**Manual De Usuario: Cómo Crear Una App Desde Cero En Netbeans Usando Java Fx**

**1. Configuración del proyecto**

Antes de comenzar, asegúrate de tener las siguientes herramientas instaladas:

* JDK: Instala Java Development Kit, versión 13 o superior.
* JavaFX SDK: Descárgalo en la web desde su página principal
* Apache POI: Biblioteca para manejar archivos Excel.

### Además, instala un IDE como:

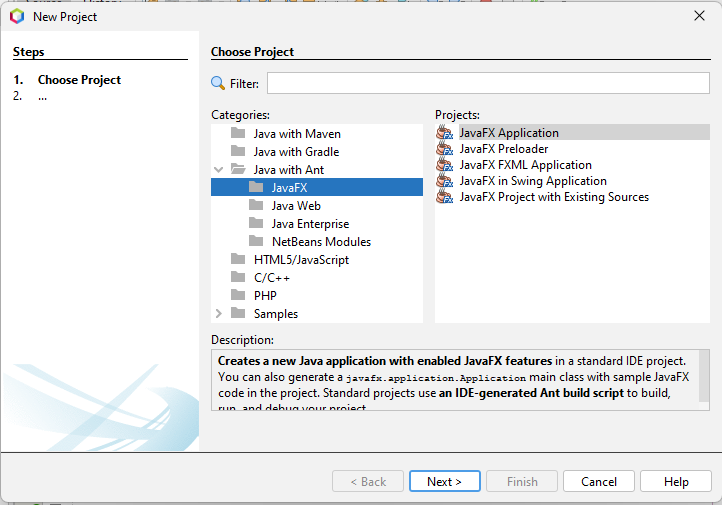
* **NetBeans**
* Configura las bibliotecas necesarias siguiendo la documentación del IDE.

## ****2. Crear el proyecto base****

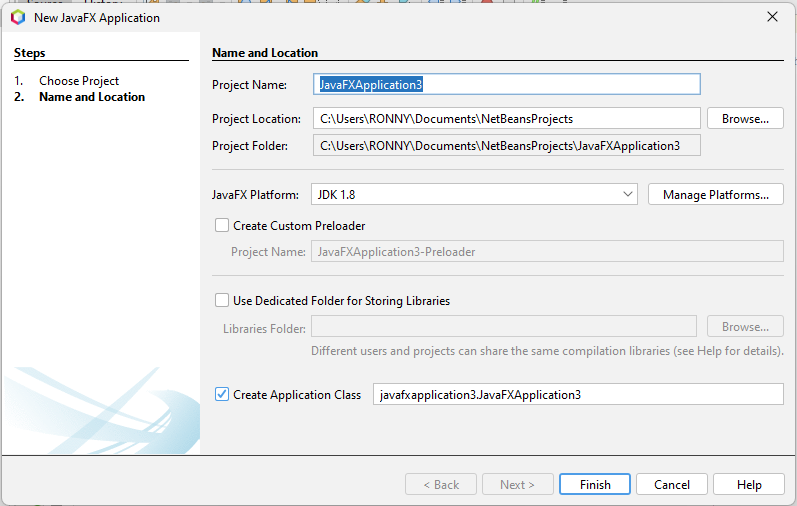
### Pasos para configurar el proyecto:

1. Crea un nuevo proyecto Java en tu IDE.

CVCV

* Una vez que hayas seleccionado **New Project** y que se te despliegue la siguiente ventana asegúrate de tener seleccionado **Javafx** en la sección **Categories** y junto con **Javafx Application** en la sección **Project.**

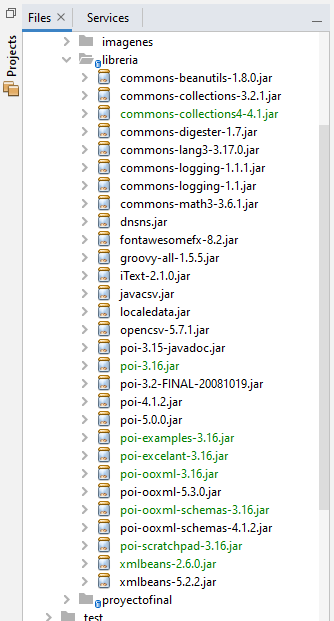
CV

* Luego de haber tenido estas opciones seleccionadas, le das clic en **Next.**
* ****Se te abrirá la siguiente ventana, en donde pondrás el nombre de tu proyecto **(Project Name)**, en nuestro caso es Proyectofinal.

