

**2º D.A.M.**

**Programación Multimedia y Dispositivos Móviles**

**Paso a Paso - Cafetería**

**Kevin Martínez Leiva**

## Índice

Base de Datos.....	3
Esquema inicial de la aplicación.....	5
Acceso a MySQL desde Android Studio.....	6
LoginActivityMain.....	8
ActivityPedirCafe.....	12
ActivityNuevoPedido.....	16
ActivityPanelControl.....	21
ActivityPedidos.....	24
DetallePedido.....	29
Ejemplos de ejecución.....	34

## Base de Datos

El primero paso es crear la base de datos con la que va a trabajar la aplicación de la Cafetería. Se trata de una base de datos en un servidor de MySQL con algunos datos de prueba para probar el funcionamiento de la aplicación.

Contiene las tablas

- Productos: almacena los datos de los productos que ofrece la cafetería.
- Usuarios: usuarios que pueden acceder a la aplicación; la categoría AD es para las cuentas de administradores y la categoría US es para usuarios corrientes que solo pueden realizar pedidos a la cafetería.
- Pedidos: datos administrativos de un pedido, donde el Estado puede ser 'curso' o 'finalizado' para distinguir entre los pedidos que están en curso en este momento y los que ya están finalizados.
- DetallePedidos: detalle de cada uno de los productos junto con su cantidad de cada producto que compone un pedido.

Script SQL de la base de datos:

```
-- Base de datos Cafeteria --
-- BBDD: Cafeteria, USER: UsuarioCafeteria, PASS: usuario

-- Borrado de tablas antes de insercion --
DROP TABLE IF EXISTS DetallePedidos;
DROP TABLE IF EXISTS Pedidos;
DROP TABLE IF EXISTS Productos;
DROP TABLE IF EXISTS Usuarios;

-- Tabla Productos --
CREATE TABLE Productos (
  Cod_producto char(3),
  Nombre varchar(15),
  Precio float,
  Leche boolean,
  CONSTRAINT pk_productos PRIMARY KEY (Cod_producto)
);

-- Tabla Usuarios --
CREATE TABLE Usuarios (
  Nombre varchar(15),
  Pass varchar(20),
  Telefono varchar(12),
  Mail varchar(45),
  Categoria char(2),
  CONSTRAINT pk_usuarios PRIMARY KEY (Nombre)
);

-- Tabla Pedidos --
CREATE TABLE Pedidos (
  Cod_pedido int,
  FechaPedido date,
  Estado varchar(15),
  Usuario varchar(15),
  CONSTRAINT pk_pedidos PRIMARY KEY (Cod_pedido),
  CONSTRAINT fk_pedidos_usuarios FOREIGN KEY (Usuario)
    REFERENCES Usuarios (Nombre) ON UPDATE cascade ON DELETE restrict
);
```

```
-- Tabla DetallePedidos --
CREATE TABLE DetallePedidos(
  Producto char(3),
  Pedido int,
  Cantidad int,
  CONSTRAINT pk_detallepedidos PRIMARY KEY (Producto, Pedido),
  CONSTRAINT fk_detallepedidos_productos FOREIGN KEY (Producto)
    REFERENCES Productos (Cod_producto) ON UPDATE cascade ON DELETE restrict,
  CONSTRAINT fk_detallepedidos_pedidos FOREIGN KEY (Pedido)
    REFERENCES Pedidos (Cod_pedido) ON UPDATE cascade ON DELETE restrict
);

-- Datos Productos --
INSERT INTO Productos VALUES('P01', 'Cafe solo', 1.10, false);
INSERT INTO Productos VALUES('P02', 'Cafe americano', 1.50, false);
INSERT INTO Productos VALUES('P03', 'Cafe c/leche', 1.30, true);
INSERT INTO Productos VALUES('P04', 'Cafe bombon', 2.00, true);
INSERT INTO Productos VALUES('P05', 'Infusion', 1.00, false);

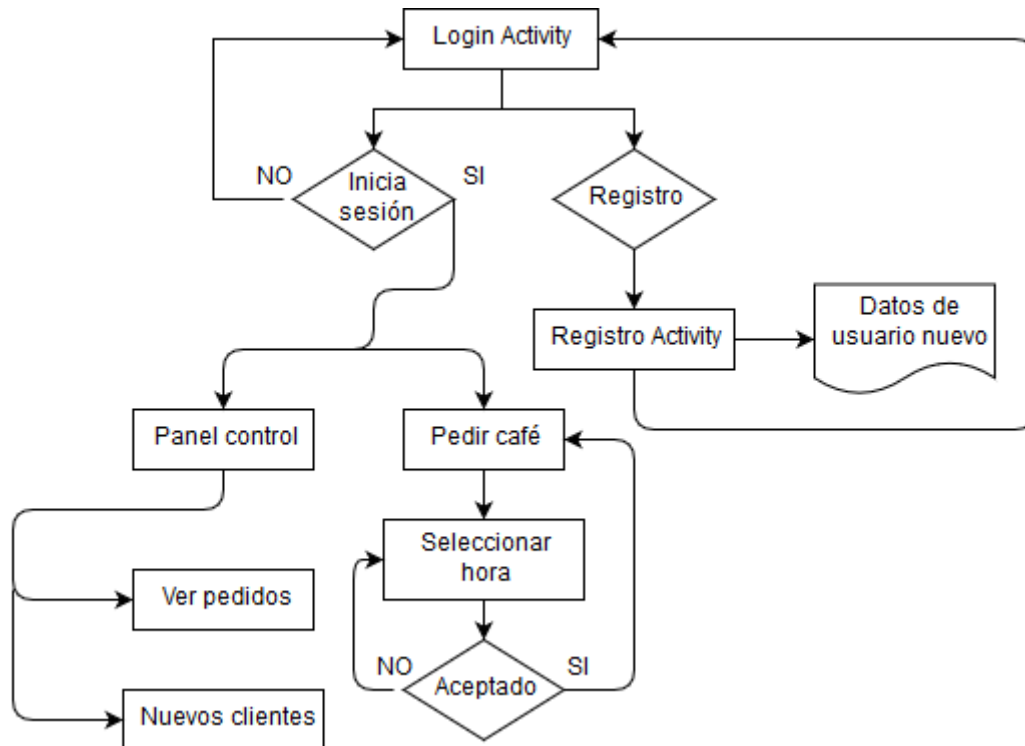
-- Datos Usuarios --
INSERT INTO Usuarios VALUES('admin', 'admin', '000000000000', 'admin@cafeteriakevindani.com', 'AD');
INSERT INTO Usuarios VALUES('usuario', 'usuario', '000000000000', 'usuario@gmail.com', 'US');

-- Datos Pedidos -- fecha en YYYY/MM/DD
INSERT INTO Pedidos VALUES(1, '2019/02/26', 'finalizado', 'usuario');
INSERT INTO Pedidos VALUES(2, '2019/02/27', 'finalizado', 'usuario');
INSERT INTO Pedidos VALUES(3, '2019/02/28', 'curso', 'usuario');

-- Datos DetallePedidos --
INSERT INTO DetallePedidos VALUES('P01', 1, 1);
INSERT INTO DetallePedidos VALUES('P03', 1, 2);
INSERT INTO DetallePedidos VALUES('P05', 2, 2);
INSERT INTO DetallePedidos VALUES('P03', 3, 1);
INSERT INTO DetallePedidos VALUES('P04', 3, 1);
```

## Esquema inicial de la aplicación

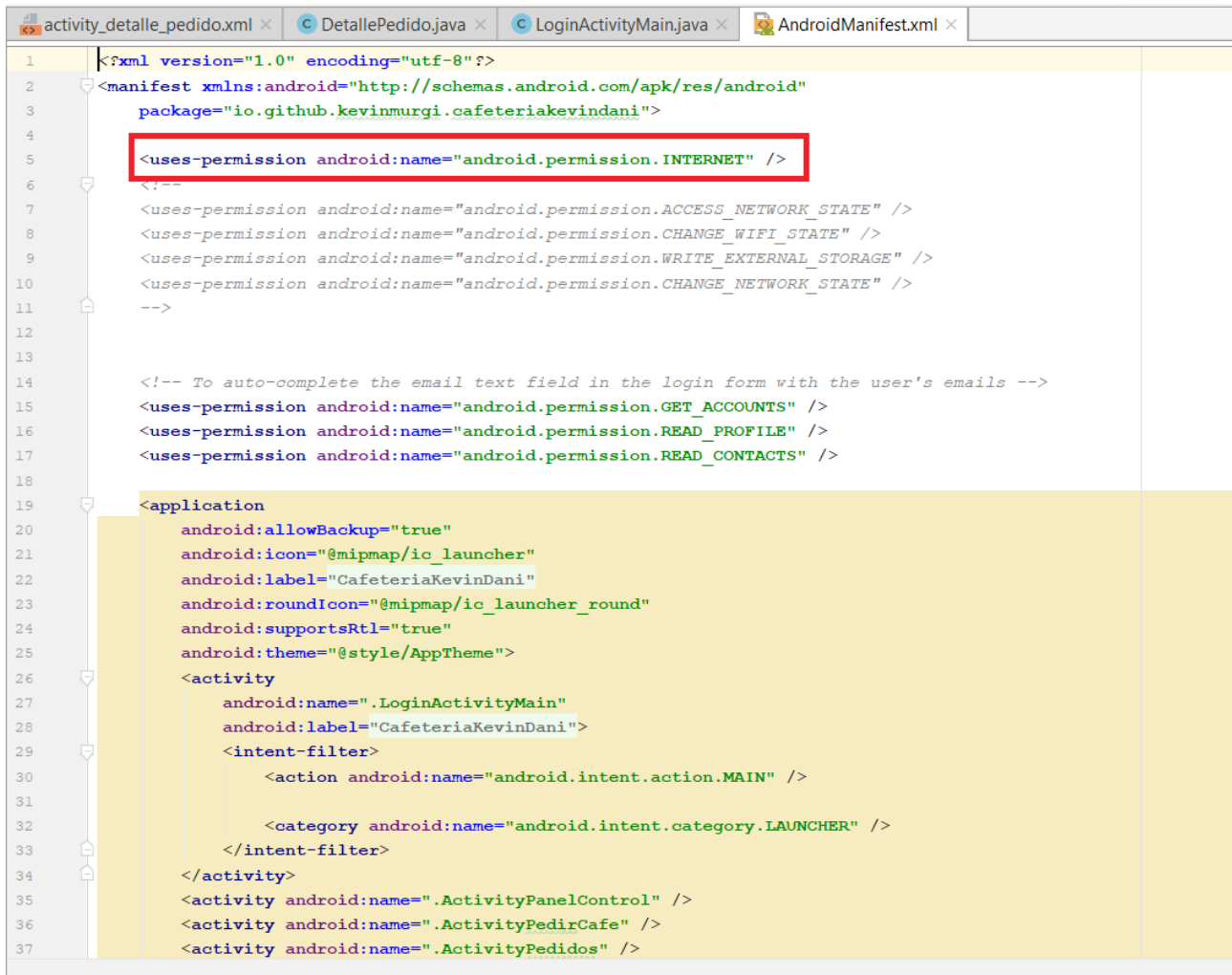
Previamente a empezar a programar la aplicación se ha realizado un esquema orientativo del funcionamiento que debería tener.



