

## MODUL 11

### Dasar Dasar RecyclerView



#### CAPAIAN PEMBELAJARAN

---

Mahasiswa dapat membuat aplikasi yang menerapkan RecyclerView.



#### KEBUTUHAN ALAT/BAHAN/SOFTWARE

---

1. Android Studio 3.4.
2. Handphone Android versi 7.0 (Nougat)
3. Kabel data USB.
4. Driver ADB.



#### DASAR TEORI

---

### 1. Selamat datang

#### Pengantar

Codelab ini mengajarkan Anda cara menggunakan RecyclerView untuk menampilkan daftar item. Berdasarkan konsep yang dipelajari dalam aplikasi pelacak tidur dari seri codelab sebelumnya, Anda mempelajari cara yang lebih baik dan lebih fleksibel untuk menampilkan data. Aplikasi baru Anda akan memperbarui pelacak tidur untuk menggunakan RecyclerView dengan arsitektur yang direkomendasikan.

#### Apa Yang Harus Anda Ketahui

Anda harus terbiasa dengan:

- Membangun antarmuka pengguna (UI) dasar menggunakan aktivitas, fragmen, dan tampilan.
- Menavigasi antar fragmen, dan menggunakan safeArgs untuk meneruskan data antar fragmen.
- Menggunakan model tampilan, transformasi, dan LiveData serta pengamatnya.
- Membuat database Room, membuat DAO, dan menentukan entitas.
- Menggunakan coroutine untuk tugas database dan tugas lain yang berjalan lama.

## Apa yang akan Anda pelajari

- Cara menggunakan RecyclerView dengan Adaptor dan ViewHolder untuk menampilkan daftar item.

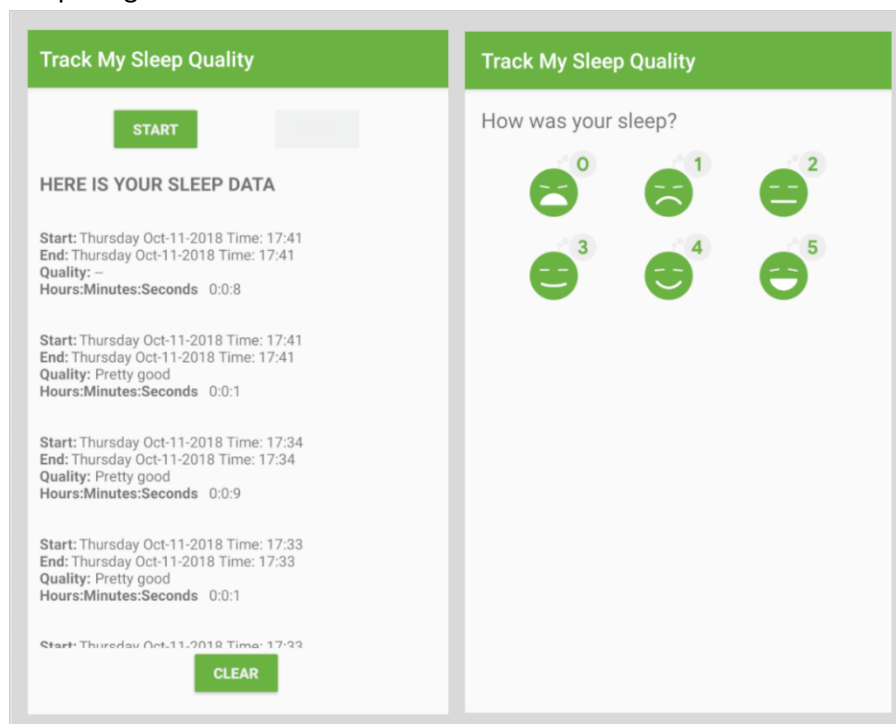
## Apa yang akan Anda lakukan

- Ubah aplikasi TrackMySleepQuality dari pelajaran sebelumnya untuk menggunakan RecyclerView untuk menampilkan data kualitas tidur.

## 2. Ikhtisar aplikasi

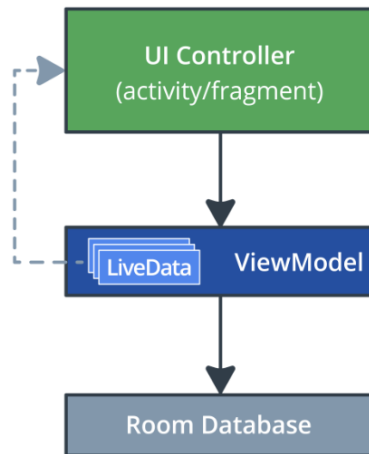
Dalam codelab ini, Anda membuat bagian RecyclerView dari sebuah aplikasi yang melacak kualitas tidur. Aplikasi ini menggunakan database Room untuk menyimpan data tidur dari waktu ke waktu.

Aplikasi pelacak tidur pemula memiliki dua layar, yang diwakili oleh fragmen, seperti yang ditunjukkan pada gambar di bawah ini.

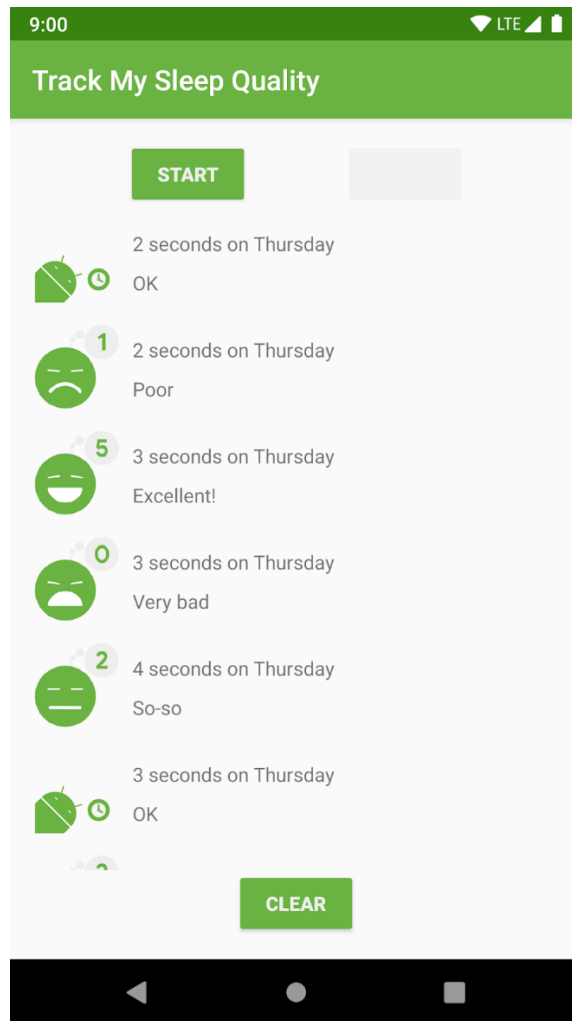


Layar pertama, ditampilkan di sebelah kiri, memiliki tombol untuk memulai dan menghentikan pelacakan. Layar ini juga menampilkan semua data tidur pengguna. Tombol **Clear** secara permanen menghapus semua data yang telah dikumpulkan aplikasi untuk pengguna. Layar kedua, ditampilkan di sebelah kanan, adalah untuk memilih peringkat kualitas tidur.

