

Mobile Application Development

Mobile Platforms and Application Development fundamentals

Lecture Plan

- Introduction to App Development
- Mobile Platforms and Application Development fundamentals
- Mobile Interface Design Concepts and UI/UX Design
- Introduction to Android Operating System
- **Main Components of Android Application**
- Sensors and Media Handling in Android Applications
- Data Handling in Android Applications
- Android Application Testing and security aspects

Difference between DVM and ART

- Dalvik Virtual Machine (DVM) was the original virtual machine used by Android before Android 5.0 Lollipop. It was designed specifically for the mobile environment and optimized for low-memory devices. DVM used a just-in-time (JIT) compiler to translate bytecode to machine code at runtime, which allowed for faster performance compared to other virtual machines at the time.
- Android Runtime (ART) was introduced with Android 5.0 Lollipop as a replacement for Dalvik. ART uses Ahead-of-Time (AOT) compilation, which means that code is compiled to native machine code during installation rather than at runtime. This approach provides faster startup times and improved overall performance, but it requires more storage space to store the compiled code.

Key differences between Dalvik and ART

- **Compilation method:** Dalvik uses JIT compilation, while ART uses AOT compilation.
- **Performance:** ART is generally faster than Dalvik because it compiles code ahead of time, leading to faster startup times and improved overall performance.
- **Storage requirements:** ART requires more storage space than Dalvik because it compiles code to native machine code during installation, while Dalvik compiles code at runtime.
- **Compatibility:** While both virtual machines can run the same Android apps, some apps may not work properly on ART if they are not compatible with the AOT compilation process.
- **Read more:** <https://developer.android.com/guide/platform>



Android Core Building Blocks

- **Activities:** Activities represent a single screen with a user interface. They are responsible for interacting with the user and managing the app's UI.
- **Services:** Services are components that run in the background and perform long-running operations. They are used to perform tasks that do not require user interaction, such as downloading files or playing music.
- **Broadcast Receivers:** Broadcast Receivers are components that respond to system-wide broadcast messages. They allow an app to receive and respond to events, such as a low battery warning or a network connection change.
- **Content Providers:** Content Providers are components that manage a shared set of app data that can be accessed by other apps. They allow an app to share data with other apps, such as contacts, images, or videos.

Activity Manager

- This class gives information about, and interacts with, activities, services, and the containing process.
- Several of the methods in this class are for informational or debugging reasons only, and your app's runtime behavior should not be altered by using them.
- Most application developers should not have the need to use this class, most of whose methods are for specialized use cases.
- However, a few methods are more broadly applicable.
 - For instance, `isLowRamDevice()` enables your app to detect whether it is running on a low-memory device and behave accordingly.
 - `clearApplicationUserData()` is for apps with reset-data functionality.

Activity

Activity is a core component of the Android operating system that provides a single, focused thing that a user can do.

Each activity represents a screen in the app's user interface and is responsible for managing user interactions, displaying UI elements, and responding to events such as button clicks or screen rotations.

Activities are created and managed by the Android framework and can be launched by other activities, system components, or apps.

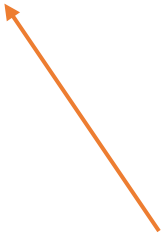
To create an activity, you need to define a subclass of the Activity class and implement its lifecycle methods, such as `onCreate()`, `onPause()`, and `onDestroy()`.

The lifecycle of an activity is an important concept to understand, as it determines how the activity behaves in response to various system events, such as when the user switches to another app or rotates the screen.

Activities can also be configured to work with other Android components, such as fragments, services, and broadcast receivers, to create more complex and flexible apps.

