LAPORAN RESMI
MODUL V
FRAGMENT

PEMROGRAMAN BERGERAK

NAMA	**:**	**SEVIN DIAS ANDIKA**
N.R.P	**:**	**210441100105**
DOSEN	**:**	**Ir. ACH. DAFID, S.T., M.T.**
ASISTEN	**:**	**DAVID NASRULLOH**
TGL PRAKTIKUM	**:**	**07 APRIL 2023**

Disetujui : .. APRIL 2022
Asisten

DAVID NASRULLOH
190441100060

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

BAB I
PENDAHULUAN

1.1 Latar Belakang

Fragman adalah bagian UI aplikasi Anda yang dapat digunakan kembali. Sebuah fragmen menentukan dan mengelola tata letaknya sendiri, memiliki siklus proses sendiri, serta dapat menangani peristiwa inputnya sendiri. Fragmen tidak dapat berjalan sendiri. Fragmen harus *dihosting* oleh aktivitas atau fragmen lain. Hierarki tampilan fragmen menjadi bagian dari, atau *dilampirkan ke*, hierarki tampilan host.

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.

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 masingmasing metode callback fragmen

1.2 Tujuan

- ➢ Mahasiswa dapat membuat aplikasi dengan menggunakan fragment.
- ➢ Mahasiswa dapat membuat Aplikasi dengan Fragment.

BAB II
DASAR TEORI

2.1 Dasar Teori

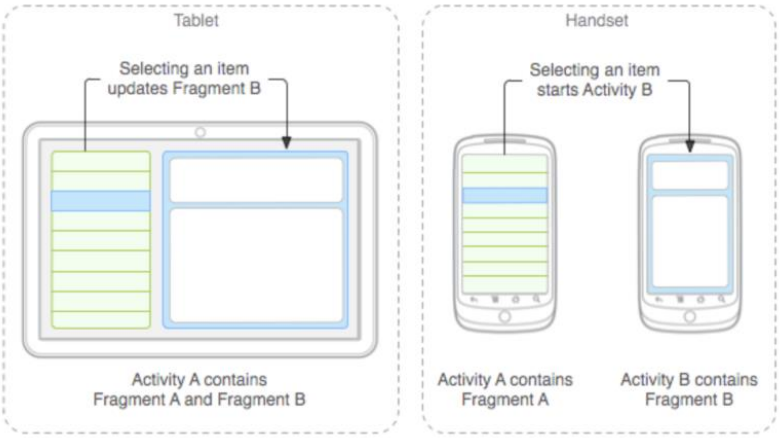
Pengertian Fragment

Android Fragment adalah bagian yang dapat digunakan kembali dari antarmuka pengguna android activity yang digunakan untuk membuat UI yang dinamis dan fleksibel. Fragmen memiliki siklus hidup itu sendiri tetapi selalu tertanam dengan aktivitas sehingga siklus hidup fragmen secara langsung dipengaruhi oleh siklus hidup aktivitas host dan fragmen yang diterimanya memiliki input peristiwa sendiri.

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 back-stack merupakan catatan transaksi fragmen yang terjadi. Dengan back-stack pengguna dapat membalikkan transaksi fragmen (mengarah mundur), dengan menekan tombol Kembali.

Di aplikasi Android kita dapat menggunakan beberapa fragmen dalam satu aktivitas untuk membuat UI Multi-Pane dan juga dapat menggunakan satu fragmen dalam beberapa aktivitas. Fragment Manager bertanggung jawab untuk menambah / menghapus atau mengganti fragmen pada waktu berjalan di mana pun aktivitas.

Bila Kita menambahkan fragmen sebagai bagian dari layout aktivitas, fragmen itu akan berada dalam ViewGroup di hierarki tampilan aktivitas tersebut dan fragmen mendefinisikan layout tampilannya sendiri. Kita bisa menyisipkan fragmen ke dalam layout aktivitas dengan mendeklarasikan fragmen dalam file layout aktivitas, sebagai elemen <fragment>, atau dari kode aplikasi dengan menambahkannya ke ViewGroup yang ada.



