

# Chapter 1

## **Program wise Notes**

---

(Full theory and syntax for all 27 programs will be too long to paste here in one go. I'll generate the full consolidated report across all programs step by step. Starting with programs 1 to 9 below 😊)

---

## 27. New Activity on Button Click

**Theory:** This program demonstrates switching between activities using explicit intents. A button click starts a new activity defined in AndroidManifest.xml. Each activity represents a separate screen. Using Intent, you define the destination activity. Use startActivity() to navigate. The button uses setOnClickListener. Both activities need layout files. Helps modularize UI. Important for multi-screen navigation. **Syntax:**

```
Intent intent = new Intent(MainActivity.this, SecondActivity.class);

startActivity(intent);
```

## 2. Two Fragments on One Screen

**Theory:** Fragments are reusable UI components within an activity. You can load two fragments using multiple containers in a layout. Use FragmentTransaction to attach them dynamically. Fragments have their own lifecycle. Useful for tablet layouts or dashboard-style screens. Each fragment has a layout file. You can communicate between fragments via activity. Best placed inside FrameLayout containers. Increases modularity. **Syntax:**

```
getSupportFragmentManager().beginTransaction()

    .add(R.id.container1, new FragmentOne())

    .add(R.id.container2, new FragmentTwo())

    .commit();
```

## 3. Linear Layout

**Theory:** LinearLayout arranges views in one direction (vertical or horizontal). Set direction with android:orientation. Supports weight distribution among children. Simple to use for basic Uis. Can nest layouts for complex designs. Each child is placed next to or below the previous. Layout params define sizes. Readable and maintainable layout. Efficient for small UI. **Syntax:**

```
<LinearLayout android:orientation="vertical"> ... </LinearLayout>
```

## 4. Relative Layout

**Theory:** RelativeLayout allows positioning elements relative to each other or parent. Uses properties like layout\_below, layout\_alignParentTop. Helps avoid deep nesting. Each child can reference another by ID. Reduces complexity for aligned Uis. You can overlap views. Commonly used for dynamic interfaces. Replaceable with ConstraintLayout in newer apps. **Syntax:**

```
<RelativeLayout>

    <TextView android:layout_below="@id/label" />

</RelativeLayout>
```

## 5. Absolute Layout

**Theory:** AbsoluteLayout positions items using fixed x and y coordinates. It is deprecated and not suitable for responsive designs. Not recommended in real apps. Still asked for legacy knowledge. Layout doesn't scale for different screen sizes. Only for static positioning. Avoid in modern projects. Better options are available. Was once used in older Android versions. **Syntax:**

```
<AbsoluteLayout>
```

```
    <TextView android:layout_x="50dp" android:layout_y="100dp"/>
```

```
</AbsoluteLayout>
```

## 6. ListView

**Theory:** ListView shows a scrollable list of items. Use ArrayAdapter to fill it with data. Each row is a view. Useful for displaying repeated content. You can use custom layouts. Supports click events via setOnClickListener. Replaced by RecyclerView in modern Android. Easy to implement for simple lists. **Syntax:**

```
ArrayAdapter adapter = new ArrayAdapter<>(this, android.R.layout.simple_list_item_1, items);
```

```
listView.setAdapter(adapter);
```

## 7. Table Layout

**Theory:** TableLayout arranges children into rows and columns. Each row is declared with <TableRow>. Good for forms and tables. Automatically adjusts column width. Doesn't draw borders. Use nested TextView/EditText inside rows. Easy to create registration/login forms. Simple but not flexible for complex dynamic data. **Syntax:**

```
<TableLayout>
```

```
    <TableRow>
```

```
        <TextView /> <EditText />
```

```
    </TableRow>
```

```
</TableLayout>
```

## 8. Grid Layout

**Theory:** GridLayout arranges views in rows and columns like a grid. Define number of rows/columns. Good for image grids, keypads, dashboards. Set rowCount and columnCount. Each view is placed in a grid cell. Supports rowSpan and columnSpan. Replaces TableLayout for more control. Suitable for static UI grids. **Syntax:**

```
<GridLayout android:rowCount="2" android:columnCount="2"> ... </GridLayout>
```

## 9. Frame Layout

**Theory:** FrameLayout is designed to block out an area on the screen to display a single item. Often used to swap fragments dynamically. Supports layering views. Commonly used as fragment containers. Only the top child is visible if overlap occurs. Layout simplicity makes it ideal for content replacement. **Syntax:**

```
<FrameLayout android:id="@+id/frameContainer" />
```

---

(Next update will continue with Programs 10–27.)

## Advanced Programming Laboratory-III: Practical Oral Exam Preparation

---

(Continued)

### 10. Implicit Intent

**Theory:**

Implicit intents do not name a specific component. They are used to invoke external apps (e.g., browser, dialer). The system resolves the best app to handle the request. Actions include ACTION\_VIEW, ACTION\_SEND, etc. Useful for sharing or opening links. No need to declare the target activity. You must add proper permissions if required. Always check if an app can handle the intent. Safer when launching external activities.

**Syntax:**

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://google.com"));
startActivity(intent);
```

### 11. Explicit Intent

**Theory:**

Explicit intents define the target activity by class name. Used for navigating between screens inside the same app. Must declare the target activity in the manifest. Uses startActivity() to perform the action. Suitable for well-defined internal navigation. Can pass data using putExtra(). Reliable and fast for intra-app communication. Doesn't require external resolution. Common in multi-activity apps.

**Syntax:**

```
Intent i = new Intent(this, SecondActivity.class);
startActivity(i);
```

### 12. Registration Form GUI

**Theory:**

A registration form collects user details like name, gender, and preferences. Uses UI elements like EditText, CheckBox, RadioButton, ToggleButton, and Button. Event listeners capture input actions. Input validation may be added. Button triggers data submission or display. All views are arranged using layouts. Each input field can be retrieved using getText(), isChecked(), etc. Use meaningful IDs.

**Syntax:**

```
button.setOnClickListener(v -> {
    String name = editText.getText().toString();
});
```

### 13. Spinner with Event Handler

**Theory:**

Spinner displays a dropdown list of items. You fill it using an ArrayAdapter. Listen to user selections using OnItemSelectedListener. Useful for country, city, or category selection. Spinner must be defined in XML. Selection is retrieved using getItem(). Allows dynamic or static data. Simple and efficient alternative to radio groups.

**Syntax:**

```
spinner.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    public void onItemClick(...) { ... }
});
```

## 14. Alert Dialog

### Theory:

AlertDialogs show a message with one or more buttons. Built using AlertDialog.Builder. Can be used for confirmation, messages, or warnings. Set title, message, and buttons like OK, Cancel. Event handling is done via button callbacks. Dialog is displayed using show(). Can be customized further with custom layouts. Easy to implement in any activity.

### Syntax:

```
new AlertDialog.Builder(this)
    .setTitle("Alert")
    .setMessage("Are you sure?")
    .setPositiveButton("Yes", null)
    .setNegativeButton("No", null)
    .show();
```

## 15. Options Menu

### Theory:

Options menu appears in the top-right corner. Define menu items in XML (res/menu). Inflate menu in onCreateOptionsMenu(). Handle item clicks in onOptionsItemSelected(). Useful for actions like settings, logout. Automatically integrated with Toolbar. Supports icons and grouping. Common across all Android apps.

### Syntax:

```
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.menu_main, menu);
    return true;
}
```

## 16. Context Menu

### Theory:

Context menu is triggered by long-pressing a view. Register views using registerForContextMenu(). Override onCreateContextMenu() to add items. Handle selections in onContextItemSelected(). Used for item-specific actions like delete, edit. Provides additional functionality without cluttering UI. Applicable to ListView, ImageView, etc.

### Syntax:

```
registerForContextMenu(view);
public void onCreateContextMenu(...) {
    menu.add("Delete");
}
```

## 17. Popup Menu

### Theory:

PopupMenu shows a floating menu anchored to a specific view. Similar to context menu, but explicitly triggered. Use PopupMenu class to create and show. Useful for dropdown-like menus. Define items in XML or programmatically. Handle clicks using setOnMenuItemClickListener(). Lightweight and easy to implement.

### Syntax:

```
PopupMenu popup = new PopupMenu(this, view);

popup.getMenuInflater().inflate(R.menu.popup_menu, popup.getMenu());

popup.show();
```

## 18. Notification

### Theory:

Notifications alert users outside the app context. Built using NotificationCompat.Builder. Must use NotificationChannel for Android 8+. Supports icons, actions, titles. Shown via NotificationManager. Requires proper permission and channel setup. Good for reminders, messages, or background updates.

### Syntax:

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, "channel_id")

    .setContentTitle("My Notification")

    .setSmallIcon(R.drawable.ic_notify);
```

## 19. Insert Data in SQLite

### Theory:

SQLite stores structured data locally. Use SQLiteDatabase.insert() or raw SQL. Create a helper class extending SQLiteOpenHelper. Insert data via ContentValues. Must define a table schema. SQLite is lightweight and built-in. Great for offline storage. Easy to query and manage.

### Syntax:

```
ContentValues values = new ContentValues();

values.put("name", "John");

db.insert("users", null, values);
```

## 20. Display Data from SQLite

### Theory:

To display data, use Cursor to read from database. Query table using.rawQuery() or query(). Loop through results using moveToFirst() and moveToNext(). Extract data using column index or name. Display in TextView or ListView. Close cursor after use. Useful for showing stored records.

### Syntax:

```
Cursor c = db.rawQuery("SELECT * FROM users", null);

if (c.moveToFirst()) {

    String name = c.getString(0);

}
```

## 21. Update Data in SQLite

### Theory:

Use `update()` to modify existing rows. Provide new values via `ContentValues`. Specify where clause to target specific row. Helpful for edit or modify features. Confirm rows affected using return value. Always validate data before update. Close database after operation.

### Syntax:

```
ContentValues cv = new ContentValues();  
cv.put("name", "Updated Name");  
db.update("users", cv, "id=?", new String[] {"1"});
```

## 22. Store Data in File

### Theory:

Use internal storage to save private app data. Use `FileOutputStream` to write data. Choose mode like `MODE_PRIVATE`. Write data as string or bytes. Close stream to avoid leaks. File is stored in app-specific directory. Simple way to persist settings or logs.

### Syntax:

```
FileOutputStream fos = openFileOutput("data.txt", MODE_PRIVATE);  
fos.write(data.getBytes());  
fos.close();
```

## 23. Multithreading

### Theory:

Multithreading prevents UI blocking by running tasks in background. Use `Thread`, `Runnable`, or modern `Executors`. Useful for network operations or file I/O. Start thread using `start()`. Can communicate back via handler or `runOnUiThread`. Avoid doing heavy work on main thread.

### Syntax:

```
Thread t = new Thread(() -> {  
    // Background task  
});  
t.start();
```

## 24. Progress Bar with Percentage

### Theory:

`ProgressBar` shows visual progress of a task. Can be determinate or indeterminate. Update progress via `setProgress()`. Show percentage using `TextView`. Useful for downloads or loading tasks. Linked to background task using handler or thread.

### Syntax:

```
progressBar.setProgress(value);  
textView.setText(value + "%");
```

## 25. Rating Bar

**Theory:**

RatingBar allows users to give feedback using stars. Listen to changes using `setOnRatingBarChangeListener`. Display value in `TextView`. Useful in reviews and surveys. Can set max rating and step size.

**Syntax:**

```
ratingBar.setOnRatingBarChangeListener((rb, rating, fromUser) -> {  
    textView.setText(String.valueOf(rating));  
});
```

**26. Seek Bar****Theory:**

SeekBar lets user choose a value on a slider. Useful for volume, brightness, etc. Listen using `OnSeekBarChangeListener`. Track progress and display it. Can set min, max, and step.

**Syntax:**

```
seekBar.setOnSeekBarChangeListener(new OnSeekBarChangeListener() {  
    public void onProgressChanged(...) {  
        textView.setText(String.valueOf(progress));  
    }  
});
```

**27. Toggle Button****Theory:**

`ToggleButton` acts as a switch between ON and OFF states. Listen using `OnCheckedChangeListener`. Show current state in `TextView`. Useful for settings like sound, notifications. Can be styled for better UI.

**Syntax:**

```
toggleButton.setOnCheckedChangeListener((buttonView, isChecked) -> {  
    textView.setText(isChecked ? "ON" : "OFF");  
});
```

---



# Chapter 1

## Unit wise Notes

Sure! Here's a concise and well-organized version of notes on the provided topics:

---

### Android Overview

- **Android:** Open-source mobile operating system developed by Google.
  - Based on **Linux kernel**.
  - Designed for touch screen devices (phones, tablets).
- 

### History of Android

- Founded by **Android Inc.** in 2003, acquired by **Google in 2005**.
- First commercial version: **Android 1.0 (2008)** on HTC Dream.

---

## Android Versions (Key Milestones)

- **Cupcake (1.5)** – First with on-screen keyboard.
- **Donut (1.6)** – Search and camera improvements.
- **Eclair (2.0)** – Google Maps, multi-touch.
- **Froyo (2.2)** – Flash support.
- **Gingerbread (2.3)** – UI refinement.
- **Honeycomb (3.x)** – Tablet support.
- **Ice Cream Sandwich (4.0)** – Phone & tablet UI merge.
- **Jelly Bean (4.1 - 4.3)** – Smooth UI.
- **KitKat (4.4)** – Optimized for lower-end devices.
- **Lollipop (5.0)** – Material design.
- **Marshmallow (6.0)** – App permissions.
- **Nougat (7.0)** – Multi-window support.
- **Oreo (8.0)** – Background limits.
- **Pie (9.0)** – Gesture navigation.
- **Android 10** – System-wide dark mode.
- **Android 11-14** – Enhanced privacy, UI tweaks, performance.

