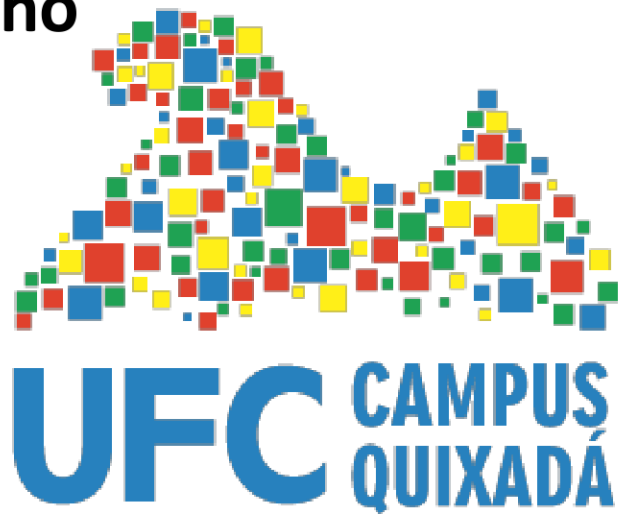
# Desenvolvimento de software para Dispositivos Móveis

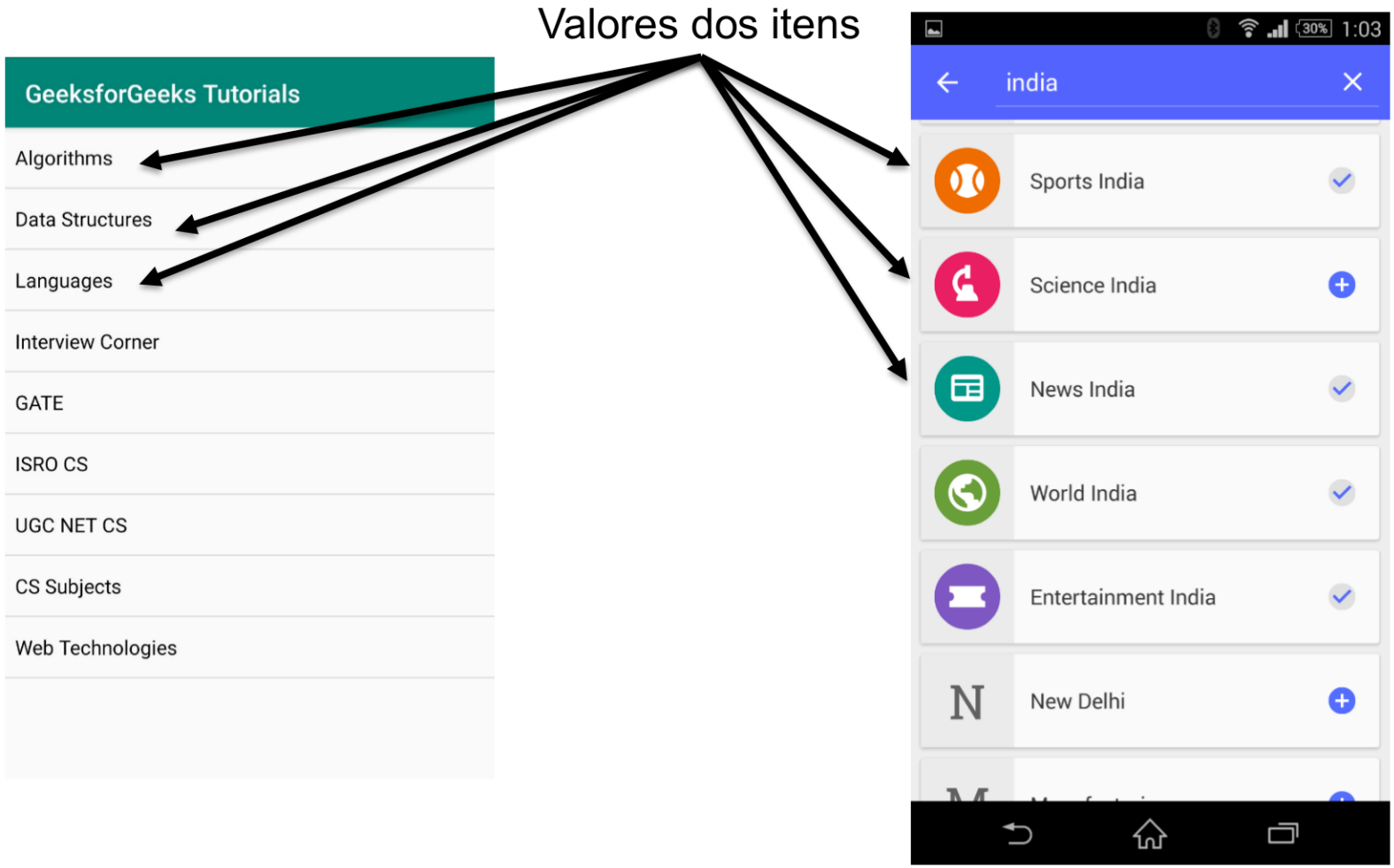
**Adapters: ListView, Custom ListView, RecyclerView**


**Prof. Sidartha Carvalho**



# Adapters

- Servem para popular alguns Widgets



# Adapters

- Adapter
  - Ponte entre uma fonte de dados e um Adapter View.
  - Provê acesso aos dados dos itens
  - Na prática, o Adapter é necessário para fornecer uma interface conhecida de como acessar os dados de um item, permitindo que o Android crie os itens dentro de listas para o usuário
  - Fontes de dados
    - ArrayList, HashMap, SQLite, etc
  - Adapter View – Componente de UI
    - ListView, RecyclerView, GridView, etc

# Adapters

- Simple Adapter
- Custom Adapter
- Recycler View

# Simple Adapter

- Usado para popular o widget ListView

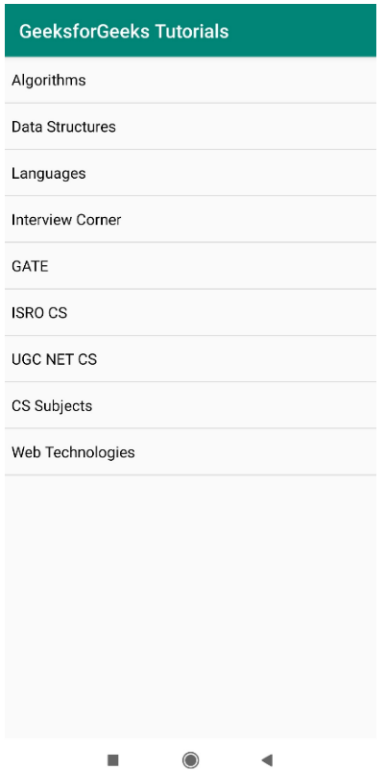
1. Criar a fonte dos dados

2. Criar um objeto adapter
   - ArrayAdapter (Context, LayoutItem, fonteDados);

3. Linkar o widget ListView com o Adapter criado

# Simple Adapter

```
// 1- AdapterView: ListView
lv_cidades = findViewById(R.id.lv_cidades);

// 2- Data Source: String Array
String[] cidades = {"Quixada", "Fortaleza", "Quixeramobim","Banabuiu", "Madalena"};

// 3- Adapter: acts as a bridge between the
//          'data source' and the 'AdapterView'
ArrayAdapter<String> adapter = new ArrayAdapter<>(
      this,
      android.R.layout.simple_list_item_1,
      cidades
);

// 4- Link Listview with the Adapter
lv_cidades.setAdapter(adapter);
```



# Adapters

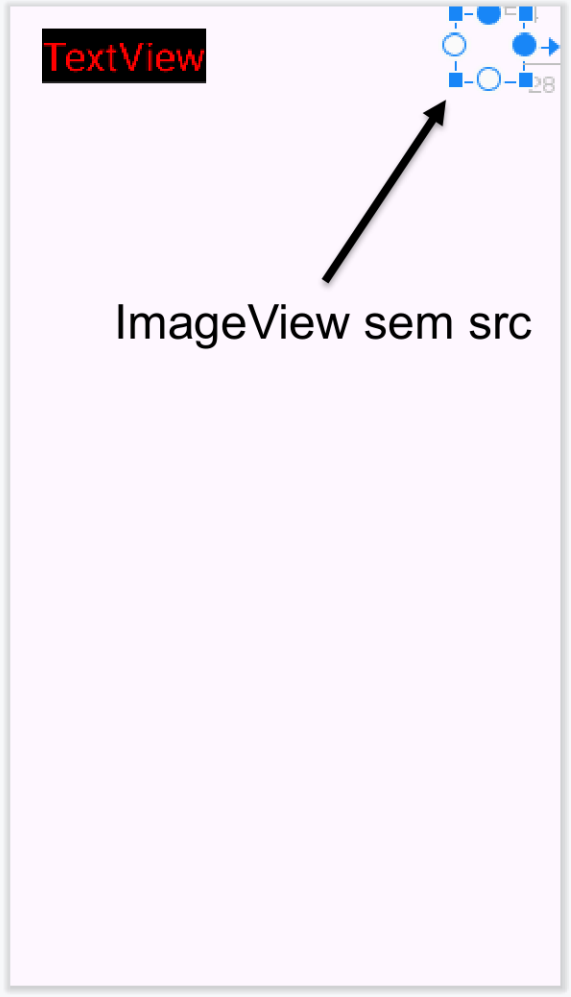
- Simple Adapter
- Custom Adapter
- Recycler View

# Custom Adapter

1. Criar o layout do item que será exibido na lista
2. Criar o layout da MainActivity com o ListView
3. Linkar o elemento da ListView na MainActivity
4. Criar a classe Java para ser o Adapter customizado (extends BaseAdapter)

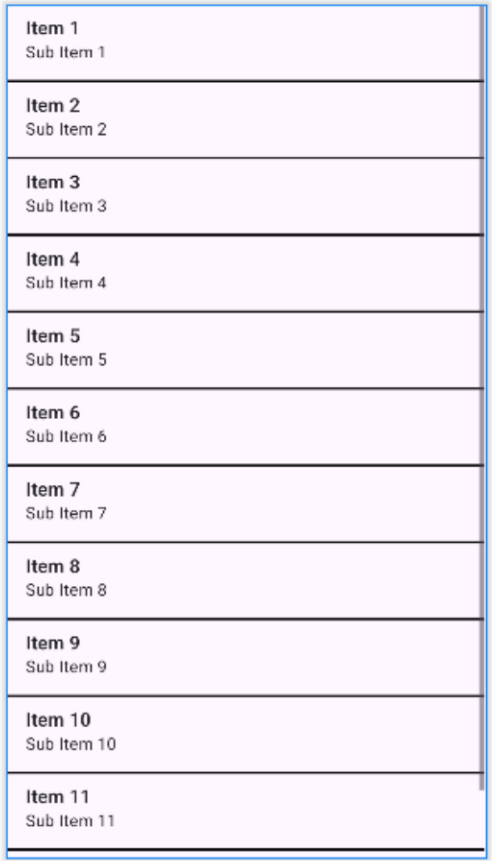
# Custom Adapter: Step 1

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/tv_item"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="24dp"
        android:layout_marginTop="16dp"
        android:background="@color/black"
        android:text="TextView"
        android:textColor="#FF0000"
        android:textSize="30dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <ImageView
        android:id="@+id/iv_item"
        android:layout_width="51dp"
        android:layout_height="50dp"
        android:layout_marginTop="4dp"
        android:layout_marginEnd="28dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



# Custom Adapter: Step 2

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".customadapter.ListViewCustomAdapterExample">

    <ListView
        android:id="@+id/lv_custom_adapter"
        android:layout_width="409dp"
        android:layout_height="729dp"
        tools:layout_editor_absoluteX="1dp"
        tools:layout_editor_absoluteY="1dp" />
</androidx.constraintlayout.widget.ConstraintLayout>
```



# Custom Adapter: Step 3

```
public class ListViewCustomAdapterExample extends AppCompatActivity {

    ListView listview;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_list_view_custom_adapter_example);

        // 1- AdapterView: ListView
        listview = findViewById(R.id.lv_custom_adapter);

        // 2- Data Source: String Array
        String[] cidades = {"Quixada", "Fortaleza", "Quixeramobim","Banabuiu", "Madalena"};


        // 3- Adapter: acts as a bridge between the
        //          'data source' and the 'AdapterView'
        ListViewCustomAdapter adapter = new ListViewCustomAdapter(this, cidades);

        // Link Listview with the Adapter
        listview.setAdapter(adapter);

    }
}
```

# Custom Adapter: Step 4.1

```
public class ListViewCustomAdapter extends BaseAdapter {

   Context context;
   String[] items;

   ListViewCustomAdapter(Context context, String[] items){
      this.context = context;
      this.items = items;
   }

   @Override
   public int getCount() {
      return items.length;  // Returns the number of items in your data source
   }

   @Override
   public Object getItem(int position) {
      return items[position]; // Returns the data item at the given position
   }

   @Override
   public long getItemId(int position) {
      return position; // Returns a unique Identifier for the item at the given position
   }
```

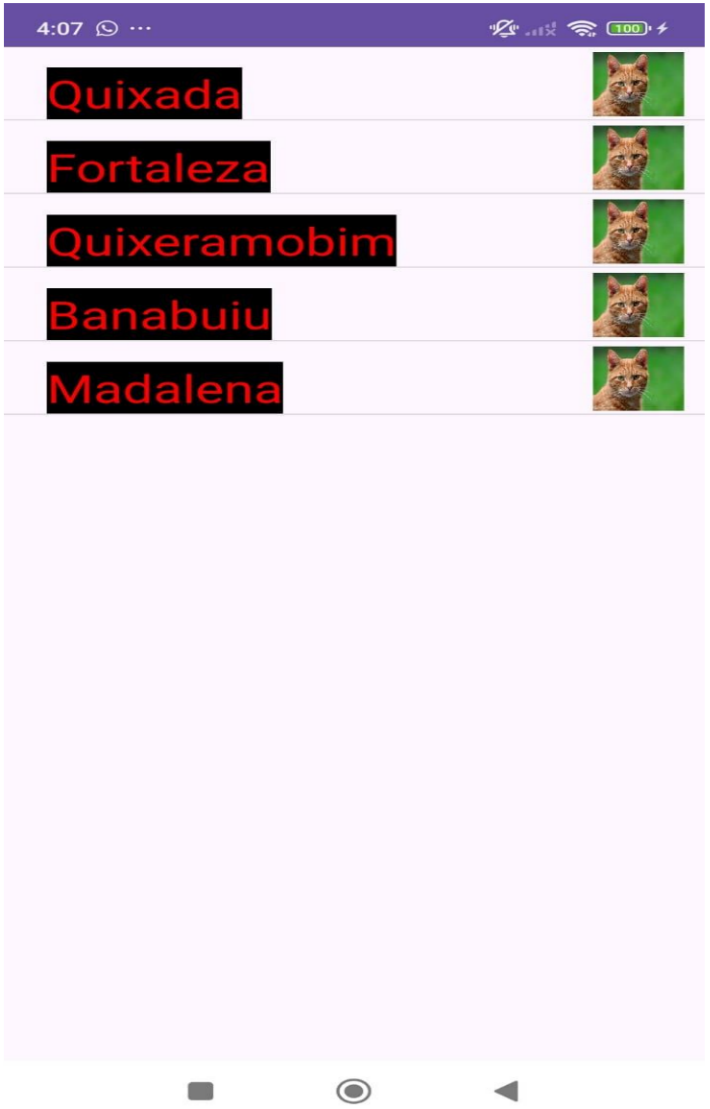
**CONTINUA NO PRÓXIMO SLIDE…**

# Custom Adapter: Step 4.2

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
   ViewHolder holder;

   if (convertView == null){
      // convertView: is a recycled View that you can reuse to
      //           improve the performance of your list.
      convertView= LayoutInflater.from(context)
            .inflate(R.layout.item_custom_layout_list_view, parent, false);

      holder = new ViewHolder();
      holder.tv_item = convertView.findViewById(R.id.tv_item);
      holder.iv_item = convertView.findViewById(R.id.iv_item);
      convertView.setTag(holder);

   }else{
      // Reusing the View (that's recycled)
      holder = (ViewHolder) convertView.getTag();
   }

   // Set the data to the view
   holder.tv_item.setText(items[position]);

   // Set the image...
   holder.iv_item.setBackgroundResource(R.drawable.img);

   // Binding data to views within the convertView
   return convertView; // Displays the data at a position in the data set
}

static class ViewHolder{
   // Holds references to the views within an item layout
   TextView tv_item;
   ImageView iv_item;

}
}
```

# Custom Adapter: Resultado

- No exemplo, usamos a mesma imagem para todos os itens, mas cada item pode receber uma imagem diferente. Inclusive, podemos alterar nosso modelo de Item para armazenar a referência para a imagem.

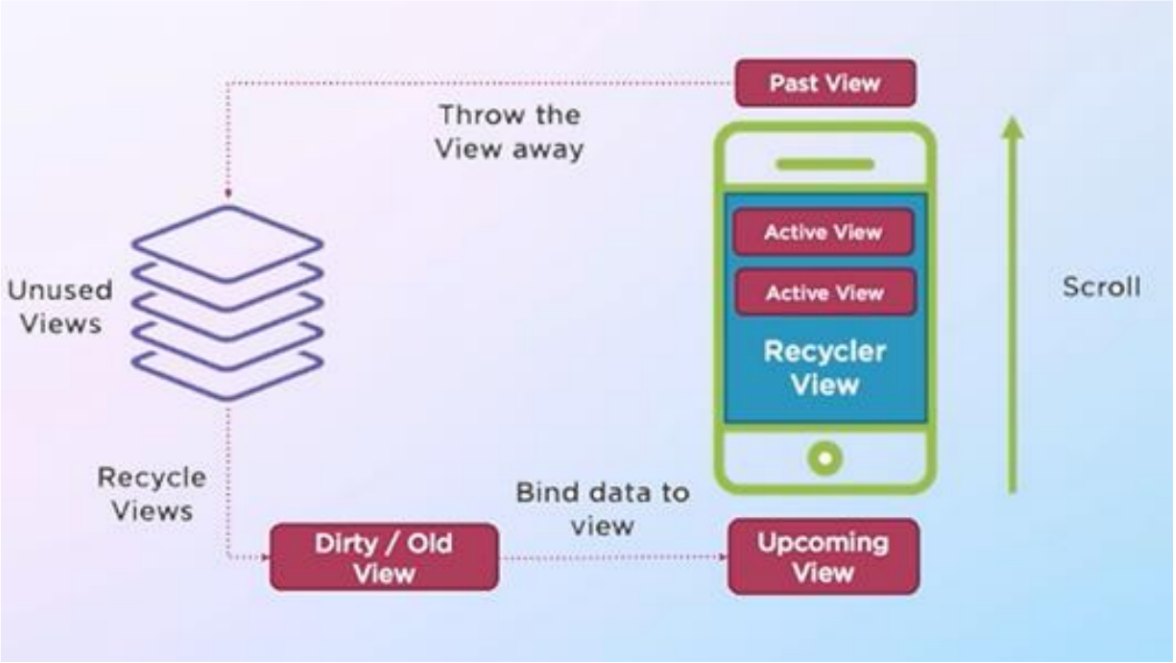


# Adapters

- Simple Adapter
- Custom Adapter
- Recycler View

# Recycler View

- Permite economizar recursos, mantendo na memoria somente os elementos que estão sendo visualizados no momento pelo usuário.
- Sempre que um item for “scrollado” para fora da tela, para cima ou para baixo, ele será removido da memória.
- Os novos itens serão renderizados e carregados na memória conforme a tela for rolando.



# Recycler View

- O widget RecyclerView exige que haja um Custom Adapter, similar ao que fizemos no exemplo anterior do Custom ListView.

# Recycler View

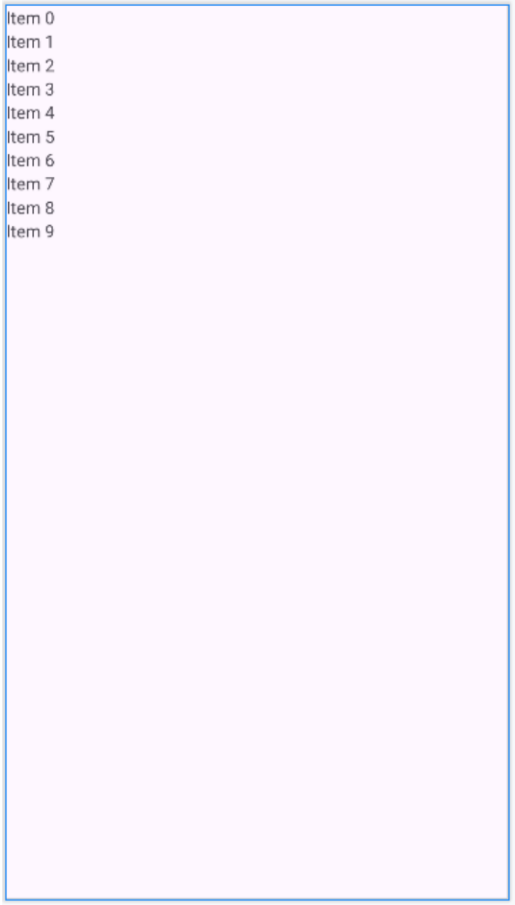
1. MainActivity
   - Referência a UI RecyclerView
   - Definir o Layout e o Adapter
2. Criação do XML contendo o RecyclerView Widget
3. list_item.xml
   - Criação do modelo XML que define cada item da lista
4. Item.java
   - Classe Java para permitir o acesso aos atributos do Item
5. ItemArrayAdapter.java
   - extends RecyclerView.Adapter<ItemArrayAdapter.ViewHolder>
   - Implementar vários métodos obrigatórios

# Recycler View: Step 1 - MainActivity

```
// 1- AdapterView: ListView
rv_cidades = findViewById(R.id.rv_cidades);

// Initializing list view with the custom adapter
ArrayList <Item> itemList = new ArrayList<Item>();

ItemArrayAdapter itemArrayAdapter = new ItemArrayAdapter(R.layout.list_item, itemList);

rv_cidades = (RecyclerView) findViewById(R.id.rv_cidades);

rv_cidades.setLayoutManager(new LinearLayoutManager(this));

rv_cidades.setAdapter(itemArrayAdapter);

// Populating list items
for(int i=0; i<100; i++) {
   itemList.add(new Item("Item " + i));
}
```

# Recycler View: Step 2 - activity_recycler_view_example.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".RecyclerViewExample">

    <androidx.recyclerview.widget.RecyclerView
        android:layout_width="409dp"
        android:layout_height="729dp"
        android:id="@+id/rv_cidades"
        tools:layout_editor_absoluteX="1dp"
        tools:layout_editor_absoluteY="1dp"
        />
</androidx.constraintlayout.widget.ConstraintLayout>
```



# Recycler View: Step 3 - list_item.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical" android:layout_width="match_parent"
   android:layout_height="wrap_content">
   <TextView
      android:id="@+id/row_item"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:textSize="30dp"
      android:gravity="center"/>
</LinearLayout>
```

# Recycler View: Step 4 - Item.java

```
public class Item {
    private String name;

    public Item(String n) {
        name = n;
    }
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

# Recycler View: Step 5 - ItemArrayAdapter.java

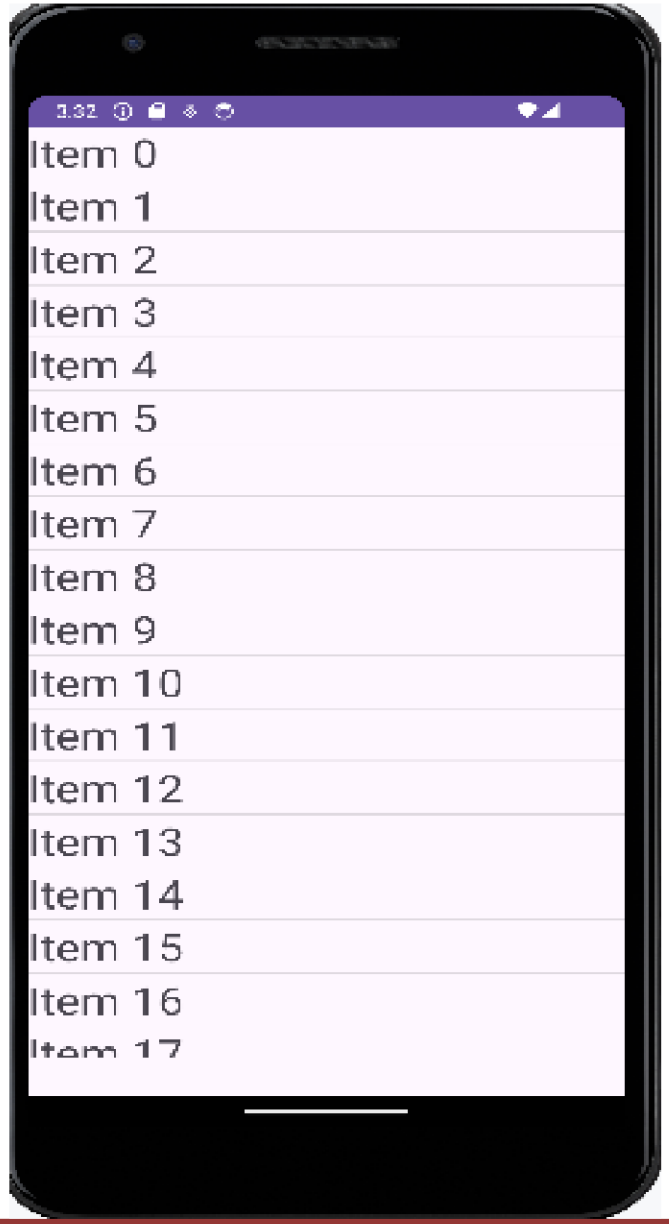
```
public class ItemArrayAdapter extends RecyclerView.Adapter<ItemArrayAdapter.ViewHolder> {

    private int listItemLayout;
    private ArrayList<Item> itemList;

    // Constructor of the class
    public ItemArrayAdapter(int layoutId, ArrayList<Item> itemList) {
        listItemLayout = layoutId;
        this.itemList = itemList;
    }

    // get the size of the list
    @Override
    public int getItemCount() {
        return itemList == null ? 0 : itemList.size();
    }

    // specify the row layout file and click for each row
    @Override
    public ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(listItemLayout, parent, false);
        ViewHolder myViewHolder = new ViewHolder(view);
        return myViewHolder;
    }
```

**CONTINUA NO PROXIMO SLIDE…**

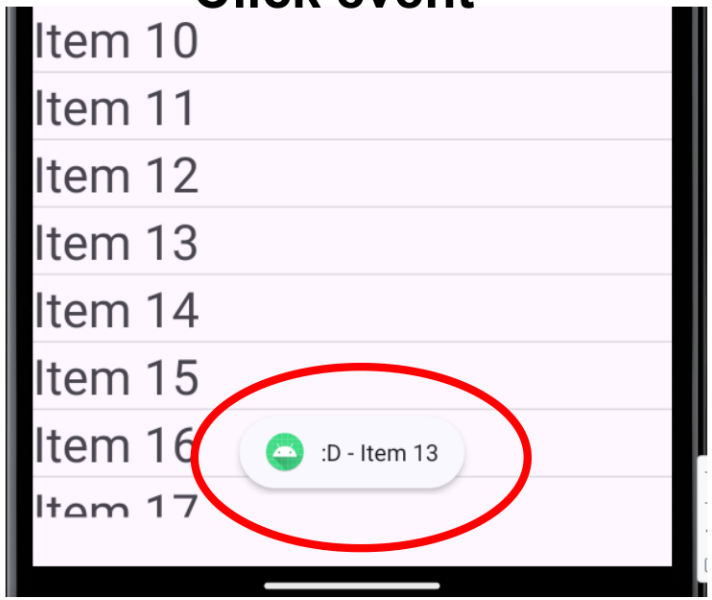
# Recycler View: Step 5 - ItemArrayAdapter.java

```
    // load data in each row element
    @Override
    public void onBindViewHolder(final ViewHolder holder, final int listPosition) {
        TextView item = holder.item;
        item.setText(itemList.get(listPosition).getName());
    }

    // Static inner class to initialize the views of rows
    static class ViewHolder extends RecyclerView.ViewHolder implements View.OnClickListener {
        public TextView item;
        public ViewHolder(View itemView) {
            super(itemView);
            itemView.setOnClickListener(this);
            item = (TextView) itemView.findViewById(R.id.row_item);
        }

