Kode : Asset Manager

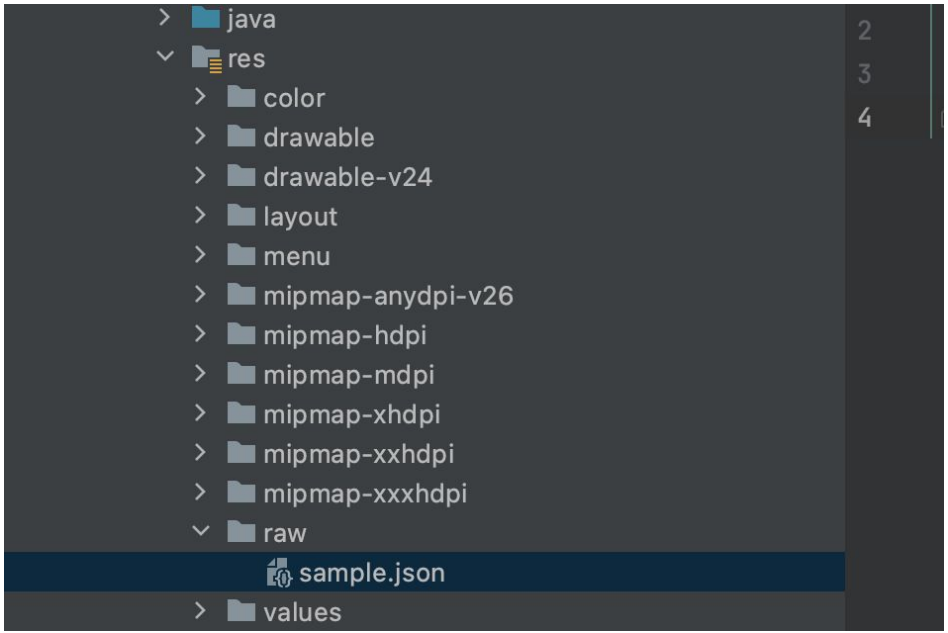
```
sayHelloButton.setOnClickListener {  it: View!

    val json = assets.open( fileName: "sample.json")
        .bufferedReader()
        .use { it.readText() }

    Log.i( tag: "JSON", json)
```

Raw Resource

Raw Resource

- Salah satu masalah ketika menggunakan Asset Manager adalah, kita harus mengetikkan lokasi resource nya menggunakan string. Hal ini akan rentan kesalahan, terutama jika lokasi file tidak ada, maka otomatis aplikasi kita akan error. Dan error ini tidak bisa dideteksi ketika kompilasi
- Android juga sebenarnya mendukung Raw Resource, memang tidak se flexible AssetManager, namun pada kasus yang tidak terlalu banyak assetnya, kita bisa gunakan Raw Resource
- Raw Resource akan secara otomatis men-generate id di class R, sehingga kesalahan tidak akan mungkin terjadi ketika aplikasinya sudah berjalan

Raw Resource



Kode : Raw Resource

```
sayHelloButton.setOnClickListener {  it: View!

    val sample = resources.openRawResource(R.raw.sample)
        .bufferedReader()
        .use { it.readText() }
    Log.i( tag: "RAW", sample);
```

Context

Context

- Dalam Android, terdapat object bernama Context. Context berisi global informasi tentang environment aplikasi. Context merupakan Abstract Class yang implementasinya sudah dilakukan oleh Android.
- Context memungkinkan kita mengakses resource, membuka activity, menerima intent, broadcasting, dan lain-lain
- https://developer.android.com/reference/android/content/Context

Jenis Context

- Di Android, terdapat 2 jenis Context, Application Context dan Activity Context
- Application Context merupakan context yang tersedia di aplikasi Android. Dan hanya dibuat sekali, alias singleton
- Application Context bisa diakses dengan function getApplicationContext()
- Activity Context merupakan context tersedia di Activity. Misal jika kita punya MainActivity, maka akan ada Activity Context untuk MainActivity itu
- Activity Context bisa diakses langsung dengan Activity nya, karena Activity adalah turunan dari Context

Kapan Menggunakan Context

- Application Context digunakan jika kita ingin membuat object yang singleton yang membutuhkan Context
- Activity Context digunakan jika kita ingin membuat object yang hanya digunakan di Activity tersebut
- Jangan menggunakan Activity Context untuk object yang singleton, karena nanti otomatis referensi ke Activity tersebut akan selalu ada, dan otomatis object Activity tidak bisa dihapus di memory Android, hal ini bisa menyebabkan memory leak

Application

Application

- Application adalah base class untuk maintain global application state
- Secara default Android akan membuat object Application sendiri, namun kita juga bisa membuat class Application sendiri, asal harus turunan dari class Application
- Dan untuk mendaftarkan class Application yang kita buat, kita harus daftarkan di Manifest File dengan tag “android:name”
- https://developer.android.com/reference/android/app/Application
- Untuk mendapatkan object Application, kita bisa menggunakan function getApplication() di Activity

