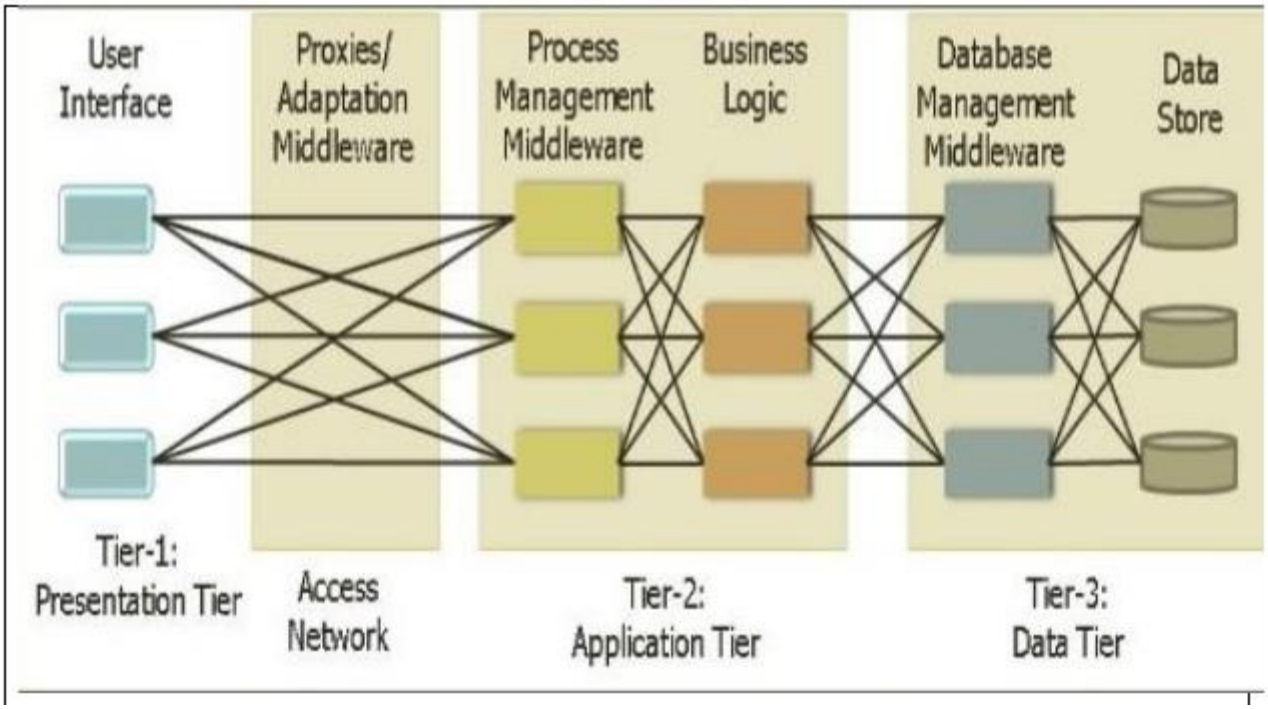
Unit 1 (Introduction to Mobile Development)

Mobile Computing Architecture



1. Presentation Layer (UI):

- Displays data to users and allows interaction.
- Requests data from the Business Logic Layer.
- Uses technologies like Dynamic HTML and client-side scripting.

2. Business Logic Layer:

- Acts as a server for client requests.
- Implements business rules and interacts with the Data Layer.
- Can function independently of the client, improving scalability.

3. Data Access Layer:

- Manages data storage and retrieval using a DBMS.
- Provides data to the Business Logic Layer.
- Reduces dependency on storage mechanisms, allowing flexibility.

Applications of 1G to 5G Networks

1G (First Generation)

- Analog voice communication.
- Used in early mobile phones (brick-sized).
- Limited coverage, poor security, and no data services.

2G (Second Generation)

- Digital voice communication with SMS & MMS.
- Basic internet access (GPRS, EDGE).
- Improved call quality and security.

3G (Third Generation)

- Faster internet, video calling, and mobile TV.
- Used in smartphones for web browsing and social media.
- Enabled mobile banking and GPS-based services.

4G (Fourth Generation)

- High-speed internet for HD video streaming and gaming.
- Supports VoLTE (better voice quality) and IoT devices.
- Used in ride-hailing apps, cloud computing, and smart homes.

5G (Fifth Generation)

- Ultra-fast speeds for AR/VR, AI, and smart cities.
- Low latency for autonomous vehicles and remote surgeries.
- Massive IoT connectivity for smart grids and industrial automation.

Handoff

Handoff is the process of transferring an ongoing call or data session from one cell tower (or base station) to another without interruption when a mobile user moves.

Types of Handoff:

1. Hard Handoff (Break-before-Make)

- Connection with the current tower is broken before switching to a new one.
- Used in 2G (GSM) and older systems.

2. Soft Handoff (Make-before-Break)

- Mobile connects to a new tower before disconnecting from the old one.
- Used in 3G (CDMA) and newer technologies for smoother transitions.

Roaming

1. Roaming is a service that allows mobile users to use their network services (calls, SMS, and data) outside their home network, either in a different region or country, by connecting to another operator's network.

2. It Ensures connectivity while traveling.

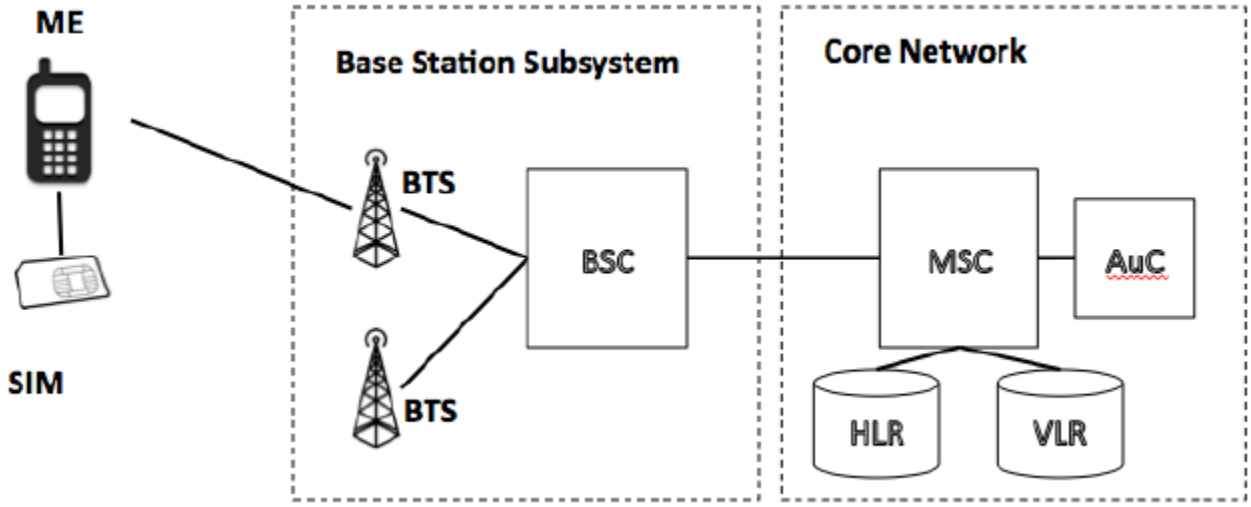
3. Allows seamless communication across different networks and countries.

4. The Charges may vary depending on agreements between operators.

Types of Roaming:

1. **National Roaming** – When a user moves outside their home network but within the same country (e.g., using another operator's network in remote areas).
2. **International Roaming** – When a user travels abroad and connects to a foreign network.
3. **Inter-Standard Roaming** – Switching between different network technologies (e.g., from LTE to 5G).

