Modul Fragment

Teori Tentang 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 back-stack 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 elemen <fragment>, atau dari kode aplikasi kita dengan menambahkannya ke ViewGroup yang ada.

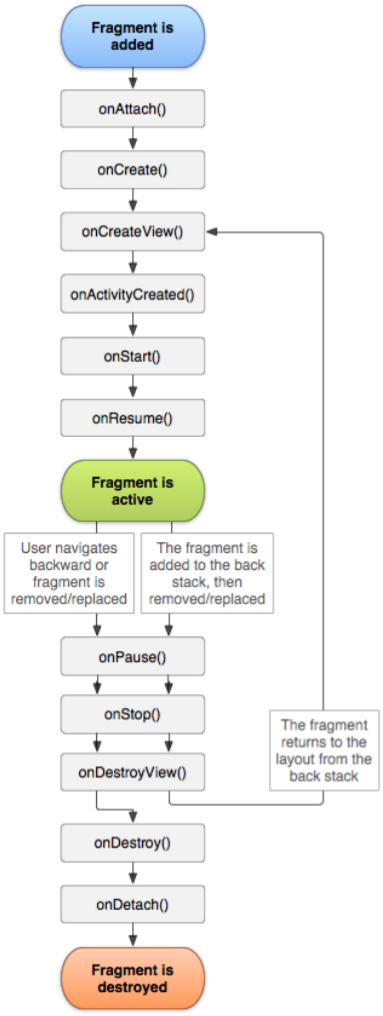
Kita akan membahas cara membuat aplikasi menggunakan fragmen, termasuk cara fragmen mempertahankan statusnya bila ditambahkan ke back-stack aktivitas, berbagi kejadian dengan aktivitas, dan fragmen lain dalam aktivitas, berkontribusi pada bilah aksi aktivitas, dan lainnya.

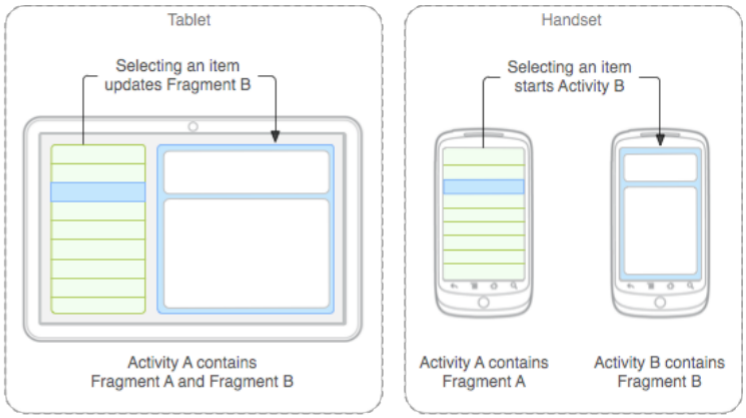
Android memperkenalkan fragmen di Android 3.0 (API level 11), terutama untuk mendukung desain UI yang lebih dinamis dan fleksibel pada layar besar, seperti tablet. Karena layar tablet jauh lebih besar daripada layar handset, maka lebih banyak ruang untuk mengombinasikan dan bertukar komponen UI. Fragmen 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 waktu proses dan mempertahankan perubahan itu di back-stack yang dikelola oleh aktivitas. Mode-mode tersebut kini tersedia secara luas melalui library dukungan fragmen.

Misalnya, aplikasi berita bisa menggunakan satu fragmen untuk menampilkan daftar artikel di sebelah kiri dan fragmen lainnya untuk menampilkan artikel di sebelah kanan—kedua fragmen ini muncul di satu aktivitas, berdampingan, dan masing-masing fragmen memiliki serangkaian metode callback daur hidup dan menangani kejadian masukan penggunanya sendiri.

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.





Gambar 1. Contoh bagaimana dua modul UI yang ditentukan oleh fragmen bisa digabungkan ke dalam satu aktivitas untuk desain 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 cukup, 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.

Gambar Daur hidup fragmen (saat aktivitasnya berjalan).

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 diperluas.
2. ViewGroup akan menjadi induk dari layout yang diperluas. container perlu diteruskan agar sistem menerapkan parameter layout ke tampilan akar layout yang diperluas, yang ditetapkan dalam tampilan induk yang akan dituju.

3. Boolean yang menunjukkan apakah layout yang diperluas harus dilampirkan ke ViewGroup (parameter kedua) selama pemekaran. (Dalam hal ini, ini salah karena sistem sudah memasukkan layout yang diperluas 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 <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 fragment.