## Acceso a MySQL desde Android Studio

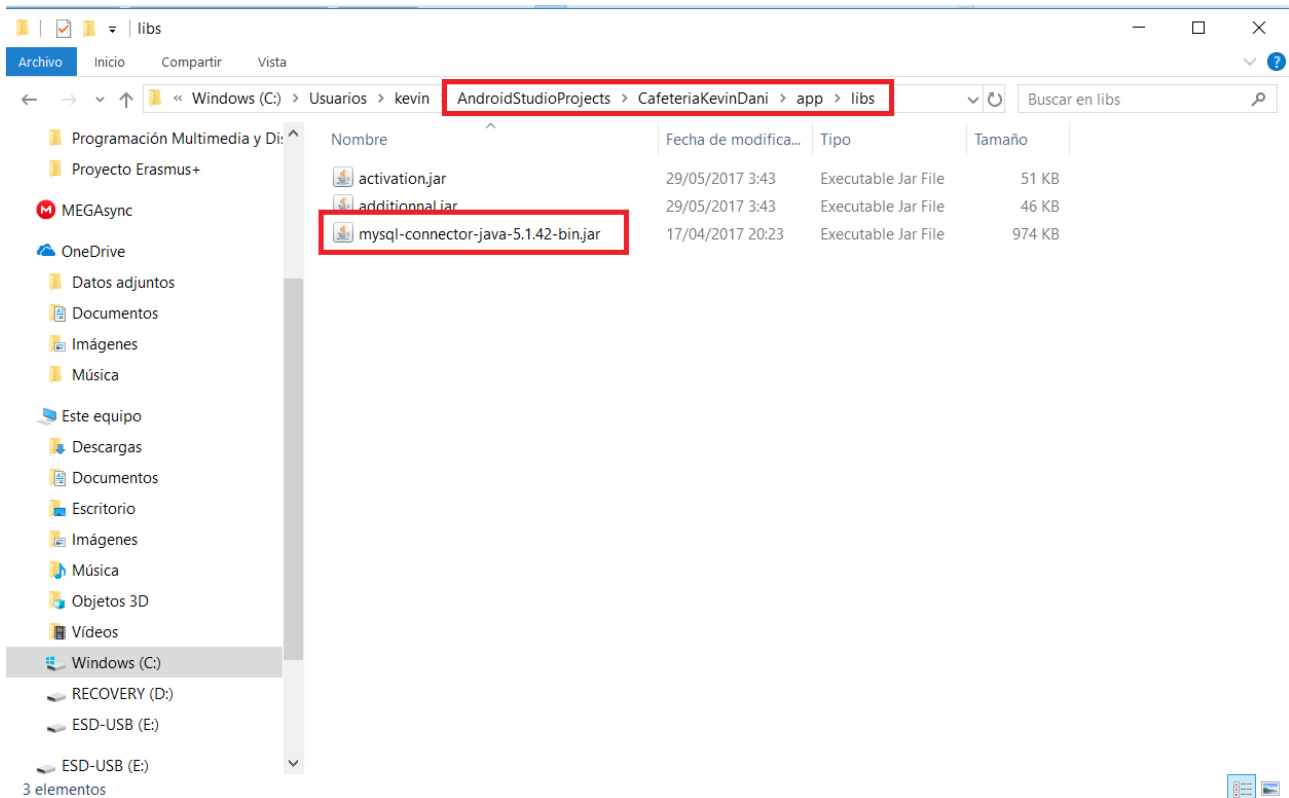
Para poder acceder al servidor MySQL desde una aplicación de Android Studio hay que tener en cuenta varias cosas:

- En el manifest de la aplicación hay que darle permisos para que pueda acceder a Internet.



```
1 <?xml version="1.0" encoding="utf-8"?>
2 <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3     package="io.github.kevinmurgi.cafeteriakevindani">
4
5     <uses-permission android:name="android.permission.INTERNET" />
6
7     <!--
8     <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
9     <uses-permission android:name="android.permission.CHANGE_WIFI_STATE" />
10    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
11    <uses-permission android:name="android.permission.CHANGE_NETWORK_STATE" />
12    -->
13
14    <!-- To auto-complete the email text field in the login form with the user's emails -->
15    <uses-permission android:name="android.permission.GET_ACCOUNTS" />
16    <uses-permission android:name="android.permission.READ_PROFILE" />
17    <uses-permission android:name="android.permission.READ_CONTACTS" />
18
19    <application
20        android:allowBackup="true"
21        android:icon="@mipmap/ic_launcher"
22        android:label="CafeteriaKevinDani"
23        android:roundIcon="@mipmap/ic_launcher_round"
24        android:supportRtl="true"
25        android:theme="@style/AppTheme">
26        <activity
27            android:name=".LoginActivityMain"
28            android:label="CafeteriaKevinDani">
29            <intent-filter>
30                <action android:name="android.intent.action.MAIN" />
31
32                <category android:name="android.intent.category.LAUNCHER" />
33            </intent-filter>
34        </activity>
35        <activity android:name=".ActivityPanelControl" />
36        <activity android:name=".ActivityPedirCafe" />
37        <activity android:name=".ActivityPedidos" />
38    </application>
39 </manifest>
```

- En el directorio Aplicación/app/libs hay que poner la librería necesaria para conectar a MySQL con Java.

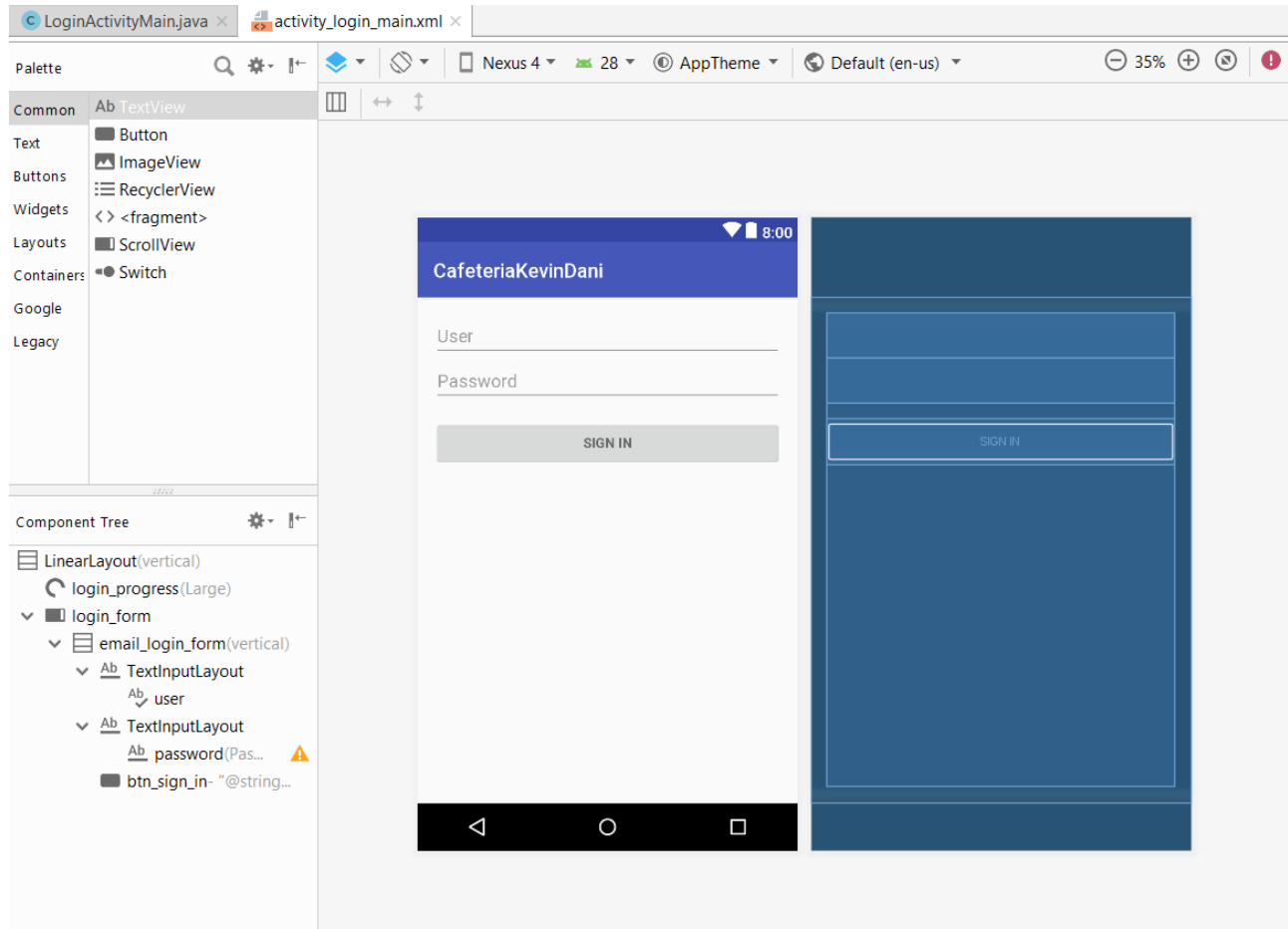


- La conexión a la base de datos debe realizarse en segundo plano, por ello en esta aplicación todos los accesos a la base de datos se realizan utilizando una clase que extiende (o hereda) de **AsyncTask**, de forma que se realiza la conexión en segundo plano y no afecta directamente al rendimiento de la aplicación. A lo largo del Paso a paso se pueden ver varios ejemplos de esto en las capturas de pantalla del código Java.

## LoginActivityMain

La primera actividad que muestra la aplicación es **LoginActivityMain** cuya funcionalidad es básicamente solicitar los datos de usuario para acceder a la aplicación ya sea como administrador o como un usuario corriente, como hemos visto en el diseño de la base de datos.

Diseño:





### Funcionamiento:

Cuando el usuario presiona el botón de **SIGN IN** (o Iniciar sesión en castellano), se lanza el evento `OnClick` de dicho botón. En este evento lo primero que se hace es comprobar que los campos de nombre de usuario y contraseña no estén vacíos, para evitar accesos innecesarios a la base de datos. En caso de que tengan contenido tanto el nombre como la contraseña ejecuta una nueva instancia de la clase **ComprobarUsuario**, que hereda de **AsyncTask**. Su funcionalidad es acceder a la base de datos y comprobar los datos de usuario recibidos: en caso de que el usuario no exista se informa al usuario, y en caso de que si exista, se comprueba su categoría (o bien AD o bien US) para mandarla a una actividad u otra dependiendo si se trata de un administrador de la cafetería o se un usuario corriente de la misma.

NOTA: La clase Java que representa esta actividad contiene como atributos estáticos los datos para conectar con la base de datos de MySQL para ser utilizados en toda la aplicación, de forma que en caso de haber un cambio de servidor o similar solo habría que cambiarlo aquí. De la misma manera es en esta clase donde se carga el Driver para poder trabajar con MySQL.

NOTA2: A lo largo de toda la aplicación el usuario va recibiendo información del estado la ejecución (Ejemplo: "Comprobando datos de usuario...") mediante **TOAST**, y todos los accesos a la base de datos se registran mediante **Logs**.

