LAPORAN RESMI
MODUL III
INTENTS, MENU DAN DIALOG
PEMROGRAMAN BERGERAK

NAMA	**:**	**SITI NUR ALFI HIDAYAH**
N.R.P	**:**	**200441100106**
DOSEN	**:**	**ACHMAD ZAIN NUR, S.KOM.,M.T.**
ASISTEN	**:**	**FAUZIAH REZA OKTAVIYANI**
TGL PRAKTIKUM	**:**	**8 MEI 2023**

Disetujui : …MEI 2023
Asisten

FAUZIAH REZA OKTAVIYANI
19.04.411.00075

LABORATORIUM BISNIS INTELIJEN SISTEM
PRODI SISTEM INFORMASI
JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNIK
UNIVERSITAS TRUNOJOYO MADURA

BAB I
PENDAHULUAN

1.1 Latar Belakang

Android Studio adalah sebuah Integrated Development Environment (IDE) yang digunakan untuk mengembangkan aplikasi Android. IDE ini dikembangkan oleh Google dan merupakan salah satu alat yang paling populer digunakan oleh para pengembang aplikasi Android.

Fragment adalah salah satu komponen penting dalam pengembangan aplikasi Android menggunakan Android Studio. Fragment dapat dianggap sebagai bagian dari antarmuka pengguna yang dapat ditempatkan di dalam aktivitas (Activity) untuk memungkinkan tampilan dan perilaku yang modular.

Sebelum diperkenalkannya Fragment, pengembang Android hanya dapat menggunakan Activity untuk mengelola antarmuka pengguna. Namun, dengan semakin kompleksnya aplikasi, penggunaan Activity saja menjadi kurang efisien dan sulit dikelola. Fragment diciptakan untuk mengatasi masalah ini dengan membagi tampilan dan logika menjadi beberapa bagian yang dapat digunakan kembali.

1.2 Tujuan

- Mahasiswa dapat membuat aplikasi dengan menggunakan fragment.
- Mahasiswa dapat membuat Aplikasi dengan database.