CV

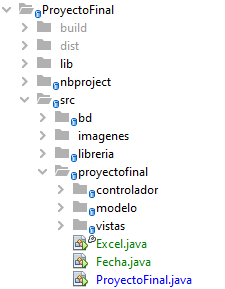
* Luego de haber escrito el nombre de tu proyecto, le das clic en **Finish.**

1. Configura JavaFX agregando las bibliotecas necesarias:



## ****3. Estructura del proyecto****

Organiza los archivos del proyecto de la siguiente manera:



* En la carpeta principal del proyecto final, ubicamos la carpeta la cual se llama **src**, luego le damos clic en la subcarpeta **proyectofinal.**
* Al abrir esta subcarpeta se te desplegaran los siguientes archivos:
* ***Controlador:*** Es el número de clases Java que gestionan eventos de las vistas FXML.
* ***modelo:*** Es el número de clases de lógica de negocio (Producto, Fecha, etc.)
* ***vistas:*** Es el número de Archivos FXML que definen las interfaces gráficas
* ***ProyectoFinal:*** # Clase principal

## ****4. Creación de las vistas gráficas****

### **Para crear un fxml de en el proyecto, debes crear primero un empaquetado (source packages) denominado como vistas.**

### 

### **Luego dentro de este empaquetado comienzas a crear los fxml como se muestra en la gráfica, denominándolo con el nombre que desees asignarle, en nuestro caso login, carrito, etc.**

### 

### Después de crear todos los fxml necesarios, puedes empezar a diseñar en ****Scene Builder,** guardándose de tal manera en cada vista todas las modificaciones y arreglos que agregues en él. En nuestro caso sería de esta manera :**

## ****1. Login.fxml****

### Código FXML

### <AnchorPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="proyectofinal.controlador.LoginController">

### <children>

### <Label layoutX="250.0" layoutY="50.0" text="Iniciar Sesión" style="-fx-font-size: 20;" />

### <TextField fx:id="txtUsuario" layoutX="200.0" layoutY="120.0" promptText="Usuario" />

### <PasswordField fx:id="txtPassword" layoutX="200.0" layoutY="180.0" promptText="Contraseña" />

### <Button fx:id="btnLogin" layoutX="250.0" layoutY="240.0" text="Ingresar" />

### <Hyperlink fx:id="linkRegistro" layoutX="260.0" layoutY="290.0" text="Crear cuenta" />

### </children>

### </AnchorPane>

### Explicación:

1. **AnchorPane**: Contenedor principal de 400 px por 600 px.
2. **Label**: Título que dice "Iniciar Sesión".
3. **TextField (txtUsuario)**: Campo para ingresar el nombre de usuario.
4. **PasswordField (txtPassword)**: Campo para ingresar la contraseña de forma enmascarada.
5. **Button (btnLogin)**: Botón que ejecuta la acción de iniciar sesión.
6. **Hyperlink (linkRegistro)**: Permite redirigir a una vista de registro.

## ****2. Carrito.fxml****

### Código FXML

### <AnchorPane prefHeight="600.0" prefWidth="800.0" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="proyectofinal.controlador.CarritoController">

### <children>

### <Label layoutX="50.0" layoutY="20.0" text="Carrito de Compras" style="-fx-font-size: 18;" />

### <TableView fx:id="tablaCarrito" layoutX="50.0" layoutY="60.0" prefHeight="400.0" prefWidth="700.0">

### <columns>

### <TableColumn fx:id="colProducto" text="Producto" />

### <TableColumn fx:id="colPrecio" text="Precio" />

### <TableColumn fx:id="colCantidad" text="Cantidad" />

### </columns>

### </TableView>

### <Button fx:id="btnComprar" layoutX="650.0" layoutY="480.0" text="Comprar" />

### </children>

### </AnchorPane>

### Explicación:

1. **Label**: Encabezado para indicar que es el carrito.
2. **TableView (tablaCarrito)**: Tabla con tres columnas para mostrar los productos seleccionados:
   * **Producto**: Nombre del producto.
   * **Precio**: Precio por unidad.
   * **Cantidad**: Número de unidades seleccionadas.
3. **Button (btnComprar)**: Botón para proceder con la compra.

