ITW202: Mobile Application

Unit IV: Developing for Android

Ms. Sonam Wangmo

Gyalpozhing College of Information Technology
Royal University of Bhutan

May 20, 2021

AsyncTask and AsyncTaskLoader

AsyncTask and AsyncTaskLoader

There are several ways to do background processing in Android. Two of those ways are:

- You can do background processing directly, using the AsyncTask class.
- You can do background processing indirectly, using the Loader framework and then the AsyncTaskLoader class.

The main thread

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

- The UI thread dispatches events to the appropriate user interface (UI) widgets.
- The UI thread is where your app interacts with components from the Android UI toolkit (components from the android.widget and android.view packages).

The main thread

- Independent path of execution in a running program
- Code is executed line by line
- App runs on Java thread called "main" or "UI thread"
- Draws UI on the screen
- Responds to user actions by handling UI events

The main thread

- The UI thread needs to give its attention to drawing the UI and keeping the app responsive to user input.
- If everything happened on the UI thread, long operations such as network access or database queries could block the whole UI.

The main thread

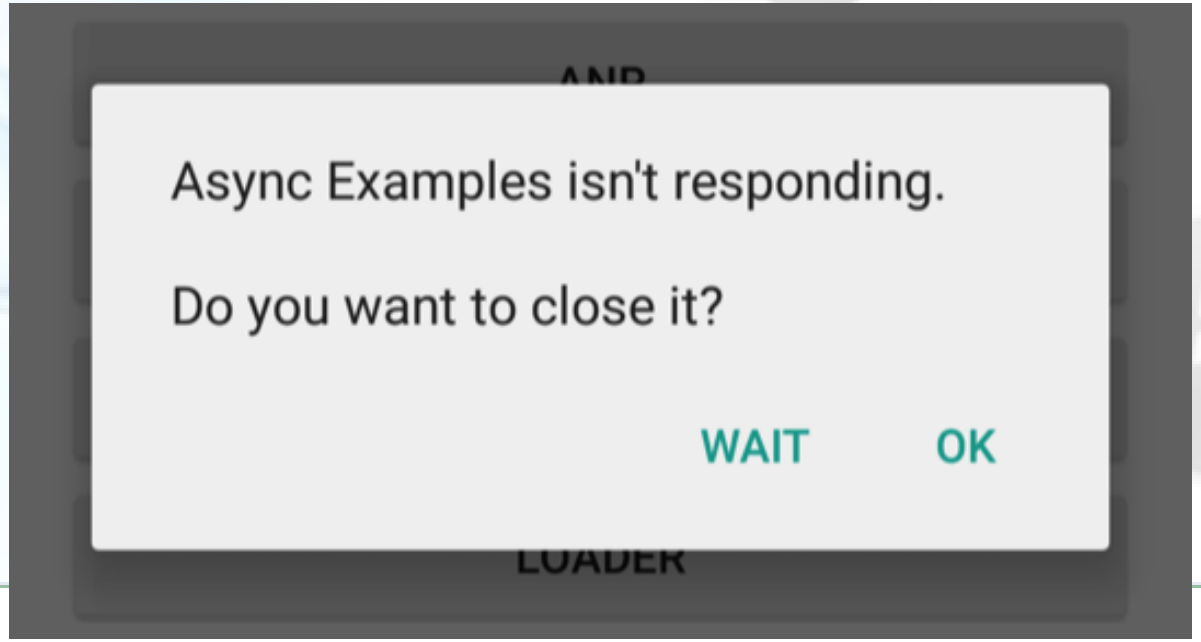
- From the user's perspective, the app would appear to hang. Even worse, if the UI thread were blocked for more than a few seconds (about 5 seconds currently) the user would be presented with the "application not responding" (ANR) dialog.
- The user might decide to quit your app and uninstall it.

