SENATI

Diseño y Desarrollo de Aplicaciones Móviles

MATERIAL TÉCNICO DE APOYO

RDA
RECURSO DIDÁCTICO PARA EL APRENDIZAJE

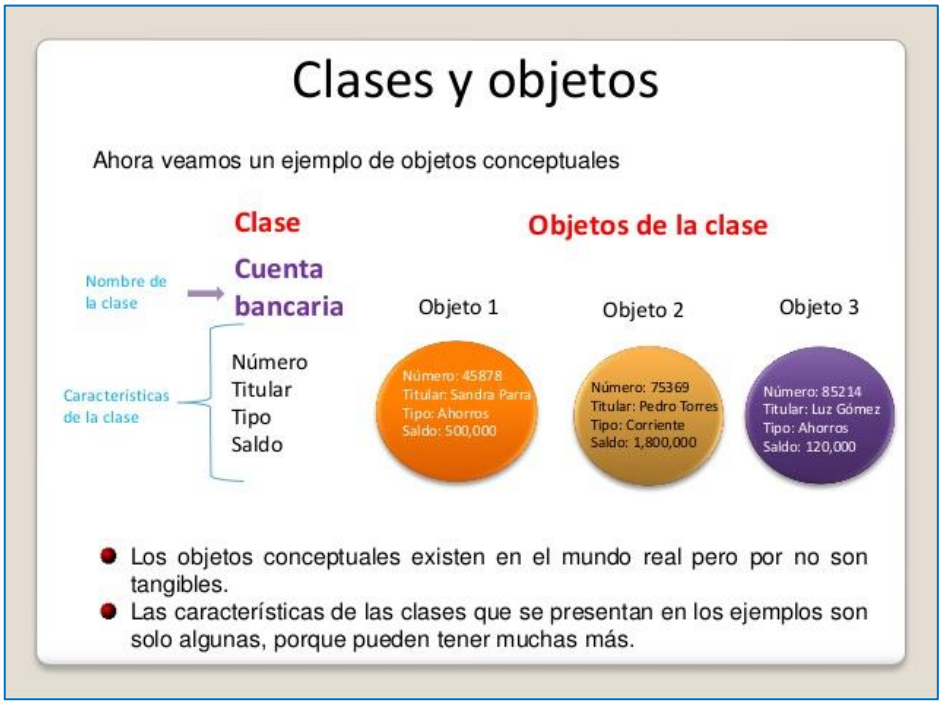
TAREA N° 01

Utiliza el entorno Android Studio y Java.

1. PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA.

La Programación Orientada a Objetos (POO) es un paradigma de programación centrado en el uso de objetos y clases para diseñar y desarrollar software. En el contexto de Java, que es uno de los lenguajes más utilizados para aplicaciones móviles Android, los principios de POO son fundamentales.

- **Conceptos clave en la POO con Java:**
  - **Clases y objetos:**
    - Una clase es un plano o molde que define las características y comportamientos (atributos y métodos) que los objetos tendrán.
    - Un objeto es una instancia de una clase.
    - Ejemplo: Si tienes una clase Coche, puedes crear varios objetos Coche, cada uno con diferentes valores para los atributos (marca, modelo, color, etc.).



Ejemplo de clases y objetos

  - **Encapsulamiento:**
    - Este principio oculta los detalles internos del objeto y solo permite la interacción con él a través de métodos públicos.

- Se logra utilizando modificadores de acceso como private, public, y protected.

- **Herencia:**

  - Permite que una clase (subclase) herede atributos y métodos de otra clase (superclase). Esto promueve la reutilización de código.

  - Ejemplo: Si tienes una clase Vehículo, puedes crear una subclase Coche que herede características generales de Vehículo y agregue funcionalidades específicas.

- **Polimorfismo:**

  - Este concepto permite que una misma operación se comporte de diferentes maneras en diferentes clases. Se logra mediante sobrecarga y sobreescritura de métodos.

  - Ejemplo: Un método arrancar() en la clase Vehículo podría comportarse de manera diferente en una clase Coche y otra clase Motocicleta.

- **Abstracción:**

  - La abstracción implica crear clases y métodos abstractos que solo proporcionan la estructura, dejando que las subclases implementen los detalles. En Java, esto se puede lograr mediante el uso de clases abstractas e interfaces.

- **Aplicación en Android:**

  - Las aplicaciones móviles Android están escritas en Java, y la POO permite estructurar el código de manera modular y reutilizable.

  - Cada actividad en una aplicación Android puede ser vista como un objeto que interactúa con otros objetos, lo que facilita la creación de aplicaciones grandes y complejas.

2. ANDROID STUDIO.

Android Studio es el entorno de desarrollo integrado (IDE) oficial para crear aplicaciones Android. Proporciona todas las herramientas necesarias para desarrollar, depurar, y probar aplicaciones Android de manera eficiente.

