1. What is Mobile App?

A mobile application commonly referred to as a mobile app, is a software application designed to run on mobile devices such as smart phones and tablets. These applications are developed specially for mobile platforms and are typically downloaded and installed from app stores or marketplaces like the Apple App store for IOS devices and Google Play store for Android devices.

Example: An example of Mobile Application is Instagram, a popular app for sharing photos and videos with friends and followers. Users can upload images and videos, apply filters and share them on their profile or with a selected audience.

- **Apps are divided into two broad categories:**

Native apps and web apps

Native App/ Mobile App:

- **A Native App** is a type of **mobile app** that is built specifically for a particular platform, such as iOS for iPhones/iPads, Android for Android devices, or Windows for Windows-based devices**.**
- **Native apps** are built for specific platforms like **iOS (Swift, Objective-C)** and **Android (Kotlin, Java)**.
- **Native apps** seamlessly integrate with device hardware, accessing features like **camera, GPS, accelerometer, and notifications** for better performance.
- **Uber**: Uses **GPS** to track location, **notifications** for ride updates, and **camera** for driver verification, ensuring a smooth ride experience, **accelerometer** for movement detection.
- When you want to get a native app for your device, you usually go to the app stores (e.g., Apple App Store, Google Play Store), making them easily find and download the apps.
- Native apps often provide better performance and responsiveness compared to web apps, as they are optimized for the specific platform they are built for.
- **Examples** of popular native apps include Instagram, WhatsApp, and Pokémon GO, Spotify, Google maps,.

Web App:

- Web apps are accessed through a web browser and do not need to be downloaded or installed on a device.
- They are developed using web technologies such as HTML, CSS, and JavaScript, and **Web apps** are **platform-independent**; they work on **any device or operating system** with a web browser, without requiring separate versions.
- They can be easily updated, as changes made to the web app are immediately reflected for all users accessing it through the web.

There are several types of apps currently available.

- **Gaming** – Candy Crush, PUBG Mobile
- **Social Media** – Facebook, Instagram
- **Video Streaming** – Netflix, YouTube
- **Music Streaming** – Spotify, Apple Music

2. Mobile Application Development:

Mobile Application development is the process of creating Software application that run on Mobile Devices like Smartphones and Tablets.

Stages of Mobile Application Development:

1. **Planning** – Define app purpose, features, and target users. Analyze market trends and competitors.
2. **Design** – Create wireframes, UI/UX design, and navigation flow for a smooth user experience.
3. **Development** – Write a code using languages like Java Swift, or Kotlin. Use tools like Android studio or Xcode for frontend and Backend development.
4. **Testing:** – Identify and fix bugs, optimize performance, ensure security, and check compatibility on different Devices.
5. **Deployment:** Publish app to app stores like Apple App Store or Google Play Store following platform guidelines.
6. **Maintenance and Updates:** Monitor app performance, gather user feedback, and release updates for improvements, and new features.

3. What is Mobile technology?

- The term "mobile" comes from Latin, meaning "ability to move" or "portable." Mobile technology means technology that is portable, allowing wireless communication and data transfer anywhere using mobile devices like phones and tablets.
- Includes two-way communication devices, computing gadgets, and networking technologies. Enables users to perform tasks from anywhere, adding flexibility and convenience**.** A **smartphone** (computing gadget) uses **4G/5G networks** (networking technology) to make **voice and video calls** (two-way communication).

➢ **Mobile Technology Evolution**

Global Connectivity: Enables voice, text, image, and video sharing across the world.
Beyond Communication: Offers internet access, GPS navigation, gaming, streaming, and augmented reality.
Early Beginnings: Started with limited radio frequencies and two-way communication.
User Adoption: Evolved from two-way radios to modern smart phones and tablets.
Smartphone Revolution: Rapid growth and widespread adoption transformed global communication.

4. History of Mobile Technologies

Mobile technology has evolved over time, improving communication, speed, and connectivity. Below is a simple overview of mobile networks from 1G to 5G:

1G (First Generation)

- Introduced in the 1980s, 1G was the first analog cellular network that enabled basic voice communication. It used AMPS (Advanced Mobile Phone System) and FDMA (Frequency Division Multiple Access) for signal transmission. The phones were large and bulky, with poor sound quality, limited capacity, and no roaming. Despite these drawbacks, 1G set the stage for future mobile technologies.

2G (Second Generation)

- Introduced in the early 1990s, 2G was the first digital cellular network, offering improved sound quality, better security, and higher capacity. Key technologies included GSM (Global System for Mobile Communication) and CDMA (Code Division Multiple Access), which improved network efficiency and encryption. This generation introduced SMS and MMS, roaming, and basic data services, with GPRS (General Packet Radio Service) later enhancing mobile internet access and enabling multimedia messaging.

3G (Third Generation)

- Introduced in the early 2000s, 3G became a global standard, focusing on high-speed data applications. It significantly improved data transfer rates, enabling mobile internet access, video calling, and faster data services. Key technologies included UMTS, CDMA and EDGE, offering speeds of 384 kbps and higher. With 3G networks, users could browse the web; download data faster, and stream videos, marking a major shift towards mobile broadband connectivity.

4G (Fourth Generation)

- Introduced in the late 2000s, 4G brought significant improvements in data speeds, enabling seamless video streaming, online gaming, and faster mobile internet. It operates on packet-switching technology for efficient data transmission. Key technologies include LTE (Long-Term Evolution) for high-speed connectivity and WiMAX (Worldwide Interoperability for Microwave Access) as an alternative in some regions. With 4G, users experienced lower latency, better network efficiency, and enhanced multimedia capabilities.

5G (Fifth Generation)

- The latest generation of mobile technology, 5G, promises even faster data speeds, lower latency, and increased network capacity. It facilitates the development of new applications like augmented reality (AR), virtual reality (VR), and the Internet of Things (IoT). 5Gis reported to be up to 100 times faster at sending and receiving signals than 4G.

5. Different Mobile Technologies:

Mobile communication technologies have evolved significantly over the decades, starting from simple voice calls to high-speed internet and advanced multimedia capabilities.

5.1 Cellular Networks:

Mobile communication evolved from **1G** (1980s) using analog signals for voice (2.4 Kbps) to **2G** (1990s) with digital signals (GSM, CDMA) supporting SMS, MMS (64 Kbps), and **GPRS/EDGE** for internet (384 Kbps). **3G** (2000s) introduced video calls and streaming (2 Mbps) with **HSDPA/HSUPA** boosting speeds to **14 Mbps** (downlink) and **5.76 Mbps** (uplink). **4G** (2010s) introduced **LTE** for HD streaming and gaming (1 Gbps), improved with **LTE-A** (3 Gbps). **5G** (2019) introduced **NR** for ultra-low latency and IoT (10 Gbps).

5.2 Mobile operating systems:

Android – Open-source OS by **Google**, used in smartphones and smart devices. Supports Google Play Store apps and customization.
IOS – Closed-source OS by **Apple** for iPhones and iPads. Known for security and smooth user interface.
Windows Phone – OS by **Microsoft** with a tile-based interface, discontinued in **2019**.
Blackberry OS – Business-focused OS with strong security, discontinued in **2022**.
Symbian OS – Popular OS by **Nokia** for low-power smartphones, discontinued.
Tizen – Linux-based OS by **Samsung** for smartwatches, TVs, and IoT devices.
HarmonyOS – Cross-platform OS by **Huawei** for smartphones and smart devices.

5.3 Mobile development frameworks:

Native Development – Apps built specifically for a single platform using platform-specific languages.

Frameworks: Swift (iOS), Kotlin (Android), Objective-C, Java

Cross-Platform Development – Apps built using a single codebase for multiple platforms.

Frameworks: Flutter, React Native, Xamarin, Kotlin Multiplatform

Hybrid Development – Apps built using web technologies and wrapped in a native shell.

Frameworks: Ionic, Cordova, NativeScript

5.4 Mobile web technologies:

Mobile web technologies enable responsive, fast, and interactive web experiences on mobile devices:

1. **HTML5** – For creating mobile-friendly web pages.
2. **JavaScript** – For dynamic and interactive content.
3. **PWA** – Combines web and app features, supports offline use.

