Task 10.1P

not breaking the rules

1. Some rules are not to be broken

a) ‘Thread. Sleep’ snippet.

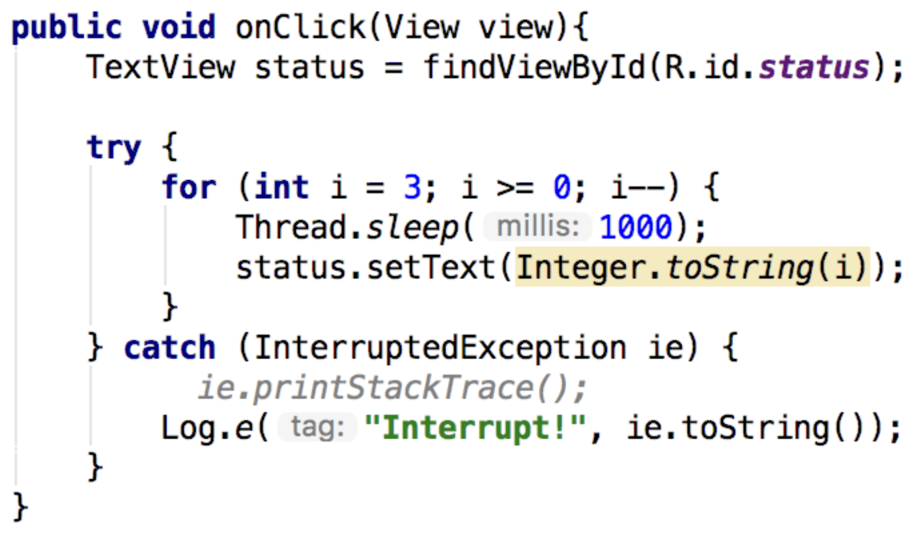
As far as I can tell, nothing interesting happens.
Technically the countdown works in that you wait the requisite amount of time, though the app just looks like it’s hanging – the shadow on the button remains ‘pressed’ for a while, until the text changes from the default ‘hello world’ to ‘0’.

That said, this shows up in Logcat:

> *5432-5432/com.example.a9726446.a10_1p I/Choreographer: Skipped 239 frames! The application may be doing too much work on its main thread.*

Essentially, it’s breaking one of the rules of Android’s single thread model – it blocks the main thread.
(https://developer.android.com/guide/components/processes-and-threads,last accessed 1/11/18)

This isn’t advised as it prevents all other events from being dispatched, including draw events, giving the hanging effect described above.



