|  |
| --- |
| http://www.becas.sep.gob.mx/images/logo.png  TECNOLÓGICO NACIONAL DE MÉXICO  Instituto Tecnológico de Chihuahua II |
| Practica 3 unidad 3 |
| Una base de datos del ife. |
|  |
|  |
| 08/05/2016  Unidad de la Materia: III  Nombre del Alumno: Jose Luis Castro Lozoya  No. Control: 12550479  Materia: programación de dispositivos móviles plataforma 2 |

# Introducción

Confeccionar un programa que permita almacenar los datos de los votantes de

una elección. Crear la tabla votantes y definir los campos IFE(documento de identidad),

nombre del votante, dirección donde vota y numero de casilla donde vota.

El programa debe permitir:

1. Creación de usuarios

2. Consulta por el IFE(para saber donde vota).

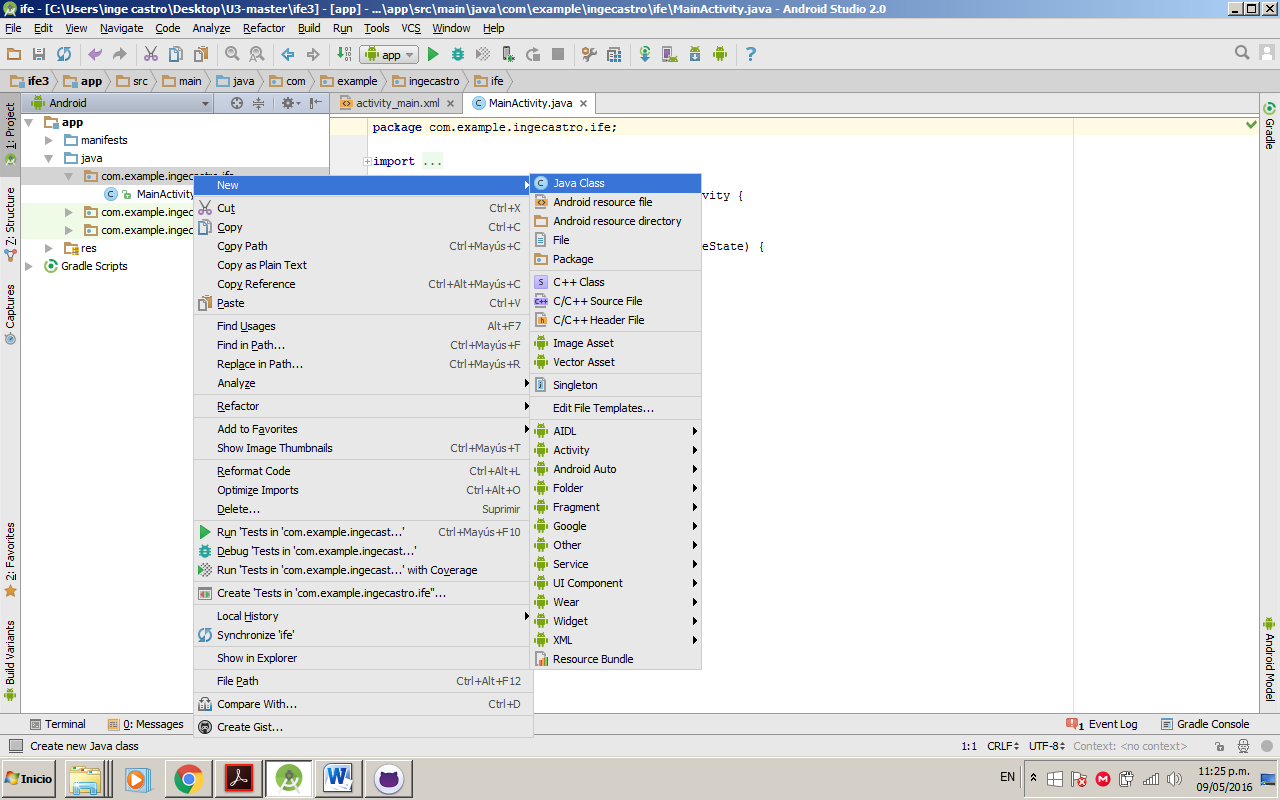
3. Borrado de personas.

