Threads and ExecutorService

A well known problem

stackoverflow Questions Jobs Documentation BETA Tags Users Search... Log In Sign Up

How to fix android.os.NetworkOnMainThreadException?

Ask Question

1562

I got an error while running my Android project for RssReader.

Code:

```
URL url = new URL(urlToRssFeed);
SAXParserFactory factory = SAXParserFactory.newInstance();
SAXParser parser = factory.newSAXParser();
XMLReader xmlreader = parser.getXMLReader();
RssHandler theRSSHandler = new RssHandler();
xmlreader.setContentHandler(theRSSHandler);
InputSource is = new InputSource(url.openStream());
xmlreader.parse(is);
return theRSSHandler.getFeed();
```

413

And it shows below error:

```
android.os.NetworkOnMainThreadException
```

How can I fix this issue?

android networkonmainthread thread-exceptions

asked 5 years, 11 months ago

viewed 827988 times

active 9 days ago

BLOG

A Dive Into Stack Overflow Jobs Search

Looking for a job?

Full Stack Web Application Developer

TeamSystem Spa *Milano, Italy*

€25K - €30K

javascript sql

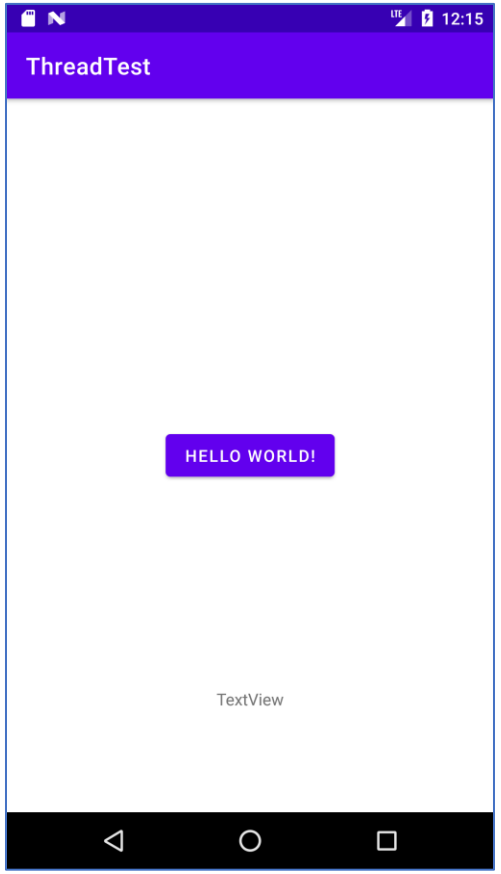
NetworkOnMainThreadException

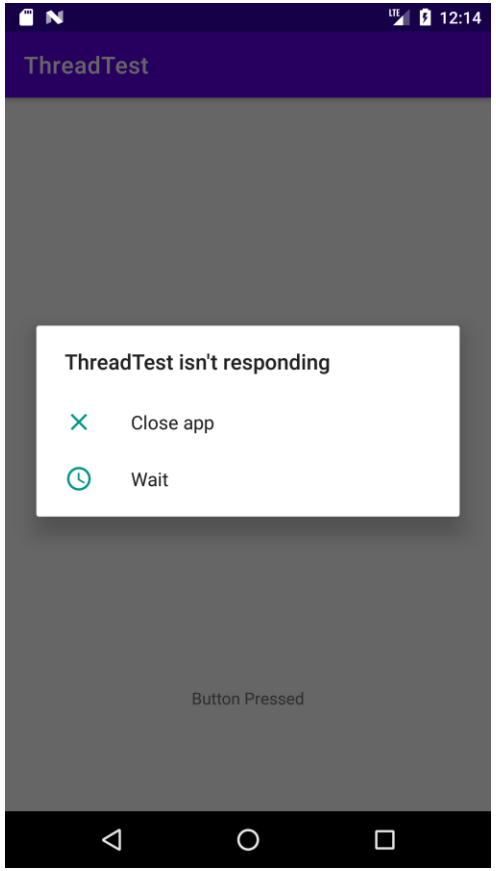
- An application crashes on Android versions 3.0 and above if attempts to perform a **networking operation** on its main thread.
- The main thread or "UI thread" dispatches events to the appropriate views/widgets and thus is very important.
- If the UI thread runs **long operations** such as
  - Opening a Socket connection (i.e. new Socket()).
  - HTTP requests (i.e. HTTPClient and HTTPUrlConnection).
  - Attempting to connect to a remote MySQL database.
  - Downloading a file (i.e. Downloader.downloadFile()).
- From the **user's perspective**, the application will appear to be frozen.

