

**UNIVERSIDAD ACADEMICA DE LA REGION NORTE DE GUERRERO UT**

**Docente**; ING; Abel Jerónimo Varga

**Alumno:**

* Roberto Chauteco Bello

**Documentación: 4Figuras Geométricas**

Chilapa De Álvarez Guerrero, noviembre

INDICE

[Ilustración 1ARCHIVO MANISFEST. 3](#_Toc150474401)

[Ilustración 2ListView\_Personalizado 4](#_Toc150474402)

[Ilustración 3Archivo XML Estructura Y Diseño De La Interfaz De La App De 4 Figuras Geométricas. 6](#_Toc150474403)

[Ilustración 4 Archivo Activity.java 7](#_Toc150474404)

[Ilustración 5Aplicación CUBO 9](#_Toc150474405)

[Ilustración 6Estructura y diseño del cubo 11](#_Toc150474406)

[Ilustración 7Triangulo\_Equilatero 13](#_Toc150474407)

[Ilustración 8DISEÑO TRIANGULO\_EQUILATERO 15](#_Toc150474408)

[Ilustración 9Figura Hexágono 17](#_Toc150474409)

[Ilustración 10DISEÑO Hexágono 19](#_Toc150474410)

[Ilustración 11Figura Hexágono 21](#_Toc150474411)

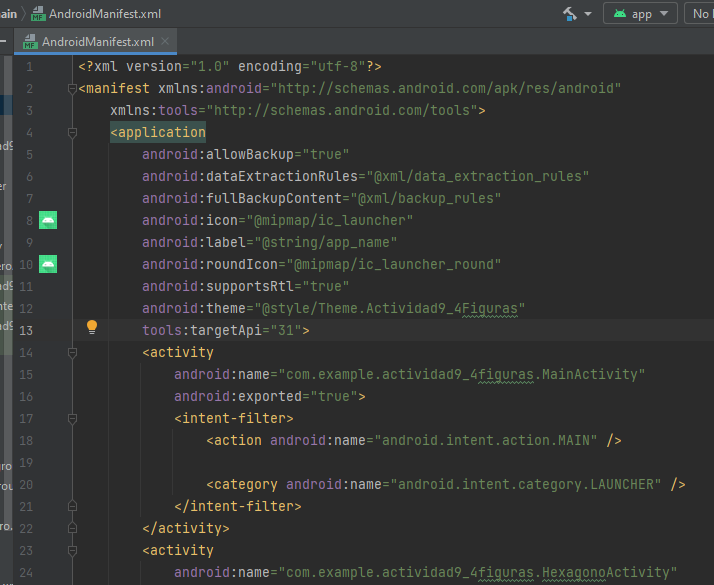
[Ilustración 12DISEÑO Pentágono 23](#_Toc150474412)

## **ARCHIVO MANISFEST.**

Ilustración ARCHIVO MANISFEST.

Este archivo de manifiesto describe la configuración y los componentes de tu aplicación Android, incluyendo las actividades que forman parte de la aplicación y las propiedades generales de la aplicación, como su icono y nombre, es decir el archivo manifiesto es fundamental para que el sistema Android comprenda cómo interactuar con tu aplicación y para garantizar su funcionamiento adecuado.

* Además, este archivo es donde mandamos a llamar los activitys que son principalmente anclas que nos ayudaran a que al momento que el usuario al darle clic en la imagen de la figura lo dirija a otro activitys.
* Como lo son los <activity> que su principal funcionamiento es indicar que se está declarando una actividad en el manifiesto de la aplicación y que las actividades son componentes fundamentales en una aplicación Android y representan pantallas o ventanas con las que los usuarios pueden interactuar.
* Android: exported="true", este principal atributo indica que la actividad puede ser iniciada por otros componentes o aplicaciones, es decir cuando se establece en "true", significa que otras aplicaciones o componentes pueden abrir esta actividad.

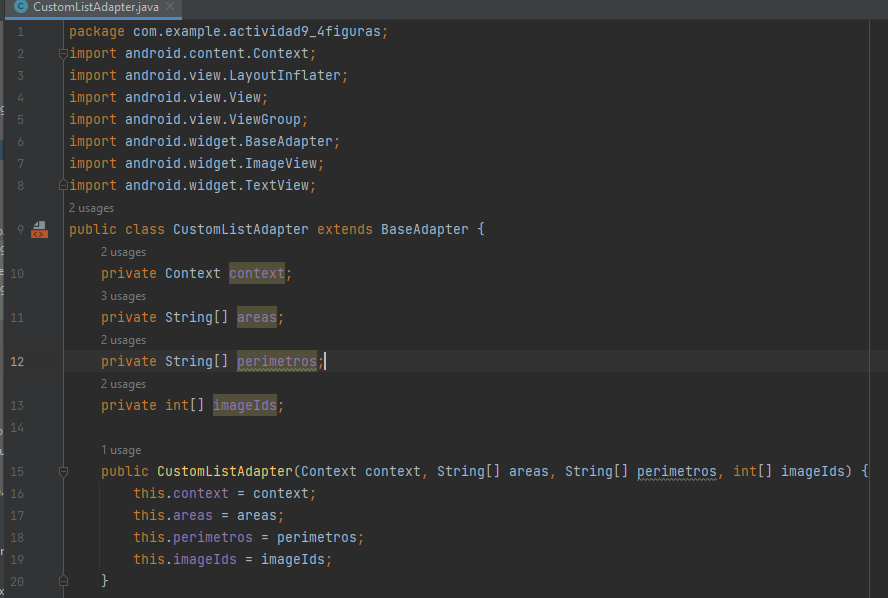


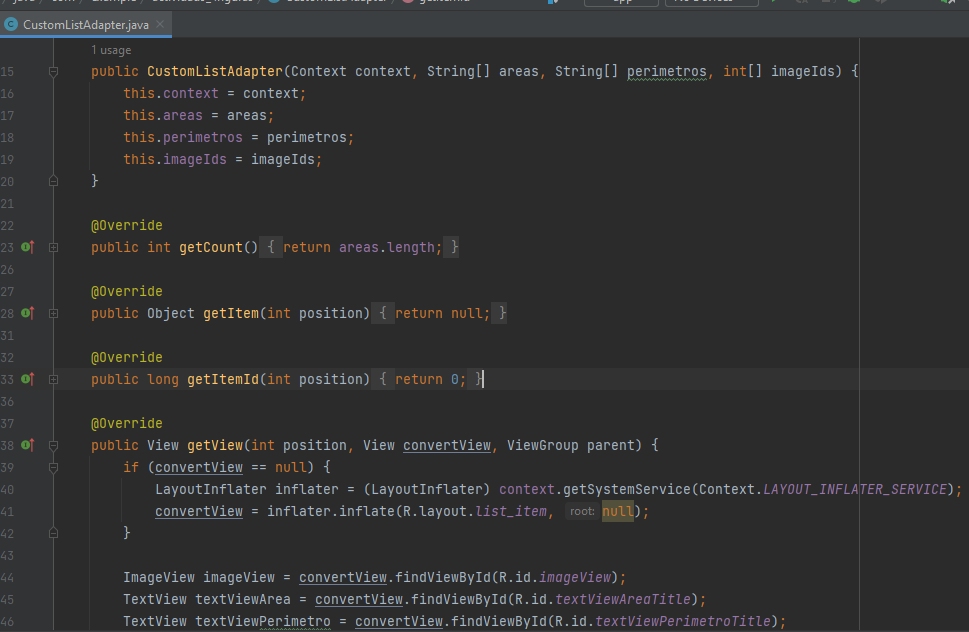
## **ListView\_Personalizado**

Ilustración ListView\_Personalizado

La clase CustomListAdapter se utiliza para personalizar la forma en que se muestran los elementos en una lista en una aplicación de Android, es decir Toma los datos de las áreas, perímetros e imágenes proporcionadas en los arrays y los muestra en una vista personalizada definida en el archivo de diseño list\_item.xml, luego, esta clase se utilizaría en una actividad de Android para configurar un componente de lista, como un ListView, con estos elementos personalizados.