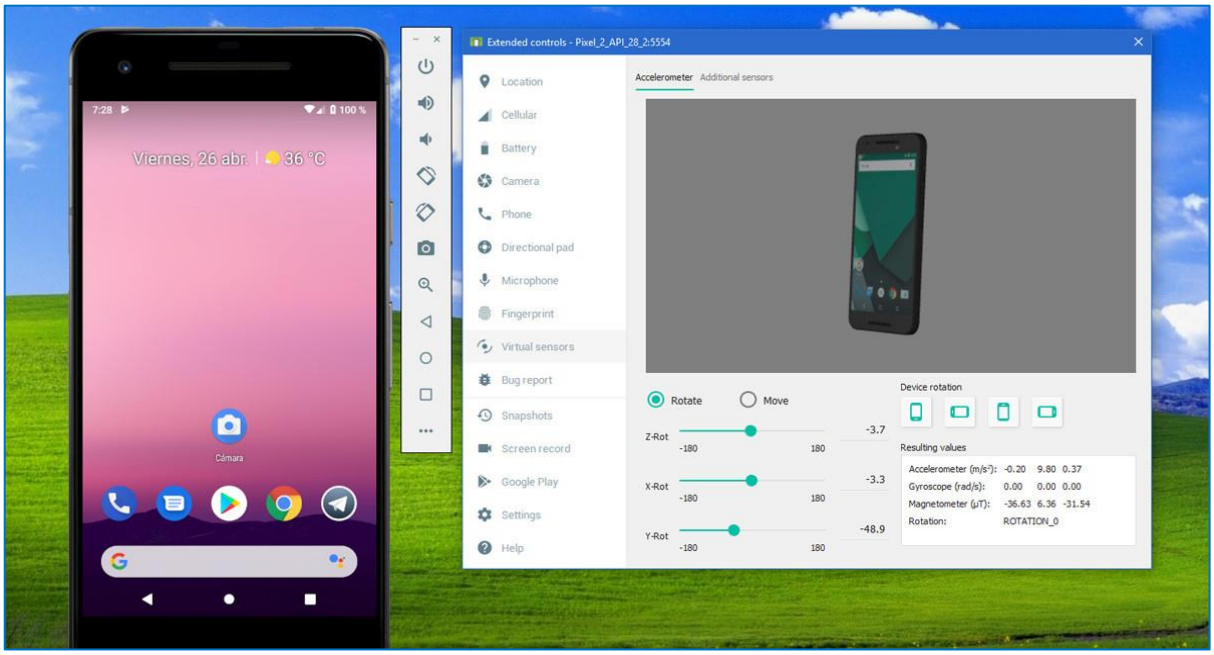
Logo de Andrpid Studio

- **Características principales de Android Studio:**
  - **Interfaz gráfica intuitiva:** Facilita la creación de interfaces de usuario mediante el uso de arrastrar y soltar elementos visuales.
  - **Compatibilidad con Java y Kotlin:** Aunque el desarrollo en Java es común, también soporta Kotlin, que es otro lenguaje de programación moderno y oficial para Android.
  - **Android Virtual Device (AVD):** Permite crear emuladores para probar aplicaciones en diferentes versiones de Android y tamaños de pantalla.
  - **Herramientas de depuración:** Android Studio tiene un depurador robusto que permite rastrear errores, observar el comportamiento de la aplicación en tiempo real y resolver problemas de rendimiento.
  - **Herramientas de rendimiento:** Permite analizar la CPU, el uso de memoria, y la eficiencia de la aplicación, ayudando a optimizar su rendimiento.
- **Flujo básico de trabajo:**
  - **Creación del proyecto:** Los proyectos en Android Studio contienen un archivo AndroidManifest.xml, que define la configuración de la aplicación y los recursos como imágenes y cadenas de texto.
  - **Diseño de la interfaz:** Las interfaces gráficas (UI) se diseñan utilizando XML, permitiendo una estructura clara y separada del código lógico de la aplicación.

- **Escritura de código Java:** La lógica de la aplicación se implementa en clases Java, vinculadas a las actividades definidas en el AndroidManifest.xml.

3. EMULADORES PARA DISPOSITIVOS MÓVILES.

Un emulador de Android es una herramienta que permite simular un dispositivo móvil en la computadora para probar y depurar aplicaciones sin necesidad de un dispositivo físico.



Vista de un emulador para dispositivos móviles en Windows.

- **Características principales de los emuladores:**

  - **Simulación de varios dispositivos:** Puedes emular diferentes versiones de Android (desde las más antiguas hasta las más recientes) y dispositivos con diferentes resoluciones y tamaños de pantalla.

  - **Herramientas de depuración:** Los emuladores permiten ejecutar las aplicaciones en diferentes configuraciones para detectar errores o fallos que pueden ocurrir en determinados dispositivos.

  - **Pruebas de funcionalidad:** Se pueden probar aspectos como la respuesta de la aplicación al girar la pantalla, cambios en la conectividad de red, o el rendimiento de la batería.

- **Creación de un emulador (Android Virtual Device - AVD):**

  - En Android Studio, puedes crear un AVD desde el menú "AVD Manager" y elegir el tipo de dispositivo que deseas emular (teléfonos, tabletas, wearables, etc.).

- Los emuladores también permiten simular sensores de dispositivos como GPS, cámara, giroscopio, entre otros.

- **Ventajas de usar emuladores:**

  - **Costo cero:** No necesitas comprar dispositivos físicos para cada versión de Android o tamaño de pantalla.

  - **Pruebas rápidas:** Se pueden hacer pruebas rápidas sin necesidad de conectar un dispositivo físico cada vez.

4. LENGUAJE XML.

XML (Extensible Markup Language) es un lenguaje de marcado utilizado en Android para definir la interfaz gráfica de usuario y otros recursos como configuraciones, animaciones, y menús.

- **Uso de XML en Android:**

  - **Layout de interfaces gráficas:** Los archivos de diseño en XML permiten definir las posiciones, tamaños, y estilos de los componentes de la UI, como botones, imágenes, y textos.

    - **Ejemplo:** Un archivo activity_main.xml puede definir la estructura visual de la pantalla principal de la aplicación.