4. Modificación de datos.

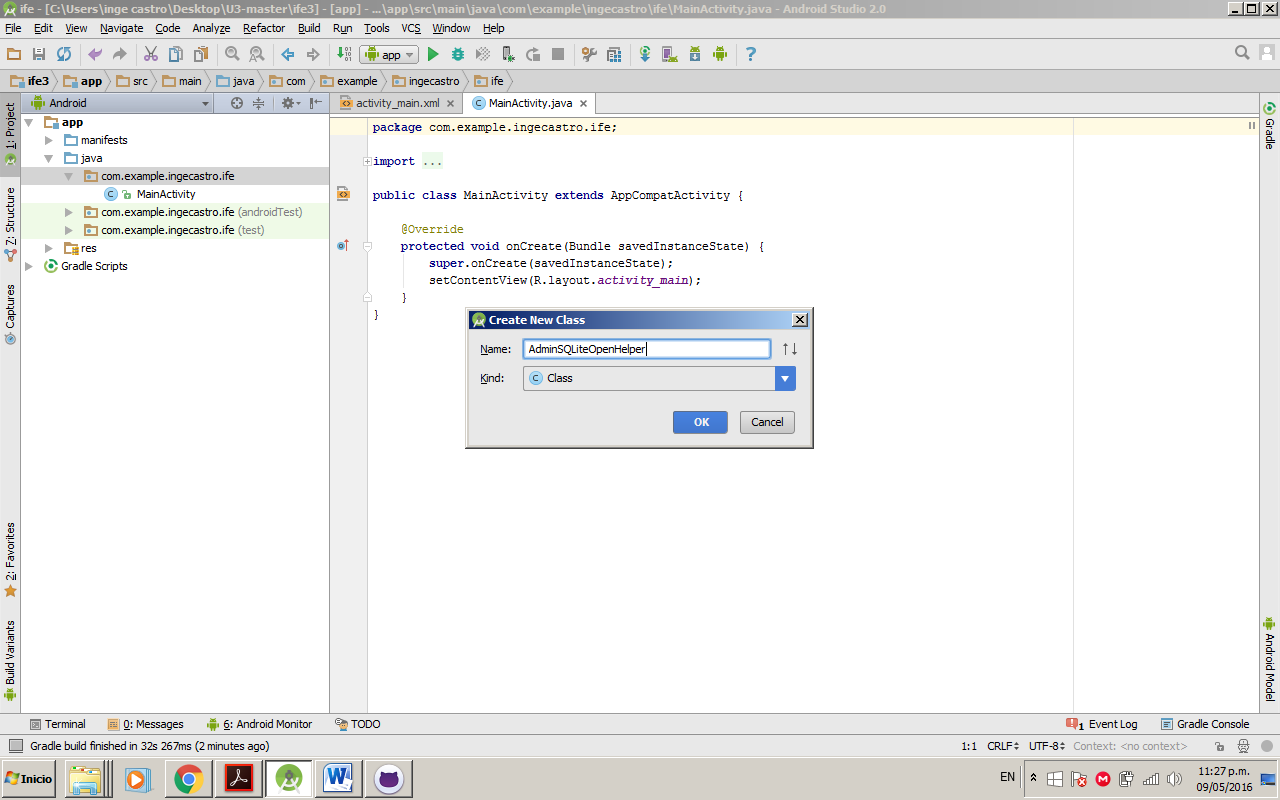
# Desarrollo

Lo primero que haremos es crear una clase que herede de SQLiteOpenHelper. Esta clase nos permite crear la base de datos y actualizar la estructura de tablas y datos iniciales.

Para crear una nueva clase desde Android Studio procedemos a presionar el botón derecho del mouse sobre la carpeta que contienen todos los archivo java del proyecto y seleccionamos New - > Java Class:



En este diálogo ingresamos el nombre de nuestra clase, en nuestro ejemplo la llamaremos AdminSQLiteOpenHelper