### Código JAVA:

```
LoginActivityMain.java x activity_login_main.xml x
20
21 public class LoginActivityMain extends AppCompatActivity {
22     // Atributos estaticos para la conexion a la BBDD
23     private static final String DRIVER_MYSQL = "com.mysql.jdbc.Driver";
24     protected static final String CONECTOR = "jdbc:mysql://192.168.1.72/Cafeteria?useUnicode=true&useJDBCCompliantTimezoneShift=true&use
25     protected static final String USUARIO = "UsuarioCafeteria";
26     protected static final String PASSWORD = "usuario";
27
28     // Atributos de clase
29     private AutoCompleteTextView tv_usuario;
30     private EditText et_pass;
31     private Button btn_signIn;
32     protected static String nom_usuario;
33     private String pass_usuario;
34
35     // Metodo On Create
36     @Override
37     protected void onCreate(Bundle savedInstanceState) {
38         super.onCreate(savedInstanceState);
39         setContentView(R.layout.activity_login_main);
40
41         // Llama a diversos metodos para cargar el driver de MySQL, inflar los componentes y establecer sus propiedades
42         this.cargarDriver();
43         this.inflar();
44         this.setOnClickSignIn();
45     }
46
47     /**
48      * Carga del driver para MySQL.
49      */
50     private void cargarDriver() {
51         try {
52             Class.forName(DRIVER_MYSQL);
53             Log.i( tag: "Carga Driver JDBC: ", msg: "Driver cargado de forma correcta.");
54         } catch (ClassNotFoundException e) {
55             Log.e( tag: "Carga Driver JDBC: ", msg: "No se ha podido cargar el driver JDBC: " + e.getMessage());
56         }
57     }
58 }
```

```

LoginActivityMain.java x activity_login_main.xml x
59  /**
60   * Metodo inflar() que infla los controles del Layout
61   */
62   private void inflar(){
63       this.tv_usuario = findViewById(R.id.user);
64       this.et_pass = findViewById(R.id.password);
65       this.btn_singin = findViewById(R.id.btn_sign_in);
66   }
67
68   /**
69   * Al iniciar sesion se comprueba que haya datos de inicio, en cuyo caso se realiza el acceso a la BBDD.
70   */
71   private void setOnClickSignIn(){
72       this.btn_singin.setOnClickListener((v) -> {
73           // 1. Recoger nombre y pass
74           nom_usuario = tv_usuario.getText().toString();
75           pass_usuario = et_pass.getText().toString();
76
77           // 2. Iniciar AsyncTask
78           if(!nom_usuario.equals("") && !pass_usuario.equals("")){
79               Toast.makeText( context: LoginActivityMain.this, text: "Comprobando datos...", Toast.LENGTH_SHORT).show();
80               new ComprobarUsuario().execute();
81           }
82       });
83   }
84
85   /**
86   * Dependiendo del tipo de usuario que ha logueado se le manda a una actividad u otra.
87   */
88   protected void navegar(boolean isLoggedIn, String categoria){
89       if(isLoggedIn){
90           Intent intent = null;
91           if(categoria.toUpperCase().startsWith("AD")){
92               intent = new Intent( packageContext: LoginActivityMain.this, ActivityPanelControl.class);
93               startActivity(intent);
94               Toast.makeText( context: LoginActivityMain.this, text: "Bienvenido, administrador", Toast.LENGTH_SHORT).show();
95           } else if(categoria.toUpperCase().startsWith("US")){
96               intent = new Intent( packageContext: LoginActivityMain.this, ActivityPedirCafe.class);
97           }
98       }
99   }

```

```

100  /**
101   * Clase ComprobarUsuario que extiende de AsyncTask
102   * Accede a la BBDD y comprueba que el usuario que intenta
103   * iniciar sesion existe, en cuyo caso se llama al metodo
104   * navegar() de la clase y le manda a la siguiente actividad.
105   */
106  protected class ComprobarUsuario extends AsyncTask<Void,Void,ResultSet>{
107      // Atributos
108      private Boolean isLoggedIn = false;
109      private String categoria = "";
110      private Connection conexion;
111      private Statement sentencia;
112      private ResultSet resultado;
113
114      // Do in background
115      @Override
116      protected ResultSet doInBackground(Void... voids) {
117          try {
118              // Conectamos con la BBDD
119              conexion = DriverManager.getConnection(LoginActivityMain.CONECTOR, LoginActivityMain.USUARIO, LoginActivityMain.PASSWORD);
120              Log.i( tag: "BBDD: ", msg: "Se ha conectado de forma satisfactoria con la base de datos.");
121
122              // Creamos una sentencia y realizamos la consulta pertinente
123              sentencia = conexion.createStatement();
124              resultado = sentencia.executeQuery( sql: "select * from Usuarios where Nombre = '" + nom_usuario + "' and Pass = '" + pass_usuario + "'");
125
126          } catch (SQLException sqlE) {
127              Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
128                  "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
129          }
130
131          return resultado;
132      }
133  }

```

```
// On post execute
@Override
protected void onPostExecute(ResultSet resultSet) {
    super.onPostExecute(resultSet);

    try{
        // Recorremos el resultado de la consulta y guardamos la categoria del usuario
        while(resultSet.next()){
            this.categoria = resultSet.getString( columnLabel: "Categoria");
            isLoggedIn = true;
        }

        // Cerramos la conexion con la BBDD
        resultSet.close();
        sentencia.close();
        conexion.close();

        // Mandamos al usuario a la siguiente actividad
        navegar(isLoggedIn, categoria);
    } catch (SQLException sqlE) {
        Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
            "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
    }
}
```

Como se puede ver en el método **navegar(boolean, String)**, en caso de que el usuario que ha iniciado sesión sea un administrador, se le manda a la actividad **ActivityPanelControl**, y en caso de que sea un usuario corriente de la cafetería, se le manda a la actividad **ActivityPedirCafe**. Vamos a explicar/analizar primero la parte de un usuario corriente y, posteriormente, la de un administrador.

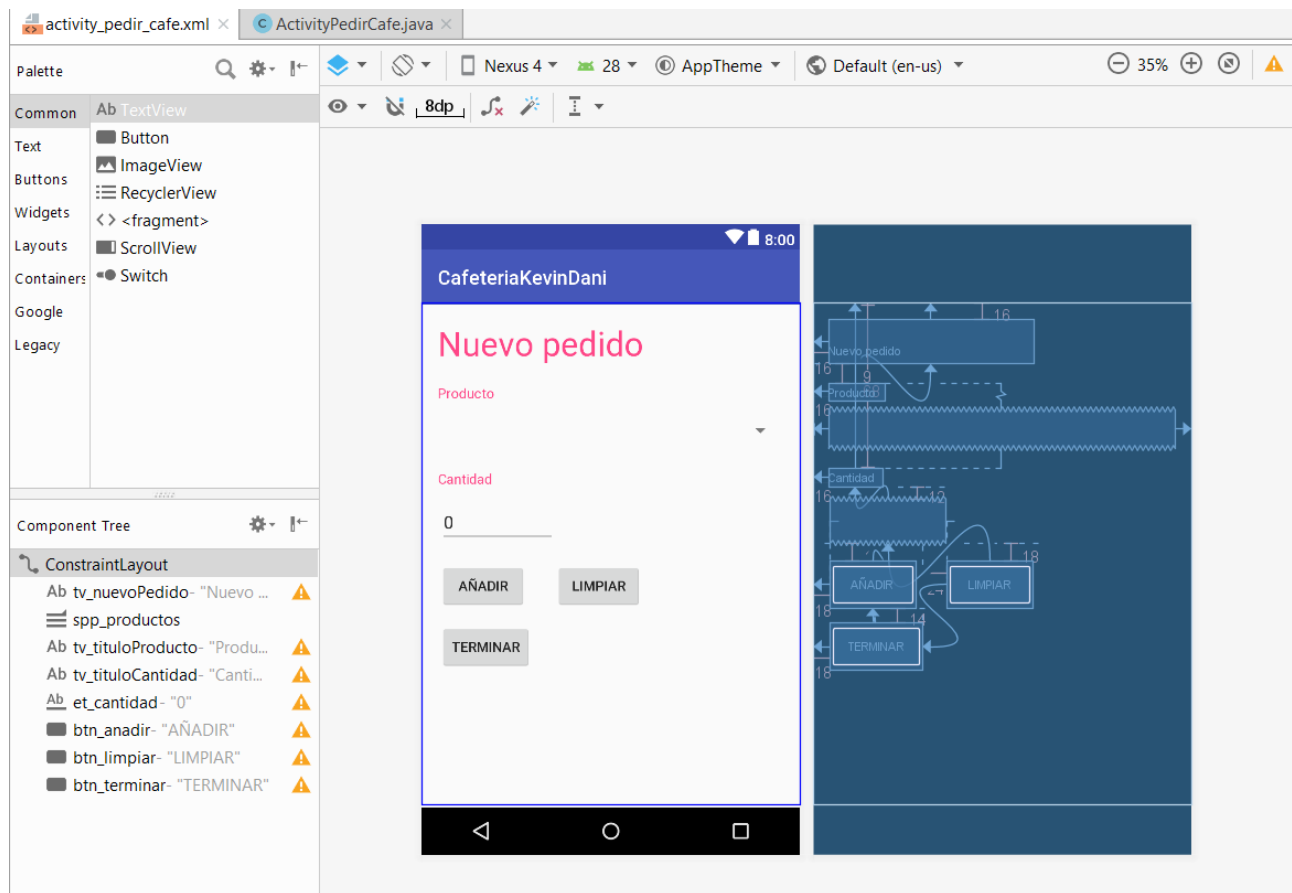
## ActivityPedirCafe

En esta actividad en donde los usuarios seleccionan los productos que quieren que compongan su pedido así como la cantidad del mismo.

Para ello hay un Spinner con los productos que pueden elegir y un campo numérico donde establecen la cantidad solicitada.

Si se presiona el botón **AÑADIR**, se añade ese producto al pedido; si se presiona el botón **LIMPIAR**, se ponen los datos por defecto; y si se presiona el botón **TERMINAR**, se accede a la página donde ver los datos de todo el pedido.

Diseño:



Funcionamiento:

Mediante la clase Java **RecogerNomProductos** se accede a la base de datos para recoger el nombre de todos los productos que ofrece la cafetería, y los carga en un **Spinner** haciendo uso de un **ArrayAdapter**.

Una vez está preparada la actividad, el usuario selecciona el producto y la cantidad del mismo deseada. Al presionar el botón **Añadir**, la aplicación primero comprueba que la cantidad solicitada no sea 0, para evitar accesos innecesarios a la base de datos, en caso contrario, haciendo uso de la clase **BuscarProducto**, accede de nuevo a la base de datos, busca todos los datos de ese producto (Precio unitario, código...) y lo añade a un **ArrayList** que contiene los datos del nuevo pedido. Si por el contrario se presiona el botón **Limpiar**, lo que se hace es poner todos los valores por defecto (Spinner a la posición 0 y cantidad a 0). Por último, si se presiona el botón **Terminar** se manda al usuario a la actividad **ActivityNuevoPedido** donde se muestran todos los datos del nuevo pedido.

Código:

```
public class ActivityPedirCafe extends AppCompatActivity {
    // Atributos estaticos
    protected static ArrayList<Producto> productosPedido = new ArrayList<>();

    // Atributos de clase
    private Spinner spp_productos;
    private EditText et_cantidad;
    private Button btn_anadir;
    private Button btn_limpiar;
    private Button btn_terminar;
    private ArrayList<String> nomProductos = new ArrayList<>();

    @Override
    public void onBackPressed() {
        Intent data = new Intent( packageContext: ActivityPedirCafe.this, LoginActivityMain.class);
        startActivity(data);
    }

    /**
     * Metodo onCreate
     * @param savedInstanceState
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pedir_cafe);

        // Lo de siempre
        inflar();
        setButtonListeners();
        new RecogerNomProductos().execute();
    }
}
```

```

/**
 * Metodo inflar() que infla los controles
 */
private void inflar(){
    spp_productos = findViewById(R.id.spp_productos);
    et_cantidad = findViewById(R.id.et_cantidad);
    btn_anadir = findViewById(R.id.btn_anadir);
    btn_limpiar = findViewById(R.id.btn_limpiar);
    btn_terminar = findViewById(R.id.btn_terminar);
}

/**
 * Metodo setButtonListeners() que establece el comportamiento
 * de los botones al ser seleccionados
 */
private void setButtonListeners(){
    btn_anadir.setOnClickListener((v) -> {
        if(!String.valueOf(et_cantidad.getText()).trim().equals("0") && !String.valueOf(et_cantidad.getText()).trim().equals("")){
            new BuscarProducto().execute();
        } else {
            Toast.makeText(context: ActivityPedirCafe.this, text: "La cantidad no puede ser 0", Toast.LENGTH_SHORT).show();
        }
    });

    btn_limpiar.setOnClickListener((v) -> { limpiarCampos(); });

    btn_terminar.setOnClickListener((v) -> {
        Intent data = new Intent(packageContext: ActivityPedirCafe.this, ActivityNuevoPedido.class);
        startActivity(data);
    });
}

/**
 * Metodo limpiarCampos()
 * pone todos los valores del producto por defecto
 */
private void limpiarCampos(){
    spp_productos.setSelection(0);
    et_cantidad.setText("0");
}

/**
 * Clase RecogerNomProductos
 * conecta con la BBDD y almacena el nombre de todos los productos,
 * posteriormente los muestra a través del spinner
 */
protected class RecogerNomProductos extends AsyncTask<Void, Void, ResultSet>{
    // Atributos de clase
    Connection conexion;
    Statement sentencia;
    ResultSet resultado;

    // doInBackground
    @Override
    protected ResultSet doInBackground(Void... voids) {
        try {
            // Conectamos con la BBDD
            conexion = DriverManager.getConnection(LoginActivityMain.CONECTOR, LoginActivityMain.USUARIO, LoginActivityMain.PASSWORD);
            Log.i(tag: "BBDD: ", msg: "Se ha conectado de forma satisfactoria con la base de datos.");

            // Creamos una nueva sentencia y realizamos la consulta pertinente
            sentencia = conexion.createStatement();
            resultado = sentencia.executeQuery(sql: "select Nombre from Productos");

        } catch (SQLException sqlE) {
            Log.e(tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
                "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
        }

        return resultado;
    }
}

