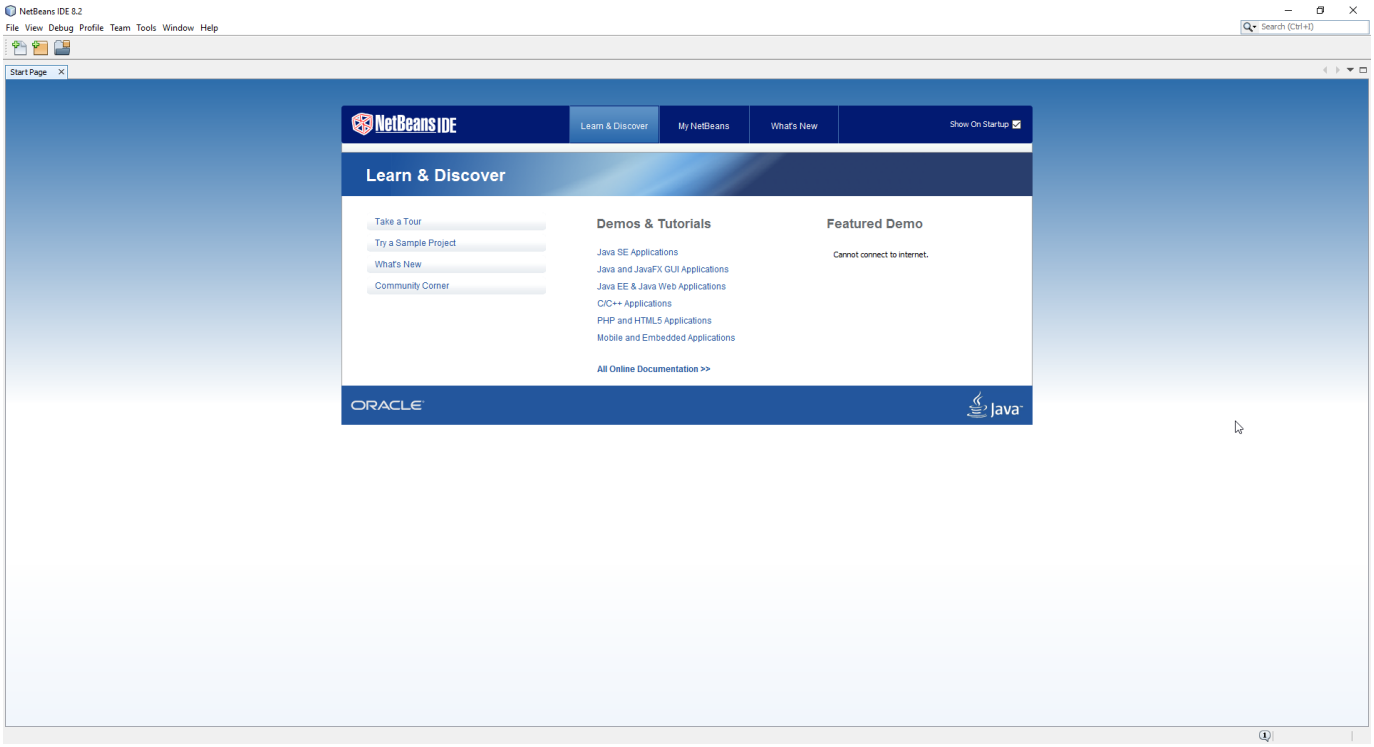
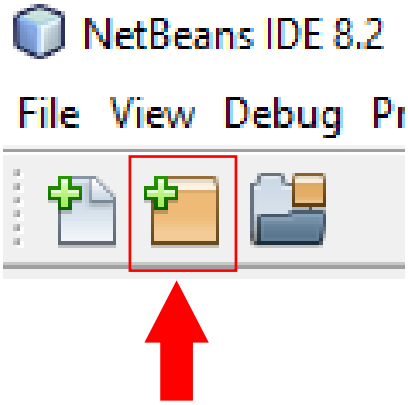
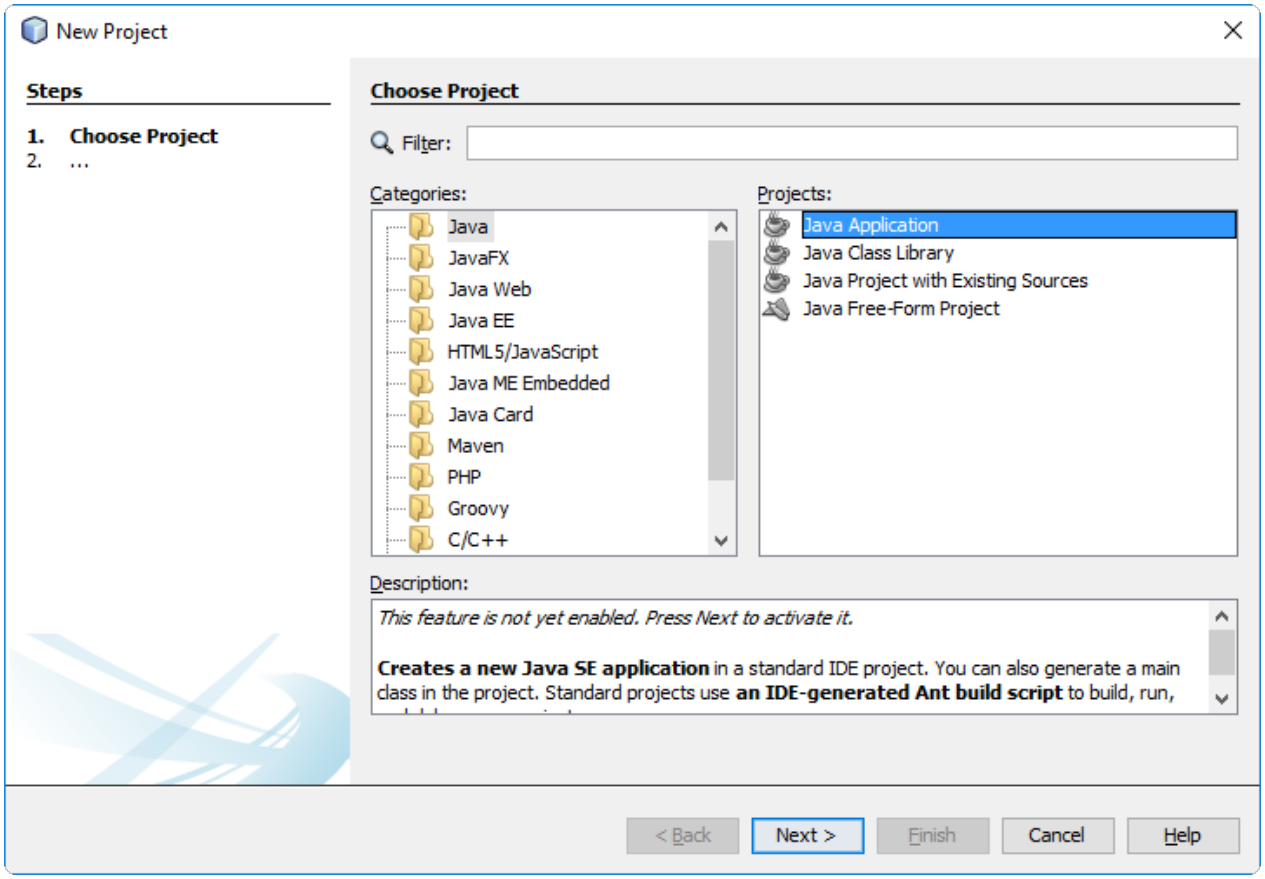
TP 1: Conversor Pulgadas/Centímetros

