

INFORME DE ELABORACIÓN DE LA APLICACIÓN LISTA DE LA COMPRA

1. Diseño de la interfaz en QtJambi Designer:

I. Diseño de la pantalla principal (Main Window):

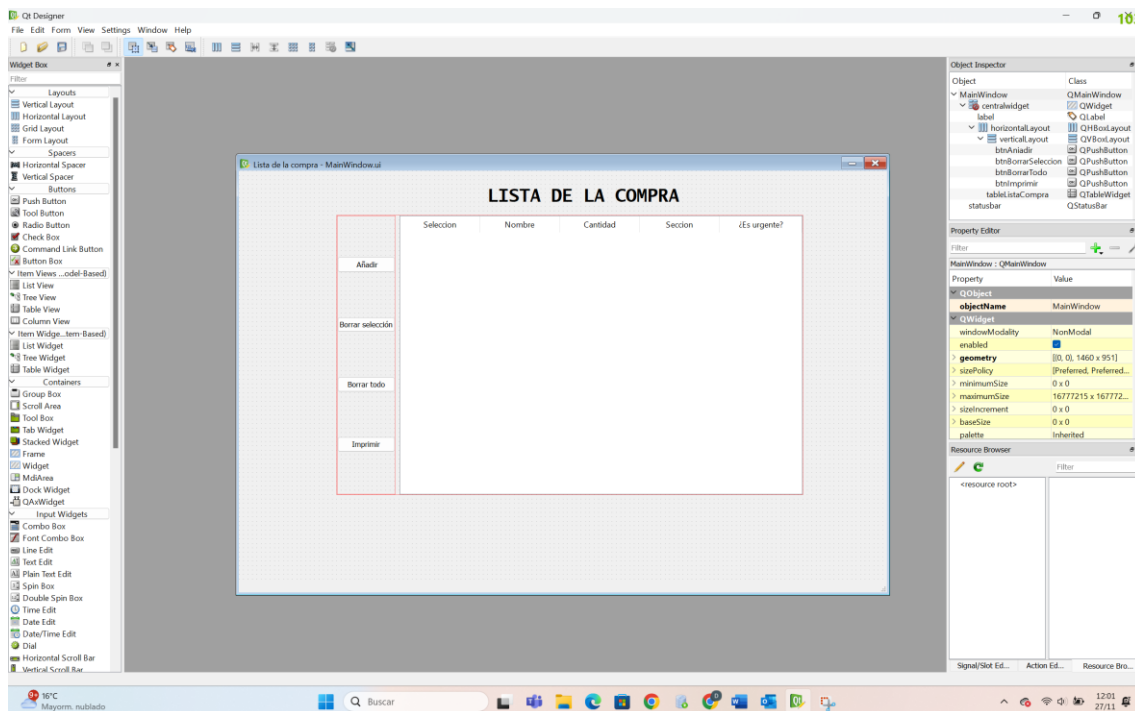
- **Tabla de la Lista de la Compra:** Usaremos QTableWidget con las siguientes columnas:
 - **Checkbox (para selección)**
 - **Nombre del producto**
 - **Cantidad**
 - **Sección del super**
 - **Urgente**
- **Botones:**
 - **"Borrar Todo":** Un botón para borrar todos los elementos de la tabla.
 - **"Borrar Selección":** Un botón para borrar solo las filas seleccionadas.
 - **"Añadir":** Un botón que abrirá una ventana secundaria para añadir un nuevo producto.
 - **"Imprimir":** Un botón que abrirá una ventana que simula la impresión en formato texto.

Ventana de Añadir Producto:

- Campos para capturar:
 - **Cantidad:** QSpinBox para valores numéricos positivos.
 - **Nombre:** QLineEdit.
 - **Sección del super:** QComboBox con las opciones predefinidas.
 - **Urgente:** QCheckBox.
- Botón para confirmar el añadido del producto con una imagen.

Ventana de Simulación de Impresión:

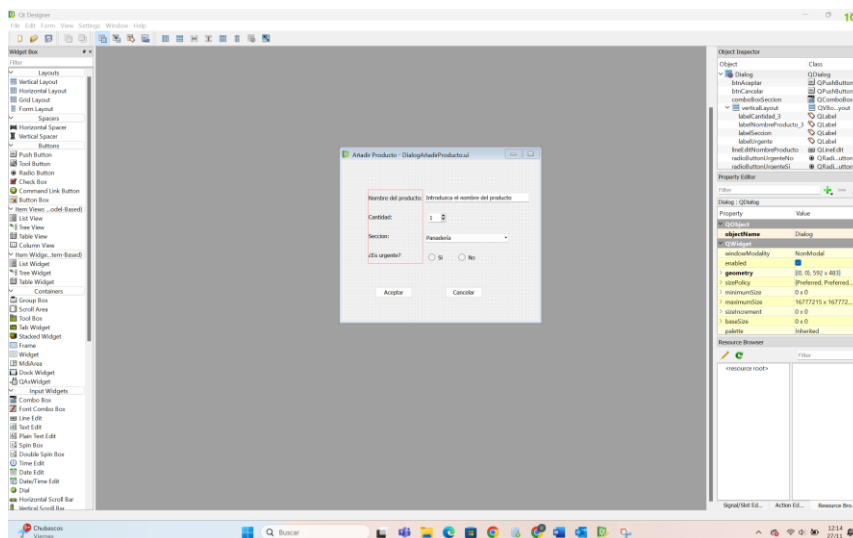
- **Cuadro de texto:** QTextEdit que muestra los elementos de la tabla como una lista de texto.



II. Diseño de la interfaz DialogAñadirProducto:

Se usaron los siguientes QWidgets:

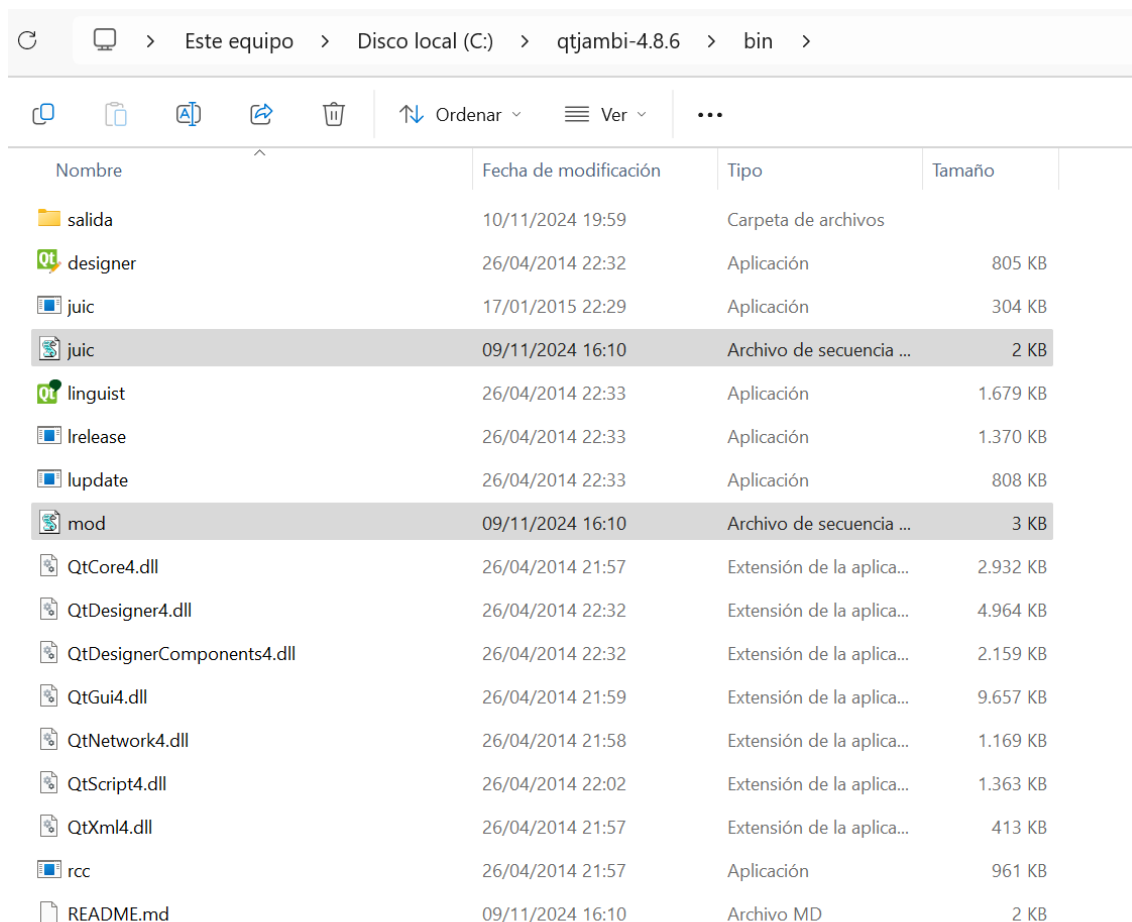
- **Nombre del producto:** Un QLineEdit para introducir texto.
- **Cantidad:** Un QSpinBox para seleccionar la cantidad.
- **Sección:** Un QComboBox para seleccionar la Sección del supermercado.
- **Dos botones:**
 - **“Aceptar”:** Confirma la selección del producto y carga la información en la tabla.
 - **“Cancelar”:** Cancela la creación del producto y vuelve a la pantalla principal.