Aplikasi ini menggunakan arsitektur yang disederhanakan dengan pengontrol UI, **ViewModel**, dan **LiveData**. Aplikasi ini juga menggunakan database **Room** untuk membuat data sleep tetap ada.



Daftar malam tidur yang ditampilkan di layar pertama berfungsi, tetapi tidak cantik. Aplikasi ini menggunakan pemformat kompleks untuk membuat string teks untuk tampilan teks dan angka untuk kualitas. Selain itu, desain ini agak rumit yang menurunkan skalabilitas kami. Setelah Anda memperbaiki semua masalah ini di codelab ini, aplikasi terakhir memiliki fungsi yang sama dengan aplikasi asli tetapi layar utama yang ditingkatkan lebih mudah dibaca:



### 3. Konsep: RecyclerView

Menampilkan daftar atau kisi data adalah salah satu tugas UI paling umum di Android. Daftar bervariasi dari yang sederhana hingga yang sangat kompleks. Daftar tampilan teks mungkin memperlihatkan data sederhana, seperti daftar belanja. Daftar yang kompleks, seperti daftar tujuan liburan yang dianotasi, mungkin menampilkan banyak detail kepada pengguna di dalam kotak scroll dengan header.

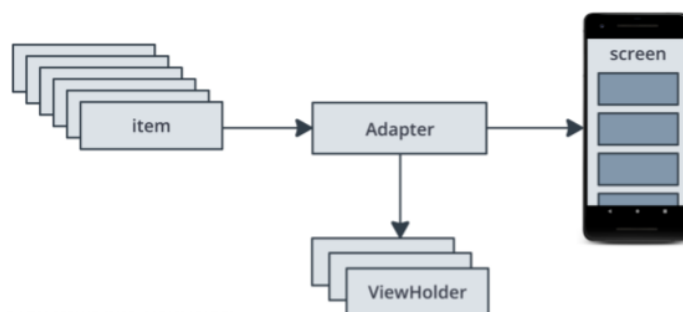
Untuk mendukung semua kasus penggunaan ini, Android menyediakan widget **RecyclerView**.



Manfaat terbesar RecyclerView adalah sangat efisien untuk daftar besar:

- Secara default, RecyclerView hanya berfungsi untuk memproses atau menggambar item yang saat ini terlihat di layar. Misalnya, jika daftar Anda memiliki seribu elemen tetapi hanya 10 elemen yang terlihat, RecyclerView hanya cukup berfungsi untuk menggambar 10 item di layar. Saat pengguna menggulir, RecyclerView mencari tahu item baru apa yang seharusnya ada di layar dan cukup berfungsi untuk menampilkan item tersebut.
- Saat sebuah item bergulir dari layar, tampilan item tersebut didaur ulang. Itu berarti item tersebut diisi dengan konten baru saat bergulir ke layar. Perilaku RecyclerView ini menghemat banyak waktu pemrosesan dan membantu daftar bergulir dengan lancar.
- Saat sebuah item berubah, alih-alih menggambar ulang seluruh daftar, RecyclerView bisa memperbarui satu item itu. Ini adalah keuntungan efisiensi yang sangat besar saat menampilkan daftar panjang item yang kompleks!

## Menerapkan RecyclerView



Untuk menampilkan data Anda di RecyclerView, Anda memerlukan bagian-bagian berikut:

- Data untuk ditampilkan.
- Instance **RecyclerView** yang ditentukan dalam file layout Anda, sebagai wadah untuk tampilan.

- Tata letak untuk satu item data. Jika semua item daftar terlihat sama, Anda dapat menggunakan tata letak yang sama untuk semuanya, tetapi itu tidak wajib. Tata letak item harus dibuat secara terpisah dari tata letak fragmen, sehingga tampilan item pada satu waktu dapat dibuat dan diisi dengan data.
- Manajer tata letak. Pengelola tata letak menangani organisasi (tata letak) komponen UI dalam sebuah tampilan.
- Pemegang tampilan (view holder). Pemegang tampilan memperluas kelas **ViewHolder**. Ini berisi informasi tampilan untuk menampilkan satu item dari tata letak item. Pemegang tampilan juga menambahkan informasi yang digunakan RecyclerView untuk memindahkan tampilan secara efisien di sekitar layar.
- **Adaptor**. Adaptor menghubungkan data Anda ke RecyclerView. Ini menyesuaikan data sehingga dapat ditampilkan dalam ViewHolder. RecyclerView menggunakan adaptor untuk mengetahui cara menampilkan data di layar.



## PRAKTIK

---

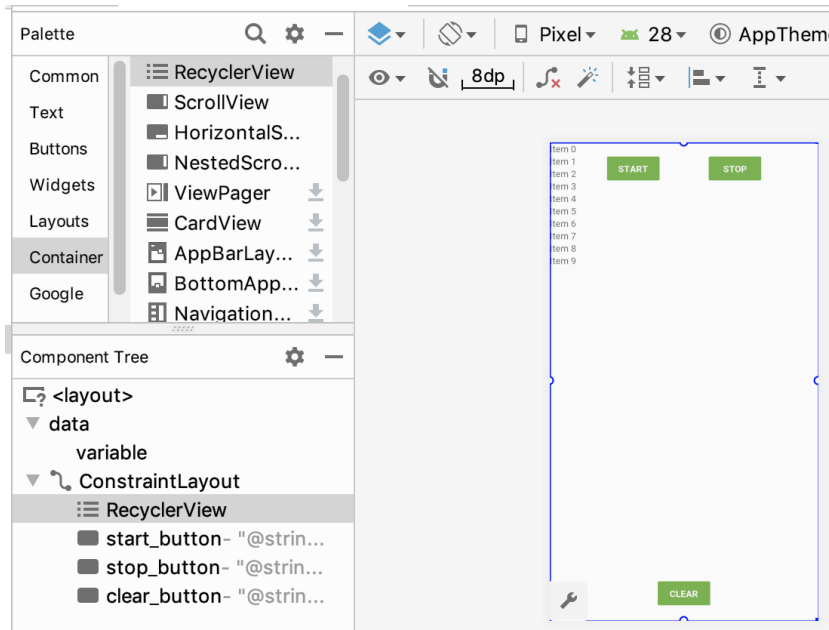
### 4. Tugas: Menerapkan RecyclerView dan Adaptor

Dalam tugas ini, Anda menambahkan **RecyclerView** ke file tata letak Anda dan menyiapkan Adapter untuk mengekspos data tidur ke RecyclerView.