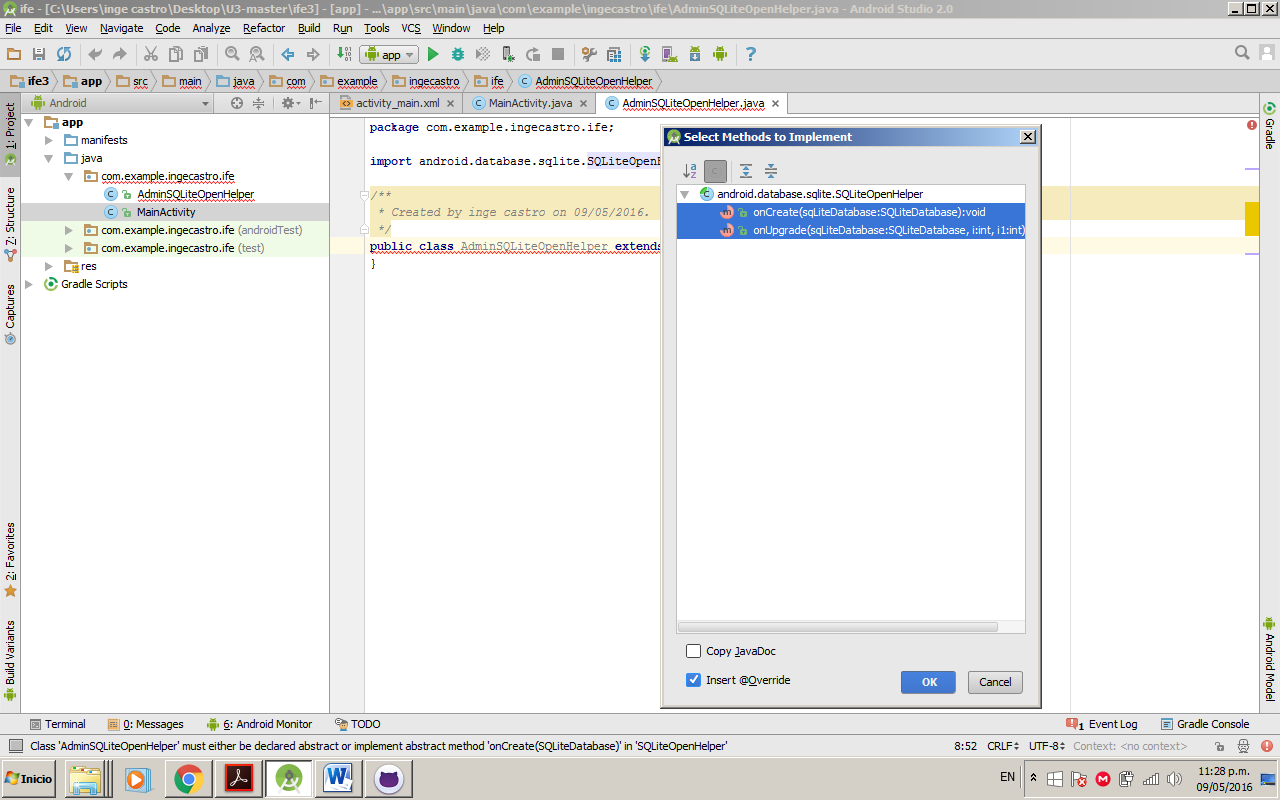


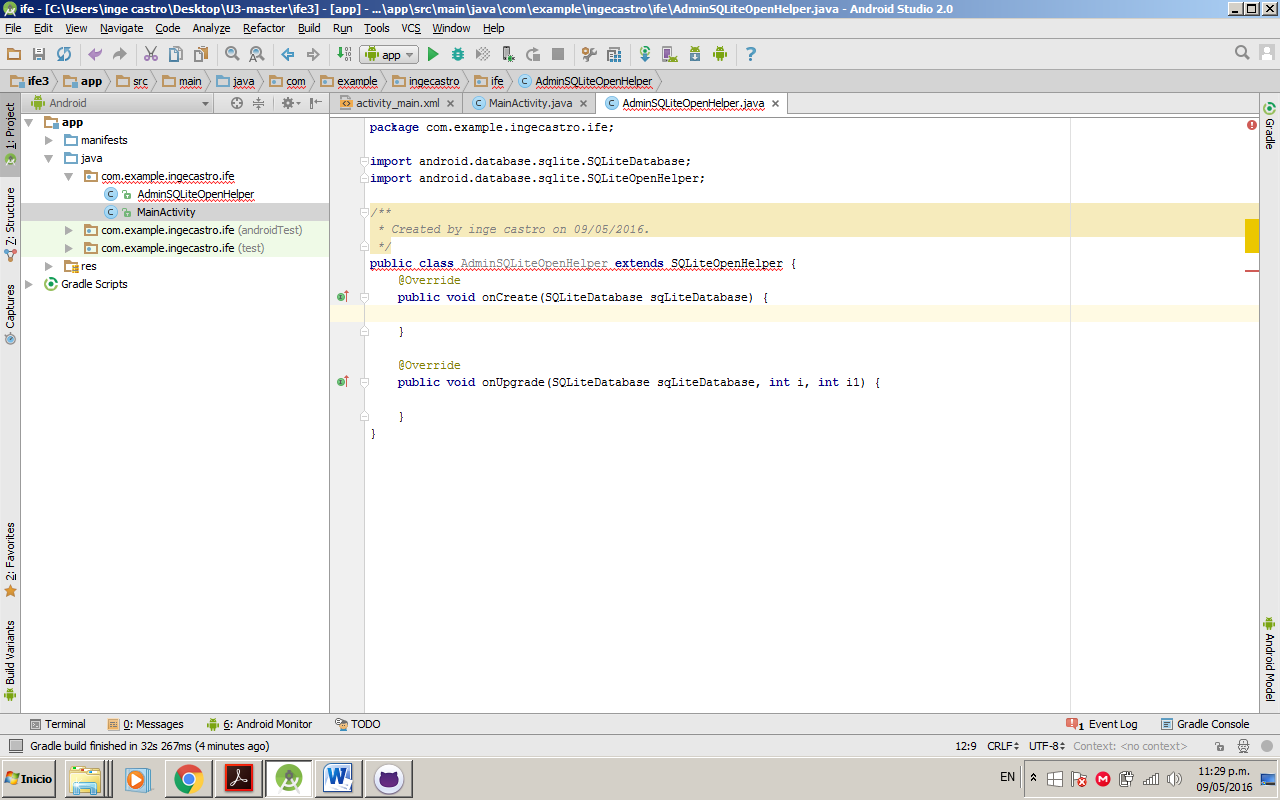
Ahora tenemos que codificar esta clase que tiene por objetivo administrar la base de datos que crearemos. Primero hacemos que nuestra clase herede de la clase SQLiteOpenHelper:

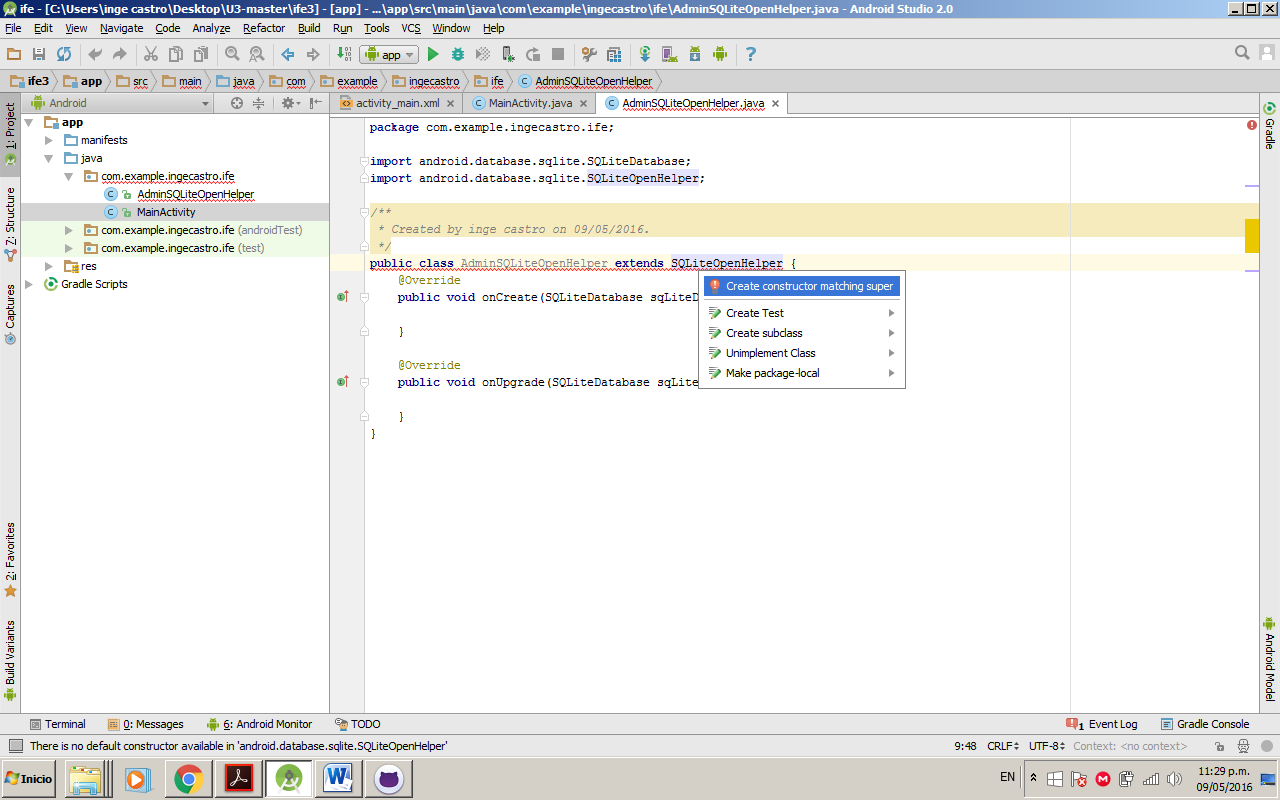
public class AdminSQLiteOpenHelper extends SQLiteOpenHelper{

}

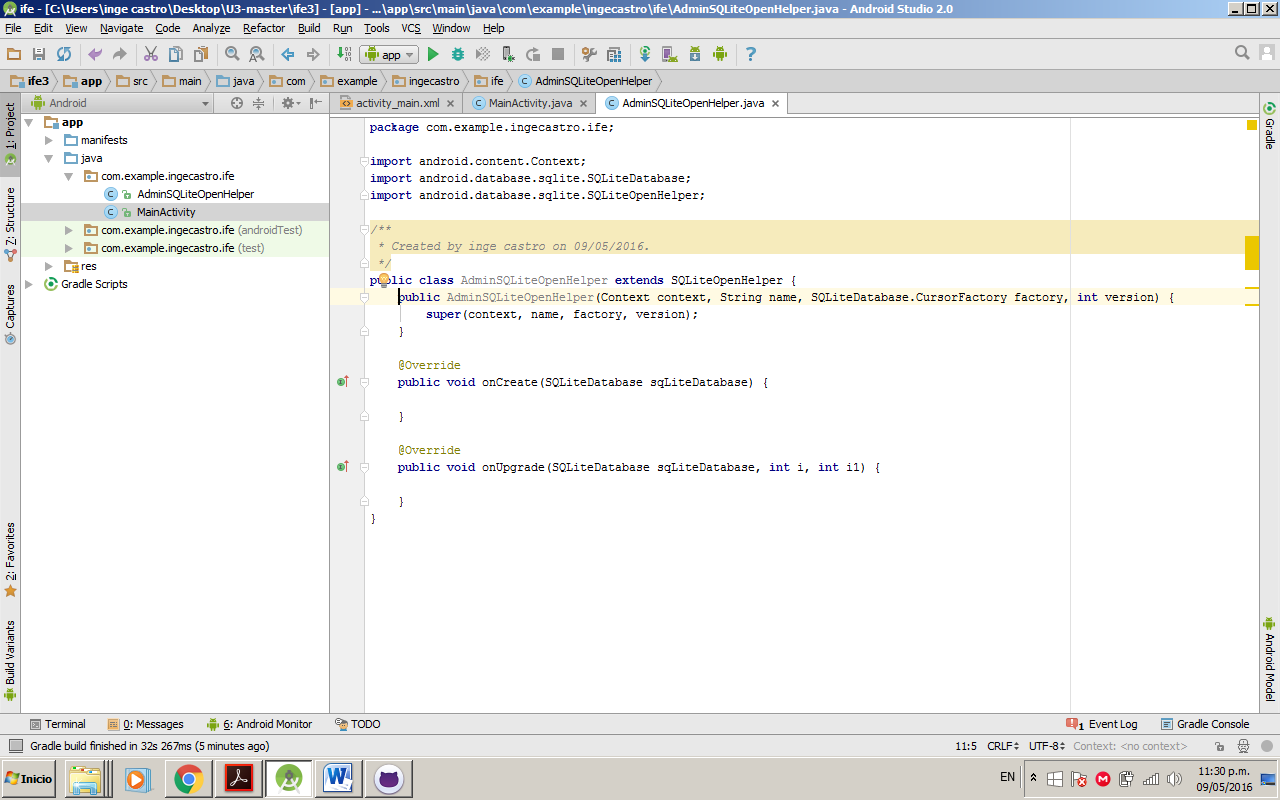
La clase SQLiteOpenHelper requiere que se implementen dos métodos obligatoriamente onCreate y onUpgrade, podemos hacer que el Android Studio nos ayude en su codificación así no tenemos que tipearlos nosotros: presionamos la teclas "ALT" y "ENTER" teniendo el cursor sobre el nombre de la clase "AdminSQLiteOpenHelper" y nos aparece un menú donde seleccionamos "Implement Methods" y luego un diálogo donde seleccionamos los métodos a implementar:



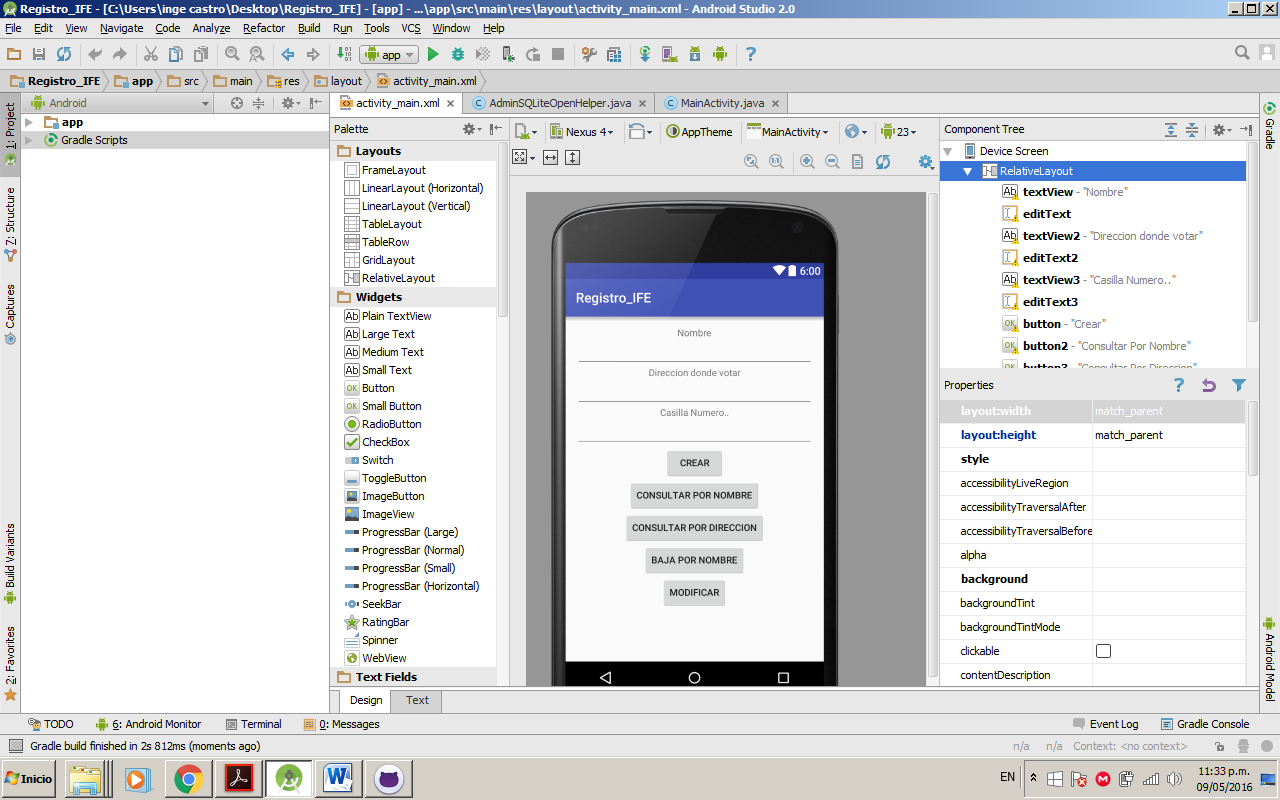




Pero todavía nos marca un error ya que la clase padre SQLiteOpenHelper implementa constructores que tienen parámetros, entonces tenemos que definir el constructor en esta clase, para ello estando el cursor sobre el nombre de la clase "AdminSQLiteOpenHelper" presionamos nuevamente las teclas "ALT" y "Enter" y en el menú contectual que aparece seleccionamos "Create constructor matching super". Aparece un diálogo con todos los constructores que implementa la clase padre (seleccionammos el que tiene tres parámetros):



Ahora implementemos la interfaz visual para resolver nuestro problema. Debemos crear en nuestro archivo activity\_main.xml la siguiente interfaz



Como vemos disponemos tres TextView, tres EditText y cinco Button.

