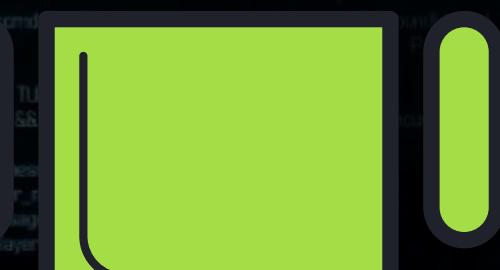


# AWAKELAB

## #programmingbootcamp

# ELEMENTS & LISTENER





## ¿QUÉ VAMOS A VER?

- Box Model.
- Unidades de medida.
- ¿ListView, ScrollView o RecyclerView?
- Librerías.
- CardView.
- RecyclerView.
- Toolbar.
- OnClick en RecyclerView.



# ¡VAMOS A COMENZAR!

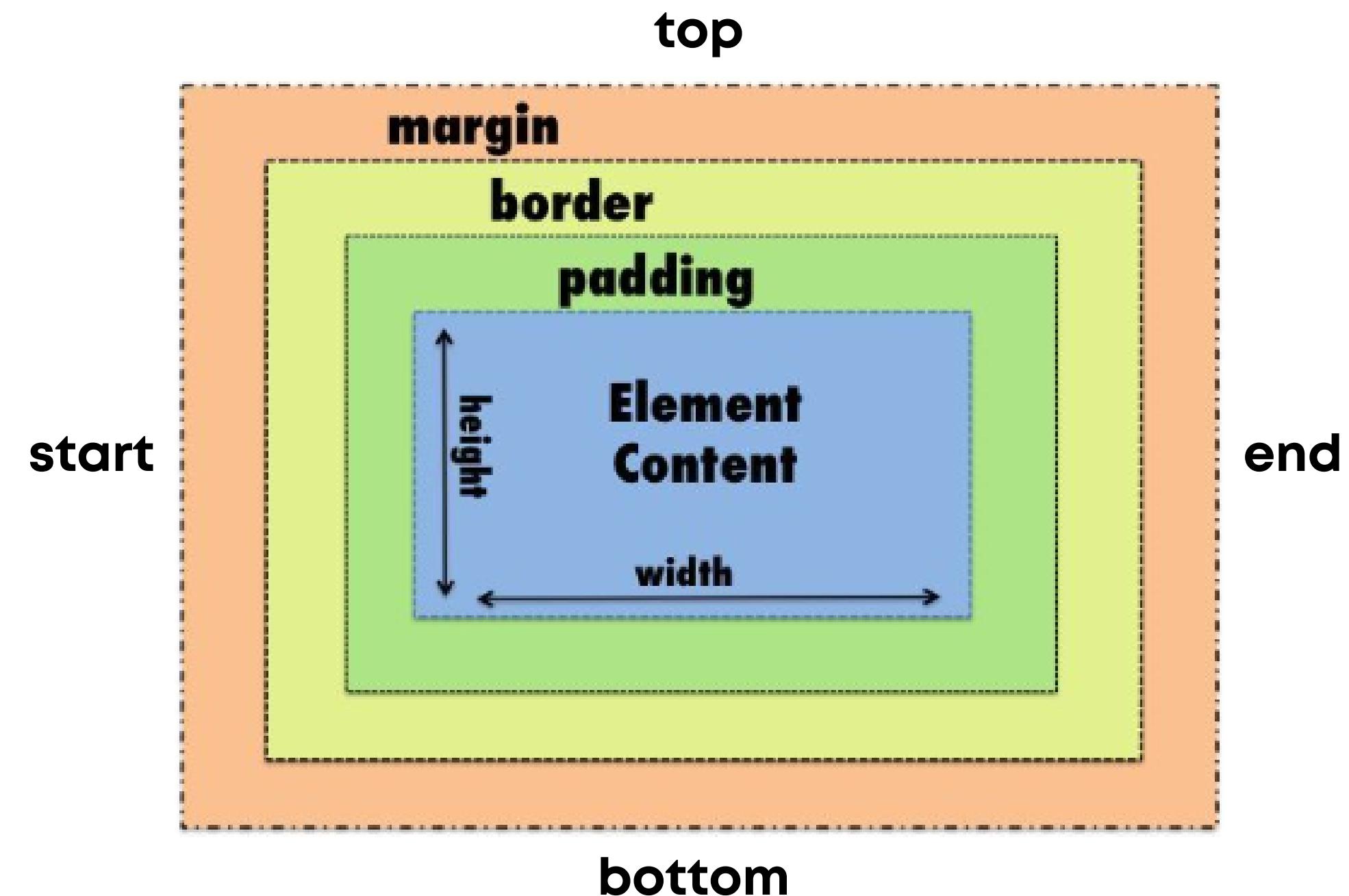




# BOX MODEL



# BOX MODEL





# UNIDADES DE MEDIDA



# UNIDADES DE MEDIDA

<b>dp</b>	Píxeles independientes de la densidad.	Unidad abstracta que se basa en la densidad física de la pantalla.
<b>sp</b>	Píxeles independientes de la escala.	Similar a <b>dp</b> , pero también se ajusta según las preferencias de tamaño que tenga el usuario en su dispositivo. <b>Recomendado para el tamaño de las fuentes.</b>
<b>px</b>	Píxeles.	Corresponde a los píxeles reales en la pantalla. <b>No es una opción recomendada.</b>
<b>mm</b>	Milímetros.	Según el tamaño físico de la pantalla.
<b>in</b>	Pulgadas.	Según el tamaño físico de la pantalla.