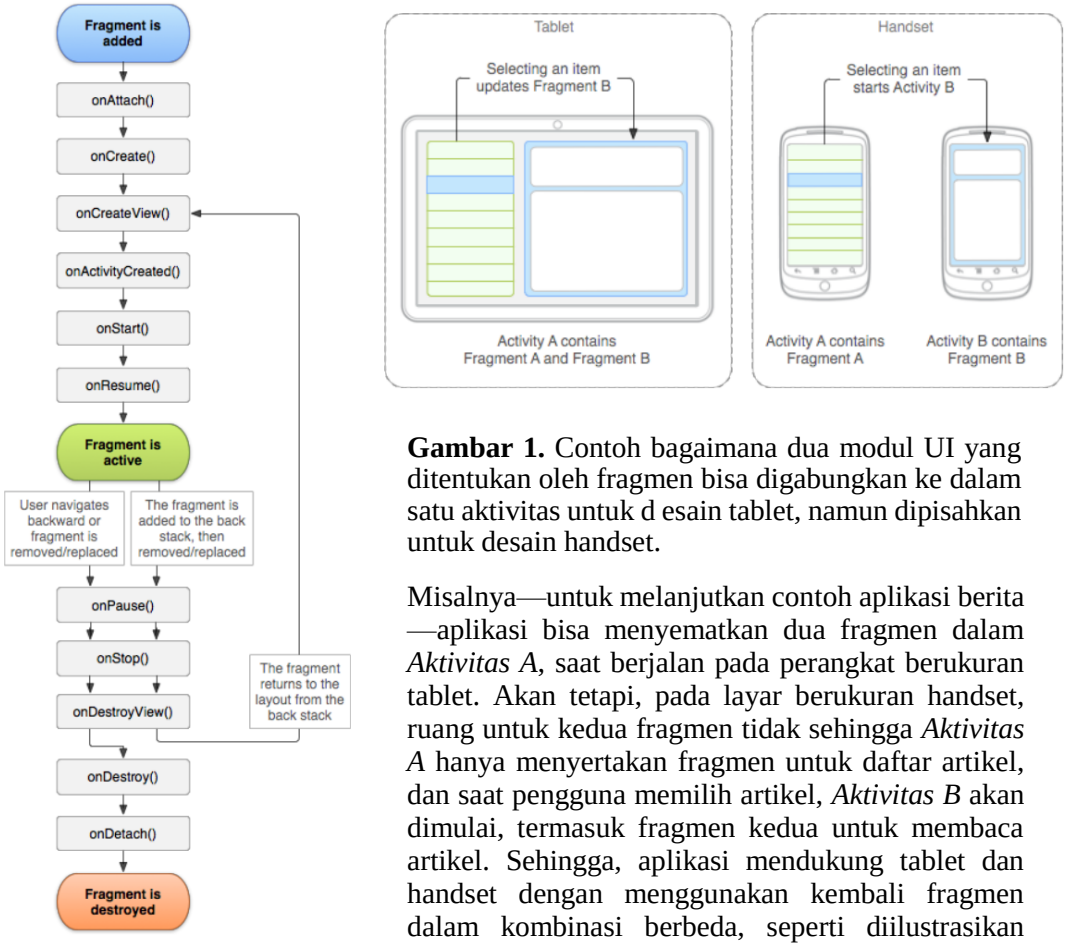
BAB II
DASAR TEORI

2.1 Fragment

Fragment mewakili perilaku atau bagian dari antarmuka pengguna dalam FragmentActivity. Kita bisa mengombinasikan beberapa fragmen dalam satu aktivitas untuk membangun UI multipanel dan menggunakan kembali sebuah fragmen dalam beberapa aktivitas. Kita bisa menganggap fragmen sebagai bagian modular dari aktivitas, yang memiliki daur hidup sendiri, menerima kejadian masukan sendiri, dan yang bisa kita tambahkan atau hapus saat aktivitas berjalan (semacam "subaktivitas" yang bisa digunakan kembali dalam aktivitas berbeda). Fragmen harus selalu tersemat dalam aktivitas dan daur hidup fragmen secara langsung dipengaruhi oleh daur hidup aktivitas host-nya. Misalnya, saat aktivitas dihentikan sementara, semua fragmen di dalamnya juga dihentikan sementara, dan bila aktivitas dimusnahkan, semua fragmen juga demikian. Akan tetapi, saat aktivitas berjalan (dalam status daur hidup dilanjutkan), Kita bisa memanipulasi setiap fragmen secara terpisah, seperti menambah atau membuangnya. Saat melakukan transaksi fragmen, Kita juga bisa menambahkannya ke back-stack yang dikelola oleh aktivitas—setiap entri backstack merupakan catatan transaksi fragmen yang terjadi. Dengan back-stack pengguna dapat membalikkan transaksi fragmen (mengarah mundur), dengan menekan tombol Kembali. Bila kita menambahkan fragmen sebagai bagian dari layout aktivitas, fragmen tersebut berada di ViewGroup di dalam hierarki tampilan aktivitas dan fragmen menentukan layout tampilannya sendiri. Kita bisa menyisipkan fragmen ke dalam layout aktivitas dengan mendeklarasikan fragmen dalam file layout aktivitas, sebagai <fragment>, atau dari kode aplikasi kita elemen menambahkannya ke ViewGroup yang ada.

Kita akan membahas cara membuat aplikasi menggunakan fragmen, termasuk cara fragmen mempertahankan statusnya bila ditambahkan ke back-stack aktivitas, berbagi kejadian dengan aktivitas, dan fragmen lain dalam aktivitas, berkontribusi pada bilah aksi aktivitas, dan lainnya. Filosofi Desain Android memperkenalkan fragmen di Android 3.0 (API level 11), terutama untuk mendukung desain UI yang lebih dinamis dan fleksibel pada layar besar, seperti

tablet. Karena layar tablet jauh lebih besar daripada layar handset, maka lebih banyak ruang untuk mengombinasikan dan bertukar komponen UI. Fragmen memungkinkan desain seperti itu tanpa perlu mengelola perubahan kompleks pada hierarki tampilan. Dengan membagi layout aktivitas menjadi beberapa fragmen, kita bisa mengubah penampilan aktivitas saat



Gambar 1. Contoh bagaimana dua modul UI yang ditentukan oleh fragmen bisa digabungkan ke dalam satu aktivitas untuk d esain tablet, namun dipisahkan untuk desain handset.

Misalnya—untuk melanjutkan contoh aplikasi berita—aplikasi bisa menyematkan dua fragmen dalam *Aktivitas A*, saat berjalan pada perangkat berukuran tablet. Akan tetapi, pada layar berukuran handset, ruang untuk kedua fragmen tidak sehingga *Aktivitas A* hanya menyertakan fragmen untuk daftar artikel, dan saat pengguna memilih artikel, *Aktivitas B* akan dimulai, termasuk fragmen kedua untuk membaca artikel. Sehingga, aplikasi mendukung tablet dan handset dengan menggunakan kembali fragmen dalam kombinasi berbeda, seperti diilustrasikan dalam gambar 1.

handset, kita bisa menggunakan kembali fragmen dalam konfigurasi layout yang berbeda untuk mengoptimalkan pengalaman pengguna berdasarkan ruang layar yang tersedia. Misalnya, pada handset, fragmen mungkin perlu dipisahkan untuk menyediakan UI panel tunggal bila lebih dari satu yang tidak cocok dalam aktivitas yang sama. Cukup,