Pada gambar di atas, dicontohkan bagaimana dua modul UI yang didefinisikan oleh fragmen bisa digabungkan ke dalam satu activity untuk desain tablet namun dipisahkan untuk desain handset.

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.

Kita harus mendesain masing-masing fragmen sebagai komponen aktivitas modular dan bisa digunakan kembali. Yakni, karena setiap fragmen mendefinisikan layoutnya dan perilakunya dengan callback daur hidupnya sendiri, kita bisa memasukkan satu fragmen dalam banyak aktivitas, sehingga kita harus mendesainnya untuk digunakan kembali dan mencegah memanipulasi satu fragmen dari fragmen lain secara langsung. Ini terutama penting karena dengan fragmen modular kita bisa mengubah kombinasi fragmen untuk ukuran layar yang berbeda. Saat mendesain aplikasi untuk mendukung tablet maupun 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.

Membuat Fragmen

Untuk membuat fragment, kita membuat subkelas fragment (atau subkelas yang ada). kelas fragment memiliki kode yang mirip seperti Activity. Kelas ini memiliki metode callback yang serupa dengan activity seperti onCreate(), onStart(), onPause(), dan onStop().

- onAttach() metode ini panggil pertama kali bahkan sebelum onCreate() callback dan setelah fragmen telah dipasangkan ke activity
- 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 (UI)untuk yang pertama kali. Untuk menggambar UI fragmen, Anda harus mengembalikan View dari metode ini yang menjadi akar layout fragmen. Hasil yang dikembalikan bisa berupa null jika fragmen tidak menyediakan UI.
- onActivityCreated() Metode ini dipanggil setelah Activity onCreate() Callback telah menyelesaikan eksekusi. Metode ini merupakan indikasi untuk activity tersebut telah menyelesaikan eksekusi sebelum kita mencoba mengakses dan memodifikasi elemen UI dari activity secara bebas
- onStart() metode yang dipanggil setelah fragmen terlihat pada activity
- onResume() metode ini dipanggil ketika pengguna berinteraksi dengan fragmen dalam activity setelah Activity onResume() callback

Karena sebuah fragment tidak lagi digunakan, maka ia akan melewati serangkaian reverse callback

- onPause() metode ini dipanggil ketika fragmen tidak lagi berinteraksi dengan pengguna baik karena aktivitasnya sedang ditunda atau operasi fragmen mengubahnya dalam activity.

- onStop() metode ini dipanggil ketika fragmen tidak lagi berinteraksi dengan pengguna baik karena aktivitasnya dihentikan atau operasi fragmen mengubahnya dalam activity
- onDestroy() metode ini dipanggil untuk memungkinkan fragmen membersihkan resources yang terkait dengan view yang ada pada activity
- onDestroyView() metode ini dipanggil untuk melakukan pembersihan akhir dari status fragmen
- onDetach() metode ini dipanggil ke fragmen yang tidak lagi dikaitkan dengan aktivitasnya

Biasanya kita harus mengimplementasikan setidaknya metode alur onCreate(), onCreateView(), dan onPause().

- onSaveInstanceState() callback ini disebut dimana kita diizinkan untuk menyimpan beberapa data mengenai peristiwa fragmen tepat sebelum aplikasi di pause sehingga pengguna kembali ke aplikasi merekadengan mendapatkan data yang disimpan. Disini dibutuhkan Bundle sehingga kita dapat menyimpan data sebagai key atau nilai.

Kegunaan Fragment di Android

Sebelum Fragment diperkenalkan, kita hanya dapat menampilkan satu activity di layar pada satu waktu tertentu sehingga tidak dapat membagi layar dan mengontrol bagian yang berbeda secara terpisah. Dengan bantuan Fragment, kita dapat membagi layar di berbagai bagian dan mengontrol bagian-bagian yang berbeda secara terpisah.