Kode : Application

```
class MyApplication : Application() {

    override fun onCreate() {
        super.onCreate()
        Log.i( tag: "APP",  msg: "Application created")
    }

}
```

Kode : Manifest File

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.programmerzamannow.helloworld">

    <application
        android:name=".MyApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Hello World"
```

Device Compatibility

Device Compatibility

- Android di desain untuk bisa berjalan pada berbagai perangkat, dari smartphone, tablet sampai tv
- Banyaknya target perangkat yang bisa menjalankan Android menjadikan salah satu keuntungan, sehingga aplikasi kita bisa ditargetkan untuk banyak jenis perangkat
- Namun, agar aplikasi kita bisa berjalan dengan baik di berbagai jenis perangkat yang berbeda, aplikasi kita harus memastikan bisa mentoleransi ketersediaan fitur pada perangkat keras, dan juga UI yang harus flexible yang bisa beradaptasi dengan ukuran layar yang berbeda-beda

Controlling Application Availability

- Android mendukung banyak sekali fitur yang bisa digunakan oleh aplikasi kita melalui Android Platform API. Beberapa fitur ada yang berbasis hardware (misal compass sensor dan fingerprint), beberapa berbasis software (misal app widgets), dan beberapa tergantung versi platform
- Tidak semua perangkat Android mendukung semua fitur, jadi kita harus mengontrol aplikasi kita berdasarkan fitur yang terdapat pada perangkat nya.
- Untuk mengontrol perangkat yang bisa menjalankan aplikasi kita, kita bisa menggunakan dua cara, secara declarative atau programmatic
- Declarative artinya kita simpan pada konfigurasi aplikasi, hal ini menyebabkan kondisi wajib
- Programmatic artinya kita cek di kode aplikasi kita, jadi pengecekan dilakukan saat aplikasi berjalan

Device Feature di Manifest File

- Contoh jika kita mewajibkan sebuah fitur ada di device, dan jika tidak ada maka aplikasi kita tidak bisa dijalankan, maka kita bisa tambahkan di manifest file
- Contoh, jika aplikasi kita berbasis compass sensor, maka tidak berguna juga juga device nya tidak memiliki fitur tersebut, sehingga kita bisa mewajibkan perangkat yang memiliki fitur compass sensor
- Jika perangkat tidak memiliki fitur itu, secara otomatis dia tidak bisa menginstall aplikasi kita

Kode : Device Feature di Manifest File

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.programmerzamannow.helloworld">

    <uses-feature android:name="android.hardware.fingerprint" android:required="true"/>

    <application
        android:name=".MyApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Hello World"
```

Device Feature di Kode

- Namun jika misal fitur yang kita butuhkan hanya optional, misal contohnya fingerprint ada fitur yang boleh ada boleh tidak, kita bisa cukup melakukan pengecekan di kode program kita saja
- Kita bisa menggunakan class PackageManager untuk mengeceknya
- https://developer.android.com/reference/android/content/pm/PackageManager
- Dan untuk mendapatkan object PackageManager, kita bisa gunakan functtion getPackageManager()

Kode : Package Manager

```
    if (packageManager.hasSystemFeature(PackageManager.FEATURE_FINGERPRINT)) {
        Log.i( tag: "FEATURE",  msg: "enabled fingerprint feature")
    } else {
        Log.i( tag: "FEATURE",  msg: "disabled fingerprint feature")
    }
}
```

Platform Version

- Saat kita membuat Android, kadang kita ingin mendukung versi dari platform yang minimal versi berapa
- Dan setiap versi platform Android rilis, biasanya akan membawa fitur baru, sehingga kita perlu tahu juga, fitur yang terdapat di Android tersebut ada sejak versi platform berapa

Kode : Platform Version

```
android {
    compileSdk 32

    defaultConfig {
        applicationId "com.programmerzamannow.helloworld"
        minSdk 29
        targetSdk 32
        versionCode 1
        versionName "1.0"
```

Kode : Pengecekan Versi Platform

```
if (Build.VERSION.SDK_INT < Build.VERSION_CODES.S) {
    // menggunakan versi dibawah 32
    Log.i( tag: "VERSION",  msg: "disabled some features")
}
```

Screen Compatibility

Screen Compatibility

- Android berjalan di perangkat dengan berbagai ukuran, dari smartphone, tablet sampai tv
- Untuk mengkategorisasikan perangkat berdasarkan ukuran layar, Android mendefinisikan dalam dua karakteristik, screen size (ukuran layar) dan screen density (kepadatan pixel layar, atau DPI)
- Untuk mempermudah dan menggeneralisasi semua varian ini, Android membagi menjadi beberapa grup
- Screen size : small, normal, large dan xlarge
- Screen density : mdpi (medium), hdpi (high), xhdpi (extra high), xxhdpi (extra-extra high) dan others
- Secara default, aplikasi kita akan kompatibel dengan semua screen size dan dencity, karena Android akan melakukan penyesuaian layout UI dan gambar untuk tiap perangkat layar
- Namun untuk pengalaman yang lebih baik, ada baiknya kita menggunakan resource yang sesuai dengan tiap jenis layar

