Rakamin Academy

schoters

Virtual Internship Experience

Optimizing Source Code

- Membuat dokumentasi kode
- Melakukan optimisasi kode

1. Membuat dokumentasi kode

Kode yang Anda tulis bukan milik Anda, dan dapat dipastikan bahwa selama siklus hidup proyek, pengembang lain perlu memodifikasinya atau menggunakannya. Jangan menjadi mangsa egoisme pengembang stereotip: "Kode saya sangat mudah. Bukan salahku jika kamu tidak memahaminya." Sebaliknya, ingatlah bahwa pada titik tertentu Anda akan menjadi pengembang baru di tim lagi, dan tulis dokumentasi yang ingin Anda baca dalam situasi itu.

Jika Anda berada di kamp pemrograman ekstrim, maka tanggapan Anda adalah bahwa "Kode adalah dokumentasi." Sampai taraf tertentu, kami setuju. Tidak ada yang bisa terlalu menekankan pentingnya kelas, metode, dan variabel yang disebutkan dengan baik. Kami telah melihat (banyak yang membuat kami cemas!) variabel dalam kode tertulis "profesional" yang dinamai hal-hal seperti . Kami tidak akan pernah mendapatkan kembali waktu yang dihabiskan setelah sepotong spaghetti itu. Jadi ya, waktu dan usaha yang dihabiskan untuk membuat nama baik dan menulis kode bersih sangat berharga.

Prinsip lain dari pemrograman ekstrim adalah bahwa komunikasi tim yang dekat membuat dokumentasi tidak perlu. Sementara sistem sedang dibuat, itu mungkin benar. Tetapi kemungkinannya adalah bahwa suatu hari, kode harus dipertahankan oleh seseorang yang tidak berada di tim asli. Daripada membuat orang itu meluangkan waktu untuk membaca kode bersih Anda, luangkan beberapa menit untuk dokumentasi untuk meringkas apa yang dilakukan metode Anda yang disebutkan dengan baik.

Memilih alat dokumentasi

Standar di Java adalah Javadocs. Jika Anda termasuk orang yang berpikir Javadocs default terlihat jelek, perlu diingat bahwa Anda dapat membuat doclet XHTML khusus untuk memberikan dokumentasi yang tampan dan profesional. Untuk bahasa lain, ada alat serupa, seperti NDoc untuk C# dan DOC ++ untuk C dan C ++. Ada juga Doxygen, yang memiliki integrasi SonarQube yang bagus

(kami akan membahasnya di bagian "Plugin Terkait") dan dapat digunakan untuk beberapa bahasa, termasuk Java, C, C ++, Python, dan PHP.

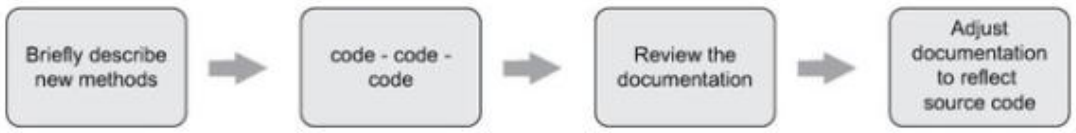
Anda bahkan dapat menggunakan kombinasi alat untuk hasil terbaik dan menyesuaikannya agar sesuai dengan kebutuhan Anda. Secara umum, mereka semua akan memberi Anda cara yang cerdas dan efektif untuk menghasilkan dokumentasi kode sumber Anda berdasarkan komentar yang diberikan.

Kapan harus mendokumentasikan

Pertama, dokumentasikan seperti kode Anda. Sebelum Anda memulai kelas baru, tuliskan beberapa komentar yang menjelaskan tujuannya. Hal yang sama berlaku untuk metode / fungsi. Jelaskan secara singkat apa yang diharapkan mereka lakukan.

Setelah menyelesaikan kode, tinjau dokumentasi dan perbarui untuk mencerminkan perubahan apa pun yang mungkin Anda buat Ikuti langkah yang sama kapan pun Anda perlu memodifikasi sepotong kode.