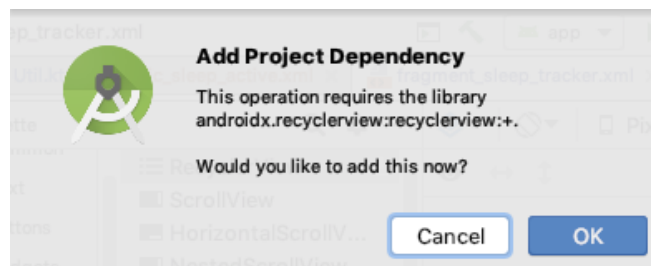
#### Langkah 1: Tambahkan RecyclerView dengan LayoutManager

Pada langkah ini, Anda mengganti **ScrollView** dengan **RecyclerView** di file **fragment\_sleep\_tracker.xml**.

1. Unduh aplikasi [RecyclerViewFundamentals-Starter](#) dari GitHub.
2. Bangun dan jalankan aplikasi. Perhatikan bagaimana data ditampilkan sebagai teks sederhana.
3. Buka file layout **fragment\_sleep\_tracker.xml** di tab **Design** di Android Studio.
4. Di panel **Component Tree**, hapus **ScrollView**. Tindakan ini juga menghapus **TextView** yang ada di dalam **ScrollView**.
5. Di panel **Palette**, gulir daftar jenis komponen di sebelah kiri untuk menemukan **Container**, lalu pilih.
6. Seret **RecyclerView** dari panel **Palette** ke panel **Component Tree**. Tempatkan **RecyclerView** di dalam **ConstraintLayout**.



7. Jika sebuah dialog terbuka menanyakan apakah Anda ingin menambahkan dependensi, klik **OK** untuk mengizinkan Android Studio menambahkan dependensi recyclerview ke file Gradle Anda. Mungkin perlu beberapa detik, lalu aplikasi Anda disinkronkan.



8. Buka file module **build.gradle**, gulir hingga akhir, dan catat dependensi baru, yang terlihat mirip dengan kode di bawah ini:

```
implementation 'androidx.recyclerview:recyclerview: 1.0.0'
```

9. Beralih kembali ke **fragment\_sleep\_tracker.xml**.

10. Di tab **Code**, cari kode **RecyclerView** yang ditunjukkan di bawah ini:

```
<androidx.recyclerview.widget.RecyclerView
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

11. Berikan **RecyclerView** sebuah **id** dengan **sleep\_list**.

```
android:id="@+id/sleep_list"
```

12. Posisikan **RecyclerView** untuk mengambil bagian layar yang tersisa di dalam **ConstraintLayout**. Untuk melakukan ini, batasi bagian atas RecyclerView ke tombol

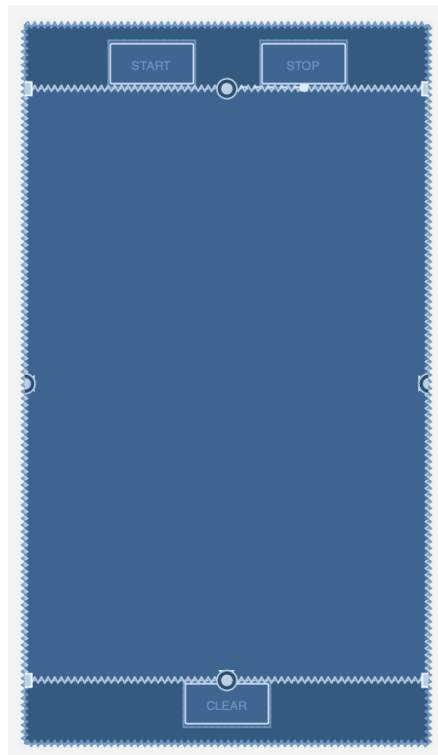
**Start**, bagian bawah ke tombol **Clear**, dan setiap sisi ke induk. Setel lebar dan tinggi tata letak menjadi 0 dp di Layout Editor atau XML, menggunakan kode berikut:

```
android:layout_width="0dp"
android:layout_height="0dp"
app:layout_constraintBottom_toTopOf="@+id/clear_button"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/stop_button"
```

13. Tambahkan pengelola tata letak ke XML **RecyclerView**. Setiap **RecyclerView** membutuhkan pengelola tata letak yang memberi tahu cara memposisikan item dalam daftar. Android menyediakan **LinearLayoutManager**, yang secara default meletakkan item dalam daftar vertikal dengan baris lebar penuh

```
app:layoutManager="androidx.recyclerview.widget.LinearLayoutManager"
```

14. Beralih ke tab **Desain** dan perhatikan batasan yang ditambahkan telah menyebabkan **RecyclerView** meluas untuk mengisi ruang yang tersedia.



## Langkah 2: Buat tata letak item daftar dan pemegang tampilan teks

RecyclerView hanyalah sebuah wadah. Pada langkah ini, Anda membuat tata letak dan infrastruktur untuk item yang akan ditampilkan di dalam RecyclerView.

Untuk mendapatkan RecyclerView yang berfungsi secepat mungkin, pertama-tama Anda menggunakan item daftar sederhana yang hanya menampilkan kualitas tidur sebagai angka.



Untuk ini, Anda memerlukan pemegang tampilan (view holder), **TextItemViewHolder**. Anda juga membutuhkan tampilan, **TextView**, untuk data. (Di langkah selanjutnya, Anda mempelajari lebih lanjut tentang pemegang tampilan dan cara menyusun semua data tidur.)

1. Buat file tata letak bernama **text\_item\_view.xml**. Tidak masalah apa yang Anda gunakan sebagai elemen root, karena Anda akan mengganti kode template.
2. Di **text\_item\_view.xml**, hapus semua kode yang diberikan.
3. Tambahkan **TextView** dengan **padding 16dp** di awal dan akhir, dan ukuran teks **24sp**. Biarkan lebarnya cocok dengan induknya, dan tinggi membungkus konten. Karena tampilan ini ditampilkan di dalam **RecyclerView**, Anda tidak perlu menempatkan tampilan di dalam **ViewGroup**.

