Tarea para DI02

Juan A. García Muelas

nov. 2021

En esta tarea vamos a practicar la creación de un interfaz de usuario utilizando **QtJambi**. La tecnología usada es Swing y QtJambi. Debes ser muy cuidadoso en este apartado y tener claro la tecnología que estás usando para realizar el interfaz de usuario.

Recapitulando, estos son los requisitos que debe tener tu aplicación para esta tarea:

1. Existirá una pantalla **redimensionable** que contenga una tabla con la lista de la compra. También habrá un **botón "borrar todo", un botón "borrar selección"**, un **botón "añadir",** un **botón "imprimir".** Los botones de borrar actuarán sobre el contenido de la tabla borrando todos los elementos o solo los elementos seleccionados mediante un click en el checkbox de la primera columna "X". El botón añadir desplegará una nueva ventana con los campos para introducir un nuevo elemento y el botón imprimir desplegará una nueva ventana que mostrará en formato texto los elementos y sus datos, contenidos en la tabla, como si de una impresión se tratase.
2. Para cada producto que se vaya a dar de alta se deben cumplimentar los siguientes datos:
    1. Cantidad, será un número positivo.
    2. Nombre.
    3. Sección del super, a elegir entre: panadería, pescadería, frutería, carnicería, charcutería, conservas, perfumería, general.
    4. Si se trata o no de un producto urgente.
3. Para añadir cada elemento a la lista, se usará el botón añadir que ha de contender una imagen.
4. Los elementos añadidos se mostrarán en la de tabla.
5. El botón imprimir simulará una impresora, y permitirá mostrar la lista de la compra en formato texto a través de un cuadro de diálogo.
6. No será necesario que los datos introducidos en el sistema persistan entre diferentes ejecuciones del programa (cada vez que se arranque el programa, no habrá ningún producto en la lista de la compra).

Como en los criterios de evaluación se valora positivamente la originalidad y los cambios, he pensado que podía ser una buena idea plantearse la resolución de la tarea a través de una interfaz optimizada, más sencilla que la propuesta, que facilita el uso de la aplicación para un futuro usuario.

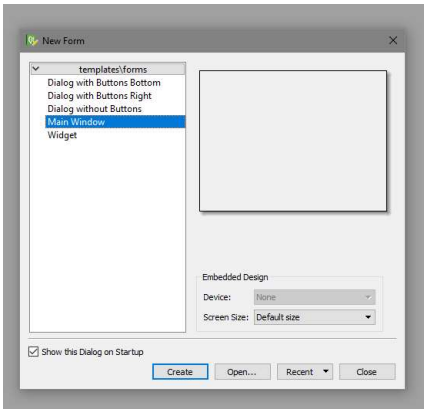
Por ello, he creado dos proyectos: un primer proyecto que se desarrolla bajo esta premisa, y un segundo proyecto que se ejecuta con las prescripciones y vistas solicitadas en el enunciado.

Para mejorar la comprensión de los proyectos creados, mostraré primero la opción personalizada, y tras las capturas de su ejecución, la opción del enunciado.

1. Diseño de Interfaz mediante Qt Designer

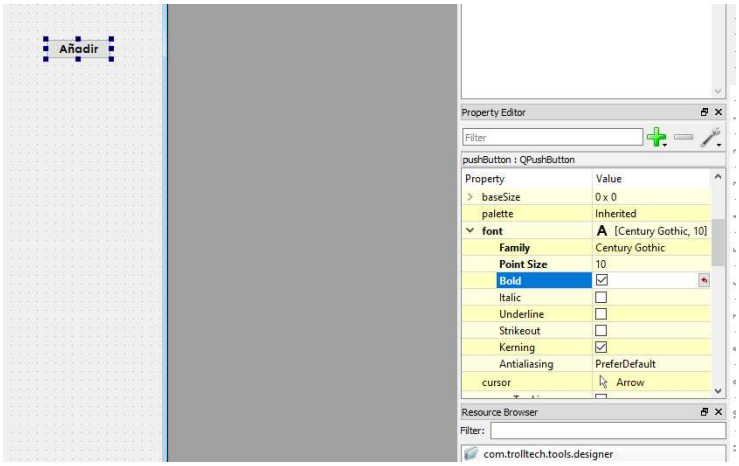
Como pide la tarea, procedemos a la instalación de **QT Jambi** y tras ello, abrimos el **Qt Designer**, donde se diseña la interfaz que se va a utilizar para su ejecución.

Se crea un **`mainwindow`** para personalizarlo a la medida de los requisitos pedidos y se guarda con la extensión **`.jui`** .



Creación de MainWindow en Qt Designer

Añadimos un primer botón y vamos personalizando sus propiedades, como los estilos de fuente, o su **`tooltiptext`**:

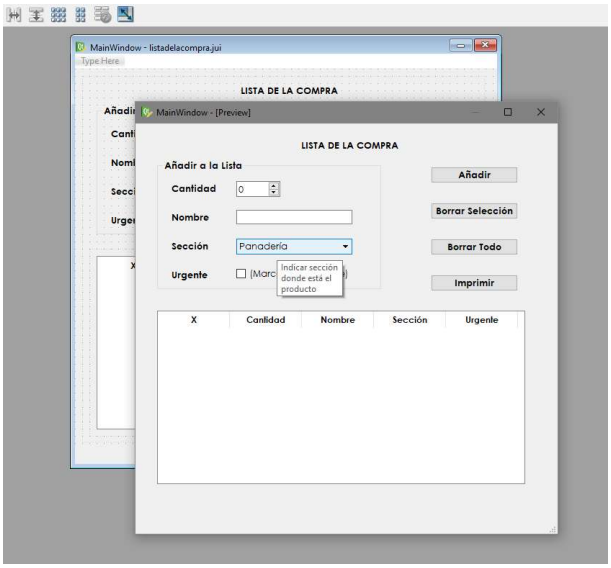


Modificando atributos de QPushButton

Se crean los distintos elementos de la interfaz, decidiendo iniciarla con una etiqueta Label a modo de "titulo" y dos **`widget`** que contendrán por un lado el apartado de creación de la lista, por medio de un **`Group Box`** que incorpora un **`Spin Box`** con inicialización a 0 para evitar números negativos en la cantidad, un **`Line Edit`** para el nombre, un **`Combo box`** con las distintas secciones y un **`Check Box`** para marcar en caso de ser urgente. A la derecha del **`Group Box`**, se alojan los **`Push Button`** que relizarán las distintas opciones.

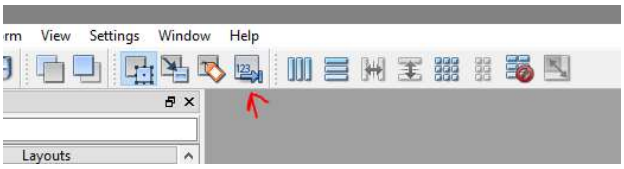
Debajo de este primer widget, se aloja otro con un **`Table widget`** que mostrará lo que se vaya apuntando.

Con la opción **`Ctrl+R`** podemos ver la **vista previa** para ir afinando la presentación.



Vista previa de la interfaz

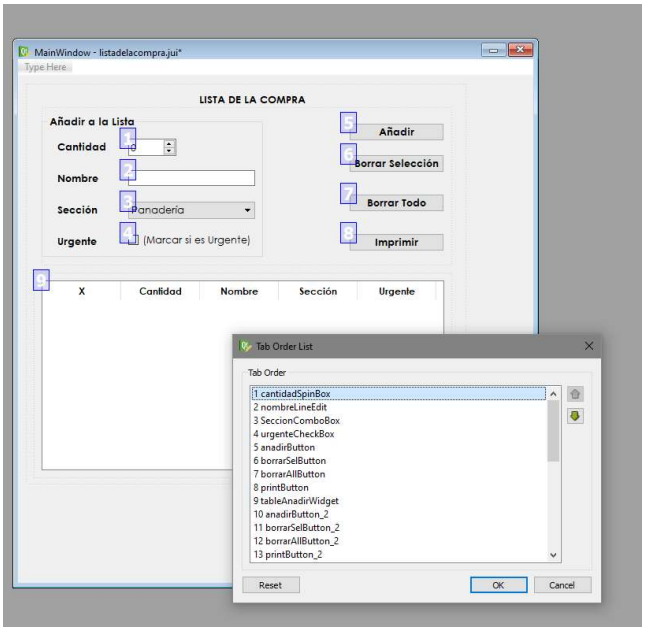
Con los rasgos principales ya predefinidos, se modifican las **tabulaciones.**



Barra de Herramientas Qt Designer - Tabs

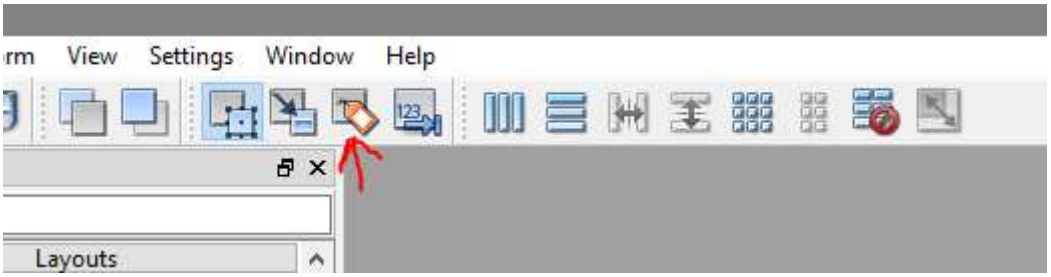
Al mostrarnos el orden, puede modificarse mediante el menú secundario y reordenando según nuestro criterio.