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="¡Hola Mundo!" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Presioname" />
</LinearLayout>
```

- **Ventajas de usar XML en Android:**

  - **Separación del diseño y la lógica:** Permite mantener el código lógico (Java) separado de la UI, haciendo el código más limpio y fácil de mantener.

  - **Compatibilidad con diferentes dispositivos:** Los archivos XML pueden ajustarse para crear diseños responsivos que se adapten a diferentes tamaños de pantalla y resoluciones.

  - **Reutilización de recursos:** Al definir recursos como colores, cadenas de texto o estilos en archivos XML, se pueden reutilizar en diferentes partes de la aplicación.

- **Archivos XML importantes en un proyecto Android:**

  - **res/layout:** Contiene los archivos de diseño para las actividades y fragmentos.

  - **res/values:** Contiene recursos como colores, cadenas de texto, y dimensiones.

  - **AndroidManifest.xml:** Define componentes clave de la aplicación como actividades, permisos, y configuraciones generales.

TAREA N°02

Diseña y crea interfaz gráfica de usuario.

1. PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA.

La Programación Orientada a Objetos (POO) es uno de los paradigmas más usados para crear aplicaciones móviles, especialmente en Android, donde Java sigue siendo un lenguaje principal (aunque también se utiliza Kotlin). La POO se centra en el uso de clases y objetos para organizar y gestionar el código de una manera modular y reutilizable.

- **Principios clave de POO en Java:**
  - **Encapsulamiento:**
    - Es el principio que protege el acceso directo a los atributos de los objetos, permitiendo que se acceda a ellos solo a través de métodos.
    - **Ejemplo:** En una aplicación móvil, podrías encapsular los detalles de una transacción bancaria, permitiendo que otros componentes solo puedan verificar el estado, sin modificar los datos directamente.

Ventajas y Desventajas de encapsulamiento

  - **Herencia:**
    - Las clases pueden heredar propiedades y métodos de otras clases. Esto es útil para crear jerarquías de objetos y reutilizar código.

- **Ejemplo:** En Android, Activity es una clase base de la cual heredan todas las actividades de la aplicación.

- **Polimorfismo:**
  - Permite que las clases implementen los mismos métodos de diferentes maneras.
  - En Java, la implementación es a través de la sobrescritura y sobrecarga de métodos.

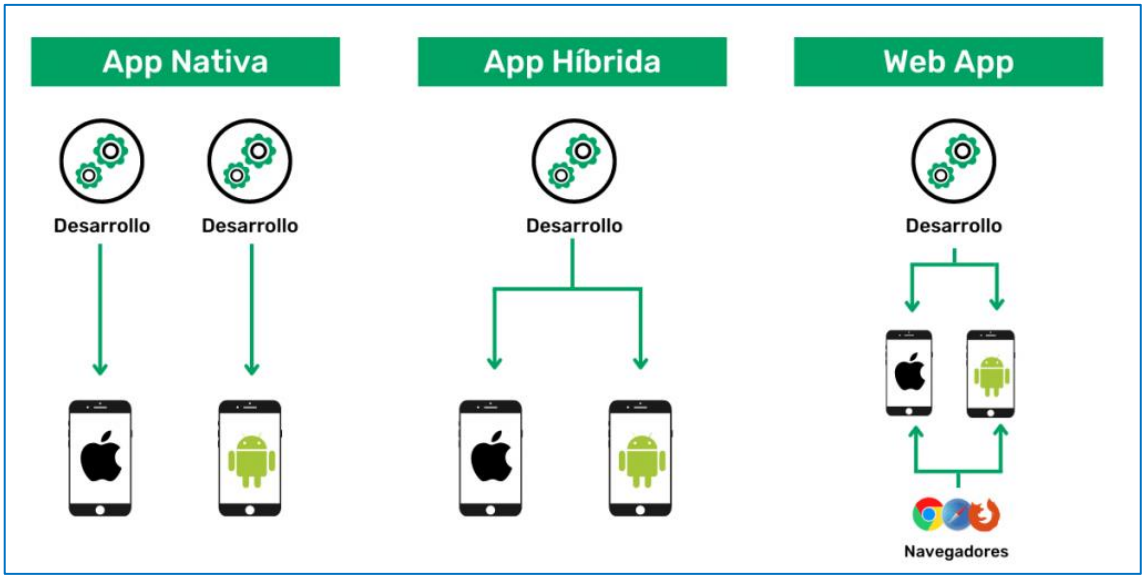
- **Abstracción:**
  - En lugar de manejar objetos de forma genérica, la abstracción permite modelar entidades del mundo real de manera más precisa.
  - Las interfaces y clases abstractas son ejemplos de cómo se implementa la abstracción en Java.

- **Aplicación en Android:**

  Las aplicaciones móviles en Android son ricas en objetos, como actividades, vistas, fragmentos, y servicios, todos creados a partir de clases que siguen los principios de la POO.

2. TIPOS DE APLICACIONES MÓVILES:

Existen varios tipos de aplicaciones móviles, cada una con diferentes características y requerimientos de desarrollo. A continuación, se describen los tres tipos principales:



Tipos de aplicaciones móviles.

- **Nativas.**

  Las aplicaciones nativas son aquellas que se desarrollan específicamente para un sistema operativo (SO) móvil particular, como Android o iOS.

  - **Características:**

    - **Lenguaje:** Desarrolladas con lenguajes nativos del sistema operativo (Java o Kotlin para Android, Swift para iOS).

    - **Rendimiento:** Estas aplicaciones suelen tener el mejor rendimiento, ya que están optimizadas para el hardware y el sistema operativo del dispositivo.

    - **Acceso completo a hardware y APIs:** Las aplicaciones nativas pueden interactuar directamente con los componentes del dispositivo, como la cámara, GPS, sensores, etc.

    - **Interfaz de usuario:** Las aplicaciones nativas permiten un diseño de interfaz más fluido y coherente con las pautas de diseño del sistema operativo.

  - **Ventajas:**

    - Mejor rendimiento y experiencia de usuario (UX).

    - Acceso completo a las funcionalidades del dispositivo.

  - **Desventajas:**

    - Mayor costo de desarrollo, ya que hay que desarrollar y mantener versiones separadas para cada plataforma.

- **Hibrida.**

  Las aplicaciones híbridas son una combinación de tecnologías web y nativas. Se desarrollan utilizando tecnologías web (HTML, CSS, JavaScript) y luego se empaquetan en un contenedor que permite su distribución en tiendas de aplicaciones.

  - **Características:**

    - **Frameworks populares:** Ionic, React Native, y Cordova son algunos de los frameworks que se utilizan para desarrollar aplicaciones híbridas.

    - **Código compartido:** Permite escribir el código una vez y ejecutarlo en varias plataformas, lo que reduce el tiempo de desarrollo.

- **Acceso limitado al hardware:** Aunque se pueden acceder a algunas funciones del hardware del dispositivo a través de plugins, no se tiene el mismo control que con las aplicaciones nativas.

- **Ventajas:**
  - Menor costo de desarrollo y mantenimiento.
  - Desarrollado una vez, se puede implementar en múltiples plataformas.
- **Desventajas:**
  - Rendimiento inferior en comparación con las aplicaciones nativas.
  - No siempre se pueden optimizar completamente las interfaces de usuario.

- **AppWeb.**

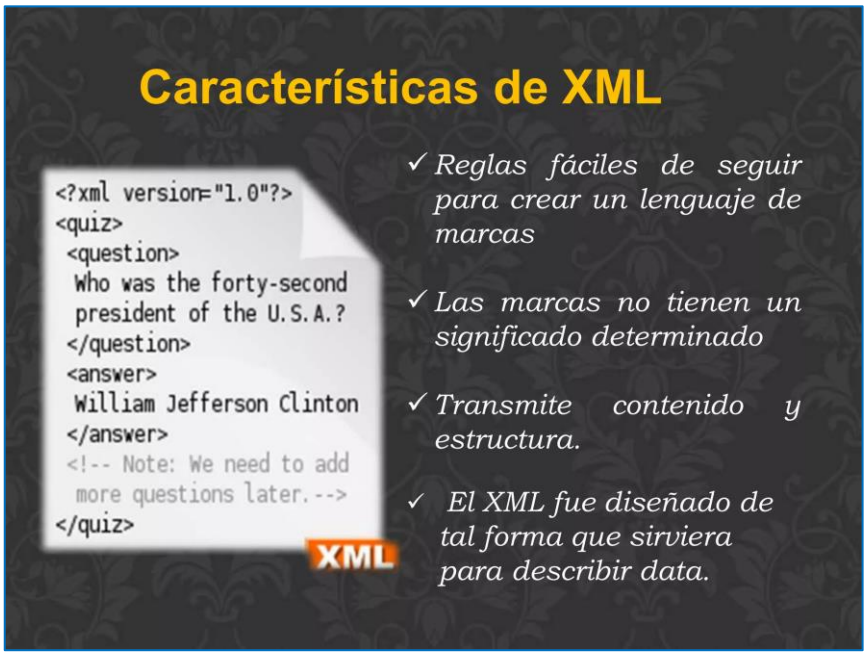
  Las aplicaciones web son básicamente sitios web optimizados para dispositivos móviles. Se accede a ellas a través de un navegador web y no requieren instalación desde una tienda de aplicaciones.

  - **Características:**
    - **Desarrollo con tecnologías web:** Se utilizan HTML5, CSS y JavaScript.
    - **Distribución:** No requieren descarga; los usuarios acceden directamente a la aplicación desde un navegador.
    - **Responsive design:** Generalmente están diseñadas para adaptarse a diferentes tamaños de pantalla.
  - **Ventajas:**
    - No se necesita descargar ni instalar, lo que elimina la barrera de entrada.
    - Se puede acceder desde cualquier dispositivo con un navegador web.
  - **Desventajas:**
    - No tienen acceso completo al hardware del dispositivo.

- Rendimiento inferior en comparación con las aplicaciones nativas y híbridas.

3. LENGUAJE XML.

XML (Extensible Markup Language) es un lenguaje de marcado que Android utiliza principalmente para describir las interfaces gráficas de usuario (UI) y otros recursos como menús, estilos, y configuraciones.



Características de XML.

- **Uso de XML en Android:**
  - **Diseño de interfaz gráfica:** El diseño de las actividades y fragmentos de una aplicación se realiza utilizando archivos XML. Estos archivos definen la disposición, los elementos de la UI y sus propiedades.
    - **Ejemplo:** Un archivo XML para una pantalla de inicio puede definir componentes como botones, imágenes, y textos en un LinearLayout o ConstraintLayout.
  - **Ejemplo de un layout básico en XML:**

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:text="Bienvenido" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Ingresar" />
    </LinearLayout>
```

