



Kotlin

Kotlin in Android Studio & OOP in Kotlin

Daftar Isi

A. Membuat Virtual Device dengan Emulator	3
B. Menjalankan Emulator Android	5
C. Debugging Android	7
D. Basic Kotlin ArrayList & ArrayOf	14
E. Companion Object	18
F. Constructor Kotlin	19
G. Class and Object in Kotlin	21
H. Getter Setter Kotlin	24
I. Inheritance pada Kotlin	26
J. Nested dan Inner Class	28
K. Enum Pada Kotlin	29
References	32

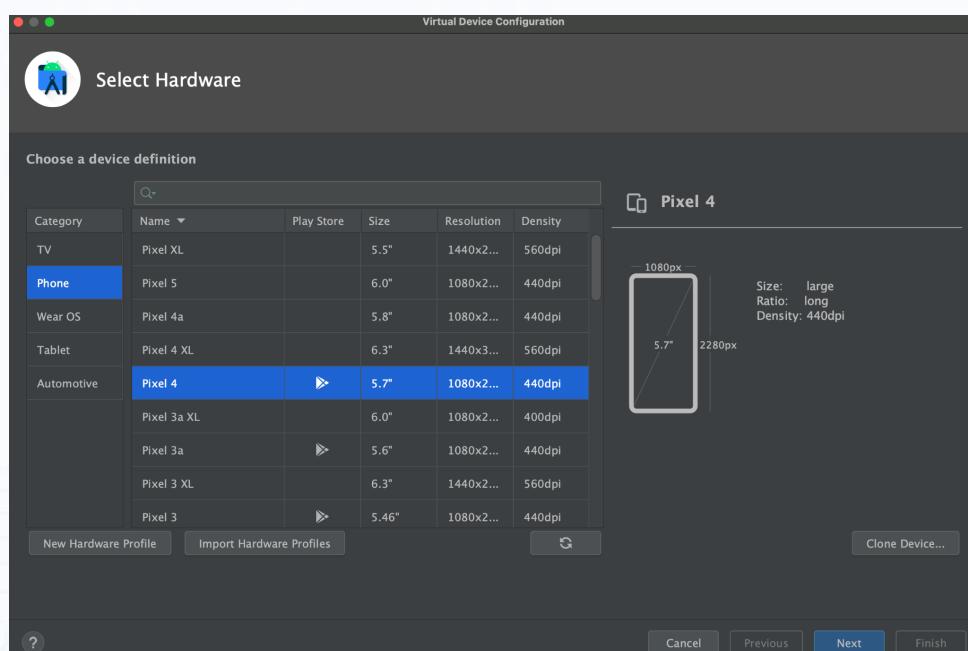
A. Membuat Virtual Device dengan Emulator

Perangkat Virtual Android (AVD) adalah konfigurasi yang menetapkan karakteristik ponsel dan tablet Android, Wear OS, Android TV, atau perangkat Automotive OS yang ingin kita simulasikan dalam Android Emulator. Pengelola Perangkat adalah antarmuka yang dapat diluncurkan dari Android Studio untuk membantu kita dalam membuat dan mengelola AVD.

Berikut ini merupakan langkah-langkah untuk membuat AVD baru:

1. Buka Device Manager
2. Klik Create Device

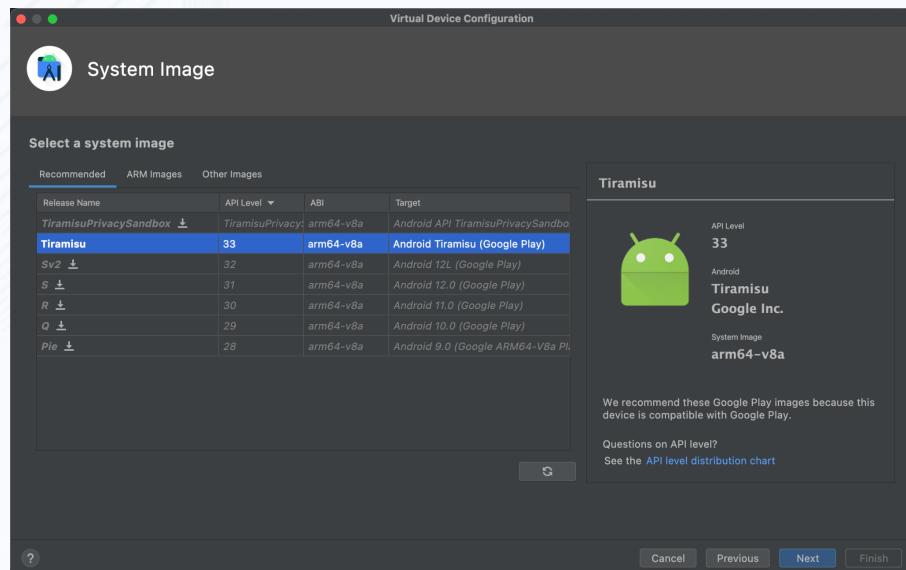
Jendela Select Hardware akan ditampilkan.



Ingat bahwa hanya beberapa profil hardware yang menyatakan menyertakan Play Store. Ini menunjukkan bahwa beberapa profil tersebut sepenuhnya mematuhi CTS dan dapat menggunakan image sistem yang menyertakan aplikasi Play Store.

3. Pilih profil hardware, lalu klik Next.

Jika tidak melihat profil hardware yang diinginkan, kita dapat membuat atau mengimpor profil hardware. Halaman System Image akan muncul.



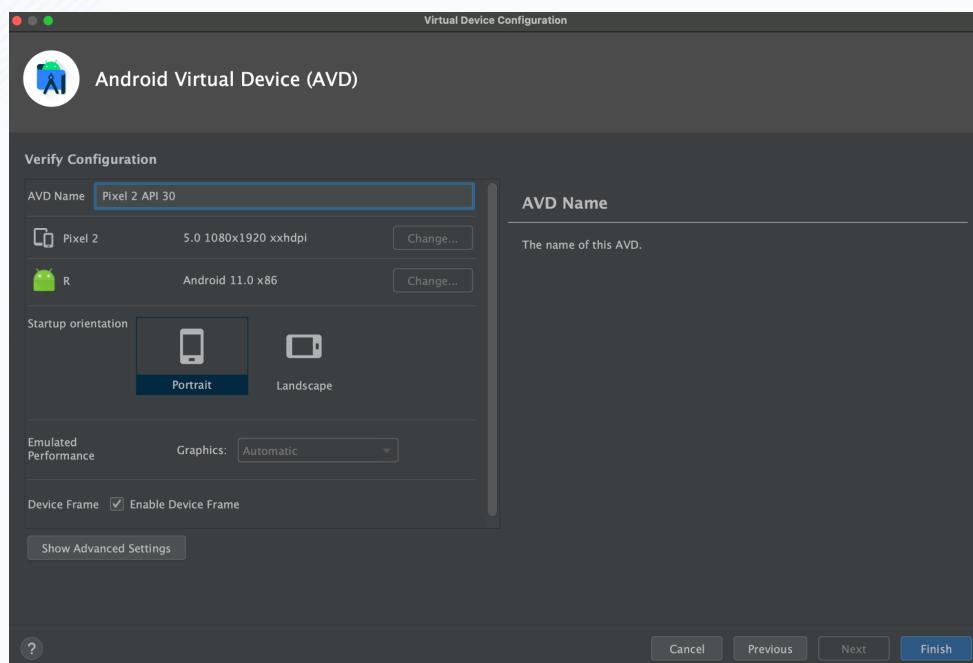
4. Pilih image sistem untuk API level tertentu, lalu klik Next.

