

UNIVERSIDAD DE CORDOBA

FACULTAD DE INGENIERIAS

PROGRAMA INGENIERIA DE SISTEMAS

CURSO: Programación II

TEMA: Ordenamiento de arreglos por el método *QuickSort*.

DESCRIPCION:

En el siguiente documento se hace una descripción del funcionamiento de uno de los métodos de ordenamiento más eficiente como lo es el método de *QuickSort*, partiendo de una explicación textual de la manera en que este método organiza los elementos de un arreglo y como utiliza para ello el concepto de recursividad y el uso de pivotes, para luego pasar a presentar en forma grafica como paso a paso este método ordena un arreglo de números enteros, ilustrando además los mecanismos de elección del pivote y su valor respectivo, incluyendo los llamados recursivos del algoritmo a lo largo del proceso seguido para ordenar el vector unidimensional.

Seguidamente se presenta un problema de ordenamiento de arreglos para ser resuelto con el método *QuickSort*, haciéndose la solución con un diagrama de clases en UML con las relaciones adecuadas que posteriormente se implementa con un proyecto de consola usando el IDE *NetBeans* Microsoft Windows. Finalmente se desarrolla el código del programa principal indicando los pasos para el proceso de compilación y añadiéndose una captura de ventana de la ejecución del programa.

OBJETIVO: Diseñar en UML e implementar en el lenguaje Java una clase para ordenar de forma ascendente un arreglo de tamaño dinámico que contiene cadenas de caracteres usando el método de ordenamiento *QuickSort*, realizando además el correspondiente programa de consola con el IDE *NetBeans* en Microsoft Windows.

PALABRAS CLAVES: Métodos de ordenamiento de vectores en Java, ordenamiento por el método *QuickSort*, recursividad en Java, arreglos de objetos en Java, aplicaciones de consola en Java con *NetBeans*.

1. Método de ordenamiento QuickSort

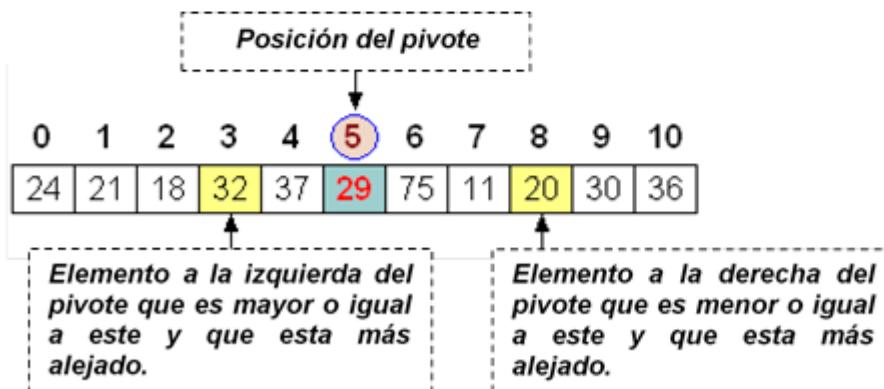
El método de ordenamiento *QuickSort* (rápido en inglés) fue desarrollado por el británico C.A. Hoare doctor en computación y es considerado uno de los métodos de ordenamiento más rápidos y del cual podemos encontrar varias implementaciones al respecto. En cuanto a su funcionamiento este método de ordenamiento se basa en la idea de "divide y vencerás" pues considera que ordenar un arreglo es más simple si lo divido en varios "sub arreglos" de tamaño menor y ordeno cada uno de estos, razón por la cual también se le conoce como ordenamiento por partición. Ahora bien, estos "sub arreglos" o "particiones" del arreglo se realizan en torno a un elemento de este, al cual llamamos pivote y en general puede ser cualquier elemento del arreglo, incluso pudiendo escogerse al azar; pero no obstante a ello, de acuerdo a la elección del pivote y al conjunto de valores particulares contenidos en el arreglo puede variar la eficiencia del método. Aun así muchas de las implementaciones que encontramos para el método *QuickSort* usan como pivote al primer elemento del arreglo o aquel que se encuentra en la mitad del mismo.

Independientemente del modo de elección del pivote, para el caso de un ordenamiento ascendente, el algoritmo utiliza al pivote para ubicar a la izquierda todos los elementos del arreglo que son menores al pivote; pasando entonces a la derecha del pivote todos aquellos elementos que son mayores que este. Una vez hecho esto el proceso se repite recursivamente para un "sub arreglo" o parte del este comprendida por los elementos ubicados a la izquierda del pivote; luego se hace lo mismo recursivamente para la otra parte del arreglo que comprende los elementos ubicados a la derecha del pivote. En ambos casos se repite el mismo mecanismo de elección del pivote inicial, es decir se elige un nuevo pivote para el lado izquierdo y otro para el lado derecho; en todo caso el proceso continua repitiéndose recursivamente hasta que los "sub arreglos" tenga un tamaño de uno, punto en el que el arreglo ya estará ordenado.

Como puede deducirse, en la primera llamada del método el pivote se elije entre los elementos del arreglo; es decir, entre los elementos desde la posición 0 hasta la N-1 (siendo N el tamaño del vector). Si por ejemplo para un ordenamiento ascendente el pivote inicial es el que se encuentra en la posición "P", entonces se procede a mover a la izquierda del elemento P todos aquellos elementos menores que el pivote lo cual equivale a ubicarlos en posiciones $i < P$. De manera análoga se mueven a la derecha del elemento "P" todos aquellos elementos mayores que el pivote; es decir, ubicando a dichos elementos

en posiciones $i > P$. El llamado recursivo del método consiste entonces en elegir otro pivote pero considerando una parte del vector comprendida a la izquierda de "P", la cual estará comprendida por los elementos ubicados en el arreglo en posiciones como mínimo 0 y como máximo $P-1$; por lo que consecuentemente el valor de este nuevo pivote será menor que "P". A esta parte del vector se le aplica el mismo proceso descrito anteriormente; es decir, moviendo a la izquierda del pivote los valores menores que este y a la derecha los mayores que el pivote. Igualmente se elige otro pivote tomando en cuenta solo los elementos ubicados a la derecha de "P", el cual tendrá como posición un valor mayor que "P" que como mínimo será $P+1$ y como máximo $N-1$; efectuándose entonces para este segmento el mismo proceso indicado para el segmento de la derecha del pivote.

Siendo más precisos en la forma en que se escogen los valores a mover a la izquierda y a la derecha del pivote, vale la pena indicar que en realidad en el segmento del arreglo ubicado a la izquierda del pivote se busca un elemento que sea mayor o igual a este y se intercambia con otro elemento que es menor o igual al pivote pero que se busca en el segmento ubicado a la derecha del pivote. Para ser más eficiente el algoritmo, el elemento mayor o igual que el pivote ubicado a su izquierda se localiza de izquierda a derecha; es decir, será el elemento mayor o igual que el pivote ubicado más a la izquierda de este. Análogamente el elemento menor o igual que el pivote ubicado a su derecha se localiza de derecha a izquierda y en consecuencia será el elemento menor o igual que el pivote ubicado más a su derecha. En otras palabras se intercambia el elemento mayor o igual al pivote localizado más lejos de este a su izquierda, con el elemento menor o igual al pivote ubicado más lejos de este a su derecha. Para evidenciar un poco más la idea anterior considere la siguiente imagen:



En este caso se intercambia los elementos de los puestos 3 y 8; es decir, el 32 con el 20, aun cuando después del 32 existe otro mayor (37) que el pivote (29) y que antes del 20 existe otro menor (11) que el pivote.

2. Representación del método de QuickSort.

Como ejemplo consideraremos el siguiente vector de diez caracteres:

0	1	2	3	4	5	6	7	8	9
'F'	'B'	'K'	'J'	'L'	'D'	'A'	'E'	'G'	'C'

Vector desde izquierda=0 hasta derecha=9

Con izquierda=0 y derecha=9 el centro es: $(0 + 9)/2 = 4$ y el pivote es: Vec[4] = 'L'

0	1	2	3	4	5	6	7	8	9
'F'	'B'	'K'	'J'	'L'	'D'	'A'	'E'	'G'	'C'

En la siguiente imagen el elemento mayor o igual que el pivote ('L') más lejano a este desde la izquierda (0) es $\text{Vec}[4] = 'L'$ y el elemento menor o igual que el pivote ('L') más lejano a este desde la derecha (9) es $\text{Vec}[9] = 'C'$

0	1	2	3	4	5	6	7	8	9
'F'	'B'	'K'	'J'	'L'	'D'	'A'	'E'	'G'	'C'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[4] > \text{Vec}[9]$? , es ' L '>' C '? si, intercambiamos estos elementos quedando el arreglo así:

0	1	2	3	4	5	6	7	8	9
'F'	'B'	'K'	'J'	'C'	'D'	'A'	'E'	'G'	'L'

En la imagen de abajo el elemento mayor o igual que el pivote ('L') más lejano a este desde la izquierda (5) es $\text{Vec}[9] = 'L'$ y el elemento menor o igual que el pivote ('L') más lejano a este desde la derecha (8) es $\text{Vec}[8] = 'G'$

0	1	2	3	4	5	6	7	8	9
'F'	'B'	'K'	'J'	'C'	'D'	'A'	'E'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[9] > \text{Vec}[8]?$, es ' L '>' G '? No.

Llamada recursiva para la porción del vector desde izquierda=0 hasta derecha=8

Con izquierda=0 y derecha=8 el centro es: $(0 + 8)/2 = 4$ y el pivote es: Vec[4] = 'C'

0	1	2	3	4	5	6	7	8	9
'F'	'B'	'K'	'J'	'C'	'D'	'A'	'E'	'G'	'L'

La siguiente imagen muestra que el elemento mayor o igual que el pivote ('C') más lejano a este desde la izquierda (0) es $\text{Vec}[0] = 'F'$ y el elemento menor o igual que el pivote ('C') más lejano a este desde la derecha (8) es $\text{Vec}[6] = 'A'$

0	1	2	3	4	5	6	7	8	9
'F'	'B'	'K'	'J'	'C'	'D'	'A'	'E'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[0] > \text{Vec}[6]?$, es ' $F' > 'A$ '? si, intercambiamos estos elementos quedando el arreglo así:

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'K'	'J'	'C'	'D'	'F'	'E'	'G'	'L'

En la imagen presentada abajo el elemento mayor o igual que el pivote ('C') más lejano a este desde la izquierda (1) es $\text{Vec}[2] = 'K'$ y el elemento menor o igual que el pivote ('C') más lejano a este desde la derecha (5) es $\text{Vec}[4] = 'C'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'K'	'J'	'C'	'D'	'F'	'E'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[2] > \text{Vec}[4]?$, es ' $K' > 'C$ '? si, intercambiamos estos elementos quedando el arreglo así:

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'J'	'K'	'D'	'F'	'E'	'G'	'L'

De acuerdo a la imagen siguiente el elemento mayor o igual que el pivote ('C') más lejano a este desde la izquierda (3) es $\text{Vec}[3] = 'J'$ y el elemento menor o igual que el pivote ('C') más lejano a este desde la derecha (3) es $\text{Vec}[2] = 'C'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'J'	'K'	'D'	'F'	'E'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[3] > \text{Vec}[2]$? , es ' J '>' C '? No

Llamada recursiva para la porción del vector desde izquierda=0 hasta derecha=2

Con izquierda=0 y derecha=2 el centro es: $(0 + 2)/2 = 1$ y el pivote es: $\text{Vec}[1] = 'B'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'J'	'K'	'D'	'F'	'E'	'G'	'L'

↑

Como se aprecia en la imagen de abajo el elemento mayor o igual que el pivote ('B') más lejano a este desde la izquierda (0) es $\text{Vec}[1]='B'$ y el elemento menor o igual que el pivote ('B') más lejano a este desde la derecha (2) es $\text{Vec}[1]='B'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'J'	'K'	'D'	'F'	'E'	'G'	'L'

↑

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[1] > \text{Vec}[1]$? , es ' B '>' B '? No

Llamada recursiva para la porción del vector desde izquierda=3 hasta derecha=8

Con izquierda=3 y derecha=8 el centro es: $(3 + 8)/2 = 5$ y el pivote es: $\text{Vec}[5] = 'D'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'J'	'K'	'D'	'F'	'E'	'G'	'L'

↑

En la siguiente imagen el elemento mayor o igual que el pivote ('D') más lejano a este desde la izquierda (3) es $\text{Vec}[3]='J'$ y el elemento menor o igual que el pivote ('D') más lejano a este desde la derecha (8) es $\text{Vec}[5]='D'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'J'	'K'	'D'	'F'	'E'	'G'	'L'

↑ ↑

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[3] > \text{Vec}[5]$? , es ' J '>' D '? si, intercambiamos estos elementos quedando el arreglo así:

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'K'	'J'	'F'	'E'	'G'	'L'

En la imagen de abajo el elemento mayor o igual que el pivote ('D') más lejano a este desde la izquierda (4) es $\text{Vec}[4] = \text{'K'}$ y el elemento menor o igual que el pivote ('D') más lejano a este desde la derecha (4) es $\text{Vec}[3] = \text{'D'}$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'K'	'J'	'F'	'E'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[4] > \text{Vec}[3]$? es $\text{'K'} > \text{'D'}$? No

Llamada recursiva para la porción del vector desde izquierda=4 hasta derecha=8

Con izquierda=4 y derecha=8 el centro es: $(4 + 8)/2 = 6$ y el pivote es: $\text{Vec}[6] = \text{'F'}$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'K'	'J'	'F'	'E'	'G'	'L'

La siguiente imagen muestra que el elemento mayor o igual que el pivote ('F') más lejano a este desde la izquierda (4) es $\text{Vec}[4] = \text{'K'}$ y el elemento menor o igual que el pivote ('F') más lejano a este desde la derecha (8) es $\text{Vec}[7] = \text{'E'}$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'K'	'J'	'F'	'E'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[4] > \text{Vec}[7]$? es $\text{'K'} > \text{'E'}$? si, intercambiamos estos elementos quedando el arreglo así:

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'J'	'F'	'K'	'G'	'L'

En la imagen presentada abajo el elemento mayor o igual que el pivote ('F') más lejano a este desde la izquierda (5) es $\text{Vec}[5] = \text{'J'}$ y el elemento menor o igual que el pivote ('F') más lejano a este desde la derecha (6) es $\text{Vec}[6] = \text{'F'}$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'J'	'F'	'K'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[5] > \text{Vec}[6]$? , es ' J '>' F '? si, intercambiamos estos elementos quedando el arreglo así:

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'K'	'G'	'L'

Llamada recursiva para la porción del vector desde izquierda=4 hasta derecha=5

Con izquierda=4 y derecha=5 el centro es: $(4 + 5)/2 = 4$ y el pivote es: $\text{Vec}[4] = 'E'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'K'	'G'	'L'

De acuerdo a la imagen siguiente el elemento mayor o igual que el pivote ('E') más lejano a este desde la izquierda (4) es $\text{Vec}[4]='E'$ y el elemento menor o igual que el pivote ('E') más lejano a este desde la derecha (5) es $\text{Vec}[4]='E'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'K'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[4] > \text{Vec}[4]$? , es ' E '>' E '? No

Llamada recursiva para la porción del vector desde izquierda=6 hasta derecha=8

Con izquierda=6 y derecha=8 el centro es: $(6 + 8)/2 = 7$ y el pivote es: $\text{Vec}[7] = 'K'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'K'	'G'	'L'

Como se aprecia en la imagen de abajo el elemento mayor o igual que el pivote ('K') más lejano a este desde la izquierda (6) es $\text{Vec}[7]='K'$ y el elemento menor o igual que el pivote ('K') más lejano a este desde la derecha (8) es $\text{Vec}[8]='G'$

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'K'	'G'	'L'

Comparamos las posiciones de los elementos mayor y menor: es $\text{Vec}[7] > \text{Vec}[8]$? , es ' K '>' G '? si, intercambiamos estos elementos quedando el arreglo así:

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'G'	'K'	'L'

Llamada recursiva para la porción del vector desde izquierda=6 hasta derecha=7

Con izquierda=6 y derecha=7 el centro es: $(6 + 7)/2 = 6$ y el pivote es: Vec[6] = 'J'

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'G'	'K'	'L'



En la siguiente imagen el elemento mayor o igual que el pivote ('J') más lejano a este desde la izquierda (6) es Vec[6]='J' y el elemento menor o igual que el pivote ('J') más lejano a este desde la derecha (7) es Vec[7]='G'

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'J'	'G'	'K'	'L'



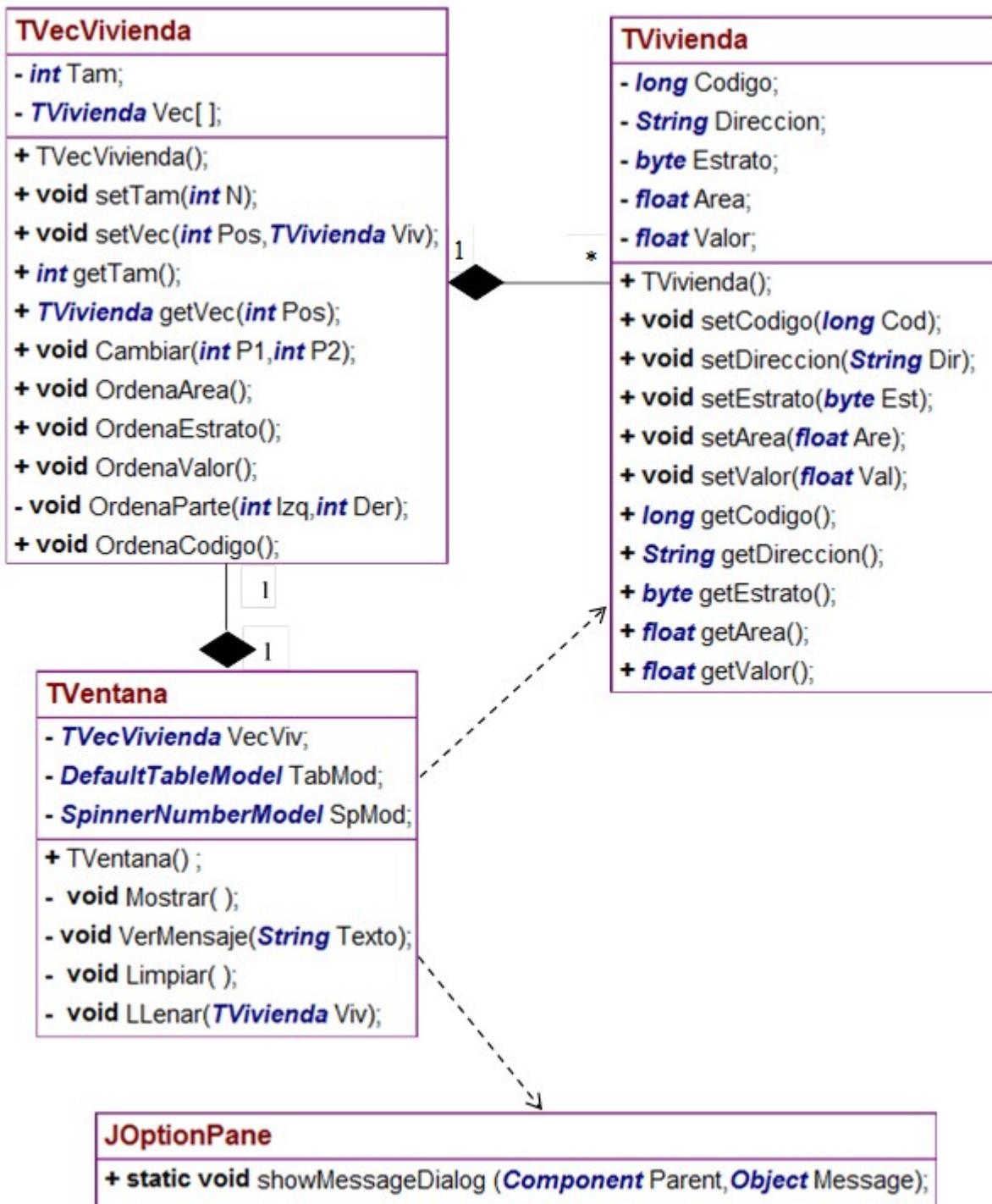
Comparamos las posiciones de los elementos mayor y menor: es Vec[6] > Vec[7]?, es 'J' > 'G'? si, intercambiamos estos elementos quedando el arreglo ya ordenado así:

0	1	2	3	4	5	6	7	8	9
'A'	'B'	'C'	'D'	'E'	'F'	'G'	'J'	'K'	'L'

3. Presentación ejemplo ordenamiento QuickSort

Diseñar en UML e implementar en Java las clases necesarias para un arreglo de objetos que contiene un conjunto de viviendas descritas por su código, dirección, estrato, área y valor de avalúo. El arreglo con los datos de cada vivienda debe llenarse desde el programa principal según la información ingresada por el usuario, además de disponer un menú de selección para cada una de las operaciones implementadas. Se deben ordenar las viviendas así: Ascendentemente por código usando QuickSort, descendenteamente por área con intercambio, descendenteamente por estrato con inserción y descendenteamente por valor por el método de Shell.