Global System for Mobile Communication(GSM) Architecture



GSM is a second-generation (2G) digital mobile network standard used for voice and data communication.

1. MS : MS stands for Mobile System. MS comprises user equipment and software needed for communication with a mobile network. Mobile System (MS) = Mobile Equipment(ME) + Subscriber Identity Module (SIM).

2. BTS : BTS stands for Base Transceiver Station which facilitates wireless communication between user equipment and a network.

3. BSC : BSC stands for Base Station Controller. You can consider the BSC as a local exchange of your area which has multiple towers and multiple towers have BTS.

4. MSC : MSC stands for Mobile Switching Center. MSC is associated with communication switching functions such as call setup, call release and routing, call tracing and call forwarding.

5. VLR : VLR stands for Visitor Location Register. VLR is a database which contains the exact location of all mobile subscribers currently present in the service area of MSC.

6. AUC : AUC stands for Authentication Center. AUC authenticates the mobile subscriber that wants to connect in the network.

Unit 2 (Introduction to Android)

What is Android

1. Android is an open-source operating system primarily used for mobile devices, such as smartphones and tablets.
2. It was developed by Android Inc., which was later acquired by Google in 2005.
3. Android is based on the Linux kernel and is designed to be highly customizable and flexible.
4. Android is open-source, meaning the underlying source code is available for anyone to use, modify, and distribute.
5. This allows device manufacturers (like Samsung, LG, etc.) to customize it to fit their hardware and brand.
6. Android has seen numerous updates and versions, each with improvements, new features, and bug fixes.
7. These are often named after desserts or sweets (e.g., Cupcake, Donut, Ice Cream Sandwich),
8. As of 2024, Android is the most widely used mobile operating system in the world, with over 70% of the global smartphone market share.
9. This means that more than 3 billion active Android devices are in use worldwide.

Android Architecture

Android architecture consists of four main layers:

1. **Applications**: This top layer includes pre-installed apps (e.g., home, contacts, camera) and third-party apps downloaded from the Play Store (e.g., chat apps, games).

2. **Application Framework**: Provides essential classes for creating Android apps, offering abstractions for hardware access and UI management.
3. **Android Runtime (ART)**: Contains core libraries and the Dalvik Virtual Machine (DVM), which powers Android apps. DVM is optimized for Android, allowing efficient multiple instance execution, similar to the Java Virtual Machine (JVM).
4. **Platform Libraries**: These include C/C++ and Java-based libraries, such as Media, Graphics, OpenGL, and Surface Manager, which support app development.
5. **Linux Kernel**: The core of Android, managing hardware drivers (e.g., display, camera, Bluetooth) and providing an abstraction layer between the hardware and other Android components. It handles memory, power, and device management.

This layered architecture ensures that Android devices can run efficiently, while providing a flexible platform for developers and users alike.

Steps to create new android project

To create a new Android project in **Android Studio**, follow these steps:

Step 1: Install Android Studio

1. **Download Android Studio** from the *official website*.
2. **Install** it by following the setup instructions for your operating system (Windows, macOS, or Linux).

Step 2: Start Android Studio

1. Open **Android Studio** after installation.
2. Click on **"Start a new Android Studio project"**.

Step 3: Configure Your Project

1. **Choose a Project Template**: Select a project template based on your app needs. For example:
    - **Empty Activity** (default template)
    - **Basic Activity** or **Navigation Drawer Activity** (if your app has more advanced UI needs).

2. Click **Next**.

Step 4: Set Up Project Details

1. **Name your app**: Choose a name for your application.
2. **Package Name**: This uniquely identifies your app on the Google Play Store (usually in reverse domain format, e.g., `com.example.myapp`).
3. **Save Location**: Choose the folder where your project will be saved.
4. **Language**: Select Java or Kotlin (Kotlin is the recommended language).
5. **Minimum API Level**: Choose the minimum Android version your app will support. It determines the lowest version of Android the app can run on.
6. Click **Finish**.

Step 5: Wait for Gradle Sync

Once the project is created, Android Studio will sync the project with **Gradle** (the build system). This might take a few moments.

Step 6: Start Coding

1. **MainActivity** will be created by default, and you can start coding in this file.
2. You can add your layout in the `res/layout/activity_main.xml` file.
3. Use **AndroidManifest.xml** to define your app's configuration and permissions.

Dalvik Virtual Machine

1. The **Dalvik Virtual Machine (DVM)** was a key component of Android's architecture, responsible for executing Android applications.
2. It is a register-based virtual machine designed specifically for mobile devices.
3. Unlike the Java Virtual Machine (JVM), which is stack-based, Dalvik uses a more memory-efficient approach suited for resource-constrained devices like smartphones.
4. When an Android app is compiled, its Java source code is first converted to Java bytecode, then transformed into **DEX format** to run on Dalvik.
5. DVM was built to allow efficient multitasking, enabling multiple app instances to run simultaneously without draining too much system memory or processing power. However, Dalvik was eventually replaced by **Android Runtime (ART)** in Android 5.0 (Lollipop).

6. In summary, Dalvik was a critical part of early Android systems, enabling apps to run on limited hardware resources.

How to install Android Studio

To install **Android Studio**, follow these steps based on your operating system (Windows, macOS, or Linux).

For Windows:

1. **Download Android Studio**:
   - Go to the official *Android Studio download page*.
   - Click the **"Download Android Studio"** button.
2. **Run the Installer**:
   - Once the download is complete, run the **.exe** file
   - Follow the installation prompts.
   - Accept the terms and choose the installation options (default is fine for most users).
3. **Install Android Studio**:
   - Click **Install** and wait for the installation to complete.
4. **Launch Android Studio**:
   - Once the installation is finished, launch **Android Studio** from the Start menu.
5. **Initial Setup**:
   - Android Studio will ask you to download additional components (SDKs, virtual device images, etc.).
   - Complete the setup by following the prompts to install necessary tools.

Types of Screen Orientation

1. **Portrait Orientation**:
   - The screen is taller than it is wide (vertical alignment). This is the default orientation for most mobile devices when held upright.
   - Example: Most apps like messaging or reading apps are used in portrait mode.
2. **Landscape Orientation**:

- The screen is wider than it is tall (horizontal alignment). This is typically used for watching videos, playing games, or using certain apps like photo editors.
- Example: Videos or games like racing games often use landscape mode.

Unit 3 (User Interface Screen Elements)

Toast

1. A Toast is a small, temporary notification message that appears on the screen to provide feedback to the user.

2. It is commonly used in Android and web applications to display short messages without requiring user interaction.

3. Toast messages automatically disappear after a short duration.

4. Following is an example

```
Toast.makeText(getApplicationContext(), "This is a Toast message!",
Toast.LENGTH_SHORT).show();
```

- getApplicationContext(): Context of the app.
- "This is a Toast message!": Message to display.
- Toast.LENGTH_SHORT: Duration of the toast message.
- .show(): Displays the toast.

Snackbar

1. A **Snackbar** is a UI element that provides a brief, non-intrusive message to users, similar to a **Toast**, but with additional functionality.

2. It usually appears at the bottom of the screen and can include an **action button** (e.g., "Undo").

3. Snackbar is commonly used in **Material Design** applications.

4. Following is an example:

```
Snackbar.make(findViewById(android.R.id.content), "Message deleted",
Snackbar.LENGTH_LONG)

        .setAction("Undo", new View.OnClickListener() {

            @Override

            public void onClick(View v) {

                // Action to undo delete

            }

        }).show();
```

Button

1. A Button is a a widget which can be pressed, or clicked, by the user to perform an some action.
2. Android buttons are GUI components which are sensible to taps (clicks) by the user.
3. When the user clicks on button in an Android app, the app respond to the clicks by executing suitable code written in the function onClick(View v).
4. There are different types of buttons used in android such as CompoundButton, ToggleButton,RadioButton.
5. We can perform action on button using different ways such as calling listener on button or adding onClick property of button in activity's xml file.