En este caso, colocaba en primer orden los botones, y se ha cambiado ese orden para darle prioridad al **`Group Box.`**



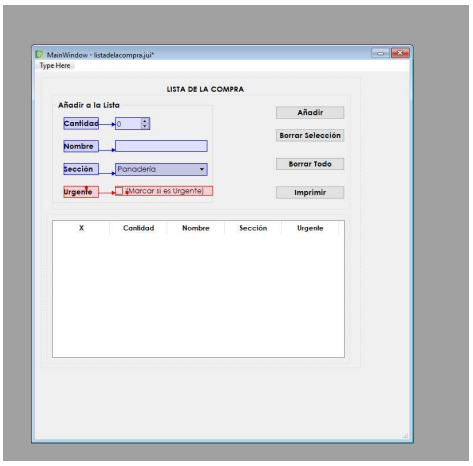
Ordenación de tabulaciones

Lo siguiente es editar lo **`buddies`** que enlazarán los elementos:



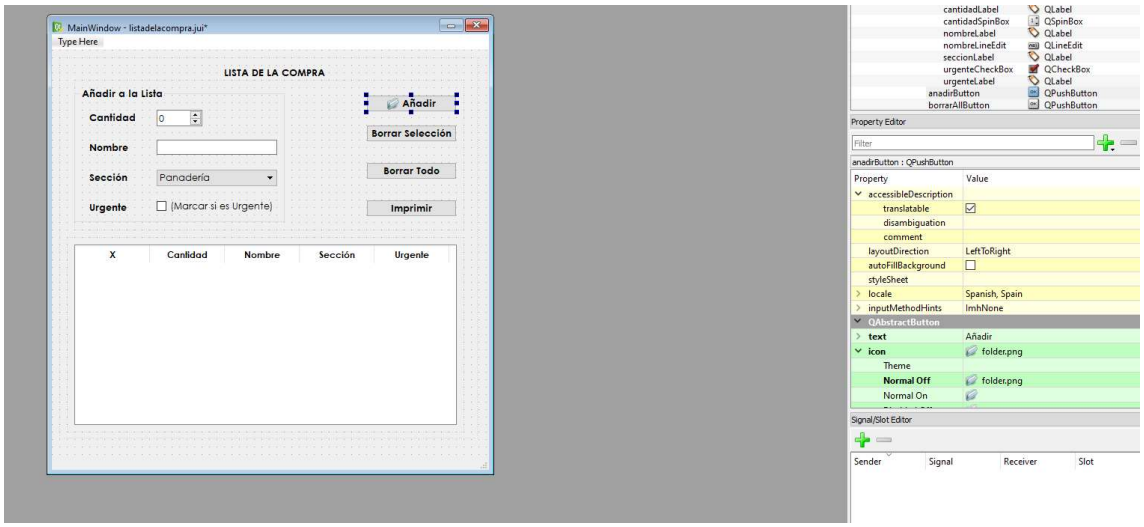
Barra de Herramientas Qt Designer - buddies

Haciendo click sobre cada uno, podemos arrastrar la flecha para enlazarlos.



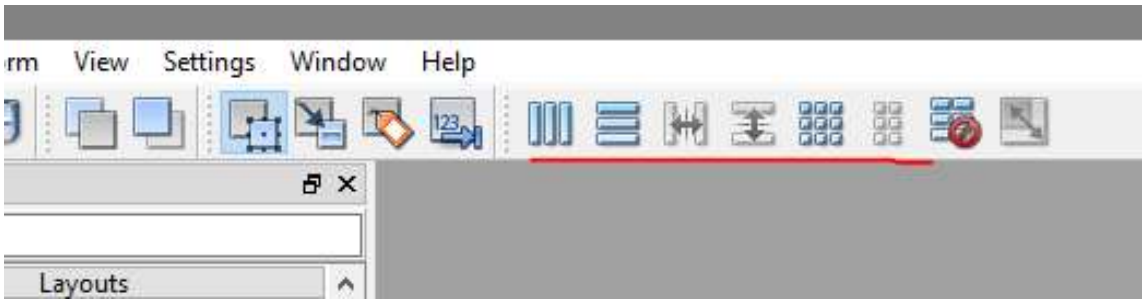
Enlazado de elementos

Nos piden que el botón de Añadir contenga una imagen. Se aprovecha, ya que me parece apropiada (y tendremos así enlazado luego el código), una de las imágenes que ya incorpora como iconos la aplicación desde las **propiedades del editor**, donde hacemos la selección.



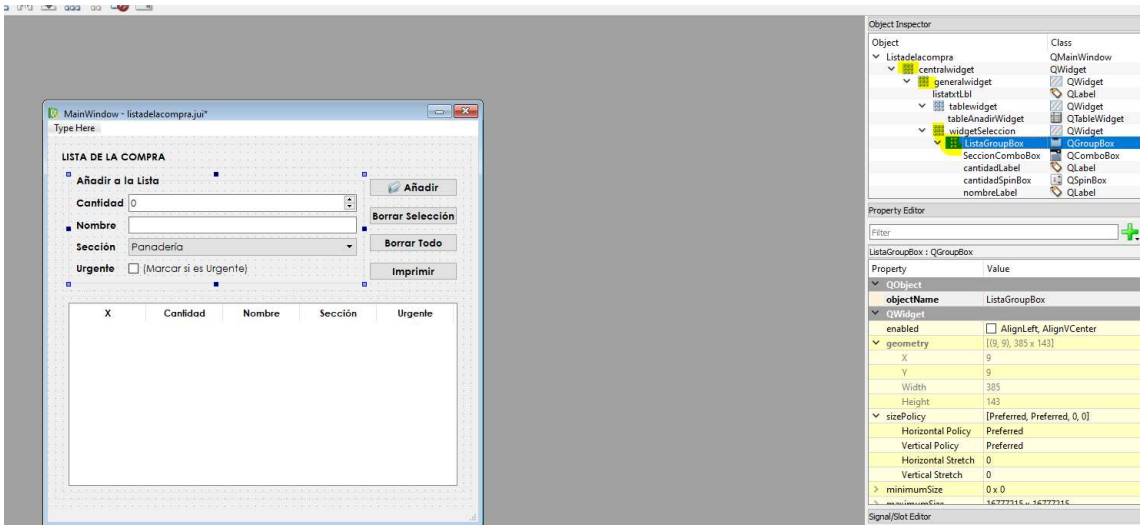
Añadir icono/imagen

Para rematar la parte de diseño, acomodamos mediante los **`layouts`** las distintas partes de la interfaz para conseguir que sea **redimensionable** sin perder visibilidad.



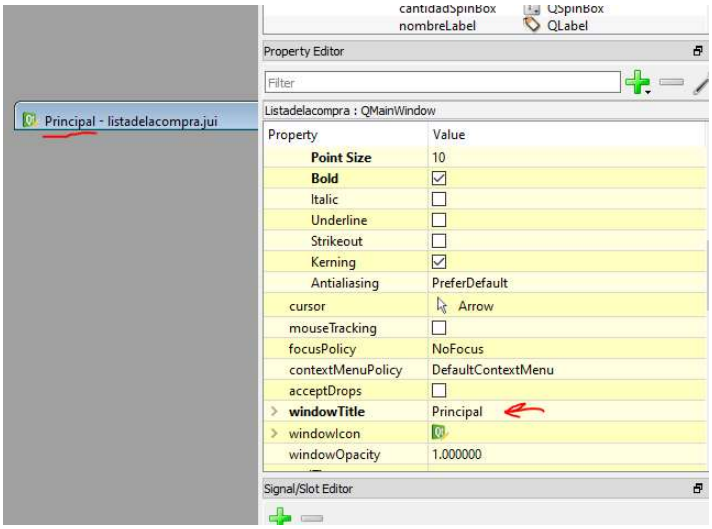
Barra de Herramientas Qt Designer - Layouts

Por el diseño propuesto, he añadido un **`Form Layout`** para la parte del **`Group Box`**, el resto, pueden acomodarse a la pantalla mediante **`Grid Layout`**.



Ajuste de Layouts

Como detalle, puede personalizarse también el icono y el **`title`** de la ventana.

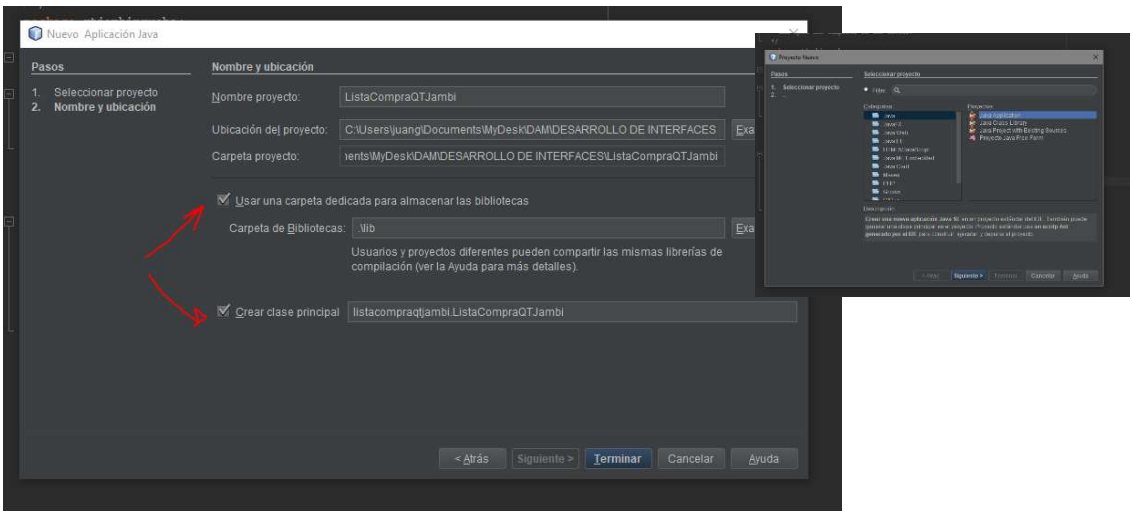


Modificar title o icono ventana

Con la interfaz diseñada, guardamos los datos y abrimos **NetBeans**.

