Topic for discussion:

MacroBenchmark library for measuring app performance.

Workshop hosted by Juli
(Android Engineer)

What is Microbench mark?

The Jetpack **Microbenchmark** library allows you to quickly benchmark your Android native code (Kotlin or Java) from within Android Studio. The library handles warmup, measures your code performance and allocation counts, and outputs benchmarking results to both the Android Studio console and a JSON file with more detail.

Use the Macrobenchmark library for testing larger use-cases of your application, including application startup and complex UI manipulations, such as scrolling a **RecyclerView** or running animations.

We recommend to **profile your code** before writing a benchmark. This helps you find expensive operations that are worth optimizing. It can also expose why the operations are slow by showing what is happening while they run. These could be running on a low-priority thread, sleeping due to disk access, or unexpectedly calling into an expensive function, like bitmap decoding.

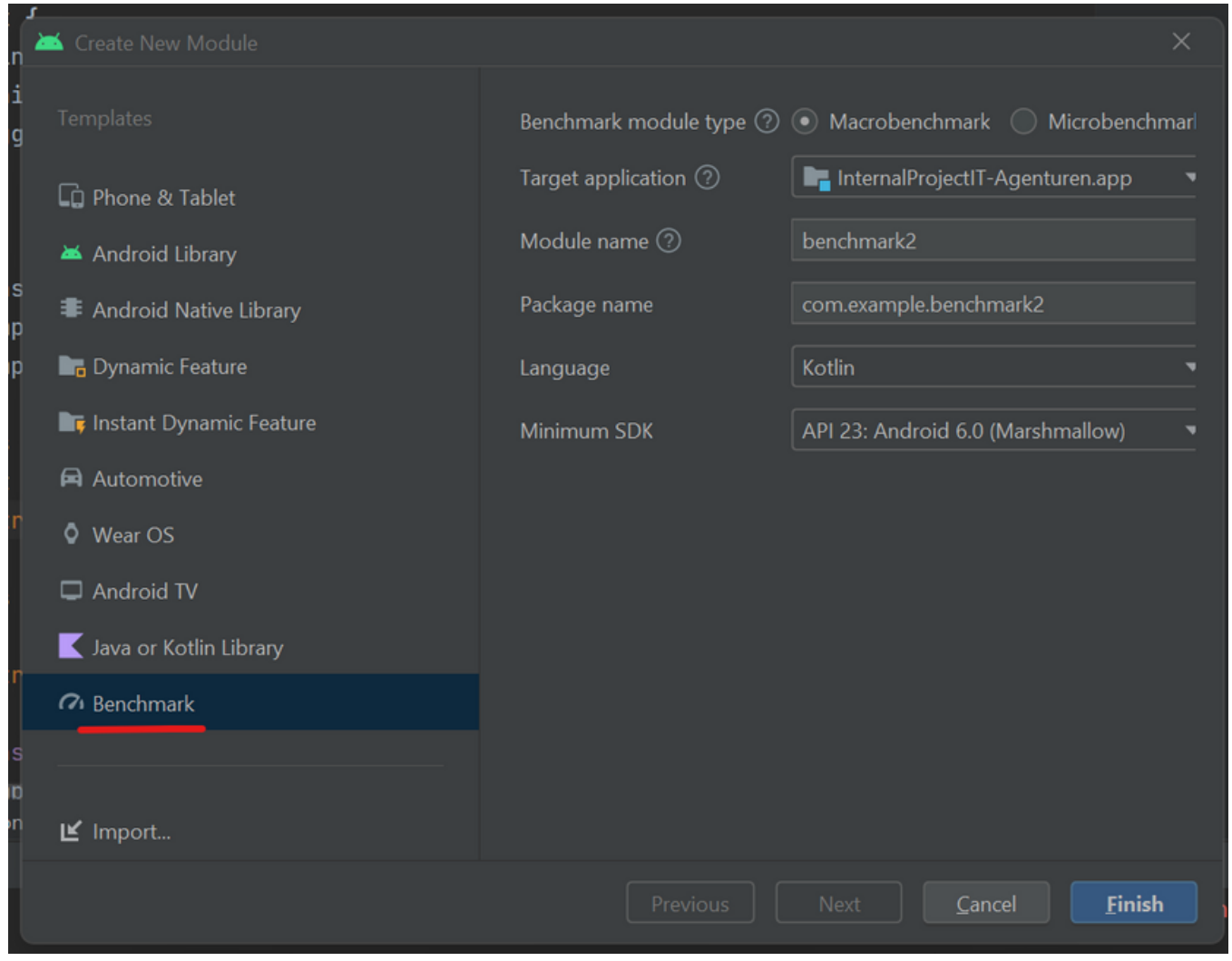
Project setup

- Right-click your project or module in the **Project** panel in Android Studio and click **New > Module**.

- Select **Benchmark** from the **Templates** pane.

- You can customize the target application (the app to be benchmarked), as well as package and module name for the new macrobenchmark module.

- Click **Finish**.



Set up the application

To benchmark an app (called the target of the macro benchmark), that app must be **profileable**, which enables reading detailed trace information. Enable this in the **<application>** tag of the app's **AndroidManifest.xml**

```
<application
    android:name=".MyApp"
    android:allowBackup="true"
    android:hardwareAccelerated="false"
    android:icon="@mipmap/ic_launcher"
    android:label="InternalProjectIT-Agenturen"
