Logcat and Android Activity Lifecycle

Logcat

Logcat is a powerful tool used to view system messages, logs from your app (like `Log.d()`, `Log.i()`, etc.), stack traces when your app crashes, and messages from other apps or the system itself.

How to Open Logcat in Android Studio

1. Run your app on a device or emulator.
2. Go to the **bottom bar** in Android Studio and click on **Logcat**.
   - If you don't see it, try: `View > Tool Windows > Logcat`.

Key Features of Logcat

Feature	Description
Search bar	Filter logs by keyword (e.g., your app tag, method name, etc.).
Device and App selection	Choose the device/emulator and the running app whose logs you want to see.
Log level filter	Choose from: Verbose, Debug, Info, Warn, Error, Assert. Useful for filtering noise.
Logcat Tags	You can tag your log messages (e.g., `Log.d("MyActivity", "Debug message")`) and filter by tag.
Pause and Clear buttons	Useful when logs are overwhelming. Pause to stop auto-scrolling; clear to refresh.

Logging Levels and Methods

```
Log.v("TAG", "Verbose log");
Log.d("TAG", "Debug log");
Log.i("TAG", "Info log");
Log.w("TAG", "Warning log");
Log.e("TAG", "Error log");
```

Use appropriate log levels depending on the severity and intent of the message.

Example

Create a New Android Project

In Android Studio:

1. **New Project** → Empty Activity
2. Language: **Kotlin**
3. Click **Finish**

Android Studio generates a simple `MainActivity.kt` that looks like:

```
package com.example.myapp

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle

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

Add Meaningful Logs in `MainActivity.kt`

```
package com.example.myapp

import android.os.Bundle
import android.util.Log
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    companion object {
        private const val TAG = "MainActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        if (BuildConfig.DEBUG) {
            Log.d(TAG, "onCreate: MainActivity launched")
```

```
        }

        simulateNetworkCall()
    }

    private fun simulateNetworkCall() {
        if (BuildConfig.DEBUG) {
            Log.d(TAG, "simulateNetworkCall: Simulating network request...")
        }

        try {
            val result = 10 / 0  // Will throw an exception (divide by zero)
        } catch (e: Exception) {
            if (BuildConfig.DEBUG) {
                Log.e(TAG, "simulateNetworkCall: Error occurred", e)
            }
        }
    }
}
```

Now, when you **run the app**, you'll see clean, filtered logs in Logcat.

Use Logcat Filters

1. Run the app on your emulator or device.
2. Open the **Logcat** window in Android Studio (bottom panel).
3. At the top:
   - Select your **device/emulator**.
   - Select your **app process** (`com.example.myapp`).
4. In the search bar, type `MainActivity` to filter by the tag.
5. Or create a custom filter:
   - Click the dropdown (≡ or ⚙️) in the top left of Logcat.
   - Choose **Edit Filter Configuration**.
   - Set:
     - Filter Name: `MyAppLogs`
     - Log Tag: `MainActivity`
     - Package Name: `com.example.myapp`
   - Click OK

Now Logcat only shows logs that matter to your app.

Create a `Logger.kt` Helper (Optional but Recommended)

This keeps logging clean and centralized:

```
// Logger.kt
package com.example.myapp

import android.util.Log

object Logger {
    fun d(tag: String, message: String) {
        if (BuildConfig.DEBUG) {
            Log.d(tag, message)
        }
    }

    fun e(tag: String, message: String, throwable: Throwable? = null) {
        if (BuildConfig.DEBUG) {
            Log.e(tag, message, throwable)
        }
    }
}
```

Use it in `MainActivity.kt` like this:

```
Logger.d(TAG, "Logger: Activity started")
Logger.e(TAG, "Logger: Something went wrong", Exception("Fake crash"))
```

Final Touch: Logging with Kotlin Extensions (Clean & Safe)

`LogExtensions.kt`

```
package com.example.myapp

import android.util.Log

fun Any.logD(message: String) {
    if (BuildConfig.DEBUG) {
        Log.d(this::class.java.simpleName, message)
    }
}

fun Any.logE(message: String, throwable: Throwable? = null) {
    if (BuildConfig.DEBUG) {
        Log.e(this::class.java.simpleName, message, throwable)
```