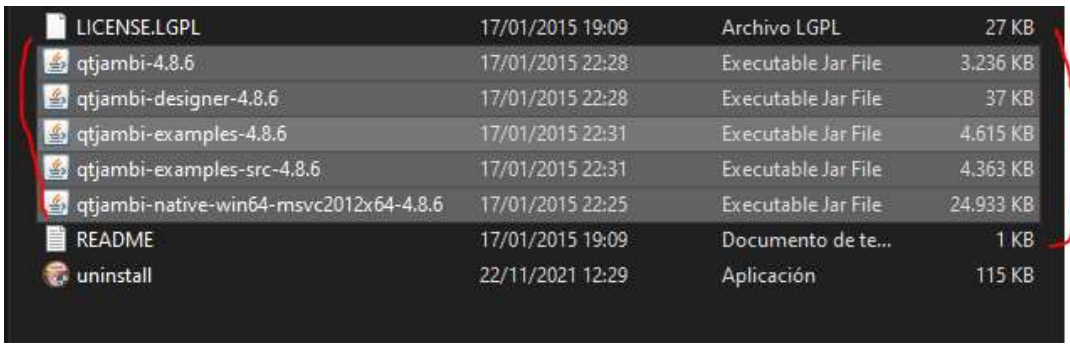
2. Añadir interfaz a NetBeans

Se crea un nuevo proyecto de la forma habitual en NetBeans, asegurando que hacemos check sobre el uso de librerías.



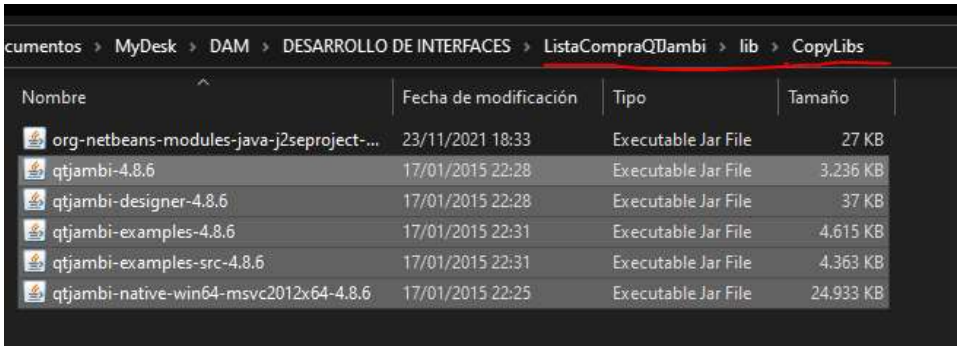
Crear nuevo proyecto

Vamos a nuestro directorio de **QTJAmbi** (en mi caso **`C:\qtjambi-4.8.6`**) y copiamos los **`jar`**.



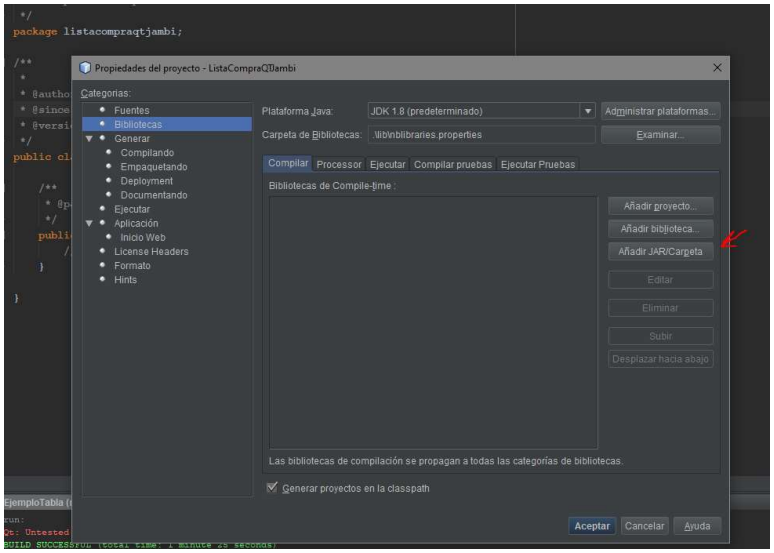
Copiar archivos jar Qt Jambi

En el proyecto recién creado, accedemos a la carpeta **`lib/CopyLibs`**, donde pegamos los archivos.



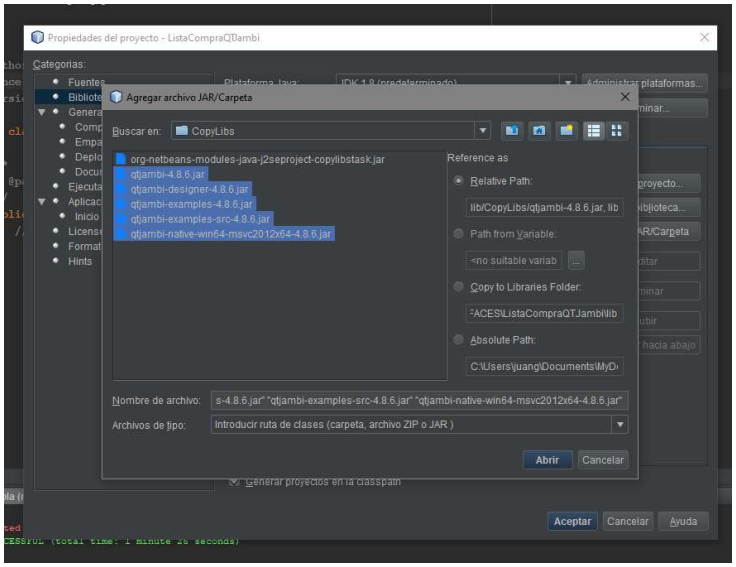
Copiar archivos jar Qt Jambi en CopyLibs

Lo siguiente es irnos a las **propiedades** de nuestro proyecto y desde **Bibliotecas**, pulsar sobre **`Añadir JAR/Carpeta`**

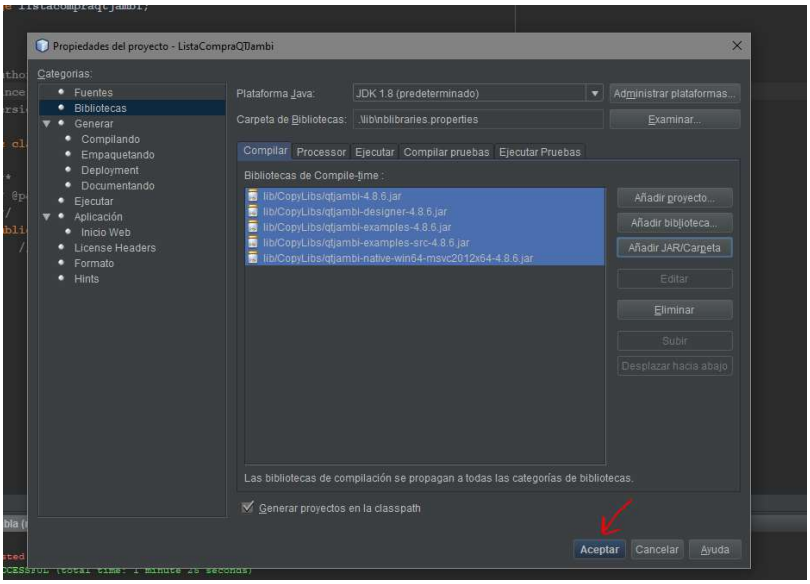


Añadir Jar a proyecto

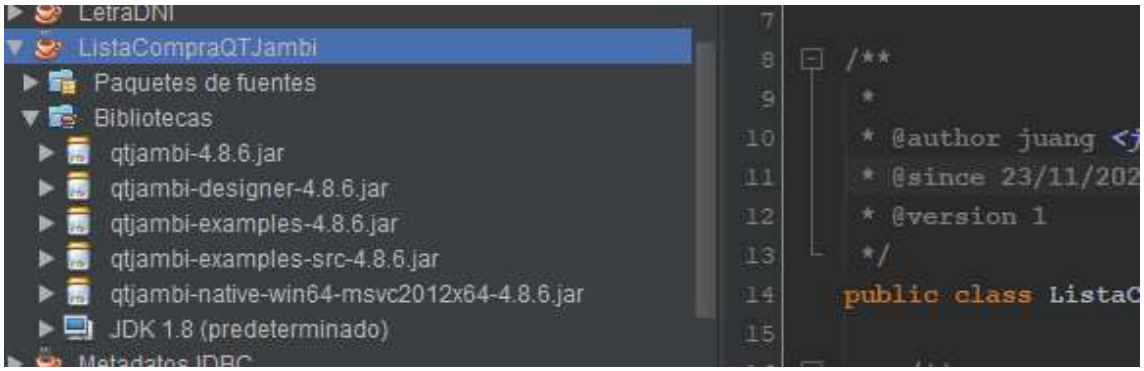
Copiamos los archivos y los pegamos (sino aparecieran ya en nuestro directorio) dentro de **`CopyLibs`**)