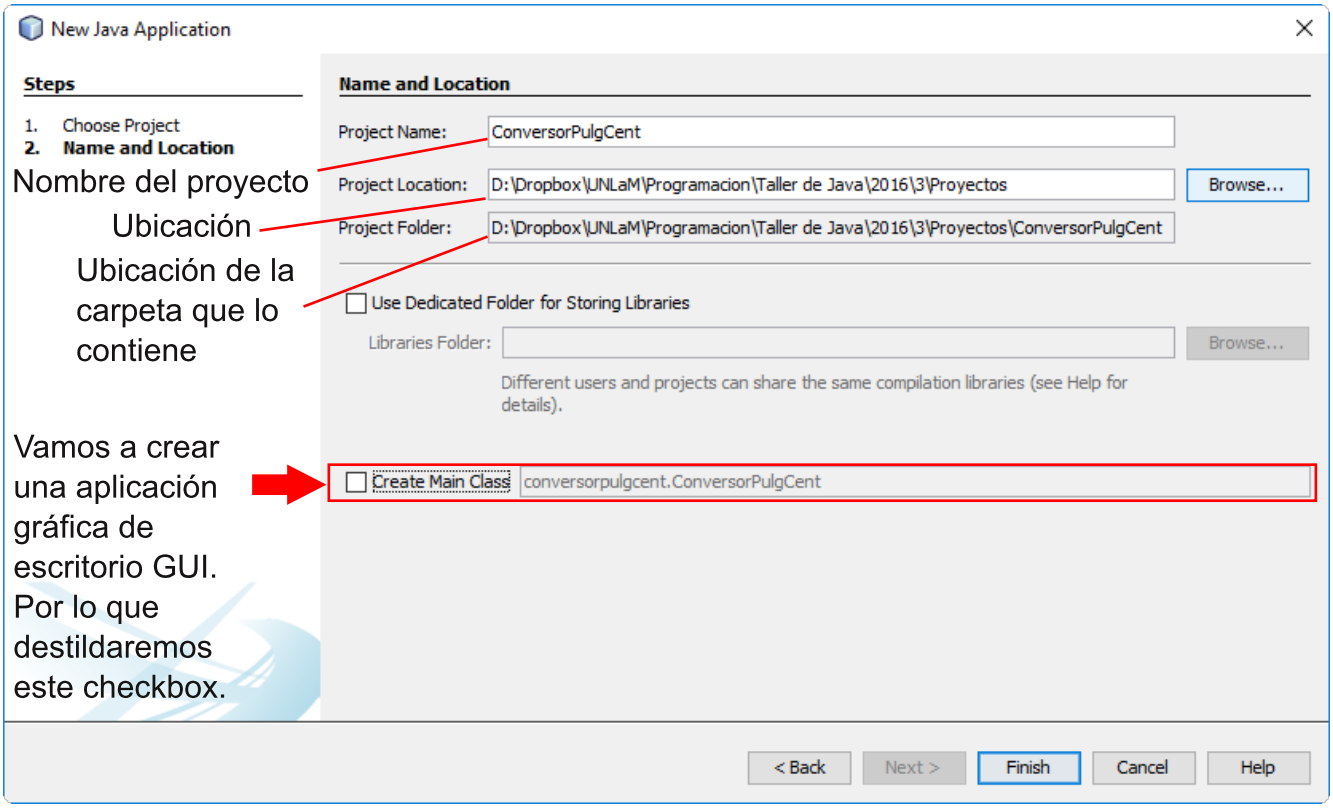




Con este botón creamos un nuevo proyecto.

Seleccionamos la Categoría Java, Proyecto Java Application. Este tipo de proyectos es para crear aplicaciones ejecutables (Que ejecutan en una máquina virtual).
El proyecto Java Class Library, se utiliza para crear bibliotecas.





Nombre del proyecto

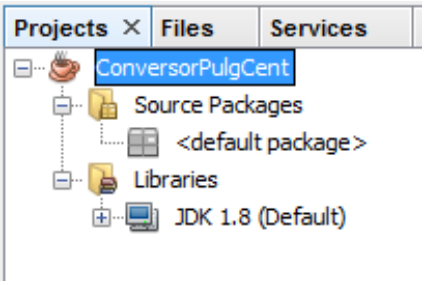
Ubicación

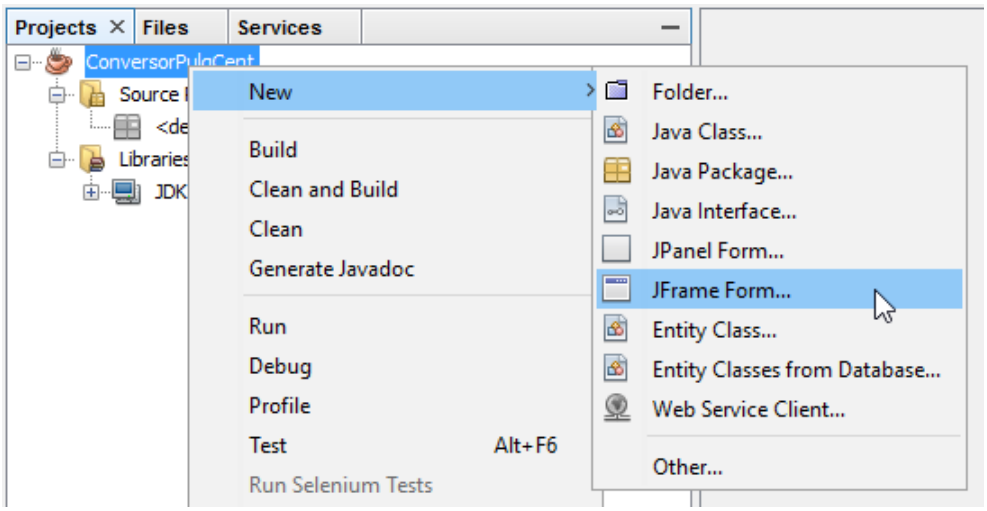
Ubicación de la carpeta que lo contiene

Vamos a crear una aplicación gráfica de escritorio GUI. Por lo que destildaremos este checkbox.