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()
```

Dalam contoh ini, newFragment menggantikan fragmen apa saja (jika ada) yang saat ini berada dalam kontainer layout yang diidentifikasi melalui ID R.id.fragment_container. Dengan memanggil addToBackStack(), transaksi yang diganti disimpan ke back-stack sehingga pengguna bisa membalikkan transaksi dan mengembalikan fragmen sebelumnya dengan menekan tombol *Kembali*. FragmentActivity lalu secara otomatis mengambil fragmen dari back-stack melalui onBackPressed(). Jika kita menambahkan beberapa perubahan pada transaksi—seperti add() atau remove()—dan memanggil addToBackStack(), maka semua perubahan yang diterapkan sebelum Kita memanggil commit() akan ditambahkan ke back-stack sebagai transaksi tunggal dan tombol *Kembali* akan membalikkannya bersama-sama.

Urutan menambahkan perubahan pada FragmentTransaction tidak berpengaruh, kecuali:

1. Kita harus memanggil commit() paling akhir.
2. Jika Kita menambahkan beberapa fragmen ke container yang sama, maka urutan penambahannya akan menentukan urutan munculnya dalam hierarki tampilan.

Jika kita tidak memanggil addToBackStack() saat melakukan transaksi yang membuang fragmen, maka fragmen itu akan dimusnahkan bila transaksi diikat dan pengguna tidak bisa mengarah kembali
ke sana. Sedangkan, jika Kita memanggil addToBackStack() saat menghapus fragmen, maka fragmen itu akan *dihentikan* dan nanti dilanjutkan jika pengguna mengarah kembali.

Memanggil commit() tidak langsung menjalankan transaksi. Namun sebuah jadwal akan dibuat untuk dijalankan pada thread UI aktivitas (thread "utama") begitu thread bisa melakukannya. Akan tetapi, jika perlu Kita bisa memanggil executePendingTransactions() dari thread UI untuk segera mengeksekusi transaksi yang diserahkan oleh commit(). Hal itu biasanya tidak perlu kecuali jika transaksi merupakan dependensi bagi tugas dalam thread lain.

Berkomunikasi dengan Aktivitas

Meskipun Fragment diimplementasikan sebagai objek yang tidak bergantung pada FragmentActivity dan bisa digunakan dalam banyak aktivitas, instance tertentu dari fragmen secara langsung terkait dengan aktivitas yang menjadi hostnya. Khususnya, fragmen bisa mengakses instance FragmentActivity dengan getActivity() dan dengan mudah melakukan tugas-tugas seperti mencari tampilan dalam layout aktivitas:

Kotlin Java

val listView: View? = activity?.findViewById(R.id.list)

Demikian pula, aktivitas Kita bisa memanggil metode di fragmen dengan mendapatkan referensi ke Fragment dari menggunakan findFragmentById() atau 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:

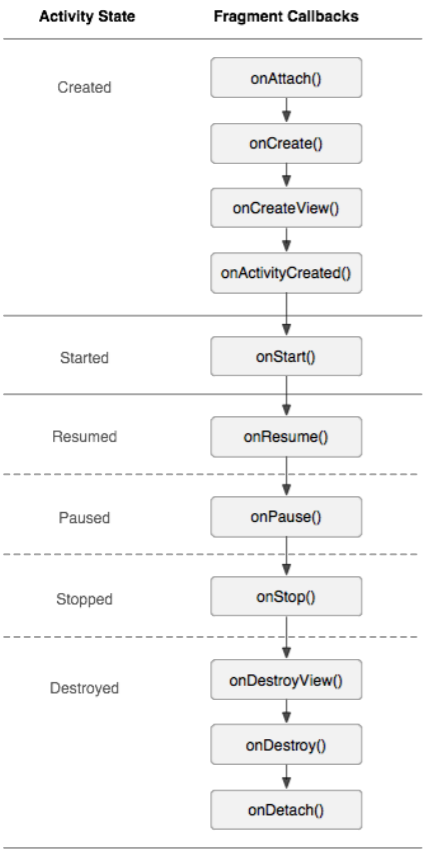
```
public class FragmentA : ListFragment() {

    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 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 onPause(), maka masing-masing fragmen dalam aktivitas akan menerima onPause(). Namun fragmen memiliki beberapa callback daur hidup ekstra, yang menangani interaksi unik dengan aktivitas untuk melakukan tindakan seperti membangun 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.