Añadir Jar a proyecto

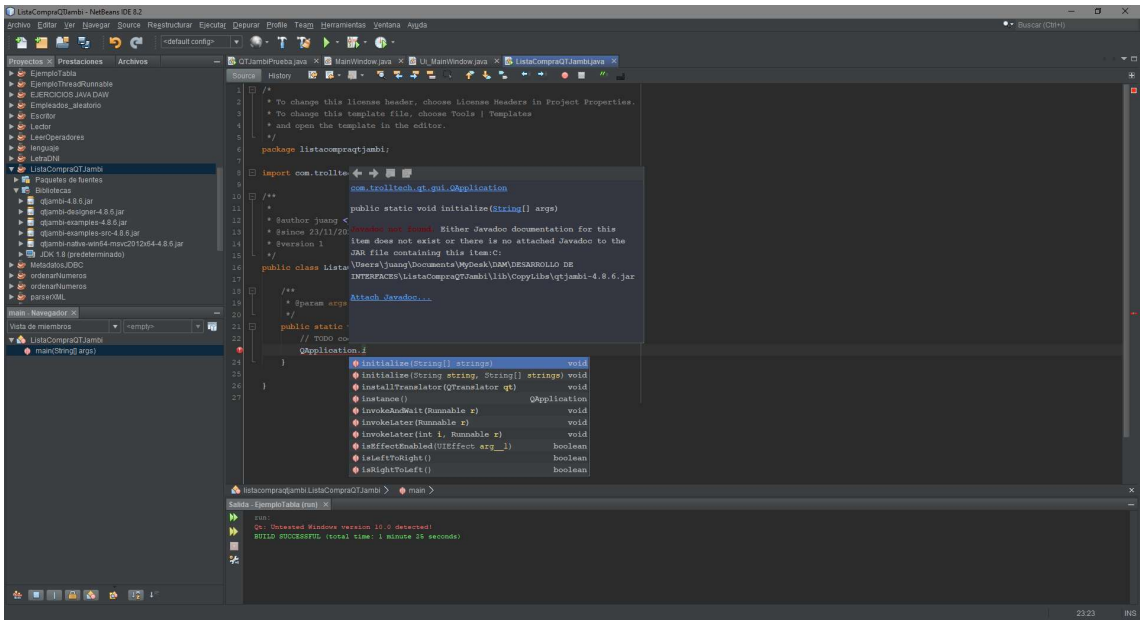


Añadir Jar a proyecto

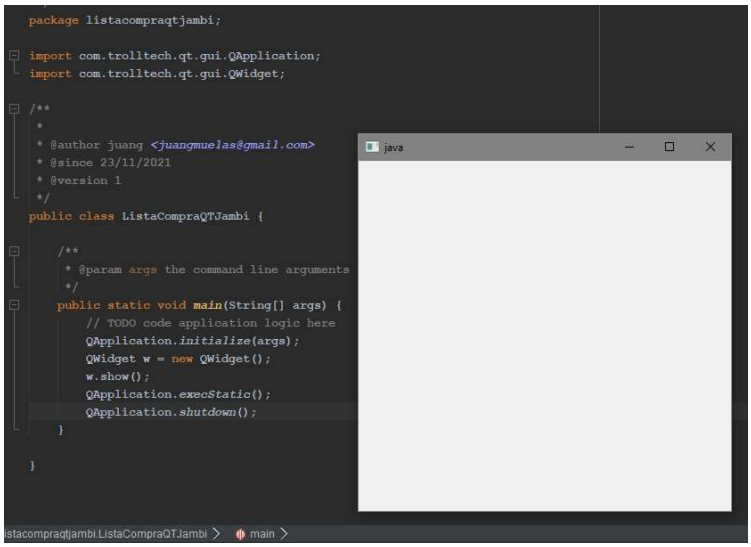


Librería Jar en proyecto

Ahora, ya podemos comprobar que tenemos la carpeta añadida y que detecta su uso según generamos nuestro código. Con unas pocas líneas de código, ejecutamos para comprobarlo:



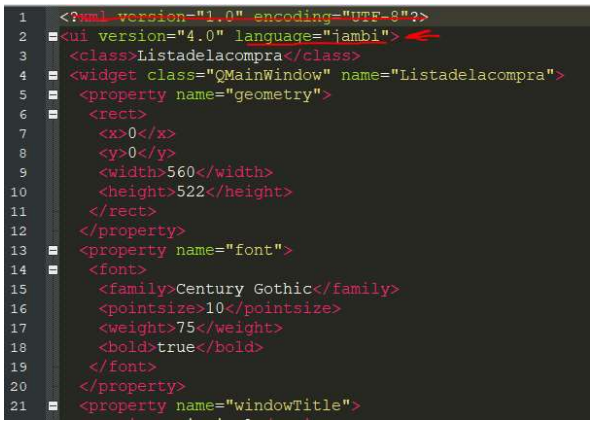
Comprobando acceso a QT



Comprobando acceso a QT

Con las comprobaciones previas hechas, volvemos a nuestro archivo **`listadelacompra.jui`** y lo abrimos desde cualquier editor, para comprobar que utilizamos **Jambi**, y eliminar el encabezado de codificación **XML**.

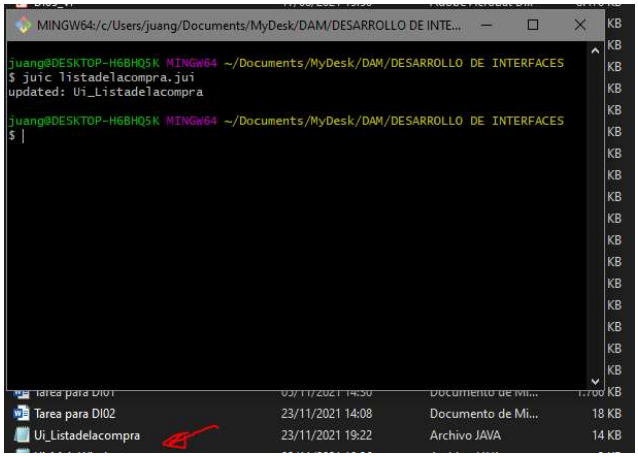
Tras ello, procedemos a guardar los cambios antes de pasar a la consola de comandos.



listadelacompra.jui para modificar

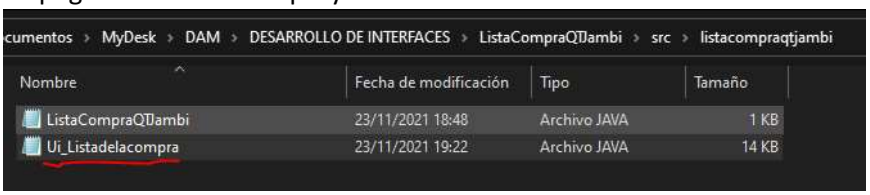
Abrimos ahora la consola en el directorio donde tenemos el archivo y ejecutamos **`juic listadelacompra.jui`**

Esto nos crea el archivo **`UI_listadelacompra.java`**



Juic de listadelacompra.jui en la consola

Lo pegamos en nuestro proyecto:



Archivos java

Lo abrimos en **Netbeans** y le añadimos el **`package`** para poder utilizarlo.

```
7   ********************************************
8   package listacompraqtjambi;
9   import com.trolltech.qt.core.*;
10  import com.trolltech.qt.gui.*;
11
12  public class Ui_Listadelacompra implements com.t
13  {
14      public QWidget centralwidget;
15      public QGridLayout gridLayout;
```

Detalle Package

Generamos una nueva clase para trabajar con los objetos de este archivo, llamada **`ListadelaCompra.java`**, recordando que debe extender de **`QMainWindow`**.