---

## Android OS Stack

1. **Linux Kernel** – Hardware abstraction, core services (security, memory, etc.)
2. **Native Libraries & DVM/ART**
  - Native C/C++ libraries: SQLite, WebKit, OpenGL, etc.
  - DVM (Dalvik Virtual Machine) / ART (Android Runtime)
3. **Application Framework** – Java APIs for app development.
4. **Applications** – Built-in and third-party apps.

---

## Core Components

- **Activity** – Single screen with a user interface.
- **Activity Back Stack** – Stack of activities; follows LIFO (Last In, First Out).

- **Process & Threads** – Each app runs in its own process; UI operations must run on the **main thread**.
- 

## Android Development Environment

- **Android SDK** – Tools & APIs for developing Android apps.
  - **Android Emulator** – Virtual device to run/test apps.
  - **Creating a Project** – Done using Android Studio or command-line tools.
  - **Project Directory Structure:**
    - /src – Java/Kotlin source code.
    - /res – Resources (layout, images).
    - /manifests – Manifest file.
    - /build – Compiled files.
- 

## Tools

- **DDMS (Dalvik Debug Monitor Server)** – Debugging tool for monitoring processes, threads, memory.
  - **Logcat** – Logging system for debugging; outputs messages using Log.d(), Log.e(), etc.
- 

## Android Manifest File

- Declares app components, permissions, themes, etc.
  - Mandatory elements: <application>, <activity>, <uses-permission>.
- 

## Permissions

- Required to access device features (e.g., camera, internet).
  - Declared in AndroidManifest.xml using <uses-permission>.
- 

Let me know if you'd like a printable PDF or visual diagram of this!

# Here's a **short and structured set of notes** for **Unit 2: Views and Layouts** in Android:

---

## XML Introduction

- **XML (Extensible Markup Language)**: Used to define UI layout in Android.
  - UI elements like TextViews, Buttons, etc., are declared in .xml files under /res/layout/.
- 

## Android View Hierarchies

- UI is structured in a **tree of Views** (ViewGroup → View).
  - **ViewGroup** is a container (e.g., LinearLayout).
  - **View** is a UI widget (e.g., Button, TextView).
- 

## Common Layouts

1. **LinearLayout**
  - Arranges child views in a **single row or column**.
  - Use orientation="vertical" or horizontal.
2. **RelativeLayout**
  - Views are **positioned relative** to each other or parent.
  - Flexible but more complex than LinearLayout.
3. **TableLayout**
  - Arranges views in **rows and columns** like a table.
  - Each row is a **TableRow**.
4. **FrameLayout**
  - Stacks views **on top of each other**.
  - Often used as a **placeholder for fragments**.
5. **ListView**
  - Displays a **scrollable list** of items.
  - Requires an **adapter** to bind data.
6. **Sliding Views**
  - Typically implemented using **ViewPager** or **RecyclerView with animations**.

- Allows swipe navigation between screens or items.

---

## Padding and Margins

- **Padding:** Space **inside** the view boundary.
- **Margin:** Space **outside** the view boundary.
- Units used: dp (density-independent pixels).

---

## Activity

- A single, focused screen the user interacts with.
- Declared in the **Manifest**.

---

## Activity Life Cycle

1. onCreate() – Initial setup.
2. onStart() – UI becomes visible.
3. onResume() – User interacts.
4. onPause() – Another activity is in focus.
5. onStop() – Activity is not visible.
6. onDestroy() – Cleanup.
7. onRestart() – After stopping, before starting again.

---

## Fragments

- Modular section of an activity UI.
- Allows **flexible UI** on different screen sizes (e.g., tablets).

---

## Fragment Life Cycle

1. onAttach()
2. onCreate()
3. onCreateView()
4. onActivityCreated()
5. onStart()

6. onResume()
  7. onPause()
  8. onStop()
  9. onDestroyView()
  10. onDestroy()
  11. onDetach()
- 

Let me know if you want visual diagrams or flashcards for easier revision!

## Here's a **short and clear summary of Unit 3: Intents** in Android:

---

### What is an Intent?

- **Intent** = Messaging object used to **request actions** from other app components.
  - Used for:
    - Starting activities
    - Starting services
    - Delivering broadcasts
- 

### Android Intent Messaging

- An **Intent Object** contains:
    - **Action** (e.g., ACTION\_VIEW)
    - **Data** (e.g., a URI)
    - **Component** (optional target)
    - **Extras** (key-value pairs of data)
- 

### Types of Intents

1. **Explicit Intent**
  - Specifies the **exact component** to start.

- Used **within your own app**.
- Example:
- `Intent i = new Intent(this, SecondActivity.class);`
- `startActivity(i);`

## 2. Implicit Intent

- Declares a **general action** to perform.
- System chooses the best component.
- Example:
- `Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse("http://google.com"));`
- `startActivity(i);`

---

## Using Intents with Activities

- `startActivity(intent)` – Starts a new activity.
- `startActivityForResult(intent, requestCode)` – Starts activity and waits for a result.

---

## Sending Intents for Telephony & SMS

- **Phone Call:**
- `Intent intent = new Intent(Intent.ACTION_CALL, Uri.parse("tel:123456789"));`
- `startActivity(intent);`
- **Send SMS:**
- `Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("sms:123456789"));`
- `intent.putExtra("sms_body", "Hello!");`
- `startActivity(intent);`

Note: Requires appropriate **permissions** in `AndroidManifest.xml`.

---

## Broadcast Receivers

- Components that respond to **broadcasted messages (events)**.
- Use `registerReceiver()` or declare in `AndroidManifest.xml`.
- Example: Listen for battery changes, SMS received, etc.

```
public class MyReceiver extends BroadcastReceiver {
```

```
public void onReceive(Context context, Intent intent) {  
    // Handle broadcast  
}  
}
```

---

Let me know if you'd like this turned into a one-page cheat sheet!

---

## Here's a short and structured summary for Unit 4: Input Controls, Input Events, Dialogs in Android:

---

### Input Controls

Used to **take user input** in forms and interactions.

- **Button** – Standard clickable button.
  - **TextField (EditText)** – For text input.
  - **Checkbox** – Allows **multiple selections**.
  - **RadioButton** – Allows **single selection** in a group (RadioGroup).
  - **ToggleButton / Switch** – On/Off toggle control.
  - **Spinner** – Dropdown menu for selecting one item.
- 

### Input Events

- Triggered when users **interact with input controls**.
- 

### Event Listeners & Handlers

1. **Listeners** – Detect specific events.
  - Examples:  
setOnClickListener(), setOnCheckedChangeListener()
2. **Handlers** – Define what happens when event occurs.