The Main thread must be fast

- Hardware updates screen every 16 milliseconds
- UI thread has 16 ms to do all its work
- If it takes too long, app stutters or hangs

Users uninstall unresponsive apps

- If the UI waits too long for an operation to finish, it becomes unresponsive
- The framework shows an Application Not Responding (ANR) dialog



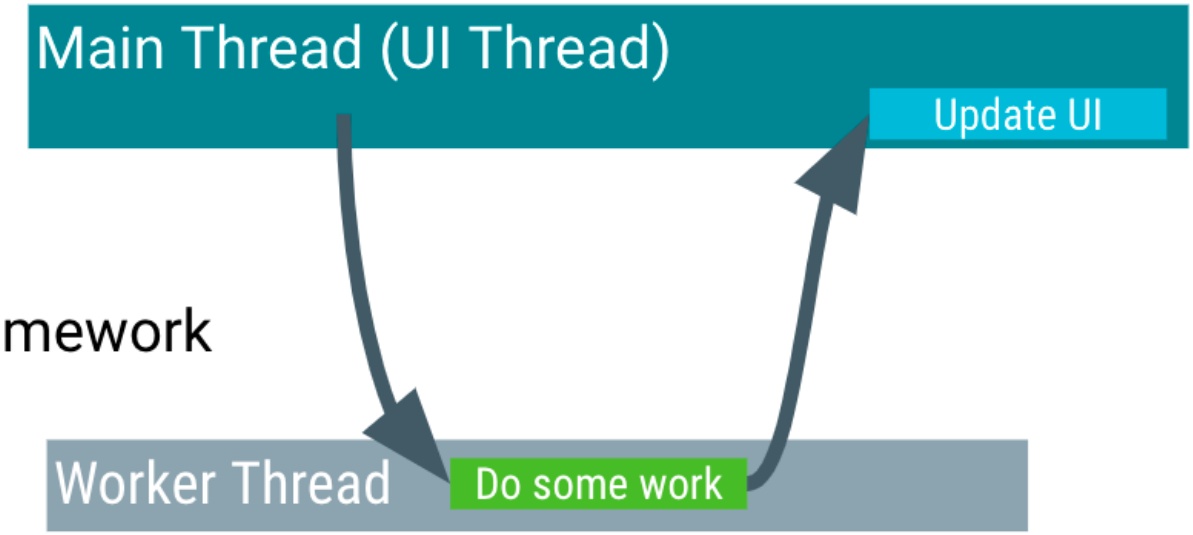
What is a long running task?

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

Background threads

Execute long running tasks on a background thread

- AsyncTask
- The Loader Framework
- Services



Two rules for Android threads

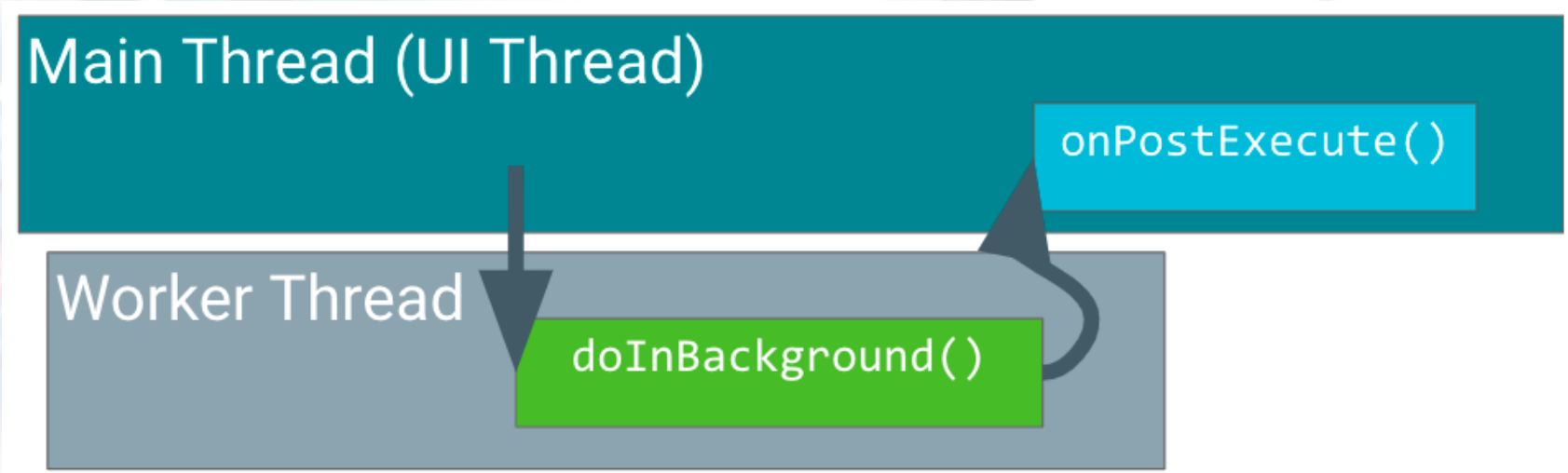
- Do not block the UI thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread
- Do not access the Android UI toolkit from outside
  - the UI thread
  - Do UI work only on the UI thread

AsyncTask

What is AsyncTask?