# ¿LISTVIEW, SCROLLVIEW O RECYCLERVIEW?



# LISTVIEW

- Componentes de Android que se utiliza para mostrar una lista con elementos desplazables.
- Es fácil de implementar.
- No es muy efectivo para listas extensas, ya que puede sobrecargar la aplicación.
- Puntualmente es muy útil cuando se utilizan listas pequeñas con una cantidad limitada de datos que podamos manejar sin ningún problema. Mejor aún si muchos de esos datos son estáticos.





# SCROLLVIEW

- Tal y como lo dice su nombre, es una agrupación que nos permite desplazar las vistas que tengamos dentro del componente.
- Su uso ideal está en el desplazamiento de layouts complejos y que no sean repetitivos.
- No está optimizado para contener listas extensas de elementos homogéneos.
- Se puede configurar un desplazamiento horizontal o vertical.





# RECYCLERVIEW

- Es un componente avanzado de Android que se utiliza para mostrar grandes conjuntos de datos o vistas que se requieran cargar a medida que las necesitamos.
- Recicla las vistas que ya no visualizamos, por ende, mejora el rendimiento de nuestra aplicación y reduce el consumo de recursos.
- Tiene más flexibilidad a la hora de personalizar sus elementos.
- Ideal para listas y elementos que cambian en tiempo de ejecución.





# LIBRERÍAS



# LIBRERÍAS

- Son un conjunto de código reutilizable que podemos incluir en nuestros proyectos para añadir funcionalidades que necesitamos y ya existen.
- Nos permiten reutilizar código, ahorrando tiempo y esfuerzo. No es necesario inventar la rueda otra vez.
- Vamos a sincronizar dos librerías que utilizaremos a lo largo de los proyectos en nuestras clases.
- **app > Gradle Scripts > libs.versions.toml**





# LIBRERÍAS

- **[versions]**
- cardview = “1.0.0”
- recyclerview = “1.3.2”
  
- **[libraries]**
- cardview = { module = “*androidx.cardview:cardview*”, version.ref = “*cardview*” }
- recyclerview = { module = “*androidx.recyclerview:recyclerview*”, version.ref = “*recyclerview*” }
  
- **¡¡Esto no es todo!!**
- Parte del elemento ya está cargado en la librería del proyecto, pero antes de poder sincronizar, debemos implementar los objetos en las dependencias.



# LIBRERÍAS

- **app > Gradle Scripts > build.gradle (Module :app)**
- Una vez estemos dentro del archivo, debemos reconocer el apartado de “**dependencies**” que suele estar al final del todo.
- Utilizaremos la palabra reservada “**implementation**”, al igual que “**libs**”.
- **implementation libs.cardview**
- **implementation libs.recyclerview**
- Una vez implementemos correctamente nuestros componentes, seleccionamos “**Sync Now**” en la esquina superior derecha. (top | end)