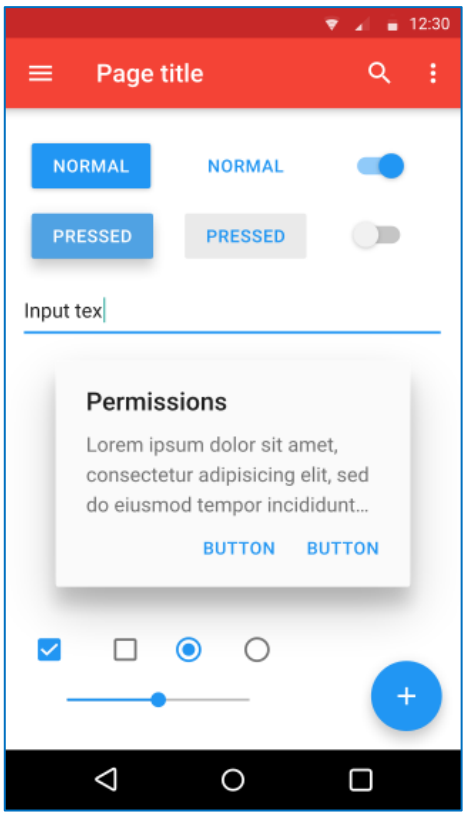
- **Ventajas del uso de XML:**

  - **Separación del código lógico y el diseño:** Mantener la lógica en archivos Java y la interfaz en XML facilita la organización y mantenimiento del código.

  - **Reutilización de recursos:** Los recursos definidos en XML, como colores, estilos o cadenas de texto, pueden ser reutilizados en diferentes partes de la aplicación.

4. MATERIAL DESIGN PARA ANDROID.

Material Design es un lenguaje de diseño creado por Google que establece pautas para crear interfaces gráficas consistentes, intuitivas, y atractivas en aplicaciones Android. Estas pautas cubren todos los aspectos visuales, desde el uso de colores hasta animaciones y transiciones.



Una vista de los elementos que conforman el material design.

- **Principios de Material Design:**
  - **Superficies y sombras:** Las interfaces deben crear una sensación de profundidad utilizando sombras y efectos de elevación para diferenciar elementos.
  - **Transiciones animadas:** Las transiciones y animaciones entre diferentes estados de la interfaz deben ser suaves y proporcionar una retroalimentación visual clara al usuario.
  - **Colores y tipografías:** El uso de una paleta de colores específica y tipografías consistentes ayuda a mantener una identidad visual uniforme en toda la aplicación.
- **Componentes comunes en Material Design:**
  - **Botones flotantes de acción (FAB):** Son botones circulares que realizan una acción principal en la pantalla.
  - **AppBar y ToolBar:** Proporcionan una barra de herramientas en la parte superior de la pantalla que facilita la navegación y ofrece opciones de interacción.
  - **Tarjetas (Cards):** Elementos visuales que agrupan información en una forma de tarjeta, con sombras y bordes redondeados.
- **Implementación en Android Studio:**
  - Android Studio facilita la creación de aplicaciones siguiendo el diseño Material mediante el uso de bibliotecas como com.google.android.material.
- **Ejemplo de un botón flotante (FAB) en XML:**

```
<com.google.android.material.floatingactionbutton.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    android:src="@drawable/ic_add"
    android:contentDescription="Agregar"
    android:tint="@color/white"
        android:backgroundTint="@color/colorAccent" />
```

- **Ventajas de usar Material Design:**
  - Ofrece una experiencia de usuario coherente y atractiva.

- Está bien documentado y ampliamente utilizado, lo que facilita encontrar ejemplos y recursos.

TAREA N° 03

Crea programas con almacenamiento de datos en SQLite.

1. PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA.

La Programación Orientada a Objetos (POO) en Java es fundamental para el desarrollo de aplicaciones móviles robustas y escalables. En el contexto de aplicaciones móviles con SQLite, la POO te permite organizar y estructurar el código para manejar el almacenamiento y gestión de datos de forma eficiente.

- **Instancias.**

  Una instancia es un objeto concreto que se crea a partir de una clase. En Java, una clase actúa como un plano o molde que define las características y comportamientos de los objetos. Al crear una instancia de una clase, asignamos memoria para un objeto único con atributos y comportamientos definidos.

  - **Ejemplo de creación de una instancia:**

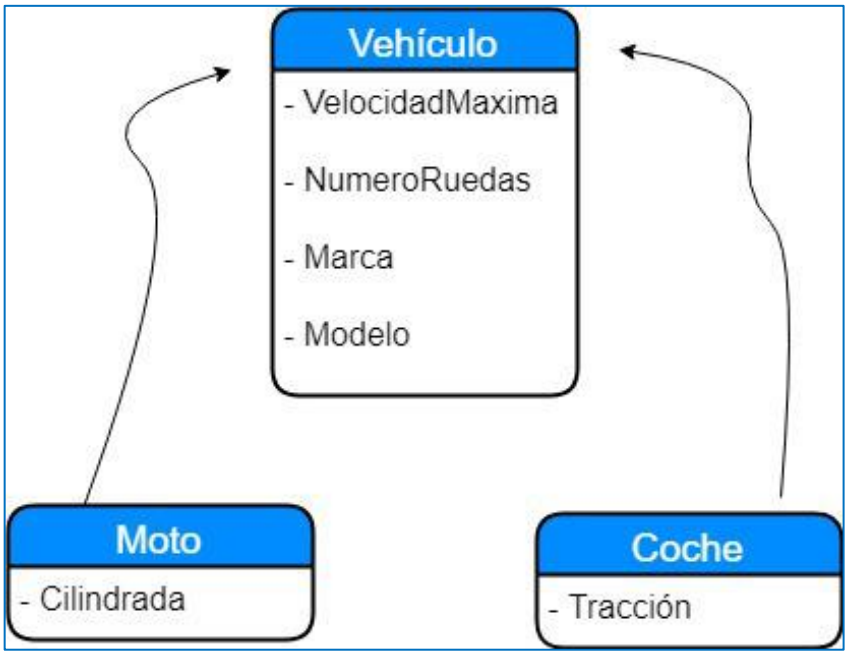
```
public class Usuario {
    String nombre;
    int edad;

    // Constructor
    public Usuario(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public void mostrarInfo() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad);
    }
}

public class Main {
    public static void main(String[] args) {
        // Crear una instancia de la clase Usuario
        Usuario usuario1 = new Usuario("Juan", 25);
        usuario1.mostrarInfo();  // Salida: Nombre: Juan, Edad: 25
    }
     }
```

  En este ejemplo, usuario1 es una instancia de la clase Usuario. En el

contexto de una aplicación móvil con SQLite, podrías tener instancias para representar entidades como productos, usuarios o pedidos que se almacenarán en la base de datos.

