MODUL PRAKTIKUM PEMROGRAMAN UNTUK PERANGKAT BERGERAK 1

Indra Azimi, S.T., M.T.

Reza Budiawan, S.T., M.T., OCA

Cahyana S.T., M.Kom.

D3 Rekayasa Perangkat Lunak Aplikasi
Telkom University

Daftar Isi

Modul 06: Activity & Fragment Lifecycle

1. Overview

Pada modul kali ini kita akan coba memahami siklus hidup (lifecycle) dari activity & fragment. Hal penting untuk diketahui, karena kita perlu mengetahui kapan komponen aplikasi seperti animasi, music, atau sensor harus dijalankan. Penempatan kode untuk menjalankan komponen tersebut bergantung pada method yang mewakili lifecycle dari activity/fragment.

Pada kasus lain, ketika terjadi perubahan orientasi layar (rotate screen), maka state dari aplikasi akan kembali seperti saat aplikasi pertama kali dijalankan. Bayangkan dalam aplikasi untuk simulasi ujian, yang sudah menampilkan nomor ke-10, dan penggunanya harus mengulang kembali mengerjakan dari nomor 1 ketika rotate screen terjadi secara tidak sengaja. Hal ini terjadi karena developer belum mengetahui sifat daur hidup dari activity/fragment aplikasi Android.

Materi pada modul ini diambil dari udacity.com (Developing Android Apps with Kotlin), lesson ke-4 tentang "Activity & Fragment Lifecycle". Hasil akhir dari aplikasi yang digunakan pada modul ini dapat diakses dari github udacity: https://github.com/udacity/andfun-kotlin-dessert-pusher/tree/Step.05-Solution-Implement-onSaveInstanceState.

2. Getting Started

Modul kali ini menggunakan starter code dari aplikasi DessertClicker. Silahkan download project Android Studio di https://github.com/google-developer-training/android-kotlin-fundamentals-starter-apps/tree/master/DessertClicker-Starter. Buka project ini di Android Studio masing-masing. Aplikasi DessertClicker memiliki 2 buah file kotlin, dan 1 buah layout. Kita akan mencoba memahami daur hidup Activity dan Fragment menggunakan project ini. Setelah memastikan aplikasi dapat berjalan dengan baik di smartphone/emulator, lakukan proses init git, hingga melakukan initial commit ke repo yang disediakan.

Dilarang keras untuk copy – paste kode dari modul/sumber lain!

Ngoding pelan-pelan akan membuat kamu lebih jago di masa depan.
Lakukan commit setiap selesai 1 sub-task. Selamat ngoding!

3. Task

Pada banyak tasks yang akan dikerjakan berikutnya, ada banyak langkah pembuatan class. Pastikan tidak menghapus kode package dari class yang terbentuk. Sama seperti class di bahasa pemrograman Java, kode package ini merupakan penanda tempat class tersebut berada.

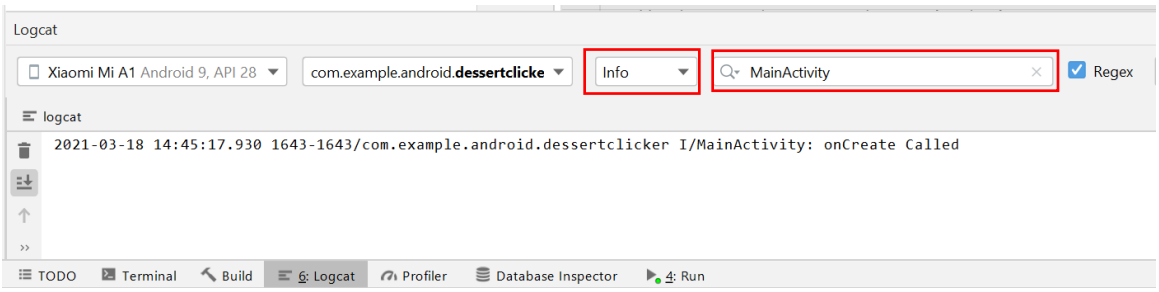
3.1. Mempelajari method onCreate dan Logging

Pada task ini, kita akan mempelajari daur hidup activity dengan mengamati cara kerja dari method onCreate dan logging. Logging merupakan proses memunculkan catatan/pesan pendek pada Logcat ketika aplikasi berjalan. Hal ini sudah kita lakukan beberapa kali pada modul sebelumnya dengan menggunakan keyword Log.d.