4. Diagrama UML de clases.

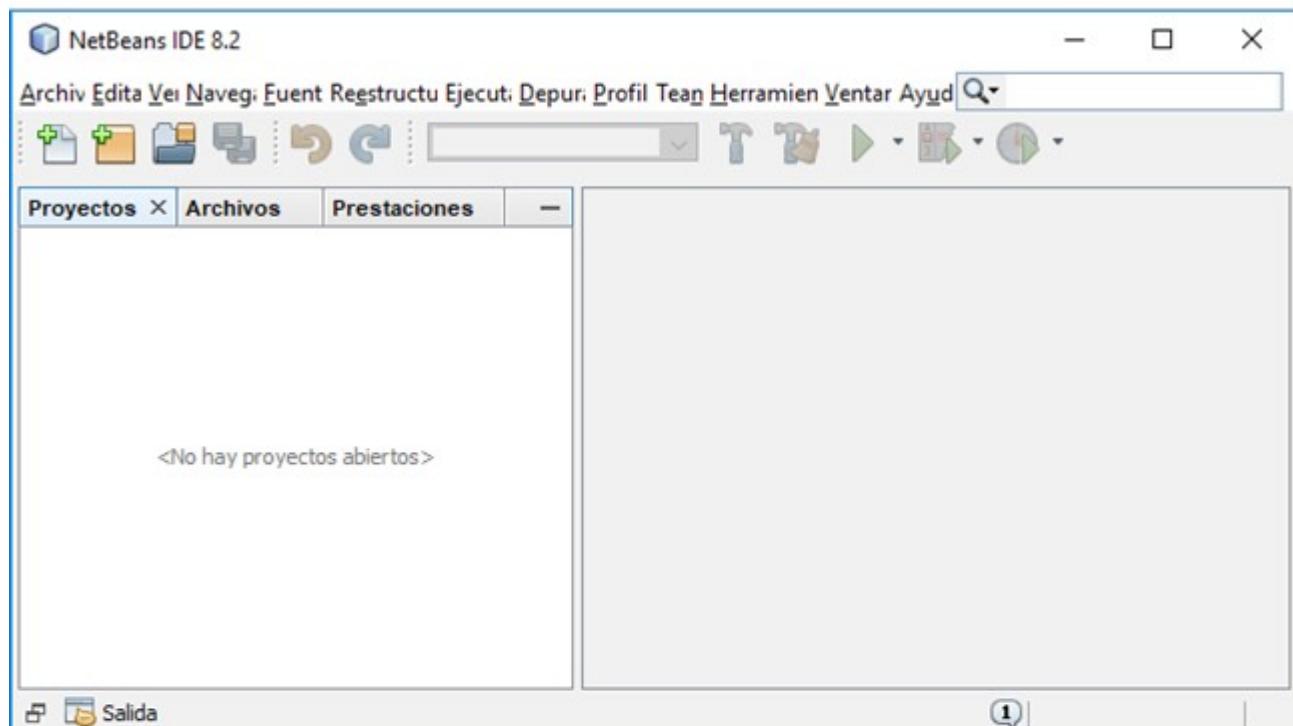


5. Implementación del diseño de clases.

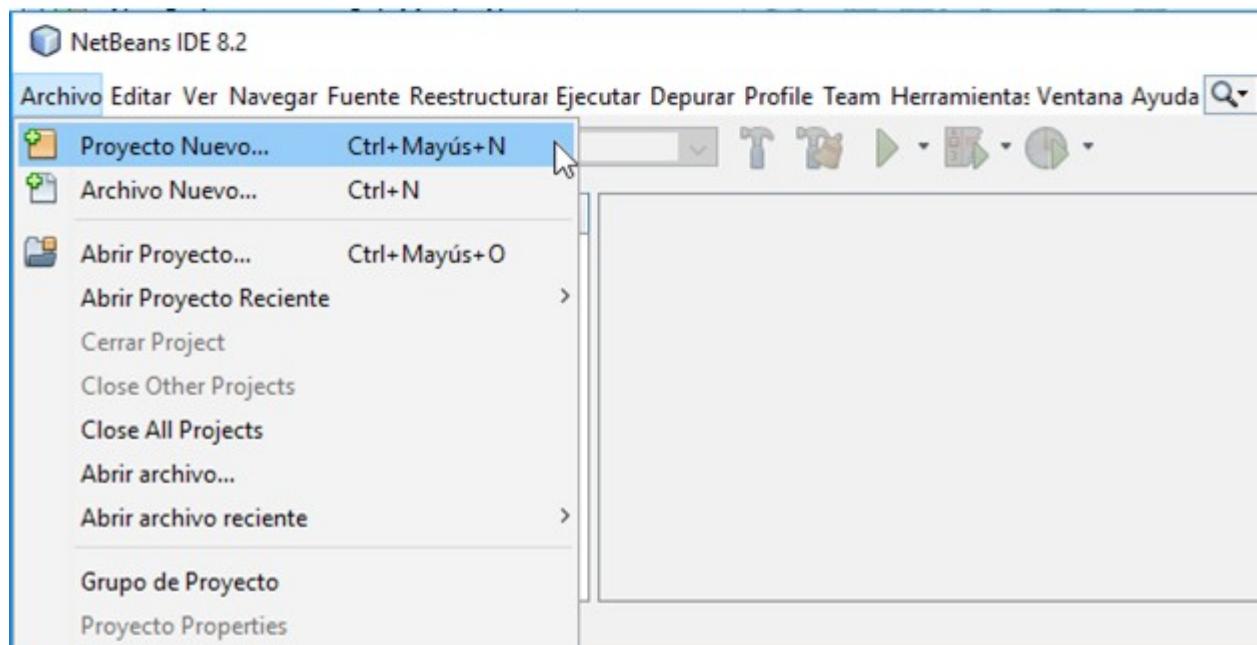
El diseño UML anterior se implementará usando el lenguaje de programación Java, para lo cual se desarrollará una aplicación de consola con el IDE *NetBeans* siguiendo para ello los siguientes pasos:

a. Creación del proyecto de consola en NetBeans

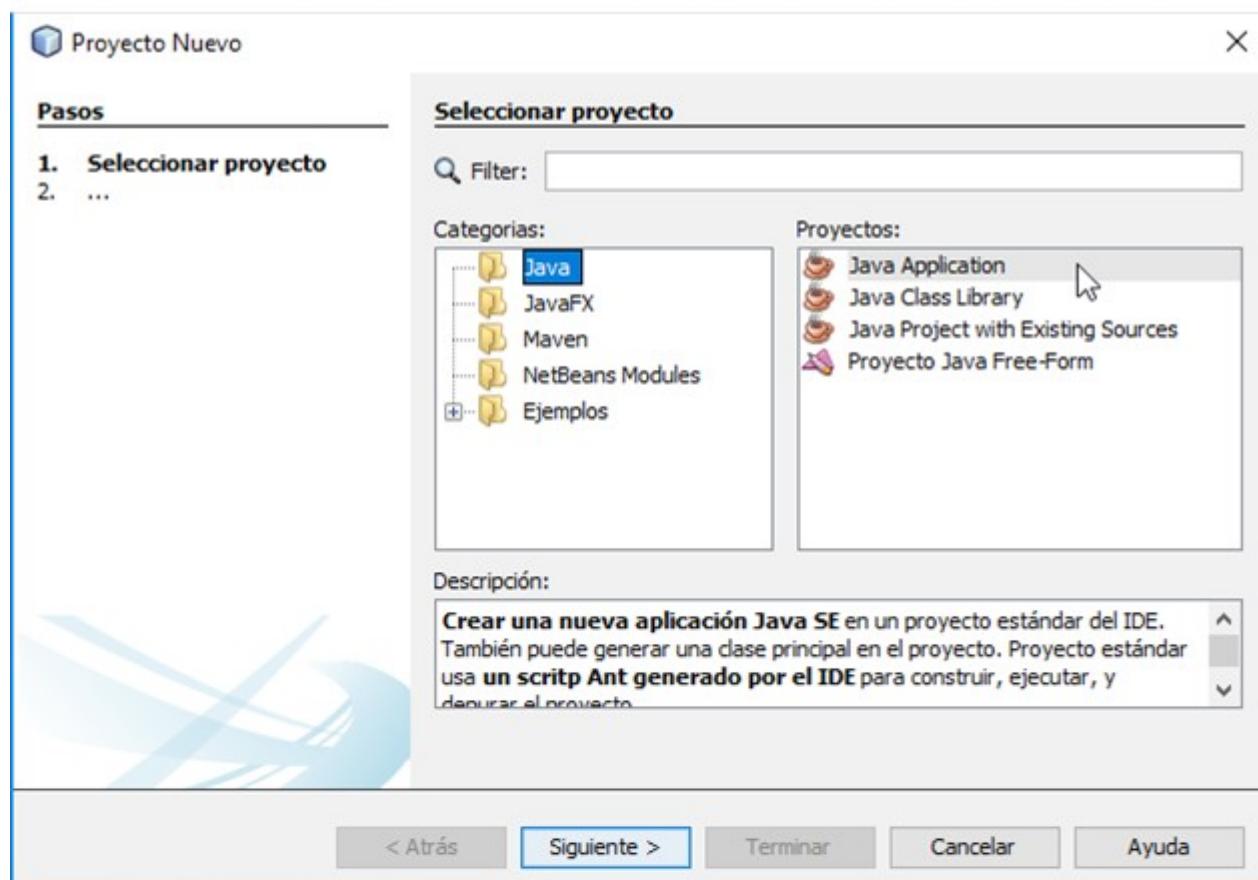
Inicie el IDE de *NetBeans* con lo cual se muestra una pantalla como siguiente:



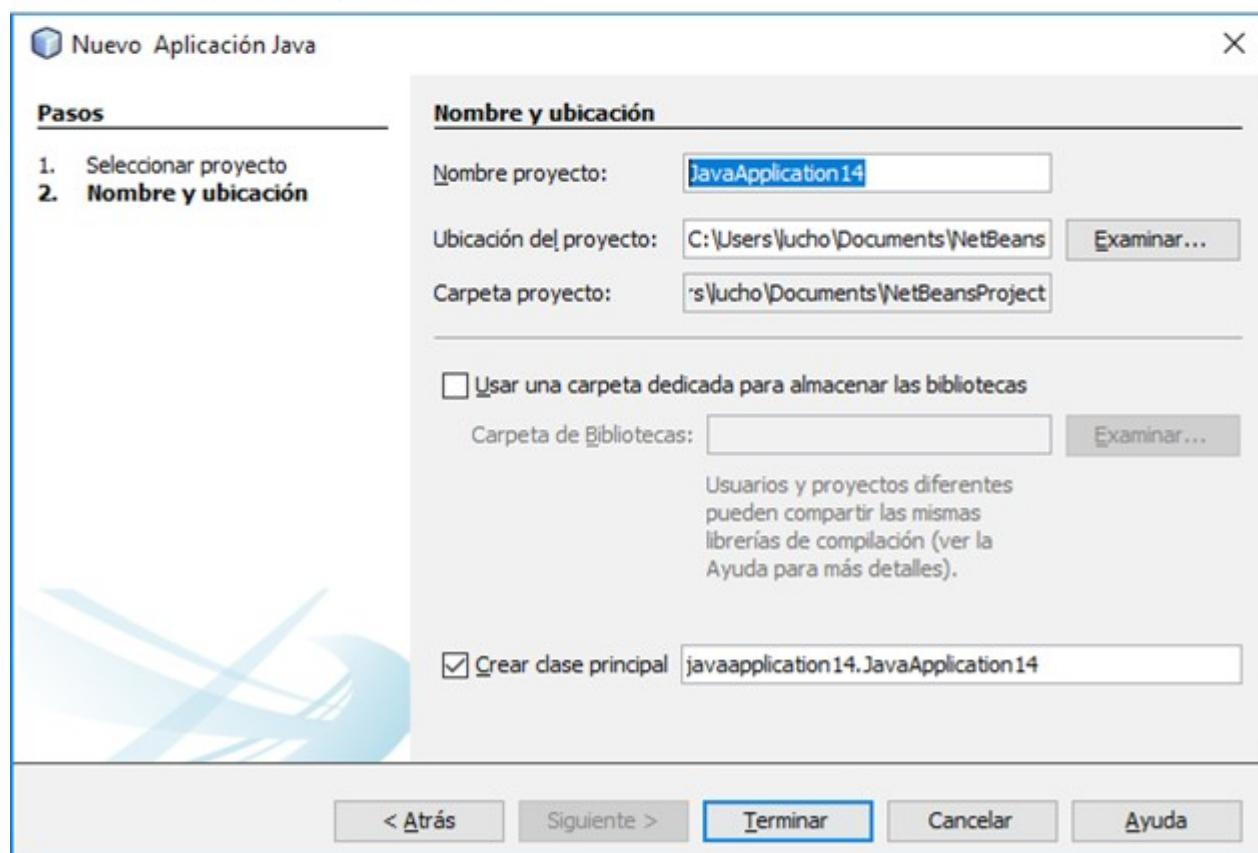
Para crear el proyecto entre por la opción **Archivo + Proyecto Nuevo**, tal como lo indica la siguiente imagen:



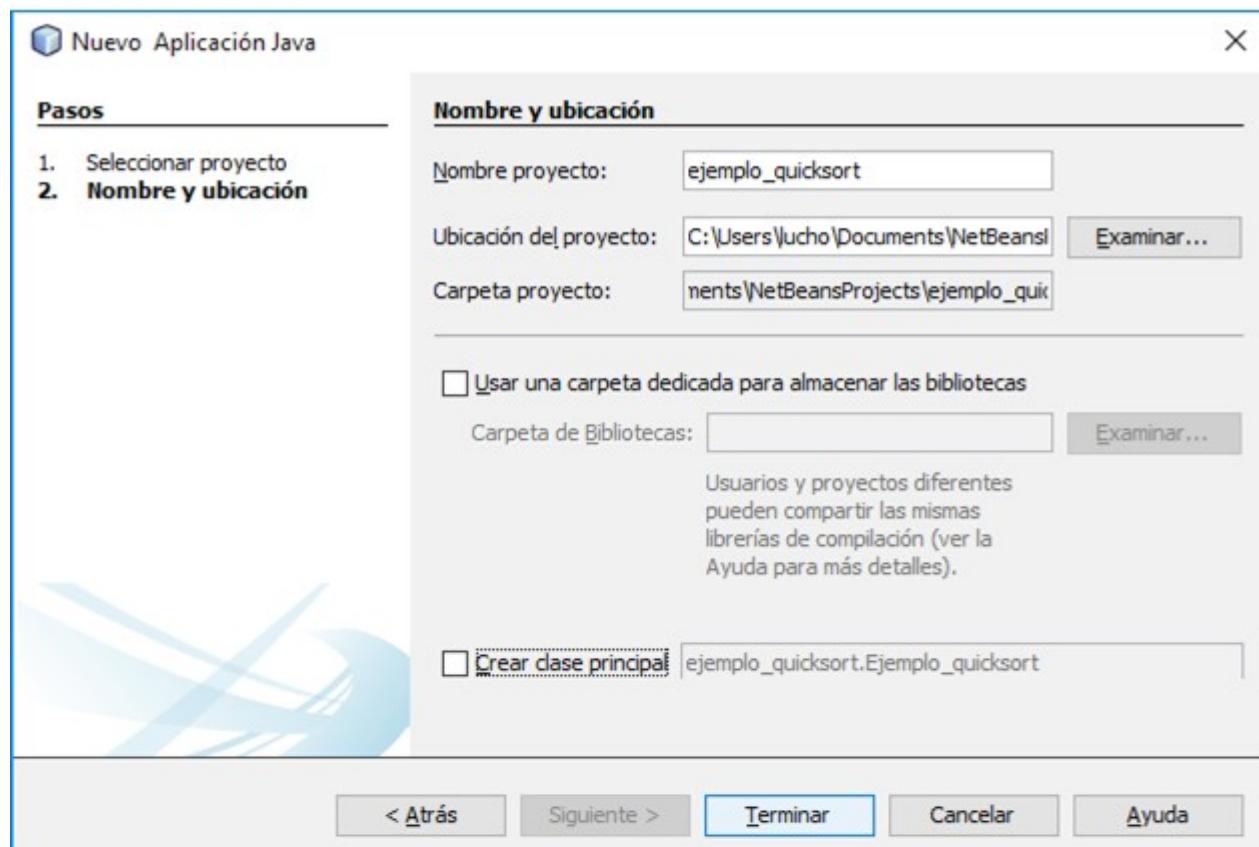
Ahora en la siguiente ventana seleccione el nodo Java en panel de **Categorías**, en el panel de proyectos seleccione el nodo "**Java Application**" y pulse el botón "**Siguiente**".



