جامعة محمد خيضر
Mohamed Khider
UNIVERSITY

# Mobile Applications (MobApps) Course

Dr. Meftah ZOUAI
Associate Professor, University of Biskra
LINFI Laboratory, Computer science department,
Mohamed Khider University, 07000, Biskra, Algeria
meftah.zouai@univ-biskra.dz

https://www.youtube.com/meftahzouai

# General Information

**UE Title:** UEF1

**Subject Title:** Mobile Applications (AppMob)

**Credits:** 5

**Coefficients**: 3

**Number of weeks:** 14 -16 weeks

**Semester workload:** 42 hours

**Weekly workload:** lecture (1:30) + practical work (1:30)

**Evaluation method:** Continuous assessment(40%)+Examination (60%)

# Outline

- Introduction to Mobile Apps
- Android Platform
- Activities and Resources
- Graphical Interfaces and Widgets
- Menus and Dialog Boxes
- Sensors
- AndroidManifest.xml and Communication between Components
- Data Management (Database…)
- Security and private life
- Playstore

# Chapiter I:

# Introduction to Mobile Apps

# Mobile devices


Mobile devices, also known as mobile phones, are handheld electronic devices that are used for communication and a variety of other purposes. They have become an essential part of our daily lives, providing us with the ability to make calls, send messages, access the internet, and run various applications and programs.

Mobile phones come in different shapes, sizes, and specifications, but they all have similar basic features, such as a touch screen display, a camera, a microphone, and a speaker. The first mobile phones were simple devices with basic functions, but they have evolved over the years to become sophisticated devices with a wide range of capabilities.

Today, mobile phones are available from a variety of manufacturers, are also equipped with various sensors, such as accelerometers, gyroscopes, and GPS, which provide users with location-based services and augmented reality experiences.

# Evalution of Mobile devices

The evaluation of a mobile phone involves assessing its performance, design, features, and overall value for money. Some of the key factors to consider when evaluating a mobile phone include:

1. Performance: This includes the phone's processor speed, RAM, storage capacity, and overall speed and responsiveness of the device.
2. Display: The quality of the phone's screen, including its resolution, color accuracy, and brightness, is important to consider when evaluating a mobile phone.
3. Camera: The quality of the phone's camera is an important factor to consider, particularly if you plan on using the phone to take photos or videos.
4. Battery life: The battery life of a phone is an important factor to consider, especially if you plan on using the phone for extended periods of time without access to a power source.
5. Operating system: The operating system of a phone is important to consider, as it will determine the types of apps that are available, as well as the overall user experience of the device.
6. Design and build quality: The overall design and build quality of a phone is also important, as it will determine how comfortable and easy the phone is to use, as well as how durable it is likely to be over time.
7. Features: The features of a phone, such as its waterproofing, fingerprint sensor, and facial recognition, are important to consider when evaluating a mobile phone.
8. Price: Finally, the price of the phone is an important factor to consider, as it will determine whether the phone is a good value for money.

# Evalution of Mobile phone

1983 1994 1996 1999 2000 2004 2007 2013 2018 2023

# Mobile operating system (OS)

A mobile operating system (OS) is a software platform that is specifically designed to run on mobile devices, such as smartphones and tablets. It manages the device's hardware and software resources and provides a platform for running apps and other services.

Mobile operating systems are designed to provide a seamless user experience, with features such as touch-based interfaces, intuitive navigation, and access to a large app store. They also provide a range of security features to protect the device and its data.

# Mobile operating system (OS)

A comparison between mobile operating systems can be based on several factors, including:

1. User interface: The way the operating system presents information and allows users to interact with their devices is important. Some operating systems have a more intuitive interface, while others offer more customization options.
2. App store: The availability and selection of apps is a crucial factor for many users. Some operating systems have a large and well-established app store, while others have a more limited selection.
3. Security: The security features and protection against malware and hacking are important for many users. Some operating systems are known for having strong security features, while others may have more vulnerabilities.
4. Hardware compatibility: Some operating systems work better on certain hardware configurations and can take advantage of specific features, while others may be more limited.
5. Updates and support: The frequency and quality of updates and support provided by the operating system's manufacturer is important for many users. Some operating systems are known for providing regular and reliable updates, while others may have a more limited lifespan.
6. Performance and battery life: The performance and efficiency of the operating system can impact the device's speed and battery life. Some operating systems are optimized for performance and battery life, while others may be more resource-intensive.

# Mobile operating system (OS)


Symbian is a mobile operating system that was developed for smartphones by Symbian Ltd., a consortium of companies that included Nokia, Ericsson, and Psion. It was one of the earliest smartphone operating systems and was widely used in the early 2000s.

Symbian was designed to run on a wide range of devices, from entry-level smartphones to high-end devices. It was known for its customization options and large app store, which made it a popular choice for early smartphone users.

# Mobile operating system (OS)


Tizen: is a free, open-source, Linux-based operating system used on a variety of devices, including smartphones, tablets, smart TVs, and wearables. Tizen was developed by the Linux Foundation and is supported by leading companies in the technology industry, such as Samsung that is used on Samsung smartwatches and other devices.

# Mobile operating system (OS)


Blackberry OS: Developed by Blackberry, Blackberry OS is the operating system used on Blackberry smartphones. It is known for its strong security features and focus on productivity.

# Mobile operating system (OS)


Windows Phone was a mobile operating system developed by Microsoft Corporation and released in 2010. It was the successor to Windows Mobile and was designed to compete with iOS and Android. Windows Phone featured a unique user interface called Metro, which was based on live tiles that could display real-time information. The platform was discontinued in 2017, and Microsoft has since shifted its focus to developing apps for iOS and Android.

# Mobile operating system (OS)


iOS: Developed by Apple, iOS is the operating system used on all of Apple's mobile devices, including iPhones and iPads. It is known for its user-friendly interface and high-quality app store.

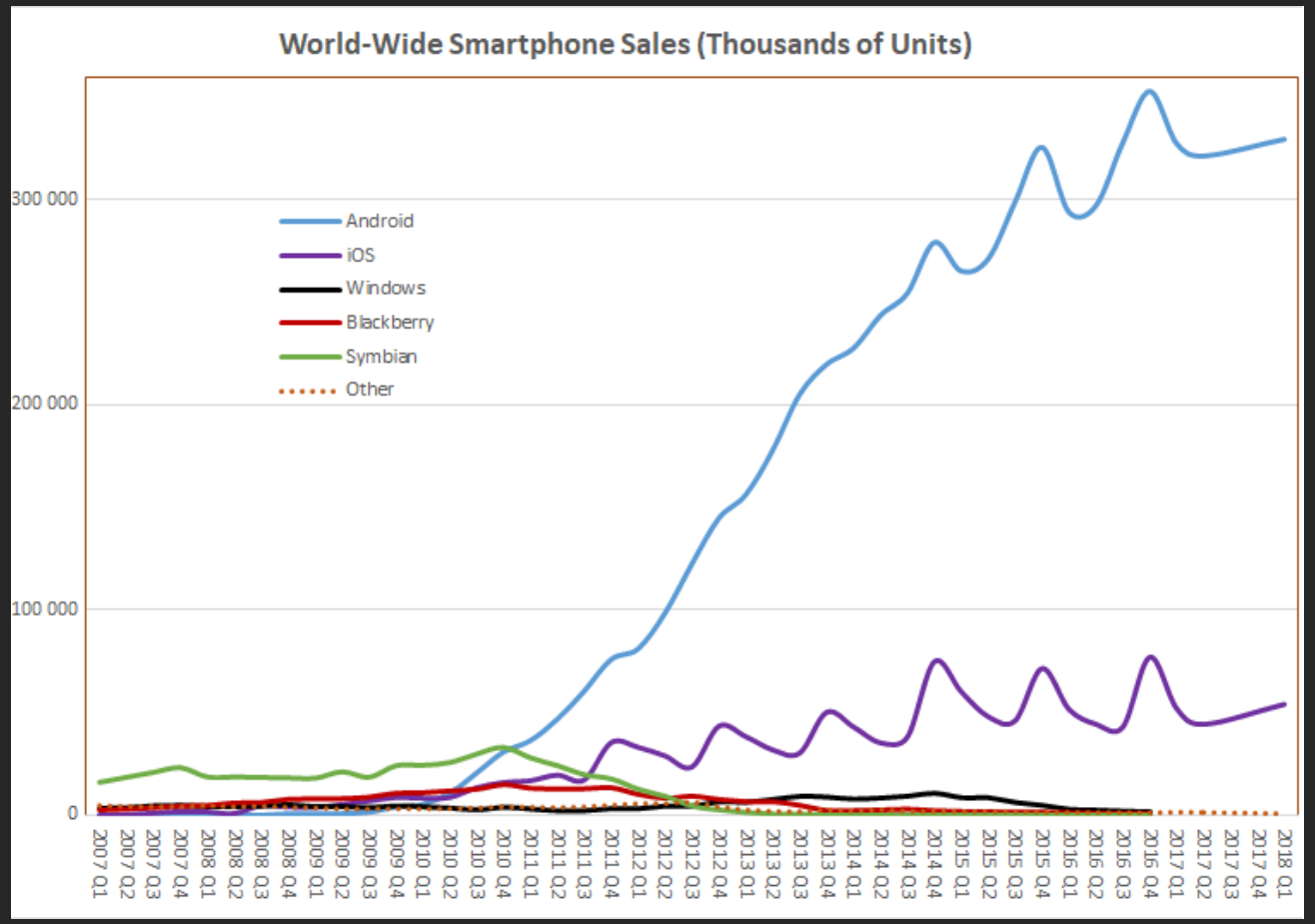
# Mobile operating system (OS)

Android: Developed by Google, Android is an open-source operating system that is used on a wide range of devices, from smartphones to tablets and smart TVs. It is known for its customization options and large app store.

# Mobile operating system (OS)


HarmonyOS is a cross-platform operating system developed by Huawei, a Chinese multinational technology company. It is designed to run on a wide range of devices, including smartphones, smart TVs, wearables, and smart home devices.

# World-wide Smartphone Sales



# Mobile apps

A mobile app, also known as a mobile application, is a software program designed to run on mobile devices such as smartphones, tablets, and wearables. These apps are created to perform a variety of functions, ranging from entertainment, education, and productivity to e-commerce, social networking, and news delivery.

Mobile apps are developed using specialized tools and programming languages, and they often utilize the device's hardware features such as the camera, GPS, and sensors to provide a seamless user experience. With the growing popularity of mobile devices, mobile apps have become an integral part of our daily lives, with millions of apps available for download on app stores worldwide.

# Mobile apps categories

Mobile apps can be broadly categorized into the following types:

1. Utility apps: These apps provide practical functions and features, such as calculators, converters, and organizers.
2. Lifestyle apps: These apps are designed to help users improve their daily life and routines, such as fitness and wellness apps, cooking apps, and travel apps.
3. Gaming apps: These apps offer interactive and engaging gaming experiences, including arcade, puzzle, and strategy games.
4. Social media apps: These apps provide users with a platform to connect and communicate with others, including social networks, messaging apps, and dating apps.
5. Education and learning apps: These apps offer educational content and resources, including language learning apps, educational games, and e-books.
6. Productivity apps: These apps help users increase their productivity and efficiency, including task management apps, note-taking apps, and project management apps.
7. Entertainment apps: These apps offer users access to movies, music, and other forms of entertainment, including streaming services and virtual reality apps.
8. Business and finance apps: These apps help users manage their finances and work, including banking apps, accounting apps, and invoicing apps.

# Mobile apps Development

Mobile App Development refers to the process of creating software applications for mobile devices such as smartphones and tablets. It involves designing, coding, testing, and deploying mobile applications that are capable of running on multiple platforms, including Android, iOS, and others.

The field of mobile app development is constantly evolving, with new technologies and development tools emerging regularly. As a result, there is a growing demand for skilled mobile app developers who can design and create high-quality, user-friendly apps that meet the needs of both individual and enterprise users.

Mobile app development is a multidisciplinary field that draws on skills from areas such as computer science, graphic design, and user experience design. It requires a strong understanding of programming languages, software development tools, and mobile platforms, as well as the ability to think creatively and problem-solve.

# Types of mobile apps development


Native App

Hyprid App



Web App

Progressive Web App

# Native app development

This involves building an app for a specific platform, such as iOS or Android, using the platform's native programming language and development tools. Native apps provide the best performance and user experience, but require separate development for each platform.

**Advantages:**

- Best performance and user experience
- Access to all native features of the platform
- Ability to use platform-specific design guidelines
- Better security due to platform-specific app stores

**Disadvantages**

- Separate development for each platform
- Higher development costs
- Longer development time
- Higher maintenance costs

# Hybrid app development

This involves using a combination of web technologies, such as HTML, CSS, and JavaScript, to create an app that can run on multiple platforms. Hybrid apps offer cross-platform compatibility, but may have limited access to native features and may not perform as well as native apps.

**Advantages:**

- Cross-platform compatibility
- Lower development costs
- Faster development time
- Easy to maintain

**Disadvantages**

- Limited access to native features
- Reduced performance compared to native apps
- Inconsistent user experience across platforms
- Dependence on third-party libraries

# Cross-platform app development

This involves using a single codebase and development tools to create an app that can run on multiple platforms. Cross-platform app development can save time and resources, but may require trade-offs in terms of performance and user experience.

**Advantages:**

- Cross-platform compatibility
- Lower development costs
- Faster development time
- Easy to maintain

**Disadvantages**

- Reduced performance compared to native apps
- Inconsistent user experience across platforms
- Dependence on third-party libraries
- Limited access to native features

# Progressive Web App (PWA) development

This involves building a web-based app that provides a native-like experience, using web technologies such as HTML, CSS, and JavaScript. PWAs can be accessed from any device with a web browser and can be installed on the user's device like a native app.

**Advantages:**

- Cross-platform compatibility
- Lower development costs
- Easy to maintain
- Available offline

**Disadvantages:**

- Reduced performance compared to native apps
- Limited access to native features
- Dependence on web technologies
- Limited discoverability compared to native apps

# Mobile App Development Platform

A Mobile App Development Platform (MADP) is a software platform that allows developers to build and deploy mobile apps across multiple platforms and devices. MADPs provide a set of tools and services that simplify the mobile app development process, including the ability to create, test, and deploy mobile apps without requiring extensive knowledge of programming languages.

MADPs typically offer a visual interface and a drag-and-drop design system, which enables developers to create apps without the need for extensive coding. Some MADPs offer integrated development environments (IDEs) that enable developers to write code using popular programming languages, such as Java, Kotlin, Swift, and JavaScript.

# Mobile App Development Platform

Here are some of the most popular Mobile App Development Platforms (MADPs) available today:

1.Flutter: Developed by Google, Flutter is an open-source mobile app development framework that allows developers to build high-quality native apps for Android, iOS, and the web.

2.React Native: Developed by Facebook, React Native is an open-source framework that enables developers to build mobile apps for both Android and iOS platforms using a single codebase.

3.Xamarin: Owned by Microsoft, Xamarin is a cross-platform development framework that allows developers to build apps for Android, iOS, and Windows platforms using C# and .NET.

4.PhoneGap: Developed by Adobe, PhoneGap is a cross-platform app development framework that allows developers to create apps for Android, iOS, and Windows using HTML, CSS, and JavaScript.

5.Ionic: An open-source framework built on top of Angular, Ionic allows developers to build mobile apps for iOS, Android, and the web using HTML, CSS, and JavaScript.

# Mobile App Development IDE

Here are some of the most popular Mobile App IDEs (Integrated Development Environments) available today:

1.Android Studio: Android Studio is the official IDE for Android app development, developed by Google and based on the IntelliJ IDEA platform.
2.Xcode: Xcode is the official IDE for iOS app development, developed by Apple.
3.Visual Studio: Visual Studio is a popular IDE developed by Microsoft that provides tools for developing a variety of applications, including mobile apps. Eclipse: Eclipse is an open-source IDE that provides tools for developing a variety of applications, including mobile apps. It supports programming languages such as Java, C++, and Python, and it provides features such as code completion, debugging, and performance analysis tools.
4.IntelliJ IDEA: IntelliJ IDEA is a popular IDE developed by JetBrains that provides tools for developing a variety of applications, including mobile apps. It supports programming languages such as Java, Kotlin, and Scala, and it provides features such as code completion, debugging, and performance analysis tools.

# Deciding on a Mobile App Development Platform

**When deciding on a mobile app development platform, there are several factors to consider. Here are some of the most important ones**

- **Target audience:** Who is your app's target audience and what devices do they use? For example, if your target audience is mainly iOS users, you may want to consider developing your app for iOS.
- **Project scope:** What features do you want to include in your app and what is your budget? Depending on your project scope, some platforms may be more suitable than others.
- **Development skills:** What development skills do you have and what skills would you need to learn to develop your app on a particular platform?
- **Time to market:** How quickly do you want to get your app to market? Some platforms offer faster development times, while others offer more features and customization options.
- **Revenue potential:** How will you generate revenue from your app and what platform offers the best revenue potential for your app?
- **Maintenance and support:** How much maintenance and support will your app require and what platform offers the best support options?
- **Market share:** What platform has the largest market share and what platform offers the most potential for growth?
- **Device compatibility:** Will your app be compatible with a variety of devices and operating systems or will it be specific to one platform?

# Chapiter II:

# Android Platform

# Android Platform

The Android Platform refers to the software environment on which Android applications run. It includes the Android operating system, which provides a set of core services and APIs that are used by applications to access device features, as well as the libraries and frameworks that developers use to build Android applications.

The Android Platform is designed to be open and flexible, with a modular architecture that allows for easy customization and integration with other software components. It includes a variety of system-level features and services, including user interface elements, multimedia support, networking, and security.

The Android Platform also includes the Android Runtime, which is responsible for executing Android applications. The Android Runtime includes the Dalvik Virtual Machine (DVM), which executes code written in the Java programming language, as well as the Android Native Development Kit (NDK), which allows developers to write native code in C or C++.

# Android OS architecture

The architecture of the Android operating system is designed to provide a high-level of abstraction over the underlying hardware and support a wide range of devices with different configurations. The Android architecture is composed of several key components, including:

1. **Linux Kernel**
2. **Native Libraries**
3. **Android Runtime**
4. **Application Framework**
5. **Applications**

# Linux Kernel

**Linux Kernel:** The Linux kernel provides the basic foundation for the Android operating system, including hardware abstraction and management, process management, memory management, and other system services.

