ToiletPaper #112

Kotlin Coroutines

Author: Marco Pfattner / Senior Software Architect/ Business Division Automotive Bavaria

Problem

Certain actions should run in the background to avoid blocking the main thread of the app. Typically threads or AsyncTasks are used for this: the action is runs in the background, and finally the result is displayed on the main thread.

Solution

Kotlin offers a more elegant way with coroutines. Coroutines are lightweight threads that are dispatched to a Kotlin-managed thread pool if you use existing dispatchers. This is comparable with GCD from iOS.

Example

An image should be downloaded and displayed:

```
private fun downloadImageWithCoroutines() {
    GlobalScope.launch(Dispatchers.Main) {
        binding.status.text = "Downloading image ..."
        // Download the image on an IO dispatcher without blocking the main thread
        val image = withContext(Dispatchers.IO) { downloadImage() }
        // Continue on the main thread
        binding.image.setImageDrawable(image)
    }
}
```

In the example above, the status is first set on the main thread, followed by an asynchronous download of the image. The special thing about this is that the main thread is not blocked, but only when the result is available the rest of the code continues to run on the main dispatcher.
Technically, Kotlin solves this by executing the code as continuation after `withContext`. This enables you to rewrite the `downloadImage` method so that it passes the result to a continuation and does not return it directly. Of course the caller of the method has to take care of the threading:

```
private fun downloadImage(continuation: (Drawable?) -> Unit) {
    val drawable = url.openStream()?.use {
        Drawable.createFromStream(it, "image")
    }
    continuation(drawable)
}
```

A special strength is the combination of several `async` calls. The `async` calls can be executed simultaneously on different IO threads:

```
// Start multiple requests
val image1 = async(Dispatchers.IO) { downloadImage1() }
val image2 = async(Dispatchers.IO) { downloadImage2() }
// Wait for the results and show the images
binding.image1.setImageDrawable(image1.await())
binding.image2.setImageDrawable(image2.await())
```

Further Aspects

- Kotlin Under the Hood
- Suspending Functions
- Continuations