```
<?xml version="1.0" encoding="utf-8"?>
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:textSize="24sp"
    android:paddingStart="16dp"
    android:paddingEnd="16dp"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" />
```

4. Buka **Util.kt**. Gulir ke akhir dan tambahkan definisi yang ditampilkan di bawah ini, yang membuat kelas **TextItemViewHolder**. Letakkan kode di bagian bawah file, setelah tanda kurung tutup terakhir. Kode masuk ke **Util.kt** karena view holder ini bersifat sementara, dan Anda menggantinya nanti.

```
class TextItemViewHolder(val textView: TextView):
    RecyclerView.ViewHolder(textView)
```

5. Jika Anda diminta, impor **android.widget.TextView** dan **androidx.recyclerview.widget.RecyclerView**.

### Langkah 3: Buat **SleepNightAdapter**

Tugas inti dalam mengimplementasikan **RecyclerView** adalah membuat adaptor.

Anda memiliki pemegang tampilan sederhana untuk tampilan item, dan tata letak untuk setiap item. Dengan kedua bagian ini, Anda sekarang dapat membuat adaptor. Adaptor membuat view holder dan mengisinya dengan data untuk ditampilkan oleh **RecyclerView**.

1. Dalam paket **sleeptracker**, buat kelas Kotlin baru bernama **SleepNightAdapter**.
2. Buat kelas **SleepNightAdapter** memperluas **RecyclerView.Adapter**. Kelas tersebut disebut **SleepNightAdapter** karena ia mengadaptasi objek **SleepNight** menjadi sesuatu yang dapat digunakan oleh **RecyclerView**. Adaptor perlu mengetahui view holder yang akan digunakan, jadi melewati **TextItemViewHolder**. Impor komponen yang diperlukan saat diminta, dan Anda akan melihat kesalahan, karena ada metode wajib untuk diterapkan.

```
class SleepNightAdapter: RecyclerView.Adapter <TextItemViewHolder> () {}
```

3. Di tingkat teratas **SleepNightAdapter**, buat variabel **listOf SleepNight** untuk menyimpan data.

```
var data = listOf <SleepNight> ()
```

4. Dalam **SleepNightAdapter**, override **getItemCount ()** untuk mengembalikan ukuran daftar tidur malam dalam data. **RecyclerView** perlu mengetahui berapa banyak item yang dimiliki adaptor untuk ditampilkan, dan ia melakukannya dengan memanggil **getItemCount ()**.

```
override fun getItemCount() = data.size
```

5. Dalam **SleepNightAdapter**, override fungsi **onBindViewHolder ()**, seperti yang ditunjukkan di bawah ini.

Fungsi **onBindViewHolder ()** dipanggil oleh **RecyclerView** untuk menampilkan data untuk satu item daftar pada posisi yang ditentukan. Jadi metode **onBindViewHolder ()** mengambil dua argumen: view holder, dan posisi data untuk diikat. Untuk aplikasi ini, pemegangnya adalah **TextItemViewHolder**, dan posisinya adalah posisi dalam daftar.

```
override fun onBindViewHolder(holder: TextItemViewHolder, position: Int) {  
}
```

6. Di dalam **onBindViewHolder ()**, buat variabel untuk satu item pada posisi tertentu dalam data.

```
val item = data [position]
```

7. **ViewHolder** yang baru saja Anda buat memiliki properti yang disebut **textView**. Di dalam **onBindViewHolder ()**, setel teks **textView** ke nomor kualitas tidur. Kode ini hanya menampilkan daftar angka, tetapi contoh sederhana ini memungkinkan Anda melihat bagaimana adaptor memasukkan data ke view holder dan ke layar.

```
holder.textView.text = item.sleepQuality.toString ()
```

8. Dalam **SleepNightAdapter**, timpa dan terapkan **onCreateViewHolder ()**, yang dipanggil saat **RecyclerView** membutuhkan pemegang tampilan.

Fungsi ini mengambil dua parameter dan mengembalikan **ViewHolder**. Parameter **parent**, yang merupakan grup tampilan yang menampung pemegang tampilan, selalu **RecyclerView**. Parameter **viewType** digunakan jika ada beberapa tampilan di **RecyclerView** yang sama. Misalnya, jika Anda meletakkan daftar tampilan teks, gambar, dan video di

**RecyclerView** yang sama, fungsi **onCreateViewHolder ()** perlu mengetahui jenis tampilan yang akan digunakan.

```
override fun onCreateViewHolder(parent: ViewGroup, viewType: Int):  
    TextItemViewHolder {  
    }
```

9. Di **onCreateViewHolder ()**, buat instance **LayoutInflater**.

Layout inflater (pemecah tata letak) tahu cara membuat tampilan dari tata letak XML. **context** berisi informasi tentang cara mengembangkan tampilan dengan benar. Dalam adaptor untuk tampilan pendaur ulang, Anda selalu melewati dalam konteks grup tampilan induk, yaitu **RecyclerView**.

```
val inflater = LayoutInflater.from (parent.context)
```

10. Di **onCreateViewHolder ()**, buat **view** dengan meminta **inflater** untuk memekarkannya (inflate).

Melewatkan tata letak XML untuk tampilan tersebut, dan grup tampilan induk untuk tampilan tersebut. Argumen ketiga, boolean, adalah **attachToRoot**. Argumen ini harus **false**, karena **RecyclerView** menambahkan item ini ke hierarki tampilan untuk Anda saat waktunya tiba.

```
val view = inflater  
    .inflate(R.layout.text_item_view, parent, false) as TextView
```

11. Di **onCreateViewHolder ()**, kembalikan **TextItemViewHolder** yang dibuat dengan **view**.

