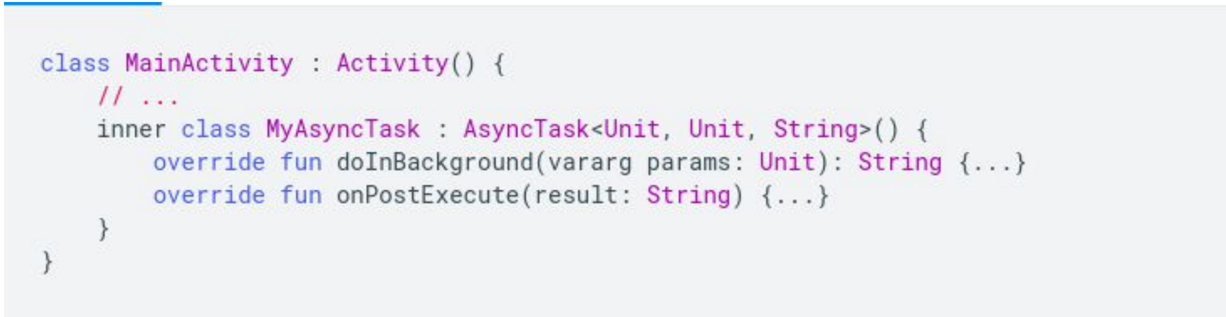
Increasing performance in an Android application

Working with threads

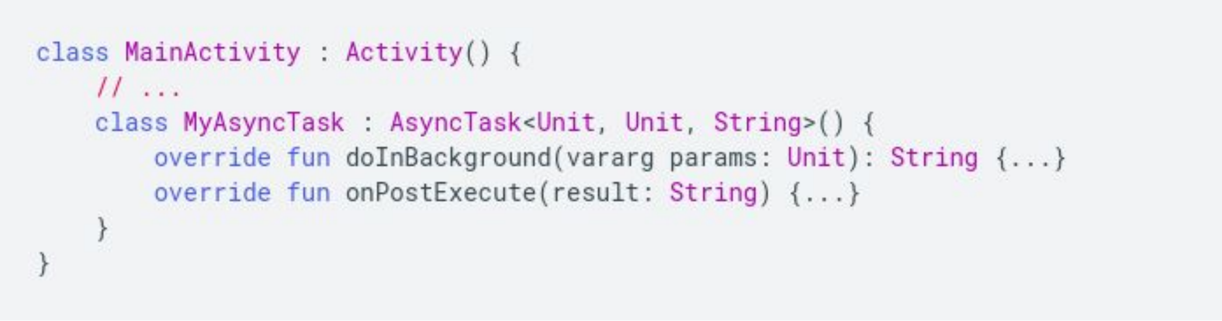
- Most of the performance issues can be solved if you know how to work with it
- 16 ms delay is enough to observe lack of UI responsiveness on main thread
    - with 5 seconds delay there is an ANR error
- UI Objects are not thread safe
- Threads cost a minimum of 64k of memory each
- setThreadPriority() method

Working with threads

Implicit reference



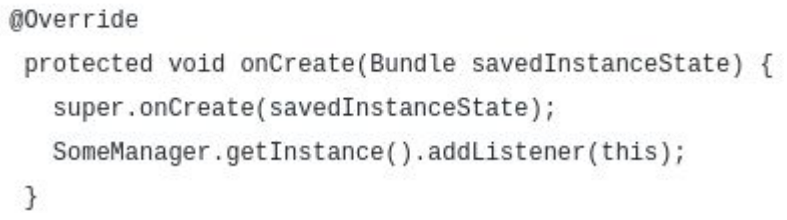
```
class MainActivity : Activity() {
    // ...
    inner class MyAsyncTask : AsyncTask<Unit, Unit, String>() {
        override fun doInBackground(vararg params: Unit): String {...}
        override fun onPostExecute(result: String) {...}
    }
}
```



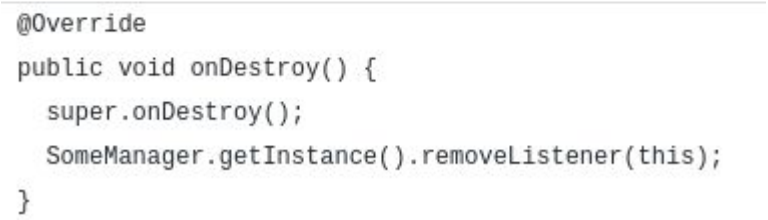
```
class MainActivity : Activity() {
    // ...
    class MyAsyncTask : AsyncTask<Unit, Unit, String>() {
        override fun doInBackground(vararg params: Unit): String {...}
        override fun onPostExecute(result: String) {...}
    }
}
```

Managing memory leaks

- It’s important to take care of it!!!
- Avoid static references
- Unregister your events and handlers
- Understand the architecture before coding



```
@Override
 protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   SomeManager.getInstance().addListener(this);
 }
```



```
@Override
public void onDestroy() {
  super.onDestroy();
  SomeManager.getInstance().removeListener(this);
}
```

