7.2: Menghubungkan ke Internet dengan AsyncTask dan AsyncTaskLoader

Daftar Isi:

Dalam praktik ini Anda akan menggunakan AsyncTask untuk memulai tugas latar belakang yang mendapatkan data dari internet menggunakan REST API sederhana. Anda akan menggunakan Google API Explorer untuk belajar cara menanyakan Book Search API, mengimplementasikan kueri ini dalam thread worker menggunakan AsyncTask, dan menampilkan hasilnya dalam UI. Lalu Anda akan mengimplementasikan ulang tugas latar belakang yang sama menggunakan AsyncTaskLoader, yang akan lebih efisien dalam memperbarui UI, menangani masalah kinerja, dan meningkatkan keseluruhan UX.

Yang harus sudah Anda KETAHUI

Dari praktik sebelumnya, Anda harus sudah bisa:

- Membuat sebuah aktivitas.
- Menambahkan TextView ke layout untuk aktivitas tersebut.
- Mengimplementasikan fungsionalitas onClick ke tombol dalam layout.
- Mengimplementasikan AsyncTask dan menampilkan hasilnya dalam UI.
- Meneruskan informasi di antara aktivitas sebagai ekstra.

Yang akan Anda PELAJARI

Dalam praktik ini Anda akan belajar:

- Menggunakan Google API Explorer untuk menginvestigasi API Google dan melihat respons JSON terhadap permintaan http.
- Menggunakan Books API sebagai contoh API mengambil data dari internet dan menjaga UI agar cepat dan responsif. Anda tidak akan mempelajari Books API secara detail dalam praktik ini. Aplikasi Anda hanya akan menggunakan fungsi penelusuran book paling sederhana. Untuk mengetahui selengkapnya tentang Books API lihat dokumentasi referensi Books API.
- Parse hasil JSON yang dikembalikan dari kueri API.
- Mengimplementasikan AsyncTaskLoader yang mempertahankan data setelah perubahan konfigurasi.
- Memperbarui UI menggunakan callback Loader.

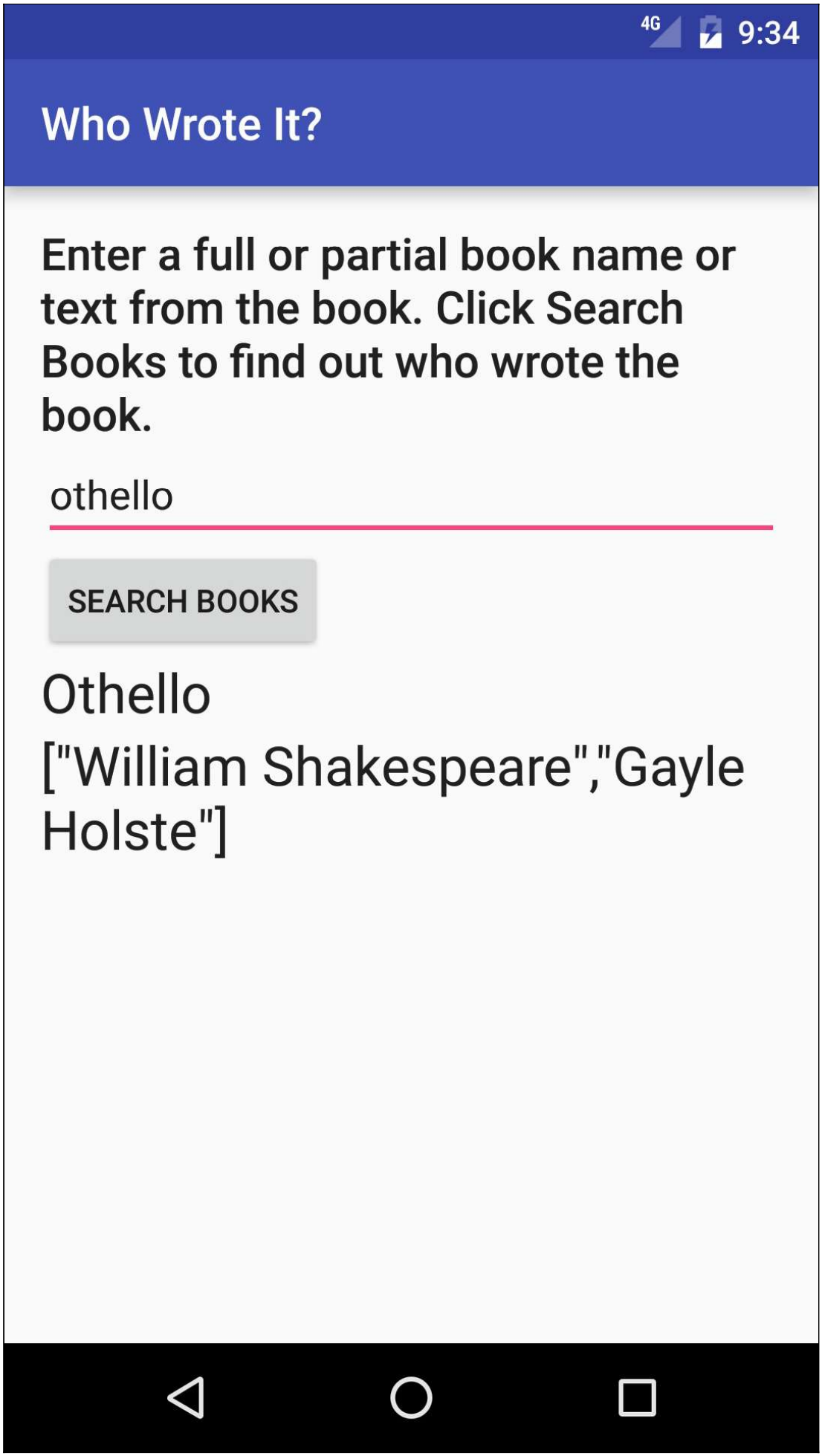
Yang akan Anda LAKUKAN

Dalam praktik ini Anda akan:

- Menggunakan Google API Explorer untuk mempelajari tentang fitur penelusuran sederhana API Books.
- Membuat aplikasi "Who Wrote It?" yang menanyakan Books API menggunakan thread worker dan menampilkan hasilnya dalam UI.
- Memodifikasi aplikasi "Who Wrote it?" untuk menggunakan AsyncTaskLoader, sebagai ganti AsyncTask.