Membuat Fragmen

Untuk membuat fragmen, kita harus membuat subclass Fragment (atau subclass-nya yang ada). Class Fragment memiliki kode yang mirip seperti Activity. Class ini memiliki metode

callback yang serupa dengan aktivitas, seperti onCreate(), onStart(), onPause(), dan onStop(). Sebenarnya, jika kita mengonversi aplikasi Android saat ini untuk menggunakan fragmen, kita mungkin cukup memindahkan kode dari metode callback aktivitas ke masing-masing metode callback fragmen. Biasanya, kita harus mengimplementasikan setidaknya metode daur hidup berikut ini:

onCreate() : Sistem akan memanggilnya saat membuat fragmen. Dalam implementasi, kita harus melakukan inisialisasi komponen penting dari fragmen yang ingin dipertahankan saat fragmen dihentikan sementara atau dihentikan, kemudian dilanjutkan.

onCreateView() : Sistem akan memanggilnya saat fragmen menggambar antarmuka penggunanya untuk yang pertama kali. Untuk menggambar UI fragmen, kita harus mengembalikan View dari metode ini yang menjadi root layout fragmen. Hasil yang dikembalikan bisa berupa null jika fragmen tidak menyediakan UI.

onPause() : Sistem akan memanggil metode ini sebagai indikasi pertama bahwa pengguna sedang meninggalkan fragmen kita (walau itu tidak selalu berarti fragmen sedang dimusnahkan). Di sinilah biasanya kita harus mengikat perubahan yang harus dipertahankan di luar sesi pengguna saat ini (karena pengguna mungkin tidak akan kembali).

Kebanyakan aplikasi harus mengimplementasikan setidaknya tiga metode ini untuk setiap fragmen, tetapi ada beberapa metode callback lain yang juga harus kita gunakan untuk menangani berbagai tahap daur hidup fragmen. Perhatikan bahwa kode yang mengimplementasikan aksi daur hidup dari komponen dependen harus ditempatkan di komponen itu sendiri, bukan dalam implementasi callback fragmen.

Menambahkan antarmuka pengguna

Fragmen biasanya digunakan sebagai bagian dari antarmuka pengguna aktivitas dan menyumbangkan layoutnya sendiri ke aktivitas. Untuk menyediakan layout fragmen, kita harus mengimplementasikan metode callback onCreateView(), yang dipanggil sistem Android bila tiba saatnya fragmen menggambar layoutnya. Implementasi kita atas metode ini harus mengembalikan View yang menjadi root layout fragmen. Untuk mengembalikan layout dari onCreateView(), Kita bisa memekarkannya dari resource layout yang

ditentukan di XML. Untuk membantu melakukannya, onCreateView() menyediakan objek LayoutInflater.

Misalnya, terdapat subclass Fragment yang memuat layout dari file example_fragment.xml:

```
class ExampleFragment : Fragment() {
override fun onCreateView( inflater:
LayoutInflater, container: ViewGroup?,
savedInstanceState: Bundle?
): View {
    // Inflate the layout for this fragment
    return inflater.inflate(R.layout.example_fragment, container, false) }
}
```

Parameter container yang diteruskan ke onCreateView() adalah induk ViewGroup (dari layout aktivitas) tempat layout fragmen akan disisipkan. Parameter savedInstanceState adalah Bundle yang menyediakan data tentang instance fragmen sebelumnya, jika fragmen dilanjutkan.

Metode inflate() mengambil tiga argumen:

1. ID sumber daya layout yang ingin dimekarkan.
2. ViewGroup akan menjadi induk dari layout yang dimekarkan. container perlu diteruskan agar sistem menerapkan parameter layout ke tampilan akar layout yang dimekarkan, yang ditetapkan dalam tampilan induk yang akan dituju.
3. Boolean yang menunjukkan apakah layout yang dimekarkan harus dilampirkan ke ViewGroup (parameter kedua) selama pemekaran. (Dalam hal ini, ini salah karena sistem sudah memasukkan layout yang dimekarkan ke dalam container—meneruskan true akan membuat tampilan grup berlebih dalam layout akhir.) Kita kini telah melihat cara membuat fragmen yang menyediakan layout. Berikutnya, kita perlu menambahkan fragmen ke aktivitas.

Menambahkan fragmen ke aktivitas

Biasanya, fragmen berkontribusi pada sebagian UI ke aktivitas host, yang disematkan sebagai bagian dari hierarki tampilan keseluruhan aktivitas. Ada dua cara untuk menambahkan fragmen ke layout aktivitas:

Deklarasikan fragmen dalam file layout aktivitas.