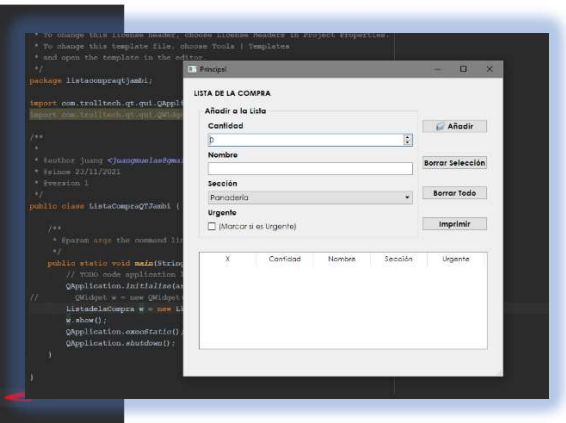
Inicializamos un objeto de **`Ui_ListadelaCompra`** y creamos el constructor.

```
*/
package listacompraqtjambi;

import com.trolltech.qt.gui.QMainWindow;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 23/11/2021
 * @version 1
 */
public class ListadelaCompra extends QMainWindow {
    private Ui_Listadelacompra ui;

    public ListadelaCompra(){
        ui = new Ui_Listadelacompra();
        ui.setupUi(this);
    }
}
```

Constructor clase ListadelaCompra.java

En nuestra clase principal modificamos el **`widget`** creado para la prueba para que apunte ahora a nuestra clase y probamos la interfaz añadida.

```
6   package listacompraqtjambi;
7
8   import com.trolltech.qt.gui.QApplication;
9   import com.trolltech.qt.gui.QWidget;
10
11  /**
12   *
13   * @author juang <juangmuelas@gmail.com>
14   * @since 23/11/2021
15   * @version 1
16   */
17  public class ListaCompraQTJambi {
18
19      /**
20       * @param args the command line arguments
21       */
22      public static void main(String[] args) {
23          // TODO code application logic here
24          QApplication.initialize(args);
25  //          QWidget w = new QWidget();
26          ListadelaCompra w = new ListadelaCompra();
27          w.show();
28          QApplication.execStatic();
29          QApplication.shutdown();
30      }
31
32  }
33
```

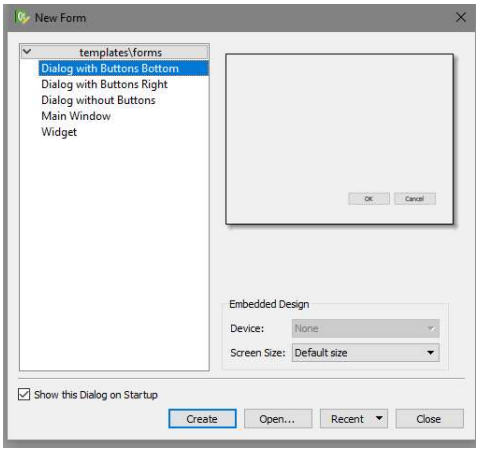


Llamada a ListadelaCompra.java y prueba de interfaz

Con el botón imprimir, debe mostrarnos una nueva ventana que nos muestre la lista confeccionada.

Para ello, volvemos al editor para poder crearla siguiendo las pautas de la anterior interfaz.

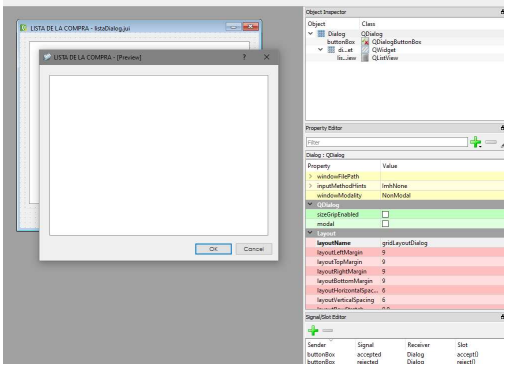
Creamos el nuevo **`Dialog`** para mostrar la lista, y aprovechando las opciones, se elige un diseño con dos botones.



Creación de un Dialog en Qt Designer

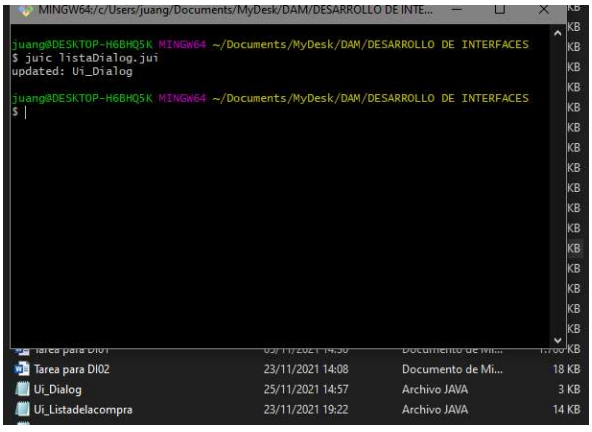
Como en el primer **`frame`**, se personaliza cada parte de nuestra selección, se añaden un **`QListWidget`** para mostrar luego el listado y se añaden los **`gridLayouts`** para redimensionarla.

Los **`Slots`** vienen ya configurados con la plantilla.



Modificaciones para personalizar ventana

Hacemos los mismos pasos para pasar nuestro archivo **`listaDialog.jui`** a java



listaDialog.jui

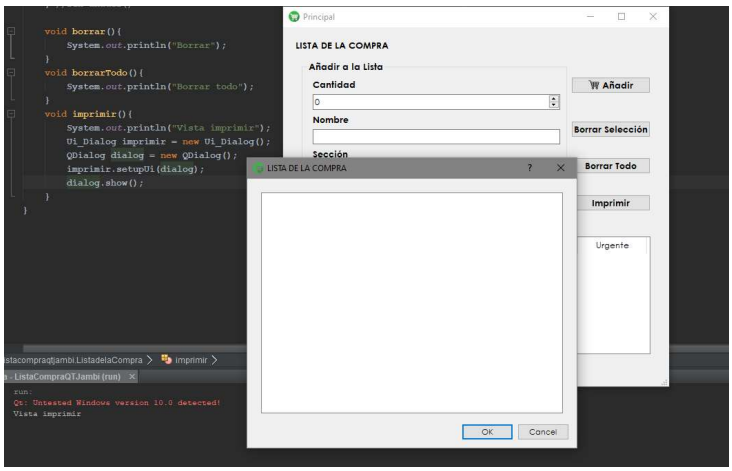
Copiado en nuestro directorio, también podemos usarlo en el proyecto tras haber añadido el **package**.

3. Desarrollo funcional en NetBeans

Si bien ya se ha visto en el punto anterior la estructura con la que va a ejecutarse este programa, nos queda añadirles funcionalidad a lo componentes, y eso se hace desde la clase **ListadelaCompra.java**

Creamos las funciones (vacías por ahora) con las que estructuramos cada acción: **anadir()**, **borrar()**, **borrarTodo()** e **imprimir()**. Como acabamos de crear el nuevo cuadro de **Dialog**, comprobaremos a través de esta última que se muestra correctamente, creando un nuevo objeto **dialog** y llamando al método **show().**

Con ello, tendremos una parte de la función imprimir ya creada. El resto se podrá obtener al consultar la tabla.



Vista Dialog con la función imprimir()

Este archivo **Ui_Dialog.java**, se mantiene casi inalterado respecto a los cambios y personalizaciones hechas en el diseñador, salvo el añadido del **package** comentado anteriormente y la modificación del icono, aprovechando que sólo debemos modificar la ruta.

Modificar icono ventana

Para poder mostrar datos en esta vista, deberemos darle funcionalidad al botón añadir, por lo que es el siguiente paso:

Lo primero, es recoger los datos que se reciben desde la aplicación.

Luego, se genera la tabla donde se mostrará un **checkbox** en la primera columna.

A través de **QTableWidgetItem**, se recogen los valores de las variables para mostrarlos, asignándolos mediante la orden **setItem()**.

Hasta aquí ya podemos conseguir y mostrar los datos en la tabla.

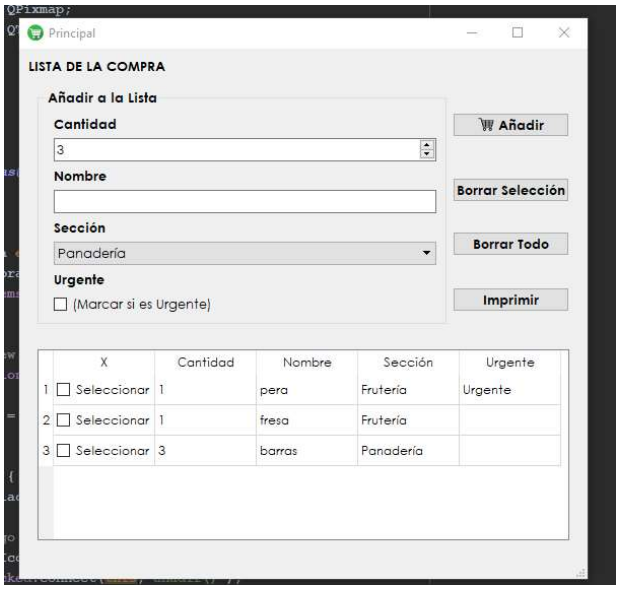


Tabla con el listado

Lo siguiente es guardar esos datos en un **`ArrayList`**, que se usará para las otras funciones

```
items = new ArrayList<String>();
items.add(cantidad);
items.add(nombre);
items.add(seccion);
items.add(urgente);

listado = items.get(0)+" * ".concat(items.get(1))+" * ".concat(items.get(2))+" * ".concat(items.get(3));
System.out.println(listado);
//Vamos a añadir las "columnas"
for(int i = 0; i < 4 ; i++){
    list.add(new ArrayList<String>());
}

//Agregamos los datos que se recojan en la tabla
list.get(0).add(items.get(0));
list.get(1).add(items.get(1));
list.get(2).add(items.get(2));
list.get(3).add(items.get(3));
```

Guardando datos en ArrayList

Con esto, se aprovecha para finalizar la función **`imprimir()`**.