Ringkasan Aplikasi

Anda akan membangun aplikasi yang berisi bidang EditText dan Tombol. Pengguna memasukkan nama buku dalam EditText dan mengeklik tombol. Tombol mengeksekusi AsyncTask yang menanyakan API Penelusuran Google Book untuk menemukan penulis dan judul buku yang dicari pengguna. Hasilnya diambil dan ditampilkan dalam bidang TextView di bawah tombol. Setelah aplikasi bekerja, Anda akan memodifikasi aplikasi untuk menggunakan AsyncTaskLoader sebagai ganti kelas AsyncTask.


4G
9:34
Who Wrote It?
Enter a full or partial book name or text from the book. Click Search Books to find out who wrote the book.
othello
SEARCH BOOKS
Othello
["William Shakespeare","Gayle Holste"]

Tugas 1. Menjelajahi Books API

Dalam praktik ini Anda akan menggunakan API Google Books untuk mencari informasi tentang buku, seperti penulis dan judulnya. API Google Books menyediakan akses pemrograman ke layanan Penelusuran Google Book menggunakan REST API. Ini adalah layanan yang sama yang digunakan di belakang layar saat Anda mengeksekusi penelusuran secara manual di Google Books. Anda bisa menggunakan Google API Explorer dan Penelusuran Google Book pada browser untuk memverifikasi bahwa aplikasi Android Anda mendapatkan hasil yang diharapkan.

1.1 Mengirimkan Permintaan API Books

1. Buka Google APIs Explorer (bisa ditemukan di https://developers.google.com/apis-explorer/).
2. Klik **Books API**.
3. Temukan (**Ctrl-F** atau **Cmd-F**) **books.volumes.list** dan klik nama fungsi tersebut. Anda seharusnya bisa melihat laman web yang mencantumkan berbagai parameter fungsi API Books yang melakukan penelusuran buku.
4. Dalam bidang `q` masukkan nama buku, atau sebagain nama buku. Parameter `q` adalah satu-satunya bidang yang diwajibkan.
5. Gunakan bidang `maxResults` dan `printType` untuk membatasi hasil ke 10 buku yang cocok yang dicetak. Bidang `maxResults` mengambil nilai integer yang membatasi jumlah hasil per kueri. Bidang `printType` mengambil satu dari tiga argumen string: `all`, yang tidak membatasi hasil menurut tipe sama sekali; `books`, yang hanya memberikan hasil buku dalam bentuk cetak; dan `magazines` yang memberikan hasil majalah saja.
6. Pastikan switch "Authorize requests using OAuth 2.0" di bagian atas formulir dinonaktifkan. Klik **Execute without OAuth** di bagian bawah formulir.
7. Gulir ke bawah untuk melihat Permintaan dan Respons.

Bidang `Request` adalah contoh Uniform Resource Identifier (URI). URI adalah string yang memberikan nama atau menemukan sumber daya tertentu. URL adalah tipe URI tertentu untuk mengidentifikasi dan menemukan sumber daya web. Untuk API Books, permintaannya adalah URL yang berisi penelusuran sebagai parameter (mengikuti parameter `q`). Perhatikan bidang kunci API setelah bidang kueri. Untuk alasan keamanan, saat mengakses API publik, Anda biasanya perlu mendapatkan kunci API dan menyertakannya di dalam Permintaan Anda. Namun, API spesifik ini tidak memerlukan kunci, sehingga Anda dapat meninggalkan porsi URI Permintaan dalam aplikasi.

1.2 Menganalisis Respons API Books

Di bagian bawah laman Anda bisa melihat Respons terhadap kueri. Respons menggunakan format JSON, yang merupakan format umum untuk respons kueri API. Dalam laman web API Explorer, kode JSON diformat dengan baik agar dapat dibaca oleh manusia. Dalam aplikasi Anda, repons JSON akan dikembalikan dari layanan API sebagai string tunggal, dan Anda perlu melakukan parsing pada string tersebut untuk mengesktrak informasi yang diperlukan.

1. Dalam bagian Respons, temukan nilai untuk kunci "title". Perhatikan bahwa hasil ini memiliki nilai dan kunci tunggal.
2. Termukan nilai untuk kunci "authors". Perhatikan bahwa kunci ini berisi larik nilai.
3. Dalam praktik ini Anda hanya akan mengembalikan judul dan penulis item pertama.

Tugas 2: Membuat "Who Wrote it?" Aplikasi

Setelah Anda familier dengan metode API Books yang akan digunakan, sekarang waktunya menyiapkan layout aplikasi.

2.1 Membuat proyek dan antarmuka pengguna

1. Buat proyek aplikasi bernama **Who Wrote it?** dengan satu aktivitas, menggunakan Template Empty Activity.
2. Tambahkan elemen UI berikut di dalam file XML, menggunakan LinearLayout vertikal sebagai tampilan root—tampilan yang berisi semua tampilan lain di dalam file XML layout. Pastikan LinearLayout menggunakan `android:orientation="vertical"`:

Tampilan	Atribut	Nilai
TextView	android:layout_width android:layout_height android:id android:text android:textAppearance	wrap_content wrap_content @+id/instructions @string/instructions @style/TextAppearance.AppCompat.Title
EditText	android:layout_width android:layout_height android:id android:inputType android:hint	match_parent wrap_content @+id/bookInput text @string/input_hint
Button	android:layout_width android:layout_height android:id android:text android:onClick	wrap_content wrap_content @+id/searchButton @string/button_text searchBooks
TextView	android:layout_width android:layout_height android:id android:textAppearance	wrap_content wrap_content @+id/titleText @style/TextAppearance.AppCompat.Headline
TextView	android:layout_width android:layout_height android:id android:textAppearance	wrap_content wrap_content @+id/authorText @style/TextAppearance.AppCompat.Headline

3. Dalam file strings.xml, tambahkan sumber daya string berikut ini:

```
<string name="instructions">Enter a book name, or part of a
book name, or just some text from a book to find
the full book title and who wrote the book!</string>
<string name="button_text">Search Books</string>
<string name="input_hint">Enter a Book Title</string>
```

4. Buat metode bernama `searchBooks()` dalam MainActivity.java untuk menangani tindakan tombol onClick. Seperti pada semua metode onClick, yang satu ini memerlukan `View` sebagai parameter.

2.2 Menyiapkan Aktivitas Utama

Untuk menanyakan API Books, Anda perlu mendapatkan masukan pengguna dari EditText.