Dalam hal ini, Kita bisa menetapkan properti layout fragmen seakan-akan sebuah tampilan.

Misalnya, berikut ini adalah file layout untuk aktivitas dengan dua fragmen:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/an
droid"                    android:orientation="horizontal"
android:layout_width="match_parent"
android:layout_height="match_parent">
   <fragment android:name="com.example.news.ArticleListFragment"
android:id="@+id/list"          android:layout_weight="1"
android:layout_width="0dp"
         android:layout_height="match_parent" />
   <fragment android:name="com.example.news.ArticleReaderFragment"
android:id="@+id/viewer"          android:layout_weight="2"
android:layout_width="0dp"
android:layout_height="match_p
arent" /> </LinearLayout>
```

Atribut android:name dalam <fragment> menetapkan class Fragment untuk dibuat instance-nya dalam layout.

Saat sistem membuat layout aktivitas, sistem membuat instance setiap fragmen sebagaimana yang ditetapkan dalam layout dan memanggil metode onCreateView() masing-masing, untuk mengambil setiap fragmen. Sistem akan menyisipkan View yang dikembalikan oleh fragmen secara langsung, menggantikan elemen <fragment>. Atau, secara programatis tambahkan fragmen ke ViewGroup yang ada. Kapan saja saat aktivitas

berjalan, Kita bisa menambahkan fragmen ke layout aktivitas. Kita cukup menetapkan ViewGroup di tempat memasukkan fragmen.

Untuk membuat transaksi fragmen dalam aktivitas (seperti menambah, membuang, atau mengganti fragmen), kita harus menggunakan API dari FragmentTransaction. Kita bisa mengambil instance FragmentTransaction dari FragmentActivity seperti ini:

```
val fragmentManager = supportFragmentManager val
fragmentTransaction = fragmentManager.beginTransaction()
```

Selanjutnya kita bisa menambahkan fragmen menggunakan metode add(), dengan menetapkan fragmen yang akan ditambahkan dan tampilan tempat menyisipkannya. Sebagai contoh:

```
val fragment = ExampleFragment()
fragmentTransaction.add(R.id.fragment_container, fragment)
fragmentTransaction.commit()
```

Argumen pertama yang diteruskan ke add() adalah ViewGroup tempat fragmen harus dimasukkan, yang ditetapkan oleh ID resource, dan parameter kedua merupakan fragmen yang akan ditambahkan. Setelah membuat perubahan dengan FragmentTransaction, Kita harus memanggil commit() untuk menerapkan perubahan.

Mengelola Fragmen

Untuk mengelola fragmen dalam aktivitas, kita perlu menggunakan FragmentManager. Untuk mendapatkannya, panggil getSupportFragmentManager() dari aktivitas kita. Beberapa hal yang dapat Kita lakukan dengan FragmentManager antara lain:

1. Dapatkan fragmen yang ada di aktivitas dengan findFragmentById() (untuk fragmen yang menyediakan UI dalam layout aktivitas) atau findFragmentByTag() (untuk fragmen yang menyediakan atau tidak menyediakan UI).
2. Tarik fragmen dari back-stack, dengan popBackStack() (menyimulasikan perintah *Kembali* oleh pengguna).
3. Daftarkan listener untuk perubahan pada back-stack, dengan addOnBackStackChangedListener().

Melakukan Transaksi Fragmen

Fitur menarik terkait penggunaan fragmen di aktivitas adalah kemampuan menambah, membuang, mengganti, dan melakukan tindakan lain dengannya, sebagai respons atas interaksi pengguna. Setiap set perubahan yang kita lakukan untuk aktivitas disebut transaksi dan kita bisa melakukan transaksi menggunakan API di FragmentTransaction. Kita juga bisa menyimpan setiap transaksi ke back-stack yang dikelola aktivitas, sehingga pengguna bisa mengarah mundur melalui perubahan fragmen (mirip mengarah mundur melalui aktivitas).

Kita bisa memperoleh instance FragmentTransaction dari FragmentManager seperti ini:

```
val fragmentManager = supportFragmentManager val fragmentTransaction
```

= fragmentManager.beginTransaction()

Setiap transaksi merupakan serangkaian perubahan yang ingin dilakukan pada waktu yang sama. Kita bisa menyiapkan semua perubahan yang ingin dilakukan untuk transaksi mana saja menggunakan metode seperti add(), remove(), dan replace(). Kemudian, untuk menerapkan transaksi pada aktivitas, kita harus memanggil commit(). Akan tetapi, sebelum memanggil commit(), kita mungkin perlu memanggil addToBackStack(), untuk menambahkan transaksi ke back-stack transaksi fragmen. Back-stack ini dikelola oleh aktivitas dan memungkinkan pengguna kembali ke status fragmen sebelumnya, dengan menekan tombol *Kembali*. Misalnya, dengan cara ini kita bisa mengganti satu fragmen dengan yang fragmen lain, dan mempertahankan status sebelumnya di back-stack:

```
val newFragment = ExampleFragment() val transaction =
supportFragmentManager.beginTransaction()
transaction.replace(R.id.fragment_container, newFragment)
transaction.addToBackStack(null) transaction.commit()
```

findFragmentByTag(). Sebagai contoh:

```
val fragment =
supportFragmentManager.findFragmentById(R.id.example_fragment) as
ExampleFragment
```

Membuat callback kejadian pada aktivitas

Dalam beberapa kasus, kita mungkin perlu fragmen untuk membagikan kejadian atau data dengan aktivitas dan/atau fragmen lain yang di-host oleh aktivitas. Untuk membagikan data, buat ViewModel bersama, seperti diuraikan dalam Membagikan data antar bagian fragmen di panduan ViewModel. Jika harus menyebarkan kejadian yang tidak dapat ditangani dengan ViewModel, Kita dapat mendefinisikan antarmuka callback di dalam fragment dan mengharuskan kejadian host menerapkannya. Saat aktivitas menerima callback melalui antarmuka, aktivitas akan bisa berbagi informasi itu dengan fragmen lain dalam layout jika perlu. Misalnya, jika sebuah aplikasi berita memiliki dua fragmen dalam aktivitas—satu untuk menampilkan daftar artikel (fragmen A) dan satu lagi untuk menampilkan artikel (fragmen B)—maka fragmen A harus memberi tahu aktivitas bila item daftar dipilih sehingga aktivitas bisa memberi tahu fragmen B untuk menampilkan artikel. Dalam hal ini, antarmuka OnArticleSelectedListener dideklarasikan di dalam fragmen A:

```
public class FragmentA : ListFragment() {    ...
   // Container Activity must implement this interface    interface
OnArticleSelectedListener {       fun onArticleSelected(articleUri:
Uri)
   }
   ...
}
```

Selanjutnya aktivitas yang menjadi host fragmen akan mengimplementasikan antarmuka OnArticleSelectedListener dan menggantikan onArticleSelected() untuk memberi tahu fragmen B mengenai kejadian dari fragmen A. Untuk memastikan bahwa aktivitas host mengimplementasikan antarmuka ini, metode callback fragmen A onAttach() (yang dipanggil sistem saat menambahkan fragmen ke aktivitas) membuat instance OnArticleSelectedListener dengan membuat Activity yang diteruskan ke onAttach():

```
public class FragmentA : ListFragment() {

    var listener: OnArticleSelectedListener? = null
    ...
    override fun onAttach(context: Context) {
        super.onAttach(context)
        listener = context as? OnArticleSelectedListener if
        (listener == null) {
            throw ClassCastException("$context must implement
                                     OnArticleSelectedListener")
        }

    }
    ...
}
```

Jika aktivitas belum mengimplementasikan antarmuka , maka fragmen akan melontarkan ClassCastException. Jika berhasil, anggota mListener yang menyimpan referensi ke implementasi aktivitas OnArticleSelectedListener, sehingga fragmen A bisa berbagi kejadian dengan aktivitas, dengan memanggil metode yang didefinisikan oleh antarmuka OnArticleSelectedListener. Misalnya, jika fragmen A adalah ekstensi dari ListFragment, maka setiap kali pengguna mengklik item daftar, sistem akan memanggil onListItemClick() di fragmen, yang selanjutnya memanggil onArticleSelected() untuk berbagi kejadian dengan aktivitas:

```
public class FragmentA : ListFragment() {
   var listener: OnArticleSelectedListener? = null    ...
  override fun onListItemClick(l: ListView, v: View, position: Int,
                                                id: Long) {
     // Append the clicked item's row ID with the content provider Uri          val noteUri: Uri
=                    ContentUris.withAppendedId(ArticleColumns.CONTENT_URI, id) //
     Send the event and Uri to the host activity
listener?.onArticleSelected(noteUri)    }    ...
}
```

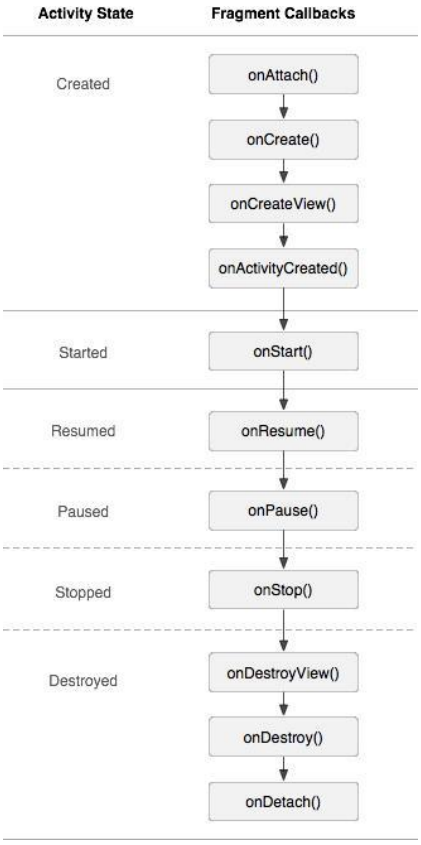
Parameter id yang diteruskan ke onListItemClick() merupakan ID baris dari item yang diklik, yang digunakan aktivitas (atau fragmen lain) untuk mengambil artikel dari ContentProvider aplikasi.

Menambahkan item ke Bilah Aplikasi

Fragmen kita bisa menyumbangkan item menu ke Menu Opsi aktivitas (dan, konsekuensinya, bilah aplikasi) dengan mengimplementasikan onCreateOptionsMenu(). Agar metode ini bisa menerima panggilan, Kita harus memanggil setHasOptionsMenu() selama onCreate(), untuk menunjukkan bahwa fragmen ingin menambahkan item ke Menu Opsi. Jika tidak, fragmen tidak menerima panggilan ke onCreateOptionsMenu(). Setiap item yang selanjutnya Kita tambahkan ke Menu Opsi dari fragmen akan ditambahkan ke item menu yang ada. Fragmen juga menerima callback ke onOptionsItemSelected() bila item menu dipilih. Kita juga bisa mendaftarkan tampilan dalam layout fragmen untuk menyediakan menu konteks dengan memanggil registerForContextMenu().

Bila pengguna membuka menu konteks, fragmen akan menerima panggilan ke onCreateContextMenu(). Bila pengguna memilih item, fragmen akan menerima panggilan ke onContextItemSelected().

Menangani Daur Hidup Fragmen



Gambar 3. Efek daur hidup aktivitas pada daur hidup fragmen.

Mengelola daur hidup fragmen mirip sekali dengan mengelola daur hidup aktivitas. Seperti aktivitas, fragmen bisa berada dalam tiga status:

Dilanjutkan

Fragmen terlihat dalam aktivitas yang berjalan.

Dihentikan sementara

Aktivitas lain berada di latar depan dan memiliki fokus, namun aktivitas tempat fragmen berada masih terlihat (aktivitas latar depan sebagian terlihat atau tidak menutupi seluruh layar).

Dihentikan

Fragment tidak terlihat. Aktivitas host telah dihentikan atau fragmen telah dihapus dari aktivitas namun ditambahkan ke back-stack. Fragmen yang dihentikan masih hidup (semua status dan informasi anggota masih disimpan oleh sistem). Akan tetapi, fragmen tidak terlihat lagi oleh pengguna dan akan dimatikan jika aktivitas dimatikan.

Seperti halnya aktivitas, kita dapat mempertahankan status UI fragment di seluruh

perubahan konfigurasi dan habisnya proses menggunakan kombinasi onSaveInstanceState(Bundle), ViewModel, serta penyimpanan lokal persisten. Perbedaan paling signifikan dalam daur hidup antara aktivitas dan fragmen ada pada cara penyimpanannya dalam back-stack masing-masing. Aktivitas ditempatkan ke dalam back-stack aktivitas yang dikelola oleh sistem saat dihentikan, secara default (sehingga pengguna bisa mengarah kembali ke aktivitas dengan tombol *Kembali*). Namun, fragmen ditempatkan ke dalam back-stack yang dikelola oleh aktivitas host hanya jika kita secara eksplisit meminta instance tersebut disimpan dengan memanggil addToBackStack() selama transaksi yang menghapus segmen tersebut. Jika tidak, pengelolaan daur hidup fragmen mirip sekali dengan mengelola daur hidup aktivitas; berlaku praktik yang sama.

Mengoordinasi dengan daur hidup aktivitas

Daur hidup aktivitas tempat fragmen berada akan memengaruhi secara langsung siklus hidup fragmen sedemikian rupa sehingga setiap callback daur

hidup aktivitas menghasilkan callback yang sama untuk masing-masing fragmen. Misalnya, bila aktivitas menerima

dalam aktivitas akanonPause(). Namun fragmen memiliki beberapa callback menerima daur hidup

ekstra, yang menangani interaksi unik dengan aktivitas untuk melakukan tindakan seperti membangun

onPause(), maka masing-masing fragmen dan memusnahkan UI fragmen. Metode callback tambahan ini adalah:

onAttach() Dipanggil bila fragmen telah dikaitkan dengan aktivitas (Activity diteruskan di sini). onCreateView() Dipanggil untuk membuat hierarki tampilan yang dikaitkan dengan

fragmen. onActivityCreated() Dipanggil bila metode onCreate() aktivitas telah dikembalikan.

onDestroyView() Dipanggil bila hierarki tampilan yang terkait dengan fragmen dihapus.

onDetach()Dipanggil bila fragmen diputuskan dari aktivitas.

Alur daur hidup fragmen, karena dipengaruhi oleh aktivitas host-nya, diilustrasikan oleh gambar 3. Dalam gambar tersebut, kita bisa melihat bagaimana setiap status aktivitas yang berurutan menentukan metode callback mana yang mungkin diterima fragmen. Misalnya, saat aktivitas menerima callback onCreate(), fragmen dalam aktivitas akan menerima tidak lebih dari callback onActivityCreated(). Setelah status aktivitas diteruskan kembali, Kita bisa bebas menambah dan membuang fragmen untuk aktivitas tersebut. Sehingga, hanya saat aktivitas berada dalam status dilanjutkan, daur hidup fragmen bisa berubah secara independen. Akan tetapi, saat aktivitas meninggalkan status dilanjutkan, fragmen akan kembali didorong melalui daur hidupnya oleh aktivitas.

BAB III
TUGAS PENDAHULUAN

3.1 Soal

1. Apa yang kamu ketahui tentang fragment dan apa bedanya dengan activity?
2. Sama dengan activity fragment juga memiliki siklus hidup (lifecycle) Jelaskan!
3. Apa saja keuntungan yang didapat jika menggunakan fragment?

3.2 Jawab

1. fragment merupakan bagian dari sebuah activity yang mana sebuah fragment tidak akan ada bila tidak ada sebuah activity karena fragment membutuhkan akses dari activity untuk dapat dijalankan. Perbedaan dengan activity adalah salah satu komponen yang ada diandroid studio yang berfungsi untuk menampilkan user interface (UI) dari aplikasi yang akan dibuat, biasanya diletakkan pada setContentView.
2. Tahapan life cycle
   * OnAttach() : dipanggil ketika fragment terhubung dengan activity yang dapat digunakan sebagai konteks
   * OnCreate() : dipanggil saat fragment dibuat
   * OnCreateView : dipanggil ketika tampilan layout untuk fragment dibuat
   * OnStart() : dipanggil ketika fragment mulai terlihat
   * OnStop() : dipanggil ketika fragment tidak terlihat
3. Kelebihan :
   * Tidak perlu memasukkan nama file fragment kedalam "AndroidManifest" yang diperlukan oleh activity.
   * fungsi yang berada pada activity dapat langsung digunakan dalam fragment tersebut tanpa harus membuat ulang.

BAB IV

IMPLEMENTASI

4.1 Soal

1. Buatlah sebuah aplikasi bebas dimana aplikasi tersebut memiliki 2 atau lebih fragment yang dapat berpindah satu sama lain dengan membawa data

4.2 Jawab

a. Source Code

MainActivity.kt