* Como podemos observas tenemos principalmente que pertenece y describe el nombre de la clase.
* Después tenemos cada una de las importaciones de las clases necesarias para tu que nuestra app funcione, es decir Importamos las clases relacionadas con el manejo de vistas y adaptadores en Android.
* Definición de la clase CustomListAdapter, que extiende BaseAdapter, es decir CustomListAdapter se utilizará como un adaptador personalizado para llenar vistas dentro de un Adapter View, como un ListView.
* Posteriormente Declaramos las variables de instancia que nos servirán para almacenar el contexto de la aplicación, dentro de arrays de cadenas de áreas y perímetros, y un array de identificadores de imágenes.
* Declaración del constructor de la clase CustomListAdapter, que acepta y almacena, las arrays de las áreas, perímetros y los identificadores de imágenes como parámetros

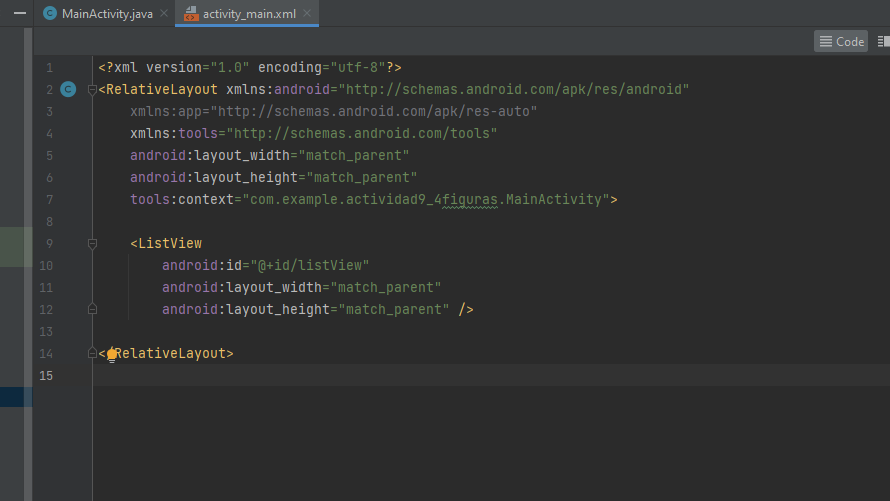


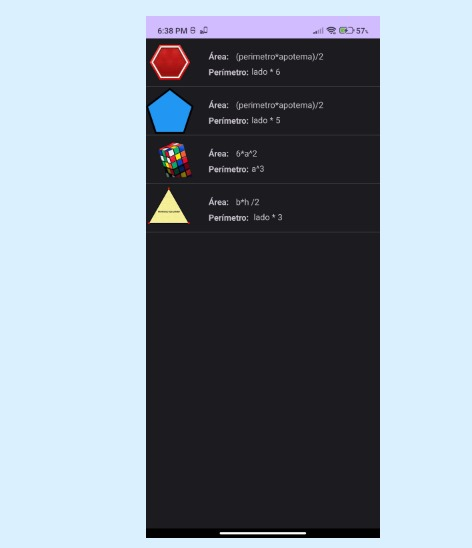


* Después Inicialización de las variables de instancia con los valores proporcionados en el constructor, que contiene las siguientes funciones.
* this. context = context;
* this.areas = areas;
* this.perimetros = perimetros;
* this.imageIds = imageIds;
* Dentro de este es el método getCount (), es importante implementar un adaptador personalizado, que debe devolver el número de elementos que se mostrarán en la vista, como por ejemplo en este caso, simplemente devuelve la longitud del array areas, que determina cuántos elementos se mostrarán.
* En este es el método getItemId (int position), devuelve un ID único para un elemento en una posición específica, es decir de igual manera que el método anterior, siempre devuelve 0, lo que implica que no se está utilizando para proporcionar identificadores son únicos.
* Este es el método más importante, getView (int position, View convertView, ViewGroup parent), dentro de este se crea y configura la vista que representa un elemento en la posición especificada en la lista dentro del bloque condicional if (convertView == null) se utiliza para comprobar si la vista convertView, se crea una nueva vista a partir del diseño especificado en R.layout.list\_item, esta vista se almacena en convertView.

## **Archivo XML Estructura Y Diseño De La Interfaz De La App De 4 Figuras Geométricas.**

Ilustración Archivo XML Estructura Y Diseño De La Interfaz De La App De 4 Figuras Geométricas.

En esta estructura de diseño se mandan a traer las imágenes, que son principalmente las que se muestran al inicio de la interfaz, es decir este archivo XML define la estructura de la interfaz de usuario de la actividad principal, dentro de la interfaz el ListView, muestra las figuras geométricas y sus fórmulas de áreas y perímetros, y que permitirá a los usuarios seleccionar una figura.