# Native Libraries

**Native Libraries:** The native libraries provide a set of APIs for accessing the device's hardware, such as the camera, sensors, and network interfaces. The libraries are implemented in C/C++ and are tightly integrated with the Linux kernel.

# Android Runtime

**Android Runtime:** The Android runtime consists of two main components: the Dalvik virtual machine and the Core Libraries. The Dalvik virtual machine is responsible for executing the code written in Java and ensuring that each app runs in its own sandboxed environment. The Core Libraries provide a comprehensive set of APIs for developers to build their apps, including support for graphics, data storage, and networking.

# Application Framework

**Application Framework:** The Application Framework is a set of higher-level APIs that developers use to build their apps. The framework provides a rich set of services, such as Activity and Service management, Content Provider, and Notification management. The framework also provides a set of tools for managing the user interface, such as the View System, which provides a flexible and powerful way to build and manage user interfaces.

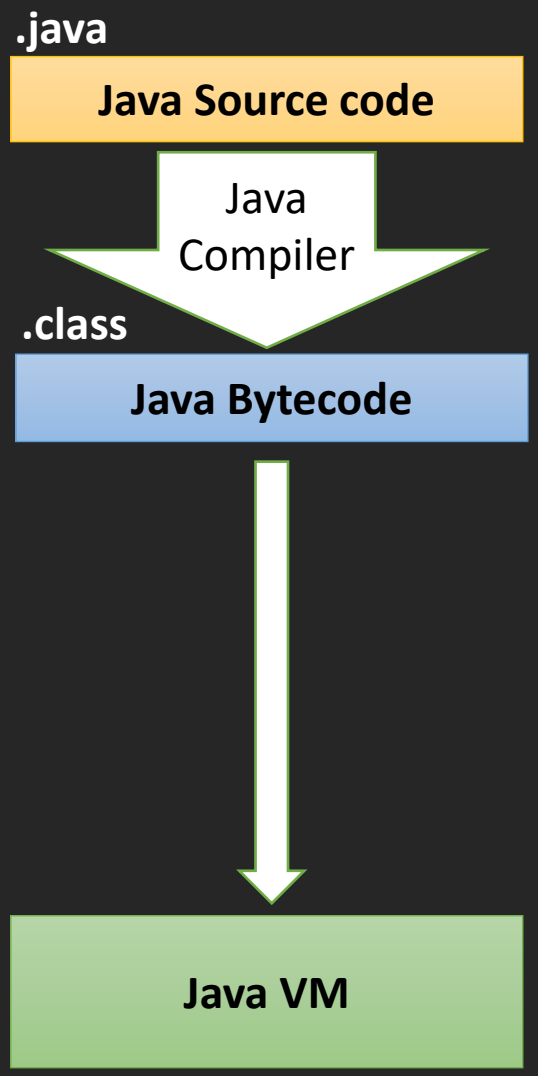
# Applications

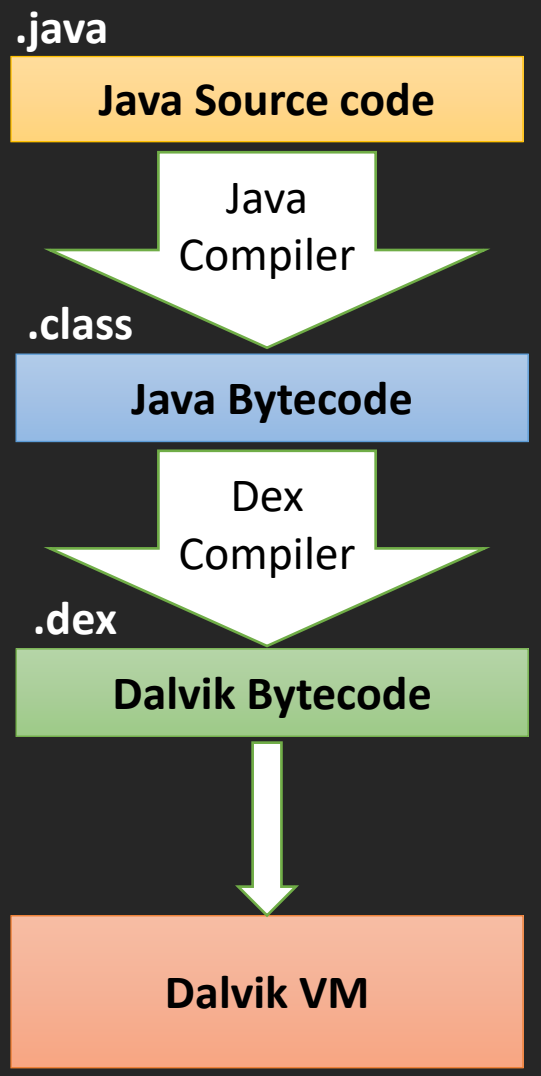
**Application layer:** The top-level component of the Android architecture is the set of apps that run on the device. These apps are built using the Application Framework and the Core Libraries, and they provide a wide range of functionality, including games, productivity tools, and social networking.

# Android app compilation process

The Android operating system uses a unique and detailed compilation process to turn the source code of an Android app into an executable package that can run on an Android device. The following is a more detailed explanation of the Android compilation process

# Android app compilation process



# Android studio

Android Studio is an Integrated Development Environment (IDE) designed specifically for developing Android applications. It was developed by Google and is based on the IntelliJ IDEA platform. Some key features of Android Studio include:

1. Code editor: The code editor provides code completion, code highlighting, and error checking to make coding easier.
2. Layout Editor: The layout editor provides a visual interface for designing the UI of an app.
3. Emulator: Android Studio includes an emulator that allows developers to test their apps on a virtual device.
4. Debugging tools: Android Studio includes a set of debugging tools that allow developers to identify and fix bugs in their code.
5. Gradle build system: Android Studio uses the Gradle build system to build, test, and deploy apps.
6. Plugins: Android Studio supports a wide range of plugins that can be used to extend the functionality of the IDE.
7. Integrated Development Environment: Android Studio provides an integrated environment that allows developers to write, test, and deploy Android apps without leaving the IDE.

# Android SDK

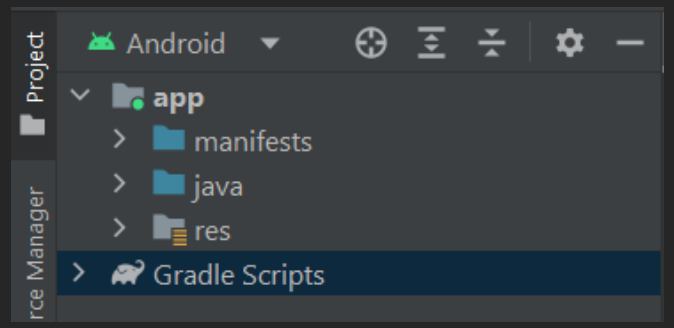
SDK stands for Software Development Kit. It is a collection of software development tools in one installable package. SDKs are used to develop applications for a specific platform, such as Android or iOS. An Android SDK includes libraries, tools, sample code, and documentations required to develop Android applications.

- The Android SDK includes the following components:

1. Android Emulator: A virtual device that runs on your computer to test your app.
2. Android SDK Tools: A set of tools including ADB (Android Debug Bridge), which allows you to communicate with an emulator or device, and AAPT(Android Asset Packaging Tool), which packages resources into an APK.
3. Android Platform-Tools: A set of tools that provide additional functionality for debugging and uploading your app.
4. Android Support Library: A set of support libraries that help you build compatibility into your app.
5. Google APIs: A set of APIs that allow you to access Google services from your app.
6. Sample code and documentation: A set of sample code and documentation to help you get started with developing Android applications.

# Android Project

These are some of the important directories and files that are included in an Android project:




1.Manifest: The AndroidManifest.xml file is the main configuration file for an Android application. It contains information about the app, such as the app name, package name, permissions required by the app, and activities that make up the app.

2.Java: This directory contains the Java source code files for the app. The main activity file is located here, as well as any other Java classes that make up the app.

3.Res: The res directory contains all of the non-code resources that are used by the app, such as images, layout files, strings, and styles.

4.Gradle Scripts: The build.gradle files contain the project and app-level settings for the Gradle build system. These files are used to configure the build process for the app, including dependencies and compilation options.

# Manifest file

The AndroidManifest.xml file is an important component of an Android project that provides essential information about the app to the Android system. It contains details about the app's package name, version, components such as activities, services, broadcast receivers, and content providers, permissions required by the app, and other metadata.

The manifest file is required for every Android app and serves as a sort of blueprint for how the app will interact with the Android operating system. It must be located in the root of the project's source folder, and it is automatically created when you create a new Android project in Android Studio.

In summary, the manifest file is a crucial component of an Android app that outlines the app's structure and defines how it interacts with the operating system.

# Manifest file

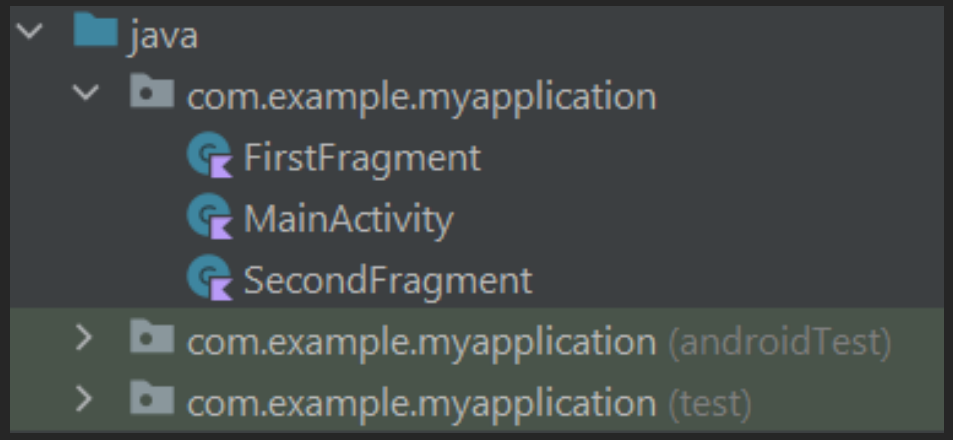
Here is an example of an AndroidManifest.xml file for a simple app:

```xml
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:name=".MyApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

This file specifies the package name of the app, any required permissions (in this case, INTERNET), the name and icon of the app, and the main activity that will be launched when the app is opened. The activity is specified with an intent filter that tells Android that this activity is the main one and should be launched when the app is opened.

# Java folder

The "java" folder in an Android project contains all the Java source code for the application. This includes all the classes and other code used to build the app's functionality. This is where developers will write the majority of the application code, including defining the user interface, managing data and storage, handling input events, and implementing other application logic. The code written in the "java" folder is compiled into bytecode that runs on the Android device, allowing the app to run and provide its functionality to users.



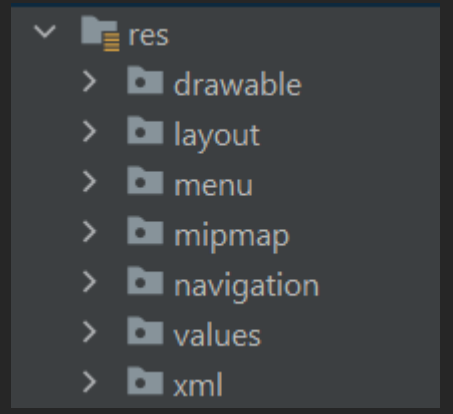
# Java folder

In an Android project, you may have several folders containing Java code, including:

- app/src/main/java: This is the main source code folder for your project, where you write your application code. It contains the MainActivity.java file, which is the entry point of your application.
- app/src/androidTest/java: This is the folder where you write your Android instrumentation tests. These are tests that are designed to run on an Android device or emulator.
- app/src/test/java: This is the folder where you write your unit tests. These tests are designed to run on the JVM, without any Android dependencies.

These folders are organized based on the type of code you are writing and the type of tests you are performing. By separating the code into different folders, it is easier to manage and maintain your codebase.

# Res folder




The res folder in an Android project contains all the resources used by the application, such as layout files, images, strings, and more. Here are some of the common types of resources you'll find in the res folder:

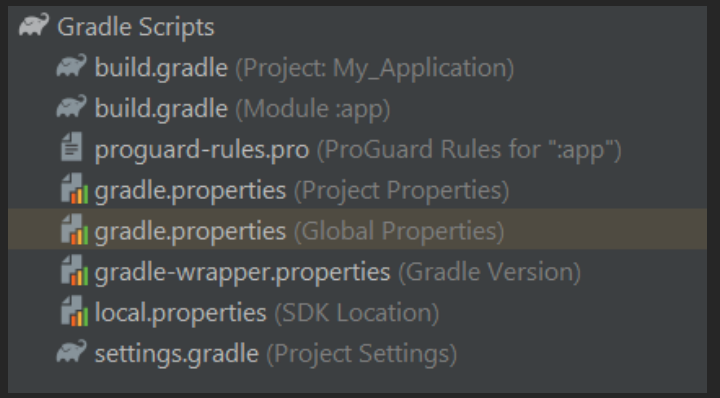
- layout: Contains the XML files that define the user interface of the app.

drawable: Contains images and other resources used in the app.

- values: Contains XML files that define various values used throughout the app, such as strings, colors, and dimensions.
- anim: Contains XML files that define animations used in the app.
- menu: Contains XML files that define menus used in the app.

These are just a few examples of the resources you might find in the res folder. The specific contents of this folder can vary depending on the needs of your application.

# Gradle scripts **folder**




Gradle is a build automation tool that is used to build and manage Android projects. In the Android Studio project, there are several Gradle scripts:

1.build.gradle (Project): This script applies to the entire project and is used to configure build settings for all the modules in the project.
2.build.gradle (Module): This script applies to the specific module in the project and is used to configure build settings for that module.
3.gradle.properties: This file contains the properties that configure the build system itself.
4.settings.gradle: This file is used to specify which modules to include in the project.
These Gradle scripts play a crucial role in building and managing Android projects. They define the dependencies, plugins, and configurations needed to build the project.

# Chapiter III:

# Android mobile app activities and resources

# Introduction to Activities and Resources

In the context of mobile app development, activities and resources are essential concepts that form the core of the user interface and application functionality. Activities represent different screens or windows in the app, while resources are the visual and audio assets that populate the app's interface.

# Definition of Activities

In Android, an activity is a component of an app that represents a single screen with a user interface. It can contain UI elements such as buttons, text fields, images, and other interactive widgets. Activities are designed to handle user interactions, such as button clicks, and to respond to user input. They are an important part of the Android app architecture and provide a way for developers to create dynamic and interactive user experiences.

# Definition of Resources

In Android, resources refer to the files and static content that an app uses, such as images, layouts, strings, colors, and styles. Resources are compiled and packaged within the app, making it easy for developers to access and use them in their code. They can be accessed and used in different parts of an app, including the UI, code, and manifest file, among others. Using resources in an app ensures consistency, reduces repetition, and simplifies maintenance.

# Creating Activities

To create an Activity in Android Studio, follow these steps:

1.Open your project in Android Studio.

2.Right-click on the app folder in the Project view, then select New -> Activity -> Empty Activity.

3.In the dialog that appears, enter the name of the Activity in the Activity name field.

4.Choose the layout and menu options you want for the Activity.

5.Click Finish to create the Activity.

After you create the Activity, you can add code to it in the Java file and create the layout in the XML file.

# Creating Activities

To create an Activity in Android Studio, follow these steps:

1.Open your project in Android Studio.

2.Right-click on the app folder in the Project view, then select New -> Activity -> Empty Activity.

3.In the dialog that appears, enter the name of the Activity in the Activity name field.

4.Choose the layout and menu options you want for the Activity.

5.Click Finish to create the Activity.

After you create the Activity, you can add code to it in the Java file and create the layout in the XML file.

# Activity Lifecycle

The Activity Lifecycle refers to the different states that an activity can be in at any given time while it's running. It's important for Android developers to understand the Activity Lifecycle in order to manage the behavior of their app as it transitions between different states.

The Activity Lifecycle is composed of the following methods:

1.onCreate(): This method is called when the activity is first created. It's where you initialize the activity, including setting up the UI and other components.

2.onStart(): This method is called when the activity becomes visible to the user. It's where you can start animations or other visual effects.

3.onResume(): This method is called when the activity is ready to interact with the user. It's where you can start or resume any running processes.

4.onPause(): This method is called when the activity is about to lose focus. It's where you should save any unsaved data or stop any running processes.

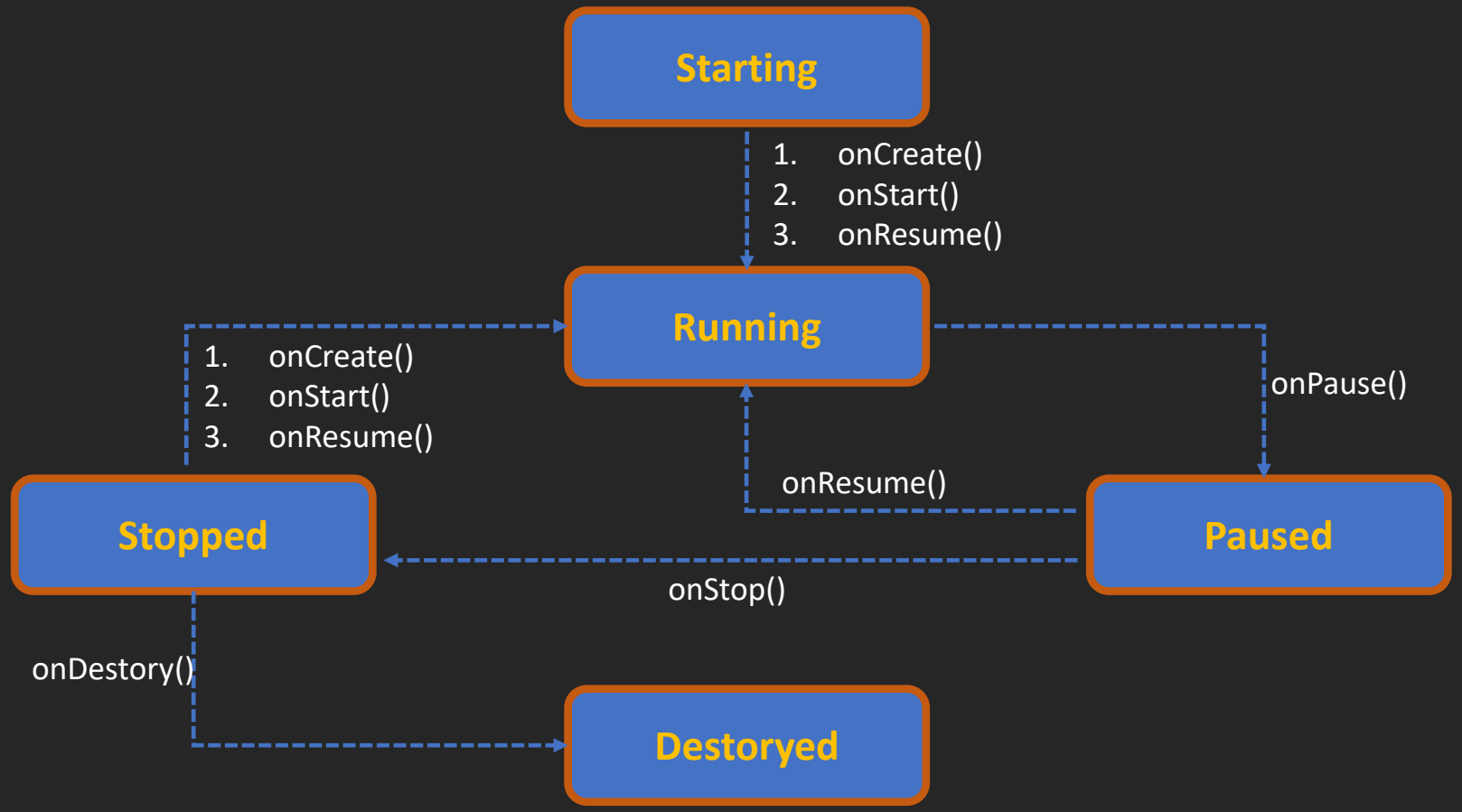
5.onStop(): This method is called when the activity is no longer visible to the user. It's where you can stop any running processes that are not needed.

6.onRestart(): This method is called when the activity is being restarted from a stopped state.

7.onDestroy(): This method is called when the activity is being destroyed. It's where you should release any resources that are no longer needed.

By understanding the Activity Lifecycle and the methods that are called during each phase, you can create more responsive and reliable apps that behave predictably for users.

# Activity Lifecycle



# Activity Class Example