Con un bucle **`for`**, recorremos el listado y lo enviamos al **`Dialog`**.

```
void imprimir(){
    System.out.println("Vista imprimir");
    Ui_Dialog imprimir = new Ui_Dialog();
    QDialog dialog = new QDialog();

    imprimir.setupUi(dialog);
    dialog.show();
    imprimir.listadoWidget.addItem("LISTA DE LA COMPRA");
    System.out.println(list.size());

    for(int i = 0; i <= list.get(0).size()-1 ; i++){
        String lista = ""+list.get(0).get(i)+"\t "+list.get(1).get(i)+"\t "+list.get(2).get(i)+"\t "+list.get(3).get(i);
        System.out.println(list);
        imprimir.listadoWidget.addItem(lista);
          i++;
    }
}//fin imprimir()
```

Bucle for para mostrar la lista

En la siguiente imagen podemos comprobar el resultado:

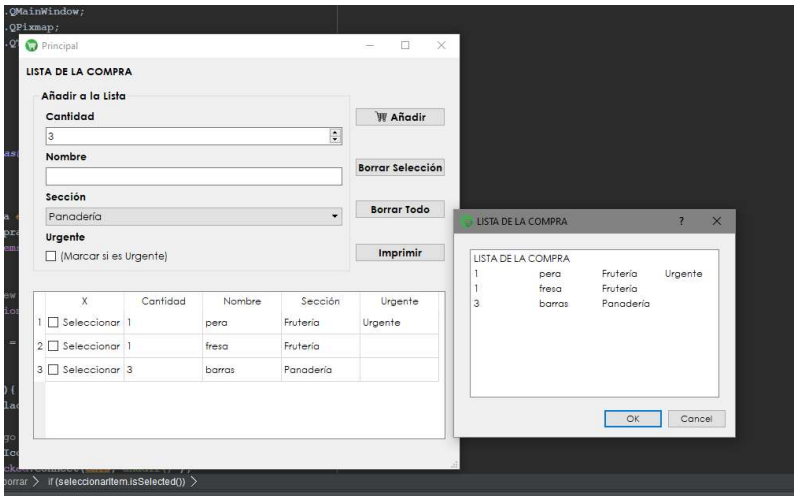


Tabla y listado en Dialog

La función **`borrarTodo()`** es sencilla, pues con poner el contador de filas a cero nos llega para su borrado en nuestra tabla y con la función **`clear()`**, limpiamos la lista.

```
void borrarTodo(){
    System.out.println("Borrar todo");
    ui.tableAnadirWidget.setRowCount(0);
    list.clear();
}
```

Función borrarTodo()

Finalmente queda la función **`borrar()`**.

Como en **`borrarTodo()`**, consta de dos partes: una que borra en el listado y otra que recoge los datos del array para borrarlos y que no pasen a la lista de impresión.

Lo primero se solventa con un bucle **`for`** que recorre las filas.

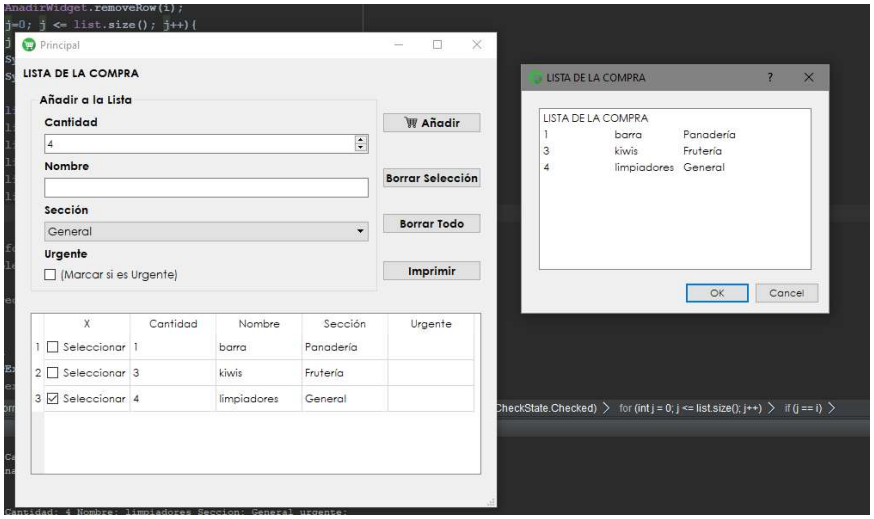
```
void borrar(){
    System.out.println("Borrar");
    //Para comprobar en consola
    String select="";
    //Item a seleccionar
    QTableWidgetItem itemSel;
    try {
        //Recorremos la tabla
        for(int i = 0; i <= ui.tableAnadirWidget.rowCount(); i++){
            //El item será el que esté la fila "i" columna 0
            itemSel = ui.tableAnadirWidget.item(i,0);

            //Si está checkeado
            if(itemSel.checkState() == Qt.CheckState.Checked ){
                //Recojo el nombre para la consola
                select= items.get(1);
                  System.out.println("NOMBRE CAMPO SELECCIONADO: " + select);

                //borrar la fila
                ui.tableAnadirWidget.removeRow(i);
                for(int j=0; j <= list.size(); j++){
                    if( j ==  i){
                          System.out.println("INDICE " + j + ": "+items.get(j));
                          System.out.println("ITEMS A BORRAR:");
                          System.out.println(list.get(i).set(i,items.get(j)));
                       list.get(i).remove(list.get(i).set(i,items.get(j)));
                    j--;
                    }
                } //fin for lista
            //Vuelve a recorrer por si hay más checked
            i--;
            }
        }//fin for tabla
    } catch (NullPointerException e){
        //Capturando la excepción
        System.out.println(e.getMessage());
    }catch (ArrayIndexOutOfBoundsException exc){
```

Función borrar()

Si una está **checked**, lo selecciona y con **removeRow()**, se elimina de la tabla. Antes de cerrar el **for**, debemos forzar una recursividad en el chekeo, decrementando la variable que usamos como contador.

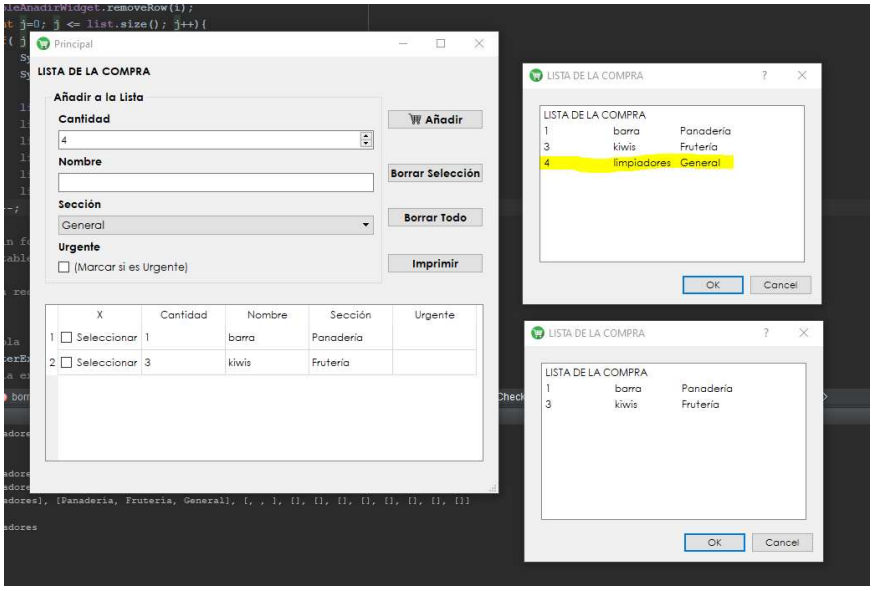
Lo siguiente, será hacer un bucle similar, pero que recoja los datos del **ArrayList** y mediante **remove()** se borren, volviendo antes de salir de la condición a hacer un repaso recursivo sobre los array.

Volviendo a probar el ejercicio, se comprueba el funcionamiento.



Función borrar()

Tras el borrado de la línea 3, nos muestra sólo los datos restantes.



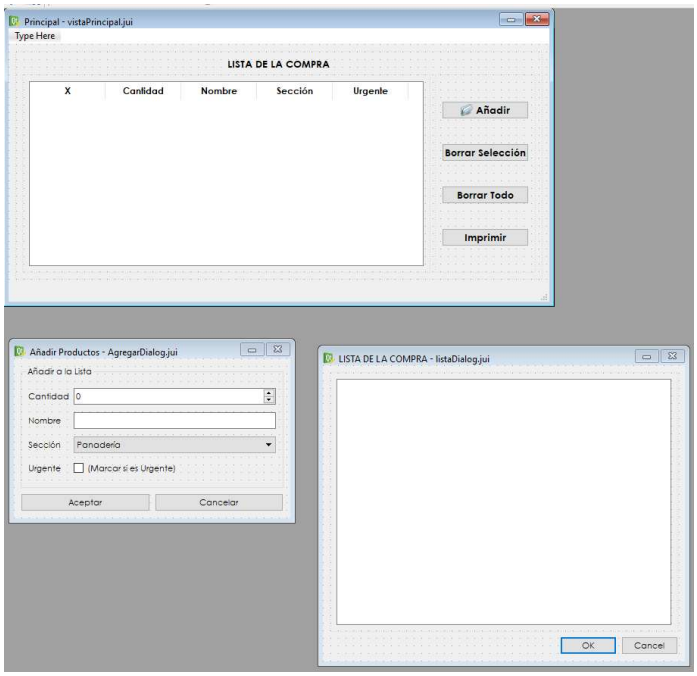
Función borrar()

OPCIÓN 2 SEGÚN ENUNCIADO.

Tras desarrollar la forma que me parecía más óptima para realizar una aplicación sencilla y práctica de "Lista de la Compra", paso a mostrar cómo sería su desarrollo según el enunciado propuesto.

