R8 vs Proguard in Android

Proguard sounds familiar to all Android developers. We use it for reducing the size, improving the performance of the application by shrinking unused resources.

Google released R8 as a replacement of Proguard to help developers shrink the code with the better-generated output (APK). They are considered much faster compared to Proguard.

In this blog, we are going to talk about,

- What is R8?
- How does R8 shrinking work?

- R8's comparison with Proguard

What is R8?

R8 is a tool that converts our java byte code into an optimized dex code. It iterates through the whole application and then it optimizes like removing unused classes, methods, etc. It runs on the compile time. It helps us to reduce the size of the build and to make our app to be more secure. R8 uses Proguard rules to modify its default behavior.

How does R8 shrinking work?

While optimizing the code, R8 reduces the code of our application, and then APK size is reduced.

To reduce the APK size, we have three different techniques:

1. **Shrinking or Tree Shaking:** Shrinking is the process of removal of unreachable code from our Android project. R8 performs some static analysis to get rid of unreachable code and removes the uninstantiated object.
2. **Optimization:** This is used to optimize the code for size. It involves dead code removal, unused argument removal, selective inlining, class merging, etc.
3. **Identifier Renaming:** In this process, we obfuscate the class name and other variable names. For example, if the name of the class is "**MainActivity**", then it will be obfuscated to "**a**" or something else but smaller in size.

How to enable R8 Shrinking in your app?

R8 is present by default in our application but to enable R8 shrinking in our application, set the `minifyEnabled` to `true` in our app's main *`build.gradle`* file.

```
android {
    ...
    buildTypes {
        release {
            minifyEnabled true
        }
    }
}
```

Now, let's compare both R8 and Proguard.

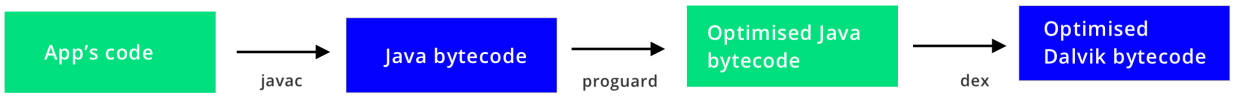
R8's comparison with Proguard

So, let's now compare both R8 and Proguard both and see how it fares,

- With the Android app using Gradle plugin above 3.4.0 or more the project uses **R8** by default and no longer uses the **Proguard** to perform optimizations. But, it uses Proguard rules only.
- R8 effectively inlines the container classes and removes unused class, fields, and methods. Proguard reduces the app size by 8.5% and compared to R8 which reduces the code by 10%.
- R8 has more Kotlin support compared to Proguard.
- R8 gives better outputs than Proguard, and to do so faster than Proguard does, thereby reducing overall build time.

So, let's now compare how both Proguard and R8 performs.

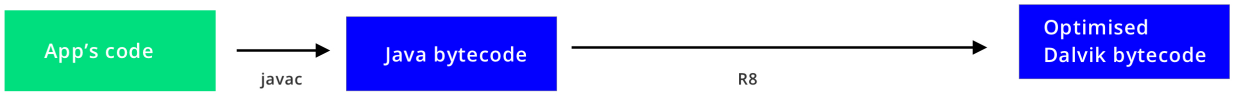
Proguard.



While using Proguard, the Applications code is converted to Java bytecode by the Java compiler. After the conversion, it is then optimized by Proguard using the rules which we have written. Then dex converts it to optimized Dalvik byte code.

This is roughly a 4 step process to convert it to Dalvik bytecode.

R8.



While using R8, first the app's code is converted to Java bytecode by the java compiler and then using R8 directly, it converts the java byte code in Dalvik bytecode.

By using R8, it directly reduces the steps of conversion of Java bytecode to Dalvik Bytecode from 2 to 1.

- Proguard applies 520 peephole optimizations compared to R8 which is very less. Peephole optimizations are the optimizations that are performed on a set of compiler-generated code to improve the performance of the code by making it shorter and faster.
- In both Proguard and R8, we have to handle the reflection by writing the custom configuration.
- R8 is faster compared to Proguard in the execution of converting the code.

Optimization Comparison between Proguard and R8.

Let us discuss a few features supported by both Proguard and R8.

For example, Proguard and R8 both make the methods private in the code. Both of them also remove unused class, fields, or even methods in the project which are not in use. Both of them support the simplification of Enum types. They both inline methods, merge codes, etc.

Proguard also makes the classes final whereas R8 is not able to do it. But comparing R8 which is highly supported by Kotlin, optimizes the Kotlin construct which is not possible with Proguard.

Now, If you also want to enable aggressive optimization in R8 and reduce the size more, enable the following in *gradle.properties,*

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

Conclusion

With R8 coming as a default compile-time optimizer, it reduces the size of the app.

Happy learning.

Team MindOrks :)