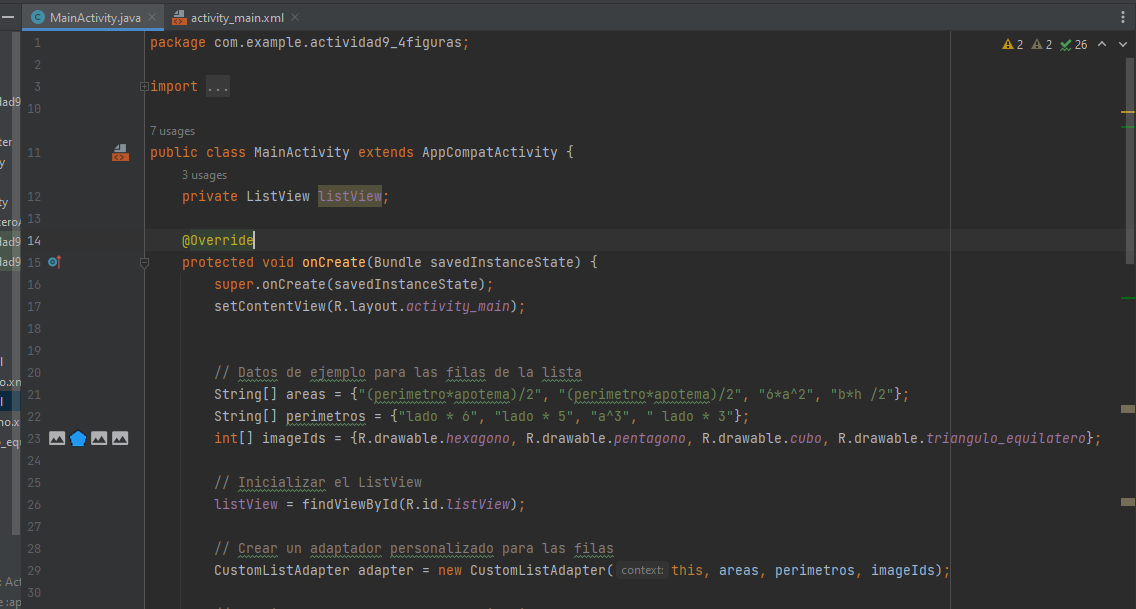


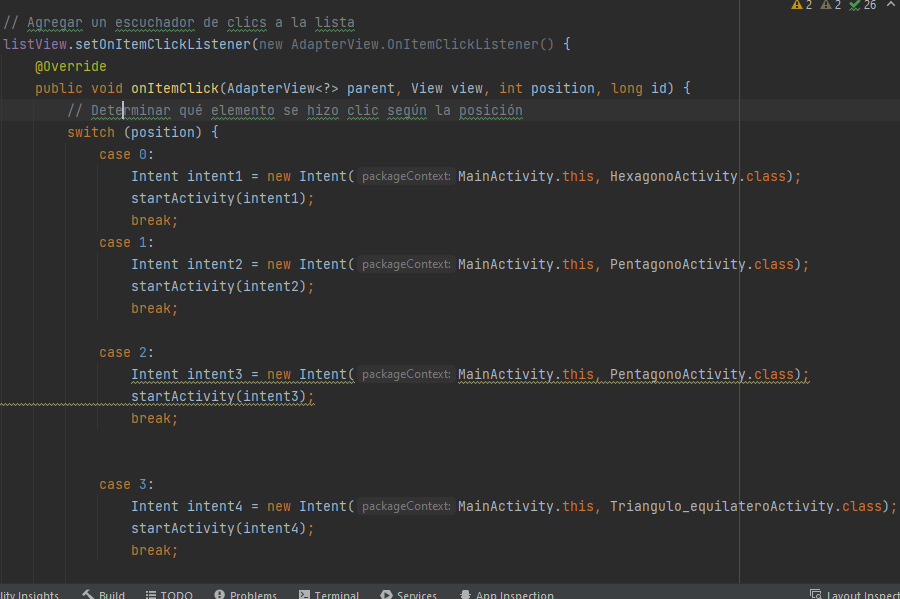
## Archivo Activity.java

Ilustración Archivo Activity.java

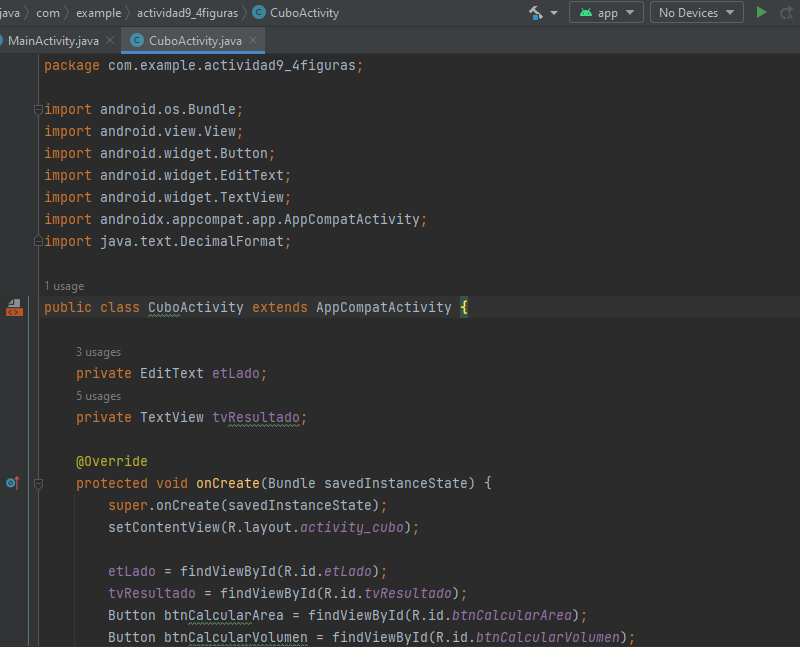
Dentro de este archivo MainActivity.java crea una lista de figuras geométricas con fórmulas de áreas y perímetros, junto con imágenes representativas, es decir su función es que el usuario al dar click en la imagen te dirige a otro activity correspondiente ala figura esa figura geométrica para calcular y mostrar sus propiedades, la actividad principal (MainActivity) es responsable de configurar y mostrar la lista, así como de manejar las acciones de clic en los elementos de la lista.

* Dentro del onCreate creamos y Definimos un array de cadenas (áreas, perímetros) que contienen las fórmulas de áreas y perímetros correspondientes a figuras geométricas, también creamos un array de enteros que lo llamos (imageIds) que almacena identificadores de recursos de imágenes correspondientes a las figuras.
* Se inicia el ListView: es decir a qui se obtiene el ListView definido en el diseño (R.id. listView) utilizando findViewById, esta referencia se almacena en la variable listView. El ListView es donde mostrarás la lista de figuras geométricas.
* Adapter: es un adaptador personalizado llamado CustomListAdapter, este adaptador se encarga de tomar los datos de ejemplo (fórmulas de áreas, fórmulas de perímetros y recursos de imágenes) y llenar las filas de la lista con estos datos, es decir el adaptador es necesario para mostrar los datos en el ListView de una manera personalizada.
* (OnItemClickListener) a un ListView llamado listView, es decir este escuchador se encarga de manejar las interacciones de los usuarios con los elementos de la lista y responder iniciando actividades específicas según el elemento seleccionado.
* Cuando un usuario hace clic en un elemento de la lista, el ListView detecta la acción del clic y notifica al escuchador de clics que se ha producido una acción.
* El escuchador de clics OnItemClickListener anónimo se crea e implementa en el lugar, y se asocia con el ListView usando listView.setOnItemClickListener(...). Este escuchador tiene un método onItemClick que se ejecutará cuando se haga clic en un elemento de la lista.
* El método onItemClick recibe cuatro parámetros, parent: Es el Adapter View que contiene los elementos de la lista (en este caso, el ListView).
* view: Es la vista del elemento de la lista en la que se hizo clic.
* position: Es la posición del elemento en la lista que se hizo clic (comienza desde 0).
* id: Es el ID del elemento en la lista (en este caso, siempre se establece en 0).
* Dentro del método onItemClick, se utiliza una declaración switch basada en la variable position para determinar qué elemento de la lista se hizo clic. La variable position se utiliza para identificar el índice del elemento en la lista.
* Finalmente, se utiliza el método startActivity para iniciar la actividad correspondiente a la que se hizo clic.



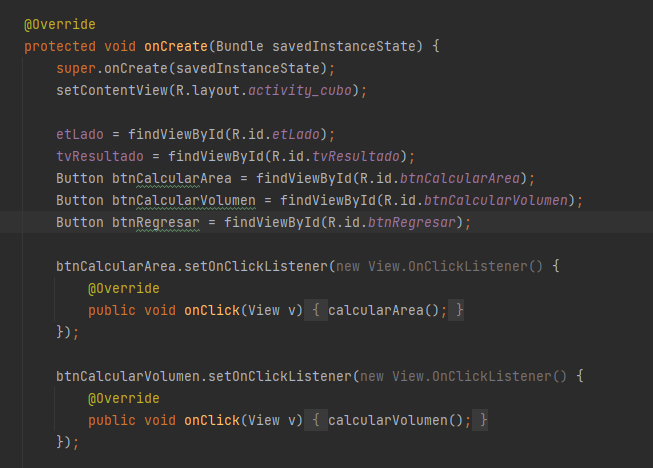


## **Aplicación CUBO**

Ilustración Aplicación CUBO

Dentro de archivo Cubo.Activity, importamos las distintas clases de Android y Java creadas para que nuestra app funcione, es decir incluyendo cada uno de los elementos de interfaz de usuario como Editex, TextView, Button, y AppCompatActivity, así como DecimalFormat para formatear resultados numéricos.