## ****3. ListaDeDeseos.fxml****

### Código FXML

***<AnchorPane prefHeight="600.0" prefWidth="800.0" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="proyectofinal.controlador.ListaDeseosController">***

***<children>***

***<Label layoutX="50.0" layoutY="20.0" text="Lista de Deseos" style="-fx-font-size: 18;" />***

***<TableView fx:id="tablaDeseos" layoutX="50.0" layoutY="60.0" prefHeight="400.0" prefWidth="700.0">***

***<columns>***

***<TableColumn fx:id="colProducto" text="Producto" />***

***<TableColumn fx:id="colPrecio" text="Precio" />***

***<TableColumn fx:id="colNotas" text="Notas" />***

***</columns>***

***</TableView>***

***<Button fx:id="btnAgregarCarrito" layoutX="50.0" layoutY="480.0" text="Agregar al Carrito" />***

***</children>***

***</AnchorPane>***

### Explicación:

1. **TableView (tablaDeseos)**: Tabla para listar los productos guardados como favoritos.
2. **Button (btnAgregarCarrito)**: Botón para mover productos de la lista de deseos al carrito.

## ****4. OpcionesUsuario.fxml****

### Código FXML

***<AnchorPane prefHeight="400.0" prefWidth="600.0" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="proyectofinal.controlador.OpcionesUsuarioController">***

***<children>***

***<Label layoutX="50.0" layoutY="20.0" text="Opciones de Usuario" style="-fx-font-size: 18;" />***

***<Button fx:id="btnEditarPerfil" layoutX="200.0" layoutY="100.0" text="Editar Perfil" />***

***<Button fx:id="btnCambiarPassword" layoutX="200.0" layoutY="160.0" text="Cambiar Contraseña" />***

***<Button fx:id="btnCerrarSesion" layoutX="200.0" layoutY="220.0" text="Cerrar Sesión" />***

***</children>***

***</AnchorPane>***

### Explicación:

1. **Button (btnEditarPerfil)**: Para editar la información del perfil.
2. **Button (btnCambiarPassword)**: Para cambiar la contraseña del usuario.
3. **Button (btnCerrarSesion)**: Para salir de la sesión actual.

## ****5. Inicio.fxml****

### Código FXML

***<AnchorPane prefHeight="600.0" prefWidth="800.0" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml" fx:controller="proyectofinal.controlador.InicioController">***

***<children>***

***<Label layoutX="50.0" layoutY="20.0" text="Bienvenido a la Tienda" style="-fx-font-size: 24;" />***

***<ListView fx:id="listaCategorias" layoutX="50.0" layoutY="80.0" prefHeight="400.0" prefWidth="200.0" />***

***<TableView fx:id="tablaProductos" layoutX="300.0" layoutY="80.0" prefHeight="400.0" prefWidth="450.0">***

***<columns>***

***<TableColumn fx:id="colProducto" text="Producto" />***

***<TableColumn fx:id="colPrecio" text="Precio" />***

***</columns>***

***</TableView>***

***</children>***

***</AnchorPane>***

### Explicación:

1. **ListView (listaCategorias)**: Muestra las categorías disponibles.
2. **TableView (tablaProductos)**: Muestra los productos según la categoría seleccionada.

## ****6. Logo.fxml****

### Código FXML

### <AnchorPane prefHeight="200.0" prefWidth="600.0" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml">

### <children>

### <ImageView layoutX="200.0" layoutY="50.0" fitHeight="100.0" fitWidth="200.0" image="@../../images/logo.png" />

### </children>

### </AnchorPane>

### Explicación:

1. **ImageView**: Muestra el logotipo de la tienda. La imagen debe estar en la carpeta resources/images.