```java
public class MainActivity extends AppCompatActivity {

    private static final String TAG = "MainActivity";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.i(TAG, "onCreate() called");
    }

    @Override
    protected void onStart() {
        super.onStart();
        Log.i(TAG, "onStart() called");
    }

    @Override
    protected void onResume() {
        super.onResume();
        Log.i(TAG, "onResume() called");
    }

    @Override
    protected void onPause() {
        super.onPause();
        Log.i(TAG, "onPause() called");
    }

    @Override
    protected void onStop() {
        super.onStop();
        Log.i(TAG, "onStop() called");
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.i(TAG, "onDestroy() called");
    }

    @Override
    protected void onRestart() {
        super.onRestart();
        Log.i(TAG, "onRestart() called");
    }
}
```

# Managing multiple Activities

Managing multiple activities refers to how the app handles the different states of various activities in the application, such as creating, pausing, resuming, and destroying. The app can navigate between different activities using an intent, which starts a new activity and passes the control to the new activity.

To manage multiple activities, the app developer needs to use different methods provided by the Android OS, such as startActivity(), finish(), and onActivityResult(), etc. The startActivity() method starts a new activity, the finish() method is used to finish the current activity, and the onActivityResult() method is called when the activity returns a result.

The application can keep track of the different states of each activity by implementing the Activity class's various callback methods, which include onCreate(), onStart(), onResume(), onPause(), onStop(), onRestart(), and onDestroy(). The app can use these methods to save the activity's state when it is paused or destroyed and to restore it when it is resumed.

To manage multiple activities efficiently, the app developer needs to design the app in such a way that it uses resources efficiently, such as memory and CPU usage, and minimizes the number of activities to reduce the overhead of starting and stopping activities.

# Managing multiple Activities

To manage multiple activities in an Android app, you can use the following approaches:

Start an Activity: You can start a new activity by creating an Intent and calling the startActivity() method. This will create a new instance of the activity and add it to the top of the activity stack.

Finish an Activity: When you no longer need an activity, you can call the finish() method to remove it from the activity stack.

Pass data between Activities: You can pass data between activities by adding extras to the intent that starts the new activity, or by using a Bundle object.

Launch modes: You can use launch modes to control how new activities are launched and how they interact with existing activities in the activity stack.

Back stack: You can manage the back stack of activities by setting flags on the intent that starts the new activity, or by using the TaskStackBuilder class to build an artificial back stack.

# Definition of an Activity in the AndroidManifest.xml file.

In the AndroidManifest.xml file, the activity is defined as an <activity> tag that contains information about the activity, such as its class name, the application label, icon, required permission, intent filters, etc. This tag is placed in the root <application> element.

Here's an example of an <activity> tag in the AndroidManifest.xml file:

```
<activity
    android:name=".MainActivity"
    android:label="@string/app_name"
    android:icon="@drawable/app_icon"
    android:permission="android.permission.ACCESS_FINE_LOCATION">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

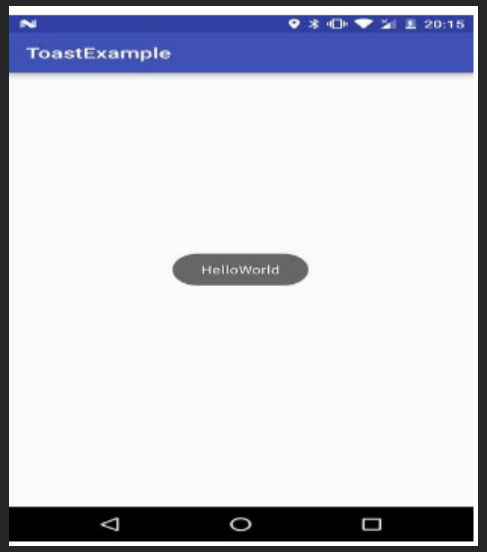
In this example, the activity is named "MainActivity" and is defined with a label and an icon. It also requires the "ACCESS_FINE_LOCATION" permission. Additionally, the <intent-filter> tag specifies that this activity is the main (launcher) activity of the application, which will be launched when the user clicks on the application icon on the home screen.

# Definition of an Activity in the AndroidManifest.xml file.

Here's an example of an Android application with a single activity that displays the message "Hello" in a Toast:

In the MainActivity.java file, add the following code:

```java
import android.os.Bundle;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        // Display the message "Hello" in a Toast
        Toast.makeText(this, "Hello", Toast.LENGTH_SHORT).show();
    }
}
```




In this example, the MainActivity activity is defined in the AndroidManifest.xml file as the main (launcher) activity of the application. When the application is launched, the activity is created and the message "Hello" is displayed in a Toast.

# Resources

In Android app development, resources refer to elements such as images, layout files, string files, styles, colors, dimensions, and other files used to build the user interface and app features.

These resources are organized by type and language configuration and can be easily referenced from the source code. They are often used in conjunction with XML layouts to define the appearance and behavior of UI components. Resources are important for creating user-friendly and customizable apps for different configurations.

# Resources

Here is a table that lists some of the common types of resources used in Android app development:

| Resource Type | Description |
|---|---|
| Images | PNG, JPEG, and GIF files used to display pictures, logos, icons, and other visual elements within the app. |
| Layouts | XML files that define the structure and appearance of the user interface, including the placement of UI components such as buttons, text fields, and images. |
| Strings | Text strings used within the app, such as menu items, button labels, and error messages, can be stored in a string resource file. |
| Styles | A style resource can be defined to apply a consistent look and feel to UI elements throughout the app. |
| Colors | A color resource can be defined to store color values that can be applied to UI elements such as backgrounds, text, and borders. |
| Dimensions | A dimension resource can be defined to store sizes such as height, width, and margins that can be used in the layout files. |
| Drawables | Resources that define graphic images and animation. |
| Animations | Resources that define how to perform tweened animations. |
| Menus | Resource files that define app menus and options. |
| Values | Resources that define general values, such as integers, booleans, and arrays. |

This is not an exhaustive list, but it covers some of the most commonly used resource types in Android app development.

# Resources

Here's an example of how you might use some of these resource types in an Android app:

Images: You can include an image in your app by adding it to the "drawable" folder in your app's resources. For example, if you have an image file called "my_image.png", you can reference it in your layout XML file like this:

```
<ImageView
    android:src="@drawable/my_image"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

# Resources

Strings: You can define a string resource in the "strings.xml" file in your app's resources. For example:

```
<string name="hello_world">Hello World!</string>
```

You can then reference this string in your layout XML file or Java code like this:

```
<TextView
    android:text="@string/hello_world"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content" />
```

# Resources

Colors: You can define a color resource in the "colors.xml" file in your app's resources. For example:

```
<color name="my_color">#ff0000</color>
```

You can then reference this color in your layout XML file or Java code like this:

```
<Button
android:text="Click me!"
android:background="@color/my_color"
android:layout_width="wrap_content"
android:layout_height="wrap_content" />
```

# Resources

Access resources in Java code:

Strings: You can access a string resource in Java code using the getString() method of the Resources class. For example:

```
String myString = getResources().getString(R.string.hello_world);
```

Colors: You can access a color resource in Java code using the getColor() method of the Resources class. For example:

```
int myColor = getResources().getColor(R.color.my_color);
```

Dimensions: You can access a dimension resource in Java code using the getDimension() method of the Resources class. For example:

```
float myDimension = getResources().getDimension(R.dimen.my_dimension);
```

# Resources

You can access an image resource in Java code using the getDrawable() method of the Resources class. For example:

```
Drawable myImage = getResources().getDrawable(R.drawable.my_image);
```

This will retrieve the image with the name "my_image" from the "drawable" folder in your app's resources. Note that the getDrawable() method was deprecated in API level 22. In newer versions of Android, you should use the getDrawable() method of the ContextCompat class instead, like this:

```
Drawable myImage = ContextCompat.getDrawable(this, R.drawable.my_image);
```

This will retrieve the image with the name "my_image" from the "drawable" folder in your app's resources, and the this parameter should be replaced with a reference to your Activity or Context. Once you have the Drawable object, you can use it to set the image of an ImageView or other UI element in your app.

# Resources

You can apply a style to a view or a group of views in Java code using the setStyle() method of the View class. Here's an example:

```
TextView myTextView = findViewById(R.id.my_text_view);
myTextView.setStyle(R.style.MyTextStyle);
```

This will apply the style with the name "MyTextStyle" to the myTextView view. The style should be defined in your app's resources, in the "styles.xml" file. Here's an example of a style definition:

```
<style name="MyTextStyle">
    <item name="android:textColor">#FFFFFF</item>
    <item name="android:textSize">18sp</item>
    <item name="android:fontFamily">@font/my_custom_font</item>
</style>
```

# Chapiter IV:

# User Interface (UI)

# Introduction

The User Interface (UI) in Android refers to the visual and interactive components of an Android application that allow users to interact with the app and perform various tasks. The UI is an essential part of any Android app, as it determines how users will interact with the app and how they will perceive its functionality and usability.

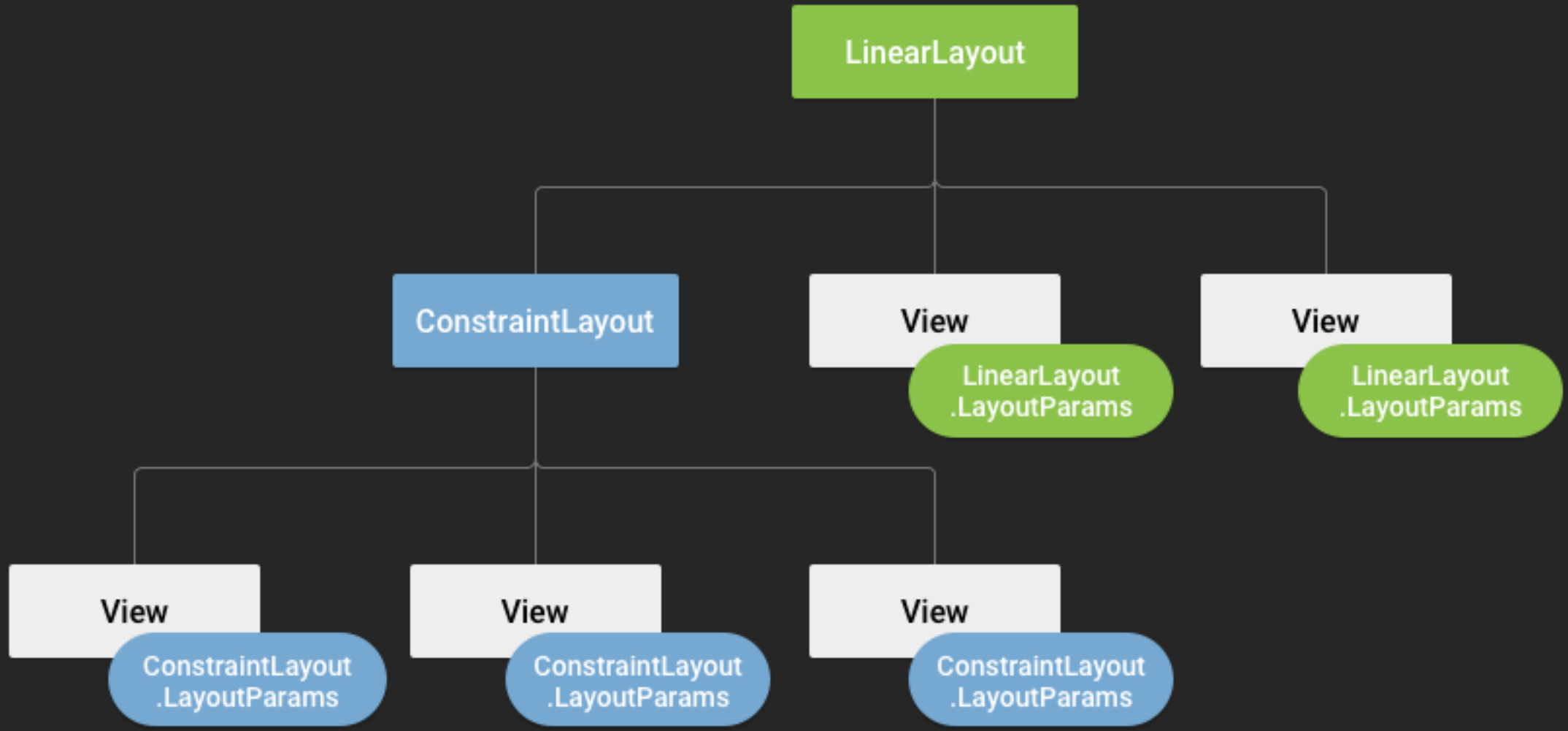
The Android UI is based on the Material Design guidelines developed by Google, which provide a set of design principles and best practices for creating beautiful, intuitive, and consistent user interfaces across all Android devices.

# Layouts

Layouts are an essential part of the Android user interface (UI) design. A layout defines the visual structure for a user interface in an Android app, such as where UI elements should be placed and how they should be arranged. In other words, layouts define the look and feel of an app and its user interface.

Android provides a variety of layout types that developers can choose from to design their app's UI. Each layout has its own unique set of properties, and selecting the right layout for your app can make a significant difference in the overall user experience. Understanding the different types of layouts and how to use them effectively is crucial for creating well-designed and user-friendly apps.

# The general structure of a layout in Android



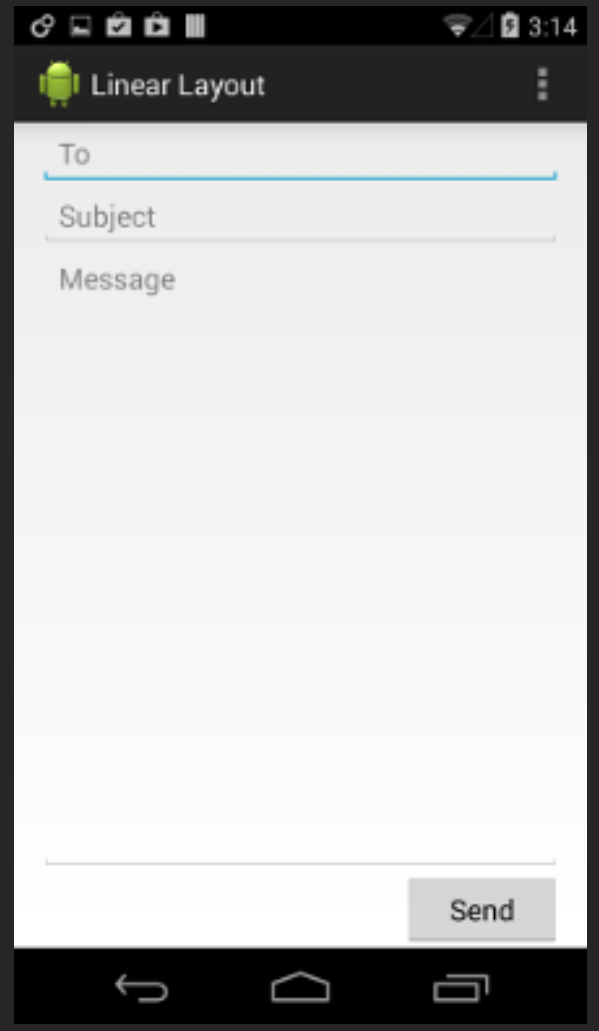
# Types of Layouts

There are several types of layouts available in Android, each with its own unique characteristics and intended use cases. Here are some of the most commonly used layouts:

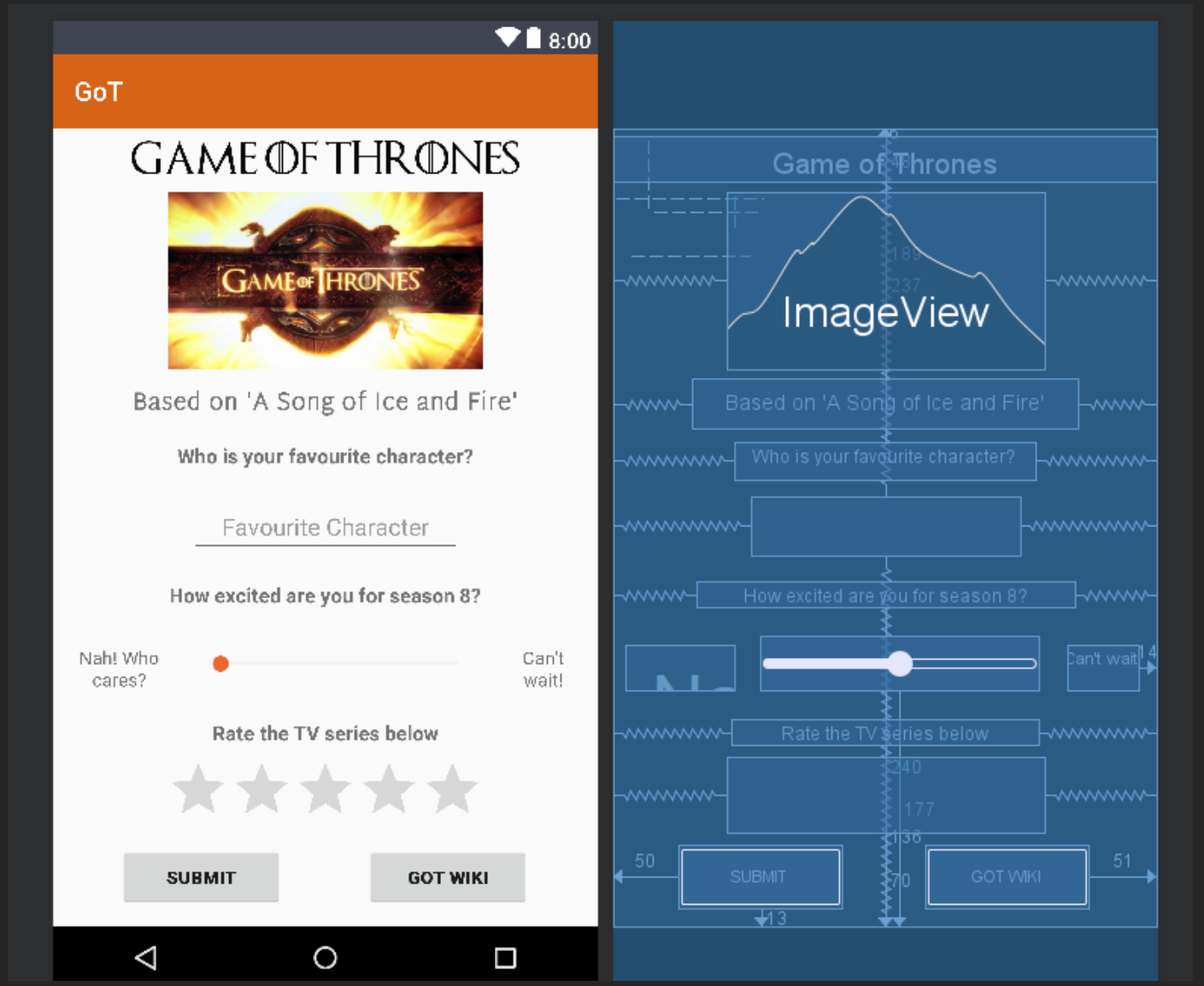
| Layout Type | Description | Use Cases |
|---|---|---|
| Linear Layout | Arranges UI elements in a single row or column | Forms, lists, simple UI designs |
| Relative Layout | Arranges UI elements relative to one another or to the parent container | Complex UI designs that require precise positioning of elements |
| Constraint Layout | Similar to a relative layout but allows for more precise positioning of UI elements | Complex UI designs with elements that need to be constrained to one another |
| Table Layout | Arranges UI elements in rows and columns, similar to an HTML table | Displaying tabular data |
| Grid Layout | Arranges UI elements in a grid-like structure, with a fixed number of rows and columns | Displaying multiple items in a grid-like pattern |
| Frame Layout | Allows for only one child view to be displayed at a time | Displaying a single view that takes up the entire screen |

# Layout Example: Linear Layout

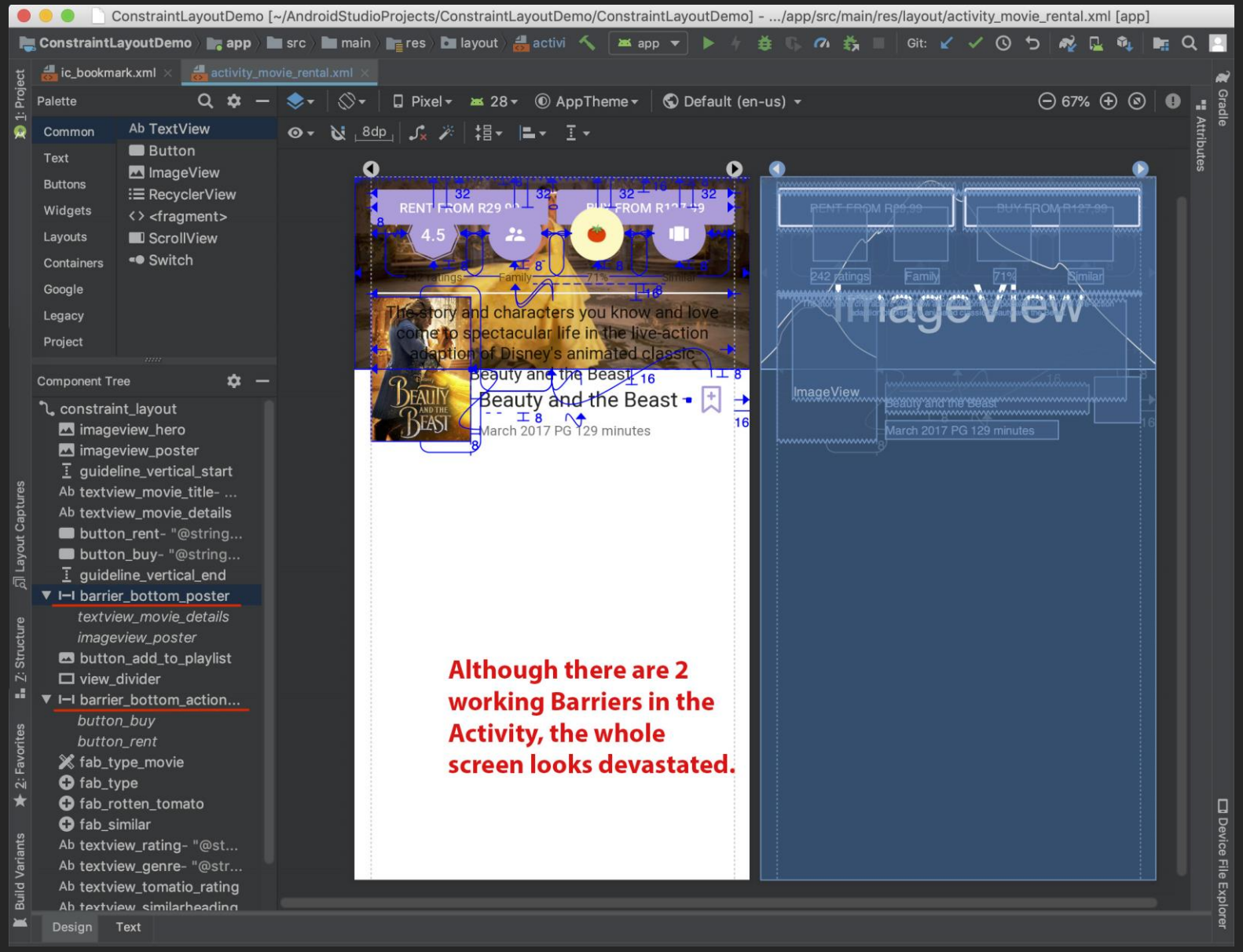
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:orientation="vertical" >
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/to" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="@string/subject" />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:gravity="top"
        android:hint="@string/message" />
    <Button
        android:layout_width="100dp"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:text="@string/send" />
</LinearLayout>
```

