Practical 9: To perform debugging and testing of android app using tools like Logcat, Android debug bridge, DDMS.

Debugging and Testing:

Debugging is the process of finding and fixing errors (bugs) or unexpected behaviour in our code. All code has bugs, from incorrect behaviour in our app, to behaviour that excessively consumes memory or network resources, to actual app freezing or crashing.

Bugs can result for many reasons:

- Errors in design or implementation
- Android framework limitations (or bugs)
- Missing requirements or assumptions for how the app should work
- Device limitations (or bugs)

Use the debugging, testing, and profiling capabilities in Android Studio to help us reproduce, find, and resolve all of these problems. Those capabilities include:

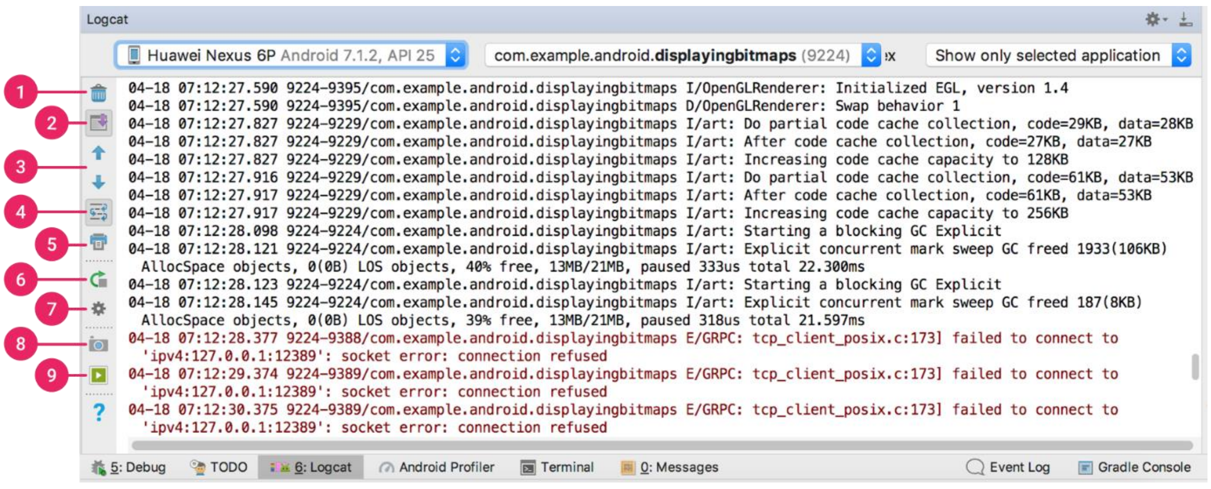
- Logcat
- Android Debug Bridge
- DDMS (Dalvik Debug Monitor Server)

LOGCAT:

We can use **log** class to send messages to the Android system log, and view those messages in Android Studio in the **Logcat** pane.

the Android SDK includes a useful logging utility class called android.util.Log. The class allows us to log messages categorized based severity; each type of logging message has its own message. Here is a listing of the message types, and their respective method calls, ordered from lowest to highest priority:

- The **Log.v()** method is used to log verbose messages.
- The **Log.d()** method is used to log debug messages.
- The **Log.i()** method is used to log informational messages.
- The **Log.w()** method is used to log warnings.
- The **Log.e()** method is used to log errors.
- The **Log.wtf()** method is used to log events that should never happen (“wtf” being an abbreviation for “What a Terrible Failure”, of course). We can think of this method as the equivalent of Java’s assert method.


Logcat
Huawei Nexus 6P Android 7.1.2, API 25
com.example.android.displayingbitmaps (9224)
Show only selected application
1
2
3
4
5
6
7
8
9
5: Debug
TODO
6: Logcat
Android Profiler
Terminal
0: Messages
Event Log
Gradle Console

```
04-18 07:12:27.590 9224-9395/com.example.android.displayingbitmaps I/OpenGLRenderer: Initialized EGL, version 1.4
04-18 07:12:27.590 9224-9395/com.example.android.displayingbitmaps D/OpenGLRenderer: Swap behavior 1
04-18 07:12:27.827 9224-9229/com.example.android.displayingbitmaps I/art: Do partial code cache collection, code=29KB, data=28KB
04-18 07:12:27.827 9224-9229/com.example.android.displayingbitmaps I/art: After code cache collection, code=27KB, data=27KB
04-18 07:12:27.827 9224-9229/com.example.android.displayingbitmaps I/art: Increasing code cache capacity to 128KB
04-18 07:12:27.916 9224-9229/com.example.android.displayingbitmaps I/art: Do partial code cache collection, code=61KB, data=53KB
04-18 07:12:27.917 9224-9229/com.example.android.displayingbitmaps I/art: After code cache collection, code=61KB, data=53KB
04-18 07:12:27.917 9224-9229/com.example.android.displayingbitmaps I/art: Increasing code cache capacity to 256KB
04-18 07:12:28.098 9224-9224/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
04-18 07:12:28.121 9224-9224/com.example.android.displayingbitmaps I/art: Explicit concurrent mark sweep GC freed 1933(106KB)
 AllocSpace objects, 0(0B) LOS objects, 40% free, 13MB/21MB, paused 333us total 22.300ms
04-18 07:12:28.123 9224-9224/com.example.android.displayingbitmaps I/art: Starting a blocking GC Explicit
04-18 07:12:28.145 9224-9224/com.example.android.displayingbitmaps I/art: Explicit concurrent mark sweep GC freed 187(8KB)
 AllocSpace objects, 0(0B) LOS objects, 39% free, 13MB/21MB, paused 318us total 21.597ms
04-18 07:12:28.377 9224-9388/com.example.android.displayingbitmaps E/GRPC: tcp_client_posix.c:173] failed to connect to
 'ipv4:127.0.0.1:12389': socket error: connection refused
04-18 07:12:29.374 9224-9389/com.example.android.displayingbitmaps E/GRPC: tcp_client_posix.c:173] failed to connect to
 'ipv4:127.0.0.1:12389': socket error: connection refused
04-18 07:12:30.375 9224-9389/com.example.android.displayingbitmaps E/GRPC: tcp_client_posix.c:173] failed to connect to
 'ipv4:127.0.0.1:12389': socket error: connection refused
```