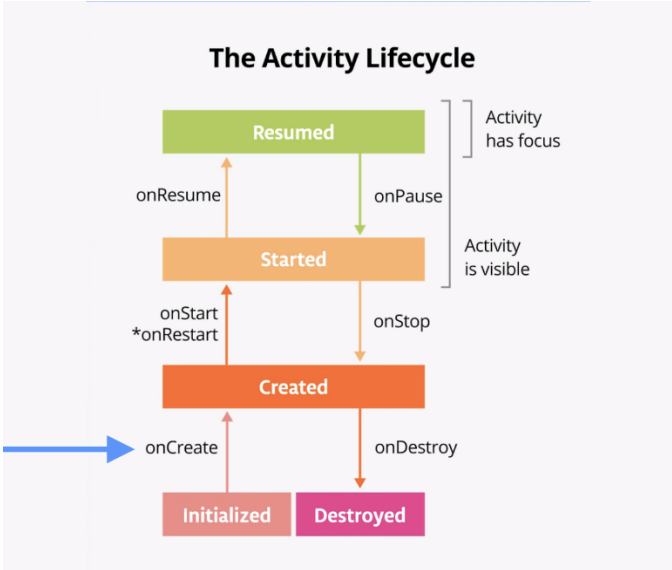
Langkah pertama yang perlu kita lakukan yaitu menambahkan kode Log pada method onCreate:

```
...
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    Log.i("MainActivity", "onCreate Called")

    // Use Data Binding to get reference to the views
    binding = DataBindingUtil.setContentView(this, R.layout.activity_main)
    ...
}
...
```

Kode di atas, kita menambahkan Log.i, dengan kata lain log yang ditambahkan bukan lagi untuk kepentingan debug (Log.d) tapi menampilkan informasi (Log.i). Jalankan project, dan perhatikan pada bagian Logcat akan muncul tulisan “onCreate Called”. Pastikan pada bagian type memilih “Info”, dan pada bagian fileter menuliskan tag yang sesuai, yaitu “MainActivity” (sesuai parameter pertama pada kode Log.i).



Informasi ini muncul ketika method onCreate dipanggil, sedangkan pemanggilan dari method ini berkaitan dengan daur hidup Activity. Daur hidup ini diperlihatkan pada gambar di bawah ini.

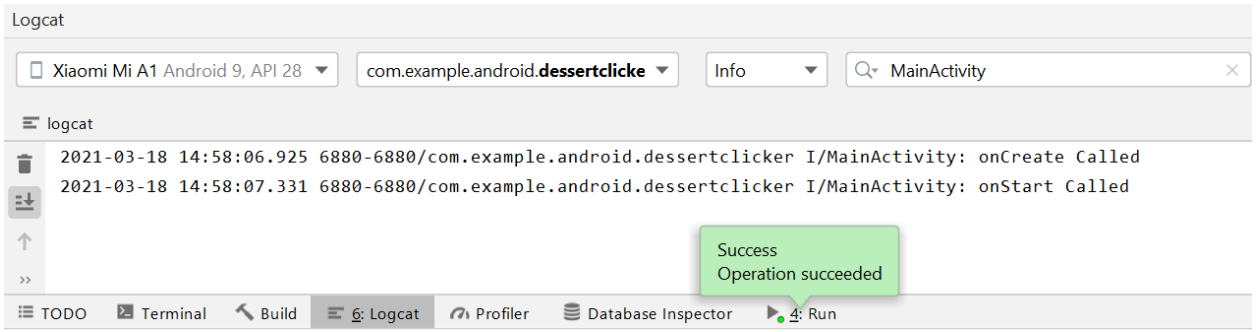


Dengan memperhatikan gambar di atas, dapat kita pahami bahwa tulisan “onCreate Called” pada log muncul karena method onCreate dieksekusi saat project di-run. Daur hidup ini terdiri dari stage initialized → created → started → resumed → started → created → destroyed. Setiap staging ini diwakili oleh method-nya masing-masing mulai dari onCreate hingga onDestroyed.