Gambar 5.5. Proses dokumentasi sederhana



Bagian sumber mana yang akan mendokumentasikan

Setiap kelas publik dan masing-masing metode dan properti publiknya adalah kandidat untuk dokumentasi. Tetapi tidak perlu mendokumentasikan setter dan getter atau metode lain yang sangat sederhana sehingga tanda tangan itu sendiri adalah penjelasannya. Cuplikan berikut menunjukkan salah satu anti-pola dokumentasi berlebihan favorit kami:

```
private String name;
/**
* Returns the name
* @return name
```

```
**/
Public String getName()
```

Dalam hal ini, Javadoc tidak memberikan apa pun yang tidak dapat diperoleh pembaca dari tanda tangan metode itu sendiri — yang berarti itu sepenuhnya berlebihan dan, seperti duplikasi lainnya, harus dihindari. Dokumentasi yang berlebihan, seperti kode redundan, perlu dipertahankan tetapi biasanya tidak mendapat perhatian. Sebaliknya, itu hanya menambahkan kebisingan ke kode sumber Anda. Atau lebih buruk lagi, itu berhenti mencerminkan tujuan sebenarnya dari kode karena tidak terus up to date, dan menambahkan kebingungan bukan kejelasan.

Informasi apa yang harus disertakan

Selain menjelaskan tujuan setiap kelas atau metode, ada baiknya anda menyertakan deskripsi parameter input dan output serta pengecualian yang mungkin dilemparkan. Bila sesuai, Anda mungkin juga ingin menyertakan hal-hal seperti versi API yang memperkenalkan fitur tertentu. Sertakan semua yang menurut Anda penting, dan perlu diingat bahwa dokumentasi harus cukup lengkap sehingga pembaca tidak perlu meminta klarifikasi. IDE populer seperti Eclipse dan NetBeans menawarkan banyak cara untuk memfasilitasi proses ini, termasuk cangkang dokumentasi yang kami sebutkan sebelumnya. Mereka mengisi informasi dasar untuk Anda dan membiarkan Anda fokus pada pekerjaan nyata.

Apa yang harus Anda hindari sama pentingnya dengan apa yang Anda dokumentasikan. Cobalah untuk tidak mengomentari baris kode asli; hapus mereka, sebagai gantinya. Jika Anda membutuhkannya kembali, Anda selalu dapat kembali ke versi file sebelumnya dari sistem kontrol sumber Anda. Jadi mengapa menambahkan noise ke kode Anda?

Apakah akan menghindari komentar dalam tubuh metode atau fungsi dapat diperdebatkan. Banyak orang berpendapat bahwa jika kode Anda membutuhkan komentar untuk dapat dimengerti, yang sebenarnya dibutuhkan adalah refactoring. Terlepas dari apakah Anda setuju, pastikan Anda setuju (atau

tidak setuju) sebagai sebuah tim, sehingga semua orang berada di halaman yang sama.

2. **Melakukan optimisasi kode**

Saat Anda menggunakan Android Studio 3.4 atau plugin Android Gradle 3.4.0 dan yang lebih baru, R8 adalah compiler default yang mengubah bytecode Java project Anda menjadi format DEX yang berjalan pada platform Android. Namun, saat Anda membuat project baru menggunakan Android Studio, penyusutan, obfuscation, dan pengoptimalan kode tidak diaktifkan secara default. Hal itu karena pengoptimalan waktu kompilasi ini meningkatkan waktu build project dan dapat memunculkan bug jika Anda tidak menyesuaikan kode yang perlu dipertahankan secara memadai.

Jadi, sebaiknya aktifkan tugas-tugas waktu kompilasi ini saat membuat versi final aplikasi yang Anda uji sebelum dipublikasikan. Untuk mengaktifkan penyusutan, obfuscation, dan pengoptimalan, sertakan yang berikut dalam file build.gradle level project Anda.

```
android {
    buildTypes {
        getByName("release") {
            // Enables code shrinking, obfuscation, and optimization for only
            // your project's release build type.
            isMinifyEnabled = true

            // Enables resource shrinking, which is performed by the
            // Android Gradle plugin.
            isShrinkResources = true

            // Includes the default ProGuard rules files that are packaged with
            // the Android Gradle plugin. To learn more, go to the section about
            // R8 configuration files.
            proguardFiles(
                getDefaultProguardFile("proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        }
    }
    ...
}
```

Gambar 1. build.gradle untuk aktifkan R8

R8 menggunakan file aturan ProGuard untuk mengubah perilaku defaultnya dan memahami struktur aplikasi Anda dengan lebih baik, seperti class yang berfungsi sebagai titik masuk ke dalam kode aplikasi Anda. Meskipun Anda dapat mengubah beberapa file aturan ini, beberapa aturan dapat otomatis dibuat oleh fitur waktu kompilasi, seperti AAPT2, atau diwarisi dari dependensi library aplikasi Anda.