App Not Responding: a Basic Example

- Touch the button two times...

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
    fun buttonClick(view: View) {
        var i = 0
        while (i <= 20) {
            try {
                Thread.sleep(1000)
                i++
            } catch (e: Exception) {   }
        }
        textView.text = "Button Pressed"
    }
}
```





ExecutorService and Thread

- ExecutorService and Thread, **what should I use**?
- **Thread**
  - Long task in general.
  - For tasks in parallel use Multiple threads (traditional mechanisms)
- **ExecutorService**
  - We know the maximum number of concurrent threads
  - Prevents the overhead of creating multiple threads (by using worker threads)

Threads

ANDROID

Threads and processes

- By default, all *components* of the same application run in the *same process and thread* (called *main-thread* or *UI-thread*).
- The *main-thread* is in charge of
  - dispatching events to user interface widgets
  - drawing the events of the UI
- All *components* that run in the same process *are instantiated in the UI thread*, and *system calls* to each component *are dispatched from that thread*.
  - Thus, methods that *respond to system callbacks always run in the UI thread of the process*.

Issue 1: Do not block the UI-thread

- Performing *long operations* in the UI-thread *will block the UI*.
- If the UI thread is blocked, *no event can be dispatched*, including drawing events.
  - *User's perspective*: the application appears to hang.
  - *Android check*: if the UI-thread is blocked for more than a few seconds (5s currently) the user is presented "**application not responding**" (ANR) dialog.
- Hence, if you have operations to perform that are not instantaneous, you should make sure to do them in *separate threads* ("*background*" or "*worker*" threads).

Threads in Kotlin

- Here's how you can instantiate and start a thread Kotlin-style

```
import kotlin.concurrent.thread

class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)

      println("running  from thread(): ${Thread.currentThread()}")
      thread(start = true) {
         println("running  from thread(): ${Thread.currentThread()}")
      }
   }
}
```

```
> running from thread(): Thread[main,5,main]
> running from thread(): Thread[Thread-2,5,main]
```

Kotlin comes with a standard library function thread that **creates** a thread and **runs** the specified [block] of code.

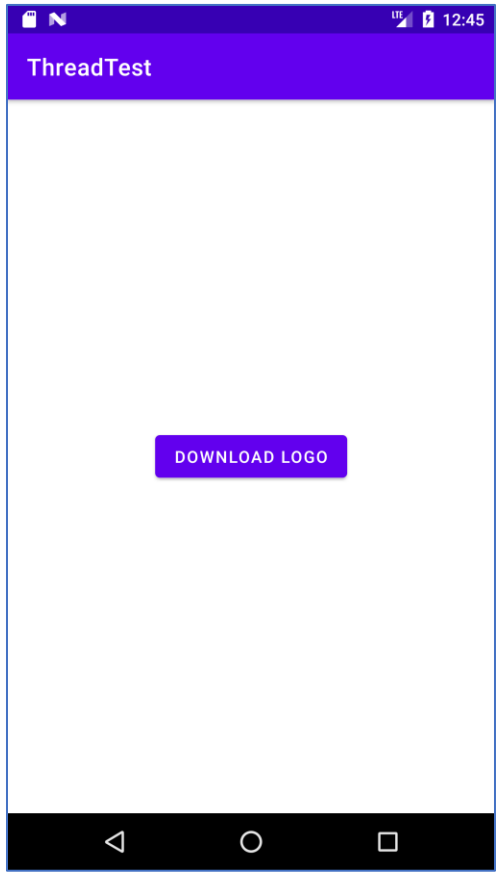
```
public fun thread(
   start: Boolean = true,
   isDaemon: Boolean =    false,
   contextClassLoader: ClassLoader? = null,
   name: String? = null,
   priority: Int = -1,
   block: () -> Unit
)
```

Threads: simple solution

Access the network using a Thread

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        thread(start = true) {
            val url = URL("https://www.uninsubria.it/sites/all/themes/uninsubria/logo.png")
            val bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream())
            val msg = "" + bmp.getWidth() + "x" + bmp.getHeight()
            Log.i("Image Size", msg)
        }
    }
}
```