1. Dalam MainActivity.java, buat variabel anggota untuk EditText, TextView penulis dan TextView judul.
2. Inisialisasi variabel ini dalam `onCreate()` .
3. Dalam metode `searchBooks()` , dapatkan teks dari widget EditText dan konversikan ke `String` , menetapkannya ke variabel string.

```
String queryString = mBookInput.getText().toString();
```

Catatan: `mBookInput.getText()` mengembalikan jenis data "Editable" yang perlu dikonversi menjadi string.

2.3 Membuat AsyncTask kosong

Sekarang Anda siap untuk terhubung ke internet dan menggunakan Book Search REST API. Konektivitas jaringan terkadang lamban atau mengalami penundaan. Hal ini bisa menyebabkan aplikasi menjadi tidak menentu dan lambat, jadi sebaiknya Anda tidak membuat koneksi jaringan pada thread UI. Jika Anda mencoba koneksi jaringan pada thread UI, Waktu Proses Android mungkin akan mengeluarkan NetworkOnMainThreadException untuk memperingatkan Anda bahwa ini bukanlah ide yang baik.

Gunakan AsyncTask untuk membuat koneksi jaringan:

1. Buat kelas Java baru bernama `FetchBook` dalam **aplikasi/java** yang diperluas `AsyncTask`. AsyncTask memerlukan tiga argumen:
   - Parameter masukan.
   - Indikator kemajuan.
   - Jenis hasil.

   Parameter tipe generik untuk tugas adalah `<String, Void, String>` since the AsyncTask takes a `String` as the first parameter (the query), `Void` since there is no progress update, and `String` karena parameter ini mengembalikan string sebagai hasilnya (respons JSON).
2. Implementasikan metode yang diperlukan, `doInBackground()`, dengan meletakkan kursor pada teks yang digarisbawahi merah, menekan **Alt + Enter** (**Opt + Enter** di Mac) dan memilih **Implement methods**. Pilih **doInBackground()** dan klik **OK**. Pastikan parameternya dan kembalikan tipe yang jenisnya benar (Ini memerlukan larik String dan mengembalikan String).
   i. Klik menu **Code** dan pilih **Override methods** (atau tekan **Ctrl + O**). Pilih metode **onPostExecute()**. Metode `onPostExecute()` mengambil `String` sebagai parameter dan mengembalikan `void`.
3. Untuk menampilkan hasil dalam TextView, Anda harus memiliki akses ke TextView yang ada di dalam AsyncTask. Buat variabel anggota dalam FetchBook AsyncTask untuk dua TextView yang menunjukkan hasilnya, dan inisialisasi keduanya dalam konstruktor. Anda akan menggunakan konstruktor ini dalam MainActivity untuk meneruskan TextView ke AsyncTask.

Kode solusi untuk FetchBook:

```
public class FetchBook extends AsyncTask<String,Void,String>{
        private TextView mTitleText;
        private TextView mAuthorText;

   public FetchBook(TextView mTitleText, TextView mAuthorText) {
       this.mTitleText = mTitleText;
       this.mAuthorText = mAuthorText;
   }

   @Override
   protected String doInBackground(String... params) {
       return null;
   }

   @Override
   protected void onPostExecute(String s) {
       super.onPostExecute(s);
   }
}
```

2.4 Membuat kelas NetworkUtils dan membangun URI

Dalam langkah ini, Anda akan membuka koneksi internet dan menanyakan API Books. Bagian ini memiliki banyak kode, jadi ingat untuk membuka dokumentasi developer untuk Menyambungkan ke Jaringan jika Anda merasa buntu. Anda akan menulis kode untuk terhubung ke internet dalam kelas helper bernama NetworkUtils.

1. Buat kelas Java baru bernama NetworkUtils dengan mengeklik **File > New > Java Class** dan hanya mengisi bidang "Name".
2. Buat variabel `LOG_TAG` unik untuk digunakan di semua kelas NetworkUtils untuk membuat catatan log:

```
private static final String LOG_TAG = NetworkUtils.class.getSimpleName();
```

3. Buat metode statis baru bernama `getBookInfo()` yang mengambil `String` sebagai parameter (yang akan menjadi istilah penelusuran) dan mengembalikan `String` (respons String JSON dari API yang Anda periksa sebelumnya).

```
static String getBookInfo(String queryString){}
```

4. Buat dua variabel lokal berikut dalam `getBookInfo()` yang akan dibutuhkan nanti untuk membantu menyambungkan dan membaca data yang datang.

```
HttpURLConnection urlConnection = null;
BufferedReader reader = null;
```

5. Buat variabel lokal lain di akhir `getBookInfo()` untuk memasukkan respons mentah dari kueri dan mengembalikannya:

```
String bookJSONString = null;
return bookJSONString;
```

   Jika Anda ingat permintaan dari laman web API Books, Anda akan memperhatikan bahwa semua permintaan dimulai dengan URI yang sama. Untuk menentukan tipe sumber daya, Anda menambahkan parameter kueri ke URI basis. Memisahkan semua parameter kueri ini ke dalam konstanta dan mengombinasikannya menggunakan Uri.Builder adalah praktik biasa agar bisa digunakan lagi untuk URI yang berbeda. Kelas Uri memiliki metode yang mudah, `Uri.buildUpon()` yang mengembalikan URI.Builder yang bisa kita gunakan.

   Untuk aplikasi ini, Anda akan membatasi jumlah dan jenis hasil yang dikembalikan untuk meningkatkan kecepatan kueri. Untuk membatasi kueri, Anda hanya mencari buku yang dicetak.

6. Buat konstanta anggota berikut dalam kelas NetworkUtils:

```
private static final String BOOK_BASE_URL =  "https://www.googleapis.com/books/v1/volumes?"; // Base URI for the
 Books API
private static final String QUERY_PARAM = "q"; // Parameter for the search string
private static final String MAX_RESULTS = "maxResults"; // Parameter that limits search results
private static final String PRINT_TYPE = "printType";   // Parameter to filter by print type
```

7. Buat blok try/catch/finally skeleton dalam `getBookInfo()`. Di sinilah Anda akan membuat permintaan HTTP. Kode untuk membangun URI dan mengeluarkan kueri akan masuk ke dalam blok try. Blok catch digunakan untuk menangani masalah apa pun dengan membuat permintaan HTTP dan blok finally untuk menutup koneksi jaringan setelah Anda selesai menerima data JSON dan mengembalikan hasilnya.

```
try {
...
} catch (Exception ex) {
...
} finally {
   return bookJSONString;
}
```