Activity Initial code

```
class MainActivity : AppCompatActivity() {  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
    }  
}
```

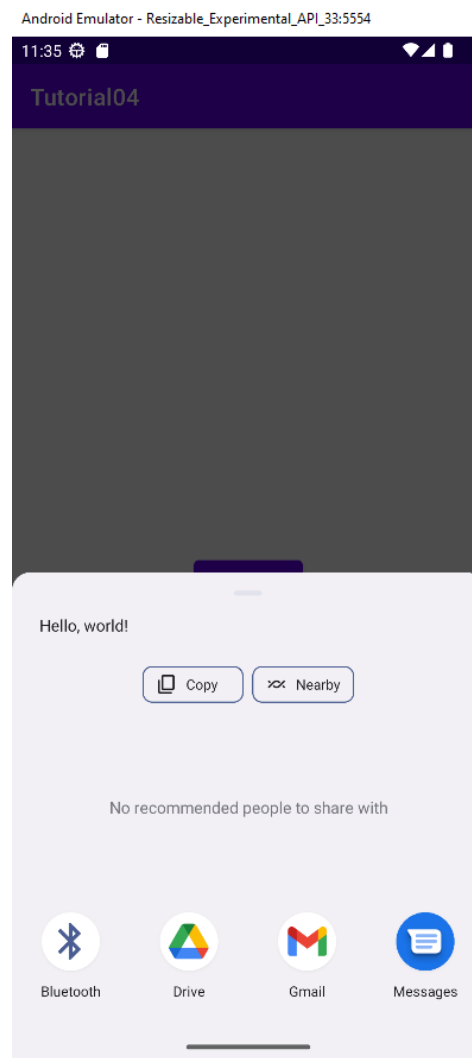


UI will be loaded to the screen from
this method

Intents

- An Intent is an object that can be used to request an action from another app component.
- It is a messaging system used to communicate between Android components such as activities, services, broadcast receivers, and content providers.
- The Intent provides a way to start an activity, send a broadcast, or deliver a message to another app component.
- It can also carry data between components using extras, which are key-value pairs.
- Intents can be explicit or implicit. An explicit intent specifies the component to be invoked by name, while an implicit intent describes the type of action to be performed and leaves the system to find an appropriate component to handle it.
- Android Intent is a powerful and flexible mechanism that enables communication between different app components and facilitates app integration.

Implicit intent



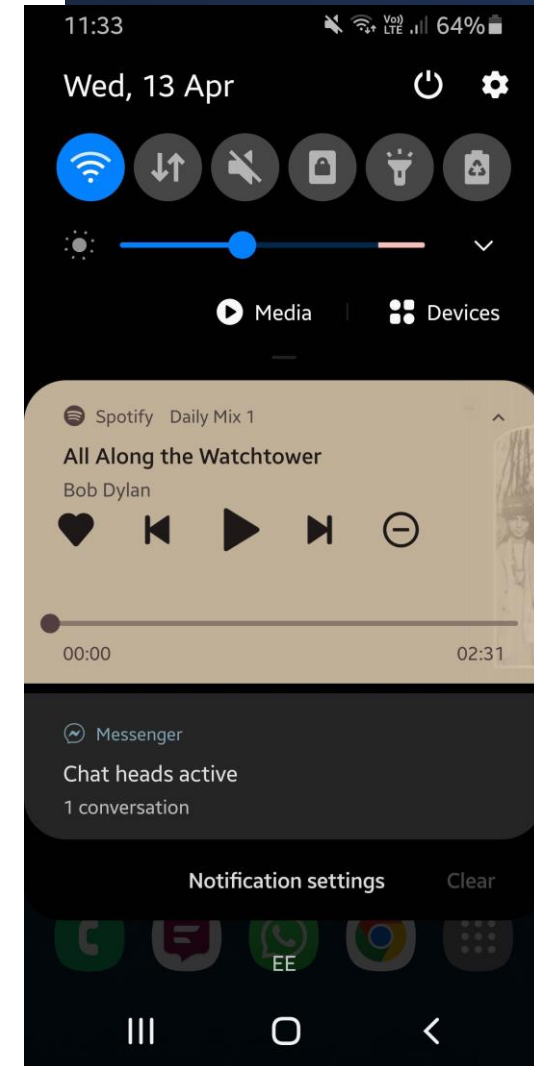
```
button.setOnClickListener { it: View!
    val intent = Intent(Intent.ACTION_SEND)
    intent.type = "text/plain"
    intent.putExtra(Intent.EXTRA_TEXT, value: "Hello, world!")
    startActivity(Intent.createChooser(intent, title: "Share via"))
}
```

Explicit intent

```
btnBack.setOnClickListener {  
    val intent = Intent(this, MainActivity::class.java)  
    startActivity(intent)  
    finish()  
}
```

