



Android lecture 5

Threading, Coroutines

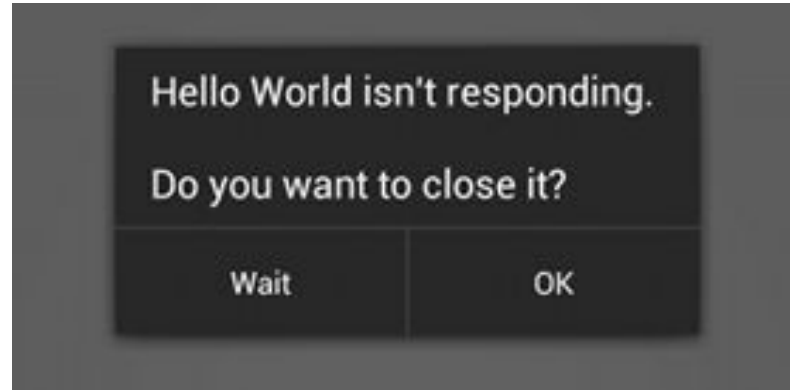
Background processing

“Some people, when confronted with a problem, think, “I know, I’ll use threads,” and then two they have problems.”

Background processing

- Threads
- Handler
- ~~AsyncTask~~ - Deprecation in Android 11 (API-30)
- ~~Loader~~ deprecated Android 9 (API-28)
- Kotlin coroutines
- RxJava

Motivation



Keep your application responsive

Background processing

- Avoid long running operations on Main/UI thread
 - Files, database, network
- Most component runs on Main thread by default
- 5 second to ANR (10s BroadcastReceiver)

Background processing

- Main thread = UI thread
- Never block UI thread

Background processing - issues

- Activities can be restarted
- Memory leaks
- Crashes

Thread

- `java.lang.Thread`

```
Thread() {  
    override fun run() {  
        // Long running operation  
        . . .  
    }  
}.start()
```

- Standard java thread
- Simple way how to offload work to the background
- UI can't be updated from background

Handler

- `android.os.Handler`
- Sends and processes messages
- Instance is bound to thread/message queue of the thread creating it
 - Scheduling messages and `Runnables` to be executed at some point in future
 - Enqueue an action to be performed on different thread

Handler

Receiving message on UI thread

- Overriding handleMessage(Message)

```
val handler = object : Handler() {  
    override fun handleMessage(msg: Message?) {  
        Txt_username_value.text =  
            msg.data.getString("data_key")  
    }  
}
```

Send message from background

- Obtain message is more effective than create new instance
- Requires reference to handler

```
val message = handler.obtainMessage()  
message.arg1 = 1001  
handler.sendMessage(message)
```

Looper and Handler

- Looper
 - Class that runs a message loop for a thread
 - UI thread has its own Looper
 - `Looper.getMainLooper()`
- Handler
 - Provides interaction with the message loop

HandlerThread

- Holds a queue of task
- Other task can push task to it
- The thread processes its queue, one task after another
- When queue is empty, it blocks until something appears

Async task - DEPRECATED

- `android.os.AsyncTask`
- Simplify running code on background
- `AsyncTask<Params, Progress, Result>`
 - Params - The type of the parameters sent to the task upon execution
 - Progress - type of progress unit published during background operation
 - Result - type of result of background operation

AsyncTask - methods

- `onPreExecute()`
 - UI thread, before executing, show progress bar
- `doInBackground(Params...)`
 - Background thread
 - `publishProgress(Progress...)`
 - Returns Result
- `onProgressUpdate(Progress...)`
 - UI thread
 - For updating progress, params are values passed in `publishProgress`
 - `onPostExecute(Result)`
 - UI thread
 - Returned value from `doInBackground` is passed as parameter

AsyncTask - canceling

- `cancel(boolean)` - Cancel execution of task
- `isCancelled()` - call often in `doInBackground` to stop background processing as quick as possible
- `onCancelled(Result)` - called instead of `onPostExecute()` in case task was cancelled