Tab Recommended mencantumkan image sistem yang direkomendasikan. Tab lainnya berisi daftar yang lebih lengkap. Panel sebelah kanan menunjukkan image sistem yang dipilih. Image x86 berjalan paling cepat dalam emulator.

Jika melihat Download di sebelah image sistem, kita harus mengkliknya untuk mendownload image sistem. Kita harus terhubung ke Internet untuk mendownloadnya.

Penting untuk mengetahui API level perangkat target karena aplikasi tidak akan dapat berjalan pada image sistem dengan API level yang lebih rendah dari yang disyaratkan oleh aplikasi, seperti yang ditetapkan dalam atribut minSdkVersion file manifest aplikasi.

Jika aplikasi kita mendeklarasikan elemen <uses-library> dalam file manifes, aplikasi memerlukan image sistem tempat menyimpan library eksternal tersebut. Jika kita ingin menjalankan aplikasi pada emulator, buat AVD yang menyertakan library yang diperlukan. Untuk melakukannya, kita mungkin perlu menggunakan komponen add-on untuk platform AVD; misalnya, add-on Google API yang berisi library Google Maps. Halaman Verify Configuration akan muncul.

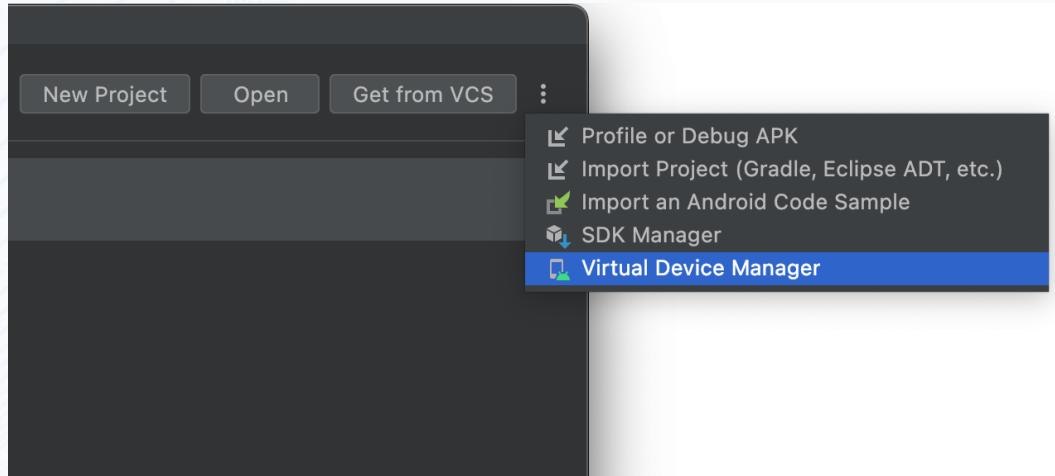


5. Ubah AVD properties sesuai kebutuhan, lalu klik Finish. Klik Show Advanced Settings untuk menampilkan setelan lainnya, seperti skin. AVD baru akan muncul di tab Virtual pada Pengelola Perangkat dan menu drop-down target.

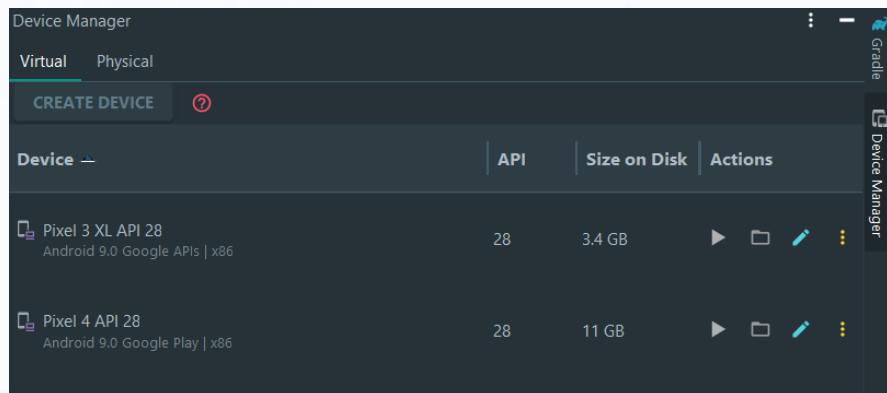
B. Menjalankan Emulator Android

Untuk membuka Pengelola Perangkat baru, lakukan salah satu hal berikut:

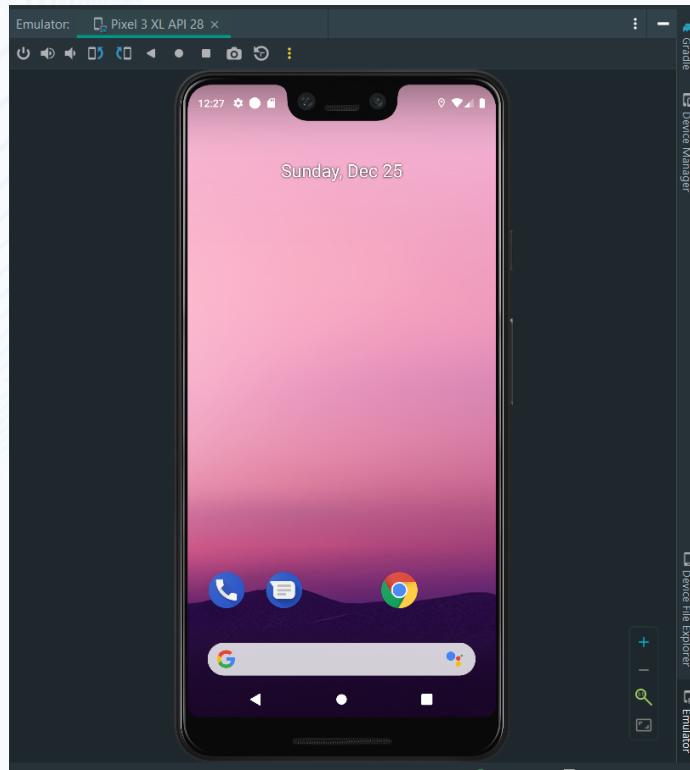
1. Dari layar Selamat Datang Android Studio, pilih More Actions > Virtual Device Manager.