Secara default, saat project Android Studio pertama kali dibentuk, hanya method onCreate yang muncul, sedangkan method lainnya tidak tertulis pada MainActivity.kt. Method-method ini dapat kita tambahkan dengan cara melakukan penulisan method override. Berikutnya, kita akan membuat method onStart pada MainActivity.kt, dan menuliskan log “onStart Called” dan memastikan tulisan tersebut muncul pada Logcat. Caranya, tekan Ctrl+O pada keyboard, cari method onStart() dan klik Enter. Setelah method onStart terbentuk, tambahkan log dengan tulisan “onStart Called”. Kode akhir yang terbentuk pada MainActivity.kt untuk method onStart ditunjukkan pada kode di bawah ini.

```
...
override fun onStart() {
    super.onStart()
    Log.i("MainActivity", "onStart Called")
}
...
```

Setelah memastikan kode benar, jalankan kembali project, dan cek bagian Logcat.



Terlihat pada bagian Logcat muncul info “onStart Called” setelah “onCreate Called” yang berarti method onStart dijalankan setelah method onCreate sesuai dengan daur hidup dari Activity. Jika info bagian Logcat sudah ditampilkan sesuai contoh, silahkan lakukan commit sesuai judul dari task 3.1 ini.

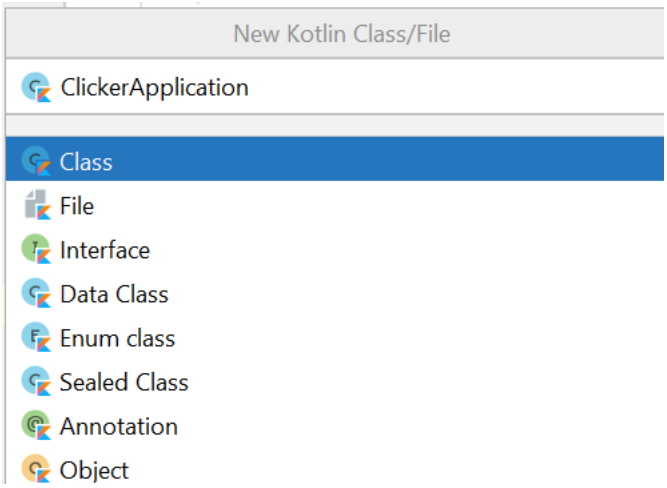
3.2. Menggunakan Timber untuk Logging

Pada task kali ini, kita akan mengganti cara untuk menuliskan log dari penulisan Log.i menjadi menggunakan library Timber. Dengan menggunakan library ini, kita tidak perlu lagi menuliskan “tag” lagi, karena Timber akan membuat tag sesuai nama class secara otomatis. Langkah pertama yang perlu dilakukan adalah menambahkan dependency pada app/build.gradle. Setelah menambahkan dependency, klik “Sync now” yang muncul di bagian kanan atas Android Studio.

```
dependencies {
    ...
    implementation 'com.jakewharton.timber:timber:4.7.1'
}
```

Library Timber menggunakan Application class untuk kepentingan inisialisasi pertama kalinya, dan penggunaannya yang bersifat global di seluruh project. Application class sendiri merupakan base class yang mengandung global application state untuk keseluruhan app. Class ini di-create sebelum Activity dibentuk dan dapat memyimpan global state. Sistem operasi android menggunakan class Application untuk berinteraksi dengan app yang kita buat.

Untuk membuat Application class ini, buat sebuah class dengan cara klik kanan pada package dessertclicker, pilih New → Kotlin Class/File. Berikan nama ClickerApplication, dan pilih tipe "Class".