- **Herencia.**

  La herencia es un concepto que permite a una clase derivar o heredar propiedades y métodos de otra clase. Esto fomenta la reutilización del código y facilita la creación de jerarquías de clases.



Ejemplo de herencia con vehículo.

  - **Ejemplo de herencia en Java:**

```
public class Persona {
    String nombre;
    int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }
}

public class Estudiante extends Persona {
    String matricula;

    public Estudiante(String nombre, int edad, String matricula) {
        super(nombre, edad);  // Llama al constructor de la clase padre
        this.matricula = matricula;
    }
```

```
    public void mostrarInfo() {
        System.out.println("Nombre: " + nombre + ", Edad: " + edad + ",
     Matrícula: " + matricula);
    }
     }
```

En este caso, la clase Estudiante hereda los atributos y métodos de Persona, y añade el atributo matricula. En una aplicación móvil, podrías tener clases que heredan características comunes de una clase base para gestionar diferentes tipos de usuarios o datos.

- **Porlimorfismo.**

El polimorfismo permite que los objetos de diferentes clases se traten como objetos de una clase común. En Java, el polimorfismo se logra mediante sobrecarga y sobrescritura de métodos.

- **Ejemplo de polimorfismo:**

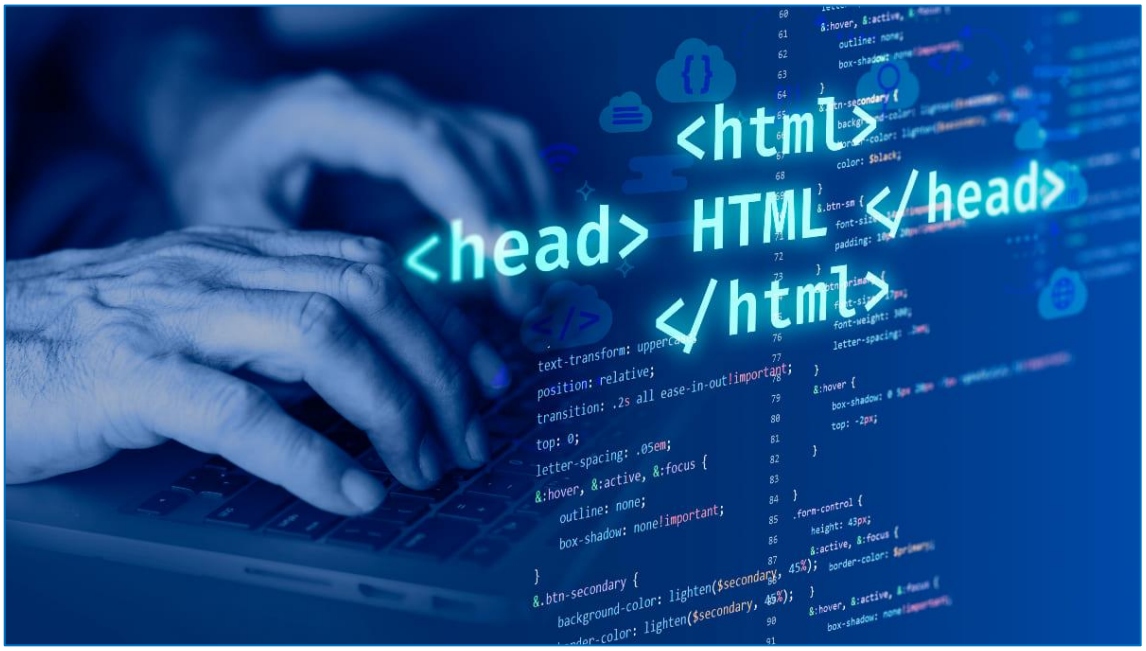
```
public class Animal {
    public void hacerSonido() {
        System.out.println("El animal hace un sonido.");
    }
}

public class Perro extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("El perro ladra.");
    }
}

public class Gato extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("El gato maúlla.");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal miAnimal = new Animal();
        Animal miPerro = new Perro();
        Animal miGato = new Gato();

        miAnimal.hacerSonido();  // Salida: El animal hace un sonido.
        miPerro.hacerSonido();   // Salida: El perro ladra.
```

```
        miGato.hacerSonido();    // Salida: El gato maúlla.
    }
     }
```

El polimorfismo es útil cuando se quiere ejecutar comportamientos diferentes en objetos que comparten una estructura base. En una aplicación que utiliza SQLite, podrías tener diferentes tipos de consultas o transacciones que se comportan de manera diferente, pero comparten la misma interfaz.

2. LENGUAJE XML.

XML (Extensible Markup Language) es ampliamente utilizado en Android para definir la estructura y los componentes de la interfaz de usuario. Sin embargo, también se puede usar para otros propósitos como la configuración y almacenamiento de datos simples. En el caso de aplicaciones móviles con SQLite, XML es crucial para definir los diseños (layouts) de las actividades y vistas.



Lenguaje XML.

- **Ejemplo de un layout XML para una actividad con SQLite:**

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/label_nombre"
        android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:text="Nombre" />

<EditText
    android:id="@+id/input_nombre"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

<Button
    android:id="@+id/btn_guardar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Guardar" />