Android Services

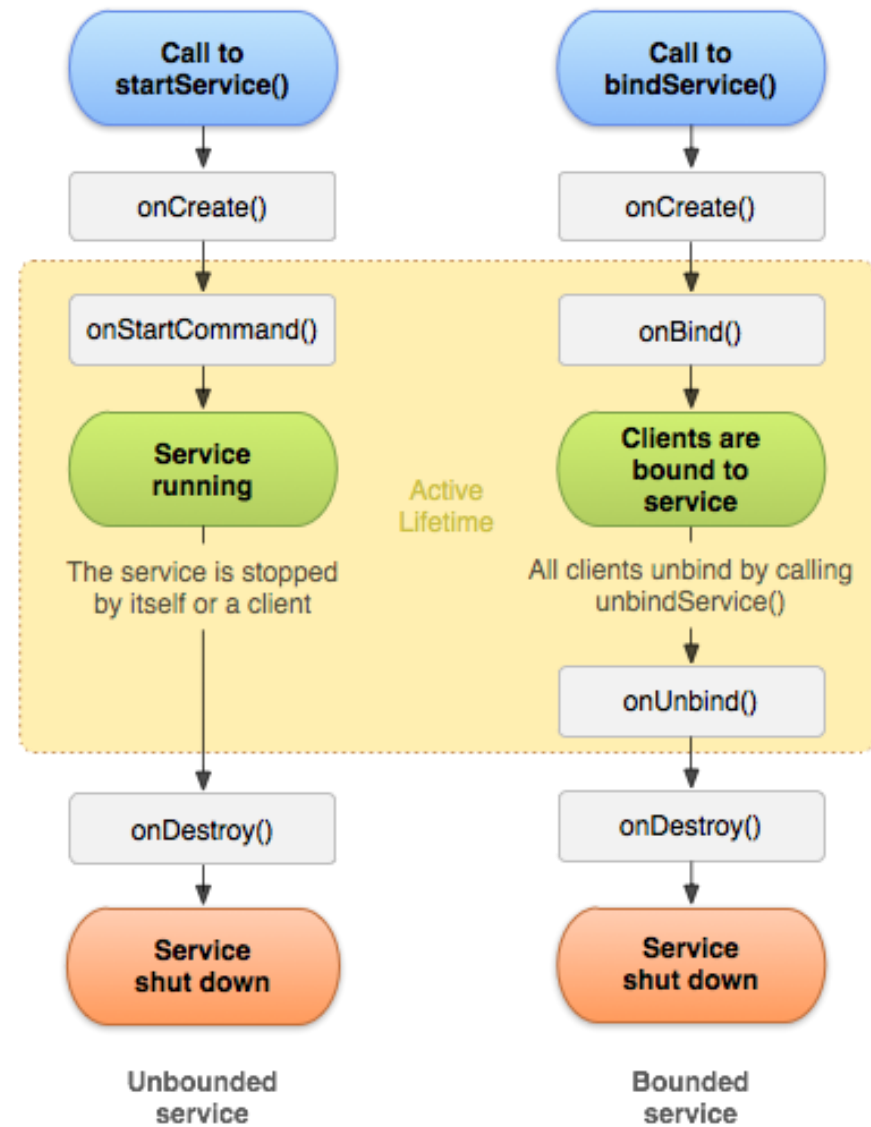
- Android Services are a fundamental component of the Android operating system.
- They enable developers to create long-running background tasks that can perform operations even when an app is not in the foreground.
- This enables developers to build applications that can perform complex tasks in the background without interrupting the user experience.