2. Setelah membuka project, pilih View > Tool Windows > Device Manager dari panel menu utama. Maka akan muncul tab window Device Manager di sebelah sisi kanan area editor seperti pada gambar di bawah ini.



3. Klik icon button play pada salah satu device / emulator yang ingin dijalankan pada window Device Manager. Tunggu beberapa saat untuk Android Studio menjalankan dan menampilkan emulator di window Emulator seperti gambar di bawah ini.



C. Debugging Android

Android Studio menyediakan debugger yang memungkinkan kita melakukan hal-hal berikut ini dan banyak lagi:

- Memilih perangkat untuk men-debug aplikasi kita.
- Menetapkan breakpoints dalam kode Java, Kotlin, dan C/C++.
- Memeriksa variabel dan mengevaluasi ekspresi pada saat runtime.

Mengaktifkan proses debug

Sebelum mulai men-debug, kita perlu mempersiapkan hal-hal berikut:

1. Aktifkan proses debug di perangkat :

Jika kita menggunakan emulator, proses ini diaktifkan secara default.

Namun, untuk perangkat yang terhubung, kita perlu mengaktifkan proses debug dalam opsi developer perangkat.

2. Jalankan varian build yang dapat di-debug:

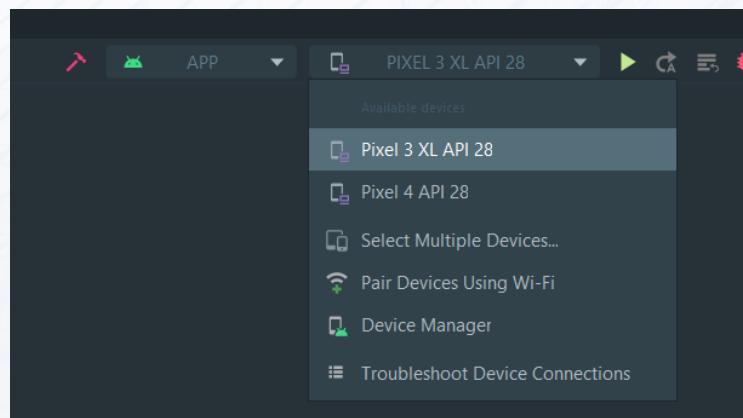
Kita harus menggunakan build variant yang menyertakan debuggable true dalam konfigurasi build. Biasanya, kita cukup memilih varian "debug" default yang disertakan dalam setiap project Android Studio (meskipun tidak terlihat dalam file build.gradle). Namun, jika kita menentukan jenis build baru yang seharusnya dapat di-debug, kita harus menambahkan `debuggable true` ke jenis build tersebut:

```
android {  
    buildTypes {  
        customDebugType {  
            debuggable true  
            ...  
        }  
    }  
}
```

Memulai proses debug

Kita dapat memulai sesi proses debug dengan langkah berikut:

1. Tetapkan beberapa breakpoints dalam kode aplikasi.
2. Pada toolbar, pilih perangkat untuk men-debug aplikasi kita dari menu drop-down perangkat target.

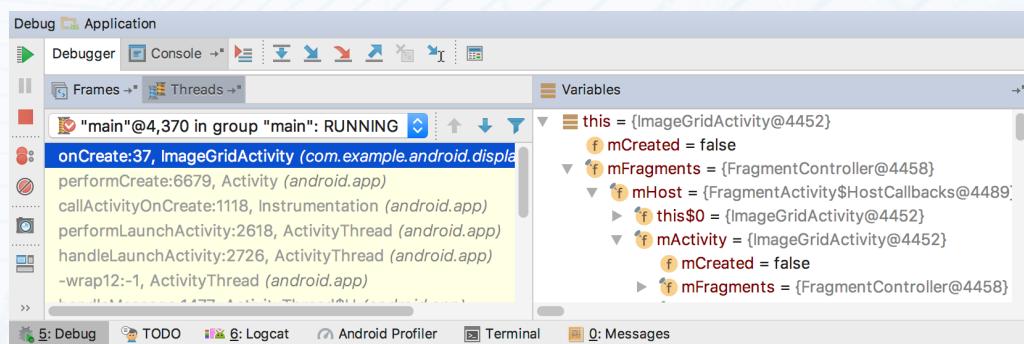


Jika tidak memiliki perangkat yang dikonfigurasi, kita harus menghubungkan perangkat melalui USB atau membuat AVD untuk menggunakan Android Emulator.

3. Di toolbar, klik Debug .

Jika kita melihat dialog yang menanyakan apakah kita ingin "switch from Run to Debug" (beralih dari Run ke Debug), berarti aplikasi kita sudah berjalan di perangkat dan aplikasi akan dimulai ulang untuk memulai proses debug. Jika kita ingin instance aplikasi yang sama tetap berjalan, klik Cancel Debug, lalu instal debugger ke aplikasi yang sedang berjalan. Jika tidak, Android Studio akan membuat APK, menandatanganinya dengan kunci debug, menginstalnya di perangkat yang kita pilih, dan menjalankannya.

4. Jika jendela Debug tidak terbuka, pilih View > Tool Windows > Debug (atau klik Debug  di kolom jendela alat), lalu klik tab Debugger, seperti ditunjukkan pada gambar di bawah ini.



Memasang debugger ke aplikasi yang sedang berjalan

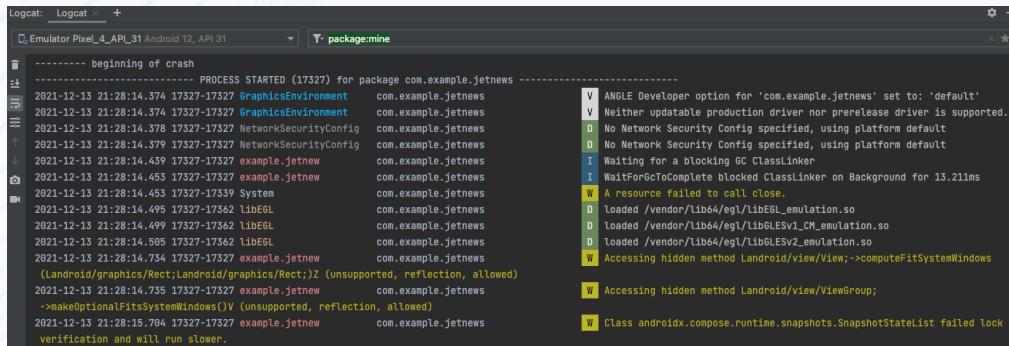
Jika aplikasi kita sudah berjalan di perangkat, kita dapat memulai proses debug tanpa memulai ulang aplikasi dengan cara berikut:

1. Klik Attach debugger to Android process .
2. Pada dialog Choose Process, pilih proses yang kita inginkan untuk dipasangi debugger. Jika kita menggunakan emulator atau perangkat yang telah di-root, kita dapat mencentang Show all processes untuk melihat semua proses. Dari menu drop-down Use Android Debugger Settings from, kita bisa memilih konfigurasi run/debug yang ada. (Untuk kode C dan C++, ini memungkinkan kita menggunakan kembali perintah startup LLDB, perintah post-attach LLDB, dan direktori simbol dalam konfigurasi yang ada.) Jika kita tidak memiliki konfigurasi run/debug yang ada, pilih Create New. Pilihan ini mengaktifkan menu drop-down Debug Type, tempat kita dapat memilih tipe debug lainnya. Secara default, Android Studio menggunakan tipe debug Auto guna memilih opsi debugger terbaik untuk kita berdasarkan apakah project kita menyertakan kode Java atau C/C++.
3. Klik OK. Jendela Debug akan muncul.