Untuk menyusutkan ukuran aplikasi Anda lebih lanjut, R8 akan memeriksa kode Anda di tingkat yang lebih dalam untuk menghapus kode yang tidak terpakai atau, jika mungkin, menulis ulang kode Anda agar tidak terlalu panjang. Berikut ini beberapa contoh pengoptimalan tersebut:

Jika kode Anda tidak pernah mengambil cabang else {} untuk pernyataan if/else tertentu, R8 mungkin akan menghapus kode untuk cabang else {}.
Jika kode Anda memanggil metode hanya di satu tempat, R8 mungkin akan menghapus metode itu dan menyisipkannya secara inline di situs panggilan tunggal.

Jika R8 menentukan bahwa sebuah class hanya memiliki satu subclass unik, dan class itu sendiri tidak dipakai (misalnya, class dasar abstrak hanya digunakan oleh satu class implementasi konkret), maka R8 dapat menggabungkan dua class dan menghapus class dari aplikasi.

R8 tidak memungkinkan Anda menonaktifkan atau mengaktifkan pengoptimalan terpisah, atau mengubah perilaku pengoptimalan. Bahkan, R8 mengabaikan semua aturan ProGuard yang mencoba mengubah pengoptimalan default, seperti -optimizations dan - optimizationpasses. Pembatasan ini penting karena, seiring berkembangnya R8, mempertahankan perilaku standar untuk pengoptimalan akan membantu tim Android Studio memecahkan masalah dan menyelesaikan masalah apa pun yang mungkin Anda temui dengan mudah.

Menyesuaikan kode yang perlu dipertahankan

Dalam sebagian besar situasi, file aturan ProGuard default (proguard-android-optimize.txt) sudah cukup bagi R8 untuk hanya menghapus kode yang tidak digunakan. Namun, beberapa situasi sulit dianalisis dengan benar oleh R8 dan akibatnya R8 mungkin menghapus kode yang sebenarnya diperlukan aplikasi Anda. Beberapa contoh di mana R8 mungkin salah menghapus kode antara lain:

- Jika aplikasi Anda memanggil metode dari Java Native Interface (JNI)
- Jika aplikasi Anda mencari kode saat runtime (seperti dengan refleksi)

Pengujian aplikasi akan menunjukkan error yang disebabkan oleh kode yang salah dihapus, tetapi Anda juga dapat memeriksa kode apa yang telah dihapus dengan membuat laporan kode yang dihapus.

Untuk memperbaiki error dan memaksa R8 agar mempertahankan kode tertentu, tambahkan baris -keep dalam file aturan ProGuard. Contoh:

```
-keep public class MyClass
```

Atau, Anda dapat menambahkan anotasi @Keep ke kode yang ingin dipertahankan. Menambahkan @Keep pada suatu class akan mempertahankan seluruh class tersebut apa adanya. Menambahkannya pada suatu metode atau kolom akan mempertahankan metode/kolom tersebut (dan namanya) serta nama class-nya tetap utuh. Perhatikan bahwa anotasi ini hanya tersedia saat menggunakan AndroidX Annotations Library dan saat Anda menyertakan file aturan ProGuard yang dikemas dengan plugin Android Gradle, seperti dijelaskan di bagian tentang cara mengaktifkan penyingkatan.

Ada banyak pertimbangan yang harus Anda buat saat menggunakan opsi -keep; untuk informasi selengkapnya tentang menyesuaikan file aturan, baca Panduan ProGuard. Bagian Pemecahan Masalah dalam panduan tersebut menjelaskan masalah umum lain yang mungkin Anda alami jika kode dihapus.