2. Conversión de los archivos .ui en archivos .java:

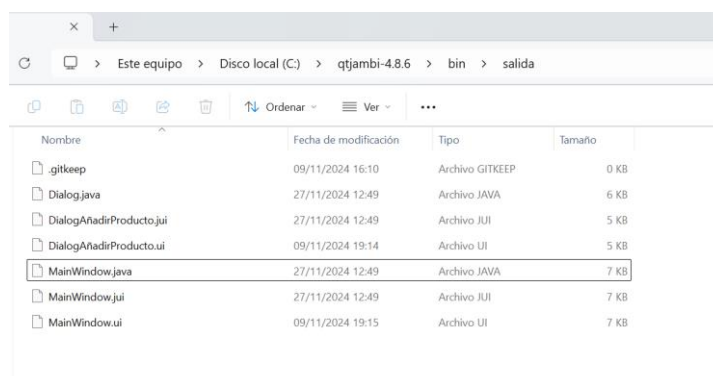
Al guardar los archivos en QtJambi genera dos xml con ambas interfaces. Para poder trabajar con estas en NetBeans, es necesario convertirlas a archivos java primero. Para ello, se han usado dos scripts que he localizado por la web:

- Script juic: transforma los archivos .ui en archivos .jui.
- Script mod: transforma los archivos .jui en archivos .java.



Nombre	Fecha de modificación	Tipo	Tamaño
salida	10/11/2024 19:59	Carpeta de archivos	
designer	26/04/2014 22:32	Aplicación	805 KB
juic	17/01/2015 22:29	Aplicación	304 KB
juic	09/11/2024 16:10	Archivo de secuencia ...	2 KB
linguist	26/04/2014 22:33	Aplicación	1.679 KB
lrelease	26/04/2014 22:33	Aplicación	1.370 KB
lupdate	26/04/2014 22:33	Aplicación	808 KB
mod	09/11/2024 16:10	Archivo de secuencia ...	3 KB
QtCore4.dll	26/04/2014 21:57	Extensión de la aplica...	2.932 KB
QtDesigner4.dll	26/04/2014 22:32	Extensión de la aplica...	4.964 KB
QtDesignerComponents4.dll	26/04/2014 22:32	Extensión de la aplica...	2.159 KB
QtGui4.dll	26/04/2014 21:59	Extensión de la aplica...	9.657 KB
QtNetwork4.dll	26/04/2014 21:58	Extensión de la aplica...	1.169 KB
QtScript4.dll	26/04/2014 22:02	Extensión de la aplica...	1.363 KB
QtXml4.dll	26/04/2014 21:57	Extensión de la aplica...	413 KB
rcc	26/04/2014 21:57	Aplicación	961 KB
README.md	09/11/2024 16:10	Archivo MD	2 KB

Estos scripts transforman los archivos .ui que se han ubicado previamente en la carpeta salida. Obteniendo así los archivos con los que podemos trabajar ya directamente en NetBeans.

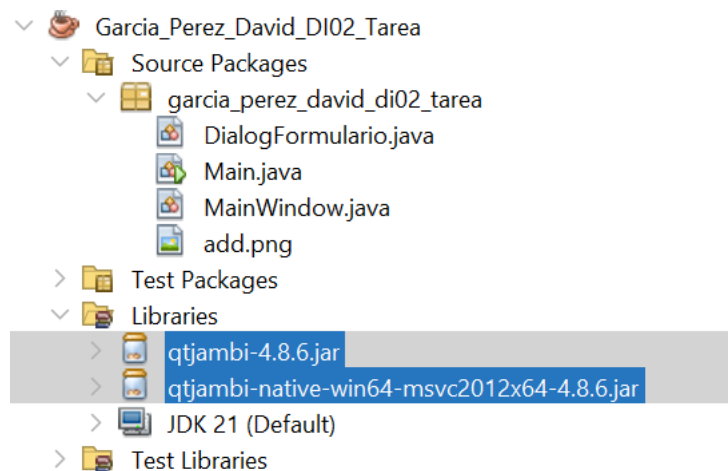


Nombre	Fecha de modificación	Tipo	Tamaño
.gitkeep	09/11/2024 16:10	Archivo GITKEEP	0 KB
Dialog.java	27/11/2024 12:49	Archivo JAVA	6 KB
DialogAñadirProducto.jui	27/11/2024 12:49	Archivo JUI	5 KB
DialogAñadirProducto.ui	09/11/2024 19:14	Archivo UI	5 KB
MainWindow.java	27/11/2024 12:49	Archivo JAVA	7 KB
MainWindow.jui	27/11/2024 12:49	Archivo JUI	7 KB
MainWindow.ui	09/11/2024 19:15	Archivo UI	7 KB