Removing deprecated APIs

- Know and use proper APIs
- Refactor your dependencies
- Update your dependencies and tools periodically
- Tips:
    - Prefer RecyclerView over ListView

Avoid abuse

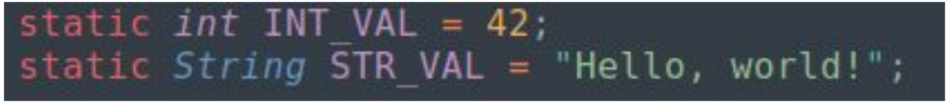
- Don’t call private APIs by reflection
- Using ***adb shell am*** to communicate with other processes should be avoided
- Don’t use ***Runtime.exec*** to communicate with processes

Prefer static methods over virtual methods

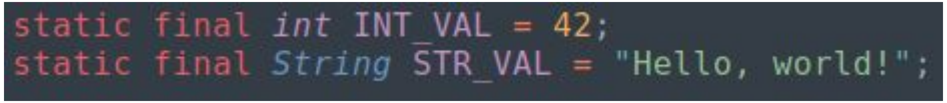
- What does virtual methods means?
- Static methods are 15~20% faster
- Static methods won’t alter object state

Use static final for constants

- Non-final fields get their value by a ***<clinit>*** method
- For primitive types and Strings, *final* fields use an optimization
- Use snake case (all caps)
  - no increase in performance, but good practice

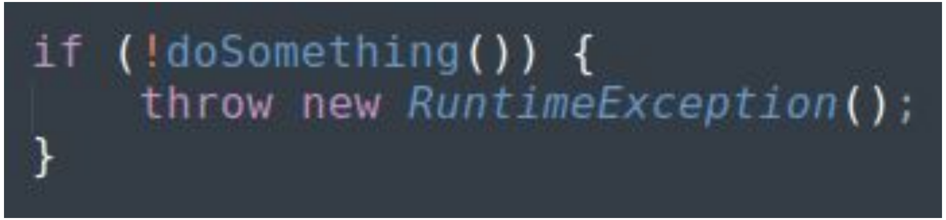


```
static int INT_VAL = 42;
static String STR_VAL = "Hello, world!";
```

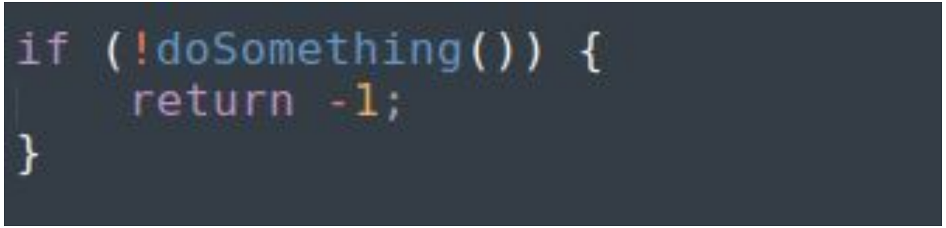


```
static final int INT_VAL = 42;
static final String STR_VAL = "Hello, world!";
```

Don’t use exceptions to control the flow



```
if (!doSomething()) {
    throw new RuntimeException();
}
```



```
if (!doSomething()) {
    return -1;
}
```

```
int sum = 0;
for (int i = 0; i < array.length; ++i) {
    sum += array[i].splat;
}
```

Use for-each loop instead of for loop

```
int sum = 0;
for (Foo a : array) {
    sum += a.splat;
}
```

Use profilers to profile performance

- Always measure before and after optimizing code
- Sometimes the obvious is **not** better

Use profilers to profile performance

- Always measure before and after optimizing code
- Sometimes the obvious is **not** better

```
sort(array.begin(), array.end());
for (int i = 0; i < 100000; ++i) {
  for (int v : array) {
    if (v >= 128) {
      // Code
    }
  }
}
```

Avoid using float

- Usually, floating point types are 2x slower than integer types
- doubles are 2x larger than floats

Layout performance improvements

- Reuse layouts with includes and merges
- Be careful with layout hierarchies
- Use the Hierarchy Viewer and Lint to optimize your layouts

Know and use libraries

- Prefer mature and well known libraries
- Don’t reinvent the wheel if you don’t need to

Use native methods carefully

- Writing native (e.g. C/C++) code can be dangerous
  - There’s a cost related to the interoperability
  - The JIT can’t easily optimize native code
  - You need to compile the native code for each architecture you wish to run on
  - $$$

References

https://heartbeat.fritz.ai/increasing-performance-in-an-android-application-1086640aeef

https://developer.android.com/training/articles/perf-tips

https://stackoverflow.com/questions/11227809/why-is-processing-a-sorted-array-faster-than-processing-an-unsorted-array