```
return TextItemViewHolder(view)
```

12. Adaptor perlu memberi tahu **RecyclerView** jika datanya telah berubah, karena **RecyclerView** tidak mengetahui apa pun tentang data tersebut. Itu hanya tahu tentang pemegang tampilan yang diberikan adaptor padanya.

Untuk memberi tahu **RecyclerView** saat data yang ditampilkannya telah berubah, tambahkan penyetel khusus ke variabel data yang ada di bagian atas kelas **SleepNightAdapter**. Di penyetel, berikan data nilai baru, lalu panggil **notifyDataSetChanged ()** untuk memicu penggambaran ulang daftar dengan data baru.

```
var data = listOf<SleepNight>()  
    set(value) {  
        field = value  
        notifyDataSetChanged()  
    }
```

#### Langkah 4: Beri tahu RecyclerView tentang Adaptor

**RecyclerView** perlu mengetahui tentang adaptor yang akan digunakan untuk mendapatkan pemegang tampilan.

1. Buka **SleepTrackerFragment.kt**.
2. Di **onCreateView ()**, buat adaptor. Letakkan kode ini setelah pembuatan model **ViewModel**, dan sebelum pernyataan **return**.

```
val adapter = SleepNightAdapter()
```

3. Setelah Anda mendapatkan referensi ke objek binding, kaitkan adaptor dengan **RecyclerView**.

```
binding.sleepList.adapter = adapter
```

4. Bersihkan dan buat ulang proyek Anda untuk memperbarui objek binding.

Jika Anda masih melihat kesalahan di sekitar **binding.sleepList** atau **binding.FragmentSleepTrackerBinding**, batalkan cache dan mulai ulang. (Pilih **File > Invalidate Caches / Restart**.)

Jika Anda menjalankan aplikasi sekarang, tidak ada kesalahan, tetapi Anda tidak akan melihat data apa pun yang ditampilkan saat Anda mengetuk **Start**, lalu **Stop**.

#### Langkah 5: Masukkan data ke adaptor

Sejauh ini Anda memiliki adaptor, dan cara untuk mendapatkan data dari adaptor tersebut ke **RecyclerView**. Sekarang Anda perlu memasukkan data ke adaptor dari **ViewModel**.

1. Buka **SleepTrackerViewModel**.
2. Temukan variabel **nights**, yang menyimpan semua malam tidur, yang merupakan data untuk ditampilkan. Variabel **nights** disetel dengan memanggil **getAllNights ()** pada database.
3. Hapus **private** dari **nights**, karena Anda akan membuat pengamat yang perlu mengakses variabel ini. Deklarasi Anda akan terlihat seperti ini:

```
val nights = database.getAllNights()
```

4. Dalam paket **database**, buka **SleepDatabaseDao**.
5. Temukan fungsi **getAllNights ()**. Perhatikan bahwa fungsi ini mengembalikan daftar nilai **SleepNight** sebagai **LiveData**. Artinya, variabel **nights** berisi **LiveData** yang terus diperbarui oleh **Room**, dan Anda dapat mengamati **nights** untuk mengetahui kapan berubah.
6. Buka **SleepTrackerFragment**.
7. Dalam **onCreateView ()**, setelah **ViewModel** dibuat dan Anda memiliki referensinya, buat pengamat pada variabel **nights**.

Dengan menyediakan **viewLifecycleOwner** fragmen sebagai pemilik siklus proses, Anda bisa memastikan pengamat ini hanya aktif saat **RecyclerView** ada di layar.

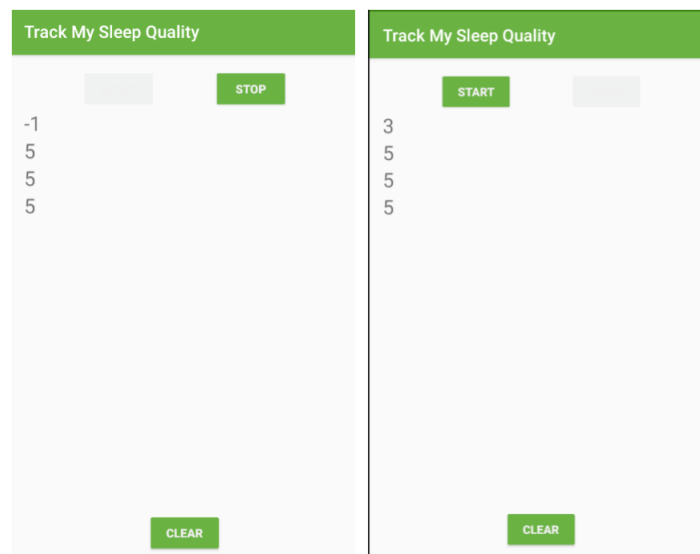
```
sleepTrackerViewModel.nights.observe(viewLifecycleOwner, Observer {  
    })
```

8. Di dalam observer (pengamat), setiap kali Anda mendapatkan nilai bukan nol (untuk **nights**), tetapkan nilai ke **data** adaptor. Ini adalah kode lengkap untuk pengamat dan pengaturan datanya:

```
sleepTrackerViewModel.nights.observe(viewLifecycleOwner, Observer {  
    it?.let {  
        adapter.data = it  
    }  
})
```

9. Buat dan jalankan kode Anda.

Anda akan melihat nomor kualitas tidur sebagai daftar, jika adaptor Anda berfungsi. Tangkapan layar di sebelah kiri menunjukkan -1 setelah Anda mengetuk **Start**. Tangkapan layar di sebelah kanan menunjukkan nomor kualitas tidur yang diperbarui setelah Anda mengetuk **Stop** dan pilih peringkat kualitas.



### Langkah 6: Jelajahi bagaimana pemegang tampilan didaur ulang

**RecyclerView** mendaur ulang pemegang tampilan, yang artinya dapat digunakan kembali. Saat tampilan bergulir dari layar, **RecyclerView** menggunakan kembali tampilan untuk item yang akan digulir ke layar.

Karena pemegang tampilan ini didaur ulang, pastikan **onBindViewHolder ()** menyetel atau menyetel ulang penyesuaian apa pun yang mungkin telah disetel item sebelumnya pada pemegang tampilan.

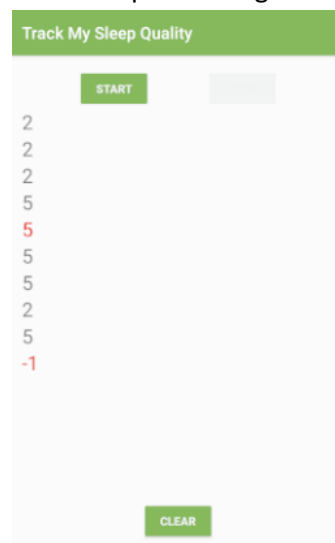
Misalnya, Anda dapat menyetel warna teks menjadi merah di pemegang tampilan yang memiliki peringkat kualitas yang kurang dari atau sama dengan 1 dan menunjukkan kualitas tidur yang buruk.

1. Di kelas **SleepNightAdapter**, tambahkan kode berikut di akhir **onBindViewHolder ()**.