## ****7. CardProduct.fxml****

### Código FXML

### <AnchorPane prefHeight="150.0" prefWidth="200.0" xmlns="http://javafx.com/javafx" xmlns:fx="http://javafx.com/fxml">

### <children>

### <ImageView fx:id="imagenProducto" layoutX="10.0" layoutY="10.0" fitHeight="100.0" fitWidth="180.0" />

### <Label fx:id="nombreProducto" layoutX="10.0" layoutY="120.0" text="Producto" />

### <Label fx:id="precioProducto" layoutX="150.0" layoutY="120.0" text="$0.00" />

### </children>

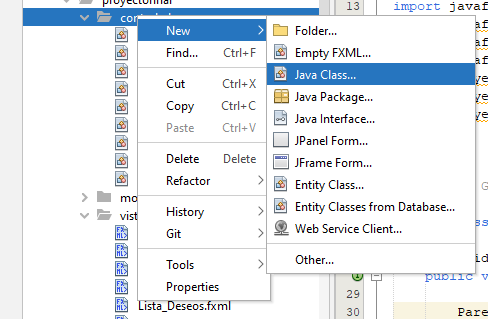
### </AnchorPane>

### Explicación:

1. **ImageView (imagenProducto)**: Muestra la imagen del producto.
2. **Label (nombreProducto)**: Muestra el nombre del producto.
3. **Label (precioProducto)**: Muestra el precio del producto.

## ****5 Creación de las vistas gráficas****

### **Para crear un controlador en el proyecto, debes crear primero un empaquetado (source packages) denominado como controlador**



### **Luego dentro de este empaquetado comienzas a crear las clases (java class) como se muestra en la gráfica, denominándolo con el nombre que desees asignarle, en nuestro caso logincontroller, carritocontroller, etc.**

### Después de crear todas las clases controladoras necesarias, **En nuestro caso sería de esta manera:**

## ****1. Controlador de Login (LoginController.java)****

### Código

### package proyectofinal.controlador;

### import javafx.fxml.FXML;

### import javafx.scene.control.Button;

### import javafx.scene.control.PasswordField;

### import javafx.scene.control.TextField;

### import javafx.scene.control.Alert;

### import javafx.scene.control.Alert.AlertType;

### public class LoginController {

### // Campo de texto para el usuario

### @FXML

### private TextField txtUsuario;

### // Campo para la contraseña

### @FXML

### private PasswordField txtPassword;

### // Botón para iniciar sesión

### @FXML

### private Button btnLogin;

### // Método que se ejecuta al inicializar la vista

### @FXML

### private void initialize() {

### // Configuración del botón para validar credenciales al hacer clic

### btnLogin.setOnAction(event -> validarCredenciales());

### }

### // Método para validar el usuario y contraseña

### private void validarCredenciales() {

### String usuario = txtUsuario.getText(); // Obtiene el texto ingresado en el campo de usuario

### String password = txtPassword.getText(); // Obtiene el texto ingresado en el campo de contraseña

### // Validación básica: usuario "admin" y contraseña "1234"

### if ("admin".equals(usuario) && "1234".equals(password)) {

### mostrarAlerta(AlertType.INFORMATION, "Ingreso exitoso", "¡Bienvenido!"); // Muestra mensaje de éxito

### } else {

### mostrarAlerta(AlertType.ERROR, "Error de autenticación", "Usuario o contraseña incorrectos."); // Mensaje de error

### }

### }

### // Método para mostrar una alerta al usuario

### private void mostrarAlerta(AlertType tipo, String titulo, String mensaje) {

### Alert alerta = new Alert(tipo);

### alerta.setTitle(titulo); // Título de la ventana emergente

### alerta.setContentText(mensaje); // Mensaje que se mostrará

### alerta.showAndWait(); // Muestra la alerta y espera una respuesta

### }

### }

### Explicación

1. **txtUsuario**: Campo de entrada donde el usuario ingresa su nombre.
2. **txtPassword**: Campo para ingresar la contraseña, con el texto enmascarado.
3. **btnLogin**: Botón que ejecuta la lógica de validación al ser presionado.
4. **validarCredenciales**: Comprueba que el usuario y contraseña sean correctos.
5. **mostrarAlerta**: Muestra un mensaje de confirmación o error, dependiendo del resultado.