Latihan Praktikum

1. Kita akan membuat aplikasi yang menggunakan fragment dan komunikasi data dengan konsep ViewModel.
2. Buat project baru seperti biasanya, kemudian cek pada dependencies di build.gradle. Jika belum ada silakan ditambhakn implementation berikut. kemudian sinkron.

```
dependencies {

    implementation 'androidx.appcompat:appcompat:1.0.2'
    implementation
    'com.google.android.material:material:1.0.0'

    implementation 'androidx.lifecycle:lifecycle-
    extensions:2.0.0' implementation 'androidx.legacy:legacy-
    support-v4:1.0.0' implementation
    'androidx.constraintlayout:constraintlayout:1.1.3'
}
```

3. Kemudian, buka pada activity_mail.xml. Ubah kodenya sehingga menjadi seperti berikut

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.coordinatorlayout.widget.CoordinatorLayout
xmlns:android="http://schemas.android.com/apk/res/andro
id"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/too
    ls" android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <com.google.android.material.appbar.AppBarLay
        out android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme">

        <com.google.android.material.tabs.TabLayout
            android:id="@+id/tabs"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:background="?attr/colorPrimary"
            app:tabTextColor="@android:color/background_light" />
    </com.google.android.material.appbar.AppBarLayout>

    <androidx.viewpager.widget.ViewPager
        android:id="@+id/view_pager"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        app:layout_behavior="@string/appbar_scrolling_view_behavior"
        />
</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

4. Kemudian buatlah sebuah class untuk view model, dan tambahkan kode pada class seperti berikut.

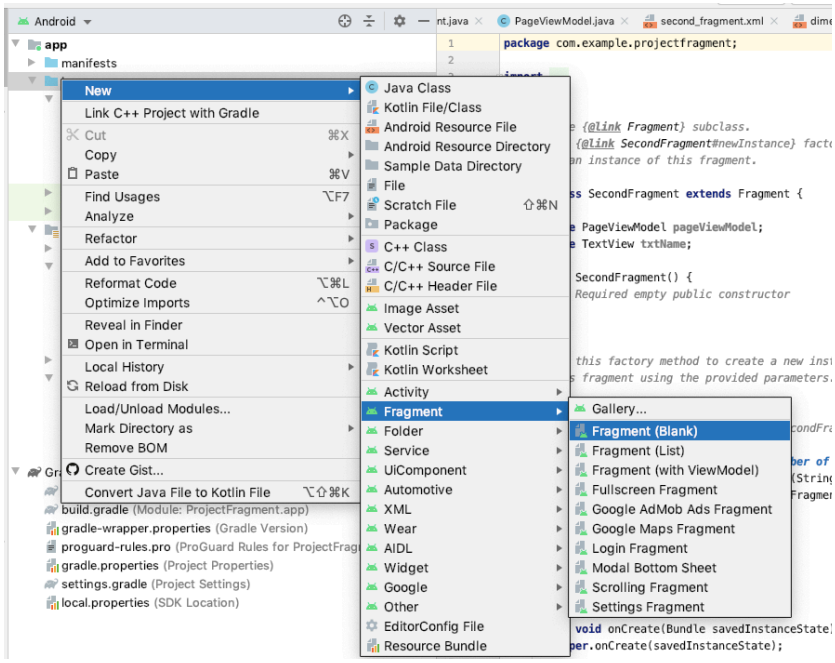
Kode kotlin

```
import androidx.lifecycle.LiveData
import androidx.lifecycle.MutableLiveData
import androidx.lifecycle.ViewModel

class CommunicationViewModel : ViewModel() {
    private val mName = MutableLiveData<String>()

    val name: LiveData<String>
        get() = mName

    fun setName(name: String) {
        mName.value = name
    }
}
```

Kode Java

```
import androidx.lifecycle.LiveData;
import androidx.lifecycle.MutableLiveData;
import androidx.lifecycle.ViewModel;

public class PageViewModel extends ViewModel {
    private MutableLiveData<String> mName = new MutableLiveData<>();

    public void setName(String name) {
        mName.setValue(name);
    }
    public LiveData<String> getName() {
        return mName;
    }
}
```

5. Selanjutnya buatlah sebuah fragment dengan nama fragment_first.xml.



```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent">

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="72dp"
        android:layout_height="72dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="24dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:src="@drawable/user_avatar"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/textInputLayout"
   style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginLeft="16dp"
        android:layout_marginTop="32dp"
        android:layout_marginEnd="16dp"
        android:layout_marginRight="16dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/textInputTextName"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="Masukkan Nama" />
    </com.google.android.material.textfield.TextInputLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

6. Kemudian tambahkan kode pada file FirstFragment

Kode kotlin

```
import android.os.Bundle
import android.text.Editable
import android.text.TextWatcher
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment
import androidx.lifecycle.ViewModelProviders
import com.google.android.material.textfield.TextInputEditText
```

