

CPSC475/575

Threads

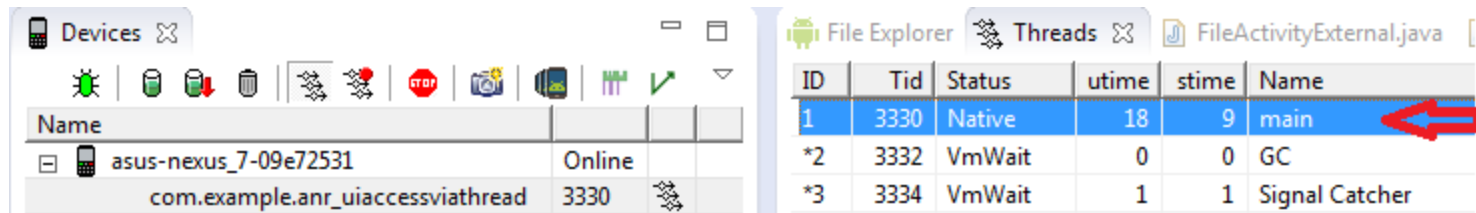
Today

- The 2 rules
- Updating UI with `AsyncTask`
- Handling Rotations
- **No Synchronization between Threads Yet**

The 2 Rules

- **DO NOT BLOCK THE UI THREAD**
 - Long-running code in main thread will make GUI controls nonresponsive and sometimes generate an ANR.
- **ONLY THE UI THREAD CAN ACCESS UI ELEMENTS**
 - Background threads are prohibited from updating UI.

what's the UI Thread? Its called main



The screenshot shows the Android Studio interface. On the left, the 'Devices' tab is active, displaying a list of virtual devices. The 'asus-nexus_7-09e72531' device is selected, showing its status as 'Online' and its package name 'com.example.anr_uiaccessviathread'. On the right, the 'Threads' tab is active, displaying a table of threads. The 'main' thread is highlighted in blue, and a red arrow points to it. The table has columns for ID, Tid, Status, utime, stime, and Name.

ID	Tid	Status	utime	stime	Name
1	3330	Native	18	9	main
*2	3332	VmWait	0	0	GC
*3	3334	VmWait	1	1	Signal Catcher

Nonresponsive GUI Controls

- **Solution**

- *Move time-consuming operations (network access, file access, database access, image manipulation or any long running task) to other threads*
- **Runnables**— most granular, hardest to get right, useful for small tasks requiring 1 thread
- **ExecutorService** – A framework to manage threadpools, lots of flexibility, much easier to get right
- **AsyncTask** – Android specific wrapper around runnable
 - Very useful for task that are run off the UI thread that need to interact with UI Thread elements
 - Methods for starting and stopping, UI updating and returning a result




Threads Cannot Update UI

- **Solutions (alternatives)**
 - Wait until all threads are done, then update UI
 - When multithreading improves performance, but total wait time is small - If 1 thread then use runnable, if many use ExecutorService (not addressed here)
 - Use AsyncTask to divide tasks between background and UI threads
 - Especially when developing for Android from beginning and you have a lot of incremental GUI updates.

AsyncTask

- **Scenario**

- Total wait time might be large, so you want to show intermediate results (progressbar) 
- You are designing code to divide the work between GUI and non-GUI code

- **Approach (4 steps)**

- onPreExecute
- **doInBackground**
- **onProgressUpdate**
 - publishProgress
- onPostExecute or onCancelled

AsyncTask: Quick Example

- **Task itself**

```
private class ImageDownloadTask extends AsyncTask<String, Void, View> {  
    public View doInBackground(String... urls) {  
        //return view  
    }  
  
    public void onPostExecute(View viewToAdd) {  
        //  
    }  
}
```

- **Invoking task**

```
String imageAddress = "http://...";  
ImageDownloadTask task = new ImageDownloadTask();  
task.execute(imageAddress);
```

AsyncTask Details: Constructor

- **Class is genericized with three arguments**
 - `AsyncTask<ParamType, ProgressType, ResultType>`
- **Interpretation**
 - **ParamType**
 - This is the type you pass to execute, which in turn is the type that is send to `doInBackground`. Both methods use varargs, so you can send any number of params.
 - **ProgressType**
 - This is the type that you pass to `publishProgress`, which in turn is passed to `onProgressUpdate` (which is called in UI thread). Use `Void` if you do not need to display intermediate progress.
 - **ResultType**
 - This is the type that you should return from `doInBackground`, which in turn is passed to `onPostExecute` (which is called in UI thread).

AsyncTask Details: doInBackground

- **Idea**

- This is the code that gets executed in the background. It **must not** update the UI.
- It takes as arguments whatever was passed to execute
- It returns a result that will be later passed to onPostExecute in the UI thread.