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="it.uninsubria.helloworld">

    <uses-permission android:name="android.permission.INTERNET"/>

    <application ...
```


12:45
ThreadTest
DOWNLOAD LOGO

onClick()


1:07
ThreadTest
DOWNLOAD LOGO


android
developer lab

Example: wrong solution 1

```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    fun buttonClick(view: View) {
        val url = URL("https://www.uninsubria.it/sites/all/themes/uninsubria/logo.png")
        val bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream())
        imageView.setImageBitmap(bmp)
}
}
```

Since version 3.0 Honeycomb (API level 11) you cannot perform network operations on the main UI thread.
An attempt will cause a `NetworkOnMainThreadException`.

Thread management

- Android natively supports a *multi-threading* environment.
- Threads are created like in Java

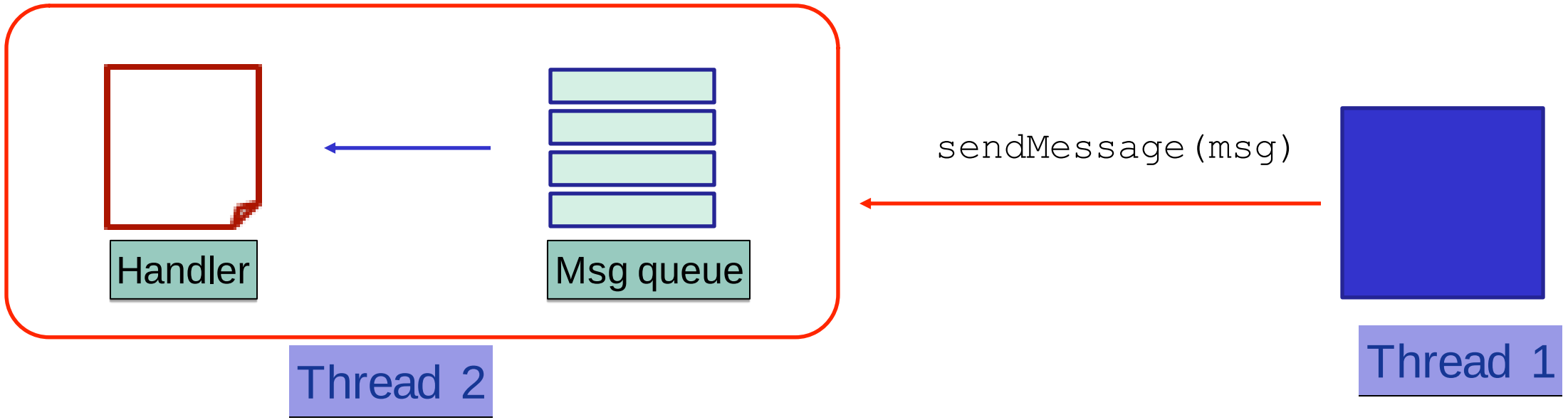
```
class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }

    fun buttonClick(view: View) {
        thread(start = true) {
            val url = URL("https://www.uninsubria.it/sites/all/themes/uninsubria/logo.png")
            val bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream())
            imageView.setImageBitmap(bmp)
        }
    }
}
```