Pada class ini, berikan sifat dari Application class.

```
class ClickerApplication : Application() {
}
```

Berikutnya, override method onCreate (Ctrl+O → cari method onCreate, tekan Enter). Dan tambahkan kode untuk menginisialisasi library Timber.

```
import android.app.Application
import timber.log.Timber

class ClickerApplication : Application() {
    override fun onCreate() {
        super.onCreate()
        Timber.plant(Timber.DebugTree())
    }
}
```

Selanjutnya, gunakan class ini pada manifest. Buka file manifest > AndroidManifest.xml, tambahkan atribut android:name pada tag application dengan value nama class Application yang sudah kita bentuk, yaitu ClickerApplication.

```
<manifest ...>
    <application
        android:name=".ClickerApplication"
        android:allowBackup="true"
        ...>
        ...
    </application>
</manifest>
```

Langkah berikutnya, pada MainActivity.kt, ganti kode Log.i dengan memanggil class Timber.

```
...
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    Log.i("MainActivity", "onCreate Called")
    Timber.i("onCreate called")
    ...
}

override fun onStart() {
    super.onStart()
    Log.i("MainActivity", "onStart Called")
    Timber.i("onStart Called")
}
...
```

Run project, dan pastikan aplikasi berjalan dengan baik, dan perhatikan bagian Logcat. Informasi yang ditampilkan sama seperti task 3.1, karena kita melakukan pengubahan cara menampilkan log dari penggunaan Log.i ke menggunakan library Timber.

Untuk lebih memahami alur siklus Activity, override semua method pada setiap stage, yaitu onResume; onPause; onStop; onDestroy; onRestart. Berikan log untuk semua method ini.

```
override fun onResume() {
    super.onResume()
    Timber.i("onResume Called")
}

override fun onPause() {
    super.onPause()
    Timber.i("onPause Called")
}
```

```
override fun onStop() {
    super.onStop()
    Timber.i("onStop Called")
}

override fun onDestroy() {
    super.onDestroy()
    Timber.i("onDestroy Called")
}


override fun onRestart() {
    super.onRestart()
    Timber.i("onRestart Called")
}
```

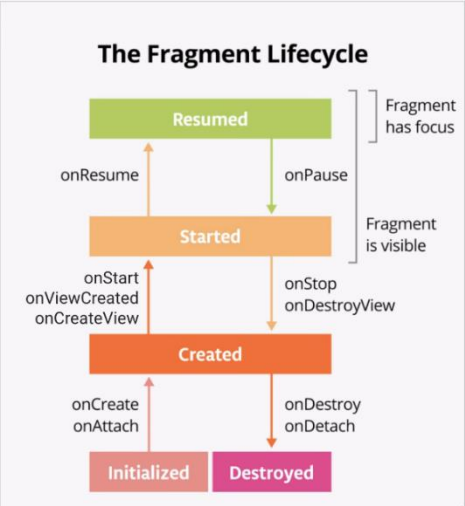
Setelah menambahkan method ini, run kembali project, lalu amati tulisan pada Logcat seiring melakukan beberapa hal pada smartphone, seperti keluar tanpa menutup app (dengan menekan home button), melakukan rotasi, menutup aplikasi, dan aktivitas lainnya. Cek tulisan yang muncul pada Logcat sekaligus diagram lifecycle dari Activity. Pastikan kamu mendapat pemahaman yang baik terkait hal ini.

Sebagai bantuan, kamu dapat membuat tabel pengamatan seperti contoh di bawah ini:

Aksi	Catatan	Simpulan
Pertama kali dijalankan	onCreate → onStart → onResume	App berjalan pada stage “Resumed”
Pertama kali dijalankan & dirotasi	onCreate → onStart → onResume → onPause → onStop → onDestroy → onCreate → onStart → onResume	App berjalan pada stage “Resumed”, ketika di-rotate app diteruskan lifecycle-nya sampai stage “Destroyed” dan di-create kembali sampai staged “Resumed”
Pertama kali dijalankan & ditutup dengan menekan home button	…	…
Pertama kali dijalankan dan ditutup secara normal	…	…

Aksi	Catatan	Simpulan
Pertama kali dijalankan dan menekan tombol share (menu pojok kanan atas)	…	…
… *<tuliskan kemungkinan/ skenario lain>*		

Setelah mengamati daur hidup Activity, kamu bisa bereksperimen menggunakan project modul minggu sebelumnya (modul 05) untuk mempelajari daur hidup Fragment.