El código fuente de nuestro Activity es el siguiente:

package ar.com.tutorialesya.proyecto019;

import android.content.ContentValues;

import android.database.Cursor;

import android.database.sqlite.SQLiteDatabase;

import android.support.v7.app.AppCompatActivity;

import android.os.Bundle;

import android.view.View;

import android.widget.EditText;

import android.widget.Toast;

public class MainActivity extends AppCompatActivity {

private EditText et1,et2,et3;

@Override

protected void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.activity\_main);

et1=(EditText)findViewById(R.id.editText);

et2=(EditText)findViewById(R.id.editText2);

et3=(EditText)findViewById(R.id.editText3);

}

public void alta(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod = et1.getText().toString();

String descri = et2.getText().toString();

String pre = et3.getText().toString();

ContentValues registro = new ContentValues();

registro.put("codigo", cod);

registro.put("descripcion", descri);

registro.put("precio", pre);

bd.insert("articulos", null, registro);

bd.close();

et1.setText("");

et2.setText("");

et3.setText("");

Toast.makeText(this, "Se cargaron los datos del artículo",

Toast.LENGTH\_SHORT).show();

}

public void consultaporcodigo(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod = et1.getText().toString();

Cursor fila = bd.rawQuery(

"select descripcion,precio from articulos where codigo=" + cod, null);

if (fila.moveToFirst()) {

et2.setText(fila.getString(0));

et3.setText(fila.getString(1));

} else

Toast.makeText(this, "No existe un artículo con dicho código",

Toast.LENGTH\_SHORT).show();

bd.close();

}

public void consultapordescripcion(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String descri = et2.getText().toString();

Cursor fila = bd.rawQuery(

"select codigo,precio from articulos where descripcion='" + descri +"'", null);

if (fila.moveToFirst()) {

et1.setText(fila.getString(0));

et3.setText(fila.getString(1));

} else

Toast.makeText(this, "No existe un artículo con dicha descripción",

Toast.LENGTH\_SHORT).show();

bd.close();

}

public void bajaporcodigo(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod= et1.getText().toString();

int cant = bd.delete("articulos", "codigo=" + cod, null);

bd.close();

et1.setText("");

et2.setText("");

et3.setText("");

if (cant == 1)

Toast.makeText(this, "Se borró el artículo con dicho código",

Toast.LENGTH\_SHORT).show();

else

Toast.makeText(this, "No existe un artículo con dicho código",

Toast.LENGTH\_SHORT).show();

}

public void modificacion(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod = et1.getText().toString();

String descri = et2.getText().toString();

String pre = et3.getText().toString();

ContentValues registro = new ContentValues();

registro.put("codigo", cod);

registro.put("descripcion", descri);

registro.put("precio", pre);

int cant = bd.update("articulos", registro, "codigo=" + cod, null);

bd.close();

if (cant == 1)

Toast.makeText(this, "se modificaron los datos", Toast.LENGTH\_SHORT)

.show();

else

Toast.makeText(this, "no existe un artículo con el código ingresado",

Toast.LENGTH\_SHORT).show();

}

}

Recordar de inicializar la propiedad onClick de cada botón para enlazarlo con el método respectivo: "alta","consultaporcodigo","consultapordescripcion","bajaporcodigo" y "modificacion".

1 - Alta de datos.

Cuando se presiona el botón "ALTA" se ejecuta el método "alta" recordemos inicializar la propiedad "onClick" del botón desde la ventana de visualización del archivo XML.

Lo primero que hacemos en este método es crear un objeto de la clase que planteamos anteriormente y le pasamos al constructor this (referencia del Activity actual), "administracion" (es el nombre de la base de datos que crearemos en el caso que no exista) luego pasamos null y un uno indicando que es la primer versión de la base de datos (en caso que cambiemos la estructura o agreguemos tablas por ejemplo podemos pasar un dos en lugar de un uno para que se ejecute el método onUpgrade donde indicamos la nuestra estructura de la base de datos)

Luego de crear un objeto de la clase AdminSqLiteOpenHelper procedemos a crear un objeto de la clase SQLiteDataBase llamando al método getWritableDatabase (la base de datos se abre en modo lectura y escritura).

Creamos un objeto de la clase ContentValues y mediante el método put inicializamos todos tos campos a cargar.

Seguidamente llamamos al método insert de la clase SQLiteDatabase pasando en el primer parámetro el nombre de la tabla, como segundo parámetro un null y por último el objeto de la clase ContentValues ya inicializado (este método es el que provoca que se inserte una nueva fila en la tabla articulos en la base de datos llamada administracion)

Borramos seguidamente los EditText y mostramos un mensaje para que conozca el operador que el alta de datos se efectuó en forma correcta:

public void alta(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod = et1.getText().toString();

String descri = et2.getText().toString();

String pre = et3.getText().toString();

ContentValues registro = new ContentValues();

registro.put("codigo", cod);

registro.put("descripcion", descri);

registro.put("precio", pre);

bd.insert("articulos", null, registro);

bd.close();

et1.setText("");

et2.setText("");

et3.setText("");

Toast.makeText(this, "Se cargaron los datos del artículo",

Toast.LENGTH\_SHORT).show();

}

2 - Consulta de articulo por código.

Cuando se presiona el botón "CONSULTA POR CODIGO" se ejecuta el método consultaporcodigo:

public void consultaporcodigo(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod = et1.getText().toString();

Cursor fila = bd.rawQuery(

"select descripcion,precio from articulos where codigo=" + cod, null);

if (fila.moveToFirst()) {

et2.setText(fila.getString(0));

et3.setText(fila.getString(1));

} else

Toast.makeText(this, "No existe un artículo con dicho código",

Toast.LENGTH\_SHORT).show();

bd.close();

}

En el método consultaporcodigo lo primero que hacemos es crear un objeto de la clase AdminSQLiteOpenHelper y obtener una referencia de la base de datos llamando al método getWritableDatabase.