```
E/AndroidRuntime: FATAL EXCEPTION: Thread-2
    Process: it.uninsubria.pdm.threadtest, PID: 6468
    android.view.ViewRootImpl$CalledFromWrongThreadException: Only
the original thread that created a view hierarchy can touch its views.
        at android.view.ViewRootImpl.checkThread(ViewRootImpl.java:6855)
```

UI toolkit is not thread-safe

- A piece of code is *thread-safe* if it only manipulates shared data structures in a manner that *guarantees safe execution by multiple threads at the same time*.
- *UI toolkit is not thread-safe*, motivations:
  - simpler implementation
  - lower execution overhead
  - simpler interfaces
- *Consequences*:
  - you cannot update UI-components from a worker thread
  - to update UI-components from a worker thread *you must use a message-passing mechanism for thread communication*.

Message-passing



- Each thread is associated with:
  - a message queue
  - an handler of the messages
- A message is an object that can be sent/received.

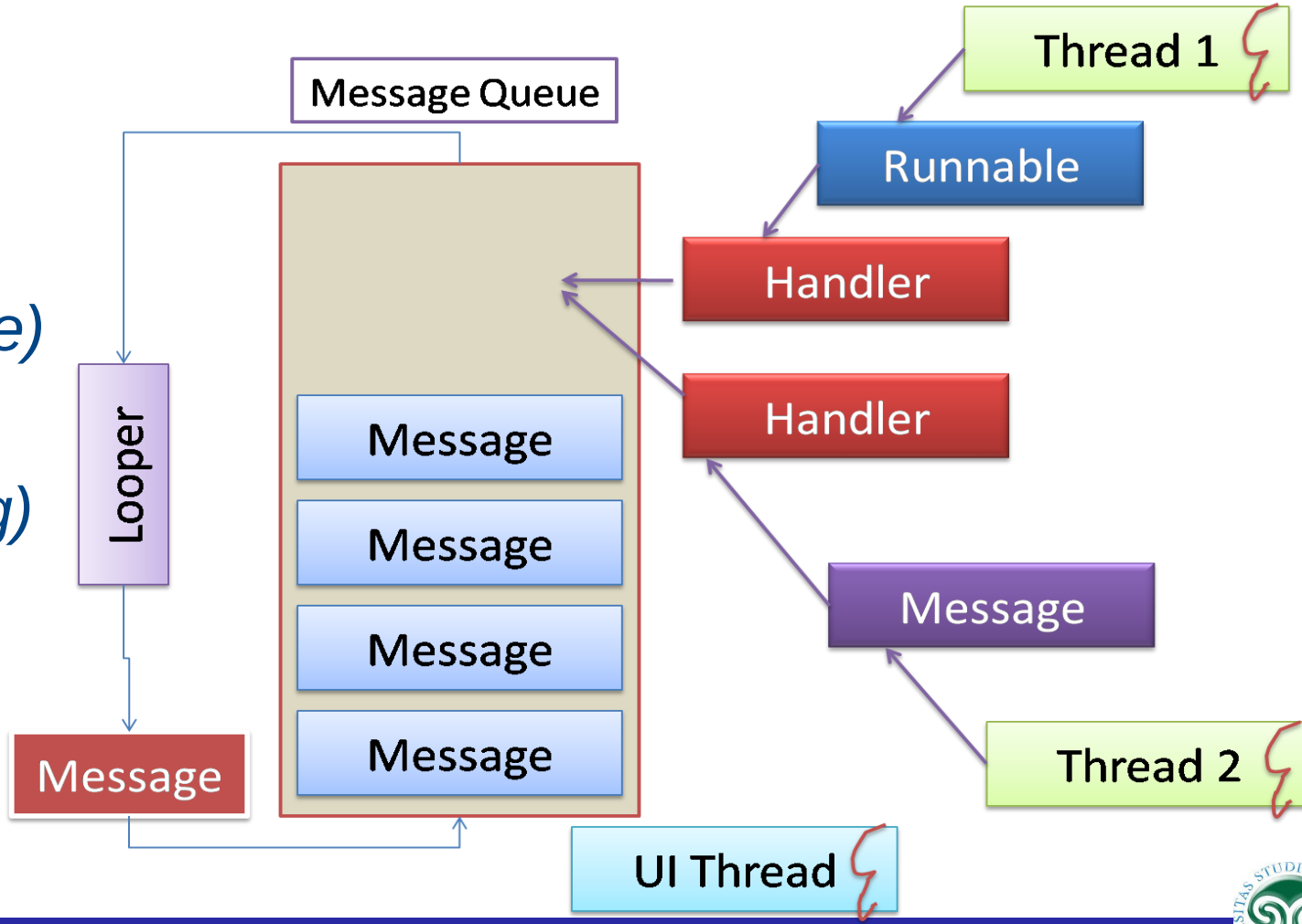
Solutions

- To fix this problem, Android offers several ways to access the UI thread from other threads.
- Here is a list of methods that can help:

***Activity**.runOnUiThread(Runnable)*

***View**.post(Runnable)*

***View**.postDelayed(Runnable, long)*



Solution 1

Activity class

- `public final void runOnUiThread(Runnable action)`

Runs the specified action on the UI thread.

If the current thread is the UI thread, then the action is executed immediately. If the current thread *is not the UI thread*, the action is *posted to the event queue of the UI thread*.