Setelah meng-explore daur hidup Activity & Fragment, lakukan commit dengan commit message sesuai dengan judul task 3.2 ini.

3.3. Menggunakan Timer pada Lifecycle

Pada modul kali ini kita akan mempelajari jalannya animasi atau music pada sebuah aplikasi Android dengan sifat lifecycle dari Activity/Fragment. Hanya saja, kali ini kita menggunakan timer sebagai pengganti animasi atau music tersebut. Bayangkan timer ini terdapat aplikasi, dan kita perlu memahami behaviour-nya untuk dapat menerapkan control yang baik terhadap timer (animasi/music) tersebut.

Langkah awal pada task ini adalah membuat timer pada aplikasi Dessert Clicker. Kode dari timer tersebut telah dituliskan pada file DessertTimer.kt. Uncomment kode pada class ini, dengan cara block semua kode, dan tekan Ctrl+/ (tombol "Ctrl" dan tombol "/"). Class ini memiliki 2 method: startTimer dan stopTimer. Sesuai namanya, startTimer akan memulai timer, memunculkan waktu yang berjalan pada log setiap detiknya, dan stopTimer menghentikan timer tersebut.

```
class DessertTimer {

    var secondsCount = 0

    private var handler = Handler(Looper.getMainLooper())
    private lateinit var runnable: Runnable


    fun startTimer() {
        runnable = Runnable {
            secondsCount++
            Timber.i("Timer is at : $secondsCount")
            handler.postDelayed(runnable, 1000)
        }

        handler.postDelayed(runnable, 1000)
    }

    fun stopTimer() {
        handler.removeCallbacks(runnable)
    }
}
```

Gunakan method & class DessertTimer pada MainActivity.kt dengan menuliskan objeknya sebagai property class, dan memanggil konstruktornya pada method onCreate.

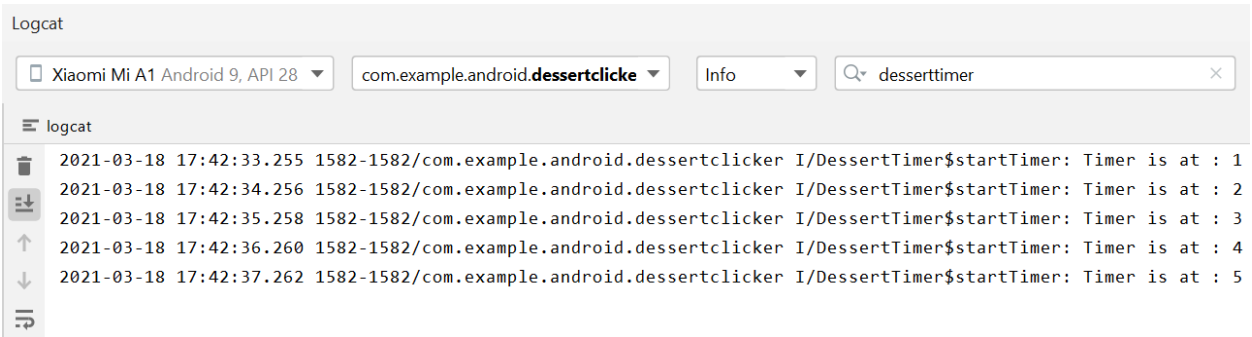
```
class MainActivity : AppCompatActivity() {

    ...
    private lateinit var binding: ActivityMainBinding
    private lateinit var dessertTimer: DessertTimer

    ...
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        binding.dessertButton.setOnClickListener {
            onDessertClicked()
        }
        dessertTimer = DessertTimer()
```

```
        ...
        binding.dessertButton.setImageResource(currentDessert.imageId)
    }

    ...
}
```