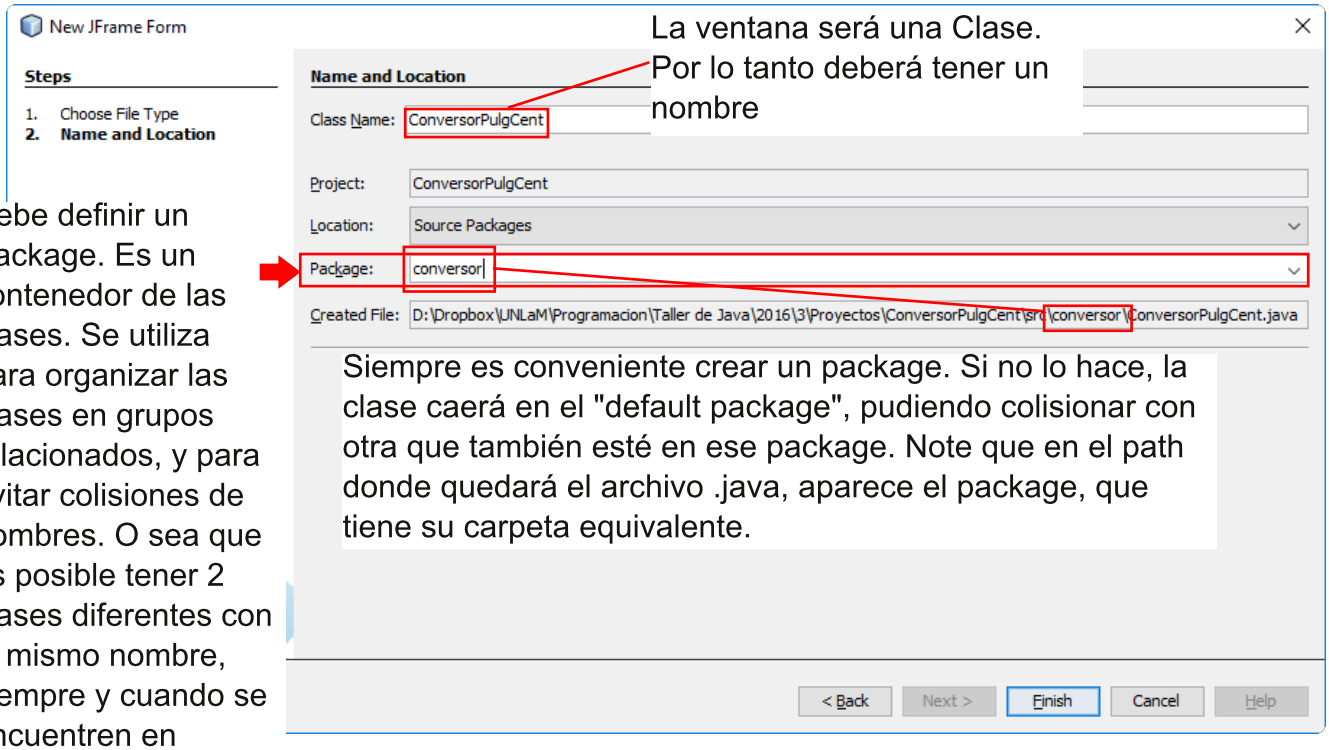
Ese checkbox nos permite crear una Main Class. Esta Main Class la crearemos después, en forma automática, al crear la ventana de la aplicación.

En la pestaña Projects vemos el árbol del proyecto.
La carpeta Source Packages contendrá los archivos de código fuente (.java).
La carpeta Libraries contiene las bibliotecas con las que trabajará el proyecto. De momento sólo la JDK, que se incluye por defecto.





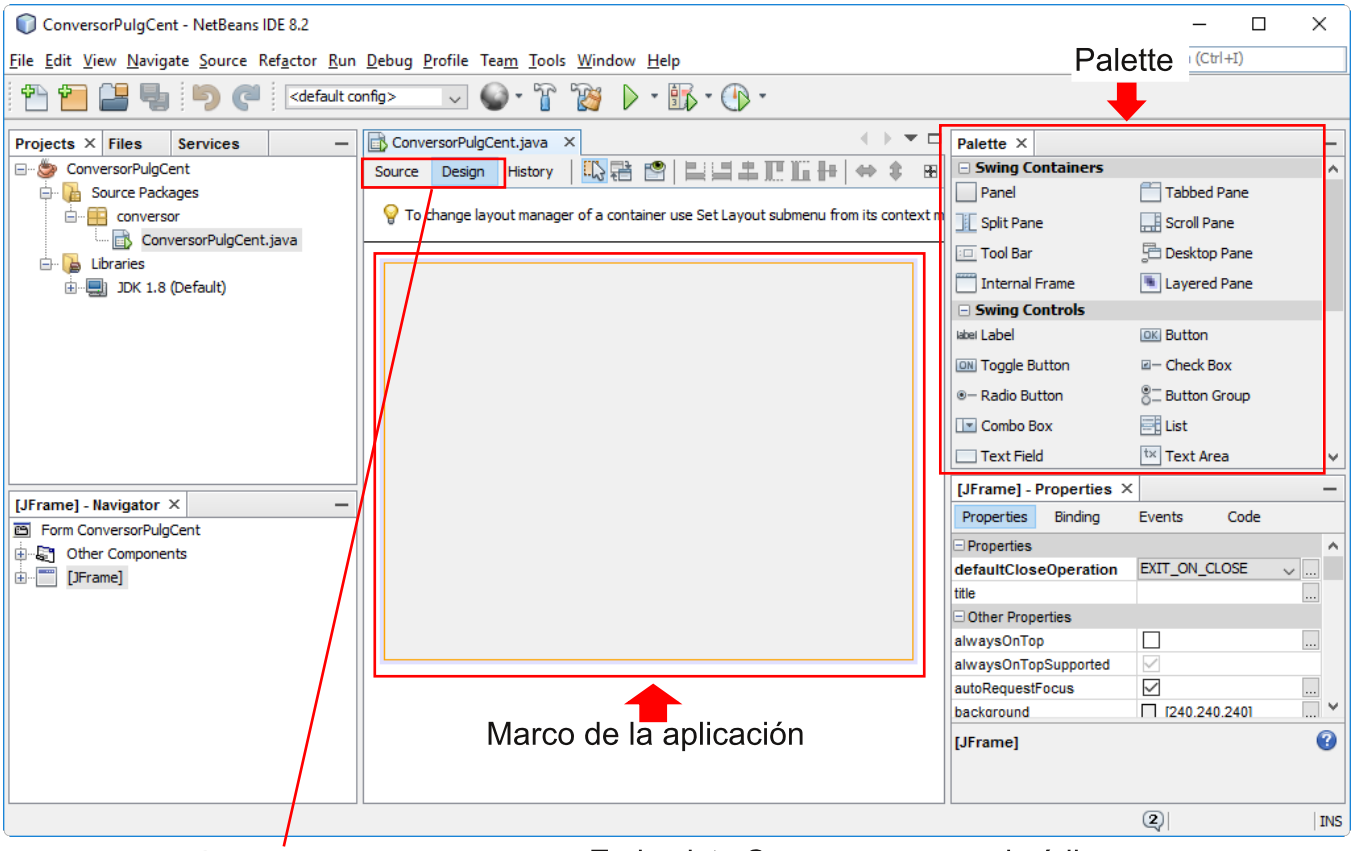
Procedemos a crear la ventana de la aplicación.
Hacemos click derecho en el nombre del proyecto, seleccionamos New, JFrame Form...