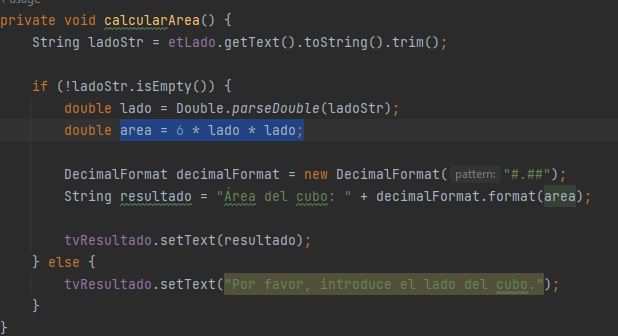
* Después creamos la clase Cubo.Activity
* Declaración de variables de instancia para Editex y TextView, que se utilizarán para interactuar con la interfaz de usuario.

Posteriormente creamos el método onCreate (), que realiza la inicialización de la actividad cuando se crea, Se establece el contenido de la actividad a partir del archivo de diseño activity\_cubo.xml, y luego, se obtienen los elementos de interfaz de usuario (Editex, TextView y tres botones) mediante sus identificadores, como se muestran en a siguiente imagen.

* Y configuramos un listener de clic para el botón "Calcular Área" Cuando se hace clic en este botón, se llama al método calcular Área (), y para configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el are del Cubo con la siguiente formula Area= 6 Lado \*Lado, es la suma o multiplicación de sus 6 lado por lado del cubo.

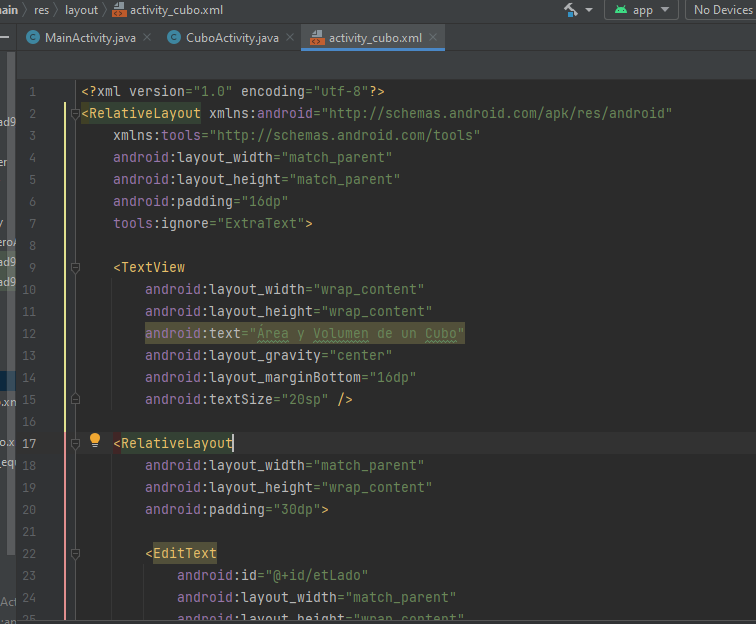


* Realizamos la configuramos un listener de clic para el botón "Calcular Volumen" Cuando se hace clic en este botón, se llama al método calcular Volumen (), y para configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el are del Cubo con la siguiente formula volumen = lado \* lado \* lado; y el resultado es el volumen del cubo.