Pixel Densities

- Saat kita akan mendukung perangkat yang kepadatan pixelnya berbeda-beda, maka agak menyulitkan ketika kita ingin membuat komponent yang ukurannya fix, misal menggunakan pixel
- Contoh jika kita menggunakan komponen dengan ukuran 100px , maka akan terlihat besar di layar yang density nya kecil, tapi akan terlihat kecil jika diukuran layar yang density nya besar
- Oleh karena itu, di Android, kita jarang menggunakan ukuran px, melainkan ukuran dp (density-independent pixels)
- Ukuran dp bisa secara otomatis membesar sesuai dengan screen density.
- Jika kita ingin menghitung berapa pixel dari dp, kita bisa gunakan rumus :
  px = dp * (dpi / 160)

https://developer.android.com/training/multiscreen/screendensities

Density qualifier	Description
`ldpi`	Resources for low-density (*ldpi*) screens (~120dpi).
`mdpi`	Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.)
`hdpi`	Resources for high-density (*hdpi*) screens (~240dpi).
`xhdpi`	Resources for extra-high-density (*xhdpi*) screens (~320dpi).
`xxhdpi`	Resources for extra-extra-high-density (*xxhdpi*) screens (~480dpi).
`xxxhdpi`	Resources for extra-extra-extra-high-density (*xxxhdpi*) uses (~640dpi).

Kode : Density Independent Pixels

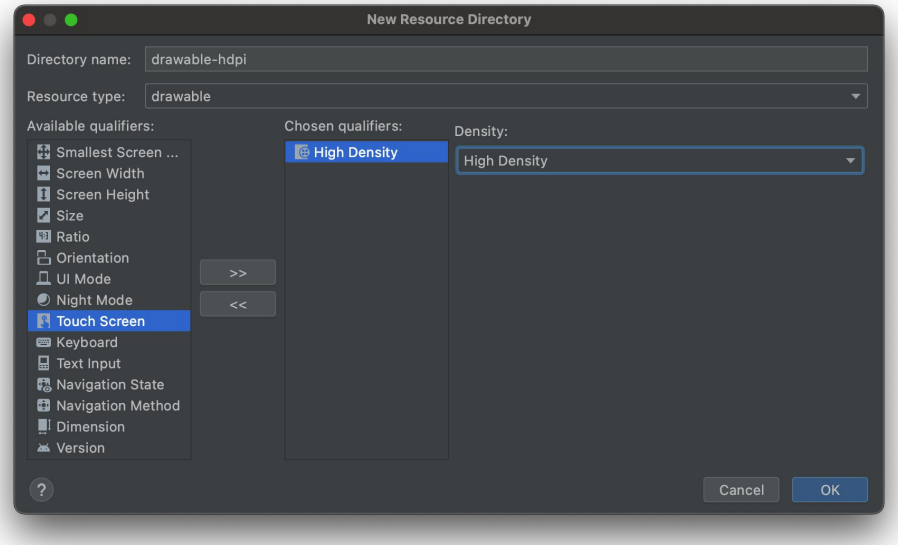
```
<ImageView
    android:layout_width="100dp"
    android:layout_height="100dp"
    android:layout_gravity="center_horizontal"
    android:contentDescription="PZN Logo"
    android:src="@drawable/pzn" />
```

Resource Compatibility

Resource Compatibility

- Pada materi Localization, kita sudah tahu bahwa resource itu bisa mendukung multi bahasa
- Di Android, resource juga mendukung multi compatibility
- Misal, ketika kita membuat gambar ukuran 100px, gambar ini akan terlihat bagus di perangkat dengan density rendah, tapi ketika menggunakan perangkat dengan density tinggi, maka gambar itu ada kemungkinan akan pecah
- Oleh karena itu, kadang kita perlu mendukung beberapa jenis resource tergantung dari perangkat yang ingin kita dukung