La ventana será una Clase. Por lo tanto deberá tener un nombre

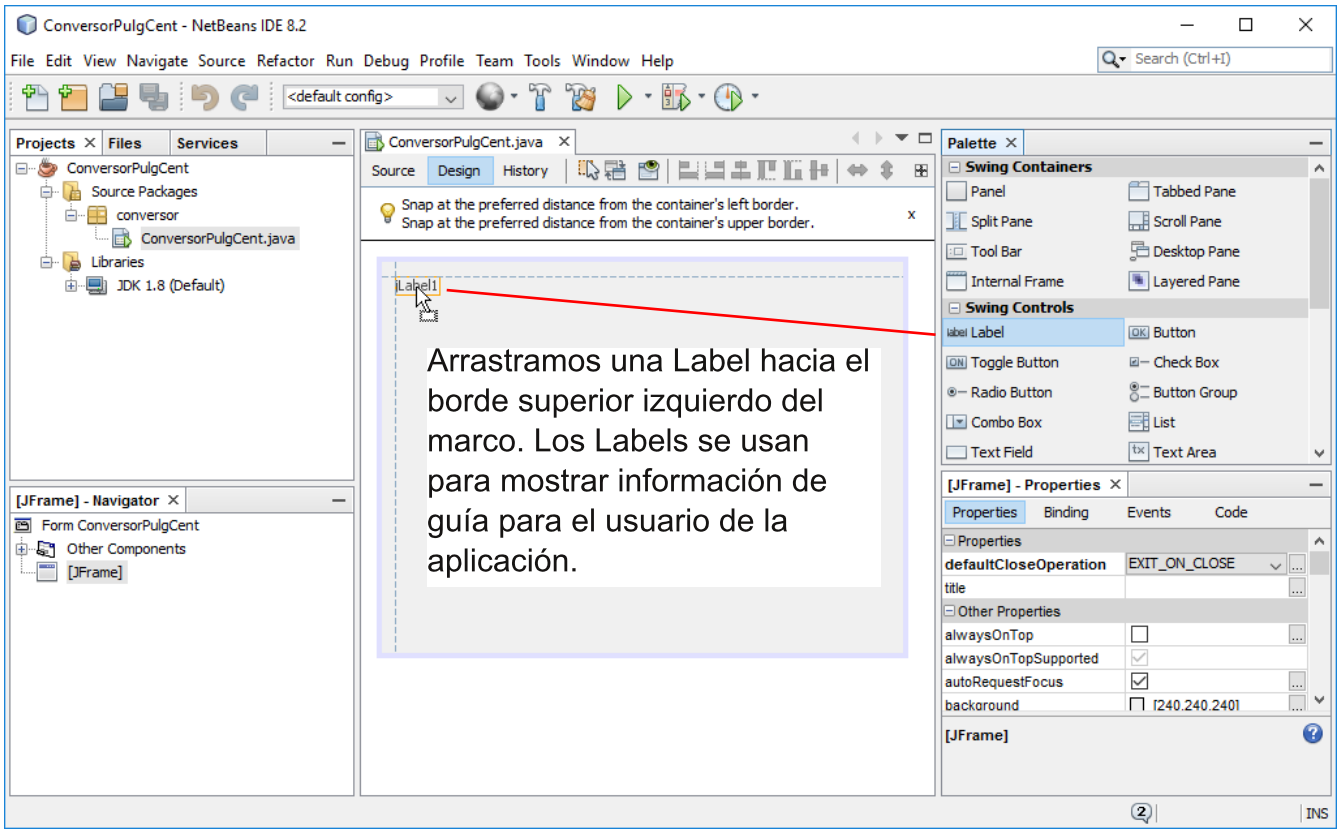
Debe definir un Package. Es un contenedor de las clases. Se utiliza para organizar las clases en grupos relacionados, y para evitar colisiones de nombres. O sea que es posible tener 2 clases diferentes con el mismo nombre, siempre y cuando se encuentren en diferentes packages.

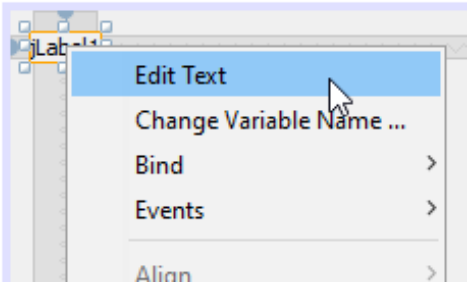
Siempre es conveniente crear un package. Si no lo hace, la clase caerá en el "default package", pudiendo colisionar con otra que también esté en ese package. Note que en el path donde quedará el archivo .java, aparece el package, que tiene su carpeta equivalente.



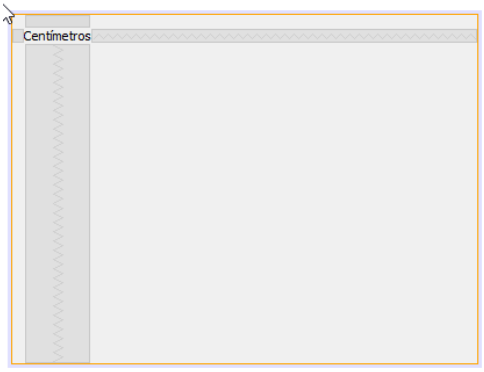
Hay 2 vistas: Source y Design. La vista en la que nos encontramos es Design, que nos permite crear la ventana en forma visual. Podremos arrastrar los controles que están en la Palette, y ubicarlos en el Marco de la aplicación.

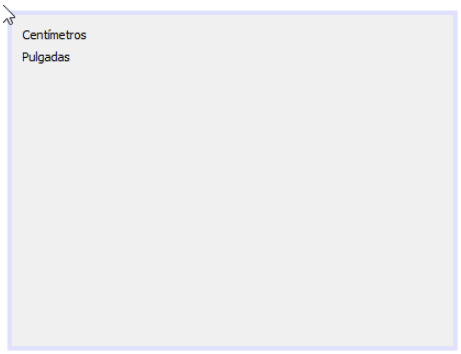
En la vista Source veremos el código que se genera cuando arrastramos, redimensionamos, renombramos los controles. En esta vista codificaremos la lógica de la aplicación.



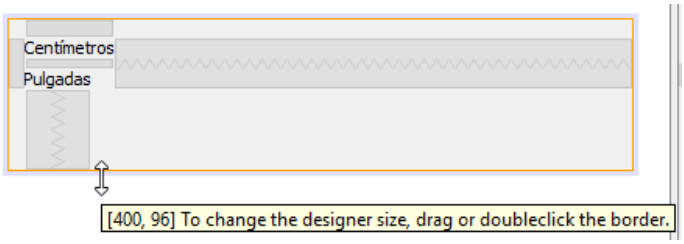


Editamos el texto del label con click derecho, Edit Text.