5.5 Connectivity and communication:

Connectivity and communication enable mobile devices to exchange data and interact with networks

1. **Wi-Fi** – Wireless local network for internet access.
2. **Bluetooth** – Short-range wireless communication between devices.
3. **NFC (Near Field Communication)** – Short-range data exchange (e.g., payments).

5.6 Sensors and hardware Innovations:

Sensors and hardware innovations enhance the functionality and user experience of mobile devices:

1. **Accelerometer** – Measures device orientation and movement.

2. **Gyroscope** – Detects rotational movement.
3. **Proximity Sensor** – Detects nearby objects to turn off the screen during calls.
4. **Ambient Light Sensor** – Adjusts screen brightness based on lighting.
5. **Fingerprint Sensor** – Enables secure biometric authentication.

5.7 Location Based Services (LBS):

Location-Based Services (LBS) provide information and services based on a user's geographic location:

1. **GPS (Global Positioning System)** – Provides accurate location tracking using satellites.
2. **Geofencing** – Triggers actions when entering or leaving specific areas.
3. **Location Sharing** – Allows users to share real-time location.

5.8 Cloud Storage:

Cloud storage allows users to store, access, and manage data over the internet:

1. **Google Drive** – Cloud service by **Google** for storing files and collaboration.
2. **iCloud** – Apple's cloud service for backing up and syncing files across devices.
3. **Dropbox** – Cloud-based storage for file sharing and syncing.

5.9 **Emerging technologies :** These are transforming industries and improving user experiences.

1. **Artificial Intelligence (AI)** – Enables machines to mimic human intelligence.
2. **Machine Learning (ML)** – Allows systems to learn and improve from data.
3. **Blockchain** – Secure, decentralized system for recording transactions.
4. **Internet of Things (IoT)** – Connects smart devices to collect and exchange data.
5. **Augmented Reality (AR)** – Overlays digital content onto the real world.
6. **Virtual Reality (VR)** – Immersive digital environment using headsets.

5.**11** AI **in Mobile and Google Assistant**

- **AI in Mobile** enhances functionality and user experience through automation and learning.
  - **Voice Assistants** – AI-powered assistants like **Google Assistant** and **Siri** for voice-based interaction.
- **Google Assistant** is an AI-powered virtual assistant by **Google** that enhances mobile experiences.
  - **Voice Commands** – Controls devices and performs tasks using voice.

6. Key Mobile Application Services :

Certainly! When it comes to **mobile application services**, there are several crucial aspects to consider. Let's explore some of the key services:

1. **User Sign-up/Sign-in and Management** – Manages user registration, login, and account recovery through email or social login (Facebook, Google, Twitter).
2. **Social Login** – Allows users to sign in using existing social media credentials, simplifying authentication and improving user experience.
3. **Analytics and User Engagement** – Tracks user behavior (e.g., session duration, interactions) to improve app performance and engagement.

4. **Push Notifications** – Sends real-time updates and reminders to keep users informed and engaged, improving retention.
5. **Real Device Testing** – Ensures app compatibility and performance across different devices and operating systems using real hardware.
6. **In-App Purchases** – Allows users to buy premium content, subscriptions, or features within the app.
7. **Payment Integration** – Supports secure payment options using services like Google Pay, Apple Pay, and credit cards.
8. **Cloud Synchronization** – Enables data syncing across multiple devices using cloud storage.
9. **Location-Based Services (LBS)** – Uses GPS and geofencing to offer location-based content and services.
10. **Customer Support and Feedback** – Integrates live chat, FAQs, and feedback forms to improve user support and satisfaction.

Remember that these services play a critical role in delivering a successful mobile app experience. Whether you're developing for iOS, Android, or cross-platform, thoughtful implementation of these services contributes to user satisfaction and app success.

Andriod:

Introduction to Android

Android is a mobile operating system developed by **Google**, based on the **Linux kernel** and other open-source software, designed for touch-screen devices like smartphones and tablets. It was created by **Android, Inc.** and acquired by **Google** in **2005**.

Open Handset Alliance (OHA) – A consortium of **84 companies** (Google, Samsung, Intel, eBay, etc.) established on **November 5, 2007**, to develop and promote open standards for mobile devices using the Android platform.
Android supports a wide range of applications and hardware based on the **ARM architecture**. It's open-source, allowing developers to modify and distribute it freely. Google also developed:

- **Android TV** – For smart TVs
- **Android Auto** – For cars
- **Wear OS** – For smartwatches

The unified development environment allows developers to create apps that work on various Android-powered devices. Android's low cost, flexibility, and large app ecosystem make it highly popular.

Features of Android

1. **Storage** – Uses **SQLite**, a lightweight relational database, for data storage.
2. **Connectivity** – Supports **GSM, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, and WiMAX**.
3. **Messaging** – Supports both **SMS** and **MMS**.
4. **Web Browser** – Based on **WebKit** and **Chrome's V8** JavaScript engine.
5. **Media Support** – Supports formats like **MP3, MP4, H.264, JPEG, PNG, GIF, BMP, AAC, and WAV**.
6. **Hardware Support** – Supports **Accelerometer, Camera, Compass, Proximity Sensor, and GPS**.
7. **Multi-Touch** – Enables use of multiple touch inputs simultaneously.
8. **Multi-Tasking** – Allows running multiple apps at once.
9. **Flash Support** – Android **2.3** supports **Flash 10.1**.
10. **Tethering** – Allows sharing of internet connection via USB or Wi-Fi hotspot.

Android Architecture:

Android architecture is a structured framework that organizes various components to ensure smooth operation and efficient application development. It consists of five key layers:

1. Linux Kernel

The Linux Kernel forms the foundation of Android. It manages low-level device drivers for hardware components like the camera, display, and audio. It also handles essential operations such as memory management, process scheduling, and system security. The kernel ensures that the hardware communicates effectively with the software layer, making Android reliable and stable.

2. Libraries

Above the Linux Kernel is a set of libraries written in **C/C++** that provide core features for Android. These include:

- **WebKit** – Powers the web browser.
- **SQLite** – Manages data storage in a lightweight relational database.
- **OpenGL** – Handles 2D and 3D graphics rendering.
- **SSL** – Provides internet security through encryption.
  These libraries enable smooth performance and enhanced multimedia capabilities.

3. Android Runtime (ART)

Android Runtime (ART) replaced the older **Dalvik Virtual Machine (DVM)** to improve app performance and memory efficiency. Each app runs in its own isolated instance of ART, ensuring that processes are managed independently, enhancing security and stability. ART converts app code into machine code at installation time (Ahead-of-Time compilation), improving execution speed and reducing resource usage.

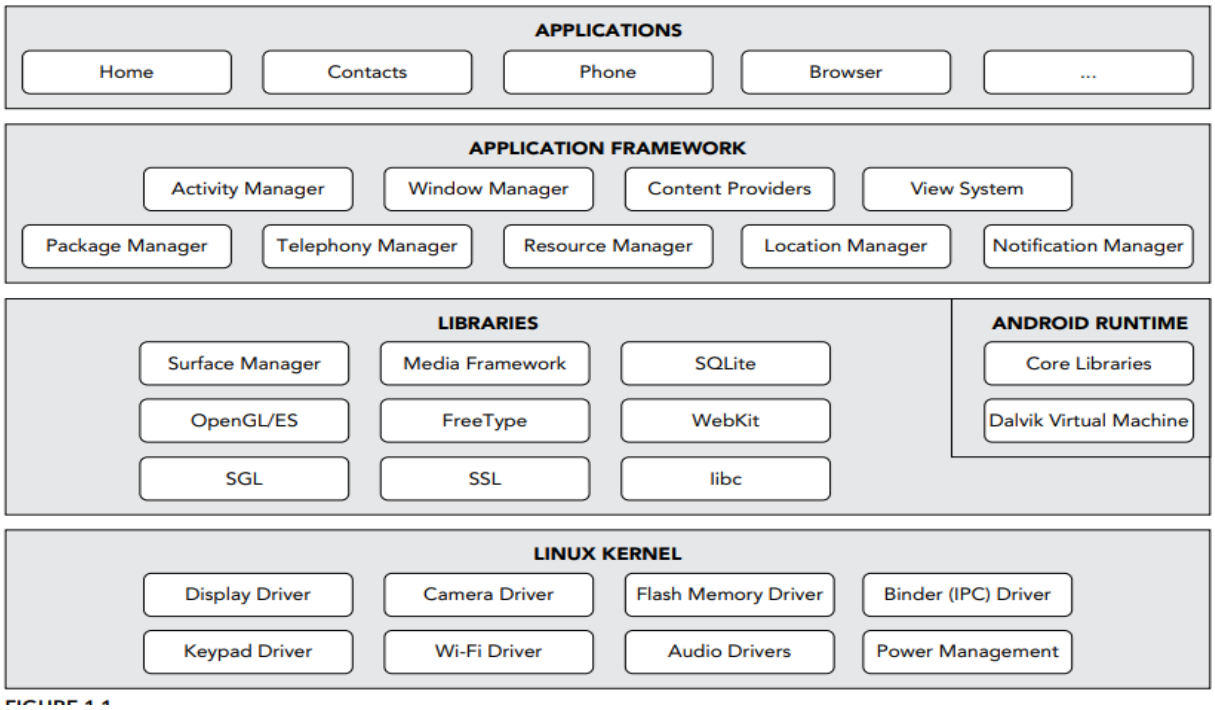
4. Application Framework

The application framework exposes the Android OS capabilities to developers through Java classes and APIs. Key components include:

- **Activity Manager** – Manages app lifecycle and user interactions.
- **Content Provider** – Enables data sharing between apps.
- **Notification Manager** – Handles alerts and push notifications.
- **Resource Manager** – Provides access to app resources like layouts and strings.
- **View System** – Manages user interface components and layouts.
  This framework helps developers create apps that are consistent and responsive across devices.

5. Applications

At the top layer are the pre-installed and user-installed apps. Pre-installed apps include **Contacts**, **Camera**, **Gallery**, and **Browser**. Developers create user-installed apps using **Java** or **Kotlin**, leveraging the framework services to ensure compatibility and performance across different devices.



Android Application Components

Android applications are built using four main components that define their structure and behavior:

1. Activities

An activity represents a single screen in an app. It manages the user interface and handles user interactions. For example, a social media app may have activities for logging in, browsing posts, and sending messages. The activity lifecycle is managed by the **Activity Manager**.

```
public class MainActivity extends Activity {

}
```

2. Services

Services handle long-running background tasks. They continue to function even if the user switches to another app. Examples include playing music, fetching data from the internet, and processing notifications. Services improve the user experience by ensuring uninterrupted functionality.

```
public class MyService extends Service {

}
```

3. Broadcast Receivers

A Broadcast Receiver is a component that allows an app to receive and respond to broadcast messages (called **intents**) from other apps or the system. These messages can indicate events like network connectivity changes, low battery, or custom events from other applications. Broadcast Receivers enable apps to react to such events even when they are not actively running.

```
public class MyReceiver extends BroadcastReceiver {

public void onReceive (context, intent){}

}
```

4. Content Providers

Content providers manage shared app data. They allow apps to access and modify data stored in files, SQLite databases, or cloud storage. For example, a messaging app can access the contact list using a content provider. Data requests are managed using the **ContentResolver** class.

```
public class MyContentProvider extends ContentProvider {

public void onCreate(){}

}
```

Additional Components

1. **Fragments** – Reusable UI components that allow dynamic and responsive layouts, useful for multi-panel interfaces on tablets and large screens.
2. **Intents** – Messaging objects that enable communication between components, like starting activities or launching services.
3. **Views** – Basic building blocks of the user interface, including buttons, text fields, and images.
4. **ViewGroups** – Containers for views and other ViewGroups, used to arrange and align UI elements.
5. **Widgets** – Interactive components that display dynamic content like weather updates and clocks on the home screen.
6. **Notifications** – Real-time alerts that appear in the status bar, allowing users to take action even when the app is not running.

Exploring the Development Environment:

To create robust and efficient android applications, it is essential to have the right tools and frameworks, the Android studio requires several components to develop and test android Applications efficiently.

Android SDK:

The **Android SDK** is a set of tools, libraries, and documentation from Google for building Android apps.

Key Features:

- **Libraries and APIs:** Access to UI components, data storage, and networking features.
- **Development Tools:** Includes compilers, debuggers, and build tools.
- **Documentation:** Provides guides on APIs, best practices, and development.
- **Sample Code:** Offers ready-to-use code snippets and templates.
- **Updates and Support:** Regular updates for new features and compatibility.

Android Studio:

Android Studio is the official Integrated Development Environment (IDE) for Android app development with code editing's and visual design features. It is designed to streamline the entire app development process. It serves as a comprehensive platform for designing, coding testing, and debugging Android applications.

Key features:

- **Code Editor:** Provides syntax highlighting, code completion, refactoring, and navigation for better coding efficiency.
- **Layout Editor:** Drag-and-drop tool to create responsive app interfaces with real-time previews.
- **SDK Manager:** Manages Android SDK versions and essential tools.
- **Device Manager:** Creates and manages virtual devices for testing on different configurations.
- **APK Analyzer:** Analyzes APK size, contents, and dependencies to improve performance.
- **Built-in Emulator:** Tests apps on virtual devices with various screen sizes and Android versions.
- **Device Testing:** Tests apps on real devices connected via USB.
- **Version Control:** Supports Git and other systems for tracking changes and collaboration.
- **Plugin Ecosystem:** Wide range of plugins to extend functionality and integrate with external tools.
- **Performance Profiling:** Analyzes CPU, memory, and network usage to improve performance.
- **Build Tools:** Uses Gradle to automate builds, manage dependencies, and configure variants.
- **Integrated Debugger:** Helps identify and fix bugs by setting breakpoints and inspecting code.

Android Emulators:

Android Emulators are virtual devices that simulate the behavior of real Android devices for testing and debugging apps. They allow developers to test and debug applications on various device configurations without needing physical devices.

Key Features:

- **Device Configurations:** Simulate different screen sizes, resolutions, and Android versions.
- **Device Testing:** Test apps on various virtual devices without needing physical ones.
- **Debugging:** Analyze and fix app issues through simulated scenarios.

- **Performance Testing:** Optimize resource usage and identify bottlenecks.
- **Multi-Device Testing:** Test on multiple virtual devices simultaneously.
- **Advanced Features:** Simulate calls, SMS, location, and other hardware functions.

ADB (Android Debug Bridge):

ADB is a command-line tool that connects a development machine to an Android device or emulator for debugging and testing. ADB offers the various debugging and troubleshooting functionalities.

Key Features:

- **Installing and Debugging:** Install and debug apps directly on connected devices.
- **Device Shell Access:** Execute commands and troubleshoot directly on the device.
- **File Transfer:** Move files between a computer and Android device.
- **Managing Device State:** Change permissions and simulate different test scenarios.

Gradle Build System:

Gradle is the build automation tool used in Android studio to manage project dependencies, build Configurations, and Tasks efficiently.

Key Features:

- **Dependency Management:** Simplifies handling libraries and resources in projects.
- **Build Configurations:** Supports different build types and product flavors (e.g., debug, release).
- **Task Automation:** Automates tasks like compiling, testing, and packaging apps.
- **Integration with Android Studio:** Works seamlessly with Android Studio for efficient development.

Exploring the IDE (Andriod Studio):

1. Project Structure

- **Project View**: Displays files and directories in a hierarchical format for easy navigation.
- **Android View**: Groups Android-specific files like manifests, resources, and Java/Kotlin classes.

2. Code Editor

- **Code Editing**: Central area for writing Java/XML code with syntax highlighting, auto-completion, and error checking.
- **Code Navigation**: Use shortcuts like "Go to Declaration" to move through classes and methods efficiently.

3. Design Editor

- **Layout Editor**: Drag-and-drop UI elements with real-time previews across different screen sizes.

- **Component Tree**: Displays the UI hierarchy for better structure and organization.

4. Toolbar

Provides quick access to build, run, sync, debug, and version control tools.

5. Run & Debug

- **Run**: Executes the app on an emulator or physical device.
- **Debug**: Set breakpoints, inspect variables, and step through code for troubleshooting.

6. Menu

Includes options like File, Edit, View, Navigate, Code, Build, Run, Tools, VCS, Window, and Help for project management.

7. Logcat

Displays logs, errors, and warnings generated by the app for debugging.

8. Device Manager

Creates and manages virtual devices to test apps on different configurations.