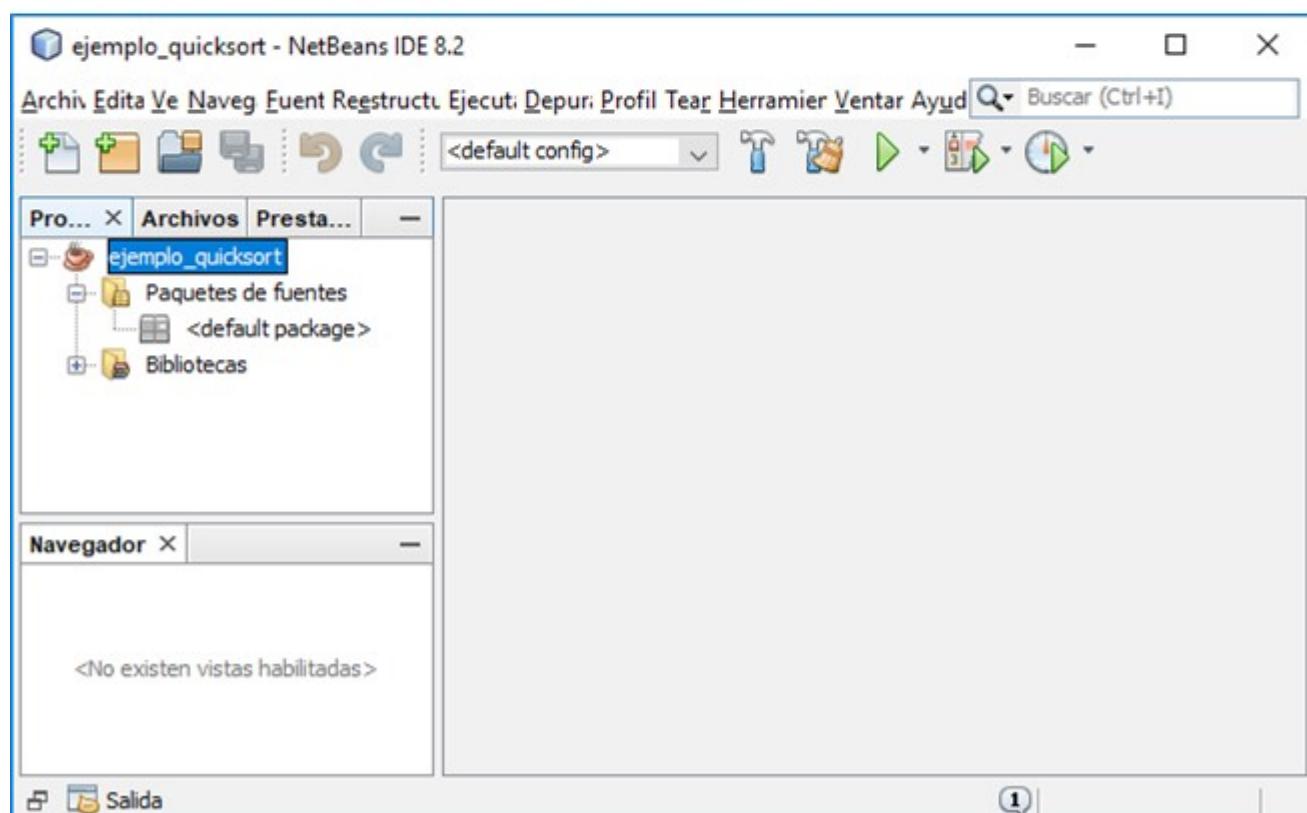
Presentándose la ventana siguiente:



En esta ventana en nombre del proyecto ponga **ejemplo_quicksort** como **Nombre de proyecto**, desmarque la casilla “**Crear clase principal**” y haga click en el botón “**Terminar**”.

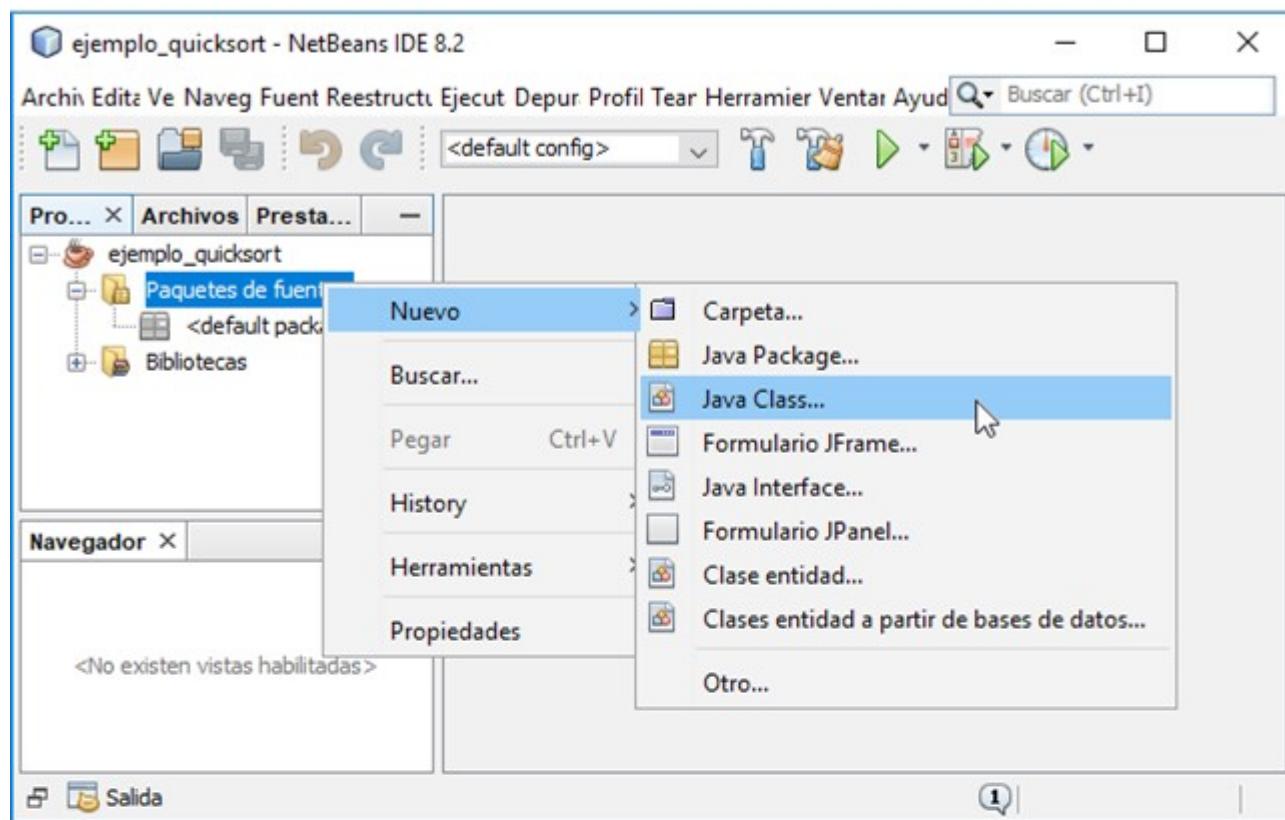


La imagen de abajo muestra el estado inicial del proyecto

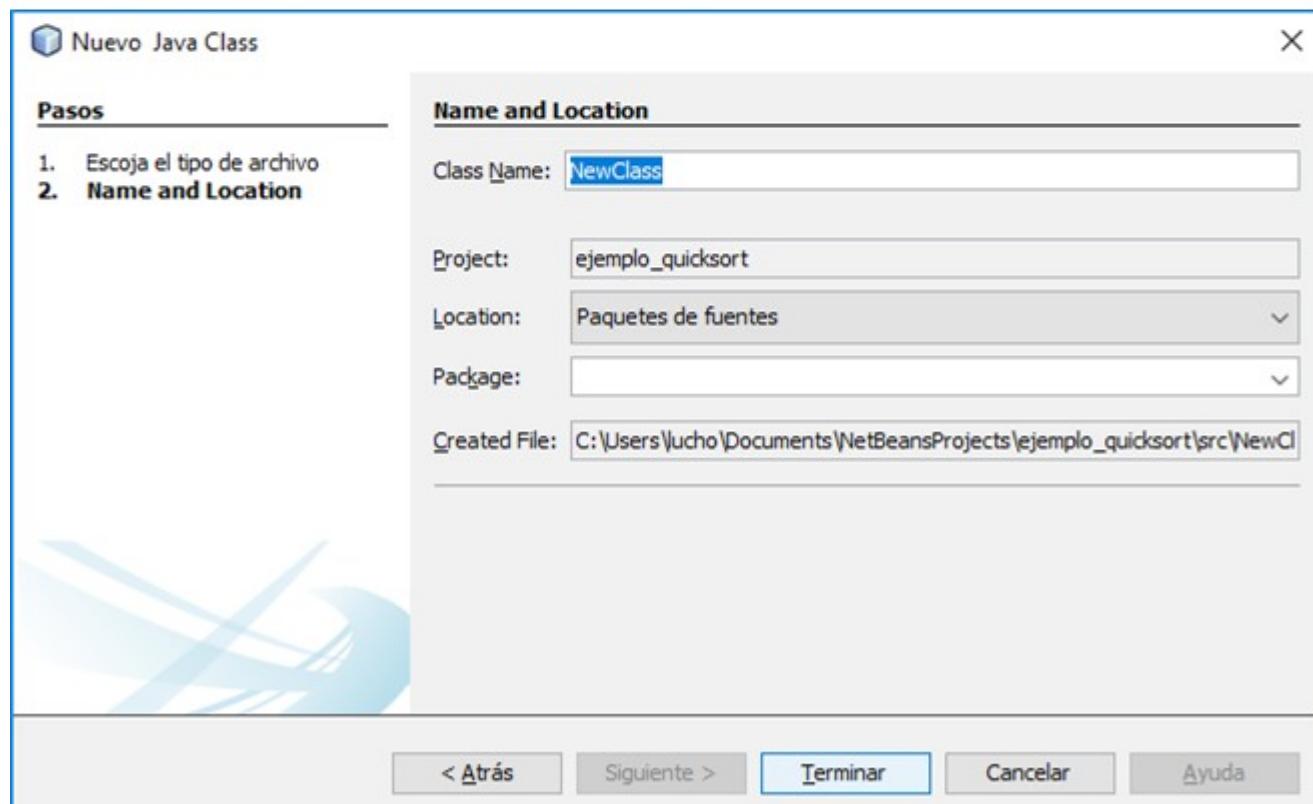


b. Creación de la clase *TVivienda*.

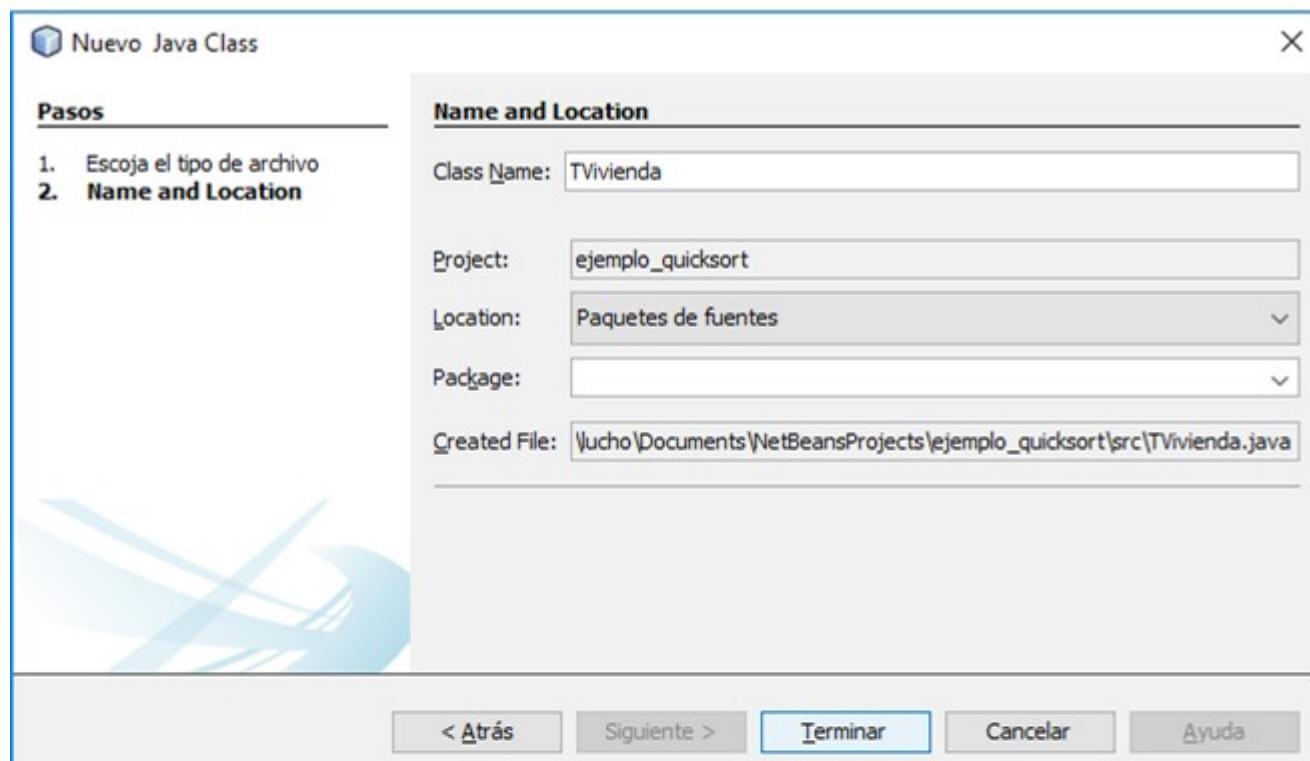
Para crear la clase *TVivienda* haga click derecho en el nodo **Paquetes de fuentes** y seleccione las opciones **Nuevo + Java Class** como lo indica la siguiente imagen:



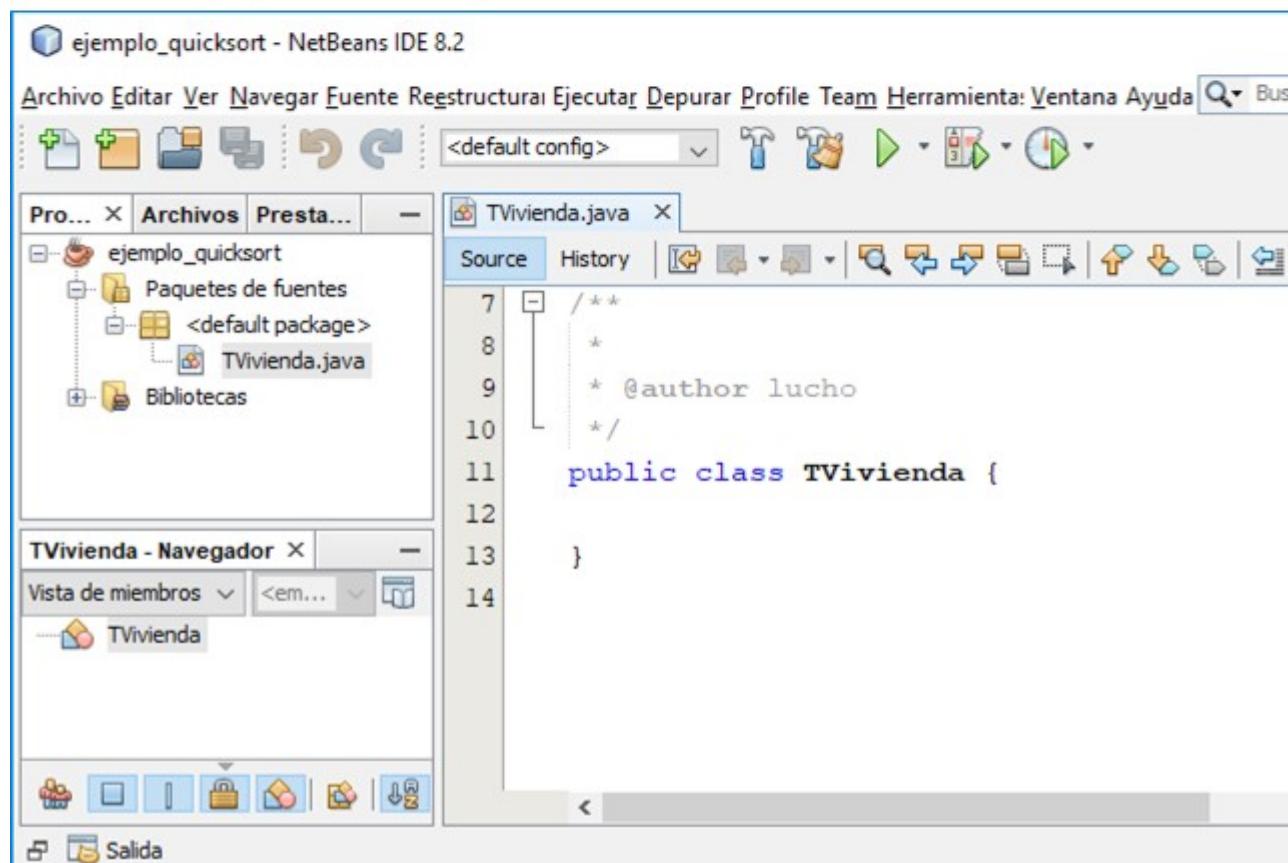
Con ello se despliega la ventana ilustrada abajo:



En esta ventana ingrese *TVivienda* por nombre de clase (***Class Name***), como se ve en la siguiente imagen. Después haga click en el botón “***Terminar***”:



Quedando el proyecto como se aprecia a continuación.



c. Implementación de la clase TVivienda

Asegúrese de que el código para el archivo *TVivienda* quede de la siguiente manera:

The screenshot shows a Java code editor window with the title bar "TVivienda.java". The menu bar includes "Source" and "History". Below the menu is a toolbar with various icons for file operations. The code itself is a Java class definition:

```
1  public class TVivienda {  
2      private long Código;  
3      private String Dirección;  
4      private byte Estrato;  
5      private float Área;  
6      private float Valor;  
7  
8      public TVivienda(){  
9          Código=0;  
10         Dirección="";  
11         Estrato=0;  
12         Área=0;  
13         Valor=0;  
14     }  
15  
16     public void set Código(long Cod){  
17         Código=Cod;  
18     }  
19  
20     public void set Dirección(String Dir){  
21         Dirección=Dir;  
22     }  
23  
24     public void set Estrato(byte Est){  
25         Estrato=Est;  
26     }  
27  
28     public void set Área(float Are){  
29         Área=Are;  
30     }  
31  
32     public void set Valor(float Val){  
33         Valor=Val;  
34     }  
35  
36     long get Código(){  
37         return Código;  
38     }  
39  
40     public String get Dirección(){  
41         return Dirección;  
42     }
```