Button Attributes

Sr.No	Name	Description
1	**android:background**	This is a drawable to use as the background.
2	**android:contentDescription**	This defines text that briefly describes content of the view.
3	**android:id**	This supplies an identifier name for this view.
4	**android:onClick**	This is the name of the method in this View's context to invoke when the view is clicked.
5	**android:visibility**	This controls the initial visibility of the view.

Toggle Button

1. A **Toggle Button** allows users to switch between two states, such as ON/OFF or ENABLED/DISABLED.
2. It retains its state until the user changes it again.
3. Commonly used for settings like Wi-Fi, Bluetooth, or Dark Mode.
4. Can have different visual styles (text, icons, or color changes).
5. Provides immediate feedback when toggled.

Radio Button

1. A **Radio Button** is used when a user must select **only one option** from a group.
2. Once selected, clicking another option automatically deselects the previous one.
3. Commonly used in forms, surveys, and multiple-choice questions.

4. Grouped together to represent mutually exclusive choices.
5. Typically displayed as small circular buttons with labels.

Switch

1. A **Switch** is a UI component used to turn a setting ON or OFF.
2. It functions similarly to a **Toggle Button** but usually has a more interactive sliding design.
3. Commonly used for system settings like notifications, Wi-Fi, and sound.
4. Provides a **clear state change** (e.g., color change when switched ON).
5. Designed to be user-friendly with smooth transitions.

Image Button

1. An **Image Button** is a button that uses an **image or icon** instead of text.
2. Provides a more **visual and interactive experience** compared to text buttons.
3. Commonly used in apps for actions like "Play," "Pause," "Like," or "Share."
4. Can have different states (default, pressed, disabled).
5. Helps improve UI aesthetics and **reduces text clutter** in an interface.

TextView

1. A **TextView** is a UI component used to display **static text** in an application.
2. It **cannot be edited** by the user.

3. Often used for labels, instructions, or headings in an app.
4. Can support different styles like bold, italic, or colored text.
5. Can display **dynamic content** by updating text programmatically.

EditText

1. **EditText** is an input field where users can **enter or modify text**.
2. Commonly used for **forms, search bars, and login fields**.
3. Supports **different input types** like text, numbers, password, and email.
4. Can include **hints** to guide users on what to enter.
5. Can trigger events like validation or auto-suggestions based on input.

CheckBox

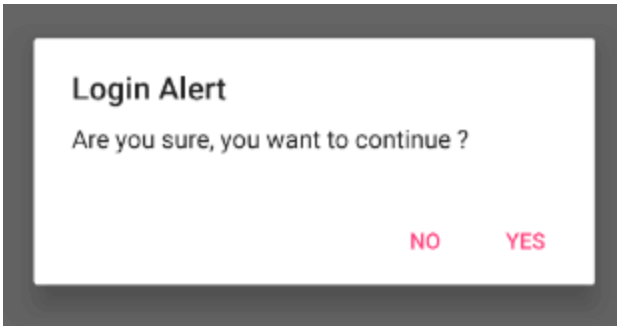
1. A **CheckBox** allows users to select **multiple** options from a list.
2. It can be **checked or unchecked**, representing a binary choice.
3. Commonly used in **forms, settings, and preference selections**.
4. Can be grouped together, but selections **are independent** (unlike radio buttons).
5. Provides a **visual tick mark** when selected, improving user experience.

Alert Dialog

1. An AlertDialog is a pop-up window that appears on top of the screen to grab user attention.

2. It is used for important alerts, confirmations, or user actions (e.g., "Delete this file?").
3. Typically includes a title, message, and action buttons (e.g., "OK" and "Cancel").
4. It blocks interaction with the rest of the app until dismissed.
5. Can include additional elements like checkboxes, input fields, or lists for user input.
6. Alert Dialog code has three methods:

- setTitle() method for displaying the Alert Dialog box Title.
- setMessage() method for displaying the message.
- setIcon() method is use to set the icon on Alert dialog box.



Bottom Sheet

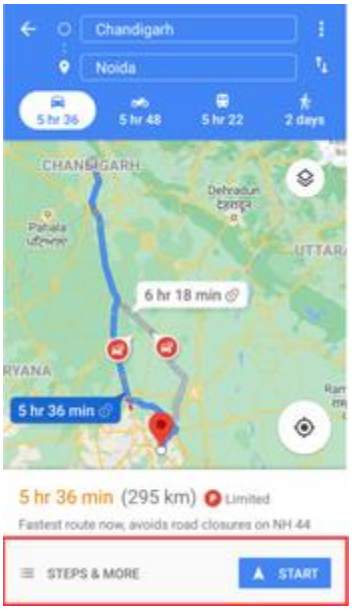
1. A **Bottom Sheet** is a UI panel that slides **up from the bottom** of the screen.
2. It is used to display **additional options, menus, or actions** without interrupting the main screen.
3. Can be **modal (blocks interaction)** or **persistent (always visible in part of the screen)**.
4. Commonly used for **sharing options, file selection, and additional settings**.

5. Provides a **smooth, gesture-friendly experience**, allowing users to **swipe down to dismiss**.

The two types of bottom sheets in mobile UI design are:

1. **Persistent Bottom Sheet**

- Remains visible on the screen even when the user interacts with other parts of the UI.
- Typically used for navigation, filters, or settings.
- Example: Google Maps displaying route options at the bottom.



2. Modal Bottom Sheet

- Appears temporarily and requires user interaction to dismiss.
- Used for displaying additional options or actions.

- Example: A share menu in apps like WhatsApp or Instagram

Spinner

1. A **Spinner** is a UI component used to select a value from a list of options.
2. It functions like a dropdown menu but takes up less space.
3. When clicked, it expands to show all available choices.
4. Commonly used for selecting items like dates, categories, or settings.
5. In **Android development**, `Spinner` is a built-in widget in XML and Java/Kotlin.
6. Spinners improve user experience by simplifying selection processes.
7. They can be **customized** with different styles, colors, and behaviors.
8. A **loading spinner** is different—it indicates background processing, like fetching data.
9. Spinners are widely used in forms, filters, and UI settings.

Date Picker & Time Picker

1. Date Picker

- A UI component for selecting a date from a calendar interface.
- Implemented using **DatePickerDialog** or **DatePicker** widget.
- Allows users to pick a day, month, and year.
- Supports different date formats like **DD/MM/YYYY** or **MM/DD/YYYY**.

- Commonly used in forms, event scheduling, and reminders.

2. Time Picker

- A UI component for selecting a specific time (hours & minutes).
- Implemented using **TimePickerDialog** or **TimePicker** widget.
- Supports both **12-hour (AM/PM)** and **24-hour format**.
- Used in alarm apps, meeting schedulers, and booking systems.

Attributes of Date Picker

Sr.No	Name	Description
1	**datePickerMode**	Value can be spinner or calendar. If set to calendar, it will display a calendar which let you choose date. If set to spinner, it will display a spinner to let you choose date.
2	**calendarViewShown**	This is a boolean value, only take effect when **datePickerMode** is spinner. If set to true, it will display a claendar.
3	**spinnersShown**	This is a boolean value also, only take effect when **datePickerMode** is spinner. If set to true, it will display a spinner.
4	**minDate**	Set the optional minimum date in mm/dd/yyy format.
5	**maxDate**	Set the maximum date to select, format is mm/dd/yyyy.
6	**startYear**	Set optional start year.
7	**endYear**	Set optional end year

Attributes of Time picker

Sr.No	Name	Description
1	**timePickerMode()**	Value can be spinner or clock. When set it' value to clock, it will display a clock. When set is's value to spinner will display a spinner.
2	**headerBackground**	This is a Color or Drawable resource id which can be set to TimePicker's header background.
3	**is24HourView()**	Check whether it is 24 hour time system.
4	**setIs24HourView(boolean 24HourView)** :	Set if use 24 hour time system.
5	**getHour()**	Get current hour integer value.
6	**getMinute()**	Get current minute integer value.
7	**setOnTimeChangedListener()** :	Set call back listener when time is changed.