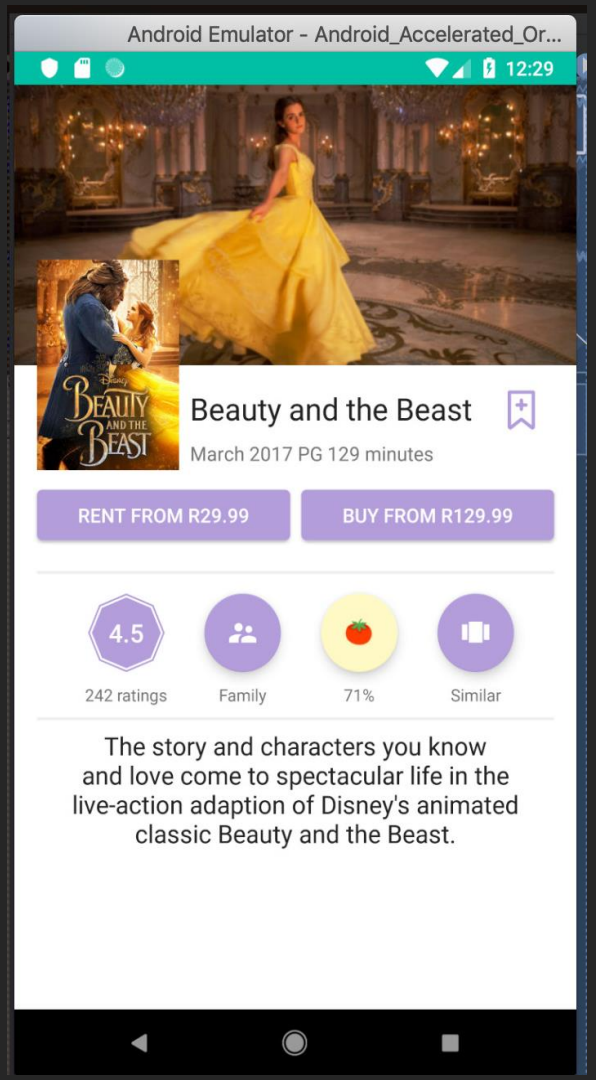


# Layout Example: Relative Layout



# Layout Example: Constraint Layout



# Views

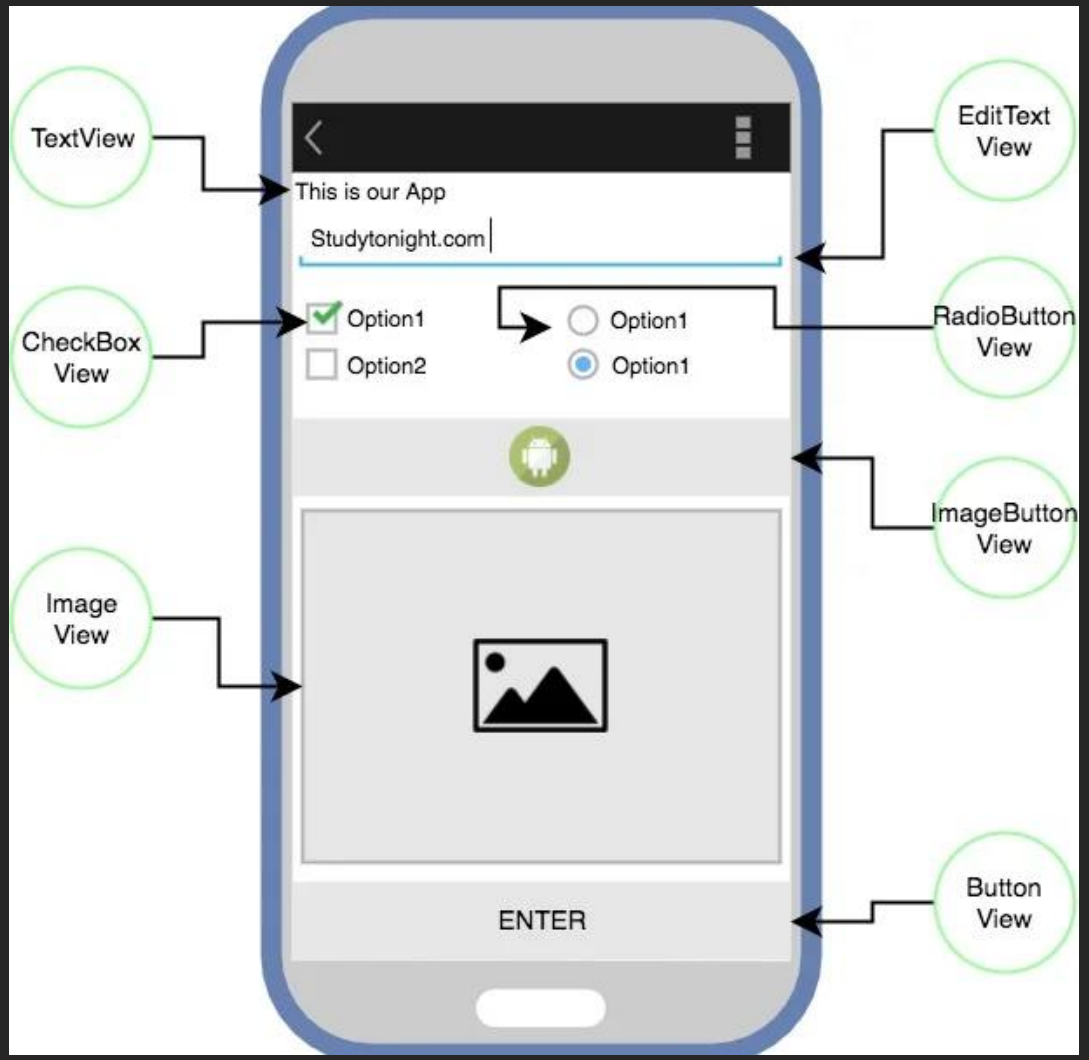
In Android app development, Views refer to the visual elements or components that make up the user interface (UI) of an app. Views can be simple elements such as text, images, and buttons, or more complex elements such as lists, grids, and web views.

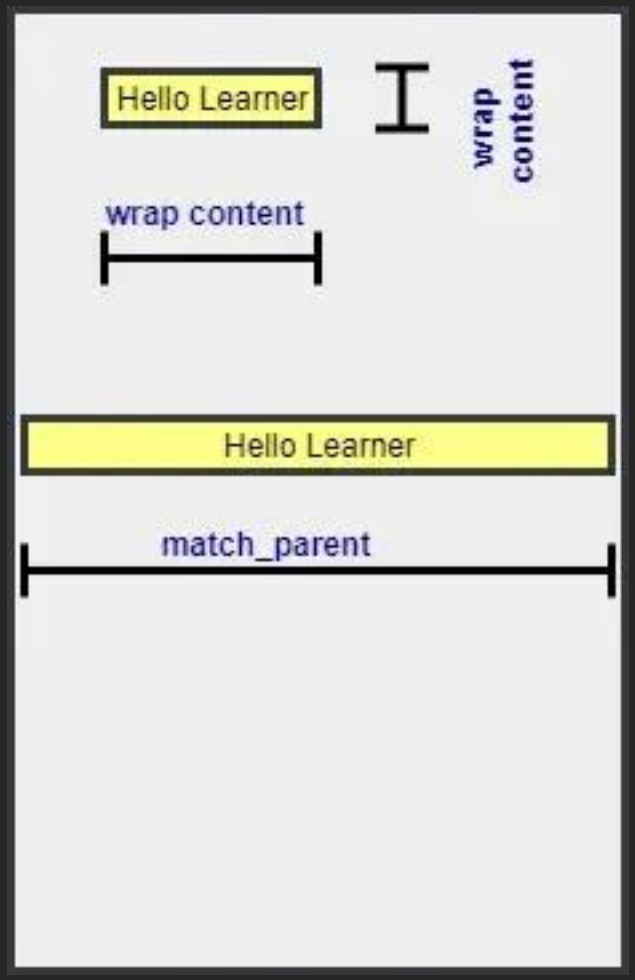
Views are implemented using the View class and its subclasses such as TextView, EditText, Button, ImageView, ListView, and others. Each View has its own properties and methods that define its behavior and appearance, and they can be customized to meet specific requirements.