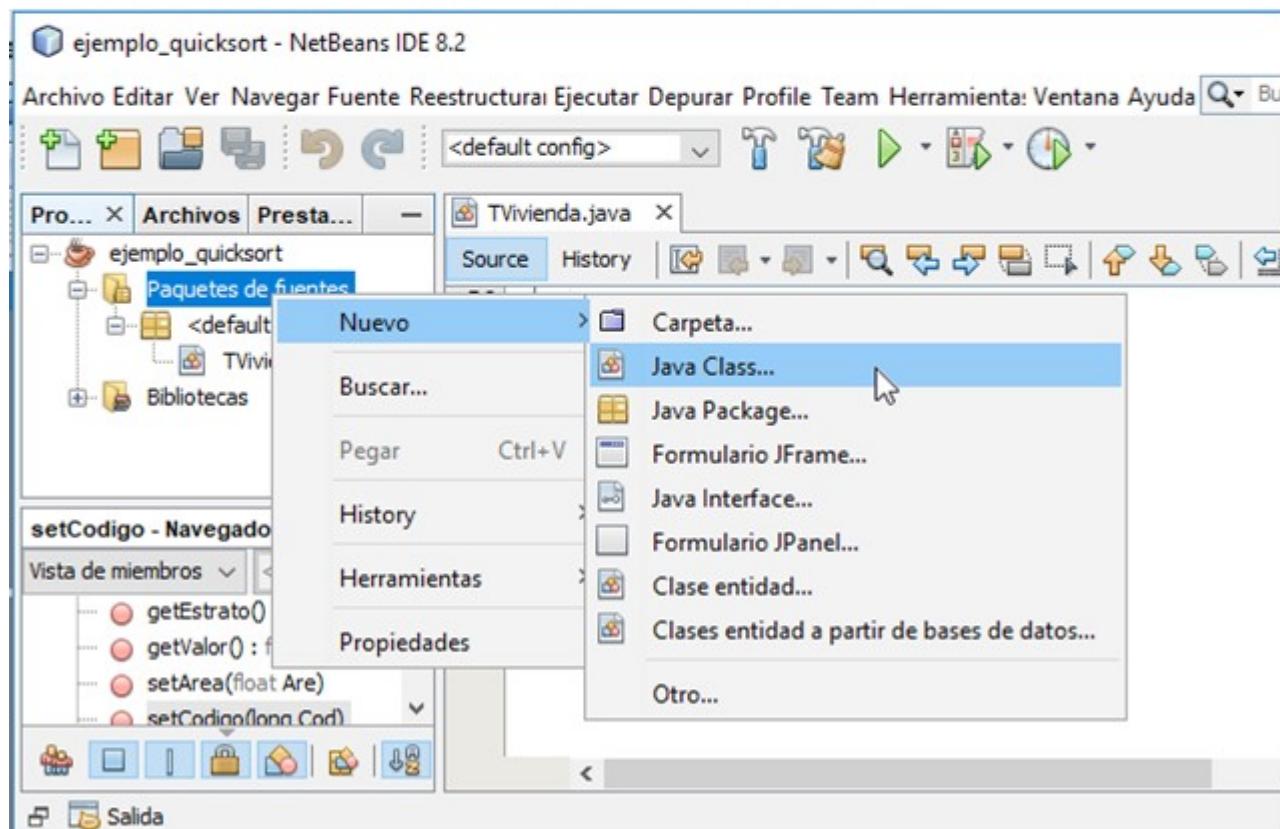
```

43
44     public byte getEstrato(){
45         return Estrato;
46     }
47
48     public float getArea(){
49         return Area;
50     }
51
52     public float getValor(){
53         return Valor;
54     }
55
56 }

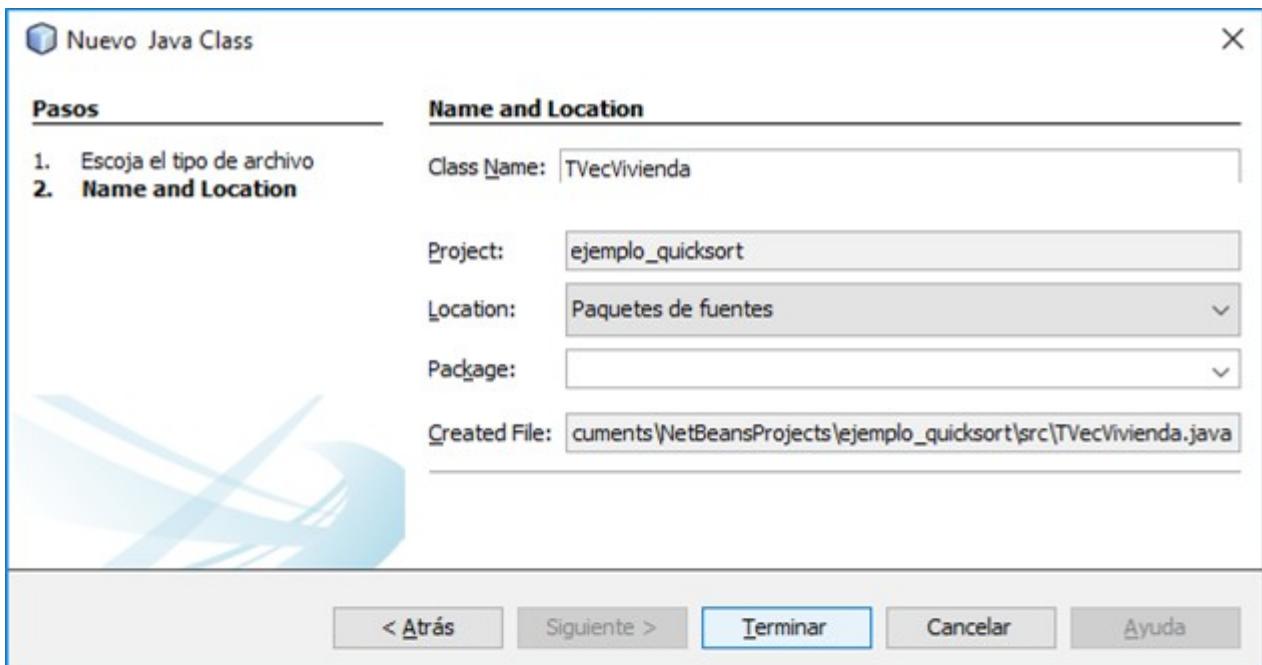
```

d. Creación de la clase *TVecVivienda*.

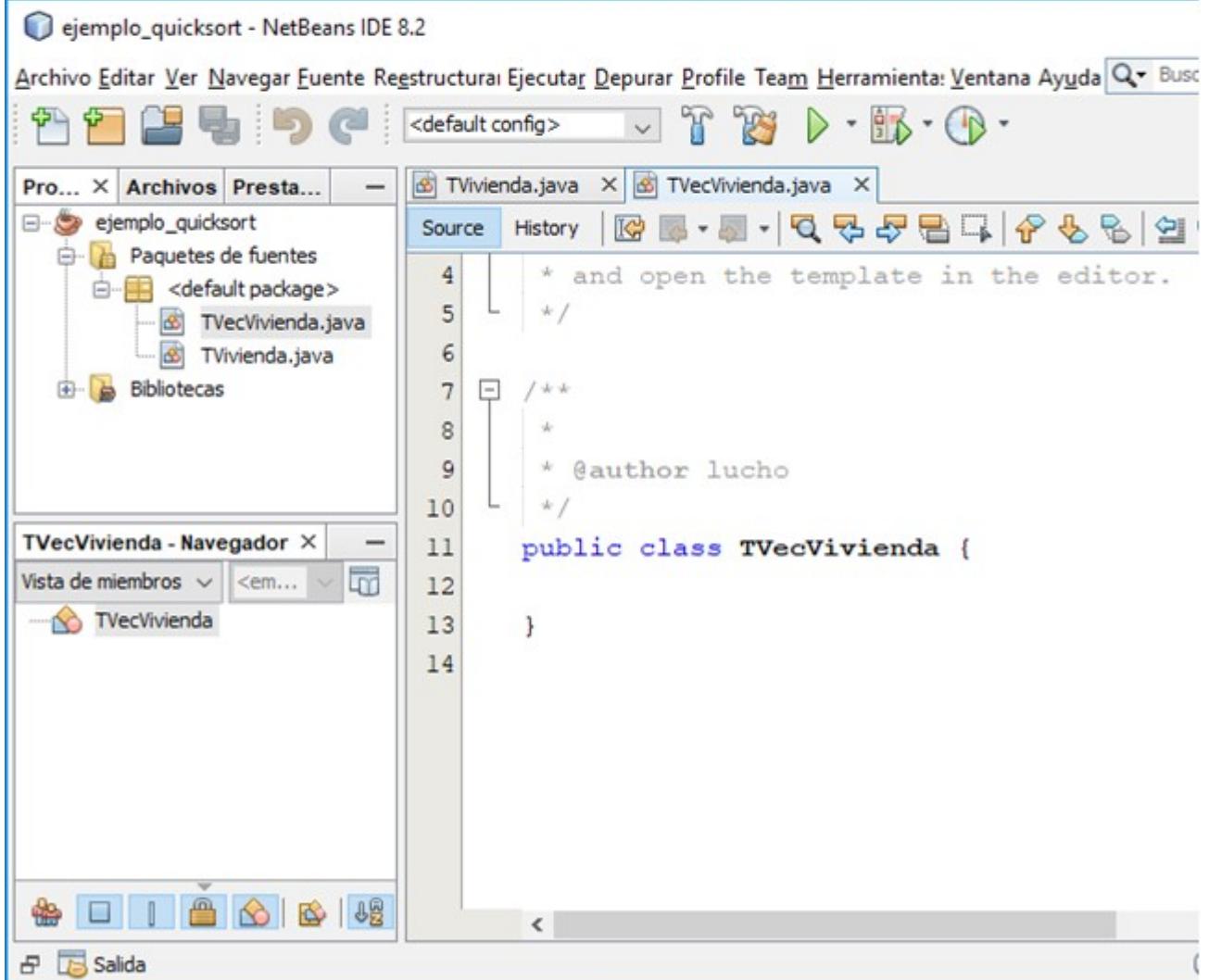
Para crear la clase *TVecVivienda* haga click derecho en el nodo **Paquetes de fuentes** y seleccione las opciones **Nuevo + Java Class** como lo indica abajo:



Una vez hecho estos pasos se mostrara una ventana similar a la expuesta en el caso anterior para la creación de la clase “*TVivienda*”, pero esta vez debe ingresar en la entrada titulada “**Class Name**” en nombre *TVecVivienda*, tal como se aprecia en la imagen de abajo de la siguiente página. Una vez ingrese el nombre de la clase haga click en el botón “**Terminar**”.



La siguiente imagen muestra cómo queda el proyecto después de crear esta clase:



e. Implementación de la clase *TVecVivienda*

Asegúrese de que el código para el archivo *TVecVivienda* quede de la siguiente manera:

The screenshot shows a Java code editor window titled "TVecVivienda.java". The code is a class definition for "TVecVivienda" with several methods: constructor, setters for Tam and Vec, getters for Tam and Vec, and a method to swap elements at indices P1 and P2.

```
1 public class TVecVivienda {
2     private int Tam;
3     private TVivienda Vec[];
4
5     public TVecVivienda(){
6         Tam=0;
7         Vec=null;
8     }
9
10    public void setTam(int N){
11        int i;
12        Tam=N;
13        if(Tam>0){
14            Vec=new TVivienda[Tam];
15            for(i=0;i<Tam;i++){
16                Vec[i]=new TVivienda();
17            }
18        }
19        else
20            Vec=null;
21    }
22
23    public void setVec(int Pos,TVivienda Viv){
24        Vec[Pos]=Viv;
25    }
26
27    public int getTam(){
28        return Tam;
29    }
30
31    public TVivienda getVec(int Pos){
32        return Vec[Pos];
33    }
34
35    public void Cambiar(int P1,int P2){
36        TVivienda Viv;
37        Viv=Vec[P1];
38        Vec[P1]=Vec[P2];
39        Vec[P2]=Viv;
40    }
}
```

```

41
42     public void OrdenaArea() {//Método intercambio
43         int i,j;
44         for(i=0;i<Tam;i++){
45             for(j=i+1;j<Tam;j++){
46                 if(Vec[j].getArea()<Vec[j].getArea()){
47                     Cambiar(i,j);
48                 }
49             }
50         }
51     }
52
53     public void OrdenaEstrato() {//Método Insercion
54         int i,j;
55         for(i=1;i<Tam;i++){
56             j=i;
57             while(j>0){
58                 if(Vec[j-1].getEstrato()>Vec[j].getEstrato()){
59                     Cambiar(j-1,j);
60                     j=j-1;
61                 }
62                 else
63                     break;
64             }
65         }
66     }
67
68     public void OrdenaValor() {//Método shell
69         int i,j,d;
70         d=Tam/2;
71         while(d>=1){
72             for(i=d;i<Tam;i++){
73                 j=i;
74                 while(j-d>=0){
75                     if(Vec[j-d].getValor()<Vec[j].getValor()){
76                         Cambiar(j-d,j);
77                         j=j-d;
78                     }
79                     else
80                         break;
81                 }
82             }
83             d=d/2;
84         }
85     }
86

```

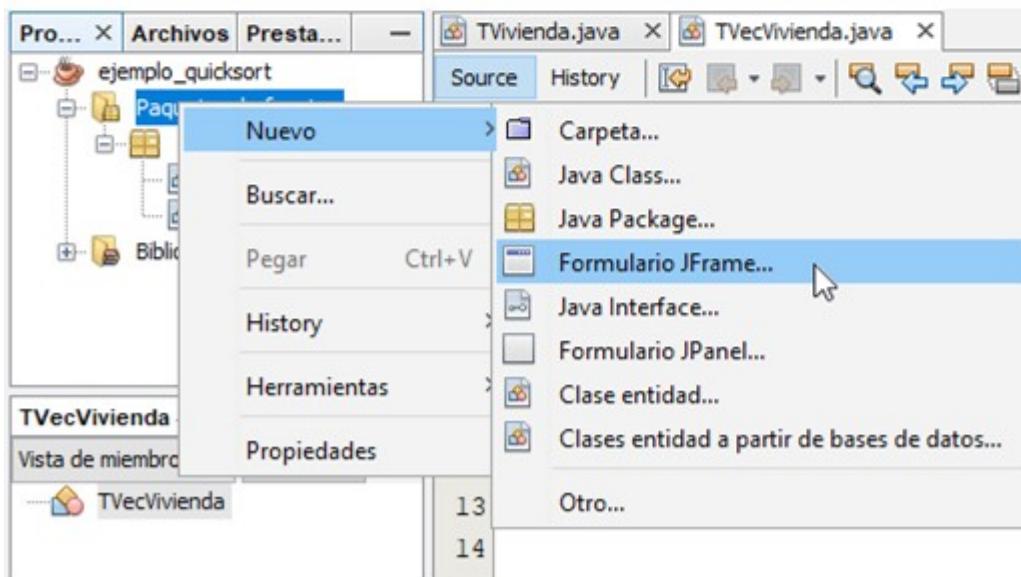
```

87  private void OrdenaParte(int Izq,int Der){
88      int i,d,centro;
89      float pivot;
90      centro=(Izq + Der)/2;
91      pivot=Vec[centro].getValor();
92      i=Izq;
93      d=Der;
94      while(i<=d){
95          while(i<Der && Vec[i].getValor()<pivot)
96              i++;
97          while(d>Izq && Vec[d].getValor()>pivot)
98              d--;
99          if(i<=d){
100             Cambiar(i,d);
101             i++;
102             d--;
103         }
104     }
105     if(i<Der)
106         OrdenaParte(i,Der);
107     if(d>Izq)
108         OrdenaParte(Izq,d);
109   }
110
111  public void OrdenaCodigo(){//Método QuickSort
112      OrdenaParte(0,Tam-1);
113  }
114
115
116 }

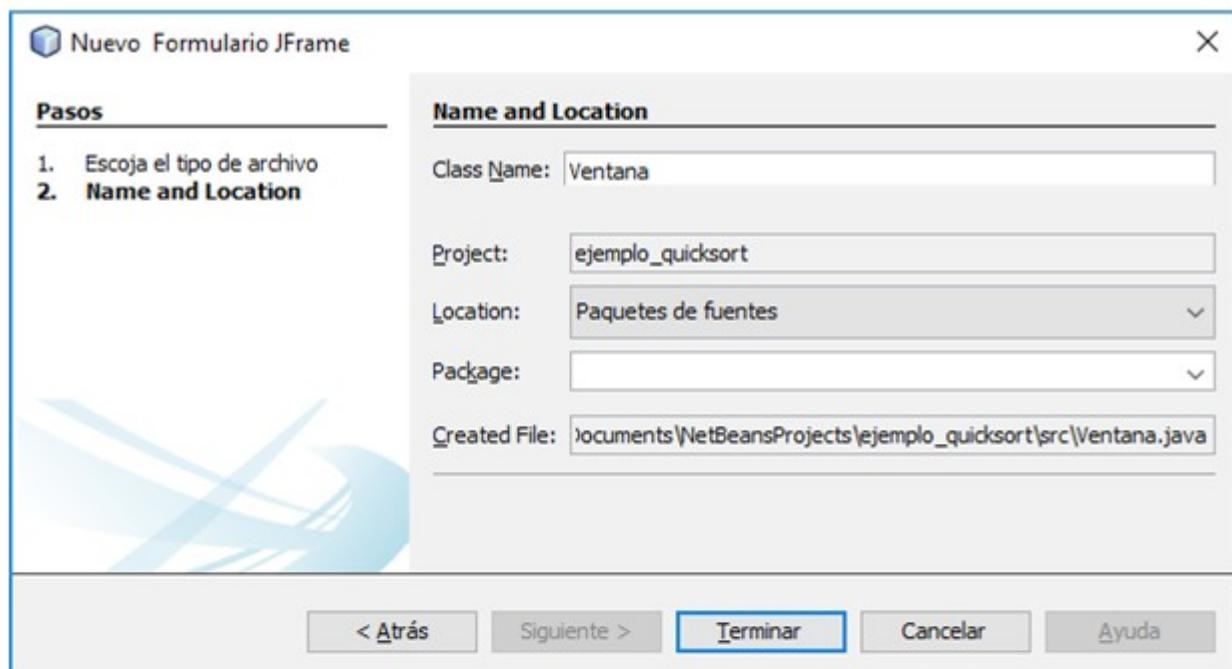
```

f. Diseño gráfico de la ventana

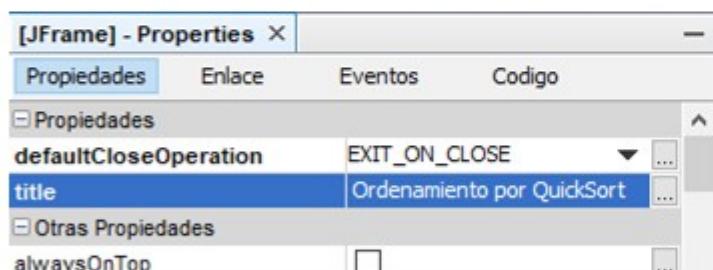
- Haga click derecho en nodo **Paquetes de fuentes** del proyecto, escoja la opción **Nuevo** y luego **Formulario JFrame**, tal como se muestra en la siguiente imagen:



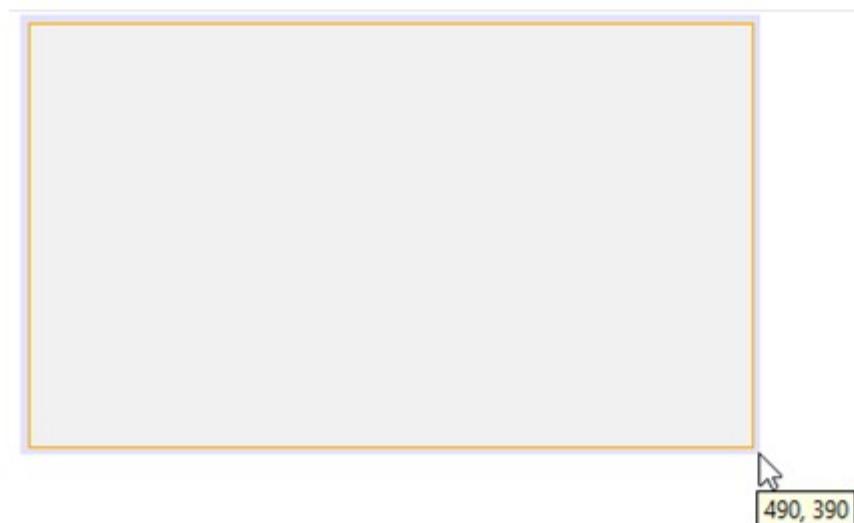
- En la ventana desplegada, en la entrada **Class Name**, ingrese el nombre para la clase (**Ventana**) y haga click en el botón **Terminar**.



- 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 **Ordenamiento por QuickSort** como el título para la ventana y pulse enter:



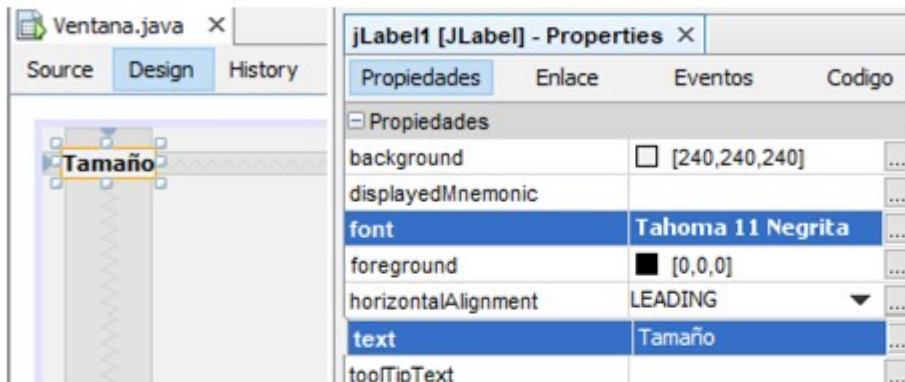
- Redimensione la ventana (sujetándola por el nodo inferior derecho) hasta que tenga un tamaño de 490 (ancho) por 390 (alto).