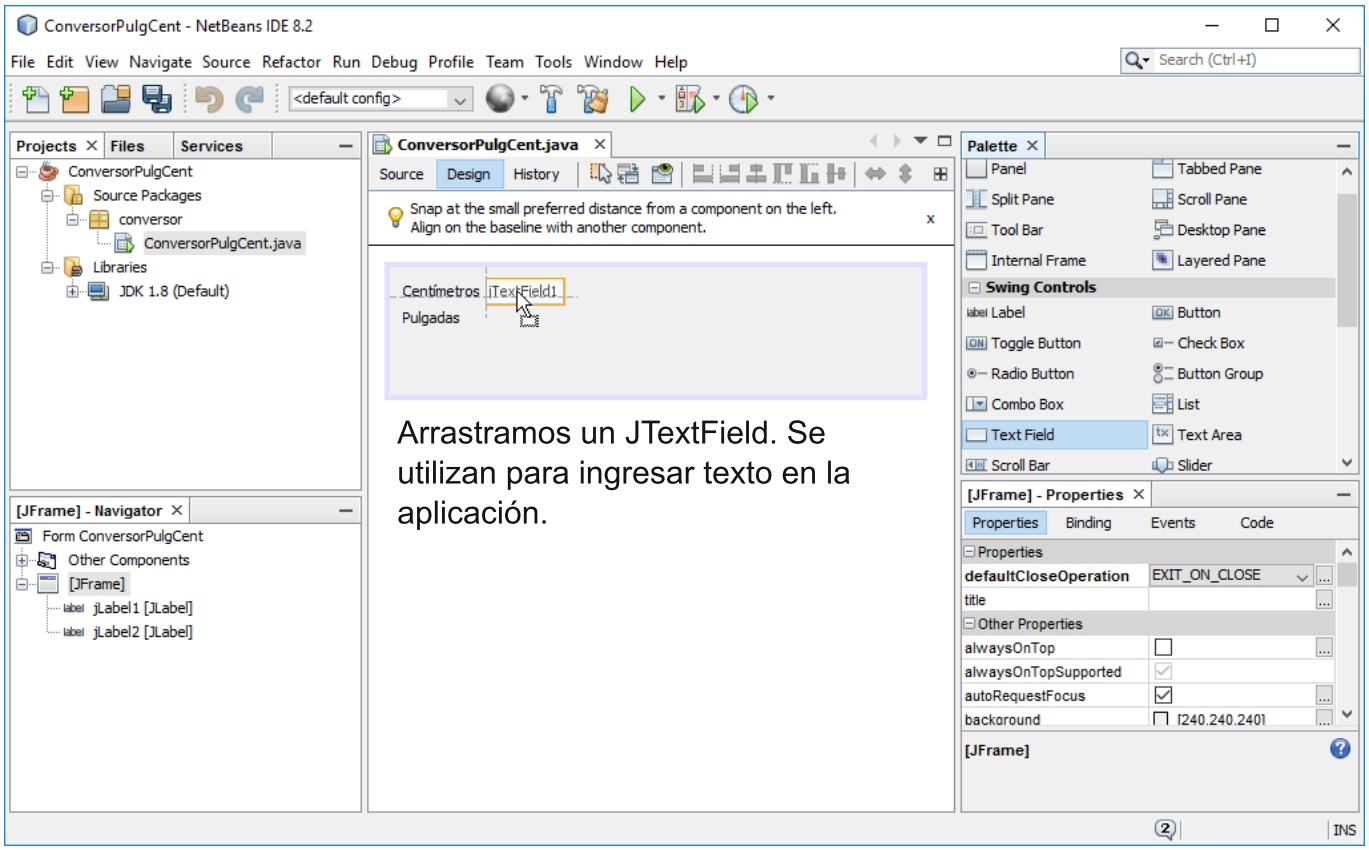




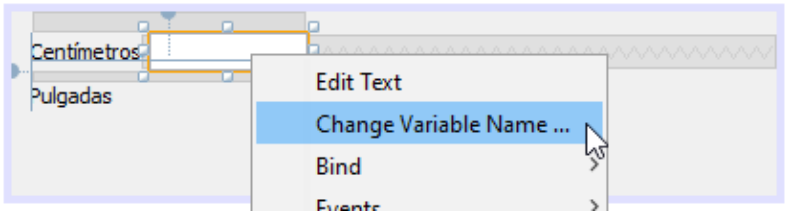
Arrastramos otra label y le ponemos texto Pulgadas.



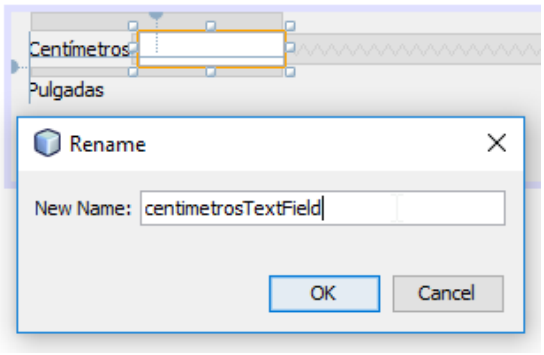
Observe que puede redimensionar la ventana y cualquier control, arrastrando el borde hacia el tamaño deseado.



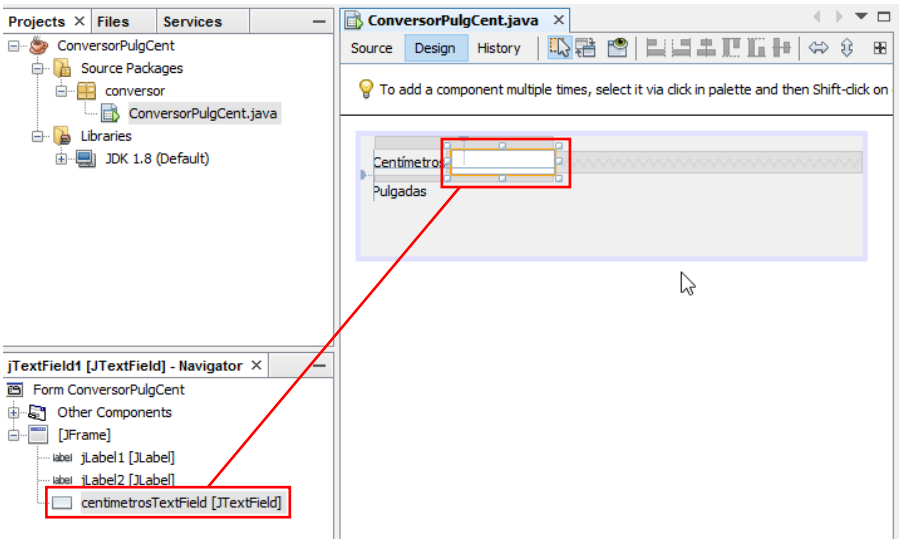
Arrastramos un JTextField. Se utilizan para ingresar texto en la aplicación.



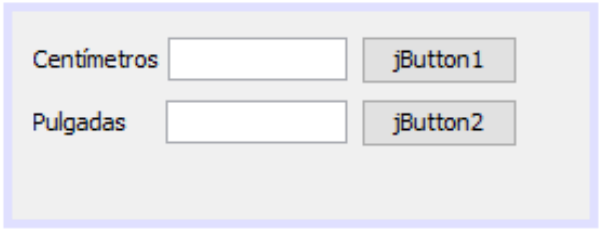
Borramos el texto que viene por defecto mediante Edit Text. Cambiaremos el nombre de la variable. Los controles son objetos y los objetos son variables. Estas variables serán referenciadas en el código que escribiremos para que funcione la aplicación. Por lo tanto, los nombres que le pongamos a las variables deberán elegirse con cuidado para evitar confusiones en el código.



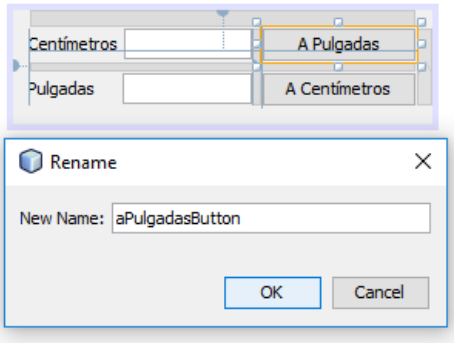
Como el usuario ingresará aquí los centímetros a convertir, y el control es un TextField, lo llamaremos centimetrosTextField.
Observe la convención utilizada para nombrar a las variables: La primer palabra comienza en minúscula, y el resto de las palabras comienzan en mayúscula, sin guiones entre palabras. La única excepción son los nombres de las clases, que siempre empiezan con mayúscula. A esta notación se la conoce como Camel Case, y será la utilizada en el TCP.



Debajo de la pestaña Proyects, se encuentra el Navigator, que se utiliza para ubicar los controles de la ventana cuando no es sencillo ubicarlos en el marco, o cuando no están dentro del mismo.

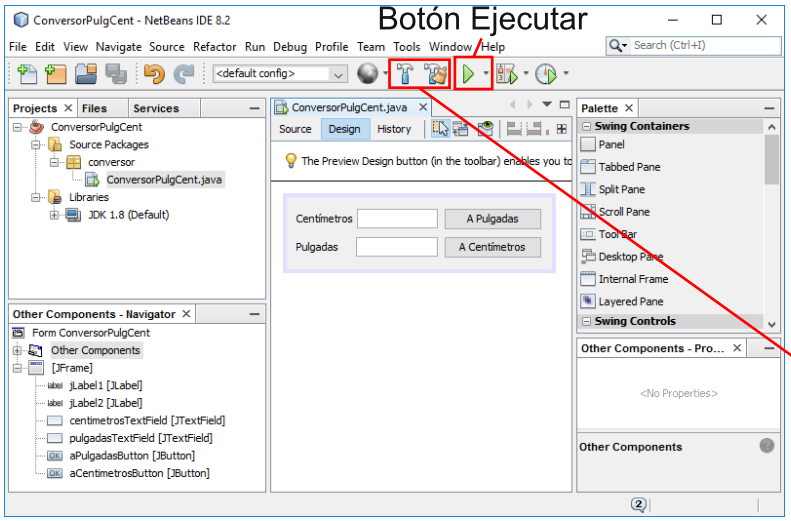


Agregamos otro TextField y 2 Buttons. Los botones nos permiten ejecutar la lógica de la aplicación. Cada vez que el usuario presione un botón, se ejecutará una función.