## ****2. Controlador de Carrito (CarritoController.java)****

### Código

### package proyectofinal.controlador;

### import javafx.fxml.FXML;

### import javafx.scene.control.Button;

### import javafx.scene.control.TableView;

### import javafx.scene.control.TableColumn;

### public class CarritoController {

### // Tabla que muestra los productos en el carrito

### @FXML

### private TableView<?> tablaCarrito;

### // Columna para el nombre del producto

### @FXML

### private TableColumn<?, ?> colProducto;

### // Columna para el precio del producto

### @FXML

### private TableColumn<?, ?> colPrecio;

### // Columna para la cantidad seleccionada

### @FXML

### private TableColumn<?, ?> colCantidad;

### // Botón para finalizar la compra

### @FXML

### private Button btnComprar;

### // Inicialización de la vista

### @FXML

### private void initialize() {

### btnComprar.setOnAction(event -> procesarCompra());

### }

### // Método que se ejecuta al presionar el botón de compra

### private void procesarCompra() {

### System.out.println("Procesando compra...");

### // Aquí se implementaría la lógica de compra (cálculos, confirmaciones, etc.)

### }

### }

### Explicación

1. **tablaCarrito**: Muestra los productos añadidos al carrito en forma de tabla.
2. **colProducto**: Columna que muestra el nombre de los productos.
3. **colPrecio**: Columna que muestra el precio de cada producto.
4. **colCantidad**: Indica cuántas unidades de cada producto se han seleccionado.
5. **btnComprar**: Permite procesar la compra de todos los productos en el carrito.

## ****3. Controlador de Inicio (InicioController.java)****

### Código

### package proyectofinal.controlador;

### import javafx.fxml.FXML;

### import javafx.scene.control.ListView;

### import javafx.scene.control.TableView;

### import javafx.scene.control.TableColumn;

### public class InicioController {

### // Lista que muestra las categorías de productos

### @FXML

### private ListView<String> listaCategorias;

### // Tabla que muestra los productos según la categoría seleccionada

### @FXML

### private TableView<?> tablaProductos;

### // Columna para el nombre del producto

### @FXML

### private TableColumn<?, ?> colProducto;

### // Columna para el precio del producto

### @FXML

### private TableColumn<?, ?> colPrecio;

### // Inicialización de la vista

### @FXML

### private void initialize() {

### // Añadir categorías a la lista

### listaCategorias.getItems().addAll("Electrónica", "Ropa", "Alimentos", "Hogar");

### // Configurar el evento al seleccionar una categoría

### listaCategorias.getSelectionModel().selectedItemProperty().addListener((obs, oldVal, newVal) -> cargarProductos(newVal));

### }

### // Método que carga los productos de una categoría

### private void cargarProductos(String categoria) {

### System.out.println("Cargando productos de la categoría: " + categoria);

### // Aquí se implementaría la lógica para cargar los productos de la categoría seleccionada

### }

### }

### Explicación

1. **listaCategorias**: Muestra todas las categorías disponibles (por ejemplo, Electrónica, Ropa, etc.).
2. **tablaProductos**: Tabla que despliega los productos correspondientes a una categoría.
3. **cargarProductos**: Filtra y muestra los productos según la categoría seleccionada.

## ****4. Controlador de Lista de Deseos (ListaDeseosController.java)****

### Código

### package proyectofinal.controlador;

### import javafx.fxml.FXML;

### import javafx.scene.control.Button;

### import javafx.scene.control.TableView;

### import javafx.scene.control.TableColumn;

### public class ListaDeseosController {

### // Tabla que muestra los productos en la lista de deseos

### @FXML

### private TableView<?> tablaDeseos;

### // Columna que muestra el nombre del producto

### @FXML

### private TableColumn<?, ?> colProducto;

### // Columna que muestra el precio del producto

### @FXML

### private TableColumn<?, ?> colPrecio;

### // Columna que muestra notas o comentarios sobre el producto

### @FXML

### private TableColumn<?, ?> colNotas;

### // Botón para agregar un producto de la lista de deseos al carrito

### @FXML

### private Button btnAgregarCarrito;

### // Inicialización de la vista

### @FXML

### private void initialize() {

### // Configura el botón para agregar productos al carrito

### btnAgregarCarrito.setOnAction(event -> agregarAlCarrito());

### }

### // Método para agregar productos al carrito

### private void agregarAlCarrito() {

### System.out.println("Producto añadido al carrito.");

### // Implementar lógica para mover productos de la lista de deseos al carrito

### }

### }

### Explicación

1. **tablaDeseos**: Muestra una lista de productos que el usuario desea guardar para futuras compras.
2. **colProducto**: Columna que contiene el nombre del producto.
3. **colPrecio**: Columna que muestra el precio del producto.
4. **colNotas**: Permite al usuario añadir comentarios sobre los productos.
5. **btnAgregarCarrito**: Permite al usuario mover productos desde la lista de deseos al carrito de compras.