Android Debug Bridge:

Android Debug Bridge (adb) is a versatile command-line tool that lets us communicate with a device. ADB is a **part of Android SDK** The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that we can use to run a variety of commands on a device. It is a client-server program that includes three components:

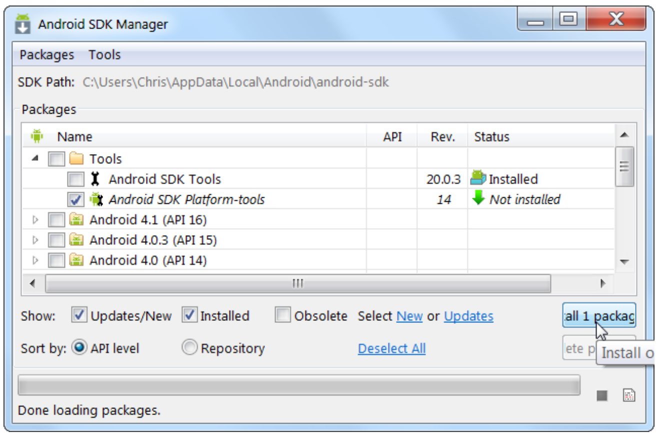
- **A client**, which sends commands. The client runs on development machine. We can invoke a client from a command-line terminal by issuing an adb command.
- **A daemon (adbd)**, which runs commands on a device. The daemon runs as a background process on each device.
- **A server**, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

Enable adb debugging:

We have to use USB debugging under Developer Options for using adb with a device.

We can now connect our device with USB. We can verify that our device is connected by executing adb devices from the ***android_sdk***/platform-tools/ directory.

Setup Android SDK



SDK Path: C:\Users\Chris\AppData\Local\Android\android-sdk

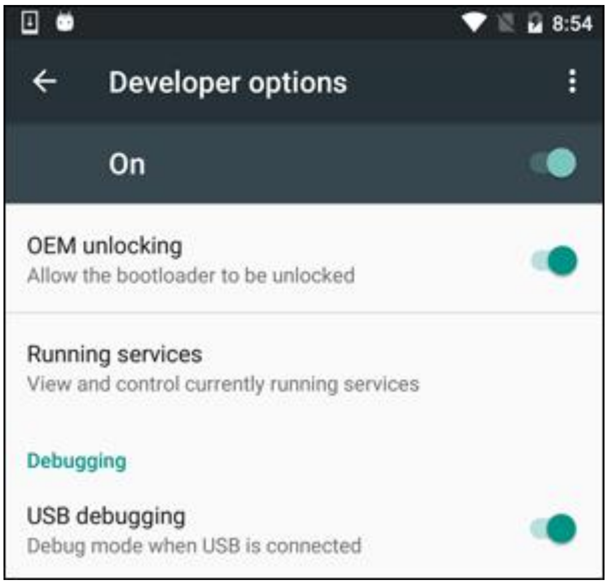
Name	API	Rev.	Status
Tools			
Android SDK Tools		20.0.3	Installed
✓ Android SDK Platform-tools		14	Not installed
Android 4.1 (API 16)			
Android 4.0.3 (API 15)			
Android 4.0 (API 14)			

Show: [✓ Updates/New] [✓ Installed] Obsolete

Sort by: [✓ API level] Repository

Done loading packages.