8. Bangun URI permintaan dalam blok try:

```
//Build up your query URI, limiting results to 10 items and printed books
Uri builtURI = Uri.parse(BOOK_BASE_URL).buildUpon()
       .appendQueryParameter(QUERY_PARAM, queryString)
       .appendQueryParameter(MAX_RESULTS, "10")
       .appendQueryParameter(PRINT_TYPE, "books")
       .build();
```

9. Konversi URI ke URL:

```
URL requestURL = new URL(builtURI.toString());
```

2.5 Membuat Permintaan

Membuat permintaan API melalui internet adalah hal yang cukup umum. Karena Anda mungkin akan menggunakan fungsionalitas ini lagi, Anda mungkin ingin membuat kelas utilitas dengan fungsionalitas ini atau mengembangkan subkelas yang berguna untuk kemudahan sendiri. Permintaan API ini menggunakan kelas HttpURLConnection yang dikombinasikan dengan InputStream dan StringBuffer untuk mendapatkan respons JSON dari web. Jika pada satu sisi prosesnya gagal dan InputStream atau StringBuffer kosong, proses akan mengembalikan null yang menandakan bahwa kueri gagal.

1. Dalam blok try metode `getBookInfo()`, buka koneksi URL dan buat permintaan.

```
urlConnection = (HttpURLConnection) requestURL.openConnection();
urlConnection.setRequestMethod("GET");
urlConnection.connect();
```

2. Baca respons menggunakan InputStream dan StringBuffer, lalu konversikan ke `String`:

```
InputStream inputStream = urlConnection.getInputStream();
StringBuffer buffer = new StringBuffer();
if (inputStream == null) {
   // Nothing to do.
   return null;
}
reader = new BufferedReader(new InputStreamReader(inputStream));
String line;
while ((line = reader.readLine()) != null) {
   /* Since it's JSON, adding a newline isn't necessary (it won't affect
      parsing) but it does make debugging a *lot* easier if you print out the
      completed buffer for debugging. */
   buffer.append(line + "\n");
}
if (buffer.length() == 0) {
   // Stream was empty.  No point in parsing.
   return null;
}
bookJSONString = buffer.toString();
```

3. Tutup blok try dan log pengecualiannya dalam blok catch.

```
catch (IOException e) {
   e.printStackTrace();
   return null;
}
```

4. Tutup kedua urlConnection dan variabel pembaca dalam blok finally:

```
finally {
   if (urlConnection != null) {
       urlConnection.disconnect();
   }
   if (reader != null) {
       try {
           reader.close();
       } catch (IOException e) {
           e.printStackTrace();
       }
   }
}
```

Catatan: Setiap kali koneksi gagal, kode ini akan mengembalikan null. Ini berarti `onPostExecute()` harus memeriksa parameter masukannya untuk string null dan memungkinkan pengguna mengetahui bahwa koneksi gagal. Strategi

penanganan kesalahan ini sederhana, karena pengguna tidak tahu mengapa koneksinya gagal. Solusi yang lebih baik untuk aplikasi produksi adalah menangani setiap poin kesalahan secara berbeda agar pengguna mendapatkan masukan yang tepat.

5. Log nilai variabel `bookJSONString` sebelum mengembalikannya. Sekarang Anda sudah selesai dengan metode `getBookInfo()`.

```
Log.d(LOG_TAG, bookJSONString);
```

6. Dalam metode AsyncTask `doInBackground()`, panggil metode `getBookInfo()`, meneruskan istilah penelusuran yang Anda dapatkan dari argumen `params` yang diteruskan oleh sistem (ini adalah nilai pertama dalam larik `params`). Kembalikan hasil dari metode ini dalam metode `doInBackground()`:

```
return NetworkUtils.getBookInfo(params[0]);
```

7. Sekarang setelah AsyncTask disiapkan, Anda perlu meluncurkannya dari MainActivity menggunakan metode `execute()`. Tambahkan kode berikut ke metode `searchBooks()` dalam MainActivity.java untuk meluncurkan AsyncTask:

```
new FetchBook(mTitleText, mAuthorText).execute(mQueryString);
```

8. Jalankan aplikasi Anda. Eksekusi penelusuran. Aplikasi Anda akan crash. Lihat Log untuk memeriksa apa yang menyebabkan kesalahan. Anda seharusnya melihat baris berikut:

```
Caused by: java.lang.SecurityException: Permission denied (missing INTERNET permission?)
```

Kesalahan ini menunjukkan bahwa Anda belum menyertakan izin untuk mengakses internet dalam file AndroidManifest.xml. Terhubung ke internet menimbulkan masalah keamanan baru, karena itulah aplikasi Anda tidak memiliki konektivitas secara default. Anda harus menambahkan izin secara manual dalam bentuk tag `<uses-permission>`; dalam AndroidManifest.xml.

2.6 Menambahkan izin internet

1. Buka file **AndroidManifest.xml**.
2. Semua izin aplikasi harus diletakkan dalam file AndroidManifest.xml di luar tag `<application>`;. Anda harus memastikan untuk mengikuti urutan tempat tag didefinisikan dalam AndroidManifest.xml.
3. Tambahkan tag xml berikut di luar tag `<application>`:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

4. Bangun dan jalankan aplikasi Anda lagi. Menjalankan kueri seharusnya menghasilkan string JSON dicetak ke Log.

2.7 Parse string JSON

Sekarang Anda memiliki respons yang benar untuk kueri, Anda harus melakukan parsing kepada hasil untuk mengekstrak informasi yang ingin Anda tampilkan dalam UI. Untungnya, Java memiliki kelas yang sudah ada yang membantu parsing dan menangani data jenis JSON. Proses ini dan pembaruan UI akan terjadi dalam metode `onPostExecute()`.

Ada kemungkinan metode `doInBackground()` tidak mengembalikan string JSON yang diharapkan. Misalnya, try catch mungkin gagal dan mengeluarkan pengecualian, jaringan mungkin sudah habis waktunya atau kesalahan yang tidak dapat ditangani mungkin terjadi. Pada kasus-kasus ini, metode JSON akan gagal melakukan parsing pada data dan akan mengeluarkan pengecualian. Karena itu Anda harus melakukan parsing dalam blok try, dan blok catch harus menangani kasus di mana data yang tidak lengkap dan tidak benar dikembalikan.