```

```
// onPostExecute
@Override
protected void onPostExecute(ResultSet resultSet) {
    super.onPostExecute(resultSet);

    try{
        // Recorremos el resultado de la consulta y almacenamos los nombres en un ArrayList
        while(resultSet.next()){
            nomProductos.add(resultSet.getString( columnLabel: "Nombre"));
        }

        // Configuramos el adapter y se lo asociamos al Spinner
        ArrayAdapter<String> adapter = new ArrayAdapter<>( context: ActivityPedirCafe.this, android.R.layout.simple_spinner_item, nomProductos);
        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        spp_productos.setAdapter(adapter);

        // Cerramos la conexion
        resultSet.close();
        sentencia.close();
        conexion.close();

    } catch (SQLException sqlE) {
        Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
            "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
    }
}

/**
 * Clase BuscarProducto
 * conecta con la BBDD y busca el producto que ha sido añadido al pedido,
 * posteriormente lo almacena en un array, que compone el pedido
 */
protected class BuscarProducto extends AsyncTask<Void, Void, ResultSet>{
    // Atributos de clase
    Connection conexion;
    Statement sentencia;
    ResultSet resultado;

    // doInBackground
    @Override
    protected ResultSet doInBackground(Void... voids) {
        try {
            // Conectamos con la BBDD
            conexion = DriverManager.getConnection(LoginActivityMain.CONECTOR, LoginActivityMain.USUARIO, LoginActivityMain.PASSWORD);
            Log.i( tag: "BBDD: ", msg: "Se ha conectado de forma satisfactoria con la base de datos.");

            // Creamos una nueva sentencia y realizamos la consulta pertinente
            sentencia = conexion.createStatement();
            resultado = sentencia.executeQuery( sql: "select * from Productos where Nombre = '" + (String)spp_productos.getSelectedItem() + "'");

        } catch (SQLException sqlE) {
            Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
                "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
        }

        return resultado;
    }
}

// onPostExecute
@Override
protected void onPostExecute(ResultSet resultSet) {
    super.onPostExecute(resultSet);

    try{
        // Recorremos el resultado de la consulta y almacenamos los nombres en un ArrayList
        while(resultSet.next()){
            productosPedido.add(new Producto(resultSet.getString( columnLabel: "Cod_producto"), resultSet.getString( columnLabel: "Nombre"),
                resultSet.getFloat( columnLabel: "Precio"), Integer.parseInt(String.valueOf(et_cantidad.getText())), resultSet.getInt( columnLabel:
            ));

        }

        // Cerramos la conexion
        resultSet.close();
        sentencia.close();
        conexion.close();

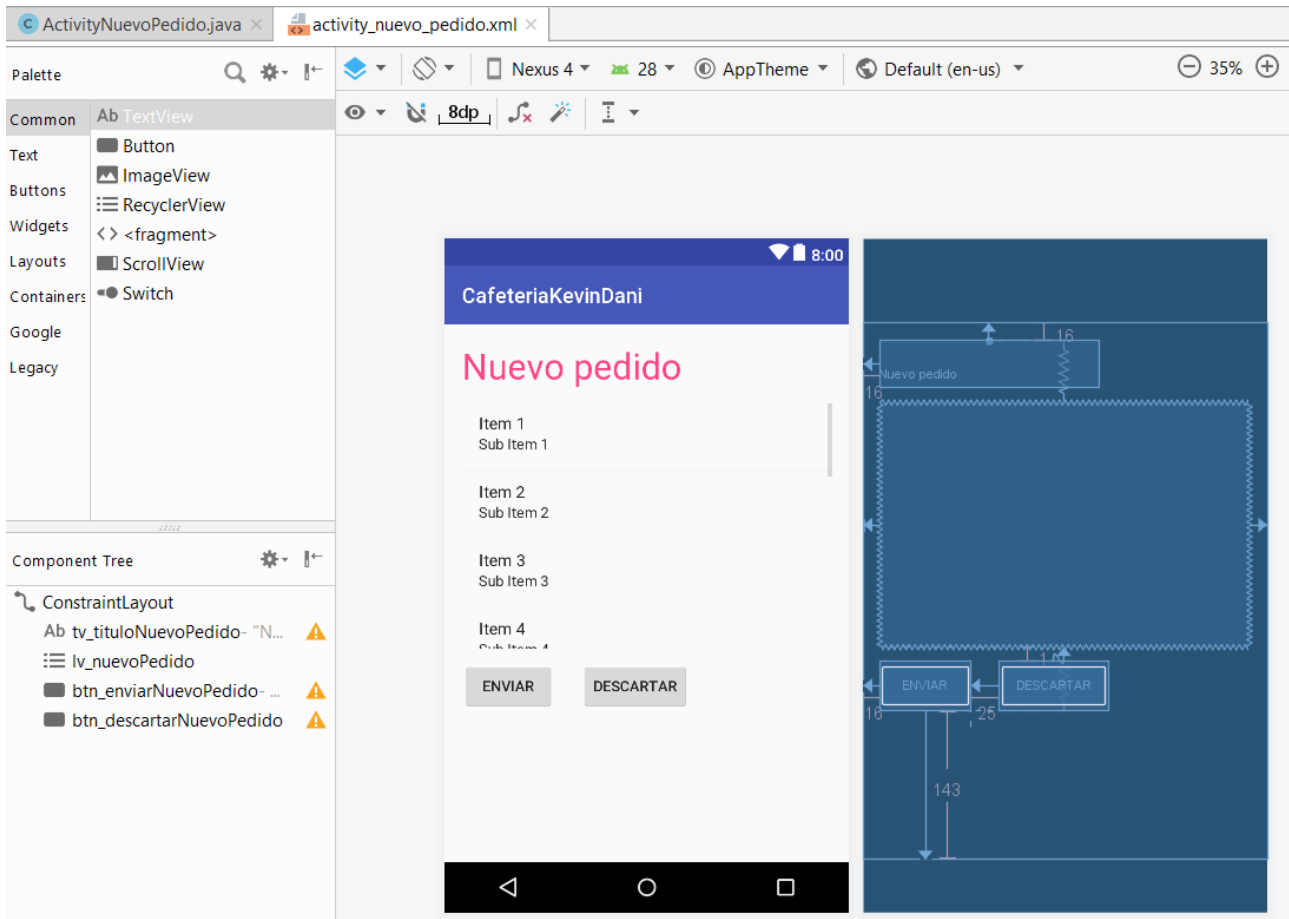
        limpiarCampos();

    } catch (SQLException sqlE) {
        Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
            "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
    }
}
}
```

## ActivityNuevoPedido

En esta actividad el usuario puede ver en un **ListView** todos los productos que componen su pedido y puede decidir entre enviar el pedido o cancelarlo.

Diseño:





Funcionamiento:

Haciendo uso de un la clase **AdapterProductos**, y el **ArrayList** con todos los productos elegidos por el usuario, se muestran en un **ListView**. Si el usuario presiona el botón **Enviar**, se accede a la base de datos para insertar todos los registros correspondientes para que se lleve a cabo el pedido, y si por el contrario presiona el botón **Descartar**, lo que hace es vaciar el pedido.

AdapterProductos:

```
public class AdapterProductos extends BaseAdapter {
    // Atributos
    ArrayList<Producto> listaProductos = new ArrayList<>();

    @Override
    public int getCount() { return listaProductos.size(); }

    @Override
    public Object getItem(int position) { return listaProductos.get(position); }

    @Override
    public long getItemId(int position) { return position; }

    @Override
    public View getView(int position, View vista, ViewGroup parent) {
        if(vista == null){
            Context cnt = parent.getContext();
            vista = LayoutInflater.from(cnt).inflate(R.layout.esqueleto_producto, root: null);
        }

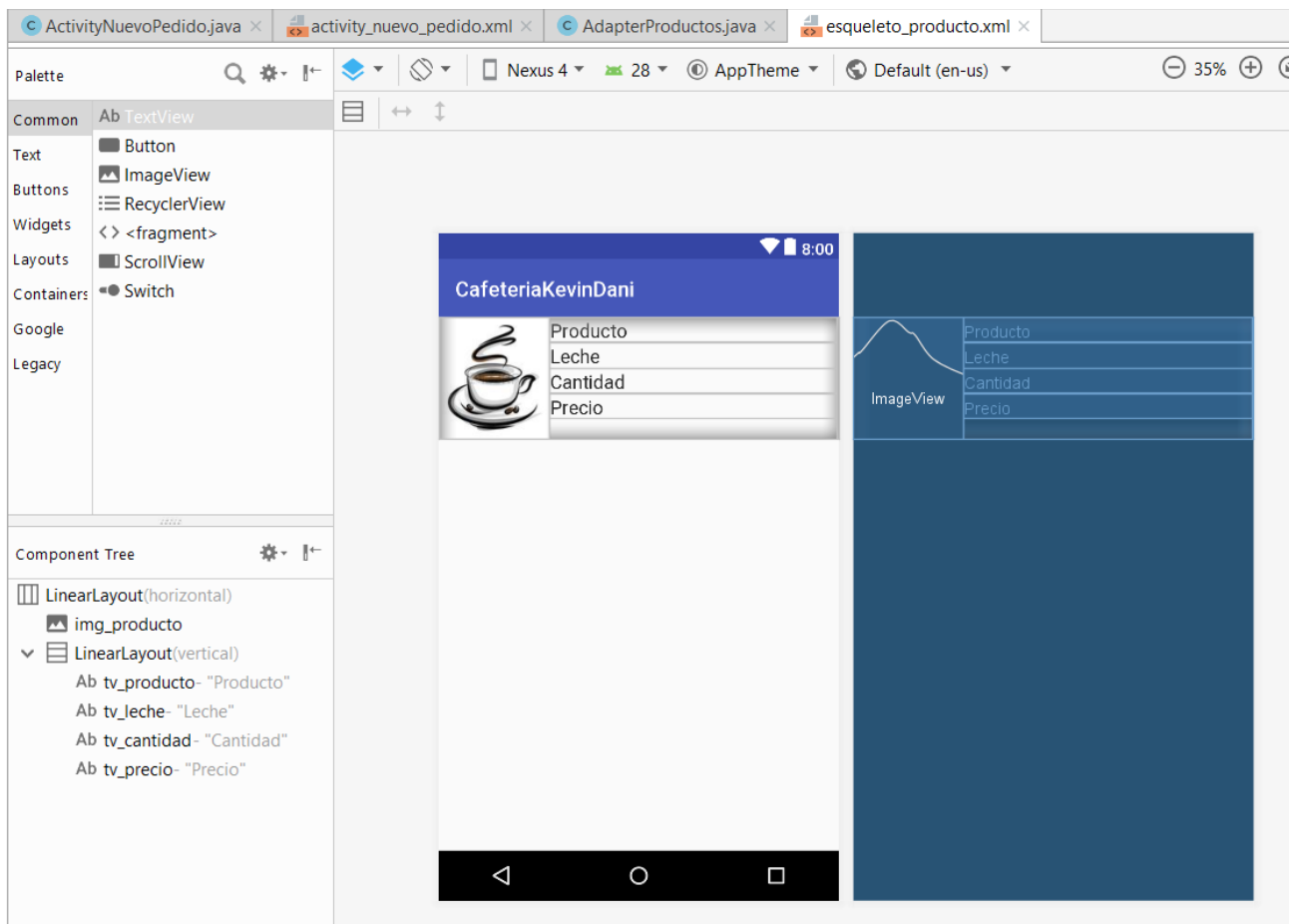
        Producto productoActual = listaProductos.get(position);

        //ImageView imgProducto = vista.findViewById(R.id.img_pedido);
        TextView producto = vista.findViewById(R.id.tv_producto);
        TextView leche = vista.findViewById(R.id.tv_leche);
        TextView cantidad = vista.findViewById(R.id.tv_cantidad);
        TextView precio = vista.findViewById(R.id.tv_precio);

        producto.setText(productoActual.getNombre());
        leche.setText("Leche? " + (productoActual.isLeche() == true ? "Si" : "No"));
        cantidad.setText("Cantidad: " + String.valueOf(productoActual.getCantidad()));
        precio.setText("Precio: " + String.valueOf(productoActual.getPrecio() + "€"));

        return vista;
    }
}
```