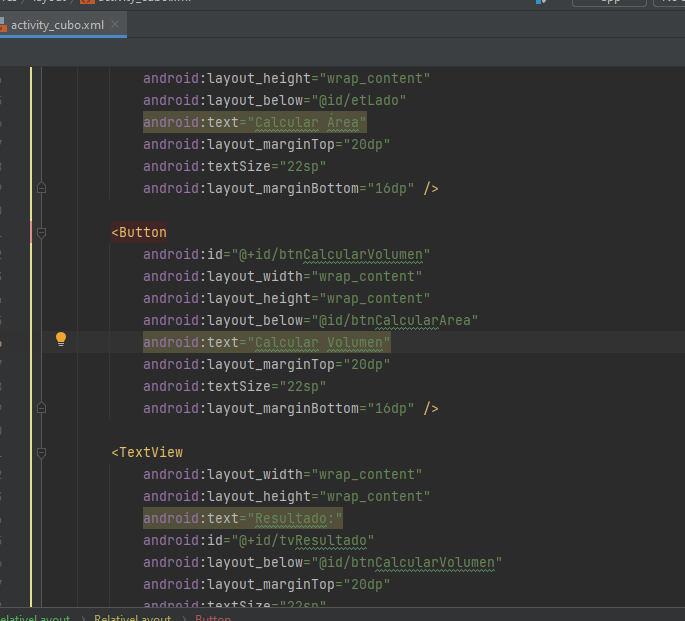
## **Estructura y diseño del cubo**

Ilustración Estructura y diseño del cubo



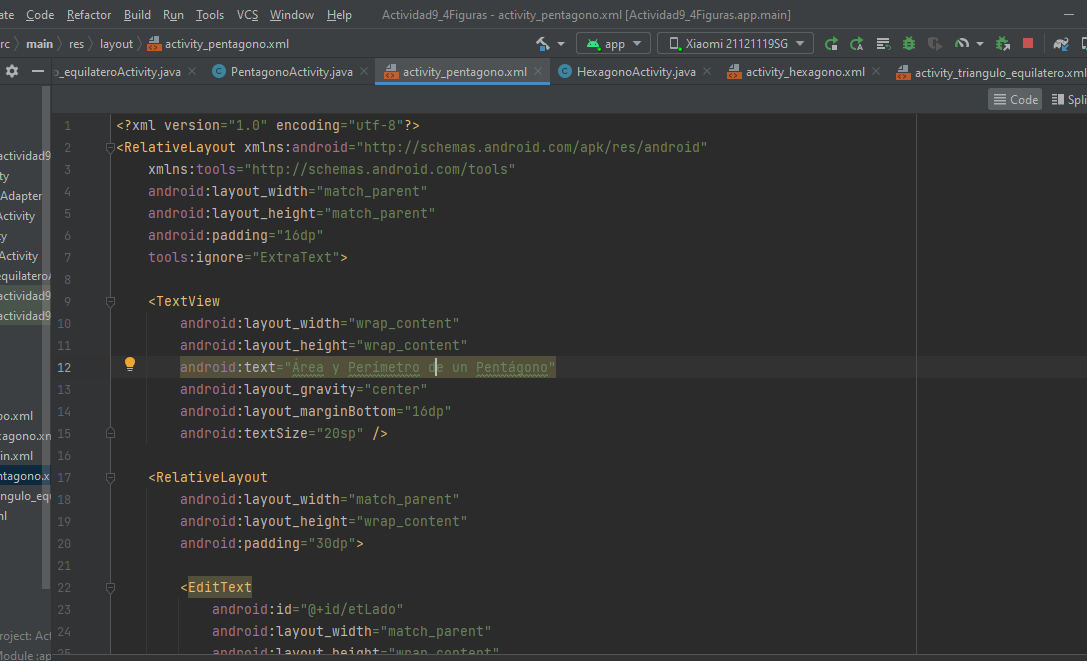
## **Aplicación Pentágono.**

* Dentro de este archivo activity.cubo.xml tenemos al contenedor principal, que es un RelativeLayout, que permite organizar los elementos de la interfaz de usuario de forma relativa. Se establece un relleno de 16dp en todos los lados y se ignora un aviso específico de las herramientas de diseño.
* Después definimos un TextView sin contenido inicial (android:text=""), es decir se establece su anchura y altura para envolver el contenido (wrap\_content), se posiciona en el centro (layout\_gravity="center") y se le asigna un margen en la parte inferior de 16dp.
* Creamos otro RelativeLayout anidado dentro del contenedor principal con un relleno de 30dp, esto se utiliza para agrupar los elementos relacionados con la entrada del usuario y los botones de acción.
* Agregamos un Editex con un identificador único (@+id/etLado) para hacer
* referencia dentro del código Java se utiliza para que el usuario ingrese el valor del lado del cubo, se establece un tipo de entrada de número decimal y se asigna un tamaño de texto.
* Agregamos un Button para calcular el área del cubo, y se le asigna un identificador (@+id/btnCalcularArea) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un Button para calcular el volumen del cubo, y se le asigna un identificador (@+id/btnCalcularArea) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un botón (btnRegresar) para regresar a la actividad anterior se le asigna un identificador, se posiciona debajo del TextView de resultado y se centra en el padre y también se le asigna un color de fondo.



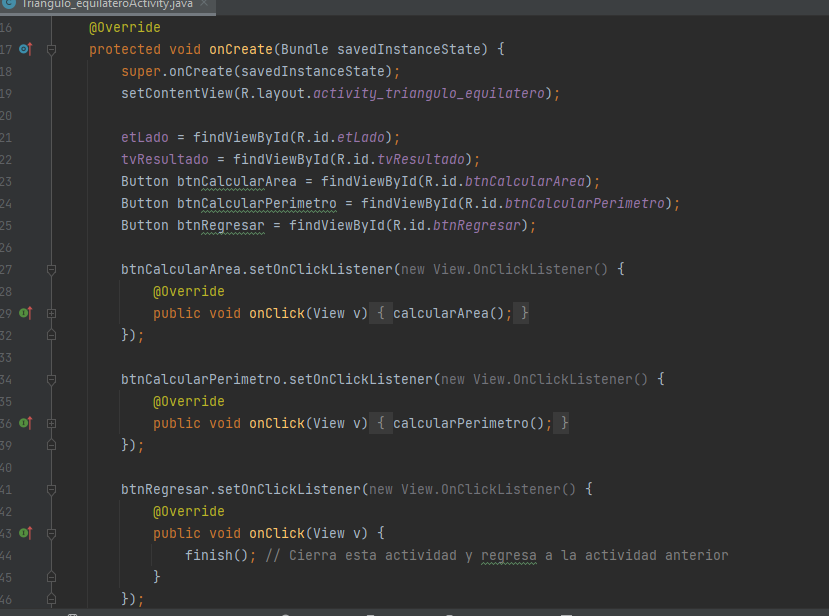
## **Triangulo\_Equilatero**

Ilustración Triangulo\_Equilatero

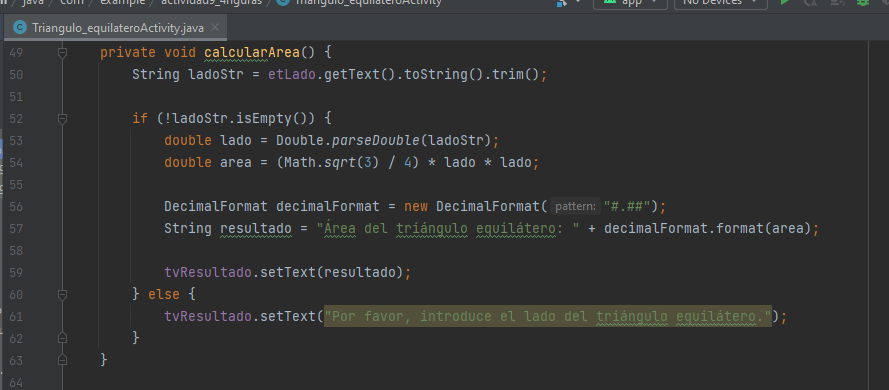


Dentro de archivo Triangulo\_equilatero. Activity, importamos las distintas clases de Android y Java creadas para que nuestra app funcione, es decir incluyendo cada uno de los elementos de interfaz de usuario como Editex, TextView, Button, y AppCompatActivity, así como DecimalFormat para formatear resultados numéricos.

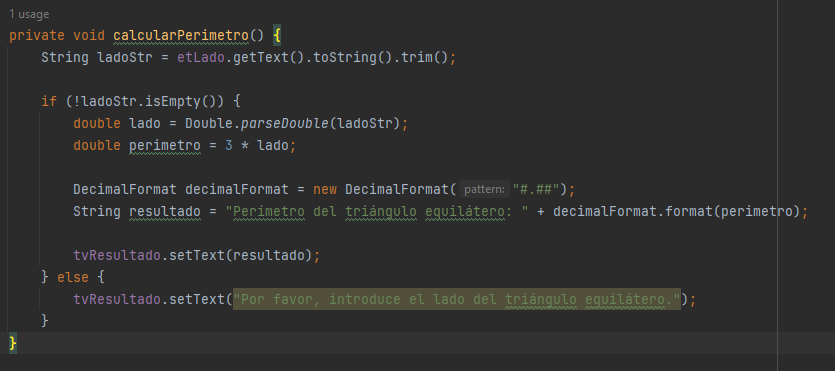
* Después creamos la claseTrianguloEquilatero.Activity
* Declaración de variables de instancia para Editex y TextView, que se utilizarán para interactuar con la interfaz de usuario.

Posteriormente creamos el método onCreate (), que realiza la inicialización de la actividad cuando se crea, Se establece el contenido de la actividad a partir del archivo de diseño activity\_triangulo\_equilatero.xml, y luego, se obtienen los elementos de interfaz de usuario (Editex, TextView y tres botones) mediante sus identificadores, como se muestran en a siguiente imagen.

* Y configuramos un listener de clic para el botón "Calcular Área" Cuando se hace clic en este botón, se llama al método calcular Área (), y para configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el are del Cubo con la siguiente formula area = (Math.sqrt(3) / 4) \* lado \* lado.

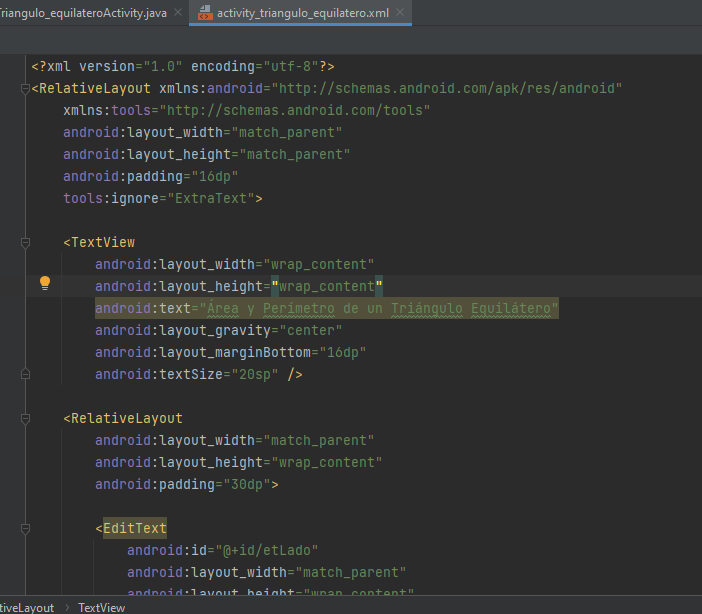


* Realizamos la configuramos un listener de clic para el botón "Calcular perímetro"
* Cuando se hace clic en este botón, se llama al método calcular Volumen (), y para configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el perímetro con la siguiente formula perímetro = 3 \* lado;