Untuk melakukan parsing pada data JSON dan menangani pengecualian yang mungkin, lakukan hal berikut:

1. Dalam `onPostExecute()`, tambahkan blok try/catch di bawag panggilan ke `super`.

2. Gunakan kelas JSON Java bawaan (`JSONObject` dan `JSONArray`) untuk mendapatkan larik JSON item hasil dalam blok try.

```
JSONObject jsonObject = new JSONObject(s);
JSONArray itemsArray = jsonObject.getJSONArray("items");
```

3. Iterasi melalui `itemsArray` , memeriksa judul dan informasi penulis setiap buku. Jika keduanya bukan null, keluar dari loop dan perbarui UI; jika tidak, terus periksa daftarnya. Dengan cara ini, hanya entri dengan judul dan penulis yang akan ditampilkan.

```
//Iterate through the results
for(int i = 0; i<itemsArray.length(); i++){
   JSONObject book = itemsArray.getJSONObject(i); //Get the current item
   String title=null;
   String authors=null;
   JSONObject volumeInfo = book.getJSONObject("volumeInfo");

   try {
       title = volumeInfo.getString("title");
       authors = volumeInfo.getString("authors");
   } catch (Exception e){
       e.printStackTrace();
   }

   //If both a title and author exist, update the TextViews and return
   if (title != null && authors != null){
       mTitleText.setText(title);
       mAuthorText.setText(authors);
       return;
   }
}
```

4. Jika tidak ada hasil yang memenuhi kriteria memiliki penulis dan judul yang valid, setel TextView judul untuk membaca "No Results Found", dan hapus TextView `authors` .
5. Dalam blok catch, cetak kesalahan ke log, setel TextView judul ke "No Results Found", dan hapus TextView `authors` .

Kode solusi:

```
//Method for handling the results on the UI thread
@Override
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    try {
       JSONObject jsonObject = new JSONObject(s);
       JSONArray itemsArray = jsonObject.getJSONArray("items");
       for(int i = 0; i<itemsArray.length(); i++){
           JSONObject book = itemsArray.getJSONObject(i);
           String title=null;
           String authors=null;
           JSONObject volumeInfo = book.getJSONObject("volumeInfo");


           try {
               title = volumeInfo.getString("title");
               authors = volumeInfo.getString("authors");
           } catch (Exception e){
               e.printStackTrace();
           }


           if (title != null && authors != null){
               mTitleText.setText(title);
               mAuthorText.setText(authors);
               return;
           }
       }


       mTitleText.setText("No Results Found");
       mAuthorText.setText("");


   } catch (Exception e){
       mTitleText.setText("No Results Found");
       mAuthorText.setText("");
       e.printStackTrace();
   }
}
```

Tugas 3. Mengimplementasikan praktik terbaik UI

Anda sekarang memiliki aplikasi yang berfungsi dan menggunakan API Books untuk mengeksekusi penelusuran buku. Tetapi, ada beberapa hal yang tidak berjalan seperti yang diharapkan:

- Saat pengguna mengeklik **Search Books**, keyboard tidak muncul, dan tidak ada indikasi bagi pengguna bahwa kueri sebenarnya sedang dieksekusi.
- Jika tidak ada koneksi jaringan, atau bidang penelusuran kosong, aplikasi masih mencoba menanyakan API dan gagal tanpa memperbarui UI dengan benar.
- Jika Anda memutar layar selama kueri, AsyncTask akan terputus koneksinya dari Aktivitas, dan tidak dapat memperbarui UI dengan hasilnya.

Anda akan memperbaiki masalah ini pada bagian selanjutnya.

3.1 Menyembunyikan Keyboard dan Memperbarui TextView

Pengalaman pengguna penelusuran tidak intuitif. Ketika tombol ditekan, keyboard tetap terlihat dan kita tidak akan mengetahui bahwa kueri sedang berjalan. Salah satu dari solusinya adalah dengan secara terprogram menyembunyikan keyboard dan memperbarui salah satu TextView hasil untuk membaca "Loading..." saat kueri sedang dikerjakan. Untuk menggunakan solusi ini, Anda bisa:

1. Menambahkan kode berikut ke metode `searchBooks()` untuk menyembunyikan keyboard saat tombol ditekan:

```
InputMethodManager inputManager = (InputMethodManager)
                    getSystemService(Context.INPUT_METHOD_SERVICE);
inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
                    InputMethodManager.HIDE_NOT_ALWAYS);
```

2. Menambahkan satu baris kode di bawah panggilan untuk mengeksekusi tugas FetchBook yang mengubah TextView judul untuk menbaca "Loading..." dan menghapus TextView penulis.
3. Mengekstrak sumber daya String.

3.2 Mengelola status jaringan dan kasus bidang penelusuran kosong

Kapan pun aplikasi menggunakan jaringan, aplikasi itu perlu menangani kemungkinan koneksi jaringan tidak tersedia. Sebelum mencoba terhubung ke jaringan dalam AsyncTask atau AsyncTaskLoader, aplikasi harus memeriksa status koneksi jaringan.

1. Modifikasi metode `searchBooks()` untuk memeriksa kedua koneksi jaringan dan apakah ada teks dalam bidang penelusuran sebelum mengeksekusi tugas FetchBook.
2. Perbarui UI dalam kasus tidak ada koneksi internet atau tidak ada teks dalam bidang penelusuran. Tampilkan penyebab kesalahan dalam TextView.

Kode solusi:

```
public void searchBooks(View, view) {

   String queryString = mBookInput.getText().toString();

   InputMethodManager inputManager = (InputMethodManager)
           getSystemService(Context.INPUT_METHOD_SERVICE);
   inputManager.hideSoftInputFromWindow(getCurrentFocus().getWindowToken(),
           InputMethodManager.HIDE_NOT_ALWAYS);

   ConnectivityManager connMgr = (ConnectivityManager)
           getSystemService(Context.CONNECTIVITY_SERVICE);
   NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();

   if (networkInfo != null && networkInfo.isConnected() && queryString.length()!=0) {
       new FetchBook(mTitleText, mAuthorText).execute(queryString);
       mAuthorText.setText("");
       mTitleText.setText(R.string.loading);
   }

   else {
       if (queryString.length() == 0) {
           mAuthorText.setText("");
           mTitleText.setText("Please enter a search term");
       } else {
           mAuthorText.setText("");
           mTitleText.setText("Please check your network connection and try again.");
       }
   }
}
```

Tugas 4. Migrasi ke AsyncTaskLoader

Saat menggunakan AsyncTask, AsyncTask tidak bisa memperbarui UI jika perubahan konfigurasi terjadi saat tugas latar belakang sedang berjalan. Untuk mengatasi situasi ini, Android SDK menyediakan serangkaian kelas bernama loader yang didesain secara spesifik untuk memuat data ke dalam UI secara asinkron. Jika Anda menggunakan loader, Anda tidak perlu khawatir loader akan kehilangan kemampuannya untuk memperbarui UI dalam aktivitas yang awalnya membuatnya. Framework Loader tidak bekerja bagi Anda dengan mengasosiasikan ulang loader dengan Aktivitas yang tepat saat perangkat mengubah konfigurasinya. Ini berarti jika Anda memutar perangkat saat tugas sedang berjalan, hasilnya akan ditampilkan dengan benar dalam Aktivitas setelah data dikembalikan.