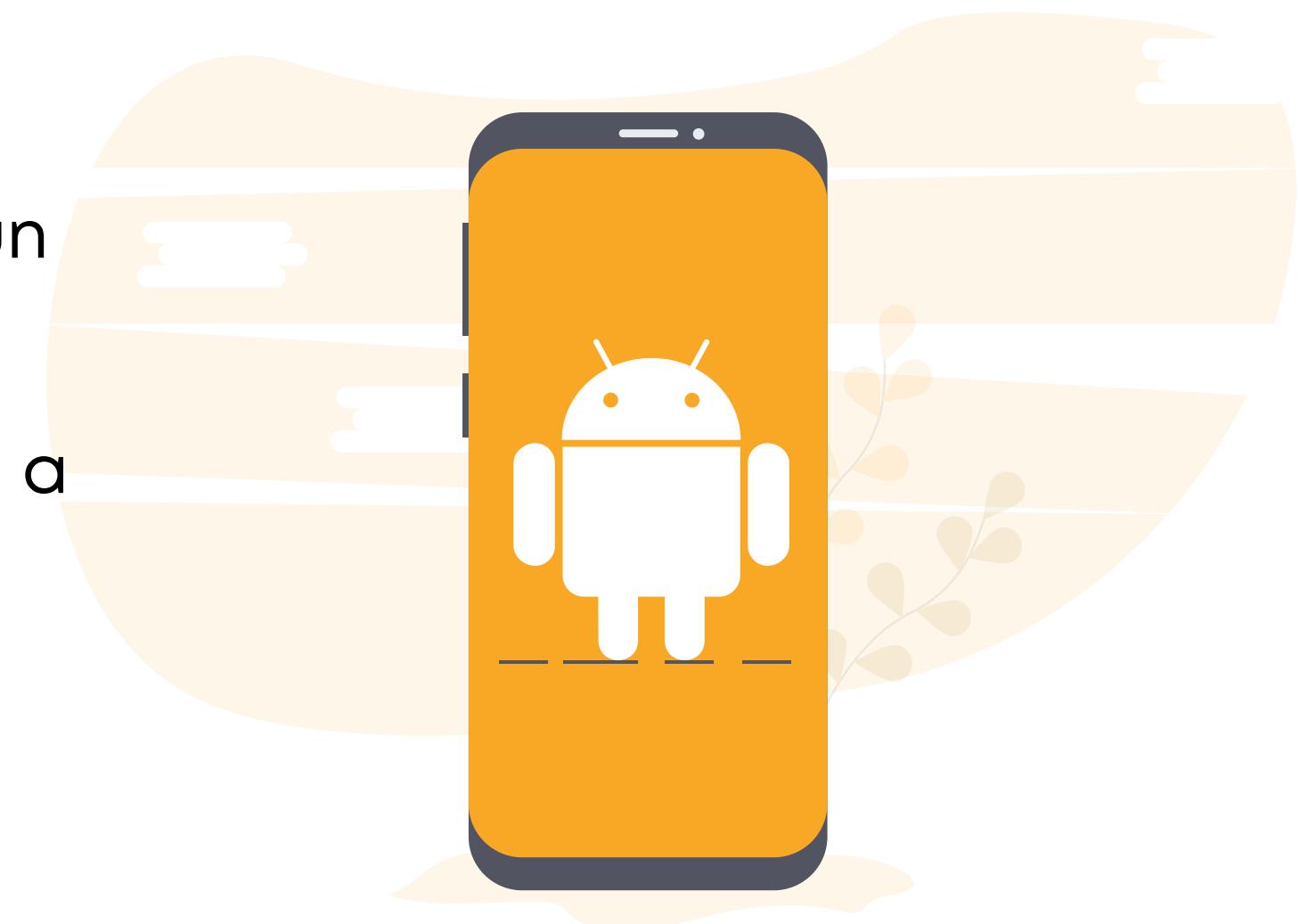


# CARDVIEW



## LISTA DE ELEMENTOS .XML

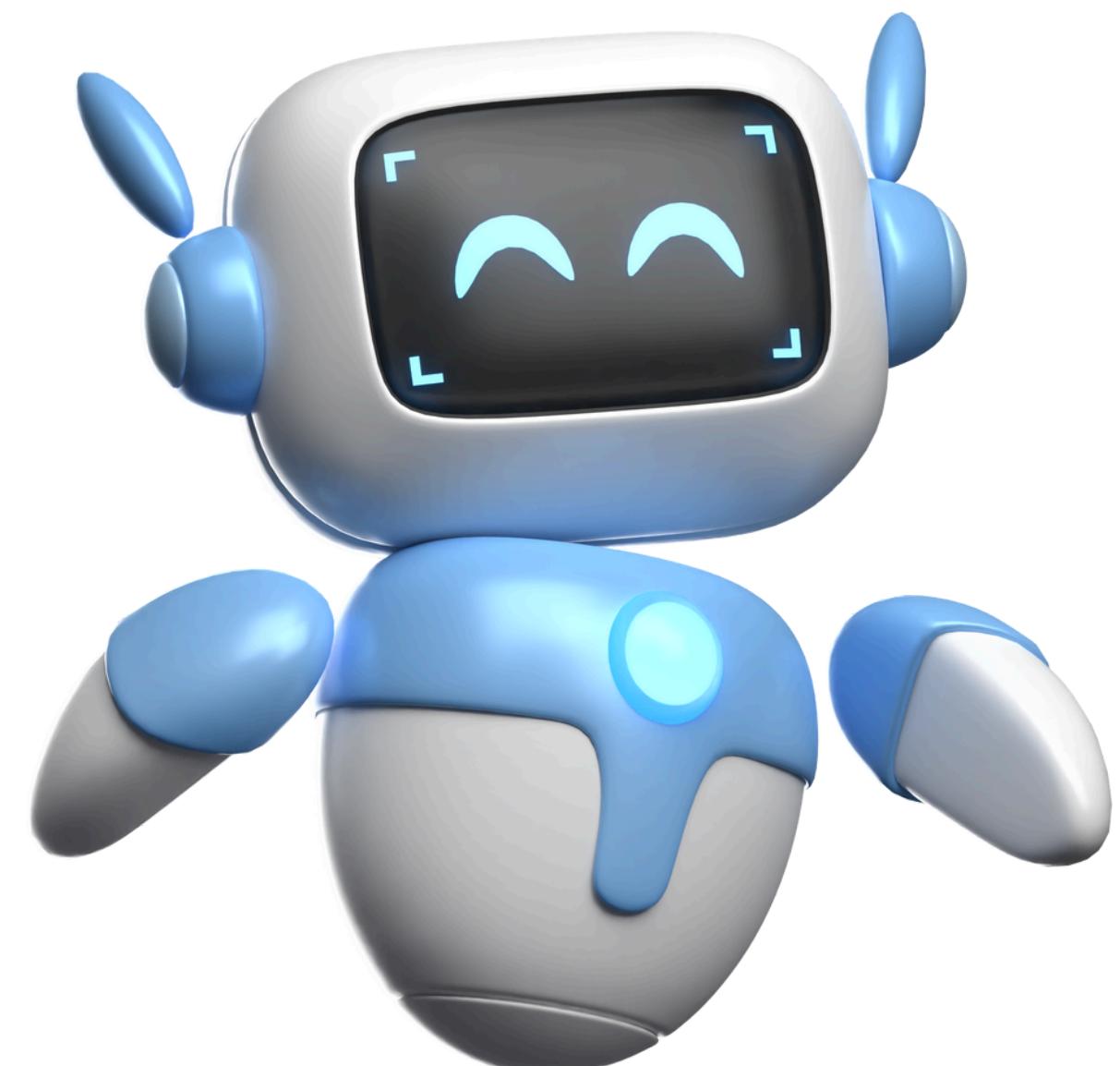
- Quizás **RecyclerView** es una elección más compleja, pero a la larga su funcionalidad es mucho mejor y más estable.
- Lo primero que haremos será crear un **archivo.xml** dentro de la carpeta **layout**.
- Este archivo va a contener el diseño de la **CardView**, evitando así que debamos agregar un elemento cada vez que la queramos utilizar.
- Es importante tener en cuenta que **CardView** va a ser un contenedor que nos permite utilizar diferentes **Layouts** dentro de su proceso de creación.