Sebelum Fragment diperkenalkan, kita hanya dapat menampilkan satu activity di layar pada satu waktu tertentu sehingga tidak dapat membagi layar dan mengontrol bagian yang berbeda secara terpisah. Dengan bantuan Fragment, kita dapat membagi layar di berbagai bagian dan mengontrol bagian-bagian yang berbeda secara terpisah.

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/android"
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_parent" />

</LinearLayout>

Atribut android:name dalam 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 . 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:

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)    }    ...
}
```

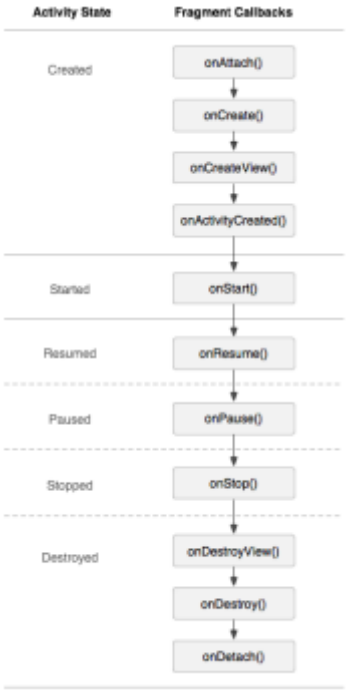
Parameter id yang diteruskan ke onListItemClick() merupakan ID baris dari item yang diklik, yang digunakan aktivitas (atau fragmen lain) untuk mengambil artikel dari ContentProvider aplikasi.

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 backstack 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 akan menerima onPause(). Namun fragmen memiliki beberapa callback 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 itu fragment?
2. Jelaskan dengan bahasamu sendiri cara membuat fragment di kotlin!
3. Jelaskan perbedaan antara fragment dan activity di kotlin!
4. Bagaimana cara berkomunikasi antara fragment dengan Actifity di Kotlin?
5. Sebutkan yang anda ketahui tentang Back Stack di fragment Kotlin!

3.2 Jawaban

1. Fragment adalah bagian dari ui apikasi anda yang dapat digunakan kembali

2. > Buatlah kelas kotlin baru dan tentukan nama kelas sebagai fragment anda, Seperti "My fragmet".
   > Pastikan bahwa kelas tersebut mewarisi kelas fragmet dari Andro id dengan menambahkan "fragmet" setelah nama kelas, Seperti berikut "class Myfragment : fragment()"
   > Tentukan tampilan yang ingin Anda buat menggunakan metode onCreateView().
   > Anda dapat menambahkan kode untuk mengakses tampilan didalam metode on View Create().
   > Anda dapat menggunakan fragment didalam Activity anda.

3. Fragment digunakan membuat bagian-bagian kecil dari antarmuka pengguna yang dapat digunakan kembali, Sedangkan Activity digunakan untuk membuat tampilan utama atau layout besar dalam aplikasi

4. > Menggunakan interface
   > Menggunakan ViewModel
   > Menggunakan SetResult

5. > Menggunakan fragment ke BackStack
   > Menghapus fragment dari BackStack
   > Mengambil nama fragment yang teratas di BackStack
   > Mencegah fragment tertentu dari dimasukan ke dalam BackStack.

BAB IV
IMPLEMENTASI

4.1 Tugas Praktikum

1. Buatlah Clone tampilan dari aplikasi yang sudah ada yang menerapkan tab layout. Contoh Whatsapp Messenger, google drive, dsb.

4.2 Source Code

1. Activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:orientation="vertical"
   tools:context=".MainActivity">
   <com.google.android.material.tabs.TabLayout
      android:id="@+id/tablayout"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      app:tabIndicatorColor="@color/purple_500"
      app:tabSelectedTextColor="@color/purple_700"
      app:tabTextAppearance="@style/TabLayoutTextStyle"
      app:tabTextColor="@color/black"/>
   <androidx.viewpager.widget.ViewPager
      android:layout_width="match_parent"
      android:layout_height="0dp"
      android:id="@+id/viewPager"
      android:background="@color/white"
      android:layout_weight="1"/>
</LinearLayout>
```