```
if (item.sleepQuality <= 1) {  
    holder.textView.setTextColor(Color.RED) // red  
}
```

2. Jalankan aplikasinya.
3. Tambahkan beberapa data kualitas tidur rendah, dan angkanya berwarna merah.
4. Tambahkan peringkat tinggi untuk kualitas tidur sampai Anda melihat angka merah tinggi di layar.

Saat **RecyclerView** menggunakan kembali pemegang tampilan, pada akhirnya **RecyclerView** menggunakan kembali salah satu pemegang tampilan merah untuk peringkat kualitas tinggi. Peringkat tinggi secara keliru ditampilkan dengan warna merah.



5. Untuk memperbaikinya, tambahkan pernyataan lain untuk mengatur warna menjadi hitam jika kualitasnya tidak kurang dari atau sama dengan satu.

Dengan kedua kondisi eksplisit, view holder akan menggunakan warna teks yang benar untuk setiap item.

```
if (item.sleepQuality <= 1) {  
    holder.textView.setTextColor(Color.RED) // red  
} else {  
    // reset  
    holder.textView.setTextColor(Color.BLACK) // black  
}
```

6. Jalankan aplikasi, dan angkanya harus selalu memiliki warna yang benar.

Selamat! Anda sekarang memiliki **RecyclerView** dasar yang berfungsi penuh.

## 5. Tugas: Membuat ViewHolder untuk semua data tidur

Dalam tugas ini, Anda mengganti dudukan tampilan sederhana dengan yang dapat menampilkan lebih banyak data untuk tidur malam.

**ViewHolder** sederhana yang Anda tambahkan ke **Util.kt** hanya membungkus **TextView** dalam **TextItemViewHolder**.

```
class TextItemViewHolder (val textView: TextView): RecyclerView.ViewHolder  
(textView)
```

Jadi mengapa **RecyclerView** tidak hanya menggunakan **TextView** secara langsung? Baris kode yang satu ini menyediakan banyak fungsi. **ViewHolder** mendeskripsikan tampilan item dan metadata tentang tempatnya di dalam **RecyclerView**. **RecyclerView** mengandalkan fungsionalitas ini untuk memposisikan tampilan dengan benar saat daftar bergulir, dan untuk melakukan hal-hal menarik seperti tampilan animasi saat item ditambahkan atau dihapus di Adaptor.

Jika **RecyclerView** memang perlu mengakses tampilan yang disimpan di **ViewHolder**, **RecyclerView** bisa melakukannya menggunakan properti **itemView** pemegang tampilan. **RecyclerView** menggunakan **itemView** saat mengikat item untuk ditampilkan di layar, saat menggambar dekorasi di sekitar tampilan seperti batas, dan untuk mengimplementasikan aksesibilitas.

### Langkah 1: Buat tata letak item

Pada langkah ini, Anda membuat file tata letak untuk satu item. Tata letak terdiri dari **ConstraintLayout** dengan **ImageView** untuk kualitas tidur, **TextView** untuk durasi tidur, dan **TextView** untuk kualitas sebagai teks. Karena Anda pernah melakukan tata letak sebelumnya, salin dan tempel kode XML yang disediakan.

1. Buat file sumber daya tata letak baru dan beri nama **list\_item\_sleep\_night**.
2. Ganti semua kode di file dengan kode di bawah ini. Kemudian biasakan diri Anda dengan tata letak yang baru saja Anda buat.

```
<?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="wrap_content">

<ImageView
    android:id="@+id/quality_image"
    android:layout_width="@dimen/icon_size"
    android:layout_height="60dp"
    android:layout_marginStart="16dp"
    android:layout_marginTop="8dp"
    android:layout_marginBottom="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    tools:srcCompat="@drawable/ic_sleep_5" />

<TextView
    android:id="@+id/sleep_length"
    android:layout_width="0dp"
    android:layout_height="20dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="16dp"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toEndOf="@+id/quality_image"
    app:layout_constraintTop_toTopOf="@+id/quality_image"
    tools:text="Wednesday" />

<TextView
    android:id="@+id/quality_string"
    android:layout_width="0dp"
    android:layout_height="20dp"
    android:layout_marginTop="8dp"
    app:layout_constraintEnd_toEndOf="@+id/sleep_length"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="@+id/sleep_length"
    app:layout_constraintTop_toBottomOf="@+id/sleep_length"
    tools:text="Excellent!!!" />
</androidx.constraintlayout.widget.ConstraintLayout>

```

3. Beralih ke tab **Desain** di Android Studio. Dalam tampilan Desain, tata letak Anda terlihat seperti tangkapan layar di kiri bawah. Dalam tampilan Cetak Biru, ini terlihat seperti tangkapan layar di sebelah kanan.



## Langkah 2: Buat ViewHolder

1. Buka **SleepNightAdapter.kt**.



2. Buat kelas di dalam **SleepNightAdapter** yang disebut **ViewHolder** dan buat itu memperluas **RecyclerView.ViewHolder**.

```
class ViewHolder(itemView: View) : RecyclerView.ViewHolder(itemView){}
```

3. Di dalam **ViewHolder**, dapatkan referensi ke tampilan. Anda memerlukan referensi ke tampilan yang akan diperbarui **ViewHolder** ini. Setiap kali Anda mengikat **ViewHolder** ini, Anda perlu mengakses gambar dan kedua tampilan teks. (Anda mengonversi kode ini untuk menggunakan data binding nanti.)

```
val sleepLength: TextView = itemView.findViewById(R.id.sleep_length)
val quality: TextView = itemView.findViewById(R.id.quality_string)
val qualityImage: ImageView = itemView.findViewById(R.id.quality_image)
```

### Langkah 3: Gunakan ViewHolder di SleepNightAdapter

1. Dalam definisi signature kelas **SleepNightAdapter**, alih-alih **TextItemViewHolder**, gunakan **SleepNightAdapter.ViewHolder** yang baru saja Anda buat.