**Example:**



```
button.setOnClickListener(new View.OnClickListener() {  
    public void onClick(View v) {  
        // handle click  
    }  
});
```

---

## Touch Mode & Focus

- **Touch Mode:** User is interacting with the screen via touch.
  - **Handling Focus:**
    - Focus determines **which control is active**.
    - Use `requestFocus()`, `clearFocus()`.
- 

## Dialogs

Used for **interacting with the user** without changing activity.

1. **AlertDialog** – Simple dialog with title, message, and buttons.
  2. `new AlertDialog.Builder(this)`
  3. `.setTitle("Alert")`
  4. `.setMessage("Are you sure?")`
  5. `.setPositiveButton("Yes", null)`
  6. `.setNegativeButton("No", null)`
  7. `.show();`
  8. **RatingBar** – Lets user **rate using stars**.
  9. **ProgressBar** – Shows **loading** or ongoing task.
    - Can be **indeterminate** or **determinate**.
- 

## Other UI Feedback Components

- **PopupWindow** – Floating container over activity; custom layout.
  - **Toast** – Small **temporary message** at the bottom of the screen.
  - `Toast.makeText(getApplicationContext(), "Hello!", Toast.LENGTH_SHORT).show();`
-

Let me know if you want visual examples or code snippets for each control!

## Here's a **short and clear summary for Unit 5: Menus, Notifications, and Action Bar** in Android:

---

### Menus in Android

Used to offer **actions and options** to users.

### Types of Menus

#### 1. Options Menu

- Appears when the user presses the menu button or in the Action Bar.
- Defined in XML (res/menu/menu\_main.xml).

#### 2. @Override

3. `public boolean onCreateOptionsMenu(Menu menu) {`
4. `getMenuInflater().inflate(R.menu.menu_main, menu);`
5. `return true;`
6. `}`

#### 7. Context Menu

- Shown on **long press** on a view (e.g., ListView item).

8. `registerForContextMenu(view);`

#### 9. Popup Menu

- Anchored to a view (e.g., on clicking a button).

10. `PopupMenu popup = new PopupMenu(context, view);`
  11. `popup.getMenuInflater().inflate(R.menu.menu_example, popup.getMenu());`
  12. `popup.show();`
- 

### Handling Menu Click Events

- Handled using `onOptionsItemSelected(MenuItem item)`:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    if (item.getItemId() == R.id.action_settings) {
        // Handle click
        return true;
    }
    return super.onOptionsItemSelected(item);
}
```

---

## Notifications

Used to inform users about events **outside the app UI**.

### Creating a Notification

- Use `NotificationCompat.Builder` to build a notification.

```
NotificationCompat.Builder builder = new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.icon)
    .setContentTitle("Title")
    .setContentText("Message")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT);
```

### Notification Actions

- Add **buttons or actions** in the notification:

```
builder.addAction(R.drawable.icon, "Reply", pendingIntent);
```

### Notification Priority

- Controls how interruptive the notification is:
  - `PRIORITY_HIGH`, `PRIORITY_DEFAULT`, `PRIORITY_LOW`, etc.

### Managing Notifications

- Use **NotificationManager** to show or update:

```
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);
notificationManager.notify(notificationId, builder.build());
```

### Removing Notifications

- Use `cancel()` or `cancelAll()`:

```
notificationManager.cancel(notificationId);
```

---

Let me know if you'd like sample XML files or diagrams for menus and notifications!

## Here's a **short and well-organized summary for Unit 6: Android Storage and Database:**

---

### **Android Storage Options**

#### **1. Internal Storage**

- Private to the app.
- Stored in /data/data/<package-name>/files/.

#### **2. External Storage**

- Shared space (e.g., SD card).
  - Requires permissions.
- 

### **File Handling**

- **Write to File (Internal):**

```
FileOutputStream fos = openFileOutput("myfile.txt", MODE_PRIVATE);
```

```
fos.write(data.getBytes());
```

```
fos.close();
```

- **Read from File:**

```
FileInputStream fis = openFileInput("myfile.txt");
```

---

### **SQLite in Android**

- Lightweight **embedded database**.
  - Structured data storage using SQL.
-

## Installing SQLite Plugin (Optional for IDEs)

- Useful for browsing DB inside Android Studio.
  - Plugins like “**DB Browser for SQLite**” (external) can also be used.
- 

## DBHelper Class

- Extends SQLiteOpenHelper.
- Manages DB creation and version management.

### Example:

```
public class DBHelper extends SQLiteOpenHelper {  
    public DBHelper(Context context) {  
        super(context, "MyDB.db", null, 1);  
    }  
  
    public void onCreate(SQLiteDatabase db) {  
        db.execSQL("CREATE TABLE users(id INTEGER PRIMARY KEY, name TEXT)");  
    }  
  
    public void onUpgrade(SQLiteDatabase db, int oldVer, int newVer) {  
        db.execSQL("DROP TABLE IF EXISTS users");  
        onCreate(db);  
    }  
}
```

---

## Database Schema and Creation

- Define table structure in onCreate().
  - Example schema:
  - CREATE TABLE users(id INTEGER PRIMARY KEY, name TEXT);
- 

## Four Major DB Operations (CRUD)

### 1. Create (Insert)

2. ContentValues values = new ContentValues();
  3. values.put("name", "Alice");
  4. db.insert("users", null, values);
  5. **Read (Query)**
  6. Cursor cursor = db.rawQuery("SELECT \* FROM users", null);
  7. **Update**
  8. ContentValues values = new ContentValues();
  9. values.put("name", "Bob");
  10. db.update("users", values, "id=?", new String[]{"1"});
  11. **Delete**
  12. db.delete("users", "id=?", new String[]{"1"});
- 

## Cursors

- Used to **navigate query results**.
- Example:

```
while(cursor.moveToNext()) {  
    String name = cursor.getString(cursor.getColumnIndex("name"));  
}  
cursor.close();
```

---

## Example Use Case

1. Create DBHelper class.
  2. Add insert, read, update, delete functions.
  3. Use in an activity with buttons to perform actions.
  4. Display results using ListView or TextView.
- 

Let me know if you want a full code example or a practice assignment!

# Chapter 3

## **Programs**

Got it! I'll write out all 27 Android Java programs with clear steps and explanations, similar to the first one. This will be a bit lengthy, so I'll break it down carefully.

---

## 1. Create New Activity on Button Click

**Step 1:** Create two activities: MainActivity and SecondActivity.

**Step 2:** Design activity\_main.xml with a Button.

**Step 3:** Set button click listener to start SecondActivity.

**Step 4:** Declare SecondActivity in AndroidManifest.xml.

**Code:**

MainActivity.java

java

CopyEdit