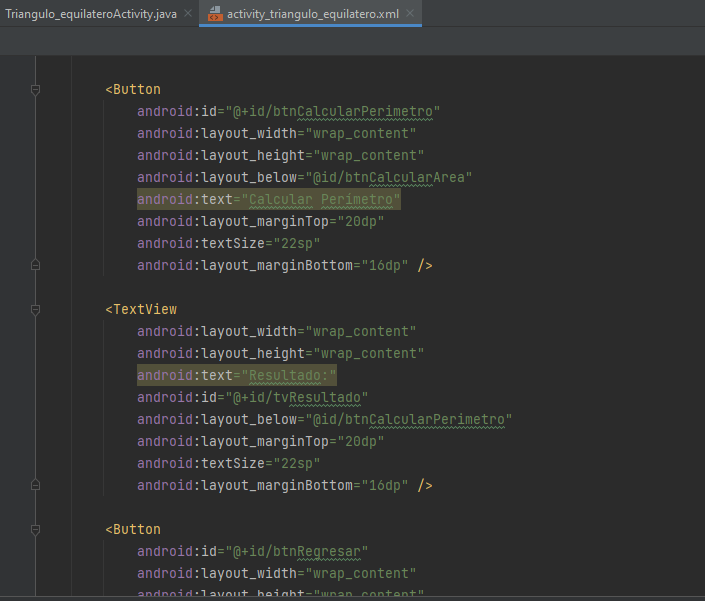


## **DISEÑO TRIANGULO\_EQUILATERO**

Ilustración DISEÑO TRIANGULO\_EQUILATERO

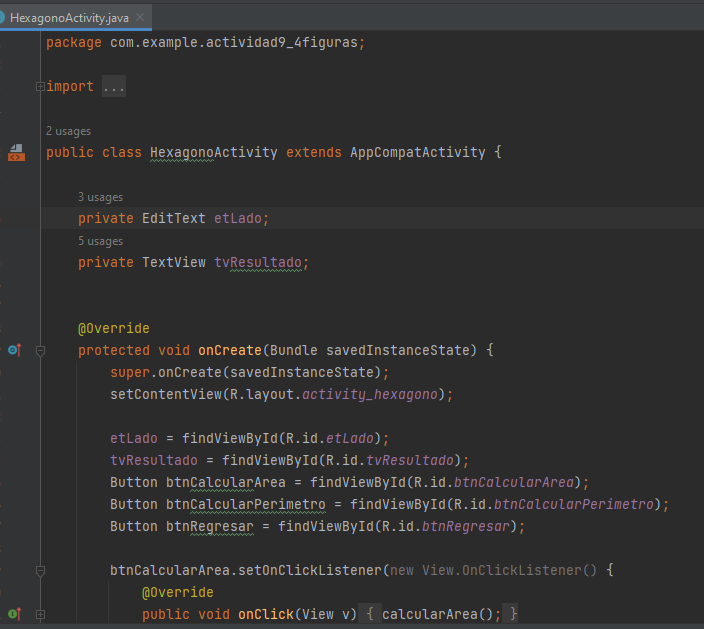


* Dentro de este archivo activity.triangulo\_equilatero.xml tenemos al contenedor principal, que es un RelativeLayout, que permite organizar los elementos de la interfaz de usuario de forma relative, se establece un relleno de 16dp en todos los lados y se ignora un aviso específico de las herramientas de diseño.
* Después definimos un TextView sin contenido inicial (android:text=""), es decir se establece su anchura y altura para envolver el contenido (wrap\_content), se posiciona en el centro (layout\_gravity="center") y se le asigna un margen en la parte inferior de 16dp.
* Creamos otro RelativeLayout anidado dentro del contenedor principal con un relleno de 30dp, esto se utiliza para agrupar los elementos relacionados con la entrada del usuario y los botones de acción.
* Agregamos un Editex con un identificador único (@+id/etLado) para hacer
* referencia dentro del código Java se utiliza para que el usuario ingrese el valor del lado del cubo, se establece un tipo de entrada de número decimal y se asigna un tamaño de texto.
* Agregamos un Button para calcular el área del Triangulo\_equilatero, y se le asigna un identificador (@+id/btnCalcularArea) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un Button para calcular el perímetro del Triangulo\_equilatero, y se le asigna un identificador (@+id/btnCalcularPerimetro) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un botón (btnRegresar) para regresar a la actividad anterior se le asigna un identificador, se posiciona debajo del TextView de resultado y se centra en el padre y también se le asigna un color de fondo.



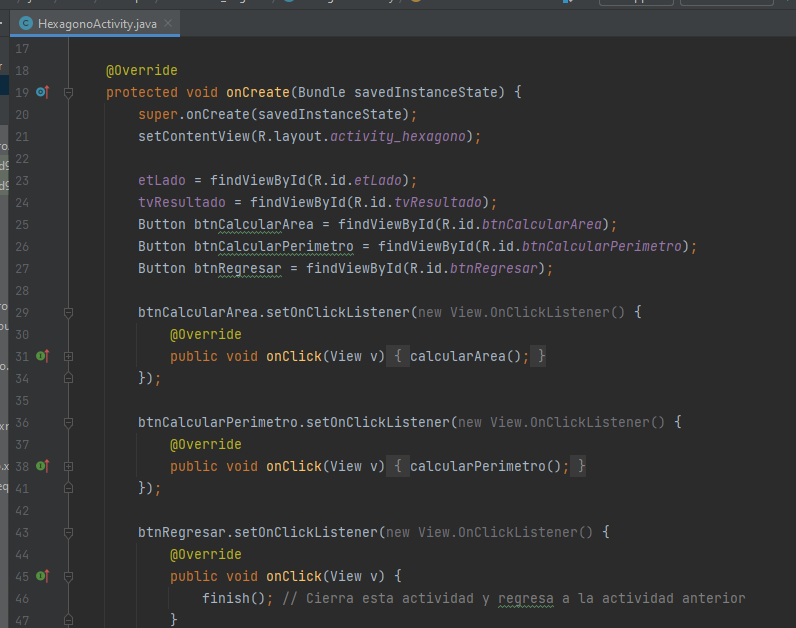
## **Figura Hexágono**

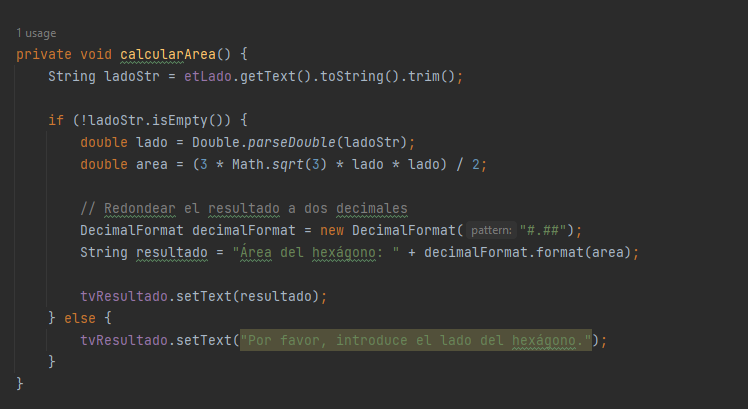
Ilustración Figura Hexágono



Dentro de archivo Hexágono. Activity, importamos las distintas clases de Android y Java creadas para que nuestra app funcione, es decir incluyendo cada uno de los elementos de interfaz de usuario como Editex, TextView, Button, y AppCompatActivity, así como DecimalFormat para formatear resultados numéricos.