3. Creación del proyecto en NetBeans

Una vez obtenidas las clases .java con las interfaces elaboradas en Qrjambi, pasamos a crear el proyecto java para asignar la funcionalidad de los componentes de esta. Para poder utilizar las clases MainWindow y Dialog generadas en QtJambi, simplemente las arrastramos hasta nuestro proyecto e introducimos la línea de package en nuestro código.

Antes de nada, para poder trabajar con el código generado y que no aparezcan los errores, es necesario añadir las librerías correspondientes:



A continuación, implementamos la lógica del programa:

I. Clase principal (Main):

Se encarga de inicializar la aplicación mostrando la ventana principal (Main Window) y ejecutando el bucle de eventos de esta.

```
package garcia_perez_david_di02_tarea;

import com.trolltech.qt.gui.QApplication;

/**
 * Clase principal de la aplicación.
 *
 * Esta clase contiene el método 'main' que sirve como punto de entrada para la ejecución de la aplicación Qt.
 * Inicializa la aplicación Qt, crea la ventana principal (MainWindow) y luego ejecuta el bucle de eventos de la aplicación.
 */

public class Main {

    public static void main(String[] args) {
        // Inicializar la aplicación Qt con los argumentos proporcionados
        QApplication.initialize(args);

        // Crear la ventana principal
        MainWindow principal = new MainWindow();
        // Configurar la interfaz de usuario de la ventana principal
        principal.setupUi(principal);
        // Mostrar la ventana principal en pantalla
        principal.show();
        // Ejecutar el bucle de eventos de la aplicación Qt
        QApplication.execStatic();
    }
}
```

II. Clase Main Window (Ventana principal de la aplicación)

En esta clase se ha incorporado la funcionalidad a cada uno de los botones. Para ello se han creado distintos métodos:

○ **mostrarFormulario():**

```
/** Abre un formulario modal para añadir un nuevo producto a la lista de compra.
 *
 * Este método crea una instancia de la clase `DialogFormulario`, configura su interfaz y lo muestra como un diálogo modal.
 * Si el usuario confirma la acción (haciendo clic en el botón "Aceptar" del formulario), se obtienen los datos introducidos,
 * y estos se añaden a la tabla de productos mediante el método `agregarProductoATabla()`.
 */

public void mostrarFormulario() {
    DialogFormulario dialog = new DialogFormulario();
    dialog.setupUi(dialog);
    if (dialog.exec() == QDialog.DialogCode.Accepted.value()) {
        String[] datos = dialog.obtenerDatosProducto();
        agregarProductoATabla(datos);
    }
}
```

○ **agregarProductoATabla():**

```
//Método para agregar el producto que se ha rellenado en el formulario a la tabla.
/**
 * Agrega un nuevo producto a la tabla `tableListaCompra`.
 *
 * Este método inserta una nueva fila en la tabla y añade un `QCheckBox` en la primera columna
 * para marcar o desmarcar el producto. A continuación, los datos del producto se insertan en las
 * siguientes columnas de la tabla. Los datos se proporcionan como un arreglo de cadenas (`String[]`).
 *
 * @param datos Un arreglo de `String` que contiene la información del producto:
 * - datos[0]: Nombre del producto
 * - datos[1]: Cantidad del producto
 * - datos[2]: Sección (categoría) del producto
 * - datos[3]: Urgencia del producto (por ejemplo, "Sí" o "No")
 */

private void agregarProductoATabla(String[] datos) {
    // Obtener el número actual de filas en la tabla
    int row = tableListaCompra.rowCount();

    // Insertar una nueva fila al final de la tabla
    tableListaCompra.insertRow(row);

    // Crear el QCheckBox y agregarlo en la primera celda de la fila
    QCheckBox checkBox = new QCheckBox();
    checkBox.setChecked(false); // Estado inicial: desmarcado
    tableListaCompra.setCellWidget(row, 0, checkBox); // Agregar el checkbox en la primera celda.

    // Rellenar las celdas de la fila con los datos proporcionados
    for (int col = 0; col < datos.length; col++) {
        // Insertar los datos a partir de la segunda columna (col + 1)
        tableListaCompra.setItem(row, col+1, new QTableWidgetItem(datos[col]));
    }
}
```