Seguidamente definimos una variable de la clase Cursor y la inicializamos con el valor devuelto por el método llamado rawQuery.

La clase Cursos almacena en este caso una fila o cero filas (una en caso que hayamos ingresado un codigo existente en la tabla articulos), llamamos al método moveToFirst() de la clase Cursos y retorna true en caso de existir un articulo con el codigo ingresado, en caso contrario retorna cero.

Para recuperar los datos propiamente dichos que queremos consultar llamamos al método getString y le pasamos la posición del campo a recuperar (comienza a numerarse en cero, en este ejemplo la columna cero representa el campo descripcion y la columna 1 representa el campo precio)

3 - Consulta de articulo por descripción.

Cuando se presiona el botón "CONSULTA POR DESCRIPCION" se ejecuta el método consultapordescripcion:

public void consultapordescripcion(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String descri = et2.getText().toString();

Cursor fila = bd.rawQuery(

"select codigo,precio from articulos where descripcion='" + descri +"'", null);

if (fila.moveToFirst()) {

et1.setText(fila.getString(0));

et3.setText(fila.getString(1));

} else

Toast.makeText(this, "No existe un artículo con dicha descripción",

Toast.LENGTH\_SHORT).show();

bd.close();

}

En el método consultapordescripcion lo primero que hacemos es crear un objeto de la clase AdminSQLiteOpenHelper y obtener una referencia de la base de datos llamando al método getWritableDatabase.

Seguidamente definimos una variable de la clase Cursor y la inicializamos con el valor devuelto por el método llamado rawQuery.

Es importante notar en el where de la clausula SQL hemos dispuesto comillas simples entre el contenido de la variable descri:

"select codigo,precio from articulos where descripcion='" + descri +"'", null);

Esto es obligatorio para los campos de tipo text (en este caso descripcion es de tipo text)

4 - Baja o borrado de datos.

Para borrar uno o más registros la clase SQLiteDatabase tiene un método que le pasamos en el primer parámetro el nombre de la tabla y en el segundo la condición que debe cumplirse para que se borre la fila de la tabla. El método delete retorna un entero que indica la cantidad de registros borrados:

public void bajaporcodigo(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod= et1.getText().toString();

int cant = bd.delete("articulos", "codigo=" + cod, null);

bd.close();

et1.setText("");

et2.setText("");

et3.setText("");

if (cant == 1)

Toast.makeText(this, "Se borró el artículo con dicho código",

Toast.LENGTH\_SHORT).show();

else

Toast.makeText(this, "No existe un artículo con dicho código",

Toast.LENGTH\_SHORT).show();

}

5 - Modificación de datos.

En la modificación de datos debemos crear un objeto de la clase ContentValues y mediante el método put almacenar los valores para cada campo que será modificado. Luego se llama al método update de la clase SQLiteDatabase pasando el nombre de la tabla, el objeto de la clase ContentValues y la condición del where (el cuanto parámetro en este ejemplo no se lo emplea)

public void modificacion(View v) {

AdminSQLiteOpenHelper admin = new AdminSQLiteOpenHelper(this,

"administracion", null, 1);

SQLiteDatabase bd = admin.getWritableDatabase();

String cod = et1.getText().toString();

String descri = et2.getText().toString();

String pre = et3.getText().toString();

ContentValues registro = new ContentValues();

registro.put("codigo", cod);

registro.put("descripcion", descri);

registro.put("precio", pre);

int cant = bd.update("articulos", registro, "codigo=" + cod, null);

bd.close();

if (cant == 1)

Toast.makeText(this, "se modificaron los datos", Toast.LENGTH\_SHORT)

.show();