Membuat Resource dengan Compatibility



Debugging

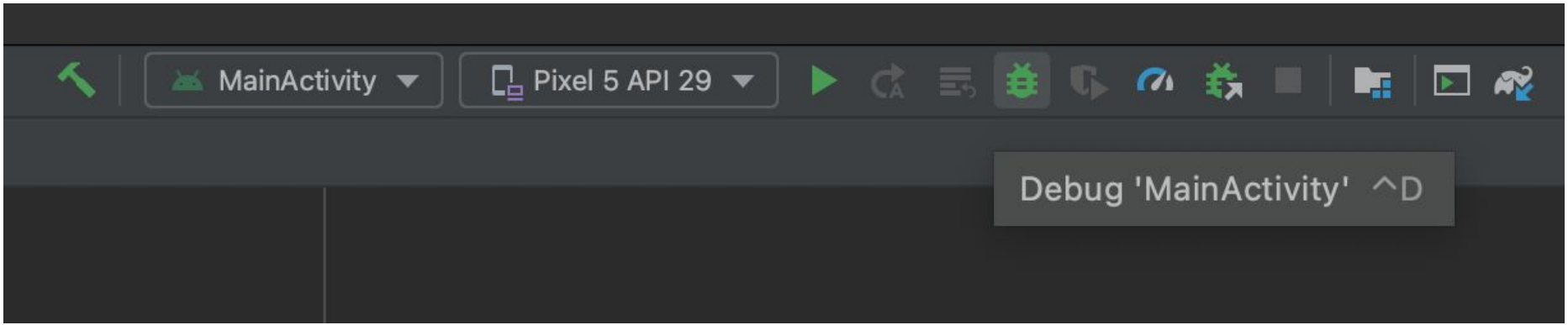
Debugging

- Saat membuat aplikasi, kesalahan sering terjadi
- Dan kadang, agak sulit untuk mencari kesalahan tersebut, sehingga kita butuh menelusuri alur kode program kita satu per satu
- Untungnya, Android memiliki fitur debugging, dimana kita bisa menghentikan program yang sedang berjalan, dan melanjutkan jalannya program secara bertahap sesuai yang kita inginkan
- Fitur ini sangat tergantung dengan Android Studio

Kode : Debugging

```
7
8       private fun printHello(name: String) {
9           Log.i( tag: "DEBUG", name)
0       }
1   }
2
```

Menjalankan dalam Mode Debug



Testing

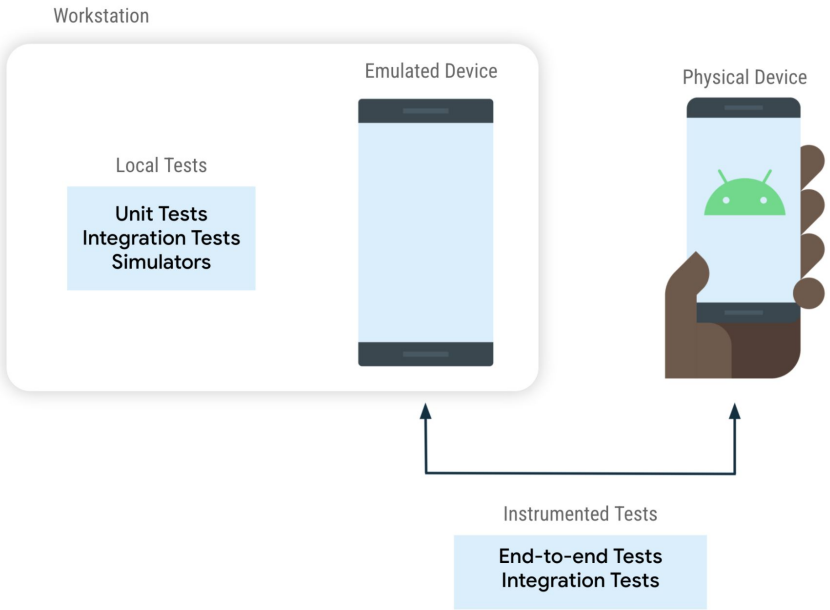
Testing

- Testing adalah salah satu tahapan dalam pembuatan aplikasi, termasuk di Android
- Android sudah mendukung testing dengan baik, baik itu unit test sampai end to end test

Jenis Testing di Android

- Instrumented Test, yaitu test yang di jalankan di perangkat Android, baik itu physical atau emulator. Aplikasi akan di install di perangkat Android, lalu test akan melakukan pengetesan pada aplikasi yang berjalan dengan mengirim perintah-perintah. Instrumented Test bisa dibilang juga Integration Test atau End to End Test.
- Local Test, yaitu test yang dieksekusi di development machine kita, misal di laptop atau di server. Biasanya local test itu kecil dan cepat, dan biasanya di dalam local test tidak membutuhkan device Android untuk berjalan. Kita bisa bilang juga dengan nama Unit Test

Jenis Testing di Android



Kenapa Automation Testing Penting?