Menggunakan log sistem

Log sistem akan menampilkan pesan sistem saat kita men-debug aplikasi. Pesan ini berisi informasi dari aplikasi yang berjalan di perangkat. Jika kita ingin menggunakan log sistem untuk men-debug aplikasi, pastikan kode kita menulis pesan log dan mencetak pelacakan tumpukan untuk pengecualian

ketika aplikasi berada dalam tahap pengembangan. Berikut ini merupakan tampilan window logcat pada Android Studio.



The screenshot shows the Android Studio Logcat interface. The title bar says "Logcat: Logcat x +". Below it, a dropdown menu shows "Emulator Pixel_4_API_31 Android 12, API 31". A search bar contains the text "package:mine". The main area displays log entries from the application "com.example.jetnews". The log starts with "PROCESS STARTED (17327) for package com.example.jetnews". It shows various system and application logs, including network security config, EGL loading, and reflection calls. At the bottom, there's a warning about a failed lock verification. The right side of the screen has a vertical toolbar with icons for file operations like copy, paste, and delete.

Menulis pesan log dalam kode kita

Untuk menulis pesan log dalam kode, gunakan class Log. Pesan log membantu kita memahami alur eksekusi dengan mengumpulkan output debug sistem saat kita berinteraksi dengan aplikasi. Pesan log dapat memberitahukan bagian mana dari aplikasi kita yang gagal.

Contoh berikut menunjukkan bagaimana kita dapat menambahkan pesan log untuk menentukan apakah informasi status sebelumnya tersedia ketika aktivitas dimulai:

```
import android.util.Log
...
private val TAG: String = MyActivity::class.java.simpleName
...
class MyActivity : Activity() {
    ...
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        if (savedInstanceState != null) {
            Log.d(TAG, "onCreate() Restoring previous state")
            /* restore state */
        } else {
            Log.d(TAG, "onCreate() No saved state available")
            /* initialize app */
        }
    }
}
```

Selama pengembangan, kode kita juga dapat menangkap pengecualian dan menulis pelacakan tumpukan ke log sistem:

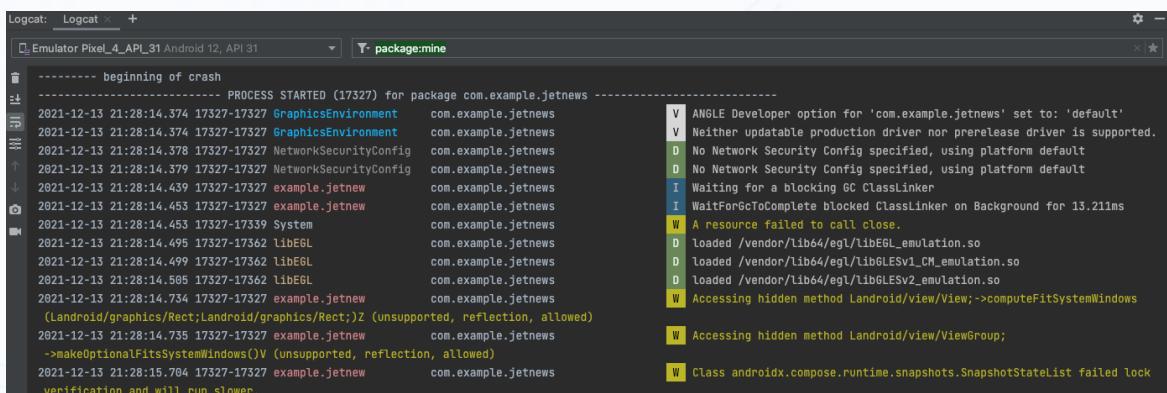
```

fun someOtherMethod() {
    try {
        ...
    } catch (e : SomeException) {
        Log.d(TAG, "someOtherMethod()", e)
    }
}

```

Menampilkan log sistem

Kita dapat menampilkan dan memfilter pesan debug dan pesan sistem lainnya di jendela Logcat. Misalnya, kita dapat melihat pesan saat pembersihan sampah memori dilakukan, atau pesan yang kita tambahkan ke aplikasi dengan class Log. Untuk menggunakan logcat, mulai proses debug, lalu pilih tab Logcat di toolbar bagian bawah seperti ditunjukkan pada gambar di bawah ini.



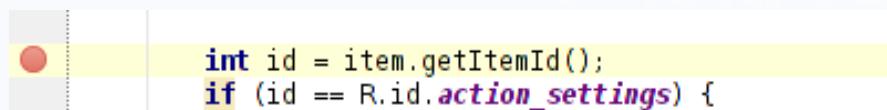
The screenshot shows the Android Studio Logcat window. The title bar says 'Logcat: Logcat'. Below it, there's a dropdown menu showing 'Emulator Pixel_4_API_31 Android 12, API 31' and a search bar with 'package:mine'. The main area displays log messages from the application. At the top, it says '----- beginning of crash -----'. The log entries are color-coded with icons: V (blue), D (orange), I (green), W (yellow), and E (red). Some entries are truncated with '...', and others have file paths like 'Landroid/graphics/Rect;J' or 'com/example/jetnews'. The log ends with 'verification and will run slower.'

Bekerja dengan breakpoints

Android Studio mendukung beberapa jenis breakpoints yang memicu tindakan debug yang berbeda. Jenis yang paling umum adalah breakpoints baris yang menjeda eksekusi aplikasi pada baris kode yang ditetapkan. Saat dijeda, kita dapat memeriksa variabel, mengevaluasi ekspresi, kemudian melanjutkan eksekusi baris demi baris untuk menentukan penyebab error

waktu proses. Untuk menambahkan breakpoints baris, lakukan langkah berikut:

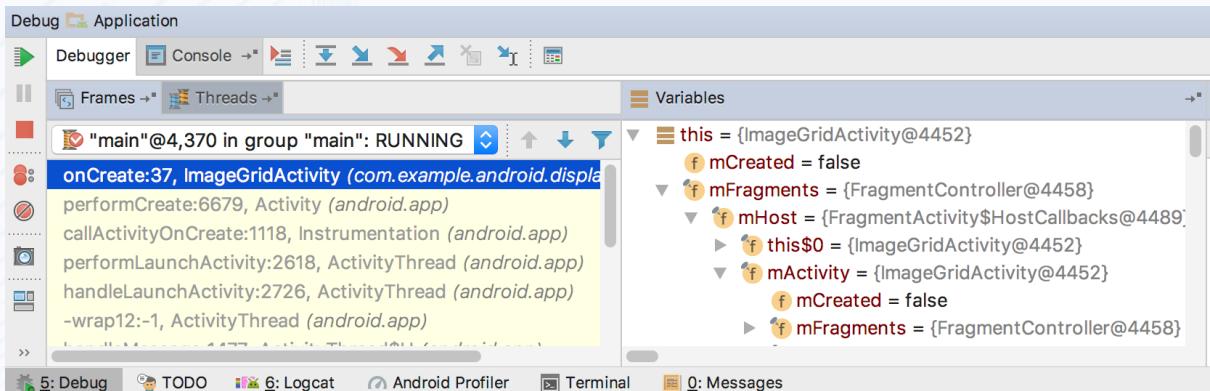
1. Temukan baris kode tempat eksekusi akan dijeda dan klik bagian tepi kiri di sepanjang baris kode tersebut, atau letakkan titik sisipan pada baris dan tekan Control+F8 (di Mac, Command+F8).
2. Jika aplikasi sudah berjalan, kita tidak perlu mengupdatenya untuk menambahkan breakpoints. Cukup klik Attach debugger to Android process  . Jika tidak, mulai proses debug dengan mengklik Debug .



Saat eksekusi kode mencapai breakpoints, Android Studio akan menjeda eksekusi aplikasi kita. Selanjutnya kita dapat menggunakan alat di tab Debugger untuk mengidentifikasi status aplikasi:

- Untuk memeriksa struktur objek dari sebuah variabel, luaskan struktur tersebut dalam tampilan Variables. Jika tampilan Variables tidak terlihat, klik Restore Variables View .
- Untuk mengevaluasi ekspresi pada titik eksekusi saat ini, klik Evaluate Expression .
- Untuk maju ke baris berikutnya dalam kode (tanpa memasukkan metode), klik Step Over .
- Untuk maju ke baris pertama dalam panggilan metode, klik Step Into .
- Untuk maju ke baris berikutnya di luar metode saat ini, klik Step Out .
- Untuk terus menjalankan aplikasi secara normal, klik Resume Program .

Saat Android Studio men-deploy aplikasi kita ke perangkat target, jendela Debug akan terbuka dengan tab atau tampilan sesi debug untuk setiap proses debugger, seperti ditunjukkan pada gambar di bawah ini.



D. Basic Kotlin ArrayList & ArrayOf

Array dan ArrayList umumnya digunakan dalam bahasa pemrograman seperti Java dan turunannya (Kotlin). Array adalah struktur data yang membantu menyimpan elemen data dari tipe yang sama. Itu statis. Oleh karena itu, tidak mungkin untuk menyimpan lebih banyak elemen di dalamnya daripada ukuran array yang dideklarasikan.

Di sisi lain, ArrayList adalah kelas Koleksi panjang variabel. Menggunakan kelas itu, programmer dapat membuat struktur data ArrayList. Keuntungan utama dari ArrayList adalah dinamis. Dengan kata lain, programmer dapat menambah atau menghapus elemen sesuai kebutuhan. Saat menggunakan ArrayList, pemrogram dapat menggunakan metode yang telah ditentukan seperti menambah, menghapus, dll. Selain itu, dapat berisi elemen duplikat, dan juga mempertahankan urutan data yang dimasukkan.

Berikut ini merupakan contoh inisialisasi variabel ArrayList yang bertipe data String dan elemen-elemen langsung dimasukkan bersamaan dengan inisialisasi variabel tersebut.

```
fun main() {
    val namaNasabah = arrayListOf<String>("Asep", "Budi", "Cantika")

    for(nama in namaNasabah) println(nama)
}
```

Kode di atas akan menghasilkan output sebagai berikut :

```
Asep  
Budi  
Cantika
```

Kita juga dapat membuat ArrayList kosong yang bertipe data string dan nantinya akan ditambahkan beberapa elemen dengan menggunakan metode add().

```
fun main() {  
    var namaKota = ArrayList<String>()  
    namaKota.add("Jakarta")  
    namaKota.add("Bandung")  
  
    println("Nama kota 1 :")  
    for(kota in namaKota) println(kota)  
  
    namaKota.add( 1 , "Semarang")  
    println("\nNama kota setelah penambahan baru :")  
    for(kota in namaKota) println(kota)  
}
```

Kode di atas akan menghasilkan output sebagai berikut.

```
Nama kota 1 :  
Jakarta  
Bandung  
  
Nama kota setelah penambahan baru :  
Jakarta  
Semarang  
Bandung
```

Kita juga dapat menambahkan semua elemen dari *collection* yang ditentukan ke dalam *list* saat ini dengan menggunakan metode addAll(). Berikut ini merupakan contoh kode programnya :

```
fun main() {  
    val keranjangLama = arrayListOf("Apel", "Mangga", "Nanas")  
    val keranjangBaru = arrayListOf<String>()  
    keranjangBaru.addAll(keranjangLama)  
  
    println("Isi pada Keranjang Baru :")  
    for(buah in keranjangBaru)  
        println(buah)  
}
```

Kode di atas akan menghasilkan output sebagai berikut.

```
Isi pada Keranjang Baru :  
Apel  
Mangga  
Nanas
```

Untuk mendapatkan nilai dari elemen tertentu pada sebuah ArrayList kita dapat menggunakan metode `get()` yang sudah disediakan oleh Kotlin, berikut ini merupakan contoh implementasinya :

```
fun main() {  
    var namaKota = ArrayList<String>()  
    namaKota.add("Jakarta")  
    namaKota.add("Bandung")  
    namaKota.add("Semarang")  
    namaKota.add("Medan")  
  
    println("Nama kota ke-2 : ${namaKota.get(1)}")  
    println("Nama kota ke-4 : ${namaKota.get(3)}")  
}
```

Kode di atas akan menghasilkan output sebagai berikut.

```
Nama kota ke-2 : Bandung  
Nama kota ke-4 : Medan
```

Class Kotlin ArrayList memiliki beberapa metode yang dapat digunakan untuk menghapus elemen dari instance-nya. Beberapa metode penghapusan ini adalah seperti yang ditunjukkan di bawah ini:

- `clear()`
- `remove()`
- `removeAll()`
- `retainAll()`
- `removeLast()`

Metode `clear()` akan menghapus semua elemen dari ArrayList seperti yang ditunjukkan di bawah ini:

```
val myArrayList = arrayListOf(1, 2, 3)

println(myArrayList) // [1, 2, 3]
myArrayList.clear()

println(myArrayList) // []
```

Metode `remove()` akan menghapus elemen yang kita tentukan sebagai argumennya. Ketika `ArrayList` berisi banyak elemen identik, hanya elemen pada indeks terendah yang akan dihapus. Ketika elemen tidak ditemukan, metode ini tidak melakukan apa-apa:

```
val myArrayList = arrayListOf(1, 2, 3, 1)

myArrayList.remove(1)
println(myArrayList) // [2, 3, 1]

myArrayList.remove(20)
println(myArrayList) // [2, 3, 1]
```

Metode `removeAll()` menerima daftar sebagai argumennya, dan menghapus semua elemen dalam `ArrayList` yang identik dengan yang ada dalam daftar. Berikut adalah contoh metode `removeAll()`:

```
val myArrayList = arrayListOf(1, 2, 3, 1)
val myList = listOf(1, 2, 3)

myArrayList.removeAll(myList)
println(myArrayList) // []
```

Seperti yang kita lihat dari contoh di atas, metode `removeAll()` juga akan menghapus semua duplikat elemen yang ada di daftar.