- **Code**

```
private class SomeTask extends AsyncTask<Type1, Void, Type2> {  
    public Type2 doInBackground(Type1... params) {  
        return(doNonUiStuffWith(params));  
    } ...  
}  
...  
new SomeTask().execute(type1VarA, type1VarB);
```

The ... in the doInBackground declaration is actually part of the Java syntax, indicating varargs.

AsyncTask Details: onPostExecute

- **Idea**

- This is the code that gets **executed on the UI thread**. It can update the UI.
- It takes as argument whatever was returned by doInBackground

- **Code**

```
private class SomeTask extends AsyncTask<Type1, Void, Type2> {  
    public Type2 doInBackground(Type1... params) {  
        return(doNonUiStuffWith(params));  
    }  
    public void onPostExecute(Type2 result) { doUiStuff(result); }  
}  
...  
new SomeTask(). execute(type1VarA, type1VarB);
```

AsyncTask Details: Other Methods

- **onPreExecute**
 - **Invoked by the UI thread** before doInBackground starts
- **publishProgress**
 - Sends an intermediate update value to onProgressUpdate. From background thread. You call this from code that is in doInBackground. The type is the middle value of the class declaration.
- **onProgressUpdate**
 - **Invoked by the UI thread.** Takes as input whatever was passed to publishProgress.
- **Note**
 - All of these methods can be omitted.

AsyncTask Details: Cancel()

- **Idea**
 - Call `myAsyncTask.cancel(true);`
 - Sets internal canceled flag
 - Periodically check `isCanceled()` in `doInBackground` Code
 - If canceled, `onCancelled()` is called verses `onPostExecute`

**What happens when the
phone rotates?**

AsyncTask: Configuration changes

- **Problem**

- Start an AsyncTask and then phone rotates
- Activity is destroyed and restarted
- AsyncTask however is still running
- What about all the references the AsyncTask has to original activity?
- Use `onRetainNonConfigurationInstance()` when activity being destroyed to save ref to thread
- Recapture thread in new Activities `onCreate()`..

AsyncTask: Configuration changes

- **Solution**

- In AsyncTask object, have get/set methods for setting its parent activity.

```
static class RotationAwareTask extends AsyncTask<Void, Void, Void> {  
    int progress = 0;  
    FixedAsyncTask activity = null;  
  
    //=====  
    //important do not hold a reference so garbage collector can grab old defunct dying activity  
    void detach() {  
        activity = null;  
    }  
  
    void attach(FixedAsyncTask activity) {  
        this.activity = activity;  
    }  
    //=====|
```

AsyncTask: Configuration changes

- **Solution**

- Then when phone rotates have parent Activity listen for `onRetainNonConfigurationInstance()`
- Detach Async task from soon to die parent (GC happy)

```
public class FixedAsyncTask extends Activity {  
    |  
    |  
    |  
    @Override  
    public Object onRetainNonConfigurationInstance() {  
        task.detach();  
        return (task);  
    }  
}
```


AsyncTask: Configuration changes

- **Solution**

- Then in onCreate recapture wayward thread

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.fixedasyncntaskview);

    bar = (ProgressBar) findViewById(R.id.progressBar1);
    task = (RotationAwareTask) getLastNonConfigurationInstance();

    //if a thread was retained then grab it
    if (task == null) {
        task = new RotationAwareTask(this);
        task.execute();
    } else {
        task.attach(this);
        updateProgress(task.getProgress());

        if (task.getProgress() >= 100) {
            markAsDone();
        }
    }
}
```

AsyncTask: Configuration changes

- Demo
6_Thread_AsyncTask_OnRetainNonConfigurationInstance

AsyncTask: One last bit

- **Standard implementation will execute 1 AsyncTask at a time (even if you try to run many at once)**

```
UpdateTask myTask = new UpdateTask();  
myTask.execute();
```

- **To do more than 1 at a time**

```
UpdateTask myTask = new UpdateTask();  
myTask.executeOnExecutor(AsyncTask.THREAD_POOL_EXECUTOR);
```

Summary

- **Update UI incrementally with AsyncTask**

- One update per task, but several updates per group of tasks

- How to setup - genericized with three arguments

- `AsyncTask<ParamType, ProgressType, ResultType>`

- How to Run - `myUpdateTask.execute()`

- What is run in separate thread `doInBackground` and `publishProgress`

- How to cancel - `myUpdateTask.cancel(true)`

- **Results** `onPostExecute` and `onCanceled`

- How to recover from orientation changes, keyboard slideouts etc

- How to communicate? Not yet just `publishProgress` and `onProgressUpdate`

Reading

- **JavaDoc**
 - AsyncTask
 - <http://developer.android.com/reference/android/os/AsyncTask.html>
- **Tutorial: Processes and Threads**
 - <http://www.javamex.com/>