9. Gradle Build System

- Automates tasks like compiling code and generating APKs.
- Manages dependencies and allows multiple build types (debug/release) and flavors (free/paid).

10. Version Control

Integrates with Git for tracking changes, rollback, and team collaboration.

11. Settings & Preferences

Customizes themes, key bindings, code formatting, and build configurations.

12. Help & Documentation

Provides access to Android Studio documentation, tutorials, and community support.

13. Plugins

Enhances IDE functionality with additional tools and language support.

14. SDK Manager

Downloads, installs, and updates necessary Android SDK components for development.

Debugging your Android Application in Android Studio

- In Android Studio, debugging is the process of running your Android app step-by-step to identify and fix issues such as crashes, incorrect outputs, and performance problems before releasing them to users.

Steps Involved in Debugging the Application:

1. **Set Breakpoints**

   Breakpoints allow the app to pause at specific points to inspect the state of the app.

   To set Breakpoints:

   - Open the file in Android Studio.
   - Click on the left margin next to the line of code where you want to pause the execution.
   - A red dot will appear, indicating that the breakpoint is set.

2. **Manage Breakpoints**

   You can view and manage all breakpoints from:

   Run > View Breakpoints (or press Ctrl + Shift + F8).

Connect Device or Start an Emulator:

- To debug the app, you need to run it on a physical device or an emulator. Connect your Physical Android device using a USB cable. Or Start an emulator Go to Tools > Device Manager. Select an emulator from the list and click Launch.

Run the App in Debug Mode:

- Click the Debug icon in the top toolbar of Android Studio to run the app in debug mode. Select a connected device or emulator to deploy the app for debugging.

Navigate Through the App:

- While in debug mode, interact with the app to trigger the code execution. The app will pause at the set breakpoints, allowing you to inspect variables and understand the program flow.

Inspect Variables and evaluate Expressions:

- While the app is paused at a breakpoint, you can inspect the values of variables and test expressions. Use the Variables window to check the current state of variables.
- Right-click in the debug window and select Evaluate Expression to modify or test variable values at runtime.

Step Through the code:

- Android Studio provides the following debugging controls to navigate through the code line by line:
- Step Over (F8): Executes the current line without entering functions. Stops at the next line after the function call.
- Step Into (F7): Enters the method or function call on the current line and pauses at the first line inside it.

- Step Out (Shift + F8): Exits the current method and returns to the caller.

Evaluate Logs and Expressions

- Use Logcat to view real-time logs and debug messages.
- Add logs using:
- Log.d("DEBUG", "Value of x: " + x);
- Filter logs using tags or keywords. Right-click in the debug window and select Evaluate Expression to modify or test variable values at runtime.

Use Profiling Tools

- This tool provides insights into app performance; CPU usage, memory allocation, and network activity, helping Developers optimize their applications for better efficiency and user experience.

Publishing your android Application:

Step 1: Preparing an app for release

- Before publishing an app, several steps must be completed to ensure it is ready for release.
- **Testing:** Test the app on different devices, screen sizes, and Android versions to ensure compatibility and proper functionality under various conditions.
- **Optimization:** Optimize the app by using Proguard to shrink and obfuscate code, reducing app size and improving performance.
- **Security** means protecting the app from threats by preventing unauthorized access and safeguarding user data from misuse or breaches.
- **Compliance:** Ensure the app complies with all Google Play store policies and guidelines.

Step 2. Steps to Generate a Signed APK:

- To publish your app on the **Google Play Store**, you must generate a **signed APK** (Android Package), which is the compiled, executable version of your app.
- **Signing an APK** means adding a digital signature to your app to prove that you are the developer. It helps Google and users trust that the app is safe and hasn't been changed by anyone else.
- Go to **Build → Generate Signed APK** in the menu bar.
- Create a new key store or select an existing one.
- Fill in key store details (path, password, key alias, validity, certificate info).
- After signing, click **OK** to return to the Generate Signed APK window.
- In the Generate Signed APK window, click **Next** to review and finish the process.
- Once the signed APK is generated, go to the **Google Play Console** (https://play.google.com/apps/publish/) to upload it.

Step:3 Creating a Google play Developer Account:

- Before publishing an app on the Google play store, we need to create a google Play Developer account. This requires a One – time registration fee.
- Go to the google play developer Console website
- Sign in with a google Account or create new one
- Follow the on – screen instructions to complete the registration Process and pay the registration fee.

Step 4: Uploading an App to the Google Play Console

- Once you have a signed APK file and a Google Play Developer account, follow these steps to upload your app:
- **Log In** – Sign in to the Google Play Console.
- **Create Application** – Click on "Create Application" and enter details like the app title, description, and screenshots.
- **Upload APK** – Upload the signed APK file and provide any additional required information.
- **Set Pricing and Distribution** – Choose whether the app is free or paid, and select the target regions.
- **Publish** – Click **"Save"** and **"Publish." to submit app in to Google play store.**

Obtaining the Required Tools:

The **Java Development Kit (JDK)** is a crucial tool for Android development, as it provides the necessary environment to compile and run Java-based applications. Android Studio relies on Java for writing Android applications, especially for older versions of Android before Kotlin became the preferred language.

Installing JDK for Android Studio:

- Download the latest **JDK (Java SE Development Kit)** from Oracle's website or use an open-source version like **OpenJDK**.

- After installation, configure the **JAVA_HOME** environment variable.
- Android Studio automatically detects JDK, but you can set it manually under **File → Project Structure → SDK Location**.

    - Android Studio is the official IDE for Android app development, providing tools for coding, designing, testing, and debugging.
    - **Key Features:** Intelligent code editor, Gradle-based build system, Emulator, Layout Editor, and Performance Analyzer.
    - **Supported Languages:** Primarily Kotlin and Java, with some support for C++.
    - **System Requirements:** Requires at least 8GB RAM, JDK, and Windows/macOS/Linux OS. **Installation:** Download from the official Android Developer site, install required SDK components, and configure settings.

Chapter-2 :

Using Activities - Fragments and Intents in Android: Working with activities, Using Intents, Fragments, Using the Intent Object to Invoke Built–in Application

Introduction to Activities in Android:

- An activity is a class that represents a single screen in android. It is like window or frame of Java.
- By the help of activity, you can place all your UI components (button, label, text field etc.) or widgets in a single screen.
- Activity is one of the most important components for any android app.
- It is similar to the main () function in different programming languages.
- It is the main entry point for user interaction.
- Any application, don't matter how small it is (in terms of code and scalability), has at least one Activity class. You can have multiple activities in your app.
- All your activities must be declared in the manifest file, with their attributes.
- Android Activity Lifecycle is controlled by 7 methods of android.app.Activity class. The android Activity is the subclass of ContextThemeWrapper class.

Lifecycle of an android activity:

Every activity has different functions throughout its life, onCreate (), onStart (), onResume (), onPause (), onStop (), onRestart (), onDestroy ().

Figure 3-2 shows the life cycle of an activity and the various stages it goes through—from when the activity is started until it ends.

- **The Activity class defines the following Methods:**

- **OnCreate()** — Called when the activity is first created
- **OnStart()** — Called when the activity becomes visible to the user
- **OnResume()** — Called when the activity starts interacting with the user **or** activity is becoming visible to the user and is ready to start receiving user interactions.
- **OnPause()** — Called when the current activity is being paused and the previous activity is being resumed. This is where you typically pause ongoing processes or save transient data.
- **OnStop()** — Called when the activity is no longer visible to the user.
- **OnDestroy()** — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
- **OnRestart()** — Called when the activity has been stopped and is restarting again.

➢ By default, the activity created for you contains the onCreate() event. Within this event handler is the code that helps to display the UI elements of your screen.

Demo Android App to Demonstrate Activity Lifecycle in Android

```
package com.example.activity101;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
public class MainActivity extends AppCompatActivity
{
String tag = "Lifecycle Step";
@Override
protected void onCreate(Bundle savedInstanceState)
{
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
Log.d(tag, "In the onCreate() event");
}
public void onStart()
```

```
{
super.onStart();
Log.d(tag, "In the onStart() event");
}
public void onRestart()
{
super.onRestart();
Log.d(tag, "In the onRestart() event");
}
public void onResume()
{
super.onResume();
Log.d(tag, "In the onResume() event");
}
public void onPause()
{
super.onPause();
Log.d(tag, "In the onPause() event");
}
public void onStop()
{
super.onStop();
Log.d(tag, "In the onStop() event");
}
public void onDestroy()
{
super.onDestroy();
Log.d(tag, "In the onDestroy() event");
}
}
```

Explanation:

1. package com. example.activity101;

This declares the package name for the Java file (activity101.java).