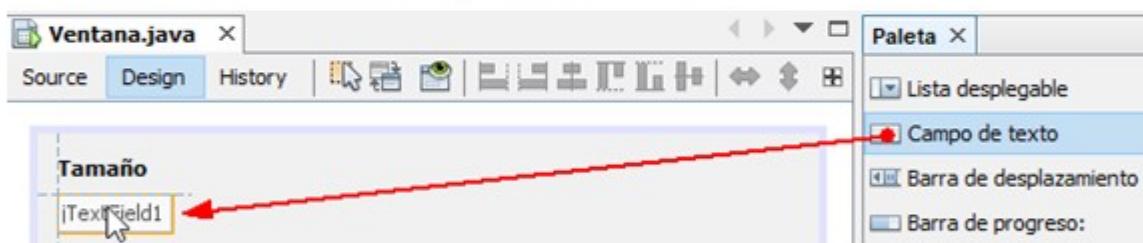
- Arrastre un **JLabel** (etiqueta) sobre la ventana hasta el punto indicado:



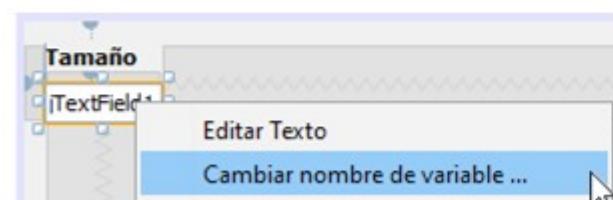
- En la propiedad **text** escriba **Tamaño** y en la propiedad **font** ponga el texto en negrita.



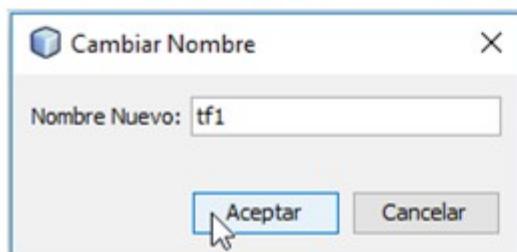
- Arrastre un **JTextField** hasta el punto indicado:



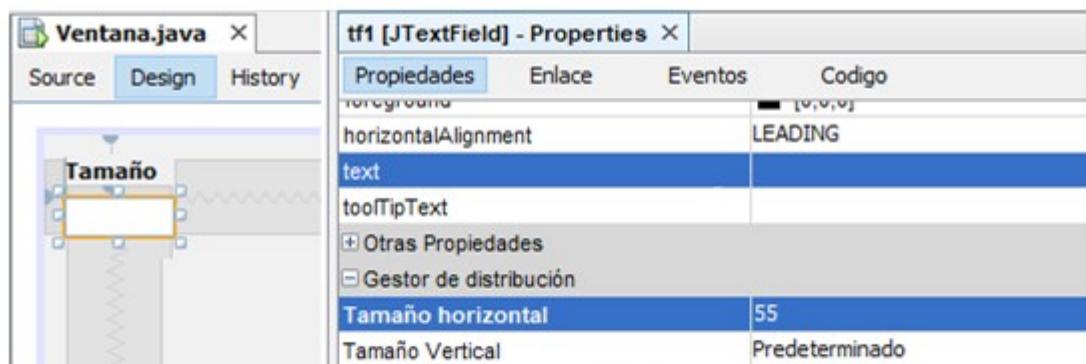
- Haga click derecho sobre el **JTextField** y seleccione la opción “**cambiar nombre de variable ...**”



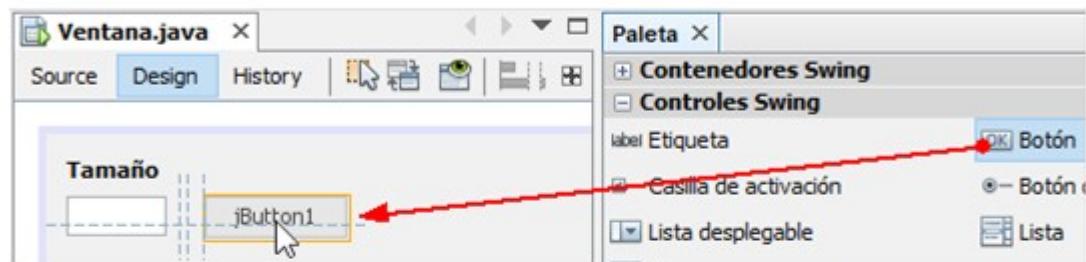
- Escriba **tf1** como nombre de instancia y haga click en el botón **Aceptar**.



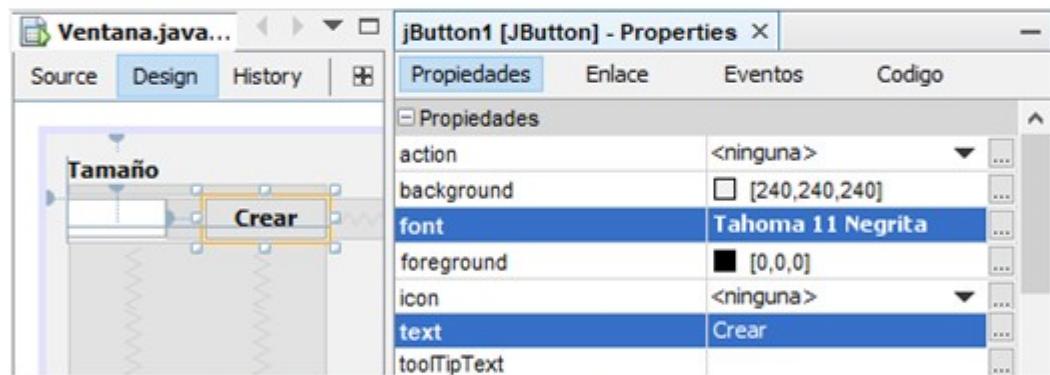
- En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo “**Gestor de distribución**”, ponga 55 como valor para la propiedad **Tamaño horizontal**.



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



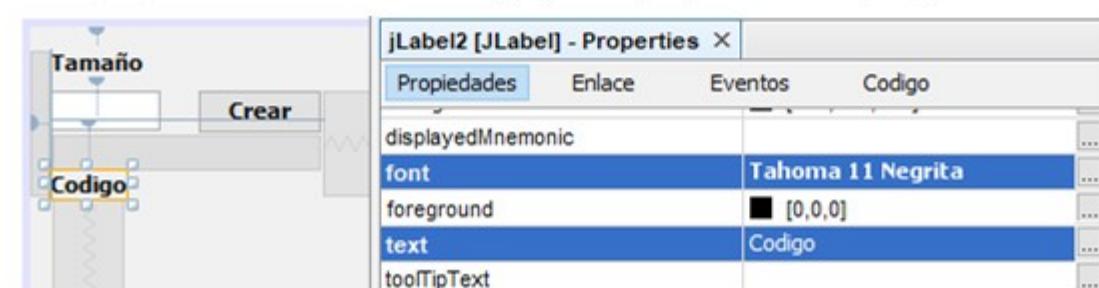
- En el inspector de propiedades, ponga **Crear** en la propiedad **text** y en la propiedad **font** asigne texto en negrita.



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



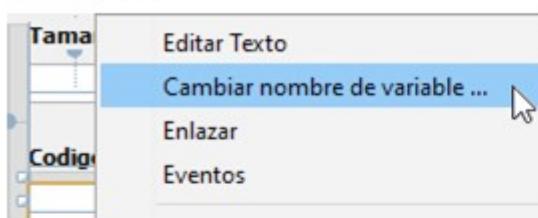
- En la propiedad **text** escriba **Código** y en la propiedad **font** ponga el texto en negrita.



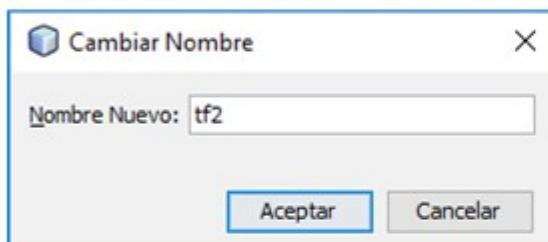
- Arrastre un **JTextField** hasta la posición señalada:



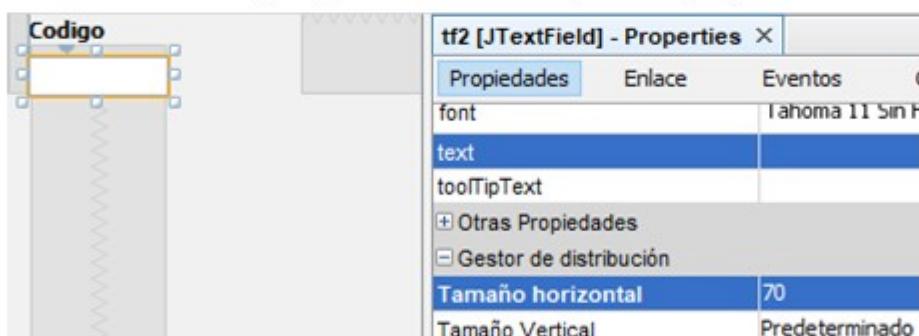
- Haga click derecho sobre el **JTextField** y seleccione la opción “**cambiar nombre de variable ...**”



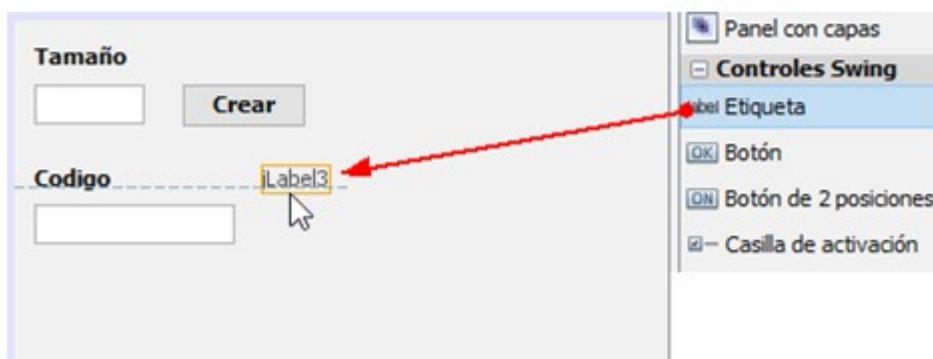
- Asigne **tf2** como nombre de instancia y haga click en el botón **Aceptar**.



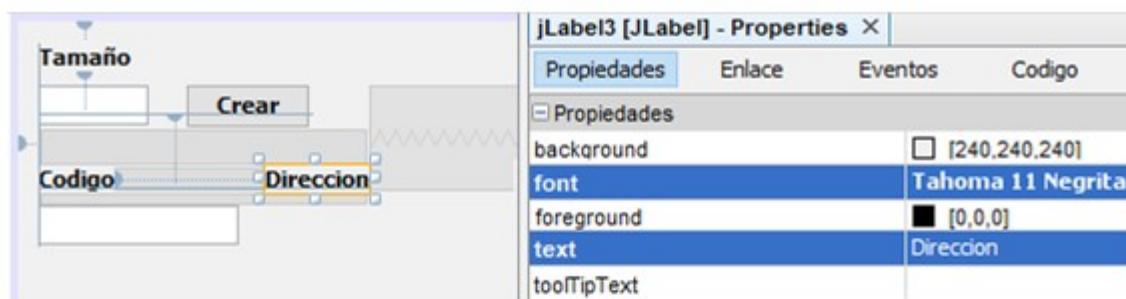
- En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo “**Gestor de distribución**”, ponga 55 como valor para la propiedad **Tamaño horizontal**:



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



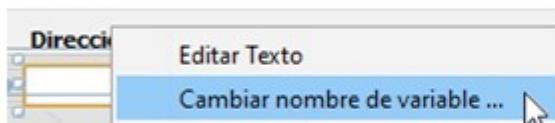
- En el inspector de propiedades, para la propiedad **text** escriba **Dirección** y en la propiedad **font** ponga el texto en negrita.



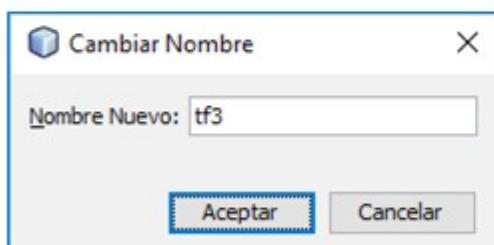
- Arrastre hasta este punto un **JTextField**



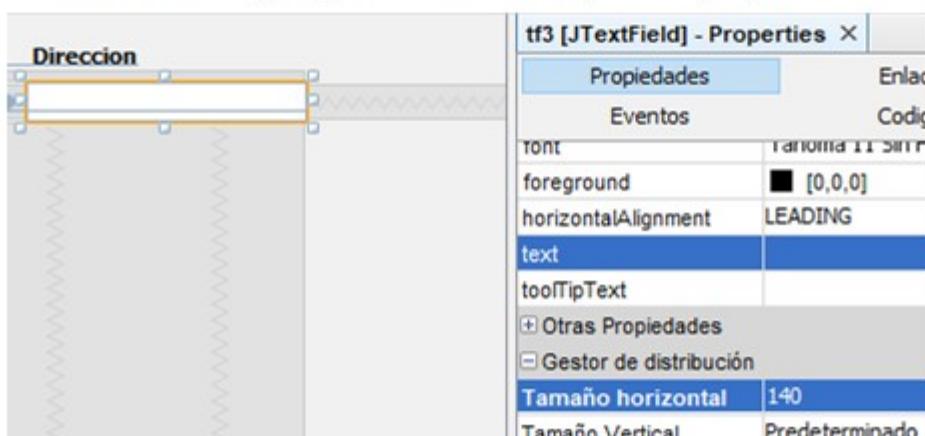
- Haga click derecho sobre el **JTextField** y seleccione la opción “**cambiar nombre de variable ...**”



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



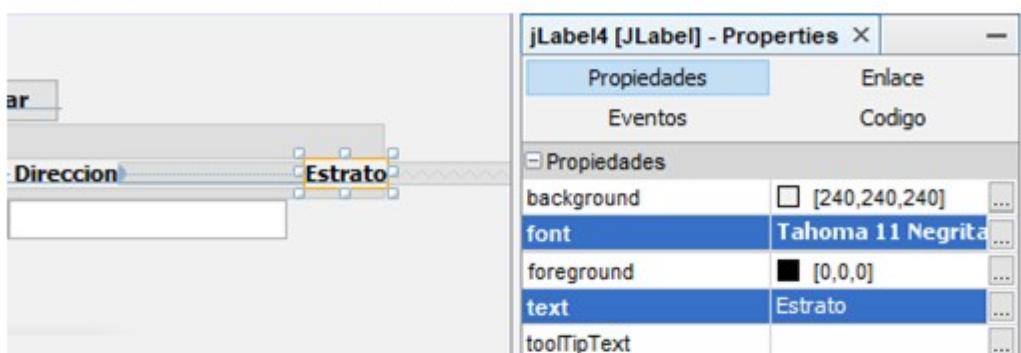
- En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo “**Gestor de distribución**”, ponga 140 como valor para la propiedad **Tamaño horizontal**.



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



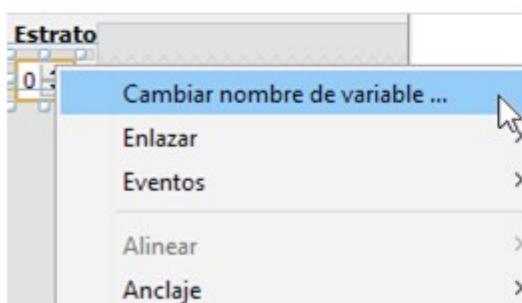
- En el inspector de propiedades, para la propiedad **text** escriba **Estrato** y en la propiedad **font** ponga el texto en negrita.



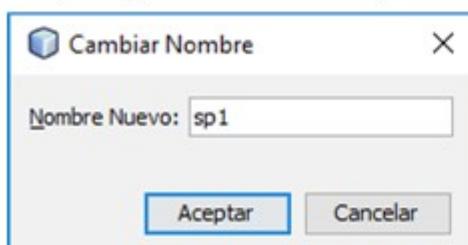
- Arrastre un **JSpinner** hasta la posición señalada:



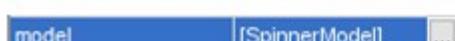
- Haga click derecho sobre el **JSpinner** y seleccione la opción “**cambiar nombre de variable ...**”



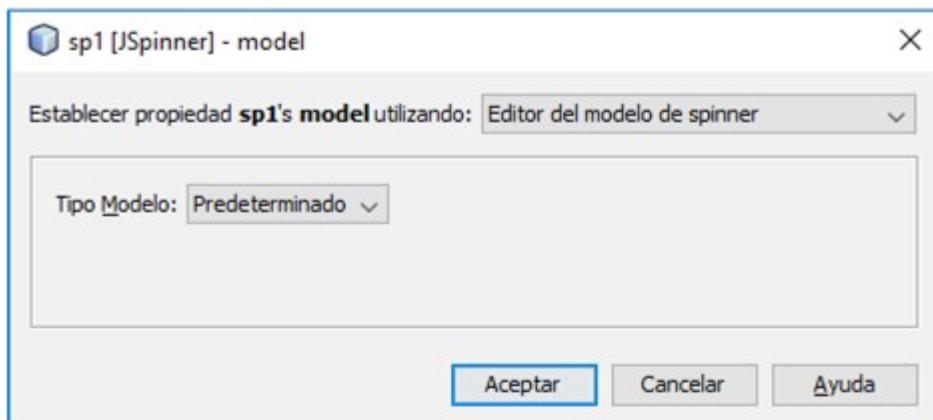
- Asigne **sp1** como nombre para el **JSpinner** y haga click en el botón **Aceptar**.