Metode `retainAll()` adalah kebalikan dari metode `removeAll()`. Metode ini akan menghapus semua elemen yang tidak ada dalam daftar:

```
val myList = arrayListOf(1, 2, 3, 1)  
val myList = listOf(2, 3)  
  
myArrayList.retainAll(myList)  
println(myArrayList) // [2, 3]
```

Metode `removeLast()` memungkinkan Anda untuk menghapus elemen pada indeks terakhir dari variabel `ArrayList`. Lihatlah contoh berikut:

E. Companion Object

Di Kotlin, jika ingin menulis sebuah fungsi atau member variabel di suatu kelas agar bisa dipanggil tanpa melalui sebuah objek, kita dapat melakukannya dengan menulis member atau method tersebut di dalam companion object suatu kelas. Jadi, dengan mendeklarasikan companion object, kita bisa mengakses member dari suatu kelas tanpa melalui objek.

Menulis sebuah companion object dapat dilakukan dengan menggunakan keyword `companion object` diikuti dengan namanya.

```
class CompanionClass {  
  
    companion object CompanionObject {  
  
    }  
}  
val obj = CompanionClass.CompanionObject
```

Kita tidak wajib memberikan nama bagi sebuah companion object sehingga penulisan di atas bisa disederhanakan menjadi:

```
class CompanionClass {  
  
    companion object {  
  
    }  
}  
val obj = CompanionClass.Companion
```

Untuk memanggil method atau member variabel yang dideklarasikan di dalam companion object adalah sebagai berikut:

```
class ToBeCalled {  
    companion object Test {  
        fun callMe() = println("You are calling me :)")  
    }  
}  
fun main(args: Array<String>) {  
    ToBeCalled.callMe()  
}
```

Output dari kode di atas adalah “You are calling me :)”.

F. Constructor Kotlin

Constructor pada kotlin berguna untuk membuat objek. Constructor mirip dengan metode deklarasi, dan mirip dengan class yang tidak mengembalikan apapun.

Kotlin memiliki dua jenis constructor, pertama adalah primary constructor atau constructor primer dan yang kedua adalah secondary constructor atau constructor sekunder. Satu class Kotlin dapat memiliki satu constructor utama, dan satu atau beberapa constructor sekunder. Constructor Java menginisialisasi variabel anggota, namun di Kotlin constructor primer menginisialisasi class, sedangkan constructor sekunder membantu memasukkan beberapa logika tambahan sambil menginisialisasi yang sama. Constructor utama dapat dideklarasikan pada level header class seperti contoh berikut.

```
class Buku(val namaBuku: String, var authorBuku: String) {  
    // class body  
}
```

Dalam contoh di atas, kita telah mendeklarasikan Constructor utama di dalam tanda kurung. Di antara dua kolom tersebut, nama depan bersifat “read only” atau hanya untuk dibaca karena dinyatakan sebagai “val”, sedangkan usia kolom dapat diedit. Dalam contoh berikut, kita akan menggunakan constructor utama.

```
fun main(args: Array<String>) {
    val buku1 = Buku("Belajar Kotlin", "Anonim")
    println("Nama Buku = ${buku1.namaBuku}")
    println("Author Buku = ${buku1.authorBuku}")
}
class Buku(val namaBuku: String, var authorBuku: String) {
```

Maka kode di atas secara otomatis akan menginisialisasi dua variabel dan menghasilkan output :

Nama Buku = Belajar Kotlin di Codekey

Author Buku = Anonim

Seperi yang sudah disebutkan sebelumnya, Kotlin memungkinkan kita untuk membuat satu atau lebih constructor sekunder untuk class kita. Constructor sekunder ini dibuat menggunakan kata kunci “constructor”. Hal ini diperlukan setiap kali kita ingin membuat lebih dari satu constructor di Kotlin atau kapan pun kita ingin menyertakan lebih banyak logika dalam constructor utama dan kita tidak dapat melakukannya karena constructor utama mungkin dipanggil oleh class lain. Berikut kita tampilkan contoh untuk kita, di mana kita telah membuat constructor sekunder dan menggunakan contoh di atas untuk mengimplementasikan hal yang sama.

```
fun main(args: Array<String>) {
    val Siswa = Siswa("Andi", 13)
    print("${Siswa.class}"+"-${Siswa.nama}"+"-${Siswa.umur}")
}
class Siswa(val nama: String, var usia: Int) {
    val class:String = "SMP class 1"
    constructor(nama : String , usia :Int , class :String):this(nama,usia) {
    }
}
```

Hal yang perlu diingat, sejumlah constructor sekunder dapat dibuat, namun, semua constructor tersebut harus memanggil constructor primer secara langsung atau tidak langsung. Kemudian, kode di atas akan menghasilkan output sebagai berikut:

G. Class and Object in Kotlin

Kotlin mendukung pemrograman berorientasi objek (OOP) serta pemrograman fungsional. Pemrograman berorientasi objek didasarkan pada objek (Object) dan kelas (class) waktunya. Kotlin juga mendukung pilar bahasa OOP seperti enkapsulasi, *Inheritance*, dan polimorfisme.

Class

Class pada Kotlin sedikit mirip dengan class pada Java. Class disini adalah cetak biru untuk objek yang memiliki properti umum. Class Kotlin dideklarasikan menggunakan kata kunci class (Class keyword). Class Kotlin memiliki *Header class* yang menentukan parameter tipenya, konstruktor, dll. Dan badan kelas yang dikelilingi oleh tanda kurung kurawal. Contoh sintaks deklarasi Class Kotlin :

```
class namaClass{ // class header  
    // property  
    // member function  
}
```

Contoh Class pada Kotlin :

```
class Account {  
  
    var acc_no: Int = 0  
    var name: String? = null  
    var amount: Float = 0f  
  
    fun deposit() {  
        //deposite code  
    }  
  
    fun checkBalance() {  
        //balance check code  
    }  
}
```

Kelas akun (account) memiliki tiga properti acc_no, nama, jumlah dan dua fungsi anggota yakni deposit () dan checkBalance (). Di Kotlin, properti harus diinisialisasi atau dideklarasikan sebagai abstrak. Di kelas di atas, properti acc_no diinisialisasi sebagai 0, nama sebagai null dan jumlah 0f.