Rating Bar

1. A UI component that allows users to give ratings using stars.
2. Implemented using the **RatingBar** widget in Android.
3. Supports fractional ratings (e.g., 3.5 stars).
4. Commonly used in review systems, feedback forms, and app ratings.
5. Can be customized with different star icons and colors.

```
<RatingBar
    android:id="@+id/ratingBar"
    android:layout_width="wrap_content"
```

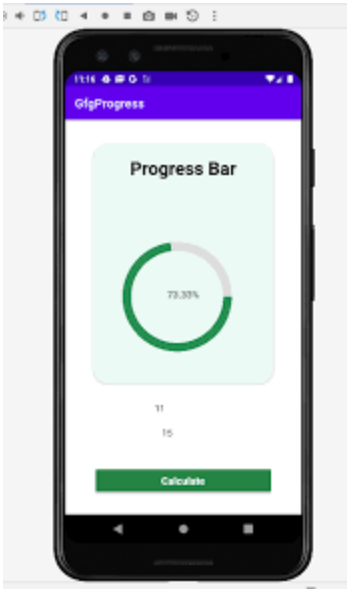
```
    android:layout_height="wrap_content"
    android:numStars="5"
    android:stepSize="0.5"
    android:rating="3.5"/>
```

Progress Bar

1. A UI component that visually represents progress.
2. Implemented using the **ProgressBar** widget in Android.
3. Two types:
    - **Determinate Progress Bar** (shows exact progress, e.g., file downloads).
    - **Indeterminate Progress Bar** (shows loading animation, e.g., loading screens).
4. Can be displayed as a **horizontal bar** or a **circular spinner**.

```
<ProgressBar
    android:id="@+id/progressBar"
    style="@android:style/Widget.ProgressBar.Horizontal"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:progress="50"
```

```
    android:max="100"/>
```



Unit 4 (Android Development Elements)

Android Terminologies

XML	In Android, XML file is used for designing the application's user interface like creating layouts, views, buttons, text fields etc. and it is used in parsing data feeds from the internet.
View	A view is an UI which occupies rectangular area on the screen to draw and handle user events.
Layout	Layout is the parent of view. It organizes all the views in a proper manner on the screen.
Activity	An activity is a device's screen which users see and interacts. User can place UI elements in any order in the created window of user's choice.
Emulator	An emulator is an Android virtual device through which you can select the target Android version or platform to run and test your own developed application.
Manifest file	It is a metadata file for every application. This file contains all the essential information about the application like app icon, app name, launcher activity. User can set all the permission like Send SMS, Bluetooth, Camera in this file.
Service	Services are the process which runs in the Background. It is not associated with any activity as there is no UI. Any other component of the application can start a service and this service will continue to run even when the user switches from one application to another.
Broadcast Receiver	Broadcast Receiver is another building block of Android application development which allows you to register for system and application events. It works in such a way that, when the event triggers for the first time all the registered receivers through this broadcast receiver will get notified for all the events by Android Runtime.

Content Providers	Content Providers are used to share data between two applications. This can be implemented in two ways: 1. When you want to implement the existing content provider in another application. 2. When you want to create a new content provider that can share its data with other applications
Intent	Intent is a message passing mechanism which is used to communicate between two or more components like services, activities , broadcast receiver etc. Intent can also be used to start an activity or service or to deliver broadcast messages.

Context in Android

-
-
-

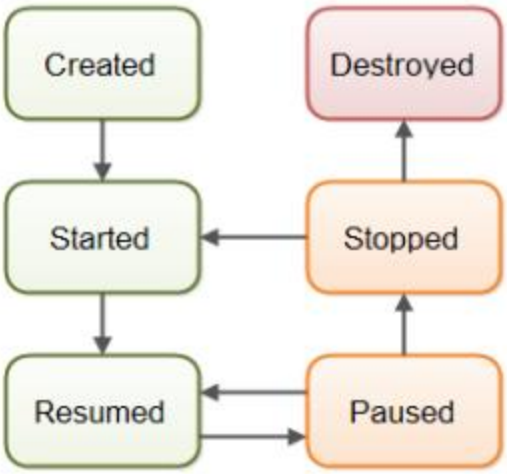
1.
    -
    -
    -
2.
    -
    -
    -

Activity in Android

-
-
-

-
-
-
-

Activity Lifecycle



1.
2.
3.
4.
5.
6.

Intent in Android

-
-

1.
    -
    -
2.
    -
    -

-
-
-

Linking Two Activities using Intent

-
-

Notification Service in Android

-
-
-
-

Broadcast in Android

-
-

1.
    -
        -
        -
2.
    -
    -

1.
    -
2.
    -

Adapter in Android

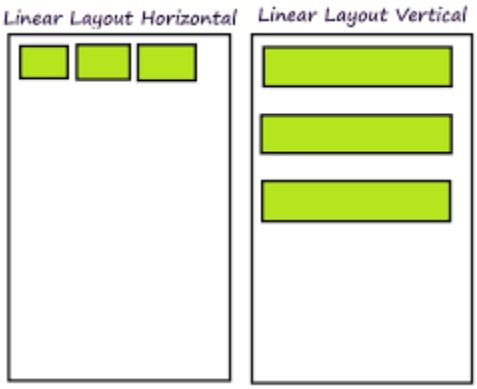
-
-

1.
    -
2.
    -
3.
    -
4.
    -

Unit 5 (Android Terminologies and Resource Handling)

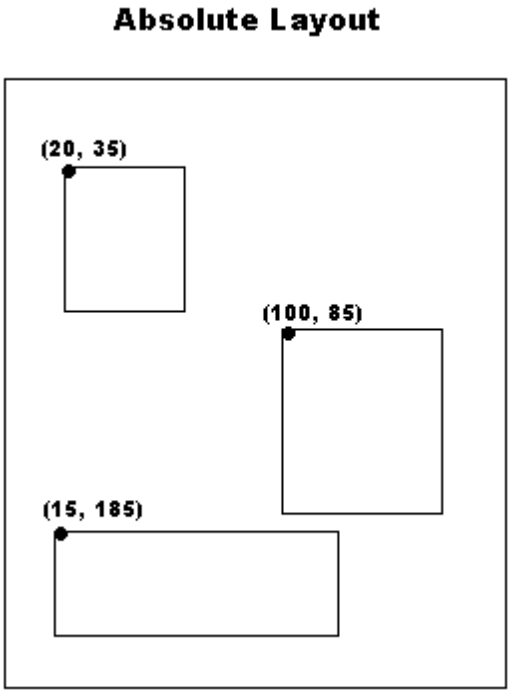
LinearLayout

1. Arranges child views in a single row (horizontal) or column (vertical).
2. Uses `android:orientation="horizontal"` or `android:orientation="vertical"`.
3. Supports `layout_weight` for distributing space among views.
4. Best for simple UI structures.
5. Nested `LinearLayout` can affect performance negatively.
6. Default alignment is **left for horizontal** and **top for vertical**.
7. Each child must specify `layout_width` and `layout_height`.
8. Supports gravity (`android:gravity`) for aligning child elements.
9. Can be combined with ScrollView for scrolling layouts.
10. Suitable for forms, login pages, and basic UIs.