2. import android.support.v7.app.AppCompatActivity;
   import android.os.Bundle;
   import android.util.Log;

These import statements bring in classes from the Android framework that is necessary for this activity. AppCompatActivity is the base class for activities that use the Support Library action bar features. Bundle is used for passing data between activities. Log is used for logging messages.

3. public class MainActivity extends AppCompatActivity
{

When a class extends another class in Java, it means that the subclass (in this case, MainActivity) inherits all the fields and methods from the super class (AppCompatActivity).

4. String tag = "Lifecycle Step";

This declares a string variable tag and initializes it with the value "Lifecycle Step". This tag will be used in logging messages to identify them.

5. @Override
protected void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   Log.d(tag, "In the onCreate() event");
}

- This method is called when the activity is first created. It sets the content view to a layout defined in activity_main.xml file. Then it logs a debug message indicating that onCreate() event has occurred.
- When an activity is destroyed and recreated due to configuration changes (such as screen rotation), Android preserves certain data from the activity's previous state when it's recreated. This preserved data is stored in the savedInstanceState bundle.

6.
```
@Override
public void onStart() { /* onStart code */ }

@Override
public void onRestart() { /* onRestart code */ }

@Override
public void onResume() { /* onResume code */ }

@Override
public void onPause() { /* onPause code */ }

@Override
public void onStop() { /* onStop code */ }

@Override
public void onDestroy() { /* onDestroy code */ }
```

These methods are overrides of the lifecycle callback methods provided by the AppCompatActivity class. Each of these methods is called by the Android system at specific points in the activity's lifecycle.

Logging Messages:

In each of the overridden methods, there's a call to Log.d() to log a debug message indicating the current lifecycle event.

Intents or Linking Activities Using Intents:

- An Android application can contain zero or more activities. When your application has more than one activity, you often need to navigate from one to another.
- In Android, you navigate between activities through what is known as intent.
- Intent is a messaging object used to request any action from another app component. Intent's most common use is to launch a new activity from the current activity.
- Intent facilitates communication between different components. Intent object is used to call other activities.
- The intent is used to launch an activity, start the services, broadcast receivers, display a web page, dial a phone call, send messages from one activity to another activity, and so on.

Common Use cases for Intents include:

1. **Starting Activities:** Use intents to launch new activities within your application or to launch activities from other applications.

For example, imagine you have a button in your app that says "Open Camera". When a user taps that button, you can use intent to ask the Android system to open the camera app.

2. **Broadcasting Messages:** Use intents to send broadcast messages within your application or to other applications, allowing them to receive and respond to events.
   Ex: let's say you have a music player app and you want to notify other apps whenever a new song starts playing.

- You would use a broadcast intent to send a message saying "Hey, a new song is playing!
- Other apps, like maybe a notification app or a social media app, can listen for this message. When they receive it, they can do things like show a notification saying what song is playing or share the song information on social media.
- Sent when the device finishes booting up, allowing apps to start up services or perform initialization tasks.

3. **Invoking Services:** Use intents to start services that perform background tasks or handle long-running operations.
    - Services: Background workers in your app that perform tasks without needing to be in the foreground.
    - Intents: Messages used to start services.

    Example
    - Create a service to download a file.
    - Register the service in the manifest.
    - Start the service using intent from an activity.

    Using services and intents allows your app to perform tasks like downloading files or playing music in the background, ensuring the main thread remains responsive and providing a smoother user experience.

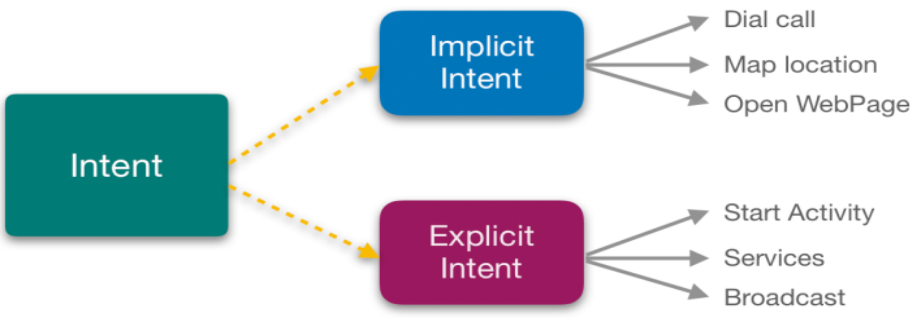
4. **Passing Data:** Intents can carry data (extra information) as key-value pairs, allowing components to exchange information, such as passing data between activities or between different parts of your application.
    - For example, let's say you have an app with two screens, and you want to send a message from the first screen to the second screen. You would use intent to carry that message.
    - **First Screen (Calling Screen):** You create intent and put the information about the call, like the caller's name, as extras.
    - **Second Screen (Receiving Screen):** You get the information about the call from the intent.

Methods and their Description:

Methods	Description
Context.startActivity()	This is to launch a new activity or get an existing activity to be action.
Context.startService()	This is to start a new service or deliver instructions for an existing service.
Context.sendBroadcast()	This is to deliver the message to broadcast receivers.

TYPES OF INTENT

- Intents are of two types:



- Explicit Intent is used to invoke a specific target component. It is used to switch from one activity to another activity in the same application. It is also used to pass data by invoking the external class.

- Explicit Intents specify the target component by providing the exact class name of the component to be invoked.

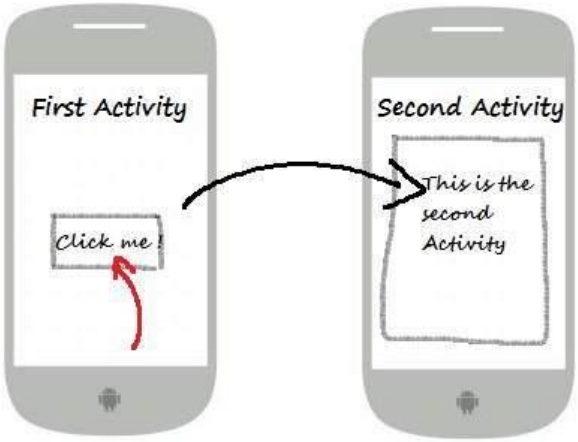
Example:-

1. We can use explicit intent to start a new activity when the user invokes an action or plays music in the background or on click button go to another activity.

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

   - MainActivity.this specifies the current context, which is the MainActivity.
   - SecondActivity.class specifies the target activity to which you want to navigate.
   - In Explicit we use the name of component which will be affected by Intent. **For Example:** If we know class name then we can navigate the app from One Activity to another activity using Intent.

2. In amazon app if you go to Home page you can see prime, fresh, mobiles, electronics etc. If you click prime it transit to prime page activity likewise other tab also works.



IMPLICIT INTENT:

- An implicit intent is used when you want to perform an action, but you don't care which component performs it.

- The system will determine the appropriate component to handle the intent based on the available components that can respond to it.
- Example: Suppose you want to open a website URL in a web browser. You don't know which browser the user prefers, so you'll use an implicit intent:

```
//Intent object and open the webpage
Intent intent = new
Intent(Intent.ACTION_VIEW,Uri.parse("https://example.com"));
startActivity(intent); //call a webpage
```

- Intent.ACTION_VIEW is the action you want to perform, which is viewing content.
- Uri.parse("https://example.com") specifies the data you want to view, which is a website URL.
  - Explicit intents are used for navigating within your own by specifying the target component, while implicit intents are used for performing actions where the system determines the appropriate component to handle the request.

Using the Intent Object to Invoke Built–in Application:

- One of the key aspects of Android programming is using the intent to call activities from other Applications. An Application can call many built-in Applications, which are included with an Android device.
- Suppose, you want to load a Web page, which is not part of your Application. You can use the Intent object to invoke the built-in Web Browser to display the Web page, instead of building your own Web Browser for this purpose.
- Add three buttons are in activity_main.xml file as Web Browser. Call here and display Map.