La funcionalidad de un **BaseAdapter** ya ha sido explicado en múltiples ocasiones a lo largo del curso. Para este se utiliza el esqueleto **esqueleto\_producto**:



## Código de NuevoPedido:

```
public class ActivityNuevoPedido extends AppCompatActivity {
    // Atributos
    private ListView listNuevoPedido;
    private Button btnEnviar;
    private Button btnDescartar;
    private AdapterProductos adapter = new AdapterProductos();

    @Override
    public void onBackPressed() {
        Intent data = new Intent( packageContext: ActivityNuevoPedido.this, ActivityPedirCafe.class);
        startActivity(data);
    }

    /**
     * Metodo onCreate
     * @param savedInstanceState
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_nuevo_pedido);

        // Lo de siempre
        inflar();
        configurarListView();
        setButtonListeners();
    }

    /**
     * Metodo inflar() que infla los controles de la parte grafica
     */
    private void inflar(){
        listNuevoPedido = findViewById(R.id.lv_nuevoPedido);
        btnEnviar = findViewById(R.id.btn_enviarNuevoPedido);
        btnDescartar = findViewById(R.id.btn_descartarNuevoPedido);
    }

    /**
     * Metodo configurarListView() que configura el adapter del ListView
     */
    private void configurarListView(){
        adapter.listaProductos = ActivityPedirCafe.productosPedido;
        listNuevoPedido.setAdapter(adapter);
    }

    /**
     * Metodo serButtonLusteners()
     */
    private void setButtonListeners(){
        btnEnviar.setOnClickListener((v) -> {
            new InsertarPedido().execute();
            Toast.makeText( context: ActivityNuevoPedido.this, text: "Realizando pedido...", Toast.LENGTH_SHORT).show();
        });

        btnDescartar.setOnClickListener((v) -> {
            finalizarPedido();
            Toast.makeText( context: ActivityNuevoPedido.this, text: "Pedido eliminado", Toast.LENGTH_SHORT).show();
        });
    }

    private void finalizarPedido(){
        ActivityPedirCafe.productosPedido.clear();
        Intent data = new Intent( packageContext: ActivityNuevoPedido.this, ActivityPedirCafe.class);
        startActivity(data);
    }
}
```

```

*/
protected class InsertarPedido extends AsyncTask<Void, Void, ResultSet>{
    // Atributos
    private Connection conexion;
    private Statement sentencia;
    private ResultSet resultado;
    private PreparedStatement sentenciaPreparada1;
    private PreparedStatement sentenciaPreparada2;

    @Override
    protected ResultSet doInBackground(Void... voids) {
        try {
            // Conectamos con la BBDD
            conexion = DriverManager.getConnection(LoginActivityMain.CONECTOR, LoginActivityMain.USUARIO, LoginActivityMain.PASSWORD);
            Log.i( tag: "BBDD:", msg: "Se ha conectado de forma satisfactoria con la base de datos.");

            sentencia = conexion.createStatement();
            resultado = sentencia.executeQuery( sql: "select max(Cod_pedido) as Maximo from Pedidos");
            resultado.next();
            int ultimoPedido = resultado.getInt( columnLabel: "Maximo");
            ultimoPedido++;

            // Creamos una nueva sentencia y ejecutamos la consulta pertinente
            String sql = "insert into Pedidos values(?, ?, ?, ?)";
            sentenciaPreparada1 = conexion.prepareStatement(sql);
            sentenciaPreparada1.setInt( parameterIndex: 1, ultimoPedido);
            sentenciaPreparada1.setString( parameterIndex: 2, x: "2019-02-28");
            sentenciaPreparada1.setString( parameterIndex: 3, x: "curso");
            sentenciaPreparada1.setString( parameterIndex: 4, LoginActivityMain.nom_usuario);
            sentenciaPreparada1.executeUpdate();

            // Realizamos una insercion por cada producto del pedido
            String sql2 = "insert into DetallePedidos values(?, ?, ?)";
            sentenciaPreparada2 = conexion.prepareStatement(sql2);
            for(Producto p : ActivityPedirCafe.productosPedido){
                sentenciaPreparada2.setString( parameterIndex: 1, p.getCodigo());
                sentenciaPreparada2.setInt( parameterIndex: 2, ultimoPedido);
                sentenciaPreparada2.setInt( parameterIndex: 3, p.getCantidad());
            }

            sentenciaPreparada2.setString( parameterIndex: 1, p.getCodigo());
            sentenciaPreparada2.setInt( parameterIndex: 2, ultimoPedido);
            sentenciaPreparada2.setInt( parameterIndex: 3, p.getCantidad());
            sentenciaPreparada2.executeUpdate();

        } catch (SQLException sqlE) {
            Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
                "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
        }

        return null;
    }

    // On post execute
    @Override
    protected void onPostExecute(ResultSet resultSet) {
        super.onPostExecute(resultSet);

        try{
            // Cerramos la conexion
            sentencia.close();
            sentenciaPreparada1.close();
            sentenciaPreparada2.close();
            conexion.close();

            finalizarPedido();
            Toast.makeText( context: ActivityNuevoPedido.this, text: "Pedido registrado correctamente", Toast.LENGTH_SHORT).show();

        } catch (SQLException sqlE) {
            Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
                "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
        }
    }
}
}

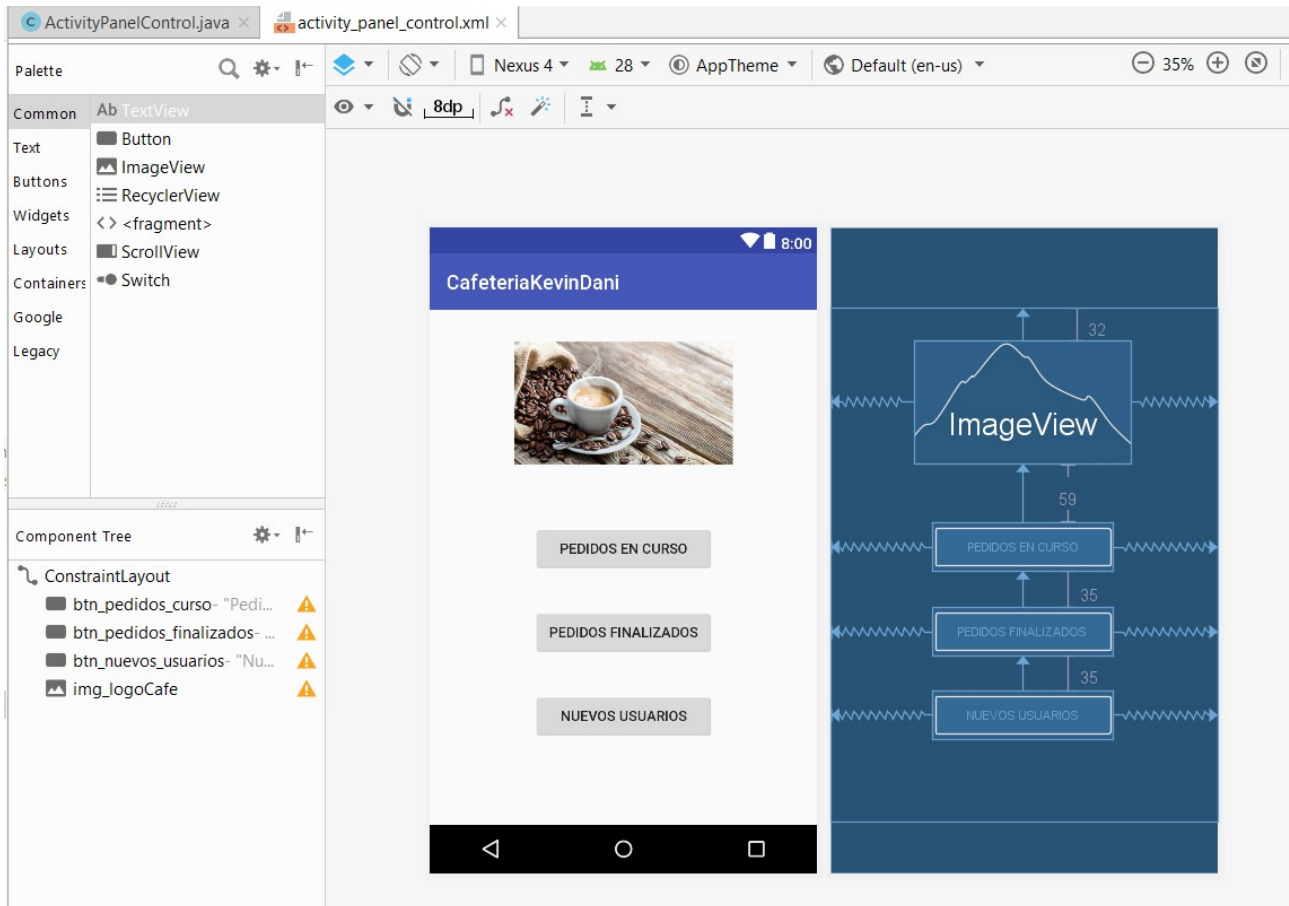
```

Con esto la parte del usuario de la cafetería quedaría completa, ahora vamos a ver la parte de un administrador.

## ActivityPanelControl

En esta actividad de gestionan todas las opciones que tiene un administrador, que por ahora son: Ver pedidos que están en curso, ver pedidos ya terminados y ver nuevos usuarios que quieran registrarse en la aplicación (esta última aún no está implementada).

Diseño:



### Funcionamiento:

El funcionamiento de esta actividad es muy sencillo, simplemente cada uno de los botones, con un evento OnClick, manda al usuario a una actividad u otra.

Tanto los pedidos en curso como los pedidos ya finalizados utilizan la misma actividad, pero con la instancia de la clase **Intent** se le pasa un String diciendo si tiene que mostrar unos pedidos u otros, y al acceder a la base de datos solamente rescata unos registros y otros.