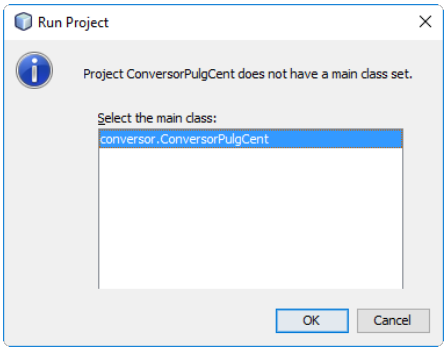
Le cambiamos el texto a los botones a través de Edit Text a "A Pulgadas" y "A Centímetros". También cambiamos el nombre de las variables a aPulgadasButton y aCentimetrosButton.

Con esto terminamos el diseño gráfico de la ventana. La aplicación puede ejecutar, gracias a que Netbeans generó el código, a mientras creábamos la ventana en forma visual.



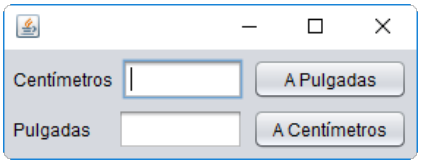
Antes de ejecutar la aplicación hay que compilarla, a menos que esté activa la opción "Compile on Save", que realiza la compilación ni bien se guarda el archivo.

Estos botones son para compilar y limpiar y compilar respectivamente. Este último borra los archivos generados tras la última compilación y recompila todos los archivos del proyecto. También es el responsable de generar el archivo .jar, que es el binario ejecutable de la aplicación, necesario para ejecutar la misma fuera de Netbeans.

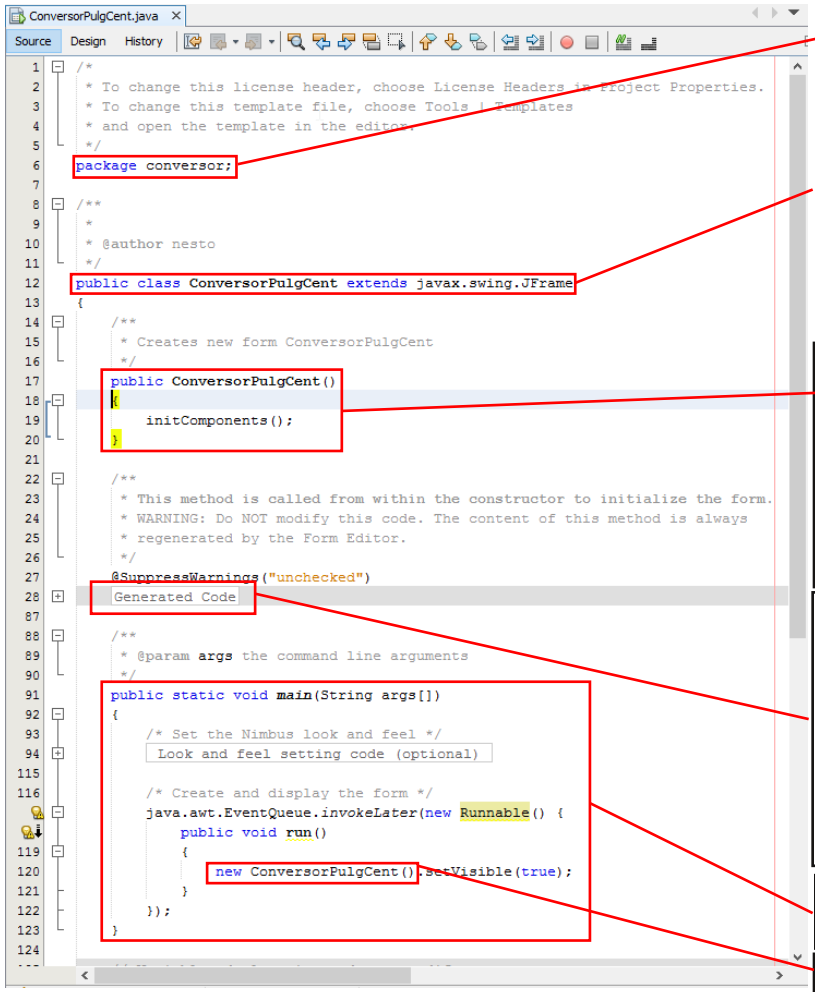


La primera vez que ejecute la aplicación, le pedirá que le indique cuál es la Main Class. Esta clase es la que tiene la función main, que es el punto de entrada del programa. Sólo tenemos una, así que le damos OK.

La aplicación en funcionamiento. Por supuesto que al presionar los botones no sucederá nada. Para que suceda, habrá que asociar el "Evento" de presionar el botón a una "Función" que escribiremos en breve.



Antes de eso, vamos a familiarizarnos con la vista Source:



```
1   /*
2    * To change this license header, choose License Headers in Project Properties.
3    * To change this template file, choose Tools | Templates
4    * and open the template in the editor.
5    */
6   package conversor;
7
8   /**
9    *
10   * @author nesto
11   */
12  public class ConversorPulgCent extends javax.swing.JFrame
13  {
14      /**
15       * Creates new form ConversorPulgCent
16       */
17      public ConversorPulgCent()
18      {
19          initComponents();
20      }
21
22      /**
23       * This method is called from within the constructor to initialize the form.
24       * WARNING: Do NOT modify this code. The content of this method is always
25       * regenerated by the Form Editor.
26       */
27      @SuppressWarnings("unchecked")
28      Generated Code
87
88      /**
89       * @param args the command line arguments
90       */
91      public static void main(String args[])
92      {
93          /* Set the Nimbus look and feel */
94          Look and feel setting code (optional)
115
116         /* Create and display the form */
            java.awt.EventQueue.invokeLater(new Runnable() {
119             public void run()
120             {
121                 new ConversorPulgCent().setVisible(true);
122             }
123         });
124     }
```

Esta línea indica el package en el que se encuentra la clase.

Esta es la definición de la clase ConversorPulgCent. La ventana de la aplicación es un objeto de esta clase. En esta clase está el código que permite generar la ventana. El contenido de la clase va entre llaves.

Esta función que se llama igual que la clase, se denomina "Constructor", y es la función que "construye" al objeto. Se ejecuta al utilizar la palabra "new". Avanzaremos sobre esto en los próximos TPs.

Oculto en Generated Code se encuentra la función initComponents() que se invoca en el constructor, que es la encargada de crear los controles, ubicarlos en la ventana, colocar los textos, darles un tamaño, etc.

La función main. Es la primera en ejecutar de toda la aplicación.

Aquí se está construyendo la ventana.

```
160
161     // Variables declaration - do not modify
162     private javax.swing.JButton aCentimetrosButton;
163     private javax.swing.JButton aPulgadasButton;
164     private javax.swing.JTextField centimetrosTextField;
165     private javax.swing.JLabel jLabel1;
166     private javax.swing.JLabel jLabel2;
167     private javax.swing.JTextField pulgadasTextField;
168     // End of variables declaration
169 }
```

Estos son los "Atributos" de la clase. Son las variables que componen a los objetos. Cada objeto que se cree, tendrá las variables aquí definidas. En este caso, los atributos son los controles de la ventana. Ud. puede agregar atributos según sea necesario.