Code:

```
<Button
android:id="@+id/buttonwebbrowser"
android:text="Web Browser"
```

```
android:onClick="onClickBrowseWeb"/>
<Button
android:id="@+id/buttondocall"
android:text="Call Here"
android:onClick="onClickCallHere"/>
<Button
android:id="@+id/buttondisplaymap"
android:text="Display Map"
android:onClick="onClickDisplayMap"/>
```

Add three methods are in MainActivity.java class to display the Website in the Web Browser. Call on any mobile number and display the map by Google maps.

Code:

```
public void onClickBrowseWeb(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.co.in"));
    startActivity(intent);
  }
  public void onClickCallHere(View view) {
    Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse("tel:+91xxxxxxxxxx"));
    startActivity(intent);
  }

                    Or
setContentView(R.layout.activity_main);
String[] phoneNumbers = {
      "+91xxxxxxxxxx", // India
      "+1xxxxxxxxxx",  // USA/Canada
      "+44xxxxxxxxxx", // UK
      "+61xxxxxxxxxx", // Australia
      "+81xxxxxxxxxx"  // Japan
    };

private void startCallIntent(String phoneNumber) {
    Intent intent = new Intent(Intent.ACTION_CALL);
    intent.setData(Uri.parse("tel:" + phoneNumber));
```

```
    startActivity(intent);
}
```

To display the Map based upon lat,long Co ordinates

```
public void onClickDisplayMap(View view) {
    Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("geo:28.7041,77.1025"));
    startActivity(intent);
}
```

Or

To display the Map based upon Location name

```
public void onClickDisplayMap(View view) {
    String location = editTextLocation.getText().toString();
    if (!location.isEmpty()) {
        Uri geoLocation = Uri.parse("geo:0,0?q=" + Uri.encode(location));
        showMap(geoLocation);
    }
}
```

In the first button, create an object of Intent and then pass the two arguments to its constructor. The action and the data will be,

```
1. Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse("http://www.google.co.in"));


2. startActivity(intent);
```

The action here is represented by the Intent.ACION_VIEW constant. We used the parse() method of the URL class to convert a URL string into a URL object.

For the second button, we can dial a specific number by passing in the telephone number in the portion.

```
1.  Intent intent=new Intent(Intent.ACTION_DIAL, URL.parse("tel:+91xxxxxxxxxx"));
```

```
2. startActivity(intent);
```

In this case, the dialler will display the number to be called. The user must still click the dial button to dial the number. If you want to directly call the number without the user intervention, change the action, given below,

```
1. Intent intent=new Intent(Intent.ACTION_CALL, URL.parse("tel:+91xxxxxxxxxx"));
2. startActivity(intent);
```

For this, you need to assign the Android.permission.CALL_PHONE permission to your Application, defined in the manifest file.

You can simply omit the data portion to display the dialer without specifying any number.

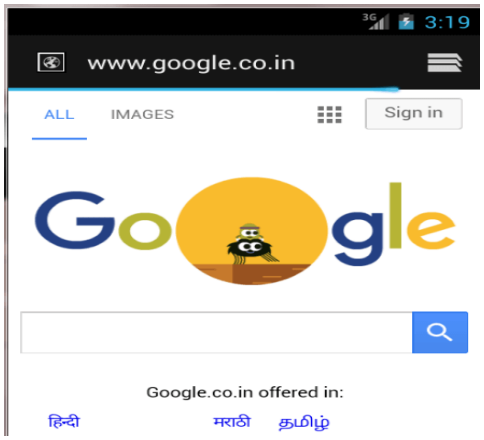
```
1. Intent intent=new Intent(Intent.ACTION_DIAL);
2. startActivity(intent);
```

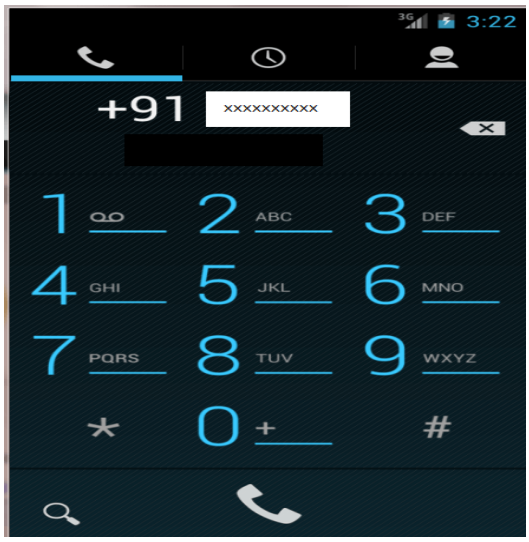
The third button displays a map using the ACTION_VIEW constant. Here we use “geo” in place of “http”.

```
1. Intent intent=new Intent(Intent.ACTION_VIEW,Uri.parse("geo:28.7041,77.1025"));
2. startActivity(intent);
```

In this app, we have used the Intent class to invoke some of the built-in Applications in Android like Browser, Phone, and Maps).

- Now, click on the first option. Google page will open in the Browser.
- Click the second option. It will open a dial-up option to make a call to the specified mobile number.
- To load the Maps Application, click the display map button. To display the Maps Application, you need to run the Application on an AVD, which supports the Google APIs or you have to run this app on your device.





Intent filter

- Implicit intent uses the intent filter to serve the user request.
- The intent filter specifies the types of intents that an activity, service, or broadcast receiver can respond to.
- Intent filters are declared in the Android manifest file.
- Intent filter must contain <action>

1. **<actions>** In this, you can keep all the actions you wish your intent to accept.

Constant	Target Component	Action
ACTION_CALL	Activity	Initiate a phone call
ACTION_EDIT	Activity	Display data for the user to edit
ACTION_BATTERY_LOW	broadcast receiver	A warning that the battery is low
ACTION_HEADSET_PLUG	broadcast receiver	A headset has been plugged into the device, or unplugged from it.

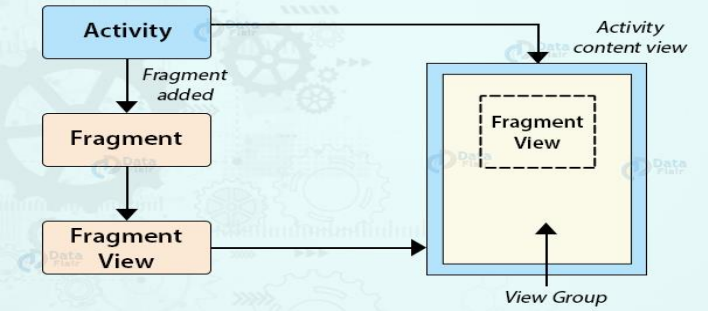
2. **<data>** It defines the type of data that the intent will take.

- Example: I f the action field is ACTION_EDIT, the data field would contain the URI of the document to be displayed for editing.

3. **<Category>** It represents the name of the category that the intent will accept.A string containing additional information about the kind of component that should handle the intent.