        @Override
        public void onClick(View view) {
            Toast.makeText(view.getContext(), ":D - " + item.getText(), Toast.LENGTH_SHORT).show();
            Log.d("onclick", "onClick " + getLayoutPosition() + " " + item.getText());
        }
    }
}
```

# Recycler View: Resultado




**Click event**



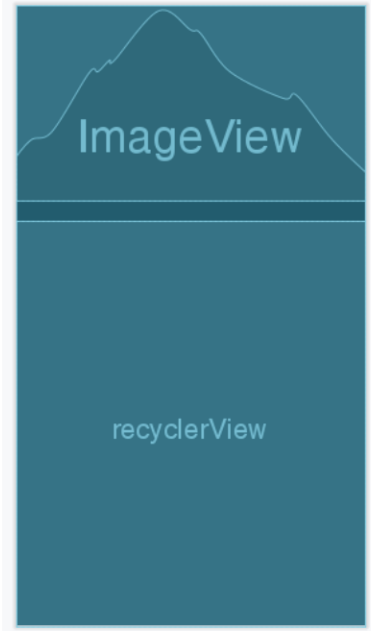
# EXERCÍCIO - ListView

- PlanetApp
  - Ao clicar em cada item mostrar informações sobre o planeta em uma nova activity com seu próprio layout.

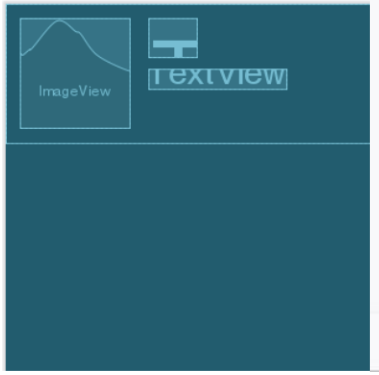
# EXERCÍCIO - RecyclerView

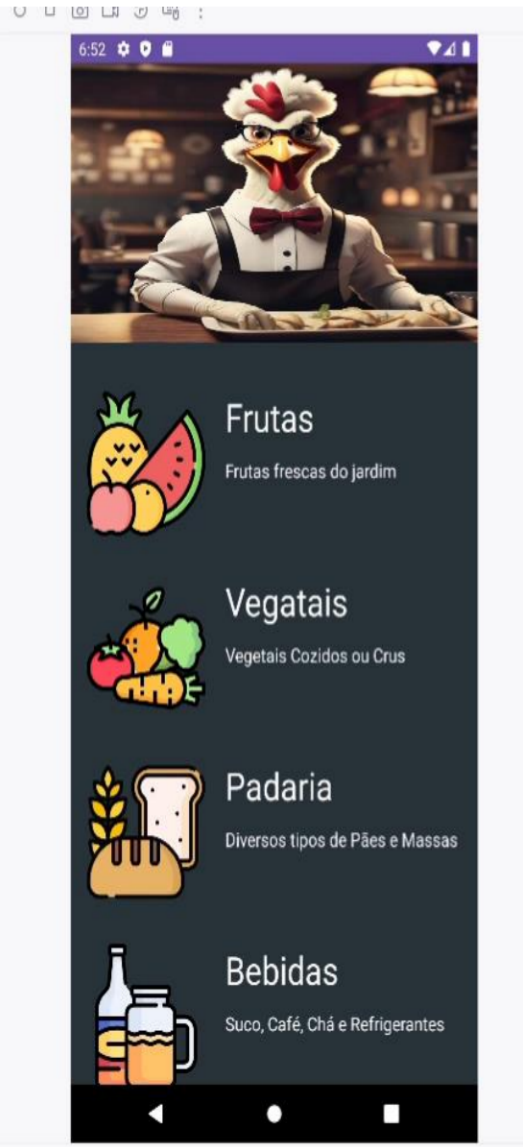
- MarketApp
  - RecyclerView rolável com o menu do restaurante
  - Ao clicar em cada item, mostrar um Toast indicando onde foi clicado

Main Layout




Item Layout



# Referências

- https://developer.android.com/

- https://developer.android.com/courses/fundamentals-training/

# Dúvidas?