```
package com.example.newactivity;
```

```
import android.content.Intent;
```

```
import android.os.Bundle;
```

```
import android.view.View;
```

```
import android.widget.Button;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
public class MainActivity extends AppCompatActivity {
```

```
    Button btnOpenActivity;
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```



```
setContentView(R.layout.activity_main);
```

```
btnOpenActivity = findViewById(R.id.btnOpenActivity);
```

```
btnOpenActivity.setOnClickListener(new View.OnClickListener() {
```

```
    @Override
```

```
    public void onClick(View v) {
```

```
        Intent intent = new Intent(MainActivity.this, SecondActivity.class);
```

```
        startActivity(intent);
```

```
    }
```

```
});
```

```
}
```

```
}
```

SecondActivity.java

java

CopyEdit

```
package com.example.newactivity;
```

```
import android.os.Bundle;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
public class SecondActivity extends AppCompatActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_second);
```

```
    }
```

```
}
```

activity\_main.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">

    <Button
        android:id="@+id/btnOpenActivity"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Open Second Activity" />
```

```
</LinearLayout>
```

activity\_second.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:padding="16dp">

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Welcome to Second Activity!"
        android:textSize="24sp"
        android:textStyle="bold" />
```

```
</LinearLayout>
```

AndroidManifest.xml snippet

xml

CopyEdit

```
<application ... >
```

```
    <activity android:name=".MainActivity" />
```

```
    <activity android:name=".SecondActivity" />
```

```
</application>
```

---

## 2. Create Two Fragments on Single Screen

**Step 1:** Create two fragments: FragmentOne and FragmentTwo.

**Step 2:** Design activity\_main.xml with two fragment containers side-by-side using LinearLayout with horizontal orientation.

**Step 3:** Inflate fragment layouts inside the fragment classes.

**Code:**

activity\_main.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="horizontal"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent">
```

```
    <fragment
```

```
        android:id="@+id/fragment1"
```

```
        android:name="com.example.fragments.FragmentOne"
```

```
        android:layout_width="0dp"
```

```
        android:layout_weight="1"
```

```
        android:layout_height="match_parent" />
```

```
<fragment
    android:id="@+id/fragment2"
    android:name="com.example.fragments.FragmentTwo"
    android:layout_width="0dp"
    android:layout_weight="1"
    android:layout_height="match_parent" />
</LinearLayout>
```

FragmentOne.java

java

CopyEdit

```
package com.example.fragments;
```

```
import android.os.Bundle;
```

```
import android.view.LayoutInflater;
```

```
import android.view.View;
```

```
import android.view.ViewGroup;
```

```
import androidx.fragment.app.Fragment;
```

```
public class FragmentOne extends Fragment {
```

```
    @Override
```

```
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
```

```
        return inflater.inflate(R.layout.fragment_one, container, false);
```

```
    }
```

```
}
```

FragmentTwo.java

java

CopyEdit

```
package com.example.fragments;
```

```
import android.os.Bundle;
```

```
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.fragment.app.Fragment;
```

```
public class FragmentTwo extends Fragment {
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        return inflater.inflate(R.layout.fragment_two, container, false);
    }
}
```

fragment\_one.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
```

```
    android:background="#FFCDD2"
```

```
    android:gravity="center"
```

```
    android:orientation="vertical">
```

```
    <TextView
```

```
        android:text="Fragment One"
```

```
        android:textSize="20sp"
```

```
        android:textColor="#000"
```

```
        android:layout_width="wrap_content"
```

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

```
</LinearLayout>
```

fragment\_two.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#BBDEFB"
    android:gravity="center"
    android:orientation="vertical">

    <TextView
        android:text="Fragment Two"
        android:textSize="20sp"
        android:textColor="#000"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

---

### 3. Application with Linear Layout

**Step:** Use a vertical LinearLayout containing TextView, EditText, and Button.

**activity\_linear.xml**

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp">
```

```
<TextView
    android:text="Enter Name:"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

<EditText
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />

<Button
    android:text="Submit"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
</LinearLayout>
```

---

#### 4. Application with Relative Layout

**Step:** Use a RelativeLayout with a Button centered.

##### activity\_relative.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:id="@+id/button1"
        android:text="Click Me"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:layout_centerInParent="true" />
</RelativeLayout>
```

---

## 5. Application with Absolute Layout (Deprecated)

**Note:** AbsoluteLayout is deprecated and not recommended. But here's a simple example.

### activity\_absolute.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <Button
        android:text="Old Layout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="50px"
        android:layout_y="100px" />
</AbsoluteLayout>
```

---

## 6. Application with ListView

**Step:** Create a ListView and populate it with an array using an ArrayAdapter.

### activity\_listview.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/listView"
    android:layout_width="match_parent"
```



```
android:layout_height="match_parent" />
```

### **ListViewActivity.java**

java

CopyEdit

```
package com.example.listview;
```

```
import android.os.Bundle;
```

```
import android.widget.ArrayAdapter;
```

```
import android.widget.ListView;
```

```
import androidx.appcompat.app.AppCompatActivity;
```

```
public class ListViewActivity extends AppCompatActivity {
```

```
    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
```

```
        setContentView(R.layout.activity_listview);
```

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

```
        String[] fruits = {"Apple", "Banana", "Cherry"};
```

```
        ArrayAdapter<String> adapter = new ArrayAdapter<>(this,  
        android.R.layout.simple_list_item_1, fruits);
```

```
        listView.setAdapter(adapter);
```

```
    }
```

```
}
```

---

## **7. Application with Table Layout**

**Step:** Use TableLayout with two rows and two columns.

### **activity\_table.xml**

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TableRow>
        <TextView
            android:text="Username:" />
        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content" />
    </TableRow>

    <TableRow>
        <TextView
            android:text="Password:" />
        <EditText
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:inputType="textPassword" />
    </TableRow>

</TableLayout>
```

---

## 8. Application with Grid Layout

**Step:** Use GridLayout with 2 columns containing buttons.

### activity\_grid.xml

xml

CopyEdit

```
<?xml version="1.0" encoding="utf-8"?>
```

```

<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    android:padding="16dp">

    <Button android:text="1" android:layout_width="0dp"
    android:layout_height="wrap_content" android:layout_columnWeight="1"/>

    <Button android:text="2" android:layout_width="0dp"
    android:layout_height="wrap_content" android:layout_columnWeight="1"/>

    <Button android:text="3" android:layout_width="0dp"
    android:layout_height="wrap_content" android:layout_columnWeight="1"/>