# PARÁMETROS

- **android:animatedLayoutChanges:**
  - Al marcar su estatus como “true”, estamos habilitando animaciones automáticas que se visualizarán cuando existan cambios en el diseño.
- **app:cardCornerRadius:**
  - Viene a definir el radio de las esquinas del CardView, permitiéndonos tener diseños más estilizados.
- **app:cardUseCompatPadding="true"**
  - Al establecerlo en true, este atributo agregará relleno adicional que ayudará a mantener una homogeneidad del trabajo antiguo-nuevo.



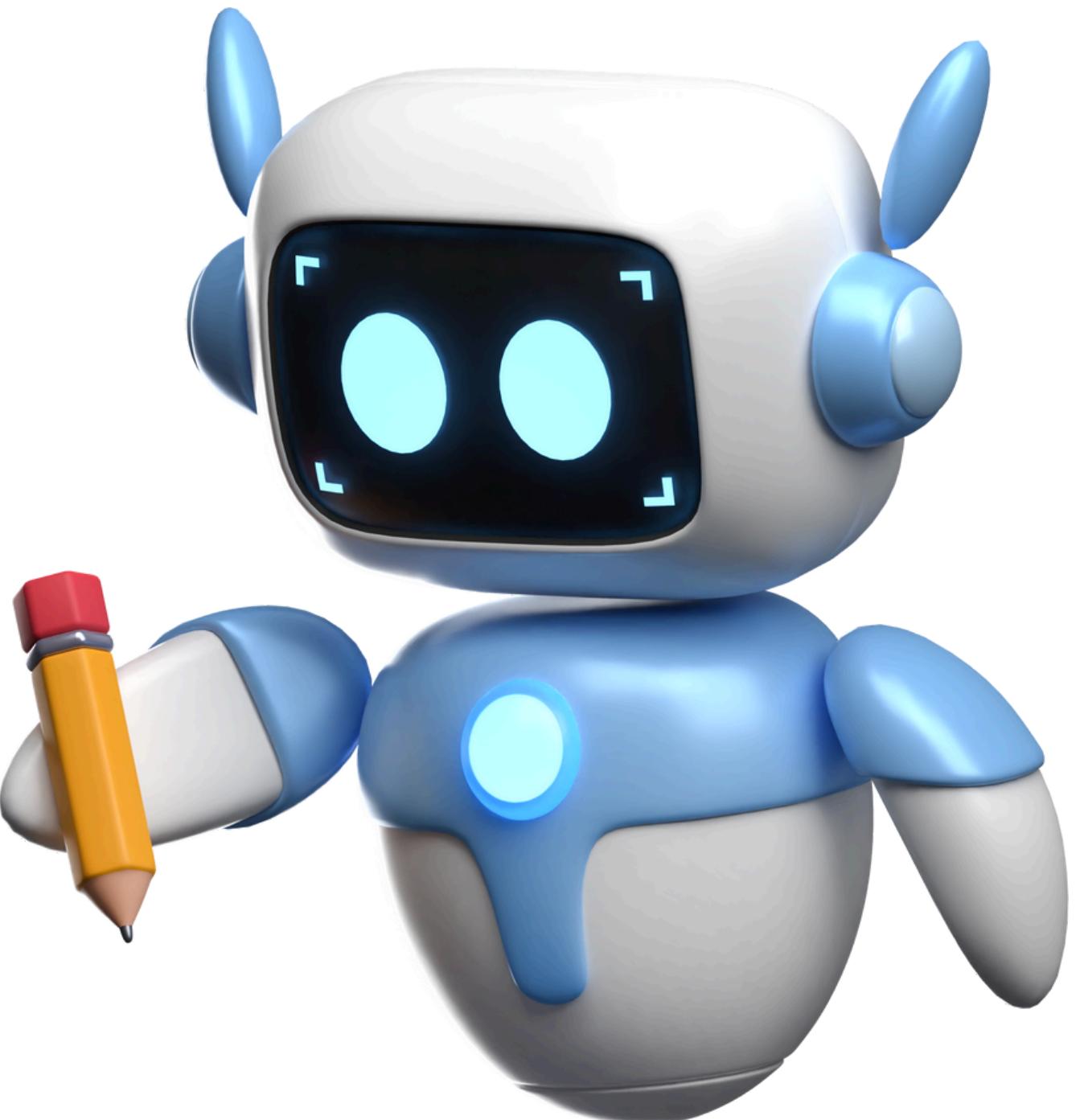


# RECYCLERVIEW



# CLASE PARA LOS ELEMENTOS

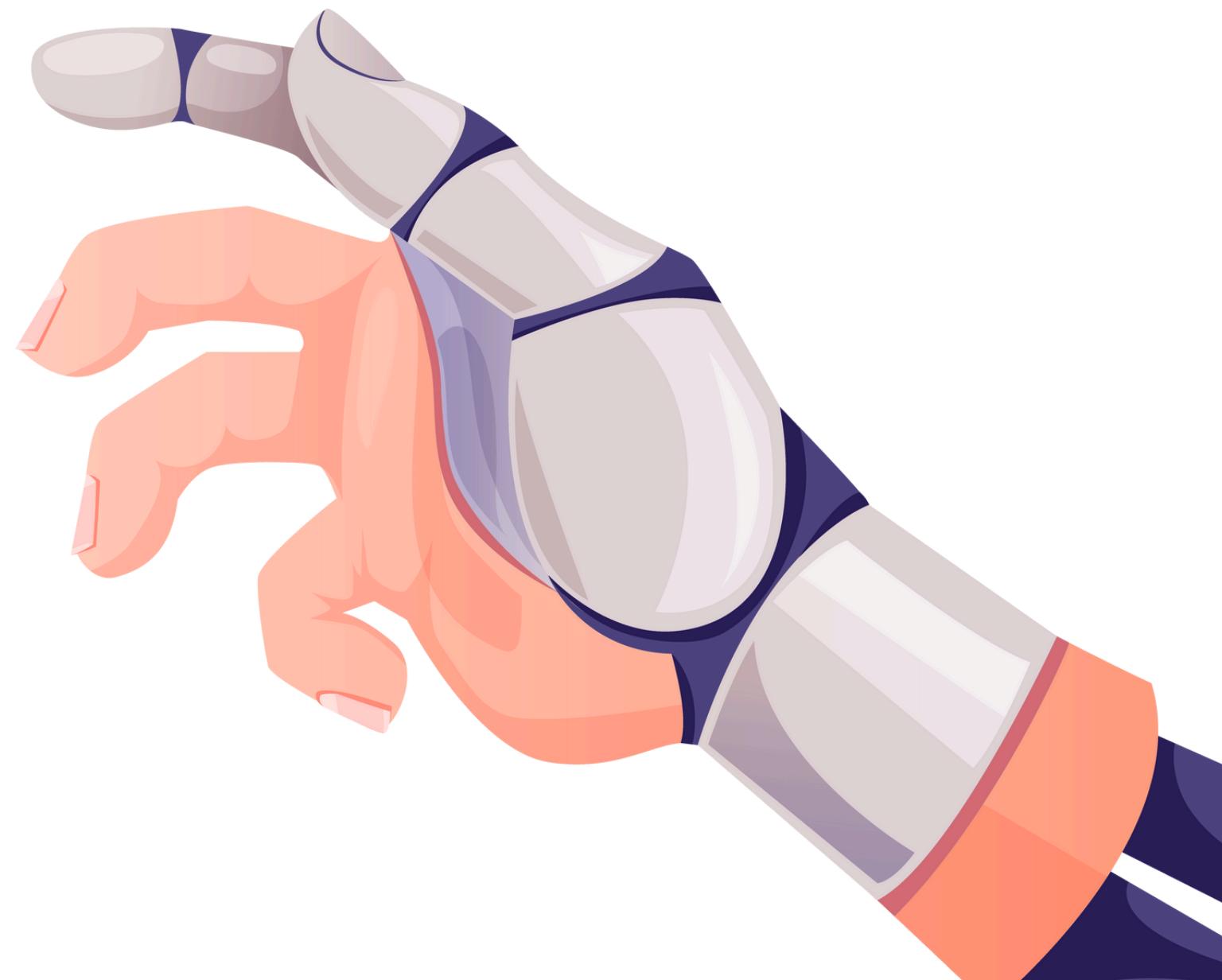
- Una vez tengamos listo el diseño de cada **CardView**, debemos dejar un poco de lado los colores y animaciones, para pasar al proceso lógico.
- En la misma carpeta que se encuentra nuestro **MainActivity**, vamos a crear una nueva **java.class**, que contendrá lo más básico de una clase en sí.