# Types of views

| View | Description |
|---|---|
| Button | A clickable button that performs an action when clicked |
| TextView | A non-editable text view that displays text to the user |
| EditText | A view that allows the user to input text |
| ImageView | A view that displays an image resource |
| CheckBox | A view that allows the user to select one or more items from a set of options |
| RadioButton | A view that allows the user to select one option from a set of options |
| Spinner | A view that displays a dropdown list of options for the user to select from |
| SeekBar | A view that allows the user to select a value from a range of values by sliding a thumb along a horizontal track |
| RatingBar | A view that allows the user to rate something by selecting a number of stars |
| ProgressBar | A view that displays a progress indicator, typically used to indicate the progress of a background task |
| Switch | A view that allows the user to toggle a binary setting |
| ToggleButton | A view that allows the user to toggle a binary setting, similar to a switch but with a different appearance |
| WebView | A view that displays web content within the app |

# Views Examples



# Manipulating of UI

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="15dp"
    android:text="TextView"
    tools:layout_editor_absoluteX="115dp"
    tools:layout_editor_absoluteY="334dp" />
```

Manipulating the  he different UI elements and modifying  d behavior. This can be done by obtaining  eir assigned IDs and using methods provi

For example, to modify the text of a TextView element, we can use the setText() method, as shown in the following code snippet:

```
TextView textView = findViewById(R.id.textViewId);
textView.setText("New Text");
```

# Manipulating of UI

Similarly, to set the visibility of a View element to be hidden, we can use the setVisibility() method with the View.INVISIBLE constant, like this:

```
View view = findViewById(R.id.viewId);
view.setVisibility(View.INVISIBLE);
```

# Manipulating of UI

To handle user input, we can attach event listeners to UI elements using setOnClickListener() method, like this:

```
Button button = findViewById(R.id.buttonId);
button.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        // Handle button click event
    }
});
```

# Event listeners in Android

| Event Listener | Description |
|---|---|
| OnClickListener | Responds to clicks on a view |
| OnLongClickListener | Responds to long clicks on a view |
| OnTouchListener | Responds to touch events on a view |
| OnKeyListener | Responds to key presses |
| OnFocusChangeListener | Responds to changes in focus |
| OnCheckedChangeListener | Responds to changes in the checked state of a compound button |
| OnSeekBarChangeListener | Responds to changes in the progress of a seek bar |
| OnItemSelectedLister | Responds to the selection of an item in a spinner |
| TextWatcher | Responds to changes in the text of an editable view |
| OnEditorActionListener | Responds to editor actions, such as "done" or "next" |
| GestureDetector.OnGestureListener | Responds to gestures, such as flings, swipes, and taps |
| GestureDetector.OnDoubleTapListener | Responds to double tap gestures |
| ScaleGestureDetector.OnScaleGestureListener | Responds to scaling gestures |
| View.OnDragListener | Responds to drag and drop events |
| View.OnLayoutChangeListener | Responds to changes in the layout of a view |
| ViewTreeObserver.OnGlobalLayoutListener | Responds to global layout changes, such as changes in the size or position of a view |
| ViewTreeObserver.OnScrollChangedListener | Responds to changes in the scrolling position of a view |
| ViewTreeObserver.OnPreDrawListener | Responds before a view is drawn, allowing for modifications to the view tree |
| ViewTreeObserver.OnWindowFocusChangeListener | Responds to changes in the focus state of a window |
| View.OnSystemUiVisibilityChangeListener | Responds to changes in the system UI visibility, such as when the status bar or navigation bar is shown or hidden |

# Chapiter V:

# Intents and Broadcast Receivers

# Introduction

Intents and Broadcast Receivers are fundamental components of the Android application framework. Intents allow applications to request actions from other components, such as starting activities or services, while Broadcast Receivers enable applications to receive and respond to broadcast messages sent by other applications or the system.

# Intents

**Intents** are asynchronous messages which allow application components to request functionality from other Android components. Intents **allow you to interact** with components from the same applications as well as with components contributed by other applications.

For example, an activity can start an external activity for taking a picture.

Intents are **objects** of the android.content.Intent type. Your code can send them to the Android system defining the components you are targeting.

**For example**, via the startActivity() method you can define that the intent should be used to start an activity.

# Intent Filters

| Component | Description |
| --- | --- |
| Action | Specifies the general type of operation that the Intent is requesting. For example, "ACTION_VIEW" is used to request that the system display data to the user. |
| Data | Specifies the URI or data that the Intent should operate on. For example, a web URL for a browser Intent. |
| Category | Provides additional information about the Intent's action or component. For example, "CATEGORY_BROWSABLE" for a browser Intent. |
| Component | Specifies the explicit class name of the component that should handle the Intent. Used when you want to launch a specific component within your application. |
| Extras | Provides additional data to the component that receives the Intent, such as key-value pairs of data. |

# Example: Intent Filters

```
// Create a new Intent with action and data
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.example.com"));

// Set category and component
intent.addCategory(Intent.CATEGORY_BROWSABLE);
intent.setComponent(new ComponentName("com.android.browser", "com.android.browser.BrowserActivity"));

// Add extras
intent.putExtra("key1", "value1");
intent.putExtra("key2", 123);

// Start the activity
startActivity(intent);
```

# Intent Filters: Action (1)

The Intent class defines a number of action constants, including these:

| Constant | Target component | Action |
|---|---|---|
| ACTION_CALL | **activity** | Initiate a phone call. |
| ACTION_EDIT | **activity** | Display data for the user to edit. |
| ACTION_MAIN | **activity** | Start up as the initial activity of a task, with no data input and no returned output. |
| ACTION_SYNC | **activity** | Synchronize data on a server with data on the mobile device. |

# Intent Filters: Action (2)

The Intent class defines a number of action constants, including these:

| Constant | Target component | Action |
|---|---|---|
| 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. |
| ACTION_SCREEN_ON | **Broadcast Receiver** | The screen has been turned on. |
| ACTION_TIMEZONE_CHANGED | **Broadcast Receiver** | The setting for the time zone has changed. |

# Intent Filters: Data (1)

The Intent class defines a number of action constants, including these:

- The URI of the data to be acted on and the MIME type of that data. Different actions are paired with different kinds of data specifications.

- **The URI:** At the highest level a URI reference (Uniform Resource Identifiers) in string form has the syntax
  - [scheme:]scheme-specific-part[#fragment]
  - For example:

```
mailto:java-net@java.sun.com
news:comp.lang.java
```

- Short for Multipurpose Internet Mail Extensions, a specification for formatting non-ASCII messages so that they can be sent over the Internet. For Examples:
  - For Text  "text/plain"
  - For Image :"image/jpeg“ , "image/bmp“ ,"image/gif“ , "image/jpg“ ,"image/png"
  - For Video: "video/wav“, "video/mp4"

# Intent Filters: Data (2)

**Some examples of Action/Data Pair**

| | Action/Data Pair & Description |
|---|---|
| 1 | **ACTION_VIEW content://contacts/people/1**<br>Display information about the person whose identifier is "1". |
| 2 | **ACTION_DIAL content://contacts/people/1**<br>Display the phone dialer with the person filled in. |
| 3 | **ACTION_VIEW tel:123**<br>Display the phone dialer with the given number filled in. |
| 4 | **ACTION_DIAL tel:123**<br>Display the phone dialer with the given number filled in. |
| 5 | **ACTION_EDIT content://contacts/people/1**<br>Edit information about the person whose identifier is "1". |

# Intent Filters: Category

**Some examples of Action/Data Pair**

- The Intent class defines several category constants, including these (More than 29 category):

| Constant | Meaning |
|---|---|
| CATEGORY_BROWSABLE | The target activity can be safely invoked by the browser to display data referenced by a link — for example, an image or an e-mail message. |
| CATEGORY_DEFAULT | Set if the activity should be an option for the default action (center press) to perform on a piece of data. |
| CATEGORY_APP_MAPS | Used with ACTION_MAIN to launch the maps application. |
| CATEGORY_LAUNCHER | an activity on the top of stack, whenever application will start, the activity containing this category will be opened first.. |

# Intent Filters: Extras

- This will be in **KEY-VALUE PAIRS** for additional information that should be delivered to the component handling the intent. The extras can be set and read using the **putExtras**() and **getExtras**() methods respectively (More than 26 Extras):
- **For examples** :
    - **EXTRA_EMAIL**: A String[] holding e-mail addresses that should be delivered to.
    - **EXTRA_HTML_TEXT**: A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.

```
Intent i=new Intent(context,SendMessage.class);
i.putExtra("id", user.getUserAccountId()+"");
 i.putExtra("name", user.getUserFullName());
context.startActivity(i);
```

# Intent Filters: Flags (1)

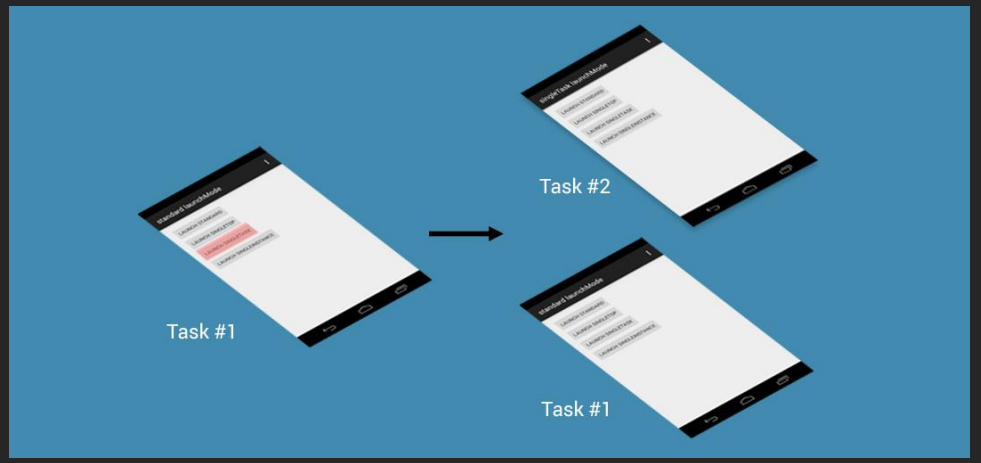
- Flags defined in the Intent class that function as metadata for the intent. The flags may instruct the Android system how to launch an activity and how to treat it after it's launched. (More than 33  Flags):
  - FLAG_ACTIVITY_CLEAR_TASK - Clear any existing tasks on this stack before starting the activity.
  - FLAG_ACTIVITY_NEW_TASK - Start the activity in a new task or reuse an existing task tied to that activity.

```
Intent i=new Intent(this, Sample.class);
i.setFlags(Intent. FLAG_ACTIVITY_NEW_TASK);
startActivity(i);
```

# Intent Filters: Flags (2)

- How FLAG_ACTIVITY_NEW_TASK work:
  - With normal intent StartActivity


  - With FLAG_ACTIVITY_NEW_TASK:




# Intent Filters: <intent-filter> Manifest tag

- A single **IntentFilter** may include more than one *<action>,<category> and <data>*. In this case the target component must be able to handle any or all combination of Intents.
- To allow several specific combinations of *<action>, <category> and <data>*, define multiple **IntentFilters.**
- Each **IntentFilter** will be tested against an incoming Intent to satisfy the *<action>, <category> and <data>* tests. The Intent will be delivered to the component only if it can pass all the tests for at least one **IntentFilter.**
- In order to receive implicit intents, you must include the CATEGORY_DEFAULT category in the intent filter

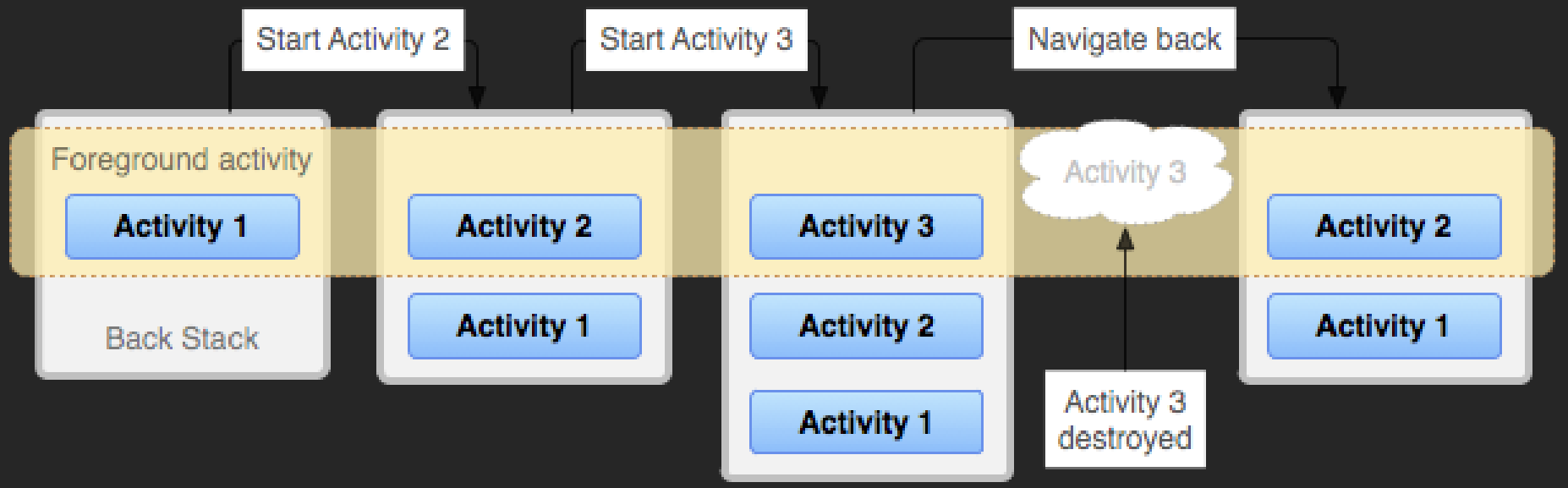
```
<activity android:name="TestActivity">
    <intent-filter>
        <action android:name="android.intent.action.SEND"/>
        <category android:name="android.intent.category.DEFAULT"/>
        <data android:mimeType="text/plain"/>
    </intent-filter>
</activity>
```

# Intent Filters: User Permissions

- In order to access external components in android, permissions needs to be set in AndroidManifest.xml file.

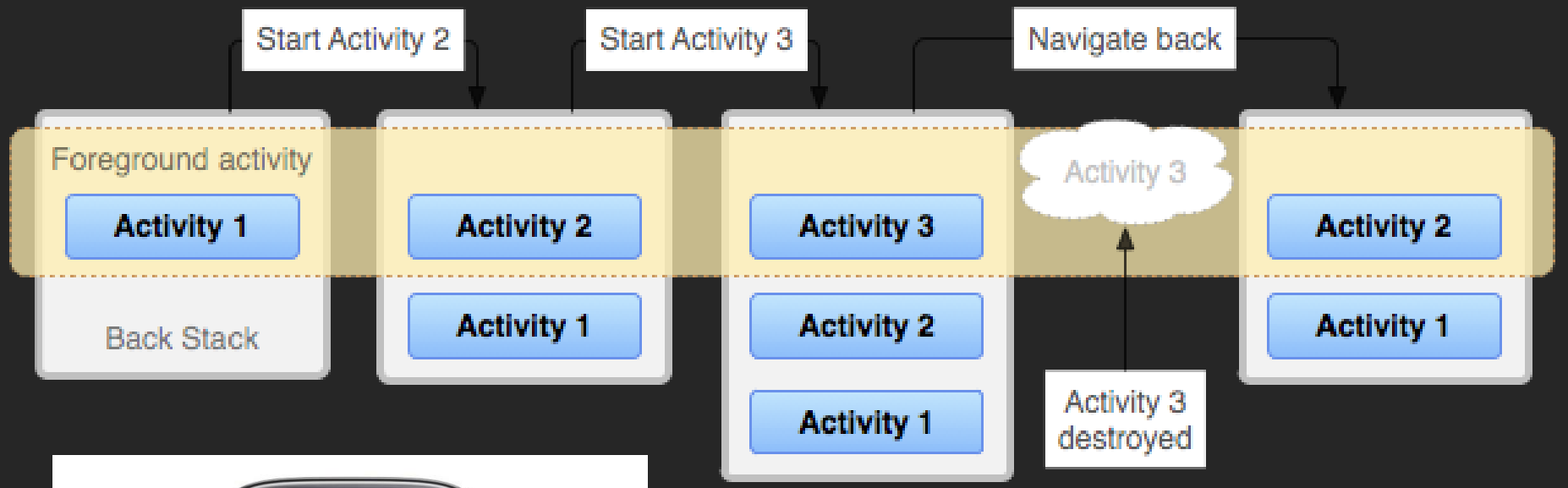
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.commonintent"
    android:versionCode="1"
    android:versionName="1.0" >
   <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="19" />
    <uses-permission android:name="android.permission.CALL_PHONE"/>
    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"/>
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <uses-permission android:name="android.permission.INTERNET"/>
</manifest>
```

# Activity Stack




- The activities are arranged in a stack, called task, in the order in which each activity is opened.
- When the current activity starts another, the new activity is pushed on the top of the stack and takes focus. The previous activity remains in the stack, but is stopped.

# Activity Stack





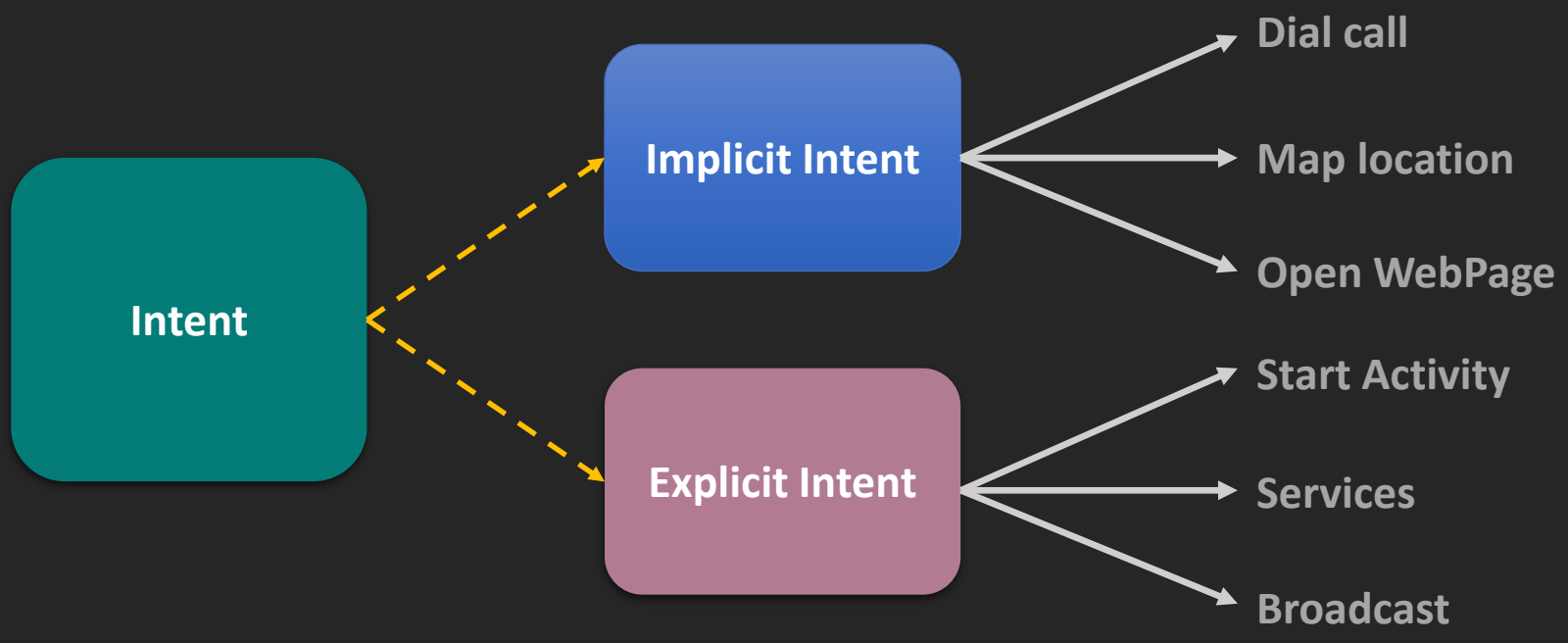
- When the user presses the *Back* button, the current activity is popped from the top of the stack and the previous activity resumes

# Types of Intents

Android supports two types of intents: explicit or implicit

**Explicit intents** must know the Java class name of the component that they want to start

**Implicit intents** must know the action type they want to be done for them



# Implicit Intents

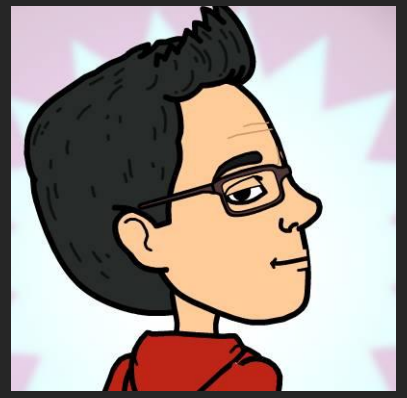
Implicit Intents: These Intents are used to request that the system perform a specific action, such as viewing a web page, taking a photo, or playing a video. With an implicit Intent, you specify the action that you want to perform (using the Action property), along with any additional data or categories that are required. The system then searches for a suitable component that can handle the Intent. For example, you might use an implicit Intent to open a web page:

```
Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse("https://www.example.com"));
startActivity(intent);
```

Implicit Intents are more flexible than Explicit Intents, because they allow you to request that the system perform a specific action without knowing in advance which application will handle the request. However, they can also be less predictable, because the system might offer multiple applications that can handle the same action.