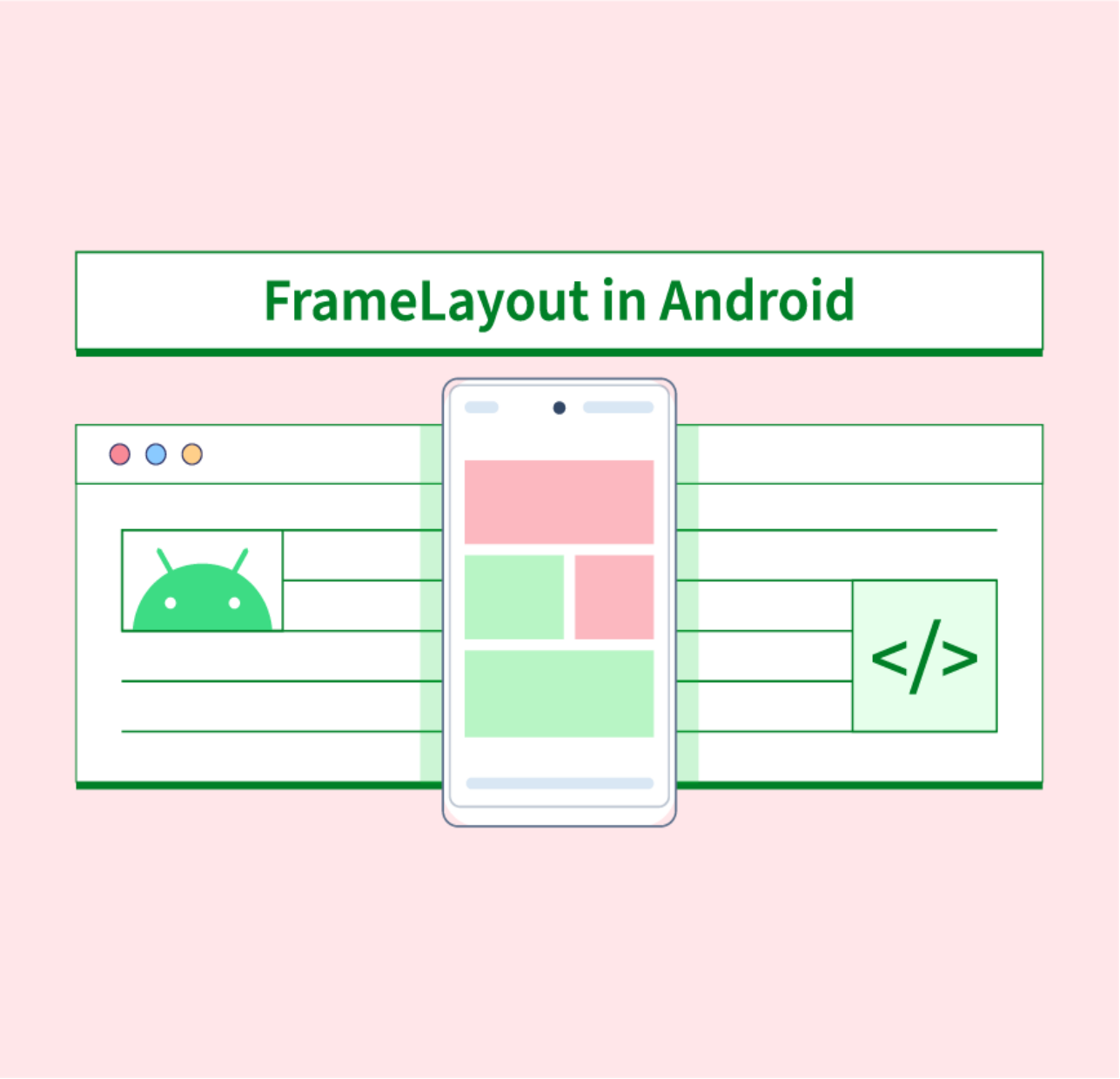
AbsoluteLayout

1. Positions views at absolute (x, y) coordinates.
2. Uses `android:layout_x` and `android:layout_y`.
3. Not responsive, so it was removed in later versions.
4. Difficult to maintain across different screen sizes.
5. Leads to UI issues on different devices.
6. No flexibility for dynamic content adjustment.
7. Consumes more memory compared to other layouts.
8. Not recommended for modern app development.
9. Replaced by **ConstraintLayout** and **RelativeLayout**.
10. Can be found in very old Android projects.



FrameLayout

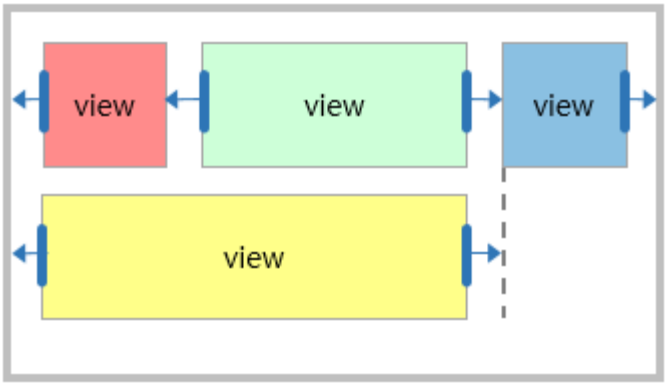
1. Stacks child views on top of each other.
2. Used for overlaying elements (e.g., image over text).
3. Best for displaying a single child or layered views.
4. Takes only the space required for its largest child.
5. Supports `layout_gravity` to position child views inside it.
6. Used in splash screens, modals, and dialogs.
7. Does not support margins between stacked views.
8. Ideal for fragment containers in Android apps.
9. Allows **animation effects** between views easily.
10. Commonly used in `BottomNavigationView` and `ViewPager`.



RelativeLayout

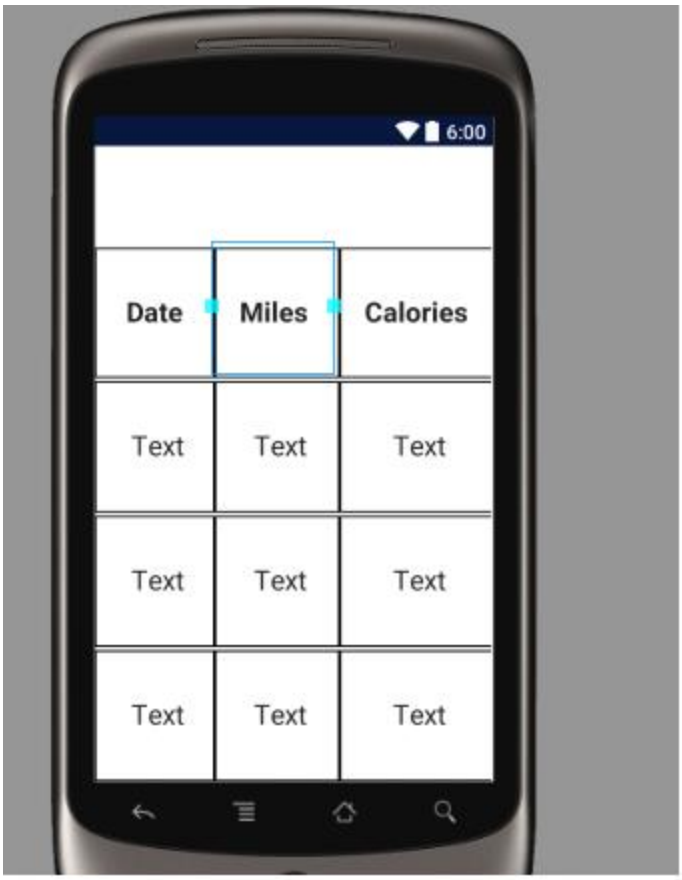
1. Positions views relative to each other or the parent.
2. Uses attributes like `layout_alignParentTop`, `layout_below`, etc.
3. More flexible than LinearLayout but complex for large UIs.
4. Reduces nested layouts, improving performance.
5. Supports margins, padding, and gravity settings.
6. Allows centering a view with `android:layout_centerInParent`.