    android:largeHeap="true"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.InternalProjectITAgenturen">
    <profileable
        android:shell="true"
        tools:targetApi="q" />
```

Configure the benchmarked app as close to the release version (or production) as possible. Set it up as non-debuggable and preferably with minification on, which improves performance. You typically do this by creating a copy of the release variant, which performs the same, but is signed locally with debug keys. Alternatively you can use **initWith** to instruct Gradle to do it for you

```
buildTypes {
    release {
        minifyEnabled false
        proguardFiles getDefaultProguardFile('
    }
    benchmark {
        signingConfig signingConfigs.debug
        matchingFallbacks = ['release']
        debuggable false
    }
```

Create a macrobenchmark class

Benchmark testing is provided through the MacrobenchmarkRule JUnit4 rule API in the Macrobenchmark library. It contains a measureRepeated method which allows you to specify various conditions on how the target application should be run and benchmarked.

You need to at least specify the packageName of the target application, what metrics you want to measure and how many iterations the benchmark should run.

```
 * Run this benchmark from Studio to see startup measureme
 * for investigating your app's performance.
 */
@RunWith(AndroidJUnit4::class)
class ExampleStartupBenchmark {
    @get:Rule
    val benchmarkRule = MacrobenchmarkRule()

    @Test
    fun startup() = benchmarkRule.measureRepeated(
        packageName = "TARGET_PACKAGE",
        metrics = listOf(StartupTimingMetric()),
        iterations = 5,
        startupMode = StartupMode.COLD
    ) {  this: MacrobenchmarkScope
        // Press home button before each run
        // to ensure the starting activity isn't visible.
        pressHome()
        // starts default launch activity
        startActivityAndWait()
    }
}
```