Object

Object (Objek) adalah entitas waktu nyata atau mungkin merupakan entitas logis yang memiliki status dan perilaku. Ia memiliki karakteristik:

- *State*: mewakili nilai suatu objek.
- *Behaviour*: mewakili fungsionalitas suatu objek.

Objek digunakan untuk mengakses properti dan fungsi anggota class. Kotlin memungkinkan untuk membuat banyak objek dari sebuah class

Cara membuat Object

Object di dalam kotlin bisa dibuat dalam 2 langkah. Pertama yaitu membuat referensi (reference) dan kemudian membuat objeknya :

```
var obj1 = className()  
  
var obj2 = className()
```

Di sini obj1 dan obj2 adalah referensi dan className () adalah objek.

Cara Mengakses Class Property dan Member Function

Property (Properti) dan Member Function (fungsi anggota) class diakses oleh operator menggunakan objek. Sebagai contoh:

```
obj.deopsit()  
obj.name = Ajay
```

Mari kita buat contoh yang mana akan mengakses keduanya dengan menggunakan operator “.”.

```
class Account {  
  
    var acc_no: Int = 0  
    var name: String = ""  
    var amount: Float = 0.toFloat()  
  
    fun insert(ac: Int,n: String, am: Float ) {  
        acc_no=ac  
        name=n  
        amount=am  
        println("Account no: ${acc_no} holder :${name} amount :${amount}")  
    }  
  
    fun checkBalance() {  
        //balance check code  
    }  
}  
  
fun main(args: Array<String>){  
  
    Account()  
    var acc= Account()  
    acc.insert(832345,"Ankit",1000f) //accessing member function  
    println("${acc.name}") //accessing class property  
}
```

Kalau kita jalankan maka akan menghasilkan output berikut :

```
Account no: 832345 holder :Ankit amount :1000.0  
  
Ankit
```

H. Getter Setter Kotlin

Properti adalah bagian penting dari bahasa pemrograman apa pun. Di Kotlin, kita dapat mendefinisikan properti dengan cara yang sama seperti kita mendeklarasikan variabel lain.

Di Kotlin penginisialisasi properti getter dan setter adalah opsional. Kita juga dapat menghilangkan tipe properti, jika dapat disimpulkan dari penginisialisasi. Sintaks deklarasi properti hanya-baca atau tidak dapat diubah berbeda dari yang dapat diubah dalam dua cara:

- Variabel dimulai dengan val, bukan var.
- Variabel tidak membolehkan setter.

```
fun main(args : Array<String>) {  
    var x: Int = 0  
    val y: Int = 1  
    x = 2 // In can be assigned any number of times  
    y = 0 // It can not be assigned again  
}
```

Dalam kode di atas, kita mencoba untuk memberi nilai lagi ke 'y' tetapi menunjukkan error waktu kompilasi karena tidak dapat menerima perubahan.

Di Kotlin, setter digunakan untuk menyetel nilai variabel apa pun dan getter digunakan untuk mendapatkan nilainya. Getter dan Setter dihasilkan secara otomatis dalam kode. Mari kita tentukan properti 'nama', di kelas, 'Perusahaan'. Tipe data 'nama' adalah String dan kita akan menginisialisasinya dengan beberapa nilai default.

```
class Company {  
    var name: String = "Defaultvalue"  
}
```

Kode di atas sama dengan kode ini:

```
class Company {  
    var name: String = "defaultValue"  
        get() = field // getter  
        set(value) { field = value } // setter  
}
```

Kita membuat instance objek 'c' dari kelas 'Perusahaan [...]''. Saat kita menginisialisasi properti 'nama', properti tersebut diteruskan ke nilai parameter setter dan set 'bidang' ke nilai. Saat kita mencoba mengakses properti name dari objek, kita mendapatkan field karena kode ini `get() = field`. Kita bisa mendapatkan atau mengatur properti dari objek kelas menggunakan notasi `dot[.]`

```
val c = Company()  
c.name = "GeeksforGeeks" // access setter  
println(c.name) // access getter (Output: GeeksforGeeks)
```

Kita bisa membuat memodifikasi setter dan getter variabel pada Kotlin seperti contoh kode di bawah ini.

```

class Registration( email: String, pwd: String, age: Int , gender: Char) {

    var email_id: String = email
        // Custom Getter
        get() {
            return field.toLowerCase()
        }
    var password: String = pwd
        // Custom Setter
        set(value){
            field = if(value.length > 6) value else throw IllegalArgumentException("Passwords is too small")
        }

    var age: Int = age
        // Custom Setter
        set(value) {
            field = if(value > 18 ) value else throw IllegalArgumentException("Age must be 18+")
        }
    var gender : Char = gender
        // Custom Setter
        set (value){
            field = if(value == 'M') value else throw IllegalArgumentException("User should be male")
        }
}

fun main(args: Array<String>) {

    val geek = Registration("PRAVEENRUHIL1993@GMAIL.COM","Geeks@123",25,'M')

    println("${geek.email_id}")
    geek.email_id = "GEEKSFORGEEKS@CAREERS.ORG"
    println("${geek.email_id}")
    println("${geek.password}")
    println("${geek.age}")
    println("${geek.gender}")

    // throw IllegalArgumentException("Passwords is too small")
    geek.password = "abc"

    // throw IllegalArgumentException("Age should be 18+")
    geek.age= 5

    // throw IllegalArgumentException("User should be male")
    geek.gender = 'F'
}

```

Output:

```

praveenruhil1993@gmail.com
geeksforgeeks@careers.org
Geeks@123
25
M

```

I. Inheritance pada Kotlin

Inheritance adalah fitur penting dari bahasa pemrograman berorientasi objek. Inheritance memungkinkan Anda untuk mewarisi fitur kelas yang ada (basis atau kelas induk) ke kelas baru (kelas turunan atau

kelas anak). Kelas utama disebut kelas super (kelas induk) dan kelas yang mewarisi superclass disebut subclass (kelas anak). Subclass berisi fitur superclass dan juga fitur miliknya sendiri. Konsep inheritance diperbolehkan ketika dua atau lebih kelas memiliki properti yang sama. Ini memungkinkan penggunaan kembali kode. Kelas turunan hanya memiliki satu kelas dasar tetapi mungkin memiliki beberapa antarmuka sedangkan kelas dasar dapat memiliki satu atau lebih kelas turunan.