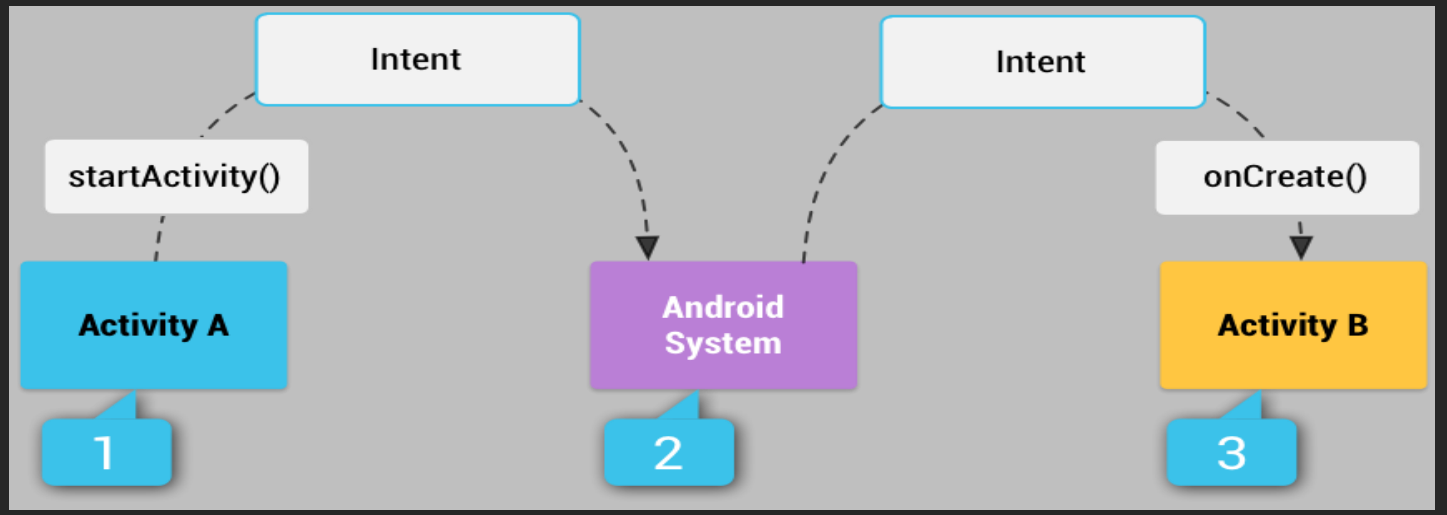
# Implicit Intents: Example

- Implicit intent identifies the activities by describing the required services on the specified data



Can you direct me to someone, who can help me setup the home Internet, and I am using cable connection, living in EL ALIA , Biskra

# Implicit Intents: Example




1. Activity A creates an Intent with an Action description and passes it to **`startActivity().`**
2. The Android System searches all apps for an intent filter that matches the intent.
3. When a match is found, the system starts the matching activity (Activity B) by invoking its **`onCreate()`** method and passing it the Intent.

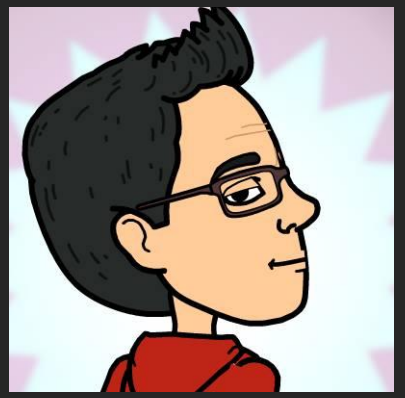
# Explicit Intents

Explicit Intents: These Intents are used to start a specific component, such as an Activity, Service, or BroadcastReceiver, within your own application. With an explicit Intent, you specify the component's name (using the Component property) and the exact package name of the application that contains the component. For example, you might use an explicit Intent to start a specific Activity within your application:

```
Intent intent = new Intent(this, MyActivity.class);
startActivity(intent);
```

# Explicit Intents: Example

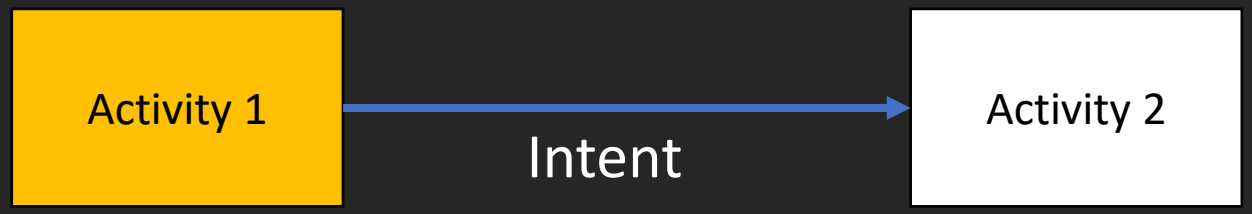
- Assuming that you need to get to a AT technician to help you set up your home Internet.



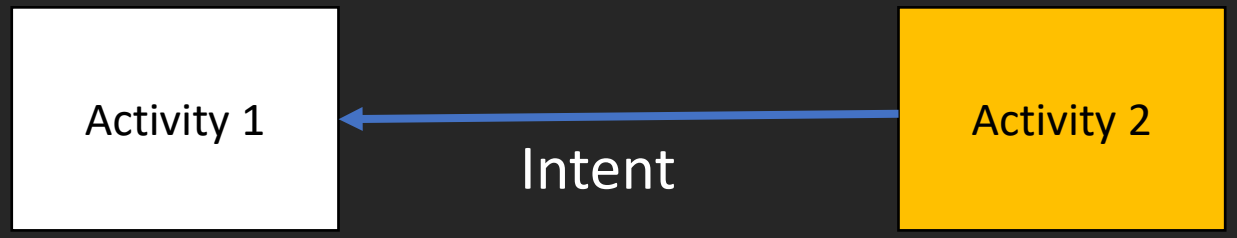
Can you direct me to Mohamed in technician group?

Android OS

# Explicit Intents: Start an Activity with Intent




```
Intent intent = new Intent(this, MyActivity.class);
startActivity(intent);
```




```
Intent AnotherIntent = new Intent(Activity2.this, Activity1.class);
startActivity(AnotherIntent);
```

# Methods of the Intent class

| Method Name | Description |
|---|---|
| setAction(String action) | Sets the action for the Intent, specifying what kind of action should be performed. |
| setData(Uri data) | Sets the data for the Intent, specifying the data that should be acted upon. |
| setType(String type) | Sets the MIME type for the Intent, specifying the type of data that is being passed. |
| setComponent(ComponentName component) | Sets the component for the Intent, specifying the name of the component that should be started. |
| addCategory(String category) | Adds a category to the Intent, specifying additional information about the action to be performed. |
| putExtra(String name, boolean value) | Adds a boolean extra to the Intent. |
| putExtra(String name, int value) | Adds an integer extra to the Intent. |
| putExtra(String name, String value) | Adds a string extra to the Intent. |
| getAction() | Returns the action for the Intent. |
| getData() | Returns the data for the Intent. |
| getType() | Returns the MIME type for the Intent. |
| getComponent() | Returns the component for the Intent. |
| getCategories() | Returns the categories for the Intent. |
| getExtras() | Returns the extras for the Intent. |

# Passing Data between Activities

Activity 1

Intent

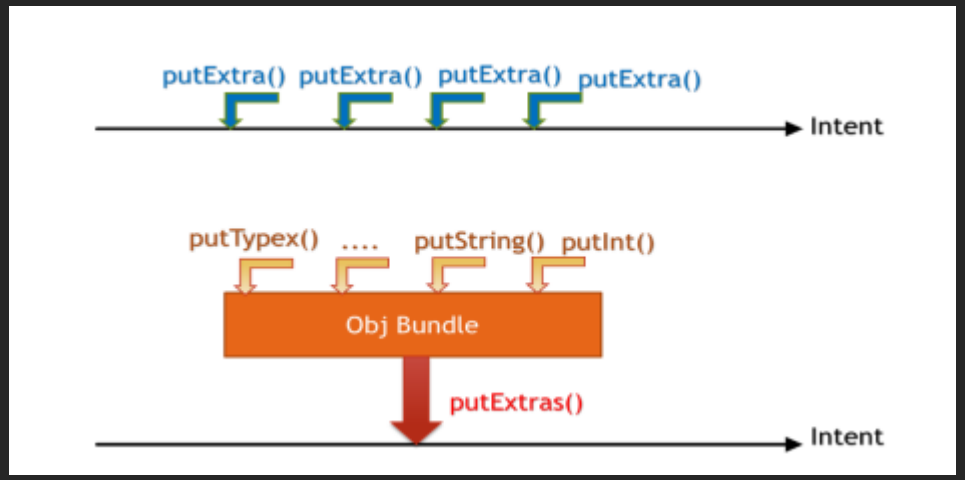
Activity 2

```
Intent intent = new Intent(Activity1.this, Activity2.class);
Bundle bundle = new Bundle();
bundle.putString("key_name", "name");
bundle.putString("key_age", "age");

intent.putExtras(bundle);
intent.putExtra("key_id", "id");
intent.putExtra("key_address", "address");
startActivity(intent);
```

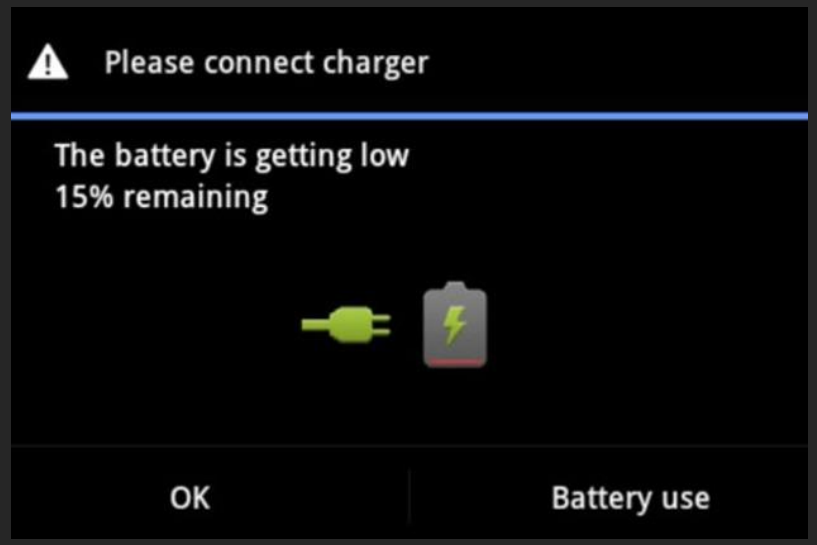
putXXX() to load data

```
// In Activity2.java
// Retrieve the intent
Intent intent  = getIntent();
// Retrieve the string data in the intent
String name = intent.getStringExtra("key_name");
String age = intent.getStringExtra("key_age");
String id = intent.getStringExtra("key_id");
String address =
intent.getStringExtra("key_address");
```

getXXX() to retrieve data



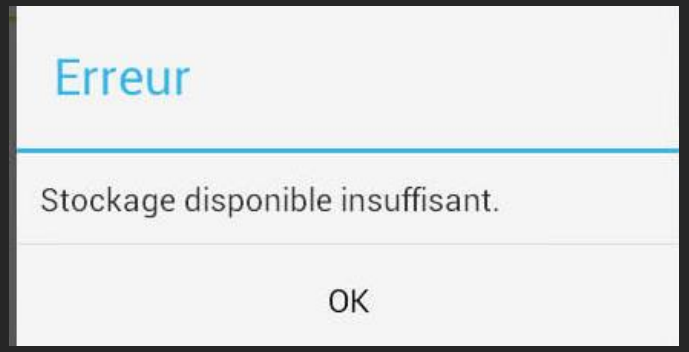
# Broadcast Receivers




For instance, a Broadcast receiver triggers battery Low notification that you see on your mobile screen.

Other instances caused by a Broadcast Receiver are new friend notifications, new friend feeds, new message etc. on your Facebook app.

In fact, you see broadcast receivers at work all the time. Notifications like incoming messages, WIFI Activated/Deactivated message etc. are all real-time announcements of what is happening in the Android system and the applications.





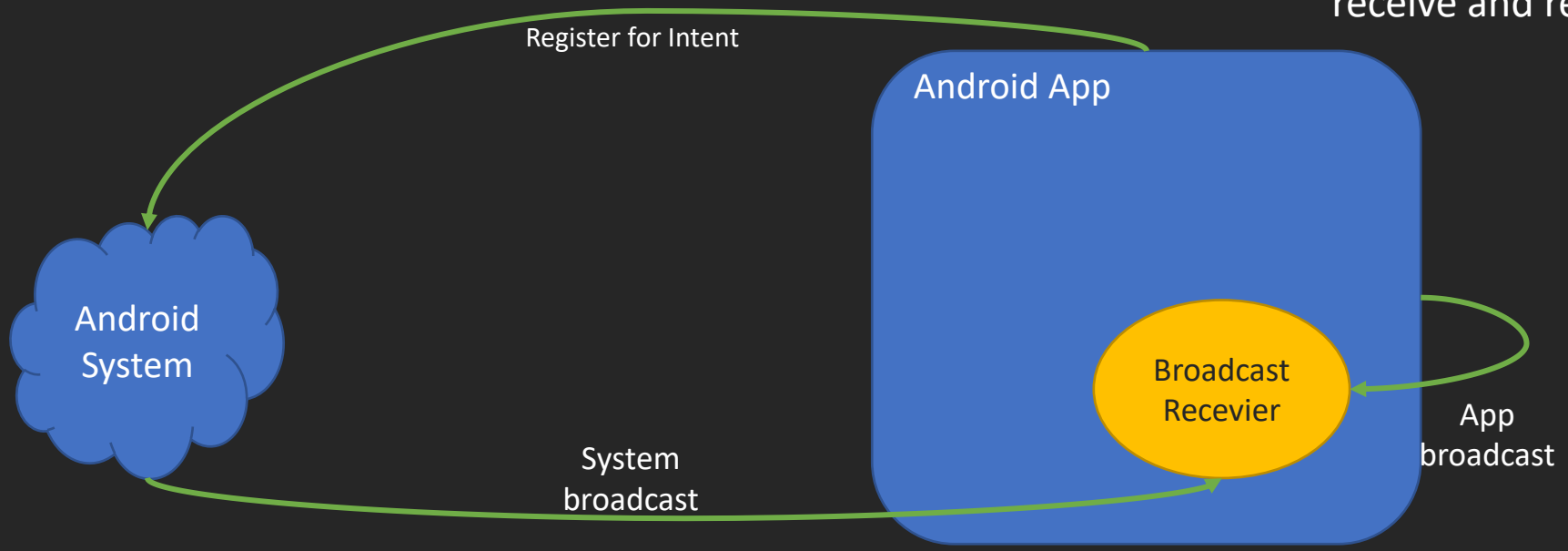
- Facebook Messages / Incomming Text Messages
- Image Downloading Complete
- Wife Connected / Disconnected

# Broadcast Receivers

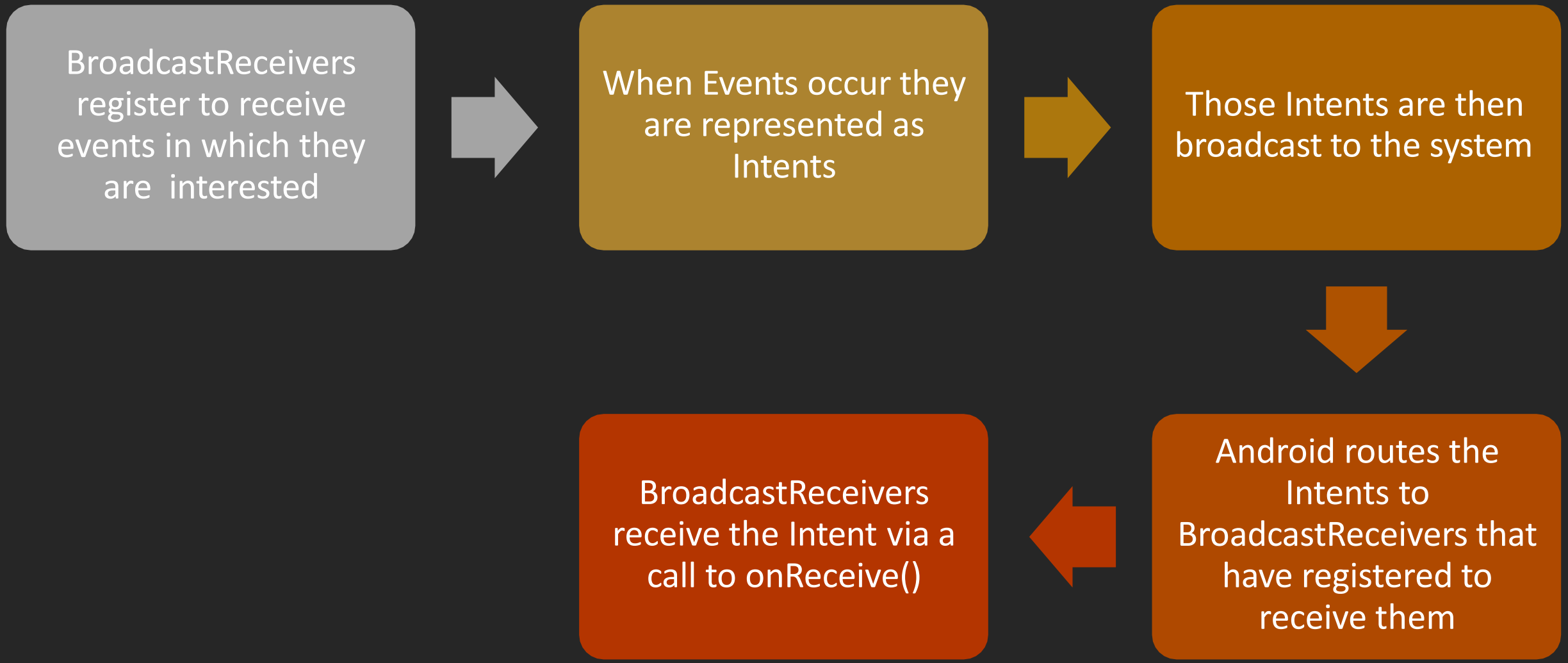
A broadcast receiver (short receiver ) is an Android component which allows you to register for system or application events. All registered  receivers for an event are notified by the Android runtime once this event happens.