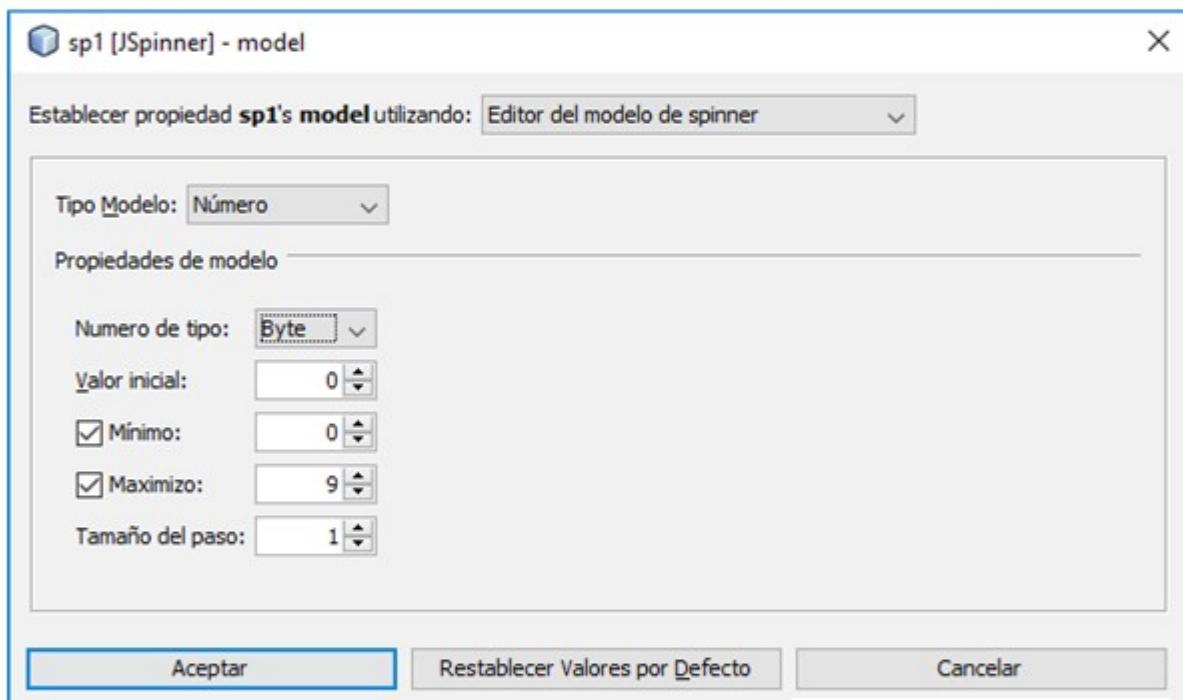
- En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



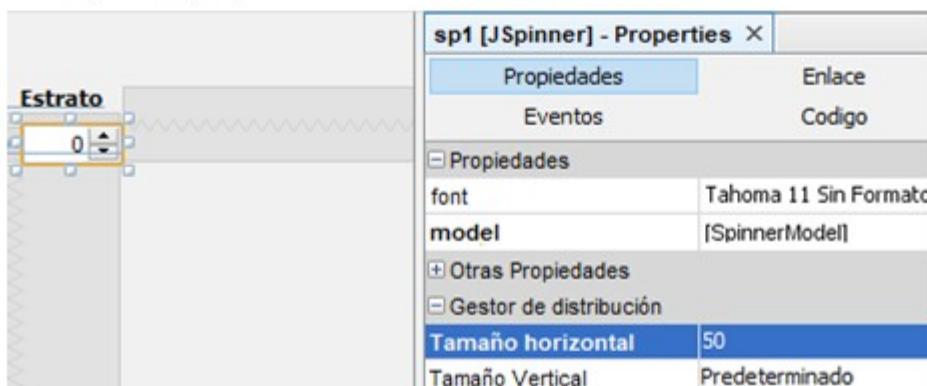
- Con ello vera una pantalla como esta:



- En **tipo de modelo** seleccione **Número**; en **Número de tipo** seleccione **Byte**, en **Valor inicial** ponga 0, marque la casilla **Mínimo** y asegúrese de que su valor sea 0, marque la casilla **Máximo** y ponga el valor de 9; finalmente haga click en el botón **Aceptar**.



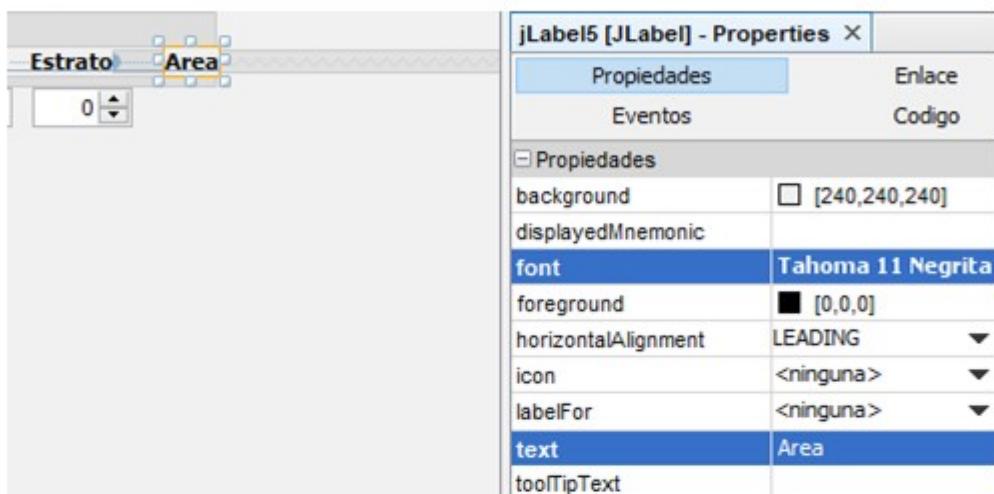
- En el inspector de propiedades en el nodo “**Gestor de distribución**”, ponga 50 como valor para la propiedad **Tamaño horizontal**:



- Arrastre un **JLabel** hasta el punto mostrado abajo:



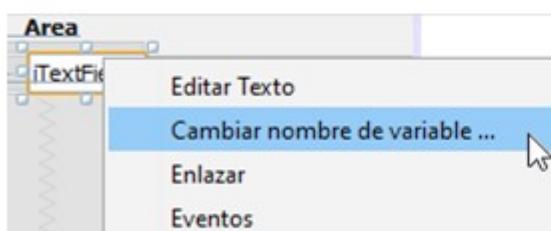
- En el inspector de propiedades, para la propiedad **text** escriba **Área** y en la propiedad **font** ponga el texto en negrita.



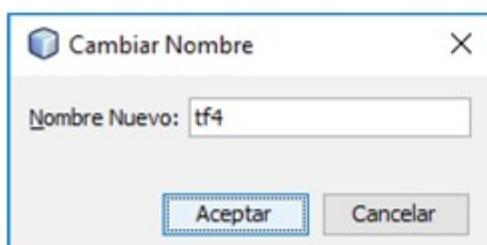
- Arrastre hasta este punto un **JTextField**



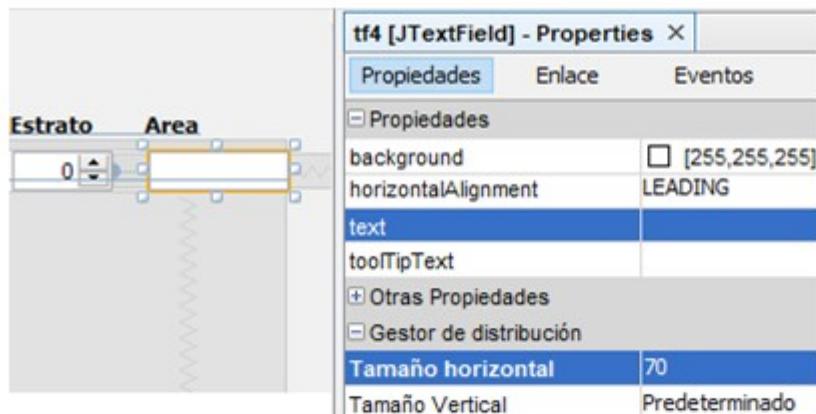
- Haga click derecho sobre el **JTextField** y seleccione la opción “**cambiar nombre de variable ...**”



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



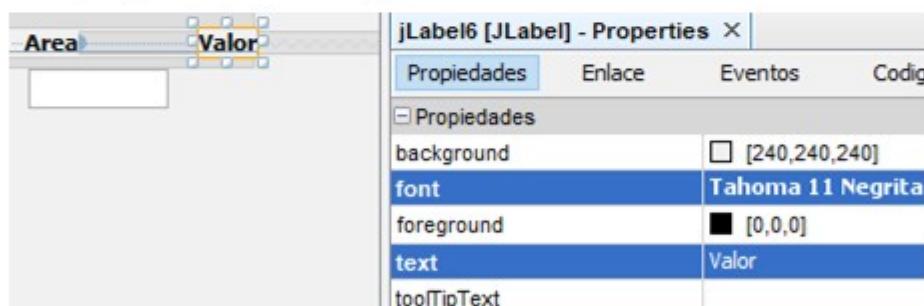
- En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo “**Gestor de distribución**”, ponga 70 como valor para la propiedad **Tamaño horizontal**.



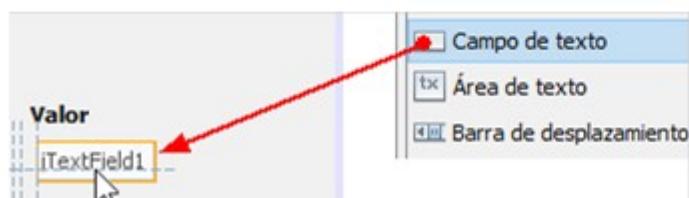
- Arrastre un **JLabel** hasta el punto mostrado abajo:



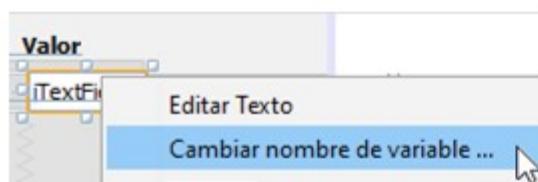
- En el inspector de propiedades, para la propiedad **text** escriba **Valor** y en la propiedad **font** ponga el texto en negrita.



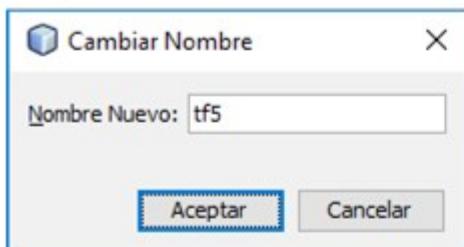
- Arrastre hasta este punto un **JTextField**



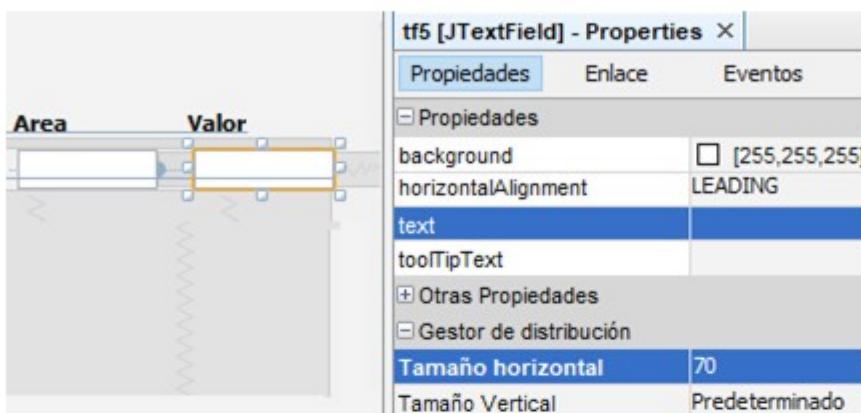
- Haga click derecho sobre el **JTextField** y seleccione la opción “**cambiar nombre de variable ...**”



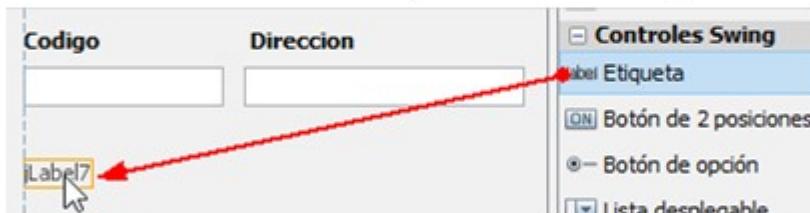
- Asigne **tf5** como nombre de instancia y haga click en el botón **Aceptar**.



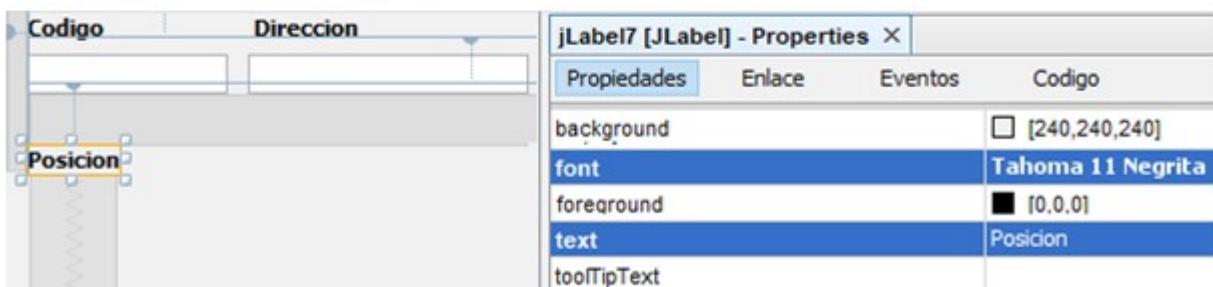
- En el inspector de propiedades borre el valor de la propiedad **text** y en el nodo “**Gestor de distribución**”, ponga 70 como valor para la propiedad **Tamaño horizontal**.



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



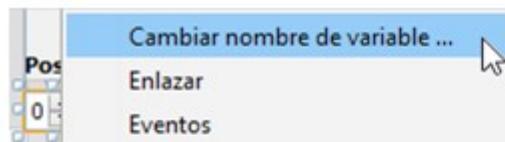
- En el inspector de propiedades, para la propiedad **text** escriba **Valor** y en la propiedad **font** ponga el texto en negrita.



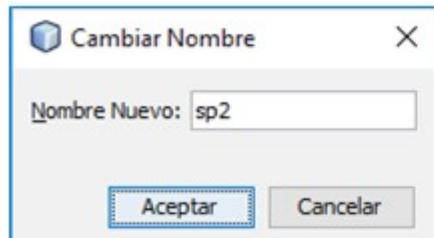
- Arrastre un **JSpinner** hasta la posición señalada:



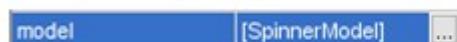
- Haga click derecho sobre el **JSpinner** y seleccione la opción “**cambiar nombre de variable ...**”



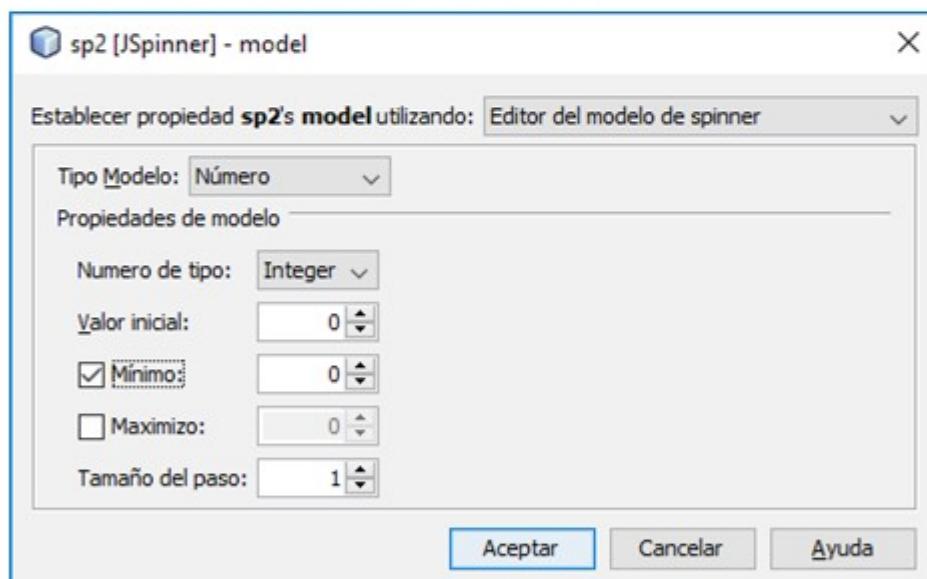
- Asigne **sp2** como nombre para el **JSpinner** y haga click en el botón **Aceptar**.



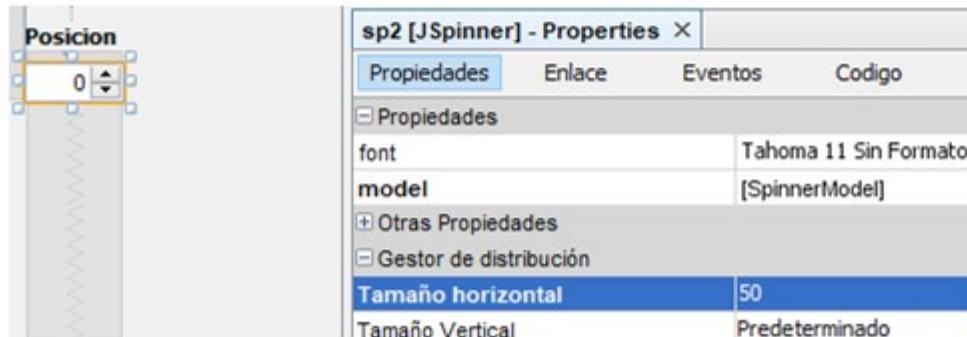
- En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



- En **tipo de modelo** seleccione **Numero**, 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**.



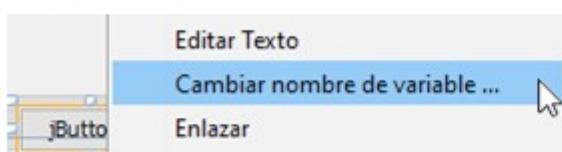
- En el inspector de propiedades en el nodo “**Gestor de distribución**”, ponga 50 como valor para la propiedad **Tamaño horizontal**.



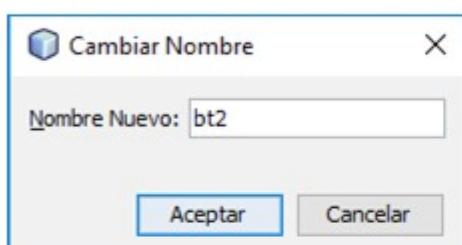
- Arrastre hasta la ventana un **JButton** en el punto señalado:



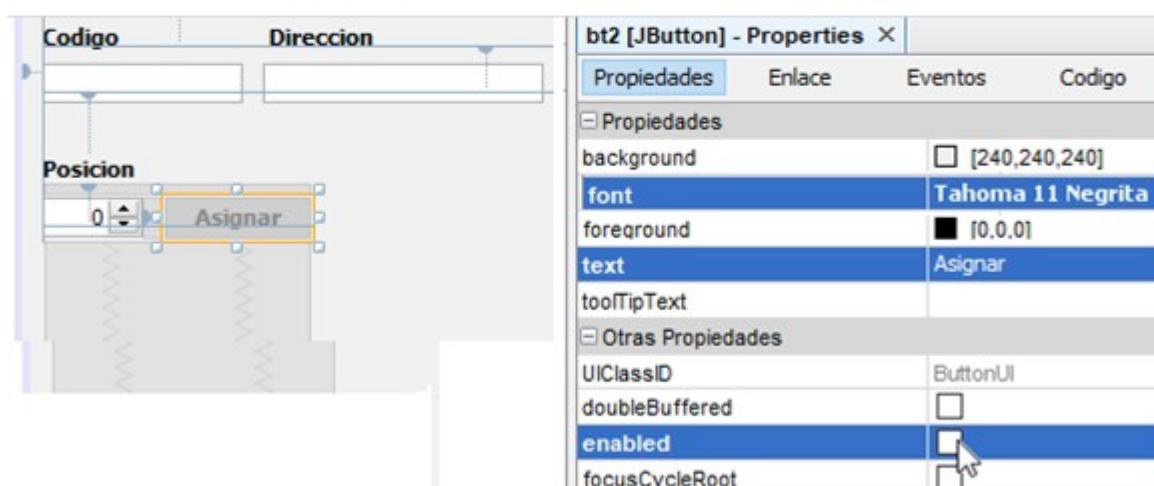
- Haga click derecho sobre el **JButton** y seleccione la opción “**cambiar nombre de variable ...**”



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



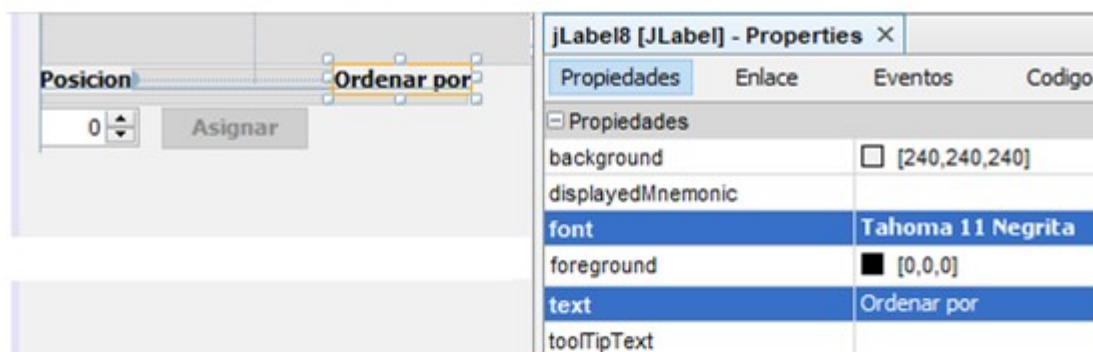
- En el inspector de propiedades, ponga **Asignar** en la propiedad **text**, en la propiedad **font** asigne texto en negrilla y en el nodo “**Otras propiedades**” desmarque la casilla a la derecha de la propiedad **enabled**, para que el botón inicialmente este deshabilitado.



