15. Lesson 08/05/23

Async Task

Contents

Threads

The Main Thread

When an Android app starts, it creates the **main thread**, which is often called the UI thread.

The **UI Thread** need to give its attention to drawing the UI and keeping the app responsive to user input.

- App runs on Java thread called “main” or “UI thread”.
- UI threads draws UI on the screen.

Users uninstall unresponsive apps

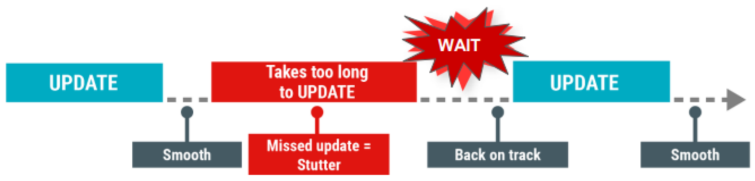
If the UI waits too long for an operation to finish:

→ It becomes **unresponsive**

→ User not happy! (The framework shows an Application Not Responding (ANR) dialog).

The Main thread must be fast

- Hardware updates screen every 16 milliseconds (60 fps).
- UI thread has 16 ms to do all its work.
- If it takes too long, app seems to hang or to be blocked.



What is a long running task?

Examples of possible long running task:

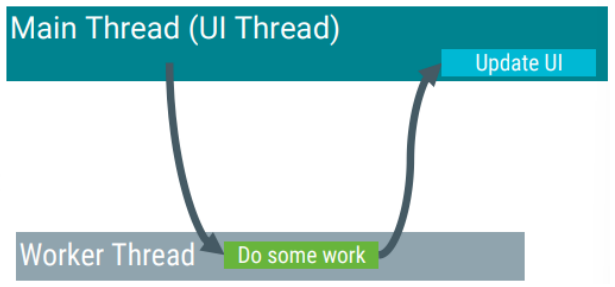
- Network operations
- Long calculations
- Downloading / uploading files
- Processing images
- Loading data
- Interacting with Databases

Background threads

Solution: execute long running tasks on a background thread.

HOW? → Async Task

- Kotlin coroutines
- RxJava
- Executors



Two rules for Android threads

1. Do not block the UI thread
    - Complete each task in less than 16 ms for each screen

- Run slow non-UI tasks on a non-UI thread

2. Do not access the Android UI toolkit from outside the UI thread
   - Do UI work only on the UI thread

AsyncTask

What is AsyncTask?

AsyncTask allows to:

1. Perform background operations on a worker thread.
2. Publish results on the UI thread.

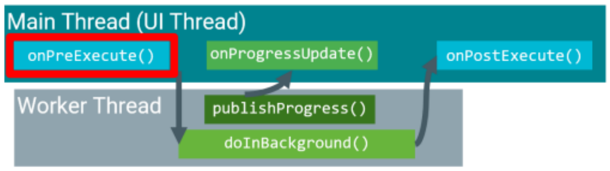
Without needing to directly manipulate threads or handlers.

A worker thread is any thread which is not the main or UI thread.

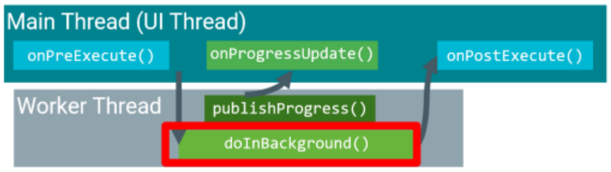
AsyncTask Execution Steps

When **AsyncTask is executed**, it goes through several **steps**:

- ***onPreExecute()*** → is invoked on the UI thread before the task is executed.
  - Normally used to set up the task.



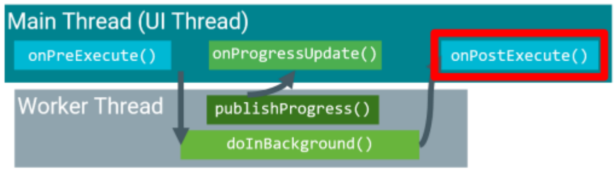
- ***doInBackground()*** → is invoked on the background thread immediately after *onPreExecute()* finishes.
  - Performs a background computation, returns a result, and passes the result to *onPostExecute().*



  - The *doInBackground()* method can also call *publishProgress(Progress …)* to publish one or more units of progress.

- ***onProgressUpdate()***
  - Runs on the main thread
  - Receives calls from *publishProgress()* from background thread

- ***onPostExecute()*** → runs on the UI thread after the background computation has finished.
  - The result of the background computation is passed to this method as a parameter.



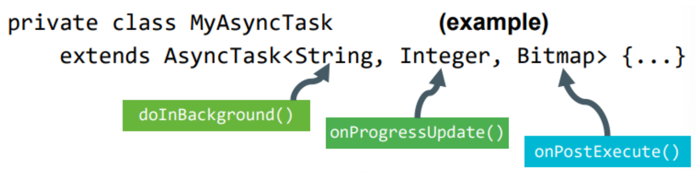
Creating an AsyncTask

Subclass AsyncTask:

```
private class MyAsyncTask
  extends AsyncTask<type1, type2, type3> {...}
```

1. **“Params”** → Provide data type (*type1*) sent to *doInBackground().*
2. **“Progress”** → Provide data type (*type2*) of progress units for *onProgressUpdate().*
3. **“Result”** → Provide data type (*type3*) of result for *onPostExecute().*

MyAsyncTask class definition example



```
private class MyAsyncTask           (example)
    extends AsyncTask<String, Integer, Bitmap> {...}
```

- String → could be query, URI fro filename.
- Integer → percentage completed, steps done.
- Bitmap → an image to be displayed.
- (Use Void if no data passed.)