Dalam tugas ini Anda akan menggunakan loader spesifik bernama AsyncTaskLoader. AsyncTaskLoader adalah subkelas abstrak Loader dan menggunakan AsyncTask untuk memuat data di latar belakang dengan efisien.

Catatan: Saat menggunakan AsyncTask, Anda mengimplementasikan metode `onPostExecute()` dalam *AsyncTask* untuk menampilkan hasilnya di layar. Saat menggunakan AsyncTaskLoader, Anda mendefinisikan metode callback dalam aktivitas untuk menampilkan hasilnya.
Loader menyediakan banyak fungsionalitas tambahan, bukan hanya untuk menjalankan tugas dan menghubungkan kembali ke Aktivitas. Misalnya, Anda bisa melampirkan loader ke sumber data dan membuatnya secara otomatis memperbarui elemen UI saat data dasarnya berubah. Loader juga bisa diprogram untuk melanjutkan memuat jika terganggu.

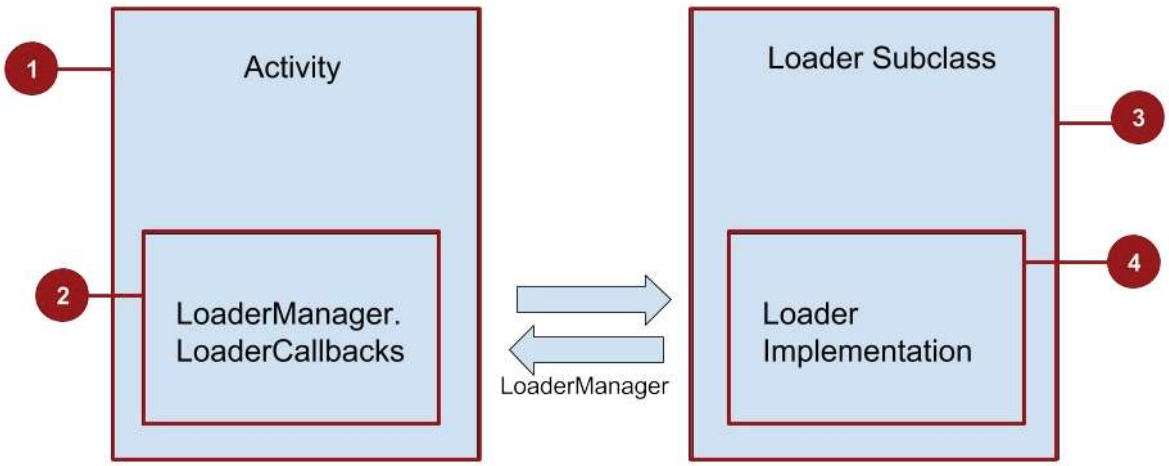
Mengapa harus menggunakan AsyncTask jika AsyncTaskLoader jauh lebih berguna? Jawabannya adalah tergantung situasi. Jika tugas latar belakang mungkin akan selesai sebelum perubahan konfigurasi terjadi, dan tidak perlu memperbarui UI, AsyncTask mungkin akan cukup. Framework Loader sebenarnya menggunakan AsyncTask di belakang layar untuk menjalankan fungsinya.

Prinsipnya, gunakan AsyncTaskLoader sebagai ganti AsyncTask jika pengguna mungkin akan memutar layar saat pekerjaan sedang berlangsung, atau jika UI perlu diperbarui saat pekerjaan selesai.

Dalam latihan ini Anda akan mempelajari cara menggunakan AsyncTaskLoader sebagai ganti AsyncTask untuk menjalankan kueri API Books. Anda akan mempelajari selengkapnya tentang penggunaan loader yang lain pada pelajaran berikutnya.

Mengimplementasikan Loader memerlukan komponen berikut:

- Kelas yang memperluas kelas Loader (dalam kasus ini, AsyncTaskLoader).
- Aktivitas yang mengimplementasikan kelas LoaderManager.LoaderCallbacks.
- Instance LoaderManager.



1. Aktivitas.
2. The LoaderManager.LoaderCallbacks.
3. Subkelas Loader.
4. Implementasi Loader.

LoaderManager secara otomatis memindahkan loader melalui siklus hidupnya tergantung status data dan Aktivitas. Misalnya, LoaderManager memanggil `onStartLoading()` saat loader diinisialisasi dan dimusnahkan saat Aktivitas dimusnahkan.

LoaderManager.LoaderCallbacks adalah serangkaian metode dalam Aktivitas yang dipanggil oleh LoaderManager saat loader dibuat, ketika data sudah selesai dimuat, dan saat loader disetel ulang. LoaderCallbacks bisa mengambil hasil tugas dan meneruskannya kembali ke UI Aktivitas.

Subkelas Loader berisi detail pemuatan data, biasanya menggantikan paling tidak `onStartLoading()`. Subkelas ini juga berisi fitur tambahan seperti mengamati sumber data untuk perubahan dan caching data secara lokal.

Subkelas Loader mengimplementasikan metode callback siklus hidup Loader seperti `onStartLoading()`, `onStopLoading()` dan `onReset()`. Subkelas loader juga berisi metode `forceLoad()` yang menginisialisasi pemuatan data. Metode ini tidak dipanggil secara otomatis saat loader dimulai karena beberapa penyiapan biasanya diperlukan sebelum pemuatan dilakukan. Implementasi paling sederhana akan memanggil `forceLoad()` dalam `onStartLoading()` yang akan menyebabkan pemuatan setiap LoaderManager memulai Loader.