## ****5. Controlador de Opciones de Usuario (OpcionesUsuarioController.java)****

### Código

### package proyectofinal.controlador;

### import javafx.fxml.FXML;

### import javafx.scene.control.Button;

### public class OpcionesUsuarioController {

### // Botón para editar el perfil del usuario

### @FXML

### private Button btnEditarPerfil;

### // Botón para cambiar la contraseña del usuario

### @FXML

### private Button btnCambiarPassword;

### // Botón para cerrar la sesión

### @FXML

### private Button btnCerrarSesion;

### // Inicialización de la vista

### @FXML

### private void initialize() {

### // Configuración de cada botón con su acción correspondiente

### btnEditarPerfil.setOnAction(event -> editarPerfil());

### btnCambiarPassword.setOnAction(event -> cambiarPassword());

### btnCerrarSesion.setOnAction(event -> cerrarSesion());

### }

### // Método para editar el perfil del usuario

### private void editarPerfil() {

### System.out.println("Editar perfil del usuario.");

### // Implementar lógica para modificar datos del perfil

### }

### // Método para cambiar la contraseña del usuario

### private void cambiarPassword() {

### System.out.println("Cambiar contraseña del usuario.");

### // Implementar lógica para cambiar la contraseña

### }

### // Método para cerrar sesión

### private void cerrarSesion() {

### System.out.println("Cerrar sesión del usuario.");

### // Implementar lógica para cerrar sesión y redirigir al login

### }

### }

### Explicación

1. **btnEditarPerfil**: Botón que permite al usuario modificar su información personal.
2. **btnCambiarPassword**: Botón que inicia el proceso para cambiar la contraseña del usuario.
3. **btnCerrarSesion**: Botón que cierra la sesión del usuario actual.

## ****6. Controlador de Logo (LogoController.java)****

### Código

### package proyectofinal.controlador;

### public class LogoController {

### // Esta clase puede permanecer vacía si el logo es un elemento visual estático

### // Si necesitas animaciones o interacción, se puede agregar lógica aquí

### }

### Explicación

1. **Este controlador es opcional y puede estar vacío** si el logo no requiere interacciones o lógica adicional.
2. Si se desean animaciones, cambios dinámicos o acciones al interactuar con el logo, este sería el lugar para implementarlas.

## ****7. Controlador de Card Producto (CardProductoController.java)****

### Código

### package proyectofinal.controlador;

### import javafx.fxml.FXML;

### import javafx.scene.control.Label;

### import javafx.scene.image.ImageView;

### public class CardProductoController {

### // Imagen del producto

### @FXML

### private ImageView imagenProducto;

### // Nombre del producto

### @FXML

### private Label nombreProducto;

### // Precio del producto

### @FXML

### private Label precioProducto;

### // Método para configurar los datos de un producto

### public void setDatosProducto(String nombre, String precio, String rutaImagen) {

### // Configura el nombre del producto

### nombreProducto.setText(nombre);

### // Configura el precio del producto

### precioProducto.setText(precio);

### // Configura la imagen del producto

### imagenProducto.setImage(new javafx.scene.image.Image(rutaImagen));

### }

### }

### Explicación

1. **imagenProducto**: Muestra una imagen visual del producto.
2. **nombreProducto**: Etiqueta que contiene el nombre del producto.
3. **precioProducto**: Etiqueta que contiene el precio del producto.
4. **setDatosProducto**: Método público que permite establecer los valores de un producto. Es útil para cargar dinámicamente la información en la vista.