Segala sesuatu di Kotlin secara default adalah final, oleh karena itu, kita perlu menggunakan kata kunci “open” di depan deklarasi kelas agar dapat diizinkan untuk diturunkan. Berikut adalah contoh penerapan inheritance:

```
import java.util.Arrays

open class Ucapan {
    fun pesan () {
        print("Selamat Belajar")
    }
}
class Selamat: Ucapan(){ // pewarisan terjadi menggunakan konstruktor default
}

fun main() {
    var view = Selamat()
    view.pesan()
}
```

Potongan kode di atas akan menghasilkan output berikut.

```
Selamat Belajar
```

Lalu, bagaimana jika pada kasus tertentu kita ingin mengganti metode pesan () di kelas anak. Kemudian, kita perlu mempertimbangkan contoh berikut. di mana kita membuat dua kelas dan mengganti salah satu dari fungsinya ke dalam kelas anak.

```
import java.util.Arrays

open class Ucapan {
    open fun pesan () {
        print("Selamat Belajar")
    }
}
class Selamat: Ucapan() { // pewarisan terjadi menggunakan konstruktor default
    override fun pesan() {
        print("Belajar Kotlin Bersama Codekey")
    }
}
fun main() {
    var view = Selamat()
    view.pesan()
}
```

Potongan kode di atas akan memanggil metode inheritance atau pewarisan kelas anak dan akan menghasilkan keluaran berikut di browser. Seperti Java, Kotlin juga tidak mengizinkan banyak inheritance. Berikut adalah hasil dari kode di atas.

Belajar Kotlin Bersama Codekey

J. Nested dan Inner Class

Inner class atau yang biasanya disebut juga nested class merupakan class yang berada di dalam class, berikut ini adalah contohnya.

```
class Person {

    inner class PersonName {

    }

}
```

Bukankah kita juga bisa langsung menuliskan class dalam class tanpa menggunakan inner? Ya, memang bisa, akan tetapi jika kita tidak menuliskan inner kita tidak akan bisa mengakses semua member pada outer class atau luar kelas, contohnya seperti berikut ini :

```
class Person {  
    val name = "UdaCoding"  
  
    inner class PersonName {  
  
        fun callUda() = name //Disini variable name akan mendapati error  
    }  
}
```

K. Enum Pada Kotlin

Seperti bahasa pemrograman yang menerapkan object oriented programming lainnya, dalam bahasa Kotlin juga dikenal dengan istilah enumeration. Dalam pemrograman, terkadang muncul kebutuhan untuk mengetikkan sesuatu hanya untuk menambahkan nilai yang pasti, oleh karena itu konsep enum pun diperkenalkan dan digunakan di berbagai bahasa pemrograman termasuk Kotlin.

Kotlin enum berisikan daftar nama dari konstanta. Enumeration pada kotlin memiliki tipe khusus yang mengindikasikan sesuatu yang memiliki angka atau nilai yang mungkin. Tidak seperti Java, Kotlin enum sendiri berbentuk kelas.

Saat kita memodelkan data aplikasi, beberapa konsep akan memiliki serangkaian kemungkinan yang terbatas. Contohnya adalah aplikasi yang terdapat fitur dark mode. Dalam kasus seperti ini, kita mendefinisikan enum dengan dua hal yang telah ditentukan: LIGHT dan DARK. Oleh karena itu tidak akan ada hal lain yang mungkin muncul karena dua kasus tersebut saling terkait.

Ketika kita memiliki bidang kemungkinan yang terdefinisi dengan baik, enum adalah fitur yang sangat bagus untuk digunakan. Dengan menggunakan Enum, kode kita memiliki pemeriksaan waktu kompilasi yang mencegah pengembang menggunakan kasus yang tidak ditentukan. Kita tidak dapat mereferensikan mode aplikasi yang disebut COLORFUL jika satu-satunya opsi adalah LIGHT dan DARK.

Karakteristik yang sama membuat enum tidak cocok untuk kumpulan data variabel atau dinamis. Kita tidak dapat mendefinisikan enum saat

runtime. Namun di sisi lain, Enum juga lebih baik untuk digunakan sebagai tambahan kinerja, karena kita tidak perlu membandingkan nilai String.

Berikut ini adalah beberapa hal penting yang harus diketahui tentang Enum Kotlin:

- Konstanta enum bukan hanya kumpulan konstanta tetapi ia juga memiliki properti, metode, dan lain sebagainya.
- Masing-masing konstanta enum Kotlin bertindak sebagai instance terpisah dari kelas dan dipisahkan dengan koma.
- Enum meningkatkan keterbacaan kode dengan menetapkan nama yang telah ditentukan sebelumnya ke konstanta.
- Instance kelas enum tidak dapat dibuat menggunakan konstruktor.

Berikut ini adalah dasar penggunaan enum dari type-safe Enum:

```
enum class ArahAngin {  
    UTARA, SELATAN, BARAT, TIMUR  
}
```

Masing-masing konstanta enum adalah objek dan konstanta enum dipisahkan dengan koma. Kemudian, berikut ini adalah cara mendefinisikan enum Kotlin dengan menggunakan keyword enum di depan suatu kelas seperti ini:

```
enum class Hari{  
    MINGGU,  
    SENIN,  
    SELASA,  
    RABU,  
    KAMIS,  
    JUMAT,  
    SABTU  
}
```

Menginisialisasi Kotlin Enum

Pada bahasa pemrograman Kotlin, enum juga bisa didapatkan dengan konstruktor seperti Java Enum. Karena konstanta enum adalah turunan dari kelas Enum, konstanta dapat diinisialisasi dengan meneruskan nilai tertentu

ke konstruktor utama. Berikut adalah contoh untuk menentukan warna pada card atau kartu:

```
enum class Kartu(val color: String) {  
    Hearts("green"),  
    Spades("blue"),  
}
```

Kita bisa dengan mudah mengakses warna dari kartu dengan menggunakan kode berikut ini:

```
val view = Kartu.Spades.color
```

Properti Enum dan Metode

Seperti di Java dan bahasa pemrograman lainnya, kelas enum Kotlin memiliki beberapa properti dan fungsi bawaan yang dapat digunakan oleh programmer. Berikut ini adalah properti dan metode utama.

Properti Kelas Enum :

- ordinal: Properti ini menyimpan nilai ordinal dari konstanta, yang biasanya merupakan indeks berbasis nol.
- name: Properti ini menyimpan nama konstanta.

Metode Utama Kelas Enum :

- value : Metode ini mengembalikan daftar semua konstanta yang didefinisikan dalam kelas enum.
- valueOf: Metode ini mengembalikan konstanta enum yang ditentukan dalam enum, cocok dengan string input. Jika konstanta, tidak ada di enum, maka `IllegalArgumentException` dilemparkan.

References

<https://developer.android.com/studio/run/managing-avds?hl=id>

<https://developer.android.com/studio/debug?hl=id>

<https://perbedaan.budisma.net/perbedaan-array-dan-arraylist.html>

<https://www.geeksforgeeks.org/kotlin-list-arraylist/>

<https://sebhastian.com/arraylist-kotlin/>

<https://lobothijau.medium.com/companion-object-di-kotlin-8741ed0b1b74>

<https://codekey.id/kotlin/constructor-pada-kotlin/>

<https://www.techfor.id/class-dan-object-pada-bahasa-program-kotlin/>

<https://www.geeksforgeeks.org/kotlin-setters-and-getters/>

<https://codekey.id/kotlin/inheritance-kotlin-2/>

<http://www.barajacoding.or.id/mengenal-inner-class-pada-kotlin/>

<https://codekey.id/kotlin/kotlin-enum/>