```
class SleepNightAdapter: RecyclerView.Adapter<SleepNightAdapter.ViewHolder>() {
```

Perbarui **onCreateViewHolder ()**:

2. Ubah signature **onCreateViewHolder ()** untuk mengembalikan **ViewHolder**.
3. Ubah inflator tata letak untuk menggunakan sumber daya tata letak yang benar, **list\_item\_sleep\_night \*.\***
4. Hapus cast ke **TextView**.
5. Sebagai ganti mengembalikan **TextItemViewHolder**, kembalikan **ViewHolder**.

Berikut adalah fungsi **onCreateViewHolder ()** yang telah diperbarui:

```
override fun onCreateViewHolder(
    parent: ViewGroup, viewType: Int): ViewHolder {
    val inflater =
        LayoutInflater.from(parent.context)
    val view = inflater
        .inflate(R.layout.list_item_sleep_night,
            parent, false)
    return ViewHolder(view)
}
```

Perbarui **onBindViewHolder ()**:

6. Ubah signature **onBindViewHolder ()** sehingga parameter **holder** adalah **ViewHolder**, bukan **TextItemViewHolder**.
7. Di dalam **onBindViewHolder ()**, hapus semua kode, kecuali definisi **item**.
8. Tentukan nilai **val res** yang menyimpan referensi ke **resources** untuk tampilan ini.

```
val res = holder.itemView.context.resources
```

9. Setel teks tampilan teks **sleepLength** ke durasi. Salin kode di bawah ini, yang memanggil fungsi pemformatan yang disediakan dengan kode awal.

```
holder.sleepLength.text = convertDurationToFormatted (item.startTimeMilli,
item.endTimeMilli, res)
```

10. Ini memberikan kesalahan, karena **convertDurationToFormatted ()** perlu ditentukan. Buka **Util.kt** dan hapus komentar kode dan impor terkait untuknya. (Pilih **Code > Comment with Line comments.**)

11. Kembali ke **onBindViewHolder ()** di **SleepNightAdapter.kt**, gunakan **convertNumericQualityToString ()** untuk menyetel **quality**.

```
holder.quality.text = convertNumericQualityToString (item.sleepQuality, res)
```

12. Anda mungkin perlu mengimpor fungsi ini secara manual.

```
import com.example.android.trackmysleepquality.convertDurationToFormatted
import com.example.android.trackmysleepquality.convertNumericQualityToString
```

13. Segera setelah Anda mengatur kualitas, atur ikon yang benar untuk kualitas tersebut. Ikon **ic\_sleep\_active** baru disediakan untuk Anda di kode awal.

```
holder.qualityImage.setImageResource(when (item.sleepQuality) {
    0 -> R.drawable.ic_sleep_0
    1 -> R.drawable.ic_sleep_1
    2 -> R.drawable.ic_sleep_2
    3 -> R.drawable.ic_sleep_3
    4 -> R.drawable.ic_sleep_4
    5 -> R.drawable.ic_sleep_5
    else -> R.drawable.ic_sleep_active
})
```

14. Berikut adalah fungsi **onBindViewHolder ()** yang telah diperbarui, mengatur semua data untuk **ViewHolder**:

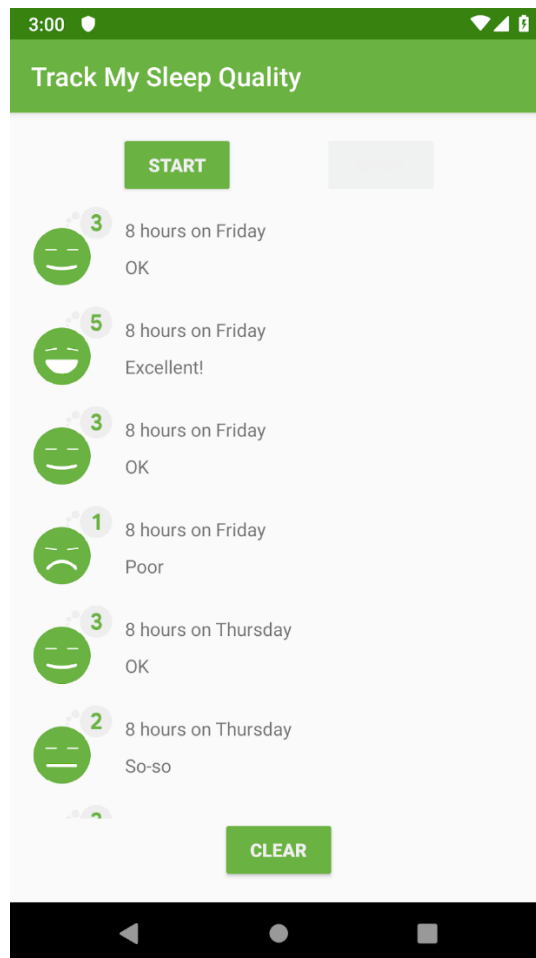
```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val item = data[position]
    val res = holder.itemView.context.resources
    holder.sleepLength.text = convertDurationToFormatted(item.startTimeMilli,
item.endTimeMilli, res)
    holder.quality.text= convertNumericQualityToString(item.sleepQuality,
res)
    holder.qualityImage.setImageResource(when (item.sleepQuality) {
        0 -> R.drawable.ic_sleep_0
        1 -> R.drawable.ic_sleep_1
        2 -> R.drawable.ic_sleep_2
        3 -> R.drawable.ic_sleep_3
        4 -> R.drawable.ic_sleep_4
```