Types of Android Services

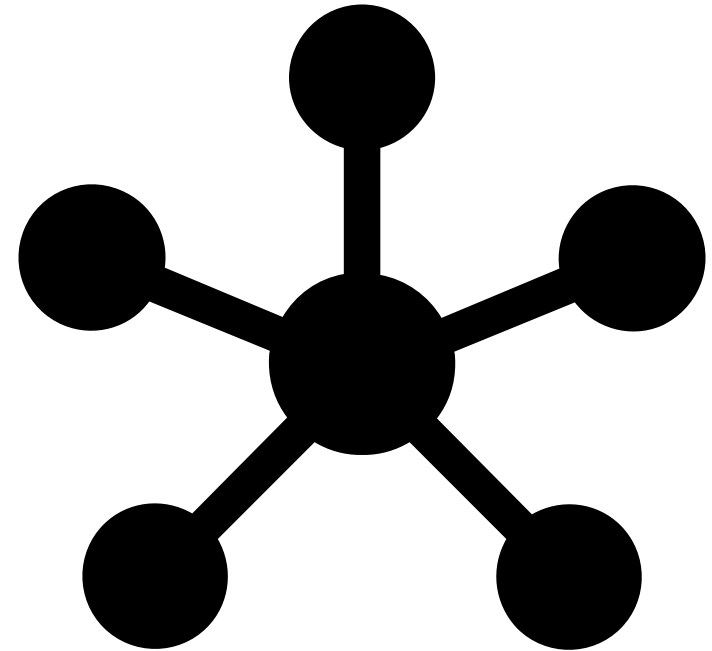
- There are two types of Android Services: Started Services and Bound Services.
- Started Services: These services are started by calling `startService()` method and can run indefinitely in the background, even after the app is closed.
- Bound Services: These services are bound to a client component (such as an Activity or a Fragment) by calling `bindService()` method, and they provide a client-server interface for communication between components.

Service Lifecycle



Service Communication

- Services can communicate with other components of an app using IPC mechanisms such as intents.
- A service can send an intent to a broadcast receiver to notify other components of an app of a change in state.
- A service can also communicate with a client component (such as an Activity or a Fragment) using a Messenger or a Binder.



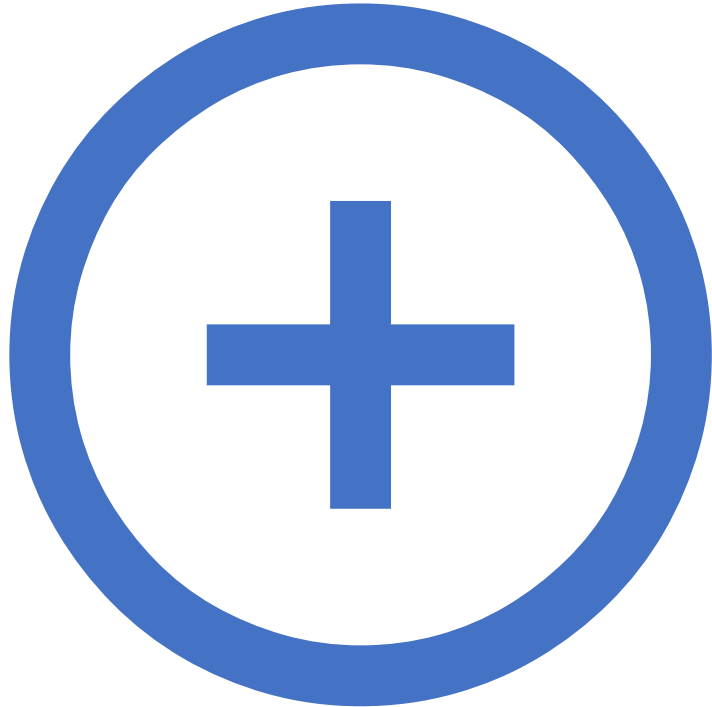
Best practices on using services

- It's critical to use services wisely to guarantee top performance and efficiency.
- Services should be designed to use as few resources as possible and to release those resources when they are no longer needed.
- It is also important to ensure that services do not drain the device's battery by running indefinitely in the background.



Broadcast Receivers

- Android Broadcast Receivers are a fundamental component of the Android operating system that enable apps to receive and respond to system-wide or app-specific broadcast messages.
- Broadcast messages are sent by the system or other apps, and they can be used to trigger specific actions or events within an app.



Types of Broadcast Receivers

- There are two types of Broadcast Receivers: Static and Dynamic.
- Static Broadcast Receivers are declared in the app's manifest file and are used to listen for system-wide broadcast messages.
- Dynamic Broadcast Receivers are registered programmatically in an app's code and are used to listen for app-specific broadcast messages.

Broadcast Receiver Lifecycle

- Broadcast Receivers have a lifecycle that consists of several callback methods, such as `onReceive()`.
- The `onReceive()` method is called when a broadcast message is received by the receiver.
- The receiver can then process the message and perform any necessary actions or trigger any necessary events.