```
fun buttonClick(view: View) {
    thread(start = true) {
        val url = URL("https://www.uninsubria.it/sites/all/themes/uninsubria/logo.png")
        val bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream())
        this.runOnUiThread {
            imageView.setImageBitmap(bmp)
        }
    }
}
```

Solution 2: View ad-hoc solution

- `public boolean post(Runnable action)`

View class

Causes the `Runnable` to be added to the message queue.
The runnable *will be run on the UI-thread*.

Returns true if the `Runnable` was successfully placed in to the message queue.

```
fun buttonClick(view: View) {
    thread(start = true) {
        val url = URL("https://www.uninsubria.it/sites/all/themes/uninsubria/logo.png")
        val bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream())
        imageView.post {
            imageView.setImageBitmap(bmp)
        }
    }
}
```

Solution 3: View ad-hoc solution

- `public boolean postDelayed(Runnable action, long delayMillis)`

Causes the `Runnable` to be added to the message queue.

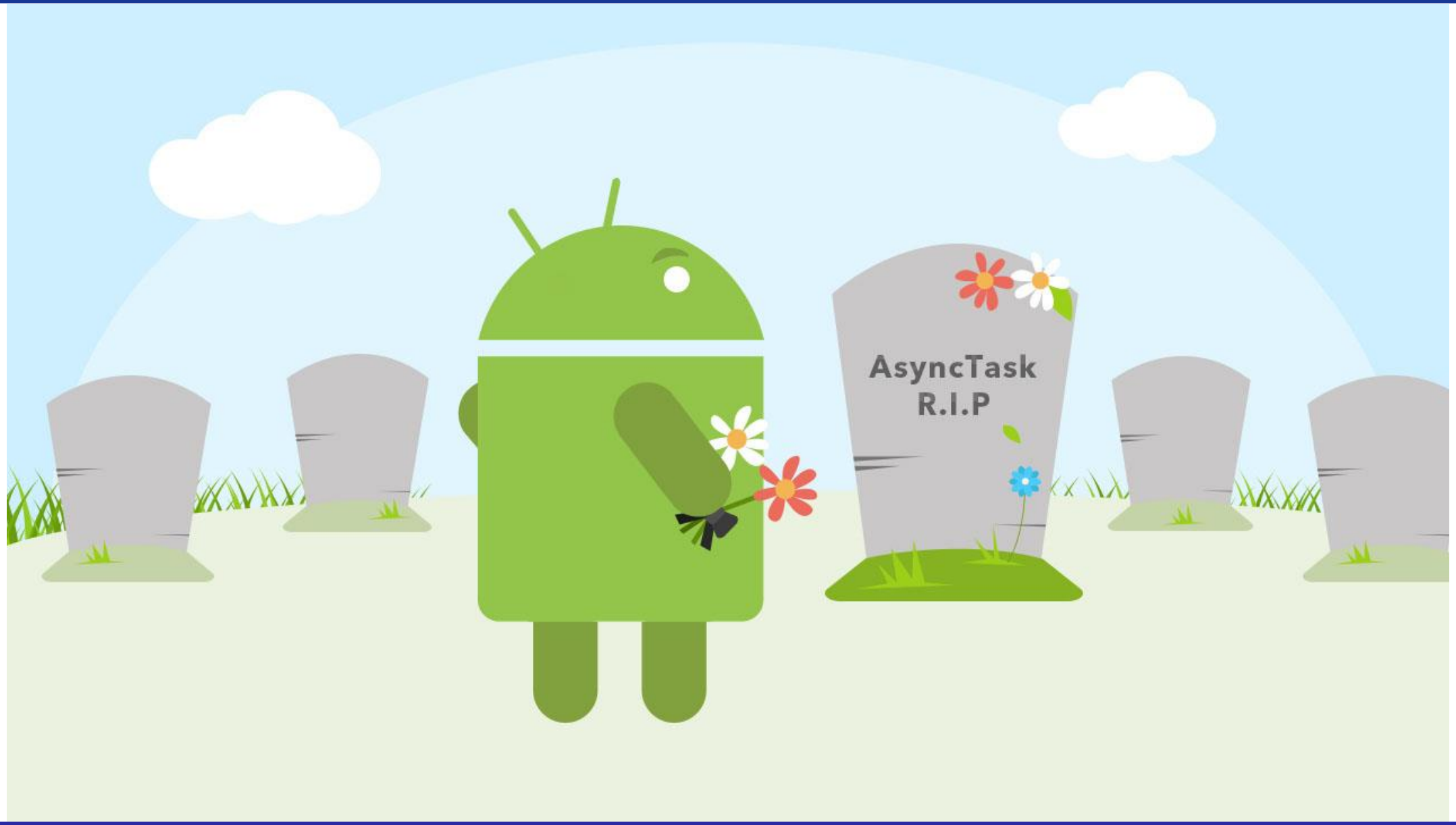
View class