```
package com.example.myapplication

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)

        val fragmentManager = supportFragmentManager
        val firstFragment =  FirstFragment()
        val fragment =
fragmentManager.findFragmentByTag(FirstFragment::class.java.simple
Name)
        if(fragment !is FirstFragment){
            fragmentManager
                .beginTransaction()
                .add(R.id.frame_container, firstFragment,
FirstFragment::class.java.simpleName)
                .commit()

        }

    }

}
```

FirstFragment.kt

```
package com.example.myapplication

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button

// TODO: Rename parameter arguments, choose names that match
// the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
private const val ARG_PARAM1 = "param1"
private const val ARG_PARAM2 = "param2"

/**
```

```
 * A simple [Fragment] subclass.
 * Use the [FirstFragment.newInstance] factory method to
 * create an instance of this fragment.
 */
class FirstFragment : Fragment(), View.OnClickListener {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        arguments?.let {
            param1 = it.getString(ARG_PARAM1)
            param2 = it.getString(ARG_PARAM2)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_first,
container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val btnCategory: Button =
view.findViewById(R.id.nextButton)
        btnCategory.setOnClickListener(this)
    }

    companion object {

        // TODO: Rename and change types and number of parameters
        @JvmStatic
        fun newInstance(param1: String, param2: String) =
            FirstFragment().apply {
                arguments = Bundle().apply {
                    putString(ARG_PARAM1, param1)
                    putString(ARG_PARAM2, param2)
                }
            }
    }

    override fun onClick(p0: View) {
        if(p0.id == R.id.nextButton){
            val secondFragment = SecondFragment()
            val fragmentManager = parentFragmentManager
            fragmentManager
                .beginTransaction()
                .apply {
                    replace(R.id.frame_container, secondFragment,
SecondFragment::class.java.simpleName)
                    addToBackStack(null)
                    commit()
                }
```