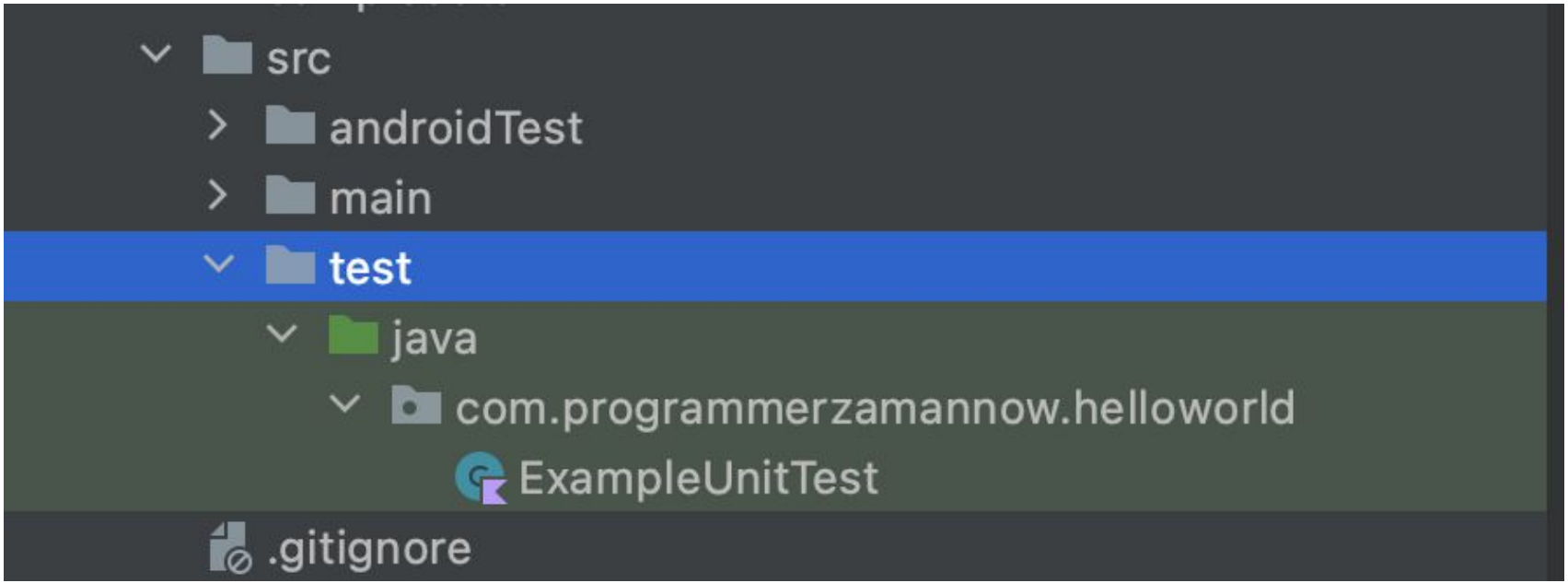
- Dengan menjalankan test terhadap aplikasi kita secara konsisten, kita bisa memverifikasi kebenaran aplikasi kita, fungsionalitas aplikasi, sebelum kita merilis aplikasi ke public
- Sebenarnya kita bisa saja melakukan test secara manual, dengan cara menjalankan aplikasi di Device, lalu mencoba semua fitur.
- Namun, manual test sulit untuk dikerjakan secara konsisten, rentang kesalahan dan jika butuh cepat, tidak bisa dilakukan secara otomatis juga.
- Oleh karena itu automation test sangat penting dalam pengembangan aplikasi, terutama aplikasi Android

Local Test

Local Test

- Android menggunakan JUnit untuk melakukan Unit Test atau Local Test
- Pada saat dibuatnya video ini, versi JUnit yang digunakan oleh Android masih menggunakan JUnit 4, padahal versi terbaru adalah JUnit 5
- Namun jangan khawatir, karena jika kita sudah terbiasa menggunakan JUnit 5, harusnya akan lebih mudah menggunakan JUnit 4

Folder Unit Test



Instrumentation Test

Instrumentation Test

- Unit Test hanya bisa digunakan untuk melakukan pengetesan kode program kita
- Saat kita membuat aplikasi Android, kadang banyak sekali ketergantungan dengan sistem operasi Android nya itu sendiri, oleh karena itu kadang agak sulit untuk membuat unit test ketika kita butuh melakukan pengetesan interaksi dengan UI aplikasi kita
- Android mendukung Instrumentation Test, atau sederhananya adalah UI Test Automation
- Dengan menggunakan Instrumentation Test, kita bisa membuat automation test mirip robot, yang mensimulasikan pengguna aplikasi kita

Espresso

- JUnit hanya digunakan untuk library Unit Test, untuk melakukan UI Test, Android menggunakan library bernama Espresso
- Dengan Espresso, kita bisa membuat UI Test dengan mudah
- Ada banyak sekali fitur Espresso, dan kita akan belajar secara bertahap di roadmap kelas Android ini
- https://developer.android.com/training/testing/espresso

Activity Scenario

- Saat kita melakukan instrumentation test, biasanya kita akan melakukan UI Test, dan untuk menampilkan UI, kita biasanya butuh Activity
- Dalam Instrumentation Test, kita bisa menggunakan ActivityScenario untuk menjalankan sebuah Activity
- https://developer.android.com/reference/androidx/test/core/app/ActivityScenario