4.1 Membuat AsyncTaskLoader

1. Salin proyek WhoWroteIt, untuk mempertahankan hasil dari praktik sebelumnya. Ganti nama proyek yang disalin menjadi **WhoWroteItLoader**.
2. Buat kelas baru dalam direktori Java bernama BookLoader.
3. Buat agar kelas BookLoader memperluas AsyncTaskLoader dengan tipe berparameter. .
4. Pastikan Anda mengimpor loader dari Pustaka Dukungan v4.
5. Implementasikan metode yang diperlukan (`loadInBackground()`). Perhatikan kemiripan antara metode ini dan metode `doInBackground()` awal dengan AsyncTask.
6. Buat konstruktor untuk kelas baru Anda. Dalam Android Studio, kemungkinan deklarasi kelas akan tetap digarisbawahi dengan warna merah karena konstruktor tidak cocok dengan implementasi superkelas. Dengan adanya kursor pada baris deklarasi kelas, tekan **Alt + Enter** (**Option + Enter** di Mac) dan pilih **Create constructor matching super**. Ini akan membuat konstruktor dengan konteksnya sebagai parameter.

Define onStartLoading()

1. Tekan **Ctrl + O** untuk membuka metode Override, dan pilih **onStartLoading**. Metode ini dipanggil oleh sistem saat Anda memulai loader.
2. Loader tidak akan mulai memuat data sampai Anda memanggil metode `forceLoad()`. Di dalam stub metode `onStartLoading()`, panggil `forceLoad()` untuk memulai metode `loadInBackground()` saat Loader dibuat.

Definisikan loadInBackground()

1. Buat variabel anggota `mQueryString` yang akan menampung kueri `String`, dan modifikasi konstruktor untuk mengambil `String` sebagai argumen dan menetapkannya ke variabel `mQueryString`.
2. Dalam metode `loadInBackground()`, panggil metode `getBookInfo()` yang meneruskan `mQueryString`, dan kembalikan hasilnya untuk mengunduh informasi dari API Books:

```
@Override
public String loadInBackground() {
   return NetworkUtils.getBookInfo(mQueryString);
}
```

4.2 Memodifikasi MainActivity

Sekarang Anda harus mengimplementasikan Callback Loader dalam MainActivity untuk menangani hasil dari metode `loadInBackground()` AsyncTaskLoader.

1. Tambahkan implementasi `LoaderManager.LoaderCallbacks` ke deklarasi kelas Aktivitas Utama, yang berparameter dengan tipe `String`:

```
public class MainActivity extends AppCompatActivity
               implements LoaderManager.LoaderCallbacks<String>{
```

2. Implementasikan metode yang diperlukan: `onCreateLoader(), onLoadFinished(), onLoaderReset()`. Letakkan kursor teks pada baris tanda tangan kelas dan masukkan **Alt + Enter** (**Option + Enter** di Mac). Pastikan semua metode dipilih.

   Catatan: Jika impor untuk `Loader` dan `LoaderManager` dalam MainActivity tidak cocok dengan impor untuk `AsyncTaskLoader` untuk kelas `BookLoader`, Anda akan mendapatkan beberapa kesalahan mengetik dalam callback.

Pastikan semua impor berasal dari Pustaka Dukungan Android.

Loader menggunakan kelas Bundle untuk meneruskan informasi dari aktivitas memanggil ke LoaderCallbacks. Anda bisa menambahkan data primitif ke bundel dengan metode `putType()` yang tepat.

Untuk memulai loader, Anda memiliki dua opsi:

- `initLoader()` : Metode ini membuat loader baru jika belum ada dan bergerak dalam Bundel argumen. Jika loader ada, Aktivitas memanggil akan dikaitkan kembali dengannya tanpa memperbarui Bundel.
- `restartLoader()` : Metode ini sama dengan initLoader() kecuali jika metode ini menemukan loader yang sudah ada, metode ini akan memusnahkan dan membuat ulang loader dengan Bundel yang baru.

Kedua metode didefinisikan dalam LoaderManager, yang mengelola semua instance Loader yang digunakan dalam Aktivitas (atau Fragmen). Setiap Aktivitas memiliki satu instance LoaderManager yang bertanggung jawab terhadap siklus hidup dan Loader yang dikelolanya.

Saat ini, FetchBook AsyncTask dipicu saat pengguna menekan tombol tersebut. Anda perlu memulai loader dengan Bundel baru setiap kali tombol ditekan. Untuk melakukannya, Anda perlu mengedit metode onClick untuk tombol itu.

1. Dalam metode `searchBooks()` , yang merupakan metode onClick untuk tombol tersebut, ganti panggilan untuk mengeksekusi tugas FetchBook dengan panggilan untuk `restartLoader()` , yang meneruskan string kueri yang Anda dapatkan dari EditText dalam Bundel:

   ```
   Bundle queryBundle = new Bundle();
   queryBundle.putString("queryString", queryString);
   getSupportLoaderManager().restartLoader(0, queryBundle,this);
   ```

   Metode `restartLoader()` mengambil tiga argumen:

   - Loader `id` (berguna jika Anda mengimplementasikan lebih dari satu loader dalam aktivitas).
   - Argumen `Bundle` (tempat data yang diperlukan oleh loader disimpan).
   - Instance LoaderCallbacks yang Anda implementasikan dalam aktivitas. Jika ingin loader membawa hasil ke MainActivity, tetapkan `this` sebagai argumen ketiga.
2. Periksa metode Override dalam kelas LoaderCallbacks. Metode ini adalah:

   - `onCreateLoader()` : Dipanggil saat Anda membuat instance Loader.
   - `onLoadFinished()` : Dipanggil ketika tugas loader sudah selesai. Ini adalah tempat di mana Anda menambahkan kode untuk memperbarui UI dengan hasilnya.
   - `onLoaderReset()` : Menghapus sumber daya yang tersisa.

Anda hanya akan mendefinisikan dua metode pertama, karena model data saat ini adalah string sederhana yang tidak membutuhkan perhatian ekstra saat loader disetel ulang.

Implementasikan onCreateLoader()

1. Dalam `onCreateLoader()` , kembalikan instance kelas BookLoader, meneruskan `queryString` yang didapatkan dari Bundel argumen:

   ```
   return new BookLoader(this, args.getString("queryString"));
   ```

Implementasikan onLoadFinished()

1. Perbarui `onLoadFinished()` untuk memproses hasil, yang merupakan respons String JSON mentah dari API Books.
   i. Salin kode dari `onPostExecute()` dalam kelas FetchBook ke to `onLoadFinished()` dalam MainActivity, dengan mengecualikan panggilan ke `super.onPostExecute()` .
   ii. Ganti argumen ke konstruktor JSONObject dengan data `String` yang diteruskan.
2. Jalankan aplikasi Anda.