```
fun buttonClick(view: View) {
    thread(start = true) {
        val url = URL("https://www.uninsubria.it/sites/all/themes/uninsubria/logo.png")
        val bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream())
        imageView.postDelayed({
                imageView.setImageBitmap(bmp)
            }, 1000L)
    }
}
```

AsyncTask

android developer lab

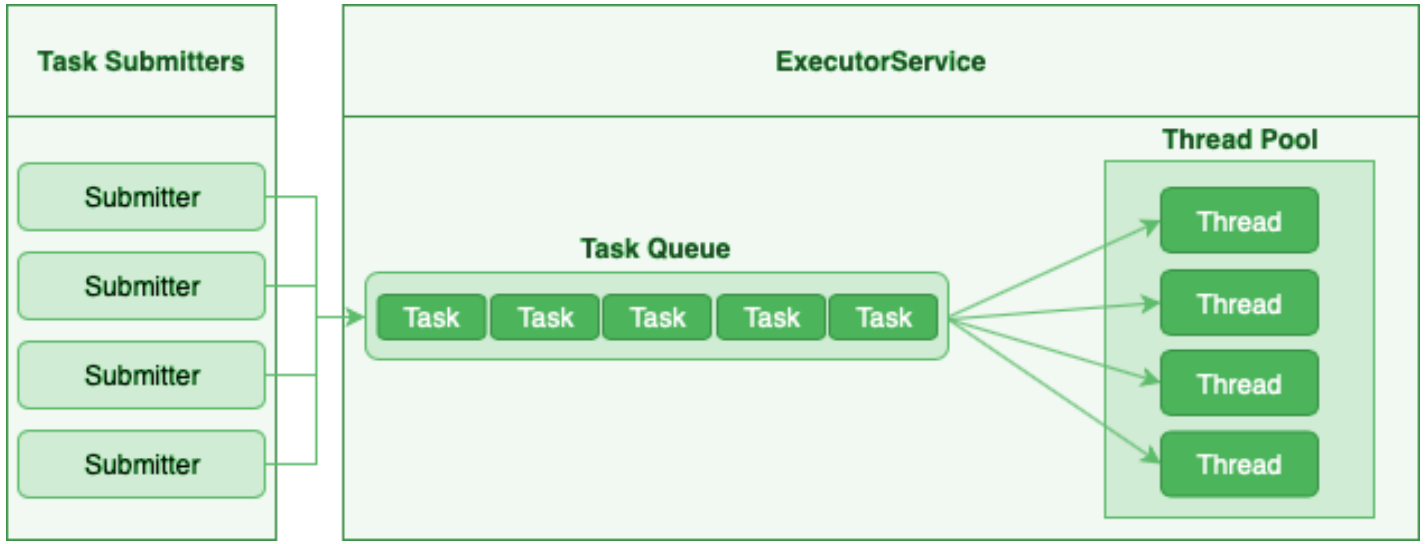
AsyncTask



Executors

Thread Pools

- The Executor Framework in Java is an attempt to separate task submission from task execution.



- When we submit a new task to the executor:
  - If one of the threads is available, it processes the task.
  - Otherwise, the executor adds the new task to its queue.
  - When a thread finishes the current task, it picks up another one from the queue.

Thread Pools

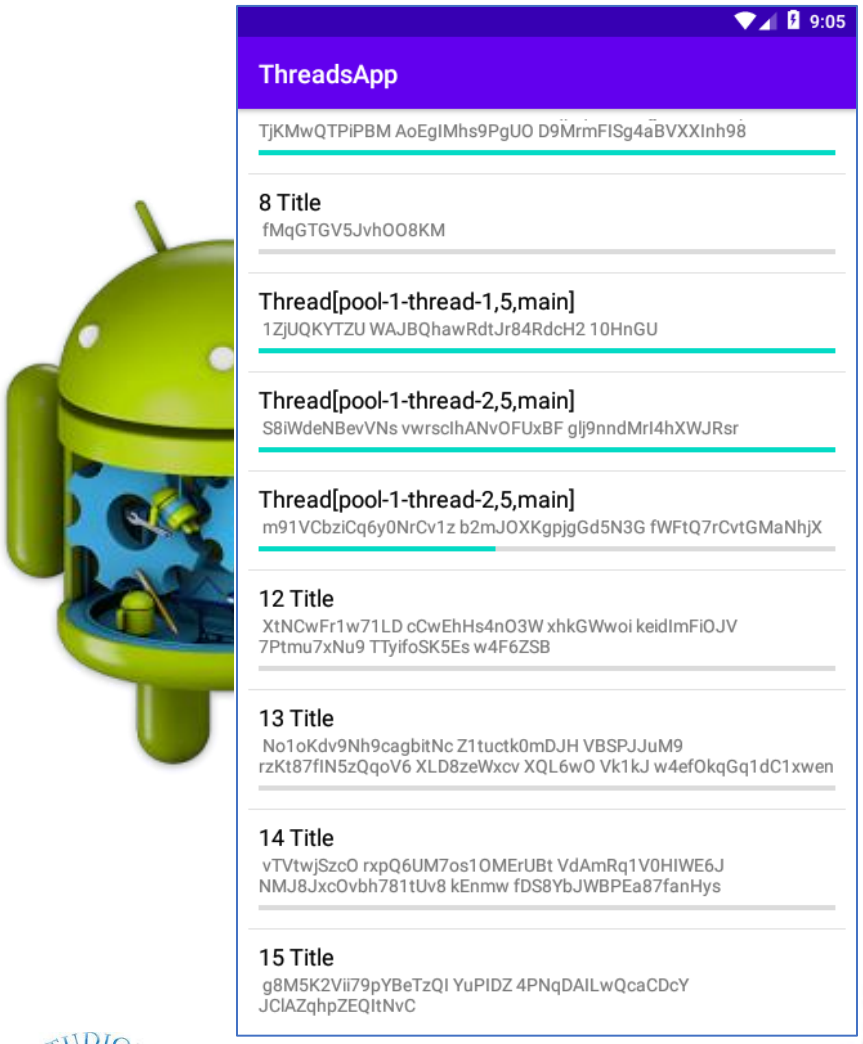
- The Executors helper class contains several methods for the creation of preconfigured thread pool instances.
- **Executors**
  The Executor interface has a single execute method to submit Runnable instances for execution.
- **ExecutorService**
  contains a large number of methods to control the progress of the tasks .
- **ThreadPoolExecutor**
  is an extensible thread pool implementation with lots of parameters and hooks for fine-tuning.

java.util.concurrent.Executors

- The Executor's execute() method takes a Runnable.
- You can use a Handler to enqueue an action to be performed on a different thread