<TextView
    android:id="@+id/output_resultado"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
 </LinearLayout>
```

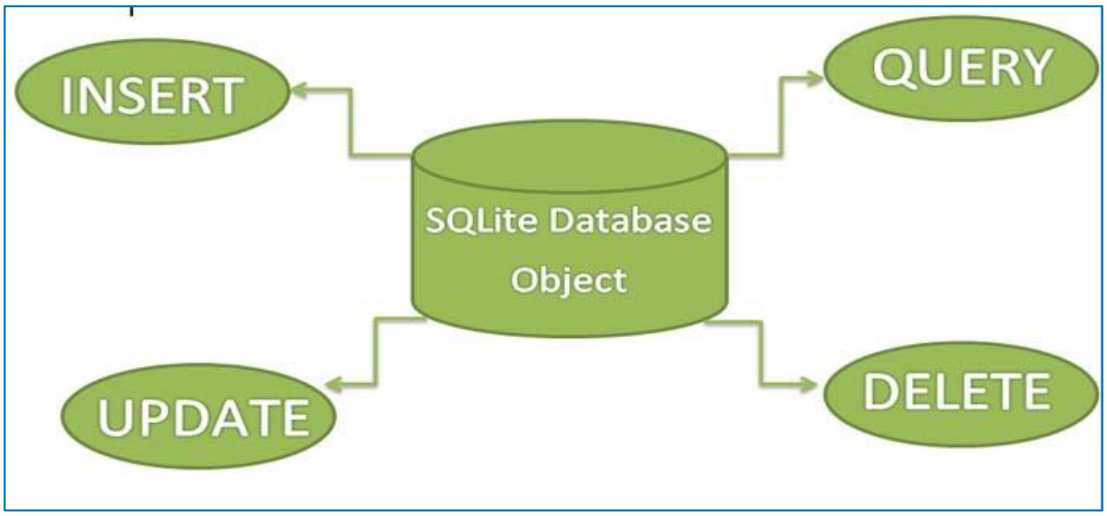
Este archivo XML describe una interfaz simple con un campo de entrada de texto y un botón para guardar los datos en una base de datos SQLite. La interacción con estos elementos ocurre en la lógica del programa (en archivos Java).

3. CONSULTAS SQL Y BASE DE DATOS.

SQLite es una base de datos relacional embebida que es muy ligera y es parte del SDK de Android. No requiere un servidor separado y las bases de datos se almacenan localmente en el dispositivo del usuario. Las consultas SQL son esenciales para interactuar con los datos almacenados en SQLite.

Logo de SQLite.

- **Creación de una base de datos en SQLite**



Comandos SQL en SQLite.

En Android, puedes crear y gestionar una base de datos SQLite utilizando la clase SQLiteOpenHelper. A continuación, se muestra un ejemplo básico de cómo crear una base de datos y una tabla.

```
public class MiDatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "miBaseDeDatos.db";
    private static final int DATABASE_VERSION = 1;

    // Constructor
    public MiDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    // Crear tablas
    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_TABLE = "CREATE TABLE usuarios (" +
                "id INTEGER PRIMARY KEY AUTOINCREMENT," +
                "nombre TEXT," +
                "edad INTEGER)";
        db.execSQL(CREATE_TABLE);
    }

    // Actualizar base de datos
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
     newVersion) {
        db.execSQL("DROP TABLE IF EXISTS usuarios");
        onCreate(db);
    }
     }
```

- **Insertar datos.**

  Para insertar datos en la base de datos SQLite, se utiliza el método insert() que proporciona la clase SQLiteDatabase.

```
public void insertarUsuario(String nombre, int edad) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues valores = new ContentValues();
    valores.put("nombre", nombre);
    valores.put("edad", edad);

    db.insert("usuarios", null, valores);
    db.close();
     }
```

- **Consultar datos.**

  Para consultar datos de la base de datos, se puede utilizar el método query() o ejecutar sentencias SQL directas con rawQuery().

```
public Cursor obtenerUsuarios() {
    SQLiteDatabase db = this.getReadableDatabase();
    return db.rawQuery("SELECT * FROM usuarios", null);
     }
```

  - **Iterar sobre el cursor:**

```
public void mostrarUsuarios() {
    Cursor cursor = obtenerUsuarios();

    if (cursor.moveToFirst()) {
        do {
            String nombre =
     cursor.getString(cursor.getColumnIndex("nombre"));
            int edad = cursor.getInt(cursor.getColumnIndex("edad"));
            System.out.println("Nombre: " + nombre + ", Edad: " + edad);
        } while (cursor.moveToNext());
    }
    cursor.close();
     }
```

- **Actualizar y eliminar datos.**

  También puedes realizar operaciones de actualización y eliminación de datos en la base de datos SQLite.

- **Actualizar un registro:**

```
public void actualizarUsuario(int id, String nuevoNombre, int
    nuevaEdad) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues valores = new ContentValues();
    valores.put("nombre", nuevoNombre);
    valores.put("edad", nuevaEdad);

    db.update("usuarios", valores, "id = ?", new
    String[]{String.valueOf(id)});
    db.close();
    }
```

- **Eliminar un registro:**

```
public void eliminarUsuario(int id) {
    SQLiteDatabase db = this.getWritableDatabase();
    db.delete("usuarios", "id = ?", new String[]{String.valueOf(id)});
    db.close();
    }
```

TAREA N° 04

Usa y crea aplicaciones con API REST.

1. PROGRAMACIÓN ORIENTADA A OBJETOS CON JAVA.

La Programación Orientada a Objetos (POO) es esencial para el desarrollo de aplicaciones robustas y escalables en Java. Es la base del paradigma de programación que organiza el software en torno a objetos, cada uno con atributos y comportamientos definidos por clases. Al trabajar con APIs REST, Java permite estructurar los datos de manera eficiente, utilizando objetos que se mapean a los datos obtenidos desde las API.

API y API REST.

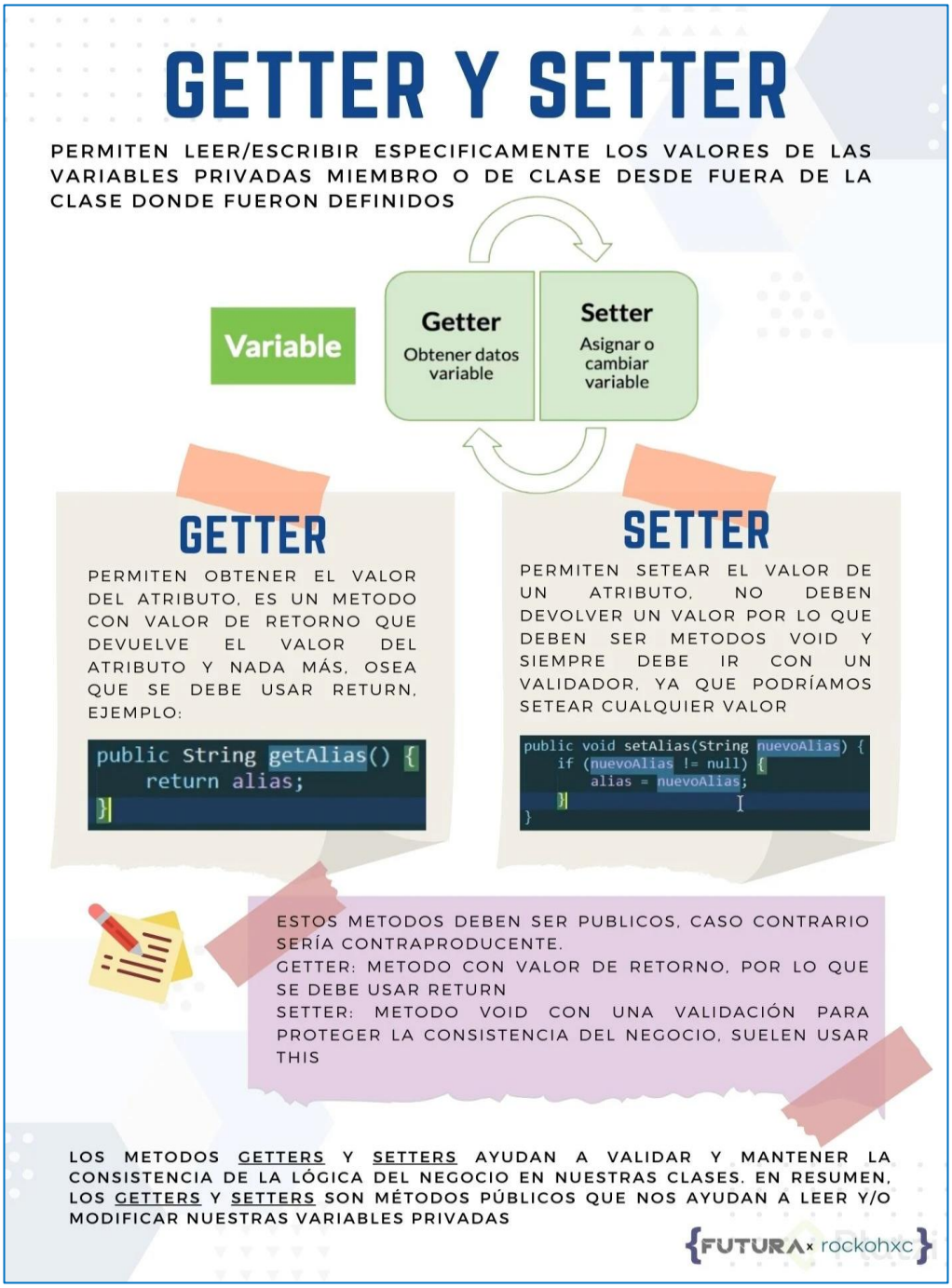
- **Clases e instancias**

  En POO, una clase es el plano que define las características y comportamientos de un objeto, mientras que una instancia es un objeto creado a partir de una clase.

- **Ejemplo básico de clase e instancia:**

