Posicion 0

Asignar

Operaciones Total Nomina

Calcular

Nombres	Sexo	Profesion	Sueldo
---------	------	-----------	--------

g. Implementación de la ventana

- ➡ Vaya al código fuente de la ventana y antes de la línea de declaración de la clase, añada las siguientes importaciones (solo el código que está dentro del cuadro rojo):

```
import javax.swing.JOptionPane;
import javax.swing.SpinnerNumberModel;
import javax.swing.table.DefaultTableModel;

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

- ➡ Más abajo, debajo de la línea de declaración de la clase, añada las siguientes instancias como atributos de la clase de la ventana (solo el código que está dentro del cuadro rojo):

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

    private TEmpresa VE;
    private DefaultTableModel TabMod;
    private SpinnerNumberModel SpMod;
```

- ➡ Más abajo en el método constructor, debajo del llamado al método *initComponents()*; agregue las inicializaciones de las instancias declaradas anteriormente (solo el código que está dentro del cuadro rojo), además de añadir más abajo, la implementación para los métodos *Mostrar()*, *VerMensaje()*, *Limpiar()* y *LLenar()*:

```
public Ventana() {
    initComponents();

    VE=new TEmpresa();
    TabMod=(DefaultTableModel) tab.getModel();
    SpMod=(SpinnerNumberModel) spl.getModel();
}

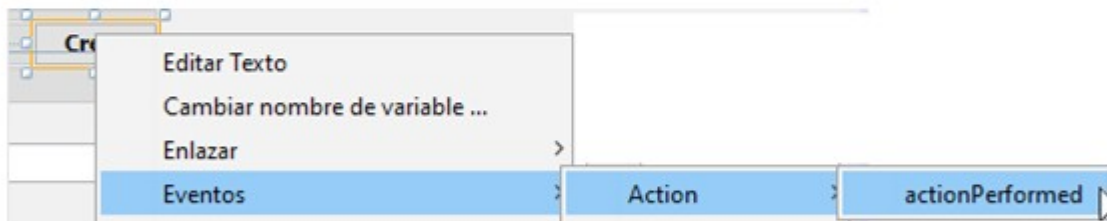
private void Mostrar() {
    int i;
    Trabajador Tra;
    for(i=0;i<VE.getTam();i++){
        Tra=VE.getVecEmp(i);
        TabMod.setValueAt(Tra.getNombres(),i,0);
        TabMod.setValueAt(Tra.getSexo()=="M"? "Masculino": "Femenino",i,1);
        TabMod.setValueAt(Tra.getProfesion(),i,2);
        TabMod.setValueAt(Tra.getSueldo(),i,3);
    }
}
```

```

private void VerMensaje(String Texto) {
    JOptionPane.showMessageDialog(null, Texto);
}
private void Limpiar() {
    tf2.setText("");
    tf3.setText("");
    tf4.setText("");
    tf2.grabFocus();
}
private void LLenar(Trabajador Tra) {
    Tra.setNombres(tf2.getText());
    Tra.setProfesion(tf3.getText());
    if(cb1.getSelectedIndex()==0) {
        Tra.setSexo('M');
    }
    else{
        Tra.setSexo('F');
    }
    Tra.setSueldo(Float.parseFloat(tf4.getText()));
}

```

- ➡ 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 {}, para lo cual, primero borre el comentario que dice: **//TODO add your handling code here:** quedando así:

```

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
}

```

- ➡ Ahora entre las llaves escriba la implementación del evento de la siguiente manera (código del cuadro rojo):

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  
    int N;  
    N=Integer.parseInt(tf1.getText());  
    VE.setTam(N);  
    TabMod.setRowCount(N);  
    SpMod.setMaximum(N-1);  
    bt2.setEnabled(N>0);  
    bt3.setEnabled(N>0);  
    VerMensaje("Vector Creado");  
    Limpiar();  
}
```

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



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