Kode : Activity Scenario

```
@RunWith(AndroidJUnit4::class)
class MainActivityTest {

    lateinit var activityScenario: ActivityScenario<MainActivity>

    @Before
    fun setUp() {
        activityScenario = ActivityScenario.launch(MainActivity::class.java)
    }

    @After
    fun tearDown() {
        activityScenario.close()
    }
```

Activity Scenario Rule

- Menjalankan Activity secara manual menggunakan Activity Scenario tidak direkomendasikan ketika kita membuat Instrumentation Test
- Kita bisa memanfaatkan ActivityScenarioRule yang bisa di integrasikan dengan JUnit Rule, yang secara otomatis bisa menjalankan Activity ketika unit test mulai dan menghentikan Activity ketika unit test selesai
- https://developer.android.com/reference/androidx/test/ext/junit/rules/ActivityScenarioRule

Kode : Activity Scenario Rule

```
 */
@RunWith(AndroidJUnit4::class)
class MainActivityTest {

    @get:Rule
    var activityScenarioRule = ActivityScenarioRule(MainActivity::class.java)
```

Espresso Package

- Espresso berisikan banyak class yang bisa digunakan untuk mempermudah kita melakukan Instrumentation Test
- Karena terlalu banyak, jadi kita akan membahasnya sambil kelasnya berjalan, dan sambil kita praktekan
- Package Espresso ada di androidx.test.espresso
- https://developer.android.com/reference/androidx/test/espresso/package-summary?hl=en

Kode : Instrumentation Test

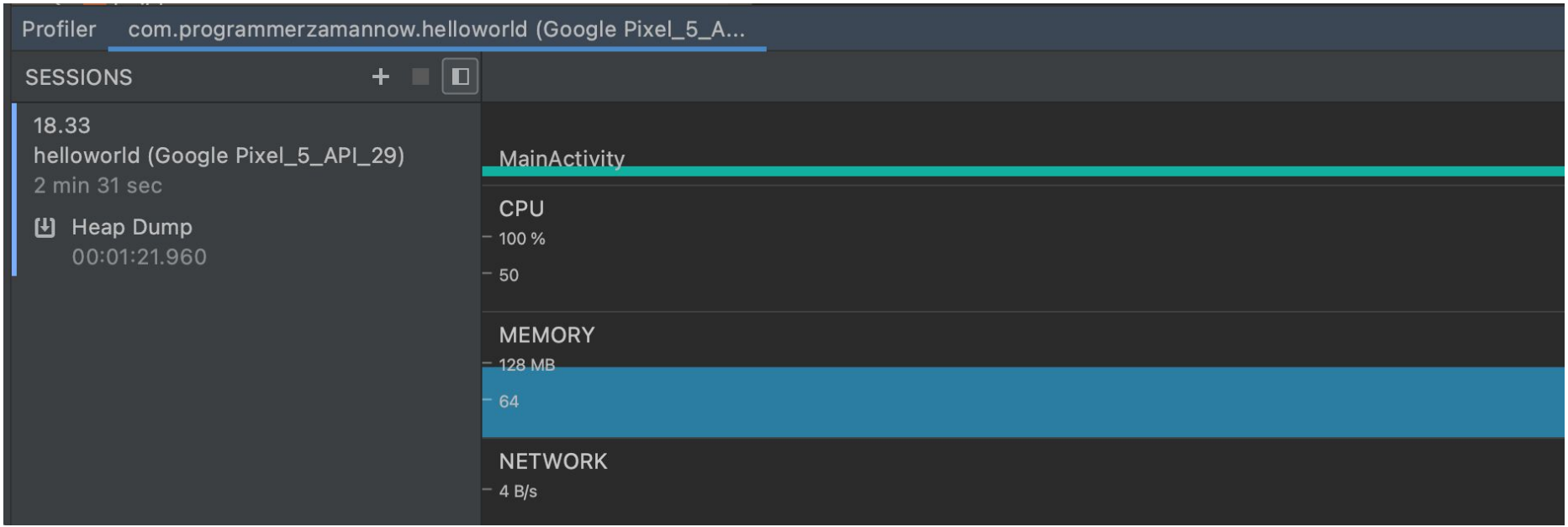
```
@Test
fun testMainActivity() {
    val context = ApplicationProvider.getApplicationContext<Context>()
    val name = "Eko"

    Espresso.onView(ViewMatchers.withId(R.id.nameEditText))
        .perform(ViewActions.typeText(name))

    onView(withId(R.id.sayHelloButton))
        .perform(click())

    onView(withId(R.id.sayHelloTextView))
        .check(matches(withText(context.getString(R.string.sayHelloTextView, name))))
}
```