# ADAPTADOR

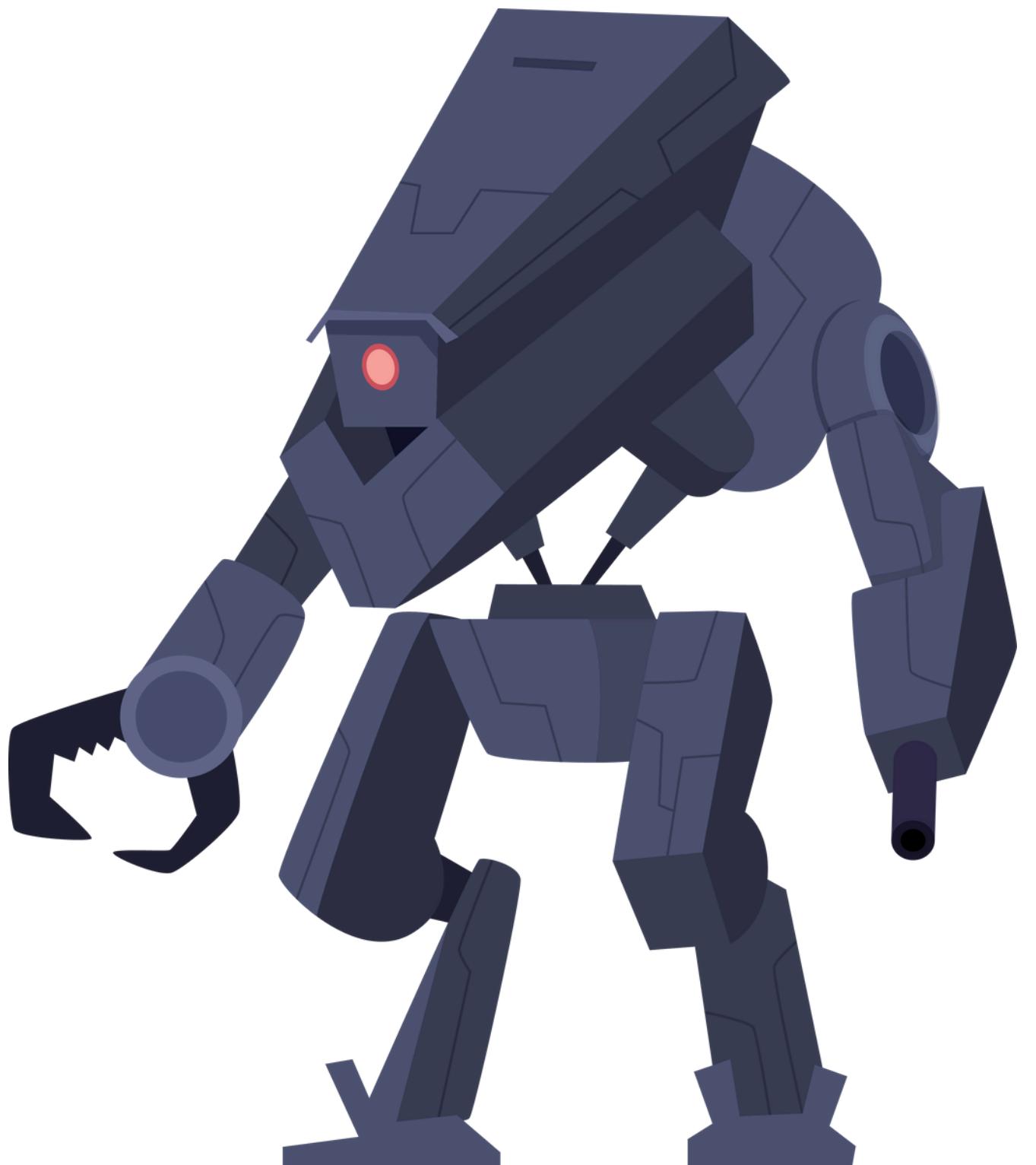
- El siguiente paso se puede tornar un poco complejo y engorroso de entender, sobre todo cuando recién se está aprendiendo, pero es sumamente necesario para el correcto funcionamiento de **RecyclerView**.
- Crearemos una nueva java.class que va a extender de  
**RecyclerView.Adapter<ListAdapter.ViewHolder>**
- **LayoutInflater:** Se utiliza para “inflar” vistas dinámicas en tiempo de ejecución, es decir, permitir que se modifiquen sin la necesidad de terminar el proyecto.





## MÉTODO EN MAIN

- Una vez tengamos nuestro adaptador creado, podemos crear un método en la pestaña de MainActivity que nos permita ingresar uno o más CardView a nuestro proyecto.
- **setHasFixedSize:** Lo utilizamos para optimizar el rendimiento de nuestro RecyclerView. Al tenerlo en true, evitamos cálculos innecesarios dentro de la lista.
- **setLayoutManager:** Organiza la posición de los elementos dentro de un RecyclerView.
- **setAdapter:** Se utiliza para conectar un RecyclerView con un adaptador en específico.