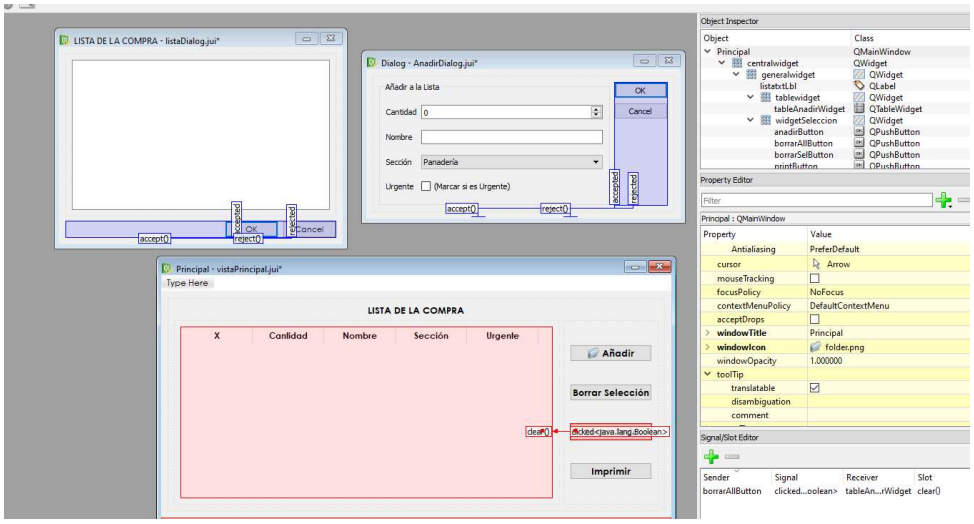
En él nos piden una pantalla con una tabla y los botones. Añadir, nos abre el diálogo de adicción e imprimir nos muestra la lista.

Desde el diseñador, se crean los distintos archivos **`jui`**, de igual forma que en la parte anterior.



Detalle diseño interfaces Qt Designer.

Añadimos personalizaciones e imágenes.

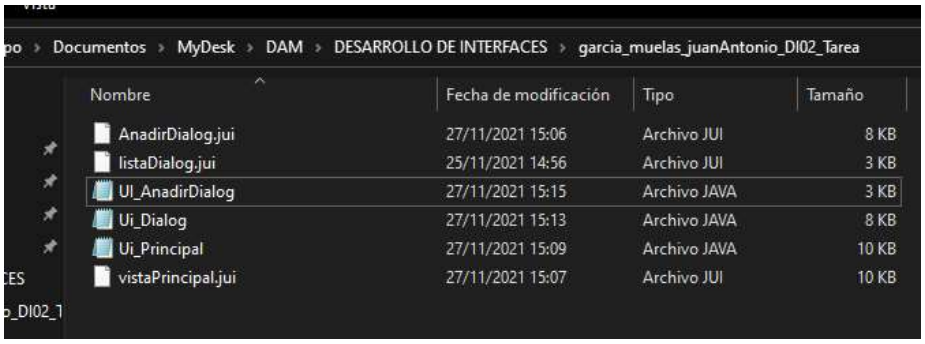


Detalle manejo propiedades en QtDesigner.

Desde el **CMD** creamos los archivos java

```
juang@DESKTOP-H6BHQ5K MINGW64 ~/Documents/MyDesk/DAM/DESARROLLO DE INTERFACES/g
rcia_muelas_juanAntonio_DI02_Tarea
$ juic listaDialog.jui
updated: Ui_Dialog

juang@DESKTOP-H6BHQ5K MINGW64 ~/Documents/MyDesk/DAM/DESARROLLO DE INTERFACES/g
rcia_muelas_juanAntonio_DI02_Tarea
$ juic vistaPrincipal.jui
updated: Ui_Principal

juang@DESKTOP-H6BHQ5K MINGW64 ~/Documents/MyDesk/DAM/DESARROLLO DE INTERFACES/g
rcia_muelas_juanAntonio_DI02_Tarea
$ juic AnadirDialog.jui
```

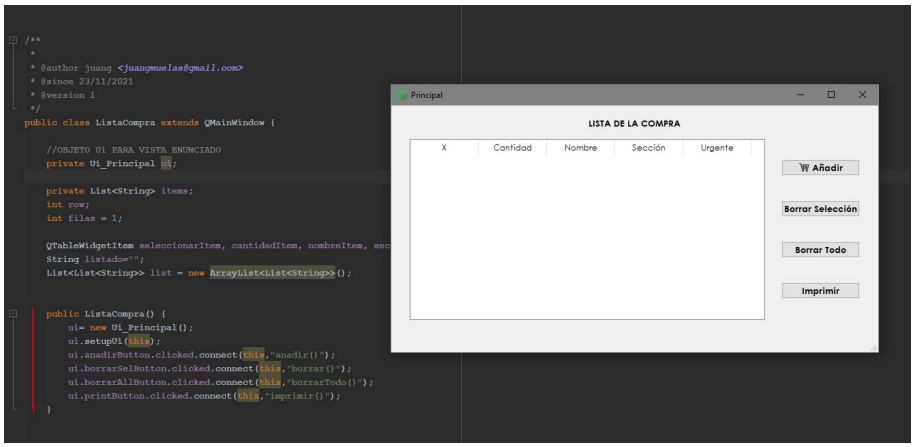
Compilación de archivos jui a java.



Archivos java generados tras compilación.

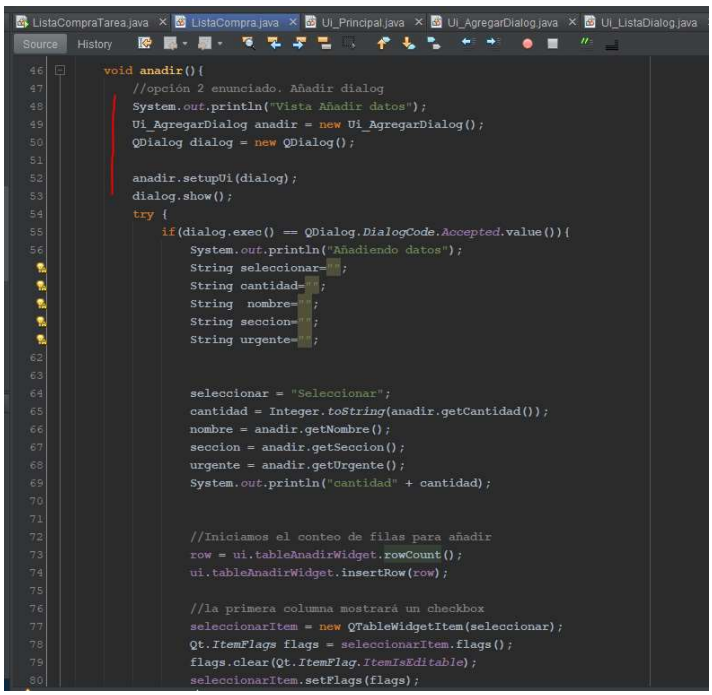
Añadimos archivos a nuestro proyecto, al igual que las librerías de **`qtJambi`**, del mismo modo descrito en el primer apartado (pág. 6-7).

Se crea **`ListaCompra.java`** con el nuevo constructor para poder usar el archivo **`Ui_Principal.java`**.



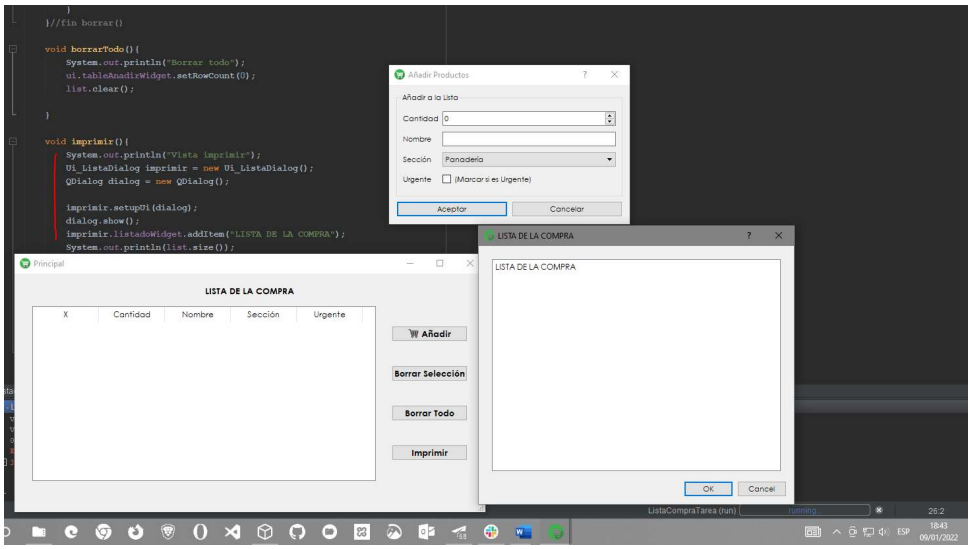
Detalle constructor ListaCompra e interfaz.

Accedemos a las funciones y en **`anadir()`**, creamos un objeto **`Ui_AgregarDialog`** para hacer visible esa nueva ventana de diálogo.