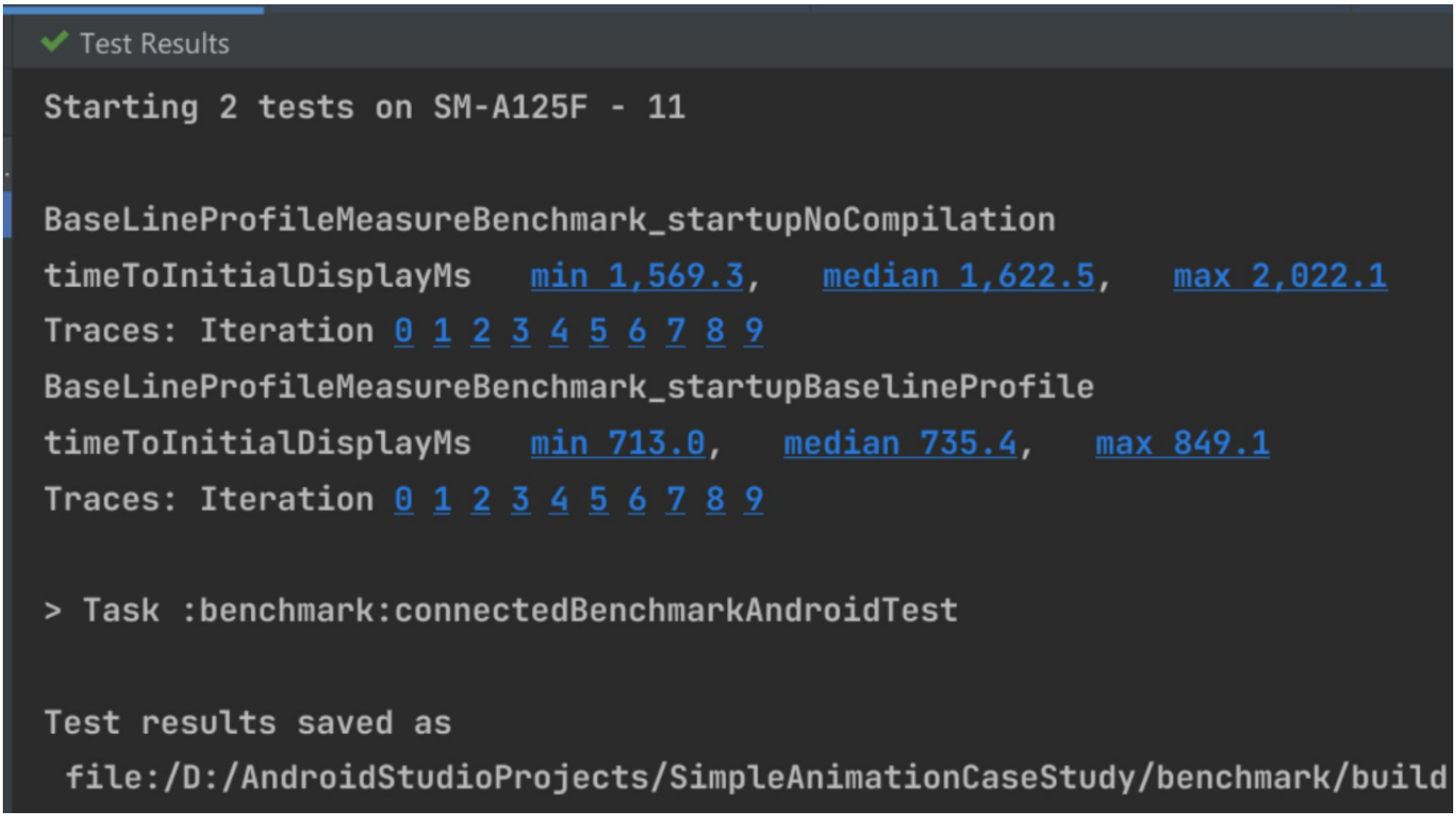
Run the benchmark

You can also run all benchmarks in a Gradle module from the command line by executing the connectedCheck command:

```
$ ./gradlew :macrobenchmark:connectedCheck
```

You should benchmark on real devices and not on Android emulators. If you attempt to run the benchmarks on an emulator, it will fail at runtime with a warning that it’s likely to give incorrect results. While technically you can run it on an emulator, you’re basically measuring your host machine performance — if it’s under heavy load, your benchmarks will appear slower and vice versa.

During execution, the benchmark will start and stop your app several times (based on iterations) and afterwards it will output results to Android Studio




it Agenturen