La función initComponents.

Aquí se crean los controles.

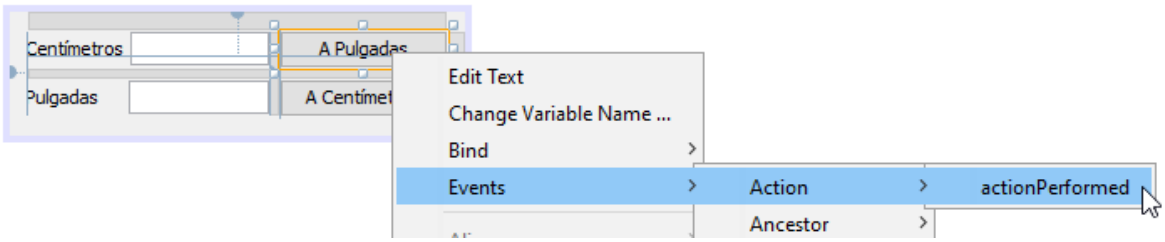
Aquí se setean los textos.

Aquí se ubican dentro del marco, se define su tamaño, y su comportamiento al redimensionar la ventana.

Vamos a crear nuestra función que ejecutará cada vez que se presione un botón:

Para esto, debemos ir a la vista Design.

Hacemos click derecho sobre el botón que queremos asociarle la función. Luego en el menú seleccionamos Events, Action, actionPerformed. Este evento se produce cada vez que el usuario presiona un botón.




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

Observe que automáticamente pasamos a la vista Source, al punto donde se generó la función. El nombre de la misma es la concatenación del nombre del botón y el nombre del evento que se asoció. Aquí irá el código que permitirá realizar la conversión de centímetros a pulgadas. Todo lo que está grisado no puede editarse, dado que fue generado en la vista Design, y es desde esa vista que podrá ser modificado o borrado.

La interacción con el usuario será la siguiente: El usuario ingresa un número en el campo "Centímetros", luego presiona "A Pulgadas", el programa hace la conversión a pulgadas, y muestra el resultado en el campo "Pulgadas". Lo opuesto sucederá con la conversión a centímetros.

Al ingresar un número a un TextField, este ingresa como texto, por lo que habrá que convertir la variable cadena a una variable flotante para poder operar aritméticamente. Luego de la operación, el flotante resultante habrá que convertirlo a cadena para poder ingresarlo en el otro TextField.

```
private void aPulgadasButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String centimetrosString = centimetrosTextField.getText();
    double centimetros = Double.valueOf(centimetrosString);
    double pulgadas = centimetros / 2.54;
    String pulgadasString = String.valueOf(pulgadas);
    pulgadasTextField.setText(pulgadasString);
}
```

Obtenemos la cadena de texto del TextField centímetros.

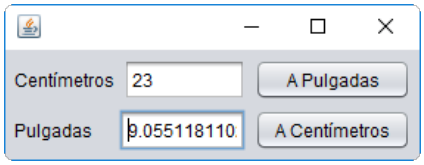
Convertimos la String en un double, mediante la función valueOf() de la clase Double.

Convertimos los centímetros en pulgadas.

Seteamos el TextField de Pulgadas con la String que convertimos.

Convertimos el double en String mediante la función valueOf() de la clase String. Siempre la función valueOf a utilizar se encuentra en la clase que es destino de la conversión. Por ejemplo, en este caso queremos convertir a String, por eso la invocamos desde esta clase.

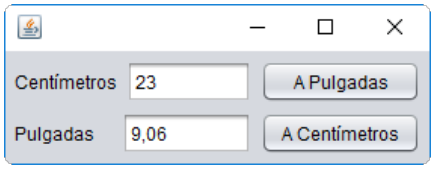
Le queda a Ud. desarrollar la lógica del otro botón.



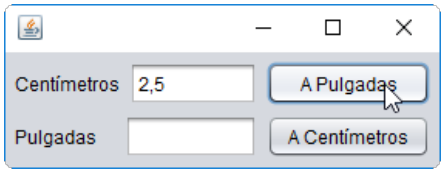
Observe la cantidad de decimales que muestra el campo pulgadas. Esto se debe a que la función String.valueOf() no puede saber con cuántos decimales quiero convertir el valor. Entonces usa una cantidad por defecto.

Para limitar la cantidad de decimales, podemos usar la función String.format(), que recibe como argumentos la cadena de formato, al estilo printf(), y la/s variable/s a convertir. Con .2 indico que quiero formatearlo con 2 decimales.

```
private void aPulgadasButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String centimetrosString = centimetrosTextField.getText();

    double centimetros = Double.valueOf(centimetrosString);

    double pulgadas = centimetros / 2.54;

    String pulgadasString = String.format("%.2f", pulgadas);

    pulgadasTextField.setText(pulgadasString);
}
```



Intento convertir el valor 2,5 a pulgadas, y no lo realiza. Sino que además aparece esto en la ventana Output:




```
Output - ConversorPulgCent (run)
run:
Exception in thread "AWT-EventQueue-0" java.lang.NumberFormatException: For input string: "2,5"
        at sun.misc.FloatingDecimal.readJavaFormatString(FloatingDecimal.java:2043)
        at sun.misc.FloatingDecimal.parseDouble(FloatingDecimal.java:110)
        at java.lang.Double.parseDouble(Double.java:538)
        at java.lang.Double.valueOf(Double.java:502)
        at conversor.ConversorPulgCent.aPulgadasButtonActionPerformed(ConversorPulgCent.java:100)
        at conversor.ConversorPulgCent.access$000(ConversorPulgCent.java:12)
        at conversor.ConversorPulgCent$1.actionPerformed(ConversorPulgCent.java:50)
        at javax.swing.AbstractButton.fireActionPerformed(AbstractButton.java:2062)
        at javax.swing.AbstractButton$Handler.actionPerformed(AbstractButton.java:2348)
        at javax.swing.DefaultButtonModel.fireActionPerformed(DefaultButtonModel.java:402)
        at javax.swing.DefaultButtonModel.setPressed(DefaultButtonModel.java:259)
        at javax.swing.plaf.basic.BasicButtonListener.mouseReleased(BasicButtonListener.java:252)
        at java.awt.Component.processMouseEvent(Component.java:6533)
        at javax.swing.JComponent.processMouseEvent(JComponent.java:3324)
        at java.awt.Component.processEvent(Component.java:6298)
        at java.awt.Container.processEvent(Container.java:2236)
        at java.awt.Component.dispatchEventImpl(Component.java:4889)
        at java.awt.Container.dispatchEventImpl(Container.java:2294)
        at java.awt.Component.dispatchEvent(Component.java:4711)
        at java.awt.LightweightDispatcher.retargetMouseEvent(Container.java:4888)
        at java.awt.LightweightDispatcher.processMouseEvent(Container.java:4525)
        at java.awt.LightweightDispatcher.dispatchEvent(Container.java:4466)
        at java.awt.Container.dispatchEventImpl(Container.java:2280)
        at java.awt.Window.dispatchEventImpl(Window.java:2746)
        at java.awt.Component.dispatchEvent(Component.java:4711)
        at java.awt.EventQueue.dispatchEventImpl(EventQueue.java:758)
        at java.awt.EventQueue.access$500(EventQueue.java:97)
        at java.awt.EventQueue$3.run(EventQueue.java:709)
        at java.awt.EventQueue$3.run(EventQueue.java:703)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:80)
        at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:90)
        at java.awt.EventQueue$4.run(EventQueue.java:731)
        at java.awt.EventQueue$4.run(EventQueue.java:729)
        at java.security.AccessController.doPrivileged(Native Method)
        at java.security.ProtectionDomain$JavaSecurityAccessImpl.doIntersectionPrivilege(ProtectionDomain.java:80)
        at java.awt.EventQueue.dispatchEvent(EventQueue.java:728)
        at java.awt.EventDispatchThread.pumpOneEventForFilters(EventDispatchThread.java:201)
        at java.awt.EventDispatchThread.pumpEventsForFilter(EventDispatchThread.java:116)
        at java.awt.EventDispatchThread.pumpEventsForHierarchy(EventDispatchThread.java:105)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:101)
        at java.awt.EventDispatchThread.pumpEvents(EventDispatchThread.java:93)
        at java.awt.EventDispatchThread.run(EventDispatchThread.java:82)
```