```
fun buttonClick(view: View) {
   val executor = Executors.newSingleThreadExecutor()
   val handler = Handler(Looper.getMainLooper())
   executor.execute {
        val url = URL("https://www.uninsubria.it/sites/all/themes/uninsubria/logo.png")
        val bmp = BitmapFactory.decodeStream(url.openConnection().getInputStream())
        handler.post{
          imageView.setImageBitmap(bmp)
        }
}
```

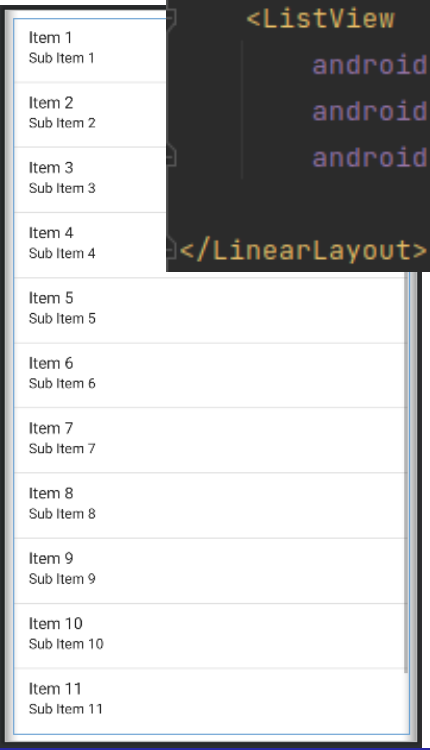
https://developer.android.com/guide/background/threading



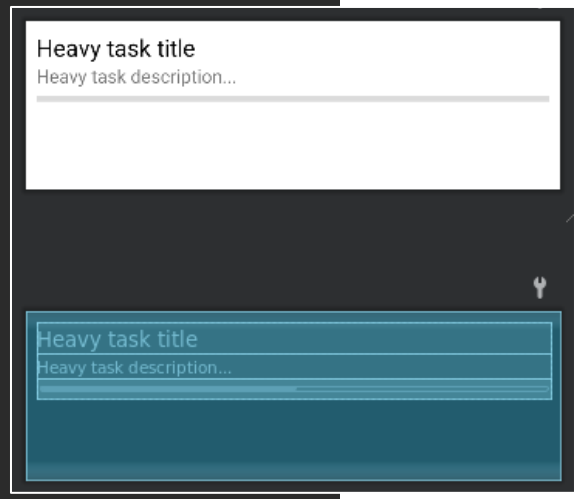
Multi Threads Example

Create the layout

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android
    xmlns:app="http://schemas.android.com/apk/res-a
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="8dp"
    tools:context=".MainActivity">

    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```