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



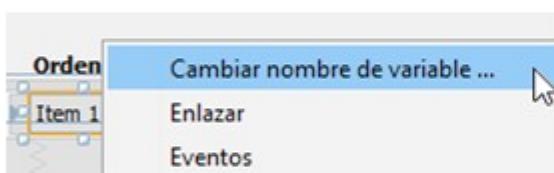
- En el inspector de propiedades, para la propiedad **text** escriba **Ordenar por** y en la propiedad **font** ponga el texto en negrita.



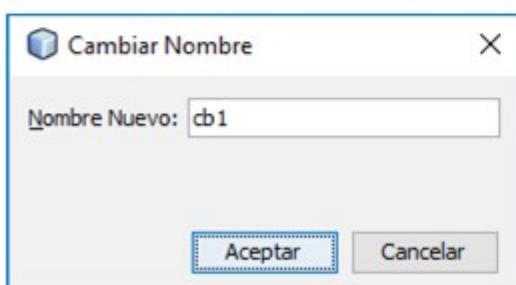
- Arrastre un **JComboBox** hasta la ventana según la posición indicada:



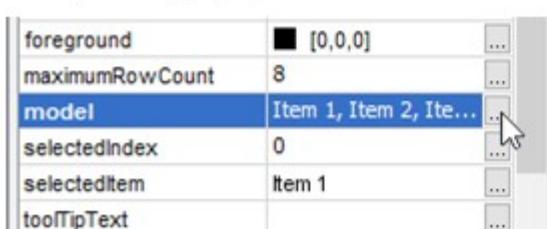
- Haga click derecho sobre el **JComboBox** y seleccione la opción “**cambiar nombre de variable ...**”



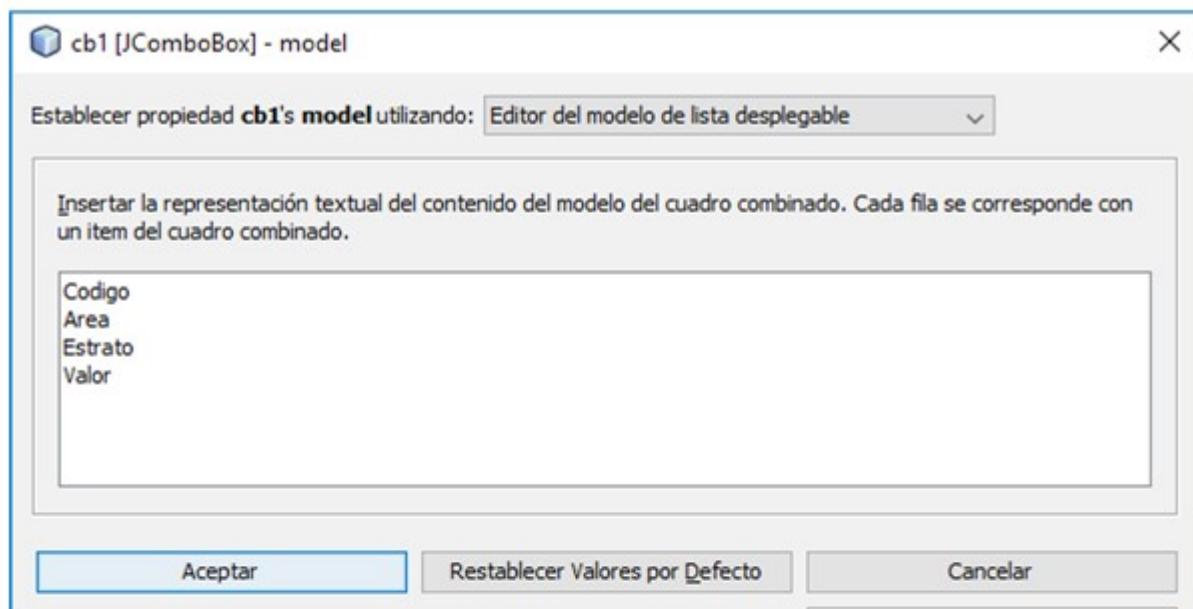
- Asigne **cb1** como nombre de instancia y haga click en el botón **Aceptar**.



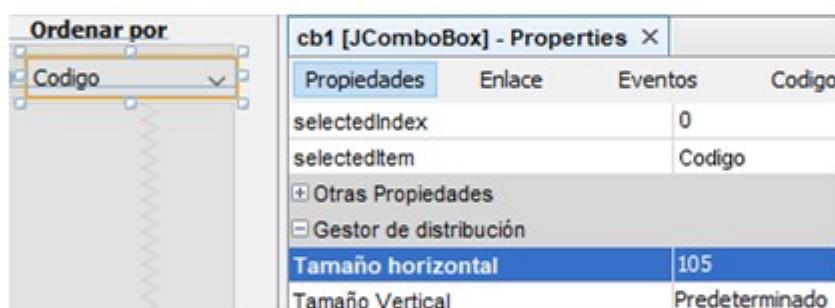
- En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



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



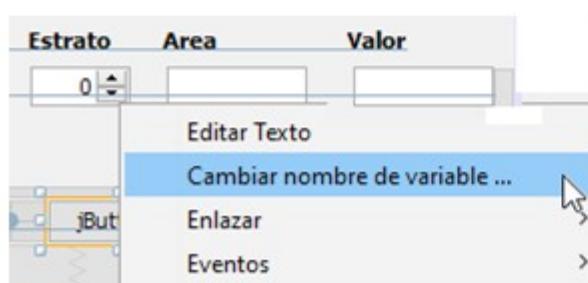
- En el inspector de propiedades en el nodo “**Gestor de distribución**”, ponga 105 como valor para la propiedad **Tamaño horizontal**:



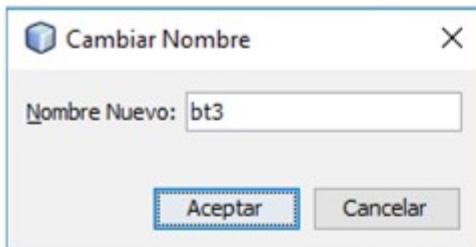
- Arrastre hasta la ventana un **JButton** en el punto señalado:



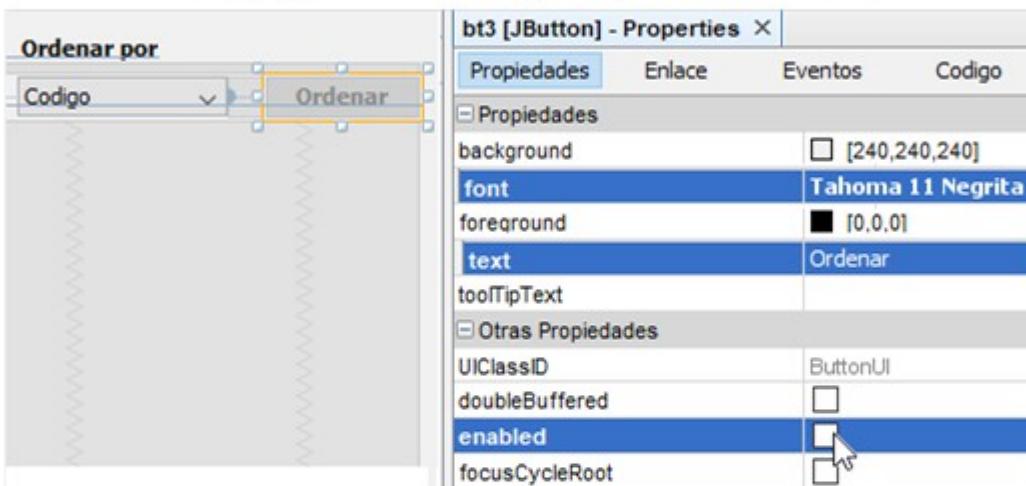
- Haga click derecho sobre el **JButton** y seleccione la opción “**cambiar nombre de variable ...**”



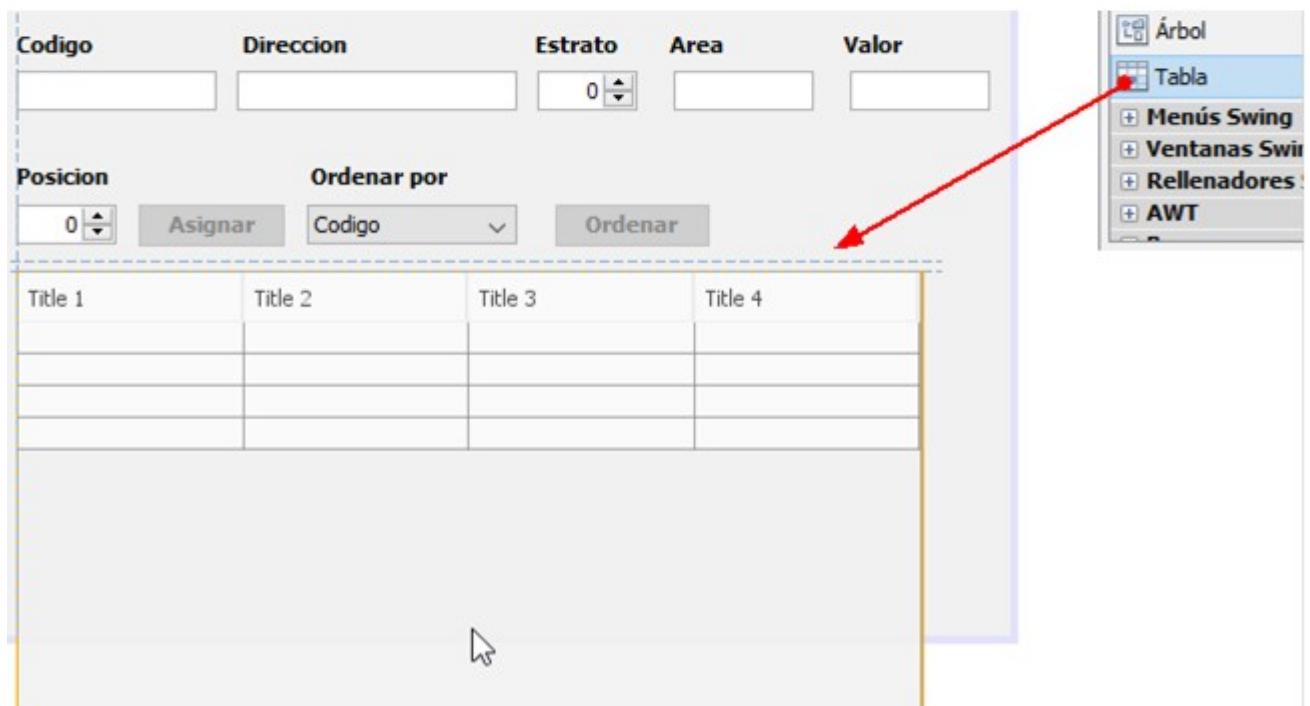
- Asigne **bt3** como nombre para el **JButton** y haga click en el botón **Aceptar**.



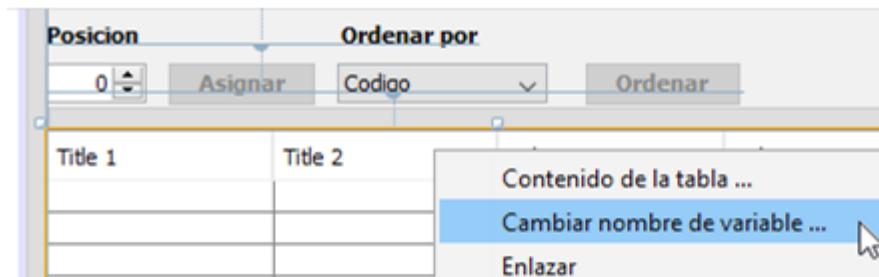
- En el inspector de propiedades, ponga **Asignar** en la propiedad **text**, en la propiedad **font** asigne texto en negrilla y en el nodo “**Otras propiedades**” desmarque la casilla a la derecha de la propiedad **enabled**, para que el botón inicialmente este deshabilitado.



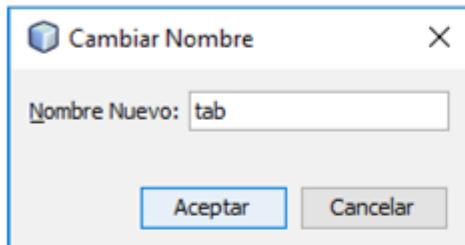
- Arrastre hasta el punto señalado un control **JTable**



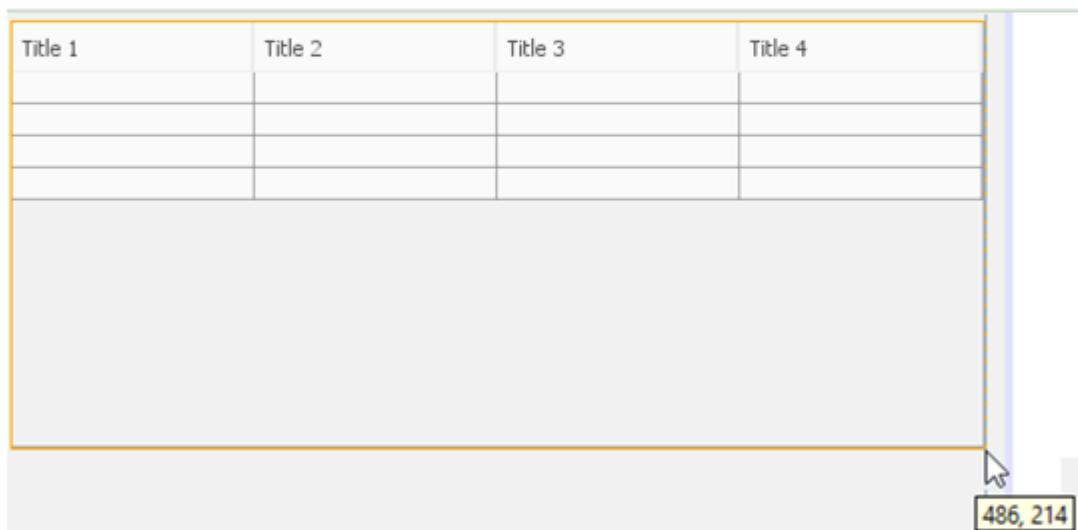
- ➁ Haga click dentro de la tabla para seleccionarla, luego haga click derecho sobre el **JTable** y seleccione la opción “**cambiar nombre de variable ...**”



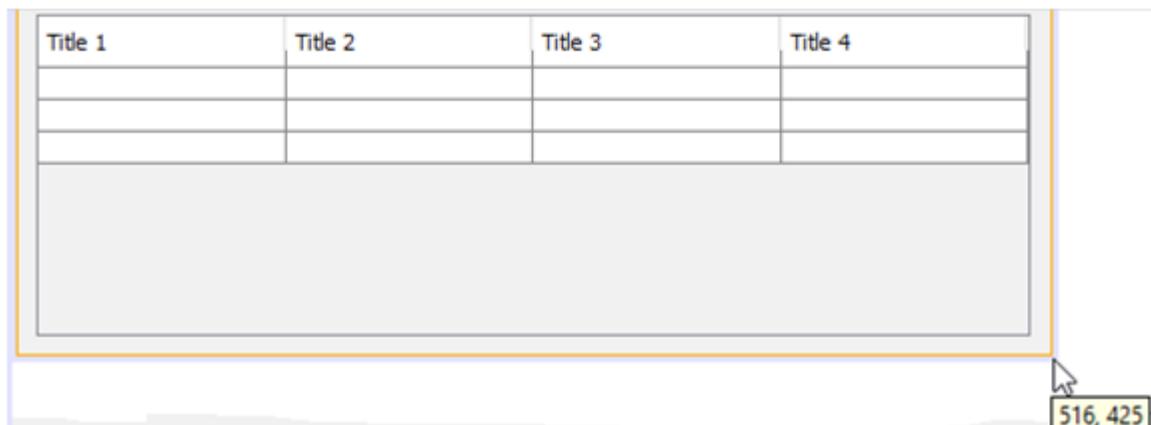
- ➂ Asigne **tab** como nombre para el **JTable** y haga click en el botón **Aceptar**.



- ➃ Tome la tabla por el nodo inferior derecho y redimensionela para que tenga el tamaño indicado en el cuadro amarillo de abajo (486,214).



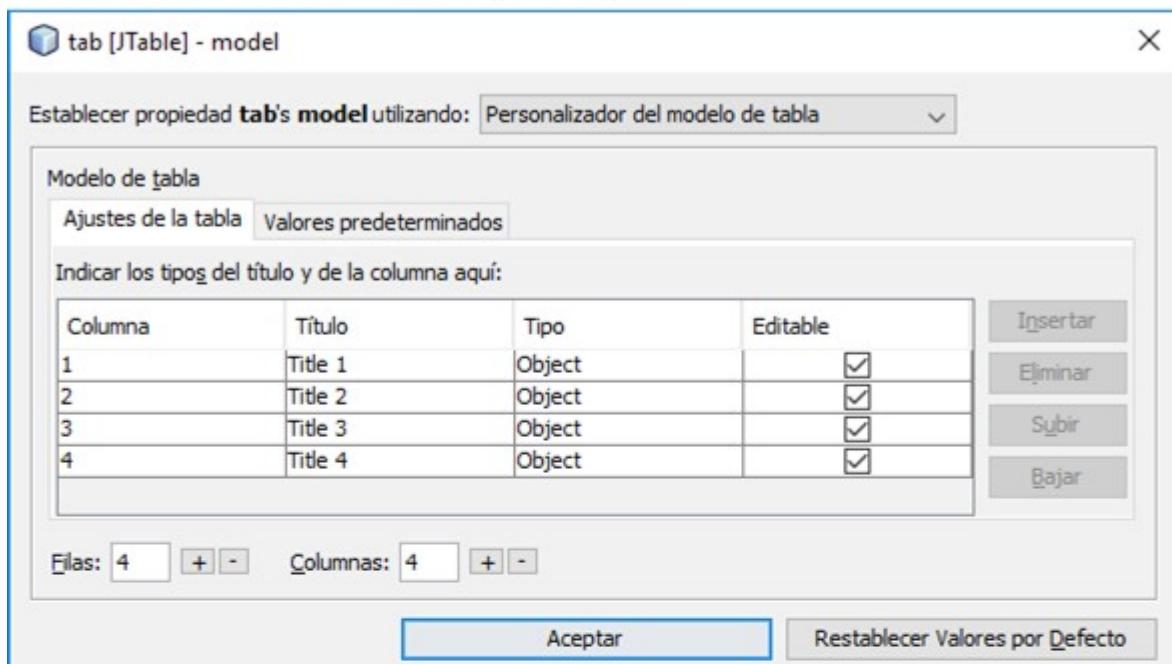
- ➄ De igual manera, tome el **JFrame** por el nodo inferior derecho y redimensionela para que tenga el tamaño adecuado (516,425).