```

        5 -> R.drawable.ic_sleep_5
        else -> R.drawable.ic_sleep_active
    })
}

```

15. Jalankan aplikasi Anda. Tampilan Anda akan terlihat seperti gambar di bawah, menunjukkan ikon kualitas tidur, bersama dengan teks untuk durasi tidur dan kualitas tidur.



## 6. Tugas: Tingkatkan kode Anda

**RecyclerView** Anda sekarang sudah selesai! Anda telah mempelajari cara mengimplementasikan **Adapter** dan **ViewHolder**, dan Anda menggabungkannya untuk menampilkan daftar dengan **Adaptor RecyclerView**.

Kode Anda sejauh ini menunjukkan proses pembuatan adaptor dan view holder (pemegang tampilan). Namun, Anda dapat meningkatkan kode ini. Kode untuk ditampilkan dan kode untuk mengelola pemegang tampilan bercampur, dan **onBindViewHolder ()** mengetahui detail tentang cara memperbarui ViewHolder.

Dalam aplikasi produksi, Anda mungkin memiliki beberapa pemegang tampilan, adaptor yang lebih kompleks, dan beberapa pengembang yang membuat perubahan. Anda harus menyusun kode Anda sehingga semua yang terkait dengan pemegang tampilan hanya di pemegang tampilan.

### Langkah 1: Refactor onBindViewHolder ()

Dalam langkah ini, Anda memfaktorkan ulang kode dan memindahkan semua fungsionalitas pemegang tampilan ke **ViewHolder**. Tujuan dari pemfaktoran ulang ini bukan untuk mengubah tampilan aplikasi bagi pengguna, tetapi membuatnya lebih mudah dan lebih aman bagi pengembang untuk mengerjakan kode. Untungnya, Android Studio memiliki alat untuk membantu.

1. Di **SleepNightAdapter.kt**, di fungsi **onBindViewHolder ()**, pilih semuanya kecuali pernyataan untuk mendeklarasikan variabel **item**.
2. Klik kanan, lalu pilih **Refactor > Extract > Function**. (**Refactor > Function** untuk Android Studio 4.x)
3. Beri nama fungsi **bind** dan terima parameter yang disarankan. Klik OK.

Fungsi **bind ()** ditempatkan di bawah **onBindViewHolder ()**.

```
private fun bind(holder: ViewHolder, item: SleepNight) {
    val res = holder.itemView.context.resources
    holder.sleepLength.text = convertDurationToFormatted(item.startTimeMilli,
item.endTimeMilli, res)
    holder.quality.text = convertNumericQualityToString(item.sleepQuality,
res)
    holder.qualityImage.setImageResource(when (item.sleepQuality) {
        0 -> R.drawable.ic_sleep_0
        1 -> R.drawable.ic_sleep_1
        2 -> R.drawable.ic_sleep_2
        3 -> R.drawable.ic_sleep_3
        4 -> R.drawable.ic_sleep_4
        5 -> R.drawable.ic_sleep_5
        else -> R.drawable.ic_sleep_active
    })
}
```

4. Buat fungsi ekstensi dengan signature berikut:

```
private fun ViewHolder.bind(item: SleepNight) {...}
```

5. Potong dan tempel fungsi **ViewHolder.bind ()** ini ke kelas dalam **ViewHolder** di bagian bawah **SleepNightAdapter.kt**.
6. Jadikan **bind ()** publik.
7. Impor **bind ()** ke adaptor, jika perlu.
8. Karena sekarang ada di **ViewHolder**, Anda bisa menghapus bagian **ViewHolder** dari signature. Berikut adalah kode terakhir untuk fungsi **bind ()** di kelas **ViewHolder**.

```

fun bind(item: SleepNight) {
    val res = itemView.context.resources
    sleepLength.text = convertDurationToFormatted(
        item.startTimeMilli, item.endTimeMilli, res)
    quality.text = convertNumericQualityToString(
        item.sleepQuality, res)
    qualityImage.setImageResource(when (item.sleepQuality) {
        0 -> R.drawable.ic_sleep_0
        1 -> R.drawable.ic_sleep_1
        2 -> R.drawable.ic_sleep_2
        3 -> R.drawable.ic_sleep_3
        4 -> R.drawable.ic_sleep_4
        5 -> R.drawable.ic_sleep_5
        else -> R.drawable.ic_sleep_active
    })
}

```

Sekarang di **SleepNightAdapter.kt**, fungsi **onBindViewHolder ()** menunjukkan kesalahan referensi yang belum terselesaikan pada **bind (holder, item)**. Fungsi ini sekarang menjadi bagian dari kelas dalam **ViewHolder** jadi kita perlu menentukan **ViewHolder** tertentu. Ini juga berarti kita bisa menghapus argumen pertama.

```

override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val item = data[position]
    holder.bind(item)
}

```

## Langkah 2: Refactor onCreateViewHolder

Metode **onCreateViewHolder ()** di adaptor saat ini memekarkan tampilan dari sumber tata letak untuk **ViewHolder**. Namun, inflasi tidak ada hubungannya dengan adaptor, dan semuanya berkaitan dengan **ViewHolder**. Inflasi harus terjadi di **ViewHolder**.

1. Di **onCreateViewHolder ()**, pilih semua kode di badan fungsi.
2. Klik kanan, lalu pilih **Refactor > Extract > Function**.
3. Beri nama fungsi **from** dan terima parameter yang disarankan. Klik **OK**.
4. Letakkan kursor pada nama fungsi **from**. Tekan **Alt + Enter** untuk membuka menu tujuan.
5. Pilih **Move to companion object**. Fungsi **from ()** harus berada dalam objek pendamping (companion) sehingga bisa dipanggil pada kelas **ViewHolder**, tidak dipanggil pada instance **ViewHolder**.
6. Pindahkan **companion object** (objek pendamping) ke kelas **ViewHolder**.
7. Jadikan **from ()** publik.
8. Dalam **onCreateViewHolder ()**, ubah pernyataan **return** untuk mengembalikan hasil panggilan **from ()** di kelas **ViewHolder**.

Metode **onCreateViewHolder ()** dan **from ()** Anda yang telah selesai akan terlihat seperti kode di bawah ini, dan kode Anda harus dibuat dan dijalankan tanpa kesalahan.

```
    override fun onCreateViewHolder(parent: ViewGroup, viewType:
Int): ViewHolder {
        return ViewHolder.from(parent)
    }
```

```
companion object {
    fun from(parent: ViewGroup): ViewHolder {
        val inflater = LayoutInflater.from(parent.context)
        val view = inflater
            .inflate(R.layout.list_item_sleep_night, parent, false)
        return ViewHolder(view)
    }
}
```

9. Ubah signature kelas **ViewHolder** sehingga konstruktornya bersifat private. Karena **from ()** sekarang menjadi metode yang mengembalikan instance **ViewHolder** baru, tidak ada alasan bagi siapa pun untuk memanggil konstruktor **ViewHolder** lagi.

```
class ViewHolder private constructor(itemView: View) :
RecyclerView.ViewHolder(itemView){
```

10. Jalankan aplikasinya. Aplikasi Anda harus dibuat dan dijalankan sama seperti sebelumnya, yang merupakan hasil yang diinginkan setelah pemfaktoran ulang.



## LATIHAN

---



## TUGAS

---

Buat aplikasi baru dengan mengembangkan project di atas



## REFERENSI

---

1. <https://developer.android.com/codelabs/kotlin-android-training-recyclerview-fundamentals#0>