- **borrarTodo():**

```
/**
 * Elimina todos los elementos de la tabla `tableListaCompra`.
 *
 * Este método borra el contenido de todas las celdas de la tabla sin eliminar la estructura de las columnas.
 * Después de borrar los contenidos, también establece el número de filas en 0, eliminando todas las filas existentes.
 */
public void borrarTodo() {
    // Elimina el contenido de todas las celdas de la tabla, manteniendo la estructura de las columnas
    tableListaCompra.clearContents();

    // Ajusta el número de filas a 0, eliminando todas las filas de la tabla
    tableListaCompra.setRowCount(0);
}
```

- **borrarSeleccion():**

```
/**
 * Elimina todas las filas seleccionadas de la tabla `tableListaCompra`.
 *
 * Recorre cada fila de la tabla desde la última hasta la primera, revisando si el
 * QCheckBox en la primera columna está marcado. Si está marcado, elimina la fila correspondiente.
 *
 * Este método sirve para eliminar uno o varios elementos de la tabla a la vez,
 * basándose en la selección del usuario.
 */
public void borrarSeleccion() {
    // Recorre la tabla desde la última fila hasta la primera para evitar problemas de indexación
    for (int i = tableListaCompra.rowCount() - 1; i >= 0; i--) {

        // Obtiene el widget QCheckBox de la primera columna de la fila actual
        QCheckBox checkBox = (QCheckBox) tableListaCompra.cellWidget(i, 0);
        // Si el checkbox está marcado, elimina la fila
        if (checkBox.isChecked()) {
            tableListaCompra.removeRow(i);
        }
    }
}
```

- **mostrarProductos():**

```
/**
 * Muestra un cuadro de diálogo con la lista de productos en la tabla `tableListaCompra`.
 *
 * Este método recorre todas las filas de la tabla y extrae los detalles del producto
 * (nombre, cantidad, sección y urgencia). Luego, crea un mensaje con estos detalles y
 * lo muestra en un QMessageBox con el título "Lista de Compra".
 *
 * Si la tabla está vacía, el cuadro de diálogo simplemente mostrará el mensaje sin detalles.
 */
public void mostrarProductos() {
    // Obtener los datos de la tabla en formato de texto
    StringBuilder productosTexto = new StringBuilder();
    // Recorrer todas las filas de la tabla para recopilar la información
    for (int fila = 0; fila < tableListaCompra.rowCount(); fila++) {
        String producto = tableListaCompra.item(fila, 1).text(); //Columna de nombre del producto
        String cantidad = tableListaCompra.item(fila, 2).text(); //Columna de cantidad
        String seccion = tableListaCompra.item(fila, 3).text(); //Columna de seccion
        String urgente = tableListaCompra.item(fila, 4).text(); //Columna de urgencia

        // Construir la cadena de texto con la información del producto
        productosTexto.append("Producto: ").append(producto)
            .append(", Cantidad: ").append(cantidad).append(", Seccion: ").append(seccion).append(", Es Urgente: ")
            .append(urgente).append("\n");
    }

    // Crear y configurar el QMessageBox para mostrar la información
    QMessageBox messageBox = new QMessageBox(this);
    messageBox.setWindowTitle("Lista de Compra"); //Establece el titulo de la ventana
    messageBox.setText("Productos en la lista de compra:"); //Introduce texto en el cuadro de mensaje
    messageBox.setInformativeText(productosTexto.toString()); // Agregar la lista de productos al cuadro de mensaje
    messageBox.setIcon(QMessageBox.Icon.Information); //Icono de información
    messageBox.exec(); // Muestra el cuadro de diálogo.
}
```

Para insertar una imagen en el botón “Añadir” hacemos uso del método `setIcon()`:

```
btnAniadir = new QPushButton(layoutWidget);
btnAniadir.setObjectName("btnAniadir");
//Se incluye la ruta del directorio del proyecto donde se encuentra el icono.
btnAniadir.setIcon(new QIcon("src\\garcia_perez_david_di02_tarea\\add.png"));
```

Por último, asignamos los métodos que queremos que se inicien a cada botón:

```
/**
 * Conecta los botones de la interfaz a los métodos correspondientes para manejar eventos.
 *
 * - btnAniadir: Abre un formulario para añadir un nuevo producto.
 * - btnBorrarSeleccion: Elimina los elementos seleccionados en la tabla.
 * - btnBorrarTodo: Borra todos los elementos de la lista de compra.
 * - btnImprimir: Muestra un cuadro de diálogo con los productos en la lista de compra.
 */
btnAniadir.clicked.connect(this, "mostrarFormulario()");
btnBorrarSeleccion.clicked.connect(this, "borrarSeleccion()");
btnBorrarTodo.clicked.connect(this, "borrarTodo()");
btnImprimir.clicked.connect(this, "mostrarProductos()");
```

III. Clase DialogFormulario (Ventana del formulario para añadir productos)

Esta clase incluye la interfaz del formulario para crear el producto. En esta clase se añadirá un método para que recoja los valores introducidos y devuelva un array de String, que se maneja por el método `mostrarFormulario()` para insertar los datos a la tabla.