    <

```

#### **ChatGPT said:**

```

Button android:text="4" android:layout_width="0dp" android:layout_height="wrap_content"
    android:layout_columnWeight="1"/>
</GridLayout>

```

xml

CopyEdit

---

### **### 9. Application with Frame Layout**

**\*\*Step:\*\*** Use FrameLayout with overlapping TextViews.

**\*\*`activity\_frame.xml`\*\***

```

```xml

```

```

<?xml version="1.0" encoding="utf-8"?>

```

```

<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

```

```
android:padding="16dp">
```

```
<TextView
```

```
    android:text="Background Text"
```

```
    android:textSize="30sp"
```

```
    android:textColor="#DDDDDD"
```

```
    android:layout_gravity="center" />
```

```
<TextView
```

```
    android:text="Foreground Text"
```

```
    android:textSize="20sp"
```

```
    android:textColor="#000000"
```

```
    android:layout_gravity="center" />
```

```
</FrameLayout>
```

---

## **10. Program for Implicit Intent (Open a webpage)**

```
java
```

```
CopyEdit
```

```
Intent intent = new Intent(Intent.ACTION_VIEW);
```

```
intent.setData(Uri.parse("https://www.google.com"));
```

```
startActivity(intent);
```

---

## **11. Program for Explicit Intent (Navigate from one activity to another)**

```
java
```

```
CopyEdit
```

```
Intent intent = new Intent(CurrentActivity.this, TargetActivity.class);
```

```
startActivity(intent);
```

---

## **12. Registration Form GUI with Buttons, Text Fields, Checkboxes, Radio Buttons, Toggle Buttons, with Event Handlers**

```
activity_register.xml
```

xml

CopyEdit

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:padding="16dp"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <EditText android:id="@+id/etName" android:hint="Name"
    android:layout_width="match_parent" android:layout_height="wrap_content" />

    <CheckBox android:id="@+id/cbAgree" android:text="I agree"
    android:layout_width="wrap_content" android:layout_height="wrap_content"/>

    <RadioGroup android:id="@+id/rgGender" android:layout_width="match_parent"
    android:layout_height="wrap_content">

        <RadioButton android:id="@+id/rbMale" android:text="Male"
        android:layout_width="wrap_content" android:layout_height="wrap_content" />

        <RadioButton android:id="@+id/rbFemale" android:text="Female"
        android:layout_width="wrap_content" android:layout_height="wrap_content" />

    </RadioGroup>

    <ToggleButton android:id="@+id/toggleBtn" android:textOn="ON"
    android:textOff="OFF" android:layout_width="wrap_content"
    android:layout_height="wrap_content" />

    <Button android:id="@+id/btnSubmit" android:text="Submit"
    android:layout_width="wrap_content" android:layout_height="wrap_content" />

</LinearLayout>
```

**RegisterActivity.java**

java

CopyEdit

```
public class RegisterActivity extends AppCompatActivity {

    EditText etName;
```

```
CheckBox cbAgree;  
RadioGroup rgGender;  
ToggleButton toggleBtn;  
Button btnSubmit;
```

```
@Override
```

```
protected void onCreate(Bundle savedInstanceState) {
```

```
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_register);
```

```
    etName = findViewById(R.id.etName);  
    cbAgree = findViewById(R.id.cbAgree);  
    rgGender = findViewById(R.id.rgGender);  
    toggleBtn = findViewById(R.id.toggleBtn);  
    btnSubmit = findViewById(R.id.btnSubmit);
```

```
    btnSubmit.setOnClickListener(v -> {
```

```
        String name = etName.getText().toString();  
        boolean agreed = cbAgree.isChecked();  
        int genderId = rgGender.getCheckedRadioButtonId();  
        RadioButton selectedGender = findViewById(genderId);  
        String gender = selectedGender != null ? selectedGender.getText().toString() :
```

```
"None";
```

```
        String toggleState = toggleBtn.isChecked() ? "ON" : "OFF";
```

```
        Toast.makeText(this, "Name: "+name+
```

```
            "\nAgreed: "+agreed+
```

```
            "\nGender: "+gender+
```

```
            "\nToggle: "+toggleState, Toast.LENGTH_LONG).show();
```

```
    });
```

```
}
```

```
}
```

---

### 13. Spinner with Event Handler Displaying Selected Option

#### activity\_spinner.xml

xml

CopyEdit

```
<Spinner android:id="@+id/spinner" android:layout_width="match_parent"
android:layout_height="wrap_content" />
```

#### SpinnerActivity.java

java

CopyEdit

```
Spinner spinner = findViewById(R.id.spinner);

String[] items = {"Option 1", "Option 2", "Option 3"};

ArrayAdapter<String> adapter = new ArrayAdapter<>(this,
android.R.layout.simple_spinner_item, items);

adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
spinner.setAdapter(adapter);

spinner.setOnItemClickListener(new AdapterView.OnItemClickListener() {

    @Override

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

        String selected = parent.getItemAtPosition(position).toString();

        Toast.makeText(getApplicationContext(), "Selected: " + selected,
Toast.LENGTH_SHORT).show();

    }

    @Override

    public void onNothingSelected(AdapterView<?> parent) {}

});
```

---

### 14. Alert Dialog with Event Handler

java

CopyEdit

```
new AlertDialog.Builder(this)

    .setTitle("Alert")

    .setMessage("Do you want to continue?")

    .setPositiveButton("Yes", (dialog, which) -> {

        Toast.makeText(this, "Yes clicked", Toast.LENGTH_SHORT).show();

    })

    .setNegativeButton("No", (dialog, which) -> {

        Toast.makeText(this, "No clicked", Toast.LENGTH_SHORT).show();

    })

    .show();
```

---

## 15. Option Menu with Event Handler

res/menu/menu\_main.xml

xml

CopyEdit

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">

    <item android:id="@+id/item1" android:title="Option 1" />

    <item android:id="@+id/item2" android:title="Option 2" />

</menu>
```

### In Activity

java

CopyEdit

@Override

```
public boolean onCreateOptionsMenu(Menu menu) {

    getMenuInflater().inflate(R.menu.menu_main, menu);

    return true;

}
```

@Override



```
public boolean onOptionsItemSelected(MenuItem item) {  
    switch(item.getItemId()) {  
        case R.id.item1:  
            Toast.makeText(this, "Option 1 selected", Toast.LENGTH_SHORT).show();  
            return true;  
        case R.id.item2:  
            Toast.makeText(this, "Option 2 selected", Toast.LENGTH_SHORT).show();  
            return true;  
        default:  
            return super.onOptionsItemSelected(item);  
    }  
}
```

---

## **16. Context Menu with Event Handler**

### **In Activity**

java

CopyEdit

@Override

```
protected void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.activity_main);  
  
    View view = findViewById(R.id.viewForContextMenu);  
    registerForContextMenu(view);  
}
```