Mengaktifkan pengoptimalan yang lebih agresif

R8 menyertakan serangkaian pengoptimalan tambahan yang tidak diaktifkan secara default. Anda dapat mengaktifkan pengoptimalan tambahan ini dengan menyertakan baris berikut dalam file gradle.properties project Anda:

```
android.enableR8.fullMode=true
```

Karena pengoptimalan tambahan membuat R8 berperilaku berbeda dengan ProGuard, Anda mungkin harus menyertakan aturan ProGuard tambahan guna menghindari masalah runtime. Sebagai contoh, misalkan kode Anda mereferensikan class melalui Java Reflection API. Secara default, R8 akan berasumsi bahwa Anda bermaksud memeriksa dan memanipulasi objek dari class tersebut pada waktu proses—meski sebenarnya kode Anda tidak melakukannya—dan secara otomatis mempertahankan class tersebut dan penginisialisasi statisnya.

Namun, saat Anda menggunakan "mode penuh", R8 tidak membuat asumsi ini dan, jika R8 menyatakan bahwa kode Anda tidak pernah menggunakan class itu selama waktu proses, maka R8 akan menghapus class tersebut dari DEX akhir aplikasi Anda. Artinya, jika ingin mempertahankan class ini dan penginisialisasi statisnya, Anda harus menyertakan aturan keep dalam file aturan Anda untuk melakukan hal itu.

Membuat laporan kode yang dihapus (atau dipertahankan)

Untuk membantu Anda memecahkan masalah R8 tertentu, sebaiknya lihat laporan semua kode yang dihapus R8 dari aplikasi Anda. Untuk setiap modul yang ingin Anda peroleh laporannya, tambahkan -printusage <output-dir>/usage.txt ke file aturan kustom Anda. Saat Anda mengaktifkan R8 dan membuat aplikasi, R8 akan meng-output laporan yang berisi jalur dan nama file yang Anda tentukan. Laporan kode yang dihapus terlihat mirip dengan berikut ini:

```
androidx.drawerlayout.R$attr
androidx.vectordrawable.R
androidx.appcompat.app.AppCompatDelegateImpl
    public void setSupportActionBar(androidx.appcompat.widget.Toolbar)
    public boolean hasWindowFeature(int)
    public void setHandleNativeActionModesEnabled(boolean)
    android.view.ViewGroup getSubDecor()
    public void setLocalNightMode(int)
    final androidx.appcompat.app.AppCompatDelegateImpl$AutoNightModeManager getAutoNightModeManager()
    public final androidx.appcompat.app.ActionBarDrawerToggle$Delegate getDrawerToggleDelegate()
    private static final boolean DEBUG
    private static final java.lang.String KEY_LOCAL_NIGHT_MODE
    static final java.lang.String EXCEPTION_HANDLER_MESSAGE_SUFFIX
...
```

Gambar 2. Laporan kode yang dihapus

Jika Anda ingin melihat laporan titik entri yang ditentukan R8 dari aturan keep project Anda, sertakan -printseeds <output-dir>/seeds.txt di file aturan kustom Anda. Saat Anda mengaktifkan R8 dan membuat aplikasi, R8 akan meng-output laporan yang berisi jalur dan nama file yang Anda tentukan. Laporan titik masuk yang dipertahankan terlihat mirip dengan berikut ini:

```
com.example.myapplication.MainActivity
androidx.appcompat.R$layout: int abc_action_menu_item_layout
androidx.appcompat.R$attr: int activityChooserViewStyle
androidx.appcompat.R$styleable: int MenuItem_android_id
androidx.appcompat.R$styleable: int[] CoordinatorLayout_Layout
androidx.lifecycle.FullLifecycleObserverAdapter
...
```

Memecahkan masalah penyusutan resource

Saat Anda menyusutkan resource, jendela Build akan menampilkan ringkasan resource yang dihapus dari aplikasi. (Anda harus mengklik Toggle view di sisi kiri jendela terlebih dahulu untuk menampilkan output teks mendetail dari Gradle.) Contoh:

```
:android:shrinkDebugResources
Removed unused resources: Binary resource data reduced from
2570KB to 1711KB: Removed 33%
:android:validateDebugSigning
```

Gradle juga membuat file diagnostik bernama resources.txt di <module-name>/build/outputs/mapping/release/ (folder yang sama dengan file output ProGuard). File ini menyertakan detail seperti resource mana yang mereferensikan resource lain, dan resource mana yang digunakan atau dihapus.

Daftar Pustaka

https://developer.android.com/studio/build/shrink-code
https://livebook.manning.com/book/sonarqube-in-action/chapter-5/1