7. Views can be aligned left, right, or below another view.
8. Can position views dynamically based on screen size.
9. Replaced by **ConstraintLayout** for better performance.
10. Used in complex UI designs like chat applications.



TableLayout

1. Arranges views in rows and columns like a table.
2. Uses `TableRow` for each row.
3. Best for form-based UI layouts.
4. Cells auto-adjust based on content size.
5. Does not display borders by default.
6. Supports spanning multiple columns using `android:layout_span`.
7. Can be difficult to style compared to other layouts.
8. Not recommended for complex UIs with dynamic data.
9. Works well for structured data like invoices, forms.
10. Alternative: **RecyclerView with GridLayoutManager** for flexibility.



ListView

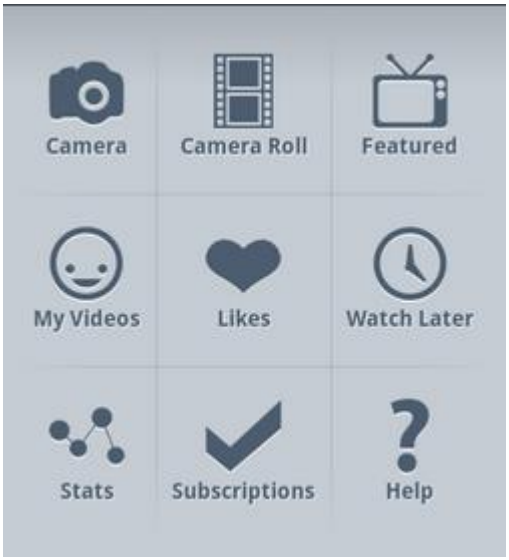
1. Displays a vertically scrollable list of items.
2. Uses an **Adapter** (e.g., `ArrayAdapter`, `BaseAdapter`).
3. Efficient for displaying large lists with `ViewHolder` pattern.
4. Supports item selection and click events using `setOnItemClickListener()`.
5. Can be customized using a **custom layout and adapter**.
6. Less efficient compared to `RecyclerView`.
7. Allows smooth scrolling with proper optimizations.
8. Can use `DividerItemDecoration` for separating items.
9. Works well for simple lists like contacts or messages.
10. Replaced by `RecyclerView` for better performance and flexibility.

GeeksforGeeks Tutorials

- Algorithms
- Data Structures
- Languages
- Interview Corner
- GATE
- ISRO CS
- UGC NET CS
- CS Subjects
- Web Technologies

GridView

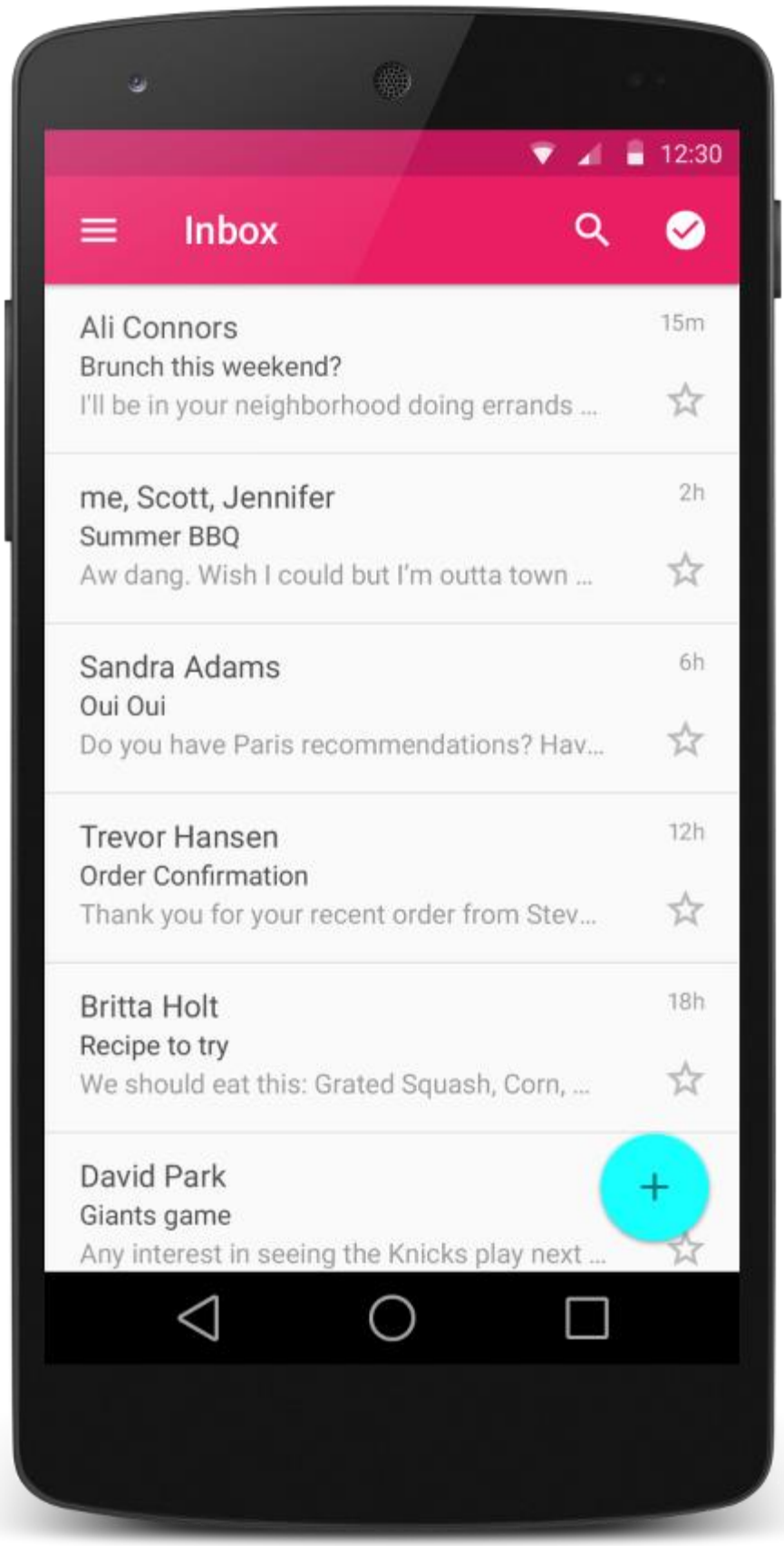
1. Displays items in a **grid format** (rows and columns).
2. Uses an **Adapter** to populate items dynamically.
3. Supports automatic column adjustment based on screen width.
4. Uses `android:numColumns="auto_fit"` for dynamic sizing.
5. Best for image galleries, app menus, or product listings.
6. Supports click events using `setOnItemClickListener()`.
7. Can be customized with a custom adapter for complex layouts.
8. Scrolls **vertically by default**, but can be nested in `ScrollView`.
9. Performance drops with large datasets (use `RecyclerView` instead).
10. `RecyclerView` with `GridLayoutManager` is the recommended alternative.