- A worker thread is any thread which is not the main or UI thread.
- Use the AsyncTask class to implement an asynchronous, long-running task on a worker thread.
- AsyncTask allows you to perform background operations on a worker thread and publish results on the UI thread without needing to directly manipulate threads or handlers.

What is AsyncTask?



AsyncTask

When AsyncTask is executed, it goes through four steps:

1. onPreExecute() is invoked on the UI thread before the task is executed. This step is normally used to set up the task, for instance by showing a progress bar in the UI.

AsyncTask

2. doInBackground(Params...) is invoked on the background thread immediately after onPreExecute() finishes. This step 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.

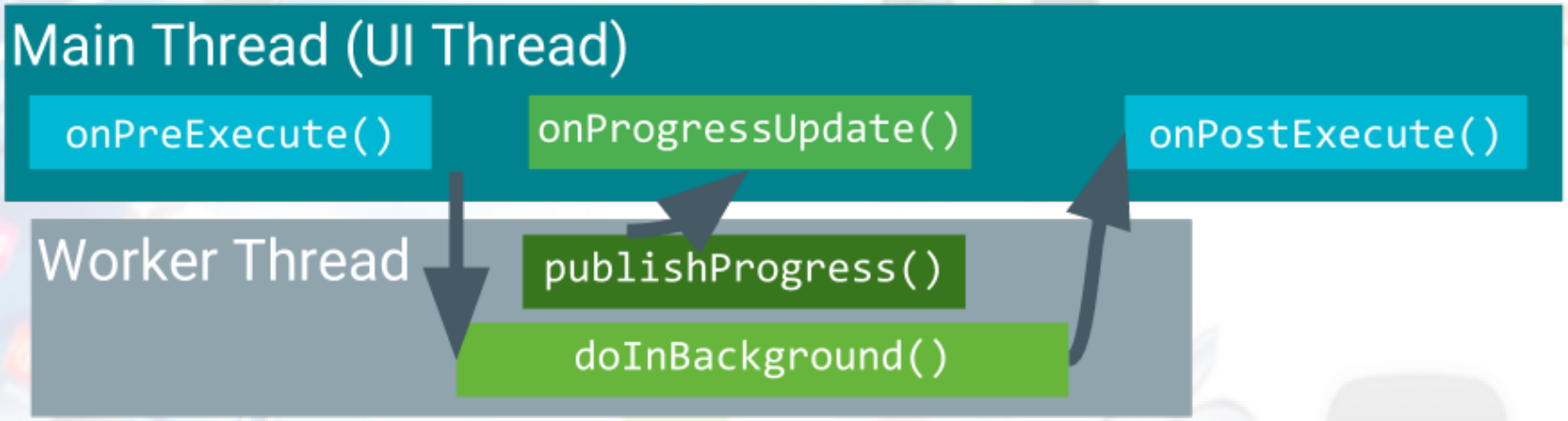
AsyncTask

3. onProgressUpdate(Progress...) runs on the UI thread after publishProgress(Progress...) is invoked. Use onProgressUpdate() to report any form of progress to the UI thread while the background computation is executing. For instance, you can use it to pass the data to animate a progress bar or show logs in a text field.

AsyncTask

4. onPostExecute(Result) 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.

AsyncTask


Main Thread (UI Thread)
onPreExecute()
onProgressUpdate()
onPostExecute()
Worker Thread
publishProgress()
doInBackground()

AsyncTask usage

1. Define a subclass of AsyncTask
2. Provide data type sent to doInBackground()
3. Provide data type of progress units for onProgressUpdate()
4. Provide data type of result for onPostExecute()

AsyncTask parameters

In your subclass of AsyncTask, provide the data types for three kinds of parameters:

1. "Params" specifies the type of parameters passed to doInBackground() as an array.