* Después creamos la clase Hexágono. Activity
* Declaración de variables de instancia para Editex y TextView, que se utilizarán para interactuar con la interfaz de usuario.

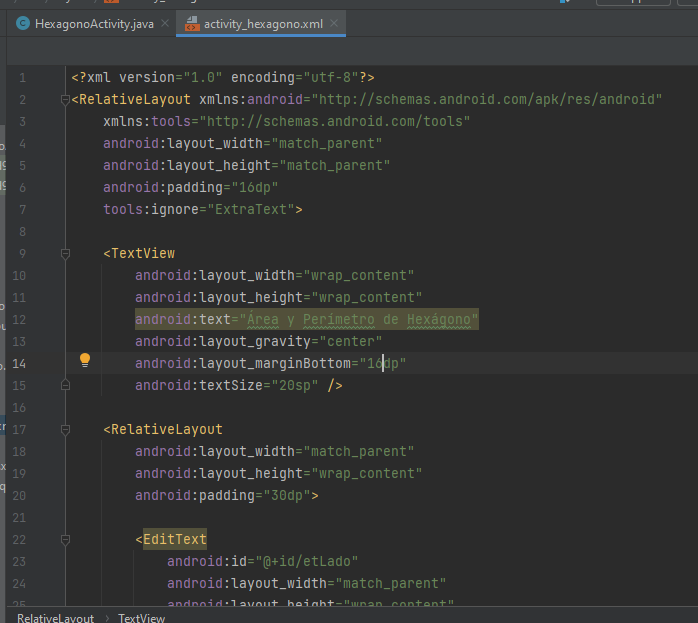
Posteriormente creamos el método onCreate (), que realiza la inicialización de la actividad cuando se crea, Se establece el contenido de la actividad a partir del archivo de diseño activity\_hexágono.xml, y luego, se obtienen los elementos de interfaz de usuario (Editex, TextView y tres botones) mediante sus identificadores, como se muestran en a siguiente imagen.

* Y configuramos un listener de clic para el botón "Calcular Área" Cuando se hace clic en este botón, se llama al método calcular Área (), y para configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el are del Hexágono con la siguiente formula area = area = (3 \* Math.sqrt(3) \* lado \* lado) / 2;
* Realizamos la configuramos un listener de clic para el botón "Calcular perímetro"
* Cuando se hace clic en este botón, se llama al método calcular perímetro (), y para
* configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el perímetro con la siguiente formula perímetro = perímetro = 6 \* lado;

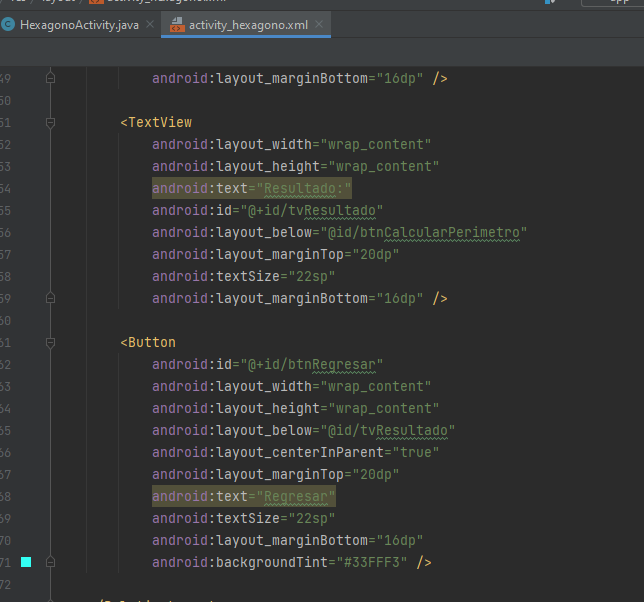


## **DISEÑO Hexágono**

Ilustración DISEÑO Hexágono

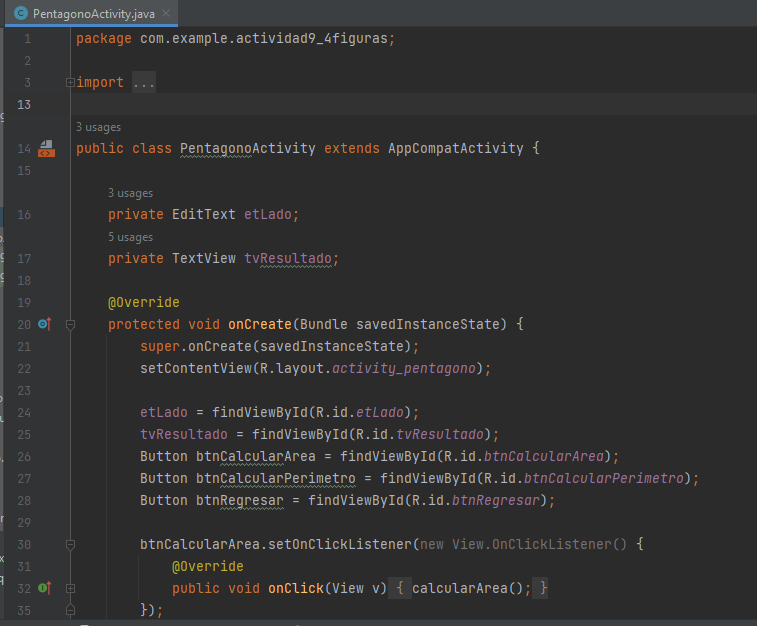


* Dentro de este archivo activity. \_ Hexágono.xml tenemos al contenedor principal, que es un RelativeLayout, que permite organizar los elementos de la interfaz de usuario de forma relative, se establece un relleno de 16dp en todos los lados y se ignora un aviso específico de las herramientas de diseño.
* Después definimos un TextView sin contenido inicial (android:text=""), es decir se establece su anchura y altura para envolver el contenido (wrap\_content), se posiciona en el centro (layout\_gravity="center") y se le asigna un margen en la parte inferior de 16dp.
* Creamos otro RelativeLayout anidado dentro del contenedor principal con un relleno de 30dp, esto se utiliza para agrupar los elementos relacionados con la entrada del usuario y los botones de acción.
* Agregamos un Editex con un identificador único (@+id/etLado) para hacer
* referencia dentro del código Java se utiliza para que el usuario ingrese el valor del lado del cubo, se establece un tipo de entrada de número decimal y se asigna un tamaño de texto.
* Agregamos un Button para calcular el área del Hexágono, y se le asigna un identificador (@+id/btnCalcularArea) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un Button para calcular el perímetro del Hexágono, y se le asigna un identificador (@+id/btnCalcularPerimetro) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un botón (btnRegresar) para regresar a la actividad anterior se le asigna un identificador, se posiciona debajo del TextView de resultado y se centra en el padre y también se le asigna un color de fondo.