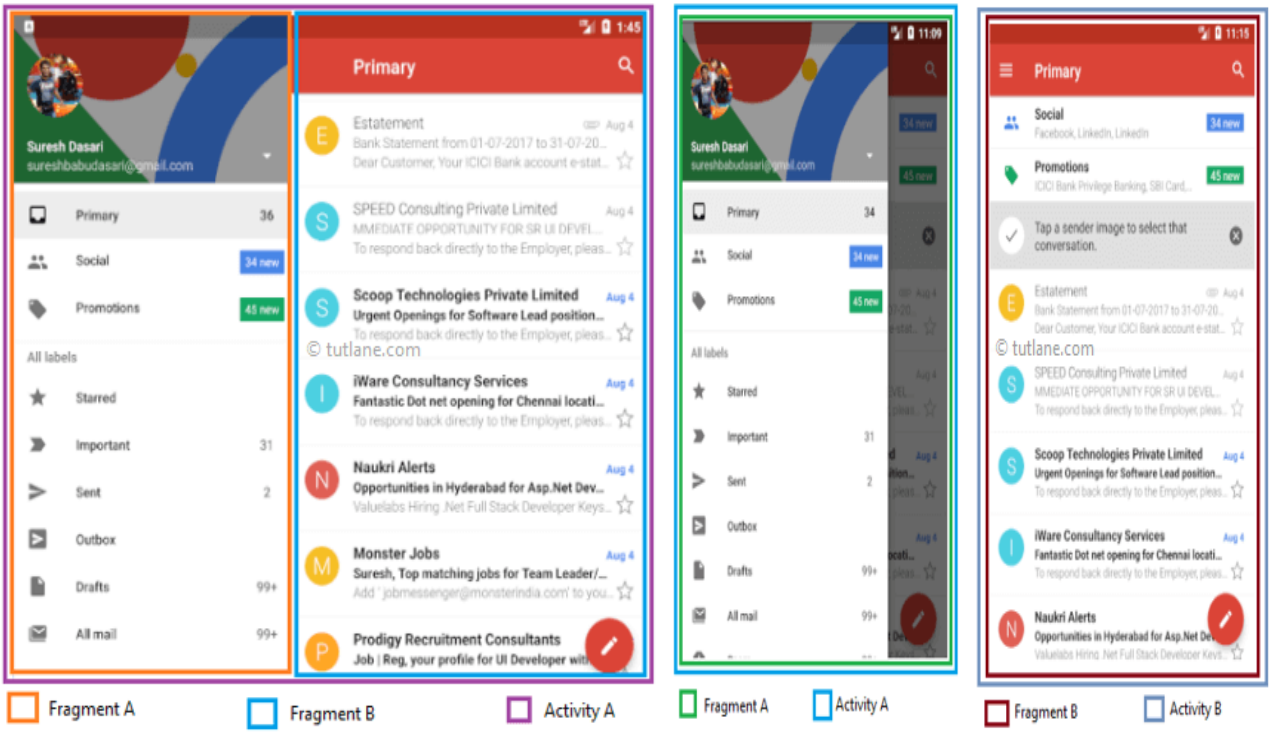
Android Fragments:

- Android Fragment is a Graphical User Interface component of Android. It resides within the Activities of an Android application. It represents a portion of UI that the user sees on the screen.
- Android Fragments cannot exist outside an activity. Another name for Fragment can be **Sub-Activity** as they are part of Activities.
- Fragments are always embedded in Activities; Multiple Fragments can be added to single activity.

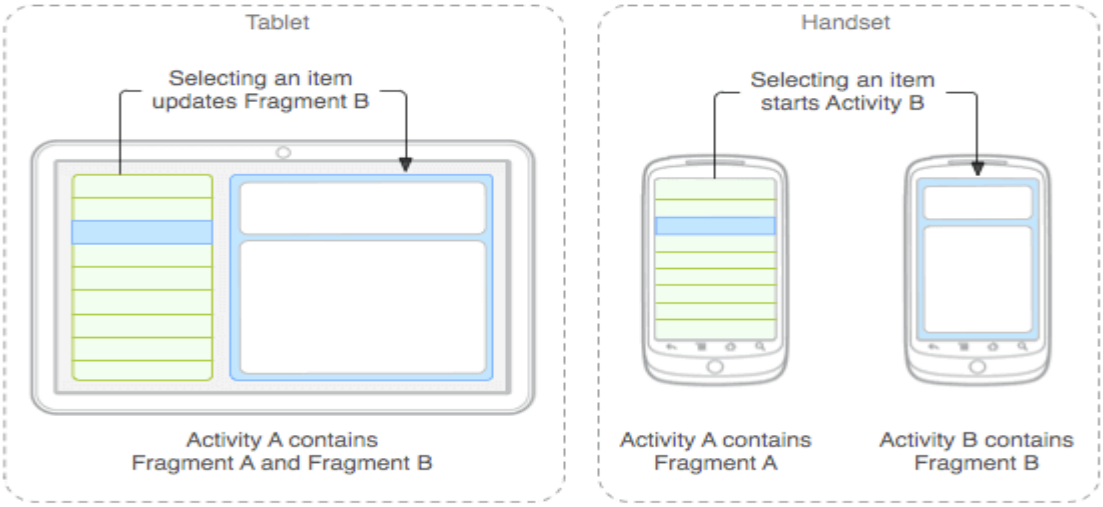


How Fragment Interacts with Activity in Different Devices:

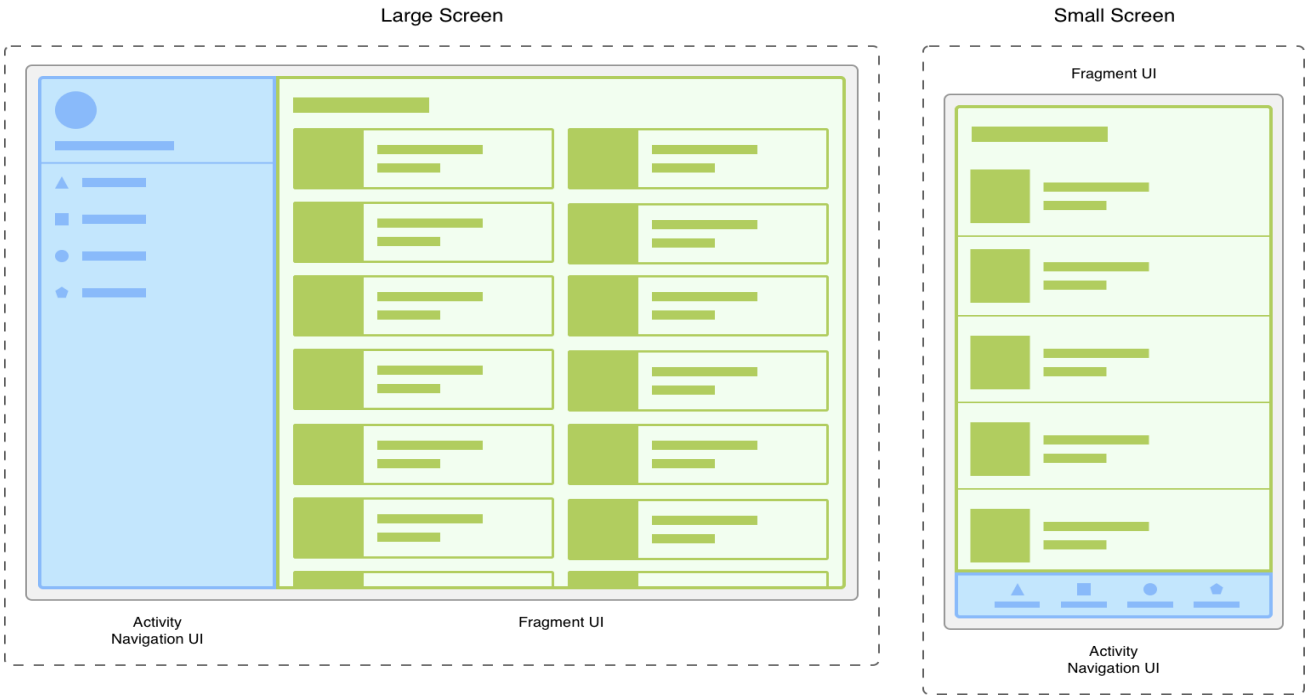
- If you observe above example for **Tablet** we defined an **Activity A** with two fragments such as one is to show the list of items and second one is to show the details of item which we selected in first fragment.
- For **Handset** device, there is no enough space to show both the fragments in single activity, so the **Activity A** includes first fragment to show the list of items and the **Activity B** which includes another fragment to display the details of an item which is selected in **Activity A**.
- For example, **GMAIL app** is designed with multiple fragments, so the design of GMAIL app will be varied based on the size of device such as tablet or mobile device.