RecyclerView

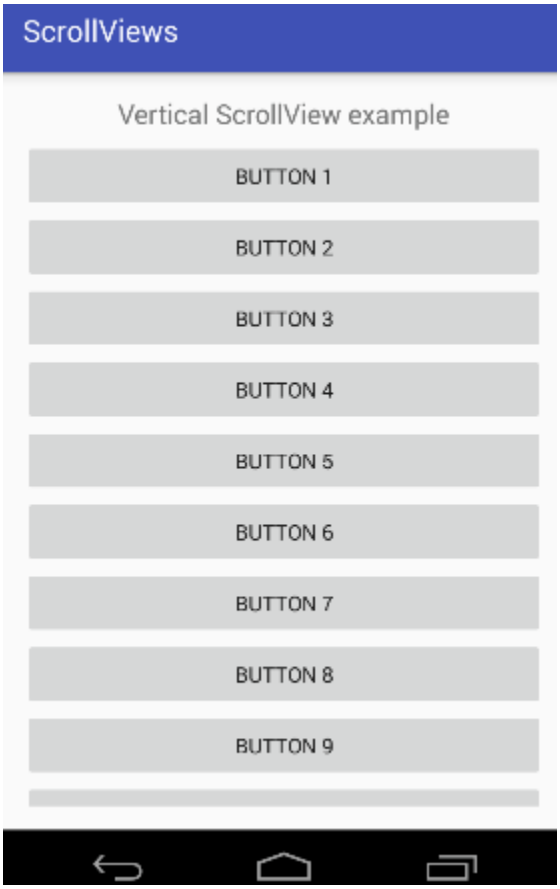
1. An advanced and efficient replacement for `ListView` and `GridView`.
2. Uses a **ViewHolder pattern** for improved performance.
3. Supports **linear, grid, and staggered layouts** with `LayoutManager`.
4. Can implement infinite scrolling and dynamic loading.
5. Supports swipe gestures and animations.

6. Works well with large datasets by reusing views efficiently.
7. Requires `RecyclerView.Adapter` and `RecyclerView.ViewHolder`.
8. Can be combined with `DiffUtil` for efficient updates.
9. Ideal for feeds, chat applications, and product listings.
10. More flexible than `ListView`, but requires more setup.


12:30
Inbox
Ali Connors
15m
Brunch this weekend?
I'll be in your neighborhood doing errands ...
me, Scott, Jennifer
2h
Summer BBQ
Aw dang. Wish I could but I'm outta town ...
Sandra Adams
6h
Oui Oui
Do you have Paris recommendations? Hav...
Trevor Hansen
12h
Order Confirmation
Thank you for your recent order from Stev...
Britta Holt
18h
Recipe to try
We should eat this: Grated Squash, Corn, ...
David Park
Giants game
Any interest in seeing the Knicks play next ...

ScrollView

1. Allows vertical scrolling for a **single child view**.
2. Wraps multiple views inside a **LinearLayout** (vertical).
3. Useful for displaying long content (e.g., forms, articles).
4. `android:fillViewport="true"` makes the content fill the screen.
5. Not suitable for lists (use `RecyclerView` instead).
6. Nested `ScrollView` can cause performance issues.
7. Supports both **vertical and horizontal scrolling** using `HorizontalScrollView`.
8. Can include buttons and interactive elements inside it.
9. Works well for static pages, but **not for dynamic lists**.
10. Can be combined with `NestedScrollView` for better scrolling behavior.



WebView

1. Used to display web pages inside an Android app.
2. Loads URLs using `webView.loadUrl("https://example.com")`.
3. Supports JavaScript with `webSettings.setJavaScriptEnabled(true)`.
4. Can be used to display **HTML content** from local or remote sources.
5. Supports file uploads and form handling.
6. Requires **internet permission** in `AndroidManifest.xml`.
7. Can interact with JavaScript via `WebViewClient`.
8. Supports **custom error handling** for failed page loads.
9. Can be used for embedding web apps inside Android apps.
10. Heavy on performance, so avoid excessive WebView usage in apps.
11. Example: When we click a link in Instagram, instead of opening the chrome browser it opens the webview inside the app.

Unit 6 (Android User Interface Elements)

File System in Android

The Android file system is based on a **Linux-based structure** and follows a hierarchical arrangement. It is divided into different partitions and directories, each serving specific purposes.

Key File System Partitions

1. **/system**
   - Stores the Android OS, system libraries, and default apps.
   - Read-only partition (modifiable only with root access).
2. **/data**
   - Contains app data, user settings, and databases.
   - Private to each app (isolated using Android's sandboxing).
3. **/cache**
   - Stores temporary files to speed up app processes.
4. **/sdcard (or /storage/emulated/0)**
   - Stores user-accessible files (photos, videos, downloads).
   - Used for external storage.
5. **/vendor**
   - Contains device-specific drivers and configurations.
6. **/boot**
   - Stores the kernel and bootloader, enabling the device to start.

Internal and External Storage in Android

Feature	Internal Storage	External Storage
Accessibility	Private to the app (sandboxed)	Accessible by multiple apps (r permission)
Persistence	Data is deleted when the app is uninstalled	Data remains even after app u
Security	More secure (app-private)	Less secure (accessible by oth permission is granted)
Usage	Storing app-specific files (e.g., settings, databases)	Storing media, documents, an
Performance	Faster (stored in device memory)	Slower (especially if on an SD
Capacity	Limited to internal storage size	Can be expanded using an SD
Example Use Cases	App preferences, login sessions, database storage	Photos, videos, downloads, an

---

Private Files in Android

1. **Definition**: Private files are files that are stored in an app's internal storage and are accessible only by that app.
2. **Security**: These files are sandboxed, meaning no other app or user can access them without root permissions.
3. **Types of Private Files**:
    - **App Configuration Files** (SharedPreferences)
    - **Databases** (SQLite)
    - **Cache Files** (Temporary storage)
    - **User-Specific Data** (Drafts, logs, etc.)
4. **Persistence**: Private files are deleted automatically when the app is uninstalled.

5. **Access & Management**:
    - Can be accessed using `getFilesDir()` or `getCacheDir()`.
    - Read/write operations do not require storage permissions.

CRUD in SQLite Database

CRUD stands for **Create, Read, Update, and Delete**, which are the four basic operations used in a database. In Android, SQLite is used for managing structured data, and CRUD operations allow apps to interact with the database.

1. Create (INSERT)

- Adds new records to the database.
- Example: Adding a new user to a "Users" table.

2. Read (SELECT)

- Retrieves data from the database.
- Example: Fetching all user records from the "Users" table.

3. Update (UPDATE)

- Modifies existing records in the database.
- Example: Changing the age of a user in the "Users" table.

4. Delete (DELETE)

- Removes records from the database.
- Example: Deleting a specific user from the "Users" table.

Cursor

1. A Cursor is a pointer that helps retrieve and navigate through data from an SQLite database.
2. It allows reading multiple rows of data from a database query.
3. Used in `SELECT` queries to fetch records from tables.
4. It can move between rows using methods like `moveToNext()`, `moveToFirst()`, and `moveToPrevious()`.
5. Data is accessed using methods like `getString()`, `getInt()`, and `getFloat()`.
6. Always close the cursor (`cursor.close()`) after use to free memory.

7. Helps in fetching user details or other records from a database table.
8. Cursors handle large datasets efficiently without loading everything into memory at once.
9. This makes database operations smoother in Android apps.

ContentValues

1. ContentValues is a key-value storage structure used to insert or update data in an SQLite database.
2. It stores column names as keys and their corresponding values as data.
3. Used with `insert()` and `update()` methods to modify database records.
4. Supports various data types like integers, strings, and booleans.
5. Helps in adding or updating records without writing complex SQL queries.
6. Data is added using the `put(key, value)` method.
7. Simplifies database operations by allowing structured data storage.
8. Frequently used for storing user details, settings, or app-related data.
9. This makes database management easier and more efficient in Android apps.