Registering Broadcast Receivers



Broadcast Receivers can be registered statically in the app's manifest file, or dynamically in the app's code using the `registerReceiver()` method.



When registering a Broadcast Receiver, developers must specify the broadcast message they wish to listen for using an `IntentFilter`.

Best Practices on using Broadcast Receivers

- To ensure optimal performance and efficiency, it is important to use Broadcast Receivers judiciously.
- Broadcast Receivers should be designed to use as few resources as possible and to release those resources when they are no longer needed.
- It is also important to ensure that Broadcast Receivers do not drain the device's battery by running indefinitely in the background.

Examples for Broadcast Receivers

Battery Level Receiver

- This receiver can be used to detect when the device's battery level changes.
- When the battery level changes, the receiver can perform specific actions, such as displaying a notification or changing the behavior of the app.

Network Connectivity Receiver

- This receiver can be used to detect changes in network connectivity.
- When the device connects to a network or loses network connectivity, the receiver can perform specific actions, such as fetching data from a server or displaying an error message to the user.

SMS Receiver

- This receiver can be used to detect when the device receives an SMS message.
- When an SMS message is received, the receiver can perform specific actions, such as displaying a notification or triggering a specific event within the app.

Screen State Receiver

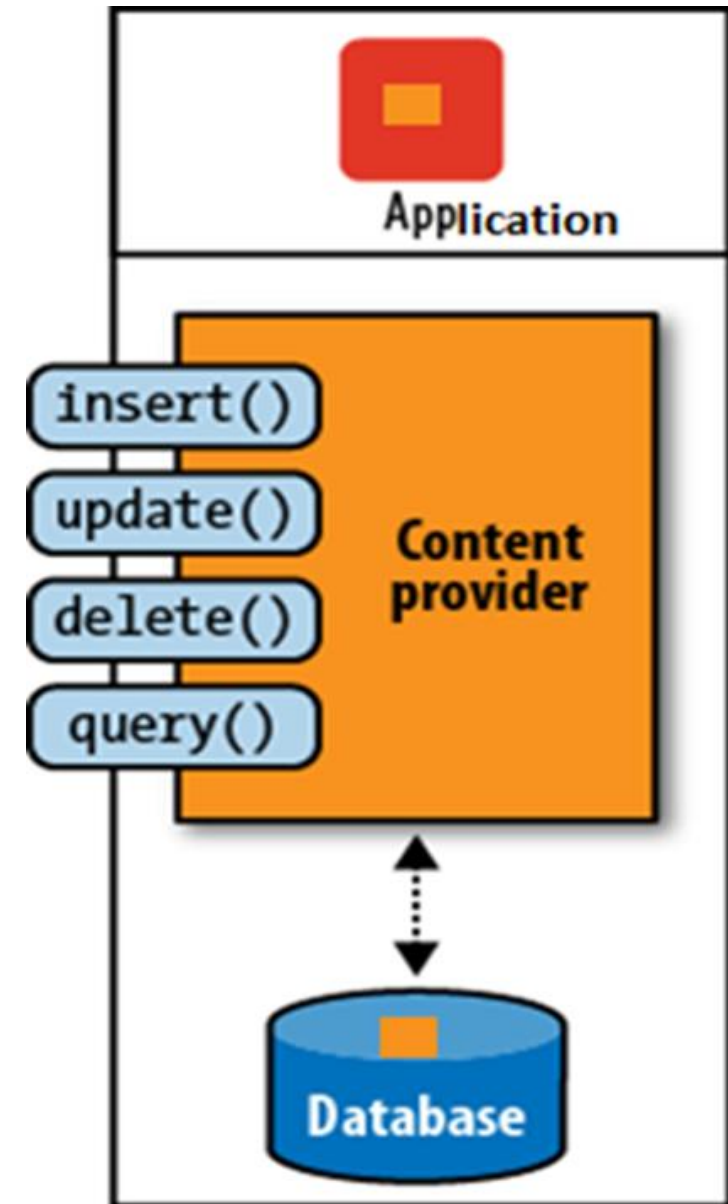
- This receiver can be used to detect changes in the device's screen state, such as when the screen is turned on or off.
- When the screen state changes, the receiver can perform specific actions, such as starting or stopping a service or changing the behavior of the app.