```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/an
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="8dp">

    <TextView
        android:id="@+id/rowTextTitle"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Heavy task title"
        android:textSize="18sp"
        android:textColor="@color/black"/>

    <TextView
        android:id="@+id/rowTextDescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Heavy task description..."/>

    <ProgressBar
        android:id="@+id/rowProgressBar"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        style="@style/Widget.AppCompat.ProgressBar.Horizontal"/>

</LinearLayout>
```



Create the structure

```
class MainActivity : AppCompatActivity() {
   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)
      val data = arrayOfNulls<HeavyTask>(100)
      for (n in 0..99){
         data[n] = HeavyTask("Title", "Description", false)
      }
      val adapter = MyAdapter(this, data)
      listView.adapter = adapter
   }
}

data class HeavyTask(
   var title: String,
   var description: String,
   var visible: Boolean,
   var max: Int = 100,
   var progress: Int = -1
)
```

```
class MyAdapter(val context: Context,
          val data: Array<HeavyTask?>) : BaseAdapter() {
override fun getCount(): Int {
   return data.size
}
override fun getItem(pos: Int): HeavyTask? {
   return data[pos]
}
override fun getItemId(pos: Int): Long {
   return data[pos].hashCode().toLong()
}

override fun getView(pos: Int,
    convertView: View?, parent: ViewGroup?): View? {
…
return newView
}
```

Create the ListView row

```
override fun getView(pos: Int, convertView: View?, parent: ViewGroup?): View? {
    var newView = convertView
    if (newView == null){
        newView = LayoutInflater.from(context).inflate(R.layout.row_item_listview, parent, attachToRoot: false)
    }
    val first: Int = (parent as ListView).firstVisiblePosition
    val last: Int = parent.lastVisiblePosition
    if(last != -1){ // last == -1 vuol dire non ancora creato
        for (i in (0 until first) + (last+1 until data.size))
            data[i]?.visible = false
        for (i in (first..last))
            data[i]?.visible = true
    }
    doHeavyTask(executor, (newView as View).rowProgressBar, newView.rowTextDescription , data[pos]!!)
    newView.rowTextDescription.text = data[pos]?.description
    newView.rowTextTitle.text = "List position: $pos"
    newView.rowProgressBar.progress = data[pos]?.progress!!
    return newView
}
```

```
// val executor = Executors.newSingleThreadExecutor()
val executor = Executors.newFixedThreadPool( nThreads: 1)
```

Simulate a Heavy Task

```
private fun doHeavyTask(executor: ExecutorService,
                        pBar: ProgressBar, textView: TextView, task: HeavyTask) {
    if (task.progress > -1 ) return // already started

    val handler = Handler(context.mainLooper)

    executor.execute { // new Task (Runnable)
        while (task.progress < task.max){
            val rndStep = (1..50).random()

            // simulo un'operazione pesante
            Thread.sleep( millis: rndStep*100L)

            task.progress += rndStep
            task.description = Thread.currentThread().toString()
            handler.post {
                if (task.visible) {
                    textView.text = task.description
                    pBar.progress = task.progress
                }
            }
        }
    }
}
```