○ `obtenerDatosProducto()`:

```
/**
 * Método para obtener los datos del producto desde los controles de la interfaz.
 *
 * Este método recoge los valores ingresados por el usuario en los campos del formulario:
 * el nombre del producto, la cantidad, la sección y si es urgente. Si el nombre del producto
 * está vacío, muestra un mensaje de advertencia y retorna 'null' para indicar que no se puede
 * obtener la información. Si los datos son válidos, los devuelve en un arreglo de cadenas.
 *
 * @return Un arreglo de cadenas que contiene el nombre del producto, la cantidad, la sección
 * y si es urgente. Si el nombre del producto está vacío, se devuelve 'null'.
 */
public String[] obtenerDatosProducto() {

    String nombre = lineEditNombreProducto.text();
    if (nombre.isEmpty()) {
        // Mostrar un mensaje de error o advertencia si el nombre está vacío
        QMessageBox.warning(this, "Advertencia", "El nombre del producto no puede estar vacío.");
        return null;
    }

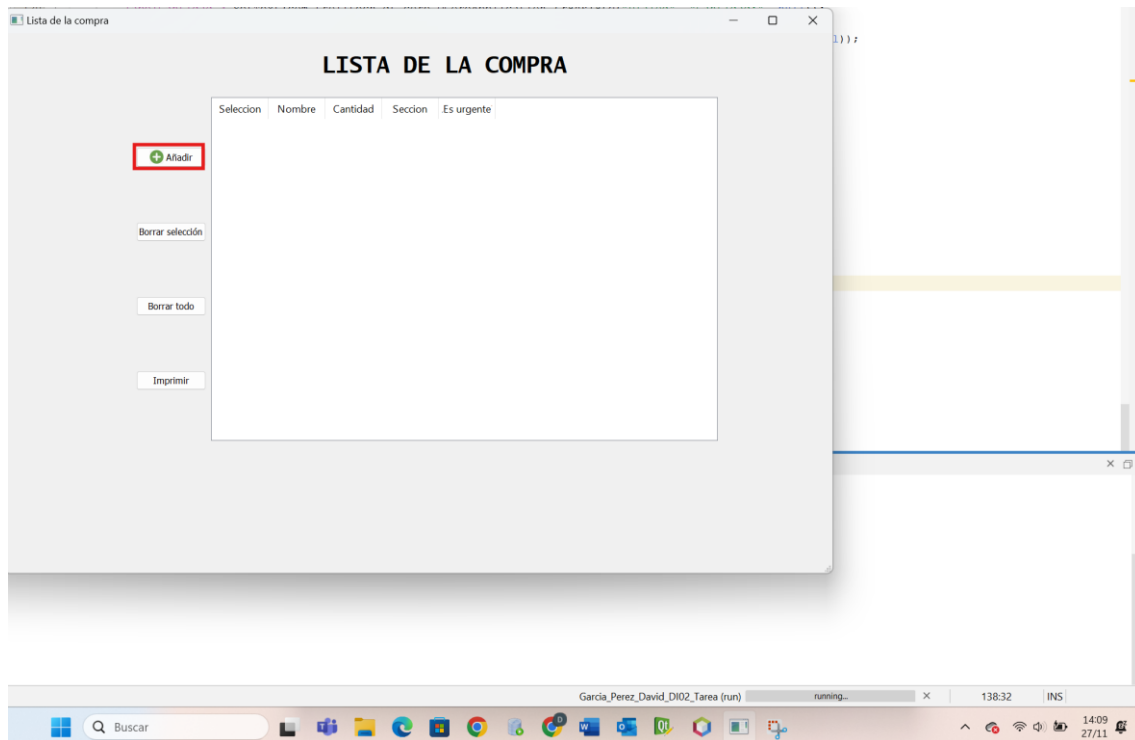
    int cantidad = spinBoxCantidad.value();
    String seccion = comboBoxSeccion.currentText();
    String urgente = radioButtonUrgenteSi.isChecked() ? "Si" : "No";

    System.out.println("Nombre: " + nombre);
    System.out.println("Cantidad: " + cantidad);
    System.out.println("Sección: " + seccion);
    System.out.println("Urgente: " + urgente);

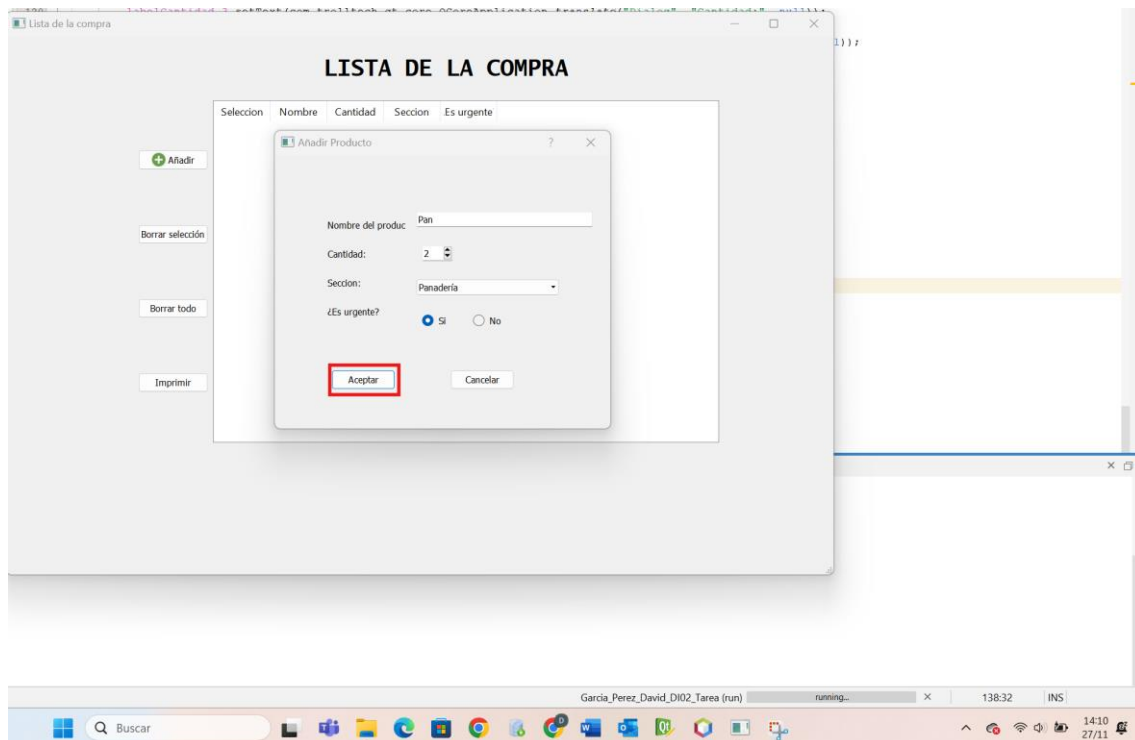
    return new String[]{nombre, String.valueOf(cantidad), seccion, urgente};
}
```