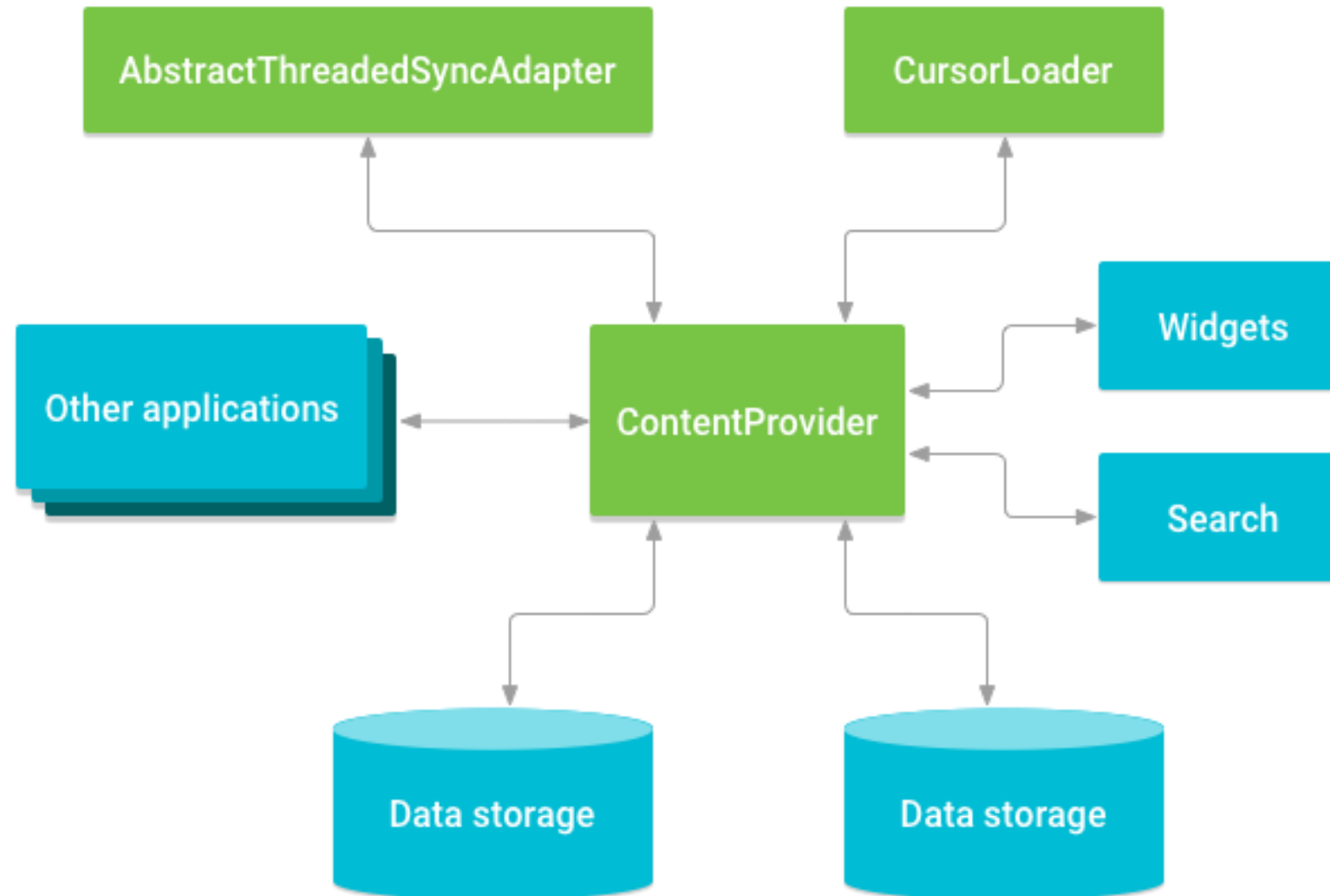
Power State Receiver

- This receiver can be used to detect changes in the device's power state, such as when the device is plugged in or unplugged.
- When the power state changes, the receiver can perform specific actions, such as starting or stopping a service or changing the behavior of the app.