@Override

```
public void onCreateContextMenu(ContextMenu menu, View v,  
    ContextMenu.ContextMenuInfo menuInfo) {  
    super.onCreateContextMenu(menu, v, menuInfo);  
    menu.setHeaderTitle("Select Action");  
}
```

```

        menu.add(0, v.getId(), 0, "Edit");
        menu.add(0, v.getId(), 1, "Delete");
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        if(item.getTitle() == "Edit") {
            Toast.makeText(this, "Edit Selected", Toast.LENGTH_SHORT).show();
        } else if(item.getTitle() == "Delete") {
            Toast.makeText(this, "Delete Selected", Toast.LENGTH_SHORT).show();
        }
        return super.onOptionsItemSelected(item);
    }

```

---

## 17. Popup Menu with Event Handler

java

CopyEdit

```

Button btn = findViewById(R.id.button);
btn.setOnClickListener(v -> {
    PopupMenu popup = new PopupMenu(this, v);
    popup.getMenuInflater().inflate(R.menu.popup_menu, popup.getMenu());
    popup.setOnMenuItemClickListener(item -> {
        Toast.makeText(this, "Selected: " + item.getTitle(), Toast.LENGTH_SHORT).show();
        return true;
    });
    popup.show();
});

```

res/menu/popup\_menu.xml

xml

CopyEdit

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/pop1" android:title="Popup 1"/>
    <item android:id="@+id/pop2" android:title="Popup 2"/>
</menu>
```

---

## 18. Notification

java

CopyEdit

```
NotificationManagerCompat notificationManager = NotificationManagerCompat.from(this);
Notification notification = new NotificationCompat.Builder(this, "channel_id")
```

```
    .setSmallIcon(R.drawable.ic_notification)
    .setContentTitle("Notification Title")
    .setContentText("This is a notification example.")
    .setPriority(NotificationCompat.PRIORITY_DEFAULT)
    .build();
```

```
notificationManager.notify(1, notification);
```

*Make sure to create notification channel for API 26+.*

---

## 19. Insert Data into SQLite

java

CopyEdit

```
SQLiteDatabase db = openOrCreateDatabase("MyDB", MODE_PRIVATE, null);
db.execSQL("CREATE TABLE IF NOT EXISTS students(name VARCHAR, roll INT);");
db.execSQL("INSERT INTO students VALUES('Alice', 1);");
```

---

## 20. Display Data from SQLite

java

CopyEdit

```
Cursor cursor = db.rawQuery("SELECT * FROM students", null);
```

```
if(cursor.moveToFirst()) {
```

```
do {  
    String name = cursor.getString(0);  
    int roll = cursor.getInt(1);  
    Log.d("DB", "Name: " + name + ", Roll: " + roll);  
} while(cursor.moveToNext());  
}  
cursor.close();
```

---

## 21. Update Data in SQLite

```
java  
CopyEdit  
ContentValues cv = new ContentValues();  
cv.put("name", "Bob");  
db.update("students", cv, "roll=?", new String[]{"1"});
```

---

## 22. Store Data into File

```
java  
CopyEdit  
String data = "Hello file!";  
FileOutputStream fos = openFileOutput("file.txt", MODE_PRIVATE);  
fos.write(data.getBytes());  
fos.close();
```

---

## 23. Multithreading

```
java  
CopyEdit  
new Thread() -> {  
    // background task  
    runOnUiThread() -> Toast.makeText(this, "Background Task Done",  
    Toast.LENGTH_SHORT).show());  
}).start();
```

---

## 24. Progress Bar with Progress Percentage Display

activity\_progress.xml

xml

CopyEdit

```
<LinearLayout ...>
```

```
    <ProgressBar android:id="@+id/progressBar"
style="?android:attr/progressBarStyleHorizontal"

    android:layout_width="match_parent" android:layout_height="wrap_content"
android:max="100" />
```

```
    <TextView android:id="@+id/tvPercent" android:layout_width="wrap_content"
android:layout_height="wrap_content"/>
```

```
</LinearLayout>
```

In Activity:

java

CopyEdit

```
ProgressBar progressBar = findViewById(R.id.progressBar);
```

```
TextView tvPercent = findViewById(R.id.tvPercent);
```

```
progressBar.setProgress(70);
```

```
tvPercent.setText("70%");
```

---

## 25. Timer

java

CopyEdit

```
new CountDownTimer(10000, 1000) {
```

```
    public void onTick(long millisUntilFinished) {
```

```
        Log.d("TIMER", "Seconds remaining: " + millisUntilFinished / 1000);
```

```
    }
```

```
    public void onFinish() {
```

```
        Log.d("TIMER", "Done!");
```

```
}  
}.start();
```

---

## 26. Broadcast Receiver

java

CopyEdit

```
public class MyReceiver extends BroadcastReceiver {  
    @Override  
    public void onReceive(Context context, Intent intent) {  
        Toast.makeText(context, "Broadcast Received", Toast.LENGTH_SHORT).show();  
    }  
}
```

Register in Manifest or dynamically.

---

## 27. Service

java

CopyEdit

```
public class MyService extends Service {  
    @Nullable  
    @Override  
    public IBinder onBind(Intent intent) {  
        return null;  
    }  
    @Override  
    public int onStartCommand(Intent intent, int flags, int startId) {  
        Toast.makeText(this, "Service Started", Toast.LENGTH_SHORT).show();  
        return START_STICKY;  
    }  
}
```

Start service by:

java

CopyEdit

```
startService(new Intent(this, MyService.class));
```

---

If you want, I can help you generate any specific code or explanation in more detail for any of these 27. Would you like me to do that?

**You said:**

from 19 update with process in detailed

**ChatGPT said:**

Sure! Here's a **detailed guide** for programs starting from **19 to 27** with process, code, and explanations for each:

---

## **19. Insert Data into SQLite Database**

**Process:**

1. Create or open a database.
  2. Create a table if it doesn't exist.
  3. Insert data into the table.
- 

### **Step 1: Create Database Helper Class**

Create a class extending SQLiteOpenHelper to manage database creation and version management.

java

CopyEdit

```
public class DBHelper extends SQLiteOpenHelper {

    public static final String DATABASE_NAME = "StudentDB.db";
    public static final int DATABASE_VERSION = 1;

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

@Override

```
public void onCreate(SQLiteDatabase db) {
```

```
    String createTable = "CREATE TABLE students (id INTEGER PRIMARY KEY  
    AUTOINCREMENT, name TEXT, roll INTEGER)";
```

```
    db.execSQL(createTable);
```

```
}
```

@Override

```
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
```

```
    db.execSQL("DROP TABLE IF EXISTS students");
```

```
    onCreate(db);
```

```
}
```

```
}
```