```
class FirstFrament : Fragment() {
    private var communicationViewModel: CommunicationViewModel? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        communicationViewModel =
       ViewModelProviders.of(requireActivity()).
                    get(CommunicationViewModel::class.java)
    }

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

    override fun onViewCreated(view: View,
                savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        val nameEditText = view.findViewById<TextInputEditText>
               (R.id.textInputTextName)
        nameEditText.addTextChangedListener(
               object : TextWatcher {
            override fun beforeTextChanged(
               charSequence: CharSequence, i: Int, i1: Int, i2: Int) {
            }

            override fun onTextChanged(charSequence: CharSequence,
                      i: Int, i1: Int, i2: Int) {
              communicationViewModel!!.setName(charSequence.toString())
            }

            override fun afterTextChanged(editable: Editable) { }
        })
    }

    companion object {
        fun newInstance(): FirstFrament {
            return FirstFrament()
        }
    }
}
```

Kode java

```
import android.os.Bundle;
import android.text.Editable;
import android.text.TextWatcher;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.ViewModelProviders;
import com.fragmentcommunicationexample.R;
import com.google.android.material.textfield.TextInputEditText;
```

```
public class FirstFragment extends Fragment {
     private PageViewModel pageViewModel;
public FirstFragment() {
// Required empty public constructor
}
/**
* Create a new instance of this fragment
* @return A new instance of fragment FirstFragment.
*/
public static FirstFragment newInstance() {
     return new FirstFragment();
}

     @Override public void onCreate(@Nullable Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
// init ViewModel
     pageViewModel =
     ViewModelProviders.of(requireActivity()).get(PageViewModel.class);
}
     @Override
     public View onCreateView(LayoutInflater inflater, ViewGroup container,
     Bundle savedInstanceState) {

// Inflate the layout for this fragment
     return inflater.inflate(R.layout.fragment_first, container, false);
     }

     @Override public void onViewCreated(@NonNull View view, @Nullable
     Bundle savedInstanceState) {
     super.onViewCreated(view, savedInstanceState);

     TextInputEditText nameEditText =
     view.findViewById(R.id.textInputTextName);

     // Add Text Watcher on name input text
     nameEditText.addTextChangedListener(new TextWatcher() {

     @Override public void beforeTextChanged(CharSequence charSequence, int
     i, int i1, int i2) {
     }

     @Override public void onTextChanged(CharSequence charSequence, int i,
     int i1, int i2) {
     pageViewModel.setName(charSequence.toString());
     }

     @Override public void afterTextChanged(Editable editable) {
     }
     });
     }
}
```

7. Buat fragment kedua, beri nama fragment_second.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".SecondFragment">

    <ImageView
        android:id="@+id/imageView2"
        android:layout_width="72dp"
        android:layout_height="72dp"
        android:layout_marginStart="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginTop="24dp"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:src="@drawable/user_avatar"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <TextView
        android:id="@+id/textViewName"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:gravity="center"
        android:hint="User Display Name"
        android:textColor="@color/colorPrimaryDark"
        android:textSize="22sp"
        android:textStyle="bold"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        tools:text="Haloo Dunia!" />

    <TextView
        android:id="@+id/textView"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:text="Selamat datang"
        android:textAlignment="center"
        android:textSize="22sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView2" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

8. Kemudian tambahkan kode pada file SecondFragment

Kode kotlin

```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.TextView
import androidx.fragment.app.Fragment
import androidx.lifecycle.Observer
import androidx.lifecycle.ViewModelProviders

class SecondFragment : Fragment() {
    private var communicationViewModel: CommunicationViewModel? = null
    private var txtName: TextView? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        communicationViewModel = ViewModelProviders.
               of(requireActivity()).
               get(CommunicationViewModel::class.java)
    }

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(R.layout.fragment_second,
               container, false)
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState) txtName =
        view.findViewById(R.id.textViewName)
        communicationViewModel!!.name.observe(requireActivity(),
                       Observer { s -> txtName!!.text = s })
    }

    companion object {
        fun newInstance(): SecondFragment {
            return SecondFragment()
        }
    }
}
```

Kode java

```
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;
import androidx.annotation.NonNull;
import androidx.annotation.Nullable;
import androidx.fragment.app.Fragment;
import androidx.lifecycle.Observer;
import androidx.lifecycle.ViewModelProviders;
import com.fragmentcommunicationexample.R;

public class SecondFragment extends Fragment {
 private PageViewModel pageViewModel;
 private TextView txtName;
 public SecondFragment () {
// Required empty public constructor
}

/**
* Use this factory method to create a new instance of this fragment.
*
* @return A new instance of fragment SecondFragment.
*/
 public static SecondFragment newInstance() {
 return new SecondFragment();
}
        @Override
        public void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