Content Providers

- Android Content Providers are a fundamental component of the Android operating system that allow apps to securely share data with other apps.
- Content Providers enable apps to expose a structured set of data to other apps and provide a mechanism for data storage and retrieval.
- There are two types of Content Providers: built-in and custom.
- Built-in Content Providers are part of the Android operating system and provide access to system-wide data, such as contacts, media, and settings.
- Custom Content Providers are created by app developers and provide access to app-specific data, such as user preferences and app settings.





Content Provider Components

Content Providers consist of several key components, including a content provider class, a data contract, and a set of URIs.

The content provider class implements the methods for accessing and modifying data, while the data contract defines the structure of the data and its relationships.

The set of URIs specifies the location of the data and how it can be accessed.

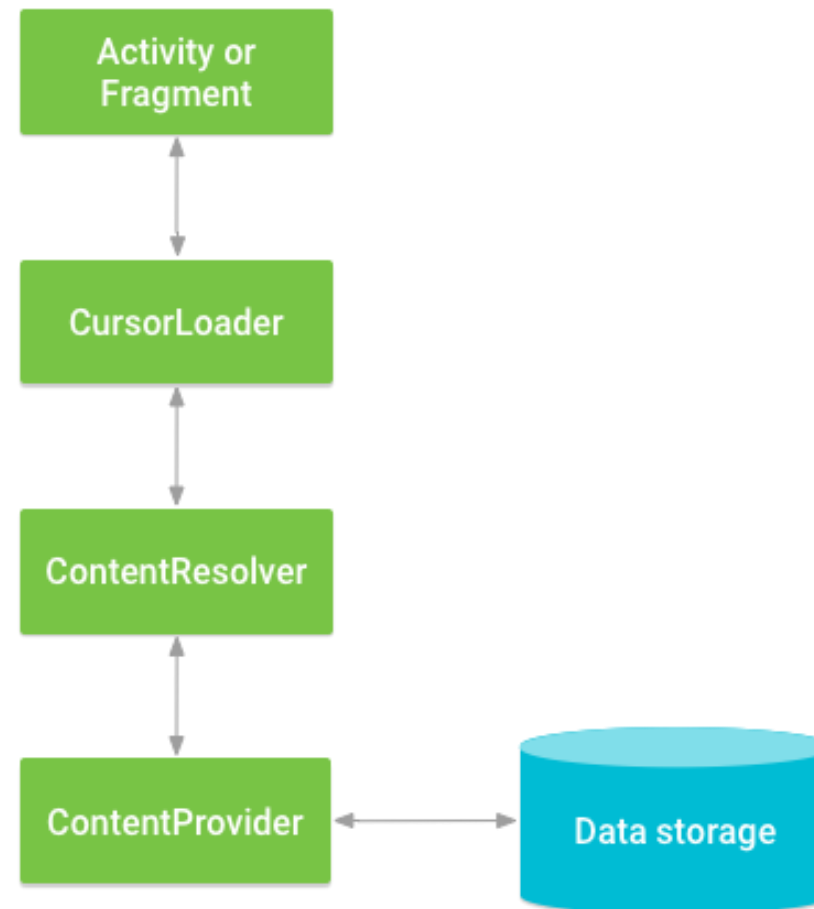


Content Provider Permissions

- Content Providers require permissions to access and modify data, and these permissions must be declared in the app's manifest file.
- Permissions can be set at the level of individual data items or at the level of the entire content provider.
- App developers must ensure that the permissions they grant to Content Providers are appropriate and do not compromise user privacy or security.

Using Content Providers

- Other apps can access Content Providers by sending requests using a ContentResolver object.
- The ContentResolver object sends requests to the Content Provider's methods, which perform the requested operations on the data.
- Developers can use Content Providers to enable sharing of data between different components of their own app or between different apps.



Examples of Content Providers

Contacts
Provider

Media Store
Provider

Settings
Provider

Calendar
Provider

Call Log
Provider

User
Dictionary
Provider

Thank you!

- References

- <https://developer.android.com/guide/components/activities/intro-activities>
- <https://developer.android.com/guide/components/services>
- <https://developer.android.com/reference/android/content/BroadcastReceiver>
- <https://developer.android.com/guide/topics/providers/content-provider-basics>