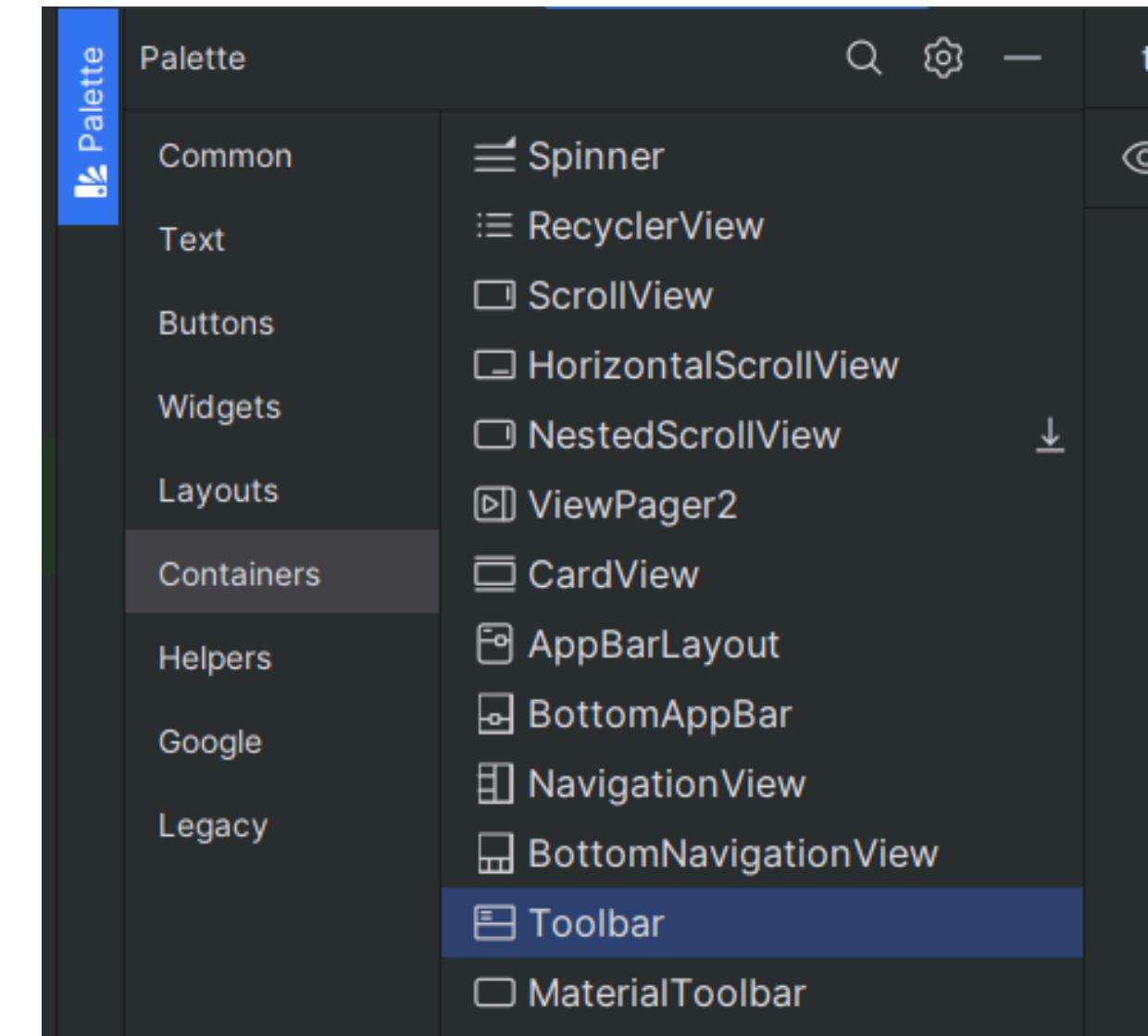


# TOOLBAR



# TOOLBAR

- Uno de los elementos más utilizados en diferentes aplicaciones es la Toolbar que cuenta con múltiples características.
- No solo otorga un plus de diseño, también se pueden enlazar diferentes tipos de menú o botones que generen nuevas pestañas o acciones.
- Es importante tener en cuenta que para visualizarlo, debemos hacer referencia a la toolbar en el MainActivity.