Advantages and Disadvantages of SQLite Database

Advantages	Disadvantages
Lightweight – Does not require a separate server.	**Not Suitable for Large-Scale Application** for small to medium-sized apps.
No Setup Required – Comes pre-installed with Android.	**Limited Concurrent Access** – Not ideal fo environments.
Fast and Efficient – Operates with minimal overhead.	**Lacks Advanced Features** – No built-in us or stored procedures.
ACID Compliant – Ensures data integrity and reliability.	**Limited Security** – No built-in encryption party support).
Uses a Single File – Database is stored in a single file, making it easy to manage.	**Write Operations Can Be Slow** – Not as f database servers.
Supports SQL – Allows standard SQL queries for data manipulation.	**Not Scalable** – Not suitable for high-volu apps.

Steps to Publish an Android App on the Google Play Store

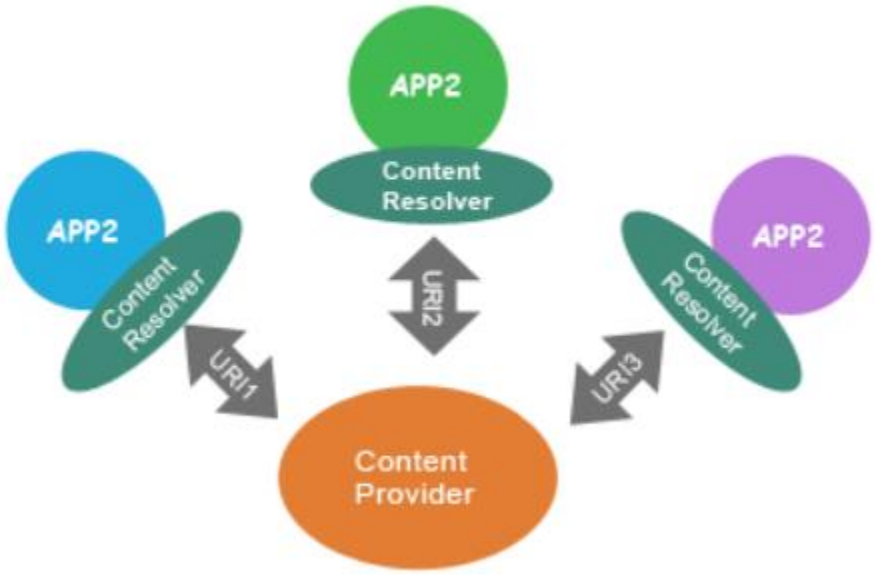
1. **Create a Google Play Developer Account**
   - Sign up at *Google Play Console*.
   - Pay a one-time registration fee of ₹2500.
2. **Prepare Your App**
   - Ensure your app follows Google Play policies.
   - Test the app for bugs and crashes.
3. **Generate a Signed APK or AAB**
   - Use Android Studio to create a signed APK or AAB.
   - Sign it with a keystore for security.

4. **Create a New App in Play Console**
    - Click **"Create App"** and enter app details like name, category, and language.
5. **Complete Store Listing**
    - Add a description, app icon, screenshots, and a feature graphic.
6. **Set Up Pricing and Distribution**
    - Choose **Free or Paid** and select target countries.
7. **Fill in the Content Rating Questionnaire**
    - Helps Google assign an age rating to your app.
8. **Upload App Bundle (AAB) or APK**
    - Navigate to the **"App Releases"** section and upload your signed file.
9. **Review and Submit for Approval**
    - Ensure all sections are filled correctly.
    - Click **“Submit for Review”** and wait for Google’s approval.
    - Once approved, your app will be live on the Play Store!

Unit 7 (Providers)

Content Providers in Android

1. Content Providers manage access to a structured set of data and enable data sharing between apps.
2. They act as an interface to interact with a database or other data sources.
3. Used to **read, write, update, and delete** data from one app to another securely.
4. Apps use **ContentResolver** to request data from a Content Provider.
5. Data is accessed using **URIs (Uniform Resource Identifiers)**, like `content://contacts/people/1`.
6. Commonly used Content Providers:
    - **ContactsProvider** (Access contacts)
    - **MediaStore** (Access images, videos, and audio)
    - **CalendarProvider** (Access calendar events)
7. Custom Content Providers can be created for an app's private database.
8. Essential for enabling **data sharing between apps** while maintaining security and access control.



Content Resolver in Android

1. Content Resolver is a component that allows an app to interact with a **Content Provider**.
2. It acts as a bridge between an app and the data stored in another app or database.
3. Used to perform **CRUD (Create, Read, Update, Delete) operations** on data.
4. Accesses data using **URIs (Uniform Resource Identifiers)** like `content://contacts/people/1`.
5. Methods provided by Content Resolver:
    - `query()` – Reads data from a Content Provider.
    - `insert()` – Adds new data.
    - `update()` – Modifies existing data.
    - `delete()` – Removes data.
6. Example Use Case: Fetching contacts, images, or messages stored on the device.
7. Ensures **secure and structured** access to shared data across apps.

Differences Between Content Resolver and Content Provider

Feature	Content Provider	Content Resolver
Purpose	Manages and provides access to app data	Requests data from a Content Provider
Role	Acts as a **data source**	Acts as a **client** that interacts with the provider
Operations	Defines how data is stored and shared	Performs CRUD operations (query, insert, update, delete)
Usage	Used by apps that share their data	Used by apps that need to access data from another app
Example	ContactsProvider stores contact details	A messaging app uses Content Resolver to access contacts

---

Steps to Create a Content Provider in Android

1. **Extend ContentProvider Class**: Create a new class that inherits from `ContentProvider` to manage data sharing.
2. **Define a Unique URI**: Set up a **Uniform Resource Identifier (URI)** to identify and access the content provider.
3. **Implement CRUD Methods**: Override methods like `query()`, `insert()`, `update()`, and `delete()` to handle data operations.
4. **Register in Manifest**: Declare the content provider in the `AndroidManifest.xml` file with the required authority.
5. **Access Data Using Content Resolver**: Other apps use `ContentResolver` to request and modify data through the provider.

Built-in Content providers

Provider	Purpose
AlarmClock	Set Alarm within the alarm clock application
Browser	Browser history and Bookmarks
CalenderContract	Calender and event Information
CallLog	Sent and received calls
ContactsContract	Phone Contact database or phonebook
MediaStore	Audio/Visual data on the phone and external storage
SearchRecentSuggestions	Create search suggestions appropriate to the application
Settings	Systemwide Device settings and preferences
UserDictonary	A dictionary of user-defined words for use with predictive text input
VoicemailContract	A Single unified place for the user to manage voice mail content from different sources

---

Unit 8 (Receivers)

Broadcast Receiver in Android

1. A **Broadcast Receiver** is a component that listens for system-wide or app-specific broadcast messages.
2. It responds to events like battery low, incoming calls, network changes, or custom app notifications.
3. Works in the **background** and does not have a user interface.
4. Uses **Intents** to receive and process broadcast messages.
5. Can be **declared in the manifest** (static) or **registered at runtime** (dynamic).
6. Common examples:
    - `BOOT_COMPLETED` – Trigger an action when the device starts.
    - `BATTERY_LOW` – Notify the user about low battery.
    - `CONNECTIVITY_CHANGE` – Detect internet connection changes.
7. Helps apps react to **system events efficiently** without running in the background continuously.

Implementing a Broadcast Receiver in Android

1. Create a Broadcast Receiver Class

java Copy Edit

```java
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.widget.Toast;

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

2. Register in `AndroidManifest.xml` (Static Registration)

xml Copy Edit

```xml
<receiver android:name=".MyBroadcastReceiver">
    <intent-filter>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"/>
    </intent-filter>
</receiver>
```