Enable USB debugging:



Developer options

On

OEM unlocking
Allow the bootloader to be unlocked

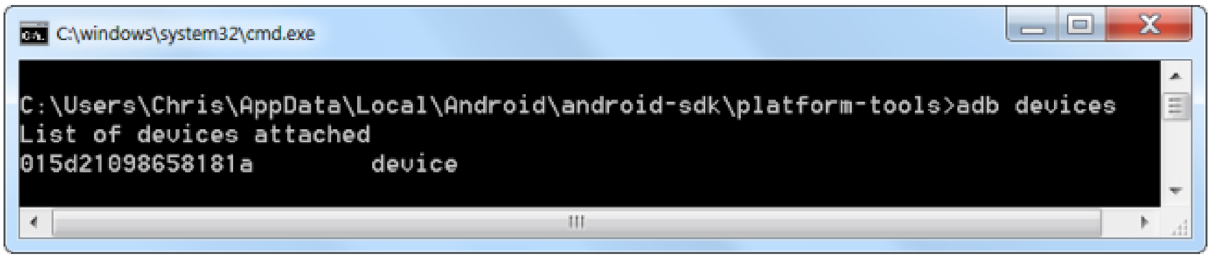
Running services
View and control currently running services

Debugging

USB debugging
Debug mode when USB is connected

Test ADB connection:

```
adb devices
```



```
C:\Users\Chris\AppData\Local\Android\android-sdk\platform-tools>adb devices
List of devices attached
015d21098658181a        device
```

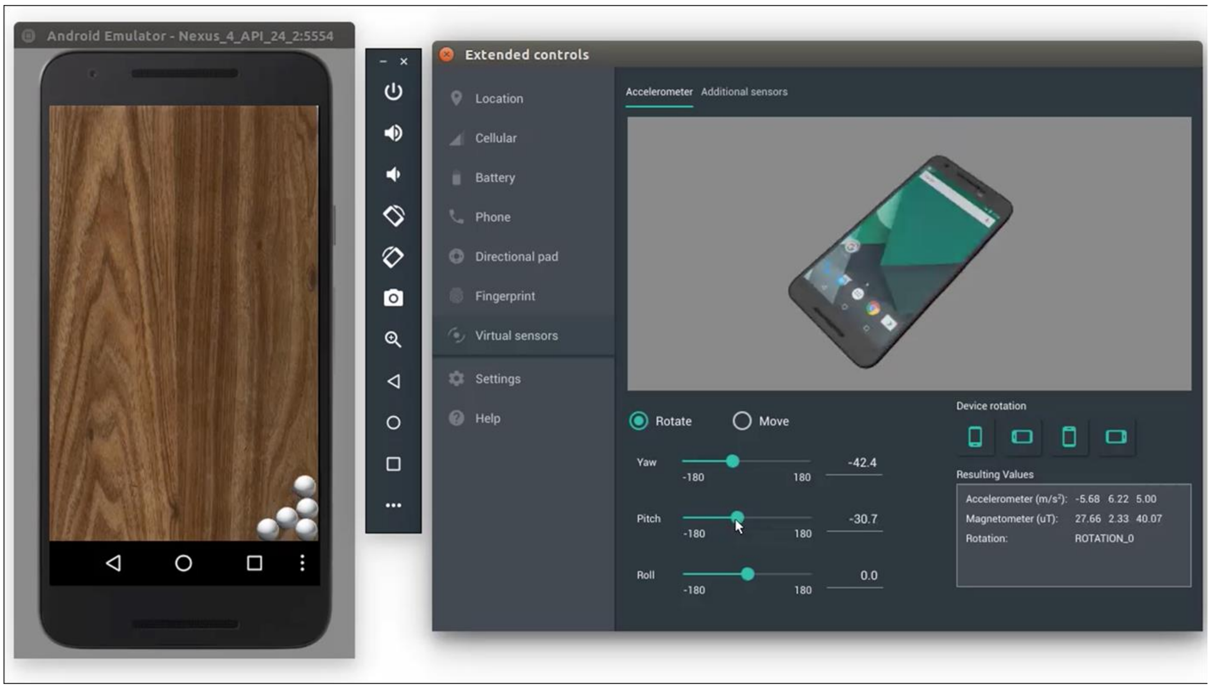
Now, we can run ADB commands to use the tool in console for both Debugging and Testing of the application. Further details and guide to the ADB can be seen here.

DDMS (Dalvik Debug Monitor Server):

The DDMS is a debugging tool used in the Android platform. The Dalvik Debug Monitor Service is downloaded as **part of the Android SDK**. Some of the services provided by the DDMS are port forwarding, on-device screen capture, on-device thread and heap monitoring, and radio state information.

It allows developers to spot bugs in applications running on either an emulator or an actual Android device. Its feature, known as Emulator Control allows developers to simulate phone states and activities. For example, it can simulate different types of networks which can have different network characteristics such as speed and latency.

This debugging tool can be integrated into the Eclipse IDE by adding the ADT (Android Development Tools) plug-in. Otherwise, it can be accessed from the command line and will automatically connect to any running emulator.



Yaw -42.4
Pitch -30.7
Roll 0.0

Accelerometer (m/s^2): -5.68 6.22 5.00
Magnetometer (uT): 27.66 2.33 40.07
Rotation: ROTATION_0

DDMS is deprecated. Its features have been replaced by other new features. Instead of this, we use Android Profiler in Android Studio 3.0 and higher to profile your app's CPU, memory, and network usage. To perform other debugging tasks, such as sending commands to a connected device to set up port-forwarding, transfer files, or take screenshots, then use the Android Debug Bridge (adb), Android Emulator, Device File Explorer, or Debugger window.