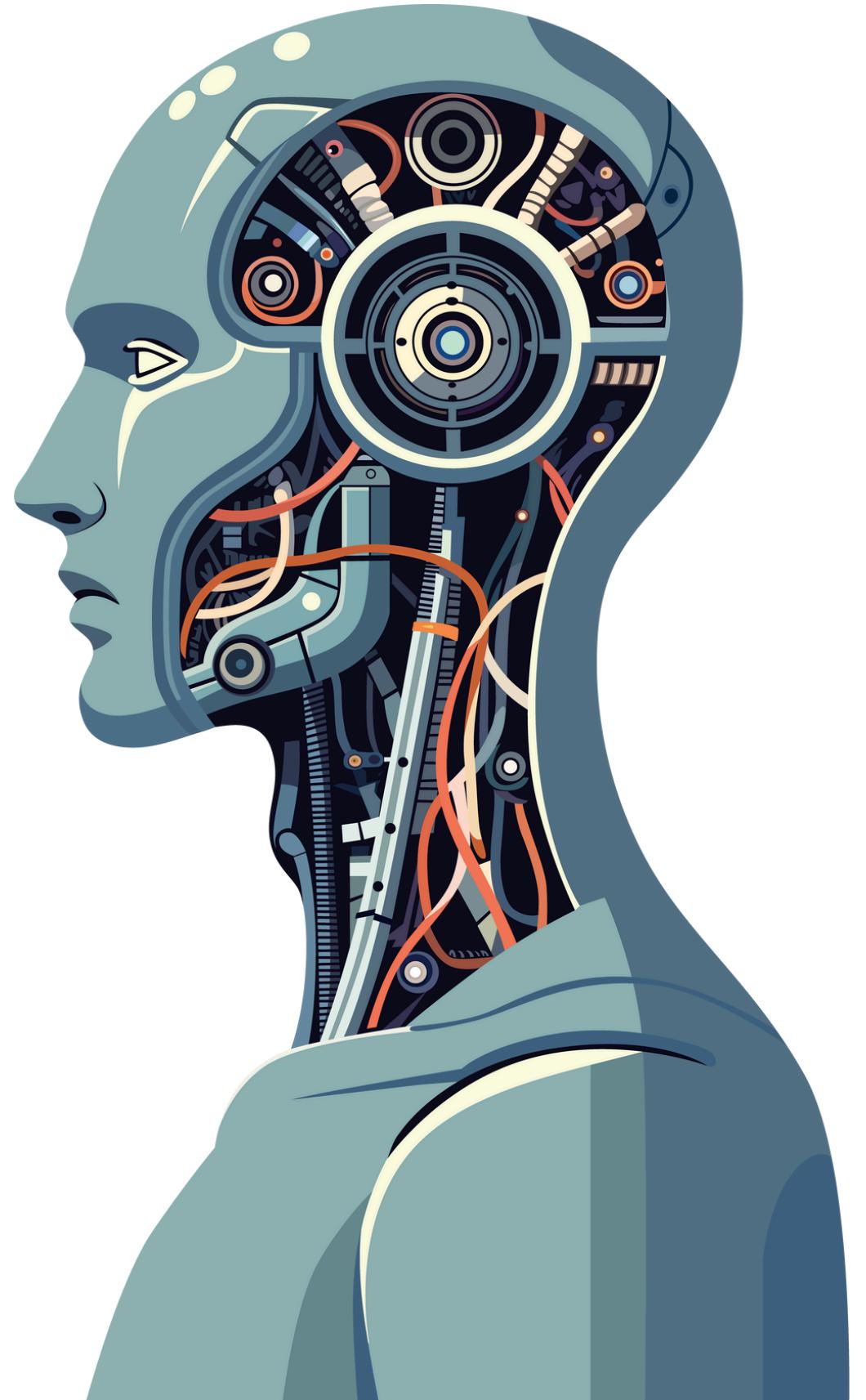


# ONCLICK EN RECYCLERVIEW



# ONCLICK EN RECYCLERVIEW

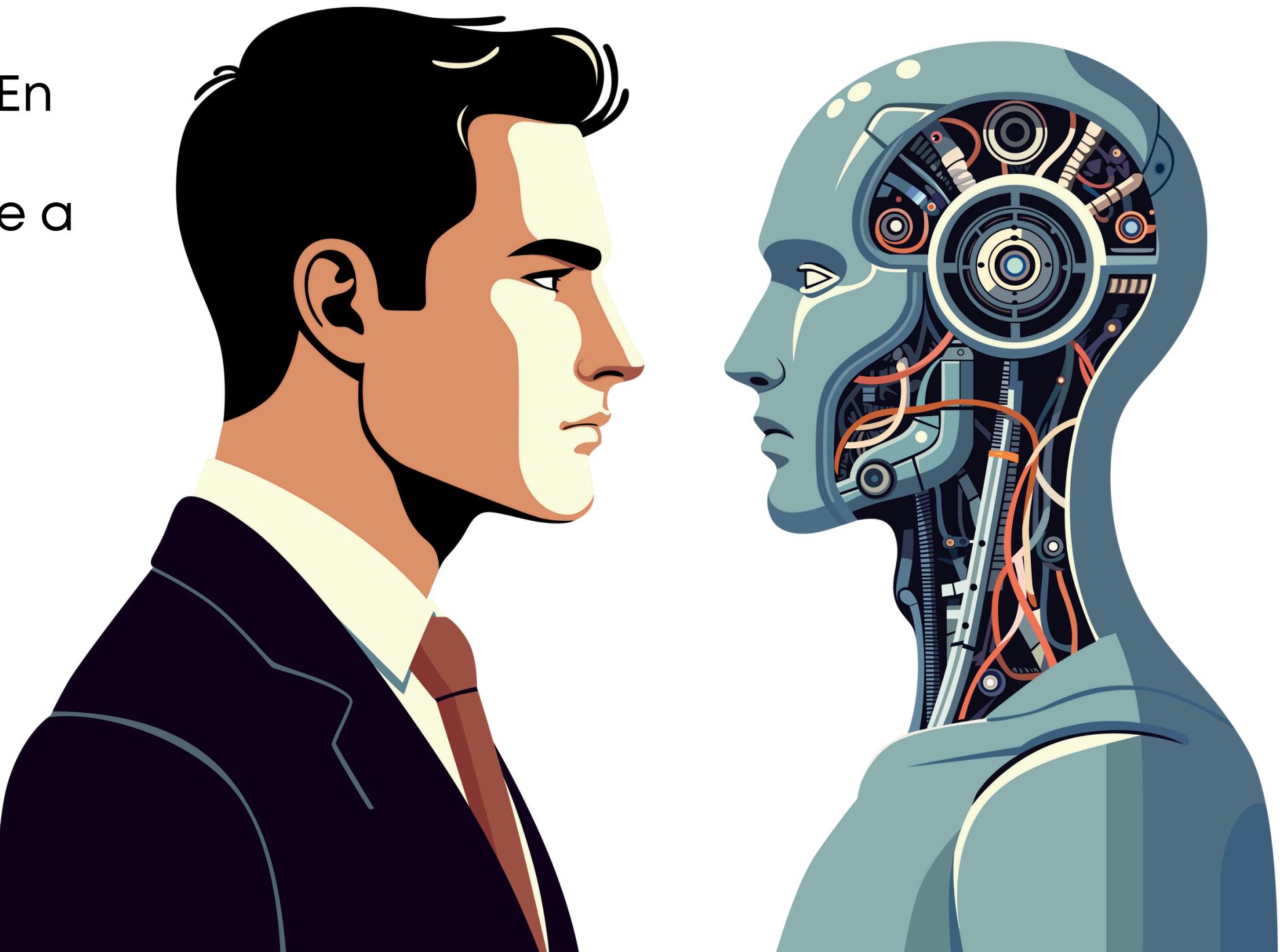
- Lo primero es ir directamente al adaptador e implementar una interfaz de **View.OnClickListener**
- Con esto se implementará automáticamente el método **onClick(View view)**.
- Procederemos a crear una variable privada del tipo interfaz **View.OnClickListener**.
- Dentro de nuestro onCreateViewHolder, crearemos una vista de nuestro método: **view.setOnClickListener(this)**.





# ONCLICK EN RECYCLERVIEW

- Lo siguiente es crear un método de tipo void que pueda modificar nuestro **listener** con el nuevo que está recibiendo.
- Con eso listo, podemos aplicar una validación sobre el método **onClick**: En caso de que **listener** no sea nulo, se disparará el método correspondiente a **onClick**.





# ONCLICK EN RECYCLERVIEW

- Una vez tengamos nuestro adaptador configurado, nos dirigimos al **MainActivity**, para capturar directamente el click que necesitamos.
- En base al objeto del adaptador, invocaremos al método **setOnItemClickListener**.
- Dentro de él crearemos una notificación con la ayuda del método **Toast**.





**GRACIAS POR  
LA ATENCIÓN**