### Código:

```
public class ActivityPanelControl extends AppCompatActivity {
    // Atributos de clase
    private Button btn_pedidosEnCurso;
    private Button btn_pedidosFinalizados;
    private Button btn_nuevosUsuarios;

    @Override
    public void onBackPressed() {
        Intent data = new Intent( packageContext: ActivityPanelControl.this, LoginActivityMain.class);
        startActivity(data);
    }

    /**
     * Metodo onCreate
     * @param savedInstanceState
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_panel_control);

        // Inflamos los controles y les asignamos su funcionalidad
        inflar();
        setButtonListeners();
    }

    /**
     * Metodo que realiza el inflado de los controles del layout
     */
    private void inflar(){
        btn_pedidosEnCurso = findViewById(R.id.btn_pedidos_curso);
        btn_pedidosFinalizados = findViewById(R.id.btn_pedidos_finalizados);
        btn_nuevosUsuarios = findViewById(R.id.btn_nuevos_usuarios);
    }
}
```

```

} /**
 * Metodo que establece los listeners de los botones
 */
} private void setButtonListeners() {
}   btn_pedidosEnCurso.setOnClickListener((v) -> {
    Intent data = new Intent( packageContext: ActivityPanelControl.this, ActivityPedidos.class);
    data.putExtra( name: "TipoPedidos", value: "curso");
    startActivity(data);
    Toast.makeText( context: ActivityPanelControl.this, text: "Cargando pedidos en curso...", Toast.LENGTH_SHORT).show();
  });

  btn_pedidosFinalizados.setOnClickListener((v) -> {
    Intent data = new Intent( packageContext: ActivityPanelControl.this, ActivityPedidos.class);
    data.putExtra( name: "TipoPedidos", value: "finalizado");
    startActivity(data);
    Toast.makeText( context: ActivityPanelControl.this, text: "Cargando pedidos finalizados...", Toast.LENGTH_SHORT).show();
  });

  btn_nuevosUsuarios.setOnClickListener((v) -> {
    Intent data = new Intent( packageContext: ActivityPanelControl.this, ActivityUsuarios.class);
    startActivity(data);
    Toast.makeText( context: ActivityPanelControl.this, text: "Cargando solicitudes nuevas...", Toast.LENGTH_SHORT).show();
  });
}
}

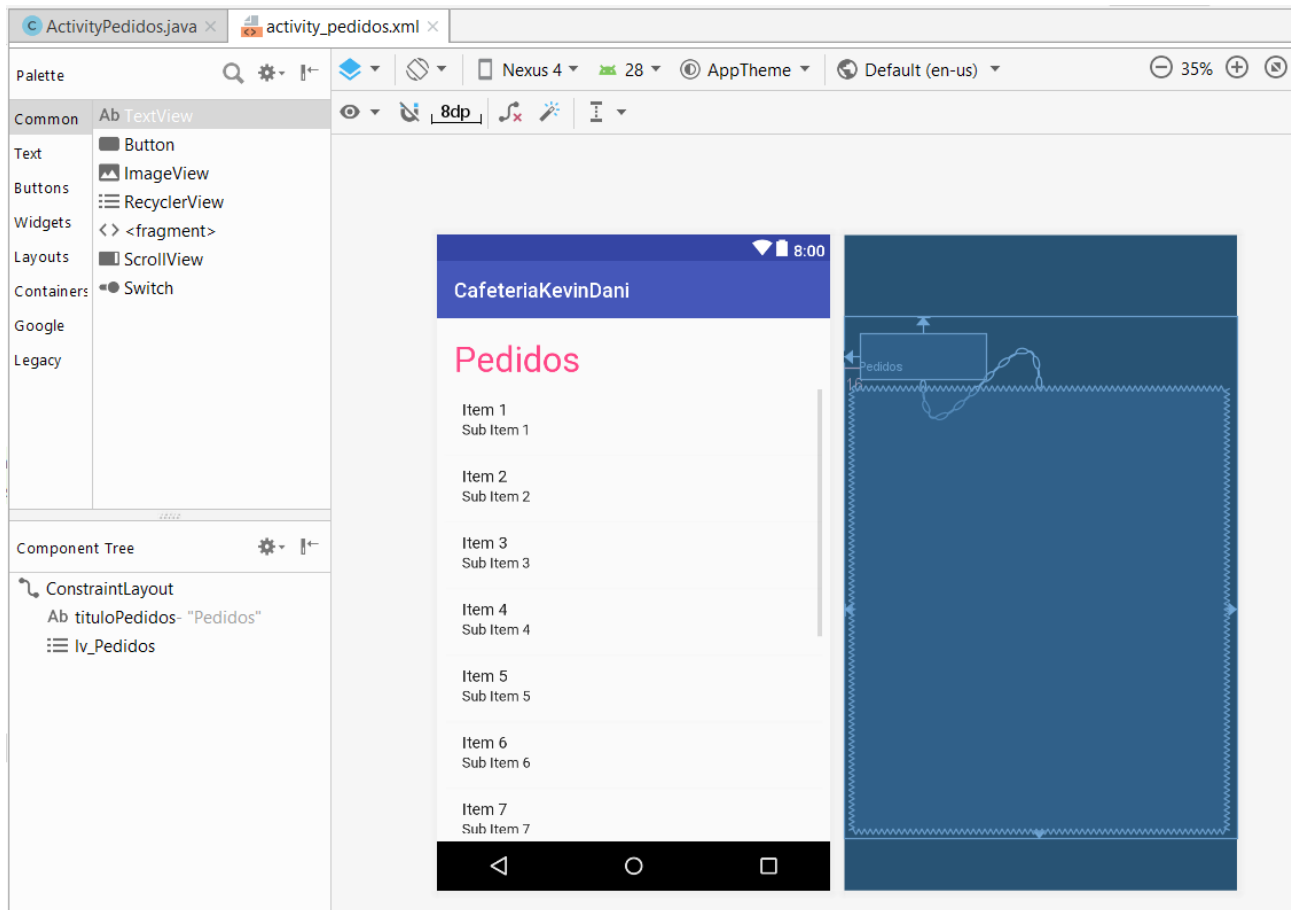
```

La actividad que muestra los pedidos es **ActivityPedidos**, y la que muestra los usuarios es **ActivityUsuarios**. Como ya he mencionado anteriormente, la de los usuarios aún no está implementada por lo que está vacía.

## ActivityPedidos

Esta actividad recibe a través de un objeto **Intent** un String que le indica si los pedidos que debe mostrar son los que están en curso o los que ya están terminados. Teniendo esto en cuenta esto, accede a la base de datos y muestra los datos de los pedidos usando un **ListView**. Al seleccionar alguno de los pedidos nos lleva a otra actividad donde se muestran los detalles del pedido (productos, precio total...).

Diseño:



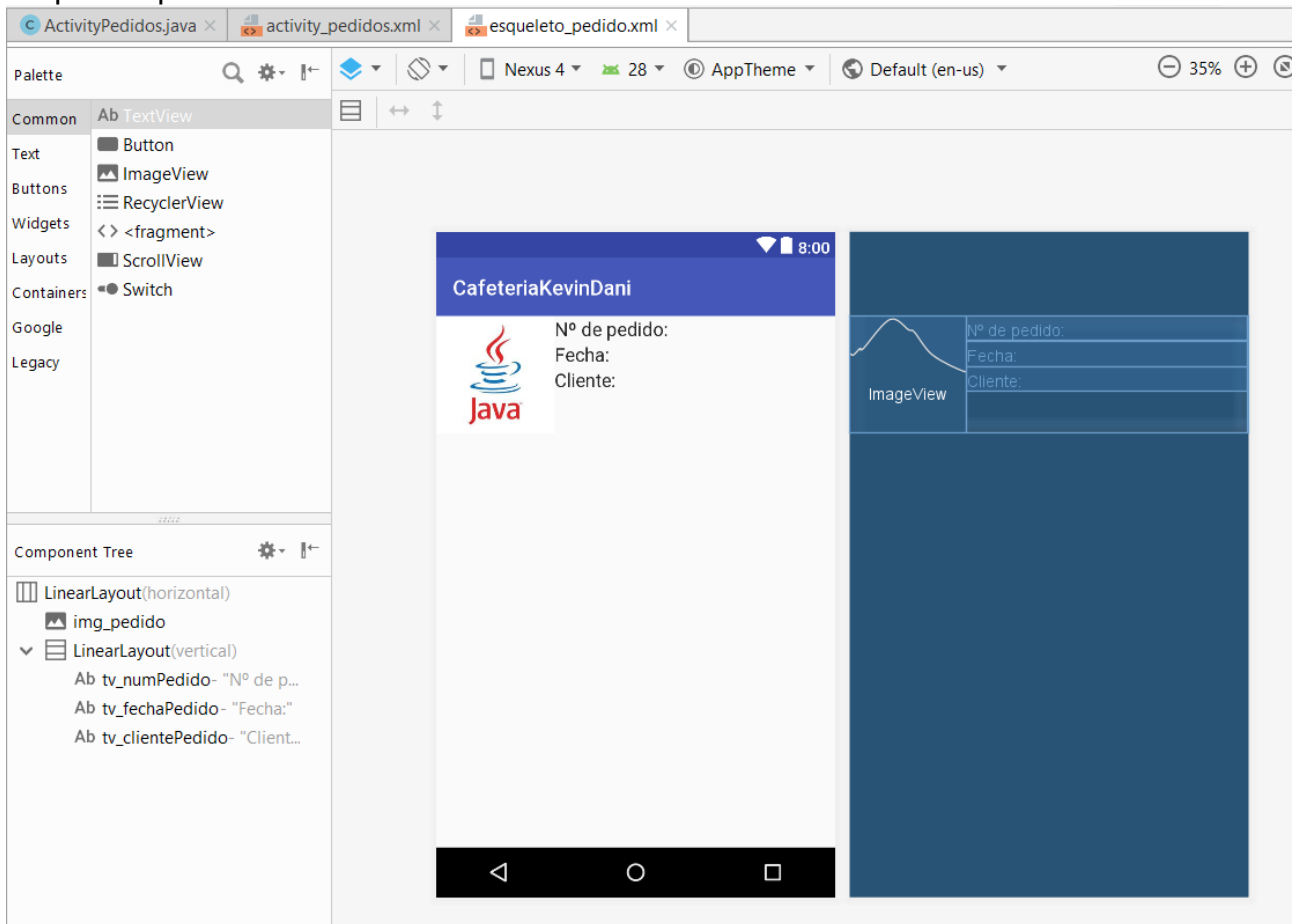
Como ya sabemos, para el **ListView** se ha hecho una clase que extiende de **BaseAdapter**, que en este caso se llama **AdapterPedidos**, y un esqueleto XML para organizar los datos, que en este caso es **esqueleto\_pedido**.