2. MainActivity.kt

```
 package com.example.fragment
import FragmentAdapter
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.viewpager.widget.ViewPager
import com.google.android.material.tabs.TabLayout
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      var viewPager = findViewById(R.id.viewPager) as ViewPager
      var tablayout = findViewById(R.id.tablayout) as TabLayout
      val fragmentAdapter = FragmentAdapter(supportFragmentManager)
      fragmentAdapter.addFragment(HomeFragment(),"Chat")
      fragmentAdapter.addFragment(ChatFragment(),"Status")
      fragmentAdapter.addFragment(SettingsFragment(),"Panggilan")
      viewPager.adapter = fragmentAdapter
      tablayout.setupWithViewPager(viewPager)
   }
}
```

3. Fragment_chat.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context=".ChatFragment">
   <!-- TODO: Update blank fragment layout -->
   <TextView
```

```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />
</FrameLayout>
```

4. chatFragment.kt

```
package com.example.fragment
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
class ChatFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_chat, container, false)
    }
}
```

5. fragment_home.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
```

```
    tools:context=".HomeFragment">
    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />
</FrameLayout>
```

6. homeFragment.kt

```
package com.example.fragment
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
class HomeFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_home, container, false)
    }
}
```

7. fragmentsetting.xml

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

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SettingsFragment">
    <!-- TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:text="@string/hello_blank_fragment" />
</FrameLayout>
```

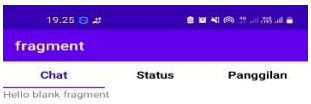
8. settingfragment.kt

```
package com.example.fragment
import android.os.Bundle
import androidx.fragment.app.Fragment
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
class SettingsFragment : Fragment() {
    // TODO: Rename and change types of parameters
    private var param1: String? = null
    private var param2: String? = null
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_settings, container, false)
    }
}
```

9. fragmentadapter.kt

```
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager
import androidx.fragment.app.FragmentStatePagerAdapter
class FragmentAdapter(fm : FragmentManager) :
FragmentStatePagerAdapter(fm,
   BEHAVIOR_RESUME_ONLY_CURRENT_FRAGMENT) {
   var fragmentList : ArrayList<Fragment> = ArrayList()
   var fragmenttitle : ArrayList<String> = ArrayList()
   override fun getCount(): Int {
      return fragmentList.size
   }
   override fun getItem(position: Int): Fragment {
      return fragmentList[position]
   }
   override fun getPageTitle(position: Int): CharSequence? {
      return fragmenttitle[position]
   }
   fun addFragment(fragment: Fragment,title : String){
      fragmentList.add(fragment)
      fragmenttitle.add(title)
   }
}
```

4.3 Hasil



BAB V
PENUTUP

5.1 Analisa

Dari hasil praktikum, praktikan menganalisa bahwa fragman adalah bagian UI aplikasi Anda yang dapat digunakan kembali. Sebuah fragmen menentukan dan mengelola tata letaknya sendiri, memiliki siklus proses sendiri, serta dapat menangani peristiwa inputnya sendiri. Fragmen tidak dapat berjalan sendiri. Fragmen harus *dihosting* oleh aktivitas atau fragmen lain. Hierarki tampilan fragmen menjadi bagian dari, atau *dilampirkan ke*, hierarki tampilan host.

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 masingmasing metode callback fragmen

5.2 Kesimpulan

1. Fragman adalah bagian UI aplikasi Anda yang dapat digunakan kembali.
2. 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.
3. onCreateView() : Sistem akan memanggilnya saat fragmen menggambar antarmuka penggunanya untuk yang pertama kali.
4. onPause() : Sistem akan memanggil metode ini sebagai indikasi pertama bahwa pengguna sedang meninggalkan fragmen kita (walau itu tidak selalu berarti fragmen sedang dimusnahkan).
5. Class Fragment memiliki kode yang mirip seperti Activity. Class ini memiliki metode callback yang serupa dengan aktivitas, seperti onCreate(), onStart(), onPause(), dan onStop().