    // initialise ViewModel here
        pageViewModel =
        ViewModelProviders.of(requireActivity()).get(PageViewModel.class);
        }

       @Override
       public View onCreateView(LayoutInflater inflater, ViewGroup container,
       Bundle savedInstanceState) {
  // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_second, container, false);
       }

       @Override
```

```
        public void onViewCreated(@NonNull View view, @Nullable Bundle
        savedInstanceState) {
        super.onViewCreated(view, savedInstanceState);
        txtName = view.findViewById(R.id.textViewName);
        pageViewModel.getName().observe(requireActivity(), new
        Observer<String>() {

        @Override
        public void onChanged(@Nullable String s) {
        txtName.setText(s);
        }
       });
     }
}
```

9. Buat sebuat adapter dengan nama View PagerAdapter

Kode Kotlin

```
import android.content.Context
import androidx.annotation.StringRes
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager import
androidx.fragment.app.FragmentPagerAdapter

class ViewPagerAdapter(private val mContext: Context, fm:
FragmentManager) :
     FragmentPagerAdapter(fm) {

     override fun getItem(position: Int): Fragment {
          return if (position == 0) {


               FirstFrament.newInstance()
          } else {
               SecondFragment.newInstance()
          }
     }

     override fun getPageTitle(position: Int): CharSequence? { return
          mContext.resources.getString(TAB_TITLES[position])
     }

     override fun getCount(): Int {
          return 2
     }

       companion object {
           @StringRes
          private val TAB_TITLES = intArrayOf(R.string.tab_text_1,
                 R.string.tab_text_2)
     }
}
```

Kode Java

```
import android.content.Context;
import androidx.annotation.Nullable;
import androidx.annotation.StringRes;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentManager;
import androidx.fragment.app.FragmentPagerAdapter;
import com.fragmentcommunicationexample.R;
/**
* A [FragmentPagerAdapter] that returns a fragment corresponding to
* one of the sections/tabs/pages.
*/

public class ViewPagerAdapter extends FragmentPagerAdapter {
@StringRes
  private static final int[] TAB_TITLES = new int[] { R.string.tab_text_1,
  R.string.tab_text_2 };
  private final Context mContext;

  public ViewPagerAdapter(Context context, FragmentManager fm) {
  super(fm);
  mContext = context;
  }

        @Override
        public Fragment getItem(int position) {
      // getItem is called to instantiate the fragment for the given page.
      if (position == 0) {
        return FirstFragment.newInstance();
      } else {
        return SecondFragment.newInstance();
        }
      }

        @Nullable
        @Override
        public CharSequence getPageTitle(int position) {
        return mContext.getResources().getString(TAB_TITLES[position]);
        }

        @Override
        public int getCount() {
        // Show 2 total pages.
        return 2;
        }
  }
```

10. Kemudian, buka **MainActivity** dan tuliskan kode program, sehingga menjadi seperti berikut

Kode kotlin

```
import androidx.appcompat.app.AppCompatActivity import
android.os.Bundle
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        view_pager.adapter = ViewPagerAdapter( this,
                        supportFragmentManager)
        tabs.setupWithViewPager(view_pager)
    }
}
```

Kode Java

```
import android.os.Bundle;
import androidx.appcompat.app.AppCompatActivity;
import androidx.viewpager.widget.ViewPager;
import com.fragmentcommunicationexample.ui.main.ViewPagerAdapter;
import com.google.android.material.tabs.TabLayout;

public class MainActivity extends AppCompatActivity {

  @Override
  protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      ViewPagerAdapter viewPagerAdapter =
      new ViewPagerAdapter(this, getSupportFragmentManager());
      ViewPager viewPager = findViewById(R.id.view_pager);
      viewPager.setAdapter(viewPagerAdapter);
      TabLayout tabs = findViewById(R.id.tabs);
      tabs.setupWithViewPager(viewPager);
  }
}
```

11. Jalankan dan lihat hasilnya.

Tugas Praktikum

Buat aplikasi baru dengan mengembangkan project diatas boleh dengan menambahkan isian dengan data pribadi (nama, nomor telpon, alamat, dll). Kemudian tambahkan splashscreen atau menggunakan swipe di awal aplikasinya. Desain silakan disesuaikan dengan selera masing-masing.

Referensi

1. https://androidwave.com/fragment-communication-using-viewmodel/
2. https://kotlinlang.org/docs/reference/
3. https://developer.android.com/kotlin
4. https://developer.android.com/courses/kotlin-android-fundamentals/toc
5. https://developer.android.com/kotlin/resources