```
public class Usuario {
    private String nombre;
    private String email;

    // Constructor
    public Usuario(String nombre, String email) {
        this.nombre = nombre;
        this.email = email;
    }

    // Métodos getter y setter
    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

En el contexto de una API REST, podrías recibir un JSON con la información de un usuario, y luego mapear esos datos a una instancia de la clase Usuario para gestionarlo dentro de la aplicación.



Getter y Setter.

- **Herencia.**

  La herencia permite que una clase derive de otra, reutilizando código común y proporcionando una estructura organizada. En una aplicación con APIs REST, podrías tener una clase base con atributos y métodos compartidos por diferentes tipos de datos.

  - **Ejemplo de herencia:**

```
public class Persona {
    private String nombre;
    private int edad;
```

```
    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    public String getNombre() {
        return nombre;
    }
}

public class Empleado extends Persona {
    private String departamento;

    public Empleado(String nombre, int edad, String departamento) {
        super(nombre, edad);
        this.departamento = departamento;
    }

    public String getDepartamento() {
        return departamento;
    }
 }
```

En este ejemplo, Empleado hereda los atributos y métodos de Persona, permitiendo reutilizar código para manejar diferentes entidades de un sistema.

- **Polimorfismo.**

  El polimorfismo permite que los objetos se comporten de manera diferente dependiendo de su tipo, incluso cuando comparten la misma interfaz o clase base.

  - **Ejemplo de polimorfismo:**

```
public class ApiRequest {
    public void ejecutar() {
        System.out.println("Ejecutando petición API...");
    }
}

public class GetRequest extends ApiRequest {
    @Override
    public void ejecutar() {
        System.out.println("Ejecutando GET...");
    }
}
```

```
public class PostRequest extends ApiRequest {
    @Override
    public void ejecutar() {
        System.out.println("Ejecutando POST...");
    }
}

public class Main {
    public static void main(String[] args) {
        ApiRequest getRequest = new GetRequest();
        ApiRequest postRequest = new PostRequest();

        getRequest.ejecutar(); // Salida: Ejecutando GET...
        postRequest.ejecutar(); // Salida: Ejecutando POST...
    }
     }
```

El polimorfismo en Java permite manejar diferentes tipos de solicitudes de API (GET, POST) con el mismo código base, ajustándose a diferentes comportamientos.

2. LEER DATOS DEL TIPO JSON EN JAVA.

Formato Json.

En una aplicación móvil que consume APIs REST, los datos generalmente se reciben en formato JSON (JavaScript Object Notation). Este formato es fácil de leer tanto para humanos como para máquinas, y se utiliza ampliamente en APIs REST. En Java, puedes usar bibliotecas como Gson o Jackson para procesar y manipular JSON.

- **Biblioteca Gson para leer JSON.**

  La biblioteca Gson de Google es muy utilizada para convertir datos JSON en objetos Java y viceversa.

  - **Ejemplo de conversión de JSON a un objeto Java:**

```
import com.google.gson.Gson;

public class Main {
    public static void main(String[] args) {
        // Cadena JSON
        String json = "{ 'nombre': 'Juan', 'email': 'juan@example.com' }";

        // Crear instancia de Gson
        Gson gson = new Gson();

        // Convertir JSON a objeto Java
        Usuario usuario = gson.fromJson(json, Usuario.class);

        // Mostrar los datos del objeto
        System.out.println("Nombre: " + usuario.getNombre());
        System.out.println("Email: " + usuario.getEmail());
    }
     }
```

  En este ejemplo, el JSON se convierte en una instancia de la clase Usuario. Esto es muy útil cuando se manejan respuestas de API en formato JSON.

- **Enviar y recibir datos JSON con API REST.**

  Para enviar datos JSON en una solicitud API, puedes usar HttpURLConnection o bibliotecas como Retrofit o OkHttp.

  - **Ejemplo de envío de datos JSON con Retrofit:**

```
public interface ApiService {
    @POST("usuarios")
    Call<Usuario> crearUsuario(@Body Usuario usuario);
     }
```

Retrofit facilita la gestión de solicitudes HTTP, mapeando los datos JSON directamente en objetos Java.

3. LENGUAJE XML.

XML (Extensible Markup Language) es otro formato comúnmente utilizado en Android, sobre todo para definir la interfaz de usuario (layouts) y la configuración de ciertos aspectos de la aplicación. Si bien JSON es el formato preferido para intercambiar datos entre una API y una aplicación, XML aún es relevante para configurar y estructurar componentes en Android.

- **Uso de XML en Android.**

  En Android, XML se utiliza para definir elementos como layouts y vistas. Cada actividad o fragmento tiene un archivo XML que describe su interfaz gráfica.