4. Prueba de la aplicación

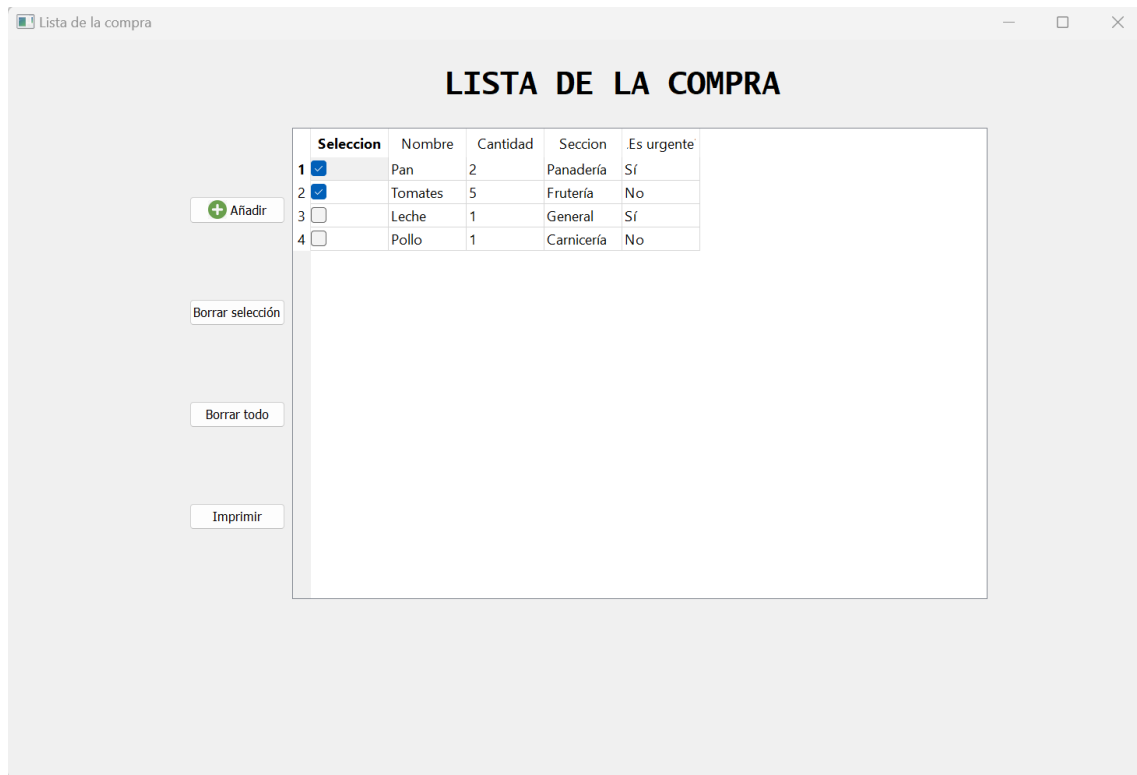
Si pulsamos el botón “Añadir” se abre la ventana Añadir Producto.