AsyncTask parameters

2. "Progress" specifies the type of parameters passed to publishProgress() on the background thread. These parameters are then passed to the onProgressUpdate() method on the main thread.

AsyncTask parameters

3. "Result" specifies the type of parameter that doInBackground() returns. This parameter is automatically passed to onPostExecute() on the main thread.

AsyncTask parameters

Specify a data type for each of these parameter types, or use Void if the parameter type will not be used. For example:

```
public class MyAsyncTask extends AsyncTask
<String, Void, Bitmap>
```

AsyncTask parameters

In this class declaration:

- The "Params" parameter type is String, which means that MyAsyncTask takes one or more strings as parameters in doInBackground(), for example to use in a query.

AsyncTask parameters

- The "Progress" parameter type is Void, which means that MyAsyncTask won't use the publishProgress() or onProgressUpdate() methods.
- The "Result" parameter type is Bitmap. MyAsyncTask returns a Bitmap in doInbackground(), which is passed into onPostExecute().

Limitations of AsyncTask

AsyncTask is impractical for some use cases:

- Changes to device configuration cause problems.

  AsyncTask cannot connect to Activity anymore. New AsyncTask created for every config change

- Old AsyncTask objects stay around, and your app may run out of memory or crash.

  If the activity that created the AsyncTask is destroyed, the AsyncTask is not destroyed along with it.

When to use AsyncTask

AsyncTask is impractical for some use cases:

- Short or interruptible tasks
- Tasks that do not need to report back to UI or user
- Lower priority tasks that can be left unfinished
- Use AsyncTaskLoader otherwise

Loaders

What is a Loader?

- Loading data can be memory intensive, and you want the data to be available even if the device configuration changes. For these situations, use loaders, which are classes that facilitate loading data into an activity.

What is a Loader?

To Sum Up: Loaders

- Provides asynchronous loading of data
- Reconnects to Activity after configuration change
- Can monitor changes in data source and deliver new data

Why use loaders?

- Execute tasks OFF the UI thread
- LoaderManager handles configuration changes for you
- Efficiently implemented by the framework
- Users don't have to wait for data to load

What is a LoaderManager?

- Loaders use the LoaderManager class to manage one or more loaders.
- LoaderManager includes a set of callbacks for when the loader is created, when it's done loading data, and when it's reset.
- Can manage multiple loaders:

  loader for database data, for AsyncTask data, for internet data...

Starting a loader

- Creates and starts a loader, or reuses an existing one, including its data
- Use restartLoader() to clear data in existing loader

Starting a loader

- Use the LoaderManager class to manage one or more Loader instances within an activity or fragment.
- Use initLoader() to initialize a loader and make it active.

Starting a loader

For example:

```
getLoaderManager().initLoader(0, null, this);
```

Implementing AsyncTaskLoader

AsyncTaskLoader

AsyncTaskLoader is the loader equivalent of AsyncTask. AsyncTaskLoader provides a method, loadInBackground(), that runs on a separate thread.

AsyncTask ——————> AsyncTaskLoader

doInBackground() ——-> loadInBackground()

onPostExecute() ——————> onLoadFinished()

Steps for AsyncTaskLoader subclass

- Subclass AsyncTaskLoader
- Implement constructor
- loadInBackground()
- onStartLoading()

Subclass AsyncTaskLoader

```
public static class StringListLoader
        extends AsyncTaskLoader<List<String>> {

    public StringListLoader(Context context, String queryString) {
        super(context);
        mQueryString = queryString;
    }
}
```

loadInBackground()

```
public List<String> loadInBackground() {
    List<String> data = new ArrayList<String>;
    //TODO: Load the data from the network or from a database
    return data;
}
```

onStartLoading()

When restartLoader() or initLoader() is called, the LoaderManager invokes the onStartLoading() callback.

Implement loader callbacks in Activity

- onCreateLoader() — Create and return a new Loader for the given ID
- onLoadFinished() — Called when a previously created loader has finished its load
- onLoaderReset() — Called when a previously created loader is being reset making its data unavailable

THANK YOU