Berikutnya, kita harus memperkirakan di mana tepatnya pemanggilan startTimer & stopTimer diberikan. Bayangkan timer adalah animasi/music/sensor yang berjalan pada sebuah aplikasi. Animasi ini sebaiknya diputar ketika aplikasi sedang dalam kondisi on screen (terlihat oleh pengguna) dan dihentikan ketika aplikasi dalam kondisi off screen (tidak terlihat oleh pengguna). Dengan informasi ini, dan mengetahui sifat lifecycle dari Activity, kita dapat menaruh kode pemanggilan startTimer pada method onStart, dan pemanggilan stopTimer pada method onStop. Hal ini dikarenakan startTimer dipanggil ketika app memulai kondisi on screen dan onStop dipanggil ketika app dalam kondisi off screen.

```
...
override fun onStart() {
    super.onStart()
    Timber.i("onStart Called")
    dessertTimer.startTimer()
}

...

override fun onStop() {
    super.onStop()
    Timber.i("onStop Called")
    dessertTimer.stopTimer()
}
...
```

Run project, dan cek di bagian Log, akan terlihat timer yang berjalan.



Perhatikan behaviournya:

1. Ketika aplikasi dalam kondisi on screen → timer akan berjalan.
2. Ketika aplikasi berada dalam kondisi off screen (tekan home button) → maka timer berhenti; dan ketika membuka aplikasi kembali → timer melanjutkan perhitungannya.
3. Ketika aplikasi berada dalam kondisi off screen karena keluar app (tekan tombol back) → timer berhenti; dan ketika membuka aplikasi kembali → timer memulai perhitungan dari 1 kembali.

Behaviour di atas, didapatkan ketika kita menempatkan startTimer dan stopTimer dengan tepat. Coba tempatkan method startTimer pada onCreate, maka aplikasi tidak akan melanjutkan perhitungannya setelah app dalam kondisi off screen (karena menekan home button). Sedangkan penempatan stopTimer pada method yang salah akan berdampak pada tidak dihentikannya timer pada kondisi yang seharusnya.

Amati hal ini dengan baik sampai memahami sifat dari lifecycle Activity dan penggunaannya pada suatu hal yang sifatnya continue untuk dijalankan selama aplikasi berjalan. Jika sudah memahami hal ini, lakukan commit dengan commit message berupa judul dari task 3.3 ini.

3.4. Menerapkan Lifecycle Observer

Pada task sebelumnya, kita menuliskan kode startTimer dan stopTimer pada method onStart dan onStop. Dengan kata lain, Activity yang menentukan pemanggilan kedua method tersebut. Akan tetapi, ada pendekatan lain yang bisa dilakukan, dengan kondisi class DessertTimer yang akan memantau stage dari Activity. Jika Activity dalam stage create → start yang menjalan onStart, maka DessertTimer akan menjalankan method startTimer yang dia miliki, dan begitupula onStop.

Dengan kata lain, DessertTimer bertindak sebagai pemantau dari daur hidup Activity atau bahasa lainnya, lifecycle observer. Skenario ini dianggap membuat kode kita less error-prone. Hal ini akan kita terapkan pada aplikasi kita.

Langkah pertama untuk menerapkan lifecycle observer ini yaitu menambahkan sifat LifecycleObserver, dan memberikan referensi dari objek lifecycle pada class DessertTimer. Sehingga, pada DessertTimer.kt dapat dituliskan kode berikut.

```
class DessertTimer(lifecycle: Lifecycle) : LifecycleObserver {
    ...
}
```

Selanjutnya, tambahkan init block (kode yang pertama kali dijalankan saat objek class dibentuk), yang berisi kode untuk memberitahukan bahwa DessertTimer adalah observer dari objek referensi lifecycle.

```
class DessertTimer(lifecycle: Lifecycle) : LifecycleObserver {

    init{
        lifecycle.addObserver(this)
    }
}
```

Berikutnya, untuk memberitahukan bahwa method startTimer dijalankan saat onStart dan stopTimer dieksekusi saat onStop, berikan annotasi berikut pada masing-masing method:

```
...
@OnLifecycleEvent(Lifecycle.Event.ON_START)
fun startTimer() {
    ...
}


@OnLifecycleEvent(Lifecycle.Event.ON_STOP)
fun stopTimer() {
    ...
}
```

Class DessertTimer sekarang sudah bertindak sebagai Lifecycle Observer. Akan tetapi, perubahan kode DessertTimer yang membutuhkan 1 parameter, membuat kode MainActivity.kt error. Kita akan memperbaiki hal ini dengan menambahkan parameter lifecycle-nya. Pada MainActivity.kt, cari kode yang error, dan tambahkan objek lifecycle dari MainActivity.

```
class MainActivity : AppCompatActivity() {
    ...
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        ~~dessertTimer = DessertTimer()~~
        dessertTimer = DessertTimer(this.lifecycle)
```

```
        ...
    }
    ...
}
```

Berikutnya, karena DessertTimer sudah menjadi lifecycle observer, pemanggilan method stopTimer dan startTimer pada method onStart & onStop di MainActivity.kt tidak lagi diperlukan.