For example, applications can register for the ACTION_BOOT_COMPLETED system event which is fired once the Android system has completed the boot process.

What is Broadcast Reciever...?
Base class for components that receive and react to events



# Broadcast Receivers Process



# Broadcast Receivers Registration

## Static Registration

Android Manifest

Put <receiver> and <intent-filter>tags in AndroidManifest.xml

## Dynamic Registration

Create receiver object

Set When broadcast event will fire

Register new broadcast receive

Reciever class which extends BroadcastReceiver

# Implementing the Broadcast Receiver

1. **Create a subclass of Android's BroadcastReceiver**
2. **Implement the onReceive() method:**

```
public class MyBroadcastReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    // Handle the received broadcast here
  }
}
```

- Following are the two arguments of the onReceive() method:
  - **Context:** This is used to access additional information, or to start services or activities.
  - **Intent:** The Intent object is used to register the receiver.

```
<receiver android:name=".MyBroadcastReceiver">
  <intent-filter>
    <action
android:name="android.intent.action.ACTION_NAME" />
  </intent-filter>
</receiver>
```

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <!-- Other elements -->

    <application>

        <!-- Other elements -->

        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
            </intent-filter>
        </receiver>

    </application>
</manifest>
```

# Implementing the Broadcast Receiver

Send a broadcast from within your app or from another app. To send a broadcast from within your app, use the following code:

```
Intent intent = new Intent();
intent.setAction("android.intent.action.ACTION_NAME");
sendBroadcast(intent);
```

# Broadcast Event

The following table lists a few important system events :

| Broadcast Event | Description |
|---|---|
| android.intent.action.BOOT_COMPLETED | Sent when the device has completed booting up. |
| android.intent.action.BATTERY_CHANGED | Sent when the battery status changes, such as when the device is unplugged or the battery level changes. |
| android.net.conn.CONNECTIVITY_CHANGE | Sent when the network state changes, such as when the device connects to or disconnects from a network. |
| android.intent.action.PACKAGE_ADDED | Sent when an app is installed on the device. |
| android.intent.action.PACKAGE_REMOVED | Sent when an app is uninstalled from the device. |
| android.intent.action.PHONE_STATE | Sent when the phone state changes, such as when a call is received or the call is ended. |
| android.intent.action.SCREEN_ON | Sent when the screen is turned on. |
| android.intent.action.SCREEN_OFF | Sent when the screen is turned off. |
| android.intent.action.TIMEZONE_CHANGED | Sent when the timezone on the device changes. |

# Chapiter VI:
# Data storage and Data manipulation
# (SQLite)

# Introduction

Data storage is a critical aspect of mobile application development, as it determines how applications store and manage data. Android provides developers with several options for storing and managing data, ranging from internal storage to cloud-based solutions. Choosing the right data storage solution for an application depends on various factors such as the type and amount of data that needs to be stored, performance requirements, and the need for real-time synchronization.

# Data storage methods

**Shared Preferences**

Used to sore the data in Key Value paire

**File Storage**

Used to store raw files on memory card or SD-card

- External file storage
- Internal file storage

**Content Providers**

Used to store semi-structured data with user configurable data access

**Cloud Storage**

Used to store third party data storage , that involves Parse API Google…firebase ….

**Database**

Used to store private data for the app here the structured data

- SQL (SQLite)
- NoSQL (Firebase)

# SQLite

- SQLite is a software library that provides full relational database capability for Android applications.
- Each application that uses SQLite has its own instance of the database, which is by default accessible only from the application itself.
- The database is stored in the following folder:

  **/data/data/<package-name>/databases**
- A Content Provider can be used to share the database information with other applications.

# Features of SQLite

- ACID transactions (atomic, consistent, isolated, and durable)
- Zero-configuration – no setup or administration.
- Implements most of SQL92.
- Complete database is stored in a single file.
- Supports terabyte-sized databases and gigabyte-sized strings and blobs.
- Small code footprint (less than 325KB fully configured).
- Self-contained: no external dependencies.
- C source code is in the public domain.

# SQLite Datatypes

- Most SQL database systems use static typing;  i.e., the type of a value is determined by the column in which the value is stored.
- SQLite uses a more general dynamic type system – the  type of a value is associated with the value itself, not with its column.
- Think of column types as hints.  It is possible to store a string in an integer column and vice versa.

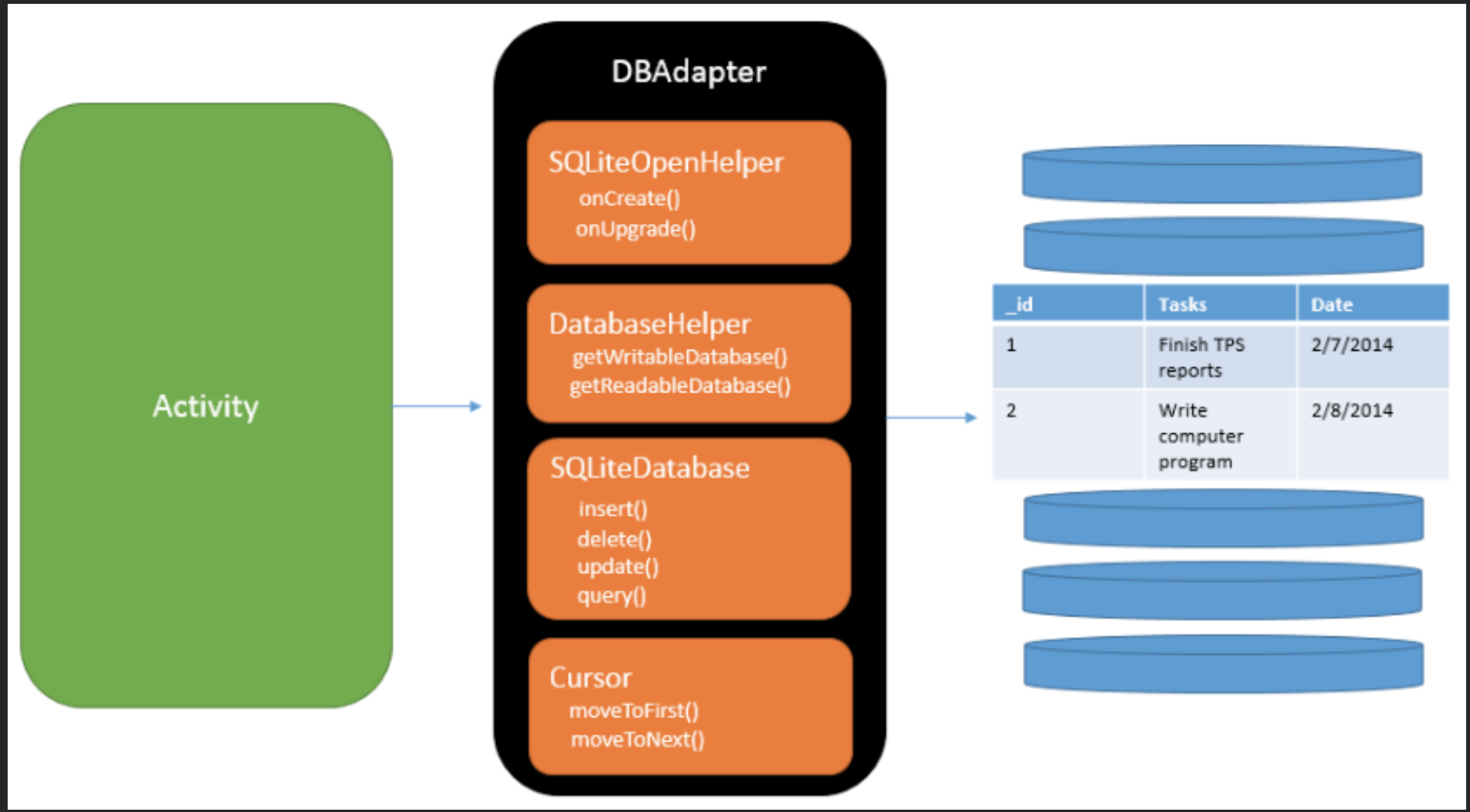
# SQLite Datatypes

- Each value stored in an SQLite database has one of the following storage classes:
  - NULL. The value is a NULL value.
  - INTEGER. The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
  - REAL. The value is a floating point value, stored as an 8-byte IEEE floating point number.
  - TEXT. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
  - BLOB. The value is a blob of data, stored exactly as it was input.
- Note that a storage class is slightly more general than a type.

# SQLite Datatypes

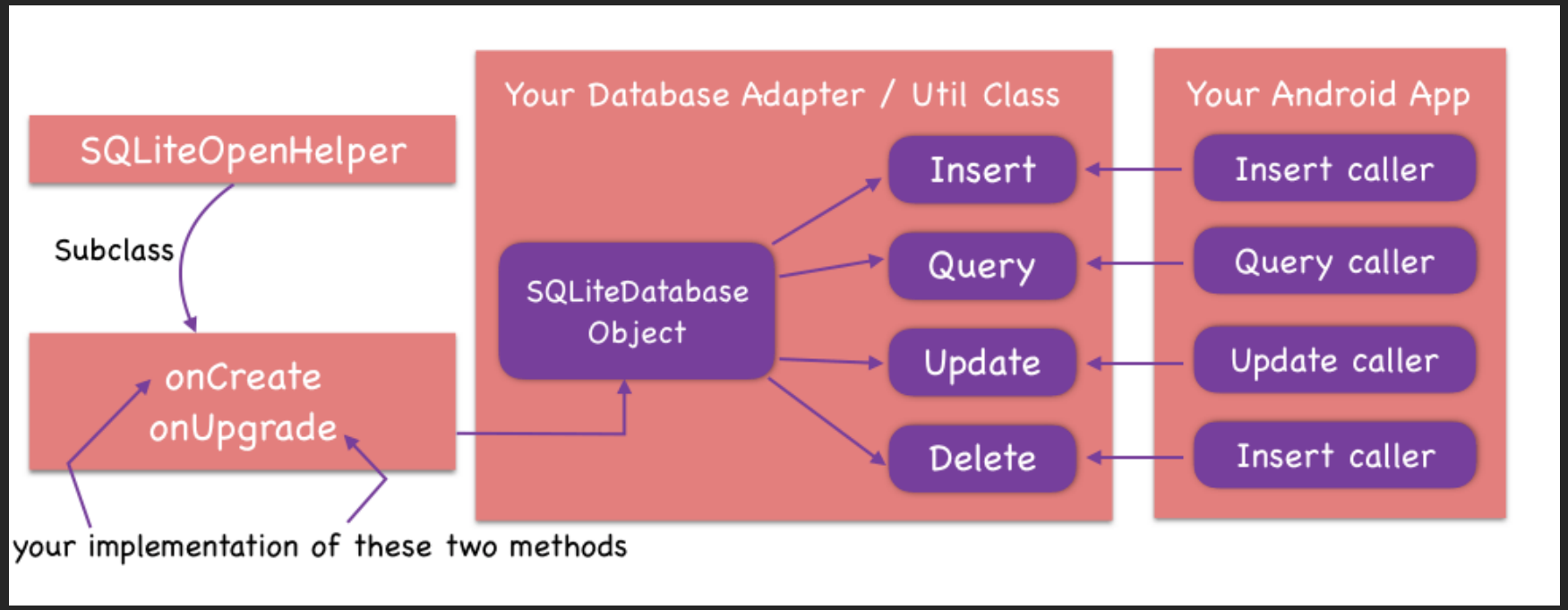
- Each column in a SQLite database is assigned one of the following type affinities:
  - TEXT
  - INTEGER
  - NONE
  - NUMERIC
  - REAL
- SQLite does not have a separate Boolean storage class. Boolean values are stored as integers 0 (false) and 1 (true).
- SQLite does not have a storage class set aside for storing dates and/or times. Instead, the built-in date and time functions of SQLite are capable of storing dates and times as TEXT, REAL, or INTEGER values.

# SQLite Database Adapter



# SQLite Database Adapter: SQLiteOpenHelper

SQLiteOpenHelper is a base class provided by Android to facilitate the management of SQLite databases in an Android application. It provides methods for creating, upgrading, and managing database versions.



# SQLite Database Adapter: SQLiteOpenHelper

Example: 1.Create a class that extends SQLiteOpenHelper:

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
    // Define constants for the database name, version, etc.

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

    @Override
    public void onCreate(SQLiteDatabase db) {
        // Create database tables using SQL queries
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // Update the database structure when the version changes
    }
}
```

# SQLite Database Adapter: SQLiteOpenHelper

2. In your code, create an instance of your SQLiteOpenHelper class:

```
MyDatabaseHelper dbHelper = new MyDatabaseHelper(context);
```

3. Obtain an instance of the database for reading/writing:

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

4. Use the methods provided by the SQLiteDatabase class to interact with the database, such as insert(), update(), delete(), query(), etc.

```
// Example of inserting data into a table
ContentValues values = new ContentValues();
values.put("column1", value1);
values.put("column2", value2);
long id = db.insert("table_name", null, values);
```

5. Close the database when you're done using it:

```
db.close();
```

# SQLite Database Adapter: SQLiteOpenHelper

2. In your code, create an instance of your SQLiteOpenHelper class:

```
MyDatabaseHelper dbHelper = new MyDatabaseHelper(context);
```

3. Obtain an instance of the database for reading/writing:

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

4. Use the methods provided by the SQLiteDatabase class to interact with the database, such as insert(), update(), delete(), query(), etc.

```
// Example of inserting data into a table
ContentValues values = new ContentValues();
values.put("column1", value1);
values.put("column2", value2);
long id = db.insert("table_name", null, values);
```

5. Close the database when you're done using it:

```
db.close();
```

# SQLite Data Manipulation: Insert

1. Use SQL statement

```
public void execSQL(String sql, Object[] bindArgs)
```

```
db.execSQL("INSERT INTO accounts(name, amount) VALUES(?, ?)",
 new Object[]{"Jack", 3000});
db.close();
```

2. Use ContentValues

```
public long insert(String table, String nullColumnHack,
ContentValues values)
```

```
ContentValues values = new ContentValues();
contentValues.put("name", "Lucy");
contentValues.put("amout", 4000); long id =
db.insert("account", null, values); db.close();
```

# SQLite Data Manipulation: Query or select

**NB**: you couldn't use execSQL() to query data. Because execSQL() method has no return value.
Alternatively, you can use rawQuery() or query() method of SQLiteDatabase class to select and query data.

**1. rawQuery()**

```
public android.database.Cursor rawQuery(String sql, String[] selectionArgs)
```

```
Cursor cursor = db.rawQuery("SELECT name, amount FROM accounts
WHERE name = ?", new String[]{"Lucy"});
```

**2. query()**

```
public Cursor query(String table, String[] columns, String
selection, String[] selectionArgs, String groupBy, String having,
String orderBy, String limit)
```

```
Cursor cursor = db.query("accounts", null, null, null, null, null,
"amount DESC", "3");
```

**Using of Cursor: traverse query results**

```
while(cursor.moveToNext()) {
String name =
cursor.getString(cursor.getColumnIndex("name"));
float amount =
cursor.getFloat(cursor.getColumnIndex("amount")
);
buffer.append(name + ", " + amount + "\n"); }
db.close();
```

# SQLite Data Manipulation: Traverse query results

| Method | Description |
| --- | --- |
| moveToFirst() | Moves the cursor to the first row of the result set. |
| moveToLast() | Moves the cursor to the last row of the result set. |
| moveToNext() | Moves the cursor to the next row in the result set. |
| moveToPrevious() | Moves the cursor to the previous row in the result set. |
| moveToPosition(int) | Moves the cursor to the specified position in the result set. |

# SQLite Data Manipulation: Traverse query results

| Method | Description |
|---|---|
| moveToFirst() | Moves the cursor to the first row of the result set. |
| moveToLast() | Moves the cursor to the last row of the result set. |
| moveToNext() | Moves the cursor to the next row in the result set. |
| moveToPrevious() | Moves the cursor to the previous row in the result set. |
| moveToPosition(int) | Moves the cursor to the specified position in the result set. |
| getCount() | Returns the number of rows in the result set. |
| getColumnName(int) | Returns the name of the column at the specified index. |
| getFloat(int) | Retrieves the value of the specified column as a float. |
| getInt(int) | Retrieves the value of the specified column as an integer. |
| getString(int) | Retrieves the value of the specified column as a string. |
| isFirst() | Checks if the cursor is pointing to the first row in the result set. |
| isLast() | Checks if the cursor is pointing to the last row in the result set. |
| isBeforeFirst() | Checks if the cursor is positioned before the first row. |
| isAfterLast() | Checks if the cursor is positioned after the last row. |

# SQLite Data Manipulation: Update & Delete

**1. update**

```
public int update(String table, ContentValues values, String whereClause, String[] whereArgs)
```

```
ContentValues values = new ContentValues(); values.put("amount", 5000);
int rows = db.update("accounts", values, "name = ?", new String[]{"Jack"});
```

**2. delete**

```
public int delete(String table, String whereClause, String[] whereArgs)
```

```
int rows = db.delete("accounts", "name = ?", new String[]{"Jack"});
```