```
private void bt2ActionPerformed(java.awt.event.ActionEvent evt) {  
    int pos;  
    pos=(int) SpMod.getValue();  
    LLenar(VE.getVecEmp(pos));  
    if(pos+1<VE.getTam()) {  
        SpMod.setValue(pos+1);  
    }  
    Limpiar();  
    Mostrar();  
}
```

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

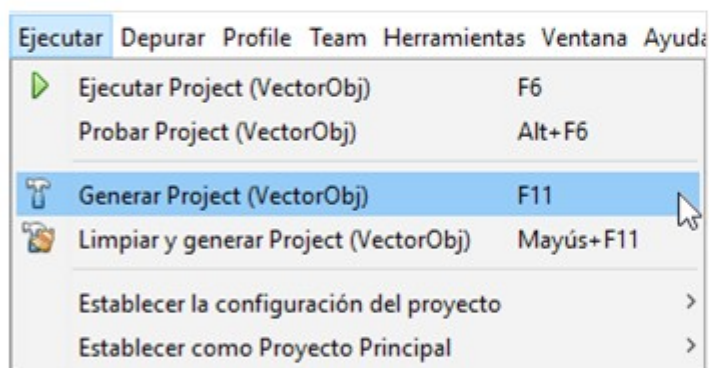


- ➡ El código para implementar el evento de este último botón es como sigue:

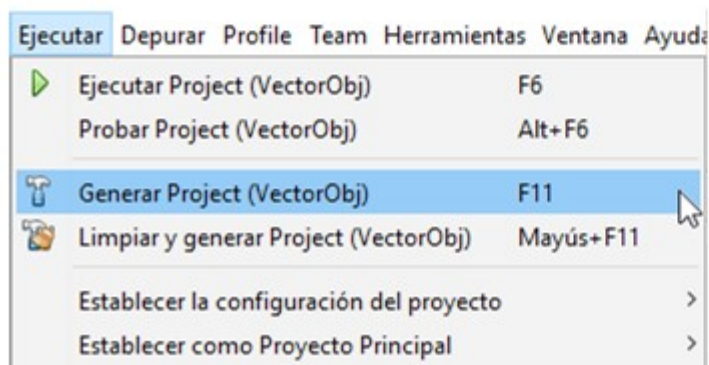
```
private void bt3ActionPerformed(java.awt.event.ActionEvent evt) {  
    String Res;  
    Res=cb2.getSelectedItem() + " = ";  
    switch(cb2.getSelectedIndex()) {  
        case 0:Res+=VE.TotNomina();break;  
        case 1:Res+=VE.MaxSueldo();break;  
        case 2:Res+=VE.PromSueMuj();break;  
        case 3:Res+=VE.CantSueMillon();break;  
        case 4:Res+=VE.PromSueIng();break;  
    }  
    VerMensaje(Res);  
}
```

h. Compilación y ejecución del programa.

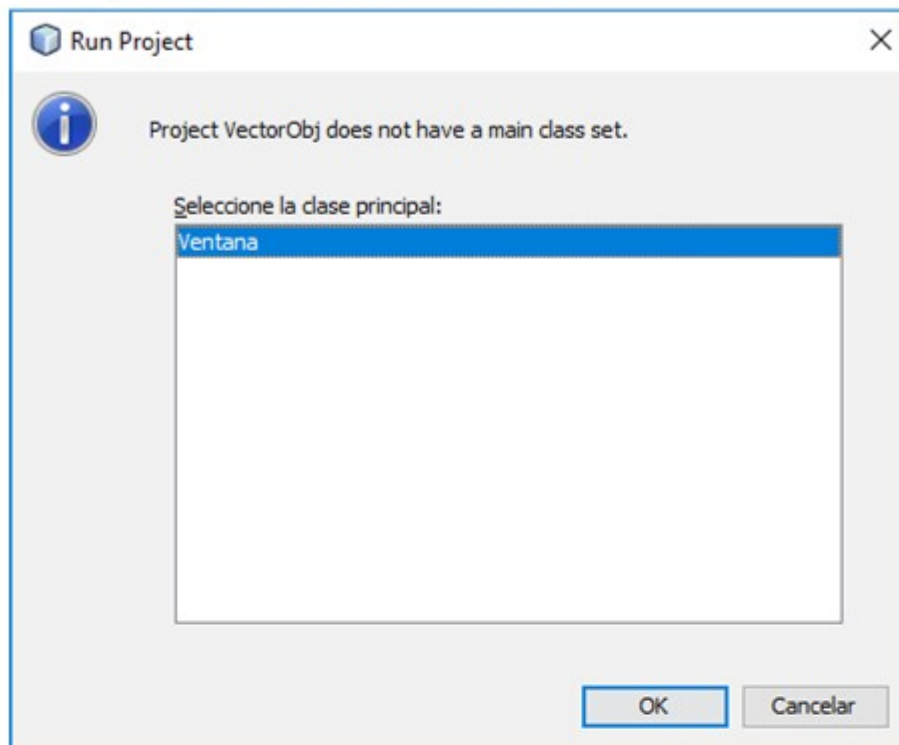
- ➡ Para compilar el programa vaya por la opción “Ejecutar” + “Generar Project” del menú principal o pulse la tecla F11.



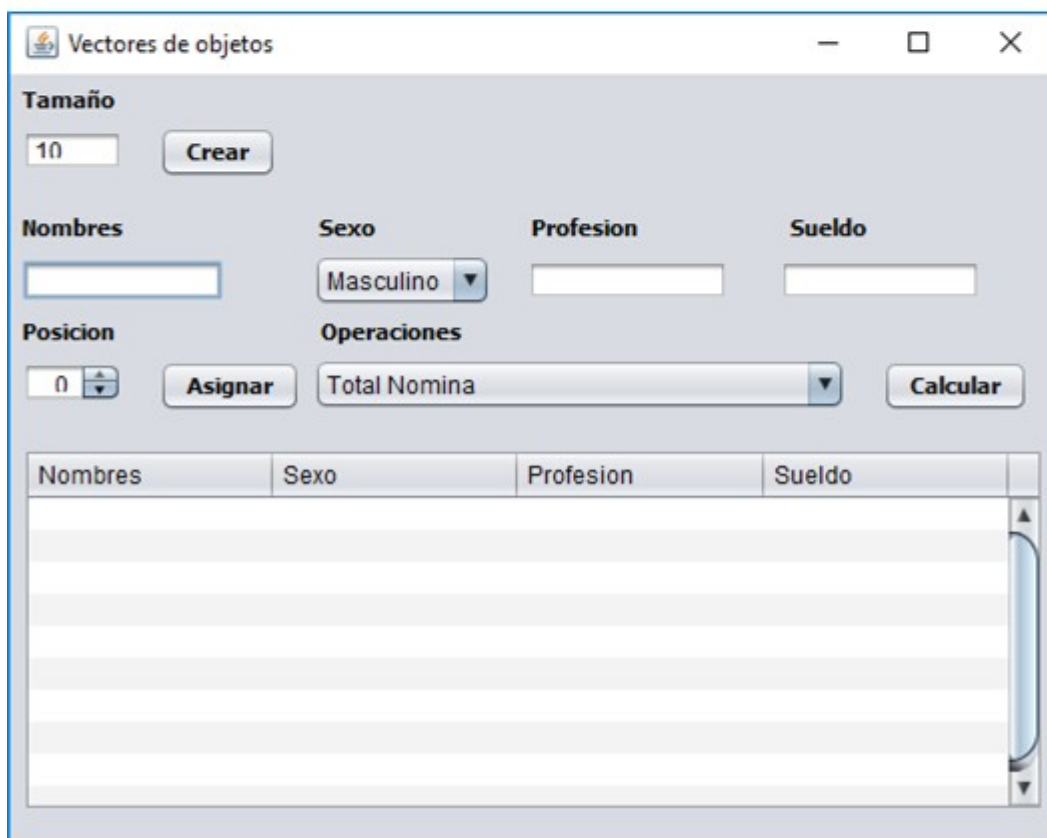
- ➡ Corrija los errores según los mensajes indicados por el compilador comparando con el código fuente anterior; luego puede correr el programa con la opción “Ejecutar” + “Ejecutar Project” del menú principal, o pulsando la tecla F6:



- 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**.



- La siguiente imagen ilustra la ejecución del programa:



Preguntas y ejercicios propuestos

1. En la implementación del constructor de la clase *Trabajador* observamos la siguiente asignación `Sexo='\0'`; investigue que significa esta forma de asignación para tipos **char**.
2. Consulte para que se utiliza y cuál es la interpretación del método `compareTo()` de la clase **String**. Modifique el método `PromSuelIng()` de la clase *TEmpresa*, para que la comparación con el texto "ingeniero de sistemas" no sea sensible a mayúsculas y minúsculas.
3. Modifique el programa anterior de modo se calcule y muestre el número de contadores públicos que ganan más que el promedio.
4. Modifique el programa ejemplo de modo que la clase *TEmpresa* tenga un método que permita aumentar el sueldo en 1.5% a todos los empleados; disponga además un método en la ventana para mostrar los datos solo de los empleados que tengan una profesión dada por el usuario.
5. En las dos líneas de código que se muestran abajo, porque es necesario hacer el casting con el tipo de modelo indicado entre el paréntesis (tanto para el modelo de la **JTable** como para el modelo del **JSpinner**)