Anda seharusnya memiliki fungsionalitas yang sama seperti sebelumnya, hanya saja sekarang di dalam Loader! Satu hal yang masih tidak berfungsi. Saat perangkat diputar, data `View` hilang. Ini karena saat Aktivitas dibuat (atau dibuat ulang), Aktivitas tidak tahu bahwa ada loader yang sedang berjalan. Metode `initLoader()` dibutuhkan dalam `onCreate()` dari MainActivity untuk menghubungkan ulang loader.

3. Tambahkan kode berikut di `onCreate()` untuk menghubungkan ulang ke Loader jika sudah ada:

```
if(getSupportLoaderManager().getLoader(0)!=null){
   getSupportLoaderManager().initLoader(0,null,this);
}
```

Catatan: Jika loader ada, inisialisasikan loader. Anda hanya perlu mengaitkan kembali loader ke Aktivitas jika kueri sudah dieksekusi. Dalam status awal aplikasi, data tidak dimuat sehingga tidak ada yang perlu dipertahankan.

4. Jalankan aplikasi lagi dan putar perangkat. LoaderManager sekarang mempertahankan data di semua konfigurasi perangkat!

5. Hapus kelas FetchBook karena sudah tidak digunakan lagi.

Kode solusi

Proyek Android Studio: WhoWroteItLoader

Tantangan penyusunan kode

Catatan: Semua tantangan penyusunan kode opsional dan bukan prasyarat untuk pelajaran berikutnya.

Tantangan 1: Jelajahi API spesifik yang Anda gunakan dalam detail yang lebih lengkap dan temukan parameter penelusuran yang membatasi hasil ke buku yang dapat diunduh dalam format epub. Tambahkan parameter ke permintaan Anda dan lihat hasilnya.

Tantangan 2: Respons dari API Books berisi hasil sebanyak yang Anda setel dengan parameter `maxResults`, namun dalam implementasi ini Anda sudah mengembalikan hasil Buku valid yang pertama. Modifikasi aplikasi Anda agar data ditampilkan dalam RecyclerView yang memiliki `maxResults` entri.

Rangkuman

- Tugas yang menghubungkan ke jaringan, atau memerlukan waktu lebih untuk diproses seharusnya tidak dieksekusi pada thread UI.
  - Waktu Proses Android biasanya memiliki default StrictMode yang akan memunculkan pengecualian jika Anda mengupayakan konektivitas jaringan atau akses file pada thread UI.
- Google API Explorer adalah alat yang membantu Anda menjelajahi berbagai API Google secara interaktif.
  - API Penelusuran Books adalah serangkaian API RESTful untuk mengakses Google Books secara terprogram.
  - Permintaan API ke Google Books bentuknya adalah URL.
  - Respons ke permintaan API tersebut mengembalikan string JSON.
- Gunakan `getText()` untuk mengambil teks dari tampilan EditText. Ini dapat dikonversikan ke dalam String sederhana menggunakan `toString()`.
- Kelas URI memiliki metode bantu, `Uri.buildUpon()` yang mengembalikan URI.Builder yang dapat digunakan untuk membuat string URI.
- AsyncTask adalah kelas yang mengizinkan Anda menjalankan tugas di latar belakang, secara asinkron, sebagai ganti pada thread UI.
  - AsyncTask dapat dimulai melalui `execute()`.
  - AsyncTask tidak dapat memperbarui UI jika aktivitas yang dikontrolnya berhenti (seperti perubahan konfigurasi

pada perangkat).
    - AsyncTask harus dijadikan subkelas agar bisa digunakan. Subkelas akan mengganti paling tidak satu metode doInBackground(Params) , dan paling sering mengganti yang kedua onPostExecute(Result) juga.
- Setiap kali dieksekusi, AsyncTask melalui 4 langkah:

    1. onPreExecute(). Dipanggil pada thread UI sebelum tugas dieksekusi. Langkah ini biasanya digunakan untuk menyiapkan tugas.
    2. doInBackground(Params). Dipanggil pada thread latar belakang segera setelah onPreExecute() selesai mengeksekusi. Langkah ini digunakan untuk melakukan penghitungan latar belakang yang bisa memakan waktu lama.
    3. onProgressUpdate(Progress). Dipanggil pada thread UI setelah Anda memanggil doInBackground ke publishProgress(Progress).
    4. onPostExecute(Result). Dipanggil pada thread UI setelah penghitungan latar belakang selesai. Hasil penghitungan latar belakang diteruskan ke metode ini sebagai parameter.
- AsyncTaskLoader adalah Loader yang setara dengan AsyncTask. Loader ini menyediakan metode, `loadInBackground()` , yang berjalan pada thread terpisah dan yang hasilnya secara otomatis dibawa ke thread UI (ke callback LoaderManager `onLoadFinished()`).
- Anda harus mengonfigurasi izin jaringan dalam file manifes Android untuk terhubung ke internet:

```
<uses-permission android:name="android.permission.INTERNET">
```

- Gunakan kelas JSON Java bawaan (`JSONObject` dan `JSONArray`) untuk membuat dan melakukan pada parsing string JSON.

- Loader mengizinkan pemuatan data dalam Aktivitas secara asinkron.
    - Loader dapat digunakan untuk membangun ulang komunikasi ke UI saat sebuah Aktivitas dihentikan sebelum tugas selesai (misalnya karena perangkat diputar).
    - AsyncTaskLoader adalah Loader yang menggunakan kelas helper AsyncTask di belakang layar untuk melakukan pekerjaan di latar belakang, di luar thread utama.
    - Loader dikelola oleh LoaderManager; satu atau beberapa Loader dapat ditetapkan dan dikelola oleh satu LoadManager.
    - LoaderManager mengizinkan Anda untuk menghubungkan Aktivitas yang baru dibuat dengan Loader menggunakan `getSupportLoaderManager().initLoader()` .

Konsep terkait

Dokumentasi konsep terkait ada di Dasar-Dasar Developer Android: Konsep.

- Terhubung ke Internet dengan AsyncTask dan AsyncTaskLoader

Ketahui selengkapnya

Dokumentasi Developer Android

Panduan

- Terhubung ke Jaringan
- Mengelola Status Jaringan
- Loader

Referensi

- AsyncTask
- AsyncTaskLoader