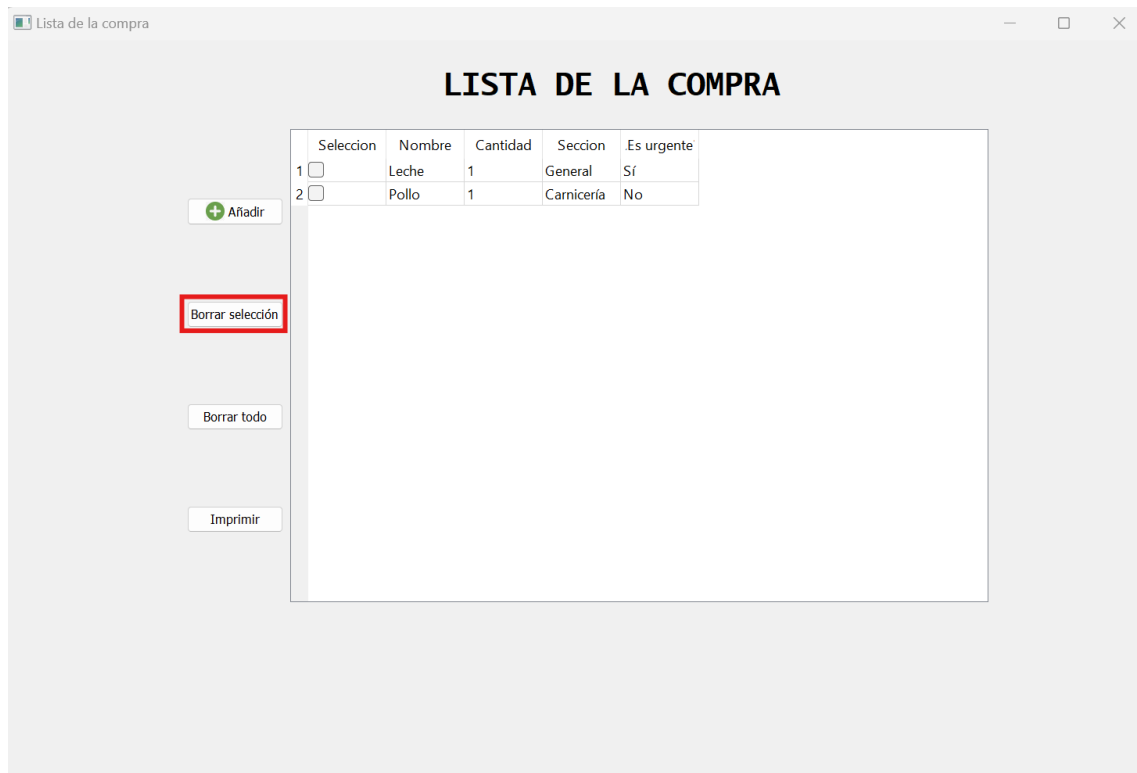
Si rellenamos los campos correspondientes y damos al botón aceptar, se cargan los datos del producto en la tabla.



Continuamos rellenando la con más productos para comprobar la funcionalidad del resto de botones.



Marcando el checkbox de seleccion y pulsando el botón “Borrar selección”, se borran los productos marcados:



Si pulsamos el botón “Borrar todo”, se borran todos los productos introducidos en la tabla.



Si pulsamos el botón “Imprimir” se muestran los productos de la tabla:

