**Definirea unui ToolBar cu Menu**

**Adăugarea unui app bar se realizează într-o manieră simplă**, dar are dezavantajul de nu include toate proprietățile noi adăugate. O altă modalitate de a adăuga în aplicație un app bar este utilizarea componentei grafice Toolbar.

**Adăugarea unui Toolbar se realizează prin următoarele etape:**

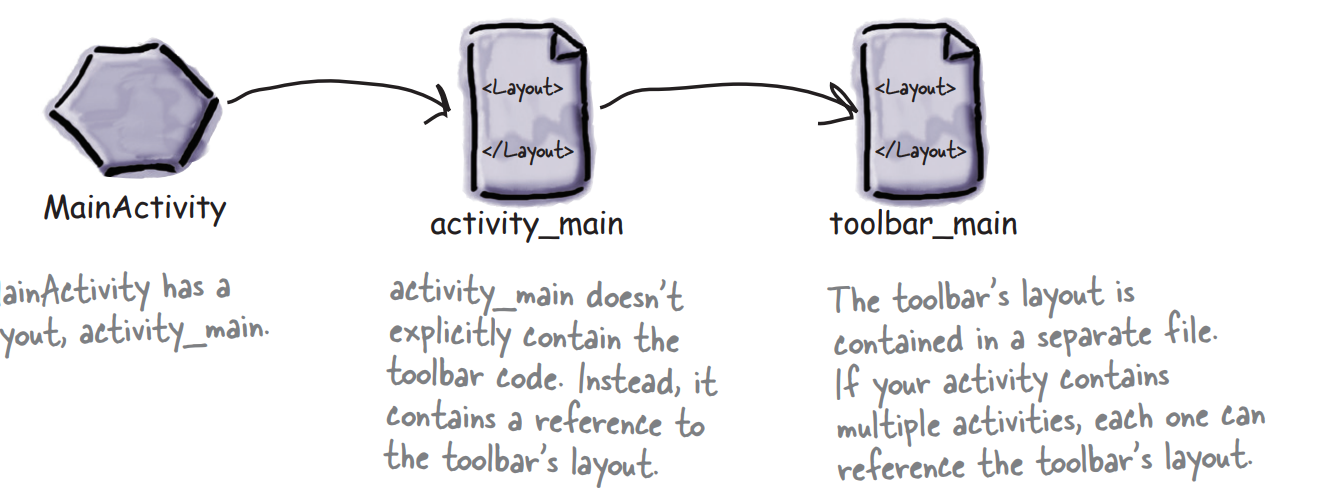
1. Se alege o temă care nu presupune adăugarea unui app bar, în fișierul theme.xml

<style name="Theme.TestActionBar" parent="Theme.AppCompat.Light.NoActionBar">

1. Se adaugă în fișieul layout.xml componenta Toolbar

<androidx.appcompat.widget.Toolbar  
 android:id="@+id/toolbar"  
 android:layout\_width="409dp"  
 android:layout\_height="wrap\_content"  
 android:layout\_marginTop="4dp"  
 android:background="?attr/colorPrimary"  
 android:minHeight="?attr/actionBarSize"  
 android:theme="?attr/actionBarTheme"  
 app:layout\_constraintEnd\_toEndOf="parent"  
 app:layout\_constraintStart\_toStartOf="parent"  
 app:layout\_constraintTop\_toTopOf="parent" />

**Observație**: componenta Toolbar se poate insera într-un layout separat, astfel încât să fie inserat ulterior în mai multe activități ale aplicației.



P1. Se definește componenta grafică toolbar într-un fișier xml

**File -> New -> Layout resource file**

*<?*xml version="1.0" encoding="utf-8"*?>*<androidx.appcompat.widget.Toolbar  
xmlns:android="http://schemas.android.com/apk/res/android"  
android:layout\_width="match\_parent"  
android:layout\_height="wrap\_content"  
android:background="?attr/colorPrimary"/>

P2. În fișierul layout al activității se folosește proprietate include layout pentru a include layoutul ce conține toolbar

<include  
 layout="@layout/toolbar\_main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/toolbar"  
 />