Se produjo una Excepción: NumberFormatException. Una excepción es un objeto que se retorna cuando ocurre una situación que la función no puede resolver. En este caso, la función Double.valueOf() no pudo convertir el 2,5 a String, entonces en lugar de retornar la String, retorna la Excepción. Esta excepción se produce porque valueOf() espera el punto para separar parte entera de la parte decimal, no la coma.

Una función puede ser invocada en múltiples lugares. Entonces, ¿Cómo saber cuál de las llamadas fue la que provocó la excepción? La salida de la ventana Output tiene la solución: Observe que una línea más abajo de donde aparece Double.valueOf(), se encuentra la función ConversorPulgCent.aPulgadasButtonActionPerformed(). Esta es la función que realizamos anteriormente, y es la que invoca a Double.valueOf(). Lo que estamos viendo en Output es la pila del programa al momento de producirse la excepción. La primer función, Double.parseDouble() es la que efectivamente "lanza" la excepción. La línea de abajo contiene a la función que la llamó, Double.valueOf(), y a esta la llamó la que está debajo, que es ConversorPulgCent.aPulgadasButtonActionPerformed(). Si hacemos click en el link, vamos a la línea correspondiente en el código.

```
private void aPulgadasButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String centimetrosString = centimetrosTextField.getText();

    double centimetros = Double.valueOf(centimetrosString);

    double pulgadas = centimetros / 2.54;

    String pulgadasString = String.format("%.2f", pulgadas);

    pulgadasTextField.setText(pulgadasString);
}
```

Al clickear el link de ConversorPulgCent.java, vamos a esta línea.

Double.valueOf() "lanza" la excepción. Como consecuencia, aPulgadasButtonActionPerformed() finaliza lanzando también la excepción, sin ejecutar las líneas que están debajo de valueOf().

Yo quiero que mi aplicación pueda convertir el valor aunque el usuario lo escriba con coma. Para eso, vamos a pasarle a Double.valueOf() una cadena modificada, reemplazando la coma por punto.

```
private void aPulgadasButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String centimetrosString = centimetrosTextField.getText();

    centimetrosString.replace(',', '.');

    double centimetros = Double.valueOf(centimetrosString);

    double pulgadas = centimetros / 2.54;

    String pulgadasString = String.format("%.2f", pulgadas);

    pulgadasTextField.setText(pulgadasString);
}
```

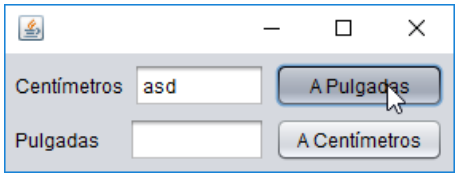
Utilizamos la función String.replace(), la que reemplaza la primer ocurrencia del primer carácter con la del segundo.

Sin embargo, al ejecutar la aplicación, observamos que vuelve a lanzar la excepción. Esto se debe a que replace(), al igual que todos las funciones de String, no modifican al objeto, sino que generan uno nuevo, el cual retornan.

```
private void aPulgadasButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String centimetrosString = centimetrosTextField.getText();

    centimetrosString = centimetrosString.replace(',', '.');

    double centimetros = Double.valueOf(centimetrosString);

    double pulgadas = centimetros / 2.54;

    String pulgadasString = String.format("%.2f", pulgadas);

    pulgadasTextField.setText(pulgadasString);
}
```

Al guardar la String que se retorna en centimetrosString, me quedo con la nueva String, y desecho la anterior, que ya no necesito.

Ahora quiero que cuando el usuario ingrese algo que no es un número, muestre un mensaje de error al usuario. Para lograr esto, es necesario "Atrapar" la excepción.



Para atrapar las excepciones utilizamos el bloque try/catch. En el bloque try colocamos la/s funciones que pueden lanzar la/s excepcion/es. En el bloque catch se coloca el tratamiento que se dará a la excepción, en caso de que se produzca.

Si la excepción se produce, mostraremos una ventana con un mensaje al usuario.

Al atrapar la excepción, ya no retornará la función. Ahora si necesitamos que retorne, hacemos un return.

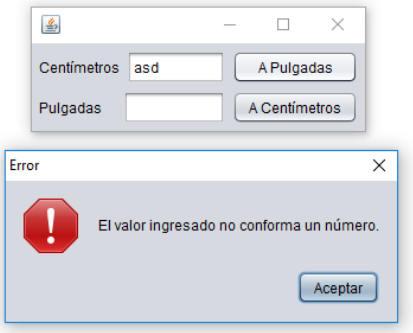
```
private void aPulgadasButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String centimetrosString = centimetrosTextField.getText();

    centimetrosString = centimetrosString.replace(',', '.');

    try
    {
        double centimetros = Double.valueOf(centimetrosString);
    }
    catch(NumberFormatException ex)
    {
        JOptionPane.showMessageDialog(this, "El valor ingresado no conforma un número.", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    double pulgadas = centimetros / 2.54;

    String pulgadasString = String.format("%.2f", pulgadas);

    pulgadasTextField.setText(pulgadasString);
}
```

Observe que la variable centímetros está subrayada en rojo. Es un error de compilación, que indica que no puede encontrar la variable. Esto ocurre porque la variable fue definida dentro del bloque try, por lo tanto, sólo existe dentro de ese bloque. Si queremos que exista luego del bloque try, debemos definirla fuera de éste, en el bloque de la función.

Ahora el error se ha ido.

```
private void aPulgadasButtonActionPerformed(java.awt.event.ActionEvent evt)
{
    String centimetrosString = centimetrosTextField.getText();

    centimetrosString = centimetrosString.replace(',', '.');

    double centimetros;

    try
    {
        centimetros = Double.valueOf(centimetrosString);
    }
    catch(NumberFormatException ex)
    {
        JOptionPane.showMessageDialog(this, "El valor ingresado no conforma un número.", "Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    double pulgadas = centimetros / 2.54;

    String pulgadasString = String.format("%.2f", pulgadas);

    pulgadasTextField.setText(pulgadasString);
}
```

Lo ejecutamos...



Ahora deberá modificar el conversor, de manera que utilice sólo un botón. Para lograrlo, puede utilizar una bandera que le indique cuál fue el último campo editado. Luego, el botón convertir se fijará en el valor de la bandera y sabrá en qué sentido realizar la conversión. Puede saber cuál fue el último campo editado asociando el evento "Focus Lost" (que se dispara cada vez que un control pierde el foco del teclado) a cada TextField, y modificando la bandera en cada función asociada. La bandera no podrá ser una variable local, porque debe sobrevivir al fin de la función, y poder ser utilizada por la función que acciona el botón. Por lo tanto deberá ser un atributo de la clase, que vive el tiempo que vive el objeto.