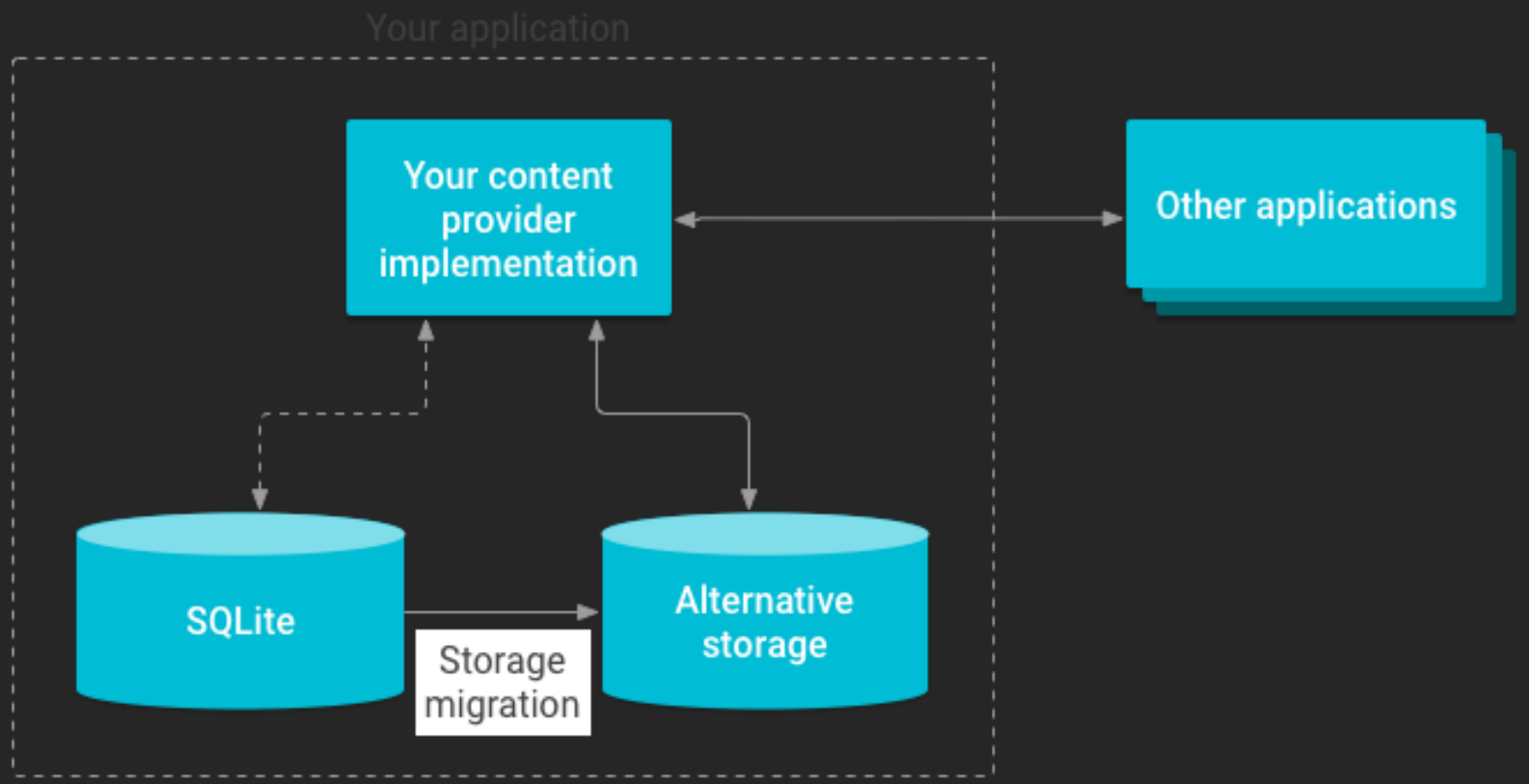
# Data Manipulation: Content Provier

Content Providers can assist an application in managing access to its stored data and provide a means to share data with other applications. They encapsulate data and provide mechanisms to define data security (access control).

Content Providers essentially serve as the standard interface that connects data within one process with code executed in another process. However, a Content Provider can be configured to securely allow other applications to access and modify the application's data.

Typically, Content Providers are used to share data from an application's SQLite database with other applications. If there is no intention to share an application's data, the Content Provider component may not be necessary. Once a Content Provider is set up, external applications must provide a URI to access the exposed data, which the Content Provider will analyze to return the appropriate data.

# Data Manipulation: Content Provier



# Data Manipulation: Create a Content Provier

1. Define the Contract: Create a Java class to define the contract for your SQLite database. This class will define the table names, column names, and other constants related to your database schema.

```
public final class MyContract {

private MyContract() {}

public static class MyTable {
public static final String TABLE_NAME = "my_table";
public static final String COLUMN_ID = "id";
public static final String COLUMN_NAME = "name"; }

 }
```

# Data Manipulation: Create a Content Provier

2. Create a Database Helper: Create a subclass of SQLiteOpenHelper to manage database creation and version management. Override the onCreate() and onUpgrade() methods as per your requirements.

```
public class MyDatabaseHelper extends SQLiteOpenHelper {
 private static final String DATABASE_NAME = "my_database.db";
private static final int DATABASE_VERSION = 1;
public MyDatabaseHelper(Context context) {
super(context, DATABASE_NAME, null, DATABASE_VERSION);
 }

@Override
public void onCreate(SQLiteDatabase db) {
String createTableQuery = "CREATE TABLE " + MyContract.MyTable.TABLE_NAME + " (" + MyContract.MyTable.COLUMN_ID + "
INTEGER PRIMARY KEY, " + MyContract.MyTable.COLUMN_NAME + " TEXT)";
db.execSQL(createTableQuery);
}
@Override
 public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
// Implement upgrade logic if needed

}
}
```

# Data Manipulation: Create a Content Provier

3. Implement Content Provider: Create a subclass of ContentProvider to define the content provider for your database. Override the necessary methods like onCreate(), query(), insert(), update(), delete(), etc.

```
public class MyContentProvider extends ContentProvider {
private MyDatabaseHelper databaseHelper;

@Override
public boolean onCreate() {
 databaseHelper = new MyDatabaseHelper(getContext());
return true;
}
```

```
@Nullable
@Override
 public Cursor query(@NonNull Uri uri, @Nullable String[]
projection, @Nullable String selection, @Nullable String[]
selectionArgs, @Nullable String sortOrder)
 {
SQLiteDatabase db = databaseHelper.getReadableDatabase();
Cursor cursor = db.query(MyContract.MyTable.TABLE_NAME,
projection, selection, selectionArgs, null, null, sortOrder);
 cursor.setNotificationUri(getContext().getContentResolver(),
uri);
 return cursor;
}
```

# Data Manipulation: Create a Content Provier

```
@Nullable
@Override
 public Uri insert(@NonNull Uri uri, @Nullable ContentValues values) {
SQLiteDatabase db = databaseHelper.getWritableDatabase();
long id = db.insert(MyContract.MyTable.TABLE_NAME, null, values);
if (id != -1) {
getContext().getContentResolver().notifyChange(uri, null);
return ContentUris.withAppendedId(uri, id);
}

return null;
}
// Implement other methods like update(), delete(), etc., based on your
requirements
@Nullable
@Override
public String getType(@NonNull Uri uri) {
return null; // Modify as per your requirements
 }}
```

# Data Manipulation: Create a Content Provier

4. Declare Content Provider in Manifest: Add the necessary entries to your app's manifest file to declare the content provider.

```
<manifest
xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <application ...>
        ...
        <provider
            android:name=".MyContentProvider"
            android:authorities="com.example.myapp.provider"
            android:exported="false" />
        ...
    </application>

</manifest>
```

# Using the sqlite3 Command-Line

- Start a virtual device (emulator) and launch the ADB shell to connect to the database from a command prompt.

```
C:>adb devices
List of devices attached
emulator-5554   device

C:>adb -s emulator-5554 shell
generic_x86:/ $ su
generic_x86:/ # sqlite3
  /data/data/edu.citadel.android.emergency/databases/emergency
  .db
SQLite version 3.9.2 2015-11-02 18:31:45
Enter ".help" for usage hints.
sqlite>
```

Note: Does not work with a real device.

response

- Enter SQL statements and commands at the prompt.
  - SQL statements must be terminated by a semicolon.
  - SQLite commands start with a period.

# Selected SQLite Commands

- .databases List names/files of attached databases
- .exit Exit this program
- .header(s) ON|OFF Turn display of headers on or off
- .help Show all SQLite commands
- .import FILE TABLE Import data from FILE into TABLE
- .mode column Left-aligned columns. (See .width)
- .read FILENAME Execute SQL in FILENAME
- .schema Show the CREATE statements
- .width NUM NUM … Set column widths for “column” mode

# Example: Using the sqlite3 Command-Line

```
sqlite> .mode column
sqlite> .width 3 15 12
sqlite> .headers on
sqlite> select * from emergency_contacts;
_id  name            phone_num
---  --------------  ------------
1    Emergency - 911  911
2    Consolidated Di  843-743-7200
3    Home            123-456-7890
4    Mary - Home      234-567-8901
5    Jim - Cell       345-678-9012
6    Billy - Cell     456-789-0123
7    Monica - Home    567-890-1234
8    Tommy - Cell     567-890-1234
sqlite>
```

# Chapiter VII:

# Services

# Introduction(1)

A service is typically designed to run in the background, without a visible user interface, and it can be implemented as a daemon or a thread.

The concept of a singleton is often used to ensure that there is only one active instance of the service at any given time. This allows for resource sharing and coordination of access to the service's functionalities.

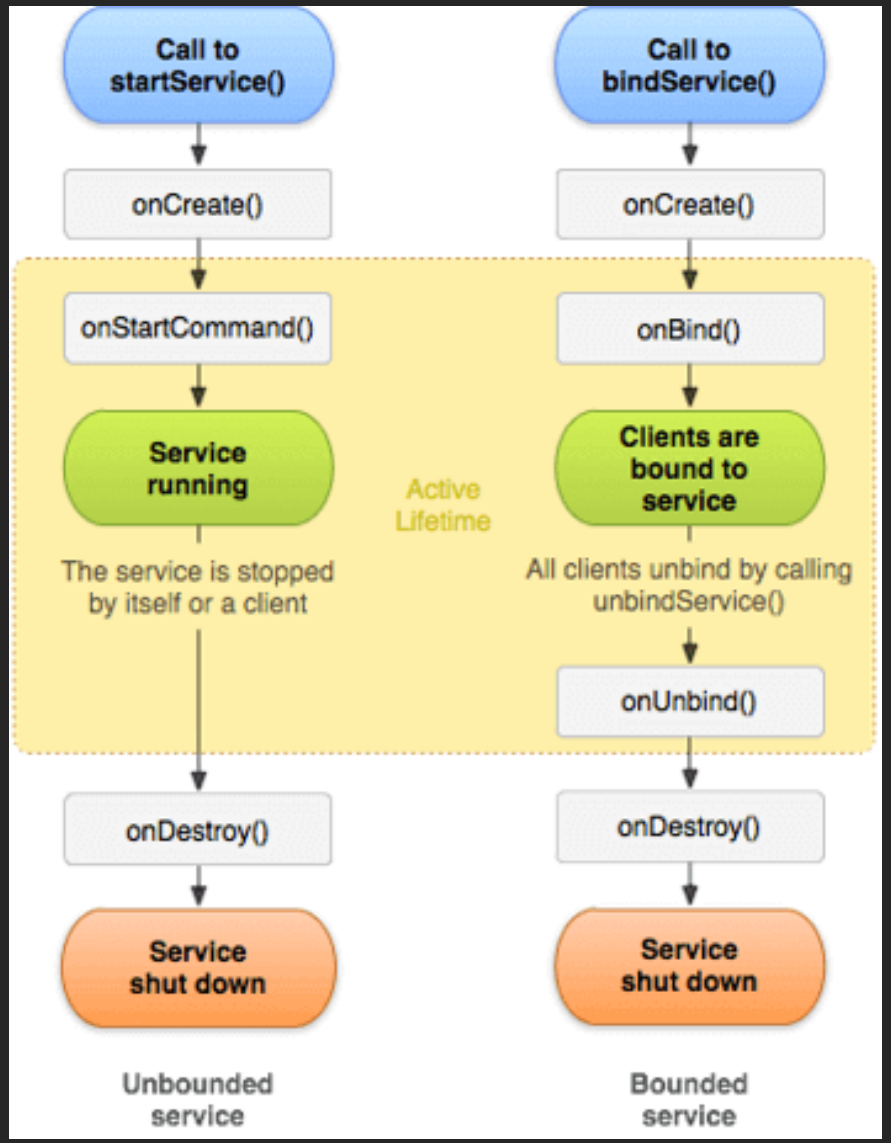
However, if the service no longer has any users, it can be destroyed to save resources. When a new client connects to the service, a new instance of the singleton can be recreated.

# Introduction(2)

In the specific case where the service launches threads, it is important to consider the behavior when destroying and recreating the singleton. If a thread is launched by the service, but the service is then destroyed and recreated, the new instance of the service will not be aware of the previous thread and will launch a new separate thread. This can lead to thread management issues, such as unfinished threads or conflicts between threads.

To avoid such problems, it is necessary to implement appropriate logic to manage the threads when destroying and recreating the service. This may involve properly stopping and terminating ongoing threads before destroying the service or implementing a mechanism to track and resume threads when recreating the service.

# Service Life Cycle



# Types of Services

These are the three different types of services:

| Foreground | Background | Bound |
| --- | --- | --- |
| Foreground services are long-running services that provide ongoing user-visible functionality. They display a persistent notification to indicate their operation and importance. Foreground services are commonly used for tasks that require continuous execution, such as music playback, navigation, or ongoing downloads. | A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service. | Bound services are used for inter-component communication, allowing other components (such as activities or other services) to bind to them and interact through the bindService() method. Bound services offer methods and interfaces that can be accessed by the binding component, enabling a client-server relationship. |

# Service VS Thread

| Service | Thread |
| --- | --- |
| An Android component that runs in the background, independent of any user interface. | A unit of execution within a process. |
| Designed for long-running tasks or providing functionality to other components. | Used for achieving concurrent and parallel execution. |
| Can be started and stopped by other components using methods like startService() and stopService(). | Created, managed, and controlled by the operating system or programming language runtime. |
| Performs operations even when the app is not actively visible or interacted with. | Used to handle background tasks, network operations, UI updates, or computationally intensive operations. |
| Can be started as a foreground service with a persistent notification. | Execution can be sequential or concurrent, depending on how threads are utilized. |
| Supports inter-component communication through binding. | Communication between threads typically involves shared data or inter-thread messaging mechanisms. |
| Primarily used for background tasks, such as data synchronization or network requests. | Used to achieve concurrency, parallelism, or responsive user interfaces. |
| Lifecycle management involves starting, stopping, and binding to services. | Threads are created, executed, and terminated based on the application's logic and requirements. |
| Managed by the Android system, which may optimize resource usage and terminate services in certain situations. | Managed by the operating system or runtime, which provides thread scheduling and resource allocation. |

# Foreground Service

A foreground service is a type of service in Android that provides ongoing user-visible functionality while running in the background. It is designed for tasks that require continuous execution and ongoing interaction with the user, even when the app is not in the foreground or actively being interacted with.

Foreground services are typically used for operations that have high importance, such as playing music, performing location tracking, or managing ongoing downloads. By running as a foreground service, the app indicates to the user that there is an ongoing operation that they should be aware of.

# Foreground Service

Key characteristics of foreground services in Android include:

| Characteristic | Description |
| --- | --- |
| Persistent Notification | Foreground services display a persistent notification in the notification bar to inform the user about the ongoing operation. |
| Increased Priority | Foreground services have a higher priority than regular background services, reducing the likelihood of termination by the system. |
| User Attention | Foreground services are meant to capture and maintain the user's attention, even when the app is not actively used. |
| Lifecycle Management | Foreground services need to be explicitly started using **startForeground()** and can be stopped with **stopForeground()** or **stopService().** |
| User Experience Considerations | Designing clear and relevant notifications and user interactions is crucial for providing a seamless and non-intrusive experience. |

# Background Service

Background services are typically used for operations that do not need immediate user attention but are necessary for the overall functionality of the app. Examples include data synchronization, network requests, periodic updates, or any task that requires continuous or periodic execution.

Unlike foreground services, background services do not display a persistent notification to the user and operate with lower priority. They are designed to optimize system resources and minimize battery consumption, while still allowing tasks to be executed in the background.

# Background Service

Key characteristics of background services in Android include:

| Characteristic | Description |
| --- | --- |
| Independent Execution | Background services run independently of the app's user interface, allowing tasks to continue even when the app is not active. |
| Long-Running Tasks | Background services are suitable for performing long-running tasks or operations without direct user interaction. |
| No User Interface | Background services typically do not have a visible user interface and work silently in the background. |
| Lifecycle Management | The Android system manages the lifecycle of background services, allowing them to be started and stopped by other components. |
| Limited System Resources | Background services should be mindful of system resources, such as CPU and battery usage, to optimize performance. |
| No User Attention | Background services do not require continuous user interaction and focus on performing tasks in the background. |
| Not Intended for Parallelism | Background services are typically designed for sequential execution and may not be suitable for parallel tasks. |
| Ongoing Operations | Background services support continuous or periodic operations that persist even when the app is not in the foreground. |

# Bound Service

A bound service in Android is a type of service that allows components, such as activities or other services, to bind to it and interact with it through an interface defined by the service. It establishes a client-server relationship between the service and the component binding to it.

# Bound Service

| Characteristic | Description |
| --- | --- |
| Binding | Components can bind to a bound service using bindService() method, establishing a client-server relationship with the service. |
| Interface | Bound services define an interface (typically implemented as an IBinder) for clients to interact with the service. |
| Lifecycle | Bound services' lifecycle is dependent on the components bound to it. The service is typically destroyed when all clients unbind. |
| Local or Remote Binding | Bound services can be bound either locally within the same process or remotely across different processes using inter-process communication (IPC). |
| Data Sharing | Bound services facilitate data sharing between the service and its clients, allowing clients to access and modify shared data. |
| Contextual Lifetime | Bound services are typically used when the service's lifespan is tied to the lifespan of its bound component(s). |

# Service Example: (1)

```
public class MyService extends Service {

    private static final int NOTIFICATION_ID = 1;
    private static final String CHANNEL_ID = "ForegroundServiceChannel";

    @Override
    public void onCreate() {
        super.onCreate();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        // Perform your long-running task or operations here...
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        // Cleanup or release any resources here...
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }

    private void createNotificationChannel() {


    }
}
```

# Service Example: (2)

Make sure to declare the service in the AndroidManifest.xml file:

```
<service
    android:name=". MyService "
    android:enabled="true"
    android:exported="false" />
```

To start the foreground service, you can use the following code in your activity:

```
Intent serviceIntent = new Intent(this, MyService.class);
ContextCompat.startForegroundService(this, serviceIntent);
```

بالتوفيق في
الامتحان