Profiling

Profiling

- Debugging, Unit Test dan Instrumentation Test adalah salah satu cara untuk meminimalisir kesalahan ketika membuat aplikasi
- Namun kadang-kadang ada masalah yang mungkin tidak terdeteksi oleh debugging, unit test atau bahkan instrumentation test
- Contoh yang sering terjadi adalah masalah memory leak
- Pada kasus ini, fitur Profiling sangatlah berguna
- Android mendukung fitur Profiling, dimana kita bisa memonitor aplikasi kita yang berjalan
- Kita bisa memonitor penggunaan memory, cpu, dan lain-lain

Profiler

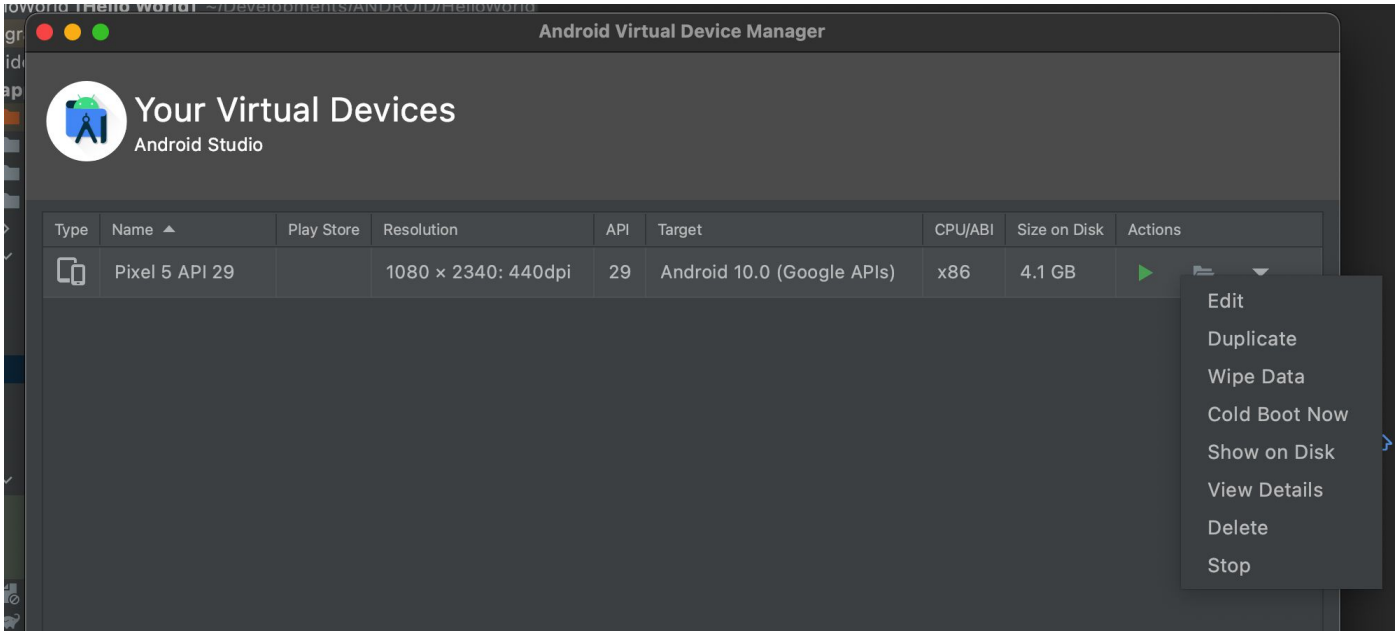

Profiler
com.programmerzamannow.helloworld (Google Pixel_5_A...
SESSIONS
18.33
helloworld (Google Pixel_5_API_29)
2 min 31 sec
Heap Dump
00:01:21.960
MainActivity
CPU
100 %
50
MEMORY
128 MB
64
NETWORK
4 B/s