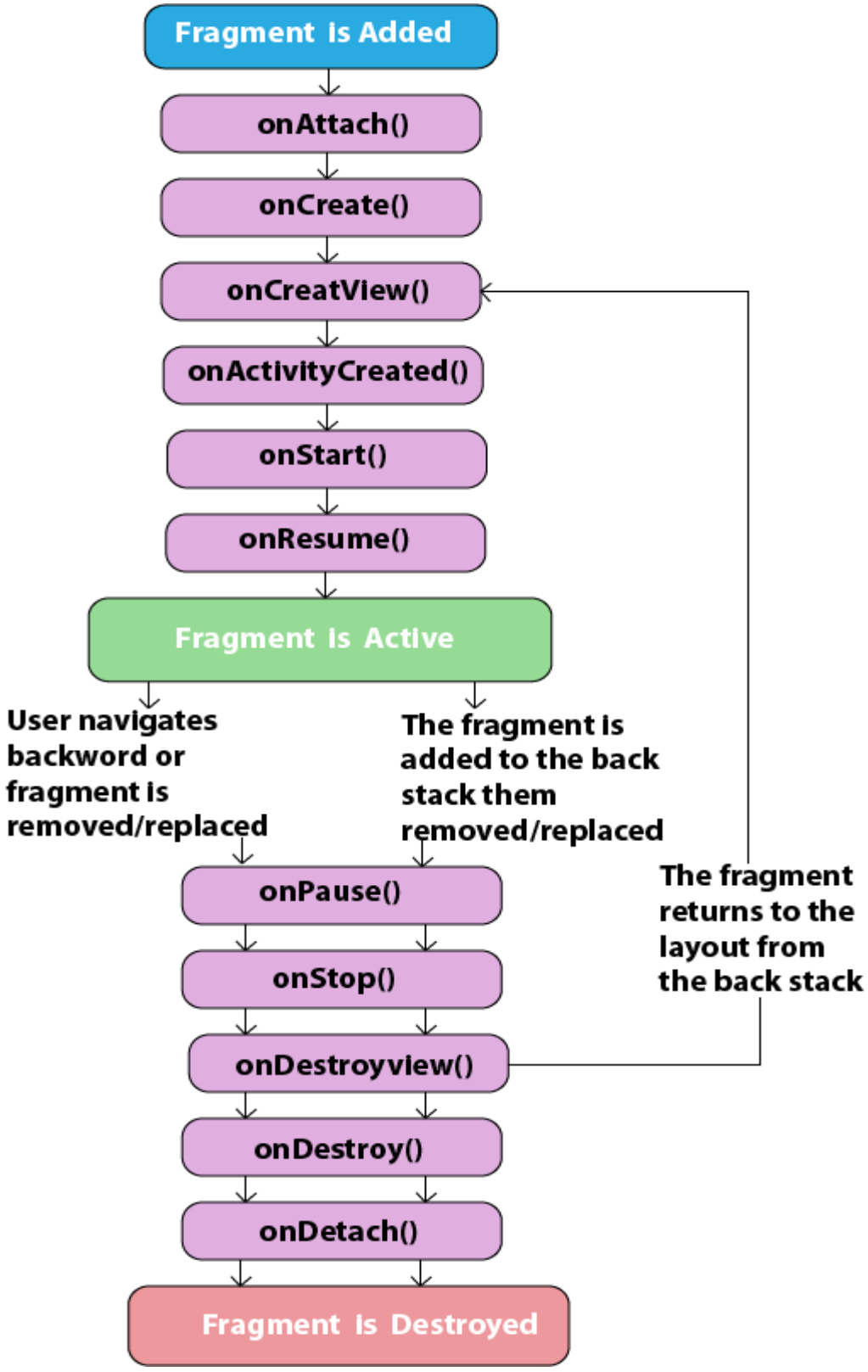
Tablet View **Mobile View**



- **In the case of mobiles, there are two activities that are:**
  - Activity 1 with Fragment A and Activity 2 with Fragment B. When we select an item from Fragment A, it gets open in the Fragment B of Activity 2.
- **In tablets, there is only one activity that is Activity 1.**
  - In **Activity 1**, there are two fragments, **Fragment A** and **Fragment B**. When we select an item from Fragment A, it gets open in Fragment B of the same activity.
- On larger screens, you might want the app to display a static navigation drawer and a list in a grid layout. On smaller screens, you might want the app to display a bottom navigation bar and a list in a linear layout.



Android Fragment Lifecycle:



Method	Description
onAttach()	It is called when the fragment has been associated with an activity.
onCreate()	It is used to initialize the fragment.
onCreteView()	It is used to create a view hierarchy associated with the fragment.
onActivityCreated()	It is called when the fragment activity has been created and the fragment view hierarchy instantiated.
onStart()	It is used to make the fragment visible.
onResume()	It is used to make the fragment visible in an activity.
onPause()	It is called when fragment is no longer visible and it indicates that the user is leaving the fragment.
onStop()	It is called to stop the fragment using the onStop() method.
onDestoryView()	The view hierarchy associated with the fragment is being removed after executing this method.
onDestroy()	It is called to perform a final clean up of the fragments state.
onDetach()	It is called immediately after the fragment disassociated from the activity.

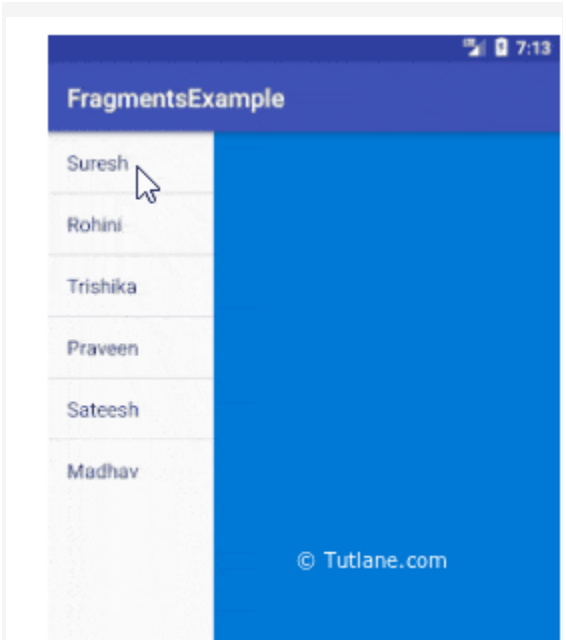
ListMenuFragment.java

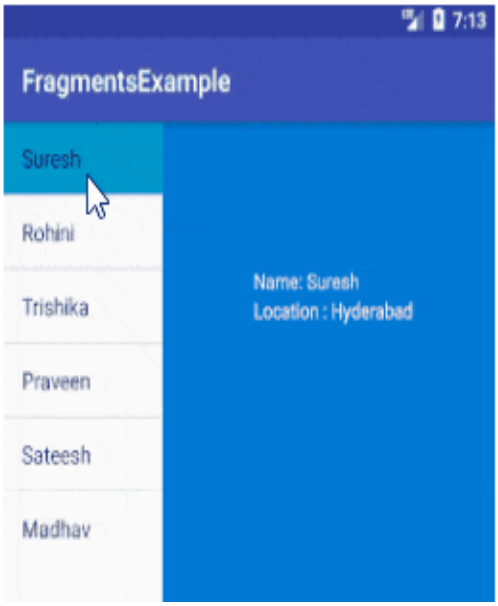
```
public class ListMenuFragment extends ListFragment {
    String[] users = new String[]
{ "Suresh","Rohini","Trishika","Praveen","Sateesh","Madhav" };

String[] location = new String[]{"Hyderabad","Guntur","Hyderabad","Bangalore","Vizag","Nag
pur"};
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
savedInstanceState) {
        View view =inflater.inflate(R.layout.listitems_info, container, false);
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(getActivity(),
                android.R.layout.simple_list_item_1, users);
```

activity_main.xml:

```
<fragment
        android:layout_height="match_parent"
        android:layout_width="350px"
        class="com.tutlane.fragmentsexample.ListMenuFragment"
        android:id="@+id/fragment"/>
    <fragment
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.tutlane.fragmentsexample.DetailsFragment"
        android:id="@+id/fragment2"/>
```





Adding Fragments with Activities:

Embedding Fragments with Activities means adding the Fragments to the respective Activity Layout. Now, there are two ways for adding multiple Fragments in one Activity:

1. Statically

To add the fragment statically, we need to mention it ourselves in the these fragments can't be replaced during the execution as they are static. Defined in XML, fixed during compile-time, and cannot be changed at runtime.

2. Dynamically

In this, we embed our Fragment in Activities dynamically using **Fragment Manager**.

Unlike Static Fragment, in this, we can add, remove or replace the Fragments at the runtime itself. Managed in code, flexible, can be added, removed, or replaced at runtime.

Types of Android Fragments

1. Single Fragments

Single fragments show only a single view for the user on the screen. These are for handheld devices such as mobile phones.

2. List Fragments

List fragments are those that have a special list view feature. In this, there's a list and the user can choose to see a Sub-Activity.

3. Fragment Transactions:

Fragment transactions are for the transition from one fragment to another. It supports switching between two fragments.