## AdapterPedidos:

```
public class AdapterPedidos extends BaseAdapter {  
    // Atributos  
    ArrayList<Pedido> listaPedidos = new ArrayList<>();  
  
    @Override  
    public int getCount() { return listaPedidos.size(); }  
  
    @Override  
    public Object getItem(int position) { return listaPedidos.get(position); }  
  
    @Override  
    public long getItemId(int position) { return position; }  
  
    @Override  
    public View getView(int position, View vista, ViewGroup parent) {  
        if(vista == null){  
            Context cnt = parent.getContext();  
            vista = LayoutInflater.from(cnt).inflate(R.layout.esqueleto_pedido, root: null);  
        }  
  
        Pedido pedidoActual = listaPedidos.get(position);  
  
        //ImageView imgPedido = vista.findViewById(R.id.img_pedido);  
        TextView tv_num = vista.findViewById(R.id.tv_numPedido);  
        TextView tv_fecha = vista.findViewById(R.id.tv_fechaPedido);  
        TextView tv_cliente = vista.findViewById(R.id.tv_clientePedido);  
  
        tv_num.setText(tv_num.getText() + " " + pedidoActual.getCodigo());  
        tv_fecha.setText(tv_fecha.getText() + " " + pedidoActual.getFecha().toString());  
        tv_cliente.setText(tv_cliente.getText() + " " + pedidoActual.getUsuario());  
  
        return vista;  
    }  
}
```

## Esqueleto pedido:



## Código de ActivityPedidos:

```
public class ActivityPedidos extends AppCompatActivity {
    // Atributos de clase
    private TextView titulo;
    private ListView listaPedidos;
    private String tipoPedidos;
    private AdapterPedidos adapter = new AdapterPedidos();

    @Override
    public void onBackPressed() {
        Intent data = new Intent( packageContext: ActivityPedidos.this, ActivityPanelControl.class);
        startActivity(data);
    }

    /**
     * onCreate method
     * @param savedInstanceState
     */
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pedidos);

        // Recogemos los datos enviados con el intent para saber que tipo de pedidos mostrar
        Intent data = getIntent();
        tipoPedidos = (String) data.getStringExtra( name: "TipoPedidos");

        // Inflamos los controles, establecemos sus propiedades y conectamos con la BBDD para mostrar los pedidos
        inflar();
        setTitulo();
        setListenerListView();

        new RecogerPedidos().execute();
    }

    /**
     * Metodo inflar() que infla los controles del Layout
     */
    private void inflar(){
        titulo = findViewById(R.id.tituloPedidos);
        listaPedidos = findViewById(R.id.lv_Pedidos);
    }

    /**
     * Metodo setTitulo() que establece el titulo de la actividad en funcion del tipo de pedidos que muestra
     */
    private void setTitulo(){
        if(tipoPedidos.equals("curso")){
            titulo.setText(titulo.getText() + " en curso");
        } else if(tipoPedidos.equals("finalizado")){
            titulo.setText(titulo.getText() + " finalizados");
        }
    }

    /**
     * Metodo que establece el listener para el ListView que muestra los pedidos
     * Manda al usuario a la actividad DetallePedido con el numero del pedido seleccionado
     */
    private void setListenerListView() {
        listaPedidos.setOnItemClickListener(new AdapterView.OnItemClickListener() {
            @Override
            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                Pedido pedido = (Pedido) parent.getItemAtPosition(position);
                Intent data = new Intent( packageContext: ActivityPedidos.this, DetallePedido.class);
                data.putExtra( name: "Pedido", String.valueOf(pedido.getCodigo()));
                data.putExtra( name: "TipoPedidos", tipoPedidos);
                startActivity(data);
                Toast.makeText( context: ActivityPedidos.this, text: "Cargando detalles del pedido " + pedido.getCodigo() + "...", Toast.LENGTH_SHORT).show();
            }
        });
    }
}
```

```
/**
 * Clase RecogerPedidos extiende de AsyncTask
 * Conecta con la BBDD y recoge todos los pedidos cuyo estado sea el recibido a través del intent
 */
protected class RecogerPedidos extends AsyncTask<Void, Void, ResultSet>{
    // Atributos
    private Connection conexion;
    private Statement sentencia;
    private ResultSet resultado;

    @Override
    protected ResultSet doInBackground(Void... voids) {
        try {
            // Conectamos con la BBDD
            conexion = DriverManager.getConnection(LoginActivityMain.CONECTOR, LoginActivityMain.USUARIO, LoginActivityMain.PASSWORD);
            Log.i( tag: "BBDD: ", msg: "Se ha conectado de forma satisfactoria con la base de datos.");

            // Creamos una nueva sentencia y ejecutamos la consulta pertinente
            sentencia = conexion.createStatement();
            if(tipoPedidos.equals("curso")){
                resultado = sentencia.executeQuery( sql: "select * from Pedidos where Estado = 'curso'");
            } else if(tipoPedidos.equals("finalizado")){
                resultado = sentencia.executeQuery( sql: "select * from Pedidos where Estado = 'finalizado'");
            }

        } catch (SQLException sqlE) {
            Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
                "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
        }

        return resultado;
    }

    // On post execute
    @Override
    protected void onPostExecute(ResultSet resultSet) {
        super.onPostExecute(resultSet);

        try{
            // Recorremos todos los datos y los almacenamos en el adapter para mostrarlos en la actividad
            while(resultSet.next()){
                Pedido nuevoPedido = new Pedido(resultSet.getInt( columnName: "Cod_pedido"), resultSet.getDate( columnName: "FechaPedido"),
                    resultSet.getString( columnName: "Estado"), resultSet.getString( columnName: "Usuario"));
                adapter.listaPedidos.add(nuevoPedido);
            }

            // Establecemos el nuevo adapter al listview
            listaPedidos.setAdapter(adapter);

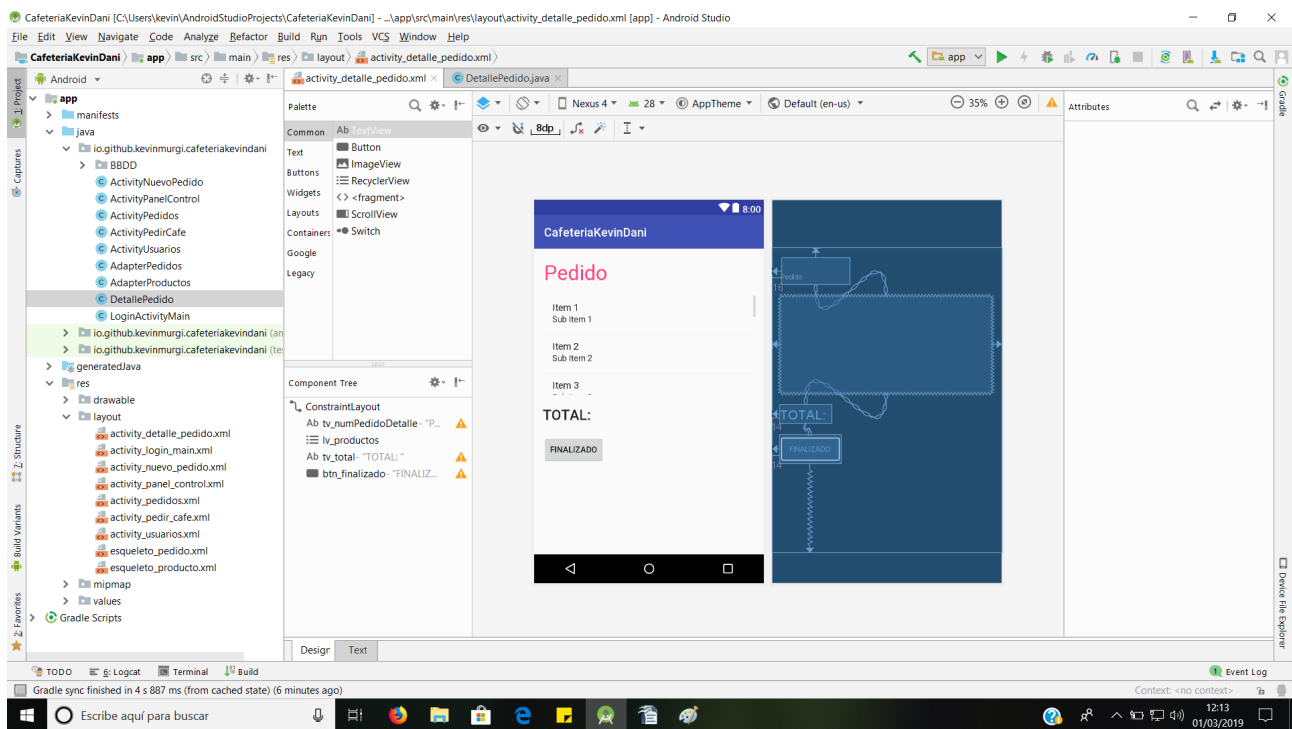
            // Cerramos la conexion
            resultSet.close();
            sentencia.close();
            conexion.close();

        } catch (SQLException sqlE) {
            Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
                "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
        }
    }
}
```

## DetallePedido

Como ya se ha mencionado anteriormente, al seleccionar alguno de los pedidos la aplicación va a una nueva actividad llamada **DetallePedido** donde se muestran más detalles del pedido y, en caso de que sea un pedido en curso, se puede dar por finalizado, de forma que pasaría a ser un pedido finalizado.

Diseño:



Para mostrar todos los productos que componen el pedido se utiliza un **ListView**. Utiliza el mismo adapter y el mismo esqueleto que en la actividad **ActivityNuevoPedido**, es decir **AdapterProductos** y **esqueleto\_producto** (Su código y diseño de pueden ver más arriba en su correspondiente apartado).

### Funcionamiento:

Esta actividad recibe con el objeto **Intent** el código del pedido, y haciendo uso de la clase **RecogerProductos**, recoge de la base de datos todos los datos de los productos que componen el pedido y luego se calcula el importe total del pedido.

En caso de que el pedido esté en curso, se habilita el botón de **Finalizar**, que al presionarlo se accede a la base de datos mediante la clase **FinalizarPedido** y actualiza el pedido con el que se está tratando para que pase a ser un pedido finalizado y deje de estar en curso.

### Código:

```
public class DetallePedido extends AppCompatActivity {  
    // Atributos de clase  
    private TextView numeroPedido;  
    private ListView listaProductos;  
    private TextView totalPedido;  
    private Button finalizarPedido;  
    private String pedido;  
    private String tipoPedido;  
    private AdapterProductos adapter = new AdapterProductos();  
  
    @Override  
    public void onBackPressed() {  
        Intent data = new Intent( packageContext: DetallePedido.this, ActivityPedidos.class);  
        data.putExtra( name: "TipoPedidos", tipoPedido);  
        startActivity(data);  
    }  
  
    /**  
     * Método onCreate  
     * @param savedInstanceState  
     */  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_detalle_pedido);  
  
        // Guardamos los datos recibidos a través del intent  
        Intent data = getIntent();  
        pedido = (String) data.getStringExtra( name: "Pedido");  
        tipoPedido = (String) data.getStringExtra( name: "TipoPedidos");  
  
        // Inflamos los datos, establecemos sus propiedades y conectamos con la BBDD para mostrar los detalles del pedido solicitado  
        inflar();  
        establecerValores();  
        setButtonListener();  
  
        new RecogerProductos().execute();  
    }  
}
```

```
7  /**
   * Metodo onRestart() que actualiza los datos mostrados
   */
   @Override
7  protected void onRestart() {
       super.onRestart();

       new RecogerProductos().execute();
   }

7  /**
   * Metodo inflar() que infla los controles de la parte grafica
   */
7  private void inflar(){
       numeroPedido = findViewById(R.id.tv_numPedidoDetalle);
       listaProductos = findViewById(R.id.lv_productos);
       totalPedido = findViewById(R.id.tv_total);
       finalizarPedido = findViewById(R.id.btn_finalizado);
   }

7  /**
   * Metodo establecerValores() que establece el titulo de la pagina
   * y el estado del boton de finalizar en funcion del estado del pedido
   * que se esta mostrando.
   */
7  private void establecerValores(){
       numeroPedido.setText("Pedido " + pedido);
       if(tipoPedido.equals("finalizado")){
           finalizarPedido.setEnabled(false);
           finalizarPedido.setClickable(false);
       }
   }

7  /**
   * Metodo setButtonListener() que establece el listener para el boton finalizar, que realiza
   * una conexion con la BBDD y cambia el estado del pedido a finalizado
   */
7  private void setButtonListener(){
       finalizarPedido.setOnClickListener((v) -> {
           new FinalizarPedido().execute();
           Toast.makeText(context, DetallePedido.this, text: "Actualizando estado...", Toast.LENGTH_SHORT).show();
       });
   }

7  /**
   * Metodo calcularTotal() que recorre todos los productos del pedido y calcula el
   * precio total del pedido
   */
7  private void calcularTotal(){
       double total = 0;
       for(Producto p : adapter.listaProductos){
           total += p.getCantidad() * p.getPrecio();
       }
       DecimalFormat df = new DecimalFormat(pattern: "###.##");
       String formatted = df.format(total);

       totalPedido.setText("TOTAL: " + formatted + "€");
   }
}
```

```
1  /**
2   * Clase RecogerProductos
3   * Conecta con la BBDD y recoge los datos de todos los productos que componen
4   * el pedido solicitado
5   */
6  protected class RecogerProductos extends AsyncTask<Void, Void, ResultSet>{
7      // Atributos de clase
8      private Connection conexion;
9      private Statement sentencia;
10     private ResultSet resultado;
11
12     @Override
13     protected ResultSet doInBackground(Void... voids) {
14         try {
15             // Conectamos con la BBDD
16             conexion = DriverManager.getConnection(LoginActivityMain.CONECTOR, LoginActivityMain.USUARIO, LoginActivityMain.PASSWORD);
17             Log.i( tag: "BBDD: ", msg: "Se ha conectado de forma satisfactoria con la base de datos.");
18
19             // Creamos una nueva sentencia y realizamos la consulta pertinente
20             sentencia = conexion.createStatement();
21             resultado = sentencia.executeQuery( sql: "select Cod_producto, Nombre, Precio, Cantidad, Leche from Productos, DetallePedidos where Cod_p
22         } catch (SQLException sqlE) {
23             Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
24                 "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
25         }
26
27         return resultado;
28     }
29
30     // On post execute
31     @Override
32     protected void onPostExecute(ResultSet resultSet) {
33         super.onPostExecute(resultSet);
34
35         try{
36             // Recorremos los datos recibidos y los almacenamos en el arraylist del adaptador,
37             // de la misma forma que en la vez anterior.
38             while(resultSet.next()){
39                 Producto nuevoProducto = new Producto(resultSet.getString( columnLabel: "Cod_producto"), resultSet.getString( columnLabel: "Nombre"),
40                     resultSet.getFloat( columnLabel: "Precio"), resultSet.getInt( columnLabel: "Cantidad"), resultSet.getInt( columnLabel: "Leche"));
41                 adapter.listaProductos.add(nuevoProducto);
42             }
43             listaProductos.setAdapter(adapter);
44
45             // Cerramos la conexion
46             resultSet.close();
47             sentencia.close();
48             conexion.close();
49
50             calcularTotal();
51         } catch (SQLException sqlE) {
52             Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
53                 "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
54         }
55     }
56 }
```