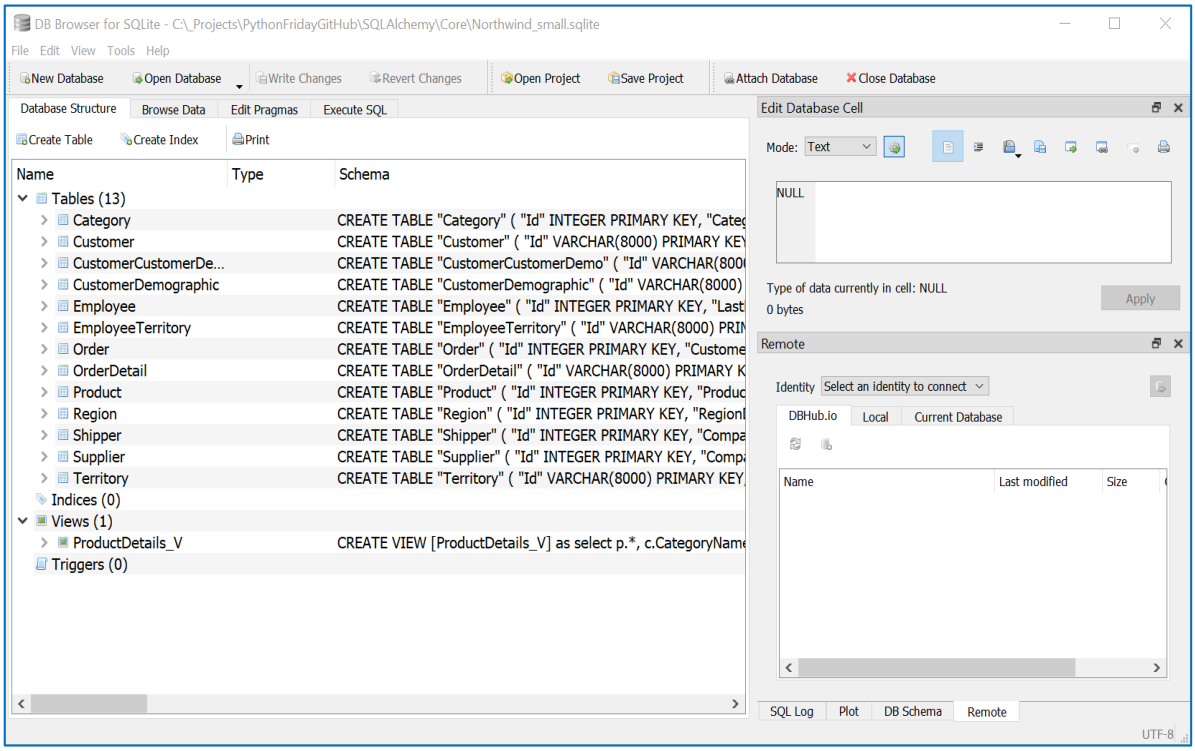
  - **Ejemplo de archivo XML de layout:**

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <TextView
        android:id="@+id/text_view"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bienvenido!" />

    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enviar" />
</LinearLayout>
```

Aquí, los elementos UI como TextView y Button se describen en un archivo XML que será inflado (inflated) en tiempo de ejecución.

4. CONSULTAS SQL Y BASE DE DATOS.

El uso de SQLite es común en aplicaciones móviles para gestionar bases de datos locales. A menudo, las aplicaciones móviles almacenan datos localmente y luego los sincronizan con un servidor a través de APIs REST.



Vista de SQLite.

Aquí es donde entran las Consultas SQL y el acceso a bases de datos.

- **Crear una base de datos con SQLite.**

  En Android, puedes gestionar bases de datos SQLite usando la clase SQLiteOpenHelper.

  - **Ejemplo de creación de una tabla en SQLite:**

```
public class MiDatabaseHelper extends SQLiteOpenHelper {

    private static final String DATABASE_NAME = "miBaseDatos.db";
    private static final int DATABASE_VERSION = 1;

    public MiDatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        String CREATE_TABLE = "CREATE TABLE usuarios (" +
                              "id INTEGER PRIMARY KEY AUTOINCREMENT, "
     +
                              "nombre TEXT, " +
                              "email TEXT)";
        db.execSQL(CREATE_TABLE);
    }
```

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int
 newVersion) {
   db.execSQL("DROP TABLE IF EXISTS usuarios");
   onCreate(db);
}
 }
```

- **Consultar y modificar datos.**

  Puedes realizar consultas SQL para obtener o modificar datos almacenados localmente.

  - **Ejemplo de insertar un usuario:**

```
public void insertarUsuario(String nombre, String email) {
   SQLiteDatabase db = this.getWritableDatabase();
   ContentValues valores = new ContentValues();
   valores.put("nombre", nombre);
   valores.put("email", email);

   db.insert("usuarios", null, valores);
   db.close();
 }
```

  - **Ejemplo de consultar datos:**

```
public Cursor obtenerUsuarios() {
   SQLiteDatabase db = this.getReadableDatabase();
   return db.rawQuery("SELECT * FROM usuarios", null);
 }
```

- **Sincronización de base de datos con API REST**

  Una aplicación móvil puede almacenar datos localmente en SQLite y sincronizar esos datos con un servidor a través de una API REST. Por ejemplo, al realizar una solicitud POST, los datos se pueden enviar en formato JSON a la API, que luego actualiza la base de datos remota.

REFERENCIAS

- Blasco, J. L. (2023a, octubre 27). Introducción a POO en Java: Objetos y clases. *Openwebinars.net*. https://openwebinars.net/blog/introduccion-a-poo-en-java-objetos-y-clases/
- Blasco, J. L. (2023b, noviembre 3). Introducción a POO en Java: Atributos y constructores. *Openwebinars.net*. https://openwebinars.net/blog/introduccion-a-poo-en-java-atributos-y-constructores/
- Coppola, M. (2022, noviembre 23). *Qué es una API REST, para qué sirve y ejemplos*. Hubspot.es. https://blog.hubspot.es/website/que-es-api-rest
- *CRUD: la base de la gestión de datos*. (2019, septiembre 4). IONOS Digital Guide; IONOS. https://www.ionos.com/es-us/digitalguide/paginas-web/desarrollo-web/crud-las-principales-operaciones-de-bases-de-datos/
- *Gson (JSON) en Java, con ejemplos*. (2017, mayo 2). Jarroba. https://jarroba.com/gson-json-java-ejemplos/
- *Introducción a XML*. (s/f). MDN Web Docs. Recuperado el 25 de octubre de 2024, de https://developer.mozilla.org/es/docs/Web/XML/XML_introduction
- k1y8j. (2017, noviembre 27). Tipos de aplicaciones móviles. *Ideas Creativas*. https://ideascreativas.com.ec/blog/tipos-de-aplicaciones-moviles/
- Martinez, A. (2021, enero 26). *Tipos de aplicaciones móviles: nativas, web e híbridas*. Future Space S.A. https://www.futurespace.es/tipos-de-aplicaciones-moviles/

RDA

RECURSO DIDÁCTICO PARA EL APRENDIZAJE