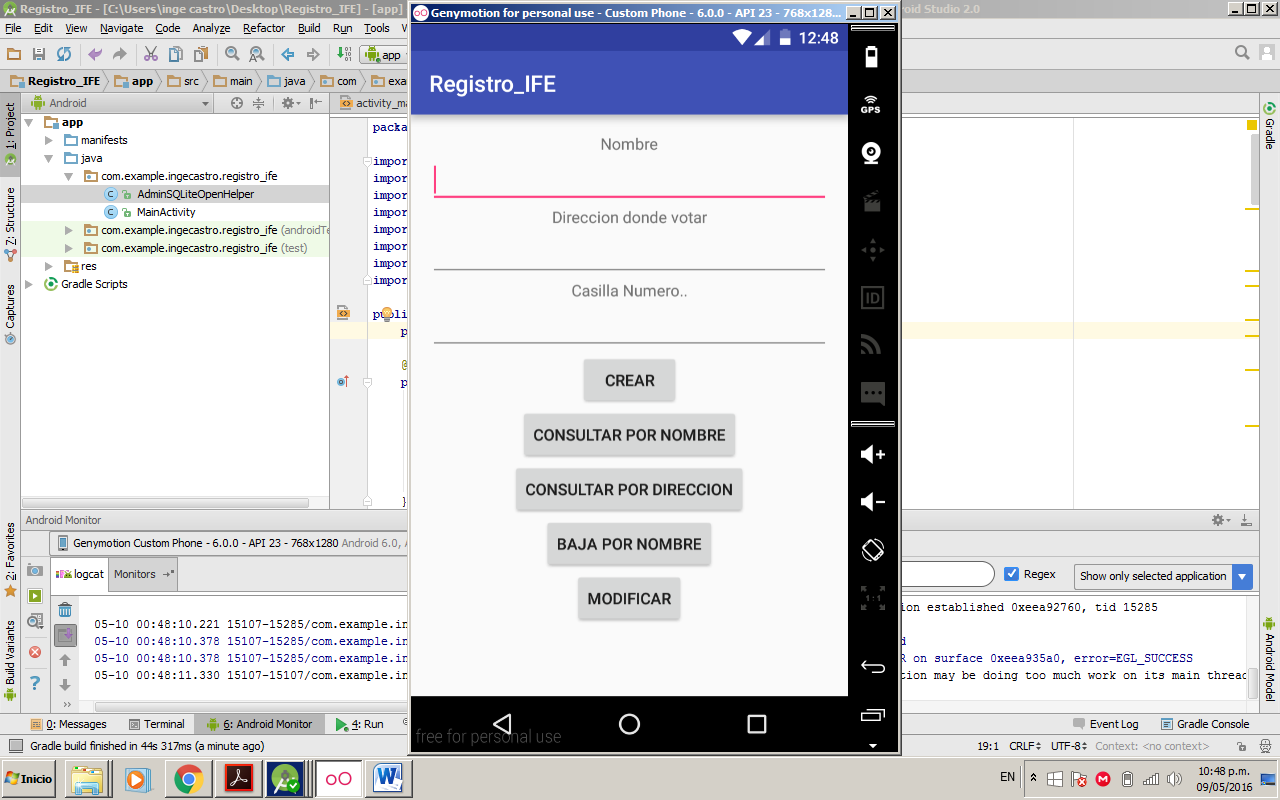
else

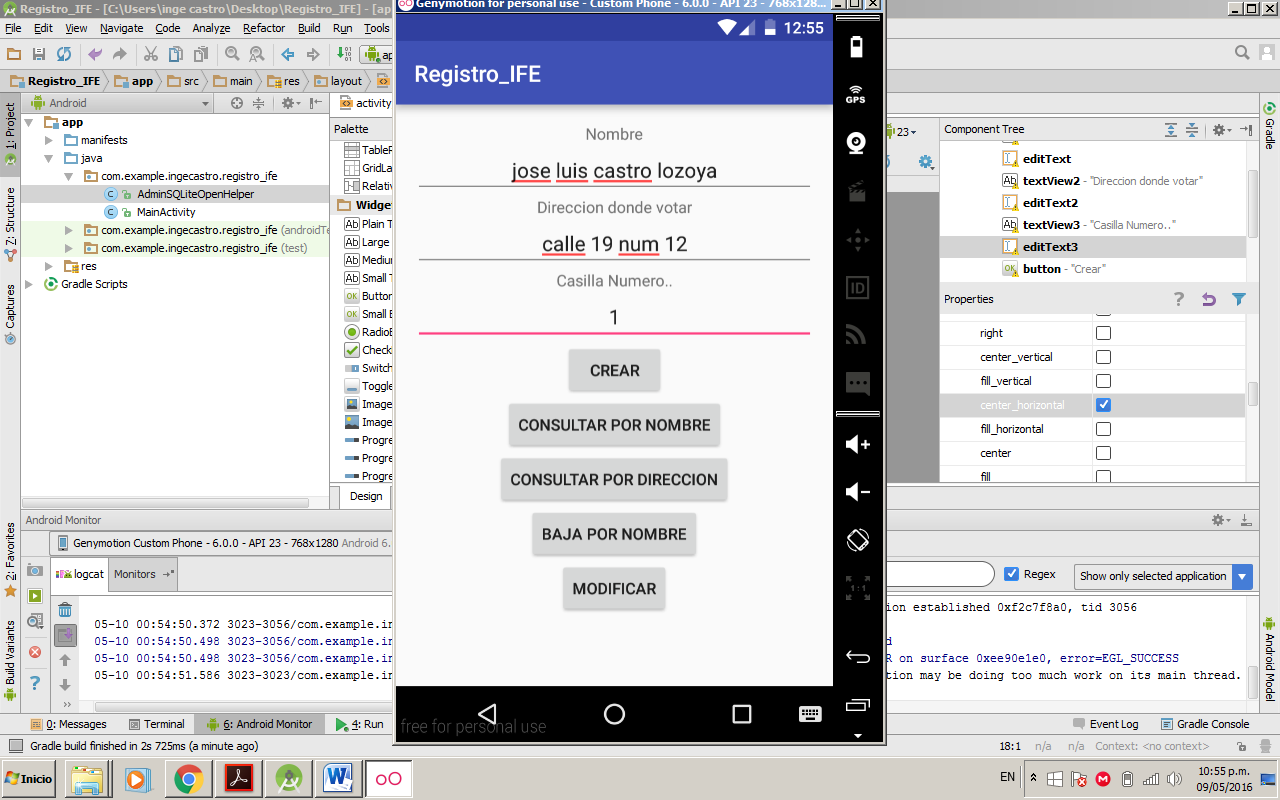
Toast.makeText(this, "no existe un artículo con el código ingresado",

Toast.LENGTH\_SHORT).show();

}

Y después se ejecuta el programa





# Conclusión.

En esta practica aprendimos como almacenar datos por medio de la clase una base de datos, muy recomendable en este caso por ser varios registros,