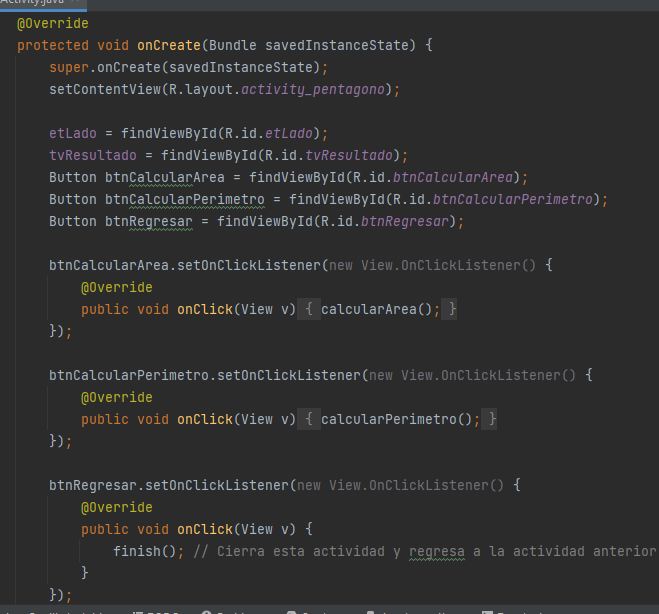
## **Figura Hexágono**

Ilustración Figura Hexágono

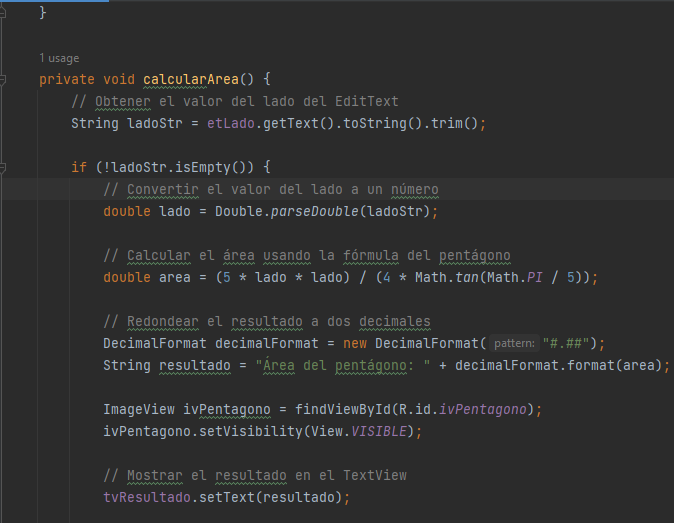


Dentro de archivo Pentágono. Activity, importamos las distintas clases de Android y Java creadas para que nuestra app funcione, es decir incluyendo cada uno de los elementos de interfaz de usuario como Editex, TextView, Button, y AppCompatActivity, así como DecimalFormat para formatear resultados numéricos.

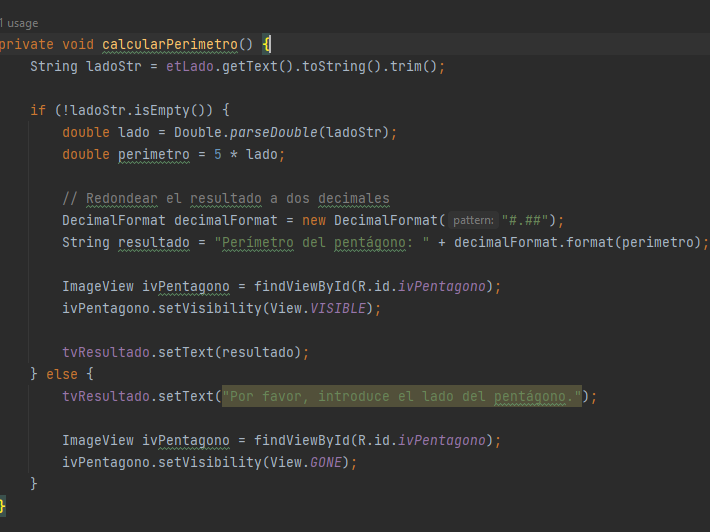
* Después creamos la clase Pentágono. Activity
* Declaración de variables de instancia para Editex y TextView, que se utilizarán para interactuar con la interfaz de usuario.

Posteriormente creamos el método onCreate (), que realiza la inicialización de la actividad cuando se crea, Se establece el contenido de la actividad a partir del archivo de diseño activity\_ pentágono.xml, y luego, se obtienen los elementos de interfaz de usuario (Editex, TextView y tres botones) mediante sus identificadores, como se muestran en a siguiente imagen.

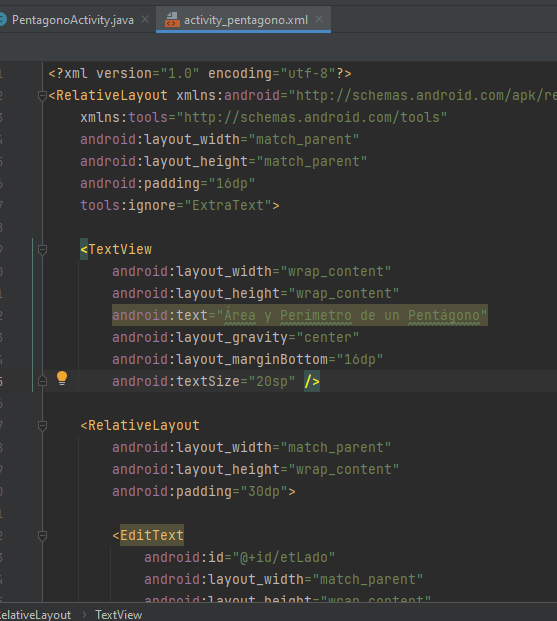
* Y configuramos un listener de clic para el botón "Calcular Área" Cuando se hace clic en este botón, se llama al método calcular Área (), y para configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el are del pentágono con la siguiente formula area = (5 \* lado \* lado) / (4 \* Math.tan (Math.PI / 5));
* Realizamos la configuramos un listener de clic para el botón "Calcular perímetro"
* Cuando se hace clic en este botón, se llama al método calcular perímetro (), y para



* configuraciones la función que nos ayudara a tener el resultado que necesitamos para poder calcular el perímetro con la siguiente formula perímetro = 5 \* lado;



## **DISEÑO Pentágono**

Ilustración DISEÑO Pentágono

* Dentro de este archivo activity. \_pentagono.xml tenemos al contenedor principal, que es un RelativeLayout, que permite organizar los elementos de la interfaz de usuario de forma relative, se establece un relleno de 16dp en todos los lados y se ignora un aviso específico de las herramientas de diseño.
* Después definimos un TextView sin contenido inicial (android:text=""), es decir se establece su anchura y altura para envolver el contenido (wrap\_content), se posiciona en el centro (layout\_gravity="center") y se le asigna un margen en la parte inferior de 16dp.
* Creamos otro RelativeLayout anidado dentro del contenedor principal con un relleno de 30dp, esto se utiliza para agrupar los elementos relacionados con la entrada del usuario y los botones de acción.
* Agregamos un Editex con un identificador único (@+id/etLado) para hacer
* referencia dentro del código Java se utiliza para que el usuario ingrese el valor del lado del cubo, se establece un tipo de entrada de número decimal y se asigna un tamaño de texto.
* Agregamos un Button para calcular el área del pentágono, y se le asigna un identificador (@+id/btnCalcularArea) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un Button para calcular el perímetro del pentágono, y se le asigna un identificador (@+id/btnCalcularPerimetro) y se posiciona debajo del EditText. También se le asigna un tamaño de texto y márgenes superior e inferior.
* Agregamos un botón (btnRegresar) para regresar a la actividad anterior se le asigna un identificador, se posiciona debajo del TextView de resultado y se centra en el padre y también se le asigna un color de fondo.