Memory leaks

- Activity runs AsyncTask which takes long time, meanwhile configuration change happens
- Anonymous and non-static inner class still keeps reference to Activity => Activity can't be garbage collected => activity leaks

Memory leaks - Solutions

- Disable configuration changes in manifest
 - Don't do this, it just hides another bugs
- ~~Retain activity instance~~
 - ~~Using `onRetainNonConfigurationInstance()` and `getLastNonConfigurationInstance()`~~ ~~deprecated~~
- WeakReference to activity/fragment or views
- Task as static inner class
- TaskFragment
 - Fragment without UI and called `setRetainInstance(true)`
- AsyncTaskLoader
- ViewModel + LiveData

Kotlin coroutines intro

Kotlin coroutines

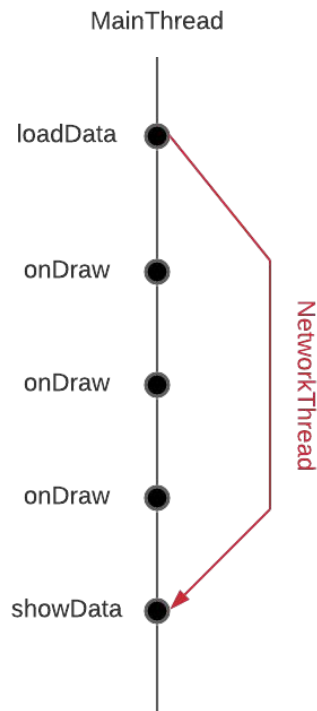
- Lightweight thread
- Uses suspending functions

Suspend function

- Function which is able to suspend its execution without blocking thread

```
suspend fun loadData() { delay(10_000) }
```

- Suspend lambda
- suspend/resume
 - Internally pass Continuation object as callback
 - Uses finite state machine under the hood
- Can be called only from other suspend function or in coroutine created by coroutine builder



Suspend functions

- `suspend` does not mean to run function on background

Suspend function

Suspend functions should be main-safe

CoroutineDispatcher

- All coroutines run in dispatcher
- Coroutine can **suspend** themselves
- Knows how to **resume** suspended coroutines

CoroutineDispatcher

Dispatchers.Main	Dispatchers.IO	Dispatchers.Default
Main thread, interact with UI, light work	Disk and network IO off the main thread	CPU intensive work
<ul style="list-style-type: none">• Call suspend functions• Call UI functions• Updating LiveData	<ul style="list-style-type: none">• Database• R/W files• Networking	<ul style="list-style-type: none">• Sorting list• Parsing JSON• DiffUtils

CoroutineDispatcher

```
override suspend fun getUser(username: String): User? = withContext(Dispatchers.IO) {  
    return@withContext GithubServiceFactory.githubService.getUser(username).body()  
}
```

Coroutine scope

- Keep track of all coroutines running inside
- Not possible to start coroutine outside of some scope
- If scope cancels, coroutines cancels
- GlobalScope - lifetime of whole application
- ViewModel.viewModelScope - extension property
 - Cancel coroutines started by current view model when it is cleared

```
class UserViewModel: ViewModel() {  
    fun fetchData(username: String) {  
        viewModelScope.launch {  
            state.value = LoadInProgress  
            fetchDataSuspend(username)  
        }  
    }  
}
```

Coroutine scope

Avoid leaking coroutines

Coroutine scope builders

- Creates new coroutine scope inside current one
- Cancellation is propagated from parent to children's
- For parallel work decomposition
- `coroutineScope` vs. `supervisorScope`
 - `coroutineScope` - cancels if any of its children fail
 - `supervisorScope` - still run if some children fail
 - Suspends until coroutines complete

Coroutines - starting

- launch
 - Fire and forget - do not return result to caller
 - Usually bridge from regular function into coroutines
 - Return Job for cancellation
 - Do not block current thread
- async/await
 - Start computation asynchronously
 - Creates coroutine and return it's future result (Deferred)
 - Await - wait until coroutine finishes and return result to the caller
 - Thrown exceptions are not signaled until await is called
- runBlocking
 - Blocks until coroutine finishes
 - Handy for initial refactoring