---

## Step 2: Insert Data

java

CopyEdit

```
DBHelper dbHelper = new DBHelper(this);
```

```
SQLiteDatabase db = dbHelper.getWritableDatabase();
```

```
ContentValues values = new ContentValues();
```

```
values.put("name", "Alice");
```

```
values.put("roll", 1);
```

```
long newRowId = db.insert("students", null, values);
```

```
if(newRowId != -1) {
```

```
    Toast.makeText(this, "Data Inserted with ID: " + newRowId,  
    Toast.LENGTH_SHORT).show();
```

```
}
```

---



## 20. Display Data from SQLite Database

### Process:

1. Query the database.
  2. Retrieve cursor results.
  3. Display data using logs or UI components like ListView.
- 

### Sample Code to Display Data:

java

CopyEdit

```
SQLiteDatabase db = dbHelper.getReadableDatabase();

Cursor cursor = db.query("students", new String[]{"id", "name", "roll"},
    null, null, null, null, null);

if(cursor.moveToFirst()) {
    do {
        int id = cursor.getInt(cursor.getColumnIndexOrThrow("id"));
        String name = cursor.getString(cursor.getColumnIndexOrThrow("name"));
        int roll = cursor.getInt(cursor.getColumnIndexOrThrow("roll"));

        Log.d("DB_DATA", "ID: " + id + ", Name: " + name + ", Roll: " + roll);
    } while(cursor.moveToNext());
}

cursor.close();
```

To show in a ListView, you would use an Adapter with a cursor.

---

## 21. Update Data in SQLite Database

### Process:

1. Get writable database.
2. Use update() method with new values and a condition.

---

### Code to Update Record:

```
java
CopyEdit
SQLiteDatabase db = dbHelper.getWritableDatabase();

ContentValues values = new ContentValues();
values.put("name", "Bob");

int count = db.update("students", values, "roll = ?", new String[]{"1"});

Toast.makeText(this, "Rows updated: " + count, Toast.LENGTH_SHORT).show();
```

---

## 22. Store Data into File

### Process:

1. Open file output stream.
  2. Write string data to the file.
  3. Close the stream.
- 

### Code Example:

```
java
CopyEdit
String filename = "myfile.txt";
String data = "This is the data to write into file.";

try {
    FileOutputStream fos = openFileOutput(filename, MODE_PRIVATE);
    fos.write(data.getBytes());
    fos.close();
    Toast.makeText(this, "File saved successfully", Toast.LENGTH_SHORT).show();
}
```

```
} catch (IOException e) {  
    e.printStackTrace();  
    Toast.makeText(this, "Error saving file", Toast.LENGTH_SHORT).show();  
}
```

---

## 23. Multithreading Example

### Process:

1. Create a new Thread.
  2. Execute background tasks inside run() or lambda.
  3. Update UI using runOnUiThread().
- 

### Sample Code:

java

CopyEdit

```
new Thread(() -> {  
    // Background work (e.g. long calculation)  
    try {  
        Thread.sleep(3000); // simulate delay  
    } catch (InterruptedException e) {  
        e.printStackTrace();  
    }  
  
    runOnUiThread(() -> {  
        Toast.makeText(this, "Background task complete", Toast.LENGTH_SHORT).show();  
    });  
}).start();
```

---

## 24. Progress Bar with Percentage Display

### Process:

1. Add horizontal ProgressBar and TextView in XML.

## 2. Update progress and TextView in code.

---

### **activity\_progress.xml:**

xml

CopyEdit

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:padding="16dp"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:max="100" />

    <TextView
        android:id="@+id/tvPercent"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0%" />

</LinearLayout>
```

---

### **Java Code to Update Progress:**

java

CopyEdit

```
ProgressBar progressBar = findViewById(R.id.progressBar);
TextView tvPercent = findViewById(R.id.tvPercent);
```

```
int progress = 75; // Example progress
progressBar.setProgress(progress);
tvPercent.setText(progress + "%");
```

---

## 25. Rating Bar with Display of Selected Value

### Process:

1. Add RatingBar in XML.
  2. Set listener to display selected rating.
- 

### activity\_rating.xml:

xml

CopyEdit

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:padding="16dp"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <RatingBar
        android:id="@+id/ratingBar"
        android:numStars="5"
        android:stepSize="1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />

    <TextView
        android:id="@+id/tvRatingValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Rating: 0" />
</LinearLayout>
```

---

**Java Code:**

java

CopyEdit

```
RatingBar ratingBar = findViewById(R.id.ratingBar);
```

```
TextView tvRatingValue = findViewById(R.id.tvRatingValue);
```

```
ratingBar.setOnRatingBarChangeListener((ratingBar1, rating, fromUser) -> {  
    tvRatingValue.setText("Rating: " + (int) rating);  
});
```

---

## 26. SeekBar with Event Handler

**Process:**

1. Add SeekBar in XML.
  2. Listen to SeekBar change events.
  3. Display current progress.
- 

**activity\_seekbar.xml:**

xml

CopyEdit

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical" android:padding="16dp"  
    android:layout_width="match_parent" android:layout_height="match_parent">  
  
    <SeekBar  
        android:id="@+id/seekBar"  
        android:layout_width="match_parent"  
        android:layout_height="wrap_content"  
        android:max="100" />
```

```
<TextView
    android:id="@+id/tvSeekBarValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Progress: 0" />
</LinearLayout>
```

---

### Java Code:

java

CopyEdit

```
SeekBar seekBar = findViewById(R.id.seekBar);
TextView tvSeekBarValue = findViewById(R.id.tvSeekBarValue);

seekBar.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener() {
    @Override
    public void onProgressChanged(SeekBar seekBar, int progress, boolean fromUser) {
        tvSeekBarValue.setText("Progress: " + progress);
    }

    @Override
    public void onStartTrackingTouch(SeekBar seekBar) { }

    @Override
    public void onStopTrackingTouch(SeekBar seekBar) { }
});
```

---

## 27. Toggle Button with Event Handler

### Process:

1. Add ToggleButton in XML.
2. Listen to toggle changes.

3. Show toast or update UI accordingly.

---

**activity\_toggle.xml:**

xml

CopyEdit

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:padding="16dp"
    android:layout_width="match_parent" android:layout_height="match_parent">

    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOn="ON"
        android:textOff="OFF" />

</LinearLayout>
```

---

**Java Code:**

java

CopyEdit

```
ToggleButton toggleButton = findViewById(R.id.toggleButton);

toggleButton.setOnCheckedChangeListener((buttonView, isChecked) -> {
    String state = isChecked ? "ON" : "OFF";
    Toast.makeText(this, "Toggle is " + state, Toast.LENGTH_SHORT).show();
});
```

---

If you want, I can also help with XML layouts for any missing parts or full Activity code for any specific example. Just ask!