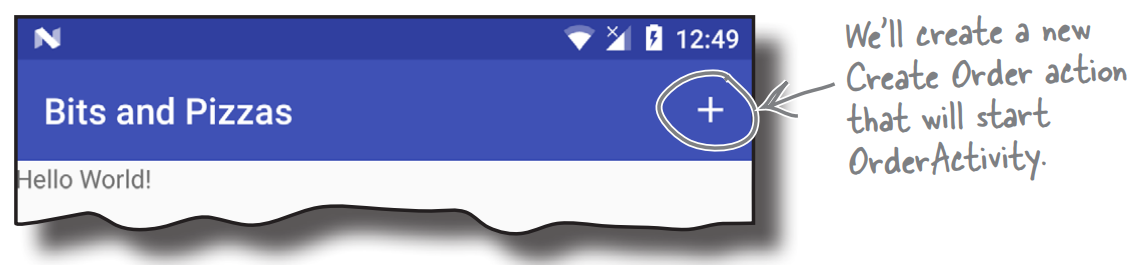
P3. Accesarea componentei toolbar din fișierul java

Toolbar toolbar = (Toolbar) findViewById(R.id.*toolbar*);  
**setSupportActionBar(toolbar);**

**Definirea acțiunilor pentru un app bar/toolbar**

În majoritatea aplicațiilor este nevoie să adăugam acțiuni în bara de aplicații. Acestea pot fi butoane sau text care prin interacțiunea cu utilizatorul conduc la un eveniment. **De exemplu, deschiderea unei noi activități.**

Aplicație practică: definiți un toolbar de forma de mai jos, care conduce la deschiderea unei noi activități



P1. **Crearea unei noi activități OrderActivity**

**P2. Inlcuderea în layout-ul activității noi a componentei toolbar definită în layoutul anterior**

<include  
 layout="@layout/toolbar\_main"  
 android:layout\_width="match\_parent"  
 android:layout\_height="wrap\_content"  
 android:id="@+id/toolbar"  
 />

P3. Accesarea componetei Toolbar în fișierul java

Toolbar toolbar = (Toolbar) findViewById(R.id.*toolbar*);  
setSupportActionBar(toolbar);

P4 În fișierul de resurse values se definește o resursă de tip String pentru numele activității în scopul de a schimba eticheta (textul) din componenta Toolbar, în raport cu denumirea activității curente

<string name="create\_order">Create Order</string>

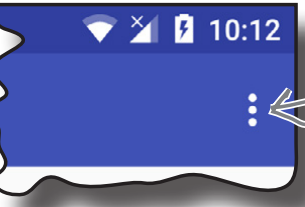
P4. Definirea unui text pentru bara de acțiune

După cum a fost specificat sus, textul care se afișeazp în bara de aplicații presupune utilizarea atributul etichetă din fișierul AndroidManifest.xml.

<activity  
 android:name=".OrderActivity"  
 android:label="@string/create\_order"  
 android:exported="false">

P5. Adăugarea unei resurse unei resurse pentru o acțiune

Pentru a adăuga o acțiune la o bară de aplicații, în general, acesteia i se atribuie o **pictogramă** și un scurt titlu de text. Pictograma este de obicei afișată dacă acțiunea apare în zona principală a barei de aplicații. Dacă acţiunea nu se potrivește în zona principală, este mutată automat în zona de afișare out (cele 3 puncte), iar titlul apare în schimb.



Pentru a adauga un text, se definește o resursă de tip String în fișierul values.xml

<string name ="create\_order\_title">Create Order</string>

**Definirea unui meniu în bara de acțiune se realizează într-un director separat, în cadrul directorul de resurse res. Astfel, se creează o referință către resursa meniu în fișierul R, care va permite accesarea ulterioară a sa.**

res - > Android Resource Directory->resurse type Menu (in directorul menu)

menu-> new Android Resource File -> my\_menu.xml

În documenul my\_menu.xml se definesc opțiunile unui meniu prin proprietatea item (titlu, icon etc), de exemplu:

<**menu xmlns:android="http://schemas.android.com/apk/res/android"**>  
  
<**item android:title="Settings" android:id = "@+id/settings"**>

</**item**>item>  
<**item android:title="Help" android:id = "@+id/help"** ></**item**>item>  
  
</**menu**>

Adăugarea efectivă a unui meniu în bara de acțiune a unei activități se realizează prin adăugarea metodei onCreateOptionsMenu() în clasa care modelează activitatea respectivă.