```
TabMod=(DefaultTableModel) tab.getModel();  
SpMod=(SpinnerNumberModel) spl.getModel();
```

6. Por qué es necesario usar una instancia de **DefaultTableModel** para el modelo del **JTable**.
7. Por qué es necesario usar una instancia de **SpinnerNumberModel** para el modelo del **JSpinner**.
8. Explique los parámetros requeridos por el método `setValueAt` de la clase **DefaultTableModel**.
9. En el código del método **Mostrar**, explique qué hace la siguiente línea de código:

```
Tra.getSexo()=='M'?"Masculino":"Femenino"
```

10. Cuál es el tipo de datos del parámetro que se le pasa al método `setMaximum` de la clase **SpinnerNumberModel**; en que consiste ese tipo de datos.
11. Explique qué hace la siguiente línea de código en el evento click del botón **Crear**

```
bt2.setEnabled(N>0);
```


12. Explique para que se usa el método **grabFocus** de un control.

13. Explique por qué es necesario hacer el casting al tipo **int** en la siguiente línea de código del evento click del botón **Asignar**

```
pos=(int) SpMod.getValue();
```

14. Explique qué diferencia hay entre el método **getSelectedItem** y el método **getSelectedIndex** de un **JComboBox**.

15. Diseñe en UML e implemente en *Java* las clases requeridas, con la correspondiente aplicación de ventana, para almacenar los datos de un conjunto de personas correspondientes a apellidos, nombres, identificación, edad, sexo y estrato social; posibilitando obtener los siguientes cálculos:

- Promedio de hombres de menos de 18 años y de estrato 2.
- Porcentaje de mujeres mayores de edad.
- Promedio de personas de los estratos 2 o 4 cuya edad no está entre 15 y 20 años.
- Total de personas entre 25 y 35 años de estratos superiores al 2.

16. Diseñe en UML e implemente en *Java* las clases requeridas, con la correspondiente aplicación de ventana, para almacenar el volumen de lluvias registradas en las regiones norte, sur, este y oeste del país a lo largo de los doce meses del año. Con esta información se debe realizar las siguientes operaciones:

- Promedio de lluvias al año de una región dada por el usuario.
- Promedio de lluvias durante el primer semestre del año.
- Cantidad total de lluvia en un mes dado por el usuario.
- Región que tiene mayor cantidad de lluvia en el primer trimestre del año.