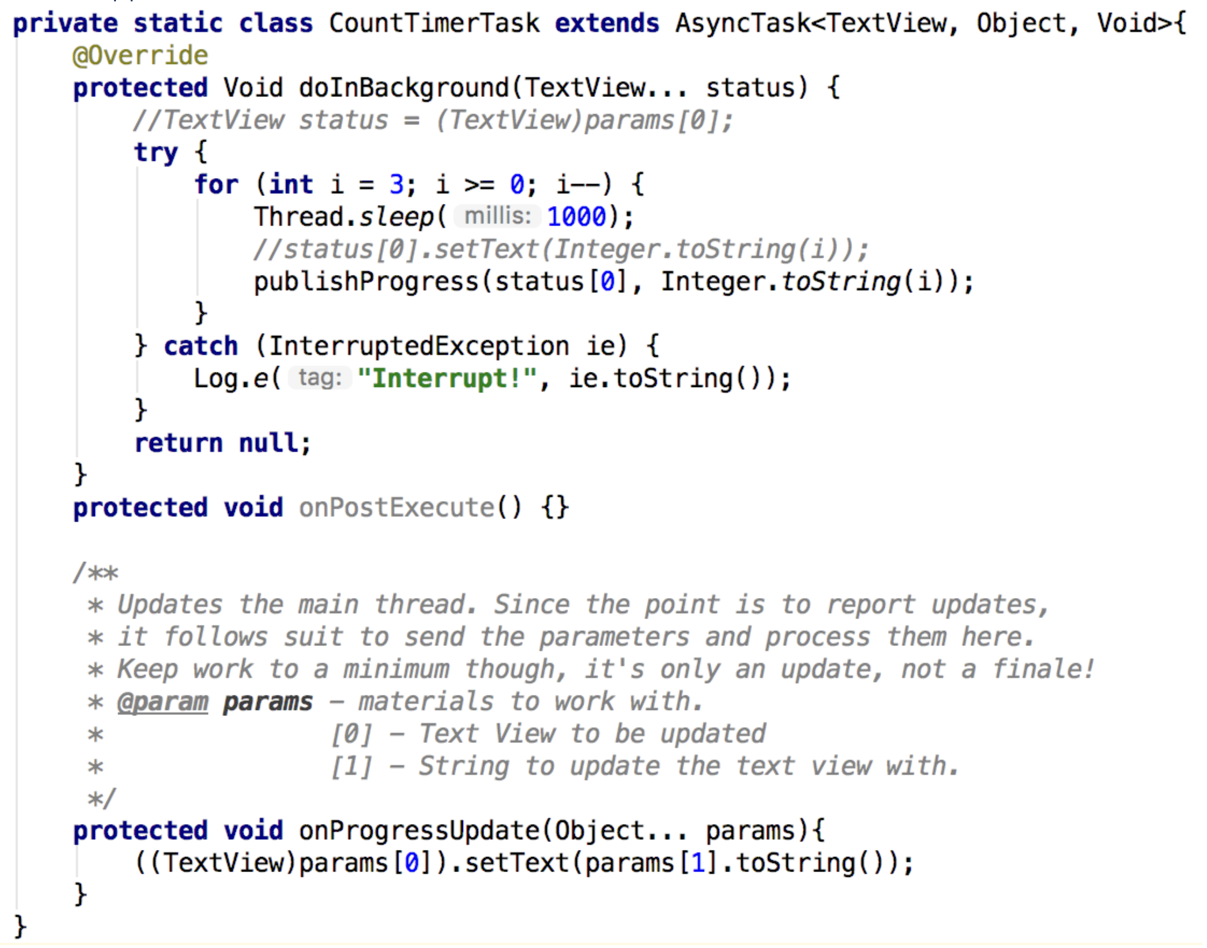
```
public void onClick(View view){
    TextView status = findViewById(R.id.status);

    try {
        for (int i = 3; i >= 0; i--) {
            Thread.sleep( millis: 1000);
            status.setText(Integer.toString(i));
        }
    } catch (InterruptedException ie) {
          ie.printStackTrace();
        Log.e( tag: "Interrupt!", ie.toString());
    }
}
```

b) Refactoring with Async Task

//With references from https://android-developers.googleblog.com/2009/05/painless-threading.html (last accessed: 1/11/18)
And the lecture slides from Module 6 (Concurrent Tasks)

Code Snippet:



```
private static class CountTimerTask extends AsyncTask<TextView, Object, Void>{
    @Override
    protected Void doInBackground(TextView... status) {
        //TextView status = (TextView)params[0];
        try {
            for (int i = 3; i >= 0; i--) {
                Thread.sleep( millis: 1000);
                //status[0].setText(Integer.toString(i));
                publishProgress(status[0], Integer.toString(i));
            }
        } catch (InterruptedException ie) {
            Log.e( tag: "Interrupt!", ie.toString());
        }
        return null;
    }
    protected void onPostExecute() {}

    /**
     * Updates the main thread. Since the point is to report updates,
     * it follows suit to send the parameters and process them here.
     * Keep work to a minimum though, it's only an update, not a finale!
     * @param params - materials to work with.
     *               [0] - Text View to be updated
     *               [1] - String to update the text view with.
     */
    protected void onProgressUpdate(Object... params){
        ((TextView)params[0]).setText(params[1].toString());
    }
}
```

Refer Appendix for screenshots of a timer ticking down

Appendix: Screenshots of a timer ticking down:

//Not that these illustrate that much haha…