```
    }
}
```

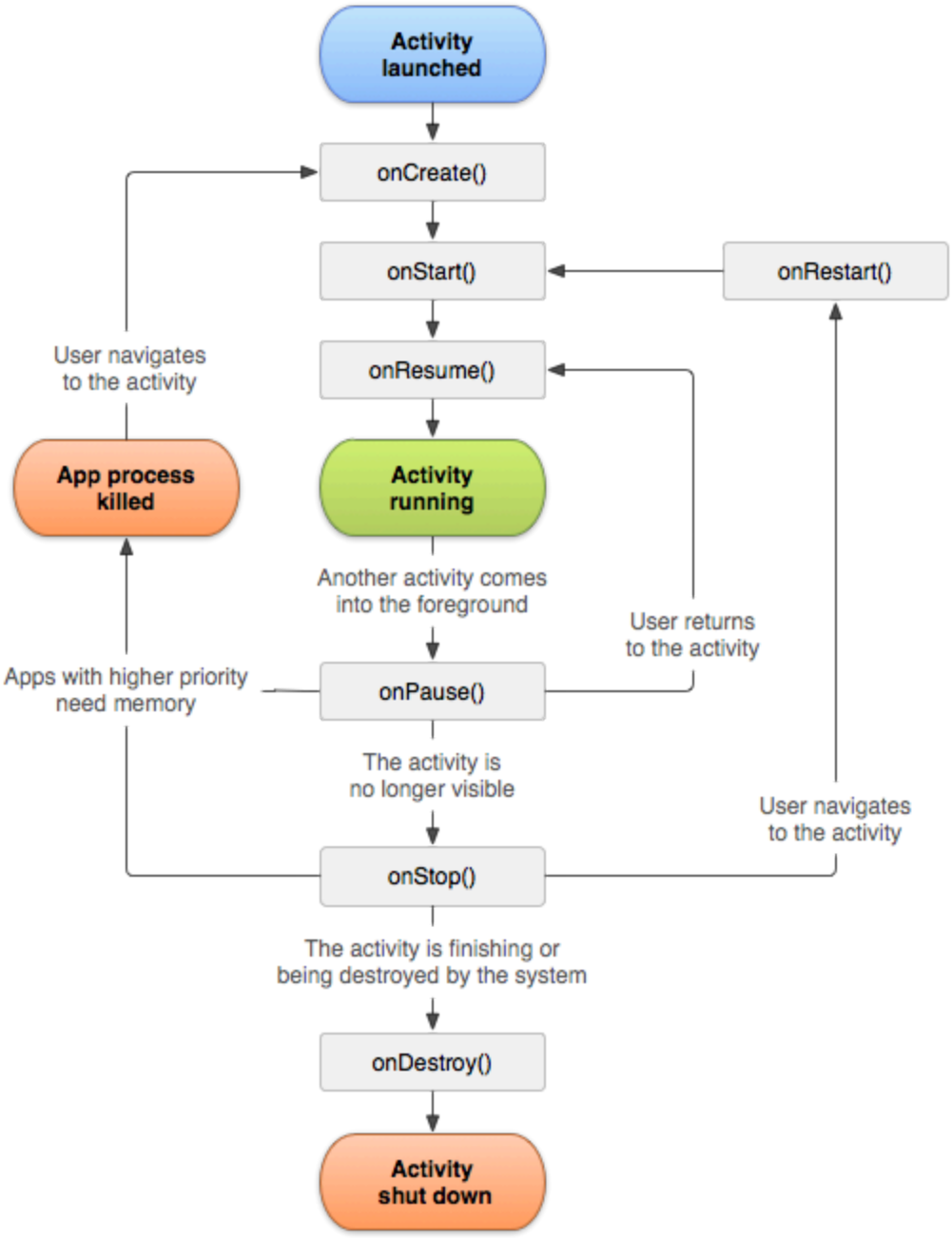
Then in `MainActivity.kt`:

```
logD("Logging something cleanly", TAG)
logE("Oops, something broke", TAG, Exception("Sample"))
```

This version avoids reflection issues, sticks to best practices, and gives you clean, reusable logging tools.

Android Activity Lifecycle

An **Activity** in Android represents a single screen with a user interface. The **activity lifecycle** is a set of methods Android calls when the activity enters or exits different states (like being visible, paused, or destroyed).



Lifecycle Method	Trigger	Typical Use
`onCreate()`	When the activity is first created (initial setup)	Initialize UI, bind views, start logic
`onStart()`	Activity is becoming visible	Prepare UI updates, resume passive actions
`onResume()`	Activity is now on screen and interactive	Start animations, camera, sensors, etc.
`onPause()`	Another activity comes in front (partially hidden)	Pause updates, animations, or save data

Lifecycle Method	Trigger	Typical Use
`onStop()`	Activity is no longer visible	Stop services, unregister receivers
`onRestart()`	Called after the activity was stopped and is about to be restarted	Re-initialize anything that was released
`onDestroy()`	Activity is finishing or being destroyed by the system	Cleanup to prevent memory leaks

Logging Setup

Is it good practice to use Logcat for activity lifecycle?

Yes, but only during development and debugging.

- **Why it's good**:
  - Helps you visualize the activity flow as users interact with your app.
  - Great for debugging unexpected behavior, especially when activities pause/stop incorrectly.
  - Can reveal memory leaks or crashes tied to lifecycle misuse.
- **What not to do**:
  - Don't leave lifecycle logs like `Log.d(TAG, "onPause called")` in production unless they're guarded with `BuildConfig.DEBUG`.
  - Avoid logging sensitive data.

`LogExtensions.kt`

This helper adds clean logging for any class:

```
package com.example.myapp

import android.util.Log

fun Any.logD(message: String) {
    if (BuildConfig.DEBUG) {
        Log.d(this::class.java.simpleName, message)
    }
}

fun Any.logE(message: String, throwable: Throwable? = null) {
    if (BuildConfig.DEBUG) {
```

```
        Log.e(this::class.java.simpleName, message, throwable)
    }
}
```

Activity with Lifecycle Logs Using Kotlin Extensions

MainActivity.kt

```
package com.example.myapp

import android.content.Intent
import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material3.Button
import androidx.compose.material3.Text
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable

class MainActivity : ComponentActivity() {

    companion object {
        private val TAG = this::class.java.simpleName
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        logD("onCreate called", TAG)

        setContent {
            MaterialTheme {
                MainScreen {
                    startActivity(Intent(this, SecondActivity::class.java))
                }
            }
        }
    }

    override fun onStart() {
        super.onStart()
        logD("onStart called", TAG)
    }

    override fun onResume() {
        super.onResume()
```

```
        logD("onResume called", TAG)
    }

    override fun onPause() {
        super.onPause()
        logD("onPause called", TAG)
    }

    override fun onStop() {
        super.onStop()
        logD("onStop called", TAG)
    }

    override fun onDestroy() {
        super.onDestroy()
        logD("onDestroy called", TAG)
    }

    override fun onRestart() {
        super.onRestart()
        logD("onRestart called", TAG)
    }
}

@Composable
fun MainScreen(onNavigate: () -> Unit) {
    Button(onClick = onNavigate) {
        Text("Go to Second Screen")
    }
}
```

SecondActivity.kt

```
package com.example.lifecycledemo

import android.os.Bundle
import androidx.activity.ComponentActivity
import androidx.activity.compose.setContent
import androidx.compose.material3.Text
import androidx.compose.material3.MaterialTheme
import androidx.compose.runtime.Composable

class SecondActivity : ComponentActivity() {

    companion object {
```

```
        private const val TAG = "SecondActivity"
    }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        logD("onCreate called", TAG)

        setContent {
            MaterialTheme {
                SecondScreen()
            }
        }
    }

    override fun onStart() {
        super.onStart()
        logD("onStart called", TAG)
    }

    override fun onResume() {
        super.onResume()
        logD("onResume called", TAG)
    }

    override fun onPause() {
        super.onPause()
        logD("onPause called", TAG)
    }

    override fun onStop() {
        super.onStop()
        logD("onStop called", TAG)
    }

    override fun onDestroy() {
        super.onDestroy()
        logD("onDestroy called", TAG)
    }

    override fun onRestart() {
        super.onRestart()
        logD("onRestart called", TAG)
    }
}

@Composable
```

```
fun SecondScreen() {
    Text("Welcome to Second Screen")
}
```

When you run and interact with your app (open, minimize, rotate, back press), Logcat will show messages.