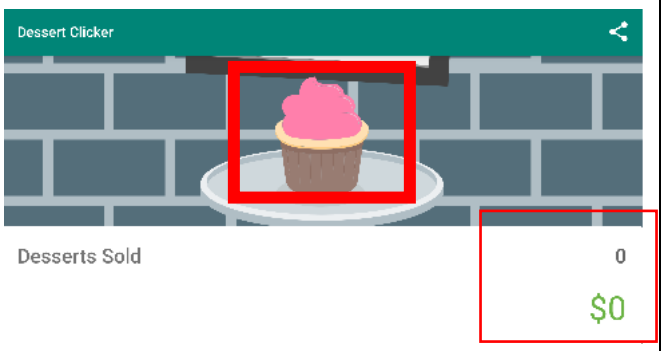
```
...
override fun onStart() {
    super.onStart()
    Timber.i("onStart Called")
    ~~dessertTimer.startTimer()~~
}

override fun onStop() {
    super.onStop()
    Timber.i("onStop Called")
    ~~dessertTimer.stopTimer()~~
}
...
```

Setelah memastikan tidak ada kode yang error, jalankan project. Pastikan timer tetap berjalan dengan behaviour yang sama dengan task 3.3. Jika sudah sesuai dan berjalan dengan baik di smartphone/emulator, lakukan commit dengan commit message sesuai dengan judul task 3.4 ini.

3.5. Mengatasi perubahan konfigurasi

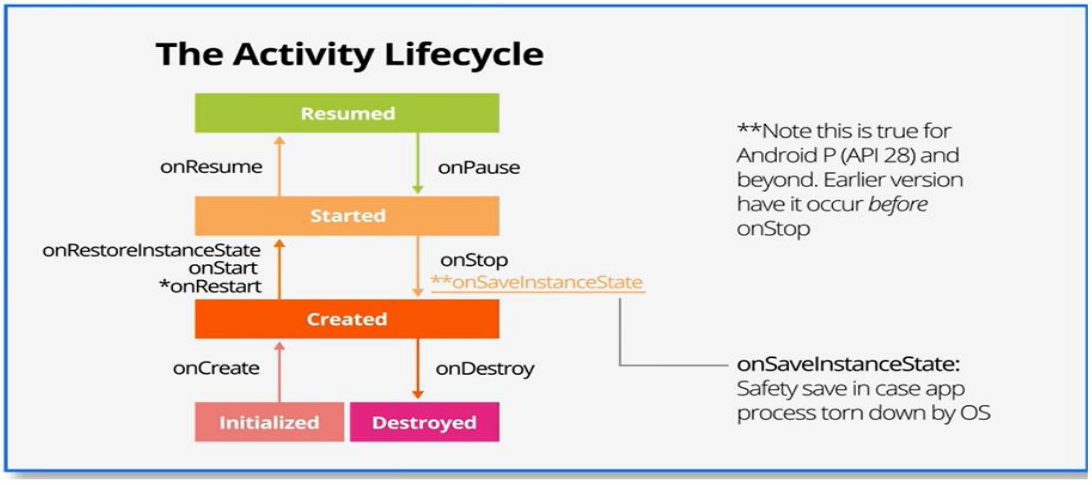
Pada task kali ini kita akan mengatasi perubahan konfigurasi yang menyebabkan hilangnya state dari aplikasi. Skenario permasalahan yaitu, lakukan klik beberapa kali pada gambar cupcake hingga gambarnya berubah dari cupcake menjadi kue lain.