**public boolean** onCreateOptionsMenu(Menu menu) {  
 MenuInflater menuInflater = getMenuInflater();  
 menuInflater.inflate(R.menu.***main\_menu***, menu);  
  
 **return super**.onCreateOptionsMenu(menu);  
}

Clasa MenuInflater permitea definirea/manipularea unui meniu cu referința definită în fișierul R, accesarea efectivă realizându-se prin apelul metodei getMenuInflater().

Selecția de către utilizator a unei opțiuni impune tratarea evenimentului prin implementarea metodei onOptionsItemSelected()

@Override  
**public boolean** onOptionsItemSelected(@NonNull MenuItem item) {  
 **int** itemId = item.getItemId();  
 **if**(itemId==R.id.*setting*)  
 {  
 Toast.*makeText*(**this**, **"You selected Setting"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 **else if**(itemId==R.id.*search*)  
 {  
 Toast.*makeText*(**this**, **"You selected Search"**, Toast.***LENGTH\_SHORT***).show();  
 }  
 **return super**.onOptionsItemSelected(item);  
}

**Aplicație practică**: Dezvoltați o aplicație Android cu o activitate ce conține în bara de acțiune un meniu cu două opțiuni, repectiv Settings, Help, și realizează afișarea unui mesaj de tip Toast ce cuprinde denumirea opțiuniii selctate.

**Definirea unui layout personalizat. Conceptul Inflate**

**Un layout modificat (sau personalizat) în Android este un fișier XML** care descrie modul în care elementele interfeței cu utilizatorul sunt aranjate și afișate. Layout-urile personalizate sunt utile pentru diverse scopuri în dezvoltarea de aplicații Android, precum:

* *Controlul Aspectului InterfețeiUtilizator*: Layout-urile personalizate permit controlul complet al aspectului și al structurii elementelor interfeței cu utilizatorul (aspecte personalizate pentru a se potrivi cu identitatea vizuală a aplicației sau pentru a oferi o experiență utilizatorilor unică)
* *Structurarea Complexității*: Cu ajutorul layout-urilor personalizate, se poate împărți ecranul în secțiuni sau zone pentru a organiza elemente diverse.

Se pot crea layout-uri personalizate pentru elemente care vor fi utilizate în mai multe locuri din aplicația dezvoltată. Acest lucru promovează reutilizarea și menținerea codului, economisind timp și efort.

* *Complexitatea Interacțiunii Utilizatorului*: Prin utilizarea layout-urilor personalizate, se pot crea ecrane complexe care răspund la diferite acțiuni ale utilizatorului, cum ar fi formulare, liste, meniuri, paginări etc.
* *Gestionarea Afișării Datelor:* Layout-urile personalizate sunt utilizate adesea pentru afișarea datelor dintr-o sursă de date (cum ar fi o bază de date sau serviciu web) într-un format structurat și bine definit (listView, gridView).

În general, layout-urile personalizate reprezintă o componentă fundamentală a dezvoltării de aplicații Android și sunt esențiale pentru a oferi o experiență utilizatorilor cât bine proiectată și adaptată la nevoile specifice unei aplicații.

**Etapele de realizare ale unui layout personalizat (custom layout)**

Definirea un layout personalizat pentru un element **ListView** presuoune utilizarea unui adaptor , precum și a noțiunii **LayoutInflater**.

**LayoutInflater** este o clasă în Android care este utilizată pentru a infla layout-urile definite în fișiere XML în cadrul aplicației. Această clasă este parte a sistemului de servicii Android și este folosită pe scară largă pentru a crea interfețe de utilizator din definiri XML.

**LayoutInflater** u este foarte util pentru a dezvolta interfețe de utilizator complexe dintr-un fișier XML, deoarece permite crearea obiectele **View** corespunzătoare în codul Java corespunzător.

Definirea unui fișierul XML de Layout Personalizat presupune efectuarea următoarelor etape:

P1. **Crearea unui Fișier xml**: Definirea unui layout XML pentru layout-ul personalizat în **directorul res/layout**. Acest fișier va defini aspectul elementelor din listă, de exemplu o imagine, un text etc.

*Exemplu*: pentru o aplicație Notes se pot defini doua elemente TextView: *title, content*

P2. **Modelul de date**: Se defineste o clasa Java pentru a modela date despre elementele grafice din layout-ul personalizat. Această clasă va conține câmpurile necesare pentru text, imagini sau alte date asociate elementului.

public class Note {

private String title;

private String content;

public Note(String title, String content) {

this.title = title;

this.content = content;

}

public String getTitle() {

return title;

}

public String getContent() {

return content;

}

}

P3. **Adaptor Personalizat**: Se definește o clasă de adaptor personalizată care extinde **RecyclerView.Adapter** sau **BaseAdapter** (în funcție de utilizarea RecyclerView sau ListView). Adaptorul va gestiona datele și legătura dintre layout-ul personalizat și datele din model.