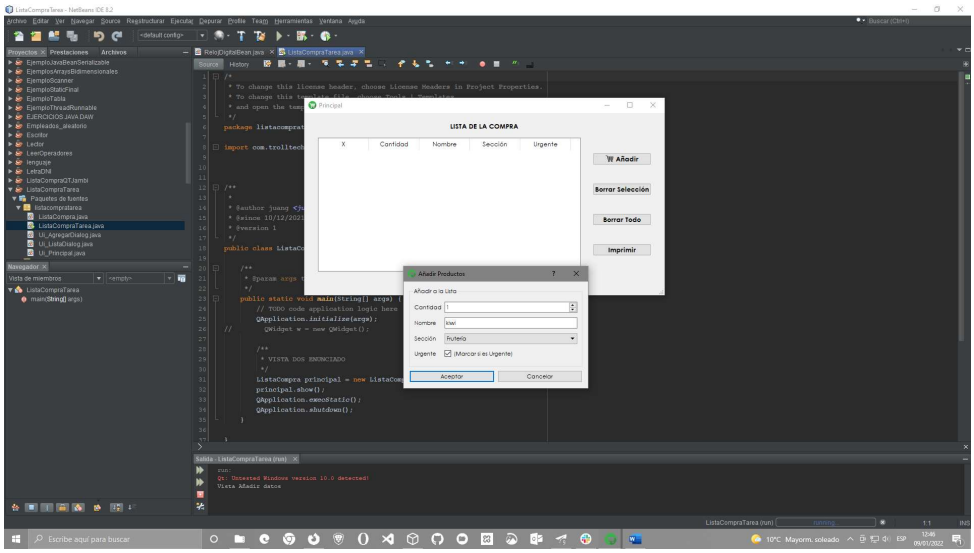
Detalle método para añadir productos a tabla y listado.

En **`imprimir()`**, hacemos los mismo con **`Ui_ListaDialog()`**.

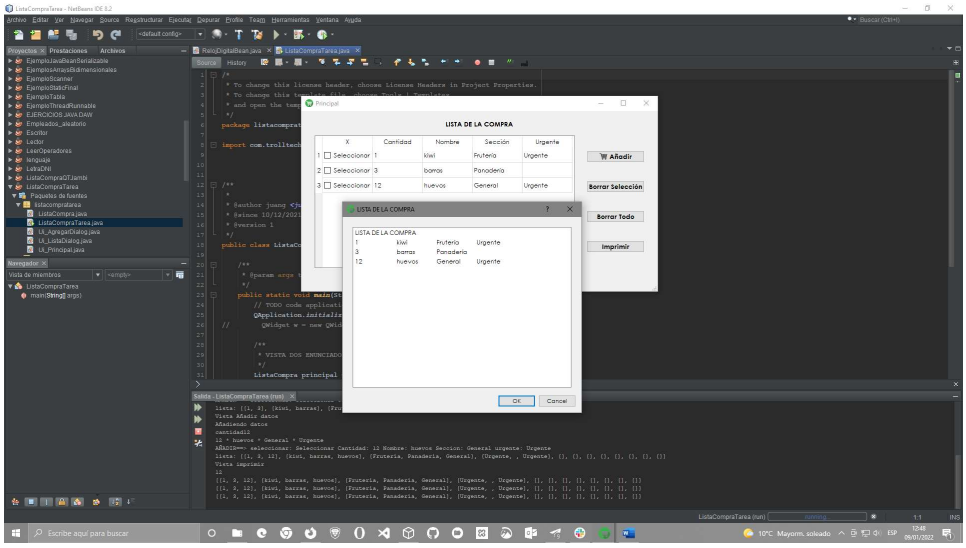


Detalle método imprimir e interfaces generadas por botones.

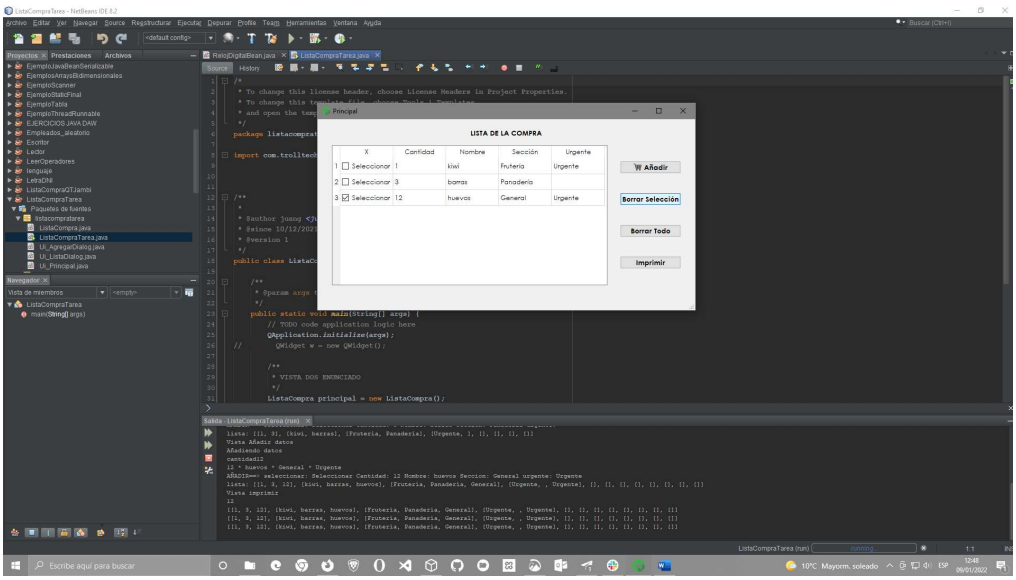
Mostradas las diferencias principales con el primer ejemplo, pasamos a ejecutar:



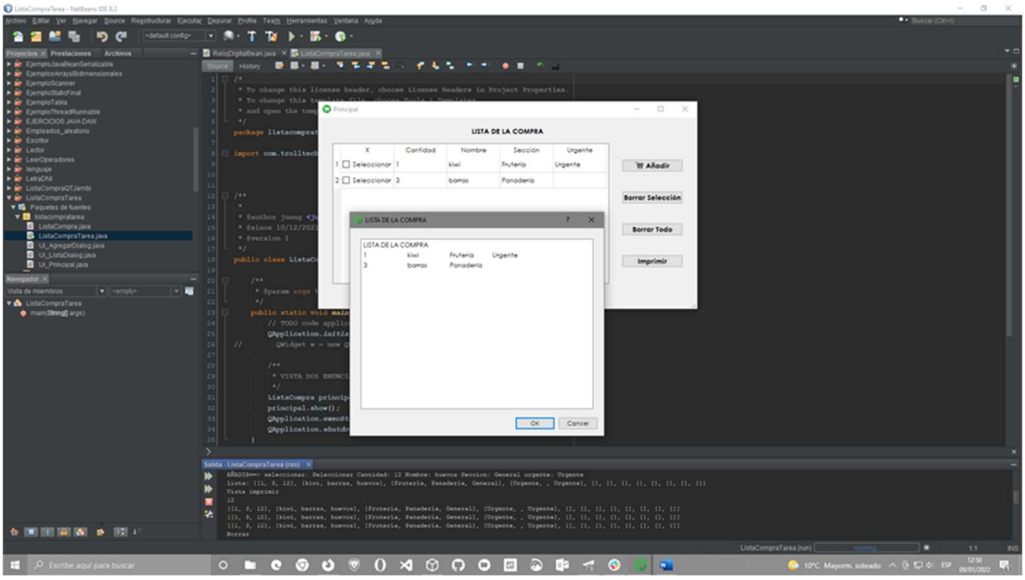
Detalle de ejecución para añadir productos.



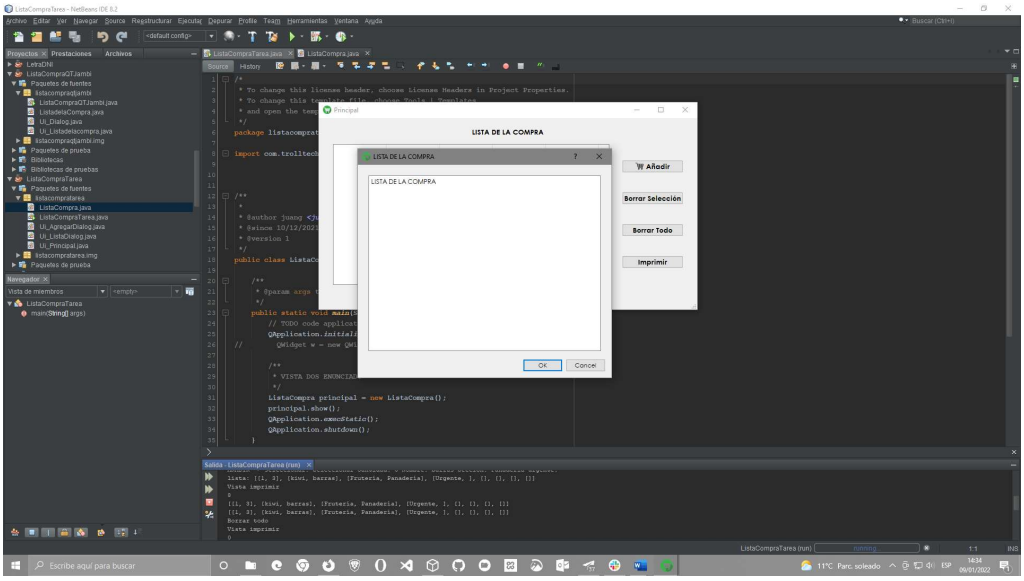
Detalle listado e impresión.



Detalle de borrar producto seleccionado.



Detalle de impresión del listado tras editarlo.



Detalle impresión listado tras borrar todo.