Perhatikan bahwa nilai dessert sold (dari variable dessertSold) dan nilai harga (dari variable revenue) sudah berubah. Berikutnya, lakukan rotasi, dan perhatikan bahwa nilai pada dessert sold serta nilai harga kembali ke angka 0. Perubahan orientasi layar (rotated screen) membuat aplikasi kembali ke kondisi seperti baru dijalankan pertama kali.

Sebelum Rotasi	Setelah Rotasi
	

Jika telah meng-observe daur hidup pada task 3.2 dengan baik, kita akan mengetahui hal ini bisa saja terjadi. Karena perubahan rotasi akan membuat daur hidup masuk ke stage destroyed dan membentuk daur hidup baru. Hal ini wajar terjadi, akan tetapi dibutuhkan cara untuk menanganinya.

Penanganan untuk hal ini dilakukan dengan menggunakan method onSaveInstanceState. OnSaveInstanceState merupakan callback yang memungkinkan kita untuk menyimpan data ketika sistem operasi men-destroy aplikasi kita. Dengan memastikan ada data yang disimpan pada method ini, data tersebut dapat diambil kembali saat app di-create kembali. Data ini disimpan dalam bentuk Bundle. Method onSaveInstanceState ini dipanggil setelah method onStop dieksekusi.



Langkah pertama pada task ini untuk mempertahankan state aplikasi adalah melakukan override method onSaveInstanceState. Pada area kode MainActivity.kt, tekan Ctrl+O di keyboard, lalu cari method onSaveInstanceState dengan 1 parameter.

```
override fun onSaveInstanceState(outState: Bundle) {
    super.onSaveInstanceState(outState)
}
```

Berikutnya, simpan nilai revenue dan dessertSold. Penyimpanan dilakukan terhadap Bundle outstate dengan menggunakan key-value pair. Key ini merupakan referensi untuk melakukan load data yang disimpan. Sebaiknya gunakan constant untuk key agar penulisannya konsisten saat penyimpanan & pengambilan data.

```
...
import timber.log.Timber

const val KEY_REVENUE = "revenue_key"
const val KEY_DESSERT_SOLD = "dessert_sold_key"

class MainActivity : AppCompatActivity() {

    ...
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
    }

    override fun onSaveInstanceState(outState: Bundle) {
        super.onSaveInstanceState(outState)
        outState.putInt(KEY_REVENUE, revenue)
        outState.putInt(KEY_DESSERT_SOLD, dessertsSold)
    }

    ...
}
```

Selanjutnya, ambil data yang disimpan. Pengambilan ini dituliskan pada method onCreate.

```
override fun onCreate(savedInstanceState: Bundle?) {
    ...

    dessertTimer = DessertTimer(this.lifecycle)

    if (savedInstanceState != null) {
        revenue = savedInstanceState.getInt(KEY_REVENUE, 0)
        dessertsSold = savedInstanceState.getInt(KEY_DESSERT_SOLD, 0)
```

```
        showCurrentDessert()
    }

    ...
}
```

Method showCurrentDessert perlu dipanggil agar dessert dapat berubah dari cupcake (bentuk default dari app) menjadi dessert sesuai dengan nilai pada variable dessertSold. Pengambilan nilai ini juga dapat dilakukan dengan melakukan override method onRestoreInstanceState. Perbedannya, method ini berjalan setelah method onStart dieksekusi.

4. Summary

Pada modul kali ini kita telah mengetahui lifecycle atau daur hidup dari Activity dan Fragment. Pada materi ini, kita memahami beberapa hal seperti penggunaan Timber untuk kepentingan logging app, penempatan yang tepat untuk menjalankan komponen yang berjalan saat on-screen & berhenti saat off-screen, dan penanganan behaviour saat terjadi configuration changes seperti orientasi layar yang berubah (rotate).

5. Challenge

Pada modul ini, kita telah mencoba menyimpan state untuk variable yang muncul di layar. Akan tetapi, hal ini belum diterapkan pada timer. Pada saat layar dirotasi, hitungan timer kembali ke angka 1. Lakukan penyimpanan state untuk variable yang menampilkan nilai timer ini (variable secondCount—terdapat pada class DessertTimer).