```
        }
    }
}
```

SecondFregment.kt

```
package com.example.myapplication

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.EditText

// TODO: Rename parameter arguments, choose names that match
// the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
private const val ARG_PARAM1 = "param1"
private const val ARG_PARAM2 = "param2"


class SecondFragment : Fragment(){
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null

    private lateinit var searchInput: EditText

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        arguments?.let {
            param1 = it.getString(ARG_PARAM1)
            param2 = it.getString(ARG_PARAM2)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.second_fragment,
container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)

        searchInput = view.findViewById(R.id.inputText)
        val btnCategory: Button = view.findViewById(R.id.Button)
        btnCategory.setOnClickListener {
            val searchData = searchInput.text.toString()
            val bundle = Bundle()
            bundle.putString("inputData",searchData)

            val receiverFragment = ThirdFragment()
            receiverFragment.arguments = bundle
```

```
            parentFragmentManager.beginTransaction()
                .apply {
                    replace(R.id.frame_container,
receiverFragment)
                    addToBackStack(null)
                    commit()
                }
        }
    }

    companion object {

        // TODO: Rename and change types and number of parameters
        @JvmStatic
        fun newInstance(param1: String, param2: String) =
            SecondFragment().apply {
                arguments = Bundle().apply {
                    putString(ARG_PARAM1, param1)
                    putString(ARG_PARAM2, param2)
                }
            }
    }


}
```

ThirdFragment.kt

```
package com.example.myapplication

import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.Button
import android.widget.ImageButton
import android.widget.TextView


private const val ARG_PARAM1 = "param1"
private const val ARG_PARAM2 = "param2"

class ThirdFragment : Fragment(),View.OnClickListener {

    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        arguments?.let {
            param1 = it.getString(ARG_PARAM1)
            param2 = it.getString(ARG_PARAM2)
        }
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
```

```
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.third_fragment,
container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState:
Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val inputData: TextView = view.findViewById(R.id.value)
        val btnCategory: ImageButton =
view.findViewById(R.id.backButton)
        val searchData = arguments?.getString("inputData")


        inputData.text = searchData + " Not Found"


        btnCategory.setOnClickListener(this)
    }

    companion object {

        // TODO: Rename and change types and number of parameters
        @JvmStatic
        fun newInstance(param1: String, param2: String) =
            ThirdFragment().apply {
                arguments = Bundle().apply {
                    putString(ARG_PARAM1, param1)
                    putString(ARG_PARAM2, param2)
                }
            }
    }

    override fun onClick(p0: View) {
        if(p0.id == R.id.backButton){
            val secondFragment = SecondFragment()
            val fragmentManager = parentFragmentManager
            fragmentManager
                .beginTransaction()
                .apply {
                    replace(R.id.frame_container, secondFragment,
SecondFragment::class.java.simpleName)
                    addToBackStack(null)
                    commit()
                }
        }
    }
}
```

b. Hasil


11:25
YouTube
mahalini
SEARCH
mahalini
mahali l
mahali it


11:25
FragmentApp
mahalini Not Found

BAB V

PENUTUP

5.1 Analisa

Dari hasil praktikum, praktikan menyimpulkan bahwa konsep Fragment dan Room Database memiliki manfaat yang signifikan dalam pengembangan aplikasi Android. Penggunaannya dapat meningkatkan efisiensi dan kemudahan pembuatan aplikasi. Namun, perlu diingat bahwa kedua konsep tersebut memiliki batasan dan memerlukan pemahaman yang cukup tentang kode dan konsep yang digunakan. Oleh karena itu, keputusan untuk menggunakan Fragment dan Room Database harus didasarkan pada kebutuhan dan kompleksitas aplikasi yang dibuat. Sebuah aplikasi Android tidak dapat dibuat hanya dengan satu komponen saja. Oleh karena itu, penting untuk mempelajari cara mengintegrasikan beberapa komponen untuk membuat aplikasi yang fungsional. Proyek ini akan digunakan untuk mengintegrasikan berbagai komponen dan membentuk sebuah aplikasi yang fungsional.

5.2 Kesimpulan

1 Fragment adalah bagian UI aplikasi yang dapat digunakan unutk menentukan dan mengelola tata letaknya sendiri, memiliki siklus proses sendiri, serta dapat menangani peristiwa inputnya sendiri.

2 Fragmen tidak dapat berjalan sendiri. Fragmen harus dihosting oleh aktivitas atau fragmen lain.

3 Fragment merupakan salah satu komponen pada Android Studio dengan fungsi yang hampir sama seperti activity tetapi memiliki “lifecycle” yang berbeda.