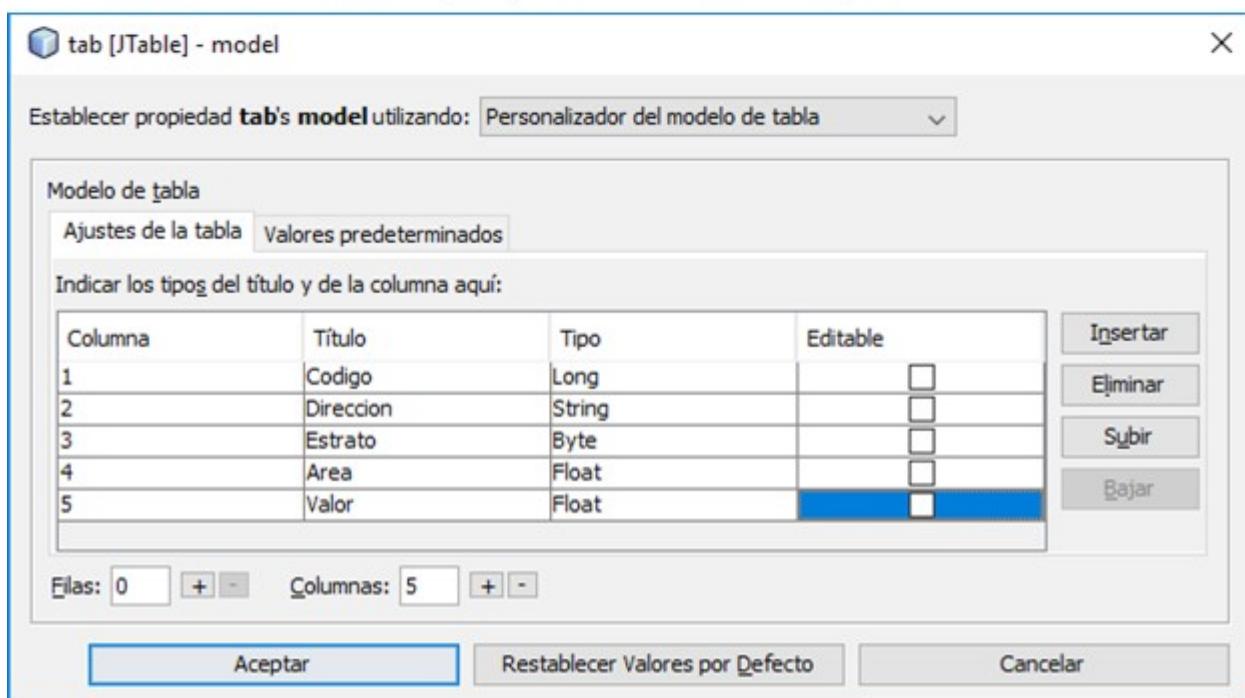
- En el inspector de propiedades, busque la propiedad **model** y haga click en el botón de tres puntos (...) que está a su derecha.



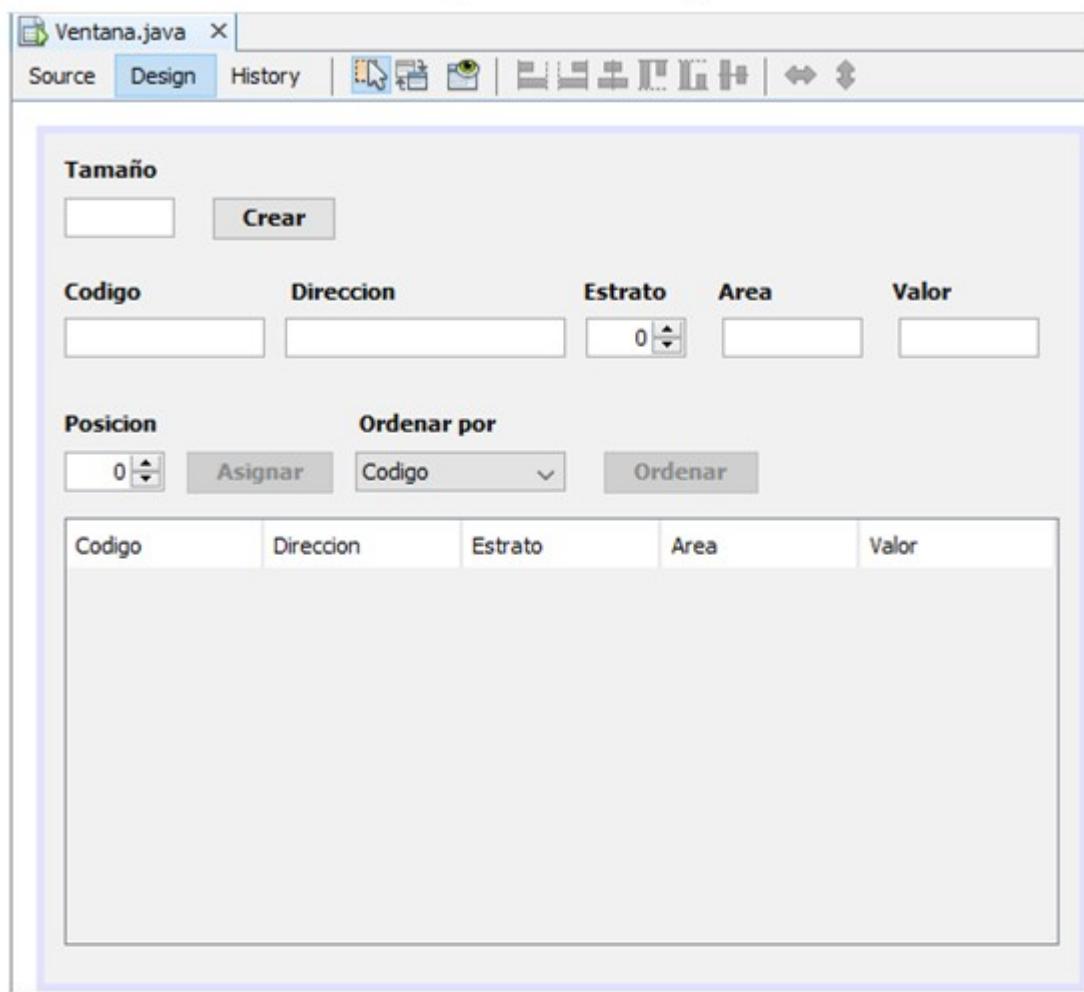
- Con ello vera una ventana como la siguiente, en la que se configuran la cantidad de filas y columnas, los títulos y tipos de columnas, además de a cuales columnas se les puede modificar su contenido en tiempo de ejecución:



- Ahora en la entrada **Filas** ponga 0 y **Columnas** ponga 5; escriba los títulos indicados en la imagen de abajo; en la columna tipo haga click sobre **Object** y escoja el tipo mostrado abajo. Luego desmarque las casillas **Editable** para no permitir cambiar el contenido de las columnas y haga click en el botón **Aceptar**.



- La ventana debería tener un aspecto como el siguiente:



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 TVecVivienda VecViv;
    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();
    VecViv=new TVecVivienda();
    TabMod=(DefaultTableModel)tab.getModel();
    SpMod=(SpinnerNumberModel)sp2.getModel();
}

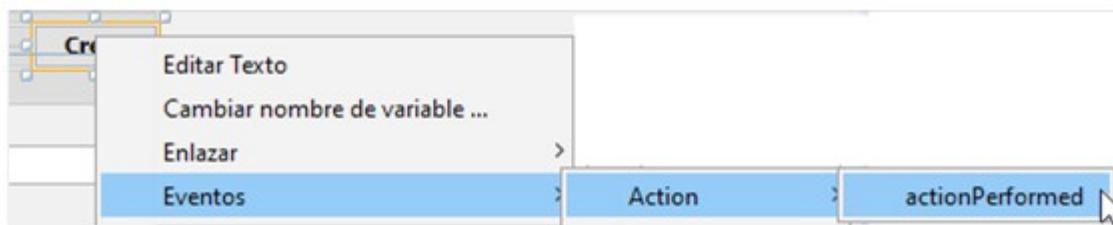
private void Mostrar() {
    int i;
    TVivienda Viv;
    for(i=0;i<VecViv.getTam();i++){
        Viv=VecViv.getVec(i);
        TabMod.setValueAt(Viv.getCodigo(),i,0);
        TabMod.setValueAt(Viv.getDireccion(),i,1);
        TabMod.setValueAt(Viv.getEstrato(),i,2);
        TabMod.setValueAt(Viv.getArea(),i,3);
        TabMod.setValueAt(Viv.getValor(),i,4);
    }
}

private void VerMensaje(String Texto){
    JOptionPane.showMessageDialog(null,Texto);
}

private void Limpiar(){
    tf2.setText("");
    tf3.setText("");
    tf4.setText("");
    tf5.setText("");
    tf2.grabFocus();
}

private void LLenar(TVivienda Viv){
    Viv.setCodigo(Long.parseLong(tf2.getText()));
    Viv.setDireccion(tf3.getText());
    Viv.setEstrato((byte)sp1.getValue());
    Viv.setArea(Float.parseFloat(tf4.getText()));
    Viv.setValor(Float.parseFloat(tf5.getText()));
}
```

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



- Borre el comentario que dice: `//TODO add your handling code here`, e implemente el evento así (código del cuadro rojo):

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

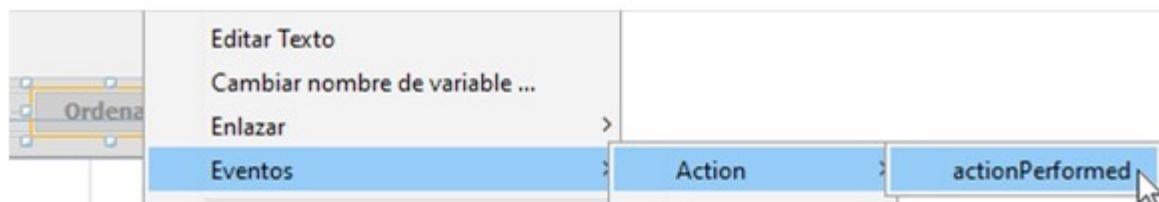
- Vuelva nuevamente a la vista diseño, haga click derecho sobre el botón **Asignar** y seleccione las opciones **Eventos + Action + actionPerformed**:



- 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(VecViv.getVec(pos));
    if(pos+1<VecViv.getTam()){
        SpMod.setValue(pos+1);
    }
    Limpiar();
    Mostrar();
}
```

- Nuevamente en la vista diseño haga click derecho sobre el botón **Ordenar** y seleccione las opciones **Eventos + Action + actionPerformed**:

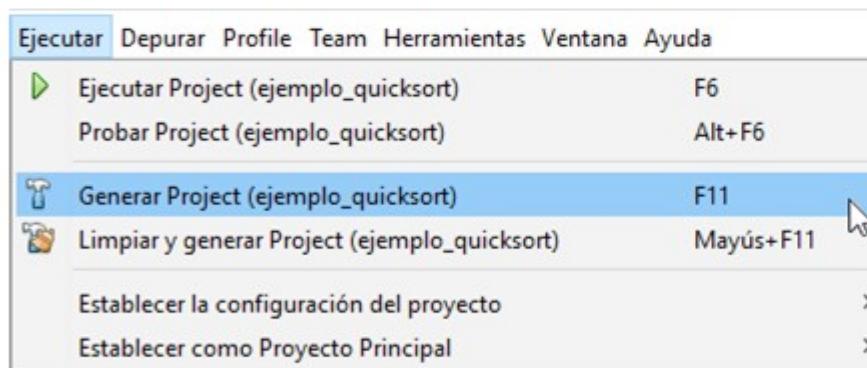


- El código para implementar el evento de este último botón es como sigue:

```
private void bt3ActionPerformed(java.awt.event.ActionEvent evt) {
    switch(cb1.getSelectedIndex()) {
        case 0:VecViv.OrdenaCodigo();break;
        case 1:VecViv.OrdenaArea();break;
        case 2:VecViv.OrdenaEstrato();break;
        case 3:VecViv.OrdenaValor();break;
    }
    Mostrar();
}
```

h. Compilación y ejecución del programa.

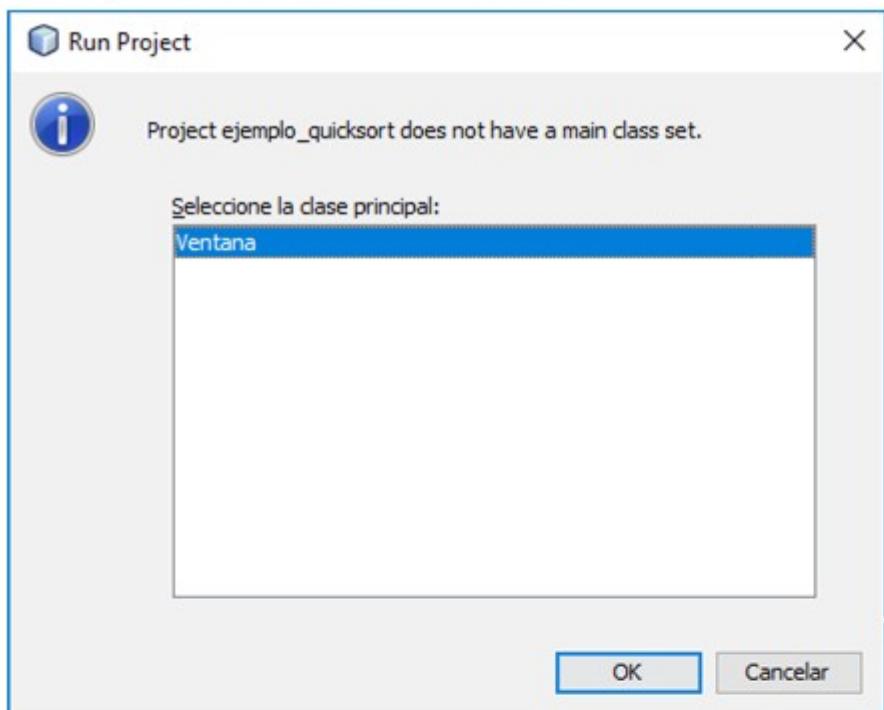
- Para compilar el programa vaya por la opción “*Ejecutar*” + “*Generar Project*” del menú principal o pulse la tecla F11.



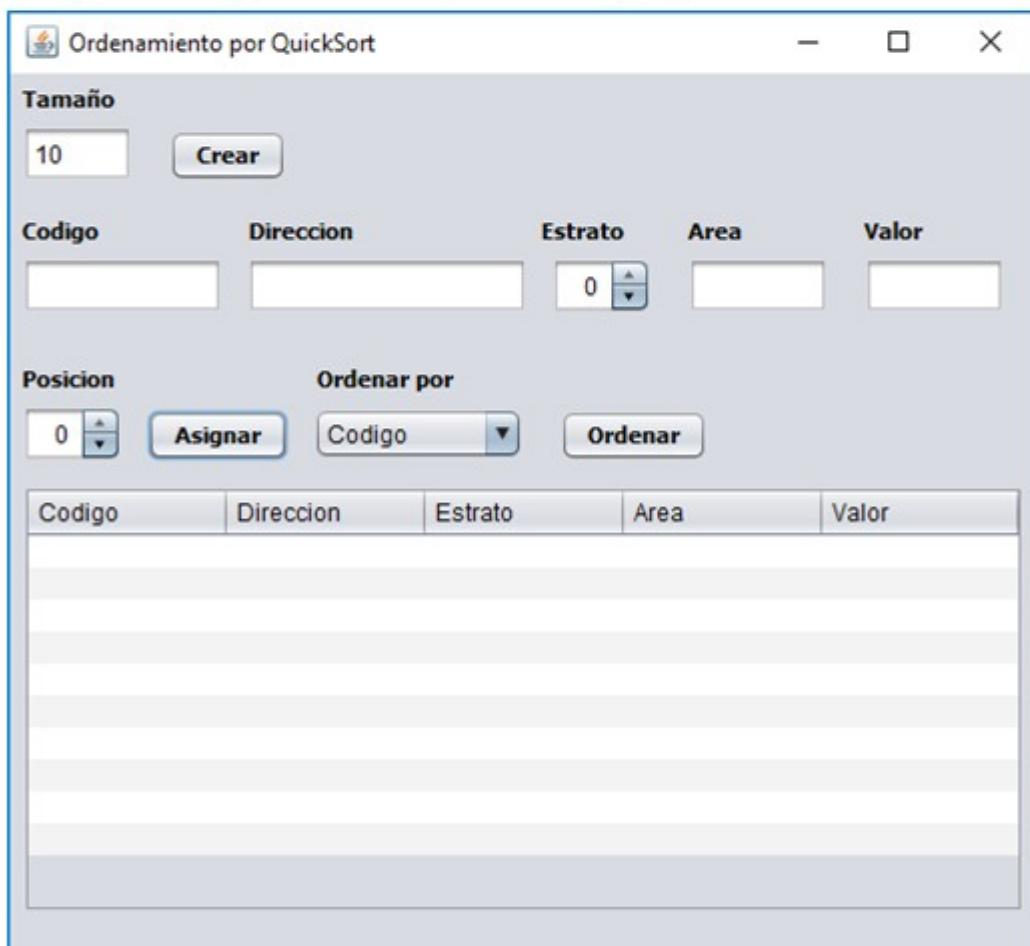
- Corrija los errores según los mensajes indicados por el compilador comparando con el código fuente anterior; luego puede correr el programa con la opción “*Ejecutar*” + “*Ejecutar Project*” del menú principal, o pulsando la tecla F6:



- 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. Según el código anterior por que no fue necesario instanciar un **SpinnerNumberModel** para el modelo del **JSpinner** sp1 (el de estrato)
2. Diseñar en UML e implementar en Java una clase con su respectiva aplicación de ventana, para un arreglo de cadenas de caracteres y que permita ordenarlo tanto ascendente como descendente por el método *QuickSort*.
3. Diseñar en UML e implementar en Java una clase con su respectiva aplicación de ventana, para un arreglo de cadenas de caracteres de tal suerte que sea ordenado descendente por el método *QuickSort* considerando comparar solo la ultima vocal encontrada de derecha a izquierda en cada cadena del vector tomando en cuenta los siguientes criterios:
 - ◆ Toda cadena que tenga una vocal minúscula siempre será menor que una que no tenga vocal.
 - ◆ Toda cadena que tenga una vocal mayúscula siempre será mayor que una que no tenga vocal.
 - ◆ Toda cadena con una vocal minúscula siempre es menor que una cadena que tenga la misma vocal en mayúscula.
 - ◆ Si las dos cadenas tienen la misma vocal sea en minúscula o en mayúscula se compara en forma natural como lo hace el lenguaje.
 - ◆ Si dos cadenas comparadas no tienen vocal se consideran iguales.
4. Diseñar en UML e implementar en Java una clase con su correspondiente aplicación de ventana, para un arreglo de cadenas de caracteres de modo que ordene los caracteres individuales de cada cadena del arreglo descendente por burbuja, además de ordenar el arreglo en general ascendente por *QuickSort* usando solo el último carácter de cada cadena.
5. Implemente en Java cada uno de los métodos de ordenamientos explicados en clase, para ordenar un arreglo de números reales, de modo que cada método sea sobrescrito por una clase hija en particular; es decir, usando herencia, clases abstractas, métodos abstractos y polimorfismo para cada uno de los métodos; es importante tener en cuenta que cada clase hija (una distinta para cada método), deben ordenar la misma instancia del arreglo declarado en la clase padre.