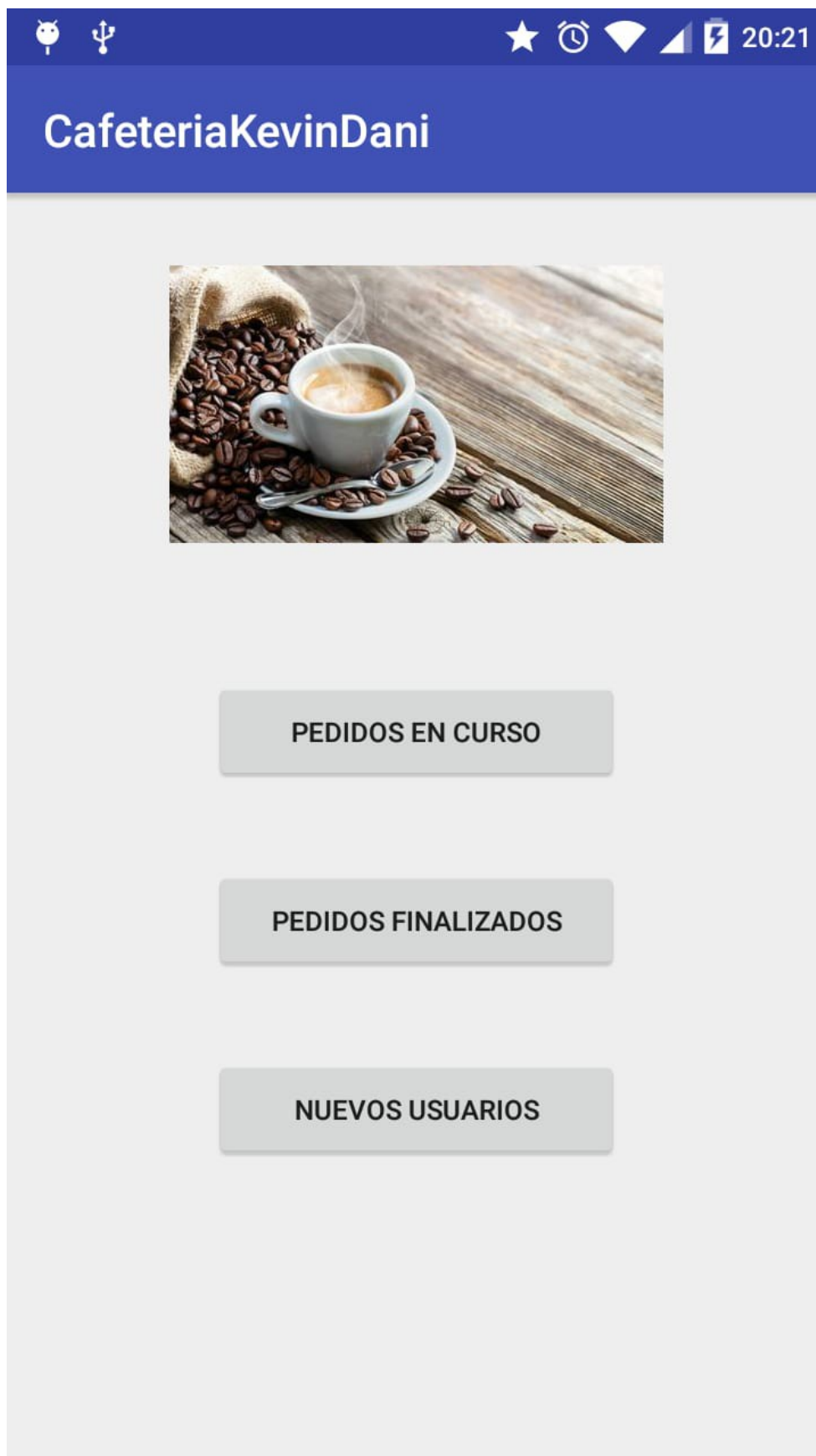
```
/**
 * Clase FinalizarPedido
 * conecta con la BBDD y actualiza los datos del pedido cambiando su estado a finalizado
 */
// On post execute
@Override
protected void onPostExecute(ResultSet resultSet) {
    super.onPostExecute(resultSet);

    try{
        // Cerramos la conexion
        sentencia.close();
        conexion.close();

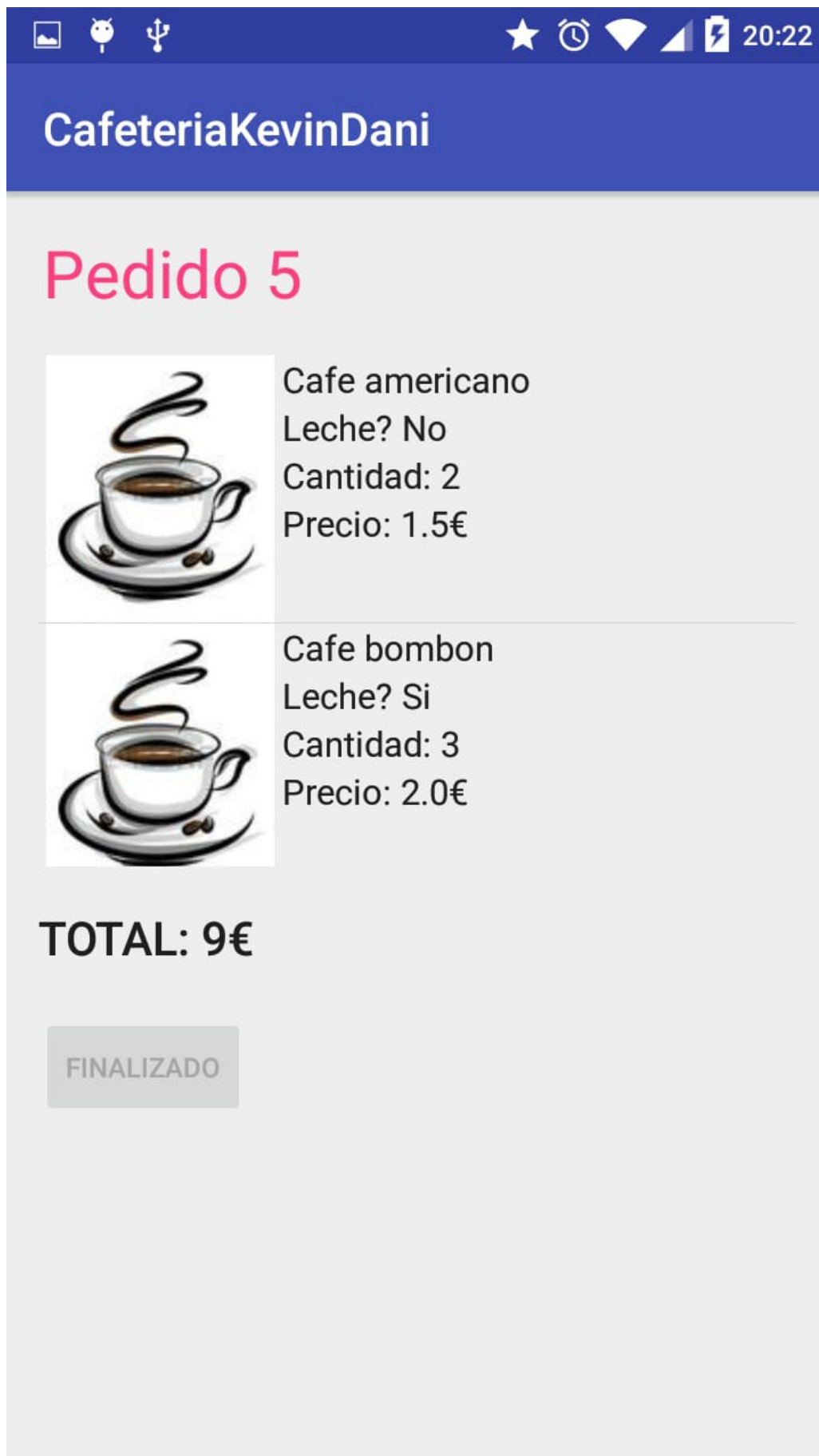
        // Deshabilitamos el boton para finalizar
        finalizarPedido.setEnabled(false);
        finalizarPedido.setClickable(false);

        // Mandamos al usuario de vuelta al panel de control
        Intent data = new Intent( packageContext: DetallePedido.this, ActivityPanelControl.class);
        startActivity(data);
        Toast.makeText( context: DetallePedido.this, text: "Pedido finalizado con éxito", Toast.LENGTH_SHORT).show();
    } catch (SQLException sqlE) {
        Log.e( tag: "\nProblemas con la BBDD:", msg: "Mensaje:\t"+sqlE.getMessage() +
            "\nEstado SQL:\t"+sqlE.getSQLState() + "\nCodigo Error:\t"+sqlE.getErrorCode());
    }
}
}
```

## Ejemplos de ejecución



	
<b>CafeteriaKevinDani</b>	
<h2>Pedidos finalizados</h2>	
	Nº de pedido: 1 Fecha: 2019-02-28 Cliente: usuario
	Nº de pedido: 2 Fecha: 2019-02-28 Cliente: usuario
	Nº de pedido: 4 Fecha: 2019-02-28 Cliente: usuario
	Nº de pedido: 5 Fecha: 2019-02-28 Cliente: usuario
	Nº de pedido: 6 Fecha: 2019-02-28



The image shows a mobile application interface for a cafeteria. At the top, there is a blue header bar with the text "CafeteriaKevinDani". Below the header, the main content area has a light gray background. The title "Nuevo pedido" is displayed in a large, bold, pink font. Underneath, there are two main input sections. The first section is labeled "Producto" in pink, with a text input field containing "Cafe c/leche" and a small downward arrow on the right. The second section is labeled "Cantidad" in pink, with a text input field containing the number "3". Below these input fields, there are three gray buttons with black text: "AÑADIR", "LIMPIAR", and "TERMINAR". The top of the screen shows a standard Android status bar with various icons and the time "20:22".