De exemplu, pentru o listă, se poate extinde clasa **ArrayAdapter<?T>**

Clasa adaptor conține următoarele elemente:

* O colecție Java pentru a reține lista de elemente
* Un contex care definește contextul layout-ului personalizat

private ArrayList<Note> notes;

private Context context;

* **O clasa statică care conține referințele elementelor grafice din layout XML**.

Aceasta este utilizată pentru a păstra referințele către elementele din layout (TextView title și TextView content) și pentru a le accesa rapid în metoda **getView().** Folosind **convertView.setTag(holder)** și **convertView.getTag(),** se poate obține și setă referințele către elementele ViewHolder asociate cu fiecare element din listă. Această abordare va face listarea mai eficientă și va reduce numărul de apeluri **findViewById(),** ceea ce va duce la o performanță mai bună a aplicației.

static class ViewHolder {

TextView title;

TextView content;

}

* Redefinirea metodei **getView**

Metoda **getView()** dintr-o clasă de adaptor, cum ar fi BaseAdapter sau ArrayAdapter, are rolul principal de a returna o vizualizare (View) care să reprezinte elementul din sursa de date (de obicei, o listă de obiecte) la o anumită poziție în lista afișată (de exemplu, ListView sau GridView. Rolul metodei getView() este:

* *I****nflarea Layout-ului***: În primul rând, getView() inflă (transformă din XML în obiect Java) layout-ul corespunzător elementului din listă, de obicei, un fișier XML care descrie aspectul elementului. Acest lucru se face folosind LayoutInflater.
* *Atribuirea Datelor la Elementele UI*: După inflarea layout-ului, getView() atribuie datele corespunzătoare elementului la elementele vizuale din layout (cum ar fi TextView, ImageView, Button, etc.). Acest lucru se face de obicei folosind metode precum setText(), setImageResource(), setBackgroundColor(), etc.
* Returnarea Vizualizării: Metoda getView() returnează vizualizarea rezultată, care reprezintă elementul din listă. Această vizualizare va fi afișată în lista dvs. la poziția specificată.

În general, **getView()** este unul dintre cele mai importante metode dintr-un adaptor, deoarece determină modul în care datele din sursa de date sunt afișate în elementele listei. Personalizarea acestei metode vă permite să creați interfețe de utilizator unice și să controlați modul în care elementele dvs. sunt afișate.

**@Override**

**public View getView**(**int** position, @Nullable View convertView, @NonNull ViewGroup parent)

{

ViewHolder holder;

if (convertView == null) {

convertView = LayoutInflater.from(context).inflate(R.layout.list\_item, parent, false);

holder = new ViewHolder();

holder.title = convertView.findViewById(R.id.note\_title);

holder.content = convertView.findViewById(R.id.note\_content);

**convertView.setTag(holder);**

} else {

holder = (ViewHolder) convertView.**getTag();**

}

Note currentNote = notes.get(position);

holder.title.setText(currentNote.getTitle());

holder.content.setText(currentNote.getContent());

return convertView;

}

P4. Inflarea ListView-ului sau GridView-ului:

* În layout-ul activității principale., se adăugă o referința către ListView pentru a afișa datele corespunzătoare. Aceasta implică adăugarea unei vizualizări cu ID-ul corespunzător și cu un android:id asociat.
* Instantierea Adaptorului implementat
* Setarea Adaptorului pentru ListView

ListView listView = findViewById(R.id.listView);

NotesAdapter notesAdapter = new NotesAdapter(this, notes);

listView.setAdapter(notesAdapter);

* Manipularea Evenimentelor sau Interacțiunilor (opțional):

Pentru a gestiona evenimente sau interacțiuni cu elementele listei, cum ar fi detectarea unui clic pe un element, se adăugă ascultători (event listeners) corespunzători pentru elementele ListView. De exemplu, se poate utiliza **setOnItemClickListener()** pentru a răspunde la clicurile pe elementele listei.

listView.setOnItemClickListener(new AdapterView.OnItemClickListener() {

@Override

public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

// Acțiunea pe care doriți să o efectuați la clic pe un element

}

});