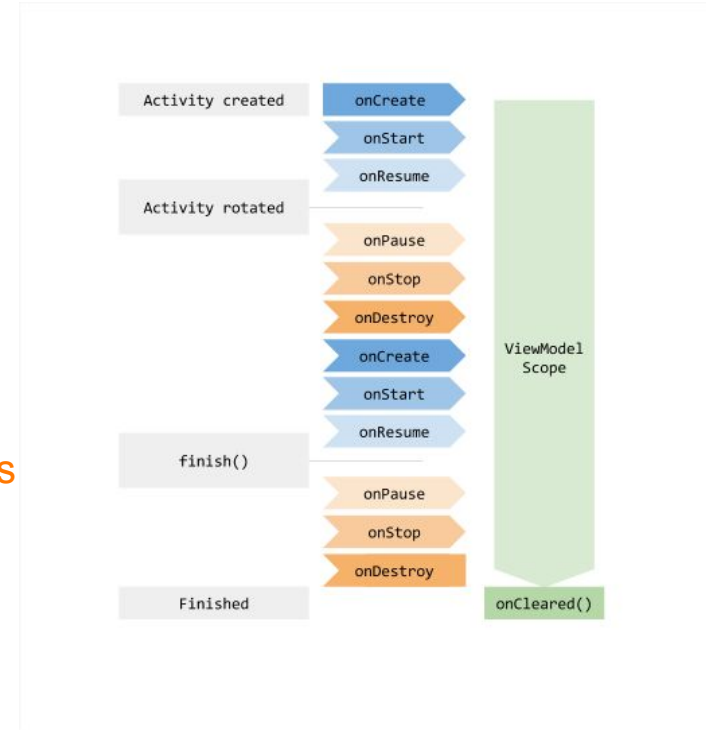
Libraries, gradle plugins, etc...

Android jetpack

- Set of libraries from google
- <https://developer.android.com/jetpack>
- Groups
 - Foundation
 - AppCompat, Android KTX, Multidex, Test
 - Architecture
 - Data binding, Lifecycles, LiveData, Navigation, Paging, Room, ViewModel, WorkManager
 - Behavior
 - Download manager, Media & playback, Notifications, Permissions, Preferences, Sharing, Slices
 - UI
 - Animations & transitions, Auto, Emoji, Fragment, Layout, Palette, TV, Wear OS

Android jetpack ViewModel

- Store and manage UI related data
- Scoped to Lifecycle when getting view model instance
- Do not hold reference to
Activity/Fragment/View/Lifecycle !!!
- Can hold LifecycleObserver but cannot observe changes



Android jetpack - LiveData

- Observable data holder
- Lifecycle aware
- Activity/Fragments from jetpack implements Lifecycle

DexCount plugin

- Computes methods count in APK
- Visualize count in nice chart
- <https://github.com/KeepSafe/dexcount-gradle-plugin>

Retrofit

- A type-safe HTTP client for Android and Java
- Simplify communication with some API service
- Configurable
 - OkHTTP 3 client - compression, timeouts
 - Supports multiple convertors
 - Gson
 - Jackson
 - Moshi
 - Protobuf
 - Wire
- <https://square.github.io/retrofit/>

OkHttp

- An HTTP & HTTP/2 client for Android and Java applications
- Supports sync/async calls
- Supports multiple addresses per URL (Load balancing, failover)
- <http://square.github.io/okhttp/>

Dagger

- Dependency injection framework
- Decouple code
- Better testing
- <https://dagger.dev/android.html>

Flipper

- Debug platform by Facebook
- Inspect
 - ViewHierarchy
 - Database
 - Shared preferences
 - Network traffic
- <https://fbflipper.com/>

LeakCanary

- Helps with finding memory leaks
- <https://github.com/square/leakcanary>

Ktor

- Multiplatform http client/server
- <https://ktor.io>

Kotlinx.serialization

- Data serialization/deserialization
 - Json
 - Protocol buffers
- <https://kotlinlang.org/docs/serialization.html>

Thank you Q&A

Feedback is appreciated

lukas.prokop@avast.com

Please use [mff-android] in subject