## ****6. Empaquetado y ejecución****

### Para ejecutar el proyecto:

1. Compila y ejecuta desde el IDE, asegurándote de incluir las bibliotecas necesarias.

## ****7. Clase Principal (Main.java)****

Esta clase es el punto de entrada de la aplicación. Se encarga de inicializar la interfaz gráfica y definir las configuraciones básicas del entorno.

package proyectofinal;

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

// Método principal de Java para iniciar la aplicación

public static void main(String[] args) {

launch(args); // Llama a la aplicación JavaFX

}

// Método start de JavaFX, que configura y lanza la ventana principal

@Override

public void start(Stage stage) throws Exception {

// Carga la interfaz principal desde el archivo FXML

FXMLLoader loader = new FXMLLoader(getClass().getResource("/proyectofinal/vista/inicio.fxml"));

// Configura la escena con el diseño cargado

Scene scene = new Scene(loader.load());

// Configuración de la ventana principal

stage.setTitle("Proyecto Final - TiendaApp");

stage.setScene(scene);

stage.show(); // Muestra la ventana

}

}

### Explicación

1. **main**: Este método es el punto de entrada que invoca launch(), el método estándar de JavaFX para iniciar la aplicación.
2. **start**: Este método se ejecuta después de launch() y es donde se define la ventana principal.
3. **FXMLLoader**: Carga el diseño de la interfaz definido en el archivo inicio.fxml.
4. **Scene**: Representa la ventana principal de la aplicación. Aquí se define qué contenido se mostrará.
5. **stage.setTitle**: Configura el título de la ventana principal.
6. **stage.show()**: Muestra la ventana gráfica.

## ****9. Clase para Exportación a Excel (ExcelUtil.java)****

Esta clase se encarga de gestionar la exportación de datos desde la aplicación hacia un archivo Excel.

package proyectofinal.util;

import org.apache.poi.ss.usermodel.\*;

import org.apache.poi.xssf.usermodel.XSSFWorkbook;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.List;

public class ExcelUtil {

// Método para exportar datos a un archivo Excel

public static void exportarAExcel(String nombreArchivo, List<String[]> datos) throws IOException {

// Crear un libro de trabajo

Workbook workbook = new XSSFWorkbook();

// Crear una hoja dentro del libro

Sheet sheet = workbook.createSheet("Datos Exportados");

// Agregar datos a la hoja

int filaIndex = 0;

for (String[] fila : datos) {

Row row = sheet.createRow(filaIndex++);

int celdaIndex = 0;

for (String celda : fila) {

Cell cell = row.createCell(celdaIndex++);

cell.setCellValue(celda);

}

}

// Escribir el archivo a disco

try (FileOutputStream outputStream = new FileOutputStream(nombreArchivo)) {

workbook.write(outputStream);

}

// Cerrar el libro de trabajo para liberar recursos

workbook.close();

}

}

### Explicación

1. **Workbook**: Representa el archivo Excel completo. En este caso, usamos XSSFWorkbook, que genera archivos .xlsx.
2. **Sheet**: Una hoja dentro del libro Excel donde se escribirán los datos.
3. **Row y Cell**: Representan las filas y celdas dentro de la hoja, respectivamente.
4. **List<String[]> datos**: Lista de arrays de cadenas que contienen los datos a exportar.
5. **FileOutputStream**: Crea el archivo Excel en el sistema, escribiendo los datos en él.
6. **Uso del método**:
   * Se pasa el nombre del archivo como nombreArchivo.
   * datos contiene las filas y celdas que se desean exportar.

## ****3. Clase Java para Operaciones Generales (JavaUtil.java)****

Esta clase contiene métodos auxiliares para operaciones generales en la aplicación. Es una especie de "caja de herramientas".

### Código

package proyectofinal.util;

import java.text.SimpleDateFormat;

import java.util.Date;

public class JavaUtil {

// Método para obtener la fecha actual formateada

public static String obtenerFechaActual() {

SimpleDateFormat formatoFecha = new SimpleDateFormat("yyyy-MM-dd");

return formatoFecha.format(new Date());

}

// Método para validar una cadena no vacía

public static boolean validarCadena(String cadena) {

return cadena != null && !cadena.trim().isEmpty();

}

// Método para formatear un precio como moneda

public static String formatearComoMoneda(double cantidad) {

return String.format("$%.2f", cantidad);

}

}

### Explicación

1. **obtenerFechaActual**:
   * Devuelve la fecha actual en formato yyyy-MM-dd.
   * Utiliza la clase SimpleDateFormat para formatear la fecha.
2. **validarCadena**:
   * Comprueba que una cadena no sea nula ni esté vacía.
   * Es útil para evitar errores de datos faltantes o incorrectos.
3. **formatearComoMoneda**:
   * Convierte un valor numérico en formato de moneda (e.g., $123.45).
   * Utiliza el método String.format para aplicar el formato.