Reset Android Studio

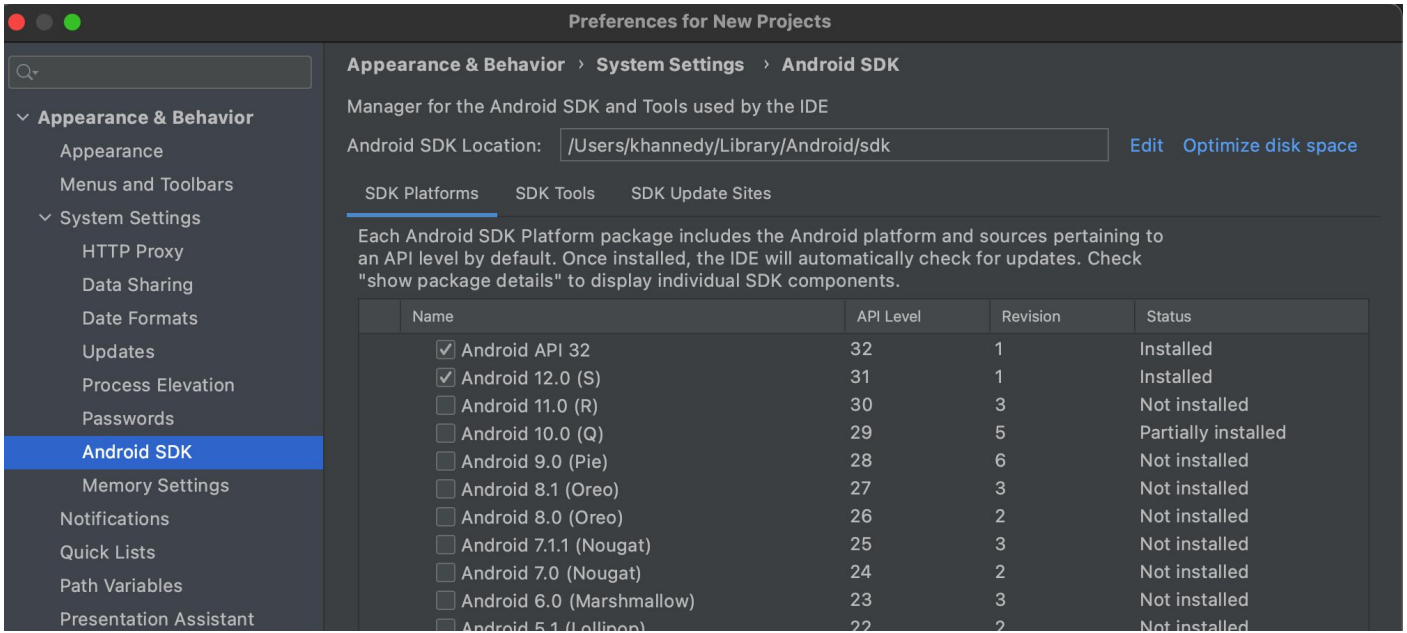
Reset Android Studio

- Kadang ada kalanya kita sering menghadapi error ketika membuka atau menjalankan project Android, entah error di Android SDK nya, atau error di Android Studio nya
- Pada kasus yang error seperti ini, yang tidak ada hubungannya dengan kode program kita, kita bisa mencoba melakukan reset Android Studio kita
- Harapannya adalah, kita bisa membuka Android Studio seperti ketika pertama kali kita menginstall Android Studio

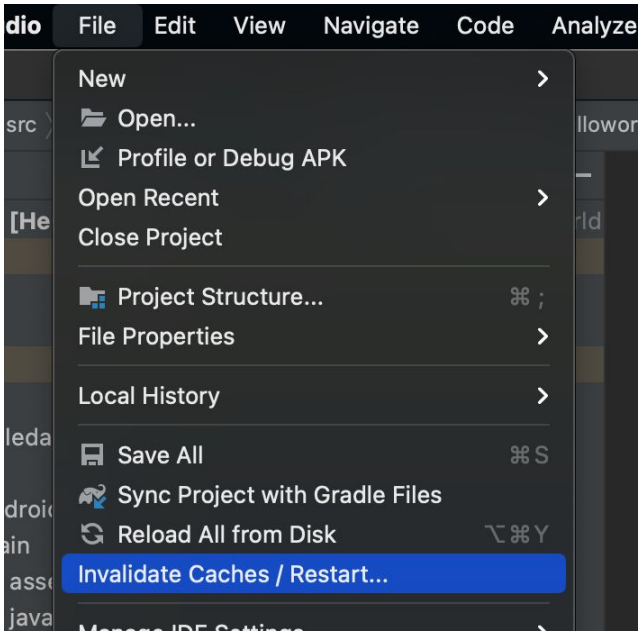
Hapus Emulator



Hapus Android SDK

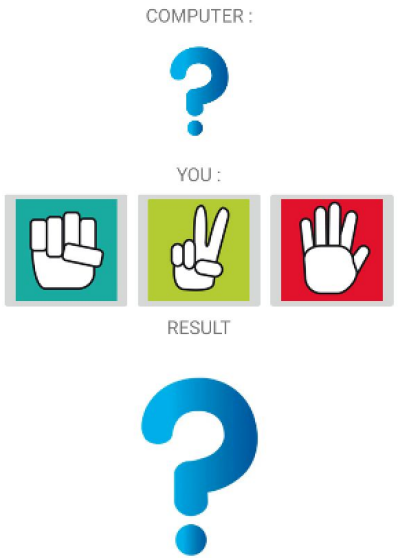


Invalidate Cache



Membuat Game Batu Gunting Kertas

Game Batu Gunting Kertas



Materi Selanjutnya

Materi Selanjutnya

- Android Activity
- Android Layout
- Android App Navigation
- Android UI
- Dan lain-lain